Row Match Backend Project *Muhammed Orhun Gale*

Before running

1) Works on 8080 port
2) Requires Redis service on 6379
3) Requires MySQL on 3306 → Needs root user and its password
4) Can be changed from application.properties

API

1) User Progress
   a) **api/user/CreateUserRequest**
      Request body format → { “username”: (string), “password”: (string) }

   b) **api/user/AuthenticateUserRequest** → Returns JWT for further authentication
      Request body format → { “username”: (string), “password”: (string) }

   c) **api/user/UpdateLevelRequest**
      Request body format → {}
      Requires “Authorization” header

2) Tournament
   a) **api/tournament/EnterTournamentRequest**
      Request body format → {}
      Requires “Authorization” header

   b) **api/tournament/GetRankRequest**
      Request body format → {“ID”: (long), “tournament”: (long)}
      Requires “Authorization” header

   c) **api/tournament/GetLeaderboardRequest**
      Request body format → {“group”: (long)}
      Requires “Authorization” header

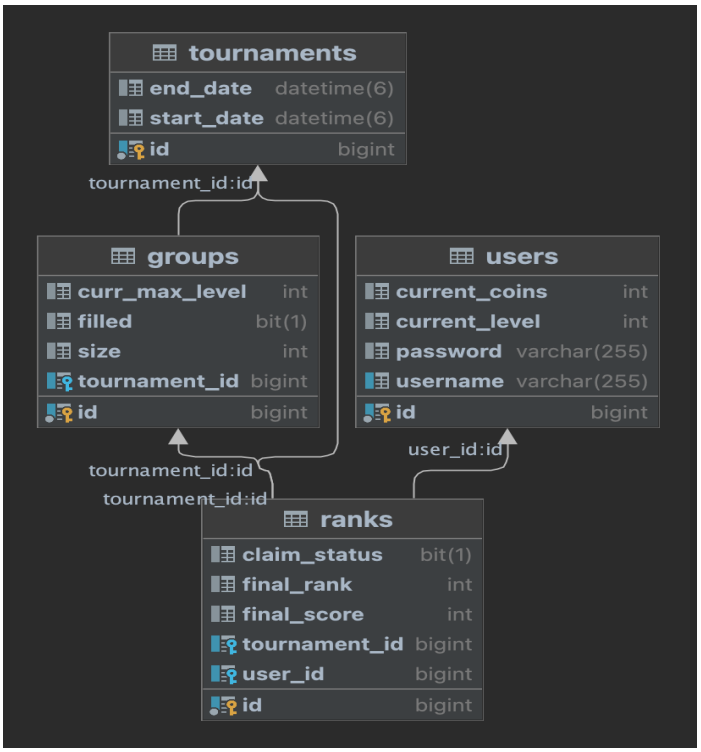
   d) **api/tournament/ClaimRewardRequest**
      Request body format → { “tournament”: (long) }
      Requires “Authorization” header

e) **api/tournament/ManuelTournamentStartRequest**
→ Starts tournament for testing purposes
Request body format → {}
Requires “Authorization” header

Design

1) Database Structure

To implement the main structure of the application, two main entities are determined namely **users** and **tournaments**. While the user table is used to store critical information of players, it is also used with tournament table to define **groups** and **ranks** of players in a specific tournament. The reason why the ranks table is defined by using a user ID (unique key) and a tournament ID (unique key) is because a user-tournament tuple uniquely defines the rank of a user on the given tournament. Also, since each tournament should have unique groups, tournament ID is used to define a unique group.



2) Security

To make the application secure, Spring Boot Security and JSON Web Tokens (JWT) are used to authenticate users. Authentication is established as follows:

1) At the start, Spring Security starts an AuthenticationManager, a JWTAuthenticationFilter, a PasswordEncoder and a SecurityFilterChain beans.
2) Every request that arrives at the application is directed to the endpoint directly if the request has direct access permission or to the AuthenticationManager if it does not have direct permission by SecurityFilterChain
3) Only two requests have direct access: ***CreateUserRequest*** and ***AuthenticateUserRequest***. ***CreateUserRequest*** is executed without further steps; however, ***AuthenticateUserRequest*** requires the user’s “password”. User passwords are stored encrypted (BCryptPasswordEncoder).

4) AuthenticationManager directs a request to the JWTAuthenticationFilter to check whether the request has been authenticated or not. If it is authenticated, the request is directed to the endpoint.

3) Performance

To provide users with a real-time leaderboard where they can see their ranking in their group, the leaderboard service functions are implemented by using Redis. Besides real-time availability, achieving high throughput/low latency for ***GetLeaderboardRequest***s is also aimed. To do that, the TournamentService starts a Redis cache which is named as rowMatchBackend and creates an entry for each player that enters the tournament with user ID and group ID. When a user that enters a tournament calls ***UpdateLevelRequest***, server also calls a function of TournamentService which is named as ***updateUserProgressRealtime*** and updates the user progress at the cache.

4) Scheduled Services

To start a tournament everyday between 00.00 - 20.00, Spring Scheduler and Cron are used.