EMPRESAS IT SCRAPPER

Documentação do programa e elaborado por João Cardoso, nº11

MARÇO 7, 2024

IEFP
N-EFBG-EFA NS Pro - Programador(a) de Informática 01-23
UFCD 24 (10794) - Programação avançada com Python
Formador: Marcos De Lima Alvarães
Formando: João Pedro Miranda Cardoso, nº11

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
EUROPA
REPÚBLICA PORTUGUESA

Índice

Breve introdução do projeto

O programa de Empresas TI scrapper é uma ferramenta onde o utilizador pode obter informações sensíveis sobre empresas TI em Portugal.

Através de SQL queries, o utilizador pode descobrir informações como nome, website, rating etc...

O administrador tem a capacidade de ativar o programa para executar o bot, que fará o download automático da recolha de dados.
Este projecto implementa funcionalidades para scrape de informações de empresas do website pt.teamlyzer.com e inseri-las numa base de dados MySQL.

Documentação do Ficheiro Python

App.py

Este arquivo Python é um aplicativo Flask que serve como um sistema para webScrape de dados de empresas de TI em Portugal, além de fornecer funcionalidades de login, consultas de utilizadors e administração de dados.

Ele inclui rotas para páginas como login, logout, consulta de utilizador e página do administrador. Quando o utilizador faz login com sucesso, ele é redirecionado para a página correspondente com base em seu papel de utilizador ou administrador.

As principais funções incluem:

login(): Verifica as credenciais do utilizador no banco de dados e redireciona para a página adequada (utilizador ou admin) após o login.

logout(): Encerra a sessão do utilizador.

queryUser(): Executa uma consulta SQL inserida pelo utilizador e exibe os resultados na página, se houver.

adminPage(): Página dedicada à administração do sistema, permitindo adicionar ou excluir empresas, bem como atualizar os índices de seus sites.

Além disso, o aplicativo Flask é configurado para se conectar a um banco de dados MySQL utilizando as variáveis de configuração fornecidas no arquivo config.py.

```
def connect_to_database():  # Definir a função connect_to_database
    return connect(  # Retornar uma conexão à base de dados
        host=mysql_database_host,  # Utilizar a variável
mysql_database_host do ficheiro config.py
        user=mysql_database_user,  # Utilizar a variável
mysql_database_user do ficheiro config.py
        password=mysql_database_user_password,  # Utilizar a variável
mysql_database_user_password do ficheiro config.py
        database=mysql_database_name,  # Utilizar a variável
mysql_database_name do ficheiro config.py

        charset='utf8mb4',  # Utilizar o conjunto de caracteres utf8mb4
(suporta emojis) e charset português
        collation='utf8mb4_unicode_ci'  # Utilizar a colação
utf8mb4_unicode_ci (suporta emojis)
    )


@app.route('/')  # Decorador que associa a função index à rota /
def index():  # Definir a função index (página inicial de origem)

    return render_template('base.html')  # Renderizar o modelo base.html


@app.route('/login', methods=['GET', 'POST'])  # Decorador que associa a
função login à rota /login
def login():
    if request.method == 'POST':
        # Get username and password from the form
        username = request.form['username']
        password = request.form['password']
```

```
        db = connect_to_database()
        cursor = db.cursor()
        cursor.execute('SELECT * FROM users WHERE username=%s AND
password=%s', (username, password))
        user = cursor.fetchone()  # Obter o primeiro registo da tabela
users com o username e password fornecidos
        cursor.close()  # Fechar o cursor

        if user:
            session['username'] = user[1]  # Store the username in the
session
            if user[3] == 'user':
                return redirect(url_for('queryUser'))  # Redirect to the
admin dashboard
            else:
                return redirect(url_for('adminPage'))  # Redirect to the
user dashboard

    return render_template('login.html')  # Render the login page with an
error message


@app.route('/logout')  # Decorador que associa a função logout à rota
/logout
def logout():
    session.pop('username', None)  # Remove the username from the session
and none is the default value
    return redirect(url_for('login'))


@app.route('/queryUser', methods=['GET', 'POST'])  # Decorador que
associa a função success à rota /success
def queryUser():  #

    if request.method == 'POST':
        query = request.form['query']
        table, column_names = executeQuery(query) # Executar a query e
obter a tabela e os nomes das colunas da tabela
        return render_template('queryUser.html', table=table,
column_names=column_names) #renderiza a página com a tabela (se houver
query
    else:
        return render_template('queryUser.html') #renderiza a página com
a tabela vazia para o caso de não haver query


def executeQuery(query):
    db = connect_to_database()
    cursor = db.cursor()
```

```
    cursor.execute(query)
    table = cursor.fetchall()
    column_names = [column[0] for column in cursor.description]  #
Receber os nomes das colunas da tabela resultante da query
    cursor.close()
    db.close()
    return table, column_names # Retornar a tabela e os nomes das colunas

@app.route('/adminPage', methods=['GET', 'POST'])  # Decorador que
associa a função success à rota /success
def adminPage():
    message = ""
    message2 = ""
    if request.method == 'POST':
        empresa_eliminar = request.form.get('deleteInputField') # Obter o
valor do campo de texto com o nome da empresa a eliminar
        if empresa_eliminar: #Em python empty strings é falso e strings
com conteudo é verdadeiro
            if eliminar_empresa(empresa_eliminar): # Se a empresa for
eliminada com sucesso
                message += "Empresa eliminada com sucesso! "
            else: # Se a empresa não existir
                message += "Empresa para eliminar não existe!"


        empresa_adicionar = request.form.get('addInputField') # Obter o
valor do campo de texto com o nome da empresa a adicionar
        if empresa_adicionar:
            if scrape_empresa(empresa_adicionar):
                message += "Empresa adicionada com sucesso!"
            else:
                message += "Empresa para adicionar não existe!"

        link_min=request.form.get('inputField1')
        link_max=request.form.get('inputField2')
        if link_min and link_max: # Se link_min e link_max não forem
vazios
            lista_empresas_novas=[]
            lista_empresas_novas=atualizar_website_index(int(link_min),
int(link_max)) # Atualizar o website_index das empresas entre link_min e
link_max com cast para int
            lista_empresas_novas_str = ', '.join(map(str,
lista_empresas_novas)) # Converter a lista para uma string separada por
vírgulas e adicionar à mensagem
            message2 += f"Empresas atualizadas com sucesso:
{lista_empresas_novas_str}"
        else:
            message2 += "Não foi possível atualizar as empresas!"
```

```
    return render_template('adminPage.html', message=message,
message2=message2) # Renderizar a página adminPage.html com a mensagem



if __name__ == '__main__':  # Se o módulo for executado como script (não
foi importado)
    app.run(debug=True)  # Executar o servidor web embutido do Flask em
modo de depuração
# Para executar o servidor web embutido do Flask, executar o comando
python app.py no terminal
```

Variáveis de Ambiente

PROXY: Variável de ambiente que armazena o proxy a ser utilizado para fazer os pedidos HTTP. Esta variável é utilizada na configuração dos proxies para as requisições.

```
PROXY = os.getenv('PROXY') #Este é o proxy que vamos usar para fazer os
pedidos
proxies = {
    "http": PROXY,
    "https": PROXY,
}
```

Funções Principais

scrape_empresa(website_index):

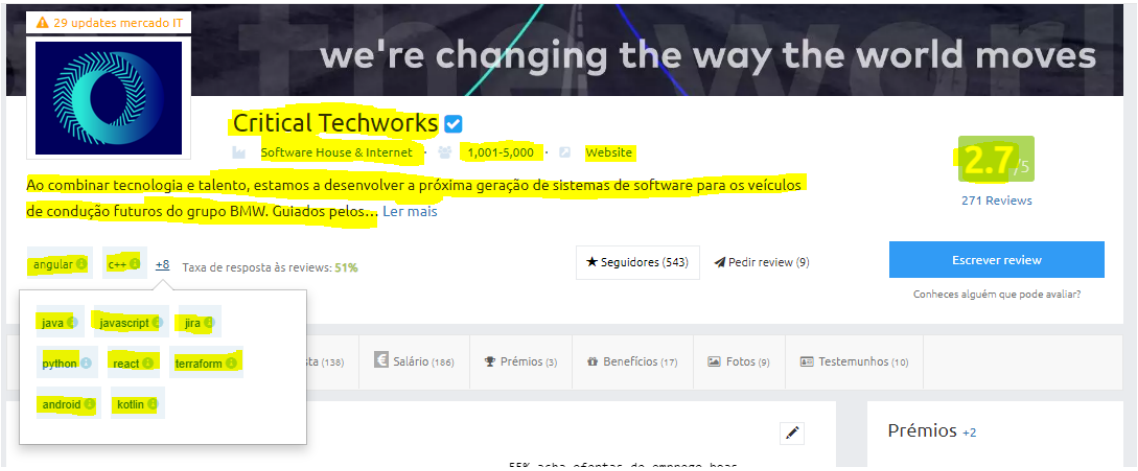
Esta função recebe um índice de empresa do website pt.teamlyzer.com e faz uma requisição HTTP para obter informações sobre a empresa.

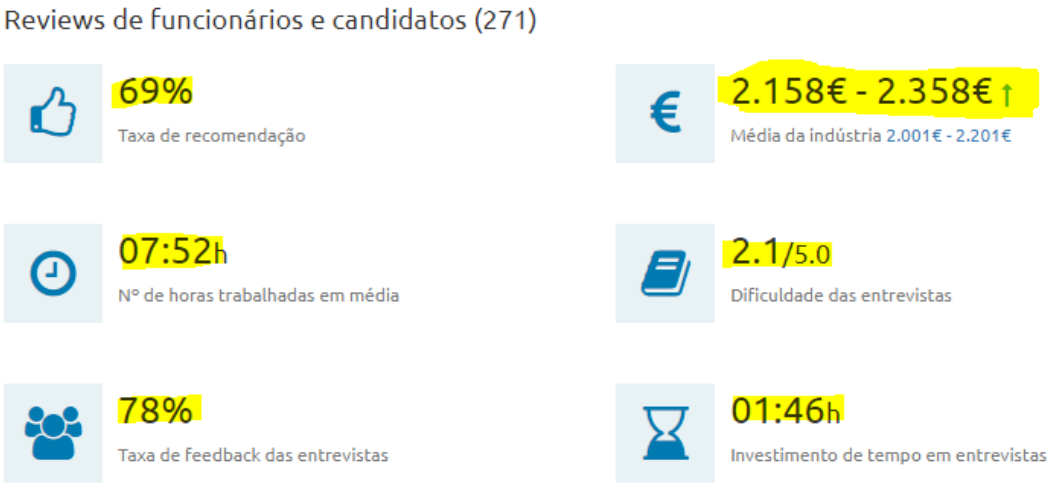
Utiliza a biblioteca BeautifulSoup para analisar o HTML da página e extrair as informações relevantes.

Constrói consultas SQL para inserir as informações da empresa na base de dados MySQL, tratando possíveis erros como o código de erro 404.

Retorna True se as informações foram inseridas com sucesso, caso contrário retorna False.

As imagens abaixo representa visualmente o que a função scrape_empresa(website_index) recolhe para cada URL.






```
def scrape_empresa(website_index):
    response = requests.get("https://pt.teamlyzer.com"+website_index, proxies=proxies) #Aqui estamos a fazer o pedido à página
    soup= BeautifulSoup(response.text, "html.parser") #Aqui estamos a criar um objeto BeautifulSoup

    info_total = soup.select('div.col-lg-8:not(.ethical_ad)') #Seleccao de todas as divs com a classe col-lg-8 e que não tenham a classe ethical_ad

    #encontrar dentro do soup <h1>ERRO 404!</h1>
    if soup.find("h1").text == "ERRO 404!":
        print("Encontrou um erro 404!")
        return #terminar funcao

    #SQL para inserir os dados na tabela empresa
    sql = f"""
    INSERT INTO `bdptanalyser`.`empresa` (
        `nome`, `website`, `descricao`, `rating`, `taxa_recomendacao`,
        `salario_medio`, `horas_trabalho`, `dificuldade`, `tx_feedback`,
```

```
        `investimento_tempo`, `website_index`, `trabalhador_id`,
`industria_id`
    )
    VALUES (
        '{nome_empresa(info_total)}', '{website(info_total)}',
'{descricao(info_total)}', '{rating(info_total)}',
        '{tx_recomendacao(info_total)}', '{salario_medio(info_total)}',
'{horas_trabalho(info_total)}',
        '{dificuldade_entrevista(info_total)}',
'{tx_feedback(info_total)}', '{investimento_tempo(info_total)}',
'{website_index}',
        (SELECT id FROM bdptanalyser.trabalhador WHERE
numero='{trabalhadores(info_total)}'),
        (SELECT id FROM bdptanalyser.industria WHERE
designacao='{industria(info_total)}')
    )
    ON DUPLICATE KEY UPDATE
        nome = VALUES(nome),
        website = VALUES(website),
        descricao = VALUES(descricao),
        rating = VALUES(rating),
        taxa_recomendacao = VALUES(taxa_recomendacao),
        salario_medio = VALUES(salario_medio),
        horas_trabalho = VALUES(horas_trabalho),
        dificuldade = VALUES(dificuldade),
        tx_feedback = VALUES(tx_feedback),
        investimento_tempo = VALUES(investimento_tempo),
        website_index = VALUES(website_index),
        trabalhador_id = VALUES(trabalhador_id),
        industria_id = VALUES(industria_id)
    ;
    """
    numero_linhas=inserir_base_dados(sql)

    #SQL para inserir os dados na tabela tecnologias porque esta tabela é
uma tabela de muitos para muitos
    stacks = tecnologias(soup)
    for tecnologia in stacks:
        sqlTech = f"""
        INSERT INTO `bdptanalyser`.`tecnologias_has_empresa`
(`tecnologias_id`, `empresa_id`)
        SELECT t.id, e.id
        FROM bdptanalyser.tecnologias t
        JOIN empresa e ON e.website_index = '{website_index}'
        WHERE t.nome = '{tecnologia}'
        AND NOT EXISTS (
            SELECT 1
            FROM bdptanalyser.tecnologias_has_empresa te
            WHERE te.tecnologias_id = t.id
```

```
            AND te.empresa_id = e.id
        );
        """
        inserir_base_dados(sqlTech)
    # Execute the SQL statement
    return numero_linhas>0 #Se o numero de linhas afetadas for maior que
0 true, senao false
```

atualizar_website_index(min_page, max_page):

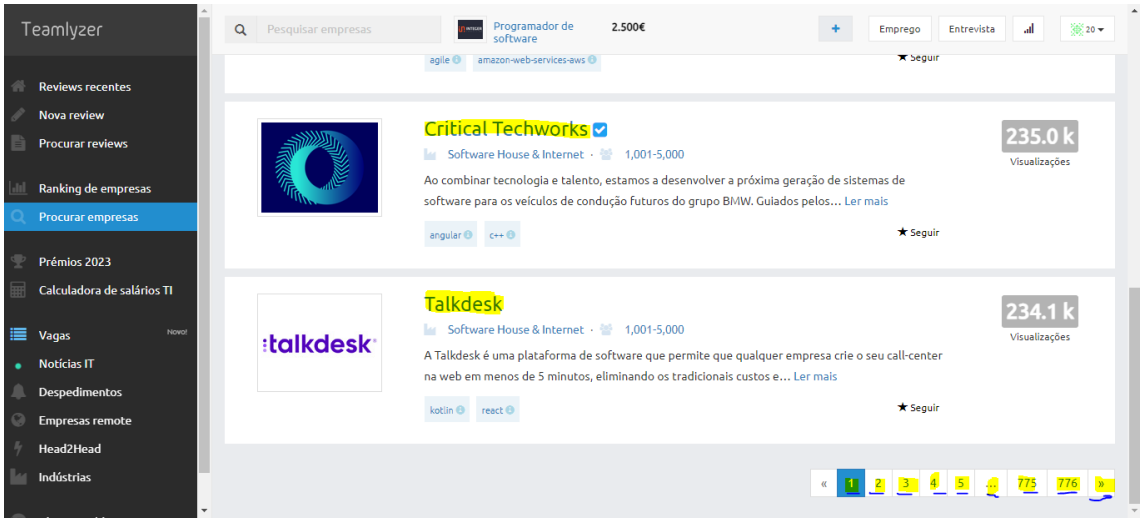
Esta função atualiza os índices das empresas na base de dados verificando a existência de novas empresas entre os índices min_page e max_page.

Utiliza a função get_companies(min_page, max_page) para obter a lista de empresas a serem verificadas.

Para cada empresa na lista, verifica se já existe na base de dados. Se não existir, utiliza a função scrape_empresa(website_index) para Scrape as informações e inseri-las na base de dados.

Retorna uma lista com os índices das novas empresas adicionadas.

Aqui é um screenshot da informação recolhida desta *pagina*. O que está amarelo, é a informação recolhida da função atualizar_website_index(min_page, max_page).




```
def atualizar_website_index(min_page, max_page):
    lista_empresas_verificar = get_companies(min_page, max_page) #Obter a
lista de empresas no site
    lista_empresas_base_dados = buscar_base_dados("SELECT website_index
FROM bdptanalyser.empresa") #Obter a lista de empresas na base de dados
    lista_empresas_novas=[]

    for website_index in lista_empresas_verificar: #Iterar sobre todas as
empresas no site
```

```
        if website_index not in lista_empresas_base_dados: #Se a empresa
não estiver na base de dados
            lista_empresas_novas.append(website_index) #Adicionar a
empresa à lista de empresas novas
            scrape_empresa(website_index) #Scrape a empresa
    return lista_empresas_novas #Devolver a lista de empresas novas
```

eliminar_empresa(website_index):

Esta função elimina uma empresa específica da base de dados utilizando o índice do website.

Antes de eliminar a empresa, utiliza a função eliminar_tecnologias(website_index) para eliminar as tecnologias associadas à empresa.

Retorna True se a empresa foi eliminada com sucesso, caso contrário retorna False.

```
def eliminar_empresa(website_index):
    eliminar_tecnologias(website_index) #Eliminar as tecnologias da
empresa antes de eliminar a empresa por causa da foreign key
    sql = f"DELETE FROM bdptanalyser.empresa WHERE
website_index='{website_index}'"
    numero_linhas = inserir_base_dados(sql)
    return numero_linhas>0 #Se o numero de linhas afetadas for maior que
0 true, senao false
```

eliminar_tecnologias(website_index):

Esta função elimina as tecnologias associadas a uma empresa específica da base de dados.

Retorna True se as tecnologias foram eliminadas com sucesso, caso contrário retorna False.

```
def eliminar_tecnologias(website_index):
    # SQL statement to delete a specific row from the table
    sql = f"DELETE FROM bdptanalyser.tecnologias_has_empresa WHERE
empresa_id IN (Select id FROM empresa WHERE
website_index='{website_index}')"
    inserir_base_dados(sql)
    numero_linhas = inserir_base_dados(sql)
    return numero_linhas>0 #Se o numero de linhas afetadas for maior que
0 true, senao false
```

Funções de Scrape Informações da Empresa

As funções nome_empresa(info_total), industria(info_total), trabalhadores(info_total), website(info_total), descricao(info_total), rating(info_total), tecnologias(soup), tx_recomendacao(info_total), salario_medio(info_total), horas_trabalho(info_total), dificuldade_entrevista(info_total), tx_feedback(info_total), investimento_tempo(info_total) segmentam informações específicas sobre a empresa a partir do HTML analisado pelo BeautifulSoup.

Cada função recebe o objeto BeautifulSoup correspondente à página HTML da empresa e retorna as informações específicas.

```
def nome_empresa(info_total):
    nome_empresa=info_total[0].find("h1", class_="reduce-h1 text-default-
xs").text.strip() #Encontramos aqui o nome da empresa
    return nome_empresa.replace("'", "-") #Aqui estamos a substituir
todas as aspas simples por um hifen para evitar problemas com a query

def industria(info_total):
    industria=info_total[0].find_all("div", class_="center_mobile hidden-
xs company_add_details")[0].text.strip().splitlines()
    #Encontramos aqui a industria da empresa e dividimos usando "line"
como separador
    return industria[0]

def trabalhadores(info_total):
    industria=info_total[0].find_all("div", class_="center_mobile hidden-
xs company_add_details")[0].text.strip().splitlines()
    #Encontramos aqui a industria da empresa e dividimos usando "line"
como separador
    return industria[2]

def website(info_total):
    allUrls=info_total[0].find_all("div", class_="center_mobile hidden-xs
company_add_details") #Aqui estamos a encontrar todas as divs com a
classe center_mobile
    urls=allUrls[0].find_all('a', href=lambda href: href and
href.startswith('http://') or href.startswith('https://') or
href.startswith('www.')) #Aqui estamos a encontrar todas as tags a que
tenham .com no href
    for url_tag in urls: #Aqui estamos a iterar sobre todas as tags a que
encontramos
        url = url_tag['href'] #Aqui estamos a obter o valor do atributo
href
    if len(urls) == 0:
        url = ""

    return url

def descricao(info_total):
```

```
    descricao=info_total[0].find("div", class_="ellipsis
center_mobile").text.strip() #Encontramos aqui a descricao da empresa
    descricao=descricao.replace("'", "-") #Aqui estamos a substituir
todas as aspas simples por um hifen para evitar problemas com a query
    descricao=descricao.replace('"', "-") #Aqui estamos a substituir
todas as aspas simples por um hifen para evitar problemas com a query
    return descricao

def rating(info_total):
    rating=info_total[0].select("div.text-center") #Aqui estamos a
encontrar todas as divs com a classe text-center

    #Aqui estamos a dividir a string em duas partes, usando " / " como
separador e a
    #ficar com a primeira parte
    rating=rating[0].text.strip().split("/", 1)[0]
    return rating

################################### Tecnologia
#############################################################################
#########################
def tecnologias(soup):

    try:
        #find in info_total the dive in tags voffset2 tags_popover
        tag_container=soup.find("div", class_="wrapper-tags-reviews-
answered-rate") #Aqui estamos a encontrar todas as divs com a classe
voffset2 e tags_popover
        # Find all <a> tags within the <div> with class "tags voffset2
tags_popover"
        tag_div = tag_container.find('div', class_='tags voffset2
tags_popover')
        tag_links = tag_div.find_all('a')

        # Extract tag names and URLs
        tag_info = [(link.text, link['href']) for link in tag_links]

        # Extract tags from the button's data-content attribute
        button_content = soup.find('button', class_='btn btn-link
button_plus')['data-content']
        button_soup = BeautifulSoup(button_content, 'html.parser')
        additional_tags = button_soup.find_all('a')

        # Add additional tags to tag_info list
        tag_info += [(tag.text, tag['href']) for tag in additional_tags]

        # Print tag names and URLs
        tag=[]
        for name in tag_info:
```

```
            tag.append(name[0])
    except:
        tag = ""
    return tag
################################### Reviews de funcionários e candidatos
#############################################################################
def tx_recomendacao(info_total):
    try:
        tx_recomendacao = info_total[0].find_all("p", class_="size-
h2")[0].text.strip()
    except IndexError:
        tx_recomendacao = "" #vazio se nao tiver recomendacao

    return tx_recomendacao


def salario_medio(info_total):

    try:
        salario_medio=info_total[0].find_all("p", class_="size-
h2")[1].text.strip() #Encontramos aqui o salario medio da empresa
    except IndexError:
        salario_medio = "" #vazio se nao tiver recomendacao

    return salario_medio

def horas_trabalho(info_total):
    try:
        horas_trabalho=info_total[0].find_all("p", class_="size-
h2")[2].text.strip() #Encontramos aqui o horas medio da empresa
    except IndexError:
        horas_trabalho = "" #vazio se nao tiver recomendacao

    return horas_trabalho

def dificuldade_entrevista(info_total):
    try:
        dificuldade_entrevista=info_total[0].find_all("p", class_="size-
h2")[3].text.strip().split("/", 1)[0]  #Encontramos aqui a dificuldade da
entrevista da empresa
    except IndexError:
        dificuldade_entrevista = "" #vazio se nao tiver recomendacao

    return dificuldade_entrevista

def tx_feedback(info_total):

    try:
```

```
        tx_feedback=info_total[0].find_all("p", class_="size-
h2")[4].text.strip() #Encontramos aqui a taxa de feedback da empresa
    except IndexError:
        tx_feedback = "" #vazio se nao tiver recomendacao

    return tx_feedback

def investimento_tempo(info_total):
    try:
        investimento_tempo=info_total[0].find_all("p", class_="size-
h2")[5].text.strip() #Encontramos aqui o investimento de tempo da empresa
    except IndexError:
        investimento_tempo = "" #vazio se nao tiver recomendacao

    return investimento_tempo
```

Funções de Base de Dados

As funções connect_to_database(), inserir_base_dados(sql), buscar_base_dados(sql), get_website_(column_name) conectam-se à base de dados MySQL, executam consultas SQL e recuperam informações da base de dados, respetivamente.

A função inserir_base_dados(sql) executa a consulta SQL fornecida e retorna o número de linhas afetadas pela consulta.

```
def connect_to_database(): # Definir a função connect_to_database
    mysql_database_user = "Raquel"
    mysql_database_user_password = "Silva1234"
    mysql_database_name = "bdptanalyser"
    mysql_database_host = "62.28.39.135"

    return connect( # Retornar uma conexão à base de dados
        host=mysql_database_host, # Utilizar a variável
mysql_database_host do ficheiro config.py
        user=mysql_database_user, # Utilizar a variável
mysql_database_user do ficheiro config.py
        password=mysql_database_user_password, # Utilizar a variável
mysql_database_user_password do ficheiro config.py
        database=mysql_database_name, # Utilizar a variável
mysql_database_name do ficheiro config.py
        charset='utf8mb4', # Utilizar o conjunto de caracteres utf8mb4
(suporta emojis) e charset português
        collation='utf8mb4_unicode_ci' # Utilizar a colação
utf8mb4_unicode_ci (suporta emojis)
    )

def inserir_base_dados(sql):
```

```
    import mysql.connector # Importar o módulo mysql.connector. Dá yellow
porque não está a ser usado, mas está a ser usado
    numero_linhas = 0
    try:
        # Connectar à base de dados
        db = connect_to_database()
        cursor = db.cursor()

        # Execute the SQL statement
        cursor.execute(sql)

        # Fazer commit da transação
        db.commit()

        numero_linhas = cursor.rowcount #Obter o número de linhas
afetadas pela query


    except mysql.connector.Error as err:
        print("Error:", err)

    finally:
        # Close MySQL connection
        if db.is_connected():
            cursor.close()
            db.close()
    return numero_linhas #Devolve o número de linhas afetadas pela query

def buscar_base_dados(sql):
    import mysql.connector
    try:
        # Connectar à base de dados
        db = connect_to_database()
        cursor = db.cursor()

        # Execute the SQL statement
        cursor.execute(sql)

        # Fetch the result
        results = cursor.fetchall()
        return results

    except mysql.connector.Error as err:
        print("Error:", err)

    finally:
        # Close MySQL connection
        if db.is_connected():
            cursor.close()
```

```
            db.close()
```

Funções Auxiliares

As funções get_companies(min_page, max_page) e scrape_tech(url) são funções auxiliares que auxiliam na obtenção de informações sobre as empresas e as tecnologias associadas a partir do website pt.teamlyzer.com.

Execução Principal

A execução principal do script está comentada (if __name__ == '__main__':). Quando descomentada, permite executar o script diretamente, Scrape informações de empresas e atualizar a base de dados.

Para usar este ficheiro, basta importá-lo no projeto Python e chamar as funções relevantes conforme necessário. Certifique-se de configurar as variáveis de ambiente necessárias, como o proxy, e de ter uma base de dados MySQL configurada corretamente para armazenar as informações das empresas.

```
def get_website_(column_name):
    try:
        # Connect to MySQL
        db = connect_to_database()
        cursor = db.cursor()

        # SQL statement to select specific text from the table
        sql = "SELECT {column1} FROM
`empresa`".format(column1=column_name)

        # Execute the SQL statement
        cursor.execute(sql)

        # Fetch the result
        results = cursor.fetchall()

        # Extract the specific text for every row
        specific_texts = [row[0] for row in results] #devolve a lista com
os valores da coluna website_index

        return specific_texts #devolve a lista com os valores da coluna
website_index

    except mysql.connector.Error as err:
        print("Error:", err)

    finally:
        # Close MySQL connection
        if db.is_connected():
            cursor.close()
            db.close()
```

```python
def get_companies(min_page, max_page):
    #Este loop irá correr todas as empresas existentes da pagina min e
max.
    #Assim descobrimos o index de cada empresa no site ptAnalyser para
percorrar depois toda a informaçao
    companies = [] #Dicionario com as empresas e os links
    for i in range(min_page, max_page):
        URL = "https://pt.teamlyzer.com/companies/?page=" + str(i)
        page = requests.get(URL)
        soup = BeautifulSoup(page.content, "html.parser")
        result= soup.find_all("h3", class_="voffset0") #
        #companies para lista

        for r in result:
            companies.append(f"{r.a['href']}") #nome da empresa e link a
um dicionario



    return companies

def scrape_tech(url="https://pt.teamlyzer.com/companies/"):

    response = requests.get(url, proxies=proxies)
    soup = BeautifulSoup(response.text, features="html.parser")

    tech=[]
    stack_container = soup.find_all("div", class_="form_field")[3]
    options = stack_container.find_all("option")
    for option in options:
        tech.append(option.text)
    #stack= stack_container.find_all("option")
    #print(stack)
    return tech #devolve a lista com as tecnologias

################################### Função principal
##############################################################################
#if __name__ == '__main__':

    #for i in range(0, len(website_index)):
     #  scrape_empresa(website_index[i])
      # print(website_index[i])
#    scrape_empresa('/companies/slim-business-solutions')

#novasEMpresas=atualizar_website_index(774, 779) # Atualizar o
website_index das empresas entre 1 e 2
#print(novasEMpresas)
```

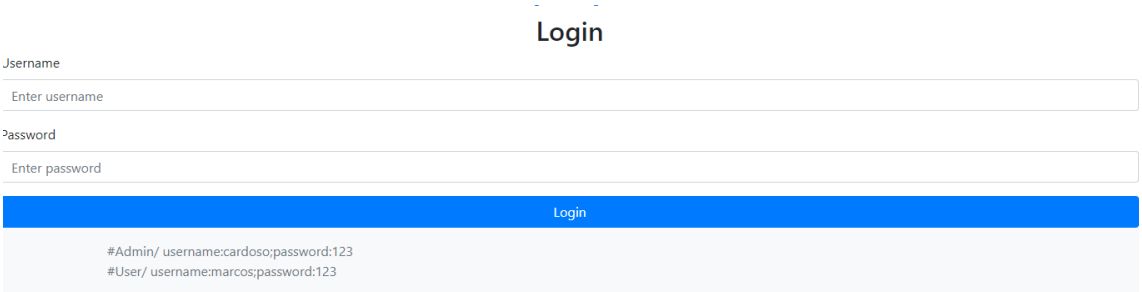
HTML

Index

Esta página constitui a estrutura básica de uma página web do programa. Utiliza o framework Bootstrap para o estilo e a formatação da página. O cabeçalho contém metadados e links para folhas de estilo e scripts necessários. A barra de navegação no topo permite ao utilizador fazer login e logout. O conteúdo principal da página é exibido dentro de um bloco chamado "content", onde são fornecidas informações sobre o programa de scrapping de dados, incluindo sua finalidade e funcionalidades. Por fim, no rodapé da página, são fornecidas credenciais para utilizador e administrador.

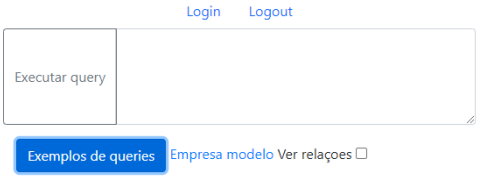
Login

A Página de Login é o ponto de entrada para os utilizadores acessarem o programa, onde podem introduzir as suas credenciais de autenticação. Com um layout simples e responsivo, apresenta campos para nome de utilizador e palavra-passe, além de uma área para exibição de mensagens de erro.

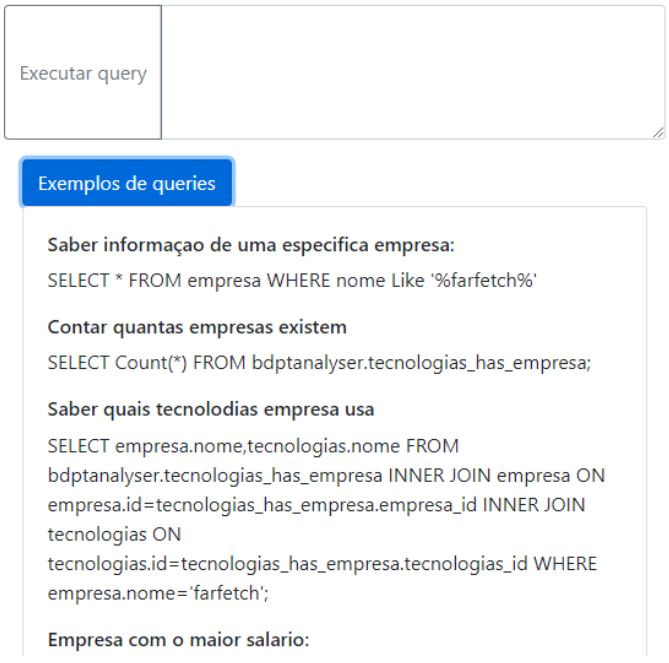


QueryUser

A página de QueryUser apresenta um formulário para execução de consultas SQL, onde os utilizadores podem introduzir as suas consultas e enviá-las para processamento. Além disso, disponibiliza exemplos de consultas predefinidas, permitindo aos utilizadores consultar informações específicas da base de dados. Após a submissão da consulta, os resultados são exibidos numa tabela abaixo do formulário. Adicionalmente, é possível visualizar relações entre entidades ao ativar a opção de visualização. O layout é responsivo, adaptando-se a diferentes dispositivos, e é suportado por um conjunto de scripts jQuery e Bootstrap para funcionalidades interativas e estilização.

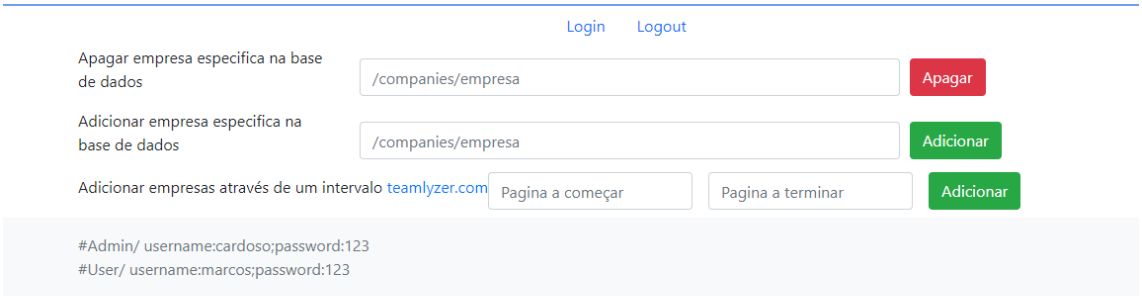



```
#Admin/ username:cardoso;password:123
#User/ username:marcos;password:123
```



AdminPage

A página AdminPage oferece funcionalidades de administração para manipulação de dados na base de dados. Dividida em seções, permite apagar empresas específicas, adicionar empresas manualmente e adicionar empresas em massa através de um intervalo de páginas fornecido pelo site teamlyzer.com. Cada seção é acompanhada por um formulário que captura os dados necessários e os envia para processamento no backend. Além disso, mensagens de sucesso são exibidas após a conclusão das operações, fornecendo feedback ao utilizador. O layout é responsivo, garantindo uma experiência consistente em diferentes dispositivos.



Base de dados

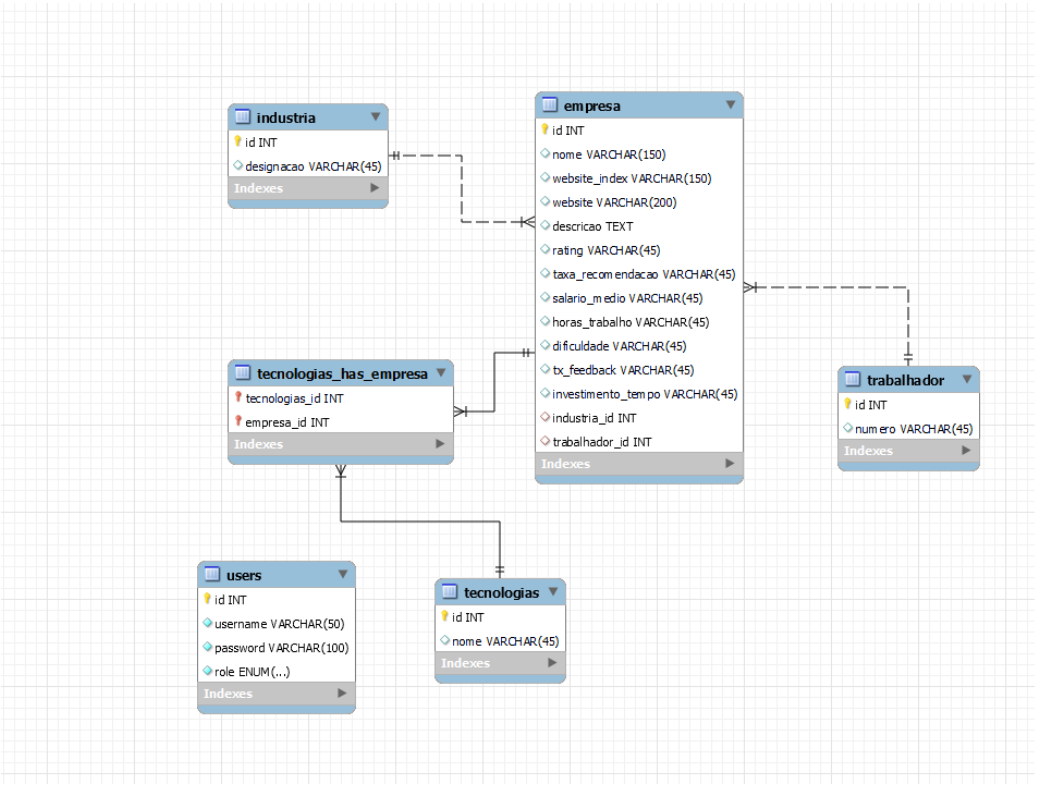


Figura 1:Estrutura da base de dados da bdptanalyser

Foi criada a base de dados *bdptanalyser* para guardar dados deste projeto.

A tabela *empresa* possui uma estrutura que armazena informações detalhadas sobre diferentes empresas. Cada entrada na tabela é identificada por um ID único, que é autoincrementado. Além disso, a tabela inclui campos para o nome da empresa, o índice do website, o website, a descrição da empresa, o rating, a taxa de recomendação, o salário médio, as horas de trabalho, a dificuldade das entrevistas, a taxa de feedback e o investimento de tempo. Existem também campos para as chaves estrangeiras industria_id e trabalhador_id, que se referem às indústrias e trabalhadores associados, respectivamente. A tabela utiliza o mecanismo InnoDB e o conjunto de caracteres utf8mb3. Há também restrições de chave estrangeira definidas para garantir a integridade referencial com as tabelas industria e trabalhador.

A tabela *tecnologias* armazena diferentes tecnologias, cada uma identificada por um ID único e um nome.

A tabela *industria* armazena informações sobre indústrias, onde cada indústria é identificada por um ID único e um nome de designação.

A tabela *tecnologias_has_empresa* é uma tabela de junção que mapeia a relação entre tecnologias e empresas. Ela armazena os IDs das tecnologias e das empresas relacionadas, garantindo a integridade referencial com as tabelas tecnologias e empresa.

A tabela *trabalhador* armazena informações sobre os trabalhadores, cada um identificado por um ID único e um número.

Bibliografia

Primeiro demo scrapper com a biblioteca Beautiful Soup (Real Python): Tutorial introdutório fornecido pelo Real Python, apresentando os conceitos básicos de web scraping utilizando a biblioteca Beautiful Soup em Python.

Youtube: Fonte de vídeos educativos e tutoriais que oferecem insights sobre desenvolvimento web, web scraping e manipulação de dados, contribuindo para uma compreensão mais profunda desses tópicos.

StackOverflow: Plataforma onde desenvolvedores podem fazer perguntas e encontrar respostas sobre uma variedade de tópicos de programação, fornecendo soluções para problemas específicos encontrados durante o desenvolvimento do projeto.

Deep.Ai: Assistente de IA para compreensão de diversos tópicos e otimização de algoritmos.