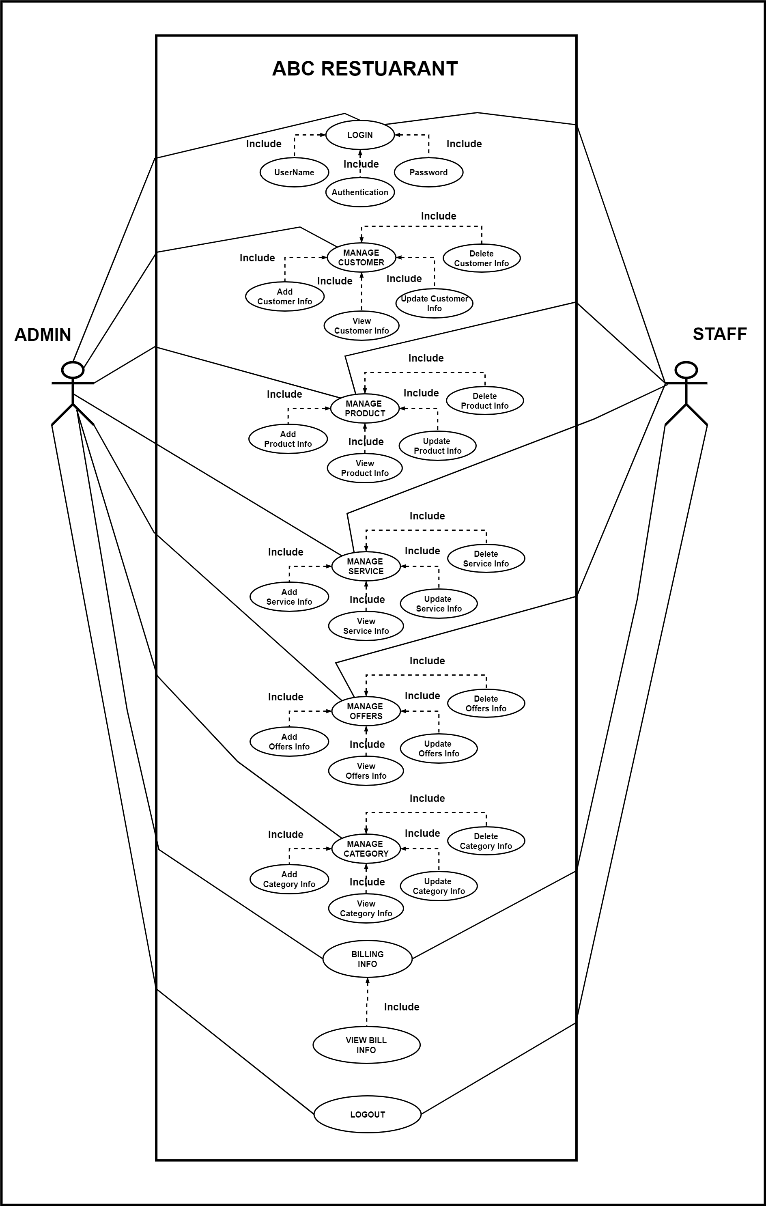
# **Acknowledgement**

In order to successfully complete the project, we reached out to several esteemed individuals for their guidance and assistance, and we are immensely grateful to them. We extend our heartfelt thanks to Lecture, the course instructor at the ICBT Campus, for providing us with invaluable assignment advice during multiple discussions. Furthermore, the constructive feedback received from our classmates, which greatly contributed to enhancing the project's quality, deserves special recognition.

# **Task A**

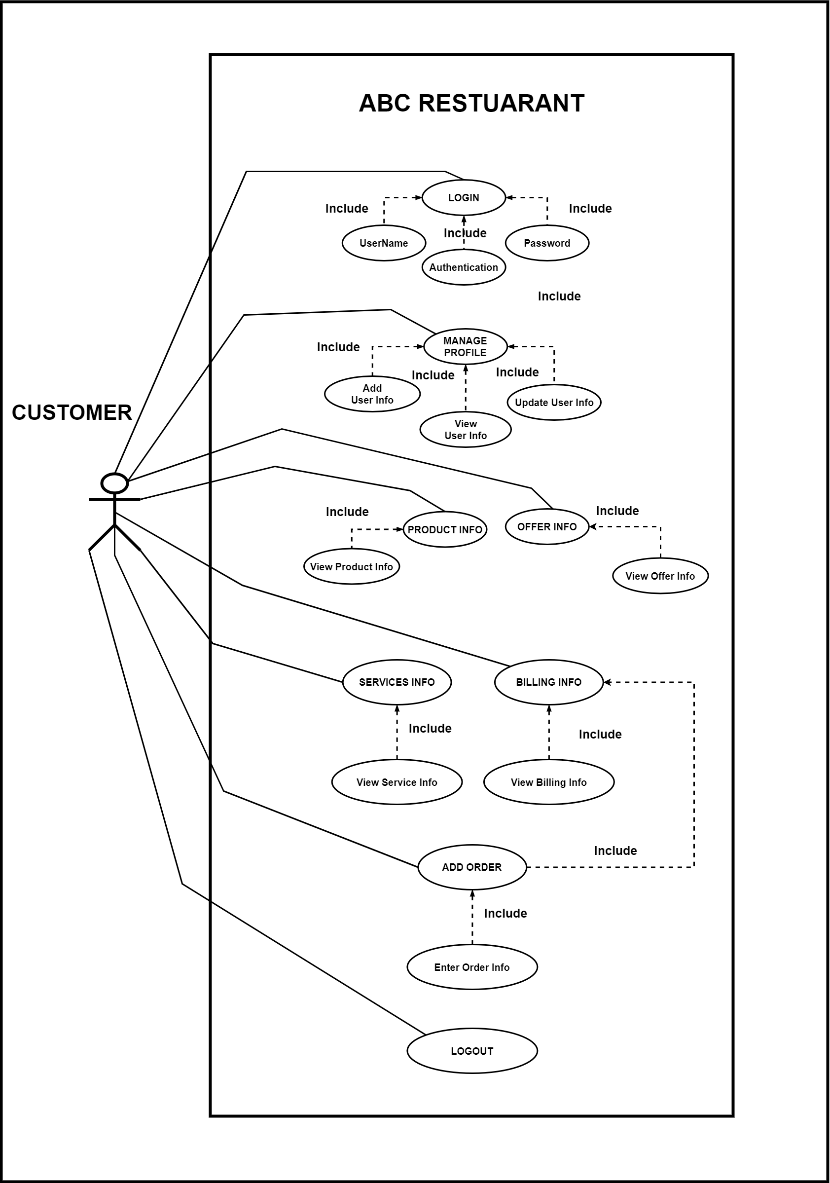
## **UML Diagrams**

### **Admin and Staff Use case Diagram**



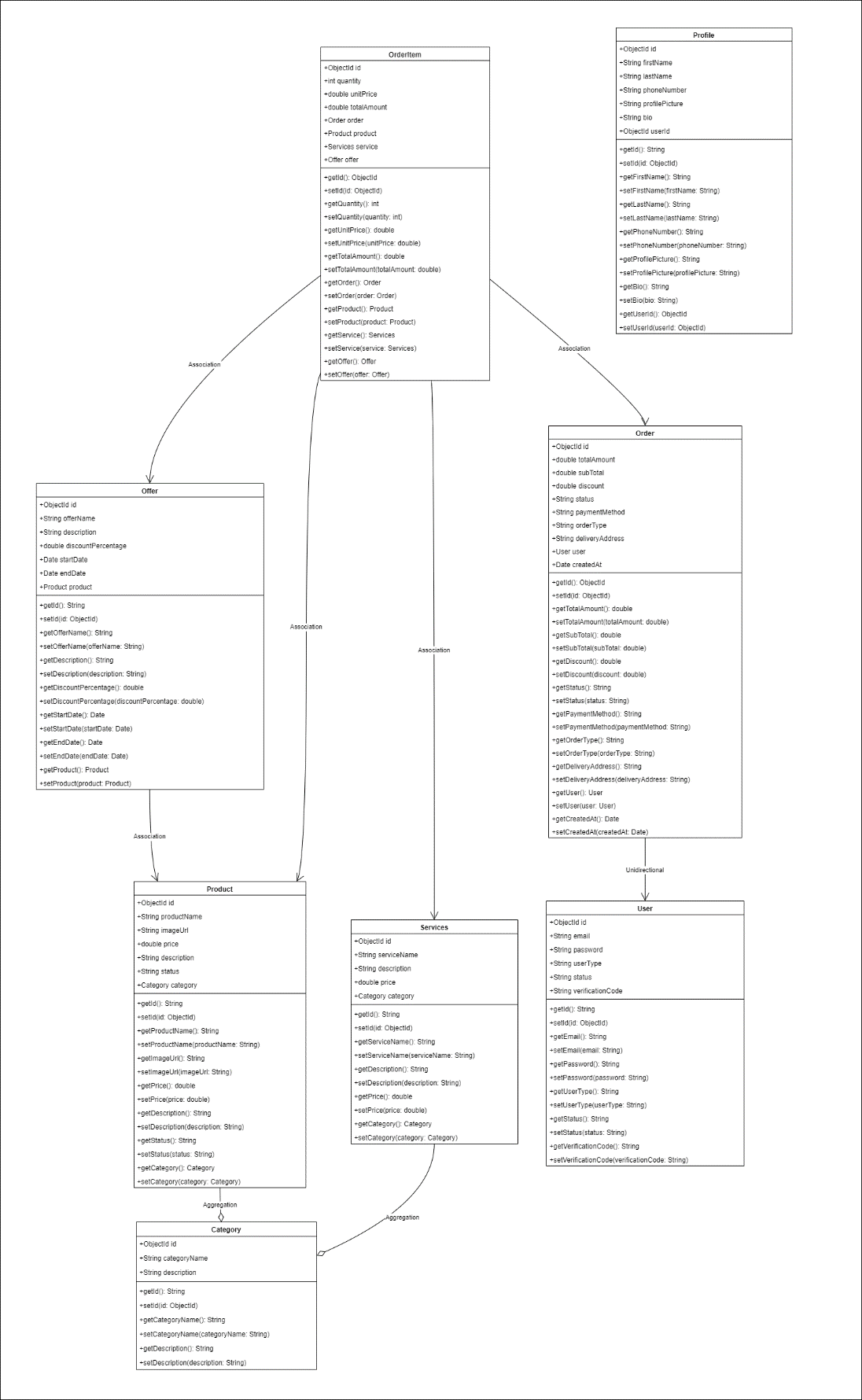
The use case graphic clearly illustrates the roles of Administration and Staff in the ABC Restaurant system. Each user has the option to verify payment details, change a number of system properties (such as customers, products, services, offers, and categories), and log in before logging out. The system is designed to enable both the staff and administrator to do their duties efficiently, within the confines of their respective access privileges.

### **Customer Use case Diagram**



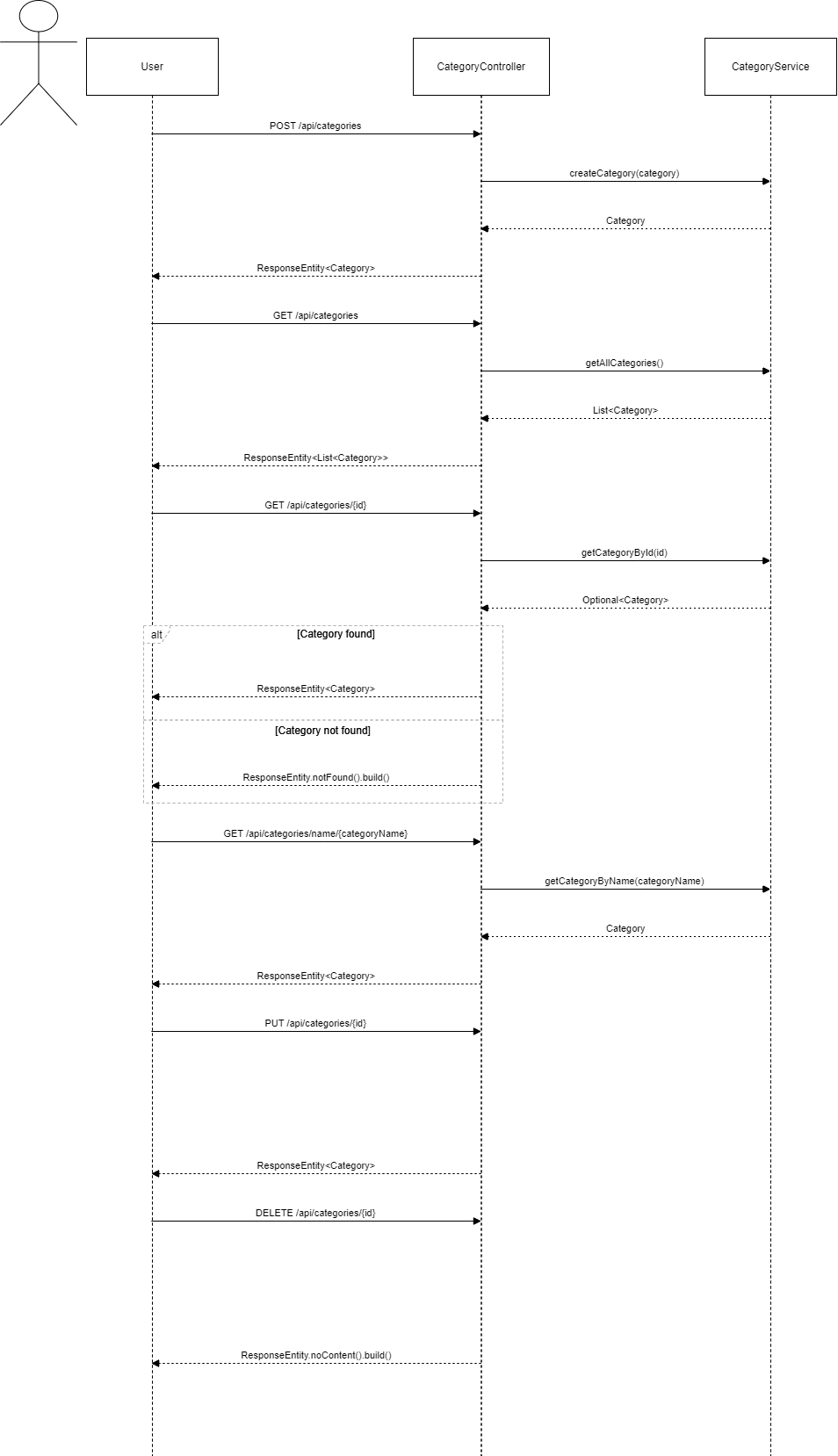
This use case diagram illustrates the many interactions a customer may have with the ABC Restaurant system. The illustration demonstrates how a customer makes purchases, verifies billing information, manages their profile, and views offers, products, and services. Ensuring the security of their experience, the user may access and depart the system with safety.

### **Class Diagram**

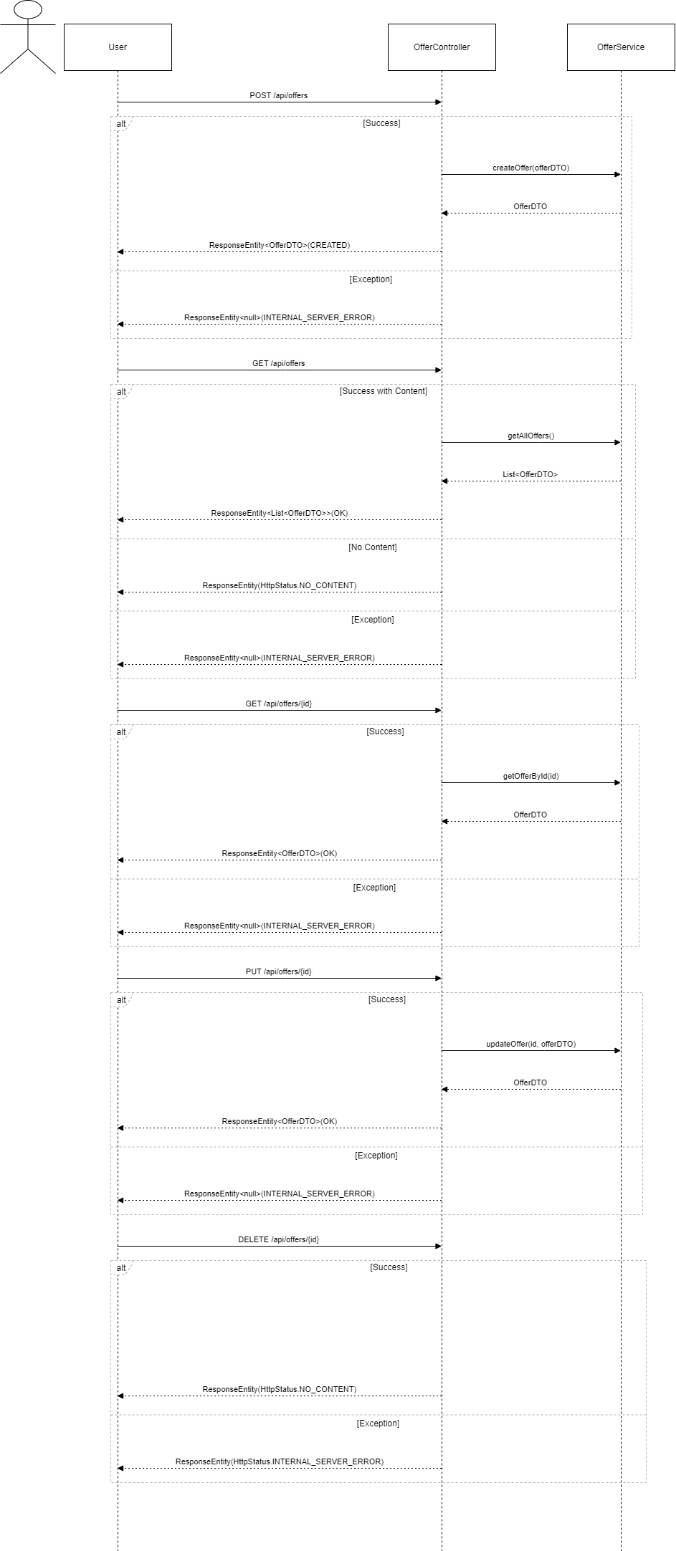


This class diagram, which illustrates the interactions between numerous entities including users, profiles, orders, items, services, deals, and categories, accurately captures the structure of the ABC Restaurant system. It provides a clear image of the attributes, functions, and relationships between each class, which facilitates understanding the system's architecture and data flow.

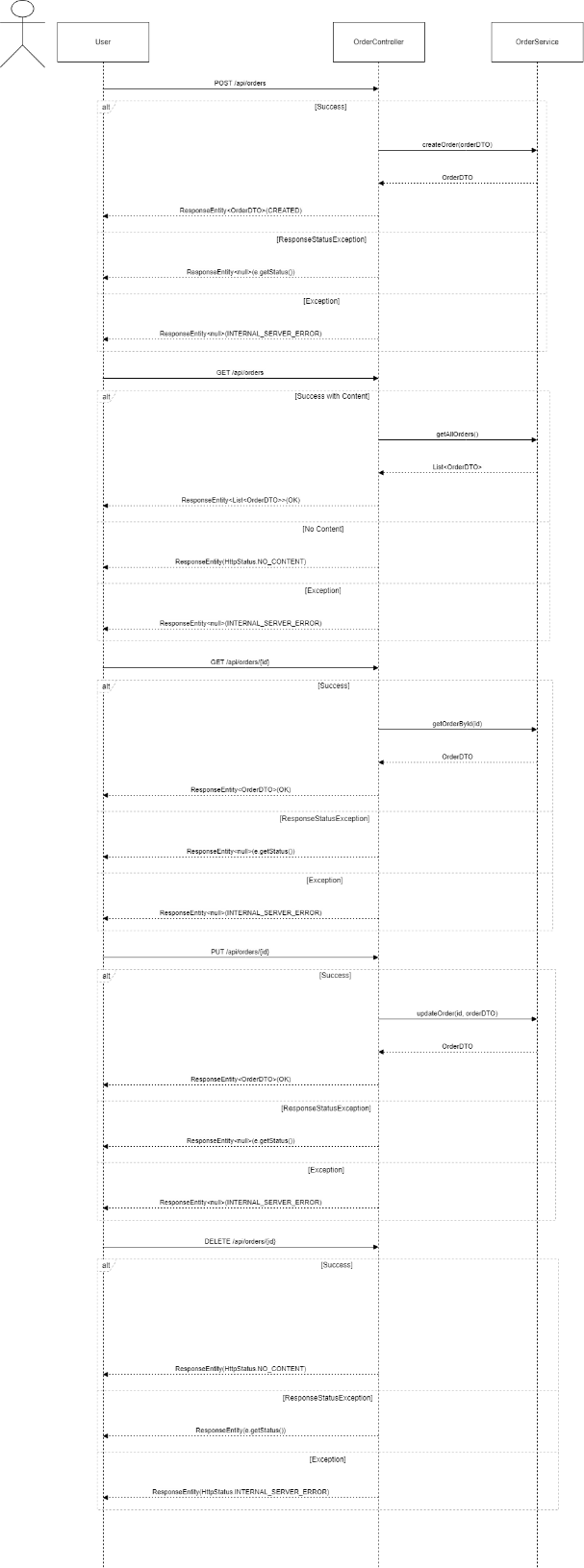
### **Category - Sequence Diagram**



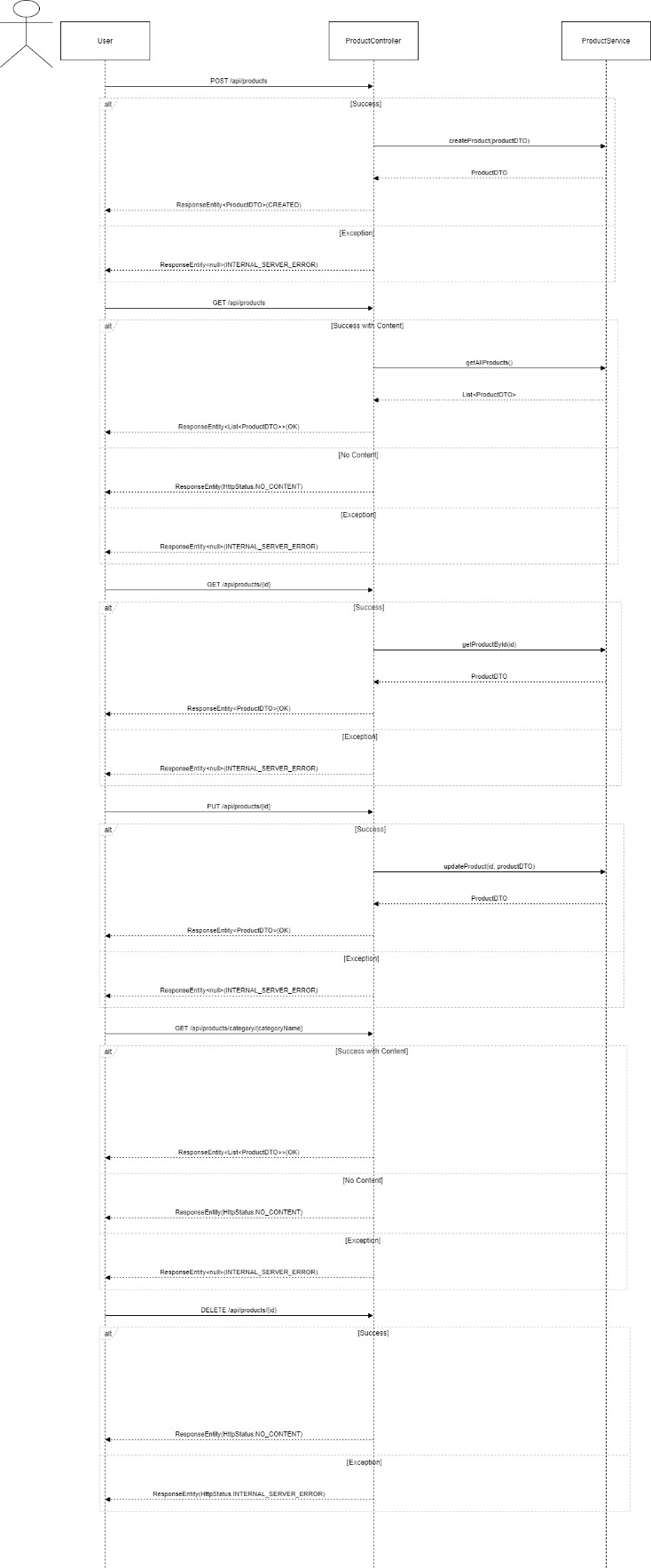
### **Offer - Sequence Diagram**



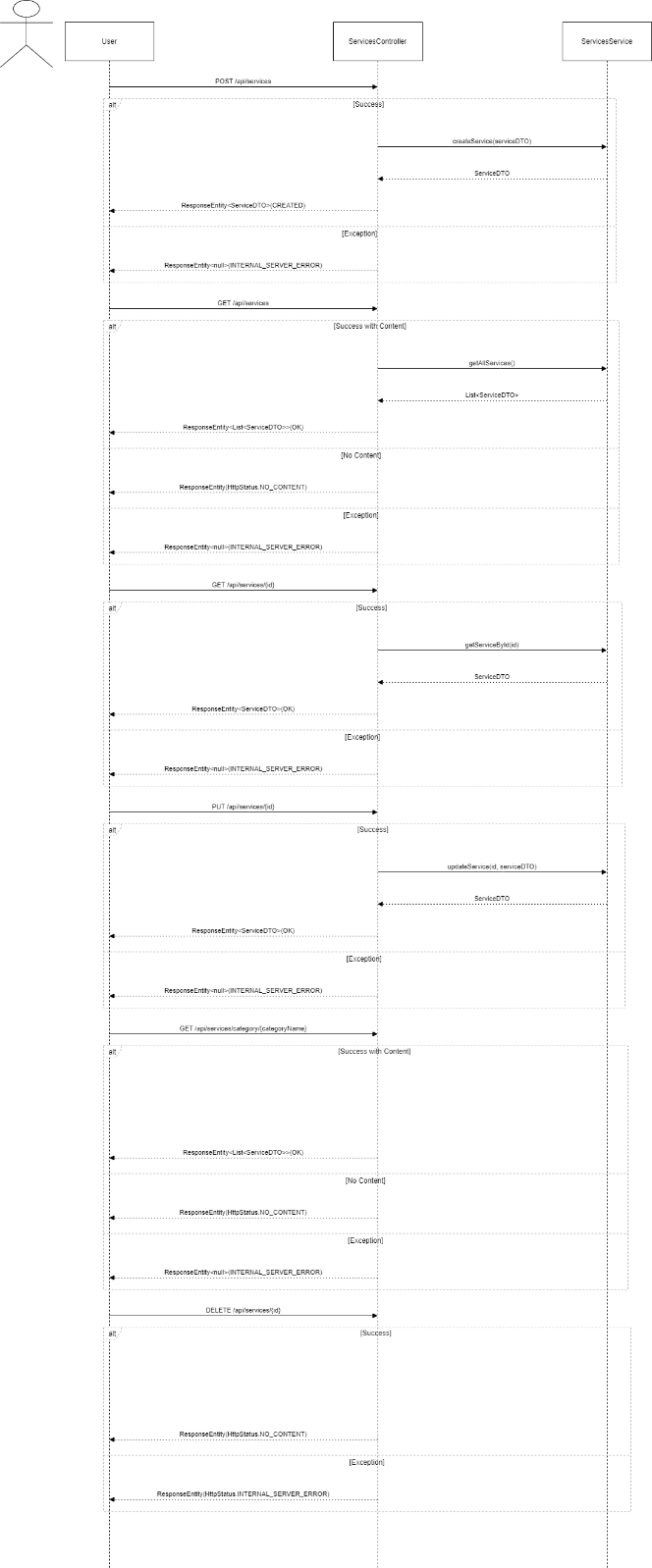
### **Order - Sequence Diagram**



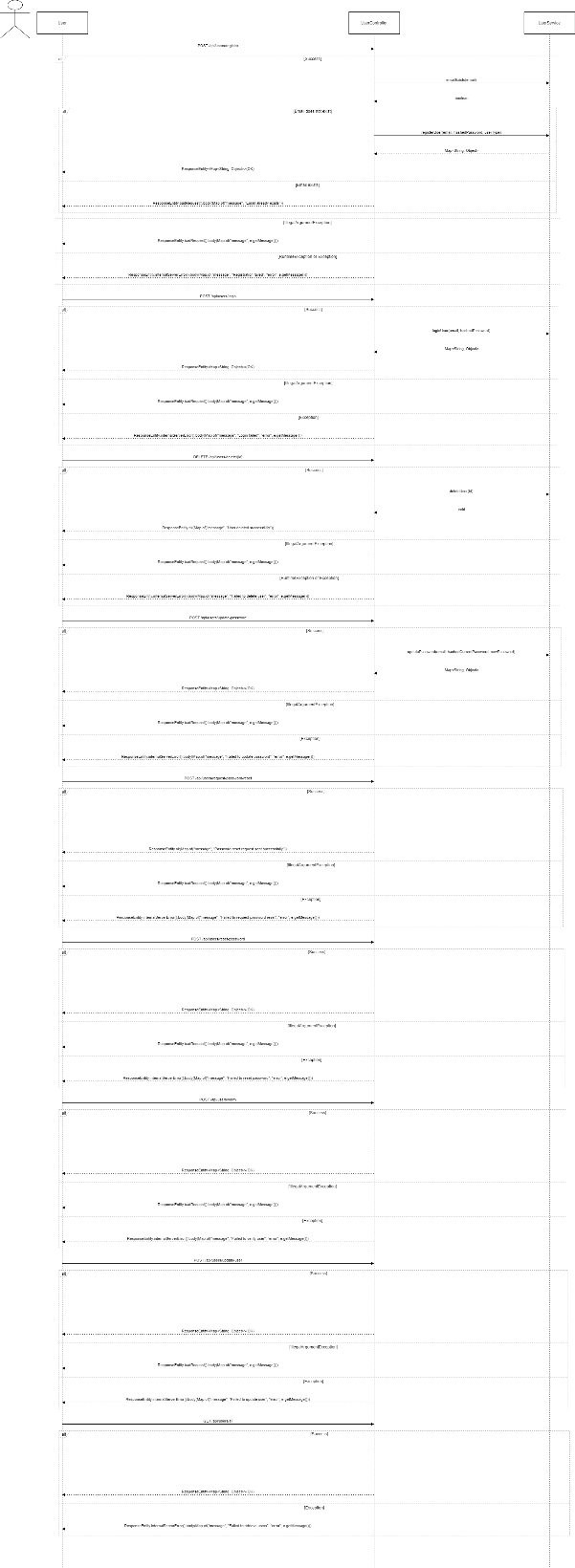
### **Product - Sequence Diagram**



### **Services - Sequence Diagram**



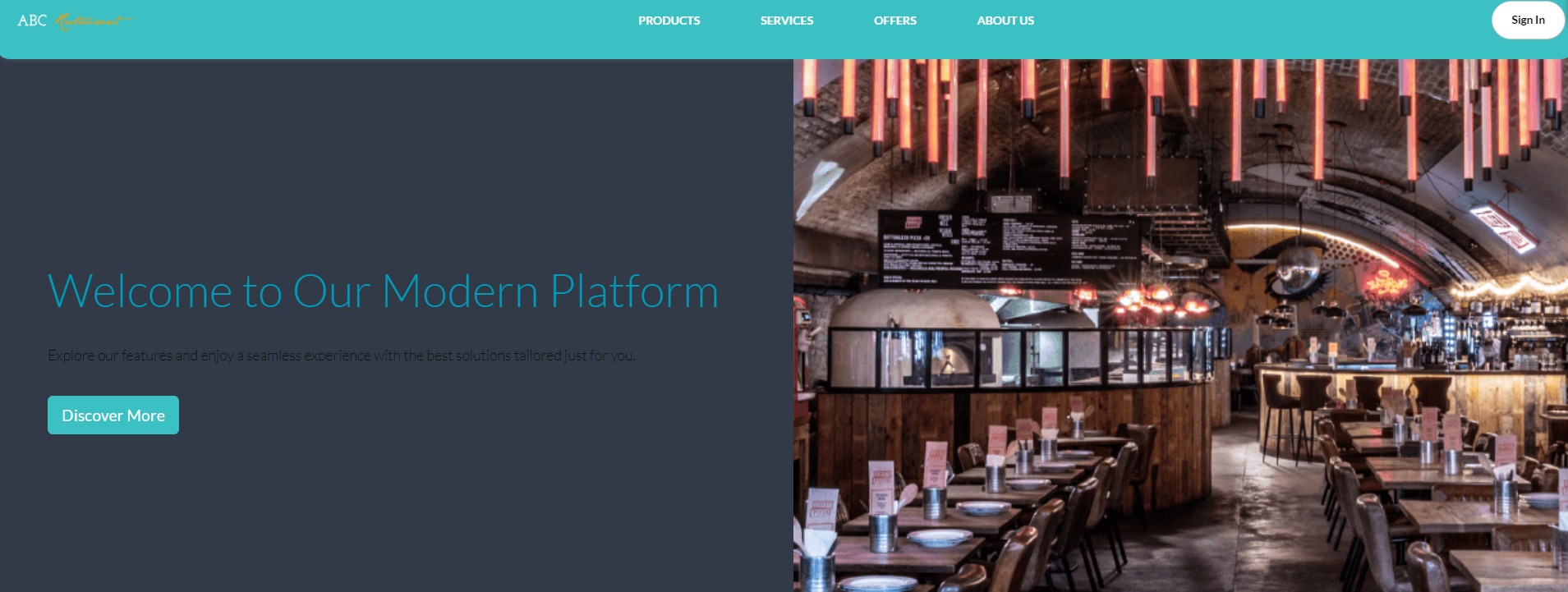
### **Users - Sequence Diagram**

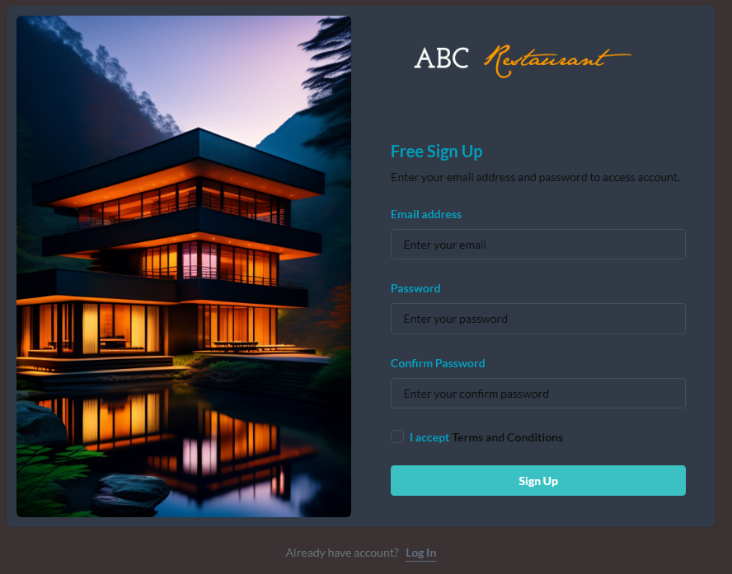


# **Task B**

**User Interfaces**

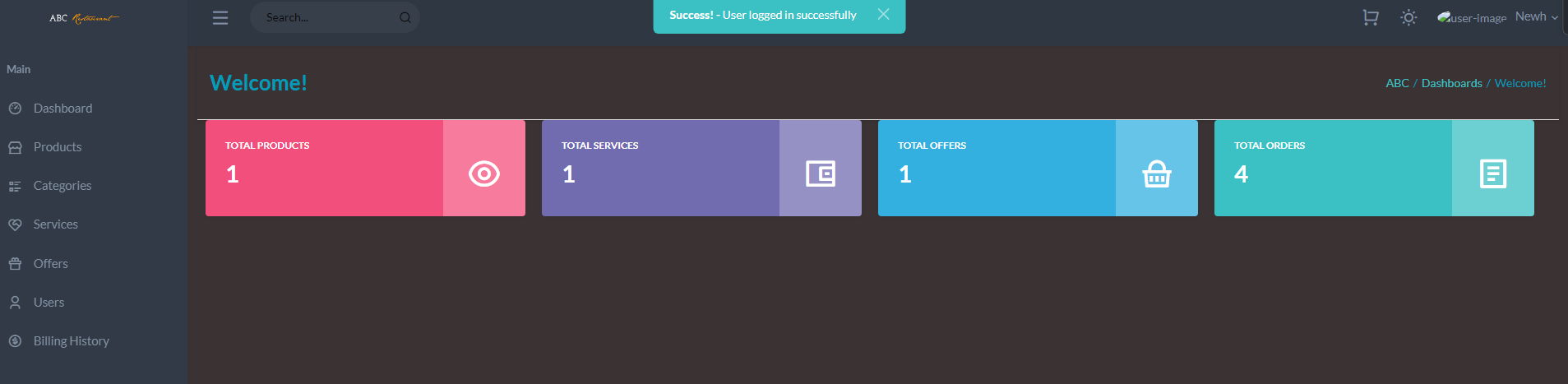
1. Home Page



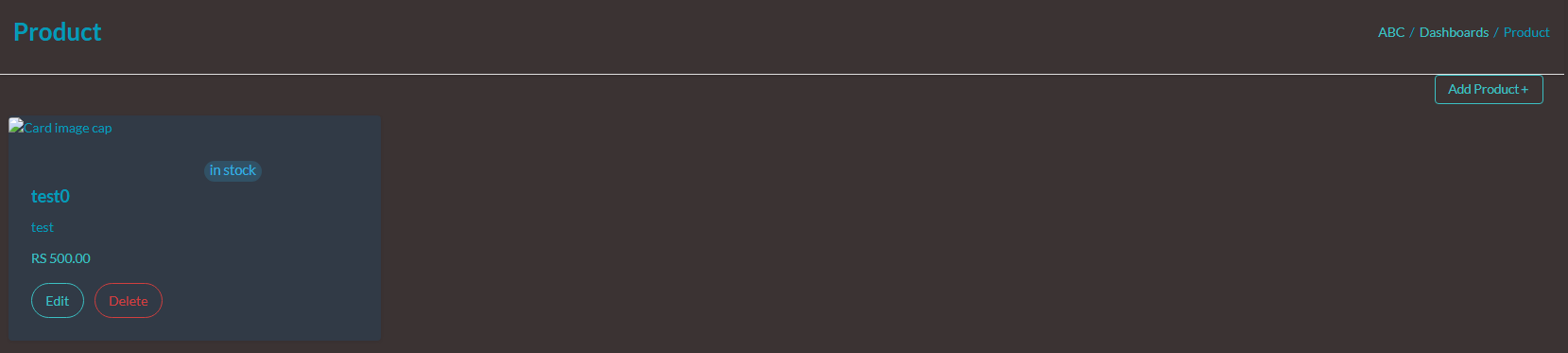
1. Sign In / Sign Up



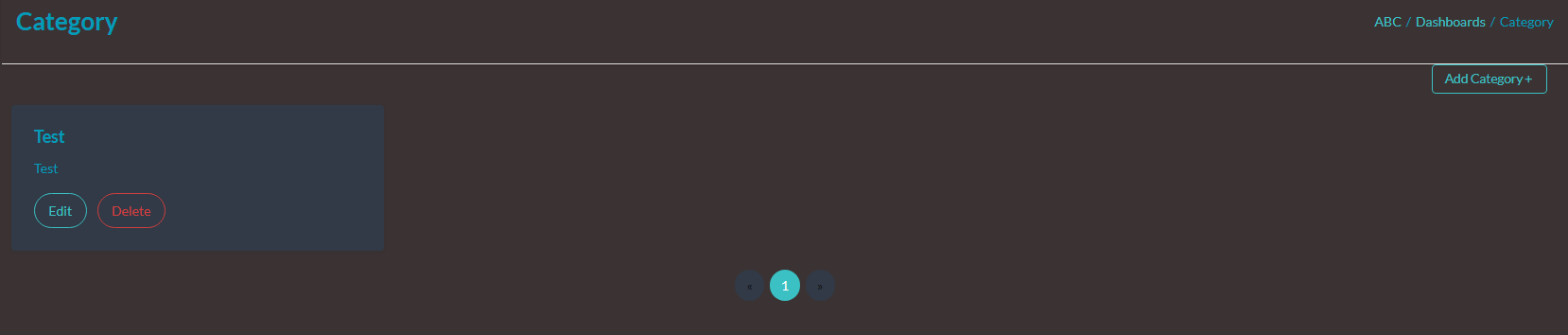
1. Admin Dashboard



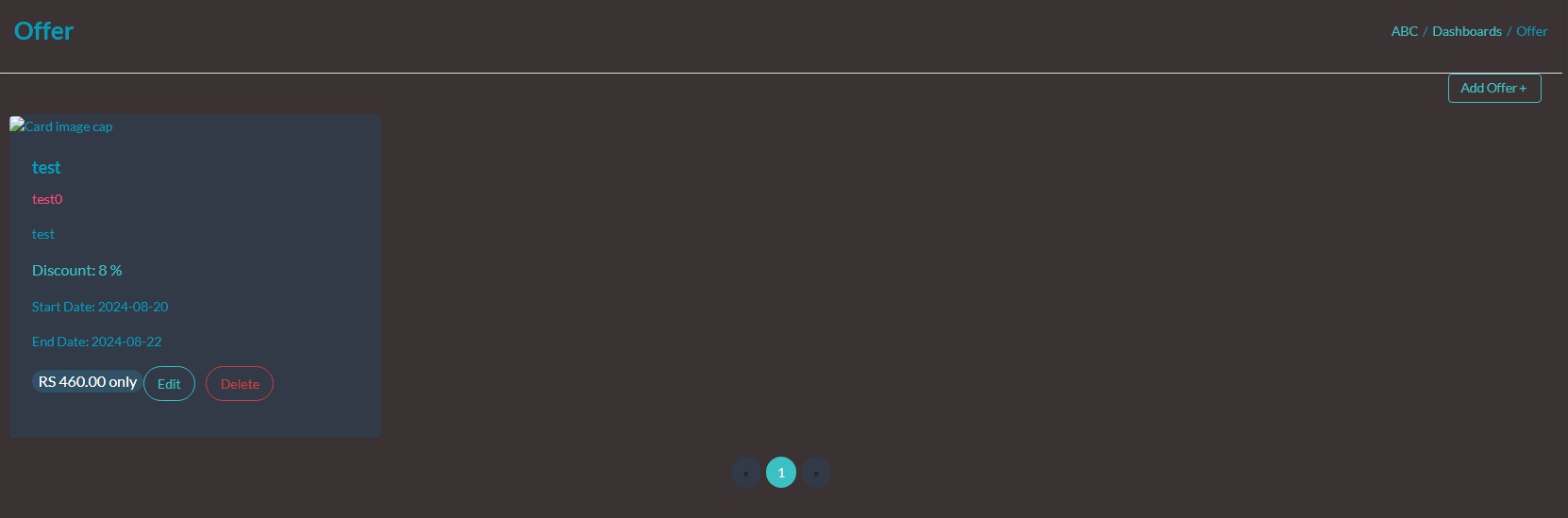
1. Admin Manage Product



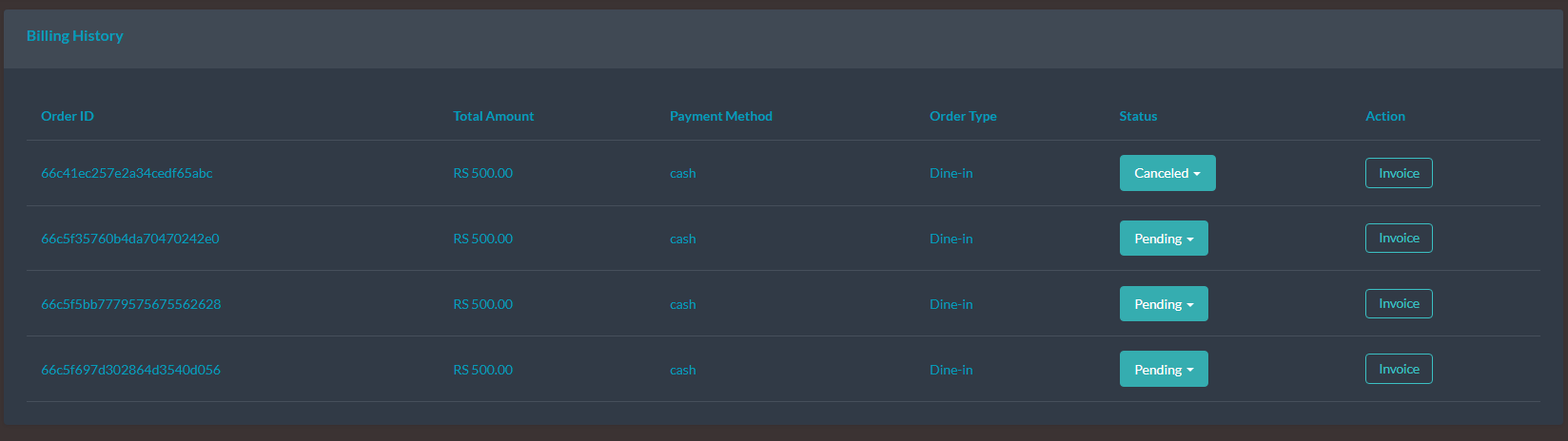
1. Admin Manage Category



1. Admin Manage Offer



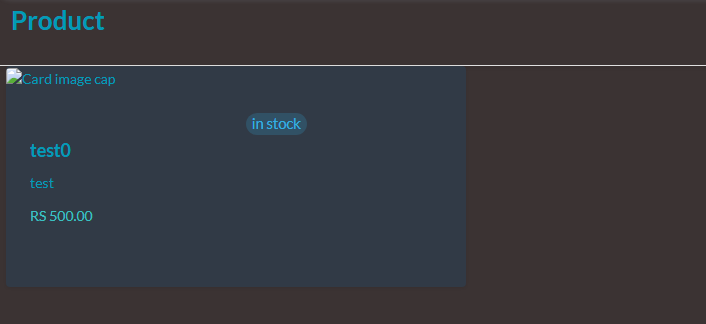
1. Admin Manage Billing History



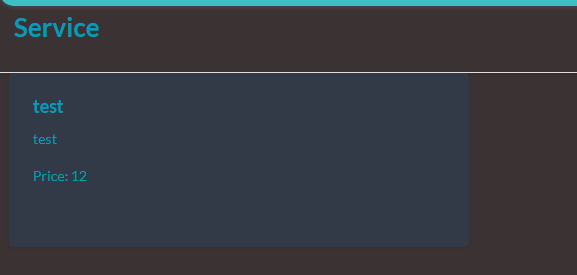
1. Staff Dashboard



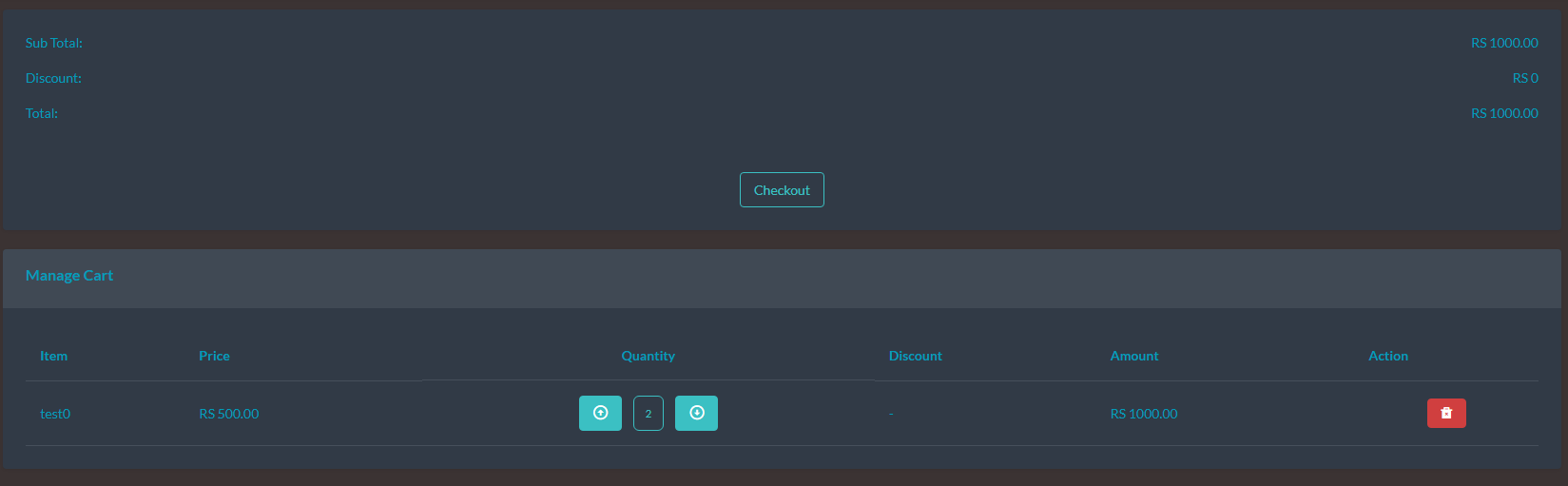
1. View Product



1. View Service



1. User Cart

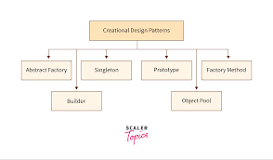


## **Design Patterns**

Well-tested answers to common issues in software development are found in design patterns. These practical solutions, which have been carefully crafted and thoroughly examined by experts in software design, provide programmers effective ways to deal with design challenges in a timely and effective manner. Code that is arranged in accordance with these design principles is simpler to reuse, adapt, and maintain. These patterns may lead developers away from common mistakes and risks, which can enhance the quality, stability, and maintainability of software. There are several design pattern substitutes, each with a unique set of benefits and applications.(Javatpoint, 2019)

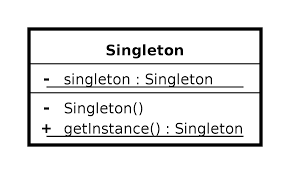
### **Creational design patterns**

Product creation processes are the main focus of the area of creational design patterns. The main goal of these patterns is to guarantee that object creation complies with the specified limitations. Moreover, they provide developers techniques that promote adaptability and reuse while building objects.

. 

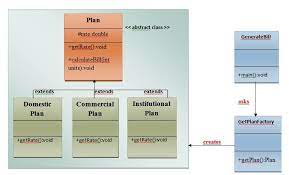
### **The singleton patterns**

When a single object has to coordinate events throughout the whole system, singleton patterns guarantee that a class only produces one instance. By creating a single point of access, this method makes it easier to coordinate system-wide actions.



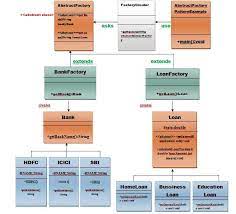
### **The factory patterns**

Subclasses may still choose which class to instantiate, and users can utilise factory patterns to create objects. This technique enables for the delegation of object instantiation chores to several subclasses, which is extremely helpful when a large number of objects need to be produced with various parameters or settings.



### **The abstract factory patterns**

The creation of connected or dependent families of objects is facilitated by abstract patterns, which eliminate the need to explicitly define each class that makes up the family. This method makes it easier to generate interconnected items without needing to know the exact type of each object, which is helpful when many related things need to be made at once.



### **The Factory Design Pattern**

One of the most popular design patterns in the Java programming community is the factory design pattern, sometimes referred to as the factory method. Despite the fact that this pattern "defines an interface to construct an object," as is often said, it is important to realise that subclasses are free to choose the class they want to instantiate. Furthermore, a class deferral mechanism may be used to extend the Factory process. A specific factory class is designated by this design pattern to generate an object of the "Object () [native code]" type on the client's behalf. We rely on this factory to create the objects in this scenario while concealing the difficulties involved in putting them into practice. Using a well defined interface is essential to getting access to produced objects.

(Tutorialspoint, 2022)

# **Task C**

## **Test Plan and test-driven development**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Test Case Name | Scenario | Expected Result |
| TC001 | Login - Empty Field Validation | User attempts to log in with empty email or password fields. | System displays error message: "Email and Password are required." |
| TC002 | Login - Invalid Email Format | User attempts to log in with an invalid email format. | System displays error message: "Invalid email format." |
| TC003 | Signup - Empty Field Validation | User attempts to sign up with empty required fields. | System displays error message: "All fields are required." |
| TC004 | Signup - Invalid Email Format | User attempts to sign up with an invalid email format. | System displays error message: "Invalid email format." |
| TC005 | Signup - Password Mismatch | User attempts to sign up with mismatched password and confirm password. | System displays error message: "Passwords do not match." |
| TC006 | Forget Password - Empty Field | User attempts to reset the password without entering an email address. | System displays error message: "Email is required." |
| TC007 | Forget Password - Invalid Email | User attempts to reset the password with an invalid email format. | System displays error message: "Invalid email format." |
| TC008 | Email Verification - Invalid Link | User clicks on an invalid or expired email verification link. | System displays error message: "Invalid or expired link." |
| TC009 | Email Verification - Successful | User clicks on a valid email verification link. | System verifies the email and updates the user status to verified. |
| TC010 | CRUD Products - Admin | Admin adds, updates, or deletes a product. | Product is successfully added, updated, or deleted. |
| TC011 | CRUD Categories - Admin | Admin adds, updates, or deletes a category. | Category is successfully added, updated, or deleted. |
| TC012 | CRUD Service - Admin | Admin adds, updates, or deletes a service. | Service is successfully added, updated, or deleted. |
| TC013 | CRUD Offers - Admin | Admin adds, updates, or deletes an offer. | Offer is successfully added, updated, or deleted. |
| TC014 | CRUD Users - Admin | Admin adds, updates, or deletes a user. | User is successfully added, updated, or deleted. |
| TC015 | View Billing History - Admin | Admin views the billing history of users. | Billing history is successfully displayed. |
| TC016 | Edit Profile - Admin | Admin edits their profile information. | Profile information is successfully updated. |
| TC017 | CRUD Products - Staff | Staff adds, updates, or deletes a product. | Product is successfully added, updated, or deleted. |
| TC018 | CRUD Service - Staff | Staff adds, updates, or deletes a service. | Service is successfully added, updated, or deleted. |
| TC019 | CRUD Offers - Staff | Staff adds, updates, or deletes an offer. | Offer is successfully added, updated, or deleted. |
| TC020 | View Billing History - Staff | Staff views the billing history of users. | Billing history is successfully displayed. |
| TC021 | View Products - Users | Users view the list of available products. | Products are successfully displayed to the user. |
| TC022 | View Services - Users | Users view the list of available services. | Services are successfully displayed to the user. |
| TC023 | View Offers - Users | Users view the list of available offers. | Offers are successfully displayed to the user. |
| TC024 | View Billing History - Users | Users view their billing history. | Billing history is successfully displayed to the user. |
| TC025 | Edit Profile - Users | Users edit their profile information. | Profile information is successfully updated. |

## **Test Cases**

|  |  |
| --- | --- |
| Test Case ID | TC001 |
| Test Objective | Verify login with empty fields |
| Test Scenario | User attempts to log in without entering any data |
| Text Input | “” |
| Expected Result | System should display an error message: "Email and Password are required." |

|  |  |
| --- | --- |
| Test Case ID | TC002 |
| Test Objective | Verify login with invalid email format |
| Test Scenario | User attempts to log in with an invalid email format |
| Text Input | tharindu.perera@invalid |
| Expected Result | System should display an error message: "Invalid email format." |

|  |  |
| --- | --- |
| Test Case ID | TC003 |
| Test Objective | Verify signup with empty fields |
| Test Scenario | User attempts to sign up without entering any data |
| Text Input | “” |
| Expected Result | System should display an error message: "All fields are required." |

|  |  |
| --- | --- |
| Test Case ID | TC004 |
| Test Objective | Verify signup with valid data |
| Test Scenario | User signs up with all valid information |
| Text Input | Nimali Jayasundara, nimali.jaya@gmail.com, Password123! |
| Expected Result | System should create a new account and redirect to the dashboard. |

|  |  |
| --- | --- |
| Test Case ID | TC005 |
| Test Objective | Verify password reset with empty email |
| Test Scenario | User attempts to reset password without entering an email |
| Text Input | “” |
| Expected Result | System should display an error message: "Email is required." |

|  |  |
| --- | --- |
| Test Case ID | TC006 |
| Test Objective | Verify email verification with invalid link |
| Test Scenario | User clicks on an invalid or expired email verification link |
| Text Input |  |
| Expected Result | System should display an error message: "Invalid or expired link." |

|  |  |
| --- | --- |
| Test Case ID | TC007 |
| Test Objective | Verify email verification with valid link |
| Test Scenario | User clicks on a valid email verification link |
| Text Input |  |
| Expected Result | System should verify the email and update the user status to verified. |

|  |  |
| --- | --- |
| Test Case ID | TC008 |
| Test Objective | Admin - Add Product |
| Test Scenario | Admin adds a new product with all required information |
| Text Input | Product Name: Laptop Bag, Category: Accessories |
| Expected Result | Product should be successfully added to the inventory. |

|  |  |
| --- | --- |
| Test Case ID | TC009 |
| Test Objective | Admin - Update Product |
| Test Scenario | Admin updates an existing product |
| Text Input | Update Product Name from "Laptop Bag" to "Premium Laptop Bag" |
| Expected Result | Product information should be successfully updated. |

|  |  |
| --- | --- |
| Test Case ID | TC010 |
| Test Objective | Admin - Delete Product |
| Test Scenario | Admin deletes an existing product |
| Text Input | Select Product: Premium Laptop Bag |
| Expected Result | Product should be successfully deleted from the inventory. |

|  |  |
| --- | --- |
| Test Case ID | TC011 |
| Test Objective | Admin - Add Category |
| Test Scenario | Admin adds a new category |
| Text Input | Category Name: Electronics |
| Expected Result | Category should be successfully added to the system. |

|  |  |
| --- | --- |
| Test Case ID | TC012 |
| Test Objective | Admin - Update Category |
| Test Scenario | Admin updates an existing category |
| Text Input | Update Category Name from "Electronics" to "Electronic Gadgets" |
| Expected Result | Category information should be successfully updated. |

|  |  |
| --- | --- |
| Test Case ID | TC013 |
| Test Objective | Admin - Delete Category |
| Test Scenario | Admin deletes an existing category |
| Text Input | Select Category: Electronic Gadgets |
| Expected Result | Category should be successfully deleted from the system. |

|  |  |
| --- | --- |
| Test Case ID | TC014 |
| Test Objective | Admin - Add Service |
| Test Scenario | Admin adds a new service with all required information |
| Text Input | Service Name: Installation |
| Expected Result | Service should be successfully added to the system. |

|  |  |
| --- | --- |
| Test Case ID | TC015 |
| Test Objective | Admin - Update Service |
| Test Scenario | Admin updates an existing service |
| Text Input | Update Service Name from "Installation" to "Home Installation" |
| Expected Result | Service information should be successfully updated. |

|  |  |
| --- | --- |
| Test Case ID | TC016 |
| Test Objective | Admin - Delete Service |
| Test Scenario | Admin deletes an existing service |
| Text Input | Select Service: Home Installation |
| Expected Result | Service should be successfully deleted from the system. |

|  |  |
| --- | --- |
| Test Case ID | TC017 |
| Test Objective | Admin - Add Offer |
| Test Scenario | Admin adds a new offer with all required information |
| Text Input | Offer Name: Summer Sale |
| Expected Result | Offer should be successfully added to the system. |

|  |  |
| --- | --- |
| Test Case ID | TC018 |
| Test Objective | Admin - Update Offer |
| Test Scenario | Admin updates an existing offer |
| Text Input | Update Offer Name from "Summer Sale" to "Summer Mega Sale" |
| Expected Result | Offer information should be successfully updated. |

|  |  |
| --- | --- |
| Test Case ID | TC019 |
| Test Objective | Admin - Delete Offer |
| Test Scenario | Admin deletes an existing offer |
| Text Input | Select Offer: Summer Mega Sale |
| Expected Result | Offer should be successfully deleted from the system. |

|  |  |
| --- | --- |
| Test Case ID | TC020 |
| Test Objective | Admin - Add User |
| Test Scenario | Admin adds a new user with all required information |
| Text Input | Name: Samantha Silva, Email: samantha.silva@gmail.com |
| Expected Result | User should be successfully added to the system. |

|  |  |
| --- | --- |
| Test Case ID | TC021 |
| Test Objective | Admin - Update User |
| Test Scenario | Admin updates an existing user |
| Text Input | Update Name from "Samantha Silva" to "Samantha Perera" |
| Expected Result | User information should be successfully updated. |

|  |  |
| --- | --- |
| Test Case ID | TC022 |
| Test Objective | Admin - Delete User |
| Test Scenario | Admin deletes an existing user |
| Text Input | Select User: Samantha Perera |
| Expected Result | User should be successfully deleted from the system. |

|  |  |
| --- | --- |
| Test Case ID | TC023 |
| Test Objective | View Billing History - Admin |
| Test Scenario | Admin views the billing history of users |
| Text Input |  |
| Expected Result | Billing history should be successfully displayed. |

|  |  |
| --- | --- |
| Test Case ID | TC024 |
| Test Objective | Edit Profile - Admin |
| Test Scenario | Admin edits their profile information |
| Text Input | Name: Tharindu Perera, Email: tharindu.perera@gmail.com |
| Expected Result | Profile information should be successfully updated. |

|  |  |
| --- | --- |
| Test Case ID | TC025 |
| Test Objective | Login - Successful |
| Test Scenario | User logs in with correct email and password |
| Text Input | Email: kavinda.fernando@gmail.com, Password: SecurePass123 |
| Expected Result | User should be successfully logged in and redirected to the dashboard. |

# **Task D**

## **GitHub Repository**

* GitHub Link -

Repo Image

## **Code Explanation**

**Category Controller**

****

The primary logic for managing categories is included in the CategoryService, which the controller utilises to carry out various category-related operations. When a new category has to be created, the createCategory method—annotated with @PostMapping—is used. This function receives the Category object from the request, creates the category, processes it, and provides a 200 (OK) response. There are two ways you may get categories: A single category is obtained by its ID using the getCategoryById method, which also makes use of @GetMapping. If the category is discovered, a 200 response is returned; if not, a 404 (Not discovered) reply is returned. A list of all categories is returned by the getAllCategories function, which is marked with @GetMapping. To obtain categories by name, you can also use the getCategoryByName method, which is mapped using @GetMapping and produces a 200 response. An existing category may be modified using the updateCategory function, which is indicated by @PutMapping. It returns the modified category with a 200 status after receiving the ID and the changed data as input. Last but not least, the @DeleteMapping-related deleteCategory function allows a category to be removed by its ID. The deletion was successful, and no content was returned, according to the 204 (No material) result that is returned.

**Image Controller**

****

The uploadImage method, which is designated with @PostMapping("/upload"), is used to handle image uploads. This method accepts a MultipartFile from the client and makes use of the imageService.UploadImage(file) should be used to finish the upload. Upon successful completion, the filename and a 200 (OK) status are sent back. In the event of an error, the method logs the 500 (Internal Server Error) status, logs the exception, and returns an error message. On the other hand, the retrieveImage method may be used to retrieve an image by its filename; this can be expressed as @GetMapping("/retrieve/{fileName}"). If the image is successfully located, a 200 (OK) status and the appropriate JPEG format image data are provided. If there is a problem or the image cannot be found, the method returns 404 (Not located).

**Offer Controller**



The createOffer method, mapped to @PostMapping, is responsible for creating a new offer using the provided OfferDTO. If creation is successful, it returns the produced offer along with a 201 (generated) status. In the event that problems occur throughout the process, the method returns a 500 status (Internal Server Error). The getAllOffers method acquires all of the available offers by using @GetMapping. If the retrieval is successful, the offers are returned with a 200 (OK) status; otherwise, a 204 (No Content) result is returned. The status is 500 if there are any problems throughout this procedure. To get an offer by its ID, use the getOfferById method, which is also mapped to @GetMapping. If the retrieval is successful, the method returns the offer with a 200 (OK) status; if there are any problems, it returns a 500 status. To update an existing offer, use the updateOffer method (annotated with @PutMapping) with the provided ID and OfferDTO. The updated offer is provided with a 200 (OK) status when an update is successful, and a 500 status is returned in the event of a problem. Finally, the @DeleteMapping-connected deleteOffer method is used to remove an offer based on its ID. If the deletion is successful, a 204 (No Content) status is issued; if there are any problems throughout the process, a 500 status is given.

**Order Controller**

****

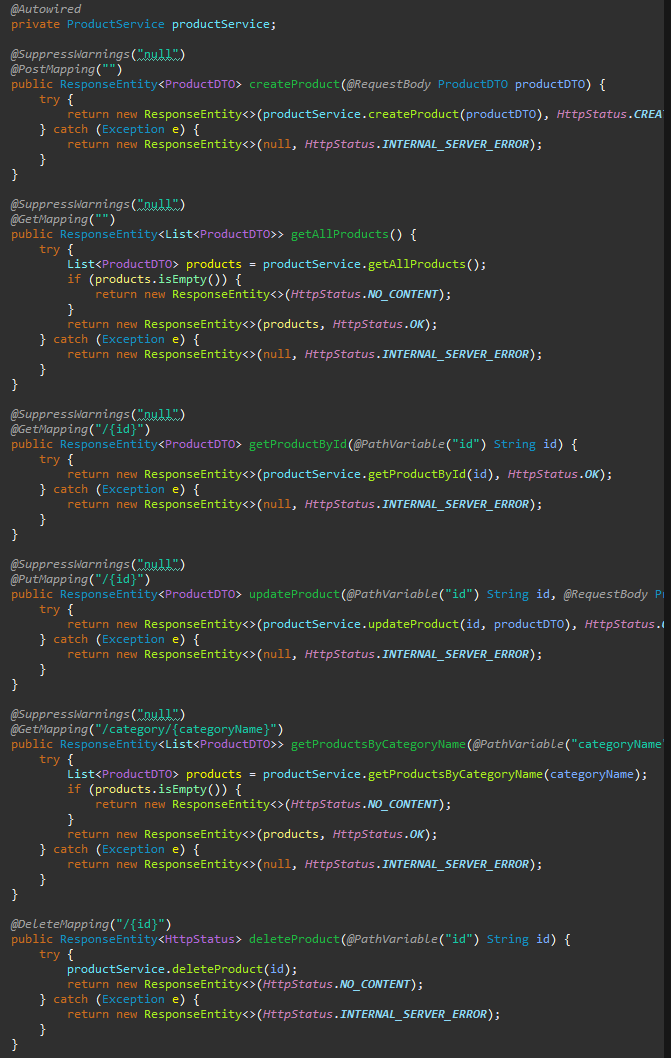
The createOrder function, marked with @PostMapping, is meant to create a new order using the provided OrderDTO. If creation is successful, it returns the created order with a 201 (Created) status. When a specific problem occurs, such a ResponseStatusException, the method returns the related error status. If there are any more unanticipated errors, a 500 (Internal Server Error) status is shown. The getAllOrders method, which is mapped to @GetMapping, is used to obtain all orders. If no orders are discovered, a 204 (No Content) status is returned; if orders are located, a 200 (OK) status is issued. The status is 500 if there are any problems throughout this procedure. Using the getOrderById method, which is also linked to @GetMapping, an order may be obtained by ID. If the method is successful, it returns the order with a 200 (OK) status. A 500 status is caused by other issues; if a ResponseStatusException is raised, the specific error status is returned. The @PutMapping updateOrder method updates an existing order using the provided ID and OrderDTO. If the modification is successful, the modified order is returned with a 200 (OK) response. Unexpected errors produce a 500 status; a ResponseStatusException raises an exception and returns the given error status. Finally, the deleteOrder method, mapped to @DeleteMapping, may be used to remove an order by its ID. If the deletion is successful, a 204 (No Content) answer is sent back. In the case of a ResponseStatusException, the function returns the specific error status; other problems get a 500 status.

**Order Item Controller**

****

Using the supplied OrderItemDTO, the createOrderItem method, mapped to @PostMapping, is in charge of producing a new order item. The method returns the generated item with a 201 (generated) status once creation is successful. A 500 (Internal Server Error) status is issued in the case of an error. All order items are retrieved using the @GetMapping-related getAllOrderItems function. A 204 (No Content) status is returned if no items are detected, while a 200 (OK) status is given if things are found. If there are any issues throughout this operation, the status is 500. To obtain an order item by its ID, use the getOrderItemById function, which is also mapped to @GetMapping. The method returns the item with a 200 (OK) status if the retrieval is successful; failures are handled by returning a 500 status. An order item may be deleted by its ID using the @DeleteMapping-linked deleteOrderItem function. Any failures result in a 500 status, but successful deletions return 204 (No Content). Furthermore, all order items connected to a certain order ID are deleted using the deleteOrderItemsByOrderId function, which is mapped to @DeleteMapping as well. The method returns a 204 (No Content) status if it is successful; faults result in a 500 status. Last but not least, the @GetMapping function getOrderItemsByOrderId obtains every order item connected to a given order ID. A 204 (No Content) status is returned if no items are detected, while a 200 (OK) status is given if things are found. If there are any issues throughout this operation, the status is 500.

**Product Controller**

****

The createProduct function, mapped to @PostMapping, is responsible for creating a new product using the provided ProductDTO. If creation is successful, it returns the newly created product with a 201 (formed) status. In the event of an error, the function returns a 500 (Internal Server Error) status. Using the getAllProducts method connected to @GetMapping, all accessible commodities are fetched. In the event that no goods are identified, it returns a 200 (OK) value; otherwise, it returns a 204 (No Content) status. A 500 status is issued in the event that there are any issues with this retrieval process. Use the getProductById method, which is mapped to @GetMapping as well, to get a product by its ID. When retrievals are successful, the product is returned with a 200 (OK) status; otherwise, errors result in a 500 status. The updateProduct method connected to @PutMapping makes changes to an existing product using the provided ID and ProductDTO. If the update is successful, the method returns the updated product with a 200 (OK) status; if there are any problems, it returns a 500 status. The getProductsByCategoryName method, mapped to @GetMapping, retrieves products by name in their category. If no items are found, a 204 (No Content) status is returned; if things are found, a 200 (OK) status is issued. The status is 500 if there are any issues throughout this process. Finally, the @DeleteMapping-related deleteProduct method may be used to remove a product by its ID. 500 is returned if there are any problems; 204 (No Content) is returned if deletion is completed.

**User Controller**

****

The registerUser function may be used to register a new user using their email address and password. First, the email address is compared to a database to determine whether it already exists. whether it doesn't, the password is hashed and the user is added as a "customer." Following a successful registration, it provides the user's information along with a 200 (OK) status. Any issues that come up throughout this process are denoted by the appropriate status codes. The loginUser function controls user login by confirming the password and email provided. If the credentials are genuine, the procedure delivers user data with a 200 (OK) status; if not, an internal server error 500 status is returned.

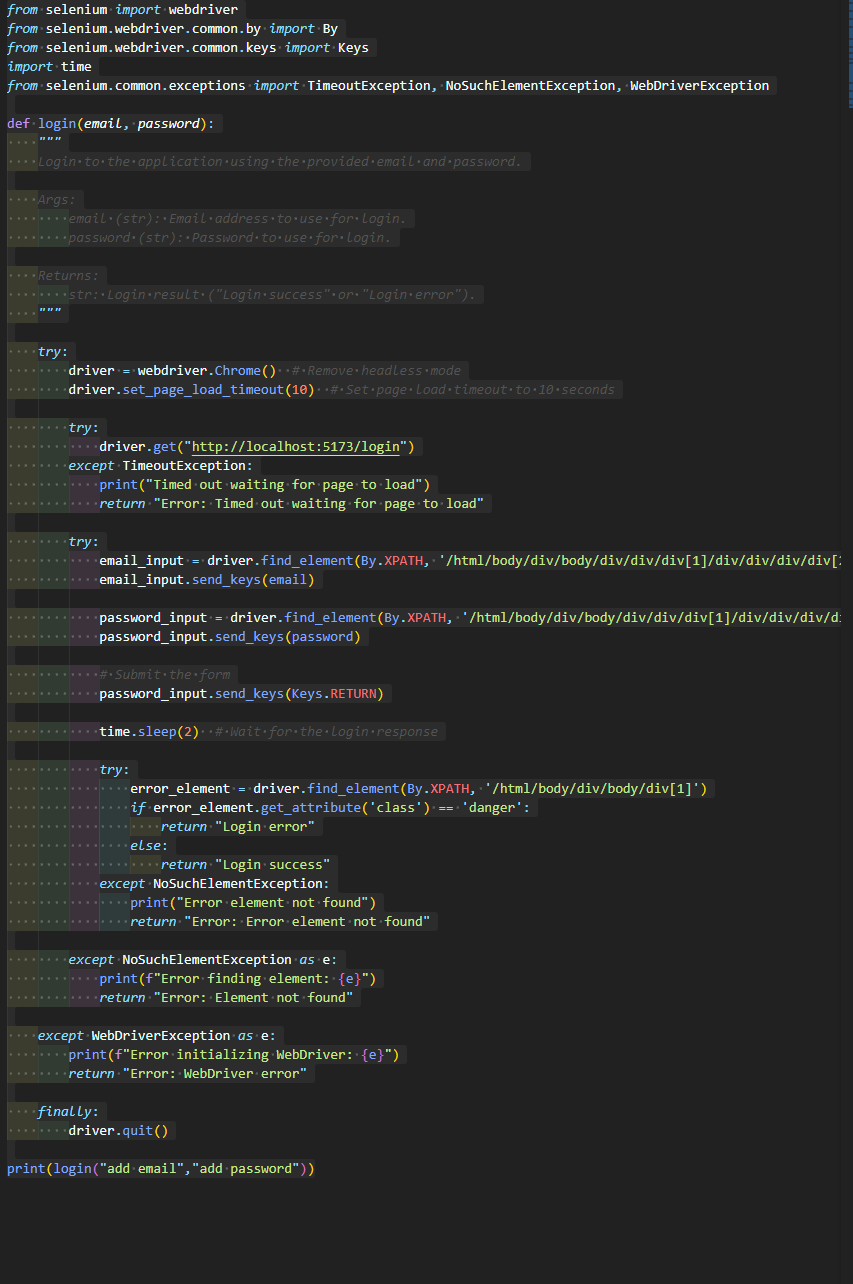
Use the deleteUser function to remove a user by their ID. Any issues yield a 500 status, whereas removals that are successful get a 200 (OK) status with a success message. Users may utilise the updatePassword method to update their passwords by providing their email address, old password, and new password. If the update is successful, the method returns a 200 (OK) status; otherwise, it returns a 500 status.

The process of changing the password is initiated using the requestPasswordReset method. A success message and a 200 (OK) status are returned if it is successful; a 500 status is given if there are any problems. By utilising the resetPassword feature, users may reset their password along with a verification number. If the reset is successful, it returns a 200 (OK) status; if an error occurs, it returns a 500 status.

The verifyUser method is in charge of using a verification code to confirm a user. When anything goes wrong, the status is 500; when it works, the status is 200 (OK). Use the updateUser method to update user data, such as type and status. An update that is successful returns a 200 (OK) status; failures result in a 500 status.

The hashPassword function is used to hash passwords using the SHA-256 algorithm. If the hashing process fails, a RuntimeException is produced. Lastly, the getAllUsers method is used to obtain every user. After a successful retrieval, users get a 200 (OK) status; any errors result in a 500 status.

**Selenium Testing**

****

The script automates the login process for a web application using Selenium WebDriver. Chrome is the recommended browser for initialising the Selenium WebDriver. The necessary modules are then installed in order to interface with web components, handle problems, and perform actions such as entering and submitting forms.

The login function of the script takes two inputs: an email address and a password. The initial stages in the process include opening the user-visible Chrome browser and visiting the login page URL, which is http://localhost:5173/login. The script then waits for the page to fully load, sometimes for up to 10 seconds. It returns a timeout error message if it doesn't.

After the page loads, the script uses its XPath selectors to find and interact with the email and password fields on the login form. It sends the RETURN key to submit the form after the specified credentials have been entered. The script checks the page for an error element after a submit to make sure the login was successful. An element with the "danger" class indicates a failed login attempt; in this scenario, the script displays a "Login error" message. If no such element is detected, the script returns the message "Login success" and considers the login successful.

Throughout the script, try-except blocks are utilised to handle common errors like as missing objects, issues with page loading, and WebDriver failures. Finally, to prevent resource leaks, the script ensures that the WebDriver object is closed by calling driver at the end of the run.stop(), regardless of the success or failure of the login attempt.

# **References**

Andrew, 2023. *blog/software-requirements-specification-srs-document/.* [Online]   
Available at: https://relevant.software/blog/software-requirements-specification-srs-document/  
[Accessed 31 August 2023].

Bandakkanavar, R., 2023. *projects/sample-software-requirements-specificationsrs-report-airline-database.* [Online]   
Available at: https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database  
[Accessed 31 August 2023].

Javatpoint, 2019. *design-patterns-in-java.* [Online]   
Available at: https://www.javatpoint.com/design-patterns-in-java  
[Accessed 31 August 2023].

Tutorialspoint, 2022. *design\_pattern/index.htm.* [Online]   
Available at: https://www.tutorialspoint.com/design\_pattern/index.htm  
[Accessed 31 August 2023].