Universidade do Minho
Escola de Ciências

WEB SCRAPING

Criação de uma CLI (Command Line Interface) com base no website Teamlyzer

Discentes:

Filipe Araújo(A112467)

Marta Rocha (A112387)

Olga Marynych (A79235)

Docentes:

Prof. Tiago Baptista

Prof. Nuno Macedo

Prof.Pedro Rangel Henriques

dezembro, 2025

Índice

Introdução

Este projeto dá continuidade ao processo de recolha, armazenamento e processamento de anúncios de emprego na área das Tecnologias de Informação, iniciado no primeiro trabalho prático. Nesta fase, recorre-se ao processo de *web scraping* do site pt.teamlyzer.com.

O trabalho contempla o desenvolvimento de três interfaces de linha de comandos (CLI), apresentando para cada uma o respetivo código em Python e o *output* obtido no terminal.

Embora este relatório seja independente do relatório do primeiro trabalho prático, todo o código Python foi desenvolvido no mesmo ficheiro, projetoFinal.py. Devido à extensão de alguns comandos, o código foi organizado e delimitado através de comentários, permitindo separar claramente os dois trabalhos práticos e as respetivas alíneas.

Por fim, importa referir que os nomes dos comandos foram definidos em inglês, seguindo os exemplos apresentados no enunciado.

O trabalho encontra-se organizado em dois capítulos principais: o Capítulo 1 descreve, passo a passo, os objetivos de cada CLI e detalha a implementação do respetivo código Python; o Capítulo 2 apresenta a execução dos comandos e o respetivo output no terminal, permitindo verificar o correto funcionamento das CLI e os resultados obtidos.

1. Construção e Funcionamento de Cada CLI

Neste capítulo descreve-se, passo a passo, o objetivo de cada CLI e a implementação do código Python para cada uma delas.

1.1 Comando “get”

Objetivo: Esta CLI permite obter informações detalhadas sobre um anúncio de emprego específico a partir do seu *jobID*, consultando a API do ITJobs.pt e complementando com dados sobre a empresa que publica a vaga, extraídos do site pt.teamlyzer.com. O utilizador fornece o *jobID* como argumento e pode optar por exportar os resultados para CSV. A função retorna informações adicionais sobre a empresa, incluindo o rating geral (valor entre 0 e 5), a descrição da empresa e os principais benefícios de trabalhar nela.

Inicialmente, a aplicação consulta a API do ITJobs para recuperar os dados do trabalho, incluindo informações básicas sobre a empresa que o publica.

```
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)
```

Para complementar estes dados, é utilizada uma função auxiliar que acessa páginas web e retorna o conteúdo HTML, garantindo que a aplicação consiga ler a informação mesmo diante de eventuais erros de conexão.

```
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
```

Em seguida, o nome da empresa é processado para gerar o "*slug*" correto necessário para construir a URL da página correspondente no site Teamlyzer. Este processo envolve a normalização do nome, substituição de caracteres especiais e teste de alternativas caso o *slug* gerado inicialmente não seja válido, garantindo a correspondência correta da página da empresa.

```python
#Exemplo: 'Team.IT' -> 'team-it', 'We Are META' -> 'we-are-meta'.
#Mantém o mesmo nome da função para compatibilidade.
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)
```

Com o URL da empresa, a função faz a leitura da página HTML utilizando a função auxiliar ler_html, abordada anteriormente. Caso a página não esteja acessível, retorna valores padrão indicando a indisponibilidade dos dados. Se a página estiver acessível, a função procede à extração de informações específicas.

```
# 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
```

O rating da empresa é obtido a partir de um elemento HTML específico, processando o texto para capturar o valor numérico correto. Para a descrição, a função localiza o elemento correspondente, remove segmentos indesejados como links de “ler mais/menos” e limita o texto aos primeiros 1000 caracteres.

```
# 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
```

Para os benefícios, a função percorre diferentes seções da página, incluindo uma página alternativa caso a seção principal não exista, garantindo a captura de todos os itens relevantes e eliminando duplicados. O resultado final é armazenado como uma string separada por ponto e vírgula.

```
# 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)
```

Por fim, a função busca o salário médio nos painéis da página que contêm o ícone de euro, filtrando valores incorretos como percentagens, limpando o HTML residual e extraindo apenas os valores monetários. Se nenhum valor válido for encontrado, o campo recebe uma indicação de “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
                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
```

O dicionário final retornado contém, portanto, quatro campos principais: *teamlyzer_rating*, *teamlyzer_description*, *teamlyzer_benefits* e *teamlyzer_salary*, fornecendo uma visão completa e estruturada sobre a empresa associada ao anúncio de emprego.

```
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}", err=True)
        return {
            "teamlyzer_rating": 0.0,
            "teamlyzer_description": "Página não acessível",
            "teamlyzer_benefits": "",
            "teamlyzer_salary": ""
        }

    resultados = {}
```

Finalmente, a CLI apresenta os resultados no terminal em formato JSON, facilitando a leitura. Opcionalmente, o utilizador pode exportar os dados para um ficheiro CSV, contendo os campos essenciais do anúncio, o que permite análises posteriores ou armazenamento local. Esta integração entre os dados da API do ITJobs e o Teamlyzer oferece uma visão completa e estruturada sobre cada vaga, tornando a CLI uma ferramenta prática e eficiente para a recolha e organização de informações sobre ofertas de emprego.

1.2 Comando “Statistics”

Objetivo: Esta CLI realiza a contagem de vagas por tipo ou nome da posição e por região. A função processa os dados de um ficheiro JSON contendo os anúncios de emprego, organiza as contagens por combinação de zona e tipo de trabalho, e gera um ficheiro CSV contendo três colunas: Zona, Tipo de Trabalho e Nº de vagas. Este ficheiro permite a análise estatística da distribuição de vagas por região e função.

Ao ser executada, a função tenta carregar o ficheiro *empregos.json*; caso o ficheiro não exista, informa o utilizador e encerra a execução, garantindo que os dados necessários estejam disponíveis.

```
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)
```

Em seguida, percorre todos os anúncios de emprego contidos no ficheiro, extraindo para cada anúncio a(s) localização(ões) e o título da posição. Para cada combinação de zona e tipo de trabalho, é mantido um contador que acumula o número de vagas correspondentes.

```
    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
```

Após a contagem, os dados são organizados numa lista de dicionários, com colunas padronizadas: Zona, Tipo de Trabalho e Nº de vagas. Esta lista é ordenada por zona e, dentro de cada zona, por tipo de trabalho, facilitando a leitura e posterior análise.

```
    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"]))
```

Por fim, a função exporta os resultados para um ficheiro CSV, cujo nome pode ser definido pelo utilizador via opção de linha de comando, permitindo assim a criação de relatórios e estatísticas externas de forma prática.

```
    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!")
```

1.3 Comando “list_skills”

Objetivo: Esta CLI permite recolher as principais dez competências (denominadas “stack tecnológica” no website) associadas a uma determinada profissão. A função analisa a página correspondente à profissão no Teamlyzer, contabiliza a frequência de cada competência e apresenta, para cada uma, o seu nome e o número de ocorrências. Os resultados podem ser visualizados no terminal em formato JSON e, opcionalmente, exportados para um ficheiro CSV, permitindo análises posteriores ou integração em relatórios externos.

O utilizador fornece o nome da profissão como argumento e, opcionalmente, pode exportar os resultados para um ficheiro CSV. A função inicia a execução realizando uma requisição HTTP à página correspondente à profissão, obtendo o HTML e processando-o com BeautifulSoup para extrair os elementos que contêm as competências mencionadas nos anúncios de emprego.

```
@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')
```

Em seguida, a função percorre todos os elementos encontrados, contabilizando a ocorrência de cada competência. A classe da *div* usada para localizar os elementos foi definida com base na inspeção da estrutura do site *Teamlyzer*, garantindo que apenas os elementos corretos fossem capturados.

Após a contagem, os resultados são ordenados em ordem decrescente de frequência, sendo selecionadas as dez competências mais relevantes. Estes dados são apresentados no terminal em formato JSON, facilitando a visualização imediata.

```
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)
```

Se o utilizador optar pela exportação, a função gera um ficheiro CSV contendo o nome da profissão, a competência e o número de vezes em que esta foi identificada, permitindo análises posteriores ou utilização em relatórios externos. Esta abordagem fornece uma visão rápida das competências mais valorizadas para cada função, integrando de forma eficiente a extração de dados web com a contagem estatística das habilidades.

```
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)
```

2. CLI: execução dos comandos

Neste capítulo, apresenta-se a execução de cada comando, acompanhada do respetivo output mostrado no terminal, permitindo verificar o funcionamento das CLI e os resultados obtidos.

2.1 Comando “get”

Para o exemplo obtido, utilizou-se o ID=508050.

Exemplo de execução:

```
python projetoFinal.py get 508050
```

Output obtido:

```
 A extrair informações da empresa...
{
    "id": 508050,
    "teamlyzer_rating": 2.6,
    "teamlyzer_description": "A KWAN é uma empresa 100% portuguesa, que faz parte do
Grupo Rupeal, dedicada ao IT Staffing, consultoria e outsourcing. A KWAN existe porqu
e um developer acreditou numa forma diferente de recrutar: Tratar as pessoas com o re
speito que merecem. O nosso objectivo é sermos o teu go-to-place para aconselhamento
de carreira e ajudar-te a chegar ao teu dream job.",
    "teamlyzer_benefits": "Flexibilidade de horário; Mais de 22 dias de férias; Permi
te período sabático; Seguro de saúde; Suporte psicológico; Flex benefits; Plafond cer
tificações e conferências; Programa de descontos; Igualdade racial; LGBTQ+ friendly;
CI / CD; Metodologias agéis; Pair programming; Tecnologias de ponta; Equipa com senio
ridade elevada; Incentivo à criatividade & inovação; Depende do projeto / equipa",
    "teamlyzer_salary": "1.916€ - 2.116€"
}
```

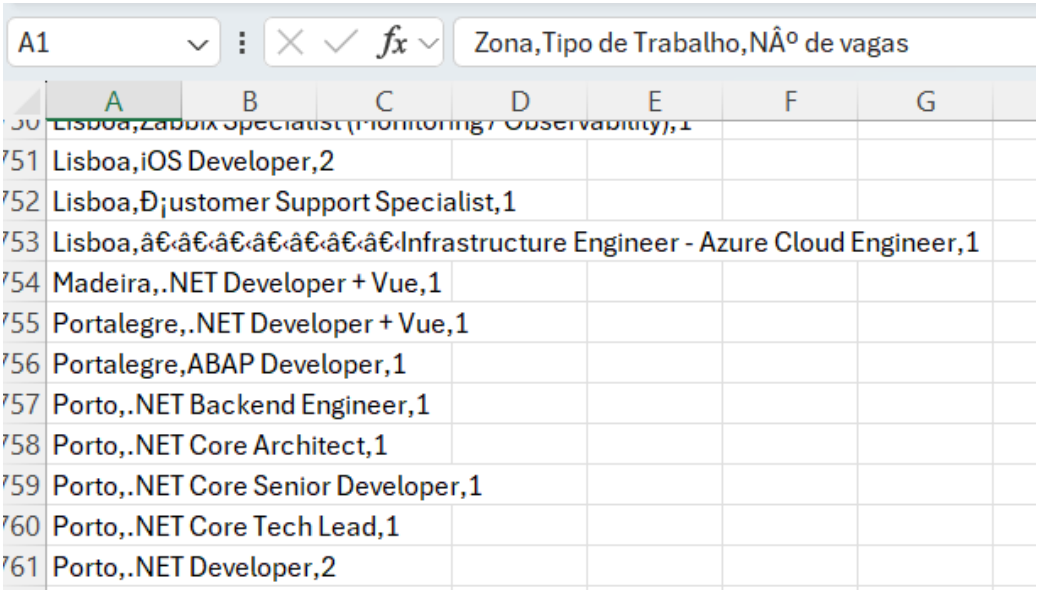
2.2 Comando “statistics”

Exemplo de execução:

```
python projetoFinal.py statistics zone
```

Output obtido:

```
Ficheiro de exportação 'statistics_zone.csv' criado com sucesso!
```



	A	B	C	D	E	F	G
751	Lisboa,iOS Developer,2						
752	Lisboa,Ð¡ustomer Support Specialist,1						
753	Lisboa,â€‹â€‹â€‹â€‹â€‹â€‹Infrastructure Engineer - Azure Cloud Engineer,1						
754	Madeira,.NET Developer + Vue,1						
755	Portalegre,.NET Developer + Vue,1						
756	Portalegre,ABAP Developer,1						
757	Porto,.NET Backend Engineer,1						
758	Porto,.NET Core Architect,1						
759	Porto,.NET Core Senior Developer,1						
760	Porto,.NET Core Tech Lead,1						
761	Porto,.NET Developer,2						

2.3 Comando “list_skills”

Para o exemplo de execução foi utilizada a profissão “fullstack (developer)”.

Exemplo de execução:

```
python projetoFinal.py list_skills fullstack
```

Output obtido:


```
[
  {
    "skill": "react",
    "count": 8
  },
  {
    "skill": "java",
    "count": 6
  },
  {
    "skill": "sql",
    "count": 6
  },
  {
    "skill": "angular",
    "count": 5
  },
  {
    "skill": "python",
    "count": 4
  },
  {
    "skill": "node.js",
    "count": 4
  },
  {
    "skill": "azure",
    "count": 4
  },
  {
    "skill": "machine-learning",
    "count": 3
  },
  {
    "skill": "artificial-intelligence",
    "count": 3
  },
  {
    "skill": "amazon-web-services-aws",
    "count": 3
  }
]
```

CONCLUSÃO

O desenvolvimento deste trabalho prático permitiu consolidar os conhecimentos adquiridos em *Web Scraping*, manipulação de APIs REST e processamento de dados em Python. A implementação das três CLI demonstrou a capacidade de integrar informações de diferentes fontes, organizar dados de forma estruturada e fornecer outputs tanto no terminal quanto em ficheiros CSV, facilitando a análise e utilização posterior.

Apesar do sucesso técnico, a equipa enfrentou algumas dificuldades no processo de gestão de versões com o Git. Inicialmente, cada membro trabalhava em ficheiros separados sem sincronizar corretamente com o ramo principal, o que levou a conflitos durante as tentativas de *pull request*. Esta situação obrigou-nos a reorganizar o código fora do repositório, unindo todas as funcionalidades num único ficheiro funcional, que foi posteriormente adicionado diretamente à *main* já totalmente pronto.

Esta experiência evidenciou não apenas a importância de um planeamento cuidadoso do código e da integração de funcionalidades, mas também a necessidade de práticas corretas de controlo de versões, mostrando que a colaboração eficiente depende tanto da organização do trabalho como do domínio técnico da linguagem.