Universidade do Minho
Escola de Engenharia

João Miguel Pereira Lopes Guedes

Integração de Sistemas ABI em Saúde

Integração de Sistemas ABI em Saúde
João Miguel Pereira Lopes Guedes
UMinho | 2024

Outubro de 2024

Universidade do Minho
Escola de Engenharia

João Miguel Pereira Lopes Guedes

Integração de Sistemas ABI em Saúde

Relatório Intermédio Trabalho de Dissertação de Mestrado
Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação de
Professor Manuel Filipe Vieira Torres dos Santos
e coorientação de
Doutor Júlio Miguel Marques Duarte

Outubro de 2024

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio, nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

RESUMO

Integração de Sistemas ABI em Saúde

A integração de sistemas para *Adaptive Business Intelligence*(ABI) na indústria da saúde têm o potencial de revolucionar a forma como as organizações abordam a análise de dados e a tomada de decisão. Ao fornecer conhecimentos em tempo real e passíveis de ação e permitir que as organizações se adaptem e evoluam continuamente, o conceito de ABI tem o potencial de conduzir a melhores resultados, reduzir custos e melhorar a qualidade geral dos cuidados de saúde dos pacientes.

O presente trabalho de dissertação explora a integração de sistemas ABI na indústria dos cuidados de saúde, investigando os benefícios, desafios e oportunidades de implementação. O estudo foi conduzido através de uma combinação de revisão de literatura, estado da arte e casos de estudo de tentativas de integração dos diversos componentes de sistemas ABI nas operações de diversas organizações. Após os casos de estudo terem sido reunidos e revisão de literatura completa foi realizado um plano e desenvolvimento de uma solução arquitetónica de integração de um sistema ABI que torne possível a interação com ambientes externos por meio de recursos de interoperabilidade, bem como um protótipo prático de forma a testar esta solução. Durante o desenvolvimento da solução, o foco de investigação estendeu-se, também, para as melhores práticas e metodologias no desenvolvimento moderno, eficiente e seguro de software.

Palavras-chave: Adaptive Business Intelligence; Previsão; Otimização; Decisão; Data Mining; Saúde; Interoperação.

ABSTRACT

ABI Systems Integration in Healthcare

The integration of adaptive business intelligence (ABI) systems into the healthcare industry has the potential to revolutionize the way organizations approach data analysis and decision making. By providing real-time, actionable insights and enabling organizations to continuously adapt and evolve, ABI has the potential to drive better outcomes, reduce costs, and improve the overall quality of patient care.

This dissertation work explores the integration of ABI systems in the healthcare industry, investigating the benefits, challenges, and opportunities of implementation. The study was conducted through a combination of literature review and case studies of healthcare organizations that have successfully integrated ABI systems into their operations. After the case studies were gathered and literature review completed a plan and development of an ABI implementation solution architecture was undertaken that could interact with external environments through interoperability resources, as well as a practical prototype to test said architecture. During the development of the solution, the research focus also extended to best practices and methodologies in modern, efficient and secure software development.

Keywords: Adaptive Business Intelligence; Prediction; Optimization; Decision; Data Mining; Healthcare; Interoperation.

ÍNDICE

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros ii

Declaração de Integridade iii

Resumo iv

Abstract v

Lista de Abreviaturas e Siglas xiii

Lista de Figuras xv

Lista de Tabelas xviii

1. Introdução 19

1.1 Enquadramento e Motivação 19

1.2 Objetivos e Resultados Esperados 20

2. Metodologias e Tecnologias 21

2.1 Cross Industry Standard Process for Data Mining (CRISP-DM) 21

2.2 Design Science Research (DSR) 23

2.3 Método de Revisão Bibliográfica 25

2.4 SCRUM Solo 26

2.5 Ferramentas para o desenvolvimento 29

3. Estado da Arte 30

3.1 Business Intelligence 30

3.2 Sistemas ABI 31

3.2.1 Data Mining 32

3.2.2 Previsão 33

3.2.3 Otimização 34

3.2.4 Adaptabilidade 36

3.3 Integração de Sistemas 36

3.4 Integração de Sistemas ABI (Casos de Estudo) 38
3.4.1 Previsão (Casos de Estudo) 39
3.4.2 Otimização (Casos de Estudo) 40
3.4.3 Adaptabilidade e Arquitetura de Sistemas ABI (Casos de Estudo) 42
3.4.4 Conclusões Acerca dos Casos de Uso 43
3.5 Bases de Dados 43
3.5.1 Normalização de modelos de dados 44
3.5.2 Soluções ORM 45
3.6 Padrões de Arquitetura de microserviços 46
3.6.1 Padrões de Orquestração e Coordenação 47
3.6.2 Padrões de Implementação 47
3.6.3 Padrões de armazenamento de dados 48
3.7 Padrões de Design de Software 48
3.7.1 Padrão de Pipeline 49
3.7.2 Padrão de Estratégia 49
3.7.3 Padrão MVC 50
3.8 Encriptação de Dados 50
3.9 JSON Web Tokens 52
3.10 Expressões Regulares 53
3.11 React.js 54
3.12 Servidores HTTP 57
3.12.1 Nginx 58
3.13 Controlo de Versão 59
3.13.1 Git 60
3.14 Estratégias de Branching 62

3.15 SemVer 63
3.16 Continuous Integration/Continuous Deployment 64
3.17 Conventional Commits 65
3.18 Virtualização de Software 65
3.18.1 Docker 66
3.18.2 Docker Compose 67
3.19 Testes Unitários 68
3.20 Testes E2E 70
3.21 Segurança Web 70
3.21.1 OWASP Top 10 71
3.21.2 Broken Access Control 72
3.21.3 Cryptographic Failures 73
3.21.4 Injection 73
3.21.5 Insecure Design 74
3.21.6 Security Misconfiguration 74
3.21.7 Vulnerable and Outdated Components 74
3.21.8 Identification and Authentication Failures 75
3.21.9 Software and Data Integrity Failures 75
3.21.10 Security Logging and Monitoring Failures 75
3.21.11 Server Side Request Forgery 76
4. Interoperabilidade 76
4.1 Padrão HL7 77
4.1.1 Padrão HL7 V2 78
4.1.2 Padrão HL7 FHIR 80
5. Proposta E Conceção do Sistema 82

5.1 Definição do Âmbito 82
5.2 Funcionamento do Sistema 83
5.2.1 Funcionamento Base dos Operadores Arbitrários do Sistema 86
5.2.2 Extração Arbitrária de Dados de recursos HL7 87
5.2.3 Extração Arbitrária de Dados de recursos FHIR 89
5.2.4 Validação Arbitrária de Dados de Entrada 91
5.2.5 Preprocessamento Arbitrário de Dados de Entrada 92
5.3 Arquitetura Concetual do Sistema 96
5.4 Requisitos do Sistema 97
5.4.1 Requisitos Funcionais 98
5.4.2 Requisitos Não Funcionais 99
6. Desenvolvimento do Sistema 100
6.1 Base de Dados 101
6.2 Arquitetura Tecnológica 102
6.3 Padrões de Arquitetura de Microserviços 105
6.4 Modelo/Template De Microserviço 105
6.4.1 Node.js 106
6.4.2 Padrão MVC 107
6.4.3 Estrutura do Modelo/Template 107
6.5 Microserviço de Interoperabilidade 108
6.5.1 Interface com Base de Dados 111
6.5.2 Encriptação de Dados 111
6.5.3 Pré-População da Base de Dados 112
6.5.4 Autenticação 113
6.5.5 Mecanismos de execução de configurações 113

6.5.6 Medidas de Segurança 116
6.6 Microserviço Mirth Connect 118
6.6.1 Mecanismo de Interface com a REST API Mirth 119
6.6.2 Medidas de Segurança 120
6.7 Microserviço de Gestão de Mensagens 120
6.7.1 Mecanismo de Interface com Gestores de Mensagens 121
6.7.2 Medidas de Segurança 122
6.8 Interface Gráfica 123
6.8.1 Antd Design 124
6.8.2 Layout da Interface Gráfica 125
6.8.3 Autenticação 126
6.8.4 Conteúdos/ *Views* Desenvolvidos 128
6.9 Controlo de Versões 130
6.9.1 Estratégia de Gestão de *Branches* 130
6.9.2 Controlo de versões semântico 132
6.10 Hospedagem de repositórios 133
6.11 Continuous Integration/Continuous Deployment 134
6.11.1 Github Actions 135
6.11.2 Estrutura Concetual de CI/CD do sistema 135
6.11.3 Estruturação de *commits* 137
6.11.4 Virtualização de Software e Docker 138
6.11.5 Docker Compose 142
6.11.6 Automatização de Testes Unitários 143
6.11.7 Registo de *Containers* 144
6.11.8 Ambiente de Desenvolvimento e Testes 145

7. Testes 147

7.1 Testes *e2e* 148

7.2 Testes de Arquitetura 150

7.3 Teste de modelos 151

7.3.1 Preditor de hospitalização 152

7.3.2 Predição de Diabetes 155

8. Conclusões e Reflexões 158

9. Trabalho Futuro 161

10. Bibliografia 162

Lista de Abreviaturas e Siglas

ABI	Adaptative Business Intelligence / Sistema de Inteligência Empresarial Adaptável
BI	Business Intelligence / Inteligência de Negócio
DSR	Design Science Research / Design de Pesquisa Científica
CRISP-DM	Cross-Industry Standard Process for Data Mining / Processo Padrão Inter-Indústrias para Data Mining
HL7	Health Level Seven
FHIR	Fast Healthcare Interoperability Resources
NFV	Network Function Virtualization / Virtualização da Função de Rede
DM	Data Mining
ML	Machine Learning
DL	Deep Learning
UCS	Unidade de Cuidados de Saúde
API	Aplication Programming Interface / Interface de programação de aplicações
JSON	JavaScript Object Notation / Notação de Objetos de JavaScript
XML	Extensible Markup Language / Linguagem extensível de Marcação
HTTP	Hypertext Transfer Protocol / Protocolo de transferência de hipertexto
REST	Representational State Transfer / Transferência de Estado Representacional
ID/id	Elemento identificador
Regex	Expressão Regular / Regular Expression
OWASP	Projeto Mundial Aberto de Segurança das Aplicações / Open Worldwide Application Security Project
SQL	Structured Query Language / Linguagem de consulta estruturada
ACID	Atomic, Consistent, Isolated and Durable / Atómicas, Consistentes, Isoladas e Duradouras
ERD/DER	Entity-Relationship Model / Diagrama de Entidades e Relacionamentos
ORM	Object-Relational Mapping / Mapeamento Objeto-Relacional
CPU	Central Processing Unit
GPU	Graphics Processing Unit
NPM	Node Package Manager / Gestor de Pacotes Node
HA	High Availability / Alta Disponibilidade

BFF	Backend for Frontend / Backend para Frontend
MVC	Model-View-Controller / Modelo-Visão-Controlador
JWT	JSON Web Token
CORS	Cross Origin Resource Sharing / Partilha de Recursos entre Origens
DOM	Document Object Model / Modelo de Objeto de Documento
URL	Uniform Resource Locator / Localizador Uniforme de Recursos
RSA	Rivest-Shamir-Adleman
HMAC	Hash Based Message Authentication Code / Código de Autenticação de mensagens baseadas em hash
PRF	Pseudo-random Function / Função Pseudoaleatória
VCS	Version Control System / Sistema de Controlo de Versão
IaC	Infrastructure as Code
IDE	Integrated Development Environment / Ambiente de Desenvolvimento Integrado
CI	Continuous Integration / Integração Contínua
CD	Continuous Deployment / Lançamento Contínuo
CD	Continuous Delivery / Entrega Contínua
UT	Unit Test / Teste Unitário
GHCR	GIthub Container Registry / Registo de *Containers* Github
DSI	Departamento de Sistemas de Informação
SIL	Software in the Loop / *Software* no ciclo
E2E	End-To-End / Ponta-a-Ponta
OWASP	Open Web Application Security Project / Projeto Aberto de Segurança de Aplicações Web
BAC	Broken Access Control / Controlo de Acesso Comprometido
RBAC	Role Based Access Control / Controlo de Acesso Baseado em Função
SSRF	Server-Side Request Forgery / Falsificação de pedidos do lado do servidor

LISTA DE FIGURAS

Figura 1 - Cross Industry Standard Process for Data Mining (CRISP-DM) (adaptado de (Azevedo & Santos, 2008)) 23
Figura 2 - Processo de Design Science Research (DSR) (adaptado de (Vaishnavi et al., 2004)) 24
Figura 3 - Linha cronológica SCRUM do desenvolvimento 28
Figura 4 - Sistema Adaptive Business Intelligence (adaptado de (Michalewicz et al., 2007)) 32
Figura 5 - Componente preditiva de ABI (adaptado de (Michalewicz et al., 2007)) 34
Figura 6 - Componente de Otimização de ABI (adaptado de (Michalewicz et al., 2007)) 36
Figura 7 - Componente de Adaptabilidade de ABI (adaptado de (Michalewicz et al., 2007)) 36
Figura 8 - Arquitetura Service Oriented Architecture (SOA) 37
Figura 9 - Arquitetura de Microserviços 38
Figura 10 - Arquitetura Network Function Virtualization (NFV) 38
Figura 11 - Diagrama ORM 46
Figura 12 - Exemplo de uma pipeline de um browser (adaptado de (Vermeulen et al., 1995)) 49
Figura 13 - Exemplo do padrão de Estratégia (adaptado de (Gamma et al., 1994)) 49
Figura 14 - Fluxo de autenticação por JWT 52
Figura 15 - Estrutura JSON Web Token 53
Figura 16 - Servidores HTTP listados por popularidade (adaptado de (Kithulwatta et al., 2022)) 57
Figura 17 - Teste de Performance de ficheiros de 1KB em servidores diferentes (adaptado de (Guolang et al., 2018)) 58
Figura 18 - Estrutura de um *commit* padronizado (adaptado de (Huskey & Huskey, 2023)) 65
Figura 19 - Processo de acondicionamento de software através do Docker (adaptado de (Hasan Ibrahim et al., 2021)) 67
Figura 20 - OWASP Top 10 (adaptado de (Veres & Wilkinson, 2023)) 72
Figura 21 - Controlo de Acesso Baseado em *Roles* 73
Figura 22 - Exemplo Mensagem HL7 79
Figura 23 - Publicações de estudos sobre o FHIR ao longo do tempo (adaptado de (Vorisek et al., 2022)) 81
Figura 24 - Exemplo Recurso FHIR JSON/XML 82
Figura 25 - Funcionamento Base do Sistema de Interoperabilidade 84

Figura 26 - Conceito de um mapeamento HL7 88
Figura 27 - Conceito de um mapeamento FHIR 89
Figura 28 - Conceito da extração de dados FHIR 91
Figura 29 - Conceito do mecanismo de validação arbitrária 91
Figura 30 - Exemplo de um validador 92
Figura 31 - Conceito de Mecanismo de Preprocessamento 93
Figura 32 - Conceito Implementação segura de execução dinâmica de código 95
Figura 33 - Exemplo de um Preprocessador 95
Figura 34 - Arquitetura Concetual do Sistema de Interoperabilidade 96
Figura 35 - Arquitetura Tecnológica do sistema de interoperabilidade 102
Figura 36 - Estrutura de Ficheiros de Template de Microserviços 108
Figura 37 - Arquitetura Microserviço de Interoperabilidade 109
Figura 38 - Algoritmo de encriptação Bcrypt 112
Figura 39 - Mecanismo de Execução de Configurações 114
Figura 40 - Arquitetura Microserviço Mirth Connect 118
Figura 41 - Arquitetura Microserviço de Gestão de Mensagens 120
Figura 42 - Mecanismo de comunicação para gestão de mensagens 122
Figura 43 - Arquitetura Interface Gráfica 123
Figura 44 - Layout Interface Gráfica 125
Figura 45 - Router Interface Gráfica 126
Figura 46 - Mecanismo de Autenticação e Encaminhamento Interface Gráfica 127
Figura 47 – Estratégia de *Branching* 131
Figura 48 – Versionamento Semântico de um Microserviço 132
Figura 49 – Organização ABI-Interoperability-Thesis no Github 134
Figura 50 – Estratégia de Automação CI/CD 136
Figura 51 – Exemplos de *commits* padronizados 138
Figura 52 – Fluxo de acondicionamento de microserviços do sistema de interoperabilidade 139
Figura 53 – Fluxo de acondicionamento da aplicação Web do sistema de interoperabilidade 140
Figura 54 – Fluxos de consumo das partes do sistema de interoperabilidade 141
Figura 55 – Execução do sistema através de Docker Compose 142
Figura 56 – Exemplo de integração de testes automáticos no CI/CD 144
Figura 57 – Imagem do microserviço de interoperabilidade no GHCR 145

Figura 58 – Integração do sistema em ambiente de testes 146
Figura 59 – Caso de observação HL7 154

LISTA DE TABELAS

Tabela 1 – Lista de ferramentas 29
Tabela 2 - Tabela de Diferenças entre bases de dados relacionais e não relacionais 44
Tabela 3 - Lista de Requisitos Funcionais do Sistema de Interoperabilidade 98
Tabela 4 - Lista de Requisitos Não-Funcionais do Sistema de Interoperabilidade 100
Tabela 5 - Distribuição dos componentes do sistema pelo padrão MVC 107
Tabela 6 - Componentes de interface gráfica 128
Tabela 7 - Predefinição da configuração Docker Compose 142
Tabela 8 - Constituição Docker Compose no Ambiente de Testes 147
Tabela 9 - Plano de testes E2E 149
Tabela 10 - Plano de testes de arquitetura 150
Tabela 11 - Resultados dos testes de arquitetura 150
Tabela 12 - Mapeamentos HL7 dos atributos do modelo predição de hospitalização 152
Tabela 13 - Mapeamentos FHIR dos atributos do modelo predição de hospitalização 154
Tabela 14 – Resultados dos testes para o modelo de predição de hospitalização 154
Tabela 15 - Atributos e distribuições HL7 para predição de diabetes 155
Tabela 16 - Atributos e distribuições FHIR para predição de diabetes 156
Tabela 17 - Resultados dos testes para o modelo de predição de diabetes 157

1. INTRODUÇÃO

1.1 Enquadramento e Motivação

A integração de sistemas de Adaptive Business Intelligence na Saúde é algo amplamente discutido, não só devido à capacidade de suportar os dados que são gerados diariamente, mas também devido à expectativa de inserir a evolução de um sistema inteligente na melhoria dos cuidados prestados à sociedade, apoiando paralelamente os administradores hospitalares na gestão das decisões estratégicas inerentes ao funcionamento de uma organização de Saúde. (Ashfaq & Nowaczyk, 2019; Lopes et al., 2021)

Para encurtar os caminhos da transparência e da interoperabilidade destas soluções, a comunidade científica está a realizar vários estudos para assegurar arquiteturas tecnológicas suficientemente capazes de satisfazer os requisitos(Aceto et al., 2020; Tian et al., 2019), com implementações generalizadas a certas especialidades médicas e entidades hospitalares, procurando meios para assegurar a elevada precisão dos sistemas baseados em IA (F. Wang et al., 2019). Um dos aspetos críticos na adoção desta tecnologia prende-se com a interoperação com outros sistemas. A capacidade dos sistemas ABI de interoperar com outros sistemas, num ecossistema em saúde, permite que se estabeleça uma comunicação bidirecional através do qual sejam enviados pedidos (e.g. previsão, otimização) e resultados.

Este projeto de dissertação pretende abordar e explorar o potencial que a interoperabilidade possui em relação à implementação de sistemas ABI no setor da saúde bem como o desenvolvimento de uma arquitetura capaz de promover esta atividade. Neste projeto houve duas grandes etapas de pesquisa, a de interoperabilidade e o seu estado atual de implementação no setor da saúde, e o estado atual do desenvolvimento moderno de *software*. Todos os aspetos pesquisados encontram-se descritos neste documento e serviram como base para toda a conceção e desenvolvimento da solução apresentada.

No primeiro capítulo é apresentada a motivação por detrás deste trabalho de investigação bem como são descritos os objetivos esperados. No capítulo seguinte são descritas todas as metodologias e ferramentas utilizadas para a realização do trabalho. É destacado o uso do DSR (*Design Science Research*) como paradigma de investigação, o método de revisão bibliográfica e SCRUM solo como metodologia de desenvolvimento de *software*.

O terceiro capítulo corresponde às conclusões retiradas e documentação da revisão da literatura reunida. Este tópico cobre o estado atual de todos os conceitos explorados ao longo do trabalho desde os conceitos de ABI e as suas implementações no setor da saúde até às vulnerabilidades de *software* analisadas. Desta forma este tópico serve como fundamento para os restantes tópicos que compõe todo o trabalho de investigação e desenvolvimento.

O tópico de interoperabilidade explora e descreve o próprio conceito de interoperabilidade bem como os padrões e recursos estudados ao longo do trabalho.

O tópico seguinte de proposta e conceção da solução descreve a proposta da solução de uma forma intangível focando-se apenas nos conceitos e fluxos de alto nível. Aqui o sistema é descrito de forma concetual começando pelo âmbito e pressupostos e passando para a arquitetura em si com foco nas suas componentes individuais. Neste tópico de conceção foram definidas as características mais importantes da solução como a arquitetura de microserviços, o fluxo tecnológico, os padrões de desenvolvimento de *software* e os requisitos do sistema.

O sexto tópico é o mais extenso e descreve o processo do desenvolvimento da solução proposta. É neste tópico que são abordadas as características e assuntos mais técnicos da solução. Aqui foram descritos os modelos de dados, a arquitetura tecnológica, os padrões de desenvolvimento dos microserviços bem como o desenvolvimento detalhado de cada um. Também foi descrito os mecanismos de funcionamento e características de cada microserviço incluindo protocolos e comunicação e medidas de segurança.

É também explorado o desenvolvimento da interface gráfica que acompanha o sistema, o mecanismo do controlo de versões de todas as bases de código mantidas no projeto, sistema de controlo de contribuição e iteração e a automação de testes, virtualização e implementação via integração contínua (CI/CD).

O sétimo tópico corresponde aos testes feitos à solução. Estes testes pretendem transmitir a eficácia do sistema desenvolvido de forma quantitativa. Foram realizados testes end-to-end de forma a testar a solução completa, testes de arquitetura com diferentes configurações e testes com modelos reais.

O trabalho conclui com os tópicos de conclusão e reflexão onde são compiladas e discutidas todas as conclusões que foram extraídas ao longo do projeto e tenta colocar este projeto como uma base para os futuros estudos e desenvolvimentos em relação ao uso de interoperabilidade em sistemas ABI.

Foi também escrito um tópico extra para trabalho futuro onde estão compiladas todas as melhorias, iterações e temas de pesquisas futuras que irão acrescentar valor à solução atual.

1.2 Objetivos e Resultados Esperados

Será possível, através de uma arquitetura, promover a formulação de sistemas ABI no contexto dos sistemas de saúde?

O principal objetivo deste trabalho será responder a esta questão. Para atingir esse fim foram desenvolvidos alguns objetivos a partir de um conjunto de casos de uso da área da saúde:

- Investigar sobre formas de atingir a interoperação entre sistemas por forma a receber pedidos e enviar resultados de modelos de Data Mining e/ou Otimização recorrendo a standards da área;
- Desenvolver técnicas de integração de diferentes protocolos de comunicação na saúde;
- Levantamento de requisitos para o sistema com base na pesquisa de interoperabilidade e conclusões retiradas;
- Investigar acerca das melhores e mais eficientes práticas de desenvolvimento de sistemas informáticos e Software bem como a seleção das tecnologias e uma arquitetura capaz de satisfazer todos os requisitos levantados;
- No âmbito da fase 6 da metodologia CRISP-DM, definir uma arquitetura de interoperação que permita a partir de outros sistemas solicitar a execução de modelos e o envio de resultados (e.g. previsões);
- Criar um protótipo recorrendo a tecnologias de programação que permita implementar e testar a arquitetura definida;
- Demonstrar a eficiência da abordagem.

Todos estes objetivos foram tidos em consideração em todas as fases de desenvolvimento do presente trabalho sendo que foram reavaliados a cada passo e decisão. Desta forma foi possível garantir o cumprimento de todos eles. O seu cumprimento é esclarecido e discutido nas conclusões do documento.

2. METODOLOGIAS E TECNOLOGIAS

Neste capítulo serão abordadas as metodologias a utilizar no contexto do projeto para a utilização dos modelos de Data Mining e para conduzir todo o processo de investigação e produção do *deliverable* final. Também será abordado o método e processo da revisão bibliográfica, que detalha a forma como foi reunida toda a literatura e casos de estudo, bem como a análise exploratória da mesma de forma a garantir a sua relevância tanto temporal como temática e atualidade.

2.1 Cross Industry Standard Process for Data Mining (CRISP-DM)

O CRISP-DM (Cross Industry Standard Process for Data Mining) é uma metodologia amplamente reconhecida e altamente considerada para o desenvolvimento de projetos de Data Mining e análise preditiva. Esta abordagem sistemática fornece um quadro estruturado para a identificação e resolução de problemas empresariais através da utilização da análise e modelação dos dados.

No seu núcleo, o CRISP-DM foi concebido para ajudar as organizações a tomar decisões orientadas para os dados e a melhorar o seu desempenho global. A metodologia é altamente flexível e pode ser aplicada a uma vasta gama de problemas empresariais, desde a segmentação de clientes até à manutenção preditiva.

O processo CRISP-DM é composto por seis etapas-chave, sendo que todas são fundamentais para o sucesso do projeto global (Figura 1):

Business Understanding (Compreensão do Negócio): Constitui a 1ª fase do CRISP-DM, e é crucial para estabelecer as bases para o resto do projeto. O objetivo desta fase é compreender os requisitos do projeto de uma perspetiva empresarial. Neste desenrolar, os objetivos do projeto são claramente definidos e é desenvolvido um plano preliminar para os atingir. Por fim o conhecimento empresarial dos requisitos será convertido numa definição do problema para Data Mining(Azevedo & Santos, 2008).

Data Understanding (Compreensão dos dados): Nesta fase, o foco desloca-se para os dados em si. A recolha inicial de dados é realizada, e os dados são analisados para identificar quaisquer problemas de qualidade de dados, assim como para obter uma visão dos mesmos. Esta fase é fundamental para assegurar que os dados utilizados no projeto são relevantes e precisos. Além disso, podem ser detetados subconjuntos interessantes dos dados, e podem ser formadas teorias de forma descobrir/gerar informações ocultas(Azevedo & Santos, 2008).

Data Preparation (Preparação dos dados): Esta fase abrange todas as atividades envolvidas na construção do conjunto de dados final a partir dos dados em bruto originais. A fase de preparação dos dados é, frequentemente, a fase mais demorada do quadro CRISP-DM, mas é também a fase que pode ter o maior impacto no sucesso do projeto uma vez que um modelo é tão melhor quanto melhores forem os dados utilizados para o seu desenvolvimento e treino. O objetivo desta fase é limpar, transformar e organizar os dados de uma forma que seja adequada para a, posterior, análise(Azevedo & Santos, 2008).

Modeling (Modelação): Nesta fase, várias técnicas de modelação são selecionadas e aplicadas aos dados. Os parâmetros dos modelos são calibrados para valores ótimos, e os resultados são analisados para determinar qual o modelo mais adequado, bem como hiperparâmetros e cenários, para o problema em questão. Esta fase é onde se realiza a extração de dados propriamente dita, e é fundamental para o sucesso do projeto(Azevedo & Santos, 2008).

Evaluation (Avaliação): Nesta fase, o(s) modelo(s) desenvolvido(s) na fase anterior é(são) cuidadosamente avaliado(s). O processo de avaliação é utilizado para determinar se o modelo cumpre os objetivos empresariais e para identificar quaisquer áreas que necessitem de melhoria. Além disso, as etapas executadas para construir o modelo são revistas para garantir que foram executadas corretamente. Esta fase é fundamental para garantir a qualidade e precisão dos resultados(Azevedo & Santos, 2008).

Deployment (Implementação): A fase final da estrutura CRISP-DM é a integração ou implementação do(s) modelo(s) resultante(s). Esta fase envolve a organização e apresentação do conhecimento adquirido no projeto de uma forma que seja útil para o cliente. Esta fase é fundamental para assegurar que os resultados do projeto sejam passíveis de ação e possam ser utilizados para tomar decisões informadas. A fase de implementação marca o fim do projeto de Data Mining, mas os resultados obtidos com o projeto podem ser utilizados para informar futuros projetos e trabalho e para melhorar os processos empresariais em geral(Azevedo & Santos, 2008).

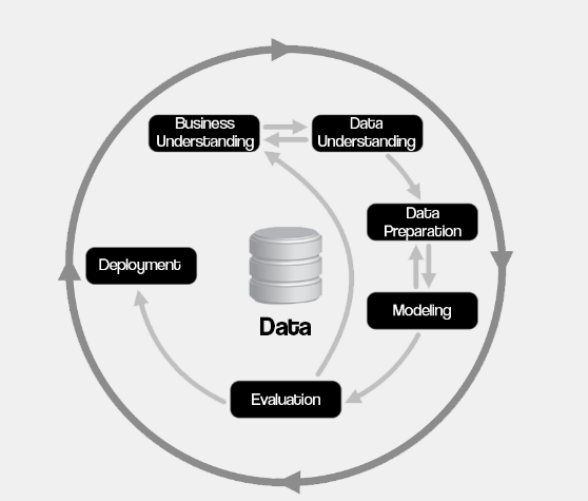

Figura 1 - Cross Industry Standard Process for Data Mining (CRISP-DM) (adaptado de (Azevedo & Santos, 2008))

Ao longo deste projeto apenas será abordada a sexta fase deste processo, o **Deployment**, ou integração visto que o escopo do mesmo apenas trata da integração dos modelos numa arquitetura de interoperação com sistemas ABI capaz de os utilizar eficazmente assumindo que estes já foram desenvolvidos.

2.2 Design Science Research (DSR)

O Design Science Research é uma abordagem de investigação que procura desenvolver o conhecimento através da conceção e criação de artefactos, como por exemplo software, modelos de processo, estudos e resultados, com o objetivo de resolver problemas específicos ou abordar necessidades

específicas(Vaishnavi et al., 2004). O objetivo do DSR é criar soluções para problemas práticos e avaliar a sua eficácia em vez da simples descrição e explicação de fenómenos. Por esta razão, esta abordagem é muito utilizada em campos mais práticos como os sistemas de informação, informática, engenharia e gestão. Os artefactos produzidos aquando da utilização do DSR devem ser de alta qualidade, inovadores e capazes de abordar problemas práticos de uma forma significativa, cuja avaliação é fundamental de forma a assegurar a sua eficácia, fiabilidade e utilidade (Vaishnavi et al., 2004).

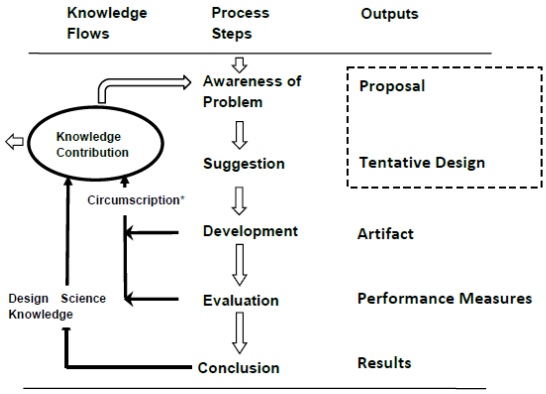

Figura 2 - Processo de Design Science Research (DSR) (adaptado de (Vaishnavi et al., 2004))

Por norma, existem 5 fases para o DSR (Figura 2):

Awareness of Problem/Sensibilização para o Problema – As fontes de um problema de investigação interessante podem ser variadas, tais como novos avanços na indústria, identificação de questões dentro de um campo relevante, ou leitura numa disciplina relacionada. O resultado desta fase é uma proposta para um novo projeto de investigação que pode ser formal ou informal (Vaishnavi et al., 2004). Esta fase deu origem à questão de investigação e, por conseguinte, ao presente documento dedicado à sua exploração.

Suggestion/Sugestão – A fase de Sugestão no processo de DSR sucede a fase da proposta inicial e está ligada à proposta desenvolvida com base na Sensibilização para o Problema. Envolve uma etapa criativa em que se prevê uma nova funcionalidade baseada numa combinação única de elementos existentes ou novos e existentes. (Vaishnavi et al., 2004). Esta fase foi vital para a conceção e desenho da solução apresentada neste trabalho. Aqui foram exploradas múltiplas soluções existentes, tecnologias e abordagens que pudessem contribuir para o desenvolvimento da solução.

Development/Desenvolvimento – O design provisório já se encontra avançado e a sua implementação varia em função do artefacto a ser criado. Porém o foco é a conceção e não a construção efetiva do artefacto, cuja implementação em si pode ser muito padronizada (Vaishnavi et al., 2004). O desenvolvimento tratou-se da fase mais longa do processo sendo que englobou o desenvolvimento da solução bem como a produção da respetiva documentação.

Evaluation/Avaliação – Na fase de avaliação do DSR, o artefacto construído é avaliado com base nos critérios declarados na proposta inicial. Se o artefacto se desviar das expectativas, a fase de avaliação irá conter uma subfase analítica onde são feitas hipóteses e teorias para explicar o seu comportamento fora dos parâmetros esperados. Esta fase conduz frequentemente a outra ronda de sugestões, uma vez que os resultados da avaliação são alimentados no processo de investigação (Vaishnavi et al., 2004). Esta fase correspondeu à avaliação crítica dos resultados e à sua justaposição em relação aos objetivos estabelecidos. As expetativas foram cumpridas de uma forma geral pelo que não houve necessidade de elaborar análises adicionais.

Conclusion/Conlusão – A fase de conclusão é a fase final do esforço de investigação e marca a conclusão do ciclo de investigação. Esta fase caracteriza-se por consolidar os resultados da investigação e por comunicar os conhecimentos adquiridos de forma concisa e clara. Os resultados são frequentemente categorizados em conhecimento "firme" que pode ser aplicado repetidamente e "pontas soltas" que podem justificar mais trabalho de investigação para o futuro. A comunicação é importante nesta fase, uma vez que é necessário posicionar a investigação que está a ser relatada e fazer um forte argumento para a sua contribuição de conhecimento. As expectativas para os resultados desta fase podem variar com base no tipo de contribuição do conhecimento e no estado do conhecimento na área de investigação (Vaishnavi et al., 2004). Neste trabalho de investigação os resultados foram analisados e os objetivos revistos sendo que as expetativas foram cumpridas. Foi esclarecido exatamente de que forma estes foram cumpridos bem como o seu significado para o presente trabalho. Adicionalmente também foram delineadas algumas áreas de melhoria e investigação complementar na secção destinada ao trabalho futuro.

2.3 Método de Revisão Bibliográfica

A revisão bibliográfica inicial centrou-se, principalmente, no livro *Adaptive Business Intelligence* de Michalewicz (Michalewicz et al., 2007), o que proporcionou uma base teórica sólida acerca dos conceitos de *Data Mining*, *Machine Learning*, Otimização e a necessidade de adaptabilidade nos sistemas de BI. Este foi um recurso essencial e importante para fundamentar o entendimento do conceito de sistemas ABI bem como as suas diversas aplicações. Em conjunção com literatura focada na implementação de modelos de ML e DM (Faria et al., 2022a; Lopes et al., 2021) esta investigação inicial ajudou a formar uma ideia bastante compreensiva do estado atual da implementação e utilização de sistemas ABI na

saúde atualmente. Isto permitiu iniciar o processo de conceção da solução apresentada no presente documento de investigação.

Conforme o progresso do projeto e da solução foram surgindo questões tanto técnicas como teóricas acerca das diversas fases e componentes que abrangem a organização e desenvolvimento moderno de software. Foi necessário efetuar, e incluir neste documento, uma análise mais aprofundada de literatura técnica e académica que aborda a construção e a gestão eficiente de sistemas de software complexos, com especial foco nas exigências modernas de flexibilidade, escalabilidade, performance, eficiência e segurança.

Para a revisão bibliográfica foram consultadas múltiplas fontes, incluindo artigos científicos e livros especializados, obtidos principalmente através das plataformas **Google Scholar**, **Science Direct** e **ResearchGate**. Na primeira fase desta pesquisa foram utilizadas como palavras-chave: Adaptive Business Intelligence; Prediction Models; Optimization Models; Adaptability Modules and approaches; Healthcare; Hospital 4.0; Data Mining; Microservices. Na segunda fase, mais focada no desenvolvimento de software, foram utilizados termos como: Best practices; Software Design Patterns; System Architecture; Database Architecture; Database Optimization; Virtualization; CI/CD; Software Testing; Web Security; OWASP.

A análise dos recursos académicos reunidos permitiu o desenvolvimento de uma solução compreensiva, eficiente e tematicamente relevante sem descorar as melhores práticas, segurança e os conceitos mais importantes no desenvolvimento moderno de software.

2.4 SCRUM Solo

A metodologia adotada para o desenvolvimento do sistema de interoperabilidade foi o SCRUM, porém fortemente adaptada para um âmbito individual onde existe apenas uma pessoa responsável pelo processo completo desde a conceção à entrega.

O SCRUM foi escolhido precisamente porque é uma metodologia ágil, iterativa e incremental onde o trabalho é dividido em pequenas porções e é realizado em iterações onde é possível a constante revisão de requisitos e contacto com as partes interessadas sendo que qualquer mudança necessária ao produto pode ser abordada de uma forma breve e atempada (Schwaber, 2010).

Sendo assim, a metodologia SCRUM foi adaptada para o desenvolvimento individual de software na medida que houve várias partes desta metodologia que permaneceram praticamente iguais, mas houve outras que deixaram de fazer muito sentido ou foram adaptadas ao desenvolvimento individual.

A metodologia SCRUM adaptada para o desenvolvimento individual é algo já experimentado e documentado em alguns casos de uso com um nível considerável de sucesso e benefícios como a organização do trabalho, uma estrutura bem organizada para lidar com projetos complexos, entrega do trabalho e resultados atempadamente e desenvolvimento e melhoria contínua durante todo o processo (Pagotto et al., 2016).

Neste aspeto o SCRUM individual que foi implementado durante o desenvolvimento do sistema de interoperabilidade consiste em:

- **Backlog de Produto**: Criação de uma lista de prioridades sobre as características e funcionalidades base do sistema. Esta lista foi sofrendo alterações e incrementos ao longo do processo de desenvolvimento;
- **Planeamento de Sprint**: Seleção de um subconjunto de itens de trabalho oriundas do Backlog de produto que irão ser abordadas durante a sprint em questão;
- **Sprint**: Períodos de uma semana onde os itens selecionados serão abordados e desenvolvidos;
- **Revisão de Sprint**: Demonstração do progresso e trabalho realizado durante a sprint às partes interessadas bem como a troca de ideias e feedback;
- **Retrospetiva de Sprint**: Reflexão acerca do sucedido durante a sprint, nomeadamente naquilo que correu bem e mal bem como a sugestão de estratégias para a prevenção daquilo que correu mal em futuras sprints.

O planeamento do desenvolvimento do sistema está diretamente ligado e dependente da metodologia adotada para esse efeito pelo que foi estabelecida, inicialmente, uma lista de tarefas chave ordenadas por prioridade, ou Backlog do produto, que seriam abordadas no decorrer das sprints. Este Backlog de produto foi incrementado e atualizado de forma contínua à medida que o sistema ia crescendo e as tarefas iam sendo cumpridas. Desta forma o produto foi sempre melhorando e crescendo de uma forma incremental e controlada.

Este processo de desenvolvimento desenrolou-se durante **7 meses** iniciando-se dia **1 de março de 2023** e terminando a **26 de setembro de 2023** e estendeu-se por exatamente **30 sprints** distintas. Como foi mencionado anteriormente o trabalho a realizar foi constantemente incrementado no *Backlog* de produto sendo que este seria considerado durante o planeamento das sprints e realizado durante a execução da *sprint* onde estivesse integrado. Seguindo o ciclo iterativo das 30 sprints atingiu-se o final do processo de desenvolvimento onde é possível observar todo o trabalho realizado.

Na Figura 3 é possível observar um excerto do trabalho realizado na forma de itens de trabalho e ter uma visão geral acerca de todo o desenvolvimento bem como as dependências entre eles.

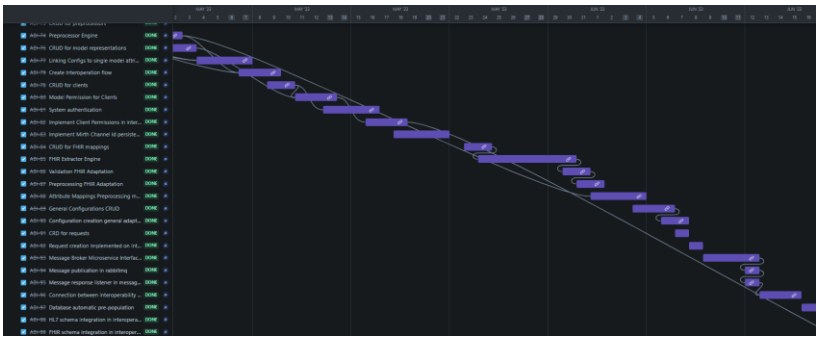

Figura 3 - Linha cronológica SCRUM do desenvolvimento

Todos os itens de trabalho foram documentados e criados ao longo do processo de desenvolvimento seguindo as melhores práticas e convenções da documentação de itens de trabalho para fácil entendimento, leitura e desenvolvimento como:

- **Nome claro e conciso**: Com um nome claro e conciso o trabalho necessário a fazer documentado no item será mais fácil de compreender e executar;
- **Descrição completa**: Na descrição dos itens de trabalho consta tudo o que um desenvolvedor necessita para realizar o trabalho de início a fim sem necessária a consulta da mais documentação;
- **Prioridade**: Todos os itens foram dotados de uma prioridade para facilitar o balanço de abordagem no planeamento das sprints;
- **Dependências**: Ao longo do processo de desenvolvimento existiram muitos itens de trabalho que dependiam do trabalho realizado noutros itens de trabalho por isso dependências entre itens foram criadas para facilitar a compreensão e o fluxo de desenvolvimento e evitar atrasos e mudanças de âmbito desnecessárias no meio de uma sprint (Silva et al., 2017);
- **Critérios de aceitação ou definição de completude**: Todos os itens foram dotados de uma definição de completude que traduz em que estado se pode aceitar o item como completo. Este aspeto é muito importante para melhor definir o âmbito do trabalho a realizar num dado item de trabalho (Silva et al., 2017).

Devido à implementação do SCRUM e a atenção às melhores práticas de documentação e organização o desenvolvimento desenrolou-se de uma forma calma, atempada e contínua sendo que o projeto esteve em constante crescimento semanal. O planeamento contínuo das sprints auxiliou bastante em manter o âmbito de cada semana claro e conciso, mas também à melhoria e levantamento de novos requisitos de forma contínua e controlada sendo que no final do processo de desenvolvimento foi obtido um sistema completo e modular pronto a ser implementado em qualquer ambiente e sujeito a testes.

2.5 Ferramentas para o desenvolvimento

A seguinte tabela lista todas as ferramentas utilizadas no decorrer do presente trabalho de dissertação.

Tabela 1 – Lista de ferramentas

Ferramenta	Descrição	Papel no Sistema
VS Code	IDE para desenvolvimento de aplicações e edição de código	Desenvolvimento e execução dos microserviços e para a criação da interface de frontend
Node.js	Ambiente de execução JavaScript com base no motor V8 do Google Chrome	Desenvolvimento e programação dos microserviços do sistema e para a criação da interface de frontend
Express	Framework para o desenvolvimento de aplicações web	Microserviços REST API e roteamento backend do sistema
MySQL	Sistema para a gestão de Base de Dados Relacional	Armazenamento e gestão dos dados do sistema
RabbitMQ	Plataforma para gestão de mensagens	Comunicação assíncrona entre o sistema de interoperabilidade e o sistema ABI
PostgreSQL	Sistema para a gestão de Base de Dados	Base de Dados da plataforma Mirth Connect
Mirth Connect	Plataforma de integração de recursos de saúde eletrónica	Integração e manipulação de recursos HL7
Git	Ferramenta para controlo de versões	Controlo de versão de todos os componentes do sistema
Github	Plataforma de hospedagem de repositórios de código fonte e controlo de versão	Controlo de versões e hospedagem dos repositórios de código fonte
Github actions	Automação de fluxo de trabalho CI/CD	Automação de implementação contínua dos microserviços e publicação de imagens
Github Container Registry	Gestão e distribuição de containers	Armazenar e distribuir containers contruídos a partir do código fonte dos microserviços e interface de frontend
React	Framework de JavaScript para a construção de interfaces gráficas na forma de aplicações web	Interface gráfica frontend do sistema
Ant Design	Biblioteca de componentes de UI	Auxílio no desenvolvimento da interface gráfica frontend do sistema
Docker	Plataforma para a contentorização para o desenvolvimento e implementação de aplicações	Criar e implementar os *containers* que formam todo o sistema
Docker Compose	Ferramenta para a definição e execução de aplicações que contenham múltiplos *containers*	Definir e orquestrar os múltiplos *containers* do sistema como um único sistema
Portainer	Gestão de *containers* docker	Facilitar a gestão e monitorização dos diferentes *containers* do sistema
Figma	Plataforma web para design de interface gráficas	Desenvolvimento dos *mockups* da interface gráfica de frontend
Diagrams.net	Plataforma *open source* para desenvolvimento de diagramas	Diagramas de representação do sistema e documentação
Swagger	Framework para documentação de APIs	Documentação de todos os microserviços
Sequelize	ORM para Node.js e Typescript	Interface com a base de dados relacional MySQL
JSON Web Tokens	Padrão *open source* para tokens de autenticação e autorização	Tokens para a autenticação e segurança do sistema
Bcrypt	Biblioteca para geração e validação de hash seguras	Encriptação de informação sensível do sistema

Monaco Editor	Editor de código para interfaces gráficas web	Escrita e alteração de scripts de preprocessamento
Jira	Plataforma de gestão de projetos	Gestão e processo de SCRUM para o desenvolvimento da solução
Postman	Plataforma de colaboração para desenvolvimento de APIs	Desenvolvimento e teste de todas as funcionalidades dos microserviços
Luna Modeler	Ferramenta de modelação de dados em DER para bases de dados relacionais	Desenvolvimento do diagrama DER para o modelo de dados relacional
Moon Modeler	Ferramenta de modelação de dados para bases de dados não relacionais	Desenvolvimento do diagrama ER do modelo de dados não relacional
MobaXTerm	Ferramenta para interfaces remotas	Conexão e interação com a Máquina Virtual Linux
Monaco Editor	Ferramenta de código na forma de uma API para aplicações Web	Edição de *scripts* na aplicação Web de Frontend
Ubuntu Linux	Distribuição Linux baseada em Debian para servidores e computação Cloud	Sistema operativo de escolha da máquina virtual que corre o ambiente de testes
Cypress	Ferramenta de teste de aplicações web	Criação e execução dos testes e2e do sistema
JMeter	Ferramenta para testes de carga em serviços	Testes de carga da solução

3. ESTADO DA ARTE

Neste tópico será abordado o estado e entendimento atual acerca dos conceitos, literatura e tecnologia, assim como a literatura relevante aos mesmos, referentes à implementação, arquitetura e requisitos de sistemas de ABI e interoperabilidade na área da saúde e desenvolvimento de software.

3.1 Business Intelligence

Business Intelligence (BI) é uma abordagem estratégica à gestão e transformação da informação em *insights*, que poderá ser utilizada para informar a tomada de decisão em toda uma organização. Envolve a recolha sistemática, armazenamento e análise de dados de múltiplas fontes para fornecer às organizações uma compreensão abrangente das suas operações internas e externas(Negash, 2004). O objetivo do BI é fornecer às organizações informação relevante, precisa e passível de ação que apoie a tomada de decisão informada, melhore o desempenho empresarial e ajude a impulsionar o crescimento. O BI engloba uma vasta gama de tecnologias, ferramentas e metodologias, incluindo armazenamento de dados, mineração de dados (Data Mining), relatórios e visualização, e gestão do desempenho. Estas tecnologias permitem às organizações capturar, armazenar e analisar grandes quantidades de dados estruturados e não estruturados de múltiplas fontes, tais como sistemas transacionais, sistemas de gestão de relações com clientes (CRM), e plataformas de redes sociais(Negash, 2004). Esta informação pode então ser transformada em *insights* e utilizada para apoiar a tomada de decisões em vários departamentos e níveis da organização.

Um dos principais benefícios do BI é que ajuda as organizações a identificar tendências, padrões, e relações dentro dos seus dados. Isto pode fornecer às organizações conhecimentos valiosos sobre o comportamento dos clientes, tendências de mercado, e desempenho operacional, permitindo-lhes tomar decisões orientadas por dados que melhoram os resultados do negócio(Negash, 2004). Por exemplo, ao analisar dados de vendas, um retalhista pode descobrir que um determinado produto está a vender bem em determinadas localizações geográficas, permitindo-lhes tomar decisões informadas sobre gestão de inventário e distribuição de produtos.

O BI também permite que as organizações monitorizem o desempenho e acompanhem o progresso em direção a metas e objetivos. Isto pode ajudar as organizações a identificar áreas a melhorar e a fazer os ajustamentos necessários para assegurar que estão a cumprir os seus objetivos e a fornecer valor aos seus clientes.

3.2 Sistemas ABI

Os sistemas de Inteligência Empresarial Adaptáveis, ou *Adaptive Business Intelligence* (ABI) são um tipo de tecnologia desenhados para apoiar as organizações na tomada de decisão informada com base na análise de dados em tempo real (Lopes et al., 2020). Os sistemas ABI visam fornecer às organizações uma abordagem abrangente e dinâmica à análise dos dados, à tomada de decisão e à aprendizagem contínua. São compostos por vários componentes inter-relacionados, incluindo recolha e gestão de dados, análises avançadas, técnicas de modelação e ferramentas de tomada de decisão (Michalewicz et al., 2007).

Os sistemas ABI permitem às organizações monitorizar, analisar e dar sentido a conjuntos de dados grandes e complexos, tanto estruturados como não estruturados, gerados por fontes internas e externas. Os dados recolhidos são processados e transformados em *insights* passíveis de ação, que poderão, posteriormente, ser utilizados para tomar decisões informadas que se alinhem com as metas e objetivos da organização.

Um dos principais benefícios dos sistemas ABI é a sua capacidade de aprender e evoluir continuamente (Michalewicz et al., 2007). Estes sistemas são concebidos para recolher e processar dados em tempo real, o que permite às organizações responder às mudanças no seu ambiente de negócios de forma rápida e eficaz. Isto também permite que as organizações monitorizem o impacto das suas decisões e façam modificações conforme necessário, assegurando que as suas decisões sejam sempre informadas pelos dados e *insights* mais recentes(Michalewicz et al., 2007).

Os sistemas ABI são altamente flexíveis e personalizáveis, o que significa que as organizações podem adaptá-los de modo a satisfazer as suas necessidades e exigências específicas. Isto permite que as organizações obtenham o máximo valor dos seus dados, bem como assegurem que a sua análise de dados e os seus processos de tomada de decisão sejam bem-adaptados às suas necessidades específicas (Lopes et al., 2021).

O conceito de ABI é utilizado quando, num único sistema, estão combinados modelos preditivos para a identificação de futuros cenários, assim como uma base de otimização que procura a melhor solução para um determinado problema com base nesses cenários (Figura 4). Sendo assim um sistema ABI estará sempre pronto para responder às perguntas **o que vai acontecer no futuro?** e **Qual será a melhor decisão nessa altura?** (Lopes et al., 2020)

Embora os sistemas ABI ofereçam muitos benefícios, existem também desafios associados à sua implementação. Estes podem incluir a complexidade da integração de dados, a necessidade de competências especializadas para implementar e manter os sistemas e a necessidade de medidas robustas de segurança e privacidade para proteger dados sensíveis.

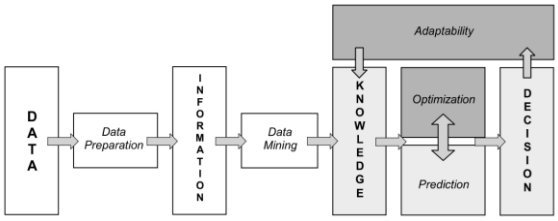

Figura 4 - Sistema Adaptive Business Intelligence (adaptado de (Michalewicz et al., 2007))

3.2.1 Data Mining

O Data Mining (Mineração de dados) é um processo computacional destinado a descobrir informação valiosa previamente desconhecida em grandes conjuntos de dados (como por exemplo bases e bancos de dados). Envolve a utilização de algoritmos matemáticos, estatísticos e de *machine learning* para extrair *insights*, conhecimento, padrões, relações, e tendências a partir de conjuntos de dados sem qualquer relação aparente (Michalewicz et al., 2007). O objetivo final da extração de dados é fornecer às organizações uma compreensão mais profunda dos seus dados e apoiar a tomada de decisões com base no conhecimento extraído dos mesmos.

O processo de Data Mining envolve múltiplas etapas, incluindo a recolha, limpeza e preparação de dados, modelação de dados, interpretação e avaliação dos resultados obtidos. Os dados recolhidos são

frequentemente muito volumosos, complexos e não estruturados, o que dificulta a sua compreensão e utilização sem ferramentas computacionais especializadas.

Os algoritmos para Data Mining analisam os dados de forma a identificar padrões, relações, e correlações, e utilizam estes conhecimentos para fazer previsões ou informar/ajudar na tomada de decisão(Michalewicz et al., 2007). Os resultados do processo de Data Mining podem ser utilizados em qualquer contexto organizacional produtor de dados para apoiar a resolução de problemas, no planeamento estratégico, melhorar produtos e serviços, e impulsionar a inovação dentro da organização.

3.2.2 Previsão

No contexto de Data Mining, a previsão refere-se ao processo de utilização de técnicas de análise de dados para fazer uma previsão sobre eventos ou tendências futuras. O objetivo da previsão é utilizar dados passados de forma a construir e treinar um modelo que possa ser utilizado para efetuar previsões sobre resultados futuros proporcionando assim conhecimento útil e ajudando na tomada de decisão (Figura 5).

Existem quatro grandes áreas no que toca a métodos de previsão(Michalewicz et al., 2007):

- Métodos Matemáticos – Representam o sistema ou processo sob a forma de equações matemáticas. São utilizados para descrever relações entre variáveis e para efetuar previsões de resultados futuros com base em dados passados. (Ex: Regressão Linear, Regressão Logística e modelos de Séries Temporais);
- Métodos de Distância – Utilizam uma métrica de distância para comparar e classificar dados sob a forma de pontos. São normalmente utilizados em *clustering*, onde os pontos são agrupados por semelhança e em classificação onde são atribuídas categorias predefinidas a cada grupo de pontos (Ex: k-nearest neighbors (KNN) e k-means);
- Métodos Lógicos – Utilizam regras lógicas para realizar a previsão. São frequentemente utilizados em sistemas de apoio à decisão, onde um conjunto de regras é utilizado para tomar decisões com base nos *inputs*. (Ex: árvores e tabelas de decisão)
- Métodos de Heurística Moderna – Utilizam a heurística ou "regras de ouro" de forma a encontrar soluções aproximadas para um problema. A sua natureza torna-os extremamente populares em algoritmos de otimização, onde estamos à procura da melhor combinação de parâmetros, mas também na previsão. (Ex: Redes Neurais e algoritmos genéticos)

Após a recolha e preparação dos dados recorre-se a estes métodos de previsão de forma a construir e treinar modelos de previsão.

Os modelos de previsão podem ser usados para fazer previsões sobre uma variedade de resultados, tais como tendências de vendas, comportamento de clientes e até preços de ações. Ao analisar padrões e relações em dados passados, um modelo preditivo pode identificar os fatores mais suscetíveis de influenciar resultados futuros (através da análise correlacional) e utilizar esta informação para fazer previsões precisas.

A previsão é um aspeto central em ABI, pois ajuda as organizações a prepararem-se para o futuro fornecendo conhecimento e *insights* com base em eventos e dados passados. Por exemplo, ao utilizar modelos de previsão para analisar dados de vendas, um retalhista poderá ser capaz de identificar os clientes com maior probabilidade de fazer uma compra num futuro próximo, permitindo-lhes direcionar os seus esforços de marketing de forma mais eficaz.

Embora o processo de previsão seja algo muito poderoso, nem sempre é exato pelo que se torna importante avaliar a fiabilidade de um modelo de previsão antes de o utilizar para tomar qualquer decisão(Faria et al., 2022b).

Figura 5 - Componente preditiva de ABI (adaptado de (Michalewicz et al., 2007))

3.2.3 Otimização

A otimização refere-se ao processo de encontrar a melhor solução para um determinado problema, ajustando variáveis e parâmetros dentro de um qualquer modelo (Figura 6). O objetivo da otimização é encontrar a combinação de variáveis e parâmetros que resultem no melhor resultado possível, tais como o lucro máximo, o custo mínimo, ou a maior precisão (Michalewicz et al., 2007).

A otimização é prevalecente em várias indústrias pelo que todas elas visam a excelência e encontram-se na procura constante por formas de otimizar a sua atividade, como por exemplo a redução de custos ou o aumento da eficiência (Fernandes et al., 2022).

Ao longo dos anos foram desenvolvidas, as chamadas, técnicas clássicas de forma a abordar a otimização como a programação linear, programação dinâmica e programação de fluxo de rede. Porém, recentemente surgiu uma nova classe de técnicas de otimização, referida como a heurística moderna (Michalewicz et al., 2007). As técnicas que formam esta nova classe diferem das técnicas clássicas na medida que empregam uma abordagem mais flexível e adaptável para encontrar soluções. Foram

concebidas para explorar o espaço de solução e encontrar soluções quase ótimas em vez da solução ótima concreta e são muito úteis para resolver problemas complexos de otimização com múltiplos objetivos e restrições. Fazem parte da heurística moderna técnicas como a otimização local, o stochastic hill climber, o simulated annealing, o tabu search e algoritmos evolutivos (Michalewicz et al., 2007).

- **Otimização Local (Hill Climbing)** – Na otimização local a função de avaliação determina a qualidade de uma solução sob a forma de uma topografia de colinas e vales. Encontrar a melhor solução é comparável a procurar um cume/pico numa cadeia de montanhas onde a visibilidade é limitada, pelo que apenas podemos tomar decisões locais. A técnica de Hill Climbing trata-se de um processo básico de otimização local que envolve melhorar a solução ótima de forma iterativa através da seleção de novas soluções com base na solução atual, escolhendo aquela com melhor pontuação na medida de qualidade. No entanto, o resultado do Hill Climb é dependente da solução inicial, podendo apenas fornecer soluções ótimas locais que se podem desviar da solução ótima global. É uma técnica de fácil implementação, porém pode ser um desafio aquando da presença de muitas soluções ótimas locais;
- **Stochastic Hill Climber –** É uma modificação do algoritmo Hill Climbing onde a ideia é introduzir um elemento probabilístico no processo de aceitação de novas soluções a fim de se escapar à solução ótima local permanente;
- **Simulated Annealing –** Utilizado para encontrar uma solução ótima global para um dado problema de otimização. É similar ao stochastic hill climbing na medida em que pode aceitar uma solução inferior como uma nova solução atual, porém um algoritmo SA altera o valor de temperatura (T) ao longo do *runtime*. No início do processo o valor de T é alto fazendo com que se assemelhe a uma pesquisa aleatória, mas à medida que T desce o processo começa a assemelhar-se cada vez mais a um hill climbing.
- **Tabu Search –** Utilizado para encontrar uma solução aproximada num problema de otimização. É inspirado pela ideia de permitir que as soluções desconsiderem os movimentos recentes de forma a fugir das soluções ótimas locais sendo que soluções recentes sejam "tabu" ou proibidas. O algoritmo gera, iterativamente, novas soluções e seleciona a melhor, sendo que ao mesmo tempo mantém um registo das soluções recentemente utilizadas e evita que sejam utilizadas de novo demasiado cedo.

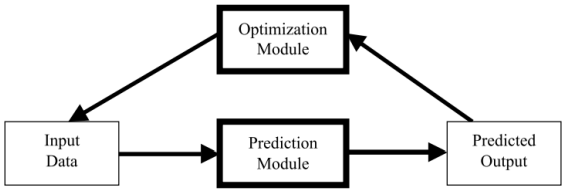

Figura 6 - Componente de Otimização de ABI (adaptado de (Michalewicz et al., 2007))

3.2.4 Adaptabilidade

A adaptabilidade refere-se à capacidade dos dados e sistemas analíticos de uma organização evoluírem e mudarem em resposta a novas necessidades empresariais, condições de mercado e ambiente e tecnologias emergentes, tudo fatores em constante mudança nos dias de hoje (Michalewicz et al., 2007). Abrange os processos, ferramentas e infraestruturas necessárias para apoiar a integração de novas fontes de dados e tecnologias, bem como a capacidade de modificar ou melhorar rapidamente os sistemas existentes para satisfazer requisitos em mudança. A adaptabilidade significa ser capaz de responder a novas informações e *insights* em tempo real, tornando-a uma componente essencial para as organizações permanecerem competitivas e tomarem decisões informadas com base em dados em constante mudança (Michalewicz et al., 2007). Realizar a previsão e otimização é um ótimo começo, porém as previsões de hoje poderão ser erradas/irrelevantes amanhã pelo que é necessário que haja uma capacidade de fazer ajustes com base nos erros (Michalewicz et al., 2007).

O modulo de adaptabilidade relaciona os *outputs* e *inputs* recentes de um modelo de forma a identificar erros e inconsistências indicativas de erros de previsão/otimização e procede a ajustes constantes nos parâmetros do modelo em questão (Figura 7).

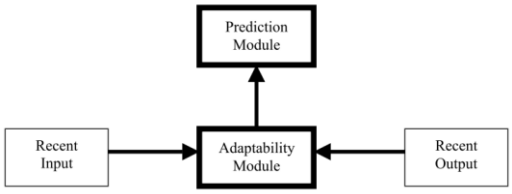

Figura 7 - Componente de Adaptabilidade de ABI (adaptado de (Michalewicz et al., 2007))

3.3 Integração de Sistemas

A integração de Sistemas de Informação refere-se ao processo de combinar diferentes tecnologias de informação (TI), fontes de dados e sistemas computacionais dentro de uma organização, de forma que

seja possível a comunicação e troca de informação como um único sistema que tem por objetivo melhorar a eficiência, reduzir custos e proporcionar uma melhor experiência ao utilizador final.

Ao elaborar a arquitetura para a implementação de um sistema de informação, é crucial considerar que tecnologias e protocolos serão empregues, que sistemas e soluções computacionais serão integrados e como serão organizados, com o objetivo de cumprir os requisitos levantados para o mesmo de forma eficaz e rentável.

Problemas como a escalabilidade e rentabilidade do sistema têm de ser considerados, não só no desenho da arquitetura, mas também na escolha das tecnologias a utilizar sendo que se destaca a tecnologia de virtualização e "conteinerização" pelas suas características altamente eficientes e rápidas aplicada num contexto de redundância de rede ou redundância de serviços de forma a atingir um sistema altamente acessível ou disponível (High Availability/HA).

No contexto de arquitetura destacam-se dois grandes conceitos que, algo parecidos, possuem diferenças relevantes a ter em conta na hora da escolha:

SOA (Arquitetura Orientada a Serviços) – Construção e implementação de serviços de forma centralizada com o objetivo de decompor um sistema de grande dimensão em serviços mais pequenos, controláveis e internamente reutilizáveis que podem, facilmente, ser combinados para formar uma solução completa (Figura 8) (Erl, 2005).

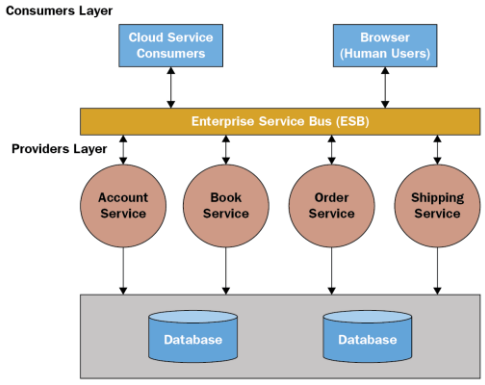

Figura 8 - Arquitetura Service Oriented Architecture (SOA)

Microserviços - Construção e implementação de aplicações complexas como um conjunto de pequenos serviços, que podem ser desenvolvidos, implementados e mantidos de forma independente, os quais comunicam entre si através de APIs. A abordagem de microserviços permite uma maior escalabilidade, resiliência, flexibilidade e acessibilidade bem como a capacidade de alteração dos serviços sem afetar todo o sistema (Figura 9)(Hawilo et al., 2019).

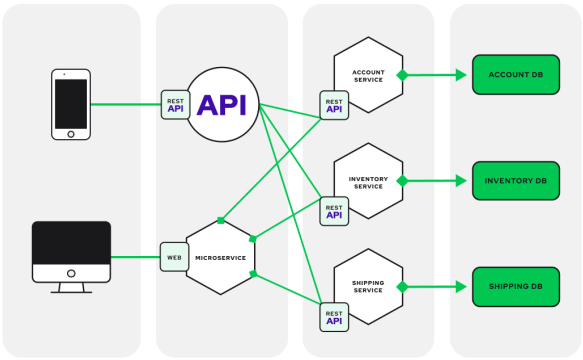

Figura 9 - Arquitetura de Microserviços

A grande diferença nestas duas abordagens está no nível da granularidade e no foco de cada uma, enquanto SOA se foca na criação de uma plataforma centralizada para a integração de serviços, os microserviços estão focados na divisão de um sistema em serviços de desenvolvimento, implementação e funcionamento independente (Cerny et al., 2017).

Na vanguarda para a implementação e gestão dos serviços que compõe ambas as arquiteturas está a tecnologia de virtualização que permite a criação de ambientes virtuais singulares capazes de hospedar microserviços, mais concretamente o conceito de arquitetura de *Network Function Virtualization* (NFV) que possui muitas vantagens sobre o desenvolvimento tradicional da função de rede. Neste contexto cada VNFC (Virtual Network Function Container) é definido como um serviço que executa um conjunto limitado de funções com uma pequena base de código (Hawilo et al., 2019). A natureza modular dos serviços facilita uma transição gradual para versões posteriores dos mesmos tornando a sua manutenção muito mais fácil. Novos serviços podem ser integrados, serviços existentes podem ser atualizados, aumentados ou até replicados sem perturbar o funcionamento do sistema (Figura 10).

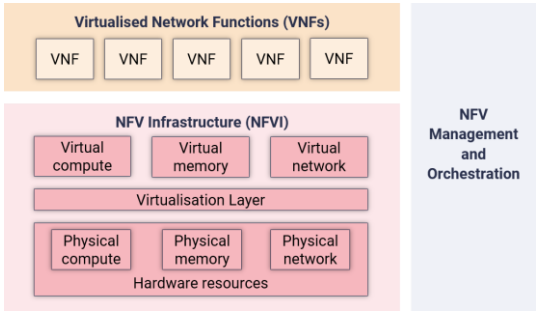

Figura 10 - Arquitetura Network Function Virtualization (NFV)

3.4 Integração de Sistemas ABI (Casos de Estudo)

Ao longo do desenvolvimento do presente documento foram analisados alguns casos de estudo e artigos científicos relacionados com a arquitetura, implementação, conceitos e tecnologias para o desenvolvimento de sistemas ABI na área da saúde.

3.4.1 Previsão (Casos de Estudo)

Na pesquisa pelos modelos que constituem um sistema ABI e a sua implementação no setor da saúde foram encontrados alguns artigos e casos que representam e abordam a utilização da metodologia CRISP-DM na recolha, preparação e utilização de dados de origem no setor da saúde para o desenvolvimento de modelos e algoritmos de previsão capazes de gerar *insights* passíveis de ação e conhecimento, outrora desconhecido, com base nesses mesmos dados.

O tipo de modelo, algoritmo ou técnica de previsão utilizados varia consoantes os requisitos do problema ou a natureza da questão de investigação, porém todos os casos estudados geram conhecimento novo de valor com base em dados históricos através de modelos cuja integração num sistema passaria pela execução em *scripts* ou pela serialização e acesso isolado ao próprio modelo.

Os casos estudados justificam a sua própria existência com o facto de que as instituições de saúde geram, diariamente, um grande volume de dados com o potencial de ajudar a tomada de decisão, quer num contexto administrativo, como também num contexto de diagnóstico clínico (Faria et al., 2022b).

O primeiro caso, **Predictive analytics for hospital discharge flow determination**, centra-se na aplicação da análise preditiva na gestão de fluxo de pacientes, particularmente na Unidade de Cuidados Intensivos (UCI). O objetivo deste estudo é otimizar a utilização dos recursos hospitalares de forma a diminuir a *Lenght of Stay (LOS)* ou período de estada dos pacientes na UCI tratando-se assim de um problema administrativo (Faria et al., 2022b). Para atingir este objetivo são empregues modelos de *Machine Learning* para analisar os dados passados e informações clínicas de forma a prever a incógnita LOS. Foi seguida a metodologia CRISP-DM para o estudo e desenvolvimento dos modelos sendo testadas várias combinações de cenários, modelos e conjuntos de dados o que resultou num modelo final capaz de prever com bastante fiabilidade o período LOS de um paciente aquando da sua admissão na UCI. A integração deste modelo num sistema ABI seria uma mais-valia pelo que apenas seria necessária a sua serialização para uso posterior, ou a disponibilização da *script* para treino e utilização do modelo. O modelo resultante deste estudo enquadrar-se-ia na componente Preditiva de um sistema ABI sendo que poderia ser chamado por outros serviços do mesmo ou pelo ambiente externo.

O segundo caso, **Prediction of Mental Illness Associated with Unemployment Using Data Mining**, centra-se na utilização de técnicas de *Data Mining* para a previsão de doenças mentais assim

como correlacionar o seu desenvolvimento com o desemprego e outras potencias causas. Seguindo a mesma norma do artigo anterior, é explorada a metodologia CRISP-DM durante o desenvolvimento dos modelos, porém a ferramenta RapidMiner foi utilizada para o processo de *Data Mining* resultando em conclusões promissoras com métricas de avaliação como a **precisão, sensibilidade** e **especificidade** a reportar valores acima de 90%, mesmo implementando a técnica de *Cross Validation* com 10 *folds*(Gonçalves et al., 2020). Este modelo seria uma adição brilhante a qualquer sistema ABI direcionado à área da saúde pelo que apenas seria necessária a sua extração e capacidade de uso independente. O modelo enquadrar-se-ia na componente preditiva do sistema ABI que, aquando de ser chamado, teria de apenas retornar as previsões com base nos *inputs*.

Os artigos estudados provam, não só, que há um grande valor na implementação de técnicas de *Data Mining* e algoritmos de previsão no setor da saúde, como também demonstram a sua efetividade (Gonçalves et al., 2020). A implementação de qualquer modelo preditivo num sistema ABI é apenas condicionada pelo acesso isolado ao modelo, pelo que se este parâmetro for cumprido a sua implementação torna-se algo replicável e automatizável dentro de um dos muitos serviços que compõe um sistema ABI.

3.4.2 Otimização (Casos de Estudo)

A componente de Otimização num sistema ABI consiste em utilizar técnicas e modelos de otimização de forma a encontrar a melhor solução possível para um determinado problema representado por uma série de *inputs*. A ideia será utilizar o *output* de um modelo de previsão para servir como *input* no modelo de otimização sendo que este irá devolver a melhor solução para o problema em mãos com base nos dados previstos na componente de previsão.

Para o estudo desta componente foram analisados dois estudos que se concentram na utilização de técnicas de otimização moderna para encontrar uma solução ótima para um problema hospitalar administrativo ou diagnóstico.

O primeiro caso estudado, **Decision models on therapies for intensive medicine**, centra-se na aplicação de modelos de otimização no campo da medicina intensiva, mais especificamente na Unidade de Cuidados Intensivos (UCI). Foi utilizado o algoritmo de otimização moderna Stochastic Hillclimb de forma a identificar os melhores medicamentos para tratar uma condição específica, como a pneumonia causada pela covid-19, considerando os três principais medicamentos administrados bem como o custo do tratamento (Passos et al., 2022). Foram realizados seis cenários de teste utilizando diferentes combinações do número de iterações e probabilidade de aceitar uma nova solução de forma a escapar

às melhores soluções locais. O estudo ainda destaca a importância da utilização de modelos de otimização em conjunção com modelos preditivos de forma a reduzir o erro de previsão. Esta interação surge da necessidade dos modelos de Data Mining serem otimizados ao longo do tempo, constantemente treinados para aumentarem a sua acuidade e retornarem soluções mais fidedignas (Passos et al., 2022). Após a modelação e uso dos modelos foi averiguado que bisacodyl é a medicação mais eficaz no tratamento da Pneumonia causada pela covid-19 assim como possui o preço mais baixo (Passos et al., 2022). O caso de estudo revela a eficácia dos modelos de otimização na resolução de problemas logísticos e complexos na área da saúde, porém é bastante limitado pelo que apenas um modelo de otimização foi testado. De forma a retirar melhores resultados deviam ter sido testados múltiplos modelos de otimização com múltiplas configurações e cenários.

O segundo caso, **Adaptive Business Intelligence in healthcare – A platform for optimizing surgeries**, foi um dos casos mais importantes nesta revisão pelo que demonstra a interação entre modelos preditivos e modelos de otimização assim como o potencial da abordagem ABI aquando da sua aplicação no setor da saúde. O artigo centra-se no problema complexo de agendamento de cirurgias em blocos operatórios hospitalares. A combinação de dados operacionais e ferramentas analíticas permite que o sistema ABI faça uma previsão acerca dos tempos de cirurgia, com base no metadados acerca da cirurgia, e com base nesses tempos utilizar modelos de previsão para otimizar o processo de agendamento, reduzindo assim o desperdício de recursos (Ferreira et al., 2019). Sendo o sistema foi dividido em duas componentes, previsão onde foi utilizada a ferramenta Weka e otimização onde foi utilizado o RStudio. A componente de previsão irá prever o tempo de duração de uma cirurgia e a componente de otimização com base nestas previsões irá tentar encontrar a solução ótima de agendamento de forma a minimizar o desperdício de tempo e turnos, prevenir atrasos e aumentar a eficiência da agenda (Ferreira et al., 2019). Na componente de previsão foram utilizadas as técnicas **linear regression, k-nearest neighbors, decision tree, support vector regresion** e **multi-layer perceptron**. Para a componente de otimização foram apenas utilizados os algoritmos **hill climb** e **simulated annealing**.

Para integrar os modelos de otimização num sistema ABI seria necessário o acesso isolado a cada um deles para poderem integrar a componente de otimização do sistema que estaria em constante comunicação com a componente preditiva do mesmo sistema. Assim as previsões realizadas pelos modelos de previsão serviriam como *input* para os modelos de otimização de forma que estes pudessem encontrar uma solução ótima com base nos dados previstos. Esta capacidade permitiria aos analistas explorarem as soluções ótimas com base no futuro, ainda que incerto.

3.4.3 Adaptabilidade e Arquitetura de Sistemas ABI (Casos de Estudo)

No contexto de um sistema ABI a adaptabilidade não só corresponde à capacidade do sistema se conseguir adaptar a novos ambientes e fatores com escalabilidade, mas também ao facto de que as soluções dadas por um sistema ABI podem tornar-se rapidamente desatualizadas sendo que é necessária uma componente de adaptabilidade que irá, incrementalmente, ajustar os modelos existentes com base nas soluções recentes.

O 1º caso estudado, **Adaptive Business Intelligence: A New Architectural Approach**, foca-se na proposta de uma arquitetura para um sistema ABI para responder à rápida acumulação de dados úteis pelas instituições de saúde. A arquitetura proposta permite que o sistema(Lopes et al., 2020):

- Aceda a dados de fontes diferentes;
- Transforme dados em informação;
- Faça previsões com base em resultados de Data Mining;
- Modelos de otimização integrados com os modelos preditivos de forma a permitir a exploração de novas e melhores soluções;
- Modulo de adaptabilidade integrado nos modelos de otimização de forma a adaptar os resultados ao contexto do problema;
- Visualização do conhecimento adquirido pelo processo arquitetural do sistema.

Para responder às necessidades de escalabilidade e adaptabilidade a arquitetura está planeada com a integração dos diversos serviços como microserviços através da ferramenta de containerização e virtualização Docker (Lopes et al., 2020). Toda a comunicação entre os microserviços neste ecossistema de NFV será realizada via protocolos web fornecendo inputs à componente de previsão. A arquitetura neste artigo está bastante bem conseguida dando foco à integração dos modelos de otimização com os modelos de previsão, transformando assim um sistema preditivo tradicional num sistema ABI, porém a componente de adaptabilidade responsável pela correção errónea atualização gradual dos modelos está muito pobre não tendo qualquer especificação à parte da sua integração interna com os modelos de otimização.

O segundo caso estudado, **Adaptive Business Intelligence platform and its contribution as a support in the evolution of Hospital 4.0**, explora o papel que um Sistema ABI poderá ter na evolução para o Hospital 4.0 e expõe as limitações da 1ª arquitetura planeada através da ferramenta Docker. Esta arquitetura provou ser eficiente, porém existem algumas limitações que necessitam de ser abordadas como uma grande dependência em bases de dados poderosas, processo virtualizados únicos, pouca

tolerância a falhas e dificuldades na gestão dos processos e dados (Lopes et al., 2021). De forma a ultrapassar estas limitações é proposto a utilização de tecnologias de *Big Data*, *cloud computing* e o conceito de adaptabilidade. As tecnologias propostas, **HDFS, Storm, Kafka, Hive, HBase, Kubernetes, Docker, CloudStack** e **Apache Spark** pretendem melhorar a eficiência do sistema no processamento e gestão dos dados e aumentar a estabilidade e disponibilidade/acessibilidade do sistema (Lopes et al., 2021). Neste artigo o conceito de adaptabilidade do sistema foi, novamente, mencionado, porém não houve especificação quer tecnológica quer arquitetural que o abordasse.

A revisão de literatura conduzida obteve alguns bons resultados no que toca à visão atual das arquiteturas ABI aplicadas à saúde na medida da implementação de uma componente preditiva integrada com uma componente de otimização, porém também revelou a maturidade, ou a falta dela, do conceito de adaptabilidade em sistemas de informação.

3.4.4 Conclusões Acerca dos Casos de Uso

Todos os casos de estudo analisados são extremamente relevantes na justificação da utilização de sistemas ABI e modelos de ML/DM como ferramentas de apoio, melhoria e aceleração na área da saúde e diagnóstico, tratamento de pacientes e organização da profissão médica. Qualquer um destes casos de estudo comprova a eficácia da implementação deste tipo de tecnologia nesta área e contexto, as suas dificuldades e benefícios (Faria et al., 2022b; Gonçalves et al., 2020) tendo sempre em comum a necessidade do desenvolvimento arbitrário de software para a sua utilização e implementação. Todas as soluções exploradas partilham da falta de flexibilidade e tolerância para mudanças de ambiente ou até mesmo atualizações feitas ao produto final que serão os modelos (Lopes et al., 2020). A necessidade de garantir a robustez, modularidade, facilidade de uso ou implementação e a ininterruptabilidade arquitetónica não foram temas abordados ou explorados no leque de literatura reunida pelo que estes serão os requisitos base na proposta e desenvolvimento da arquitetura de interoperabilidade.

3.5 Bases de Dados

Na base de qualquer sistema informático complexo está uma base de dados da qual este depende para efetuar a maioria das suas operações. Para poder obter um sistema eficaz, rápido e escalável é muito importante que exista uma boa representação, modelação e planeamento do fluxo de dados que passará e será armazenado no sistema. Este fluxo de dados e o seu planeamento estão muito dependentes do tipo de base de dados a utilizar. Para este fim existem duas grandes opções de bases de dados,

relacionais ou não relacionais pelo que cada um possui as suas vantagens e desvantagens situacionais (Jatana et al., 2012). Estas estão presentes na Tabela 2.

Tabela 2 - Tabela de Diferenças entre bases de dados relacionais e não relacionais

Comparação	Base de Dados Relacional	Base de Dados Não Relacional
Diferenças Chave	• Armazena dados em tabelas representativas de entidades • Tabelas estão ligadas por relacionamentos, o que permite que os dados sejam reutilizados por toda a base de dados • Utilizam uma linguagem padrão (SQL) para pesquisar e manipular os dados	• Armazena dados em muitos formatos diferentes como JSON ou XML; • Não possuem um esquema fixo o que permite muito mais flexibilidade; • Utilizam uma linguagem de consulta própria o que pode dificultar uma mudança súbita de base de dados
Vantagens	• Transações atómicas, consistentes, isoladas e duradouras (ACID); • Ferramentas para a garantia da integridade dos dados; • Exigência de um esquema que define a estrutura dos dados e ajuda a garantir a consistência dos mesmos	• São altamente escaláveis o que as torna muito eficientes ao manusear quantidades grandes de dados; • São muito flexíveis pelo que são uma ótima escolha quando é espectável que a estrutura dos dados se altere ao longo do tempo; • São extremamente rápidas especialmente quando a consulta é do tipo chave-valor
Desvantagens	• Podem ser lentas para consultas complexas, especialmente quando existem muitos dados • Não são ideias para o armazenamento de dados não estruturados como imagens, vídeos ou ficheiros	• Não há garantia de transações ACID pelo que poderá levar à possível corrupção de dados e erros; • Não possuem mecanismos para a garantia de integridade dos dados; • Não permitem a exigência de um esquema de dados pelo que a consistência dos dados também não é garantida
Representação de Dados	• DER/ERD • Relacionamentos bem definidos entre tabelas	• Modelo de Dados • *Nesting* de dados e referências entre coleções

3.5.1 Normalização de modelos de dados

A normalização consiste no processo de organização dos dados numa base de dados de forma a minimiza a redundância e aperfeiçoar a integridade dos dados. Quando as regras de normalização não são tidas em consideração é muito provável que a base de dados acabe por guardar os mesmos dados em vários locais diferentes sendo que isto consiste num desperdício de espaço e memória bem como aumenta os riscos de inconsistências nos dados como por exemplo um utilizador com duas moradas diferentes (Kolahi & Libkin, 2006). Bases de dados normalizadas são, também, muito mais fáceis de manter, consultar e manipular pelo que todas as operações nela poderão ser traduzidas por consultas

simples. A normalização de bases de dados também melhora a sua segurança pelo que torna o acesso a informação sensível mais difícil por parte de utilizadores não autorizados (Kolahi & Libkin, 2006; Lee, 1995).

Existem, essencialmente, três formas normais, cada uma servindo para eliminar uma forma de redundância diferente. As três formas normais mais comuns são:

- **1ª Forma Normal (1NF)**: Requer que todos os valores de atributos de uma tabela sejam atómicos não podendo ser subdivididos. Exige, também, que cada atributo tenha um nome único;
- **2ª Forma Normal (2NF)**: Requer a normalização 1NF e que todos os atributos não-chave de uma tabela sejam totalmente dependentes da chave primária sendo que são exclusivamente determinados por esta;
- **3ª Forma Normal (3NF)**: Requer a normalização 2NF e que todos os atributos não-chave de uma tabela sejam diretamente dependentes da chave primária pelo que nenhum atributo não-chave pode ser transitivamente dependente da chave primária.

Na pesquisa bibliográfica realizada sobre este tópico destacaram-se dois artigos *"On Redundancy vs Dependency Preservation in Normalization: An Information-Theoretic Study of 3NF"* de Solmaz Kolahi e Leonid Libkin bem como *"Justifying Database Normalization: A Cost/Benefit Model"* por Heeseok Lee onde os seus autores apresentaram fortes justificações tanto teóricas como práticas para a utilização da normalização na construção de uma base de dados.

Kolahi e Libkin demonstram que a norma 3NF possui o menor preço de preservação de dependências enquanto minimiza a redundância dos dados de entre todas as formas normais para este fim (Kolahi & Libkin, 2006).

Lee propõe um modelo de custo/benefício para determinar a forma normal ótima para uma base de dados baseando-se na redução de anomalias, requisitos de armazenamento e tempos de resposta de transações concluindo que a forma normal 3NF é a forma ótima uma vez que as suas vantagens são superiores aos seus custos e à sua superioridade em relação às restantes formas normais (Lee, 1995).

3.5.2 Soluções ORM

Soluções ORM, ou Mapeamentos Relacionais de Objetos, consistem numa técnica de programação que tem como objetivo a conversão de dados entre uma base de dados relacional e um paradigma orientado a objetos pelo que toda a comunicação e interface com a base de dados é realizada através de um objeto

representativo desta abstraindo assim toda a linguagem de consulta e possíveis erros de implementação (Torres et al., 2017). Este comportamento e abstração estão representados na Figura 11 onde é possível verificar o conceito da conversão de linhas de dados relacionais em objetos de forma a facilitar todo o desenvolvimento e interface.

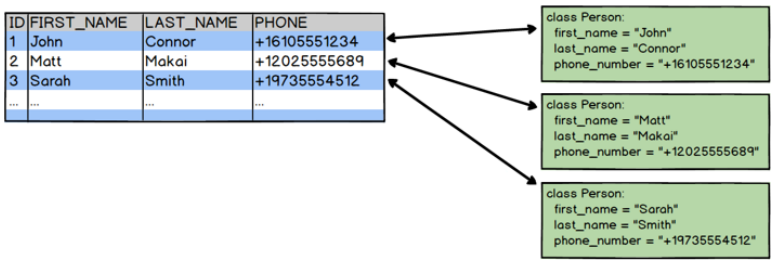

Figura 11 - Diagrama ORM

Esta camada de abstração permite que o desenvolvimento seja mais fácil e agradável proporcionando várias vantagens atrativas:

- **Produtividade**: Soluções ORM podem automatizar muitas tarefas que, geralmente, estão associadas à interface com uma base de dados como a criação, atualização, alteração, inserção e consulta de dados bem como a gestão das relações entre as diferentes tabelas da mesma. Sendo que estas operações base se encontram abstraídas e já implementadas o desenvolvedor tem mais disponibilidade para se focar nos aspetos de valor da sua aplicação ou sistema;
- **Qualidade de código**: Uma vez que impõe restrições e regras na interface com a base de dados os ORMs tornam todo o processo de interação mais coerente e rígido de forma a evitar situações erróneas e garantir a consistência de interface;
- **Redução da complexidade**: Sendo que ORMs proporcionam uma camada de abstração para estabelecer uma interface com uma base de dados, tornam o código necessário para o fazer mais simples e fácil de compreender e manter.

3.6 Padrões de Arquitetura de microserviços

O conceito de microserviço é algo complexo e relativamente recente, portanto existe alguma confusão no que toca ao seu desenvolvimento correto e eficiente, porém existem vários padrões bem estabelecidos para conceber e implementar arquiteturas com base em microserviços (Taibi et al., 2018). Estes padrões foram concebidos, desenvolvidos e estudados para otimizar e focar o desenvolvimento de arquiteturas de microserviços ágeis, escaláveis e resilientes. O estudo realizado por Davide Taibi e Valentina

Lenarduzzi agrupa estes padrões em 3 grandes grupos que podem e devem ser utilizados em conjunto de forma a maximizar a eficiência e robustez do sistema.

3.6.1 Padrões de Orquestração e Coordenação

Estes padrões focam-se na forma como os diferentes microserviços comunicam entre si bem como na gestão destas interações. O conceito de microserviço, tipicamente, constitui programas pequenos e independentes sendo que se torna necessário coordenar as suas funcionalidades por meio de mecanismos adicionais.

- **Padrão de API Gateway** – Ponto de entrada central que é capaz de encaminhar o tráfego de pedidos para os serviços adequados. Trata-se do ponto de convergência dos diversos serviços individuais sendo que é capaz de fazer uso de todas as suas funcionalidades. Este padrão simplifica a interação entre os clientes e os microserviços e, tipicamente, lida com as preocupações principais como autenticação, autorização e monitorização num serviço só. Constitui, no entanto, um ponto único de falha pelo que se este serviço falhar a comunicação entre cliente e microserviço fica impedida.
- **Padrão de descoberta de serviços** – Microserviços são, por natureza, programas implementados de forma independente e separada pelo que é necessário que haja um processo de descoberta destas implementações para que o sistema possa funcionar como um todo. Este problema é tipicamente resolvido com a integração de um registo de serviços. A integração desta solução pode ser imposta ao cliente sendo que aumenta a sua complexidade de uso, ou ao servidor aumentado os custos de manutenção de mais um serviço.
- **Padrão Híbrido** – Combina o padrão de *API Gateway* com um *message bus* ou gestor de mensagens para efetuar a comunicação. Esta estratégia aumenta a complexidade da solução bem como o percurso que uma mensagem tem de efetuar desde que parte do cliente até que chega a um microserviço mas garante e facilita o fluxo de comunicação.

3.6.2 Padrões de Implementação

Os padrões de implementação focam-se na forma como os diversos microserviços são empacotados, implementados e escalados em ambientes diferentes.

- ***Multiple services per host*** – Múltiplos microserviços são implementados e executados num só sistema hospedeiro quer este seja uma máquina física, uma máquina virtual ou um *container*.

Esta abordagem é bastante atrativa pelo que é capaz de poupar em recursos e simplifica a gestão dos serviços. Porém constitui um ponto único de falha e obriga a que os diversos microserviços partilhem os mesmos recursos. Isto não costuma ser um problema para a maioria das implementações, porém é um fator limitador a considerar aquando da escolha arquitetónica.

- ***Single service per host*** – Cada microserviço é implementado no seu respetivo ambiente isolado por exemplo um microserviço por máquina física, máquina virtual ou *container*. Esta abordagem é mais dispendiosa, em comparação, porque faz uso de ambientes dedicados e à separação uniforme dos recursos alocados para um sistema.

3.6.3 Padrões de armazenamento de dados

Estes padrões lidam com a forma como os dados são armazenados, geridos e utilizados pelos diferentes microserviços. Sendo programas independentes é necessário que cada um consiga fazer a gestão dos respetivos dados sem afetar a performance dos outros microserviços.

- **Padrão Database-per-service** – Cada microserviço usufrui da sua base de dados dedicada de forma a não interagir com os dados dos restantes. Isto permite o desenvolvimento e escalamento de forma independente e é a abordagem ideal quando o isolamento dos serviços é crucial. No entando, conforme o número de serviços, aumenta tanto a complexidade como o custo de manutenção do sistema.
- **Padrão Database Cluster** – Todos os dados do sistema são armazenados num único servidor de base de dados sendo que cada microserviço apenas tem acesso a uma determinada porção deste (*schema*). Esta abordagem não só simplifica e baixa o custo de manutenção do sistema como também permite melhor escalabilidade e desempenho uma vez que pode ser escalada em vários nós. No entando há um risco maior de falha sendo que a componente de base de dados torna-se um recurso partilhado.
- **Padrão Shared Database** – Múltiplos microserviços partilham a mesma instancia de base de dados. É o padrão mais simples e acessível sendo que apenas é necessário existir uma instancia de base de dados para todos os microserviços. Vai um pouco contra o conceito de independência dos microserviços e torna mais difícil o processo de escalamento de recursos. à semenlhança do *cluster* de base de dados, também constitui um ponto único de falha.

3.7 Padrões de Design de Software

3.7.1 Padrão de Pipeline

Este padrão funciona através de uma série de operadores individuais que trabalham de forma sequencial sendo que o output de um operador será o input do próximo e cada um está completamente dependente do sucesso do operador anterior imediato. Caso algum dos operadores falhe numa simples verificação o processo é parado e um erro ou mensagem tratado será enviado de volta para o remetente ou desencadeante do processo (Vermeulen et al., 1995). Desta forma é possível promover a modularidade dos operadores, permitindo que estes sejam desenvolvidos de forma independente e que sejam reutilizados.

Este padrão é extremamente similar ao padrão de cadeia de responsabilidade (Gamma et al., 1994) porém está focado na transformação e tratamento dos dados do que na passagem e atribuição de responsabilidade em cadeia. Este comportamento pode ser observado na Figura 12.

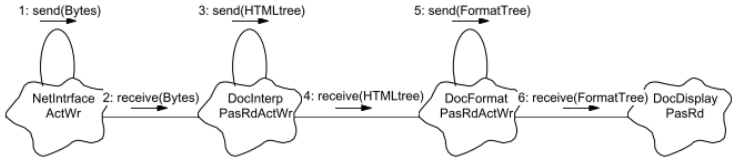

Figura 12 - Exemplo de uma pipeline de um browser (adaptado de (Vermeulen et al., 1995))

3.7.2 Padrão de Estratégia

O padrão de estratégia define uma família de algoritmos e encapsula cada um deles como uma classe separada permitindo que sejam intercambiáveis entre si dentro de um sistema. Isto permite que um determinado algoritmo seja selecionado durante o tempo de execução de um programa o que permite promover a flexibilidade e desassociar o código que constitui um algoritmo da sua própria implementação.

Segundo este padrão cada algoritmo é representado por um objeto de estratégia que serão executados através de uma interface comum, o que permite a modularidade e extensibilidade destes algoritmos. Sendo assim é possível alterar o comportamento de um sistema sem alterar a sua estrutura subjacente. Este comportamento encontra-se representado na Figura 13.

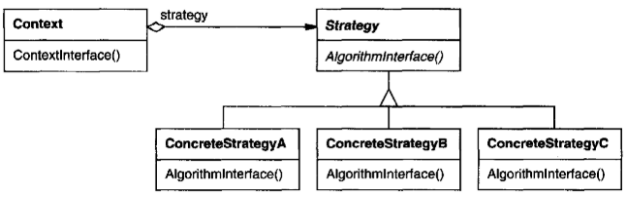

Figura 13 - Exemplo do padrão de Estratégia (adaptado de (Gamma et al., 1994))

3.7.3 Padrão MVC

Um dos padrões mais bem conhecidos e populares para a estruturação e desenvolvimento de software é o MVC (Model View Controller) que como o nome indica consiste em três componentes lógicas separadas Model, View e Controller que tornam o desenvolvimento, manutenção e a extensão da aplicação mais fáceis (Majeed & Rauf, 2018):

- **Model**: Camada responsável por armazenar e gerir os dados da aplicação ou serviço;
- **View**: Camada responsável pela apresentação dos dados ao utilizador;
- **Controller**: Camada responsável pelo tratamento e execução das interações com o utilizador e pela atualização das outras camadas em conformidade.

3.8 Encriptação de Dados

Na atividade de segurança no desenvolvimento e manutenção de sistemas de informação a questão ou preocupação não está em se vai haver alguma falha de segurança, mas sim quando pelo que se parte sempre do pressuposto que os piores cenários acabarão por acontecer. Por esta razão medidas proativas têm de ser tomadas sendo que uma das principais preocupações num sistema está nas informações sensíveis armazenadas em bases de dados. No caso de uma base de dados possuir informações sensíveis como credenciais de forma legível ou facilmente compreensíveis ou deduzíveis, um acesso não autorizado por qualquer meio seria catastrófico para todo o sistema bem como os seus utilizadores. Sendo assim a forma e métodos de armazenamento de informação altamente sensível são de extrema importância.

A criptografia existe, precisamente, para este fim sendo que consiste na prática da conversão de dados e informação num formato ilegível com o auxílio de algoritmos de encriptação cuidadosamente desenvolvidos e estudados (Sriramya & Karthika, 2015). Sendo assim a encriptação de informação sensível impede que esta seja comprometida mesmo que o acesso à base de dados esteja completamente acessível a um atacante. Para além do mencionado a criptografia também permite a assinatura digital dos dados de forma a garantir a autenticidade dos mesmo e prevenir a sua manipulação. Com base nestes fatores é possível dizer que a aplicação de estratégias e métodos criptográficos para encriptação e armazenamento de informação altamente sensível garante uma segurança robusta da mesma.

Foram comparados três dos maiores, melhores e mais conhecidos nomes em algoritmos de derivação de chaves baseadas em credenciais ou senhas **PBKDF2, Bcrypt** e **Scrypt**:

- **PBKDF2**: Algoritmo concebido para ser lento e computacionalmente dispendioso de forma a dificultar ataques de força bruta. Utiliza uma função pseudoaleatória (PRF) para gerar uma chave a partir da informação a encriptar e um sal. Esta função é aplicada numeras vezes o que aumenta, ao mesmo tempo, tanto o seu tempo de execução, mas também o tempo e recursos necessários para a sua decifração;
- **Bcrypt**: Algoritmo concebido para ser mas intensivo e exigente no cálculo da encriptação de dados. Este algortitmo é deliberadamente dispendioso para que qualquer tipo de ataque ou tentativa desonesta de decifração se torne mais complexa e demorada. O Bcrypt utiliza uma PRF baseada na cifra *BlowFish* (Ertaul et al., 2016; Sriramya & Karthika, 2015) e um sal de forma a gerar uma chave encriptada. Esta PRF, à semelhança do algoritmo PBKDF2, é aplicada várias vezes para a geração de uma chave, porém de uma forma diferente do PBKDF2;
- **Scrypt**: É o algoritmo mais intensivo em termos de memória necessária para a sua execução tornando-o o algoritmo mais difícil de decifrar. Como tal também é o algoritmo de implementação mais difícil e desempenho mais lento. O Scrypt utiliza uma função sequencial, particularmente computacionalmente exigente, para gerar uma chave a partir da informação a encriptar e um sal. Esta função foi concebida para ser muito exigente e difícil de paralelizar o que torna o algoritmo resistente a ataques de alta performance como ataques baseados Unidades de Processamento Gráfico (ou GPUs).

De forma geral, todos estes algoritmos utilizam um sal e funções de encriptação com níveis diferentes de exigência computacional de forma a converter algum pedaço de informação numa chave encriptada segura pelo que são todos considerados seguros. Quando comparados em ambientes idênticos o algoritmo Scrypt é o que apresenta maior capacidade de segurança visto que é o mais computacionalmente exigente e resistente contra-ataques baseados em CPU ou GPU. O algoritmo PBKDF2 é o mais rápido dos três, sendo que não é tao intensivo e o que utiliza algoritmos mais simples, porém igualmente seguros. A diferença de rapidez ou performance do PBKDF2 e Bcrypt é praticamente irrelevante, sendo que o Bcrypt ainda que mais lento que o PBKDF2, continua a ser notavelmente rápido enquanto é mais robusto. Sendo assim o Bcrypt consiste num compromisso excelente entre desempenho e segurança das suas chaves que o torna ideal e a recomendação da comunidade para a maioria dos casos (Ertaul et al., 2016). Algoritmos como o Scrypt deverão ser utilizados em circunstâncias excecionais onde é necessário o nível máximo de segurança possível mesmo que venha com um custo de desempenho como é o caso de *blockchain* ou cripto moedas.

3.9 JSON Web Tokens

JWT trata-se uma norma aberta que define um método moderno, compacto e autónomo de transmitir informação ou dados de forma segura entre sistemas ou entidades no formato JSON. A utilização de JWTs é considerada segura e as informações neles contidas são consideradas fiáveis uma vez que são assinadas digitalmente e todo o processo de validação roda a volta destas assinaturas. Devido a esta premissa JWTs são normalmente utilizados para controlo de autenticação e autorização em aplicações Web (normalmente baseadas em Javascript e Node.js devido à sua afinidade pelo formato JSON). O fluxo de utilização e implementação normal de um JWT para fins de autenticação passa pelo pedido inicial de autenticação ao servidor que, se válido, irá gerar um *token* JWT e enviá-lo de volta para o emissor do pedido. Este *token* poderá, então, ser armazenado e enviado juntamente com todos os pedidos posteriores que necessitem de autorização. Para estes pedidos o servidor irá validar o *token* JWT apegado e, se a validação for bem-sucedida, retornar os recursos requisitados (Jones, 2015; Jones et al., 2015; Sheffer et al., 2020b). Dentro destes JWTs geralmente são passadas informações essenciais para a identificação de um dado utilizador como o seu valor de identificação na base de dados. Este comportamento pode ser observado na representação abaixo (Figura 14).

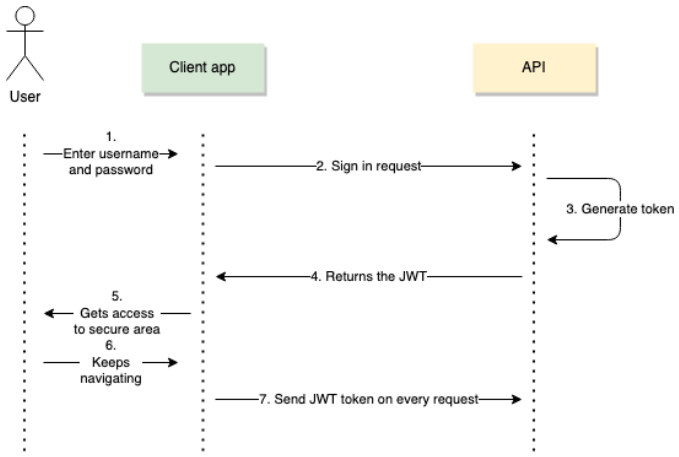

Figura 14 - Fluxo de autenticação por JWT

A utilização de JWTs é bastante atraente, especialmente quando emparelhada com uma Rest API, por várias razões, nomeadamente:

- **Compacto**: JWTs são codificados com a utilização do algoritmo ***base64url*** (Jones, 2015), o que os torna extremamente compactados e eficientes em termos de utilização de espaço. Isto torna-os ideais para uso em aplicações Web onde preocupações como a banda larga são muito relevantes;

- **Autónomo**: Um JWT possui toda a informação necessária para a sua identificação e verificação não sendo necessário qualquer tipo de chamadas, funcionalidades ou serviços adicionais. Isto torna-os muito eficientes e escaláveis o que, mais uma vez, é perfeito para uso num microserviço ou REST API (Jones, 2015; Jones et al., 2015; Sheffer et al., 2020b);
- **Seguro**: JWTs podem ser assinados digitalmente utilizando um vasto leque de algoritmos de criptografia como RSA ou HMAC (Jones, 2015) que os torna extremamente seguros e previne tentativas de manipulação dos dados neles contidos.

Os JWTs possuem uma estrutura bem definida de forma a garantir a integridade e autenticidade dos dados que carregam e são compostos por três componentes:

- **Cabeçalho/Header**: O cabeçalho ou *header* de um JWT contém a informação acerca do JWT em si como o seu tipo e que algoritmo foi utilizado para a assinatura digital como RSA ou HMAC (Jones, 2015);
- **Carga/Conteúdo/Payload**: A carga ou *payload* de um JWT contém, precisamente, os dados ou informação que ele carrega como por exemplo o identificador de um utilizador;
- **Assinatura**: A assinatura é a parte final do JWT e é utilizada para verificar ou validar a autenticidade e integridade do JWT e por consequência confirmar que o remetente deste JWT de facto é quem diz ser e que o seu conteúdo ou carga não foram manipulados ou alterados.

Esta estrutura é observável na Figura 15 representada abaixo.

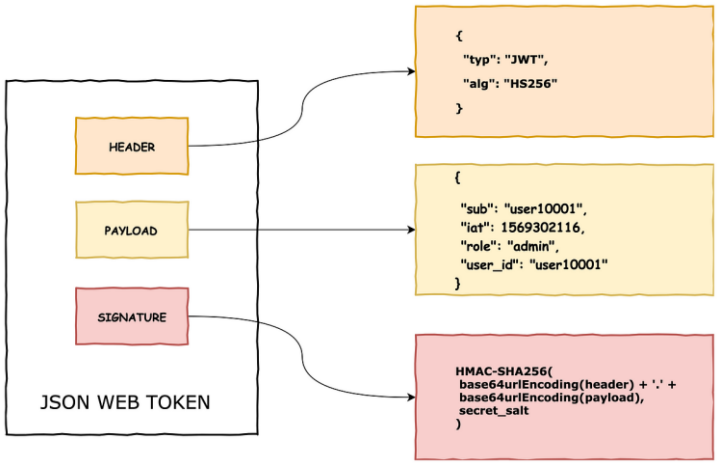

Figura 15 - Estrutura JSON Web Token

3.10 Expressões Regulares

Expressões Regulares (regex) são uma ferramenta muito poderosa e versátil para a correspondência de padrões e manipulação de cadeias de caracteres. Através de regras específicas é possível definir padrões de pesquisa que podem ser utilizados para corresponder, extrair ou substituir partes de uma cadeia de caracteres. O regex é altamente flexível permitindo efetuar pesquisas complexas e precisas através da escrita de expressões compostas por literais, metacaracteres e operadores. O motor Regex processa uma cadeia de caracteres para encontrar correspondências com base nos padrões definidos permitindo efetuar tarefas eficientes de processamento de caracteres (Ficara et al., 2008; Thompson, 1968).
Algumas das funcionalidades mais poderosas deste motor são:

- Pesquisa e manipulação de texto;
- Correspondências e padrões de texto;
- Extração de dados de texto;
- Validação de texto;
- Geração de texto.

As expressões regex são extremamente potentes, fáceis de escrever/criar e capazes de cobrir, praticamente, qualquer caso de uso, porém é sempre necessário haver um motor que seja capaz de as executar.

3.11 React.js

O React é uma *framework* de desenvolvimento de interfaces de utilizador de código aberto baseada na linguagem de programação *javascript* e, como a maioria dos projetos de código aberto, é mantida por uma grande empresa o Meta (outrora conhecido como Facebook) e por uma grande comunidade de desenvolvedores individuais. Trata-se de uma *framework* declarativa, eficiente e flexível o que a torna uma excelente opção para a tarefa de desenvolvimento de aplicações Web modernas (Rawat & Mahajan, 2020; Rocha et al., 2020).
As funcionalidades e características React pode ser decomposto em três grandes conceitos que são a base do seu sucesso, popularidade e eficiência:

- **Programação declarativa**: Apenas é necessário definir ou descrever de forma exata como a interface se deve comportar. O React está responsável por toda a lógica de atribuição e atualização da DOM de forma automática e dinâmica proporcionando um grande poder de atualização de estados de forma abstraída a qualquer desenvolvedor. (Rocha et al., 2020). A definição de código declarativo também permite que a legibilidade do código seja muito maior;

- **DOM Virtual**: O React utiliza uma DOM virtual para atualizar a interface de utilizador de uma forma muito eficiente. Esta DOM virtual, como o próprio nome indica, é uma representação leve da DOM real sendo que quando o estado da aplicação é alterado o React procede para a alteração da DOM virtual e apenas depois atualiza, conforme a DOM virtual e o necessário, a DOM real. Sendo assim a DOM real é atualizada em incrementos com base na DOM virtual sempre que necessária de uma forma muito eficiente e controlada. Isto torna as aplicações e interfaces desenvolvidas com React extremamente rápidas e eficientes (Rawat & Mahajan, 2020; Rocha et al., 2020);
- **Arquitetura baseada em componentes**: Uma interface desenvolvida com React pode ser decomposta em diversos componentes reutilizáveis, independentes e autónomos. Isto permite decompor uma funcionalidade complexa em componentes mais pequenos, também subdividíeis se necessário, que podem vir a ser reutilizados noutras funcionalidades. Esta arquitetura e padrão de desenvolvimento tem um impacto muito grande na qualidade, eficiência e simplicidade do código desenvolvido (Rawat & Mahajan, 2020; Rocha et al., 2020).

Tendo estas características em mente o React possui características muito apelativas sendo que, segundo a pesquisa bibliográfica, existem sérios benefícios na sua utilização para o desenvolvimento de uma aplicação web moderna:

- **Performance**: A DOM virtual do React é muito eficiente na atualização seletiva e eficiente da DOM real (interface de utilizador) sendo que as aplicações são extremamente rápidas (Rawat & Mahajan, 2020; Rocha et al., 2020);
- **Escalabilidade**: Aplicações desenvolvidas em React são de escalabilidade fácil e potente pelo que são desenvolvidas com base em componentes. Isto significa que as novas funcionalidades são muito fáceis e podem ser desenvolvidas de forma autónoma sem tocar no resto do código, Também as alterações de funcionalidades tornam-se muito fáceis graças à reutilizabilidade dos próprios componentes (Rawat & Mahajan, 2020; Rocha et al., 2020);
- **Manutenção**: Código desenvolvido para aplicações React é de fácil leitura e manutenção visto que os componentes são desenvolvidos de forma declarativa sendo que se torna mais fácil de identificar erros ou adicionar novas funcionalidades (Rawat & Mahajan, 2020; Rocha et al., 2020);
- **Experiencia de desenvolvimento**: O React possui muitas funcionalidades que tornam a vida de um desenvolvedor mais fácil como a utilização de JSX que permite a escrita direta de HTML em lógica Javascript, ou *hot-reloading* que consiste na recompilação do código de cada vez que

o desenvolvedor guarda as suas mudanças de forma a que estas sejam imediatamente visíveis no browser. A comunidade de desenvolvimento também presta muito apoio na forma de ajuda em fóruns ou através de ferramentas e melhorias contínuas à *framework* (Rawat & Mahajan, 2020; Rocha et al., 2020).

Todas as características do React tornam-no numa excelente escolha para o desenvolvimento de qualquer aplicação Web e interface de utilização sendo que a surpresa não é grande quando o React é a *framwork* de desenvolvimento Web mais popular e utilizada pela comunidade desenvolvedora (Rocha et al., 2020). Porém, para além de tudo o que o torna bom, o React também dispõe de alguns desafios cuja consideração será necessária para uma boa decisão de *framework* de desenvolvimento:

- **Curva de Aprendizagem**: É bastante aceitável na comunidade que React pode ser difícil de aprender especialmente para desenvolvedores sem muita experiência devido ás suas nuances e particularidades que não só tornam o desenvolvimento mais simples mas também mais complexo e confuso para aprendizagem;
- **Complexidade de Ecossistema**: O ecossistema React é muito vasto e complexo pelo que conta com uma grande variedade de ferramentas e bibliotecas para programação distintas e com diferentes âmbitos. Isto pode tornar a escolha correta das ferramentas uma tarefa árdua e incerta pelo que muitas vezes um desenvolvedor encontra-se "preso" sem conseguir começar ou prosseguir com o seu projeto porque não consegue escolher uma. O acompanhamento das atualizações e novos desenvolvimentos no ecossistema React também prova ser uma tarefa muito difícil;
- **Otimização**: Embora o React seja muito rápido por si é, também, importante otimizar o código e lógica desenvolvidos pelo que é muito fácil obter uma aplicação mais lenta pelo simples facto de uma programação não eficiente. Este aspeto é particularmente impactante em desenvolvedores não familiarizados com técnicas de otimização de desempenho na execução de algoritmos;
- **Gestão de estado**: Para controlar a DOM virtual o React disponibiliza bastantes APIs para gestão do estado da aplicação. É incrivelmente fácil fazer uma gestão fraca e não otimizada dos estados da aplicação e dos seus diversos componentes pelo que isto consiste num grande desafio a desenvolvedores não experientes. Uma má gestão de estado resultará numa aplicação com fraca otimização e não muito flexível. Este ponto é, ainda, agravado aquando da utilização de uma biblioteca de gestão de estado complexa como Redux (Matthias, 2016).

3.12 Servidores HTTP

Servidores HTTP são programas de *software* que facilitam a solicitação de recursos ou ficheiros estáticos via pedidos HTTP com grande conveniência, facilidade e conveniência. Estes servidores são programas que escutam e aguardam por pedidos HTTP vindos de diversos clientes HTTP, como navegadores Web ou outros clientes, identificam e retornam o recurso solicitado do seu sistema e ambiente de ficheiros ou bases de dados de volta ao cliente (Guolang et al., 2018). Os servidores HTTP são essenciais e constituem a base para a *World Wide Web* uma vez que são eles que fornecem os recursos Web aos navegadores dos utilizadores em todo o mundo.

Durante a pesquisa bibliográfica efetuada foram considerados e comparados os maiores e mais populares nomes em termos de servidores HTTP como **Apache**, **Nginx**, **LiteSpeed** e **Microsoft IIS**, representados na Figura 16, que foram testados em ambientes de velocidade e arquitetura idênticos.

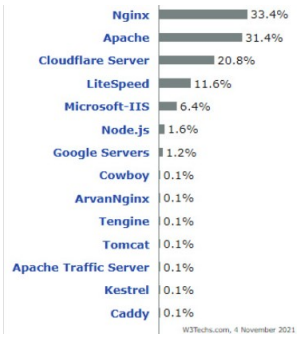

Figura 16 - Servidores HTTP listados por popularidade (adaptado de (Kithulwatta et al., 2022))

Todos estes servidores possuem um ótimo desempenho pelo que são os principais atores no setor, porém o **Nginx** destaca-se em termos de rapidez de serviço e deveria ser o servidor de escolha aquando da hospedagem e serviço de ficheiros de pequena escala (Guolang et al., 2018). Foram realizados uma série de testes com ambientes de concorrência e ficheiros com tamanhos diferentes como é indicativo na Figura 17 e, consistentemente, tanto o Nginx como LightSpeed demonstram possuir uma maior rapidez e eficiência no que toca ao serviço de ficheiros de pequena dimensão.

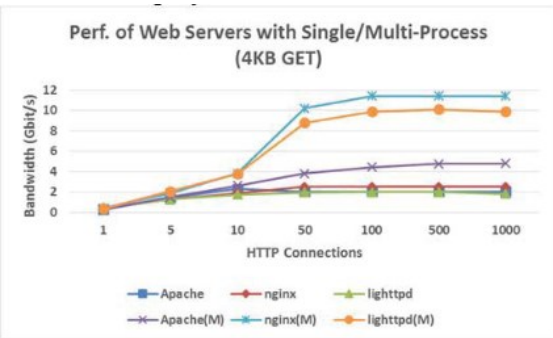

Figura 17 - Teste de Performance de ficheiros de 1KB em servidores diferentes (adaptado de (Guolang et al., 2018))

A pesquisa revela que qualquer um destes servidores é perfeitamente capaz e bem-adaptado para servir qualquer aplicação Web de pequena-média dimensão. Também revela que o paradigma atual mais popular e eficiente está na "*conteinerização*" e virtualização destes servidores sendo que tanto Apache como Nginx foram testados e foi concluído que ambas as abordagens são extremamente eficazes e estáveis pelo que apresentam ótimos valores de rapidez e desempenho como 100% de disponibilidade sob testes de stress (Kithulwatta et al., 2022). Sendo assim é possível concluir que qualquer um dos servidores anteriormente mencionados é uma ótima escolha e encontra-se muito bem preparado para o objetivo de hospedagem de aplicações centradas em rede a nível empresarial (Kithulwatta et al., 2022).

3.12.1 Nginx

O Nginx é um servidor HTTP de elevado desempenho com uma enorme popularidade devido à sua estabilidade e conjunto vasto de funcionalidades, à sua configuração simples, fácil implementação e manutenção e também ao seu baixo consumo de recursos (Soni, 2016).

A grande esmagadora popularidade do servidor Nginx provém do seu leque extenso de características e funcionalidades que, eficiente e elegantemente, proporcionam a melhor experiência de integração, implementação e distribuição de recursos estáticos via pedidos HTTP utilizando os melhore e mais recentes paradigmas e práticas de desenvolvimento. O Nginx foi desenvolvido para abordar o problema C10K que consiste na dificuldade de um servidor conseguir lidar dez mil conexões concorrentes (Soni, 2016). O Nginx resolve este problema através da sua arquitetura orientada a eventos o que permite o tratamento e processamento de uma grande quantidade pedidos ou conexões simultâneas utilizando o mínimo de recursos possível (Soni, 2016).

De uma forma resumida o Nginx engloba um conjunto de características que o tornam muito apelativo como opção para serviço de recursos Web:

- **Desempenho**: O Nginx é conhecido pelo seu elevado desempenho pelo que consegue processar mais de dez mil conexões simultâneas tornando-o ideal para soluções desenhadas para lidar com elevado nível de tráfego (Soni, 2016);
- **Eficiência**: O Nginx serve o conteúdo estático guardado no seu sistema de ficheiros de forma extremamente eficiente utilizando a funcionalidade de *cache* sendo que ficheiros frequentes ficam guardados em memória temporária não sendo necessária a sua busca ao local de origem no servidor sempre que são solicitados (Soni, 2016);
- **Versatilidade**: Para além de um servidor HTTP o Nginx também pode operar como um *proxy* reverso, um balanceador/gestor de carga ou um sistema de *cache* HTTP (Soni, 2016);
- **Facilidade de Uso**: A configuração do Nginx é extremamente simples e sua utilização de recurso é muito baixa quando comparados com os restantes servidores HTTP (Soni, 2016).

3.13 Controlo de Versão

Um dos conceitos mais importantes no que toca ao desenvolvimento de software é a organização do trabalho realizado. O controlo de versão trata-se do processo de acompanhamento e gestão de todas as alterações e atualizações dos ficheiros pertencentes a um projeto de software ao longo do tempo (Zolkifli et al., 2018).

A utilização de um sistema de controlo de versões (VCS) traz muitos benefícios e funcionalidades úteis à atividade de desenvolvimento de software:

- **Qualidade**: Um sistema de controlo de versões permite, de uma forma muito fácil, a gestão, acompanhamento e atualização de qualquer ficheiro na fonte de código sendo trivial efetuar alterações ou reverter alterações erróneas melhorando assim a qualidade geral do código desenvolvido;
- **Colaboração**: Estes sistemas permitem e promovem a colaboração de vários desenvolvedores no mesmo projeto sendo que se tornam ferramentas essências para a tarefa de desenvolvimento de software que, normalmente, dispõe de equipas inteiras para realizar o trabalho;
- **Manutenção**: Facilitam a manutenção do software desenvolvido/ em desenvolvimento pelo que permite o acompanhamento transparente e completo de qualquer ficheiro numa base/fonte de código. Isto facilita, não só, a identificação de problemas/*bugs*, a sua correção e implementação fácil.

As grandes vantagens dos sistemas de controlo de versões provêm das suas muitas funcionalidades que, de forma fácil e conveniente, ajudam a melhorar a experiência de desenvolvimento:

- **Histórico de Versão**: Os VCS permitem o acompanhamento de todas as mudanças na base de ficheiros que compõe um projeto bem como quem as fez e quando. Isto é extremamente útil na procura de problemas (ou da sua fonte) e, caso necessário, a reversão de alguma mudança problemática;
- ***Branching*:** Uma das principais funcionalidades de um VCS é a criação e gestão de diferentes *branches* que constituem versões diferentes do mesmo software com funcionalidades ou alterações distintas relativas à própria *branch*. Isto permite o desenvolvimento de novas funcionalidades ou a correção de problemas sem nunca tocar ou afetar a versão principal do projeto. Estas mudanças, depois de confirmadas poderão ser integradas de forma segura e com confiança à *branch* ou versão principal no projeto;
- **Colaboração:** Outra funcionalidade fundamental dos VCS é a possibilidade de colaboração de múltiplos desenvolvedores no mesmo projeto de forma organizada e transparente de modo a não haver conflitos.

Existem duas grandes categorias de VCS que constituem todas as soluções no ecossistema de desenvolvimento de software, centralizados e distribuídos:

- **VCS Centralizado**: Este tipo de VCS guarda todos os ficheiros relativos a um projeto num repositório central pelo que quando um desenvolvedor pretende fazer uma alteração a um ficheiro necessita de realizar o *check-out* do ficheiro, fazer a alteração e fazer o *check-in* do ficheiro de volta para o repositório;
- **VCS Distribuído**: Este tipo de VCS guarda uma cópia completa de um repositório de um projeto na máquina de todos os desenvolvedores envolvidos, pelo que permite o desenvolvimento de qualquer ficheiro de forma autónoma sendo que o desenvolvedor apenas tem de submeter as alterações feitas para o repositório central.

3.13.1 Git

O Git trata-se de um VCS Distribuído que se tem tornado cada vez mais relevante e uma parte integral do desenvolvimento de software moderno. Desenvolvido e lançado em 2005 por Linus Torvalds, o Git fornece uma plataforma muito robusta e flexível de forma a facilitar e trivializar o processo de

acompanhamento de alterações no código-fonte, colaboração entre programadores e a garantia de integridade do histórico de desenvolvimento de um projeto (Chacon & Straub, 2023).

A utilização do Git no desenvolvimento de todo o software para o sistema de interoperabilidade utiliza as funcionalidades e características da plataforma descritas com base no livro ***Pro Git*** por Scott Chacon e Ben Straub:

- ***Branching* e *Merging*** – Um desenvolvedor, ao utilizar o Git, pode criar *branches* para trabalhar em funcionalidades ou correções específicas de forma separada sem nunca afetar a base de código principal. A funcionalidade de *merge* das mudanças e alterações nas diferentes *branches* permite o desenvolvimento paralelo e assegura a sua integração ininterrupta.
- ***Snapshots*** – Ao contrário de outras soluções para controlo de versõe, o Git captura *snapshots* de um projeto completo em pontos específicos no desenvolvimento. Cada *commit* representa um *snapshot* completo tornando a navegação pelo histórico de um projeto rápida e eficiente.
- ***Pushs* e *Pulls*** – No ambiente distribuído proporcionado pelo Git, os diferentes contribuidores para um mesmo projeto podem fazer *push* das suas mudanças mais recentes bem como o *pull* dessas mesmas mudanças de forma a assegurar a sincronização bem estruturada em todo o processo de desenvolvimento.
- **Gestão de *branches*** – As funcionalidades de *branching* fornecidas pelo Git permitem o isolamento para as tarefas de desenvolvimento bem como a sua respetiva integração sem problemas;
- **Eficiência e Desempenho** – O Git foi desenhado e desenvolvido com eficiência e rapidez em mente sendo que as alterações locais podem ser efetuadas e a natureza distribuída minimiza a necessidade de acesso constante à rede hospedeira. A arquitetura interna do Git assegura um controlo de versão responsivo e altamente escalável para projetos independentemente do seu âmbito ou dimensão;
- **Ambientes de Desenvolvimento Integrados** – A enorme adoção do Git nos variados ambientes de desenvolvimento integrados (IDEs) provém da sua popularidade e presença estabelecida da *stack* de desenvolvimento moderno sendo que as IDEs mais populares como o Visual Studio Code, Eclipse ou IntelliJ IDEA possuem suporte para integração de Git por predefinição. Da mesma forma que o Git está presente em praticamente todas as IDEs mais conhecidos, também se pode contar com a sua presença e integração em qualquer outra ferramenta de apoio ao desenvolvimento de renome;

3.14 Estratégias de Branching

A funcionalidade de *branching* é uma das funcionalidades mais importantes de um sistema de controlo de versões sendo que permite a criação de cópias isoladas do código-fonte onde podem ser realizadas alterações autónomas sem afetar o produto principal. Para complementar o *branching* existe o *merging* que permite a integração ininterrupta das mudanças presentes em diversas *branches* numa só sem conflitos (Emad et al., 2012; Phillips et al., 2011). Através destes dois conceitos básicos nascem as estratégias de *branching* onde, para um projeto, deve ser decidido o fluxo que as alterações devem seguir para serem integradas de forma controlada, rápida e eficiente. Devido à natureza personalizável da implementação destas funcionalidades as estratégias adotadas no desenvolvimento de software tendem a ser adaptadas aos requisitos de cada projeto não havendo um padrão propriamente dito para este fim. Por mais diferentes que as estratégias de *branching* possam ser de projeto para projeto, geralmente falando, existem conceitos e padrões de ouro que se verificam em, praticamente, todos os projetos geridos por VCSs:

- ***Branches* de longo prazo** - Estas *branches* são consideradas "intocáveis" pelo que nenhum desenvolvedor pode realizar alterações nelas de forma direta e sem uma revisão dos seus pares, colegas ou *code owners*. Como convenção estas *branches* constituem versões sensíveis do código-fonte que carecem de controlo e cuidado completos e estão sujeitas a um acompanhamento e regramento rigoroso. Estas *branches* tipicamente são as de lançamento de versão ou de desenvolvimento;
- ***Branches* de curto prazo –** Como o próprio nome indica as *branches* de curto prazo são cópias isoladas do código-fonte destinadas ao desenvolvimento de uma funcionalidade nova ou uma correção pelo que todas as mudanças feitas serão, posteriormente, integradas nas *branches* de longo prazo através de *Pull Requests* devidamente estruturados e regrados. Para a realização destas integrações as mudanças estão, geralmente, sujeitas a revisão de pares pelo que nenhuma poderá seguir para a uma *branch* de longo prazo sem a aprovação de colaboradores autorizados. Após a integração das mudanças as *branches* de curto prazo deverão ser apagadas.
- **Gestão de permissões em *branching* –** Todas as *branches* de longo prazo necessitam de ser devidamente configuradas de forma a restringir a forma e por quem podem ser alteradas e iteradas. Grupos de revisores devem ser criados com a confiança de que estes ficarão

responsáveis pela aprovação de novas mudanças bem como o número de aprovações necessário para a integração das mesmas.

3.15 SemVer

A SemVer é uma convenção que dita que todo software, nomeadamente bibliotecas e dependências, deve ser devidamente dotado de uma versão padronizada para comunicar que tipo de mudanças foram realizadas desde a última versão. Sendo assim o SemVer utiliza um formato de versão composto por três partes pelo que cada uma transmite um significado específico:

- **Major (principal**) – Trata-se do primeiro número (número referente à posição esquerda) do formato de versão e indica uma mudança significativa no software sendo que da versão x.0.0 para a x+1.0.0 houve uma, ou várias, alterações que são incompatíveis com versões antigas. A natureza sensível de uma versão *major* faz com que o lançamento da mesma careça da devida documentação para informar os utilizadores o que mudou exatamente;
- **Minor (secundária)** – Uma versão secundária acontece quando a nova iteração do código de produção apresenta novas funcionalidades que vêm incrementar o leque de utilidade do software. Por exemplo o incremento de uma versão 1.x.0 para 1.x+1.0 significa que, à partida, existem novas funcionalidades no software. O lançamento de uma versão secundária já é algo mais frequente, sempre que uma funcionalidade ou conjunto de funcionalidades novo está pronta para entrar em produção, e também carece de uma documentação e lista das funcionalidades novas acrescentadas;
- **Patch (correção)** – A versão de correção é lançada sempre que acontece uma correção de um erro ou *bug* sendo que assim que esta está validada no ambiente de teste deve ser lançada uma versão de correção de produção com o arranjo. Por exemplo o incremento de uma versão 1.1.x para 1.1.x+1 significa que a nova versão corrigiu um qualquer erro anteriormente descoberto. Esta é, de longe, o tipo de versão mais comum sendo que quanto maiores os projetos mais correções irão necessitar. O lançamento de uma versão de correção necessita da devida documentação do que foi corrigido bem como a lista de alterações efetuadas.

A SemVer estabeleceu um padrão excelente para o versionamento de software sendo que transmite imensa informação importante de forma clara com o mínimo de complicação. A popularidade da convenção SemVer torna-a algo muito enraizado nas ferramentas de desenvolvimento de software

moderno sendo que existe muita capacidade de integração e interoperabilidade ininterrupta com outros sistemas (Decan & Mens, 2019; Raemaekers et al., 2017).

3.16 Continuous Integration/Continuous Deployment

A automação é uma parte integral no desenvolvimento de software moderno sendo que permite garantir o cumprimento das melhores práticas estudadas de forma automática e consistente sendo que os desenvolvedores libertam tempo para se focarem nas tarefas de valor e não em tarefas monótonas e repetitivas. Uma boa implementação de automação num projeto de software iterativo permite que um desenvolvedor, equipa ou organização possa fornecer software de alta qualidade com maior frequência, qualidade e fiabilidade.

No núcleo da automação de processos em desenvolvimento de software estão três grandes conceitos ***Continuous Integration*** (integração contínua), ***Continuous Delivery*** (entrega contínua) e ***Continuous Deployment*** (lançamento contínuo):

- **Continuous Integration (CI)** – A integração contínua foca-se na automatização da integração de alterações e mudanças no código-fonte de vários colaboradores num repositório central garantindo, de forma automática, que o código está sempre num estado funcional e que mantém sempre o nível de qualidade (Shahin et al., 2017). Este objetivo é facilmente atingido através do desenvolvimento de testes automáticos que devem ser executados assim que uma nova mudança é submetida fornecendo *feedback* imediato acerca de possíveis erros, quebras ou inconsistências que sejam detetadas. Ao automatizar estas tarefas o CI permite a captura precoce de possíveis problemas e quebras no código bem como permite garantir a estabilidade e consistência do código desenvolvido;
- **Continuous Delivery (CD)** – A entrega contínua estende o CI na medida que automatiza os passos de construção e teste do software garantindo que este está pronto para ser lançado ou iterado com confiança. O foco do CD (Entrega Contínua), contudo, não está na automação de testes, mas sim na automação da integração das mudanças num ambiente de testes à medida que elas vão sendo criadas permitindo assim ter uma visão mais completa das ramificações das mesmas (Shahin et al., 2017);
- **Continuous Deployment (CD)** – O lançamento contínuo estende o conceito de CD (Entrega contínua) na medida que, para além da execução automática de testes e construção do software, tem como principal foco a automatização da integração de todas as mudanças aprovadas e

testadas, que já passaram pelos processos anteriores, num ambiente de produção ou na entrega destas aos utilizadores o mais rapidamente possível sem qualquer intervenção humana. Na automação de processos de gestão e integração de software o lançamento tipicamente constitui a última fase pela qual uma mudança passa até chegar ao cliente pelo que o foco nos testes é crucial nos processos anteriores (Shahin et al., 2017).

3.17 Conventional Commits

O padrão estabelecido pelo *Conventional Commits* estabelece algumas diretivas para a melhor identificação, categorização e compreensão das mudanças realizadas num *commit* que passam pela especificação do tipo do mesmo, âmbito e descrição segmentada. É possível observar esta estrutura na seguinte Figura 18

```
<type>[optional scope]: <description>
[optional body]
[optional footer(s)]
```

Figura 18 - Estrutura de um *commit* padronizado (adaptado de (Huskey & Huskey, 2023))

Sendo assim cada mensagem de *commit* irá necessitar de ser facultada com as seguintes componentes:

- **Tipo** – Especifica o tipo de *commit*. Os tipos estão limitados à seguinte lista: *feat; fix; docs; style; refactor; perf; test; build; ci; chore; revert.* O significado destes tipos pode ser inferido pela sua simples designação autoexplicativa;
- **Âmbito** – Componente opcional que especifica e segrega de forma mais profunda as diferentes partes de um código-fonte;
- **Descrição** – Breve resumo ou sumário das alterações realizadas num dado *commit* escrito de forma clara e concisa utilizando verbos imperativos para descrever as ações tomadas;
- **Corpo** – Utilizado para transmitir informações mais detalhadas acerca das alterações realizadas como justificações, passos tomados para as mesmas ou quaisquer efeitos colaterais que elas tenham;
- **Rodapé** – Pode ser utilizado para referenciar quaisquer outras informações relevantes como por exemplo o link para o item criado que levou ás mudanças realizadas no *commit*.

3.18 Virtualização de Software

O livro "*Linux Containers and Virtualization: A Kernel Perspective*" explora os dois conceitos e técnicas mais populares e eficazes com base no nível de abstração no contexto de virtualização sendo que são baseados em *containers* ou em máquinas virtuais (Jain, 2020; Potdar et al., 2020) sendo que cada abordagem é mais correta ou é melhor aplicada em cenários específicos.

Num estudo realizado por Amit Potdar e Mohammed Moin Mulla publicado em 2020 (Potdar et al., 2020) o desempenho e aptidão de ambas as técnicas foi testado e avaliado sendo que na maioria dos casos:

- Containers são mais eficientes e possuem melhor performance que máquinas virtuais;
- Containers partilham o kernel do sistema operativo ao passo que as máquinas virtuais virtualizam um kernel próprio o que leva a uso mais baixo de recursos por parte dos containers;
- Containers não necessitam de um sistema operativo completo para operar o que resulta num uso mais baixo de recursos;
- Containers não necessitam de uma camada adicional de rede sendo que apresentam menos latência na comunicação;
- Containers apresentam tempos de arranque mais rápidos e eficientes porque não necessitam de arrancar um sistema operativo completo.

Estas conclusões sugerem que a virtualização de software é mais eficiente e simples se for implementada na forma de *containers*.

3.18.1 Docker

O Docker é uma tecnologia de virtualização leve que permite o acondicionamento de *software* às unidades autónomas designados de *containers*. Estes *containers* partilham o kernel do sistema operativo hospedeiro, mas apresentam o seu próprio sistema de ficheiros isolado, o que os torna muito mais leves como mais eficientes do que máquinas virtuais (Potdar et al., 2020; W. Wang, 2022). Outro grande ponto para a utilização do Docker está na portabilidade destes *containers* sendo que um *software* quando empacotado num *container* poderá se executado sem qualquer tipo de problemas em qualquer ambiente ou máquinas desde que esta possua uma instalação Docker.

Segundo Wei Wang o Docker apresenta várias vantagens e características apelativas para a aplicação de virtualização por *container* o que o tornam a tecnologia de escolha para muitos desenvolvedores e computação de nuvem:

- **Portabilidade** – *Containers* Docker podem ser executados em qualquer máquina desde que esta possua uma instalação Docker independentemente do tipo de máquina, sistema operativo

ou ambiente de execução. Isto torna muito fácil a implementação de aplicações e *software* em qualquer ambiente;

- **Isolação** – Os *containers* Docker são executados e mantidos de forma isolada tanto uns dos outros como da máquina hospedeira o que os torna mais seguros e fidedignos;
- **Reprodutibilidade** – *Containers* Docker são reproduzíveis o que significa que irão ser executados e correr sempre da mesma forma independentemente do ambiente atingindo um altíssimo nível de estabilidade e consistência na execução de software;
- **Escalabilidade** – Devido à natureza autónoma e isolada dos *containers* Docker, estes podem ser escalados facilmente de forma a melhor satisfazer as necessidades de um sistema.

O processo de acondicionamento de *software* através do Docker é relativamente simples pelo que necessita sempre de partir de uma imagem inicial, preparar o ambiente virtual para a execução do software e executá-lo, instruções estas que fazem parte de um ficheiro de configuração designado por *Dockerfile*. Através deste *Dockerfile* é possível gerar uma imagem virtualizada do *software* sendo que esta pode, então, ser executada em qualquer máquina ou ambiente com total confiança e controlo de acessibilidade na forma de um *container*. Estas noções e fluxo estão representados na Figura 19.

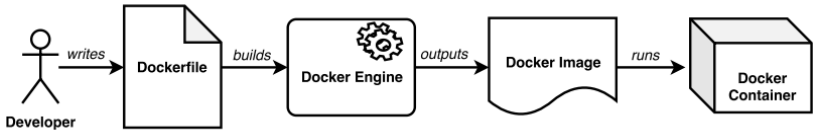

Figura 19 - Processo de acondicionamento de software através do Docker (adaptado de (Hasan Ibrahim et al., 2021))

3.18.2 Docker Compose

O *Docker Compose* é uma ferramenta IaC (Infrastructure as Code) que permite definir, de forma exata e detalhada, os *containers* que farão parte da aplicação ou sistema bem como arrancá-los todos com um simples comando (Hasan Ibrahim et al., 2021). Esta ferramenta permite tratar um sistema *multi-container* como um só abstraindo o utilizador das diversas partes móveis que o possam constituir.

A utilização do *Docker Compose* simplifica o processo de implementação e gestão de aplicações ou sistemas complexos, facilita a definição e gestão das relações entre os diversos *containers* e componentes de um sistema e fornece uma forma consistente de executar aplicações ou sistemas complexos em ambientes diferentes enquanto mantém todas as características, configurações e funcionalidades do Docker.

Num estudo realizado por Hasan Ibrahim foi averiguado que a grande maioria das aplicações ou sistemas compostos por múltiplos *containers* são geridos pelo *Docker Compose* sendo que é uma ferramenta extremamente popular com muito apoio e atenção pelo que é uma ótima escolha para orquestração de *containers* (Hasan Ibrahim et al., 2021) num sistema *multi-container*. Também existem outras ferramentas para o mesmo fim como o *Kubernetes* (Gupta & Hohn, 2019), porém tendem a ser um pouco elaboradas demais visto que têm como objetivo orquestrar projetos muito grandes estendendo-se por centenas de *containers*. Para sistemas de, relativamente, pequena escala, o *Docker Compose* é a escolha mais correta porque não acrescenta complexidade desnecessária à arquitetura tornando-a o mais simples possível enquanto cumpre todos os requisitos de um orquestrador.

Sendo assim, seguindo o fluxo de consumo das diferentes partes do sistema, este é construído ou instalado através da orquestração com o *Docker Compose* sendo que todos os serviços se encontram bem definidos e interligados num simples ficheiro designado por d*ocker-compose.yaml*. Este ficheiro apresenta as seguintes componentes:

- **Versão** – Especificação da versão do *Docker Compose*;
- **Serviços** – Definição e configuração de todos os *containers* que fazem parte do sistema. A configuração dos *containers* vai para além da simples configuração por variáveis de ambiente que os microserviços deste sistema necessitam, também engloba normas de comportamento como o reinício de um serviço em caso de erro, portas a expor, mapeamento de volumes e relações entre serviços como esperar que um serviço arranque para arrancar outro;
- **Volumes** – Diretorias reservadas para a persistência dos sistemas de ficheiros dos vários serviços

3.19 Testes Unitários

Qualquer *software* necessita de ser testado pelo que se torna muito perigoso quando esta prática não é cumprida ou os testes não são amplos ou adequadaos ou até mesmo quando não são realizados com frequência. Possuir um fraco ciclo ou fluxo de testes resulta num *software* igualmente pobre, inconsistente e repleto de erros.

O *Unit Test* ou teste unitário (UT) é uma abordagem de teste de *software* através do qual as diversas unidades individuais do código-fonte são testadas de forma a averiguar se estas apresentam um comportamento apto para utilização. O objetivo destes testes é verificar que cada unidade de código está

em perfeitas condições de funcionamento de forma isolada produzindo sempre os mesmos resultados esperados minimizando assim os fatores de erro (Marc, 2019; Martin, 2014; Moroz, 2019).

A prática de testes unitários é muito importante pelo que é uma ajuda indispensável no auxílio e identificação de erros numa fase inicial do ciclo de desenvolvimento de qualquer funcionalidade ou alteração do código-fonte. A correção de erros nesta fase inicial permite poupar imenso tempo, especialmente quando o código-fonte começar a atingir níveis de dimensão e complexidade consideráveis. Ao testar o código é muito mais provável que este se comporte como o esperado e que não parta esporadicamente ou quando sofre alterações facilitando a manutenção e aumentando a qualidade e consistência do mesmo.

A prática de testes unitários, quando aplicada, necessita de ser como uma barreira a todas as alterações que partirem um teste sendo que todos têm de passar para a mudança poder seguir. Algumas das vantagens à adoção de testes unitários num projeto são:

- **Deteção precoce de erros ou problemas** – Os UTs permitem a deteção de erros ou problemas bem como a sua correção no início do ciclo de desenvolvimento que é quando a sua correção é menos dispendiosa**;**
- **Melhoria na qualidade do *software*** – Melhoram a qualidade do *software* tornando-o mais robusto, fiável e fácil de manter***;***
- **Aumento da confiança do desenvolvedor** – Um desenvolvedor pode fazer quaisquer alterações que pretender com confiança focando-se apenas na mudança pelo que à partida nunca irá introduzir um erro grave sem que este seja apanhado nos testes unitários**;**
- **Redução de risco de regressões** – Ajudam a reduzir o risco de regressões pelo que o constante teste do *software* torna a introdução de alterações que quebrem funcionalidades já existentes muito pouco provável.

Os testes unitários devem ser executados na sua totalidade depois de alterações terem sido realizadas localmente pelo que o processo apenas pode seguir depois destes passarem. Caso os testes e a revisão de pares passem as alterações podem ser integradas no código-fonte definitivo.

Para manter a qualidade e consistência entre os testes foram definidas algumas regras para o seu desenvolvimento:

- **Isolamento** – Cada teste deve isolar uma e apenas uma única unidade de código-fonte das restantes. Esta prática ajuda a manter os testes focados e simples;

- **Comportamento** – Cada teste deve verificar um e apenas um comportamento esperado da unidade sob teste;
- **Testes pequenos e focados** – Cada teste deve focar numa unidade de código e num único comportamento;
- **Nomes descritivos** – O nome de cada teste unitário deve ser representativo do próprio teste sendo que deve ser autoexplicativo.

3.20 Testes E2E

Esta metodologia de testes baseia-se no conceito de testes de "caixa negra" (Jamil et al., 2017; Raiküla & Applications, 2023) onde o foco está no teste a partir da perspetiva externa do utilizador em vez da examinação e conhecimento interno do código o que permite ter uma noção mais completa do estado atual de um projeto uma vez que cenários reais de utilização são simulados. Esta abordagem permite a deteção de erros ou problemas mais subtis e difíceis de identificar por teste unitário oferecendo uma garantia e cobertura abrangente e compreensiva de todas as funcionalidades disponíveis bem como o seu comportamento aumentando a confiança no lançamento de novas versões.

Em sistemas e aplicações mais complexos estes testes tornam-se indispensáveis pelo que começam a existir demasiadas partes móveis e pontos de possível rutura que ficarão fora de controlo sem uma estratégia adequada. Devido a estas necessidades de garantia de qualidade e vantagens de implementação os testes E2E são, não só uma mais-valia no ciclo de vida de desenvolvimento, como também constituem um fortíssimo candidato a automação para facultação aos testes unitários (Raiküla & Applications, 2023).

Esta eficácia foi demonstrada por Karina Raiküla em *"*Implementation of Automated End-To-End Testing in Web Applications*"* onde são avaliados três casos de uso em que se verificou que a prática de automação de testes E2E desempenha um papel crucial para garantir a qualidade e fiabilidade de sistemas baseados em Web ajudando a identificar e corrigir defeitos no mesmo (Raiküla & Applications, 2023)

3.21Segurança Web

A segurança de uma solução de *software* é das características mais importantes que esta pode ter sendo que toda a informação e acesso fornecido por ela aos utilizadores são considerados sensíveis tornando-se muito importante sua proteção.

O foco na segurança de uma solução permite a proteção dos dados sensíveis da organização que faz uso dela como informações de clientes, financeiras e propriedade intelectual. A exposição deste tipo de dados a atacantes constituiria uma porta de entrada para quaisquer fins maliciosos que estes possam ter (S. Aljabri et al., 2023). *Cyber*-ataques constituem uma ameaça séria ao desenvolvimento e utilização de *software* porque podem perturbar ou interromper o fluxo de informação e trabalho diário de uma organização sendo que a proteção contra eles é uma das maiores prioridades de qualquer organização (S. Aljabri et al., 2023; Bach-Nutman, 2020a; Veres & Wilkinson, 2023; Wen & Katt, 2023).

A cibersegurança consiste na prática da proteção de sistemas, redes, aplicações e programas informáticos de ataques digitais que se destinam, tipicamente, a aceder, alterar ou destruir informações e recursos sensíveis de utilizadores ou organizações (S. Aljabri et al., 2023; Veres & Wilkinson, 2023; Wen & Katt, 2023).

Devido ao rápido crescimento, adoção e desenvolvimento de serviços e soluções Web, os esforços no campo da cibersegurança têm sido cada vez mais voltados para este paradigma. Este ramo de cibersegurança foca-se nas melhores e mais robustas práticas de desenvolvimento de aplicações Web de forma a garantir a confidencialidade, integridade e disponibilidade dos dados (Bach-Nutman, 2020a; Veres & Wilkinson, 2023; Wen & Katt, 2023).

3.21.1 OWASP Top 10

A OWASP (ou *Open Web Application Security Project*) é uma organização sem fins lucrativos que fornece orientação e recursos para garantir a proteção e segurança de aplicações e sistemas baseados na *Web* (Veres & Wilkinson, 2023). O recurso mais conhecido e bem reconhecido pela comunidade científica é o *OWASP TOP 10* que consiste numa lista das vulnerabilidades mais perigosas, críticas e bem estudadas (Bach-Nutman, 2020b; Veres & Wilkinson, 2023; Wen & Katt, 2023). Esta lista existe para que os desenvolvedores, programadores, profissionais de segurança e organizações possam identificar e mitigar os possíveis riscos associados às suas aplicações e sistemas baseados em *Web*. Estes recursos estão publicamente disponíveis, acessíveis e muito bem documentados sendo que a tarefa de aprendizagem e estudo das vulnerabilidades documentadas torna-se extremamente fácil.

As vulnerabilidades que constam neste *Top 10* estão representadas na Figura 20, sendo que a sua versão mais atualizada é de 2021.

Figura 20 - OWASP Top 10 (adaptado de (Veres & Wilkinson, 2023))

3.21.2 Broken Access Control

Ocupando o primeiro lugar na lista *OWASP TOP 10*, a vulnerabilidade de *Broken Access Control* (ou BAC) é uma vulnerabilidade que ocorre quando o acesso a dados ou recursos sensíveis não é corretamente controlado ou definido sendo que possíveis atacantes podem obter acesso não autorizado a estes mesmos recursos ou à execução de ações não autorizadas (Maruf Hassan et al., 2018; Sharma, 2023). Esta vulnerabilidade é considerada uma ameaça muito séria à integridade de um sistema ou aplicação informática visto que certos dados ou ações devem apenas ser acedidos ou executados por pessoas ou entidades específicas. O acesso ou execução indevida a dados ou ações/comandos compromete toda a integridade do sistema.

Sendo assim o BAC pode ser causado por vários fatores gerais como:

- Mecanismos pobres ou mal configurados de controlo de acesso;
- Erros no código-fonte.

Um dos métodos mais populares e eficazes na proteção de dados e recursos sensíveis em sistemas informáticos é o RBAC (ou Controlo de acesso baseado em *roles* ou papéis) que consistem num sistema de segurança capaz de facultar acesso, com vários graus de limitação, a diferentes tipos de utilizadores com base na sua *role* ou função designada (Uddin et al., 2019). Um mecanismo RBAC pode ser implementado de muitas formas diferentes não havendo uma convenção propriamente dita para a sua conceção e implementação, contudo existem algumas noções base que terão de fazer parte de qualquer RBAC:

- ***Roles* –** Título do papel ou função que um determinado tipo de utilizadores tem no sistema sendo que cada *role* possui um conjunto de permissões que lhe são concedidas no sistema;
- **Permissões** – Constituem as regras para o que um determinado utilizador pode fazer e aceder, bem como o que não pode;

- **Grupos** – Conjuntos de utilizadores que partilham da mesma *role* no sistema sendo que se torna mais fácil, simplesmente, acrescentar um utilizador a um grupo de uma determinada *role* do que atribuir permissões singulares a esse utilizador.

O mecanismo RBAC é uma solução muito apelativa aquando da proteção de recursos e definição de permissões no *design* de um sistema pela sua simplicidade de compreensão e implementação, pela sua grande eficácia, quando bem desenhado e implementado e pela sua flexibilidade (Sharma, 2023; Uddin et al., 2019). Na Figura 21 é possível observar como, tipicamente, funciona este mecanismo.

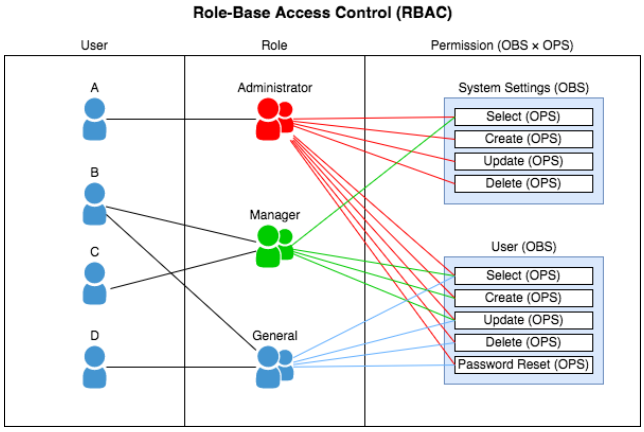

Figura 21 - Controlo de Acesso Baseado em *Roles*

3.21.3 Cryptographic Failures

Anteriormente conhecido como exposição de informação sensível, esta vulnerabilidade consiste na utilização incorreta, ou na falta completa de utilização, de técnicas de criptografia no manuseamento e transferência de dados sensíveis numa aplicação ou sistema informático (Sharma, 2023). Falhas criptográficas podem levar à exposição de dados sensíveis ou até mesmo ao comprometimento completo de uma aplicação ou sistema devido à implementação de algoritmos fracos ou desatualizados, manuseamento impróprio de chaves criptográficas e à falta de encriptação de dados sensíveis (Sharma, 2023).

3.21.4 Injection

Uma das vulnerabilidades mais bem conhecidas e estudadas na comunidade científica, a injeção de código ou instruções maliciosas por meio de controladores e código mal implementados permite a alteração e controlo sobre bases de dados (Dorai & Kannan, 2011; Kaushik & Majumdar, 2019; Sharma, 2023), código (Staicu et al., 2016a) e qualquer tipo de tecnologia ou implementação baseada em comandos singulares. Esta vulnerabilidade ocorre quando o utilizador é capaz de interagir com uma dada

tecnologia ou sistema por meio de *inputs* sendo que estes não são validados ou "sanitizados" e, portanto, irão ser executados tal e qual foram passados pelo utilizador (Sharma, 2023). Como por exemplo um sistema onde existem uma Base de Dados relacional e que permite, através de um Rest API, a pesquisa pela coluna "nome" com uma expressão SQL SELECT pré-feita. Neste caso um utilizador pode, perfeitamente, construir o seu próprio comando SQL malicioso e passá-lo como atributo em vez de um nome, se o sistema não validar o parâmetro passado o comando malicioso será executado na Base de Dados sem qualquer problema.

3.21.5 Insecure Design

A vulnerabilidade de *design* inseguro é algo difícil de definir sendo que se trata de uma área bastante ampla e particular de sistema para sistema e constitui uma ameaça muito séria à sua integridade, visto que falhas de implementação são facilmente mitigadas como criptografia ou injeções, mas falhas de *design* não. Dependendo da fase do desenvolvimento de um sistema ou aplicação uma falha grave de *design* pode significar o fim ou o cancelamento do mesmo (M. Aljabri et al., 2022). Um sistema mal pensado trará sempre falhas que não serão corrigíveis ou mitigáveis pela mais perfeita das implementações sendo que o sistema de interoperabilidade foi concebido com base em *designs* de software altamente estudados e aprovados pela comunidade científica como as melhores soluções e fluxos lógicos para a resolução e execução das suas funções específicas (Gamma et al., 1994).

3.21.6 Security Misconfiguration

Más configurações de segurança constituem uma grande ameaça ao funcionamento seguro de um sistema ou aplicação devido a configurações inseguras pré-definidas, configurações incompletas, cabeçalhos HTTP mal configurados e mensagens de erros capazes de expor informação sensível acerca do sistema (M. Aljabri et al., 2022). Esta vulnerabilidade é algo bastante difícil de averiguar num sistema devido à quantidade de configurações possíveis facultadas por todas as suas tecnologias integrantes sendo que a sua única solução está no estudo e implementação correta das mesmas num dado sistema.

3.21.7 Vulnerable and Outdated Components

Esta vulnerabilidade possui um conceito extremamente simples e parte da noção de que todo o *software* é desenvolvido a partir de *software* já existente na forma de bibliotecas e APIs de abstração de funcionalidades. Este tipo de vulnerabilidade dá-se a partir do momento em que um destes componentes

carrega uma vulnerabilidade conhecida (M. Aljabri et al., 2022). Esta vulnerabilidade é particularmente preocupante para sistemas baseados em Web devido à sua grande dependência de bibliotecas e APIs. A única forma de mitigação deste tipo de vulnerabilidade está no estudo extenso das várias tecnologias, bibliotecas e APIs a utilizar, bem como a sua monitorização e acompanhamento sendo extremamente importante a atualização das mesmas para as suas versões mais seguras ao longo do tempo.
Ao contrário de muitas outras vulnerabilidades a *vulnerable and outdated componentes* não é algo que se possa dar como mitigado, sendo que é perfeitamente plausível o surgimento de uma nova vulnerabilidade numa biblioteca ou API que comprometa todo o sistema de um dia para o outro como foi o caso com a biblioteca de Java ***log4j*** em 2022 (Juvonen et al., 2022).

3.21.8 Identification and Authentication Failures

Esta vulnerabilidade está relacionada com o método ou mecanismo de autenticação montado e implementado para uma dada aplicação ou sistema informático. A tarefa de autenticação é uma das mais importantes num sistema visto que está encarregue de facultar as diferentes funcionalidades e recursos do sistema a diferentes utilizadores de forma correta (M. Aljabri et al., 2022). A implementação de um sistema fraco irá resultar na disponibilização de funcionalidades e recursos aos utilizadores de forma errada e abrirá portas à utilização fraudulenta da própria aplicação ou sistema informático (M. Aljabri et al., 2022).

3.21.9 Software and Data Integrity Failures

O desenvolvimento de *software* moderno está cada vez mais complexo sendo que novas convenções e estratégias são concebidas, implementadas e estudadas todos os dias, porém nem todas as estratégias são as mais corretas e isentas de falhas. Esta vulnerabilidade está relacionada com a integridade mal conseguida ou não garantida do código fonte ou dos dados que fluem por um dado sistema. Desta forma, atacantes poderão abusar de certas falhas nestas componentes para ganhar acessos indevidos a dados sensíveis, aprender e estudar a arquitetura de um dado sistema. Outro perigo seria a introdução de uma nova funcionalidade mal desenvolvida e não testada que pode vir a comprometer ou facilitar qualquer uma das vulnerabilidades até agora descritas.

3.21.10 Security Logging and Monitoring Failures

Uma convenção extremamente útil e necessária no desenvolvimento de *software* moderno é a criação e armazenamento de *log*s ou registos em todas as componentes passíveis desta tarefa. O armazenamento de *logs* são a solução para a minimização e ambiguidade da informação exposta pelas mensagens de erro aos utilizadores finais (M. Aljabri et al., 2022).

A forma como estes *logs* são criados e geridos dependem unicamente do *software* em desenvolvimento bem como o seu propósito, sendo que estes devem conter informação útil e necessária à manutenção do *software*. Uma má estratégia para a gestão e armazenamento destes *logs* pode resultar no acesso fraudulento de informações sensíveis destinadas aos olhos dos desenvolvedores do sistema o que poderá permitir que atacantes possam descobrir diversos detalhes acerca da solução.

3.21.11 Server Side Request Forgery

SSRF (*Server-Side Request Forgery*) é uma das vulnerabilidades mais perigosas identificadas pela OWASP, sendo que esta surge quando um sistema baseado em Web permite que um utilizador influencie qualquer pedido a sistemas internos por parte do servidor através de parâmetros (M. Aljabri et al., 2022). Mais uma vez a manipulação indevida dos parâmetros destinados à interface entre utilizador e sistema constituem o ponto de entrada para esta vulnerabilidade sendo que se torna facilmente mitigável através da validação e saneamento de todos os *inputs* realizados pelo utilizador. A vulnerabilidade de SSRF é particularmente preocupante em sistemas baseados em microserviços devido à sua necessidade de intercomunicação interna.

4. INTEROPERABILIDADE

As organizações e instituições de saúde têm, ao longo do tempo recolhido, gerado e/ou acumulado quantidades colossais de dados acerca dos seus pacientes quer por motivos históricos quer pela esperança de que algum dia estes dados possam conter algum tipo de informação útil, porém quanto mais dados existem mais difícil se torna a atividade de extração de informação. Existe, portanto, um grande potencial, praticamente, inexplorado na análise e extração de informação destes dados históricos quer já estejam armazenados quer estejam a ser gerados neste preciso momento (Lopes et al., 2020).

Algo que também se tornou aparente na minha pesquisa foi que toda a implementação e integração de modelos de Inteligência Artificial/Data Mining/Otimização ou/em sistemas ABI necessita da customização de processos e/ou software levando a uma complexidade elevada aquando da adoção de uma ou múltiplas soluções.

Sendo assim o foco deste trabalho de investigação e desenvolvimento será uma arquitetura e sistema capazes de utilizar as informações já existentes nas organizações de saúde para interagir e interoperar com sistemas ABI e/ou sistemas capazes de utilizar e correr modelos.

Após uma pesquisa relacionada com o tipo de informação utilizada, gerada e trocada por organizações de saúde cheguei, rapidamente, à conclusão de que o padrão HL7 é o padrão adotado por, praticamente, todo o setor da saúde e então será com foco no HL7 que a arquitetura e sistema serão pensados e desenvolvidos.

4.1 Padrão HL7

Health Level Seven ou HL7 trata-se de um conjunto de padrões e normas internacionais para a troca e interoperabilidade padronizada de informações médicas e relevantes ao setor da saúde. Foi desenvolvido da década de 1980 para responder e resolver a necessidade da intercomunicação e interoperabilidade entre diferentes sistemas de saúde. As normas HL7 são utilizadas por uma grande parte de organizações de cuidados de saúde como Hospitais, clínicas e farmácias (Noumeir, 2019).

As normas e convenções definidas pelo HL7 têm o foco centrado na sintaxe apenas, definindo o formato e conteúdo das mensagens eletrónicas trocadas entre sistemas diferentes de saúde (Miranda et al., 2012) permitindo assim que a comunicação possa ser feita a partir de plataformas de software e hardware diferentes, criadas por desenvolvedores diferentes e que ferramentas possam ser feitas com a confiança de que serão bem integradas desde que obedeçam ao padrão (Noumeir, 2019). Precisamente por ser um padrão ao comunicar com outro sistema pressupõe-se que este esteja equipado com as ferramentas necessárias para receber e compreender a mensagem de forma a haver comunicação livre de erros ou problemas de suporte.

A padronização HL7 atua na camada de aplicação, sétima camada ou nível sete no modelo de comunicações de sete camadas da **Interligação de Sistemas Abertos** ou ***Open Systems Interconnections*** **(OSI)** definido pela **Organização Internacional da Padronização** ou ***Internation Organization for Standardization*** (AlQudah et al., 2021).

Para além da facilidade de uso, implementação e desenvolvimento o padrão também ajuda a garantir que os dados de cuidados de saúde, que são por natureza muito sensíveis e confidenciais, sejam transmitidos de forma exata e segura (Noumeir, 2019).

Estas normas desempenham um papel importante na melhoria da eficiência e da qualidade dos cuidados de saúde, ao permitir a interoperabilidade entre diferentes sistemas ajudam a reduzir o custo dos

cuidados de saúde, a melhorar a segurança dos pacientes e a facilitar a investigação clínica visto que é como dizer que "toda a gente fala a mesma linguagem" (Miranda et al., 2012; Noumeir, 2019).

Isto é extremamente pertinente para o presente trabalho de investigação e desenvolvimento visto que é praticamente garantido que, não importe a organização, o HL7 será o padrão e formato das suas informações e dados de saúde.

Da mesma forma que qualquer outro projeto ou iniciativa o HL7 é, também, um projeto em desenvolvimento e está em constante melhoria sendo que existem várias versões e conforme o passar do tempo novos padrões dentro do HL7 **(HL7, HL7 FHIR, HL7 CDA)** foram criados e possuem níveis de aderência diferentes.

4.1.1 Padrão HL7 V2

Nas três últimas décadas a versão 2 do HL7 (HL7v2) tem sido o padrão principal na troca de dados clínicos (AlQudah et al., 2021; Miranda et al., 2012; Noumeir, 2019) tanto pela sua idade como também pela época em que saiu e começou a haver grande adesão. Atualmente, grande parte dos sistemas de informação contam com HL7v2 para se desenvolverem interfaces padronizadas capazes de se conectarem entre si e facilitar a troca de informação.

Este padrão cobre uma grande variedade de domínios como a Administração de Pacientes, Pedidos e Resultados Laboratoriais e Relatórios de Saúde Pública (AlQudah et al., 2021).

Na sua essência este padrão trata-se das regras a seguir aquando da geração/criação de uma mensagem para comunicação com outro sistema de informação. Desta forma uma mensagem HL7 é formada pelas seguintes componentes e características (Oemig & Blobel, 2011):

- **Segmentos**: Os segmentos são os blocos de construção das mensagens HL7 e são identificados por um código de três letras como "PID" ou "PV1" para representar diferentes segmentos e âmbitos de cada secção numa mensagem;
- **Campos**: São os elementos individuais de dados dentro de um segmento. São identificados numericamente (1,2,3,4,...) e fisicamente separados por um "|";
- **Subcampos**: São os elementos individuais de cada campo. São identificados numericamente (1,2,3,4,...) e fisicamente separados por um "^";
- **Tipos de Dados**: Os tipos de dados definem o formato e a gama de valores permitidos para cada campo e subcampo. O HL7v2 suporta uma grande variedade de tipos como valores de texto, numéricos, datas e códigos *unix*;

- **Repetição**: Campos e componentes podem ser repetidos várias vezes dentro de um segmento uma vez que isto permite a representação de estruturas de dados mais complexas;
- **Opcionalidade**: Os segmentos, campos e componentes podem ser opcionais e não tem de estar todos presentes em todas as mensagens tornando a estrutura das mensagens mais flexível e adaptável a diferentes casos.

De todos os segmentos possíveis nas mensagens HL7 apenas 1 é obrigatório estar em todas as mensagens, **MSH (Message Header)**. Este segmento representa o objetivo/intuito da mensagem, quem a enviou, qual é o seu destino e algumas especificações acerca da sintaxe da mensagem. Neste segmento também é obrigatório constar o tipo e *trigger* da mensagem juntamente com os seus códigos respetivos.

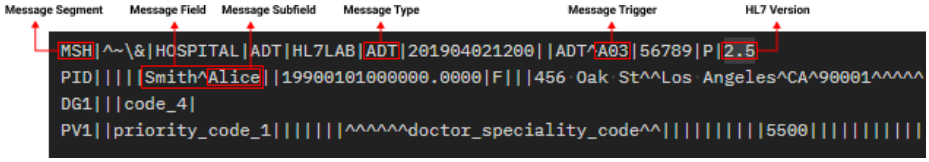

Figura 22 - Exemplo Mensagem HL7

Na Figura 22 é possível observar todas as características das mensagens HL7 mencionadas acima sendo que esta mensagem em particular é proveniente do remetente "HOSPITAL", foi enviada dia 02/04/2019, é do tipo **ADT (Admit, Discharge, Transfer)**, foi criada com HL7v2.5 em mente e possui 3 segmentos de informação **PID (Patient Identification), DG1 (Diagnosis)** e **PV1 (Patient Visit)** todos eles devidamente populados com uma série de campos**.**

Para construir um sistema onde é possível, de forma automática e dinâmica, extrair informação destas mensagens será necessário saber exatamente onde se encontra cada pedaço de informação e para poder localizar qualquer campo ou subcampo numa mensagem HL7 apenas é necessário saber:

- **Tipo da Mensagem**: Nem todas as mensagens nem todos os tipos de mensagens vão ter obrigatoriamente todos os segmentos e campos mas há mensagens que tendem a ter certos campos de uma forma mais obrigatória que outras;
- **Trigger da Mensagem**: O trigger da mensagem, quando combinado com um tipo de mensagem permitem, por convenção, a passagem mais comum de certos segmentos e campos;
- **Segmento,Campo e Subcampo**: Estas três características permitem localizar qualquer informação em qualquer mensagem.

Em teoria, quando estas informações são reunidas será possível criar um mapeamento que nos permite procurar informações específicas em mensagens específicas. Estes mapeamentos serão a base do sistema de Interoperabilidade no que toca ao suporte HL7.

4.1.2 Padrão HL7 FHIR

Fast Healthcare Interoperability Resources, ou FHIR, é um dos padrões desenvolvidos pela organização "Health Level Seven International" e lançado formalmente em Setembro de 2013 para a troca e intercâmbio de informação relativa aos cuidados de saúde de forma eletrónica. Tal como o HL7 trata-se de uma especificação de formato de dados e API que define como a informação deve ser estruturada, construída e distribuída. O FHIR foi concebido e desenvolvido para ser altamente flexível e extensível de forma que possa ser utilizado para suportar virtualmente todo o tipo de sistemas e aplicações independentemente de quem as faça (Vorisek et al., 2022).

No momento da escrita do presente documento o FHIR encontra-se em ascensão bastante acelerada na popularidade e adoção na indústria da saúde e cuidados médicos e com muito bons motivos, este padrão possui bastantes vantagens face a outros métodos de troca de informação médica:

- **Interoperabilidade**: O FHIR foi concebido e desenhado para ser interoperável com outros sistemas de saúde sendo que informação pode, facilmente, ser trocada por sistemas diferentes;
- **Flexibilidade**: O FHIR é flexível e extensível precisamente para poder ser utilizado de forma a dar suporte a virtualmente qualquer tipo de aplicação ou sistema;
- **Escalabilidade**: O padrão FHIR é altamente escalável de forma a poder suportar estruturas de dados complexas assim como grandes quantidades de dados;
- **Segurança**: A informação de pacientes, quando emitida num recurso FHIR, tem de seguir as devidas normas de segurança para proteger todos os dados sensíveis do mesmo.

Por estas vantagens o padrão FHIR está em grande crescimento e utilização no setor tecnológico para a troca de informação entre sistemas, como sugere o gráfico presente na Figura 23 que representa o número de publicações feitas de artigos e estudos acerca do FHIR ao longo do tempo, e as suas aplicações variam muito pois está concebido para independentemente do sistema, se utilizar FHIR, está capaz de comunicar com outros sistemas. Aplicações do FHIR são, por exemplo, Registos Eletrónicos de saúde (EHRs), sistemas de apoio à decisão clínica (CDSSs), sistemas de vigilância de saúde pública ou pesquisa (Vorisek et al., 2022).

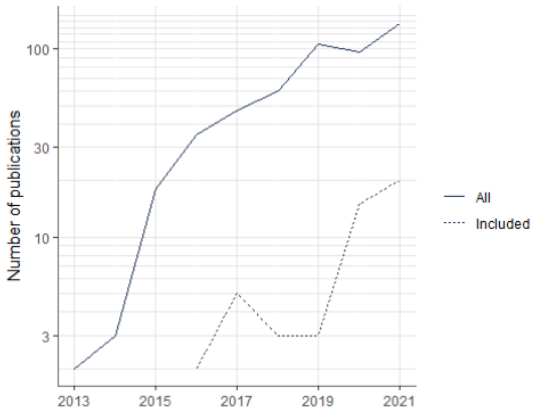

Figura 23 - Publicações de estudos sobre o FHIR ao longo do tempo (adaptado de (Vorisek et al., 2022))

De uma forma sucinta, a estrutura FHIR fundamenta-se no conceito de **Recursos**, ou *Resources*, os quais são unidades autónomas e modulares de informação sobre cuidados de saúde e atividade médica, organizadas de maneira hierárquica na qual cada recurso possui um conjunto exclusivo de atributos ou campos associados de acordo com o seu tipo (Ayaz et al., 2023). Estes recursos podem ser colocados uns dentro dos outros de forma a ser possível o suporte de estruturas de dados mais complexas e também podem ser ligados de forma a representar relações entre tipos de informação diferentes.

A estrutura FHIR é composta por:

- **Recursos:** Unidades autónomas e modulares de informação sobre cuidados médicos. Cada tipo de recurso representa um tipo de informação médica como **Paciente**, uma **Observação** ou uma **Medicação** (Ayaz et al., 2023)**;**
- **Campos:** Definem a estrutura dos dados contidos num recurso segundo o **Tipo** do recurso e constituem o total da informação médica nele contida. Por exemplo um recurso Paciente terá de conter, obrigatoriamente, os campos **nome, data de nasciemento** e **género** (Ayaz et al., 2023)**;**
- **Tipos de Dados:** Definem o tipo e estrutura de dados que cada campo pode assumir, como por exemplo o campo nome pode assumir o tipo de dado ***String*** mas o campo data de nascimento já terá de assumir o tipo ***Date*** (Ayaz et al., 2023)**;**
- **Extensões:** Extensões podem ser utilizadas nos recursos FHIR para, como o próprio nome indica, estender as capacidades e cobertura de um recurso com campos e informação customizada que não está definida no núcleo do padrão e especificação FHIR (Ayaz et al., 2023).

Sendo o FHIR uma abordagem relativamente moderna quando comparado ao HL7 base a representação dos seus recursos já assume um formato de dados muito mais recente e fácil de trabalhar (Ayaz et al., 2023). No momento da escrita do presente documento apenas JSON, XML e Turtle (TTL) são suportados

o que já permite um desenvolvimento moderno de sistemas de interoperabilidade no setor da saúde (Ayaz et al., 2023; Vorisek et al., 2022) e como é possível observar na Figura 24 são equivalentes e intercambiáveis atingindo exatamente os mesmos fins independentemente do formato escolhido.

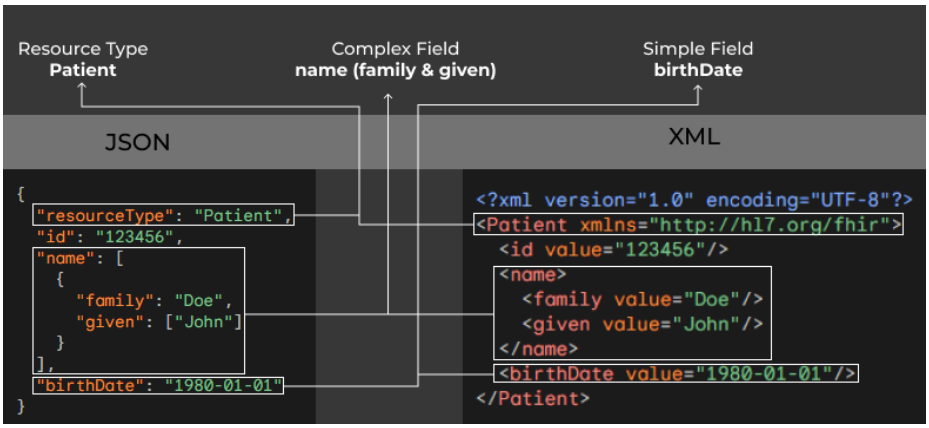

Figura 24 - Exemplo Recurso FHIR JSON/XML

Olhando para o estado atual da interoperabilidade nos sistemas de informação no setor da saúde começa a tornar-se claro onde será o foco num sistema de interoperabilidade para sistemas ABI bem como os possíveis usos e vantagens que a integração deste tipo de recursos possa vir a ter.

5. Proposta E Conceção do Sistema

Com base na informação recolhida e pesquisa efetuada a grande oportunidade e foco deste trabalho de dissertação está na utilização de sistemas ABI(Faria et al., 2022; Fernandes et al., 2022; Ferreira et al., 2019; Gonçalves et al., 2020; Lopes et al., 2020) para correr modelos preditivos e de otimização utilizando toda a informação colossal na forma de dados históricos, mensagens e recursos de interoperabilidade partilhados por todo o setor informático na área da saúde de forma a auxiliar no processo de tomada de decisão, gerar conhecimento e melhorar toda a atividade.

5.1 Definição do Âmbito

O sistema de interoperabilidade será uma nova camada no bolo que é um sistema ABI. Servirá para, de uma forma ininterrupta, utilizar e extrair a informação que já está disponível nos recursos de interoperabilidade já utilizados pelas organizações de saúde e passá-la para um sistema ABI existente ou qualquer outro tipo de sistema que faça uso deste tipo de informação.

Como tal é possível afirmar que este sistema surge num ambiente com dois grandes pressupostos:

- **Clientes-alvo possuem/comunicam utilizando recursos de interoperabilidade**: O sistema será concebido para utilizar recursos de interoperabilidade como mensagens HL7 ou recursos FHIR como *input* para execução de modelos arbitrários num dado sistema ABI. Isto requer que qualquer cliente necessite de providenciar algum destes recursos pelo que a pesquisa bibliográfica revela que não será um problema devido à sua popularidade e convenção no setor da saúde;
- **Infraestrutura necessita de um sistema ABI pronto a ser utilizado**: Este sistema apenas irá interoperar entre Cliente - Sistema ABI e entre Sistema ABI – Cliente pelo que necessita já de um sistema ABI ou um sistema parecido que seja capaz de, arbitrariamente, correr modelos preditivos ou de otimização aquando fornecido algum *input*.

5.2 Funcionamento do Sistema

Para o funcionamento foi tido em conta que tipo de modelos vão estar a ser corridos e de que forma. Os modelos que se pretendem correr foram ou serão desenvolvidos tendo por base a metodologia CRISP-DM que dita que, para desenvolver um modelo, é necessário recolher dados relevantes, percebê-los, documentá-los, tratá-los e processá-los de forma que sejam úteis no desenvolvimento de um modelo (Azevedo & Santos, 2008). Ao analisar a metodologia e processos gerais de desenvolvimento e criação de modelos preditivos foi possível retirar algumas conclusões e fundamentos vitais no desenvolvimento deste sistema:

- **Dois tipos de Dados**: Geralmente falando, os dados que entram para o treino de um modelo são sempre um entre dois, **categóricos** ou **contínuos**, onde as categóricas fazem parte de uma escala, hierarquia finita ou categorias ao passo que as contínuas não têm propriamente limite ou uma classe a que pertencer (Ke et al., 2023). Este fator é muito importante porque também dita que tipo de tratamento os dados irão sofrer antes de estarem prontos para treinar e, por sua vez, correr no modelo;
- **Apenas valores numéricos**: Qualquer que seja o modelo desenvolvido, os seus dados de treino serão sempre tratados para a forma de valores numéricos (Azevedo & Santos, 2008; Faria et al., 2022b; Gonçalves et al., 2020) pelo que para a sua utilização novos dados também terão de o ser. Este tratamento também irá variar muito entre dados diferentes e tipos de dados diferentes;

- **Quando os modelos acabam de ser treinados e testados não costumam mudar**: Aquando do término do desenvolvimento, treino e avaliação de um modelo, este costuma ficar acessível e continuar acessível sem grandes alterações desde a altura da sua implementação (Azevedo & Santos, 2008). Se um modelo estiver pronto a utilizar num sistema ABI, muito provavelmente, não irá sofrer alterações nem ficar indisponível pelo que outros sistemas podem contar com a sua presença como garantida.

Estas observações foram a base da conceção do modelo funcional do sistema de Interoperabilidade pelo que é importantíssimo saber como um modelo é desenvolvido para saber a forma como se vai interagir com o mesmo. O funcionamento base do sistema conta com algumas "partes móveis" e individualmente configuradas para obter o máximo de customização e configuração de forma a poder, com um alto nível de sucesso, utilizar recursos de interoperabilidade como entrada a um sistema ABI. A representação deste funcionamento é observável na seguinte Figura 25.

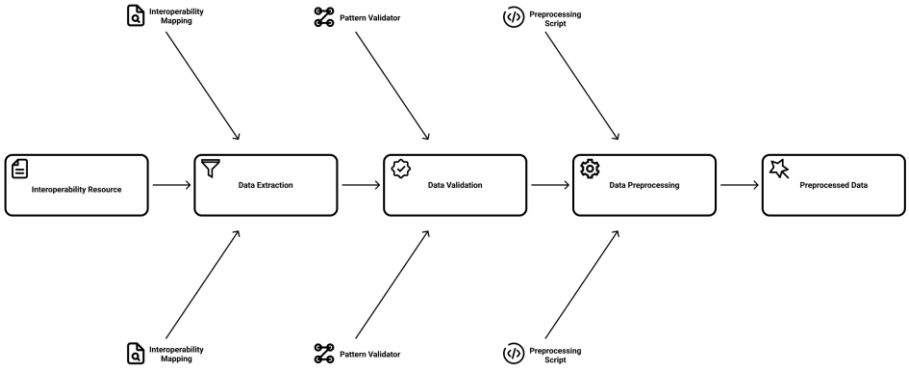

Figura 25 - Funcionamento Base do Sistema de Interoperabilidade

Este funcionamento é a base do sistema de interoperabilidade e explica como é possível partir de um recurso de interoperabilidade, ou vários, e, com alta confiança, converter a informação pertinente a um determinado modelo num pedido adequado ao mesmo. É de notar ainda que este processo permite qualquer caso de uso identificado, quer seja o uso arbitrário de um modelo, como também o treino de um novo modelo. Resumidamente trata-se da recriação das primeiras fases da metodologia CRISP-DM de extração e transformação de dados oriundos de recursos de interoperabilidade garantindo assim máxima compatibilidade com modelos desenvolvidos através da mesma metodologia.

Este fluxo segue e adapta o padrão de design de *Pipeline (Vermeulen et al., 1995)*, um dos padrões de *design* de software mais bem conhecidos pela sua eficácia e simplicidade.

Este processo conta com três grandes operadores na tarefa do processamento de um recurso de interoperabilidade:

- **Extração de Dados**: Este é o primeiro operador na corrente. Neste passo serão extraídas as informações necessárias para formar um pedido completo a um determinado modelo. Esta extração será feita à base de **mapeamentos** altamente configuráveis e customizáveis por design e únicos por modelo e atributo. Neste passo um ou mais recursos serão avaliados e todos os mapeamentos serão experimentados tirando os que, entretanto, foram encontrados, até todos os mapeamentos estarem cumpridos. Caso algum mapeamento não seja cumprido, o processo é interrompido e uma mensagem é enviada informando que o pedido está incompleto;
- **Validação de Dados**: Este operador existe para duas grandes finalidades, para garantir que um *input* externo não é maligno uma vez que estes *inputs* irão passar por todo o sistema e serão guardados para efeitos históricos e também para garantir que o *input* bate certo com o *input* esperado nas lógicas de preprocessamento caso contrário este seria algo inconsistente. Repetindo o design dos mapeamentos este passo também conta com validadores individuais altamente configuráveis e únicos por modelo e atributo para garantir o máximo de flexibilidade e compatibilidade com qualquer preprocessador. Falhando pelo menos um validador o processo é abortado e é enviada uma mensagem informando que alguma informação passada num recurso não é válida;
- **Preprocessamento de Dados**: Este será o último operador na cadeia e conta com a informação extraída para ter o que processar e conta que esta esteja num certo formato para ter como a processar. Ambas estes requisitos são garantidos e reforçados através dos operadores anteriores. Chegando a este operador na cadeia pode-se correr o processador com confiança pois uma *String* será sempre uma *String* com o mesmo tipo de formatação e uma data será sempre uma data com a mesma formatação. Por esta garantia e alta confiabilidade é possível aplicar o mesmo design dos operadores anteriores pelo que o sistema conta com preprocessadores singulares e únicos por modelo e atributo para garantir que cada modelo possa ter um preprocessamento à medida caso necessário. Neste operador também é possível alguma coisa falhar pelo desenvolvimento defeituoso de algum preprocessador, nesse caso o processo é interrompido e é enviada uma mensagem a informar da falha sucedida.

Á partida pode parecer um processo simples, porém parte de alguns pressupostos não tão simples e cujo cumprimento é vital para o processo funcionar como apresentado:

- É possível criar mapeamentos para, de forma arbitrária, extrair algum tipo de informação, onde quer que ela esteja, de recursos HL7;

- É possível criar mapeamentos para, de forma arbitrária, extrair algum tipo de informação, onde quer que ela esteja, de recursos FHIR;
- É possível criar validadores para, de forma arbitrária, validar um certo dado;
- É possível criar preprocessadores para, de forma arbitrária, transformar um certo dado de entrada num dado de saída.

Responder e garantir estes pressupostos é vital tanto para o funcionamento base do sistema como para a sua proposta.

5.2.1 Funcionamento Base dos Operadores Arbitrários do Sistema

Devido à natureza volátil e altamente mutável dos requisitos que cada atributo de um qualquer modelo possui será necessário adotar/desenvolver um mecanismo que permita a reutilização do mesmo algoritmo de extração/validação/preprocessamento, porém com total permutabilidade do conteúdo e lógica do algoritmo. Tal como o funcionamento de uma consola de jogos e um jogo, o operador será como a consola e o mapeamento, validador ou preprocessador será o jogo.

Este comportamento pode ser descrito e o funcionamento será baseado no *design pattern* de ***Strategy*** onde é definida uma família de algoritmos que serão encapsulados de forma a tornarem-se intercambiáveis. Este padrão permite então que o conteúdo do algoritmo varie conforme a situação em tempo de execução em vez de o código ter de estar presente no código fonte tornando-o mais flexível e reutilizável (Gamma et al., 1994).

Para este padrão ser possível será necessária a criação de um construtor destes pequenos objetos executáveis assim como um motor que os consiga correr sendo que no tempo de execução, ou *runtime*, apenas se passam os objetivos executáveis, de forma arbitrária, à classe ou motor capaz de os executar (Gamma et al., 1994).

A adoção deste padrão proporciona várias vantagens em comparação ao desenvolvimento tradicional estático de código como a total dinamicidade e customização de uma determinada parte do sistema e garantindo o funcionamento e suporte em praticamente qualquer caso. Porém não é o padrão mais indicado para todas as situações sendo que aumenta a complexidade de uma solução e aumenta o risco de erros sendo que nunca haverá uma lógica principal, mas sim muitas lógicas criadas e executadas de forma dinâmica.

Todas as implementações dos operadores arbitrários do sistema de interoperabilidade (extração, validação e preprocessamento) serão adaptações deste padrão de software pelo que todos terão um construtor de objetos executáveis e um motor de execução preparado para executá-los.

5.2.2 Extração Arbitrária de Dados de recursos HL7

Seguindo o padrão de design de ***Strategy*** (Gamma et al., 1994) será necessário criar mapeamentos HL7, que permitam tendo uma mensagem extrair qualquer informação, um motor de criação destes mapeamentos e um motor de execução.

Partindo do formato das mensagens HL7 um mapeamento para extrair um qualquer dado específico irá precisar de ter definidos:

- **Tipo de Mensagem:** O Tipo da mensagem será a primeira verificação pelo que cada tipo de mensagem possui uma série de atributos que lhe são convencionalmente exclusivos;
- **Trigger da Mensagem:** O Trigger da mensagem será avaliado em conjunção com o tipo de mensagem pois a combinação destas duas características dita que segmentos e campos é espectável encontrar dentro da mensagem;
- **Segmento:** O Segmento é a classe do campo que queremos extrair com o mapeamento;
- **Campo e Subcampo:** O campo e ou combinação com um subcampo representam exatamente a informação que se pretende extrair da mensagem.

Todas estas características constituem um mapeamento HL7 que terá de ser executado num motor HL7 em conjunção com uma mensagem de forma a retirar ou não a informação pretendida.

Esta seria uma tarefa bastante árdua e pesada tendo em conta o formato único do HL7 sendo necessário criar um *parser* de raiz assumindo os riscos apegados a uma tarefa desta natureza como possíveis erros e *bugs* e/ou falta de suporte em alguns ou muitos casos de uso. Ao invés a melhor opção será mesmo utilizar uma das muitas soluções abertas, ou *open-source*, que permitam a manipulação completa deste formato de dados.

Para este efeito, depois de alguma pesquisa, foi averiguado que a melhor ferramenta para este fim será a ferramenta de integração **Mirth Connect**.

O Mirth Connect é, precisamente, uma plataforma de integração de dados relativos aos cuidados de saúde, como HL7, *open source* que pode ser utilizada para a ligação entre diversos sistemas, transformar ou encaminhar dados entre eles (J. Lin et al., 2019; Rodriguez et al., 2017).

O Mirth, como plataforma, possui bastantes vantagens que o fazem parecer a escolha ideal para interação, gestão e manipulação de recursos HL7 e outros recursos de saúde:

- **Custo**: Sendo uma plataforma *open source* a utilização do mirth não necessita de uma licença ou de um investimento monetário, porém alguns *plugins* que proporcionam funcionalidades adicionais requerem compra (J. Lin et al., 2019);

- **Suporte/Escalabilidade/Desempenho**: O Mirth é um projeto de longa data, tendo a sua primeira versão sido lançada em 2006 pelo que teve bastante tempo para maturar e se tornar uma ferramenta útil e fidedigna com grande capacidade de suporte, escalabilidade e desempenho (Rodriguez et al., 2017);
- **Flexibilidade**: O Mirth é uma ferramenta muitíssimo flexível pelo que permite ao utilizador programar os seus próprios canais e lógica de forma completamente livre em JavaScript. Os canais Mirth podem ser implementados em qualquer fluxo de negócio pois o Mirth oferece um leque grande de suporte de integrações como *listeners* Sockets ou HTTP (J. Lin et al., 2019; Rodriguez et al., 2017).

A utilização do Mirth neste sistema passará pela criação de um simples canal capaz de fazer a extração de informação das mensagens HL7. Este canal estará disponível para ser utilizado pelo sistema como uma simples *programming API* para auxílio ao desenvolvimento do motor HL7 e estará, também, devidamente protegido por um par de credenciais conhecido apenas ao sistema.

Sendo assim, o mecanismo de execução de mapeamentos HL7 será feito através do Mirth Connect e surge sob a forma de um canal capaz de receber um mapeamento e uma mensagem HL7 e retornar a informação do campo pretendido alcançável/disponível através de um *HTTP listener*.

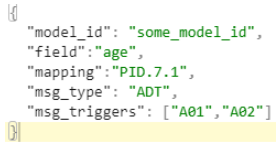

Figura 26 - Conceito de um mapeamento HL7

Como se pode observar na Figura 26, um mapeamento para recursos HL7 terá de conter os metadados que o ligam a um e um só atributo de um dado modelo, neste caso o atributo *age* ou idade de um modelo com identificador *some_model_id*, bem como todos os dados considerados essenciais para a localização de um valor ou atributo dentro de uma mensagem HL7. Neste caso em particular o atributo idade estará mapeado ao sétimo campo do segmento PID (Patient Identification) em mensagens do tipo ADT com os triggers de mensagem A01 e A02. Passando este mapeamento e uma qualquer mensagem HL7 ao canal Mirth este será capaz de localizar a informação mapeada e retorná-la com sucesso, caso a mensagem não possua nada neste campo o canal deverá retornar uma mensagem customizada de forma a assinalar que a mensagem não possui a informação requisitada pelo mapeamento.

A integração deste mecanismo no sistema tornará possível a configuração e mapeamento completo, de forma dinâmica, de todos os atributos de um dado modelo para que possam ser, arbitrariamente, extraídos de recursos HL7.

5.2.3 Extração Arbitrária de Dados de recursos FHIR

Para a extração de dados de recursos FHIR a abordagem será a mesma da extração de recursos HL7 na medida que terão de existir mapeamentos referentes a recursos FHIR, um algoritmo capaz da geração dos mesmos e outro capaz da sua execução sendo que o fluxo de lógica será idêntico tendo um mapeamento e um recurso FHIR será possível a extração de um dado específico.

Para este fim é necessário ter em consideração o padrão FHIR e aquilo que caracteriza um dado específico neste âmbito:

- **Tipo de recurso**: Tal como acontece com o HL7, no FHIR cada tipo de recurso dita, convencionalmente, que tipos de dados irão nele constar. Por exemplo será muito mais provável encontrar o atributo **data de nascimento** num recurso do tipo Paciente do que num recurso do tipo Organização;
- **Estrutura do Recurso**: O FHIR é um padrão representado pelas estruturas de dados JSON ou XML que são, por design, estruturas não ordenadas de par-valor com alto grau de possível granularidade (B. Lin et al., 2012) pelo que o mapeamento irá depender quase unicamente do fluxo de atributos e valores de cada campo FHIR.

Com estas características em mente será possível criar um objeto de mapeamento que, tendo um qualquer recurso FHIR, irá ser capaz de programar a extração dinâmica de qualquer informação arbitrariamente definida no mapeamento.

Para este fim a criação e execução destes mapeamentos será possível através de qualquer motor de *parsing* quer de JSON ou XML. Maior parte das linguagens de programação de alto nível já (como por exemplo JavaScript, Python, C# ou Golang) possuem suporte nativo para estes formatos de dados e ferramentas ou APIs para a sua manipulação (B. Lin et al., 2012).

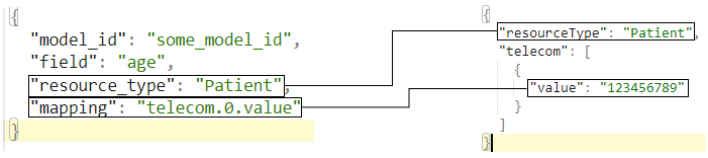

Figura 27 - Conceito de um mapeamento FHIR

Como é possível observar na Figura 27, o conceito para um mapeamento FHIR é extremamente similar ao de um mapeamento HL7 para não aumentar, desnecessariamente, o nível de complexidade da solução e aumentar a legibilidade e facilidade de desenvolvimento no suporte de recursos de interoperabilidade. Sendo assim os mapeamentos para recursos FHIR contam na mesma com os metadados, garantindo a sua ligação a um atributo de um modelo, mas também com o tipo de recurso

e o fluxo de dados esperado. Sendo o formato FHIR representado em JSON ou XML o mapeamento em si pode ser feito com uma notação que defina exatamente dentro do recurso onde está a informação pretendida. Para poder desenvolver esta notação foi necessário ter em consideração os tipos de estruturas de dados possíveis de representar com estes dois formatos:

- **Valores simples**: Tanto JSON como XML seguem a estrutura de atributo-valor sendo que este é o caso mais simples onde um atributo possui um valor simples como um *string*, um número ou um booleano (B. Lin et al., 2012);
- **Listas**: Ambos os formatos de dados estão preparados, de formas diferentes, para poder representar estruturas mais complexas de dados como uma lista de elementos. Estes elementos podem assumir a forma de qualquer outra estrutura de dados suportada tanto por JSON ou XML (B. Lin et al., 2012);
- ***Nested Objects***: Ambos os formatos de dados estão preparados para conter outro objeto JSON/XML contidos dentro dos seus atributos como um valor. Estes objetos dizem-se *nested* pois estão contidos dentro de um objeto pai e têm à sua disposição todo o leque de possíveis estruturas de dados para poder representar dentro de si. Esta é, de longe, a característica mais importante a ter em consideração aquando do desenvolvimento da notação de mapeamento pois a maior parte dos atributos representados pelo FHIR requerem que haja algum nível de *nesting* para estarem corretamente representados (B. Lin et al., 2012).

Tendo estes aspetos em consideração é perfeitamente possível representar qualquer dado em qualquer posição num objeto JSON ou XML através de uma notação que define todos os atributos, que têm de estar presentes até chegar ao valor desejado, separados por pontos ".".

A notação "**atributo.atributo.atributo.valor_numerico.atributo**" consegue representar através do nome dos atributos por onde um algoritmo visitante terá de passar para chegar ao valor. Para lidar com listas a notação será heurística sendo que se existir um valor numérico simples separado por dois pontos o visitante deve assumir que o atributo anterior se trata de uma lista e visitar o elemento com índice idêntico ao valor numérico.

O conceito do funcionamento da execução destes mapeamentos está representado na Figura 28.

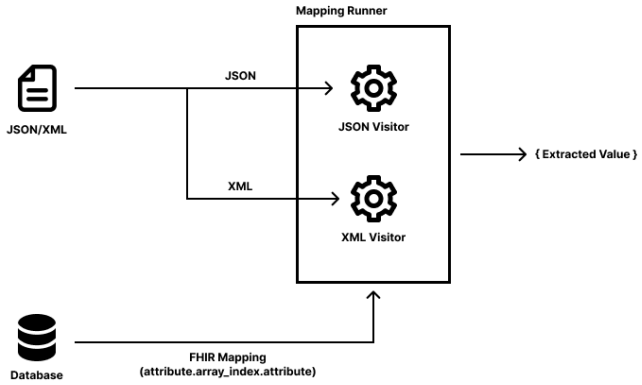

Figura 28 - Conceito da extração de dados FHIR

Estando a notação bem definida, voltando ao caso representado na Figura 27 o atributo *age* do modelo com id *some_model_id* estará mapeado para o atributo *value* de um objeto *nested* que constitui o elemento na 1ª posição da lista que constitui o atributo *telecom* de um recurso do tipo Paciente.

5.2.4 Validação Arbitrária de Dados de Entrada

A validação dos dados de entrada é a operação executada aquando da extração de todos os valores pretendidos dos recursos utilizando as estratégias propostas nos pontos anteriores. Utilizando o mesmo padrão de design que a extração de valores, a validação conta com vários validadores que podem ser arbitrariamente utilizados para validar um qualquer valor de forma dinâmica.

O funcionamento do mecanismo de validação pode ser observado na Figura 29. Este mecanismo conta com um valor de entrada oriundo do operador anterior (Extração) e será validado a partir de um validador vindo da base de dados e criado especificamente para esse valor. O mecanismo de validação irá, então, validar se o valor de entrada respeita ou não as regras estabelecidas no validador como por exemplo se a regra for **ser um número acima de 10000**, o número 999 ou até a palavra "João" serão considerados como não validos.

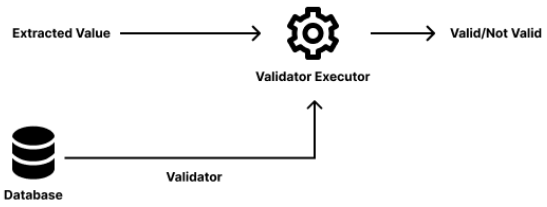

Figura 29 - Conceito do mecanismo de validação arbitrária

A tarefa e atividades de validação são algo essencial na criação e desenvolvimento de software, sendo que aumentam a segurança de utilização pois tudo o que seja valores de entrada ou *inputs* serão

validados rigorosamente de forma a não poderem causar quaisquer problemas uma vez dentro de um sistema e para maximizar a consistência nos seus fluxos de dados (Ficara et al., 2008; Thompson, 1968).

A razão principal para a existência deste mecanismo de validação é precisamente para garantir a consistência nos valores extraídos dos recursos de interoperabilidade pois estes terão de ser tratados sempre da mesma forma pelos preprocessadores na etapa seguinte do processo. Sem esta validação o preprocessamento tornar-se-ia inconsistente pois imaginando que a lógica de transformação seria multiplicar um valor por 1000, caso o valor extraído fosse uma data ou uma letra/palavra resultaria num erro ou até numa possível injeção de conteúdo maligno (Thompson, 1968) levando-nos à segunda razão pela qual este mecanismo existe, a segurança. A existência e imposição é um grande passo e esforço, que quando associado a todas as outras medidas de segurança, contribui para a garantia de segurança de todo o sistema de interoperabilidade.

Para este fim, o uso de Expressões Regulares, ou *Regular Expressions (regex)* (Ficara et al., 2008; Thompson, 1968) revelou ser ideal devido à sua eficiência e flexibilidade.

```
{
  "model_id": "some_model_id",
  "field": "age",
  "source_type": "hl7",
  "pattern": "/^[0-9]+$/"
}
```

Figura 30 - Exemplo de um validador

Como é possível observar na Figura 30 um objeto de validação ou validador para o sistema de interoperabilidade é bastante simples e conta com os metadados de identificação do mesmo e de ligação a um atributo *age* num modelo com identificador *some_model_id*. Também conta com um atributo de identificação de recurso para poder haver diferenciação entre validações para atributos HL7 e atributos FHIR e com o a expressão regular em si que servirá para a validação usando um mecanismo de execução de expressões. Sendo assim no caso específico no exemplo o atributo *age* deste modelo oriundo de uma mensagem HL7 terá de ser obrigatoriamente um valor numérico segundo a expressão "/^[0-9]+$/".

5.2.5 Preprocessamento Arbitrário de Dados de Entrada

A validação dos dados de entrada é a operação executada aquando da extração de todos os valores pretendidos dos recursos utilizando as estratégias propostas nos pontos anteriores. Utilizando o mesmo padrão de design que a extração de valores, a validação conta com vários validadores que podem ser arbitrariamente utilizados para validar um qualquer valor de forma dinâmica.

O preprocessamento é a última operação no processamento dos recursos de interoperabilidade no sistema e é executado aquando da validação de todos os atributos extraídos com sucesso para poder preparar ou transformá-los de acordo com os requisitos específicos de um modelo. É nesta etapa que valores como M ou F (masculino ou feminino) serão transformados nos valores 1 ou 2, respetivamente, de acordo com o solicitado pelo modelo.

Á semelhança dos mecanismos anteriormente descritos o mecanismo de preprocessamento de valores segue, também, o padrão de *design* de software de Strategy pelo que existem os seguintes conceitos:

- **Preprocessador**: Objeto autónomo capaz de conter um algoritmo ou lógica de preprocessamento criados/definidos circunstancialmente;
- **Gerador de Preprocessadores**: Mecanismo para gerar preprocessadores completamente customizáveis por *input* de utilizador;
- **Executor de Preprocessadores**: Mecanismo capaz de executar ou correr os preprocessadores com base na execução dinâmica de código de uma forma segura e robusta;

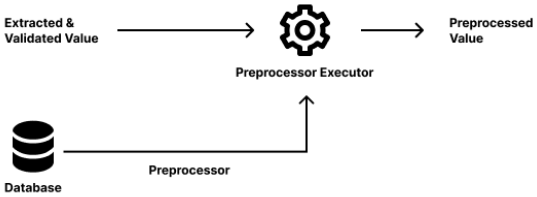

Figura 31 - Conceito de Mecanismo de Preprocessamento

O funcionamento do mecanismo de preprocessamento pode ser observado na Figura 31. Este mecanismo conta com um valor de entrada proveniente do mecanismo de validação pelo que parte do princípio de que, para um atributo de um modelo, este foi extraído e validado com sucesso e será processado ou transformado a partir do preprocessador vindo da base de dados e criado especificamente para esse mesmo atributo. Este preprocessadores serão escritos pelos utilizadores através da escrita arbitrária de código e lógica capaz de transformar o dado de entrada no dado de saída desejado através da execução dinâmica de código. O mecanismo de validação existe para garantir a confiança na escrita da lógica de transformação sendo que qualquer dado de entrada terá sempre o mesmo formato e será sempre transformado da mesma forma.

Uma solução para a criação e processamento arbitrário para transformação de dados é a execução dinâmica de código sendo que pequenos excertos de código desconhecidos do código fonte que são executados em *runtime* (Staicu et al., 2016b; Vasilakis et al., 2021). Esta solução permite a cobertura

completa de qualquer caso porém possui sérios problemas e preocupações de segurança (Staicu et al., 2016b; Vasilakis et al., 2021):

- **Execução de código malicioso**: A funcionalidade de execução dinâmica de código pode permitir a injeção de código malicioso que será executado pelo servidor quando implementada com estratégias de mitigação pobres**;**
- **Acesso a objetos e APIs sensíveis**: Quando código é dinamicamente executado pelo servidor, todos as interfaces e APIs a ele disponíveis também estarão disponíveis ao código dinâmico como interfaces de bases de dados ou até outro métodos;
- **Escalamento de privilégios**: Ao correr um excerto de código de forma dinâmica no servidor qualquer terá os mesmos acessos e privilégios que o próprio servidor. Isto é uma grande preocupação e vulnerabilidade quando implementado sem os devidos padrões de segurança.

A execução dinâmica de código é uma funcionalidade que, por *design*, é extremamente insegura e só deve ser implementada quando todos os seus benefícios forem necessários ao sistema e quando todos os padrões de segurança forem cumpridos.

Com base na pesquisa efetuada os benefícios possíveis que esta abordagem traz são:

- **Flexibilidade**: Ao executar código dinamicamente um sistema poderá executar todo o tipo de excertos de código para qualquer ocasião ou caso de uso sem ser necessária a alteração constante do código fonte;
- **Eficiência**: A abstração trazida por esta abordagem reduz de forma drástica o tamanho do código fonte pelo que é muito mais eficiente apenas guardar em memória todo o código e logica necessários a uma funcionalidade específica;
- **Manutenção**: A encapsulação de métodos complexos e situacionais na forma de componentes mutáveis e reutilizáveis torna a manutenção e futuras atualizações e suportes uma tarefa muito mais fácil, acessível e rápida.

Para a implementação desta abordagem será necessária a escrita de excertos de código sendo que a melhor escolha serão linguagem que não necessitam de ser compiladas e podem simplesmente correr em *runtime* como **JavaScript, Python, Ruby, Groovy, PowerShell** ou **Bash** (Staicu et al., 2016b; Vasilakis et al., 2021). Cada uma destas linguagens de programação está equipada para conseguir executar qualquer excerto codificado nela mesma em formato de String com diversos, porém similares, graus de segurança (Staicu et al., 2016b; Vasilakis et al., 2021).

Qualquer implementação desta técnica que corra diretamente no servidor será sempre uma importante falha de segurança no sistema e uma prática extremamente irresponsável pelo que uma das estratégias de mitigação mais utilizadas é, precisamente, a execução do código fora do servidor.

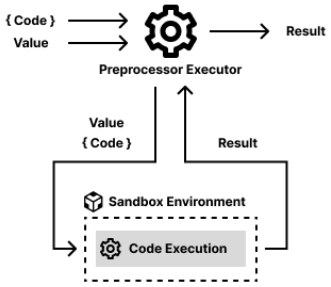

Figura 32 - Conceito Implementação segura de execução dinâmica de código

A Figura 32 representa uma implementação segura para a abordagem de execução dinâmica de código pelo que a execução do código em si acontece dentro de um ambiente de *sandbox* de forma separada e isolada do servidor anulando qualquer acesso que algum código malicioso pudesse ter (Staicu et al., 2016b). Esta técnica utiliza a tecnologia de *sandbox*, que se trata de um ambiente virtual onde aplicações podem ser executadas isoladamente umas das outras e do sistema operativo subjacente não tendo qualquer acesso aos seus recursos (Laurén et al., 2017).

No caso do sistema de interoperabilidade desenvolvido a tecnologia de *sandbox* irá garantir a segurança dos recursos e estado do sistema em si fazendo com que o código que compõe os preprocessadores seja executado separadamente num ambiente *sandbox*. Acima de tudo esta estratégia garante os benefícios da utilização da execução dinâmica de código enquanto reduz ao máximo os seus riscos.

Para além disso a limitação de módulos ou funções da linguagem na qual o script será codificado também é uma excelente adição, em conjunto com a execução em *sandbox*, para a garantia de segurança desta funcionalidade.

```
{
  "preprocessor_name": "multiply-by-1000",
  "preprocessor_type": "hl7",
  "description": "Multiplies the user input by 1000",
  "preprocessor_script": "result = input_data * 1000"
}
```

Figura 33 - Exemplo de um Preprocessador

Como indica a Figura 33, um preprocessador poderá contar com os metadados identificadores dele mesmo como o nome e em que contexto ele deverá ser aplicado, descrição para descrever o seu comportamento e o excerto de código em si. O excerto de código conta com duas interfaces de programação necessárias ao funcionamento correto da lógica de implementação:

- result: O *result* é a variável que irá assumir o valor final do valor processado.

- Input_data: É a variável que assume o valor de entrada extraído dos recursos de interoperabilidade. O fluxo para um preprocessador será sempre transformar o ***input_data*** num ***result***.

Sabendo isto, no caso apresentado na Figura 33 consiste num preprocessador chamado "multiply-by-1000", a ser utilizado no contexto de processamento de recursos HL7, completo com uma descrição que diz que o dado de entrada será multiplicado por 1000 e um excerto de código que traduz exatamente este comportamento atribuindo a multiplicação do **input_data** ao **result**.

5.3 Arquitetura Concetual do Sistema

O sistema foi concebido com uma arquitetura de microserviços em mente onde cada um deles é uma implementação autónoma de software e desempenha funções muito específicas de forma isolada (Di Francesco et al., 2019; Hawilo et al., 2019), porém comunicam entre si através de pedidos HTTP para poder formar um sistema completo. Esta abordagem modular permite que cada parte do sistema seja desenvolvida, atualizada e implementada isoladamente do resto do sistema aumentando a complexidade do mesmo, mas facilitando tanto o processo de desenvolvimento como o de suporte (Di Francesco et al., 2019).

A arquitetura proposta consiste em três microserviços, uma base de dados e uma instância Mirth Connect que formam o funcionamento interno do sistema, um serviço de interface de utilizador e um gestor de mensagens que servirá de intermediário entre o sistema de interoperabilidade e sistema ABI. O funcionamento da arquitetura pode ser observado na seguinte Figura 34.

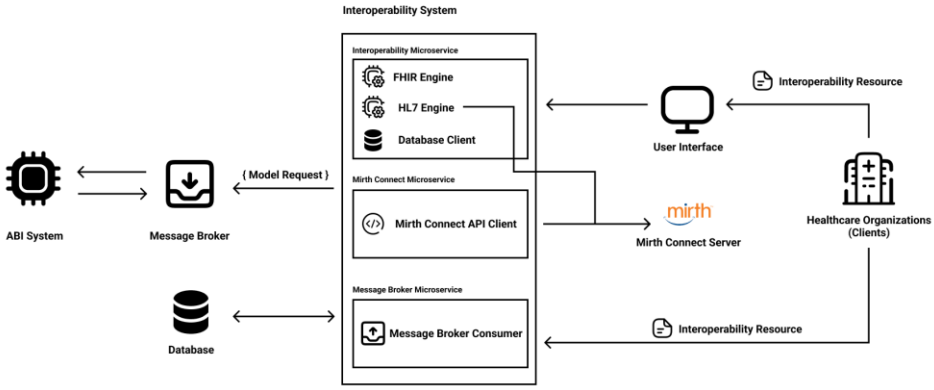

Figura 34 - Arquitetura Concetual do Sistema de Interoperabilidade

Como é observável na representação arquitetónica do sistema na Figura 34, o sistema é composto por três microserviços individuais que realizam funções específicas do seu âmbito de forma isolada e que operam com as tecnologias de base de dados, gestor de mensagens e Mirth Connect bem como contém toda a informação necessária para tal. Esta arquitetura também foi desenhada e pensada com a capacidade de integração de múltiplas combinações de tecnologias em mente para bases de dados e gestores de mensagens. A utilização do sistema será feita de duas formas, pela interface de utilizador que abstrai ao máximo o funcionamento interno do mesmo ou pelos *endpoints* REST disponíveis ao utilizador sendo que isto torna possível e fácil a integração do sistema em ambientes ou sistemas já existentes.

O sistema é composto por três microserviços:

- Microserviço de Interoperabilidade: Este microserviço contém todos os segredos e credenciais para interação com a base de dados do sistema bem como toda a lógica de interoperabilidade e *backend*;
- Microserviço Mirth Connect: Contém todos os segredos e credenciais para a conexão e interface com a plataforma Mirth Connect. Este microserviço fará uso da REST API do Mirth para poder fazer a gestão e interação de todas as funcionalidades necessárias do mesmo;
- Microserviço de Gestor de Mensagens: Contém todos os segredos e credenciais para a conexão e interface com um sistema ou tecnologia de gestão de mensagens.

5.4 Requisitos do Sistema

Este sistema de interoperação, para satisfazer as necessidades e oportunidades detetadas, necessita de uma lista bem definida e estruturada de requisitos para direcionar o desenvolvimento do projeto, mas também para servir como um indicador de progresso.

É importante desenvolver e conceber estes requisitos para assegurar que o processo de desenvolvimento é devidamente direcionado, corre sem grandes problemas e que o resultado não será pobre ou incompleto (Daun et al., 2023).

Os requisitos para um sistema podem e devem ser divididos em duas categorias, **funcionais** e **não-funcionais**.

- **Requisitos Funcionais**: Estes requisitos descrevem o comportamento específico de um sistema, o que este deve fazer, de que forma e como deve responder aos *inputs* de um dado

utilizador. São, geralmente, escritos de uma forma clara e concisa através de uma linguagem não ambígua (Daun et al., 2023);

- **Requisitos não-funcionais**: Descrevem as qualidades e características gerais de um dado sistema, definem a forma como o sistema se deve comportar no que toca à sua velocidade, confiabilidade e segurança. Devido à sua natureza mais subjetiva são consideravelmente mais difíceis de definir e medir (Daun et al., 2023).

5.4.1 Requisitos Funcionais

Para o sistema proposto foram apontados os requisitos funcionais observáveis na seguinte Tabela 3 e não funcionais na Tabela 4.

Tabela 3 - Lista de Requisitos Funcionais do Sistema de Interoperabilidade

Requisito	Descrição	Importância	Estado
Sistema é capaz de fazer a Gestão da Representação de Modelos	O sistema terá de ser capaz de criar, ler, atualizar e apagar representações dos Modelos que correm no sistema ABI	10	Completo
Sistema é capaz de fazer a Gestão de Clientes	O sistema terá de ser capaz de criar, ler, atualizar e apagar contas e credenciais de clientes	10	Completo
Sistema é capaz de extrair informação arbitrariamente de recursos de Interoperabilidade	O sistema terá de ser capaz de, de forma arbitrária, extrair qualquer pedaço de informação de um recurso de interoperabilidade (HL7 e FHIR)	10	Completo
Sistema é capaz de fazer a gestão de mapeamentos de interoperabilidade	O sistema terá de ser capaz de criar, ler, atualizar e apagar qualquer mapeamento de Interoperabilidade (HL7 e FHIR)	10	Completo
Sistema é capaz de fazer a gestão de validadores	O sistema terá de ser capaz de criar, ler, atualizar e apagar validadores	10	Completo
Sistema é capaz de fazer a gestão de preprocessadores	O sistema terá de ser capaz de criar, ler, atualizar e apagar **preprocessadores**	10	Completo
Sistema é capaz de processar um pedido com um recurso	O sistema terá de ser capaz de receber um recurso e fazer a extração, validação e preprocessamento do mesmo conforme o tipo de recurso e modelo escolhidos	10	Completo
Sistema é capaz de processar um pedido com múltiplos recursos	O sistema terá de ser capaz de processar um pedido mas extrair a informação necessária de múltiplos recursos	10	Completo
Sistema é capaz de diferenciar pedidos de acordo com o tipo de recurso	O sistema terá de ser capaz de fazer a distinção entre recursos HL7 e FHIR para poder efetuar a extração, validação e preprocessamento em conformidade	10	Completo
Sistema é capaz de ler e interpretar a documentação dos padrões de Interoperabilidade	O sistema terá de ser capaz de ler e interpretar a documentação dos padrões de Interoperabilidade (HL7/FHIR) para poder dar ao utilizador a opção de criar mapeamentos	10	Completo
Sistema é capaz de interoperar com outros sistemas	O sistema terá de ser capaz de comunicar e interoperar com outros sistemas	10	Completo

CI/CD de todos os serviços	Todos os serviços que compõe o sistema terão de estar devidamente configurados com um gestor de versão (git) e uma versão nova deverá ser lançada aquando de alterações na *branch* principal.	10	Completo
Sistema é utilizável tanto pela interface gráfica como programaticamente como API	O sistema tem de ser utilizável tanto pela interface gráfica como programaticamente através de pedidos HTTP	8	Completo
Sistema é capaz de atribuição arbitrária de mapeamentos a clientes	O sistema terá de ser capaz de fazer a distinção de clientes aquando do processamento de um pedido. Clientes diferentes poderão ter mapeamentos HL7/FHIR diferentes customizados para si mesmos	8	Completo
Sistema possui um mecanismo de autenticação seguro	O sistema terá um mecanismo seguro de autenticação que segue as melhores práticas da programação e com distinção de cargos de contas (administrador e cliente)	8	Completo
Sistema possui um mecanismo de permissões entre os clientes e os modelos	O sistema terá de possuir um mecanismo que previne os utilizadores de utilizar um modelo ao qual não têm acesso	8	Completo
Sistema é totalmente configurável	O sistema terá de ser totalmente configurável através de um ficheiro singular de configuração sendo que todos os segredos internos estão apenas disponíveis ao administrador do sistema que o configurou	7	Completo
Sistema possui uma interface fácil de navegar	O sistema terá de ter uma interface visual fácil de navegar e que permita ao utilizar tomar proveito e controlar todo o sistema	5	Completo
Configurador de sistema automático	Devido à complexidade da configuração do sistema, uma interface capaz de gerar o ficheiro de configuração automaticamente será uma grande ajuda na facilidade de implementação e na diminuição dos erros	4	Completo
Sistema possui um sistema de comunicação de *tickets* entre cliente e administrador	O sistema terá de possuir um sistema de *tickets* para o cliente poder comunicar de forma clara com o administrador para pedir permissões, novos mapeamentos ou reportar *bugs*	3	Completo
Suporte a múltiplas tecnologias de Bases de Dados	O sistema terá a capacidade de ligação a diferentes tipos de bases de dados	2	Trabalho Futuro
Suporte a múltiplas tecnologias de Gestão de Mensagens	O sistema terá a capacidade de ligação a diferentes tipos de gestores de mensagens	2	Trabalho Futuro
Suporte de mais tipos de recursos de interoperabilidade	O sistema terá a capacidade de suportar e processar mais tipos de recursos à parte de HL7 e FHIR	2	Trabalho Futuro
Suporte de mais formatos FHIR	O sistema será capaz de suportar mais formatos de recursos FHIR	2	Trabalho Futuro

5.4.2 Requisitos Não Funcionais

De forma a seguir as melhores práticas e objetivos do desenvolvimento moderno de software foram apontados alguns requisitos não funcionais para o sistema de interoperabilidade. Estes requisitos estão

apresentados na Tabela 4 acompanhados de uma breve descrição, grau de importância no ecossistema de software moderno e o seu estado de realização.

Tabela 4 - Lista de Requisitos Não-Funcionais do Sistema de Interoperabilidade

Requisito	Descrição	Importância	Estado
Sistema Modular	O sistema tem de ser composto por múltiplos serviços especializados e responsáveis por cada parte do mesmo.	10	Completo
Sistema Rápido	O sistema deverá ser rápido e todas as funcionalidades devem estar otimizadas para ocuparem o menor tempo e recursos	10	Completo
Sistema Robusto	O sistema deverá ser robusto e nunca deve parar a sua execução, todos os erros devem ser tratados e devidamente arquivados mas isto nunca deve interferir no funcionamento base do sistema	10	Completo
Sistema Seguro	O sistema deverá ser seguro em todas as suas funcionalidades e seguir as melhores práticas de programação em cada contexto de funcionalidade	10	Completo
Sistema Distribuído	O sistema deverá funcionar de forma distribuída na medida que todos os serviços são modulares e podem ser hospedados em máquinas e ambientes diferentes	8	Completo
Serviços Documentados	Os serviços que compõe o sistema deverão estar devidamente documentados e *controllers* expostos através de um ficheiro **swagger**	8	Completo

6. DESENVOLVIMENTO DO SISTEMA

Este tópico irá abordar todo o planeamento e desenvolvimento realizados para a concretização do sistema de interoperabilidade, técnicas e estratégias utilizadas bem como o seu funcionamento e atualização/suporte de uma forma mais aprofundada.

O resultado desta fase de desenvolvimento foi um sistema modular de microserviços com o potencial de estabelecer interfaces com diferentes sistemas de bases de dados e de gestão de mensagens e capaz de extrair informação de recursos de interoperabilidade, validar essa informação e processá-la de acordo com os desejos e requisitos do operador do sistema de uma forma dinâmica e ininterrupta. Qualquer parte deste sistema pode ser executada ou configurada para correr em máquinas ou ambientes diferentes e está preparada para ser continuamente atualizada e lançada.

Todo o sistema foi desenvolvido e desenhado para ser altamente configurável pelo operador de sistema sem nunca descorar ou comprometer a segurança e melhores práticas de programação e

desenvolvimento tendo em conta as vulnerabilidades mais comuns e perigosas reportadas pela organização OWASP.

6.1 Base de Dados

O sistema de interoperabilidade tem como objetivo o suporte de vários tipos de bases de dados sendo que o fluxo de dados teve de ser desenvolvido de forma que consiga ser adotado por bases de dados relacionais e não relacionais.

O modelo de dados que irá permitir o funcionamento do sistema consistirá de:

- **Modelos de ABI**: Estes serão os dados representativos dos modelos preditivos e de otimização que fazem parte do sistema ABI a integrar sendo que a granularidade do foco do sistema serão os atributos destes modelos sendo que a maioria das funcionalidades do sistema terá por base este par de modelo-atributo;
- **Administradores do Sistema**: Credenciais pertencentes aos administradores do sistema. É uma entidade de dados extremamente comum na conceção de sistemas informáticos;
- **Clientes**: Utilizadores do sistema podem aceder aos diferentes modelos de acordo com uma permissão e também podem ter mapeamentos específicos para si mesmos;
- **Mapeamentos**: Informação que permite a extração arbitrária de informação de recursos de interoperabilidade;
- **Validadores**: Informação que permite a validação de um valor;
- **Preprocessadores**: Informação que permite o preprocessamento e transformação de um valor;
- **Pedidos**: Dados históricos de um processo completo de um ou mais recursos de interoperabilidade para um modelo;
- **Canais Mirth Connect**: Informação que permite a integração de canais mirth connect no sistema;
- **Questões ou *issues*:** Conversas entre cliente e administrador para solicitar ajuda, permissões ou mapeamentos específicos.

Para o uso com bases de dados relacionais foi elaborado um modelo relacional que conta com 15 tabelas interligadas, otimizadas para consulta eficiente e reutilização de dados. Este modelo foi também normalizado até à 3ª forma normal (3NF) (Kolahi & Libkin, 2006) reduzindo assim a redundância e aumentando a eficiência do sistema.

Foi também desenvolvido um modelo não relacional para uso com bases de dados não relacionais que estrutura os dados de forma mais simples e direta. Neste modelo algumas tabelas foram simplesmente substituídas por atributos de coleções pai, reduzindo o uso de chaves estrangeiras entre coleções, facilitando a inserção e leitura de dados.

Durante o desenvolvimento prático da solução apenas foi criado suporte para bases de dados relacionais pelo que o suporte dos dois tipos não traria mais valor à solução apresentada no presente documento, porém esta tarefa encontra-se destacada como trabalho futuro.

6.2 Arquitetura Tecnológica

Com base na arquitetura concetual, requisitos e de funcionamento o sistema foi desenvolvido na sua totalidade permitindo a modelação completa com o detalhe e foco não só no seu funcionamento e fluxo de dados, mas também das tecnologias implementadas.

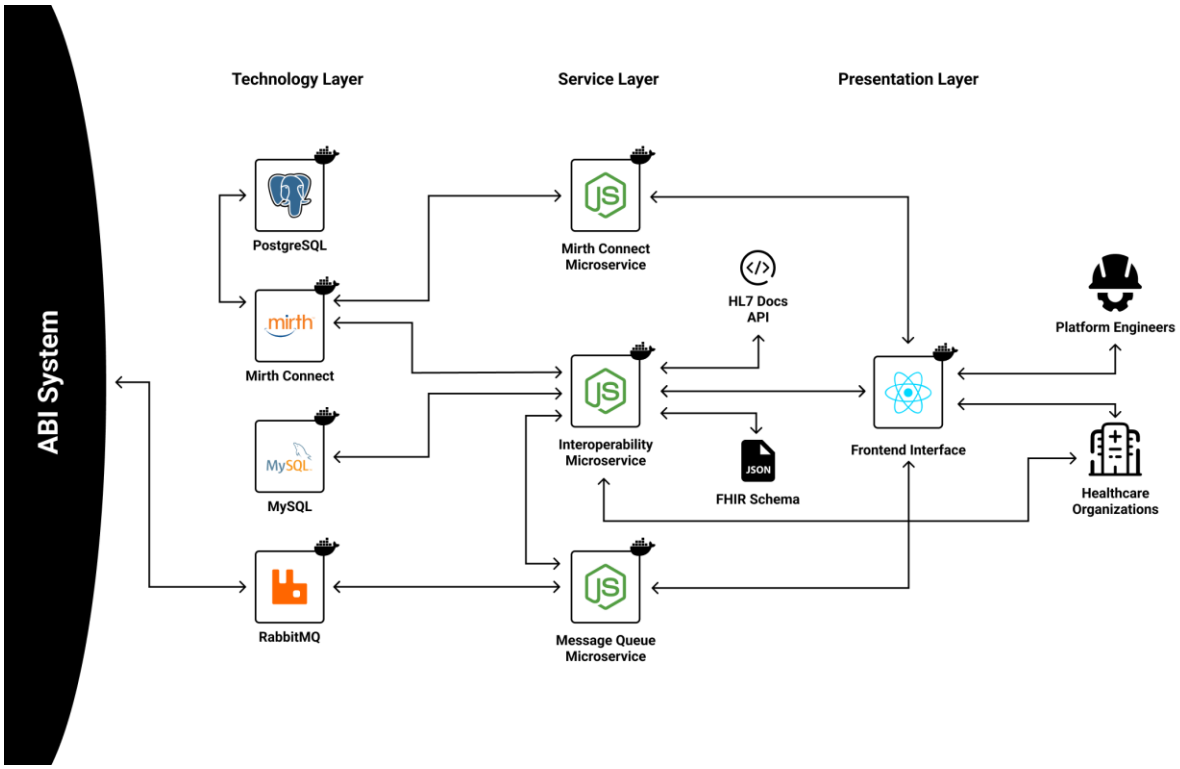

Figura 35 - Arquitetura Tecnológica do sistema de interoperabilidade

Como é possível verificar na Figura 35 representada acima, a arquitetura final do sistema é muito semelhante à sua arquitetura concetual pelo que conta com uma arquitetura de microserviços baseada em três microserviços principais, uma aplicação de interface gráfica, quatro tecnologias para o armazenamento de dados e intercomunicação e dois utilizadores alvo. Todos os componentes do sistema estão preparados para serem executados através de *containers* Docker ou melhor uma configuração

multi-container o que permite a distribuição completa do sistema por várias máquinas ou ambientes. Para uma documentação mais detalhada e facilidade de compreensão a arquitetura foi dividida em três grandes camadas distintas, de acordo com a funcionalidade dos componentes que as constituem, Tecnologias, Serviços e Apresentação.

- **Camada Tecnológica**: Esta camada agrega toda a tecnologia na forma de serviços que é necessária ao funcionamento do sistema como o RabbitMQ para intercomunicação com sistemas externos, o MySQL para a base de dados do sistema, o Mirth Connect para o manuseamento de recursos HL7 e o PostgreSQL como base de dados para o Mirth Connect. O sistema foi concebido para que cada uma destas tecnologias não seja obrigatória e intercambiável com mais tecnologias de bases de dados e gestão de mensagens. O instanciamento destas tecnologias também não necessita de ser feito no mesmo ambiente que o resto do sistema pelo que este foi concebido para poder lhes poder aceder de forma segura em qualquer ambiente desde que esta ligação seja devidamente configurada;
- **Camada de Serviços**: Esta camada consiste em toda a lógica e é responsável por todo o funcionamento e comportamento do sistema. Foi dividida em três microserviços pelo que cada um deles é restrito a um âmbito muito específico e apenas têm acesso às credenciais necessárias para as funções relacionadas com o mesmo.
    - **Microserviço de Message Queue:** Este microserviço é responsável pela interface e interação com a tecnologia de gestão de mensagens (RabbitMQ) de forma a propagar os pedidos recebidos e processados pelo sistema para sistemas externos, neste caso o sistema ABI, e de receber, tratar e propagar todas as respostas que sejam passadas para o RabbitMQ de volta para o sistema. Toda a informação e segredos para a conexão ao RabbitMQ estão restritos ao âmbito deste microserviço e são altamente configuráveis de forma a permitir a ligação e interface com instâncias RabbitMQ que se encontrem fora do mesmo ambiente;
    - **Microserviço de Mirth Connect:** Este microserviço é responsável pela interface e manuseamento da REST API da plataforma de integração Mirth Connect de forma a permitir e garantir que o sistema consiga realizar as operações que dela dependam. Toda a informação e segredos para a conexão ao Mirth Connect estão restritos ao âmbito deste microserviço e são altamente configuráveis de forma a permitir a ligação e interface com instâncias Mirth Connect que se encontrem fora do mesmo ambiente;

- **Microserviço de Interoperabilidade:** Trata-se do microserviço central no sistema pelo que é este que contém toda a lógica de interoperabilidade e utiliza as funcionalidades disponibilizadas pelos restantes microserviços para realizar todas as funções do sistema. É responsável pela interface e manuseamento da base de dados (MySQL) que é o coração do fluxo de dados e funcionamento geral do sistema devido às estratégias de design de software implementadas. Este microserviço constitui o Backend principal do sistema sendo que as funcionalidades mais importantes estão nele centradas e possui toda a informação e segredos para a conexão à Base de Dados restritos a seu âmbito sendo estas altamente configuráveis para possibilitar a conexão a bases de dados que se encontrem fora do mesmo ambiente.

- **Camada de Apresentação**: Esta camada constitui a componente de interação direta com os utilizadores finais. Qualquer que seja o utilizador a abstração do resto do sistema dá-se através desta camada. Consiste em apenas uma instância de uma aplicação de interface gráfica web baseada em React que permite ao utilizador a utilização abstraída de todos os microserviços como um sistema completo de forma intuitiva, rápida e segura (Rawat & Mahajan, 2020).

A natureza modular e desapegada do conceito de *container* e microserviço permite que estes sejam implementados de forma direta e fácil em qualquer tipo de ambiente e o suporte contínuo, atualizações, melhorias e monitorização passam a ser muito mais direcionados e autónomos ao seu respetivo âmbito quer seja relativo à base de dados, à gestão de mensagens ou à integração Mirth Connect.

Outro aspeto característico deste tipo de arquitetura de software é a facilidade de implementação de novas funcionalidades por meio da atualização e incremento dos serviços em si como também pela integração de novas tecnologias e plataformas de apoio ao sistema como:

- **Monitorização**: A monitorização de um sistema é algo muito importante pois qualquer erro é percetível e perseguível pelas partes responsáveis pela manutenção do sistema. Ferramentas como o **Portainer** tornam esta tarefa trivial pelo que permitem o acesso a todos os registos de todas as partes do sistema;
- **Controlo de Tráfego**: Qualquer sistema informático necessita de lidar com a quantidade de tráfego que por ele passa quer ela seja grande ou pequena e soluções como o **Traefic** ou **HaProxy** tornam esta tarefa algo trivial e de implementação direta de forma que nenhum dos serviços seja sobrecarregado e o sistema em si possua alta disponibilidade.

6.3 Padrões de Arquitetura de Microserviços

O sistema de interoperabilidade faz uso de vários padrões arquitetónicos para sistemas baseados em microserviços. Devido à sua natureza de simplificação dos sistemas ABI, o sistema de interoperabilidade terá de ser o mais simples possível enquanto garante um bom equilíbrio entre desempenho e custo. Sendo assim este sistema apresenta uma estrutura que faz uso de todos os grupos arquitetónicos estudados na literatura para minimizar os custos de implementação e maximizar a eficiência na sua execução.

Como padrão de coordenação é utilizado o API Gateway, que é o padrão recomendado por Davide Taibi e Valentina Lenarduzzi no seu estudo, devido à sua facilidade de integração e alta eficiência. A descoberta de serviços foi descartada optando-se pela configuração dos diferentes serviços com os endereços exatos dos restantes de forma direta. Esta abordagem é a mais simples e direta, porém torna impossível o dinamismo entre soluções sendo sempre necessária a reimplementação aquando de mudanças no endereço de um serviço.

O padrão de implementação seguido foi o de um único serviço por *host* sendo que cada microserviço deverá ser executado no seu respetivo ambiente isolado na forma de *containers*.

O padrão de armazenamento de dados escolhido foi o de base de dados dedicada por cada microserviço sendo que deverá existir uma base de dados por cada microserviço que necessite. Devido à arquitetura particular deste sistema apenas um serviço faz uso desta base de dados sendo que os restantes estão limitados aos seus respetivos recursos.

Os padrões escolhidos para o sistema de interoperabilidade tornam o sistema eficiente e robusto fazendo uso das características mais únicas de um sistema baseado em microserviços.

6.4 Modelo/Template De Microserviço

Uma vez que o sistema conta com três microserviços desenvolvidos de forma idêntica, porém com âmbitos distintos torna-se necessária, seguindo as melhores práticas de desenvolvimento de software, a definição de um *template* ou modelo que servirá como base para o desenvolvimento dos microserviços em si para manter a consistência, normas de documentação e segurança por todos eles.

O *template* foi concebido tendo por base as características chaves na conceção de um microserviço:

- **Definição do protocolo de comunicação:** Na conceção de um sistema composto por microserviços é importante a identificação de quantos/quais microserviços serão precisos e de que forma irão comunicar (Gonzalez, 2016);

- **Autonomia:** Cada microserviço deve ser um componente autónomo possível desenvolvimento, implementação e gestão de forma independente (Gonzalez, 2016).
- **Implementáveis:** É importante considerar de que forma e onde eles serão implementados para produção (Gonzalez, 2016);
- **Microserviços pequenos e focados:** Manter o âmbito de cada microserviço o mais pequeno possível e focado de forma a facilitar o seu desenvolvimento, teste e manutenção (Gonzalez, 2016).
- **Tolerância a falhas:** Os microserviços devem ser desenvolvidos e implementados no sistema de forma que se por acaso algum deles falhe o sistema não para de correr através das melhores práticas de programação que tomam por garantido que toda a comunicação pode e vai eventualmente falhar (Gonzalez, 2016).

6.4.1 Node.js

Para este efeito o *template* foi desenvolvido utilizando **Node.js**, mais concretamente a framework de desenvolvimento de microserviços na forma de REST APIs **Express.js**. Esta foi a tecnologia escolhida para o *template* que irá dar origem à lógica completa de Backend do sistema pelas suas várias características apelativas neste ambiente:

- **Leve e escalável**: O Node.js é um ambiente de *runtime* muito leve e facilmente escalável muito adequado para o conceito de microserviço. Apesar de ser apenas *single-thread* (só consegue executar uma tarefa ou processo de cada vez de forma sequencial) a sua arquitetura orientada a eventos permite o tratamento e processamento de vários pedidos ou tarefas em simultâneo. Esta é a característica que tornam os microserviços desenvolvidos com Node.js altamente escaláveis (Gonzalez, 2016);
- **Programação assíncrona**: O Node.js é está muito bem preparado para programação assíncrona que é algo essencial ao desenvolvimento de microserviços uma vez que estes necessitam de comunicar quer entre si quer com serviços externos ao sistema sem estarem bloqueados (Gonzalez, 2016);
- **Enorme Comunidade e Ecossistema**: O Node.js possui uma comunidade muito grande e muito ativa pelo que existe uma enorme variedade de bibliotecas e estruturas já disponíveis para facilitar o desenvolvimento de microserviços (Gonzalez, 2016);

- **Desempenho**: O Node.js é um ambiente de interpretação e execução de código extremamente rápido e eficaz o que o torna ideal para o desenvolvimento de microserviços e o tratamento de grandes volumes de dados se necessário (Gonzalez, 2016);
- **Flexibilidade**: O Node.js é muito flexível permitindo praticamente uma infinidade de formas diferentes de desenvolvimento de microserviços e pode ser utilizado ou implementado numa variedade de plataformas já preparadas para execução de microserviços desenvolvidos com Node (Gonzalez, 2016);
- **Fácil Implementação**: Após o microserviço estar desenvolvido é muito fácil implementá-lo se foi desenvolvido com Node.js uma vez que praticamente todos os serviços de hospedagem e implantação possuem suporte direto ao Node (Gonzalez, 2016);
- **Experiência de Desenvolvimento**: Por último a experiência de desenvolvimento com Node.js é muito agradável e amigável sendo que é muito fácil aprender e aplicar novos conceitos e bibliotecas devido à natureza intérprete do próprio *runtime* ao invés de compilação e à grande comunidade de desenvolvedores que constantemente disponibilizam novas ferramentas e prestam ajuda (Gonzalez, 2016).

Pelas razões acima apresentadas o Node.js é, justificavelmente, a opção mais popular no que toca a desenvolvimento de microserviços e não só, praticamente todo o desenvolvimento de aplicações e sistemas baseados em web.

6.4.2 Padrão MVC

Sendo cada um dos microserviços apenas parte de um sistema completo a implementação do padrão MVC será adaptada conforme o âmbito e responsabilidade de cada um deles sendo que a camada de View será responsabilidade completa da aplicação de interface gráfica de Frontend ficando apenas as camadas de Controller e Model ao encargo dos microserviços de *Backend*. A distribuição dos diversos componentes do sistema pelas camadas do padrão MVC pode ser observada na Tabela 5.

Tabela 5 - Distribuição dos componentes do sistema pelo padrão MVC

Componente	**Camadas MVC**
Microserviço de Interoperabilidade	*Model* e *Controller*
Microserviço de Gestão de Mensagens	*Controller*
Microserviço Mirth Connect	*Controller*
Web Application	*View*

6.4.3 Estrutura do Modelo/Template

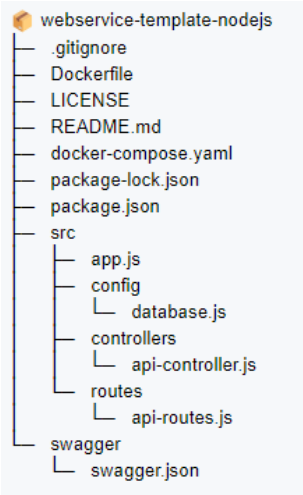

Figura 36 - Estrutura de Ficheiros de Template de Microserviços

Como indica na Figura 36 o *template* para a base de todos os microserviços conta com:

- **Documentação** – Inclui ficheiros que fornecem informação essenciais sobre o projeto e os termos de uso do mesmo;
- **Virtualização** – Ficheiros de configuração da tecnologias de virtualização de software **Docker**;
- **Gestão de dependências** – Ficheiros de gestão de dependências Node.js;
- **Estrutura do código fonte** – Proporciona uma estrutura bem definida do código fonte que facilita o desenvolvimento do microserviço em rotas e controladores.
- **Middlewares** – Estrutura de *middleware* para implementação fácil de regras de negócio;
- **Ligação à base de dados** – Estrutura e mecanismo de ligação a bases de dados caso seja necessário;
- **Especificação da API** – Definições pré-estruturadas para facilitar a documentação de REST API

Com este *template* base definido os microserviços podem ser desenvolvidos com confiança que as melhores práticas do desenvolvimento de microserviços e de estruturação de aplicações baseadas em Node.js estão empregues dando origem a microserviços rápidos, eficazes e escaláveis de fácil manutenção, atualização e implementação.

6.5 Microserviço de Interoperabilidade

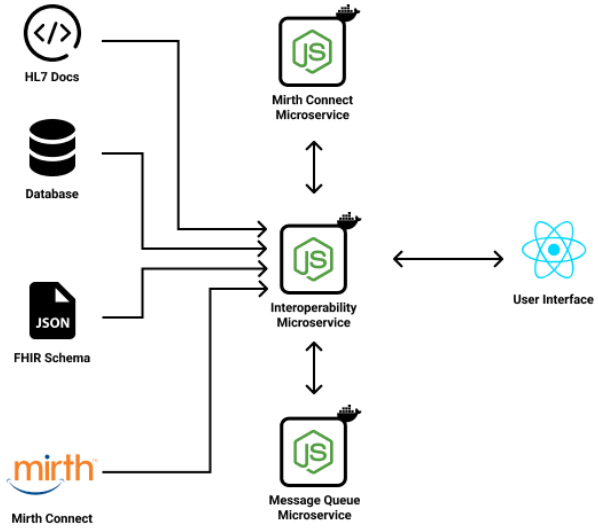

Figura 37 - Arquitetura Microserviço de Interoperabilidade

O microserviço de interoperabilidade é o microserviço central do sistema e consiste no ponto de entrada do mesmo pelo que comunica com os restantes microserviços para fazer uso de todo o sistema e das suas funcionalidades. Como está representado na Figura 37 consiste, praticamente, no Backend do sistema propriamente dito pelo que faz interface com a base de dados, disponibiliza todos os dados necessários ao Frontend, possui um esquema de dados para consulta do padrão FHIR, ligação a uma REST API de documentação HL7, comunicação com canais Mirth Connect assim como toda a lógica de gestão de mapeamentos, validadores e preprocessadores necessários ao processamento de recursos de interoperabilidade bem como os recursos e funcionalidades para esse fim.

O microserviço de interoperabilidade foi desenvolvido com base no *template* definido e descrito acima sendo que é baseado em Node.js e segue o padrão MVC (Majeed & Rauf, 2018). Seguindo o padrão MVC este microserviço foca-se nas componentes de Model e Controller pelo que contém todos os esquemas e modelos de dados para interface com a base de dados e todos os controladores que tornam possível todas as funcionalidades do sistema de interoperabilidade.

À semelhança de todos os outros microserviços o de interoperabilidade é, também, totalmente configurável através de um conjunto de variáveis de ambiente que irão ditar o seu comportamento e conexões. Estas configurações passam pela porta onde o serviço irá correr, credenciais de acesso à base de dados bem como os endereços completos onde os restantes microserviços estarão a correr.

Este microserviço consiste numa REST API (*Representational State Transfer*) (Doglio, 2018) onde todas as funcionalidades do sistema são acessíveis e executáveis através de pedidos HTTP diretamente para o serviço sendo que a comunicação com os restantes microserviços também realizada através deste protocolo.

As funcionalidades do microserviço de interoperabilidade resumem-se a:

- **Gestão de modelos**: Controlo sobre a gestão dos modelos preditivos e otimização presentes no sistema, o seu estado de implementação, os seus suportes de interoperabilidade e dos seus atributos;
- **Gestão de Configurações**: Controlo sobre a gestão dos mapeamentos, validadores e preprocessadores respetivos a cada atributo respetivo a um modelo;
- **Gestão de Validadores Gerais**: Controlo sobre a gestão dos validadores gerais do sistema,
- **Gestão de Preprocessadores Gerais**: Controlo sobre a gestão dos preprocessadores gerais do sistema;
- **Gestão de Mapeamentos de Atributos:** Controlo sobre a gestão dos pares chave-valor para processamento de variáveis categóricas;
- **Gestão de Clientes**: Controlo sobre a gestão de clientes, das suas credenciais e das suas permissões para acesso aos modelos;
- **Mecanismos de Interoperabilidade e processamento de recursos**: Extração, validação e processamento de dados de recursos de interoperabilidade;
- **Gestão de Pedidos de clientes:** Controlo sobre a gestão de pedidos de clientes a diferentes modelos assim como os dados extraídos e processados que contam no pedido;
- **Gestão de Administradores e Autenticação**: Gestão das credenciais de administradores e lógica de autenticação do sistema segura por JWTs devidamente assinados;
- **Integração de canais Mirth Connect**: Funcionalidades de integração de modelos Mirth Connect na forma do armazenamento e validação de canais de Mirth Connect através do elemento identificador do mesmo;
- **Gestão de *Issues***: Controlo sobre a gestão dos *issues* de clientes e o seu estado bem como as suas mensagens respetivas;
- **Consulta de esquemas de interoperabilidade**: Funcionalidade de *parsing* e consulta dos esquemas de recursos de interoperabilidade HL7 e FHIR diferenciados por versão;
- **População automática da Base de Dados**: Funcionalidades de população automática da base de dados com os dados iniciais necessários ao funcionamento do sistema;
- **Compilação de Informação para o Frontend**: Funcionalidades de compilação de informação para apresentação na camada de visão ou aplicação de Frontend;
- **Interação com os restantes microserviços**: Funcionalidades de interação com os restantes microserviços de interface com o gestor de mensagens e Mirth Connect.

6.5.1 Interface com Base de Dados

O microserviço de Interoperabilidade necessita de ter acesso e de estabelecer uma interface com a base de dados do sistema de forma a poder persistir, guardar, consultar, alterar e operar sobre os mesmos de forma a permitir todas as funcionalidades que fazem parte do sistema de interoperabilidade. A forma mais fácil e inata para estabelecer esta interface é, precisamente, o uso das próprias APIs disponibilizadas pelas diferentes tecnologias de bases de dados através das suas respetivas linguagens ou notações de consulta como por exemplo expressões SQL para bases de dados relacionais. Esta abordagem garante a capacidade de implementação de 100% de todas as funcionalidades disponibilizadas pelas APIs assim como o controlo completo sobre a interface e troca de dados, mas também torna muito fácil qualquer lapso ou descuido na sua implementação segura. Sendo que, segundo a OWASP (*Open Worldwide Application Security Project*), a injeção de dados maliciosos em bases de dados consiste numa das mais comuns e mais perigosas vulnerabilidades em sistemas baseados em web (Bach-Nutman, 2020b) outras soluções foram exploradas como ORMs (*Object Relational Mapping*).

O *Sequelize* foi escolhido como ORM para este microserviço devido à sua popularidade com Node.js e compatibilidade com várias bases de dados relacionais, como MySQL, PostgreSQL e Microsoft SQL Server. Oferece uma camada de abstração segura para operações SQL, minimizando riscos de segurança, como injeções (Bach-Nutman, 2020b), além de facilitar a manutenção e evolução do modelo de dados. O *Sequelize* permite migrações, validações nativas, cache de pesquisas frequentes, suporte a transações e *hooks* para automação.

De momento apenas bases de dados relacionais estão suportadas através do Sequelize no microserviço de interoperabilidade sendo que apenas o MySQL está suportado, porém o desenvolvimento deste microserviço teve em consideração a possibilidade de suporte de outras bases de dados ou de bases de dados NoSQL sendo que está preparado para o desenvolvimento de trabalho futuro.

6.5.2 Encriptação de Dados

Com base na revisão bibliográfica, o algoritmo de escolha para a encriptação de dados sensíveis no sistema de interoperabilidade foi o Bcrypt. Como já foi mencionado o Bcrypt é um algoritmo de encriptação de dados desenvolvido para ser intencionalmente dispendioso que utiliza uma combinação de funções de derivação de chaves, cifras de bloco e sais. A função de derivação de chave é utilizada para gerar uma chave encriptada a partir da informação real e de um sal. A cifra de bloco é, então, utilizada para encriptar a chave. O sal é um valor aleatório que é adicionado à informação real antes de

esta ser transformada numa chave encriptada sendo que cada pedaço de informação resulte numa chave encriptada diferente mesmo quando são exatamente iguais.

No microserviço de interoperabilidade a encriptação de dados sensíveis pelo algoritmo Bcrypt segue o seguinte fluxo:

- Um sal gerado de forma aleatória;
- O sal é anexado à informação ou dado sensível a encriptar;
- A informação anexada com o sal é transformada numa *hash* através de uma função de derivação de chaves;
- A *hash* gerada é, então, encriptada através de uma cifra de bloco;
- A chave encriptada é guardada na base de dados sendo que apenas o microserviço de interoperabilidade tem a capacidade de a ler ou desencriptar;

Desta forma toda a informação sensível no sistema de interoperabilidade está armazenada de forma ilegível e segura na base de dados apresentando uma estrutura como a que está representada na Figura 38.

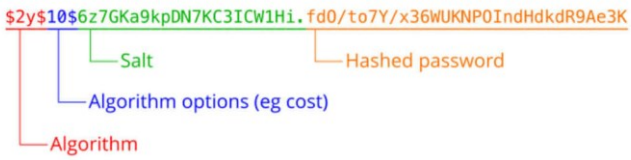

Figura 38 - Algoritmo de encriptação Bcrypt

6.5.3 Pré-População da Base de Dados

Tendo em conta a natureza modular do funcionamento do mecanismo de interoperabilidade e de forma a que não haja qualquer configuração inicial por parte do utilizador o microserviço de interoperabilidade, automaticamente, cria algumas entradas na base de dados caso estas não existam quando é lançado. Desta forma quer seja a primeira instalação quer estas entradas tenham sido apagadas, elas serão sempre verificadas e criadas. Estas entradas consistem na configuração base do sistema sendo que nas credenciais da conta de administrador e nos validadores e preprocessadores predefinidos que servem para dar um suporte geral à maior parte dos casos de validação e preprocessamento sem que o utilizador tenha de criar os seus próprios personalizados. Visto que ainda são bastantes validações e *scripts* de preprocessamento este passo poupa ao utilizador muito tempo e fornece as mesmas funcionalidades

em todas instalações. Para além do mais a forma como este sistema de pré-população está implementado permite a sua atualização contínua sendo que novas versões do microserviço de interoperabilidade podem e irão constar de mais e melhores validadores e preprocessadores predefinidos.

6.5.4 Autenticação

A autenticação é uma parte extremamente importante no desenvolvimento de software na medida em que todas as operações que requerem algum tipo de acesso, permissões ou qualquer atividade que implique a gestão de contas ou mais do que um utilizador, requerem algum tipo de autenticação de forma a que o sistema identifique o utilizador que esteja a realizar o pedido. Imediatamente geram-se questões acerca de segurança como a forma como são passados os dados de utilizador entre cliente e servidor, forma de encriptação desses dados ou prevenção de manipulação evitando acessos fraudulentos.

De forma a poder abordar e mitigar estas questões de segurança foi decidido que a autenticação do microserviço de interoperabilidade e, por conseguinte, de todo o sistema será baseada em JSON Web Tokens (ou JWTs). Sendo assim a autenticação no sistema de interoperabilidade é responsabilidade do microserviço de interoperabilidade e este segue as melhores práticas da implementação de autenticação por JWT. Desta forma o microserviço de interoperabilidade emite um JWT devidamente assinado, quando um pedido de acesso é facultado com credenciais corretas, que deve ser armazenado e passado juntamente com todos os pedidos seguintes que exijam autenticação de forma a poder identificar corretamente o remetente do pedido.

6.5.5 Mecanismos de execução de configurações

Todos os mecanismos responsáveis pelo processamento e execução de configurações (extração, validação e preprocessamento) necessitam de configurações singulares concebidas para existência e funcionamento exclusivo por atributo. Por design todos estes mecanismos correm de forma sequencial e dependente sendo que os pontos de saída de uns mecanismos são os pontos de entrada de outros sem tolerar quaisquer falhas ou configurações em falta. Sendo assim nenhum mecanismo pode ser executado sem o mecanismo anterior ter acabado com sucesso. Todos os mecanismos irão percorrer de forma iterativa (Liu & Stoller, 1999) todos os atributos de um dado modelo escolhido e irão consultar a base de dados por configurações respetivas a esse atributo. Esta configuração será executada no

respetivo âmbito de cada mecanismo. Estando todos os atributos percorridos o resultado será passado ao próximo mecanismo. No acaso de algum atributo não possuir alguma configuração respetiva na base de dados todo o processo é interrompido de forma controlada e mensagens de notificação enviadas. Este comportamento está representado na Figura 39.

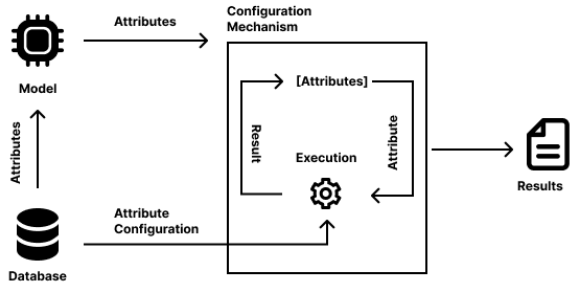

Figura 39 - Mecanismo de Execução de Configurações

Como já referido o sistema de interoperabilidade conta com três tipos de mecanismos de execução de configurações dinâmicas que juntos compõe todo o fluxo de interoperabilidade sendo que todas as restantes funcionalidades servem para configurar ou dar suporte a este fluxo.

- **Mecanismo de Extração de dados**: O mecanismo de extração de dados, como o próprio nome indica é capaz de, através de uma configuração (mapeamento) e de um recurso de interoperabilidade (HL7 ou FHIR), extrair exatamente um dado desejado. Este mecanismo conta com dois motores de extração:
    - **Motor HL7**: Este motor faz uso do Mirth Connect para o manuseamento e execução de mapeamentos de forma a extrair um valor desejado e devolvê-lo ou não caso este não seja encontrado. Este canal de Mirth foi desenvolvido e encontra-se completamente integrado dentro do sistema sendo que a sua criação e configuração é totalmente abstraída pelo microserviço Mirth Connect. O motor HL7 em si trata da gestão dos valores devolvidos pelo Mirth;
    - **Motor FHIR**: A extração de dados de recursos FHIR com base em mapeamentos foi manualmente implementada na forma de um algoritmo de visita iterativa devido à falta de suporte pela parte do Mirth. Ambas abordagens iterativa e recursiva foram exploradas, porém neste caso de uso em particular a iteração mostrou-se mais eficiente sendo que se trata da melhor opção quando:
        - A tarefa ou problema é algo complexo e o desempenho e eficiência são uma grande preocupação (Liu & Stoller, 1999);

- A tarefa em si pode levar a *stack overflow* (Liu & Stoller, 1999; Rinard et al., 2004);
- É necessário muito controlo acerca do fluxo de dados do algoritmo (Liu & Stoller, 1999);

Sendo que a tarefa de parsing e manuseamento de um tipo de recurso não é uma tarefa leve e é expectável que este algoritmo seja executado muito frequentemente a velocidade é um fator muito importante e cumpre os requisitos para ser uma tarefa realizada de forma eficiente por iteração.

- **Mecanismo de Validação de dados**: Este mecanismo é o mais simples de todos os mecanismos uma vez que faz uso de validadores baseados em expressões regulares (regex (Ficara et al., 2008; Thompson, 1968)) para fazer a validação de valores que foram extraídos no mecanismo de extração. Na mesma medida que os outros mecanismos o processo também é individual por atributo e todos os atributos e valores extraídos são validados individualmente na forma de um algoritmo de visita iterativo. A função de validação em si é dada pela simples utilização de expressões regulares presentes nos validadores na base de dados pela função de teste ou execução de expressões regulares nativa ao Node.js de forma a averiguar se o dado extraído segue ou não os padrões de validação estabelecidos para esse atributo (Ficara et al., 2008; Thompson, 1968);
- **Mecanismo de Preprocessamento de Dados**: O mecanismo de processamento de dados é o mais complexo do sistema e também o mais vulnerável pelo que este permite a execução dinâmica de código JavaScript. à semelhança dos mecanismos anteriormente descritos o preprocessamento dos dados é, também, realizado de forma única por atributo de modelo sendo que é necessária a escrita ou escolha de um *script* de preprocessamento para cada atributo. Estes *scripts* são guardados na base de dados e na hora de processar um atributo específico de um modelo são consultados e executados em *runtime* que se torna possível uma vez que JavaScript dispõe de ferramentas de execução de código dinâmico (Laurén et al., 2017; Ojamaa & Düüna, 2013; Staicu et al., 2016b). Esta abordagem torna a funcionalidade de preprocessamento extremamente versátil e flexível, mas também constitui uma enorme ameaça a nível de segurança do sistema sendo que qualquer código pode ser executado(Laurén et al., 2017; Ojamaa & Düüna, 2013). Por esta razão, o mecanismo de preprocessamento integra duas estratégias para mitigar esta ameaça e melhorar a segurança do sistema, *sandboxing,* que consiste em correr o código de forma dinâmica, mas num ambiente completamente isolado do

código fonte, e *whitelisting,* que consiste numa lista com uma quantidade limitada de *tokens* ou expressões consideradas seguras para a escrita de *scripts* dinâmicos (Laurén et al., 2017; Ojamaa & Düüna, 2013). Desta forma este mecanismo foi implementado da forma mais segura possível mantendo todos os benefícios da execução dinâmica de código.

6.5.6 Medidas de Segurança

O microserviço de interoperabilidade lida com dados e operações muito sensíveis pelo que estas necessitam de estar implementadas de forma segura. Sendo assim foram identificadas algumas preocupações de segurança com base na natureza das funcionalidades deste microserviço:

- **Encriptação de dados**: Dados sensíveis como credenciais de autenticação necessitam de ser encriptados para armazenamento na base de dados de forma segura utilizando uma chave secreta, definida pelo administrador e apenas o administrador do sistema. Na eventualidade de um acesso não autorizado à base de dados, todos os dados sensíveis encontram-se encriptados por um algoritmo e chave seguros (Singh et al., 2015). A utilização desta técnica combinada com a autenticação baseada em JWT (Jones, 2015; Sheffer et al., 2020a)constitui uma excelente estratégia de mitigação para a vulnerabilidade de *Cryptographic Failures* (Ertaul et al., 2016; Sriramya & Karthika, 2015);
- **Autenticação segura:** A utilização de JWT (JSON Web Tokens) é um dos padrões de segurança mais bem aceites para construir lógicas de autenticação seguras no que toca a sistemas modernos baseados em Web (Sheffer et al., 2020a). Sendo assim todas as funcionalidades (rotas) deste microserviço encontram-se protegidas pela necessidade de providenciar um token corretamente assinado pelo algoritmo de assinatura. A utilização de JWTs impede a adulteração ou manipulação dos tokens de sistema para escalagem de privilégios ou acessos não autorizados a funcionalidades do sistema (Sheffer et al., 2020a);
- **Execução dinâmica de código**: A funcionalidade de execução dinâmica de código é ao mesmo tempo extremamente poderosa e proporciona grande liberdade de criação, mas também resulta numa das maiores vulnerabilidades em sistemas informáticos que permite a qualquer atacante correr código de forma arbitrária no sistema (*Dynamic Schemes for Speculative Execution of Code*, 1998; Ojamaa & Düüna, 2013; Staicu et al., 2016b). Sendo assim a implementação desta funcionalidade deve ser cuidadosamente pensada e executada segundo as melhores e mais seguras práticas de programação, no microserviço de interoperabilidade esta implementação deu-se através da execução de código dinâmico por meio de um *sandbox*

(Laurén et al., 2017) complementado com uma lista de palavras e expressões permitidas sendo que tudo o resto será considerado perigoso e, portanto, não admitido. Esta medida contribui para a mitigação da vulnerabilidade de *Insecure Design*;

- **Injeções e conteúdo malicioso**: Num sistema informático que recebe qualquer tipo de entradas por parte de um utilizador a ameaça de injeção de conteúdo malicioso quer ele seja injeções de *queries* SQL ou código para execução no servidor está sempre presente e caso não seja mitigada será descoberta, utilizada e abusada por atacantes ao sistema (Dorai & Kannan, 2011; Kaushik & Majumdar, 2019). Para prevenir este tipo de vulnerabilidade e consequente ataque o microserviço de interoperabilidade, seguindo as melhores práticas recomendadas pela comunidade, implementa uma forte validação de corpo de pedidos e funções sendo que nada que fuja às validações por Regex ou pela estrutura do pedido é admitido (Dorai & Kannan, 2011). A utilização de ferramentas de ORM, no caso deste microserviço o Sequelize (Barsoti & Gibertoni, 2020), também limita bastante o alcance deste tipo de ataques uma vez que as consultas à base de dados são abstraídas por uma camada de APIs que não permitem a execução direta de comandos;
- **Desatualização de dependências:** Muitas vezes as vulnerabilidades não originam pelo código desenvolvido, mas sim por vulnerabilidades nas dependências utilizadas sendo que é muito importante mantê-las atualizadas (Dann et al., 2023). O microserviço de interoperabilidade apenas utiliza dependências de grande renome e prestígio sendo que estas são consideradas seguras nos seus respetivos âmbitos na sua versão mais recente até ao momento da escrita do presente documento, e portanto, encontra-se a seguir as melhores práticas no que toca à integração das suas dependências.
- **Comunicação:** Outra grande preocupação de segurança num sistema de microserviços é a forma como eles comunicam entre si sendo que é muito fácil a captura de pacotes de dados quando esta não é executada de forma segura (Zdun et al., 2023). O microserviço de interoperabilidade previne esta vulnerabilidade pela troca de informação encriptada através do protocolo de comunicação HTTPS e a configuração (Sharma, 2023; Uddin et al., 2019)CORS pelo que apenas remetentes e recetores autorizados podem enviar e receber mensagens de forma legível. Esta medida permite mitigar a vulnerabilidade de SSRF (M. Aljabri et al., 2022) .
- **Controlo de Acesso:** Implementação de RBAC (Sharma, 2023; Uddin et al., 2019) para proteção de dados e recursos sensíveis bem como para limitar, organizar e controlar as funções às quais os utilizadores do sistema têm acesso. Este RBAC possui duas *roles* distintas,

administrador e utilizador, sendo que cada sessão atribuída a um dado cliente tem estas *roles* por base.

6.6 Microserviço Mirth Connect

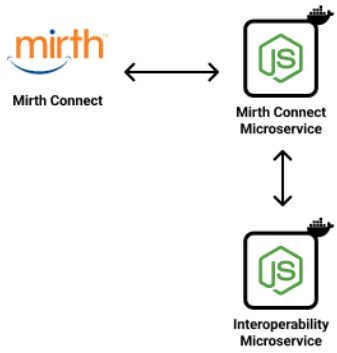

Figura 40 - Arquitetura Microserviço Mirth Connect

O Microserviço Mirth Connect é uma componente de suporte ao microserviço de interoperabilidade responsável pelo controlo e manuseamento da plataforma Mirth Connect através da sua REST API. O âmbito deste microserviço é extremamente reduzido em comparação ao de interoperabilidade, porém é fruto da divisão de responsabilidades pelas diferentes partes do sistema sendo que o microserviço Mirth Connect é a única parte do sistema que tem acesso aos segredos e credenciais de conexão à REST API Mirth. Sendo um microserviço num sistema composto por múltiplos, o microserviço Mirth Connect existe de forma isolada do resto do sistema fazendo apenas parte da sua arquitetura a plataforma Mirth Connect e o microserviço de interoperabilidade com o qual existe a troca de alguma informação para armazenamento na base de dados. Esta arquitetura esta representada na Figura 40.

Este microserviço tem por base o *template* de microserviços discutido e documentado acima, portanto tem como base o Node.js e Express, sendo que utiliza a biblioteca de cliente HTTP "axios" para realizar a comunicação, quer com a REST API Mirth, quer com o microserviço de Interoperabilidade. Segue a estrutura MVC sendo que apenas se foca na parte de Controller e apenas disponibiliza várias funcionalidades referentes à utilização da REST API Mirth ao resto do sistema.

Conforme os restantes microserviços pertencentes ao sistema o microserviço Mirth Connect também é totalmente configurável de forma a moldar o seu comportamento e capacidade de conexão a instâncias Mirth Connect, mesmo que se encontrem em ambientes diferentes, com base na configuração de variáveis de ambiente. Estas variáveis cobrem a porta onde o serviço irá correr, as credenciais para interface com o Mirth Connect e endereço completo do serviço de interoperabilidade.

Tendo o *template* por base também consiste numa REST API (Doglio, 2018) bem configurada com o mecanismo CORS (Nicholas, n.d.) de forma a limitar e barrar pedidos não autorizados e cujas funcionalidades estão disponíveis através de rotas acessíveis pelo protocolo HTTP. Estas funcionalidades são:

- **Controlo sobre a consulta de informação**: A REST API Mirth possui uma variedade de rotas que permitem a consulta de qualquer informação na mesma;
- **Controlo sobre a criação de canais**: A REST API Mirth possui uma rota responsável pela criação de canais uma vez que estes são totalmente configuráveis via JSON/XML tornando-se assim possível a criação automática de canais totalmente configurados e prontos a utilizar;
- **Controlo do estado e mutação dos canais**: O microserviço Mirth Connect consegue alterar qualquer aspeto de um canal Mirth por meio da rota de UPDATE da REST API. Outro aspeto importante que também possível modificar é o estado de implementação do canal.

6.6.1 Mecanismo de Interface com a REST API Mirth

A interface com a REST API Mirth foi implementada neste microserviço de uma forma bastante simples utilizando apenas um simples Client HTTP (axios). Porém foram necessárias algumas preparações e considerações que estão disponíveis para leitura e consulta na documentação oficial do Mirth Connect:

- **Geração de um token de acesso**: A API de cliente do Mirth (REST API) encontra-se protegida por um sistema que utiliza o esquema de Autentiação Básica (John et al., 1999) que consiste na utilização uma Header do protocolo HTTP "Authorization" para enviar um par de credenciais de acesso para o servidor. Nesta abordagem ambas as partes da credencial de acesso (geralmente nome de utilizador e palavra-passe) encontram-se separadas por dois pontos (:) e codificados em Base64. O conteúdo da header Authorization consta, portanto, da palavra "Basic" seguida de um espaço e das credenciais codificadas em Base64 (John et al., 1999). Estes são precisamente os passos que o microserviço Mirth Connect segue para gerar um token de autenticação para utilização com a API de cliente do Mirth;
- **Resposta JSON**: Para melhor integração com o Node.js foi necessária a configuração da Header "Accept" para "application/json" uma vez que a API de cliente do Mirth envia as respostas aos pedidos em XML por defeito. Fazer o parsing de XML para JSON seria um passo adicional desnecessário quando o Mirth é configurável para retorno de respostas em JSON diretamente;

- **X-Requested-With**: Esta Header é necessária para comunicar ao Mirth que quem está a realizar o pedido HTTP é um cliente HTTP caso contrário o Mirth recusa o pedido.

6.6.2 Medidas de Segurança

Este microserviço possui um âmbito muitíssimo mais pequeno que o de interoperabilidade sendo que as ameaças e preocupações de segurança serão reduzidas, mas foram tidas em consideração de forma a fortificar e assegurar todas as frentes do sistema:

- **Comunicação entre microserviços**: A comunicação entre microserviços foi realizada através do protocolo HTTP de forma segura, sendo que o microserviço Mirth Connect está devidamente configurado com o CORS e ambos os microserviços em interação possuem as suas rotas e funcionalidades protegidas com uma camada de autenticação implementada com JWTs;
- **Armazenamento de dados sensíveis**: Dados sensíveis como as credenciais de conexão ao Mirth Connect não se encontram no código fonte, mas sim são passados na forma de variáveis de ambiente para aumentar a segurança de implementação e o controlo dinâmico desta informação ao executar o microserviço (Environment Variables: What They Are and How To Use Them, 2023).

6.7 Microserviço de Gestão de Mensagens

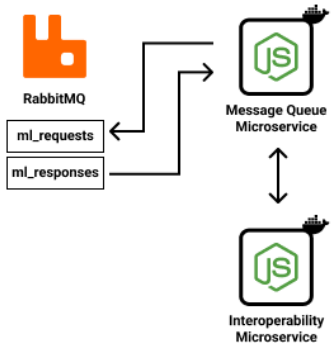

Figura 41 - Arquitetura Microserviço de Gestão de Mensagens

À semelhança do microserviço Mirth Connect, o microserviço de gestão de mensagens também consiste numa extensão e suporte ao microserviço de interoperabilidade disponibilizando funcionalidades para interface com um gestor de mensagens como o RabbitMQ. Este microserviço é responsável por toda a interface com a plataforma de gestão de mensagens e contém todos os segredos e credenciais para o fazer de forma segura sendo que é a única componente de todo o sistema com essa responsabilidade e

acesso. Fazendo parte de um sistema composto por múltiplos microserviços interligados o microserviço de gestão de mensagens comunica diretamente com o microserviço de interoperabilidade quando necessário e faz interface direta com o gestor de mensagens, mais concretamente duas filas ou *queues* de mensagens ml_requests e ml_responses. Este comportamento está representado na Figura 41.

Baseado no *template* de desenvolvimento de microserviços este microserviço tem por base o Node.js e Express tendo as suas funcionalidades expostas e acessíveis através de pedidos HTTP na forma de uma REST API mas também implementa um *listener* capaz de captar quaisquer mudanças ou mensagens no gestor de mensagens que é executado em paralelo com a REST API.

Segundo a estrutura MVC este microserviço tem o seu foco na componente de Controller e apenas disponibiliza as suas funcionalidades referentes à interface com um gestor de mensagens ao resto do sistema.

Exatamente da mesma forma que os restantes microserviços, o microserviço de gestão de mensagens também é totalmente configurável através das suas variáveis de ambiente de forma a moldar o seu comportamento e capacidades de conexão a gestores de mensagens, mesmo que em ambientes diferentes ou separados. Estas variáveis especificam a porta onde o serviço irá correr, as credenciais para interface com a plataforma de gestão de mensagens e o endereço do serviço de interoperabilidade.

Sendo uma REST API o microserviço de gestão de mensagens possui todas as suas funcionalidades devidamente expostas e acessíveis ao resto do sistema via pedidos HTTP feitos de forma autenticada e segura:

- **Criação e publicação de novas mensagens**: Utilizando a fila de mensagens **ml_requests** a propagação de um novo pedido de execução de modelos em sistemas ABI é feita à plataforma de gestão de mensagens (RabbitMQ) sendo que todos os sistemas que se encontrem conectados a esta fila receberão a mensagem para a tratarem e executarem;
- **Captura de respostas**: Através da fila de mensagens **ml_responses** a resposta dada por um modelo num sistema ABI terá de ser propagada para o gestor de mensagens de forma a ser capturada pelo microserviço de gestão de mensagens que irá, conforme a resposta, propagá-la para o microserviço de interoperabilidade que por sua vez irá atualizar o estado do pedido respondido na base de dados.

6.7.1 Mecanismo de Interface com Gestores de Mensagens

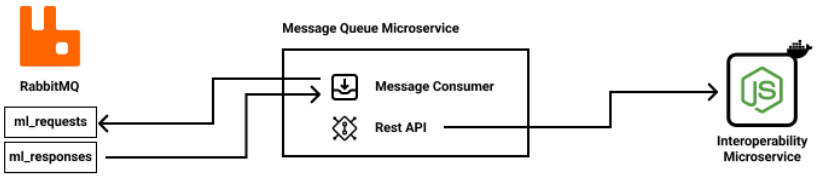

Figura 42 - Mecanismo de comunicação para gestão de mensagens

Como está representado na Figura 42 o microserviço de gestão de mensagens dispõe de duas interfaces para comunicação e interação com o gestor de mensagens (RabbitMQ) e o microserviço de Interoperabilidade na forma de uma interface de interação em tempo real e uma REST API respetivamente.

O consumidor de mensagens tem a capacidade de, em tempo real, enviar mensagens para a fila de **ml_requests** e de receber ou capturar mensagens da fila **ml_responses** sendo que isto torna possível a comunicação assíncrona com sistemas externos, como um sistema ABI, que necessitem de acesso aos pedidos de execução de modelos assim como o tratamento e propagação das suas respetivas respostas.

A REST API existe para estender as capacidades do microserviço de gestão de mensagens e interação com as plataformas para esse mesmo fim ao resto do sistema que é composto apenas pelo microserviço de interoperabilidade.

6.7.2 Medidas de Segurança

À semelhança do microserviço Mirth Connect, o microserviço de gestão de mensagens também possui um âmbito extremamente reduzido em relação ao microserviço de interoperabilidade pelo que serve como uma extensão das suas funcionalidades para interface com plataformas de gestão de mensagens e recetor/captor de mensagens em tempo real. Por esta razão as preocupações base e mitigações implementadas foram exatamente as mesmas passando pela comunicação segura e protegida bem como o armazenamento e carregamento seguro de segredos de conexão através de variáveis de ambiente configuráveis pelo administrador do sistema.

Porém devido ao facto de que à natureza sensível da informação que passa por este microserviço foram tidas em conta mais algumas preocupações acrescidas:

- **Conexão e interface seguras**: O método de conexão e interface com a plataforma de gestão de mensagens para onde são propagados todos os pedidos efetuados com sucesso é das partes mais importantes e vulneráveis deste microserviço caso não seja desenvolvido ou implementado

corretamente. Por esta razão a biblioteca *amqplib* foi escolhida devido à sua popularidade e reputação para interfaces limpas e seguras com gestores de mensagens;

- **Validação de mensagens**: Os dados e mensagens propagados aos gestores de mensagens serão diretamente capturados por qualquer sistema que esteja conectado ao gestor de mensagens pelo que a circulação de mensagens maliciosas quer pelo sistema de interoperabilidade quer por qualquer outro sistema externo constitui um risco grave de segurança. Por esta razão as informações recebidas pelo sistema são sujeitas a uma forte validação e preprocessamento no microserviço de interoperabilidade, mas também passam por uma validação de caracteres ilegais no microserviço de gestão de mensagens;
- **Tratamento de erros**: O não tratamento de erros pode levar à quebra do sistema de interoperabilidade em si mas também à exposição de informação sensível não desejada aos utilizadores finais como informação acerca das tecnologias, versões ou funcionamento da lógica do sistema. A exposição deste tipo de informação poderá abrir mais oportunidades de ataque ao sistema pelo que todos os erros estão a ser tratados apenas retornando informação ambígua ao utilizador final sendo que a informação referente ao exato erro em si é registada numa consola à qual apenas o administrador do sistema consegue aceder.

6.8 Interface Gráfica

A plataforma ou aplicação de interface gráfica ou interface de utilizador é uma parte muito importante do sistema pelo que consiste na forma de interface que o utilizador dispõe para interagir com o sistema de uma forma abstraída e simples. Por *design*, a interface gráfica do sistema de interoperabilidade encontra-se completamente separada dos microserviços de Backend sendo que a comunicação é realizada através de pedidos HTTP.

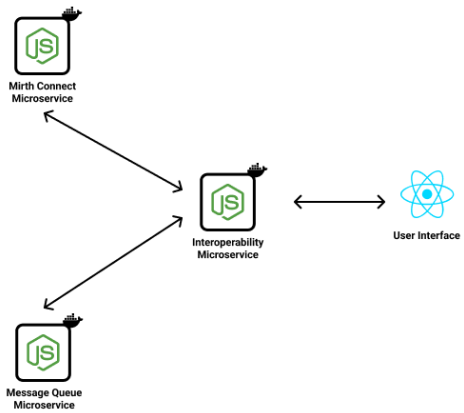

Figura 43 - Arquitetura Interface Gráfica

Como a Figura 43 sugere a interface gráfica consiste numa aplicação Web desenvolvida utilizando o *framework* React.js (Rawat & Mahajan, 2020) que dispõe de um completo arsenal para o auxílio ao desenvolvimento web e comunica apenas com o microserviço de interoperabilidade para realizar todas as operações e funcionalidades do sistema.

Sendo esta aplicação o Frontend do sistema consiste na camada de View segundo a estrutura MVC pelo que está responsável pela interface com todas as funcionalidades do sistema via comunicação com o Backend e por toda a apresentação dos dados ao utilizador.

Esta aplicação de Frontend é servida através do servidor HTTP Nginx devido à sua popularidade, facilidade de uso e implementação e grande rapidez e eficiência de serviço (Guolang et al., 2018; Kithulwatta et al., 2022; Soni, 2016).

O uso do servidor Nginx permite que a aplicação Web seja construída, reduzida e otimizada ao máximo para execução em produção na forma de ficheiros estáticos.

6.8.1 Antd Design

Fazendo uso direto da modularidade dos componentes React a utilização de uma biblioteca de componentes, ainda que não essencial, podem poupar muito tempo e esforço a um desenvolvedor ao disponibilizarem componentes pré-construídos de possível integração e utilização em vários projetos melhorando assim a consistência e qualidade do código desenvolvido. Tendo a maior parte, senão todo, o design dos componentes resolvido um desenvolvedor apenas tem de se focar e desenvolver a aplicação em si, o seu funcionamento, lógica e disposição. Para além do mencionado estas bibliotecas são uma ótima forma de um desenvolvedor se manter sempre atualizado quantos ás tendências de design mais recentes e melhores práticas (Praneeth Reddy, 2021).

O Ant Design é uma das bibliotecas mais populares em todo o ecossistema React ideal para aplicações empresariais tendo todas as funcionalidades, componentes e suporte para que qualquer requisito seja satisfeito. Focando-se nas necessidades de específicas de ambientes empresariais fornece um vasto leque de componentes muito bem desenvolvidos e otimizados normalmente encontrados ou utilizados nestes ambientes como tabelas, formulários, gráficos e menus.

A facilidade de integração do Ant Design vem muito da sua documentação abrangente. Esta encontra-se muito bem escrita e de fácil navegação abordando todo o processo de utilização desde a instalação inicial e configuração até à utilização individual e customização de componentes.

Outro grande foco desta biblioteca é o seu forte desempenho e otimização dos seus componentes visto que estes são conceitos e requisitos cruciais para o manuseamento de grandes volumes de dados e exigências de tráfego comuns em ambientes empresariais.

Para além do mencionado o Ant Design oferece, também, funcionalidades essenciais, como acessibilidade garantindo que qualquer aplicação é adequada a todos os tipos de utilizadores de todas as capacidades. Ainda no tópico de design o Antd Design oferece um elevado grau de personalização, permitindo que um desenvolvedor possa adaptar qualquer componente ao seu próprio estilo/gosto ou ao design padrão escolhido para a aplicação em desenvolvimento (Praneeth Reddy, 2021).

Para além do mencionado todos os componentes proporcionados pelo Ant Design foram cuidadosamente desenvolvidos com um grau muito elevado de qualidade sendo que será muito raro encontrar algum tipo de erro ou funcionamento não esperado e foram desenvolvidos através de design responsivo pelo que poderão brilhar em qualquer tipo de ecrã ou monitor.

A pesquisa bibliográfica neste aspeto revelou que o Ant Design é perfeito em casos de uso como **Sistemas CRM, Sistemas ERP, Sistemas de contabilidade, Sistemas de RH, Sistemas de gestão de conteúdos** entre outros (Praneeth Reddy, 2021).

Tendo em conta o âmbito da aplicação de interface gráfica do sistema de interoperabilidade o Ant Design não só se qualificou como possível candidato a biblioteca de componentes como se destacou das restantes por todas as características acima descritas.

6.8.2 Layout da Interface Gráfica

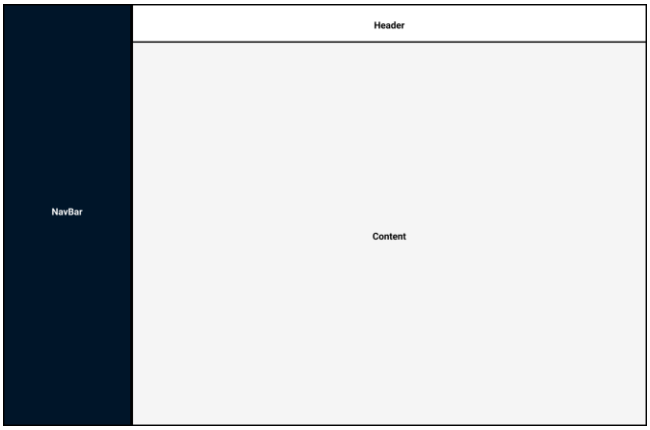

Figura 44 - Layout Interface Gráfica

Como é possível verificar na Figura 44 a disposição ou *layout* da interface gráfica segue um dos padrões mais comuns e utilizados em toda a indústria de desenvolvimento de aplicações web e interfaces gráficas onde existem duas colunas e duas linhas de componentes. Por esta razão este padrão designa-se por *2-*

column-layout e geralmente é acompanhado por uma terceira linha de rodapé pelo que o *layout* da interface gráfica é apenas uma adaptação.

Sendo assim a interface conta com as seguintes componentes:

- **Header/Cabeçalho**: Visto que toda a funcionalidade de navegação se encontra coberta pela barra lateral de navegação o cabeçalho desta interface terá apenas a funcionalidade de *cal to action* deste sistema como por exemplo o botão de fechar a sessão;
- **NavBar/Barra de Navegação**: Este componente contém todo o mecanismo para acionar/ativar a navegação na interface gráfica. Contém também o logotipo e nome da aplicação e sistema;
- **Content/Conteúdo**: Contém todo o mecanismo e lógica de navegação. Neste componente serão apresentados diferentes componentes de forma condicional com base no estado da navegação da aplicação de interface gráfica. É o único componente de apresentação condicional capaz de ser trocado de forma dinâmica.

Desta forma todo o conteúdo da aplicação de interface gráfica é estático sendo que a única parte que troca de forma dinâmica é o componente de Conteúdo. É no componente de conteúdo que tudo acontece dentro da interface servindo os outros dois componentes como suporte à navegação.

A componente de conteúdo é apresentada ou *rendered* de forma condicional com base nos parâmetros que são passados no URL. Estes parâmetros são tratados por um *router* ou encaminhador de tráfego que, com base no URL, irá apresentar um componente de conteúdo especifico de forma dinâmica. Este comportamento encontra-se representado na Figura 45.

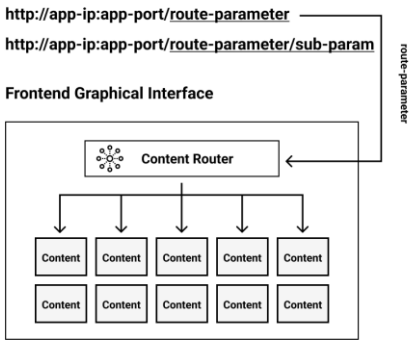

Figura 45 - Router Interface Gráfica

6.8.3 Autenticação

O Backend do sistema de interoperabilidade encontra-se protegido através da verificação de assinaturas JWT sendo que para a autenticação é necessário providenciar um *token* assinado de forma válida que irá comunicar ao sistema quem está a realizar a operação. A geração deste *token* acontece no Backend através da funcionalidade de *login* sendo que será necessária a integração de uma página de *login* sem qualquer relação com a aplicação base da interface bem como um mecanismo de roteamento/encaminhamento capaz de apresentar diferentes partes do sistema conforme o utilizador estiver ou não autenticado e for ou não administrador/cliente do sistema.

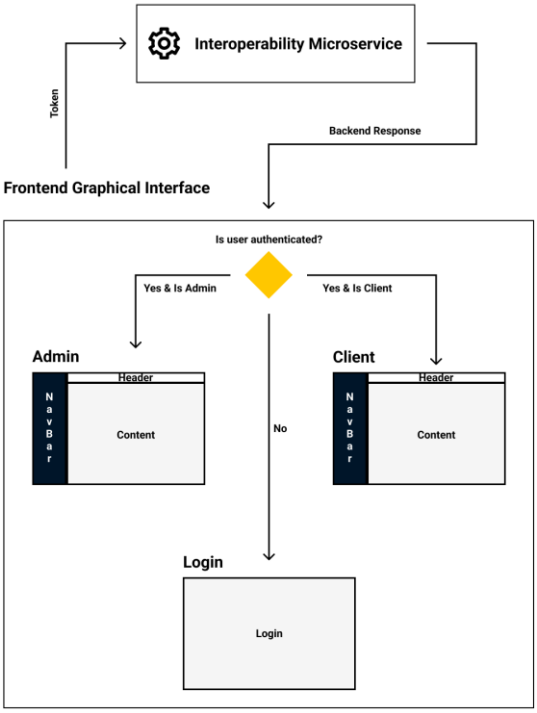

Figura 46 - Mecanismo de Autenticação e Encaminhamento Interface Gráfica

A Figura 46 representa o comportamento e mecanismo de encaminhamento da interface gráfica ao lidar com a autenticação do Backend do sistema de interoperabilidade sendo que podem acontecer três coisas aquando da sua utilização:

- Interface gráfica não possui um token ou possui um token inválido/expirado: Neste caso a interface irá apresentar a página de login ao utilizador sendo que quer sem um token quer com um token inválido a utilização do sistema é impossível;
- Utilizador é Administrador: Neste caso o token na posse da interface gráfica é de um utilizador administrador do sistema sendo que a resposta do Backend especifica o utilizador referente ao *token* e o seu cargo no sistema. Sendo assim a interface apresenta a área de administrador que segue a estrutura ou disposição documentada acima na Figura 46 e está feita à medida para

todas as necessidades e funcionalidades ás quais um administrador do sistema tem acesso como a criação de modelos, mapeamentos, cliente preprocessadores, validadores entre outras operações sensíveis do sistema;

- Utilizador é Cliente: Neste caso o token na posse da interface gráfica é de um utilizador cliente do sistema sendo que a resposta do Backend especifica o utilizador referente ao *token* e o seu cargo no sistema. Sendo assim a interface apresenta a área de cliente que segue a estrutura ou disposição documentada acima na Figura 46 e está feita à medida para todas as necessidades e funcionalidades ás quais um cliente do sistema tem acesso como acesso aos modelos implementados, teste e uso dos mesmos e outras operações de conta como a criação de *tickets*;

6.8.4 Conteúdos/ *Views* Desenvolvidos

O desenvolvimento desta interface desenrolou-se ao longo de três *sprints*, ou seja, três semanas onde, com base no que está documentado acima foram desenvolvidos bastantes componentes mutáveis para o componente de Conteúdo. Estes componentes encontram-se listados na seguinte tabela.

Tabela 6 - Componentes de interface gráfica

Componente	Descrição	Acesso	Funcionalidades Disponíveis
Login	Página de Login onde o utilizador poderá autenticar-se.	Front-Office	O utilizador poderá efetuar o login quer como administrador quer como cliente.
Dashboard	Página principal da interface onde o utilizador poderá visualizar vários aspetos do sistema de forma rápida.	Admin Cliente	O utilizador pode consultar de forma rápida todas as informações relevantes aos modelos, pedidos ao sistema e *tickets* com base no seu cargo.
Pedidos	Página onde estão presentes todos os pedidos feitos ao sistema.	Admin Cliente	O utilizador pode observar todos os pedidos realizados ao sistema e ver todos os seus metadados com base no seu cargo.
Modelos	Página com uma visão geral de todos os modelos do sistema.	Admin Cliente	O utilizador pode observar todos os modelos no sistema incluindo os seus metadados com base no seu cargo. Caso o utilizador for administrador tem a capacidade de criar novos modelos.
Clientes	Página com uma visão geral de todos os clientes do sistema	Admin	O administrador do sistema pode observar todos os clientes e respetivos metadados. O administrador do sistema pode criar novos clientes
Mapeamento de Atributos	Página com uma visão geral de todos os mapeamentos de atributos por modelo	Admin	O administrador do sistema pode observar quantos mapeamentos existem por modelo. O administrador do sistema pode criar novos mapeamentos.

Validadores	Página com uma visão geral de todos os validadores no sistema	Admin	O administrador do sistema pode consultar todos os validadores que existem no sistema, atualizá-los e apaga-los; O administrador do sistema pode criar validadores; O administrador do sistema pode testar os validadores.
Preprocessadores	Página com uma visão geral de todos os preprocessadores no sistema	Admin	O administrador do sistema pode consultar todos os preprocessadores que existem no sistema, atualizá-los e apagá-los; O administrador do sistema pode criar preprocessadores; O administrador do sistema pode testar os preprocessadores.
Problemas/*Issues*	Página com uma visão geral de todos os problemas criados por clientes.	Admin Cliente	O utilizador pode observar todos os problemas criados no sistema conforme o seu cargo.
Conversação e Detalhes de Problemas/*Issues*	Página com informações acerca de um *issue* e janela de conversação	Admin Cliente	Tanto o administrador como o cliente criador de um *issue* podem enviar mensagens para a conversa do mesmo, alterar o seu estado (aberto/fechado) ou apaga-lo.
Sobre	Página com informações acerca do sistema	Admin Cliente	O utilizador pode observar o estado de todos os microserviços no sistema e métricas do sistema Mirth Connect dependendo do seu cargo.
Definições de conta	Página com informações acerca da conta de um cliente autenticado	Cliente	O cliente pode consultar todas as suas informações e tem acesso a funcionalidades de gestão de conta e tokens.
Detalhes de Modelo	Página com informações acerca de um modelo e funcionalidades de gestão de modelos	Admin Cliente	O utilizador pode consultar todos os dados e detalhes de um modelo incluindo os mapeamentos dos atributos para os recursos de interoperabilidade, validadores e preprocessadores. O utilizador pode testar o modelo utilizando diferentes tipos de recursos de interoperabilidade O administrador do sistema pode criar, apagar ou atualizar qualquer uma destas configurações. O administrador do sistema pode fazer a implementação de um modelo.
Detalhes de pedido	Página com informações acerca de um pedido.	Admin Cliente	O utilizador pode consultar os metadados de um pedido como o seu estado e resposta mas também pode ver os atributos satisfeitos, valores extraídos e processados bem como o recurso de interoperabilidade original
Canais	Página com uma visão geral dos canais Mirth Connect integrados no sistema	Admin	O administrador do sistema pode observar uma lista de todos os canais Mirth Connect integrados no sistema e os seus respetivos metadados. O administrador do sistema pode apagar um canal.
Detalhes de canal	Página com os detalhes de um canal Mirth Connect	Admin	O administrador do sistema pode consultar todos os detalhes e informações acerca do canal Mirth Connect.

			O administrador do sistema pode fazer o *deploy* do canal ou desfaze-lo. O administrador pode apagar o canal.
Detalhes de Cliente	Página com os detalhes de um cliente do sistema	Admin	O administrador do sistema pode consultar todos os detalhes e informações acerca de um cliente. Também tem acesso a funcionalidades de controlo de acesso aos diferentes modelos do sistema bem como à atribuição de mapeamentos personalizados.
Detalhes de mapeamento de atributo	Página com os detalhes de todos os mapeamentos de um atributo de um modelo do sistema.	Admin	O administrador do sistema pode consultar todos os pares chave-valor para os mapeamentos de um qualquer atributo de um modelo.

6.9 Controlo de Versões

O Git foi a ferramenta de escolha para a gestão de versões e do software desenvolvido ao longo de todo o processo de desenvolvimento devido às suas características extremamente apelativas (Zolkifli et al., 2018). As melhores práticas foram seguidas como o *commit* de alterações realizados de forma regular para garantir um histórico de trabalho bem definido, mensagens de *commit* claras e concisas de forma a maximizar a boa comunicação e transparência do trabalho realizado, nomeação convencional de *branches* sendo que através dos simples nomes das mesmas é possível inferir a alteração realizada e a utilização de *Pull Requests* (Merges) para a integração das diferentes *branches* na base central de código. Estas práticas foram implementadas para a garantia da qualidade do código e trabalho desenvolvido bem como para a sua documentação clara e concisa (Chacon & Straub, 2023).

6.9.1 Estratégia de Gestão de *Branches*

Fazendo uso das características e funcionalidades modernas para o desenvolvimento de software oferecidas pelo Git é necessário ter bem estabelecido uma boa estratégia de gestão das diferentes *branches* e versões do software desenvolvido. Isto permite estabelecer uma base sólida e robusta para o controlo transparente das diferentes versões do mesmo.

Sendo assim os repositórios de todo o código-fonte respetivo ao sistema de interoperabilidade seguem as seguintes práticas para o controlo de versões possuindo um fluxo bem definido que se encontra representado na Figura 47.

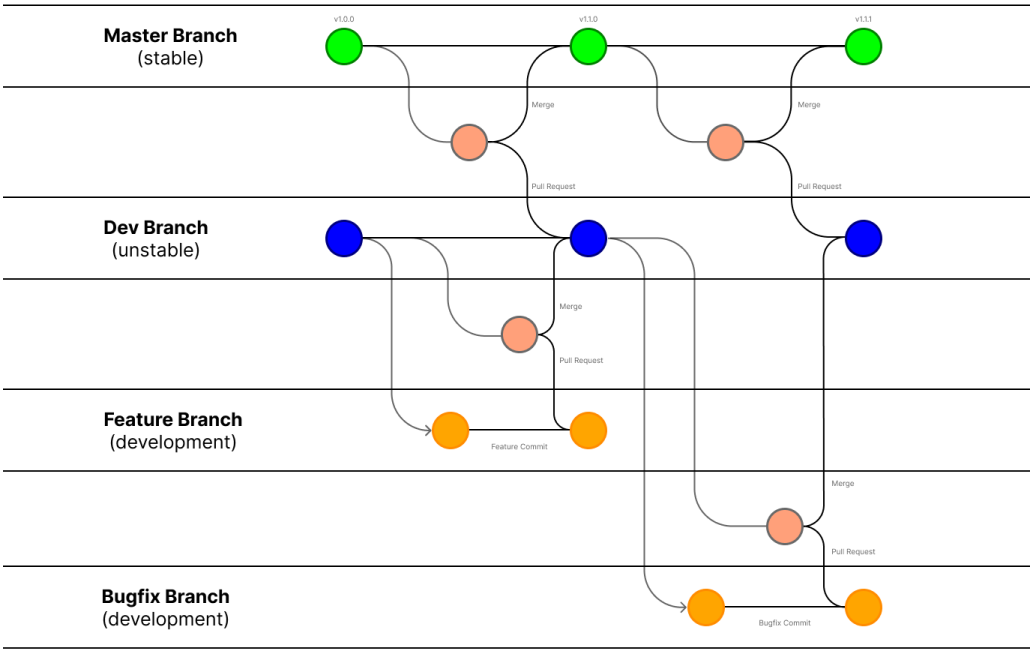

Figura 47 – Estratégia de *Branching*

O fluxo representado no diagrama presente na Figura 47 demonstra a estratégia de *branching* adotada para todo o desenvolvimento realizado sendo que conta com duas *branches* de longo prazo, *dev* e *master* para testar mudanças realizadas e hospedar a versão definitiva do código respetivamente, bem como várias *branches* de curto prazo para o desenvolvimento de novas funcionalidades e alterações.

- *Branch master* – Esta *branch* possui todo o código na sua forma definitiva consiste sempre na versão mais recente do projeto sendo que será este o código utilizado em produção. Sendo que esta *branch* constitui a versão de produção do código-fonte, uma integração nova vinda da *dev branch* implica um incremento na versão do projeto (por exemplo versão 1.0.0 passa a 1.0.1);
- *Branch dev* – Esta *branch* possui todo o código e funcionalidades que ainda estão para ser testados sendo que é expectável que uma grande parte destas esteja instável e, portanto, ainda não se encontram prontas para avançar para a *master branch*. O código presente nesta *branch* constitui a versão de teste mais recente do projeto sendo que as suas funcionalidades estão sujeitas a testes e experiências antes de serem integradas no master *branch*;
- *Feature Branches* – A iteração dos repositórios e código-fonte desenvolvido para o sistema conta com a criação de *branches* de curto prazo cujo âmbito é altamente objetivo sendo que a adição de uma nova funcionalidade ou correção de um erro carece da criação de uma *branch*, do

commit das mudanças necessárias e da integração da mesma com a *dev branch*. Após os devidos testes utilizando o código fonte nesta *branch* faz-se a integração das mudanças para a *master branch* onde passarão a ser definitivas e prontas a utilizar em produção. Toda as integrações de mudanças nas *branches* de longo prazo carecem de uma revisão de pares.

Desta forma, é garantido que todo o trabalho realizado e código desenvolvido para as diferentes partes do sistema de interoperabilidade se encontra bem documentado e estruturado de forma a poder entregar iterações controladas e consistentes das mesmas.

6.9.2 Controlo de versões semântico

Focando agora no master *branch* é expectável que esta tenha várias iterações ao longo do seu ciclo de vida iterativo sendo que é necessário haver uma forma de as poder distinguir facilmente. Cada iteração nova nesta *branch* traduz-se num número incerto de incrementos ou alterações pelo que é apenas uma integração das alterações na *dev branch*. Esta incerteza provoca confusão e torna o software propício a falhar ou partir por erros de suporte de versões (Raemaekers et al., 2017), por exemplo se numa iteração houver 5 tabelas na base de dados e na seguinte houver apenas 4, devido a uma remoção, irá haver conflitos onde o sistema espera 5 tabelas. De forma a evitar este tipo de problemas é necessário padronizar as diferentes iterações do código de produção sendo que, de longe, a forma mais popular e bem recebida pelo ecossistema de desenvolvimento trata-se de *semantic versioning* (SemVer) (Decan & Mens, 2019; Raemaekers et al., 2017). Todo o código-fonte desenvolvido para o sistema de interoperabilidade apresenta, para a sua *branch* de produção, uma versão devidamente formatada bem como um *devlog* (ou registo de desenvolvimento) representativo das mudanças integradas para o lançamento da mesma. Um exemplo destas versões semânticas pode ser observado na Figura 48.

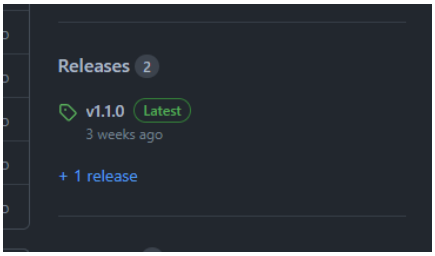

Figura 48 – Versionamento Semântico de um Microserviço

Adotando estas regras, convenções e estratégias é possível afirmar que o software desenvolvido para o sistema de interoperabilidade, quer seja microserviços quer seja aplicação web, encontra-se devidamente desenvolvido, organizado, documentado e publicado sendo que também segue as melhores práticas documentadas pela comunidade científica até ao momento da escrita do presente documento.

6.10 Hospedagem de repositórios

De forma a poder armazenar os repositórios que contém todo o código do sistema é necessária a utilização de uma plataforma de hospedagem de repositórios, de preferência com suporte nativo para Git, o VCS de escolha. O Github foi a plataforma escolhida devido ao seu vasto leque de funcionalidades topo de gama, bem como a facilidade de uso e integração com outros ecossistemas (Cosentino et al., 2017).

A revisão de literatura *"A systematic mapping study of software development with GitHub"* por Valerio Cosentino e Jordi Cabot sugere que o Github é uma plataforma com o potencial de revolucionar a forma como software é desenvolvido bem como o seu uso promove a melhoria das práticas de desenvolvimento. Valerio e Jordi apontam várias vantagens no uso da plataforma como:

- **Comunidade vasta e ativa** – O Github possui uma comunidade de desenvolvedores muito vasta e ativa o que significa que existe uma quantidade extensa de recursos para fornecer ajuda;
- **Controlo de Versões** – Utilizando o Git, o Github possui um excelente sistema e integração para o controlo de versões facilitando muito o acompanhamento das mudanças e a colaboração num base de código;
- **Acompanhamento de Itens** – O Github possui um sistema de *items* ou *issues* que são como tarefas a realizar ou problemas a abordar no futuro sendo que a integração dos mesmos com todas as operações do Git é algo muito útil para a organização do trabalho;
- **Hospedagem de código-fonte** – Como funcionalidade primária o Github é uma plataforma de hospedagem de repositórios geridos por Git sendo que oferece hospedagem gratuita de código-fonte;
- ***Open source*** – O Github é uma plataforma *open source* o que significa que todas as suas funcionalidades e código são transparentes e acessíveis a toda a comunidade sendo que isto abre portas para a colaboração direta com os seus utilizadores e desenvolvedores promovendo um ambiente de colaboração positivo e interativo;
- **Segurança** – Hospedando código no Github já é um bom passo para garantir a segurança do mesmo sendo que a plataforma oferece muitas funcionalidades precisamente para esse fim como autenticação de dois fatores e *code scanning*.

Assim sendo o código fonte das diferentes partes do sistema de interoperabilidade encontra-se hospedado no Github dentro de uma organização criada especificamente para o desenvolvimento do mesmo. Cada repositório contém todo o código fonte do seu respetivo projeto de forma autónoma e

isolada dos restantes repositórios. Fazem parte desta organização os repositórios dos microserviços que constituem o backend do sistema, a aplicação web que constitui o Frontend do sistema e a aplicação micro-frontend (MFE) para a abstração da complexidade da configuração e execução do sistema. Esta organização pode ser observada na Figura 49.

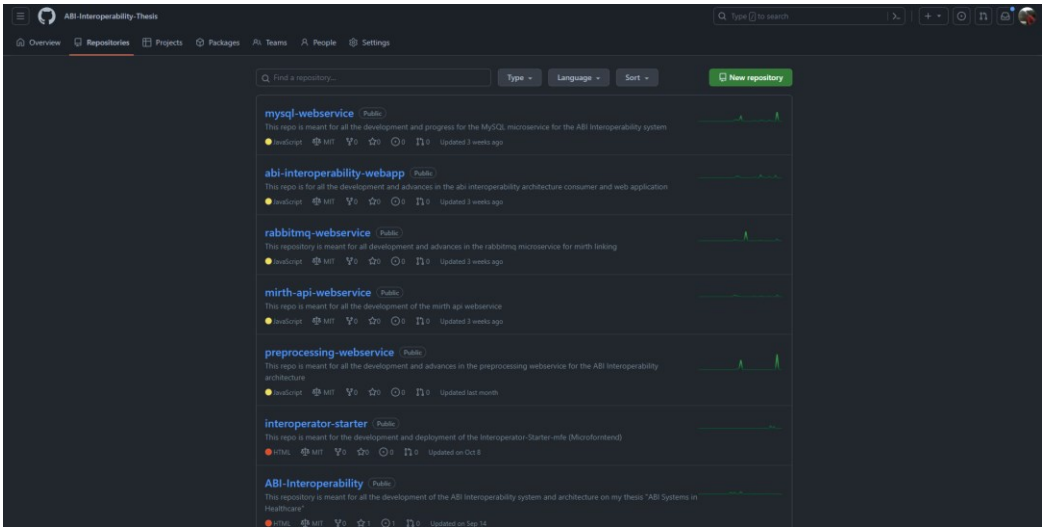

Figura 49 – Organização ABI-Interoperability-Thesis no Github

6.11 Continuous Integration/Continuous Deployment

Tendo por base as vantagens apontadas na revisão bibliográfica, todos os repositórios criados para a hospedagem do código-fonte das diversas partes do sistema de interoperabilidade seguem os conceitos de CI/CD na medida que apresentam automação em:

- Execução de testes de forma a garantir a estabilidade e consistência do código nas versões de longo prazo;
- Integração automática num ambiente de teste de forma a poder haver testes manuais e uma visão geral de todas as alterações numa simulação de um ambiente de produção;
- Lançamento e disponibilização do software automático catalogado com versões semânticas bem diferenciadas e devidamente documentadas

Ao aplicar os três conceitos principais da automação no controlo de versões durante o desenvolvimento de software é possível afirmar que o sistema de interoperabilidade usufrui de todas as vantagens associadas à mesma sendo que traz produtividade aumentada, melhoria da qualidade do código desenvolvido e desenvolvimento, ou integração, mas rápida.

As medidas implementadas contribuem para a minimização de quaisquer descuidos, erros ou falhas que possam passar para o produto final mitigando assim a vulnerabilidade de *Software and Data Integrity Failures* (M. Aljabri et al., 2022).

6.11.1 Github Actions

Tirando completo proveito da plataforma de hospedagem Github a automatização de todos os fluxos ou *workflows* de desenvolvimento, teste e lançamento foi realizada através de *Github Actions*. *Github actions* é uma plataforma de CI/CD que permite aos desenvolvedores num projeto automatizar todos os seus fluxos de desenvolvimento de software proporcionando completa liberdade na construção de automações (Decan et al., 2022; Kinsman et al., 2021).

A plataforma de *Github Actions* apresenta várias automações pré-feitas (Kinsman et al., 2021) que realizam ações ou tarefas simples e comuns como o *checkout* de um repositório, *linting* de código ou lançamento de avisos via e-mail. Para além de todo um ecossistema muito útil de ações pré-feitas, o verdadeiro valor da plataforma *Github Actions* vem da capacidade de um desenvolvedor criar as suas próprias *actions* e, portanto, personalizar completamente à medida toda a automação desejada para cada projeto gerido pelo Github (Decan et al., 2022; Kinsman et al., 2021).

6.11.2 Estrutura Concetual de CI/CD do sistema

De uma forma concetual a automação implementada no processo de desenvolvimento das diversas partes do sistema de interoperabilidade deve cobrir o teste automático das funcionalidades chaves das respetivas componentes, a integração da sua respetiva versão mais atual com todas as mudanças para teste num ambiente reservado para esse fim, o lançamento automático de novas versões bem como a sua disponibilização aos utilizadores finais.

Estas automações devem ser integradas no ecossistema de desenvolvimento fazendo uso das *branches* existentes e respetivas regras pelo que todo o comportamento e automação se encontra representado na figura abaixo.

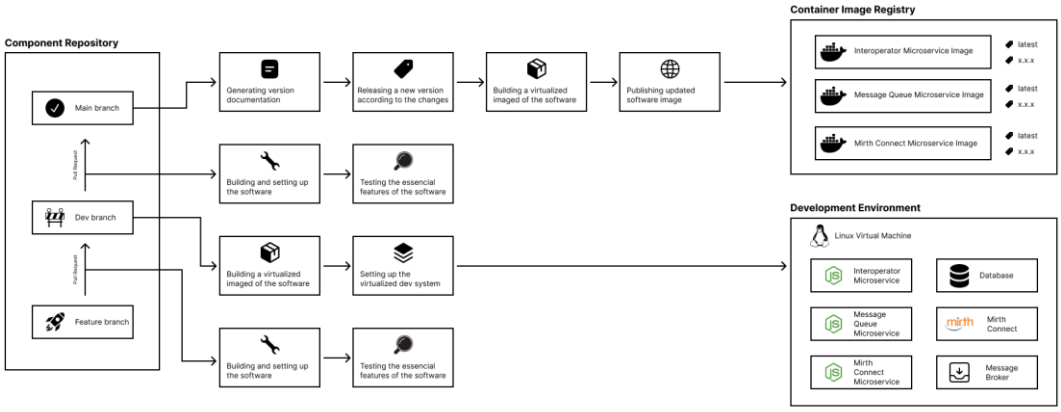

Figura 50 – Estratégia de Automação CI/CD

Como é possível observar na Figura 50 a estratégia de automação para o CI/CD dos vários componentes do sistema de interoperabilidade conta com vários passos condicionados pelo progresso feito por uma determinada mudança pela estratégia de *branching* sendo que segue a seguinte ordem:

- ***Pull Request* de uma *feature branch*** – Sendo que as *feature branches* servem para o desenvolvimento de uma nova funcionalidade ou correção, eventualmente estas mudanças terão de ser submetidas e integradas para a *dev branch* através de um *pull request*. Este *Pull Request* carece de uma revisão humana por parte dos *code owners* para poder ser aceite/aprovado, porém também carece da execução dos testes automáticos que garantem a estabilidade e consistência do software. Caso estes dois requisitos sejam cumpridos as mudanças poderão ser integradas na *dev branch*;
- **Integração automática no ambiente de teste** – Imediatamente após a integração de mudanças novas na *dev branch*, é iniciado o processo de construção de uma imagem *Docker* do software sendo que esta nova imagem atualizada será integrada e lançada no ambiente de testes. Sendo assim numa questão de segundos qualquer alteração efetuada na *dev branch* estará automaticamente lançada e disponível para teste no ambiente de testes;
- ***Pull Request* de *dev* para *main* –** Uma vez confirmadas e testadas as novas mudanças no ambiente de testes, é seguro fazer a integração das mesmas na *main branch* sendo que para isso é necessário efetuar um *Pull Request* partindo da *dev branch* para a *main branch*. Para a conclusão deste *Pull Request* será necessária não só a revisão humana dos *code owners* como a passagem de todos os testes automáticos que testam todas as funcionalidades base do software de forma a garantir a consistência do mesmo. Depois da aprovação dos *code owners* e

do sucesso de todos os testes automáticos o *Pull Request* pode ser completado e as mudanças integradas na *main branch*;

- **Lançamento automático de uma versão atualizada de software** – Imediatamente depois da entrada de novas mudanças na *main branch* uma lista detalhada das mesmas será automaticamente gerada detalhando aquilo que foi acrescentado ou alterado e de que forma. Com base naquilo que foi alterado ou acrescentado, uma versão nova é automaticamente lançada seguindo os padrões estabelecidos pela SemVer (Decan & Mens, 2019; Raemaekers et al., 2017), uma imagem atualizada do software é construída e publicada num registo de imagens de *containers* onde poderá ser publicamente acedida ou utilizada pelos utilizadores finais através das *tags* de versão ou *latest*.

Estando estabelecida a estratégia de automação, todas as componentes do sistema de interoperabilidade seguem-na rigorosa ou aproximadamente sendo que a intervenção humana para a garantia da sua consistência, qualidade e distribuição é mínima ou inexistente minimizando erros humanos e libertando tempo para tarefas realmente importantes.

6.11.3 Estruturação de *commits*

Uma das principais melhores práticas aquando da utilização do Git é a criação de *commits* ou mais concretamente a estruturação das suas mensagens ou notas (Phillips et al., 2011; Raemaekers et al., 2017). Dada a completa liberdade na escrita das mensagens de *commit* é extremamente provável que desenvolvedores diferentes, ou até o mesmo, escrevam mensagens diferentes com diferentes formatos e níveis de especificidade o que irá contribuir para uma documentação inconsistente e difícil de ler e compreender. Para resolver esta tendência é necessário estabelecer alguma estrutura na forma como as mensagens de *commit* são escritas de forma que estas consigam transmitir, de forma consistente, as mudanças com um bom grau de detalhe independentemente do desenvolvedor que as realizou (Huskey & Huskey, 2023; Jiang & McMillan, 2017).

Um dos padrões mais populares e bem recebidos pela comunidade científica no que toca à escrita de mensagens de *commit* é o padrão estabelecido pelo *Conventional Commits* (*Conventional Commits*, 2023) que encorajam a escrita de notas de forma **descritiva**, **acionável**, **consistente** e **informativa**, algo que irá melhorar não só a legibilidade das mesmas como também facilitar a compreensão de alterações realizadas (Huskey & Huskey, 2023).

Outro aspeto muito forte para a padronização da escrita de mensagens de *commit* é a criação de ferramentas de automatização para a geração automática de *devlogs* ou o treino de algoritmos de

inteligência artificial (Huskey & Huskey, 2023; Jiang & McMillan, 2017) que serão tão melhores quanto mais consistentes forem as mensagens.

Seguindo, então, as normas estabelecidas pelo *Conventional Commits* (*Conventional Commits*, 2023) todos os *commits* realizados para as diferentes componentes do sistema de interoperabilidade encontram-se bem e consistentemente documentadas sendo que é através desta consistência e padrão utilizado que é possível realizar a geração automática de um *devlog* (ou lista de mudanças efetuadas na versão). Alguns exemplos de *commits* estruturados podem ser observados na Figura 51.

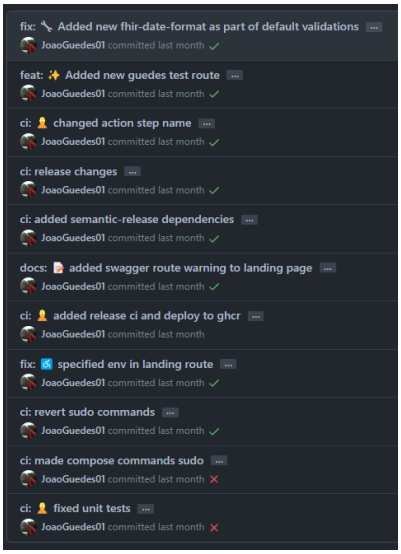

Figura 51 – Exemplos de *commits* padronizados

6.11.4 Virtualização de Software e Docker

No desenvolvimento moderno de *software* o padrão e melhor prática de integração e implementação do mesmo reside na virtualização devido às suas inúmeras vantagens de eficiência na utilização de recursos bem como portabilidade, estabilidade compatibilidade, flexibilidade e custos (Jain, 2020).

Tendo em consideração as vantagens levantadas na revisão bibliográfica, o âmbito do sistema de interoperabilidade e a sua arquitetura baseada em microserviços a escolha de virtualização por meio de *containers* é a mais correta e eficiente (Potdar et al., 2020).

De longe, a tecnologia de virtualização por *containers mais* popular e reconhecida é o Docker sendo que a literatura nas suas diversas aplicações, desempenho e segurança é muito vasta (Potdar et al., 2020; W. Wang, 2022).

Como é de esperar tipos de *software* diferentes carecem de um fluxo de acondicionamento diferentes feitos à sua medida pelo que no âmbito do sistema de interoperabilidade os diversos microserviços serão

acondicionados com base na execução em Node.js ao passo que a aplicação web terá por base um servidor HTTP e seguirão estratégias diferentes.

Sendo que todos os microserviços pertencentes ao sistema seguem a mesma arquitetura, utilizam a mesma tecnologia de execução e foram criados através do mesmo modelo, o seu fluxo de acondicionamento é idêntico pelo que acomoda todas as necessidades do acondicionamento de uma aplicação Node.js.

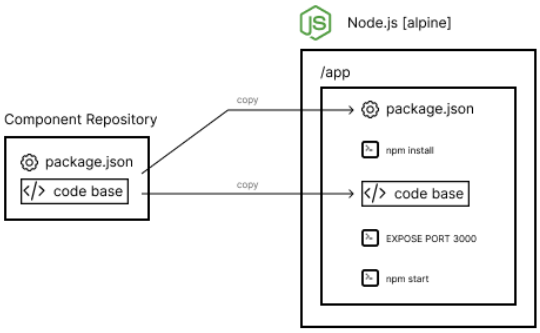

Figura 52 – Fluxo de acondicionamento de microserviços do sistema de interoperabilidade

Como é possível observar na Figura 52 os microserviços seguem um fluxo muito simples sendo que a construção de uma imagem passa por:

- **Iniciação através de Node.js: alpline** – A imagem dos microserviços parte de uma imagem *alpine* de Node.js que consiste numa versão mais leve e otimizada da tecnologia para a execução em *containers*;
- **Criação de uma diretoria raíz** – Criação da diretoria *app* para a hospedagem de todos os ficheiros necessários à execução do microserviço;
- **Cópia do ficheiro de dependências** – Todas as dependências necessárias à execução de uma aplicação Node.js estão presentes no ficheiro *package.json* sendo que este é o primeiro ficheiro a copiar para a diretoria raíz do *container*;
- **Preparação do ambiente e instalação de dependências** – De forma a executar uma aplicação node a única preparação necessária é a instalação das dependências a partir do ficheiro *package.json* sendo que elas são instaladas através do comando *npm install*;
- **Cópia do código-fonte** – Depois de instaladas as dependências todo o código-fonte necessário é copiado para a diretoria raiz do *container*;
- **Configuração do *container*** – Para estes microserviços a única configuração necessária é a exposição da porta onde estarão a ser executados;

- **Execução da aplicação** – Por fim é apenas necessário executar o microserviço fazendo uso do *script* de arranque *npm start* definido no *package.json*.

O acondicionamento da aplicação Web já tem de ser e seguir uma estratégia e fluxo bastante diferentes dos microserviços pelo que não só a forma de execução, mas a parametrização e configuração através das variáveis de ambiente estão condicionadas pela natureza de compilação prévia do React (Rawat & Mahajan, 2020; Rocha et al., 2020).

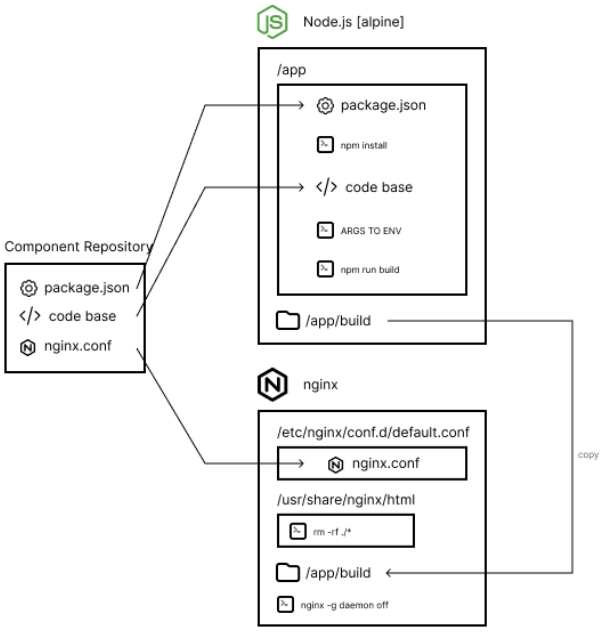

Figura 53 – Fluxo de acondicionamento da aplicação Web do sistema de interoperabilidade

Como é possível observar na Figura 53 o processo de acondicionamento da aplicação Web (Frontend do sistema) passa por dois passos distintos, o do construção e compilação da aplicação através do código-fonte e a hospedagem do código compilado num servidor HTTP. De uma forma mais detalhada os passos necessários para a construção de um *container* para a aplicação Web são:

- **Iniciação através do Node.js: *alpine* e preparação do ambiente Node** – Uma vez que o React é baseado em Node.js o processo inicial para a preparação do *software* é idêntico ao dos microserviços;
- **Criação de variáveis de ambiente a partir de parâmetros** – Devido à necessidade de compilação do React não é possível estabelecer variáveis de ambiente depois da mesma sendo que estas terão de ser passadas como parâmetros antes da imagem ser construída;
- **Compilação do código-fonte** – Compilação do código-fonte React através do *script* de *build* disponibilizado pela *framework*. Este passo irá gerar uma pasta *build* com o código compilado pronto a ser servido;

- **Iniciação da segunda Fase** – Através do Nginx cria-se e limpa-se o ambiente para o servidor poder ser executado;
- **Cópia da configuração do Nginx** – O ficheiro de configuração *nginx.conf* é copiado para a diretoria correta de forma a que o Nginx siga as configurações corretas. Nestas configurações constam os mapeamentos para a diretoria alvo bem como estratégias para tratamento dos erros e pedidos sem sucesso;
- **Cópia do código compilado** – A diretoria *build* é copiada para o caminho de serviço correto do Nginx de forma a que a aplicação compilada possa ser servida;
- **Execução dos servidor Nginx** – Execução do comando de serviço do Nginx.

Desta forma todas as componentes do sistema de interoperabilidade podem ser acondicionadas, com sucesso, em *containers* de forma a poderem ser integradas na arquitetura planeada de forma ininterrupta, rápida, rentável e de fácil manutenção.

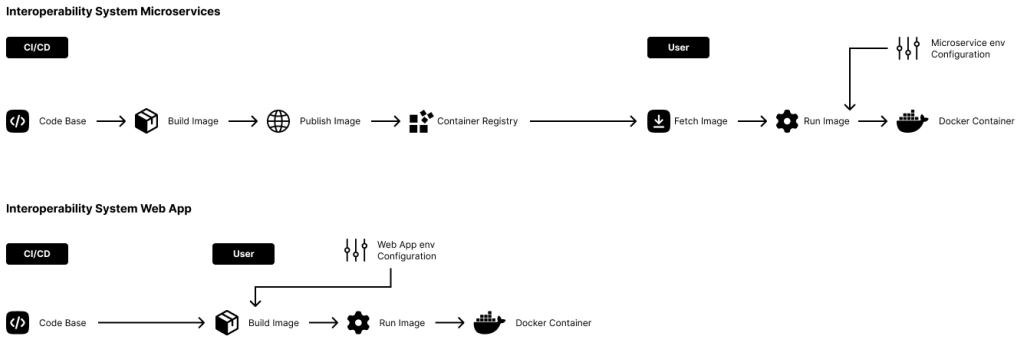

Figura 54 – Fluxos de consumo das partes do sistema de interoperabilidade

Como a Figura 54 indica, devido às diferenças na configuração de cada uma das diferentes partes do sistema, estas também terão uma forma diferente de serem acessadas sendo que para os microserviços o utilizar apenas tem de ir buscar a imagem previamente construída pelo processo de CI/CD e apenas correr o *container* facultando as devidas configurações documentadas na secção de desenvolvimento do presente documento e na documentação do repositório. Em relação à aplicação web é impossível facultar configurações na forma de variáveis de ambiente a uma aplicação React compilada sendo que esta compilação terá de ser feita do lado do utilizador. Sendo assim o utilizador terá de construir uma imagem com base no código-fonte presente no repositório (*master branch*) facultando neste passo as configurações sob a forma de parâmetros de contrução. Após a imagem estar construída o utilizador pode corrê-la obtendo assim um *container* que serve a aplicação web do sistema.

6.11.5 Docker Compose

Embora seja perfeitamente possível montar e executar o sistema de interoperabilidade através da preparação individual de todos os microserviços e componentes do mesmo, fazê-lo desta forma seria desnecessariamente laborioso. É, precisamente, para casos de uso como este onde um só sistema ou aplicação é composto por vários *containers* que existe a ferramenta *Docker Compose* (Hasan Ibrahim et al., 2021).

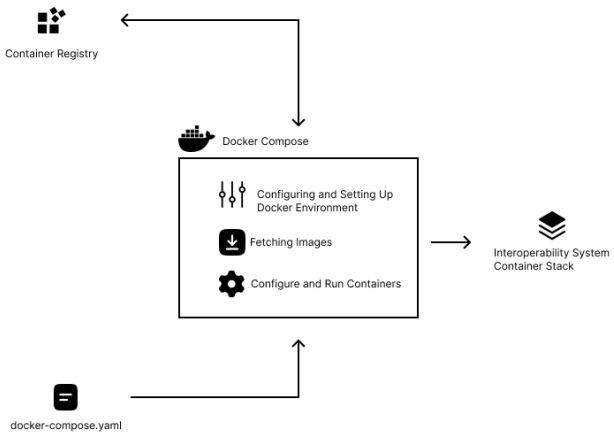

Figura 55 – Execução do sistema através de Docker Compose

Como representa a Figura 55 o consumo e execução de todas as componentes do sistema permanece igual, porém a utilização do *Docker Compose* permite a completa abstração da complexidade da busca, ou construção, configuração e execução das imagens pelo que pode ser tudo configurado através de um simples ficheiro de configuração dando origem ao sistema completo através de um só comando de execução.

Correndo assim o sistema é necessária uma predefinição para o *docker-compose.yaml* sendo que esta está representada na Tabela 7.

Tabela 7 - Predefinição da configuração Docker Compose

Serviço	**Obrigatoriedade**	**Imagem**	**Persistência por volume**	**Versão**
Microserviço de Interoperabilidade	Obrigatório	Imagem respetiva publicada no registo de *containers*	Não	última
Microserviço de Gestão de Mensagens	Obrigatório	Imagem respetiva publicada no registo de *containers*	Não	última
Microserviço Mirth Connect	Obrigatório	Imagem respetiva publicada no registo de *containers*	Não	última

Aplicação Web	Obrigatório	Imagem construída através do Dockerfile público no respetivo repositório	Não	última
MySQL	Não Obrigatório	Imagem pública disponibilizada no Docker Hub	Sim	8.0
RabbitMQ	Não Obrigatório	Imagem pública disponibilizada no Docker Hub	Sim	3.13.0
Mirth Connect	Não Obrigatório	Imagem pública disponibilizada no Docker Hub	Sim	4.3.0

Na Tabela acima é possível observar os detalhes acerca da configuração do sistema através do *Docker Compose* sendo que todos os componentes são obrigatórios menos as tecnologias em si. Como já foi mencionado os microserviços foram desenvolvidos de forma que a conexão à respetiva tecnologia alvo não necessite de ser local, sendo que o endereço de conexão é configurável. Também é possível compreender que há componentes do sistema que são montadas de forma diferentes pelo que é tudo configurável através do *docker-compose.yaml*. Por último também é possível verificar as versões escolhidas para as tecnologias. É muito importante que estas versões estejam bem definidas pelo que apenas é possível garantir a total compatibilidade do trabalho desenvolvido com uma e uma só versão das tecnologias alvo visto que, como foi anteriormente referido, versões novas geralmente significam mudanças significativas que partam algum tipo de funcionalidade em versões prévias.

6.11.6 Automatização de Testes Unitários

Tendo por base o conceito de Teste Unitário, a *framework* de testes *Jest* foi utilizada para desenvolver todos os testes de todas as componentes do sistema de interoperabilidade sendo que oferece um vasto leque de funcionalidades úteis para testar *software* como simulações, "espiões" e *snapshots* de testes anteriores (Moroz, 2019).

Tendo em conta a natureza repetitiva dos testes unitários é natural que o passo seguinte para a melhoria do desenvolvimento, qualidade e manutenção do *software* seja a sua automação. A automação dos testes unitários no código-fonte é o último passo para garantir que a consistência e prevenir regressões no *software* sendo que nenhuma mudança pode passar para revisão de pares sem que os testes passem todos. Neste aspeto a automação dos testes desenvolvidos com o *Jest* é tão simples como integrar uma execução da totalidade dos testes através desta ferramenta diretamente no fluxo de CI/CD. Esta integração encontra-se visível na Figura 56.

Figura 56 – Exemplo de integração de testes automáticos no CI/CD

6.11.7 Registo de *Containers*

Um registo de *containers* é uma parte importante no ciclo de desenvolvimento de um *software* destinado à integração e disponibilização pelo meio de virtualização por *container*. Trata-se de um repositório centralizado para o armazenamento e gestão de diversas imagens de *containers* (Galic, 2020) de forma segura e eficiente. São uma parte integral no ecossistema de *containers* pelo que permitem que os utilizadores possam, facilmente, encontrar e fazer uso das imagens conforme as suas necessidades. Segundo um estudo comparativo realizado em 2020 por Petar Galic (Galic, 2020) sugere várias vantagens para o seu uso e integração no ciclo de desenvolvimento de *software* como o acesso facilitado a imagens, capacidade de versionamento e histórico de imagens, armazenamento seguro de imagens e a replicação das mesmas por vários registos atingindo alta disponibilidade. Para além do mencionado a utilização de um registo central de imagens de *containers* ajuda a melhorar a eficiência de desenvolvimento, implementação e lançamento de *software* reduzindo o tempo e esforço tradicionalmente necessários para os atingir.

Fazendo uso das funcionalidades fornecidas pelo Github as diversas componentes passíveis de lançamento através de imagem de *container* do sistema de interoperabilidade fazem uso da plataforma *Github Container Registry* (GHCR) para o armazenamento, lançamento, histórico e gestão das suas respetivas imagens bem como versões de imagens.

Fazendo referência à Figura 50 o último passo na automação da entrada de mudanças para a *main branch* é a publicação de uma nova imagem no registo de *containers* GHCR. Para este fim foram

definidas duas regras que necessitam de ser cumpridas de forma a manter a consistência no lançamento do *software*:

- Qualquer imagem nova que seja publicada tem de ser catalogada com a versão exata da *release* que lhe deu origem de forma a permitir uma pesquisa e utilização por versão;
- Qualquer imagem que seja publicada necessita de ser catalogada com a expressão *"latest"* de forma a permitir uma pesquisa e utilização da imagem mais recente em qualquer ponto do desenvolvimento.

Estas regras foram pensadas com o intuito de permitir que um utilizador possa pesquisar e fazer uso das diferentes componentes do sistema através de uma versão específica desejada ou da versão mais recente de forma fácil e eficiente. Estas regras são verificáveis na Figura 57 onde é possível verificar que a versão mais recente do microserviço de interoperabilidade é a **v.1.1.0** e que esta também está catalogada com *lastest*. Versões anteriores também se encontram disponíveis como a **v1.0.0**.

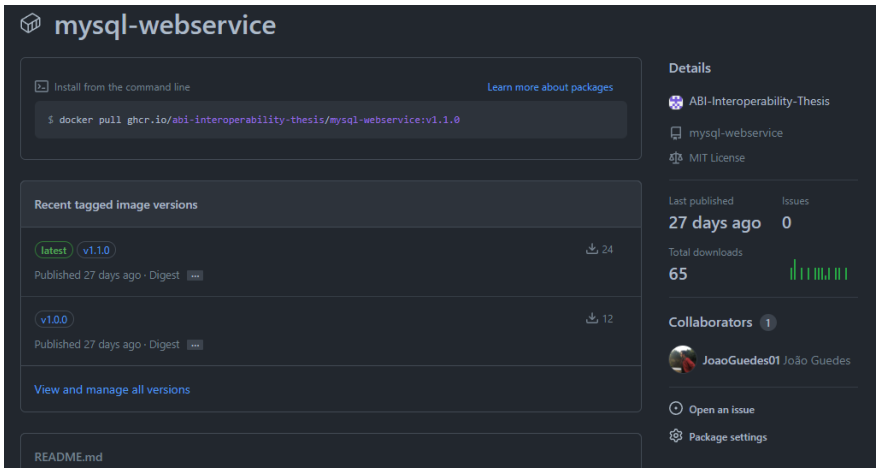

Figura 57 – Imagem do microserviço de interoperabilidade no GHCR

O catálogo das imagens é uma ferramenta muito poderosa e versátil pelo que todas as componentes de *software* do sistema de interoperabilidade podem e devem ter a sua própria versão mas outros usos mais criativos também são possíveis como a catalogação de todas as componentes com o rótulo extra de *abi-interoperability-v1* para simbolizar que todas as componentes nas respetivas versões de lançamento estão aptas e fazem parte da primeira versão do sistema de interoperabilidade garantindo o perfeito funcionamento entre elas.

6.11.8 Ambiente de Desenvolvimento e Testes

Fazendo uso da máquina virtual Ubuntu Linux (Potdar et al., 2020; W. Wang, 2022) disponibilizada pelo DSI foi montado um sistema de testes que visa a simulação da integração do sistema de

interoperabilidade num cenário de testes alusivo a um cenário de utilização real. Sendo assim esta máquina virtual irá correr o sistema de interoperabilidade na última versão de todos os seus componentes proporcionando uma imagem exata do estado a atual do *software* em qualquer fase do desenvolvimento e teste de novas funcionalidades ou correção de erros.

A implementação deste ambiente de testes teve por base o método e abordagem de teste de *software* SIL desenvolvido e demonstrado por Xiang Chen e Meranda Salem (Xiang et al., 2008). Ao longo do artigo os autores exploram o método de teste de produtos através da simulação *software in the loop* (SIL) em tempo real.

Uma simulação SIL consiste num método para simular e observar o comportamento de um sistema através da execução do mesmo num ambiente virtual desenvolvido para reproduzir as condições de execução num ambiente e condições de uso reais (Xiang et al., 2008). Esta abordagem permite efetuar, em tempo real, uma avaliação da qualidade global do sistema bem como a deteção de erros de uma forma precoce.

Com base neste conceito todas as mudanças realizadas e integradas na *dev branch* de cada um dos componentes do sistema de interoperabilidade são automaticamente refletidas e integradas num ambiente de testes.

Fazendo, novamente, referência à Figura 54 é possível observar que quando mudanças novas são integradas na *branch* de desenvolvimento (dev) é criada uma imagem do *software* que é implementada, de forma automática, no ambiente de testes. Esta integração é realizada com base na hospedagem de um *Github Actions Runner* diretamente na máquina virtual que é diretamente chamado pela *pipeline* de CI/CD e trata do processo de integração. Sendo assim a automatização é executada através de um *action runner* (Zhu et al., 2023) armazenado dentro da VM Linux e conectado à plataforma Github para construir uma imagem nova mais recente catalogada localmente como "ci" seguida da recriação completa do sistema através de um ficheiro docker-compose.yaml configurado para executar apenas imagens "ci". Este comportamento pode ser observado na Figura 58.

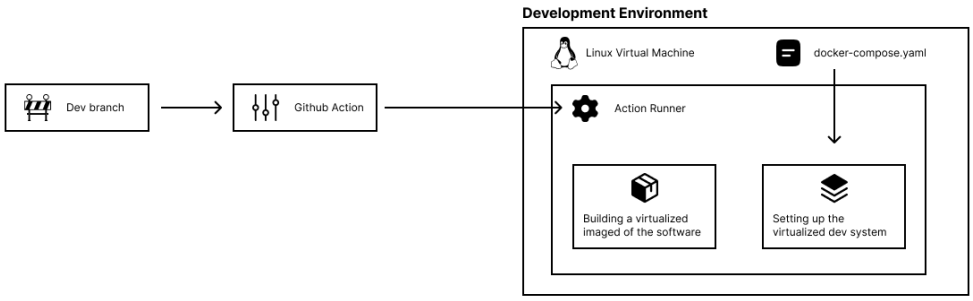

Figura 58 – Integração do sistema em ambiente de testes

Desta forma é possível refletir e testar todas as mudanças e estados do sistema de interoperabilidade na sua totalidade e explorar todas as suas funcionalidades simulando um ambiente de integração real.

Tabela 8 - Constituição Docker Compose no Ambiente de Testes

Serviço	**Imagem**	**Persistência por volume**	**Versão**
Microserviço de Interoperabilidade	Imagem respetiva presente no sistema com o rótulo de "ci"	Não	N/A
Microserviço de Gestão de Mensagens	Imagem respetiva presente no sistema com o rótulo de "ci"	Não	N/A
Microserviço Mirth Connect	Imagem respetiva presente no sistema com o rótulo de "ci"	Não	N/A
Aplicação Web	Imagem construída através do Dockerfile público no respetivo repositório na *branch* de desenvolvimento	Não	N/A
MySQL	Imagem pública disponibilizada no Docker Hub	Sim	8.0
RabbitMQ	Imagem pública disponibilizada no Docker Hub	Sim	3.13.0
Mirth Connect	Imagem pública disponibilizada no Docker Hub	Sim	4.3.0
PostgreSQL	Imagem pública disponibilizada no Docker Hub	Sim	16.1
Portainer	Imagem pública disponibilizada no Docker Hub	Sim	*latest*

Na Tabela 8 é possível observar a constituição do ficheiro docker-compose.yaml utilizado para a execução do sistema no ambiente de testes. É importante destacar que as imagens utilizadas para os componentes do sistema utilizam todas o rótulo de "ci". Isto garante a integração das atualizações mais recentes presentes em imagens com este rótulo. Outro aspeto a destacar é a utilização da plataforma *Portainer* para a gestão e monitorização de todos os elementos presentes no sistema na forma de *containers*.

7. TESTES

O teste da solução constituí uma parte integral no ciclo de vida de desenvolvimento de *software* sendo que tem como objetivo a avaliação e verificação da prontidão e aptitude da mesma. O processo de teste de *software* permite averiguar se uma determinada solução cumpre os requisitos e funções estabelecidos de forma a garantir a sua qualidade, fiabilidade e usabilidade aquando da entrega. É nesta fase que é ideal identificar e corrigir problemas e erros na solução pelo que esta constitui a fase onde estas atividades estão no seu estado menos dispendioso (Jamil et al., 2017).

Sendo assim foram identificadas três grandes categorias que irão auxiliar e fornecer uma ideia completa acerca do estado de funcionamento e refinamento do sistema sendo que foram testadas as suas características-base:

- Distribuição de componentes e ligação em rede;
- Funcionalidades chave do sistema;
- Avaliação da eficácia da solução.

Os testes desenvolvidos constituem um auxílio imprescindível na quantificação, qualificação e validação da conclusão e cumprimento de todos os requisitos estabelecidos para o sistema sendo que facultam uma forma clara e concisa de os avaliar iterativamente ao longo do ciclo de vida do desenvolvimento contínuo.

Os testes descritos na presente secção foram executados com base no sistema de interoperabilidade implementado no ambiente de testes sendo que este constitui a versão mais recente de todos os seus componentes.

7.1 Testes *e2e*

Partindo da noção de que os testes unitários já cobrem o teste de todas as funcionalidades base do sistema ao nível da unidade de código, o sistema já apresenta um grau considerável de robustez e consistência, sendo que todas as funcionalidades e métodos são automaticamente testados de forma isolada garantindo o seu funcionamento independente. Contudo, tendo em conta a complexidade e as diferentes partes móveis do sistema de interoperabilidade, é necessário testá-lo como um todo garantindo que tudo no sistema apresenta um comportamento esperado não apenas funções e unidades singulares de código.

Uma excelente forma de garantir que o sistema é testado na sua totalidade é através de testes *end-to-end* (E2E) (Raiküla & Applications, 2023) onde uma dada aplicação ou sistema é testado de forma completa de início a fim (ou de ponta a ponta) simulando casos de uso reais.

O Cypress é uma ferramenta muito útil e poderosa que permite a criação e automatização de testes E2E criada especificamente para o teste de aplicações baseadas em Web sendo que é um candidato perfeito para o desenvolvimento destes testes no sistema de interoperabilidade. A sua fácil implementação e utilização, rapidez e eficiência na execução de testes, conjunto rico de recursos e funcionalidades e integração ininterrupta em fluxos de CI/CD modernos torna-o uma das mais populares soluções para testes E2E no ecossistema de desenvolvimento moderno (Raiküla & Applications, 2023).

Sendo assim o Cypress foi utilizado para desenvolver e executar todos os testes E2E do sistema de interoperabilidade sendo que foi necessária a criação de um plano de testes a executar sendo que cada um visa testar uma funcionalidade específica do sistema. Estes testes, bem como os seus respetivos estados de funcionamento encontram-se abaixo representados na Tabela 9.

Tabela 9 - Plano de testes E2E

Teste	**Descrição**	**Componentes Testadas**	**Estado**
Validação da Dashboard	Validação das informações disponíveis na Dashboard de Admins e Clientes	Aplicação Web; MS Interoperabilidade	Sucesso
Validaçao de pedidos	Validação das informações disponíveis nos pedidos e pedido em particular	Aplicação Web; MS Interoperabilidade	Sucesso
Criação de canais Mirth	Criação do canal principal Mirth Connect e validação da sua criação	Aplicação Web; MS Interoperabilidade; MS Mirth Connect	Sucesso
Criação de um Modelo	Criação de um modelo, validação	Aplicação Web; MS Interoperabilidade;	Sucesso
Configuração de um modelo	Criação de mapeamentos, validadores e preprocessadores para o modelo (HL7 e FHIR)	Aplicação Web; MS Interoperabilidade;	Sucesso
Teste do modelo	Teste do funcionamento do modelo com uma mensagem HL7 e FHIR pre-feita	Aplicação Web; MS Interoperabilidade; MS Mirth Connect; MS Gestão de mensagens	Sucesso
Atribuição de permissões a clientes	Atribuição e remoção de permissões de utilização dos modelos	Aplicação Web; MS Interoperabilidade; MS Mirth Connect; MS Gestão de mensagens	Sucesso
Criação de Mapeamentos de Atributos	Criação e validação de mapeamento de atributos	Aplicação Web; MS Interoperabilidade; MS Mirth Connect; MS Gestão de mensagens	Sucesso
Criação Validadores	Criação e validação de uso de validadores em modelos	Aplicação Web; MS Interoperabilidade; MS Mirth Connect; MS Gestão de mensagens	Sucesso
Criaçao de Preprocessadores	Criação e validação de uso de preprocessadores em modelos	Aplicação Web; MS Interoperabilidade; MS Mirth Connect; MS Gestão de mensagens	Sucesso
Criação e resposta de *issues*	Criação, validação, resposta e mudança de estado de *issues*	Aplicação Web; MS Interoperabilidade;	Sucesso
Eliminação de recursos	Eliminação de todos os recursos criados nos testes anteriores	Aplicação Web; MS Interoperabilidade; MS Mirth Connect;	Sucesso

A partir deste plano de testes foi possível automatizar, através do Cypress, todas as funcionalidades, num geral, do sistema de interoperabilidade simulando casos de uso reais testando assim o sistema de ponta a ponta. No momento da escrita do presente documento estes são os testes que garantem o bom funcionamento de todo o sistema e encapsulam toda a sua funcionalidade e casos de utilização básica sendo que todos passaram com sucesso cumprindo os requisitos funcionais estabelecidos.

7.2 Testes de Arquitetura

Um dos pontos principais do sistema de interoperabilidade está na sua facilidade de integração e modularidade dos seus componentes sendo que, independentemente da sua disposição, o sistema deve sempre apresentar um comportamento correto e previsível.

Uma excelente forma de efetuar este tipo de testes é com testes de sanidade (Reshma et al., 2022) onde as funcionalidades básicas de um *software* são testadas de forma a averiguar se estas apresentam o comportamento esperado. Estes testes têm como objetivo a identificação de defeitos importantes que sejam capazes de impedir a utilização da solução da forma concessionada. Maior parte da funcionalidade básica, em termos unitários, já está coberta através dos constantes testes unitários que impedem a integração de alterações de rutura, porém no contexto de teste de arquitetura os testes necessitam de intervenção manual.

Para este efeito foram testadas várias configurações da arquitetura do sistema de interoperabilidade sendo que todas foram submetidas ao mesmo plano de testes representado pela Tabela 10.

Tabela 10 - Plano de testes de arquitetura

Teste	Descrição	Relevância
Comunicação entre Microserviços (T1)	Os diferentes microserviços conseguem estabelecer conexão entre si	Não
Comunicação com as tecnologias do sistema (T2)	Os microserviços conseguem estabelecer conexão com as diferentes tecnologias do sistema	Não
Testes E2E (T3)	O sistema passou nos testes e2e	Não

Este plano de testes foi executado em todas as diferentes configurações do sistema sendo que foram atingidos elevados graus de robustez e flexibilidade de solução tendo todos os cenários apresentado grande capacidade de adaptabilidade e funcionamento. Estes resultados encontram-se listados na Tabela 11.

Tabela 11 - Resultados dos testes de arquitetura

Configuração	Descrição	T1	T2	T3

Execução local	Todos os componentes correm na mesma máquina	Sucesso (+/- 4 ms)	Sucesso (+/- 4 ms)	Sucesso (3:20 mins)
Execução remota de microserviços	Microserviços correm na máquina B enquanto as tecnologias correm na máquina A	Sucesso (+/- 6 ms)	Sucesso (+/- 50 ms)	Sucesso (3:37 mins)
Execução remota de tecnologias	Microserviços correm na máquina A enquanto as tecnologias correm na máquina B	Sucesso (+/- 5 ms)	Sucesso (+/- 64 ms)	Sucesso (3:53 mins)
Execução remota da aplicação web	Backend do sistema corre na máquina A enquanto a aplicação web corre na máquina B	Sucesso (+/- 6 ms)	Sucesso (+/- 6 ms)	Sucesso (3:21 mins)

Como é possível verificar pelos resultados o sistema de interoperabilidade apresenta um elevado grau de flexibilidade e uma robustez capaz de assegurar o seu funcionamento independentemente da forma como o utilizador final pretender integrá-lo sendo que em todas as configurações testadas o sistema apresentou sempre um comportamento esperado e razoavelmente rápido. Como seria de esperar a configuração que apresenta a comunicação mais rápida é a local sendo que todas as componentes se encontram no mesmo ambiente e máquina sendo que a mais lenta é a execução remota de tecnologias pela razão inversa.

Depois de realizados os testes as diferentes configurações apresentam valores de sucesso idênticos, porém valores de rapidez de conexão diferentes sendo que existe uma configuração mais rápida e mais lenta. Estes valores são, contudo, negligenciáveis devido ao seu pequeno intervalo de diferença explicável pela mudança de ambiente e respetivo aumento compreensível na latência das comunicações. No que diz respeito ao sistema nada pode ser feito em relação a aumentos de latência por hospedagem em máquinas diferentes.

Sendo assim os requisitos não funcionais do sistema de interoperabilidade foram considerados cumpridos.

7.3 Teste de modelos

Finalmente é possível testar a eficácia do sistema através da sua utilização direta para a execução de três modelos de *machine learning*. Para a execução destes modelos foi necessária a sua documentação e integração no sistema de interoperabilidade pelo que foram testados utilizando 20 recursos HL7 oriundas do centro Hospitalar do Tâmega e Sousa.

Todos os atributos dos modelos foram devidamente documentados e mapeados utilizando a documentação do HL7 e FHIR de forma que a informação necessária para que estes possam ser executados seja representada da forma mais correta em ambos os padrões.

Como é de esperar nem toda a informação estará disponível nas mensagens utilizadas pelo que foi necessário proceder a algumas modificações das mesmas ou a criação e utilização de mapeamentos personalizados.

Os seguintes testes demonstram que mesmo não havendo informação suficiente nos recursos de interoperabilidade o sistema consegue ser configurado de forma a garantir a execução dos modelos alvo. Sendo assim, é possível definir padrões e normas gerais que, quando devidamente configuradas, permitem a execução consistente dos diversos modelos disponíveis num sistema ABI através de recursos de interoperabilidade. Estas normas ficam ao critério do(s) administrador(es) do sistema pelo que este(s) está(ão) encarregue(s) de integrar novos modelos com o sistema de interoperabilidade e fazer a sua manutenção.

Sendo que a maior parte dos atributos dos modelos a testar se focam muito em informações pessoais acerca do paciente em questão, o suporte de HL7/FHIR irá também focar-se em tipos de recursos capazes de integrar esta informação. Sendo assim os recursos de teste são do tipo ADT (Admit, Discharge & Transfer) (Noumeir, 2019) sendo que estes são utilizados para comunicar dados demográficos de pacientes, informações sobre as suas visitas e o seu estado.

7.3.1 Preditor de Hospitalização

O modelo de hospitalização foi o primeiro modelo a ser testado e integrado no sistema de interoperabilidade pelo que foi utilizado como modelo de teste para todo o desenvolvimento do mesmo. O modelo de previsão de hospitalização consiste num modelo preditivo que, através de 11 atributos, é capaz de prever com algum grau de certeza se um paciente irá ou não ficar hospitalizado na instituição de saúde que visitou.

Para a integração deste modelo foi necessário o estudo cuidado de todos estes atributos de forma a poder fazer uma ligação clara e correta aos atributos dos recursos interoperabilidade (HL7 & FHIR). Estes atributos, as suas descrições e respetivos mapeamentos HL7 estão representados na tabela 12 ao passo que os mapeamentos FHIR estão na tabela 13.

Tabela 12 - Mapeamentos HL7 dos atributos do modelo predição de hospitalização

Descrição	Campo HL7	Tipo de Mensagem	Triggers	Validação	Processamento
Número de Identificação do episódio	PV1-19.1 (Visit Number)	ADT	A01,A02,A03,A04	Validação numérica	Direto
Hora de admissão (segundos)	PV1-44.1 (Admit Date/Time)	ADT	A01,A02,A03,A04	Validação de Data	Cálculo de segundos

Código de causa médica	DG1-3.1 (Diagnosis Code)	ADT	A01,A02,A03,A04	Validação de caracteres	Correspondência com BD
Código de especialidade médica	PV1-9.7 (Consulting Doctor Degree)	ADT	A01,A02,A03,A04	Validação de caracteres	Correspondência com BD
Código de Prioridade (1-6)	PV1-2 (Patient Class)	ADT	A01,A02,A03,A04	Validação de caracteres	Correspondência com BD
Idade do paciente	PID-7 (Date of Birth)	ADT	A01,A02,A03,A04	Validação de Data	Cálculo de idade
Código de traigem	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ORU	R01,R02	Validação de caracteres	Correspondência com BD
Sexo de um paciente (1 – Male, 2 - Female)	PID-8.1 (Administrative Sex)	ADT	A01,A02,A03,A04	Validação de caracteres	Correspondência com BD
Código Municipal	PID-11.9 (Temporary Location)	ADT	A01,A02,A03,A04	Validação de caracteres	Correspondência com BD
Número de lotação no momento da admissão	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ORU	R01,R02	Validação numérica	Direto
Duração da estimada da estada (segundos)	PV2-10.1 (Estimated Length of inpatient Stay)	ADT	A01,A02,A03,A04	Validação numérica	Conversão de dias para segundos

Como é possível verificar na tabela 12 todos os atributos do modelo de hospitalização são mapeáveis a atributos HL7 com bastante exatidão sendo que cada um deve ser tratado e manipulado de forma diferente e controlada. O sistema de interoperabilidade foi construído com estas nuances em mente sendo que qualquer operação de processamento ou validação é criável e aplicável a qualquer atributo HL7 que seja necessário. Muitos dos casos são de fácil integração como o número de visita, mas outros já necessitam de alguma transformação como o cálculo da idade através da data de nascimento.

Ainda assim existem casos onde a informação necessária simplesmente não consta no padrão HL7 como é o caso do COD_VIA_VERDE ou a afluência sendo que as mensagens ORU juntamente com o campo OBX foram reservadas para estes casos sendo que é necessária a validação e configuração do campo OBX-3.1 para a identificação da informação e OBX-5.1 para a informação em si. Estes casos constituirão apenas uma minoria dos casos devido à abrangência do padrão HL7 porém são necessários para garantir o máximo de compatibilidade com qualquer modelo/atributo.

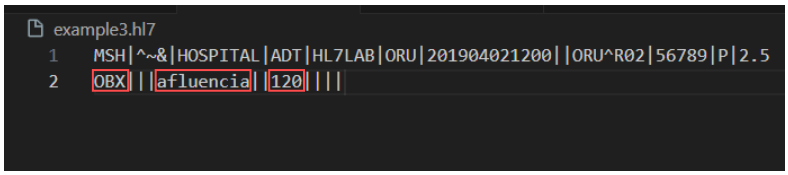

Figura 59 – Caso de observação HL7

Na Figura 59 encontra-se representado o caso do atributo de afluência que não possui uma ligação direta ao padrão HL7 e impossibilita o seu mapeamento direto. Neste caso para o atributo afluência está definido o valor de 120. Sendo assim para o caso extraordinário de não haver mapeamentos possíveis o sistema de interoperabilidade irá validar o identificador da observação bem como o seu valor.

Tabela 13 - Mapeamentos FHIR dos atributos do modelo predição de hospitalização

Descrição	**Campo FHIR**	**Tipo de Mensagem**	**Validação**	**Processamento**
Número de Identificação do episódio	identifier	Encounter	Validação Numérica	direto
Hora de admissão (segundos)	Period	Encounter	Validação de data	Conversão de data para segundos
Código de causa médica	Code	Condition	Validação de caracteres	Correspondência com BD
Código de especialidade médica	Qualification	Practitioner	Validação de caracteres	Correspondência com BD
Código de Prioridade (1-6)	Class	Encounter	Validação de caracteres	Correspondência com BD
Idade do paciente	BirthDate	Patient	Validação de caracteres	Conversão de data de nascimento para idade
Código de traigem	Severity	Condition	Validação de caracteres	Correspondência com BD
Sexo de um paciente (1 – Male, 2 - Female)	Gender	Patient	Validação de caracteres	Correspondência com BD
Código Municipal	Address	Patient	Validação de caracteres	Correspondência com BD
Número de lotação no momento da admissão	operationalStatus	Location	Validação Numérica	Direto
Duração da estimada da estada (segundos)	Length	Encounter	Validação Numérica	Conversão para segundos

Tabela 14 – Resultados dos testes para o modelo de predição de hospitalização

Cenário	**Descrição**	**Resultado**

Execução das mensagens	As mensagens foram executadas *as is*	Sucesso (84% de compatibilidade)
Execução das mensagens atualizadas	Atualização das mensagens para conterem todas as informações necessárias	Sucesso (100% de compatiblidade)
Execução das mensagens com mapeamentos personalizados	As mensagens forma executadas *as is* mas o sistema está configurado com mapeamentos personalizados ás informações que constam nas mensagens	Sucesso (100% de compatibilidade)
Execução de mensagens Erróneas	As mensagens foram alteradas para serem propositadamente erróneas	Sucesso (0% de Erros)

Na tabela 14 estão representados os resultados obtidos nos testes realizados com o conjunto de mensagens de teste sendo que é possível afirmar que todos os testes foram realizados com sucesso. Nos diversos cenários de teste o sistema comportou-se conforme o esperado sendo que mensagens incompletas, ou que não cumprem os requisitos informacionais do modelo, não irão acionar nenhum modelo. A configuração dos modelos é totalmente flexível sendo que é sempre possível a integração de qualquer modelo e tratamento de qualquer tipo de informação. É de realçar que tratamento de erros do sistema é robusto não havendo erros aquando do tratamento de mensagens mal construídas ou erróneas.

Por último, os mapeamentos escolhidos parecem ser bastante bons sendo que as mensagens de teste contêm 84% da informação necessária, contudo há sempre a possibilidade da escolha de melhores mapeamentos fruto de pesquisas e estudos futuros.

7.3.2 Predição de Diabetes

O modelo de predição de diabetes foi um modelo obtido através da plataforma *Kaggle* onde o autor documentou extensivamente o processo do seu desenvolvimento através de um "caderno"/*notebook* Jupyter. Este modelo conta com 8 atributos relativos à informação de saúde de um determinado paciente e através deles consegue fazer uma boa estimativa em relação à probabilidade do desenvolvimento de diabetes.

Á semelhança do modelo de predição de hospitalização, para o modelo de predição de diabetes também foi necessário estudar todos os 8 atributos e tentar fazer a ligação mais simples e consistente aos respetivos congéneres em HL7/FHIR. Estes atributos e respetivas ligações encontram-se representados na seguinte tabela 15.

Tabela 15 - Atributos e distribuições HL7 para predição de diabetes

Descrição	**Campo HL7**	**Tipo de Mensagem**	**Triggers**	**Validação**	**Processamento**
Número de vezes que o	OBX-3.1 (Observation Identifier)	ADT	A01,A02,A03,A04	Validação numérica	Direto

paciente esteve grávido	OBX-5.1 (Observation Value)				
Concentração de glucose	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ADT	A01,A02,A03,A04	Validação numérica	Direto
Pressão sanguínea diastólica	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ADT	A01,A02,A03,A04	Validação numérica	Direto
Espessura da dobra do trícep	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ADT	A01,A02,A03,A04	Validação numérica	Direto
Insulina sérica de 2 horas	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ADT	A01,A02,A03,A04	Validação numérica	Direto
Índice de Massa corporal	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ADT	A01,A02,A03,A04	Validação numérica	Direto
Função do pedigree de diabetes	OBX-3.1 (Observation Identifier) OBX-5.1 (Observation Value)	ADT	A01,A02,A03,A04	Validação numérica	Direto
Idade do paciente	PID-7 (Date of Birth)	ADT	A01,A02,A03,A04	Validação de Data	Conversão de data para idade

Como é possível observar os atributos deste modelo são altamente específicos e não são facilmente retratados através do padrão HL7 V2 pelo que foi necessário recorrer à observação na maioria deles. No entanto é sempre possível fazer uma análise mais profunda onde uma combinação de um campo e método de preprocessamento podem resultar numa melhor estratégia de extração e conversão de dados. A distribuição equivalente para o padrão HL7 FHIR encontra-se na tabela 16.

Tabela 16 - Atributos e distribuições FHIR para predição de diabetes

Descrição	**Campo FHIR**	**Tipo de Mensagem**	**Validação**	**Processamento**
Número de vezes que o	ValueInteger	Observation	Validação numérica	Direto

paciente esteve grávido				
Concentração de glucose	valueRatio	Observation	Validação numérica (rácio)	Direto
Pressão sanguínea diastólica	valueRatio	Observation	Validação numérica (rácio)	Direto
Espessura da dobra do trícep	valueQuantity	Observation	Validação numérica	Direto
Insulina sérica de 2 horas	valueRatio	Observation	Validação numérica (rácio)	Direto
Índice de Massa corporal	valueRatio	Observation	Validação numérica (rácio)	Direto
Função do pedigree de diabetes	valueString	Observation	Validação de caracteres	Direto
Idade do paciente	birthDate	Patient	Validação de Data	Conversão de data para idade

Tabela 17 - Resultados dos testes para o modelo de predição de diabetes

Cenário	**Descrição**	**Resultado**
Execução das mensagens	As mensagens foram executadas *as is*	Sucesso (12,5% de compatibilidade)
Execução das mensagens atualizadas	Atualização das mensagens para conterem todas as informações necessárias	Sucesso (100% de compatiblidade)
Execução das mensagens com mapeamentos personalizados	As mensagens forma executadas *as is* mas o sistema está configurado com mapeamentos personalizados ás informações que constam nas mensagens	Sucesso (100% de compatibilidade)
Execução de mensagens Erróneas	As mensagens foram alteradas para serem propositadamente erróneas	Sucesso (0% de Erros)

Analisando a tabela 17, correr as mensagens sem quaisquer alterações apenas retorna uma taxa de sucesso de 12,5% correspondente apenas à data de nascimento/idade do paciente. Sendo que os restantes atributos consistem em observações específicas para uso com o modelo. Por esta razão há um aumento para 100% quando as mensagens são alteradas para passar a conter as informações em falta na forma de observações.

Estes testes revelam que o grau de eficácia do uso de recursos de interoperabilidade em conjunção com modelos de *machine learning* depende quase unicamente do modelo a ser utilizado sendo que os dados necessários para o correr podem, ou não, ser facilmente representados neles. Os casos cobertos abordam 2 casos distintos, onde os atributos são facilmente representados através de campos HL7 e onde não o são. Em qualquer um dos casos foi possível alcançar a integração completa devido à grande cobertura do HL7 mas também à sua flexibilidade.

8. Conclusões e Reflexões

O presente documento conta com 126 capítulos e subcapítulos que resumem de uma forma compreensiva o estado atual da tecnologia de interoperabilidade no setor da saúde bem como a sua potencial aplicação com sistemas ABI. Também é, nestes capítulos, abordado com bastante detalhe todo o processo de desenvolvimento de um protótipo de sistema que visa fazer interface com sistemas ABI com base em recursos de interoperabilidade.

Através da revisão de literatura e análise ao estado da arte, efetuados nas fases iniciais deste projeto de dissertação, foi possível ter uma noção clara e extensa acerca das práticas e soluções mais adotadas na atualidade em relação à implementação e integração de sistemas ABI em diversas áreas, nomeadamente na saúde. Esta análise foi fundamental para entender como diferentes organizações de saúde têm implementado sistemas ABI. Através da combinação de técnicas de Data Mining, previsão, otimização e adaptabilidade, os sistemas ABI respondem com um grau elevado de sucesso aos requisitos exigentes presentes na área da saúde. Porém a sua integração é bastante complicada, pouco flexível e demorada sendo que as soluções estudadas foram feitas à medida limitando-se apenas ao seu âmbito de integração. Estes desafios, juntamente com diversas questões referentes à segurança dos dados em circulação, constituíram as motivações base para este projeto bem como para todo o sistema desenvolvido.

O setor da saúde trata-se de um caso particular pelo que o conceito de interoperabilidade não é apenas relevante, mas também vital, para o desenvolvimento, integração e comunicação com os seus diversos sistemas e plataformas existentes. Através da adoção de um mesmo padrão de comunicação e representação de dados é possível garantir a comunicação e integração de qualquer sistema no ecossistema da saúde. Para o presente projeto foram considerados e estudados os padrões mais populares e amplamente adotados, HL7 e FHIR.

Com base nos desafios e limitações levantadas no estado da arte, bem como a universalidade dos padrões de interoperabilidade existentes, propõe-se uma camada adicional ao modelo tradicional de ABI. Esta camada de interoperabilidade visa agilizar e integrar a informação contida em diversos recursos tornando-a compatível e utilizável pelos modelos característicos utilizados em sistemas ABI.

Este Sistema de interoperabilidade foi idealizado com base no pressuposto de que o cliente utiliza algum dos padrões de interoperabilidade estudados bem como a existência de uma interface para a execução dos diversos modelos. Utilizando diversos padrões de desenvolvimento de software o sistema é composto por três grandes pilares, a extração exata de informação dos recursos de interoperabilidade, a validação

do seu conteúdo e pré processamento para garantir a execução correta nos modelos disponíveis. Estas etapas ocorrem de forma sequencial seguindo o padrão de "Pipeline", sendo que os dados passam por diversas fases de transformação em que qualquer falha nalguma destas corresponde ao cancelamento de todo o processo. As etapas em si seguem o padrão de "Fábrica" pelo que existe um algoritmo central que é executado de forma modular através de diferentes configurações.

Para lidar com os recursos HL7 foi imediatamente considerada a plataforma Mirth Connect e para a comunicação entre sistemas foi proposto o método de *message queue*. Devido à existência destes diferentes marcos de operação o sistema proposto segue a arquitetura de microserviços de forma a poder modularizar e distribuir âmbitos, ambientes e responsabilidades diferentes e limitadas por diferentes componentes separados.

Nesta fase também foram levantados todos os recursos, funcionais e não funcionais, do sistema que serviram como KPIs para o progresso e desenvolvimento do protótipo.

Para o desenvolvimento do protótipo do sistema de interoperabilidade foi utilizado uma adaptação da metodologia SCRUM para uso singular onde todas as tarefas e responsabilidades foram adotadas por mim. Isto permitiu manter o trabalho organizado e incrementável através de pequenas, mas controladas, iterações utilizando a plataforma Jira.

Foi desenvolvido um modelo de dados tanto para estruturas relacionais como não relacionais de Bases de Dados segundo a terceira forma normal de representação de dados. Esta abordagem permite a integração futura do sistema com outros tipos de bases de dados e possibilidade de integração mais ampla e flexível.

Foram estudados diversos padrões e melhores práticas para o desenvolvimento de microserviços sendo que a estrutura MVC foi adotada. Isto significa que todos os microserviços em conjunto assumem as responsabilidades de *Model*, *View* e *Controller*.

Para garantir a uniformidade e consistência em todos os microserviços foi desenvolvido um *template* com base nas melhores práticas de desenvolvimento incluindo documentação, separação de ficheiros, testes, configuração por variáveis de ambiente, virtualização e integração contínua. Com base neste *template* foram desenvolvidos três microserviços cada um responsável por um marco do sistema, interoperabilidade, Mirth Connect e *message queue* bem como uma interface gráfica.

Todos estes componentes foram desenvolvidos fazendo uso da tecnologia de controlo de versão Git e o código hospedado/iterado na plataforma Github. Dando uso ao controlo de versões, mudanças significativas são catalogadas de acordo com o padrão de controlo de versões semântico. Este processo foi automatizado por meio de processos de integração e implantação contínuas através da plataforma de

Github Actions. Todo o código desenvolvido é catalogado e libertado na forma de virtualização por Docker sendo que estas imagens virtuais estão hospedadas no Github Container Registry.

De forma a agilizar o processo de lançamento de alterações e versões foi também desenvolvida uma estratégia de *branching* composta por uma *branch* main e uma *branch* dev sendo que é sempre necessária a criação de *pull requests* para o incremento a qualquer uma destas *branches*. Sendo assim o fluxo será sempre de uma *feature branch* para a *branch* dev e só depois para a *branch* master. Isto permite a segmentação e teste controlado de cada iteração feita.

Cada microserviço também conta com uma série de testes unitários que necessitam de correr com sucesso no processo de CI/CD para a integração de mudanças nas *branches* principais.

Foram também desenvolvidos testes de ponta a ponta (*end-to-end*) de forma a poder testar o sistema como um todo e não apenas uma só parte utilizando a ferramenta de testes Cypress por 12 cenários de teste diferentes. Estes testes são também um requisito obrigatório na integração de alterações da *branch* dev para master.

De forma a testar o sistema de forma manual foram reunidas algumas mensagens reais e foram testadas com 2 modelos, ou as suas representações, obtendo graus de sucesso diferentes pelo que diferentes modelos carecem de tipos de dados diferentes. Nenhum dos modelos foi capaz de ser executado utilizando apenas as mensagens tal e qual elas foram proporcionadas devido ás limitações dos próprios recursos de interoperabilidade, mas em ambos os casos o sistema foi capaz de ser adaptado e configurado para o estilo e informação contida nelas. Acima de tudo estes testes provaram e cimentaram a capacidade de adaptação e flexibilidade do sistema desenvolvido.

Por último foram levantadas e estudadas as vulnerabilidades mais comuns em aplicações Web seguindo os padrões estabelecidos pela OWASP sendo que o sistema foi construído com elas em mente. Sendo assim o *design* e desenvolvimento do sistema respeita e previne todas as vulnerabilidades estudadas devido à programação defensiva que foi, ativamente, praticada e ao seguimento das melhores práticas de programação.

Como resultado foi desenvolvido todo um protótipo funcional do sistema proposto de fácil integração (por virtualização Docker), seguro e bem estruturado que permite a interface ininterrupta com diversos modelos de sistemas ABI através de recursos de interoperabilidade.

Este trabalho de investigação permitiu-me explorar o estado da arte, não apenas na integração de sistemas ABI no setor na saúde, mas também das melhores práticas em todo o processo de desenvolvimento de software.

Através do trabalho realizado é possível afirmar que todos os objetivos foram atingidos pelo que o sistema apenas interage com modelos já implementados, por outras palavras na sua última fase da metodologia CRISP-DM. Foram investigadas e implementadas formas de estabelecer comunicação entre sistemas na forma de *message queue.* Todo o Sistema foi desenvolvido através do estudo das melhores práticas de programação em todos os aspetos do mesmo sendo que foram estudados os protocolos e padrões da indústria da saúde bem como integrados no Sistema de interoperabilidade.
Desta forma é sim possível promover a formulação e implementação de sistemas ABI no contexto de sistemas de saúde através de uma arquitetura de interoperabilidade respondendo assim à questão de investigação proposta no capítulo 1.2 do presente documento.

9. TRABALHO FUTURO

O presente trabalho de investigação e desenvolvimento iniciou e fez grandes progressos na direção da integração de sistemas ABI por meio da interoperabilidade no setor da saúde, porém existem claras limitações e imensa margem para melhoria uma vez que as bases de investigação estão já cobertas.
Tendo em conta o estado atual deste projeto e do sistema foram reunidas algumas iniciativas para trabalho futuro e melhoria:

- **Treino de Modelos –** O sistema apenas faz interface com modelos já treinados e implementados pelo que conta já com a última fase do CRISP-DM. Outro uso para os recursos de interoperabilidade seria a extração de informação para o treino de modelos em fases iniciais do seu desenvolvimento. Por outras palavras o sistema de interoperabilidade iria assumir as primeiras fases de extração e transformação de dados do CRISP-DM;
- **Novos recursos de Interoperabilidade –** Apenas HL7 e FHIR são suportados de momento pelo que isto constitui uma limitação rígida para integração com outro tipo de sistemas que utilize outro tipo de notação ou padrão para comunicação;
- **Suporte de novos formatos de dados –** Apenas o formato original HL7 e JSON para FHIR são suportados no sistema atual pelo que será necessário expandir este suporte para outros formatos de dados pelo que nem todos os sistemas no setor da saúde estão restringidos a estes formatos;
- **Suporte de novos tipos de modelos –** O sistema atual apenas suporta modelos de predição e otimização com base em *inputs* numéricos e/ou alfabéticos o que limita bastante os modelos

integráveis. Modelos capazes de utilizar imagens ou ficheiros de som como *input* seriam, por exemplo, uma ótima adição ao suporte do sistema;

- **Estudo e avaliação de segurança –** O sistema foi desenvolvido utilizando como guia os padrões de segurança estabelecidos pela OWASP pelo que as preocupações base de segurança estão, à partida, cobertas e exploradas. Porém existem muitas outras preocupações de segurança a ser exploradas em tarefas direcionadas para a avaliação desta de uma forma mais profunda;
- **Suporte de novas tecnologias –** Apenas MySQL, para base de dados, e RabbitMQ, para gestão de mensagens em fila são suportadas, porém o sistema foi desenvolvido visando a troca e suporte de múltiplas tecnologias para estes fins. Seria necessário o estudo das novas tecnologias a adotar bem como o seu suporte feito manualmente de forma similar ao já implementado;
- **Casos de uso para o Mirth Connect –** O Mirth Connect está apenas a ser utilizado para o *parsing* de mensagens HL7 pelo que será necessária a exploração das diferentes funções e capacidades desta tecnologia que sejam relevantes à interoperabilidade;
- **Combinação de múltiplos atributos –** A versão atual do sistema consegue extrair um e um só atributo, validar a sua informação e processá-la de uma forma pré configurada. Esta é a abordagem básica para atingir a interoperabilidade, porém alguns casos podem carecer de extrações mais complexas como por exemplo o cálculo do BMI que necessita de dividir o peso de um paciente pela sua altura ao quadrado. Para este caso seria necessária a extração de 2 atributos distintos, a altura e peso do paciente, e a sua utilização num cálculo.

Em suma a presente investigação e projeto representam um avanço significativo para a integração de sistemas ABI por meio da interoperabilidade no setor da saúde e estabelece uma base sólida e bem estruturada para futuros desenvolvimentos. As iniciativas propostas para trabalhos futuros destacam-se como oportunidades promissoras para o aprofundamento e expansão do impacto deste trabalho de investigação. É, portanto, importante que novos trabalhos de investigação se alinhem com as recomendações aqui reunidas, garantindo assim a evolução contínua e eficaz desta área de estudo e desenvolvimento.

10. BIBLIOGRAFIA

Aceto, G., Persico, V., & Pescapé, A. (2020). Industry 4.0 and Health: Internet of Things, Big Data, and Cloud Computing for Healthcare 4.0. In *Journal of Industrial Information Integration* (Vol. 18). Elsevier B.V. https://doi.org/10.1016/j.jii.2020.100129

Aljabri, M., Alhetelah, B., Aldossary, M., Althubiany, M., Al-Homeed, N., Alotaibi, O., & Alsaqer, S. (2022). *Testing and Exploiting Tools to Improve OWASP Top Ten Security Vulnerabilities Detection*. https://doi.org/10.1109/cicn.2022.137

Aljabri, S., Almalki, A., & Altalhi, A. (2023). Cyber Security Risks for Global Businesses and Solutions Expected. In *International Journal of Multidisciplinary Innovation and Research Methodology (IJMIRM)* (Vol. 2, Issue 2).

AlQudah, A. A., Al-Emran, M., & Shaalan, K. (2021). Medical data integration using HL7 standards for patient's early identification. *PLoS ONE*, *16*(12 December). https://doi.org/10.1371/journal.pone.0262067

Ashfaq, A., & Nowaczyk, S. (2019). *Machine learning in healthcare - a system's perspective*. https://doi.org/10.1145/1235

Ayaz, M., Pasha, M. F., Alahmadi, T. J., Abdullah, N. N. B., & Alkahtani, H. K. (2023). Transforming Healthcare Analytics with FHIR: A Framework for Standardizing and Analyzing Clinical Data. *Healthcare (Switzerland)*, *11*(12). https://doi.org/10.3390/healthcare11121729

Azevedo, A., & Santos, M. F. (2008). *KDD, SEMMA AND CRISP-DM: A PARALLEL OVERVIEW*.

Bach-Nutman, M. (2020a). *Understanding The Top 10 OWASP Vulnerabilities*.

Bach-Nutman, M. (2020b). *Understanding The Top 10 OWASP Vulnerabilities*.

Barsoti, N., & Gibertoni, D. (2020). IMPACTO QUE O SEQUELIZE TRAZ PARA O DESENVOLVIMENTO DE UMA API CONSTRUÍDA EM NODE.JS COM EXPRESS.JS. *Revista Interface Tecnológica*, *17*(2), 231–243. https://doi.org/10.31510/infa.v17i2.964

Cerny, T., Donahoo, M. J., & Pechanec, J. (2017). Disambiguation and comparison of SOA, microservices and self-contained systems. *Proceedings of the 2017 Research in Adaptive and Convergent Systems, RACS 2017*, *2017-January*, 228–235. https://doi.org/10.1145/3129676.3129682

Chacon, S., & Straub, B. (2023). *Pro Git*.

Conventional Commits. (2023). https://www.conventionalcommits.org/

Cosentino, V., Izquierdo, J. L. C., & Cabot, J. (2017). A Systematic Mapping Study of Software Development with GitHub. *IEEE Access*, *5*, 7173–7192. https://doi.org/10.1109/ACCESS.2017.2682323

Dann, A., Hermann, B., & Bodden, E. (2023). *UPCY: Safely Updating Outdated Dependencies*.

Daun, M., Grubb, A. M., Stenkova, V., & Tenbergen, B. (2023). A systematic literature review of requirements engineering education. *Requirements Engineering*, *28*(2), 145–175. https://doi.org/10.1007/s00766-022-00381-9

Decan, A., & Mens, T. (2019). What Do Package Dependencies Tell Us About Semantic Versioning? In *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*.

Decan, A., Mens, T., Mazrae, P. R., & Golzadeh, M. (2022). *On the Use of GitHub Actions in Software Development Repositories*. https://doi.org/10.5281/zenodo.6634682

Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, *150*, 77–97. https://doi.org/10.1016/j.jss.2019.01.001

Doglio, F. (2018). REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development, Second Edition. In *REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development, Second Edition*. Apress Media LLC. https://doi.org/10.1007/978-1-4842-3715-1

Dorai, R., & Kannan, V. (2011). SQL Injection-Database Attack Revolution and Prevention. In *Journal of International Commercial Law and Technology* (Vol. 6).

Dynamic Schemes for Speculative Execution of Code. (1998).

Emad, S., Christian, B., & Thomas, Z. (2012). *The Effect of Branching Strategies on Software Quality*.

Environment Variables: What They Are and How To Use Them. (2023).

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology and Design*. Pearson Education. www.serviceoriented.ws.

Ertaul, L., Kaur, M., & Gudise, V. (2016). *Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms*.

Faria, M., Barbosa, A., Guimarães, T., Lopes, J., & Santos, M. (2022a). Predictive analytics for hospital discharge flow determination. *Procedia Computer Science*, *210*(C), 248–253. https://doi.org/10.1016/j.procs.2022.10.145

Faria, M., Barbosa, A., Guimarães, T., Lopes, J., & Santos, M. (2022b). Predictive analytics for hospital discharge flow determination. *Procedia Computer Science*, *210*(C), 248–253. https://doi.org/10.1016/j.procs.2022.10.145

Fernandes, I., Lopes, J., Braga, J., & Santos, M. F. (2022). Applying optimization models in the scheduling of medical exams. *Procedia Computer Science*, *201*(C), 696–701. https://doi.org/10.1016/j.procs.2022.03.093

Ferreira, J., Portela, F., Machado, J., & Santos, M. F. (2019). *Adaptive Business Intelligence in healthcare-A platform for optimising surgeries*.

Ficara, D., Giordano, S., Procissi, G., Vitucci, F., Antichi, G., & Di Pietro, A. (2008). *An Improved DFA for Fast Regular Expression Matching*.

Galic, P. (2020). *The Design and Experimental Use of CReB, a Container Registry Benchmark*.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns Elements of Reusable Object-Oriented Software*.

Gonçalves, C., Ferreira, D., Neto, C., Abelha, A., & Machado, J. (2020). Prediction of mental illness associated with unemployment using data mining. *Procedia Computer Science*, *177*, 556–561. https://doi.org/10.1016/j.procs.2020.10.078

Gonzalez, D. (2016). *Developing microservices with Node.js : learn to develop efficient and scalable microservices for server-side programming in Node.js using this hands-on guide*.

Guolang, L., Jianjun, X., Chaoqun, W., & Jufang, Z. (2018). *A Performance Comparison of HTTP Servers in a 10G-40G Network*.

Gupta, A., & Hohn, A. (2019). *kubernetes*.

Hasan Ibrahim, M., Sayagh, M., & Hassan, A. E. (2021). *A Study of How Docker Compose is Used to Compose Multi-component Systems*. https://github.com/SAILResearch/replication-21-ibrahim-dockercompose

Hawilo, H., Jammal, M., & Shami, A. (2019). Exploring Microservices as the Architecture of Choice for Network Function Virtualization Platforms. *IEEE Network*, *33*(2), 202–210. https://doi.org/10.1109/MNET.2019.1800023

Huskey, S., & Huskey, S. J. (2023). *Committing to Reproducibility and Explainability] {Committing to Reproducibility and Explainability: Version Control as Research Journal Committing to Reproducibility and Explainability: Version Control as Research Journal*. https://github.com/

Jain, S. M. (2020). Linux Containers and Virtualization: A Kernel Perspective. In *Linux Containers and Virtualization: A Kernel Perspective*. Springer International Publishing. https://doi.org/10.1007/978-1-4842-6283-2

Jamil, M. A., Arif, M., Abubakar, N. S. A., & Ahmad, A. (2017). *Software Testing Techniques: A Literature Review*. 177–182. https://doi.org/10.1109/ict4m.2016.045

Jatana, N., Puri, S., Ahuja, M., Kathuria, I., & Gosain, D. (2012). *A Survey and Comparison of Relational and Non-Relational Database*. www.ijert.org

Jiang, S., & McMillan, C. (2017). *Towards Automatic Generation of Short Summaries of Commits*. http://arxiv.org/abs/1703.09603

John, F., Phillip, H.-B., Jeffery, H., Scott, L., Paul, L., Ari, L., & Lawrence, S. (1999). *HTTP Authentication -Basic and Digest Access Authentication*.

Jones, M. (2015). *JSON Web Token (JWT)*. http://www.rfc-editor.org/info/rfc7519.

Jones, M., Campbell, B., & Mortimore, C. (2015). *JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants*.

Juvonen, A., Costin, A., Turtiainen, H., & Hamalainen, T. (2022). On Apache Log4j2 Exploitation in Aeronautical, Maritime, and Aerospace Communication. *IEEE Access*, *10*, 86542–86557. https://doi.org/10.1109/ACCESS.2022.3198947

Kaushik, P., & Majumdar, R. (2019). Impact of SQL Injection in Database Security. *2017 6th International Conference on Reliability, Infocom Technologies and Optimization: Trends and Future Directions, ICRITO 2017*, *2018-January*. https://doi.org/10.1109/ICRITO.2017.8342471

Ke, C., Yang, W., Yuan, Q., & Li, L. (2023). Partial sufficient variable screening with categorical controls. *Computational Statistics and Data Analysis*, *187*. https://doi.org/10.1016/j.csda.2023.107784

Kinsman, T., Wessel, M., Gerosa, M. A., & Treude, C. (2021). *How Do Software Developers Use GitHub Actions to Automate Their Workflows?* http://arxiv.org/abs/2103.12224

Kithulwatta, W. M. C. J. T., Jayasena, K. P. N., Kumara, B. T. G. S., & Rathnayaka, R. M. K. T. (2022). Performance Evaluation of Docker-based Apache and Nginx Web Server. *2022 3rd International Conference for Emerging Technology, INCET 2022*. https://doi.org/10.1109/INCET54531.2022.9824303

Kolahi, S., & Libkin, L. (2006). *On Redundancy vs Dependency Preservation in Normalization: An Information-Theoretic Study of 3NF*.

Laurén, S., Rauti, S., & Leppänen, V. (2017). A survey on application sandboxing techniques. *ACM International Conference Proceeding Series*, *Part F132086*, 141–148. https://doi.org/10.1145/3134302.3134312

Lee, H. (1995). JUSTIFYING DATABASE NORMALIZATION: A COST/BENEFIT MODEL. In *Information Processing & Management* (Vol. 31, Issue I).

Lin, B., Chen, Y., Chen, X., & Yu, Y. (2012). Comparison between JSON and XML in Applications Based on AJAX. *Proceedings - 2012 International Conference on Computer Science and Service System, CSSS 2012*, 1174–1177. https://doi.org/10.1109/CSSS.2012.297

Lin, J., Ranslam, K., Shi, F., Figurski, M., & Liu, Z. (2019). Data migration from operating EMRs to OpenEMR with mirth connect. *Studies in Health Technology and Informatics*, *257*, 288–292. https://doi.org/10.3233/978-1-61499-951-5-288

Liu, Y. A., & Stoller, S. D. (1999). *From recursion to iteration: what are the optimizations?*

Lopes, J., Braga, J., & Santos, M. F. (2021). Adaptive Business Intelligenceplatform and itscontribution as a support in the evolution of Hospital 4.0. *Procedia Computer Science*. www.sciencedirect.comwww.elsevier.com/locate/procedia1877-0509

Lopes, J., Guimarães, T., & Santos, M. F. (2020). Adaptive business intelligence: A new architectural approach. *Procedia Computer Science*, *177*, 540–545. https://doi.org/10.1016/j.procs.2020.10.075

Majeed, A., & Rauf, I. (2018). *MVC Architecture: A Detailed Insight to the Modern Web Applications Development*.

Marc, C. (2019). *Evaluating Functional Programming for Software Quality in REST APIs*.

Martin, R. C. (2014). *Agile software development, principles, patterns, and practices*.

Maruf Hassan, M., Asraf Ali, M., Bhuiyan, T., & Hasan Sharif, M. (2018). *Quantitative Assessment on Broken Access Control Vulnerability in Web Applications*. https://www.researchgate.net/publication/328956656

Matthias, C. (2016). *React and Redux*. www.uni-oldenburg.de/svs

Michalewicz, Z., Schmidt, M., Michalewicz, M., & Chiriac, C. (2007). *Adaptive Business Intelligence*. Springer.

Miranda, M., Salazar, M., Portela, F., Santos, M., Abelha, A., Neves, J., & Machado, J. (2012). Multi-agent Systems for HL7 Interoperability Services. *Procedia Technology*, *5*, 725–733. https://doi.org/10.1016/j.protcy.2012.09.080

Moroz, B. (2019). *UNIT TEST AUTOMATION OF A REACT-REDUX APPLICATION WITH JEST AND ENZYME Title Unit Test Automation of a React-Redux Application with Jest and Enzyme 108 pages 12 pages of appendices Commissioned by*.

Negash, S. (2004). Business Intelligence. *Communications of the Association for Information Systems*, *13*. https://doi.org/10.17705/1CAIS.01315

Nicholas, G. (n.d.). *Cross-Origin Resource Sharing*.

Noumeir, R. (2019). Active Learning of the HL7 Medical Standard. *Journal of Digital Imaging*, *32*(3), 354–361. https://doi.org/10.1007/s10278-018-0134-3

Oemig, F., & Blobel, B. (2011). A formal analysis of HL7 Version 2.x. *Studies in Health Technology and Informatics*, *169*, 704–708. https://doi.org/10.3233/978-1-60750-806-9-704

Ojamaa, A., & Düüna, K. (2013). *Assessing the Security of Node.js Platform*.

Pagotto, T., Fabri, J. A., L´erario, A., & Gonçalves, J. A. (2016). *Scrum Solo Scrum solo Software process for individual development*. http://scrumsolo.wordpress.com.

Passos, M., Duarte, J., Silva, A., Manuel, M., & Quintas, C. (2022). Decision models on therapies for intensive medicine. *Procedia Computer Science*, *210*(C), 230–235. https://doi.org/10.1016/j.procs.2022.10.142

Phillips, S., Sillito, J., & Walker, R. (2011). *Branching and Merging: An Investigation into Current Version Control Practices*.

Potdar, A. M., Narayan, D. G., Kengond, S., & Mulla, M. M. (2020). Performance Evaluation of Docker Container and Virtual Machine. *Procedia Computer Science*, *171*, 1419–1428. https://doi.org/10.1016/j.procs.2020.04.152

Praneeth Reddy, M. (2021). Analysis of Component Libraries for React JS. *IARJSET*, *8*(6), 43–46. https://doi.org/10.17148/iarjset.2021.8607

Raemaekers, S., van Deursen, A., & Visser, J. (2017). Semantic versioning and impact of breaking changes in the Maven repository. *Journal of Systems and Software*, *129*, 140–158. https://doi.org/10.1016/j.jss.2016.04.008

Raiküla, K., & Applications, W. (2023). *Implementation of Automated End-To-End Testing in Web Applications*.

Rawat, P., & Mahajan, A. N. (2020). ReactJS: A Modern Web Development Framework. In *International Journal of Innovative Science and Research Technology* (Vol. 5, Issue 11). www.ijisrt.com

Reshma, S. G., Mohan Kumar, H. P., & Manu, A. G. (2022). Smoke Test Execution in Software Application Testing. *4th International Conference on Emerging Research in Electronics, Computer Science and Technology, ICERECT 2022*. https://doi.org/10.1109/ICERECT56837.2022.10059686

Rinard, M., Cadar, C., Dumitran, D., Roy, D. M., & Leu, T. (2004). *A Dynamic Technique for Eliminating Buffer Overflow Vulnerabilities (and Other Memory Errors)*. www.securityfocus.com

Rocha, A., Associação Ibérica de Sistemas e Tecnologias de Informação, Institute of Electrical and Electronics Engineers. Spain Section, & Institute of Electrical and Electronics Engineers. (2020). *humanportal – A React.js case study*.

Rodriguez, J. C. C., Stäubert, S., & Löbe, M. (2017). Automated import of clinical data from HL7 messages into open clinica and tran SMART using mirth connect. *Studies in Health Technology and Informatics*, *228*, 317–321. https://doi.org/10.3233/978-1-61499-678-1-317

Schwaber, K. (2010). *What Is Scrum?* http://volaroint.com/posts/3.html

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. In *IEEE Access* (Vol. 5, pp. 3909–3943). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2017.2685629

Sharma, P. (2023). *Securing Your Web Application A Deep Dive into OWASP Top 3 Security Risks*.

Sheffer, Y., Hardt, D., & Jones, M. B. (2020a). *JSON Web Token Best Current Practices Abstract*. https://www.rfc-editor.org/info/rfc8725

Sheffer, Y., Hardt, D., & Jones, M. B. (2020b). *RFC 8725 JSON Web Token Best Current Practices Abstract*. https://www.rfc-editor.org/info/rfc8725

Silva, A., Araújo, T., Nunes, J., Perkusich, M., Dilorenzo, E., Almeida, H., & Perkusich, A. (2017). A systematic review on the use of Definition of Done on agile software development projects. *ACM International Conference Proceeding Series*, *Part F128635*, 364–373. https://doi.org/10.1145/3084226.3084262

Singh, G., Kumar, B., Jassi, J. S., Amity University, Institute of Electrical and Electronics Engineers, Institute of Electrical and Electronics Engineers. Uttar Pradesh Section, & Consortium in Association with the University of Northampton (2015 : Noida, I. (2015). *Database Security Using Encryption*.

Soni, R. (2016). Nginx. In *Nginx*. Apress. https://doi.org/10.1007/978-1-4842-1656-9

Sriramya, P., & Karthika, R. A. (2015). *PROVIDING PASSWORD SECURITY BY SALTED PASSWORD HASHING USING BCRYPT ALGORITHM*. *10*(13). www.arpnjournals.com

Staicu, C.-A., Pradel, M., Livshits, B., Darmstadt, T. U., & Livshits, B. (2016a). *Understanding and Automatically Preventing Injection Attacks on Node.js*. https://nodesecurity.io/advisories/

Staicu, C.-A., Pradel, M., Livshits, B., Darmstadt, T. U., & Livshits, B. (2016b). *Understanding and Automatically Preventing Injection Attacks on Node.js*. https://nodesecurity.io/advisories/

Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). *Architectural Patterns for Microservices: a Systematic Mapping Study*. http://microservices.io/patterns/index.html

Thompson, I. (1968). *Regular Expression Search Algorithm The Algorithm* (Vol. 11, Issue 6).

Tian, S., Yang, W., Grange, J. M. Le, Wang, P., Huang, W., & Ye, Z. (2019). Smart healthcare: making medical care more intelligent. *Journal of Global Health*, *3*(3), 62–65. https://doi.org/10.1016/j.glohj.2019.07.001

Torres, A., Galante, R., Pimenta, M. S., & Martins, A. J. B. (2017). Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, *82*, 1–18. https://doi.org/10.1016/j.infsof.2016.09.009

Uddin, M., Islam, S., & Al-Nemrat, A. (2019). A Dynamic Access Control Model Using Authorising Workflow and Task-Role-Based Access Control. *IEEE Access*, *7*, 166676–166689. https://doi.org/10.1109/ACCESS.2019.2947377

Vaishnavi, V., Kuechler, B., & Petter, S. (2004). *DESIGN SCIENCE RESEARCH IN INFORMATION SYSTEMS*.

Vasilakis, N., Staicu, C. A., Ntousakis, G., Kallas, K., Karel, B., Dehon, A., & Pradel, M. (2021). Preventing Dynamic Library Compromise on Node.js via RWX-Based Privilege Reduction. *Proceedings of the ACM Conference on Computer and Communications Security*, 1821–1838. https://doi.org/10.1145/3460120.3484535

Veres, A.-C., & Wilkinson, M. (2023). *An Exploration of Current Techniques in OWASP Vulnerability Detection and Improvement Opportunities An Exploration of Current Techniques in OWASP Vulnerability Detection and Improvement Opportunities Internship Project*.

Vermeulen, A., Beged-Dov, G., & Thompson, P. (1995). *The Pipeline Design Pattern*.

Vorisek, C. N., Lehne, M., Klopfenstein, S. A. I., Mayer, P. J., Bartschke, A., Haese, T., & Thun, S. (2022). Fast Healthcare Interoperability Resources (FHIR) for Interoperability in Health Research: Systematic Review. *JMIR Medical Informatics*, *10*(7). https://doi.org/10.2196/35724

Wang, F., Casalino, L. P., & Khullar, D. (2019). Deep Learning in Medicine - Promise, Progress, and Challenges. In *JAMA Internal Medicine* (Vol. 179, Issue 3, pp. 293–294). American Medical Association. https://doi.org/10.1001/jamainternmed.2018.7117

Wang, W. (2022). Research on Using Docker Container Technology to Realize Rapid Deployment Environment on Virtual Machine. *Proceedings - 2022 8th Annual International Conference on Network and Information Systems for Computers, ICNISC 2022*, 541–544. https://doi.org/10.1109/ICNISC57059.2022.00112

Wen, S. F., & Katt, B. (2023). A quantitative security evaluation and analysis model for web applications based on OWASP application security verification standard. *Computers and Security*, *135*. https://doi.org/10.1016/j.cose.2023.103532

Xiang, C., Salem, M., Das, T., & Xiaoqun, C. (2008). Real Time Software-in-the-Loop Simulation for Control Performance Validation. *Simulation*, *84*(9), 457–471. https://doi.org/10.1177/0037549708097420

Zdun, U., Queval, P. J., Simhandl, G., Scandariato, R., Chakravarty, S., Jelic, M., & Jovanovic, A. (2023). Microservice Security Metrics for Secure Communication, Identity Management, and Observability. *ACM Transactions on Software Engineering and Methodology*, *32*(1). https://doi.org/10.1145/3532183

Zhu, H.-N., Guan, K. Z., Furth, R. M., & Rubio-González, C. (2023). *ACTIONSREMAKER: Reproducing GITHUB ACTIONS*. https://github.com/bugswarm/actions-remaker,

Zolkifli, N. N., Ngah, A., & Deraman, A. (2018). Version Control System: A Review. *Procedia Computer Science*, *135*, 408–415. https://doi.org/10.1016/j.procs.2018.08.191