**UNT CSCE 5430 - Software Engineering**

**Assignment 3 - Microservices**

Dr. Lotfi ben Othmane

**1. Goal**

The goal of this assignment is to practice with microservices and learn the basics about them.

**2. Preparation**

**2.1 Prerequisites**

Install the following software.

1. Java 22
2. Intellij Idea IDE
3. Postman (<https://www.postman.com/downloads/>)

**2.2. Cloning the repository**

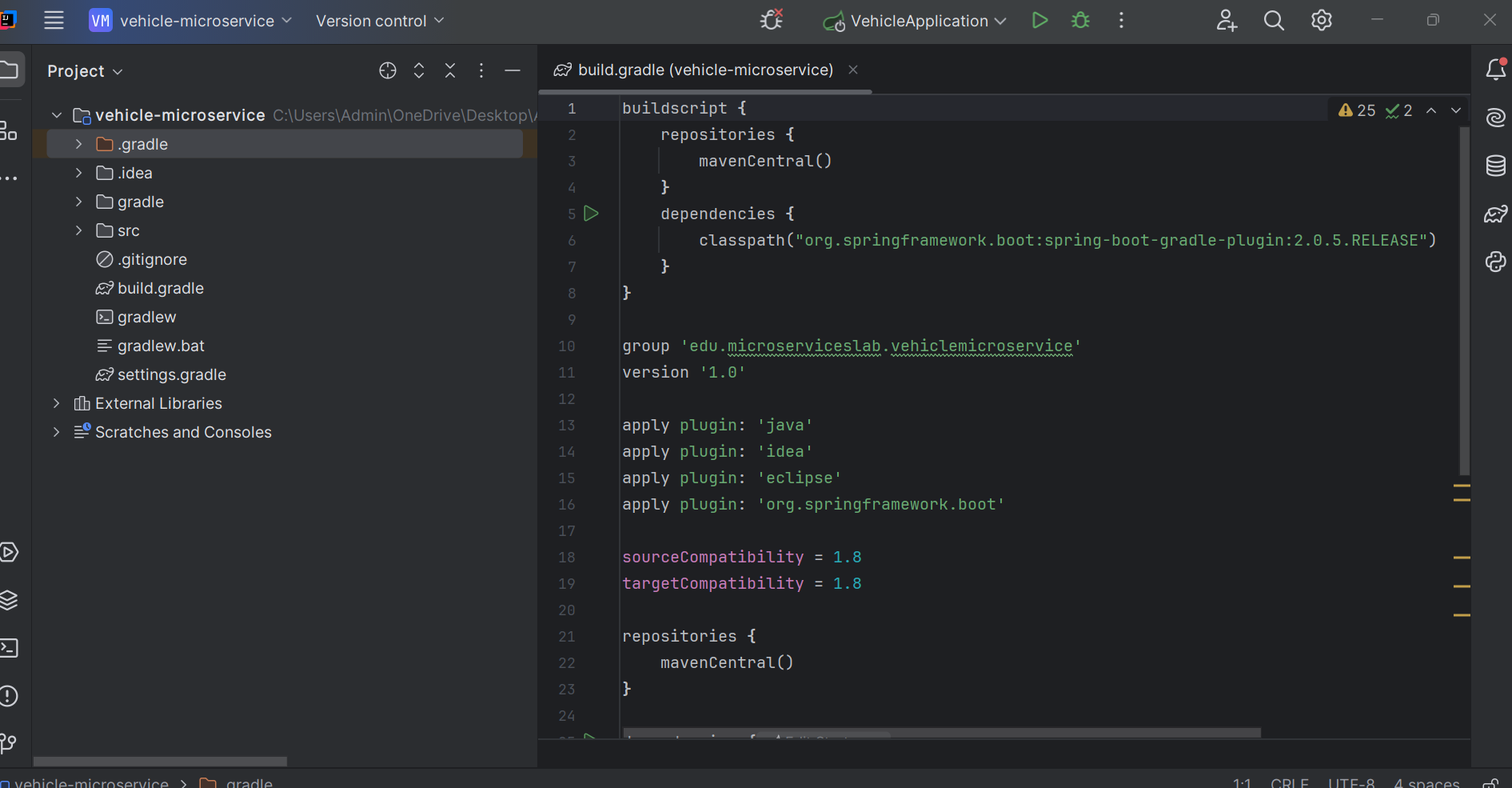
Use the following steps to get a copy of the project that you will extend in this assignment

1. Navigate via the command line to the directory you want to download the project to.
2. Clone the project from <https://github.com/LBOTeaching/CSCE5430.git>

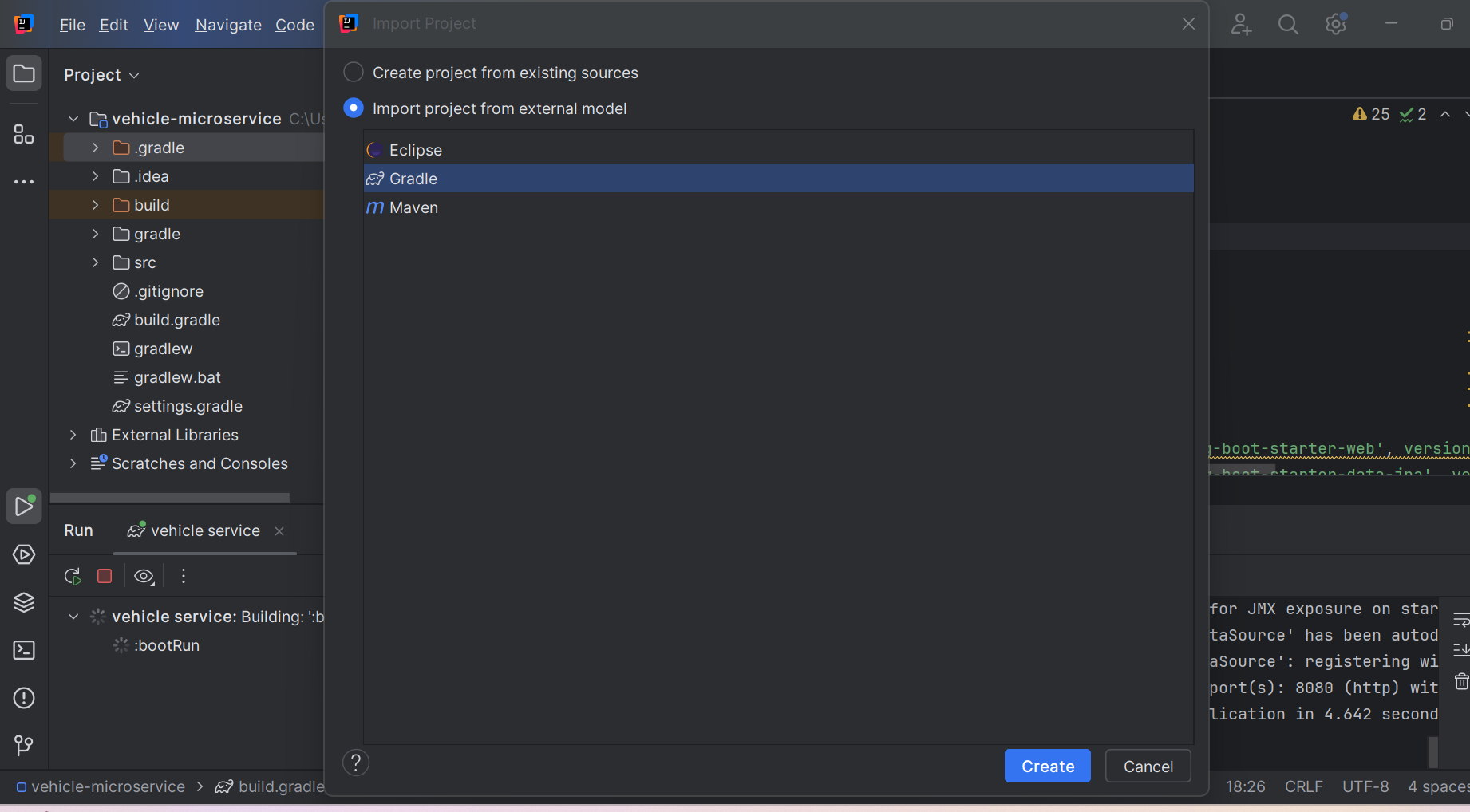
**2.3 Importing a microservice project using Intellij**

To develop microservices in most IDEs, each micro-service must be opened as a new project. Use the following steps to import each of the three micro-services that compose the project.

1. In Intellij, select *File > New > Project* from existing sources. The, navigate to the *vehicle-microservice* folder and hit OK.



1. Make sure the project is imported as a Gradle project.

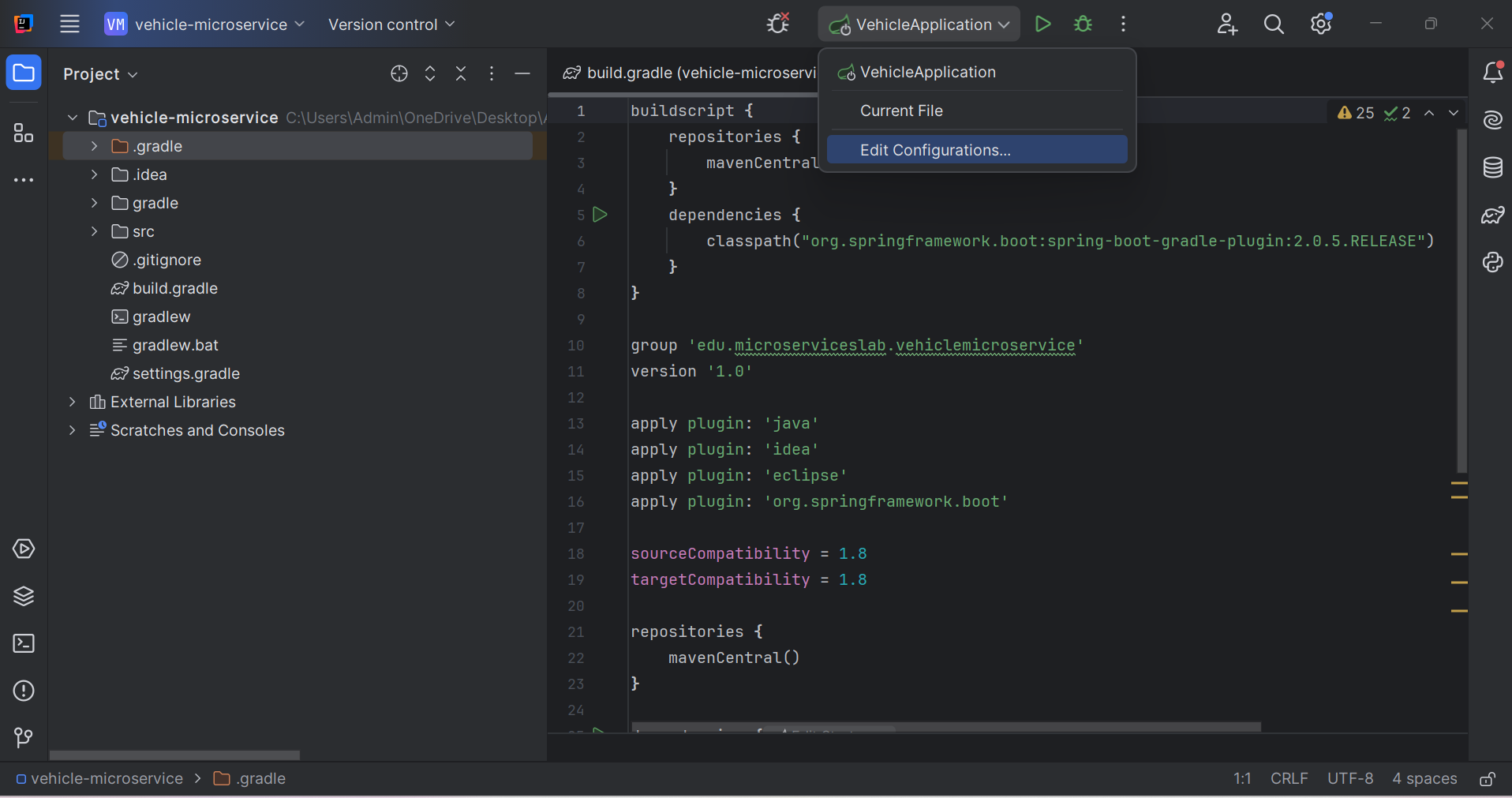


**Note:** You do not have to install *Gradle* on a system to run the project because a working implementation is packaged with the code.

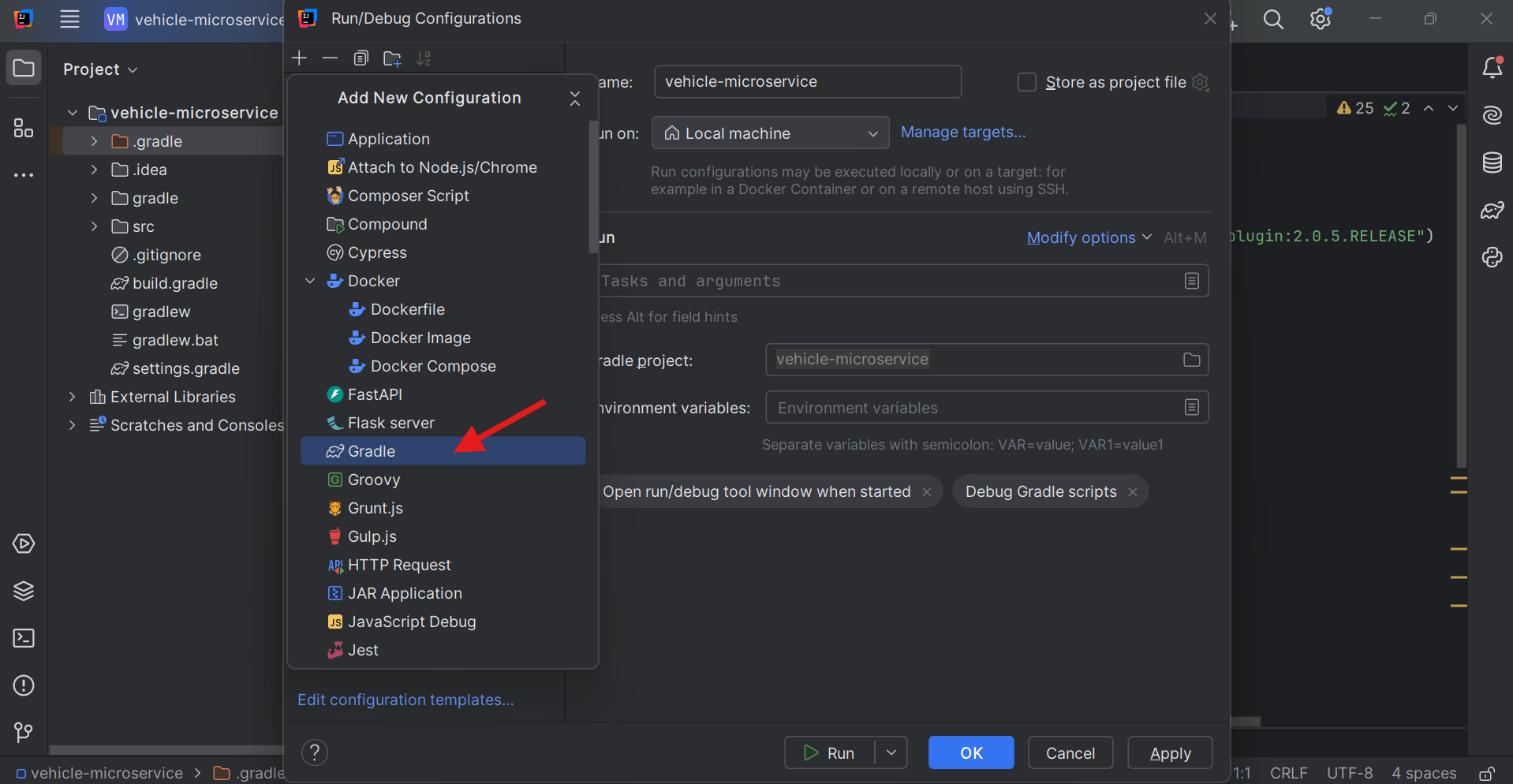
* 1. **Running the microservice using Intellij**

To run the microservice, you will need to create a new Gradle clean boot run configuration. The steps are:

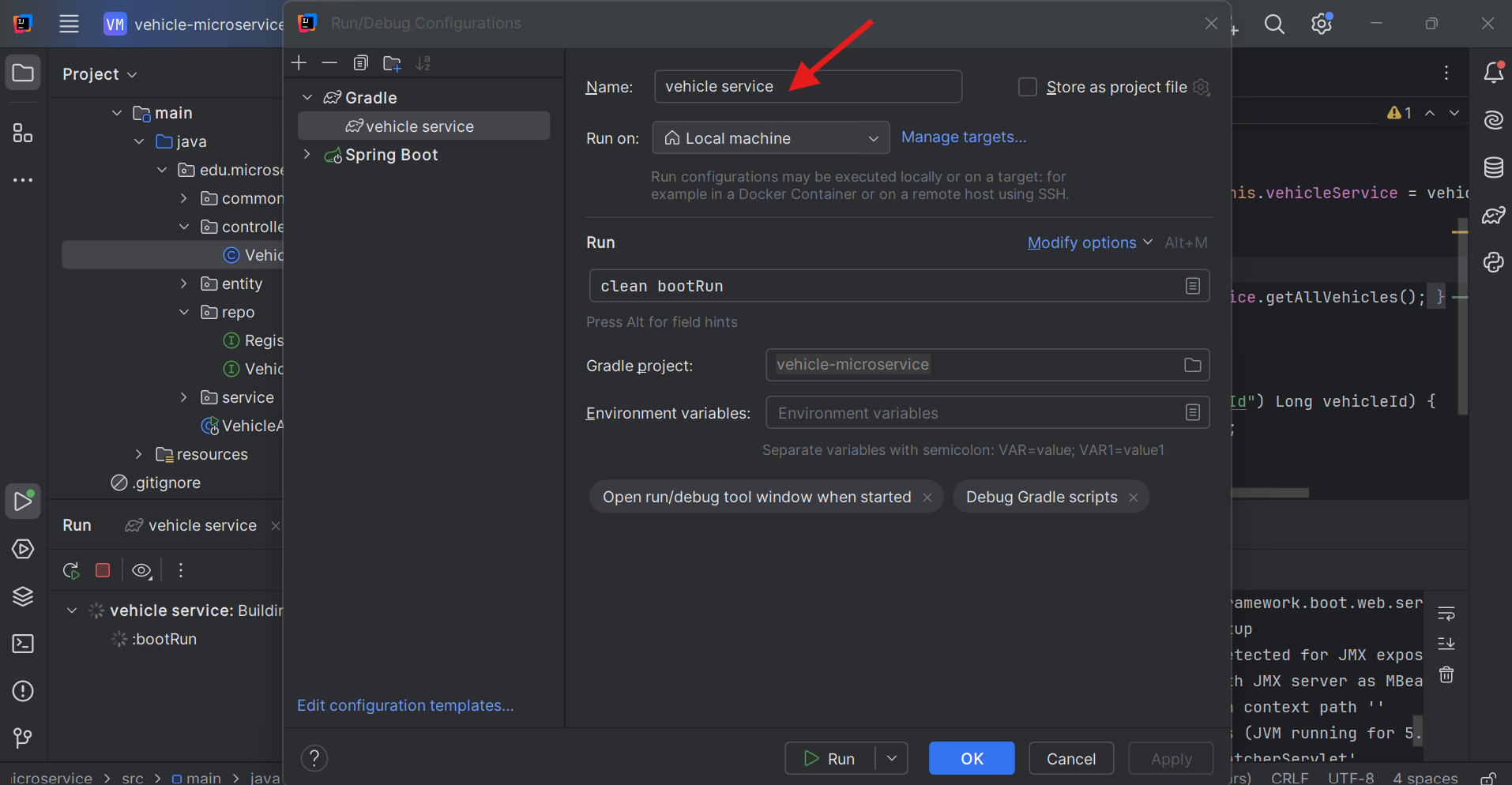
1. Create new Gradle configuration from ‘Edit Configurations’.

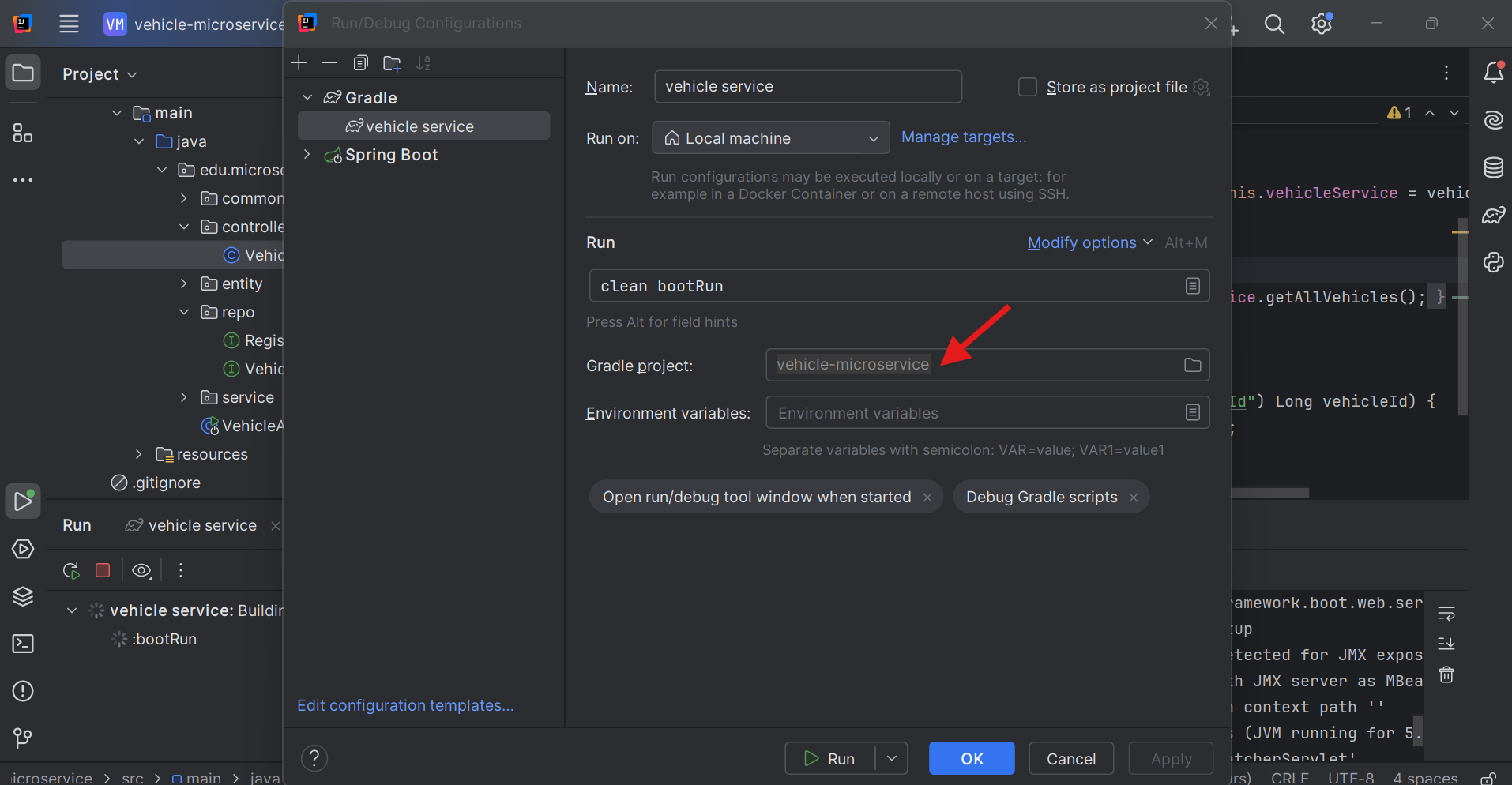


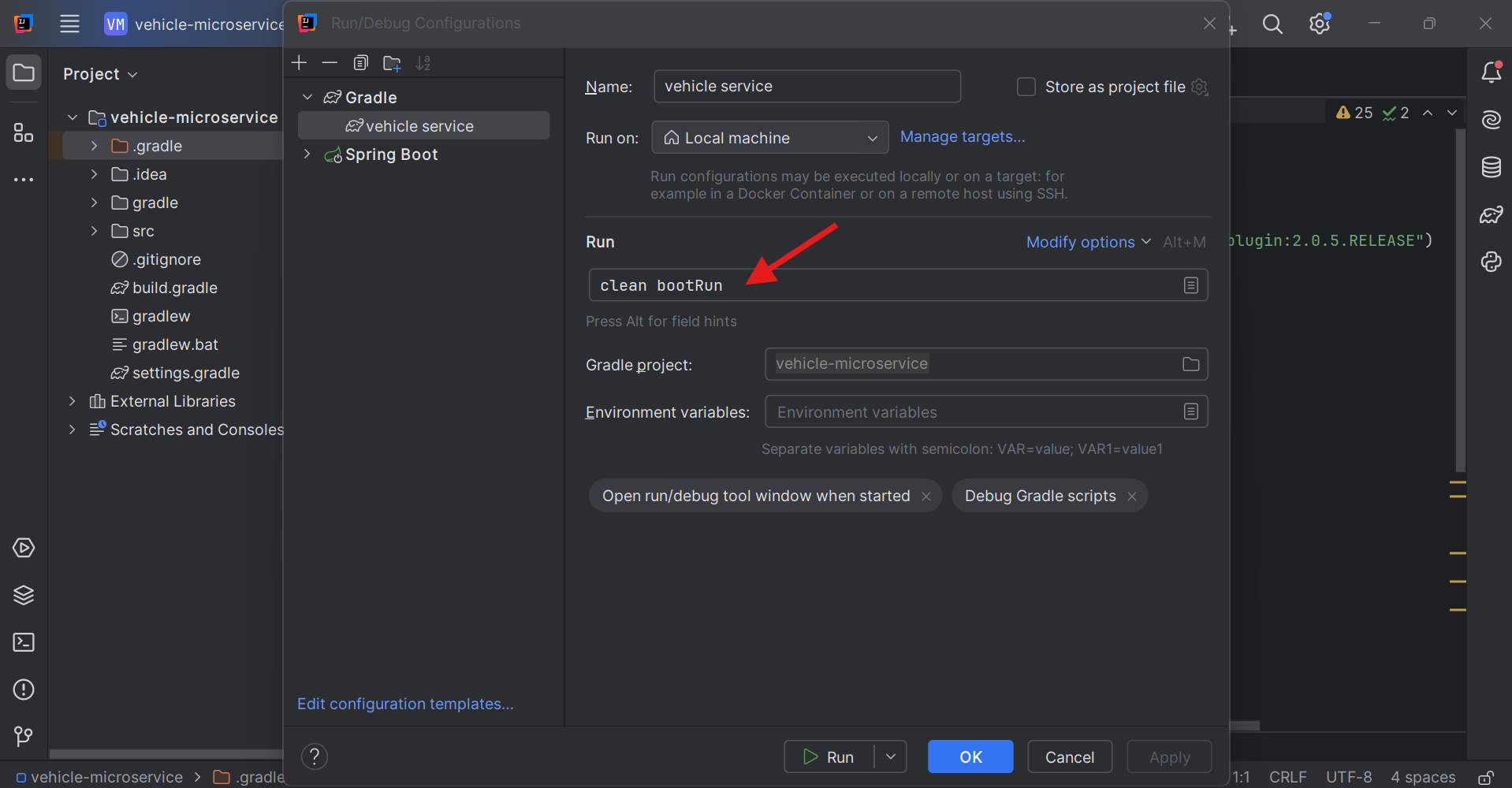
1. Click on the plus icon in the top left corner and select Gradle.



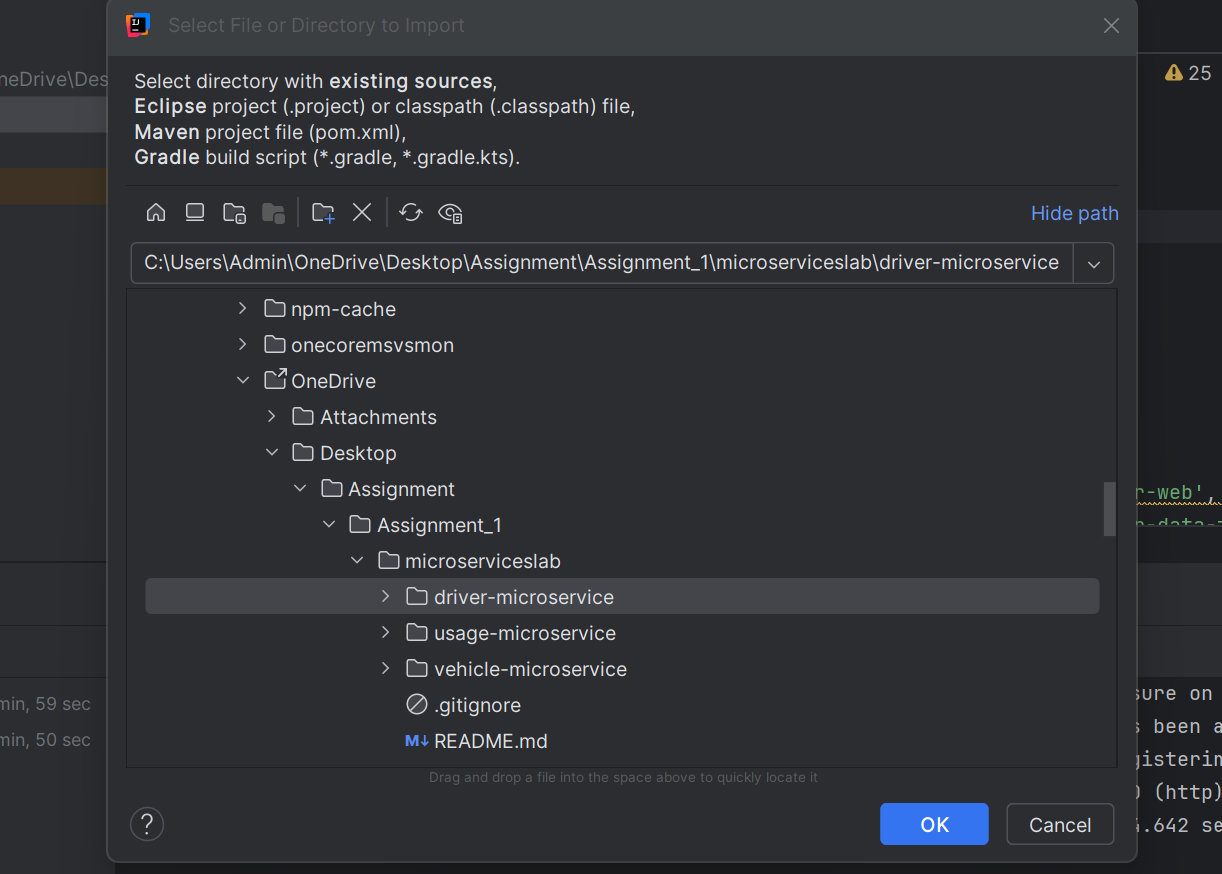
1. Enter a name for the task.



1. For Gradle project field, click on the folder icon on the right to select the correct microservice.
2. In the tasks field enter ‘**clean bootRun’**.



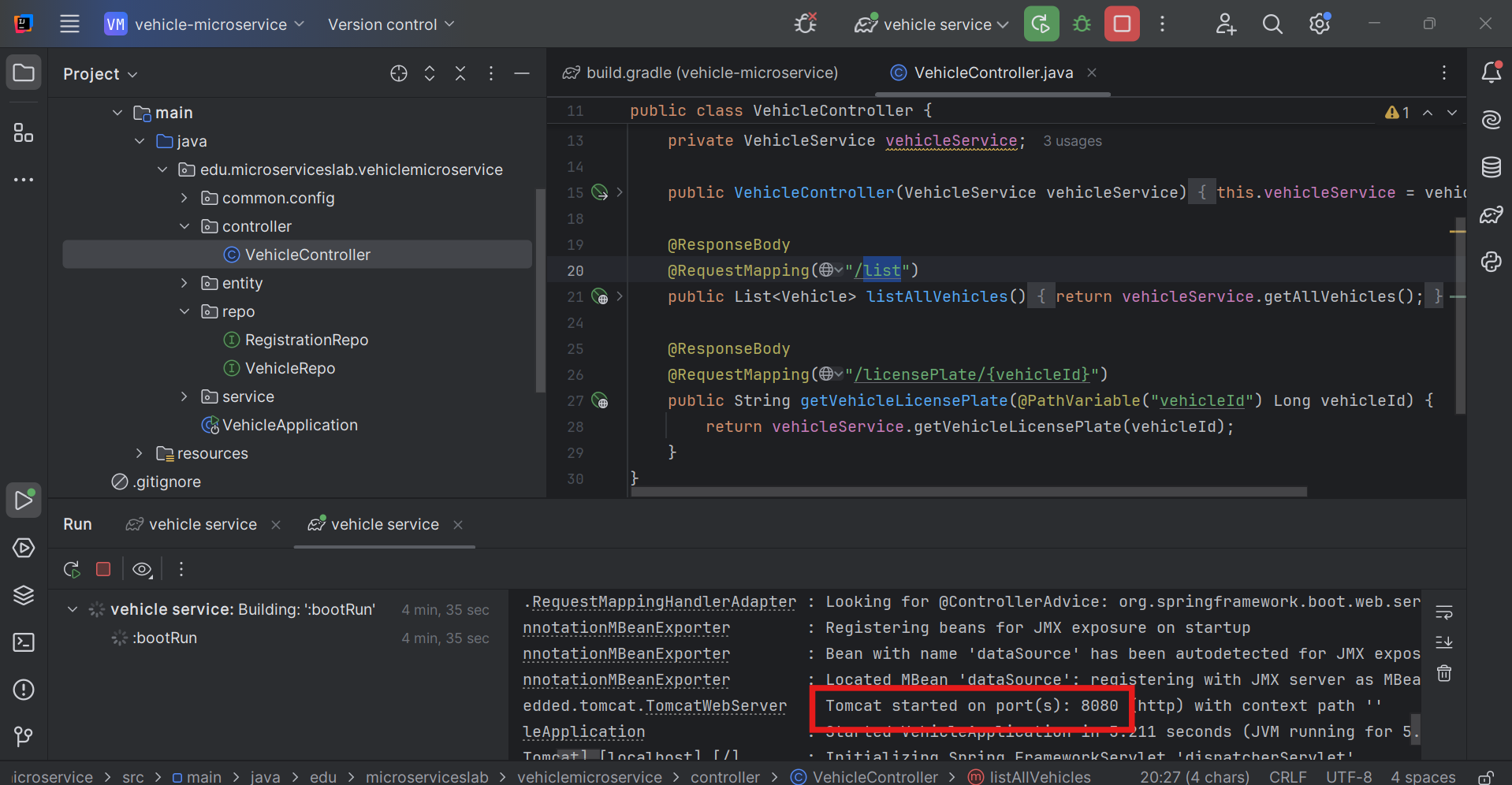
1. Leave the rest empty. Your run configuration should look like above figure, Click on Apply and then OK.
2. Run the microservice by selecting the new task you created from the drop-down beside the green run button in the toolbar and then click the run button.
3. Similarly, connect to the existing **driver** and **usage** microservice projects by Navigate to Files -> new -> Project from existing sources.

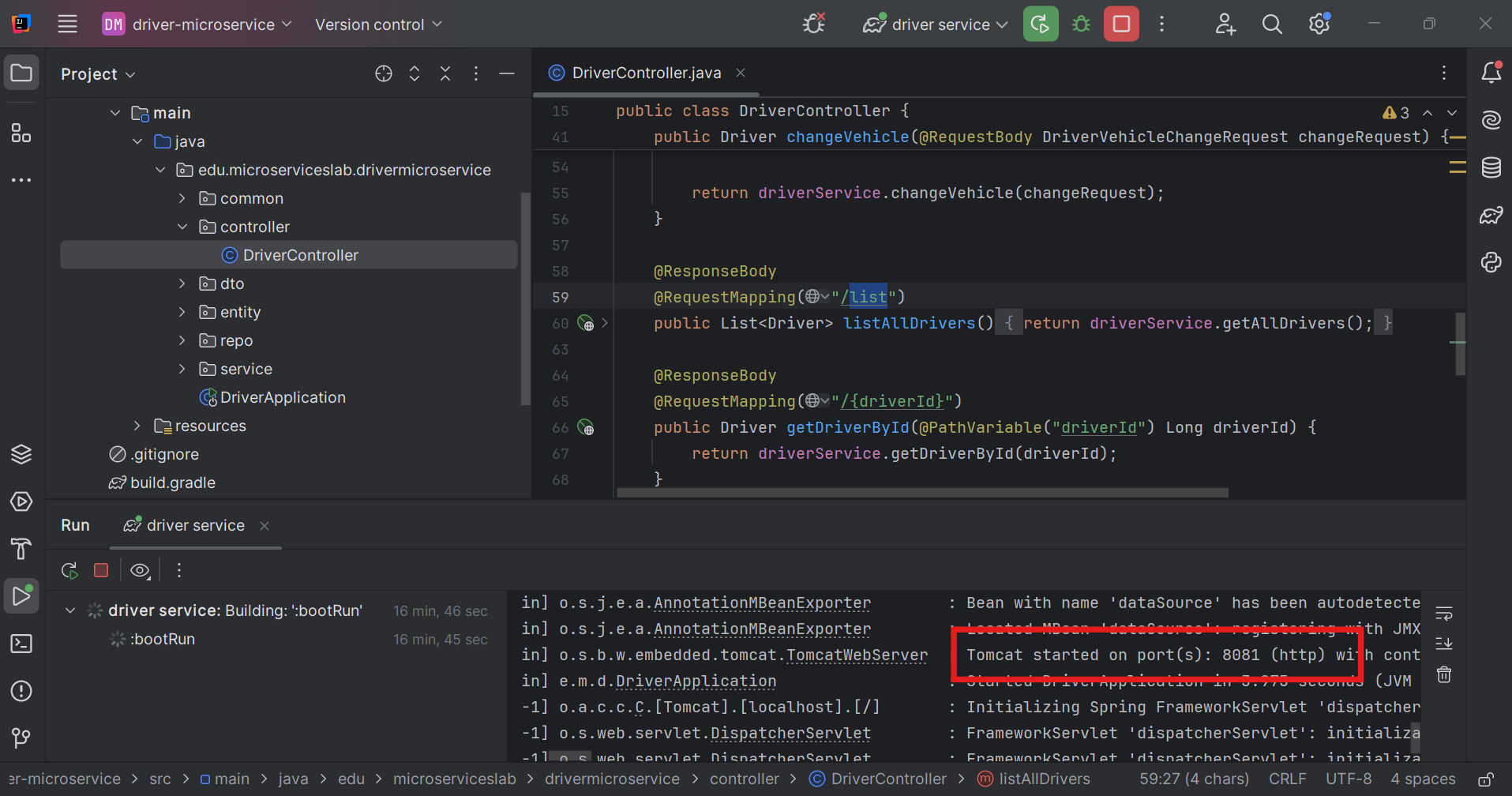


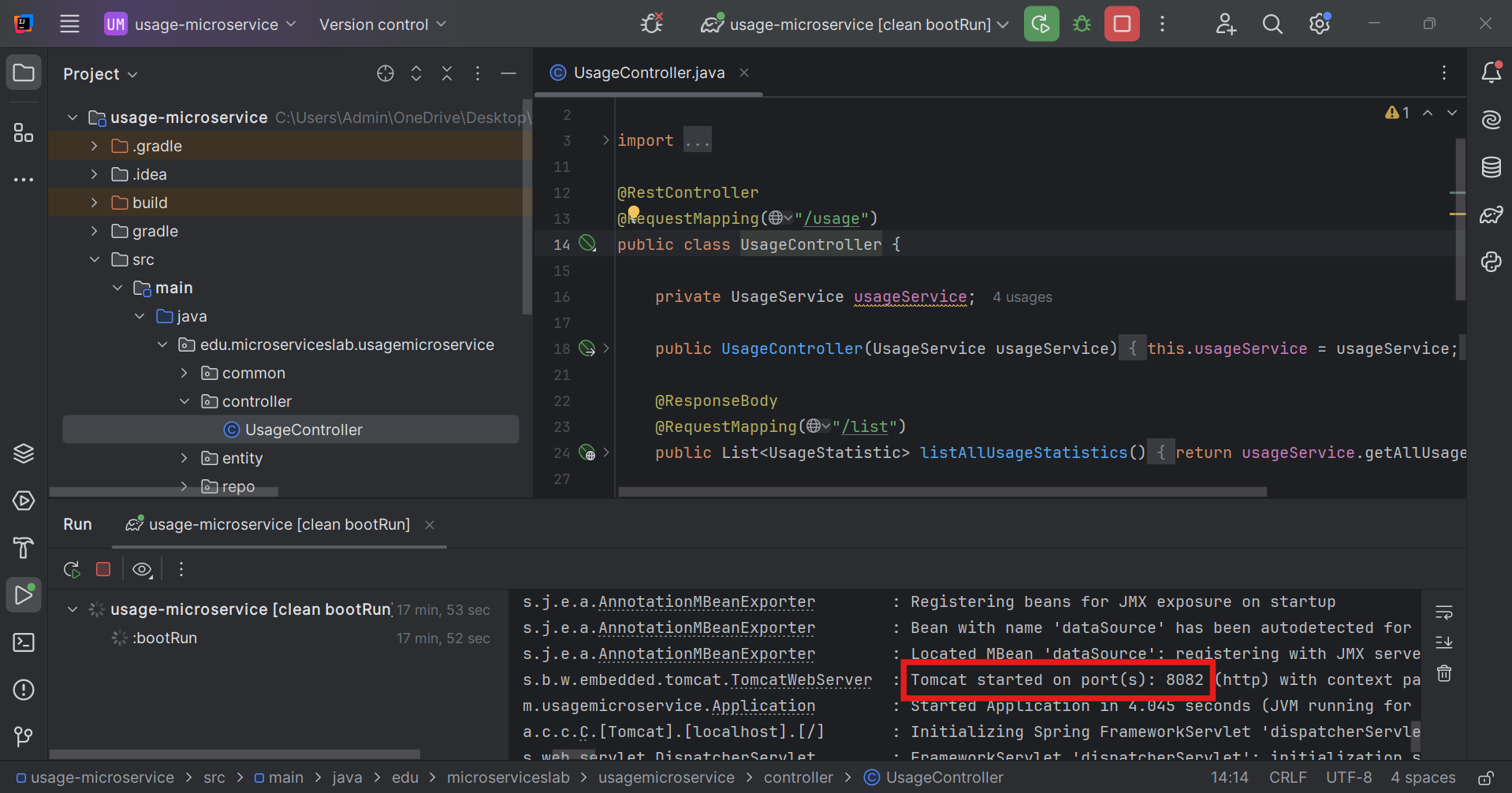
1. Select **Gradle** and open in new window. Then Edit configuration to create new profile and run those projects.

The microservices will be available to respond to requests, each on a separate port as follows:

* The vehicle-microservice will be available at http://localhost:8080.
* The driver-microservice will be available at http://localhost:8081.
* The usage-microservice will be available at http://localhost:8082.







Spring Boot Gradle projects provide several functionalities by using pre-defined core tasks including *clean*, *build*, *bootRun*, and *test*, which help the developers to create and run different code segments. The task *clean*removes all the folders that are automatically generated when a microservice is built or run. This ensures that the files generated by the previous builds are removed so they do not contaminate new builds. Debugging code when the build files are contaminated compiled can be very frustrating. It is recommended, therefore, to run the *clean* task on each build, although, it may be not necessary for some cases. The *build* task compiles the project and resolves all code dependencies. The *bootRun* task is a mix of several smaller tasks and is provided by the Spring Boot Gradle plugin. The task *clean* *bootRun*, when run for each of a set of microservices, makes them available on different ports. The developers can string the tasks together to create more complicated tasks or create new tasks for customized functionalities.

**Postman**

We use *Postman* to interact with the microservices using HTTP requests and responses. The tool should be installed from [Link](https://learning.getpostman.com/docs/postman/launching_postman/installation_and_updates/)

**3. Work description**

The assignment is composed of warming, coding, and reflection parts.

**3.1 Warming up (3 points)**

This part includes three questions:

1. (1 point) Open the project in your IDE and make sure you can get all of the microservices running. *How do the microservices communicate with each other?*
2. (1 point)} Look up for *Constructor Injection* for Java Spring on the Internet. *How are different classes passed to each together (or instantiated) in Spring*?
3. (1 point) Look up *gradle build dependencies*. Gradle is used primarily as a dependency management solution, and is used both in Spring Boot applications and in Android development. *What is a gradle 'dependency', and how does it help development?*

Write your answers in a section called *Warming up* and address each of the questions separately. Make sure you do not copy text from the Internet.

**3.2 Coding (7 points)**

This part includes 5 exercises, described below.

**Exercise 1 --Extend vehicle microservice (1 point)**

Create a Controller method for adding vehicles in the vehicle micro-service. A license registration object will be provided with the Vehicle and should also be saved.  
  
To test your code, POST a request to path  *http://localhost:8080/vehicle/add,* with JSON input  
  
*{*  
*"make":"Chevrolet",*  
*"model":"Silverado 1500",*  
*"modelYear":2009,*  
*"registration": {*  
*"licensePlate":"TOWME2",*  
*"licensedTo":"Ford Towers"*  
*}}*  
  
The JSON output should be:  
*{*  
*"id":[auto-generated],*  
*"make":"Chevrolet",*  
*"model":"Silverado 1500",*  
*"modelYear":2009,*  
*"registration": {*  
*"id":[auto-generated]*  
*"licensePlate":"TOWME2",*  
*"licensedTo":"Ford Towers"*  
*}}*

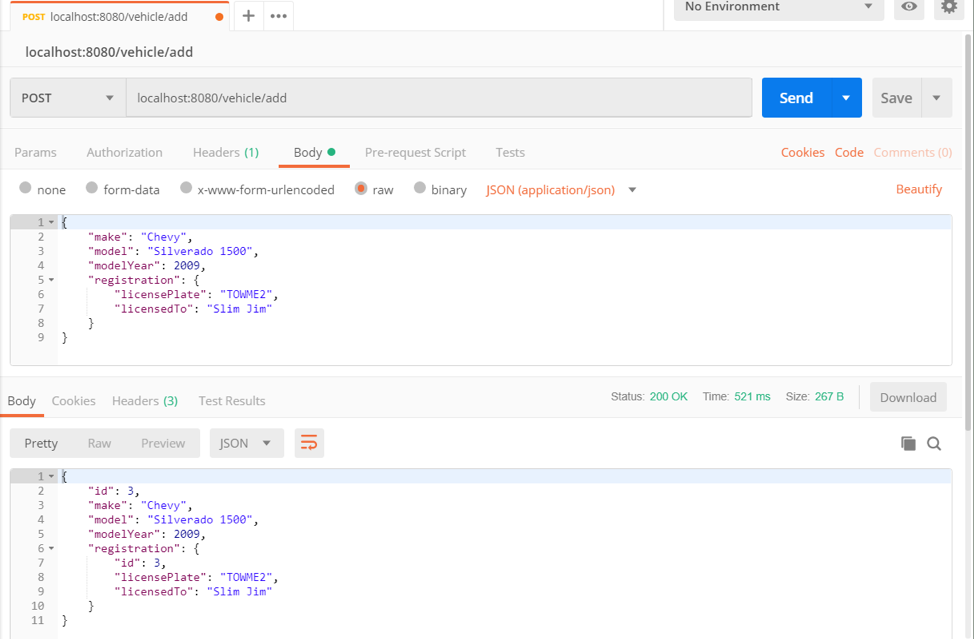


Figure 1. Testing adding a new vehicle using Postman.

Figure 1 shows the use of Postman to test the created method.  
Testing adding a new driver using Postman.

**Exercise 2 - Extend driver microservice (2 points)**

Add a *phoneNumber String* field to the *driver* object in the *driver- microservice* and update the utility methods at the bottom of the class. Don't forget to update the database schema and add phone numbers to existing data.  
  
To test your code, POST a request to path *http://localhost:8081/driver/add*, with JSON input:  
  
*{*  
*"firstName":"Jenny",*  
*"lastName":"Doe",*  
*"phoneNumber":"(555) 867-5309",*  
*"vehicleId":[new vehicle ID]*  
*}*  
  
The JSON output should be:  
*{*  
*"id":[auto-generated]*  
*"firstName":"Jenny",*  
*"lastName":"Doe",*  
*"phoneNumber":"(555) 867-5309",*  
*"vehicleId":[same vehicle ID as JSON input]*  
*}*

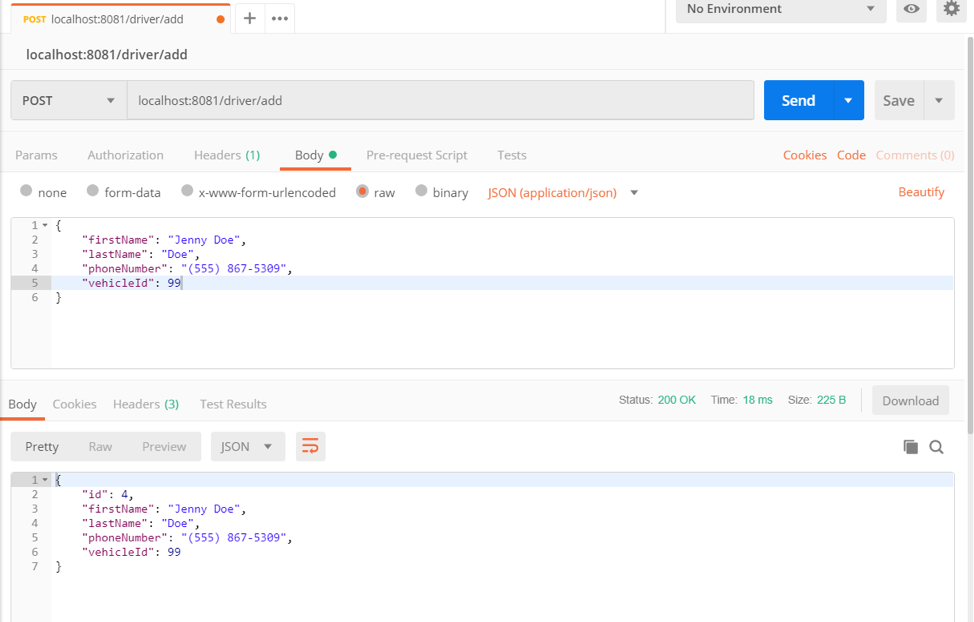
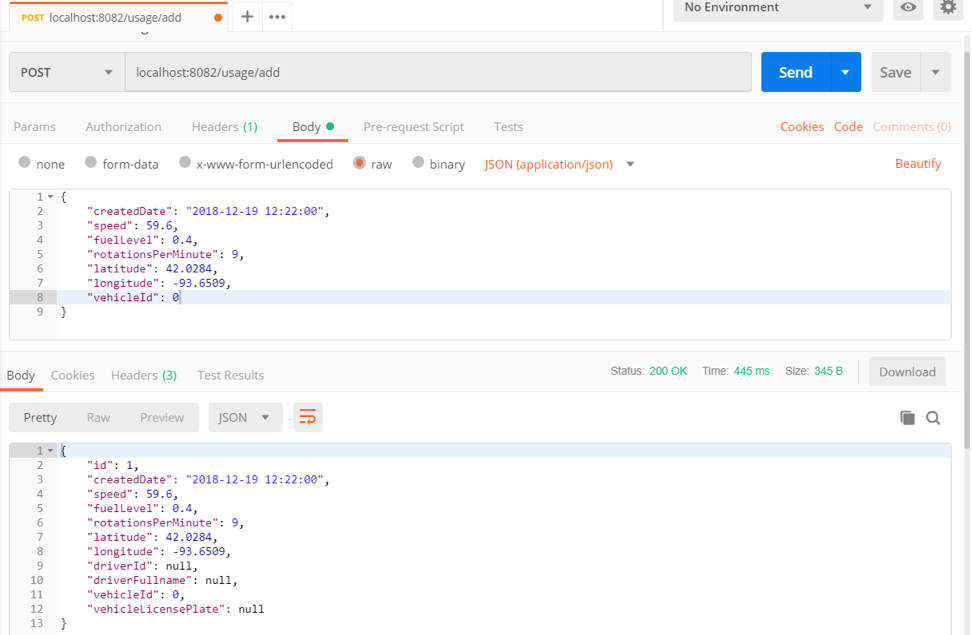


Figure 2: Testing adding a new driver using Postman.

Figure 2 shows the use of Postman to test the created method.

**Exercise 3 -- Extend usage microservice (2 points)**

Create a method for adding usage statistic data in the usage-microservice.   
  
  
Figure 3: Testing adding a new usage using Postman.

To test your code, POST a request to path http://localhost:8082/usage/add, with JSON input:

*{*  
*"createdDate":"2018-12-19 12:22:00",*  
*"speed":59.6,*  
*"fuelLevel":0.4,*  
*"rotationsPerMinute":9,*  
*"latitude":42.0284,*  
*"longitude":-93.6509,*  
*"vehicleId":0*  
*}*  
  
The JSON output should be:  
*{*  
*"id":[auto-generated],*  
*"createdDate":"2018-12-19 12:22:00",*  
*"speed":59.6,*  
*"fuelLevel":0.4,*  
*"rotationsPerMinute":9,*  
*"latitude":42.0284,*  
*"longitude":-93.6509,*  
*"driverId":0,*  
*"driverFullname":"Gigi Wentworth (877-CASH-NOW)",*  
*"vehicleId":0,*  
*"vehicleLicensePlate":"MPASTA"*  
*}*

Figure 3 shows the use of Postman to test the created method. In some scenarios, your code although correct, returns NULL for the vehicle information, as in the figure. We consider both output as correct.

**Exercise 4 - Enforce consistency (2 points)**

Improve your code to enforce consistency by  adding checks for (1) null statistic data and (2) existence of driver information and (3) vehicle information when appropriate.  
  
**Exercise 5  - Optional (2 extra credit points)**

Implement Swagger and provide Swagger-generated REST API documentation for each microservice.

Please make a note in your report that you did the exercise for the extra credit question, if that applies.

**3.3 Reflection (4 points)**

1) Assume that you discovered a bug in the vehicle microservice. (2 points)

a) Could the bug cause a crash of all the three microservices?

b) Which microservice(s) do you need to modify to fix the bug?

c) Assume that you fixed the bug, do you need to deploy all the three microservices again?

2) Would your answers for three questions above be the same if the application were monolithic? (1 point)

3) What lessons did you learn from the assignment? (1 point)

**4. Submission requirements and grading criteria**

Submit a zip file that includes a report answering questions 1-3 and the code of each microservice starting with the root folder.

**5. Acknowledgment**  
Special thanks to Thomas Stackhouse and Nischay Venkatram for helping to develop this assignment.