Table of Contents

[**Purpose of this document:** 2](#_Toc19486256)

[**Problem Statement:** 2](#_Toc19486257)

[**Expected Result:** 2](#_Toc19486258)

[**Overview of the System:** 2](#_Toc19486259)

[**Assumptions:** 2](#_Toc19486260)

[**Improvements:** 2](#_Toc19486261)

[**Tools Used:** 3](#_Toc19486262)

[**Architecture and Flow:** 3](#_Toc19486263)

[**Service End Points:** 3](#_Toc19486264)

[**Steps followed for Building Application:** 10](#_Toc19486265)

[**Source Code:** 12](#_Toc19486266)

[**Steps to run the application:** 12](#_Toc19486267)

[**Testing the application:** 16](#_Toc19486268)

[**Running Test cases from IDE:** 16](#_Toc19486269)

[**Running Test cases from Command Prompt:** 18](#_Toc19486270)

## **Purpose of this document:**

This document contains the high level design details of User Management REST Service. It includes the architecture of the system and use cases.

## **Problem Statement:**

The objective is to implement a rest-service which is able to:

* create new user with contact data
* return user by id
* return user by name
* add additional mail/phone data
* update existing mail/phone data
* delete user

The data objects are defined as followed:

User:

id: <int>

lastName: <string>

firstName: <string>

emails: List<Email>

phoneNumbers: List<PhoneNumber>

Email:

id: <int>

mail: <string>

PhoneNumber:

id: <int>

number: <string>

## **Expected Result:**

Implement a rest service with multiple end points specified in the problem statements.

## **Overview of the System:**

User Management system maintains the user contact information. It maintains the user contact information like firstname, lastname and multiple emails and phone numbers.

## **Assumptions:**

1. Security for the actions (end points) is not provided.
2. Validations are not taken care due to time constraint.
3. Complete exception handling is not done.

## **Improvements:**

1. Security can be provided with any one of the authentication mechanisms. Spring security is one good choice to provide the security for the urls based on the roles of the user.
2. Validations can be taken care, not null values, unique values.
3. Exception handling can be taken care in better way by creating the custom exceptions.

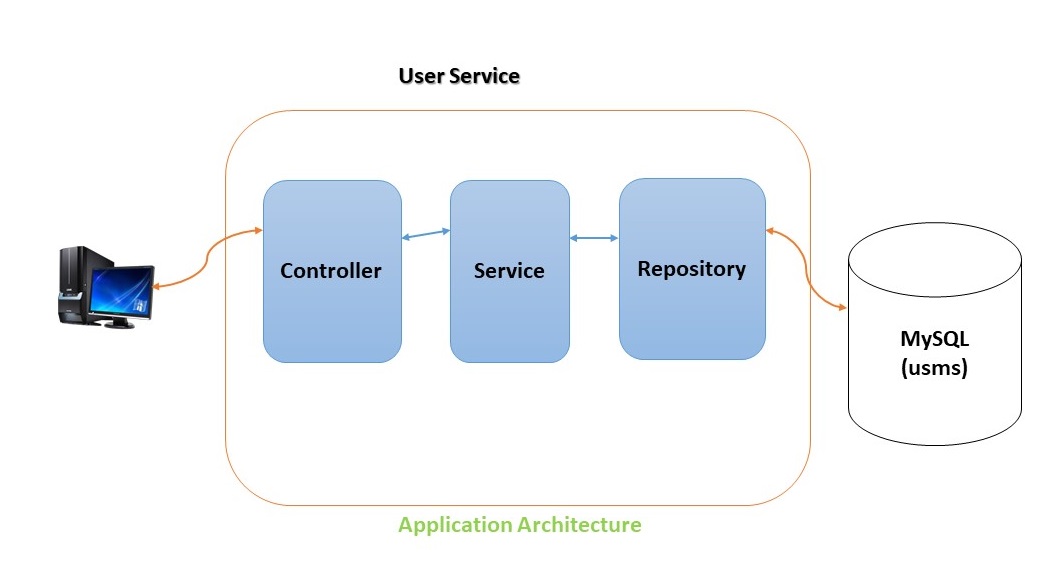
## **Tools Used:**

1. Java 1.8
2. Spring Boot 2.x
3. Spring data JPA+Hibernate
4. IntelliJ IDEA
5. Mysql 11.x Database
6. Maven.

## **Architecture and Flow:**

User management service built as multi layered application for clear separation of responsibilities. **Followed MVC architecture** to differentiate the backend logic and front end logic.

I have chosen **Spring Boot 2.x** for MVC web framework and **Spring Data JPA** as a layer to do database operations.



User Service contains below layers as show in the above architecture diagram.

1. Controller (Web layer)
2. Service (For complete business logic)
3. Repository (Database related logic)

Data given to the application will be stored in the My SQL database.

## **Service End Points:**

For all requests, created the generic response with the following fields.

**Success:** This fields indicates, whether the request is successful or not. Possible values are

True - > success

False - > failed

**Message:** This field contains the success or failure message.

**Data:** This field contains the actual data (user data or empty if it is update/delete).

#### **1. findById**

**URL:** [http://localhost:8080/users/{userId}](http://localhost:8080/users/%7buserId%7d)

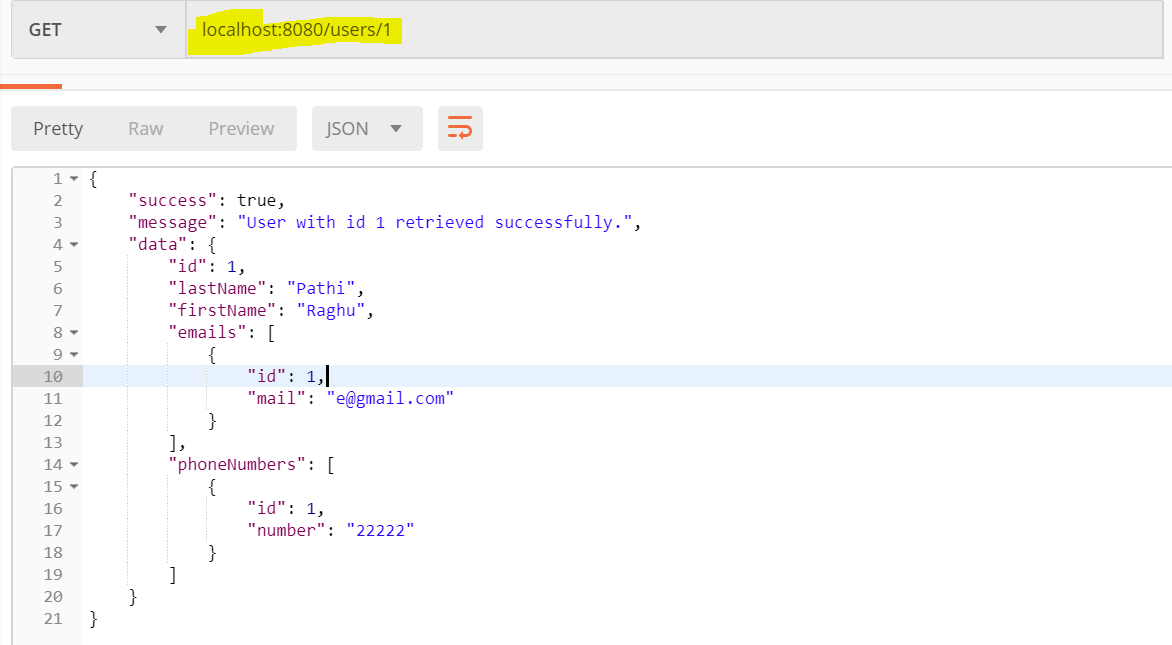
**Method:** GET

**Description:** This is responsible for find the User by the given id. User has to provide the {userId} for which user has to search.

**Sample Input:**

[**http://locahost:8080/users/1**](http://locahost:8080/users/1)

**Sample Output:**



#### **2. Create new user with contact data:**

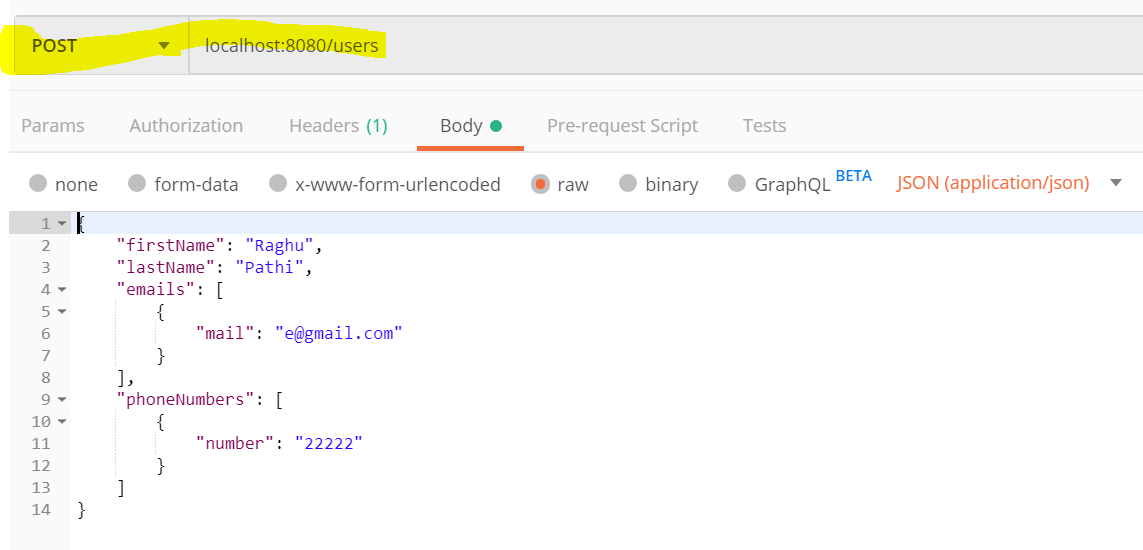
**URL:** <http://localhost:8080/users>

**Method:** POST

**Description:** This is responsible to save the user data with contact information. We have to give the json data which needs to saved. Format should be as defined in the sample input.

**Sample Input:**

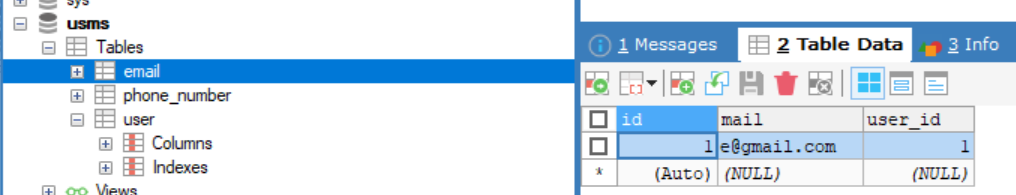
[**http://locahost:8080/users**](http://locahost:8080/users)

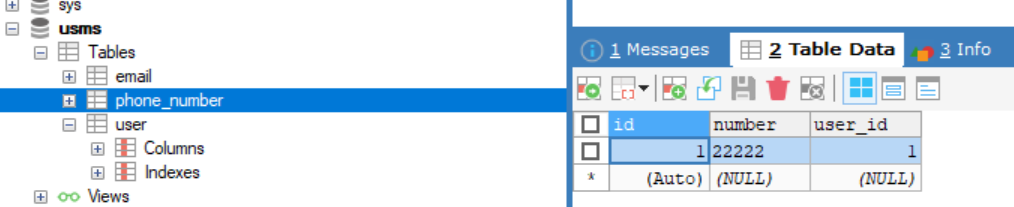


**Sample Output:**









#### **3. return User by Name:**

**URL:** [http://localhost:8080/users?name=”Raghu](http://localhost:8080/users?name=) Pathi”

**Method:** GET

**Description:** This is responsible to retrieve the User based on the given name. In the url name is the request parameter. This name can contain two words separated by space. First word treated as first name and second word treated as last name. Last name is optional that is second word is optional.

If both words are given it will search for exact matched(case will be ignored), if one word given then it will search only in first name.

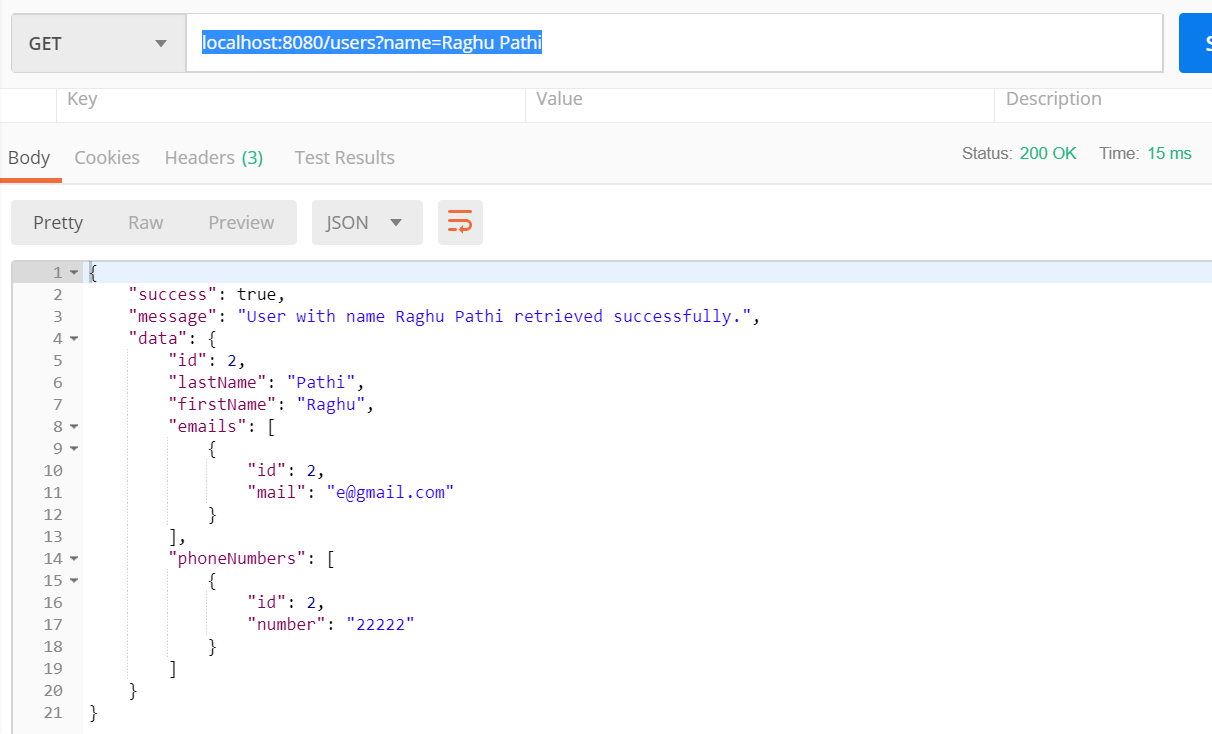
Raghu Pathi 🡺 FirstName LastName

**Sample Input:**

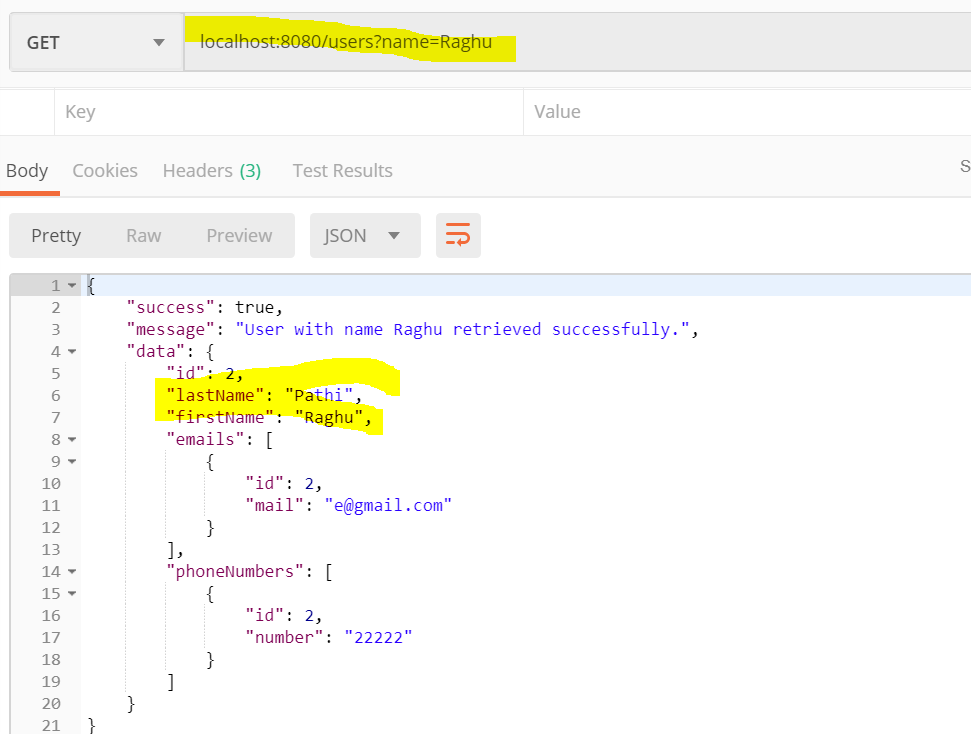
[**http://locahost:8080/users?name=Raghu**](http://locahost:8080/users?name=Raghu) **Pathi**

**Sample Output:**

[**http://locahost:8080/users?name=Raghu**](http://locahost:8080/users?name=Raghu) **Pathi**



[**http://locahost:8080/users?name=Raghu**](http://locahost:8080/users?name=Raghu)



#### **4.** **add/update additional mail/phone:**

**URL:** [**http://**localhost:8080/users/{userId}/contact](http://localhost:8080/users/%7buserId%7d/contact)

**Method:** PUT

**Description:** This service end point responsible for updating/adding additional mail/phone data to existing user. We have to provide the userId as shown in the above url.

If we provide the IDS (primary key in the email, phonenumber table) of email/phone then it will update the existing data. With this end point we can upadate the user firstname and last name aswell.

If we don’t provide the IDs then it will insert the new email/phone number for the given user.

**Sample Input:**

**Sample Input to update Phone/Email for existing user:**

{

"lastName": "Pulishetti",

"firstName": "Raghupathi",

"emails": [

{

"id": 1,

"mail": "abc@gmail.com"

}

],

"phoneNumbers": [

{

"id": 1,

"number": "111"

}

]

}

**Sample Input to insert Phone/Email for existing user:**

{

"lastName": "Pulishetti",

"firstName": "Raghupathi",

"emails": [

{

"mail": "abc@gmail.com"

}

],

"phoneNumbers": [

{

"number": "111"

}

]

}

**Sample Output:**



#### **5. delete User:**

**URL:** <http://localhost:8080/users/1>

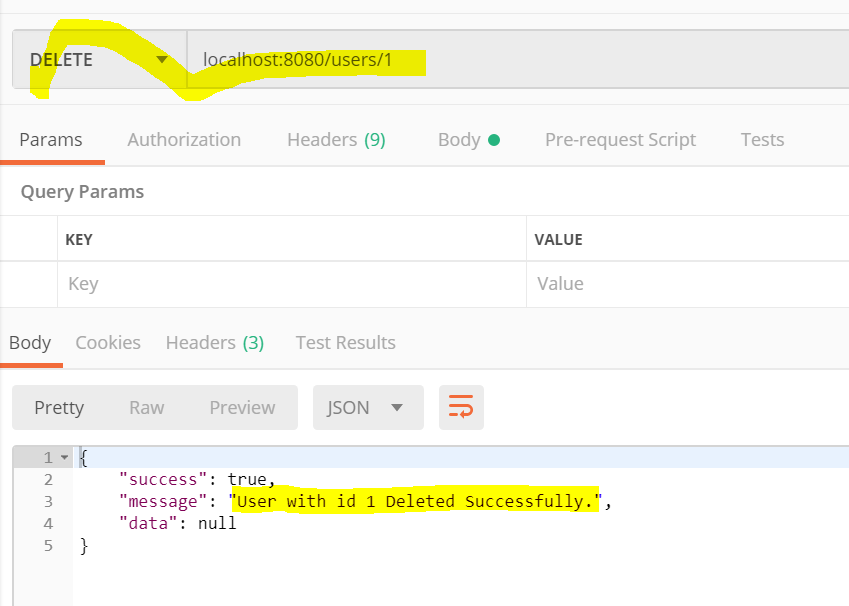
**Method:** DELETE

**Description:** This is responsible delete the existing user. We have to provide the userId for which we have to delete the data. It deletes the phone, email data as well.

**Sample Input:**

[**http://locahost:8080/users/1**](http://locahost:8080/users/1)

**Sample Output:**

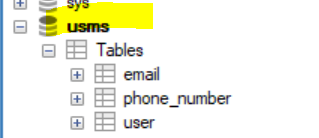
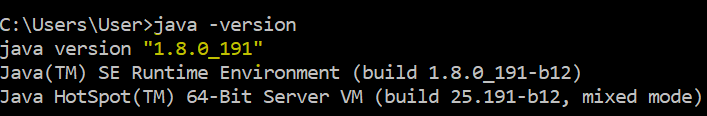


## **Steps followed for Building Application:**

**Step 1:**

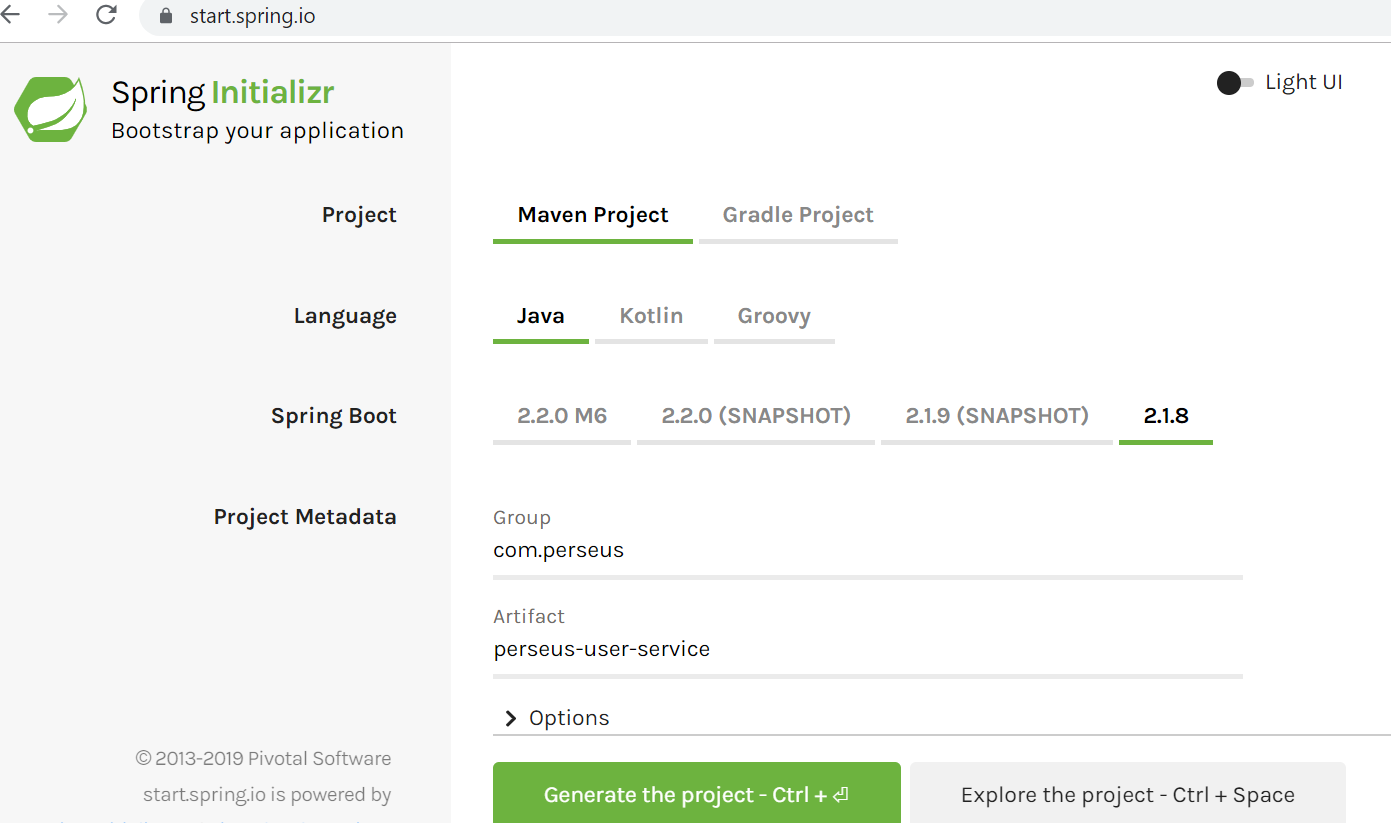
Checked/installed for all the prerequisite software’s available on system. Prerequisite software’s are.

1. Java 1.8
2. Maven Build Tool
3. Intellij IDEA
4. Mysql with **usms** schema

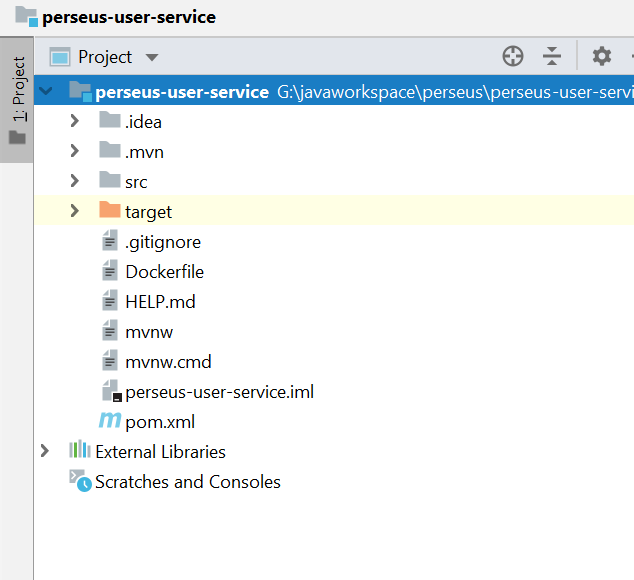
**Step 2:**

Create the spring boot application by providing application information and adding the web, mysql, jpa dependencies and download from the spring initializer website.



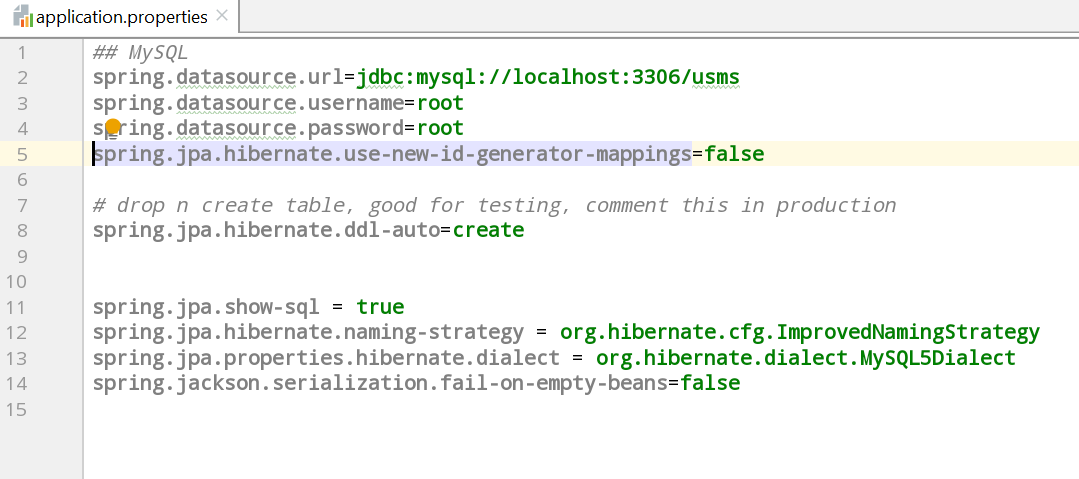
**Step 3:**

Import the created application into the IntelliJ IDE using import options while IntelliJ startup.



**Step 5:**

Open the **application.conf** file and add the database configuration properties in **application.conf** file. It will give the flexible to change the database urls, usernames and passwords with ease.



**Step 6:**

Create the entity, controller, service, repository, dto classes with necessary methods to provide the user-service end points.

## **Source Code:**

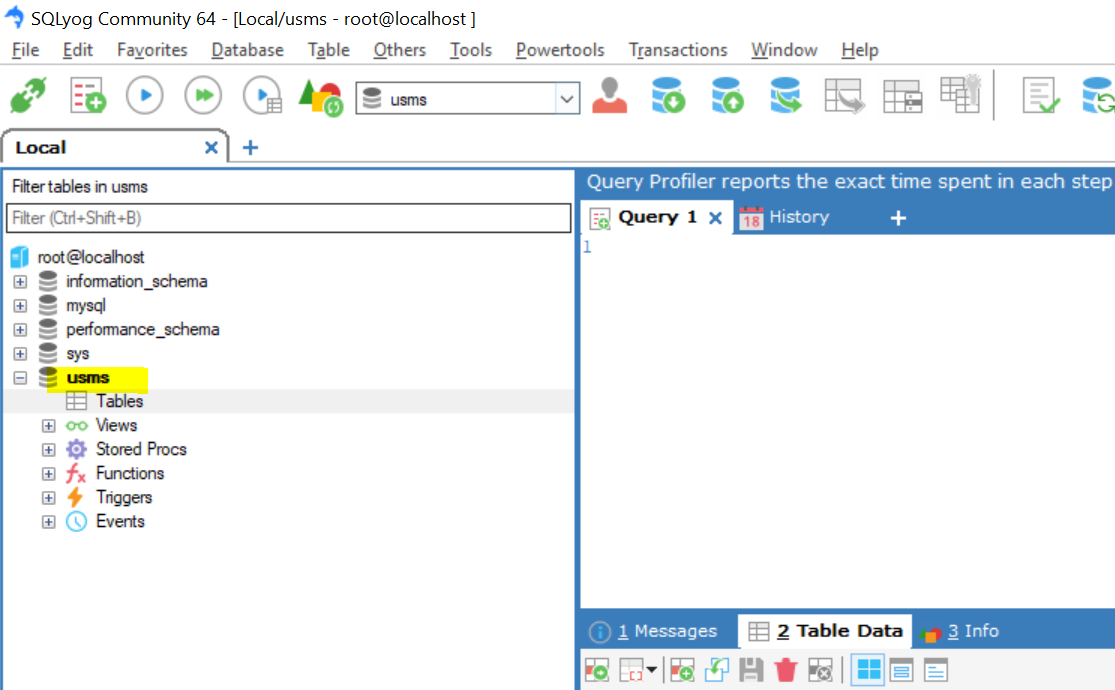


## **Steps to run the application:**

**Step 1:**

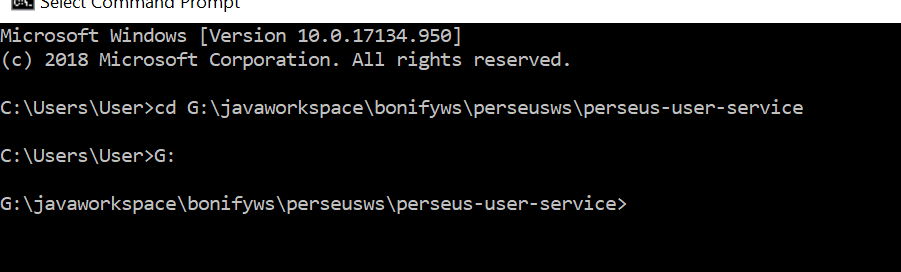
Make sure that all the prerequisites **(Java 1.8, Maven, Mysql with usms database)** are on system. Clone the git URL into the local or unzip the above attached folder into the local.

[**https://github.com/raghupulishetti/perseus-user-service.git**](https://github.com/raghupulishetti/perseus-user-service.git)



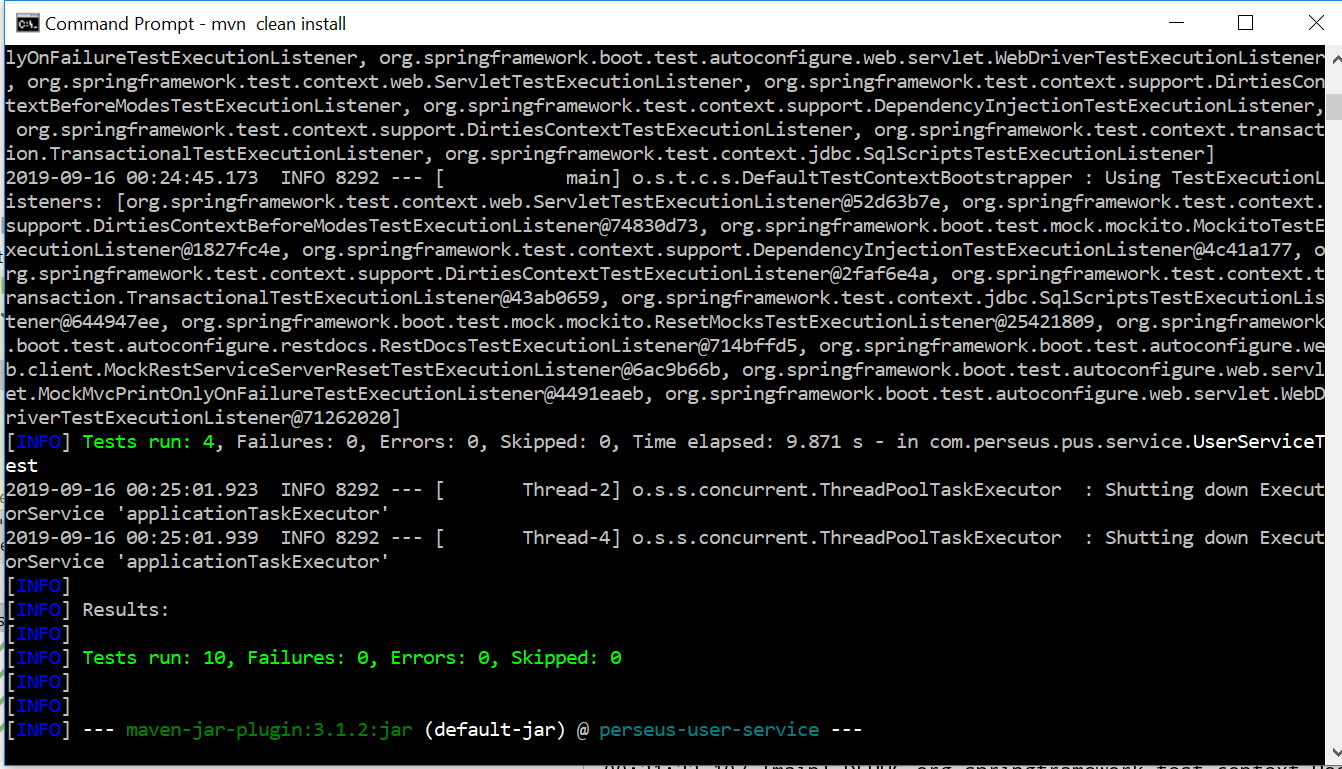
**Step 2:**

We can run the application from IDE or from command prompt. To run from command prompt, go to the project folder where we cloned it from git or upzipped folder from above zip file.



**Step 3:**

Run **mvn clean install** command to build the application. It will build the application along with test cases.



**Step 4:**

Run the jar using **java -jar perseus-user-service-0.0.1-SNAPSHOT.jar** by moving into the target folder.



Or Run the below command

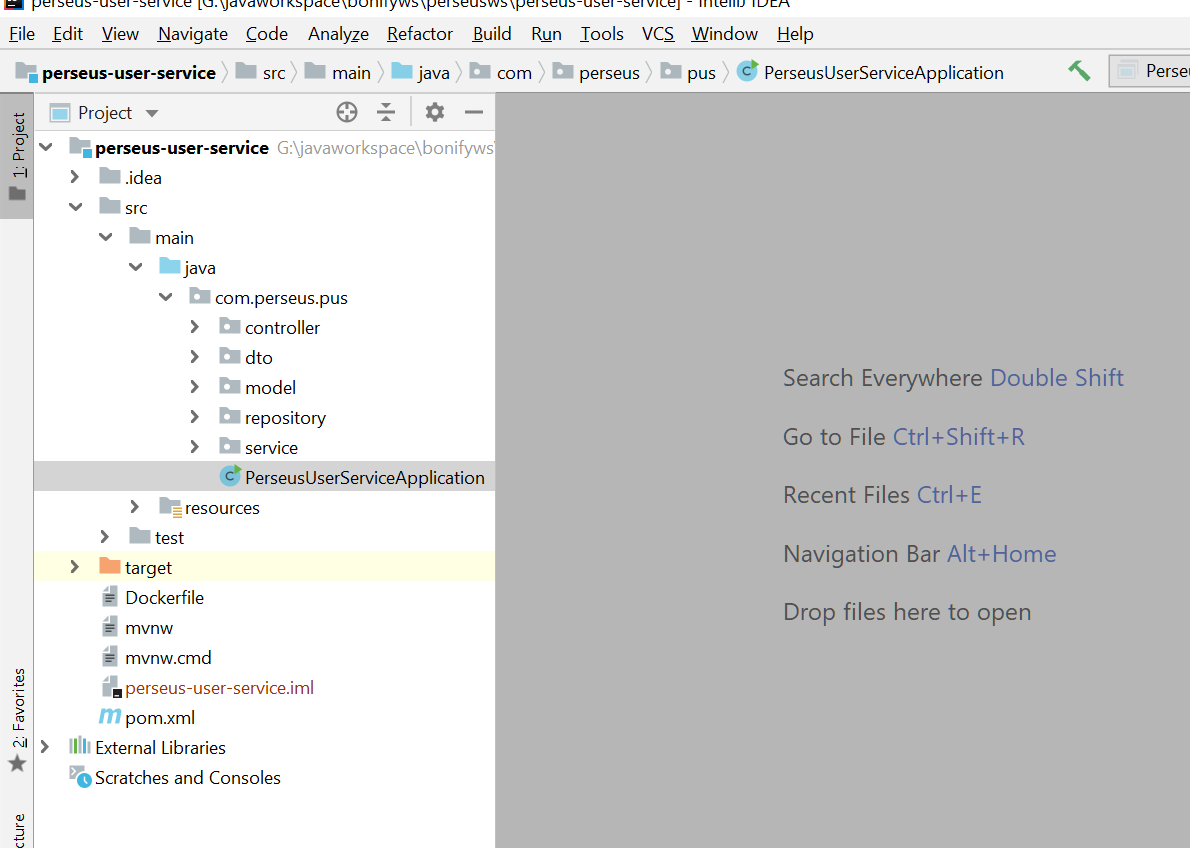
**mvn spring-boot:run**

Application will be started on 8080 port.

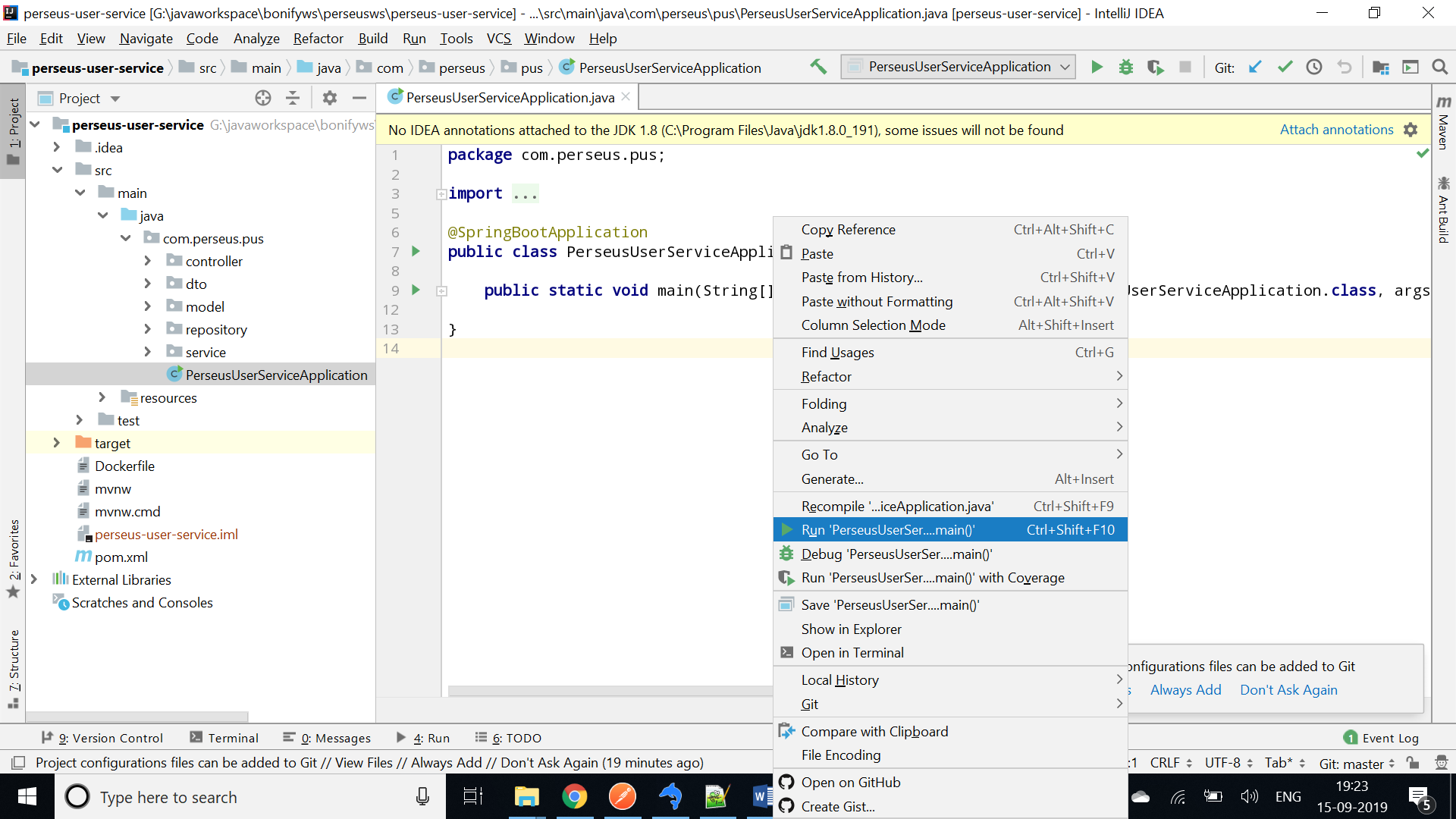
**Step 5:**

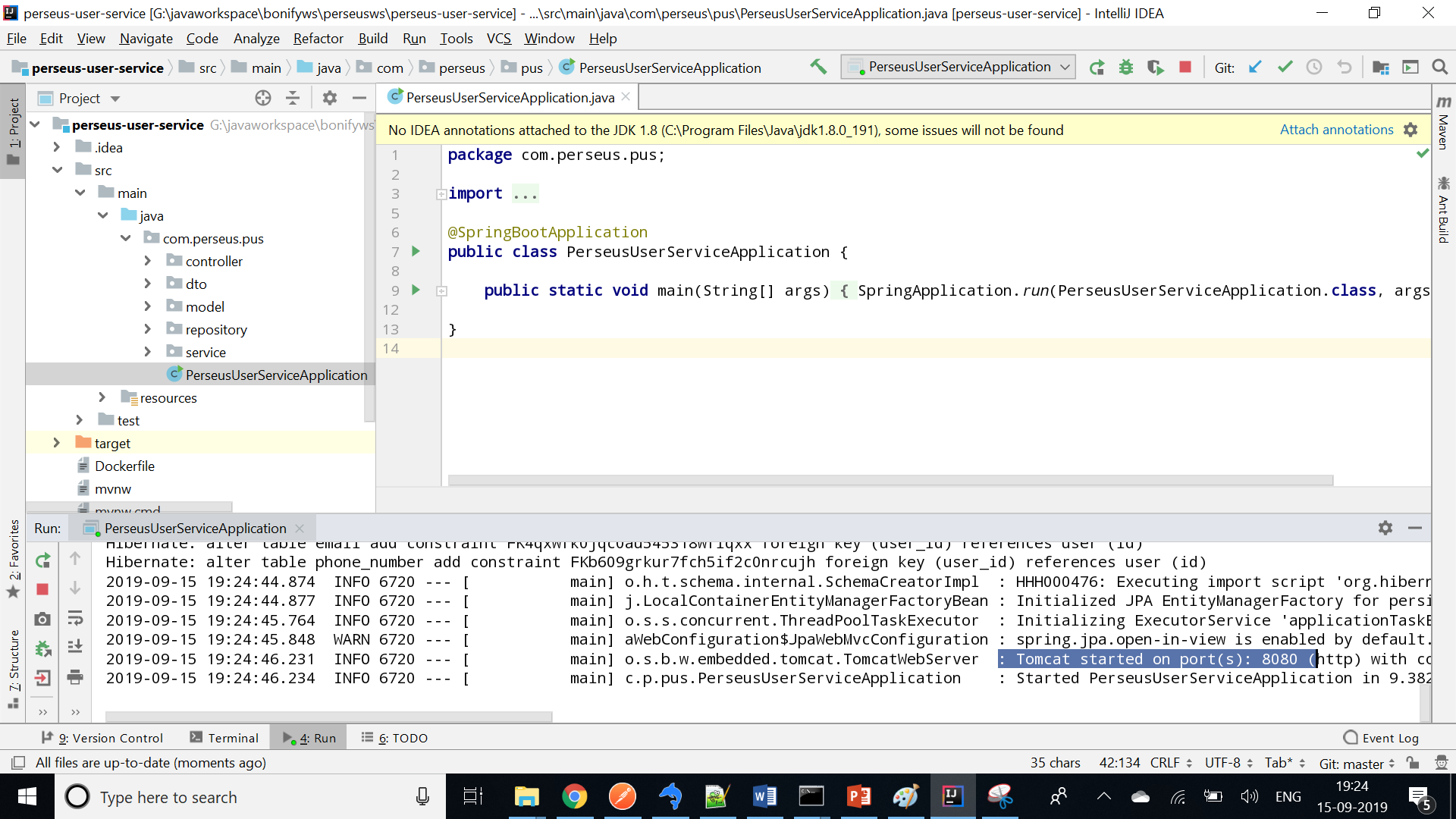
Now we are ready to test our application.

**If we want to run the application from IDE, import the application into IntelliJ IDEA.**



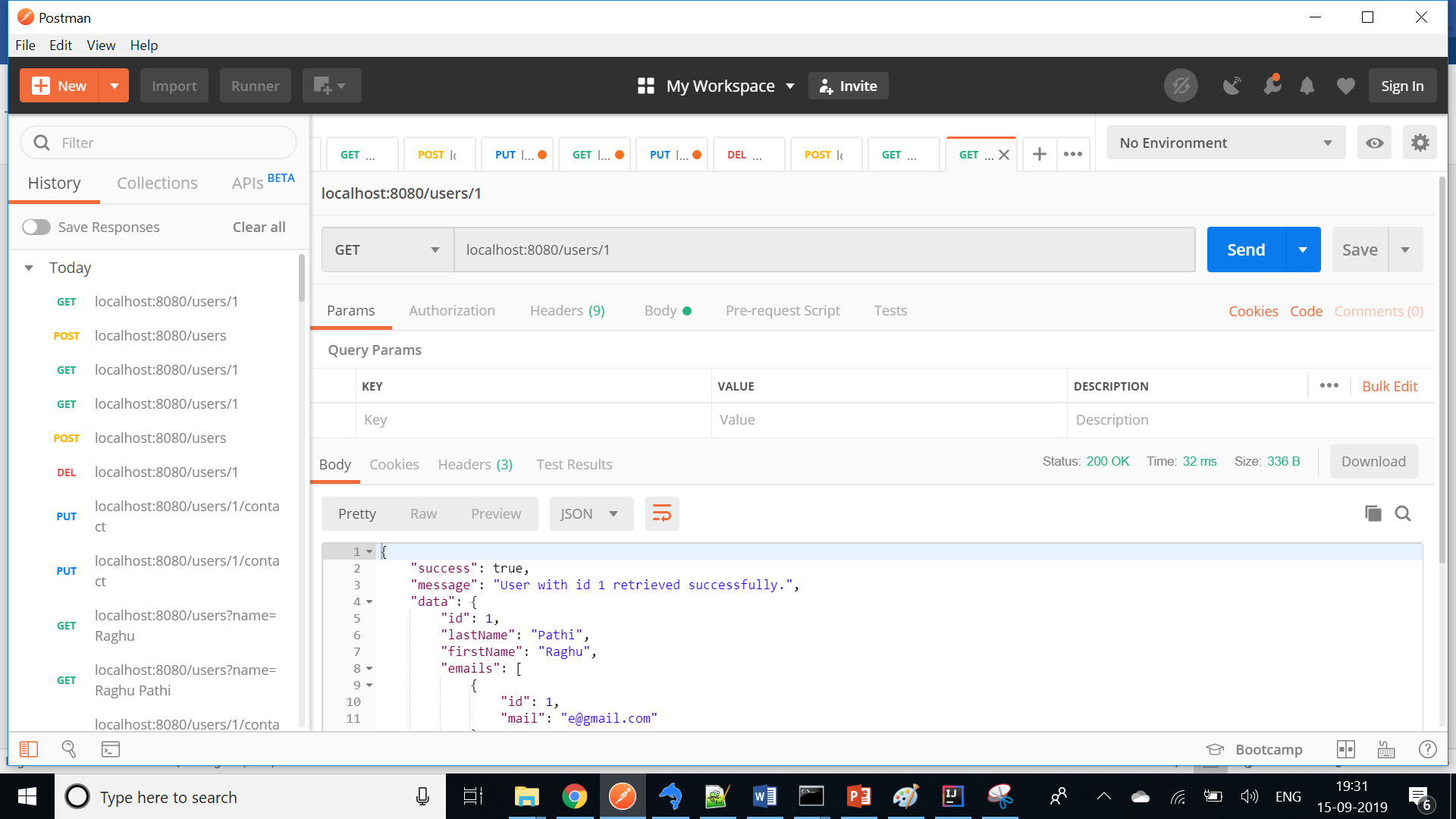
**Run the PerseusUserServiceApplication.java by right clicking in the file.**





## **Testing the application:**

I have tested the application using PostMan. Below is the sample test, all end points explained above.

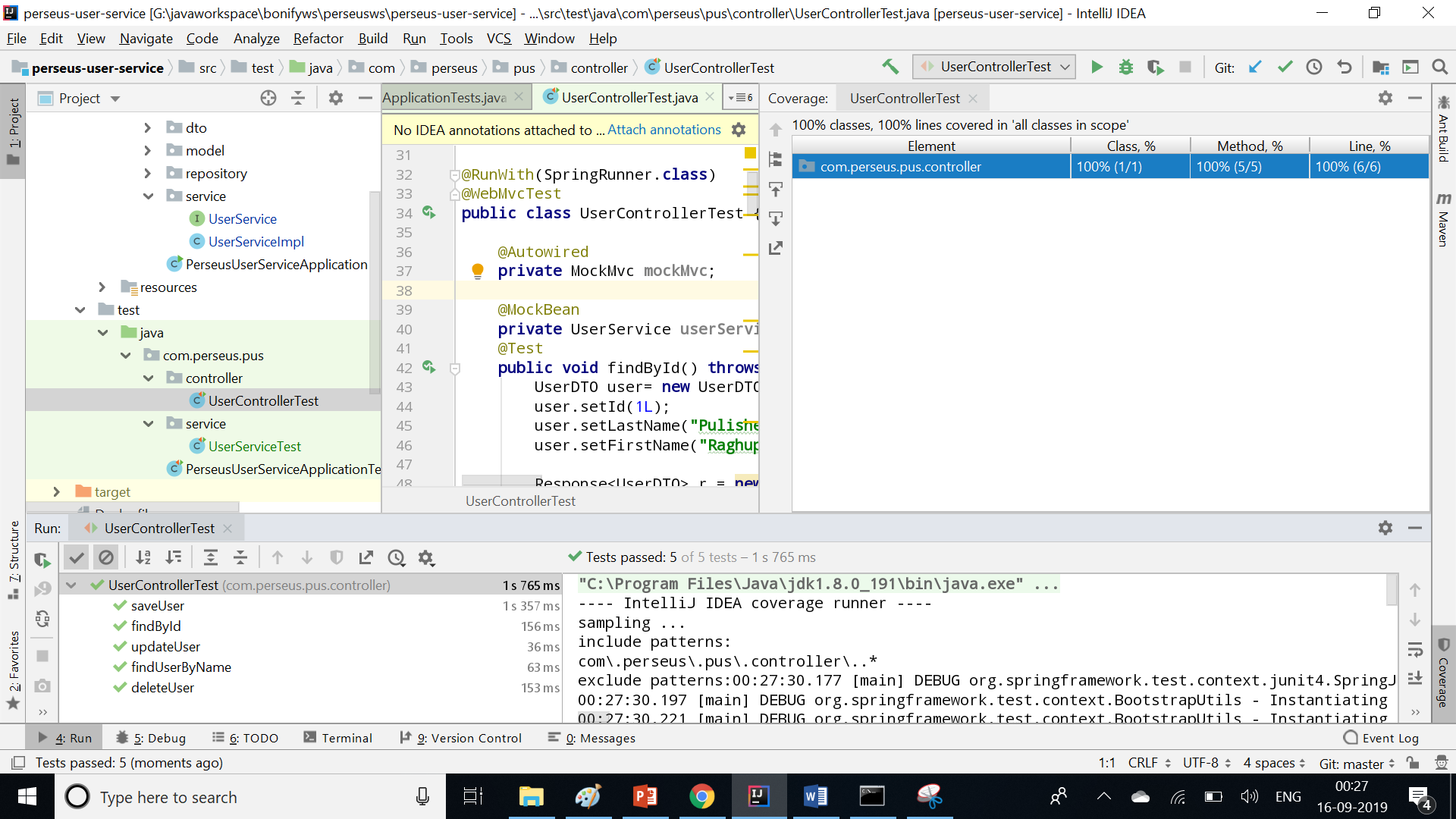


## **Running Test cases from IDE:**

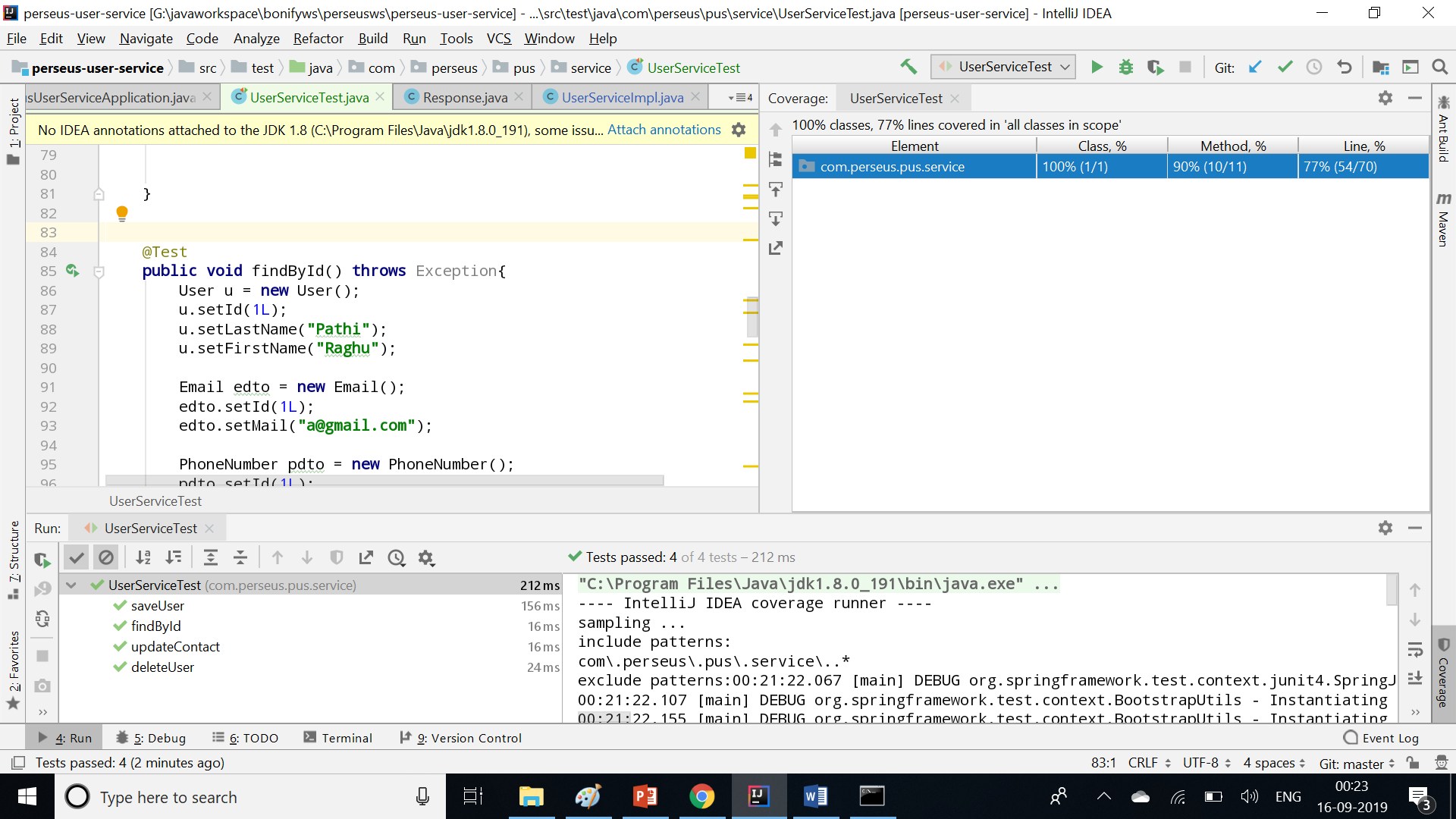
Run the test cases by selecting Run by right clicking in the file.

**UserControllerTest.java**





**UserServiceImpl.java**



## **Running Test cases from Command Prompt:**

Run the junits by running mvn clean install command from project home directory. It will execute 10 test cases and those must be passed.

