sportyshoes.com

Sports Shoes Portal Online

Prototype of the Application

Name : Anurag Sharma

GitHub : https://github.com/Instantgaming2356/JAVAFSD-Project03

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.

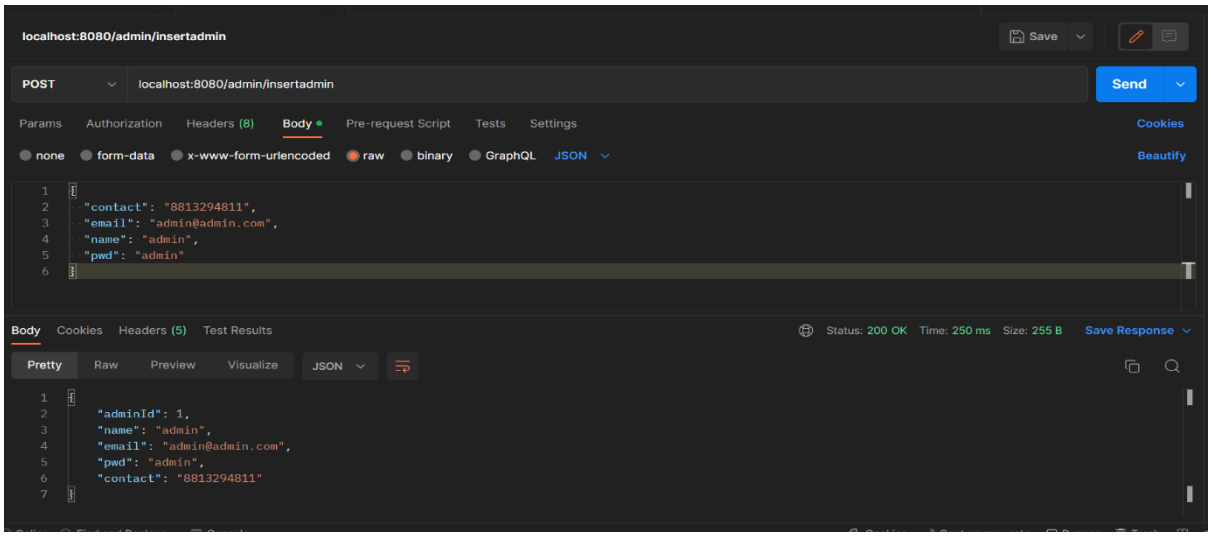
Sprint 5:

- 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

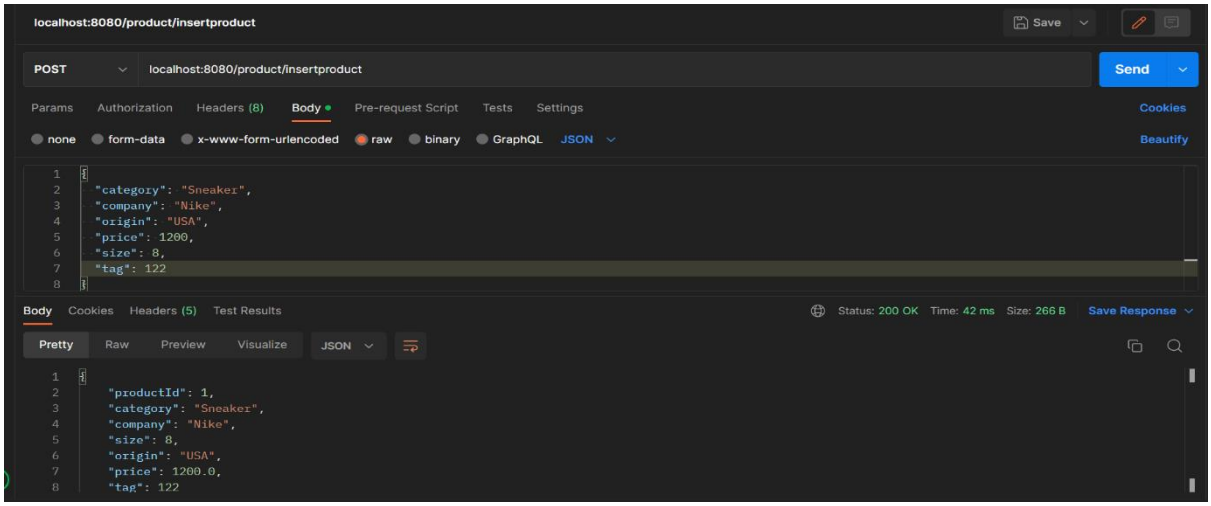
Documentation of the functionality:

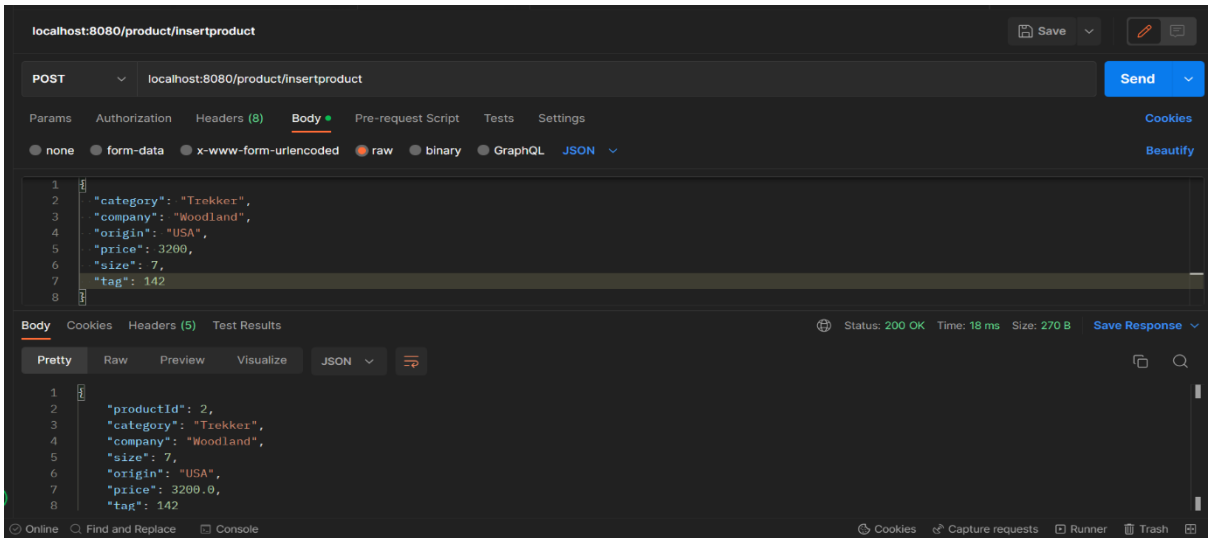
Here are some of the screenshots of testing API through various controllers.

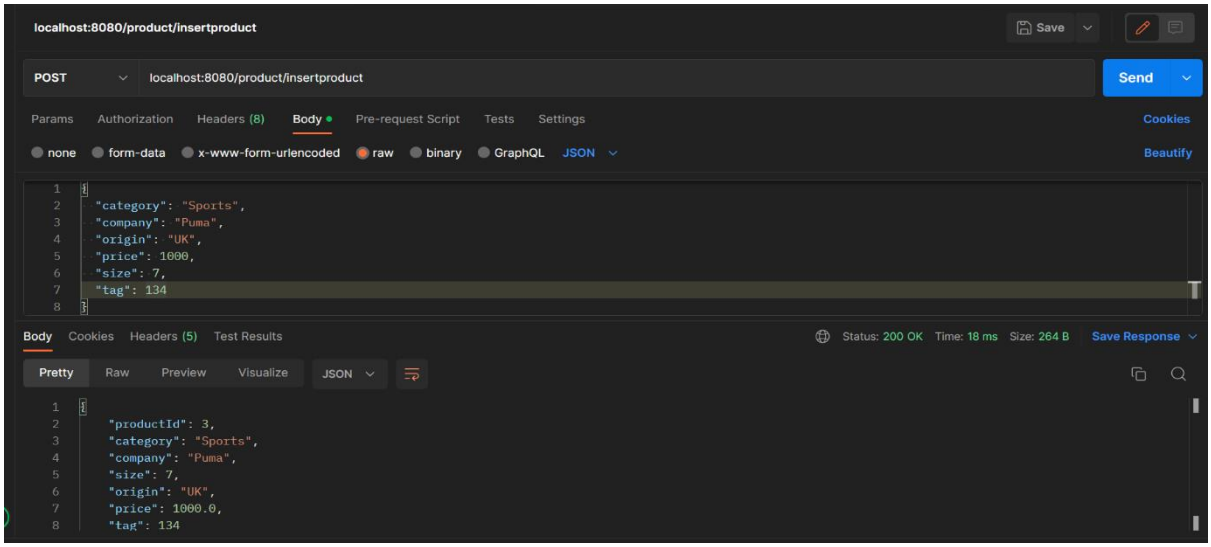
1: Login And Register (Admin Section)



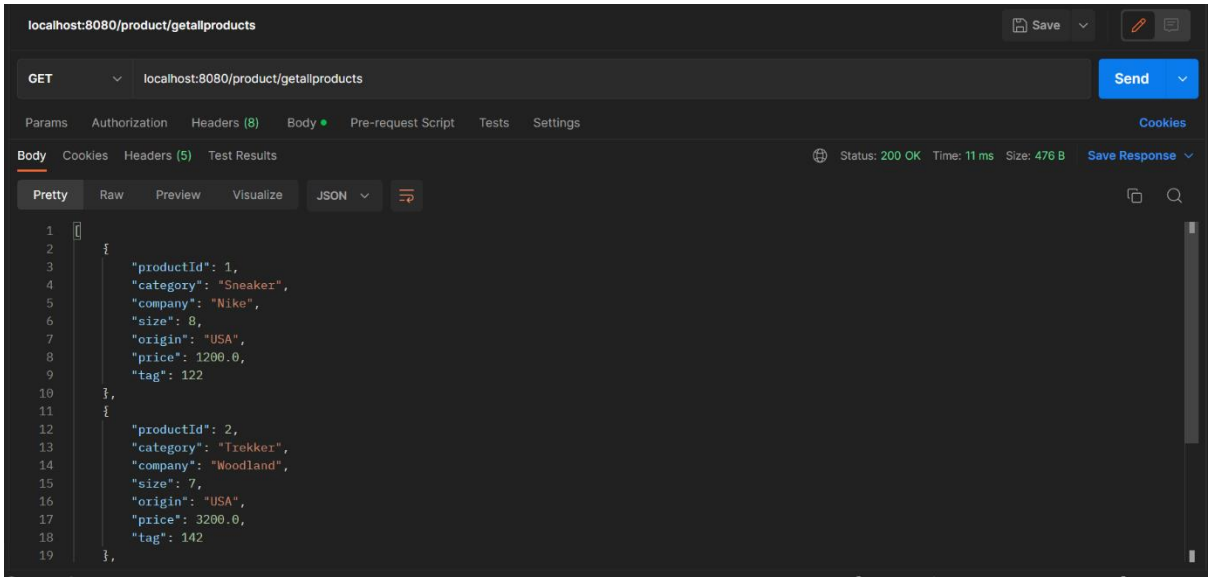
2: Inserting Product (Admin Section)



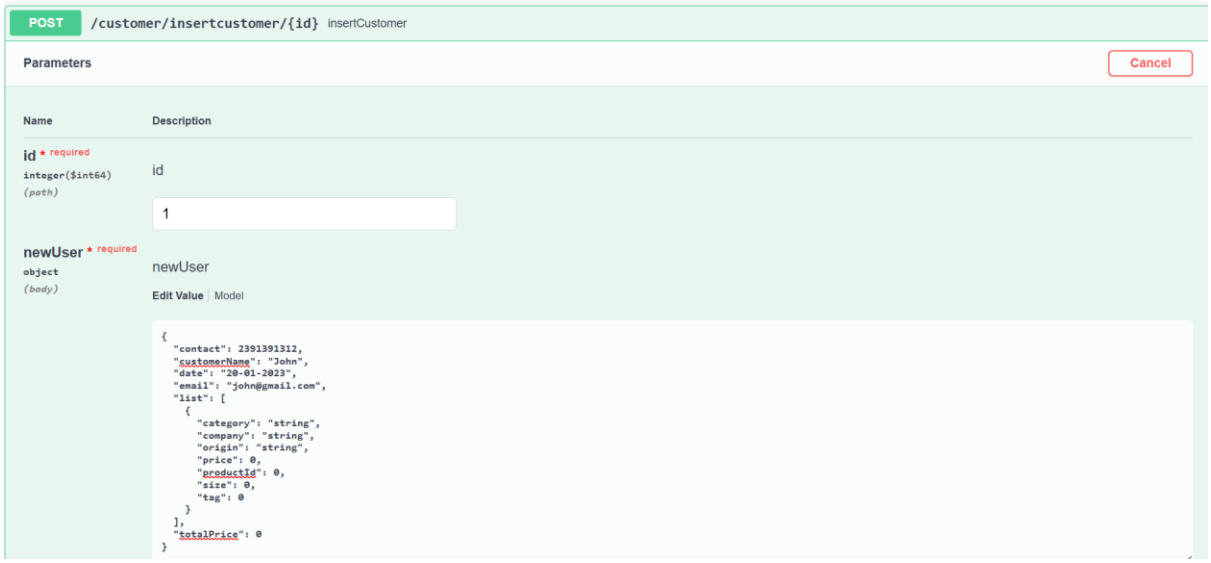




3: Display All Available Products (Admin Section)



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)
(path)

custid

1

prodid * required
integer($int64)
(path)

prodid

2

user * required
object
(body)

user

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)

POST /user/signup insertUser

Parameters

Cancel

Name	Description

newUser * required
object
(body)
	newUser
	Edit Value | Model
	{
	"userContact": 2345910910,
	"userEmail": "jack@gmail.com",
	"userName": "Jack",
	"userPwd": "12345"
	}

Cancel

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

http://localhost:8080/user/insertproductinuser/1/1

Server response

Code	Details

200	Response body
	{
	"customerId": 3,
	"customerName": "Jack",
	"email": "jack@gmail.com",
	"contact": 2345910910,
	"date": "string",
	"totalPrice": 1200,
	"list": [
	{
	"productId": 4,
	"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 12:34:33 GMT
	keep-alive: timeout=60
	transfer-encoding: chunked

Responses

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

custid * required
integer($int64)
(path)
custid
3

prodid * required
integer($int64)
(path)
prodid
2

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


```
{
  "contact": 0,
  "customerId": 0,
  "customerName": "string",
  "date": "22-01-2023",
  "email": "string",
  "list": [
    {
      "category": "string",
      "company": "string",
      "origin": "string",
      "price": 0,
      "productId": 0,
      "size": 0,
      "tag": 0
    }
  ],
  "totalPrice": 0
}
```


Cancel

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 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> 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>
```

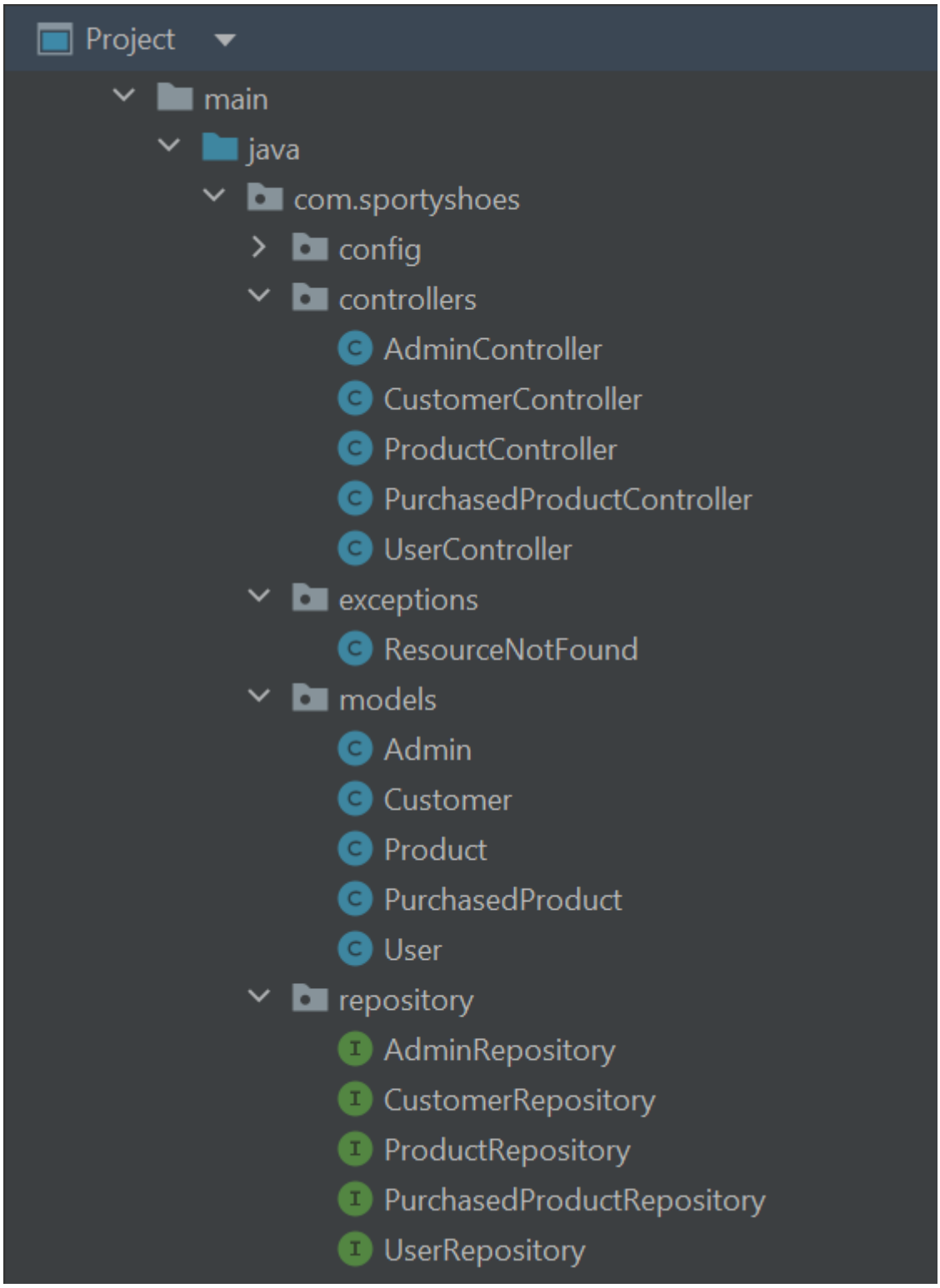
```
mysql> select * from user;
+---------+--------------+----------------+-----------+----------+
| user_id | user_contact | user_email     | user_name | user_pwd |
+---------+--------------+----------------+-----------+----------+
|       1 |   2345910910 | jack@gmail.com | Jack      | 12345    |
+---------+--------------+----------------+-----------+----------+
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> _
```

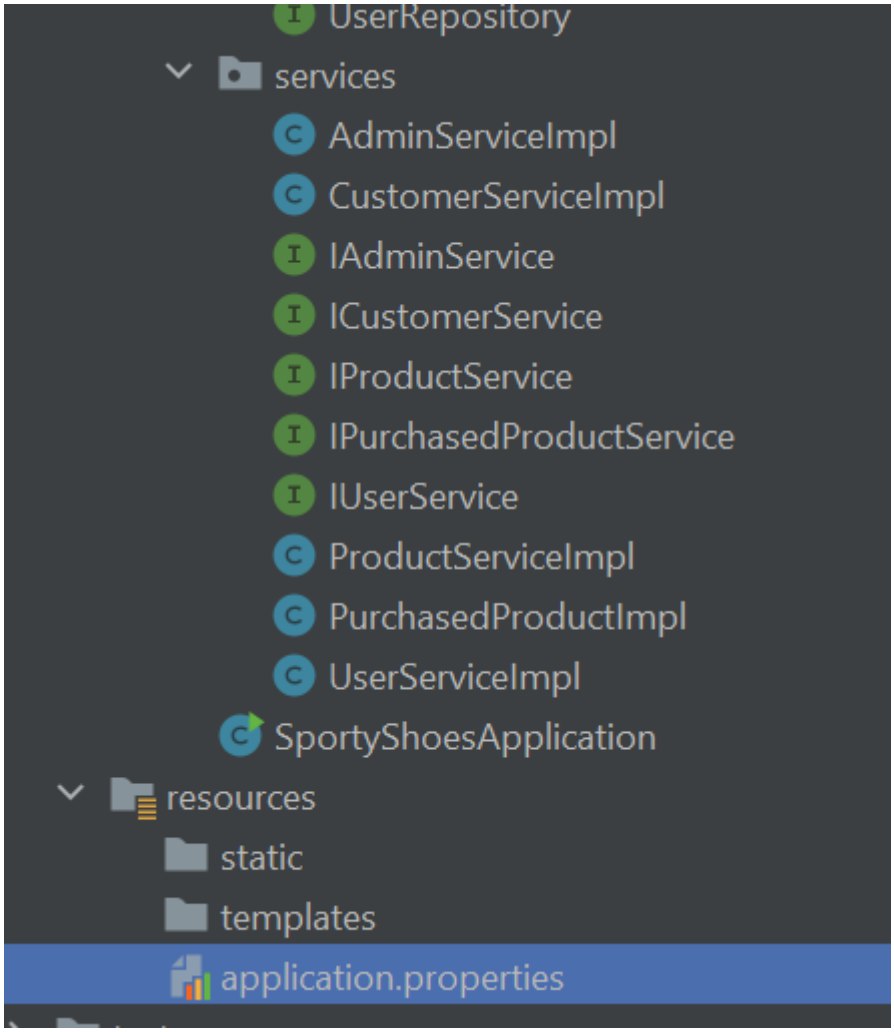
```
MySQL 8.0 Command Line Client
2 rows in set (0.00 sec)

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.

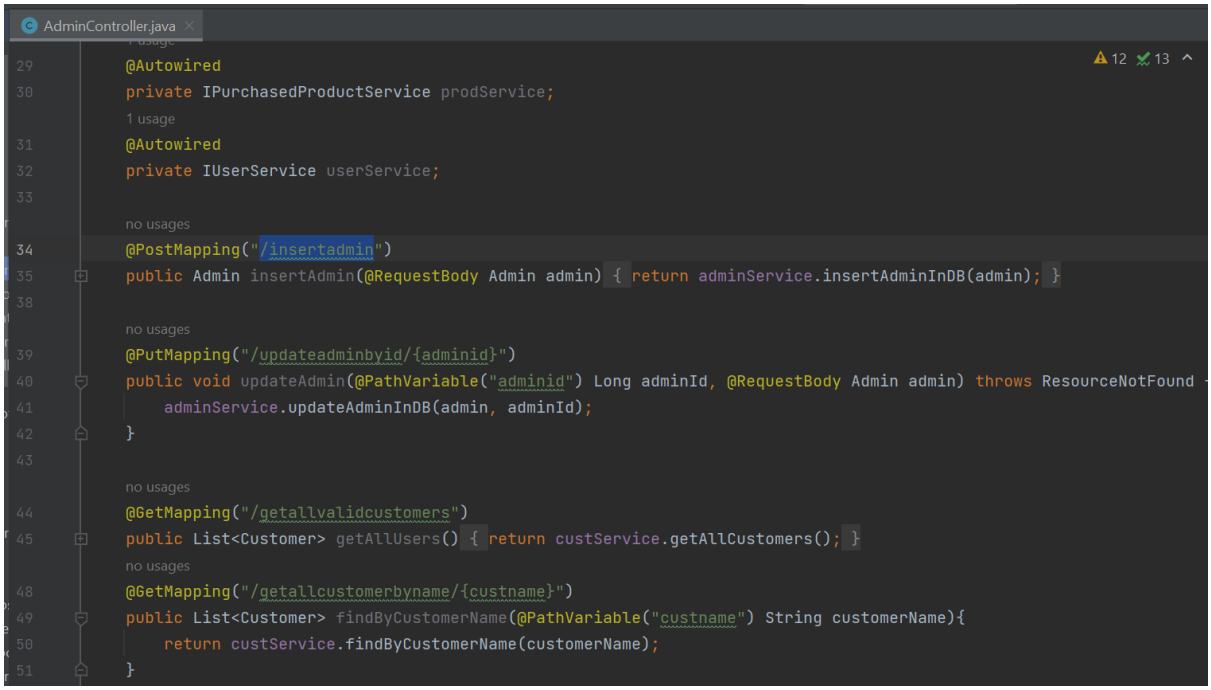
1> Project Structure





2> Controller

a) Admin Controller



```
AdminController.java
    @Autowired
    private IPurchasedProductService prodService;
    1 usage
    @Autowired
    private IUserService userService;

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

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

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

b) Customer Controller

```java
AdminController.java  CustomerController.java

35
        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
AdminController.java  CustomerController.java  ProductController.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
AdminController.java  CustomerController.java  ProductController.java  ResourceNotFound.java

1  package com.sportyshoes.exceptions;
2
3  public class ResourceNotFound extends Exception {
4
5      public ResourceNotFound(String exMsg) { super(exMsg); }
8  }
9
```

4> Models

a) Admin

```java
@Table(name = "admin")
public class Admin {

    2 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "admin_id")
    private Long adminId;

    3 usages
    @Column(name = "admin_name")
    private String name;
    3 usages
    @Column(name = "admin_email")
    private String email;
    3 usages
    @Column(name = "admin_password")
    private String pwd;
    3 usages
    @Column(name = "admin_contact")
    private String contact;

    no usages
    public Admin() {
    }
```

b) Customer

```java
    private Float totalPrice;

    2 usages
    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "customer_id")
    private Set<PurchasedProduct> list = new HashSet<>();

    public Customer() { totalPrice = (float) 0; }
    public Customer(String customerName, String email, Long contact, String date, Float totalPrice) {
        this.customerName = customerName;
        this.email = email;
        this.contact = contact;
        this.date = date;
        this.totalPrice = totalPrice;
    }

    no usages
    public Long getCustomerId() { return customerId; }

    no usages
    public void setCustomerId(Long customerId) { this.customerId = customerId; }

    1 usage
    public String getCustomerName() { return customerName; }
```

c) Product

```java
    3 usages
    public int getSize() { return size; }
    no usages
    public void setSize(int size) { this.size = size; }

    4 usages
    public String getOrigin() { return origin; }
    1 usage
    public void setOrigin(String origin) { this.origin = origin; }

    7 usages
    public Float getPrice() { return price; }
    2 usages
    public void setPrice(Float price) { this.price = price; }

    4 usages
    public int getTag() {
        return tag;
    }
    1 usage
    public void setTag(int tag) { this.tag = tag; }
}
```

d) PurchasedProduct

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

    no usages
    public int getSize() { return size; }
    3 usages
    public void setSize(int size) { this.size = size; }

    1 usage
    public String getOrigin() { return origin; }
    4 usages
    public void setOrigin(String origin) { this.origin = origin; }

    3 usages
    public Float getPrice() { return price; }
    5 usages
    public void setPrice(Float price) { this.price = price; }

    1 usage
    public int getTag() {
        return tag;
    }
    4 usages
    public void setTag(int tag) { this.tag = tag; }

}
```

e) User

```java
    public void setUserEmail(String userEmail) {
        this.userEmail = userEmail;
    }

    1 usage
    public String getUserPwd() {
        return userPwd;
    }

    1 usage
    public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
    }

    2 usages
    public Long getUserContact() {
        return userContact;
    }

    1 usage
    public void setUserContact(Long userContact) {
        this.userContact = userContact;
    }

}
```

5> Repository

a) Admin Repository

```java
package com.sportyshoes.repository;

import com.sportyshoes.models.Admin;

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

2 usages
@Repository
public interface AdminRepository extends JpaRepository<Admin, Long> {
}
```

b) Product Repository

```java
package com.sportyshoes.repository;

import com.sportyshoes.models.Product;

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

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

c) Customer Repository

```java

import com.sportyshoes.models.Customer;

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

import java.util.List;

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

    // Custom Queries
    1 usage
    public List<Customer> findByDate(String date);
    1 usage
    public List<Customer> findByCustomerName(String customerName);

}
```

6> Services

=> Interfaces

a) IProduct Interface

```java
import com.sportyshoes.exceptions.ResourceNotFound;
import com.sportyshoes.models.Product;

import java.util.List;

3 usages  1 implementation
public interface IProductService {

    // CRUD Operations
    1 usage  1 implementation
    public Product insertProductInDB(Product product);
    1 usage  1 implementation
    public List<Product> getAllProducts();
    1 usage  1 implementation
    public void updateProductInDB(Product product,Long productId) throws ResourceNotFound;
    1 usage  1 implementation
    public void deleteProductInDB(Long productId);
    1 usage  1 implementation
    public Product getProductInDB(Long productId) throws ResourceNotFound;

}
```

=> Classes

a) ProductServiceImpl

```java
IminRepository.java  ProductRepository.java  CustomerRepository.java  IAdminService.java  IProductService.java  ProductServiceImpl.java
    private ProductRepository productRepo;

    1 usage
    @Override
    public Product insertProductInDB(Product product) { return productRepo.save(product); }

    1 usage
    @Override
    public List<Product> getAllProducts() { return productRepo.findAll(); }

    1 usage
    @Override
    public void updateProductInDB(Product product, Long productId) throws ResourceNotFound {
        Product existingProduct = productRepo.findById(productId).get();
        if(product != null) {
            // Update existing Product Details with new Details.
            existingProduct.setCategory(product.getCategory());
            existingProduct.setCompany(product.getCompany());
            existingProduct.setOrigin(product.getOrigin());
            existingProduct.setPrice(product.getPrice());
            existingProduct.setPrice(product.getPrice());
            existingProduct.setTag(product.getTag());
            productRepo.save(existingProduct);
        }else {
            throw new ResourceNotFound( exMsg: "Product not found");
```

b) AdminServiceImpl

```java
ductRepository.java  CustomerRepository.java  IAdminService.java  IProductService.java  ProductServiceImpl.java  AdminServiceImpl.java
public class AdminServiceImpl implements IAdminService {

    3 usages
    @Autowired
    private AdminRepository adminRepo;
    no usages
    @Autowired
    private CustomerRepository custRepo;

    1 usage
    @Override
    public Admin insertAdminInDB(Admin admin) { return adminRepo.save(admin); }

    1 usage
    @Override
    public void updateAdminInDB(Admin admin, Long adminId) throws ResourceNotFound {
        Admin existingAdmin = adminRepo.findById(adminId).get();
        if(admin != null) {
            // Update existing Admin with Password.
            existingAdmin.setPwd(admin.getPwd());
            adminRepo.save(existingAdmin);
        } else {
            throw new ResourceNotFound( exMsg: "Customer not found");
        }
    }
```

c) UserServiceImpl

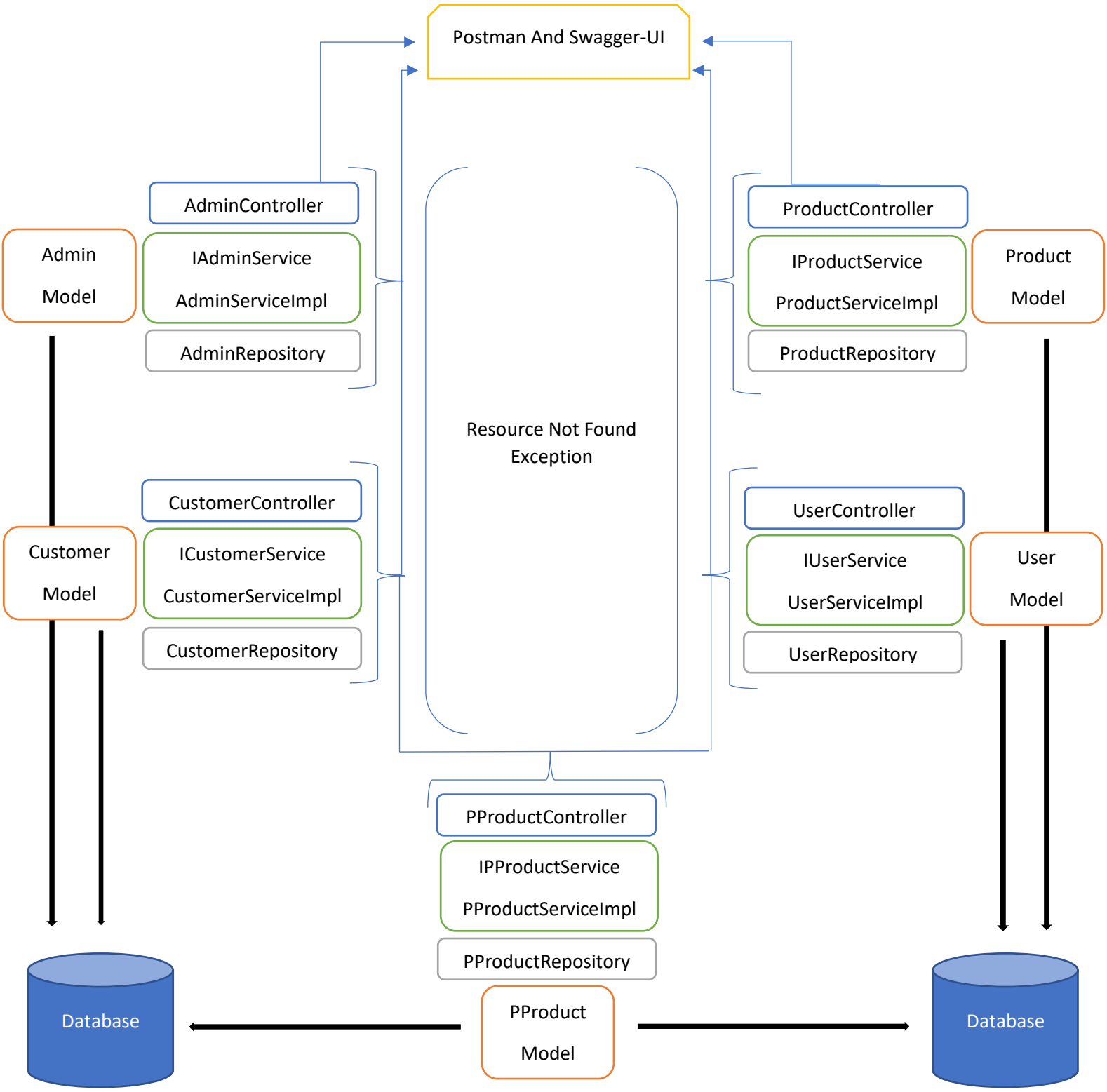
```java
CustomerRepository.java  IAdminService.java  IProductService.java  ProductServiceImpl.java  AdminServiceImpl.java  UserServiceImpl.java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

no usages
@Service
public class UserServiceImpl implements  IUserService {

    5 usages
    @Autowired
    UserRepository userRepo;
    2 usages
    @Autowired
    ProductRepository productRepo;
    1 usage
    @Autowired
    CustomerRepository customerRepo;
    3 usages
    Float total = (float) 0;

    1 usage
    @Override
```

7> application.properties

IAdminService.java | IProductService.java | ProductServiceImpl.java | AdminServiceImpl.java | UserServiceImpl.java | application.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&allowPublicKeyRetrieval=t
spring.datasource.username = root
spring.datasource.password = cisco

#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



- 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.