DOCUMENTATION - TOLL PARKING LIBRARY

24 mai 2020

Yassine ES-SAIYDY

Table des figures

Chapitre 1

Build Toll Parking Library from resources

1.1 How to build Toll Parking Library project

Step-by-step documentation for deploying and running **Toll Parking Library** on your local machine :

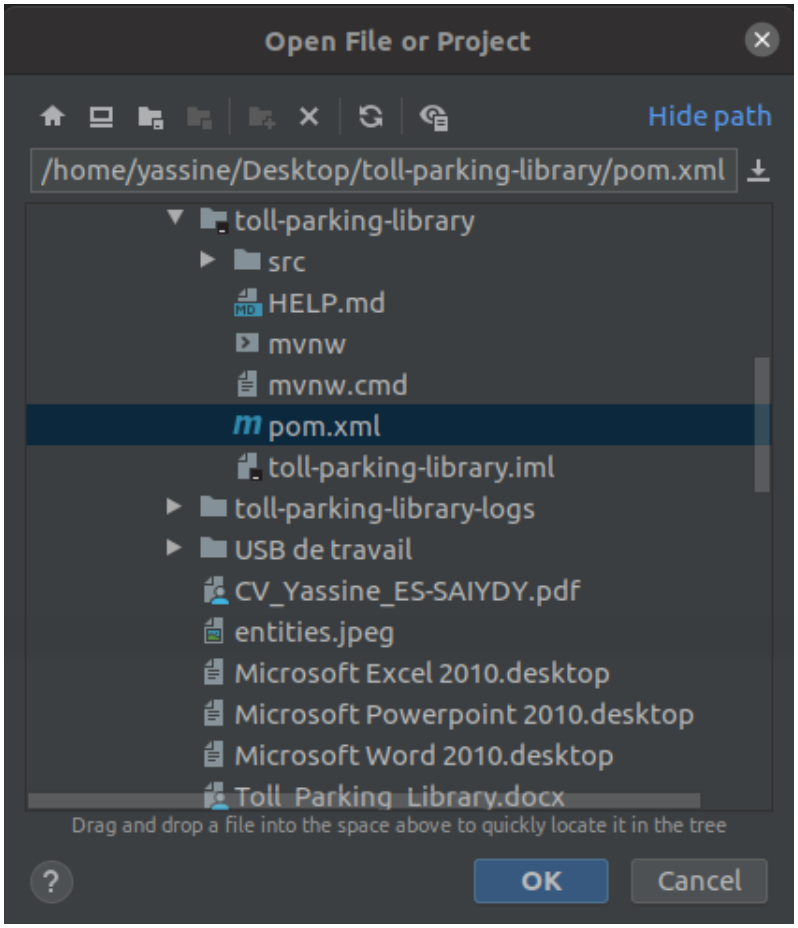
1. We will be building this project from scratch using following tools and technologies :
   (a) IntelliJ IDEA **Community**
   (b) Spring Boot 2.3.0.RELEASE
   (c) Maven
   (d) MySQL database system and MySQL Workbench
   (e) Hibernate
   (f) Java 11
2. Start by going to the IntelliJ IDEA download site : IntelliJ IDEA **Community**.

3. Clone the "Toll Parking Library" project directory into a new directory on your local machine with the following command :

```
git clone git@github.com:YasinSaidi/toll-parking-library.git
```

4. Launch IntelliJ IDEA and import the project :

Figure 1.1: Select the **Toll Parking Library**'s pom.xml file to import



5. Execute the following Maven objectives : **mvn clean install**, and it should end with a message "BUILD SUCCESS".

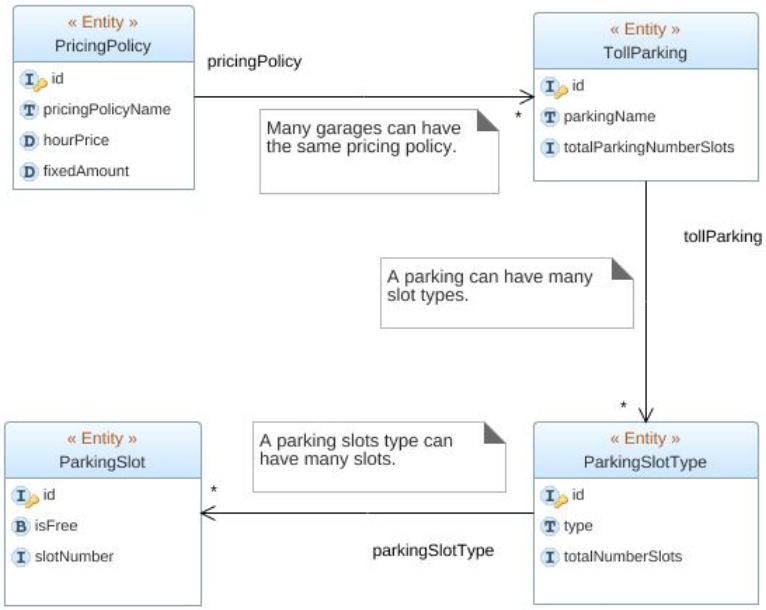
Chapitre 2

Functional context of the application

2.1 Java API - Generate and save random data in the application database

1. Before running the application on your local machine, you will need to create a new schema in the connected "MySQL" server and name it : **toll-parking-library-database**.
2. Run "toll-parking-library" as Java application in IntelliJ IDEA **Community** : TollParkingLibraryApplication.main() -**The application starts on port : 7777 (http)**-, **JPA** will generate the following tables from the entities classes :
   (a) 'toll-parking-library-database'.**pricing_policy**
   (b) 'toll-parking-library-database'.**toll_parking**
   (c) 'toll-parking-library-database'.**parking_slot_type**
   (d) 'toll-parking-library-database'.**parking_slot**

Figure 2.1: Model "Toll Parking Library" database in a UML diagram



3. Use the following Java API :

   http ://localhost :7777/generateRandomData/pricingPolicies

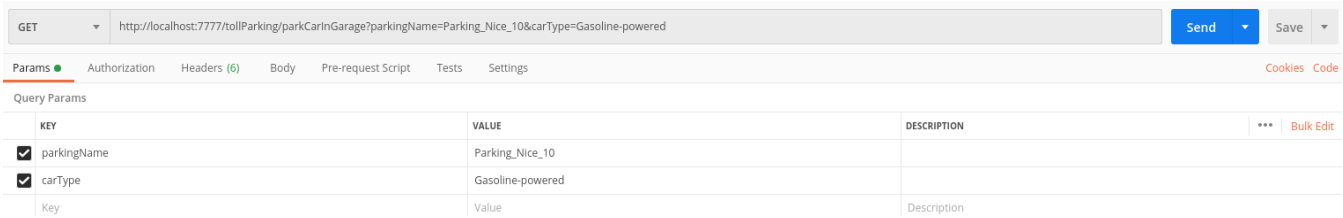
   To generate and save data in the application database to be able to test "**Toll Parking Library**" Java APIs correctly.

2.2 "Toll Parking Library" Java APIs

After running TollParkingLibraryApplication.main(), open your browser and go to the following link : http ://localhost :7777/swagger-ui.html, you will find the JAVA APIs specification : List of endpoints, HTTP methods, API parameters, and API responses.

2.2.1 Java API : Park car in the garage

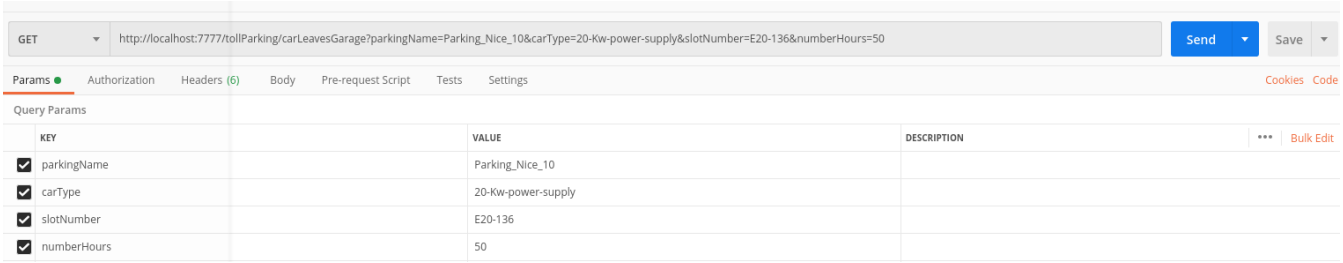
Figure 2.2: Park car in the garage



1. **Request parameter** : parkingName : From the following database table 'toll-parking-library-database'.toll_parking, you can get a valid parking name and use it to test the different APIs.
   Example : 'Parking_Paris_8'.
2. **Request parameter** : carType : we have three parking slot type : "Gasoline-powered", "20-Kw-power-supply", and "50-Kw-power-supply".

2.2.2 Java API : The car leaves the garage

Figure 2.3: The car leaves the garage



1. **Request parameter** : parkingName : From the following database table 'toll-parking-library-database'.toll_parking, you can get a valid parking name and use it to test the different APIs.

   Example : 'Parking_Paris_8'.

2. **Request parameter** : carType : we have three parking slot type : "Gasoline-powered", "20-Kw-power-supply", and "50-Kw-power-supply".

3. **Request parameter** : slotIdentifier : It consists of the slot code and the slot number :

   (a) **G** : "Gasoline-powered".

   (b) **E20** : "20-Kw-power-supply".

   (c) **E50** : "50-Kw-power-supply".

   Examples : 'G-100', 'E20-150', 'E50-200'...

4. **Request parameter** : numberHours : An integer represents the number of hours a car spent in the parking.