Tecnologias e Programação Web

Professor Hélder Zagalo

# Online Music Player

Diana Miranda, 107457

José Gameiro, 108840

Miguel Figueiredo, 108287

DETI

Universidade de Aveiro

23 de Dezembro 2023

# Índice

# 1. Introdução

No âmbito deste segundo trabalho prático, concentramo-nos na aplicação prática das Frameworks Angular e Django REST para conceber uma aplicação web, seguindo a linha do primeiro projeto. Este relatório visa oferecer uma visão concisa da implementação realizada, destacando a estrutura adotada, as principais funcionalidades incorporadas na aplicação, bem como as orientações essenciais para aceder remotamente e localmente tanto ao backend quanto ao frontend.

# 2. Tecnologias Utilizadas

Conforme solicitado no enunciado, a estrutura frontend é elaborada mediante o uso do framework Angular, ao passo que o backend, encarregado dos serviços Web, é concretizado através da utilização do framework Django REST.

Refinámos os conhecimentos adquiridos em sala de aula, aprimorando-os por meio de pesquisas para integrar tecnologias mais avançadas. Entre essas inovações, destaca-se a implementação de autenticação por meio do Json Web Token Authenthication através do package djangorestframework-simplejwt, representando uma melhoria substancial na segurança e funcionalidade do sistema.

# 3. Web Services

Através da Django Rest Framework foram definidos REST Web Services que irão ser consumidos pelo cliente. Nesta secção não nos iremos dedicar a enumerar todos os endpoints (e views correspondentes) que os compõem, mas iremo-nos dedicar à explicação do processo de autenticação com JWT e seu impacto na proteção de determinados endpoints. O processo de autenticação com JWT baseia-se em 2 tokens: um token de acesso de tempo de vida curto e um token de refresh de tempo de vida longo que serve para obter um novo access token.

O processo de autenticação é feito através do consumo dos seguintes endpoints:

- **ws/auth/sign-in** - não só gera o token de acesso como injeta no token o atributo para no cliente verificar os privilégios do utilizador (is_superuser). Por motivos de segurança, o access token vai ser retornado na resposta enquanto o refresh token vai ser passado através de um http-only cookie que não é passível de ser acedido através de client-side scripts fornecendo assim um maior nível de segurança. Quando chegar ao cliente, o access token vai ser guardado no local storage (para facilitar a implementação) enquanto o refresh token irá ser incluído nos requests automaticamente visto que está guardado em cookies. Esta dualidade de métodos de armazenamento também significa que ambos os tokens não podem ser roubados explorando o mesmo tipo de vulnerabilidades;
- **ws/auth/sign-up** – segue a mesma linha que o sign-in mas cria um utilizador novo e gera um access e refresh token novos (visto que o utilizador não existia). Aqui também se destaca a implementação do método create() do UserSerializer que permite fazer o hashing da password quando se cria o utilizador através do serializer;
- **ws/auth/refresh** - responsável pela criação de um novo access token a partir de um refresh token válido (passado nos cookies). É de notar que não há nenhum endpoint para atualizar o refresh token e se o refresh token expirar o utilizador irá ter de voltar a fazer o login na aplicação (e consequentemente consumir o endpoint ws/auth/sign-in) para obter tanto um novo access como refresh token;

Estando tudo isto definido, o django irá averiguar através do token (ou a inexistência do mesmo) se o user está autenticado e se é administrador. Com recurso à anotação **@permission_classes** e às permissões **IsAuthenticated** e **IsAdminUser** conseguimos definir a autorização do acesso às várias views. Qualquer request feito pelo cliente para uma view marcada com **IsAuthenticated** tem de incluir o token nos seus headers tendo esse token de corresponder a um superuser no caso da permissão **IsAdminUser**.

# 4. Web App

O propósito central de nossa aplicação reside na criação de um sistema de streaming de música online. Neste ambiente, os utilizadores têm a oportunidade de explorar uma ampla variedade de músicas disponíveis na aplicação, podendo criar playlists, obter informações detalhadas sobre artistas e álbuns, e organizar uma fila de reprodução personalizada. A flexibilidade é concedida aos utilizadores para ouvir as músicas na ordem de sua preferência, proporcionando uma experiência de audição verdadeiramente personalizada.

## 4.1. Funcionalidades

- Permite ouvir músicas;
- Possui uma página principal com músicas agrupadas por géneros musicais;
- Permite procurar músicas simultaneamente pelo nome, artista, banda, álbum e género;
- Possui páginas com as informações dos artistas/bandas;
- Possui páginas com as informações dos álbuns;
- Permite a criação de várias playlists e ordenação de músicas dentro de cada playlist;
- Permite a criação de uma queue;
- Permite colocar *like* em músicas;
- Como administrador, existe uma AdminPage que permite adicionar, editar e remover Músicas, Géneros de Músicas, Álbuns, Artistas e Bandas.

## 4.2. Serviços

Todos os serviços desenvolvidos permitem abstrair os componentes da tarefa de obtenção de dados ao servidor. Todos eles irão, portanto, proporcionar o consumo dos endpoints dos webservices disponibilizados através da Django Rest Framework, entre outras coisas como algumas funções de utilidade.

Os serviços desenvolvidos para as entidades album, artist, band, genre e playlist são responsáveis pelas operações básicas de CRUD (Create, Read, Update and Delete). Em cada caso, estes serviços permitem criar, ler, editar ou eliminar objetos correspondentes. Para além disto,

é de notar que o serviço das playlists também permite ordenar as diferentes músicas de cada playlist.

O serviço relacionado à entidade music não suporta apenas as operações CRUD para músicas, mas também oferece funcionalidades de pesquisa, permitindo encontrar músicas com base em títulos, artistas, bandas, álbuns ou géneros correspondentes a uma string fornecida. Além disso, gere operações como dar like ou dislike numa música e manipula a adição, remoção e recuperação de músicas na fila de reprodução (queue).

O serviço de performer concentra-se em fornecer acesso a todos os performers (banda ou artista) na base de dados, recuperar o nome de um performer por meio do ID e obter detalhes específicos de um performer.

Por último, o serviço de autenticação (auth) gere todas as operações relacionadas à autenticação do utilizador como por exemplo o login e signup de um utilizador. Nos pedidos efetuados ao servidor destaca-se o armazenamento da resposta do pedido (access token) em localStorage e a utilização de "**credentials: include**" no pedido fetch para o http-only cookie referido anteriormente ser guardado automaticamente. Para além disso, o serviço possibilita verificar o estado de um utilizador: verificar o seu user_id, verificar se o utilizador está logged in e verificar o seu grau de privilégios (neste caso é só verificar se é superuser). Todas estas funcionalidades são atingidas através do decode no cliente do JWT referido anteriormente.

## 4.3. Componentes

No nosso projeto, desenvolvemos um total de 22 componentes, sendo que maior parte destes apresentam código referentes a uma página em si e ao seu funcionamento. No entanto, existem outros componentes que são reutilizados em várias páginas diferentes.

O componente about-us apresenta um código muito simples pois este apresenta a página about us que contém apenas informações estáticas e não é necessário fazer nenhuma chamada ao web service desenvolvido.

Os componentes add-edit-album, add-edit-artist, add-edit-band, add-edit-genre e add-edit-music, contêm as funcionalidades, de um utilizador que seja administrador, de adicionar e/ou editar uma das seguintes entidades:

- Álbum;
- Artista;
- Banda;
- Género;
- Música.

Como se pode deduzir pelo nome estes componentes são reutilizados tanto para a adição como edição das várias entidades. Estes componentes utilizam os serviços criados para cada entidade para conseguirem ter acesso às suas informações, modificarem os dados da mesma ou para adicionarem um novo elemento à base de dados.

O componente admin, através da diretiva routerLink, contém ligações para os outros componentes onde se pode realizar operações de criar ou editar para cada entidade e visualizar os elementos presentes na base de dados para as entidades referidas acima. Esta página só fica visível a um utilizador caso este tenha uma role de admin.

Os componentes albums, artists, bands, genres e musics são também páginas para utilizadores com role de admin (verificado através da variável isSuperUser do AuthService), em que se disponibilizam todos os elementos que se encontram na base de dados que pertençam a uma das entidades referidas acima. Para além disto também contém duas funcionalidades que são a de atualizar os dados e de eliminar um elemento, sendo que se a opção de atualizar os dados for escolhida é redirecionado para um dos componentes add-edit-[entidade].

O componente artist-details contém código referente aos detalhes de um artista e pode ser acedido através do nome de um artista que se encontre em uma música da home page, nos detalhes de uma playlist, na fila de músicas ou ainda nas páginas do admin de mostrar todos os artistas e bandas presentes na base de dados. Esta página disponibiliza os detalhes de um artista como o seu nome, a sua descrição e, no caso de ser uma banda, os membros da banda, no qual,

o nome do membro também redireciona para a página dos detalhes desse membro. Também apresenta os álbuns do artista ou da banda, bem como as músicas de cada um e é possível reproduzir as músicas só de um artista ou só de um álbum. Isto é possível devido ao componente playbar que será mencionado e detalhado mais adiante. Esta página oferece também as funcionalidades de dar ou remover o like a uma música, adicionar uma música à fila de músicas, criar uma playlist e adicionar uma música a uma playlist. No entanto, estas últimas funcionalidades só estão disponíveis se o utilizador estiver autenticado (cortesia do AuthService).

O componente auth apresenta a página de login/signup. Caso o utilizador queira efetuar login apresenta um formulário com apenas dois campos o username e a password, sendo verificado se o username e a password estão de acordo com os dados guardados na base de dados (armazenamento dos tokens abstraído pelo AuthService). No caso de o utilizador querer efetuar signup terá de preencher um formulário com quatro campos (email, username, password e confirmar password). Quando o formulário é submetido é feito a verificação de todos os valores introduzidos e caso estejam todos de acordo com as contraints definidas na base de dados o utilizador é registado e redirecionado para o home (armazenamento dos tokens novamente abstraído pelo AuthService).

O componente error-display é usado nos vários componentes em que existem formulários para submeter dados como é o caso do login ou das páginas de administrador. Em termos visuais, este componente é um simples modal que mostra os erros que um utilizador possa ter cometido na submissão de um formulário. Estes erros estão guardados em cada um dos componentes que possui formulários e são passados para o componente error-display através do @Input decorator.

O componente homepage apresenta o código desenvolvido para a página inicial, nesta é possível visualizar todas as músicas que se encontram na base de dados, agrupadas por género (com máximo de 5 géneros por página - caso o número de géneros seja superior aparece um elemento no final da página que permite a navegação para outras páginas que contenham os restantes géneros). Tal como o componente artist-details, para cada música também é possível dar ou remover like numa música, criar uma playlist e adicionar uma música e adicionar músicas

à queue de músicas. Estas funcionalidades também só são disponiblizadas ao utilizador caso o utilizador esteja logged in (informação proporcionada pelo AuthService). Existe também uma barra de pesquisa que permite a pesquisa de músicas pelo nome da música, do artista/banda, do álbum e do género mantendo a regra de só puder mostrar cinco géneros por página caso o resultado de pesquisa retorne muitos dados.

O componente navbar encontra-se presente em quase todas as páginas, visto que, facilita a navegação do utilizador pelo nosso website. Esta apresenta as ligações:

- Home – link para o componente homepage;
- About Us – link para o componente about-us;
- Playlist - link para o componente playlist (mais detalhado em diante). Só é acessível caso o utilizador tenha efetuado login/signup, caso contrário irá ser redirecionado para a página do login/signup;
- Song Queue - link para o componente queue-list (mais detalhado em diante). Também só é acessível caso o utilizador tenha efetuado login/signup.
- Log In – link para o componente auth;
- Log Out - permite ao utilizador fazer log out da sua conta. Só aparece esta opção caso o utilizador esteja autenticado, sendo depois redirecionado para a página de login/signup.
- Administrator Panel – link para o componente admin que só é mostrado ao utilizador caso tenha efetuado login e tenha o role de admin (verificado através da variável isSuperUser do AuthService).

O componente playbar é um componente existente nos seguintes componentes: homepage, artist-details, playlist-details e queue-list. Este apresenta funções que permitem ao utilizador reproduzir música no website mostrar o progresso da música e avançar ou retroceder para uma nova música. Tal como referido no componente artist-details, são usadas as funções deste componente para permitir ao utilizador reproduzir músicas que pertençam apenas a um determinado artista/banda, a um determinado álbum, a uma playlist e músicas que estão presentes na queue de músicas.

O componente playlist contém o código que o nosso grupo desenvolveu para mostrar as playlists existentes e algumas informações sobre estas, como o nome dela, o utilizador que a criou, o número de músicas que contém e uma opção para a eliminar. Ao ser clicado numa playlist o utilizador é redirecionado para a página dos detalhes de uma playlist à qual corresponde o componente playlist-details, em que apresenta as músicas de uma determinada playlist, oferecendo a possibilidade de dar ou remover like nas mesmas ou até reordenar as músicas arrastando-as (a nova ordem definida persiste na base de dados).

Por último, temos o componente queue-list, que apresenta as músicas que foram adicionadas à fila de músicas. Também é possível dar ou remover like numa música que esteja presente na fila, remover uma música em específico ou remover todas as músicas que se encontrem nesta.

## 4.4. App Component

Devido à sua importância para a nossa aplicação o componente principal app tem um capítulo dedicado a ele. Como estamos a utilizar JWT para autenticação é necessário lidar com a expiração dos tokens e com a incorporação do access token em cada um dos pedidos feitos aos web services.

Com esse objetivo em mente, neste componente criou-se um interceptor com recurso a uma técnica do tipo monkey patching que intercepta cada fetch request e adiciona processamento antes e depois da call do mesmo (não podíamos utilizar HttpInterceptor's do angular porque este apenas interceptam o httpClient). Neste caso, definimos apenas comportamento adicional para antes da chamada do fetch original tratar de uma possível expiração do acess token e adicionar um access token válido a cada request. Portanto, antes de cada fetch adicional é verificado se existe um token e se ele está a menos de um minuto e meio de expirar. Se for o caso, é feito um pedido com o httpClient (para não ser interceptado também) para se renovar o access token antes de fazer a query pretendida/original.

No entanto, há componentes onde são feitos vários pedidos ao servidor. Por isso, para não ser feita uma tentativa de renovação do access token por cada pedido, utilizamos um processo de controlo de acesso com recurso a uma variável booleana e uma promise partilhada para assim que um pedido de refresh seja feito os vários "fetches" fiquem à espera da mesma resposta (e do novo access token – através do operador await todos os "fetches" ficam à espera que a mesma promise seja fulfilled). No entanto, caso o refresh token esteja expirado as informações do utilizador serão limpas (authService.clean()) e ele será redirecionado para a página de login para se autenticar.

### 4.5. Guards

Um dos objetivos da nossa webapp era disponibilizar algumas das nossas funcionalidades a qualquer utilizador (esteja ou não autenticado), bloqueando algumas delas para utilizadores autenticados ou utilizadores com um maior grau de privilégios (superuser).

Com esse propósito em mente, foram utilizados routeGuards para prevenir que utilizadores naveguem para partes da aplicação para à qual não tem a autorização necessária. Foram definidos portanto 2 guards:

- auth.guard.ts - verifica se o utilizador está logged in com recurso ao AuthService que valida o access token atual. Caso não passe na validação, o utilizador será redirecionado para a página de autenticação.
- Superuser.guard.ts - verifica se o utilizador está logged in e se tem privilégios de administrador com recurso ao AuthService que valida o token e extrai atributo is_superuser do mesmo. Caso a validação não seja bem sucedida, o utilizador será novamente redirecionado para o componente de autenticação.

### 4.6. Detalhes da Implementação a destacar

A utilização do Angular como framework para o desenvolvimento do frontend ajudou bastante, visto que este permite a reutilização de componentes já desenvolvidos e também a criação de serviços que executavam as chamadas ao backend.

Relativamente à reutilização de componentes, a nossa web app contém alguns elementos que são necessários em outras páginas como é o caso da playbar, ou seja, a barra de reprodução de música. Esta barra era necessária estar presente nas seguintes páginas:

- Página inicial com as músicas agrupadas por género;
- Páginas dos detalhes de um artista/banda com os respetivos álbuns;
- Página com as músicas de uma determinada playlist;
- Página com todas as músicas presentes na queue;

Por isso nós decidimos criar um componente à parte e incorporá-lo nas páginas necessárias. No entanto, foi necessário adaptar a cada página pois algumas apresentavam diferentes modos de reprodução de música como é o caso da página das playlists em que só podia reproduzir músicas que estivessem presentes na playlist, ou como o caso da página dos detalhes de um artista em que é necessário permitir a reprodução de músicas pertencentes a um artista em específico ou a um único álbum.

Também reutilizámos o componente Navbar para a maior parte das páginas desenvolvidas visto que era um elemento que necessitava de estar presente em todas elas, e a sua incorporação nas outras páginas ajudou-nos a simplificar o código da página e a evitar a repetição de código.

Um outro componente que reutilizámos noutras páginas foi o error-display, pois este indica os possíveis erros que um utilizador poderá fazer na submissão de um formulário. É de destacar a utilização de @Input para passar informação do componente pai para o componente filho. Este componente foi necessário nos seguintes componentes que continham formulários:

- Componentes do administrador para adicionar ou editar informações de uma entidade;
- Da página do login/signup.

É ainda de focar a utilização de guards para validar a autorização dos vários componentes, a integração de JWT ao longo da aplicação utilizando não só o localStorage mas também http-only cookies e a criação de um interceptor usando monkey patching para em cada fetch adicionar

um access token válido ou redirecionar o utilizador para o componente de autenticação (se o refresh token expirou).

## 5. Como correr a aplicação

- **Localmente:**

  - Backend:

    *cd MusicPlayer/*

    *# for translation to work in ubuntu (I don't know about MAC)*

    *sudo apt-get install gettext # (translation on the django website)*

    *python3 -m venv venv*

    *source venv/bin/activate*

    *pip install -r requirements.txt*

    *django-admin compilemessages*

    *python3 manage.py makemigrations MusicPlayer*

    *python3 manage.py migrate*

    *python3 manage.py createsuperuser python3 manage.py runserver*

  - Frontend:

    *cd MusicPlayerClient/*

    *npm install*

    *ng serve*

- **Remotamente**
  - https://deploy-music-player.vercel.app
  - http://dianarrmiranda.pythonanywhere.com/
  - https://github.com/Migas77/online_music_player

- **Credenciais**
  - Admin:
    - Email: admin@gmail.com
    - Username: admin

        - Password: ADMINtpw12.
        - ('.' incluído na password)
    - User normal sem privilégios de admin:
        - Email: user1@gmail.com
        - Username: user1
        - Password: User123.
        - ('.' incluído na password)

(apenas necessário username e password para login)

# 6. Conclusão

Em conclusão, este relatório expôs o percurso que o nosso grupo teve no desenvolvimento da nossa aplicação web, usando as Frameworks Angular e Django REST. Destacámos a estrutura adotada, as principais funcionalidades implementadas e fornecemos orientações cruciais para o acesso remoto e local ao front-end. Este trabalho não consolidou apenas os conhecimentos adquiridos, mas também evidenciou a capacidade de integrar de forma eficaz estas ferramentas na conceção de soluções web robustas e funcionais.