**User Management REST API Documentation**

Tools/Technologies used :

- Java 11

- MySQL

- Spring Framework (including additional modules)

- Jackson Project

- Eclipse IDE

- MySQL Workbench

- Postman

- Spring Initializr

**About the project :**

The REST API is created as a Maven Project with Spring Boot, using Spring Initializr for setting up the initial project structure and adding all needed, non version conflicting dependencies to the project classpath.

The datasource and JDBC connection properties are configured in the **application.properties** file.

Spring Security is used for securing all exposed API endpoints. Authorized users with user roles/authorities are stored in the “user\_directory” schema in the database with passwords stored using bcrypt one way encryption as per Spring team recommendation. The API uses JDBC authentication and all passwords are set (for testing purposes only) to “pass123”. Different users are granted access to HTTP methods/requests according to their role. All configuration is done in the Java source code in the “SecurityConfig” class.

In the Java source code, the main resource/entity of the API, **User**, is mapped to the **user\_directory** schema in the database.

Standardized best practice approach is used while developing the project with three main layers - controller layer, service layer and DAO layer. The controller layer exposes API endpoints and delegates HTTP method calls to the service layer which delegates the calls to the DAO layer.

In this case, in the DAO layer, the **UserRepository** extends **JpaRepository** which provides all CRUD methods and functionality also including sorting and pagination. Custom search by first name or last name functionality is added in the **UserRepository** using JPQL query, accepting a query parameter. The search is case insensitive.

In the service layer, custom business logic is added for throwing custom exceptions.

In the controller layer, custom exception handling is implemented with custom error response. Exception handling is implemented using Spring AOP.

Jackson is responsible for automatically converting Java objects to JSON.

**Functionality :**

The API offers full CRUD functionality. Accessing all endpoints requires basic authentication.

The test password for all authenticated users is “pass123”.

At endpoint <http://localhost:8080/api/users> , for all authenticated user roles, HTTP method **GET** can be used for reading all users as JSONs or reading a single user by identifier :

- [http://localhost:8080/api/users/](http://localhost:8080/api/users)

- [http://localhost:8080/api/users/1](http://localhost:8080/api/users)

- [http://localhost:8080/api/users/5](http://localhost:8080/api/users)

... etc.

If the operation is successful, a user or all users are read, otherwise if the id of a user is not found, a custom error response is returned as JSON.

Users ca be sorted by different criteria :

- [http://localhost:8080/api/users?sort=firstName](http://localhost:8080/api/users)

- [http://localhost:8080/api/users?sort=lastName](http://localhost:8080/api/users)

- [http://localhost:8080/api/users?sort=dateOfBirth](http://localhost:8080/api/users)

... etc.

Also descending ordering can be used :

- [http://localhost:8080/api/users?sort=firstName,desc](http://localhost:8080/api/users)

... etc.

Users ca be read with pagination with configurable page number and size. Page numbering is zero based :

- [http://localhost:8080/api/users?page=0&size=2](http://localhost:8080/api/users)

... etc.

Search functionality is included where a search term is provided and a list of users is returned, where the search term is found:

- [http://localhost:8080/api/users/search?query=ana](http://localhost:8080/api/users)

... etc.

At endpoint <http://localhost:8080/api/users> , for MANAGER and ADMIN authenticated user roles, HTTP method **POST** can be used for creating a user by providing a JSON with the required information :

{

"id": 2,

"firstName": "Zeratul",

"lastName": "Fromaiur",

"dateOfBirth": "1998-03-31",

"phoneNumber": "+359 123 123 321",

"email": "zeratul@blizzard.com"

}

It does not matter if an id field is passed within the JSON. New user will be created with a new unique id.

At endpoint <http://localhost:8080/api/users> , for MANAGER and ADMIN authenticated user roles, HTTP method **PUT** can be used for updating an existing user by providing a JSON with the required information:

{

"id": 2,

"firstName": "Zeratul",

"lastName": "Fromaiur",

"dateOfBirth": "1998-03-31",

"phoneNumber": "+359 123 123 321",

"email": "zeratul@blizzard.com"

}

If the id field is modified, a new user is created, if not - the current user is updated with the modified data.

At endpoint <http://localhost:8080/api/users> , for ADMIN authenticated user role, HTTP method **DELETE** can be used to delete a user where an id is provided :

- [http://localhost:8080/api/users/1](http://localhost:8080/api/users)

[-http://localhost:8080/api/users/5](http://localhost:8080/api/users)

... etc.

If the operation is successful, confirmation message is received, otherwise a custom error response is returned as JSON.