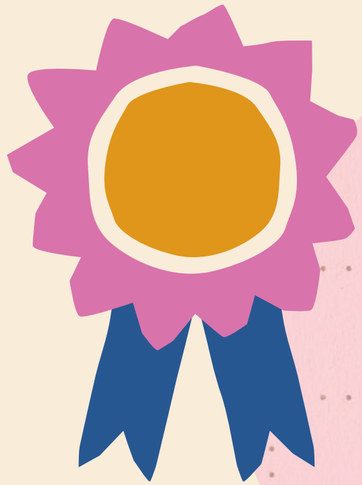
Capstone 1: Amazon Clone

By: Durrah Dahash



5 extra endpoints

First endpoint: Discount product’s price

The **discountOfProduct** endpoint applies a discount to a specific product for a particular user. It validates the user_id and product_id, calculates the new price based on the discount amount, and updates the product's price.

Benefits for website:

1- Enables targeted discounts for specific users, increasing customer engagement and loyalty.

2-Discounts make products more appealing, attracting customers to view and purchase them.

User service

```java
public int discountOfProduct(String user_id, String product_id, double discount_amount) {  1 usage

    List<Double> allowedDiscounts = Arrays.asList(5.0, 10.0, 15.0, 20.0, 25.0);

    ArrayList<Product> products = productService.getProducts();
    if (discount_amount <= 0 || !allowedDiscounts.contains(discount_amount)) {
        return -1;
    }

    for (User u : users) {
        if (u.getId().equals(user_id)) {

            for (Product p : products) {

                if (p.getId().equals(product_id)) {
                    double priceAfterDiscount = (p.getPrice() - (p.getPrice() * discount_amount / 100));
                    p.setPrice(priceAfterDiscount);
                    return 1;
                }
            }
            return -2; //product not found
        }
    } //End for
    return -3;  //user not found
}
```

User controller

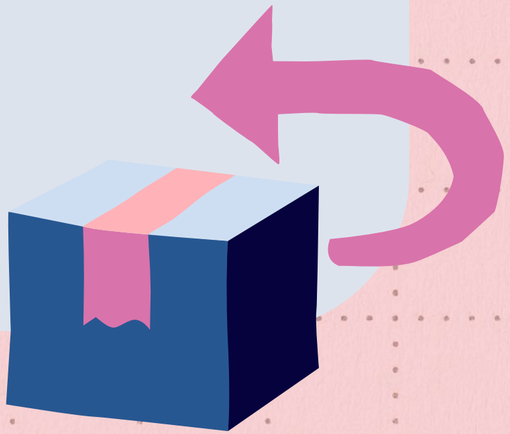
```java
//1. extra endpoint: Apply a discount amount on specific product

@PutMapping("/discount/{user_id}/{product_id}/{discount_amount}")
public ResponseEntity discountOfProduct(@PathVariable String user_id, @PathVariable String product_id, @PathVariable double discount_amount)

    int result = userService.discountOfProduct(user_id, product_id, discount_amount);

    switch (result) {
        case -1:
            return ResponseEntity.status(400).body(new ApiResponse("This discount amount not valid!"));

        case -2:
            return ResponseEntity.status(400).body(new ApiResponse("Product with this ID not found!"));

        case -3:
            return ResponseEntity.status(400).body(new ApiResponse("User with this ID not found!"));

        case 1:
            return ResponseEntity.status(200).body(new ApiResponse("Discount applied successfully for this product!"));

        default:
            return ResponseEntity.status(400).body(new ApiResponse("Not found!"));

    }
```

Second endpoint: Return a product

The **returnProduct** endpoint allows a user to return a previously purchased product. It validates the user_id, product_id, and merchant_id, updates the merchant's stock by increasing it by one, and refunds the product price to the user's balance.

Benefits for website:

Increased Customer Satisfication:
A seamless return process builds trust, encouraging customers to shop more frequently on the platform.



MerchantStock service

(1)

```
public int returnProduct(String user_id, String product_id, String merchant_id) {

    ArrayList<User> users = userService.getUsers();
    ArrayList<Product> products = productService.getProducts();
    ArrayList<Merchant> merchants = merchantService.getMerchants();


    User user = null;
    for (User u : users) {
        if (u.getId().equals(user_id)) {
            user = u; //User is found
            break;
        }
    }
    if (user == null)
        return -1;

    Product product = null;
    for (Product p : products) {
        if (p.getId().equals(product_id)) {
            product = p;  //Product is found
            break;
        }
```

```
if (product == null)
    return -2;


Merchant merchant = null;
for (Merchant m : merchants) {
    if (m.getId().equals(merchant_id)) {
        merchant = m; //Merchant is found
        break;
    }
}

if (merchant == null)
    return -3;


// check if the product was purchased by the user
if (user.getPurchasedProducts() == null || !user.getPurchasedProducts().contains(product_id)) {
    return -4; // Product not purchased by user
}
```

(2)

```
        user.setPurchasedMerchants(new ArrayList<>());

        if (!user.getPurchasedMerchants().contains(merchant_id)) {
            user.getPurchasedMerchants().add(merchant_id); // Add merchant to the list if not already added
        }


        for (MerchantStock merchantStock : merchantStocks) {

            if (merchantStock.getProduct_id().equals(product_id) && merchantStock.getMerchant_id().equals(merchant_id)) {
                merchantStock.setStock(merchantStock.getStock() + 1);
            }
        }


        double returnAmount = product.getPrice();
        user.setBalance(user.getBalance() + returnAmount); //return amount to user

        return 1;
    }
```

MerchantStock controller

```java
@PutMapping("/return-product/{user_id}/{product_id}/{merchant_id}")
public ResponseEntity returnProduct(@PathVariable String user_id, @PathVariable String product_id, @PathVariable String merchant_id) {

    int returnStatus = merchantStockService.returnProduct(user_id, product_id, merchant_id);

    switch (returnStatus) {
        case -1:
            return ResponseEntity.status(400).body(new ApiResponse("User with this ID not found!"));
        case -2:
            return ResponseEntity.status(400).body(new ApiResponse("Product with this ID not found!"));
        case -3:
            return ResponseEntity.status(400).body(new ApiResponse("Merchant with this ID not found!"));

        case -4:
            return ResponseEntity.status(400).body(new ApiResponse("This product not purchased from user!"));

        case 1:
            return ResponseEntity.status(200).body(new ApiResponse("Return operation done successfully!"));
        default:
            return ResponseEntity.status(400).body(new ApiResponse("Not found!"));
    }

}
```

Third endpoint: Rate the merchant

The **merchantRating** endpoint allows users to rate a merchant on a scale of 0 to 5. It validates the merchant_id, and rating, updates the merchant's total number of ratings, and recalculates the average rating based on the new input.

Benefits for website:

1-Merchants are motivated to maintain high ratings by providing better products.

2- Encourages user interaction by giving them a platform to share their feedback.

MerchantStock service

```java
public int merchantRating(String user_id, String merchant_id, int rating) {

    ArrayList<User> users = userService.getUsers();
    ArrayList<Merchant> merchants = merchantService.getMerchants();

    if (rating < 0 || rating > 5) {
        return -2; // Invalid rating
    }

    User user = null;
    for (User u : users) {
        if (u.getId().equals(user_id)) {
            user = u;
            break;
        }
    }
    if (user == null) {
        return -3; // User not found
    }

    Merchant merchant = null;
    for (Merchant m : merchants) {
        if (m.getId().equals(merchant_id)) {
    if (merchant == null) {
        return -1; // Merchant not found
    }

    // Check if the user has purchased from this merchant
    if (user.getPurchasedMerchants() == null || !user.getPurchasedMerchants().contains(merchant_id)) {
        return -4; // User has not purchased from this merchant
    }


    merchant.setNumOfRating(merchant.getNumOfRating() + 1);
    merchant.setRatingAvg(
            ((merchant.getRatingAvg() * (merchant.getNumOfRating() - 1)) + rating) / merchant.getNumOfRating()
    );

    return 1; // Success
}
```

MerchantStock controller

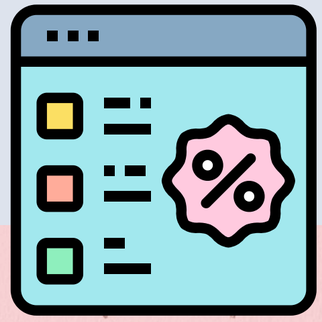
```java
@PutMapping("/merchant-rating/{user_id}/{merchant_id}/{rating}")
public ResponseEntity merchantRating(@PathVariable String user_id, @PathVariable String merchant_id, @PathVariable int rating) {


    int ratingResult = merchantStockService.merchantRating(user_id, merchant_id, rating);

   switch ((ratingResult)){

        case -2:
            return ResponseEntity.status(400).body(new ApiResponse("Invalid rating value! Must be in this range(0-5)."));
        case -3:
            return ResponseEntity.status(400).body(new ApiResponse("User not found!"));
        case -1:
            return ResponseEntity.status(400).body(new ApiResponse("Merchant not found!"));

        case -4:
            return ResponseEntity.status(400).body(new ApiResponse("User doesn't purchase from this merchant until now!"));

        case 1:
            return ResponseEntity.status(200).body(new ApiResponse("Rating created to this merchant successfully!"));

        default:
            return ResponseEntity.status(400).body(new ApiResponse("Not found!"));
```

Fourth endpoint: Discount by category

The **discountByCategory** endpoint applies a discount to all products within a specific category. It verifies the user_id to ensure the user has an "Admin" role, then checks for the product existence, and updates the prices of the associated products based on the discount amount.

Benefits for website:

Makes discounted categories visible and attractive to customers, boosting traffic and engagement.



User service

```java
public int discountByCategory(String user_id, String category_id, double discount_amount) {  1 usage

    ArrayList<Product> products = productService.getProducts();
    for (User u : users) {
        if (u.getId().equals(user_id)) {

            if (!u.getRole().equalsIgnoreCase( anotherString: "Admin")) {
                return -1;  //Must be an admin
            }

            boolean isProductFound = false; //in this category
            for (Product p : products) {

                if (p.getCategory_id().equals(category_id)) {
                    isProductFound = true;


                    double discountAmount = p.getPrice() * (discount_amount / 100);
                        double priceAfterDiscount = p.getPrice() - discountAmount;

                        //update product's price
                        p.setPrice(priceAfterDiscount);

                    }
                }
                if(!isProductFound) {
                    return -2;
                }
                //product not found

                return 1;

            }

        } //End for
        return -3;  //user not found
```

User controller

```java
@PutMapping("/discount-byCategory/{user_id}/{category_id}/{discount_amount}")
public ResponseEntity discountByCategory(@PathVariable String user_id, @PathVariable String category_id,@PathVariable double discount_amount){

    int discountStatus = userService.discountByCategory(user_id, category_id, discount_amount);

    switch (discountStatus){

        case -1:
            return ResponseEntity.status(400).body(new ApiResponse("User's role must be Admin to apply discount on specific category!"));

        case 1:
            return ResponseEntity.status(200).body(new ApiResponse("Discount applied successfully for products in category: " + category_id));

        case -2:
            return ResponseEntity.status(400).body(new ApiResponse("No product found in category: " + category_id));

        case -3:
            return ResponseEntity.status(400).body(new ApiResponse("User with this ID not found!"));

        default:
            return ResponseEntity.status(400).body(new ApiResponse("Not found!"));

    }
```

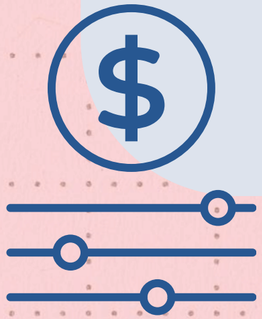
Fifth endpoint: Find product by range

The **findByPriceRange** endpoint filters products within a specified price range (minPrice to maxPrice). It iterates through all products, checks their price, and adds matching products to a list for retrieval.

Benefits for website:

1-Allows users to find products that fit their budget quickly and easily.

2-Assential feature for e-commerce platforms, improving usability and satisfaction.



Product service

```
public ArrayList<Product> findByPriceRange(double minPrice, double maxPrice) {  no usages
    ArrayList<Product> productsByPrice = new ArrayList<>();

    for (Product product : products) {
        if (product.getPrice() >= minPrice && product.getPrice() <= maxPrice) {
            productsByPrice.add(product);
        }
    }

    return productsByPrice;
}
```

Product controller

```java
//5. extra: User can search for a product with specific price range
@GetMapping("/products-byPrice/{minPrice}/{maxPrice}")
public ResponseEntity findByPriceRange(@PathVariable double minPrice, @PathVariable double maxPrice){

    ArrayList<Product> products = productService.findByPriceRange(minPrice, maxPrice);

    return ResponseEntity.status(200).body(products);


}
```

Thank you all for your time