INDUSTRIAL TRAINING PROJECT REPORT

ON

SPRING BOOT APPLICATION

FOR

ELECTRONIC STORE

SUBMITTED TO

HINDALCO INDUSTRIES LTD. MURI,

RANCHI,JHARKHAND

REPORT SUBMITTED BY

KRITIKA DAS

(BTECH /CSE21068/2101020068)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Index

S.NO	TITLE
1	CERTIFICATE
2	ACKNOWLEDGEMENT
3	DECLARATION
4	INTRODUCTION
5	PROPOSED MODEL
6	PRODUCT CODE
7	PRODUCT OUTPUT
8	CONCLUSION

CERTIFICATE

This is to certify that Ms. Kritika Das (Reg. No:2101020068) Computer Science VI semester student of CV RAMAN GLOBAL UNIVERSITY BHUBANESWAR, has done project work entitled “ELECTRONIC STORE” in SPRING BOOT under the guidance of our senior faculties towards the fulfillment of the project during the period of 22 May 2024 to 22 July 2024.

Under the guidance of

MR.MAHTAB ALAM

Project In charge

MR.ATUL KUMAR

Head.IT Department

HINDALCO INDUSTRIES LTD.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude and appreciation to everyone who contributed to the successful completion of this project and made it possible for me to present this comprehensive analysis and optimization of production processes at Hindalco Industries Ltd.

I am deeply grateful to the management team at Hindalco Industries Ltd.,for granting me the opportunity to conduct this study and providing access to the necessary resources, facilities, and data. Their encouragement and trust in my abilities empowered me to delve into complex aspects of production processes and analyze them comprehensively.

I would like to thank the entire staff and colleagues at Hindalco Industries Ltd. who generously shared their insights and expertise during interviews, discussions, and data collection phases. Their cooperation and willingness to assist me were invaluable in gaining a holistic understanding of the production operations and challenges.

Special thanks are due to my peers and friends who offered support, encouragement, and understanding throughout this journey. Their moral support and constructive discussions have been a source of motivation and inspiration.

In conclusion, this project would not have been possible without the collective efforts and support of all individuals and entities mentioned above. I am deeply appreciative of their contributions and assistance in making this project a success.

Sincerely

Kritika Das

Declaration

I, Kritika Das, hereby declare that this project report titled "Comprehensive Analysis and Optimization of Production Processes in Hindalco Industries Ltd." is a result of my own work and research, conducted under the guidance of Mr Atul Kumar, Head.IT Department at Hindalco Industries Ltd.

All sources of information and data used in this report have been duly acknowledged. This work has not been submitted, in whole or in part, for any other degree or qualification at this or any other university or institution.

I acknowledge that Hindalco Industries Ltd. provided access to the necessary resources and facilities for conducting this research. I take full responsibility for the accuracy and integrity of the information presented in this report.

ELECTRONIC STORE

July 19, 2024

Introduction

An electronic store is a retail establishment that specializes in selling a wide range of electronic products. These stores offer various items including, but not limited to, an electronic store is a specialized retail outlet that offers a wide array of electronic devices and gadgets. These stores provide consumers with products such as smartphones, laptops, televisions, home appliances, and computer accessories. They cater to the tech-savvy customer base by ensuring that the latest innovations and high-quality electronic goods are readily available. The primary goal of an electronic store is to offer convenience, variety, and excellent customer service, enhancing the shopping experience for customers.

In addition to providing a diverse range of products, electronic stores often offer value-added services such as technical support, installation services, and product demonstrations. Many electronic stores also feature knowledgeable staff who can assist customers in making informed decisions based on their specific needs and preferences. With the increasing integration of online and offline retail channels, electronic stores frequently offer online shopping options, allowing customers to browse, compare, and purchase products from the comfort of their homes while still enjoying the benefits of in-store pickup and after-sales service

PROJECT OVERVIEW

This Electronic Store project is a sophisticated e-commerce platform built using Spring Boot, designed to provide a seamless shopping experience for users. The application allows users to browse and purchase a wide range of electronic devices and gadgets, including smartphones, laptops, televisions, home appliances, and computer accessories.

Key Functionalities:

1. **User Management:**
    - **Authentication and Authorization:** Implemented using Spring Security to ensure secure access to the platform. Users can register, log in, and access their accounts securely.
    - **Role-Based Access:** Different roles (e.g., admin, customer) are managed to control access to various parts of the application.
2. **Shopping Cart:**
    - **Add Items to Cart:** Users can add products to their shopping cart. The application ensures that product details such as price and quantity are accurately updated.
    - **Remove Items from Cart:** Users can remove individual items from their cart.
    - **Clear Cart:** Provides functionality for users to clear all items from their cart.
    - **View Cart:** Users can view the contents of their cart, including product details and total price.
3. **Product Management:**
    - **Product Catalog:** A comprehensive catalog of electronic products is available for browsing. Products are categorized for easy navigation.
    - **Product Details:** Detailed information about each product, including specifications, pricing, and availability.
4. **Order Management:**

    - **Checkout Process:** A streamlined checkout process allows users to review their cart, enter shipping details, and place orders.
    - **Order History:** Users can view their past orders and track the status of current orders.
5. **Custom Exceptions:**
    - Implemented custom exception handling to manage errors gracefully and provide informative feedback to users.
6. **Data Transfer and Mapping:**
    - **ModelMapper:** Utilized for efficient mapping between entity objects and data transfer objects (DTOs), ensuring smooth data flow within the application.

Technologies Used:

- **Spring Boot:** Provides the foundation for the application, offering a robust and scalable backend framework.
- **Spring Security:** Ensures secure authentication and authorization.
- **ModelMapper:** Facilitates seamless data transfer between different layers of the application.
- **H2 Database:** A lightweight, in-memory database used for development and testing.

This project aims to deliver a user-friendly, secure, and efficient online shopping experience, leveraging the power of Spring Boot to handle complex backend operations with ease.

PROPOSED MODEL:

1. Auth Controller

- Endpoints:

-GET /auth/current: Retrieves information about the current authenticated user.

-POST /auth/login: Handles user authentication and generates a JWT token for accessing protected resources.

2. Basic Error Controller

- Endpoints:

- GET /error: Handles HTML error responses.

- HEAD /error: Handles HTML error responses.

- POST /error: Handles HTML error responses.

- PUT /error: Handles HTML error responses.

- DELETE /error: Handles HTML error responses.

- OPTIONS /error: Handles HTML error responses.

- PATCH /error: Handles HTML error responses.

3. Cart Controller

- Endpoints:

- GET /carts/{userId}: Retrieves the cart for a specific user.

- POST /carts/{userId}: Adds an item to the cart for a specific user.

- DELETE /carts/{userId}: Clears the cart for a specific user.

- DELETE /carts/{userId}/items/{itemId}: Removes a specific item from the cart for a user.

4. Category Controller

- Endpoints:

- GET /categories: Retrieves all categories of products.

- POST /categories: Creates a new product category.

- GET /categories/{categoryId}: Retrieves details of a specific category.

- PUT /categories/{categoryId}: Updates details of a specific category.

- DELETE /categories/{categoryId}: Deletes a specific category.

- GET /categories/{categoryId}/products: Retrieves all products associated with a specific category.

- POST /categories/{categoryId}/products: Creates a new product associated with a specific category.

- PUT /categories/{categoryId}/products/{productId}: Updates the category association of a specific product.

5. Home Controller

- Endpoints:

- GET /test: Endpoint for testing purposes.

6. Order Controller

- Endpoints:

- GET /orders: Retrieves all orders.

-POST /orders: Creates a new order.

-DELETE /orders/{orderId}: Deletes a specific order.

- GET /orders/users/{userId}: Retrieves all orders associated with a specific user.

7. Product Controller

- Endpoints:

-GET /products: Retrieves all products.

-POST /products: Creates a new product.

-GET /products/{productId}: Retrieves details of a specific product.

-PUT /products/{productId}: Updates details of a specific product.

-DELETE /products/{productId}: Deletes a specific product.

-GET /products/image/{productId}: Retrieves the image associated with a specific product.

- POST /products/image/{productId}: Uploads an image for a specific product.

- GET /products/live: Retrieves all live products.

- GET /products/search/{query}: Searches for products based on a search query.

8. User Controller

- Endpoints:

  - GET /users: Retrieves all users.

  - POST /users: Creates a new user.

  - GET /users/{userId}: Retrieves details of a specific user by user ID.

  - PUT /users/{userId}: Updates details of a specific user.

  - DELETE /users/{userId}: Deletes a specific user.

  - GET /users/email/{email}: Retrieves details of a user by email address.

  - GET /users/image/{userId}: Retrieves the image associated with a specific user.

  - POST /users/image/{userId}: Uploads an image for a specific user.

  - GET /users/search/{keywords}: Searches for users based on keywords.

PRODUCT CODES:

A)CONTROLLERS

1)userController

```
package com.lcwd.electronic.store.controllers;


import com.lcwd.electronic.store.dtos.ApiResponseMessage;

import com.lcwd.electronic.store.dtos.ImageResponse;

import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.dtos.UserDto;

import com.lcwd.electronic.store.services.FileService;

import com.lcwd.electronic.store.services.UserService;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiOperation;

import io.swagger.annotations.ApiResponse;

import io.swagger.annotations.ApiResponses;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;
```

```
import org.springframework.util.StreamUtils;

import org.springframework.web.bind.annotation.*;

import org.springframework.web.multipart.MultipartFile;


import javax.servlet.http.HttpServletResponse;

import javax.validation.Valid;

import java.awt.*;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;



@RestController

@RequestMapping("/users")

@Api(value = "UserController", description = "REST APIs related to perform user
operations !!")

public class UserController {


   @Autowired

   private UserService userService;


   @Autowired

   private FileService fileService;
```

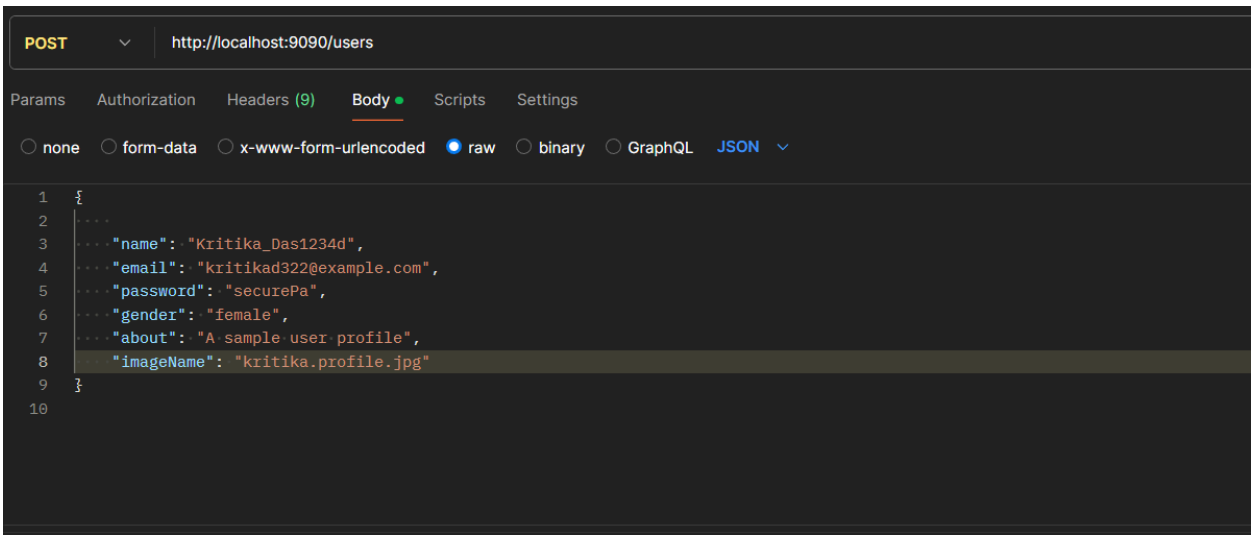
```
    @Value("${user.profile.image.path}")
    private String imageUploadPath;

    private Logger logger = LoggerFactory.getLogger(UserController.class);

    //create
    @PostMapping
    @ApiOperation(value = "create new user !!")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "Success | OK"),
            @ApiResponse(code = 401, message = "not authorized !!"),
            @ApiResponse(code = 201, message = "new user created !!")
    })
    public ResponseEntity<UserDto> createUser(@Valid @RequestBody UserDto
userDto) {
        UserDto userDto1 = userService.createUser(userDto);
        return new ResponseEntity<>(userDto1, HttpStatus.CREATED);
    }

    //update
    @PutMapping("/{userId}")
    public ResponseEntity<UserDto> updateUser(
            @PathVariable("userId") String userId,
            @Valid @RequestBody UserDto userDto
```

```
    ) {
        UserDto updatedUserDto = userService.updateUser(userDto, userId);
        return new ResponseEntity<>(updatedUserDto, HttpStatus.OK);
    }


    //delete
    @DeleteMapping("/{userId}")
    public ResponseEntity<ApiResponseMessage> deleteUser(@PathVariable String
userId) {
        userService.deleteUser(userId);
        ApiResponseMessage message
                = ApiResponseMessage
                .builder()
                .message("User is deleted Successfully !!")
                .success(true)
                .status(HttpStatus.OK)
                .build();

        return new ResponseEntity<>(message, HttpStatus.OK);
    }


    //get all
    @GetMapping
```

```
    @ApiOperation(value = "get all users", tags = {"user-controller"})
    public ResponseEntity<PageableResponse<UserDto>> getAllUsers(
            @RequestParam(value = "pageNumber", defaultValue = "0", required =
false) int pageNumber,
            @RequestParam(value = "pageSize", defaultValue = "10", required =
false) int pageSize,
            @RequestParam(value = "sortBy", defaultValue = "name", required =
false) String sortBy,
            @RequestParam(value = "sortDir", defaultValue = "asc", required =
false) String sortDir
    ) {
        return      new      ResponseEntity<>(userService.getAllUser(pageNumber,
pageSize, sortBy, sortDir), HttpStatus.OK);
    }

    //get single
    @GetMapping("/{userId}")
    @ApiOperation(value = "Get single user by userid !!")
    public ResponseEntity<UserDto> getUser(@PathVariable String userId) {
        return      new      ResponseEntity<>(userService.getUserById(userId),
HttpStatus.OK);
    }

    //get by email
    @GetMapping("/email/{email}")
    public ResponseEntity<UserDto> getUserByEmail(@PathVariable String email) {
```

```
        return      new      ResponseEntity<>(userService.getUserByEmail(email),
HttpStatus.OK);

    }


    //search user

    @GetMapping("/search/{keywords}")

    public    ResponseEntity<List<UserDto>>    searchUser(@PathVariable    String
keywords) {

        return      new      ResponseEntity<>(userService.searchUser(keywords),
HttpStatus.OK);

    }


    //upload user image

    @PostMapping("/image/{userId}")

    public                                          ResponseEntity<ImageResponse>
uploadUserImage(@RequestParam("userImage") MultipartFile image, @PathVariable
String userId) throws IOException {

        String imageName = fileService.uploadFile(image, imageUploadPath);

        UserDto user = userService.getUserById(userId);

        user.setImageName(imageName);

     UserDto userDto = userService.updateUser(user, userId)       ImageResponse
imageResponse                                                              =
ImageResponse.builder().imageName(imageName).success(true).message("image    is
uploaded successfully ").status(HttpStatus.CREATED).build();

        return new ResponseEntity<>(imageResponse, HttpStatus.CREATED);
```

```
    }

    //serve user image

    @GetMapping(value = "/image/{userId}")

    public void serveUserImage(@PathVariable String userId, HttpServletResponse
response) throws IOException {

        UserDto user = userService.getUserById(userId);

        logger.info("User image name : {} ", user.getImageName());

        InputStream    resource    =    fileService.getResource(imageUploadPath,
user.getImageName());

        response.setContentType(MediaType.IMAGE_JPEG_VALUE);

        StreamUtils.copy(resource, response.getOutputStream());

    }

}
```



user_id	about	user_email	gender	user_image_name	user_name	user_password
07a1d7a1-1857-40b0-a8a2-23de9cf0a130	I am Normal User	durgesh@gmail.com	Male	default.png	durgesh	$2a$10$5ymMdQZphiVeZLFJmoC89e7Llw0j68u...
1c28c19c-3f62-4de2-8ae7-64cc42560457	I am a student intern	pankaj@dev.com	male	pankaj.png	pankaj.beh1234	$2a$10$bKhQgZM75KO17C.UaMsObuNqH0tFo...
3b7a3a34-e178-434f-8fd2-5cdb03908489	A sample user profile	ravi@example.com	male	profile.jpg	Ravi1234d	$2a$10$xoqC0kzeFD7HkPDCqMAM7ueBrMRvN...
4aea0fb7-fd2d-40ad-bb93-203199672f78	A sample user	john.doe@example.com	male	profile.jpg	John Doe	abcde
51b3cdd0-1fee-4c64-91be-def1418ae1c5	A sample user profile	johndoe@devel.com	male	profile.jpg	JohDoe1234d	securePa
5e794b80-c2ef-4ea0-b51f-f2e510054d1d	A sample user profile	johndoe@adityabirla.com	male	JohnDoe.jpg	JohnDoe1234d	securePa
609e3391-7237-4476-bec7-2c074bdba18b	A sample user	john.doe@develop.com	male	profile1.jpg	John_Doe	abcde
7473fe73-49c8-400a-ae46-41e0af9f1ede	I am admin User	admin@gmail.com	Male	default.png	admin	$2a$10$WAgtxGZWkfFOuO2bQ4Va9Oov.3OYN...
95e2b017-e6fc-4cd5-bc59-5f425f3da7c9	A sample user profile	ram@example.com	male	profile.jpg	Ram1234d	securePa
a1e74ade-b82f-4bde-bbf6-c8875d80f143	I am a student intern	kritikad322@gmail.com	xyz	kritikas.png	Kritika123	$2a$2a$2a$10$N6vwuTLeFmav.shSpQ7HnO1m...
a816032c-35f1-448b-9ae7-025b6a477a3c	A sample user	john.doe@eam.com	male	profile1.jpg	John_Doe123	abcde
cc8a858e-be74-4be0-a0d8-43378f54ec5a	A sample user	john.doe@dev.com	male	profile.jpg	John Doe	abcde

2)ProductController

```
package com.lcwd.electronic.store.controllers;


import com.lcwd.electronic.store.dtos.*;

import com.lcwd.electronic.store.services.FileService;

import com.lcwd.electronic.store.services.ProductService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.util.StreamUtils;

import org.springframework.web.bind.annotation.*;

import org.springframework.web.multipart.MultipartFile;


import javax.servlet.http.HttpServletResponse;
```

```
import java.io.IOException;

import java.io.InputStream;


@RestController

@RequestMapping("/products")

public class ProductController {


   @Autowired

   private ProductService productService;


   @Autowired

   private FileService fileService;


   @Value("${product.image.path}")

   private String imagePath;


   //create

   @PreAuthorize("hasRole('ADMIN')")

   @PostMapping

   public  ResponseEntity<ProductDto>  createProduct(@RequestBody  ProductDto
productDto) {

        ProductDto createdProduct = productService.create(productDto);

        return new ResponseEntity<>(createdProduct, HttpStatus.CREATED);

   }
```

```
    //update
    @PreAuthorize("hasRole('ADMIN')")
    @PutMapping("/{productId}")
    public ResponseEntity<ProductDto> updateProduct(@PathVariable String
productId, @RequestBody ProductDto productDto) {
        ProductDto updatedProduct = productService.update(productDto,
productId);
        return new ResponseEntity<>(updatedProduct, HttpStatus.OK);
    }



    //delete
    @PreAuthorize("hasRole('ADMIN')")
    @DeleteMapping("/{productId}")
    public ResponseEntity<ApiResponseMessage> delete(@PathVariable String
productId) {
        productService.delete(productId);
        ApiResponseMessage responseMessage =
ApiResponseMessage.builder().message("Product is deleted successfully
!!").status(HttpStatus.OK).success(true).build();
        return new ResponseEntity<>(responseMessage, HttpStatus.OK);


    }
```

```
    //get single


    @GetMapping("/{productId}")
    public ResponseEntity<ProductDto> getProduct(@PathVariable String productId)
{
        ProductDto productDto = productService.get(productId);
        return new ResponseEntity<>(productDto, HttpStatus.OK);
    }


    //get all
    @GetMapping
    public ResponseEntity<PageableResponse<ProductDto>> getAll(
            @RequestParam(value = "pageNumber", defaultValue = "0", required =
false) int pageNumber,
            @RequestParam(value = "pageSize", defaultValue = "10", required =
false) int pageSize,
            @RequestParam(value = "sortBy", defaultValue = "title", required =
false) String sortBy,
            @RequestParam(value = "sortDir", defaultValue = "asc", required =
false) String sortDir


    ) {
        PageableResponse<ProductDto>              pageableResponse              =
productService.getAll(pageNumber, pageSize, sortBy, sortDir);
        return new ResponseEntity<>(pageableResponse, HttpStatus.OK);
    }
```

```
    //get all live
//    /products/live
    @GetMapping("/live")
    public ResponseEntity<PageableResponse<ProductDto>> getAllLive(
            @RequestParam(value = "pageNumber", defaultValue = "0", required =
false) int pageNumber,
            @RequestParam(value = "pageSize", defaultValue = "10", required =
false) int pageSize,
            @RequestParam(value = "sortBy", defaultValue = "title", required =
false) String sortBy,
            @RequestParam(value = "sortDir", defaultValue = "asc", required =
false) String sortDir


    ) {
        PageableResponse<ProductDto>              pageableResponse              =
productService.getAllLive(pageNumber, pageSize, sortBy, sortDir);
        return new ResponseEntity<>(pageableResponse, HttpStatus.OK);
    }


    //search
    @GetMapping("/search/{query}")
    public ResponseEntity<PageableResponse<ProductDto>> searchProduct(
            @PathVariable String query,
```

```
            @RequestParam(value = "pageNumber", defaultValue = "0", required =
false) int pageNumber,

            @RequestParam(value = "pageSize", defaultValue = "10", required =
false) int pageSize,

            @RequestParam(value = "sortBy", defaultValue = "title", required =
false) String sortBy,

            @RequestParam(value = "sortDir", defaultValue = "asc", required =
false) String sortDir


    ) {

        PageableResponse<ProductDto>              pageableResponse              =
productService.searchByTitle(query, pageNumber, pageSize, sortBy, sortDir);

        return new ResponseEntity<>(pageableResponse, HttpStatus.OK);

    }


    //upload image

    @PreAuthorize("hasRole('ADMIN')")

    @PostMapping("/image/{productId}")

    public ResponseEntity<ImageResponse> uploadProductImage(

            @PathVariable String productId,

            @RequestParam("productImage") MultipartFile image

    ) throws IOException {

        String fileName = fileService.uploadFile(image, imagePath);

        ProductDto productDto = productService.get(productId);

        productDto.setProductImageName(fileName);
```

```
        ProductDto updatedProduct = productService.update(productDto,
productId);

        ImageResponse response =
ImageResponse.builder().imageName(updatedProduct.getProductImageName()).messag
e("Product image is successfully uploaded
!!").status(HttpStatus.CREATED).success(true).build();

        return new ResponseEntity<>(response, HttpStatus.CREATED);


    }



    //serve image


    @GetMapping(value = "/image/{productId}")

    public void serveUserImage(@PathVariable String productId,
HttpServletResponse response) throws IOException {

        ProductDto productDto = productService.get(productId);

        InputStream resource = fileService.getResource(imagePath,
productDto.getProductImageName());

        response.setContentType(MediaType.IMAGE_JPEG_VALUE);

        StreamUtils.copy(resource, response.getOutputStream());


    }


}
```

product_id	added_date	description	discounted_price	live	price	quantity	stock	title
0083827f-43d6-4101-949d-e4c61bc2a7fa	2024-07-02 09:43:37.600000	this tv has very good quality and very good fea...	100000	1	180000	50	1	Lg tv
1e40851e-c844-43e4-8d37-6eb49486711f	2024-06-28 17:07:24.874000	this phone has very good camera	100000	1	120000	50	1	Samsung s22 ul
39753081-ec3d-4a89-80a3-df3fc229b002	2024-06-28 17:02:30.339000	this is very good iphone launched in 2022	85000	1	90000	100	0	Iphone 14 max
728ef0e7-d682-4dfb-9d09-4113d31fb1d5	2024-06-29 06:54:13.338000	this tv has very good quality and very good fea...	100000	1	150000	50	1	Samsung tv
c9fc79d6-d3ab-40ec-ab35-b1436e272b09	2024-06-28 17:05:58.661000	this is very good iphone launched in 2021	80000	1	80000	50	1	Iphone 13 max

POST http://localhost:9090/products Send

Params Auth Headers (9) Body Scripts Tests Settings Cookies

raw JSON Beautify

```
{
    "title":"Iphone 14 max pro",
    "description":"this is very good iphone launched in 2022",
    "price":90000,
    "discountedPrice":85000,
    "quantity":100,
    "live":true,
    "stock":false

```

Body 201 Created 847 ms 430 B Save as example

Pretty Raw Preview Visualize JSON

```
{
    "productId": "39753081-ec3d-4a89-80a3-df3fc229b002",
    "title": "Iphone 14 max pro",
    "description": "this is very good iphone launched in 2022",
    "price": 90000,
    "discountedPrice": 85000,
    "quantity": 100,
    "addedDate": "2024-06-28T11:32:30.339+00:00",
    "live": true,
    "stock": false
}
```

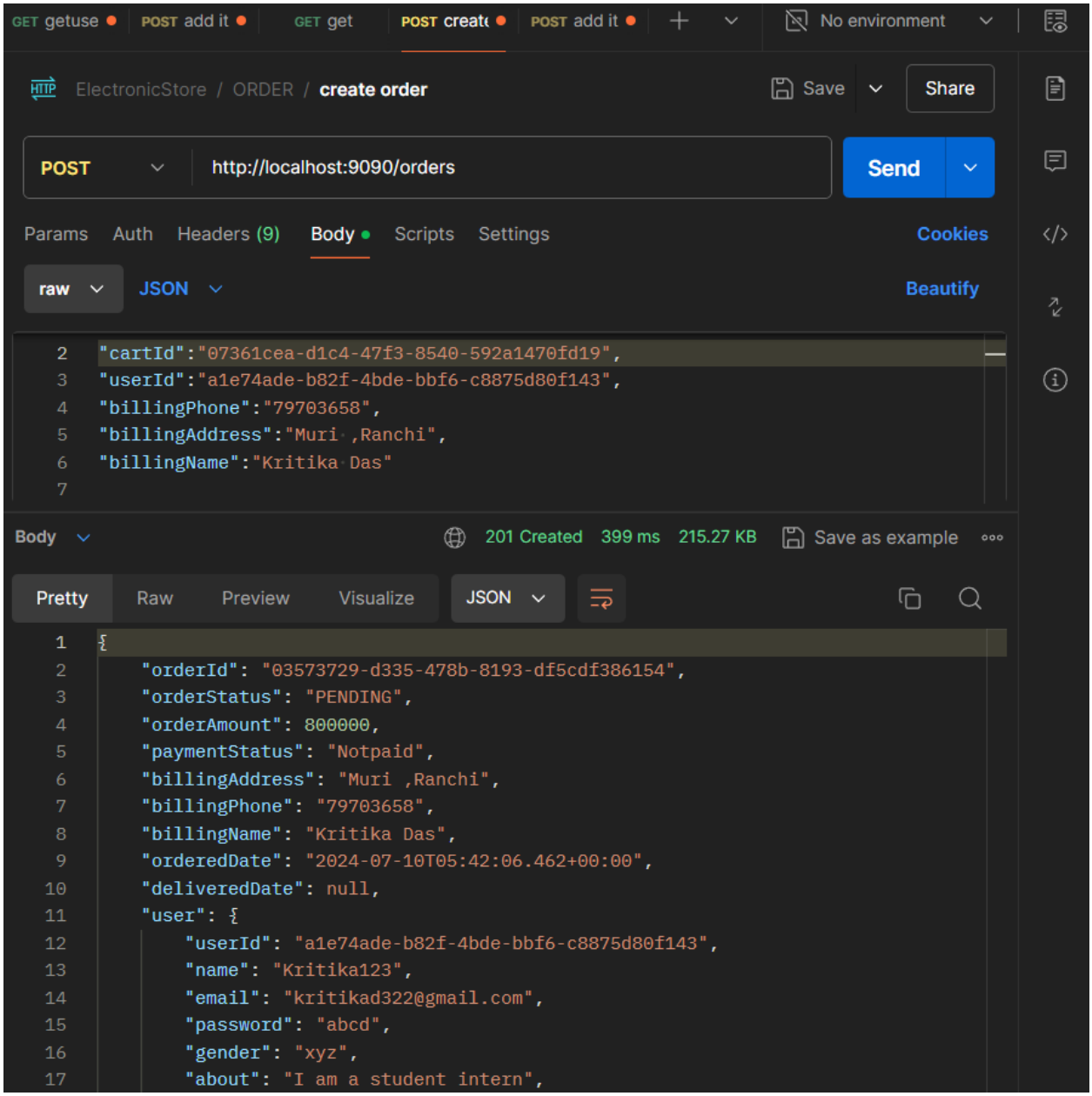
3)OrderController

```
package com.lcwd.electronic.store.controllers;


import com.lcwd.electronic.store.dtos.ApiResponseMessage;

import com.lcwd.electronic.store.dtos.CreateOrderRequest;

import com.lcwd.electronic.store.dtos.OrderDto;

import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.services.OrderService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.*;


import javax.validation.Valid;

import java.util.List;


@RestController

@RequestMapping("/orders")

public class OrderController {
```

```
    @Autowired
    private OrderService orderService;


    //create
    @PostMapping
    public ResponseEntity<OrderDto> createOrder(@Valid @RequestBody
CreateOrderRequest request) {
        OrderDto order = orderService.createOrder(request);
        return new ResponseEntity<>(order, HttpStatus.CREATED);
    }


    @PreAuthorize("hasRole('ADMIN')")
    @DeleteMapping("/{orderId}")
    public ResponseEntity<ApiResponseMessage> removeOrder(@PathVariable String
orderId) {
        orderService.removeOrder(orderId);
        ApiResponseMessage responseMessage = ApiResponseMessage.builder()
                .status(HttpStatus.OK)
                .message("order is removed !!")
                .success(true)
                .build();
        return new ResponseEntity<>(responseMessage, HttpStatus.OK);
```

```
    }


    //get orders of the user


    @GetMapping("/users/{userId}")
    public ResponseEntity<List<OrderDto>> getOrdersOfUser(@PathVariable String
userId) {
        List<OrderDto> ordersOfUser = orderService.getOrdersOfUser(userId);
        return new ResponseEntity<>(ordersOfUser, HttpStatus.OK);
    }


    @GetMapping
    public ResponseEntity<PageableResponse<OrderDto>> getOrders(
            @RequestParam(value = "pageNumber", defaultValue = "0", required =
false) int pageNumber,
            @RequestParam(value = "pageSize", defaultValue = "10", required =
false) int pageSize,
            @RequestParam(value = "sortBy", defaultValue = "orderedDate",
required = false) String sortBy,
            @RequestParam(value = "sortDir", defaultValue = "desc", required =
false) String sortDir
    ) {
        PageableResponse<OrderDto> orders = orderService.getOrders(pageNumber,
pageSize, sortBy, sortDir);
        return new ResponseEntity<>(orders, HttpStatus.OK);
    }
```

}



order_id	billing_address	billing_name	billing_phone	delivered_date	order_amount	order_status	ordered_date	payment_status	user_id
03573729-d335-478b-8193-df5cdf386154	Muri ,Ranchi	Kritika Das	79703658	NULL	800000	PENDING	2024-07-10 11:12:06.462000	Notpaid	a1e74ade-b82f-4bde-bbf6-c8875d80f143
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4)CategoryController

```
package com.lcwd.electronic.store.controllers;


import com.lcwd.electronic.store.dtos.ApiResponseMessage;

import com.lcwd.electronic.store.dtos.CategoryDto;

import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.dtos.ProductDto;

import com.lcwd.electronic.store.services.CategoryService;

import com.lcwd.electronic.store.services.ProductService;

import org.apache.catalina.connector.Response;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.*;


import javax.validation.Valid;

import java.awt.print.Pageable;


@RestController

@RequestMapping("/categories")

public class CategoryController {
```

```
    @Autowired
    private CategoryService categoryService;
    @Autowired
    private ProductService productService;


    //create
    @PreAuthorize("hasRole('ADMIN')")
    @PostMapping
    public ResponseEntity<CategoryDto> createCategory(@Valid @RequestBody
CategoryDto categoryDto) {
        //call service to  save object
        CategoryDto categoryDto1 = categoryService.create(categoryDto);
        return new ResponseEntity<>(categoryDto1, HttpStatus.CREATED);
    }


    //update
    @PreAuthorize("hasRole('ADMIN')")
    @PutMapping("/{categoryId}")
    public ResponseEntity<CategoryDto> updateCategory(
            @PathVariable String categoryId,
            @RequestBody CategoryDto categoryDto
    ) {
        CategoryDto updatedCategory = categoryService.update(categoryDto,
categoryId);
```

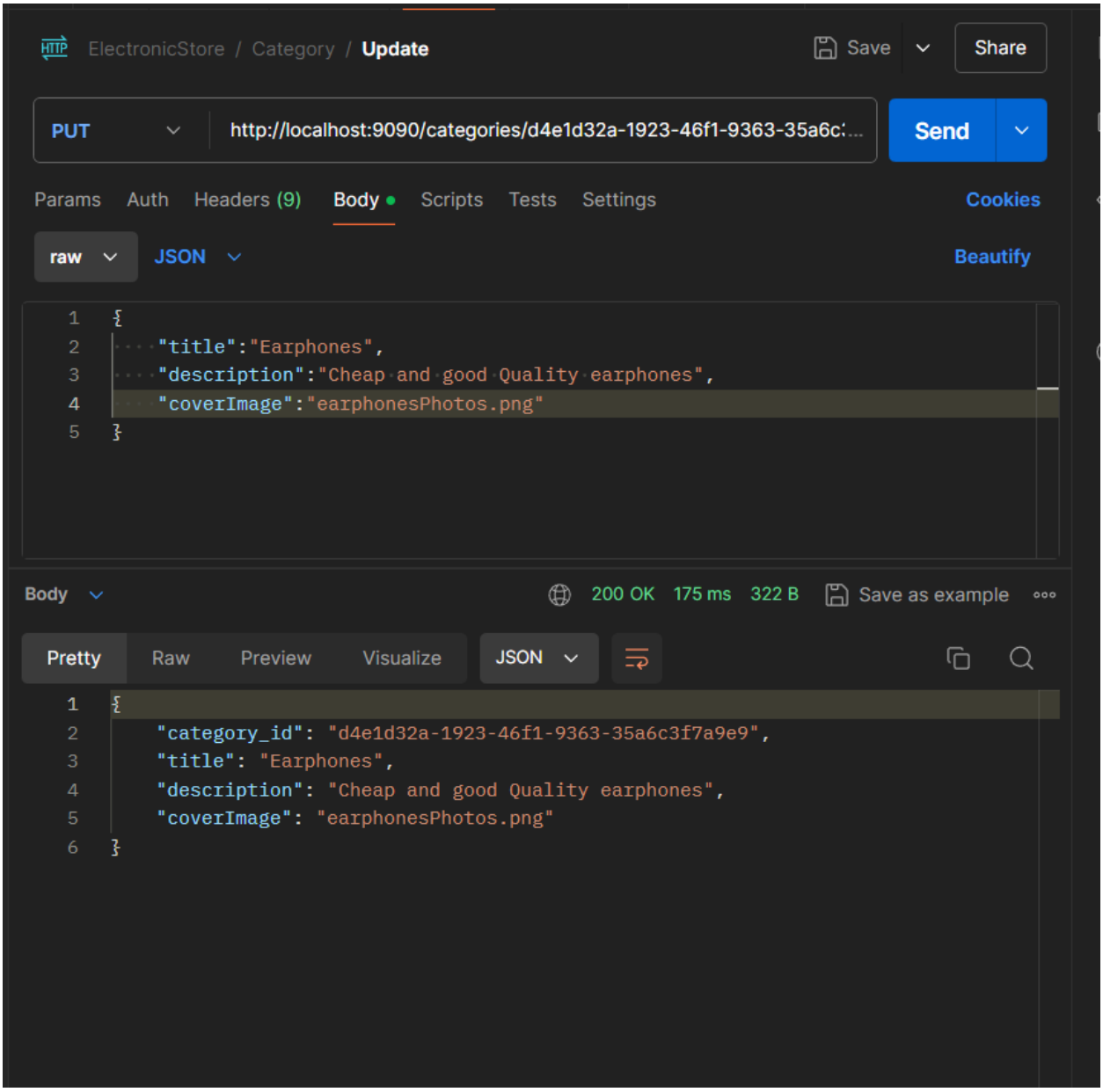
```
        return new ResponseEntity<>(updatedCategory, HttpStatus.OK);

    }


    //delete

    @PreAuthorize("hasRole('ADMIN')")

    @DeleteMapping("/{categoryId}")

    public ResponseEntity<ApiResponseMessage> deleteCategory(

            @PathVariable String categoryId

    ) {

        categoryService.delete(categoryId);

        ApiResponseMessage responseMessage =
ApiResponseMessage.builder().message("Category is deleted successfully
!!").status(HttpStatus.OK).success(true).build();

        return new ResponseEntity<>(responseMessage, HttpStatus.OK);




    }


    //get all


    @GetMapping

    public ResponseEntity<PageableResponse<CategoryDto>> getAll(

            @RequestParam(value = "pageNumber", defaultValue = "0", required =
false) int pageNumber,
```

```
            @RequestParam(value = "pageSize", defaultValue = "10", required =
false) int pageSize,

            @RequestParam(value = "sortBy", defaultValue = "title", required =
false) String sortBy,

            @RequestParam(value = "sortDir", defaultValue = "asc", required =
false) String sortDir




    ) {

        PageableResponse<CategoryDto>              pageableResponse              =
categoryService.getAll(pageNumber, pageSize, sortBy, sortDir);

        return new ResponseEntity<>(pageableResponse, HttpStatus.OK);

    }



    //get single

    @GetMapping("/{categoryId}")

    public     ResponseEntity<CategoryDto>     getSingle(@PathVariable     String
categoryId) {

        CategoryDto categoryDto = categoryService.get(categoryId);

        return ResponseEntity.ok(categoryDto);

    }



    //create product with category

    @PostMapping("/{categoryId}/products")

    public ResponseEntity<ProductDto> createProductWithCategory(
```

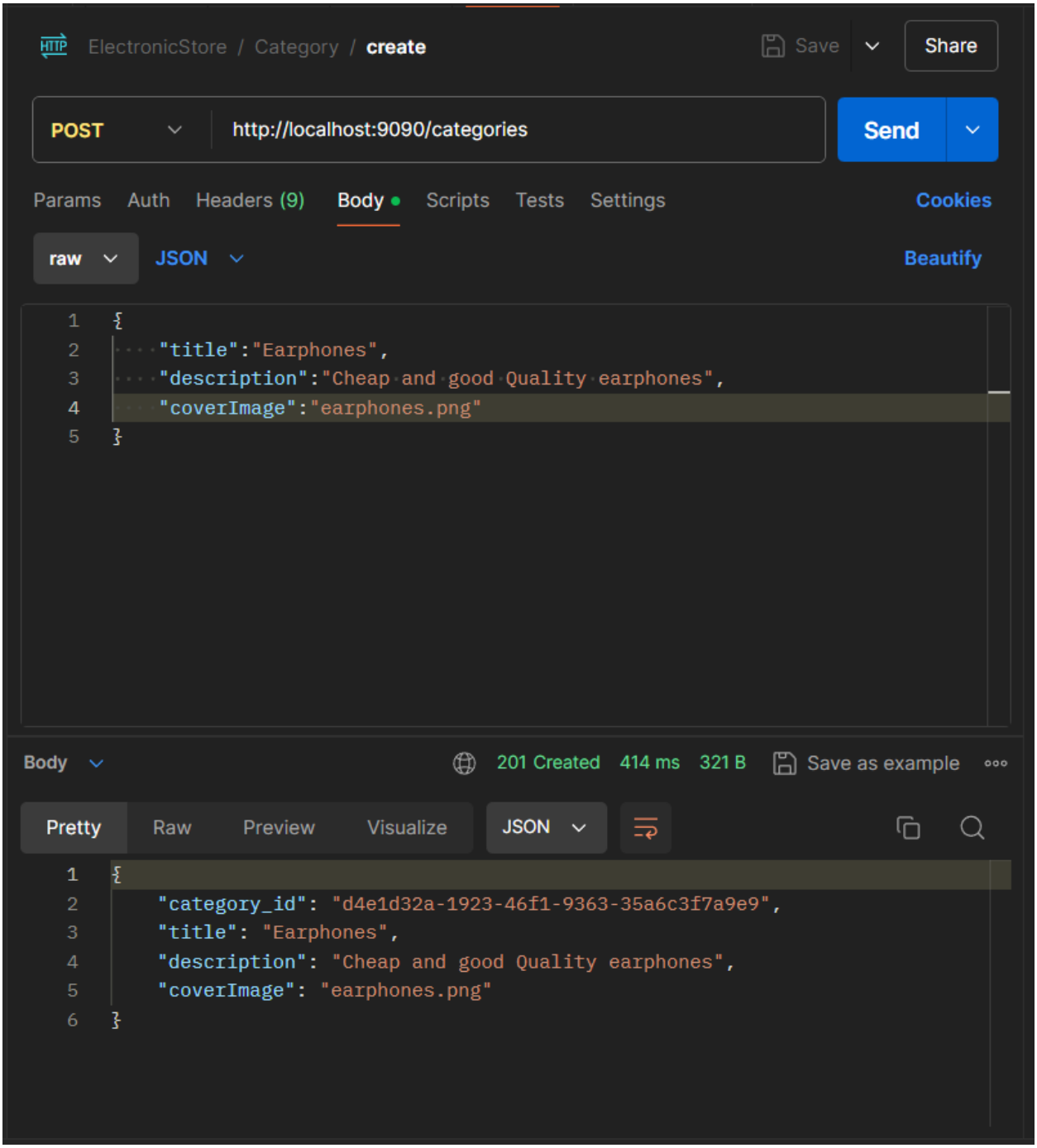
```
            @PathVariable("categoryId") String categoryId,
            @RequestBody ProductDto dto
    ) {
        ProductDto productWithCategory = productService.createWithCategory(dto,
categoryId);
        return new ResponseEntity<>(productWithCategory, HttpStatus.CREATED);
    }

    //update category of product
    @PutMapping("/{categoryId}/products/{productId}")
    public ResponseEntity<ProductDto> updateCategoryOfProduct(
            @PathVariable String categoryId,
            @PathVariable String productId
    ) {
        ProductDto    productDto    =    productService.updateCategory(productId,
categoryId);
        return new ResponseEntity<>(productDto, HttpStatus.OK);
    }

    //get products of categories
    @GetMapping("/{categoryId}/products")
    public ResponseEntity<PageableResponse<ProductDto>> getProductsOfCategory(
            @PathVariable String categoryId,
```

```
            @RequestParam(value = "pageNumber", defaultValue = "0", required =
false) int pageNumber,

            @RequestParam(value = "pageSize", defaultValue = "10", required =
false) int pageSize,

            @RequestParam(value = "sortBy", defaultValue = "title", required =
false) String sortBy,

            @RequestParam(value = "sortDir", defaultValue = "asc", required =
false) String sortDir




    ) {


        PageableResponse<ProductDto>                    response                    =
productService.getAllOfCategory(categoryId,pageNumber,pageSize,sortBy,sortDir)
;

        return new ResponseEntity<>(response, HttpStatus.OK);


    }



}
```

id	cover_image	category_desc	category_title
518eb679-2463-428a-9139-aed48836e5ad	Trending.png	this category contains product related to trending	Trending products
72d11264-e9cc-463e-bbe6-78c02adeca61	Top.png	this category contains product related to top 5 i...	Top 5 products
b7b8f6b7-215f-4214-805a-cb74f94e6fd4	TV.png	this category contains product related to monit...	TV
d4e1d32a-1923-46f1-9363-35a6c3f7a9e9	earphones.png	Cheap and good Quality earphones	Earphones
d83b3a4b-0e37-4597-b7fd-bc3817975945	Mobile.png	this category contains product related to Mobile...	Mobile Phones
NULL	NULL	NULL	NULL



```
{
  "content": [
    {
      "category_id": "d4e1d32a-1923-46f1-9363-35a6c3f7a9e9",
      "title": "Earphones",
      "description": "Cheap and good Quality earphones",
      "coverImage": "earphones.png"
    },
    {
      "category_id": "d83b3a4b-0e37-4597-b7fd-bc3817975945",
      "title": "Mobile Phones",
      "description": "this category contains product related to Mobile phones ",
      "coverImage": "Mobile.png"
    },
    {
      "category_id": "72d11264-e9cc-463e-bbe6-78c02adeca61",
      "title": "Top 5 products",
      "description": "this category contains product related to top 5 in this month",
      "coverImage": "Top.png"
    },
    {
      "category_id": "518eb679-2463-428a-9139-aed48836e5ad",
      "title": "Trending products",
      "description": "this category contains product related to trending",
      "coverImage": "Trending.png"
    },
    {
      "category_id": "b7b8f6b7-215f-4214-805a-cb74f94e6fd4",
      "title": "TV",
      "description": "this category contains product related to monitors and tv",
      "coverImage": "TV.png"
    }
  ],
  "pageNumber": 0,
  "pageSize": 10,
  "totalElements": 5,
  "totalPages": 1,
  "lastPage": true
}
```

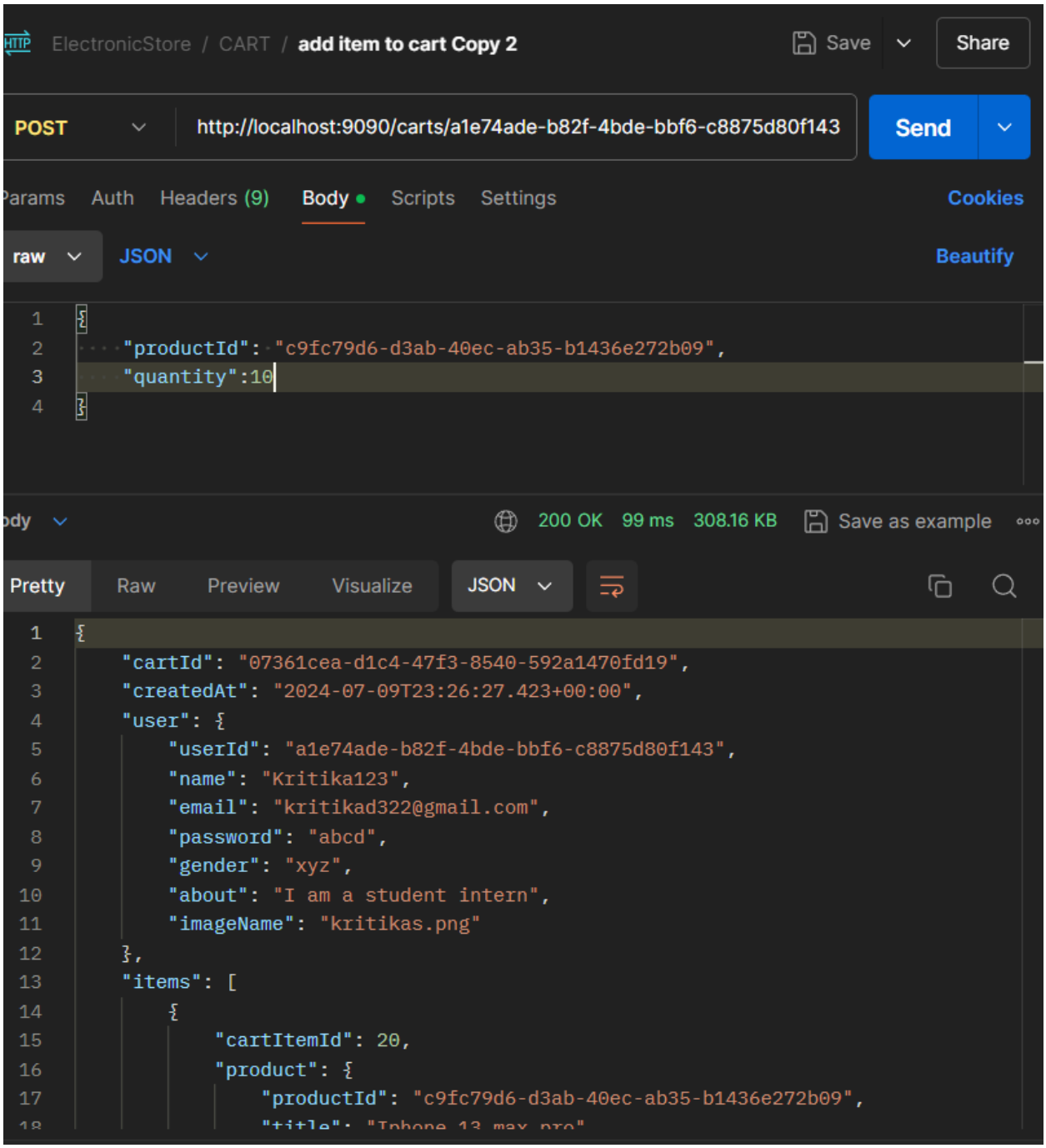


5)CartController

```java
package com.lcwd.electronic.store.controllers;


import com.lcwd.electronic.store.dtos.AddItemToCartRequest;
import com.lcwd.electronic.store.dtos.ApiResponseMessage;
import com.lcwd.electronic.store.dtos.CartDto;
import com.lcwd.electronic.store.services.CartService;
import lombok.Getter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;


import javax.persistence.PersistenceUnit;


@RestController
@RequestMapping("/carts")
public class CartController {


    @Autowired
    private CartService cartService;


    //add items to cart
    @PostMapping("/{userId}")
```

```
    public ResponseEntity<CartDto> addItemToCart(@PathVariable String userId,
@RequestBody AddItemToCartRequest request) {
        CartDto cartDto = cartService.addItemToCart(userId, request);
        return new ResponseEntity<>(cartDto, HttpStatus.OK);
    }

    @DeleteMapping("/{userId}/items/{itemId}")
    public ResponseEntity<ApiResponseMessage> removeItemFromCart(@PathVariable
String userId, @PathVariable int itemId) {
        cartService.removeItemFromCart(userId, itemId);
        ApiResponseMessage response = ApiResponseMessage.builder()
                .message("Item is removed !!")
                .success(true)
                .status(HttpStatus.OK)
                .build();
        return new ResponseEntity<>(response, HttpStatus.OK);

    }

    //clear cart
    @DeleteMapping("/{userId}")
    public ResponseEntity<ApiResponseMessage> clearCart(@PathVariable String
userId) {
        cartService.clearCart(userId);
```

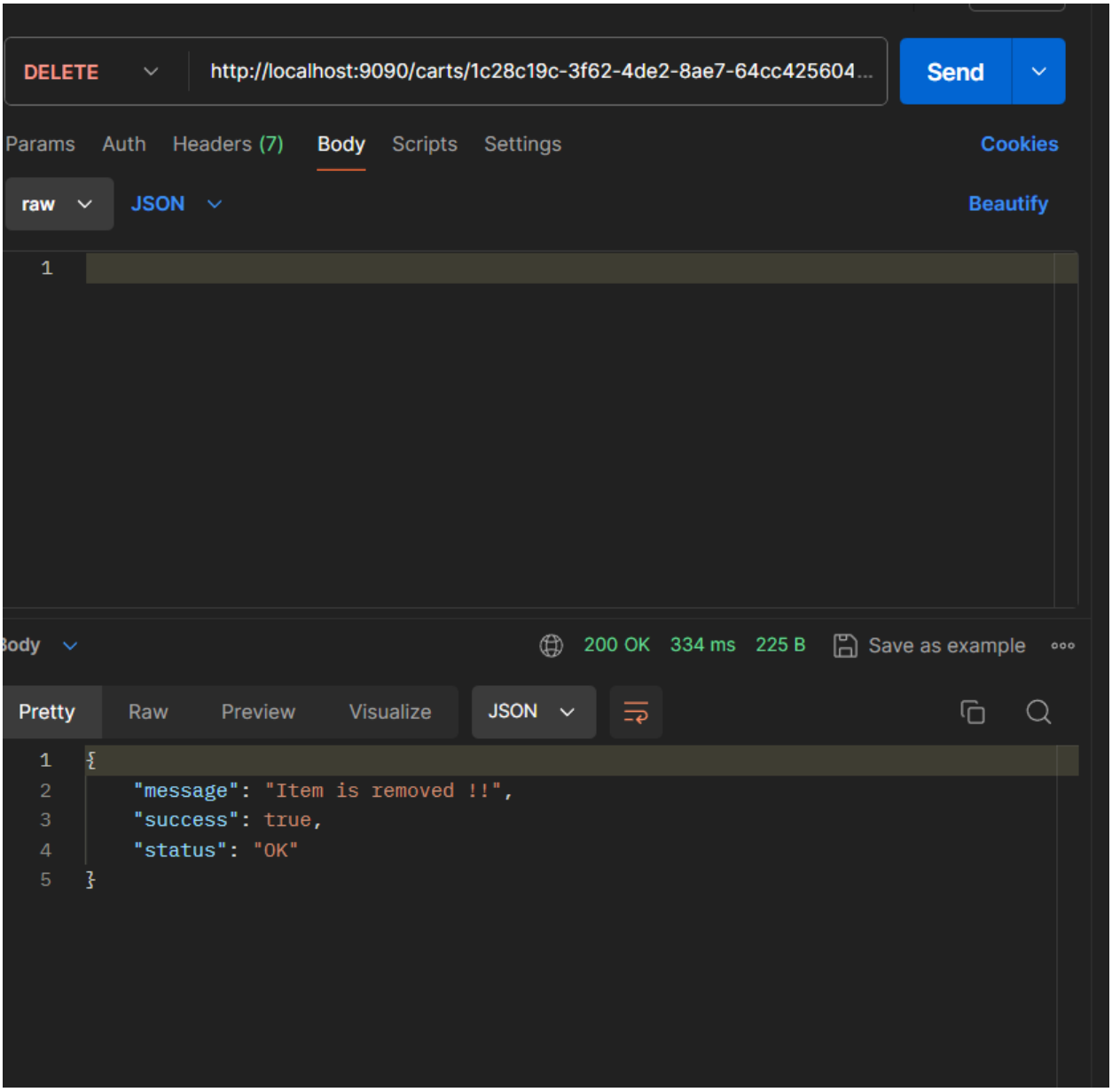
```
        ApiResponseMessage response = ApiResponseMessage.builder()
                .message("Now cart is blank !!")
                .success(true)
                .status(HttpStatus.OK)
                .build();
        return new ResponseEntity<>(response, HttpStatus.OK);

    }

    //add items to cart
    @GetMapping("/{userId}")
    public ResponseEntity<CartDto> getCart(@PathVariable String userId) {
        CartDto cartDto = cartService.getCartByUser(userId);
        return new ResponseEntity<>(cartDto, HttpStatus.OK);
    }

}
```

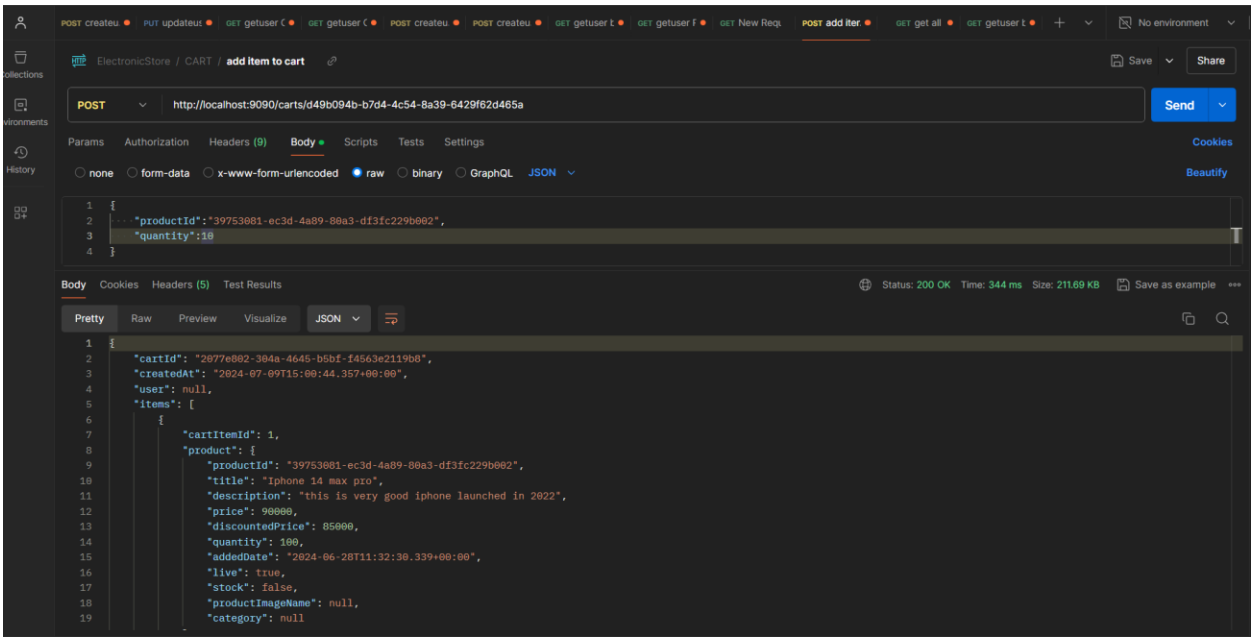

ElectronicStore / CART / add item to cart Copy 2
Save
Share
POST
http://localhost:9090/carts/a1e74ade-b82f-4bde-bbf6-c8875d80f143
Send
Params
Auth
Headers (9)
Body
Scripts
Settings
Cookies
raw
JSON
Beautify
{
"productId": "c9fc79d6-d3ab-40ec-ab35-b1436e272b09",
"quantity":10
}
200 OK
99 ms
308.16 KB
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"cartId": "07361cea-d1c4-47f3-8540-592a1470fd19",
"createdAt": "2024-07-09T23:26:27.423+00:00",
"user": {
"userId": "a1e74ade-b82f-4bde-bbf6-c8875d80f143",
"name": "Kritika123",
"email": "kritikad322@gmail.com",
"password": "abcd",
"gender": "xyz",
"about": "I am a student intern",
"imageName": "kritikas.png"
},
"items": [
{
"cartItemId": 20,
"product": {
"productId": "c9fc79d6-d3ab-40ec-ab35-b1436e272b09",

```
1    SELECT * FROM electronic_store.cart_items;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

cart_item_id	quantity	total_price	cart_id	product_id
1	10	900000	2077e802-304a-4645-b5bf-f4563e2119b8	39753081-ec3d-4a89-80a3-df3fc229b002
10	6	600000	a534cc71-c2cf-4f45-8e5e-d94a652bdfbf	1e40851e-c844-43e4-8d37-6eb49486711f
NULL	NULL	NULL	NULL	NULL


DELETE
http://localhost:9090/carts/1c28c19c-3f62-4de2-8ae7-64cc425604...
Send
Params
Auth
Headers (7)
Body
Scripts
Settings
Cookies
raw
JSON
Beautify
1
200 OK
334 ms
225 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "Item is removed !!",
3 "success": true,
4 "status": "OK"
5 }



6)AuthController

```
package com.lcwd.electronic.store.controllers;


import com.lcwd.electronic.store.dtos.JwtRequest;

import com.lcwd.electronic.store.dtos.JwtResponse;

import com.lcwd.electronic.store.dtos.UserDto;

import com.lcwd.electronic.store.exceptions.BadApiRequestException;

import com.lcwd.electronic.store.security.JwtHelper;

import com.lcwd.electronic.store.services.UserService;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiOperation;

import org.modelmapper.ModelMapper;
```

```
import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import
org.springframework.security.authentication.UsernamePasswordAuthenticationToke
n;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.web.bind.annotation.*;


import java.security.Principal;


@RestController

@RequestMapping("/auth")

@Api(value = "AuthController", description = "APIs for Authentication!!")

public class AuthController {


   @Autowired

   private UserDetailsService userDetailsService;

   @Autowired

   private ModelMapper modelMapper;
```

```
    @Autowired
    private AuthenticationManager manager;

    @Autowired
    private UserService userService;

    @Autowired
    private JwtHelper helper;

    @PostMapping("/login")
    public ResponseEntity<JwtResponse> login(@RequestBody JwtRequest request) {
        this.doAuthenticate(request.getEmail(), request.getPassword());
        UserDetails                          userDetails                          =
userDetailsService.loadUserByUsername(request.getEmail());
        String token = this.helper.generateToken(userDetails);
        UserDto userDto = modelMapper.map(userDetails, UserDto.class);
        JwtResponse response = JwtResponse.builder()
                .jwtToken(token)
                .user(userDto).build();
        return new ResponseEntity<>(response, HttpStatus.OK);
    }
```

```
    private void doAuthenticate(String email, String password) {


        UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(email, password);
        try {
            manager.authenticate(authentication);
        } catch (BadCredentialsException e) {
            throw new BadApiRequestException(" Invalid Username or Password
!!");
        }


    }



    @GetMapping("/current")
    public ResponseEntity<UserDto> getCurrentUser(Principal principal) {
        String name = principal.getName();
        return new
ResponseEntity<>(modelMapper.map(userDetailsService.loadUserByUsername(name),
UserDto.class), HttpStatus.OK);
    }


}
```

7)HomeController

```
package com.lcwd.electronic.store.controllers;


import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


@RestController
@RequestMapping("/test")
public class HomeController {
   @GetMapping
   public String testing() {
       return "Welcome to electronic store";
   }
}
```

B)ENTITIES:

1)User:

```
package com.lcwd.electronic.store.entities;
import lombok.*;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import javax.persistence.*;
```

```
import java.util.*;

import java.util.stream.Collectors;

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Entity

@Table(name = "users")

public class User implements UserDetails {

@Id

    //@GeneratedValue(strategy = GenerationType.IDENTITY)

    private String userId;

@Column(name = "user_name")

    private String name;

    @Column(name = "user_email", unique = true)

    private String email;

    @Column(name = "user_password", length = 500)

    private String password;

    private String gender;

    @Column(length = 1000)

    private String about;

    @Column(name = "user_image_name")
```

```
    private String imageName;

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade =
CascadeType.REMOVE)

    private List<Order> orders = new ArrayList<>();

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)

    private Set<Role> roles = new HashSet<>();

    @OneToOne(mappedBy = "user",cascade = CascadeType.REMOVE)

    private  Cart cart;

    //must have to implement

    //TODO:

    @Override

    public Collection<? extends GrantedAuthority> getAuthorities() {

        Set<SimpleGrantedAuthority> authorities = this.roles.stream().map(role
->                                                                    new
SimpleGrantedAuthority(role.getRoleName())).collect(Collectors.toSet());

        return authorities;

    }

    @Override

    public String getUsername() {

        return this.email;

    }

    @Override

    public String getPassword() {

        return this.password;
```

```
    }
@Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Overrid
public boolean isEnabled() {
        return true;
    }
}
```

2)Cart:

```
package com.lcwd.electronic.store.entities;
import lombok.*;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
```

```
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;

import java.util.*;

import java.util.stream.Collectors;

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Entity

@Table(name = "users")

public class User implements UserDetails {

    @Id

    //@GeneratedValue(strategy = GenerationType.IDENTITY)

    private String userId;

    @Column(name = "user_name")

    private String name;

    @Column(name = "user_email", unique = true)

    private String email;

    @Column(name = "user_password", length = 500)

    private String password;

    private String gender;

    @Column(length = 1000)
```

```
    private String about;


    @Column(name = "user_image_name")
    private String imageName;


    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade =
CascadeType.REMOVE)
    private List<Order> orders = new ArrayList<>();


    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private Set<Role> roles = new HashSet<>();
    @OneToOne(mappedBy = "user",cascade = CascadeType.REMOVE)
    private  Cart cart;
    //must have to implement
    //TODO:
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Set<SimpleGrantedAuthority> authorities = this.roles.stream().map(role
->                                                                    new
SimpleGrantedAuthority(role.getRoleName())).collect(Collectors.toSet());
        return authorities;
    }@Override
    public String getUsername() {
        return this.email;
```

```
    }
@Override
    public String getPassword() {
        return this.password;
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

3)CartItem

```
package com.lcwd.electronic.store.entities;

import lombok.*;

import org.springframework.beans.factory.annotation.Autowired;

import javax.persistence.*;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Builder

@Entity

@Table(name = "cart_items")

public class CartItem {

 @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private int cartItemId;

    @OneToOne

    @JoinColumn(name = "product_id")

    private Product product;

    private  int quantity;

    private  int totalPrice;

//    mapping cart

    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name = "cart_id") private  Cart cart;}
```

4)Category

```
package com.lcwd.electronic.store.entities;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import javax.persistence.*;

import java.util.ArrayList;

import java.util.List;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name = "categories")

public class Category {

    @Id

    @Column(name = "id")

    private String categoryId;


    @Column(name = "category_title", length = 60, nullable = false)

    private String title;
```

```
    @Column(name = "category_desc", length = 500)
    private String description;
    private String coverImage;
    // other attributes if you have...
    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Product> products = new ArrayList<>();
}
```

5)Order

```
package com.lcwd.electronic.store.entities;

import lombok.*;

import javax.persistence.*;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;


@AllArgsConstructor

@NoArgsConstructor

@Getter

@Setter

@Builder

@Entity

@Table(name = "orders")

public class Order {
```

```
    @Id
    private String orderId;
    //PENDING,DISPATCHED,DELIVERED,
    //enum
    private String orderStatus;
    //NOT-PAID, PAID
    //enum
    //boolean- false=>NOTPAID  || true=>PAID
    private String paymentStatus;
    private int orderAmount;

    @Column(length = 1000)
    private String billingAddress;
    private String billingPhone;
    private String billingName;
    private Date orderedDate;
    private Date deliveredDate;
    //user
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id")
    private User user;

    @OneToMany(mappedBy = "order", fetch = FetchType.EAGER, cascade =
CascadeType.ALL)
```

```
    private List<OrderItem> orderItems = new ArrayList<>();

}
```

6)orderItem

```
package com.lcwd.electronic.store.entities;

import lombok.*;

import javax.persistence.*;


@AllArgsConstructor

@NoArgsConstructor

@Getter

@Setter

@Entity

@Builder

@Table(name = "order_items")

public class OrderItem {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private  int orderItemId;

    private  int quantity;

    private  int totalPrice;

    @OneToOne

    @JoinColumn(name = "product_id")

    private  Product product;
```

```
    @ManyToOne
    @JoinColumn(name = "order_id")
    private  Order order;
}
```

7)Product

```
package com.lcwd.electronic.store.entities;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
import java.util.Date;
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "products")
public class Product {
    @Id
    private String productId;
    private String title;
```

```
    @Column(length = 10000)
    private String description;
    private int price;
    private int discountedPrice;
    private int quantity;
    private Date addedDate;
    private boolean live;
    private boolean stock;
    private String productImageName;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "category_id")
    private  Category category;



}
```

8)Role

```
package com.lcwd.electronic.store.entities;


import lombok.*;


import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
```

```
@Getter

@Setter

@Builder

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name = "roles")

public class Role {



   @Id

   private String roleId;


   private String roleName;}
```

C)CONFIG

1)ProjectConfig

```
package com.lcwd.electronic.store.config;


import org.modelmapper.ModelMapper;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class ProjectConfig {


   @Bean
   public ModelMapper mapper(){
       return new ModelMapper();
   }


}
```

2)SecurityConfig

```
package com.lcwd.electronic.store.config;


import com.lcwd.electronic.store.security.JwtAuthenticationEntryPoint;
import com.lcwd.electronic.store.security.JwtAuthenticationFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
```

```
import
org.springframework.security.config.annotation.authentication.builders.Authent
icationManagerBuilder;

import
org.springframework.security.config.annotation.authentication.configuration.Au
thenticationConfiguration;

import
org.springframework.security.config.annotation.method.configuration.EnableGlob
alMethodSecurity;

import
org.springframework.security.config.annotation.web.builders.HttpSecurity;

import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

import
org.springframework.security.web.authentication.UsernamePasswordAuthentication
Filter;


@Configuration

@EnableGlobalMethodSecurity(prePostEnabled = true)
```

```
public class SecurityConfig {


   @Autowired
   private UserDetailsService userDetailsService;
   @Autowired
   private JwtAuthenticationEntryPoint authenticationEntryPoint;
   @Autowired
   private JwtAuthenticationFilter authenticationFilter;


   private final String[] PUBLIC_URLS = {


           "/swagger-ui/**",
           "/webjars/**",
           "/swagger-resources/**",
           "/v3/api-docs",
           "/v2/api-docs"
   };
@Bean
  public  SecurityFilterChain  securityFilterChain(HttpSecurity  http)  throws
Exception {


       http.csrf()
               .disable()
               .cors()
```

```
                .disable()
                .authorizeRequests()
                .antMatchers("/auth/login")
                .permitAll()
                .antMatchers(HttpMethod.POST, "/users")
                .permitAll()
                .antMatchers(HttpMethod.DELETE, "/users/**").hasRole("ADMIN")
                .antMatchers(PUBLIC_URLS)
                .permitAll()
                .anyRequest()
                .authenticated()
                .and()
                .exceptionHandling()
                .authenticationEntryPoint(authenticationEntryPoint)
                .and()
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);


        http.addFilterBefore(authenticationFilter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
```

```
    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider daoAuthenticationProvider = new
DaoAuthenticationProvider();

daoAuthenticationProvider.setUserDetailsService(this.userDetailsService);
        daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return daoAuthenticationProvider;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration builder) throws Exception {
        return builder.getAuthenticationManager(); }}
```

3)SwaggerConfig

```
package com.lcwd.electronic.store.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.*;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spi.service.contexts.SecurityContext;
import springfox.documentation.spring.web.plugins.Docket;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

@Configuration
public class SwaggerConfig {

   @Bean
   public Docket docket() {
       return new Docket(DocumentationType.SWAGGER_2)
               .apiInfo(getApiInfo())

.securityContexts(Collections.singletonList(getSecurityContext()))
               .securitySchemes(Collections.singletonList(getSchemes()))
               .select()
               .apis(RequestHandlerSelectors.any())
               .paths(PathSelectors.any())
```

```
                .build();
    }

    private SecurityContext getSecurityContext() {
        return SecurityContext.builder()
                .securityReferences(getSecurityReferences())
                .build();
    }

    private List<SecurityReference> getSecurityReferences() {
        AuthorizationScope[] scopes = {new AuthorizationScope("Global", "Access
Everything")};
        return Collections.singletonList(new SecurityReference("JWT", scopes));
    }

    private ApiKey getSchemes() {
        return new ApiKey("JWT", "Authorization", "header");
    }

    private ApiInfo getApiInfo() {
        return new ApiInfo(
                "Electronic Store Backend : APIS",
                "Made by KRITIKA DAS",
                "1.0.0V",
```

```
                "Terms of service",
                new Contact("Kritika Das", "www.linkedin.com/in/kritika-das-3040b2246", "kritika@example.com"),
                "License of API",
                "API license URL",
                Collections.emptyList()
        );
    }
}
```

D)DTOS

1)AddItemToCartRequest

```
package com.lcwd.electronic.store.dtos;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@AllArgsConstructor
@NoArgsConstructor
@Getter
```

```
@Setter
public class AddItemToCartRequest {

   private  String productId;

   private  int quantity;

}
```

2)ApiResponseMessage

```
package com.lcwd.electronic.store.dtos;



import lombok.*;
import org.springframework.http.HttpStatus;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ApiResponseMessage {
   private String message;
   private boolean success;
```

```
    private HttpStatus status;


}
```

3)CartDto

```
package com.lcwd.electronic.store.dtos;


import lombok.*;


import java.util.ArrayList;
import java.util.Date;
import java.util.List;


@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class CartDto {


    private String cartId;
    private Date createdAt;
    private UserDto user;
    private List<CartItemDto> items = new ArrayList<>();
```

```
}
```

4)CartItemDto

```
package com.lcwd.electronic.store.dtos;


import com.lcwd.electronic.store.entities.Cart;

import com.lcwd.electronic.store.entities.CartItem;

import com.lcwd.electronic.store.entities.Product;

import com.lcwd.electronic.store.entities.User;

import lombok.*;


import javax.persistence.*;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;


@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class CartItemDto {


   private int cartItemId;
```

```
    private ProductDto product;

    private int quantity;

    private int totalPrice;


}
```

4)CategoryDto

```
package com.lcwd.electronic.store.dtos;


import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;


import javax.persistence.Column;

import javax.persistence.Id;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.Size;


@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor
```

```
public class CategoryDto {



    private String categoryId;


    @NotBlank(message = "title is required !!")
    @Size(min = 4, message = "title must be of minimum 4 characters")
    private String title;



    @NotBlank(message = "Description required !!")
    private String description;


    private String coverImage;


}
```

5)CreateOrderRequest

```
package com.lcwd.electronic.store.dtos;


import lombok.*;


import javax.validation.constraints.NotBlank;
import java.util.ArrayList;
```

```
import java.util.Date;
import java.util.List;


@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
@ToString
public class CreateOrderRequest {


    @NotBlank(message = "Cart id is required !!")
    private String cartId;


    @NotBlank(message = "Cart id is required !!")
    private String userId;



    private String orderStatus = "PENDING";
    private String paymentStatus = "NOTPAID";
    @NotBlank(message = "Address is required !!")
    private String billingAddress;
    @NotBlank(message = "Phone number is required !!")
```

```
    private String billingPhone;

    @NotBlank(message = "Billing name  is required !!")

    private String billingName;




}
```

6)ImageResponse

```
package com.lcwd.electronic.store.dtos;


import lombok.*;

import org.springframework.http.HttpStatus;


@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class ImageResponse {

    private String imageName;

    private String message;

    private boolean success;

    private HttpStatus status;

}
```

7)JwtRequest

```
package com.lcwd.electronic.store.dtos;



import io.swagger.annotations.ApiModelProperty;

import lombok.*;


@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Builder

@ToString

public class JwtRequest {


   private String email;

   private String password;


}
```

8)JwtResponse

```
package com.lcwd.electronic.store.dtos;


import lombok.*;
```

```
@Setter

@Getter

@AllArgsConstructor

@NoArgsConstructor

@Builder

@ToString

public class JwtResponse {

   private String jwtToken;

   private UserDto user;

}
```

9)OrderDto

```
package com.lcwd.electronic.store.dtos;


import com.lcwd.electronic.store.entities.OrderItem;

import com.lcwd.electronic.store.entities.User;

import lombok.*;


import javax.persistence.*;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;
```

```
@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Builder

@ToString

public class OrderDto {


   private String orderId;

   private String orderStatus="PENDING";

   private String paymentStatus="NOTPAID";

   private int orderAmount;

   private String billingAddress;

   private String billingPhone;

   private String billingName;

   private Date orderedDate=new Date();

   private Date deliveredDate;

   //private UserDto user;

   private List<OrderItemDto> orderItems = new ArrayList<>();



}
```

10)OrderItemDto

```java
package com.lcwd.electronic.store.dtos;


import com.lcwd.electronic.store.entities.Order;
import com.lcwd.electronic.store.entities.Product;
import lombok.*;


import javax.persistence.*;


@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class OrderItemDto {



    private int orderItemId;


    private int quantity;


    private int totalPrice;


    private ProductDto product;
```

```
}
```

11)PageableResponse

```
package com.lcwd.electronic.store.dtos;


import lombok.*;


import java.util.List;


@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class PageableResponse<T> {


   private List<T> content;

   private int pageNumber;

   private int pageSize;

   private long totalElements;

   private int totalPages;

   private boolean lastPage;
```

```
}
```

12)ProductDto

```
package com.lcwd.electronic.store.dtos;


import com.fasterxml.jackson.annotation.JsonProperty;

import com.lcwd.electronic.store.entities.Category;

import lombok.*;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;


import javax.persistence.Column;

import javax.persistence.Id;

import javax.servlet.http.HttpServletRequest;

import java.util.Date;


@AllArgsConstructor

@NoArgsConstructor

@Getter

@Setter

@ToString

public class ProductDto {

   private String productId;
```

```
    private String title;

    private String description;

    private int price;

    private int discountedPrice;

    private int quantity;

    private Date addedDate;

    private boolean live;

    private boolean stock;

    private String productImageName;

    private CategoryDto category;

}
```

13)RoleDto

```
package com.lcwd.electronic.store.dtos;

import lombok.*;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class RoleDto {

    private String roleId;

    private String roleName;

}
```

14)UserDto

```
package com.lcwd.electronic.store.dtos;

import com.lcwd.electronic.store.entities.Role;

import com.lcwd.electronic.store.validate.ImageNameValid;

import io.swagger.annotations.ApiModelProperty;

import lombok.*;

import javax.persistence.Column;

import javax.persistence.Id;

import javax.validation.constraints.Email;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.Pattern;

import javax.validation.constraints.Size;

import java.util.HashSet;

import java.util.Set;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class UserDto {

   private String userId;

   @Size(min = 3, max = 20, message = "Invalid Name !!")

   @ApiModelProperty(value = "user_name", name = "username", required = true,
notes = "user name of new user !!")
```

```
    private String name;

    @Pattern(regexp    =    "^[a-z0-9][-a-z0-9._]+@([-a-z0-9]+\\.)+[a-z]{2,5}$",
message = "Invalid User Email !!")

    @NotBlank(message = "Email is required !!")

    private String email;

    @NotBlank(message = "Password is required !!")

    private String password;

    @Size(min = 4, max = 6, message = "Invalid gender !!")

    private String gender;

    @NotBlank(message = "Write something about yourself !!")

    private String about;

    @ImageNameValid

    private String imageName;


    private Set<RoleDto> roles = new HashSet<>();

}
```

E)EXCEPTIONS

1)BadApiRequestException

```
package com.lcwd.electronic.store.exceptions;


public class BadApiRequestException extends RuntimeException {


    public BadApiRequestException(String message) {
```

```
        super(message);
    }
    public BadApiRequestException() {
        super("Bad Request !!");}}
```

2)GlobalExceptionsHandler

```
package com.lcwd.electronic.store.exceptions;

import com.lcwd.electronic.store.dtos.ApiResponseMessage;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.FieldError;

import org.springframework.validation.ObjectError;

import org.springframework.web.bind.MethodArgumentNotValidException;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.HashMap;

import java.util.List;

import java.util.Map;


@RestControllerAdvice

public class GlobalExceptionHandler {
```

```
    private Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class);

    @ExceptionHandler(ResourceNotFoundException.class)

    public ResponseEntity<ApiResponseMessage> resourceNotFoundExceptionHandler(ResourceNotFoundException ex) {

        logger.info("Exception Handler invoked !!");

        ApiResponseMessage response = ApiResponseMessage.builder().message(ex.getMessage()).status(HttpStatus.NOT_FOUND).success(true).build();

        return new ResponseEntity(response, HttpStatus.NOT_FOUND);

    }


    @ExceptionHandler(MethodArgumentNotValidException.class)

    public ResponseEntity<Map<String, Object>> handleMethodArgumentNotValidException(MethodArgumentNotValidException ex) {

        List<ObjectError> allErrors = ex.getBindingResult().getAllErrors();

        Map<String, Object> response = new HashMap<>();

        allErrors.stream().forEach(objectError -> {

            String message = objectError.getDefaultMessage();

        String field = ((FieldError) objectError).getField();

            response.put(field, message);

        });


        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
```

```
    }



    @ExceptionHandler(BadApiRequestException.class)

    public                                    ResponseEntity<ApiResponseMessage>
handleBadApiRequest(BadApiRequestException ex) {

        logger.info("Bad api request");

        ApiResponseMessage                          response                          =
ApiResponseMessage.builder().message(ex.getMessage()).status(HttpStatus.BAD_RE
QUEST).success(false).build();

        return new ResponseEntity(response, HttpStatus.BAD_REQUEST);



    }


}
```

3)RequestNotFoundException

```
package com.lcwd.electronic.store.exceptions;

import lombok.Builder;


@Builder

public class ResourceNotFoundException extends  RuntimeException{


    public ResourceNotFoundException(){

        super("Resource not found !!");
```

```
    }

    public ResourceNotFoundException(String message){

        super(message);}}
```

F)HELPER

```
package com.lcwd.electronic.store.helper;

import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.dtos.UserDto;

import com.lcwd.electronic.store.entities.User;

import org.modelmapper.ModelMapper;

import org.springframework.data.domain.Page;

import java.util.List;

import java.util.stream.Collectors;

public class Helper {

    public static <U, V> PageableResponse<V> getPageableResponse(Page<U> page,
Class<V> type) {

        List<U> entity = page.getContent();

        List<V>     dtoList     =     entity.stream().map(object     ->     new
ModelMapper().map(object, type)).collect(Collectors.toList());



        PageableResponse<V> response = new PageableResponse<>();

        response.setContent(dtoList);

        response.setPageNumber(page.getNumber());

        response.setPageSize(page.getSize());

        response.setTotalElements(page.getTotalElements());
```

```
        response.setTotalPages(page.getTotalPages());
        response.setLastPage(page.isLast());
        return response;}}
```

G)REPOSITORIES

1)CartItemRepository

```
package com.lcwd.electronic.store.repositories;
import com.lcwd.electronic.store.entities.CartItem;
import org.springframework.data.jpa.repository.JpaRepository;
public interface CartItemRepository extends JpaRepository<CartItem,Integer> {
}
```

2)CartRepository

```
package com.lcwd.electronic.store.repositories;
import com.lcwd.electronic.store.entities.Cart;
import com.lcwd.electronic.store.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;
import javax.swing.text.html.Option;
import java.util.Optional;
public interface CartRepository extends JpaRepository<Cart, String> {
   Optional<Cart> findByUser(User user);}
```

3)CategoryRepository

```
package com.lcwd.electronic.store.repositories;
import com.lcwd.electronic.store.entities.Category;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface CategoryRepository extends JpaRepository<Category, String>

{}
```

4)OrderItemRepository

```
package com.lcwd.electronic.store.repositories;

import com.lcwd.electronic.store.entities.OrderItem;

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

public interface OrderItemRepository extends JpaRepository<OrderItem,
Integer>{}
```

5)OrderRepository

```
package com.lcwd.electronic.store.repositories;

import com.lcwd.electronic.store.entities.Order;

import com.lcwd.electronic.store.entities.User;

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

import java.util.List;

public interface OrderRepository extends JpaRepository<Order, String> {

   List<Order> findByUser(User user);}
```

6)ProductRepository

```
package com.lcwd.electronic.store.repositories;


import com.lcwd.electronic.store.entities.Category;

import com.lcwd.electronic.store.entities.Product;

import org.springframework.data.domain.Page;
```

```
import org.springframework.data.domain.Pageable;

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

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, String> {

   //search

   Page<Product> findByTitleContaining(String subTitle, Pageable pageable);

   Page<Product> findByLiveTrue(Pageable pageable);

   Page<Product> findByCategory(Category category,Pageable pageable);}
```

7)RoleRepository

```
package com.lcwd.electronic.store.repositories;

import com.lcwd.electronic.store.entities.Role;

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

public interface RoleRepository extends JpaRepository<Role,String> {

}
```

8)UserRepository

```
package com.lcwd.electronic.store.repositories;

import com.lcwd.electronic.store.entities.User;

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

import java.util.List;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, String>

{

   Optional<User> findByEmail(String email);
```

```
    Optional<User> findByEmailAndPassword(String email,String password);

    List<User> findByNameContaining(String keywords);}
```

H)SECURITY

1)JwtAuthenticationEntryPoint

```
package com.lcwd.electronic.store.security;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

import javax.servlet.ServletException;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.io.PrintWriter;

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

    @Override

    public void commence(HttpServletRequest request, HttpServletResponse
response, AuthenticationException authException) throws IOException,
ServletException {

        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);

        PrintWriter writer = response.getWriter();

        writer.println("Access Denied !! " + authException.getMessage());

    }
```

```
}
```

2)JwtAuthenticationFilter

```
package com.lcwd.electronic.store.security;

import io.jsonwebtoken.ExpiredJwtException;

import io.jsonwebtoken.MalformedJwtException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

Import
org.springframework.security.web.authentication.UsernamePasswordAuthentication
Filter;

import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource
;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;


import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

@Component

public class JwtAuthenticationFilter extends OncePerRequestFilter {

   private Logger logger = LoggerFactory.getLogger(OncePerRequestFilter.class);

   @Autowired

   private JwtHelper jwtHelper;

   @Autowired

   private UserDetailsService userDetailsService;

   @Override

   protected      void      doFilterInternal(HttpServletRequest      request,
HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {

        String requestHeader = request.getHeader("Authorization");

              logger.info(" Header :  {}", requestHeader);

        String username = null;

        String token = null;

        if (requestHeader != null && requestHeader.startsWith("Bearer")) {

            //looking good

            token = requestHeader.substring(7);

            try {


                username = this.jwtHelper.getUsernameFromToken(token);
```

```
            } catch (IllegalArgumentException e) {
                logger.info("Illegal Argument while fetching the username !!");
                e.printStackTrace();
            } catch (ExpiredJwtException e) {
                logger.info("Given jwt token is expired !!");
                e.printStackTrace();
            } catch (MalformedJwtException e) {
                logger.info("Some changed has done in token !! Invalid Token");
                e.printStackTrace();
            } catch (Exception e) {
                e.printStackTrace();

            }


        } else {
            logger.info("Invalid Header Value !! ");
        }


        //
        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
```

```
            UserDetails                        userDetails                        =
this.userDetailsService.loadUserByUsername(username);

            Boolean    validateToken    =    this.jwtHelper.validateToken(token,
userDetails);

            if (validateToken) {

                UsernamePasswordAuthenticationToken    authentication    =    new
UsernamePasswordAuthenticationToken(userDetails,                          null,
userDetails.getAuthorities());

                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));


SecurityContextHolder.getContext().setAuthentication(authentication);

            } else {

                logger.info("Validation fails !!");

            }

        }

        filterChain.doFilter(request, response);

    }

}
```

3)JwtHelper

```
package com.lcwd.electronic.store.security;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;
```

```java
import java.util.function.Function;


import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;


import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;


import org.springframework.stereotype.Component;


@Component
public class JwtHelper {


    //requirement :
    public static final long JWT_TOKEN_VALIDITY = 5 * 60 * 60;



    @Value("${jwt.secret}")
    private String secret;
```

```
    //retrieve username from jwt token
    public String getUsernameFromToken(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }

    //retrieve expiration date from jwt token
    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

    public <T> T getClaimFromToken(String token, Function<Claims, T>
claimsResolver) {
        final Claims claims = getAllClaimsFromToken(token);
        return claimsResolver.apply(claims);
    }
    private Claims getAllClaimsFromToken(String token) { return
Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
    }
    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }

    //generate token for use
```

```
    public String generateToken(UserDetails userDetails) {

        Map<String, Object> claims = new HashMap<>();

        return doGenerateToken(claims, userDetails.getUsername());

    }

    private String doGenerateToken(Map<String, Object> claims, String subject)
{

        return
Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new
Date(System.currentTimeMillis()))

                .setExpiration(new       Date(System.currentTimeMillis()       +
JWT_TOKEN_VALIDITY * 1000))

                .signWith(SignatureAlgorithm.HS512, secret).compact();

    }

    public Boolean validateToken(String token, UserDetails userDetails) {

        final String username = getUsernameFromToken(token);

        return          (username.equals(userDetails.getUsername())          &&
!isTokenExpired(token));

}}
```

I)VALIDATE

1)ImageNameValid

```
package com.lcwd.electronic.store.validate;

import javax.validation.Constraint;

import javax.validation.Payload;

import java.lang.annotation.*;

@Target({ElementType.FIELD, ElementType.PARAMETER})
```

```
@Retention(RetentionPolicy.RUNTIME)

@Documented

@Constraint(validatedBy = ImageNameValidator.class)

public @interface ImageNameValid {

   //error message

   String message() default "Invalid Image Name !!";

   Class<?>[] groups() default {};

   Class<? extends Payload>[] payload() default {};



}
```

2)ImageNameValidator

```
package com.lcwd.electronic.store.validate;


import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import javax.validation.ConstraintValidator;

import javax.validation.ConstraintValidatorContext;


public class ImageNameValidator implements ConstraintValidator<ImageNameValid,
String> {

   private Logger logger = LoggerFactory.getLogger(ImageNameValidator.class);

   @Override

   public boolean isValid(String value, ConstraintValidatorContext context) {
```

```
        logger.info("Message from isValid : {} ", value);
        //logic
        if (value.isBlank()) {
            return false;
        } else {
            return true;}}}
```

J)SERVICES

1)CartService

```
package com.lcwd.electronic.store.services;


import com.lcwd.electronic.store.dtos.AddItemToCartRequest;
import com.lcwd.electronic.store.dtos.CartDto;


public interface CartService {


   //add items to cart:
   //case1: cart for user is not available: we will create the cart and then
add the item
   //case2: cart available add the items to cart
   CartDto addItemToCart(String userId, AddItemToCartRequest request);


   //remove item from cart:
```

```
    void removeItemFromCart(String userId,int cartItem);

    //remove all items from cart
    void clearCart(String userId);

    CartDto getCartByUser(String userId);
}
```

2)CategoryService

```
package com.lcwd.electronic.store.services;

import com.lcwd.electronic.store.dtos.CategoryDto;
import com.lcwd.electronic.store.dtos.PageableResponse;

public interface CategoryService
{

    //create
    CategoryDto create(CategoryDto categoryDto);

    //update
    CategoryDto update(CategoryDto categoryDto, String categoryId);

    //delete
```

```
    void delete(String categoryId);

    //get all
    PageableResponse<CategoryDto> getAll(int pageNumber, int pageSize, String sortBy, String sortDir);

    //get single category detail
    CategoryDto get(String categoryId);

    //search:
}
```

3)FileService

```
package com.lcwd.electronic.store.services;

import org.springframework.web.multipart.MultipartFile;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

public interface FileService {
```

```
    String uploadFile(MultipartFile file, String path) throws IOException;


    InputStream getResource(String path,String name) throws
FileNotFoundException;


}
```

4)OrderService

```
package com.lcwd.electronic.store.services;


import com.lcwd.electronic.store.dtos.CreateOrderRequest;

import com.lcwd.electronic.store.dtos.OrderDto;

import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.entities.Order;

import org.springframework.data.domain.Pageable;


import java.util.List;


public interface OrderService {


    //create order

    OrderDto createOrder(CreateOrderRequest orderDto);


    //remove order

    void removeOrder(String orderId);
```

```
    //get orders of user
    List<OrderDto> getOrdersOfUser(String userId);

    //get orders
    PageableResponse<OrderDto> getOrders(int pageNumber, int pageSize, String sortBy, String sortDir);

    //order methods(logic) related to order

}
```

5)ProductService

```
package com.lcwd.electronic.store.services;

import com.lcwd.electronic.store.dtos.PageableResponse;
import com.lcwd.electronic.store.dtos.ProductDto;
import com.lcwd.electronic.store.entities.Product;
import org.springframework.data.domain.Pageable;

import java.util.List;

public interface ProductService {
```

```
    //create
    ProductDto create(ProductDto productDto);

    //update
    ProductDto update(ProductDto productDto, String productId);

    //delete
    void delete(String productId);

    //get single

    ProductDto get(String productId);

    //get all
    PageableResponse<ProductDto> getAll(int pageNumber, int pageSize, String sortBy, String sortDir);

    //get all : live
    PageableResponse<ProductDto> getAllLive(int pageNumber, int pageSize, String sortBy, String sortDir);

    //search product
```

```
    PageableResponse<ProductDto> searchByTitle(String subTitle, int pageNumber,
int pageSize, String sortBy, String sortDir);



    //create product with category

    ProductDto createWithCategory(ProductDto productDto,String categoryId);



    //update category of product

    ProductDto updateCategory(String productId,String categoryId);


    PageableResponse<ProductDto>     getAllOfCategory(String     categoryId,int
pageNumber,int pageSize,String sortBy, String sortDir);


    //other methods



}
```

6)UserService

```
package com.lcwd.electronic.store.services;


import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.dtos.UserDto;
```

```
import com.lcwd.electronic.store.entities.User;


import java.util.List;


public interface UserService {


    //create

    UserDto createUser(UserDto userDto);




    //update

    UserDto updateUser(UserDto userDto, String userId);


    //delete


    void deleteUser(String userId);




    //get all users

    PageableResponse<UserDto> getAllUser(int pageNumber, int pageSize, String
sortBy, String sortDir);


    //get single user by id

    UserDto getUserById(String userId);
```

```
    //get  single user by email
    UserDto getUserByEmail(String email);

    //search user
    List<UserDto> searchUser(String keyword);

    //other user specific features

}
```

J.1)Impl

1)CartServiceImpl

```
package com.lcwd.electronic.store.services.impl;

import com.lcwd.electronic.store.dtos.AddItemToCartRequest;
import com.lcwd.electronic.store.dtos.CartDto;
import com.lcwd.electronic.store.entities.Cart;
import com.lcwd.electronic.store.entities.CartItem;
import com.lcwd.electronic.store.entities.Product;
import com.lcwd.electronic.store.entities.User;
import com.lcwd.electronic.store.exceptions.BadApiRequestException;
import com.lcwd.electronic.store.exceptions.ResourceNotFoundException;
import com.lcwd.electronic.store.repositories.CartItemRepository;
```

```java
import com.lcwd.electronic.store.repositories.CartRepository;
import com.lcwd.electronic.store.repositories.ProductRepository;
import com.lcwd.electronic.store.repositories.UserRepository;
import com.lcwd.electronic.store.services.CartService;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.UUID;
import java.util.concurrent.atomic.AtomicReference;
import java.util.stream.Collectors;

@Service
public class CartServiceImpl implements CartService {

    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private UserRepository userRepository;
```

```
    @Autowired
    private CartRepository cartRepository;

    @Autowired
    private ModelMapper mapper;

    @Autowired
    private CartItemRepository cartItemRepository;

    @Override
    public CartDto addItemToCart(String userId, AddItemToCartRequest request) {

        int quantity = request.getQuantity();
        String productId = request.getProductId();

        if (quantity <= 0) {
            throw new BadApiRequestException("Requested quantity is not valid
!!");
        }

        //fetch the product
        Product product = productRepository.findById(productId).orElseThrow(()
-> new ResourceNotFoundException("Product not found in database !!"));
```

```
        //fetch the user from db
        User user = userRepository.findById(userId).orElseThrow(() -> new
ResourceNotFoundException("user not found in database!!"));


        Cart cart = null;
        try {
            cart = cartRepository.findByUser(user).get();
        } catch (NoSuchElementException e) {
            cart = new Cart();
            cart.setCartId(UUID.randomUUID().toString());
            cart.setCreatedAt(new Date());
        }


        //perform cart operations
        //if cart items already present; then update
        AtomicReference<Boolean> updated = new AtomicReference<>(false);
        List<CartItem> items = cart.getItems();
        items = items.stream().map(item -> {


            if (item.getProduct().getProductId().equals(productId)) {
                //item already present in cart
                item.setQuantity(quantity);
                item.setTotalPrice(quantity * product.getDiscountedPrice());
                updated.set(true);
```

```
            }

            return item;

        }).collect(Collectors.toList());


//          cart.setItems(updatedItems);


        //create items
        if (!updated.get()) {
            CartItem cartItem = CartItem.builder()
                    .quantity(quantity)
                    .totalPrice(quantity * product.getDiscountedPrice())
                    .cart(cart)
                    .product(product)
                    .build();
            cart.getItems().add(cartItem);
        }



        cart.setUser(user);
        Cart updatedCart = cartRepository.save(cart);
        return mapper.map(updatedCart, CartDto.class);
```

```
    }


    @Override
    public void removeItemFromCart(String userId, int cartItem) {
        //conditions

        CartItem cartItem1 = cartItemRepository.findById(cartItem).orElseThrow(() -> new ResourceNotFoundException("Cart Item not found !!"));
        cartItemRepository.delete(cartItem1);

    }


    @Override
    public void clearCart(String userId) {
        //fetch the user from db
        User user = userRepository.findById(userId).orElseThrow(() -> new ResourceNotFoundException("user not found in database!!"));
        Cart cart = cartRepository.findByUser(user).orElseThrow(() -> new ResourceNotFoundException("Cart of given user not found !!"));
        cart.getItems().clear();
        cartRepository.save(cart);
    }


    @Override
```

```
    public CartDto getCartByUser(String userId) {

        User user = userRepository.findById(userId).orElseThrow(() -> new
ResourceNotFoundException("user not found in database!!"));

        Cart cart = cartRepository.findByUser(user).orElseThrow(() -> new
ResourceNotFoundException("Cart of given user not found !!"));

        return mapper.map(cart, CartDto.class);

    }

}
```

2)CategoryServiceImpl

```
package com.lcwd.electronic.store.services.impl;


import com.lcwd.electronic.store.dtos.CategoryDto;

import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.entities.Category;

import com.lcwd.electronic.store.exceptions.ResourceNotFoundException;

import com.lcwd.electronic.store.helper.Helper;

import com.lcwd.electronic.store.repositories.CategoryRepository;

import com.lcwd.electronic.store.services.CategoryService;

import org.modelmapper.ModelMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;
```

```java
import org.springframework.stereotype.Service;

import org.springframework.ui.ModelMap;


import java.util.UUID;


@Service

public class CategoryServiceImpl implements CategoryService {


    @Autowired

    private CategoryRepository categoryRepository;


    @Autowired

    private ModelMapper mapper;


    @Override

    public CategoryDto create(CategoryDto categoryDto) {

        //creating categoryId:randomly


        String categoryId = UUID.randomUUID().toString();

        categoryDto.setCategoryId(categoryId);

        Category category = mapper.map(categoryDto, Category.class);
```

```
        Category savedCategory = categoryRepository.save(category);

        return mapper.map(savedCategory, CategoryDto.class);

    }


    @Override

    public CategoryDto update(CategoryDto categoryDto, String categoryId) {


        //get category of given id

        Category                              category                              =
categoryRepository.findById(categoryId).orElseThrow(()          ->          new
ResourceNotFoundException("Category not found with given id !!"));

        //update category details

        category.setTitle(categoryDto.getTitle());

        category.setDescription(categoryDto.getDescription());

        category.setCoverImage(categoryDto.getCoverImage());

        Category updatedCategory = categoryRepository.save(category);

        return mapper.map(updatedCategory, CategoryDto.class);

    }


    @Override

    public void delete(String categoryId) {

        //get category of given id

        Category                              category                              =
categoryRepository.findById(categoryId).orElseThrow(()          ->          new
ResourceNotFoundException("Category not found with given id !!"));
```

```
        categoryRepository.delete(category);

    }


    @Override

    public PageableResponse<CategoryDto> getAll(int pageNumber, int pageSize,
String sortBy, String sortDir) {

        Sort sort = (sortDir.equalsIgnoreCase("desc")) ?
(Sort.by(sortBy).descending()) : (Sort.by(sortBy).ascending());

        Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);

        Page<Category> page = categoryRepository.findAll(pageable);

        PageableResponse<CategoryDto> pageableResponse =
Helper.getPageableResponse(page, CategoryDto.class);

        return pageableResponse;

    }


    @Override

    public CategoryDto get(String categoryId) {

        Category category =
categoryRepository.findById(categoryId).orElseThrow(() -> new
ResourceNotFoundException("Category not found with given id !!"));

        return mapper.map(category, CategoryDto.class);

    }

}
```

3)CustomUserDetailService

```
package com.lcwd.electronic.store.services.impl;
```

```
import com.lcwd.electronic.store.entities.User;

import com.lcwd.electronic.store.repositories.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import
org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;


@Service

public class CustomUserDetailService implements UserDetailsService {


   @Autowired

   private UserRepository userRepository;


   @Override

   public     UserDetails     loadUserByUsername(String     username)     throws
UsernameNotFoundException {

       User user = userRepository.findByEmail(username).orElseThrow(() -> new
UsernameNotFoundException("User with given email not found !!"));

       return user;

   }

}
```

4)FileServiceImpl

```
package com.lcwd.electronic.store.services.impl;



import com.lcwd.electronic.store.exceptions.BadApiRequestException;

import com.lcwd.electronic.store.services.FileService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;


import java.io.*;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.UUID;


@Service

public class FileServiceImpl implements FileService {


    private Logger logger = LoggerFactory.getLogger(FileServiceImpl.class);


    @Override

    public String uploadFile(MultipartFile file, String path) throws IOException
{
```

```
        //abc.png
        String originalFilename = file.getOriginalFilename();
        logger.info("Filename : {}", originalFilename);
        String filename = UUID.randomUUID().toString();
        String extension = originalFilename.substring(originalFilename.lastIndexOf("."));
        String fileNameWithExtension = filename + extension;
        String fullPathWithFileName = path  + fileNameWithExtension;

        logger.info("full image path: {} ", fullPathWithFileName);
        if (extension.equalsIgnoreCase(".png") || extension.equalsIgnoreCase(".jpg") || extension.equalsIgnoreCase(".jpeg")) {

            //file save
            logger.info("file extension is {} ", extension);
            File folder = new File(path);
            if (!folder.exists()) {
                //create the folder
                folder.mkdirs();

            }

            //upload
```

```
            Files.copy(file.getInputStream(), Paths.get(fullPathWithFileName));

            return fileNameWithExtension;


        } else {

            throw new BadApiRequestException("File with this " + extension + "
not allowed !!");

        }



    }


    @Override

    public InputStream getResource(String path, String name) throws
FileNotFoundException {

        String fullPath = path + File.separator + name;

        InputStream inputStream = new FileInputStream(fullPath);

        return inputStream;

    }



}
```

5)OrderServiceImpl

```
package com.lcwd.electronic.store.services.impl;
```

```
import com.lcwd.electronic.store.dtos.CreateOrderRequest;
import com.lcwd.electronic.store.dtos.OrderDto;
import com.lcwd.electronic.store.dtos.PageableResponse;
import com.lcwd.electronic.store.entities.*;
import com.lcwd.electronic.store.exceptions.BadApiRequestException;
import com.lcwd.electronic.store.exceptions.ResourceNotFoundException;
import com.lcwd.electronic.store.helper.Helper;
import com.lcwd.electronic.store.repositories.CartRepository;
import com.lcwd.electronic.store.repositories.OrderRepository;
import com.lcwd.electronic.store.repositories.UserRepository;
import com.lcwd.electronic.store.services.OrderService;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;


import java.util.Date;
import java.util.List;
import java.util.UUID;
import java.util.concurrent.atomic.AtomicReference;
```

```
import java.util.stream.Collectors;


@Service
public class OrderServiceImpl implements OrderService {


    @Autowired
    private UserRepository userRepository;


    @Autowired
    private OrderRepository orderRepository;


    @Autowired
    private ModelMapper modelMapper;


    @Autowired
    private CartRepository cartRepository;




    @Override
    public OrderDto createOrder(CreateOrderRequest orderDto) {


        String userId = orderDto.getUserId();

        String cartId = orderDto.getCartId();
```

```
        //fetch user

        User user = userRepository.findById(userId).orElseThrow(() -> new
ResourceNotFoundException("User not found with given id !!"));


        //fetch cart

        Cart cart = cartRepository.findById(cartId).orElseThrow(() -> new
ResourceNotFoundException("Cart with given id not found on server !!"));


        List<CartItem> cartItems = cart.getItems();


        if (cartItems.size() <= 0) {

            throw new BadApiRequestException("Invalid number of items in cart
!!");


        }


        //other checks


        Order order = Order.builder()

                .billingName(orderDto.getBillingName())

                .billingPhone(orderDto.getBillingPhone())

                .billingAddress(orderDto.getBillingAddress())

                .orderedDate(new Date())

                .deliveredDate(null)
```

```
                .paymentStatus(orderDto.getPaymentStatus())
                .orderStatus(orderDto.getOrderStatus())
                .orderId(UUID.randomUUID().toString())
                .user(user)
                .build();

//        orderItems,amount

        AtomicReference<Integer> orderAmount = new AtomicReference<>(0);
        List<OrderItem> orderItems = cartItems.stream().map(cartItem -> {
//            CartItem->OrderItem
            OrderItem orderItem = OrderItem.builder()
                    .quantity(cartItem.getQuantity())
                    .product(cartItem.getProduct())
                    .totalPrice(cartItem.getQuantity()                    *
cartItem.getProduct().getDiscountedPrice())
                    .order(order)
                    .build();

            orderAmount.set(orderAmount.get() + orderItem.getTotalPrice());
            return orderItem;
        }).collect(Collectors.toList());

        order.setOrderItems(orderItems);
```

```
        order.setOrderAmount(orderAmount.get());


        //
        cart.getItems().clear();
        cartRepository.save(cart);
        Order savedOrder = orderRepository.save(order);
        return modelMapper.map(savedOrder, OrderDto.class);
    }


    @Override
    public void removeOrder(String orderId) {


        Order order = orderRepository.findById(orderId).orElseThrow(() -> new
ResourceNotFoundException("order is not found !!"));
        orderRepository.delete(order);


    }


    @Override
    public List<OrderDto> getOrdersOfUser(String userId) {
        User user = userRepository.findById(userId).orElseThrow(() -> new
ResourceNotFoundException("User not found !!"));
        List<Order> orders = orderRepository.findByUser(user);
```

```
        List<OrderDto>     orderDtos     =     orders.stream().map(order     ->
modelMapper.map(order, OrderDto.class)).collect(Collectors.toList());

        return orderDtos;

    }



    @Override

    public PageableResponse<OrderDto> getOrders(int pageNumber, int pageSize,
String sortBy, String sortDir) {

        Sort       sort       =       (sortDir.equalsIgnoreCase("desc"))       ?
(Sort.by(sortBy).descending()) : (Sort.by(sortBy).ascending());

        Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);

        Page<Order> page = orderRepository.findAll(pageable);

        return Helper.getPageableResponse(page, OrderDto.class);

    }

}
```

6)ProductServiceImpl

```
package com.lcwd.electronic.store.services.impl;


import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.dtos.ProductDto;

import com.lcwd.electronic.store.entities.Category;

import com.lcwd.electronic.store.entities.Product;

import com.lcwd.electronic.store.exceptions.ResourceNotFoundException;
```

```java
import com.lcwd.electronic.store.helper.Helper;
import com.lcwd.electronic.store.repositories.CategoryRepository;
import com.lcwd.electronic.store.repositories.ProductRepository;
import com.lcwd.electronic.store.services.ProductService;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.List;
import java.util.UUID;

@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    private ProductRepository productRepository;
    @Autowired
    private ModelMapper mapper;
```

```
@Autowired
private CategoryRepository categoryRepository;


//other dependency
@Override
public ProductDto create(ProductDto productDto) {



    Product product = mapper.map(productDto, Product.class);


    //product id
    String productId = UUID.randomUUID().toString();
    product.setProductId(productId);
    //added
    product.setAddedDate(new Date());
    Product saveProduct = productRepository.save(product);
    return mapper.map(saveProduct, ProductDto.class);
}


@Override
public ProductDto update(ProductDto productDto, String productId) {
```

```
        //fetch the product of given id

        Product product = productRepository.findById(productId).orElseThrow(()
-> new ResourceNotFoundException("Product not found of given Id !!"));

        product.setTitle(productDto.getTitle());

        product.setDescription(productDto.getDescription());

        product.setPrice(productDto.getPrice());

        product.setDiscountedPrice(productDto.getDiscountedPrice());

        product.setQuantity(productDto.getQuantity());

        product.setLive(productDto.isLive());

        product.setStock(productDto.isStock());

        product.setProductImageName(productDto.getProductImageName());


//        save the entity

        Product updatedProduct = productRepository.save(product);

        return mapper.map(updatedProduct, ProductDto.class);

    }


    @Override

    public void delete(String productId) {

        Product product = productRepository.findById(productId).orElseThrow(()
-> new ResourceNotFoundException("Product not found of given Id !!"));

        productRepository.delete(product);

    }
```

```
    @Override
    public ProductDto get(String productId) {
        Product product = productRepository.findById(productId).orElseThrow(()
-> new ResourceNotFoundException("Product not found of given Id !!"));
        return mapper.map(product, ProductDto.class);
    }

    @Override
    public PageableResponse<ProductDto> getAll(int pageNumber, int pageSize,
String sortBy, String sortDir) {
        Sort sort = (sortDir.equalsIgnoreCase("desc")) ?
(Sort.by(sortBy).descending()) : (Sort.by(sortBy).ascending());
        Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);
        Page<Product> page = productRepository.findAll(pageable);
        return Helper.getPageableResponse(page, ProductDto.class);
    }

    @Override
    public PageableResponse<ProductDto> getAllLive(int pageNumber, int pageSize,
String sortBy, String sortDir) {
        Sort sort = (sortDir.equalsIgnoreCase("desc")) ?
(Sort.by(sortBy).descending()) : (Sort.by(sortBy).ascending());
        Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);
        Page<Product> page = productRepository.findByLiveTrue(pageable);
        return Helper.getPageableResponse(page, ProductDto.class);
```

```
    }


    @Override
    public PageableResponse<ProductDto> searchByTitle(String subTitle, int
pageNumber, int pageSize, String sortBy, String sortDir) {
        Sort sort = (sortDir.equalsIgnoreCase("desc")) ?
(Sort.by(sortBy).descending()) : (Sort.by(sortBy).ascending());
        Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);
        Page<Product> page = productRepository.findByTitleContaining(subTitle,
pageable);
        return Helper.getPageableResponse(page, ProductDto.class);
    }


    @Override
    public ProductDto createWithCategory(ProductDto productDto, String
categoryId) {
        //fetch the category from db:
        Category category =
categoryRepository.findById(categoryId).orElseThrow(() -> new
ResourceNotFoundException("Category not found !!"));
        Product product = mapper.map(productDto, Product.class);


        //product id
        String productId = UUID.randomUUID().toString();
        product.setProductId(productId);
        //added
```

```
        product.setAddedDate(new Date());

        product.setCategory(category);

        Product saveProduct = productRepository.save(product);

        return mapper.map(saveProduct, ProductDto.class);




    }


    @Override

    public ProductDto updateCategory(String productId, String categoryId) {

        //product fetch

        Product product = productRepository.findById(productId).orElseThrow(()
-> new ResourceNotFoundException("Product of given id not found !!"));

        Category category =
categoryRepository.findById(categoryId).orElseThrow(() -> new
ResourceNotFoundException("Category of given id not found !!"));

        product.setCategory(category);

        Product savedProduct = productRepository.save(product);

        return mapper.map(savedProduct, ProductDto.class);

    }


    @Override

    public PageableResponse<ProductDto> getAllOfCategory(String categoryId, int
pageNumber, int pageSize, String sortBy, String sortDir) {
```

```
        Category                                category                                =
categoryRepository.findById(categoryId).orElseThrow(()          ->          new
ResourceNotFoundException("Category of given id not found !!"));

        Sort       sort       =       (sortDir.equalsIgnoreCase("desc"))       ?
(Sort.by(sortBy).descending()) : (Sort.by(sortBy).ascending());

        Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);

        Page<Product>    page    =    productRepository.findByCategory(category,
pageable);

        return Helper.getPageableResponse(page, ProductDto.class);

    }

}
```

7)UserServiceImpl

```
package com.lcwd.electronic.store.services.impl;


import com.lcwd.electronic.store.dtos.ApiResponseMessage;

import com.lcwd.electronic.store.dtos.PageableResponse;

import com.lcwd.electronic.store.dtos.UserDto;

import com.lcwd.electronic.store.entities.Role;

import com.lcwd.electronic.store.entities.User;

import com.lcwd.electronic.store.exceptions.ResourceNotFoundException;

import com.lcwd.electronic.store.helper.Helper;

import com.lcwd.electronic.store.repositories.RoleRepository;

import com.lcwd.electronic.store.repositories.UserRepository;

import com.lcwd.electronic.store.services.UserService;

import org.modelmapper.ModelMapper;
```

```
import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;


import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.NoSuchFileException;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.List;

import java.util.UUID;

import java.util.stream.Collectors;


@Service

public class UserServiceImpl implements UserService {
```

```
@Autowired
private UserRepository userRepository;


@Autowired
private ModelMapper mapper;


@Value("${user.profile.image.path}")
private String imagePath;
@Autowired
private PasswordEncoder passwordEncoder;


@Value("${normal.role.id}")
private String normalRoleId;


@Autowired
private RoleRepository roleRepository;


private Logger logger = LoggerFactory.getLogger(UserServiceImpl.class);




@Override
public UserDto createUser(UserDto userDto) {
```

```
        //generate unique id in string format
        String userId = UUID.randomUUID().toString();
        userDto.setUserId(userId);
        //encoding password
        userDto.setPassword(passwordEncoder.encode(userDto.getPassword()));
        // dto->entity
        User user = dtoToEntity(userDto);



        //fetch role of normal and set it to user
        Role role = roleRepository.findById(normalRoleId).get();
        user.getRoles().add(role);
        User savedUser = userRepository.save(user);
        //entity -> dto
        UserDto newDto = entityToDto(savedUser);
        return newDto;
    }



    @Override
    public UserDto updateUser(UserDto userDto, String userId) {


        User user = userRepository.findById(userId).orElseThrow(() -> new
ResourceNotFoundException("User not found with given id !!"));
```

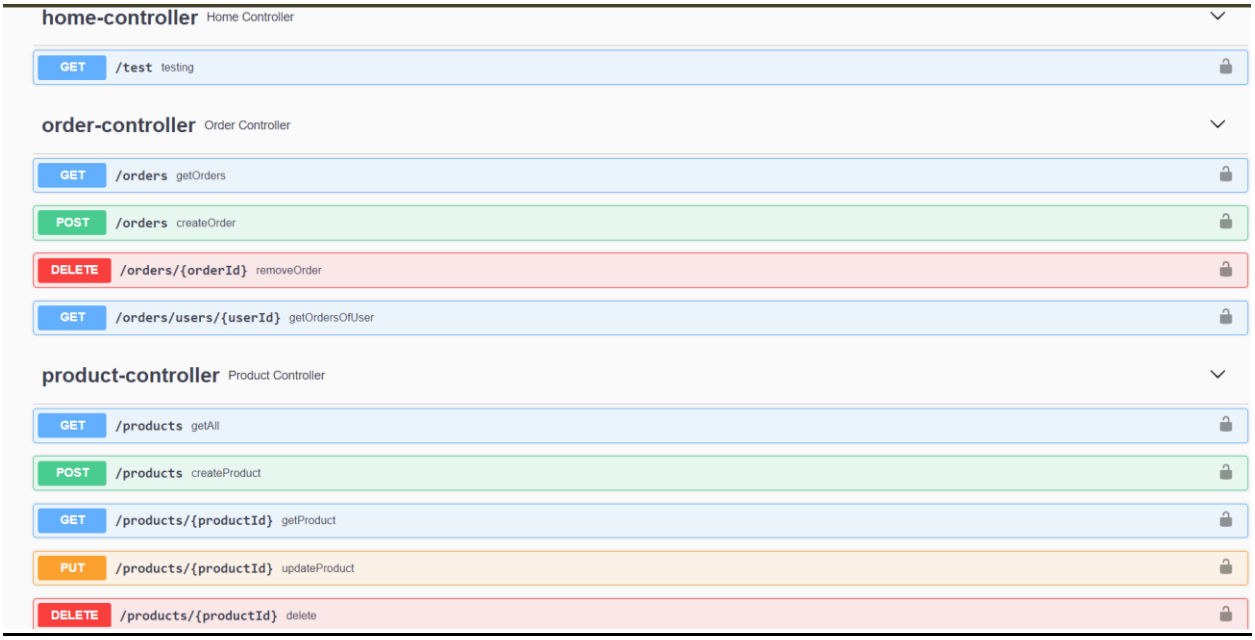
```
        user.setName(userDto.getName());

        //email update

        user.setAbout(userDto.getAbout());

        user.setGender(userDto.getGender());

        user.setPassword(userDto.getPassword());

        user.setImageName(userDto.getImageName());


        //save data

        User updatedUser = userRepository.save(user);

        UserDto updatedDto = entityToDto(updatedUser);

        return updatedDto;

    }


    @Override

    public void deleteUser(String userId) {

        User user = userRepository.findById(userId).orElseThrow(() -> new
ResourceNotFoundException("User not found with given id !!"));



        //delete user profile image

        //images/user/abc.png

        String fullPath = imagePath + user.getImageName();


        try {
```

```
            Path path = Paths.get(fullPath);
            Files.delete(path);
        } catch (NoSuchFileException ex) {
            logger.info("User image not found in folder");
            ex.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //delete user
        userRepository.delete(user);

    }

    @Override
    public PageableResponse<UserDto> getAllUser(int pageNumber, int pageSize,
String sortBy, String sortDir) {

        Sort sort = (sortDir.equalsIgnoreCase("desc")) ?
(Sort.by(sortBy).descending()) : (Sort.by(sortBy).ascending());

//        pageNumber default starts from 0
        Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);
```

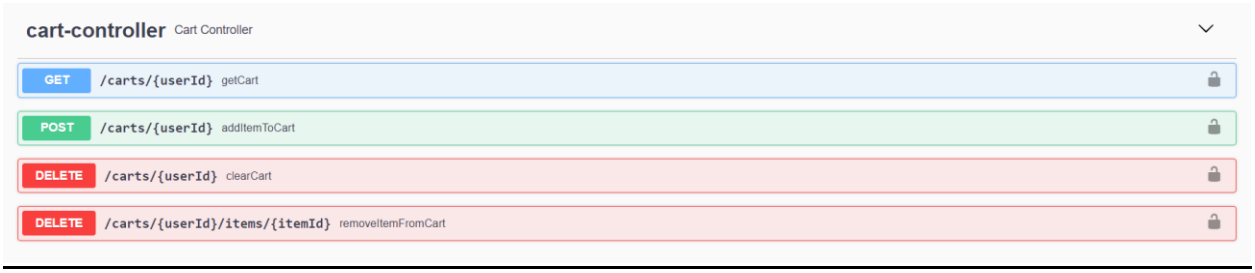
```
        Page<User> page = userRepository.findAll(pageable);


        PageableResponse<UserDto> response = Helper.getPageableResponse(page,
UserDto.class);


        return response;

    }


    @Override

    public UserDto getUserById(String userId) {

        User user = userRepository.findById(userId).orElseThrow(() -> new
ResourceNotFoundException("user not found with given id !!"));

        return entityToDto(user);

    }


    @Override

    public UserDto getUserByEmail(String email) {

        User user = userRepository.findByEmail(email).orElseThrow(() -> new
ResourceNotFoundException("User not found with given email id !!"));

        return entityToDto(user);

    }


    @Override

    public List<UserDto> searchUser(String keyword) {
```

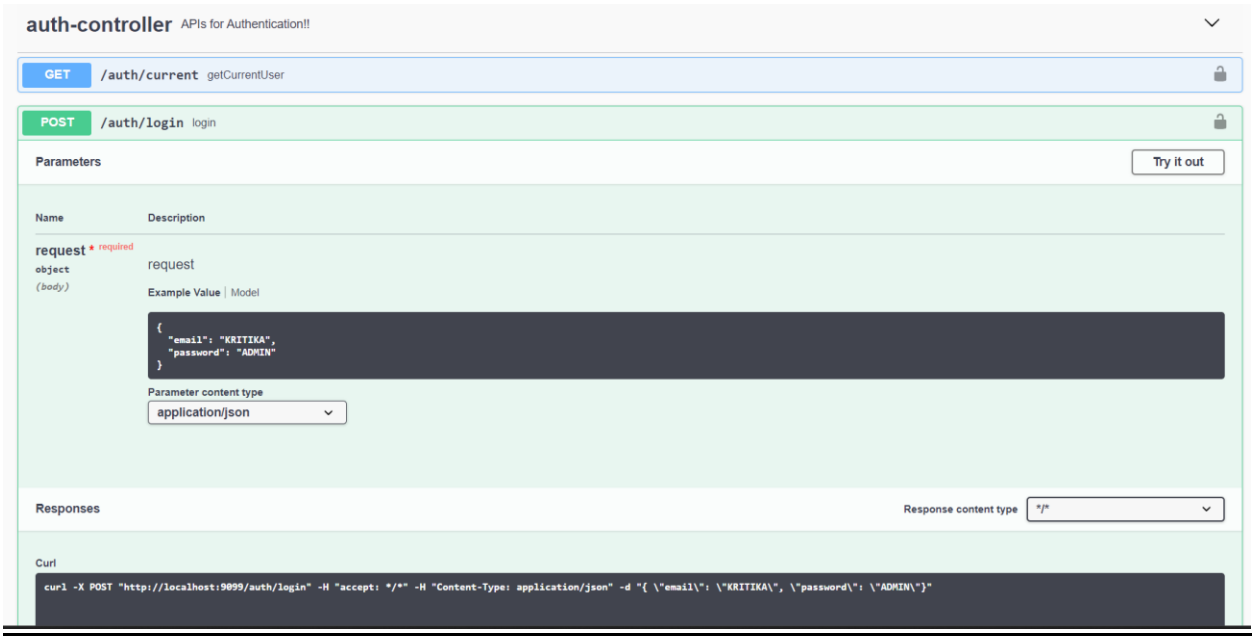
```
        List<User> users = userRepository.findByNameContaining(keyword);

        List<UserDto>      dtoList      =      users.stream().map(user      ->
entityToDto(user)).collect(Collectors.toList());

        return dtoList;

    }

    private UserDto entityToDto(User savedUser) {

        return mapper.map(savedUser, UserDto.class);

        return mapper.map(userDto, User.class);

    }

}
```

PROJECT OUTPUT




category-controller Category Controller
GET /categories getAll
POST /categories createCategory
GET /categories/{categoryId} getSingle
PUT /categories/{categoryId} updateCategory
DELETE /categories/{categoryId} deleteCategory
GET /categories/{categoryId}/products getProductsOfCategory
POST /categories/{categoryId}/products createProductWithCategory
PUT /categories/{categoryId}/products/{productId} updateCategoryOfProduct


cart-controller Cart Controller
GET /carts/{userId} getCart
POST /carts/{userId} addItemToCart
DELETE /carts/{userId} clearCart
DELETE /carts/{userId}/items/{itemId} removeItemFromCart


auth-controller APIs for Authentication!!
GET /auth/current getCurrentUser
POST /auth/login login
Parameters
Try it out
Name
Description
request * required
object
(body)
request
Example Value | Model
{
"email": "KRITIKA",
"password": "ADMIN"
}
Parameter content type
application/json
Responses
Response content type
/
Curl
curl -X POST "http://localhost:9099/auth/login" -H "accept: */*" -H "Content-Type: application/json" -d "{ \"email\": \"KRITIKA\", \"password\": \"ADMIN\"}"

order-controller Order Controller

product-controller Product Controller

user-controller REST APIs related to perform user operations !!

Models

Electronic Store Backend : APIS 1.0.0V

[Base URL: localhost:9099/]
http://localhost:9099/v2/api-docs

Made by KRITIKA DAS

Terms of service
Kritika Das - Website
Send email to Kritika Das
License of API

Authorize

auth-controller APIs for Authentication!!

basic-error-controller Basic Error Controller

cart-controller Cart Controller

category-controller Category Controller

home-controller Home Controller

CONCLUSION

In conclusion, the development of the Electronic Store project has been a comprehensive endeavor aimed at creating a robust platform for online retail operations. Through the utilization of modern technologies such as Spring Boot for backend development and Swagger for frontend interface design, the project has achieved its objectives of scalability, efficiency, and user-friendly interaction.

The implemented API endpoints, as depicted in the diagram, provide a structured approach to accessing various functionalities including user authentication, cart management, product categorization, order processing, and user management. Each endpoint serves a specific purpose in facilitating seamless interactions between users and the electronic store platform.

Throughout the project, emphasis was placed on adhering to industry best practices, ensuring security measures were integrated at every level, and optimizing performance to handle concurrent user requests effectively. The use of DTOs and controllers streamlined data handling processes while maintaining clarity and maintainability within the codebase.

Looking ahead, future enhancements could include expanding product features, integrating advanced analytics for business insights, and enhancing the frontend interface for improved user experience. These advancements would further solidify the Electronic Store's position as a competitive player in the online retail market.

In conclusion, the Electronic Store project not only showcases technical proficiency in software development but also underscores the commitment to delivering a reliable and scalable solution for modern e-commerce needs.