



**ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO**

CTeSP Cibersegurança, Redes e Sistemas Informáticos

Lousada – 2º Ano

Unidade Curricular

Programação II

Trabalho Prático

Alunos:

Rui Alexandre Borba Vitorino – 8190479

Luis Miguel da Silva de Sá – 8190448

Nelson de Jesus Jorge Laureano – 8190460

**Docente:
Ana Filipe Barros Duarte**

Índice

Índice	II
Índice de Figuras	III
Introdução	IV
I. Objetivos do trabalho.....	5
II. Breve explicação das tecnologias	6
III. Configuração Inicial	8
IV. Estrutura do Django.....	11
i. Estrutura geral	11
ii. Ficheiros importantes	12
V. Ficheiro views.py.....	14
VI. Classe Product	15
iii. Estrutura da tabela Product	15
iv. Código do ficheiro product.py	17
VII. Classe Supplier	22
v. Estrutura da tabela Supplier.....	22
vi. Código do ficheiro supplier.py.....	23
VIII. Alterações realizadas	24
IX. Verificação de resultados.....	26
X. Conclusões	30
XI. Referências	31

Índice de Figuras

Figura 1 Criação de uma diretoria com o comando md.....	8
Figura 2 Criação de um ambiente virtual.....	8
Figura 3 Inserção da diretoria no IDE Visual Studio Code	9
Figura 4 Seleção do ambiente virtual	9
Figura 5 Criação de terminal no ambiente virtual	10
Figura 6 Atualização do pip.....	10
Figura 7 Instalação do Django.....	10
Figura 8 URLs presentes no ficheiro mysite/urls.py	12
Figura 9 Ficheiro index.html em uso (com alterações).....	12
Figura 10 Alguns conteúdos da base de dados.....	13
Figura 11 URLs relativos aos ficheiros HTML	13
Figura 13 Colunas presentes na tabela Product	15
Figura 14 Dados presentes na tabela Product	16
Figura 15 Métodos de obtenção de informação (nome produto e unidades em stock)	17
Figura 16 Read de um produto	17
Figura 17 Métodos principais da classe Product	18
Figura 18 Criação e remoção de um produto	19
Figura 19 Estrutura da tabela Supplier	22
Figura 20 Métodos de listagem do código supplier.py	23
Figura 21 Opção required no create.html	24
Figura 22 Colocação do símbolo do euro e nome do produto.....	24
Figura 23 Nome do produto ao eliminar	25
Figura 24 Alerta de stock a 0	25
Figura 25 Criação de um produto	26
Figura 26 Produto criado com sucesso (Cookies)	26
Figura 27 Requerimento de inserção de informações	27
Figura 28 Pop-up de aviso de tipo inválido de informação.....	27
Figura 29 Alterações com sucesso a um produto	28
Figura 30 Verificação de estatísticas e informações de produtos	28
Figura 32 Remoção de um produto	29
Figura 33 Remoção do product Chai com sucesso	29

Introdução

Presente neste relatório, encontra-se a resolução do trabalho prático proposto pela docente Ana Filipe Barros Duarte da unidade curricular de Ferramentas de Segurança Informática, da turma de 2º ano do CTeSP de Cibersegurança, Redes e Sistemas Informáticos.

O trabalho consiste na criação de várias classes e respetivos métodos, nomeadamente de um produto e um fornecedor, de modo a realizar certas operações em um website gerido pelo framework Django, atingindo-se assim o objetivo de gestão de produtos e o fornecedor associado.

Durante o relatório encontrar-se-á figuras ilustrativas com as operações de gestão a realizar, bem como uma explicação do código que fora necessário criar.

Na conclusão, encontra-se uma análise final do trabalho prático e eventual críticas e dificuldades encontradas durante a realização do mesmo.

I. Objetivos do trabalho

Como indicado no enunciado do trabalho prático, usando o método CRUD (Create, Read, Update, Delete), ou seja, deverá ser possível realizar qualquer etapa de gestão de dados relacionados com o negócio fictício usado como exemplo.

Existindo vários dados na base de dados a utilizar (de nome, Northwind), sendo os principais os dados relativos ao produto, fornecedor, e eventualmente, as entregas.

Ambas as classes irão estar interligadas e a receber informações provenientes do website, e fá-lo através do ficheiro views.py, sendo então necessário criar código nesse ficheiro, contendo em comentário, os tópicos principais a escrever, para colocar o sistema em funcionamento.

Resumindo, é possível gerar os seguintes tópicos com objetivos:

- Criação das classes necessárias (produto e fornecedor)
 - E criação dos respetivos métodos CRUD em cada classe
- Fazer uso do ficheiro views.py
- Uso correto do paradigma POO

Após a criação do código necessário, e de eventuais testes de funcionamento, o website e a gestão dos produtos e o seu fornecedor deverá ser possível.

II. Breve explicação das tecnologias

Django – Framework Web Python, que permite rapidamente lançar um website e respetiva implementação de aplicações no mesmo, de forma simples e segura, usando diversos modelos e princípios; isto é, dividindo a aplicação em “camadas” para que o desenvolvedor apenas se tenha que preocupar com a sua tarefa em específico, aumentando-se a produtividade.

Framework – conjunto de códigos de baixo nível, escritos de forma a serem escaláveis, seguros e robustos, permitindo ao desenvolver final apenas se focar na criação de códigos de alto nível, como a criação e integração da sua aplicação no respetivo framework já existente (pe. website).

Python – linguagem de programação de alto nível, sem necessidade de compilação de código, suportando múltiplos paradigmas de programação como orientação a objetivos (POO), programação funcional e estruturada, e, devido à sua fácil leitura e escritura de código, é uma das mais utilizadas globalmente, em diversas áreas, desde aplicações, *scripts*, módulos de sistemas operativos.

SQL – em português, linguagem de consulta estruturada, utilizada na pesquisa e gestão de tabelas e dados presentes em uma base de dados relacional, e semelhante ao Python, devido à sua geral facilidade, é utilizada globalmente, tendo a particularidade de apresentar o resultado e não o caminho para o mesmo, ao contrário de outras linguagens de consulta.

Visual Studio Code – IDE grátis (em português, ambiente de desenvolvimento integrado) criado pela Microsoft, multiplataforma, que permite a edição de códigos em diversas linguagens de programação, suportando extensões e tornando o desenvolvimento de código completo em qualquer ambiente/linguagem.

Ambiente Virtual (Python) – de modo a facilitar o suporte a específicas bibliotecas, módulos, programas, requeridos pelas aplicações, estas que não se encontram na instalação padrão do Python, o uso de um ambiente virtual é a solução, este que é uma diretoria contendo uma instalação específica do Python e respetivos pacotes necessários (pe. Django e pip), estando totalmente separado da instalação local do Python.

Chave Primária – chave única que identifica em uma tabela um respetivo dado, semelhante e normalmente de nome, ID, ou seja, identificador. Não poderá ser nulo, nem existir valores duplicados.

Chave Estrangeira – é um identificador, isto é, uma chave primária de outra tabela que está agora na tabela atual, referenciando e interligando dados (pe. SupplierID na tabela Product).

Classe – “blueprint” ou “template” para a criação de um objeto, que terá métodos associados. O exemplo geral seria que um Animal seja a Classe e Andar seria um método da classe Animal.

Método – funções que contem certa operação a realizar relacionada com a sua classe.

Query / queries – código em SQL que permite realizar certa tarefa perante dados/tabelas/colunas na base de dados, como por exemplo, seleção de produtos associados com certo fornecedor, ou eliminação de produto.

III. Configuração Inicial

Explicando e exemplificando a configuração inicial de um ambiente virtual, onde se poderia realizar um projeto utilizando o framework Django e respetivo website.

Começando em criar uma diretoria, onde se situará todos as diretorias e ficheiros necessários, de forma tradicional (botão direito do rato no ambiente de trabalho, criar pasta) ou com o seguinte comando no terminal de escolha, `md "nome_diretoria"`.

```
PS C:\Users\RandomPenguin\Desktop\CRSI\PII> md projeto

Directory: C:\Users\RandomPenguin\Desktop\CRSI\PII

Mode                LastWriteTime         Length Name
----                -
d-----          17-Jan-21   06:13 PM              projeto

PS C:\Users\RandomPenguin\Desktop\CRSI\PII> |
```

Figura 1 Criação de uma diretoria com o comando md

Procedendo, ir-se-á criar um ambiente virtual, aconselhado durante a unidade curricular, mas também como boa prática durante o desenvolvimento de um projeto, novamente e agora exclusivamente através da linha de comandos, com o executável do [Python](#) e o argumento `-m venv "nome_diretoria"`.

```
PS C:\Users\RandomPenguin\Desktop\CRSI\PII\projeto> python -m venv ambientevirtual
PS C:\Users\RandomPenguin\Desktop\CRSI\PII\projeto> ls .\ambientevirtual\

Directory: C:\Users\RandomPenguin\Desktop\CRSI\PII\projeto\ambientevirtual

Mode                LastWriteTime         Length Name
----                -
d-----          17-Jan-21   06:15 PM              Include
d-----          17-Jan-21   06:15 PM              Lib
d-----          17-Jan-21   06:15 PM              Scripts
-a----          17-Jan-21   06:15 PM              89 pyvenv.cfg

PS C:\Users\RandomPenguin\Desktop\CRSI\PII\projeto> |
```

Figura 2 Criação de um ambiente virtual

Estando as configurações básicas criadas, é necessário, agora no editor de código, neste caso no Visual Studio Code, abrir a diretoria recentemente criada, usando o atalho de teclado (padrão) **Ctrl + K, Ctrl + O** ou navegando até à aba **File > Open Folder...**

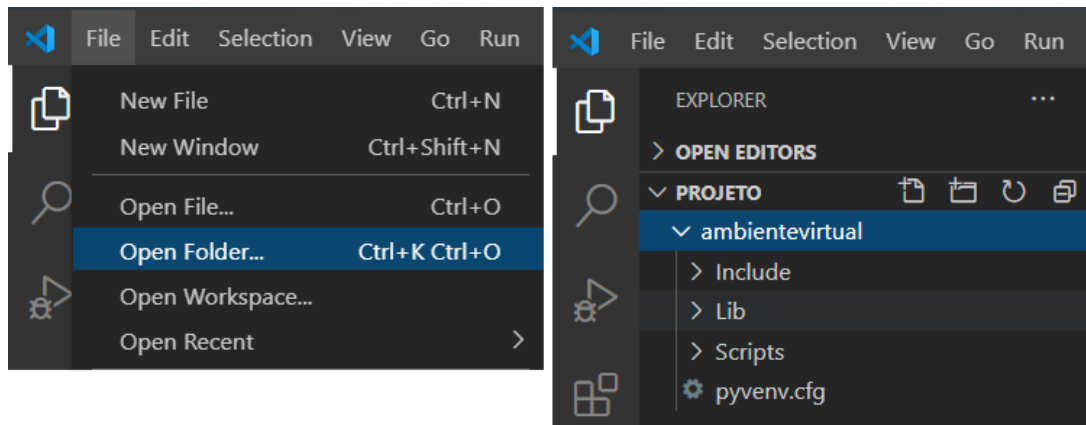


Figura 3 Inserção da diretoria no IDE Visual Studio Code

Como será possível observar na figura, o ambiente virtual encontra-se na diretoria, como de esperar, podendo-se assim proceder à escolha desse ambiente virtual e à instalação de pacotes necessários.

Na aba **View, Command Palette** (ou atalho de teclado **Ctrl + Shift + P**, padrão) seleciona-se a opção **Python Interpreter**, e ir-se-á seleccionar a opção que conterà o caminho da nossa diretoria e entre parenteses o nome do ambiente virtual.

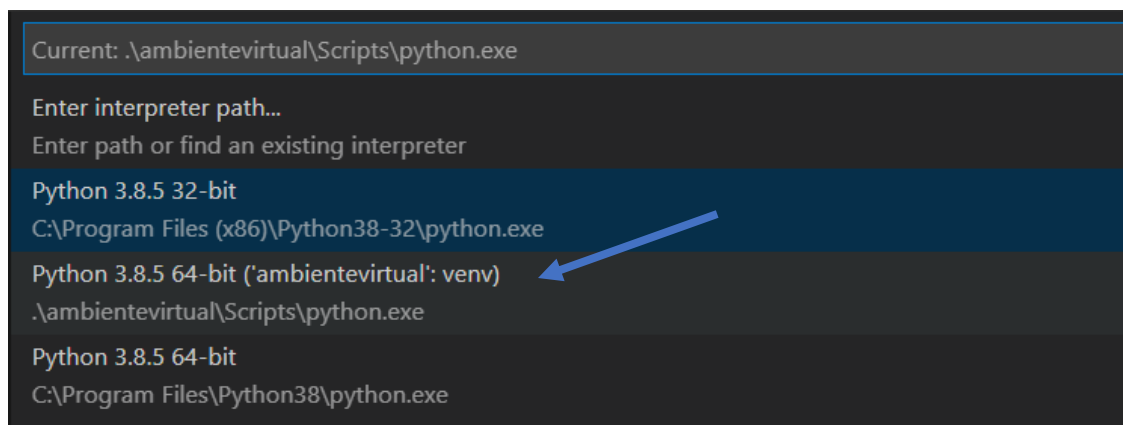


Figura 4 Seleção do ambiente virtual

Estando agora o ambiente virtual selecionado, é possível aceder a um terminal específico navegando até à aba **Terminal**, **New Terminal** (ou atalho de teclado, **Ctrl + Shift + ç**), e veremos que a verde, estará o nome do nosso ambiente virtual.

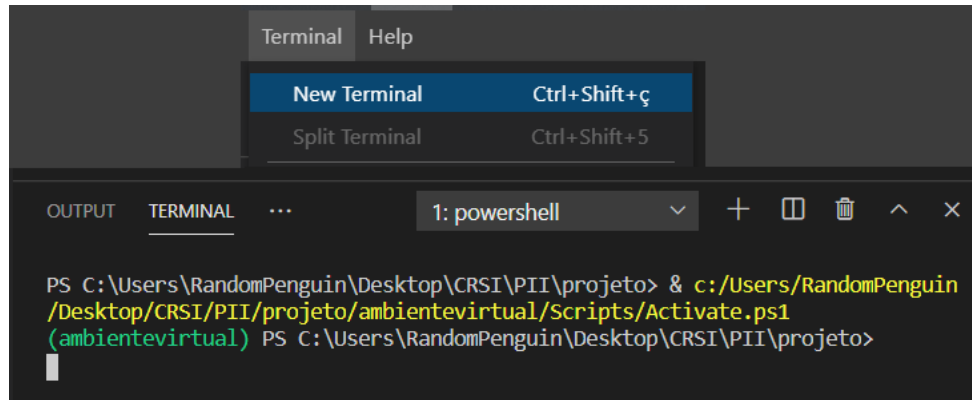


Figura 5 Criação de terminal no ambiente virtual

Aqui poder-se-á então instalar os pacotes, quase finalizando a configuração, mas como boa prática, a atualização do instalador (pip) é recomendada, usando o comando **pip install --upgrade pip**.

```
(ambientevirtual) PS C:\Users\RandomPenguin\Desktop\CRSI\PII\projeto> pip install --upgrade pip
Collecting pip
  Using cached pip-20.3.3-py2.py3-none-any.whl (1.5 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 20.1.1
    Uninstalling pip-20.1.1:
      Successfully uninstalled pip-20.1.1
```

Figura 6 Atualização do pip

E procedendo à última etapa, a instalação do Django, com o comando **pip install django**.

```
(ambientevirtual) PS C:\Users\RandomPenguin\Desktop\CRSI\PII\projeto> pip install django
Collecting django
  Using cached Django-3.1.5-py3-none-any.whl (7.8 MB)
Collecting asgiref<4,>=3.2.10
  Using cached asgiref-3.3.1-py3-none-any.whl (19 kB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.1-py3-none-any.whl (42 kB)
Collecting pytz
  Using cached pytz-2020.5-py2.py3-none-any.whl (510 kB)
Installing collected packages: sqlparse, pytz, asgiref, django
Successfully installed asgiref-3.3.1 django-3.1.5 pytz-2020.5 sqlparse-0.4.1
(ambientevirtual) PS C:\Users\RandomPenguin\Desktop\CRSI\PII\projeto>
```

Figura 7 Instalação do Django

Após tudo isto poderemos começar a criar a nossa aplicação como irá ser demonstrado nos próximos tópicos.

IV. Estrutura do Django

i. Estrutura geral

O Django, seja uma instalação de raiz, ou a instalação a utilizar fornecida pela docente da unidade curricular, apresenta uma estrutura, e é importante conhecê-la para simplesmente entender o funcionamento geral da aplicação, podendo assim criar código eficiente, e tal como um dos objetivos do Django indica, este foi feito para evitar a repetição de código e reutilizar métodos já criados.

Diretoria	Função	Ficheiros importantes
mysite	Códigos principais	urls.py
mysite/templates	<i>Templates</i> HTML	create.html delete.html index.html read.html statistics.html update.html
db	Base de dados	Northwind.db
polls	Códigos associados	product.py supplier.py urls.py views.py
env	Ambiente Virtual	<i>apenas a instalação do Django é importante</i>

ii. Ficheiros importantes

Começando com o ficheiro `urls.py` da diretoria principal do projeto, neste caso `mysite`, este que irá conter, como o nome indica, os URLs do website, já existindo no código as opções necessárias.

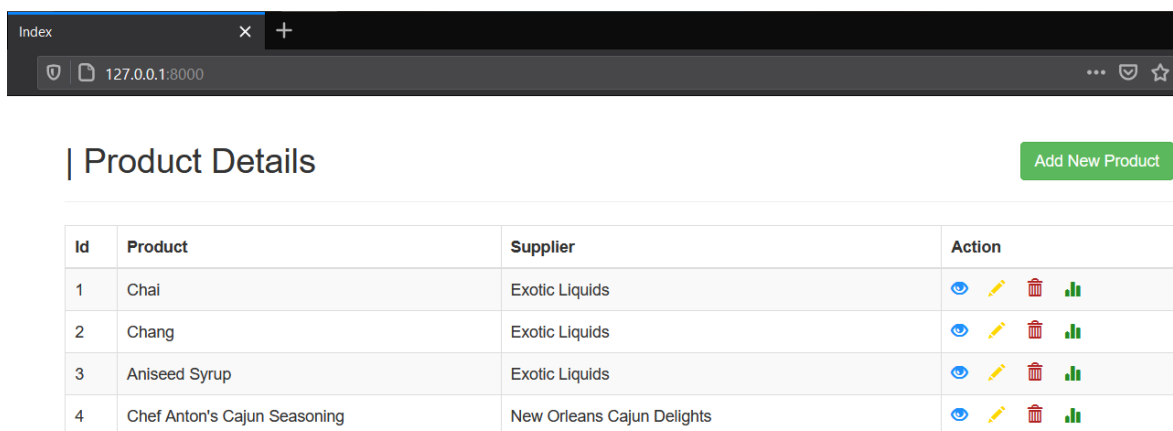
Foi apenas referido este ficheiro pois é uma parte importante de todos os projetos web com o Django.

```
urlpatterns = [
    path('', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

Figura 8 URLs presentes no ficheiro `mysite/urls.py`

Explicando, existe o URL principal, denominado pelas aspas `' '`, que irá atribuir as URLs presentes no ficheiro `urls.py` da diretoria `polls` (que irá conter os códigos a realizar), e o URL de administração do Django.

Na diretoria `mysite/templates/ficheiros.html` encontram-se os ficheiros base de cada website escritos em HTML, nomeadamente das páginas a encontrar durante a criação, remoção, atualização, visualização de estatísticas sobre os produtos e também o website principal, de nome, `index.html`.



















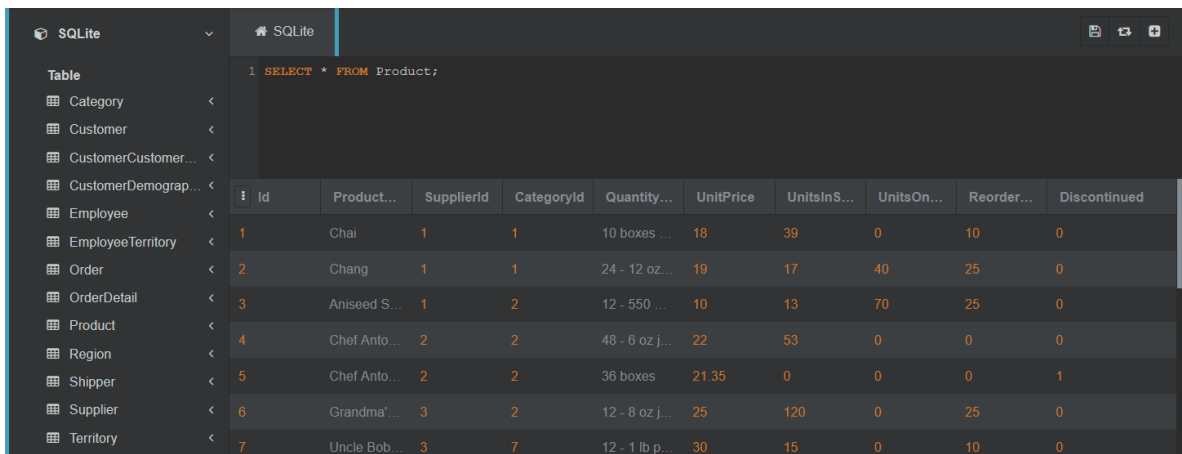
Id	Product	Supplier	Action
1	Chai	Exotic Liquids	   
2	Chang	Exotic Liquids	   
3	Aniseed Syrup	Exotic Liquids	   
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	   

Figura 9 Ficheiro `index.html` em uso (com alterações)

Na diretoria de nome, [db](#), encontra-se o ficheiro da base de dados a utilizar, esta que têm como nome, [Northwind.db](#), e é bastante utilizada na criação de projetos de aprendizagem.

Com a ajuda de um website, referenciado mais em detalhe posteriormente no relatório, é possível visualizar alguns dos conteúdos que a base de dados contém.



The screenshot shows a SQLite database viewer interface. On the left, a sidebar lists various database tables: Category, Customer, CustomerCustomer..., CustomerDemograp..., Employee, EmployeeTerritory, Order, OrderDetail, Product, Region, Shipper, Supplier, and Territory. The 'Product' table is selected. The main area displays a SQL query: `1 SELECT * FROM Product;` Below the query, a table of data is shown with the following columns: Id, Product..., SupplierId, CategoryId, Quantity..., UnitPrice, UnitsInS..., UnitsOn..., Reorder..., and Discontinued. The data rows are as follows:

Id	Product...	SupplierId	CategoryId	Quantity...	UnitPrice	UnitsInS...	UnitsOn...	Reorder...	Discontinued
1	Chai	1	1	10 boxes ...	18	39	0	10	0
2	Chang	1	1	24 - 12 oz ...	19	17	40	25	0
3	Aniseed S...	1	2	12 - 550 ...	10	13	70	25	0
4	Chef Anto...	2	2	48 - 6 oz j...	22	53	0	0	0
5	Chef Anto...	2	2	36 boxes	21.35	0	0	0	1
6	Grandma'...	3	2	12 - 8 oz j...	25	120	0	25	0
7	Uncle Bob...	3	7	12 - 1 lb p...	30	15	0	10	0

Figura 10 Alguns conteúdos da base de dados

Terminando, e chegando à parte mais importante, relativamente ao trabalho a desenvolver, na diretoria de nome [polls](#), será onde os nossos ficheiros de códigos a criar serão colocados ([product.py](#) e [supplier.py](#)), contendo também o ficheiro [views.py](#), este que irá interligar todos os códigos, seja HTML ou Python, de forma a disponibilizar as funcionalidades a desenvolver de forma simples e centralizada.

Nesta diretoria existe outro ficheiro de URLs, que irá fornecer as URLs mais “locais” para o website, indicando a localização dos ficheiros HTML e o respetivo URL de acesso, (pe [127.0.0.1:8000/read/](#) > [read.html](#)).

```
urlpatterns = [
    path('', views.index, name='index'),
    path('read/<int:key>/', views.read, name='read'),
    path('create', views.create, name='create'),
    path('update/<int:key>/', views.update, name='update'),
    path('delete/<int:key>/', views.delete, name='delete'),
    path('statistics/<int:key>/', views.statistics, name='statistics'),
]
```

Figura 11 URLs relativos aos ficheiros HTML

V. Ficheiro views.py

Embora as operações a realizar sejam efetuadas usando outros códigos, como `product.py` e `supplier.py`, é com o `views.py` que os dados são “transferidos” pelo website e colocados na sua forma correta para serem devidamente utilizados em qualquer método posterior.

É o caso do método `index()`, que irá listar todos os produtos (e o nome do fornecedor associado) na página principal no website (`index.html`), usando o método `list_products()` no ficheiro `product.py`.

Mencionando também o próximo método, `read()`, este que irá listar várias informações sobre o produto, que semelhante ao método `index()`, irá buscar essas informações aos respetivos métodos no ficheiro `product.py`.

Relativamente a outras operações como criação, atualização, remoção e listagem de estatísticas, os comentários presentes no código foram respondidos, com ligeiras alterações efetuadas, estas que irão ser mencionadas posteriormente no relatório.

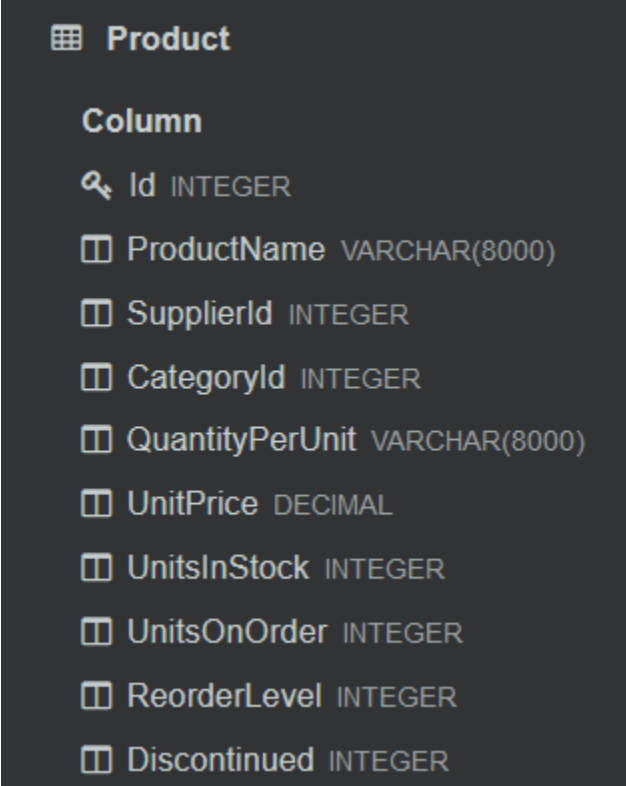
VI. Classe Product

Passando então para a criação da primeira classe necessária e pretendida pelo trabalho, a classe de nome **Product** (em português, Produto), irá conter todos os dados relacionados com, obviamente o produto, tendo também, como chave estrangeira, o ID do fornecedor, SupplierID.

iii. Estrutura da tabela Product

É importante conhecer a estrutura da tabela **Product** na base de dados, de modo a entender que dados irão ser geridos, e como esta se interliga com outros dados/tabelas.

Usando o website SQL OnLine IDE [\[1\]](#), este que permite visualizar e executar queries SQL diretamente no browser, é possível observar alguns dados presentes bem como a estrutura da tabela:



Product	
Column	
Id	INTEGER
ProductName	VARCHAR(8000)
SupplierId	INTEGER
CategoryId	INTEGER
QuantityPerUnit	VARCHAR(8000)
UnitPrice	DECIMAL
UnitsInStock	INTEGER
UnitsOnOrder	INTEGER
ReorderLevel	INTEGER
Discontinued	INTEGER

Figura 12 Colunas presentes na tabela Product

Verifica-se então na figura, que existem diversas colunas, desde o nome do produto, como uma chave estrangeira do fornecedor (que irá obviamente ser de bastante utilidade), colunas relacionadas com a quantidade e preço, outras com informações sobre possíveis entregas, e até mesmo uma chave estrangeira da categoria do produto.

Usando um comando SQL de forma a se obter todas as informações de todas as colunas (**SELECT ***) da tabela Product (**FROM Product**), veremos alguns dados já presentes na base de dados e na respetiva tabela.

```
1 SELECT * FROM Product
```

	Id	Product...	SupplierId	CategoryId	QuantityPerUnit	UnitPrice	UnitsInS...	UnitsOn...	Reorder...	Discontinued
1		Chai	1	1	10 boxes x 20 bags	18	39	0	10	0
2		Chang	1	1	24 - 12 oz bottles	19	17	40	25	0
3		Aniseed S...	1	2	12 - 550 ml bottles	10	13	70	25	0
4		Chef Anto...	2	2	48 - 6 oz jars	22	53	0	0	0
5		Chef Anto...	2	2	36 boxes	21.35	0	0	0	1
6		Grandma'...	3	2	12 - 8 oz jars	25	120	0	25	0
7		Uncle Bob...	3	7	12 - 1 lb pkgs.	30	15	0	10	0
8		Northwoo...	3	2	12 - 12 oz jars	40	6	0	0	0

Figura 13 Dados presentes na tabela Product

Existindo agora uma noção básica da estrutura da tabela **Product** e eventuais ligações com outras tabelas (**Supplier** e **Category**), poder-se-á proceder à criação da classe em um novo ficheiro, que terá como nome [product.py](#).

iv. Código do ficheiro product.py

Existindo vários métodos dentro da única classe de nome Product, estes que realizam as operações CRUD ou obtêm outras informações em particular, como o nome do produto, ou as unidades em stock.

```
def get_product_unitPrice(self, key):
    self.connect()

    self.cursor.execute('SELECT UnitPrice FROM Product WHERE Id=?', (key,))
    unitPrice, = self.cursor.fetchall()

    self.disconnect()
    return unitPrice[0]

def get_product_unitsInStock(self, key):
    self.connect()

    self.cursor.execute('SELECT UnitsInStock FROM Product WHERE Id=?', (key,))
    unitsInStock, = self.cursor.fetchall()

    self.disconnect()
    return unitsInStock[0]
```

Figura 14 Métodos de obtenção de informação (nome produto e unidades em stock)

Aquando se necessita de obter informação, realiza-se um **SELECT** a certa coluna da respetiva tabela, onde o ID será o enviado pelos *forms* do Django, basicamente usando o ID do produto selecionado no website.

| View Record

Product Id

10

Product Name

Ikura

Supplier

Tokyo Traders

Unit Price

31 €

Units in Stock

31 Ikura(s)

Método get_productName()

Método get_product_unitPrice()

Figura 15 Read de um produto

Existindo chamadas de vários métodos durante qualquer operação, nomeadamente, os métodos de ligação e desconexão à base de dados, estes que estão presentes no início da classe, bem como a localização da base de dados no nosso sistema, facilitando assim a sua utilização; e não esquecendo de importar o módulo `sqlite3` (presente na primeira linha), de forma a se obter as funcionalidades de gestão da base de dados.

Outro método importante é o de listagem de produtos, este que irá realizar um **INNER JOIN**, isto é, uma junção de certa coluna de outra tabela, neste caso, `SupplierID` (da tabela `Supplier`) para a tabela `Product`, podendo assim listar o fornecedor, incluindo o seu nome, de certo produto.

```
import sqlite3

class Product:
    def connect(self):
        self.connection = sqlite3.connect(r'C:\Users\RandomPenguin\Desktop\CRSI\PII\TPPII\db\Northwind.sqlite')
        self.cursor = self.connection.cursor()

    def disconnect(self):
        self.connection.close()

    def list_product(self):
        self.connect()

        self.cursor.execute('SELECT Product.Id, Product.ProductName, Supplier.companyname \
        FROM Product INNER JOIN Supplier ON Product.SupplierId = Supplier.Id')
        info_product = self.cursor.fetchall()

        self.disconnect()
        return info_product
```

Figura 16 Métodos principais da classe `Product`

De modo a responder aos objetivos principais do trabalho, como a criação de um produto, criou-se os respetivos métodos, semelhante aos acima referidos, alterando apenas as queries SQL e os dados a serem passados do website para o método.

```
def create_product(self, nomeProduto, nomeSupplier, unitsPrice, unitsInStock):
    self.connect()

    self.cursor.execute("SELECT * FROM Product WHERE id = (SELECT MAX(id) FROM Product)")
    lastid, = self.cursor.fetchall()

    self.cursor.execute("INSERT INTO Product \
    VALUES (?, ?, ?, 0, 0, ?, ?, 0, 0, 0)", (int(lastid[0] + 1), nomeProduto, nomeSupplier, unitsPrice, unitsInStock))
    self.connection.commit()
    self.disconnect()

def delete_product(self, key):
    self.connect()

    self.cursor.execute("DELETE FROM Product WHERE Id=?", (key,))
    self.connection.commit()
    self.disconnect()
```

Figura 17 Criação e remoção de um produto

A query SQL usada para criar um produto, ou **qualquer outra operação que realize alterações** ao esquema ou aos dados presentes na base de dados requer um `commit()`, de forma a confirmar o que fora feito, de certa forma é um guardar alterações feitas a um ficheiro Word, por exemplo.

Um aspeto a referir durante a criação do produto, nomeadamente ao ID do novo produto, este que irá obter o último ID presente na coluna ProductID usando um query e acrescentando um valor; isto pois, devido a alguma motivo desconhecido não se conseguiu utilizar a função `cursor.lastrowid`.

Já estando os métodos criados relativamente às operações de listagem, criação e remoção, faltará a operação de atualização de certo produto.

```
def update_product(self, productName, supplierName, unit_price, units_in_stock, key): #DONE
    self.connect()

    self.cursor.execute("UPDATE Product SET ProductName=?, SupplierId=?, UnitPrice=?, UnitsInStock=? WHERE Id=?", \
        (productName, supplierName, unit_price, units_in_stock, key))
    self.connection.commit()
    self.disconnect()
```

Este, que usando a query SQL **UPDATE**, irá atualizar os valores presentes no website, tais como nome, ID do fornecedor, preço por unidade e unidades em stock.

Tal como a operação de criação de um produto, existe a interligação de tabelas, obtendo através do ID do fornecedor o seu nome, sendo de fácil compreensão para o utilizador final. Esta ligação e respetiva query é efetuada no ficheiro [views.py](#), este que já fora explicado, estando o código referente presente na figura 17.

Relativamente à última secção do ficheiro `product.py`, que irá criar certas estatísticas sobre o stock existentes e respetivos valores, consoante uma taxa em percentagem, e quantidades em várias entregas, novamente usando queries SQL com o `SELECT`.

```
def get_total_including_taxes(self, key, tax):
    self.connect()

    total = (self.get_total_without_taxes(key) * (1 + tax / 100))

    self.disconnect()
    return total

def get_total_quantity(self, key):
    self.connect()

    self.cursor.execute('SELECT SUM(quantity) FROM OrderDetail WHERE productid=?', (key,))
    quantity, = self.cursor.fetchall()

    self.disconnect()
    return quantity[0]
```

No método de nome `get_total_including_taxes()`, ir-se-á obter o valor retornado pelo método `get_total_without_taxes()`, (não mostrado na figura) onde simplesmente se irá realizar uma multiplicação dos valores preço por unidade em stock, e então multiplicando pela taxa, obtendo-se assim o valor total com a respetiva taxa de 23%.

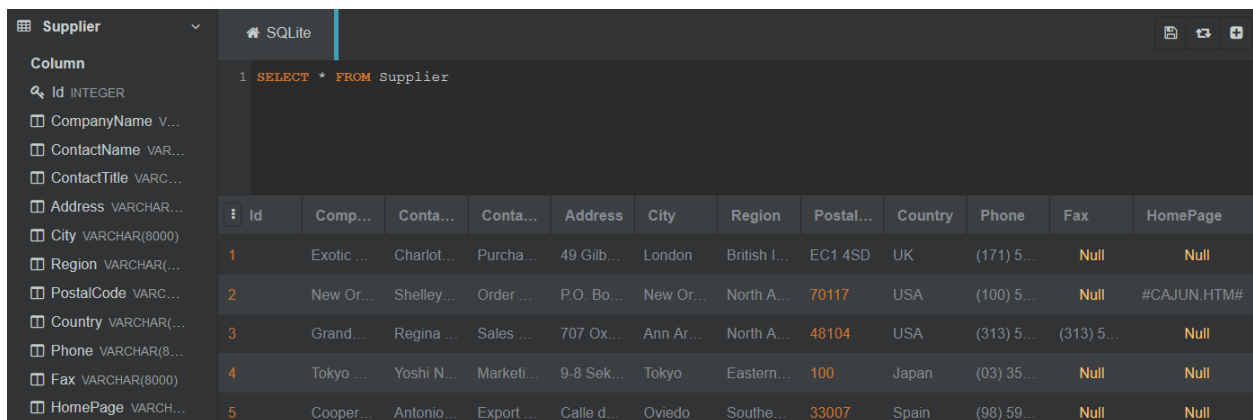
No método de nome `get_total_quantity()`, a query SQL já realiza a operação necessária, neste caso que seria somar (`SUM`) a quantidade de produtos relativos a várias entregas do produto em questão (usando o `WHERE=id`), assim, evitou-se múltiplos queries à base de dados e eventual cálculo no código.

VII. Classe Supplier

Sendo esta classe e ficheiro de fácil criação, mesmo apresentando uma estrutura de maior dimensão comparando com a tabela produto, isto devido apenas ser necessário algumas informações em particular.

v. Estrutura da tabela Supplier

Usando novamente o website SQLOnline, é possível observar a estrutura da tabela e os campos que iremos necessitar, que serão o nome e o ID.



The screenshot shows a SQLite database interface. On the left, a sidebar lists the columns of the 'Supplier' table: Id (INTEGER), CompanyName (VARCHAR), ContactName (VARCHAR), ContactTitle (VARCHAR), Address (VARCHAR), City (VARCHAR(8000)), Region (VARCHAR), PostalCode (VARCHAR), Country (VARCHAR), Phone (VARCHAR(8000)), Fax (VARCHAR(8000)), and HomePage (VARCHAR). The main area displays a SQL query: '1 SELECT * FROM Supplier'. Below the query, a table view shows the data for the 'Supplier' table. The table has 13 columns: Id, Comp..., Conta..., Conta..., Address, City, Region, Postal..., Country, Phone, Fax, and HomePage. The data is as follows:

Id	Comp...	Conta...	Conta...	Address	City	Region	Postal...	Country	Phone	Fax	HomePage
1	Exotic ...	Charlot...	Purcha...	49 Gilb...	London	British I...	EC1 4SD	UK	(171) 5...	Null	Null
2	New Or...	Shelley...	Order ...	P.O. Bo...	New Or...	North A...	70117	USA	(100) 5...	Null	#CAJUN.HTM#
3	Grand...	Regina ...	Sales ...	707 Ox...	Ann Ar...	North A...	48104	USA	(313) 5...	(313) 5...	Null
4	Tokyo ...	Yoshi N...	Marketi...	9-8 Sek...	Tokyo	Eastern...	100	Japan	(03) 35...	Null	Null
5	Cooper...	Antonio...	Export ...	Calle d...	Oviedo	Southe...	33007	Spain	(98) 59...	Null	Null

Figura 18 Estrutura da tabela Supplier

vi. Código do ficheiro supplier.py

Consequentemente o código do ficheiro e a classe é de simples e rápida criação, tendo apenas os métodos de ligação e desconexão à base de dados, e dois métodos, um de listagem geral dos dados de certo fornecedor e outro de listagem de um nome de certo fornecedor dado respetivo ID.

Sendo o último método o que poderá gerar mais confusão, contendo um **LEFT JOIN** para interligação do ID do produto com o ID do fornecedor associado, isto pois, em qualquer tópico do website é mostrado o nome do fornecedor em vez do seu ID, visto não ser fiável nem eficiente a utilização de IDs de inúmeros fornecedores ao contrário de o seu nome, principalmente para o utilizador final.

Explicando também o porquê da utilização de um **LEFT JOIN** e não um **INNER JOIN** como acontece em métodos anteriores, isto pois, apenas se necessita de dados da tabela “à esquerda” (Product) e de todos os dados da tabela “à direita” (Supplier); enquanto ao se utilizar um **INNER JOIN**, ir-se-á mostrar valores interligados entre as duas tabelas, como nome e ID associado.

```
def list_suppliers(self):
    self.connect()

    self.cursor.execute('SELECT Id, CompanyName FROM Supplier')
    info_supplier = self.cursor.fetchall()

    self.disconnect()
    return info_supplier

def get_supplierName(self, key):
    self.connect()

    self.cursor.execute('SELECT companyname FROM Supplier LEFT JOIN Product ON Supplier.id = Product.supplierid WHERE product.id=?', (key,))
    supplierName, = self.cursor.fetchall()

    self.disconnect()
    return supplierName[0]
```

Figura 19 Métodos de listagem do código supplier.py

VIII. Alterações realizadas

Algumas alterações foram realizadas devido a ter existindo tempo e principalmente curiosidade, sendo a mais básica a alteração das cores dos ícones do ficheiro [index.html](#).

Outra alteração fora a colocação da opção **required** nos formulários HTML, desta forma ao se criar ou atualizar o produto é necessário que todos os valores sejam inseridos.

```
<div class="form-group">
  <label for="id_product">New Product</label>
  <input style="width: 550px;" type="text" id="id_product" name="product" class="form-control" required>
</div>
<div class="form-group">
  <label for="id_supplier">Pick up Supplier</label>
  <select id="id_supplier" name="supplier" multiple style="height: 220px; width: 550px;">
    {% for key, value in suppliers %}
      <option value="{{ key }}">{{ value }}</option>
    {% endfor %}
  </select>
</div>
<div class="form-group">
  <label for="id_product">Unit Price: </label>
  <input style="width: 550px;" type="text" id="id_product" name="unit_price" class="form-control" required>
</div>
<div class="form-group">
  <label for="id_product">Units in Stock: </label>
  <input style="width: 550px;" type="text" id="id_product" name="units_in_stock" class="form-control" required>
</div>
```

Figura 20 Opção required no create.html

Uma pequena alteração visual, que até pode passar despercebida, foi adição do símbolo euro e o nome do produto durante a visualização do mesmo, tendo sido efetuado a alteração também no ficheiro [read.html](#).

Unit Price

19 €

Units in Stock

17 Chang(s)

Figura 21 Colocação do símbolo do euro e nome do produto

Como poderia ter sido notado, durante a remoção de um produto, o nome deste irá ser mostrado, evitando uma possível eliminação de um produto errado, confirmando o nome.

| Delete Record

Are you sure you want to delete this product?

| Product: Original Frankfurter grüne Soße

Figura 22 Nome do produto ao eliminar

Por último, adicionou-se nos ficheiros [views.py](#), [read.html](#) e [statistics.html](#), código para alertar o utilizador caso o stock de um produto seja zero (0), emitindo o aviso no website. Caso o stock do produto seja positivo, ir-se-á simplesmente mostrar o stock existente.

```
def read(request, key): #DONE
    product = Product()

    productName = product.get_productName(key)
    unitPrice = product.get_product_unitPrice(key)
    unitsInStock = product.get_product_unitsInStock(key)

    suppliers = Supplier()

    supplierName = suppliers.get_supplierName(key)

    if str(unitsInStock) == "0":
        stock_at_0 = True
    else:
        stock_at_0 = False

    #do not change the lines bellow
    #tambem foi passado o stock_at_0
    t = get_template('read.html')
    html = t.render({'id':key, 'name':productName, 'supplier':supplierName, \
        'unit_price':unitPrice, 'units_in_stock':unitsInStock, 'stock_at_0':stock_at_0})
    return HttpResponse(html)
```

| View Record

Product Id	2
Product Name	Chang
Supplier	Exotic Liquids
Unit Price	19 €
Units in Stock	Stock is at 0
<input type="button" value="Back"/>	

Figura 23 Alerta de stock a 0

IX. Verificação de resultados

Verificando todas as operações pretendidas para realização, bem como as pequenas alterações.

Começando pela criação de um produto, e visualização do mesmo criado.

| Create Product

Please fill this form and submit to add a new Product to the database.

New Product

Pick up Supplier

Exotic Liquids
New Orleans Cajun Delights
Grandma Kelly's Homestead
Tokyo Traders
Cooperativa de Quesos 'Las Cabras'
Mayumi's
Pavlova, Ltd.
Specialty Biscuits, Ltd.
PB Knäckebröd AB
Refrescos Americanas LTDA
Hill Country Sweeteners, LLC

Unit Price:

Units in Stock:

Figura 24 Criação de um produto













76	Bananas	Pavlova, Ltd.	   
77	Santa Cookies	Grandma Kelly's Homestead	   
78	Cookies	Grandma Kelly's Homestead	   

Figura 25 Produto criado com sucesso (Cookies)

Atualizando um produto, usando também o exemplo para a verificação da alteração de requerimento de alteração de campos (o que também acontece durante a criação de um produto, sendo necessário a inserção de todos os campos indicados).

| Update Product

Please edit the input values and submit to update the product

Product Id: 77

Product Name

XmasCookies

Previous was: Santa Cookies

Pick up Supplier

Previous was: Grandma Kelly's Homestead

Exotic Liquids
New Orleans Cajun Delights
Grandma Kelly's Homestead
Tokyo Traders
Cooperativa de Quesos 'Las Cabras'

Unit Price

Por favor preencha este campo.

0

Previous was: 556

Submit Cancel

Figura 26 Requerimento de inserção de informações

Durante a alteração de um produto (ou a criação de um mesmo), caso o código detete que valores incorretos estão a ser introduzidos, como por exemplo, texto no preço unitário, após submeter, o seguinte *pop-up* é apresentado, e a página será atualizada automaticamente (não submetendo as informações).

| Create Product

Please fill this form and submit to add a new Product to the database.

New Product

Pick up Supplier

Invalid type buddy. Are you sure you got everything right?

OK

Figura 27 Pop-up de aviso de tipo inválido de informação

Após a alteração, visto que o stock será 0, e o nome fora alterado, as modificações irão ser mostradas nas estatísticas, incluindo o aviso de stock a 0.

| View Statistics

Product Name

XmasCookies

Total value in stock without taxes

0.0 €

Total value in stock with taxes

0.0 €

Tax Value = 23 %

Total quantity ordered

791 XmasCookies(s)

Total quantity ordered

Stock is at 0

Back

Figura 28 Alterações com sucesso a um produto

Aproveitando para a visualização das estatísticas de produtos, seja no produto recentemente criado ou em qualquer outro, e sem esquecer o primeiro objetivo, a listagem de informações sobre o produto.

| View Statistics

Product Name

Cookies

Total value in stock without taxes

34.5 €

Total value in stock with taxes

42.435 €

Tax Value = 23 %

Total quantity ordered

None Cookies(s)

Back

| View Statistics

Product Name

Queso Cabrales

Total value in stock without taxes

462.0 €

Total value in stock with taxes

568.26 €

Tax Value = 23 %

Total quantity ordered

706 Queso Cabrales(s)

Back

| View Record

Product Id

1

Product Name

Chai

Supplier

New Orleans Cajun Delights

Unit Price

12 €

Units in Stock

39 Chai(s)

Back

Figura 29 Verificação de estatísticas e informações de produtos

Faltando apenas a confirmação da operação de eliminação de produtos.

| Delete Record

Are you sure you want to delete this product?

| Product: Chai

Yes

No

Figura 30 Remoção de um produto

Verifica-se então que o produto de nome Chai, foi eliminado com sucesso.

| Product Details

Add New Product









Id	Product	Supplier	Action
2	Chang	Exotic Liquids	   
3	Aniseed Syrup	Exotic Liquids	   

Figura 31 Remoção do product Chai com sucesso

X. Conclusões

Este trabalho permitiu assimilar todos os conhecimentos obtidos durante as aulas da unidade curricular, bem como explorar um pouco o framework Django, durante a tentativa de eliminação de vários produtos (que infelizmente não fora concluída com sucesso), e certamente cementar o conhecimento sobre classes e o seu objetivo, eficácia na linguagem de programação Python, e até mesmo juntar tecnologias de base de dados com código Python e HTML.

Tendo todos os elementos do grupo contribuído para a realização do mesmo, concluindo todos os tópicos individualmente e centralizando várias funcionalidades à versão final, desta forma existiu contribuição equalizada e integração de cada elemento.

XI. Referências

Todo o material utilizado durante a realização do trabalho foi disponibilizado pelo docente e disponível no Moodle.

<https://www.djangoproject.com/>

[https://en.wikipedia.org/wiki/Django_\(web_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework))

<https://hackr.io/blog/what-is-frameworks>

<https://en.wikipedia.org/wiki/SQL>

<https://code.visualstudio.com/docs/supporting/faq>

<https://docs.python.org/3/tutorial/venv.html>

<https://omannualofreelancer.com/diferenca-chave-primaria-e-chave-estrangeira/>

Repositório Github com algumas versões anteriores do código:

https://github.com/RuiVit/TP_PII