Universidade de Aveiro

Tecnologias e Programação Web

Arcade Battle

Relatório Final

Autores :
84793 Daniel Nunes
84921 Rafael Direito

Professor :
Hélder Zagalo

5 de Junho de 2019

Conteúdo

Lista de Códigos

1 Introdução

O Arcade Battle é um sistema que permite aos seus utilizadores realizarem o tratamento de artrites através de jogos lúdicos. Desta forma, utilizando uma Leap Motion, conseguimos recolher quais os gestos que um determinado utilizador está a executar, avaliá-los e utilizá-los como comandos em jogos arcade.

A aplicação aqui em análise surge como suporte para a gestão de todo este sistema. Nesta, os médicos podem introduzir novos pacientes no sistema, adicionar-lhes gestos e avaliar qual o seu desempenho ao longo de todo o tratamento.

A plataforma permite, também, gerir quais o médicos que podem aceder ao sistema e a adição de novos jogos ao mesmo.

Tendo em conta a escalabilidade da solução desenvolvida, optámos por desenvolver uma aplicação composta por 3 módulos separados, que podem ser executados em 3 ambientes distintos:

- Interface Gráfica do Médico;
- REST API;
- Base de Dados.

2 Angular

Angular é uma plataforma open-source, liderada pela Google, de aplicações web com front-end baseado em TypeScript. No segundo projeto, Angular foi a tecnologia utilizada para a interface e, para tal, foram implementadas as seguintes funcionalidades providenciadas pelo Angular:

- **Components**: Usámos components para todas as views e lógica associada às mesmas;
- **Data Binding**: Data binding foi usado para, principalmente, mostrar dados persistentes;
- **Directives**: Usámos directives para providenciar comportamento adicional a elementos do DOM;
- **Service**: Todos os dados vindos do servidor foram chamados obtidos de um service;
- **Dependency Injection**: Injeção do service em diversos componentes, por exemplo;
- **Routing**: Toda a navegação dentro da plataforma é feita através de routing;
- **Bootstrap**: Bootstrap é utilizado como ferramenta de estilo para a interface;
- **Observables**: Usados, principalmente, para lidar com programação assíncrona.

2.1 Alterações face ao primeiro projeto

2.1.1 Two-way Binding e Event Binding

Enquanto que, em Django, a parte lógica associada a uma view apenas é atualizada recorrendo a métodos adicionais, em Angular, através do two-way binding, conseguimos facilmente atualizar uma variável de acordo com a view.

Devido à simplicidade providenciada por este data binding, foi fortemente utilizado em casos como, por exemplo, os formulários. Os Django Forms utilizados no primeiro projeto acabaram por ser substituídos com recurso à utilização do ngSubmit (event binding) e do two-way binding, como foi referido anteriormente.

```
<form (ngSubmit)="addGame(photo)" enctype="multipart/form-data">
  <div class="card-body card-block">
    <div class="row form-group">
      <div class="col col-md-1 col-sm-2 ">
        <label for="id_name">Name:</label>
      </div>
      <div class="col-12 col-md-5 col-sm-4">
        <input [(ngModel)]="name" type="text" name="name"
            placeholder="Insert game's name" class="form-control"
            required="" id="id_name">
      </div>
    </div>
```

Código 1: Formulário para adicionar um novo jogo (parcial)

2.1.2 NgIf e NgFor

No que toca a diretivas, o angular apresenta vários tipos sendo que as mais utilizados no nosso projeto foram, sem dúvida, as diretivas estruturais, mais especificamente, o NgIf e o NgFor. Estas permitem facilmente substituir as template tags provenientes do Django e renderizar de uma forma naturalmente fluída o template necessário, assim como o seu conteúdo.

```
<tr *ngFor="let d of doctors">
  <td (click)="access_patient(d)">{{ d.first_name }} {{ d.last_name
     }}</td>
  <td class="text-center" (click)="access_patient(d)">{{ d.specialty
     }}</td>
  <td class="text-center" (click)="access_patient(d)">{{ d.city }}</td>
  <td class="text-center" (click)="access_patient(d)">{{ d.birth_date
     }}</td>
  <td class="text-center" (click)="access_patient(d)">{{ d.nif }}</td>
  <td (click)="access_patient(d)">{{ d.username }}</td>
  <td *ngIf="(userType == 'admin')">
    <button type="button" class="btn-sm btn-danger" data-toggle="modal"
       data-target="#deleteModal" (click)="loadInfo(d)"> Remove </button>
  </td>
</tr>
```

Código 2: Listagem de todos os médicos na plataforma

2.1.3 Service e Dependency Injection

Por fim, como já foi referido, todas as chamadas à REST API são feitas a partir de um service criado na aplicação Angular. Para que isto aconteça, é utilizada Dependency Injection nos componentes que precisam de dados provenientes do servidor, injetando o service nesses mesmos componentes. Através deste processo, conseguimos tornar os componentes mais sustentáveis, reusáveis e testáveis através da remoção de dependências hard coded.

```
import {ArcadebattleService} from '../arcadebattle.service';

(...)

export class AllDoctorsComponent implements OnInit {
  userType: string;
  doctors: object[];
  remove: any;

  constructor(private arcadeBattleService: ArcadebattleService, private
      location: Location) {
    this.userType = localStorage.getItem('userType');
  }

  (...)

  getAllDoctors() {
    this.arcadeBattleService.all_doctors()
        .subscribe(
            data => {
              this.doctors = data.data;
            }
        );
    $('#dtBasicExample tr:last').remove();
  }
```

Código 3: Injeção de um serviço e uso posterior do mesmo

3 Django REST Framework

A Django Rest Framework (DRF) é uma biblioteca do Framework Django utilizada para a implementação de REST APIs.

Esta permite-nos a utilização de políticas de autenticação e autorização, bem como serialização de dados provenientes de uma base de dados.

3.1 Alterações face ao primeiro projeto

3.1.1 Autenticação

O maior desafio da implementação do sistema utilizando uma REST API prende-se com o facto da autenticação de utilizadores ter de ser efetuada ao nível da API, em vez de na interface através da qual os utilizadores acedem ao sistema.

Para tal, recorremos a AuthTokens.

Desta forma, sempre que o utilizador envia os seus dados de login, este recebe um Token de Autenticação que deve acompanhar todos os pedidos à API que forem efetuados posteriormente.

O Token fica guardado do lado do cliente e é eliminado aquando o logout do mesmo, sendo, então, criado um novo token para cada sessão.

```
@csrf_exempt
@api_view(["POST"])
@permission_classes((AllowAny,))
def login(request):

    username = request.data.get("username")
    password = request.data.get("password")

    if username is None or password is None:
        return Response({'error': 'Please provide both username and
            password'}, status=HTTP_400_BAD_REQUEST)
```

```
user = authenticate(username=username, password=password)

if not user:
    return Response({'error': 'Invalid Credentials'},
        status=HTTP_404_NOT_FOUND)

token, _ = Token.objects.get_or_create(user=user)

#Logs
logging.info(" User: " + username +" has logged in with auth_token: "
    + token.key)

u = User.objects.get(username=username)

data = {'token': token.key,
        'user_type': get_user_type(username),
        'first_name': u.first_name,
        'last_name': u.last_name,
        'email': u.username,
        }

try: data['photo_b64'] =
    Person.objects.get(user=User.objects.get(username=username)).photo_b64
except: data['photo_b64'] = ""

return Response( data, status=HTTP_200_OK)
```

Código 4: Tratamento da autenticação de um utilizador na plataforma

3.1.2 URLs e Views

Na criação da REST API foram disponibilizados novos métodos, de forma a permitir o envio de todos os dados necessários para a visualização de informação, bem como para a introdução e atualização da mesma.

Desta forma, disponibilizámos métodos GET, POST e DELETE.

Tendo em conta a separação lógica e a melhor organização do código, criámos um novo ficheiro python (queries.py) cuja única função é a interação com a base de dados e a serialização dos dados necessários.

O ficheiro views.py, por sua vez, invoca métodos do queries.py, avaliando sempre se o utilizador tem autorização para efetuar o pedido em causa.

3.1.3 Novos módulos de Python

Para o bom funcionamento da REST API foi necessário instalar os seguintes módulos:

- *djangorestframework*
- *django-cors-headers*
- *mysqlclient*

3.1.4 Acesso a Bases de Dados Remotas

A REST API desenvolvida está apta para interagir com bases de dados remotas. Uma vez que a base de dados remota utilizada era uma base de dados MySQL, foi necessário instalar o módulo mysqlclient para que fosse possível interagir com ela.

Isto permite-nos facilmente criar uma aplicação de 3 camadas o que traz claras vantagens para a escalabilidade da mesma.

Seguem-se as configurações necessárias para permitir o acesso à base de dados remota. Estas devem ser feitas no ficheiro *settings.py*:

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'arcadebattle_db',
    'USER': "arcadebattle",
    'PASSWORD': "arcadebattle",
    'HOST': "arcadebattle-db.cysle9zhxbja.us-east-1.rds.amazonaws.com",
    'PORT': "3306",
  }
}
```

Código 5: Configurações para acesso à base de dados remota

4 Notas Finais

Git: https://github.com/danielfbnunes/arcadebattle_angularapp

REST Api: http://rdireito.pythonanywhere.com

Interface do médico: http://ec2-100-24-34-88.compute-1.amazonaws.com/

Olhando em retrospectiva, consideramos que todos os objetivos relativos a este trabalho foram atingidos. Como nota pessoal, achamos que, sem sombra de dúvidas, django é sem dúvida uma tecnologia muito apetecível para este tipo de trabalhos sendo que o angular se torna um pouco mais trabalhoso.

Em suma, concluímos que, com este trabalho, os conhecimentos de django e angular foram, sem qualquer dúvida, melhorados e que a capacidade de pesquisa e autonomia foram, também elas, bem desenvolvidas, na tentativa de obter soluções face aos problemas encontrados. É ainda de realçar que, com esta cadeira, aprendemos tecnologias que certamente serão relevantes no nosso percurso profissional.