Relatório do segundo trabalho prático de Sistemas Distribuídos

Henrique Rosa - 51923
Diogo Matos - 54466
19/01/2024

Introdução

Este trabalho foi desenvolvido como resposta ao enunciado do segundo trabalho prático da unidade curricular de Sistemas Distribuídos do curso de engenharia informática. Consiste no desenho e implementação de um Sistema de Gestão de Artistas de Rua.
A nossa aplicação inclui as seguintes funcionalidades:

- Uma divisão do sistema por perfis, com:
  → Uma parte pública, em que apenas inclui o login e o registo de conta para posteriormente aceder à parte reservada;
  → Uma parte reservada ao perfil "administrador" onde este, depois de recebida a aprovação de admin, para além de fazer tudo o que se faz na parte reservada "normal", também é possível realizar a permissão (aprovação) de admin a um user, listar artistas por estado, aprovar artistas e consultar e alterar informações de um artista;
  → Uma parte reservada ao perfil "normal" onde é possível realizar registos de artistas, listar artistas por filtros (onde também é possível ver a classificação deste) ou sem eles, listar espetáculos a decorrer, listar espetáculos já feitos e futuros de um artista, doar a um artista, listar as doações a este e dar classificações a um artista.

A solução foi desenvolvida em Java no IDE IntelliJ IDEA da JetBrains.

Solução Implementada

De modo a encontrar uma solução adequada e satisfatória, fizemos primeiro uma análise dos conteúdos lecionados nas aulas, de modo a perceber que tecnologias e ferramentas nos seriam mais úteis para esta função. Acabamos por decidir que o sistema iria ser implementado utilizando:

- Rest;
- Framework Jersey;
- Acessos à base de dados PostgreSQL por meio de JDBC;

Classes

A nossa solução dispõe das seguintes classes:

- Client → Aplicação cliente, onde o cliente pode interagir com o serviço;
- MainAppServer → Representa o servidor do serviço;
- Artists → Representa um artista;
- PostGresCon → Facilita a conexão e operações em uma base de dados PostgreSQL usando JDBC;
- Donations → Representa uma doação;
- Ratings → Representa uma classificação;
- Users → Representa um user;
- ServerResource → Responsável pelo tratamento e atendimento dos pedidos da aplicação cliente;
- Management → Responsável por interações, para inserção e consulta na BD.

Métodos utilizados em cada Classe

Classe Client:

- main() → Responsável pela leitura dos inputs do cliente com a interação com o servidor, onde apresenta o menu principal;
- menuAuth() → Menu de autenticação (mais concretamente de login e registo de user);
- opMenu() → Responsável por fazer o tratamento da opção que o user escolher no menu;
- newUser() → Faz um pedido POST ao servidor de modo a registar um novo user;
- login() → Faz um pedido GET ao servidor de modo a realizar a autenticação de um user já registado;
- insertArtist() → Faz um pedido POST ao servidor de modo a registar um novo artista;
- infoArtist() → Faz um pedido GET ao servidor de modo a receber a informação de um artista;
- updateArtist() → Faz um pedido GET ao servidor de modo a dar update das informações de um artista;
- listArtist() → Faz um pedido GET ao servidor de modo a receber uma lista de artistas com filtragem;
- listArtistsWithStatus() → Faz um pedido GET ao servidor de modo a receber uma lista de artistas por status;
- isInteger() → Responsável por verificar se certa string é um “inteiro”;
- replaceSpaces() → Responsável por trocar os espaços numa string por “%20” de forma a que as strings mandadas em pedidos GET sejam tratadas da forma correta, mantendo os dados na forma correta com que um user os pede;
- approveArtist() → Faz um pedido GET ao servidor de modo a aprovar um artista;
- approveAdmin() → Faz um pedido GET ao servidor de modo a aprovar(dar permissão) a um admin;
- listPerformancesLive() → Faz um pedido GET ao servidor de modo a receber uma lista de espetáculos a decorrer;
- listPerformancesPrevious() → Faz um pedido GET ao servidor de modo a receber uma lista de espetáculos já feitos por um artista;
- listPerformancesFuture() → Faz um pedido GET ao servidor de modo a receber uma lista de próximos espetáculos de um artista;
- donateArtist() → Faz um pedido POST ao servidor de modo a realizar uma doação a um artista;
- listDonations() → Faz um pedido GET ao servidor de modo a receber uma lista de doações recebidas por um artista;
- ratingArtist() → Faz um pedido POST ao servidor de modo a dar uma classificação a um artista;

Classe Users:

Classe contem apenas Construtor, Setters e Getters, responsáveis por definir e retornar o valor de:

- name → username de um user;
- email → email de um user;
- pwd → password de um user;
- role → tipo de user.

Classe Artists:

Classe contem apenas Construtor, Setters e Getters, responsáveis por definir e retornar o valor de:

- name → nome de um artista;
- typeArt → tipo de arte de um artista;
- locationLatitude → latitude de um artista;
- locationLongitude → longitude de um artista.

Classe Donations:

Classe contem apenas Construtor, Setters e Getters, responsáveis por definir e retornar o valor de:

- userID → ID do user que faz a doação;
- artistName → nome do artista que recebe a doação;
- value → valor da doação;

Classe Ratings:

Classe contem apenas Construtor, Setters e Getters, responsáveis por definir e retornar o valor de:

- name → nome do artista que recebe a classificação;
- rating → classificação de 0 a 10;

Classe MainAppServer:

- getBaseURI() → Devolve o URI onde vai ficar o sistema;
- startServer() → Cria um servidor Http Grizzly com serviço REST.

Classe PostGresCon:

- connect() → Estabelece uma conexão à BD;
- disconnect() →Termina a conexão à BD;
- getStatement() → Devolve o Statement, para poder ser usada nas Queries;

Classe Management:

- dataBase() → Cria um objeto PostGresCon pronto a ser usado para ser usado no acesso à BD;
- newUser() → Faz a inserção na BD com o user passado como argumento;
- login() → Faz a pesquisa na BD e devolve a informação do user encontrado ou então falha no login (credenciais erradas);
- insertArtist() → Faz a inserção na BD com o artista passado como argumento;
- infoArtist() → Faz a pesquisa na BD e devolve a informação de um artista;
- getUserInfo() → Faz a pesquisa na BD da informação de um user após este ser registado;
- updateArtist() → Faz o update da informação de um artista com a informação recebida como argumento acedendo à BD;
- listArtist() → Faz a pesquisa na BD pela lista de artista segundo uma filtragem;
- listArtistsWithStatus() → Faz a pesquisa na BD pela lista de artista por status;
- getFinalRating() → Responsável por fazer a pesquisa de todos os ratings recebidos por um artista e calcula a média dos ratings obtendo a classificação final;
- approveArtist() → Faz o update do status de um artista para “aprovado” acedendo à BD;
- approveAdmin() → Faz o update do status de um user com role “administrador” para “aprovado” acedendo à BD;
- listPerformancesLive() → Faz a pesquisa à BD pela lista de espetáculos a decorrer na data em que é feito o pedido por parte do cliente;
- listPerformancesPrevious() → Faz a pesquisa à BD pela lista de espetáculos já feitos por um artista;
- listPerformancesFuture() → Faz a pesquisa à BD pela lista de próximos espetáculos de um artista;
- donateArtist() → Faz a inserção na BD com a doação passada como argumento;
- listDonations() → Faz a pesquisa na BD pela lista de doações recebidas por um artista;
- ratingArtist() → Faz a inserção na BD com a classificação passada como argumento;

ClasseServerResource:

- newUser() → envia um pedido à classe Management para a criação de um user e retornando a informação deste quando bem sucedido;
- login() → retorna o resultado do pedido de consulta de credenciais de um user à classe Management, retornando a informação deste caso seja bem sucedida;
- insertArtist() → envia um pedido à classe Management para a criação de um artista;
- infoArtist() → retorna a informação de um artista passado como argumento;
- updateArtist() → envia um pedido à classe Management para update das informações de um artista passadas como argumento;
- listArtist() → retorna a lista de artistas por filtragem passadas como argumento;
- listArtistsWithStatus() → retorna a lista de artistas por status;
- approveArtist() → envia um pedido à classe Management para a aprovação de um artista passado como argumento;
- approveAdmin() → envia um pedido à classe Management para a aprovação de um admin passado como argumento;
- listPerformancesLive() → retorna a lista de espetáculos a decorrer;
- listPerformancesPrevious() → retorna a lista de espetáculos já feitos por um artista passado como argumento;
- listPerformancesFuture() → retorna a lista de proximos espetáculos de um artista passado como argumento;
- donateArtist() → envia um pedido à classe Management para a doação a um artista passado como argumento;
- listDonations() → retorna a lista de doações recebidas por um artista passado como argumento;
- ratingArtist() → envia um pedido à classe Management para a classificação a um artista passado como argumento;

Base de Dados

Segue o código sql para implementação da base de dados que utilizamos no trabalho:

```
CREATE TABLE artists (
    artistID SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE,
    typeArt VARCHAR(255) NOT NULL,
    locationLatitude VARCHAR(255) NOT NULL,
    locationLongitude VARCHAR(255) NOT NULL,
    status VARCHAR(20) NOT NULL
);

CREATE TABLE performances (
    performanceID SERIAL PRIMARY KEY,
    artistID INT,
    date DATE,
    locationLatitude VARCHAR(255) NOT NULL,
    locationLongitude VARCHAR(255) NOT NULL,
    FOREIGN KEY (artistID) REFERENCES Artists
);
```

```
CREATE TABLE users (
    userID SERIAL PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(15) NOT NULL,
    status VARCHAR(20) NOT NULL
);

CREATE TABLE donations (
    donationID SERIAL PRIMARY KEY,
    userID INT NOT NULL,
    artistID INT NOT NULL,
    value INT NOT NULL,
    date DATE NOT NULL,
    FOREIGN KEY(artistID) REFERENCES Artists,
    FOREIGN KEY(userID) REFERENCES users
);

CREATE  TABLE ratings (
    ratingID SERIAL PRIMARY KEY,
    artistID INT NOT NULL,
    rating INT NOT NULL,
    FOREIGN KEY(artistID) REFERENCES Artists
);
```

Descrição do funcionamento do programa

Os seguintes passos numerados apresentam o funcionamento do sistema:

1. Criar uma BD PostGres com as tabelas descritas em cima. (de realçar que não existe nenhuma funcionalidade para inserir performances, portanto para consulta destas deve ser inserido manualmente na BD).
2. Alterar config.properties consoante a BD criada e baseuri para uri base do serviço.
3. Compilar o projeto num IDE (como o IntelliJ) ou no terminal com o comando mvn compile.
4. Executar a classe MainAppServer.
5. Executar a classe Client.

Balanço crítico

A nossa solução ao enunciado **cumpre** quase todos os requerimentos do mesmo, ficando a faltar os bónus quanto ao servidor. De realçar também certos pormenores no trabalho:

- A questão da inserção de performances como foi falado na descrição do funcionamento do programa.
- A autorização foi implementada corretamente, mas inicialmente foi tentado da forma como foi lecionado nas aulas práticas. Como não se conseguiu e no enunciado não há nada a dizer como era suposto ser feito, foi realizado da forma como está.
- O sistema está preparado para funcionar no alunos.di.uevora.pt mas como nenhum dos elementos do grupo conseguiu frequentar a universidade ou aceder remotamente, não há uma base de dados em si feita dentro da universidade, apenas localmente nos nossos PCs.