**Introdução**

A escolha de um estabelecimento de ensino superior e o seu respetivo curso é uma tarefa que abrange todos os estudantes que ingressam num plano curricular. Atualmente, esta escolha é feita com base na informação proveniente do site da Direção Geral do Ensino Superior (DGES) que, por sua vez, apresenta a mesma de forma antiquada através de menus e tabelas.

O objetivo deste projeto é facilitar a escolha do estabelecimento e curso superior através de uma interface mais gráfica, com mais opções de pesquisa e com melhor apresentação de informação sobre os temas anteriormente referidos. Para isso, o projeto foi repartido em duas grandes partes:

- Um web scrawler contendo alguns scripts para popular as tabelas da base de dados

- Uma aplicação web para a consulta das tabelas anteriormente preenchidas pelo crawler.

Para o efeito, então, o workflow da operação começa no web crawler. Este irá percorrer páginas chave do site da DGES em busca de informação relativas ás instituições, cursos e a relação entre estes os dois – isto é, quais são os cursos que pertencem a uma determinada instituição. O algoritmo também irá percorrer o site <https://www.pordata.pt/> de modo a capturar informações sobre a qualidade de vida em cada cidade relevante e irá preencher as tabelas cidades e distrito com base no sítio web <http://pt.gpspostcode.com/codigo-postal/portugal/#111>

Depois da base de dados estar completamente preenchida, a nossa aplicação web irá, então, disponibilizar toda a informação com recurso ao API do Google Maps. Com o seu auxilio, pretendemos que o utilizador possa selecionar um conjunto das seguintes opções de pesquisa:

- Pesquisar por instituição/curso

- Pesquisar por nota mínima

- Pesquisar por distrito

- Pesquisar por cidade

- Pesquisar por qualidade de vida

- Pesquisar por poder de compra

- Pesquisar por ranking

Cada um dos resultados da pesquisa será indicado no mapa, bem como informações relevantes sobre a pesquisa. Ao clicar no resultado, o utilizador será levado para uma página onde poderá encontrar mais informações sobre a instituição/curso.

**Scrapper**

A framework de web crawling que escolhemos é a *scrapy* – uma framework para python. Utilizamos esta ferramenta para retirar informação de sítios web públicos e preencher as tabelas na base de dados relativas ás instituições com essa mesma informação. Isto é:

* Cidades
* Cursos
* Distritos
* Instituicoes
* Instituicoes\_has\_cursos
* Tipos\_curso
* Tipos\_ensino

**Scrapy – Arquitetura**

O Scrappy foi desenvolvido com a Twisted, uma framework de networking orientada por eventos. Deste modo, o Scrappy utiliza código assíncrono para suportar processamentos em paralelo, fazendo-o uma framework extremamente rápida e capaz

**Fluxo de dados**

**Diagram

Description automatically generated**

O fluxo de dados no Scrapy é controlado pelo motor de execução e funciona assim:

1. O motor recebe as solicitações iniciais para fazer *crawl* da *Spider* que, é o script que o utilizador da framework escreve para efetuar um scrape.
2. O motor agenda as solicitações no Agendador (Scheduler) e pede as próximas solicitações a serem rastreadas.
3. O Agendador retorna as próximas Solicitações ao Motor
4. O Motor envia as Solicitações para o Downloader, passando pelos Middlewares do Downloader
5. Assim que a página termina o download, o Downloader gera uma Resposta contendo o HTML e demais código proveniente da página em questão. Envia, depois, essa informação para o Motor passando pelos Middlewares do Downloader de novo
6. O Motor recebe a Resposta do Downloader e envia-a para a Spider, passando pelo middleware da Spider.
7. A Spider processa a Resposta e retorna itens retirados para o Motor, passando pelo Middleware da Spider
8. O Motor envia itens processados para as pipelines de item e, em seguida, envia solicitações processadas para o agendador e pede pelas próximas solicitações possíveis de rastreamento (novo url)
9. O processo repete-se da etapa 1 até que não haja mais solicitações do Agendador

**Componentes**

* Motor Scrapy: Responsável por controlar o fluxo de dados entre todos os componentes do sistema e desencadeia eventos quando certas ações ocorrem
* Agendador**:** Recebe solicitações do Motor e coloca-os numa fila para os processar quando o Motor o requisita.
* Downloader: Responsável por descarregar páginas web e encaminha-las para o Motor, que, por sua vez, as encaminha para a Spider
* Spider: Classes escritas por utilizadores da framework para analisar respostas e extrair dados
* Pipeline do item:Responsável pelo processamento dos itens depois de serem extraídos pela Spider. Tipicamente inclui tratamento da informação, validação e persistência.
* Middlewares do Downloader:Processam as solicitações entre do Motor para o Downloader e as respostas que passam do Downloader para o Motor
* Middlewares da Spider: Processam informação proveniente do Motor para a Spider ou vice-versa.

**XPath vs Seletor CSS**

O Scrapy, como todos as outras ferramentas de scrapping, utiliza ambos XPath e Seletores CSS.

O XPath é o caminho absoluto para o elemento HTML. É utilizado pelo browser para processar o HTML e o CSS da página. É relativamente mais versátil do que o seletor CSS. Contudo, dado que o seletor CSS é muito similar á biblioteca jQuery no que toda a selecionar elementos do domino (DOM), optamos por usar esta opção.

**Mais sobre pipelines**

As pipelines são constituídas por alguns métodos chave:

* open\_spider() : é chamado quando a pipeline é chamada. É o primeiro pedaço de código a correr
* process\_item() : define como é que o item é processado
* close\_spider() : é chamado quando a Spider acaba o seu trabalho

Neste projeto, tipicamente estes métodos são utilizados da seguinte maneira:

* open\_spider():

Manda um Log para o output da linha de comandos informando qual é a Spider que o está a chamar, bem como abrir a ligação á base de dados e definir o cursor para que se possa efetuar queries.

Text

Description automatically generated

* process\_item():

Efetuar o query com a informação proveniente do *yield* da Spider e retorna o item para ser apresentado no output da linha de comandos

Text

Description automatically generated

* end\_spider():

Envia um Log para o output da linha de comandos informando o encerramento da Spider e fecha a pipeline em si.

Text

Description automatically generated

**Spiders**

Como foi referido anteriormente, uma spider é um pedaço de código (tipicamente alojado numa classe) onde o utilizador da framework irá escrever os comandos para obter a informação do sítio web.

As Spiders no Scrapy derivam todas de um dos tipos de Spiders base. Nós escolhemos o *scrappy.Spider* já que é a mais abrangente e mais recente adicionada.

****

As classes que derivam desta classe base têm alguns atributos importantes que são fundamentais para o comportamento da Spider. São estas:

* start\_urls : uma lista de urls que pretendemos fazer download e explorar
* name : o nome pelo qual o scrapy sabe qual porção de código executar através da sua shell
* custom\_settings : para definir configurações especiais que, no nosso caso, é definir uma pipeline para cada Spider

Para além dos atributos, a classe base também tem alguns métodos importantes mas, apenas o método *parse()* é prescindente para a operação, já que é a primeira função a ser chamada. Este método recebe como argumento a resposta do Downloader e, através desta resposta, podemos percorrer os elementos da página, tal como se tivéssemos a selecionar um elemento em jQuery.

**Instituições**

Esta Spider serve para preencher a tabela instituições na base de dados.

No principio desta classe, encontramos o seu nome.



Para perceber como é que o nosso código consegue selecionar elementos HTML da página web, primeiro é preciso analisar a arquitetura HTML que compõe esta página.

Para isso, temos que ir á página do site da DGES relativo ás instituições. Nomeadamente, teremos que ir a <https://dges.gov.pt/guias/indest.asp>

Lá dá-mos conta que, no inicio da página, pode-se selecionar o tipo de ensino superior como critério de pesquisa.

Text

Description automatically generated

A lista é composta por elementos *<a></a>* que redirecionam o utilizador para a página que lista todas as instituições do tipo de ensino selecionar.

Como são só 5 links, podemos simplesmente iniciar a nossa Spider com a variável *start\_url* como uma lista de esses 5 links. É possível que, ao acabar um url, dá-mos outro á Spider para repetir o processo em tempo real. Mas como a variável de url reg não é sequencial, e como são só 5 urls, definimos implicitamente quais urls que esta Spider vai percorrer.

Text

Description automatically generated

Após isto, deparamo-nos com uma tabela de todas as instituições do tipo de ensino selecionado. Mas, após uma análise com o Google Chrome Dev Tools, vê-mos que esta tabela é apenas constituída por *divs* onde, o nome da instituição está num *div* com classe *box9*

Graphical user interface, text

Description automatically generated with medium confidence

Deste modo, apenas é necessário armazenar toda a informação contida dentro de cada elemento. Ao analisar melhor o elemento, vê-mos exatamente onde a informação reside

Graphical user interface, text, application

Description automatically generated

Então, no método *parse()* desta Spider, iniciamos o scrape

Text

Description automatically generated

O método response.css(‘.box9’) constitui um array de elementos. Dentro de cada um desse elementos, reside a informação que precisamos. Para mais, temos o tipo de código dado nos últimos dois dígitos do atual url.

A expressão yield serve para recebermos a informação na pipeline para ser devidamente tratada e armazenada. Temos que fazer yield num objeto dicionário obrigatoriamente.

De seguida, como temos definido configurações especiais nesta Spider e, assim, definimos o seu pipeline, podemos então definir a classe pipeline que está associada a esta Spider (InstituicoesPipeline)

**Text

Description automatically generated**

Na pipeline InstituicoesPiepeline, método process\_item encarrega-se de chamar o método store\_db, onde usa o cursor para enviar queries para a base de dados.

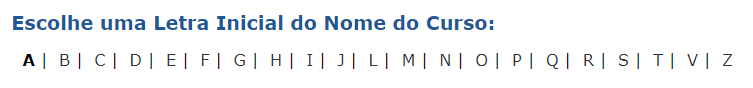
**Text

Description automatically generated**

**Cursos**

Os cursos podem ser pesquisados através da sua inicial através do seguinte link: <https://dges.gov.pt/guias/indcurso.asp?letra=A>.

Como podemos ver, no inicio da página, encontramos todos os links para o resto das letras



Contudo, já que a única variável é a letra no fim do url, que é atribuída á variável de url “letra”, podemos simplesmente trocar o ultimo caracter do link para a próxima letra alfabética. Assim, no inicio desta Spider encontramos uma função que retorna mesmo isso.

**Text

Description automatically generated**

Ao olhar para a estrura da página, vêmos que a informação relativa ao curso está em elementos div com a classe box10.

Graphical user interface, text, application

Description automatically generated

Então, da mesma maneira que a Spider anterior, temos o seguinte código:

Text

Description automatically generated

Como precisamos de alterar o url, isto é, alterar o contexto, temos que que, primeiro, alterar o url para que tenha a próxima letra incluída:



E, de seguida, efetuar a mudança de contexto:

Text

Description automatically generated with medium confidence

--- PIPELINE ---

**Distritos**

Infelizmente, o site da DGES não contém informação sobre os distritos de Portugal. Então, utilizamos este sítio web:

<http://pt.gpspostcode.com/codigo-postal/portugal/#111>

Lá, encontramos uma tabela com o nome e código de cada distrito.

A picture containing table

Description automatically generated

Assim, os atributos desta classe são os seguintes:

Text

Description automatically generated

Após analisar a estrutura HTML da página, podemos verificar que a informação que queremos está dentro de .table\_milieu tr

Text

Description automatically generated



Então, na Spider, podemos verificar que o método *parse()* contém o seguinte código:

Text

Description automatically generated

Já na pipeline, verificamos se a informação é válida (algumas linhas da coluna não têm informação pertinente) e, de seguida, efetuamos a query

Text

Description automatically generated

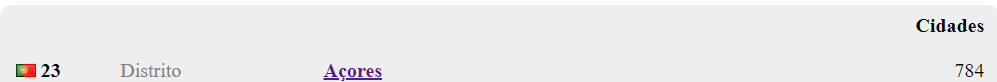
**Cidades**

Tal como os distritos, as cidades não estão disponíveis no sítio web da DGES, sendo que temos que usar o mesmo que utilizamos para os distritos. Deste modo, os atributos desta classe são os seguintes:

Text

Description automatically generated

Como já foi referido, a informação sobre os distritos está na tabela com classe CSS table\_milieu. Dentro desta tabela, podemos verificar que, ao clicar em cada distrito, o utilizador é redirecionado para uma página separada onde consta todas as cidades de cada distrito

****



Logo

Description automatically generated with low confidence

**Text

Description automatically generated**

Como se pode notar, existe algumas inconsistências na informação já que, por exemplo, o Corvo não é um município mas sim um ilha (para fins administrativos, uma cidade) e, se quisermos ver as “cidades” de por exemplo, Ponta Delgada, vê-mos que nos é apresentados apenas as freguesias de Ponta Delgada já que, no Município de Ponta Delgada, Ponta Delgada é a única cidade.

**A picture containing text

Description automatically generated**

Text

Description automatically generated

Assim, trataremos dos municípios como se fossem cidades, então, na nossa Spider primeiro necessitamos de ir para o url de cada um dos distritos no método parse()

Text

Description automatically generated

A linha assinalada faz com que a nova página seja tratada pelo método parse\_cidade

Text

Description automatically generated

Vemos que, neste método, armazenamos o ID do distrito para fins de chave estrangeira posteriormente na base de dados e, de seguida, vamos buscar o nome de cada uma das cidades e mandamos para a pipeline o dicionário representado através da expressão yield

Já na pipeline, fazemos uma confirmação de dados para garantir a sua validade e inserimos na base de dados. Como podemos ver, o campo “qualidade” está atribuído para “N/A” porque esta Spider não se encarrega da qualidade de vida da cidade, apenas dos campos mencionados acima

Text

Description automatically generated

**Códigos Postais**

A informação dos códigos postais estão localizadas dentro dos links que envolvem os nomes das cidades. Assim, esta Spider será algo similar á anterior, mas em vez de acabar no parse cidades, temos que mudar de contexto para o url mencionado.

Deste modo, os atributos desta Spider serão:

Text

Description automatically generated

E os métodos parse e parse\_cidade serão:

Text

Description automatically generated

Apenas é de notar que, em vez de fazermos yield para a pipeline, fazemos para outro método de modo a armazenar o código postal da seguinte maneira:

Text

Description automatically generated

Já na pipeline, filtramos resultados inválidos e inserimos os válidos na base de dados

Text

Description automatically generated