UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA
Licenciatura de Engenharia Informática

Sistemas Distribuidos

Relatório do 2º Trabalho de Sistemas Distribuidos 2023/2024

Gustavo Dias Gomes nº48392 Bárbara Loureiro nº48469

Introdução

O SeekArtist 2.0 - Sistema de Gestão de Artistas de Rua representa uma solução de desenvolvida para administrar e facilitar a interação entre artistas de rua, users interessados nas suas performances, e as contribuições através de donativos. Desenvolvido em Java e utilizando a tecnologia JAX-RS para serviços REST, o sistema destaca-se pela sua arquitetura modular e pela eficácia na comunicação entre cliente e servidor.

Desenvolvimento

Estrutura Geral do Código:

O sistema é dividido em diversas partes, cada uma representada por um arquivo Java, e juntas, formam um sistema completo para gestão de artistas, performances, users e donativos.

Descrição dos Arquivos

Cliente_geral.java (sd.clientes):
Interface de linha de comando para interação com o user. Este arquivo gere as interações do user, processa as entradas e comunica com o servidor para realizar ações como registrar artistas, listar performances, autenticar users, entre outras.

Artista.java (sd.rest):
Define a estrutura de dados para um 'Artista'. Este arquivo é uma classe modelo que representa os atributos e métodos de um artista no sistema.

ArtistaResource.java (sd.rest):
Gere as operações relacionadas a artistas no servidor. Este arquivo define os endpoints REST para criar, modificar, e listar artistas.

DbOperations.java (sd.rest):
Contém a lógica de negócio e operações do banco de dados. Este arquivo é crucial, pois gere todas as operações do banco de dados, como autenticação de users, registo de artistas e performances, e outras operações CRUD.

Donativo.java (sd.rest):
Define a estrutura de dados para um 'Donativo'. Semelhante a 'Artista', este arquivo é uma classe modelo para representar os atributos e métodos de um donativo.

DonativoResource.java (sd.rest):
Gere as operações relacionadas a donativos no servidor. Define endpoints REST para realizar operações relacionadas a donativos.

MainAppServer.java (sd.rest):
Ponto de entrada para iniciar o servidor HTTP. Este arquivo configura e inicia o servidor HTTP, tornando os serviços REST disponíveis para os clientes.

Performance.java (sd.rest):
Define a estrutura de dados para uma 'Performance'. Este arquivo é uma classe modelo para representar performances dos artistas.

PerformanceResource.java (sd.rest):
Gere as operações relacionadas a performances no servidor. Define endpoints REST para criar e listar performances.

PostgresConnector.java (sd.rest):
Facilita a conexão com o banco de dados PostgreSQL. Este arquivo abstrai a complexidade de estabelecer conexões com o banco de dados, sendo usado por DbOperations para interagir com o banco.

User.java (sd.rest):
Define a estrutura de dados para um 'User'. Este arquivo é uma classe modelo que representa os atributos e métodos de um user do sistema.

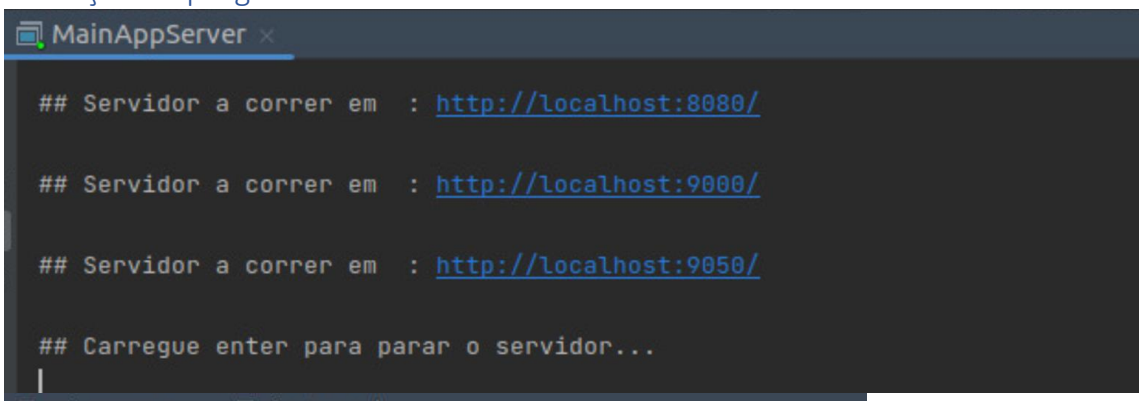
UserResource.java (sd.rest):
Gere as operações relacionadas a usuários no servidor. Define endpoints REST para autenticação, registro e listagem de usuários.

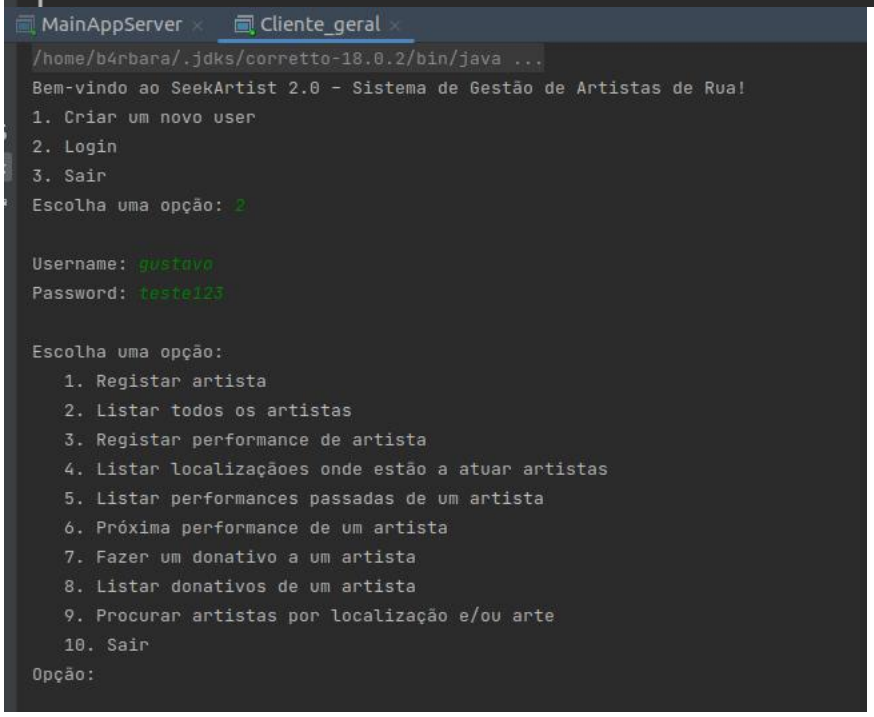
Base de Dados:

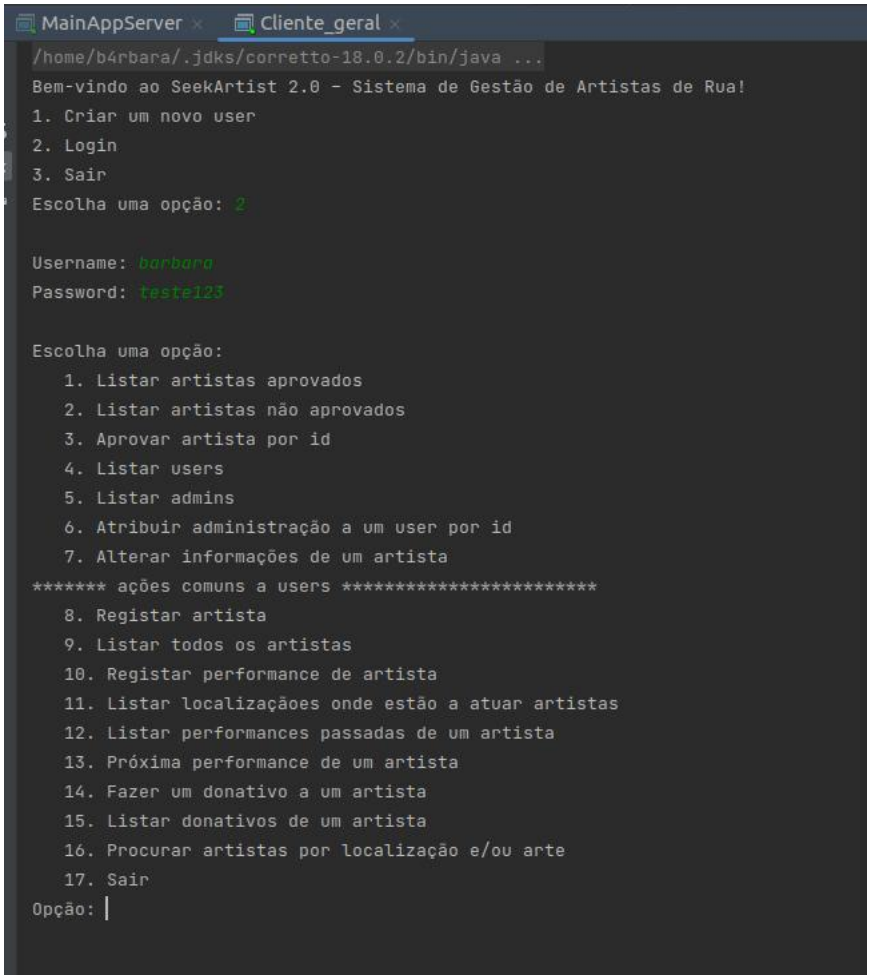
```
CREATE TABLE artistas (
   aid serial PRIMARY KEY,
   nome varchar NULL,
   tipo varchar NULL,
   estado varchar NULL
);

CREATE TABLE users (
   uid serial PRIMARY KEY,
   username varchar UNIQUE NOT NULL,
   email varchar UNIQUE NOT NULL,
   password varchar NOT NULL,
   usertype varchar NOT NULL
);

CREATE TABLE performances (
   aid int NOT NULL,
```

```
   pdata date NOT NULL,
   latitude varchar NOT NULL,
   longitude varchar NOT NULL,
   FOREIGN KEY (aid) REFERENCES artistas (aid)
);

CREATE TABLE donativos (
   aid int NOT NULL,
   valor varchar NOT NULL,
   ddata date NOT NULL,
   FOREIGN KEY (aid) REFERENCES artistas (aid)
);
```

Execução do programa







Conclusão

O projeto apresenta uma estrutura bem organizada, claramente separando as preocupações entre interação do user, lógica de negócios, operações de banco de dados e definições de modelo. Cada arquivo tem um propósito específico e contribui para a funcionalidade geral do sistema. A escolha de usar JAX-RS e PostgreSQL é adequada para um projeto de serviço REST, oferecendo flexibilidade e escalabilidade. Utilizamos o IDE Intelliji para executar o projeto.

Conseguimos implementar os objetivos propostos no enunciado com exceção aos objetivos bónus. Estamos cientes porém que o trabalho podia sofrer algumas melhorias como, uma maior validação de entrada dos inputs e um melhor tratamento da segurança.