UNIVERSIDADE
BEIRA INTERIOR

Procura e Extração de Dados Automatizada de Concursos Públicos e Respetivos Documentos

Fernando José Silva Salcedas Cruz

Relatório de Estágio para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientadora: Prof. Doutora Maria Paula Prata de Sousa
Supervisor da Latitudde: Líder Técnico João José Teles Gouveia

junho de 2025

Relatório de Estágio para obtenção do Grau de Mestre em Engenharia Informática elaborado por Fernando José Silva Salcedas Cruz, Licenciado em Engenharia Informática pela Universidade da Beira Interior, sob orientação da Doutora Maria Paula Prata de Sousa, Professora Auxiliar do Departamento de Informática da Universidade da Beira Interior, e Investigadora do Instituto de Telecomunicações no âmbito de trabalho financiado pela FCT/MECI através de fundos nacionais e, quando aplicável, cofinanciado por fundos comunitários no âmbito do UID/50008: Instituto de Telecomunicações.

Declaração de Integridade

Eu, Fernando José Silva Salcedas Cruz, que abaixo assino, estudante com o número de inscrição m13759 do 2º Ciclo de Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 10/06/2025

Assinado por: **Fernando José Silva Salcedas Cruz**
Num. de Identificação: 15059448
Data: 2025.06.10 11:27:08+01'00'

Agradecimentos

Gostaria de começar por agradecer à Professora Doutora Paula Prata por aceitar ser minha orientadora numa fase já avançada do estágio, e por toda a disponibilidade, acompanhamento e contributos prestados, especialmente na redação deste relatório, sem os quais a sua concretização teria sido muito mais difícil.
Agradeço igualmente ao Engenheiro João Gouveia, Líder Técnico na Latitudde, pelo acompanhamento e orientação técnica ao longo dos projetos desenvolvidos durante o estágio curricular. Estendo o meu agradecimento ao Grupo RIT e, em particular, a todas as pessoas com quem tive o privilégio de interagir durante o estágio, com especial destaque para a equipa da Latitudde, em especial ao escritório do Fundão, pelo ambiente, companheirismo e constante disponibilidade para ajudar, que me fizeram sentir integrado na empresa.
Agradeço profundamente à minha família, pelo apoio incondicional que sempre me deu, não apenas nesta etapa final do percurso académico, mas ao longo de toda a minha vida. Sem esse suporte, sei que não teria alcançado tudo o que alcancei até hoje.
Aos meus amigos, e em particular aos mais próximos. A vossa companhia nos bons momentos e, sobretudo, nos dias mais difíceis, foram essenciais para manter-me motivado ao longo deste percurso.
Por fim, um agradecimento muito especial ao Pedro Paixão, colega de universidade e acima de tudo um amigo, que fez também estágio curricular comigo na Latitudde, cujo contributo foi fundamental para o desenvolvimento deste projeto. Este projeto foi o que foi graças a ti.
A todos, o meu sincero obrigado.

Resumo

Este documento descreve o trabalho desenvolvido durante o estágio curricular realizado na empresa Latitudde, com início a 11 de novembro de 2024 e término a 30 de maio de 2025. O estágio consistiu no desenvolvimento de uma solução de automação de processos, com foco na extração de dados de plataformas de concursos públicos e dos respetivos documentos associados.
O objetivo principal do estágio foi a exploração de possíveis soluções e a implementação de uma abordagem para o problema descrito, recorrendo a formações técnicas em *web scraping*, assim como à exploração de estratégias para a extração de dados em documentos.
Para atingir estes objetivos, foi utilizado o Robot Framework para desenvolver um *script* de *web scraping*, uma tecnologia abordada nas formações técnicas que o estagiário realizou no primeiro mês de estágio. Foi também experimentada uma solução com `regex` para extrair informações dos documentos, tendo sido mais tarde utilizada uma abordagem com recurso à IA. Para este projeto, foi ainda desenvolvida uma aplicação *web* com Laravel para a visualização dos dados recolhidos, contudo, esta aplicação não foi desenvolvida pelo estagiário, tendo apenas colaborado na correção de *bugs*. Os vários componentes da solução desenvolvida foram colocados em *containers* para serem utilizados na infraestrutura da empresa. Durante o desenvolvimento deste projeto foi ainda utilizada a metodologia Scrum.
A solução desenvolvida apresentou resultados promissores, com a extração de dados para uma das plataformas realizada com sucesso, e para a outra com resultados menos consistentes. A extração de dados com recurso à IA, mais concretamente através de um assistente da `Assistants API` da OpenAI, apresentou bons resultados, mais positivos do que os obtidos com `regex`.
Em suma, a implementação final permitiu automatizar uma tarefa rotineira, que anteriormente exigia várias horas para ser concluída. Desta forma, é possível melhorar a produtividade da empresa, mantendo a motivação dos seus colaboradores mais elevada. Embora nem todas as etapas tenham sido concluídas a 100%, estas encontram-se praticamente finalizadas, destacando-se apenas as dificuldades com CAPTCHA numa das plataformas e a falta de um domínio para a publicação da página *web*. A automação desta tarefa representa apenas o início, existindo potencial para expandir a solução e automatizar outras tarefas com impacto semelhante na produtividade da empresa.

Palavras-chave

Web scraping, automação de tarefas, Inteligência Artificial, extração de dados em documentos, Robot Framework.

Abstract

This document describes the work carried out during the curricular internship at Latitudde, which started on November 11, 2024, and ended on May 30, 2025. The internship consisted in the development of a process automation solution, focusing on the extraction of data from public procurement platforms and their associated documents.
The main objective of the internship was to explore possible solutions and implement an approach to the identified problem, including technical training in web scraping as well as the exploration of strategies for document data extraction.
To achieve these goals, Robot Framework was used to develop a web scraping script, a technology covered in the technical courses completed by the intern during the first month of the internship. A solution using `regex` was also tested for extracting information from documents, but was later replaced by an approach based on AI. Additionally, a web application in Laravel was developed to visualize the extracted data, however, this application was not implemented by the intern, who contributed only to bug fixing. All components of the developed solution were containerized to run within the company's infrastructure. The Scrum methodology was adopted throughout the project to manage tasks and development cycles.
The developed solution showed promising results, with successful data extraction from one of the platforms, while the other yielded less consistent results. Data extraction using Artificial Intelligence, specifically through an assistant from OpenAI's `Assistants API`, produced better outcomes than those achieved with `regex`.
In summary, the final implementation enabled the automation of a previously time-consuming and repetitive task. As a result, the company can increase productivity while maintaining employee motivation. Although not all stages were completed in full, most were finalized, with only a few remaining limitations, namely, issues with CAPTCHA on one of the platforms and the absence of a domain for publishing the web page. This automation marks only the beginning, with clear potential to extend the solution to other tasks that could similarly enhance the company's operational efficiency.

Keywords

Web scraping, task automation, artificial intelligence, document data extraction, Robot Framework.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API	*Application Programming Interface*
CAPTCHA	*Completely Automated Public Turing test to tell Computers and Humans Apart*
CSS	*Cascading Style Sheets*
CI/CD	*Continuous Integration/Continuous Delivery*
CLI	*Command Line Interface*
CRUD	*Create, Read, Update, and Delete*
DOM	*Document Object Model*
DoS	*Denial of Service*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IA	Inteligência Artificial
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
IT	*Information Technology*
JSON	*JavaScript Object Notation*
LLM	*Large Language Model*
LINQ	*Language Integrated Query*
MVC	*Model-View-Controler*
NLP	*Natural Language Processing*
NLU	*Natural Language Understanding*
ORM	*Object-relational mapping*
PDF	*Portable Document Format*
RAG	*Retrieval Augmented Generation*
RGPD	Regulamento Geral de Proteção de Dados
ROI	*Return Over Investment*
RPA	*Robotic Process Automation*
SDK	*Software Development Kit*
SGBD	Sistema de Gestão de Base de Dados
SQL	*Structured Query Language*
ToS	*Terms of Service*
UBI	Universidade da Beira Interior

UI	*User Interface*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*

Capítulo 1

Introdução

O seguinte relatório foi realizado no contexto da unidade curricular "Dissertação ou Estágio em Engenharia Informática" do 2.º ano, 2.º semestre, do 2.º ciclo de estudos do curso de Engenharia Informática da Universidade da Beira Interior (UBI).

Este capítulo encontra-se dividido em três secções: enquadramento e motivação do estágio (Secção 1.1), objetivos definidos (Secção 1.2) e organização do documento (Secção 1.3).

1.1 Enquadramento e Motivação

O presente relatório apresenta o trabalho desenvolvido durante a duração do estágio. Dá continuação ao trabalho e pesquisa iniciada na primeira fase do estágio, onde já é apresentado a solução desenvolvida, com os testes e análise dos resultados.

A realização de um estágio curricular, proporciona uma experiência prática em ambiente empresarial, muitas vezes distinta das metodologias académicas aplicadas em contexto universitário. Além disso, fornece outro tipo de experiência, permitindo uma complementação e aperfeiçoamento de competências tanto a nível pessoal, académico como profissional, promovendo um primeiro contacto com o mercado de trabalho e contribuindo para a valorização do *curriculum vitae*.

O estágio em questão enquadra-se, assim, na automação de tarefas rotineiras, mais concretamente na recolha de informações relevantes acerca de concursos públicos, publicados diariamente em páginas *web*. Este tipo de tarefas são várias vezes realizadas manualmente, o que pode levar a uma redução de produtividade e motivação, dois fatores frequentemente interligados. Cada vez mais, as empresas têm em consideração estes aspetos, pelo que existe uma tendência crescente por parte das empresas em adotarem soluções de *Robotic Process Automation* (RPA), ou soluções semelhantes, que visam a automação de tarefas repetitivas e manuais, otimizando processos, reduzindo erros humanos e melhorando a produtividade e motivação no geral.

1.2 Objetivos

O estágio tem como principal objetivo o desenvolvimento de uma solução para o problema descrito anteriormente. Para isso, e seguindo o plano de estágio apresentado no próximo

capítulo, foram definidos os seguintes objetivos intermédios:

- Realização de formação técnica, por meio de cursos na plataforma Udemy [3], com vista ao aprofundamento de conhecimentos nas áreas de automação de testes, *web scraping*, com principal foco nas tecnologias Robot Framework e Scrapy –
  - *Robot Framework Test Automation – Level 1 (Selenium)* [2];
  - *Robot Framework Test Automation – Level 2* [4];
  - *Robot Framework with Python – Selenium/Application Programming Interface (API) Automation Testing* [5];
  - *Modern Web Scraping with Python using Scrapy Splash Selenium* [6].
- Estudo técnico das plataformas-alvo, com vista à compreensão da estrutura das páginas e à análise da viabilidade da automação de extração de dados;
- Análise do estado da arte, recolha de bibliografia, artigos e documentação técnica relacionada com a automação de tarefas rotineiras;
- Definição das tecnologias e ferramentas a utilizar para a implementação da solução;
- Modelação da base de dados, e diagramas relevantes para a implementação da solução;
- Implementação, realização de testes, e análise dos resultados da solução implementada.

Embora não estivesse previsto no início do estágio, o estagiário esteve também envolvido na implementação de uma funcionalidade para um cliente da empresa, tornando-se este contributo adicional igualmente num dos objetivos do estágio.

1.3 Organização do Documento

De forma a espelhar todo o trabalho elaborado, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta a motivação para a realização deste estágio, o seu enquadramento, os objetivos, e a respetiva organização do documento;
2. O segundo capítulo – **Empresa e Proposta de Estágio** – apresenta a empresa, a sua história, localização e informações relevantes, seguido por uma descrição do plano de estágio;
3. O terceiro capítulo – **Estado da Arte** – descreve os principais conceitos e tecnologias exploradas no âmbito da solução a desenvolver, bem como uma análise crítica da sua relevância e adequação;

4. O quarto capítulo – **Tecnologias e Ferramentas Utilizadas** – aborda as tecnologias e ferramentas escolhidas para o desenvolvimento e implementação da solução;

5. O quinto capítulo – **Projetos Desenvolvidos** – apresenta os projetos desenvolvidos, diagramas relevantes, modelação da base de dados e arquitetura do sistema, descrevendo com o máximo detalhe possível a implementação das soluções, assim como o contributo do estagiário;

6. O sexto capítulo – **Conclusão** – descreve as conclusões retiradas do estágio, e da solução desenvolvida. Apresenta também algumas sugestões para trabalho futuro relativamente ao tema e da solução implementada;

7. Anexo A – **Projeto Interno** – apresenta todo o material extra, relativo ao projeto principal do estágio, "Procura e Extração de Dados Automatizada de Concursos Públicos e Respetivos Documentos";

8. Anexo B – **Projeto Cliente** – apresenta todo o material extra, relativo ao projeto do cliente.

Capítulo 2

Empresa e Proposta de Estágio

2.1 Introdução

O seguinte capítulo apresenta uma breve descrição da empresa na qual foi realizado o estágio curricular, secção 2.2, e apresenta também o plano de estágio e calendário proposto previamente ao início do estágio, secção 2.3.

2.2 Descrição da Empresa

A Latitudde [7] é uma empresa multinacional com escritórios espalhados um pouco pelo mundo, com operações em Portugal, Peru, Chile, Nova Zelândia e Países Baixos. É uma das oito empresas pertencentes ao Grupo RIT [8], do qual faz parte a empresa mãe do grupo ReadinessIT [9].
Está envolvida em projetos internacionais com diversas empresas como a Altice, Atos, Outsystems, Claro, STC, Kelly, TMovil, entre outros. Emprega mais de 500 pessoas um pouco por todo o mundo com principal foco em Low-code (OutSystems, Appian e Power Apps) e no desenvolvimento de projetos que envolvem React, Angular, Java, .Net, Flutter, Ionic, entre outros.
Dedica-se à prestação de serviços, mas não limitado, de assessoria, consultoria, sistemas de informação, formação, desenvolvimento de *software*, instalação, configuração e manutenção de sistemas de informação e base de dados.
Em Portugal, os escritórios ficam localizados em Lisboa e no Fundão, com o seu principal polo de formação no Fundão. Tem implementada um conjunto de academias, permitindo aos mais jovens adquirir conhecimentos e competências para diversos projetos da empresa.

2.3 Plano de Estágio

O principal objetivo do estágio curricular descrito neste documento é a automação do processo de extração de dados de concursos públicos e dos respetivos documentos. Pretende-se também desenvolver uma aplicação *web* que apresente, de forma estruturada, os dados considerados relevantes para apoiar a tomada de decisões relativamente a concursos públicos, e, eventualmente, à participação nos mesmos. Este processo pretende promover a libertação de recursos humanos de tarefas repetitivas, contribuindo assim para um aumento da produtividade e da motivação.

Para viabilizar o desenvolvimento de uma solução eficaz, o estágio foi estruturado nas seguintes etapas:

- Formação técnica;
- Exploração das plataformas-alvo e seleção das tecnologias a utilizar;
- Desenvolvimento de *scripts* para a realização de *web scraping*;
- Exploração de abordagens para a extração de dados a partir de documentos;
- Desenvolvimento de *scripts* para extração de dados de documentos;
- Desenvolvimento de uma aplicação *web* para visualização dos dados extraídos, facilitando o processo de tomada de decisão;
- *Containerização* e publicação do projeto.

Importa referir que o estagiário participou num projeto interno da empresa, tendo sido desenvolvido maioritariamente por estagiários durante o período do estágio curricular. O estagiário teve participação ativa nas fases exploratórias e técnicas do projeto, incluindo a extração e tratamento de dados, bem como na identificação e correção de *bugs* na aplicação *web*, embora não tenha estado diretamente envolvido no seu desenvolvimento de raiz.

Quanto à duração do estágio e posterior repartição, foram apresentadas várias datas de começo e finalização do estágio curricular, e de forma a cumprir o mínimo de semanas definido pela UBI, ficaram definidas as seguintes datas:

- **Data de começo do estágio curricular:** 11 de novembro de 2024;
- **Data de finalização do estágio curricular:** 30 de maio de 2025;

Foi então estimado um calendário de atividades, apresentado na tabela 2.1, para organizar melhor o desenvolvimento da solução final.

Atividade	**Tempo estimado**
Formação técnica	1 mês
Exploração das plataformas-alvo e desenvolvimento dos *scripts* para *web scraping*	1,5 meses
Exploração de soluções para a extração de dados de documentos	2,5 meses
Desenvolvimento da aplicação *web*	1 mês
Containerização e publicação do projeto	Tempo restante do estágio

Tabela 2.1: Calendário de atividades estimado para o estágio curricular.

Durante o desenvolvimento do projeto, o trabalho foi acompanhado por reuniões diárias, no âmbito da metodologia *Scrum*, a qual se encontra implementada na empresa, sendo utilizada nos seus processos de desenvolvimento.

A fase de publicação do projeto focou-se maioritariamente na sua *containerização*, visando garantir portabilidade, escalabilidade e facilidade de manutenção das várias componentes da solução.

Todo o trabalho e distribuição de tarefas, passou pelo supervisor do estágio, o Líder Técnico João Gouveia, que ao longo do estágio acompanhou o progresso do estagiário pelos projetos em que participou.

Foi ainda realizada uma análise de possíveis riscos e estratégias de mitigação, como forma de garantir o sucesso do plano de estágio e desenvolvimento da solução. Esta análise encontra-se disponível no anexo A.1.

2.4 Conclusões

O presente capítulo apresentou a empresa onde decorreu o estágio, bem como o plano de atividades e a respetiva calendarização, de forma a ilustrar as diferentes etapas previstas e o tempo estimado para a sua concretização.

Capítulo 3

Estado da Arte

3.1 Introdução

O seguinte capítulo retrata o estado da arte, descrevendo a pesquisa realizada ao longo das várias etapas de desenvolvimento da solução, pesquisa essa relativamente aos conceitos e tecnologias consideradas relevantes para a realização do projeto.

O capítulo encontra-se organizado em seis secções. A primeira secção aborda o que é RPA e os seus principais benefícios, secção 3.2. Já na secção 3.3, é apresentado a definição de *web scraping*, como funciona e os seus usos, e também as suas implicações quando usada. A terceira secção deste capítulo, secção 3.4, faz uma breve referência às expressões regulares (`regex` ou `regexp`), mais concretamente para a linguagem de programação Python, como possível solução para o problema definido neste projeto, sendo apresentada ainda outra abordagem, consistindo no uso de uma *Large Language Model* (LLM), na secção 3.5. Foi também investigado *prompt engineering*, *Retrieval Augmented Generation* (RAG) e *fine-tuning* para a obtenção de melhores resultados através do uso de uma LLM, presentes na secção 3.6. O capítulo termina com uma reflexão crítica das tecnologias e conceitos explorados, secção 3.7, estabelecendo assim uma conexão com o capítulo subsequente, tecnologias e ferramentas utilizadas.

3.2 *Robotic Process Automation*

O RPA, trata-se de *"software technology that makes it easy to build, deploy, and manage software robots that emulate humans actions interacting with digital systems and software"* [10]. O RPA combina *"API and User Interface (UI) interactions to integrate and perform repetitive tasks between enterprise and productivity applications"* [11]. Em suma, trata-se de uma ferramenta que permite automatizar tarefas e processos, que atualmente são feitos por humanos.

No contexto empresarial, existem diversas tarefas e/ou processos que podem ser automatizadas, e o RPA *"streamlines workflows, which makes organizations more profitable, flexible, and responsive. It also increases employee satisfaction, engagement, and productivity by removing mundane tasks from their workdays."* [10]. Esta libertação de recursos humanos e também do tempo utilizado para a realização de tarefas rotineiras, permitem que os trabalhadores consigam realizar trabalho que realmente usa as suas capacidades, acelerando também o negócio da organização. E, por outro lado, a implementação destes *robots* tornam

a realização destas tarefas mais rápida e consistente quando comparado ao humano.

A utilização/implementação de um RPA traz diversos benefícios tais como:

- **Melhor resiliência:** RPA *robots* podem ajustar o fluxo de trabalho para corresponder às necessidades da organização;
- **Melhor precisão:** O uso destes *robots* reduzem o erro humano presente em algumas tarefas [12];
- **Melhor produtividade:** A automação de tarefas rotineiras pode aumentar a produtividade [13];
- **Melhor valorização do trabalhador:** RPA pode permitir que os trabalhadores foquem em tarefas e projetos mais importantes [12];
- **Melhor estado emocional dos trabalhadores:** A remoção destas tarefas morosas aos trabalhadores têm impacto direto no estado emocional do trabalhador [12].

RPA é uma tecnologia transformativa, tendo nos últimos anos, alterado como muitas empresas e organizações realizam o seu trabalho. Ter *robots* a fazer tarefas rotineiras ou com pouco valor para o trabalhador em questão, é uma prática cada vez mais comum para várias empresas, alterando o paradigma tradicional do trabalho manual. *Robots* mais avançados conseguem realizar processos cognitivos, como conversas em *chat*, interpretação de texto, compreensão de dados não estruturados. E como já referido nos benefícios, esta implementação impacta positivamente a organização e os trabalhadores.

Atualmente, esta tecnologia procura novas maneiras e mais eficientes de automatizar tarefas repetitivas, já sendo utilizada para vários ramos como o ramo financeiro, legal, de atendimento ao público, operações e *Information Technology* (IT). E com os avanços na Inteligência Artificial (IA), o RPA consegue melhorar a sua automação e implementação para ainda mais tarefas que sem as capacidades atuais da IA não seriam possíveis.

Uma ferramenta RPA deve incluir as seguintes capacidades fundamentais:

- Capacidades de Low-code para a criação de *scripts* de automação;
- Integração com aplicações da organização;
- Orquestração e administração, incluindo configuração, monitorização e segurança.

A tecnologia de automação também pode aceder a informação por meio de sistemas legados, integrando bem com outras aplicações. Isto permite ter um comportamento semelhante a um trabalhador humano a efetuar tarefas. O verdadeiro valor do RPA está na rapidez e simplicidade da integração do *frontend*.

RPA é uma das tecnologias de *software* com maior crescimento no seio das organizações, devido ao seu rápido e notório *Return Over Investment* (ROI), ao baixo investimento inicial, por não causar nenhuma disrupção noutros sistemas e devido ao seu ambiente de criação com Low-Code, o que permite aos trabalhadores construir automações simples sem grande dificuldade.

O RPA é muitas vezes confundido com IA, embora sejam tecnologias completamente diferentes. Enquanto o RPA é *process-driven* a IA é *data-driven*. Onde um RPA *robot* somente pode seguir os processos definidos, ou seja, fazer as tarefas, a IA oferece a componente do ”pensar” e do ”aprender”. Estas duas tecnologias complementam-se bem, e a sua combinação permite a automação de muitas mais tarefas e processos, como já referido anteriormente, mas também melhora o desempenho do *robot* em dada tarefa.

3.3 *Web Scraping*

O termo *web scraping* “refere-se ao processo de extração de conteúdo e dados de *sites* usando *software*” [14]. Trata-se de uma “coleção de práticas usadas para extrair automaticamente [...] dados da *web*” [14]. Em suma, a base do *web scraping* consiste em dois componentes: o *crawler* e um *scraper*. O *crawler* é o componente responsável por navegar pela *internet*, atravessando as páginas *web* alvo, enquanto o *scraper* faz a recolha das informações pretendidas.

Tanto o *crawler* como o *scraper* que operam durante o *web scraping*, variam de página para página. Para as páginas *web* mais simples, o processo é mais simples e fácil de implementar. Por exemplo, páginas que utilizem *HyperText Markup Language* (HTML) estático, são páginas que facilitam o processo de *scraping*. Porém, páginas que utilizem JavaScript para carregar os dados dinamicamente, requerem o uso de outras técnicas, como o uso do Selenium ou fazendo a extração dos dados diretamente da API, quando esta é pública e fornece os dados pretendidos.

A complexidade do *scrap* também varia consoante os dados pretendidos a extrair, visto que para certos casos, a identificação do seletor HTML correto é suficiente (como o XPath ou *Cascading Style Sheets* (CSS)), enquanto para outros casos, é necessário simular ações, como cliques, *scrolling* ou até inserção de informações (formulários ou até realizar o *login*).

A utilização de uma ferramenta de *web scraping* permite automatizar a recolha de dados, especialmente de grandes volumes de dados, de uma forma rápida e eficiente. Atualmente, “uma das principais áreas onde o *web scraping* é utilizado é no *marketing* digital” [15]. Esta recolha de dados tem como principal propósito a monitorização da concorrência e tendências no mercado, mais especificamente informações acerca de produtos, preços, avaliações, e quaisquer outros dados considerados relevantes para o ajuste de estratégias de *marketing*.

Outras áreas onde é bastante usado o *web scraping* são as seguintes:

- Recrutamento;
- Viagens;
- Mercado imobiliário;
- Investigações académicas.

Quando uma página *web* publica os dados, estes geralmente estão disponíveis para o público, estando livres para serem recolhidos, não sendo ilegal a prática de *web scraping*. No entanto, nem todos os dados na *internet* são para recolha. E para travar o *scraping*, várias páginas *web* utilizam mecanismos e ferramentas, tais como:

- *Completely Automated Public Turing test to tell Computers and Humans Apart* (CAPTCHA) – são testes feitos para diferenciar humanos de *bots*, impedindo o *scraping*;
- Limite de *requests* – é feita uma monitorização a um determinado endereço *Internet Protocol* (IP), que verifica quantos pedidos foram feitos num curto espaço de tempo, podendo bloquear temporária ou permanentemente um endereço IP de aceder à página;
- Páginas dinâmicas – algumas páginas *web* carregam os dados dinamicamente recorrendo ao *JavaScript*, o que dificulta aos *scrapers* acederem diretamente aos dados.

Existem técnicas que permitem contornar estas práticas e mecanismos, como o uso de *proxies* e *headless browsers*.

Porém, como já mencionado, nem todos os dados presentes na *internet* estão disponíveis para o público, como, mas não limitado, dados pessoais e de propriedade intelectual. Quando se faz *scrap* de "dados pessoais e de propriedade intelectual, pode tornar-se uma prática maliciosa e violar as leis da privacidade" [16], podendo levar a ações judiciais. Qualquer recolha de dados em que não exista o consentimento em compartilhar, entra também nesta prática maliciosa ou mal-intencionada, porque enquanto "muitos tipos de dados pessoais são protegidos por leis como o Regulamento Geral de Proteção de Dados (RGPD) [...], outros não são" [14].

Para além da legalidade do *web scraping*, a ética também é um fator presente nesta prática. A recolha de "dados de maneira que prejudique a *performance* de um *site* ou extraia informações confidenciais" pode resultar em consequências indesejadas. Uma prática de *scraping* que não seja moderada ou controlada pode levar a que as páginas *web* não consigam lidar com a quantidade de pedidos realizada, levando a que a página fique indisponível, ocorrendo num ataque *Denial of Service* (DoS). É então necessário regular o *scraping*, respeitando também os *Terms of Service* (ToS), o RGPD [17] e o ficheiro `robots.txt`, para não ocorrer em atividades ilegais e práticas não éticas, descritas na página ProxyScrape [18].

3.4 Expressões Regulares

Numa definição mais simples, expressões regulares são *"a pattern describing a certain amount of text"* [19]. Por norma, o termo expressões regulares costuma estar abreviado ou para `regex` ou para `regexp`, sendo que pelo documento, irá ser usada a abreviatura `regex` quando se for a referir a uma expressão regular. O `regex` fornece uma forma bastante concisa e flexível de identificar padrões, conjuntos de caracteres ou palavras que sejam de interesse numa determinada `string`, ou como se pode encontrar na documentação do Python, *"rules for the set of possible `strings` that you want to match"* [20]. Estas `regex` são escritas numa linguagem formal que consegue ser interpretada por um processador de `regex`, que é um programa que examina um dado texto e identifica as partes definidas na `regex`.
`regex` *"are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale"* [21]. Ou seja, esta abstração ao contexto, permite identificar mais conjuntos que respeitam a expressão definida.

Este termo deriva da matemática, que serviu de base para os primeiros algoritmos de busca e de tratamento de texto. Atualmente, são também usadas para procurar e substituir texto, validar formatos de texto e para filtragem de informação.

Diferentes tipos de `regex` utilizam diferentes *engines* para obter correspondências. Isto implica que é necessário utilizar a sintaxe correta para o respetivo *engine*, dado que nem todos os *engines* reconhecem a mesma sintaxe. Para este projeto, foi analisado o `regex` para o Python especificamente, visto tanto o Robot Framework e o Scrapy, tecnologias estudadas durante a formação técnica, serem Python ou baseadas em Python.

As seguintes tabelas foram retiradas da W3Schools [22], e trata da sintaxe utilizada no Python. A tabela 3.1, apresenta metacaracteres, caracteres com um significado especial para o *engine* utilizado pelo `regex` em Python, descrevendo também para que servem.

Caractere	**Descrição**
[]	Conjunto de caracteres
\	Indica uma sequência especial, também sendo possível escapar caracteres especiais
.	Qualquer caractere (exceto `new line`)
^	Começa com
$	Termina com
*	Zero ou mais ocorrências
+	Uma ou mais ocorrências
?	Zero ou uma ocorrência
{}	Exatamente o número especificado de ocorrências
|	Ou
()	Captura e agrupa

Tabela 3.1: Metacaracteres em Python.

As sequências especiais, são sequências que começam com uma barra invertida (\) e seguidas

por um dos caracteres apresentados na tabela 3.2, onde também é possível verificar o seu uso.

Caracteres	Descrição
\A	Devolve um *match* onde os caracteres especificados estão presentes no início de uma `string`
\b	Devolve um *match* onde os caracteres especificados estão presentes no início ou no fim de uma palavra
\B	Devolve um *match* onde os caracteres especificados não estejam presentes, mas não no início ou no fim de uma palavra
\d	Devolve um *match* onde a `string` contenha dígitos (0-9)
\D	Devolve um *match* onde a `string` não contenha dígitos
\s	Devolve um *match* onde a `string` contém um caractere de espaço
\S	Devolve um *match* onde a `string` não contenha nenhum caractere de espaço
\w	Devolve um *match* onde a `string` contenha palavras (caracteres de a-Z, dígitos 0-9, e o _)
\W	Devolve um *match* onde a `string` não contenha nenhuma palavra
\Z	Devolve um *match* se os caracteres especificados estão no fim de uma `string`

Tabela 3.2: Caracteres especiais em Python.

E por fim, a tabela 3.3, apresenta exemplos de conjuntos, os quais são definidos em parênteses retos ([]), e o seu uso.

Conjuntos	Descrição
[arn]	Devolve um *match* onde um dos caracteres especificados exista
[a-n]	Devolve um *match* para qualquer caractere alfabético minúsculo, entre `a` e `n`
[^arn]	Devolve um *match* para todos os caracteres, exceto para os caracteres especificados
[0123]	Devolve um *match* onde exista qualquer dos dígitos especificados
[0-9]	Devolve um *match* onde exista um dígito entre 0 e 9
[0-5][0-9]	Devolve um *match* para quaisquer dois dígitos entre 0 e 59
[a-zA-Z]	Devolve um *match* para qualquer caractere alfabético minúsculo ou maiúsculo entre `a` e `z`
[+]	Em conjuntos, os caracteres +, *, ., |, (), $, não têm nenhum significado especial, ou seja, para este caso, devolve um *match* para qualquer + na `string`

Tabela 3.3: Exemplos de conjuntos em Python.

3.5 *Large Language Model*

LLM é uma categoria de modelos de IA treinados com vastas quantidades de dados, possibilitando a estes modelos "entender e gerar linguagem natural e outros tipos de conteúdo para executar uma grande variedade de tarefas" [23].

Os LLMs são bastantes conhecidos devido ao papel que têm atualmente, porém foram precisos diversos anos até ser possível fazer a sua implementação, onde foi necessário imensos dados para aprimorar o *Natural Language Understanding* (NLU) e o *Natural Language Processing* (NLP), que "ocorreu ao lado de avanços em *machine learning*, modelos de *machine learning*, algoritmos, redes neurais e modelos de transformadores que oferecem a arquitetura para esses sistemas de IA" [23].
Sucintamente, os LLMs "são projetados para entender e gerar texto como um humano, além de outras formas de conteúdo, com base na vasta quantidade de dados utilizados para treiná-los" [23]. Como estes modelos contêm bilhões de parâmetros que permitem capturar pa-

drões complexos na linguagem, conseguem inferir do contexto para gerar respostas coerentes e relevantes. Também conseguem fazer traduções, resumos, responder a perguntas e executar tarefas relacionadas a linguagem.

No contexto da automação de processos, a IA revela-se particularmente eficaz na gestão de dados não estruturados, ao conseguir interpretar o contexto das tarefas para executar as ações requeridas de forma mais precisa e adaptada. Os LLMs, em especial, destacam-se pela sua capacidade de complementar o RPA, permitindo alargar o seu alcance a processos mais complexos e menos estruturados, que anteriormente não podiam ser facilmente automatizados. Enquanto o RPA tradicional mostra-se extremamente eficiente na execução de tarefas repetitivas e bem definidas, especialmente com dados estruturados, a sua eficácia reduz-se significativamente quando confrontada com linguagem natural, documentos longos, ou informação ambígua.
É ainda possível melhorar o desempenho deste tipo de modelos para melhor corresponder à tarefa pretendida. Como estes modelos são feitos para uso geral e embora possam já apresentar bons resultados por si só, a utilização de técnicas como *fine-tuning* e *prompt engineering*, podem melhorar e em muito os resultados obtidos. Consoante a tarefa em questão e também o seu contexto, poderá ser bastante benéfico recorrer a este tipo de técnicas. Para a solução a ser desenvolvida, foi analisado isso mesmo, com a próxima secção a apresentar as técnicas complementares investigadas.

3.6 Técnicas Complementares aos *Large Language Model*

Dentro dos LLMs foram ainda investigados os conceitos e técnicas de *prompt engineering*, RAG e *fine-tuning*, como possíveis complementações para melhorar a solução final. Foi então subdividida esta secção em três subsecções: *Prompt Engineering*, RAG e *Fine-Tuning*.

3.6.1 Prompt Engineering

Prompt engineering é uma técnica emergente que consiste no *design* e formulação estratégica de instruções (*prompts*) para guiar o comportamento de modelos de linguagem, como os LLMs, de modo a gerar respostas mais precisas, relevantes e alinhadas com os objetivos do utilizador [24].
Através da construção cuidada de *prompts*, incluindo instruções claras, contexto adequado, entre outras, é possível controlar não só o conteúdo, mas também o estilo, tom e estrutura das respostas geradas. Esta abordagem é ainda uma forma eficaz de mitigar limitações dos modelos de linguagem, tais como:

- Geração de conteúdo enviesado ou não fidedigno;
- Falta de alinhamento com necessidades específicas do utilizador;

- Respostas descontextualizadas ou incoerentes;
- Reações inesperadas a instruções mal formuladas.

De acordo com Padmaja e Lakshminarayana [24], as principais técnicas de *prompt engineering* incluem:

- Especificação clara do contexto e do *output* esperado;
- Controlo do estilo, tom e sentimento do conteúdo a ser gerado;
- Redução da ambiguidade nas respostas por meio de instruções detalhadas;
- Adaptação dos *prompts* a tarefas específicas.

Além disso, *prompt engineering* pode ser combinado com outras estratégias como *fine-tuning*, RAG e o uso de *prompts* multimodais, abrindo novas possibilidades para personalização e controlo mais fino de modelos de linguagem.

3.6.2 *Retrieval Augmented Generation*

RAG é uma técnica de otimização de *output* de um LLM, isto é, combina os LLMs com bases de conhecimentos externos, sendo estes atualizados e específicos à tarefa em questão, não dependendo só do conhecimento fornecido aos LLMs quando foram treinados. Isto permite ultrapassar limitações nos LLMs ao nível de desatualização de dados, fornecendo ainda um contexto factual e relevante no momento da geração do *output* [25].

O processo de RAG é, por norma, dividido em duas etapas:

- **Recuperação:** é feita uma pesquisa numa base de conhecimentos externa, de forma a recuperar informações relevantes para a tarefa indicada;
- **Geração:** o LLM utiliza as informações recuperadas, utilizando-as como contexto adicional para gerar um *output* mais relevante e atualizado.

Para a utilização do RAG é então preciso uma base de conhecimentos, que precisa de ser mantida e atualizada, o que por vezes pode ser um desafio, tanto na fase da criação quanto na sua manutenção contínua, garantindo que o conteúdo permaneça atualizado e relevante.

3.6.3 Fine-Tuning

O *fine-tuning* é mais uma técnica utilizada para adaptar modelos pré-treinados, como os LLMs, para obter melhores resultados para tarefas mais específicas [26]. Consiste na continuação do treino do modelo, utilizando um conjunto de dados mais específico à tarefa pretendida.

Durante este processo, os pesos do modelo são ajustados com base nos novos dados, permitindo ao modelo ajustar-se à tarefa desejada, aprendendo vocabulário, estilo, estrutura e conhecimentos mais específicos, e que em alguns casos, não existiam no seu treino inicial. Esta técnica permite que modelos que já devolvam resultados satisfatórios, possam devolver resultados mais exatos e mais relevantes para a tarefa em questão.

3.7 Reflexão Crítica das Abordagens Exploradas

Após esta recolha de informação relativa ao estado da arte para o problema descrito, é possível retirar algumas conclusões, mais concretamente na implementação da solução e as tecnologias a utilizar na solução a desenvolver:

- Embora a solução a ser desenvolvida não recorra a uma plataforma RPA tradicional, é inspirada nos seus princípios, recorrendo a diversas tecnologias e ferramentas para automatizar a tarefa rotineira descrita;
- A utilização da `regex` e LLMs são, para o mesmo fim, a extração de dados dos documentos relativos aos concursos públicos. Durante o desenvolvimento serão analisadas ambas as abordagens, começando pela `regex` por ser a mais simples. Porém, como os documentos a analisar são *Portable Document Format*s (PDFs), e não seguindo nenhum *template*, a utilização do `regex` poderá devolver resultados insatisfatórios. A utilização de um LLM poderá ter que ser necessária para extrair as informações pretendidas dos documentos;
- Quanto às diferentes estratégias para melhorar os resultados provenientes de um LLM, caso esta abordagem seja a escolhida, das três investigadas, o *prompt engineering* é a mais simples e menos dispendiosa. Isto faz com que seja extremamente provável a utilização desta estratégia, na eventualidade de ser necessário recorrer a um LLM. O RAG requer alguns recursos, sendo necessário a criação de uma base de conhecimentos, e a sua manutenção. Embora para a solução necessária seja uma estratégia útil, os seus custos associados tornam a mesma pouco apelativa. Por fim, o *fine-tuning* permite ajustar o LLM para poder devolver resultados mais personalizados para a tarefa pretendida. Para um bom *fine-tuning*, será necessário um bom *dataset* com um tamanho razoável, para obter melhorias consideráveis. A sua utilização poderá ser explorada caso seja possível reunir esse *dataset* para o treino.

3.8 Conclusões

Este capítulo abordou o estado da arte para o problema definido, identificando tecnologias e ferramentas relevantes assim como conceitos necessários para uma boa implementação da solução do problema. O capítulo termina com uma reflexão crítica ao que foi investigado estabelecendo uma ligação para o que serão as tecnologias e ferramentas utilizadas para o

desenvolvimento da solução final.

Capítulo 4

Tecnologias e Ferramentas Utilizadas

4.1 Introdução

Ao longo deste documento, utilizou-se a formatação em teletipo para destacar elementos técnicos como bibliotecas, nomes de entidades e atributos da base de dados, e valores de variáveis.

Este capítulo apresenta as tecnologias e ferramentas utilizadas para a implementação deste projeto. Aborda brevemente o que são e como foram utilizadas. Para este fim, o capítulo foi dividido em nove secções.

4.2 Robot Framework

Robot Framework é uma *framework open-source* para automação de testes e RPA, desenvolvida em Python, suportada pela Robot Framework Foundation e usada largamente pela indústria [27]. É *human-friendly* e a sua sintaxe versátil utiliza *keywords* e com suporte extensível a bibliotecas em Python, Java, e outras linguagens. É especialmente eficaz em testes de aplicações *web*, recorrendo à biblioteca `SeleniumLibrary`.

Embora esta *framework* tenha como principal uso a automação de testes, também consegue fazer *web scraping* recorrendo à biblioteca `SeleniumLibrary`, como referido anteriormente. Por esse motivo, o Robot Framework foi utilizado para o desenvolvimento dos *scripts* e *scripts complementares* para a automação do processo de extração de dados das plataformas-alvo, seguindo as componentes aprendidas durante a formação técnica, presente no anexo A.2.

4.3 Python

O Python é uma linguagem de programação de alto nível, interpretada e de propósito geral, conhecida pela sua legibilidade e simplicidade de sintaxe [28]. É frequentemente utilizada em automação de tarefas, análise de dados e IA.

O Python foi essencial na criação de bibliotecas personalizadas e na extensão do Robot Framework, dada a sua compatibilidade nativa e facilidade de integração. Foi também utilizada

para a implementação do *script* para a plataforma B.

4.4 Selenium

O Selenium é um projeto abrangente que inclui um conjunto de ferramentas e bibliotecas para automação de *browsers* [29]. É amplamente utilizado para testes automatizados de *web interfaces*, permitindo simular interações humanas como cliques, preenchimento de formulários e navegação entre páginas.

O Selenium fornece extensões para simular interações de um utilizador com diferentes *browsers*, um servidor de distribuição para escalar a alocação de *browsers* e a infraestrutura necessária para a implementação da especificação *World Wide Web Consortium* (W3C) *WebDriver*. Esta especificação permite escrever código de automação que funciona em todos os principais *browsers* modernos. A flexibilidade do Selenium e a sua compatibilidade com múltiplas linguagens de programação tornam-no uma ferramenta essencial em ambientes de testes automatizados.

Para este projeto foi utilizado para os *scripts* de *web scraping* e também usado na *containerização* da solução.

4.5 Laravel

Laravel é uma *framework* PHP *open-source* para o desenvolvimento de aplicações *web*, que segue o padrão arquitetural *Model-View-Controler* (MVC). Oferece um conjunto robusto de ferramentas e funcionalidades integradas, como sistema de rotas, *Object-relational mapping* (ORM), autenticação e *caching* [30]. Estas funcionalidades permitem acelerar o desenvolvimento de aplicações, promover boas práticas e facilitar a manutenção do código.

O Laravel foi utilizado para o desenvolvimento da aplicação *web* inteira, tanto o *frontend* como o *backend*.

4.6 PostgreSQL

O PostgreSQL é um Sistema de Gestão de Base de Dados (SGBD) relacional *open-source*, conhecido pela sua robustez, escalabilidade e conformidade com os padrões *Structured Query Language* (SQL) [31]. Suporta tipos de dados avançados, integridade referencial e transações complexas.

Foi utilizado como base de dados para armazenar os dados extraídos do *web scraping* e da extração de dados dos documentos.

4.7 Docker

Docker é uma plataforma aberta para desenvolver, empacotar e executar aplicações [32]. O Docker permite separar as aplicações da infraestrutura, facilitando a entrega rápida de *software*. Garante também que a aplicação funciona consistentemente em qualquer sistema.

O Docker foi utilizado para a *containerização* dos vários componentes da solução, criando ambientes isolados de desenvolvimento e produção, facilitando o *Continuous Integration/ Continuous Delivery* (CI/CD) e garantindo portabilidade, escalabilidade e facilidade de manutenção das várias componentes da solução.

4.8 DBeaver

O DBeaver é uma ferramenta universal de gestão de base de dados gratuita e *open-source* para desenvolvedores e para administradores de base de dados [33]. Permite interagir com diferentes SGBD, como o PostgreSQL, MySQL, entre outros.

Foi utilizado para visualizar, consultar e administrar a base de dados PostgreSQL, permitindo realizar operações *Create, Read, Update, and Delete* (CRUD) durante a fase de desenvolvimento.

4.9 Visual Studio Code

O Visual Studio Code, mais conhecido como VS Code, é um editor de código-fonte leve, mas poderoso, com suporte a múltiplas linguagens de programação e uma vasta biblioteca de extensões [34]. Oferece funcionalidades como destaque de sintaxe, terminal integrado, controlo de versão e depuração.

Este foi o *Integrated Development Environment* (IDE) utilizando para o desenvolvimento dos *scripts* para o *web scraping*, e na depuração de *bugs* da aplicação *web* desenvolvida.

4.10 Git

O Git é um sistema de controlo de versões distribuído, gratuito e *open-source*, concebido para lidar com tudo, desde pequenos a grandes projetos, com rapidez e eficiência [35]. É amplamente utilizado no desenvolvimento de *software*. Permite rastrear alterações no código-fonte, colaborar com outros programadores e manter um histórico claro do progresso do projeto.

Foi utilizado para gerir as várias versões dos ficheiros do projeto, integrar com um repositório remoto no GitLab, permitindo a colaboração entre os vários estagiários envolvidos no projeto.

4.11 Conclusões

No presente capítulo foram introduzidas as tecnologias e ferramentas utilizadas no desenvolvimento da solução. Apresenta uma pequena descrição e o seu uso para cada tecnologia e ferramenta.

Capítulo 5

Projetos Desenvolvidos

5.1 Introdução

O presente capítulo tem como intuito apresentar todo o trabalho desenvolvido para os dois projetos em que o estagiário esteve envolvido. O capítulo começa por apresentar as várias etapas de desenvolvimento no projeto interno da empresa, e projeto definido para o estágio curricular, secção 5.2, e termina com a descrição de outro projeto no qual o estagiário esteve envolvido para a implementação de uma funcionalidade, secção 5.3.

5.2 Projeto Interno – Análise de Concursos Públicos

A seguinte secção retrata a solução final desenvolvida. Este projeto é dividido em duas partes. O "*Scraper*" e a aplicação *web*. O "*Scraper*" contém todos os *scripts* e bibliotecas desenvolvidas para a extração dos dados, tanto para as páginas *web*, como para os ficheiros. Já a aplicação *web*, contém tanto o *frontend* como o *backend* da aplicação, usada para apresentar os dados extraídos pelo "*Scraper*". Por esse mesmo motivo, esta secção encontra-se dividida em três subsecções. A subsecção do "*Scraper*", subsecção da Aplicação *Web* e a Publicação do Projeto, subsecções 5.2.1, 5.2.2 e 5.2.3, respetivamente.

5.2.1 "*Scraper*"

Tal como definido no plano de estágio, a primeira fase do desenvolvimento deste projeto passa pela exploração das plataformas-alvo. O objetivo inicial era fazer a extração de dados a três plataformas diferentes, porém, devido à autenticação de uma das plataformas requerer o cartão de cidadão, essa plataforma foi colocada em espera. Sobraram duas plataformas, a plataforma A e a plataforma B. Sobre a exploração das páginas relativo à obtenção dos dados foi possível retirar-se o seguinte:

- **Plataforma A**
  Esta plataforma requer autenticação. Após a autenticação, tem-se acesso a uma lista de concursos públicos, onde, também é possível aplicar filtros para encontrar os concursos pretendidos. Só é possível fazer a filtragem mediante *keywords*. A lista de concursos apresentada na página contém as seguintes informações –
  - Objeto do contrato;
  - Nome da entidade adjudicante;
  - Prazo limite para a submissão de propostas.

Ao selecionar um concurso, pode acontecer um dos seguintes cenários –

- Entrar na página do concurso;
- Aparecer um *modal* do concurso;
- Aparecer uma página a indicar falta de permissões.

Os dois primeiros cenários apresentam dados como a entidade adjudicante e o objeto de contrato, permitindo também fazer o *download* dos ficheiros do concurso. Para finalizar, não parece haver uma API exposta para retirar os dados pretendidos.

- **Plataforma B**
  Esta plataforma também requer autenticação. Porém, ao contrário da anterior, é necessário ultrapassar um CAPTCHA após a autenticação. Com estes requisitos cumpridos é apresentado uma lista de concursos. Esta plataforma oferece melhor filtragem que a anterior, reduzindo assim o número de resultados. A listagem oferece as seguintes informações –

  - Local/zona da entidade adjudicante;
  - Descrição com o nome da entidade adjudicante e objeto de contrato;
  - O tipo de concurso e preço base;
  - Prazo limite para a submissão de propostas.

  Na página do concurso é possível observar mais informações relativamente ao concurso, fazendo *download* dos ficheiros nesse mesmo local. Por fim, os dados podem ser obtidos por meio de uma API, porém, os pedidos dos dados por parte da página só são concluídos após a conclusão do CAPTCHA.

Com esta primeira fase concluída, procedeu-se ao desenvolvimento dos *scripts*. O intuito destes *scripts* é serem executados diariamente, ao final do dia, para obter novos dados e atualizar os dados existentes, para uso posterior.

5.2.1.1 "*Script_A.robot*"

O "*Script_A.robot*" trata-se de um *script* Robot Framework, desenvolvido para fazer o *web scraping* da plataforma A. Com base da exploração realizada, foi feita a automação das várias etapas necessárias para chegar aos dados pretendidos. Para a automação deste processo é necessário automatizar os seguintes passos:

1. Autenticar na plataforma;
2. Fechar um *popup* que aparece sempre quando se entra na página e que aparece ocasionalmente mais tarde;

3. Aplicar filtros de pesquisa para encontrar os concursos pretendidos;

4. Selecionar o concurso;

5. Extrair os dados pretendidos;

6. Fazer *download* do ficheiro comprimido que contém os ficheiros do concurso;

7. Voltar à página anterior e repetir o processo a partir do passo 4 até não haver mais concursos;

8. Repetir o processo a partir do passo 3 até serem aplicados todos os filtros definidos.

Passo 1: primeiramente, é necessário efetuar o *login*. Para isso basta utilizar as credenciais fornecidas e inserir nos respetivos campos. O *script* começa por carregar essas credenciais que estão guardadas num ficheiro *JavaScript Object Notation* (JSON). De seguida, abre o *browser* e navega para a página *login* da plataforma e faz a autenticação.

Passo 2: como é um *popup* que aparece consistentemente quando se entra na página, basta clicar no botão de fechar. O maior problema, é o facto de este *popup* aparecer, aparentemente, ao fim de um tempo aleatório. Este simples *popup* é o suficiente para parar este processo de extração. Para resolver este problema, antes de se tentar entrar na página de um concurso, verifica-se a existência do *popup*, utilizando o XPath para verificar o *Document Object Model* (DOM), fechando o mesmo caso este seja detetado.

Passo 3: no início da execução do *script*, são obtidos os filtros que se encontram guardados num ficheiro JSON. O *script* começa por aplicar os filtros pela ordem definida no ficheiro, executando para cada um os passos descritos de seguida.

Passo 4: com os filtros aplicados, é então apresentada uma lista com até 30 concursos, e no caso de existir mais concursos, é utilizada a paginação. É nesta etapa do processo onde é necessário fazer diversas verificações.
Começa-se por verificar a existência do *popup* como referido no passo anterior. É depois aplicado um filtro, não na página, mas sim na lógica do *script*. Os concursos dos quais é pretendido recolher informações têm de ser concursos onde ainda é possível apresentar uma proposta até concursos onde o prazo limite para a entrega de propostas é de 60 dias, aproximadamente dois meses. Por norma, os concursos aparecem por ordem decrescente relativamente à data definida no prazo limite para a entrega de propostas, porém, foram verificados casos em que alguns concursos não seguem esta ordenação, o que faz com que não seja possível tornar este *script* mais eficiente, onde ao encontrar um concurso que já tenha ultrapassado a data limite para a entrega de propostas, aplicava o próximo filtro.
O fluxograma abaixo, figura 5.1, apresenta o fluxo desde a aplicação dos filtros à seleção do concurso para extração de dados.

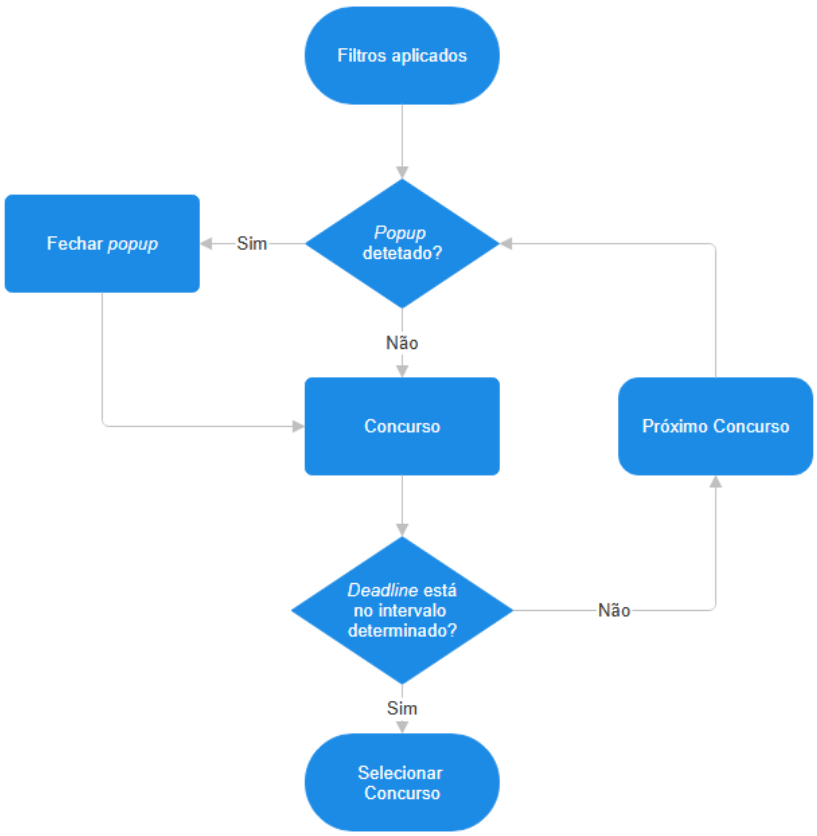

Figura 5.1: Fluxograma da verificação do *popup* e data limite para submissão de propostas.

A segunda verificação é a resposta da página ao selecionar um concurso. Durante a exploração da plataforma A, e mais tarde em testes de uma solução inicial, foram identificados três possíveis cenários ao selecionar o concurso pretendido. Esses cenários são os seguintes:

- Aparece a página do concurso;
- Aparece um *modal* do concurso;
- Aparece uma página a indicar falta de permissões.

Os dois primeiros cenários são os cenários mais frequentes, sendo que o último cenário só ocorre caso o concurso esteja encerrado ou com prazo suspenso. Para o restante dos concursos, ou se entra na página do concurso, ou abre um *modal* do concurso. Isto acontece devido à existência de uma variável que tem como valor o interesse no concurso em questão por parte do utilizador. Sempre que se vai a aceder a um concurso pela primeira vez, o utilizador não vai estar marcado como interessado, sendo necessário marcar este interesse dentro do *popup*. Porém, mesmo neste *modal*, é possível recolher dados relevantes e os documentos de interesse para a proposta. Para o objetivo do *script*, a única diferença entre o concurso estar marcado com interesse ou não estar marcado com interesse, é informações como o prazo para esclarecimentos de dúvidas, sendo este talvez o mais relevante. A verificação de que página é apresentada após selecionar o concurso é de extrema importância, devido à extração

dos dados e posterior *download* dos ficheiros estar diretamente ligado à identificação destes dados.

A seguinte figura 5.2, apresenta o fluxo para a verificação dos cenários descritos no momento da seleção de um concurso.

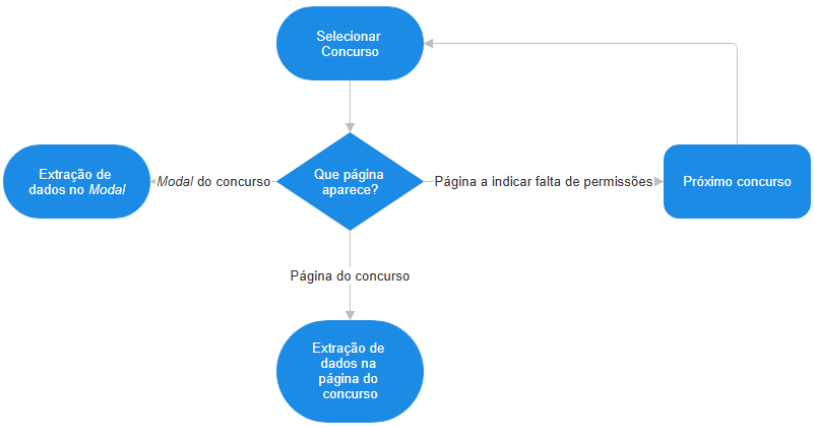

Figura 5.2: Fluxograma da verificação da página carregada após selecionar um concurso.

Passo 5: com a identificação da página apresentada, é feita a extração dos dados, com exceção para o terceiro cenário identificado no passo anterior. Para esses casos, é passado para o próximo concurso da lista. Para a página do concurso, ou seja, para os casos em que o utilizador tem o concurso marcado como interessado, são extraídos os dados e guardados na base de dados.

De notar, que para os casos em que o concurso não está marcado com interesse, no *modal* não é apresentado a data para a entrega das propostas, sendo esse dado retirado da lista dos concursos.

Para garantir que durante a mesma execução, não são extraídos dados das mesmas propostas, antes de se proceder à extração dos dados todos, é extraído o identificador do processo e comparado a uma lista que contem os identificadores das propostas já visitadas. Para o caso desse valor não existir, este é adicionado à lista. Caso já exista, o *script* passa para o próximo concurso.

O seguinte fluxograma, figura 5.3, apresenta o fluxo do *script* no processo para a extração de dados, seja este no *modal* ou na página do concurso.

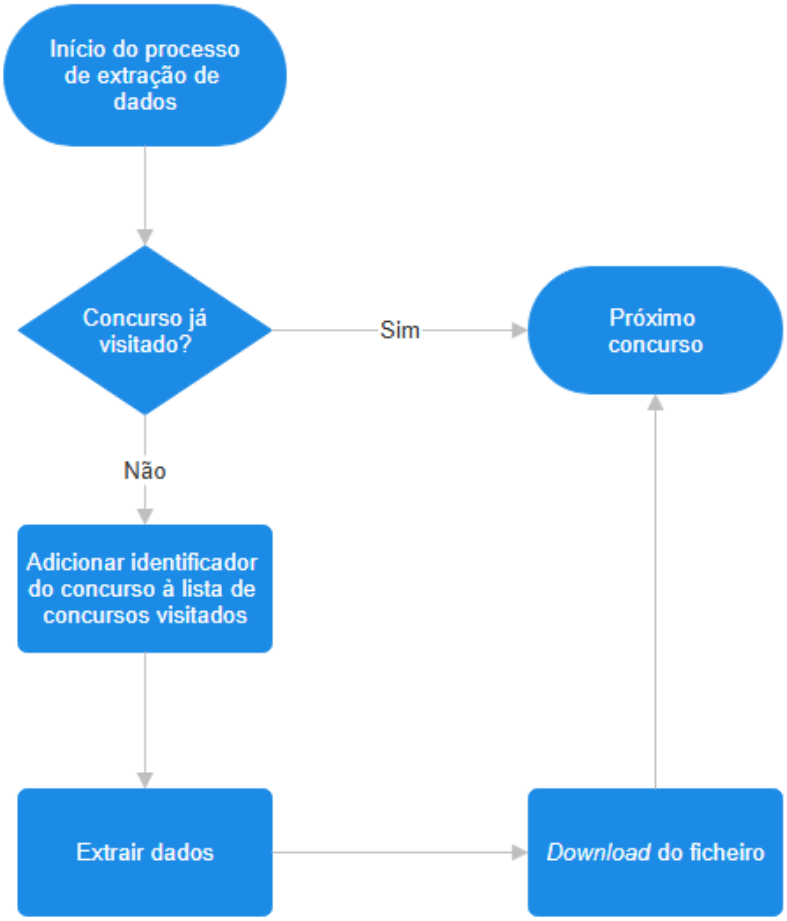

Figura 5.3: Fluxograma do processo de extração de dados do concurso.

Passo 6: na página ou *modal* do concurso, é também feito o *download* do ficheiro comprimido. Neste passo é criado uma diretoria para guardar este ficheiro, sendo a pasta identificada com o `id` da base de dados, que contêm os dados guardados relativamente ao concurso correspondente, e também ao `id` do processo, como, por exemplo, `8_123456`. É também marcado na base de dados, que foi iniciado o *download* com o *status* inicial de `failed`.

Durante a implementação do processo de *download*, foi possível observar que certos concursos demoravam mais tempo que outros para concluir o *download*. Começou-se por impor um limite de tempo para a conclusão do *download* para depois mover-se o ficheiro para a diretoria correta, guardando ainda na base de dados o valor de `success` na coluna `download_status` para os casos em que o *download* concluísse. Caso o *download* não fosse concluído nesse tempo, o ficheiro temporário criado pelo *browser*, era removido, para limpar a diretoria onde os ficheiros são guardados, e na base de dados, o `download_status` era marcado como `failed`. Porém, esta solução provou-se ineficaz, devido a casos em que ao tentar-se remover o ficheiro temporário, este já não existia, ou seja, o *download* tinha sido concluído no momento após ter esgotado o tempo limite. O *script* não fazia verificação de que concurso pertencia o ficheiro comprimido, movendo somente o primeiro ficheiro comprimido existente na diretoria. Após a primeira falha no *download*, tempo limite esgotado, caso o *script* não limpasse o ficheiro temporário, ia começar a mover ficheiros comprimidos

para os concursos errados.
Para resolver o *bug* identificado acima, o *script* deixou de esperar pela conclusão do *download*. Isto não só permite dar mais tempo para o *download* concluir, aumentando a eficácia do *download* dos ficheiros, mas também permite ao *script* continuar a extração de dados dos próximos concursos, melhorando também a sua eficiência. O *script*, no seu processo de *teardown*, limpa a diretoria onde são guardados os ficheiros dos concursos. Começa por distribuir cada ficheiro comprimido para a diretoria do concurso correspondente, atualizando o valor da coluna `download_status` para `success`. No fim deste processo, o *script* limpa a diretoria para remover ficheiros que não foram distribuídos. Com esta implementação, a taxa de sucesso de *download* de ficheiros passou para 100% nas várias execuções efetuadas após esta implementação.
A seguinte figura 5.4, apresenta o fluxo para o *download* dos ficheiros, assim como o processo de *teardown* do *script*.

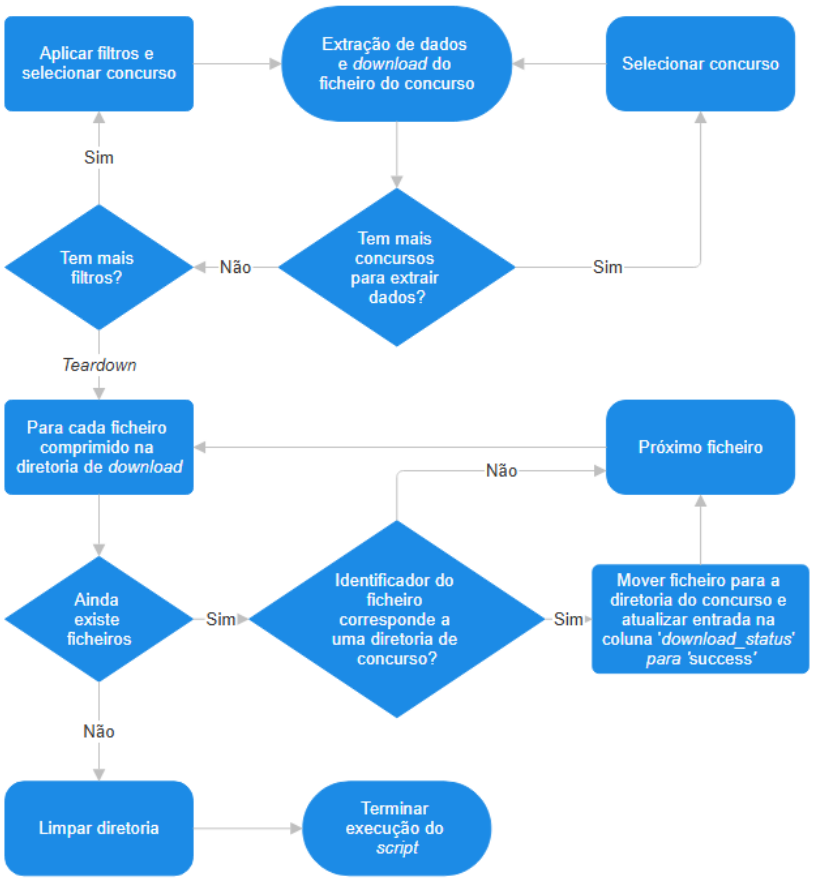

Figura 5.4: Fluxograma do *download* de ficheiros e *teardown* do *script*.

5.2.1.2 "*Script_B.py*"

O "*Script_B.py*" trata-se de um *script* Python, desenvolvido para recolher os dados utilizando a API identificada na fase de exploração. Até à data da escrita do relatório, o *script* ainda não estava finalizado, existindo uma versão do *script* que, infelizmente, não tinha garantia de conseguir fazer *web scraping* à plataforma B. Um aspeto resultante da análise da

plataforma, foi o CAPTCHA presente após a autenticação. Este CAPTCHA impede não só a utilização da plataforma, como ainda impede a utilização da API. O pedido para a API requer no seu *header* uma *cookie*, gerada após a resolução do CAPTCHA. Isto fez com que o processo de *web scraping*, mesmo com recurso à API, estivesse barrado. O *script* conseguia, por vezes, resolver o CAPTCHA, porém, alguns destes CAPTCHAs necessitavam a seleção de imagens, o que impedia por completo qualquer tentativa do *script* resolver o CAPTCHA.

Quanto à extração de dados, através da API, era possível obter as informações pretendidas, e ainda, a partir desses dados obtidos e da data, era possível também iniciar o *download* do ficheiro comprimido para cada concurso.

Devido ao problema já identificado, o desenvolvimento deste *script* esteve por várias vezes interrompido, enquanto eram exploradas outras opções. Por este motivo, o desenvolvimento do *script* não foi terminado, e durante o seu desenvolvimento, houve diversas versões deste *script*, para tentar ultrapassar de forma mais consistente o CAPTCHA.

5.2.1.3 *"Pdf.robot"*

A segunda fase do projeto é a extração de dados dos ficheiros. Esta é a fase mais importante do projeto e também a mais complexa. Para esta fase, foi também utilizado o Robot Framework para escrever o *script*, fazendo uso a bibliotecas customizadas.

Mais uma vez, para automatizar processos, é preciso primeiro identificar os passos necessários para atingir o objetivo final. Os passos definidos para a realização desta tarefa foram os seguintes:

1. Navegar até a diretoria de cada concurso, e fazer a descompressão dos ficheiros;
2. Obter os metadados dos ficheiros PDF, mais concretamente a data de criação e atualização, e calcular o *hash* de cada ficheiro PDF;
3. Verificar se os ficheiros já tinham sido todos analisados;
4. Analisar os ficheiros com recurso à OpenAI.

Passo 1: o script começa por listar todas as diretorias presentes na diretoria "*proposals*", a diretoria onde todos os ficheiros dos concursos foram guardados e organizados durante a fase de *web scraping*. É durante este processo de listagem que o *script* verifica a última vez que cada concurso foi atualizado. Como durante o *web scraping*, os concursos só são acedidos se ainda for possível enviar uma proposta, a partir do momento que o concurso não é atualizado, significa que a data limite para entrega de propostas já encerrou, ou o concurso ficou indisponível. Qualquer um dos cenários não permite enviar propostas. Por esse motivo, o *script* "*pdf.robot*" remove todos os ficheiros e a diretoria do concurso, para limpar espaço. Todas as diretorias listadas são de concursos que ainda se pode submeter uma

proposta. Essas listas são então iteradas, sendo cada concurso acedido e no caso de ainda existirem ficheiros comprimidos, é feita a descompressão recorrendo a uma função recursiva, movendo ainda todos os ficheiros para a *root* da diretoria do concurso.

Passo 2: após a descompressão, é feita uma listagem de todos os ficheiros PDF, pois os ficheiros relevantes estão neste formato. O *script* consulta os metadados dos ficheiros, com o intuito de identificar os ficheiros mais recentes para ordenação dos ficheiros. Este processo foi mais relevante para soluções iniciais, porém, para esta solução desenvolvida não é obrigatório, mas como não altera o processo de forma nenhuma, foi mantido. O cálculo do *hash*, é vital para a eficiência do *script*. Como não existe maneira de averiguar se houve alguma modificação de ficheiros durante o *web scraping*, os dados são sempre atualizados, assim como o *download* dos ficheiros. Isto garante que o "*pdf.robot*" vai sempre ter acesso aos dados mais recentes.

Passo 3: o *hash* é utilizado como comparação aos *hash* calculados para esse concurso, que foram guardados após a análise do concurso na última execução do "*pdf.robot*". É realizada uma *query* à tabela `proposals_files`, onde estão guardados os *hash* dos ficheiros para cada concurso. A *query* devolve o número de ocorrências do *hash*. Se o valor for `0`, significa que não encontrou o *hash* e uma *flag* presente no *script* fica marcada como `True`. Caso contrário, significa que o *hash* já existia e o *script* passa para o próximo ficheiro, repetindo este processo para todos os ficheiros do concurso. Esta *flag* é verificada ao fim de cada *query* e quando esta *flag* é `True` faz com seja iniciada a análise dos ficheiros.

Passo 4: para a análise, são analisados todos os ficheiros, mesmo aqueles que já tinham sido analisados, para garantir que o modelo responde com todas as informações disponíveis.

A solução final para a extração de dados foi implementada com já referido, com recurso à OpenAI. Foram testadas soluções com uso do `regex`, porém estas foram insatisfatórias. A sua implementação e testes podem ser encontradas no anexo A.3.

O mesmo aconteceu com recurso à IA. Foram testadas diversos modelos da Gemini e OpenAI, porém optou-se por utilizar a OpenAI pelo facto de a empresa já possuir uma API *key*. E dentro da OpenAI foram testadas algumas alternativas, porém só se tinha acesso a modelos que recebiam somente *prompts* como *input*, o que limitava bastante a utilização destes modelos para responder às tarefas fornecidas. Mais informações relativamente aos modelos previamente utilizados encontram-se no anexo A.4.

Foi então usado o `Assistants API` da OpenAI. O `Assistants API` permite desenvolver agentes de IA, chamados assistentes, utilizando os modelos da OpenAI. Estes assistentes têm acesso a instruções personalizadas, ferramentas, memória opcional e persistente, arquivos e dados enviados pelo utilizador, *threads* de conversação persistentes.

A figura 5.5, retirada da documentação da OpenAi [1], apresenta as várias componentes que fazem parte do `Assistants API`.

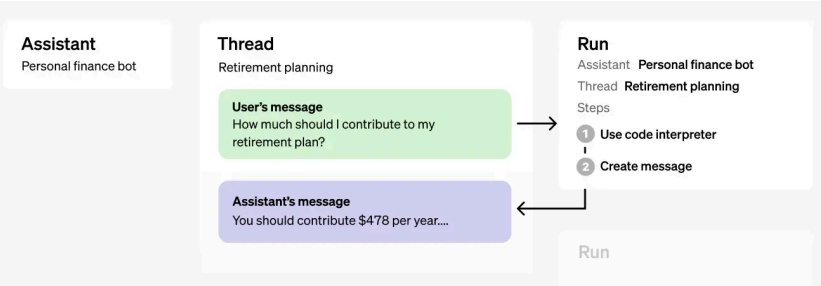

Figura 5.5: Principais componentes do `Assistants API`, disponível na documentação da OpenAi [1].

Estes componentes são os seguintes:

- Assistentes – entidades com instruções, ferramentas e configurações;
- *Threads* – cada conversa com um utilizador é um *thread*, que pode ser guardado e retomado mais tarde;
- Mensagens – mensagens guardados num *thread*;
- *Runs* – processo de execução de um assistente num *thread*;
- Ferramentas – tais como, interpretador de código, recuperação (recuperação de ficheiros), procura de ficheiros, entre outros;
- Memória – guarda contexto de longo prazo para *threads* ou utilizadores.

Foi então criado um assistente com o modelo `gpt-4o-mini`, onde foi instruído a *role* de especialista na análise e extração de informação. Foi ainda adicionada ao assistente a ferramenta para a procura de ficheiros, para uso posterior. Ao assistente foram fornecidos vários *prompts*, um para cada campo pretendido obter resposta. Estes *prompts* correspondiam às diversas tarefas, onde era descrito com detalhes as tarefas a cumprir. O objetivo de fornecer estas tarefas uma a uma é para permitir ao assistente responder com melhor detalhe e não confundir o assistente com múltiplas tarefas. Tal como referido, ao assistente foi adicionado a ferramenta para a procura de ficheiros. Isto foi, porque com o recurso do assistente, era possível fazer o envio dos ficheiros pretendidos para análise. Os ficheiros foram então carregados para uma base de dados vetorial da OpenAI, utilizada pelo assistente na execução das suas tarefas. Isto permitiu fornecer respostas com melhor contexto e sem as limitações dos *tokens*, algo que acontecia nas outras implementações devido a ter que se juntar ao *prompt* o texto do documento. Por fim, as respostas são extraídas onde no fim de responder a todas as tarefas, são guardadas na base de dados.
Os resultados obtidos com esta implementação pode ser visualizados na tabela 5.1. A tabela apresenta o resultado correspondente para os campos que se pretende extrair, realizado em 20 concursos distintos. Não são apresentados os resultados para as especificações funcionais e técnicas e para os requisitos de equipa, por estas terem sido adicionadas posteriormente.

Campos a extrair	Resultados
Preço base	Acertou 15, falhou 5
Preço anormalmente baixo	Acertou 18, falhou 2
Critérios de adjudicação	Acertou 17, deu resposta incompleta para 1, falhou 2
Documentos para a proposta	Acertou 17, deu resposta incompleta para 3, falhou 0
Prazo de execução	Acertou 16, falhou 4

Tabela 5.1: Resultados obtidos por parte do assistente para 20 concursos.

5.2.2 Aplicação *Web*

A aplicação *web* foi desenvolvida em Laravel, uma *framework* para PHP. Foi utilizado o PHP tanto para o *frontend* como para o *backend*. Esta trata-se de uma aplicação *web* simples, com o intuito de mostrar os dados disponíveis na base de dados em tabelas no *frontend* para o utilizador analisar os dados. A aplicação requer autenticação, implementada com recurso ao Microsoft Azure, onde apenas as contas da Latitudde têm acesso à aplicação *web*. Esta aplicação também permite fazer *download* dos ficheiros dos concursos, caso estes ainda permitam submissões, e o envio dos dados apresentados na página por *email*. O principal objetivo desta aplicação *web* foi o da mostragem de dados, e também como aprendizagem para os estagiários que entraram neste projeto para o desenvolvimento desta aplicação *web*.

5.2.3 Publicação do Projeto

Esta parte consistiu, na sua maioria, na *containerização* dos várias componentes do projeto. No total foi necessário a *containerização* de quatro componentes importantes para que o projeto inteiro funcionasse: o "*Scraper*", a aplicação *web*, o "*Scheduler*" e o Redis. A base de dados é uma componente necessária também, porém foi utilizada a base de dados já implementada pela empresa. O "*Scraper*" contém os *scripts* e todos os ficheiros e lógica necessários para a extração dos dados, a aplicação *web* a página *web* a ser hospedada com o Nginx, utilizado por defeito pela Latitudde, o "*Scheduler*" onde foi definido o *job* para as execuções dos *scripts* no "*Scraper*" e o Redis, utilizado como *cache*. Por fim, foi definido um volume, volume esse onde os ficheiros dos concursos foram guardados e para estarem acessíveis à aplicação *web* para o *download* desses ficheiros.

Quanto ao "*Scraper*", foi utilizada uma imagem do Python, mais concretamente a versão `3.12-slim`. Nesse *Dockerfile* foi ainda feita a instalação do Chrome e do seu respetivo *webdriver*. Para a aplicação *web*, foi utilizada uma imagem com o PHP `8.2` e Node.js necessários para correr a aplicação. Para os restantes, uma imagem do Alpine para o "*Scheduler*" e uma imagem do Redis, para a cache.

Com o *docker-compose* definido, e os respetivos *Dockerfile*, a aplicação ficou pronta para ser publicada na máquina da Latitudde. Até à escrita do relatório, a página ainda não ficou disponível, ficando em falta alguns aspetos, tais como o domínio, porém todos os componentes necessários já se encontram na máquina da empresa.

5.2.4 Modelação da Base de Dados

Os modelos de base de dados têm como principal foco a esquematização e organização dos dados de uma aplicação permitindo uma melhor compreensão sobre a mesma.
Esta secção pretende apresentar como os dados utilizados neste projeto ficaram estruturados. Como já foi referido, a base de dados utilizada neste projeto foi o PostgreSQL. Para o projeto foram definidos o modelo entidade relacionamento, na figura 5.6, e o modelo físico presente no anexo A.5.

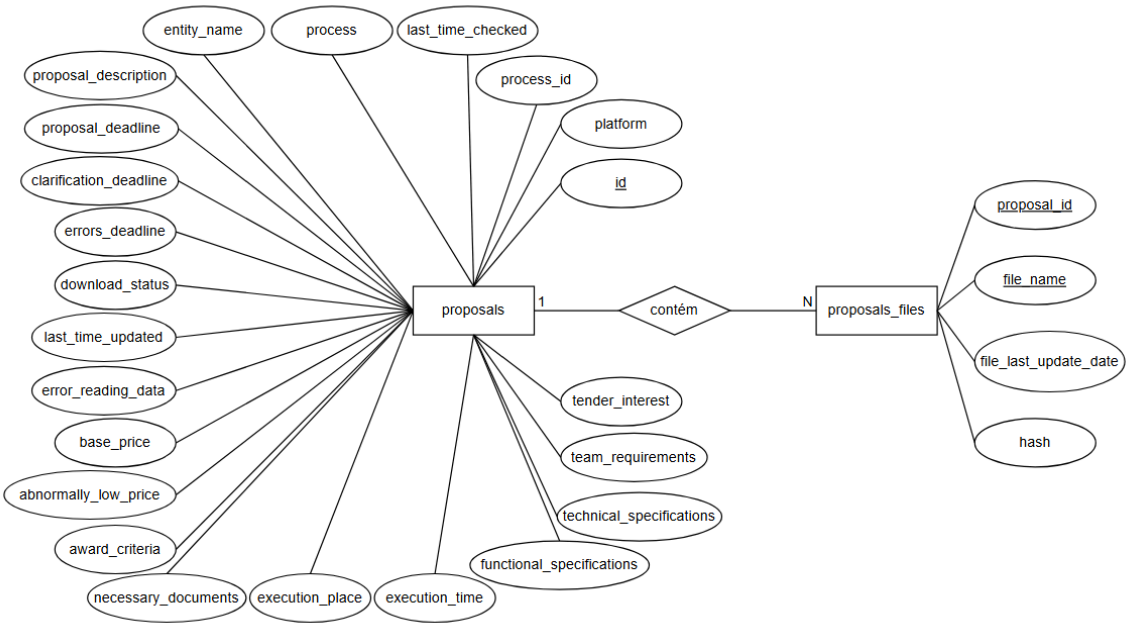

Figura 5.6: Modelo entidade relacionamento da base de dados.

Quanto ao modelo físico, as entidades e os seus atributos são descritos com melhor detalhe no resto desta subsecção.

`proposals` – esta entidade representa todos os dados relacionados com o concurso, necessários para fazer uma proposta. Esta entidade tem os seguintes atributos:

- `id` – identificador da tabela, este é do tipo `serial`, e é a chave primária da tabela;
- `platform` – identifica a que plataforma pertence o registo, tipo `varchar(255)`, e é `not null`;
- `process_id` – identificar da plataforma do processo, tipo `int`, e é `not null`;
- `last_time_checked` – indica a última vez que este registo foi atualizado, tipo `timestamp(0)`, e é `not null` com um valor de defeito `now()`, isto é, o valor é criado no momento do registo automaticamente;
- `process` – mais um identificador da plataforma para concursos onde também está marcado o interesse, tipo `varchar(255)`;

- `entity_name` – nome da entidade, tipo `varchar(255)`, e é `not null`;
- `proposal_description` – objeto do concurso, tipo `varchar`, e é `not null`;
- `proposal_deadline` – data limite para a submissão de propostas, tipo `timestamp(0)`, e é `not null`;
- `clarification_deadline` – data limite para o requisito de esclarecimentos, tipo `timestamp(0)`;
- `erros_deadline` – data limite para a submissão de erros em algum ficheiro, tipo `timestamp(0)`;
- `download_status` – estado do *download*, tipo `varchar(255)`, e é `not null`;
- `last_time_updated` – indica a última vez que os dados extraídos dos ficheiros foram atualizados, tipo `timestamp(0)`;
- `error_reading_data` – indica possíveis erros no processo de extração de dados dos ficheiros, tipo `varchar(255)`;
- `base_price` – preço base do concurso, tipo `varchar`;
- `abnormally_low_price` – preço anormalmente baixo definido no concurso, tipo `varchar`;
- `award_criteria` – critérios de adjudicação, tipo `varchar`;
- `necessary_documents` – documentos que instruem a proposta, tipo `varchar`;
- `execution_place` – local de execução do contrato, tipo `varchar`;
- `execution_time` – prazo de execução do contrato, tipo `varchar`;
- `functional_specifications` – especificações funcionais para o contrato, tipo `varchar`;
- `technical_specifications` – especificações técnicas para o contrato, tipo `varchar`;
- `team_requirements` – requisitos da equipa de desenvolvimento, tipo `varchar`;
- `tender_interest` – opinião da IA acerca do interesse de uma empresa de desenvolvimento de *software* neste concurso, tipo `varchar`;

`proposals_files` – esta entidade representa todos os ficheiros PDF pertencentes ao concurso que foram analisados. Esta entidade tem os seguintes atributos:

- `proposal_id` – identifica a que concurso este ficheiro pertence, tipo `int`, é `not null` e referência a chave primária da tabela `proposals`, sendo chave estrangeira nesta entidade. É uma das duas chaves que juntas fazem uma chave composta;
- `file_name` – nome do ficheiro, tipo `varchar(255)`, é `not null` e é a segunda chave da chave composta desta entidade;
- `file_last_updated_date` – data da última atualização do ficheiro, tipo `timestamp`, e é `not null`;
- `hash` – hash do ficheiro, tipo `varchar(255)`, e é `not null`.

5.2.5 Arquitetura do Sistema

A arquitetura da solução foi desenhada segundo uma abordagem modular e distribuída, com separação clara entre as várias componentes do projeto, como o *scraping*, a aplicação *web* e a base de dados. A Figura 5.7 ilustra a arquitetura geral do sistema, apresentando o conjunto de serviços que compõem a solução. A solução consiste num *docker-compose* com os serviços necessários para o correto funcionamento do projeto, o "*Scraper*", a aplicação *web*, o "*Scheduler*" e o Redis, com a exceção da base de dados, que está num *container* fora do *docker network* da restante solução.

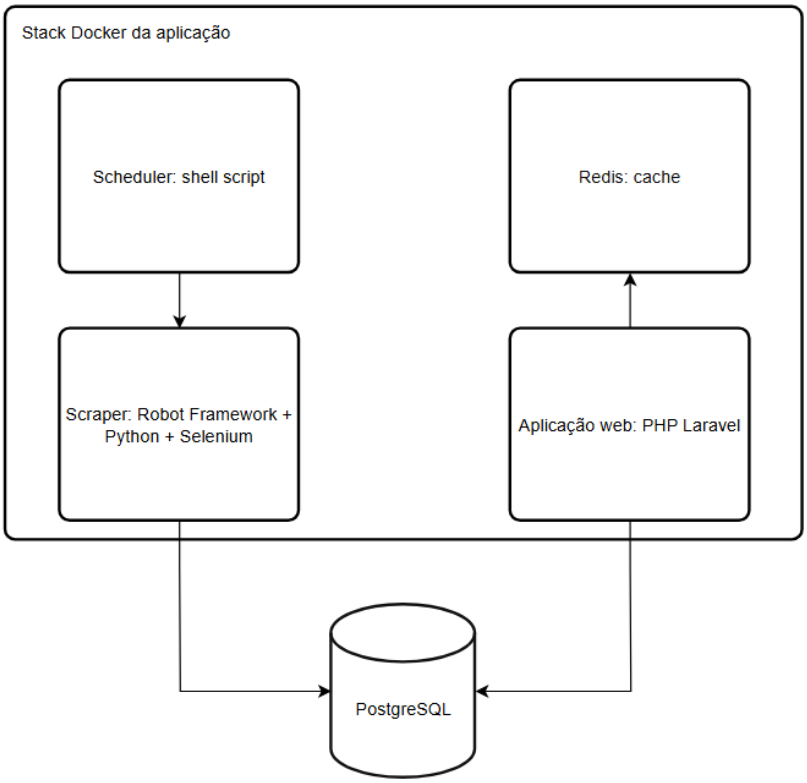

Figura 5.7: Arquitetura do sistema.

5.3 Projeto Cliente – Comparador de Preços

Durante o estágio curricular, e já numa fase com menos tarefas no projeto interno da empresa, o estagiário participou numa reunião onde foi apresentada uma funcionalidade que um dos clientes da empresa queria ver implementada na sua plataforma *e-commerce*. Esta funcionalidade foi atribuída ao estagiário, e que começou o seu desenvolvimento na semana seguinte. Para melhor contextualizar, o estagiário começou o desenvolvimento desta funcionalidade quando se iniciou a implementação da aplicação *web* para o projeto interno, no desenvolvimento da qual não participou.

Esta secção encontra-se repartida em cinco subsecções. A subsecção 5.3.1, descreve a funcionalidade solicitada, já as secções 5.3.2, 5.3.3, 5.3.4 e 5.3.5, abordam sucintamente como foi

implementada essa funcionalidade.

5.3.1 Descrição da Funcionalidade

O cliente solicitou à empresa Latitudde, com a qual já mantinha uma relação contratual, a implementação de uma nova funcionalidade no *backoffice* da aplicação *web*. Esta nova funcionalidade consiste num comparador de preços, cujo objetivo é a realização de *web scraping* sobre outras plataformas de *e-commerce*, e obter dados relevantes para o mesmo produto, permitindo assim ao cliente manter-se competitivo em termos de preços.

A informação recolhida deve ser apresentada numa tabela, permitindo visualizar os produtos da empresa, os respetivos preços, bem como os preços praticados para os mesmos produtos noutras plataformas. Pretende-se também permitir, na mesma página, a seleção dos produtos a comparar, bem como a exportação dos dados da tabela para um ficheiro `csv`.

A execução do processo de *scraping* foi definido para ocorrer com uma periodicidade semanal.

Neste projeto foram utilizadas diversas tecnologias e ferramentas, algumas em comum com o projeto interno, e outras específicas para esta funcionalidade. A lista completa das tecnologias e ferramentas adotadas encontra-se no anexo B.1.

Durante o desenvolvimento desta funcionalidade, foi realizado o levantamento de requisitos, a definição de diagramas considerados relevantes, a modelagem da base de dados e a definição da arquitetura do sistema. Estes elementos encontram-se no anexo B.2.

5.3.2 *"Comparator.robot"*

O objetivo do comparador é o *scraping* a diversas plataformas *e-commerce*, porém, até ao momento da escrita deste relatório, só foi indicado uma plataforma para comparação. A primeira etapa para esta funcionalidade foi a exploração da plataforma-alvo.
Da exploração resultou as seguintes considerações:

- Plataforma requer autenticação;
- É possível fazer a filtragem dos produtos recorrendo a cinco campos, para além do nome do produto;
- A lista dos produtos já fornecem bastante informação sem a necessidade de aceder à página do produto.

Embora seja possível fazer uma filtragem dos produtos, é bastante difícil aplicar filtros para obter o resultado pretendido. Após a exploração dos produtos existentes na base de dados,

foi possível concluir que não ia ser possível aplicar os filtros pelo simples facto de estes não serem totalmente compatíveis, o que iria levar a muitos produtos não serem encontrados.

Optou-se então a procurar os produtos pelo nome existente na base de dados. Esta forma não é extremamente eficaz, pois o produto numa plataforma, não tem que necessariamente ser idêntico a outras plataformas. Para tentar encontrar mais produtos, optou-se por procurar pelo nome do produto, e ao não encontrar nenhuma correspondência, retirar a última palavra, até encontrar uma correspondência ou ficar com apenas uma palavra. Desta forma tenta-se encontrar o produto, ou o produto mais próximo ao desejado, retirando possíveis palavras ou nomenclaturas que possam não ser idênticas de plataforma para plataforma, por exemplo, a forma com as quantidades ou tamanhos, dos produtos, são identificadas ou descritas.

Os passos automatizados para este *script* foram os seguintes:

- Autenticação na plataforma;
- Procura pelo produto correspondente.

O *script* desenvolvido, começa por procurar na base de dados as credenciais para a autenticação, desencriptando essas mesmas para serem usadas na autenticação da plataforma. Com as credenciais carregadas no *script*, o mesmo navega para a página de *login*, autenticando-se na plataforma.

Já na plataforma *e-commerce*, o *script* navega para a página da loja, ignorando um *popup* que aparece ao entrar na plataforma. Na loja, o *script* vai buscar à base de dados todos os produtos para comparação. Com os produtos carregados no *script*, este começa por tentar encontrar correspondências, utilizando a estratégia indicada anteriormente. O *script* atualiza sempre o registo do produto na tabela de comparações, mesmo não tendo encontrado nenhuma correspondência. Desta forma é possível verificar se o produto foi procurado ou não, atualizando também o histórico de operações da tabela de comparação.

5.3.3 *Frontend*

Para o *backoffice*, foi adicionado à *side-bar* já existente, o botão para navegar para a página do comparador.

Nesta página é possível não só visualizar os dados presentes na tabela de comparação existente na base de dados, como ainda é possível adicionar mais produtos para comparação, filtrar os vários produtos já existentes para comparação pelo nome, por ordenação de algumas colunas, tais como o nome, data de adição e data de análise e por plataforma pesquisada. É possível ainda verificar o histórico de cada produto ao selecionar o mesmo e exportar os dados presentes na tabela.

Quanto à tabela onde os produtos são apresentados, aparecem até no máximo 10 produtos, existindo paginação para visualizar os restantes. Os dados apresentados são os seguintes:

- Nome do produto da loja;
- Preço do produto da loja;
- Nome do produto da plataforma pesquisada;
- Preço do produto da plataforma pesquisada;
- Campanhas existentes para o produto da plataforma pesquisada;
- O nome pelo qual encontrou uma correspondência;
- O nome da plataforma pesquisada;
- Data em que o produto foi adicionado;
- Data em que o produto foi analisado pela última vez.

Quando o utilizador exporta os dados da tabela, todos os produtos presentes no comparador são guardados num ficheiro `csv`.

A nível da ordenação, é possível, como já referido, ordenar os produtos pelo nome, data de adição e data de análise. Esta data de análise corresponde à data em que o *script "comparator.robot"* procurou pelo produto. Também é possível procurar por um produto específico na tabela, utilizando uma barra de pesquisa, onde é possível inserir tanto o nome do produto como a referência do produto (esta referência é interna). A página começa a procurar pelo produto, quando forem inseridos pelo menos 3 caracteres. É também possível procurar por produtos comparados para uma ou múltiplas plataformas *e-commerce*, sendo que por defeito apresenta os produtos para todas as plataformas.

Ao selecionar qualquer produto presente na tabela, este abre um *modal*, com informações do produto, mais concretamente o histórico de procuras pelo produto, demonstrando as variações de preço para o produto e plataforma em questão.

É aqui que também é possível modificar alguns dados relativos ao produto para a plataforma selecionada. É possível indicar se é para continuar a procurar pelo produto, e se é para continuar a apresentar o produto na tabela. Ainda permite modificar o nome pelo qual se pesquisa o produto, para tentar encontrar correspondências nos casos em que não devolveu nenhuma correspondência ou devolveu uma correspondência errada. Estas alterações só são confirmadas após a pressionar um botão "Guardar" presente no *modal*.

Por fim, no topo da página, é possível adicionar um ou múltiplos produtos ao comparador. Os produtos adicionados aparecem na tabela presente na página, e serão procurados nas plataformas escolhidas na próxima execução do *script "comparator.robot"*.

Foi implementado no *text input*, *chips*, permitindo ao utilizador facilmente remover e visualizar que produtos já foram adicionados. Para adicionar os produtos, é preciso colocar a referência do produto. Para facilitar o utilizador, o *input* utiliza o mesmo mecanismo que a barra de pesquisa para a tabela, procurando pelos primeiros 300 produtos presentes na base de dados que correspondem aos caracteres já inseridos. Estes resultados aparecem por baixo do *input* e apresentam a referência do produto em conjunto com o nome do produto, permitindo ao utilizador verificar que produto está a selecionar.

Foi também solicitado para implementar um *dropdown* com todos os laboratórios dos produtos presentes na loja, permitindo assim adicionar todos os produtos correspondentes a esse laboratório. Antes de selecionar o botão para adicionar os produtos, existe outro *dropdown* com todas as plataformas fornecidas para a pesquisa dos produtos, permitindo ao utilizador selecionar qual ou quais plataformas pretende fazer uma comparação do produto.

5.3.4 *Backend*

É no *backend* que são recebidos os pedidos provenientes da página *web* para os dados pretendidos. O *backend* expõe *endpoints* aos quais a página envia pedidos do tipo `GET` e `POST`.

Para a implementação de funcionalidade, foi criado um *controller* para o comparador, adicionado um *endpoint* para o *controller* dos produtos, criadas as *entities*, *models* e por fim os *services*, que são chamados pelo *controller* para obter os dados necessários. No âmbito da funcionalidade do comparador, foi necessário implementar:

- Obtenção dos produtos do comparador;
- Obtenção das plataformas;
- Obtenção dos laboratórios;
- Obtenção dos produtos que contenham o texto inserido no *input*;
- Adicionar os produtos ao comparador;
- Modificação dos produtos adicionados ao comparador;
- Exportação dos dados da tabela, considerando os filtros aplicados.

As funcionalidades foram implementadas com recurso ao `Microsoft.EntityFrameworkCore` para a criação das *queries*, utilizando a API de *Language Integrated Query* (LINQ) para gerar instruções SQL de forma segura e eficiente.

Para a obtenção dos produtos, bem como para todas as funcionalidades relacionadas com a apresentação dos produtos na tabela, foram utilizadas variáveis de filtragem. Estas variáveis fornecem ao *backend* informações como o número de resultados por página, ordenação e

filtros aplicados pelo utilizador (como nome do produto ou plataforma).

Para adicionar produtos ao comparador, foi necessário realizar *queries* à tabela que armazena os registos de todos os produtos da loja, bem como à tabela onde estão guardados os preços correspondentes. O comparador guarda as referências dos produtos selecionados para comparação, referências essas que são verificadas antes de adicionar um novo registo, e para atualização de algum registo já existente.

A exportação dos dados segue um processo semelhante ao da obtenção dos produtos para a tabela, aplicando os mesmos filtros e ordenações definidos pelo utilizador. A diferença reside na resposta devolvida, que neste caso é um ficheiro no formato `csv`.

Os *endpoints* mais relevantes, como os utilizados para obter os dados da tabela, recorrem a *cache*, implementada com recurso ao Redis. Assim, quando um pedido idêntico é recebido, a resposta previamente armazenada é reutilizada, evitando a execução de novas *queries* à base de dados. As respostas são mantidas em *cache* durante oito horas, ou até esta ser invalidada manualmente, ou automaticamente (por exemplo, após uma atualização de dados relevantes).

Por fim, foi realizado algum trabalho no *background worker* do projeto para permitir a execução automática do *script*. Para tal, foi necessário criar um *job* no Hangfire, recorrendo a uma expressão `CRON` para definir a sua periodicidade. Esse *job* desencadeia a execução de um serviço no *background worker*, que, por sua vez, envia um pedido *Hypertext Transfer Protocol* (HTTP) para o *container* do comparador.
A próxima subsecção aborda o trabalho final realizado para que o *script* possa ser executado automaticamente.

5.3.5 Docker

O projeto do cliente tem as suas várias componentes a correr em *containers*, que por sua vez estão hospedados numa máquina virtual Linux (Ubuntu 22.04). Durante o processo de *containerização*, foi implementado um simples servidor HTTP, responsável por receber pedidos do tipo `POST`, iniciando a execução do *script* "*comparator.robot*".

Para que o *script* consiga funcionar corretamente, é necessário recorrer a um *browser* e ao respetivo *webdriver*. Por esse motivo, a solução recorre a dois *containers* distintos: um com o *script* e o servidor HTTP, nomeado de "*Comparator*", e outro com a imagem do `selenium/standalone-chrome`, que fornece o ambiente necessário para execução dos testes baseados em *browser*, nomeado de "*Selenium*".

Foi também nesta fase que se montou a *pipeline* de CI/CD, responsável por automatizar os processos de construção, teste e implementação dos *containers*. Esta *pipeline* permite ga-

rantir que quaisquer alterações ao código são validadas e publicadas automaticamente, contribuindo para um fluxo de desenvolvimento mais eficiente e controlado.

Com esta abordagem, garantiu-se que o ambiente de execução da aplicação se mantém isolado, reproduzível e facilmente escalável.

5.4 Testes e Resultados

Esta secção apresenta alguns testes e resultados realizados para validar o trabalho desenvolvido para o projeto interno e do cliente.

5.4.1 Análise de Concursos Públicos

Para validar o correto funcionamento da solução desenvolvida, foram realizados alguns testes funcionais simples à componente do *web scraping* e da aplicação *web*, de forma a testar as principais funcionalidades. Embora a implementação do "*script_B.py*" e a etapa final da publicação *web* não terem sido finalizadas, foram realizados testes ao longo do desenvolvimento. Por esse motivo, estes são apresentados nesta subsecção. A tabela 5.2, apresenta um resumo dos testes realizados, com a sua descrição e resultados esperados.

ID	Descrição do Teste	Resultado Esperado
1	Executar o "*script_A.robot*" manualmente	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
2	Executar o "*script_A.robot*" através do "*Scheduler*"	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
3	Executar o "*script_B.py*" manualmente	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
4	Executar o "*script_B.py*" através do "*Scheduler*"	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
5	Executar o "*pdf.robot*" manualmente	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
6	Executar o "*pdf.robot*" através do "*Scheduler*"	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
7	Fazer autenticação com conta válida	Acesso à página *web* desenvolvida
8	Fazer autenticação com conta inválida	Mensagem de erro
9	Visualizar os dados dos concursos	Tabela com os dados idênticos à base de dados
10	Enviar um *email* com os dados de um concurso	*Email* enviado pela conta do *backoffice* da Latitudde com os dados apresentados na página *web* do respetivo concurso selecionado
11	Fazer *download* dos ficheiros de um concurso	O *browser* inicia o *download* de um ficheiro comprimido correspondente ao concurso selecionado

Tabela 5.2: Resumo dos testes realizados à solução desenvolvida para o projeto interno.

Os testes foram alvo de análise, tendo sido efetuada uma comparação entre os resultados esperados e os obtidos. A tabela 5.3 apresenta os resultados obtidos.

ID	Resultado Obtido
1	*Script* executado com sucesso e com novos registos ou registos atualizados na base de dados
2	*Script* executado com sucesso e com novos registos ou registos atualizados na base de dados
3	*Script* falha em algumas execuções
4	*Script* falha em algumas execuções
5	*Script* executado com sucesso e com novos registos ou registos atualizados na base de dados
6	*Script* executado com sucesso e com novos registos ou registos atualizados na base de dados
7	O utilizador é redirecionado para a página *web*
8	Apresentada uma mensagem de erro, e o utilizador não é redirecionado para a página *web*
9	É apresentado uma tabela com diversos dados correspondentes aos concursos
10	Aparece uma notificação que o *email* foi enviado para os utilizadores escolhidos
11	O *download* é iniciado e concluído com sucesso, e com os ficheiros corretos e não corrompidos

Tabela 5.3: Resumo dos resultados realizados à solução desenvolvida para o projeto interno.

Como é possível observar, dos resultados obtidos, somente os testes relacionados com o "*script_B.py*" é que não tiveram os resultados esperados, devido à falta de consistência na resolução dos CAPTCHAs, como referido anteriormente.

5.4.2 Comparador de Preços

Para validar o correto funcionamento da solução desenvolvida, foram realizados alguns testes funcionais simples à funcionalidade implementada, de forma a testar as principais funcionalidades. A tabela 5.4, apresenta um resumo dos testes realizados, com a sua descrição e resultados esperados.

ID	Descrição do Teste	Resultado Esperado
1	Executar o *script* "*comparator.robot*" manualmente	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
2	Executar o *script* "*comparator.robot*" através do *background worker*	Execução do *script* sem erros e com os dados devidamente guardados na base de dados
3	Adicionar produtos sem plataforma selecionada	Mensagem de erro
4	Adicionar produtos por referência, com referências válidas	Produtos adicionados para comparação com mensagem de sucesso
5	Adicionar produtos por referência, com referências inválidas	Produtos não são adicionados para comparação e apresentação de uma mensagem de erro
6	Adicionar produtos, já existentes, por referência	Produtos não são adicionados para comparação e apresentação de uma mensagem de erro
7	Adicionar produtos, já existentes, mas ocultos, por referência	Produtos são mostrados e pesquisados novamente e apresentação de uma mensagem de sucesso
8	Adicionar produtos por laboratório	Produtos adicionados para comparação com mensagem de sucesso
9	Visualizar os dados na tabela	Produtos e os seus dados são apresentados conforme a base de dados
10	Ordenar os dados da tabela	Produtos e os seus dados são apresentados conforme a base de dados e a ordenação selecionada
11	Filtrar os dados da tabela	Produtos e os seus dados são apresentados conforme a base de dados e com os filtros selecionados
12	Exportar os dados da tabela	O *browser* inicia o *download* de todos os dados existentes na tabela e com os mesmos filtros e ordenação aplicados na tabela no momento da exportação

Tabela 5.4: Resumo dos testes realizados à solução desenvolvida para o projeto do cliente.

Os testes foram alvo de análise, tendo sido efetuada uma comparação entre os resultados esperados e os obtidos. A tabela 5.5 apresenta os resultados obtidos.

ID	Resultado Obtido
1	*Script* executado com sucesso e com novos registos ou registos atualizados na base de dados
2	*Script* executado com sucesso e com novos registos ou registos atualizados na base de dados
3	Produtos não são adicionados na tabela na base de dados e é visualizada uma mensagem de erro
4	Produtos adicionados na tabela na base de dados e é visualizada uma mensagem de sucesso
5	Produtos não são adicionados na tabela na base de dados e é visualizada uma mensagem de erro
6	Produtos não são adicionados na tabela na base de dados e é visualizada uma mensagem de erro
7	Produtos são atualizados na tabela na base de dados e é visualizada uma mensagem de sucesso
8	Produtos adicionados na tabela na base de dados e é visualizada uma mensagem de sucesso
9	Produtos na tabela na base de dados são apresentados na página *web*
10	Produtos na tabela na base de dados são apresentados na página *web* com a ordenação selecionada
11	Produtos na tabela na base de dados são apresentados na página *web* com os filtros inseridos
12	O *download* é iniciado e concluído com sucesso, e com os dados corretos

Tabela 5.5: Resumo dos resultados realizados à solução desenvolvida para o projeto do cliente.

Para a implementação do comparador de preços, todos os testes realizados devolveram os resultados esperados. Isto indica que a funcionalidade foi bem implementada, o que não impede de existir casos que não foram identificados nos testes, resultando em *bugs* no futuro. Embora esta funcionalidade tenha sido completada e com os requisitos pedidos, esta poderá

ser aprimorada no futuro.

5.5 Contributos

O projeto interno foi um projeto realizado em conjunto com mais cinco pessoas, que entraram no projeto em diferentes fases do projeto, onde tiveram os seus contributos antes de prosseguirem para outros projetos. O estagiário esteve envolvido diretamente no desenvolvimento dos seguintes componentes que fizeram parte da solução:

- Análise e exploração das plataformas-alvo;
- Desenvolvimento do "*script_A.robot*";
- Desenvolvimento do "*pdf.robot*";
- Modelação da base de dados;
- *Containerização* dos *scripts* para posterior publicação do projeto.

O estagiário contribuiu ainda nos seguintes:

- Desenvolvimento de versões iniciais do *script* para a plataforma B;
- Desenvolvimento de uma solução utilizando `regex` para a extração de dados de ficheiros dos concursos;
- Testagem dos vários componentes da solução e correção de *bugs*.

Quanto ao projeto do cliente da empresa, o estagiário foi o único desenvolvedor desta funcionalidade, estando envolvido no desenvolvimento do *script* para o *web scraping*, modelação da base de dados, *frontend*, *backend*, *containerização* da solução para o *web scraping*, *background worker*, e criação da *pipeline* para o CI/CD existente neste projeto.

5.6 Conclusões

O presente capítulo descreveu a fase de desenvolvimento dos projetos em qual o estagiário esteve envolvido, entrando em maior detalhe no projeto principal, e definido para o estágio curricular. Já para o projeto do cliente da empresa, foi descrito o trabalho realizado para a implementação da funcionalidade solicitada. O capítulo terminou com a apresentação das contribuições das diversas fases e etapas de ambos os projetos.

Capítulo 6

Conclusão

6.1 Conclusões Gerais

Este documento descreveu todo o trabalho desenvolvido na duração do estágio curricular que iniciou a 11 de novembro de 2024 e encerrou a 30 de maio de 2025.
O projeto principal do estágio, foi um projeto interno da empresa, com o principal objetivo a extração de dados de páginas *web* e de ficheiros relacionados com concursos públicos. Este projeto iniciou-se com quatro formações, concluídas num espaço de um mês, onde foi então feito uma introdução a este projeto. Este projeto ocupou a duração do estágio, com a componente da extração dos dados a ocupar a maioria do tempo. Este projeto envolveu vários estagiários, que contribuíram para diversas e diferentes partes do projeto, porém este projeto iniciou-se e concluiu com os dois mesmos estagiários.
O desenvolvimento e execução do projeto foi no geral positiva, tendo sido atingidos quase todos os objetivos. Porém, obstáculos encontrados atrasaram o desenvolvimento de certas partes da solução. Estes atrasados impediram a conclusão de todas as etapas definidas do projeto, tais como o "*script_B.py*" e a aplicação *web*, onde um ficou em falta o domínio da página. No entanto, tanto o *script* como a publicação do projeto ficaram na reta final da implementação.
O estagiário esteve também envolvido no desenvolvimento de uma funcionalidade para um cliente, um comparador de preços. O desenvolvimento desta funcionalidade foi realizado no intervalo de tempo com menor tarefas para o projeto interno, cumprindo as funcionalidades solicitadas. Até o momento da escrita deste relatório, esta funcionalidade estava no ambiente de testes, a ser testada e à espera da indicação por parte do cliente para subir esta funcionalidade para produção.

6.2 Trabalho Futuro

Com a conclusão do estágio, foram identificados alguns aspetos para trabalho futuro:

- Conclusão do "*script_B.py*" – embora este *script* esteja na reta final da sua implementação, este ainda não está operacional, o que não é o pretendido;
- Integração de mais plataformas – existem mais plataformas onde é possível obter mais concursos públicos que podem ser recorridas no futuro;
- Melhorias na extração de dados dos ficheiros – com acesso a mais recursos, é possível melhorar as respostas obtidas do modelo, como, por exemplo, a utilização de RAG;

- Mais automações de processos – embora o focal deste projeto fosse a extração de dados de concursos públicos, a essência foi a sua automação. É possível automatizar mais processos parecidos e relacionados com este para aumentar a produtividade e reduzir o tempo perdido em tarefas rotineiras;
- Adição de funcionalidades extras no comparador – por exemplo, permitir a adição de novas plataformas diretamente do *backoffice*;
- Melhoria na precisão de *match* de produtos – só a utilização do nome do produto não é suficiente para garantir se o produto existe na plataforma *e-commerce*. A análise e implementação de mais estratégias pode aumentar o número de correspondências por parte do *script "comparator.robot"*.

Bibliografia

[1] OpenAI. (2025) Assistants API overview - OpenAI. [Online]. Available: https://platform.openai.com/docs/assistants/overview xv, 31, 32

[2] Bryan Lamb. (2024) Robot Framework Test Automation - Level 1 (Selenium) | Udemy. [Online]. Available: https://www.udemy.com/course/robot-framework-level-1/ xv, 2, 54, 55

[3] Udemy, Inc. (2025) Online Courses - Learn Anything, On Your Schedule | Udemy. [Online]. Available: https://www.udemy.com 2

[4] Bryan Lamb. (2024) Robot Framework Test Automation - Level 2 | Udemy. [Online]. Available: https://www.udemy.com/course/robot-framework-2/ 2, 54

[5] Rahul Shetty. (2024) Robot Framework with Python-Selenium/API Automation Testing | Udemy. [Online]. Available: https://www.udemy.com/course/robot-framework-with-python-selenium/ 2, 54

[6] Ahmed Rafik. (2021) Modern Web Scraping with Python using Scrapy Splash Selenium | Udemy. [Online]. Available: https://www.udemy.com/course/web-scraping-in-python-using-scrapy-and-splash/ 2

[7] Latitudde. (2025) Home - Latitudde. [Online]. Available: https://latitudde.com/ 5

[8] GroupRIT. (2025) Group RIT | Driving Innovation. [Online]. Available: https://grouprit.com/ 5

[9] ——. (2025) Readiness IT. [Online]. Available: https://readinessit.com/ 5

[10] UiPath. (2025) What is Robotic Process Automation - RPA Software | UiPath. [Online]. Available: https://www.uipath.com/rpa/robotic-process-automation 9

[11] IBM. (2025) What is Robotic Process Automation (RPA)? | IBM. [Online]. Available: https://www.ibm.com/think/topics/rpa 9

[12] Guy Kirkwood. (2019) Impact of RPA on Employee Engagement | UiPath. [Online]. Available: https://www.uipath.com/blog/rpa/impact-of-rpa-on-employee-engagement-forrester 10

[13] Teresa Wu. (2020) New Research Shows Skills Gap A Concern For Workers | UiPath. [Online]. Available: https://www.uipath.com/blog/digital-transformation/new-research-shows-workers-concerned-skills-gaps 10

[14] ? (2023) O que é Web Scraping? Como Extrair Legalmente o Conteúdo da Web. [Online]. Available: https://kinsta.com/pt/base-de-conhecimento/o-que-e-web-scraping/ 11, 12

[15] Diego Dias. (?) Web Scraping: o que é e para que serve. [Online]. Available: https://www.preditiva.ai/blog/web-scraping-o-que-e-e-para-que-serve 11

[16] David González Cuautle. (2024) O que é o scraping malicioso e que medidas podem ser tomadas para minimizar os riscos? [Online]. Available: https://www.welivesecurity.com/pt/seguranca-digital/scraping-malicioso-que-medidas-minimizar-riscos/ 12

[17] IGFEJ. (2024) Regulamento Geral de Proteção de Dados (RGPD). [Online]. Available: https://igfej.justica.gov.pt/Sobre-o-IGFEJ/Regulamento-Geral-de-Protecao-de-Dados-RGPD 12

[18] ? (2022) Ética na recolha de dados na Web. [Online]. Available: https://pt.proxyscrape.com/blog/ethics-in-web-scraping 12

[19] Jan Goyvaerts. (2021) Regular Expression Tutorial - Learn How to Use Regular Expressions. [Online]. Available: https://www.regular-expressions.info/tutorial.html 13

[20] A.M. Kuchling. (2025) Regular Expression HOWTO - Python 3.14.0a4 documentation. [Online]. Available: https://docs.python.org/dev/howto/regex.html 13

[21] The Open Group. (1997) Regular Expressions. [Online]. Available: https://pubs.opengroup.org/onlinepubs/007908799/xbd/re.html 13

[22] W3Schools. (2025) Python RegEx. [Online]. Available: https://www.w3schools.com/python/python_regex.asp 13

[23] IBM. (2023) What Are Large Language Models (LLMs)? | IBM. [Online]. Available: https://www.ibm.com/br-pt/topics/large-language-models 14

[24] Rama Padmaja, Chinimilli Venkata and Lakshminarayana, S, “Enhancing Language Models Through Prompt Engineering - A Survey,” in *2024 IEEE International Conference on Intelligent Systems, Smart and Green Technologies (ICISSGT)*, 2024, pp. 117–121. 15, 16

[25] Amazon Web Services, Inc. (2025) What is RAG? - Retrieval Augmented Generation AI Explained. [Online]. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation/ 16

[26] IBM. (2025) What is Fine-Tuning? | IBM. [Online]. Available: https://www.ibm.com/think/topics/fine-tuning 16

[27] Robot Framework Foundation. (2025) Robot Framework. [Online]. Available: https://robotframework.org/ 19, 63

[28] Python Software Foundation. (2025) Welcome to Python.org. [Online]. Available: https://www.python.org/ 19

[29] Software Freedom Conservancy. (2025) Selenium. [Online]. Available: https://www.selenium.dev/ 20, 65

[30] Laravel. (2025) Laravel - The PHP Framework For Web Artisans. [Online]. Available: https://laravel.com/ 20

[31] The PostgreSQL Global Development Group. (2025) PostgreSQL: The world's most advanced open-source database. [Online]. Available: https://www.postgresql.org/ 20, 64

[32] Docker Inc. (2025) Docker: Accelerated Container Application Development. [Online]. Available: https://www.docker.com/ 21, 65

[33] DBeaver Corporation. (2025) DBeaver Community | Free Universal Database Tool. [Online]. Available: https://dbeaver.io/ 21, 65

[34] Docker Inc. (2025) Visual Studio Code - Code Editing. Redefined. [Online]. Available: https://code.visualstudio.com/ 21, 66

[35] Software Freedom Conservancy. (2025) Git. [Online]. Available: https://git-scm.com/ 22, 66

[36] Google. (2025) Home - Angular. [Online]. Available: https://angular.dev/ 63

[37] Microsoft. (2025) .NET - Crie aplicativos modernos e serviços de nuvem avançados. [Online]. Available: https://dotnet.microsoft.com/pt-br/ 64

[38] Redis. (2025) Redis - The Real-time Data Platform. [Online]. Available: https://redis.io/ 64

[39] Hangfire. (2025) Hangfire - Background jobs and worker for .NET and .NET Core. [Online]. Available: https://www.hangfire.io/ 64

Apêndice A

Projeto Interno

A.1 Riscos e Estratégias de Mitigação

Durante esta fase inicial do projeto, foi possível identificar alguns potenciais riscos que poderão comprometer ou dificultar o desenvolvimento da solução na fase seguinte. Os riscos identificados foram os seguintes, sendo fornecida uma breve descrição do risco, o nível de probabilidade de acontecer, o impacto no projeto e a sua mitigação:

- **Risco 1 - Atrasos no Desenvolvimento:** complexidade técnica, entraves em certas etapas no desenvolvimento ou até fatores externos, podem levar a um atraso no desenvolvimento da solução, onde no pior dos casos, a não conclusão do projeto. A probabilidade deste cenário ocorrer é média.

  Mitigação: ajustes no cronograma das etapas, para ficar em conformidade com o progresso do projeto. Isto tem como consequência a redução de tempo de desenvolvimento para certas etapas do projeto, o que pode levar a uma menor qualidade da solução final, como, por exemplo, a etapa da extração de dados dos documentos, etapa com maior duração prevista para obter os melhores resultados possíveis;

- **Risco 2 – *Web Scraping*:** alterações na estrutura do DOM das plataformas-alvo podem comprometer o correto funcionamento dos *scripts* de extração de dados. Este risco afeta diretamente a primeira etapa do processo automatizado, podendo torná-lo inutilizável. Embora a probabilidade de ocorrência seja considerada baixa, o seu impacto é elevado, dado que interrompe toda a cadeia de automação, não havendo atualização dos dados já obtidos nem a adição de novos dados.

  Mitigação: não é possível prevenir alterações externas ao DOM das plataformas, contudo, sempre que uma alteração seja detetada, será necessário realizar uma nova análise da estrutura da página e ajustar os *scripts* em conformidade;

- **Risco 3 - Custo e Dependência de APIs externas:** se o LLM for usado via serviços como a OpenAI, isso pode implicar custos variáveis com base no volume de chamadas. Há também o risco de mudanças nos ToS, limites de acesso ou mesmo indisponibilidade do serviço, porém a probabilidade de ocorrência é baixa. Este é um possível risco, na eventualidade da solução final recorrer a este tipo de serviços, assim como os riscos seguintes.

  Mitigação: uma avaliação antecipada do custo-beneficio para perceber, numa fase inicial, os custos associados à utilização deste tipo de serviços. Ter como alternativa

modelos *open-source*, como o Ollama, que possam ser usados localmente, mesmo que com menor capacidade;

- **Risco 4 – Imprevisibilidade nas Respostas do LLM:** os LLMs podem, em determinados contextos, gerar respostas incorretas, ambíguas ou até inventadas (fenómeno conhecido como ”alucinação”). Este risco é particularmente elevado quando o conteúdo de entrada é incompleto, mal estruturado ou pouco claro, frequente acontecer com dados provenientes de documentos não estruturados.

  Mitigação: a construção cuidadosa de *prompts* mais explícitos e orientados para a tarefa pode reduzir significativamente a imprevisibilidade dos resultados. Adicionalmente, a configuração de parâmetros como `temperature` e `top_p` deve ser ajustada para favorecer respostas mais conservadoras e factuais, minimizando a criatividade excessiva do modelo;

- **Risco 5 - Complexidade Técnica para Integração:** integrar um LLM com um fluxo automatizado pode ser tecnicamente difícil na gestão de sessões, limites de tamanho no *prompt*, gestão de falhas, entre outros. Como se trata de documentos de tamanhos variados, existe uma probabilidade mediana de, o limite do *prompt* ser ultrapassado.

  Mitigação: existem algumas opções, e consoante o serviço a ser utilizado algumas poderão ser mais viáveis que outras. A opção de fornecer o documento inteiro e depois o *prompt*, ao invés de extrair o conteúdo do ficheiro, concatenando com o *prompt*, elimina o problema do limite de tamanho do *prompt*. Outra solução é a repartição do conteúdo do documento em *chunks*, fazendo ainda uso do *overlapping chunking* para reduzir a perda de contexto entre *chunks*. Por fim, caso o serviço forneça *tiers* com menores restrições, tanto ao nível de chamadas como ao nível do tamanho do *prompt*, poderá ser benéfico a utilização desses *tiers*.

A.2 Formações Técnicas

A seguinte secção apresenta uma descrição breve da componente da formação técnica relativa ao Robot Framework, tecnologia utilizada para o desenvolvimento dos *scripts* de *web scraping*.

A.2.1 Robot Framework

Das quatro formações realizadas, três foram dedicadas ao Robot Framework. As duas primeiras formações [2] [4], são da autoria do Bryan Lamb, sendo que a sua segunda formação utiliza as bases definidas na sua primeira formação. A terceira formação [5], utiliza uma abordagem diferente das duas primeiras, fazendo uso do *learn as you go* como principal

método de aprendizagem.

Robot Framework é uma *framework* de automação de código aberto para automação de testes e RPA, desenvolvida em Python, suportada pela Robot Framework Foundation e usada largamente pela indústria. É *human-friendly* e a sua sintaxe versátil utiliza *keywords* e com suporte extensível a bibliotecas em Python, Java, e outras linguagens.
Integra-se com outras ferramentas para uma automação compreensiva sem a necessidade de pagamento de licenças, com uma grande comunidade com centenas bibliotecas de terceiros.

Robot Framework não utiliza código, sendo *keyword-driven* e contendo também capacidades para *data-driven*. De certa forma, Robot framework é Selenium (contém uma camada de abstração em cima do Selenium), mas permite testar mais do que aplicações *web*.
Como não utiliza código e utiliza linguagem como o inglês, é bastante fácil de compreender. Utiliza *keywords*, como já referido anteriormente, mas ainda permite a criação de *custom keywords*, permitindo uma leitura de resultados que faça sentido para o contexto dos testes realizados.

Os *scripts* de teste estão dependentes de bibliotecas que estão *built-in* com o Robot Framework mas também com bibliotecas externas, sendo possível instalar as dependências necessárias com o comando `pip`.

As seguintes figuras, A.1 e A.2, ilustra como as várias componentes de um projeto Robot Framework, que utiliza o `SeleniumLibrary`, comunicam entre si, sendo que a primeira mostra um projeto com apenas um *script*, e o segundo com uma estrutura mais complexa.

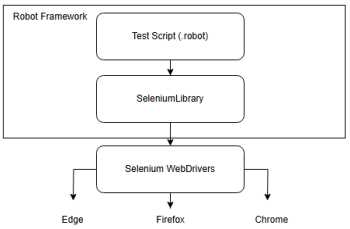

Figura A.1: Exemplo de um projeto Robot Framework com apenas um *script*, retirado da primeira formação [2].

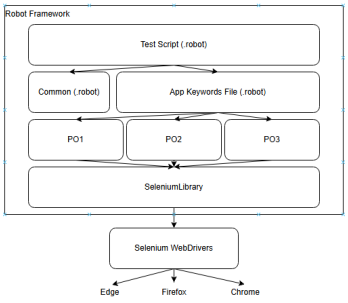

Figura A.2: Exemplo de um projeto Robot Framework complexo, retirado da primeira formação [2].

Ambas são semelhantes, com a principal diferença a forma como o projeto da figura A.2 está estruturada. Esta arquitetura é mais usual para projetos mais complexos, permitindo fazer uma melhor separação das várias componentes do teste, e para permitir uma melhor manutenção do código. Em qualquer um dos cenários, os *scripts* comunicam com o Selenium, que por sua vez, com recurso aos *webdrivers* executam as ações definidas nos respetivos *browsers*.

A.2.1.1 Estrutura de um Projeto e Ficheiros em Robot Framework

Durante as duas primeiras formações, foi sugerida e seguida a seguinte estrutura base de ficheiros: três diretorias, "*Tests*", "*Results*" e "*Resources*".
Dentro da diretoria "*Tests*" serão guardados os *scripts* de testes, que consiste nos ficheiros onde estão os vários casos de teste.
Na diretoria "*Results*", vão ficar guardados os resultados obtidos no final da execução dos testes para consulta posterior. Estes ficheiros são gerados automaticamente pelo Robot Framework, tirando capturas de ecrã em caso de falha nos testes executados. É de notar que, cada execução, subscreve qualquer resultado guardado anteriormente, exceto se indicado a não o fazer (uso de *flags* no comando de execução).
Já para a última diretoria num projeto base, "*Resources*", são guardadas todas as *keywords high level* criadas, que representam as várias capacidades que o sistema sobre teste vai conseguir executar. É boa prática, nesta diretoria, conter dois ficheiros. Um "*common.robot*", que irá conter as *keywords* que são comuns para vários *scripts* de teste, e que no cenário do uso do Selenium, é a abertura de um *browser* e também o seu encerramento. E o outro ficheiro vai conter o resto, denominado por "*AppKeyword.robot*", sendo que este ficheiro deve existir para cada *script* presente na diretoria *Tests*. Dentro desta diretoria, também é uma boa prática a criação de outra diretoria intitulada de "*PO*", que são as "*Page Objects*". Deve ser criada um ficheiro "*page object.robot*" para cada página que seja visitada durante a execução dos casos de teste. Nesses ficheiros estarão contidos os localizadores *web* e as *keywords* específicas à página correspondente.

A esta estrutura é possível adicionar mais diretorias caso seja necessário. Uma diretoria para centralizar os dados, outra para guardar bibliotecas customizadas, entre outras.
Esta estrutura em camadas, permite escrever não só testes com resultados bastante intuitivos, mas permite também uma melhor manutenção dos testes.

Por fim, é importante notar que o selenium precisa de *webdrivers* para comunicar com o *browser* correspondente, pelo que é necessário fazer o *download* do *webdriver* correspondente e para a versão correspondente, sendo necessário atualizar sempre que necessário para evitar possíveis erros. Embora estes ficheiros possam estar dentro do projeto Robot Framework, como se tratam de executáveis que se usam sempre em projetos Selenium, estes devem ser guardados fora e o seu caminho deve ser indicado nas variáveis de ambiente.

A.2.1.2 Secções de um Ficheiro Script

Um ficheiro *script* está dividido em 4 secções. "*Settings*", "*Variables*", "*Test Cases*" e "*Keywords*", por esta ordem. A secção "*Settings*" contém as chamadas às bibliotecas utilizadas, assim como à utilização de outros recursos presentes no projeto. A secção "*Variables*", é onde são definidas as variáveis para serem utilizadas durante os testes. A secção "*Test Cases*" é onde os vários casos de teste estão escritos. Por fim, a secção "*Keywords*" é onde são definidas *keywords* customizadas para a execução dos testes.
Estas secções não têm que estar sempre presentes, apenas aquelas que forem utilizadas, ou seja, no caso de um ficheiro que é utilizado para guardar variáveis, basta apenas ter a secção "*Variables*".

A.2.1.3 Variáveis

O uso de variáveis, permite centralizar os dados num certo ficheiro, mais relevante, para uso do *script* na execução dos testes. Também permite evitar a repetição de dados, quando estes são usados várias vezes no projeto, sendo que na eventualidade de estas requererem alteração, é apenas necessário fazer essa alteração uma vez.
O Robot Framework tem três tipos de variáveis: `scalar variables`, `list variables` e por fim, `dictionaries`.

As `scalar variables` guardam dados singulares, sendo que são definidas `${MY_VARIABLE} = (double space) value`, se esta for definida na secção "*Variables*" ou `${my_variable} = Set Variable(double space) value`, se for definido dentro da secção "*Test Cases*". A diferença entre as duas é o *scope* da variável. Enquanto que a primeira é global, a segunda é apenas acessível dentro do bloco onde foi definida, ou seja, local. Esta dinâmica funciona da mesma forma para as restantes variáveis, havendo ligeiras diferenças de como são definidas.

`List variables` guardam vários valores, como o nome indica, e é definida na seguinte forma, `{MY_List} = (double space) value (double space) value (double space) value ...`. Para a sua definição local, é usada a *keyword* `Create List` após o sinal de igualdade.

E por fim, as `dictionaries` são listas, mas a cada valor tem uma chave associada. São extremamente úteis para guardar valores associados a uma estrutura, isto é, dados de um utilizador, diversos *Uniform Resource Locator*s (URLs), entre outros. São definidas da seguinte forma: `&{MY_LIST} = (double space) key=valor (double space) key=valor (double space) key=valor ...`, e localmente com a *keyword* `Create Dictionary`.

Embora só existam três tipos de variáveis, é importante ter conhecimento das `automatic variables`. Estas são variáveis definidas pelo próprio Robot Framework e que contêm informações úteis quanto ao ambiente de execução, estado do teste ou caminho de arquivos. Estas variáveis são particularmente úteis para aceder a informações contextuais sobre o teste em

execução ou para configurar dinamicamente partes de um teste sem a necessidade de definir variáveis.

Para usar as variáveis e recuperar os valores basta escrever o nome da variável, precedida por um cifrão (`$`).
O sinal de igualdade na definição das variáveis não é necessário, bastando substituir por um `double space`, embora o seu uso permite entender mais fácilmente que se trata da definição de uma variável.
É boa prática a definição de variáveis globais em maiúsculo enquanto as outras são definidas em minúsculo.
E por fim, é possível em Robot Framework passar variáveis pela linha de comandos aquando da execução, sendo importante ter em conta a precedência das variáveis. A precedência é a seguinte: *Command Line Interface (CLI) > Script > Keyword*. Isto permite que sejam definidos valores estáticos no *script*, mas que possam ser alterados durante a execução para diferentes valores.

A.2.1.4 Análise Crítica das Formações

Após a conclusão dos cursos, que tiveram a duração de 1 mês, foi feita uma reunião para demonstrar os conhecimentos adquiridos. Após esta reunião, foi realizada outra para apresentar o projeto a desenvolver. A escolha da tecnologia a utilizar para a realização do *web scraping* foi deixada aos estagiários, onde em discussão, foi escolhida o Robot Framework.

A escolha do Robot Framework ao invés do Scrapy deveu-se aos seguintes:

- A fácil implementação para a automação de tarefas, dado o facto, de que o Robot Framework é utilizado maioritariamente para automação de testes;
- Embora seja uma tecnologia para automação de testes, a sua arquitetura permite facilmente, com a utilização de bibliotecas, testar páginas *web*, com a utilização do Selenium, removendo logo uma barreira do Scrapy, o JavaScript;
- Outro fator bastante importante foi a prática. Durante as formações, foi passado um número de horas significativamente superior a praticar com o Robot Framework, do que com o Scrapy, pelo que os estagiários estavam mais confortáveis com a utilização do Robot Framework.

As restantes tecnologias utilizadas, foram tecnologias solicitadas pela empresa.

A.3 Implementação com `Regex`

Para a extração de dados, recorreu-se à `regex`. O `regex` trata-se de uma solução simples para encontrar os dados pretendidos. Os dados relevantes definidos para extrair dos documentos

são os seguintes:

- Preço base e preço anormalmente baixo;
- Critérios de adjudicação;
- Documentos que instruem a proposta;
- Prazo/local de execução;
- Especificações funcionais;
- Especificações técnicas;
- Requisitos da equipa.

Inicialmente, era apenas necessário fazer a extração dos quatro primeiros, mas com o avanço do projeto foi requerido a extração dos últimos três.

Para fazer a definição das expressões, era necessário primeiro analisar os documentos para identificar os dados pretendidos, e tentar fazer a recolha desses mesmos dados. Dos vários documentos que faziam parte das propostas, os mais importantes, e aqueles que continham, por norma, os dados que se procuravam, era o ”Caderno de Encargos” e o ”Programa de Concurso”. Outro documento também relevante era o ”Anúncio de Procedimento”. Durante esta exploração foram identificados alguns problemas, tais como:

- O nome dos documentos não erão consistentes – como referido, os documentos mais relevantes eram o ”Caderno de Encargos” e o ”Programa de Concurso”. Porém, o nome dos ficheiros nem sempre era o mesmo. O ”Programa de Concurso às vezes era ”Programa de Procedimento”, ”PC”, ”PP”, ou até um nome nada relacionado com o documento. O mesmo acontecia para todos os documentos. Esta falta de consistência torna extremamente difícil ou impossível identificar o documento sem proceder à sua leitura;
- Os documentos não seguem um *template* – o facto de os documentos não seguirem todos a mesma estrutura, faz com que seja difícil identificar padrões para ajudar na extração dos dados. Não só a informação pretendida poderia estar em qualquer parte do documento, o documento poderia estar dividido em artigos, ou secções, entre outros;
- Inconsistência na identificação dos dados a extrair – esta inconsistência trata-se da forma como eram referidos os dados a extrair. O melhor exemplo é o ”Documentos que Instruem a Proposta”. Durante a exploração dos documentos, esta informação estava em secções intituladas, ”Documentos que Instruem a Proposta”, ”Documentos que Constituem a Proposta”, ”Proposta e Documentos que a Constituem”, entre outros. Isto dificulta ainda mais a identificação dos dados pretendidos.

Foram então, ao longo do desenvolvimento, definidas e testadas diversas expressões para os vários campos a encontrar. As expressões que obtiveram melhores resultados foram as seguintes:

- **Preço base** – para o preço base foi a seguinte expressão:
  `preço base.?[€]?\\s\\d{1,3}([\\.\\s]?\\s\\d{3})\\s,\\s\\d{2}\\s[€]?\\s\\((.*?)\\);`
- **Preço anormalmente baixo** – a expressão utilizada foi a seguinte:
  `(anormalmente baixo){1}\\s*(\\d+)(.*?)(?=%|anormalmente baixo)(.*?)(\\d+\\.));`
- **Critérios de adjudicação** – recorreu-se à seguinte expressão:
  `proposta economicamente mais vantajosa(.*?)(?=\\d+\\.\\d+\\.|\\d+\\.\\d);`
- **Documentos que instruem a proposta** – a expressão definida foi a seguinte:
  `(?=acompanhar as suas propostas|pelos seguintes documentos).*?:(.*?)(?=\\d+\\.\\d+\\.|\\d+\\.\\d).`

No Robot Framework, para escapar caracteres especiais, é utilizado duas barras invertidas. Os resultados apresentados na tabela A.1, foram realizados para os mesmos 30 documentos.

Campos a extrair	Resultados
Preço base	Acertou 22, falhou 8
Preço anormalmente baixo	Acertou 4, falhou 26
Critérios de adjudicação	Acertou 16, deu resposta incompleta para 5, falhou 9
Documentos para a proposta	Acertou 8, deu resposta incompleta para 7, falhou 15

Tabela A.1: Resultados obtidos da implementação com `regex`.

Como referido anteriormente, estes campos apresentados, eram apenas os campos requeridos da extração dos documentos. Como o resultado obtido não foi de todo satisfatório, esta tentativa de implementação foi abandonada antes de se testar com os novos campos. Quanto ao prazo de execução, não foi possível obter uma expressão que obtivesse resultados acertados para múltiplos ficheiros, não tendo sido contabilizada nem a expressão, nem os resultados nestes testes.

A.4 Implementação com Inteligência Artificial

Como a implementação com recurso ao `regex` não produziu os resultados esperados, foi testada a implementação com IA. Esta secção apresenta sucintamente as primeiras abordagens com a IA e os seus resultados, levando ao que foi a solução final com o uso de assistentes.

Foram testados vários modelos de IA, tanto da Gemini como da OpenAI. Porém, por se ter acesso a uma API *key* da OpenAI, o maior esforço recaiu no uso dos vários modelos disponíveis na OpenAI, com a utilização dos modelos da Gemini, como exploração inicial para uma abordagem com a IA.
As tentativas que não fizeram parte da solução final, consistiam no uso do modelo do ChatGPT, onde somente aceitava a utilização de texto. Mais concretamente, foi utilizado a API do `Chat`

`Completions`. A implementação com esta abordagem é bastante semelhante à implementação com o assistente, porém, o *input* era um único *prompt*. Para esta implementação era necessário um cuidado extra com o uso dos *tokens*, para não ultrapassar os limites definidos para o *tier* da API *key*. Estas limitações também existiam no *output* do modelo, o que resultava, por vezes, em respostas incompletas. O modelo principal utilizado foi o `gpt-4o-mini`, uma versão reduzida do `gpt-4o`. Entre as várias versões testadas, a versão baseada em `Chat Completions` com o modelo `gpt-4o-mini` foi a que apresentou os resultados mais promissores para o contexto em análise, com os resultados obtidos apresentados na tabela A.2, para os mesmos 30 concursos usados para testar a implementação com o `regex`.

Campos a extrair	**Resultados**
Preço base	Acertou 23, deu resposta incompleta para 7
Preço anormalmente baixo	Acertou 20, deu resposta incompleta para 3, falhou 7
Critérios de adjudicação	Acertou 18, deu resposta incompleta para 12
Documentos para a proposta	Acertou 10, deu resposta incompleta para 19, falhou 1
Prazo de execução	Acertou 26, deu resposta incompleta para 4

Tabela A.2: Resultados obtidos da implementação do `Chat Completions` com o modelo `gpt-4o-mini`.

A.5 Modelo Físico da Base de Dados

O modelo físico traduz o modelo entidade relacionamento para a implementação a realizar. É apresentado as tabelas e os atributos criados, assim como as suas chaves primárias, chaves estrangeiras, restrições de integridade, entre outros.
A figura A.3, apresenta o modelo físico desenvolvido.

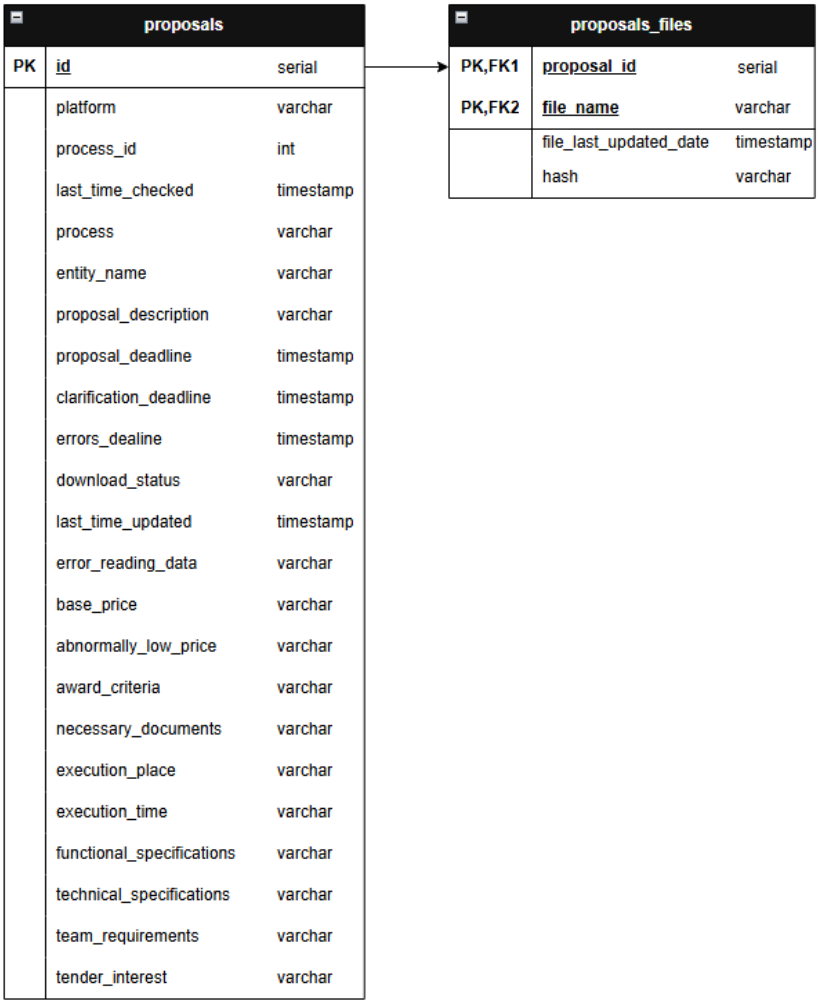

Figura A.3: Modelo físico da base de dados.

Apêndice B

Projeto Cliente

Este anexo apresenta todo o material relativo à implementação da funcionalidade solicitada pelo cliente. Como o foco principal do relatório é o projeto interno da empresa, e não este, foi colocado aqui todo o material necessário para melhor entender a implementação descrita no capítulo 5.

Este anexo foi dividido pela secção B.1, onde apresenta as tecnologias e ferramentas utilizadas para a implementação da funcionalidade solicitada, e como foram utilizadas. A secção B.2, descreve a engenharia de *software* desenvolvida para esta funcionalidade.

B.1 Tecnologias e Ferramentas Utilizadas

Para o desenvolvimento da funcionalidade foram utilizadas as tecnologias e ferramentas descritas nas subsecções que procedem.

B.1.1 Robot Framework

Robot Framework é uma *framework open-source* para automação de testes e RPA, desenvolvida em Python, suportada pela Robot Framework Foundation e usada largamente pela indústria [27]. É *human-friendly* e a sua sintaxe versátil utiliza *keywords* e com suporte extensível a bibliotecas em Python, Java, e outras linguagens. É especialmente eficaz em testes de aplicações *web*, recorrendo à biblioteca `SeleniumLibrary`.

O Robot Framework foi utilizado para o desenvolvimento dos *scripts* e *script complementares* para a automação do processo de extração de dados da plataforma-alvo.

B.1.2 Angular

O Angular é uma *web framework* que capacita os programadores a construir aplicações rápidas e fiáveis [36]. Mantido por uma equipa dedicada da Google, o Angular fornece um vasto conjunto de ferramentas, APIs e bibliotecas que simplificam e agilizam o fluxo de trabalho de desenvolvimento.

O Angular oferece uma plataforma sólida para desenvolver aplicações que escalam tanto com o tamanho da equipa como com a complexidade da base de código.

Foi utilizado no desenvolvimento do *frontend* da aplicação, utilizando a versão 16.

B.1.3 .NET

A plataforma .NET é um ecossistema de desenvolvimento da Microsoft, gratuito e *open-source*, para construir aplicações modernas em múltiplas plataformas [37]. Suporta diversas linguagens de programação, como C#, F# e VB.NET. Inclui bibliotecas, ferramentas e *frameworks* como ASP.NET para aplicações *web* e Entity Framework Core como ORM.

Foi utilizada para o desenvolvimento do *backend* da aplicação *web*, incluindo o *background worker*, utilizando a versão 6 e versão 8 do *Software Development Kit* (SDK), respetivamente.

B.1.4 PostgreSQL

O PostgreSQL é um SGBD relacional *open-source*, conhecido pela sua robustez, escalabilidade e conformidade com os padrões SQL [31]. Suporta tipos de dados avançados, integridade referencial e transações complexas.

Foi utilizado como base de dados para armazenar os dados extraídos do *web scraping* e outros dados necessários para a correta implementação da funcionalidade.

B.1.5 Redis

O Redis é um armazenamento de estruturas de dados em memória, *open-source*, utilizado como base de dados, sistema de *cache* e *message broker* [38]. É conhecido pelo seu alto desempenho, baixa latência e suporte a várias estruturas de dados, sendo especialmente adequado para aplicações que requerem acesso rápido a dados frequentemente utilizados.

Foi utilizado neste projeto como sistema de *cache*, contribuindo para o desempenho da aplicação.

B.1.6 Hangfire

Hangfire permite o agendamento e execução de tarefas em segundo plano em aplicações .NET e .NET Core, não requerendo nenhum serviço do Windows nem um processo separado [39]. É suportado por armazenamento persistente e está disponível como *open-source*, sendo gratuito para utilização comercial.

Suporta diferentes tipos de tarefas (recorrentes, atrasadas, contínuas) e fornece um painel de controlo para monitorização.

Foi utilizada para agendar a execução do *script* para o *web scraping*.

B.1.7 Docker

Docker é uma plataforma aberta para desenvolver, empacotar e executar aplicações [32]. O Docker permite separar as aplicações da infraestrutura, facilitando a entrega rápida de *software*. Garante também que a aplicação funciona consistentemente em qualquer sistema.

O Docker foi utilizado para a *containerização* dos vários componentes da solução, criando ambientes isolados de desenvolvimento e produção, facilitando o CI/CD e garantindo portabilidade, escalabilidade e facilidade de manutenção das várias componentes da solução.

B.1.8 Selenium

O Selenium é um projeto abrangente que inclui um conjunto de ferramentas e bibliotecas para automação de *browsers* [29]. É amplamente utilizado para testes automatizados de *web interfaces*, permitindo simular interações humanas como cliques, preenchimento de formulários e navegação entre páginas.

O Selenium fornece extensões para simular interações de um utilizador com diferentes *browsers*, um servidor de distribuição para escalar a alocação de *browsers* e a infraestrutura necessária para a implementação da especificação W3C *WebDriver*. Esta especificação permite escrever código de automação que funciona em todos os principais *browsers* modernos. A flexibilidade do Selenium e a sua compatibilidade com múltiplas linguagens de programação tornam-no uma ferramenta essencial em ambientes de testes automatizados.

Para este projeto foi utilizado para os *scripts* de *web scraping* e também usado na *containerização* da solução.

B.1.9 DBeaver

O DBeaver é uma ferramenta universal de gestão de base de dados gratuita e *open-source* para desenvolvedores e para administradores de base de dados [33]. Permite interagir com diferentes SGBD, como o PostgreSQL, MySQL, entre outros.

Foi utilizado para visualizar, consultar e administrar a base de dados PostgreSQL, permitindo realizar operações CRUD durante a fase de desenvolvimento.

B.1.10 Visual Studio Code

O Visual Studio Code, mais conhecido como VS Code, é um editor de código-fonte leve, mas poderoso, com suporte a múltiplas linguagens de programação e uma vasta biblioteca de extensões [34]. Oferece funcionalidades como destaque de sintaxe, terminal integrado, controlo de versão e depuração.

Este foi o IDE utilizando para o desenvolvimento dos *scripts* para o *web scraping*, aplicação *web* e do *background worker*, sendo bastante útil na depuração de *bugs*.

B.1.11 Git

O Git é um sistema de controlo de versões distribuído, gratuito e *open-source*, concebido para lidar com tudo, desde pequenos a grandes projetos, com rapidez e eficiência [35]. É amplamente utilizado no desenvolvimento de *software*. Permite rastrear alterações no código-fonte, colaborar com outros programadores e manter um histórico claro do progresso do projeto.

Foi utilizado para gerir as várias versões dos ficheiros do projeto e integração com os vários repositórios remotos no GitLab para o projeto.

B.2 Engenharia de *Software*

Nesta secção é discutida a engenharia de *software* desenvolvida para esta funcionalidade, o levantamento dos requisitos funcionais e não funcionais, o diagrama de casos de uso e outros diagramas considerados relevantes, terminando com a apresentação da modelação da base de dados e arquitetura do sistema.

B.2.1 Requisitos Funcionais e não Funcionais

Os requisitos funcionais descrevem os serviços que um determinado sistema deve fornecer, como deve reagir perante ações específicas dos utilizadores e como se comportará perante situações específicas. Para esta funcionalidade, foram levantados os seguintes requisitos funcionais, descritos na tabela B.1.

Código	Requisito Funcional
RF01	O sistema deve recolher preços de produtos semelhantes em plataformas de *e-commerce* externas
RF02	O processo de *scraping* deve ser executado automaticamente com uma periodicidade semanal
RF03	O sistema deve guardar os preços recolhidos na base de dados, associados ao produto interno
RF04	O sistema deve permitir visualizar, no *backoffice*, o produto interno e os preços recolhidos de outras plataformas numa tabela comparativa
RF05	O sistema deve registar a data da última análise de preços por produto e plataforma
RF06	O utilizador deve autenticar-se na plataforma para aceder ao comparador
RF07	O utilizador deve conseguir adicionar produtos ao comparador para ser analisado para a plataforma pretendida
RF08	O utilizador deve conseguir filtrar os dados presentes na tabela
RF09	O utilizador deve conseguir ordenar a tabela pelo nome do produto, data de adição e data de análise
RF10	O utilizador deve conseguir verificar o histórico de preços de um certo produto para uma certa plataforma ao selecionar a linha da tabela correspondente
RF11	O utilizador deve conseguir modificar alguns dados de um registo para comparação, tais como o nome com que é pesquisado, se o produto deve ser mostrado na tabela e se o produto deve continuar a ser consultado
RF12	O utilizador deve conseguir exportar os dados presentes no comparador para um ficheiro `csv`

Tabela B.1: Requisitos funcionais para o comparador de preços.

Os requisitos não funcionais estabelecem os limites e propriedades que um determinado sistema deve possuir de modo a assegurar o sucesso de todos os requisitos funcionais previamente definidos. na tabela B.1, são apresentados os requisitos não funcionais para o comparador de preços.

Código	Requisito Não Funcional
RNF01	O *scraping* deve ser executado em ambiente isolado, através de *containers* Docker
RNF02	O sistema deve estar preparado para lidar com a ausência de dados de uma ou mais plataformas sem comprometer a funcionalidade
RNF03	O tempo médio de execução do *scraping* por produto não deve ser inferior a 1 minuto
RNF04	Os dados devem ser persistidos numa base de dados PostgreSQL
RNF05	O sistema deve utilizar *cache* para melhorar o desempenho da consulta de preços no *backoffice*
RNF06	O sistema deve ser facilmente extensível para suportar novas plataformas de *e-commerce*
RNF07	Os dados recolhidos devem ser apresentados no *backoffice* com tempos de resposta inferiores a 1 segundo por consulta
RNF08	Apenas utilizadores com permissões de *admin* devem ter acesso ao comparador de preços
RNF09	A página *web* deve ser simples, intuitiva e de fácil utilização para todos os utilizadores
RNF10	A página *web* deve estar em português
RNF11	A página *web* deve ser compatível para os *browsers* mais comuns

Tabela B.2: Requisitos não funcionais para o comparador de preços.

B.2.2 Caso de Uso

Os casos de uso permitem identificar os intervenientes e as interações possíveis com o sistema. Para o caso do comparador a implementar, apenas existe um utilizador, que tem que

ter permissões de *admin* para aceder ao *backoffice* da aplicação *web*. Na figura B.1, é possível observar o diagrama de caso de uso desenvolvido para as principais funcionalidades a implementar.

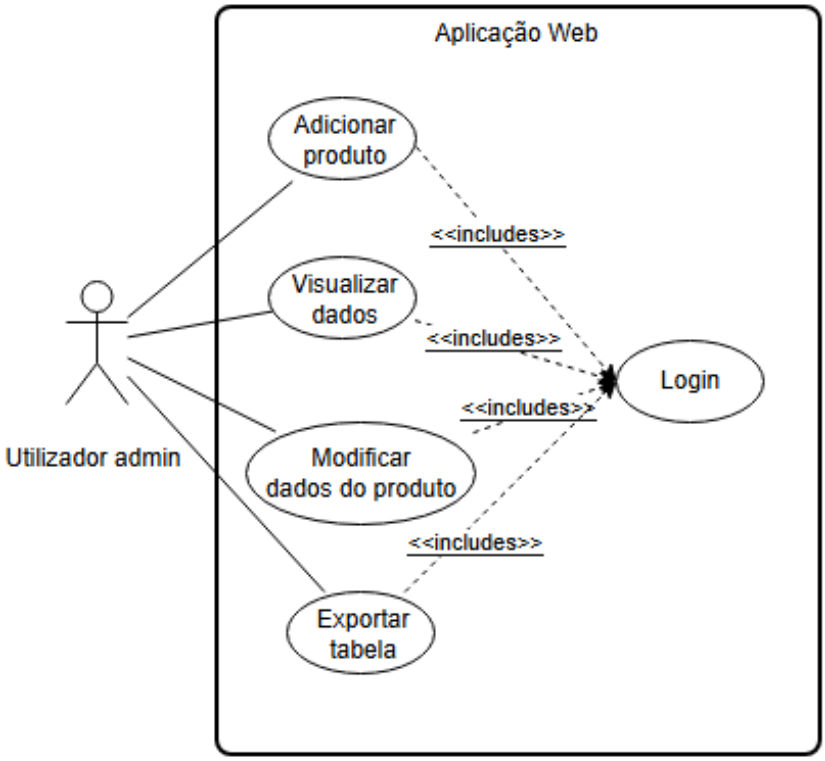

Figura B.1: Diagrama de caso de uso das principais funcionalidades da aplicação *web*.

B.2.3 Diagramas Relevantes

Esta subsecção apresenta os diagramas desenvolvidos durante a implementação da funcionalidade. Foram concebidos diagramas de atividade e *wireframes* para melhor perceber o funcionamento da aplicação *web* e como a página deveria estar implementada.

B.2.3.1 Diagramas de Atividade

O diagrama de atividades caracteriza-se por ilustrar o funcionamento das aplicações para um determinado fim, sendo que estas podem ser executadas quer por utilizadores como por componentes do *software*.

O diagrama presente na figura B.2, mostra o fluxo de ações até se chegar à página pretendida, a página do comparador, existente no *backoffice*. Para todos os diagramas o utilizador representado é um utilizador com permissões de *admin*.

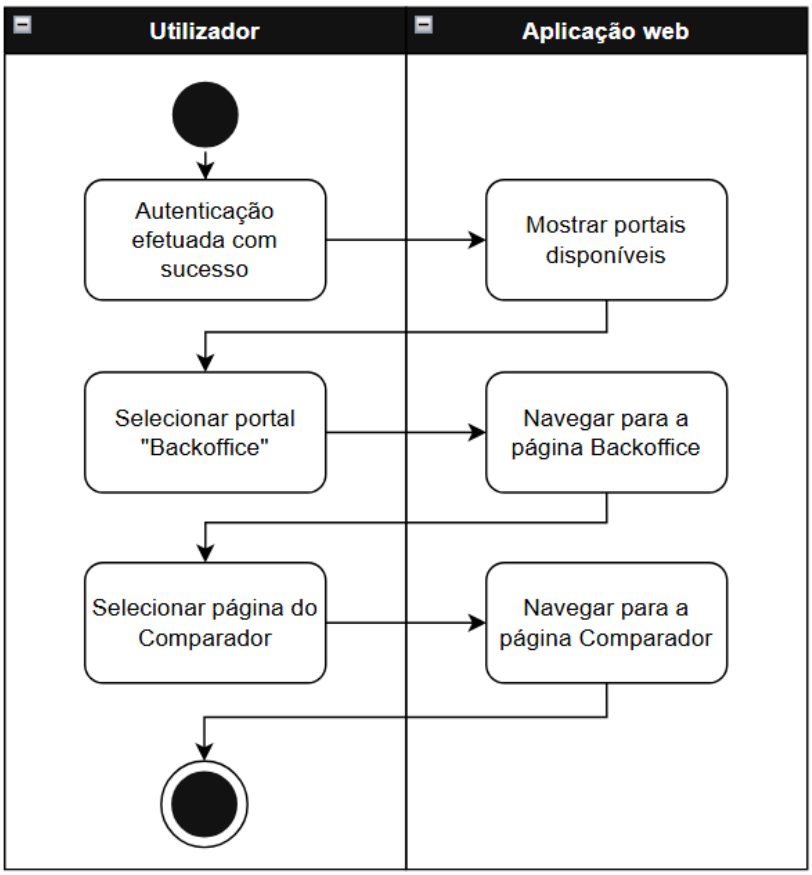

Figura B.2: Diagrama de atividades: navegação para a página do comparador.

Já a figura B.3. ilustra o fluxo de ações para a adição de produtos ao comparador.

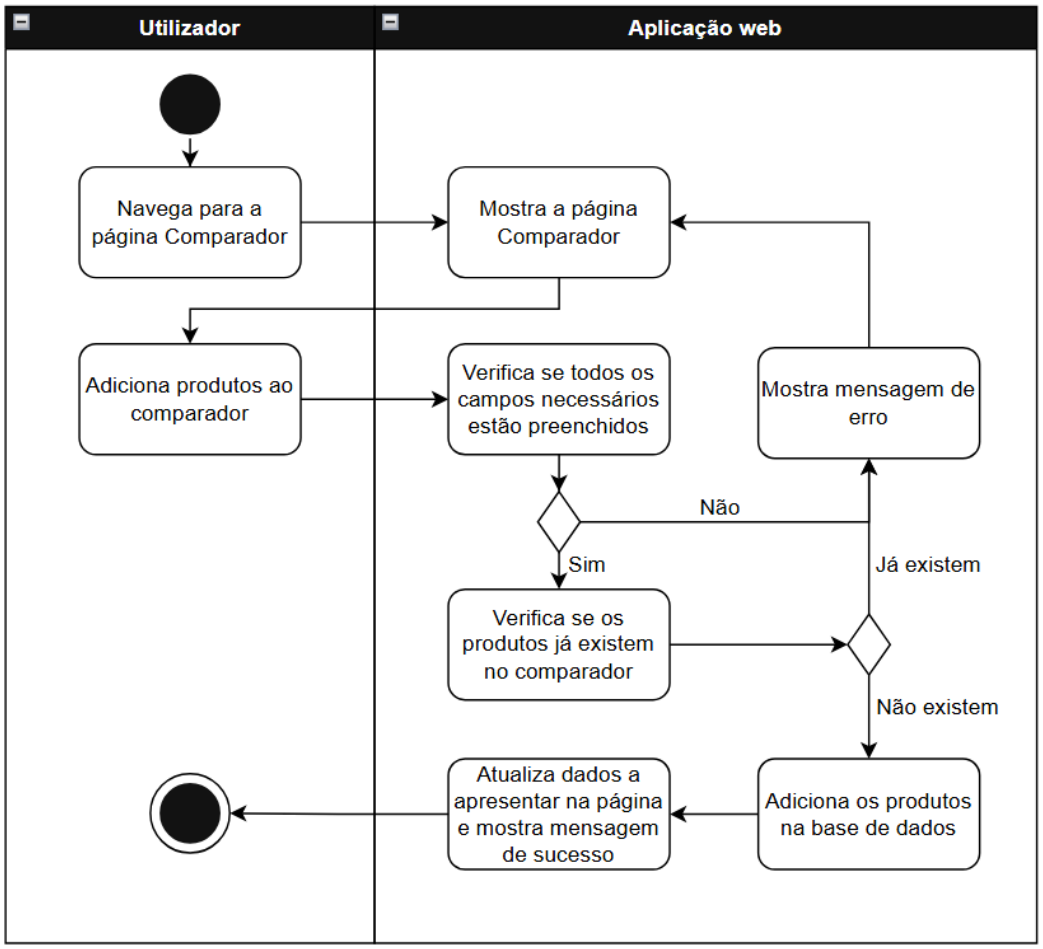

Figura B.3: Diagrama de atividades: adicionar um ou múltiplos produtos ao comparador.

Por fim, é ainda representado o fluxo de ações para a alteração de alguns dados do produto selecionado, que para este caso foi o nome com o qual o produto é procurado nas plataformas *e-commerce* selecionadas. Este diagrama pode ser observado na figura B.4.

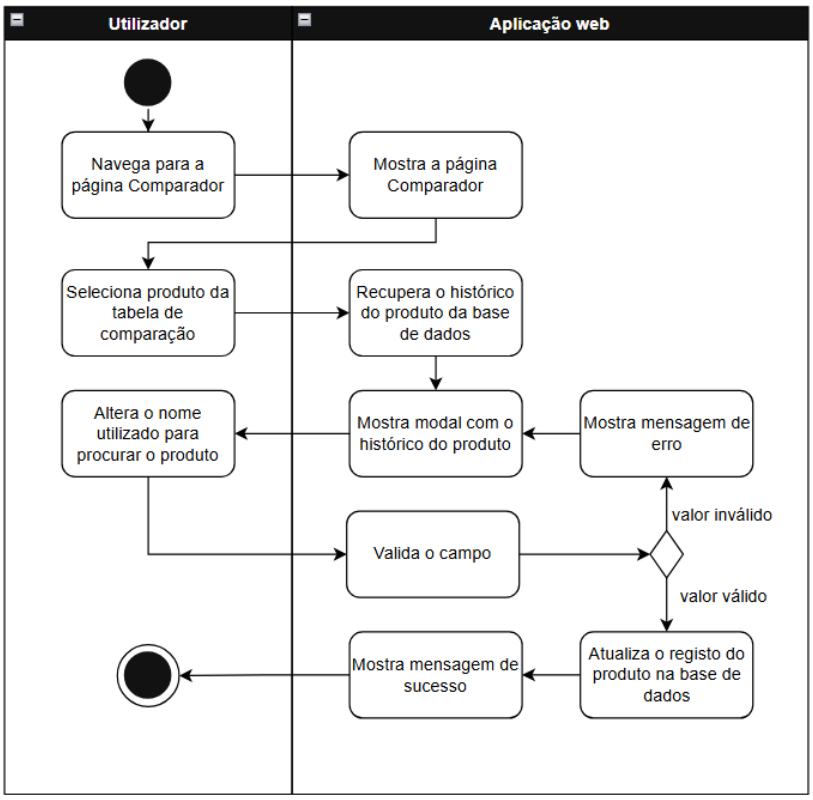

Figura B.4: Diagrama de atividades: modificar o nome utilizado para procurar o produto.

B.2.3.2 *Wireframes*

Wireframe é um rascunho utilizado para demonstrar uma ideia. Pode ser feito recorrendo a ferramentas gráficas ou até com uma caneta e papel. Para este projeto, foi feito alguns rascunhos durante as fases iniciais do desenvolvimento da funcionalidade, com as figuras B.5 e B.6 apresentarem as versões mais completas para a página *web* do comparador e do *modal* com o histórico das comparações dos produtos, respetivamente.

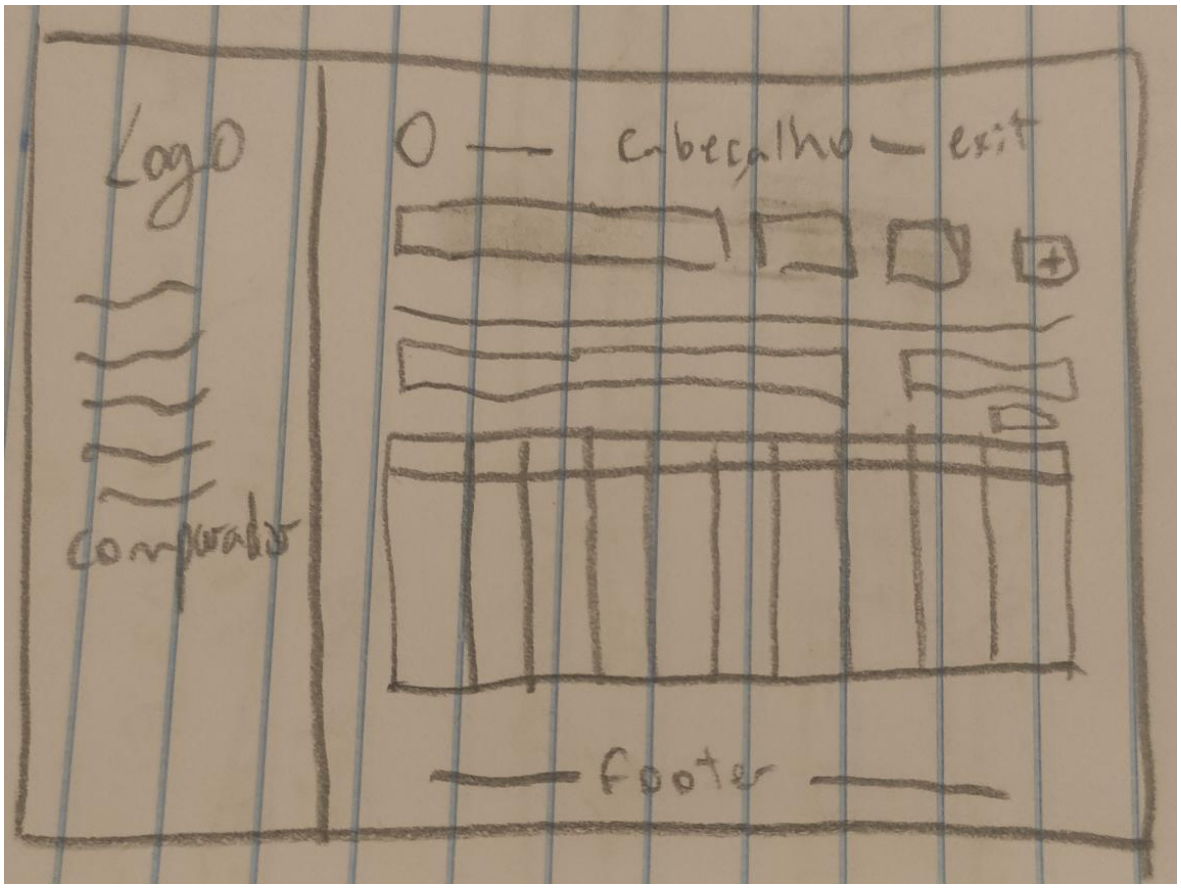

Figura B.5: *Wireframe* da página *web* do comparador.

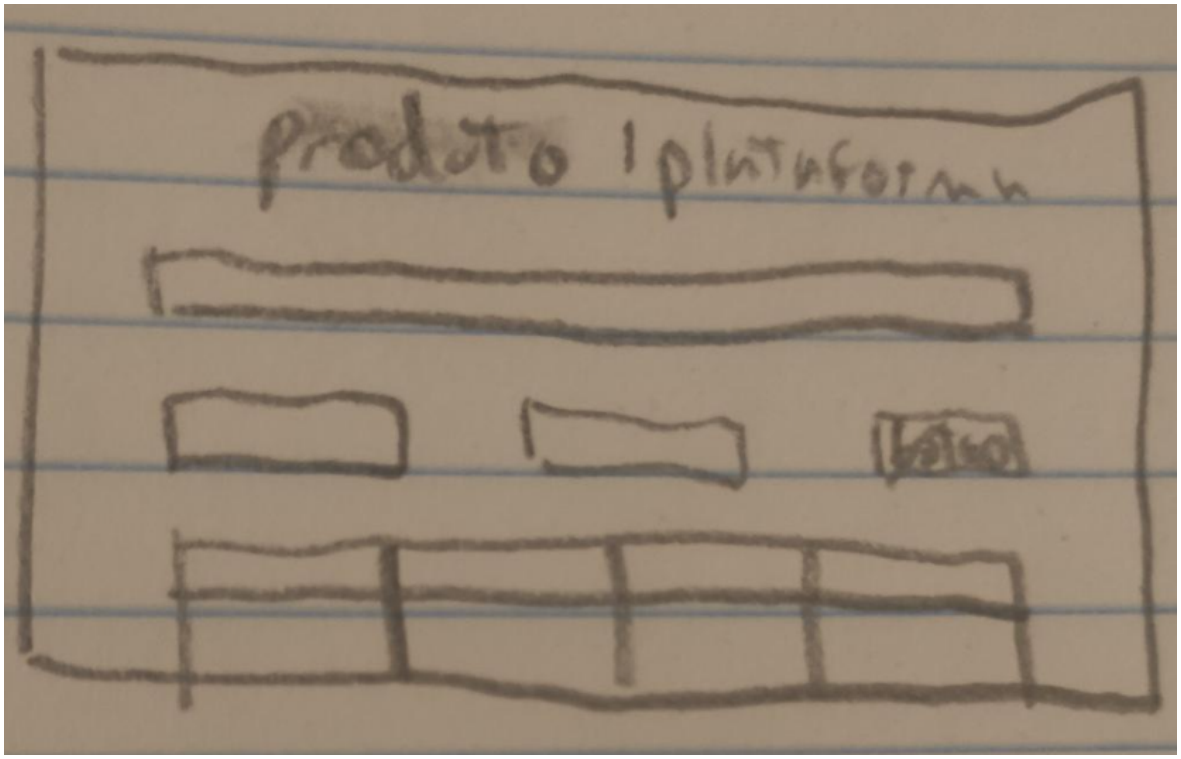

Figura B.6: *Wireframe* do histórico das comparações dos produtos.

B.2.4 Modelação da Base de Dados

Os modelos de base de dados têm como principal foco a esquematização e organização dos dados de uma aplicação, permitindo uma melhor compreensão sobre a mesma. Para a implementação desta funcionalidade, foram criadas três tabelas, sendo que uma delas serve unicamente para guardar o histórico de operações de outra tabela recorrendo a um *trigger*. Nesta secção serão apenas apresentados as tabelas novas criadas para esta funcionalidade, referindo de onde foram obtidos outros valores pertencentes a tabelas já existentes na base de dados.

B.2.5 Modelo Entidade Relacionamento da Base de Dados

O modelo entidade relacionamento define, de forma abstrata e independente da tecnologia de implementação, as entidades, os seus atributos e os relacionamentos existentes entre elas no domínio do problema. O modelo, ilustrado na figura B.7, foi elaborado com base nos requisitos identificados durante a análise do sistema e tem como principal objetivo representar a estrutura lógica dos dados de forma compreensível e organizada.

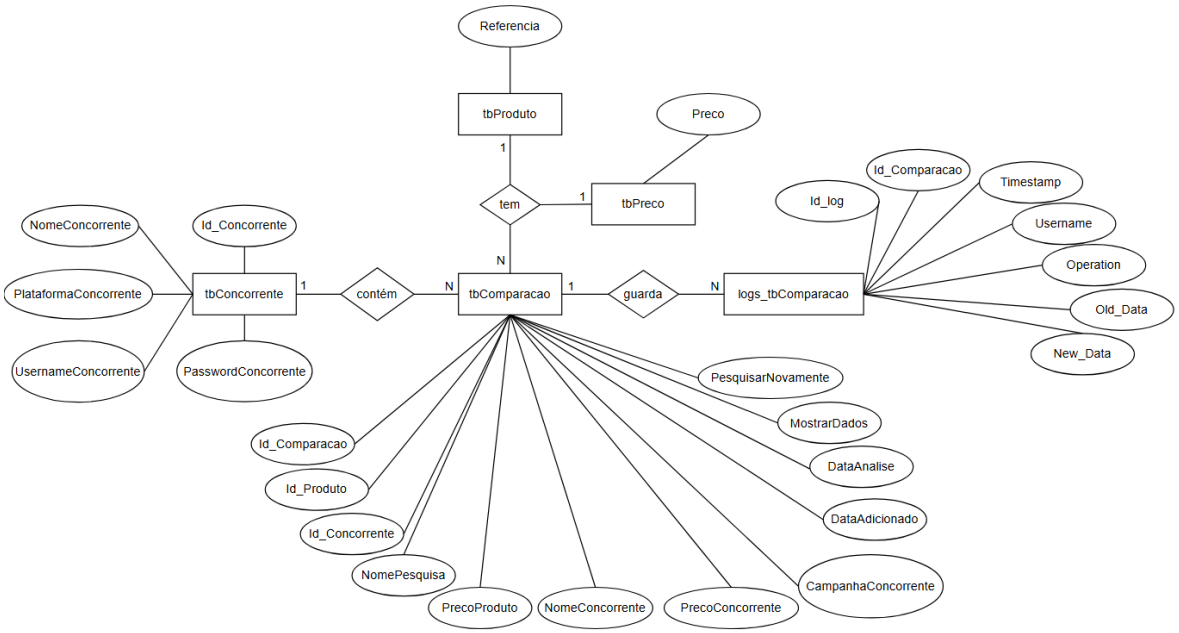

Figura B.7: Modelo entidade relacionamento para a funcionalidade do comparador.

B.2.6 Modelo Físico da Base de Dados

O modelo físico consiste na tradução do modelo entidade relacionamento para uma implementação concreta, adaptada ao PostgreSQL. Esta transformação envolve a criação de tabelas, definição de tipos de dados apropriados, chaves primárias, chaves estrangeiras, entre outras.

A figura B.8, ilustra o modelo desenvolvido, com a descrição das tabelas e dos seus atributos

com mais detalhe após a figura.

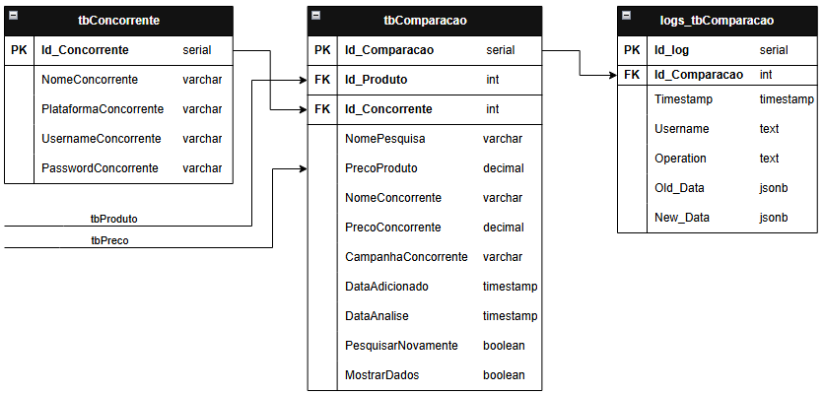

Figura B.8: Modelo físico para a funcionalidade do comparador.

`tbProduto` – esta entidade contém a informação toda para o produto em questão da loja do cliente. Para a implementação desta funcionalidade foi apenas necessário dois campos desta tabela: a `Referencia` e a `Memoria`. A `Referencia` trata-se do identificador interno do produto, onde foi utilizado para estabelecer uma relação entre o produto e a comparação com outras plataformas *e-commerce*. Já a `Memoria`, foi utilizada para pesquisar os produtos nas outras plataformas no momento do *web scraping*.
A `Referencia` e a `Memoria` são do tipo `varchar(255)`, e são `not null`.

`tbPreco` – esta entidade contém os vários preços para o produto. Para a implementação foi apenas necessário o `Preco`, sendo este um `Decimal(10,2)`.

`tbConcorrente` – esta entidade guarda as informações relativas às plataformas a fazer *web scraping*. Esta tabela contém os seguintes atributos:

- `Id_Concorrente` – identificador da tabela, do tipo `serial`, e é a chave primária da tabela;
- `NomeConcorrente` – nome do concorrente, tipo `varchar(255)`, e é `not null`;
- `PlataformaConcorrente` – URL da plataforma, tipo `varchar(255)`, e é `not null`;
- `UsernameConcorrente` – `username` encriptado para ser usado na autenticação da plataforma, tipo `varchar(255)`, e é `not null`;
- `PasswordConcorrente` – palavra-passe encriptada para ser usada na autenticação da plataforma, tipo `varchar(255)`, e é `not null`.

`tbComparacao` – esta entidade guarda os registos das diversas comparações, para os vários produtos, para as várias plataformas. Os produtos marcados para comparação, são guardados aqui, sendo atualizados sempre que for feito o *web scraping*. Esta entidade tem os seguintes atributos:

- `Id_Comparacao` – identificador da tabela, do tipo `serial`, e é a chave primária da tabela;
- `Id_Produto` – referência do produto da tabela `tbProduto`, tipo `varchar(255)`, e é `not null`;
- `Id_Concorrente` – identificador da plataforma da tabela `tbConcorrente`, tipo `int`;
- `NomePesquisa` – nome utilizado quando se encontrou uma correspondência na comparação, tipo `varchar(255)`;
- `PrecoProduto` – preço do produto da loja, presente na tabela `tbPreco`, tipo `decimal(10,2)`;
- `NomeConcorrente` – nome do produto encontrado na plataforma concorrente, tipo `varchar(255)`;
- `PrecoConcorrente` – preço do produto encontrado na plataforma concorrente, tipo `decimal(10,2)`;
- `CampanhaConcorrente` – campanhas encontradas na plataforma concorrente para o produto em questão, tipo `varchar(255)`;
- `DataAdicionado` – data da adição do produto para comparação, tipo `timestamp(0)`, e é `not null`;
- `DataAnalise` – data da última análise, tipo `timestamp(0)`;
- `PesquisarNovamente` – marca se este produto para a plataforma em questão, é para continuar a ser analisado, tipo `boolean`, com valor por defeito igual a `True`;
- `MostrarDados` – marca se este produto para a plataforma em questão, é para ser visualizado na tabela de comparação no *backoffice*, tipo `boolean`, com valor por defeito igual a `True`.

`logs_tbComparacao` – esta entidade mantém registo de todas as operações efetuadas na tabela `tbComparacao`, com recurso a um *trigger*. Esta entidade tem os seguintes atributos:

- `Id_log` – identificador da tabela, tipo `serial`, e é a chave primária da entidade;
- `Id_Comparacao` – identifica a que registo da tabela `tbComparacao` pertence este registo, tipo `int`, e é `not null`;
- `Timestamp` – guarda o *timestamp* com o fuso horário, tipo `timestampz`, e é inserido por defeito;
- `Username` – guarda o nome de quem fez a operação na tabela `tbComparacao`, tipo `text`, e é `not null`;
- `Operation` – indica qual a operação realizada, tipo `text`, e é `not null`;
- `Old_Data` – guarda os valores da tabela `tbComparacao` antes da operação, tipo `jsonb`;
- `New_Data` – guarda os valores da tabela `tbComparacao` após a operação, tipo `jsonb`.

B.2.7 Arquitetura do Sistema

A arquitetura da solução para o comparador de preços foi também desenhada segundo uma abordagem modular e distribuída. A Figura B.9 ilustra a arquitetura geral do sistema. A solução consiste num *docker-compose* com os serviços necessários para o correto funcionamento do projeto, com os *containers* nomeados de "*Comparator*" e "*Selenium*", a lidar com o *web scraping*, a aplicação *web* separada pelo *backend* e *frontend*, o *background worker* e o Redis, e com a base de dados PostgreSQL num *container* fora do *docker network* da restante solução.

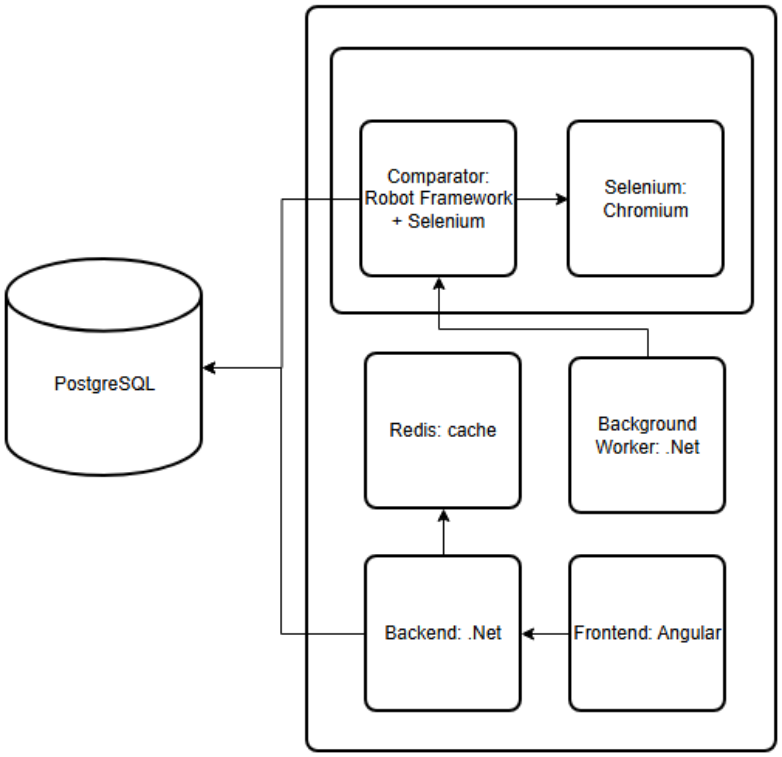

Figura B.9: Arquitetura do sistema para a funcionalidade do comparador.

Glossário

API	*Interface* que permite a comunicação entre diferentes aplicações via um conjunto de regras e definições.
ASP.NET	*Framework* de desenvolvimento da Microsoft para a criação de aplicações *web* dinâmicas.
Backend	Parte da aplicação responsável pela lógica de negócio, acesso a dados e regras do servidor.
Base de dados vetorial	Tipo de base de dados otimizada para armazenar, indexar e consultar vetores de alta dimensão, geralmente usados em aplicações de IA, como recuperação semântica e sistemas com modelos de linguagem.
C#	Linguagem de programação moderna, segura e orientada a objeto que abrange desde recursos de alto nível, como registos orientados a dados, até recursos de baixo nível, como ponteiros de função.
Cache	Armazenamento temporário de dados usados frequentemente, utilizado para melhorar o desempenho do sistema.
Cookie	Ficheiro de texto armazenado no navegador do utilizador, contendo informações que permitem manter sessões, preferências ou rastrear atividades em páginas *web*.
Container	Unidade leve e portátil que agrupa uma aplicação e todas as suas dependências, garantindo que funcione consistentemente em diferentes ambientes.
Containerização	Processo de empacotar *software* e as suas dependências em *containers*, promovendo portabilidade, consistência e escalabilidade.
CLI	*Interface* baseada em linha de comandos que permite ao utilizador interagir com o sistema ou aplicação através da introdução de comandos em texto, sem recurso a *interfaces* gráficas.

`CRON`	Agendador de tarefas recorrentes em sistemas Unix/Linux, usado para executar *scripts* ou comandos automaticamente.
CRUD	Refere-se às quatro operações básicas utilizadas em sistemas de gestão de dados.
Data-driven	Abordagem baseada na utilização intensiva de dados para tomar decisões ou guiar processos.
DevOps	Conjunto de práticas que integra equipas de desenvolvimento e operações visando automatizar e integrar processos de desenvolvimento, testes e entrega de *software*.
DoS	Ataque que visa tornar um sistema ou serviço indisponível sobrecarregando-o com tráfego malicioso.
E-commerce	Comércio eletrónico — compra e venda de bens ou serviços pela internet.
Entity Framework Core	ORM da Microsoft que permite mapear objetos .NET para bases de dados relacionais.
Fluxograma	Diagrama que representa visualmente o fluxo de um processo ou sistema, usando símbolos padronizados para ilustrar decisões, operações e sequências de atividades.
Framework	Conjunto de ferramentas, bibliotecas e boas práticas que oferece uma estrutura base para o desenvolvimento de aplicações.
Frontend	Parte visual de uma aplicação com a qual o utilizador interage diretamente.
`GET`	Método HTTP utilizado para solicitar dados de um recurso específico.
GitLab	Plataforma de *DevOps* baseada em Git que permite gestão de repositórios e CI/CD.

Header	Parte da mensagem HTTP que transporta informações de controlo, como tipo de conteúdo, autenticação ou definição de idioma, entre cliente e servidor.
Headless browser	Navegador *web* que funciona sem *interface* gráfica, permitindo simular a navegação como um utilizador real, ideal para testes automatizados e extração de dados.
HTTP	Protocolo de comunicação usado para transferir dados na *web* entre clientes (como *browsers*) e servidores.
Learn as you go	Processo de aprendizagem contínua e adaptativa. Significa adquirir conhecimento e habilidades por meio da experiência prática.
LINQ	Linguagem de consulta integrada no .NET para manipulação de dados de forma declarativa.
Low-code	Abordagem de desenvolvimento que requer pouca ou nenhuma codificação manual, geralmente utilizando *interfaces* visuais.
MVC	Padrão de arquitetura de *software* que separa a aplicação em três componentes principais: *Model*, que gere os dados e regras de negócio, *View*, que apresenta os dados ao utilizador, e *Controller* que gere a interação entre a *view* e o *model*.
MySQL	SGBD relacional amplamente utilizado, conhecido pelo seu desempenho e simplicidade.
ORM	Técnica que permite manipular bases de dados relacionais por meio de objetos em linguagens de programação orientadas a objetos.
PHP	Linguagem de programação interpretada e amplamente usada no desenvolvimento de aplicações *web* do lado do servidor.
`POST`	Método HTTP usado para enviar dados para o servidor, geralmente para criar ou atualizar um recurso.

Process-driven	Modelo de desenvolvimento centrado nos processos da organização como base da construção da solução.
Prompts multimodais	Comandos ou instruções para modelos de IA que combinam diferentes tipos de entrada, como texto, imagem ou som.
Proxy/Proxies	Servidor intermediário entre o cliente e a página de destino, usado para contornar bloqueios, proteger a identidade ou distribuir requisições em tarefas como *web scraping*.
Query	Pedido de informação enviado a uma base de dados ou a uma API, normalmente utilizando uma linguagem como SQL ou parâmetros em URLs.
Redes neurais	Conjunto de algoritmos inspirados na estrutura do cérebro humano, usados em IA para identificar padrões e resolver problemas complexos.
Scrum	Metodologia ágil para gestão de projetos que enfatiza entregas iterativas e trabalho em equipa.
SGBD	*Software* que permite criar, manter e manipular bases de dados.
SQL	Linguagem usada para comunicar com bases de dados relacionais. Permite consultar, inserir, atualizar e apagar dados.
Teardown	Processo de finalização ou limpeza executado após a execução de um teste, ou tarefa, geralmente utilizado para restaurar o ambiente ao seu estado original.
Temperature	Parâmetro usado em modelos de linguagem para controlar a aleatoriedade das respostas geradas. Valores mais baixos produzem respostas mais previsíveis, enquanto valores mais altos geram resultados mais variados e criativos.

Token	Unidade básica de texto processado por modelos de linguagem. Pode representar uma palavra inteira, parte de uma palavra ou mesmo apenas um símbolo. A contagem de *tokens* é usada para limitar o tamanho do *input/output* dos modelos.
Top_p	Também conhecido como *nucleus sampling*, é um parâmetro que limita a soma cumulativa de probabilidade dos *tokens* considerados na geração de texto. Apenas os *tokens* mais prováveis, cuja soma atinge um determinado limite, são considerados, promovendo maior controle e fluidez nas respostas.
Trigger	Mecanismo de uma base de dados que executa automaticamente uma ação definida (como *insert*, *update* ou *delete*) em resposta a determinados eventos numa tabela ou *view*.
TypeScript	Linguagem de programação desenvolvida pela Microsoft que é um superconjunto do JavaScript. Acrescenta tipagem estática e recursos modernos de desenvolvimento, ajudando a escrever código mais robusto e fácil de manter.
URL	Endereço utilizado para localizar e aceder a recursos na *internet*. Contém normalmente o protocolo, o domínio e, opcionalmente, o caminho e parâmetros.
XPath	Linguagem usada para navegar em documentos XML e selecionar elementos ou atributos.
W3C	Consórcio internacional que desenvolve padrões para garantir o crescimento e a compatibilidade da *web*. É responsável por especificações como o HTML, CSS e *WebDriver*.
WebDriver	*Interface* padronizada pelo W3C que permite controlar *browsers* de forma programática, usada por ferramentas como o Selenium.