README

Spring Boot – REST API Implementation

Prerequisite/info needed to run this Project:

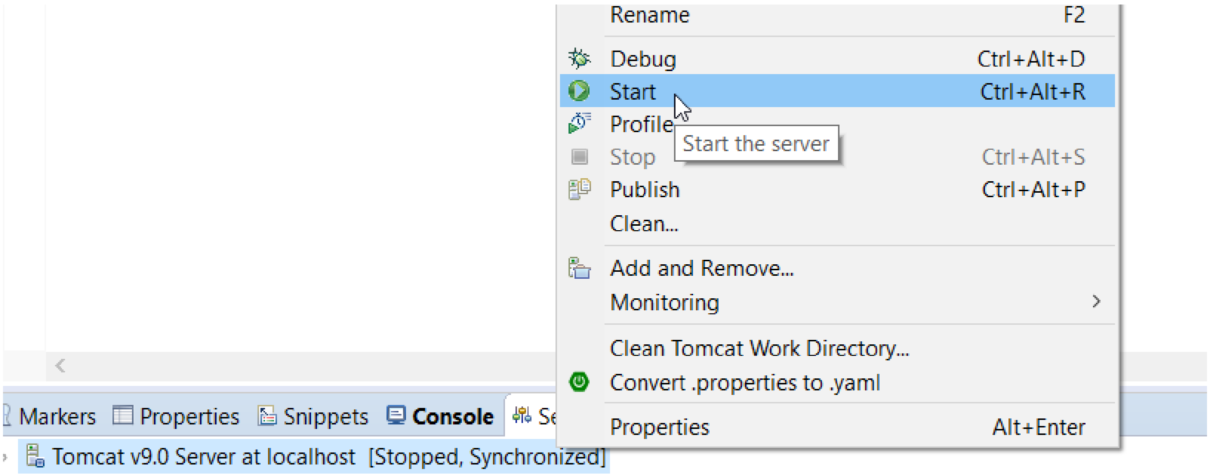
- **JDK 1.8** is used for this project. You should have this version or above, installed in your machine.
- **IDE** : This project is built using **Spring Tool Suite IDE** (STS). One can use Eclipse IDE too.
- **Server**: **Tomcat Server** is used. With STS, Tomcat is embedded by default in spring project. For Eclipse, install the Tomcat Server.
- **Database used: H2 database** (in-memory database) is used. In every rerun of the application, the previous data from this database is erased.
- **Dependencies :** These are managed via **Maven** pom.xml file

Import the Project : **(Project Name- SpringBootAssignment_Digicides)**

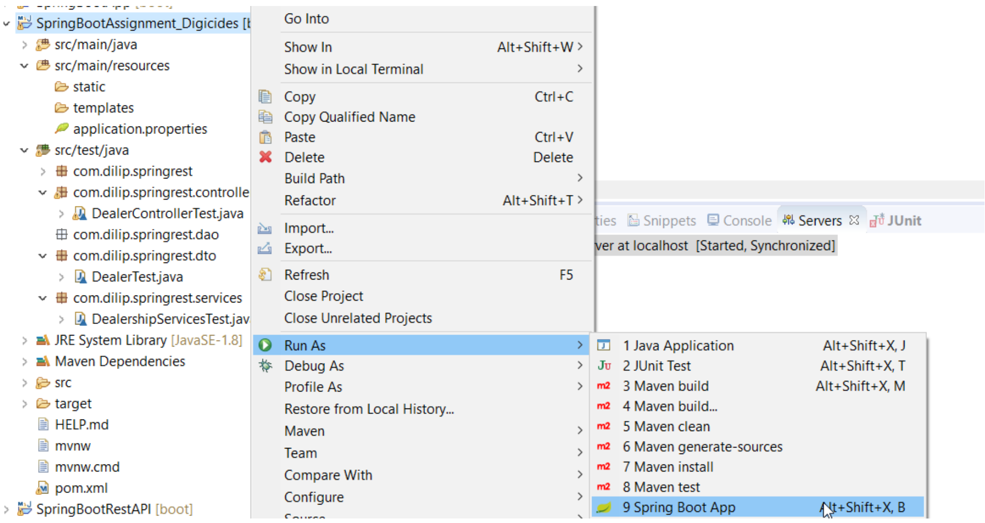
1. Clone/download the project from Github: Click here
2. After cloning the project to your local system , you have to import it in Spring Tool Suite IDE.
3. Go to File-> Import-> Select Existing Maven Project-> Browse this cloned project from your local machine ->Finish

How to Run the application:

1. Start the Tomcat server.



2. Now Run the Application as Spring Boot App: Click on project ->Run As -> Spring Boot App

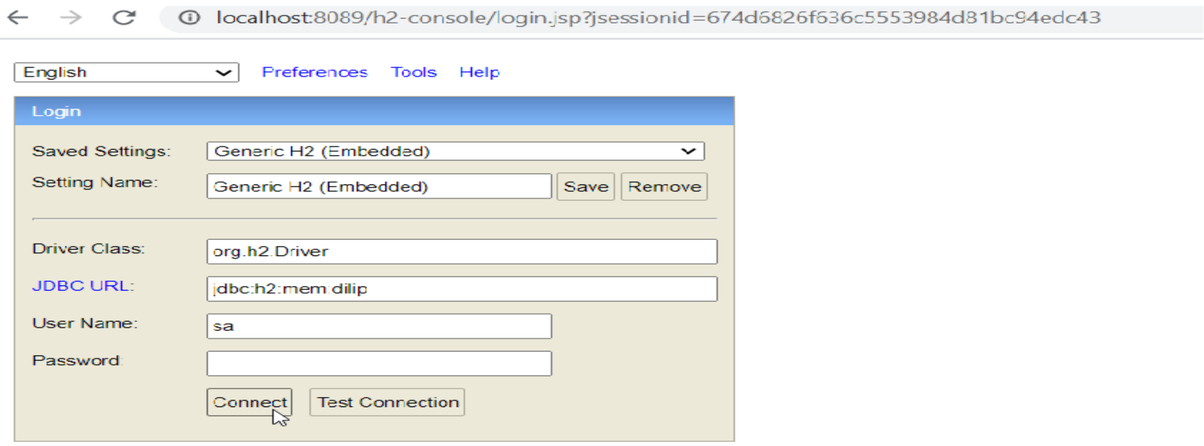


Note :

1. In this project **server port** is set to **8085**. If this port is shown as busy, then try to change the port to another one like 8086 or other in application.properties file under src/main/resources/ application.properties)
2. application.properties file also contains the H2 database configurations. These should not be changed and kept as such.

Calling the APIs:

1. **Client used**: One can use either **system browser** or **postman** as a client to fire the query/ APIs.
2. **Open the connection to H2 database:**
   a. Type the following in your browser:
      localhost:8089/h2-console and Click connect



localhost:8089/h2-console/login.jsp?jsessionid=674d6826f636c5553984d81bc94edc43

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)

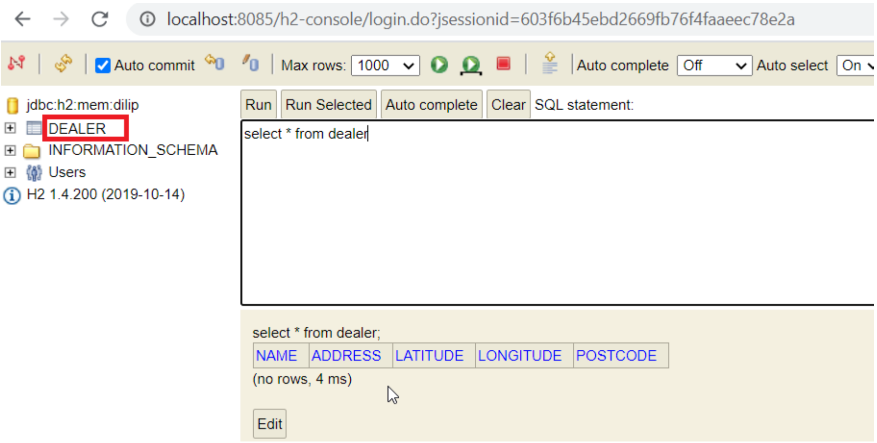
Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:dilip

User Name: sa

Password:

We can see the DEALER table created there



localhost:8085/h2-console/login.do?jsessionid=603f6b45ebd2669fb76f4faaeec78e2a

☑ Auto commit Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:mem:dilip
- DEALER
- INFORMATION_SCHEMA
- Users
- H2 1.4.200 (2019-10-14)

SQL statement:

```
select * from dealer
```

```
select * from dealer;
NAME  ADDRESS  LATITUDE  LONGITUDE  POSTCODE
(no rows, 4 ms)
```

API 1: Configuring the Dealership Details:

1. This API adds the dealers information in to the database (H2 database)
2. Call the API as below in browser or postman:

   localhost:8085/dealer?name=Dilip&address=Madhapur&postcode=500081

   (The API takes 3 query parameters as given in requirements in the assignment.)

3. Check in H2:
   a. Fire the query : select * from dealer and Click Run

   You will see the new row is added with above values of query parameters alongwith latitude and longitude in database

   Note: **name** is kept as primary key in the **dealer** table.

Likewise , you can call the API again with different values of 3 parameters.

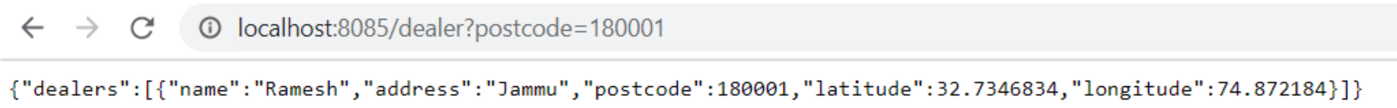
API 2:

a)Finding the nearest dealership details via postcode

1. Call the API as below in browser or postman:

   localhost:8085/ dealer?postcode=180001

2. This will return/show you the nearest dealer details fetched from H2 database in JSON format.



localhost:8085/dealer?postcode=180001

{"dealers":[{"name":"Ramesh","address":"Jammu","postcode":180001,"latitude":32.7346834,"longitude":74.872184}]}

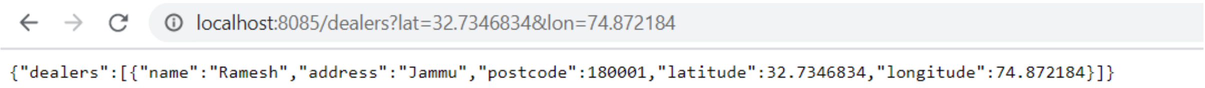
b) Finding the nearest dealership details via latitude and longitude

1. Call the API as below in browser or postman:

   localhost:8085/dealers?lat=32.7346834&lon=74.872184

   Note: We have used dealers here (In above we used dealer, so type carefully)

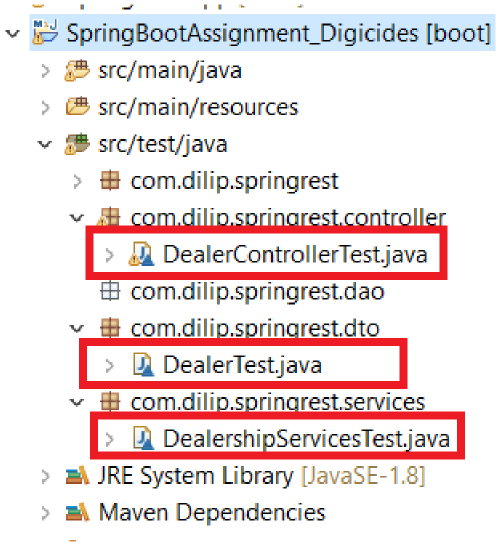
2. This will return/show you the nearest dealer details fetched from H2 database in JSON format.



```
{"dealers":[{"name":"Ramesh","address":"Jammu","postcode":180001,"latitude":32.7346834,"longitude":74.872184}]}
```

Junit Test cases:

Some test cases are also made for testing the API functionality.


SpringBootAssignment_Digicides [boot]
src/main/java
src/main/resources
src/test/java
com.dilip.springrest
com.dilip.springrest.controller
DealerControllerTest.java
com.dilip.springrest.dao
com.dilip.springrest.dto
DealerTest.java
com.dilip.springrest.services
DealershipServicesTest.java
JRE System Library [JavaSE-1.8]
Maven Dependencies

Run the test cases as:

Right click the test case -> Run as -> Junit Test