FINAL PROJECT 2

JAVA ACADEMY - XIDERAL

SEPTEMBER 17, 2024

AUTHOR: REGINA RODRIGUEZ CAMPO GARRIDO

1. Introduction

The application allows to register products in a database and notify subscribed customers when a new product is added in the store. It uses Spring Boot with JPA to manage data persistence and the Observer pattern for handling notifications. Additionally, Spring Security has been implemented, with two types of users: an administrator (ADMIN) who can add (POST) and delete (DELETE) products, and a standard user who can only view (GET) the available products.

2. Project Structure

The project is organized following a layered architecture:

- spring.jpa.dao: This package contains an interface that define repositories. In this case, it includes ProductRepository.
- spring.jpa.entity: This package holds the entity that represent the table in the database.
- spring.jpa.rest: This package contains the REST controllers responsible for handling HTTP requests. The ProductController exposes the endpoints to interact with products, while Security manages security configurations.
- spring.jpa.service: The business logic is encapsulated in this package.
- Pattern: This package contains all the resources necessary to create an Observer Pattern.

3. Implementation

3.1 Controller Layer

ProductController class, is a REST controller responsible for handling HTTP requests related to the Product entity.

```java
public class ProductController {

    private final ProductService productService;

    @Autowired
    public ProductController(ProductService theProductService) {
        productService = theProductService;
    }


    @GetMapping("/products") // Get the list of all products
    public List<Product> findAll() {
        return productService.findAll();
    }

    @GetMapping("/products/{productId}") // Get the products by id
    public Product getProduct(@PathVariable int productId) {
        Product theProduct = productService.findById(productId);
        if (theProduct == null) {
            throw new RuntimeException("Product id not found - " + productId);
        }
        return theProduct;
    }

    @GetMapping("/products/{productId}/stock") //Show Products available
    public String isStock(@PathVariable int productId) {
        boolean inStock = productService.IsStock(productId);
        String message = inStock ? "Product available" : "Product not available";
        return message;
    }

    @GetMapping("/products/discount") //discount applies to items over 15
    public List<String> discount() {
        List<Product> productWithDiscount = productService.PriceWithDiscount(25);
        return productWithDiscount.stream()
                .map(product -> "Title: " + product.getMAKEUP() + ", Price with discount: " + product.getPRICE())
                .collect(Collectors.toList());
    }

    @GetMapping("/products/filter/{price}") //filter products depending on price
    public List<Product> filterProducts(@PathVariable Double price) {
        List<Product> allProducts = productService.findAll();
        return allProducts.stream()
                .filter(product -> product.getPRICE() >= price)
                .collect(Collectors.toList());
    }

    @PostMapping("/products") // Add new product
    public Product addProduct(@RequestBody Product theProduct) {
        theProduct.setID(0);
        return productService.save(theProduct);
    }
```

```java
    @DeleteMapping("/products/{productId}") // Delete product
    public String deleteProduct(@PathVariable int productId) {
        Product tempProduct = productService.findById(productId);
        if (tempProduct == null) {
            throw new RuntimeException("Product id not found - " + productId);
        }
        productService.deleteById(productId);
        return "Deleted product id - " + productId;
    }
}
```

3.2 Service Layer

ProductServiceImp, implements the business logic for managing products in the system.

```java
@Service
public class ProductServiceImpl implements ProductService {

    @PersistenceContext
    private EntityManager entityManager;
    private ProductRepository productrepository;
    private final ProductSubject productSubject = new ProductSubject();

    @Autowired
    public ProductServiceImpl(ProductRepository theproductrepository) {
        productrepository = theproductrepository;
        productSubject.subscribe(new ProductObserver());// observer
    }

    @Override
    public List<Product> findAll() { //find all product
        return productrepository.findAll();
    }

    @Override
   public Product findById(int theId) { //find by ID
    Optional<Product> result = productrepository.findById(theId);
    return result.orElse(null);
    }

    @Override
    public boolean IsStock(int bookId) { //Checks if a product is in stock
        Product product = entityManager.find(Product.class, bookId);
        return product != null && product.getSTOCK() > 0 ;
   }

    @Override
    public List<Product> PriceWithDiscount(double discountPercentage){ //Discount with lambas
         List<Product> products =  findAll();
         return products.stream()
                 .map(product -> {
                     if(product.getPRICE() >15){
                         product.setPRICE(product.getPRICE() - (product.getPRICE() * (discountPercentage / 100)));
                     }  return product;
                 })
                 .collect(Collectors.toList());
    }

    @Transactional
    @Override
    public Product save(Product theproduct) { //Add new product or update of one
         Product savedProduct = productrepository.save(theproduct);
         productSubject.productUp(savedProduct); //notify
         return savedProduct;
    }
```

```java
   @Transactional
   @Override
   public void deleteById(int theId) {//Delete book
       productrepository.deleteById(theId);
   }

   @Override
   public List<Product> filterProducts(Predicate<Product> predicate) {//filter price
       return productrepository.findAll()
               .stream()
               .filter(predicate)
               .toList();
   }
```

3.3 Repository Layer

Repositories interacts with the database using Spring Data JPA

```java
package spring.jpa.dao;

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

public interface ProductRepository extends JpaRepository<Product, Integer> {

}
```

3.4 Entity Layer

The Product class represents a product in the system. It is mapped to the product table in the database and includes fields such as id, makeup, price, category, and stock.

```java
package spring.jpa.entity;

import jakarta.persistence.*;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name="product")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="ID")
    private int ID;

    @Column(name="MAKEUP")
    private String MAKEUP;

    @Column(name="PRICE")
    private double PRICE;

    @Column(name="CATEGORY")
    private String CATEGORY;

    @Column(name="STOCK")
    private int STOCK;


}
```

4 Spring Security Integration

Spring Security is a powerful and highly customizable authentication and access-control framework.

```java
package spring.jpa.rest;

import org.springframework.context.annotation.Bean;

@Configuration
@EnableWebSecurity
public class Security {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .httpBasic()
            .and()
            .authorizeHttpRequests(authorizeRequests ->
                authorizeRequests
                    .requestMatchers(HttpMethod.POST, "/rest/products").hasRole("ADMIN")
                    .requestMatchers(HttpMethod.DELETE, "/rest/products").hasRole("ADMIN")
                    .requestMatchers(HttpMethod.GET, "/rest/products/**").authenticated()
                    .anyRequest().authenticated()
            )
            .csrf(csrf -> csrf.disable());
        return http.build();
    }

    @Bean
    public UserDetailsManager userDetailsManager() {
        UserDetails user1 = User.builder()
                .username("user1")
                .password("{noop}123")
                .roles("ADMIN")
                .build();

        UserDetails user2 = User.builder()
                .username("user2")
                .password("{noop}123")
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(user1, user2);
    }

}
```

5. Database

```sql
CREATE DATABASE makeupDB;
use MakeupDB;

CREATE TABLE product(
ID INT AUTO_INCREMENT PRIMARY KEY,
MAKEUP VARCHAR (50) NOT NULL,
PRICE FLOAT NOT NULL,
CATEGORY VARCHAR(50)NOT NULL,
STOCK INT NOT NULL
);

INSERT INTO PRODUCT (MAKEUP, PRICE, CATEGORY, STOCK) VALUES
('Lipstick', 15.99, 'Cosmetics', 30),
('Foundation', 25.50, 'Base', 20),
('Mascara', 10.75, 'Eye', 25),
('Blush', 12.60, 'Cheek', 15),
('Eye Shadow', 8.90, 'Eye', 40),
('Nail Polish', 6.30, 'Nails', 50),
('Concealer', 18.00, 'Base', 18),
('Bronzer', 14.20, 'Cheek', 22),
('Eyeliner', 9.50, 'Eye', 28),
('Lip Gloss', 13.40, 'Cosmetics', 35);
```

7. Pattern Observer

Observer is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.

7.1 Observer

The Observer interface defines the behavior that must be implemented by all classes that wish to act as observers.

```java
package Pattern;

import spring.jpa.entity.Product;

public interface Observer {
    void update(Product product);
}
```

7.2 Subject

The Subject class is abstract and defines the basic structure for any subject in the Observer pattern. Its main job is to manage the list of watchers that are subscribed and notify them when a change occurs.

```java
package Pattern;

import java.util.ArrayList;

public abstract class Subject {

    private List<Observer> observers = new ArrayList<>();

    public void subscribe(Observer o) {
        observers.add(o);
    }

    public void notifyObservers(Product product) {
        for (Observer o : observers) {
            o.update(product);
        }
    }
}
```

7.3 Product Observer

Represents an observer who is interested in receiving updates when there is a new product.

```java
package Pattern;

import spring.jpa.entity.Product;

public class ProductObserver implements Observer {

    @Override
    public void update(Product product) {
        System.out.println("New product: " + product.getMAKEUP());
    }

}
```

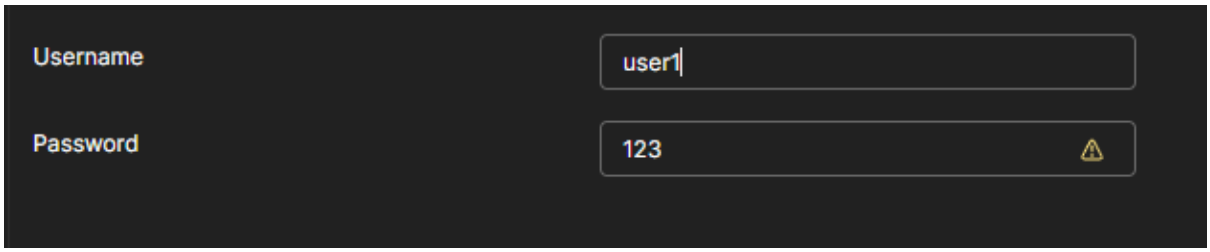
7.4 Product Subject

Is responsible for notifying observers when a product changes or is added.

```java
package Pattern;

import spring.jpa.entity.Product;

public class ProductSubject extends Subject {

    public void productUp(Product product) {
        notifyObservers(product);
    }
}
```

8. OutPut

We enter use 1 which is admi



Get Products


```json
{
    "id": 1,
    "price": 15.99,
    "stock": 30,
    "makeup": "Lipstick",
    "category": "Cosmetics"
},
{
    "id": 2,
    "price": 25.5,
    "stock": 20,
    "makeup": "Foundation",
    "category": "Base"
},
{
    "id": 3,
    "price": 10.75,
    "stock": 25,
    "makeup": "Mascara",
    "category": "Eye"
},
{
    "id": 4,
    "price": 12.6,
    "stock": 15,
    "makeup": "Blush",
    "category": "Cheek"
},
{
    "id": 5,
    "price": 8.9,
    "stock": 40,
    "makeup": "Eye Shadow",
    "category": "Eye"
},
```

FINAL PROJECT 2

Get product by id

GET http://localhost:9096/rest/products/2

Params Authorization Headers (10) Body Scripts Settings

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "id": 2,
    "price": 25.5,
    "stock": 20,
    "makeup": "Foundation",
    "category": "Base"
}
```


Check if product is available

GET http://localhost:9096/rest/products/2/stock

Params Authorization Headers (10) Body Scripts Settings

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize Text

```
Product available
```

It gives you the products with a 25 discount if the price is greater than 15

GET http://localhost:9096/rest/products/discount

Params Authorization Headers (10) Body Scripts Settings

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize JSON


```
[
    "Product: Lipstick, Price with discount: 11.9925",
    "Product: Foundation, Price with discount: 19.125",
    "Product: Mascara, Price with discount: 10.75",
    "Product: Blush, Price with discount: 12.6",
    "Product: Eye Shadow, Price with discount: 8.9",
    "Product: Nail Polish, Price with discount: 6.3",
    "Product: Concealer, Price with discount: 13.5",
    "Product: Bronzer, Price with discount: 14.2",
    "Product: Eyeliner, Price with discount: 9.5",
    "Product: Lip Gloss, Price with discount: 13.4",
    "Product: Test, Price with discount: 19.125",
    "Product: Test2, Price with discount: 22.875",
    "Product: Test3, Price with discount: 37.875",
    "Product: Test4, Price with discount: 37.875",
    "Product: Test5, Price with discount: 37.875",
    "Product: Test6, Price with discount: 37.875"
]
```

Gives products with a price over than 25

GET http://localhost:9096/rest/products/filter/25

Params Authorization Headers (10) Body Scripts Settings

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 2,
        "price": 25.5,
        "stock": 20,
        "makeup": "Foundation",
        "category": "Base"
    },
    {
        "id": 11,
        "price": 25.5,
        "stock": 50,
        "makeup": "Test",
        "category": "Cosmetics"
    },
    {
        "id": 12,
        "price": 30.5,
        "stock": 50,
        "makeup": "Test2",
        "category": "Cosmetics"
    },
    {
        "id": 13,
        "price": 50.5,
        "stock": 10,
        "makeup": "Test3",
        "category": "Cosmetics"
    },
    {
        "id": 14,
        "price": 50.5,
        "stock": 10,
        "makeup": "Test4",
        "category": "Cosmetics"
    },
    {
        "id": 15,
        "price": 50.5,
        "stock": 10,
```


If the user user 2, who is user, wants to add a product, he will not be able to because he is not authorized.

POST http://localhost:9096/rest/products

Params Authorization Headers (10) Body Scripts Settings

Auth Type

Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

Username user2

Password 123

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "timestamp": "2024-09-15T04:10:30.254+00:00",
    "status": 403,
    "error": "Forbidden",
    "message": "Forbidden",
    "path": "/rest/products"
}
```

FINAL PROJECT 2

If we change the user to an administrator it allows us to add a product

POST http://localhost:9096/rest/products

Params Authorization Headers (10) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2         "price": 15.99,
3         "stock": 30,
4         "makeup": "Test17",
5         "category": "Cosmetics"}
6
7
```

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 17,
3     "price": 15.99,
4     "stock": 30,
5     "makeup": "Test17",
6     "category": "Cosmetics"
7 }
```

In the console we get the notification that a new product has been added, here we use the observer pattern

```
New product: Test17
```