UNIVERSIDADE
BEIRA INTERIOR

Automação da Procura de Concursos Públicos de Interesse e Extração Automática de Informação dos Documentos que os Constituem

Pedro Miguel Mateus Paixão

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

Orientador: Prof. Doutor Bernardo Sequeiros
Supervisor da Entidade de Acolhimento: Líder Técnico João Gouveia

Outubro de 2025

Declaração de Integridade

Eu, Pedro Miguel Mateus Paixão, que abaixo assino, estudante com o número de inscrição M13566 de Engenharia Informática da Faculdade de Engenharia, 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 15/10/2025

Assinado por: **Pedro Miguel Mateus Paixão**
Data: 2025.10.15 11:07:33+01'00'

Agradecimentos

Gostaria de expressar o meu profundo agradecimento a todas as pessoas que, de forma direta ou indireta, contribuíram para a realização deste estágio e para a concretização deste relatório. Para além do esforço individual, o sucesso desta etapa só foi possível graças ao apoio, orientação e incentivo de todos os que me acompanharam.

Agradeço à Latitudde e, em especial ao Líder Técnico João Gouveia, por me lançar este desafio, oferecendo uma oportunidade de realizar um projeto interessante e de grande potencial e pela confiança que em mim depositou.

Agradeço ao meu colega de equipa Fernando Cruz, com o qual desenvolvi esta ferramenta, e que me acompanhou desde o primeiro dia de estágio, sempre disponível para ajudar e partilhar conhecimento.

Agradeço à minha família e em especial aos meus pais, Carla e Manuel, que possibilitaram a minha formação superior e que sempre me deram todas as condições para que a conseguisse concluir.

Agradeço à minha namorada, Francisca, pelo amor, pela paciência e pelo incentivo a dar sempre o melhor de mim.

Agradeço aos meus amigos, que me proporcionam momentos incríveis todos os dias e com os quais partilhei os melhores anos da minha vida.

Um agradecimento especial ao meu orientador, o Professor Doutor Bernardo Sequeiros, que tinha já sido uma enorme ajuda no projeto de licenciatura e que agora me acompanhou, mais uma vez, neste estágio. Pela sua disponibilidade a todas as horas, pela sua compreensão e empatia, pela sua amizade, pelo seu aconselhamento, pelo seu rigor e profissionalismo, o meu mais sincero obrigado.

Resumo

Este relatório apresenta o desenvolvimento de um programa, realizado no contexto de um estágio para a empresa *Latitudde* (entidade de acolhimento), destinado a automatizar o processo de procura e transferência de concursos públicos alocados em plataformas online e a extração de informações relevantes para a análise do interesse da empresa em participar nesses concursos.
Os concursos públicos são procedimentos formais através dos quais entidades governamentais e organizações públicas contratam bens e serviços. Para a *Latitudde*, a participação nesses concursos representa uma oportunidade estratégica de crescimento e expansão dentro da sua área de atuação. No entanto, cada concurso é constituído por múltiplos documentos, incluindo o Caderno de Encargos, o Programa do Concurso, o anúncio e eventuais anexos. Esses documentos são frequentemente extensos e, embora contenham informações similares, a sua estrutura varia significativamente entre diferentes concursos e plataformas.
Atualmente, o departamento de recursos humanos da empresa dedica um tempo considerável à análise desses documentos para determinar se a *Latitudde* reúne as condições necessárias para concorrer. O objetivo do programa desenvolvido é automatizar esse processo, extraindo rapidamente as informações essenciais e permitindo uma decisão imediata sobre o interesse da empresa na candidatura. Dessa forma, a solução proposta visa otimizar o tempo e os recursos dedicados a essa tarefa, reduzindo a necessidade de processos manuais repetitivos e garantindo uma análise mais ágil e precisa das oportunidades de mercado.

Palavras-chave

Automatização, *Robotic Process Automation*, *Web Scrapping*, Processamento Inteligente de Documentos, Modelos Linguísticos de Grande Dimensão, Desenvolvimento *Front-end*, Desenvolvimento *Back-end*

Abstract

This report presents the development of a program, carried out as part of an internship at *Latitudde* (host entity), aimed at automating the process of searching for and retrieving public tenders from online platforms, as well as extracting relevant information to assess the company's interest in participating in these tenders.
Public tenders are formal procedures through which government entities and public organizations procure goods and services. For *Latitudde*, participating in these tenders represents a strategic opportunity for growth and expansion within its field of activity. However, each tender consists of multiple documents, including the Tender Specifications, the Tender Program, the announcement, and various annexes. These documents are often extensive and, while they contain similar information, their structure varies significantly across different tenders and platforms.
Currently, the company's human resources department spends a considerable amount of time analyzing these documents to determine whether *Latitudde* meets the necessary requirements to apply. The developed program aims to automate this process by quickly extracting essential information and enabling an immediate decision regarding the company's interest in the application. Thus, the proposed solution seeks to optimize the time and resources allocated to this task, reducing the need for repetitive manual processes and ensuring a more efficient and accurate analysis of market opportunities.

Keywords

Automation, Robotic Process Automation, Web Scrapping, Intelligent Document Processing, Large Language Models, Front-end Development, Back-end Development

Índice

Lista de Figuras

Lista de Excertos de Código

Lista de Tabelas

Lista de Acrónimos

AC Automação Cognitiva

ADN Ácido desoxirribonucleico

API *Application Programming Interface*

CRM *Customer Relationship Management*

DCR *Dorset Care Record*

DL *Deep Learning*

DOM *Domain Object Model*

ERP *Enterprise Resource Planning*

GUI *Graphical User Interface*

IA Inteligência Artificial

LLM *Large Language Model*

ML *Machine Learning*

RAG *Retrieval Augmented Generation*

RH Recursos Humanos

RPA *Robotic Process Automation*

TI Tecnologias de Informação

UBS Union Bank of Switzerland

EUA Estados Unidos da América

GPT *Generative Pre-trained Transformer*

MoE *Mixture-of-Experts*

HTML *HyperText Markup Language*

JSON *JavaScript Object Notation*

CSV *Comma-Separated Values*

XML *Extensible Markup Language*

CSS *Cascading Style Sheets*

SDK *Software Development Kit*

INEM Instituto Nacional de Emergência Médica

PDF *Portable Document Format*

GPU *Graphics Processing Unit*

RAM *Random Access Memory*

JSONL *JavaScript Object Notation Lines*

OCR *Optical Character Recognition*

CAPTCHA *Completely Automated Public Turing test to tell Computers and Humans Apart*

PHP *Hypertext Preprocessor*

GPQA *Graduate-level Google-Proof Q&A*

Capítulo 1

Introdução

1.1 Enquadramento

Este relatório foi produzido no contexto da unidade curricular de Projeto de Estágio, do programa de 2º ciclo de estudos do curso de Engenharia Informática na Universidade da Beira Interior no ano letivo 2024/2025. A entidade de acolhimento é a empresa 'Latd Digital Enablers, Lda', doravante designada por *Latitudde* no contexto deste documento e o estágio está a ser realizado nos escritórios da empresa que estão situados no Fundão, de 11 de novembro de 2024 até 30 de maio de 2025. No âmbito deste estágio serão abordados os temas de Automação Robótica de Processos e Processamento Inteligente de Documentos, passando também por diversas áreas durante o desenvolvimento de uma solução, como *Web Scraping*, Bases de Dados, Ciência de Dados e Inteligência Artificial. Durante o estágio realizar-se-ão formações no uso de ferramentas específicas para a tarefa em mãos, bem como a introdução do estagiário no ambiente corporativo do desenvolvimento de software.

1.2 Sobre a Empresa

A *Latitudde* [2] é uma empresa de tecnologia, especializada na entrega de projetos completos em diversas *stacks* tecnológicas e linguagens de programação, seguindo consistentemente as melhores práticas de mercado. Foi fundada em 2021 e posiciona-se no setor de consultoria digital, oferecendo soluções detalhadas para ir ao encontro das necessidades únicas do negócio dos clientes. A empresa pertence, juntamente com mais seis outras empresas de tecnologia, ao Grupo RIT [3], que possui mais de 500 colaboradores e clientes em mais de 25 localizações ao redor do mundo. A *Latitudde* possui equipas em Portugal, Países Baixos, Chile, Peru, Brasil, Paraguai, Nova Zelândia e, mais recentemente, em Espanha e Sérvia, oferecendo serviços especializados em todas as suas operações.

1.3 Motivação

Desde os primórdios da humanidade, a procura por soluções que facilitassem as tarefas do dia-a-dia foi fundamental para a evolução da sociedade. Nos tempos antigos, a sobrevivência dependia da capacidade de caçar e coletar alimentos, atividades que exigiam grande parte do tempo e energia das pessoas. No entanto, a humanidade foi capaz de desenvolver ferramentas e técnicas que simplificaram essas atividades, permitindo que mais tempo fosse dedicado a outras áreas, como a arte, a ciência e a construção de comunidades mais complexas.
Um exemplo claro dessa evolução é a agricultura. Ao descobrir formas de cultivar e armazenar alimentos, a humanidade automatizou, em certo grau, o processo de produção de ali-

mentos. Isso não apenas garantiu a subsistência com mais segurança, mas também permitiu o surgimento de civilizações mais estruturadas. Avançando para os dias atuais, a produção e distribuição de alimentos estão tão automatizadas que a maioria das pessoas tem acesso a uma grande variedade de produtos nos supermercados, sem necessidade de se envolver diretamente no cultivo ou na produção.
No contexto atual, a automação de tarefas atravessa diversas áreas, desde a produção industrial até aos serviços das tecnologias da informação, resultado da procura por produtividade, economização de tempo e redução de esforço em tarefas repetitivas.
Neste relatório de estágio, apresenta-se o desenvolvimento de um programa que automatiza o processo de transferência de concursos públicos de plataformas *web*, bem como a extração de informações necessárias para a análise do interesse da empresa em participar nesses concursos. Este projeto procura reduzir a complexidade e o tempo gasto na análise de oportunidades, possibilitando que os recursos humanos da empresa sejam direcionados para tarefas que realmente precisam da criatividade e da tomada de decisão, gerando valor e competitividade no mercado.

1.4 Objetivos

Os principais objetivos deste estágio curricular passam por:

- **Preparação para o Mercado de Trabalho**: Proporcionar ao estagiário uma experiência prática no ambiente colaborativo de uma empresa de *software*, expondo-o às dinâmicas e práticas comuns no setor de engenharia informática.

- **Desenvolvimento de *Software* de Automação**: Criar um programa que automatize a tarefa de busca e análise de concursos públicos em diferentes plataformas, simplificando a identificação de concursos de interesse para a empresa. O programa também deverá extrair informações relevantes dos documentos associados, facilitando a tomada de decisões estratégicas.

- **Formação em Tecnologias e Métodos de Trabalho**: Capacitar o estagiário nas tecnologias específicas utilizadas no desenvolvimento do programa e nos métodos de trabalho adotados pela empresa, contribuindo para o crescimento profissional e técnico do estagiário.

1.5 Estrutura do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o âmbito e a estrutura deste documento e a motivação que está na origem do trabalho a ser desenvolvido no contexto de estágio curricular, bem como o seu enquadramento e objetivos;

2. O segundo capítulo – **Estado da Arte** – apresenta uma revisão da literatura existente sobre os temas abordados no estágio, identificando as principais tendências e desenvolvimentos nessa área, bem como as soluções existentes para o problema em questão, as suas limitações e possíveis melhorias;
3. O terceiro capítulo – **Plano de Estágio** – descreve as atividades planeadas para o período de estágio, detalhando as fases do projeto, os objetivos a serem alcançados em cada etapa e a metodologia adotada para a execução das tarefas.

Capítulo 2

Estado da Arte

2.1 Introdução

Este capítulo apresenta os principais conceitos que fundamentam o trabalho realizado durante o estágio, fornecendo o contexto teórico e prático para as atividades desenvolvidas. Entre os temas abordados, destacam-se: *Robotic Process Automation* (RPA), uma tecnologia que possibilita a execução automática de tarefas repetitivas e padronizadas; *Web Scraping*, técnica essencial para a extração de dados estruturados da web, viabilizando a recolha de informações de forma eficiente e organizada; metodologias para o processamento inteligente de documentos, com recurso a tecnologias avançadas, como *Large Language Model*s (LLMs), oferecendo soluções inovadoras para o processamento de linguagem natural e a compreensão contextual.
Além disso, este capítulo aborda as tendências e soluções atuais do mercado relacionadas a essas áreas, contribuindo para um entendimento abrangente das abordagens disponíveis e as suas aplicações práticas.

2.2 *Robotic Process Automation*

RPA é uma tecnologia que permite a automação de processos repetitivos e baseados em regras, utilizando "robôs" de *software* que simulam ações humanas, como cliques, preenchimento de formulários, *scrolling* e extração de dados, permitindo testar sistemas e interagir diretamente com interfaces de utilizador. A grande vantagem de utilizar RPA é a de não ser necessário fazer alterações nos sistemas em que é implementado, sendo uma solução não invasiva e de fácil adoção em diferentes contextos empresariais.
RPA é uma evolução natural das ferramentas de automação, pois permite que tarefas administrativas, frequentemente repetitivas e demoradas, sejam executadas de forma rápida, consistente e escalável. Além disso, destaca-se pela sua capacidade de operar 24/7, reduzindo custos operacionais e minimizando erros causados por fadiga ou distração humana.
Esta tecnologia visa substituir pessoas por automação feita de maneira 'de fora para dentro', diferindo da abordagem clássica 'de dentro para fora' para melhorar os sistemas de informação [4].

2.2.1 Características

RPA é uma solução atrativa em cenários corporativos e de desenvolvimento de *software*, pois:

- é baseado em regras: os robôs de RPA executam tarefas claramente definidas e estruturadas, onde cada passo do processo é detalhado e replicado de forma precisa [5] [6] [4]. Esta abordagem é particularmente adequada para tarefas que envolvem processos onde o fluxo de trabalho é fixo, e que pode ser descrito como uma sequência de instruções claras e não permitem uma margem muito ampla de variações ou erros, como preenchimento de formulários, processamento de faturas, testes de regressão ou testes de *Graphical User Interface* (GUI), entre outros;
- é não invasiva: ao contrário de abordagens tradicionais, ferramentas de RPA interagem diretamente com as interfaces de utilizador, sem necessidade de modificar os sistemas originais;
- é flexível: permite integração com diversas aplicações e plataformas, incluindo *legacy software* e bases de dados modernas.
- contribui para a eficiência operacional de uma organização: a automação reduz o tempo necessário para executar tarefas e aumenta a produtividade das equipas, que podem focar-se em atividades mais estratégicas.

2.2.2 *Robotic Process Automation* vs. Automação Tradicional

Embora tanto o RPA quanto a automação tradicional visem a execução de tarefas de forma automatizada, existem diferenças fundamentais entre as duas abordagens, tal como mostra a tabela 2.1.

2.2.3 Principais Ferramentas

O mercado de RPA é amplamente dominado por ferramentas como o *UiPath* [7], *Blue Prism* [8] e *Automation Anywhere* [9], que oferecem soluções comerciais robustas para empresas. [10] Estas ferramentas são reconhecidas pela sua facilidade de uso, interfaces intuitivas e suporte técnico.
O começo daquilo que hoje é o *UiPath* remonta para o ano de 2005, quando uma empresa de *outsourcing* se apercebeu da elevada procura por soluções de RPA no mercado e tomou a decisão de começar a construir uma plataforma padrão para desenvolver robôs de *software*. A sua tecnologia é atualmente utilizada em milhões de máquinas no mundo inteiro, integrado em diferentes produtos e empresas nas mais diversas áreas. [10] A plataforma oferece integração com diversas tecnologias e aplicações, como sistemas *Enterprise Resource Planning* (ERP) e *Customer Relationship Management*s (CRMs), além de contar com robustas funcionalidades de análise e monitorização.
A *Automation Anywhere*, que até 2010 era conhecida como *Tethys Solutions*, é uma ferramenta poderosa de RPA que se destaca pela sua capacidade de integrar automação com Inteligência Artificial (IA). [10] Oferece uma plataforma na nuvem, a edição *Server*, que permite desenvolver processos de automação em ambiente colaborativo com segurança centralizada, o que proporciona grande flexibilidade e escalabilidade para as empresas. Com a capacidade de integrar IA e *Machine Learning* (ML), a *Automation Anywhere* torna-se uma

Tabela 2.1: Principais diferenças entre RPA e Automação Tradicional [1].

Critério	RPA	Automação Tradicional
Integração com Sistemas Existentes	Não requer modificações nos sistemas ou infraestruturas existentes, pois interage diretamente com a interface de utilizador.	Pode requerer alterações no código do sistema existente, exigindo integração no *back-end* do sistema.
Capacidade de Imitar Ações Humanas	Foca-se na automação de tarefas repetitivas e baseadas em regras, imitando ações humanas, como clicar em botões ou inserir dados.	Executa instruções pré-definidas, sem a capacidade de imitar diretamente as ações humanas na interface de utilizador.
Requisitos de Programação	Mais acessível ao comum utilizador, sendo possível automatizar processos através de interfaces intuitivas, como fluxogramas e ferramentas *no-code* ou *low-code*, sem necessidade de conhecimentos avançados de programação.	Requer habilidades de programação para desenvolver e implementar a automatização, dependendo da ferramenta utilizada.
Tempo de Implementação	Implementação rápida devido à sua abordagem orientada a processos e à ausência de necessidade de alterações nos sistemas subjacentes.	Pode demorar vários meses para ser implementada, incluindo fases de planeamento, desenvolvimento e testes de viabilidade.
Escalabilidade	Permite a atribuição de tarefas a múltiplas máquinas virtuais, executando processos em larga escala sem necessidade de hardware adicional.	A escalabilidade pode ser limitada pela infraestrutura existente e pode exigir investimentos adicionais em hardware e software.
Flexibilidade	Fácil de adaptar a mudanças no sistema, devido à simplicidade da tecnologia.	Pode ser necessário fazer alterações em vários scripts para conseguir acompanhar alterações no sistema, tornando difícil as fases de manutenção e atualização do sistema.

escolha excelente para empresas que procuram automatização inteligente, capaz de aprender e se adaptar ao longo do tempo.
A *Blue Prism* foi fundada em 2001, quando um grupo de profissionais especialistas em automação de processos reconheceu uma enorme necessidade de automação nas atividades administrativas internas realizadas por trabalhadores de escritório [10].
O *Robot Framework* é uma ferramenta *open-source* de RPA, projetada para tornar o processo de criação, manutenção e execução de testes mais simples e eficiente. Desenvolvido com foco em flexibilidade e extensibilidade, utiliza uma abordagem baseada em palavras-chave (*Keyword-Driven Testing*), permitindo que os testes sejam escritos num formato legível e fácil de entender, mesmo por pessoas que não possuem conhecimentos avançados em programação. Esta é uma das ferramentas que irá ser utilizada para automatizar a procura de concursos públicos de interesse nas plataformas *online*.

2.2.4 Áreas de Aplicação e Exemplos de Caso de Uso

Atualmente o RPA é usado para otimizar muitos tipos de processos repetitivos em diferentes setores, sendo adotado por uma variedade de indústrias, incluindo [11]:

- Serviços Financeiros, realizando a automação de processamento de faturas, de conciliação de contas e da geração de relatórios financeiros. A título de exemplo, existe o caso

do banco suíço Union Bank of Switzerland (UBS) que, durante a pandemia do COVID--19, enfrentou um grande volume de pedidos de empréstimos a taxa zero aprovados pelo governo suíço. Sem infraestrutura suficiente para lidar com a alta demanda, acumulou 10.000 pedidos em atraso. Para solucionar esse problema, a equipa do UBS implementou um sistema de RPA, o que provocou uma redução do tempo de processamento de cada empréstimo de 40 minutos para apenas 5 minutos. Essa automação permitiu acelerar a análise e libertação dos empréstimos, otimizando recursos e reduzindo o trabalho manual [11].

- Recursos Humanos, facilitando a triagem de currículos, a gestão de admissões, o registo de funcionários e o cálculo de benefícios. Um exemplo é o de uma empresa líder no fabrico de aço nos Estados Unidos da América (EUA), que enfrentou desafios no processo de integração de novos funcionários devido à dispersão das suas operações pelo país. A necessidade de coordenação entre diferentes departamentos, como Recursos Humanos (Recursos Humanos (RH)) e Tecnologias de Informação (Tecnologias de Informação (TI)), resultava em atrasos e baixa produtividade. Para solucionar este problema, a empresa implementou RPA para automatizar a leitura e processamento de documentos dos trabalhadores, acionar cartas de oferta e gerir a atribuição de *logins* e acessos aos sistemas da organização. Como resultado, o tempo e os custos foram reduzidos, a satisfação dos funcionários aumentou e o processo de integração tornou-se mais eficiente e sem erros [12].

- Setor Público, agilizando o processamento de formulários, extração de dados de portais governamentais e validação de informações. Um caso de uso que se pode referir é o da cidade de Seattle, no estado de Washington, nos EUA, que é a casa de cerca de 750 000 habitantes e possui uma grande variedade de serviços públicos, sendo que já integrou mais de 60 automações inteligentes em diferentes áreas. Isto resultou na resolução de 6200 casos de serviços públicos, economizando (segundo [13] e até à data de publicação do caso de uso) 600 horas no processamento de dados e 1500 horas de trabalho manual.

- *E-commerce*, dando suporte à monitorização de preços, gestão de inventário e respostas automatizadas ao cliente. Um exemplo disto é o caso da *Cobmax*, um centro de vendas global. Quando a empresa assumiu um contrato com uma grande operadora de telecomunicações brasileira, o seu sistema administrativo manual tornou-se um obstáculo devido ao volume crescente de trabalho e aos erros frequentes. Ao implementar uma solução RPA, a *Cobmax* automatizou tarefas manuais como a transferência de dados entre sistemas CRM, o que reduziu significativamente os erros e o tempo gasto em tarefas repetitivas. Como resultado, a empresa conseguiu reduzir pela metade as operações de *back-office* e diminuiu em até 66% o tempo necessário para a produção de relatórios, otimizando a produtividade e a satisfação do cliente [14].

- Cuidados de Saúde, onde é usado na gestão de registos médicos, faturação e gestão de seguros, agendamento de consultas e monitorização de pacientes. Temos como exemplo disto o caso do sistema de saúde pública do Reino Unido, que implementou uma

solução de RPA para facilitar o acesso dos médicos de clínica geral ao *Dorset Care Record* (DCR), um sistema de registos clínicos [15]. Para garantir um processo seguro e eficiente, foram criadas automaticamente contas para 1500 médicos, permitindo-lhes aceder rapidamente ao histórico dos pacientes. Com isto, os médicos poupam tempo na obtenção de informações médicas e podem prestar um atendimento mais ágil e preciso, solucionando um grande desafio enfrentado pelo sistema de saúde.

No contexto do presente projeto, o RPA será utilizado para automatizar a recolha de documentos de concursos públicos em plataformas *online*. Essa abordagem reduz significativamente o tempo e o esforço necessário para realizar manualmente essas tarefas, proporcionando maior eficiência no acesso às informações.

2.2.5 Limitações

Embora a RPA tenha ganho uma grande popularidade devido às suas vantagens evidentes, como o aumento da eficiência e redução de custos e erros humanos, é importante notar que não é uma solução ideal para todos os problemas organizacionais. Pela sua natureza repetitiva e baseada em regras pré-definidas, esta tecnologia falha em lidar com operações mais complexas que não possuem um conjunto de regras estruturado. Neste tipo de casos, é ainda necessário a utilização de modelos avançados de ML ou a incomparável capacidade de análise e de tomada de decisão do ser humano. Existe também o problema de adaptação a ambientes inconstantes, como é o caso das interfaces de utilizador que sofrem mudanças na sua estrutura de tempos em tempos, o que pode levar à necessidade de alterar o robô para se adaptar a estas mudanças. Por fim existe a preocupação com questões de segurança e governança, especialmente para empresas em setores regulamentados. A forma como os robôs lidam com credenciais de *login* e dados sensíveis, como monitorizam transações suspeitas e perdas de dados e como se integram em protocolos de segurança existentes são preocupações que necessitam de soluções abrangentes para evitar riscos [16, 17].

2.2.6 Tendências Futuras

O futuro do RPA aponta para uma evolução significativa, com a crescente integração de tecnologias de IA e *Deep Learning* (DL), o que dá origem ao conceito de Automação Cognitiva (AC) [18]. A automação cognitiva permite que os processos automatizados se tornem mais inteligentes e capazes de lidar com tarefas mais complexas, até então realizadas apenas por seres humanos. Com isso, a RPA evolui para ser mais do que apenas um substituto para tarefas repetitivas: passa a ser uma ferramenta capaz de interpretar contextos complexos e oferecer uma automação mais eficiente e adaptável. A integração das LLMs, por exemplo, permite que os robôs compreendam e processem documentos não estruturados de forma mais eficaz, extraiam informações essenciais, façam análises preditivas e até mesmo gerem relatórios baseados em dados brutos de forma autónoma. Além disso, a automação cognitiva também pode melhorar a experiência do cliente, com *bots* capazes de interagir de maneira mais natural e personalizada com os consumidores, proporcionando respostas mais contextuais e adaptativas. Porém, apesar destas inovações, ainda existem desafios a ser superados,

como a integração efetiva destas tecnologias, a necessidade de treinar modelos e a adaptação organizacional, sendo que as empresas vão precisar de desenvolver as suas infraestruturas e conhecimento para integrar essas tecnologias de forma eficiente [19].
Prevê-se que a Automação Cognitiva transforme várias indústrias e empresas como a *UiPath* e *Automation Anywhere* já estão a investir fortemente no desenvolvimento deste tipo de soluções cognitivas.

2.3 *Web Scraping*

Web Scraping refere-se ao processo de extração automatizada de dados da web. Sempre que um utilizador copia manualmente informações de uma página web e as cola num documento de texto, por exemplo, está a realizar uma forma rudimentar de *Web Scraping*. Para tarefas simples, como obter uma imagem de um repositório *online* para uso numa apresentação, esse método manual pode ser suficiente. No entanto, quando a necessidade envolve a recolha de grandes volumes de dados, seja para fins pessoais, académicos ou empresariais, torna-se essencial recorrer a abordagens automatizadas que tornem o processo mais eficiente e escalável.
Neste contexto, o conceito de *Web Scraping* explorado neste documento refere-se ao uso de software capaz de simular a navegação num *browser* (replicando a interação humana) para aceder, extrair e processar informações de diferentes *websites*. Esta abordagem oferece vantagens significativas, como a velocidade de recolha de dados, a capacidade de automação e a possibilidade de programar a extração de forma recorrente e sistemática.
Ao longo desta secção, serão exploradas algumas das principais abordagens utilizadas para *Web Scraping*, as ferramentas mais relevantes para a sua implementação e as aplicações práticas desta tecnologia em diferentes domínios [20].

2.3.1 Abordagens

- **Abordagem do Mimetismo** - Baseia-se na reprodução do comportamento humano ao interagir com uma página web e é frequentemente implementada com ferramentas como *Selenium* e *Puppeteer*. O *scraper* imita as ações de um utilizador real, como navegar entre páginas, clicar em botões, preencher formulários e até mesmo executar ações que simulam movimentos do rato e tempos de espera variáveis, localizando componentes da página através de *Domain Object Model (DOM) selectors*. Embora esta abordagem seja eficaz para contornar restrições simples, pode ser mais lenta devido à necessidade de renderizar completamente as páginas, além de estar sujeita a restrições impostas por sistemas *anti-bot* que analisam padrões de navegação.

- **Abordagem da Medição de Peso** - Nesta abordagem, o *scraper* analisa os elementos da página e atribui um peso a diferentes partes do código *HyperText Markup Language* (HTML). Esse peso é baseado em diferentes características, como a frequência de determinadas palavras-chave, a estrutura dos elementos HTML e a posição relativa dos dados na página. É útil para extrair informações de páginas que seguem um padrão

estruturado e cujo conteúdo varia apenas ligeiramente. Por exemplo, num site que contenha uma lista de produtos, onde os preços, descrições e imagens seguem um formato previsível, esta técnica pode ser usada para identificar e extrair automaticamente os dados mais relevantes.

- **Abordagem Diferencial** - Baseia-se no princípio de que duas páginas do mesmo site geralmente possuem uma estrutura fixa, diferindo apenas no conteúdo principal. Elementos como barras de navegação, menus laterais e rodapés tendem a permanecer idênticos entre diferentes páginas do mesmo domínio, enquanto a informação relevante é atualizada dinamicamente. O método utilizado nesta abordagem consiste na aplicação de uma máscara que compara duas versões de uma página e sobrepõe os seus elementos estáticos, removendo apenas as diferenças encontradas. Desta forma é possível identificar rapidamente as mudanças no conteúdo principal, sem a necessidade de processar toda a página repetidamente. Como resultado, o processo de *scraping* é mais eficiente e discreto pois minimiza acessos redundantes ao site e reduz o volume de dados analisado. Esta técnica é muito utilizada para monitorizar atualizações de preços e acompanhar notícias.
- **Abordagem Baseada em *Machine Learning*** O princípio fundamental desta abordagem consiste em treinar um algoritmo de ML utilizando um grande conjunto de páginas previamente analisadas, permitindo que o modelo aprenda padrões na disposição dos conteúdos. Através de medições estatísticas, o algoritmo determina onde normalmente se encontra o bloco principal de informação e compara-o com os demais elementos da página. Com o tempo, e à medida que o modelo é exposto a mais dados, a precisão na identificação dos conteúdos relevantes melhora significativamente. A principal vantagem desta abordagem é a sua capacidade de lidar com páginas não estruturadas e adaptar-se a diferentes *layouts*, o que a torna uma boa alternativa a métodos baseados em regras fixas. Por outro lado, esta abordagem requer um volume significativo de dados de treino e pode ter um custo computacional elevado.

2.3.2 Principais Ferramentas

A escolha da ferramenta de *Web Scraping* depende do nível de complexidade da extração de dados, do tipo de site a ser analisado e do grau de automação necessário. Nesta secção, serão apresentadas algumas das principais ferramentas utilizadas para *Web Scraping*, organizadas em três categorias – extensões de *browser*, plataformas, e *frameworks* [21]:

1. **Extensões de *browser*:** São ferramentas que permitem realizar *Web Scraping* diretamente dentro do ambiente do navegador, sem a necessidade de escrever código. Estas ferramentas facilitam a extração de dados ao possibilitar a seleção visual de elementos nas páginas, com a capacidade de exportar as informações para formatos como *Comma-Separated Values* (CSV) ou JSON. São particularmente úteis para tarefas simples de *scraping*, onde a estrutura dos dados é previsível e bem definida. Exemplos comuns de extensões de *browser* incluem *Web Scraper*, *Data Scraper - Easy Web Scraping* e *Instant Data Scraper*, que são bastante utilizadas para a recolha de dados de

páginas web com *layouts* estruturados. Estas ferramentas permitem uma abordagem mais prática e acessível para utilizadores que não possuem experiência em programação, mas necessitam de realizar extrações pontuais de informações de sites.

2. **Plataformas:** São soluções baseadas em computação na nuvem ou serviços de *software* que oferecem funcionalidades avançadas para a automação da extração de dados, podendo ser configuradas de forma intuitiva para atender às necessidades de utilizadores sem experiência em programação. Plataformas como *ParseHub*, *Octoparse*, *Bright Data*, *WebHarvy* e *Content Grabber* são algumas das mais completas e mais usadas ferrramentas do mercado.

3. ***Frameworks:*** São constituídos por bibliotecas de programação projetadas para facilitar as tarefas envolvidas em todas as fases do processo de *web scraping*. Oferecem funcionalidades avançadas para navegação, extração, processamento e armazenamento de informações, permitindo maior eficiência e escalabilidade na extração de grandes volumes de dados. Embora exijam um certo nível de conhecimento em programação e composição web, estes *frameworks* permitem maior personalização e podem ser integrados facilmente noutros sistemas, tornando-se uma escolha ideal para projetos mais complexos e escaláveis. Frameworks como *Scrapy*, *Selenium*, *BeautifulSoup*, *Puppeteer* e *Playwright* são alguns dos mais reconhecidos e utilizados do mercado.

2.3.3 Áreas de Aplicação

No mundo atual, os dados tornaram-se um dos ativos mais valiosos para empresas e organizações, comparáveis ao próprio capital financeiro. A tomada de decisões estratégicas, o desenvolvimento de novos produtos e a personalização da experiência de utilizador dependem fortemente da capacidade de recolher, processar e interpretar grandes volumes de dados. A internet, como principal fonte de dados, abriga uma imensidão de informações, tanto relevantes quanto irrelevantes, estruturadas e não estruturadas, verdadeiras e falsas. Diante desse cenário, torna-se essencial adotar métodos eficientes para extrair e tratar esses dados de forma organizada e confiável. A seguir, são apresentadas algumas das principais áreas em que o *web scraping* desempenha um papel fundamental, demonstrando a sua aplicabilidade em diferentes setores [20, 22]:

- **Geração de *Leads* para *Marketing*:** Empresas utilizam o *web scraping* para recolher informações de contacto, como *e-mails* e números de telefone, de repositórios *online* e listas de empresas. Por exemplo, é possível extrair dados de plataformas como as *Páginas Amarelas* ou listas de negócios no *Google Maps*, facilitando campanhas de *marketing* direcionadas.

- **Comparação de Preços e Monitorização da Concorrência:** No setor de comércio eletrónico, *web scraping* é usado para comparar a variação dos preços de produtos e serviços oferecidos por concorrentes. As empresas podem rastrear alterações de preços em tempo real, ajustando as suas estratégias de gestão de custos para manter competitividade no mercado.

- **Imobiliário:** No setor imobiliário, o *web scraping* auxilia na recolha de detalhes de propriedades exibidas em sites como *Zillow* e *Realtor*. Além disso, é possível extrair informações de contato de agentes e proprietários, facilitando a prospeção e análise de mercado.

- **Pesquisas académicas:** Em ambientes académicos, o *web scraping* é uma ferramenta valiosa para recolher dados estruturados de múltiplas fontes, apoiando pesquisas em diversas disciplinas. Por exemplo, estudos em biomedicina utilizam o *web scraping* para extrair informações de genes e proteínas de bases de dados *online*.

- **Treino de Modelos de *Machine Learning* e *LLMs*:** É crucial dispor de conjuntos de dados extensivos e diversificados para treinar modelos inteligentes. O *web scraping* facilita a recolha desses dados de diferentes *websites*, contribuindo diretamente na qualidade e precisão do modelo.

- **Jornalismo:** No jornalismo, o *web scraping* assiste na investigação para a escrita de notícias, pois permite acompanhar muito mais informação de diferentes fontes do que apenas um humano.

- **Sistemas de Recomendação:** Muitos sistemas de recomendação utilizam *web scraping* para recolher dados que alimentam algoritmos de filtragem. Por exemplo, plataformas como a *Amazon* empregam esses sistemas para sugerir produtos aos utilizadores com base nas suas preferências e comportamentos de navegação.

2.4 *Large Language Models*

Os LLMs são um tipo de modelo de inteligência artificial que utiliza técnicas de ML para compreender e gerar linguagem humana. Estes modelos têm vindo a desempenhar um papel cada vez mais relevante no panorama tecnológico, sendo amplamente adotados por empresas e organizações para automatizar processos de comunicação e análise de dados. A principal característica dos LLMs reside na sua capacidade de processar vastos volumes de informação, sendo treinados em conjuntos de dados de grande escala. O termo "*large*" refere-se precisamente à dimensão destes modelos, cujo número de parâmetros pode ultrapassar os milhares de milhões nas versões mais avançadas da atualidade. Para atingir um desempenho elevado, estes modelos utilizam dados extraídos da Internet, abrangendo diversas fontes, como artigos científicos, livros e interações em plataformas digitais. O funcionamento dos LLMs baseia-se em técnicas de *deep learning* (que se enquadra na área de ML), permitindo a identificação de padrões e relações entre caracteres, palavras e frases através de uma análise probabilística de dados não estruturados. Este processo possibilita que o modelo aprenda sem supervisão humana explícita, tornando-se capaz de compreender o contexto e gerar respostas coerentes em diferentes domínios do conhecimento. Após a fase inicial de treino, os LLMs podem ser ajustados para tarefas específicas através de um processo conhecido como *fine-tuning*. Este ajuste pode ser realizado com novos conjuntos de dados ou através da adaptação das respostas mediante *prompts* específicos, conferindo flexibilidade ao modelo para

diferentes tipos de aplicação. A evolução dos LLMs tem favorecido avanços significativos na área da IA, na otimização de processos empresariais e no desenvolvimento de novas soluções tecnológicas baseadas em processamento de linguagem natural [23].

2.4.1 Principais Aplicações

Os LLMs têm uma ampla gama de aplicações, sendo utilizados para automatizar tarefas complexas e otimizar processos em diversas áreas. Devido à sua capacidade de compreender e processar linguagem natural em profundidade, as suas áreas de aplicação são praticamente infinitas, uma vez que conseguem interpretar e gerar conteúdos relacionados com qualquer conceito familiar ao ser humano. Um dos fins de utilização mais conhecidos é a sua aplicação como uma IA generativa, um tipo de modelo que pode produzir texto, imagens, código de programação e conteúdo audiovisual em resposta a *prompts* fornecidos pelo utilizador. Além da sua capacidade de geração de conteúdo, os LLMs estão a ser muito utilizados para desempenhar diversas funções, incluindo:

- **Assistência em programação** – Modelos como o *GitHub Copilot* auxiliam programadores ao sugerir pedaços de código e corrigir erros;
- **Processamento inteligente de documentos** – Aplicação essencial para a automação e análise de documentos, permitindo a extração de informações relevantes e a organização de grandes volumes de dados textuais. Este é um dos casos de uso mais relevantes para o projeto em desenvolvimento durante este estágio;
- **Análise de sentimentos** – Empresas utilizam esses modelos para interpretar emoções em avaliações de clientes, redes sociais e outras fontes;
- **Atendimento ao cliente e *chatbots*** - LLMs são cada vez mais integrados nos sistemas de atendimento ao cliente, permitindo que *chatbots* e assistentes virtuais ofereçam respostas mais precisas e contextualizadas. Esses modelos podem ser incorporados diretamente nos *websites* e plataformas das empresas, proporcionando interações mais naturais e eficientes;
- **Investigação na área da genética** - Estes modelos estão a ser aplicados na interpretação de sequências de Ácido desoxirribonucleico (ADN), devido à sua capacidade de processar e analisar grandes volumes de dados biológicos. Ao tratar as sequências genéticas como cadeias de texto, os LLMs podem identificar padrões e prever estruturas tridimensionais de ácidos nucleicos, auxiliando na compreensão de funções genéticas e na identificação de mutações associadas a doenças;
- **Pesquisas *online*** - Motores de busca integram LLMs para fornecer respostas mais precisas e contextuais.

2.4.2 Principais Modelos

Atualmente, diversas empresas desenvolvem e mantêm LLMs avançados, cada um com características específicas para diferentes aplicações. Estes modelos podem ser acedidos de

duas maneiras principais: através de interfaces de utilizador, ou através de APIs para integração em sistemas mais complexos. Muitos LLMs são disponibilizados com interfaces intuitivas, permitindo que os utilizadores comuniquem com o modelo de maneira semelhante a uma conversa por mensagem. Nestas interfaces, os utilizadores conseguem enviar textos, documentos ou até imagens, e observar as respostas do modelo em tempo real, facilitando a interação e obtenção de informações. Além disso, a maioria desses modelos oferece APIs, o que permite que equipas de programadores integrem essas potentes ferramentas diretamente no seu *software*. Esta vertente oferece a possibilidade de personalizar o modelo conforme o contexto de aplicação. A evolução contínua destes modelos tem gerado novas possibilidades de inovação, tornando-os cada vez mais acessíveis e úteis tanto para o público geral quanto para engenheiros de *software*, o que amplia as suas aplicações no quotidiano empresarial e pessoal. Entre os modelos mais poderosos e adotados, destacam-se [24]:

- Os modelos *Generative Pre-trained Transformer* (GPT) da *OpenAI*, incluindo o *ChatGPT-4o* e o *ChatGPT-4o mini*, representam um grande avanço em relação às versões anteriores, oferecendo maior velocidade de processamento e capacidades de lidar com texto, áudio e imagens. Com mais de 175 mil milhões de parâmetros e uma janela de contexto de 128.000 *tokens*, estes modelos são altamente eficientes em gerar e processar grandes volumes de dados. No entanto, como são modelos proprietários, o acesso completo aos seus recursos exige uma licença ou assinatura comercial;

- O *DeepSeek-R1*, da empresa chinesa de IA *DeepSeek*, é um modelo *open-source* que estabelece novos padrões de inovação neste setor. Com 671 mil milhões de parâmetros *Mixture-of-Experts* (MoE) e 37 mil milhões de parâmetros ativados por *token*, o *DeepSeek-R1* destaca-se pela sua capacidade de raciocínio e excelente desempenho em tarefas complexas, como matemática e geração de código. É 30 vezes mais eficiente e 5 vezes mais rápido que o *OpenAI-o1*, oferecendo alto desempenho a um custo reduzido. O modelo também se destaca em tarefas de reconhecimento de padrões complexos, como análise de dados genéticos e imagens médicas. Além disso, o *DeepSeek-R1* é altamente eficaz na integração com dados corporativos, como informações pessoais e registos financeiros, utilizando a técnica de *Retrieval Augmented Generation* (RAG) para personalizar interações;

- O *QWQ-32B-Preview*, o modelo mais recente da empresa *Alibaba*, é um modelo experimental com 32,5 mil milhões de parâmetros que se destaca em tarefas complexas, superando os modelos existentes em áreas como escrita de código e resolução de problemas matemáticos e lógicos. Mesmo sendo um modelo em fase de testes, obteve resultados superiores ao *OpenAI o1-preview* e ao *GPT-4* em desempenho matemático. Atualmente disponível para testes em plataformas como a *Hugging Face*, o modelo possui uma abordagem única de raciocínio ao verificar as suas respostas através de planeamento e autovalidação;

- O *EXAONE 3.0*, o LLM desenvolvido pela *LG AI Research*, lançado em dezembro de 2024, tem um excelente desempenho em diversos *benchmarks* e aplicações práticas.

Com 7,8 mil milhões de parâmetros, o modelo é capaz de entender e gerar texto humano em várias línguas e em áreas complexas como programação, matemática, patentes e química. Foi otimizado para reduzir em 56% o tempo de processamento, 35% o uso de memória e 72% os custos operacionais, mantendo alta performance. A versão de *EXAONE 3.0* foi disponibilizada como *open-source* para fins de investigação não comercial;

- O modelo multimodal *LLaMA 3.2*, lançado em setembro de 2024 pela empresa *Meta*, destaca-se pela capacidade de processar tanto texto quanto imagens, permitindo realizar análises mais profundas e gerar respostas contextualizadas, como a interpretação de gráficos, mapas e a tradução de textos extraídos de imagens. Está disponível em versões com 8, 70 e 405 mil milhões de parâmetros, oferecendo uma gama flexível para diferentes aplicações. Com uma janela de contexto de 128.000 *tokens*, o modelo é capaz de lidar com entradas de dados extensas e complexas simultaneamente, tornando-o adequado para tarefas que exigem análise de grandes volumes de informação. Tal como o *DeepSeek-R1*, o *LLaMA 3* é um modelo *open-source*, o que permite aos utilizadores implementá-lo e adaptá-lo conforme as necessidades da sua infraestrutura e requisitos específicos de segurança e personalização.

2.5 Conclusão

Neste capítulo foi apresentada uma revisão das principais áreas tecnológicas que fundamentam o projeto desenvolvido no âmbito do estágio. Após o estudo dos conceitos descritos neste capítulo, conclui-se que uma implementação conjunta de técnicas e ferramentas de RPA e *Web Scraping* são uma solução eficaz para a recolha de concursos públicos. Já as LLMs irão possibilitar a interpretação e extração inteligente de informações a partir de documentos complexos. A escrita deste capítulo revelou ser de extrema importância uma vez que a compreensão destas tecnologias e das suas aplicações é fundamental para o desenvolvimento do projeto de estágio. Com base nesta revisão, o próximo capítulo abordará o planeamento e algumas das ferramentas que irão ser utilizadas para a implementação da solução.

Capítulo 3

Implementação

3.1 Introdução

Neste capítulo são apresentadas as diferentes fases do estágio curricular, com especial foco nas tarefas desenvolvidas e na respetiva implementação técnica de cada componente do projeto. O objetivo será, como já foi referido anteriormente, desenvolver uma solução para automatizar a procura e recolha de concursos públicos, assim como a extração de informação que ajudará o departamento de Vendas a tomar decisões. Este capítulo segue uma estrutura orientada pelas tarefas inicialmente definidas no plano de estágio, permitindo uma comparação entre o previsto e o efetivamente realizado. Embora tenham ocorrido alguns ajustes ao longo do percurso, essa estrutura serve como base para organizar e contextualizar os vários processos envolvidos no desenvolvimento da solução. Além da descrição das tarefas, são igualmente apresentadas as diferentes soluções adotadas, os principais desafios encontrados durante a execução do trabalho e os resultados alcançados em cada etapa. Esta abordagem visa oferecer uma visão abrangente e fundamentada da evolução do projeto, bem como do contributo efetivo do estágio para a resolução de um problema real no contexto da empresa.

3.2 Tarefa 1: Formação Sobre Tecnologias de RPA e *Web Scraping*

A primeira tarefa efetuada no decorrer do estágio foi de formação em *Robot Framework* com *Selenium*. O objetivo desta formação foi a preparação para a automatização de tarefas em páginas *web*, nomeadamente a navegação, extração de dados e manipulação de elementos *web*, desenvolvendo assim competências práticas nas ferramentas que seriam a base da solução final.

A escolha do *Robot Framework* como tecnologia base para a automação das tarefas *web* no projeto surgiu do interesse estratégico da empresa em explorar soluções de RPA. Esta tecnologia destacou-se por ser uma ferramenta *open-source*, com uma comunidade ativa, documentação extensa e suporte para diversas bibliotecas amplamente utilizadas nas áreas de automação e testes. Para além destes fatores, o *Robot Framework* revelou-se uma tecnologia versátil. A sua compatibilidade com vários sistemas operativos, nomeadamente *Windows*, *Linux* e *macOS*, facilita a integração em diversos ambientes de desenvolvimento sem a necessidade de alterações significativas.

Outro ponto de destaque é a sua abordagem baseada em palavras-chave (*keyword-driven*), que permite escrever testes e *scripts* de forma altamente legível, reutilizável e próxima da linguagem natural. Esta característica não só reduz a complexidade para quem desenvolve

os testes, como também favorece a comunicação com partes interessadas não técnicas, como gestores de projeto ou clientes, que podem compreender o funcionamento dos testes sem conhecimentos de programação.
Durante esta fase inicial, a equipa de trabalho atribuída ao projeto teve um papel pioneiro na adoção do *Robot Framework*, sendo responsável pelos primeiros desenvolvimentos com esta tecnologia na empresa, bem como pela introdução do conceito de RPA no contexto organizacional. Este envolvimento inicial traduziu-se num papel de maior responsabilidade na condução da formação e desenvolvimento do projeto, bem como numa posição estratégica na definição de boas práticas e na consolidação do uso da ferramenta na empresa.
Ao longo desta formação foram abordados diversos blocos temáticos que permitiram adquirir uma compreensão progressiva e aplicada das ferramentas. De seguida, apresentam-se os principais tópicos explorados.

3.2.1 Conceitos Introdutórios

A formação iniciou-se com os conceitos fundamentais do *Robot Framework*, passando também pela introdução a conceitos como automação, RPA e a sua importância no contexto do desenvolvimento de *software*. Estes conceitos foram explorados no capítulo do estado da arte (2), onde se apresenta uma análise mais detalhada do seu enquadramento teórico e das suas aplicações práticas.

3.2.2 Instalação e Configuração do Ambiente

Após a introdução aos conceitos fundamentais, a formação prosseguiu com a instalação e configuração do ambiente de desenvolvimento. A instalação do *Robot Framework* e das ferramentas relacionadas é um processo rápido, simples e que ocupa pouco espaço no sistema. Requer apenas a execução de alguns comandos básicos na linha de comandos e a transferência de alguns ficheiros.
O primeiro passo consistiu na instalação do *Python*, uma vez que o *Robot Framework* é construído sobre esta linguagem. Em conjunto com o *Python*, foi também instalado o gestor de pacotes do *python*, o *PIP*, utilizado para a instalação de bibliotecas adicionais.
Seguidamente, procedeu-se à instalação do *Robot Framework*, através do comando '`pip install robotframework`', o que permitiu começar a criar e executar *script* de testes e automação. Para permitir a interação com páginas *web*, foi igualmente instalada a biblioteca *SeleniumLibrary*, também através do *pip*, que adiciona ao *Robot Framework* a capacidade de controlar *browsers* via *Selenium WebDriver*.
De forma a garantir a compatibilidade com os navegadores pretendidos, foi necessário instalar os *drivers* específicos correspondentes, como o *ChromeDriver* (para o *Google Chrome*) ou o *GeckoDriver* (para o *Mozilla Firefox*). Estes *drivers* são responsáveis por fazer a ponte entre o *Selenium* e o navegador, possibilitando a execução dos comandos de automação no ambiente real do *browser*.
Para facilitar a escrita e organização dos testes, foi instalado o ambiente de desenvolvimento integrado *PyCharm*, amplamente utilizado em projetos *python*. A configuração do *PyCharm*

incluiu a instalação de extensões como o *Robot Framework Language Server*, que oferece suporte à sintaxe específica do *Robot Framework*, realce de código, sugestões automáticas e outras funcionalidades úteis para o desenvolvimento mais eficiente.
Por fim e para uma melhor organização, foi criada uma diretoria base para guardar os projetos.

3.2.3 Estrutura de um Projeto *Robot Framework*

No seguimento da configuração do ambiente, a formação abordou a estrutura típica de um projeto *Robot Framework*, com foco na organização de diretorias e na separação lógica de responsabilidades. Um projeto bem estruturado em *Robot Framework* é geralmente dividido em três diretorias principais:

- '**Tests/**': diretoria que contém as *suites* de teste, organizadas em ficheiros '`.robot`', onde se encontram definidos os cenários de teste que irão executar ações sobre a aplicação ou sistema-alvo.
- '**Resources/**': diretoria que reúne os recursos auxiliares, como ficheiros '`.robot`' com *keywords* personalizadas (palavras-chave reutilizáveis), variáveis, ou configurações partilhadas, promovendo modularidade e reutilização de código.
- '**Results/**': diretoria que armazena os ficheiros resultantes da execução dos testes, incluindo ficheiros como '`log.html`', '`output.xml`' e '`report.html`', que facilitam a análise posterior dos testes realizados.

Com base na imagem 3.1, pode delinear-se uma arquitetura geral e bem organizada de um projeto no *Robot Framework*, que segue o princípio da separação entre os testes e as *keywords*.

3.2.4 Secções de um Ficheiro `.robot`

A próxima etapa da formação focou-se no entendimento da estrutura de um *script* `.robot`. O *Robot Framework* organiza os seus *scripts* em secções distintas, cada uma com um propósito específico. Esta estrutura clara e modular facilita a leitura, manutenção e reutilização do código. Cada secção é iniciada com três asteriscos antes e depois do nome da secção, com um espaço entre eles, tal como mostra o *template* 3.1.

```
*** [Nome da Secção] ***
```

Excerto de Código 3.1: *Template* do divisor de secções de um *script* em *Robot Framework*

Abaixo, são descritas as principais secções que podem ser encontradas num ficheiro *.robot*.

- ***Settings:*** Esta secção, ilustrada no excerto de código 3.2, é utilizada para definir configurações globais do ficheiro. Aqui pode-se importar bibliotecas, recursos e variáveis externas, definir pré e pós-condições através de *Setup* e *Teardown* e adicionar uma referência à documentação do ficheiro para descrever o objetivo e contexto do conjunto de testes.

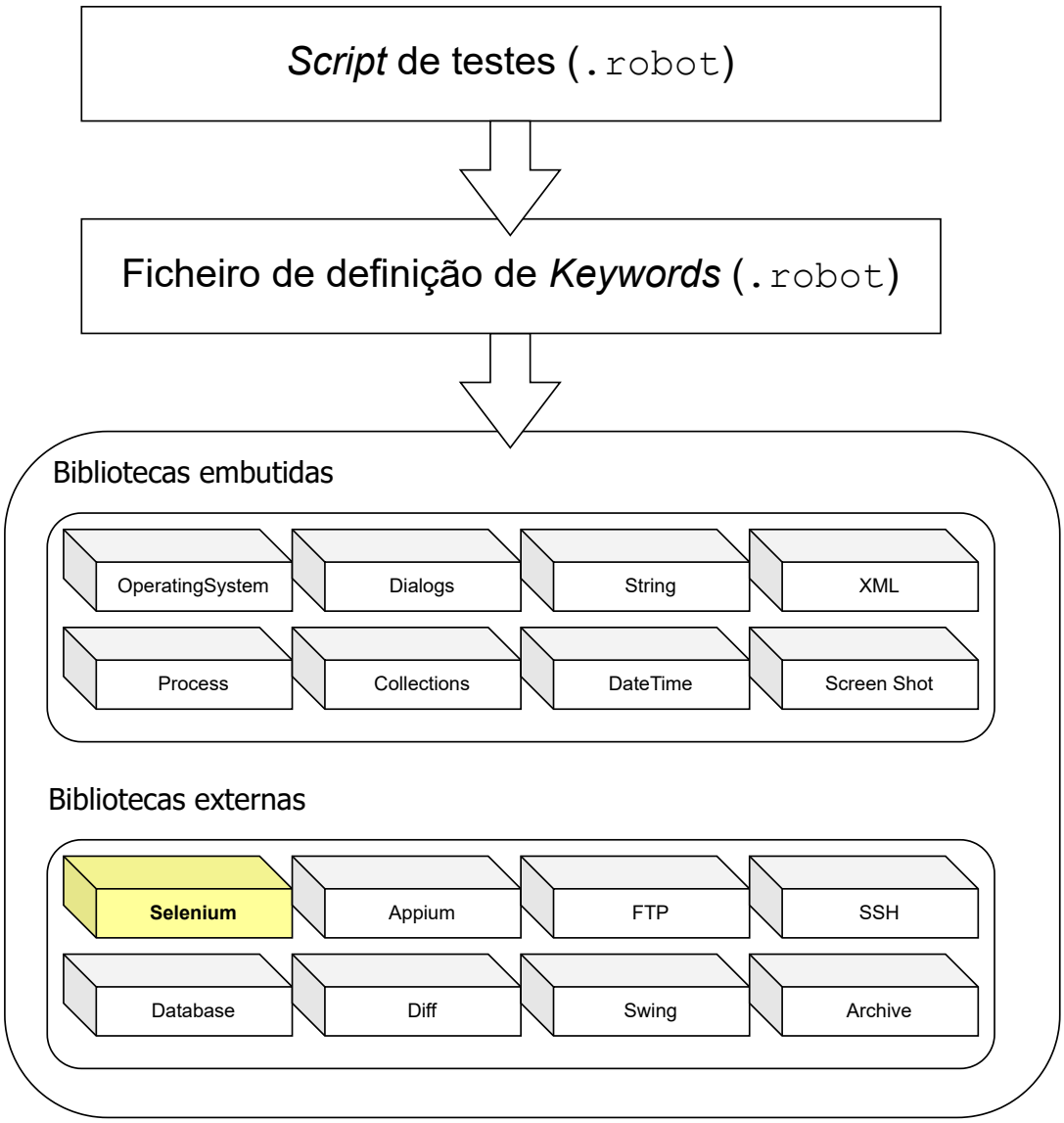

Figura 3.1: Estrutura lógica de um projeto *Robot Framework*.

```
*** Settings ***
Documentation     Este ficheiro contém testes de login e navegação.
Library           SeleniumLibrary
Resource          Keywords.robot
Suite Setup       Prepare Environment
Suite Teardown    Clean Environment
```

Excerto de Código 3.2: Exemplo de uma secção *Settings*

- ***Variables:*** Esta secção, presente no excerto de código 3.3, é utilizada para definir variáveis, que podem ser escalares, listas e dicionários.

```
*** Variables ***
${BASE_URL}           https://example.com
```

```
@{USER_CREDENTIALS}   admin   password123
&{CONFIG}             browser=Chrome   timeout=10
```

Excerto de Código 3.3: Exemplo de uma secção *Variables*

- ***Test Cases:*** Esta secção, demonstrada no excerto de código 3.4, contém os testes propriamente ditos. Cada caso de teste é definido como uma sequência de *keywords*, organizadas para validar funcionalidades ou comportamentos do sistema. No nosso caso, o objetivo não é propriamente testar o sistema mas acaba por ser necessário interagir com ele, da mesma forma que se procede para efeitos de teste.

```
*** Test Cases ***
Should be able to Login With Valid Credentials
    Open Browser To            ${BASE_URL}
    Input Text                 username     admin
    Input Text                 password     password123
    Click Button               login
    Page Should Contain        Welcome      browser=Chrome   timeout=10
```

Excerto de Código 3.4: Exemplo de uma secção *Test Cases*

- ***Keywords:*** Nesta secção, como demonstra o excerto de código 3.5, são criadas as palavra-chave personalizadas. Estas *keywords* ajudam a simplificar os testes, tornando-os mais legíveis e fáceis de manter. As *keywords* podem ser comparadas a funções em muitas linguagens de programação: encapsulam uma lógica específica e podem ser reutilizadas em diferentes partes do projeto.

```
*** Keywords ***
Open Browser To ${BASE_URL}
    Open Browser     ${BASE_URL}     ${BROWSER}
    Maximize Browser Window
```

Excerto de Código 3.5: Exemplo de uma secção *Keywords*

3.2.5 Opções de Execução e Ficheiros de Saída

No decorrer da formação foi também apreendida a forma de executar *scripts* no *Robot Framework*, capacitando os formandos a adaptar a execução dos testes a diferentes cenários e necessidades. O comando básico para executar uma suíte de testes, ou seja, um único *script* '`.robot`', é "`robot -d results tests/suite.robot`". Neste comando, a flag "`-d`" indica a diretoria onde devem ser armazenados os ficheiros resultantes da execução, como *logs* e ficheiros relacionados ao *debugging*, enquanto o caminho "`tests/suite.robot`" especifica o ficheiro `.robot` a ser executado. Além disso, foi demonstrado como executar apenas um caso de teste específico dentro de um *script*, utilizando a flag "`--include [nome da tag]`", que permite identificar e executar casos de teste marcados com uma determinada *tag*. Também foi abordada a possibilidade de executar múltiplas suítes de teste, ou seja, vários *scripts* '`.robot`', indicando apenas a diretoria onde estes se encontram, o que é particularmente útil para executar um conjunto completo de testes. Adicionalmente, foram apresentadas formas

de agendar a execução de *scripts*, utilizando ferramentas como o *cron* em sistemas *Linux* ou o Agendador de Tarefas em sistemas *Windows*, permitindo a execução periódica e automatizada dos testes. Após a execução são automaticamente gerados vários ficheiros de saída que são fundamentais para a análise dos testes realizados. Entre os ficheiros gerados, destaca-se o '`log.html`', que contém um registo detalhado da execução dos testes, incluindo informações sobre cada passo, as palavras-chave chamadas, os argumentos utilizados, os resultados das operações e eventuais mensagens de erro. Outro ficheiro importante é o '`report.html`', que apresenta um resumo visual e estruturado dos testes executados, permitindo uma visão geral rápida do estado dos testes. Por fim, o ficheiro '`output.xml`' armazena os dados brutos da execução dos testes em formato *Extensible Markup Language* (XML), sendo utilizado principalmente para integração com outras ferramentas. Esses ficheiros são de extrema importância, pois auxiliam na identificação de erros e na avaliação do desempenho dos testes, garantindo que o processo de automação seja eficiente e confiável.

3.2.6 *Locators*

Durante a formação, foi dado um destaque significativo ao entendimento de *locators*, dada a sua importância para a interação com elementos em páginas *web*. Os *locators* são fundamentais para identificar e manipular componentes como campos, botões, *links* e outros elementos de uma página durante a execução dos testes automatizados. A sua compreensão e o uso correto são essenciais para garantir a precisão e a confiabilidade dos testes. Foram apresentados os principais tipos de *locators* utilizados no *Robot Framework*, incluindo *id* e *name* de elementos HTML, *links*, expressões *XPath* e seletores *Cascading Style Sheets* (CSS). Cada um desses *locators* possui características específicas e é adequado para diferentes cenários. Por exemplo, o *id* e o *name* são frequentemente utilizados por serem atributos únicos e fáceis de identificar, enquanto o *XPath* e os seletores CSS são mais versáteis e permitem localizar elementos com base em estruturas mais complexas ou em combinações de atributos. A escolha do *locator* mais adequado depende do contexto em que será utilizado. O primeiro passo para o identificar corretamente é abrir a ferramenta "Inspecionar Elemento", disponível na maioria dos navegadores, e analisar os atributos do elemento que se pretende referenciar. Uma boa prática é testar o *locator* diretamente na ferramenta de inspeção, utilizando a funcionalidade de pesquisa para verificar se ele retorna apenas o(s) elemento(s) desejado(s). Em páginas *web* mais complexas, pode ser desafiador referenciar determinados elementos apenas com atributos simples, como *id* ou *name*. Nesses casos, é necessário criar expressões *XPath* ou seletores CSS personalizados para identificar os elementos corretamente. A ferramenta de inspeção do navegador facilita este processo, permitindo que se clique com o botão direito no elemento desejado e se selecione a opção de copiar estes atributos. No entanto, as opções automáticas fornecidas pela ferramenta nem sempre geram caminhos eficientes ou suficientemente precisos. Por isso, é importante saber construir manualmente essas expressões para lidar com casos mais desafiadores e garantir a confiabilidade dos testes. Essa abordagem prática e detalhada foi amplamente explorada durante a formação, capacitando os participantes a superar este tipo de desafios.

3.2.7 Exploração da Biblioteca *Selenium* e Projetos de Prática

Durante a formação, foram desenvolvidos projetos práticos que consistiam na realização de *scraping* em diferentes *websites*. Estes projetos foram estruturados de forma progressiva, começando com *sites* mais simples e aumentando gradualmente a complexidade à medida que os formandos adquiriam mais conhecimentos e experiência. Essa abordagem prática foi essencial para consolidar os conceitos aprendidos e para explorar as funcionalidades da biblioteca *Selenium*, uma das ferramentas centrais utilizadas no *Robot Framework*. Foi no contexto do desenvolvimento desses projetos que se exploraram as principais funções dessa biblioteca, que foi utilizada para realizar tarefas como localizar elementos, preencher formulários, clicar em botões e navegar entre páginas, entre outras ações fundamentais para a automação de testes e *scraping*. Essa prática proporcionou uma visão abrangente das capacidades do *Selenium* e da sua integração com o Robot Framework, preparando os participantes para lidar com cenários reais e desafios mais complexos.

3.2.8 Conclusão

A formação abordou uma ampla gama de temas relacionados à utilização do *Robot Framework* em conjunto com a biblioteca *Selenium*. Para além dos tópicos já apresentados, no período de aprendizagem foram também explorados os diferentes tipos de variáveis disponíveis, as estruturas de controlo e um conjunto de boas práticas. Outros pontos importantes foram o conhecimento sobre *Suite Setup* e *Teardown*, que permitem a execução de configurações iniciais e tarefas de limpeza antes e após os testes, a criação de bibliotecas personalizadas em *Python*, o que possibilita a extensão das funcionalidades do *Robot Framework* para atender a cenários específicos, e a análise complexa de *logs* gerados pela *framework*, que são essenciais para identificar problemas e otimizar os testes. A formação foi, assim, um ponto crucial para o estágio, tanto por representar o marco inicial da integração na empresa como por constituir a base técnica para o desenvolvimento do projeto subsequente. Destaca-se ainda a qualidade da formação e a abrangência dos temas abordados, que proporcionaram uma compreensão sólida e prática do uso desta ferramenta.

3.3 Tarefa 2: Levantamento de Requisitos e Desenho da Arquitetura

Apesar de se tratar de um projeto com elevado potencial para se tornar uma ferramenta extremamente útil no contexto da empresa, o seu caráter interno, aliado ao facto de estar integrado num programa de estágio, resultou num planeamento inicial relativamente simples. Não estando dependente de requisitos ou prazos impostos por um cliente externo, o planeamento deste projeto consistiu essencialmente em duas reuniões principais, complementadas por reuniões diárias mais informais, onde se discutiam ajustes, problemas, melhorias e os próximos passos a seguir. Na primeira reunião, foram apresentados os objetivos gerais do projeto, a sua finalidade no contexto empresarial, uma previsão das ferramentas a serem utilizadas e o delineamento do plano de estágio. Já na segunda reunião, foi realizado o levantamento

de requisitos e o desenho preliminar da arquitetura da solução. É importante referir que, nesse planeamento inicial, não houve preocupação em elaborar diagramas de caso de uso ou em abordar aspetos relacionados com o *frontend*. Isto deveu-se ao facto de ainda não estar definido qual equipa seria responsável pelo desenvolvimento da interface de utilizador, sendo que o foco inicial foi direcionado para os componentes de *backend* e automação. Nesta secção, será partilhado o que foi delineado durante esta fase inicial de planeamento, com destaque para o levantamento de requisitos e o desenho da arquitetura, que serviram como base para o desenvolvimento do projeto.

3.3.1 Levantamento de Requisitos

O levantamento de requisitos deste projeto foi realizado com o objetivo de definir as funcionalidades essenciais e as características não funcionais da solução, garantindo que esta atendesse às necessidades da equipa de vendas e cumprisse o propósito de otimizar a análise de concursos públicos relevantes para a empresa. Estes requisitos refletem as expectativas e objetivos definidos durante a fase de planeamento e foram divididos em requisitos funcionais e requisitos não funcionais. Os **requisitos funcionais** descrevem as funcionalidades específicas que o programa deve implementar para cumprir o seu propósito. Estes incluem:

- abrir automaticamente um navegador e aceder aos *sites* de concursos públicos definidos;
- realizar o *login* utilizando as credenciais da empresa, garantindo acesso às áreas onde os concursos públicos estão disponíveis;
- procurar e transferir concursos públicos que satisfaçam as seguintes condições:
  - o prazo para entrega de propostas deve estar compreendido num intervalo de dois meses, começando na data de execução do *script*;
  - o título ou descrição do concurso deve conter pelo menos uma das seguintes palavras-chave: “plataforma”, “desenvolvimento”, “web”, “mobile”, “movel”, “website”, “aplicacao”, “manutencao”, “digital”, “informatico”, “outsystems”, “cloud”, “low code”, “tecnologico” ou “software”;
- o programa deve ser capaz de processar automaticamente os documentos associados aos concursos públicos transferidos, utilizando LLM para extrair as seguintes informações:
  - preço base do concurso;
  - preço anormalmente baixo do concurso;
  - critério de adjudicação das propostas;
  - documentos necessários para a constituição da proposta;
  - local de execução do contrato;
  - prazo de execução do contrato;

  - especificações funcionais do concurso;
  - especificações técnicas do concurso;
  - requisitos da equipa que irá desempenhar os serviços contratados;
- as informações extraídas devem ser armazenadas numa base de dados de forma organizada e estruturada, para estarem prontas a ser utilizadas e exibidas numa interface gráfica;
- o programa deve ser executado automaticamente todos os dias às 20h, iniciando a sequência de operações necessárias para cumprir os requisitos acima.

Os **requisitos não funcionais** definem as características e as restrições técnicas que o programa deve respeitar. Estes incluem:

- o programa deve funcionar de forma completamente autónoma após a sua configuração inicial, minimizando a necessidade de intervenção manual;
- não há nenhuma restrição significativa relativamente ao tempo de execução do programa uma vez que a execução ocorrerá durante a noite e é apenas necessário garantir que toda a análise seja concluída antes do início do horário laboral, o que resulta num intervalo de tempo considerável tendo em conta a natureza do programa;
- o armazenamento das credenciais utilizadas para aceder aos *sites* de concursos públicos deve ser feito de uma maneira segura;
- a solução deve ser implementada num *container* que inclua todo o ambiente necessário para a sua execução, conferindo portabilidade ao projeto;
- a solução deve ser executada num servidor interno da empresa, garantindo o controlo e a privacidade dos dados processados;
- a base de dados utilizada deve ser uma base de dados *PostgreSQL*, uma vez que esta já está instalada no servidor interno da empresa e é a tecnologia padrão utilizada para armazenamento de dados.

3.3.2 Desenho da Arquitetura

O desenho da arquitetura do projeto foi orientada pelos requisitos levantados, adotando uma abordagem modular para garantir a separação de responsabilidades e a escalabilidade do sistema. A arquitetura pode ser representada em camadas, com cada um dos cinco principais componentes desempenhando um papel específico e interagindo com os demais de forma integrada, tal como se pode ver na figura 3.2. A base de dados é o componente central do projeto, uma vez que interage diretamente com a maioria dos outros componentes. Nela são armazenados tanto os dados recolhidos pelo *scraper* quanto as informações recolhidas pelo componente de análise e extração de informação. Além disso também fornece os dados que o *frontend* exibe para o utilizador. Como engrenagem que faz todo o sistema "trabalhar",

temos o componente do *scraper*, responsável por recolher os concursos públicos e adicionar à base de dados as entradas que identificam cada um deles. O componente de análise e extração de informação é responsável por processar os documentos recolhidos, extrair os dados relevantes e armazená-los na base de dados, associando-os aos respetivos concursos. A interagir com os dois últimos componentes, temos o *scheduler*, responsável por despoletar a execução desses componentes diariamente às 20 horas. A sua única função será atuar como o "gatilho" que inicia o fluxo de trabalho, sendo uma peça fundamental para garantir a existência de dados atualizados de forma contínua e sem intervenção manual. Por fim existe o componente do *frontend*, que funciona como a camada de apresentação, onde os dados processados são exibidos de uma forma clara e intuitiva para o utilizador final.

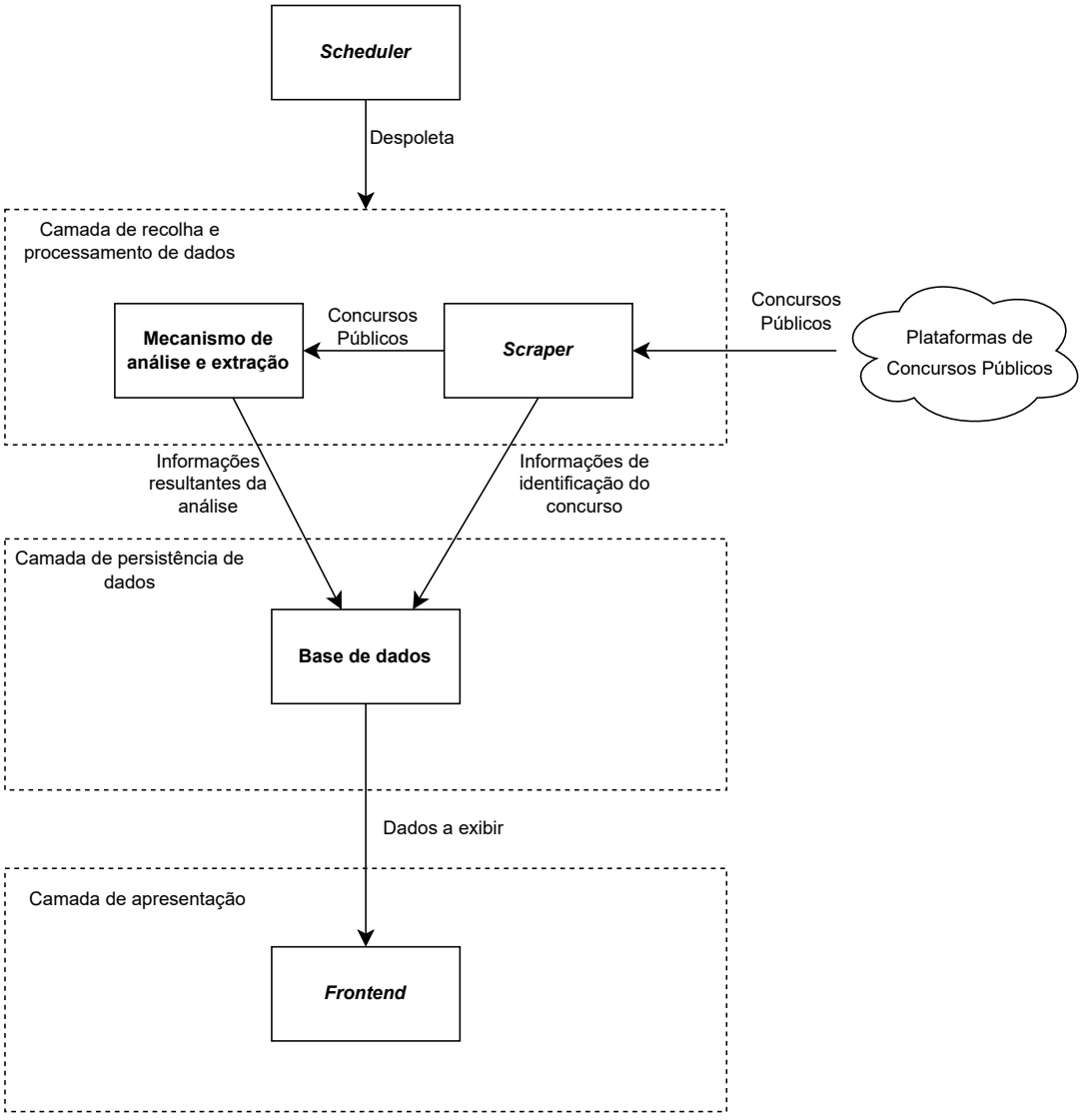

Figura 3.2: Arquitetura de *software* do projeto.

3.4 Tarefa 3: Implementação da Automatização do Processo de Procura e Transferência de Concursos Públicos de Interesse

O objetivo desta tarefa foi desenvolver um *script* automatizado para realizar a procura e transferência de concursos públicos relevantes, disponíveis em repositórios na *web*. Utilizando as tecnologias estudadas durante o período de formação do estágio, esta solução permite identificar e descarregar documentos de concursos que se alinhem com o contexto de atuação da empresa, otimizando o processo anteriormente realizado de forma manual. No contexto deste projeto, a empresa identifica oportunidades de negócio em duas plataformas principais de concursos públicos. Por razões de confidencialidade, os nomes reais dessas plataformas não serão divulgados, sendo referidas ao longo deste relatório como plataforma A e plataforma B. Inicialmente, após a conclusão das formações, foi desenvolvido um *script* de *scraping* para a plataforma A, permitindo a automatização da extração de concursos relevantes dessa fonte. Em paralelo, deu-se início ao desenvolvimento do *scraper* para a plataforma B, mas a sua continuidade acabou por ser suspensa devido a uma limitação técnica que será detalhada mais adiante neste capítulo. Com o *scraper* da plataforma A concluído, foi possível avançar para o desenvolvimento dos restantes componentes do projeto. A ausência do *scraper* para a plataforma B não comprometeu o funcionamento geral do sistema, garantindo que os objetivos do estágio continuassem a ser cumpridos.
Antes de iniciar o desenvolvimento do primeiro *script*, foi necessário criar a estrutura do projeto de acordo com as diretrizes definidas durante a formação. Com esse propósito, foram criadas as principais diretorias do projeto:

- uma destinada aos *scripts* que contêm os casos de teste, ou seja, os passos que compõem cada uma das fases do processo de *scraping*;
- uma para armazenar os recursos utilizados pelos *script*, como os ficheiros que contêm as *keywords* definidas pelo utilizador;
- uma dedicada ao armazenamento dos ficheiros de *log* resultantes da execução dos *script*;
- uma para guardar as bibliotecas personalizadas.

Para garantir a consistência do ambiente de desenvolvimento entre os membros da equipa e evitar problemas de compatibilidade, foi necessário criar um ambiente isolado para a instalação dos pacotes e dependências do projeto. Para isso, utilizou-se o *Conda*, uma ferramenta de gestão de pacotes e ambientes virtuais. Foi criado um ambiente denominado `rpa_env`, configurado a partir do ficheiro `environment.yml`, que especificava as dependências necessárias e os repositórios de onde estas deveriam ser obtidas. Esta abordagem permitiu que todos os membros da equipa pudessem replicar o ambiente de forma simples e eficiente, eliminando a necessidade de configurações manuais em diferentes máquinas.
É importante esclarecer que embora a *containerização* da aplicação fosse um requisito do projeto, optou-se inicialmente por não se utilizar o *Docker* durante as fases iniciais de desenvolvimento. Esta decisão foi motivada pela necessidade de testar e visualizar a execução

dos *scripts*, algo que o *Conda* permitia, uma vez que possibilitava observar diretamente a interação do programa com a interface gráfica do navegador. No *Docker*, por outro lado, a ausência de uma interface gráfica dificultava este processo. Como se verificaria posteriormente, esta decisão teve implicações negativas. O adiamento da *containerização* para fases mais avançadas do projeto acabou por dificultar o processo de empacotamento e execução da aplicação dentro do *container*. A tentativa de realizar a *containerização* de toda a solução numa única fase revelou-se complexa e propensa a erros, especialmente devido ao número de dependências envolvidas. Assim, concluiu-se que a integração progressiva da solução em *Docker*, de forma incremental, desde as fases iniciais e paralelamente ao desenvolvimento no ambiente *Conda*, teria sido uma abordagem mais eficaz e sustentável.
Com a estrutura do projeto definida e o ambiente devidamente configurado, foram implementados os passos necessários para automatizar a obtenção de concursos públicos de interesse na plataforma A. Abaixo, descrevem-se as etapas desenvolvidas:

1. **Inicialização do navegador e configuração da diretoria de transferências**: o primeiro passo do processo consiste em abrir o navegador e configurar a diretoria onde serão guardados os ficheiros transferidos durante a execução do *script*. Na plataforma, o conjunto de documentos que compõem cada concurso público pode ser descarregado como um ficheiro *zip*. Neste passo, o programa descarrega esse ficheiro e guarda-o localmente numa diretoria acessível ao restante projeto. Para este fim, foi criada a pasta `proposals`;

2. **Acesso à página de *login* e autenticação:** o programa procede com o acesso à página de *login* da plataforma, onde são introduzidos o *email* e a palavra-passe nos respetivos campos de entrada, seguidos de um clique no botão "Entrar". Estes dados são previamente armazenados num ficheiro em formato JSON, o qual é lido para memória no início da execução e cujos valores são atribuídos a variáveis que podem ser reutilizadas ao longo do *script*, assegurando maior segurança e flexibilidade na gestão das credenciais. Uma boa prática que foi implementada em todo o *script* foi a verificação do carregamento completo das páginas antes de prosseguir com os passos subsequentes. Por exemplo, quando o programa navega até à página de *login*, ele aguarda até que a string "Entrar" esteja visível na interface (com um tempo limite de 5 segundos). Esta prática evita que instruções sejam executadas prematuramente, o que resultaria em erros que poderiam interromper a execução do programa. Para ilustrar a simplicidade e a intuitividade da linguagem utilizada no *Robot Framework*, o excerto de código 3.6 demonstra a correspondência com a *keyword* que implementa esta funcionalidade.

```
Login
    Go To    ${URL.login}
    wait until page contains    Entrar    5s
    Input Text        xpath://input[@id='user']    ${EMAIL}
    Input Password    xpath://input[@id='pass']    ${PASSWORD}
    Click Button      xpath://button[contains(.,'Entrar')]
```

Excerto de Código 3.6: Keyword responsável por realizar o *login* na plataforma A.

3. **Fecho de *pop-ups* e aplicação de filtros de pesquisa:** Existe um *pop-up* que aparece sempre após o *login*, que serve para avisar o utilizador de possíveis informações em falta na sua conta. Antes de prosseguir é necessário fechá-lo. Desse modo, após esperar que a página principal seja totalmente carregada, o programa espera que o *pop-up* em questão fique visível e clica no botão de fechar.

4. **Filtragem de concursos com base em palavras-chave:** Após o carregamento da página principal e o fecho do *pop-up*, o *script* prossegue com a identificação dos concursos públicos relevantes. A plataforma A apresenta inicialmente uma lista extensa de concursos, ordenada por data de publicação, do mais recente para o mais antigo. No entanto, como apenas uma fração destes concursos se enquadra no âmbito de interesse da empresa, tornou-se necessário aplicar uma estratégia eficaz de filtragem, de modo a reduzir significativamente o volume de dados a analisar.

   A solução adotada consistiu em explorar o mecanismo de pesquisa interna da própria plataforma. Esta funcionalidade permite a inserção de uma *string* de texto numa caixa de pesquisa localizada acima da listagem, resultando na exibição de concursos cujo título ou descrição contenham essa *string*. Esta abordagem revelou-se não apenas eficaz, como também eficiente, uma vez que utiliza recursos já disponíveis na interface da plataforma e evita a necessidade de um pós-processamento complexo.

   A seleção das palavras-chave a utilizar neste campo foi realizada com o apoio da equipa de vendas, que, por estar em contacto diário com a plataforma, possui conhecimento aprofundado sobre os critérios mais relevantes para a filtragem de concursos. Estas palavras-chave, previamente discutidas com a equipa de desenvolvimento, encontram-se documentadas na secção 3.3.

   Para garantir a flexibilidade e manutenibilidade da aplicação, foi criado um novo ficheiro de configuração, `config.json`, com o objetivo de armazenar todas as configurações globais do projeto. Neste ficheiro foram incluídas as palavras-chave a aplicar na filtragem, permitindo que estas sejam facilmente carregadas em tempo de execução. Assim, o *script* percorre programaticamente a lista de filtros e executa uma pesquisa individual para cada um deles, dando início à extração dos concursos correspondentes. Para a manipulação de ficheiros no formato JSON, foi usada uma biblioteca do *Robot Framework*, *JSONLibrary*.

5. **Transferência e organização dos concursos públicos:** Com a sublista de concursos já filtrada, o passo seguinte consiste na transferência dos documentos e na extração preliminar das informações essenciais. Esta fase foi estruturada em diferentes etapas:

   (a) **Identificação dos concursos e datas-limite:** Numa primeira fase, o *script* obtém todos os elementos correspondentes aos concursos da sublista através de um *XPath* comum. Simultaneamente, são também extraídas as datas-limite para apresentação de propostas de cada concurso. Com estas duas listas construídas, inicia-se o processo de transferência dos documentos, respeitando as restrições definidas nos requisitos funcionais.

(b) **Gestão dinâmica de *pop-ups*:** Durante os estágios iniciais de desenvolvimento, observou-se que o *pop-up* que é exibido aquando do primeiro acesso à página principal também poderia surgir de forma aleatória após um certo intervalo de tempo, interrompendo a execução normal do *script*. Para mitigar este problema, antes de cada interação com um concurso, é sempre verificada a presença do *pop-up*. Caso esteja presente, este é fechado; caso contrário, o *script* prossegue normalmente. Esta simples verificação mostrou-se suficiente para impedir que o *pop-up* interferisse com o processo.

(c) **Validação do prazo de entrega:** De acordo com os requisitos funcionais do projeto, apenas os concursos cujo prazo de entrega estivesse compreendido num intervalo de até dois meses seriam considerados. Assim, para cada concurso é verificado se a data limite para apresentação de propostas é inferior à data atual acrescida de sessenta dias. Caso contrário, o concurso é ignorado e o *script* avança para o seguinte.

(d) **Cenários possíveis após seleção do concurso:** Quando um concurso cumpre os requisitos de prazo, o *script* clica no elemento correspondente para abrir a página do concurso. A partir daqui, três cenários são possíveis:

- O utilizador tinha previamente registado interesse no concurso em questão, pelo que é redirecionado para a página do concurso, onde estão disponíveis informações mais detalhadas e todos os documentos para transferência;
- O utilizador não tinha registado interesse no concurso em questão, pelo que é exibido um modal que permite ao utilizador registar-se como interessado ou transferir diretamente os documentos;
- O concurso em questão está encerrado ou suspenso, pelo que se é redirecionado para uma página que indica falta de permissões para aceder ao concurso, que é então ignorado pelo script.

Nos dois primeiros cenários, o *script* procede à transferência do ficheiro comprimido (*.zip*) contendo todos os documentos do concurso. Além disso, são extraídas e guardadas na base de dados informações preliminares, como o ID do concurso na plataforma, o título e uma breve descrição. No caso de concursos em que o utilizador registou interesse, é também possível obter a data-limite para esclarecimento de dúvidas, uma informação adicional relevante para o projeto.

(e) **Organização dos ficheiros transferidos:** Após o início de cada transferência, é criada uma nova diretoria na pasta `proposals`, identificada por um nome composto pelo ID interno do programa (na base de dados) e pelo ID do concurso na plataforma, seguindo o formato <id_base_de_dados>_<id_plataforma>, por exemplo, 3_928245.

(f) **Processo de transferência e a sua validação:** Durante o desenvolvimento, foram implementadas algumas otimizações que mudaram significativamente o comportamento e eficiência do *script*. Inicialmente, cada transferência tinha um tempo limite definido. Se o ficheiro fosse transferido dentro do prazo, era movido para a pasta criada e o estado *success* era registado na base de dados na

coluna relativa à conclusão da transferência. Caso contrário, a pasta era eliminada e o estado *failed* era atribuído à entrada relativa ao concurso. Foi observado que, ocasionalmente, o *download* terminava exatamente no momento em que o tempo limite expirava, resultando na eliminação prematura da pasta e numa estrutura desorganizada de ficheiros. Para resolver este problema e melhorar o desempenho, alterou-se a lógica de verificação dos *downloads*. Em vez de aguardar a conclusão de cada transferência antes de avançar, o *script* passou a iniciar os *downloads* de todos os concursos e só no final realizava a verificação global. Concretamente, para cada concurso, inicia-se o *download*, cria-se a pasta correspondente, e prossegue-se imediatamente para o próximo concurso. No final de todas as transferências, o *script* aguarda dez segundos para que os últimos *downloads* a ocorrer sejam concluídos e, em seguida, organiza os ficheiros. Como os ficheiros transferidos da plataforma A têm todos, no seu nome, o seu id da plataforma, é possível comparar com o nome das diretorias criadas e movê-los para a pasta correspondente. Apenas os ficheiros corretamente organizados são marcados como *success* na base de dados, enquanto que os ficheiros que estão na *root* da pasta `proposals` (ficheiros temporários de *downloads* inacabados criados pelo *browser* e transferências falhadas) são eliminados. Desta forma, este processo torna-se mais eficiente, reduzindo o tempo total de execução e evitando inconsistências na organização dos ficheiros transferidos.

A conclusão desta fase do projeto permitiu estabelecer uma base sólida para o desenvolvimento da tarefa seguinte, a extração de informação dos documentos dos concursos públicos. Após a implementação do processo de recolha e organização dos dados, foi possível garantir:

1. Uma fonte estruturada de informação, composta pelos documentos de cada concurso público relevante, devidamente armazenados em diretorias identificadas de forma única.
2. Uma estrutura de dados persistente, suportada por uma base de dados na qual cada concurso é representado por uma entrada própria, à qual podem ser posteriormente associadas as informações extraídas.
3. Um processo automatizado e configurável, através de um *script* robusto, parametrizável por meio de uma lista de filtros de pesquisa, que pode ser executado a qualquer momento para recolher novos concursos disponíveis na Plataforma A.

Deste modo, encontram-se reunidas todas as condições para avançar para a etapa seguinte do projeto: a extração e processamento automático das informações contidas nos documentos obtidos.

3.5 Tarefa 4: Implementação da Extração Automática de Informação dos Documentos que Constituem os Concursos

O objetivo principal desta tarefa consiste no desenvolvimento de uma solução capaz de processar automaticamente os documentos dos concursos públicos, com vista à extração das

informações mais relevantes para a análise por parte da equipa de vendas. A intenção é permitir uma avaliação rápida e objetiva acerca do interesse e viabilidade da participação da empresa em cada concurso, bem como das condições contratuais impostas pela entidade adjudicante.
Na tarefa anterior, foi automatizado o processo de recolha dos concursos públicos relevantes, eliminando a necessidade de a equipa de vendas aceder manualmente às plataformas, procurar os concursos de interesse e proceder à respetiva transferência dos documentos. Esta automatização, por si só, já representa uma significativa mais-valia, reduzindo o esforço manual e o risco de omissão de oportunidades.
Contudo, os documentos associados aos concursos públicos são frequentemente extensos e apresentam informações distribuídas por diversas secções, o que exige, até ao momento, uma leitura atenta e demorada por parte dos colaboradores. O desafio que se coloca nesta fase do projeto consiste, portanto, em desenvolver uma solução que permita extrair, de forma automatizada, os seguintes elementos-chave:

- **Preço base da proposta:** valor máximo que a entidade adjudicante está disposta a pagar pelo serviço em questão;
- **Preço anormalmente baixo:** valor que está significativamente abaixo do preço base ou dos preços normalmente praticados no mercado para o serviço em questão;
- **Critérios de adjudicação:** parâmetros definidos pela entidade adjudicante para avaliar e classificar as propostas apresentadas;
- **Documentos da proposta:** lista de documentos obrigatórios que devem constituir a proposta ao concurso;
- **Prazo de execução:** período estipulado para a execução integral do contrato, contado a partir da adjudicação.

Para contextualizar a complexidade do processamento automático destas informações, importa apresentar a estrutura típica dos documentos que compõem um concurso público:

- **Caderno de Encargos:** documento oficial que define as condições contratuais e os requisitos técnicos a cumprir na execução do contrato. A sua extensão varia significativamente, podendo conter menos de 10 páginas ou ultrapassar as 50, consoante a complexidade do concurso. É normalmente estruturado em cláusulas ou artigos e abrange prazos, penalizações, deveres e direitos das partes envolvidas, entre outras informações essenciais.
- **Programa do Procedimento (ou Programa do Concurso):** documento complementar ao Caderno de Encargos, com foco nas regras do procedimento, incluindo prazos, critérios de avaliação, requisitos formais das propostas e aspetos administrativos. Tende a ser menos extenso que o Caderno de Encargos, mas, pela experiência adquirida no desenvolvimento desta solução, estima-se que contenha, em média, entre 20 e 30 páginas.

Embora estes dois documentos sejam as principais fontes de informação para a conceção da solução, podem existir ainda outros documentos, como anexos ou listas de artigos, que, em determinados casos, contêm informações relevantes para a análise.
Para esta tarefa, foram experimentadas várias abordagens distintas, cada uma explorando técnicas e ferramentas diferentes. Estas abordagens, bem como os respetivos resultados, serão detalhadas nas próximas subsecções, de forma a evidenciar o processo de tentativa e erro que levou à escolha das soluções mais adequadas.

3.5.1 Captação de Padrões no Texto do PDF Através de Expressões Regulares

A primeira abordagem para a extração automática de informação baseou-se na utilização de expressões regulares (*regex*). A ideia inicial surgiu da observação de que, embora o conteúdo e a estrutura dos documentos dos concursos públicos variassem significativamente de caso para caso, existiam determinados padrões linguísticos e estruturais que poderiam servir como âncoras para localizar os elementos de interesse.
Para operacionalizar esta abordagem, foi desenvolvido o *script* `pdf.robot`, responsável por carregar os ficheiros PDF e aplicar o processamento de texto necessário. A biblioteca `RPA.PDF` do *Robot Framework* foi utilizada para manipular os documentos, permitindo extrair o seu conteúdo para memória e realizar pesquisas textuais sobre o mesmo.
O processo iniciou-se com a implementação da extração do preço base, tratando um ficheiro de cada vez. Nesta fase inicial, o objetivo era simples: dado um ficheiro, retornar o valor do preço base caso estivesse presente no texto.
Para reduzir o espaço de pesquisa, foi implementado um mecanismo de filtragem preliminar. Por exemplo, para o caso do preço base, eram procuradas páginas contendo termos como "preço base", "euros" ou o símbolo "€", uma vez que estes estariam, com elevada probabilidade, associados ao valor pretendido.
Com o conjunto de páginas reduzido, iniciou-se o desenvolvimento de uma primeira expressão regular destinada a capturar os valores monetários presentes nesses trechos de texto:

```
[€]?([1-9]\d{0,2})([.\s]?\d{3})*,\d{2}[€]?
```

Esta expressão foi concebida com base nos seguintes critérios observados:

- O símbolo do euro poderia surgir no início ou no fim do número, ou até estar ausente;
- O primeiro grupo de dígitos teria entre um a três dígitos, mas nunca começando por zero;
- Os separadores das casas dos milhares poderiam ser pontos ou espaços;
- Os cêntimos eram sempre representados por vírgula seguida de exatamente dois dígitos.

Desta forma, a expressão conseguia capturar valores como:

- 123,45

- 1.234,56
- 1 234 567,89
- €1.234,56
- 1.234,56€

Com o tempo, e através de um processo iterativo de tentativa e erro, foram sendo desenvolvidas expressões regulares mais específicas para cada tipo de informação a extrair. As principais expressões finais obtidas foram as seguintes:

- Preço base:

```
preço base.*?[€]?\\s*\\d{1,3}([\\.\\s]?\\s*\\d{3})*\\s*,\\s*\\d{2}\\
s*[€]?\\s*\\((.*?)\\)
```

- Preço anormalmente baixo:

```
(preço anormalmente baixo){1}\\s*(\\d+)(.*?)(?=%|anormalmente baixo)
(.*?)(\\d+\\.)
```

- Critérios de Adjudicação:

```
proposta economicamente mais vantajosa(.*?)(?=\\d+\\.\\d+\\.|\\d+\\.
\\d)
```

- Documentos que constituem a proposta:

```
((?=constituída|constituidas|composta)?(?:,| obrigatoriamente)? pel
os seguintes documentos|acompanhar as suas propostas).*?:(.*?)(?=\\
d+\\.\\d+\\.|\\d+\\.\\d)
```

- Prazo de execução:

```
(?=local\\s|prazo\\s).*?serviços.*?(?=ª|º|\\d+\\.(\\d+)*\\.*\\s)>
```

Após esta fase de desenvolvimento, as expressões foram testadas num conjunto de 30 propostas. Os resultados obtidos encontram-se resumidos na tabela 3.1. As estatísticas presentes nesta tabela e nas próximas tabelas de precisão de uma abordagem resultaram de uma observação individual das respostas do modelo e da comparação com o conteúdo do relatório. Para isso, foram registadas todas as respostas corretas e todas as respostas que estavam incompletas e que podiam induzir o utilizador em erro.

Tabela 3.1: Resultados dos testes de precisão da abordagem da REGEX para um conjunto de 30 concursos.

Campo	Taxa de acerto	Taxa de acerto incompleto
Preço base	73,32%	0,00%
Preço anormalmente baixo	13,32%	0,00%
Critérios de Adjudicação	53,32%	16,67%
Documentos da Proposta	26,67%	23,32%
Prazo de Execução	0,00%	0,00%

Os resultados foram, no geral, insatisfatórios. Embora o preço base tenha apresentado uma taxa de sucesso razoável, os restantes campos registaram uma performance muito aquém do desejado, sendo particularmente notório o insucesso total na extração do prazo de execução. A principal limitação desta abordagem deveu-se à enorme diversidade de formatos e redações presentes nos documentos. Mesmo expressões regulares cuidadosamente construídas funcionavam bem apenas para um subconjunto restrito de casos, falhando noutros. Além disso, enquanto para os valores monetários era possível identificar com relativa precisão o valor correto a extrair, para outros campos, como os critérios de adjudicação, o desafio era consideravelmente maior, uma vez que estes podiam ocupar grandes blocos de texto, por vezes estendendo-se por páginas inteiras, dificultando a sua delimitação através de expressões regulares. Assim, concluiu-se que, para este contexto e nível de heterogeneidade dos dados, seria impraticável obter expressões regulares suficientemente robustas para resolver o problema de forma satisfatória.

3.5.2 Extração da Informação Através de *Prompting* do Gemini

A experiência descrita na subsecção anterior evidenciou a necessidade de uma solução capaz de compreender o contexto semântico dos documentos, algo que as expressões regulares, pela sua própria natureza, não conseguiam alcançar. Estando o estágio a realizar-se numa época de grande popularidade e disponibilidade de LLMs, tornou-se evidente que esta tecnologia poderia constituir a abordagem mais promissora para o problema em questão.

Para uma primeira experiência, optou-se pela utilização do modelo *Gemini 1.5 Flash*, disponibilizado pela *Google*, uma vez que, à data, este se encontrava entre os modelos mais avançados e disponibilizava uma API de utilização gratuita, com limites de utilização adequados para a fase experimental do projeto. Para além disso, a *Google* disponibilizava um *Software Development Kit* (SDK) para Python, o que permitia uma fácil integração com a solução existente em *Robot Framework*.

Mantendo-se a estrutura geral da abordagem anterior, continuou a ser utilizado o *script* `pdf.robot` como componente de extração de informação dos documentos. A principal alteração residiu na substituição do código relativo ao uso de expressões regulares pelas funções de uma nova biblioteca *Python*, `gemini.py`, criada especificamente para servir como interface com o modelo da *Google*. Esta biblioteca estabelecia a ligação com a API do *Gemini*, sendo responsável por enviar os *prompts* e processar as respostas devolvidas pelo modelo.

Para tal, foi instalado no ambiente do projeto o SDK da IA generativa da *Google*, cuja utili-

zação se revelou bastante simples. A interação com a API era realizada através de uma chave de autenticação (*API key*), da seleção do modelo a utilizar e do envio do *prompt*. O excerto de código 3.7 apresenta a função principal desenvolvida para este efeito.

```
def get_gemini_response(prompt):
    genai.configure(api_key="xxxxxx")
    model = genai.GenerativeModel("gemini-1.5-flash")
    response = model.generate_content(prompt)
    return response.text
```

Excerto de Código 3.7: Função responsável pela comunicação com o modelo *Gemini 1.5 Flash*

Os primeiros testes seguiam a seguinte abordagem:

1. o texto integral de um documento PDF (um caderno de encargos ou um programa de procedimento, por serem os documentos onde se encontra mais informação alvo) era extraído;
2. o conteúdo extraído era concatenado com o *prompt* de utilizador (*prompt* que contém as instruções para o modelo);
3. o resultado desta concatenação (o *prompt* final) era enviado ao modelo através da função `get_gemini_response`;
4. a resposta gerada pelo modelo era apresentada.

O primeiro *prompt* de utilizador foi definido da seguinte forma:
"Vais receber o texto extraído de um documento PDF. Quero que analises o documento e extraias as seguintes informações: 1) Preço base, 2) Preço anormalmente baixo, 3) Lista de documentos que constituem a proposta, 4) Critérios de adjudicação, 5) Prazo de execução. Quero que tenhas atenção ao contexto. Apresenta as informações extraídas em formato de lista. É possível que não haja todas as informações pedidas no texto, nesse caso deves retornar 'Não aplicável'. Aqui está o conteúdo do documento:\n"
Mesmo utilizando este método inicial, os resultados obtidos superaram significativamente os da abordagem baseada em expressões regulares, apesar de terem sido alcançados com um esforço consideravelmente menor em termos de implementação. Para exemplificar o funcionamento desta abordagem, apresenta-se de seguida um dos resultados obtidos para o Caderno de Encargos de um concurso público do Instituto Nacional de Emergência Médica (INEM) para a prestação de assistência técnica ao servidor e base de dados da sua plataforma interna *Iteams*:

```
"
Com base no Caderno de Encargos fornecido, aqui estão as informações
extraídas em formato de lista:

*   **Preço base:** 75.305,00 € (Ano 2025 - 37.300,00 €; Ano 2026 -
    38.005,00 €).
*   **Preço anormalmente baixo:** Não aplicável.
```

```
*   **Lista de documentos que constituem a proposta:**
    *   Clausulado contratual e seus anexos.
    *   Suprimentos dos erros e das omissões das peças do procedimento,
    identificados pelo concorrente e aceites pelo órgão competente.
    *   Esclarecimentos e as retificações relativos às peças do
    procedimento.
    *   Caderno de encargos e anexos.
    *   A proposta adjudicada.
    *   Esclarecimentos sobre a proposta adjudicada, prestados pelo
    adjudicatário.
*   **Critérios de adjudicação:** Não aplicável (o documento detalha
o objeto do contrato, condições de pagamento, e obrigações, mas não
especifica os critérios de adjudicação).
*   **Prazo de execução:** Início na data da outorga e término a 31 de
dezembro de 2025.
"
```

Neste exemplo, o modelo foi capaz de identificar corretamente quase todos os campos pretendidos, com exceção da lista de documentos que constituem a proposta. O modelo devolveu a lista de documentos associada ao contrato, quando, na realidade, os documentos requeridos para a proposta estavam especificados no Programa do Procedimento e não no Caderno de Encargos.
Após a realização de testes adicionais com diferentes concursos, foi possível retirar algumas conclusões preliminares:

- O modelo demonstrou capacidade para distinguir o conteúdo do documento do texto referente ao *prompt* de instruções, mesmo quando ambos eram concatenados num único texto;
- Apesar de o texto extraído dos PDFs perder, por vezes, qualidade estrutural, sobretudo em documentos com tabelas ou esquemas complexos, o modelo conseguiu, na maioria dos casos, reconstruir e interpretar o formato corretamente;
- As respostas eram devolvidas em formato *Markdown*;
- O formato das respostas não era totalmente consistente, variando tanto na frase inicial para introduzir a resposta, como na nomenclatura dos campos, o que poderia complicar o processo de pós-processamento necessário para inserir os dados extraídos na base de dados;
- A gestão de casos sem informação carecia de uniformidade: o modelo nem sempre retornava apenas "Não aplicável" como solicitado, podendo acrescentar justificações ou mesmo respostas inferidas a partir de conhecimento prévio;
- Verificou-se uma diminuição da qualidade dos resultados para ficheiros muito extensos.

Apesar destas limitações, os resultados iniciais foram bastante promissores, sendo evidente que algumas das desvantagens, como a inconsistência no formato ou a gestão de casos sem informação, poderiam ser mitigadas através de *prompts* mais refinados e de uma extração mais detalhada do conteúdo e estrutura dos documentos. No entanto, a perda de qualidade com documentos de maior dimensão surgiu como uma potencial limitação crítica, uma vez que ainda se estava a trabalhar apenas com ficheiros individuais, mas a solução final deveria suportar a análise de múltiplos documentos por concurso, o que, inicialmente se idealizou como ainda mais texto para ser processado pelo modelo. Esta preocupação, justificou o aprofundamento de estratégias para o envio de múltiplos ficheiros antes de prosseguir para casos mais complexos.
Após os primeiros resultados promissores, iniciou-se o desenvolvimento de uma solução para enviar todos os ficheiros de um concurso ao modelo, explorando várias abordagens:

- A primeira abordagem consistiu em carregar todos os ficheiros de um concurso para memória, extrair e concatenar o texto de cada documento e enviar o texto agregado para o modelo. O *prompt* utilizado era o seguinte:

  ”Vais receber o texto extraído de vários documentos PDF. Quero que analises os documentos e extraias as seguintes informações: 1) Preço base, 2) Preço anormalmente baixo, 3) Lista de documentos que constituem a proposta, 4) Critérios de adjudicação, 5) Prazo de execução. Quero que tenhas atenção ao contexto. Apresenta as informações extraídas em formato JSON. Se identificares tabelas no texto extraído, formata a resposta de maneira a que se perceba a estrutura da tabela no PDF. No caso dos ’Critérios de adjudicação’ e dos ’Documentos que constituem a proposta’ retira TODA a informação desde o início do artigo até ao final (que é onde começa o próximo artigo). É possível que não haja todas as informações pedidas no texto, nesse caso deves retornar ’Não encontrado’. Aqui está o conteúdo do documento:\n” + [Texto do PDF]

  Como se pode observar, foram também feitas algumas alterações ao *prompt*, com a finalidade de contornar problemas detetados em iterações anteriores, solicitando explicitamente a extração em formato JSON, a preservação da estrutura de tabelas e a extração integral de artigos relevantes para os critérios de adjudicação e documentos da proposta. Esta solução revelou-se adequada apenas para concursos compostos por documentos de reduzida dimensão. Na maioria dos casos, a quantidade de texto era demasiado elevada, resultando em respostas de fraca qualidade ou, em alguns casos, no exceder do limite de *tokens* permitido pela API. Pela sua ineficácia, esta abordagem foi descartada.

- Na segunda tentativa, passou a realizar-se múltiplos pedidos ao modelo, um por cada documento do concurso. Cada pedido pedia ao modelo para devolver a resposta em formato JSON, cuja estrutura se assemelhava à seguinte:

```
{
    'Preço base': valor,
    'Preço anormalmente baixo': valor,
```

```
    'Documentos da proposta': valor,
    'Critérios de adjudicação': valor,
    'Prazo de execução': valor,
}
```

Esta estrutura era utilizada para manter o contexto entre diferentes pedidos, sendo concatenada com o *prompt* da execução seguinte. Assim, quando um novo documento era processado, o modelo recebia também a resposta acumulada das execuções anteriores e só deveria preencher os campos cujo valor se mantinha como ”Não encontrado”. O *prompt* utilizado para este efeito foi o seguinte:

’Vais receber o texto extraído de um documento PDF. Quero que analises o documento e extraias as seguintes informações: 1) Preço base, 2) Preço anormalmente baixo, 3) Lista de documentos que constituem a proposta, 4) Critérios de adjudicação, 5) Prazo de execução. Quero que tenhas atenção ao contexto. Apresenta as informações extraídas em formato JSON. Se identificares tabelas no texto extraído, formata a resposta de maneira a que se perceba a sua estrutura da tabela no PDF. No caso dos ’Critérios de adjudicação’ e dos ’Documentos que constituem a proposta’ retira TODA a informação desde o início do artigo até ao final (que é onde começa o próximo artigo). É possível que não haja todas as informações pedidas no texto, nesse caso deves retornar ’Não encontrado’. Vais também receber a resposta JSON da execução anterior deste *prompt*. Caso seja a primeira execução, essa resposta terá apenas a *string* ’Primeira execução’, que deverás substituir pela resposta nova. Caso já tenha uma resposta anterior, procura informação apenas relativas aos campos cujo valor é “Não encontrado” e não extraias informação relativa aos campos já encontrados. Aqui está o conteúdo do documento:\n’ + [Texto do PDF] + ’\nAqui está a resposta anterior:\n’ + [Resposta JSON do pedido anterior];

Este método provou ser capaz de transportar informação entre diferentes pedidos, conseguindo analisar um concurso completo em vez de apenas um ficheiro. Uma melhoria adicional consistiu em retirar a lógica da primeira execução da responsabilidade do modelo, simplificando o processo: a primeira chamada utilizava um *prompt* sem contexto, e apenas as seguintes integravam a resposta JSON anterior.

- Foi ainda desenvolvida uma terceira abordagem, pelo surgimento de um novo desafio: num dos concursos testados, um dos documentos continha texto suficiente para exceder o limite de *tokens* mesmo quando enviado individualmente. Para resolver esta limitação, implementou-se uma terceira abordagem, em que os documentos eram divididos em *chunks* de dimensão constante. O processo e os *prompts* mantiveram-se inalterados, mas cada *chunk* era tratado como uma unidade independente para não ultrapassar o limite imposto pela API.

  Mais tarde, percebeu-se que dividir todos os documentos reduzia significativamente a eficácia do modelo, pois o contexto fragmentava-se excessivamente. Assim, passou-se a calcular previamente o número de *tokens* de cada ficheiro (através de uma função

do SDK do *Gemini*) e a aplicar a divisão em *chunks* apenas aos documentos de maior dimensão, enviando os restantes na íntegra.

Em suma, embora ainda houvesse aspetos a otimizar, esta experiência demonstrou de forma clara que a utilização de uma LLM representava o caminho mais viável e eficaz para o problema de extração automática de informação de documentos de concursos públicos. O desempenho do modelo *Gemini* superou amplamente o das expressões regulares, provando que a interpretação contextual e semântica é essencial para lidar com a diversidade e complexidade dos documentos reais.
Estas conclusões motivaram a continuação do trabalho com LLMs, explorando outras opções e estratégias para melhorar a qualidade, a consistência e a escalabilidade da solução, o que conduziu à abordagem seguinte, descrita na próxima subsecção.

3.5.3 Experiências Complementares como Provas de Conceito

Apesar da abordagem com o *Gemini* ter produzido resultados bastante satisfatórios, o modelo ainda apresentava algumas respostas erradas ou incompletas, e o formato de saída não era totalmente consistente, sendo difícil atingir essa consistência apenas através de *prompt engineering*. Perante estas limitações, foi feita uma pesquisa sobre técnicas de processamento inteligente de documentos, da qual surgiram alguns conceitos e ferramentas que deram origem a duas experiências exploratórias, descritas em seguida, que serviram essencialmente como provas de conceito.

3.5.3.1 LLM Local

A primeira experiência consistiu na instalação e teste de uma LLM local, o que permitiria ter maior controlo sobre o modelo, além da possibilidade de o treinar e ajustar às necessidades específicas do projeto. Esta ideia foi discutida com o supervisor do estágio, que alertou que não estavam disponíveis ao projeto recursos computacionais adequados (nomeadamente *Graphics Processing Unit* (GPU)s) para suportar uma solução desta natureza de forma sólida. Ainda assim, foi autorizada a realização de testes numa escala local, apenas com o objetivo de avaliar a sua viabilidade.
Foi então instalado o modelo *Ollama 3.2*, na versão 3B (cerca de três mil milhões de parâmetros). Tratava-se de um modelo relativamente pequeno, adequado às limitações da máquina utilizada para os testes (armazenamento de 477 GB, placa gráfica de 128 MB e 16 GB de *Random Access Memory* (RAM)). Naturalmente, o seu desempenho era muito inferior ao do modelo *Gemini 1.5 Flash*.
O processo experimental foi desenvolvido em *Python*, seguindo uma lógica semelhante à dos primeiros testes com o *Gemini*:

- carregava-se um ficheiro PDF;
- extraía-se o conteúdo textual;
- concatenava-se o texto ao *prompt*;

- o resultado era enviado ao modelo, aguardando-se a resposta.

Os resultados não foram satisfatórios. O modelo demorava cerca de dois minutos e meio para processar ficheiros pequenos e, na maioria dos casos, não conseguia identificar corretamente os campos solicitados no *prompt*. Concluiu-se, portanto, que para implementar uma solução deste tipo com resultados significativos seria necessário dispor de *hardware* adequado, motivo pelo qual a abordagem foi abandonada antes de se avançar para o treino ou ajuste do modelo.

3.5.3.2 RAG

A segunda experiência surgiu de uma pesquisa sobre o conceito de RAG, já apresentado no capítulo do Estado da Arte 2. Este método combina recuperação de informação com geração de texto, permitindo que o modelo consulte uma base de conhecimento externa antes de responder. De forma imediata, percebeu-se o potencial desta abordagem para resolver o problema da análise de múltiplos ficheiros de um mesmo concurso: ao armazenar todos os documentos na base de conhecimento, o modelo teria acesso ao contexto completo do concurso sem necessitar de métodos adicionais como os utilizados na abordagem com o *Gemini*.
A implementação baseou-se num repositório público do *GitHub* [25] e numa vídeo-aula [26] do mesmo autor, diferindo essencialmente no facto de o modelo de linguagem utilizado ter sido o *Gemini* (em vez do *ChatGPT*, usado no tutorial) e de o objetivo não ser a criação de um *chatbot*, mas sim a extração estruturada de informação a partir dos documentos.
O processo, desenvolvido também em *Python*, consistia nas seguintes etapas:

- carregar os documentos de uma pasta;
- dividir o conteúdo em *chunks* de 1000 caracteres, com um *overlap* de 500;
- armazenar os *chunks* numa base de dados *Chroma*, que utiliza vetores de *embeddings* (vetores que representam conteúdo textual) como chaves;
- por fim, fazia-se uma *query* à base de dados com base no *prompt* definido e a consulta era comparada com os vetores armazenados, retornando-se os segmentos de texto com maior similaridade semântica.

Grande parte destas funcionalidades foi implementada através de funções biblioteca *LangChain*, uma *framework open-source* desenhada para facilitar a criação de aplicações que utilizam LLMs.
Os resultados foram satisfatórios, demonstrando que o conceito era viável e aplicável ao contexto do projeto. Contudo, o desempenho obtido foi equivalente ao da abordagem anterior com o *Gemini*, sem ganhos significativos que justificassem o aumento da complexidade técnica. Assim, o uso do RAG foi, por enquanto, colocado em suspenso nesta fase do desenvolvimento.

3.5.3.3 Conclusão

Estas experiências, embora não tenham trazido melhorias diretas ao sistema em desenvolvimento, revelaram-se fundamentais para clarificar o rumo a seguir. Permitiram compreender melhor as limitações práticas das LLMs locais e o potencial das soluções baseadas na nuvem, além de possibilitarem o conhecimento de um conceito fundamental, RAG, que posteriormente integraria a solução final do projeto.

3.5.4 Extração de Informação com *Prompting* do *ChatGPT*

Dando continuidade à abordagem que até então apresentara os melhores resultados (a utilização da API do *Gemini*), e procurando colmatar algumas das suas limitações, decidiu-se experimentar um novo modelo de linguagem, o *ChatGPT* da *OpenAI*. Ao contrário da API da *Google*, a API da *OpenAI* não disponibilizava um plano gratuito, sendo necessário recorrer a um modelo de pagamento "*pay-as-you-go*", no qual o custo é calculado com base na quantidade efetiva de *tokens* processados em cada pedido, ou seja, paga-se apenas pelo que é utilizado.
Após análise das opções disponíveis, foi escolhido o modelo *GPT-4o mini*, por oferecer um excelente equilíbrio entre custo e desempenho. De acordo com as métricas apresentadas na plataforma *LLM Stats* [27], os modelos da família *GPT* encontravam-se (a meados de fevereiro de 2025) entre os de melhor pontuação no *benchmark Graduate-level Google-Proof Q&A* (GPQA), um teste amplamente utilizado para avaliar a capacidade de raciocínio e compreensão avançada de texto em modelos de linguagem.
Assim, iniciaram-se os testes com o objetivo de avaliar se a adoção desta abordagem poderia melhorar a consistência, a precisão e o formato das respostas obtidas na fase de extração de informação.

3.5.4.1 API *ChatCompletions*

O primeiro passo consistiu em utilizar a API *Chat Completions*, que permite enviar mensagens estruturadas para o modelo e receber respostas contextuais, possibilitando a definição de comportamentos específicos através de *prompts* separados para o sistema e para o utilizador.
Para integrar a API no projeto, foi criada a biblioteca `chatgpt.py`, onde foram definidas funções para fazer pedidos ao modelo utilizando o SDK da *OpenAI*. À semelhança da abordagem anterior com o *Gemini*, foi necessário inicializar o cliente da API, obter o conteúdo do ficheiro a analisar, definir o *prompt* e enviar o pedido ao modelo para obter a resposta. O cliente era inicializado com a função demonstrada no excerto de código 3.8.

```
def get_model(api_key_json_path):
    with open(api_key_json_path, 'r', encoding='utf-8') as file:
        data = json.load(file)
        key = data.get('openai_api_key')
    client = OpenAI(api_key=key)
    return client
```

Excerto de Código 3.8: Função responsável por inicializar o cliente da API do *ChatGPT*

O conteúdo do ficheiro PDF a ser analisado era carregado com a função representada no excerto de código 3.9.

```
pdf_content = pdf_handler.get_content_in_string(pdf_handler.extract_text_from_pdf
    (path_to_file))
```

Excerto de Código 3.9: Carregamento do conteúdo do PDF.

A requisição ao modelo era feita como visível em 3.10.

```
response = client.chat.completions.create(
    messages=[
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": prompt}
    ],
    model="gpt-4o",
    temperature=0.3
)
```

Excerto de Código 3.10: Requisição ao modelo.

Nesta abordagem, o *prompt* foi dividido em duas partes. O *system prompt* definia o comportamento geral e a "personalidade" do modelo, indicando quais os campos a extrair e o objetivo da análise, enquanto o *user prompt* continha o conteúdo do PDF a ser processado. Para garantir consistência nas respostas, o valor da propriedade *temperature* foi reduzido para 0.3. Esta propriedade controla o grau de variabilidade e criatividade nas respostas do modelo, variando entre 0 e 2: valores baixos produzem respostas mais determinísticas, enquanto valores elevados geram maior diversidade. Como o objetivo era obter respostas estruturadas e consistentes, sem desvios do conteúdo do documento, 0.3 revelou-se adequado após alguns testes preliminares.

O processo de *prompt engineering* consistiu em diversas iterações de tentativa e erro, alinhadas com as *guidelines* da documentação oficial da *OpenAI* (*Prompt Engineering Guide*). Para refinar ainda mais a eficácia do *prompt*, foi utilizado um *meta prompt* (obtido também através da documentação oficial), que é uma técnica que permite gerar *prompts* mais precisos e estruturados com base em instruções de alto nível fornecidas ao modelo, resultando no *prompt* final que foi aplicado nos testes:

"És um especialista em análise de documentos e extração de informação. Vou fornecer o texto de um documento pertencente a um concurso público e uma lista de perguntas relativas à informação que tens de retirar. Deves responder às perguntas de forma coerente com a informação do documento e completa, sem desviar muito do texto do documento.

A lista de perguntas a que deves responder é a seguinte: 1) Qual é o preço base da proposta? (valor máximo que a entidade adjudicante está disposta a pagar pelo serviço) 2) Qual é o preço anormalmente baixo (valor que está significativamente abaixo do preço base ou dos preços normalmente praticados no mercado para o serviço)? 3) Quais são os documentos pela qual a proposta ao concurso deve ser constituída? 4) Quais são os critérios de adjudicação e todos os seus fatores e subfatores (parâmetros utilizados pela administração pública

para avaliar as propostas apresentadas pelos concorrentes e decidir qual delas será a vencedora)? 5) Qual é o prazo de execução (período estipulado no contrato para que o concorrente vencedor conclua a execução do serviço)?

A resposta que deves devolver deve ter um formato JSON, seguindo o seguinte *template*:

"Preço base": "Resposta à pergunta 1", "Preço anormalmente baixo": "Resposta à pergunta", "Documentos que constituem a proposta": "Resposta à pergunta 3", "Critérios de adjudicação": "Resposta à pergunta 4", "Prazo de execução": "Resposta à pergunta 5"

Caso não encontres a informação necessária para responder a alguma questão, a resposta a essa questão deve ser "Não encontrado".

Passos

1) Analisar o texto, identificando conteúdo, títulos, rodapés, parágrafos, tabelas e outros elementos que aches relevante para uma melhor compreensão do documento e a sua estrutura; 2) Identificar as secções do texto que são importantes para responder às questões feitas; 3) Com foco nas secções identificadas no passo anterior, desenvolver uma resposta para cada uma das questões.

Notas

- Assegura-te de que as respostas se mantêm alinhadas com a informação do documento.
- Assegura-te de que as respostas relativas aos 'Documentos que constituem a proposta' e 'Critérios de adjudicação' são completas e não excluem nenhum ponto importante. "

Como a API impunha um limite máximo de *tokens* por *request*, foi necessário implementar um mecanismo de processamento distribuído para cada concurso. Tal como na abordagem do *Gemini*, o objetivo era garantir que todos os documentos de um mesmo concurso fossem analisados, independentemente da sua dimensão. Assim, para cada concurso público, eram realizados múltiplos pedidos ao modelo, correspondendo um *request* por documento.

Nos casos em que o conteúdo de um documento ultrapassava o limite de *tokens* permitido pela API, o texto era automaticamente dividido em *chunks* de tamanho fixo, sendo feito um *request* por *chunk*. Desta forma, o sistema conseguia lidar tanto com concursos compostos por vários ficheiros como com documentos individuais de grandes dimensões, sem comprometer a execução do processo.

As respostas devolvidas pelo modelo eram guardadas em ficheiros JSON individuais, nomeados de acordo com o documento a que pertenciam. No caso de documentos divididos em *chunks*, era acrescentado um contador sequencial ao nome do ficheiro (por exemplo, `A1.json`, `A2.json`, ...), indicando a ordem das partes processadas.

No final da execução, uma função de agregação era responsável por reunir todas as respostas de um mesmo concurso, produzindo um único ficheiro consolidado. Caso um mesmo campo tivesse sido identificado em vários documentos (ou *chunks*), as respostas eram concatenadas no JSON final, separadas pelo caractere "|". Esta abordagem contrastava com a utilizada no *Gemini*, em que apenas a primeira ocorrência de um determinado campo era considerada. No *ChatGPT*, todas as respostas relevantes eram preservadas, permitindo uma visão mais completa da informação extraída.

Outra melhoria importante foi a introdução de um formato de resposta estruturado através do parâmetro *response_format* da API. Esta funcionalidade permitia definir explicitamente

a estrutura de saída do modelo com base num esquema JSON (*JSON Schema*), tornando desnecessário especificar o formato no próprio *prompt*. O modelo era, assim, obrigado a devolver um objeto JSON com os campos definidos, garantindo uniformidade e validade sintática nas respostas e simplificando o processo de pós-processamento. O excerto 3.11 ilustra a configuração desta estrutura.

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": prompt}
    ],
    temperature=0.3,
    response_format={
        "type": "json_schema",
        "json_schema": {
            "name": "document_analysis_response",
            "schema": {
                "type": "object",
                "properties": {
                    "preco_base": {"type": "string", "description": "preço base
                        da proposta, ou seja, valor máximo que a entidade
                        adjudicante está disposta a pagar pelo serviço"},
                    "preco_anormalmente_baixo": {"type": "string", "description":
                         "preço anormalmente baixo, ou seja, valor
                        significativamente abaixo do preço base ou dos preços
                        normalmente praticados no mercado"},
                    "criterios_de_adjudicacao": {"type": "string", "description":
                         "critérios de adjudicação e todos os seus fatores e
                        subfatores"},
                    "documentos_que_instruem_a_proposta": {"type": "string", "
                        description": "documentos que devem constituir a proposta
                         ao concurso"},
                    "prazo_de_execucao": {"type": "string", "description": "prazo
                         de execução (período estipulado no contrato para a
                        conclusão do serviço)"}
                },
                "required": [
                    "preco_base",
                    "preco_anormalmente_baixo",
                    "criterios_de_adjudicacao",
                    "documentos_que_instruem_a_proposta",
                    "prazo_de_execucao"
                ],
                "additionalProperties": False
            },
            "strict": True
        }
```

```
    },
)
```

Excerto de Código 3.11: Requisição ao modelo com especificação do formato de resposta.

Esta nova forma de estruturar o *output* trouxe uma vantagem imediata: simplificou o *prompt*, tornando-o mais direto e conciso, e garantiu que todas as respostas fossem devolvidas num formato padronizado e validável.

Os resultados obtidos com esta abordagem, que podem ser observados na tabela 3.2, foram bastante satisfatórios, apresentando alta precisão na extração do preço base, critérios de adjudicação e prazo de execução. Os critérios de adjudicação apresentavam ainda bastantes respostas incompletas, pois o modelo resumia esta secção, mesmo tendo instruções contrárias. O preço anormalmente baixo e os documentos da proposta eram os campos com menos respostas corretas. Nesta e na próxima abordagem contou-se também como respostas incompletas os campos em que, no final da extração, tinham mais do que uma resposta e nem todas eram as corretas.

Tabela 3.2: Resultados dos testes de precisão da abordagem da API *ChatCompletions* para um conjunto de 30 concursos.

Campo	Taxa de acerto	Taxa de acerto incompleto
Preço base	100,00%	0,00%
Preço anormalmente baixo	70,00%	6,67%
Critérios de Adjudicação	83,32%	16,67%
Documentos da Proposta	56,67%	40,00%
Prazo de Execução	93,32%	6,67%

3.5.4.2 *Fine Tuning* do Modelo

De modo a corrigir algumas incoerências nas respostas do modelo e melhorar a precisão na extração de certos campos, foi explorada uma funcionalidade disponibilizada pela *OpenAI* — o *fine-tuning* dos seus modelos. Esta técnica permite ajustar o comportamento de um modelo pré-treinado a um contexto ou domínio específico, fornecendo exemplos concretos de entrada e saída esperada, para que o modelo aprenda a responder de forma mais consistente e adaptada ao tipo de tarefa em questão.

Para realizar este processo, foi construído um dataset de treino no formato *JavaScript Object Notation Lines* (JSONL), conforme especificado na documentação oficial da *OpenAI*. Cada linha do ficheiro representa um exemplo de treino e contém o *prompt* de sistema, o *prompt* de utilizador e a respetiva resposta esperada.

A estrutura de cada entrada segue o formato presente no excerto 3.12.

```
{"messages": [
    {"role": "system", "content": "[prompt do sistema]"},
    {"role": "user", "content": "[conteúdo do pdf]"},
    {"role": "assistant", "content": "[resposta esperada]"}
]}
```

Excerto de Código 3.12: Estrutura de um item do dataset de treino.

Foram utilizados 30 concursos públicos para a construção do *dataset*, totalizando 133 exemplos distintos, correspondentes a ficheiros individuais ou a *chunks* de ficheiros que ultrapassavam o limite de *tokens*.

O processo de construção do *dataset* foi cuidadosamente delineado de modo a garantir a qualidade e a fiabilidade dos dados. Começou por executar-se o *script* de extração de informação sobre os 30 concursos, obtendo os ficheiros JSON resultantes de cada documento ou *chunk*. Posteriormente, foi criada uma cópia destes ficheiros numa diretoria própria, onde se procedeu à validação manual dos resultados: cada resposta extraída pelo modelo foi comparada com o conteúdo real do documento e, quando necessário, corrigida manualmente.

Depois de obtido um conjunto com 100% de respostas corretas, foi criado um *script* auxiliar responsável por gerar automaticamente o ficheiro JSONL no formato requerido. Este *script* associava cada ficheiro (ou *chunk*) ao respetivo conteúdo textual e criava as entradas correspondentes no *dataset* final.

Concluída esta fase, o ficheiro JSONL foi importado na plataforma da *OpenAI*, através da secção dedicada ao *fine-tuning*. O processo resultou num novo modelo, derivado do *GPT-4o-mini*, denominado '`ft:gpt-4o-mini-2024-07-18:personal::xxxxxx`', com o mesmo desempenho geral e capacidade linguística, mas ajustado ao contexto específico da análise de documentos de concursos públicos.

Após o processo de *fine-tuning*, o modelo foi testado em concursos diferentes daqueles utilizados para o treino, de forma a avaliar a sua capacidade de generalização. Os resultados, respresentados na tabela 3.3 mostraram melhorias significativas, sobretudo em aspetos que não tinham sido alcançados por nenhuma das abordagens anteriores. É importante referir que o processo de avaliação da precisão da abordagem é um processo demorado, por requerer uma análise individual tanto do concurso como da resposta. Nesta fase do projeto, foi pedido um pouco mais de celeridade no desenvolvimento do projeto, de modo a conseguir-se dar resultados para fazer uma demonstração à equipa de vendas e, por essa razão, foram feitos menos testes, tendo sido usado um conjunto de 11 concursos.

Tabela 3.3: Resultados dos testes de precisão da abordagem da API *ChatCompletions*, utilizando o modelo ajustado, para um conjunto de 11 concursos.

Campo	**Taxa de acerto**	**Taxa de acerto incompleto**
Preço base	100,00%	0,00%
Preço anormalmente baixo	100,00%	0,00%
Critérios de Adjudicação	90,90%	9,10%
Documentos da Proposta	90,90%	9,10%
Prazo de Execução	90,90%	0,00%

O avanço mais notável verificou-se no campo dos "documentos que constituem a proposta", tradicionalmente um dos mais difíceis de capturar com precisão. Antes do ajuste, o modelo tendia a incluir neste campo listas incorretas de documentos, confundindo frequentemente com outras listas presentes nos mesmos ficheiros (como anexos ou documentos administrati-

vos). Após o *fine-tuning*, o modelo passou a identificar corretamente apenas os documentos relevantes, alinhando-se de forma muito mais consistente com o esperado.
Outra melhoria relevante foi a capacidade do modelo para reconhecer quando não existe resposta para determinado campo. Durante a construção do *dataset*, uma das correções mais recorrentes consistiu em eliminar respostas inventadas ou incorretas que o modelo gerava em ficheiros com pouca informação, resultado da sua tentativa de preencher todos os campos obrigatórios. Com o treino ajustado, o modelo aprendeu a deixar o campo vazio quando apropriado, o que se traduziu numa redução clara do número de respostas incorretas ou repetidas no resultado final.

3.5.4.3 API *Assistants*

Apesar dos resultados muito satisfatórios obtidos com o *fine-tuning*, a extração final de informação de cada concurso ainda apresentava algumas limitações práticas. Uma das principais era o facto de poderem existir múltiplas respostas para o mesmo campo, resultantes da análise de diferentes documentos pertencentes ao mesmo concurso. Este comportamento exigia um pós-processamento adicional, que se revelou pouco linear.
Por exemplo, quando o modelo encontrava a mesma informação em mais do que um ficheiro, as respostas eram semanticamente idênticas, mas apresentavam variações na forma textual, o que dificultava a sua deduplicação automática. Um mesmo valor de preço base poderia surgir como "559.580,07 €", "O preço base é de 559.580,07 €" ou até "559.580,07 € (quinhentos e cinquenta e nove mil quinhentos e oitenta euros e sete cêntimos)". Apesar de transmitirem exatamente a mesma informação, essas diferenças de formatação impediam uma consolidação direta das respostas.
Por outro lado, nos casos em que existiam respostas realmente diferentes para o mesmo campo, poderia ser feita uma nova chamada ao modelo para decidir qual era a mais provável de estar correta. No entanto, essa abordagem apresentava duas desvantagens significativas: primeiro, o modelo continuava sem acesso ao contexto global de todos os documentos do concurso, o que limitava a sua capacidade de decisão; segundo, o aumento do número de *requests* tornaria o processo mais complexo e dispendioso.
Manter múltiplas respostas na versão final do produto seria indesejável, uma vez que poderia gerar confusão no utilizador, forçando-o a verificar manualmente os documentos originais para validar a informação. Após discutir esta limitação com o supervisor, confirmou-se que este comportamento não correspondia ao resultado esperado para a aplicação.
Face a estas limitações, tornou-se necessário adotar uma nova abordagem, uma que permitisse, de forma definitiva, dar ao modelo o contexto completo do concurso durante a análise. Foi nesse momento que se regressou ao conceito de RAG. A *OpenAI* disponibiliza uma ferramenta que implementa precisamente este tipo de solução: a API *Assistants*.
A API *Assistants* permite criar assistentes personalizados especializados em determinadas tarefas, como classificação, extração de informação ou análise de documentos. Uma das suas funcionalidades mais relevantes é o *File Search*, que possibilita o envio direto de múltiplos ficheiros para o ambiente do assistente. Estes ficheiros são automaticamente processados e armazenados numa base de dados vetorial interna, implementando uma forma nativa e

otimizada de RAG. Assim, o modelo passa a ter acesso a todo o contexto dos documentos sem necessidade de extrair manualmente o texto ou recorrer a mecanismos adicionais de *Optical Character Recognition* (OCR).

Esta abordagem apresentou algumas diferenças significativas de implementação em relação às anteriores, tanto na forma de estruturar o *prompting* como no modo de interação com o modelo.

O processo começava pela definição de um *prompt* de instruções, responsável por estabelecer o comportamento e o objetivo geral do assistente, como definido no excerto 3.13.

```
instructions_prompt = '''És um especialista em análise de documentos e extração
    de informação.
Vão ser-te fornecidos documentos que constituem um concurso público.
A tua função é usar essa base de conhecimento para extrair a informação
    pretendida.

É possível que algumas das informações que te vou pedir para extrair não se
    encontrem na tua base de conhecimento.
Caso isso aconteça, não inventes ou retornes respostas sem fundamento e responde
    exclusivamente com a string vazia ("").

Nos casos em que encontras informação, responde estritamente em formato Markdown.
    '''
```

Excerto de Código 3.13: Definição do *prompt* de instruções do *Assistant*.

O formato *Markdown* foi escolhido porque, em iterações anteriores, as respostas do modelo surgiam por vezes em HTML, o que criava inconsistências no *frontend*. Impor este formato no *prompt* não assegurou uma uniformização definitiva deste formato mas aumentou a sua consistência.

Em seguida, foi criada uma lista de *user prompts*, com uma pergunta individual para cada campo a extrair. Esta abordagem, exemplificado no excerto de código 3.14, contrasta com as anteriores, nas quais o modelo recebia um único *prompt* contendo todos os campos. A divisão por perguntas permitiu aumentar a precisão e reduzir a ambiguidade, já que o modelo se focava num único objetivo de cada vez.

```
user_prompts = {
        "concurso_de_interesse": "Consideras que uma empresa de desenvolvimento
            de software tem interesse em participar neste concurso? Ou seja, o
            serviço a contratar consiste no desenvolvimento ou manutenção de
            software? Responde exclusivamente com 'Sim' ou 'Não'.",
        "preco_base": "Qual é o valor do preço base do concurso? Está dividido
            por lotes?",
        "preco_anormalmente_baixo": "Está definido um valor para o preço
            anormalmente baixo? Se sim, qual é?",
        "criterios_de_adjudicacao": "Identifica o critério de adjudicação (
            parâmetro ou conjunto de parâmetros utilizados pela administração
            pública para avaliar as propostas apresentadas pelos concorrentes e
            decidir qual delas será a vencedora) e lista todos os seus fatores e
            subfatores.",
```

```
        "documentos_que_instruem_a_proposta": "Identifica e lista todos os
            documentos pelos quais a proposta ao concurso deve ser constituída.",
        "local_de_execucao": "Qual o local de execução do serviço?",
        "prazo_de_execucao": "Qual é o prazo de execução do serviço? Está
            dividido por lotes?",
        "especificacoes_funcionais": "Quais são os requisitos funcionais para o
            desenvolvimento do software pretendido? Há requisitos opcionais e
            obrigatórios? (Caso os requisitos funcionais não se apliquem a este
            tipo de concurso, responder exclusivamente com a string 'Não
            aplicável')",
        "especificacoes_tecnicas": "Quais são os requisitos técnicos para o
            desenvolvimento do software pretendido? Há requisitos opcionais e
            obrigatórios? (Caso os requisitos técnicos não se apliquem a este
            tipo de concurso, responder exclusivamente com a string 'Não
            aplicável')",
        "requisitos_equipa": "Quais são os perfis exigidos e os requisitos dos
            membros da equipa de desenvolvimento?"
    }
```

Excerto de Código 3.14: Lista de *user prompts* usados nas chamadas aos *assistants*.

Foi então inicializada uma instância do assistente, com as instruções definidas, o modelo selecionado e a ferramenta de *file search* ativada, como visível no excerto 3.15.

```
assistant = client.beta.assistants.create(
    name="Public Tender Analyst",
    instructions=instructions_prompt,
    model="gpt-4o-mini",
    tools=[{"type": "file_search"}],
)
```

Excerto de Código 3.15: Inicializar o cliente do *assistant*.

De seguida, criou-se uma *vector store* (excerto 3.16), responsável por armazenar os documentos do concurso em formato vetorial, o que permitiria ao modelo realizar pesquisas semânticas de forma eficiente.

```
vector_store = client.beta.vector_stores.create(name="Public Tender Documents")
```

Excerto de Código 3.16: Criação de uma *vector store*.

Os ficheiros de cada concurso eram carregados e enviados para a *vector store*, ficando imediatamente disponíveis na base de conhecimento do assistente (excerto de código 3.18).

```
file_streams = [open(path, "rb") for path in file_paths]
file_batch = client.beta.vector_stores.file_batches.upload_and_poll(
    vector_store_id=vector_store.id,
    files=file_streams
)
```

Excerto de Código 3.17: *Upload* de ficheiros para o *assistant*.

Depois disso, o *assistant* era atualizado para utilizar essa *vector store* como fonte de dados (excerto de código 3.18).

```
assistant = client.beta.assistants.update(
    assistant_id=assistant.id,
    tool_resources={"file_search": {"vector_store_ids": [vector_store.id]}},
)
```

Excerto de Código 3.18: Atualização do assistant para usar a *vector store* como base de conhecimento.

O passo seguinte consistia em fazer um pedido ao assistente para cada *user prompt* (exemplificado em 3.19). Este era o elemento que mais distinguia esta abordagem das anteriores: o modelo analisava apenas um campo de cada vez, consultando automaticamente os ficheiros mais relevantes da sua base de conhecimento para responder. Assim, o modelo baseava-se no contexto completo do concurso, encontrando a resposta mais relevante com base em similaridade semântica.

```
for prompt_key in user_prompts:
    thread = client.beta.threads.create(
        messages=[{"role": "user", "content": user_prompts[prompt_key]}]
    )

    run = client.beta.threads.runs.create_and_poll(
        thread_id=thread.id,
        assistant_id=assistant.id
    )
```

Excerto de Código 3.19: Ciclo para fazer o pedido de cada um dos user_prompts.

Após a obtenção das respostas, foi implementado um pós-processamento simples, cuja função era remover as citações automáticas que o modelo incluía para referenciar as fontes de onde retirava a informação. Essas citações continham apenas identificadores internos, sem relevância para o utilizador final, e eram removidas com função descrita no excerto de código 3.20.

```
def delete_citations(text):
    return re.sub(r"□\d+:\d†+□source", "", text)
```

Excerto de Código 3.20: Função para remover as citações da resposta do *assistant*.

Cada resposta era então associada à respetiva chave num dicionário *Python*, que posteriormente era convertido num ficheiro JSON (excerto de código 3.21).

```
def build_json_from_answers(json_dict, prompt_key, answer):
    json_dict[prompt_key] = answer
    return json_dict

json_answer = build_json_from_answers(
    json_answer,
    prompt_key,
    delete_citations(messages.data[0].content[0].text.value)
)
save_results(json_answer, output_folder + "/" + directory_name + ".json")
```

Excerto de Código 3.21: Guradar as respostas num ficheiro JSON.

Este processo apresentou ainda uma vantagem adicional: o controlo total sobre o formato de saída passou a estar do lado da aplicação, e não do modelo, o que simplificou o pós-processamento e aumentou a previsibilidade dos resultados.
Antes de apresentar os resultados, é importante referir que novos campos foram adicionados nesta fase:

- "Concurso de interesse": criado para ajudar a filtrar concursos relevantes para a área de atuação da empresa. Apesar de existir um filtro prévio no componente *scraper*, ainda passavam concursos não relacionados com o desenvolvimento de *software*; este campo, avaliado pelo modelo, funcionava como uma *flag* para sinalizar o interesse potencial de um concurso.
- "Local de execução", "Especificações funcionais", "Especificações técnicas" e "Requisitos da equipa": introduzidos após uma reunião de demonstração com a equipa de vendas, que identificou a importância de incluir estas dimensões no resultado final para melhor caracterizar o tipo de serviço.

Os resultados obtidos, representados na tabela 3.4, foram muito bons, demonstrando uma performance consistente e respostas contextualizadas. Embora, em alguns casos, o desempenho tenha sido ligeiramente inferior ao modelo ajustado via *fine-tuning*, esta abordagem apresentou uma vantagem estrutural decisiva: o acesso direto e integral ao contexto completo de todos os documentos do concurso e foco individual em cada um dos campos a extrair, permitindo respostas mais coerentes e completas. Para além disso, com a introdução de novos campos a extrair, um modelo ajustado (*fine-tuned*) teria de passar novamente por um novo processo de treino para alcançar bons resultados. Nesta abordagem, esse passo não é necessário, o que a torna significativamente mais flexível e adaptável a futuras alterações na estrutura dos dados a extrair.

Tabela 3.4: Resultados dos testes de precisão da abordagem da API *Assistants*, utilizando o modelo ajustado, para um conjunto de 20 concursos.

Campo	**Taxa de acerto**	**Taxa de acerto incompleto**
Preço base	95,00%	0,00%
Preço anormalmente baixo	90,00%	0,00%
Critérios de Adjudicação	90,00%	0,00%
Documentos da Proposta	85,00%	15,00%
Prazo de Execução	85,00%	0,00%

3.6 Tarefa 5: *Scraping* da Plataforma B

Concluído o desenvolvimento do componente responsável pela extração de informação, iniciou-se o desenvolvimento do *script* de recolha automática de concursos públicos disponíveis na plataforma B.
Esta tarefa começou logo após a conclusão do *scraping* da plataforma A, no entanto, durante a implementação para a plataforma B, surgiu um obstáculo técnico significativo: a

presença de um *Completely Automated Public Turing test to tell Computers and Humans Apart* (CAPTCHA), que impossibilitava a automatização completa do processo de recolha. Por essa razão, e dado que já se dispunha de concursos suficientes provenientes da plataforma A para prosseguir com o desenvolvimento do projeto, decidiu-se suspender temporariamente esta tarefa e prosseguir com as restantes partes do projeto.
Ao retomar esta tarefa, aplicou-se o mesmo algoritmo base de recolha de concursos utilizado anteriormente, com pequenas adaptações necessárias devido a diferenças subtis na interface entre as duas plataformas, como por exemplo o modo de seleção da data limite de apresentação de propostas, usada para filtrar os concursos dentro do intervalo de tempo pretendido, que num caso se realizava através de um calendário interativo e noutro por meio de um campo de texto.
Para contornar o problema do CAPTCHA, tentou-se inicialmente analisar o tráfego de rede da plataforma, com o intuito de perceber se, através da API interna, seria possível obter diretamente os IDs dos concursos resultantes de uma pesquisa por palavra-chave. Caso fosse possível aceder a esses IDs, seria então viável reconstruir os **URL!** (**URL!**)s das páginas individuais de cada concurso, onde se encontram disponíveis os respetivos documentos, sem necessidade de resolver o CAPTCHA. Contudo, mesmo após testes realizados no *Postman*, não foi possível obter os resultados pretendidos, dado que a API estava protegida contra acessos externos.
Como era importante obter também os concursos desta plataforma, optou-se por recorrer a uma solução externa, através do serviço *2Captcha*. Este serviço permite ultrapassar CAPTCHAs automaticamente, ao enviar o identificador do CAPTCHA para uma rede de colaboradores humanos, que o resolvem manualmente e devolvem uma chave de validação que é usada para resolver o CAPTCHA. Desta forma, foi possível automatizar novamente o processo de *scraping*, garantindo a recolha dos concursos da plataforma B sem necessidade de intervenção manual direta.

3.7 Tarefa 6: Criação do *Docker Container* e Integração da Aplicação *Web* e do *Scheduler*

Uma das tarefas inicialmente planeadas para o decorrer deste estágio acabou por ser desenvolvida por uma outra equipa, que ficou responsável pela criação da interface *web* do sistema. Esta interface foi desenvolvida com o *framework Hypertext Preprocessor (PHP) Laravel* e apresenta uma estrutura simples, funcional e intuitiva, permitindo uma interação direta com os resultados obtidos pelo sistema. A autenticação dos utilizadores foi implementada com recurso ao *Microsoft Azure*, restringindo o acesso exclusivamente às contas da *Latitudde*, garantindo assim a segurança e o controlo de acesso à aplicação. Após o *login*, o utilizador podia visualizar a lista de propostas recolhidas, selecionar uma proposta para consultar os campos extraídos pelo modelo (organizados de forma estruturada numa tabela), efetuar a transferência dos documentos correspondentes e até enviar por *email* os dados de uma proposta diretamente a partir da interface.
Para consolidar todos os componentes do projeto e garantir o seu funcionamento como um

sistema único e coeso, foi criado um *Docker container* que integrava todos os elementos desenvolvidos: o *scraper*, o componente de extração de informação, a base de dados, a interface web e o *scheduler*. O *scheduler* foi implementado através de um *cron job*, responsável por executar automaticamente o *pipeline* completo todos os dias às 20h. Este *cron job* invocava um *script Python* que, por sua vez, coordenava a execução sequencial das seguintes tarefas:

- *Scraping* da plataforma A;
- *Scraping* da plataforma B;
- Extração de informação através da LLM;

Desta forma, todo o processo desde a recolha dos concursos até à apresentação dos resultados passou a ser totalmente automatizado e centralizado dentro do ambiente *Docker*, garantindo a portabilidade do sistema.

Capítulo 4

Conclusões

O trabalho desenvolvido ao longo deste estágio resultou na criação de uma ferramenta robusta e funcional, capaz de cumprir com sucesso os objetivos inicialmente definidos e de oferecer um contributo real para a automatização de processos internos da empresa. No final do estágio, foi possível obter um sistema completo e autónomo, que diariamente, de forma automática e à mesma hora, pesquisa concursos públicos de interesse, realiza o seu armazenamento e procede à extração de informação relevante, apresentando posteriormente os resultados numa interface *web* intuitiva e de fácil utilização. Todo o sistema encontra-se integrado num ambiente *Docker*, garantindo portabilidade, escalabilidade e compatibilidade entre diferentes plataformas, o que facilita a sua manutenção e futura expansão.
Para além dos resultados técnicos alcançados, o estágio representou também uma oportunidade significativa de crescimento pessoal e profissional para o estagiário. A experiência contribuiu para o desenvolvimento de competências transversais essenciais, como o sentido de responsabilidade, a capacidade de comunicação e o trabalho em equipa. Adicionalmente, proporcionou uma visão prática do funcionamento de um ambiente corporativo na área das tecnologias de informação, permitindo uma melhor preparação para a futura inserção no mercado de trabalho.
A natureza do projeto revelou-se especialmente enriquecedora, não apenas pelo seu grau de complexidade técnica, mas sobretudo pela autonomia concedida ao longo de todo o processo. A empresa proporcionou um ambiente de grande liberdade e confiança, permitindo explorar diferentes abordagens, testar soluções e aprender com o próprio processo de tentativa e erro. Essa flexibilidade favoreceu o desenvolvimento da capacidade de decisão e de resolução de problemas de forma independente, estimulando uma postura proativa e criativa. O resultado foi um percurso de aprendizagem altamente produtivo, que culminou numa solução sólida, eficiente e com potencial de evolução contínua.

4.1 Trabalho Futuro

Qualquer projeto de engenharia de *software* encontra-se, por natureza, em constante evolução. Mesmo após atingir um estado funcional e estável, há sempre aspetos que podem ser otimizados, reformulados ou expandidos, seja para melhorar o desempenho, corrigir erros identificados em fases posteriores ou responder a novas necessidades que surgem num ambiente tecnológico em permanente mudança.
O projeto desenvolvido não é exceção a esta regra. Embora o produto final se tenha revelado plenamente funcional, a sua implementação evidenciou diversas oportunidades de melhoria, algumas identificadas durante o processo de desenvolvimento, outras emergindo da utilização prática da aplicação. Por motivos de tempo, ou por implicarem uma reestruturação

substancial de certas componentes, várias dessas ideias ficaram suspensas para fases futuras. Neste capítulo são apresentadas as principais propostas de melhoria e expansão que poderiam ser exploradas futuramente, quer com o objetivo de aperfeiçoar o que já foi construído, quer no sentido de aumentar o alcance e as capacidades do sistema.

4.1.1 Refatorização do Projeto

Uma das primeiras melhorias a considerar seria uma refatorização abrangente do projeto. Dado que o sistema foi planeado e desenvolvido integralmente por dois estagiários, é natural que existam aspectos de organização e estrutura de código que possam ser otimizados.
Embora o projeto tenha ficado logicamente bem segmentado, tal como ilustrado na figura de planeamento apresentada no capítulo da Implementação 3, há várias áreas que poderiam beneficiar de maior robustez e modularidade.
Um dos principais pontos a reforçar é o tratamento de erros e exceções. Como o sistema depende fortemente de componentes externos, está sujeito a falhas difíceis de prever:

- Sites-alvo podem alterar a sua estrutura, o que compromete o *scraping* de dados; o sistema deveria ser capaz de detetar e adaptar-se automaticamente a essas alterações, ou pelo menos reportar de forma clara o tipo de falha ocorrida;
- As chamadas aos modelos de linguagem também introduzem pontos de vulnerabilidade, com possíveis erros de comunicação, falhas internas do servidor ou alterações na API da *OpenAI*. Implementar mecanismos de *retry*, validação de respostas e monitorização automática seria essencial para aumentar a estabilidade.
- Poderiam ainda ser introduzidos testes automatizados (unitários e de integração) para garantir a integridade das principais funcionalidades após cada alteração de código.

Resumindo, uma refatorização cuidadosa permitiria tornar o projeto mais escalável e resiliente a alterações externas, assegurando uma base mais sólida para a sua manutenção e o seu crescimento futuro.

4.1.2 Eliminação Automática de Concursos Não Utilizados

Atualmente, o sistema não dispõe de um mecanismo para eliminar concursos antigos ou não utilizados armazenados no servidor. A longo prazo, essa limitação pode resultar num aumento desnecessário de espaço ocupado e numa gestão mais complexa dos ficheiros.
Uma melhoria natural seria a implementação de uma funcionalidade de limpeza automática, que poderia operar com base na data de *download* ou no estado de processamento dos concursos. Assim, apenas seriam mantidos os concursos relevantes para consulta ou histórico, otimizando o armazenamento e melhorando a organização do sistema.

4.1.3 *Chatbot* com Conhecimento de Todos os Concursos

Uma expansão particularmente interessante seria o desenvolvimento de um *chatbot* especializado com conhecimento global de todos os concursos processados até à data.

Durante a pesquisa sobre o mecanismo RAG, foi possível observar que muitos projetos aplicavam esta técnica para construir *chatbots* com conhecimento de documentos específicos, permitindo ao utilizador interagir diretamente com o sistema sobre o conteúdo desses documentos.
Seguindo essa linha, o *chatbot* poderia ser treinado ou configurado para responder a perguntas sobre os concursos disponíveis, ajudando a equipa de vendas a encontrar rapidamente oportunidades relevantes. Por exemplo, um utilizador poderia perguntar "Há algum concurso recente que envolva desenvolvimento de software para a área da saúde?" e o *chatbot* responderia com base na informação armazenada.
Para viabilizar esta abordagem, seria necessário gerar representações resumidas dos concursos, que alimentariam a base de conhecimento do assistente, tornando o sistema escalável mesmo com um grande número de documentos.

4.1.4 Integração com Outras Ferramentas Internas

Outra linha de evolução passa pela integração do sistema com outras ferramentas internas da empresa, como o *software* de gestão de projetos ou gestão de recursos humanos. Dessa forma, o modelo poderia aceder, por exemplo, às áreas de especialização dos programadores e comparar automaticamente essas competências com os requisitos técnicos de cada concurso.
Este tipo de integração abriria espaço a uma análise mais inteligente e contextualizada, permitindo à equipa identificar de forma automática os concursos mais adequados ao perfil da empresa. No entanto, devido à natureza sensível dos dados envolvidos, esta abordagem exigiria o uso de modelos privados ou instâncias de LLM hospedadas localmente, garantindo a segurança e confidencialidade da informação sensível da empresa.

4.1.5 Escrita Automática de Propostas

Finalmente, uma evolução natural e ambiciosa do projeto seria a geração automática de propostas para os concursos de interesse. Após identificar um concurso relevante, o sistema poderia gerar um documento de proposta, ou pelo menos um *template* inicial, com base nos requisitos extraídos e nas respostas obtidas pelo modelo.
Isto permitiria fechar o ciclo completo do processo, passando da identificação à resposta formal, com grande poupança de tempo e esforço para a equipa. Com uma integração adequada com dados internos (como histórico de propostas anteriores, informações da empresa e portefólio técnico), o modelo poderia produzir propostas altamente personalizadas e coerentes, prontas para revisão humana.

4.1.6 Conclusão

Neste contexto de automatização suportada por modelos de linguagem, o limite acaba por ser a própria imaginação. Há sempre novas formas de aplicar, otimizar e expandir o sistema, tornando-o cada vez mais eficaz e alinhado com as necessidades da empresa.

Bibliografia

[1] (2025) Rpa vs. traditional automation. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.javatpoint.com/rpa-vs-traditional-automation xvii, 7

[2] Latitudde. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://latitudde.com/ 1

[3] Grupo rit. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://grouprit.com/ 1

[4] W. van der Aalst, M. Bichler, and A. Heinzl, "Robotic process automation," *Robotic Process Automation*, 2018. 5, 6

[5] Robotic process automation (rpa). Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.uipath.com/rpa/robotic-process-automation 6

[6] What is rpa? Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.ibm.com/think/topics/rpa 6

[7] (2025) Uipath. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.uipath.com/ 6

[8] (2025) Blue prism. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.blueprism.com/pt/ 6

[9] (2025) Automation anywhere. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.automationanywhere.com/ 6

[10] R. Issac, R. Muni, and K. Desai, "Delineated analysis of robotic process automation tools," *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, 2018. 6, 7

[11] (2023) Os 15 principais casos de utilização de rpa (robotic process automation) por sector – finanças, cuidados de saúde, rh, contabilidade, fabrico e muito mais! Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.zaptest.com/pt-pt/os-15-principais-casos-de-utilizacao-de-rpa-robotic-process-automation-por-sector-financas-cuidac 7, 8

[12] Rpa case study. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://aqltech.com/casestudies/rpa-case-study-employee-onboarding/ 8

[13] (2021) Customer story: City of seattle city clears 6,200 cases, improving quality of services. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.automationanywhere.com/resources/customer-stories/city-of-seattle 8

[14] (2021) Cobmax use case: Removing growth obstacles with robotic process automation. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://community.ibm.com/community/user/automation/blogs/p-williams/2021/08/09/cobmax-use-case-removing-growth-obstacles-with-rob?ref=nanonets.com 8

[15] (2021) Gps in the south west turn to robots to access dorset care record. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.digitalhealth.net/2021/09/gps-in-southwest-turn-to-robots-to-access-dorset-care-record/ 9

[16] (2022) What are the limitations of rpa? Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.tutorialspoint.com/what-are-the-limitations-of-rpa 9

[17] (2024) Disadvantages of robotic process automation: Unmasking the hype. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.digitalhealth.net/2021/09/gps-in-southwest-turn-to-robots-to-access-dorset-care-record/ 9

[18] C. Engel, P. Ebel, and J. Leimeister, "Cognitive automation," *Electron Markets*, 2022. 9

[19] O que é automação inteligente? Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.automationanywhere.com/br/rpa/intelligent-automation 10

[20] R. Diouf, E. N. Sarr, O. Sall, B. Birregah, M. Bousso, and S. N. Mbaye, "Web scraping: State-of-the-art and areas of application," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 6040–6042. 10, 12

[21] (2025) The 10 best web scraping tools for 2025. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://infomineo.com/blog/best-web-scraping-tools-in-2025-top-picks-for-data-extraction/ 11

[22] Web scraping – saiba o que é e para que serve. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://pplware.sapo.pt/internet/web-scraping-saiba-o-que-e-e-para-que-serve/ 12

[23] (2023) O que são large language models (llms)? Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.redhat.com/pt-br/topics/ai/what-are-large-language-models 14

[24] (2025) Top 9 large language models as of january 2025. Data da última consulta: 3 de fevereiro de 2025. [Online]. Available: https://www.shakudo.io/blog/top-9-large-language-models 15

[25] An improved langchain rag tutorial (v2) with local llms, database updates, and testing. [Online]. Available: https://github.com/pixegami/rag-tutorial-v2 41

[26] Rag + langchain python project: Easy ai chat for your docs. [Online]. Available: https://www.youtube.com/watch?v=tcqEUSNCn8I&t=727s 41

[27] Llm leaderboard. [Online]. Available: https://llm-stats.com/ 42