Project Documentation - Spring Boot Application: Group 18

Talha Doğu Celal Kekeç Oğuzhan Ömer Taşkın

1 Introduction

This document provides an overview of the Java Spring Boot project including the architecture, technologies used, and setup instructions. The project exposes a REST API using OpenAPI specification and uses MariaDB for data persistence.

2 Project Overview

The backend for the game 'Catan'. The project provides fundamental authentication features, specifically: registration, login, password resetting, and password encryption, game functionalities: creating a game, running it and saving scores at the end. There are multiple entities with complex relationships among them. For detailed database inspection check the CENG453_ER.pdf.

3 Architecture

Describe the overall architecture and key components:

- Spring Boot - REST API, business logic
- MariaDB - Database
- OpenAPI - API specification
- Deployment architecture (e.g. Docker containers)
- Junit Jupiter API - Test Tools

4 Key Technologies

List the main technologies, frameworks, and tools used:

- Spring Boot

- Spring Data JPA
- OpenAPI/Swagger
- MariaDB
- Maven
- Docker

4.1 Software dependencies

The key dependencies used in the project are:

- **Spring Web** - For building RESTful web services. Provides RESTful routing and request handling.
- **Spring Data JPA** - Simplifies data access layer and integration with JPA.
- **MariaDB Driver** - JDBC driver for connecting to the MariaDB database.
- **SpringDoc OpenAPI** - For generating OpenAPI documentation for the REST APIs. Provides Swagger UI.
- **Lombok** - Utility library to reduce boilerplate code like getters/setters.
- **Spring Boot Test** - Helper libraries for testing Spring components.
- **Java Mail** - For sending emails like password reset. Provides utilities for SMTP mail sending.
- **Junit** - For applying unit tests.

5 REST API

There are 10 REST API endpoints, each serving a specific purpose:

- **/player:** Used to retrieve a player from the database using Session-key as a parameter.
- **/player/allPlayers:** Used to retrieve all players from the database.
- **/player/register:** Used to register a player to the database if there is no existing player with the same username or email address.
- **/player/login:** Used to log in a player if there exists a registered user with the same username and password. If user logs in successfully, the random session-key for the user is generated and assigned to him.

- **/player/reset-password:** Used when the user wants to change their password. By providing an email registered in the system as input, this function sends an email to the provided email address, and a reset-key is generated and set for the user with the same email address.
- **/player/change-password:** The second step of the password change involves the user checking their email to obtain the reset-key. They then provide this key, along with their new password, to this function. After executing this function, the password is changed, and the new password is set in the database for the user.
- **/score:** Used to save a score to the database by providing the username and score points as input.
- **/score/lastWeek:** Used to retrieve ordered scores from the last 7 days.
- **/score/lastMonth:** Used to retrieve ordered scores from the last 30 days.
- **/score/allTime:** Used to retrieve ordered scores from all time.
- **/game/playerMove:** Used by players to execute movements: building, rolling a dice, ending a turn.
- **/game/createSinglePlayer:** Used to create a game instance with a type of single player.
- **/game/botBuild:** Automated function for bots to execute building algorithms.
- **/game/allGames:** Returns all game instances in the database.
- **/game/delete:** Delete a game instance by game id.
- **/game/turn:** Get the current turn of the game.
- **/game/joinExistingGame:** Join the existing multiplayer game instance.
- **/game/createTradeOffer:** Creates a trade offer with number of resources proposed and demanded.
- **/game/createMultiPlayer:** Used to create a game instance with a type of multiplayer.
- **/game/cheat:** Used to allocate extra resources for the player.
- **/game/getGame:** Used to get game from the database using its id.
- **/game/getGameEvents:** Used to get game events from the game using game id.
- **/game/botCount:** Used to get number of bots in the game.
- **/game/deleteTradeOffer:** Used to delete trade offer from the game.

6 Database Design

The database schema can be outlined to show entities like:

- User
- Score
- Game
- Gameboard
- Settlement
- Road
- Tile
- Card
- PlayerCardDeck

7 Testing

Important reminder: Before executing tests, change database mode from 'update' to 'create-drop'. Since it may clash and cause database integrity issues. Also be aware of that it will reset the database and so will delete all entries in it. There are total of 39 unit tests to assess the following functionalities:

- Overridden equalTo functions for the entities Player and Score.
- Tests for Game entity class that test card distribution and resource allocation for buildings.
- Testing JPA functions, including finding a Player by username, email, session-key, reset-key, and checking existence using username and email. Additionally, finding Scores after a specific date and checking the order of the scores are also tested.
- Testing JPA functions for the Game: testing Create Read and Delete operations.
- Testing service methods, including registration, login, email sending, password changing. Additionally, obtaining weekly, monthly and all times scoreboard with proper ordering is also tested.
- RestAPI tests that can be executed manually using SwaggerUI.
- Frontend interface to visually test and play the game and also SwaggerUI support to check the game.