Trabalho 2

Angular Framework and Django REST Framework

Professor: Hélder Zagalo.

Trabalho realizado por:

Pedro Duarte, nº.97673

Daniel Figueiredo, nº 98498

Eva Bartolomeu, nº 98513

Índice

Conteúdos

1. Introdução

No âmbito da cadeira de Tecnologias e Programação Web, foi-nos proposto o desenvolvimento de uma aplicação web com o uso de determinadas tecnologias, o Angular para a implementação do lado do cliente, e o Django REST Framework para a elaboração do server.

O tema da aplicação web é o mesmo que o do outro projeto, é um website que tem como principal objetivo, por parte dos clientes, fazer compras de produtos online de um hipermercado.

Os objetivos do trabalho são:

- Desenvolvimento de uma aplicação web (lado cliente e servidor);
- Trabalhar coordenadamente com Angular Framework e Django REST Framework;
- Elaborar interfaces gráficas com o Angular;
- Fornecer REST Web Services, que permitem a execução de todas as funcionalidades do back-end (obtenção de dados, inserção de dados, atualização de dados, eliminação de dados, …);
- Implementação de um sistema de autenticação, utilizando a autenticação do Django Rest Framework;
- Utilizar uma arquitetura de software real n-tier, fazer o deployment para as diferentes camadas da aplicação.

2. Conceito e funcionalidades da aplicação

A aplicação web desenvolvida, serve para facilitar as compras do supermercado YourMarket, sem ser necessário se deslocar para a loja física, ou esperar em filas. O cliente com esta app, pode encomendar todos os produtos do supermercado online, de forma rápida, que irá ser entregue na sua morada.

Nesta aplicação é possível encontrar certas funcionalidades, que se diferenciam consoante o tipo de utilizadores que estão autenticados no website.

Começamos com o **utilizador anónimo**, que é o cliente que não realizou qualquer tipo de autenticação, isto é, não fez login. Este apresenta as seguintes funcionalidades:

- Executar o login/registo;
- Explorar produtos;
- Explorar produtos por categorias;
- Adicionar produtos no carrinho de compras, com determinada quantidade;

Passamos para o **utilizador autenticado**, é um cliente que teve de realizar o login, que contém as suas próprias credenciais. Possui todas as funcionalidades do utilizador anónimo mais:

- Observar informações e o estado das suas encomendas;
- Gerenciar suas informações de pagamento.

Os próximos utilizadores, já não são clientes, e têm uma plataforma diferente do cliente. Primeiramente temos o membro **staff**, dispõe dos serviços:

- Visualizar painel de acesso com informações estatísticas da loja (número de contas staff, número de contas de clientes, número de encomendas, número de produtos e produtos mais vendidos);
- Gerenciar produtos (observar, ativar/desativar , adicionar);
- Observar/adicionar categorias;
- Gerenciar as encomendas de todos os utilizadores (ver, mudar o estado/cancelar);
- Observar as contas dos clientes da app;
- Gerenciar as encomendas de um determinado utilizador(ver, mudar o estado/cancelar).

Por último, temos o **administrador**, com as mesmas ferramentas do staff, e ainda:

- Gerenciar as contas dos membros do staff (visualizar, eliminar e adicionar).

3. API (Django REST Framework)

A DRF é uma framework do Django que serve para a criação de APIs REST. Deste modo, consegue-se que a troca de dados entre frontend e back-end seja feita através da API.

3.1 - Autenticação

Para a autenticação de utilizadores no nosso sistema foram usados tokens de autenticação, próprios do Rest Framework do Django que permite obter e verificar um token no login do utilizador.

Quando um utilizador faz login, o token fica guardado na localstorage e todos os pedidos à API que necessitam de autorização são intercetados de forma a incluir esse token, sendo assim garantida a autenticidade dos utilizadores.

3.2 - Autorização

Após feito o login, nem todos os utilizadores terão acesso às mesmas coisas, como por exemplo, caso o login seja feito com uma conta de cliente normal, poderá fazer encomendas na loja assim como ver o seu carrinho e as suas encomendas, caso seja feito login como membro do staff ou admin, então poderão controlar melhor o fluxo dos clientes, produtos e encomendas dos clientes.

No backend foram criadas três classes que as classes que correspondem ás views podem herdar de forma a requisitar um determinado nível de autenticação para aceder.

3.3. - Views

É nas views que estão presentes todos os pedidos disponíveis à API, sendo estes pedidos feitos através de vários métodos tais como: GET, PUT, POST e DELETE.

```
class StaffView(AdminAuthBaseView, generics.ListAPIView):
  queryset = models.User.objects.filter(is_staff=True, is_superuser=False).all()
  serializer_class = serializers.UserSerializer

  def post(self, request: HttpRequest, format=None):
    data = {k:v for k, v in request.data.items()}

    user = models.User.objects.create(**data)
    user.set_password(data['password'])
    user.is_active = True
    user.is_staff = True
    user.save()
    return Response(self.serializer_class(user).data)
```

3.4 - Serializers

Os serializers permitem que dados complexos, como conjuntos de consultas e instâncias de modelo, sejam convertidos em tipos de dados nativos do Python que podem ser facilmente renderizados em JSON, XML ou outros tipos de conteúdo.

Estes permitem fazer verificações, tratar da informação e definir os campos a retornar e em que formato. Foi criado pelo menos um por cada modelo, havendo também em alguns casos a necessidade de diferenciar o formato dos dados para leitura e para escrita.

```
def put(self, request: HttpRequest, format=None, **kwargs):
    id = kwargs['pk']
    try:
        product = models.Product.objects.get(id=id)
    except models.Product.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = serializers.CreateProductSerializer(product, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

4. Frontend (Angular Framework)

Todo o frontend da nossa aplicação foi feito através da tecnologia Angular.

O Angular é uma framework que possui uma arquitetura baseada em componentes, ou seja, são baseadas em classes que contêm a lógica dos dados e que está associada com um template HTML. O Angular utiliza também *typescript*.

- Componentes: views e lógica proveniente
- Data Binding: dados persistentes
- Services: obter todos os dados provenientes da API
- Routing: Navegação dentro da interface

4.1 - Componentes

As componentes são a UI mais básica de uma aplicação Angular. As aplicações Angular contêm uma árvore de componentes.

As aplicações Angular possuem uma componente raiz que vai conter a lógica que permite que várias componentes se interliguem entre si e deste modo possam comunicar através do *data binding*.

4.2 - Data Binding

De modo, a termos trocas de dados entre os templates e a lógica dos dados foi utilizado, maioritariamente, o two-way binding, permitindo através da diretiva ngModel a mudança de dados automática entre o modelo de dados e a view, combinando a utilização de Property-binding com event-binding.

```
  <select name="" class="form-control form-select" [(ngModel)]="sort" (change)="sorting()">
    <option value="default" selected>Default</option>
    <option value="-price">Price: Desc</option>
    <option value="price">Price: Asc</option>
    <option value="-name">Name: Desc</option>
    <option value="name">Name: Asc</option>
  </select>
</div>

<hr class="my-5">

<div>
  <h4>Filters</h4>

  <div class="form-group mt-2 mb-4">
    <p><b>Search</b></p>
    <input type="text" class="form-control" id="textSearch" [(ngModel)]="name" (keyup)="update_search(name)" />
  </div>

  <div class="form-group mt-2 mb-4">
    <p><b>Category</b></p>

    <select name="" class="form-control form-select" [(ngModel)]="check" (change)="change_category()">
      <option *ngFor="let cat of categories" value="{{cat.id}}">{{ cat.name }}</option>
    </select>
```

De forma a haver troca de informação entre componentes pai e componentes filho decidimos utilizar decoradores @Input() associados a property binding, o que permitiu facilmente fazer um fluxo de dados.

4.3 - Services

A utilização de serviços foi essencial, para se poder separar a parte lógica das componentes e, posteriormente, injetar as componentes com estes serviços, permitindo que as componentes possam ter os comportamentos devidos.

Todos os pedidos que eram feitos à API através dos serviços, eram interceptados por um interceptor, que tinha o papel de introduzir o token de autenticação nos headers destes pedidos.

4.4 - Routing

Para lidar com a navegação de uma página para a outra, utilizamos o Angular Router. O Router permite esta navegação, interpretando uma URL do navegador como uma instrução para alterar a visualização.

```
path: '', component: AppComponent,
children: [
  {path: '', component: IndexComponent},
  {path: 'product', children: [
    {path: '', component: ProdutsClient},
    {path: ':id', component: SingleComponent}
  ]},
  {path: 'cart' , component: ViewCartComponent},
  {path: 'order', children: [
    {path: '', component: OrdersCliComponent},
    {path: ':id', component: OrderDetailsComponent}
  ]},
],
```

5. Execução da aplicação

5.1 Execução da aplicação em localhost

Para correr a aplicação web, terá de navegar para a pasta ***./frontend***, e executar o comando: ***./run.sh***. Este comando executa um script que irá instalar as dependências para a execução e ainda irá correr o website.

De modo a que a app tenha todas as funcionalidades disponíveis, também é necessário correr o REST Web Service. Basta entrar no folder ***./api*** e efetuar o comando: ***./run.sh***, instalando assim todas as ferramentas indispensáveis, e posteriormente, executa também a API.

Na raiz do projeto, também temos um script que irá correr estes dois scripts.

Poderá visualizar a aplicação web em: http://localhost:4200/ .

5.2 Visualizar o deployment da aplicação

O deployment da aplicação web foi realizado na plataforma ***netlify***, o qual podemos visualizar no seguinte url: https://upbeat-yonath-acbae2.netlify.app/ .Decidimos fazer nesta plataforma e não na sugerida (heroku), pois achamos que o netlify aparenta ser mais intuitivo, e apresenta um processo de deployment mais simples.

Já o deployment do REST Web Service foi publicado no sistema ***pythonanywhere*** , como podemos ver no endereço: https://pedrodld536.pythonanywhere.com/ .

5.3 Credenciais dos utilizadores

Credenciais de acesso dos clientes:

- username: cliente
- password: cliente

Credenciais de acesso dos staffs:

- username: staff
- password: staff

Credenciais de acesso dos admins:

- username: *admin*
- password: *admin*

6. Conclusão

Em termos de conclusão, pensamos ter atingido os objetivos propostos a fazer e pensamos também ter feito bom uso das tecnologias lecionadas pelo docente.

Com a realização deste projeto foi possível criar um bom ambiente de grupo e sobretudo aprender de modo mais aprofundada as tecnologias de Angular e também da DRF(Django REST Framework), que nos permite trabalhar com API de modo a ser mais fácil a transição de dados entre frontend e back-end.