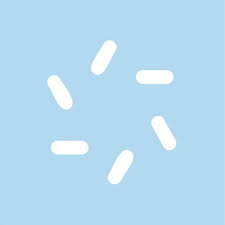
**RELATÓRIO PROJETO GRUPO**

**AMBIENTES E LINGUAGENS DE PROGRAMAÇÃO PARA CIÊNCIA DE DADOS**

PROF: TIAGO BATISTA

PROF: PEDRO HENRIQUES

Grupo 10

Alexandre Almeida, A106862

Fernando Leite, A106822

Filipa Barroso Teixeira, A106864

2024/2025

Este projeto foi desenvolvido no âmbito da cadeira de Ambientes e Linguagens de Programação para Ciência de Dados (ALPCD) e aborda a integração de uma API e técnicas de WebScraping, para a aquisição e processamento de informação sobre diversas oportunidades de emprego. A relevância deste projeto está presente na utilização de ferramentas computacionais de modo a facilitar a pesquisa e análise de dados sobre o mercado de trabalho.

O trabalho divide-se em duas grandes componentes, sendo elas a interação com a API do ITJobs, de modo a obter dados estruturados sobre determinadas vagas, e o Web Scraping de websites que complementam e enriquecem as informações presentes na API.

Para que o projeto seja conseguido e concluído de forma eficiente e prática, foram implementadas algumas bibliotecas, como por exemplo: “*typer*”, para a construção de uma interface de linha de comando (CLI) intuitiva; “*Requests*”, para que possam ser feitas requisições à API; “*BeautifulSoup*”, para processar e extrair dados das páginas web; “*sync\_playwright*”, para uma navegação em páginas que requerem interações dinâmicas; “*csv*”, para a exportação dos resultados em arquivos csv. Por fim a biblioteca “*datetime*” serve para manipular datas e horas de forma eficiente. Estas bibliotecas garantiram, então, a flexibilidade na manipulação dos dados e a permissão de consulta para o aprimorar das informações.

Ao longo do desenvolvimento do projeto, reparamos que havia muitas situações, onde se escreviam e reescreviam as mesmas linhas de código e, por isso, achamos por bem, criar funções auxiliares de modo a otimizar o mesmo.

Estas funções são “*request\_api*”, que lida com as requisições feitas à API, implementando paginação e retornando resultados formatados; “*request\_website*” que concretiza requisições HTTP para websites; por último, “*cria\_csv*”, que exporta dados em formato CSV.

Havendo funções auxiliares, há também as funções principais, que são o grande foco e motivo, pelo qual este projeto se realizou.

Temos a função “*top*” que lista as N vagas mais recentes publicadas no ITJobs; “*search*” que filtra as vagas do tipo “full-time” por localidade e empresa; “*salary*” que extrai as informações salariais de uma vaga específica, usando expressões regulares; “*skills*” que procura vagas que requerem determinadas competências/habilidades dentro de um intervalo de tempo; “*contar\_vagas\_localizacao*” , uma função extra, que se decidiu fazer focando-se na contagem de vagas disponíveis por localização; “*get*” que consiste em exibir informações detalhadas de uma vaga com base nos dados da API e com a complementação dos dados do website “AmbitionBox”; “*statistics*”, que mostra a contagem de vagas por zona junto da possibilidade de serem agrupadas por título, tipo ou ambos; “*list\_skills*” que realiza web scraping no AmbitionBox para identificar as principais skills associadas a um trabalho com contagem de ocorrências; “*get2*”que complementa a função “*get*”, havendo um acrescento de informação para cada vaga com base noutro website alternativo; “*vagas\_recomendadas*”, sendo ela a última função e também extra, que recomenda um número limitado de vagas relacionadas, com base em palavras-chave extraídas do título da vaga.

Focando e explicando agora cada função de forma mais detalhada e pormenorizada, o desenvolvimento de cada uma foi realizado da seguinte forma.

A função *top* tem como objetivo obter os N trabalhos mais recentes por meio de uma requisição à API, com a opção de exportar os resultados para um arquivo CSV. Assim como parâmetros de entrada esta função apresenta: n (int): define o número de trabalhos a serem retornados (deve ser maior que 0); csv (bool, opcional, Default: False): indica se os resultados devem ser exportados para um arquivo CSV. Por outro lado, como parâmetros de saída, a função: exibe os resultados retornados pela API no terminal; gera um arquivo CSV com os resultados (se csv=True).

Para além disto, os procedimentos de resolução são os seguintes:

* Recebimento de Parâmetros: n e csv;
* Construção dos Parâmetros da API: Um dicionário com o parâmetro limit é criado para especificar o número de resultados desejado;
* Validação do Valor de n: Caso n seja igual a 0, a função exibe uma mensagem de erro indicando que o valor deve ser maior que 0 e encerra a execução;
* Requisição à API: A função utiliza request\_api com o método list e os parâmetros definidos anteriormente;
* Validação da Resposta: A resposta é verificada para garantir que o número de trabalhos retornados seja suficiente: Se o número de resultados for menor que n, exibe-se uma mensagem informando a quantidade disponível atualmente;
* Exportação para CSV: Se o parâmetro csv for True, a função chama *cria\_csv* para salvar os resultados em um arquivo CSV.

Um exemplo de uso desta função é,  “python jobs.py 10 --csv”

Caso o número de trabalhos pedidos seja maior do que o número de trabalhos na API ,a função request\_api também garante que a execução pare depois de chegar ao numero de trabalhos pretendidos.

A função *search* permite aceder a trabalhos específicos com base em critérios como localidade, empresa e limite de resultados. Esta funcionalidade é especialmente útil para quem deseja filtrar vagas de trabalho com determinadas especificações.

A resolução desta função consistiu em:

* definir os parâmetros, sendo eles, localidade(str), empresa(str), limit(int) e csv(por default é FALSE, ou seja, os resultados não serão exportados para csv);
* configurar os parâmetros, que resumidamente significou em criar um dicionário com o limite de 1500 (número suficiente para percorrer todos os trabalhos da API) e o tipo de trabalho (“type: 1”, referindo-se a trabalhos full-time, como pedido no enunciado);
* requisição à API, sendo utilizada a função auxiliar “*request\_api*”, explicada anteriormente, de modo a recolher os dados da API com a filtragem dos parâmetros dados pelo utilizador, ou seja, os dados que correspondam exatamente ao nome da empresa fornecido, localizada na determinada região e com o limite de resultados definido;
* exibição dos dados, foi a última etapa desta função e consiste simplesmente em exibir os resultados obtidos após as filtragens e caso a opção csv seja TRUE, chama a função “*cria\_csv*”, também já explicada anteriormente para exportar os dados.

Um exemplo de uso desta função é,  “python jobs.py search Braga EmpresaX 4 --csv”

Por fim, caso nenhum resultado seja obtido, há uma exibição ao utilizador que lhe indique isso, existe a garantia de que a filtragem não seja sensível a maiúsculas/minúsculas, e ainda se o limite posto pelo utilizador for 0, é dado um aviso sem se quer se realizar a procura dos trabalhos.

A função *Salary* tem como objetivo obter informações salariais de uma vaga de emprego com base num jobID. Como parâmetros de entrada na função salary temos o seguinte: jobid (int): identificação da vaga de emprego. Como parâmetros de saída temos o seguinte:  exibição de informações sobre o salário ou uma mensagem a indicar a ausência de dados.

Assim sendo, os procedimentos para a execução desta função são os seguintes:

* Recebe o ID do trabalho como entrada;
* Envia uma requisição à API utilizando a função request\_api com o método get no URL e o parâmetro id;
* Verifica se a resposta da API é válida:
  + Se a resposta é None, informa que os dados do job não foram obtidos;
  + Se contém um erro, notifica que a vaga não foi encontrada;
* Caso a resposta seja válida:
  + Verifica se o campo wage está presente;
    - Se o campo existir, exibe o valor do salário;
    - Caso contrário, utiliza uma expressão regular para procurar informações salariais na descrição.

Um exemplo de uso desta função é,  “python jobs.py 12345”

Posto isto, a função *Salary* permite obter informações salariais de forma eficiente, verificando dados diretamente na API ou extraindo valores da descrição com expressões regulares. O tratamento de erros assegura a execução. É uma solução prática para identificar salários em vagas de emprego.

A função *skills* foca-se em encontrar os trabalhos que exigem skills específicas dentro de um intervalo de tempo definido, onde as etapas de resolução são as seguintes:

* definir os parâmetros, sendo eles, skills (list[str]), data\_inicial (str), data\_final (str), csv onde mais uma vez o default é a não exportação dos resultados;
* validação das datas, onde se converteram as mesmas para o formato datetime, verifica se estão no formato correto (YYYY-MM-DD) e se são coerentes, ou seja, se a data inicial é anterior à data final;
* preparação dos parâmetros, onde se juntam as skills da lista em uma única string separada por vírgulas;
* requisição à API, onde se usa de novo a função auxiliar “*request\_api*”, para recolher os dados da API com a filtragem dos parâmetros dados, ou seja, os dados que estejam dentro do intervalo de datas especificado e que contenha todas as skills dadas;
* exibição dos dados, que corresponde ao que já foi explicado na função abordada anteriormente.

Um exemplo de execução é “python jobs.py skills ["Python", "SQL"] 2023-01-01 2023-06-30”.

Para finalizar existe ainda a possibilidade de nenhum trabalho ser encontrado e então é enviada uma mensagem a afirmar isso.

Contar o número total de vagas disponíveis numa determinada localização é o grande objetivo desta funcionalidade extra, cujo nome dado foi “*contar\_vagas\_localizacao*”.

O procedimento de construção desta função foi o seguinte:

* definir parâmetros, que neste caso é apenas um, sendo ele localização (str);
* configurar os parâmetros, onde se definiu um limite de 1500 resultados;
* requisição à API, com a função auxiliar “*request\_api*”, para recolher os dados da API segundo a filtragem baseada no parâmetro que o utilizador deu;
* exibição dos dados, onde está presente o número total de vagas obtidas, ou uma mensagem a indicar que nenhuma vaga foi encontrada caso seja isso sucedido.

Um exemplo de uso é o seguinte: “python jobs.py contar\_vagas\_localizacao Lisboa”.

Nesta funcionalidade existe também a garantia de que a comparação de localizações seja insensível a maiúsculas/minúsculas.

A função *get* teve como objetivo a extração de informações sobre uma empresa específica, em particular os seguintes pontos: classificação(“rating”), descrição geral da empresa e lista de benefícios/políticas oferecidos.

Como parâmetros de entrada temos o job\_id(int): id do trabalho a ser pesquisado, e CSV, caso seja TRUE, o output é guardado num CSV. Como parâmetros de saída temos um dicionário com,para alem das informações da API, as seguintes chaves: rating(str):classificação da empresa; descrição(str): descrição geral; benefícios (list): lista de benefícios.

Desta forma, os procedimentos para a execução desta função são os seguintes:

* Recebe o id do trabalho como entrada;
* Constrói uma URL de pesquisa;
* Realiza uma requisição HTTP GET para obter a página HTML correspondente;
* Utiliza a biblioteca BeautifulSoup para processar o HTML e buscar elementos específicos:
  + Classificação: Procurada na tag <span> com classe correspondente;
  + Descrição: Extraída das tags <div> com classe correspondente;
  + Benefícios: Listados em tags <div> com classe correspondente.
* Utiliza try-except para lidar com erros e garantir a robustez do código;
* Retorna um dicionário com as informações obtidas ou valores padrão caso algum dado esteja indisponível.

Desta forma funcionalidade demonstrou ser eficaz na extração de informações detalhadas sobre empresas, utilizando requisições HTTP e processamento de HTML com BeautifulSoup. O uso de

tratamento de erros garantiu robustez e consistência nos resultados. A opção de exportar os dados para CSV adiciona flexibilidade à aplicação.

Um exemplo de uso é o seguinte: “python jobs.py get 12345”.

A função *statistics* permite serem mostradas estatísticas detalhadas sobre vagas de trabalho agrupadas por zona geográfica, nome do trabalho e/ou tipo de trabalho.

O procedimento desta funcionalidade passou por:

* definir os parâmetros, sendo eles mostrar(str, optional), que define o tipo de agrupamento a ser feito, ou seja. Pode ser agrupado por título, por tipo, ou por ambos. ainda há o parâmetro região(str, optional), que caso seja fornecido, refere-se a uma zona específica para filtrar as vagas;
* configurar os parâmetros, tendo sido posto um limite de 1500 resultados, mais uma vez;
* requisição à API, onde são requisitados os dados com os filtros/condições que o utilizador impôs (por exemplo, filtrar pela localização e apresentar as estatísticas quer por título quer por tipo);
* mostrar os resultados obtidos, com base nos agrupamentos considerados, ou seja. CSV com agrupamentos por título e/ou tipo e a localização correspondente caso não tenha sido especificada nenhuma. Se não forem conseguidos nenhuns resultados, aparecerá uma mensagem a afirmar tal sucedido.

Para finalizar, aqui está um exemplo de execução: “python jobs.py statistics --mostrar tipo --regiao Porto”.

A função *list\_skills* tem como objetivos: extrair habilidades (“skills”) de descrições de vagas publicadas num site de empregos; consolidar essas habilidades num formato estruturado com a frequência de ocorrência de cada habilidade; opcionalmente, pode-se salvar os dados num arquivo CSV.

Como Parâmetros de Entrada temos os seguintes: trabalho (str): Nome do trabalho que define o filtro de pesquisa; guardar (bool): Indica se os resultados devem ser salvos em um arquivo CSV. Como saídas temos o seguinte: exibe uma lista das habilidades mais frequentes e suas respetivas frequências; salva os dados em um arquivo CSV, se especificado.

Assim, os procedimentos para a execução desta função são os seguintes:

* Recebimento dos Parâmetros
  + trabalho (str): Nome do trabalho a ser pesquisado (ex. "software engineer”);
  + guardar (bool): Indica se os dados extraídos devem ser salvos em um arquivo CSV (Default: False);
* Determinação do Total de Trabalhos e Páginas
  + Solicita ao usuário o número total de trabalhos a pesquisar;
  + Calcula o número de páginas necessárias para cobrir o total de trabalhos (“paginas\_necessarias”);
  + Itera sobre cada página para coletar URLs das vagas;
* Requisições e Coleta de URLs
  + Utiliza a função *request\_website* para realizar requisições às páginas do site;
  + Extrai os URLs das vagas utilizando a biblioteca BeautifulSoup para encontrar elementos HTML com a tag <meta>e atributo itemprop=“url";
  + Valida a existência de URLs antes de prosseguir. Caso nenhum URL seja encontrado, a função encerra a execução;
* Extração de Habilidades
  + Para cada URL coletado, utiliza o Playwright para carregar a página correspondente;
  + Localiza e clica no botão “More” (se presente) para expandir a lista de habilidades;
  + Coleta todas as habilidades presentes na página, identificadas por elementos HTML com a classe .body-medium.chip;
  + Adiciona as habilidades a um dicionário skills\_count, que regista a frequência de cada habilidade;
* Consolidação dos Dados
  + Converte o dicionário skills\_count em uma lista de dicionários, onde cada item tem o formato:
    - skill: Nome da habilidade;
    - count: Frequência de ocorrência;
  + Solicita ao usuário o número de habilidades mais frequentes (“top”) que deseja visualizar;
  + Ordena as habilidades pela frequência de ocorrência e exibe as “top” habilidades;
* Salvar em CSV (Opcional)
  + Caso o parâmetro guardar seja True, salva os dados em um arquivo CSV.
  + Utiliza a biblioteca csv para criar e gravar o arquivo.

A função *list\_skills* apresenta robustez na extração de informações de habilidades em descrições de trabalhos. Para além disso, a validação de entradas e o tratamento de erros garantem uma experiência confiável e intuitiva ao usuário.

Um exemplo de uso é o seguinte: “python jobs.py list-skills “Data Scientist”.

Quase a terminar, há ainda a função “*get2*” que se foca em enriquecer as informações de uma vaga de trabalho com dados adicionais obtidos do site “companiesmarketcap”. Este website foi escolhido pois permite Web Scraping e fornece informações interessantes e diferentes, o que complementa os dados do website utilizando anteriormente.

As etapas de resolução foram as seguintes:

* definir parâmetros sendo job\_id(int) e csv onde o default é mais uma vez a não exportação dos resultados;
* requisitar a API, de modo a recolher os dados da vaga identificada pelo job\_id;
* validação dos dados da empresa, onde o nome da mesma é extraído para ser usado como parte do url na requisição ao site;
* web scraping ao site, onde é utilizada a função auxiliar “request\_website” para obter o conteúdo html do site para que, de seguida, sejam extraídas as seguintes informações: rank, que representa a posição da empresa no ranking de melhores empresas; market\_cap, que indica o valor de mercado da empresa; e descrição que fornece uma descrição geral da empresa. Caso qualquer uma destas informações não seja encontrada, é registada uma mensagem indicando a ausência dos dados;
* exportação dos dados para csv, caso o parâmetro seja TRUE e dá se uso à função “cria\_csv” para isso ser possível;
* exibição dos dados, onde as informações são mostradas no terminal.

Um exemplo de execução é então, “python jobs.py get2 12345".

Por fim, temos então uma função extra *vagas\_recomendadas*, que decidimos adicionar, uma vez que torna o projeto mais completo e intuitivo.

A função *vagas\_recomendadas* tem como objetivo recomendar vagas de emprego relacionadas com base no título da vaga fornecida por ID.

Como parâmetros de entrada a função apresenta job\_id (int): ID da vaga a ser utilizada como referência para a busca; limit (int,default=5): Número máximo de vagas relacionadas a exibir. Como saídas temos as seguintes:exibe os detalhes da vaga principal e das vagas relacionadas; mensagens claras em caso de erros ou ausência de dados.

Assim, os procedimentos para a execução desta função são os seguintes:

* Recebe como entrada: job\_id: ID da vaga principal para obter detalhes; limit: Número máximo de vagas relacionadas a exibir (padrão: 5);
* Faz uma requisição à API para obter os detalhes da vaga;
* Exibe informações sobre a vaga, incluindo: título; empresa; localização e descrição;
* Realiza uma procura de vagas relacionadas: utiliza palavras do título da vaga como base para a consulta; limita o número de resultados conforme especificado no parâmetro limit; remove vagas duplicadas pelo ID;
* Exibe as vagas relacionadas com detalhes pretendidos: título, empresa. localização e descrição.

Um exemplo de execução é então, “python jobs.py vagas\_recomendadas (12345, limit=3)”.

Este projeto, desenvolvido no âmbito da cadeira de Ambientes e Linguagens de Programação para Ciência de Dados (ALPCD), proporcionou uma abordagem prática à integração de APIs e técnicas de Web Scraping, demonstrando como ferramentas computacionais podem ser utilizadas para explorar e analisar o mercado de trabalho.

A combinação de bibliotecas como Typer, Requests, BeautifulSoup, sync\_playwright, csv e Datetime permitiu implementar soluções robustas e flexíveis para aquisição e processamento de dados. A inclusão de funções auxiliares e principais foi essencial para estruturar o projeto, destacando-se pela modularidade e reutilização de código, o que facilitou a resolução de problemas complexos e promoveu a eficiência.

Além disso, a variedade de funcionalidades implementadas – desde a extração de informações salariais e estatísticas sobre vagas até a recomendação de oportunidades e análise de competências – enriquece a aplicabilidade prática do projeto. Estas ferramentas não apenas auxiliam candidatos a emprego, mas também fornecem insights valiosos para análise de dados em grande escala, demonstrando o impacto da tecnologia no setor de recrutamento.

Por fim, o desenvolvimento deste projeto permitiu consolidar conhecimentos em linguagens de programação, manipulação de dados, e métodos de scraping, além de promover a resolução de desafios reais de forma criativa e eficiente. Esta experiência reflete a importância de alinhar teoria e prática, preparando-nos para resolver problemas concretos no contexto da ciência de dados.