

**IPV – Instituto Superior Politécnico de Viseu**  
**ESTGV – Escola Superior de Tecnologia e Gestão de Viseu**  
**Departamento de Informática**



**Relatório Projeto Final**

**Licenciatura em Engenharia Informática**  
**3º ano, 1º semestre**

**Realizado em**  
**Base de Dados II**

**Por**

**20223 – Alexandre Moreira**

**20255 - Carlos Silva**

**17852 – Gonçalo Marques**

**IPV – Instituto Superior Politécnico de Viseu**  
**ESTGV – Escola Superior de Tecnologia e Gestão de Viseu**  
**Departamento de Informática**

**Relatório Projeto Final**

**Licenciatura em Engenharia Informática**  
**3º ano, 1º semestre**

**Realizado em**  
**Base de Dados II**

**Por**  
**20223 – Alexandre Moreira**  
**20255 - Carlos Silva**  
**17852 – Gonçalo Marques**

**Docentes: Paulo Tomé, Pedro Martins, Paulo Costa**  
**Entidade: Base de Dados 2**

**Viseu, 2023**

# Índice

1.	Projeto .....	6
1.1	Estrutura .....	6
2	Diagrama .....	14
3	Base de Dados .....	16
3.1	PostgreSQL .....	16
3.1.1	Encomendas .....	16
3.1.2	Itens Encomendas .....	17
3.2	MongoDB .....	17
3.2.1	Utilizadores .....	18
3.2.2	Produtos .....	18
3.2.3	Carrinho .....	19
3.2.4	Promoções .....	19
3.2.5	Tipos de Produtos .....	19
3.3	Funções .....	20
3.3.1	GetOrdersByCustomer() .....	20
3.3.2	GetOrder() .....	20
3.3.3	getTop5MostSoldProductsByUser() .....	21
3.3.4	getMostSoldProductByType() .....	21
3.3.5	getMostSoldProductByPartner() .....	22
3.3.6	getSoldProductByPartner() .....	22
3.3.7	getOrdersByOnePartner() .....	23
3.4	Procedimentos .....	23
3.4.1	Orders_insert() .....	23
3.4.2	OrderItems_insert() .....	24
3.4.3	Update_OrderStatus() .....	24
3.4.4	UpdateAll_OrderStatus() .....	25

3.4.5	Orders_delete()	25
3.5	Trigger	25
3.6	Views	27
3.6.1	getMostPopularProduct()	27
3.6.2	getMostPopularProductThisWeek()	27
3.6.3	getTop5MostSoldProduct()	27
3.6.4	getUsersWithMoreOrdersAndHowMany()	28
3.6.5	getUsersOrdersAndHowMany()	28
3.6.6	ordersView()	28
3.6.7	getOrderStatusFalse()	29
3.6.8	getCountOrderStatusFalse()	29
4	Aplicação Web	30
4.1	Utilizadores	30
4.2	Autenticação de utilizadores	31
4.2.1	Funções	32
4.3	Navbar	34
4.4	HomePage	34
4.4.1	Utilizadores não autenticados	34
4.4.2	Cientes	35
4.4.3	Comerciais	36
4.4.4	Parceiros	37
4.4.5	Administradores	38
4.5	Listagem de produtos	39
4.6	Criação de produtos	41
4.6.1	Adicionar tipo de produto	41
4.6.2	Adicionar um novo produto	42
4.7	Alteração de produtos	43
4.8	Encomendas	44

4.9	Carrinho .....	46
4.9.1	Funções do carrinho.....	46
4.10	Promoções.....	48
5	Conclusão .....	50

---

# 1. Projeto

No âmbito da cadeira de Base de Dados II, foi no pedido para desenvolvermos loja digital. A mesma esta dividida em duas partes: consulta dos produtos e aquisição dos mesmos. Para esta última, o utilizador necessita de efetuar login com as suas respetivas credenciais para continuar as operações. Estas duas etapas diferem apenas na informação a que os utilizadores têm acesso, isto é, no caso da consulta de produtos, todos os utilizadores têm acesso aos produtos disponibilizados no site, no caso da aquisição dos mesmos, como o utilizador tem de colocar os seus dados pessoais, apenas este tem acesso a estes.

É de notar que os produtos existentes na aplicação podem ser de duas origens: da própria loja ou de um parceiro externo, sendo que no caso deste último, o preço é estipulado previamente.

Cada utilizador pode aceder ao estado das suas compras, desde que esteja autenticado, estejam estas ainda em andamento ou já efetuadas. Por último, existe ainda uma entidade responsável por adicionar, editar e eliminar produtos.

Para o desenvolvimento deste projeto, utilizamos tecnologias como *PostgreSQL*, *MongoDB*, *Django* e a livreria *Pymongo*.

## 1.1 Estrutura

Este relatório encontra-se dividido em cinco partes. No primeiro capítulo, fazemos uma contextualização do trabalho proposto assim como uma definição dos objetivos do mesmo.

No segundo capítulo, explicar-se-á a estrutura da base de dados.

No terceiro capítulo, encontra-se as diversas funções utilizadas dentro da própria base de dados.

No quarto capítulo, encontra-se a implementação do projeto, os seus requisitos e a devida explicação de como o mesmo opera.

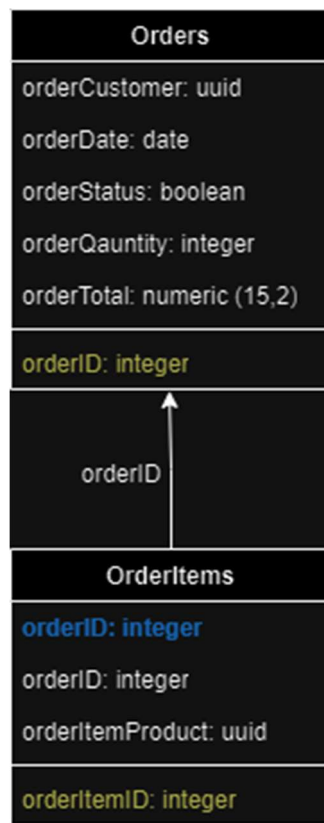
No quarto e último capítulo, encontrar-se a conclusão, explicando as dificuldades encontradas ao longo do projeto assim como a maneira como foram ultrapassadas e algumas reflexões do grupo.

---

## 2 Diagrama

A primeira coisa decidida sobre o projeto foi a estrutura do mesmo, e em seguida apresenta se a mesma.

Como em *PostgreSQL* apenas criamos as tabelas relacionais transacionais, é apresentado o diagrama das mesmas:



---

Já em *MongoDB*, criou-se as seguintes *collections* que são referentes a operações não transacionais.

Users	
<b>_id: string</b>	
name: string	
username: string	
role: string	
password: bindata	
ProductTypes	
<b>_id: string</b>	
productTypeImage: string	
productTypeName: bindata	
Sales	ProductTypes
<b>_id: string</b>	<b>_id: string</b>
productTypeID: string	productID: string
sale: string	userID: string



---

## 3 Base de Dados

De modo a dar suporte ao *website*, utilizamos os motores de base de dados *PostgreSQL* e *MongoDB*. Desta forma, todas as operações que envolvam transações foram implementadas em *PostgreSQL*, isto é, encomendas e respetivas linhas da encomenda, sendo que este último faz referência aos itens existentes em cada encomenda.

De forma a controlar os utilizadores, produtos, promoções, carrinho implementámos as *collections* necessárias para o efeito em *MongoDB*.

### 3.1 PostgreSQL

PostgreSQL é um sistema de gestão de base de dados relacional que utiliza tabelas e modelos como forma de armazenamento dos dados. O PostgreSQL oferece funcionalidades avançadas de segurança, desempenho e extensibilidade que o tornam ideal para o desenvolvimento de projetos como este.

#### 3.1.1 Encomendas

De forma a armazenar as encomendas criadas pelos clientes, criou-se a tabela “*Orders*” em que se armazenam dados referentes ao utilizador que fez a encomenda (id do mesmo), a data em que foi feita, o total a pagar, a quantidade de produtos e o estado da encomenda.

```
create table "public"."Orders"(
    "orderId"      int generated always as identity,
    "orderCustomer" uuid          not null,
    "orderDate"    date           not null default current_date,
    "orderTotal"   numeric(15,2)  not null default 0,
    "orderQuantity" int          not null default 0,
    "orderStatus"  boolean        not null default false,
    constraint "Order_pk" primary key ("orderId")
);
```

---

### 3.1.2 Itens Encomendas

Para especificar os produtos presentes em cada uma das encomendas realizadas pelos clientes, criou-se a tabela “*OrderItems*”, que contém um ID próprio, o ID da encomenda a que se refere, o ID de cada produto presente em cada linha e o seu preço.

```
create table "public"."OrderItems"(  
    "orderItemID"    int generated always as identity,  
    "orderId"        int          not null,  
    "orderItemProduct" uuid       not null,  
    "orderItemPrice" numeric(15,2) not null,  
    constraint "OrderItems_pk" primary key ("orderItemID"),  
    constraint "OrderID_fk" foreign key ("orderId") references "public"."Orders"("orderId")  
);
```

## 3.2 MongoDB

O MongoDB é um sistema de gestão de base de dados NoSQL de código aberto. Ao contrário dos sistemas de base de dados relacionais tradicionais, que armazenam dados em tabelas com esquemas predefinidos, o MongoDB guarda dados em formato de documentos JSON flexíveis, o que o torna adequado para guardar dados semiestruturados ou não estruturados.

Desta forma, consegue obter-se mais velocidade e disponibilidade no manuseio da base de dados, porém, é necessário utilizar linguagens de baixo nível (python, por exemplo).

---

### 3.2.1 Utilizadores

No caso dos utilizadores, adicionamos a respetiva *collection* no ato da criação de conta. Para o efeito, registamos qual o nome do utilizador, o *username* (identificação pela qual faz login) e a respetiva *password*. Caso seja um administrador a criar a conta, este pode escolher que tipo de conta que pretende criar.

```
doc = {"_id": doc_id, 'name': name, 'username': username,  
      'password': hashed_password, 'role': role}  
mongo_client['Users'].insert_one(doc)
```

### 3.2.2 Produtos

Relativamente aos produtos, armazenamos na *collection* correspondente, alguns dados como o nome do produto, preço, tipo de produto, entre outros dados.

```
product = {  
    '_id': doc_id,  
    'productName': productName,  
    'productType': productType,  
    'productQuantity': productQuantity,  
    'productImage': productImage,  
    'productPriceStart': productPriceStart,  
    'productPriceEnd': productPriceEnd,  
    'productDescription': productDescription,  
    'productStatus': productStatus,  
    'vendor': vendor,  
    'roleVendor': roleVendor  
}
```

---

### 3.2.3 Carrinho

Para o cliente poder comprar vários itens ao mesmo tempo em vez de os comprar 1 a 1, decidimos desde cedo que a criação de um carrinho era necessária. Na respetiva *collection* são guardados dados relativamente ao cliente em questão e aos produtos selecionados pelo mesmo.

```
cart = {  
  '_id': doc_id,  
  'userID': userID,  
  'productID': productID  
}
```

### 3.2.4 Promoções

Cada promoção esta associada a um tipo de produtos facilitando assim a criação de promoções para vários produtos do mesmo tipo.

```
sale = {  
  '_id': doc_id,  
  'productTypeID': productTypeID,  
  'sale': sale  
}
```

### 3.2.5 Tipos de Produtos

Para podermos organizar os produtos assim como dar lhes os respetivos descontos criamos a *collection productType*.

```
productType = {  
  '_id': doc_id,  
  'productTypeName': productTypeName,  
  'productTypeImage': productTypeImage  
}
```

---

## 3.3 Funções

Neste subcapítulo, são apresentadas as funções criadas no *PostgreSQL*, de forma a obter os dados desejados das tabelas existentes.

### 3.3.1 GetOrdersByCustomer()

A função apresentada em baixo recebe, por parâmetro, o *id* do cliente do qual se pretende analisar as encomendas e retorna, através de uma query, as encomendas feitas pelo mesmo.

```
create or replace function getOrdersByCustomer (customerID uuid)
    returns setof "public"."Orders"
    language plpgsql
    as $$
    begin
        return query select * from "public". "Orders"
        where "orderCustomer" = customerID
        order by "orderID" desc;
    end;
    $$;
```

### 3.3.2 GetOrder()

Dado o *id* de uma encomenda, esta retorna os dados correspondentes à mesma.

```
create or replace function getOrder(orderID int)
    returns table("orderID" int, "orderCustomer" uuid, "orderDate" date, "orderTotal" numeric, "orderStatus" boolean, "orderProductID" uuid, "orderItemPrice" numeric(15,2))
    language plpgsql
    as $$
    begin
        return query select "Orders"."orderID", "Orders"."orderCustomer", "Orders"."orderDate", "Orders"."orderTotal", "Orders"."orderStatus", "OrderItems"."orderItemProduct", "OrderItems"
        join "OrderItems" on "Orders"."orderID" = "OrderItems"."orderID"
        where "public"."Orders"."orderID" = orderID;
    end;
    $$;
```

---

### 3.3.3 getTop5MostSoldProductsByUser()

A função seguinte retorna quais os cinco produtos mais vendidos de um determinado utilizador. Sendo que o *id* desse utilizador é passado como parâmetro.

```
create or replace function getTop5MostSoldProductsByUser(customerID uuid)
    returns setof uuid
    language plpgsql
    as $$
    begin
        return query select "orderItemProduct" from "public". "OrderItems"
        join "Orders" on "OrderItems"."orderId" = "Orders"."orderId"
        where "Orders". "orderCustomer" = customerID
        group by "orderItemProduct"
        order by count(*) desc
        limit 5;
    end;
    $$;
```

### 3.3.4 getMostSoldProductByType()

Esta função recebe como parâmetro um array com os *ids* dos produtos associados a um determinado tipo de produtos e retorna o *id* daquele que é mais comprado pelos clientes (mais vendido pela loja).

```
create or replace function getMostSoldProductByType(orderProductID uuid[])
    returns uuid
    language plpgsql
    as $$
    declare mostSoldProduct uuid;
    begin
        select "orderItemProduct" into mostSoldProduct from "public". "OrderItems"
        where "orderItemProduct" = (select "orderItemProduct" from "public". "OrderItems"
        where "orderItemProduct" = any (orderProductID)
        group by "orderItemProduct"
        order by count(*) desc
        limit 1);
        return mostSoldProduct;
    end;
    $$;
```

---

### 3.3.5 getMostSoldProductByPartner()

A seguinte função retorna quais o cinco produtos mais comprados pelos clientes, tendo em conta o parceiro que o vende, para isso, esta recebe como parâmetro um array de *ids de produtos* e, após isso, faz uma query de forma a retornar quais os produtos mais comprados pelos clientes e o número de vezes que foram comprados.

```
create or replace function getMostSoldProductByPartner(orderProductID uuid[])
    returns table (count bigint, productID uuid)
    language plpgsql
    as $$
    begin
        return query select count(*), "orderItemProduct" from "public". "OrderItems"
        where "orderItemProduct" = any (orderProductID)
        group by "orderItemProduct"
        order by count(*) desc
        limit 5;
    end;
    $$;
```

### 3.3.6 getSoldProductByPartner()

Para saber qual o produto mais vendido por parceiro, criou-se a seguinte função que, recebendo como parâmetro um array de *ids de produtos* retorna qual os produto mais vendido pelo parceiro e o número de vezes que foi comprado.

```
create or replace function getSoldProductByPartner(orderProductID uuid[])
    returns table (count bigint, productID uuid)
    language plpgsql
    as $$
    begin
        return query select count(*), "orderItemProduct" from "public". "OrderItems"
        where "orderItemProduct" = any (orderProductID)
        group by "orderItemProduct"
        order by count(*) desc;
    end;
    $$;
```

---

### 3.3.7 getOrdersByOnePartner()

Recebendo como parâmetro um array de *ids* de produtos, retorna os produtos e respectivo preço a pagar.

```
create or replace function getOrdersByOnePartner(orderProductID uuid[])
    returns table (productID uuid, orderTotal numeric(15, 2))
    language plpgsql
    as $$
    begin
        return query select "orderItemProduct", "orderTotal"
        from "public". "OrderItems"
        join "Orders" on "OrderItems"."orderID" = "Orders"."orderID"
        where "orderItemProduct" = any (orderProductID);
    end;
    $$;
```

## 3.4 Procedimentos

### 3.4.1 Orders\_insert()

Para que seja possível inserir encomendas nas tabelas criadas em PostgreSQL, foi criado o procedimento Orders\_insert() que recebe como variáveis de entrada um array de ID's de produtos, o ID do cliente em causa e um array com o preço de cada produto. Neste caso é inserida uma nova linha na tabela "Orders" e faz-se uma consulta para obter o ID dos últimos dados inseridos na mesma. Após isso, é chamado outro procedimento (OrderItems\_insert()).

```
create or replace procedure Orders_insert(orderProductID uuid[], orderCustomer uuid, orderPrice numeric(15,2) [])
    language plpgsql
    as $$
    declare orderID int;
    begin
        insert into "public". "Orders" ("orderCustomer")
        values (orderCustomer);
        select "orderID" into orderID from "public"."Orders"
        order by "orderID" desc limit 1;
        call OrderItems_insert(orderID, orderProductID, orderPrice);
    end;
    $$;
```



---

### 3.4.2 OrderItems\_insert()

O procedimento OrderItems\_insert(), por sua vez, recebe como variáveis de entrada o orderID criado anteriormente, o array de ID's de produtos e o um array com o preço dos mesmos. No procedimento existe um ciclo for que irá percorrer o array de ID's de produtos e irá, para cada elemento do mesmo, inserir novos dados na tabela "OrderItems".

```
create or replace procedure OrderItems_insert(orderID int, orderItemProduct uuid[], orderItemPrice numeric(15,2)[])
language plpgsql
as $$
begin
    for i in 1..array_length(orderItemProduct, 1) loop
        insert into "public"."OrderItems"("orderID", "orderItemProduct", "orderItemPrice")
        values (orderID, orderItemProduct[i], orderItemPrice[i]);
    end loop;
end;
$$;
```

### 3.4.3 Update\_OrderStatus()

O procedimento Update\_OrderStatus() recebe por parâmetro um array de ID's de encomenda. Existe um ciclo for que percorre o array e para cada linha que contenha este ID irá alterar a variável "OrderStatus" para true. Isto serve para alterar o estado de uma encomenda.

```
create or replace procedure Update_OrderStatus(orderID int[])
language plpgsql
as $$
begin
    for i in 1..array_length(orderID, 1) loop
        update "public"."Orders"
        set "orderStatus" = true
        where "orderID" = orderID[i];
    end loop;
end;
$$;
```

---

### 3.4.4 UpdateAll\_OrderStatus()

Este procedimento foi criado para atualizar todas as encomendas cuja variável “orderStatus” seja “false”, para “true”. Isto servirá para alterar o estado de todas as encomendas.

```
create or replace procedure UpdateAll_OrderStatus()
language plpgsql
as $$
begin
    update "public". "Orders"
    set "orderStatus" = true
    where "orderStatus" = false;
end;
$$;
```

### 3.4.5 Orders\_delete()

Este procedimento permite a eliminação de uma “Order” conforme o orderID fornecido.

```
create or replace procedure Orders_delete(orderID int)
language plpgsql
as $$
begin
    delete from "public". "Orders"
    where "orderID" = orderID;
end;
$$;
```

## 3.5 Trigger

Este trigger (“Orders\_insert\_trigger”) foi criado para atuar após a inserção de dados na tabela “OrderItems”. Neste caso, quando são inseridos dados na tabela, é executada a função “OrdersItem\_insertTrigger()”. Esta função retorna “trigger” e tem duas variáveis: “totalProductItem” e “totalPrice”. A variável “totalPrice” guarda a soma do preço de todos os produtos inseridos na tabela “OrderItems” em que o ID da encomenda seja igual ao ID da nova encomenda. A variável “totalProductItem” guarda o número total de itens inseridos na tabela “OrderItems” cujo ID da encomenda é igual ao ID da nova encomenda. Por fim são atualizados os campos “orderTotal” e “orderQuantity” na tabela “Orders” com os valores das variáveis “totalPrice” e “totalProductItem” respetivamente.

```

create or replace function OrdersItem_insertTrigger()
returns trigger
language plpgsql
as $$
declare totalProductItem numeric(15,2); totalPrice numeric(15,2);
begin
    select sum("orderItemPrice") into totalPrice from "public". "OrderItems" where "orderId" = new. "orderId";
    select count(*) into totalProductItem from "public". "OrderItems" where "orderId" = new. "orderId";
    update "public"."Orders" set "orderTotal" = totalPrice, "orderQuantity" = totalProductItem where "orderId" = new."orderId";
    return new;
end;
$$;

create or replace trigger Orders_insert_trigger
after insert on "public"."OrderItems"
for each row
execute procedure OrdersItem_insertTrigger();

```

---

## 3.6 Views

### 3.6.1 getMostPopularProduct()

A view “getMostPopularProduct()” apresenta o produto mais vendido de sempre.

```
create or replace view getMostPopularProduct as
  select "orderItemProduct" from "public"."OrderItems"
  where "orderItemProduct" = (select "orderItemProduct" from "public"."OrderItems"
                              group by "orderItemProduct"
                              order by count(*) desc
                              limit 1);
```

### 3.6.2 getMostPopularProductThisWeek()

A view apresentada apresenta qual o produto mais vendido no espaço de uma semana.

```
create or replace view getMostPopularProductThisWeek as
  select "orderItemProduct" from "public"."OrderItems"
  where "orderItemProduct" = (select "orderItemProduct" from "public"."OrderItems" join "Orders" on "OrderItems"."orderID" = "Orders"."orderID"
                              where "orderDate" > current_date - 7
                              group by "orderItemProduct"
                              order by count(*) desc
                              limit 1);
```

### 3.6.3 getTop5MostSoldProduct()

A view “getTop5MostSoldProduct()” apresenta os 5 produtos mais vendidos de sempre.

```
create or replace view getTop5MostSoldProducts as
  select "orderItemProduct" from "public"."OrderItems"
  group by "orderItemProduct"
  order by count(*) desc
  limit 5;
```

---

### 3.6.4 getUsersWithMoreOrdersAndHowMany()

A view “getUsersWithMoreOrdersAndHowMany()” apresenta o cinco utilizadores com mais encomendas feitas e a quantidade das mesmas.

```
create or replace view getUsersWithMoreOrdersAndHowMany as
    select count(*), "orderCustomer", "orderItemProduct" from "public"."Orders"
        join "OrderItems" on "Orders"."orderID" = "OrderItems"."orderID"
    group by "orderCustomer", "orderItemProduct"
    order by count(*) desc
    limit 5;
```

### 3.6.5 getUsersOrdersAndHowMany()

A view “getUsersOrdersAndHowMany()” apresenta todas as encomendas feitas por cada utilizador de um só produto e quantas vezes o encomendou.

```
create or replace view getUsersOrdersAndHowMany as
    select count(*), "orderCustomer", "orderItemProduct" from "public"."Orders"
        join "OrderItems" on "Orders"."orderID" = "OrderItems"."orderID"
    group by "orderCustomer", "orderItemProduct"
    order by count(*) desc;
```

### 3.6.6 ordersView()

A view “ordersView()” apresenta todos os dados sobre todas as encomendas feitas desde sempre.

```
create or replace view OrdersView as
    select "Orders"."orderID", "orderCustomer", "orderDate", "orderTotal", "orderQuantity", "orderStatus", "orderItemID", "orderItemProduct", "orderItemPrice"
        from "public"."Orders"
        join "OrderItems" on "OrderItems"."orderID" = "Orders"."orderID";
```

---

### 3.6.7 getOrderStatusFalse()

A view “getOrderStatusFalse” apresenta quais as encomendas que ainda não foram expedidas para a casa dos clientes, ou seja, que têm o “orderStatus” a false.

```
create or replace view getOrderStatusFalse as
  select * from "public". "Orders"
  where "orderStatus" = false;
```

### 3.6.8 getCountOrderStatusFalse()

A view “getCountOrderStatusFalse()” apresenta o número de encomendas cujo estado é “false”, ou seja, que ainda não foram expedidas.

```
create or replace view getCountOrderStatusFalse as
  select count(*) from "public". "Orders"
  where "orderStatus" = false;
```

---

## 4 Aplicação Web

### 4.1 Utilizadores

Este projeto apresenta diversos tipos de usuários, cada um com funções específicas, a fim de organizar o site de forma adequada. Estes utilizadores são:

**Administrador:** O administrador é responsável pela gestão dos produtos e dos usuários do comércio. Além disso, ele tem acesso a uma página de monitoramento com estatísticas do site.

**Comercial de Tipo 1:** Os comerciais são membros da empresa que gerência o site. Os comerciais de tipo 1 podem listar e gerenciar produtos pertencentes à empresa, realizar ações como alterar produtos, criar promoções e excluir produtos.

**Comercial de Tipo 2:** À semelhança dos comerciais de tipo 1, estes também são membros da empresa que gerem o site, mas em contraste, os comerciais de tipo 2 têm acesso a um painel de estatísticas que mostra os produtos mais vendidos, os tipos de produtos mais populares e as promoções ativas.

**Parceiros:** Os parceiros são usuários responsáveis pela gestão de seus próprios produtos. Eles podem adicionar produtos de sua autoria ao site após se registarem como parceiros, permitindo que outros usuários comprem seus produtos.

**Clientes:** A maioria dos usuários são clientes. Eles têm acesso a uma página com recomendações personalizadas, mostrando os produtos mais populares no site, os mais vendidos na semana e as melhores promoções. Os clientes também podem fazer compras no site.

Este sistema de usuários e suas respectivas funções ajuda a manter o site organizado e oferecer uma experiência personalizada aos diferentes tipos de usuários.

---

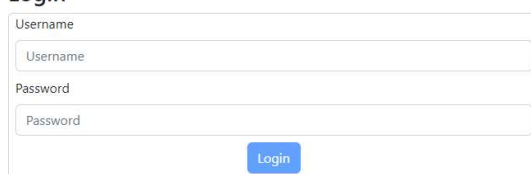
## 4.2 Autenticação de utilizadores

Para proteção de dados e gestão de encomendas, criou-se um sistema de autenticação de utilizadores.

Assim, qualquer utilizador que esteja autenticado no *website*, consegue efetuar a compra de qualquer que seja o produto disponível no site, bem como consegue acompanhar as suas compras, estejam estas ainda em fase de aquisição ou terminadas.

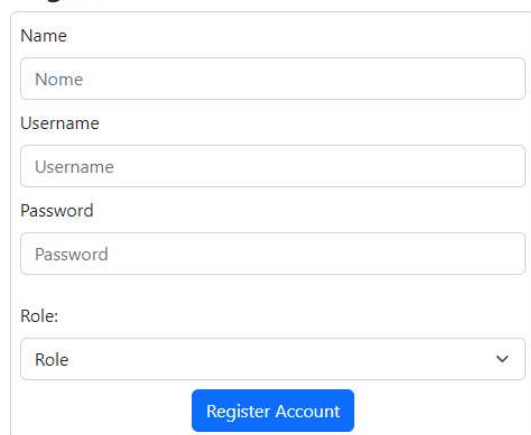
Nas imagens seguintes verifica-se quais os formulários utilizados para o login e registo de utilizadores, respetivamente.

### Login



A login form with a light gray border. It contains two input fields: 'Username' and 'Password', each with a placeholder of the same name. Below the fields is a blue button labeled 'Login'.

### Registo



A registration form with a light gray border. It contains four input fields: 'Name' (placeholder 'Nome'), 'Username' (placeholder 'Username'), and 'Password' (placeholder 'Password'). Below these is a dropdown menu labeled 'Role:' with 'Role' as the selected option and a downward arrow. At the bottom is a blue button labeled 'Register Account'.



---

### 4.2.1 Funções

Para que estes formulários sejam apresentados, criaram-se as seguintes views:

```
def register(request):
    if request.method == 'POST':
        user = {
            'name': request.POST['name'],
            'username': request.POST['username'],
            'password': request.POST['password'],
            'role': request.POST['role'],
        }
        print(user)
        result = insertUser(user['name'], user['username'],
                             user['password'], user['role'])
        print(result)
        if result:
            print('User added successfully')
        context = {}
        return render(request, 'register.html', context=context)
```

```
def generate_login_token():
    # Generate a UUID
    token = uuid.uuid4().hex
    # Return the token as a string
    return token

def login(request):
    context = {}
    if request.method == 'POST':
        user = {
            'username': request.POST['username'],
            'password': request.POST['password'],
        }
        result = getUser(user['username'], user['password'])
        if result:
            token = generate_login_token()
            request.session['login_token'] = token
            request.session['user_id'] = result
            print('User logged in successfully')
            print("User ID:", result)
            return redirect('index')
        else:
            print('Invalid credentials')
            messages.error(request, 'Credenciais inválidas')
    return render(request, 'login.html', context=context)
```

---

Como é possível verificar na figura acima, a função baseia-se num método que, dado o nome do utilizador, o *username* por ele escolhido, palavra-passe e a *role* do utilizador, adiciona-se o mesmo à base de dados.

Caso exista algum erro na criação da conta, o utilizador mantém-se na página de registo, caso contrário, é redirecionado para a página principal, já com o *login* efetuado.

Já na função de *login*, faz-se uma query à base de dados e, caso exista um utilizador com aquelas credenciais, é, à semelhança do que acontece com a criação de conta, redirecionado para a página principal. Caso não exista uma conta com as credenciais inseridas, é apresentada uma mensagem ao utilizador.

---

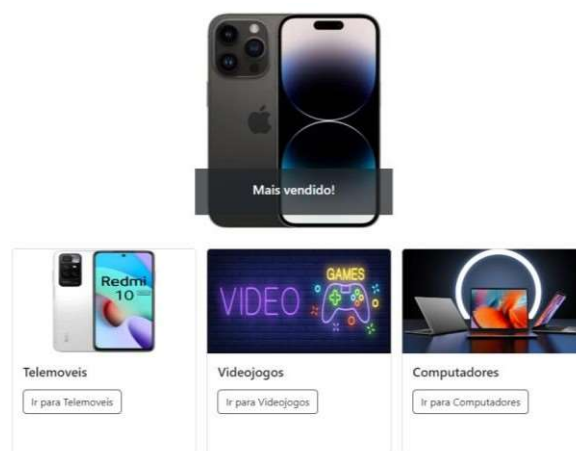
## 4.3 Navbar

Na implementação do *website* implementou-se uma barra de navegação (*navbar*) de forma a facilitar a navegação dos utilizadores pelo *website*. Cada tipo de utilizador terá acesso uma *nav* bar diferente que irá ter disponível todas as possibilidades que os mesmos podem necessitar

## 4.4 HomePage

Como dito no anteriormente, os vários tipos de utilizadores vão ter *Homepages* diferentes. Para isso, fazemos algumas verificações na *view* correspondente à página principal (Index) e, tendo em conta o tipo de utilizador, assim são apresentadas as informações pretendidas.

### 4.4.1 Utilizadores não autenticados



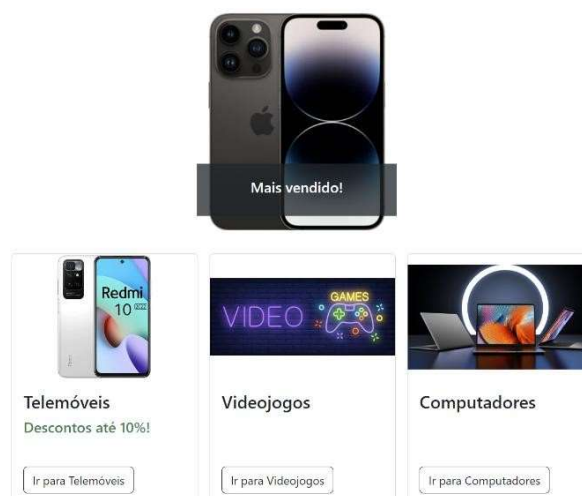
```
def index(request):
    if request.method == "GET":
        productType = list(getAllProductTypeMongoDB())
        productTypeList = []
        token = request.session.get('login_token')
        if token is not None:
            topProducts = []
            userId = request.session.get('user_id')
            userRole = int(getUserRole(request.session.get('user_id')))
            print("UserID:", userId)
            print("UserRole:", userRole)

            if userRole == 0:
                topProduct = getMostPopularProduct()
                topProductWeek = getMostPopularProductThisWeek()
                bestSale = getBiggestSale()
                #print("Most popular product Week:", topProductWeek)
                topProductStripped = topProduct[0].strip("(")
                topProductWeekStripped = topProductWeek[0].strip("(")
                print("Most popular product:", topProductStripped)
                topProductImage = getProductImageMongoDB(topProductStripped)
                topProductWeekImage = getProductImageMongoDB(topProductWeekStripped)
                bestSaleImage = getProductImageMongoDB(bestSale['productTypeID'])
                for product in productType:
                    productTypeList.append(
                        {'productTypeName': product['productTypeName'], 'productTypeImage': product['productTypeImage'], 'id': product['_id']})
                context = {'productTypeList': productTypeList,
                           "user": userRole, 'topProductImage': topProductImage, 'topProductWeekImage': topProductWeekImage, 'bestSaleImage': bestSaleImage}
```

Como é possível visualizar na imagem acima apresentada, após verificação de que o utilizador é cliente, são feitas várias queries à base de dados de modo a obter os produtos mais vendidos, quer seja de sempre, quer seja na própria semana e qual a melhor promoção atualmente disponível. A par disso, é ainda feita uma query para se apresentarem todos os tipos de produtos disponíveis no *website*.

#### 4.4.2 Clientes

Como se pode verificar, para os clientes da loja, aparecem visíveis as promoções.



### 4.4.3 Comerciais

#### Produtos mais vendidos

<b>Xiaomi Mi Mix Fold 2</b> 1530.0€ 256gb/1tb snapdragon 8 gen 1 núcleo octa 8.02 2 camera 2k + tela dobrável câmera de leica 50mp
<b>Fifa 23</b> 50€ O EA SPORTS™ FIFA 23 traz ainda mais ação e realismo do futebol para dentro de campo no Desporto Rei.
<b>GTA V</b> 50€ O jogo é simplesmente excepcional, não tenho palavras, apenas diga compre e aproveite.", "productStatus
<b>MacPro</b> 2500€ Entregue com carregador de macbook - 100% verificado e verificado pela nossa equipa de trabalhadores qualificados. Chegará a sua casa em perfeito estado de funcionamento - Garantia profissional de 12 meses 14 dias para mudar de ideias
<b>Mac Studio</b> 3200€ M1 Ultra CPU 20-core GPU 48-core Neural Engine 32-core 64 GB de memória unificada 1 TB de armazenamento SSD

[Consultar todas as encomendas](#)

#### Utilizadores com mais encomendas

<b>Utilizador: Cliente</b> Produto: Xiaomi Mi Mix Fold 2 Número de vezes que encomendou: 2
<b>Utilizador: Cliente</b> Produto: Fifa 23 Número de vezes que encomendou: 2
<b>Utilizador: Cliente</b> Produto: Mac Studio Número de vezes que encomendou: 1
<b>Utilizador: Cliente</b> Produto: MacPro Número de vezes que encomendou: 1
<b>Utilizador: Cliente</b> Produto: GTA V Número de vezes que encomendou: 1

[Consultar todas as encomendas por utilizador](#)

#### Produtos mais vendidos por Parceiro

<b>Produto: Xiaomi Mi Mix Fold 2</b> Parceiro: parceiro Número de vendas: 2
<b>Produto: MacPro</b> Parceiro: parceiro Número de vendas: 1
<b>Produto: GTA V</b> Parceiro: parceiro Número de vendas: 1

[Consultar vendas efetuadas pelo parceiro](#)

```
elif userRole == 2 or userRole == 3:
    topListUsers = []
    topListPartners = []
    listReturnedTop5MostSold = []
    listReturnedTop5MostSold.append(list(getTop5MostSoldProducts()))
    listReturnedTop5UsersMoreOrders = []
    listReturnedTop5UsersMoreOrders.append(list(getUserWithMoreOrdersAndHowMany()))
    listReturnedTop5Partners = []
    productsOfPartners = list(getProductsByPartner())
    listReturnedTop5Partners.append(list(getMostSoldProductByPartner(productsOfPartners)))
    print("Lista dos partners", listReturnedTop5Partners[0])

for products in listReturnedTop5MostSold:
    i = 1
    for product in products:
        topProductStriped = product[0].strip("")
        i += 1
        topProducts.append(
            getProductMongoDB(topProductStriped))

for products in listReturnedTop5UsersMoreOrders:
    for product in products:
        topProductStriped = product[0].strip("")
        data = topProductStriped.split(",")
        count = data[0]
        userID = data[1]
        productID = data[2]
        user = getUserByID(userID)
        productGiven = getProductMongoDB(productID)
        topListUsers.append({'count':count, 'user':user, 'product':productGiven})

for products in listReturnedTop5Partners:
    for product in products:
        topProductStriped = product[0].strip("")
        data = topProductStriped.split(",")
        productID = data[0]
        productGiven = getProductMongoDB(productID)
        topListPartners.append({'product':productGiven})
context = {'topProducts': topProducts, 'topListPartners': topListPartners,
           'topListUsers': topListUsers, "user": userRole}
```

Já nesta parte da função, fazemos referência aos comerciais. Nesta parte, pretende-se apresentar uma espécie de *dashboard* com algumas informações do *website*, assim fazem-se queries à base de dados para se obter os utilizadores com mais encomendas, os produtos mais vendidos por parceiro e os produtos mais vendidos.

---

#### 4.4.4 Parceiros

### Vendas

MacPro 7280.00€
Xiaomi Mi Mix Fold 2 7280.00€
Xiaomi Mi Mix Fold 2 1530.00€
GTA V 100.00€

```
elif userRole == 4:
    topProducts.append(list(getTop5MostSoldProducts()))
    for products in topProducts:
        i = 1
        for product in products:
            print("Product:", product)
            topProductStriped = product[0].strip("(")")
            print("Top", i, "product:", topProductStriped)
            i += 1
        topProductImage = getProductImageMongoDB(topProductStriped)
```

Quanto aos parceiros, na sua página principal, são apresentados quais dos seus produtos mais se vendem no *website*.

## 4.4.5 Administradores

### Utilizadores que utilizam o seu website

Cientes	Comerciais Tipo 1	Comerciais Tipo 2	Parceiros	Administradores
Nome: Cliente Username: client	Nome: Comercial Tipo 1 Username: ComType1	Nome: Comercial Tipo 2 Username: ComType2	Nome: Parceiro Username: parceiro	Nome: Admin Username: admin
			Nome: XPTO Username: XPTO	

```
elif userRole == 1:
    clientes = 0
    comType1 = "2"
    comType2 = "3"
    parceiro = "4"
    admin = "1"

    clientlist = []
    clientlist = list(getUsersByRole(clientes))
    print("cliente", clientlist)
    comType1list = []
    comType1list = list(getUsersByRole(comType1))
    print("1", comType1list)
    comType2list = []
    comType2list = list(getUsersByRole(comType2))
    print("2", comType2list)

    parceirolist = []
    parceirolist = list(getUsersByRole(parceiro))

    adminlist = []
    adminlist = list(getUsersByRole(admin))

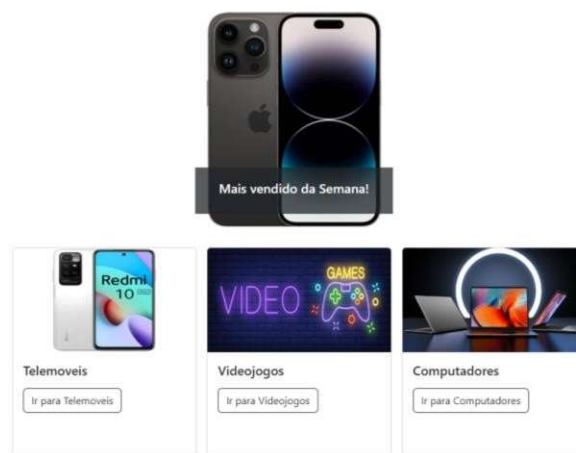
    context = {'clientlist': clientlist, 'comType1list': comType1list,
              'comType2list': comType2list, 'parceirolist': parceirolist,
              'adminlist': adminlist, "user": userRole}
```

Neste caso, os administradores terão acesso aos utilizadores que utilizam o seu *website*. Para o efeito, são feitas queries à base de dados, usando as funções especificadas previamente, de forma a obter estas informações.

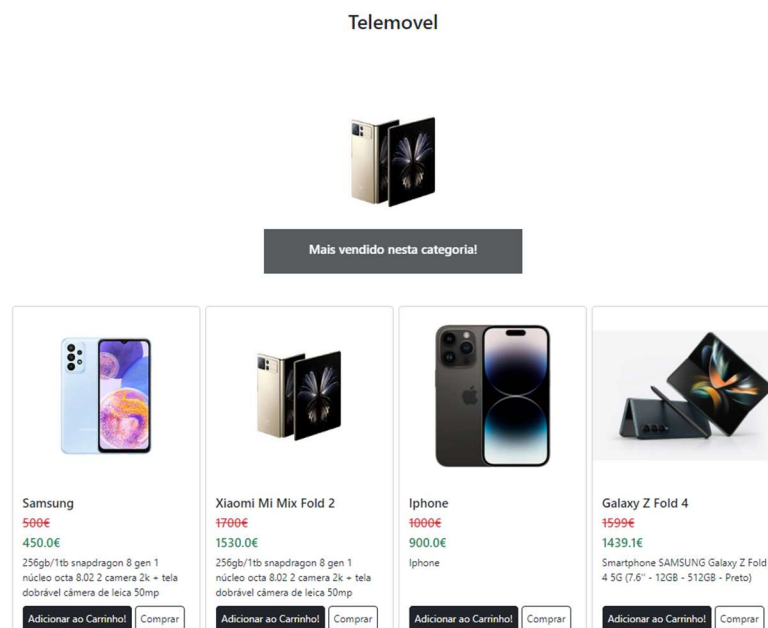
## 4.5 Listagem de produtos

Como já se referiu, para efetuar a compra de um produto, o utilizador deve estar registado e autenticado no *website*, porém, qualquer que seja o utilizador, consegue visualizar os produtos disponíveis no site (como demonstra a figura seguinte).

Esta será a página referente à apresentação dos produtos ou, *HomePage* tanto dos clientes como dos utilizadores que não estejam autenticados. É de notar que, nesta página, são apresentados os tipos de produtos existentes na loja. Para aceder aos produtos em si, deve clicar no cartão com o tipo de produto.



Acedendo ao tipo de produto pretendido, a página de apresentação será semelhante, porém, apenas estarão disponíveis produtos relacionados entre si e com o tipo previamente selecionado.





Por fim, para aceder a um produto e ver as suas características com mais pormenor, basta clicar na imagem do mesmo, o que o irá redirecionar para a página do produto que é apresentado individualmente:



Para listar os produtos, agrupados por tipo de produto, fez-se uso da *view* `listProducts`, onde se faz uma query à base de dados de forma a receber quais os tipos de produtos existentes para venda, apresentando-os na página. É de notar que se usa o id do tipo de produto, passado no url da página.

```
def listProducts(request):
    if request.method == 'GET':
        product_type_id = request.GET.get('id')
        product_type = get_one_product_type_mongo_db(product_type_id)

        products = list(get_products_by_type_mongo_db(product_type_id))
        print(products)
        product_list = []
        for product in products:
            product_list.append(
                {'product_name': product['product_name'], 'product_image': product['product_image'], 'product_price_start': product['product_price_start'], 'id': product['_id'], 'product_description': product['product_description']})
        print(product_list)
        context = {'product_type': product_type, 'product_list': product_list}
    return render(request, 'listProducts.html', context=context)
```

Para terminar a visualização de produtos, temos a apresentação individual do produto, juntamente com as suas características mais próprias, cuja função é a seguinte:

```
def product(request):
    if request.method == 'GET':
        product_id = request.GET.get('id')
        product = get_product_mongo_db(product_id)
        print(product)

    return render(request, 'product.html', {'product': product, 'id': product_id})
```

---

## 4.6 Criação de produtos

Criaram-se funções para que os utilizadores com as devidas permissões o consigam fazer. Assim, as funções criadas são relativas à criação de tipos de produtos. A primeira, resume-se à inserção de um nome e imagem para o tipo de produtos que se deseja adicionar.

### 4.6.1 Adicionar tipo de produto

Para adicionar um novo tipo de produto, o utilizador vê a seguinte página:

#### Adicione um novo tipo de produto

Tipo de produto:

Imagem do tipo de produto:

```
def addProductType(request):
    if request.method == 'POST':
        imageToConvert = io.BytesIO(request.FILES['productTypeImage'].read())
        image_data = imageToConvert.read()
        image_name = request.FILES['productTypeImage'].name

        base64_image_data = base64.encodebytes(image_data)

        productType = {
            'productTypeName': request.POST['productTypeName'],
            'productTypeImage': image_name,
        }
        result = insertProductTypeMongoDB(
            productType['productTypeName'], productType['productTypeImage'])
        if result:
            print('Product Type added successfully')
            print(productType['productTypeImage'])
            decoded_image_data = base64.decodebytes(base64_image_data)
            with open("BD2Project\\BD2app\\static\\img\\"+image_name, "wb") as fh:
                fh.write(decoded_image_data)

        context = {}
        return render(request, 'addProductType.html', context=context)
```

Como é possível visualizar na figura acima apresentada, a função consiste num método *POST* em que, dados o nome e a imagem alusivos ao tipo de produto, adiciona-se o mesmo à base de dados.

## 4.6.2 Adicionar um novo produto

Já para adicionar novos produtos à loja, deve indicar-se qual o nome, o tipo a que se associa, a quantidade disponível, o preço unitário, uma pequena descrição e, por fim, uma imagem do mesmo.

### Adicione um novo produto

Nome:

Tipo de Produto:

Selecione o tipo de produto ▼

[Para adicionar um novo tipo produto, clique aqui!](#)

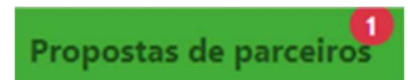
Quantidade:

Preço:

Descrição:

Imagem do produto:

Nenhum ficheiro selecionado



```
def addProduct(request):
    context = {}
    if request.method == 'POST':
        imageToConvert = io.BytesIO(request.FILES['productImage'].read())
        image_data = imageToConvert.read()
        image_name = request.FILES['productImage'].name
        base64_image_data = base64.encodebytes(image_data)

        product = {
            'productName': request.POST['productName'],
            'productImage': image_name,
            'productPriceStart': request.POST['productPriceStart'],
            'productType': request.POST['productType'],
            'productQuantity': request.POST['productQuantity'],
            'productDescription': request.POST['productDescription'],
        }
        result = insertProductMongoDB(
            product['productName'], product['productImage'], product['productPriceStart'], product['productType'], product['productQuantity'], product['productDescription'])
        if result:
            print('Product added successfully')
            print(product['productImage'])
            decoded_image_data = base64.decodebytes(base64_image_data)
            with open("B02app\\static\\img\\"+image_name, "wb") as fh:
                fh.write(decoded_image_data)

    elif request.method == 'GET':
        productType = list(getAllProductTypeMongoDB())
        productTypeList = []
        for product in productType:
            productTypeList.append(
                {'productTypeImage': product['productTypeImage'], 'productTypeImage': product['productTypeImage'], 'id':product['_id']})
        context = {'productTypeList':productTypeList}

    return render(request, 'addProduct.html', context=context)
```

À semelhança do que acontece na criação de novos tipos de produto, é feito uma nova adição à tabela dos produtos com os devidos campos preenchidos relativamente às informações do produto.

## 4.7 Alteração de produtos

Para alterar produtos à loja, na página de exibição, já aparecem alguns campos preenchidos, como o caso do nome, preço, quantidade e descrição, pelo que os utilizadores devam sempre especificar o que querem alterar. Para isso, basta apagar o conteúdo do campo e inserir o que desejam.

### Altere o produto

Nome:

Tipo de Produto:

[Para adicionar um novo tipo produto, clique aqui!](#)

Quantidade:

Preço:

Descrição:

Imagem do produto:

☐ Produto Ativo

```
def updateProduct(request):
    productupdate_id = request.GET.get('id')
    producttypelist = []
    if request.method == 'GET':
        productupdate = getProductMongoDB(productupdate_id)

        producttype = list(getAllProductTypeMongoDB())
        for product in producttype:
            producttypelist.append(
                {'producttypename': product['producttypename'], 'producttypeimage': product['producttypeimage'], 'id': product['_id']})
    else:
        imageToConvert = io.BytesIO(request.FILES['productimage'].read())
        image_data = imageToConvert.read()
        image_name = request.FILES['productimage'].name
        base64_image_data = base64.encodebytes(image_data)

        productStatusCheckBox = request.POST.get('productStatus')
        if productStatusCheckBox is not None:
            productStatus = True
        else:
            productStatus = False
        print("PRODUCT STATUS", productStatus)
        productupdate = {
            'productname': request.POST['productname'],
            'productimage': image_name,
            'productpricestart': request.POST['productpricestart'],
            'producttype': request.POST['producttype'],
            'productquantity': request.POST['productquantity'],
            'productdescription': request.POST['productdescription'],
            'productstatus': productStatus,
        }
        result = updateProductMongoDB(
            productupdate_id, productupdate['productname'], productupdate['productimage'], productupdate['productpricestart'], productupdate['producttype'], productupdate['productquantity'], productupdate['productdescription'], productupdate['productstatus'])
        if result:
            decoded_image_data = base64.decodebytes(base64_image_data)
            with open('static/img/' + image_name, 'wb') as fh:
                fh.write(decoded_image_data)
            print("Product update successfully")
            return redirect('listAllProducts')
        else:
            print("Error in product update")

    return render(request, 'updateProduct.html', {'productupdate': productupdate, 'id': productupdate_id, 'producttypelist': producttypelist})
```

---

## 4.8 Encomendas

De forma a aceder às encomendas, o utilizador deve encontrar-se autenticado no *website*. Tendo isso em conta, ao aceder à parte das encomendas, o utilizador encontra a seguinte página:

### As minhas encomendas

Encomenda: 6 Preço Total: 100.00€ Data: 2023-09-01	Em Processamento <a href="#">Detalhes</a>
Encomenda: 5 Preço Total: 1530.00€ Data: 2023-09-01	Em Processamento <a href="#">Detalhes</a>
Encomenda: 4 Preço Total: 7280.00€ Data: 2023-08-31	Aceite <a href="#">Detalhes</a>

Sendo que a *view* relativa a essa página é apresentada na imagem a seguir. Esta consiste numa simples query à base de dados, recebendo assim uma lista com todas as encomendas do cliente em questão. São apresentados o produto comprado, o preço do mesmo e a data de compra.

```
def listOrders(request):
    page = 'listOrders.html'
    context = {}
    if request.method == 'GET':
        token = request.session.get('login_token')
        if token is not None:
            page = 'listOrders.html'
            userId = request.session.get('user_id')
            print(userId)
            orders = list(getOrdersByCustomer(userId))
            ordersList = []
            for element in orders:
                for character in element:
                    data = character.split(',')
                    # print("Data:", data)
                    productID = data[1]
                    price = data[4].strip("(")
                    date = data[3]
                    print("ProductID:", productID, "Price:", price, "Date:", date)

                    product = getProductMongoDB(productID)
                    ordersList.append({'productName': product['productName'], 'price': price, 'date': date})
            context = { 'ordersList': ordersList }
    return render(request, page, context=context)
```

O utilizador tem ainda possibilidade de visualizar informações mais detalhadas das suas encomendas, acedendo a “detalhes”. Após isso, a página de apresentação será deste género:

## Encomenda número: 4

Nome do Cliente: Cliente

Montante total pago: 7280.00€

Data de encomenda: 2023-08-31

Estado:

**Aceite**

### Produto encomendado: Xiaomi Mi Mix Fold 2

Montante pago: 1530.00€

Vendedor: Parceiro

### Produto encomendado: Fifa 23

Montante pago: 50.00€

Vendedor: XPTO

### Produto encomendado: Mac Studio

Montante pago: 3200.00€

Vendedor: XPTO

### Produto encomendado: MacPro

Montante pago: 2500.00€

Vendedor: Parceiro

---

## 4.9 Carrinho

E finalmente implementamos um carrinho no *website*.

### O meu Carrinho

<b>Iphone</b> 900.0€ Iphone	<a href="#">Eliminar</a> <a href="#">Comprar</a>
<b>MacPro</b> 2500€ Entregue com carregador de macbook. - 100% verificado e verificado pela nossa equipa de trabalhadores qualificados. Chegará a sua casa em perfeito estado de funcionamento - Garantia profissional de 12 meses 14 dias para mudar de ideias	<a href="#">Eliminar</a> <a href="#">Comprar</a>

[Comprar Todos](#)

#### 4.9.1 Funções do carrinho

A função de adicionar um produto ao carrinho faz uso do método *POST*, adicionando à tabela correspondente ao carrinho as informações do produto e do utilizador que pretende comprar o produto.

```
def addCart(request):  
    page = "cart.html"  
    context={}  
    if request.method == 'POST':  
        token = request.session.get('login_token')  
        if token is not None:  
            page = "cart.html"  
            productID = request.GET.get('id')  
            userID = request.session.get('user_id')  
            result = addProductToCart(userID, productID)  
            if result:  
                print('Product added to cart successfully')  
                page = "cart.html"  
        else:  
            return redirect('login')  
    return render(request, page, context = context)
```

A função de remover produtos do carrinho, tem por base passar o id do carrinho e, após isso, faz uso de uma função previamente definida e, apaga os itens existentes no carrinho.

```
def removeCart(request):
    page = "cart.html"
    context={}
    if request.method == 'POST':
        token = request.session.get('login_token')
        if token is not None:
            page = "cart.html"
            cartID = request.GET.get('id')
            print(cartID)
            result = removeProductFromCart(cartID)
            if result == True:
                print('Product removed from cart successfully')
                page = "cart.html"
            else:
                return redirect('login')
        return render(request, page, context = context)
```

Por fim, a função de listar o conteúdo do carrinho baseia-se num método *GET* onde se vão buscar à base de dados, todos os produtos existentes no carrinho correspondente ao cliente em questão. O id do cliente é passado no *url* da página.

```
def listCart(request):
    page = "cart.html"
    context={}
    if request.method == 'GET':
        token = request.session.get('login_token')
        if token is not None:
            page = "cart.html"
            userId = request.session.get('user_id')
            print(userId)
            cartlist = list(getCartByUserMongoDB(userId))
            productCount = {} # dictionary to store product counts
            cartListProducts = []
            for cartItem in cartlist:
                product_id = cartItem['productID']
                cartID = cartItem['_id']
                str = getProductMongoDB(product_id)
                cartListProducts.append({'cartID':cartID, 'productName':str['productName'], 'productImage':str['productImage'], 'productPriceStart':str['productPriceStart'],
                'productPriceEnd':str['productPriceEnd'], 'id':str['_id'], 'productDescription':str['productDescription'], })
            print(productCount)
            print(cartListProducts)
            context={'productCount':productCount, 'cartListProducts':cartListProducts}
        else:
            return redirect('login')
    return render(request, page, context = context)
```



---

## 4.10 Promoções

Para evitar problemas, as paginas seguintes só podem ser acedidas por alguns utilizadores . Assim, tendo por base a página apresentada a seguir, os utilizadores podem controlar quais os produtos a que querem adicionar uma nova promoção.

Promoções existentes

Telemoveis 10%	Eliminar Promoção
-------------------	-------------------

Adicionar nova promoção

Para adicionar uma nova promoção, deve clicar-se no respetivo botão, obtendo a seguinte página:

Selecione o Tipo de Produto em que pretende aplicar a promoção

Telemoveis	Selecionar
Videojogos	Selecionar
Computadores	Selecionar

Por fim, a página final para adição de uma nova promoção é a seguinte

Insira a nova promoção para Telemoveis

Nova Promoção

Valor percentual

Criar Promoção

As funções seguintes foram criadas de forma a auxiliar o bom funcionamento das páginas anteriormente referenciadas.

```

def createSale(productTypeID, sale):
    doc_id = str(uuid.uuid4())
    sale = {
        '_id': doc_id,
        'productTypeID': productTypeID,
        'sale': sale
    }
    if mongo_client['Sales'].insert_one(sale):
        return True
    else:
        return False

def getSales():
    sales = mongo_client['Sales'].find()
    return sales

def deleteSaleByProductType(productTypeID):
    if mongo_client['Sales'].delete_one({'productTypeID': productTypeID}):
        return True
    else:
        return False

def deleteSaleByID(saleID):
    if mongo_client['Sales'].delete_one({'_id': saleID}):
        return True
    else:
        return False

def getProductTypeBySale(saleID):
    sale = mongo_client['Sales'].find_one({'_id': saleID})
    return sale['productTypeID']

def getSaleByProductType(productTypeID):
    sale = mongo_client['Sales'].find_one({'productTypeID': productTypeID})
    if sale is not None:
        return sale
    else:
        return None

def getBiggestSale():
    sales = mongo_client['Sales'].find().sort('sale', pymongo.DESCENDING)
    return sales[0]

```

Na imagem anteriormente apresentada, consegue visualizar-se as várias funções relativas às promoções. Para a criação de uma promoção, faz-se um *insert* na tabela **Sales** da promoção que se pretende realizar. Para se obter as promoções, os tipos de produto consoante a promoção, a promoção por tipos de produto e a maior promoção, faz-se uma query à tabela **Sales** consoante a necessidade especificada. Por fim, para se eliminarem registos na tabela, faz-se uso do *delete*.

---

## 5 Conclusão

Com a realização deste trabalho conseguimos expandir os nossos conhecimentos vistos que muito do que usamos na criação deste site envolveu muita pesquisa externa

Conseguimos de forma bem-sucedida completar muitas das implementações, embora algumas tenha se provado mais desafiantes que outras conseguimos concluir o trabalho da maneira que imaginamos.

É também de mencionar que foi a primeira vez que trabalhamos com algumas das ferramentas como Python e Django, aumentando assim os nossos conhecimentos e ao mesmo tempo criando um desafio.

Em suma, pode dizer-se que foi um projeto bastante desafiador e que superou algumas das expectativas do grupo, tanto em matéria de base de dados como em produção de código em outras linguagens.