Mestrado em Informática Médica

Processamento de Linguagem Natural em Engenharia Biomédica

Trabalho Prático 2

Realizado por:
Beatriz Rodrigues, PG53696
Bruno Machado, PG53709
Rúben Ganança, PG54203

Docentes: José Almeida e Luís Cunha

11 de junho de 2024

Índice

1. Introdução

Este relatório tem como principal objetivo desenvolver um sistema que use os ficheiros *JSON* obtidos no primeiro trabalho prático realizado na UC de Processamento de Linguagem Natural em Engenharia Biomédica. Assim sendo, numa primeira parte do trabalho foram melhorados os ficheiros obtidos no trabalho anterior e adicionadas novas informações a partir de um *site*, recorrendo à biblioteca *Beautiful Soup*.

De seguida, de forma a obter o melhor conjunto de dados possíveis, concatenou-se os *datasets* e foram removidas chaves repetidas, agrupando-se informação que fosse relevante. De modo a ser possível visualizar este conceitos ou manipulá-los criou-se também uma plataforma *web* recorrendo ao *Flask*.

2. Enriquecer o Dataset

2.1 Correção dos *datasets* anteriores

De maneira a dar continuidade ao trabalho feito previamente foi necessário corrigir alguns erros que existiam nos *datasets* antes de fazer a concatenação dos mesmos. Assim sendo, foi reutilizado do trabalho prático 1 os *scripts* que criavam cada um dos *datasets* e os respetivos documentos `xml` , de forma a poder ler novamente os `xml` e criar um novo *dataset* de cada um deles sem os erros ou gralhas encontradas.

Para este efeito algumas das coisas que foram alteradas nos ficheiros `json` foram:

- Transformação das designações dos conceitos para minúsculas, de forma a facilitar uma posterior comparação entre *datasets* de chaves idênticas;
- Remoção de pontuação existente nas chaves, uma vez que a pontuação existente, por exemplo, na designação do conceito não trás informação relevante e só dificulta a comparação de designações;
- Remoção no *wipo.json* dos sinónimos existentes nas traduções. Esta transformação é importante, uma vez que de forma a juntar as traduções em português presentes dentro de um conceito com uma mesma designação proveniente de outro `json`, a tradução em português tem de estar na forma mais simples possível para poder ser comparada.

Como se pode observar através das figuras 2.1 e 2.2, as alterações feitas tornam a comparação entre o termo em português presente no ficheiro *wipo_modificado.json* muito mais fáceis de comparar com as restantes designações dos conceitos nos outros *json* do que a tradução em português presente no documento extraído do trabalho prático 1.

```
"Traducoes": {
    "AR": "ةمزلاتمةقئاضلاةيسفنتلاة داحلا",
    "DE": "akutes Atemnotsyndrom des Erwachsenen, (syn.) ARDS",
    "ES": "síndrome de dificultad respiratoria aguda, (syn.) SDRA",
    "FR": "syndrome de détresse respiratoire aiguë, (syn.) SDRA",
    "JA": "急性呼吸窮迫症候群, (syn.) 急性呼吸促迫症候群, ARDS",
    "KO": "급성  호흡곤란  증후군, (syn.) ARDS",
    "PT": "síndrome do desconforto respiratório agudo, (syn.) SDRA",
    "RU": "острый респираторный дистресс-синдром, (syn.) ОРДС",
    "ZH": "急性呼吸窘迫综合征, (syn.) ARDS"
}
```


Figura 2.1: Documento *wipo.json* original


```
"Traducoes": {
    "AR": "ةمزلاتمةقئاضلاةيسفنتلاة داحلا",
    "DE": "akutes Atemnotsyndrom des Erwachsenen",
    "ES": "síndrome de dificultad respiratoria aguda",
    "FR": "syndrome de détresse respiratoire aiguë",
    "JA": "急性呼吸窮迫症候群",
    "KO": "급성  호흡곤란  증후군",
    "PT": "síndrome do desconforto respiratório agudo",
    "RU": "острый респираторный дистресс-синдром",
    "ZH": "急性呼吸窘迫综合征"
}
```


Figura 2.2: Documento *wipo_modificado.json* após alterações

Nos ficheiros provenientes do trabalho prático 1 foi ainda necessário remover informação manualmente, nomeadamente no ficheiro *wipo.json* original existiam traduções cujas chaves eram *strings* vazias e o seu valor era igualmente *strings* vazias. Como esta informação não é relevante e provavelmente provém de uma gralha na leitura do ficheiro original (um parágrafo em branco que foi lido como uma tradução), essas chaves foram removidas.

Seguindo o mesmo raciocínio foi removido do `json` dos conceitos de anatomia conceitos que não tinham categoria nem designação, ou seja, conceitos cuja informação nele era a sua própria designação. Estes conceitos foram removidos, uma vez que não continham informação de todo.

2.2 Glossário de termos técnicos provenientes do Centro Hospitalar de Lisboa Ocidental

Com o objetivo de ter um glossário o mais abrangente possível e com um maior número de conceitos que o obtido no primeiro trabalho prático, recorreu-se a um glossário de uma entidade de saúde para extrair de lá informação. Foi escolhido este *site*, uma vez que o glossário lá presente abrange uma grande gama de áreas da saúde (doenças, material médico ou conceitos médicos) e a entidade responsável por ele é certificada na área da saúde [1].

Deste modo, para retirar informação do *site* foi necessário recorrer à biblioteca *Beatiful Soup* que é capaz de extrair informação de documentos `html` e `xml`. Assim sendo, cada conceito encontrava-se numa *div* diferente e portanto a extração da informação decorreu como representado na figura 2.3 [2].

```
def extrai_doencas(url):
    result = requests.get(url, headers=headers)
    if result.status_code != 200:
        print(f"Failed to retrieve page: {url} with status code {result.status_code}")
        return {}

    html = result.text
    soup = BeautifulSoup(html, "html.parser")

    rows = soup.find_all("tr", class_=["row0", "row1"])

    dicionario_doencas = {}

    for row in rows:
        a_tag = row.find("a")
        p_tag = row.find("p")

        if a_tag and p_tag:
            desig = a_tag.text.strip()
            desig = desig.lower()
            desig = re.sub(r'', ' ', desig) #Remover um caracter estranho
            descri = p_tag.get_text(strip=True)
            descri = descri.lower()
            descri = re.sub(r'', ' ', descri)

            dicionario_doencas[desig] = descri

    if not dicionario_doencas:
        print(f"No diseases found on page: {url}")

    return dicionario_doencas
```

Figura 2.3: Excerto do código presente no *script glossario_site.py* para extrair informação do *site*

Como se pode inferir a partir da figura 2.3, os conceitos encontravam-se em *divs* cuja *class* intercalava entre "*row0*" e "*row1*". Dentro da classe, a designação encontrava-se dentro de uma *tag* do tipo "a" e a descrição numa *tag* do

tipo "p". Esta informação foi recolhida e inserida num ficheiro denominado por *doencas_site.json* para posteriormente ser concatenado com os outros *datasets*.

2.3 Concatenação dos *datasets* (*juncao.py*)

Com o intuito de obter um único *dataset* com as informações de todos os anteriores, foi necessário realizar a concatenação dos mesmos. Os *datasets* concatenados foram:

- *conceitos.json*: Proveniente do documento *glossario_ministerio_saude.pdf* ;
- *termos_glossario.json*: Proveniente do documento *Glossario de Termos Medicos Tecnicos e Populares.pdf* ;
- *termos_anatomia.json*: Proveniente do documento *anatomia geral.pdf* ;
- *wipo_modificado.json*: Proveniente do documento *WIPOPearl_COVID-19_Glossary.pdf*
- *doencas_site.json*: Proveniente do *site* do Centro Hospitalar de Lisboa Ocidental [1];

De modo a não haver chaves repetidas, isto é, que não houvesse mais do que um conceito com o mesmo nome no *dataset*, fez-se uma comparação de modo a que sempre que um conceito fosse adicionado ao *dataset* comparasse o seu nome com os nele já existentes. Se o conceito já existisse a informação era adicionada ao mesmo, com outra chave para a descrição, permitindo um mesmo conceito ter várias descrições diferentes. Se não existisse era adicionada uma chave nova com as informações do conceito. Na figura 2.4 é possível observar o código para este efeito, enquanto que na figura 2.5 é visível um conceito do *dataset* com várias descrições.

```python
for termo, informacao in termos.items():
    if termo.lower() in conceitos:
        # Se o termo já existir em conceitos.json, adicione uma nova chave "Descricao 2" com a nova informação
        glossario[termo]["Descricao 2"] = informacao
    else:
        # Se o termo não existir em conceitos.json, adicione-o ao dicionário conceitos com todas as suas informações
        glossario[termo] = {"Categoria": "Sem Categoria","Descricao": informacao,}

for termo, informacao in anatomia.items():
    if termo.lower() in conceitos:
        # Se o termo já existir em conceitos.json, adicione uma nova chave "Descricao Anatomia" com a nova informação
        glossario[termo]["Descricao Anatomica"] = informacao
    else:
        # Se o termo não existir em conceitos.json, adicione-o ao dicionário conceitos com todas as suas informações
        glossario[termo] = {"Categoria": "Sem Categoria","Descricao": informacao,}

for termo, informacao in doencas.items():
    if termo in conceitos:
        # Se o termo já existir em conceitos.json, adicione uma nova chave "Descricao SNS" com a nova informação
        glossario[termo]["Descricao SNS"] = informacao
    else:
        # Se o termo não existir em conceitos.json, adicione-o ao dicionário conceitos com todas as suas informações
        glossario[termo] = {"Categoria": "Sem Categoria","Descricao": informacao,}
```

Figura 2.4: Excerto do código responsável pela concatenação dos *datasets* e verificação de conceitos repetidos


```
"eczema": {
    "Categoria": "doenças",
    "Descricao": "é uma doença infl amatória da pele caracteri-zada por eritema, edema, vesículas, escomas, crostas e liquenificação. a
    "Descricao 2": " dermatite; inflamação da pele ",
    "Descricao SNS": "doença cutânea manifestada por um processo inflamatório, superficial, com características e causas diversas, mu
```


Figura 2.5: Conceito no ficheiro *glossario_geral.json* com várias descrições

Por fim, para concluir a concatenação dos *datasets* todos faltou concatenar o ficheiro *wipo_modificado.json*. A concatenação deste ficheiro foi diferente dos anteriores porque a comparação da chave não foi direta, dado que foi comparada a tradução em português dentro do *wipo_modificado.json* com o nome dos conceitos já presentes.

Se o conceito tiver exatamente o mesmo nome que a tradução em português presente no *wipo_modificado.json* a informação das traduções é acrescentada a este conceito conforme a figura 2.6.


```
"farmacoepidemiologia": {
    "Categoria": "medicamentos, vacinas e insumos",
    "Descricao": "aplicação do método e raciocínio epidemio-lógico no estudo dos efeitos, benéficos e adversos, e do uso de medicamentos
    "Traducoes": {
        "AR": "ةلديصةيلئابو",
        "DE": "Pharmakoepidemiologie",
        "ES": "farmacoepidemiología",
        "FR": "pharmaco-épidémiologie",
        "JA": "薬剤疫学",
        "KO": "약물역학",
        "PT": "farmacoepidemiologia",
        "RU": "фармакоэпидемиология",
        "ZH": "药物流行病学"
    }
```


Figura 2.6: Exemplo de um conceito com tradução

Contudo, de forma a que haja traduções para um conceito, mesmo que o nome não seja exatamente igual, foi usado um modelo do *Hugging Face* para verificar a similaridade entre palavras. O modelo usado denomina-se por *BioBERT* e é um modelo pré-treinado em textos da área biomédica e que pode ser usado para tarefas como a similaridade de textos, classificação de textos, entre outras aplicações na área de processamento de linguagem natural [3].

Deste modo, se houver palavras que entre elas tenham uma similaridade acima de 95% a tradução também é adicionada ao conceito. Para efetuar a similaridade foi feita uma função presente na figura 2.7 e o código na figura 2.8.

```python
# Carrega o modelo e o tokenizer do BioBERT
model_name = "dmis-lab/biobert-base-cased-v1.1"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

# Função para calcular a similaridade entre duas palavras usando BioBERT
def calcular_similaridade(frase1, frase2, tokenizer, model):
    inputs1 = tokenizer(frase1, return_tensors='pt')
    inputs2 = tokenizer(frase2, return_tensors='pt')

    with torch.no_grad(): #Desativa o cálculo de gradientes pois é feita apenas inferência e economiza memória e torna o processo mais rápido
        outputs1 = model(**inputs1) #Passa os tokens da primeira frase pelo modelo para obter as saídas
        outputs2 = model(**inputs2)

    embeddings1 = outputs1.last_hidden_state.mean(dim=1) #Calcula a média das representações da última camada oculta para todos os tokens da primeira frase
    embeddings2 = outputs2.last_hidden_state.mean(dim=1)

    cos = torch.nn.CosineSimilarity(dim=1) #Define a função de similaridade coseno.
    similaridade = cos(embeddings1, embeddings2).item() #Retorna a similaridade calculada entre as duas frases

    return similaridade
```

Figura 2.7: Função que calcula a similaridade entre dois conceitos

```python
# Junta de acordo com a tradução em portugues no wipo. Verifica se a tradução é igual ao nome dos conceitos ou se são de termos parecidos (95% de semelhança)
for termo, informacao in traducoes.items():
    traducao_pt = informacao["Traducoes"]["PT"].lower()
    traducao_total = informacao["Traducoes"]

    # Verificar se a tradução PT está nos conceitos
    if traducao_pt in conceitos:
        glossario[traducao_pt] = conceitos[traducao_pt]
        glossario[traducao_pt]["Traducoes"] = traducao_total
    else:

        # Verificar similaridade com as chaves em conceitos
        for conceito, dados_conceito in conceitos.items():
            conceito_lower = conceito.lower()

            if (len(conceito_lower.split()) == 1) and (len(traducao_pt.split()) == 1): # compara conceitos cujo nome só tenha 1 palavra
                similaridade = calcular_similaridade(traducao_pt, conceito_lower, tokenizer, model)

                if similaridade > 0.95: # acima de 95% de similariedade
                    glossario[conceito_lower] = dados_conceito
                    glossario[conceito_lower]["Traducoes Relacionadas"] = traducao_total
        else:
            pass
```

Figura 2.8: Concatenação do ficheiro *wipo_modificado* ao *glossario_geral.json*

Por último pode-se observar na figura 2.9 um exemplo de uma tradução adicionada a um conceito cujo nome não é idêntico ao da tradução mas é similar.

```
"retenção": {
    "Categoria": "Sem Categoria",
    "Descricao": "    conservação,   dentro   do   corpo,   de   materiais   ou   líquidos   que   se   excretamnormalmente ",
    "Traducoes Relacionadas": {
        "AR": "ءاوتحا",
        "DE": "Eindämmung",
        "ES": "contención",
        "FR": "endiguement",
        "JA": "封じ込め",
        "KO": "전파차단",
        "PT": "contenção",
        "RU": "сдерживание",
        "ZH": "围堵"
    }
```


Figura 2.9: Exemplo de um conceito com uma traduções relacionadas

3. Plataforma Web

3.1 Funcionalidades da Plataforma

De forma a que fosse possível interagir com os conceitos presentes no glossário, foi necessário implementar uma aplicação *web*. Esta aplicação foi desenvolvida através da ferramenta *Flask*, uma vez que se trata de uma ferramenta de aprendizagem simples e é também uma ferramenta com uma vasta gama de utilidades [4].

Assim sendo, a aplicação *web* contém múltiplas páginas, ou rotas, cada uma representado uma diferente funcionalidade.

3.1.1 *Home Page*

Esta página é a página que introduz o utilizador à aplicação, informando-o do propósito da aplicação e demonstra-lhe também as diferentes funcionalidades existentes. Na figura 3.1 está demonstrada a página, enquanto que na figura 3.2 está um excerto do código que indica o início do desenvolvimento da aplicação, como os *imports* de bibliotecas importantes, a leitura do `json` correspondente ao glossário e demonstra também que é instanciada a classe correspondente ao *Flask*, isto é necessário para que o Flask saiba onde procurar recursos como modelos e ficheiros estáticos [5].

Na figura 3.2 é também demonstrada a função que irá definir o conteúdo da página inicial.

Figura 3.1: *Home Page* da aplicação

```python
import shutil
from flask import Flask, render_template, request, redirect, url_for
import json
import os
from collections import defaultdict

app = Flask(__name__)

file = "C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.json"
with open(file, 'r', encoding='utf-8') as file:
    doencas = json.load(file)

def guardar_conceitos(doencas, filename):
    with open(filename, 'w', encoding='utf-8') as file:
        json.dump(doencas, file, ensure_ascii=False, indent=4)

def carregar_conceitos(file):
    with open(file, 'r', encoding='utf-8') as file:
        doencas = json.load(file)
    return doencas

@app.route("/")
def home():
    tamanho = len(doencas)
    return render_template("home.html", tamanho = tamanho)
```

Figura 3.2: Código que inicia a aplicação e função que define a *Home Page*

Note-se que cada função da aplicação manda mensagens para um documento `HTML` que posteriormente serão mostradas ao utilizador, contudo a visualização destes documentos será estudada numa próxima secção.

3.1.2 Página de Conceitos

Nesta página são listados todos os conceitos que existem no glossário, sendo que aparece o seu nome e é possível clicar sobre o conceito para aceder às suas

informações. No topo da página existe um índice de letras de forma a percorrer a página de uma forma mais interativa e fácil para o utilizador, sendo que ao clicar sobre a letra ele é encaminhado para a secção da página onde as palavras começadas por essa letra se encontram.

Outra funcionalidade ainda existente nesta página é a capacidade de procurar um termo pelo seu nome. Esta página encontra-se mostrada na figura 3.3 e o código que é responsável por listar os conceitos na figura 3.4.

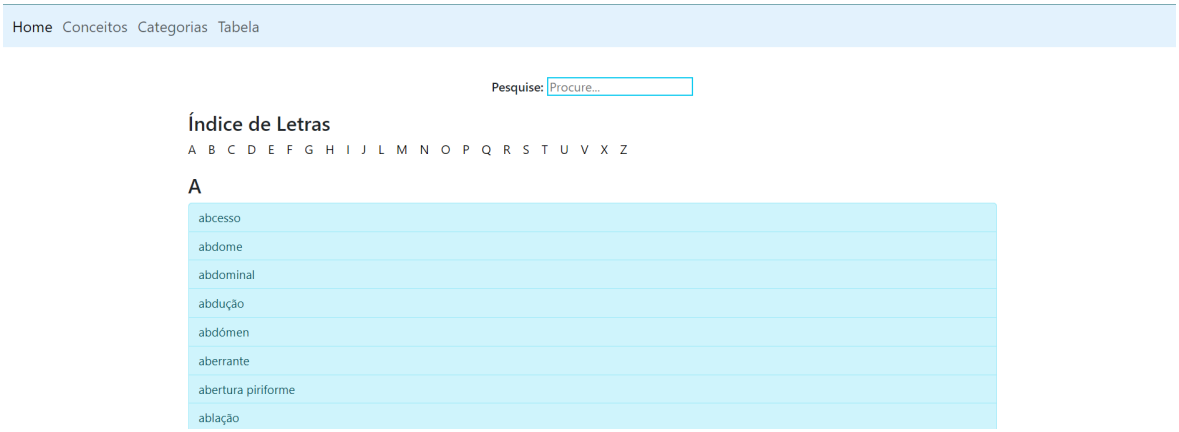

Figura 3.3: Página onde os conceitos se encontram listados

```python
@app.route("/doencas")
def listar_doencas():
    #Sempre que entra na página carrega o ficheiro para estar atualizado
    doencas = carregar_conceitos("C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.json"
    # Agrupar as doenças por letra
    doencas_por_letra = defaultdict(list)
    for doenca in doencas:
        primeira_letra = doenca[0].upper()

        # Junta as letras com acento à letra normal
        if primeira_letra == "Á":
            doencas_por_letra["A"].append(doenca)
        elif primeira_letra == "Â":
            doencas_por_letra["A"].append(doenca)
        elif primeira_letra == "É":
            doencas_por_letra["E"].append(doenca)
        elif primeira_letra == "Í":
            doencas_por_letra["I"].append(doenca)
        elif primeira_letra == "Ó":
            doencas_por_letra["O"].append(doenca)
        elif primeira_letra == "Ú":
            doencas_por_letra["U"].append(doenca)
        else:
            doencas_por_letra[primeira_letra].append(doenca)
    return render_template("doencas.html", doencas_por_letra=doencas_por_letra)
```

Figura 3.4: Código que lista os conceitos pela letra em que começa

O código que permite efetuar a pesquisa foi realizado em *javascript* e encontra-se na figura 3.5. Este código permite fazer a pesquisa de conceitos dentro da classe "*list-group-item*" que é a classe onde se encontram listados os conceitos e

permite também ocular as letras na página que não têm conceitos quando ocorre uma pesquisa.

```
// No método de pesquisa esta função oculete aquelas letras que não têm palavras
$(document).ready(function(){
    $("#myInput").on("keyup", function() {
        var value = $(this).val().toLowerCase();
        $(".list-group-item").filter(function() {
            $(this).toggle($(this).text().toLowerCase().indexOf(value) > -1);
        });

        $(".letra-section").each(function() {
            var section = $(this);
            var hasVisibleItems = section.find(".list-group-item:visible").length > 0;
            section.toggle(hasVisibleItems);
        });
    });
});
```

Figura 3.5: Código que é responsável por fazer a pesquisa dos conceitos

3.1.3 Página com a informação dos conceitos

Ao carregar sobre um conceito na página anterior o utilizador é redirecionado para a página do conceito carregado. Nesta página encontram-se mostradas ao utilizador as informações que o conceito tem, ocultando os campos para os quais o mesmo não tem informação disponível. Esta página permite ainda o utilizador retornar à página anterior, editar um conceito ou eliminá-lo. A página de um conceito, que serve para exemplo, encontra-se na figura 3.6 e a função responsável por mostrar as informações do conceito na figura 3.7.

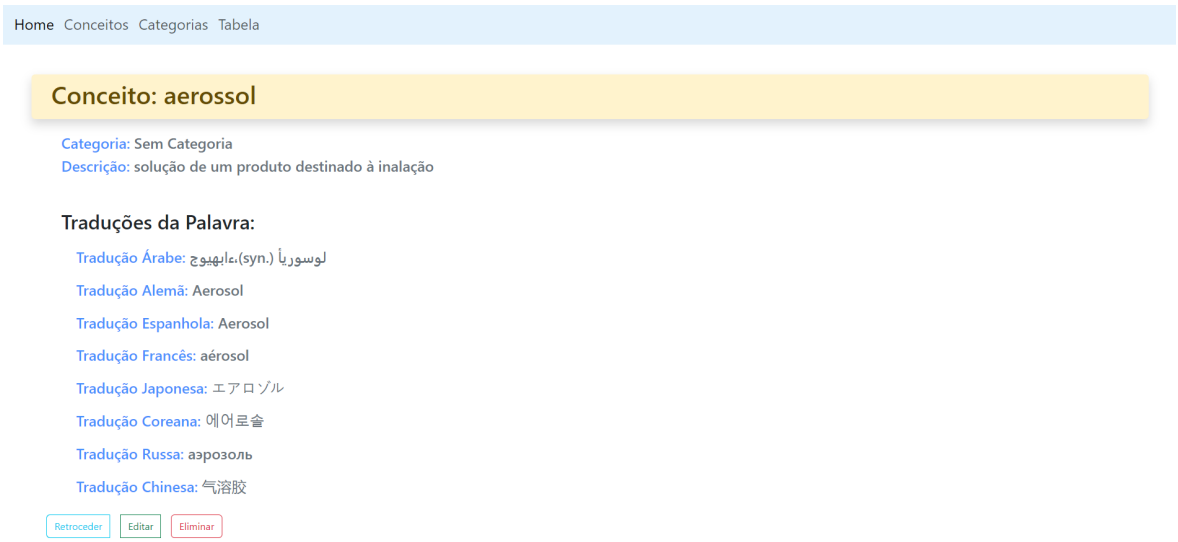

Figura 3.6: Página para visualizar as informações de um conceito

```python
@app.route("/doenca/<designacao>")
def consultar_doencas(designacao):
    desig = doencas[designacao]
    categoria = desig["Categoria"]
    descricao = desig["Descricao"]

    descricao_2 = desig.get("Descricao 2")  # O get permite que não dê erro caso a chave não exista
    descricao_sns = desig.get("Descricao SNS")
    descricao_utilizador = desig.get("Descricao Utilizador")
    traducoes = desig.get("Traducoes", {})
    traducoes_rel = desig.get("Traducoes Relacionadas", {})


    traducoes_ar = traducoes.get("AR")
    traducoes_de = traducoes.get("DE")
    traducoes_es = traducoes.get("ES")
    traducoes_fr = traducoes.get("FR")
    traducoes_ja = traducoes.get("JA")
    traducoes_ko = traducoes.get("KO")
    traducoes_pt = traducoes.get("PT")
    traducoes_ru = traducoes.get("RU")
    traducoes_zh = traducoes.get("ZH")

    traducoes_rel_ar = traducoes_rel.get("AR")
    traducoes_rel_de = traducoes_rel.get("DE")
    traducoes_rel_es = traducoes_rel.get("ES")
    traducoes_rel_fr = traducoes_rel.get("FR")
    traducoes_rel_ja = traducoes_rel.get("JA")
    traducoes_rel_ko = traducoes_rel.get("KO")
    traducoes_rel_pt = traducoes_rel.get("PT")
    traducoes_rel_ru = traducoes_rel.get("RU")
    traducoes_rel_zh = traducoes_rel.get("ZH")
```

Figura 3.7: Código que é responsável por mostrar a informação de um conceito

Por outro lado, nesta página há ainda a opção de eliminar um conceito, sendo para que esta eliminação ocorra conforme o esperado e não haja falha durante o processo, antes de ocorrer a eliminação é feito um *backup* de segurança do ficheiro e só então ocorre a eliminação do mesmo no ficheiro principal. Este processo encontra-se na figura 3.8.

```python
@app.route("/doenca/<designacao>/eliminar", methods=["POST"])
def eliminar_conceito(designacao):
    caminho_json = "C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.json"
    doencas = carregar_conceitos(caminho_json)

    # Verifica se o conceito com a designação fornecida existe
    if designacao in doencas:
        # Realiza um backup do arquivo antes de fazer alterações
        shutil.copyfile(caminho_json, "C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_backup.json")

        del doencas[designacao]

        # Guarda as alterações no arquivo JSON
        guardar_conceitos(doencas, caminho_json)

        # Redireciona para a página de listagem de doenças após a eliminação
        return redirect(url_for("listar_doencas"))

    # Caso o conceito não exista, retorna uma mensagem de erro
    return "O conceito não existe."
```

Figura 3.8: Código que é responsável eliminar um conceito

3.1.4 Página para editar um Conceito

De forma a ser possível editar um conceito foi criada uma página dedicada a tal para cada conceito. Assim sendo, esta página permite editar algumas informações

de um conceito e adicionar outras. As operações permitidas são:

- Alterar a categoria do conceito, mas apenas para categorias já existentes no glossário;
- Acrescentar traduções caso não existam ou alterar as já existentes;
- Acrescentar uma descrição do utilizador ao conceito, servindo como uma nota do utilizador.

Deste modo, não é permitido ao utilizador alterar o nome do conceito ou as descrições nele já existentes. A página de edição de um conceito encontra-se na figura 3.9. Como se pode observar todos os campos existentes do conceito estão demonstrados e bloqueados, contudo no fundo da página há a possibilidade de alterar alguns deles.

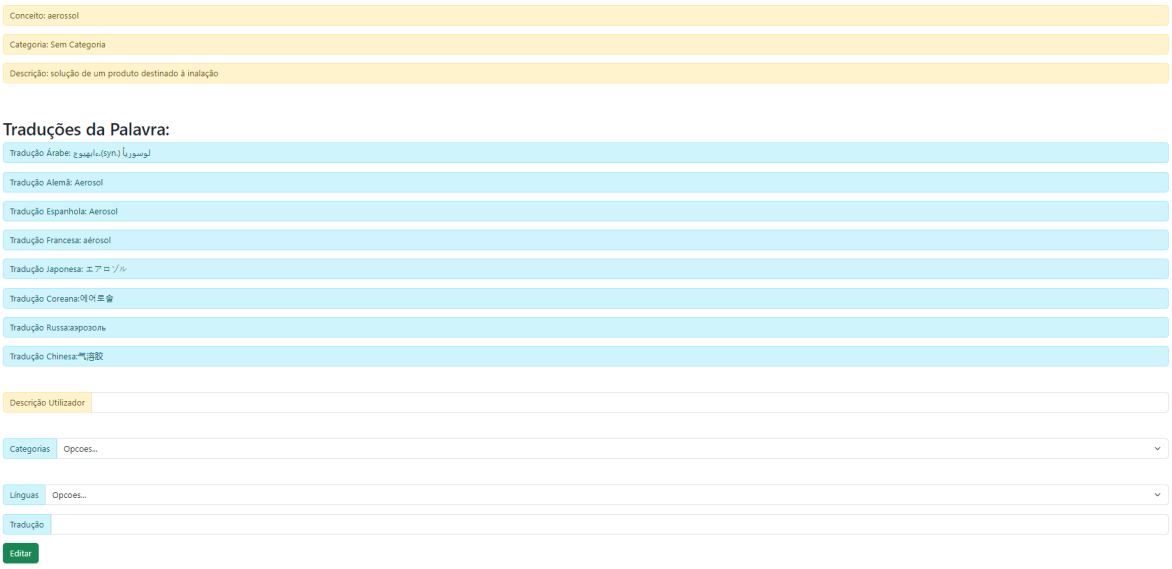

Figura 3.9: Página onde ocorre a edição de um conceito

De forma a que fosse possível exibir os dados atualizados do conceito aquando a página de edição é carregada pela primeira vez foi necessário usar o método *GET*, enquanto que para processar as alterações submetidas pelo utilizador e atualizar os dados no servidor foi usado um método do tipo *POST*. Estes dois métodos encontram-se nas figuras 3.10 e 3.11.

```python
@app.route("/doenca/<designacao>/editar", methods=["GET", "POST"])
def editar_conceitos(designacao):
    global doencas,filename

    if request.method == "GET": # É importante existir para quando a página é aberta pela primeira vez aparecerem os dados expostos
        desig = doencas[designacao]
        categoria = desig["Categoria"]
        descricao = desig["Descricao"]
        descricao_2 = desig.get("Descricao 2")
        descricao_sns = desig.get("Descricao SNS")
        descricao_utilizador = desig.get("Descricao Utilizador")
        traducoes = desig.get("Traducoes", {})
        traducoes_rel = desig.get("Traducoes Relacionadas", {})

        traducoes_ar = traducoes.get("AR")
        traducoes_de = traducoes.get("DE")
        traducoes_es = traducoes.get("ES")
        traducoes_fr = traducoes.get("FR")
        traducoes_ja = traducoes.get("JA")
        traducoes_ko = traducoes.get("KO")
        traducoes_pt = traducoes.get("PT")
        traducoes_ru = traducoes.get("RU")
        traducoes_zh = traducoes.get("ZH")

        traducoes_rel_ar = traducoes_rel.get("AR")
        traducoes_rel_de = traducoes_rel.get("DE")
        traducoes_rel_es = traducoes_rel.get("ES")
        traducoes_rel_fr = traducoes_rel.get("FR")
        traducoes_rel_ja = traducoes_rel.get("JA")
        traducoes_rel_ko = traducoes_rel.get("KO")
        traducoes_rel_pt = traducoes_rel.get("PT")
        traducoes_rel_ru = traducoes_rel.get("RU")
        traducoes_rel_zh = traducoes_rel.get("ZH")

        lista_categorias = obter_categorias("C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.json")
```

Figura 3.10: Código do método *GET* para exibir informações na página de editar

```python
def editar_conceitos(designacao):

    elif request.method == "POST": # Para alterar os dados
        categoria = request.form["categoria"]
        descricao_utilizador = request.form["descricao_utilizador"]
        nova_lingua = request.form["nova_lingua"]
        lingua_descricao = request.form["lingua_descricao"]

        if nova_lingua!="vazio" and lingua_descricao!="": #Para acrescentar uma lingua que não exista ou atualizar
            if "Traducoes" not in doencas[designacao] and "Traducoes Relacionadas" not in doencas[designacao]:  # Não existe nenhuma tradução, adiciona de raiz
                doencas[designacao]["Traducoes"] = {nova_lingua: lingua_descricao}
            elif "Traducoes" in doencas[designacao]: # Existem traducoes entao muda a chave relacionada com elas
                doencas[designacao]["Traducoes"][nova_lingua] = lingua_descricao
            elif "Traducoes Relacionadas" in doencas[designacao]: # Existem traducoes relacionadas entao muda a chave relacionada com elas. Se não houvesse
            diferenciação dava erro ou não estaria a ser atribuido corretamente
                doencas[designacao]["Traducoes Relacionadas"][nova_lingua] = lingua_descricao

        if categoria!="vazio":
            doencas[designacao]["Categoria"] = categoria

        if descricao_utilizador:
            doencas[designacao]["Descricao Utilizador"] = descricao_utilizador

        guardar_conceitos(doencas,"C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.json")

        if "glossario_backup.json" in os.listdir("C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json"): #guarda também no backup se ele
        existir, o backup está sempre atualizado
            guardar_conceitos(doencas, "C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_backup.json")

        return redirect(url_for("consultar_doencas", designacao=designacao))
```

Figura 3.11: Código do método *POST* para efetuar alterações nos conceitos

3.1.5 Página para listar as Categorias

Esta página tem o objetivo de mostrar ao utilizador todas as categorias presentes no glossário e permite-o visualizar os conceitos de acordo com a categoria que pretende aceder, garantindo-lhe uma forma de visualizar os conceitos numa outra perspetiva. A página encontra-se demonstrada na figura 3.12 e o código para encontrar as categorias no glossário e listá-las na figura 3.13.

Figura 3.12: Página com todas as categorias listadas

```python
def obter_categorias(caminho):
    doencas = carregar_conceitos(caminho)
    lista_categorias = []
    for conceito, info in doencas.items():
        categoria = info["Categoria"]
        categoria = categoria.capitalize()  # torna apenas a primeira letra em maiúscula
        if categoria not in lista_categorias:
            lista_categorias.append(categoria)
    return sorted(lista_categorias)

@app.route("/categorias")
def listar_categorias():
    caminho_json = "C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.json"
    lista_ordenada = obter_categorias(caminho_json)
    return render_template("categorias.html", lista_ordenada=lista_ordenada)
```

Figura 3.13: Código que permite encontrar e listar todas as categorias

3.1.6 Página para consultar cada Categoria

Após o utilizador carregar sobre uma categoria ele é redirecionado para uma página que, à semelhança da página que lista os conceitos, esta também lista os conceito, contudo lista apenas os pertencentes à categoria em questão.

Na figura 3.14 está um exemplo da página de uma categoria, enquanto que na figura 3.15 encontra-se o código para fazer esta operação.

Figura 3.14: Excerto da página com os conceitos pertencentes à categoria "Acidentes e violência"

```python
@app.route("/categorias/<categoria>")
def consultar_categoria(categoria):
    # Carrega o json para tar atualizado
    doencas = carregar_conceitos("C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.json")

    # Filtra os conceitos pela categoria
    conceitos_filtrados = {conceito: detalhes for conceito, detalhes in doencas.items() if detalhes['Categoria'].upper() == categoria.upper()}

    return render_template("consultar_categoria.html", categoria=categoria, conceitos=conceitos_filtrados)
```

Figura 3.15: Código que lista os conceitos pertencentes a uma categoria

3.1.7 Página da Tabela de Conceitos

Por fim, a última página existente na aplicação *web* é correspondente a uma tabela com as 3 informações mais relevantes de cada conceito: a sua designação, categoria e descrição principal. Nesta página existe também um método de pesquisa, contudo, ao contrário do método de pesquisa na página onde os conceitos estão listados, este permite fazer uma pesquisa pelo nome do conceito ou por palavras existentes na sua categoria ou descrição. A página com a tabela encontra-se na figura 3.16.

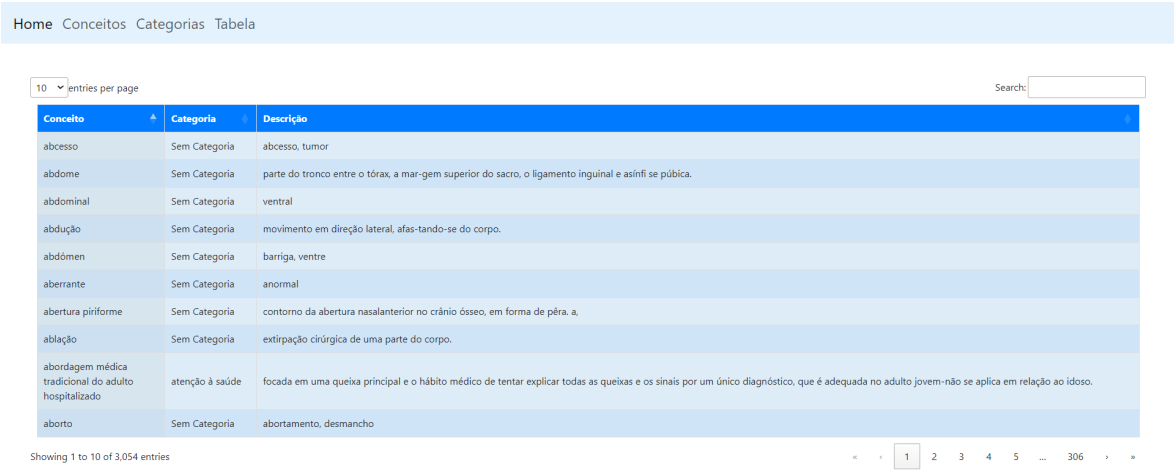

Figura 3.16: Página com a tabela de conceitos

Para que esta tabela tivesse a forma pretendida e o método de procura, foi necessário implementar uma função em *Python* visível na figura 3.17 e outra função em *JavaScript* visível na figura 3.18. Esta função em *JavaScript* utiliza a biblioteca *jQuery* junto com o *plugin DataTables* para inicializar e configurar uma tabela `HTML` de forma interativa. Esta tabela, como já referido, permite pesquisa instantânea, paginação e ordenação.

```python
# Faz a tabela e usa js
@app.route("/tabela")
def table():
    doencas = carregar_conceitos("C:/Users/ruben/OneDrive - Universidade do Minho/4ºAno/2ºSemestre/PLN/Trabalho2/json/glossario_geral.
    json")
    return render_template("tabela.html", doencas=doencas)
```

Figura 3.17: Código em *Python* para implementar a tabela

```javascript
// Função para fazer a tabela
$(document).ready( function () {
    $('#myTable').DataTable();
} );
```

Figura 3.18: Código em *JavaScript* para implementar a tabela

3.2 *Jinja*

Jinja é um poderoso mecanismo de *templates* para *Python*, amplamente utilizado no desenvolvimento de aplicações *web* com *frameworks* como o *Flask*. Ele permite a separação clara entre a lógica da aplicação e a apresentação dos dados,

facilitando a criação de páginas *web* dinâmicas e mantendo o código organizado e modular.

Assim sendo, o primeiro documento HTML criado foi o *parent.html* que atua como um *template* base ou *layout* principal na aplicação *web*. Este ficheiro define a estrutura HTML comum que outras páginas herdarão e na qual se encaixarão. Este tipo de estrutura facilita a manutenção e a consistência da aplicação, pois permite que múltiplas páginas reutilizem o mesmo *layout* base. No cabeçalho deste documento são especificadas a codificação de caracteres e os *links* para folhas de estilo externas do *Bootstrap* e *DataTables*, que são *frameworks CSS* para mudar o *design* da página e tabelas de dados, respetivamente.

Outra vantagem é o facto de permitir iterar sobre coleções de dados diretamente no HTML como a criação de ciclos *for* dentro de um documento HTML. Um exemplo da utilização desta funcionalidade é por exemplo na página referente à listagem dos conceitos. Este exemplo encontra-se visível na figura 3.21.

```
<div class="container">
    <!-- Acrescenta barra de pesquisa -->
    <h5 class="mt-5 text-center"> Pesquise: <input id="myInput" class="border border-info border-2" type="text"
    placeholder="Procure..."> </h5>

    <!-- Acrescenta um índice de letras que podem ser clicadas para direcionar à letra -->
    <div class="mt-4">
        <h2>Índice de Letras</h2>
        <div>
            {% for letra in doencas_por_letra.keys() %}
                <a class="letra-anchor" href="#{{ letra }}">{{ letra }}</a>
            {% endfor %}
        </div>
    </div>


    <!-- Expõe os conceitos -->
    {% for letra, doencas_letra in doencas_por_letra.items() %}
    <div class="mt-4 letra-section" id="{{ letra }}">
        <h2>{{ letra }}</h2>
        <div class="list-group">
            {% for doenca in doencas_letra %}
                <a href="/doenca/{{ doenca }}" class="list-group-item list-group-item-action list-group-item-info">{{ doenca }}</a>
            {% endfor %}
        </div>
    </div>
    {% endfor %}
</div>
```

Figura 3.19: Exemplo da utilização de ciclos *for* em *Jinja*

No *Jinja* é possível ainda utilizar lógica condicional diretamente nos *templates*. Este típo de lógica permite ocultar ou exibir informação dependendo da condição. No caso do exemplo da figura 3.20, referente à página em que a informação de

cada conceito é mostrada, esta lógica condicional é importante para verificar os campos de informação existentes no conceito e se ele tiver informação referente à chave em questão, a informação é mostrada, caso contrário é ocultada já que não existe.

```
{% if descricao_2 %}
    <h4> <span class="text-primary text-opacity-75"> Outra Descrição:</span> <span class="text-secondary">{{ descricao_2 }}</span> </h4>
{% endif %}
{% if descricao_sns %}
    <h4> <span class="text-primary text-opacity-75"> Descrição SNS:</span> <span class="text-secondary">{{ descricao_sns }}</span> </h4>
{% endif %}
{% if descricao_utilizador %}
    <h4> <span class="text-primary text-opacity-75"> Descrição Utilizador:</span> <span class="text-secondary">{{ descricao_utilizador }}</span> </h4>
{% endif %}

{% if traducoes %}
    <h2 class="pt-5"> Traduções da Palavra: </h2>
    {% if traducoes_ar %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Árabe:</span> <span class="text-secondary">{{ traducoes_ar }}</span> </h4>
    {% endif %}
    {% if traducoes_de %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Alemã:</span> <span class="text-secondary">{{ traducoes_de }}</span> </h4>
    {% endif %}
    {% if traducoes_es %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Espanhola:</span> <span class="text-secondary">{{ traducoes_es }}</span> </h
    {% endif %}
    {% if traducoes_fr %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Francês:</span> <span class="text-secondary">{{ traducoes_fr }}</span> </h4>
    {% endif %}
    {% if traducoes_ja %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Japonesa:</span> <span class="text-secondary">{{ traducoes_ja }}</span> </h4
    {% endif %}
    {% if traducoes_ko %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Coreana:</span> <span class="text-secondary">{{ traducoes_ko }}</span> </h4>
    {% endif %}
    {% if traducoes_ru %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Russa:</span> <span class="text-secondary">{{ traducoes_ru }}</span> </h4>
    {% endif %}
    {% if traducoes_zh %}
        <h4 class="ms-4 mt-4"> <span class="text-primary text-opacity-75"> Tradução Chinesa:</span> <span class="text-secondary">{{ traducoes_zh }}</span> </h4>
    {% endif %}
```

Figura 3.20: Exemplo da utilização de lógica condicional em *Jinja*

Para realizar alterações do *dataset* é necessário a utilização de *forms* que permitem também o utilizador interagir com o sistema, enviando-lhe dados ou alterando informações. No *Jinja*, podemos criar formulários de forma dinâmica, incorporando variáveis do *backend Python* diretamente no `HTML`.

No exemplo da figura 3.21, o formulário usado é para alterar ou adicionar informações ao glossário e como se pode constatar ele usa um método do tipo *post* dado que são enviados dados ao servidor para alterar o seu estado ou criar e atualizar informações. Relativamente à *action*, ela define o *endpoint* para o qual os dados do formulário serão enviados. O uso de `{{designacao}}` torna a ação dinâmica, direcionando para o *endpoint* específico de edição da doença com a designação fornecida.

```
<div class="container">
    <!-- Acrescenta barra de pesquisa -->
    <h5 class="mt-5 text-center"> Pesquise: <input id="myInput" class="border border-info border-2" type="text"
    placeholder="Procure..."> </h5>

    <!-- Acrescenta um índice de letras que podem ser clicadas para direcionar à letra -->
    <div class="mt-4">
        <h2>Índice de Letras</h2>
        <div>
            {% for letra in doencas_por_letra.keys() %}
                <a class="letra-anchor" href="#{{ letra }}">{{ letra }}</a>
            {% endfor %}
        </div>
    </div>


    <!-- Expõe os conceitos -->
    {% for letra, doencas_letra in doencas_por_letra.items() %}
    <div class="mt-4 letra-section" id="{{ letra }}">
        <h2>{{ letra }}</h2>
        <div class="list-group">
            {% for doenca in doencas_letra %}
                <a href="/doenca/{{ doenca }}" class="list-group-item list-group-item-action list-group-item-info">{{ doenca }}</a>
            {% endfor %}
        </div>
    </div>
    {% endfor %}
</div>
```

Figura 3.21: Exemplo da utilização de um *forms* em *Jinja*

Por fim de forma a dar alguma estética à aplicação *web* foi usada a *framework Bootstrap* como se pode ver no exemplo da figura 3.21 em que dentro das *tags* do tipo "*class*" são colocadas informações que estilizam o elemento. Outra forma de alterar a estética da página sem recorrer ao *Bootstrap* e que permite também ter um outro grau de liberdade é através do *CSS*. Na figura 3.22 pode-se observar código *CSS* para alterar a estética da página dentro das *tags* "*style*".

```
<a class="m-2 mt-4 ms-4 btn btn-block" href="categorias/{{categoria}}" role="button" style="
    background: #fce7b8;
    color: black;
    padding: 15px;
    border-radius: 5px;
    font-weight: 500;
    font-size: 17px;
    text-decoration: none;
    width: 70%;
">{{categoria}}</a>
```

Figura 3.22: Exemplo da utilização de código *CSS* em *Jinja*

4. Conclusão

O trabalho prático realizado permitiu continuar a extrair informação relevante na área médica, sendo desta vez através de um *site* e fazer o estudo do mesmo para conseguir extrair apenas o relevante. Foi também possível aplicar modelos de *Machine Learning* para averiguar a similaridade entre palavras dentro da área da biomédica.

Com este trabalho foi também possível verificar a flexibilidade e utilidade tanto do *Flask* como do *Jinja* e aplicá-los a ambos para fazer uma aplicação *web* que fosse dinâmica e permitisse manipular os dados de um glossário de termos médicos/biomédicos.

É possível então concluir que o objetivo proposto para o trabalho foi concluído com sucesso já que todas as metas foram atingidas, contudo destaca-se para um trabalho futuro a utilização de outras ferramentas que teriam sido uma grande adição ao glossário, nomeadamente utilizar uma API da *wikipédia* que ao encontrar uma página com o nome do conceito iria adicionar o seu link ao mesmo, permitindo a sua consulta.

Referências

[1] Glossário, Unidade Local de Saúde de Lisboa Ocidental https://www.chlo.min-saude.pt/index.php/component/seoglossary/1-glossario?start=0

[2] *Beautiful Soup Documentation*, https://beautiful-soup-4.readthedocs.io/en/latest/

[3] BioBert Model, https://huggingface.co/dmis-lab/biobert-v1.1/discussions?library=transformers

[4] TreinaWeb, https://www.treinaweb.com.br/blog/o-que-e-flask

[5] Flask Quickstart, https://flask.palletsprojects.com/en/3.0.x/quickstart/