sportyshoes.com

Sports Shoes Portal Online

Prototype of the Application

Name: Pravesh Kumar Patel

GitHub: https://github.com/Pravesh105/E-commerce-Website-for-Sporty-Shoes

The prototype of the application starts from the backend, and it can also directly start from the project folder. This portal allows us to do shoes management across administrator and provide CRUD methodologies across admin side. This prototype is built through various controllers which are auto wired with Repository, models, Interfaces, Exception and Services.

The implementation is done with the help of Hibernate, Maven, Spring Boot, MySql, JDK 8, IntelliJ and for testing API is done through Postman and Swagger-UI for Documentation.

Sprint Planning

The Implementation is done in five sprints which are mentioned below:

Sprint 1:

- Clarify the specification and requirements.
- Implement a model of "Product" which are available in stock with their respective attributes such as category, price, shoe size, company, origin.
- Implement a service and repository of same entity Product to perform CRUD operation on the database.
- Creating a controller through Rest API for testing the functions of all the java classes (repository, service, model, controller).

Sprint 2:

- Similarly, creating the same java classes for products which have been purchased by the customer named "PurchasedProduct".
- And the corresponding same structure is defined for customers who have bought the products using this portal. Java Class files such as Customer.java, ICustomerService.java, CustomerServiceImpl, CustomerRepository.java, CustomerController.
- Implementing a relationship between "Customer" and "PurchasedProduct" using One-To-Many associations.

Sprint 3:

- Implement a model "Admin" for registering and logging entries in the database.
- Implement functionality for changing password in admin section which consist of new password and new email.
- Implementing a functionality in which admin can fetch all the data of the customer and the available product.
- Implementing another function for procuring the records of all users (Discussed in Sprint 4) who have signed up in this portal.

Sprint 4:

- Developing the user package containing the same structure (Repository, model, controller, service).
- The user can select the product without inserting any of the user details for purchasing and being registered in the database.
- Implementing a functionality on user side where user can register, login or change its details in the database.

Sprint 5:

- Implementing a functionality on user and customer side where a person can purchase more than one product without the need of inserting its details and does not change the state of its content in the database, relationship.
- Implementing a functionality where admin can retrieve all the purchased products throught the use of foreign key in custom queries.
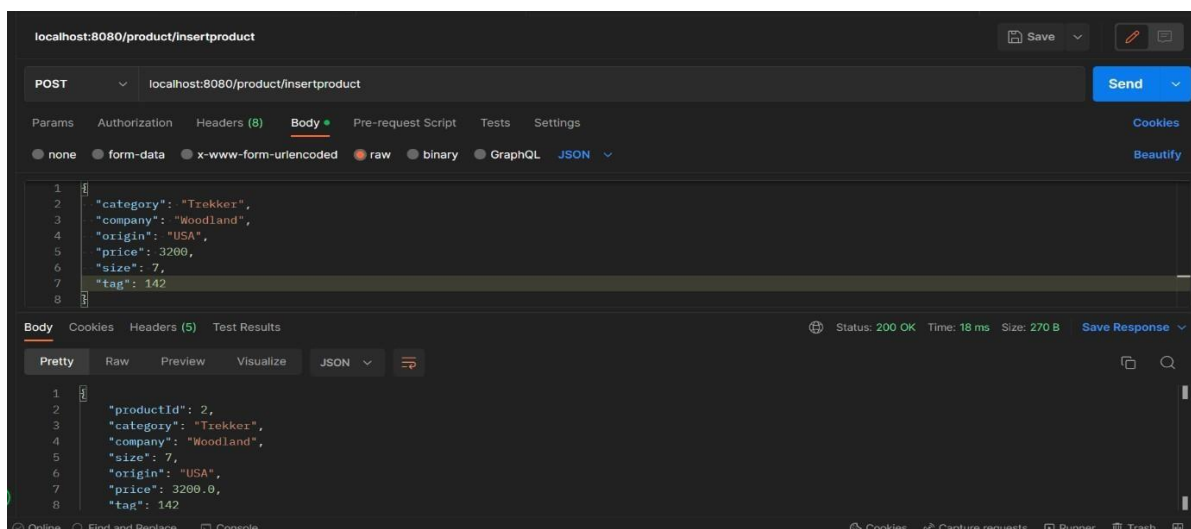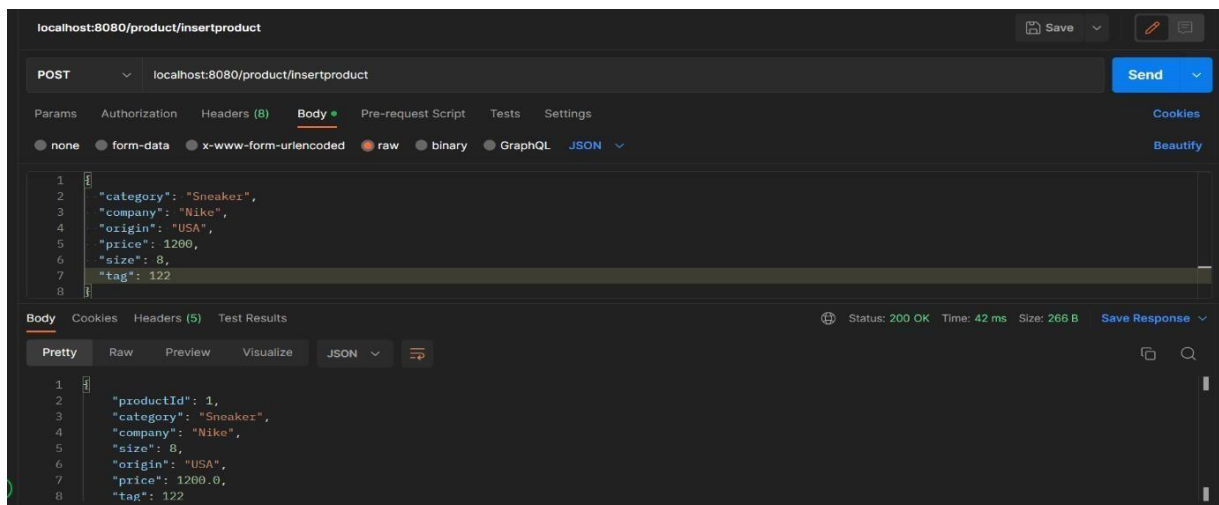- Documentation

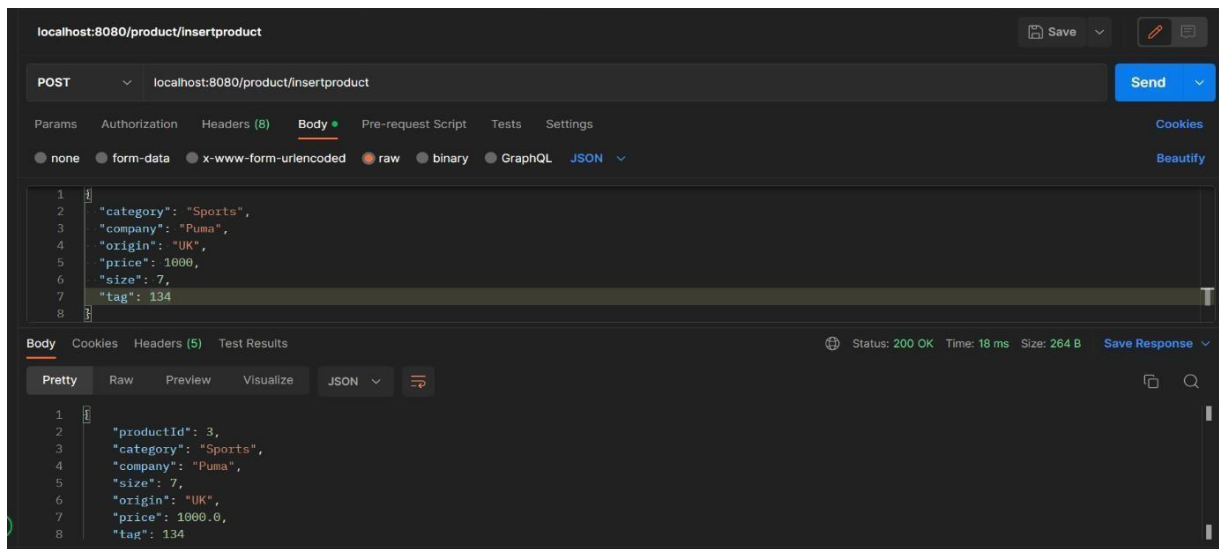## Documentation of the functionality:

Here are some of the screenshots of testing API through various controllers. 1:

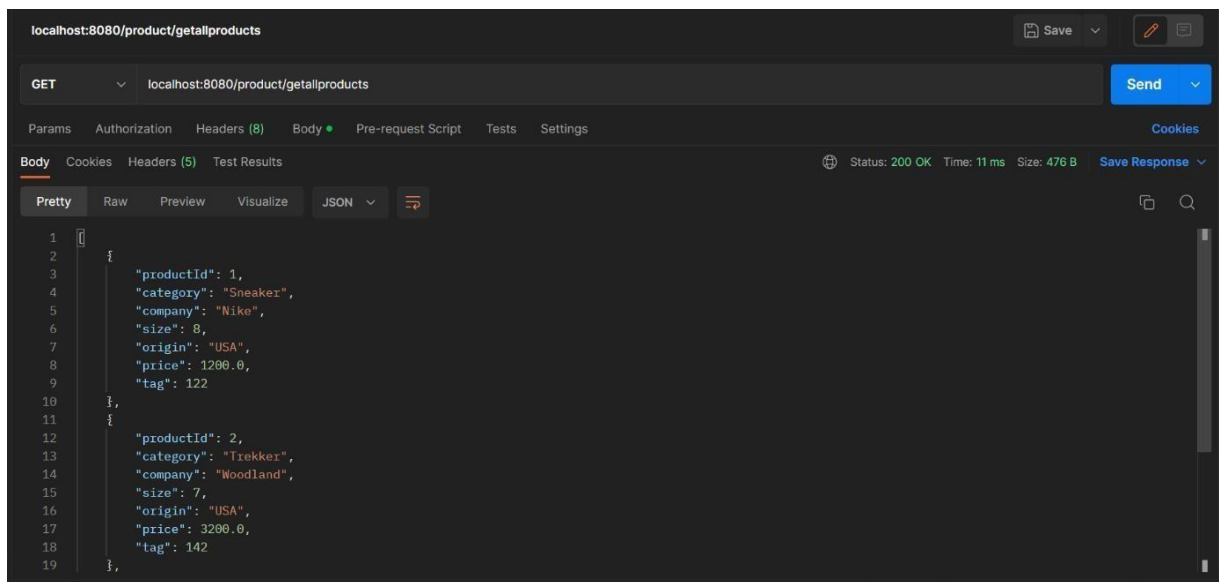Login And Register (Admin Section)
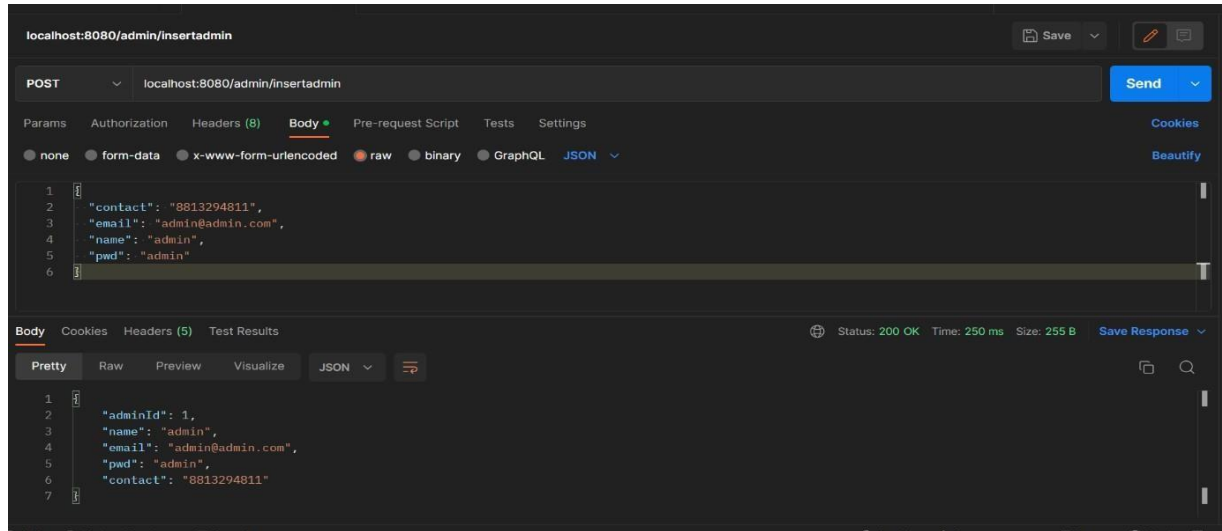
2: Inserting Product (Admin Section)

3: Display All Available Products (Admin Section)

4: Purchasing Product through productid (Customer Section)

```
http://localhost:8080/customer/insertcustomer/1
```

Server response

Code   Details

200
        Response body

        {
          "customerId": 1,
          "customerName": "John",
          "email": "john@gmail.com",
          "contact": 2391391312,
          "date": "20-01-2023",
          "totalPrice": 1200,
          "list": [
            {
              "productId": 1,
              "category": "Sneaker",
              "company": "Nike",
              "size": 8,
              "origin": "USA",
              "price": 1200,
              "tag": 122
            }
          ]
        }
                                                                    Download

        Response headers

        connection: keep-alive
        content-type: application/json
        date: Sun29 Jan 2023 12:31:08 GMT
        keep-alive: timeout=60
        transfer-encoding: chunked

Responses

5: Adding more Products through customerId and Product Id (Customer Section)

custid * required
integer($int64)   custid
(path)

        1

prodid * required
integer($int64)   prodid
(path)

        2

user * required
object   user
(body)
                      Edit Value | Model

        {
          "contact": 0,
          "customerId": 0,
          "customerName": "string",
          "date": "20-01-2023",
          "email": "string",
          "list": [
            {
              "category": "string",
              "company": "string",
              "origin": "string",
              "price": 0,
              "productId": 0,
              "size": 0,
              "tag": 0
            }
          ],
          "totalPrice": 0
        }

200
        Response body

        [
          {
            "customerId": 2,
            "customerName": "John",
            "email": "john@gmail.com",
            "contact": 2391391312,
            "date": "20-01-2023",
            "totalPrice": 4400,
            "list": [
              {
                "productId": 3,
                "category": "Trekker",
                "company": "Woodland",
                "size": 7,
                "origin": "USA",
                "price": 3200,
                "tag": 142
              },
              {
                "productId": 2,
                "category": "Sneaker",
                "company": "Nike",
                "size": 8,
                "origin": "USA",
                "price": 1200,
                "tag": 122
              }
            ]
          }
        ]
                                                                    Download

        Response headers

        connection: keep-alive
        content-type: application/json
        date: Sun29 Jan 2023 12:32:42 GMT
        keep-alive: timeout=60
        transfer-encoding: chunked

Responses

6: Register And Login (User Section)

7: Inserting Product through userId and productId (User Section)

8: Adding more products in same userId/customerId (User Section)

9: Other Functions in Admin Section

admin-controller Admin Controller

GET	/admin/getallcustomerbydate/{date}	getAllCustomersByDateAndCategory
GET	/admin/getallcustomerbyname/{custname}	findByCustomerName
GET	/admin/getallproductsbycategory/{category}	getAllProductssByCategory
GET	/admin/getallusers	getAllUsers1
GET	/admin/getallvalidcustomers	getAllUsers
POST	/admin/insertadmin	insertAdmin
PUT	/admin/updateadminbyid/{adminid}	updateAdmin

product-controller Product Controller

DELETE	/product/deleteproduct/{productid}	deleteProduct
GET	/product/getallproducts	getAllProducts
GET	/product/getproduct/{productid}	getProduct
POST	/product/insertproduct	insertProduct
PUT	/product/updateproductbyid/{productid}	updateProduct

10: Other Functions in Customer Section

customer-controller Customer Controller

PUT	/customer/addcustomerproducts/{custid}/{prodid}	updateProductInCustomer
DELETE	/customer/deletecustomer/{userid}	deleteCustomer
GET	/customer/getallcustomers	getAllCustomers
GET	/customer/getcustomer/{userid}	getCustomer
POST	/customer/insertcustomer/{id}	insertCustomer
PUT	/customer/updatecustomerbyid/{userid}	updateCustomer

purchased-product-controller Purchased Product Controller

DELETE	/purchased/deletepurchasedproduct/{productid}	deletePurchasedProduct
GET	/purchased/getallpurchasedproducts	getAllPurchasedProducts
GET	/purchased/getpurchasedproduct/{productid}	getProduct
POST	/purchased/insertpurchasedproduct	insertProduct
PUT	/purchased/updatepurchasedproductbyid/{productid}	updatePurchasedProduct

11: Other Functions in User Section

user-controller User Controller

PUT	/user/addmoreuserproducts/{custid}/{prodid}	updateProductInCustomer
GET	/user/getproducts/{userid}	getproducts
POST	/user/insertproductinuser/{userid}/{prodid}	insertProductUserInDB
POST	/user/signup	insertUser
PUT	/user/updateuserbyid/{userid}	updateUser

12: Database (MySql)

```
mysql> select * from product;
+------------+------------------+-----------------+----------------+---------------+--------------+-------------+
| product_id | product_category | product_company | product_origin | product_price | product_size | product_tag |
+------------+------------------+-----------------+----------------+---------------+--------------+-------------+
|          1 | Sneaker          | Nike            | USA            |          1200 |            8 |         122 |
|          2 | Trekker          | Woodland        | USA            |          3200 |            7 |         142 |
|          3 | Sports           | Puma            | UK             |          1000 |            7 |         134 |
+------------+------------------+-----------------+----------------+---------------+--------------+-------------+
3 rows in set (0.01 sec)

mysql>
```

```
mysql> select * from customer;
+-------------+------------------+---------------+----------------+------------------+-------------+
| customer_id | customer_contact | customer_name | purchased_date | customer_email   | total_price |
+-------------+------------------+---------------+----------------+------------------+-------------+
|           2 |       2391391312 | John          | 20-01-2023     | john@gmail.com   |        4400 |
|           4 |       2345910910 | Jack          | 22-01-2023     | jack@gmail.com   |        4400 |
+-------------+------------------+---------------+----------------+------------------+-------------+
2 rows in set (0.00 sec)

mysql>
```

```
mysql> select * from user;
+---------+--------------+----------------+-----------+----------+
| user_id | user_contact | user_email     | user_name | user_pwd |
+---------+--------------+----------------+-----------+----------+
|       1 |   2345910910 | jack@gmail.com | Jack      | 12345    |
+---------+--------------+----------------+-----------+----------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> select * from user;
+---------+--------------+----------------+-----------+----------+
| user_id | user_contact | user_email     | user_name | user_pwd |
+---------+--------------+----------------+-----------+----------+
|       1 |   2345910910 | jack@gmail.com | Jack      | 12345    |
+---------+--------------+----------------+-----------+----------+
1 row in set (0.00 sec)

mysql> select * from admin;
+----------+---------------+----------------+------------+----------------+
| admin_id | admin_contact | admin_email    | admin_name | admin_password |
+----------+---------------+----------------+------------+----------------+
|        1 | 8813294811    | admin@admin.com | admin     | admin          |
+----------+---------------+----------------+------------+----------------+
1 row in set (0.00 sec)

mysql>
```

MySQL 8.0 Command Line Client

```
2 rows in set (0.00 sec)

mysql> select * from user;
+---------+--------------+----------------+-----------+----------+
| user_id | user_contact | user_email     | user_name | user_pwd |
+---------+--------------+----------------+-----------+----------+
|       1 |   2345910910 | jack@gmail.com | Jack      | 12345    |
+---------+--------------+----------------+-----------+----------+
1 row in set (0.00 sec)

mysql> select * from admin;
+----------+---------------+----------------+------------+----------------+
| admin_id | admin_contact | admin_email    | admin_name | admin_password |
+----------+---------------+----------------+------------+----------------+
|        1 | 8813294811    | admin@admin.com | admin     | admin          |
+----------+---------------+----------------+------------+----------------+
1 row in set (0.00 sec)

mysql> select * from purchased_product;
+------------+------------------+-----------------+----------------+---------------+--------------+-------------+-------------+
| product_id | product_category | product_company | product_origin | product_price | product_size | product_tag | customer_id |
+------------+------------------+-----------------+----------------+---------------+--------------+-------------+-------------+
|          2 | Sneaker          | Nike            | USA            |          1200 |            8 |         122 |           2 |
|          3 | Trekker          | Woodland        | USA            |          3200 |            7 |         142 |           2 |
|          5 | Trekker          | Woodland        | USA            |          3200 |            7 |         142 |           4 |
|          6 | Sneaker          | Nike            | USA            |          1200 |            8 |         122 |           4 |
+------------+------------------+-----------------+----------------+---------------+--------------+-------------+-------------+
4 rows in set (0.00 sec)

mysql>
```

Source Code:

Here is some of the source code.

1> Project Structure

- Project
  - main
    - java
      - com.sportyshoes
        - config
        - controllers
          - AdminController
          - CustomerController
          - ProductController
          - PurchasedProductController
          - UserController
        - exceptions
          - ResourceNotFound
        - models
          - Admin
          - Customer
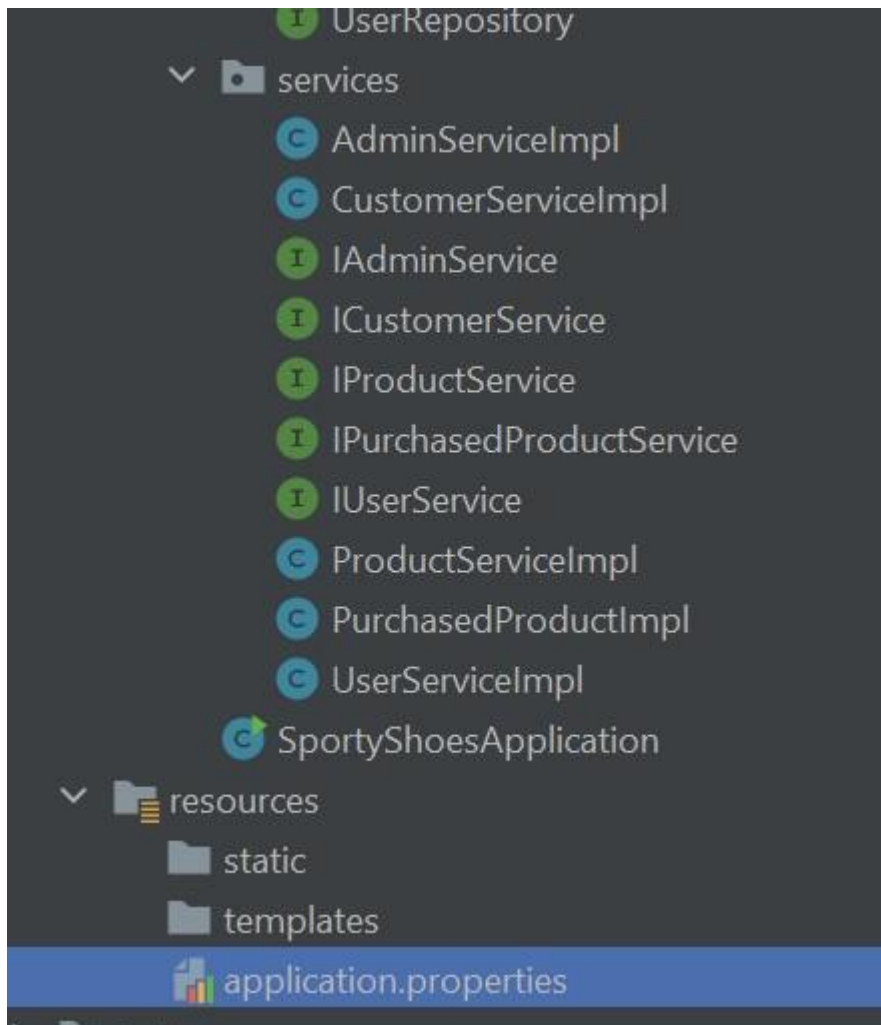   - Product
          - PurchasedProduct
          - User
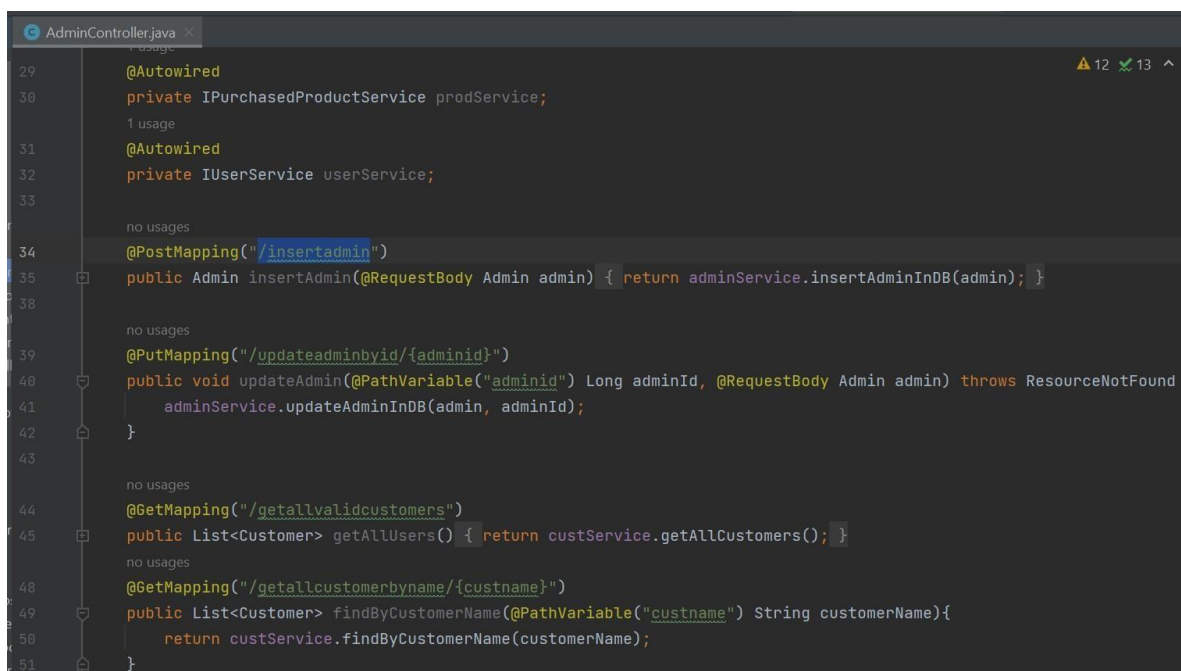   - repository
          - AdminRepository
          - CustomerRepository
          - ProductRepository
          - PurchasedProductRepository
          - UserRepository

2> Controller

a) Admin Controller


```java
29        @Autowired
30        private IPurchasedProductService prodService;
          1 usage
31        @Autowired
32        private IUserService userService;
33

          no usages
34        @PostMapping("/insertadmin")
35        public Admin insertAdmin(@RequestBody Admin admin) { return adminService.insertAdminInDB(admin); }

          no usages
39        @PutMapping("/updateadminbyid/{adminid}")
40        public void updateAdmin(@PathVariable("adminid") Long adminId, @RequestBody Admin admin) throws ResourceNotFound
41            adminService.updateAdminInDB(admin, adminId);
42        }


          no usages
44        @GetMapping("/getallvalidcustomers")
45        public List<Customer> getAllUsers() { return custService.getAllCustomers(); }
          no usages
48        @GetMapping("/getallcustomerbyname/{custname}")
49        public List<Customer> findByCustomerName(@PathVariable("custname") String customerName){
50            return custService.findByCustomerName(customerName);
51        }
```

b) Customer Controller

```java
35
                                                                                    ⚠8 ✗11 ^  ∨
       no usages
36     @GetMapping("/getcustomer/{userid}")
37     public Customer getCustomer(@PathVariable("userid") Long customerId) throws ResourceNotFound {
38         return customerService.getCustomerInDB(customerId);
39     }
40
       no usages
41     @DeleteMapping("/deletecustomer/{userid}")
42     public void deleteCustomer(@PathVariable("userid") Long customerId) {
43         customerService.deleteCustomerInDB(customerId);
44     }
45
       no usages
46     @PutMapping("/updatecustomerbyid/{userid}")
47     public void updateCustomer(@PathVariable("userid") Long userId, @RequestBody Customer user) throws ResourceNotFoun
48         customerService.updateCustomerInDB(user, userId);
49     }
50
       no usages
51     @PutMapping("/addcustomerproducts/{custid}/{prodid}")
52     public void updateProductInCustomer(@PathVariable("custid") Long userId, @PathVariable("prodid") Long prodid, @Req
53         customerService.insertProductinExistingCustomerInDB(user, userId, prodid);
54     }
```

c) Product Controller


```java
19     @RestController
20     @RequestMapping("/product")
21     public class ProductController {
22
           5 usages
23         @Autowired
24         private IProductService productService;
25
           no usages
26         @PostMapping("/insertproduct")
27         public Product insertProduct(@RequestBody Product newProduct)    {
28             return productService.insertProductInDB(newProduct);
29         }
30
           no usages
31         @GetMapping("/getallproducts")
32         public List<Product> getAllProducts() { return productService.getAllProducts(); }
35
           no usages
36         @GetMapping("/getproduct/{productid}")
37         public Product getProduct(@PathVariable("productid") Long productId) throws ResourceNotFound {
38             return productService.getProductInDB(productId);
39         }
40
           no usages
```

3> Resource Not Found Exception


```java
1   package com.demo.exceptions;

3   public class ResourceNotFound extends Exception {

5       public ResourceNotFound(String exMsg) { super(exMsg); }
8   }
9
```

4> Models

a) Admin

b) Customer

c) Product

d) PurchasedProduct

```
11    @Table(name = "purchased_product")
12    public class PurchasedProduct {
```

```
62
63        no usages
          public int getSize() { return size; }
          3 usages
66        public void setSize(int size) { this.size = size; }
69
          1 usage
70        public String getOrigin() { return origin; }
          4 usages
73        public void setOrigin(String origin) { this.origin = origin; }
76
          3 usages
77        public Float getPrice() { return price; }
          5 usages
80        public void setPrice(Float price) { this.price = price; }
83
          1 usage
84        public int getTag() {
85            return tag;
86        }
          4 usages
87        public void setTag(int tag) { this.tag = tag; }
90
91    }
```

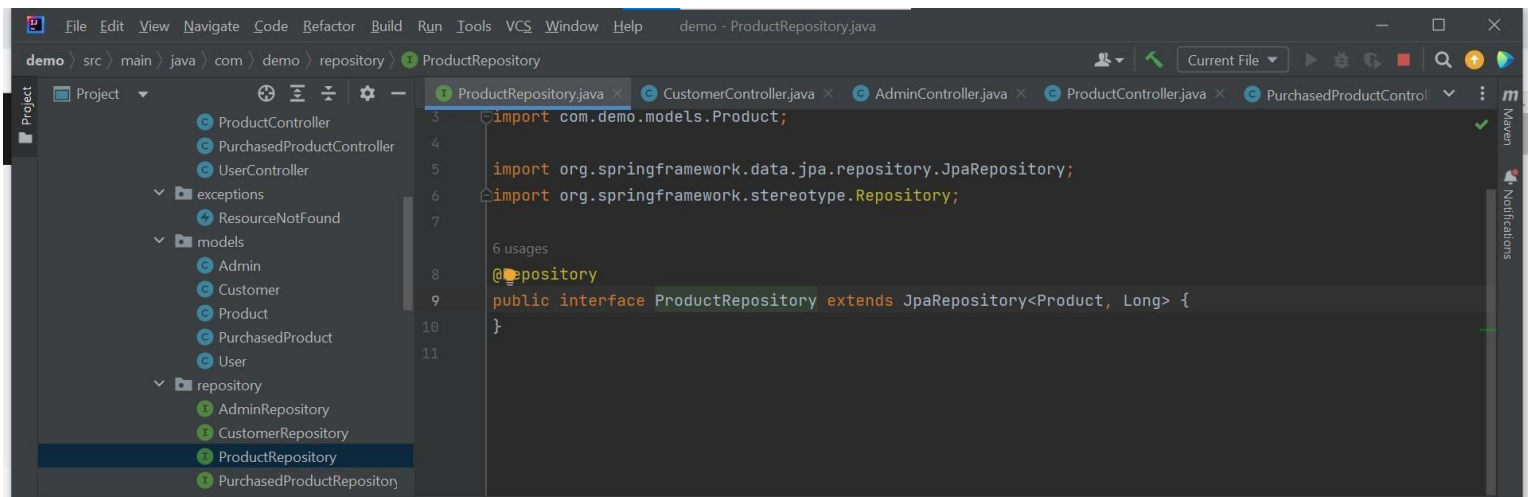e) User

```
52        public void setUserEmail(String userEmail) {
53            this.userEmail = userEmail;
54        }
55
          1 usage
56        public String getUserPwd() {
57            return userPwd;
58        }
59
          1 usage
60        public void setUserPwd(String userPwd) {
61            this.userPwd = userPwd;
62        }
63
          2 usages
64        public Long getUserContact() {
65            return userContact;
66        }
67
          1 usage
68        public void setUserContact(Long userContact) {
69            this.userContact = userContact;
70        }
71
72    }
```

5> Repository

a) Admin Repository

```
      demo › src › main › java › com › demo › repository › AdminRepository

1     package com.demo.repository;
2
3     import com.demo.models.Admin;
4
5     import org.springframework.data.jpa.repository.JpaRepository;
6     import org.springframework.stereotype.Repository;
7
      2 usages
8     @Repository
9     public interface AdminRepository extends JpaRepository<Admin, Long> {
10
11    }
```
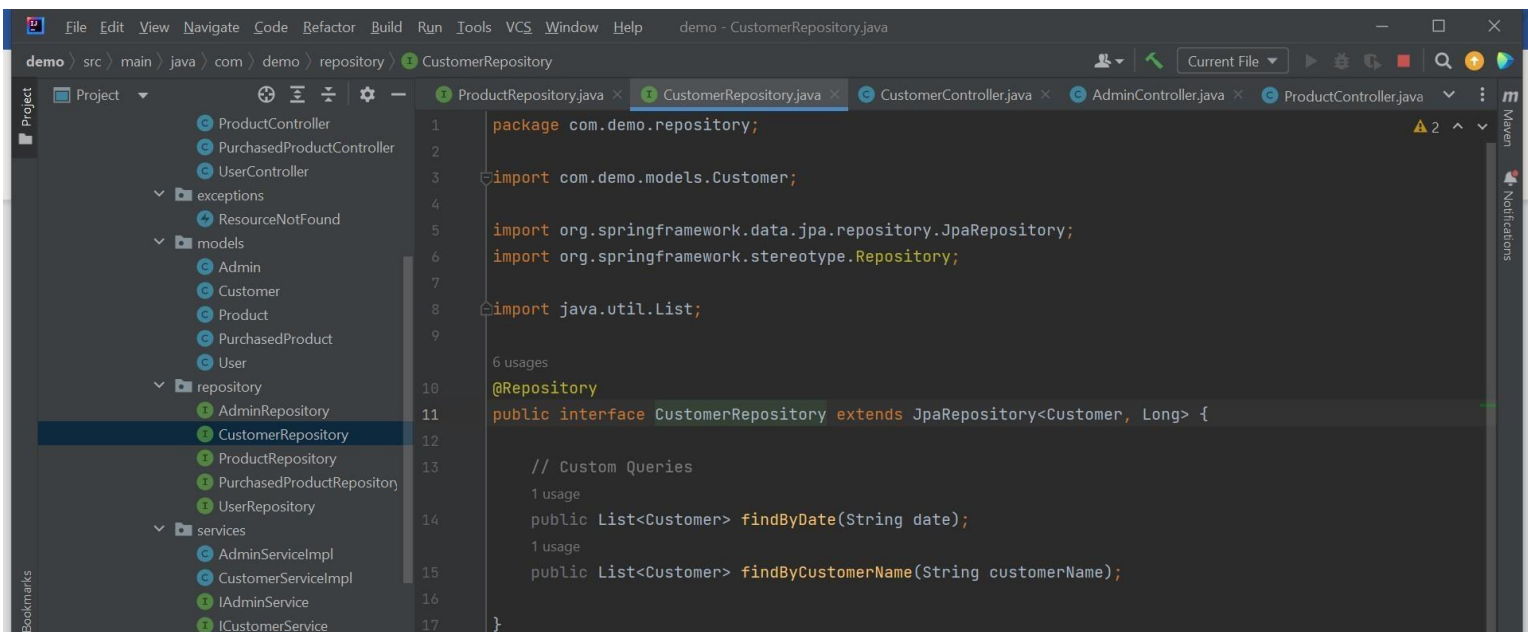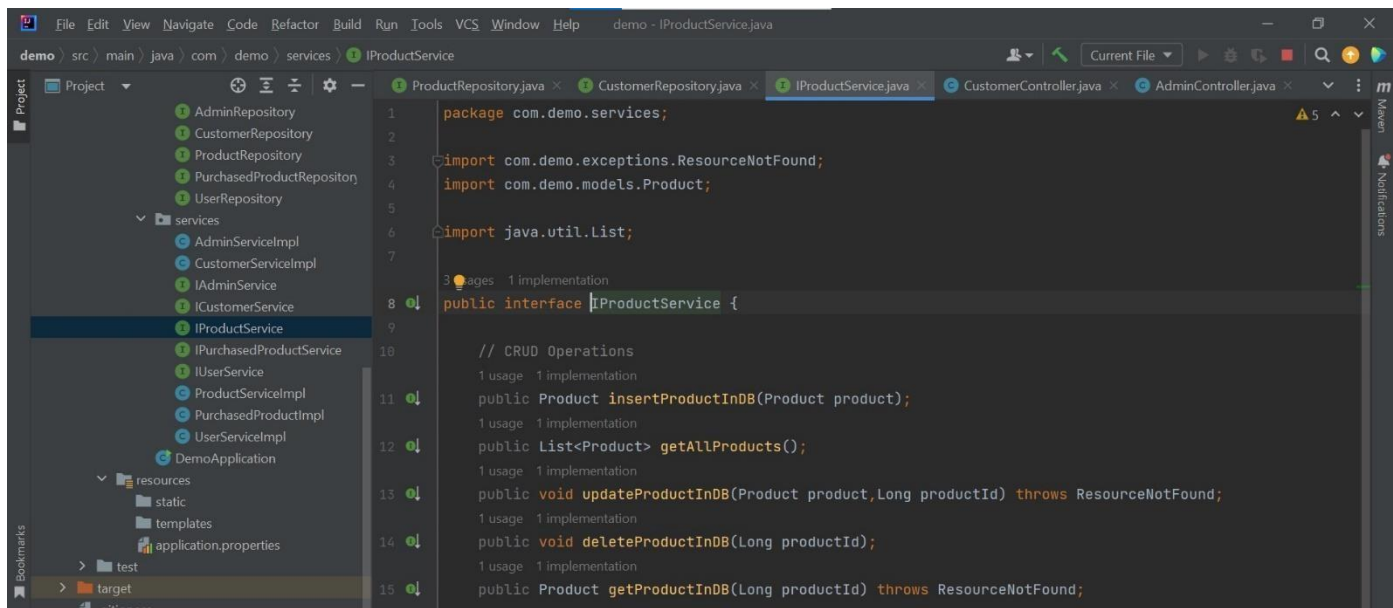
b) Product Repository

c) Customer Repository

Code in ProductRepository.java:

```java
import com.demo.models.Product;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;


@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

Code in CustomerRepository.java:

```java
package com.demo.repository;

import com.demo.models.Customer;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;


@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {

    // Custom Queries
    public List<Customer> findByDate(String date);

    public List<Customer> findByCustomerName(String customerName);

}
```
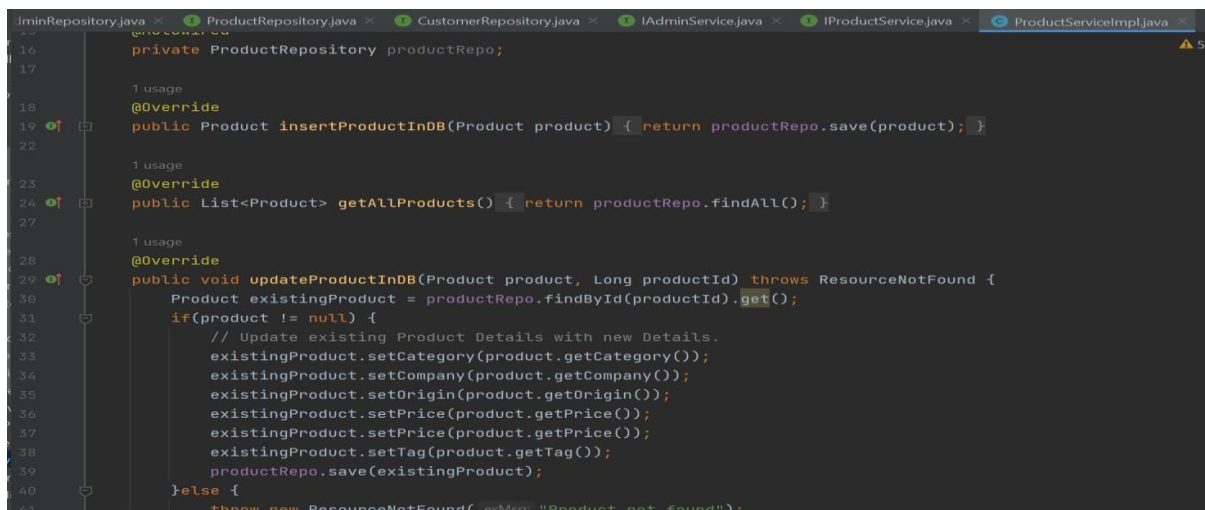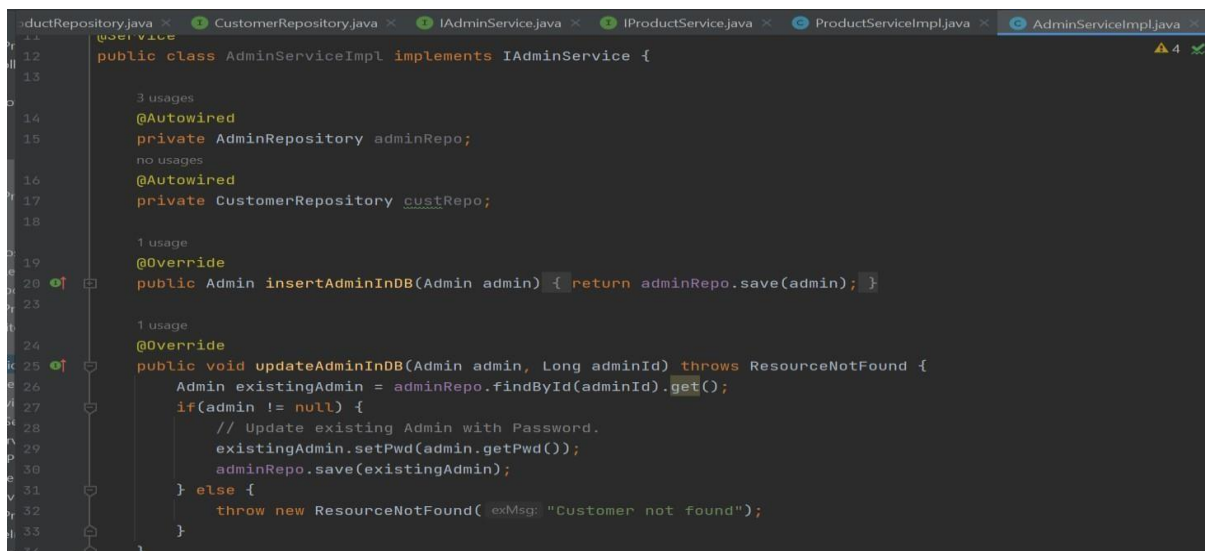
6> Services =>

Interfaces

a) IProduct Interface

=> Classes

a) ProductServiceImpl
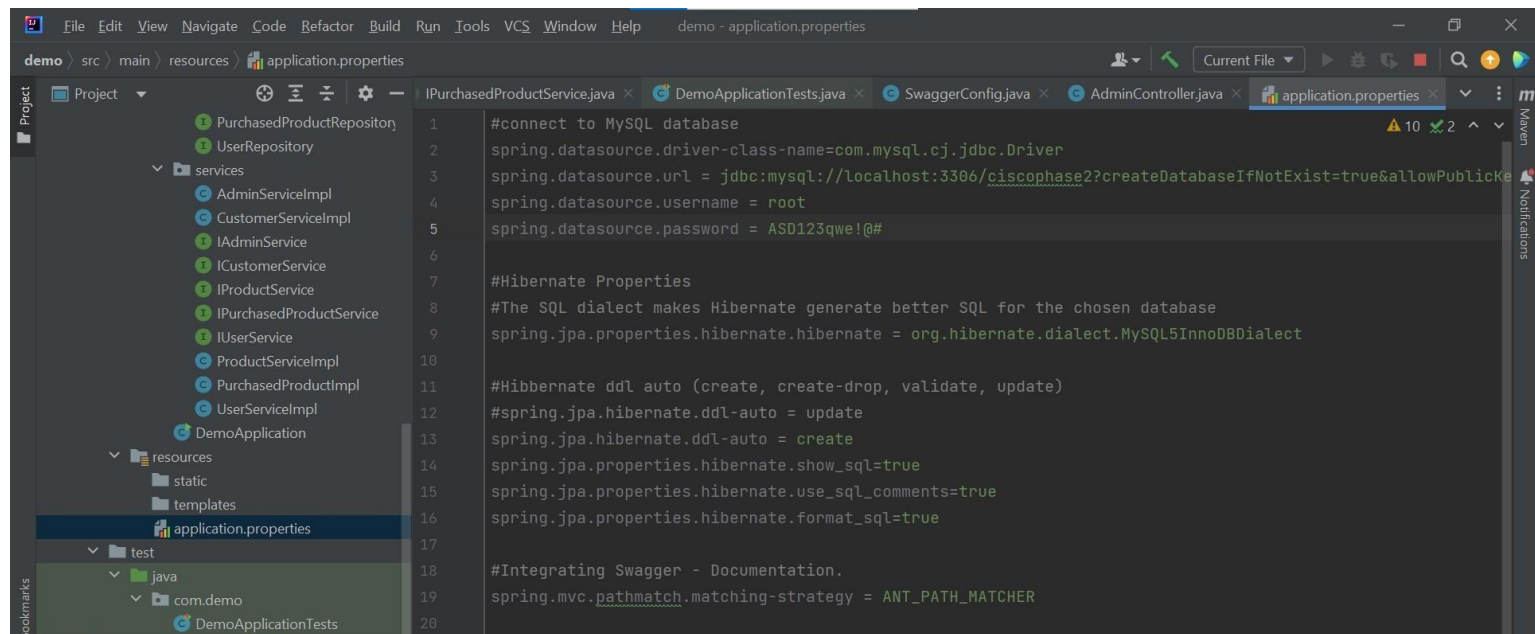
b) AdminServiceImpl

c) UserServiceImpl


```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Set;

@Service
public class UserServiceImpl implements IUserService {

    @Autowired
    UserRepository userRepo;

    @Autowired
    ProductRepository productRepo;

    @Autowired
    CustomerRepository customerRepo;

    Float total = (float) 0;

    @Override
```

7> application.properties


```properties
#connect to MySQL database
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/ciscophase2?createDatabaseIfNotExist=true&allowPublicKe
spring.datasource.username = root
spring.datasource.password = ASD123qwe!@#

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

#Hibbernate ddl auto (create, create-drop, validate, update)
#spring.jpa.hibernate.ddl-auto = update
spring.jpa.hibernate.ddl-auto = create
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.format_sql=true

#Integrating Swagger - Documentation.
spring.mvc.pathmatch.matching-strategy = ANT_PATH_MATCHER
```

Flow Diagram

Postman And Swagger-UI

AdminController

IAdminService
AdminServiceImpl

AdminRepository

Admin
Model

ProductController

IProductService
ProductServiceImpl

ProductRepository

Product
Model

Resource Not Found
Exception

CustomerController

ICustomerService
CustomerServiceImpl

CustomerRepository

Customer
Model

UserController

IUserService
UserServiceImpl

UserRepository

User
Model

PProductController

IPProductService
PProductServiceImpl

PProductRepository

Database

PProduct
Model

Database

✞ Core Concepts used in this project are mostly maven, hibernate, MySQL, spring boot, associations, java, CRUD operations in a database, Postman, and Swagger-UI.

Algorithm

Step 1> Start.

Step 2> The user has the options to go through three sections.

       Case 1: If user select "admin" section then go to step 3 & 4.

       Case 2: If user select "user" section then go to step 5 & 6.

       Case 3: If user select "customer" section, then go to step 7.

Step 3> Once a user select admin, it will show all the admin details along with showing users details that are signed up in this portal.

Step 4> An admin main page will also display the list of customers along with their purchased products. Along with this the admin can also change his credentials.

Step 5> In the "user" section, the user can register and login with their details such as email and password. And the user can purchase the product without the need of entering user details.

Step 6> And user can also check his purchased product history along with its details such as price, purchase date. And user can also change their details along with email and password.

Step 7> If user is not "admin" or "user", they can still purchase the product with the need of fulfilling their details along with getting their receipt, but they cannot access any database of their past product history.

Step 8> Stop

Conclusion

1: The prototype is robust and platform independent.

2: User can easily use the prototype and safely exit out of it.

3: As a developer, we can enhance it by introducing several new features such as dynamic web pages, guards and return once admin has been logout, routing, custom validators and can have more user-friendly by adding styling (CSS, Bootstrap), custom loaders.

4: Though this prototype is tightly connected, the data will only persist in database until server is running and gets reset with "CREATE" option of hibernate in "application.properties".

5: This prototype can also be implemented with multithreading to enable better performance.

6: And lastly, this prototype can be upgraded by making every service as standalone and will be created whenever it is invoked.