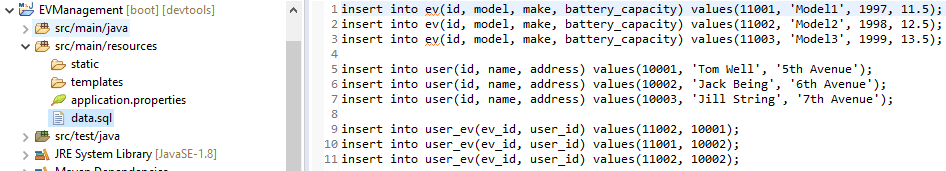
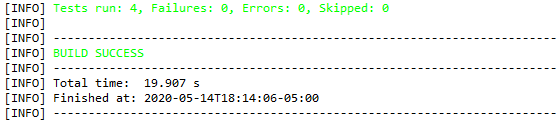
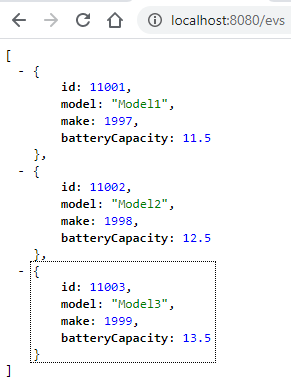
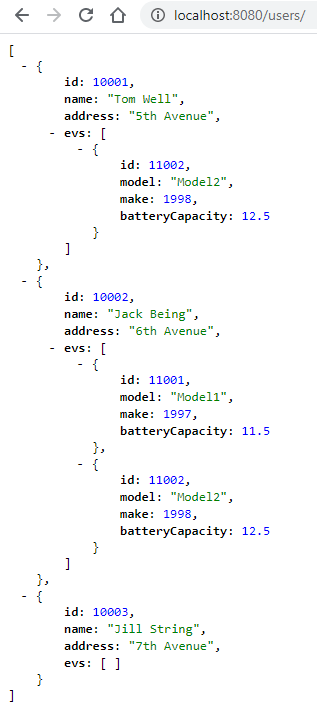
**Instructions to run the EVManagement application.**

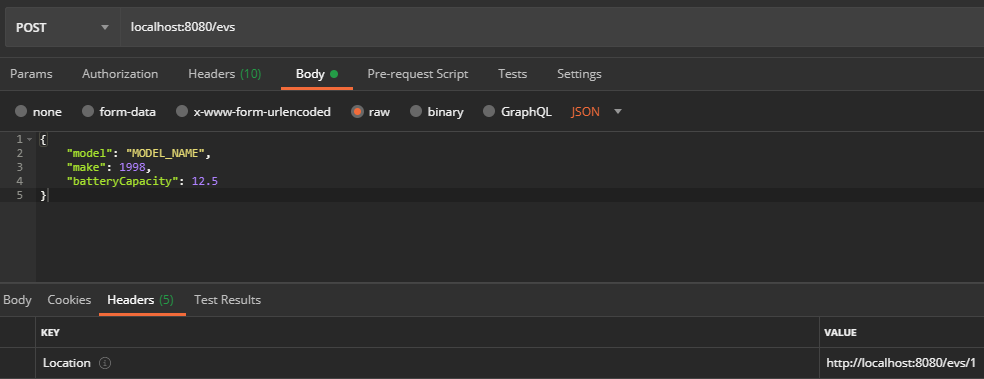
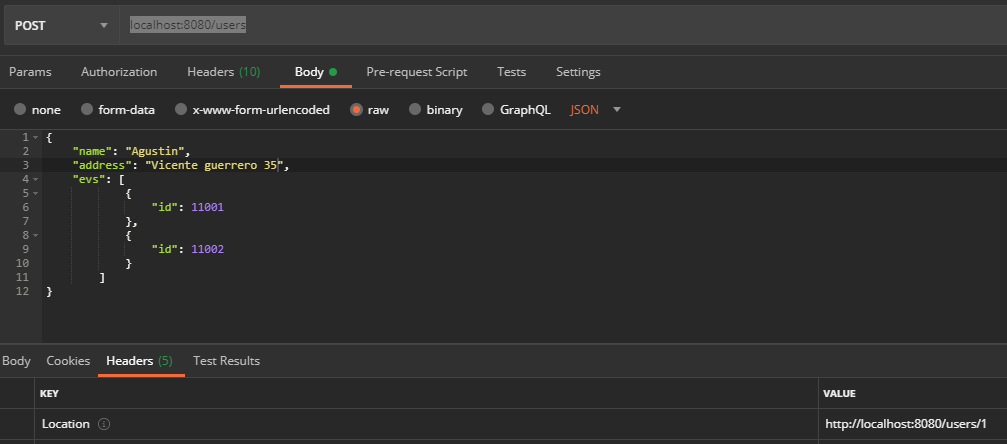
1. The repository address is the following (it is a public repository so anyone can download the code): <https://github.com/DeadmanAgus/EVManagement>
2. The application was created using the following technologies:
   1. Spring boot.
   2. Java version 8; which must be installed to be able to run the application.
   3. H2 database; this is an in-memory database so that the team can test without the need of installing or configuring anything else (but java 8).
3. The code can be downloaded (or cloned) and then exported into eclipse (sts) or another IDE (I included all the project’s files).
4. **Once** inside the repository, in the root directory there is a folder named **target** which contains a file named [**EVManagement.jar**](https://github.com/DeadmanAgus/EVManagement/blob/master/target/EVManagement.jar) which can run with the following command (or it can be ran from eclipse as well):  
   java -jar [EVManagement.jar](https://github.com/DeadmanAgus/EVManagement/blob/master/target/EVManagement.jar)
5. The application can also be started in eclipse by right clicking the file named **EvManagementApplication** then choose **Run As** then **Spring Boot App.**
6. Once loaded the project in eclipse inside the resources folder there is a **data.sql** file that will load some initial data into the database (new records can also be added using the created API), also that data is used in the unit testing.  
   
7. Three tables are used in this project, one to store Users, one to store EVs and the many to many relationship between them.
8. The unit tests can be ran from inside (or outside) eclipse they will perform the testing using the database initial data (for time reasons only 4 tests were included but they cover from controller down to DAOs):  
   
9. Once started the application the **GET** URLS will be the following:
   1. <http://localhost:8080/evs> will show all the pre-loaded EVs:



* 1. <http://localhost:8080/evs/popular> will show the most popular EV
  2. [http://localhost:8080/evs/{id}](http://localhost:8080/evs/%7bid%7d) will show info about a specific EV, example: <http://localhost:8080/evs/11001>
  3. <http://localhost:8080/users/> will show all the pre-loaded users and their EVs:



* 1. <http://localhost:8080/users/>{id} will show info about a specific User, example: <http://localhost:8080/users/10001>

1. The **POST** URLS will be the following:
   1. [http://localhost:8080/evs](http://localhost:8080/evs/) will create an EV, must be sent with the following structure (as seen in the following image it will return a header with the URL of the just created record):  
        
      
   2. <http://localhost:8080/users> will create a User, must be sent with the following structure, please notice that an array of EV ids is being sent to create the relationship between Users and User EVs (as seen in the following image it will return a header with the URL of the just created record):  
      
2. The **DELETE** URLS will be the following:
   1. [http://localhost:8080/evs](http://localhost:8080/evs/)/{id} will delete an EV by id.
   2. <http://localhost:8080/users>/{id} will delete an EV by id.

Notes:

1. The application covers what was asked in the **Nuvve Coding Test.docx** document, for timing reasons no validations nor update functionality were added.
2. In case you have any doubt or are not able to test please let me know.