

# Technical documentation

## Swagger

For the API documentation, Swagger is used. By navigating to <http://localhost:8080/swagger-ui/index.html#/>, you can view all available endpoints, including their expected input and returned responses. If you see a lock icon next to an endpoint, it means the endpoint is secured and requires authentication. Swagger also allows you to test endpoints directly. Simply expand an endpoint, click “**Try it out**”, fill in the necessary data if applicable, and then click “**Execute**” to send the request.

## Authentication

For authentication I am using [JWT](#). Navigate to the file **JwtService—method: generateToken()** to see what is stored in the token.

There are two ways to authenticate:

1. **email en password (/api/auth/login)**  
This endpoint is meant for login via the website
2. **Identifier(email of username) and pincode (/api/auth/login-pincode)**  
This endpoint is meant for login via the Unity app

### How to authenticate via Swagger?

1. Go to the endpoint **/login** or **/login-pincode**
2. Fill in the credentials of a user to log in. Here I am using the admin account.

POST /api/auth/login Login with email and password

Authenticates a user and returns a JWT token if credentials are valid.

Parameters

No parameters

Request body *required*

login-credential data

```
{  "email": "admin@vrbalance.com",  "password": "admin"}
```

Execute

- Click “execute” and copy the JWT-token it returns

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/auth/login' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "email": "admin@vyrbalance.com",
    "password": "admin"
  }'
```

Request URL

http://localhost:8080/api/auth/login

Server response


Code	Details
200	<div><div>Response body</div><div><pre>{   "status": 200,   "message": "Login successful",   "timestamp": "2025-06-18T15:01:39.243523Z",   "data": {     "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxIiwicm9sZSI6IkkFETU1OiwiaWF0IjoxNzUwMjYyNDk5fQ.2lNi30GZkg4kVz3mxN6tsyghG0Vcpjs2Iot9tLFapq0"   } }</pre></div><div><div>Download</div></div></div>

Response headers

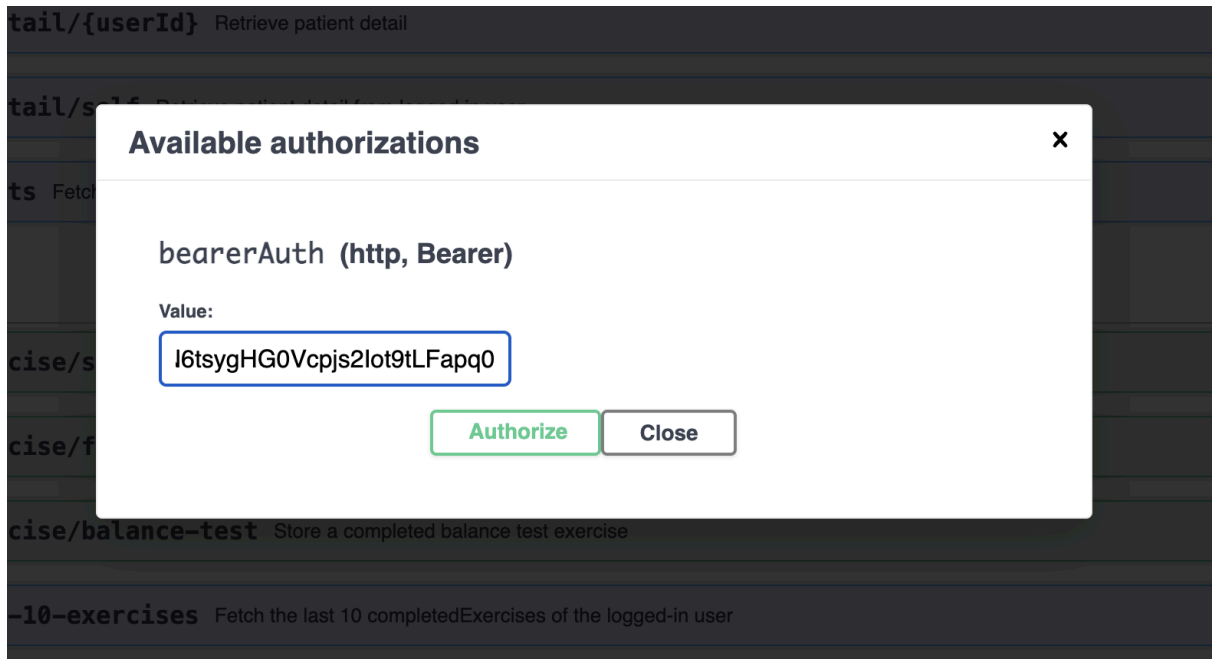
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxIiwicm9sZSI6IkkFETU1OiwiaWF0IjoxNzUwMjYyNDk5fQ.2lNi30GZkg4kVz3mxN6tsyghG0Vcpjs2Iot9tLFapq0

- Scroll back to the top and click:

Authorize



5. Paste the token and click “authorize”



6. Now you're logged in, you can tests all the endpoints with swagger

## IMPORTANT!

To let Swagger know that an endpoint is secured and requires authentication, you **must** add the following annotation to the method:

```
@SecurityRequirement(name = "bearerAuth")
```

This tells Swagger to apply the global bearerAuth security scheme to the endpoint. As a result:

- A lock icon will appear next to the endpoint in Swagger UI.
- Swagger will automatically include the JWT token you provided via the “Authorize” button when testing the endpoint.

Go to the endpoint **/me** in the userController.java to see a full explanation of how it works.

## Securing and exposing endpoints

By default, **all endpoints require authentication**. If you want to make an endpoint public (accessible without authorization), you must explicitly tell Spring Security to allow access to that URL.

### How to do this:

1. Go to the file `SecurityConfig.java`.
2. Inside the `.requestMatchers()` section, add the public URL pattern.

### Example:

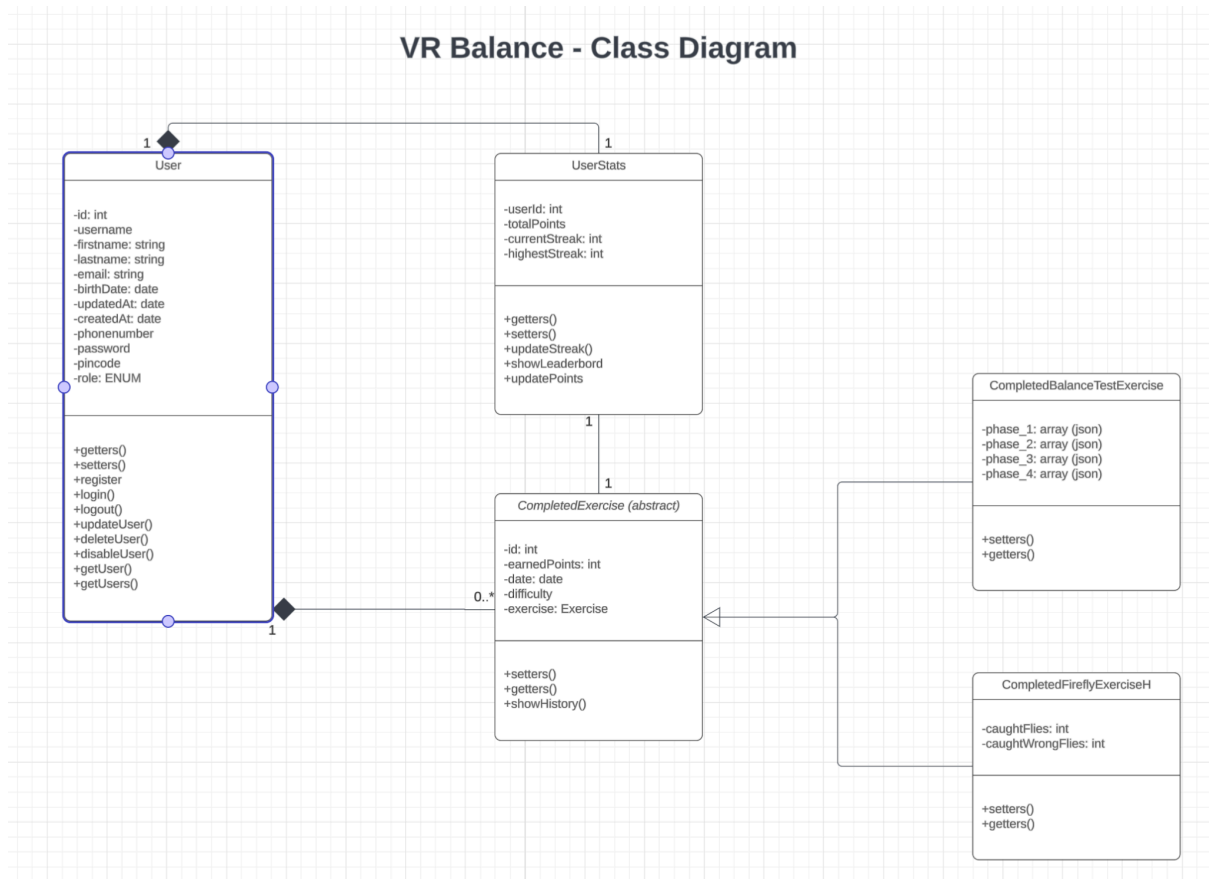
```
.requestMatchers(  
    "/api/user-stats/public/**"  
) .permitAll()
```

- The `**` wildcard means: *everything after /public/ is accessible without authentication*.
- This allows you to create both public and secured endpoints under the same controller.

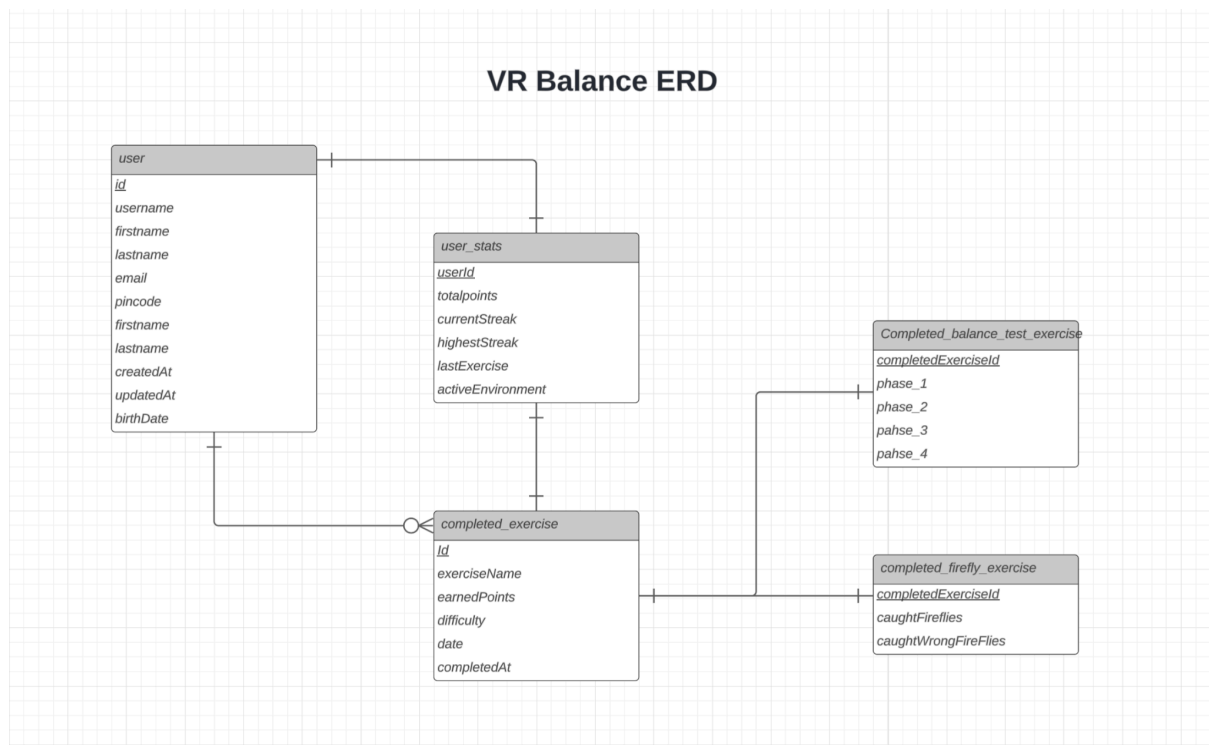
### Example in the code:

Go to `UserStatsController.java` to see how public and secured endpoints are combined within the same class.

# Class diagram



## ERD



## Register flow

A user (patient) can only be created by an account that has the role **PHYSIOTHERAPIST** or **ADMIN** via the `/api/auth/register-patient` endpoint. After registering a user an instruction is sent to the given email by the physiotherapist. Use **mailhog** to test it in development.

## Saving an exercise flow

All exercises that do not require additional data beyond what is already provided by the **CompletedExercise** class are stored directly using that class. For exercises that need to store extra fields — such as **FireflyExercise** and **BalanceTest** — you should create a subclass that extends **CompletedExercise**. You can refer to **ExerciseController.java** and **ExerciseService.java** for an example of how this has been implemented.

## Database seeder

Navigate to **backend/src/main/vr/balance/app/seed/DatabaserSeeder.java** to view all the data that gets created when the application starts. The most important method is **createUsers()**, which sets up standard test users. The other methods are only used for testing purposes and are not required for normal application use.