Sistemas Distribuídos

Sistema de Gestão de Artistas de Rua 2.0

Rafael Martins (48596), João Sousa (51497)

Universidade de Évora - 20 de janeiro de 2024

Introdução

No âmbito da cadeira de Sistemas Distribuídos, foi-nos proposto pelo docente da mesma, José Saias, o desenvolvimento de um um sistema para gerir as performances dos artistas de rua do Alentejo onde devemos implementar as aplicações servidor, cliente geral, e cliente administrador, nesta segunda parte teremos de alterar a localização para receber coordenadas, bem como alterar a implementação das doações, das atuações passadas e futuras e criar um sistema de autenticação.

Implementação

Após a leitura do enunciado, verificamos que necessitamos de mudar o cliente e o servidor, desta vez para a criação da lógica do trabalho, utilizou-se RESTful Web Service com Spring, como foi dado na atividade 11.2. Com isto, criou-se um cliente, um servidor, juntamente com a autenticação que é baseada em JWT, o utilizador quando se autentica, é passado um token para o mesmo, que irá utilizar para aceder a outros métodos. Para além disso, criou-se controladores para quando o cliente quiser fazer um HTTP Request, tem de estar autenticado, ou então se for algum método que só o administrador pode usar (.HasRole("ADMINISTRADOR")), o mesmo nega o acesso caso não tenha a role.

Classes

- **Client** - Classe com o cliente, chama métodos da classe **ClientService**.
- **Server** - Inicia o servidor spring.
- **Artista** - Classe com o construtor do artista, juntamente com getters e setters.
- **Atuacoes** - Classe com o construtor das atuações, juntamente com getters e setters.
- **User** - Classe com o construtor do user, juntamente com getters e setters.
- **JWTAuthorizationFilter** - Classe para criar o token JW, validá-lo e autenticar o utilizador.
- **ClientService** - Classe que tem os métodos para ligar-se ao servidor e solicitar um request HTTP aos controladores.
- **PostgresConnector** - Classe para tratar da ligação à base de dados.
- **DBService** - Classe com os métodos para alterar e consultar a base de dados.

Métodos por classe

Artista:

- ***Artista****(int id, String nome, String tipo _arte, boolean aprovado, String descricao, String imagem)* - Construtor da classe.
- ***getLocalizacao****()* - Retorna a localizacao do artista.
- ***getNome****()* - Retorna o nome do artista.
- ***getTipoArte****()* - Retorna o tipo de arte do artista.
- ***getRating****()* - Retorna o rating do artista.
- ***toString****()* - Método para manipular o print da classe artista.

Atuacoes:

- ***getId****()* - Retorna o id da atuação.
- ***getData****()* - Retorna a data da atuação.

User:

- ***User****(String username, String token, String role)* - Construtor da classe.
- ***getToken****()* - Retorna o token do utilizador.
- ***toString****()* - Método para manipular o print da classe User.

DBService:

Nesta classe, inicia-se a ligação à base de dados para depois utilizar-se nos métodos.

- ***adicionarArtista****(String nome, String tipoArte, boolean estado)* - Introduz uma query na base de dados em que passa o nome do artista e o tipo de arte, se a query não funcionar devolve uma mensagem a dizer "Artista não foi adicionado".
- ***adicionarAtuacao****(int id, float latitude, float longitude, LocalDate data_atuacao)* - Introduz uma query na BD em que passa os argumentos enviados para criar a atuação, se der algum erro na operação, retorna "Atuação não foi adicionada".
- ***procurarArtistasPorEstado****(boolean estado)* - Inicia uma array list do tipo **Artista**, introduz a query na base de dados e dentro de um ciclo while, enquanto houver artistas que tenham o estado indicado pelo utilizador, coloca-os na lista e retorna.

- ***procurarArtistasLocalArte****(String localizacao, String tipo_arte)* - Inicia uma array list do tipo **Artista**, introduz a query na base de dados e dentro de um ciclo while, enquanto houver artistas que tenham o local e a arte indicado pelo utilizador, coloca-os na lista e retorna.

- ***obterArtistas****()* - Inicia uma array list do tipo **Artista**, introduz a query e retorna todos os artista presentes na base de dados.

- ***obterLocalizacoesAtuar****()* - Inicia uma lista do tipo *String* e procura todas as localizacoes onde a coluna *atuar* da tabela **atuacoes** equivale a *true*, coloca as localizações na lista e devolve.

- ***aprovarArtistaPorId****(int id_artista)* - Seleciona o artista com o id desejado pelo utilizador e dá update para true caso não esteja.

- ***obterAtuacoesArtista****(int id_artista)* - Cria uma lista do tipo *String*, pesquisa todas as atuações do artista na BD, coloca na string e devolve.

- ***existeIdArtista****(int id_artista)* - Verifica na BD se existe o id do artista, é um método auxiliar para controlar os inputs do utilizador.

- ***existeArtista****(String nome, String tipo_arte)* - Verifica na BD se existe o nome do artista juntamente com esse tipo de arte.

- ***adicionarDonativo****(int id_artista, float valor)* - Adiciona um novo donativo ao artista com esse id, só se pode enviar doações a um artista, caso o mesmo esteja aprovado. Portanto neste método, retorna uma *String* a dizer "Donativo não foi adicionado. Artista não está aprovado!"se o artista não estiver aprovado, se tiver aprovado retorna "Donativo adicionado com sucesso!".

- ***obterDonativosArtista****(int id_artista)* - Inicia uma lista do tipo *String* e coloca os donativos que o artista recebeu caso hajam, depois retorna.

- ***alterarDescFoto****(int id_artista, String descricao, String foto)* - Dá update à imagem e à foto do artista na base de dados, retorna uma *String* a dizer "Alteração realizada com sucesso"caso altere, se houver algum erro retorna "Alteração não foi realizada".

- ***consultDescFoto****(int id_artista)* - Procura o artista na base de dados e retorna um objeto artista para depois utilizar o método **getImagem** e **getDescricao**.

- ***registarUtilizador****(String username, String password, String email)* - Tenta inserir o utilizador na base de dados, retorna true.

- ***existeUtilizador****(String username, String password)* - Verifica se já existe um utilizador, usado para o login.

- ***verificaRegisto****(String username, String email)* - Verifica se já existe um utilizador com esse nome ou email, retorna um boolean.

- ***isAdm****(String username)* - Verifica se o utilizador é administrador.

- ***atualizarTipoUtilizador****(String username)* - Verifica se o utilizador já é administrador ou não, se for retorna a dizer que já é, se não for, muda o role e retorna "Atualização feita com sucesso".
- ***artistaTemLocalizacao****(String name)* - Verifica se o artista já tem localização.
- ***atualizarLocalizacaoArtista****(String nome)* - Atualiza a localização do artista.
- ***verificarDataAtuacoes****()* - Se houver atuações com a data de hoje (LocalDate.now()), coloca-as com o estado a true, se não coloca a false.
- ***obterAtuacoesTempo****(int id_artista,int escolha)* - Retorna todas as atuações dependendo da escolha, se for 1, retorna todas as atuações passadas, se for 2 retorna a próxima atuação do artista.
- ***adicionarRating****(int id, float rating)* - Adiciona o rating ao artista, verifica o valor anterior que o artista tinha antes e calcula a nova média para atualizar o rating geral do artista.

ClientService:

Nesta classe, estão os HTTP Requests feitos para cada método da BD, assim, é o servidor que acede à base de dados pois são nos controladores que estão a ser chamados.

Tabelas criadas

```
CREATE TABLE users
(
username character varying(255) PRIMARY KEY,
password character varying(255) NOT NULL,
email character varying(255) NOT NULL,
user_role character varying(50) NOT NULL
);

CREATE TABLE artista
(
id_artista serial PRIMARY KEY,
nome character varying(255) NOT NULL,
tipo_arte character varying(255) NOT NULL,
estado boolean,
imagem character varying(255),
descricao character varying(255) NOT NULL,
localizacao character varying(255),
num_avaliacoes integer,
rating double precision
);

CREATE TABLE atuacoes
(
```

```
id_atuacao serial PRIMARY KEY,
id_artista integer REFERENCES artista(id_artista),
atuar boolean,
data_atuacao date,
coordenadas geometry(Point,4326)
);


CREATE TABLE donativos
(
id_donativo serial PRIMARY KEY,
id_artista integer REFERENCES artista(id_artista),
username character varying(50) REFERENCES users(username),
valor double precision NOT NULL
);
```

Conclusão

Com a realização deste trabalho ficamos a compreender melhor como é que podemos implementar uma REST API bem como funciona os controladores. Uma das maiores dificuldades foi a autenticação pois não estávamos a perceber como é que funciona um JWT, depois de melhor entender o mesmo, tivemos de refazer a autenticação para algo que fazia mais nexo. Em suma foi um projeto bastante interessante de realizar que certamente contribuiu para a nossa aprendizagem.