

**CAPSTONE PROJECT**  
**ON**  
**ShopForHome**  
**(Shopping Web Application for Home Décor Stuff)**

**WIPRO\_FSD \_JAVA**  
**Company Name –: Wipro Limited**  
**Training Partner –: Great Learning**

**Capstone Project\_Group-A\_C-VIII\_G-1**

**Under the guidance of**  
**Mr. Akash Kale**

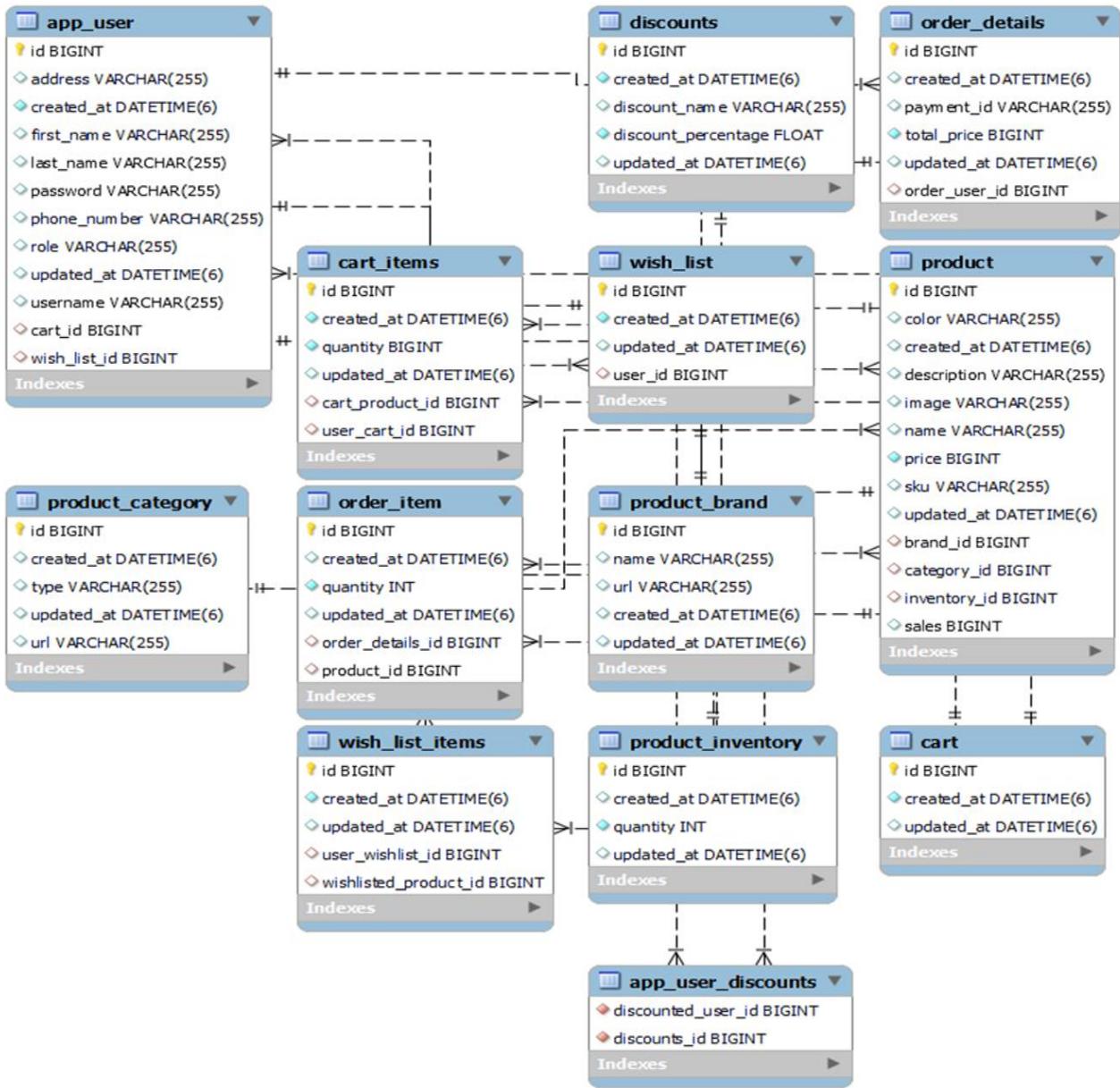
**Submitted By**

- 1. Swastika Kundu**
- 2. Vivek Kumar Singh**
- 3. Shreya Rajesh Thalkar**
- 4. Subhasis Khuntia**
- 5. Sahaj Shrimal**

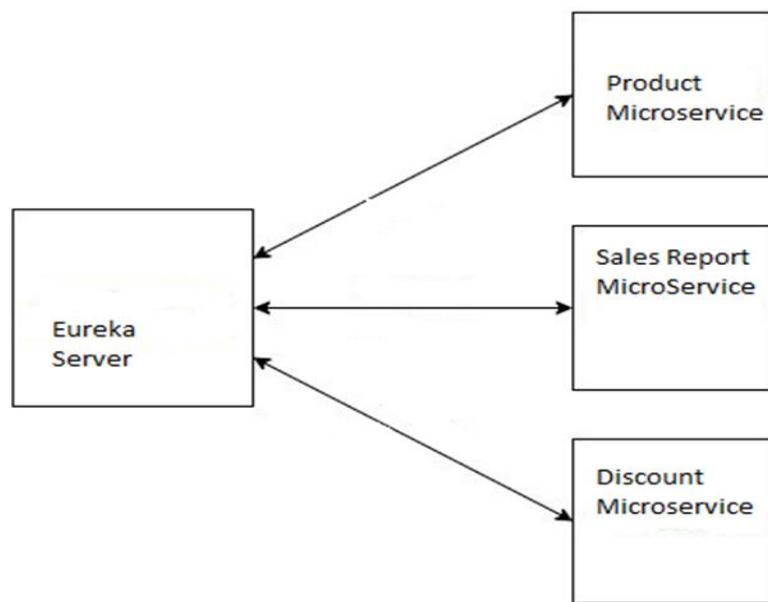
## **ABSTRACT**

Online shopping plays a great importance in the modern business environment. E-commerce is process of doing business through computer networks. A person sitting on his chair in front of a computer can access all the facilities of the Internet to buy or sell the products. Due to Covid 19, all the offline shopping stopped. So, the store wants to move to the online platforms and wants their own web application. ShopForHome has opened the door of opportunity and advantage to the firms. This project is a shopping web application for home décor stuffs. This project saves time for both the buyer and the retailer, reducing phone calls about availability, specifications, hours of operation or other information can be easily found on company and product pages.

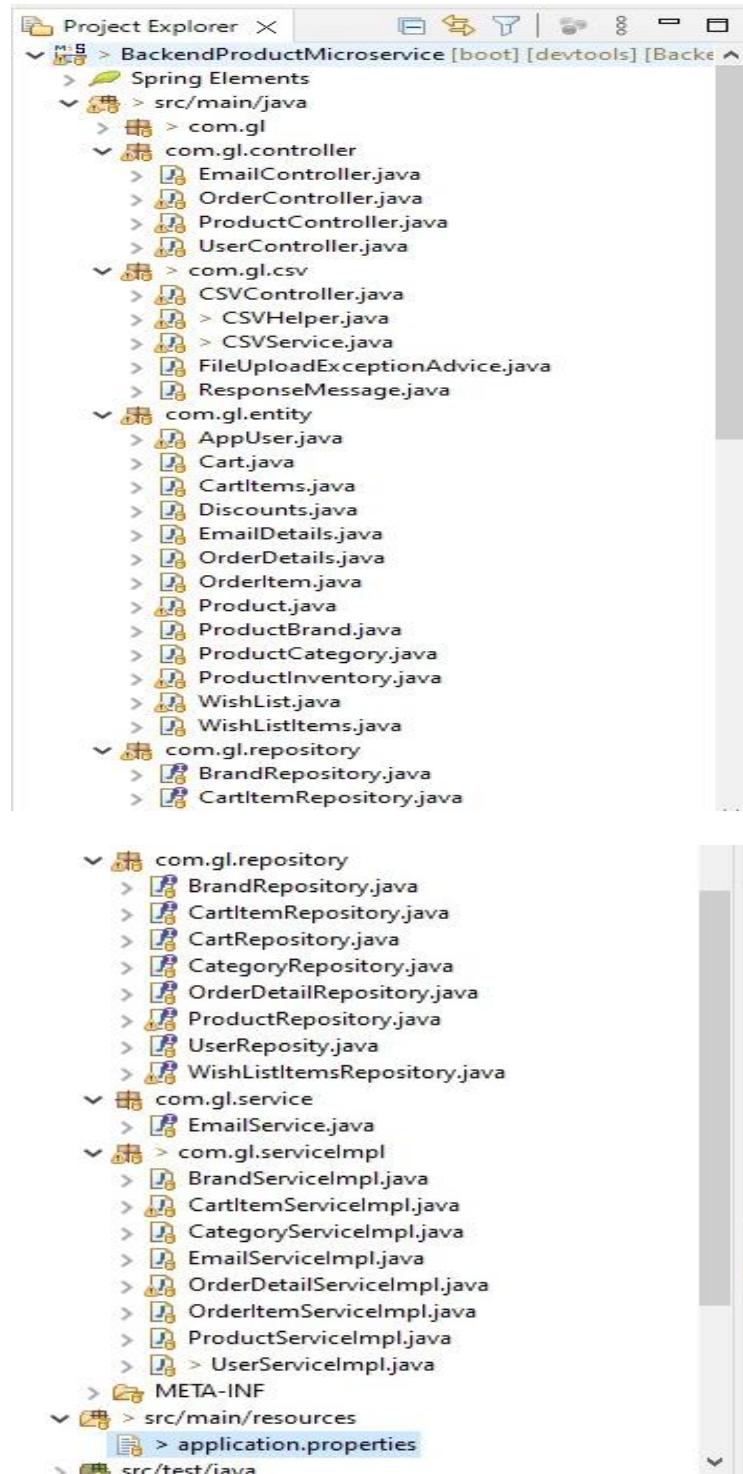
# ER DIAGRAM

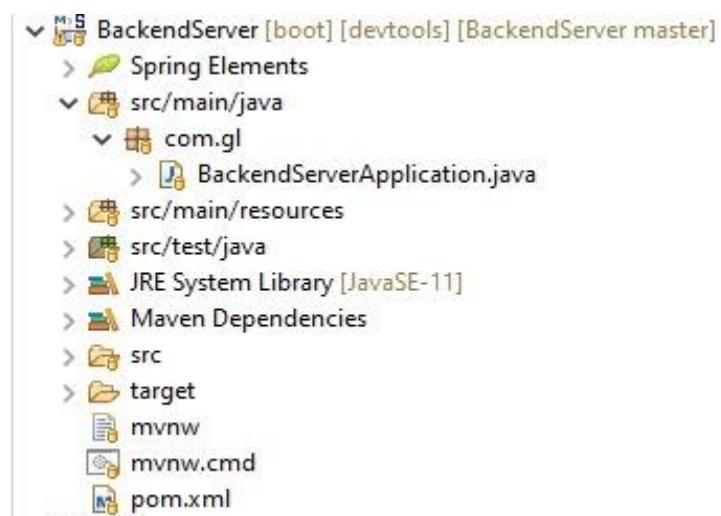
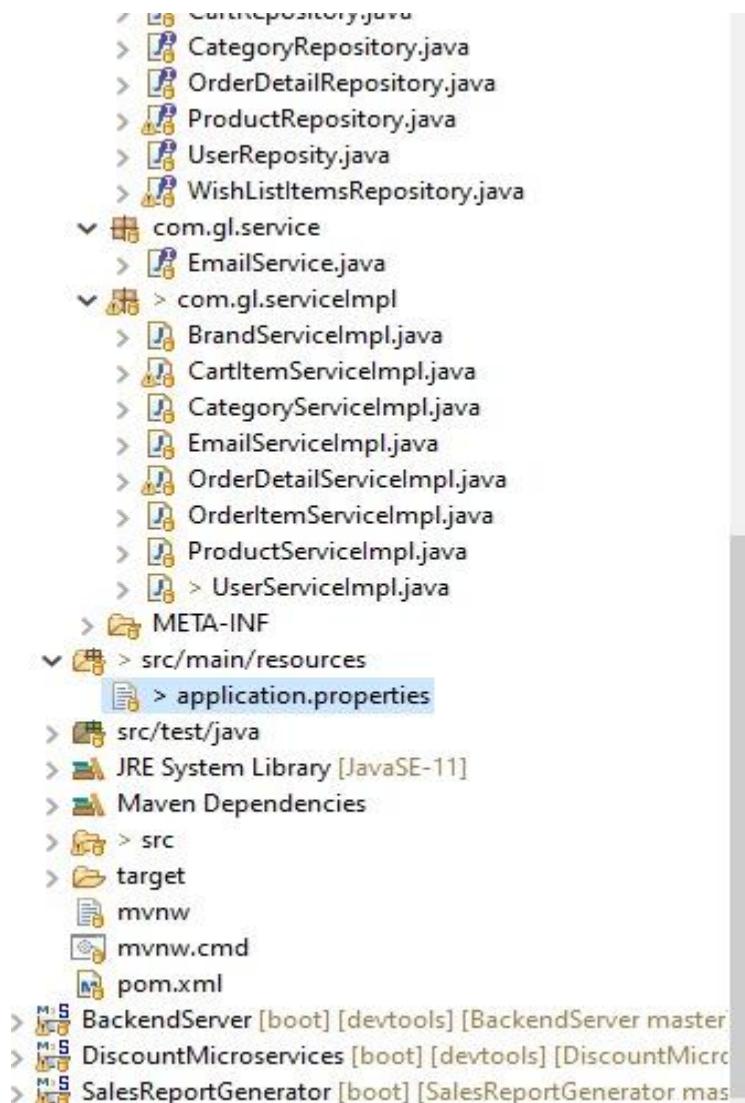


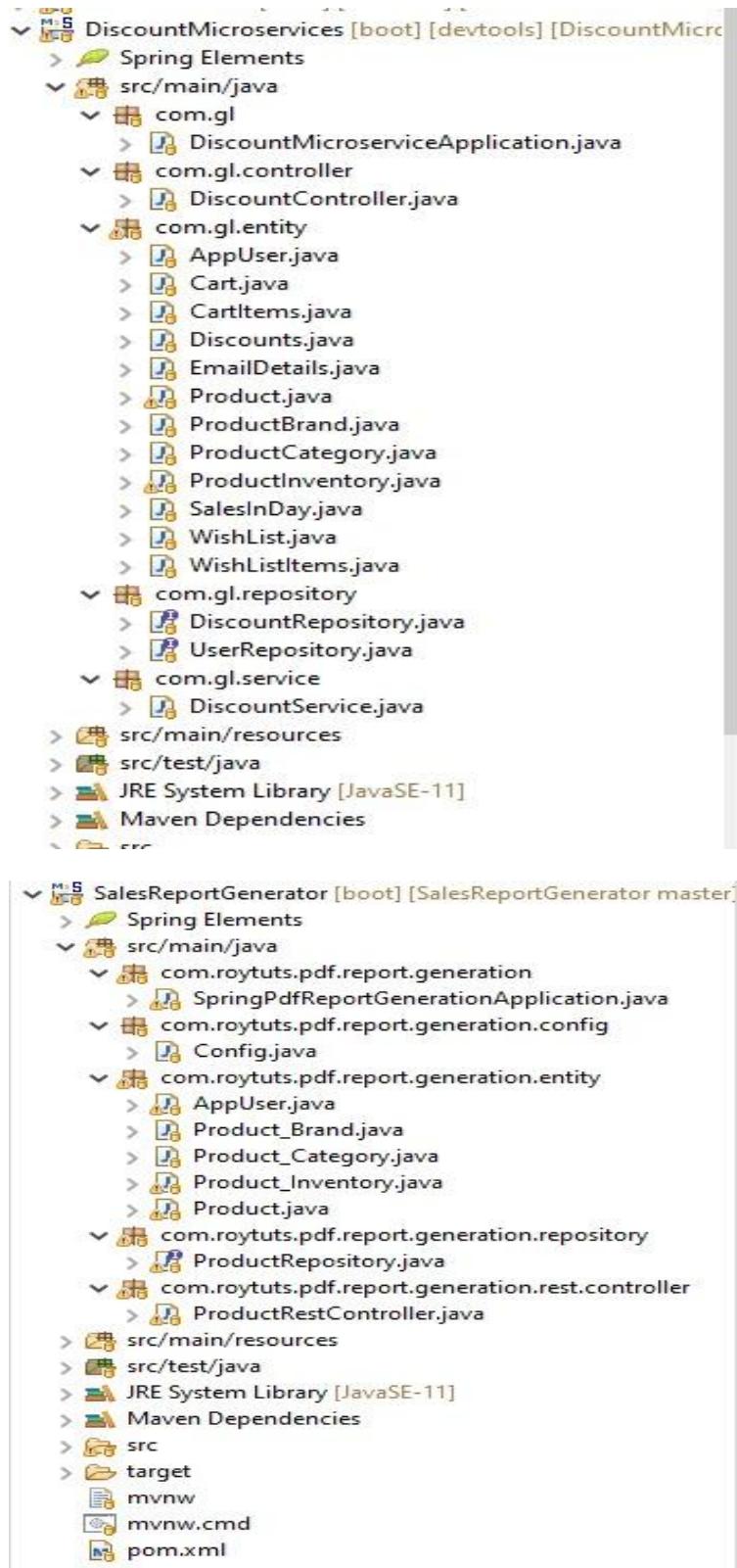
## MICROSERVICE ARCHITECTURE



# PROJECT STRUCTURE

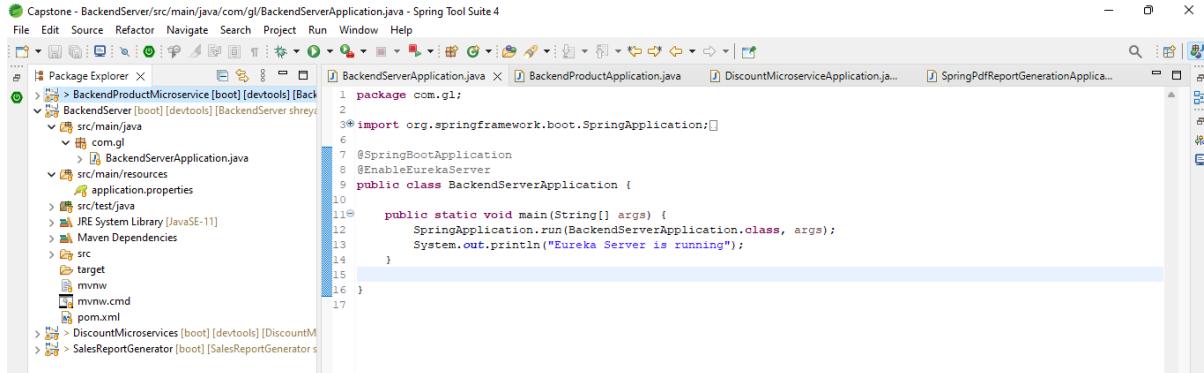






# CODE SNIPPETS

## Eureka Server



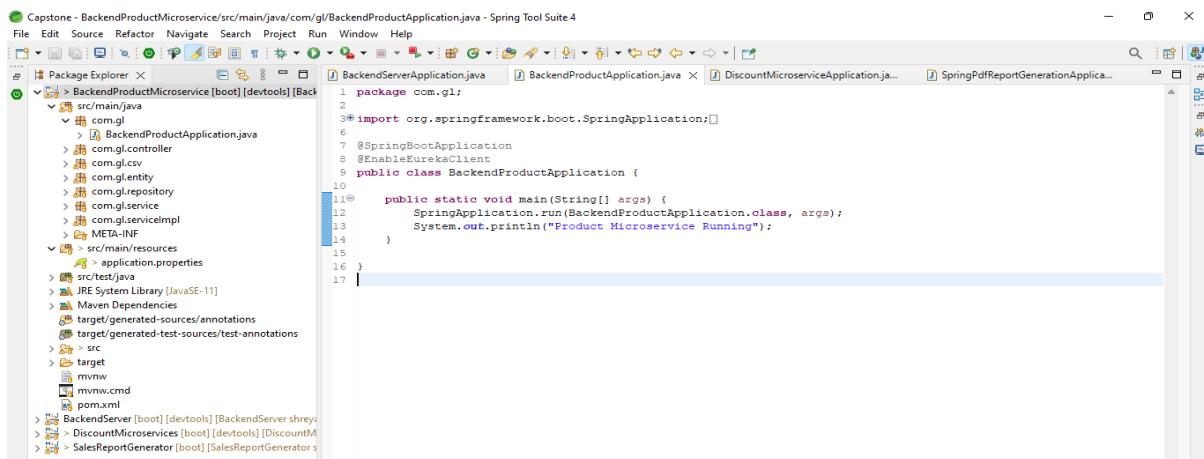
```
package com.g1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class BackendServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(BackendServerApplication.class, args);
        System.out.println("Eureka Server is running");
    }
}
```

## Product Microservice



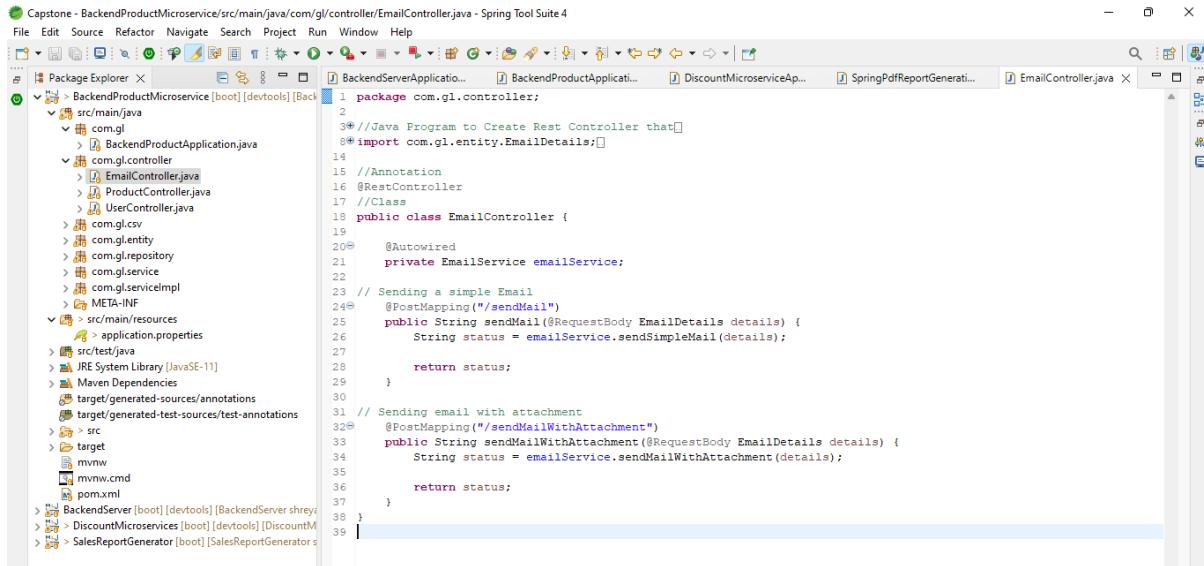
```
package com.g1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.client.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class BackendProductApplication {

    public static void main(String[] args) {
        SpringApplication.run(BackendProductApplication.class, args);
        System.out.println("Product Microservice Running");
    }
}
```

## Email Controller



```
package com.g1.controller;

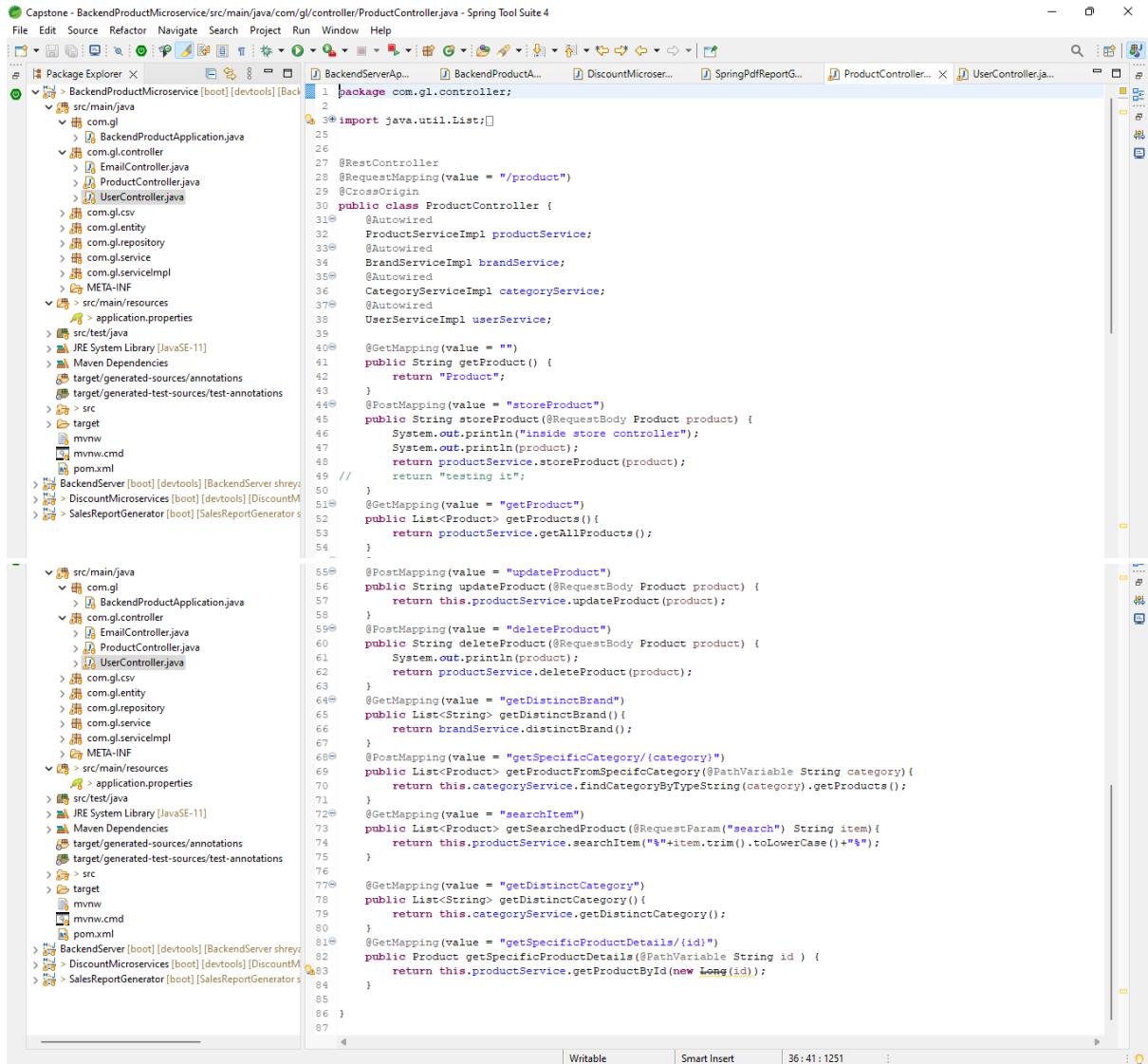
// Java Program to Create Rest Controller that
// import com.g1.entity.EmailDetails;
// Annotation
// RestController
// Class
public class EmailController {

    // Autowired
    private EmailService emailService;

    // Sending a simple Email
    @PostMapping("/sendMail")
    public String sendMail(@RequestBody EmailDetails details) {
        String status = emailService.sendSimpleMail(details);
        return status;
    }

    // Sending email with attachment
    @PostMapping("/sendMailWithAttachment")
    public String sendMailWithAttachment(@RequestBody EmailDetails details) {
        String status = emailService.sendMailWithAttachment(details);
        return status;
    }
}
```

## Product Controller



The screenshot shows the Spring Tool Suite interface with the following details:

- Title Bar:** Capstone - BackendProductMicroservice/src/main/java/com/g1/controller/ProductController.java - Spring Tool Suite 4
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbars:** Standard Java IDE toolbars.
- Left Sidebar:** Package Explorer showing the project structure:
  - BackendProductMicroservice [boot] [devtools] [BackendServer]
  - src/main/java
    - com.g1 (BackendProductApplication.java)
    - com.g1.controller (EmailController.java, ProductController.java, UserController.java)
    - com.g1.csv
    - com.g1.entity
    - com.g1.repository
    - com.g1.service (CategoryServiceImpl, ProductServiceImpl)
    - META-INF
    - src/main/resources (application.properties)
    - src/test/java
  - JRE System Library [JavaSE-11]
  - Maven Dependencies
  - target/generated-sources/annotations
  - target/generated-test-sources/test-annotations
  - src
  - target
  - mvnw
  - mvnw.cmd
  - pom.xml
- Central Area:** Code editor showing the `ProductController.java` file content.
- Bottom Status Bar:** Writable | Smart Insert | 36:41:1251 |

```
1 package com.g1.controller;
2
3 import java.util.List;
4
5
6 @RestController
7 @RequestMapping(value = "/product")
8 @CrossOrigin
9 public class ProductController {
10
11     @Autowired
12     ProductServiceImpl productService;
13
14     @Autowired
15     BrandServiceImpl brandService;
16
17     @Autowired
18     CategoryServiceImpl categoryService;
19
20     @Autowired
21     UserServiceImpl userService;
22
23     @GetMapping(value = "")
24     public String getProduct() {
25         return "Product";
26     }
27
28     @PostMapping(value = "storeProduct")
29     public String storeProduct(@RequestBody Product product) {
30         System.out.println("inside store controller");
31         System.out.println(product);
32         return productService.storeProduct(product);
33     }
34
35     @GetMapping(value = "getProduct")
36     public List<Product> getProducts() {
37         return productService.getAllProducts();
38     }
39
40     @PostMapping(value = "updateProduct")
41     public String updateProduct(@RequestBody Product product) {
42         return this.productService.updateProduct(product);
43     }
44
45     @PostMapping(value = "deleteProduct")
46     public String deleteProduct(@RequestBody Product product) {
47         System.out.println(product);
48         return productService.deleteProduct(product);
49     }
50
51     @GetMapping(value = "getDistinctBrand")
52     public List<String> getDistinctBrand() {
53         return brandService.distinctBrand();
54     }
55
56     @PostMapping(value = "getSpecificCategory/(category)")
57     public List<Product> getProductFromSpecificCategory(@PathVariable String category) {
58         return this.categoryService.findCategoryByTypeString(category).getProducts();
59     }
60
61     @GetMapping(value = "searchItem")
62     public List<Product> getSearchItem(@RequestParam("search") String item) {
63         return this.productService.searchItem(item.trim().toLowerCase() + "%");
64     }
65
66     @GetMapping(value = "getDistinctCategory")
67     public List<String> getDistinctCategory() {
68         return this.categoryService.getDistinctCategory();
69     }
70
71     @GetMapping(value = "getSpecificProductDetails/(id)")
72     public Product getSpecificProductDetails(@PathVariable String id) {
73         return this.productService.getProductById(new Long(id));
74     }
75
76     @GetMapping(value = "getProductDetails")
77     public Product getProductDetails(@PathVariable String id) {
78         return this.productService.getProductById(new Long(id));
79     }
80
81     @GetMapping(value = "getProductDetails")
82     public Product getProductDetails(@PathVariable String id) {
83         return this.productService.getProductById(new Long(id));
84     }
85
86 }
87
```

## User Controller

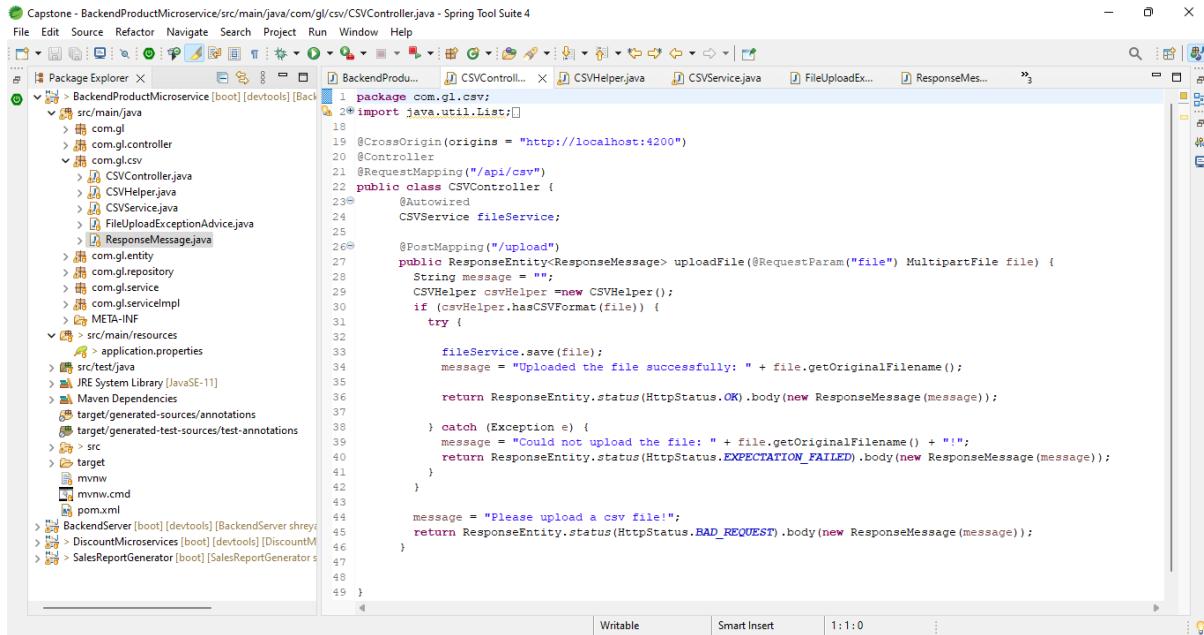
```

BackendProductMicroservice/src/main/java/com/gl/controller/UserController.java - Spring Tool Suite 4

1 package com.gl.controller;
2
3 import java.util.List;
4
5 @CrossOrigin
6 @RestController
7 @RequestMapping(value = "/user")
8 public class UserController {
9     @Autowired
10    UserServiceImpl userService;
11
12    @Autowired
13    CartItemServiceImpl cartItemService;
14
15    @Autowired
16    OrderDetailServiceImpl orderDetailService;
17
18    @PostMapping("/signin")
19    public String signin(@RequestBody AppUser user) {
20        System.out.println("inside signin controller");
21        return this.userService.saveUser(user);
22    }
23    @PostMapping("login")
24    public String login(@RequestBody Map<String, String> login) {
25        return this.userService.checkUsernameAndPassword(login.get("username"), login.get("password"));
26    }
27    @PostMapping(value = "addToCart")
28    public String addToCart(@RequestBody Cart cart) {
29        System.out.println("cart");
30        System.out.println(cart);
31        return this.userService.addToCart(cart);
32    }
33
34    @PostMapping(value = "addToWishList")
35    public String addToWishList(@RequestBody WishList wishlist) {
36        System.out.println("wishlist");
37        return this.userService.addToWishList(wishlist);
38    }
39    @PostMapping(value = "showCart")
40    public Cart showCart(@RequestBody Map<String, String> incomingUsername) {
41        String username=incomingUsername.get("username");
42        return this.userService.findUser(username).getCart();
43    }
44    @PostMapping(value = "showWishList")
45    public WishList showWishList(@RequestBody Map<String, String> incomingUsername) {
46        String username=incomingUsername.get("username");
47        return this.userService.findUser(username).getWishList();
48    }
49    @PostMapping(value = "deleteWishlistedItem")
50    public String deleteWishlistedItem(@RequestBody WishListItems item) {
51        return this.userService.deleteWishlistItem(item.getId());
52    }
53    @PostMapping(value = "deleteCartItem")
54    public String deleteCartItem(@RequestBody CartItems item) {
55        return this.userService.deleteCartItems(item);
56    }
57    @PostMapping(value = "increaseCartItemQuantity")
58    public String increaseCartItemQuantity(@RequestBody Cart cart) {
59        return this.cartItemService.increaseCartItem(cart);
60    }
61    @PostMapping(value = "decreaseCartItemQuantity")
62    public String decreaseCartItemQuantity(@RequestBody Cart cart) {
63        return this.cartItemService.decreaseCartItem(cart);
64    }
65
66    @GetMapping(value = "getAllUser")
67    public List<AppUser> getAllUser() {
68        return this.userService.getAllUser();
69    }
70    @GetMapping(value = "getUserDetail/{username}")
71    public AppUser getUserDetails(@PathVariable String username) {
72        AppUser user=this.userService.findUser(username);
73        user.setPassword(null);
74        user.setPhoneNumber(null);
75        user.setCart(null);
76        user.setWishList(null);
77        user.setCreatedAt(null);
78        user.setUpdatedAt(null);
79        return user;
80    }
81    @PostMapping(value = "placeOrder")
82    public String placeOrder(@RequestBody OrderDetails orderDetails) {
83        AppUser user=userService.findUser(orderDetails.getUsername());
84        return this.orderDetailService.saveOrderDetail(orderDetails, user);
85    }
86    @PostMapping(value = "changeUserDetails")
87    public String changeUserDetails(@RequestBody AppUser user) {
88        System.out.println(user);
89        return this.userService.changeUserDetails(user);
90    }
91    @PostMapping(value = "changePassword")
92    public String changePassword(@RequestBody Map<String, String> usernameAndPasswords) {
93        String username=usernameAndPasswords.get("username");
94        String oldPassword=usernameAndPasswords.get("oldPassword");
95        String newPassword=usernameAndPasswords.get("newPassword");
96        return this.userService.ChangePassword(username, oldPassword, newPassword);
97    }
98
99}

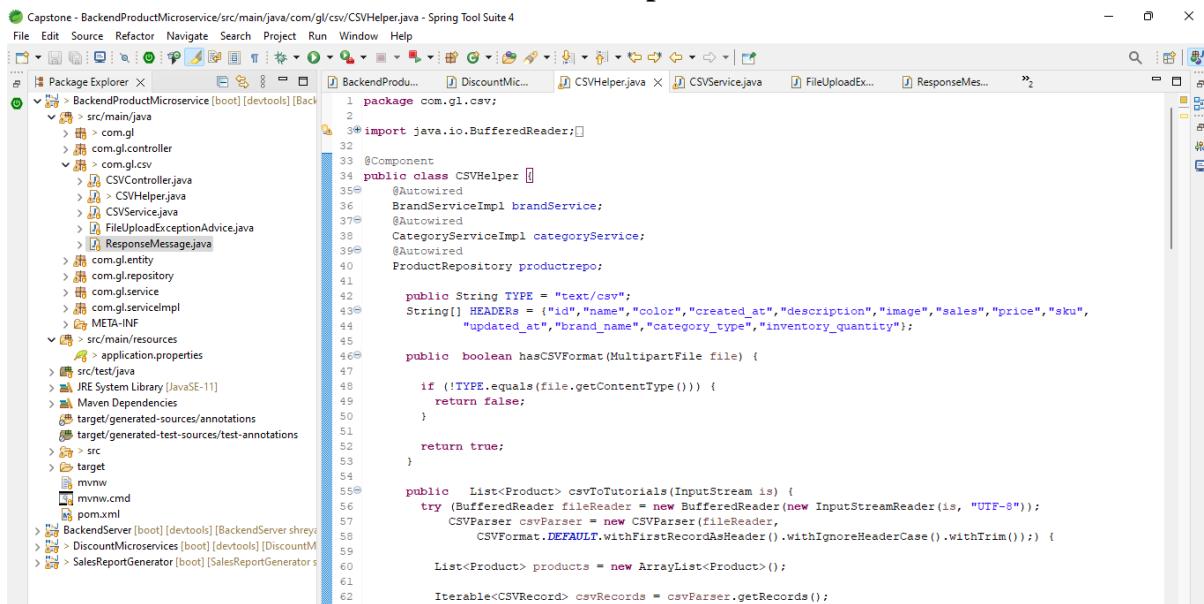
```

## CSV Controller



```
1 package com.gl.csv;
2 import java.util.List;
3
4 @CrossOrigin(origins = "http://localhost:4200")
5 @Controller
6 @RequestMapping("/api/csv")
7 public class CSVController {
8     @Autowired
9     CSVService fileService;
10
11    @PostMapping("/upload")
12    public ResponseEntity<ResponseMessage> uploadFile(@RequestParam("file") MultipartFile file) {
13        String message = "";
14        CSVHelper csvHelper = new CSVHelper();
15        if (csvHelper.hasCSVFormat(file)) {
16            try {
17                fileService.save(file);
18                message = "Uploaded the file successfully: " + file.getOriginalFilename();
19
20                return ResponseEntity.status(HttpStatus.OK).body(new ResponseMessage(message));
21            } catch (Exception e) {
22                message = "Could not upload the file: " + file.getOriginalFilename() + "!";
23                return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body(new ResponseMessage(message));
24            }
25        }
26        message = "Please upload a csv file!";
27        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new ResponseMessage(message));
28    }
29}
```

## CSV Helper



```
1 package com.gl.csv;
2
3 import java.io.BufferedReader;
4
5 @Component
6 public class CSVHelper {
7     @Autowired
8     BrandServiceImpl brandService;
9     @Autowired
10    CategoryServiceImpl categoryService;
11    @Autowired
12    ProductRepository productrepo;
13
14    public String TYPE = "text/csv";
15    String[] HEADERS = {"id", "name", "color", "created_at", "description", "image", "sales", "price", "sku",
16        "updated_at", "brand_name", "category_type", "inventory_quantity"};
17
18    public boolean hasCSVFormat(MultipartFile file) {
19
20        if (!TYPE.equals(file.getContentType())) {
21            return false;
22        }
23        return true;
24    }
25
26    public List<Product> csvToTutorials(InputStream is) {
27        try (BufferedReader fileReader = new BufferedReader(new InputStreamReader(is, "UTF-8"));
28             CSVParser csvParser = new CSVParser(fileReader,
29                     CSVFormat.DEFAULT.withFirstRecordAsHeader().withIgnoreHeaderCase().withTrim());) {
30
31        List<Product> products = new ArrayList<Product>();
32
33        Iterable<CSVRecord> csvRecords = csvParser.getRecords();
34
35        for (CSVRecord record : csvRecords) {
36            String id = record.get("id");
37            String name = record.get("name");
38            String color = record.get("color");
39            String created_at = record.get("created_at");
40            String description = record.get("description");
41            String image = record.get("image");
42            String sales = record.get("sales");
43            String price = record.get("price");
44            String sku = record.get("sku");
45            String updated_at = record.get("updated_at");
46            String brand_name = record.get("brand_name");
47            String category_type = record.get("category_type");
48            String inventory_quantity = record.get("inventory_quantity");
49
50            Product product = new Product();
51            product.setId(id);
52            product.setName(name);
53            product.setColor(color);
54            product.setCreated_at(created_at);
55            product.setDescription(description);
56            product.setImage(image);
57            product.setSales(sales);
58            product.setPrice(price);
59            product.setSku(sku);
60            product.setUpdated_at(updated_at);
61            product.setBrandName(brand_name);
62            product.setCategoryType(category_type);
63            product.setInventoryQuantity(Integer.parseInt(inventory_quantity));
64
65            products.add(product);
66        }
67    }
68}
```

The screenshot shows an IDE interface with two main panes. The left pane displays a file tree for the 'BackendServer' project, which includes source code for CSVController.java, CSVHelper.java, CSVService.java, FileUploadExceptionAdvice.java, ResponseMessage.java, and application.properties. It also shows Maven dependencies and pom.xml. The right pane shows a detailed view of the CSVController.java code, specifically the logic for reading CSV records and creating Product objects.

```
63
64     for (CSVRecord csvRecord : csvRecords) {
65         ProductBrand pb;
66         ProductCategory pc;
67         String brand=csvRecord.get("brand_name");
68         String category=csvRecord.get("category_type");
69         ProductBrand Brand=new ProductBrand();
70         if(brand!=null) {
71
72             Brand=brandService.findBrandByNameString(brand);
73
74         }
75         ProductCategory Category=new ProductCategory();
76         if(category!=null)
77             Category=categoryService.findCategoryByTypeString(category);
78         if(Brand!=null)
79             pb=Brand;
80         else
81             pb =new ProductBrand();
82         if(Category!=null)
83             pc=Category;
84         else
85             pc =new ProductCategory();
86         ProductInventory pi=new ProductInventory();
87         pi.setId(Long.parseLong(csvRecord.get("id")));
88         pi.setQuantity(Integer.parseInt(csvRecord.get("inventory_quantity")));
89         pc.setType(csvRecord.get("category_type"));
90         pb.setName(csvRecord.get("brand_name"));
91         Product product=Product.builder().
92             id(Long.parseLong(csvRecord.get("id"))).
93             name(csvRecord.get("name"))
94             description(csvRecord.get("description")).
95             sku(csvRecord.get("sku")).
96             price(Long.parseLong(csvRecord.get("price"))).
97             color(csvRecord.get("color")).
98             image(csvRecord.get("image")).
99             brand(pb).
100            category(pc).
101            inventory(pi).
102            inventory(pi).
103            sales(Long.parseLong(csvRecord.get("sales"))).
104            build();
105         productrepo.saveAndFlush(product);
106         products.add(product);
107     }
108
109     return products;
110 } catch (IOException e) {
111     e.printStackTrace();
112     throw new RuntimeException("fail to parse CSV file: " + e.getMessage());
113 }
114 catch(Exception e) {
115     e.printStackTrace();
116     throw new RuntimeException("fail to parse CSV file: " + e.getMessage());
117 }
118 }
119
120 }
```

## CSV Service

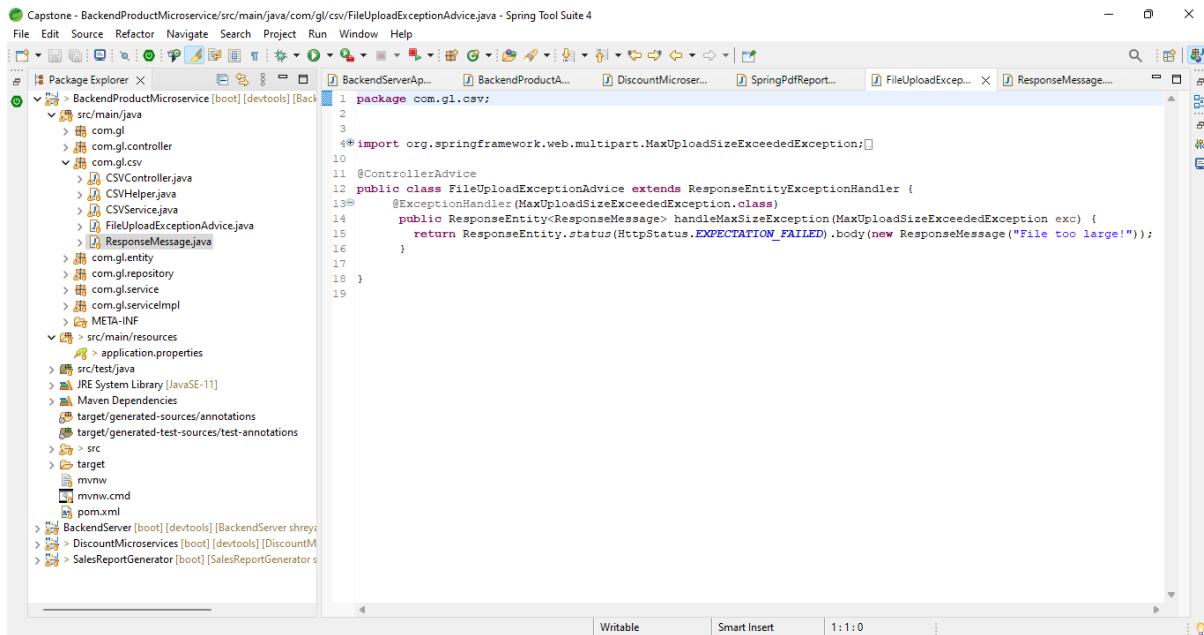
The screenshot shows the Spring Tool Suite 4 interface with the following details:

- Project Structure:** The left sidebar shows the project tree for "BackendProductMicroservice". It includes packages like com.g1.csv, com.g1.controller, and com.g1.repository, along with various Java files such as CSVController.java, CSVHelper.java, CSVService.java, FileUploadExceptionAdvice.java, ResponseMessage.java, application.properties, and several configuration files.
- Code Editor:** The main editor area displays the content of `CSVService.java`. The code defines a `CSVService` class with methods for saving CSV files and retrieving product tutorials.

```
1 package com.g1.csv;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Service
6 public class CSVService {
7     @Autowired
8     ProductRepository repository;
9     @Autowired
10    CSVHelper csvHelper;
11
12    public void save(MultipartFile file) {
13        try {
14            List<Product> products = csvHelper.csvToTutorials(file.getInputStream());
15        } catch (IOException e) {
16            throw new RuntimeException("fail to store csv data: " + e.getMessage());
17        }
18    }
19
20    // public ByteArrayInputStream load() {
21    //     List<Product> products = repository.findAll();
22    //     ByteArrayInputStream in = CSVHelper.tutorialsToCSV(products);
23    //     return in;
24    // }
25
26    public List<Product> getAllTutorials() {
27        return repository.findAll();
28    }
29
30 }
```

- Toolbars and Status Bar:** The top bar contains standard file operations (File, Edit, Source, Refactor, Navigate, Project, Run, Window, Help). The bottom status bar shows "Writtable", "Smart Insert", and "1:1:0".

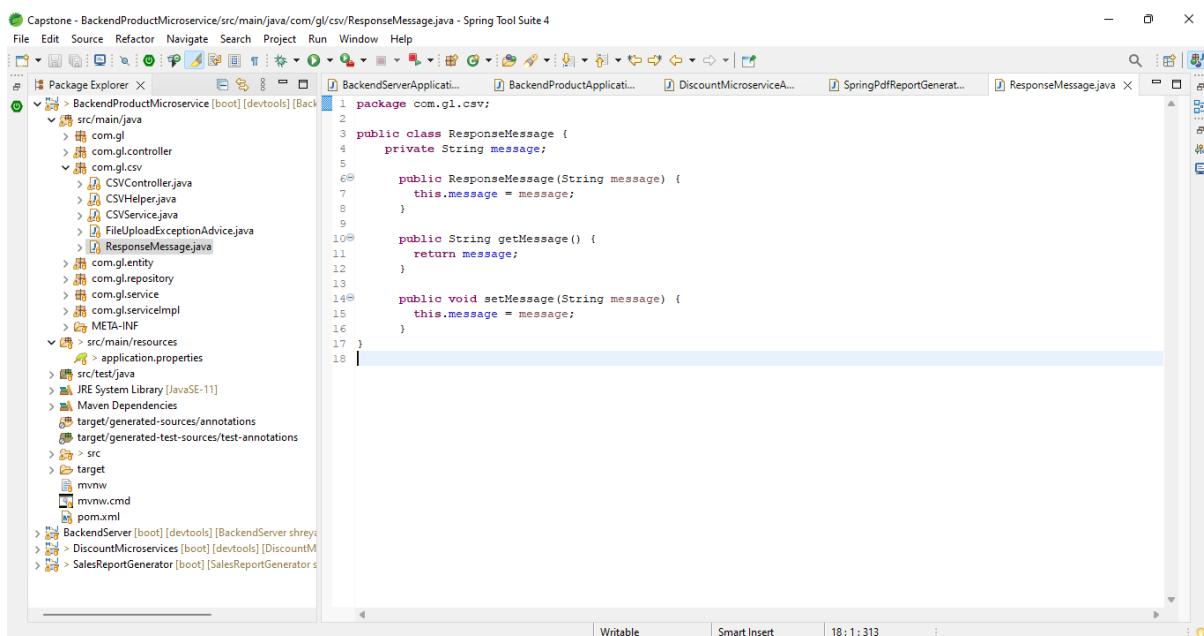
## File Upload Exception Advice



The screenshot shows the Eclipse IDE interface with the file `FileUploadExceptionAdvice.java` open in the editor. The code implements a `ResponseEntityExceptionHandler` to handle `MaxUploadSizeExceededException`.

```
1 package com.g1.csv;
2
3 import org.springframework.web.multipart.MaxUploadSizeExceededException;
4
5 @ControllerAdvice
6 public class FileUploadExceptionAdvice extends ResponseEntityExceptionHandler {
7     @ExceptionHandler(MaxUploadSizeExceededException.class)
8     public ResponseEntity<ErrorMessage> handleMaxSizeException(MaxUploadSizeExceededException exc) {
9         return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body(new ErrorMessage("File too large!"));
10    }
11 }
```

## Response Message



The screenshot shows the Eclipse IDE interface with the file `ResponseMessage.java` open in the editor. The code defines a simple message object.

```
1 package com.g1.csv;
2
3 public class ResponseMessage {
4     private String message;
5
6     public ResponseMessage(String message) {
7         this.message = message;
8     }
9
10    public String getMessage() {
11        return message;
12    }
13
14    public void setMessage(String message) {
15        this.message = message;
16    }
17 }
```

## Entities

### App User

The screenshot shows the Spring Tool Suite 4 interface with the code editor open for the `AppUser.java` file. The code defines an entity with various fields and relationships:

```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @ToString(exclude = {"wishlist", "cart", "discounts", "orderDetails"})
7 @Data
8 @Builder
9 public class AppUser {
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private long id;
13     private String username;
14     private String password;
15     private String firstName;
16     private String lastName;
17     private String address;
18     private String phoneNumber;
19     private String role;
20     @CreationTimestamp
21     @JsonIgnore
22     @Column(updatable = false, nullable = false)
23     private LocalDateTime createdAt;
24     @UpdateTimestamp
25     @JsonIgnore
26     private LocalDateTime updatedAt;
27     @OneToOne(cascade = CascadeType.ALL)
28     @JoinColumn(nullable = false)
29     private WishList wishlist;
30     @OneToOne(cascade = CascadeType.ALL)
31     private Cart cart;
32     @ManyToMany(cascade = {CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH})
33     private List<Discounts> discounts;
34     @OneToMany(mappedBy = "orderUser", cascade = CascadeType.ALL)
35     private List<OrderDetails> orderDetails;
36 }
37 }
```

## Cart

The screenshot shows the Spring Tool Suite 4 interface with the code editor open for the `Cart.java` file. The code defines an entity with various fields and relationships:

```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @Builder
8 @ToString(exclude = {"user"})
9 public class Cart {
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private long id;
13     @JsonIgnore
14     @OneToOne(mappedBy = "cart")
15     private AppUser user;
16     @CreationTimestamp
17     @Column(nullable = false, updatable = false)
18     private LocalDateTime createdAt;
19     @UpdateTimestamp
20     private LocalDateTime updatedAt;
21     @OneToMany(mappedBy = "userCart", cascade = CascadeType.ALL)
22     private List<CartItems> cartItems;
23     @Transient
24     private String username;
25 }
```

## Cart Items

The screenshot shows the Spring Tool Suite interface with the 'Cart.java' file open in the editor. The code defines a 'Cart' entity with various fields and annotations. The 'src/main/java/com/gl/entity' package is visible in the Package Explorer.

```
1 package com.gl.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Builder
10 @ToString(exclude = {"user"})
11 public class Cart {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private long id;
15     @JsonIgnore
16     @OneToOne(mappedBy = "cart")
17     private AppUser user;
18     @JsonIgnore
19     @CreationTimestamp
20     @Column(nullable = false, updatable = false)
21     private LocalDateTime createdAt;
22     @JsonIgnore
23     @UpdateTimestamp
24     private LocalDateTime updatedAt;
25     @OneToMany(mappedBy = "userCart", cascade = CascadeType.ALL)
26     private List<CartItems> cartItems;
27     @Transient
28     private String username;
29 }
```

## Discounts

The screenshot shows the Spring Tool Suite interface with the 'Discounts.java' file open in the editor. The code defines a 'Discounts' entity with various fields and annotations. The 'src/main/java/com/gl/entity' package is visible in the Package Explorer.

```
1 package com.gl.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Builder
10 public class Discounts {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private long id;
14     private String discount_name;
15     private float discount_percentage;
16     // private boolean activeOrNot;
17     @CreationTimestamp
18     @JsonIgnore
19     @Column(updatable = false, nullable = false)
20     private LocalDateTime createdAt;
21     @JsonIgnore
22     @UpdateTimestamp
23     private LocalDateTime updatedAt;
24     @ManyToOne(mappedBy = "discounts")
25     private List<AppUser> discountedUser;
26 }
```

## Email Details

The screenshot shows the Spring Tool Suite IDE interface. The title bar reads "Capstone - BackendProductMicroservice/src/main/java/com/gl/entity/EmailDetails.java - Spring Tool Suite 4". The left sidebar is the "Package Explorer" showing the project structure under "BackendProductMicroservice". The right pane displays the Java code for "EmailDetails.java".

```
1 package com.gl.entity;
2 //Java Program to Illustrate EmailDetails Class[]
3 import lombok.AllArgsConstructor;
4
5 //Annotations
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9
10 //Class
11 public class EmailDetails {
12
13     // Class data members
14     private String recipient;
15     private String msgBody;
16     private String subject;
17     private String attachment;
18 }
19
20 }
```

## Order Details

The screenshot shows the Spring Tool Suite IDE interface. The title bar reads "Capstone - BackendProductMicroservice/src/main/java/com/gl/entity/OrderDetails.java - Spring Tool Suite 4". The left sidebar is the "Package Explorer" showing the project structure under "BackendProductMicroservice". The right pane displays the Java code for "OrderDetails.java".

```
1 package com.gl.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9 @Builder
10 @ToString(exclude = "order_user")
11 public class OrderDetails {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private long id;
16     private String paymentId;
17     @ManyToOne(cascade = {CascadeType.DETACH,CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REFRESH})
18     private AppUser order_user;
19     private long totalPrice;
20     @CreationTimestamp
21     private LocalDateTime createdAt;
22     @UpdateTimestamp
23     private LocalDateTime updatedAt;
24
25     @OneToOne(mappedBy = "orderDetails", fetch = FetchType.EAGER, cascade = CascadeType.ALL)
26     private List<OrderItem> orderItem;
27
28     @Transient
29     private String username;
30 }
```

## Order Item

The screenshot shows the Spring Tool Suite 4 interface with the 'OrderItem.java' file open in the editor. The code defines a JPA entity named 'OrderItem' with various fields and annotations. The editor includes standard toolbars and a package explorer on the left.

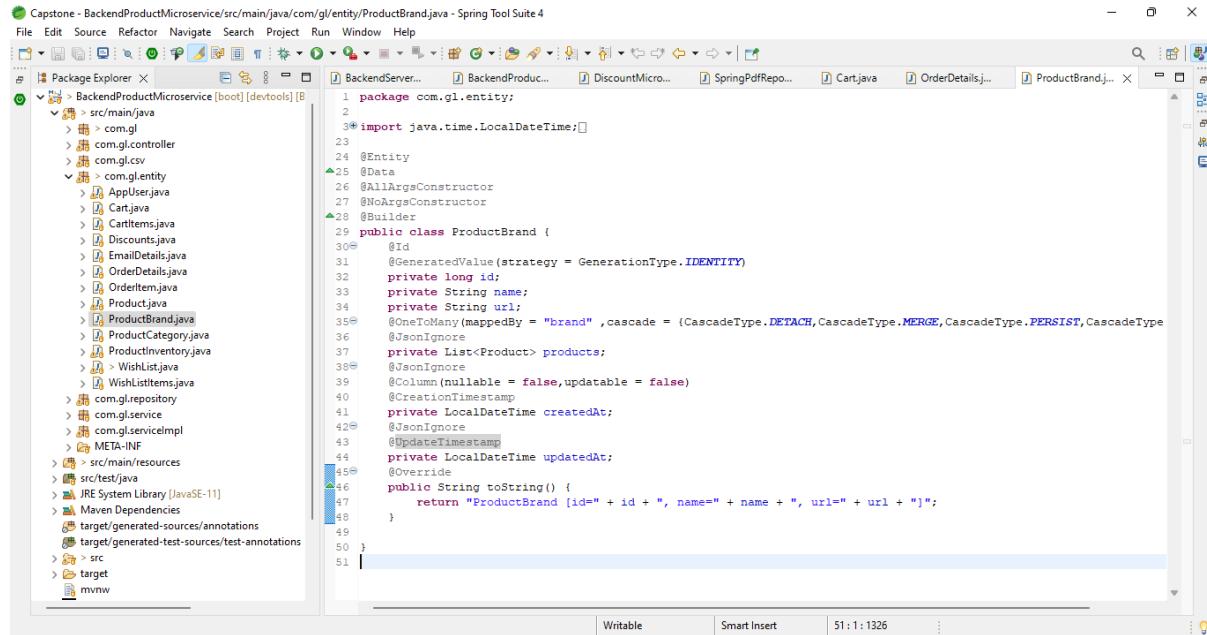
```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Builder
10 @ToString(exclude = "orderDetails")
11 public class OrderItem {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private long id;
15     private int quantity;
16
17     @JsonIgnore
18     @ManyToOne(cascade = {CascadeType.DETACH,CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REFRESH})
19     private OrderDetails orderDetails;
20
21     @ManyToOne
22     private Product product;
23
24     @CreationTimestamp
25     private LocalDateTime createdAt;
26
27     @UpdateTimestamp
28     private LocalDateTime updatedAt;
29 }
```

## Product

The screenshot shows the Spring Tool Suite 4 interface with the 'Product.java' file open in the editor. The code defines a JPA entity named 'Product' with fields for name, description, SKU, color, price, inventory, category, and brand. It also includes cascade annotations for associations. The editor includes standard toolbars and a package explorer on the left.

```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Builder
10 public class Product {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private long id;
14     private String name;
15     private String description;
16     private String sku;
17     private String color;
18     private long price;
19     @OneToOne(cascade = {CascadeType.DETACH,CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REFRESH,CascadeType.CASCADE})
20     private ProductInventory inventory;
21     private String image;
22
23     @JsonIgnore
24     @Column(updatable = false,nullable = false)
25     @CreationTimestamp
26     private LocalDateTime createdAt;
27
28     @JsonIgnore
29     @UpdateTimestamp
30     private LocalDateTime updatedAt;
31
32     @JsonIgnore
33     @ManyToOne(cascade = {CascadeType.DETACH,CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REFRESH})
34     private ProductCategory category;
35
36     @JsonIgnore
37     @ManyToOne(cascade = {CascadeType.DETACH,CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REFRESH})
38     private ProductBrand brand;
39 }
```

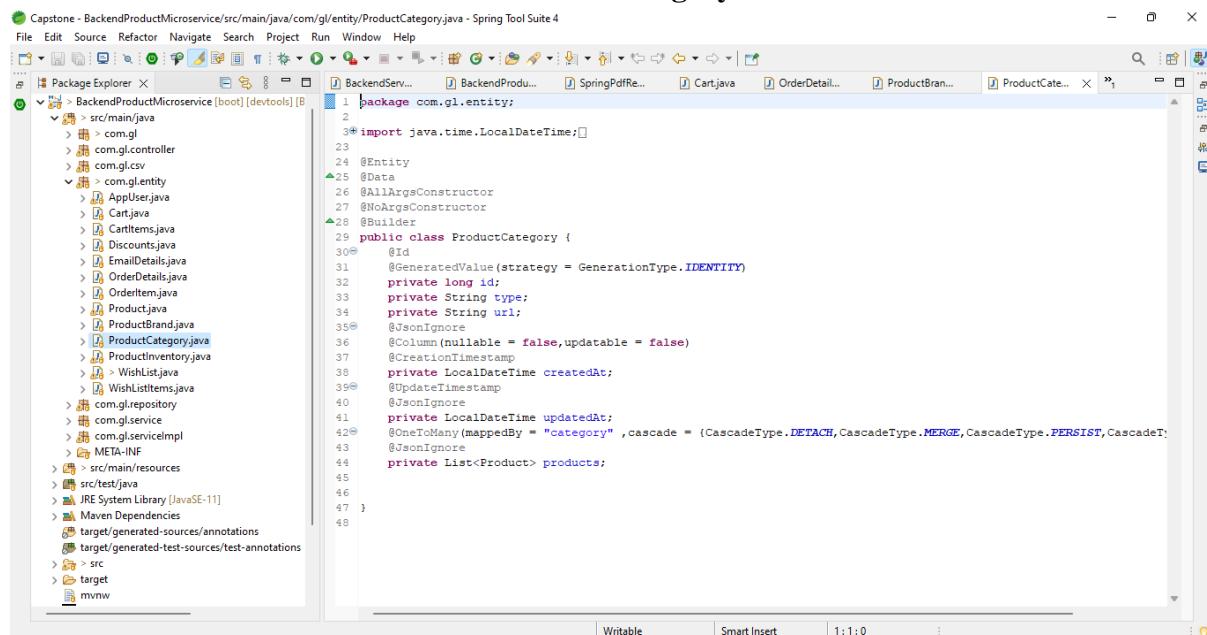
## Product Brand



The screenshot shows the Eclipse IDE interface with the 'ProductBrand.java' file open in the central editor window. The code defines a 'ProductBrand' entity with fields for id, name, and url, and a many-to-many relationship with 'Product' via the 'products' list. The code uses Lombok annotations like @Data, @Builder, and @JsonIgnore.

```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9 @Builder
10 public class ProductBrand {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private long id;
14     private String name;
15     private String url;
16     @OneToMany(mappedBy = "brand", cascade = {CascadeType.DETACH,CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REMOVE})
17     @JsonIgnore
18     private List<Product> products;
19     @Column(nullable = false, updatable = false)
20     private LocalDateTime createdAt;
21     @JsonIgnore
22     @UpdateTimestamp
23     private LocalDateTime updatedAt;
24     @Override
25     public String toString() {
26         return "ProductBrand [id=" + id + ", name=" + name + ", url=" + url + "]";
27     }
28 }
```

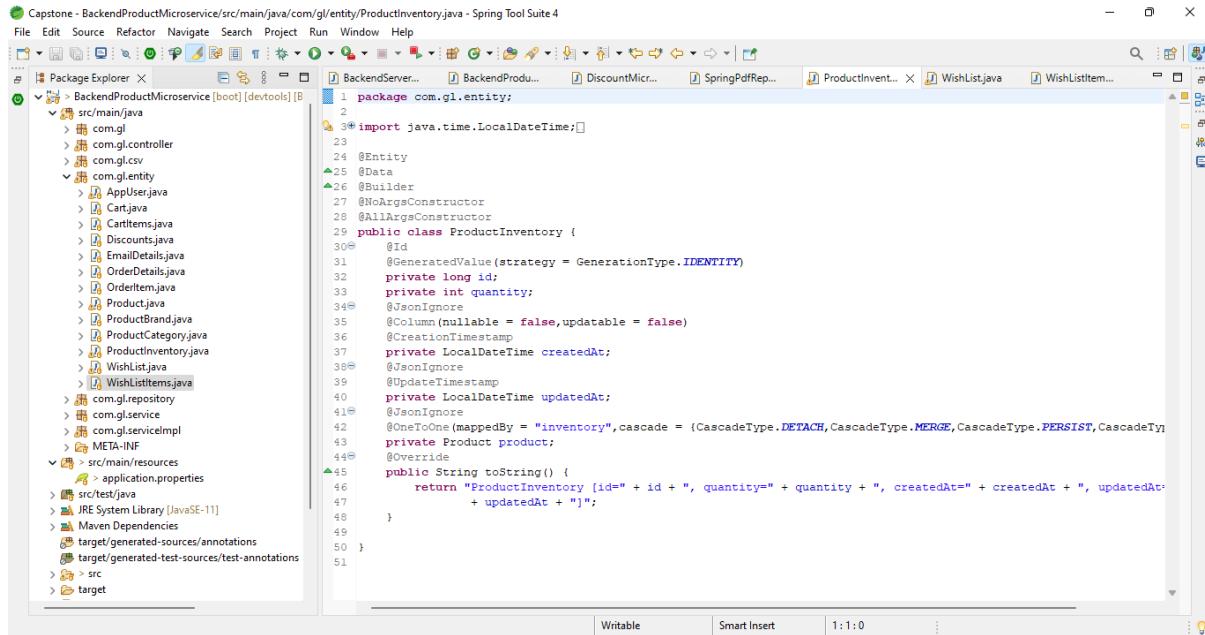
## Product Category



The screenshot shows the Eclipse IDE interface with the 'ProductCategory.java' file open in the central editor window. The code defines a 'ProductCategory' entity with fields for id, type, and url, and a many-to-many relationship with 'Product' via the 'products' list. The code uses Lombok annotations like @Data, @Builder, and @JsonIgnore.

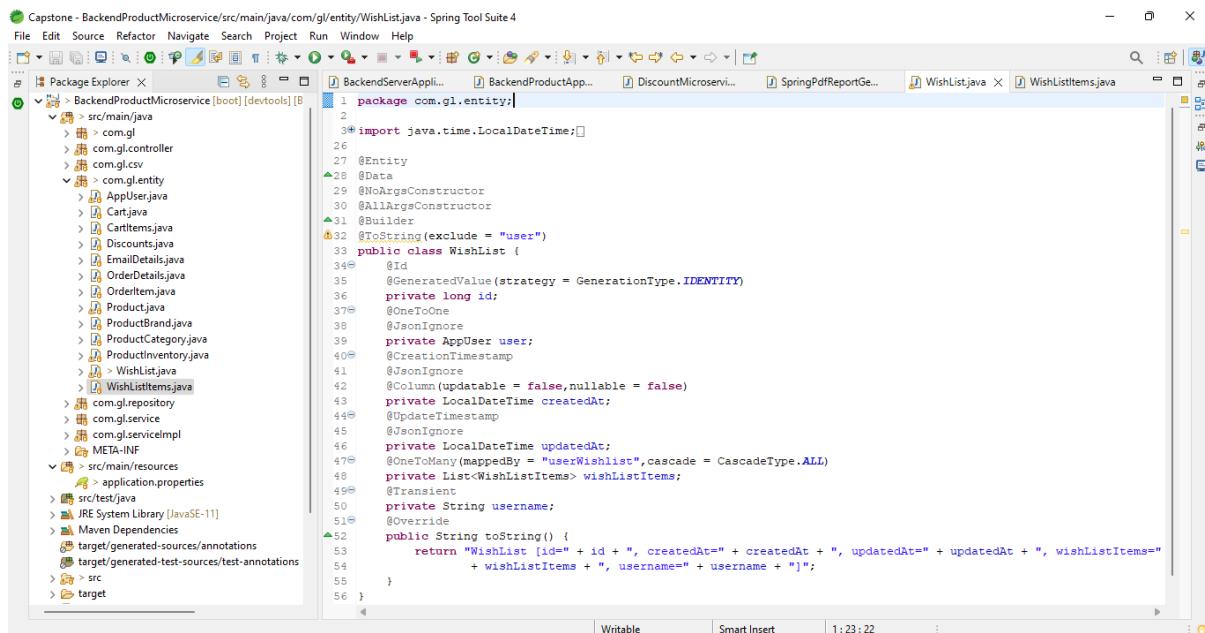
```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9 @Builder
10 public class ProductCategory {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private long id;
14     private String type;
15     private String url;
16     @JsonIgnore
17     @Column(nullable = false, updatable = false)
18     private LocalDateTime createdAt;
19     @UpdateTimestamp
20     private LocalDateTime updatedAt;
21     @OneToMany(mappedBy = "category", cascade = {CascadeType.DETACH,CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REMOVE})
22     @JsonIgnore
23     private List<Product> products;
24 }
```

## Product Inventory



```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @Builder
8 @NoArgsConstructor
9 @AllArgsConstructor
10 public class ProductInventory {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private long id;
14     private int quantity;
15     @Column(nullable = false, updatable = false)
16     @CreationTimestamp
17     private LocalDateTime createdAt;
18     @JsonIgnore
19     @UpdateTimestamp
20     private LocalDateTime updatedAt;
21     @OneToOne(mappedBy = "inventory", cascade = {CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH})
22     private Product product;
23     @Override
24     public String toString() {
25         return "ProductInventory [id=" + id + ", quantity=" + quantity + ", createdAt=" + createdAt + ", updatedAt=" + updatedAt + "]";
26     }
27 }
```

## Wishlist



```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Builder
10 @ToString(exclude = "user")
11 public class WishList {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private long id;
15     @OneToOne
16     @JsonIgnore
17     private AppUser user;
18     @CreationTimestamp
19     @JsonIgnore
20     private LocalDateTime createdAt;
21     @UpdateTimestamp
22     @JsonIgnore
23     private LocalDateTime updatedAt;
24     @OneToMany(mappedBy = "userWishlist", cascade = CascadeType.ALL)
25     private List<WishListItems> wishListItems;
26     @Transient
27     private String username;
28     @Override
29     public String toString() {
30         return "WishList [id=" + id + ", createdAt=" + createdAt + ", updatedAt=" + updatedAt + ", wishListItems=" +
31             wishListItems + ", username=" + username + "]";
32     }
33 }
```

## Wishlist Items

The screenshot shows the Eclipse IDE interface with the 'WishListItems.java' file open in the central editor. The code defines a JPA entity for wishlist items, including fields for id, creation timestamp, update timestamp, and userwishlist.

```
1 package com.g1.entity;
2
3 import java.time.LocalDateTime;
4
5 @Entity
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Builder
10 @ToString(exclude = "userWishlist")
11 public class WishListItems {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private long id;
15     @CreationTimestamp
16     @JsonIgnore
17     @Column(nullable = false, updatable = false)
18     private LocalDateTime created;
19     @UpdateTimestamp
20     @JsonIgnore
21     private LocalDateTime updated;
22     @ManyToOne(cascade = {CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH})
23     @JsonIgnore
24     private WishList userWishlist;
25     @ManyToMany
26     private Product wishlistedProduct;
27 }
```

## Brand Repository

The screenshot shows the Eclipse IDE interface with the 'BrandRepository.java' file open in the central editor. The code defines a JPA repository interface for ProductBrand entities, including methods for finding by name and ordering by name.

```
1 package com.g1.repository;
2
3 import java.util.List;
4
5 @Repository
6 public interface BrandRepository extends JpaRepository<ProductBrand, Long> {
7     @Query("select brand from ProductBrand brand where brand.name=:name")
8     public ProductBrand findBrandByName(@RequestParam("name") String name);
9     @Query("select brand.name from ProductBrand brand order by brand.name")
10    public List<String> distinctBrand();
11 }
```

## Cart Item Repository

The screenshot shows the Spring Tool Suite interface with the 'CartItemRepository.java' file open in the editor. The code defines a repository interface for managing cart items.

```
1 package com.g1.repository;
2
3 import java.util.List;
4
5 @Repository
6 public interface CartItemRepository extends JpaRepository<CartItems, Long> {
7     @Query("select item.cartProduct from CartItems item where item.cartProduct=:product and item.userCart.user.use")
8     public List<Product> getProducts(@RequestParam("product") Product product, @RequestParam("username") String use);
9
10    @Transactional
11    @Modifying
12    @Query("delete from CartItems item where item.userCart=:cart")
13    public void deleteCartItem(@RequestParam("cart") Cart cart);
14
15 }
```

## Cart Repository

The screenshot shows the Spring Tool Suite interface with the 'CartRepository.java' file open in the editor. The code defines a repository interface for managing carts.

```
1 package com.g1.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface CartRepository extends JpaRepository<Cart, Long> {
7 }
```

## Category Repository

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Capstone - BackendProductMicroservice/src/main/java/com/gl/repository/CategoryRepository.java - Spring Tool Suite 4
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Eclipse toolbar with icons for file operations.
- Left Sidebar:** Package Explorer showing the project structure:
  - BackendProductMicroservice [boot] [devtools]
  - src/main/java
    - com.gl
    - com.gl.controller
    - com.glcsv
    - com.glenity
    - com.gl.repository
      - BrandRepository.java
      - CartItemRepository.java
      - CartRepository.java
      - CategoryRepository.java
      - OrderDetailRepository.java
      - ProductRepository.java
      - UserRepository.java
      - WishListItemsRepository.java
    - com.gl.service
    - com.gl.serviceimpl
    - META-INF
  - src/main/resources
    - application.properties
  - src/test/java
  - JRE System Library [JavaSE-11]
  - Maven Dependencies
    - target/generated-sources/annotations
    - target/generated-test-sources/test-annotations
  - src
    - mvnw
    - mvnw.cmd
    - pom.xml
  - BackendServer [boot] [devtools] [BackendServer.sh]
  - DiscountMicroservices [boot] [devtools] [DiscountMicroservices.sh]

- Central Area:** Editor showing the CategoryRepository.java code:

```
1 package com.gl.repository;
2
3 import java.util.List;
4
5 @Repository
6 public interface CategoryRepository extends JpaRepository<ProductCategory, Long>{
7     @Query("select category from ProductCategory category where category.type=:type")
8     public ProductCategory findCategoryByType(@RequestParam("type") String type);
9     @Query("select cat.type from ProductCategory cat order by cat.type")
10    public List<String> getDistinctCategoryType();
11 }
12
13 }
```
- Bottom Status Bar:** Writable, Smart Insert, 1:1:0

## Order Detail Repository

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Capstone - BackendProductMicroservice/src/main/java/com/gl/repository/OrderDetailRepository.java - Spring Tool Suite 4
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Eclipse toolbar with icons for file operations.
- Left Sidebar:** Package Explorer showing the project structure (same as the previous screenshot).
- Central Area:** Editor showing the OrderDetailRepository.java code:

```
1 package com.gl.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface OrderDetailRepository extends JpaRepository<OrderDetails, Long>{
7
8 }
9
10 }
```
- Bottom Status Bar:** Writable, Smart Insert, 1:1:0

## Product Repository

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Capstone - BackendProductMicroservice/src/main/java/com/gl/repository/ProductRepository.java - Spring Tool Suite 4
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbars:** Standard toolbar on top.
- Left Sidebar:** Package Explorer showing the project structure. It includes packages like com.gl, com.gl.controller, com.glcsv, com.glentity, com.glrepository, com.glservice, and com.glserviceimpl, along with src/main/java, src/main/resources, src/test/java, and Maven Dependencies.
- Central Area:** Editor tab for ProductRepository.java. The code defines a repository interface extending JpaRepository with a searchItem method.

```
1 package com.gl.repository;
2
3 import java.util.List;
4
5 @Repository
6 public interface ProductRepository extends JpaRepository<Product, Long>{
7     @Query("select p from Product p where trim(lower(p.name)) like :item "
8           + "or trim(lower(p.brand.name)) like :item "
9           + "or trim(lower(p.description)) like :item "
10          + "or trim(lower(p.category.type)) like :item")
11     public List<Product> searchItem(@RequestParam("item") String item);
12 }
13
```

- Bottom Status Bar:** Shows Writable, Smart Insert, and 1:1:0.

## User Repository

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Capstone - BackendProductMicroservice/src/main/java/com/gl/repository/UserRepository.java - Spring Tool Suite 4
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbars:** Standard toolbar on top.
- Left Sidebar:** Package Explorer showing the project structure, identical to the Product Repository view.
- Central Area:** Editor tab for UserRepository.java. The code defines a repository interface extending JpaRepository with methods for existByUsername and findUserByUserNameAndPassword.

```
1 package com.gl.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface UserRepository extends JpaRepository<AppUser, Long>{
7     public AppUser existByUsername(@RequestParam("user") String username);
8     @Query("select user from AppUser user where user.username=:username and user.password=:password")
9     public AppUser findUserByUserNameAndPassword(@RequestParam("username") String username, @RequestParam("password") String password);
10    @Query("select user from AppUser user where user.username=:username")
11    public AppUser findUserByUserName(@RequestParam("username") String username);
12 }
13
```

- Bottom Status Bar:** Shows Writable, Smart Insert, and 1:1:0.

## Wishlist Items Repository

The screenshot shows the Spring Tool Suite 4 interface with the title "Capstone - BackendProductMicroservice/src/main/java/com/gl/repository/WishListItemsRepository.java - Spring Tool Suite 4". The left pane displays the "Package Explorer" with various Java files under the com.gl package. The right pane shows the code editor with the following content:

```
1 package com.gl.repository;
2
3 import java.util.List;
4
5 @Repository
6 public interface WishListItemsRepository extends JpaRepository<WishListItems, Long> {
7     @Query("select item.wishlistedProduct from WishListItems item where item.wishlistedProduct=:product and item.u")
8     public List<Product> getProducts(@RequestParam("product") Product product, @RequestParam("username") String use);
9 }
10
```

## Email Service

The screenshot shows the Spring Tool Suite 4 interface with the title "Capstone - BackendProductMicroservice/src/main/java/com/gl/service/EmailService.java - Spring Tool Suite 4". The left pane displays the "Package Explorer" with various Java files under the com.gl package. The right pane shows the code editor with the following content:

```
1 // Java Program to Illustrate Creation Of
2
3 package com.gl.service;
4
5 // Importing required classes
6 import com.gl.entity.EmailDetails;
7
8 // Interface
9 public interface EmailService {
10
11     // Method
12     // To send a simple email
13     String sendSimpleMail(EmailDetails details);
14
15     // Method
16     // To send an email with attachment
17     String sendMailWithAttachment(EmailDetails details);
18 }
19
20
```

## Brand Service Implementation

The screenshot shows the Spring Tool Suite interface with the following details:

- Project Explorer:** Shows the project structure under "Capstone - BackendProductMicroservice". It includes packages like com.gi.service and com.gi.serviceimpl, and various service and repository classes such as CartItemRepository, CategoryRepository, OrderDetailRepository, ProductRepository, UserRepository, WishListItemsRepository, EmailService, and BrandServiceImpl.
- Code Editor:** The main window displays the `BrandServiceImpl.java` file. The code implements the `BrandService` interface with methods for saving brands, finding brands by name or ID, and finding distinct brand names.
- Toolbars and Status Bar:** Standard Eclipse-style toolbars are at the top, and a status bar at the bottom indicates the code is "Writable" and has a line ratio of "1:1:0".

```
1 package com.gi.serviceImpl;
2
3 import java.util.List;
4
5 @Service
6 public class BrandServiceImpl {
7     @Autowired
8     private BrandRepository brandRepo;
9
10    public String saveBrand(ProductBrand brand) {
11        System.out.println(brand.getId());
12        if (!brandRepo.existsById(brand.getId())) {
13            brandRepo.save(brand);
14            return "saved successfully";
15        }
16        return "unable to save";
17    }
18
19    @Transactional
20    public List<ProductBrand> showBrand() {
21        return brandRepo.findAll();
22    }
23
24    public boolean brandExistOrNot(ProductBrand brand) {
25        return brandRepo.findByName(brand.getName())!=null;
26    }
27
28    public ProductBrand findBrandByName(ProductBrand brand) {
29        return brandRepo.findByName(brand.getName());
30    }
31    public ProductBrand findBrandByNameString(String brand) {
32        return brandRepo.findByName(brand);
33    }
34    @Transactional
35    public ProductBrand findById(long id) {
36        return brandRepo.findById(id).orElse(null);
37    }
38
39    public List<String> distinctBrand() {
40        return brandRepo.distinctBrand();
41    }
42
43}
44
45
```

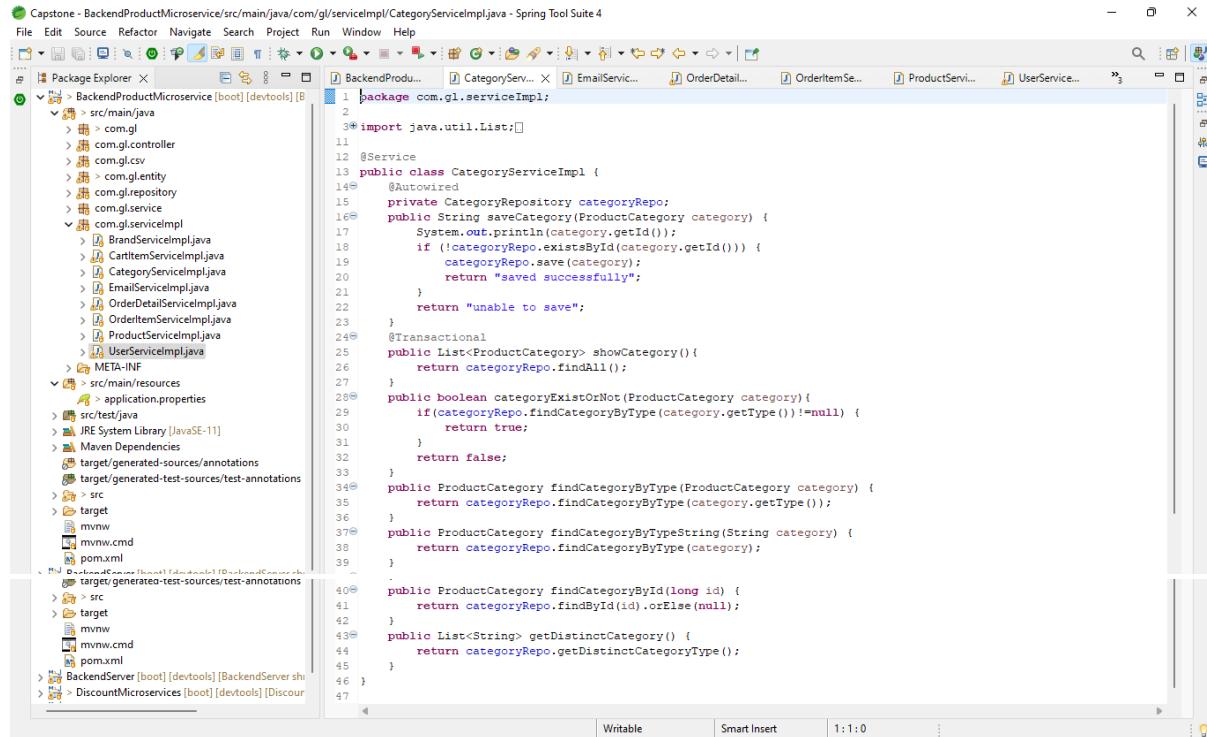
## Cart Items Service Implementation

The screenshot shows the Spring Tool Suite 4 interface with the following details:

- Project Explorer:** Shows the project structure for "BackendProductMicroservice". The package com.gl.serviceImpl contains the file CartItemServiceImpl.java.
- Code Editor:** Displays the Java code for CartItemServiceImpl.java. The code implements the CartItemService interface, utilizing various repositories and a product repository to manage cart items and products.
- Toolbars and Menus:** Standard STS toolbars and menus like File, Edit, Source, Refactor, Navigate, Project, Run, Window, Help.
- Bottom Status Bar:** Shows "Writable", "Smart Insert", and a zoom ratio of 1:1.0.

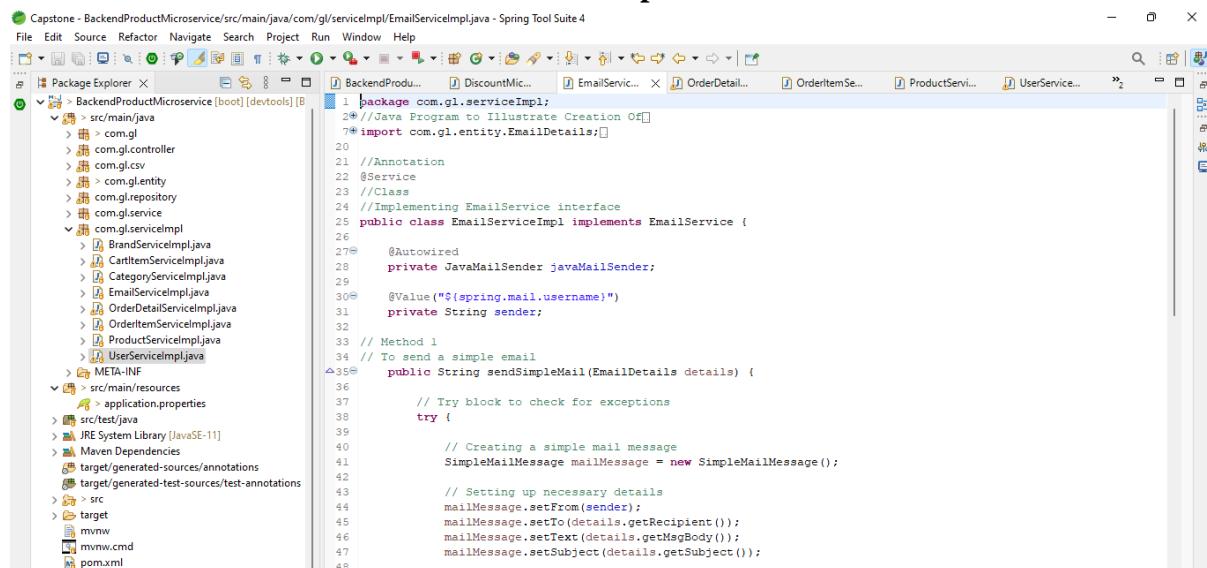
```
1 package com.gl.serviceImpl;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Service
6 public class CartItemServiceImpl {
7     @Autowired
8     private CartItemRepository cartItemRepo;
9
10    @Autowired
11    private CartRepository cartRepo;
12
13    @Autowired
14    private ProductRepository productRepo;
15
16    public String increaseCartItem(Cart cart) {
17        CartItems item = cart.getCartItems().get(0);
18        Product product = productRepo.findById(item.getCartProduct().getId()).orElse(null);
19        if (product == null) {
20            return "product doesn't exist";
21        }
22        ProductInventory inventory = product.getInventory();
23        if (inventory.getQuantity() > 0) {
24            Cart cart2 = cartRepo.findById(cart.getId()).orElse(null);
25            System.out.println(cart2);
26            if (cart2 != null) {
27                CartItems item2 = cartItemRepo.findById(item.getId()).orElse(null);
28                if (item2 != null) {
29                    item2.setQuantity(item.getQuantity());
30                    inventory.setQuantity(inventory.getQuantity() - 1);
31                    product.setSales(product.getSales() + 1);
32                    cartItemRepo.saveAndFlush(item2);
33                    productRepo.save(product);
34                    return "Item Increased";
35                }
36                return "Item doesn't exist";
37            }
38        }
39        return "Out of Stock";
40    }
41
42    public String decreaseCartItem(Cart cart) {
43        CartItems item = cart.getCartItems().get(0);
44        Product product = productRepo.findById(item.getCartProduct().getId()).orElse(null);
45        if (product == null) {
46            return "product doesn't exist";
47        }
48        ProductInventory inventory = product.getInventory();
49        Cart cart2 = cartRepo.findById(cart.getId()).orElse(null);
50        System.out.println(cart2);
51        if (cart2 != null) {
52            CartItems item2 = cartItemRepo.findById(item.getId()).orElse(null);
53            if (item2 != null) {
54                item2.setQuantity(item.getQuantity());
55                inventory.setQuantity(inventory.getQuantity() + 1);
56                product.setSales(product.getSales() - 1);
57                cartItemRepo.saveAndFlush(item2);
58                productRepo.save(product);
59                return "Item decreased";
60            }
61            return "Item doesn't exist";
62        }
63    }
64    return "User not found";
65}
66
67}
68
69}
70
71}
72
73}
74}
75}
```

## Category Service Implementation



```
1 package com.g1.serviceImpl;
2
3 import java.util.List;
4
5 @Service
6 public class CategoryServiceImpl {
7     @Autowired
8     private CategoryRepository categoryRepo;
9
10    public String saveCategory(ProductCategory category) {
11        System.out.println(category.getId());
12        if (!categoryRepo.existsById(category.getId())) {
13            categoryRepo.save(category);
14            return "saved successfully";
15        }
16        return "unable to save";
17    }
18
19    @Transactional
20    public List<ProductCategory> showCategory() {
21        return categoryRepo.findAll();
22    }
23
24    public boolean categoryExistOrNot(ProductCategory category) {
25        if (categoryRepo.findCategoryByType(category.getType()) != null) {
26            return true;
27        }
28        return false;
29    }
30
31    public ProductCategory findCategoryByType(ProductCategory category) {
32        return categoryRepo.findCategoryByType(category.getType());
33    }
34
35    public ProductCategory findCategoryByTypeString(String category) {
36        return categoryRepo.findCategoryByType(category);
37    }
38
39
40    public ProductCategory findCategoryById(long id) {
41        return categoryRepo.findById(id).orElse(null);
42    }
43
44    public List<String> getDistinctCategory() {
45        return categoryRepo.getDistinctCategoryType();
46    }
47}
```

## Email Service Implementation



```
1 package com.g1.serviceImpl;
2
3 // Java Program to Illustrate Creation Of
4 // Class
5 // Implementing EmailService interface
6 public class EmailServiceImpl implements EmailService {
7
8     @Autowired
9     private JavaMailSender javaMailSender;
10
11    @Value("${spring.mail.username}")
12    private String sender;
13
14    // Method 1
15    // To send a simple email
16    public String sendSimpleMail(EmailDetails details) {
17
18        // Try block to check for exceptions
19        try {
20
21            // Creating a simple mail message
22            SimpleMailMessage mailMessage = new SimpleMailMessage();
23
24            // Setting up necessary details
25            mailMessage.setFrom(sender);
26            mailMessage.setTo(details.getRecipient());
27            mailMessage.setText(details.getMsgBody());
28            mailMessage.setSubject(details.getSubject());
29
30        } catch (Exception e) {
31            e.printStackTrace();
32        }
33    }
34}
```

```

BackendProductMicroservice [boot] [devtools] [B]
  +-- src/main/java
    +-- com/gl
      +-- controller
      +-- csv
      +-- entity
      +-- repository
      +-- service
      +-- serviceimpl
        +-- BrandServiceimpl.java
        +-- CartItemServiceimpl.java
        +-- CategoryServiceimpl.java
        +-- EmailServiceimpl.java
        +-- OrderDetailServiceimpl.java
        +-- OrderItemServiceimpl.java
        +-- ProductServiceimpl.java
        +-- UserServiceimpl.java
      +-- META-INF
    +-- src/main/resources
      +-- application.properties
    +-- src/test/java
  +-- IRE System Library [JavaSE-11]
  +-- Maven Dependencies
    +-- target/generated-sources/annotations
    +-- target/generated-test-sources/test-annotations
  +-- src
  +-- target
    +-- mavenw
    +-- mvnw.cmd
  pom.xml

BackendServer [boot] [devtools] [BackendServer sh]
  +-- application.properties
  +-- src/test/java
  +-- IRE System Library [JavaSE-11]
  +-- Maven Dependencies
    +-- target/generated-sources/annotations
    +-- target/generated-test-sources/test-annotations
  +-- src
  +-- target
    +-- mavenw
    +-- mvnw.cmd
  pom.xml

BackendServer [boot] [devtools] [BackendServer sh]
DiscountMicroservices [boot] [devtools] [Discour

```

```

49         // Sending the mail
50         javaMailSender.send(mailMessage);
51         return "Mail Sent Successfully..";
52     }
53
54     // Catch block to handle the exceptions
55     catch (Exception e) {
56         return "Error While Sending Mail";
57     }
58 }
59
60 // Method 2
61 // To send an email with attachment
62 @public String sendMailWithAttachment(EmailDetails details) {
63     // Creating a mime message
64     MimeMessage mimeMessage = javaMailSender.createMimeMessage();
65     MimeMessageHelper mimeMessageHelper;
66
67     try {
68
69         // Setting multipart as true for attachments to
70         // be send
71         mimeMessageHelper = new MimeMessageHelper(mimeMessage, true);
72         mimeMessageHelper.setFrom(sender);
73         mimeMessageHelper.setTo(details.getRecipient());
74         mimeMessageHelper.setText(details.getMsgBody());
75         mimeMessageHelper.setSubject(details.getSubject());
76
77         // Adding the attachment
78         FileSystemResource file = new FileSystemResource(new File(details.getAttachment()));
79
80         mimeMessageHelper.addAttachment(file.getFilename(), file);
81
82         // Sending the mail
83         javaMailSender.send(mimeMessage);
84         return "Mail sent Successfully";
85     }
86
87     // Catch block to handle MessagingException
88     catch (MessagingException e) {
89
90         // Display message when exception occurred
91         return "Error while sending mail!!!";
92     }
93 }
94
95

```

Writable | Smart Insert | 1:1:0 | ...

## Order Detail Service Implementation

```

Capstone - BackendProductMicroservice/src/main/java/com/gl/serviceimpl/OrderDetailServiceImpl.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Project Run Window Help

```

```

BackendServer [boot] [devtools] [BackendServer sh]
  +-- src/main/java
    +-- com/gl
      +-- controller
      +-- csv
      +-- entity
      +-- repository
      +-- service
      +-- serviceimpl
        +-- BrandServiceimpl.java
        +-- CartItemServiceimpl.java
        +-- CategoryServiceimpl.java
        +-- EmailServiceimpl.java
        +-- OrderDetailServiceimpl.java
        +-- OrderItemServiceimpl.java
        +-- ProductServiceimpl.java
        +-- UserServiceimpl.java
      +-- META-INF
    +-- src/main/resources
      +-- application.properties
    +-- src/test/java
  +-- IRE System Library [JavaSE-11]
  +-- Maven Dependencies
    +-- target/generated-sources/annotations
    +-- target/generated-test-sources/test-annotations
  +-- src
  +-- target
    +-- mavenw
    +-- mvnw.cmd
  pom.xml

BackendServer [boot] [devtools] [BackendServer sh]
DiscountMicroservices [boot] [devtools] [Discour

```

```

29     orderDetails.getOrderItem().forEach(a->{
30         if(a.getProduct().getInventory().getQuantity()==0) {
31
32             else if (a.getQuantity() > a.getProduct().getInventory().getQuantity()) {
33                 a.setQuantity(a.getProduct().getInventory().getQuantity());
34                 int sales=a.getQuantity();
35                 Product product=a.getProduct();
36                 product.setSales(product.getSales()+sales);
37                 product.getInventory().setQuantity(product.getInventory().getQuantity()-sales);
38                 productRepo.save(product);
39             }
40             else {
41                 int sales=a.getQuantity();
42                 Product product=a.getProduct();
43                 product.setSales(product.getSales()+sales);
44                 product.getInventory().setQuantity(product.getInventory().getQuantity()-sales);
45                 productRepo.save(product);
46             }
47
48             if(a.getProduct().getInventory().getQuantity()<10) {
49                 EmailDetails emailDetails=new EmailDetails(
50                     "subhasiskhuria506@gmail.com",
51                     a.getProduct().getName()+"is now below 10",
52                     "running out of stock", "");
53                 emailServiceImpl.sendSimpleMail(emailDetails);
54             }
55         };
56         orderDetailRepo.save(orderDetails);
57         Cart cartUser.getCart();
58         cartItemRepo.deleteCartItem(cart);
59         return "Order Saved";
60     }
61 }
62

```

Writable | Smart Insert | 1:1:0 | ...

## Order Item Service Implementation

```
1 package com.g1.serviceImpl;
2
3 import org.springframework.stereotype.Service;
4
5
6 @Service
7 public class OrderItemServiceImpl {
8
9 }
```

## Product Service Implementation

```
1 package com.g1.serviceImpl;
2
3 import java.util.List;
4
5
6 @Service
7 public class ProductServiceImpl {
8
9     @Autowired
10    ProductRepository productRepo;
11
12    @Autowired
13    BrandServiceImpl brandServiceImpl;
14
15    @Autowired
16    CategoryServiceImpl categoryServiceImpl;
17
18
19    public String storeProduct(Product product) {
20        ProductBrand brand=brandServiceImpl.findBrandByName(product.getBrand());
21        ProductCategory category=categoryServiceImpl.findCategoryByType(product.getType());
22
23        if (brand!=null) {
24            product.setBrand(brand);
25        }
26        if(category!=null) {
27            product.setCategory(category);
28        }
29        System.out.println(product);
30        productRepo.saveAndFlush(product);
31        return "Product stored successfully";
32    }
33
34    @Transactional
35    public List<Product> getAllProducts(){
36        return productRepo.findAll();
37    }
38
39    public String updateProduct(Product product) {
40        if(productRepo.findById(product.getId())!=null) {
41            productRepo.saveAndFlush(product);
42            return "Updated";
43        }
44        return "Product doesn't exist";
45    }
46
47    public String deleteProduct(Product product) {
48        if(productRepo.existsById(product.getId())) {
49            productRepo.delete(product);
50            return "Deleted";
51        }
52        return "Id not found";
53    }
54
55    public List<Product> searchItem(String item){
56        return this.productRepo.searchItem(item);
57    }
58
59    public Product getProductById(Long id) {
60        return this.productRepo.findById(id).orElse(null);
61    }
62}
```

## User Service Implementation

The screenshot shows the Spring Tool Suite (STS) interface with the 'BackendProductMicroservice' project open. The left side displays the 'Package Explorer' showing the project structure, which includes 'src/main/java' with various service implementation classes like 'BrandServiceImpl.java', 'CartItemServiceImpl.java', 'CategoryServiceImpl.java', 'EmailServiceImpl.java', 'OrderDetailServiceImpl.java', 'ProductServiceImpl.java', and 'UserServiceImpl.java'. It also includes 'src/main/resources' with 'application.properties', 'src/test/java', 'META-INF', 'Maven Dependencies', and 'target' directories. The right side of the interface shows the 'UserServiceImpl.java' file content:

```
1 package com.g1.serviceimpl;
2
3 import java.util.List;
4
5 @Service
6 public class UserServiceImpl {
7     @Autowired
8     CartItemRepository cartitemdao;
9
10    @Autowired
11    UserRepository userdao;
12
13    @Autowired
14    WishListItemsRepository wishlistItemDao;
15
16    @Autowired
17    ProductRepository productRepo;
18
19    @Autowired
20    EmailServiceImpl emailService;
21
22    public String saveUser(AppUser user) {
23        if (this.userdao.findByUsername(user.getUsername()) == null) {
24            user.setWishList(WishList.Builder().user(user).build());
25            user.setCart(new Cart());
26            userdao.saveAndFlush(user);
27            return "User Saved Successfully";
28        }
29        return "User Already Exist";
30    }
31
32    public String addToCart(Cart cart) {
33        Product product = productRepo.findById(cart.getCartItems().get(0).getCartProduct().getId()).orElse(null);
34        if (cartitemdao.getProducts(cart.getCartItems().get(0).getCartProduct(), cart.getUsername()).size() != 0)
35            return "Already in your Cart";
36
37
38        if (product != null && product.getInventory().getQuantity() > 0) {
39            AppUser user = userdao.findByUsername(cart.getUsername());
40            user.getCart().getCartItems().add(cart.getCartItems().get(0));
41            user.getCart().setUser(user);
42            user.getCart().getCartItems().forEach(theCart -> theCart.setUserCart(user.getCart()));
43            System.out.println(user.getCart().getCartItems() + "cart items");
44            userdao.saveAndFlush(user);
45            ProductInventory inventory = product.getInventory();
46            inventory.setQuantity(product.getInventory().getQuantity() - 1);
47            product.setInventory(inventory);
48            product.setSales(product.getSales() + 1);
49            if (product.getInventory().getQuantity() < 10) {
50                System.out.println("stock is less than 10");
51                EmailDetails emails = new EmailDetails(
52                    "subhasishkuntia06@gmail.com",
53                    "stock of " + product.getName() + " is now below 10",
54                    product.getName() + " stocks now below 10",
55                    "");
56                emailService.sendSimpleMail(emails);
57            }
58            productRepo.save(product);
59            return "Added to Cart";
60        }
61        return "out of stock";
62    }
63
64    public String checkUsernameAndPassword(String username, String password) {
65        AppUser user = this.userdao.findUserByUsernameAndPassword(username, password);
66        if (user != null) {
67            return user.getUsername();
68        }
69        return null;
70    }
71
72
73
74
75
76
77
78
79
80
81 }
```

```

82    public String addToWishList(WishList wishlist) {
83        if (wishlistItemDao
84            .getProducts(wishlist.getItems().get(0).getWishlistedProduct(), wishlist.getUsername())
85            .size() != 0) {
86            return "Already in your wishlist";
87        }
88        AppUser user = userdao.findByUsername(wishlist.getUsername());
89        user.getWishList().getWishListItems().add(wishlist.getItems().get(0));
90        user.getWishList().setUser(user);
91        user.getWishList().getWishListItems().forEach(theWishList -> theWishList.setUserWishlist(user.getWishList);
92        System.out.println(user.getWishList().getWishListItems() + "cart items");
93        userdao.saveAndFlush(user);
94
95        return "Added to wishlist";
96    }
97
98    public AppUser findUser(String username) {
99        return this.userdao.findByUsername(username);
100    }
101
102    public String deleteWishListItem(Long id) {
103        if (this.wishlistItemDao.existsById(id)) {
104            this.wishlistItemDao.deleteById(id);
105            return "deleted";
106        }
107        return "Item doesn't exist";
108    }
109
110    public String deleteCartItems(CartItems item) {
111        Long id = item.getId();
112        if (this.cartitemdao.existsById(id)) {
113            Product product = this.productRepo.findById(item.getCartProduct().getId()).orElse(null);
114            product.setSales(product.getSales() - (int) item.getQuantity());
115            ProductInventory inventory = product.getInventory();
116            inventory.setQuantity(inventory.getQuantity() + (int) item.getQuantity());
117            product.setInventory(inventory);
118            productRepo.save(product);
119            this.cartitemdao.deleteById(id);
120            return "deleted";
121        }
122        return "Item doesn't exist";
123    }
124
125    public List<AppUser> getAllUser() {
126        List<AppUser> users = userdao.findAll();
127        users.forEach(a->{
128            a.setPhoneNumber(null);
129            a.setPassword(null);
130            a.setCart(null);
131            a.setWishList(null);
132            a.setCreatedDt(null);
133            a.setUpdatedDt(null);
134        });
135        return users;
136    }
137
138    public String changeUserDetails(AppUser user) {
139        AppUser user2=this.userdao.findUserByUsernameAndPassword(user.getUsername(), user.getPassword());
140        if(user2!=null) {
141            user2.setAddress(user.getAddress());
142            user2.setPhoneNumber(user.getPhoneNumber());
143            user2.setFirstName(user.getFirstName());
144        }
145    }

```

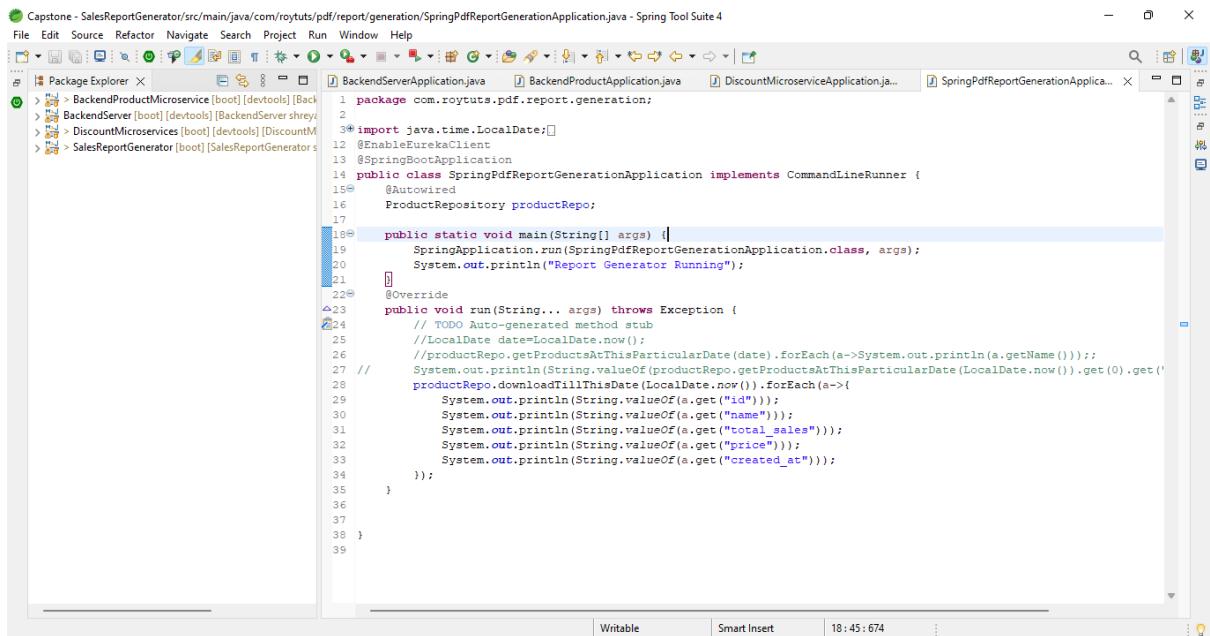
## Discount Microservice

```

1 package com.g1;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class DiscountMicroserviceApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(DiscountMicroserviceApplication.class, args);
10        System.out.println("Discount microservice running");
11    }
12
13
14 }
15

```

# Sales Report Generator



The screenshot shows the Spring Tool Suite 4 interface with the following details:

- Title Bar:** Capstone - SalesReportGenerator/src/main/java/com/roytuts/pdf/report/generation/SpringPdfReportGenerationApplication.java - Spring Tool Suite 4
- Menu Bar:** File Edit Source Refactor Navigate Project Run Window Help
- Toolbars:** Standard Java IDE toolbars.
- Left Sidebar:** Package Explorer showing projects: BackendProductMicroservice [boot] [devtools] [BackendServer [boot] [devtools] [BackendServer shrey], DiscountMicroservices [boot] [devtools] [DiscountM], SalesReportGenerator [boot] [SalesReportGenerat...].
- Central Editor:** The code editor displays the `SpringPdfReportGenerationApplication.java` file. The code implements a `CommandLineRunner` interface and overrides the `run` method to print sales data from a `ProductRepository`.

```
1 package com.roytuts.pdf.report.generation;
2
3 import java.time.LocalDate;
4
5 @EnableEurekaClient
6 @SpringBootApplication
7 public class SpringPdfReportGenerationApplication implements CommandLineRunner {
8     @Autowired
9     ProductRepository productRepo;
10
11     public static void main(String[] args) {
12         SpringApplication.run(SpringPdfReportGenerationApplication.class, args);
13         System.out.println("Report Generator Running");
14     }
15
16     @Override
17     public void run(String... args) throws Exception {
18         // TODO Auto-generated method stub
19         //LocalDate date=LocalDate.now();
20         //productRepo.getProductsAtThisParticularDate(date).forEach(a->System.out.println(a.getName()));
21         //System.out.println(String.valueOf(productRepo.getProductsAtThisParticularDate(LocalDate.now()).get(0).get('
22         productRepo.downloadTillThisDate(LocalDate.now()).forEach(a->
23             System.out.println(String.valueOf(a.get("id")));
24             System.out.println(String.valueOf(a.get("name")));
25             System.out.println(String.valueOf(a.get("total_sales")));
26             System.out.println(String.valueOf(a.get("price")));
27             System.out.println(String.valueOf(a.get("created_at")));
28         });
29     }
30
31     }
32
33 }
```

- Bottom Status Bar:** Writable | Smart Insert | 18:45:674