SPRING BOOT RESTFUL WEB SERVICE

Lab Guides

Document Code	**25e-BM/HR/HDCV/FSOFT**
Version	**1.0**
Effective Date	**01/06/2020**

Ho Chi Minh, 06/2020

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1	01/06/2020	Add new	New	KyLH	

Contents

CODE:	GSTL.M.RESTFul
TYPE:	Medium
LOC:	NA
DURATION:	2 hours

SECTION 12: Spring boot RESTful web service

Objectives:

In this spring boot RESTful web service tutorial, learn to create REST APIs using Spring boot framework which return JSON responses to client. In this Spring Boot RESTful web service tutorial, we will create ***e-commerce*** APIs step by step and test them.

Specifications/Problem Descriptions:

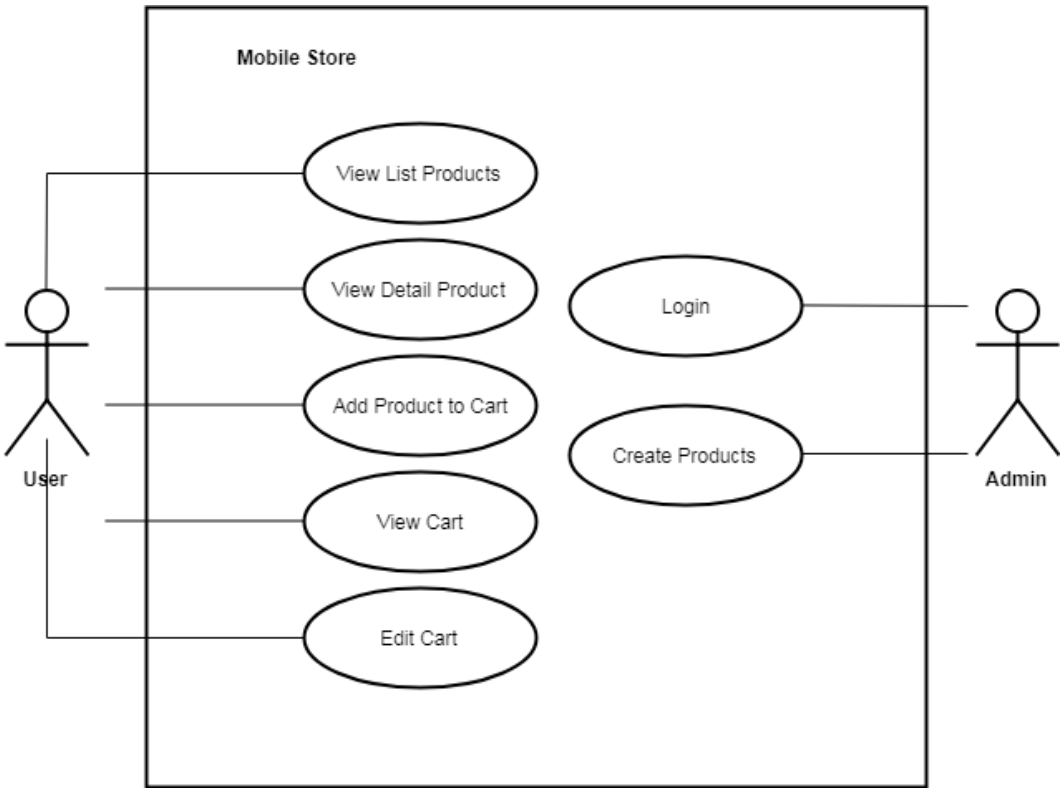
1. **Scope of Project**

   This software product will be the website for buyer and the e-commerce manager.

2. **Glossary**

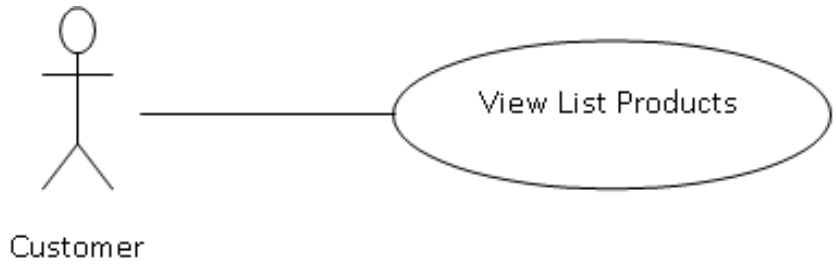
Term	Definition
Administrator	Users could login to update data
Customer	Users want to find and buy items

3. **Funtional Requirements Specification**



a. Customer Use Case

Diagram:



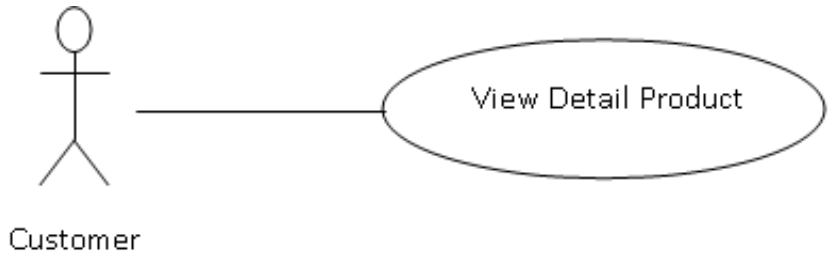
Brief Description
The customer accesses the e-commerce to view list products.

Initial Step-By-Step Description
Before this use case can be initiated, the user has already accessed the e-commerce website with role customer.

1. The customer accesses the home page.
2. The system redirects to list all products.
3. The system allows the user to perform other activities.

b. Customer View Detail Product

Diagram:



Brief Description

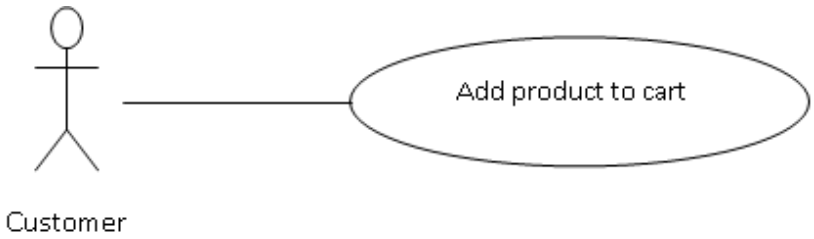
The customer accesses the e-commerce to view detail product.

Initial Step-By-Step Description
Before this use case can be initiated, the user has already accessed the e-commerce website with role customer.

1. The customer accesses the home page.
2. The system redirects to list all products.
3. The customer selects product.
4. The system redirects to product detail page.
5. The system allows the user to perform other activities

c. Customer Add product to cart

Diagram:



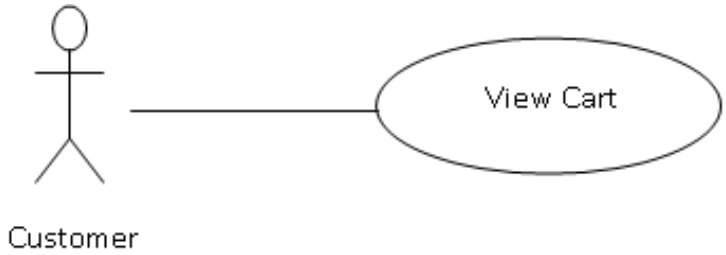
Brief Description
The customer accesses the e-commerce to add product to cart.

Initial Step-By-Step Description
Before this use case can be initiated, the user has already accessed the e-commerce website with role customer.

1. The customer accesses the Mobile Store.
2. The customer clicks button order now on each product (list all products page or detail product page).
3. The system adds selected product to cart.
4. The system allows the user to perform other activities.

d. Customer View cart:

Diagram:

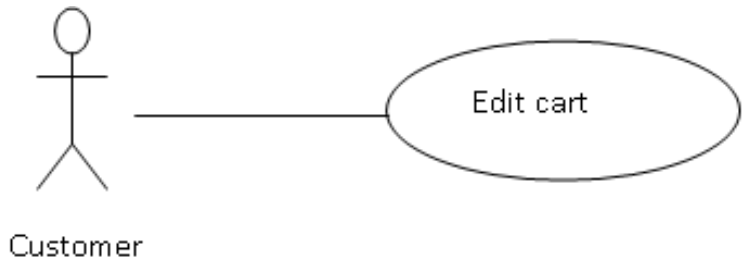


Brief Description
The customer accesses the e-commerce to transfer money to view cart

Initial Step-By-Step Description
Before this use case can be initiated, the user has already accessed the e-commerce website with role customer.

1. The customer accesses the home page.
2. The system redirects to list all products.
3. The customer clicks view cart on top right.
4. The system redirects to cart page.
5. The system allows the user to perform other activities.

e. Customer edit cart:
Diagram:

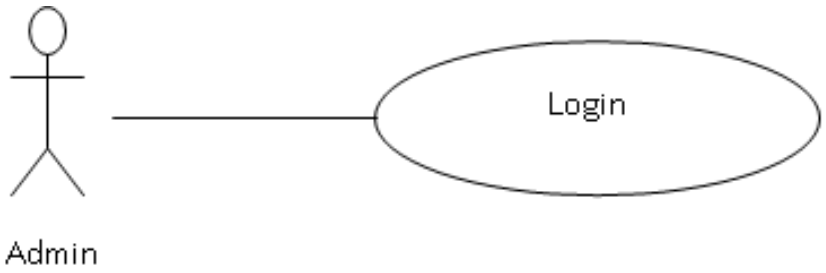


Brief Description
The customer accesses the e-commerce to edit cart.

Initial Step-By-Step Description
Before this use case can be initiated, the user has already accessed the e-commerce website with role customer.

1. The customer accesses the home page.
2. The system redirects to list all products.
3. The customer clicks view cart on top right.
4. The system redirects to cart page.
5. The system shows all selected products.
6. The customer can removes each cart item or clears all cart.
7. The system allows the user to perform other activities.

f. Admin login
Diagram:

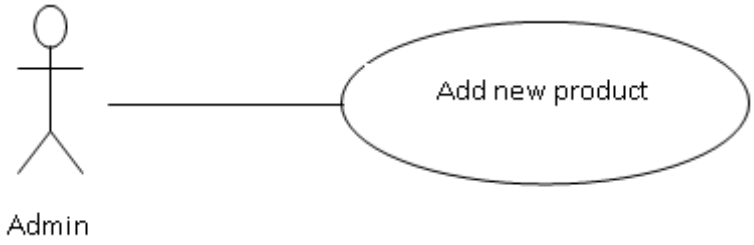


Brief Description
The admin accesses the e-commerce to login.

Initial Step-By-Step Description
Before this use case can be initiated, the user has already accessed the login page on e-commerce website.
1. The admin accesses the login page.
2. The admin fills username/password.
3. The system validates input and response error message if error happened.
4. The system redirect to add product page if correct username/password.
5. The system allows the user to perform other activities.

g. Admin add new product

Diagram:



Brief Description
The admin accesses the e-commerce to add new product.

Initial Step-By-Step Description
Before this use case can be initiated, the user has already accessed the e-commerce website with role admin.

1. The admin accesses to add product page.
2. The admin fills new product information and click add product.
3. The system insert new product to database and clear form.
4. The system allows the admin to perform other activities.

4. Database Relationship

Based on the analysis of the requirements, we decide to use the following database tables to store the persistent data for our blog application:

- **Users** table stores user information, including id, name, email, password, created date
- **ProductImage** table stores product image information, including image id, product id, path
  - ✓ Product id reference to product id of Product table
- **Product** table stores product information, including product id, name, created date, price, description, product group id, user id
  - ✓ Group id reference to group id of Group table
  - ✓ User id reference to user id of User table (who is the person create this product)
- **ProductGroup** table stores group of product information, including id, group name, price, created date
- **GroupVariant** table stores group variant information, including id, variant name, product group id
  - ✓ Product group id reference to product group id of ProductGroup table
- **OrderItem** table stores order item information, including order item id, price, order id, product id, group variant id
  - ✓ order id reference to order id of Order table
  - ✓ product id reference to product id of Product table
  - ✓ Group variant id reference to group variant id of GroupVariant table
- **Order** table stores order information, including order id, name, address, city, zip, status, comment, total price, type

Assumptions:

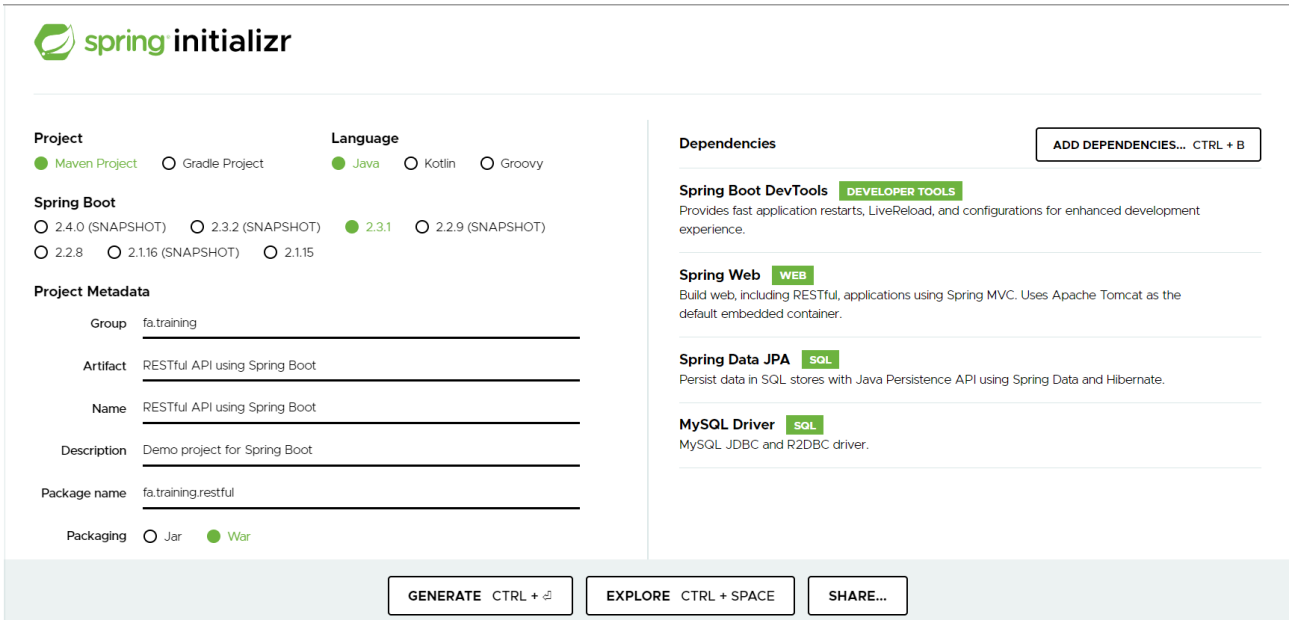
Technical Requirements:

- Eclipse
- JDK 1.8
- Maven 4.0
- MySQL 8

Guidelines

Step 1: Create the Spring Boot Project

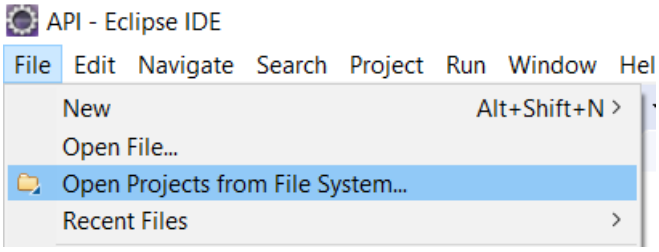
- First, go to Spring Initializr (https://start.spring.io/) and create a project with below settings



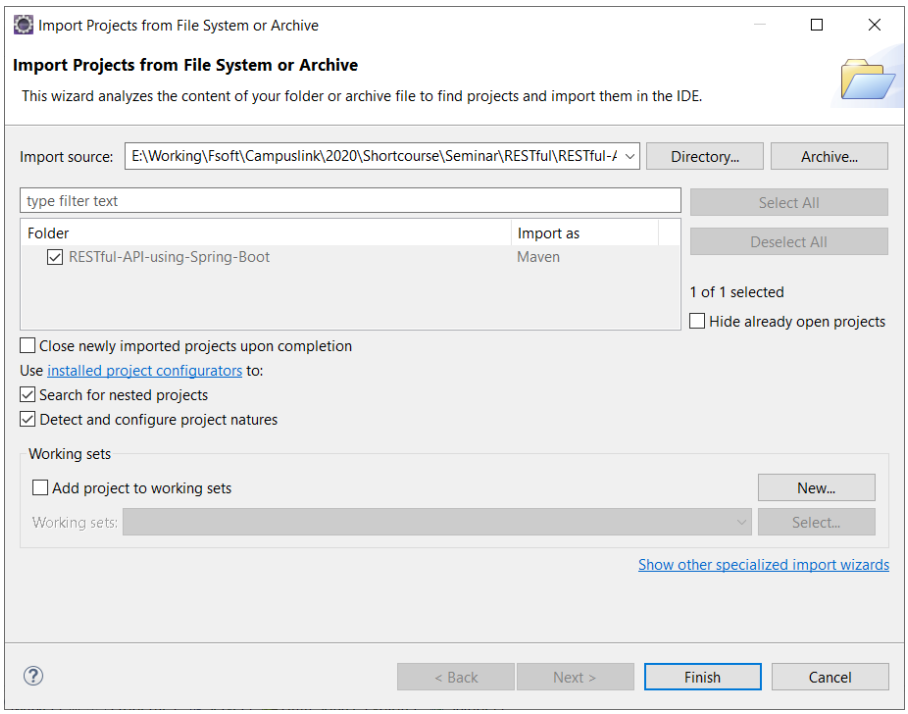
- Web — Full-stack web development with Tomcat and Spring MVC
- DevTools — Spring Boot Development Tools
- JPA — Java Persistence API including spring-data-JPA, spring-orm, and Hibernate
- MySQL — MySQL JDBC driver
- Generate, download, and import to development IDE.

Step 2: Import project into eclipse

- Open eclipse -> Select “File” -> Select “Open projects from File System”

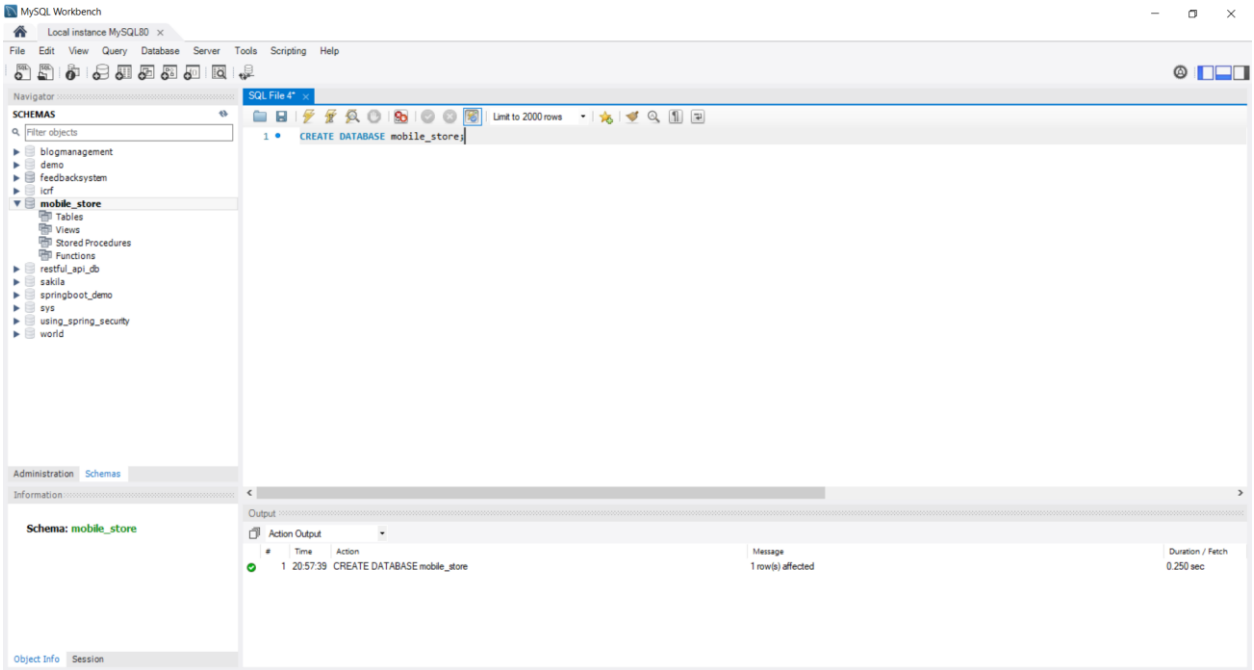


- Select “Directory” -> Select “RESTful-API-using-Spring-Boot” project -> Select “Finish”



Step 3: Define Database Configurations

- Next Create the database name called **mobile_store** in MySQL database server

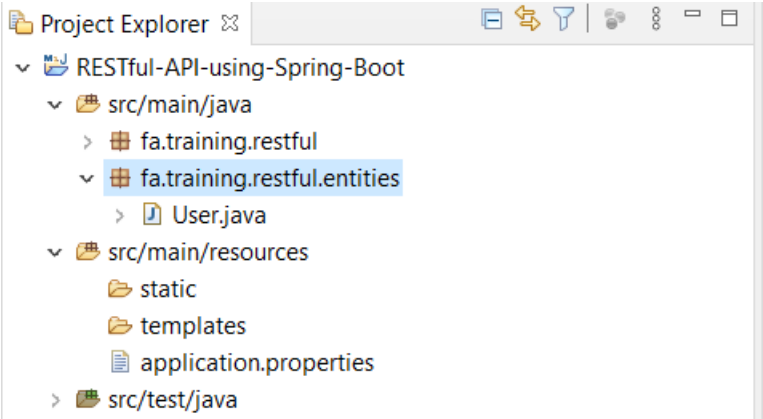


- Define connection properties in **src/main/resources/application.properties**

```
## Database Properties
spring.datasource.url = jdbc:mysql://localhost:3306/mobile_store?useSSL=false
spring.datasource.username = root
spring.datasource.password = root

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

Step 4: Create entities package

- Create All of entities class to mapping with database



Step 5: Create User class and insert code

```
1.  package fa.training.restful.entities;
2.
3.  import javax.persistence.Entity;
4.  import javax.persistence.GeneratedValue;
5.  import javax.persistence.GenerationType;
6.  import javax.persistence.Id;
7.  import javax.persistence.Table;
8.
9.  @Entity
10. @Table(name="users")
11. public class User {
12.
13.     @Id
14.     @GeneratedValue(strategy = GenerationType.AUTO)
15.     private long id;
16.
17.     private String username;
18.
19.     private String password;
20.
21.     public long getId() {
22.             return id;
23.     }
24.
25.     public void setId(long id) {
26.             this.id = id;
27.     }
28.
29.     public String getUsername() {
30.             return username;
31.     }
32.
33.     public void setUsername(String username) {
34.             this.username = username;
35.     }
36.
37.     public String getPassword() {
38.             return password;
39.     }
40.
41.     public void setPassword(String password) {
42.             this.password = password;
43.     }
44.
45. }
```

Step 6: Create JPA Data Repository Layer

```
1.  package fa.training.restful.repositories;
2.
3.  import org.springframework.data.jpa.repository.JpaRepository;
4.  import org.springframework.stereotype.Repository;
5.
6.  import fa.training.restful.entities.User;
7.
8.  @Repository
9.  public interface UserRepository extends JpaRepository<User, Long>{
10.
11.     /**
12.      * Sign in
13.      * @param username
14.      * @param password
15.      * @return User
16.      */
17.     public User findByUsernameAndPassword(String username, String password);
18.
19. }
```

Step 7: Create custom exception

```
1.  package fa.training.restful.exceptions;
2.
3.  public class ResourceNotFoundException extends Exception {
4.
5.      public ResourceNotFoundException(String message) {
6.              super(message);
7.      }
8.  }
```

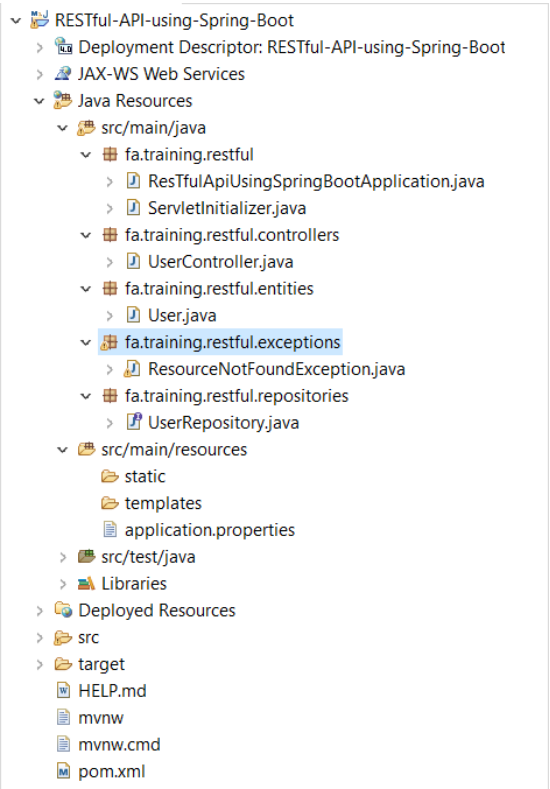
Step 8: Create Rest Controllers and Map API Requests

```
1.  package fa.training.restful.controllers;
2.
3.  import java.util.HashMap;
4.  import java.util.List;
5.  import java.util.Map;
6.
7.  import org.springframework.beans.factory.annotation.Autowired;
8.  import org.springframework.http.ResponseEntity;
9.  import org.springframework.validation.annotation.Validated;
10. import org.springframework.web.bind.annotation.DeleteMapping;
11. import org.springframework.web.bind.annotation.GetMapping;
12. import org.springframework.web.bind.annotation.PathVariable;
13. import org.springframework.web.bind.annotation.PostMapping;
14. import org.springframework.web.bind.annotation.PutMapping;
15. import org.springframework.web.bind.annotation.RequestBody;
16. import org.springframework.web.bind.annotation.RequestMapping;
17. import org.springframework.web.bind.annotation.RestController;
18.
19. import fa.training.restful.entities.User;
20. import fa.training.restful.exceptions.ResourceNotFoundException;
21. import fa.training.restful.repositories.UserRepository;
22.
23. @RestController
24. @RequestMapping("/api/v1/user")
25. public class UserController {
26.
27.     @Autowired
28.     private UserRepository userRepository;
29.
30.     /**
31.      * Get all users list.
32.      *
33.      * @return the list
34.      */
```

```
35.     @GetMapping("/list")
36.     public List<User> getAllUsers() {
37.             return userRepository.findAll();
38.     }
39.
40.     /**
41.      * Gets users by id.
42.      *
43.      * @param userId the user id
44.      * @return the users by id
45.      * @throws ResourceNotFoundException the resource not found exception
46.      */
47.     @GetMapping("/get/{id}")
48.     public ResponseEntity<User> getUserById(@PathVariable(value = "id") Long userId) throws
    ResourceNotFoundException {
49.
50.             User user = userRepository.findById(userId)
51.                             .orElseThrow(() -> new ResourceNotFoundException("User not found on:
    " + userId));
52.
53.             return ResponseEntity.ok().body(user);
54.     }
55.
56.     /**
57.      * Create user user.
58.      *
59.      * @param user the user
60.      * @return the user
61.      */
62.     @PostMapping("/add")
63.     public User create(@Validated @RequestBody User user) {
64.             return userRepository.save(user);
65.     }
66.
67.     /**
68.      * Update user response entity.
69.      *
70.      * @param userId      the user id
71.      * @param userDetails the user details
72.      * @return the response entity
73.      * @throws ResourceNotFoundException the resource not found exception
74.      */
75.     @PutMapping("/update/{id}")
76.     public ResponseEntity<User> update(@PathVariable(value = "id") Long userId,
77.                     @Validated @RequestBody User userDetails) throws ResourceNotFoundException
    {
78.
79.             User user = userRepository.findById(userId)
80.                             .orElseThrow(() -> new ResourceNotFoundException("User not found on:
    " + userId));
81.
82.             user.setPassword(userDetails.getPassword());
83.             final User updatedUser = userRepository.save(user);
84.
85.             return ResponseEntity.ok(updatedUser);
86.     }
87.
88.     /**
89.      * Delete user map.
90.      *
91.      * @param userId the user id
92.      * @return the map
93.      * @throws Exception the exception
94.      */
95.     @DeleteMapping("/delete/{id}")
```

```
    public Map<String, Boolean> delete(@PathVariable(value = "id") Long userId) throws Exception {

            User user = userRepository.findById(userId)
                            .orElseThrow(() -> new ResourceNotFoundException("User not found on: " + userId));

                    userRepository.delete(user);

                    Map<String, Boolean> response = new HashMap<>();
                    response.put("deleted", Boolean.TRUE);

                    return response;
            }

            /**
             * Sign in
             *
             * @param username
             * @param password
             * @return User
             */
            @PostMapping("/signin")
            public ResponseEntity<User> signIn(@Validated @RequestBody User u) {
                    User user = userRepository.findByUsernameAndPassword(u.getUsername(), u.getPassword());

                    if (user == null) {
                            return ResponseEntity.ok(null);
                    }

                    return ResponseEntity.ok(user);
            }
    }
```

Once completed, you will see the project structure as below:

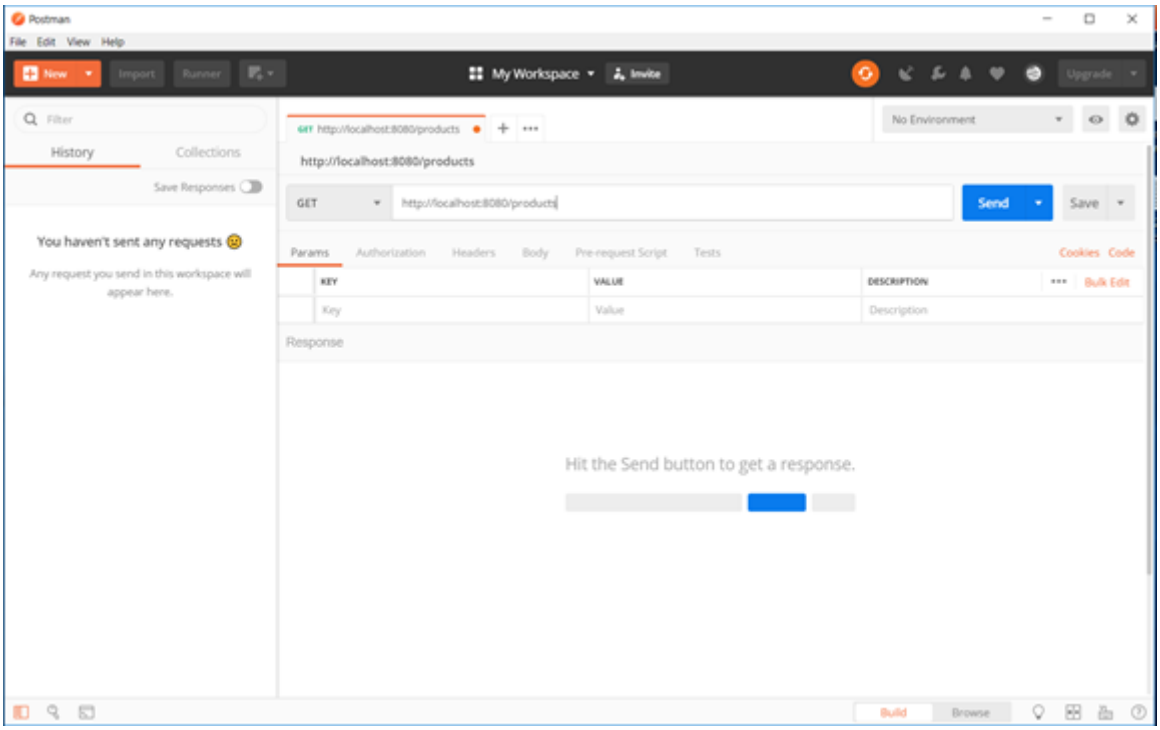


Testing

Step 1: Download and install Postman

- From link: https://www.postman.com/downloads/

Step 2: Open Postman to test RESTful API

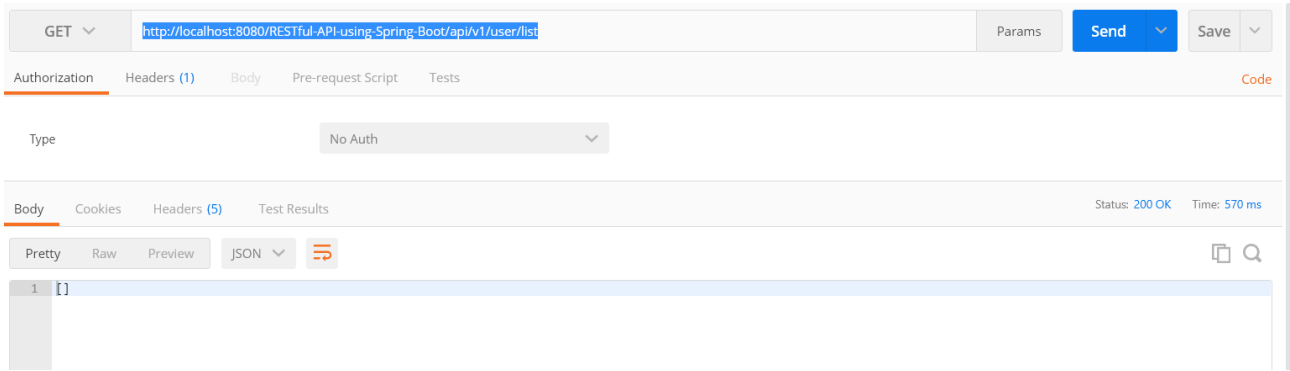


Step 3: Run this spring boot project

- After you run this project, database will be generate
  - Right click on project -> Run As -> Run on Server
- You should create at least one user account in this application

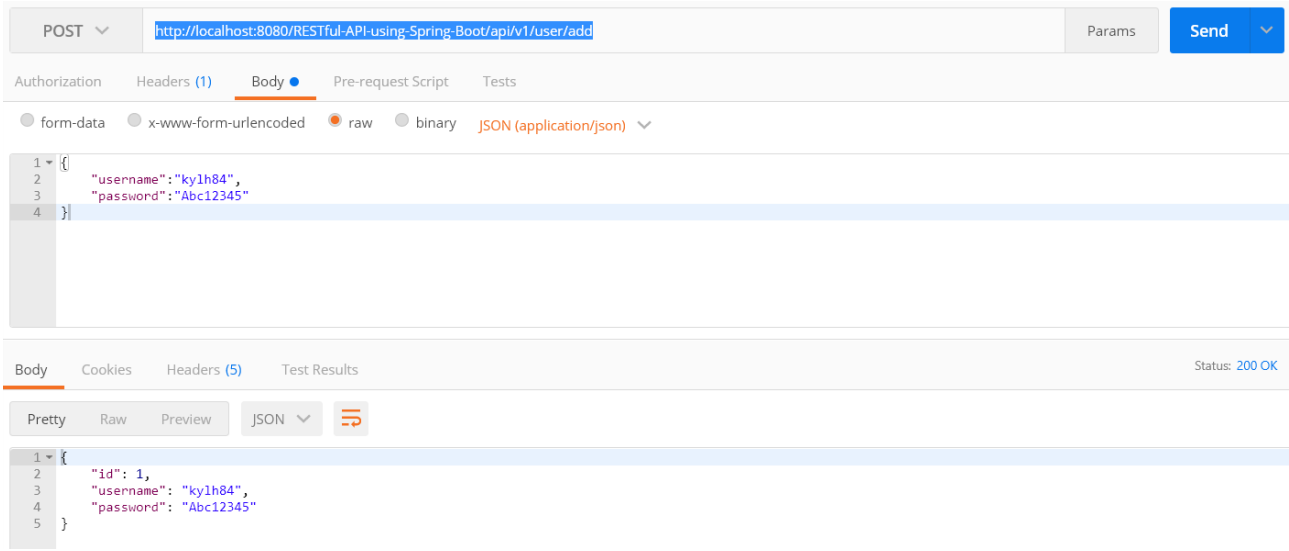
Step 4: Try to get all of users

- With this link: http://localhost:8080/RESTful-API-using-Spring-Boot/api/v1/user/list
- Method: GET



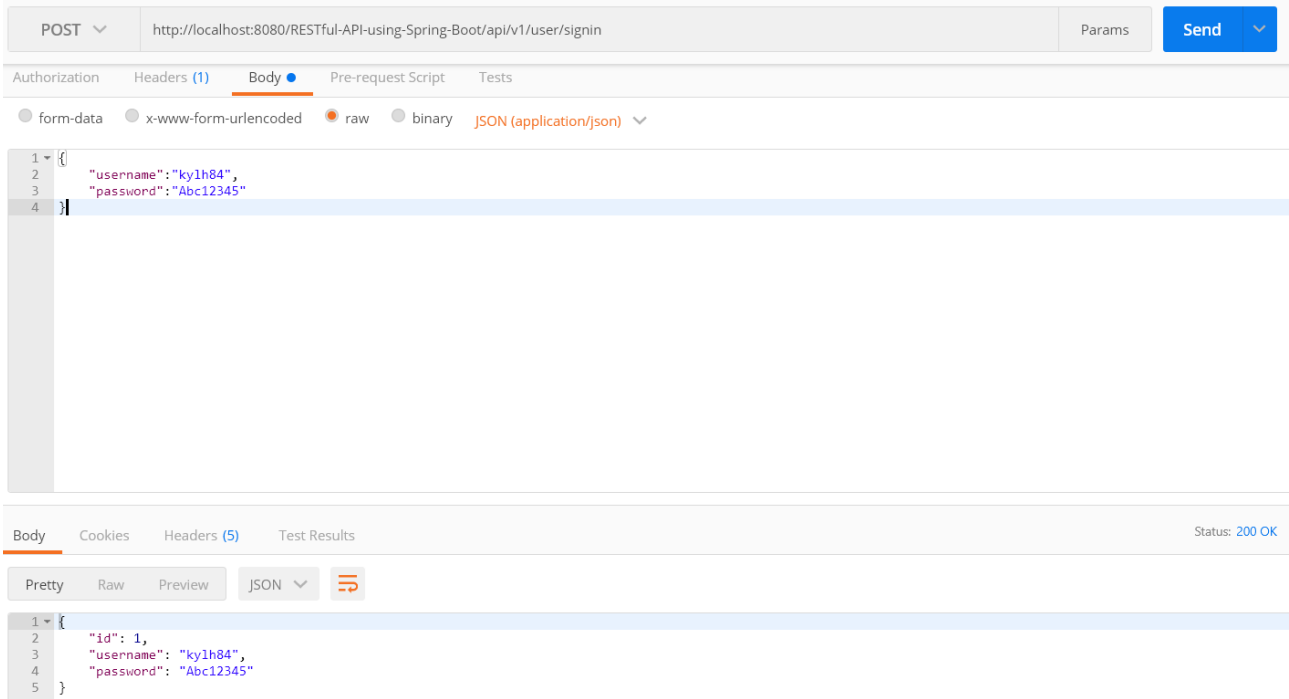
Step 5: Try to create an user

- With this link: http://localhost:8080/RESTful-API-using-Spring-Boot/api/v1/user/add
- Method: POST



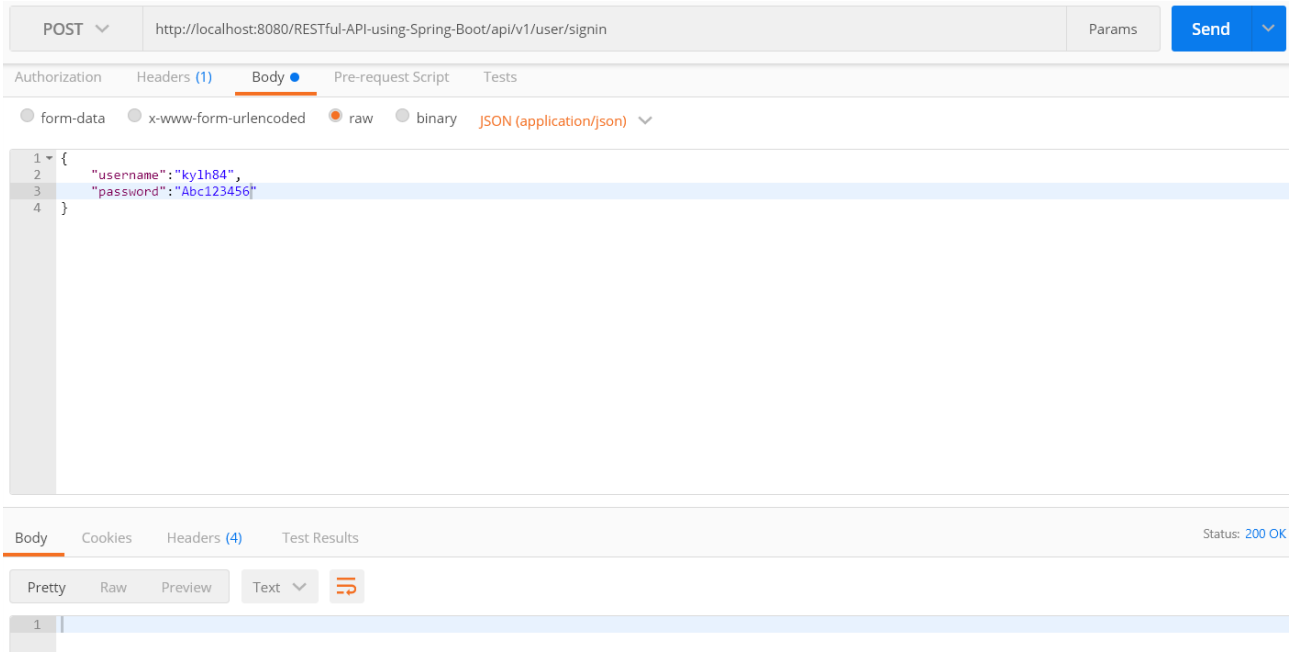
Step 6: Try to login successful

- With this link: http://localhost:8080/RESTful-API-using-Spring-Boot/api/v1/user/signin
- Method: POST



Step 7: Try to login unsuccessful

- With this link: http://localhost:8080/RESTful-API-using-Spring-Boot/api/v1/user/signin
- Method: POST



Step 8: Try to change password

- With this link: http://localhost:8080/RESTful-API-using-Spring-Boot/api/v1/user/update/1
- Method: PUT