



Sistema Automático de Publicação de Ofertas de Emprego com Recurso a Inteligência Artificial e Web Scraping

Unilinkr

2024/2025

1220784 David Miguel de Oliveira Costa Sousa

Sistema Automático de Publicação de Ofertas de Emprego com Recurso a Inteligência Artificial e Web Scraping

Unilinkr

2024/2025

1220784 David Miguel de Oliveira Costa Sousa



Licenciatura em Engenharia Informática

Julho de 2025

Orientador ISEP: **Luís Conceição**

Supervisor Externo: **Tomás Borges**

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, 3 de julho de 2025

Agradeço aos que se cruzaram no meu caminho e me fizeram chegar até aqui, com especial consideração aos meus pais e amigos, dos quais guardo bastante carinho e amor.

Resumo

O resumo que se segue tem como objetivo apresentar, de forma concisa e objetiva, os principais aspetos do trabalho desenvolvido. Contexto, objetivos e solução implementada são descritas, referindo tecnologias utilizadas e os respetivos resultados obtidos. Esta secção visa fornecer ao leitor uma visão geral do projeto, permitindo compreendê-lo rapidamente, mesmo sem uma leitura integral do relatório.

O presente relatório descreve o desenvolvimento de uma aplicação que se foca na implementação de tecnologias Web Scraping e sistemas Natural Language Processing (NLP) com Large Language Model (LLM), de forma a extrair ofertas de emprego e publicá-las na plataforma da Unilinkr. O principal objetivo deste projeto foi otimizar o processo de criação de novas ofertas, através da automatização do preenchimento de um formulário para cada oferta de emprego.

A solução implementada seguiu uma arquitetura modular, capaz de ser expansível para incorporar novas funcionalidades, seguindo uma Representational State Transfer (REST) Application Programming Interface (API), que permite o uso das funcionalidades implementadas. A ferramenta de Web Scraping permite recolher as mais variadas ofertas de emprego disponíveis na web, principalmente em plataformas semelhantes às da Unilinkr, e utilizar o sistema NLP para recolher todos os campos necessários, para o preenchimento completo do formulário.

O sistema encontra-se funcional, tendo sido altamente testado com dados simulados e com interações reais. Os resultados obtidos pela Artificial Intelligence (AI) API externa, neste contexto LLaMa API, foram acertados para informações específicas, mas não permite atingir o potencial máximo esperado, para texto ambíguo, incoerente ou insuficiente.

Desta forma, o projeto evidencia a importância de atualizar plataformas digitais para utilizarem tecnologias emergentes, as quais permitem aos utilizadores usufruírem positivamente destes sistemas, melhorando a interação e a relação dos mesmos com as organizações.

O tema deste trabalho aborda ofertas de emprego, a automatização de processos e o preenchimento de formulários. As tecnologias utilizadas incluem o processamento de linguagem natural (NLP), técnicas Web Scraping, grandes modelos de linguagem (LLM), prompt engineering e inteligência artificial (AI).

Palavras-chave: Ofertas de Emprego, Automatização de Processos, Formulário, Processamento de Linguagem Natural (NLP), Web Scraping, Grandes Modelos de Linguagem (LLM), prompt engineering, Inteligência Artificial (AI).

Abstract

The following summary aims to present, in a concise and objective manner, the main aspects of the work carried out. The context, objectives and solution implemented are described, referring to the technologies used and the respective results obtained. This section aims to provide the reader with an overview of the project, allowing them to quickly understand it, even without reading the entire report.

This report describes the development of an application that focuses on the implementation of Web Scraping technologies and NLP systems with LLM, in order to extract job offers and publish them on the Unilinkr platform. The main objective of this project was to optimise the process of creating new offers by automating the filling out of a form for each job offer.

The solution implemented followed a modular architecture, capable of being expanded to incorporate new features, following a REST API, which allows the use of the implemented features. The Web Scraping tool allows the collection of a wide variety of job offers available on the web, mainly on platforms similar to Unilinkr, and uses the NLP system to collect all the necessary fields to complete the form.

The system is functional and has been extensively tested with simulated data and real interactions. The results obtained by the external AI API, in this context LLaMa API, were accurate for specific information, but ambiguous, incoherent or insufficient text does not allow the maximum expected potential to be achieved.

Thus, the project highlights the importance of updating digital platforms to use emerging technologies, which allow users to positively enjoy these systems, improving their interaction and relationship within organizations.

The theme of this work addresses job offers, process automation, and form filling. The technologies used include natural language processing (NLP), Web Scraping techniques, large language models (LLM), prompt engineering, and artificial intelligence (AI).

Keywords: Job Offers, Process Automation, Form Filling, Natural Language Processing (NLP), Web Scraping, Large Language Models (LLM), Prompt Engineering, Artificial Intelligence (AI).

Agradecimentos

Primeiro de tudo, gostaria de agradecer ao Instituto Superior de Engenharia do Porto (ISEP) por me ter proporcionado uma experiência académica positiva, lúcida e alinhada com os meus objetivos. Agradeço pelas relações criadas ao longo do meu caminho, quer com professores e alunos, especialmente aos meus colegas de grupo Beatriz Silva, Guilherme Ribeiro e Tiago Carvalho, por me ajudarem a crescer, e ao meu orientador do ISEP, Luís Conceição, pelo seu papel durante todo este processo. Sem este instituto eu não os conhecia e sem eles eu não me tornaria na pessoa que sou hoje.

Agradeço a todas as minhas amizades mais antigas, principalmente aos meus pais, Maria e José, por se manterem do meu lado nos momentos mais difíceis, dando-me força e esperança para superá-las. Esta resiliência permitiu-me manter uma postura consciente, procurando sempre resolver problemas. O carinho que tenho por eles e o que eles refletem em mim, serviu como base para uma personalidade focada, capaz de cumprir os objetivos.

Gostaria de agradecer à Unilinkr pela oportunidade de trabalhar com eles, principalmente ao meu supervisor, Tomás Borges, por me ter apoiado durante todo o projeto e ajudado a compreender a plataforma. Também agradeço ao resto da equipa por serem excecionais à minha integração na mesma, conhecendo rostos novos e criando ligações, as quais levarei para o meu futuro.

Por fim, quero agradecer a todos aqueles que se cruzaram no meu caminho e me fizeram tomar os passos necessários até chegar aqui. Todo o processo foi custoso, exigente e de alto nível, mas a recompensa final reflete-se no conhecimento e memórias que se criaram durante todo este caminho - obrigado.

Conteúdo

- falta o glossário / Agradecimentos

Lista de Figuras	viii
Lista de Tabelas	ix
1 Introdução	1
1.1 Enquadramento	1
1.2 Contexto	2
1.3 Descrição do Problema	2
1.3.1 Objetivos	3
1.3.2 Abordagem	3
1.3.3 Contributos	4
1.3.4 Planeamento do Trabalho	5
1.4 Estrutura do Relatório	7
2 Estado da Arte	8
2.1 Trabalhos relacionados	8
2.1.1 Web Crawler & Web Scraping	8
2.1.2 Chatbots	11
2.2 Tecnologias Existentes	12
2.3 Discussão e Seleção	14
3 Análise e Desenho da Solução	16
3.1 Domínio do Problema	16
3.2 Requisitos	17
3.2.1 Requisitos Funcionais	17
3.2.2 Requisitos Não Funcionais	18
3.2.3 Palavras Chave:	19
3.3 Desenho	19
3.3.1 Visão Geral da Solução	20
3.3.2 Arquitetura de Alto Nível	20
3.3.3 Justificação da Arquitetura	32
3.3.4 Especificações Globais do Sistema	32
3.3.5 Padrão de Arquitetura e Boas Práticas	33
4 Implementação da Solução	35
4.1 Descrição da Implementação	35
4.1.1 Página do Admin	36
4.1.2 Página das Empresas	40
4.2 Testes	45
4.3 Avaliação da solução	48
5 Conclusões	50

5.1	Objetivos concretizados	50
5.2	Limitações e trabalho futuro	51
5.3	Apreciação final	52

Bibliografia	53
---------------------	-----------

Lista de Figuras

3.1	Modelo de domínio do sistema UnijobFormsBot	16
3.2	Diagrama de Casos de Uso	18
3.3	Vista lógica do sistema UnijobFormsBot (Nível 1)	21
3.4	Vista de processos para o caso de uso UniScraper (Nível 1)	21
3.5	Vista de processos para o caso de uso UniBot (Nível 1)	23
3.6	Vista física do sistema UnijobFormsBot (Nível 2)	24
3.7	Vista de implementação do sistema UnijobFormsBot (Nível 2)	25
3.8	Vista lógica do sistema UnijobFormsBot (Nível 2)	26
3.9	Vista de implementação do sistema UnijobFormsBot (Nível 3)	27
3.10	Vista lógica do sistema UnijobFormsBot (Nível 3)	28
3.11	Vista lógica do sistema UnijobFormsBot (Nível 4)	29
3.12	Vista de processos do caso de uso <i>UniScraper</i> do sistema UnijobFormsBot (Nível 4)	30
3.13	Vista de processos do caso de uso <i>UniBotUC</i> do sistema UnijobFormsBot (Nível 4)	31
4.1	Página do admin, <i>Uni Scraper</i> -	36
4.2	Resultados do Web Scraping.	38
4.3	Análise das ofertas recolhidas - Parte 1/3.	39
4.4	Análise das ofertas recolhidas - Parte 2/3.	39
4.5	Análise das ofertas recolhidas - Parte 3/3.	40
4.6	Formulário com caixa de texto pronta para comunicar com bot.	40
4.7	Formulário com caixa de texto pronta para comunicar com bot.	41
4.8	Formulário das empresas preeenchido (secção 1).	43
4.9	Formulário das empresas preeenchido (secção 2).	43
4.10	Formulário das empresas preeenchido (secção 3).	44
4.11	Formulário das empresas preeenchido (secção 4).	44
4.12	Cobertura total de testes (84%).	48

Lista de Tabelas

1.1	Planeamento Semanal do Estágio	6
1.2	Divisão dos sprints e respetivos objetivos	7
2.1	Comparação entre Websites Estáticos e Dinâmicos	10
2.2	Comparação entre Tecnologias de Web Scraping	15
2.3	Comparação entre Modelos de NLP com LLMs	15
3.1	Estrutura do modelo do formulário da oferta de emprego	33
5.1	Objetivos do projeto e respetivo estado de conclusão	50

Lista de Acrónimos

AI	Artificial Intelligence.
API	Application Programming Interface.
BERT	Bidirectional Encoder Representations from Transformers.
CPI	Consumer Price Index.
DIP	Dependency Inversion Principle.
DRY	Don't Repeat Yourself.
E2E	End-to-End.
GPT	Generative Pre-trained Transformer.
GRASP	General Responsibility Assignment Software Patterns.
HTML	HyperText Markup Language.
HTTP	HyperText Transfer Protocol.
ISEP	Instituto Superior de Engenharia do Porto.
JSON	JavaScript Object Notation.
LEI	Licenciatura em Engenharia Informática.
LLM	Large Language Model.
NLP	Natural Language Processing.
OCP	Open/Closed Principle.
OCR	Optical Character Recognition.
Q&A	Questions and Answers.
REST	Representational State Transfer.
RGPD	Regulamento Geral sobre a Proteção de Dados.
SRP	Single Responsibility Principle.
T5	Text-to-Text Transfer Transformer.

UC Use Case.

Capítulo 1

Introdução

O presente capítulo tem como objetivo introduzir e contextualizar o projeto estágio desenvolvido quer académicamente quer profissionalmente. Serão apresentados o contexto da plataforma **Unilinkr** [1] e o problema identificado, no qual será abordado objetivos, contributos e planeamento do trabalho. Também será apresentada a própria estrutura adotada para este relatório, permitindo ao leitor compreender o ponto de partida do trabalho, a sua relevância e motivações, tornando-se uma base sólida para os restantes capítulos.

O crescimento de integrações de ferramentas que recorrem a Artificial Intelligence (AI) em organizações e a própria familiaridade de utilizadores com estas tecnologias, leva muitas empresas a optarem pela integração destes sistemas nas suas plataformas. Desta forma, utilizadores podem ter muita mais interatividade com as funcionalidades existentes, permitindo auxílio durante vinte e quatro horas, através de sistemas conversacionais.

1.1 Enquadramento

O trabalho surgiu no âmbito da unidade curricular Projeto/Estágio (PESI), da Licenciatura em Engenharia Informática (LEI), do Instituto Superior de Engenharia do Porto (ISEP). O estágio desenvolvido teve como objetivo aplicar os conhecimentos adquiridos ao longo do curso, em contextos reais de mercado de trabalho.

Desta forma, a plataforma Unilinkr [1], orientada para a ligação entre estudantes e oportunidades profissionais, necessitou de otimizar o processamento de ofertas de emprego, tendo em conta o aumento de utilizadores. Criar mais ofertas de emprego é um processo custoso para as empresas, sendo possível melhorá-lo através de recolha de trabalhos semelhantes de outros websites (com Web Scraping), e preencher automaticamente formulários para novas ofertas, com recurso a sistemas Natural Language Processing (NLP). Desta forma, o processo de publicação de novos trabalhos automatiza-se, tornando-se eficiente, reduzindo assim o esforço humano.

Este projeto enquadra-se nas áreas de inteligência artificial e sistemas web, explorando domínios como NLP e Web Scraping.

Primeiro deus que se ka
a Unilinkr, só te no 17 a
plataf

1.2 Contexto

Em setembro de 2020, a Unilinkr [1] emergiu no mercado, tornando-se uma plataforma que tem como objetivo promover a autonomia financeira de estudantes do ensino superior, facilitando a ligação direta com o mundo de trabalho, podendo assim usufruir de variadas oportunidades com diferentes registos. Atualmente, a plataforma possui mais de 25.000 estudantes registados, juntamente com 150 empresas. A Unilinkr menciona que tem uma taxa de conversão de 95%, indicando que, para cada 100 empresas que publiquem uma oferta, 95 conseguirão recrutar estudantes. Este fator demonstra uma elevada eficácia na plataforma, na relação entre a oferta e a procura.

As empresas atuam com os estudantes, proporcionando serviços em áreas como restauração, apoio a eventos ou tarefas de cariz técnico e criativo. O projeto da Unilinkr também surge do facto: muitos estudantes ainda continuam a depender financeiramente de outrem. Isto pode afetar a sua autonomia e desenvolvimento pessoal e, por isso, com a Unilinkr, pretende-se valorizar a qualidade e capacidade de trabalho dos mesmos, integrando-os no mercado de trabalho, ao facilitar o acesso a experiências profissionais, durante a sua formação. [1]

Com cada vez mais estudantes a procurar trabalho e cada vez mais empresas a proporcionar essas atividades, a Unilinkr quer subir de nível e automatizar esse processo com tecnologias emergentes. Recentemente, o projeto "Linkup Thursdays" surgiu como incentivo para estudantes portugueses se reunirem, criando ligações com pessoas com os mesmos interesses e com o mesmo objetivo, tornando a experiência de aceitar ofertas da Unilinkr mais apelativa. Consequentemente, o número de utilizadores aumentará, permitindo um aumento de empresas a querer proporcionar essas experiências a estudantes.

Sendo assim, o presente projeto alinha-se com esta missão, ao procurar melhorar os mecanismos de publicação de ofertas de emprego, através da introdução de novas tecnologias que o automatizem e simplifiquem. A solução proposta visa melhorar a eficácia da plataforma, ampliando o número de ofertas disponíveis e, consequentemente, proporcionando mais alternativas para os estudantes.

Apesar de vantagens a nível curricular e profissional, a escolha deste projeto também foi influenciada por razões pessoais, visto o interesse nas áreas de AI e automatização de processos, aliada à relevância social da proposta. A oportunidade de desenvolver uma solução que combina novas tecnologias com impacto direto na vida de estudantes, tornou-se um fator determinante para a escolha deste desafio.

1.3 Descrição do Problema

A publicação de ofertas de emprego depende do preenchimento de um formulário correspondente aos detalhes necessários para publicar um trabalho. Este divide-se em vários campos que têm de ser obrigatoriamente preenchidos, de forma a criar uma oferta válida e compreensível para o estudante. O seu preenchimento é obrigatório para todas as ofertas publicadas, sendo por isso cansativo quando uma organização se propõe a publicar trabalhos frequentemente.

O administrador da plataforma e as empresas registadas, são os únicos utilizadores que têm permissões para publicar ofertas, tendo necessariamente de preencher o formulário previamente descrito. Contudo, estes podem demorar mais do que precisam e, consequentemente, desencorajar a publicação de ofertas.

b) tem que se preencher mais.

A crescer procura e que tem ao mesmo

Portanto, como a repetição contínua compromete a eficiência da plataforma e pode, ao longo do tempo, desencorajar a submissão de novas oportunidades, é necessário introduzir mecanismos que simplifiquem e acelerem este processo, recorrendo a tecnologias baseadas em AI, automatizando a criação de ofertas e melhorando a experiência do utilizador.

1.3.1 Objetivos

De acordo com o problema descrito, o projeto tem como objetivo automatizar e otimizar o processo de extração de ofertas de emprego utilizando tecnologias de Web Scraping e processamento de linguagem natural. Desta forma, o projeto está definido com objetivos capazes de definir um sistema que, ao receber informação direta vinda de uma descrição de emprego, quer enviada pelo utilizador, quer formatada de uma oferta de emprego *scraped* de um website, consiga extrair os dados necessários para preencher os campos do formulário.

Os objetivos específicos refletem-se em:

- Extrair automaticamente dados relevantes, partindo de descrições de emprego e recorrendo a técnicas de NLP;
- Implementar mecanismos de Web Scraping, capazes de recolher ofertas de emprego provenientes de fontes externas, semelhantes à plataforma Unilinkr;
- Estruturar os dados extraídos, com base nos campos exigidos pelo formulário da plataforma Unilinkr;
- Automatizar o preenchimento de formulário para novas ofertas de trabalho;
- Melhorar a eficiência do processo de publicação de ofertas.

A concretização destes objetivos permitirá otimizar o funcionamento interno da Unilinkr, ao facilitar e acelerar o processo de disponibilização de novas oportunidades de trabalho, garantindo assim a conclusão total do objetivo geral.

1.3.2 Abordagem

A abordagem adotada para o desenvolvimento do sistema foi suportada por uma metodologia ágil, mais concretamente Scrum, permitindo uma gestão iterativa e incremental das funcionalidades, tal como especificado na secção de planeamento de trabalho, do presente capítulo. Esta metodologia revelou-se adequada devido à natureza exploratória do projeto, onde requisitos podiam evoluir ao longo do tempo, sobretudo na integração de tecnologias como NLP (Processamento de Linguagem Natural) e Web Scraping. A estrutura adotada possibilitou a validação incremental das funcionalidades, reforçando a estabilidade e robustez da solução. Do ponto de vista técnico, a solução integra:

- Tecnologias de AI, como NLP, que permitem interpretar e extrair informação de descrições textuais de ofertas de emprego;
- Mecanismos de Web Scraping, que automatizam a recolha de dados relevantes a partir de websites externos;
- Validação de dados com regras de negócio e modelação, garantindo integridade e conformidade com os requisitos da plataforma Unilinkr.

A abordagem integrará tecnologias baseadas em AI, como processamento de linguagem natural, e tecnologias de extração de dados da web, juntamente com mecanismos de validação e

modelação de dados, de forma a garantir o sucesso do sistema. Para acrescentar estas novas ferramentas, recorreu-se à implementação de dois casos de uso:

- *UniBotUC* - publicar vagas com auxílio de linguagem natural (NLP), através da extração de uma descrição;
- *UniScraperUC* - extrair ofertas de empregos de websites (Web Scraping) - adicionalmente modelá-las utilizando *UniBotUC*.

NLPs e Web Scraping são técnicas fundamentais para a automatização de tarefas e a extração de conteúdo valioso, permitindo a interpretação de grandes volumes de textos, através de dados extraídos de sites ou de texto personalizado. A ferramenta de Web Scraping irá extrair detalhes relevantes sobre ofertas de emprego na web, especialmente de plataformas que também partilhem ofertas de emprego, semelhantes às oferecidas pela Unilinkr. O administrador terá uma funcionalidade na sua página que permitirá iterar sobre websites previamente definidos e obter o máximo de ofertas de emprego.

De seguida, de forma a integrar a ferramenta NLP, formata uma descrição bem definida como todos os dados extraídos da web. Esta ferramenta é a mesma que será utilizada pela página das empresas, e permite que, o utilizador envie apenas com uma descrição da oferta de emprego detalhada, para que todos os campos tenham possibilidade de serem preenchidos com informação válida.

Caso algum dado seja inválido, este será previamente validado com regras de negócio e, se necessário, com auxílio de NLP, antes de retornar ao utilizador. Esta situação implica que ambas as funcionalidades usem a mesma forma de análise dos dados e garantindo que dados inválidos sejam ignorados pelo sistema. Existem diferenças no formato do modelo dos formulários em ambas as páginas, sendo por isso necessário modelar de forma adequada e em concordância com o respetivo *frontend*.

Esta abordagem integrada, combinando técnicas de Web Scraping e NLP, permite automatizar de forma eficiente a recolha e o preenchimento de ofertas de emprego. A escolha das tecnologias utilizadas foi suportada por uma análise do estado da arte, privilegiando soluções robustas e amplamente adotadas na indústria. Assim, esta assegura uma experiência otimizada, única e interativa, tanto para os administradores como para as empresas. A implementação modular garante ainda uma fácil adaptação às necessidades da plataforma Unilinkr, promovendo: escalabilidade e reutilização futura das funcionalidades desenvolvidas.

1.3.3 Contributos

Este projeto resultou na conceção e desenvolvimento de um sistema funcional, combinando técnicas de Web Scraping, Processamento de Linguagem Natural (NLP) e mecanismos de validação de dados, os quais foram aplicados ao domínio das ofertas de emprego. Desta forma, destacam-se os seguintes outputs produzidos:

- Um módulo de Web Scraping capaz de recolher ofertas de websites externos automaticamente;
- Uma ferramenta NLP, com Large Language Model (LLM), que interpreta descrições textuais e recolhe automaticamente campos estruturados num formulário;
- Um mecanismo de validação e formatação de dados, garantindo a conformidade com as regras de negócio;

- Integração de ambos os módulos num sistema da Unilinkr, com interfaces distintas para administradores e empresas;
- A documentação do sistema, a qual inclui requisitos, arquitetura e testes.

A implementação deste sistema pode ser impactante, nos mais variados contextos, como por exemplo para a plataforma Unilinkr, pois melhorará a experiência dos utilizadores e trará novas oportunidades para a plataforma; para a comunidade académica e tecnológica, devido à demonstração de um exemplo prático que combine Web Scraping e NLPs, e a nível pessoal e profissional, porque permite consolidar matérias sobre áreas de AI, ao mesmo tempo que impacta positivamente a atual sociedade.

Sendo assim, a implementação deste sistema pode contribuir:

Para a plataforma Unilinkr:

- Reduzindo do esforço manual na criação de ofertas;
- Melhorando na experiência do utilizador para empresas e administrador;
- Potencial aumento de atividade na plataforma para novos utilizadores.

Para a comunidade académica e tecnológica:

- Demonstração de um sistema de integração entre componentes de AI e mecanismos Web Scraping;
- Estudo da eficácia de técnicas de *prompt engineering* na extração e classificação automática de dados a partir de linguagem natural;

A nível pessoal e profissional:

- Consolidação de competências em desenvolvimento de soluções baseadas em AI;
- Experiência prática na resolução de problemas reais, com impacto social direto.

1.3.4 Planeamento do Trabalho

A decisão de optar por uma boa organização do trabalho, permitiu atingir resultados positivos, devido ao estabelecimento de objetivos concretos para realizar semanalmente, concluindo metas gerais a longo prazo. Desta forma, o trabalho foi dividido sequencialmente nas seguintes fase (*milestones*):

- **Introdução à plataforma, pesquisa e revisão da literatura**

Da 1ª à 5ª semana, antes do começo da exploração e desenvolvimento do sistema, foi possível interagir e ajudar com a plataforma, resolvendo tarefas relacionadas com tarefas que tinham poucos *storypoints* (forma de representação temporal), de forma a garantir um primeiro contacto com a plataforma, obrigando a um conhecimento geral do sistema. Esta fase também teve como objetivo compreender o estado da arte nas áreas de NLP, Web Scraping, *prompt engineering* e integração com sistemas web. Permitiram ainda definir as ferramentas e bibliotecas a utilizar, iniciando a fase seguinte com o pé direito.

- **Desenvolvimento**

Até à 9ª semana deu-se a fase de desenvolvimento do sistema, interagindo com as funcionalidades principais do sistema, incluindo os módulos de extração por NLP, formas de Web Scraping e sistema de validação e modelação dos dados.

- **Integração e testes**

Da 10ª semana à 14ª semana, a base do sistema desenvolvido foi integrado com a plataforma Unilinkr, garantindo que o teste das funcionalidades através de testes unitários, testes de integração. Foram realizados testes com dados reais, de modo a validar o comportamento do sistema e a qualidade dos dados extraídos. Nesta fase também se desenvolveu o relatório da unidade curricular.

- **Demonstração**

Na última semana demonstrou-se o projeto concluído ao resto da empresa, recebendo feedback sobre o mesmo.

O planeamento completo poderá ser consultado na tabela 1.1.

Semana	Actividades
03/03 – 09/03	Configuração do ambiente e ferramentas; introdução à plataforma.
10/03 – 16/03	Primeira tarefa na plataforma; início da pesquisa e planeamento.
17/03 – 23/03	Continuação da primeira tarefa e investigação inicial.
24/03 – 30/03	Redação da Revisão da Literatura.
31/03 – 06/04	Conclusão da Revisão da Literatura.
07/04 – 13/04	Início do desenvolvimento.
14/04 – 20/04	Desenvolvimento.
21/04 – 27/04	Desenvolvimento.
28/04 – 04/05	Desenvolvimento.
05/05 – 11/05	Início da Integração com o sistema.
12/05 – 18/05	Integração com o sistema.
19/05 – 25/05	Integração com o sistema.
26/05 – 01/06	Início dos Testes e do relatório.
02/06 – 08/06	Testes e relatório.
09/06 – 16/06	Testes finais, demonstração e conclusão do relatório.

Tabela 1.1: Planeamento Semanal do Estágio

As fases correspondentes ao desenvolvimento, integração e testes do sistema foram organizadas, segundo uma metodologia SCRUM, a qual pode ser consultada na tabela 1.2. Esta consiste numa estrutura ágil para o desenvolvimento de projetos, principalmente em áreas de software. Neste contexto, o trabalho será dividido em ciclos curtos e fixos, denominados sprints, com duração entre uma a quatro semanas. Em cada sprint, são definidas metas específicas que deverão ser alcançadas até ao final do respetivo ciclo.

Foram definidas **7** user stories para serem implementadas:

- **US1** - Desenvolver um sistema NLP com LLM, capaz de extrair os campos esperados do formulário;

- **US2** - Desenvolver um sistema Web Scraping, capaz de extrair ofertas de emprego da web;
- **US3** - Modelar a estrutura dos objetos de acordo com o formulário;
- **US4** — ? ?
- **US5** - Desenvolver uma página na plataforma, para o utilizador Empresas enviar descrições personalizadas para o sistema;
- **US6** - Desenvolver uma página na plataforma, para o Administrador recolher ofertas externas;
- **US7** - Desenvolver uma base robusta para o sistema através de testes.

As user stories **US1**, **US4** e **US6** são referentes ao caso de uso *UniBotUC*, enquanto que **US2**, **US3**, **US5** e **US7** são referentes ao caso de uso *UniScraperUC*. Deste modo, estas tarefas foram agrupadas em 4 sprints, tal como representado na tabela 1.2.

Sprint	Semanas	Duração	Objetivos
1º Sprint	Semana 6 a 8	3 semanas	US1
2º Sprint	Semana 9 a 10	2 semanas	US2 e US3
3º Sprint	Semana 11 a 13	3 semanas	US4 e US5
4º Sprint	Semana 14 a 15	2 semanas	US6 e US7

Tabela 1.2: Divisão dos sprints e respetivos objetivos

1.4 Estrutura do Relatório

Para além deste capítulo introdutório, que contextualiza e aborda de forma geral o problema a desenvolver, esta dissertação contém mais **4 capítulos**, organizados de forma a refletir a evolução lógica do trabalho:

- **2 - Estado da Arte:** Descreve o panorama atual das tecnologias relevantes para o projeto, incluindo técnicas de processamento de linguagem natural, Web Scraping, *prompt engineering* e integração de sistemas. Inclui ainda uma análise de trabalhos relacionados.
- **3 - Análise e Desenho:** Apresenta os requisitos do sistema, os objetivos funcionais e não funcionais, bem como as decisões de design iniciais.
- **4 - Implementação:** Detalha o processo de desenvolvimento do sistema, abordando a arquitetura da solução, principais módulos implementados e decisões técnicas.
- **5 - Conclusões:** Apresenta as conclusões gerais do trabalho, as limitações encontradas e sugestões para desenvolvimentos futuros.

Desta forma, esta organização visa facilitar a leitura e compreensão de todas as etapas do projeto, desde a conceção à implementação final. O capítulo 2 e o capítulo 3 foram significativos para selecionar quais tecnologias utilizar e como organizar o projeto, na fase de implementação, especificada no capítulo 4. Por fim, o último capítulo, 5, baseia-se numa análise final do sistema, comparandos os objetivos definidos com os resultados obtidos.

Capítulo 2

Estado da Arte

Nesta secção são explorados os principais conceitos, ferramentas e trabalhos existentes no âmbito dos sistemas de chatbot inteligentes, do processamento de linguagem natural (NLP) e do Web Scraping. Estes servem de base teórica para o desenvolvimento do presente projeto, retirando conclusões sobre quais tecnologias a seleccionar, com uma justificação alinhada com os conteúdos existentes.

2.1 Trabalhos relacionados

A secção contém informação sobre sistemas Web Scraping, outrora implementados, com tecnologias emergentes, e sistemas NLP com recurso a LLM, com modelos de linguagem natural bastante desenvolvidos e com capacidades extraordinárias.

2.1.1 Web Crawler & Web Scraping

Na década de 90, após a criação da World Wide Web, surgiu o *Jump Station*, a primeira ferramenta capaz de extrair dados da web, através de uma técnica de rastreamento rudimentar. A tecnologia Web Crawler, que surgiu nessa época, permitia navegar automaticamente entre páginas na web, tendo sido utilizadas por motores de busca para atualizar as suas bases de dados, garantindo assim permitindo pesquisas mais profundas. A Google implementou esse sistema para aprimorar a pesquisa de informação, adicionando esses dados "ao banco de dados do mecanismo de pesquisa"[2]. Estes utilizam a ferramenta para percorrer a web, seguindo links da mesma forma que um utilizador, tal como referido no estudo "*Study of Web Crawler and its Different Types*", publicado no *Journal of Computer Engineering* [3]. Este aborda diferentes tipos de *web crawlers*, destacando as suas características, vantagens e também casos de uso. O artigo realça a importância destas ferramentas em processos como extração automatizada de dados e navegação eficiente na web. Os modelos descritos pelos autores foram "Focused Crawler", "Incremental Crawler", "Distributed Crawler" e "Parallel Crawler", apresentando abordagens adaptadas a diversos contextos, variando consoante os objetivos da recolha e a própria arquitetura do sistema.

- **Focused Crawler**

Tenta dar *download* a páginas web que estejam relacionadas, ou seja, que a sua informação tenha semelhanças, significando uma ligação forte entre ambas [3]. Domínios restritos como por exemplo webistes com informações sobre doenças raras, na área da bionformática, podem ser analisados para recolher informações precisas através de termos específicos como "*rare disease*", "*genetic mutation*", "*clinical trials*". Esta técnica

pode ser aplicada a variadas áreas com websites com esta estrutura, tal como mercado financeiro ou turismo.

- ***Incremental Crawler***

Tradicionalmente, crawlers atualizam continuamente páginas existentes, baseando-se na frequência das mudanças, substituindo documentos antigos com novos documentos [3]. Contudo, crawlers incrementais monitorizam constantemente secções como "Últimas Notícias", por isso, um crawler poderia analisar essa secção num website como a RTP, que possui páginas que representam as últimas notícias tal como aconteceu durante o Covid-19 [4] ou, mais atualmente, referente ao conflito entre Ucrânia e a Rússia [5]. Este analisaria frequentemente novas notícias, permitindo redirecionar para a mesma, sem necessidade de recorrer a uma indexação completa do site a cada iteração e apenas descarregar os novos conteúdos recolhidos online.

- ***Distributed Crawler***

Um servidor central controla múltiplos crawlers, com o objetivo de percorrer o máximo da web. Esta técnica é vantajosa pois é robusta contra *crashes* no sistema, podendo ser reconfigurada dinamicamente, redistribuindo tarefas entre crawlers, sem comprometer a operação. [3].

- ***Parallel Crawler***

Múltiplas instâncias de rastreamento operam simultaneamente, quer localmente, quer remotamente. Um crawler paralelo consiste em múltiplos processos de crawlers, sendo viável para *downloading* ficheiros num tempo razoável [3]. Crawlers locais podem ser utilizados na mesma máquina ou em múltiplas que partilhem a rede mesma rede local. Desta forma, podem partilhar recursos físicos e comunicar entre si. Crawlers remotos representam-se distribuídos geograficamente, ou em diferentes redes, via internet. Apesar de serem mais complexos de gerir, estes permitem analisar rapidamente a web e comunicar entre redes públicas.

A evolução rápida das tecnologias digitais, levou ao aumento exponencial da criação de dados diariamente na web, o que levou a uma nova necessidade: obter rapidamente conteúdo específico de um website. Desta forma, foi possível surgir com uma funcionalidade que poderia elevar web crawling, Web Scraping. Esta técnica opera sobre exploradores web, analisando o conteúdo real do website, extraíndo todos os dados em puro HTML. Desta forma, consolida a informação necessária num só ponto, recolhendo as informações requeridas, seguindo padrões HyperText Markup Language (HTML) definidos para cada website.

Inicialmente os websites seguiam padrões estáticos, o que significa que os dados disponíveis são fixos. Caso surjam novas alterações, é necessário lançar uma nova versão para a web. Desta forma, nos anos 2000, Beautiful Soup emerge como uma ferramenta que permitia realizar web crawling num website, extraíndo informação específica de acordo com os seus padrões - Web Scraping [6]. Esta biblioteca Python facilita parsing de HTML/XML, tornando-se um marca para a evolução do Web Scraping.

O surgimento de websites dinâmicos, permitiu elevar a web, devido à sua constante alteração do conteúdo, sendo gerado em tempo real. Esta evolução melhorou significativamente a experiência dos utilizadores, mas impôs novos desafios à extração automática de dados, obrigando ao surgimento de uma nova estratégia. Nos anos 2010, surgiram técnicas de web-scraping dinâmico, como Selenium [7] e Puppeteer [8]. Estas ferramentas possibilitam a simulação de interações humanas com páginas web — como *clicks*, *scrolls* e preenchimento de

formulários. Desta forma, permite-se o acesso a conteúdos carregados assíncronamente, via JavaScript. Apesar da sua eficácia, estas soluções apresentam limitações como elevados custos computacionais, maior tempo de execução e dificuldades na manutenção de scripts, face a alterações frequentes na estrutura das páginas. A tabela 2.1 estrutura uma comparação entre os dois tipos de websites.

Característica	Website Estático	Website Dinâmico
Velocidade	Rápido; Conteúdo pré-gerado.	Pode ser mais lento; Conteúdo gerado em tempo real
Segurança	Altamente seguro; Há menos vulnerabilidades, pois não há base de dados.	Menos seguro; Há mais vulnerabilidades a ataques.
Custo	Geralmente mais baixo; Requer menos recursos de servidor.	Mais alto; Exige servidores mais robustos e manutenção constante.

Tabela 2.1: Comparação entre Websites Estáticos e Dinâmicos

Um artigo publicado na *Oxford Academic -Web scraping technologies in an API world* [9] abordou a integração de Web Scraping, especialmente em contextos onde não existem Application Programming Interface (API) públicas que armazenam os dados requeridos para determinada operação. Os autores argumentam que API são atualmente o meio preferencial de acesso automático a dados, contudo o *scraping* continua a ser uma solução válida para recolher informações relevantes de websites, que não oferecem interfaces programáticas. Através de ferramentas específicas, é possível construir sistemas eficientes de extração de dados e com reduzido esforço de programação. Na área da bioinformática, existe uma parte significativa de dados relevantes que não estão acessíveis através de API públicas. Muitos repositórios de dados, focam-se mais na interação humana, do que na interoperabilidade de sistemas. Também é um facto que variados registos estão dispersos por inúmeros websites, dificultando a análise dos mesmos. Devido a esta descentralização, meta-serviços bioinformáticos como *WhichGenes* e *PathJam* foram desenvolvidos para copular informações relevantes que permitem análises mais extensas e acertadas.

A mesma plataforma aborda outro estudo que permite construir modelos estatísticos atualizados e fiáveis para acompanhar flutuações de preços, inflação e comportamento do consumidor - *Tracking and Modelling Prices Using Web-Scraped Price Microdata: Towards Automated Daily Consumer Price Index Forecasting* [[oxforacademic_web scraping_prices](#)]. Neste estudo, os autores abordam a importância e a disponibilidade de preços especificados on-line, e como Web Scraping é uma abordagem capaz de obter esses dados para, posteriormente, se prever o índice de preços ao consumidor (Consumer Price Index (CPI)). Além de ser um exemplo de extração massiva de informação da web, reflete uma automatização do processo de recolha de dados e atualização de modelos com frequência, consequentemente reduzindo o custo comparado a métodos tradicionais de recolha manual. O estudo mostra que o uso sistemático de microdados de preços recolhidos por Web Scraping é viável e eficaz para modelação estatística robusta. Oferece uma nova via para monitorizar a economia em tempo real, com custos reduzidos e maior detalhe analítico.

Devido ao crescimento da complexidade das páginas web modernas e o aumento do controlo legal dos dados dos utilizadores, é imperativo manter uma postura ética ao aplicar técnicas de Web Scraping. A sua legalidade tem sido questionada e debatida intensamente, particularmente após o caso entre "hiQ Labs" e "LinkedIn" [10]. A plataforma hiQ Labs recolhia dados públicos de perfis de utilizadores da LinkedIn, com o propósito de oferecer análises preditivas a empresas, contudo foi iniciado um processo contra a plataforma. Os tribunais norte americanos, concluíram que *scraping* de informação pública não é ilegal, limitando-se apenas a dados que não estejam acessíveis ao público sem login. A nível europeu, o protocolo GDPR impõe restrições adicionais, considerando que dados considerados pessoais não podem ser recolhidos - mesmo que expostos publicamente [11].

2.1.2 Chatbots

O ponto de partida teórico para o desenvolvimento de chatbots, remonta a 1950, quando Alan Turing, um famoso matemático e cientista de computação britânico [12], propôs uma alternativa para abordar a questão "As máquinas pensam?". Esta teve uma implementação prática, o famoso Teste de Turing [13], propondo o "Jogo da Imitação" para avaliar se uma máquina consegue parecer humana numa conversa escrita. Neste jogo, um avaliador humano comunica por escrito com dois interlocutores ocultos - um humano e uma máquina. Caso o avaliador não distinga consistentemente qual dos dois é o humano, então considera-se que a máquina demonstrou um comportamento linguístico indistinguível do humano, passando no teste. As capacidades reduzidas desta implementação, que apenas imitava e não representava a compreensão real do texto, levaram ao surgimento do primeiro chatbot, "ELIZA" [14], em 1966. Criado por Joseph Weizenbaum no MIT, foi possível simular empatia ou racionalidade humana, correspondendo a padrões, ainda que de forma superficial.

Com a popularização da internet durante os anos 2000, começaram a surgir assistentes virtuais e *webchatbots*, em diversas plataformas, para responder a FAQs ou dar suporte inteligente. Alguns destes sistemas, começaram a integrar funcionalidades básicas de aprendizagem automática, sendo escaláveis. A sua recente evolução, registou um aumento significativo na utilização de sistemas de chatbot baseados em inteligência artificial. Plataformas como o Facebook [15], registaram um aumento exponencial de contas controladas por bots, atingindo um total de 400.000 contas falsas em 2018 [16]. Este fator reforça o progresso e normalização desta área na nossa sociedade. A capacidade destas tecnologias compreenderem e replicarem comportamento humano pode e foi explorada, para que funcionalidades que necessitem de interação humana. Em contextos de recrutamento, pode-se usufruir destas ferramentas para automatizar a comunicação com candidatos e facilitar tarefas como triagem, marcação de entrevistas e recolha de dados estruturados, como por exemplo, formulários.

A empresa Mya Systems, desenvolveu um assistente conversacional capaz de gerir processos de recrutamento desde o primeiro contacto até à entrevista final, reduzindo o tempo de recrutamento em cerca de 38% [17]. Esta abordagem permitiu aumentar a eficiência sem comprometer a experiência do utilizador. Também, o chatbot *Olivia*, desenvolvido pela empresa Paradox, oferece funcionalidades como marcação automática de entrevistas e respostas em tempo real aos candidatos, demonstrando a utilidade de assistentes inteligentes na automatização de tarefas [18].

A Taylor & Francis publicou o estudo "*Chatbots for future docs: exploring medical students' attitudes and knowledge towards artificial intelligence and medical chatbots*" [19], que analisa as perceções de estudantes de medicina face à utilização de chatbots de AI na saúde.

Os resultados demonstram elevada aceitação em tarefas de administração e pesquisa, embora persistam preocupações com a segurança dos dados e a vigilância acadêmica. Apesar disso, os participantes reconheceram o potencial dos chatbots para o aumento da eficiência dos sistemas, poupando tempo e recursos, especialmente em contextos relacionados com cuidados clínicos. Este tipo de percepção positiva tem incentivado o desenvolvimento de aplicações práticas, utilizando modelos de linguagem. O artigo publicado na "National Library of Medicine"[20], aborda setores como saúde e ensino, que necessitam de extração de dados, preenchimento de formulários ou mesmo processamento de documentos, retiram vantagens perante alternativas manuais.

A implementação de uma solução prática, em Python e NLP com sistema LLM, através da extração de texto de fotos de recibos, é: "*Building a Receipt Text Extraction App with Python and LLM: Unleashing the Power of LLMs*". Neste, a extração de texto de imagens (recibos), com auxílio de ferramentas de Optical Character Recognition (OCR) como Tesseract [21] ou PaddleOCR [22], ocorre com auxílio de LLM. Inicialmente, há um pré-processamento da imagem que permite definir o texto do recibo, garantindo a extração textual com OCR. Após isso, envia-se uma entrada com esse texto para um LLM, o qual retorna uma estrutura JavaScript Object Notation (JSON) [23].

Este formato garante rápida prototipagem e versatilidade entre recibos, não sendo necessárias regras regex rígidas. Por outro lado, além do custo de chamadas a LLM, podem ocorrer "alucinações", ou seja, o LLM retornar informação falsa. Assim, estes exemplos demonstram o potencial da utilização de modelos de linguagem para aumentar a eficiência em processos de comunicação e extração de informação.

2.2 Tecnologias Existentes

A crescente necessidade de recolher dados da web, para análise e integração, em sistemas inteligentes, impulsionou o desenvolvimento de diversas ferramentas e bibliotecas orientadas ao Web Scraping. Estas permitem automatizar o acesso e a extração de grandes volumes de informação, presente em páginas web, refletindo-se numa técnica de extração automática de dados de websites, sendo por isso fundamental para a recolha de grandes volumes de informação não estruturada. Ferramentas como BeautifulSoup em Python e Cheerio em JavaScript são eficazes na extração de conteúdo de páginas estáticas. Já ferramentas como Puppeteer e Playwright são preferidas para *scraping* de websites dinâmicos, permitindo simular interações humanas com JavaScript.

Projetos desenvolvidos em Python utilizam bibliotecas como BeautifulSoup, para websites estáticos, e Scrapy e Selenium para websites dinâmicos. Adicionalmente, requests permite processar pedidos HyperText Transfer Protocol (HTTP) e pandas limpar e exportar dados [**scraping_tecnologies**]. Desta forma, com apenas uma tecnologia, podemos proporcionar a comunicação com o exterior, juntamente com o processo de *scraping*.

Contudo, devido à capacidade desta técnica, utilizadores mal-intencionados podem *scrapar* websites que possuam informações pessoais relevantes, quebrando os padrões Regulamento Geral sobre a Proteção de Dados (RGPD). Por isso, é essencial proteger plataformas na web com mecanismos anti-*scraping*. Desta forma, o processo de navegação torna-se restrito, podendo ser ultrapassado, recorrendo a estratégias como:

- **Proxies rotativos**, que alteram o IP entre requisições para evitar bloqueios baseados em IP;

- **User-agents dinâmicos**, que simulam navegadores reais, dificultando a detecção de robôs;
- **Delays aleatórios**, que ajudam a imitar o comportamento humano na navegação.

Estes métodos aumentam a eficácia da recolha de dados sem violar políticas de acesso. O uso ético de Web Scraping tem sido defendido por vários autores, destacando a importância de respeitar as normas legais e de uso de cada website [9].

Na mesma medida, tecnologias emergentes no âmbito de sistemas NLP, como LLM, surgiram como um ramo da inteligência artificial que tem como objetivo permitir que máquinas compreendam, processem e interajam com a linguagem humana. O artigo "Deep Learning", da *Scholarpedia* [24], aborda o tema do Deep Learning e em como se inspira na estrutura do cérebro humano. Este subcampo da inteligência artificial é capaz de processar dados através de múltiplas camadas de redes neuronais - tal como no ser humano. De acordo com Schmidhuber [24], esta abordagem permite ao sistema aprender representações progressivamente mais abstratas, reduzindo assim a necessidade de engenharia manual de características. O crescimento do seu poder computacional e a disponibilidade para processar grandes volumes de dados, tornaram-se fatores essenciais para o sucesso do *deep learning*. Áreas como visão computacional, reconhecimento de fala e processamento de linguagem natural, têm evoluído significativamente, impulsionadas por modelos como **Bidirectional Encoder Representations from Transformers (BERT)**, **Generative Pre-trained Transformer (GPT)** e **Text-to-Text Transfer Transformer (T5)**. Cada um deles oferece características distintas e que os tornam úteis em diferentes contextos.

- BERT é um modelo bidirecional que considera o contexto à esquerda e à direita de cada palavra, tornando-o eficaz em tarefas como classificação de texto ou resposta a perguntas, embora não seja adequado para geração de texto [25].

Como abordado por Ian Tenney, Dipanjan Das e Ellie Pavlick, em "*BERT Rediscovered the Classical NLP Pipeline*", Frequentemente, o modelo "ajusta a pipeline dinamicamente", alterando decisões de nível inferior, se as camadas superiores surgirem com nova informação [25].

- GPT, por outro lado, é um modelo auto-regressivo capaz de gerar texto coerente, e tem sido amplamente usado em sistemas de conversação como os chatbots [brown2020]. A sua capacidade de criação textual torna-o ideal para responder a utilizadores em linguagem natural, embora possa ocasionalmente gerar conteúdo impreciso.
- T5 propõe uma abordagem unificada, convertendo qualquer tarefa de NLP num problema de tradução de texto para texto, o que o torna altamente versátil [26].

Modelos LLMs, como **GPT-3** (OpenAI), **LLaMa** (Meta) e **Gemini** (Google), destacam-se pelas suas capacidades emergentes: geração contextual, raciocínio lógico e adaptação sem necessidade de re-treino. Estas características tornam-nos adequados para aplicações interativas e personalizadas. Atualmente, como referido por Mia Zhang e Juntao Li, na *ScienceDirect*, GPT-3 revelou-se como um marco para a escalabilidade de LLMs, sendo o modelo de linguagem com maior número de palavras até à data. Possui elevada geração de texto contextual, manipulação de diálogos e capacidade de compreensão textual [27].

Um artigo da *Springer Nature Link* aborda as capacidades de LLaMa, referindo que apresenta uma implementação poderosa e eficiente, mas reflete limitações em tarefas específicas. Os autores reforçam a ideia de tanto direcionar como melhorar resolução de problemas, por

Diferença
entre GPT-3 e LLaMa?

exemplo com tarefas concretas de programação. [28]. Esta é uma funcionalidade *open-source*, permitindo ser utilizada por todos.

Quanto ao Gemini, a Google afirma que, num ano, a quantidade de aplicações que o utilizam aumentou de 101 para 601, demonstrando ser uma tecnologia que deixou uma era experimental e entrou em produção de larga escala. Devido à elevada capacidade de reproduzir tarefas distintas, Gemini tem-se mostrado eficaz em diversos contextos [29], revelando ser uma ferramenta completa.

No entanto, apesar desta versatilidade, continua a ser complexo adaptar soluções pré-treinadas a dados muito específicos ou domínios fechados. Por isso, na sua adoção, ocorre frequentemente um processo de afinação (*fine-tuning*) ou integração com sistemas de regras e bases de conhecimento especializadas. Contudo, esta estratégia pode aumentar a complexidade técnica e os custos associados. Ainda assim, o crescimento acelerado da sua implementação confirma o papel emergente da AI generativa, posicionando-se como o elemento central em sistemas de automatização, análise de dados e interação inteligente com utilizadores.

No artigo "A Text Mining using Web Scraping for Meaningful Insights"[30], os autores tinham como objetivo desenvolver uma ferramenta automática para resumo de texto, com base numa metodologia extrativa. Esta resume-se a uma mineração de texto baseado em Web Scraping de websites de notícias e redes sociais. A extração de dados textuais de plataformas online, tinha como objetivo identificar padrões, tendências e opiniões relevantes, sendo necessário realçar o papel crucial de limpar e unificar dados textuais, para posteriormente serem analisados. Quando integrados, NLPs e web Scraping formam uma combinação poderosa: o *scraping* obtém os dados e a linguagem natural interpreta, classifica e estrutura essa informação. Desta forma, conseguimos automaticamente recolher informação da web, processando-a e obtendo resultados precisos para as indicações providenciadas. Sendo assim, esta complementaridade comprova que estas soluções vão além de conceitos teóricos, revelando uma aplicação real de métodos *scraping* combinados com NLPs.

Em conclusão, sistemas automáticos como o descrito em 1.2, podem usufruir destas tecnologias emergentes, de forma a atingir objetivos como o que se pretende desenvolver.

2.3 Discussão e Seleção

Ao analisar as técnicas abordadas, concluiu-se que Python era a linguagem de programação que mais se adequava, tendo em conta a quantidade de bibliotecas existentes para Web Scraping e no suporte de ferramentas AI. No sentido do Web Scraping, tecnologias como selenium, BeautifulSoup e tenacity tornaram-se apelativas devido às suas capacidades. Estas permitem interagir com a web, extrair dados e garantir ações como repetir ações e forçar esperas, respetivamente. O contacto entre utilizador e o servidor acontecerá com HTTP *requests*, por isso, uma Representational State Transfer (REST) API controlará essa parte. Python possui a biblioteca FastAPI, que permite gerir os *requests*.

Relativamente às tecnologias para implementação do sistema de Web Scraping, e tendo em conta os requisitos do projeto, BeautifulSoup revelou-se a escolha para websites estáticos e Selenium para dinâmicos. A tabela 2.2 demonstra a comparação geral entre as tecnologias existentes.

Cliente?

Tabela 2.2: Comparação entre Tecnologias de Web Scraping

Tecnologia	Estático	Dinâmico	HTTP Requests
BeautifulSoup	X		
Cheerio	X		
Puppeteer		X	
Playwright		X	
Scrapy	X		X
Selenium		X	X

A utilização de agentes chatbots poderosos e acessíveis, foi selecionada versão 3.1 do chatbot do LLaMa, o 'Ollama'. Este foi escolhido como AI devido a ser uma alternativa *open-source* e que permite utilizar variados modelos, tendo respostas adequadas para tarefas que exijam uma resposta precisa. Adicionalmente, estes dados pessoais não são partilhados com *clouds* nem com servidores de terceiros, garantindo segurança para o utilizador. A alternativa permite o seu uso por aplicações web e Python scripts, tornando-se assim uma escolha apropriada para o projeto. Numa versão anterior, ter-se-á ponderado criar um modelo AI, treiná-lo para compreender o formato dos dados, tal como perceber padrões para o preenchimento adequado dos campos, e assim preencher acertadamente, utilizando exemplos reais. Contudo, o custo temporal revelou-se extensivo, chegando a demorar mais de uma hora para treinar um conjunto de 120 exemplos. De facto, quanta mais informação fornecida, maior será o tempo de processamento, só que essa quantidade de exemplos é bastante reduzida, tendo em conta a larga estrutura da plataforma, descartando por fim essa opção e mantendo a decisão inicial.

A tabela 2.3 compara os modelos NLP com LLMs abordados, em cinco critérios principais: capacidade de classificação de texto, geração de texto coerente e contextualizado, respostas a perguntas (Questions and Answers (Q&A)), versatilidade em diferentes domínios e tarefas, e se o modelo é *open-source*. Esta comparação garante flexibilidade de integração e maior controlo sobre os dados, fundamentando assim a escolha do modelo mais adequado para as necessidades do sistema, com especial atenção à privacidade, capacidade de adaptação e facilidade de implementação.

Tabela 2.3: Comparação entre Modelos de NLP com LLMs

Modelo	Classificação	Geração de Texto	Q&A	Versatilidade	Open-Source
GPT-3		X	X	X	
LLaMa	X	X	X		X
Gemini	X	X	X	X	

Em suma, a seleção tecnológica adotada reflete um equilíbrio entre robustez, acessibilidade e adequação, tendo em conta o contexto do projeto. A escolha de Python devido às suas bibliotecas para Web Scraping, garante bastante flexibilidade e eficácia na recolha dos dados, enquanto garante uma postura ética. Também, a seleção do modelo LLaMa, através de Ollama, garante abordar NLPs, principalmente LLMs. Esta combinação poderosa, assegura uma base sólida para o sucesso do projeto, garantindo eficiência, adaptabilidade e segurança.

Capítulo 3

Análise e Desenho da Solução

A análise e desenho do sistema permitem estruturar e compreender o seu desenvolvimento, partindo de um ponto de vista mais abrangente, até atingir as especificações mais detalhadas da solução.

3.1 Domínio do Problema

O sistema compreenderá uma conexão do utilizador, que pode ser o administrador ou uma empresa, com a solução a determinar. A plataforma Unilinkr integra um agregado de utilizadores, que vão interagir diretamente com serviços do UniJobFormsBot, para que assim comuniquem com as funcionalidades estabelecidas, tal como representado na figura 3.1.

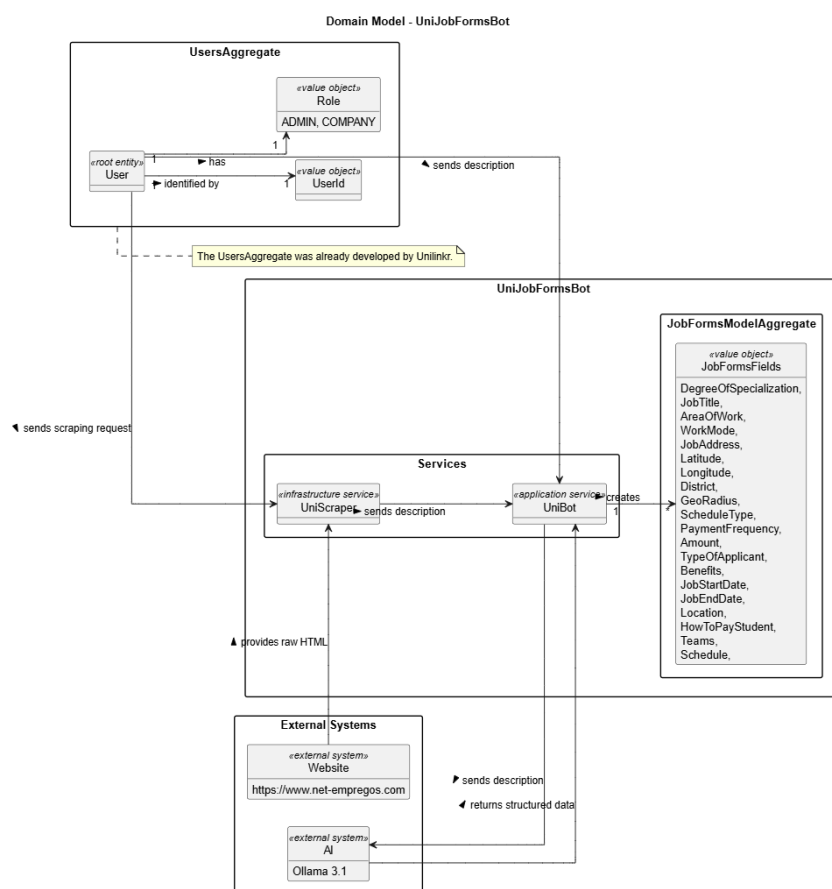


Figura 3.1: Modelo de domínio do sistema UniJobFormsBot

O agregado UserAggregate está representado, dado que o utilizador é quem interagirá com o sistema. Este pode ser do tipo "ADMIN" ou "COMPANY", e poderão interagir com o sistema e automatizar a publicação de novas ofertas para estudantes. Os serviços representados no agregado External Systems são aqueles com os quais o sistema interagirá: Websites e AI API da Ollama3.1; os quais têm duas responsabilidades essenciais:

- Comunicar com websites externos;
- Comunicar com AI API externa.

Por fim, os agregados Services e JobFormsModelAggregate representam os componentes da implementação de UnijobFormsBot. O agregado Services compreenderá os componentes que controlarão os dois casos de uso e o agregado JobFormsModelAggregate representa o modelo do formulário a ser preenchido, garantindo que os resultados obtidos ficarão disponíveis para o utilizador, para que sejam utilizados de livre vontade.

3.2 Requisitos

A definição de requisitos numa análise inicial, permite estruturar o projeto de forma a atingir o seu objetivo, evitando assim, ambiguidade ao longo do desenvolvimento. Desta forma, é possível priorizar tarefas, garantindo que funcionalidades implementadas estão alinhadas com as necessidades da plataforma. Assim, no final do projeto, é possível verificar quais objetivos foram cumpridos, quer funcionalmente, quer qualitativamente.

3.2.1 Requisitos Funcionais

Os requisitos funcionais são essenciais para definir as funcionalidades concretas a desenvolver. No contexto deste projeto, estes devem determinar as ações necessárias que o sistema deve ser capaz de realizar: extrair informação via NLP e recolha automática de dados, através de Web Scraping. A sua descrição objetiva, permite planear o sistema de forma estruturada. Estas funcionalidades devem ser testadas no final do projeto, funcionando como critério de avaliação.

- **RF1:** Integração de um sistema baseado em NLP que seja capaz de extrair informação a partir de descrições de ofertas e preencher automaticamente os campos do formulário de submissão da Unilinkr.
- **RF2:** Desenvolvimento de um sistema de Web Scraping que recolha ofertas de emprego publicadas em websites externos, obtendo o máximo de informação útil para preencher o mesmo formulário.

O primeiro requisito será realizado por utilizadores "Empresa" e o segundo pelo Administrador. Este requisito, implica a realização do primeiro para conseguir efetuar a extração das ofertas de emprego com preenchimento do formulário, tal como representados na figura 3.2.

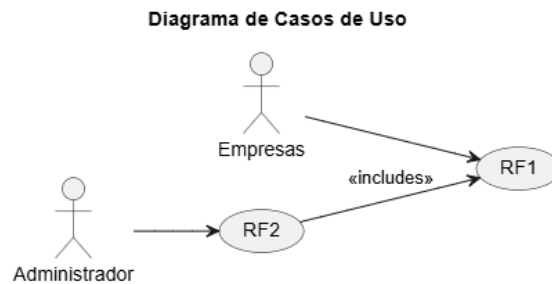


Figura 3.2: Diagrama de Casos de Uso

Ambos os requisitos têm como objetivo reduzir o tempo de preenchimento de formulários na Unilinkr, aumentar a automatização do processo e, conseqüentemente, atrair mais empresas para publicarem ofertas de emprego na plataforma.

3.2.2 Requisitos Não Funcionais

Ao descrever requisitos não funcionais, garante-se um sistema eficiente, seguro, fiável e sustentável. Avaliações como qualidade de desempenho ou questões legais, permitem melhorar a utilização do sistema pelo utilizador, protegendo-o. Apesar de não serem funcionalidades diretas, estas asseguram sustentabilidade do projeto a longo prazo, que conseqüentemente influenciará a experiência global dos utilizadores.

Para organizar e descrevê-los de forma sistemática, recorreu-se ao modelo FURPS+, uma abordagem comum em engenharia de software, agrupando os requisitos em várias categorias. A funcionalidade geral do sistema (*Functionality*), tal como regras de negócio ou segurança, a facilidade de utilização e acessibilidade para o utilizador (*Usability*), a confiabilidade do sistema (*Reliability*), a eficiência e tempo da resposta (*Performance*), a capacidade de manutenção, modulação e extensibilidade (*Supportability*) e outros requisitos, tal como compatibilidades, conformidade legal e restrições técnicas. Abaixo estão representados os requisitos não funcionais identificados para o sistema, segundo o modelo especificado:

Funcionalidade:

- O sistema deve ser compatível com os websites especificados, mesmo que apresentem layouts e estruturas variadas.

Usabilidade:

- O sistema deve ser acessível a utilizadores da plataforma Unilinkr, nomeadamente administradores e representantes de empresas, proporcionando uma experiência simples e intuitiva.

Confiabilidade:

- O sistema deve garantir fiabilidade na extração de dados e evitar falhas durante o scraping, mesmo com websites com estruturas diferentes.

Performance:

- O preenchimento automático do formulário deve ocorrer num curto período de tempo, garantindo uma resposta eficiente (<5min).

- A extração de ofertas de emprego da web deve ocorrer num curto período de tempo, garantindo uma resposta eficiente (<5min).

Suportabilidade:

- O código deve seguir uma estrutura modular, com componentes independentes, facilitando a manutenção e evolução do sistema.
- O projeto deve incluir documentação clara e atualizada, para suportar futuras melhorias e facilitar trabalhos futuros.

+:

- O sistema deve respeitar princípios de segurança, incluindo o cumprimento do RGPD, evitando a recolha ou partilha de dados pessoais sensíveis.

Os requisitos não funcionais são fundamentais para assegurar a fiabilidade, eficiência e manutenção do sistema a longo prazo, estando alinhados com as boas práticas de engenharia de software.

3.2.3 Palavras Chave:

As expressões expostas englobam tanto a temática como as funcionalidades do projeto.

- **Oferta de Emprego:** Publicação feita por uma empresa com o objetivo de recrutar candidatos.
- **Formulário de Oferta:** Estrutura digital com campos a preencher, como empresa, cargo, salário.
- **NLP (Processamento de Linguagem Natural):** Técnica de inteligência artificial que permite interpretar e gerar linguagem humana.
- **Web Scraping:** Processo automático de recolha de dados a partir de páginas web.
- **Unilinkr:** Plataforma onde são publicadas e geridas ofertas de emprego.
- **LLM (Grandes Modelos de Linguagem)** Modelos de linguagem de larga escala, como GPT ou LLaMa.
- **API REST:** Interface que permite a comunicação entre sistemas via HTTP.

3.3 Desenho

O desenho da solução, permite compreender objetivamente as funcionalidades do sistema, desde o contacto do utilizador até aos pormenores do sistema. Desta forma, define-se e planeia-se uma solução a implementar no sistema, assegurando compreensão, no futuro, permitindo assim a sua expansão.

Neste desenho, segue-se o modelo C4+1, que permite organizar a arquitetura em níveis, facilitando a comunicação entre não técnicos e desenvolvedores. Esta estrutura também representa vistas de processo, englobando todas as ações para cada caso de uso, especificado em 3.2.1.

3.3.1 Visão Geral da Solução

Os objetivos estipulados focam-se na automatização do preenchimento de formulários para a Unilinkr, aliada a um tempo de resposta elevado por parte do sistema a desenvolver - UniJobFormsBot, o qual terá as capacidades tanto de *scraping* como extração de dados. A forma mais intuitiva de permitir a ligação deste sistema ao resto da plataforma é através da implementação de duas páginas web para os dois requisitos especificados em ???. O sistema terá capacidade de receber informação sobre as ofertas de emprego através de duas fontes diferentes:

- Plataformas com ofertas de emprego;
- Descrição da oferta de emprego.

A primeira fonte será controlada pelo administrador da Unilinkr, visto que esta é uma tarefa complexa e pode ser utilizada para os mais variados propósitos. Todos os websites que a plataforma deseje pesquisar serão especificados previamente, permitindo ao utilizador iniciar de imediato a ação de *scraping*. Esta percorrerá todos os websites previamente especificados no sistema, com recurso a tecnologias definidas em 2.3, extraíndo minuciosamente todos os dados necessários para formar uma descrição de emprego. De seguida, será possível preencher, preferencialmente, todos os campos do formulário utilizando o sistema NLP com LLM. A segunda fonte estará presente na página das empresas, permitindo que estas tirem partido da automatização do preenchimento do formulário, simplesmente com um texto. O sistema utilizará igualmente o mesmo sistema, permitindo que ambos os requisitos se possam interligar.

Desta forma, o sistema UniJobFormsBot será capaz de processar ambas as descrições numa só implementação, pois ambas baseam-se no formulário da plataforma, criando um prompt que permita à AI API externa responder a questões baseadas no formulário, preenchendo todos os campos eficazmente. Assim, os campos podem ser avaliados sob regras de negócio, sendo removidos se errados. O formato de campos do tipo *date* foi um fator analisado, devido ao seu formato variado, permitindo assim que a AI API possa auxiliar na formatação do valor de acordo com regras de negócio, caso estas não passem no processo de validação. Campos com valores textuais podem passar a regras de validação mas falhar para campos indicados como desconhecidos, sendo necessária uma validação extra para essas situações. Tendo em conta a extensão do formulário, principalmente a sua estrutura separada por secções, a divisão destas tarefas por *threads* auxiliará na performance do sistema.

Com todos os dados do formulário preenchidos corretamente ou com apenas alguns campos não preenchidos, criar-se-á um modelo do formulário, baseado no frontend específico que fez o pedido. Apesar de ambos preencherem o mesmo formulário, as funcionalidades e interações continuam a ser diferentes. Desta forma, a modelação do mesmo terá considerações diferentes aquando da sua implementação.

Os pontos especificados em 3.2.2 refletem-se na divisão de funcionalidades em tarefas singulares, permitindo que cada caso de usos seja utilizado de forma individual, ao mesmo tempo que um utilizará funcionalidades do outro. As análises dos dados extraídos serão realizados sem comprometer utilizadores, permitindo uma utilização segura e confiável do sistema.

3.3.2 Arquitetura de Alto Nível

O sistema UniJobFormsBot é o responsável por guiar o sistema e conseguir cumprir as necessidades do utilizador.

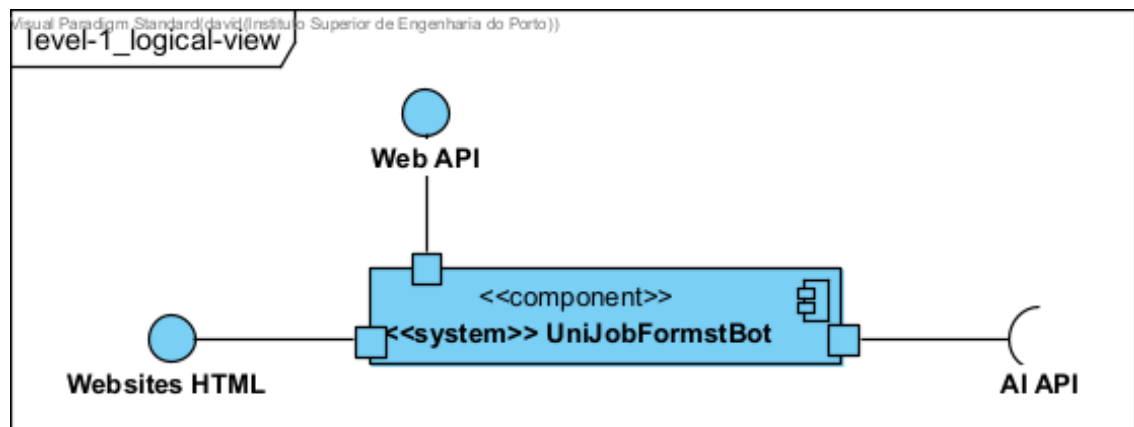


Figura 3.3: Vista lógica do sistema UniJobFormsBot (Nível 1)

Há dois requisitos funcionais definidos em 3.2.1, portanto será necessária a implementação dos casos de uso *UniBotScraper* e *UniScraperBot*, correspondendo a **RF2** e **RF1**, respetivamente.

No primeiro, *UniScraper*, o utilizador realizará uma conexão com o sistema de forma a extrair dados brutos do código HTML de um website. E *UniBot* tratará do processamento dos dados com base numa descrição de oferta de emprego, extraíndo e validando com AI API externa.

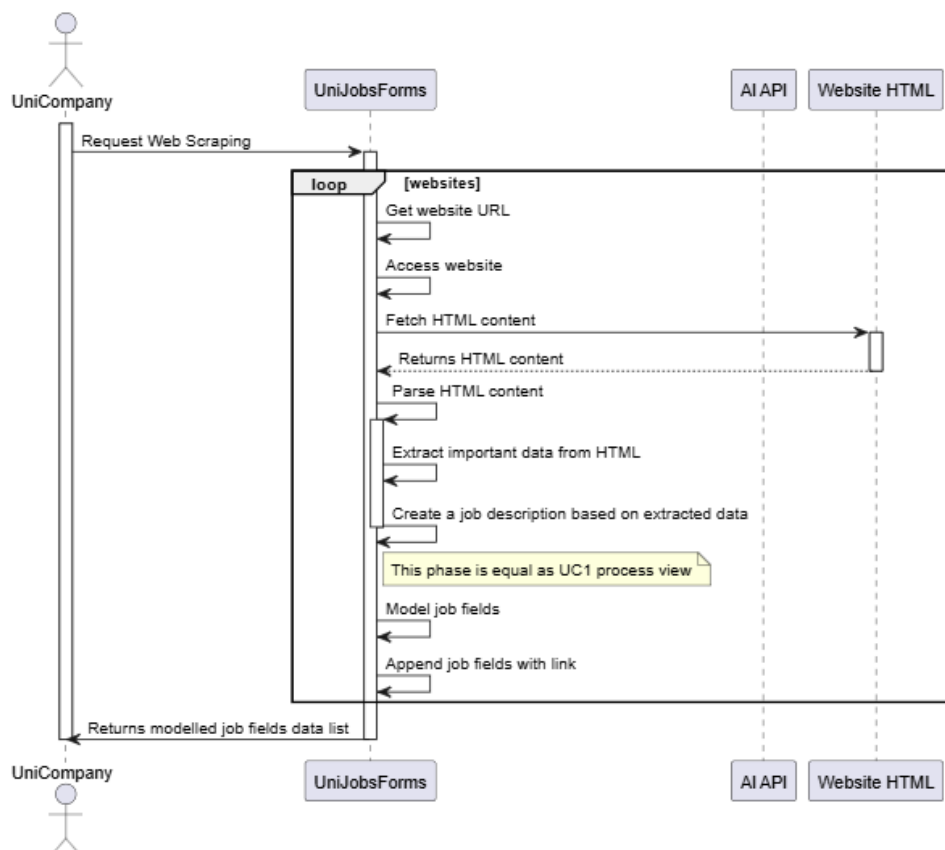


Figura 3.4: Vista de processos para o caso de uso UniScraper (Nível 1)

No caso de uso representado pela figura 3.4, espera-se que, iterando por múltiplos websites que partilham ofertas de emprego, se consiga extrair informação suficientemente capaz de produzir uma descrição. Adicionalmente, é possível utilizá-la para extrair dados e preencher um formulário de publicação de um novo trabalho. Considerando esta última afirmação, compreende-se que a ação é a mesma que *UniBotUC* (Use Case (UC)), tal como representado na figura 3.5.

Iterar sobre a figura na
página.

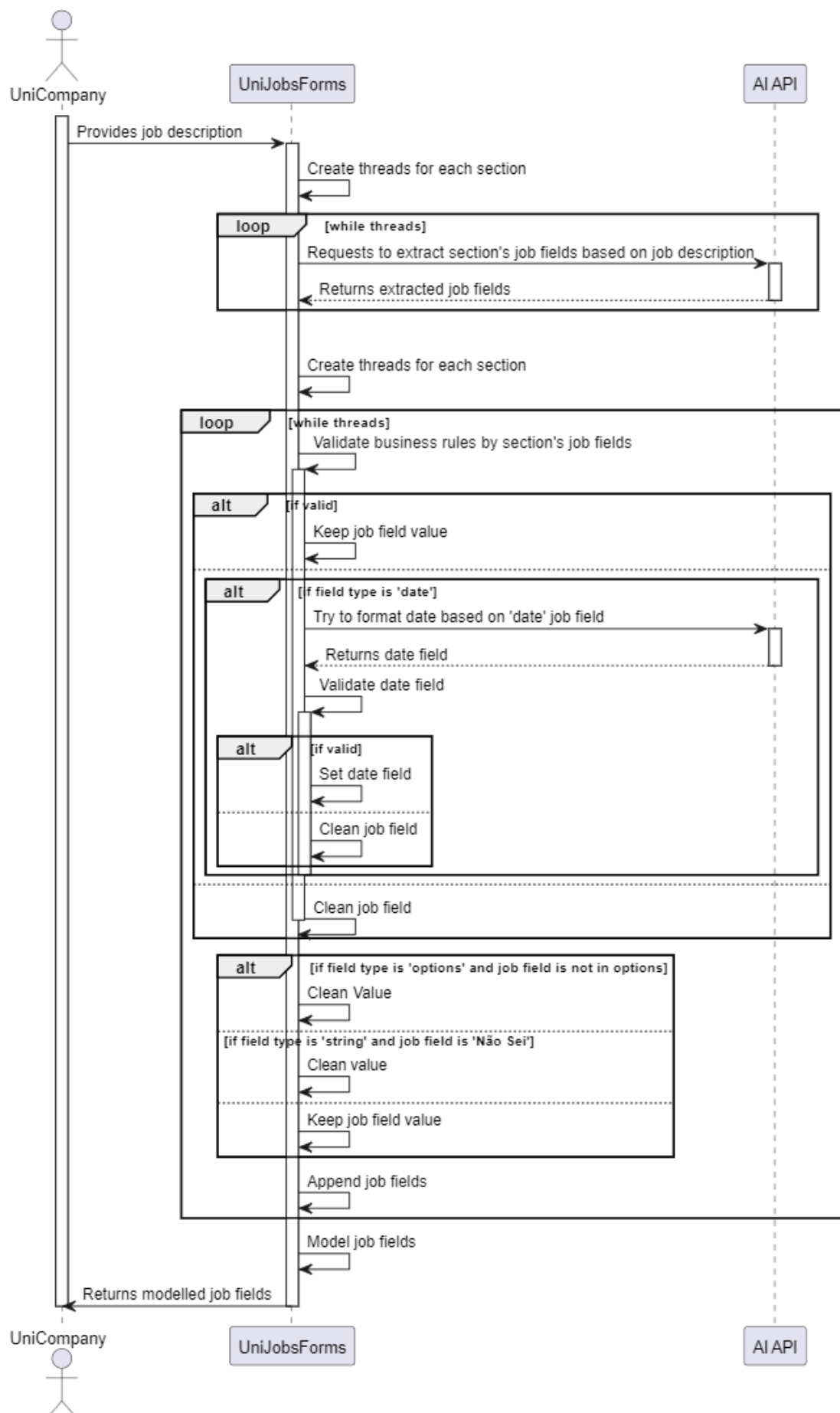


Figura 3.5: Vista de processos para o caso de uso UniBot (Nível 1)

Inicialmente, a descrição será processada por threads, cada uma responsável por uma secção do formulário. Estas irão criar uma prompt personalizada com os campos da sua secção, para enviar para a AI API, que retornará em formato JSON os campos identificados devidamente preenchidos. Para garantir a validação dos dados, para posterior construção do modelo, também serão implementadas threads, responsáveis pela validação de dados, juntamente com a limpeza de dados destacados como desconhecidos. Os campos do formulário são agrupados e constrói-se o formulário de forma precisa, permitindo retornar ao utilizador um conjunto de dados válido para uso posterior.

Desta forma, é possível concluir que, fisicamente, o ator da ação comunicará com a plataforma web da Unilinkr, e desta forma interagirá com o respetivo frontend que ativará funcionalidades de UniJobFormsBot. A figura 3.6 representa essa interação, e representa as comunicações externas que o sistema realiza, especificamente o website da *Net Empregos* e a AI API do *Ollama*. Estes sistemas externos são essenciais para o desenvolvimento de *UniScraperUC* (UC) e *UniBotUC* (UC), respetivamente, visto que permitem uma aplicação das tecnologias existentes especificadas em 2.3.

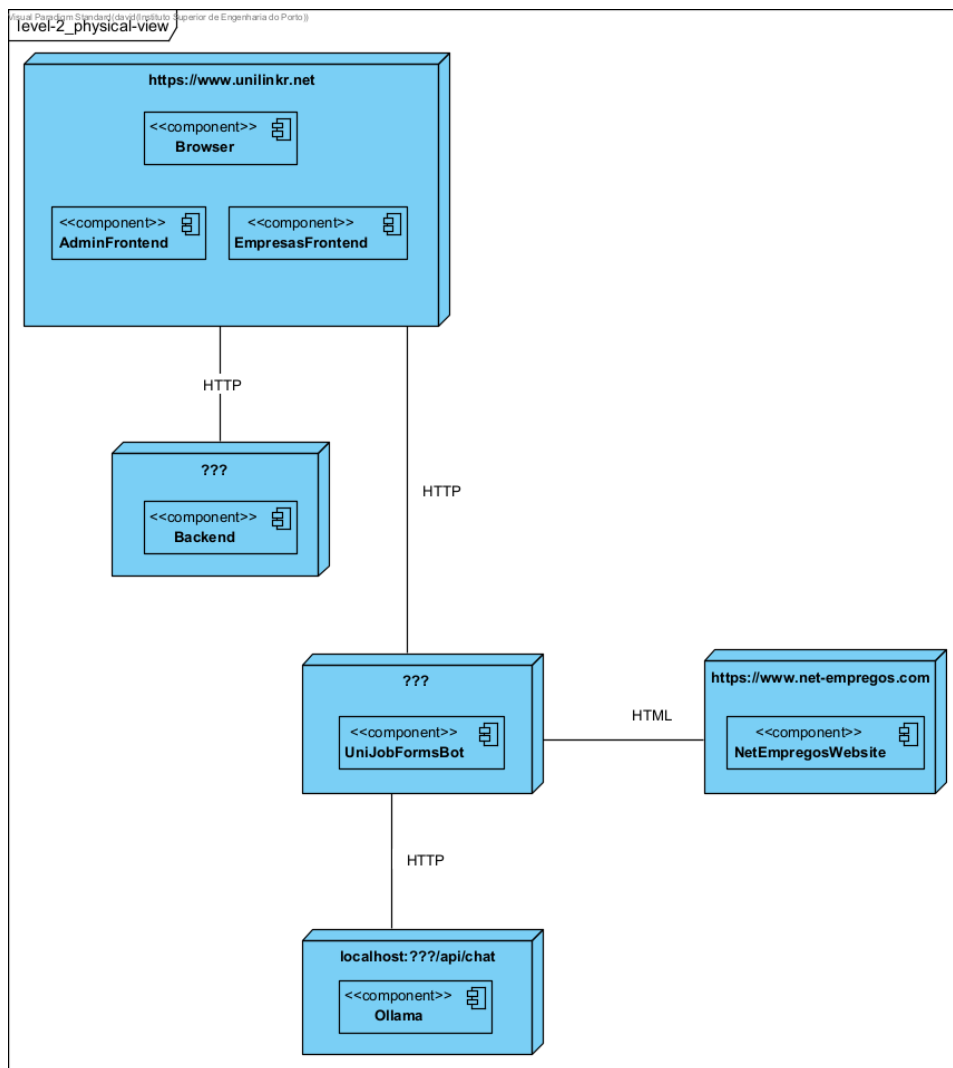


Figura 3.6: Vista física do sistema UniJobFormsBot (Nível 2)

A vista de implementação do sistema em 3.7 demonstra como a coexistência dos dois casos

de uso, juntamente com funcionalidades de processamento de dados e comunicação à AI API através de uma *gateway*.

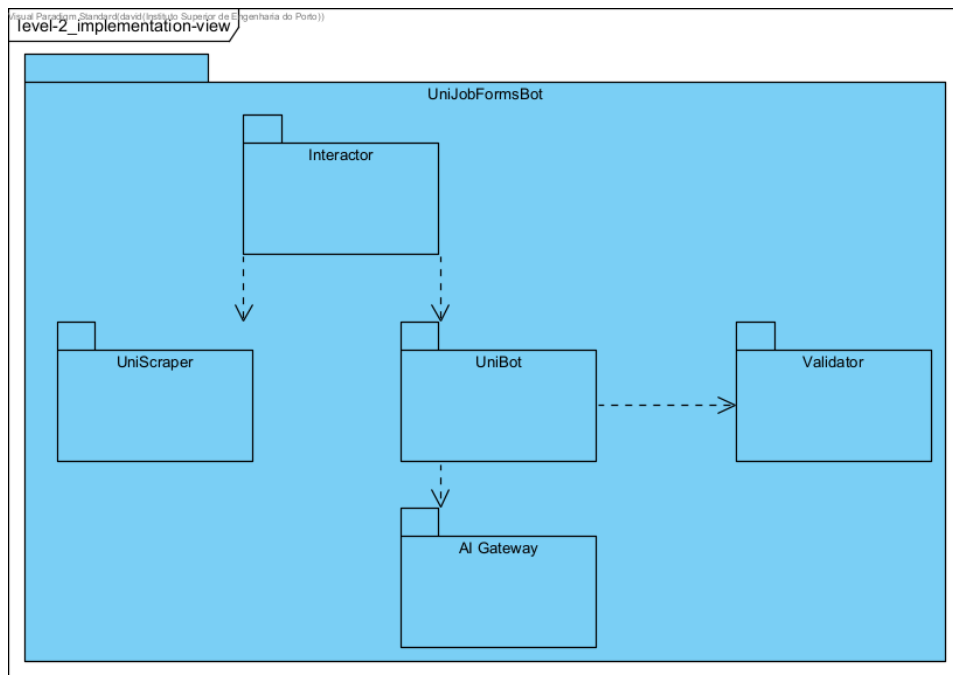


Figura 3.7: Vista de implementação do sistema UniJobFormsBot (Nível 2)

O módulo *Interactor* responsabilizar-se-á pela interação com o utilizador e controlo do fluxo, gerindo as funcionalidades previamente descritas. Este interagirá com os módulos *UniScraper* e *UniBot*. Consequentemente, este último comunicará com *AI Gateway* e *Validator*.

Desta forma, o utilizador, através da Web API, enviará um pedido para *Interactor*. Se for um pedido para iniciar Web Scraping, *UniScraper* começará a extração de HTML de Websites com ofertas de emprego, formulando uma descrição de acordo com o trabalho recolhido. Esta será posteriormente desenvolvida por *UniBotUC*, visto que se o pedido da Web API fosse referente à funcionalidade NLP, seria realizada a extração com base numa descrição enviada por um utilizador.

Nessa situação, *Interactor* receberá e enviará a descrição da oferta de emprego para *UniBot*, que se responsabilizará por:

- Extrair os campos do formulário com auxílio de *AI Gateway*;
- Validar os campos extraídos, seguindo regras de negócio especificadas em *Validator*.

Por fim, todos os dados retornam a *Interactor*, que os devolve ao utilizador que fez o pedido pela Web API, tal como se encontra especificado na figura 3.8.

Cada um dos componentes da figura 3.8, representativos dos módulos definidos na figura 3.7, são compostos por um conjunto de subcomponentes. *Interactor* é composto por *BotController*, *ScraperController* e *FlowController*, que representam os controladores do sistema e que interagem com o utilizador. Estes são capazes de iniciar ambos casos de uso: *UniScraperUC* (módulo *UniScraper*) e *UniBotUC* (módulo *UniBot*).

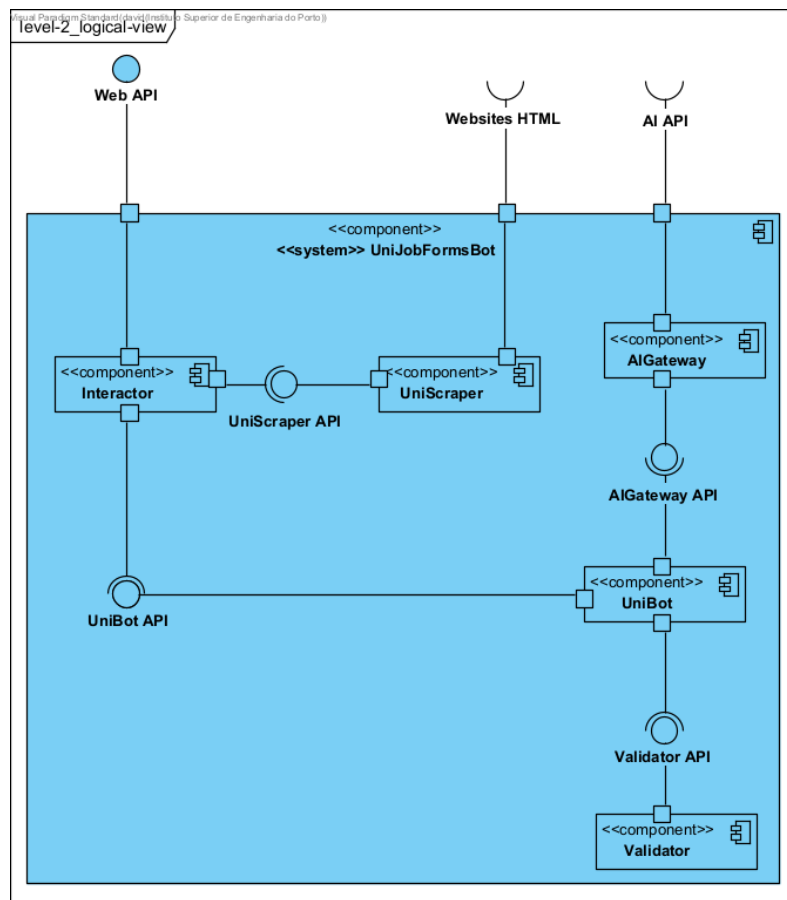


Figura 3.8: Vista lógica do sistema UnijobFormsBot (Nível 2)

O primeiro é composto por *ScraperProcessor*, focando-se na extração de ofertas de empregos de websites externos, criando descrições individuais com base na informação disponível, podendo extrair os campos do formulário correspondentes. Esta ação inicia o seguinte caso de uso, pois a extração utilizará um sistema NLP com LLM, através de *DataProcessor*.

O processo em *DataProcessor* engloba comunicação com a AI API externa, Ollama, através uma *AIGateway*. Também comunica com o módulo *Validator*, sendo este responsável por validar cada campo (componente *FieldValidator* de acordo com as regras de negócio especificadas (componente *BusinessValidator*). Desta forma é possível recolher dados concretos, limpos e validados.

Adicionalmente, como especificado na figura 3.9, componentes como *Service* e *JobModels* foram identificados, visto que serão utilizados em *UniBotUC* para a construção do modelo do formulário, de forma estruturada.

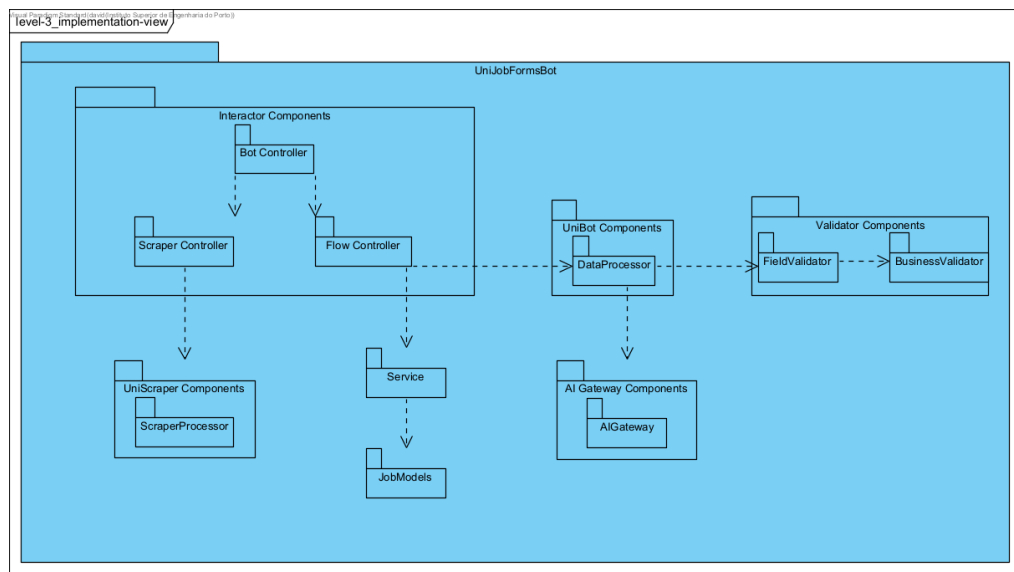


Figura 3.9: Vista de implementação do sistema UnijobFormsBot (Nível 3)

Cada um dos módulos da figura 3.9 terá funcionalidades específicas, mantendo o foco numa arquitetura modular. Como representado na figura 3.10, em Interactor, BotController receberá o pedido do utilizador e, com ScraperController começará o processo de Web Scraping, enquanto que com FlowController iniciará a extração baseada na descrição de uma oferta de emprego.

O componente ScraperController garante a interação entre o módulo Interactor e o módulo UniScraper, comunicando com ScraperProcessor, sendo este o responsável por iterar sobre todos os websites previamente definidos, para que consiga extrair trabalhos e formatar uma descrição relevante. O componente enviará o texto formatado para FlowController que tomará controlo do processo.

Neste serão criadas as threads que processarão a descrição de acordos com as secções atribuídas. As funcionalidades de extração e validação dos dados estão separadas, devido à integração de *UniScraperUC* com *UniBotUC*, contudo, ambas utilizam threads definidas da mesma maneira.

Sendo assim, este módulo comunica com UniBot, através de DataProcessor, que se responsabilizará por gerir o processo de extração e validação de dados. Inicialmente ocorre a fase de extração, por isso, a primeira interação será com AIGateway no módulo com o mesmo nome. Este componente formata uma prompt para garantir a extração dos dados requeridos e no formato correto, ao enviar o pedido à AI API externa.

Posteriormente, no módulo Validator, os dados serão processados novamente por secções em FieldProcessor, passando por um sistema de validação de campos com auxílio de BusinessValidator, e limpeza de campos de carácter desconhecido, garantindo segurança aquando da criação do modelo.

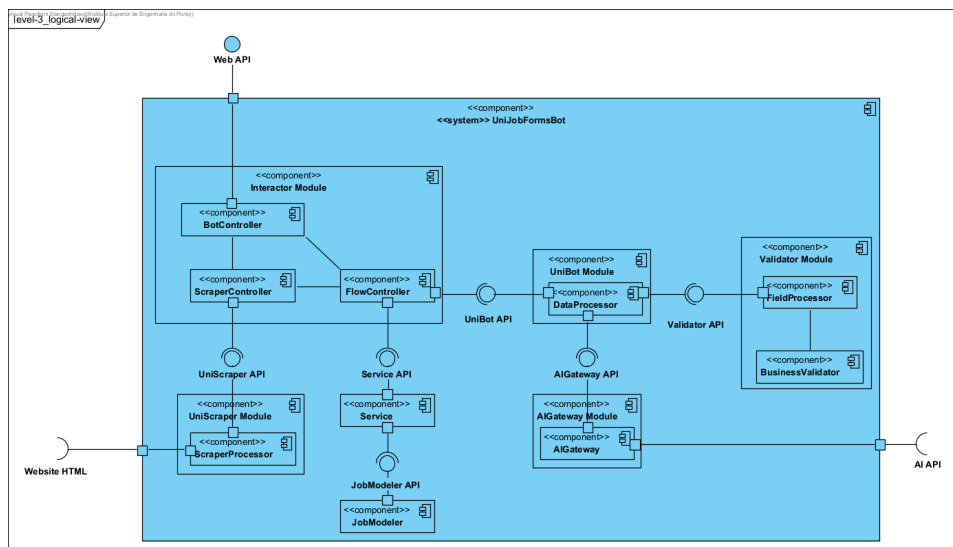


Figura 3.10: Vista lógica do sistema UnijobFormsBot (Nível 3)

Desta forma, é possível estruturar os componentes em diferentes níveis:

- **Drivers Layers:** Agrupa: Router, que se responsabiliza por direcionar os pedidos enviados pela Web API, Scraper Processor, porque contacta com informação exposta na web, e AIGateway visto que comunica com a AI API.
- **Interface Adapters Layers:**
Contém: BotController, que funciona como intermediário do Router, ScraperController, que mantém a lógica do processo de Web Scraping recebendo dados do ScraperProcessor, e FlowController, que guia o processo de extração de dados e validação desses mesmos, retornando ao BotController, tal como ScraperController.
- **Application Business Rule:**
Garante a aplicação das regras de negócio, por isso possui: DataProcessor, FieldValidator, BusinessValidator e Service. Os três primeiros componentes comunicam entre si e DataProcessor retorna informação a FlowController. O Service permitirá a construção segura do modelo.
- **Enterprise Business Rule:**
O modelo do formulário está representado em JobModeler.

Desta forma, conseguimos desenvolver o processo completo do sistema, focando e responsabilizando cada um dos módulos adequadamente. Esta representação está visível na figura 3.11, que dispõe logicamente todos os componentes especificados

Adicionalmente, o componente Router será responsável por encaminhar os pedidos provenientes do utilizador para a funcionalidade correspondente no BotController. Este último, implementado com recurso à biblioteca *FastAPI*, assumirá a responsabilidade pela gestão e tratamento adequado desses pedidos.

Neste enquadramento, o Router desempenha o papel de ponto de entrada da aplicação, expondo as diversas funcionalidades do sistema através de rotas bem definidas. Ao receber um pedido HTTP, o Router procede à invocação do método apropriado no BotController,

assegurando uma separação rigorosa entre a camada de transporte e a lógica de interação da aplicação.

O BotController assume, assim, o controlo dos fluxos de execução, articulando-se com os demais controladores de forma a satisfazer as operações solicitadas. Esta camada estabelece a interface direta com a *framework* FastAPI, beneficiando das suas capacidades intrínsecas, nomeadamente a validação automática de dados, a definição declarativa de endpoints e a gestão eficiente de dependências.

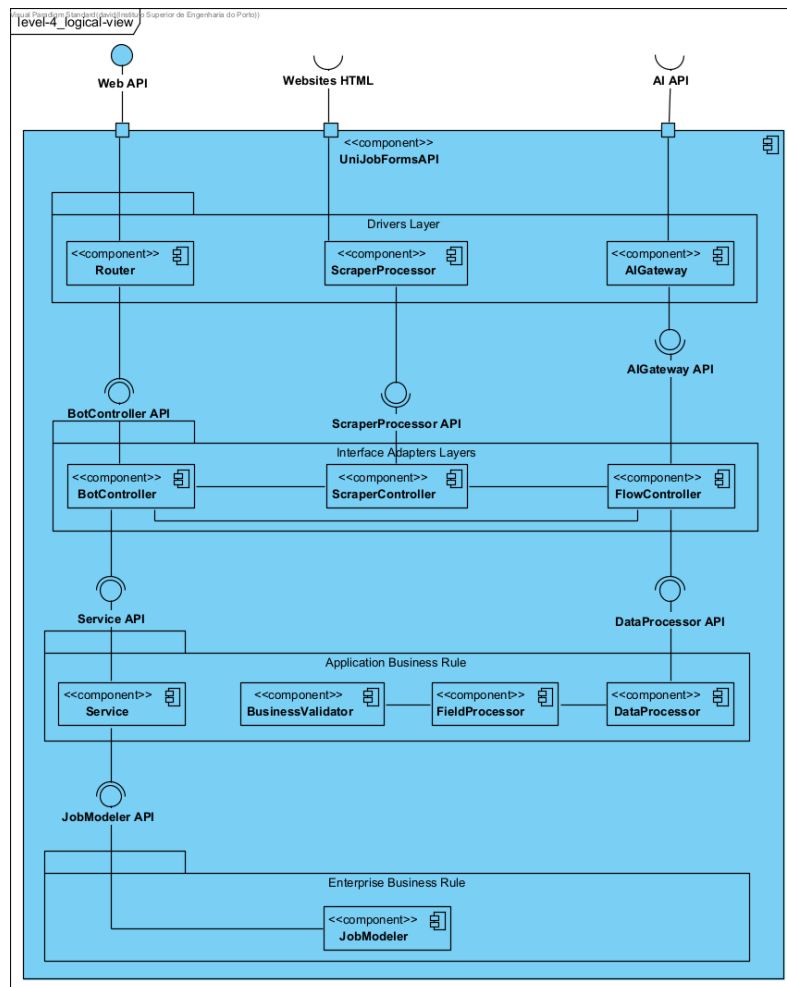


Figura 3.11: Vista lógica do sistema UnijobFormsBot (Nível 4)

O componente Router, especificado como a porta de entrada para o utilizador comunicar com UniJobFormsAPI, pode iniciar *UniScraperUC* quando o sistema receber um GET, POST *request* `/scraper`, enviando-o para ScraperController. Neste, todos os websites serão iterados, recolhendo descrições com informação relevantes por ScraperProcessor.

A solução desenhada na figura 3.12 foca-se no website da *Net Empregos* ?? <https://www.net-empregos.com>, uma plataforma com milhares de ofertas de emprego, publicadas por empresas que procuram dos mais variados candidatos. Neste, é necessário ultrapassar uma secção com *cookies*, onde o website armazena informações no navegador do utilizador como preferências de navegação, autenticação ou rastreamento. Desta forma, é possível recolher

HTML puro do website, extraíndo cards e as suas ofertas de emprego, criando uma descrição de emprego individual.

Esta será analisada e todos os campos do formulário serão extraídos da mesma forma que *UniBotUC*, que receberá a descrição de emprego. Posteriormente, o formulário, o seu url e uma marcação que identifica se os campos estão completos ou não, serão agrupados numa só lista e retornados ao utilizador para uso posterior.

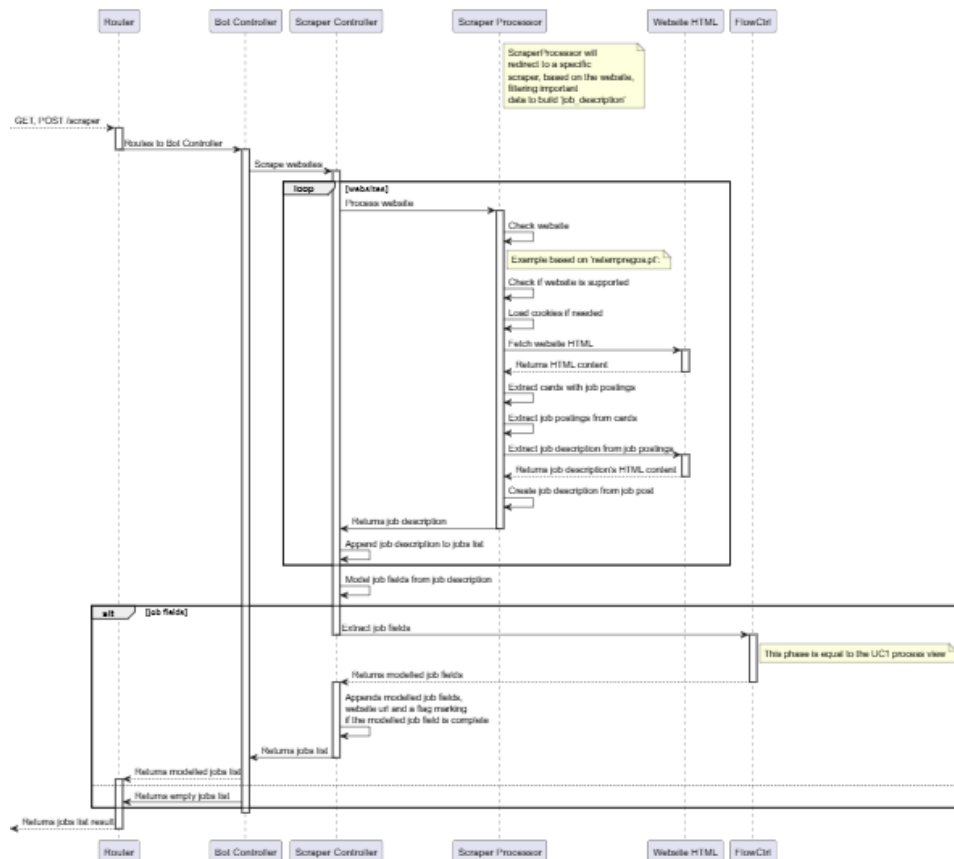


Figura 3.12: Vista de processos do caso de uso *UniScrapper* do sistema Unijob-FormsBot (Nível 4)

Como cada uma destas descrições é possível ser extraída, utilizando as mesmas funcionalidades descritas pelo caso de uso *UniBotUC*, na figura 3.13. Para esta situação, o utilizador terá de enviar um pedido HTTP semelhante a *UniScrapperUC*, contudo em vez de "/scrape", será "/chat", juntamente com uma descrição da oferta de emprego.

Router redirecionará para BotController, que extrairá os campos de acordo com o texto partilhado. FlowController criará threads para cada uma das secções do formulário, que comunicam com DataProcessor. Este componente gerirá a extração dos dados com auxílio da AIGateway, auxiliada de prompts personalizadas, recolhendo todos os campos e retornando *job fields* para serem validados.

O processo de validação e limpeza dos campos, em FieldValidator, terá a mesma estrutura de threads como na situação de extração. Cada uma das secções validará os seus campos de acordo com as regras de negócio em BusinessValidator e caso contrário, limpará esses valores. Campos do tipo *Date* são sensíveis, pois podem ser representados quer por texto

quer números e caratères não alfanuméricos. Nesta situação, a AI API auxilia na extração de uma data de acordo com o formato esperado pelo sistema, sendo validada novamente e, caso falhe, limpa definitivamente.

Campos preenchidos com dados considerados desconhecidos, como por exemplo "Não sei.", podem, inicialmente, passar nas validações sobre regras de negócio, em BusinessValidator. Esta situação ocorre, por exemplo, em respostas do tipo "texto", o que implica que certos campos possam ser validados devido à sua representação textual válida, mas o seu conteúdo considera-se desconhecido. Desta forma, estes campos representam conteúdo inválido para a plataforma, sendo, por isso, também limpos. Após este processo, as ofertas de trabalho estão prontas para serem modeladas e convertidas no formulário de trabalho.

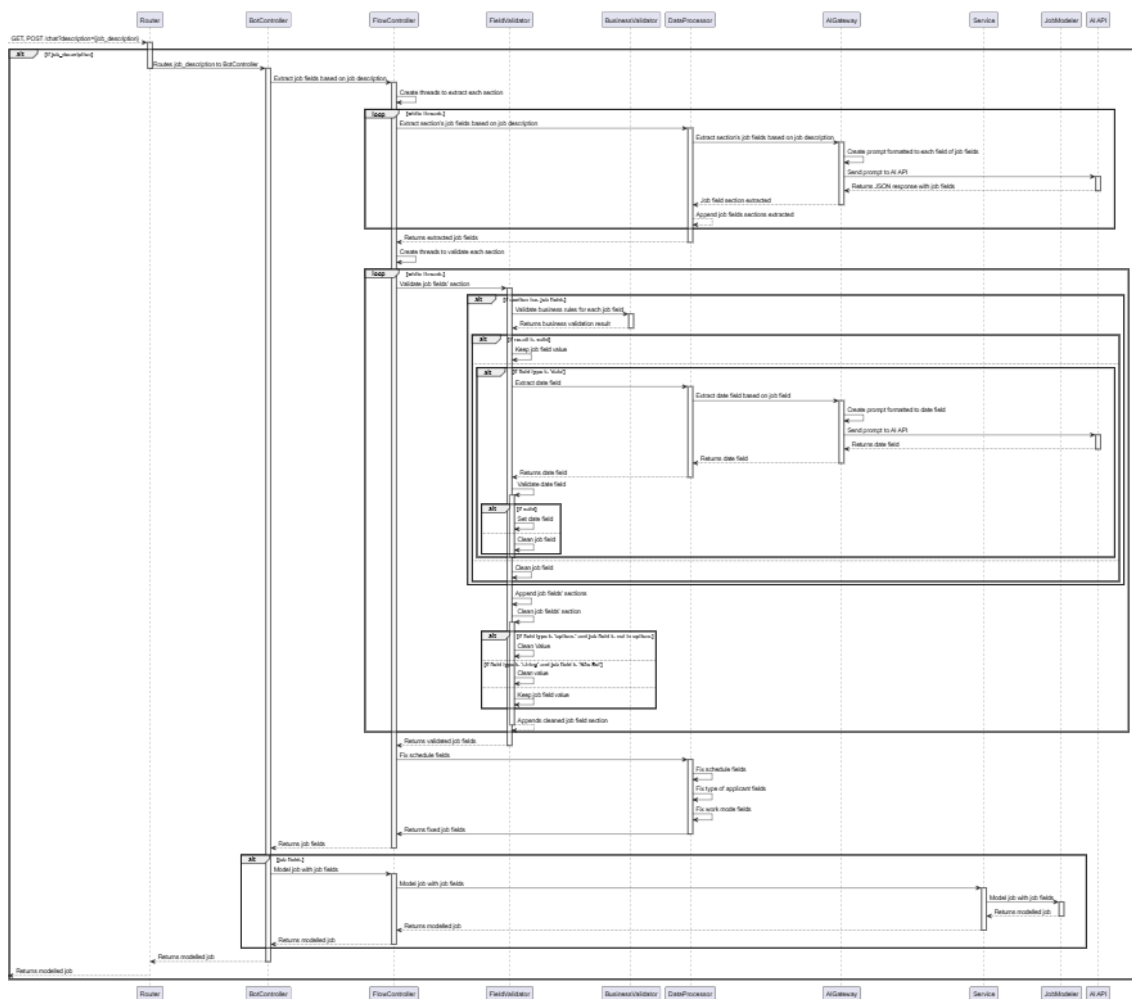


Figura 3.13: Vista de processos do caso de uso *UniBotUC* do sistema Unijob-FormsBot (Nível 4)

A estrutura lógica e funcional do sistema, exigiu o desenvolvimento desta arquitetura clara e modular. Desta forma, é possível seguir uma descrição clara da solução a implementar, garantindo que, futuramente, possa ser reanalisado e compreendido com clareza. A facilidade na comunicação, a manutenção e escalabilidade são fatores que melhoram a plataforma, servindo como referência sólida para evoluções do sistema.

3.3.3 Justificação da Arquitetura

Cada módulo foi definido com funcionalidades únicas, permitindo mantê-las de forma eficaz. UniBot e UniScraper, apesar de comunicarem, estão separados, o que permite manter uma funcionalidade sem comprometer a outra. UniBot é o único módulo capaz de comunicar com AIGateway, sendo este responsável por retornar a resposta vinda da AI API. UniBot comunica com Service para a construção do formulário, retornando o objeto de JobModeler. Por último, BotController, responde ao pedido feito por um utilizador.

Desta forma, compreende-se coesão interna em cada módulo, cumprindo a implementação dos dois casos de uso e permitindo automatizar o processo. Contudo, apesar de o automatizar, tem alguns fatores a melhorar em relação ao tempo de processamento do sistema, impedindo reduzir significativamente o tempo para o preenchimento de um formulário. A implementação da modularidade, foi expressiva na organização do projeto, porque permitiu manter uma estrutura equilibrada, através de manutenção, e com espaço para evolução futura do sistema, recorrendo a proxies rotativos e uma AI API mais forte.

A utilização de Python requer um foco extra devido às alterações que possam ser efetuadas em bibliotecas implementadas, comprometendo o sucesso do sistema. É necessário revisões sobre as bibliotecas implementadas na plataforma, apesar destas possuir soluções otimizadas, como é o caso de BeautifulSoup e ollama, bibliotecas que permitem implementações eficientes na extração de dados e comunicação com uma AI API externa, respetivamente.

A adoção de FastAPI proporciona ainda uma gestão estruturada de erros, mecanismos de autenticação e serialização de dados. Desta forma, garante à aplicação níveis elevados de robustez, desempenho e escalabilidade. Todo o ciclo de vida de um pedido — desde a sua receção até à emissão da resposta — é conduzido de forma modular e responsiva, em conformidade com os princípios da arquitetura limpa (Clean Architecture).

O facto de um dos casos de uso utilizar funcionalidades do outro, sem comprometer a execução de nenhuma delas, demonstrando assim uma não repetição de lógica (Don't Repeat Yourself). Desta forma, esta organização arquitetural promove um desenho desacoplado e altamente testável, permitindo que eventuais alterações na lógica de negócio não comprometam a camada de exposição da API. Assim, facilita-se significativamente a manutenção, extensibilidade e evolução sustentada do sistema ao longo do tempo.

Por fim, a arquitetura adotada fornece uma base sólida, flexível e escalável, que se alinha com os objetivos planeados, sendo capaz de evoluir e adaptar-se.

3.3.4 Especificações Globais do Sistema

As especificações globais do sistema permitem especificar quais regras, limitações, estruturas ou comportamentos gerais se aplicam a todo o sistema. No desenvolvimento do presente projeto, foi necessário utilizar a estrutura do formulário da plataforma.

Estrutura do Formulário

A estrutura do modelo representativo do formulário da oferta de emprego tem considerações diferentes tanto para *UniBotUc* quanto para *UniScraperUC*.

O formulário é composto por quatro secções: **details, location, schedules and vacancies e compensation and benefits**, seguindo a estrutura interna que se encontra representada na tabela 3.1.

Secções	Campos	Nome no sistema
Details	nível de especialização	degree_of_specialization
	título da oferta	job_title
	área de trabalho	work_area
	descrição	job_description
Location	modo de trabalho	work_mode
	morada	address
	raio geográfico ^[1]	geo_radius ^[1]
Schedules and Vacancies	tipo de horário	schedule_type
	data de início	{schedule_type}startDate
	data final	{schedule_type}endDate
	nome dos turnos	{schedule_type}shifts
	vagas	vacancies
Compensation And Benefits	frequência de pagamento	payment_frequency
	quantidade	amount
	como pagar ao estudante	how_to_pay_student
	quem pode concorrer	type_of_applicant
	nome das equipas	teams_ids
	benefícios	benefits

Tabela 3.1: Estrutura do modelo do formulário da oferta de emprego

3.3.5 Padrão de Arquitetura e Boas Práticas

A seleção desta arquitetura foi baseada em três fatores principais: modularidade, performance e compatibilidade com API REST. A divisão em módulos garante escalabilidade, manutenção e isolamento claro das tarefas, separando os casos de uso de forma que alterações numa das funcionalidades não afetam a outra. Desta forma, garante-se uma arquitetura limpa (Clean Architecture), permitindo que o sistema esteja organizado em diferentes camadas com responsabilidades distintas.

A camada *Drivers Layers* atua como ponto de entrada de dados externos, por exemplo chamadas HTTP vindas da Web API ou respostas provenientes da AI API externa, enquanto que *Interface Adapter Layer* controla o fluxo interno e a execução lógica dos casos de uso. A *Application Business Rule* centraliza regras como validações, extração de dados e controlo do estado do sistema. Por fim, a *Enterprise Business Rule* garante a conclusão do sistema, focando na lógica de domínio, como modelação dos dados, seguindo as estruturas do formulário para cada uma das páginas web da Unilinkr. As diferenças da modelação dependerão do utilizador que realizar o pedido, pois, apesar de semanticamente necessitarem das mesmas informações extraídas, a estrutura interna do sistema varia entre os *frontends*. Assim, é necessário modelar cada caso de forma distinta, respeitando as particularidades do interface, ao mesmo tempo que se preserva coerência no domínio.

Desta forma, esta estrutura terá um acoplamento reduzido entre componentes, permitindo uma alta coesão interna, possuindo características que suportam alta testabilidade, juntamente com facilidade para manutenção futura e, possivelmente, evolução do sistema. A arquitetura desenvolvida está orientada de acordo com princípios SOLID, um conjunto de práticas fundamentais para garantir código coeso, extensível e de fácil manutenção. Das boas práticas existentes, foram implementadas as seguintes:

- **Single Responsibility Principle:**

Single Responsibility Principle (SRP) implica que um módulo ou um componente deve ter apenas uma única responsabilidade. Por exemplo, o módulo *Validator* responsabiliza-se pelo processo de validação dos campos recolhidos. Contudo, este está composto por dois componentes: *FieldValidator* para gerir o fluxo e limpar os campos, e *BusinessValidator* para realizar a avaliação individual de cada um.

- **Open/Closed Principle:**

Open/Closed Principle (OCP) significa que os componentes desenhados podem ser modificados, sem modificar a lógica existencial. Por exemplo, novos tipos de campos podem ser adicionados, implementando novas funções para validação dos mesmos, sem afetar as funcionalidades existentes.

- **Dependency Inversion Principle:**

Dependency Inversion Principle (DIP) desenha o sistema de modo a que módulos mais altos, não dependam de implementações concretas de UC, mas sim de abstrações. Isto é visível aquando da separação da lógica de domínio (*Enterprise Business Rule*) e adaptadores de interfaces (*Driver Layers*), como representado na figura 3.11.

A integração de Clean Architecture e os princípios SOLID, permite criar uma solução robusta, testável e sustentável. Assim, o sistema pode evoluir sem comprometer a sua estabilidade. Por outro lado, o sistema reutiliza funcionalidades transversais de acordo com o princípio *Don't Repeat Yourself (DRY)*. Um exemplo claro é o uso da funcionalidade NLP, que é partilhada entre os casos de uso *UniBotUC* e *UniScraperUC*, visto que permite que a extração de dados a partir de descrições textuais, seja possível com apenas uma implementação, sendo esta escalável e reutilizável.

Além dos princípios SOLID, este projeto também seguiu padrões General Responsibility Assignment Software Patterns (GRASP), aplicando o padrão *Controller* nos controladores dos casos de uso responsáveis pelo tratamento de pedidos e respetivo direcionamento para os vários módulos do sistema. O padrão *Information Expert* foi seguido quando se responsabilizou módulos específicos, como validação (*Validator*) e extração (*UniBot*) de dados, para apenas possuírem informação relevante de acordo com a sua funcionalidade. Por fim, os padrões *Low Coupling* e *High Coesion* refletem-se na estrutura modular do código, garantindo independência entre componentes e definindo responsabilidades bem delimitadas.

A abordagem modular também permite que o sistema seja extensível, por exemplo, com a introdução de novos canais de entrada de dados, não sendo necessário alterar profundamente o núcleo da aplicação. A adoção de *FastAPI* garantiu o uso de uma framework web moderna, com alto desempenho, capaz de construir APIs em Python. Esta contribui para a compatibilidade REST, gerindo automaticamente entradas e saídas de dados, juntamente com a criação de interfaces limpas e bem documentadas.

Por fim, a organização arquitetural e as boas práticas adotadas alinham-se com os objetivos do projeto, oferecendo uma solução reutilizável e facilmente integrável na plataforma *Unilinkr*, sem comprometer o desempenho atual, nem a clareza estrutural do código.

Capítulo 4

Implementação da Solução

A aplicação a ser desenvolvida divide-se em tarefas para um novo sistema, UnijobFormsBot, para a página frontend do administrador da plataforma e para as empresas registadas na mesma.

- **UnijobFormsBot** - o sistema responsável por cumprir as tarefas estipuladas para o sistema AI de chatbot para empresas, permitindo a automatização do formulário, através da abordagem definida em 3;
- **Frontend Administrador** - página responsável por garantir a interação do administrador com a funcionalidade *scraping* do sistema UnijobFormsBot;
- **Frontend Empresas** - página responsável por permitir que empresas possam automatizar a publicação de ofertas recorrendo a descrições de empregos, através da funcionalidade NLP com LLM.

4.1 Descrição da Implementação

A implementação do sistema UnijobFormsBot, seguiu a estrutura modular previamente definida, onde cada um dos módulos suportou as suas funcionalidades devidamente. A abordagem do sistema foi desenvolvida em Python, utilizando a aplicação VSCode, para programar, e GitHub, para controlo de versões

Desta forma, a estrutura modular permite que os casos de uso interajam, construindo uma solução robusta, capaz de atingir os objetivos estipulados. Esta solução permite ser iniciada, testada e coberta numa só, englobando comandos para:

- **Iniciar:** `"python -m uvicorn src.router:app --reload"`
- **Testar:** `"python -m unittest discover -s tests"`
- **Cobrir:** `"coverage run -m unittest discover -s tests && coverage report -m"`

Na inicialização do sistema, o uso de "uvicorn" evita problemas de *path* ou *imports* relacionados ao diretório atual, e `"-- reload"` permite que o sistema possa ser atualizado e analisado em tempo real, não sendo necessário reiniciá-lo manualmente. Para testar o servidor, a biblioteca "unittest" permite rodar os testes que forem identificados por `"discover -s tests"`. Por último, recolher o nível de cobertura dos testes implica corrê-los novamente com `"coverage report -m"`, obtendo um ficheiro ".coverage" com a informação recolhida.

De forma a demonstrar visualmente os resultados obtidos, criou-se uma página Web nos respetivos casos de uso, integrada com a plataforma. Assim, utilizando JavaScript e HTML,

como nas restantes páginas do frontend, desenvolveu-se uma opção que permite comunicar com a UnijobForms API.

4.1.1 Página do Admin

A página web do administrador da Unilinkr, está configurada com uma aba lateral, Uni Scraper, que permite comunicar com o frontend estabelecido para o caso de uso. Inicialmente, nenhuma oferta de emprego está descoberta, visto que se visualiza a primeira interação do utilizador com a página. Na parte superior, existe um botão, "Run Scraper", que permite estabelecer a comunicação com a funcionalidade *scraping* de UnijobFormsBot.

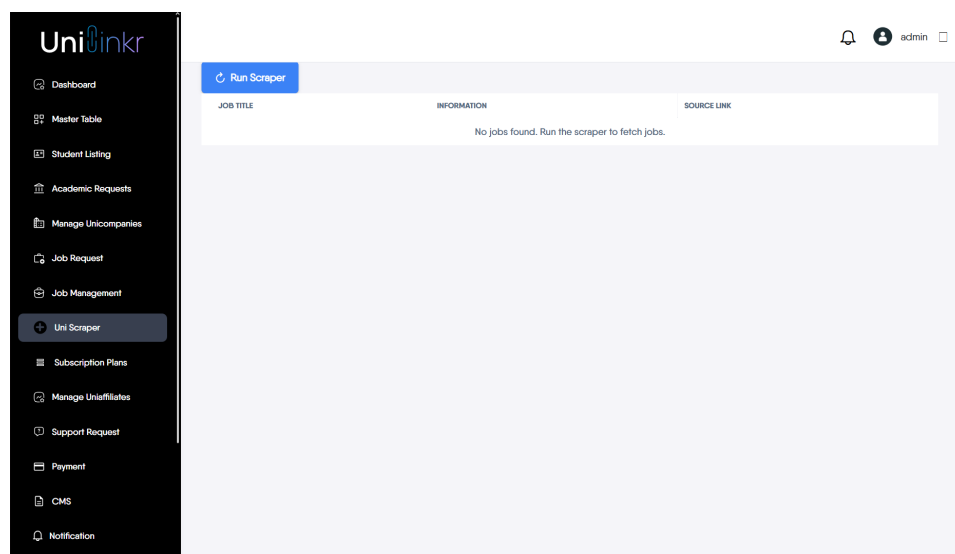


Figura 4.1: Página do admin, *Uni Scraper*-

No momento de desenvolvimento do projeto, foi abordada uma estratégia capaz de extrair com precisão, campos relevantes na página da *Net Empregos* [31], por isso os resultados remontarão a essa fonte, tal como demonstrado na figura 4.2.

Sendo assim, o processamento do sistema começa com *BotController*, que recebe o pedido redirecionado pelo *router* e comunica com os outros componentes internos, delegando responsabilidades a *ScraperController*. Este módulo comunica unicamente com o módulo *UniScraper*, o qual se responsabiliza por controlar o fluxo da extração realizado por *ScraperProcessor*.

A sua inicialização, como demonstrado no exemplo em 4.1, permite configurar um navegador *Chrome* com comportamentos específicos, como por exemplo: "*-headless=new*", que permite correr o navegador sem interface gráfica; "*-disable-gpu*", que desativa a GPU, para poder operar sem bugs em ambientes sem este mecanismo; "*-no-sandbox*", para evitar problemas de permissões; "*-window-size=1920,1080*", permite garantir que o layout das páginas seja carregado naturalmente; e "*user-agent=(...)*", permite simular um utilizador real, reduzindo as probabilidades de bloqueio por parte de plataformas, como a *Net Empregos*.

```
1 class ScraperProcessor:
2     def __init__(self, headless=True):
3         self.options = Options()
4         if headless:
5             self.options.add_argument("--headless=new")
```

```

6         self.options.add_argument("--disable-gpu")
7         self.options.add_argument("--no-sandbox")
8         self.options.add_argument("--window-size=1920,1080")
9         self.options.add_argument(
10             "user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
11             "AppleWebKit/537.36 (KHTML, like Gecko) "
12             "Chrome/91.0.4472.124 Safari/537.36"
13         )

```

Listing 4.1: Inicialização da classe ScraperProcessor

Desta forma, será possível extrair as ofertas da *Net Empregos* de forma segura, recolhendo todas as ofertas disponíveis. Para cada uma delas, serão extraídos os seus dados, de forma a formatar uma descrição baseada no texto recolhido, tal como exemplificado em 4.2. Através de BeautifulSoup é possível extrair exatamente os campos fundamentais para formular uma descrição relevante, compreensível e direta.

```

1 def parse_offer_page(self, link):
2     html = self.safe_get(link)
3     soup = BeautifulSoup(html, "html.parser")
4     job_title = soup.find("h1", class_="title")
5     job_location = soup.find("a", class_="oferta-link")
6     job_description = soup.find("div", class_="job-description mb-40 dont
7     -break-out")
8
9     description = f"""
10     Título: {job_title.text.strip() if job_title else ''}
11     Localização: {job_location.text.strip() if job_location else ''}
12     Descrição: {job_description.text.strip() if job_description else ''}
13     """
14
15     return {
16         "job_fields": self.flow_controller.extract_job_fields(description),
17         "source_link": link
18     }

```

Listing 4.2: Função para extração dos campos da oferta de emprego, retornando com o respetivo *source link*.

A função de extração dos campos do formulário, "extract_job_fields", de FlowController, processará o processo de análise da descrição de acordo com o caso de uso *UniBotUC*, visto que esta funcionalidade faz exatamente essa tarefa, seguindo a regra DRY. Essa especificação está presente em 4.1.2, e, ao retornar os campos obtidos, será formulado um dicionário com esses dados e o respetivo *source link* - o link do website correspondente à oferta real.

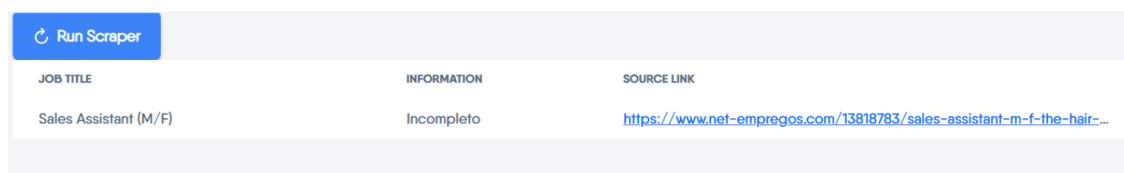
Garantir a estrutura correta dos campos é necessário para assegurar o sucesso da solução. Desta forma, no caso de uso *UniScraper*, o modelo terá as seguintes restrições:

- *code-name*: um dicionário aplicado por:
 - degree_of_specialization,
 - work_area,
 - work_mode,
 - district,
 - schedule_type,

- payment_frequency
- type_of_applicant
- id-code-name: um dicionário aplicado por:
 - how_to_pay_student

Assim, na página do administrador o formulário será preenchido corretamente para os campos existentes. Em ambos os casos, a estrutura *code-name* e *label-value* representa um conjunto de dados em que o primeiro valor é o exato nome no sistema, enquanto que o segundo é o texto que ficará visível para o utilizador. No caso do *id-label-value*, o primeiro campo será a posição de um valor numa lista de opções, permitindo, posteriormente, completar o formulário, selecionando a opção correta.

Desta forma, a extração da oferta de emprego terá o formato adequado mal chegue ao utilizador, permitindo otimizar a publicação da oferta. A figura 4.2 demonstra um exemplo real, da implementação do sistema para *Net Empregos*, tendo sido recolhida apenas uma oferta para efeitos de análise.



JOB TITLE	INFORMATION	SOURCE LINK
Sales Assistant (M/F)	Incompleto	https://www.net-empregos.com/13818783/sales-assistant-m-f-the-hair-...

Figura 4.2: Resultados do Web Scraping.

Como esperado, o sistema identificou apenas uma oferta de emprego do website, neste caso "Sales Assistant (M/F) - THE HAIR SHOP - TAVIRA PLAZA"[32]. A oferta online publicada pela empresa, descreve um colaborador para trabalhar em Faro, num regime full-time e com uma forte orientação para o atendimento personalizado. A descrição menciona que esta tem como objetivo principal garantir uma experiência comercial e de excelência, contudo, as informações requeridas para a plataforma são reduzidas. Questões sobre vagas, compensações e até horários ficam por responder, criando dificuldades ao sistema.

Sendo assim, a oferta na figura 4.2, identifica-se como "Sales Assistant (M/F)", que está de acordo com o título da oferta em *Source Link*, que representa o URL de onde a informação foi recolhida. Contudo, um dos campos apresenta-se como "Incompleto", referente a *Information*, que identifica se os campos estão todos preenchidos ou não, significando que alguns dos campos necessários estão a faltar.

A situação em questão preencheu os campos descritos nas figuras 4.3, 4.4 e 4.5. O design visual não foi completamente implementado por questões de tempo, tendo em conta que o foco era otimizar o sistema. Contudo, o resultado demonstrado no frontend do administrador, permite compreender quais campos foram preenchidos e com que informação, permitindo usabilidade por parte do utilizador.

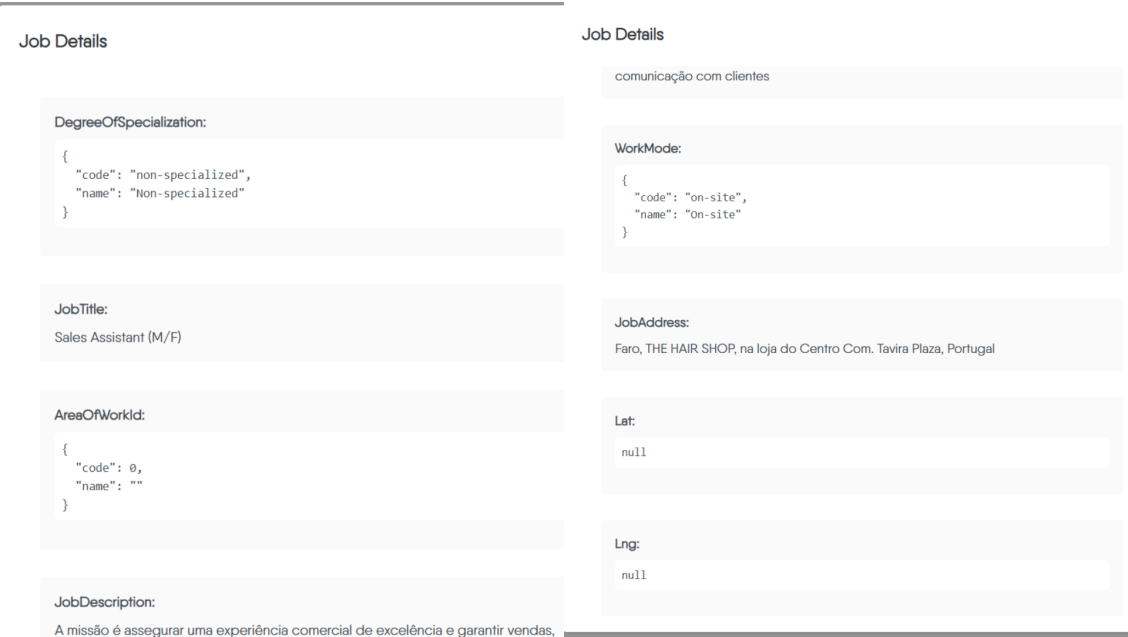


Figura 4.3: Análise das ofertas recolhidas - Parte 1/3.

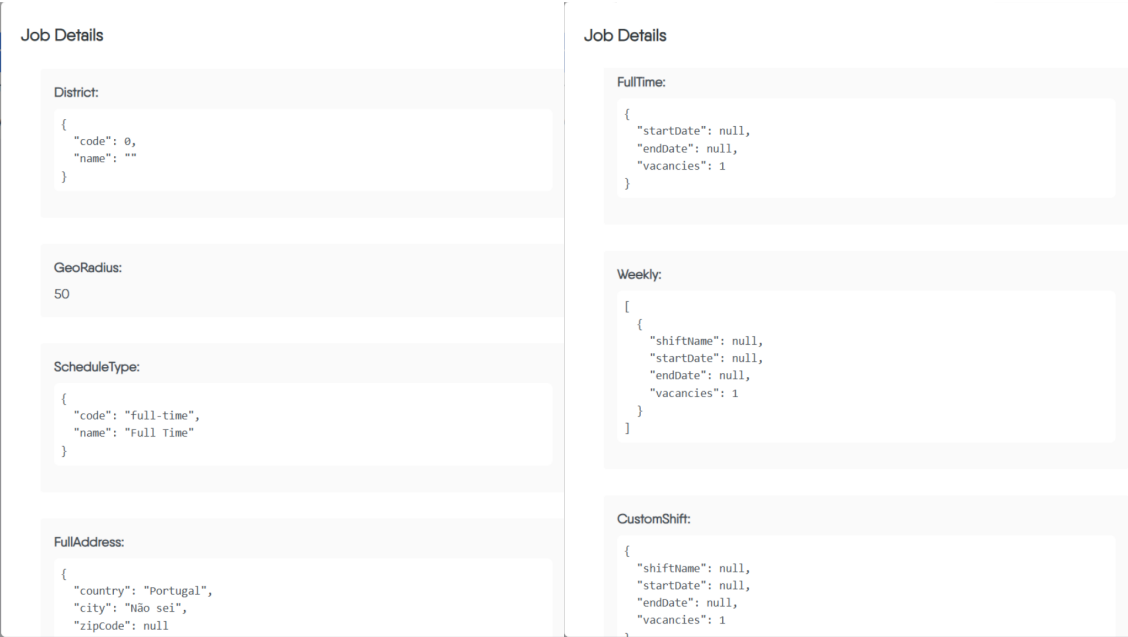


Figura 4.4: Análise das ofertas recolhidas - Parte 2/3.

Job Details

Payment Frequency:

```
{
  "code": "",
  "name": ""
}
```

Amount:

null

Benefits:

Um bom ambiente de trabalho, A possibilidade de progressão na carreira, Oportunidade de Formação especializada contínua, Incentivos mensais

How To Pay Student:

[]

Type Of Applicant:

```
{
  "code": "",
  "name": ""
}
```

[Visit Original Post](#) Create Job Post

Figura 4.5: Análise das ofertas recolhidas - Parte 3/3.

Na figura 4.5 está presente um botão "Create Job Post", que navegará para a página do formulário e preenchê-lo-á com os campos identificados, permitindo assim ao administrador criar ofertas mais variadas.

4.1.2 Página das Empresas

Para o caso de uso *UniBotUC*, também foi necessário atualizar a página frontend das empresas, adicionando um botão na página de publicação de trabalhos, denominado "Abrir Chat". Esta opção permite que a caixa de texto seja preenchida, para posteriormente ser analisada pelo UniJobFormsBot e, assim, obter os resultados esperados.

Basic Information

Provide basic information about the job you are offering

Degree of Specialization * ⓘ Job Title * ⓘ

Non-Specialized ▼ Eg. Accountant, HR

Branch * ⓘ

Select Branch ▼

Area of Work * ⓘ Schedule Type * ⓘ

Selection Area ▼ Full-time ▼

Job Description ⓘ

Paragraph ▼ **B** *I* U **☰** **☲**

✕ Close Next >

Uni-Bot ×

Por favor, descreva a oferta de emprego que deseja publicar.

Escreva a sua mensagem...

Enviar

Esperando...

? A good description increases your chances of finding the best candidates

Figura 4.6: Formulário com caixa de texto pronta para comunicar com bot.

Com base numa descrição extensiva e que reflita apropriadamente sobre a oferta de emprego, como por exemplo: "A empresa Company procura assistentes de catering para eventos

presenciais no Porto, entre 1 de agosto e 31 de outubro. Trabalho a tempo inteiro com remuneração de 1.400€/mês (Recibos Verdes). Estão incluídos benefícios como seguro de saúde e eventos de team-building, sendo necessário dar clock-in até 25 metros do local. Vagas limitadas (2). Junta-te a uma equipa dinâmica e profissional!"

Figura 4.7: Formulário com caixa de texto pronta para comunicar com bot.

O texto será enviado para UnijobFormsBot e BotController comunicará com FlowController, o qual processá-la-á. Numa primeira fase, a abordagem possuía múltiplas threads para extrair a informação por secção, realizando uma única chamada à AI sobre cada campo do formulário. Cada campo era validado individualmente da mesma forma, refletindo uma abordagem bastante dispendiosa. Alternativamente, formatou-se uma prompt com todos os campos referentes a uma única secção, extraindo com threads cada uma das secções, utilizando a função representada na figura ??.

```

1 def start_forms_threads_extract(description, job_fields):
2     threads = []
3     lock = threading.Lock()
4
5     def extract_section(section, fields):
6         result = dataProcessor.extractor(description, fields, job_fields,
7             section)
8         with lock:
9             job_fields.update(result)
10
11     for section, fields in forms["steps"].items():
12         thread = threading.Thread(
13             target=extract_section,
14             args=(section, fields)
15         )
16         threads.append(thread)
17
18     for t in threads:
19         t.start()
20
21     for t in threads:
22         t.join()

```

Listing 4.3: Função para a criação de threads que extraem secções de dados em paralelo.

Cada uma irá utilizar a função "extractor" de *DataProcessor*, utilizando a descrição enviada, os campos existentes, o objeto a preencher com os campos recolhidos e a secção atual. Assim, este módulo vai começar por formatar a descrição atual, adicionando a data atual e assim garantir que o modelo tenha uma referência temporal. Esta é enviada para a *AIGateway* através de "ai_call", como representado na figura ??

```
1 def ai_call(func, *args, max_retries=3, delay=2, **kwargs):
2     for attempt in range(1, max_retries + 1):
3         try:
4             return func(*args, **kwargs)
5         except Exception as e:
6             if attempt < max_retries:
7                 time.sleep(delay)
8             else:
9                 return None
```

Listing 4.4: Função para comunicar com uma AI API externa com tentativas e atraso entre falhas.

A função representa um fluxo em que tenta executar a função "func", com os argumentos posicionais e nomeados, "*args" e "**kwargs". De seguida, de acordo com "max_retries", tenta esta ação um máximo de 3 vezes, esperando 2 segundos entre cada tentativa, "delay".

Tendo em conta o formulário da Unilinkr, os campos variam entre datas, números e texto, sendo por isso necessário criar prompts específicas para cada estilo de *input*. Adicionalmente, inicia-se um pedido à AI, para que retorne o resultado em formato JSON, sem texto adicional. Após todas as secções extraídas, estas criam um objeto, *job_fields*, representante dos campos preenchidos. De seguida, *FlowController* realizará o processo de validação, criando novamente threads para cada uma das secções do formulário. O processo de validação tem duas etapas, representadas pelas funções: *_validator* e *_cleaner*. A primeira ação utilizará as regras de negócio definidas em *BusinessValidator*, como por exemplo texto, opção, números positivos, código postal válido e data no formato "dd/MM/aaaa". Caso este campo esteja preenchido mas no formato inválido, o sistema redirecionará o processo para *DataProcessor* que vai tratar de utilizar a AI API para extrair a data, precisa, no formato esperado, do campo recolhido. Se este valor corrigido também for inválido após uma segunda validação, será descartado, tal como outros campos que não respeitem as regras de negócio estabelecidas.

A segunda tarefa é a limpeza de campos desconhecidos, iterando sobre cada uma das secções. As secções sobre informação básica (Basic Information) e localização física da oferta de emprego (Location) são simples e não têm regras de condicionamento. No caso da secção sobre horários e vagas (Schedules & Vacancies), os dados preenchidos vão depender do tipo de horário extraído. Caso seja uma oferta a tempo inteiro, não existem turnos, e a data inicial e final representam o tempo completo do trabalho, enquanto que para outro tipo de ofertas, a informação será sempre correspondente a cada um dos turnos extraídos. Na secção de formas de compensação e benefícios (Compensation), só quando o trabalhador é do tipo *equipas* é que os respetivos nomes serão preenchidos.

Por fim, o resultado é retornado ao *FlowController* que se responsabiliza por criar o modelo do formulário e preenchê-lo de acordo com o pedido efetuado pelo utilizador. Ou seja, num pedido feito pela página das empresas, uma descrição da oferta será providenciada pelo utilizador, redirecionando-a para o caso de uso UniBot. Neste, alguns campos têm de ser formatados para dicionário, como:

- *type_of_applicant* tem de ser formatado para *code-name*;

- `location_id` tem de ser formatado para *label-value*;
- `how_to_pay_student` tem de ser formatado para *id-label-value*.

Com esta estrutura desenvolvida, há a possibilidade do sistema reconhecer e preencher todos os campos existentes de forma precisa.

O processamento de informação pode impedir o sucesso esperado pelo sistema, visto que fatores como: ambiguidade no texto, confusão por parte da AI, também conhecido como alucinações, ou uma descrição demasiado pequena, podem afetar o resultado.

De acordo com a descrição enviada, todos os dados necessários para completar o formulário estão presentes. O único campo que não foi desenvolvido foi "branch- que representa filiais ou unidades de negócio de uma empresa - visto que este tem de ser manualmente definido pelo utilizador a quando da criação da oferta de emprego.

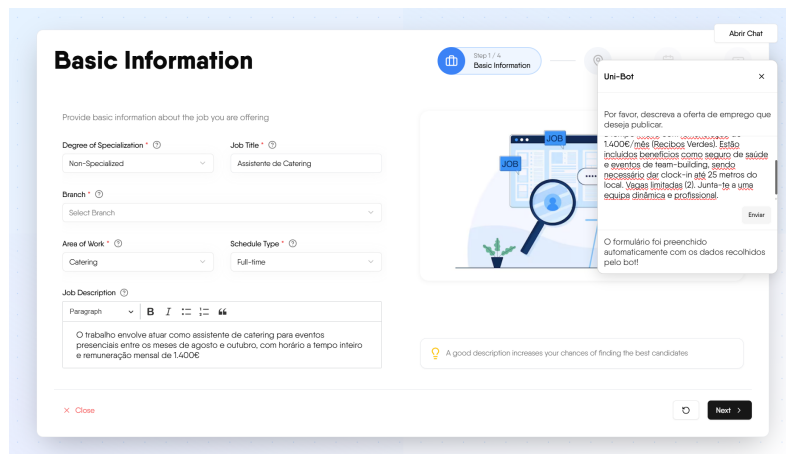


Figura 4.8: Formulário das empresas preenchido (secção 1).

Na secção "*Basic Information*", especificada na figura 4.8, os campos preenchidos relacionam-se com a descrição providenciada. Ser assistente de catering é um trabalho não especializado, o título da oferta também está de acordo com isso, tal como a área de trabalho. O horário foi corretamente identificado como *Full-time*, ou seja, a tempo inteiro, e a descrição do trabalho está relacionada com a que foi enviada ao sistema.

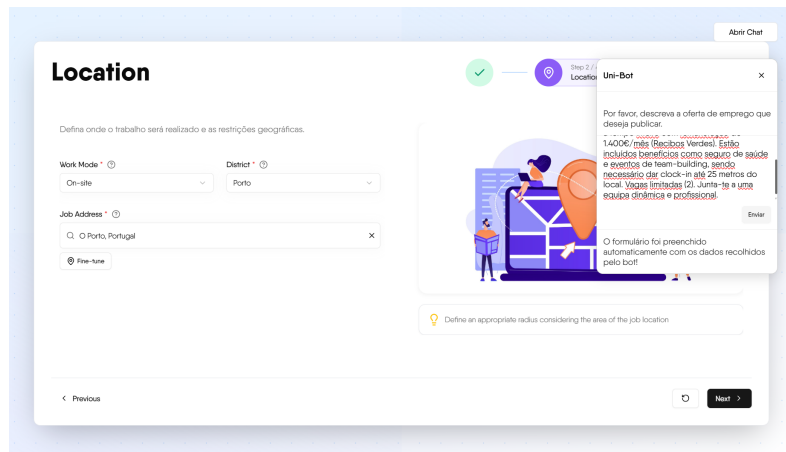


Figura 4.9: Formulário das empresas preenchido (secção 2).

Na secção "*Location*", visível na figura 4.9, o sistema identificou corretamente o modo de trabalho presencial e o distrito correspondente, Porto. Contudo, como a morada não foi estada completamente definida, só foi identificada a morada "O Porto, Portugal".

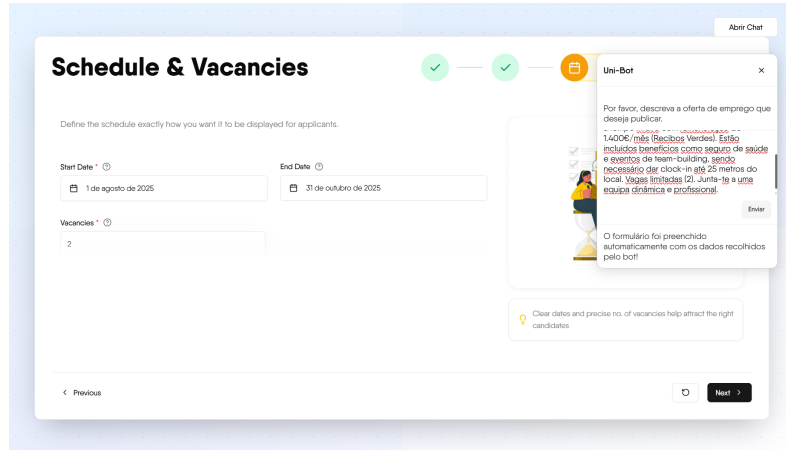


Figura 4.10: Formulário das empresas preenchido (secção 3).

A secção "*Schedules & Vacancies*", representada na figura 4.10, demonstra que o sistema identificou o número de vagas corretamente, juntamente com as datas inicial e final da oferta. O sistema desenvolvido necessita que essas sejam posteriores ao dia de hoje, por isso, se este não receber informação suficiente sobre as datas da oferta de emprego, estas serão no mínimo no dia presente.

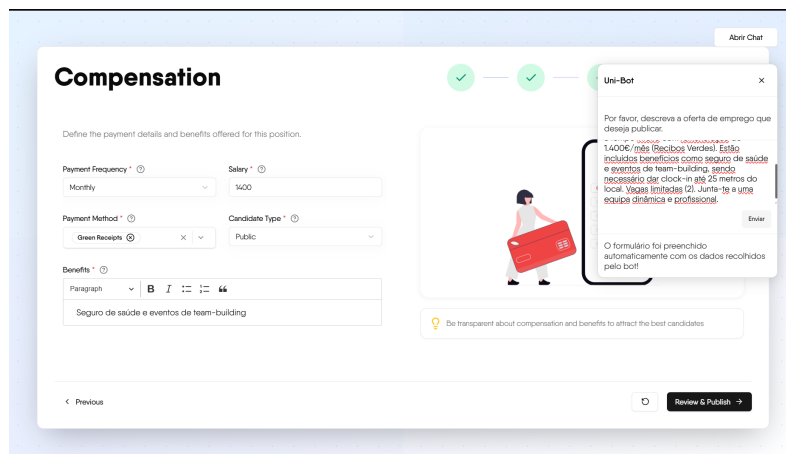


Figura 4.11: Formulário das empresas preenchido (secção 4).

Por último, a secção "*Compensation*", figura 4.11, identificou frequência de pagamento "Mensual", salário "1400", método de pagamento "Recibos Verdes", tipo de candidato "Público" e benefícios, tendo estes sido corretamente extraídos. Desta forma, compreendemos que a implementação do caso de uso *UniBotUC* foi sucedida, garantindo que o sistema consegue extrair e preencher os campos do formulário corretamente, quando se envia uma descrição de emprego extensa e relevante. Na situação em que a informação fornecida é insuficiente ou ambígua, o sistema dificilmente preencherá corretamente todos os campos, exceto aqueles que estiverem identificados.

4.2 Testes

O desenvolvimento de testes, permitiu garantir um projeto seguro e alinhado com os requisitos especificados em 3.2. Foram realizados testes unitários, de integração e End-to-End (E2E). O seu desenvolvimento foi maioritariamente manual, com recurso a algumas tecnologias como *GitHub Copilot*, que permitiram compreender como abranger mais tópicos e formas de testar a comunicação com Websites e AI API externas. Para esta secção, foi utilizada a biblioteca `unittest` e os testes foram desenvolvidos no ambiente de desenvolvimento.

Os testes unitários abrangem todos os componentes, garantindo o sucesso da sua funcionalidade individualmente. Nesta fase, erros e bugs relacionados com a má formação dos dados antes da formatação pelo modelo, como validações e limpeza de campos, foram detetados e sucessivamente corrigidos, de forma a garantir as regras definidas. As figuras ??, ?? e ?? exemplificam este tipo de testes, pois testam as funcionalidades de validação e limpeza dos dados individualmente.

/////////

A listagem 4.5,... focam se na validação dos campos, garantindo que apenas as regras de negócio são validadas quer para texto, opções e datas. A segunda figura, representa um teste sobre a validação da data de acordo, recebendo um formato inválido, corrigindo-o via AI. A figura ?? aborda testes sobre a função de limpeza, garantindo que opções inválidas e texto desconhecido eram limpos, mantendo apenas campos corretos.

/////////

```

1 def test_valida_texto_invalido(self):
2     job_fields = {"company": ""}
3     result = self.processor.validate_fields(job_fields, section="details")
4     self.assertEqual(result["company"], "")
5
6 def test_valida_data_fallback(self):
7     original_date_fixer = self.processor.data_processor.date_fixer
8     self.processor.data_processor.date_fixer = lambda date_str: "31/01/2026"
9
10    job_fields = {"start_date_full_time": "2026/01/31"}
11    result = self.processor.validate_fields(job_fields, section="schedule_and_vacancies")
12    self.assertEqual(result["start_date_full_time"], "31/01/2026")
13    self.processor.data_processor.date_fixer = original_date_fixer
14
15 def test_cleaner_remove_não_sei(self):
16    job_fields = {"company": "não sei", "degree_of_specialization": "não sei"}
17    cleaned = self.processor._cleaner(job_fields, section="details")
18    self.assertEqual(cleaned["company"], "")
19    self.assertEqual(cleaned["degree_of_specialization"], "")
20
21 def test_cleaner_opcao_invalida(self):
22    job_fields = {"degree_of_specialization": "talvez"}
23    cleaned = self.processor._cleaner(job_fields, section="details")
24    self.assertEqual(cleaned["degree_of_specialization"], "")

```

Listing 4.5: Funções para testar o comportamento de `FieldProcessor`.

Desta forma, é possível testar a integração entre componentes do mesmo módulo. As comunicações simuladas entre componentes, apesar de inicialmente detetarem falhas, foram bem-sucedidas perante o resultado final esperado. Por exemplo, na figura ??, testa-se a comunicação entre o módulo AIGateway e a AI API externa, simulando uma resposta válida do sistema.

```

1 @patch('src.ai_gateway.ai_gateway.chat')
2 def test_response_integration(self, mock_chat):
3     mock_chat.return_value = {
4         'message': {'content': '{"campo1": "valor1", "campo2": "valor2"}'}
5     }
6     prompt = "Extrai campos de teste"
7     result = ai_gateway.response(prompt)
8     self.assertEqual(result, '{"campo1": "valor1", "campo2": "valor2"}')
9
10 @patch('src.ai_gateway.ai_gateway.response')
11 def test_ai_extractor_integration(self, mock_response):
12     mock_response.return_value = '{"campo1": "valor1", "campo2": "valor2"}'
13     section = "secao"
14     section_steps = [
15         {"question": "Q1", "style": "S1", "field": "campo1"},
16         {"question": "Q2", "style": "S2", "field": "campo2"}
17     ]
18     description = "Descrição de teste."
19     result = ai_gateway.ai_extractor(section, section_steps, description)
20     self.assertIsInstance(result, dict)
21     self.assertIn("campo1", result)
22     self.assertIn("campo2", result)

```

Listing 4.6: Teste de integração entre o AIGateway e a API externa de AI.

Por fim, os testes E2E permitiram validar a operação do sistema perante todos os módulos desenvolvidos. Estes testaram o funcionamento do *router*, do seu redirecionamento e do processamento total dos dados, juntamente com a chamada à AI API, para cada um dos casos de uso, tal como representado nas figuras ?? e ??, utilizado `TestClient(app)`. Esta funcionalidade da FastAPI, cria um cliente HTTP teste, para simular chamadas à API, sem necessidade de correr o servidor local.

Desta forma, a primeira figura testa os resultados possíveis para o caso de uso *UniBotUC*, garantindo o seu sucesso quando há uma descrição, erro se esta não existir ou `None`, que representa um valor inválido, caso a descrição seja ambígua ou insuficiente. Por outro lado, a segunda figura testa o caso de uso *UniScraperUC*, simulando a recolha de várias ofertas de emprego, juntamente com a criação do seu modelo de trabalho. São testadas as situações com resultados válidos, `None` caso o processo todo falhe, ou uma lista vazia se os resultados não forem recolhidos.

```

1 def test_chat_endpoint_success(self):
2     test_description = "Café, serviço bar e com experiência"
3
4     with patch('src.interactor.bot_controller.BotController.extract_job')
5     as mock_extract:
6         mock_extract.return_value = {
7             "job_title": "Serviço Bar Experiente",
8             "specialization": "Specialized",
9             "area_of_work": "Bar Service"
10         }

```

```

10
11     response = self.client.post("/chat", json={"description":
12         test_description})
13
14     assert response.status_code == 200
15     assert "job_model" in response.json()
16     mock_extract.assert_called_once_with(description=test_description
17 )

```

Listing 4.7: Explode de Teste End-to-End para o *UniBotUC*.

```

1 def test_scraper_endpoint_success(self):
2     mock_result = [
3         {"job_title": "Data Scientist", "location": "Porto"},
4         {"job_title": "Web Developer", "location": "Lisboa"}
5     ]
6
7     with patch('src.interactor.bot_controller.BotController.scrape_jobs')
8     as mock_scrape:
9         mock_scrape.return_value = mock_result
10
11         response = self.client.get("/scraper")
12
13         assert response.status_code == 200
14         assert response.json() == mock_result

```

Listing 4.8: Teste do endpoint /scraper com controlo de erros.

Adicionalmente, foram também testadas situações que levantam erros, de forma a garantir a finalização correta do sistema, sem este *crashar*, tal como representado na figura ?? . Este, sugerido pelo GitHub Copilot, permite testar as situações em que o processo pode falhar, mas necessita de garantir a sua conclusão global, avaliando essas situações.

```

1 def test_bot_controller_extract_job_exception(self):
2     with patch.object(self.bot_controller.flow_controller, 'run_flow') as
3     mock_run:
4         mock_run.side_effect = ValueError("Invalid job description format
5 ")
6         result = self.bot_controller.extract_job("corrupted input")
7         self.assertIsNone(result)
8
9 def test_bot_controller_scrape_jobs_exception(self):
10     with patch.object(self.bot_controller.scaper_controller, 'scraper')
11     as mock_scaper:
12         mock_scaper.side_effect = ConnectionError("Network unavailable")
13         result = self.bot_controller.scrape_jobs()
14         self.assertIsNone(result)
15
16 def test_flow_controller_threading_exception(self):
17     with patch('src.interactor.flow_controller.
18 start_forms_threads_extract') as mock_extract:
19         mock_extract.side_effect = threading.ThreadError("Thread creation
20 failed")
21         result = self.flow_controller.extract_job_fields("test
22 description")
23         self.assertIsInstance(result, dict)
24
25 def test_flow_controller_validation_exception(self):
26     test_fields = {"job_title": "Developer", "invalid_field": None}
27     with patch('src.interactor.flow_controller.
28 start_forms_threads_validate') as mock_validate:

```

pagos testes e links?

```
22 mock_validate.side_effect = KeyError("Required field missing")
23 result = self.flow_controller.validate_fields(test_fields)
24 self.assertEqual(result, test_fields)
```

Listing 4.9: Teste para controlo global de exceções.

Por fim, aplicando estes estilos de testes para todas as funcionalidades, é possível analisar a sua cobertura, tal como representado na figura 4.12. Verifica-se uma cobertura total de 84%, sugerindo que a maior parte da base do código está testada, revelando a eficiência dos testes.

Name	Stmts	Miss	Cover
-----	-----	-----	-----
TOTAL	1871	307	84%

Figura 4.12: Cobertura total de testes (84%).

Apesar da boa cobertura, nota-se que 307 instruções não são exercitadas pelos testes, o que justifica a ausência de uma cobertura total de 100%. Estas podem representar fluxos alternativos ou raros e código menos crítico; contudo, podem ser testadas para assegurar uma robustez mais sólida. Importa sublinhar que a cobertura de testes é uma métrica indicativa, e não deve ser encarada como um objetivo absoluto. Nem todo o código precisa de ser testado com a mesma intensidade, portanto, as funcionalidades analisadas devem focar-se em testar a lógica, manipulação de dados e interações externas, garantindo o seu sucesso.

Os resultados obtidos permitem garantir o sucesso do sistema, tanto em termos de fiabilidade como de conformidade com os requisitos funcionais definidos em 3.2.1. A abordagem adotada contribui significativamente para detetar falhas atempadamente e aumentar a robustez da solução. Por fim, o processo de testes também contribuiu para a estabilidade do sistema, revelando-se uma solução que transparece confiança e com facilidade de manutenção. A aplicação sistemática de testes ao longo do desenvolvimento revelou-se essencial para a deteção precoce de erros, facilitando o processo iterativo de melhoria contínua da solução.

4.3 Avaliação da solução

A solução implementada aborda a estrutura especificada no capítulo 3, seguindo uma arquitetura modular, a qual facilita a organização, implementação e testes, do sistema. Ambos os casos de uso, *UniScraperUC* e *UniBotUC* foram totalmente desenvolvidos, testados e integrados com a plataforma da Unilinkr. A integração realizou-se através da implementação de duas páginas *frontend*, uma para o administrador e outra para as empresas.

A página web do admin permite a realização completa do caso de uso *UniScraperUC*, visto que:

- Aplica técnicas Web Scraping para extrair ofertas da web;
- Extrai o máximo de campos identificados, de forma a produzir uma descrição completa.

O seu desenvolvimento no sistema UnijobFormsBot e demonstração *frontend* permite, de forma intuitiva, dar a opção do utilizador identificar o nome da oferta, redirecionar para o *link* da oferta original e saber se todos os campos do formulário foram preenchidos, podendo, consequentemente, publicar essa oferta. Desta forma, a solução desenvolvida está bastante sólida para a tarefa em questão, pois alinha-se com o desenho definido, tendo obtido resultados positivos.

O segundo caso de uso, *UniBotUC*, foi completamente desenvolvido e, da mesma forma que no primeiro, integra com a plataforma Unilinkr. Nesta, foi adicionada uma página que permite utilizar o resultado do sistema, preenchendo de forma completa e corretamente o formulário. Garante-se assim que o caso de uso *UniBotUC*:

- Utiliza um sistema com NLP, capaz de extrair campos de uma oferta de emprego, baseado numa descrição;
- Modela os campos do formulário da Unilinkr, de acordo com a descrição enviada.

Nesta demonstração *frontend*, o preenchimento do formulário é realizado automaticamente assim que o sistema conclui a modelação dos dados. Se os resultados estiverem parcialmente incorretos, os campos com erros são limpos, enquanto os corretamente preenchidos serão mantidos. Atingindo definitivamente o resultado pretendido.

Os resultados obtidos estão alinhados com os objetivos previamente definidos, o que demonstra eficácia na extração e estruturação de dados. Confirmamos que, perante diferentes tipos de *inputs* e com as validações apropriadas, garantimos o preenchimento do formulário com os dados pretendidos, reforçando a robustez do sistema. Contudo, este tem aspetos a melhorar, principalmente na área da eficiência, que se revelou àquém das expectativas, devido ao custo das chamadas à AI API externa e à recolha de dados da web, iterando sobre todas as ofertas disponíveis. Alguns casos de uso para a extração de uma oferta singular no website da *Net Empregos* chegaram a demorar cerca de 15 minutos a ser concluído. Salienta-se que esta performance está inerente à interação com Ollama, o que impacta negativamente a performance global do sistema.

A validação dos neg. não funciona??

Capítulo 5

Conclusões

Neste capítulo, dar-se-á a análise do projeto, em relação aos requisitos estabelecidos em 3.2.1 e em 3.2.2. Desta forma,

5.1 Objetivos concretizados

Conclui-se assim o projeto sobre Sistema Automático de Publicação de Ofertas de Emprego com Recurso a Inteligência Artificial e Web Scraping, com a implementação de um sistema capaz de preencher formulários de ofertas de emprego de forma automática. A aplicação de um sistema com duas funcionalidades que utilizam tecnologias emergentes, tal como, Web Scraping e NLP, principalmente LLM.

A tabela 5.1 dispõe dos estados de cada um dos objetivos estipulados no capítulo 3, e tal como referido em 4.3.

Estado final dos objetivos estipulados:

Objetivo	Estado
Extraír automaticamente dados relevantes, partindo de descrições de emprego e recorrendo a técnicas NLP.	Completo
Implementar mecanismos de Web Scraping, para recolher ofertas de emprego de fontes externas, semelhantes à plataforma Unilinkr.	Completo
Estruturar os dados extraídos, baseado nos campos exigidos pelo formulário da plataforma Unilinkr.	Completo
Automatizar o preenchimento de formulário para novas ofertas de trabalho.	Completo
Melhorar a eficiência do processo de publicação de ofertas.	Incompleto

Tabela 5.1: Objetivos do projeto e respetivo estado de conclusão

O primeiro e o terceiro objetivo representam-se concluídos com a implementação do caso de uso UniBot, e o segundo através do caso de uso UniScraper. O seu desenvolvimento garante comunicação com o utilizador, com uma AI API externa e a extração de HTML da web.

A integração de ambos os casos com a plataforma está representada no quarto objetivo, estando igualmente concluído. Esta tarefa garante que ambos os casos de uso podem ser contactados individualmente, apesar de entre si comunicarem, e permitir o *display* do resultado nos frontends da Unilinkr.

Por último, a melhora da eficiência do processo de publicação de ofertas não foi totalmente atingida. De forma a concluir este objetivo, seguiu-se uma estratégia com threads e prompts

As partes
Unif e
Uniscraper
são
de

bem estruturadas, capazes de dividir tarefas eficazmente. De facto, isto garante um processamento mais rápido e em paralelo, mas devido à exigência das chamadas à AI API externa, o processo revelou-se lento.

5.2 Limitações e trabalho futuro

O trabalho realizado possui uma base robusta para progredir e evoluir, tornando-se uma ferramenta muito mais abrangente. O utilizador ao comunicar com o sistema, consegue consequentemente comunicar com uma AI API, capaz de responder concretamente, de acordo com os pedidos realizados. Um agente mais desenvolvido e potente, como GPT-4, tal como referido no capítulo 2, por exemplo, será possível tornar essa ação mais eficiente. Assim, a resposta chegaria ao utilizador de forma mais detalhada e acertada.

A escolha de operar com LLaMa continua a ser bastante sólida, tendo em conta a complexidade do sistema, mas tem pontos a focar. Por um lado, os campos demoram um tempo considerável a serem recolhidos, afetando a performance do sistema e, consequentemente, a satisfação do utilizador. Por outro, permite ao sistema evoluir e atingir uma solução mais robusta e capaz, criando um sistema que compreenda mais funcionalidades e interação com o utilizador. Esta poderia ainda progredir para ferramentas digitais como *WhatsApp* [33], uma aplicação que funciona como "mensagens on-line" e que possui uma ferramenta *WhatsApp Business Platform* [34] para empresas, a qual pode ser integrada com a plataforma através de uma *WhatsApp API*, permitindo assim melhorar ainda mais a satisfação dos utilizadores.

O desenvolvimento na área de sistemas NLP utilizando está em crescimento e várias oportunidades otimizadas poderão emergir no mercado. Esta poderão atingir elevados níveis de satisfação, quer pela capacidade de resposta do sistema, quer pelo o nível de interação com o utilizador. A publicação de ofertas de emprego através dessas tecnologias poderá ser um objetivo para uma evolução da ferramenta *UniBotUC*.

O sistema Web Scraping também apresenta robustez para os websites tratados e na integração com o sistema NLP. Esta ferramenta também tem capacidades para crescer, tendo em conta que existem múltiplos websites que compreendem ofertas de emprego semelhantes às da Unilinkr, que podem ser analisadas e extraídas pelo sistema UnijobFormsBot. Contudo, a complexidade desta operação diferencia-se entre websites, sendo um trabalho exigente para garantir o sucesso de cada uma das ofertas. A criação de cada vez mais mecanismos *anti-scraping* dificulta o progresso do sistema, mas ferramentas Web Scraping emergentes que possuam redes diversificadas de *proxies*, como *BrightData* [35], ou sejam mais especializadas na extração de dados, como *ScrapingBee* [36], são alternativas que podem ser avaliadas, desenvolvidas e integradas em UnijobFormsBot, expandindo o sistema, garantindo uma maior extração de ofertas na web.

Devido à implementação entre os dois casos de uso, que permite a modelação dessas ofertas no formato do formulário esperado na página do administrador, o sistema terá de iterar sobre inúmeras ofertas, o que é bastante custoso. A implementação de um agendador de tarefas em sistemas Unix, como *cron*, poderia garantir a extração semanal de tarefas em horários noturnos, não ocupando tempo ao administrador durante o dia, e, desta forma, obter as ofertas mais recentes no sistema.

Sendo assim, apesar das limitações abordadas, o sistema tem capacidades para evoluir e aumentar a sua performance, quer para o administrador da plataforma, quer para as empresas

registadas na plataforma, permitindo, futuramente, garantir que mais trabalhos e os trabalhos mais recentes, estarão disponíveis na Unilinkr, oferecendo mais oportunidades para jovens estudantes.

5.3 Apreciação final

O projeto UnijobFormsBot, apesar de ter representado um desafio significativo, foi simultaneamente uma oportunidade para o crescimento acadêmico, técnico e pessoal. A implementação de tecnologias emergentes, como LLM e Web Scraping, ultrapassou alguns dos conhecimentos lecionados no âmbito do curso, principal na área da Inteligência Artificial, exigindo uma abordagem autodidata, persistência e constante adaptação a novas ferramentas e metodologias.

Durante todo este percurso, consegui consolidar e aplicar conhecimentos adquiridos nas várias unidades curriculares da licenciatura de engenharia informática do ISEP [37], nomeadamente em áreas de desenvolvimento de software, processamento de linguagem natural, integração de sistemas e estruturas de dados, que se tornaram relevantes quanto mais se aprofundou esta área. Além disso, o desenvolvimento de competências transversais, tal como gestão de tempo, resolução de problemas complexos e espírito crítico, foi possível, permitindo-me crescer pessoal e tecnicamente.

A estrutura modular adotada, revelou-se uma mais-valia, porque permitiu que o sistema desenvolvido fosse tanto escalável como adaptável, a diferentes contextos e plataformas. Desta forma, reforça uma visão focada na manutenção e evolução contínua do sistema. Apesar de obstáculos como limitações de desempenho, causadas por chamadas externas à API, e os desafios inerentes ao *scraping* de websites, devido a mecanismos anti-*scraping* - existentes em *Net Empregos* [31] - foi possível encontrar soluções viáveis que permitiram desenvolver um sistema funcional e com potencial para utilização real.

Esta experiência permitiu-me elevar padrões de exigência e ter uma perspetiva mais clara de como realmente é o mercado de trabalho. Permitiu-me crescer enquanto engenheiro, pois tive de me informar sobre as mais variadas tecnologias, para tomar decisões informadas, e a lidar com sistemas complexos, tal como representado no sistema UnijobFormsBot. Concluo este projeto com um sentimento de realização e orgulho pelo percurso percorrido, ciente de que esta é apenas uma etapa inicial de um percurso que pretendo continuar a desenvolver na área da inteligência artificial e automatização de processos.

Bibliografia

- [1] Unilinkr. *Unilinkr*. <https://unilinkr.net>. (Acedido em abril de 2025). 2024.
- [2] Computer Weekly. *O que é e como funciona a web scraping*. <https://www.computerweekly.com/br/definicoe/0-que-e-e-como-funciona-a-web-scraping>. (Acedido em maio de 2025). 2024.
- [3] Trupti V. Udupure, Ravindra D. Kale e Rajesh C. Dharmik. *Study of Web Crawler and its Different Types*. <https://d1wqtxts1xzle7.cloudfront.net/71643269/1ca5ac22427b6ad56938e44f88098libre.pdf>. (Acedido em abril de 2025). 2014.
- [4] RTP. *Última Hora*. https://www.rtp.pt/noticias/economia/covid-19-a-situacao-ao-minuto-do-novo-coronavirus-no-pais-e-no-mundo_e1238269. (Acedido em julho de 2025). 2025.
- [5] RTP. *Última Hora*. https://www.rtp.pt/noticias/mundo/guerra-na-ucrania-a-evolucao-do-conflito-ao-minuto_e1616103. (Acedido em julho de 2025). 2025.
- [6] Wikipedia contributors. *Beautiful Soup (HTML parser)*. [https://en.wikipedia.org/wiki/Beautiful_Soup_\(HTML_parser\)](https://en.wikipedia.org/wiki/Beautiful_Soup_(HTML_parser)). (Acedido em abril de 2025). 2025.
- [7] Selenium. *Selenium*. <https://www.selenium.dev>. (Acedido em abril de 2025). 2004.
- [8] Puppeteer. *puppeteer*. <https://pptr.dev>. Acedido em abril de 2025. 2017.
- [9] Daniel Glez-Peña et al. *Web scraping technologies in an API world*. <https://academic.oup.com/bib/misc/15/5/788/2422275>. (Acedido em abril de 2025). 2014.
- [10] LinkedIn. *LinkedIn*. www.linkedin.com. (Acedido em junho de 2025). 2022.
- [11] USA. *HIQ LABS, INC. V. LINKEDIN CORPORATION, No. 17-16783 (9th Cir. 2022)*. <https://law.justia.com/cases/federal/appellate-courts/ca9/17-16783/17-16783-2022-04-18.html>. (Acedido em junho de 2025). 2022.
- [12] Wikipedia. *Alan Turing*. https://pt.wikipedia.org/wiki/Alan_Turing. (Acedido em julho de 2025). 2025.
- [13] Codecademy Team. *History of Chatbots*. <https://www.codecademy.com/misc/history-of-chatbots>. (Acedido em maio de 2025). 2025.
- [14] Analytics India Magazine. *https://analyticsindiamag.com/ai-features/story-eliza-first-chatbot-developed-1966/*. <https://analyticsindiamag.com/ai-features/story-eliza-first-chatbot-developed-1966/>. (Acedido em junho de 2025). 2016.
- [15] Facebook. *Facebook*. <https://www.facebook.com>. (Acedido em junho de 2025.) 2004.
- [16] Fortune Business Insights. *Tamanho do mercado de chatbot, participação e insights do setor, por componente (kit de desenvolvimento de plataforma/software e serviços), por implantação (nuvem e local), por aplicativo (site, contact centers, mídias sociais e aplicativos móveis), por setor (bancário), Serviços Financeiros e Seguros (BFSI), Varejo e comércio eletrônico, Tecnologia da Informação (TI) e Telecomunicações, Mídia e Entretenimento, Saúde e Outros e Previsão Regional, 2020-2027*. <https://www.fortunebusinessinsights.com/pt/conversational-ai-market-109850>. (Acedido em junho de 2025). 2025.
- [17] Mya Systems. *Mya Systems*. <https://www.linkedin.com/company/myasystems/>. (Acedido em maio de 2025). 2016.
- [18] Paradox. *Paradox*. <https://www.paradox.com>. (Acedido em maio de 2025). 1989.

- [19] Julia-Astrid Moldt et al. *Chatbots for future docs: exploring medical students' attitudes and knowledge towards artificial intelligence and medical chatbots*. <https://www.tandfonline.com/doi/full/10.1080/10872981.2023.2182659>. (Acedido em maio de 2025). 2023.
- [20] Josip Vrdoljak et al. *A Review of Large Language Models in Medical Education, Clinical Decision Support, and Healthcare Administration*. <https://pmc.ncbi.nlm.nih.gov/miscs/PMC11942098/>. (Acedido em maio de 2025). 2025.
- [21] tessdoc. *Tesseract User Manual*. <https://github.com/tesseract-ocr/tessdoc>. (Acedido em maio de 2025). 2019.
- [22] PaddlePaddle Authors. *PaddleOCR, Awesome multilingual OCR toolkits based on PaddlePaddle*. <https://github.com/PaddlePaddle/PaddleOCR>. (Acedido em maio de 2025). 2020.
- [23] Senthil E. *Building a Receipt Text Extraction App with Python and LLM: Unleashing the Power of LLMs*. <https://levelup.gitconnected.com/building-a-receipt-text-extraction-app-with-python-and-llm-unleashing-the-power-of-llms-b5e363bbb7df>. (Acedido em abril de 2025). 2024.
- [24] Juergen Schmidhuber. *Deep Learning*. http://www.scholarpedia.org/misc/Deep_Learning. (Acedido em maio de 2025). 2015.
- [25] Nadia Mushtaq Gardazi et al. *BERT applications in natural language processing: a review*. <https://link.springer.com/misc/10.1007/s10462-025-11162-5>. (Acedido em maio de 2025). 2025.
- [26] N.M. Gardazi, A. Daud e M.K. Malik. *mT5: A massively multilingual pre-trained text-to-text transformer*. <https://arxiv.org/abs/2010.11934>. (Acedido em abril de 2025). 2020.
- [27] Min Zhang e Juntao Li. *A commentary of GPT-3 in MIT Technology Review 2021*. <https://www.sciencedirect.com/science/misc/pii/S2667325821002193>. (Acedido em abril de 2025). 2021.
- [28] Z. Liu et al. *Instruct-Code-Llama: Improving Capabilities of Language Model in Competition Level Code Generation by Online Judge Feedback*. https://link.springer.com/chapter/10.1007/978-981-97-5669-8_11. (Acedido em abril de 2025). 2024.
- [29] Google. *601 real-world gen AI use cases from the world's leading organizations*. <https://cloud.google.com/transform/101-real-world-generative-ai-use-cases-from-industry-leaders>. (Acedido em abril de 2025). 2025.
- [30] Kishor Kumar Reddy C. *A Text Mining using Web Scraping for Meaningful Insights*. <https://iopscience.iop.org/misc/10.1088/1742-6596/2089/1/012048/pdf>. (Acedido em maio de 2025). 2021.
- [31] Net Empregos. *Net Empregos*. <https://www.net-empregos.com>. (Acedido em maio 2025). 1997.
- [32] THE HAIR SHOP. *Sales Assistant (M/F) - THE HAIR SHOP - TAVIRA PLAZA*. <https://www.net-empregos.com/13818783/sales-assistant-m-f-the-hair-shop-tavira-plaza/>. (Acedido em junho de 2025). 2025.
- [33] WhatsApp. *WhatsApp*. <https://www.whatsapp.com>. (Acedido em junho de 2025). 2009.
- [34] WhatsApp. *WhatsApp Business*. <https://business.whatsapp.com/products/business-platform>. (Acedido em junho de 2025). 2018.
- [35] Bright Data. *Limitless web data infrastructure for AI BI*. <https://brightdata.com>. (Acedido em junho de 2025). 2021.
- [36] ScrapingBee. *Tired of getting blocked while scraping the web?* <https://www.scrapingbee.com>. (Acedido em junho de 2025). 2019.
- [37] ISEP. *Instituto Superior de Engenharia do Porto*. <https://www.isep.ipp.pt>. (Acedido em abril de 2025).