



# EXERCISES — JWS

version #1.1

---



# Copyright

This document is for internal use at EPITA ([website](#)) only.

Copyright © 2022-2023 Assistants [<assistants@tickets.assistants.epita.fr>](mailto:assistants@tickets.assistants.epita.fr)

**The use of this document must abide by the following rules:**

- ▷ You downloaded it from the assistants' intranet.\*
- ▷ This document is strictly personal and must **not** be passed onto some-one else.
- ▷ Non-compliance with these rules can lead to severe sanctions.

## Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Bomberman . . . . .	5
1.2	Goal . . . . .	5
<b>2</b>	<b>Frameworks</b>	<b>6</b>
2.1	Quarkus . . . . .	6
2.2	Hibernates . . . . .	6
<b>3</b>	<b>Architecture</b>	<b>6</b>
<b>4</b>	<b>How to start</b>	<b>6</b>
4.1	Set-up the database . . . . .	6
4.2	Create a database . . . . .	7
4.3	Given Files . . . . .	7
<b>5</b>	<b>Endpoints explanation</b>	<b>9</b>
5.1	List games . . . . .	9
5.2	Game creation . . . . .	9
5.3	Get a specific game . . . . .	9
5.4	Join a game . . . . .	9
5.5	Start a game . . . . .	10
5.6	Move . . . . .	10
5.7	Set down a bomb . . . . .	10

---

\*<https://intra.forge.epita.fr>

## File Tree

```
jws/
├─ src/
│   └─ main/
│       └─ java/
│           └─ fr/
│               └─ epita/
│                   └─ assistants/
│                       └─ jws/
│                           ├── converter/
│                           │   └─ *.java (to submit)
│                           ├── data/
│                           │   ├── model/
│                           │   │   └─ *.java (to submit)
│                           │   └─ repository/
│                           │       └─ *.java (to submit)
│                           ├── domain/
│                           │   ├── entity/
│                           │   │   └─ *.java (to submit)
│                           │   └─ service/
│                           │       └─ *.java (to submit)
│                           ├── errors/
│                           │   └─ *.java (to submit)
│                           ├── presentation/
│                           │   ├── rest/
│                           │   │   └─ *.java (to submit)
│                           │   ├── request/
│                           │   │   └─ *.java (to submit)
│                           │   └─ response/
│                           │       └─ *.java (to submit)
│                           └─ utils/
│                               └─ *.java (to submit)
```

## Obligations

Obligations are **fundamental** rules shared by all subjects. They are non-negotiable and to not apply them means to face sanctions. Therefore, do not hesitate to ask for explanations if you do not understand one of these rules.

**Obligation #0: Cheating**, as well as sharing source code, tests, test tools or coding-style correction tools is **strictly forbidden** and penalized by not being graded, being flagged as a cheater and reported to the academic staff.

**Obligation #1:** Your submission repository must be **clean**. Except for special cases, which (if any) are **explicitly** mentioned in this document, an *unclean* repository may contain:

- binary files;<sup>1</sup>
- files with inappropriate privileges;
- forbidden files: `*~`, `*.swp`, `*.o`, `*.a`, `*.so`, `*.class`, `*.log`, `*.core`, etc.;
- a file tree that does not follow our specifications.

---

<sup>1</sup>If an executable file is required, please provide its sources **only**. We will compile it ourselves.



## 2 Frameworks

For this project you will be using two different frameworks: Quarkus and Hibernate.

### 2.1 Quarkus

Quarkus was designed for ease of use right from the beginning. It has features that work well with minimal configuration and is considerably less opinionated and more lightweight than other frameworks. You can use an active record pattern and define your functionalities as entity methods, use a repository pattern, or use services to implement them easily with minimal setup. Quarkus is optimized for memory density and rapid startup time. Applications running on the JVM with Quarkus can deliver nearly twice as many instances in the same amount of RAM when compared to other cloud-native Java stacks, and up to 7 times more instances when packaged as a native binary.

### 2.2 Hibernate

Hibernate ORM is the de facto standard JPA implementation and provides a fully featured Object Relational Mapper (ORM). It works beautifully in Quarkus and is a powerful tool that allows you to define your entities and relationships in a declarative way.

## 3 Architecture

Adhering to the right architectural pattern is the main goal of this project. It will be a valuable knowledge if you have to work on the backend of an intranet or an application during your internship, for example.

Your project **should** respect the architecture provided in the given files.

#### Tips

You will find more details about the expected architecture in the annexe. This file is located on the forge's intranet on the same page as this subject.

## 4 How to start

### 4.1 Set-up the database

First, let us configure a PostgreSQL-specific environment variable:

```
42sh$ echo 'export PGDATA="$HOME/postgres_data"' >> ~/.bashrc
42sh$ echo 'export PGHOST="/tmp"' >> ~/.bashrc
42sh$ source ~/.bashrc
```

This first line adds a PGDATA environment variable to your .bashrc, containing the location of your choice to store PostgreSQL's data. The second specifies the host name of the machine on which the server will run. As the value begins with a slash, it will be used as the directory containing a socket on which postgres will listen. Then, let us initialize a new PostgreSQL database cluster. It will generate default databases and configurations in the \$PGDATA directory.

```
42sh$ nix-shell -p postgresql
42sh$ initdb --locale "$LANG" -E UTF8
```

### Be careful!

If you experience a permission denied error, you may need to restore your rights using

```
42sh$ chmod 755 ~
```

## 4.2 Create a database

Your server is now up and running! You have to set the DB\_USERNAME variable to your login. PostgreSQL offers an interactive shell that connects to this server and behaves as a front-end for your operations on databases:

```
42sh$ export DB_USERNAME=<login>
42sh$ postgres -k "$PGHOST"
42sh$ psql postgres
```

psql takes the name of the database you want to connect to. Here, we choose the default database, postgres. These commands will create a brand new database named jws.

```
-- Give yourself all the rights
-- If you are not on the PIE, <login> should be your username
postgres=# ALTER ROLE "<login>" SUPERUSER;
-- Create a database named jws
postgres=# CREATE DATABASE jws OWNER <login>;
-- Exit PostgreSQL
postgres=# \q
```

After that you have to create a schema named jws.

## 4.3 Given Files

In order to start the project in good condition, a tarball containing all the necessary files for proper project functioning is available on the intranet.

You **should** decompress this tarball and open the directory with IntelliJ before reading the upcoming sections of the subject.

Once your project is opened in your IDE, you can type the following command to start the server.

```
42sh$ mvn quarkus:dev
```

It will launch your API on port 8082. We have provided you with a Hello World endpoint, so the next command will return "Hello World!".

```
42sh$ curl http://localhost:8082
```

To launch the viewer, you can type the following command from the root of your project:

```
42sh$ java -jar front-end.jar
```

You can now use your viewer through your browser at the following URI : `http://localhost:3000`

A Swagger of the project is given for you to see the different endpoints you have to implement, error codes, and the answers's format. You will find it in `src/main/resources/openapi.yaml`, you can use a website such as <https://editor.swagger.io> to see it in better conditions.

In order to pass our architecture tests, you **must** respect the following architecture :

- **data** : This directory **must** contain all the logic related to the model layer
  - **model**: This directory **must** contain all your models
  - **repository**: This directory **must** contain all your repositories
- **domain**: This directory **must** contain all the logic related to the service layer
  - **service**: This directory **must** contain all your services
  - **entity**: This directory **must** contain all your entities
- **presentation**: This directory **must** contain all the logic related to your controllers.
  - **rest**: This directory **must** contain all your REST controllers
    - \* **request**: This directory **must** contain your request DTO
    - \* **response**: This directory **must** contain your response DTO

In order to make your program work, you **must** handle the following environment variables:

- **JWS\_MAP\_PATH**: A path to a map in RLE format, you can find a default one at this path `src/test/resources/map1.rle`
- **JWS\_TICK\_DURATION**: The duration of a tick in ms
- **JWS\_DELAY\_MOVEMENT**: The delay in tick between two movements
- **JWS\_DELAY\_BOMB**: The delay in tick between two bombs set down, it is also the delay before the bomb explodes.

### Be careful!

Be careful, your code will be tested with the following commands:

```
42sh$ mvn package -Dquarkus.package.type=uber-jar -DskipTests
42sh$ java -jar target/jws-1.0-runner.jar
```

They do not have the exact same behavior as

```
42sh$ mvn quarkus:dev
```

The first command will take longer to complete, hence, you should use the first batch of commands only to make sure your code will behave as expected on the moulinette, and the second to make



## 5 Endpoints explanation

### 5.1 List games

This endpoint lists all the games registered in the database. The format to follow is in the provided Swagger.

### 5.2 Game creation

This endpoint create a new game and a first player with the name provided in the request. The initial state of the game **must** be `STARTING`.

Players must spawn at the following points with 3 lives:

- Top left (x=1, y=1)
- Top right (x=15, y=1)
- Bottom right (x=15, y=13)
- Bottom left (x=1, y=13)

IDs of games and players must start at 1.

The map is passed in a simplified RLE format in the response, the length before each kind of block is less than 10.

The blocks should have this name in the RLE file : - "M" -> Metal - "W" -> Wood - "G" -> Floor - "B" -> Bomb

### 5.3 Get a specific game

This endpoint gather informations about a specific game. All the format details of this endpoint are listed in the Swagger.

### 5.4 Join a game

All the needed information are in the Swagger. Players **must** spawn at the points defined in the *game\_creation* part.

## 5.5 Start a game

You should update the state of your game to `RUNNING`. Once again, all the remaining information about this endpoint is in the Swagger.

## 5.6 Move

This action allows the player to move. The movement is done in the given direction.

### Be careful!

The player can only move in a cardinal direction and cannot go through walls nor through bombs.

The remaining behavior of this endpoint is defined in the given Swagger.

## 5.7 Set down a bomb

This endpoint allows the player to place a bomb. The bomb can only be placed where the player currently is. If all players are dead, the state of the game should be updated to `FINISHED`.

The bomb explodes in the cardinal directions within a range of a single block. The explosion destroys only wooden blocks and takes a life from the players within its radius. You must ensure that the bomb explodes precisely `JWS_DELAY_BOMB` ticks after it has been planted.

The bomb's explosion will be tested separately from the process of placing them down.

*You mean it's working? For real this time?*