Universidade do Minho
Escola de Ciências

JOB API EXPLORER E WEB SCRAPING

Realizado por:

- Filipe Araújo(A112467)
- Marta Rocha (A112387)
- Olga Marynych (A79235)

Docentes:

- Prof. Tiago Baptista
- Prof. Nuno Macedo
- Prof. Pedro Rangel Henriques

Índice

Introdução

Este projeto analisa dados da API do itjobs.pt, focando-se na recolha, armazenamento e processamento de anúncios de emprego na área das TI. Para simplificar a interação, foi desenvolvida uma CLI (Command Line Interface) em Python, recorrendo aos módulos typer, csv, requests, json, re e datetime.

A ferramenta evita chamadas excessivas à API, extrai e filtra informação relevante e contabiliza skills técnicas e comportamentais presentes nos anúncios. O objetivo é automatizar tarefas manuais, transformando dados brutos em informações úteis e claras.

Para além da exploração da API do ITJobs, este trabalho introduz técnicas de web scraping, permitindo enriquecer os dados obtidos com informação externa do website Teamlyzer.

Organização dos dados na API do ITJobs

A API do itjobs.pt apresenta uma estrutura organizada para fornecer informação sobre ofertas de emprego na área tecnológica. Cada pedido devolve dados em JSON, incluindo metadados como o total de resultados, a página atual e o tamanho da paginação. O elemento central é a lista *results*, que reúne todas as ofertas individuais.

Cada entrada inclui detalhes como ID, título, empresa, localização, data de publicação, tags e descrição completa da vaga, além de links úteis para candidatura ou informação adicional.

A API disponibiliza ainda vários endpoints para diferentes tipos de pesquisa — por empresa, localização ou tecnologias. Em todos os casos, o formato consistente facilita a análise automática e o processamento dos dados.

Resolução do projeto TP1

Comando “dump” –
Criação/ atualização do ficheiro “empregos.json”

A partir do comando dump, é criado ou atualizado o ficheiro “*empregos.json”* com os dados mais recentes obtidos da API.

Este processo evita chamadas repetidas e ajuda a contornar o limite de requisições imposto pela API.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python jobscli.py dump
Arquivo 'empregos.json' criado/atualizado com sucesso!
```

Comando “dump” -código

```python
#cria/atualiza o ficheiro "empregos.json, para evitar requisições consecutivas à API e respetivo bloqueio"----------------------------------------------------------
@app.command(name="dump")
def dump():
    all_results = []
    page = 1
    try:
        while True:
            resposta = requests.get(url, headers=user_agent, params={
                "api_key": api_key,
                "page": page,
                "limit": 100 #para cada página existe um limite de 100 anúncios
            })

            if resposta.status_code != 200:
                typer.echo(f"Erro na requisição (página {page}): {resposta.status_code} - {resposta.reason}")
                raise typer.Exit(code=1)

            dados = resposta.json()
            results = dados.get("results", [])

            if not results:
                break

            all_results.extend(results)
            page += 1

        #Guarda todos os resultados
        final_data = {"total": len(all_results), "results": all_results}
        with open("empregos.json", "w", encoding="utf-8") as f:
            json.dump(final_data, f, ensure_ascii=False, indent=4)

        typer.echo(f"Arquivo 'empregos.json' criado/atualizado com sucesso!") #como tem vários anúncios, o processo de criação/atualização é um bocado mais demorado

    except Exception as e:
        typer.echo(f"Erro inesperado: {str(e)}")
        raise typer.Exit(code=1)
```

Comando “top” -
Lista dos N trabalhos mais recentes publicados pela itjobs.pt

Depois de criado ou atualizado o ficheiro *“empregos.json”*,o comando top permite selecionar e visualizar os *n* anúncios mais recentes.

O resultado apresenta, para cada oferta, o ID, o título da vaga, a empresa e a data de publicação, facilitando uma consulta rápida das últimas oportunidades disponibilizadas pela API.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python jobscli.py top 5
[
    {
        "ID da oferta": 507035,
        "Oferta de emprego": "Formador na área de Inteligência Artificial",
        "Empresa": "Master.D",
        "Data da publicação": "2025-11-14 21:57:31"
    },
    {
        "ID da oferta": 507049,
        "Oferta de emprego": "OutSystems Developer",
        "Empresa": "Adentis",
        "Data da publicação": "2025-11-14 21:57:14"
    },
    {
        "ID da oferta": 507044,
        "Oferta de emprego": "Analista Funcional (Banca)",
        "Empresa": "team.it",
        "Data da publicação": "2025-11-14 21:57:07"
    },
    {
        "ID da oferta": 507037,
        "Oferta de emprego": "Formadores para as áreas da Transição Digital",
        "Empresa": "CITEVE - Centro Tecnológico das Indústrias Têxtil e Vestuário de Portugal",
        "Data da publicação": "2025-11-14 21:56:36"
    },
    {
        "ID da oferta": 507043,
        "Oferta de emprego": "Senior IT Auditor",
        "Empresa": "QiBit Portugal",
        "Data da publicação": "2025-11-14 21:56:25"
    }
]
```

Comando “top” - código

```python
@app.command(name="top")
def top(
    n: int = typer.Argument(..., help="Número de empregos a listar"),
    csv_file: str = typer.Option(None, "--csv", "-c", help="Nome do ficheiro CSV para exportar")
):
    """Lista os N trabalhos mais recentes e opcionalmente exporta para CSV"""
    try:
        with open("empregos.json", "r", encoding="utf-8") as f:
            conteudo = json.load(f)
    except FileNotFoundError:
        typer.echo("Ficheiro 'empregos.json' não encontrado. Faz primeiro o dump da API.")
        raise typer.Exit(code=1)

    trabalhos = conteudo.get("results", [])
    recentes = trabalhos[:n]

    lista_csv = []
    lista_terminal=[]
    for t in recentes:
        empresa = t.get("company", {}).get("name", "Empresa desconhecida")
        titulo = t.get("title", "Sem título")
        data_publicacao = t.get("publishedAt", "")
        descricao = t.get("body", "")
        salario = t.get("salary", "")
        localizacao = t.get("location", "")
        id = t.get("id", "")

        item = {
            "ID da oferta": id,
            "Oferta de emprego": titulo,
            "Empresa": empresa,
            "Data da publicação": data_publicacao,
            "Descrição": descricao,
            "Salário": salario,
            "Localização": localizacao
        }
        lista_csv.append(item)
        lista_terminal.append({"ID da oferta": id,"Oferta de emprego": titulo,"Empresa": empresa,"Data da publicação": data_publicacao})

    # JSON para o terminal
    print(json.dumps(lista_terminal, indent=4, ensure_ascii=False))
```

Recolher a informação dos n anúncios mais “recentes” e exibir no formato JSON no terminal.

Comando “top” – exportar para CSV

```python
# Exportar CSV se for indicado
if csv_file:
    with open(csv_file, "w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=["Oferta de emprego", "Empresa", "Descrição", "Data da publicação", "Salário", "Localização"])
        writer.writeheader()
        for item in lista_csv:
            writer.writerow({k: item[k] for k in writer.fieldnames})
    typer.echo(f"CSV exportado com sucesso para '{csv_file}'")
```

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python projetoFinal.py top 2 --csv top2.csv
[
    {
        "ID da oferta": 507057,
        "Oferta de emprego": "Quality Manager (AI/Automation)",
        "Empresa": "Otonomee",
        "Data da publicação": "2025-11-16 16:33:39"
    },
    {
        "ID da oferta": 507058,
        "Oferta de emprego": "Product Owner/Project Manager",
        "Empresa": "Aubay",
        "Data da publicação": "2025-11-16 16:33:06"
    }
]
CSV exportado com sucesso para 'top2.csv'
```

Comando "search" - Listar todos os trabalhos do tipo part-time, publicados por uma determinada empresa, numa determinada localidade

O comando **search** permite procurar ofertas de emprego **part-time** com base numa localidade e numa empresa específicas. O utilizador indica a localidade, o nome da empresa e o número máximo de anúncios a apresentar.

Se não forem encontrados resultados, é apresentada uma mensagem a informar o utilizador. Caso existam anúncios, estes são mostrados no terminal em formato JSON, com a descrição limpa das principais *tags* HTML para facilitar a leitura.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python jobscli.py search Lisboa "Master.D" 10
[
    {
        "titulo": "Formador na área de Inteligência Artificial",
        "empresa": "Master.D",
        "descrição": "É apaixonado/a por tecnologia, inovação e ensino?  O futuro está a ser escrito com Inteligência Artificial, e queremos que faça parte
dessa jornada!  O nosso grupo internacional na área da formação procura um/a Formador/a de IA para liderar um curso abrangente de IA, onde são abordados des
de os fundamentos de Python e Machine Learning até tópicos avançados como Deep Learning e a aplicação prática da IA em diferentes setores.     O que procura
mos:   Formação em Engenharia Informática, Ciências da Computação, Matemática, Estatística ou áreas relacionadas;   Certificações em Machine Learning / Deep
 Learning (valorizadas);   Experiência em projetos de IA, desenvolvimento de software ou análise de dados;   Domínio de Python e experiência em Machine Lear
ning e Deep Learning (CNNs, RNNs, etc.);   CCP (Certificado de Competências Pedagógicas) - obrigatório;   Experiência comprovada como formador/a e gosto pel
a partilha de conhecimento.     O que oferecemos:   Integração num ambiente dinâmico, inovador e internacional;   Contrato de prestação de serviços em regim
e part-time;   Teletrabalho;   Um ambiente colaborativo, com espaço para crescer e inovar.     Se gosta de ensinar e inspirar, e quer contribuir para o futu
ro da formação em Inteligência Artificial, envie o seu CV para recrutamento.formacao.pt@gmail.com com a referência IA/TUT.  ",
        "data de publicação": "2025-11-14 21:57:31",
        "salário": null,
        "localização": "Lisboa"
    }
]
```

Comando “search” – código

```python
def remove_html_tags(text): #para uma melhor visualização da descrição, elimina-se as tags de html
    if not text:
        return ""
    text = re.sub(r'</p>', '  ', text) #Duplo espaço quando </p>
    clean = re.compile('<.*?>') #elimina <> e os caracteres de dentro que podem ser nenhum ou mais
    return re.sub(clean, '', text)

@app.command(name="search")
def search(local: str, empresa: str, n: int, csv_out: bool = typer.Option(False, "--csv", help="Exportar para CSV")):
    try:
        with open("empregos.json", "r", encoding="utf-8") as f:
            conteudo = json.load(f)
    except FileNotFoundError:
        typer.echo("Ficheiro 'empregos.json' não encontrado. Faz primeiro o dump da API.")
        raise typer.Exit(code=1)

    trabalhos = conteudo.get("results", [])
    filtered = []

    for job in trabalhos:
        locations = job.get("locations", [])
        found_location = False
        for loc in locations:
            if local.lower() in loc.get("name", "").lower(): #algumas localizações estão no campo do nome ("name")
                found_location = True
                break
        if not found_location:
            continue
        company_name = job.get("company", {}).get("name", "")
        if empresa.lower() not in company_name.lower():
            continue

        types = job.get("types", [])
        part_time = any("part-time" in t.get("name", "").lower() for t in types) #part-time pode estar no tipo
        title = job.get("title", "").lower()
        body = job.get("body", "").lower()
        part_time_pattern = r"part[-\s]?time"#\s -espaço em branco ; o espaço em branco e o "-" pode ser opcional "?"
        if not part_time and (re.search(part_time_pattern, title) or re.search(part_time_pattern, body)):
            part_time = True
```

Remover as tags HTML e caracteres para uma melhor formatação na descrição.

O tipo de trabalho do anúncio deve ser part-time.

Comando “search” – código

```
        if part_time:
            filtered.append(job)
    filtered = filtered[:n] #n resultados

    if not filtered:
        typer.echo("Nenhum resultado encontrado.")
        raise typer.Exit()

    output_data = []
    for job in filtered:
        locations_list = [loc.get("name", "") for loc in job.get("locations", [])] #passar para sting em vez de lista para o csv
        locations_str = ", ".join(locations_list) if locations_list else ""

        output_data.append({
            "titulo": job.get("title", ""),
            "empresa": job.get("company", {}).get("name", ""),
            "descrição": remove_html_tags(job.get("body", "")),
            "data de publicação": job.get("publishedAt", ""),
            "salário": job.get("wage", ""),
            "localização":locations_str
        })
```

Caso a empresa escolhida não possuir nenhum anúncio de part-time, então é apresentada a mensagem “Nenhum resultado encontrado”.
Os resultados são apresentados no formato json.

Comando “search” – exportar para CSV

```python
if csv_out:
    with open("search_results.csv", "w", newline="", encoding="utf-8") as f:
        if output_data:
            writer = csv.DictWriter(f, fieldnames=output_data[0].keys())
            writer.writeheader()
            writer.writerows(output_data)
        typer.echo("Resultados exportados para 'search_results.csv'")
else:
    print(json.dumps(output_data, indent=4, ensure_ascii=False))
```

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python jobscli.py search Lisboa "Master.D" 10 --csv
Resultados exportados para 'search_results.csv'
```

Comando “regime” –
Extrair o regime de trabalho de um determinado job id

Recebe o ID de um anúncio e devolve o tipo de regime associado: remoto, híbrido ou presencial. Para efeitos de teste, podem ser utilizados os ID devolvidos no comando anterior – comando “top”.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python jobscli.py regime 507035
Remoto
```

Comando “regime” - código

```python
@app.command(name="regime")
def regime_trabalho(id: int = typer.Argument(..., help="ID do emprego")):
    url1 = "https://api.itjobs.pt/job/get.json"
    resposta = requests.get(url1, headers=user_agent, params={"api_key": api_key, "id": id})

    if resposta.status_code == 200:
        vaga_emprego = resposta.json()
        descricao_vaga = vaga_emprego.get("body", "")

        hibrido = re.search(r"([Hh][íi]brido|[Hh]ybrid)", descricao_vaga)
        presencial = re.search(r"([Pp]resencial|[Oo]n-?site)", descricao_vaga)

        if vaga_emprego.get("allowRemote"):
            typer.echo("Remoto")
        elif hibrido:
            typer.echo("Híbrido")
        elif presencial:
            typer.echo("Presencial")
        else:
            typer.echo("Não identificado")
    else:
        print(f"Erro na requisição: {resposta.status_code} - {resposta.reason}")
```

Procurar padrões que indiquem hibrido ou presencial.

Comando “skills” - Contar ocorrências de skills escolhidas nas descrições dos anúncios entre duas datas

Recebe duas datas no formato AAAA-MM-DD e conta quantas vezes cada skill aparece nos anúncios publicados dentro desse intervalo (**comando “procurar”** – **comando intermédio**). Para tornar os testes mais eficientes, recomenda-se utilizar datas próximas da atual, dado que novos anúncios são publicados diariamente.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python jobscli.py skills 2025-11-14 2025-11-15
{
    "Python": 6,
    "Análise": 2,
    "Inovação": 2,
    "NAS": 2,
    "JavaScript": 1,
    "SQL": 3,
    "SQL Server": 1,
    "C": 2,
    "CSS": 1,
    "Flexibilidade": 1,
    "Gestão": 1,
    "Comunicação": 3,
    "Problem-solving": 2,
    "Communication skills": 3,
    "Frameworks": 2,
    "Desenho": 1,
    "Troubleshooting": 1,
    "Workflows": 1,
    "Data Platform": 1,
    "Databricks": 1,
    "Monitoring": 2,
    "Apache": 1,
    "Java": 1,
    "Git": 1,
    "NoSQL": 2,
    "Data engineering": 1,
    "Data modeling": 1,
    "APIs REST": 1,
    "Embedded systems": 1,
    "Software architecture": 1,
    "Software development": 1,
    "Leadership": 1,
    "Autonomy": 1,
    "Planning": 1,
    "Training": 2,
    "Organization": 1,
    "Containers": 1,
    "CI/CD": 1,
    "SaaS": 1,
    "TypeScript": 1,
    "GraphQL": 1,
    "DevOps": 1,
    "Security best practices": 1
}
```

Comando “skills” - código

```python
@app.command(name="skills")
def contar_skills(data_inicial: str, data_final: str):
    """Conta skills entre duas datas"""
    try:
        dt_ini = datetime.strptime(data_inicial, "%Y-%m-%d")
        dt_fim = datetime.strptime(data_final, "%Y-%m-%d")
    except ValueError:
        typer.echo("Datas inválidas. Usa o formato YYYY-MM-DD.")
        raise typer.Exit(code=1)

    try:
        with open("empregos.json", "r", encoding="utf-8") as f:
            conteudo = json.load(f)
    except FileNotFoundError:
        typer.echo("Ficheiro 'empregos.json' não encontrado. Primeiro faz 'dump'.")
        raise typer.Exit(code=1)

    trabalhos = conteudo.get("results", [])
    contador = {}

    #lista deduzida a partir do resultado do comando anterior. Skills técnicas e comportamentais
    lista_skills = ["Linux","Red Hat","CentOS","Ubuntu","Virtualização","Proxmox","Storage Networking","NAS","SAN","iSCSI","DNS","Mail","DHCP","LDAP",
                    "Active Directory","SSL", "Scripting","Bash","Python","Apache","Nginx","Troubleshooting","Containers","Docker","Docker Swarm","Kubernetes",
                    "Veeam Backup","Zabbix","Ethernet","Routing","NAT","IPSec VPN", "VLAN","Java","JavaScript","CI/CD","RESTful APIs","Microservices","Git",
                    "SQL","MySQL","PostgreSQL","NoSQL","Redis","Django","Node.js", "iOS","Android","React Native","Mac OS","Ansible","Capistrano","Photoshop",
                    "Figma","VBA","MS Access","SQL Server","PHP","PowerBI","Firewalls Checkpoint", "Cisco","IPv4","IPv6","IPSec","MPLS","VXLAN","TCP/IP",
                    "Switching","Networking","Industry 4.0","MES software","Product Marketing","Frameworks","Workflows","Go-to-Market (GTM)","Connect IoT",
                    "Data Platform", "Messaging frameworks","Storytelling","Product positioning","Messaging architectures","GTM playbooks",
                    "Competitive battlecards","Pitch decks","Objection handling guides", "ROI tools","Training","Workshops","Pitch libraries","Demo flows",
                    "Proof-of-value materials","Dashboards","Content usage tracking","Message adoption","Enablement impact","Campaigns", "SaaS",
                    "Cloud infrastructure (AWS, Azure, GCP)","Infrastructure as Code (Terraform, Pulumi)","Backend development","TypeScript","GraphQL",
                    "REST APIs","Data engineering", "Databricks","Snowflake","ETL pipelines","Data modeling","Frontend engineering","DevOps","Monitoring",
                    "Security best practices","ECS","EKS","GKE","AKS","Serverless platforms", "AI/ML infrastructure","Data visualization","Tableau",
                    "Embedded systems","C","C++","Real time systems","Software architecture","Design patterns","Multithreading","Multiprocess applications",
                    "Bash scripting","Communication technologies","Layer 2 and Layer 3 network applications","Cybersecurity concepts","Electronics",
                    "Laboratory instrumentation", "Version control systems","SVN","Microsoft Dynamics 365 (CRM)","Power Platform","CRM","Integration of systems",
                    "Automation of business processes","Sales","Contact Center", "Service Workspace","Management of access and data security","Agile methodologies",
                    "Technical planning tools","Spring Boot","Software development","Java backend development", "Angular","HTML","CSS","Bootstrap","APIs REST","Jira","Xray",
                    "Azure DevOps","Liderança","Gestão","Mentoring","Comunicação","Planeamento","Desenho","Análise","Inovação", "Colaboração","Trabalho em equipa",
                    "Flexibilidade","Proatividade","Dedicação","Integridade","Transparência","Confiança","Honestidade","Responsabilidade","Criatividade", "Problem-solving",
```

Converte as strings de datas para objeto datetime.

Comando “skills” – código

```
            "Analytical mind","Communication skills","Collaboration","Leadership","Analytical skills","Problem solving","Creativity","Strategic thinking","Adaptability",
            "Fast learning","Ambition","Commitment","Focus on innovation","Dynamism","Autonomy","Teamwork","Organization","Attention to detail","Passion for learning",
            "Bias for action", "Intellectual curiosity","Navigating ambiguity", "Planning"]

    # nota: \b representa "boundary", ou seja, o programa procura a palavra por inteiro (sem palavras parecidas); escape(s) - tratamento literal de cada caractere de s.
    lista_regex = {s: re.compile(r"\b" + re.escape(s) + r"\b", re.IGNORECASE) for s in lista_skills}

    for t in trabalhos:
        data_pub_str = t.get("publishedAt")
        if not data_pub_str:
            continue
        try:
            data_pub = datetime.strptime(data_pub_str.split(" ")[0], "%Y-%m-%d")
        except ValueError:
            continue

        if not (dt_ini <= data_pub <= dt_fim):
            continue

        corpo = t.get("body", "")
        for skill, rgx in lista_regex.items():
            if rgx.search(corpo):
                contador[skill] = contador.get(skill, 0) + 1

    # --- NOVO: ordenar resultados por número de ocorrências (decrescente)
    contador_ordenado = (dict(sorted(contador.items(), key=lambda x: x[1], reverse=True)))

    # --- NOVO: devolver no formato pretendido (lista com um dicionário)
    print(json.dumps([contador_ordenado], indent=4, ensure_ascii=False))
```

Incrementa o contador quando uma skill é encontrada (a partir de expressões regulares) e os resultados são apresentados de forma decrescente no formato json.

Comando “procurar” /Comando intermédio

Utilizado por nós como um **comando intermédio** para obter descrições de anúncios que incluíssem as seguintes expressões: "Experiência", "Conhecimentos", "Forte", "valoriza", "Domínio", "skills" ,"experience", "exprertise","looking for“….

Procura descrições de anúncios com possível indicação de skills (nem todos os anúncios têm)- serve como um filtro intermédio a partir do resultado, criou-se uma lista de skills usada no comando seguinte (de contagem).

Motivo: redução do campo de análise de expressões regulares associadas a skills, uma vez que nem todos os anúncios incluem as skills pretendidas.

Objetivo: criação de uma lista concreta de skills (tecnológicas e comportamentais). Dado que o output desta pesquisa permaneceu extenso, foi utilizado o apoio da Inteligência Artificial para filtrar as palavras de interesse de forma eficiente.

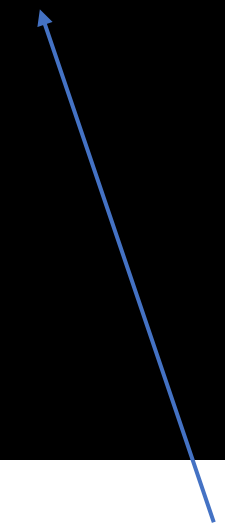
Comando “procurar” – código

```python
@app.command(name="procurar")
def procurar_palavras():
    try:
        with open("empregos.json", "r", encoding="utf-8") as f:
            conteudo = json.load(f)
    except FileNotFoundError:
        typer.echo("Ficheiro 'empregos.json' não encontrado. Faz primeiro o dump da API.")
        raise typer.Exit(code=1)

    trabalhos = conteudo.get("results", [])
    padrao = re.compile(r"(experiência|conhecimentos|forte|valoriza|domínio|skills|experience|expertise|looking\s+for)", re.IGNORECASE)

    encontrados = []

    for t in trabalhos:
        corpo = t.get("body", "")
        if padrao.search(corpo):
            encontrados.append(corpo)
    if encontrados:
        print(json.dumps(encontrados, indent=4, ensure_ascii=False))
    else:
        typer.echo("Nenhuma vaga com as palavras procuradas.")
```



Define o padrão de busca para identificar seções relevantes.

Resolução do projeto TP2

Comando "Get"– Informações sobre um jobID específico utilizando a API do itjobs.pt e adicionar informações acerca da empresa que publicita o trabalho

O comando **get** permite obter informação detalhada sobre uma oferta de emprego através do seu *jobID* (Ex: A empresa Dellent tem como id -> 506067).

Os dados base são recolhidos da API do ITJobs e posteriormente enriquecidos comas novas funcionalidades da empresa proveniente do Teamlyzer.

O output inclui o *rating* da empresa, descrição, benefícios e salário médio, apresentados em formato JSON.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python projetoFinal.py get 506067

 A extrair informações da empresa...
{
    "id": 506067,
    "teamlyzer_rating": 1.9,
    "teamlyzer_description": "A Dellent Consulting é uma empresa portuguesa especializada em serviços de consultoria e engenharia de sistemas de informação e telecomunicações.",
    "teamlyzer_benefits": "Seguro de saúde; Plafond certificações e conferências; Programa de descontos; Cultura de multi-função",
    "teamlyzer_salary": "1.458€ - 1.658€"
}
```

Comando “Get”– código

```python
#------Função auxiliar para ler HTML de uma URL-----------------------------------------------------
def ler_html(url: str):
    headers = {"User-Agent": "Mozilla/5.0 (compatible; TeamlyzerScraper/1.0)"}
    try:
        resp = requests.get(url, headers=headers, timeout=10)
        if resp.status_code == 200:
            return BeautifulSoup(resp.content, 'html.parser')
        else:
            return None
    except Exception as e:
        typer.echo(f"Erro de conexão: {e}")
        return None
```


Acesso da página web e leitura do seu conteúdo HTML. Caso algum erro de conexão, é dito o seu tipo de erro.

Comando “Get”– código

```python
def encontrar_slug_empresa(nome_empresa):
    # 1.normalizar: remover acentos e converter para ASCII
    nome_normalizado = unicodedata.normalize('NFKD', nome_empresa)
    nome_ascii = nome_normalizado.encode('ascii', 'ignore').decode('ascii')

    # 2.substituir caracteres não alfanuméricos (pontos, parênteses, etc.) por hífens
    nome_sem_especiais = re.sub(r'[^\w\s-]', '-', nome_ascii)

    # 3.substituir espaços por hífens e converter para minúsculas
    slug_final = re.sub(r'[-\s]+', '-', nome_sem_especiais).lower().strip('-')

    # 4.construir e testar a URL
    url = f"https://pt.teamlyzer.com/companies/{slug_final}"
    soup = ler_html(url)

    if soup: #pelo título, verificar se é 404
        title = soup.find('title')
        if title and "404" not in title.text.lower():
            return slug_final

    # 5. Fallback: estratégia da primeira palavra
    palavras = nome_empresa.split()
    if len(palavras) > 1:
        slug_fallback = palavras[0].lower()
        if slug_fallback != slug_final:
            url_fallback = f"https://pt.teamlyzer.com/companies/{slug_fallback}"
            soup_fallback = ler_html(url_fallback)
            if soup_fallback:
                title = soup_fallback.find('title')
                if title and "404" not in title.text.lower():
                    return slug_fallback

    #6.Para os nomes de empresas que na url do teamlyzer, tem o nome da empresa- (mais palavras)
    base_palavra = nome_empresa.split()[0].lower() if nome_empresa.split() else ""
    if base_palavra:
        base_limpa = re.sub(r'[^a-z]', '', base_palavra)
```

Como o slug é o nome da empresa, decorre-se transformações no nome da empresa para que seja correspondido corretamente ao da página web e assim organizar o conteúdo desejado.

Comando “Get”– código

```
# Buscar na listagem de empresas
    for sufixo in ['consulting', 'portugal', 'solutions', 'technology', 'group', 'outsourcing','company']:
        slug_teste = f"{base_limpa}-{sufixo}"
        url_teste = f"https://pt.teamlyzer.com/companies/{slug_teste}"
        soup_teste = ler_html(url_teste)

        if soup_teste:
            title = soup_teste.find('title')
            if title and "404" not in title.text.lower():
                # Verificação extra rápida
                titulo_texto = title.get_text().lower()
                if base_limpa in titulo_texto:
                    return slug_teste
return None
```

Para empresas que tem o seu nome identificado na página web com um dos sufixos.

Comando “Get”– código (Extrair informação da empresa)

```python
def nova_funcio(res):
    if not res:
        return {}

    nome_empresa = res.get("company", {}).get("name", "")
    if not nome_empresa:
        return {}

    # 1.Encontra slug correto
    slug_correto = encontrar_slug_empresa(nome_empresa)

    if slug_correto:
        empresa_teamlyzer = slug_correto
    else:
        #Fallback: usar nome original
        empresa_teamlyzer = nome_empresa.replace(" ", "-").lower()

    # 2.Obter a página usando ler_html
    url_principal = url_teamlyzer + empresa_teamlyzer
    soup = ler_html(url_principal)

    if not soup:
        typer.echo(f"Erro ao acessar Teamlyzer para {nome_empresa}", er
        return {
            "teamlyzer_rating": 0.0,
            "teamlyzer_description": "Página não acessível",
            "teamlyzer_benefits": "",
            "teamlyzer_salary": ""
        }
```

```python
    # 2.RATING
    rating_container = soup.find("span", class_=re.compile(r'[a-z]+_rating'))

    if rating_container:
        full_text = rating_container.get_text(strip=True) #texto completo- ex: "3.4/5"

        if '/' in full_text:
            rating_text = full_text.split('/')[0].strip() #valor antes da barra
        else:
            rating_text = full_text

        try:
            resultados["teamlyzer_rating"] = float(rating_text)
        except ValueError:
                match = re.search(r"(\d+\.\d+)", full_text) #número com decimal
                if match:
                    resultados["teamlyzer_rating"] = float(match.group(1))
                else:
                    resultados["teamlyzer_rating"] = 0.0
    else:
        resultados["teamlyzer_rating"] = 0.0

    # 3.DESCRIÇÃO
    desc_elem = soup.find("div", class_="ellipsis center_mobile")

    if desc_elem:
        for unwanted in desc_elem.find_all(class_=["read-more", "read-less", "more-link", "less-link"]):
            unwanted.decompose()  # Remove completamente do HTML
        text = desc_elem.get_text(separator=" ", strip=True) #pega todo o código restante
        text = text.replace("…", "") #tira os caracteres especiais

        resultados["teamlyzer_description"] = text[:1000]
    else:
        resultados["teamlyzer_description"] = "Descrição não encontrada"
```

Comando “Get”– código

```python
# 4.BENEFÍCIOS
beneficios_lista = []
secao_beneficios = soup.find("div", {"class": "company_values_box"})

# Se não existir, tenta na página /benefits-and-values
if not secao_beneficios:
    url_beneficios = url_principal + "/benefits-and-values"
    soup_beneficios = ler_html(url_beneficios)
    if soup_beneficios:
        secoes_beneficios = soup_beneficios.find_all("div", {"class": "company_values_box"})

        # Filtrar apenas as que têm benefícios (normalmente as col-lg-8)
        for secao in secoes_beneficios:
            if "col-lg-8" in secao.get("class", []): #itens de benificio na seccao
                itens_beneficios = secao.find_all("li", {"class": "flex_group voffset2"})

                for item in itens_beneficios:  # SEM LIMITE - pega todos os itens
                    titulo = item.find("div", {"class": "flex_details"})
                    if titulo:
                        texto = titulo.get_text(strip=True)
                        if texto and len(texto) > 2:  # Reduzi para > 2 para apanhar benefícios curtos
                            beneficios_lista.append(texto)

if not beneficios_lista and secao_beneficios:# Tentar estrutura alternativa na página principal
    itens_beneficios = secao_beneficios.find_all("li", {"class": "flex_group voffset2"})
    for item in itens_beneficios:  # SEM LIMITE - pega todos os itens
        titulo = item.find("div", {"class": "flex_details"})
        if titulo:
            texto = titulo.get_text(strip=True)
            if texto and len(texto) > 2:
                beneficios_lista.append(texto)

#se ainda não encontrou benefícios, tentar seletores alternativos
if not beneficios_lista:#procura em qualquer lugar da página principal
    all_benefits = soup.find_all("div", {"class": "flex_details"})
    for elem in all_benefits:  # SEM LIMITE - pega todos os itens
        texto = elem.get_text(strip=True)
        if texto and len(texto) > 2 and "benefício" not in texto.lower():
            beneficios_lista.append(texto)
```

O conteúdo dos benefícios pode-se encontrar em diferentes elementos HTML “div”, “li” e ,até mesmo, numa url mais complexa com + “/benefits-and-values”. Não existe limite de dados em relação aos benefícios.

Comando “Get”– código

```python
beneficios_unicos = []#elimina duplicados
for benef in beneficios_lista:
    if benef not in beneficios_unicos:
        beneficios_unicos.append(benef)

resultados["teamlyzer_benefits"] = "; ".join(beneficios_unicos) if beneficios_unicos else "Nenhuma informação"

# 5.SALÁRIO MÉDIO
todos_panels = soup.find_all("div", class_="panel mini-box")

for panel in todos_panels:
    # Verifica se este panel tem ícone de euro
    euro_icon = panel.find("i", class_="fa-eur")
    if euro_icon:
        # Agora procura o size-h2
        size_h2 = panel.find("p", class_="size-h2")
        if size_h2:
            # Pega o texto completo
            texto_completo = size_h2.get_text(strip=True)


            if '€' in texto_completo and '%' not in texto_completo: #para não apanhar as percentagens, só o salário
                texto_html = str(size_h2) #tira setas
                texto_limpo = re.sub(r'<i.*?</i>', '', texto_html)
                texto_limpo = BeautifulSoup(texto_limpo, 'html.parser').get_text(strip=True)
                valores = re.findall(r'[\d\.,]+€', texto_limpo)

                if valores:
                    resultados["teamlyzer_salary"] = " - ".join(valores)
                    break

if "teamlyzer_salary" not in resultados:
    resultados["teamlyzer_salary"] = "Nenhuma informação"


return resultados
```

Conteúdo apresentado separado por pontos e vírgulas.

A informação sobre o salário pode ser encontrada em elementos HTML associados a classes que contêm o ícone do euro (€), sendo extraídos apenas os valores monetários e excluídas percentagens.

Comando “Get”– código

```python
@app.command(name="get")
def get_job(job_id: int = typer.Argument(..., help="ID do trabalho"),csv_export: bool = typer.Option(False, "--csv", help="Exportar para CSV")):
    # 1. Obter dados do job da API ITJobs
    url_tijobs = "https://api.itjobs.pt/job/get.json"
    params = {"api_key": "5ead20f487935ddbab9b3f084acdbf63", "id": job_id}

    try:
        resp_tijobs = requests.get(url_tijobs, headers=user_agent, params=params, timeout=10)
        resp_tijobs.raise_for_status()
        job_data = resp_tijobs.json()
    except Exception as e:
        typer.echo(f"Erro ao obter job {job_id}: {e}")
        raise typer.Exit(code=1)

    # 2. Extrair informações do Teamlyzer
    typer.echo(f"\n A extrair informações da empresa...", err=True)
    teamlyzer_info = nova_funcio(job_data)

    # 3.objeto final
    final = {
        "id": job_data.get("id", ""),
        "teamlyzer_rating": teamlyzer_info.get("teamlyzer_rating", 0.0),
        "teamlyzer_description": teamlyzer_info.get("teamlyzer_description", ""),
        "teamlyzer_benefits": teamlyzer_info.get("teamlyzer_benefits", ""),
        "teamlyzer_salary": teamlyzer_info.get("teamlyzer_salary", "")
    }

    # 4. Output para terminal
    print(json.dumps(final, indent=4, ensure_ascii=False))
```

Formatar e apresentar o conteúdo no formato JSON.

Comando “Get”– exportar para CSV

```python
    # 5. Exportar para CSV se solicitado
    if csv_export:
        novas_funcionalidades_csv(final, f"anuncio_{job_id}.csv")
        typer.echo(f"\n CSV exportado: 'anuncio_{job_id}.csv'", err=True)

# ----------------------------------------------------------------------------------------------------
def novas_funcionalidades_csv(job_data: dict, filename: str):
    try:
        with open(filename, "w", newline="", encoding="utf-8") as f:
            writer = csv.writer(f)
            writer.writerow(["Campo", "Valor"])

            # Apenas os 5 campos requeridos
            writer.writerow(["ID do Job", job_data.get("id", "")])
            writer.writerow(["Rating Teamlyzer", job_data.get("teamlyzer_rating", "")])
            writer.writerow(["Descrição Empresa", job_data.get("teamlyzer_description", "")])
            writer.writerow(["Benefícios", job_data.get("teamlyzer_benefits", "")])
            writer.writerow(["Salário Médio", job_data.get("teamlyzer_salary", "")])

    except Exception as e:
        typer.echo(f"Erro ao exportar CSV: {e}", err=True)
```

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python projetoFinal.py get 506067 --c
sv

 A extrair informações da empresa...
{
    "id": 506067,
    "teamlyzer_rating": 1.9,
    "teamlyzer_description": "A Dellent Consulting é uma empresa portuguesa especializada em serviços de consultoria e engenhar
ia de sistemas de informação e telecomunicações.",
    "teamlyzer_benefits": "Seguro de saúde; Plafond certificações e conferências; Programa de descontos; Cultura de multi-funçã
o",
    "teamlyzer_salary": "1.458€ - 1.658€"
}

 CSV exportado: 'anuncio_506067.csv'
```

Comando "Statistics"– A partir da lista de trabalhos do itjobs.pt, criar uma contagem de vagas por tipo/nome da posição e por região

O comando **statistics** analisa os anúncios armazenados no ficheiro "empregos.json". A funcionalidade realiza uma contagem do número de vagas por região e por tipo/nome da posição. Os resultados são organizados de forma estruturada e exportados para um ficheiro CSV.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python projetoFinal.py statistics zon
e
Ficheiro de exportação 'statistics_zone.csv' criado com sucesso!
```

Ficheiro "statistics_zone.csv" (excerto):

```
Zona,Tipo de Trabalho,Nº de vagas
Aveiro,.NET Architect,1
Aveiro,.NET Core Architect,1
Aveiro,.NET Core Senior Developer,1
Aveiro,.NET Core Tech Lead,1
Aveiro,.NET Senior Developer,1
Aveiro,Angular Architect,1
Aveiro,Angular Senior Developer,1
Aveiro,Automation Tester (Remote),1
```

(Existem mais skills do que as apresentadas neste print, pois este é apenas um excerto do output)

Comando “Statistics”– código

```python
# Subcomando statistics
statistics_app = typer.Typer()
app.add_typer(statistics_app, name="statistics")

@statistics_app.command(name="zone")
def statistics_zone(csv_file: str = typer.Option("statistics_zone.csv", "--csv", "-c", help="Nome do ficheiro CSV de saída")):
    """
    Contagem de vagas por região (zona) e tipo/nome da posição.
    Gera ficheiro CSV com colunas: Zona, Tipo de Trabalho, Nº de vagas
    """
    try:
        with open("empregos.json", "r", encoding="utf-8") as f:
            conteudo = json.load(f)
    except FileNotFoundError:
        typer.echo("Ficheiro 'empregos.json' não encontrado. Faz primeiro o dump da API.")
        raise typer.Exit(code=1)

    trabalhos = conteudo.get("results", [])
    contador = {}

    for t in trabalhos:
        locations = t.get("locations", [])
        zonas = [loc.get("name", "Desconhecida") for loc in locations] or ["Desconhecida"]
        tipo = t.get("title", "Não especificado")
        for zona in zonas:
            key = (zona, tipo)
            contador[key] = contador.get(key, 0) + 1

    lista_csv = [{"Zona": z, "Tipo de Trabalho": t, "Nº de vagas": n} for (z, t), n in contador.items()]
    lista_csv.sort(key=lambda x: (x["Zona"], x["Tipo de Trabalho"]))

    with open(csv_file, "w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=["Zona", "Tipo de Trabalho", "Nº de vagas"])
        writer.writeheader()
        writer.writerows(lista_csv)

    typer.echo(f"Ficheiro de exportação '{csv_file}' criado com sucesso!")
```

Carrega a informação do ficheiro “empregos.json”. Caso o ficheiro não for encontrado, o utilizador deve fazer o “dump” da API.

Para cada anúncio, recolhe a informação precisa: Localização, Tipo de trabalho e o nº de vagas (contagem).

Informação (lista de dicionários) exportada para um ficheiro csv.

Comando "list_skills"– A partir da lista de trabalhos no website Teamlyzer, recolher as principais (top 10) skills (referidas como 'stack tecnológica' no website) pedidas para um determinado trabalho

O comando **list_skills** permite identificar as **10 competências mais procuradas** para uma determinada profissão. A informação é obtida através de web scraping do website Teamlyzer, analisando anúncios associados à função indicada. As skills são contabilizadas, ordenadas por frequência e apresentadas em formato JSON.

No terminal:

```
(base) C:\Users\lenovo\OneDrive\2 ano-1 semestre\Ambientes e linguagem de programação\TP1>python projetoFinal.py list_skills fullstacks
[
  {
    "skill": "software-architecture",
    "count": 1
  },
  {
    "skill": "high-availability",
    "count": 1
  },
  {
    "skill": "ruby",
    "count": 1
  },
  {
    "skill": "testing",
    "count": 1
  },
  {
    "skill": "web-services",
    "count": 1
  },
  {
    "skill": "elasticsearch",
    "count": 1
  },
```

(Existem mais skills do que as apresentadas neste print, pois este é apenas um excerto do output)

Comando “list_skills”– código

```python
@app.command(name="list_skills")
def list_skills(
    prof: str = typer.Argument(help="Profissão a analisar"),
    export_csv: bool = typer.Option(False, "--csv", help="Exportar resultados para ficheiro CSV")):

    user_agent = {'User-agent': 'Mozilla/5.0'}
    url = "https://pt.teamlyzer.com/companies/jobs?profession_role=" + prof + "&order=most_relevant"

    dados_profissao = requests.get(url, headers=user_agent)
    soup_profissao = BeautifulSoup(dados_profissao.text, 'lxml')

    contagem = {}

    skills = soup_profissao.find_all("div", {"class": "voffset2"})
    for s in skills:
        skill_element = s.find("a")
        if skill_element and skill_element.text:
            skill = skill_element.text
            if skill in contagem:
                contagem[skill] += 1
            else:
                contagem[skill] = 1

    contagem_ordenada = sorted(contagem.items(), key=lambda item: item[1], reverse=True)

    skills_formatadas = []
    for skill, count in contagem_ordenada[:10]:
        skills_formatadas.append({
            "skill": skill,
            "count": count
        })

    json_10 = json.dumps(skills_formatadas, indent=2, ensure_ascii=False)
    typer.echo(json_10)
```

“prof” –nome da profissão a analisar como argumento para a requisição HTML do seu conteúdo. Processando-o com BeautifulSoup, para extrair apenas a informação necessária.

Contagem de cada ocorrência de cada skill encontrada com em elementos “div” e organizada de forma decrescente para encontrar as 10 competências mais requisitadas.
Informação apresentada em formato json no terminal.

Comando “list_skills”– exportar para CSV

Código:

```python
if export_csv:
    try:
        with open("top10_skills.csv", 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=['profession', 'skill', 'count'])
            writer.writeheader()

            for skill_data in skills_formatadas:
                writer.writerow({
                    'profession': prof,
                    'skill': skill_data['skill'],
                    'count': skill_data['count']
                })

        typer.echo("Dados exportados para 'top10_skills.csv'")
    except Exception as e:
        typer.echo(f"Erro ao exportar para CSV: {e}", err=True)
```

No terminal: python projetoFinal.py list_skills fullstacks --csv

```
[
  {
    "skill": "testing",
    "count": 2
  },
  {
    "skill": "git",
    "count": 1
  },
  {
    "skill": "linux",
    "count": 1
  },
  {
    "skill": "python",
    "count": 1
  },
  {
    "skill": "continuous-integration",
    "count": 1
  },
  {
    "skill": "software-architecture",
    "count": 1
  },
  {
    "skill": "high-availability",
    "count": 1
  },
  {
    "skill": "ruby",
    "count": 1
  },
  {
    "skill": "web-services",
    "count": 1
  },
  {
    "skill": "elasticsearch",
    "count": 1
  }
]
Dados exportados para 'top10_skills.csv'
```

Conclusão

A aplicação desenvolvida permitiu explorar de forma eficaz a API do itjobs.pt, funcionando como um **API Explorer** para recolher e analisar anúncios de emprego. A CLI em Python tornou o processo simples e reutilizável, desde a recolha dos dados até à contagem de *skills* técnicas e comportamentais.

A integração de **web scraping** do website Teamlyzer permitiu complementar a informação da API com dados adicionais sobre empresas e competências mais procuradas.

Durante o desenvolvimento surgiram desafios como os limites de requisições, a diversidade das descrições dos anúncios e o uso do Git em equipa, ultrapassados com armazenamento local dos dados, filtragem intermédia e melhor organização do código. Estas experiências ajudaram-nos a reforçar as nossas competências técnicas.

FIM