REST APIs for game match

Contents

- Architecture
- REST APIs Design
- Datamodel Design
- Unit testing

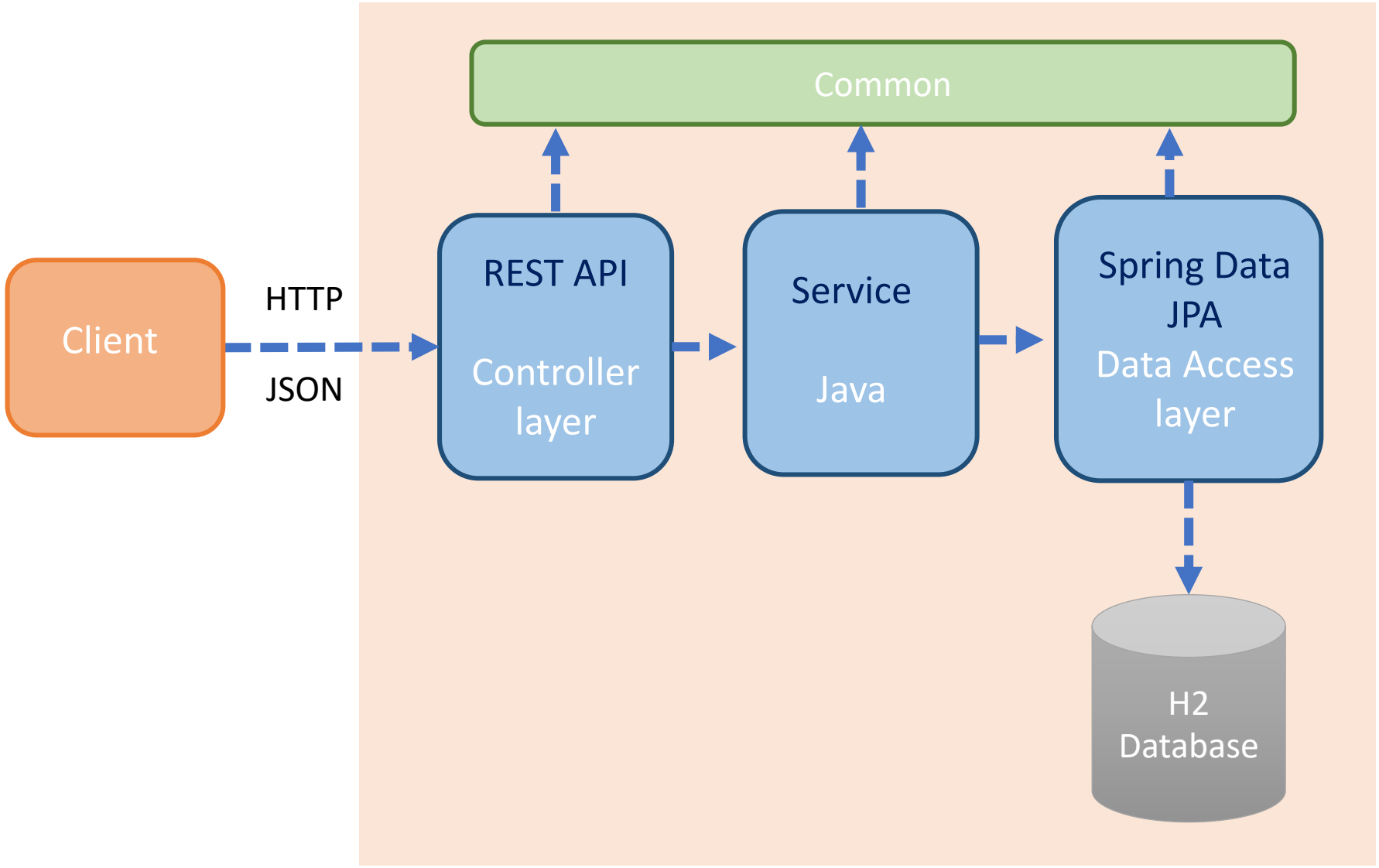
Main features include:

- User and interest information management
- Game auto-matching based on user geography and user interest game and level
- Search maximum credits based on user interest game and level

Built With

- Java - Java 8
- Spring Boot - Spring Boot REST API
- H2 Database - In-memory database
- Maven - Project management

High Level Architecture



System:

- **Controller layer**: The controller layer implements REST API.
- **Service layer**: The business logic of the system.
- **Data access layer**: The data access layer implements Spring Data JPA.
- **Data storage** The storage layer of the application uses in-memory H2 Database, a relational database
- **Common**: The Common component contains utility code (exception handling, configurations, etc.) used across the application.

Package overview:

- `gamematchrestapi.controller` : Provides the REST API.
- `gamematchrestapi.service` : Main logic of the application.
- `gamematchrestapi.entity` : Classes that represent persistable entities.
- `gamematchrestapi.repository` : Classes performs CRUD (Create, Read, Update, Delete) operations and act as the bridge to the H2 Database.
- `gamematchrestapi.common` : Common component contains custom exceptions and configuration of Swagger 3 and CORS.

REST APIs Design

User Entity

No.	API method	HTTP method	URI Path	Status Code	Description
1	getAllUsers	GET	/api/user	200 (OK)	Get all users
2	getUserById	GET	/api/user/{userId}	200 (OK)	Get user by userId
3	createUser	POST	/api/user	201 (CREATED)	Create user
4	updateUserById	PUT	/api/user/{userId}	200 (OK)	Update user by userId
5	deleteUserById	DELETE	/api/user/{userId}	200 (OK)	Delete user by userId
6	getMatchUserByGame AndLevelAndGeography	GET	/api/user/match?game=# &level=#&geography=#	200 (OK)	Get a list of matched users with same game, level and geography
7	getOtherUserMatchUser Interest	GET	/api/user/{userId}/match/ {interestId}	200 (OK)	Get a list of other users that matches user geography and user interest game and level
8	getUserWithMaxCredit ByGameAndLevel	GET	/api/user/interest/credit/ max?game=#&level=#	200 (OK)	Get a list of users with maximum credit among users with same game and level

REST APIs Design

Interest Entity

No.	API method	HTTP method	URI Path	Status Code	Description
9	getInterestByInterestId	GET	/api/user/{userId}/interest/{interestId}	200 (OK)	Get interest by interestId
10	createInterestByInterestId	POST	/api/user/{userId}/interest	201 (CREATED)	Create interest
11	updateInterestByInterestId	PUT	/api/user/{userId}/interest/{interestId}	200 (OK)	Update interest by interestId
12	updateUserInterestCreditByInterestId	PUT	/api/user/{userId}/interest/{interestId}/credit?credit=#	200 (OK)	Update user interest credit
13	deleteUserInterestByInterestId	DELETE	/api/user/{userId}/interest/{interestId}	200 (OK)	Delete interest by interestId

Swagger 3.0
API docs

Gamer Match API 1.0 OAS3

http://localhost:8080/v3/api-docs

RESTful APIs for gamer match

Servers

http://localhost:8080 - Inferred Url

user-controller User Controller

GET	/api/user	getAllUsers
POST	/api/user	Create new user
GET	/api/user/{userId}	getUserById
PUT	/api/user/{userId}	Update user
DELETE	/api/user/{userId}	Delete user and interest of the user by userId.
GET	/api/user/{userId}/match/{interestId}	Get a list of other users that matches user geography and user interest game and level
GET	/api/user/interest/credit/max	Get a list of users with maximum credit among users with same game and level
GET	/api/user/match	Get a list of matched users with same game, level and geography

interest-controller Interest Controller

POST	/api/user/{userId}/interest	Create new interest
GET	/api/user/{userId}/interest/{interestId}	getInterestByInterestId
PUT	/api/user/{userId}/interest/{interestId}	Update interest
DELETE	/api/user/{userId}/interest/{interestId}	deleteUserInterestByInterestId
PUT	/api/user/{userId}/interest/{interestId}/credit	Update user interest credit

POST `/api/user` Create new user

Create new user. Interests of user will also be created if exists. In [Request Body], [name] and [nickname] cannot be empty and should only consist of letter and number with more than 2 characters. [gender] cannot be empty and should be 'male' or 'female'. [geography] cannot be empty and should be one of 'Europe', 'Asia', 'USA'. [interestSet] can be empty. For attributes of interest, [game] cannot be empty should be one of 'fortnite', 'call of duty', 'dota', 'valhalla', 'among us'. [level] cannot be empty and should be one of 'noob', 'pro', 'invincible'. [credit] can be left empy but cannot be negative. User can only have one interest for one game

Parameters **Request body** Try it out

No parameters

Request body

Example Value | Schema

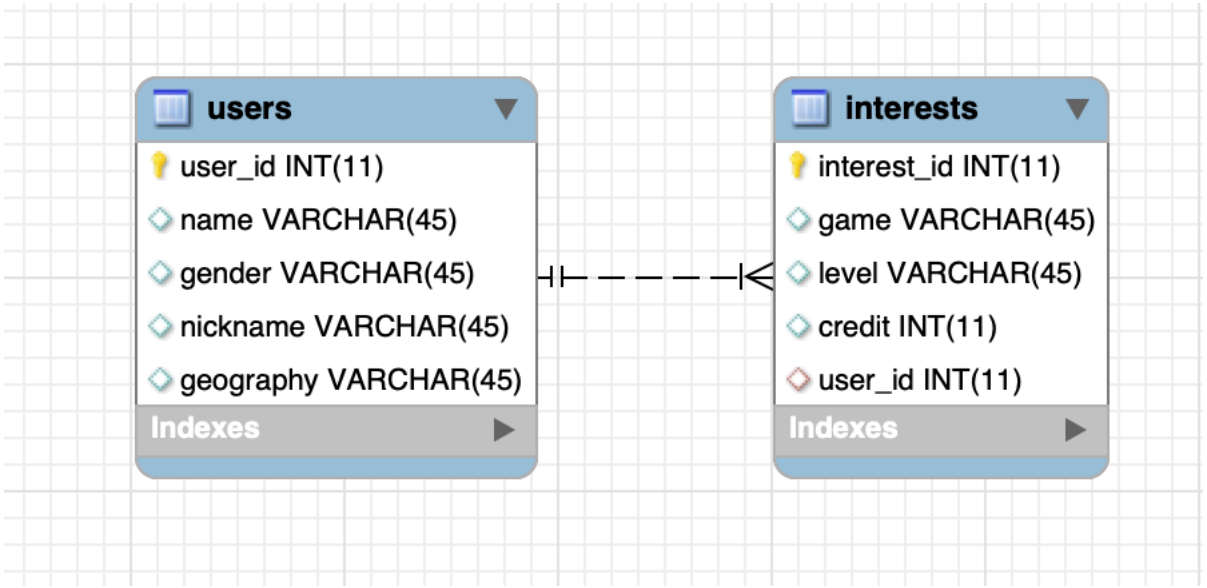
```
{
  "gender": "string",
  "geography": "string",
  "interestSet": [
    {
      "credit": 0,
      "game": "string",
      "level": "string"
    }
  ],
  "name": "string",
  "nickname": "string"
}
```

GET `/api/user/match` Get a list of matched users with same game, level and geography

Parameters **Query parameter**

Name	Description
game * required string *(query)*	Game should be one of 'fortnite', 'call of duty', 'dota', 'valhalla', 'among us' game - Game should be one of 'fortnite', 'call
level * required string *(query)*	Level should be one of 'noob', 'pro', 'invincible' level - Level should be one of 'noob', 'pro', 'in
geography * required string *(query)*	Geography should be one of 'Europe', 'Asia', 'USA' geography - Geography should be one of 'Eu

Datamodel Design



One-to-many

Common: validator

Entity	Attribute	Validator	Comments
User	name	@NotBlank @Size(min = 2, max = 45) @Pattern(regexp = "^[A-Za-z0-9]+$”)	
	gender	@NotBlank, @InStringArray	Gender should be 'male' or 'female'
	nickname	@NotBlank @Size(min = 2, max = 45) @Pattern(regexp = "^[A-Za-z0-9]+$”)	
	geography	@NotBlank, @InStringArray	Geography should be one of 'Europe', 'Asia', 'USA'
Interest	game	@NotBlank, @InStringArray	Game should be one of 'fortnite', 'call of duty', 'dota', 'valhalla', 'among us'
	level	@NotBlank, @InStringArray	Level should be one of 'noob', 'pro', 'invincible'
	credit	@PositiveOrZero	

Common: custom-validator

@InStringArray

```
// male, female
@NotBlank(message = "Gender cannot be empty")
@InStringArray(message = "Gender should be 'male' or 'female'", values = {"male","female"})
private String gender;
```

User {

gender*	string
geography*	string
interestSet	[...]
name*	string maxLength: 45 minLength: 2 pattern: ^[A-Za-z0-9]+$
nickname*	string maxLength: 45 minLength: 2 pattern: ^[A-Za-z0-9]+$
userId	integer($int64)

}

UserRequest {

gender*	string
geography*	string
interestSet	[...]
name*	string maxLength: 45 minLength: 2 pattern: ^[A-Za-z0-9]+$
nickname*	string maxLength: 45 minLength: 2 pattern: ^[A-Za-z0-9]+$

}

POST `/api/user` Create new user

Create new user. Interests of user will also be created if exists. In [Re
characters. [gender] cannot be empty and should be 'male' or 'female'
interest, [game] cannot be empty should be one of 'fortnite', 'call of du
empy but cannot be negative. User can only have one interest for one

Parameters

No parameters

Request body

Example Value | Schema

```
{
  "gender": "string",
  "geography": "string",
  "interestSet": [
    {
      "credit": 0,
      "game": "string",
      "level": "string"
    }
  ],
  "name": "string",
  "nickname": "string"
}
```

Common: exception

- ResourceNotFoundException (404 Not Found)
- InvalidRequestException (400 Bad Request)
- MethodArgumentNotValidException (400 Bad Request)

Policies

API for creating:

- Attempt to create an entity with invalid data: Throws `MethodArgumentNotValidException`.

API for retrieving:

- Attempt to retrieve an entity that does not exist: Throws `ResourceNotFoundException`.
- Attempt to retrieve an entity with invalid request parameter: Throws `InvalidRequestException`.

API for updating:

- Attempt to update an entity that does not exist: Throws `InvalidRequestException`.
- Attempt to update an entity with invalid data: Throws `MethodArgumentNotValidException`.

API for deleting:

- Attempt to delete an entity that does not exist: Throws `InvalidRequestException`.
- Cascade policy: When a parent entity is deleted, entities that have referential integrity with the deleted entity should also be deleted.

Unit testing

- The controller classes were tested with Spring Boot and `@WebMvcTest`
- In the data access layer, JPA Queries of the repository classes were tested with Spring Boot and `@DataJpaTest`
- Integration Tests with `@SpringBootTest`

Unit testing coverage

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	100% (17/ 17)	87.5% (112/ 128)	85.1% (315/ 370)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.tiwa007.gamematchrestapi	100% (1/ 1)	50% (1/ 2)	33.3% (1/ 3)
com.tiwa007.gamematchrestapi.Service	100% (2/ 2)	100% (20/ 20)	89.3% (100/ 112)
com.tiwa007.gamematchrestapi.common.config	100% (2/ 2)	100% (5/ 5)	100% (18/ 18)
com.tiwa007.gamematchrestapi.common.exception	100% (4/ 4)	47.6% (10/ 21)	40.8% (20/ 49)
com.tiwa007.gamematchrestapi.common.exception.validator	100% (1/ 1)	100% (3/ 3)	100% (8/ 8)
com.tiwa007.gamematchrestapi.controller	100% (5/ 5)	91.3% (42/ 46)	88.7% (94/ 106)
com.tiwa007.gamematchrestapi.entity	100% (2/ 2)	100% (31/ 31)	100% (74/ 74)

Thank you for your attention!