UNIVERSITY OF BIRMINGHAM

Design and Implementation of an Online Shopping Web Application ——Dubai Shop

Jiang Chufeng
2378164

Supervisor
Dr. Ahmad Ibrahim

Inspector
Dr. Eike Ritter

Dissertation submitted to the University of Birmingham

For the Degree of MSc Computer Science.

School of Computer Science

University of Birmingham

September 2022

Abstract

“Dubai Shop” is a full stack development project, and the goal of the project is to create and develop a real-world website that mimicked Amazon online shopping and to develop a comprehensive solution that provided customers with a variety of purchasing choices. It is an e-commerce website designed with Java, JS, HTML, CSS and Spring Boot, named "Dubai Shop", where customers can shop online.

"Dubai Shop" is a complete solution for customers seeking to purchase goods, and it also provides an all-in-one content management system for Admin Application users such as administrators, salespeople, and shippers who are responsible for managing the frontend content and issues of the system.

This web application is based on MVC architecture, integrated with Spring Data JPA with Hibernate to handle data access, Spring RESTful Web Services returning JSON responses in conjunction with jQuery, and Thymeleaf for creating dynamic frontend views, etc.

The project has resulted in the development of a working prototype of an Online Shopping Web Application at the completion of the project.

Key words: HTML; CSS; JS; Bootstrap; MySQL; Spring Boot

Acknowledgements

This project has been carried out under the supervision of Dr. Ahmad Ibrahim and under the inspection of Dr. Eike Ritter, and I would like to thank them for providing guidance and feedback throughout the project.

In addition, I would like to take this opportunity to thank my parents, Jiang Yuying and Jiang Zhaosheng, who have been unconditionally supporting me throughout this very challenging academic year.

Furthermore, it is also my privilege to have Dr. Mian M. Hamayun, Dr. Ahmad Ibrahim, Dr. Anis Zarrad, and Dr. Kashif Rajpoot provided me with an in-depth and comprehensive understanding of computer science through their courses in full stack development, computer system, data structures and algorithms, databases, systems design, and artificial intelligence during this academic year.

Finally, I would like to thank my classmates (2021-2022) in Dubai for their kindness and help through both academic and personal difficulties. There is nothing better than studying with them all during the year, we did our study together, we went to the cheapest PCR clinic together, we collaborated on debugging together, they alerted me when a credit card fraud occurred, and they took me home whenever they can. Studied with them all did make it quite a special year.

Table of Content

List of Figures

1 Introduction

In today's online shopping environment, multi-channel selling is a significant revenue possibility that cannot be ignored, especially since it is growing at a rapid pace. As the number of sales channels skyrockets, the process of fulfilling online orders becomes increasingly complex.

In today's marketplace, there are multiple vendors, providers, or systems used by online shopping brands to manage fulfilments. In addition to posing a significant threat to market productivity, unnecessary costs and insufficient operational transparency arise from the complexity of fulfillment workflows, guidelines, and different service providers. It is possible to miss out on a significant number of potential customers by relying solely on brick-and-mortar stores. An online store is a great way for a business to expand. A physical store will eventually reach a point where the physical store will no longer be able to grow without a streamlined ecommerce model to maximize its potential.

In this report, I will describe how I developed the web application to manage the business from developing the Admin application and the Shopping application. In hopes of providing a one-stop shopping solution for end users of the web application, it aims to provide a one-stop online shopping experience for customers.

1.1 Report Overview

This report has six parts. The first part is to stress the importance of online businesses. The second part will introduce the project specification which tell the main functions to be realized and included in the project. The third part gives a picture of how the application is going to design and how the system will be used. The fourth part is going to explain the logical designing and coding for the systems' critical modules. In addition, important real user interfaces are introduced, and validation and testing results are recorded in this part. The evaluation part and summary part follow in order.

2 Project Specification

System requirements describe the target system's capabilities and features and express the expectations of its users (Tutorialspoint, 2022). The user and system requirements are introduced in this section, followed by use case scenarios.

2.1 User Requirements

The user requirements describe what the software or the system should be able to do. The document should be written from the perspective of the end user and it is better to write in an easy to understand way and should not be technical or complex (Intersys, 2021).

There are five different types of roles divided into Admin application users and Shopping application users based on their functionality. The roles on the Admin application are the administrator who manage everything related to the website, the salesperson who handle the sales of products, and the shipper who deliver orders. Visitors and customers are the two different types of roles that reside in the Shopping application.

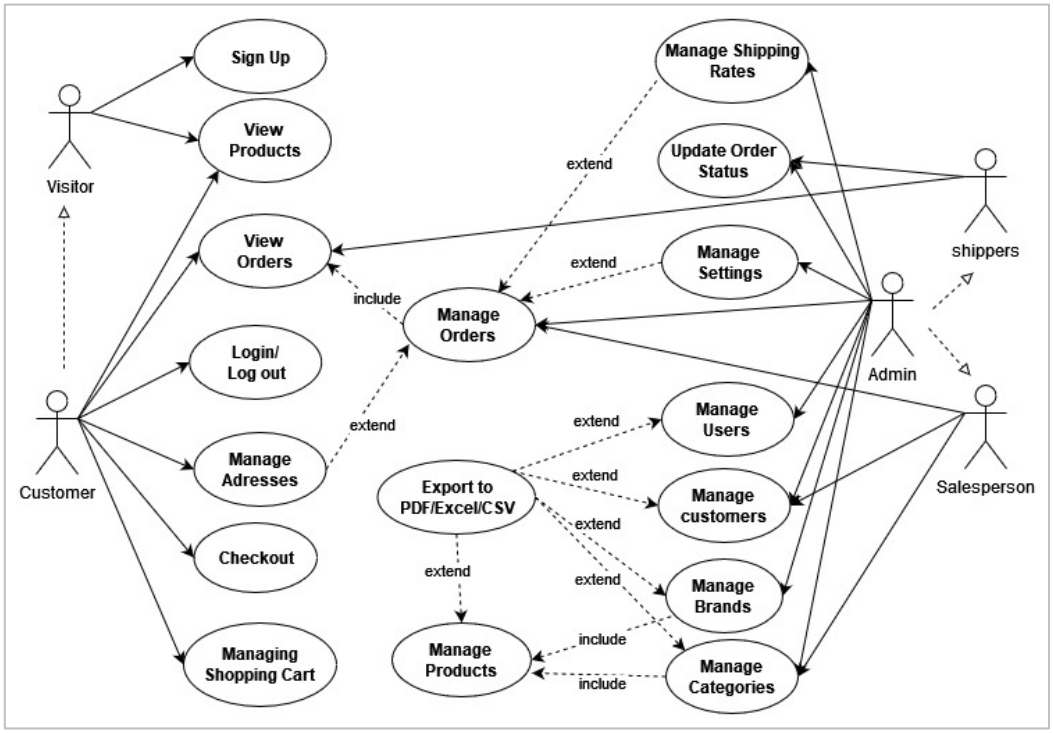


Figure 2.1.1 Overview User requirements

2.2 System Requirements

System requirements describe what software is supposed to do and how it will perform. Additionally, it describes the functionality and non-functionality of the system needed to meet the needs of all stakeholders (Lane and Krüger, 2021)

2.2.1 Functional Requirements

1. The Admin Application MUST retain information of its user including their:
   - First and last name
   - roles
   - Email
   - photo
   - passwords
2. The Admin Application MUST retain information of products:
   - Brands
   - Brands ID
   - Brand Logo
   - Product Categories
   - Product Images
   - Prices
3. The Admin Application MUST retain information of orders:
   - Order ID
   - Product ID
   - Quantities
   - Customer ID
   - Customer Address
   - Customer Name
   - Prices
   - Shipping Cost
   - Order Status
   - Order Time
   - Deliver Date
   - Payment Method
4. The Shopping Application MUST retain information of customers:
   - Customer ID
   - Customer Address
   - Customer Name
   - Customer Phone
   - Customer password
   - Customer type
5. The Shopping Application MUST retain information of shopping carts:
   - Customer ID
   - Customer Address
   - Customer Name
   - Product ID
   - Shipping Cost
   - Quantities
   - Product Price
6. The Admin Application and Shopping Application users CAN edit their personal information.
7. The Admin Application and Shopping Application MUST give out alerts when users input illegal format data or click wrong button.
8. The Admin Application MUST distinguish different types of users and give different authorities to access different modules.
9. The Shopping Application MUST display the total value of the selected items with a single line of tax charged, and shipping costs separately.
10. The Shopping Application users MUST be able to view the categories.
11. The Shopping Application users MUST be able to view items in different categories or different brands.
12. The customer SHALL be able to view product details before adding it to the cart.
13. The visitor MUST be able to register as a customer.
14. The customer MUST be able add goods to the cart.
15. The customer MUST be able to able view the shopping cart.
16. The customer MUST be able to check out goods selected in the cart.
17. The visitor MUST register using the E-mail verification code.
18. The visitor MUST register before adding goods to the shopping cart.
19. The customer CAN NOT login or register if the information is incomplete or invalid
20. The customer CAN NOT place an order without completing the order form.

21. The administrator MUST be able to view all the Admin Application and Shopping Application users' information.
22. The administrator and the salesperson MUST be able to add new goods to customers' shopping items.
23. The administrator and the salesperson MUST be able to edit a product's price, images and description.
24. The administrator and the salesperson MUST be able to delete goods from the customer's orders.
25. The salespersons SHALL be able to view all the customers' order details after they complete the order form and finish the checkout process.
26. The salespersons MUST be able to edit an product's price, images and description.
27. The shipper MUST be able to view customer's orders.
28. The shipper MUST be able to search orders placed.
29. The shipper MUST be able to update customer orders' current status (picked, shipping or delivered)

2.2.2 Non-Functional Requirements

1. The Admin Application and Shopping Application users MUST be able to access the system all year round any time, 24/7.
2. The website MUST be accessible from any devices connected to the Internet for example PC, tablet, and mobile phone.
3. Passwords MUST be encrypted rather than stored in plain text in the databases.
4. The system SHALL be able to save goods in the shopping cart when the website close unexpectedly.
5. The system MUST keep the Admin Application and Shopping Application users' data persistent when the system get updated.
6. The website MUST be easy and simple to use and does not require much training for all customers.
7. The customer SHALL be able to add a product to the shopping cart in fewer than 4 seconds.
8. The customer SHALL be able to view information about a product in fewer than 4 seconds.
9. The customer SHALL be able to check out the goods in the shopping cart within 20 seconds after making payment.
10. The shift between pages SHALL take fewer than 2 seconds.
11. The system SHOULD have scalability for developing new features on demand and implementing well under high load.
12. The Admin Application and Shopping Application SHALL be scaled separately.

2.3 Use Case Scenarios

A use case scenario is the interaction between users and the system to achieve the system's goal. Usually, it is part of the functional requirements plan or document for software development (Guthrie, 2022). Detailed steps are

described in this section for how visitors can place orders and receive products.

2.3.1 Visitor Purchasing

[Precondition]

1. A visitor is browsing the website on a laptop and wants to purchase a camera.
2. The visitor has his own phone number and E-mail.

[Flow of Events]

1. The use case starts when the visitor accesses the “Dubai Shop” website's home page.
2. The visitor clicks “Register” button to register as a customer:
   2.1. The visitor turns to the register page and fill in the register form.
   2.2. The visitor clicks “Create Account” button.
   2.3. The system sends a verification E-mail to the visitor’s account.
   2.4. The visitor opens his E-mail and receives the verification letter.
   2.5. The visitor clicks the verify link in the verification letter.
   2.6. The visitor’s account is enabled, and he registered as a customer.
3. The customer searches for the desired product by clicking the icon “Digital Cameras”, and then get into products details page.
4. After looking through descriptions for different types of cameras, the customer finds the suitable product and takes the desirable camera into his shopping cart.
5. If the customer leaves the cart, the camera stays in the cart. Next time when the customer come back, he can go to checkout or go for shopping the second time.
6. After the camera is added into the cart, the checkout process begins:
   6.1. The customer is required to check the shipping address whether it is support for shopping. If the Address is not support for shipping, the customer is required to modify his address or add another address in his address book.
   6.2. The total purchase amount is displayed including the taxes and the shipping fees. The customer is then presented with Cash/PayPal/Credit Card payment methods to choose from.
   6.3. The customer selects the payment method provided and fill in the required information.
   6.4. If the payment successfully issued, then the payments is completed, otherwise he will return to the shopping cart.
7. A message displays that the order is successful launched.
8. The system sends an order confirmation to the customer’s E-mail address.
9. The order status automatically updated as “PAID”
10. The details regarding the order are sent to the Salesperson.

[Postcondition]

1. The order has been placed.
2. The customer is waiting the salesperson to deliver and the shipper to dispatch

the camera.

2.3.2 The salesperson and the shipper Processing

[Precondition]

1. The salesperson received the order to buy camera.
2. The shipper is available
3. The product is available.

[Flow of Events]

1. The salesperson receives the order.
2. The salesperson picks up the camera in his warehouse and packs the camera.
3. The salesperson contacts the shipper to take the package.
4. The salesperson delivers the package to the shipper.
5. The shipper updates the order's status as "PICKED".
6. The shipper updates the order's status as "SHIPPING".
7. The shipper takes the package to the customer's address and gives it to the customer.
8. The shipper updates the order's status as "DELIVERD".

[Postcondition]

1. The customer received the camera.
2. Salesperson has successfully sold out a camera if the customer is not requiring of a return.
3. The shipper has completed an order.

3 Solution Design

In the Software Development Lifecycle, solution (and software) design occur between analysis and development (Lteif, 2022). To explain how the "Dubai Shop" Web Application will be designed, System architecture, MVC architecture, and Technology stack are presented, as well as UML diagrams.

3.1 System Architecture

"Dubai Shop" is running on a local development environment in a single computer. This Web Application includes Admin application and Shopping application, both running as a standalone JAR files. Spring boot Jars are self-contained Jar that has a collection of all the dependencies required for running the application (EDUCBA, 2022). The Admin application and Shopping application are connecting with same local databases MySQL, and the static resources such as site logo, product images, brand images and category images are stored in the local environment.

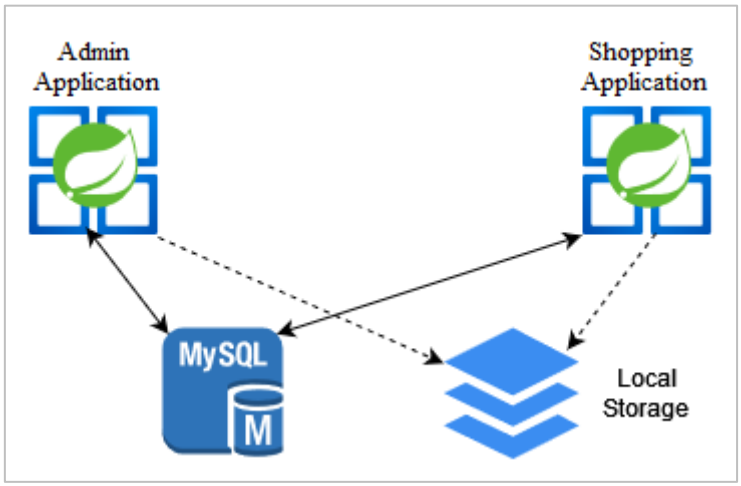


Figure 3.1.1 System Architecture

Although the Admin Application and Shopping Application are separate applications, they have some shared codes for common entities, as they are both supported by the same database. As well as their own dependencies, both Spring Boot web applications share several common dependencies, such as Spring Data JPA, MySQL JDBC driver, and Spring Web. Therefore, there are three coding components to this project: Backend codes, Frontend codes, and Common Entity codes shared between both applications.

3.2 High-level Architecture (MVC)

The MVC architecture will be used to develop “Dubai Shop” because I can practice the skills I learned in my *Full-Stack Application Development* course in my second semester.

Model–View–Controller is a popular software pattern used to break up the logic of your application into three different components (Wilkins, 2021) . A spring MVC is a certain kind of container of the Spring framework and Model View Controller, which is easy to use to build web application, implementing core spring framework features like IOC and dependency injection (Scaler Academy, 2022).

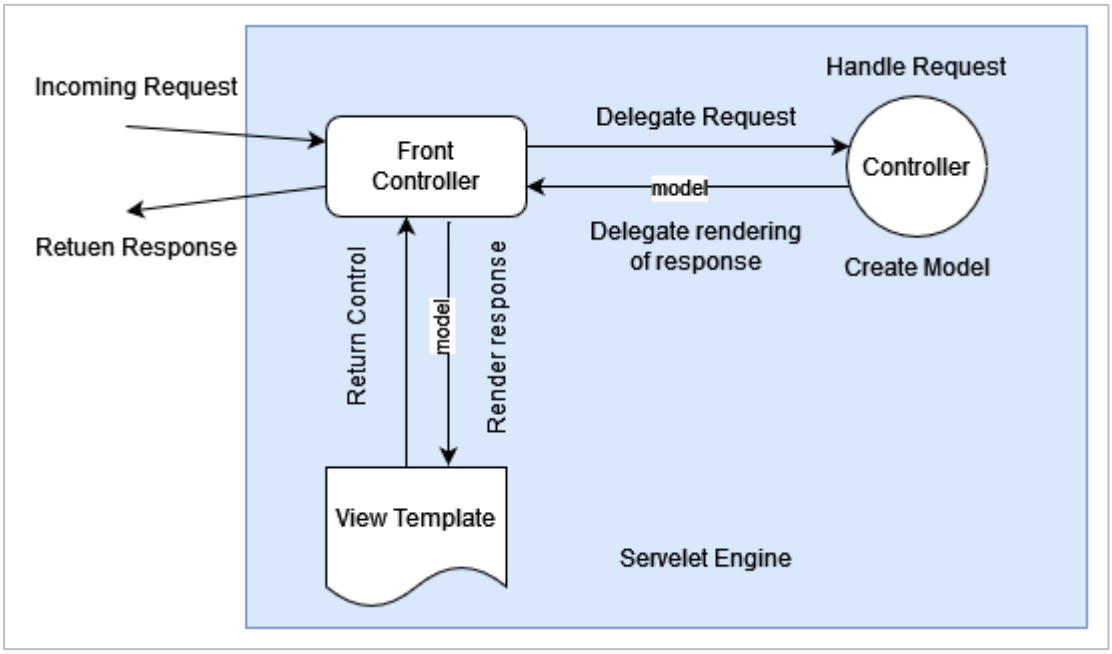


Figure 3.2.1 MVC Overview
(Source: Spring Framework Reference Documentation, Johnson, et al *2016)*

According to the "Spring MVC Tutorial" posted on the javatpoint website, introduction for the concepts of MVC terms are as follow:

- ***Model*** *- A model contains the data of the application. Objects can be single or grouped together to make up a data object.*
- ***Controller*** *- Business logic is contained within a controller.A class is marked as a controller by adding the @Controller annotation.*
- ***View*** *- In a view, information is presented in a specific format. View pages are generally created using JSP with JSTL. Additionally, spring supports Apache Velocity, Thymeleaf, and FreeMarker view technologies.*
- ***Front Controller*** *– The Servlet Dispatcher is the front controller whose job is to handle dispatches of requests in Spring Web MVC. This component manages the flow of Spring MVC applications.*

The "Dubai Shop" project utilizes the MVC architecture as well. The view layer using thymeleaf and HTML codes renders the HTML web pages to the client, such as order.html, order_form.html, and address_book.html. The view layer invokes the controller layer that uses the Spring MVC Controller, REST controller or RESTful webservices to handle the request from the clients. The controller layer calls the service layer to perform the business logic of the application. And then it will render the view that is returned to the client. The service layer depends on the repository layer that contains entity classes and repository interfaces. Below the repository is the Spring Data JPA layer which uses Hibernate framework as the implementation of Spring Data JPA. Furthermore, the Hibernate uses JDBC driver

to communicate with the underlying database instance.

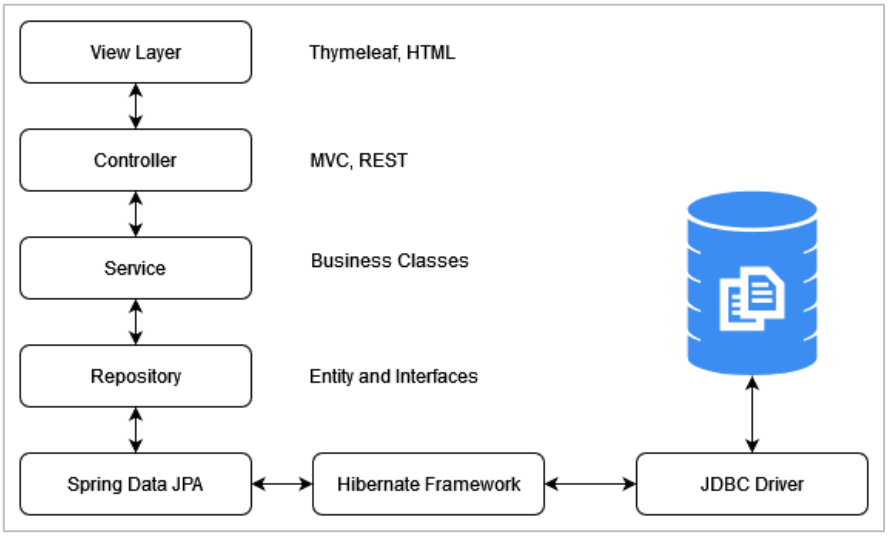


Figure 3.2.2 MVC of "Dubai Shop"

3.3 Technology Stack

3.3.1 Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible (Wikipedia, 2022). It was originally released by Sun Microsystems in 1995 as a programming language and computing platform. It provides the reliable platform upon which many services and applications are built in today's digital world that has grown from humble beginnings. Also in the future, Java will be used to develop innovative products and digital services (Oracle, 2022).

3.3.2 Spring Boot

The Spring framework is an open source, enterprise-level framework designed to help developers design standalone, production-ready JVM-based applications. Spring Boot provides three core capabilities that make Web application and microservice development easier using Spring Framework. The three core capabilities are as follows (IBM Cloud Education, 2020):

1. Autoconfiguration.
2. Create standalone applications.
3. An opinionated approach to configuration.

3.3.3 HTML and CSS

The Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) are two of the core technologies that are used to create Web pages. An HTML page describes how a Web page is structured. Markup is used to describe the structure of web pages in HTML. Content pieces are labelled by elements of the language.

For instance, you can label a paragraph, a list, a table, etc. Colors, fonts, and layout are all described in CSS, which is a language for describing how Web pages are presented. A large screen, a small screen, or a printer can all be included in the presentation due to its adaptability (W3C, 2022)

3.3.4 Thymeleaf

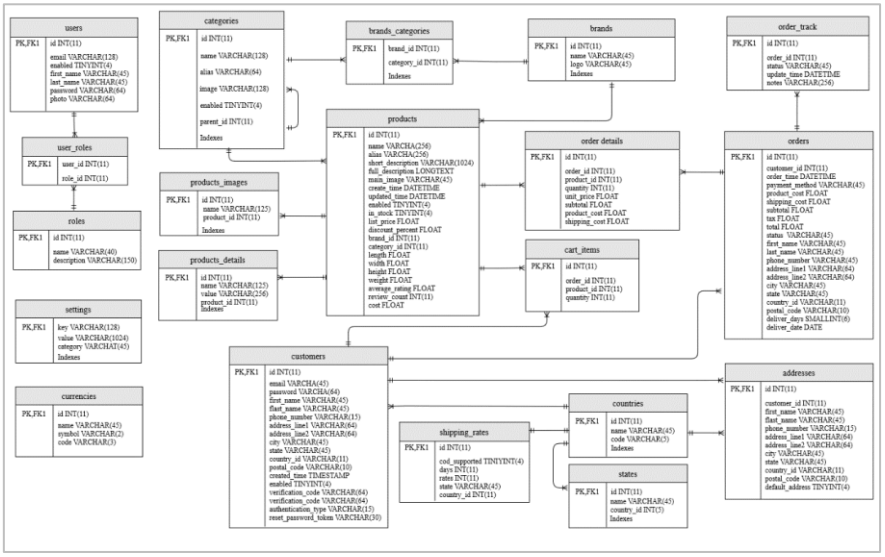
Thymeleaf is a Java template engine that runs on server-side and in standalone mode. A main goal of Thymeleaf is to make your development workflow easier through elegant natural templates - HTML that displays correctly in browsers and can also be used as static prototypes. This allows developers to work more collaboratively. As a modern HTML5 JVM web development framework, Thymeleaf can be plugged into tools including Spring Framework modules (The Thymeleaf Team, 2022).

3.3.5 Libraries and APIs

antlr:2.7.7
log4j-to-slf4j:2.17.2
jquery:3.4.1
openpdf:1.3.8
curvesapi:1.06
snakeyaml:1.30
json-path:2.7.0
poi-ooxml:4.1.0
bootstrap:4.3.1
content-type:2.2
log4j-api:2.17.2
popper.js:1.14.3
tomcat-embed-websocket:9.0.64
net.sf.supercsv:super-csv:2.4.0
spring-boot-autoconfigure:2.7.1
spring-boot-starter-tomcat:2.7.1
thymeleaf-spring5:3.0.15.RELEASE
spring-boot-starter-logging:2.7.1
spring-security-oauth2-core:5.7.2
org.apache.xmlbeans:xmlbeans:3.1.0
spring-boot-starter-data-jpa:2.7.1
spring-boot-starter-security:2.7.1
spring-boot-starter-thymeleaf:2.7.1
spring-security-oauth2-client:5.7.2
jackson-module-parameter-names:2.13.3
spring-boot-starter-oauth2-client:2.7.1
hibernate-commons-annotations:5.1.2.Final
javax.activation:javax.activation-api:1.2.0
thymeleaf-extras-springsecurity5:3.0.4.RELEASE
data:spring-data-commons:2.7.1
spring-boot:2.7.1
spring-orm:5.3.21
spring-web:5.3.21
jakarta.mail:1.6.7
FastInfoset:1.2.13
spring-jdbc:5.3.21
xmlunit-core:2.9.0
javassist:3.23.1-GA
spring-beans:5.3.21
nimbus-jose-jwt:9.22
oauth2-oidc-sdk:9.35
spring-webmvc:5.3.21
commons-compress:1.18
spring-context:5.3.21
jcip-annotations:1.0-1
jackson-databind:2.13.3
juniversalchardet:2.3.0
poi-ooxml-schemas:4.1.0
unbescape:1.1.6.RELEASE
byte-buddy-agent:1.12.11
commons-collections4:4.3
tomcat-embed-core:9.0.64
thymeleaf:3.0.15.RELEASE
jboss-logging:3.4.3.Final
webjars-locator-core:0.50
ch.qos.logback-core:1.2.11
jackson-annotations:2.13.3
jakarta.xml.bind-api:2.3.3
hibernate-core:5.6.9.Final
spring-boot-devtools:2.7.1
data:spring-data-jpa:2.7.1
mysql-connector-java:8.0.29
spring-security-config:5.7.2
ch.qos.logback-classic:1.2.11
jakarta.persistence-api:2.2.3
jakarta.transaction-api:1.3.3
javax.xml.bind:jaxb-api:2.3.1
javax.activation:activation:1.1
spring-boot-starter-aop:2.7.1
spring-boot-starter-jdbc:2.7.1
spring-boot-starter-json:2.7.1
spring-boot-starter-mail:2.7.1

3.4 Entity Relationship Diagram

An Entity Relationship Diagram is the highest abstraction for the data model. It shows the data requirements for a database graphically. Using diagrams and symbols, data modeling documents software systems. The communication of data is represented by it (University of Regina, 2017). For a close look at the ER Diagram for “Dubai Shop”, please reference to the Appendix-C: Figure 3.4.1 Entity Relationship Diagram.



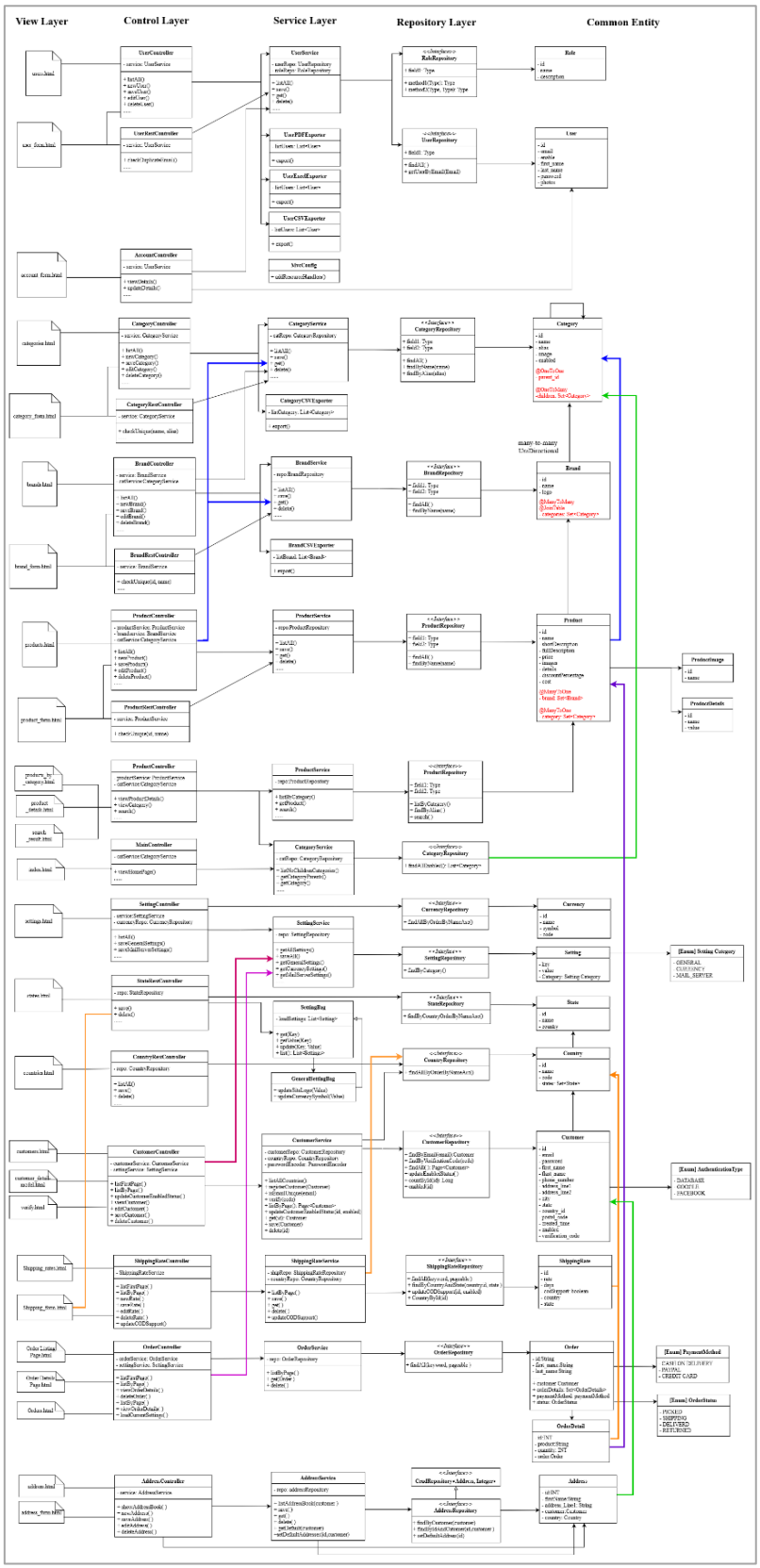
Appendix-C: Figure 3.4.1 Entity Relationship Diagram

3.5 Class Diagram

Class diagrams show how objects in an object-oriented system are built. In class diagrams, attributes and behaviors are described rather than the methods for achieving operations, which is different from the description of a dynamic model. Relationships between classes and interfaces are best illustrated with class diagrams (Sparx Systems Pty Ltd, 2022).

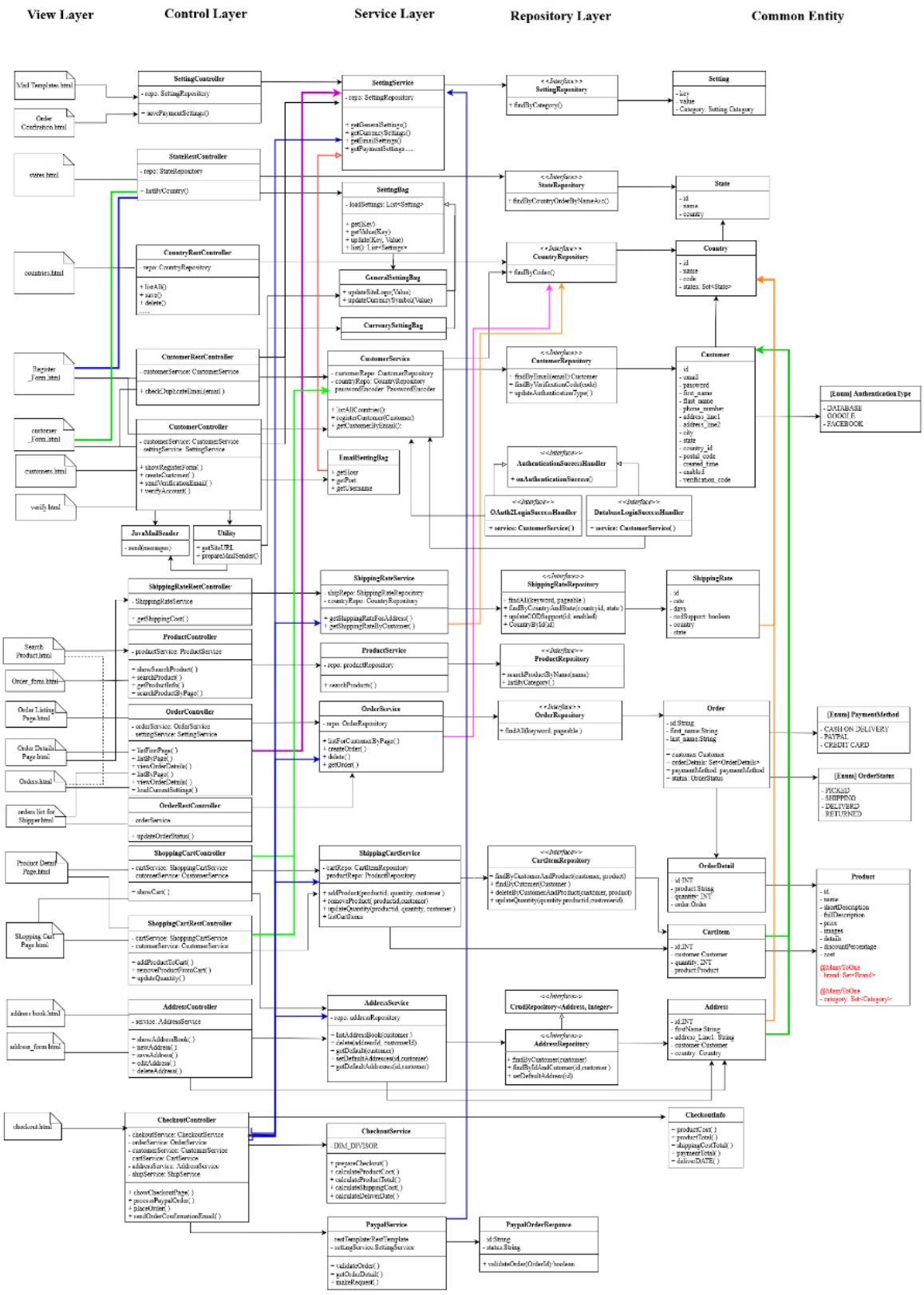
For a close look at the Class Diagram for “Dubai Shop”, please reference to the Appendix-D: Figure 3.5.1 Class Diagram-Admin Application and Figure 3.5.2 Class Diagram-Shopping Application

3.5.1 Admin Application Side



Appendix-D: Figure 3.5.1 Class Diagram-Admin Application

3.5.2 Shopping Application Side



Appendix-D: Figure 3.5.2 Class Diagram-Shopping Application

3.6 Activity Diagram

To visualize the sequence of activities, an activity diagram is used. The flow chart shows the progression of events in the activity from the start point to the finish point. The parallel processing may be used to describe situations where certain activities may be executed in parallel. A process diagram is useful for business modeling since it specifies how business activities are carried out (Sparx Systems Pty Ltd, 2022).

3.6.1 Visitor Registration and Customer Login

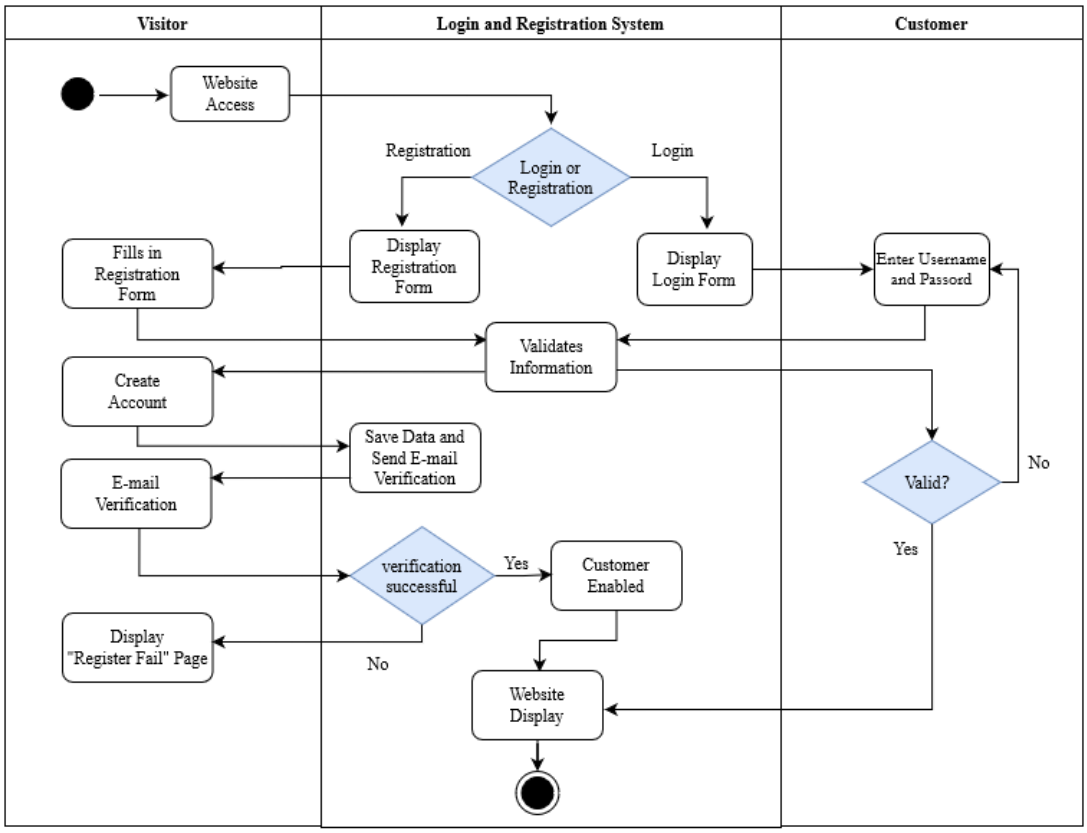


Figure 3.6.1 Activity Diagram- Visitor Registration and Customer Login

3.6.2 Customer Purchasing and Shipper Delivering

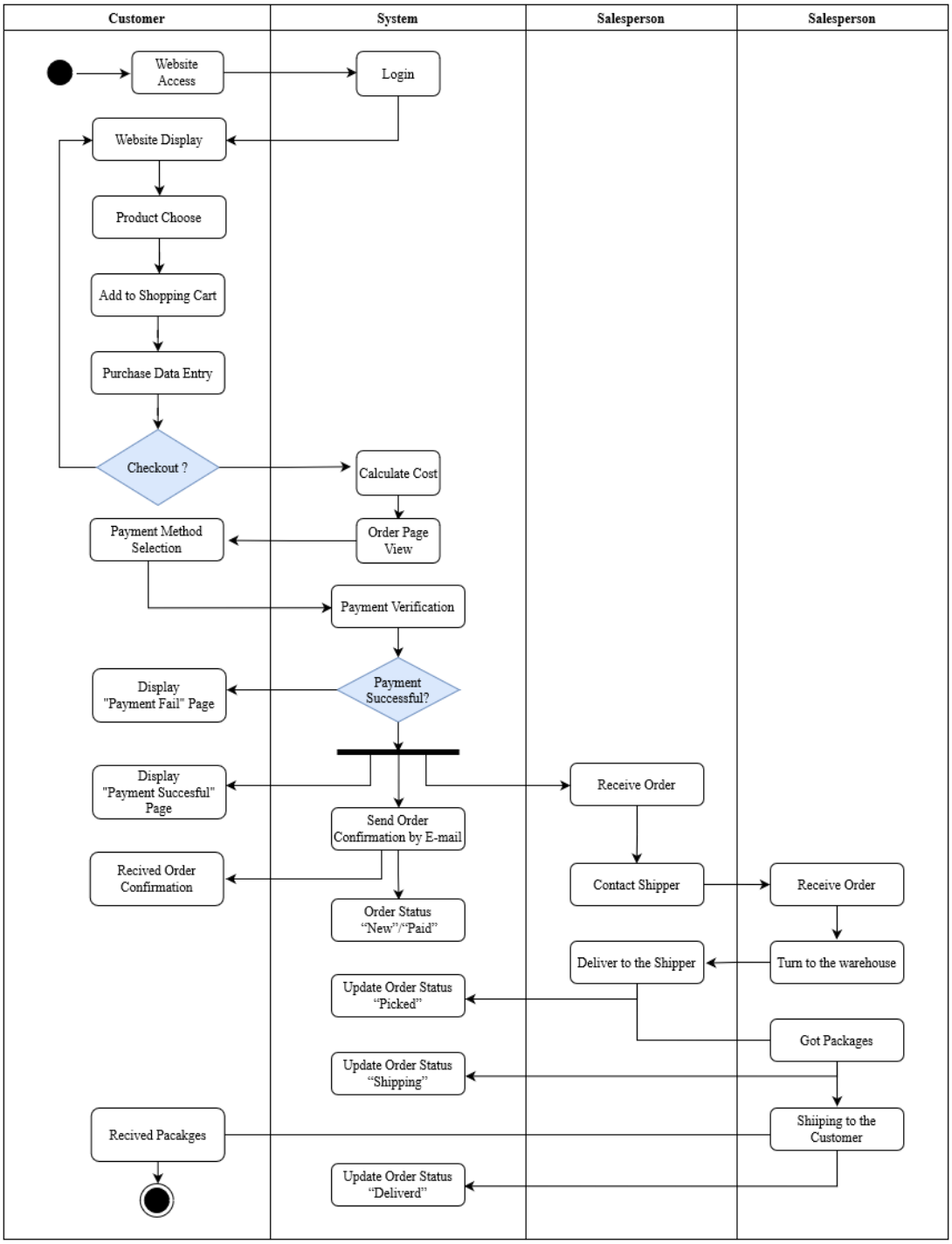


Figure 3.6.2 Activity Diagram- Customer Purchasing and Shipper Delivering

3.7 Sequence Diagram

Basically, a sequence diagram defines the sequence of events that leads to a particular outcome. In sequence diagrams, messages are less important than the order in which they occur; nevertheless, the diagram communicates both the

messages sent and their ordering between objects in a system. This information is presented in the diagram along two dimensions: on the vertical dimension, messages/calls are shown in chronological order as they occur, while on the horizontal dimension, messages/calls are shown as well as the instances of the objects they are sent to (Bell, 2004).

3.7.1 Visitor Registration

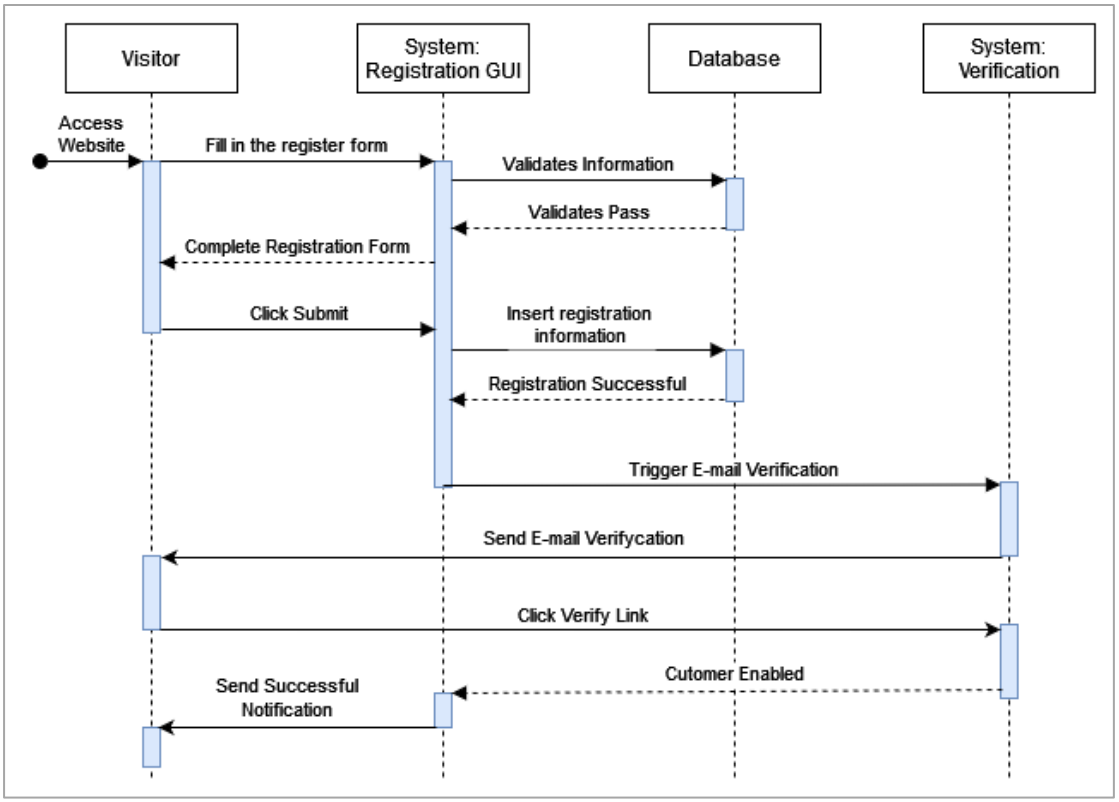


Figure 3.7.1 Sequence Diagram- Visitor Registration

3.7.2 Purchasing Products

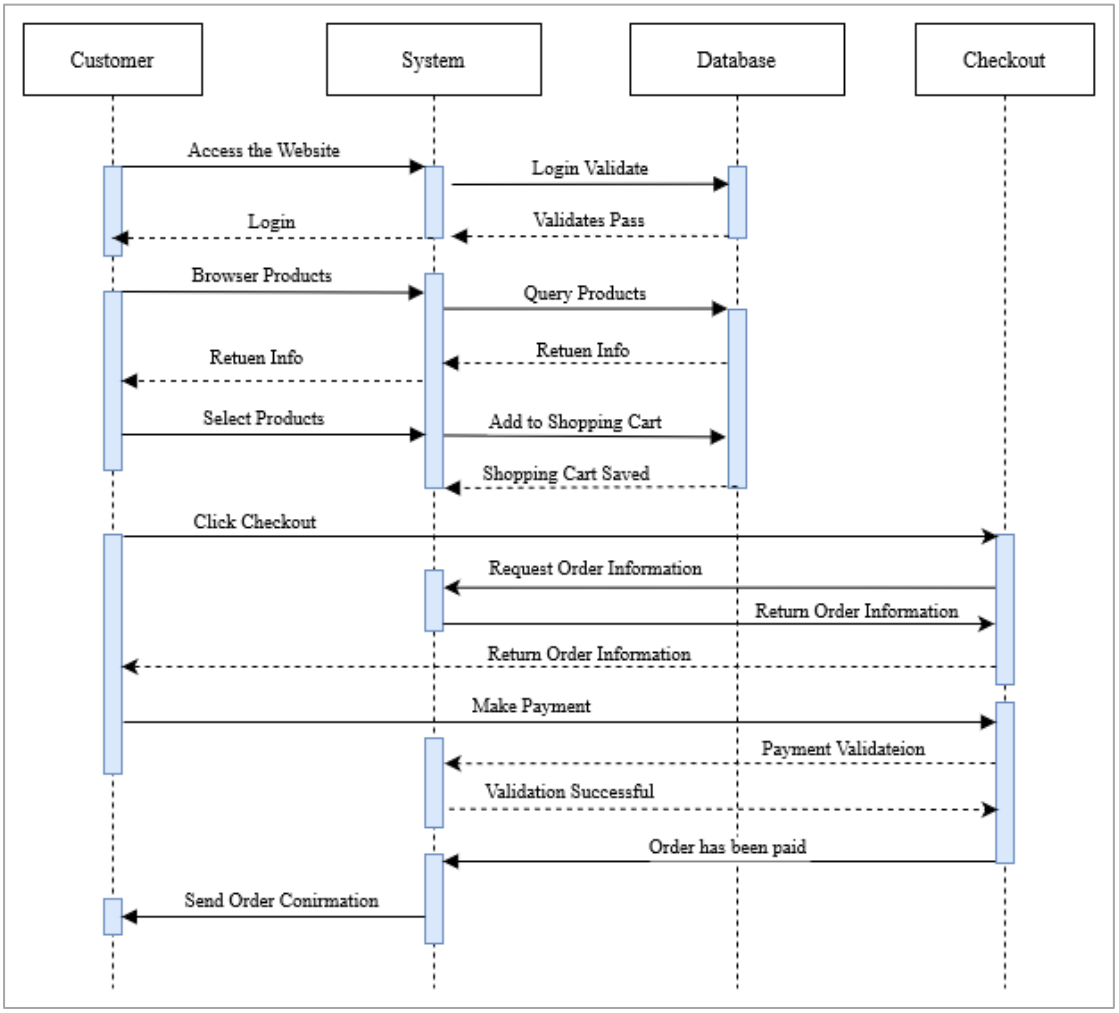


Figure 3.7.2 Sequence Diagram-Purchasing Products

3.7.3 Shipping Products

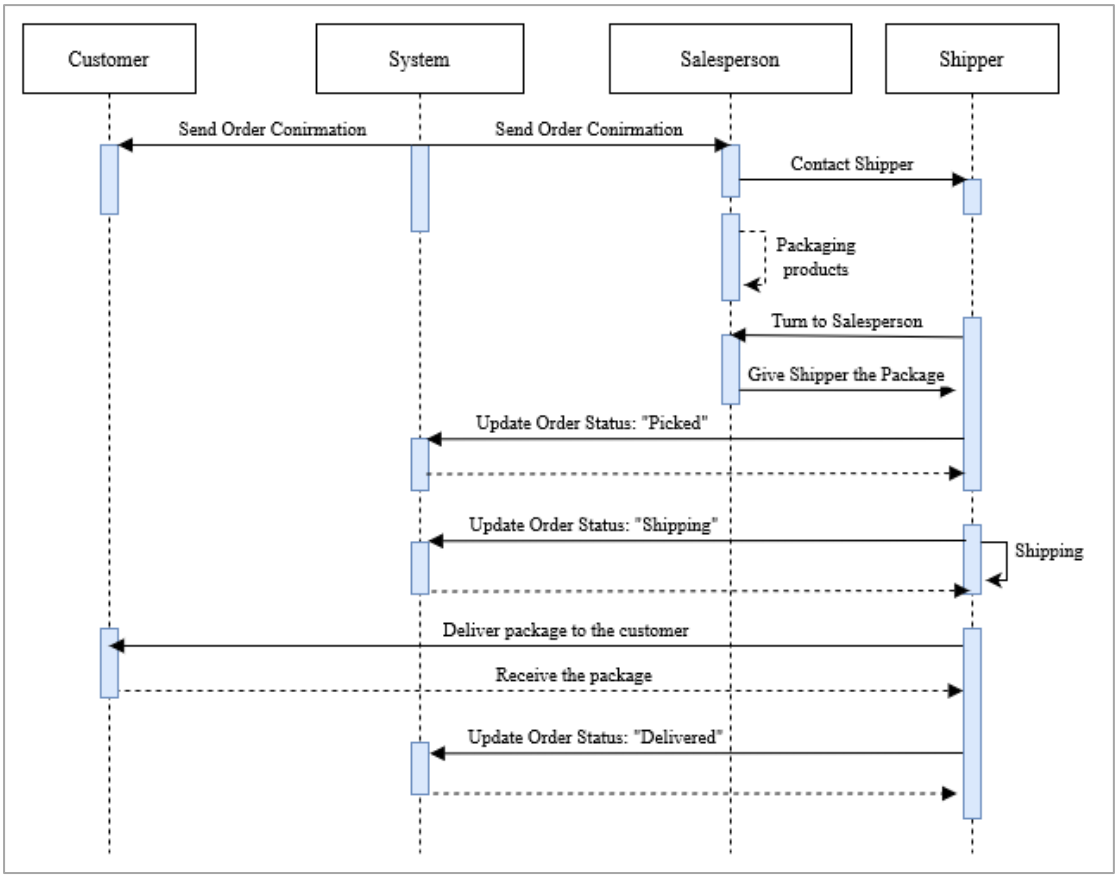


Figure 3.7.3 Sequence Diagram- Shipping Products

3.8 User Interface Mockup

3.8.1 Navigation Bar

In this project, different users have different access to different functional modules. Admin users have all access to all modules, while shipper and salesperson have access to some of the modules rather than full access. However, if a limited-access role is granted another role at the same time, he can access the combination of modules from each role. For example, if a shipper is upgraded as an Admin, he will be granted the full access right to every module.

Figure 3.8.1 UI Mockup- Navigation Bar

3.8.2 View Module Content List

This is a reusable template to design module lists of items. Users, Brans, Categories, Products are followed the same design logic that of these stuff's basic information will be displayed in rows and columns.

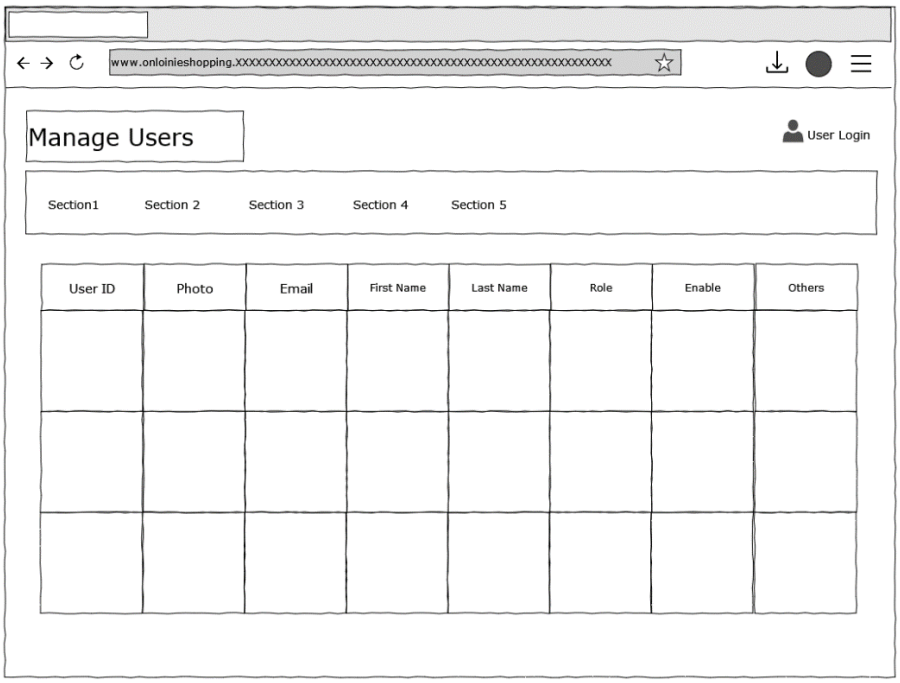


Figure 3.8.2 UI Mockup-View Module Content List

3.8.3 Product Display

In the product details page, products images are presented at the left-up side. Its name and price display in the center. Short description area gives a brief introduction of this product while in the full description area, product specification and more details will be given.

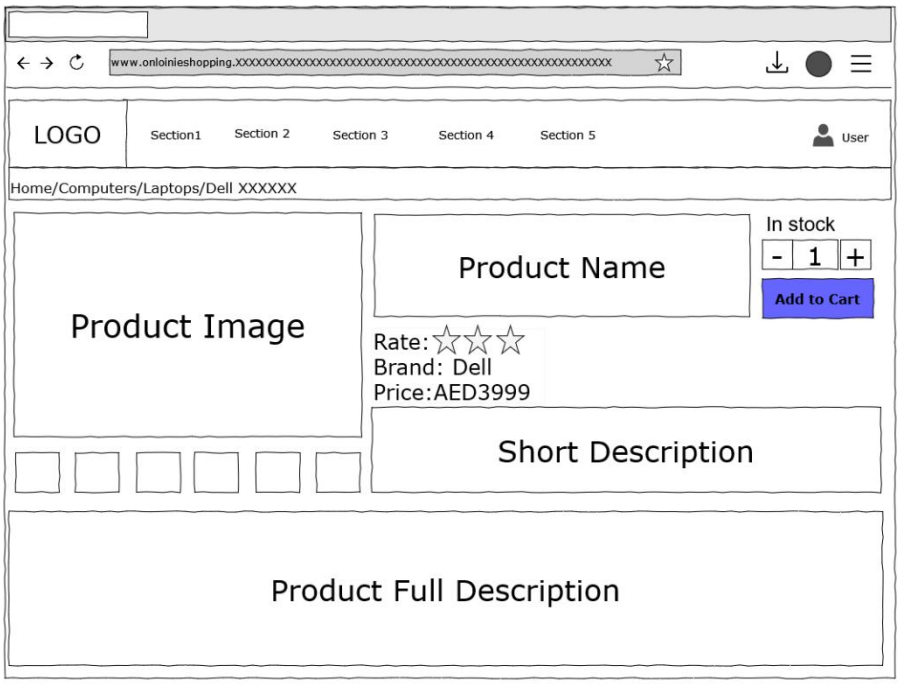


Figure 3.8.3 UI Mockup- Product Display

3.8.4 Shopping Homepage

In the shopping homepage, products are belonged to different categories, and only the categories were displayed in rows and columns in this page. When the customer or visitor click the icon of a certain category, he will be redirect to another page that lists all the related products.

The mockup for shopping homepage can be found in the later paragraph.

3.8.5 Create User and Customer Registration From

To create a Admin Application user, Name, E-mail, and password should be filled in, and at least one role must be selected. Users can upload their photos by themselves through their personal portal.

However, to register to be a customer will be more complicated. Twelve blankets must be filled in, including name, e-mail, address, phone number, country, state, city, etc. It is only possible to leave Address Line2 empty, and all other blankets must not be null.

The mockup for user creation form and customer registration form can be found in the later paragraph.

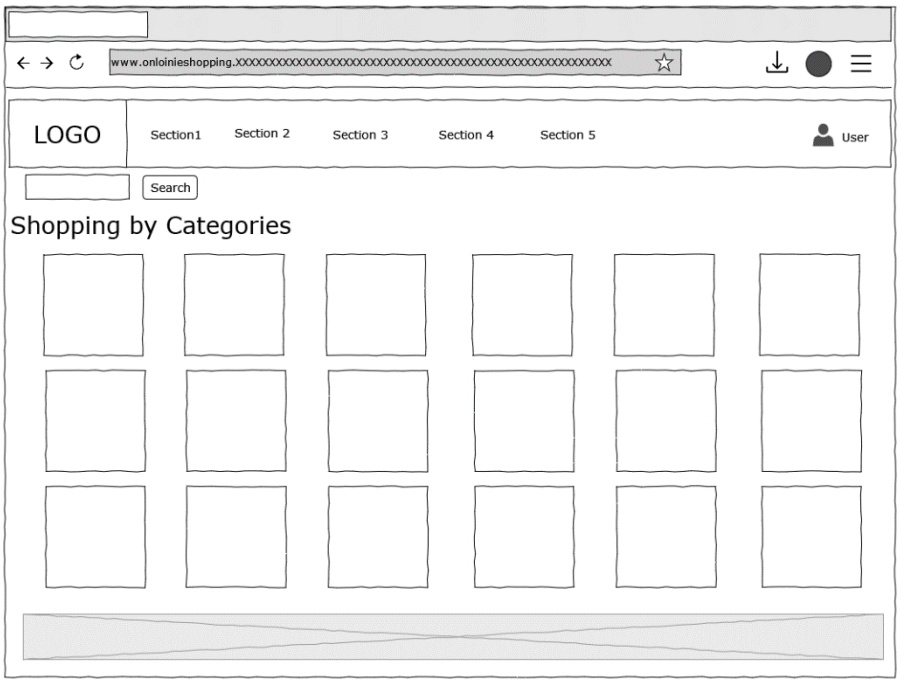


Figure 3.8.4 UI Mockup-Shopping Homepage

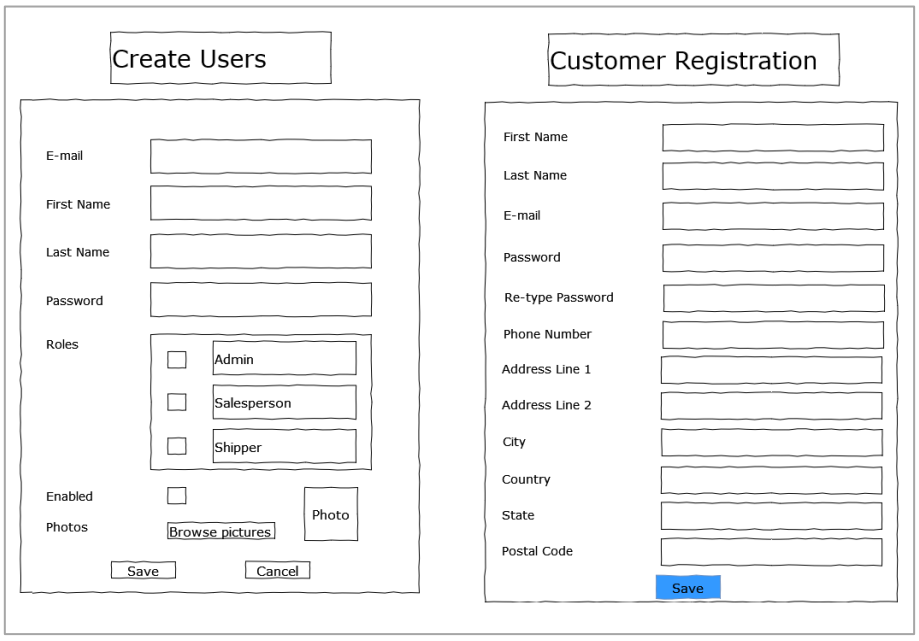


Figure 3.8.5 UI Mockup-Create User and Customer Registration From

3 Solution Implementation

In this part, several critical modules will be discussed in detail, and followed by the validation and testing. Other modules' which are relatively simple and follow the same design logic will be displayed in **Appendix-E: Module UML Diagram**. All the pictures related to module design can be found in Appendix-E

4.1 Admin User Module

4.1.1 Encoder User Password

In the "Dubai Shop" Project, the BCrypt password encoder was used to help secure the password security. It is provided by Spring Security and is a password-hashing function based on Blowfish and crypt function in Unix (Wikipedia, 2022). The hashed password stored in the database is the result of encrypting the value "OrpheanBeholderScryDoubt" 64 times with the final state from the last run of the key schedule (Raman, 2021)

Following is a description adopted from an article written by Emmanuel Hayford (2021) of how hash-functions enable authentication for admin management system:

1. A user named Jiang Chufeng, creates an admin control panel account.
2. Later, Jiang Chufeng chooses a username and password for herself.
3. The server performs a Bycrypt algorithm operation on Jiang Chufeng's password, keeps the hashed password, and stores the string in the database.
4. when Jiang Chufeng returns to the site, her login details are securely transferred to the system's backend.
5. Once the backend server receives Jiang Chufeng's login information, it searches the database and sends her password through the same hash function she used when she registered.
6. If the password hash of the password Alice entered matches the one stored in the database, Alice is granted access. Otherwise, the site denies Alice access.
7. Access is granted to Jiang Chufeng if the password hash matches the one stored in the database. Otherwise, she will not be able to access the system.

Let me explain how the code is play in this project.

Using the Bcrypt algorithm, BCryptPasswordEncoder implements the hashing of passwords. Define the BCryptPasswordEncoder as a bean in the project backend module's WebSecurityConfig java class file:

```java
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    /*
    @ Java Development Journal, 2020
    Reference: https://www.javadevjournal.com/spring/password-encoding-in-spring-security/
  */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        return new DuShopUserDetailsService();
    }

    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }
}
```

Figure 4.1.1.1 Codes Screenshots- Encoder User Password1

To encode the password when registration or edition, the PasswordEncoderin in the UserService hashed the password input. Here is how the code is doing:

```java
@Service
@Transactional
public class UserService {

    @Autowired
    private UserRepository userRepo;

    @Autowired
    private PasswordEncoder passwordEncoder;

    public User save(User user) {

        boolean isUpdatingUser = (user.getId() != null);

        if (isUpdatingUser) {
            User existingUser = userRepo.findById(user.getId()).get();
            if (user.getPassword().isEmpty()) {
                user.setPassword(existingUser.getPassword());
            } else {
                encodePassword(user);
            }
        } else {
            encodePassword(user);
        }
        return userRepo.save(user);
    }

    private void encodePassword(User user) {
        String encodedPassword = passwordEncoder.encode(user.getPassword());
        user.setPassword(encodedPassword);
    }
```

Figure 4.1.1.2 Codes Screenshots- Encoder User Password2

Above is the code to encode the password before the database storing it. Here are the main steps to achieve this:

1. We inject password provider in the method of authentication provider.
2. We inject the UserService class.
3. The custom authentication provider is used to encode the password.

Apart from the user module in the Admin Management System, frontend customers' passwords are also encrypted and stored in the databases following the same way using BCryptPasswordEncoder.

4.1.2 User Authentication

This project has the User entity class, the UserRepository interface, and WebSecurityConfig class that configures custom login page. Apart from that, a DubaiShopUserDetails class implements the UserDetails interface that defined by Spring Security, and this class wraps an instance of User object. The DubaiShopUserDetailsService, which implements the UserDetailsService defined by the Spring Security, is created to perform user login authentication. Once successful authentication, a DubaiShopUserDetails instance representing the currently logged in user will be returned by Spring Security. In the DubaiShopUserDetailsService class, the *loadUserByUsername()* method *is* overrided and calls the UserRepository's *getUserByEmail()* method that returns a User based on the email. The Spring security will validate the user's password based on the returned object to perform authentication. The WebSecurityConfig class is to inform Spring Security to use DubaiShopUserDetailsService as a UserDetailsService, and the authentication provider is database-based. At the same time, some filters in the Spring Security Filter chain behind the scenes will intercept the request from the frontend page, and the request will call the UserDetailsService to perform authentication.

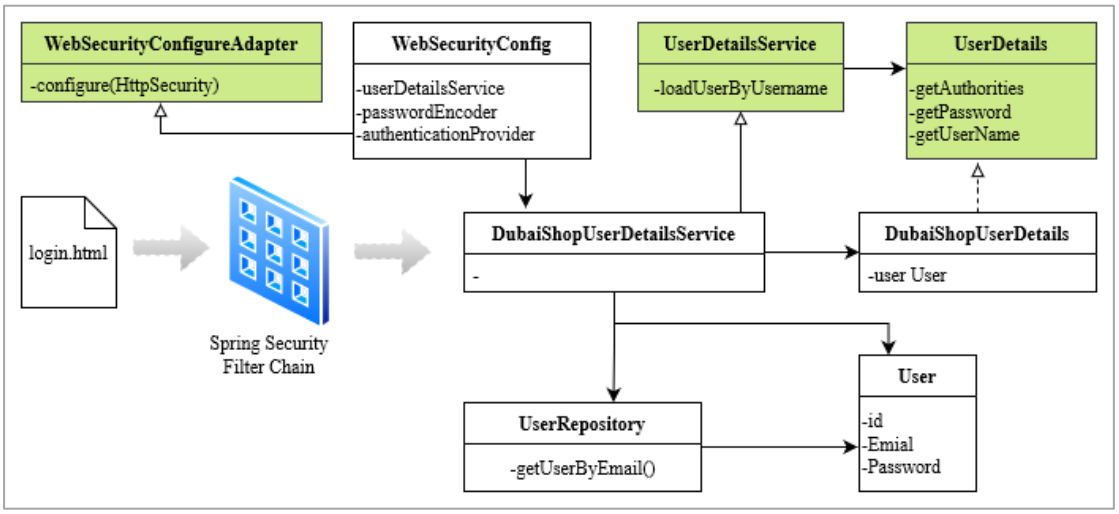


Figure 4.1.2.1 Class Diagram-User Authentication

In the Backend, different types of users have different access to certain pages. Here is the code.

```
@Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/states/list_by_country/**")
                .hasAnyAuthority("Admin", "Salesperson")
                .antMatchers("/users/**", "/settings/**", "/countries/**",
                "/states/**","/categories/**", "/brands/**", "/products/**")
                .hasAuthority("Admin")
                .antMatchers("/products/edit/**", "/products/save", "/products/check_unique")
                .hasAnyAuthority("Admin", "Salesperson")
                .antMatchers("/products", "/products/", "/products/detail/**", "/products/page/**")
                .hasAnyAuthority("Admin", "Salesperson", "Shipper")
                .antMatchers("/orders", "/orders/", "/orders/page/**", "/orders/detail/**")
                .hasAnyAuthority("Admin", "Salesperson", "Shipper")
                .antMatchers("/customers/**", "/orders/**", "/get_shipping_cost")
                .hasAnyAuthority("Admin", "Salesperson")
                .antMatchers("/orders_shipper/update/**").hasAuthority("Shipper")
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .usernameParameter("email")
                .permitAll()
                .and().logout().permitAll()
                .and()
                .rememberMe()
                .key("AbcDefgHijKlmnOpqrs_1234567890")
                .tokenValiditySeconds(7 * 24 * 60 * 60);
        ;
        http.headers().frameOptions().sameOrigin();
    }
}
```

Figure 4.1.2.2 Codes Screenshots-User Authentication1

In the Frontend, security expression “hasAnyAuthority(...)” is used to give specific access right to different users.

```
<th:block sec:authorize="hasAnyAuthority('Admin', 'Salesperson',  'Shipper')">
   <li class="nav-item dropdown">
       <a class="nav-link dropdown-toggle" data-toggle="dropdown"
          th:href="@{/products}">Products</a>
       <div class="dropdown-menu">
           <a class="dropdown-item" th:href="@{/products/new}"
              sec:authorize="hasAnyAuthority('Admin', 'Editor')">Create New</a>
           <a class="dropdown-item" th:href="@{/products}">View All</a>
       </div>
   </li>
</th:block>
```

Figure 4.1.2.3 Codes Screenshots-User Authentication2

4.1.3 Export to CSV/EXCEL/PDF

Handler methods such as *exportToCSV(), exportToExcel()* and *exportToPdf()* in the UserController is to handle the hyperlink in the frontend to export different types of files. The business classes such as UserCsvExporter, UserExcelExporter, and UserPdfExporter read the user information from a List collection of User objects and write data to a HttpServletResponse. Therefore, the fronted user can download data as CSV/EXCEL/PDF from the browser.

For CSV file, CsvBeanWriter is used, and this class is from the SuperCSV library - a free and open-source Java library to read and write CSV files (Apache, 2015). For Excel documents, some API from the Apache POI Library are used. Apache POI is

a free and open-source Java library to read and write Microsoft Office documents (The Apache Software Foundation, 2022). For PDF documents, some APIs from the Open PDF Library are used, and it is also a free and open-source Java Library to process PDF documents (Abdullah Çevik, *et al*. 2022).

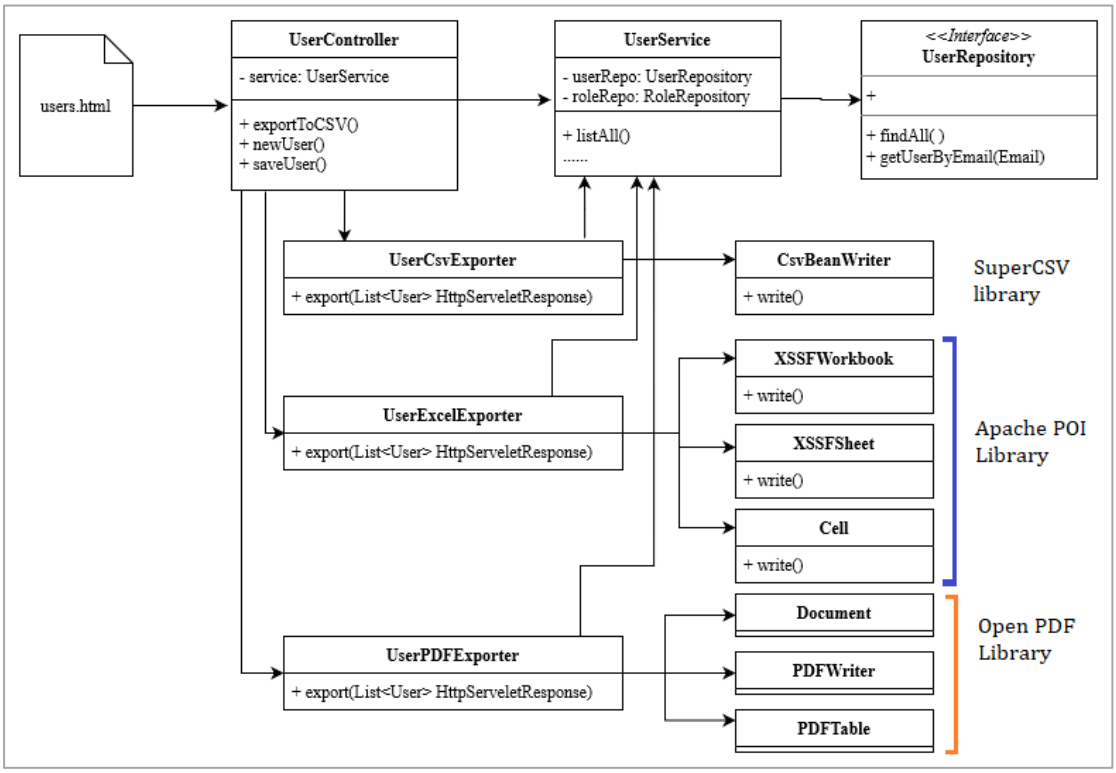


Figure 4.1.3.1 Class Diagram-Export Files

4.2 Setting Module

4.2.1 Setting Filter

In the setting module, Java filter is the most important part to apply for the entire shopping application. Using the setting filter, all requests coming into the application are intercepted by the application before being handled by any controller. A view will be dynamically updated with the setting values after the system reading the setting values from the database and putting them on the model.

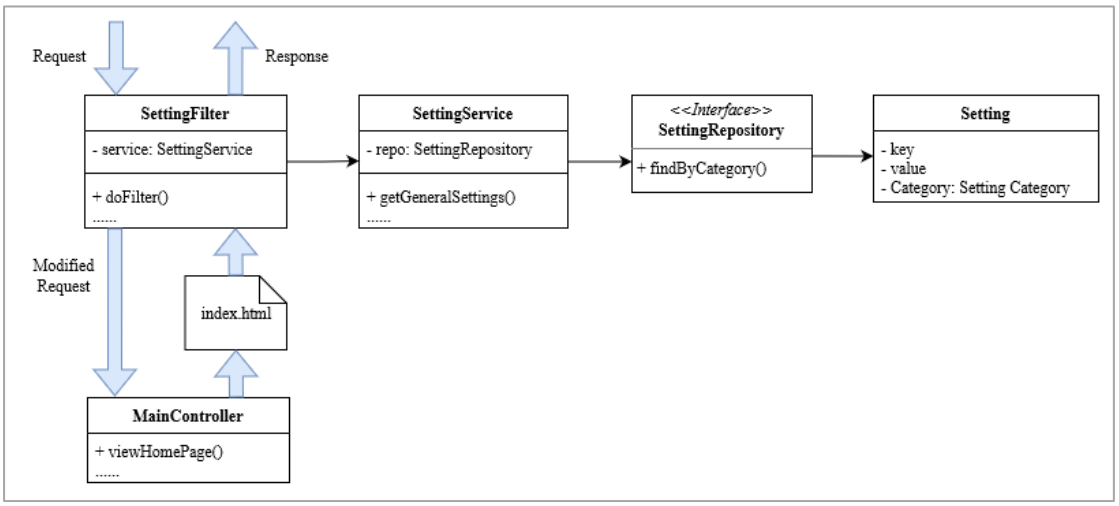


Figure 4.2.1.1 Class Diagram-Setting Filter

we use the service class to read the settings information from the database and introduce you doFilter() method. We put the setting values into a request and continues the filter chain. we have the same attitude that intercepts also requests coming to the application and the filter we modify the request by putting setting value into as our request before the requests are handled by the controllers and in the view layer. We apply the setting values for the view files before returning the response to the client by using filter.

```
@Component
public class SettingFilter implements Filter {
    @Autowired
    private SettingService service;
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain)throws IOException, ServletException {

        HttpServletRequest servletRequest = (HttpServletRequest) request;
        String url = servletRequest.getRequestURL().toString();

        if (url.endsWith(".css") || url.endsWith(".js") ||
                url.endsWith(".png") || url.endsWith(".jpg")) {
            chain.doFilter(request, response);
            return;
        }

        List<Setting> generalSettings = service.getGeneralSettings();

        generalSettings.forEach(setting -> {
            request.setAttribute(setting.getKey(), setting.getValue());
        });
        chain.doFilter(request, response);
    }
}
```

Figure 4.2.1.2 Codes Screenshots-doFilter

4.2.2 Load Country with AJAX and Restful Webservice

ALL AJAX Calls to the RESTful Webservices are performed in loading and managing Country settings.

The unique features of AJAX make it the dream of every developer, as it can be used to read data from a web server after the web page has loaded, to update a web page without reloading, and even to send data to the web server in the background (W3Schools, 2022).

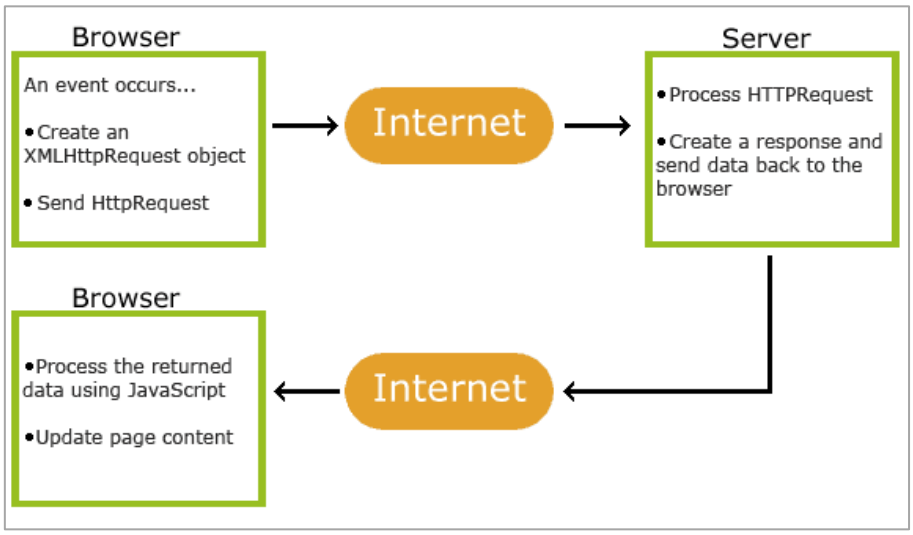


Figure 4.2.2.1 Screenshots-AJAX

The REST architectural style is an approach to designing Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP using a variety of clients. The REST protocol requires developers to use HTTP methods explicitly and in a consistent manner. The basic REST design principle establishes a one-to-one mapping using *POST* annotation to create a resource on the server, using *GET* annotation to retrieve a resource, using *PUT* annotation to update the resource, and using *DELETE* annotation to remove a resource (Alex Rodriguez, 2015).

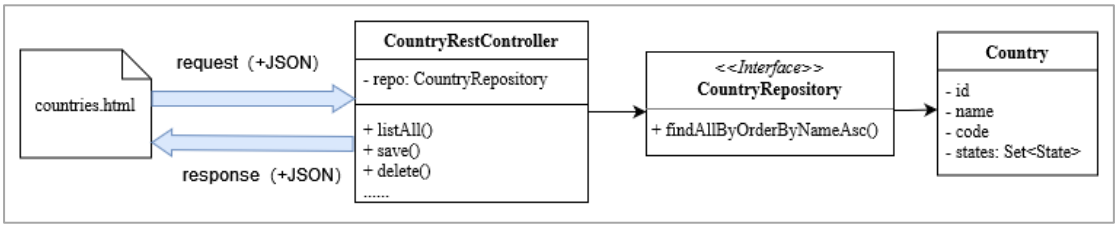


Figure 4.2.2.2 Class Diagram-Load Countries

In the “DubaiShop” project, some Restful Webservices are used by the client with some JavaScript and jQuery codes to send AJAX requests to perform the RESTful Webservices. And data will be sent along with the requests and responses when necessary. When the settings pass is loaded, it does not immediately load the countries since there are more than 200 countries in the database. The country list will be only loaded when the user clicking the “Load countries” button. Country data is used in customers module, address module, shipping rates module, and checkout module.

```
$.ajax({
    type: 'POST',
    url: url,
    beforeSend: function(xhr) {
        xhr.setRequestHeader(csrfHeaderName, csrfValue);
    },
    data: JSON.stringify(jsonData),
    contentType: 'application/json'
}).done(function(countryId) {
    $("#dropDownCountries option:selected")
    .val(countryId + "-" + countryCode);
    $("#dropDownCountries option:selected")
    .text(countryName);
    showToastMessage("The country has been updated");

    changeFormStateToNewCountry();
}).fail(function() {
    showToastMessage("ERROR: Could not connect to server");
});
```

Figure 4.2.2.3 Codes Screenshots-AJAX

In Spring, the objects are automatically converted into JSON data which is sent to the client. For creating and updating operation, a *@PostMapping* annotation is performed. For HTTP posting requests, the *@requestBody* annotation is used to convert JSON data from the client into a Java object. And for delete operation, the *@DeleteMapping* annotation is used for HTTP delete method.

```
@RestController
public class CountryRestController {

    @Autowired private CountryRepository repo;

    @GetMapping("/countries/list")
    public List<Country> listAll() {
        return repo.findAllByOrderByNameAsc();
    }

    @PostMapping("/countries/save")
    public String save(@RequestBody Country country) {
        Country savedCountry = repo.save(country);
        return String.valueOf(savedCountry.getId());
    }

    @DeleteMapping("/countries/delete/{id}")
    public void delete(@PathVariable("id") Integer id) {
        repo.deleteById(id);
    }
}
```

Figure 4.2.2.4 Codes Screenshots-RESTful Webservices

4.3 Cutomer Module

4.3.1 Customer Authentication

Customer authentication in the customer module follows the same logic with the feature of user authentication in the Admin Management System.

A DubaiShopUserDetails class implements the UserDetails interface that defined by Spring Security, and this class wraps an instance of User object. The CustomerUserDetailsService, which implements the UserDetailsService defined by the Spring Security, is created to perform customer login authentication. Once successful authentication, a CustomerUserDetails instance representing the currently logged in customer will be returned by Spring Security. In the CustomerUserDetailsService class, the *loadUserByUsername()* method is overrided and calls the UserRepository's *getUserByEmail()* method that returns a Customer based on the email. The Spring security will validate the customer's password based on the returned object to perform authentication. The WebSecurityConfig class is to inform Spring Security to use CustomerUserDetailsService as a UserDetailsService, and the authentication provider is database-based. At the same time, some filters in the Spring Security Filter chain behind the scenes will intercept the request from the frontend page, and the request will call the UserDetailsService to perform authentication.

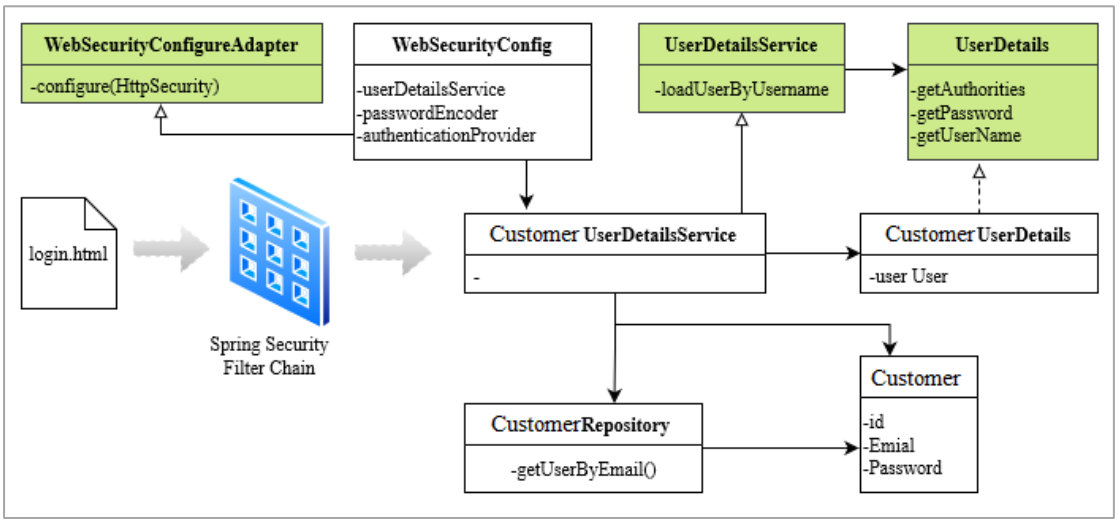


Figure 4.3.1.1 Class Diagram-Customer Authentication

Codes of customer authentication is adapted from Admin Management System's user authentication module. Therefore, no further elaboration here.

4.3.2 Google and Facebook Login

In the "DubaiShop" project, OAuth2.0 standing for "Open Authorization" is used to help visitors register or login as a customer. Authorization is carried out using the OAuth 2.0 protocol, which is considered the industry standard. Web, desktop, mobile, and living room applications can all benefit from OAuth 2.0, which

emphasizes client developer simplicity (IETF OAuth Working Group, 2013).

Client credentials for Google OAuth2 authentication can be obtained from the Google API Console. In the "Credentials" section, "OAuth2 Client ID" for "DubaiShop" web application is created. As a result, Google sets up a client id and secret for the developer.

In the coding section, the customerOAuth2Users class inherited from OAuth2User class defined by Spring Security, and it is a client library to represent user authenticated through OAuth protocol. In addition, customerOAuth2Users class also called the customerOAuth2UserService class which inherited from the DefaultOAuth2UserService. This class uses a load user method which will be invoked by Spring Security OAuth client library upon the OAuth to authorization.

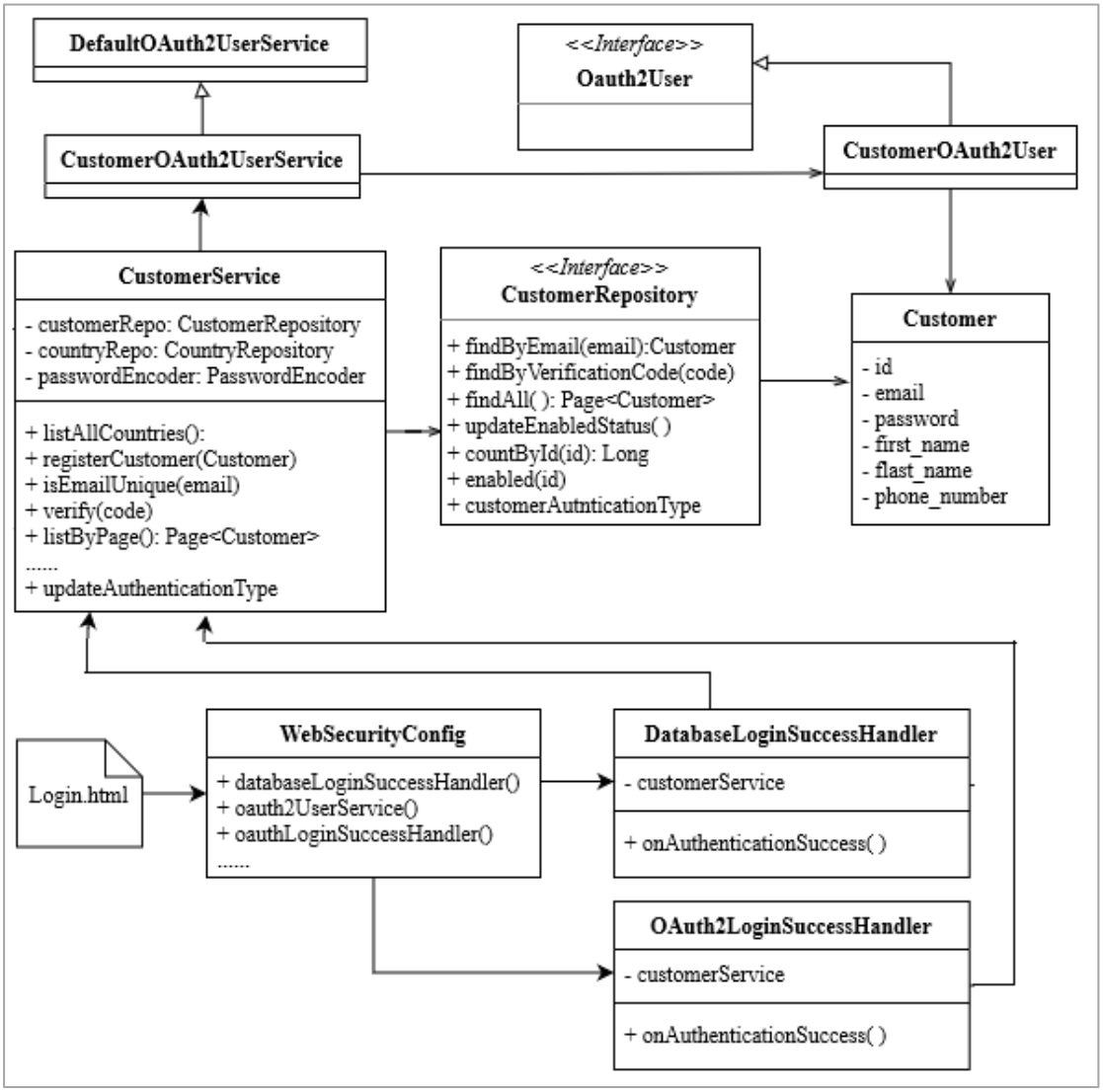


Figure 4.3.2.1 Class Diagram-Google Login

4.3.3 Customer E-mail Verification

At the time submit the signup form, a visitor will receive a verification code, and the visitor should have to click on the verification hyperlink in order to activate his personal account and to become a customer. Then the visitor can login and making payment.

The User entity class has two fields, verificationCode and enabled. The verificationCode is a unique string generated randomly during the registration process and is used in the verification process. The CustomerUserDetails class overrides the *isEnabled()* method which returns the user's enabled status. Therefore, if the visitor has not verified his account, the Spring Security will inject login. The UserServices class uses the JavaMailSender and have *register()* and *sendVerificationEmail()* methods to help send an verification email to the visitor's inbox, with the verification hyperlink. In case a user has not activated his registered account by clicking on the verification link embedded in his email, his enabled status will be false, and the user will not be able to log in (Nam Ha Minh, 2020).

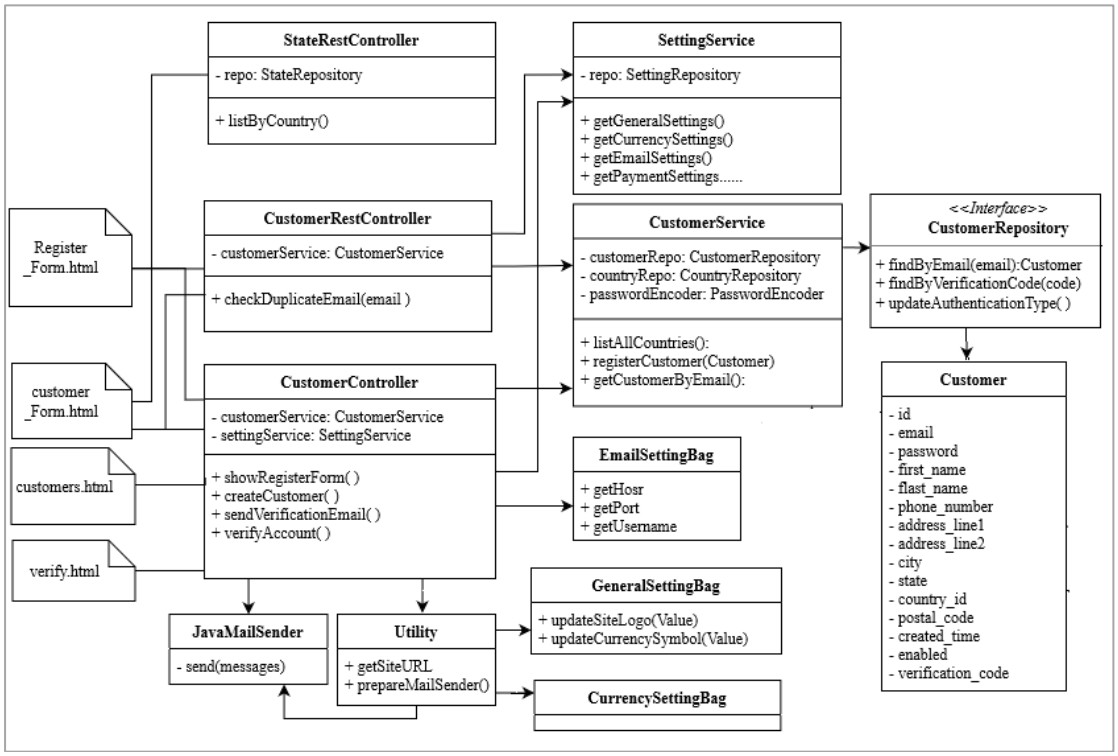


Figure 4.3.3.1 Class Diagram-E-mail Verification

4.4 Checkout Module

4.4.1 Checkout and Send Order Confirmation E-mail

The CheckoutService implements the business methods in Checkout Controller

such as *prepareCheckout()*, *calculateProductCost( )*, and *calculateShippingCost( )*. To validate the order, the PapalService class is used for help, and Paypal function will be discussed in the later paragraph. The Chckout Controller implements the handler methods such as *processPaypalOrder()*, *showCheckoutPage()*, and *placeOrder().*This Checkout Controller uses several services classes, such as CustomerService class, OrderService class, AddressService class, SettingService class, ShoppingCartService, ShippingRateService, and CheckoutService class. The CheckoutInfo presents the checkout information in the view layer, such as Checkout page and ShoppingCart Page.

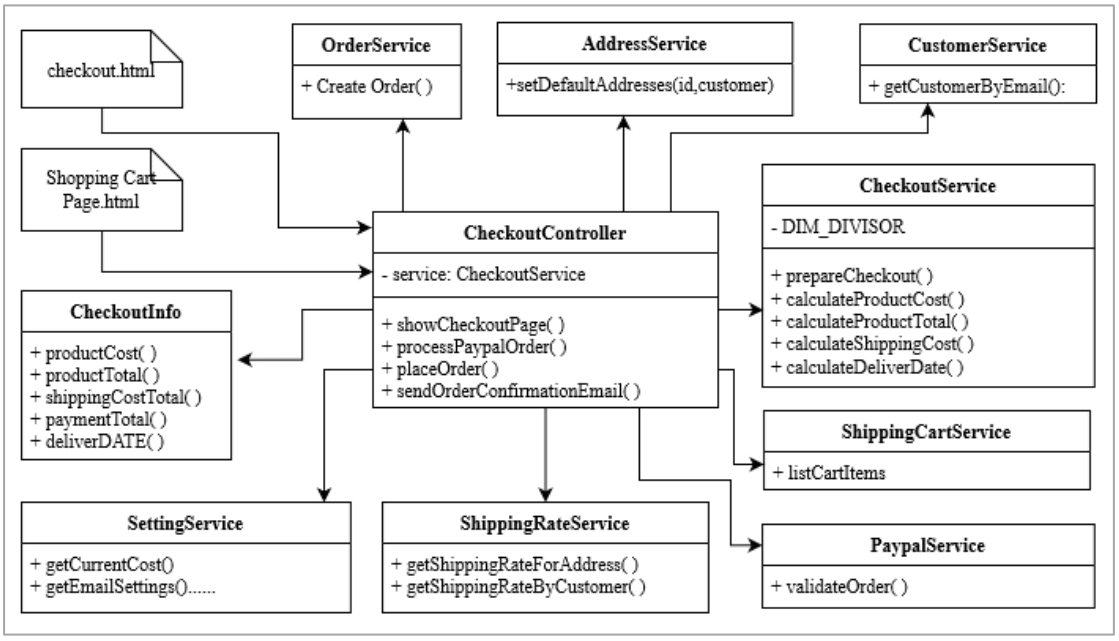


Figure 4.4.1.1 Class Diagram-Checkout

When coding order E-mail confirmation, EmailSettingBag class is used to get the confirmation subject and the content stored in the databases. The SettingService class gets currency settings which details the using currency set by the Admin management System, and help determine the order currency unit in the confirmation E-mail.

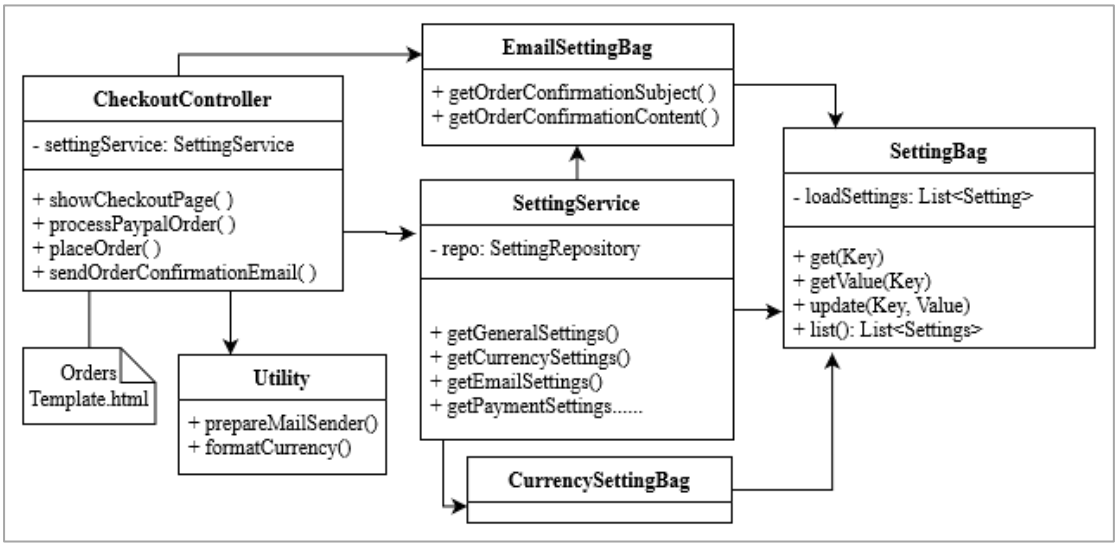


Figure 4.4.1.2 Class Diagram-Order Confirmation

4.4.2 Paypal Service

Get the client id and client secret of “DubaiShop” application by creating a PayPal developer account and creating an app in PayPal's sandbox. The PaypalOrder Response class is to map some values in the JSON response to a Java object with fields ID and status, and it also remains *validateOrder()* method in the Json response. The PaypalService class includes the *validateOrder()* method which will call the Checkout Controller class for validating a Paypal order. In the PaypalService class, the *@component* annotation is used instead of the *@service* annotation, because this class reference to the SettingService class to get the payment settings. The Checkout Controller is responsible for handling Paypal orders, which are invoked after payment is processed by the checkout-based wrapper

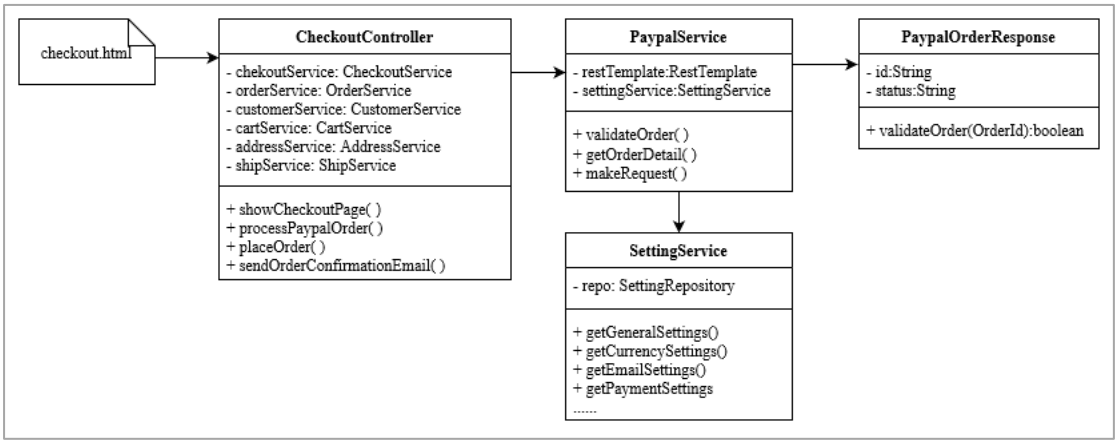


Figure 4.4.2.1 Class Diagram-PayPal Service

4.5 User Interface

4.5.1 Shopping Home

When visitor open his browser and open “DubaiShop” Website, The first thing go into his eyes is the shopping home page. Logo of the website is at the left-up corner, and there are five columns in the navigation bar which are Shopping, Payment, Shipping & Delivery, Login, and Register. At the center of this page, different categories of products was displayed in rows. If the visitor clicks the category icon or the category name under the icon, he will get into product page.

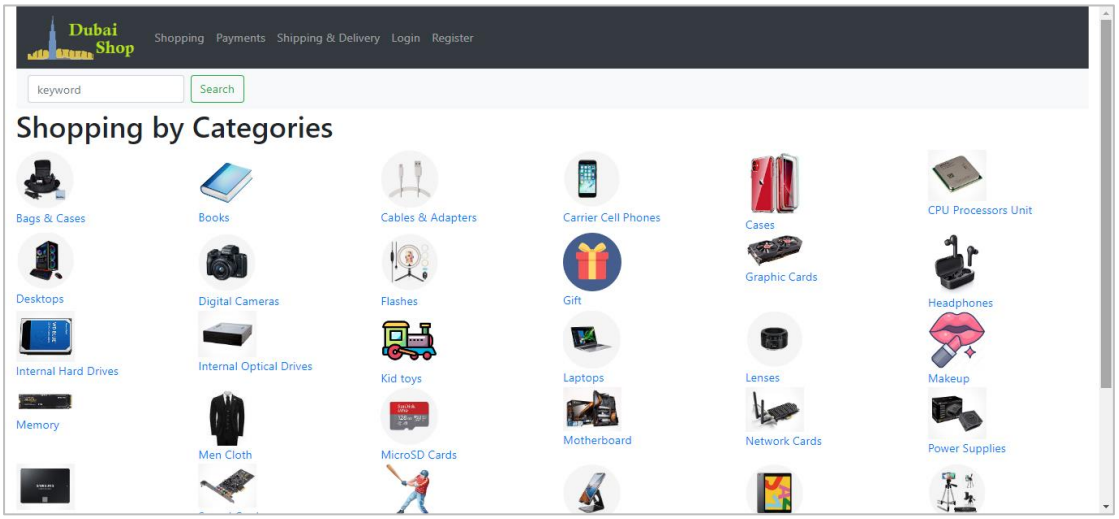


Figure 4.5.1.1 UI- Shopping Home

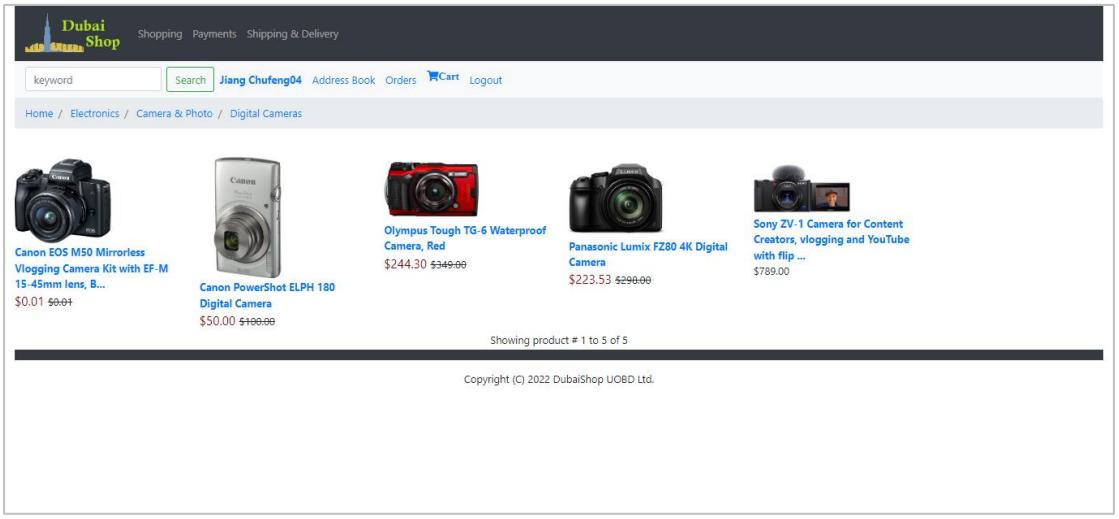
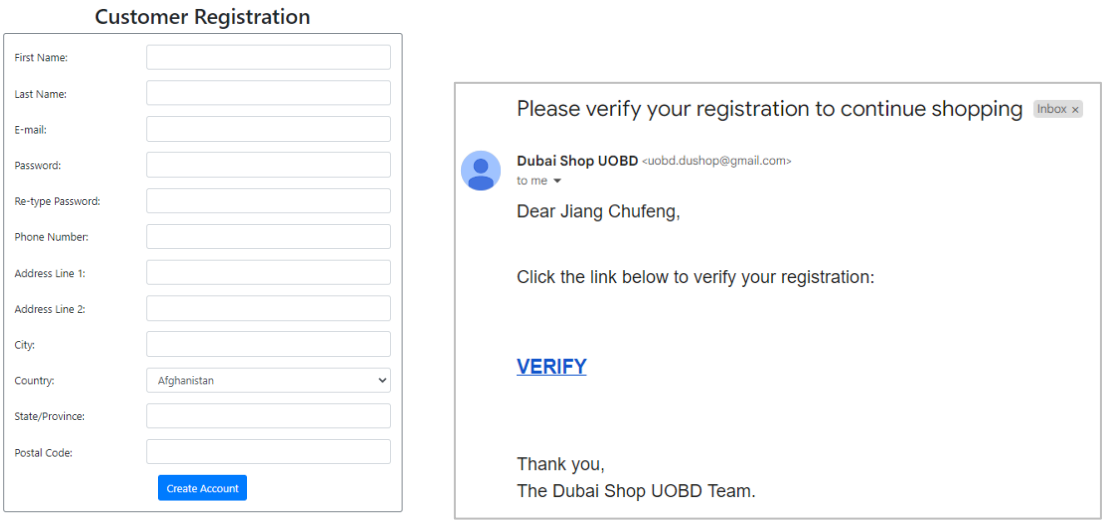


Figure 4.5.1.2 UI- Products in Category

4.5.2 Visitor Register and Customer Login

When the visitor clicks “Register”, he will get into the register page, and the register form will display in the screen. In this form, only the “Address Line 2” can be left empty, other blankets must be fill in completely, otherwise the “Create Account” button will not work.



(a) Customer registration form (b) Verification E-mail

Figure 4.5.2.1 UI- Customer Registration

Once the visitor fills out the form and click the "Create Account" button, a customer verification E-mail will be sent to his inbox. When the visitor clicks the "Verify" hyperlink in the confirmation letter, he will be redirected to the shopping website and a message will notify him that he has been registered successfully.

Once verified the account, the customer can login the website using the email address and password. Apart from register by email, the customer can login using his Google and Facebook account directly.

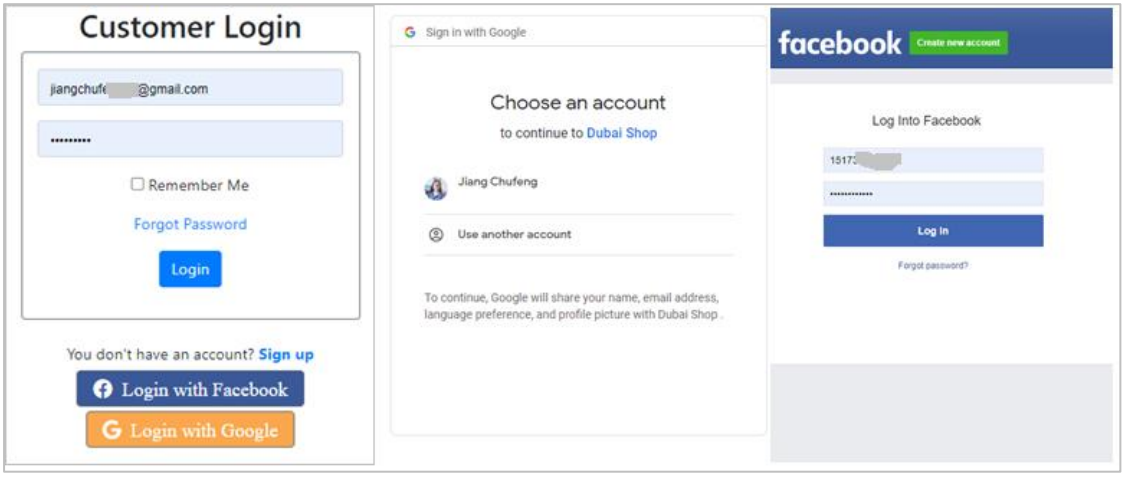


Figure 4.5.2.3 UI- Customer Login

4.5.3 Product Details

This is the Product display webpage. In this page, the product's full name, description, specification, and images from different sides are included. If the customer clicks the main image of the product, the slide show will pop up. By clicking the arrows in the two sides, customer can look through the product into detail from different angles. If the customer is satisfied with the product, he can click the "Add to cart" button to put the selected product into shopping cart.

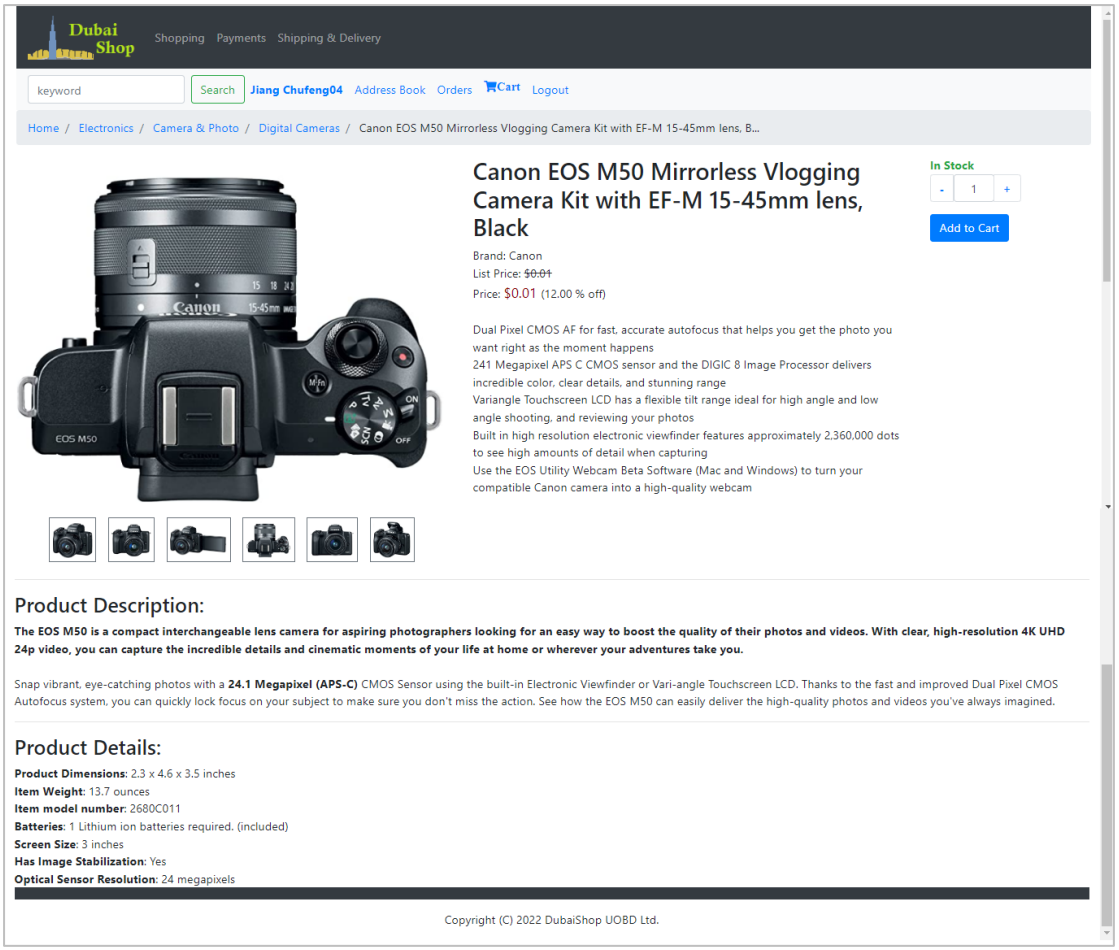


Figure 4.5.3.1 UI-Product Details

Figure 4.5.3.2 UI-Product Slide Show

4.5.4 Shopping Cart and Checkout

By clicking the “Cart” icon in the menu bar, the customer can jump to the shopping cart directly. In the shopping cart page, the customer can edit the

quantity of products, and remove the unwanted products. Once decided the products to buy, the customer can click the “Check Out” button, and the web page will redirect to the checkout counter

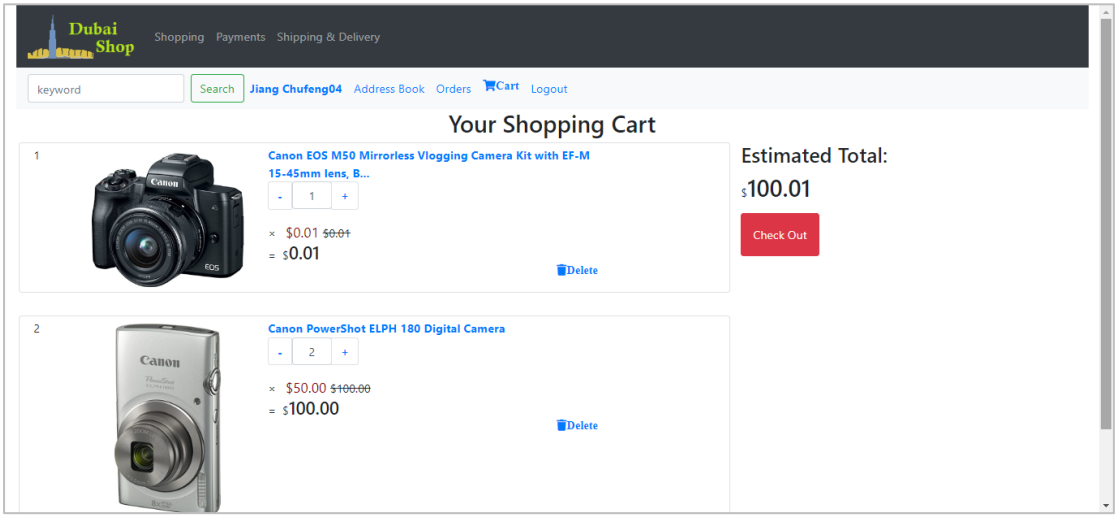


Figure 4.5.4.1 UI-Shopping Cart

In the checkout counter, customers shipping information is displayed, and the order’s summary and costs are displayed at the right side of the screen. At the bottom of the page, customers can choose the payment method. If he chooses the “Cash on Delivery”, the order will be placed immediately. If he chooses “PayPal” or “Credit Card”, a new window will pop up requiring further payment information.

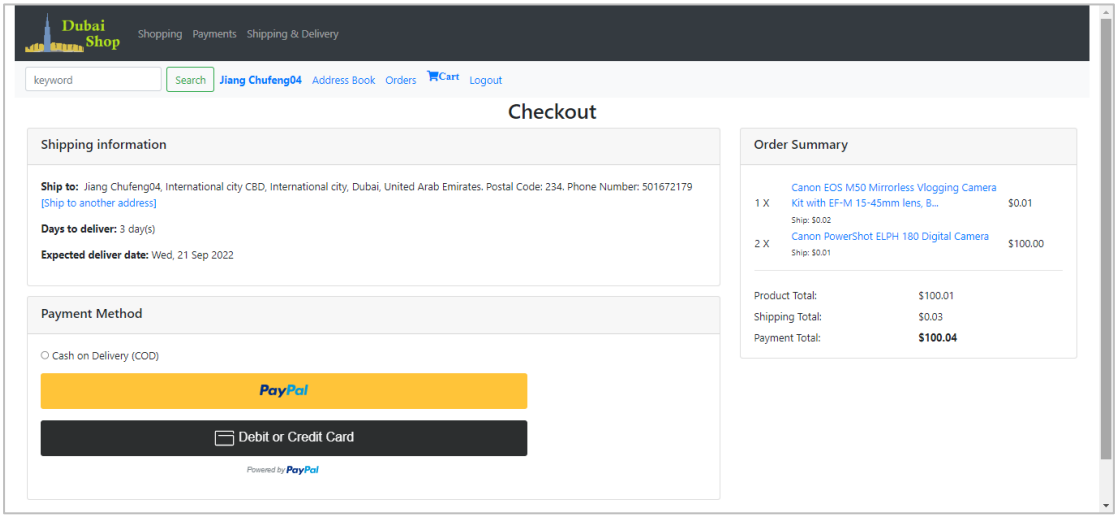


Figure 4.5.4.2 UI-Checkout

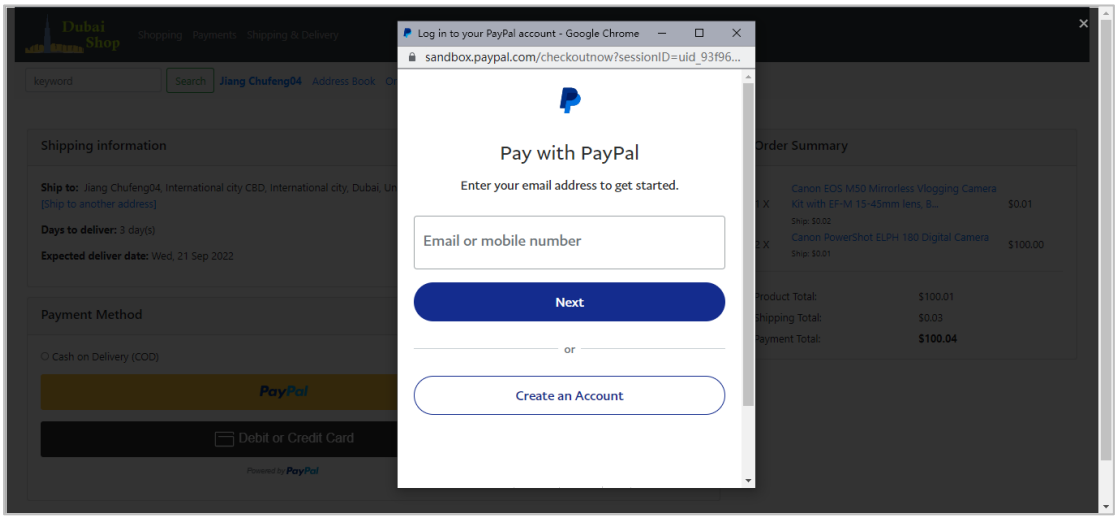


Figure 4.5.4.3 UI-PayPal

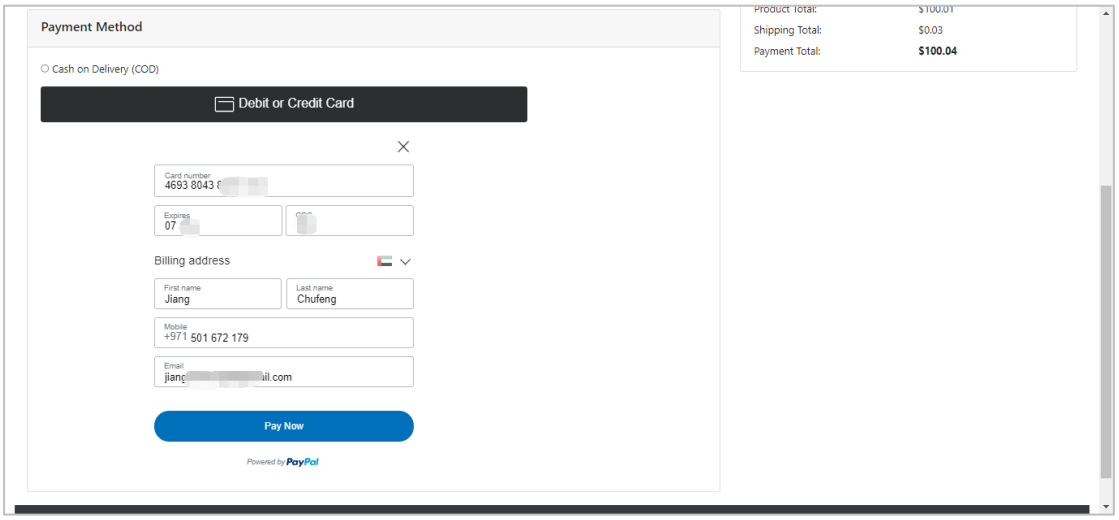


Figure 4.5.4.4 UI-Credit Card

Once the payment information is filled properly and the customer clicks the “Pay Now”, the payment will be processed through the server. If the payment was successful, an order confirmation will be sent to the customer’s inbox. Otherwise, a notification will pop up indicating the failure in payment.

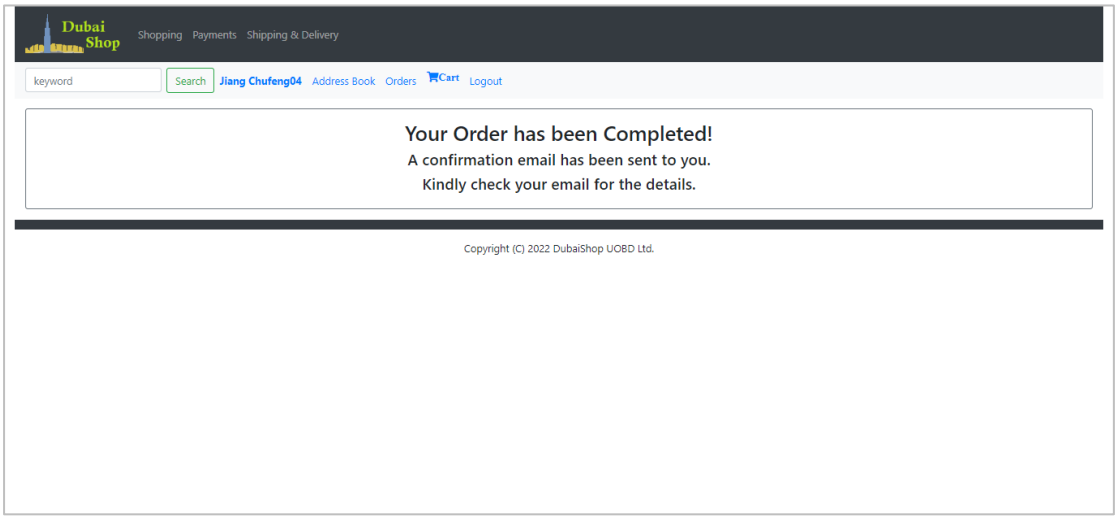


Figure 4.5.4.5 UI-Order Notification

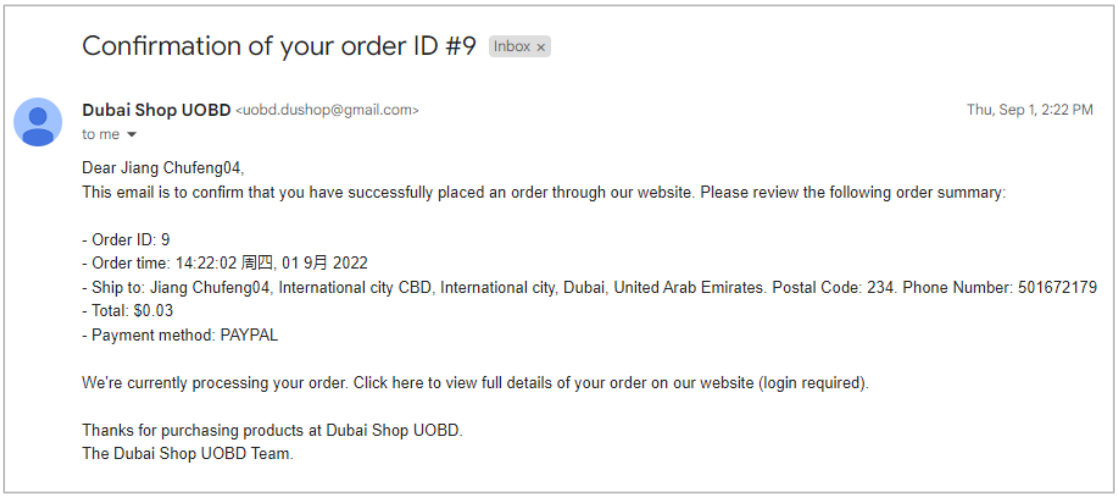


Figure 4.5.4.6 UI-Order Confirmation

4.5.5 Admin User Login

Users of the Admin Management System use another portal login the Control Panel. Different User have different interface after login.

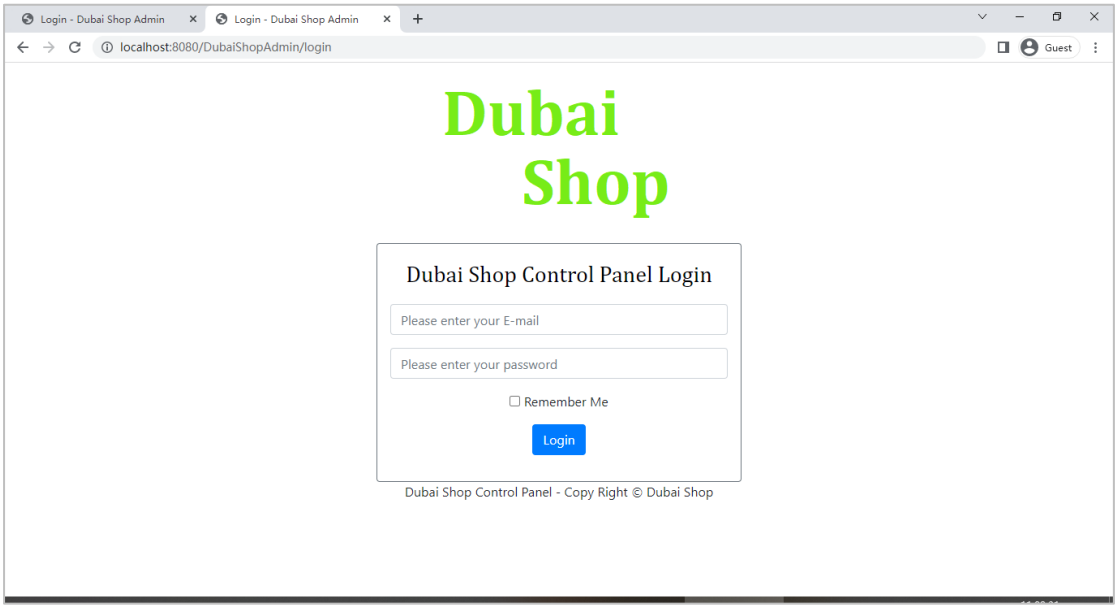


Figure 4.5.5.1 UI-Admin User Login

As shown in the **Figure 4.5.5.1**, Admin user have all access to all modules, while shipper and salesperson have access to some of the modules rather than full access. However, if a limited-access role is granted another role at the same time, he can access the combination of modules from each role. For example, if a shipper is upgraded as an Admin, he will be granted the full access right to every module.

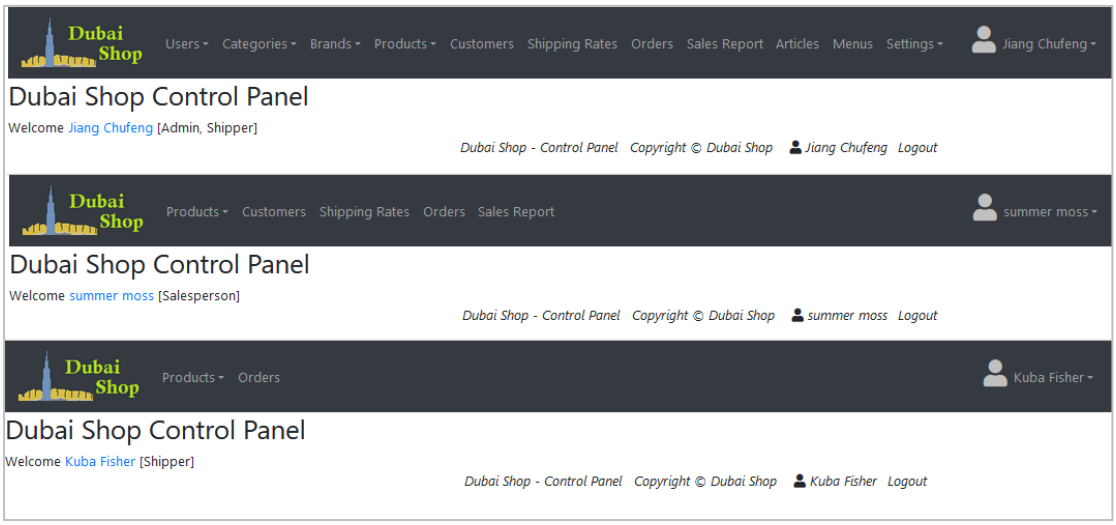


Figure 4.5.5.2 UI-Admin Navigation Bar

4.5.6 Products Management

Products are displayed in rows, and for each product, the full name, main image, brand and category of this product can be seen from the main page. In the last column, the functional button of “view details”, “edit” and “remove” are displayed in order. In this main page, the admin and salesperson can add new products to the list and can search a certain product using the “filter” search box.

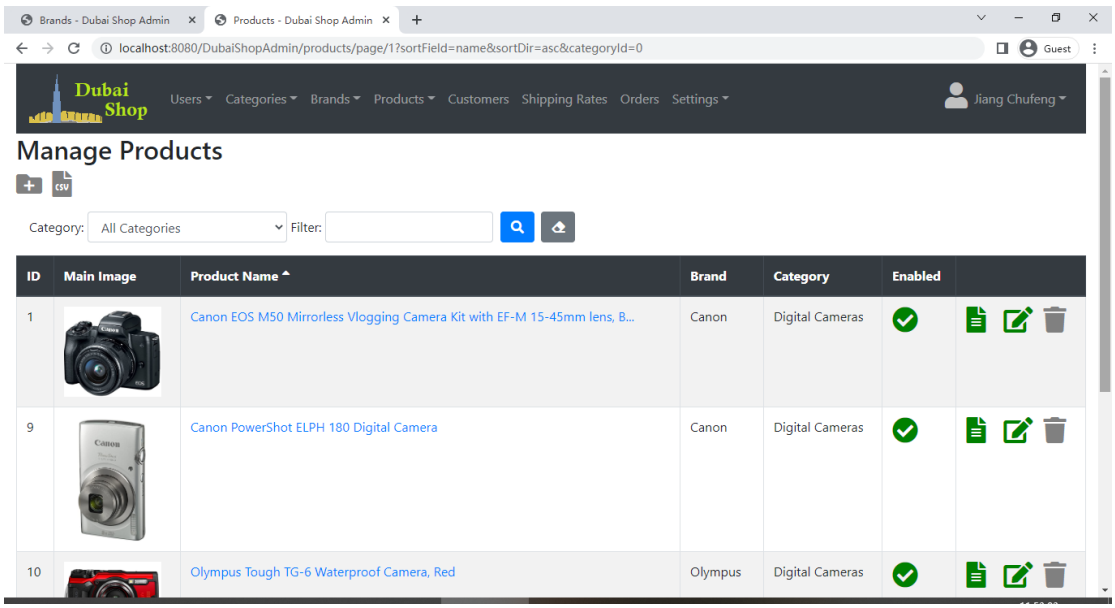


Figure 4.5.6.1 UI-Product List

When the user clicks the “edit” button, the user can edit the products full information such as products specification, description, shipping information, etc. In the description part, the rich text editor was embedded to help manage the product descriptions which is stored in the database shared by the Admin Management System and Customer Shopping System.

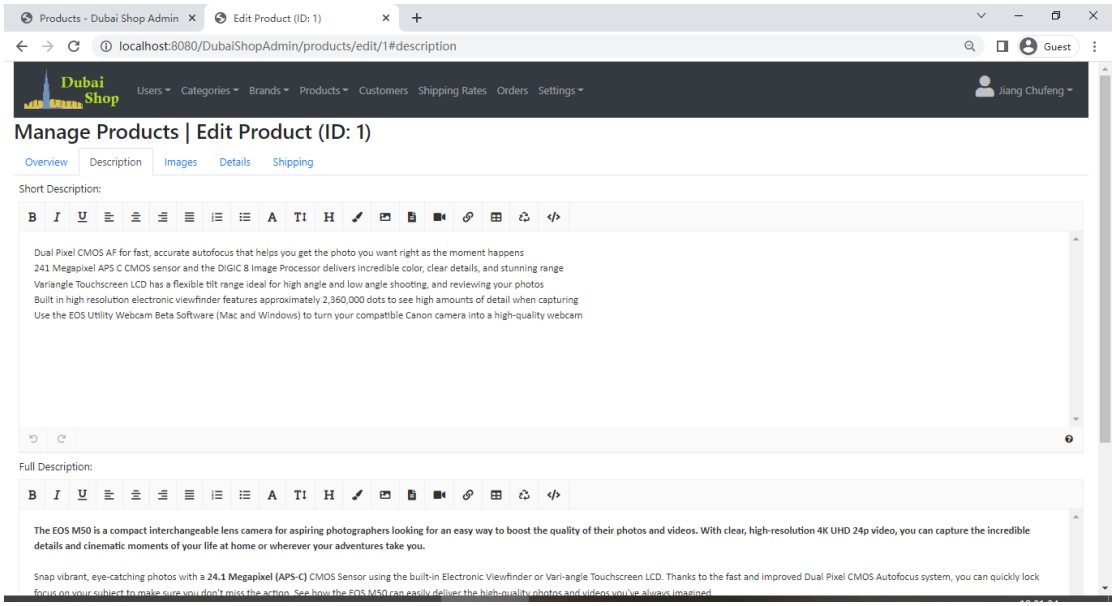


Figure 4.5.6.2 UI-Product description

4.5.7 Settings

In the settings management module, there are several sub-pages using for managing different functions. In the general settings, the user can manage the websites logo, name currency type and so on.

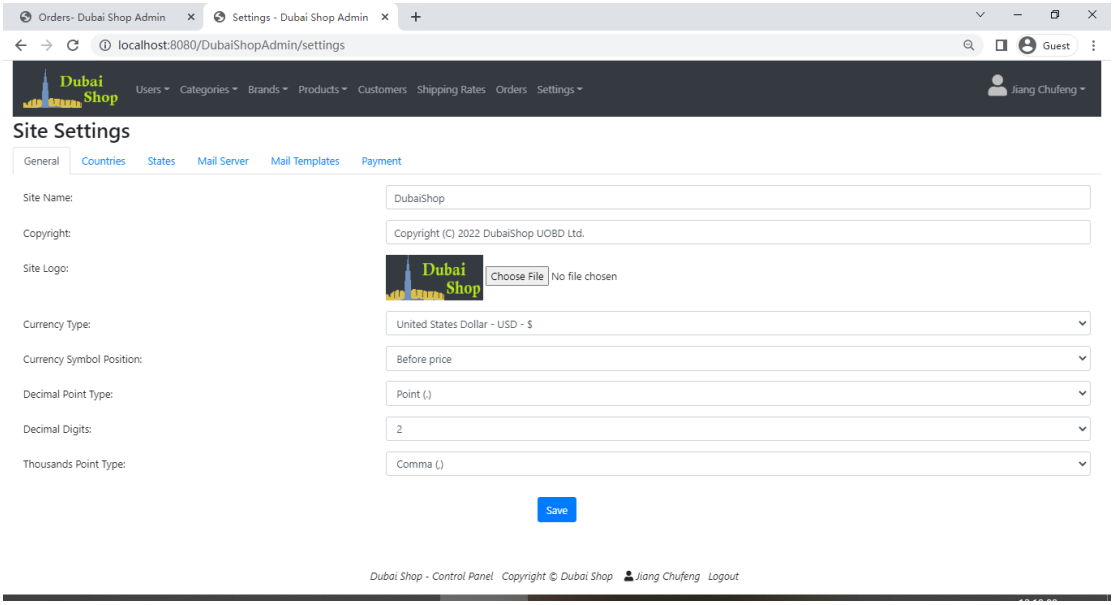


Figure 4.5.7.1 UI-Settings

Apart from general settings, countries and states information are management in this module. When clicking “Refresh country list”, the full list of country and stated will appear, and the user can add, remove, and edit a specific country or state.

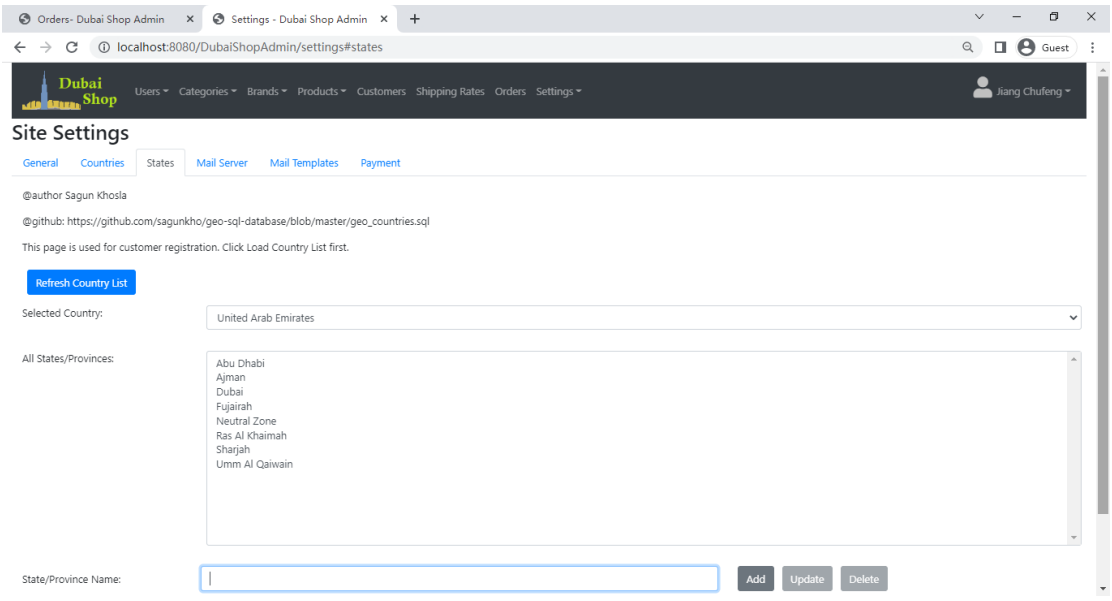


Figure 4.5.7.2 UI-Country and States Settings

Other settings such as customer verification letter template, order confirmation template, mail server, and PayPal setting can be managed in this module. All of the information is stored in the database.

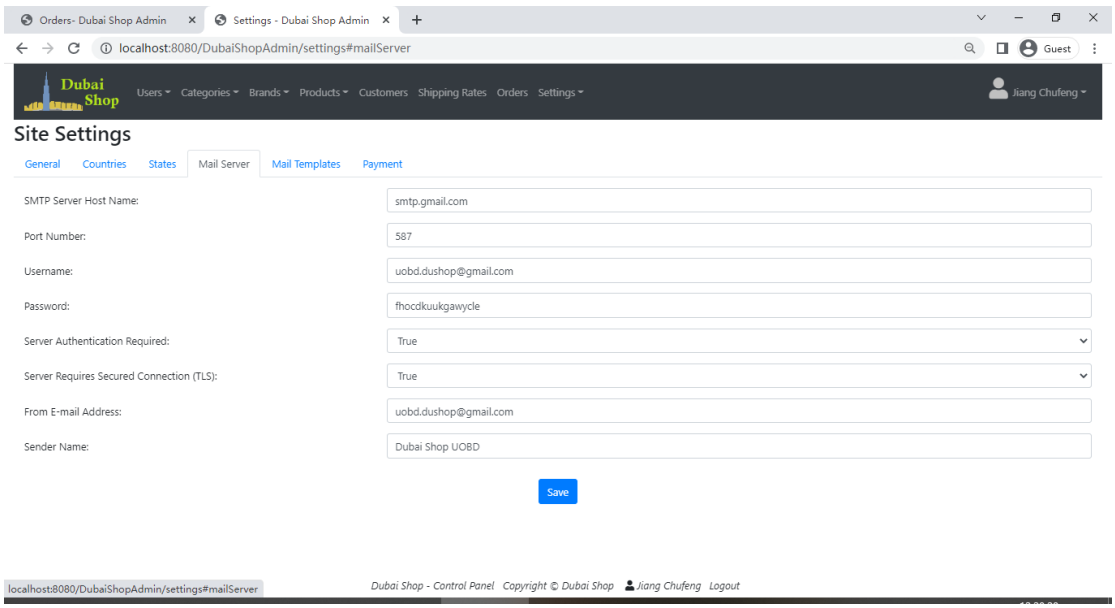


Figure 4.5.7.3 UI-Mail Server Settings

4.6 Validation and Testing

In this section, I am going to discusses how the above implementation was tested and evaluated to ensure the "Dubai Shop" project runs without error. The validation and testing are related to the use of unit test to confirm that the functions' behavior matches what was expected and generating and following test routines to make sure that the project components work together properly. The following table shows the results of main functions have been tested.

By validation and testing, one out of twenty-three test case was failed due to unknown reason. It is might because the user's PayPal Account has been long unused, and the account has been frozen.

Test Case ID	Test Description	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
B-1-1	Verify Login with Valid E-mail and Password	1.Enter valid E-mail and Password.2.Click "login" button.	E-mail:cxj164@bham.ac.uk Password:123456789	User can login and enter the system.	Login succesful	Pass
B-1-2	Verify Login with Invalid E-mail and Password	1.Enter invalid E-mail and Password.2.Click"login" button.	E-mail:cxj164@gmail.com Password:123456789	User can not login.	Login rejected	Pass
B-2-1	View Control Panel Module Information	1.Click every navigation bar menue items		The system should display correct item lists	The system displayed correct lists of items.	Pass
B-2-2	Add new items to each module	1.Click the"Add New" Button		The system should show correct blank froms.	The system showed correct blank froms.	Pass
B-2-3	Export CSV/EXCEL/PDF	1.Click the export icons under the menu bar.		Files should be auto downloaded.	Files were auto downloaded.	Pass
B-2-4	Upload images for users, products, brands, categories, Site logo	1.Click "choose file"or "upload image" button.2.Click "save" button.		The images should be displayafter uploaded, By saving and returning to the list form, old images should be replaced by the new one.	New images were displayed in the frontend.	Pass
B-2-5	Update products description.	1.Select a product and click "edit" button. 2. Modify the description in the rich text tool box. 3. click "save".		Product description should be changed.	Product description changed.	Pass
B-2-6	Search a exist product	1.Type a procuts name in the filter text area. 2.Click "search" button.	Type "Camera"	Products should be displayed if the name including the key word.	All camera related products were displayed.	Pass
B-2-7	Search a non-exist product	1.Type a procuts name in the filter text area. 2.Click "search" button.	Type "Sands"	The web page should indicate no such product.	A notification of "no such product" displayed.	Pass
B-3-1	Load country List	1.Click"Refresh Countries"		Country list should be displayed.	Country list displayed.	Pass
B-3-2	Load State List	1.Click"Refresh Countries"		State list under a certain country should be displayed.	State list displayed.	Pass
F-1-1	Customer Register	1.Click Register. 2.Fill in the register form and submit.	E-mail: jiangchufeng1@gmail.com Password:123456789	Customer should receive an verification E-mail	An verification E-mail received.	Pass
F-1-2	Customer verification	1.Click the verify link in the verification E-mail		Webpage redirect to the website and a notification of successful registered will be displayed.	Verify link was working, and the website inditatied successfully registered.	Pass
F-2-1	Customer valid account login	1.Click "login" in the navigation bar. 2.Input E-mail and password in the login form. 3. Click "login" button.	E-mail: jiangchufeng1@gmail.com Password:123456789	Cutomer login successful	Login succesful	Pass
F-2-2	Customer invalid account login	1.Click "login" in the navigation bar. 2.Input E-mail and password in the login form. 3. Click "login" button.	E-mail: jiangchu123@gmail.com Password:123456789	Cutomer login reject	Login rejected	Pass
F-2-3	Cutomer login using Google account	1.Click "Login" in the navigation bar. 2.Click "Google Login".	Personal Google Account	Google login window should pop up. After input google E-mail and password, the webpage should redirected to the shopping website.	Login succesful	Pass
F-2-4	Cutomer login using Facebook account	1.Click "Login" in the navigation bar. 2.Click " Facebook Login".	Personal Facebook Account	Facebook login window should pop up. After input Facebook E-mail or phone number and password, the webpage should redirected to the shopping website.	Login succesful	Pass
F-3-1	Add products to the shopping cart	1. Click "Add to Cart" button in the product detail display webpage	Add Canon Camera to the Shopping Cart	Products should be displayed in the Shopping Cart.	Canon Camera was in the Shopping Cart	Pass
F-3-2	Checkout using Cash On delivery	1.Click "Checkout". 2. Choose "COD" in the payment method column.		The webpage should indicate the order has been placed. An order confirmation should be sent to the customer's account.	The webpage showed the order has been placed. A order confirmation received in the E-mail inbox.	Pass
F-3-3	Checkout using PayPal	1.Click "Checkout". 2. Choose "PayPal" in the payment method column.	Personal Paypal Account	PayPal login window should pop up. After make Payment, the webpage should indicate the order has been placed, and an order confirmation should be sent to the customer's account.	PayPal Login window pop up. However, customer cannot login due to unknow reason.	Fail
F-3-4	Checkout using Credit Card	1.Click "Checkout". 2. Choose "Credit Card" in the payment method column.	Personal Credit Card Information	Credit Card information form window should be diaplayed. After complete that form and submit, the webpage should indicate the order has been placed and an order confirmation should be sent to the customer's account.	Credit Card information form displayed. After submit the credit card information, a notification of "order has been placed" came out. The order confirmation has been received.	Pass

Appendix-F: Figure 4.6 Validation and Testing Record

4 Evaluation

5.1 Requirement Evaluation

"DubaiShop" is a full-stack Web Application for personal companies to running and operating.

For its Admin Management System, there are three roles, and each role has different responsibilities. All the date related to the products, users, customers, and website settings are from the shared database which can be managed by the administrator himself. The Admin Management System can manage its products, users, customers, and website settings by "add", "edit", "remove", and "view" operations. Under the help of APIs from JavaMail, Google, Facebook, and Paypal, some advanced features have been achieved.

For its Customer Shopping System, there are two roles of which the visitor can transit to be a customer after registration. In the shopping website, customers can select the products, manage the Shopping Cart, place orders and make payment using different methods.

5.1.1 Overview Testing objectives:

1. Validate the functional and non-functional requirements of “Dubai Shop” Web Application under test requirements.
2. Make sure the “Dubai Shop” Admin management system conforms to the quality specifications defined by the owner company.
3. It is essential to identify and fix critical bugs/issues before the “Dubai Shop” goes live.

Test plans are primarily designed to ensure the integrity of the “Dubai Shop” project's functional and non-functional requirements, as well as to test the program to find bugs and errors, and therefore to ensure that the system is reliable and extensible.

5.1.2 Module Testing objectives:

1. The systems should be tested on different browsers (including Firefox, Chrome, IE and other browsers).

2. This test covers 21 test cases, including testing function requirements and non-function requirements.

   The tested function requirements are as follows:
   a. User and customer login the system, view the items list function, edit or remove the item function.
   b. Message notification function.
   c. Place order function.
   d. Checkout and make payment using different method functions.
   e. Received confirmation and verification E-mail functions.

   The tested non-function requirements are as follows:
   a. System’s security function rejecting unknow user to access.
   b. How easy to use the system regarding Admin system and Customer shopping system.

5.1.3 Reliability

“Dubai Shop” will not be crash if the application running on the local machine without interruption. Even with emergency such as unexpected exit from the frontend, the website can auto save the customer’s shopping cart details and store the information into the database. While longtime no-operation, the website will force the user or customer logout without crash.

5.1.4 Scalability

“Dubai Shop” is a project with its the backend system and the frontend system can be expanded. This project is developed using the Spring Boot framework, and it is organized by functional modules in each system. Developers can add new modules and features into this system without modifying the system’s

architecture. If the developer wants to design a product review function, he can simply create the common entities in the Common Entity module and embed the reviews module into the backend system and the frontend system.

5.1.5 Maintainability

"Dubai Shop" is easy to maintain since it's a simple structured Spring Boot project. Its functions and related codes are designed in a modular way, which making this project easy to identify and correct functional faults in the project that are preventing it from functioning as intended.

5.2 Limitation and future development

There are some limitations in this project that can be optimized for future development:

1. It's a multi-currency shopping website, however, once the administrator set the currency, the end users can only user the fixed currency unit.
2. As a shopping website, it does not have the "Question" and "Review" modules like Amazon, Carrefour, and other shopping websites. Customer can not put their questions and view the comments for the products.
3. Paypal Payment method seems problematic while it is unclear for the reason since a new PayPal account is not able to login when choosing PayPay to make payment.
4. Data safety issues arise if the database receiving malicious attacks.
5. No inventory monitoring function. Customers is not able to know whether the product has sufficient inventory or not. That will bring extra work to the salesperson to check with the warehouse.

5 Summary and Conclusions

In summary, this project is a full-stack Web Application, which contains two main sub-application, the Admin Application and the Shopping Application. This project has implemented basic functions similar to those of other real-life online shopping websites.

There are still a lot of features can be developed in the project, such as Sales Report feature, Reviews feature, and Questions feature. Apart from that, this project is running in the local machine, and future work like deploy it in the cloud platform can be done after the project is doing better.

Through this project I have obtained full-stack developing experience in designing and coding a real-life project which uses the simple while practical technologies such as Spring Boot, Java, Thymeleaf, Hibernate, Html and so on. By integrating all these skills together, I mastered practical, job-ready programming skills for a junior position.

Reference

Apache, (2015). “Super CSV” [online] Available at: http://super-csv.github.io/super-csv/index.html (Accessed 13 September 2022)

Bell, D. (2004). “Explore the UML sequence diagram” [Online] *IBM Developer* Available at: https://developer.ibm.com/articles/the-sequence-diagram/ (Accessed 12 September 2022)

Çevik, et al. (2022). “OpenPDF version 1.3.29 released 2022-07-09” [online] Available at: https://github.com/LibrePDF/OpenPDF (Accessed 13 September 2022)

EDUCBA., (2022). “Introduction to Spring Boot Executable Jar.” [online] Available at: https://www.educba.com/spring-boot-executable-jar (Accessed 03 September 2022).

Guthrie, G. (2022). “Create a use case scenario: how to think like users to improve products” [Online] *Backlog.com* Available at: https://backlog.com/blog/how-to-create-a-use-case-scenario-to-improve-products/ (Accessed 11 September 2022)

Hayford, E. (2021). “Understanding The bcrypt Hashing Function And Its Role in Rails.” [online] M*anny.codes* Available at:https://manny.codes/understanding-the-bcrypt-hashing-function-and-its-role-in-rails/ (Accessed 13 September 2022).

IBM Cloud Education, (2020). “Java Spring Boot-Learn how Java Spring Boot simplifies development of web applications and microservices with Java Spring Framework.” [Online] Available at: https://www.ibm.com/cloud/learn/java-spring-boot (Accessed 11 September 2022)

IETF OAuth Working Group, (2013). “OAuth 2.0” [online] Available at: https://oauth.net/2/ (Accessed 13 September 2022)

Intersys, (2021). “This is how to write a foolproof user requirements specification” [Online] Opinion and Expert Views. Available at:https://intersys.co.uk/2021/03/24/this-is-how-to-write-a-foolproof-user-requirements-specification/ (Accessed 11 September 2022)

Johnson, et al., (2016). “Spring Framework Reference Documentation”. [online] Available at: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/index.html (Accessed 11 September 2022).

Lane, C. and Krüger, N. (2021). “How to Write a Software Requirements Specification (SRS Document)” [Online] *Perforce.com* Available at: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document (Accessed 11 September 2022)

Lteif, G. (2022) “Solution Design and Its Role in Successful Projects” [Online] *Softwaredominos.com* Available at: https://softwaredominos.com/home/software-design-development-articles/solution-design-and-its-role-in-successful-projects/ (Accessed 11 September 2022)

Nam Ha Minh, (2020). “Spring Boot Email Verification for User Registration Tutorial” [online] *Code Java.* Available at: https://www.codejava.net/frameworks/spring-boot/email-verification-example (Accessed 13 September 2022)

Oracle, (2022). “What is Java technology and why do I need it?” [Online] Available at:

https://www.java.com/en/download/help/whatis_java.html (Accessed 11 September 2022)

Raman, R. (2021). "The Ultimate Guide to BCrypt and Authentication Protocols" [online] *clerk.dev* Available at: https://clerk.dev/blog/bcrypt-hashing-authentication-encryption
(Accessed 13 September 2022).

Rodriguez, N. (2015). "Introduction to RESTful Web services" [online] *IBM Web Developer.* Available at: https://developer.ibm.com/articles/ws-restful/ (Accessed 13 September 2022)

Scaler Academy, (2022). "Difference Between Spring MVC and Spring Boot" [online] *InterviewBit.com* Available at: https://www.interviewbit.com/blog/difference-between-spring-mvc-and-spring-boot/ (Accessed 11 September 2022).

Sparx Systems Pty Ltd, (2022). "UML 2 Tutorial - Activity Diagram" [Online] Available at: https://sparxsystems.com/resources/tutorials/uml2/activity-diagram.html (Accessed 12 September 2022)

Sparx Systems Pty Ltd, (2022). "UML 2 Tutorial - Class Diagram" [Online] Available at: https://sparxsystems.com/resources/tutorials/uml2/class-diagram.html (Accessed 12 September 2022)

The Apache Software Foundation, (2022). "Apache POI - the Java API for Microsoft Documents" [online] Available at: https://poi.apache.org/index.html
(Accessed 13 September 2022)

The Thymeleaf Team, (2022). Thymeleaf Document. [Online] Available at: https://www.thymeleaf.org/ (Accessed 11 September 2022)

Tutorialspoint, (2022). "Software Requirements" [Online] Tutorialspoint.com Available at: https://www.tutorialspoint.com/software_engineering/software_requirements.htm#
(Accessed 12 September 2022)

University of Regina, (2017). "Data Modeling and Entity Relationship Diagram" [Online] Available at: https://www.cs.uregina.ca/Links/class-info/215/erd/ (Accessed 12 September 2022)

W3C, (2022). "HTML & CSS" [Online] Available at: https://www.w3.org/standards/webdesign/htmlcss (Accessed 11 September 2022)

W3Schools, (2022). "AJAX Introduction" [online] Available at: https://www.w3schools.com/js/js_ajax_intro.asp (Accessed 13 September 2022)

Wikipedia, (2022) "Java (programming language)" [Online] Available at: https://en.wikipedia.org/wiki/Java_(programming_language) (Accessed 11 September 2022)

Wikipedia, (2022). "Bcrypt". [online] Available at: https://en.wikipedia.org/wiki/Bcrypt (Accessed 13 September 2022).

Wilkins, J. (2021). "MVC Architecture – What is a Model View Controller Framework?" [online] *FreeCodeCamp.org.* Available at: https://www.freecodecamp.org/news/mvc-architecture-what-is-a-model-view-controller-framework/ (Accessed 06 September 2022).

This Project is mainly adopted from following Yutube Video:

1. Author: The Sheryians Coding School

"E-Commerce Project in Spring Boot & Thymeleaf Playlist"
[online] Available at: https://youtu.be/z-9lNgN2QV4

2. Author: Code Java.
"Spring Boot Tutorials Playlist" [online] Available at: https://youtu.be/zDc63OHY_v8
"Spring Boot Thymeleaf Playlist" [online] Available at: https://youtu.be/-IMvhBWwQrc

3. Author: AmigosCode.
"Spring & Spring Boot Playlist" [online] Available at: https://youtu.be/8SGI_XS5OPw

4. Author: Learn Programming with Real Apps
"Completed E-Commerce with Spring MVC and Spring Data JPA Playlist" [online] Available at: https://youtu.be/hwSB50DqMkc

5. Author: Self-code
"Ecommerce Spring Boot Project step-by-step using Spring Boot, Thymeleaf, Spring Security, Hibernate, Spring Data Jpa, Lombok." [online] Available at: https://youtu.be/4ga5bcBPgzo

6. Author: Jay Gajera
"Spring E commerce project | With Code| Run step by step execution" [online] Available at: https://youtu.be/c6WWdINWSlI

Appendix

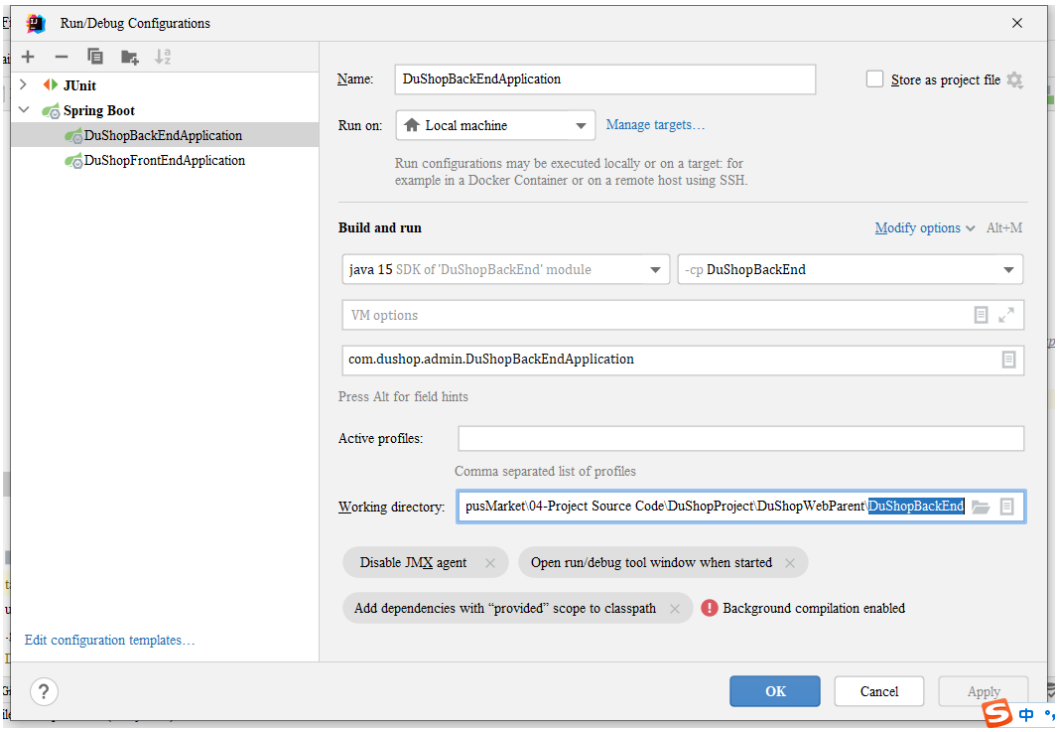
A: Source Code

The project code can be found at:
https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2021/cxj164.git

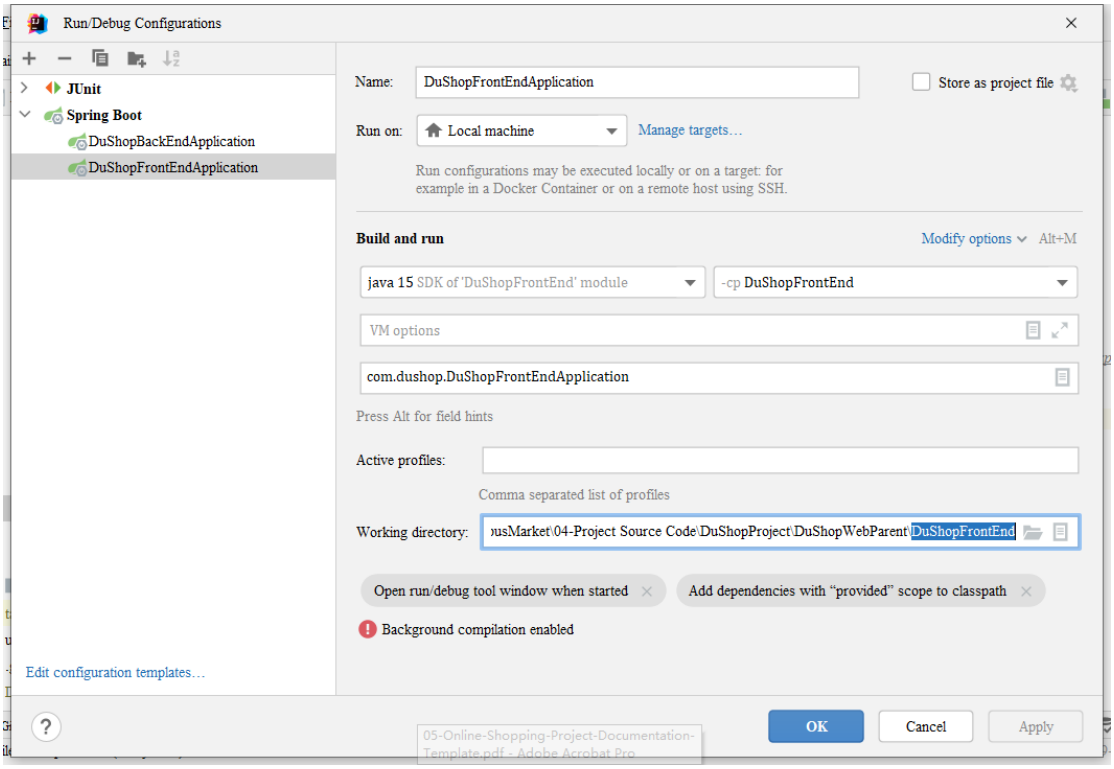
MYSQL:
- Host: localhost
- Port: 3306
- User: root
- Password: root
- Database: dushopdb

B: How To Run

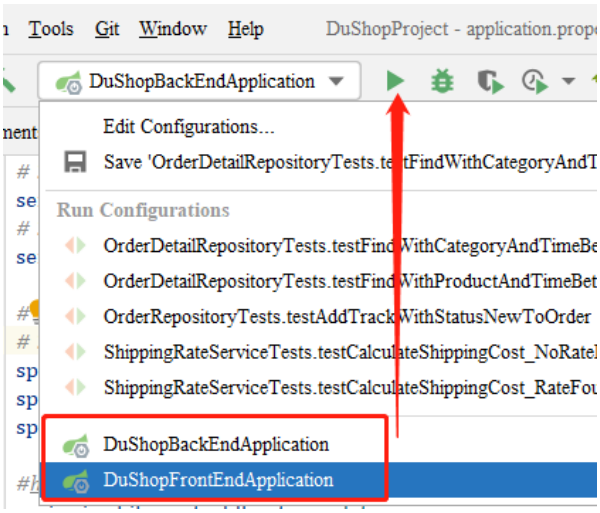
1. Install InteliJ Idea from https://www.jetbrains.com/idea/ .
2. Install JDK15 from https://jdk.java.net/15/ .
3. Pull form gitlab repository and open the "*04-Project Source Code*" folder. Souce code is stored here. Then open the source code using InteliJ Idea.
4. Wait for the InteliJ Idea to download the libraries used in this project.
5. Set the working directory of Admin Application to the "DuShopBackEnd" directory.



6. Set the working directory of Shopping Application to the "DuShopFrontEnd" directory.



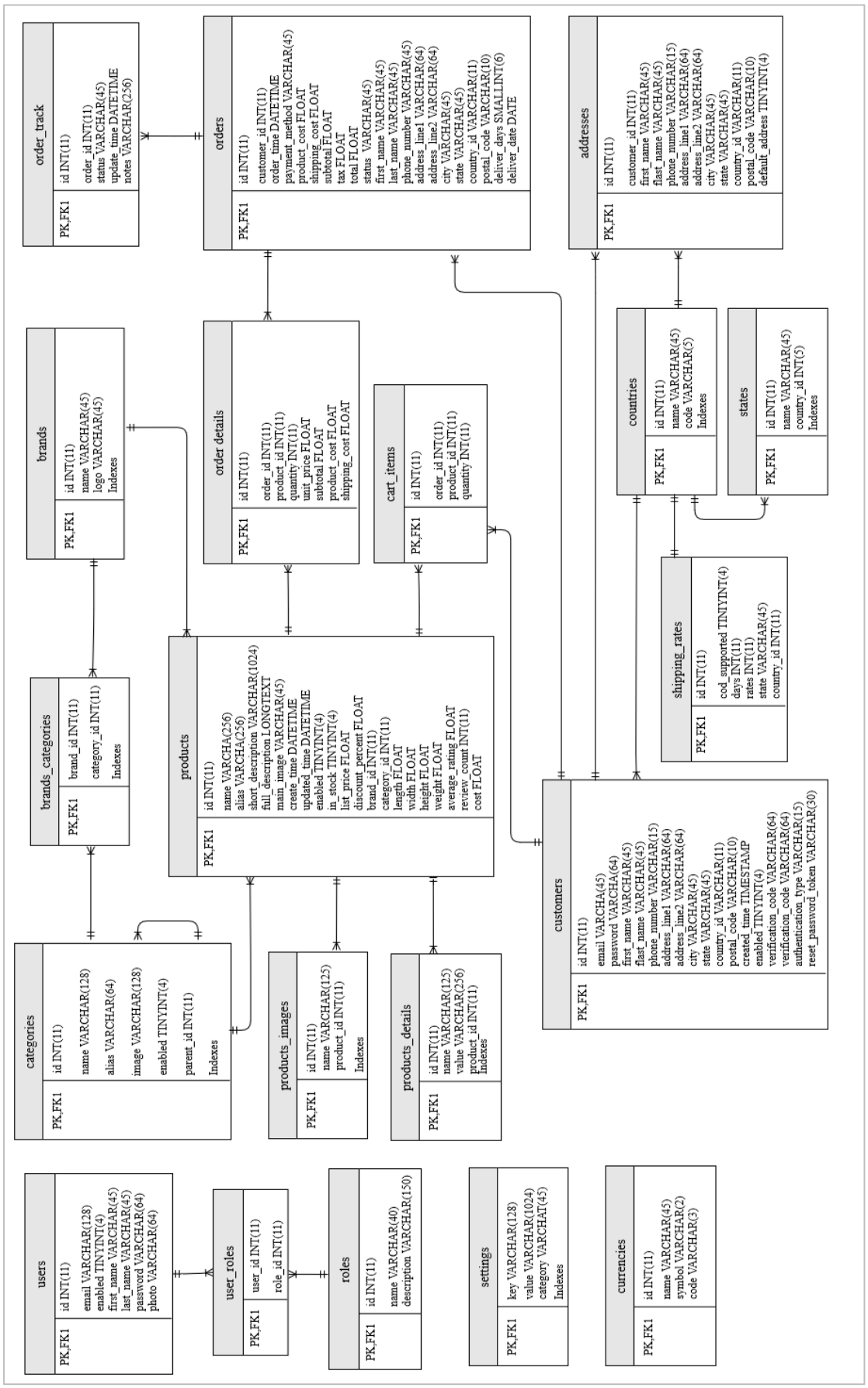
7. Select one of the Applications and Clicks "Run" Button. Repeat select, and run another Application



8. Open browser and input http://localhost:8080/DuShopAdmin/ , Admin Management System is reached.
9. Open browser and input http://localhost/DuShop, the Shopping website is reached.

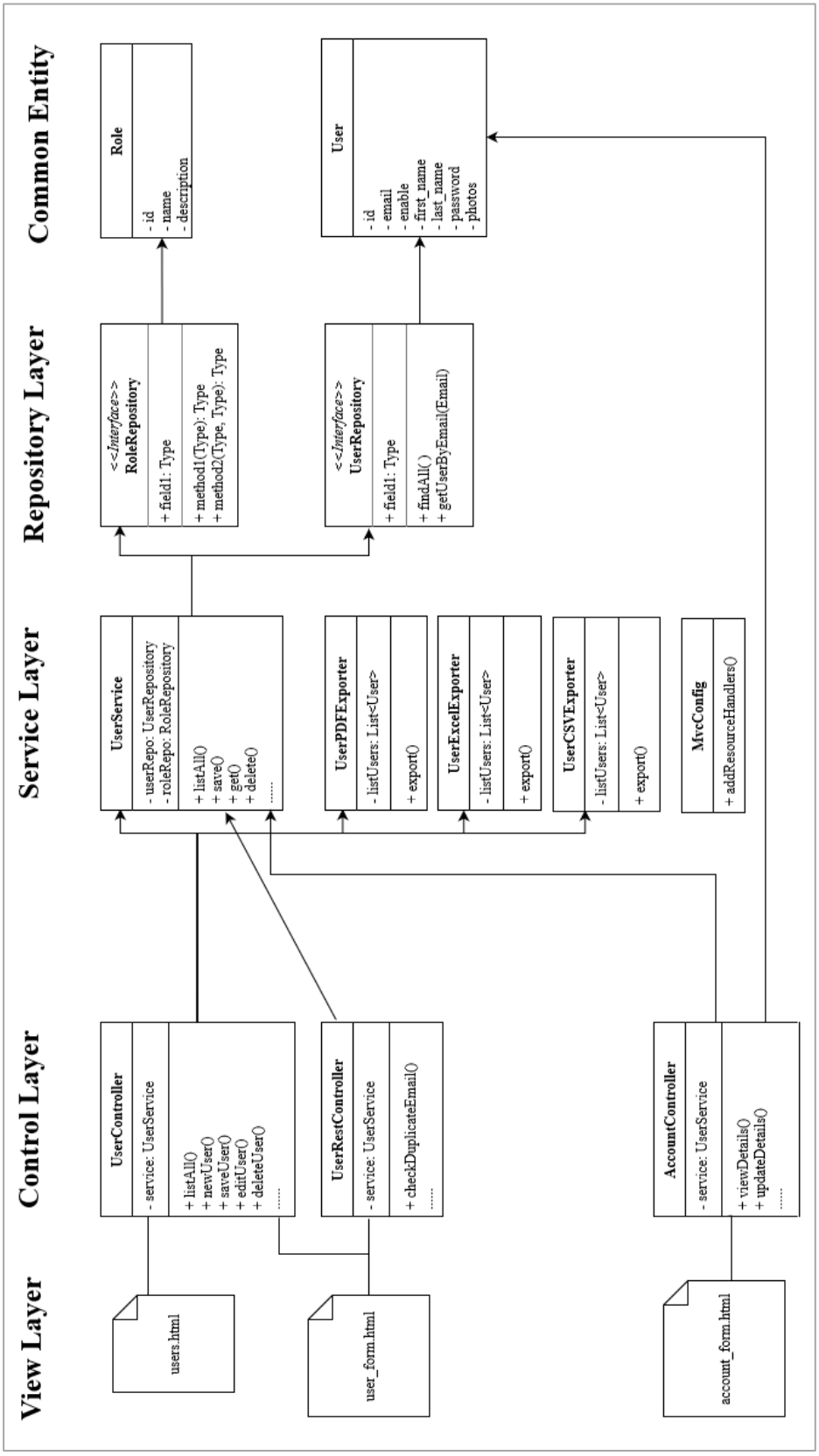
C: Entity Relationship Diagram

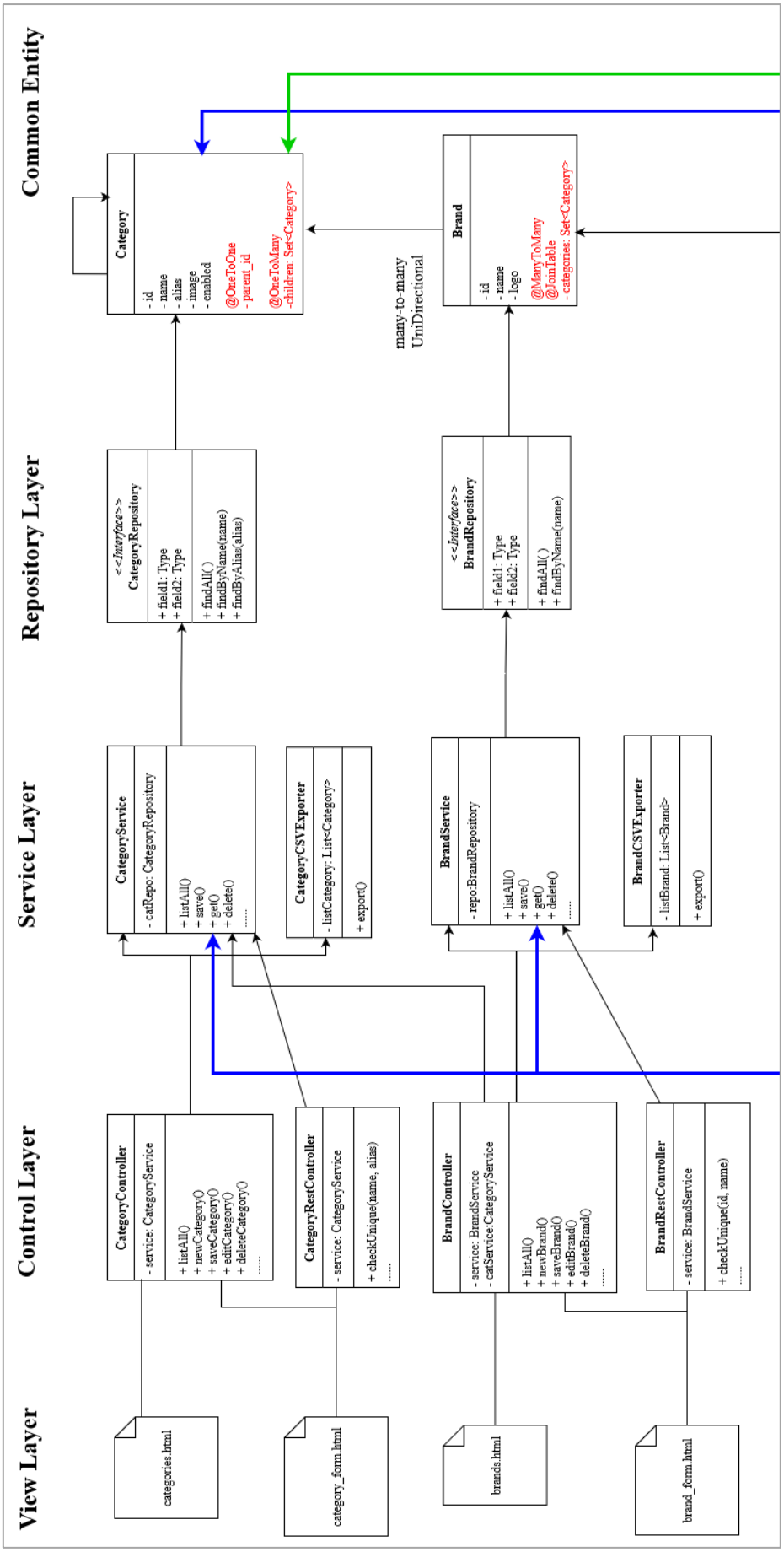
Figure 3.4.1

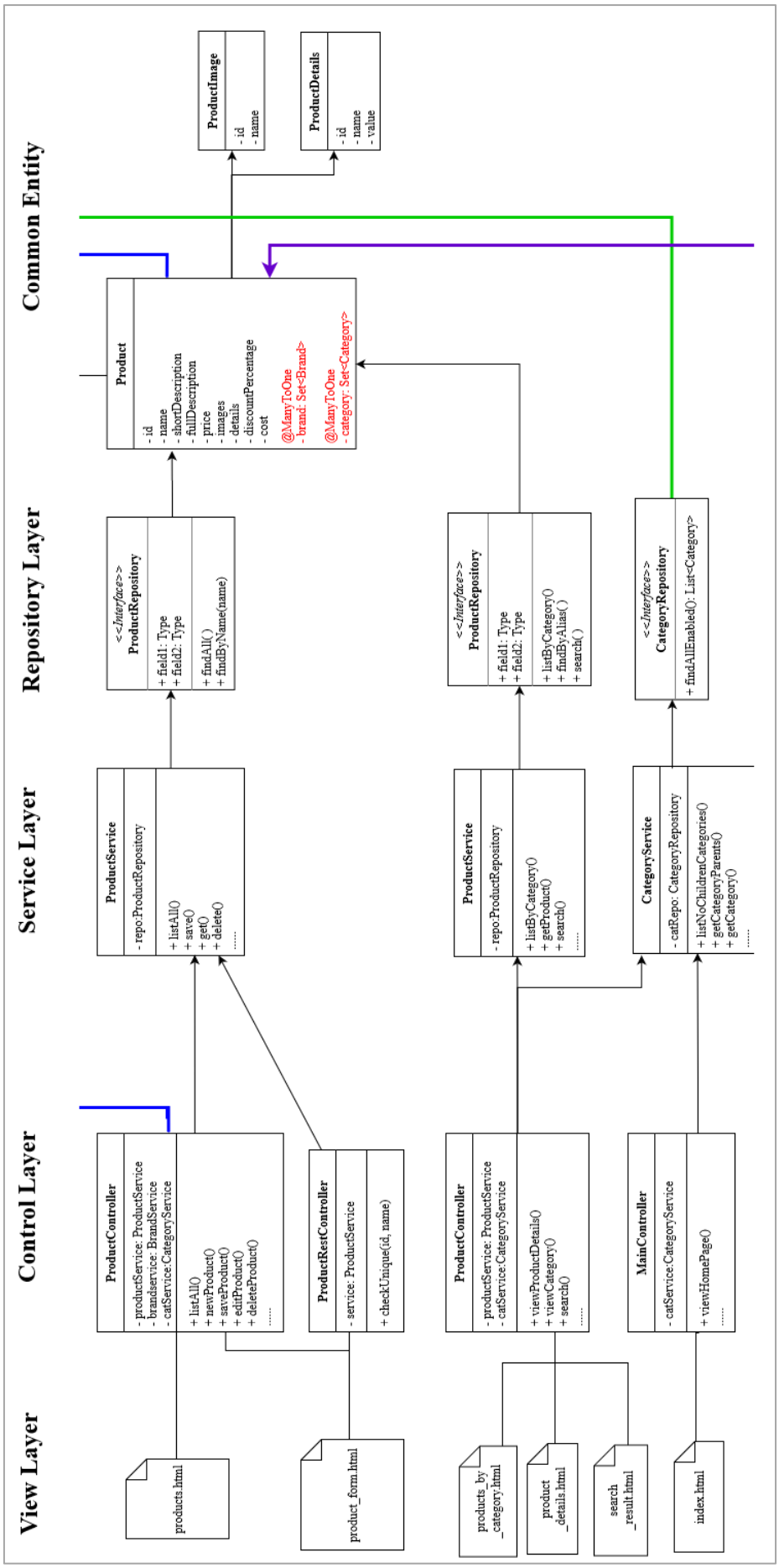

order_track
PK,FK1
id INT(11)
order_id INT(11)
status VARCHAR(45)
update_time DATETIME
notes VARCHAR(256)
orders
PK,FK1
id INT(11)
customer_id INT(11)
order_time DATETIME
payment_method VARCHAR(45)
product_cost FLOAT
shipping_cost FLOAT
subtotal FLOAT
tax FLOAT
total FLOAT
status VARCHAR(45)
first_name VARCHAR(45)
last_name VARCHAR(45)
phone_number VARCHAR(45)
address_line1 VARCHAR(64)
address_line2 VARCHAR(64)
city VARCHAR(45)
state VARCHAR(45)
country_id VARCHAR(11)
postal_code VARCHAR(10)
deliver_days SMALLINT(6)
deliver_date DATE
addresses
PK,FK1
id INT(11)
customer_id INT(11)
first_name VARCHAR(45)
last_name VARCHAR(45)
phone_number VARCHAR(15)
address_line1 VARCHAR(64)
address_line2 VARCHAR(64)
city VARCHAR(45)
state VARCHAR(45)
country_id VARCHAR(11)
postal_code VARCHAR(10)
default_address TINYINT(4)
brands
PK,FK1
id INT(11)
name VARCHAR(45)
logo VARCHAR(45)
Indexes
order details
PK,FK1
id INT(11)
order_id INT(11)
product_id INT(11)
quantity INT(11)
unit_price FLOAT
subtotal FLOAT
product_cost FLOAT
shipping_cost FLOAT
cart_items
PK,FK1
id INT(11)
order_id INT(11)
product_id INT(11)
quantity INT(11)
countries
PK,FK1
id INT(11)
name VARCHAR(45)
code VARCHAR(5)
Indexes
states
PK,FK1
id INT(11)
name VARCHAR(45)
country_id INT(5)
Indexes
brands_categories
PK,FK1
brand_id INT(11)
category_id INT(11)
Indexes
products
PK,FK1
id INT(11)
name VARCHA(256)
alias VARCHA(256)
short_description VARCHAR(1024)
full_description LONGTEXT
main_image VARCHAR(45)
create_time DATETIME
updated_time DATETIME
enabled TINYINT(4)
in_stock TINYINT(4)
list_price FLOAT
discount_percent FLOAT
brand_id INT(11)
category_id INT(11)
length FLOAT
width FLOAT
height FLOAT
weight FLOAT
average_rating FLOAT
review_count INT(11)
cost FLOAT
shipping_rates
PK,FK1
id INT(11)
cod_supported TINYINT(4)
days INT(11)
rates INT(11)
state VARCHAR(45)
country_id INT(11)
customers
PK,FK1
id INT(11)
email VARCHA(45)
password VARCHA(64)
first_name VARCHAR(45)
last_name VARCHAR(45)
phone_number VARCHAR(15)
address_line1 VARCHAR(64)
address_line2 VARCHAR(64)
city VARCHAR(45)
state VARCHAR(45)
country_id VARCHAR(11)
postal_code VARCHAR(10)
created_time TIMESTAMP
enabled TINYINT(4)
verification_code VARCHAR(64)
verification_code VARCHAR(64)
authentication_type VARCHAR(15)
reset_password_token VARCHAR(30)
categories
PK,FK1
id INT(11)
name VARCHAR(128)
alias VARCHAR(64)
image VARCHAR(128)
enabled TINYINT(4)
parent_id INT(11)
Indexes
products_images
PK,FK1
id INT(11)
name VARCHAR(125)
product_id INT(11)
Indexes
products_details
PK,FK1
id INT(11)
name VARCHAR(125)
value VARCHAR(256)
product_id INT(11)
Indexes
users
PK,FK1
id INT(11)
email VARCHAR(128)
enabled TINYINT(4)
first_name VARCHAR(45)
last_name VARCHAR(45)
password VARCHAR(64)
photo VARCHAR(64)
user_roles
PK,FK1
user_id INT(11)
role_id INT(11)
roles
PK,FK1
id INT(11)
name VARCHAR(40)
description VARCHAR(150)
settings
PK,FK1
key VARCHAR(128)
value VARCHAR(1024)
category VARCHAT(45)
Indexes
currencies
PK,FK1
id INT(11)
name VARCHAR(45)
symbol VARCHAR(2)
code VARCHAR(3)

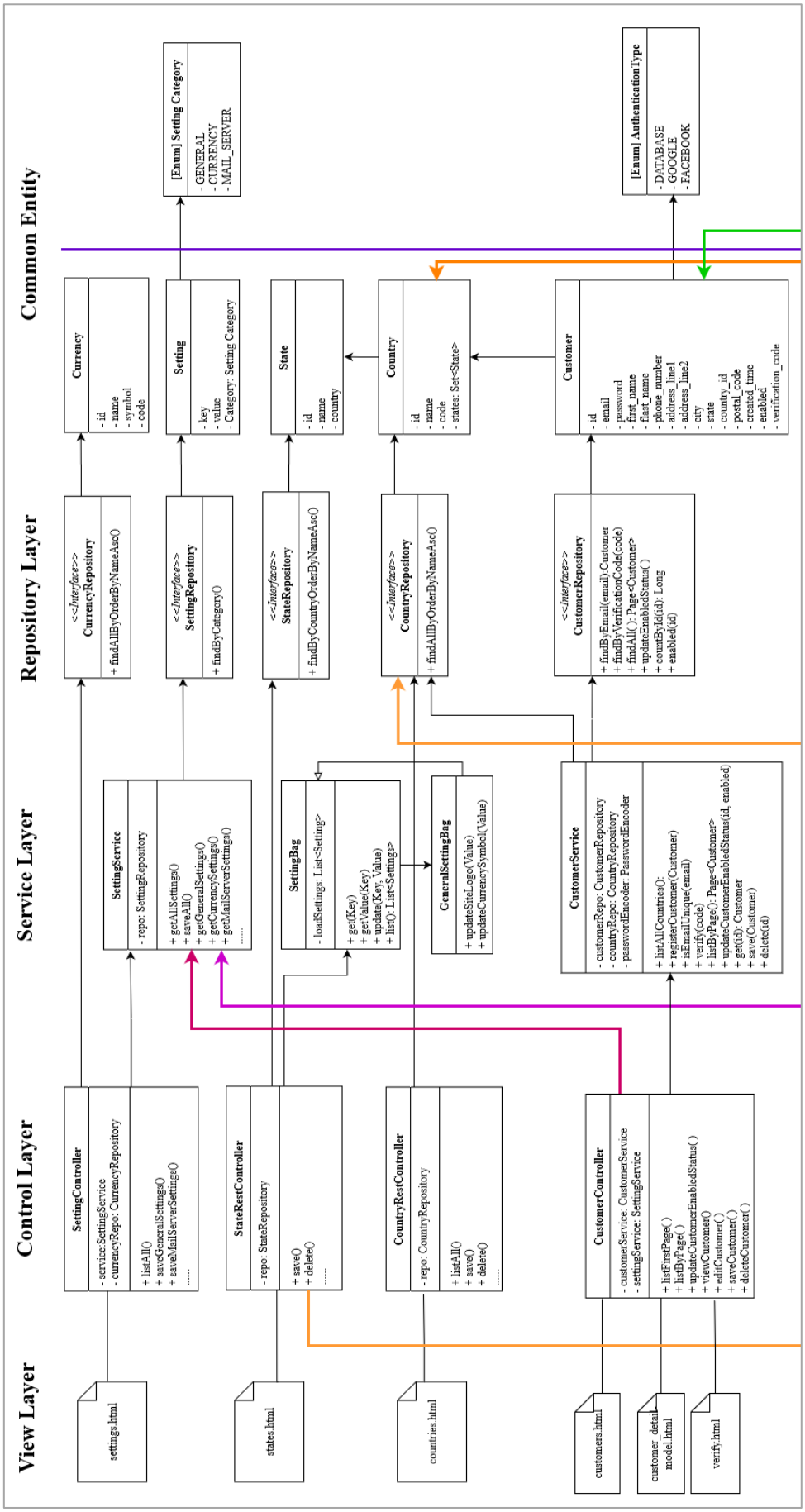
D: Class Diagram

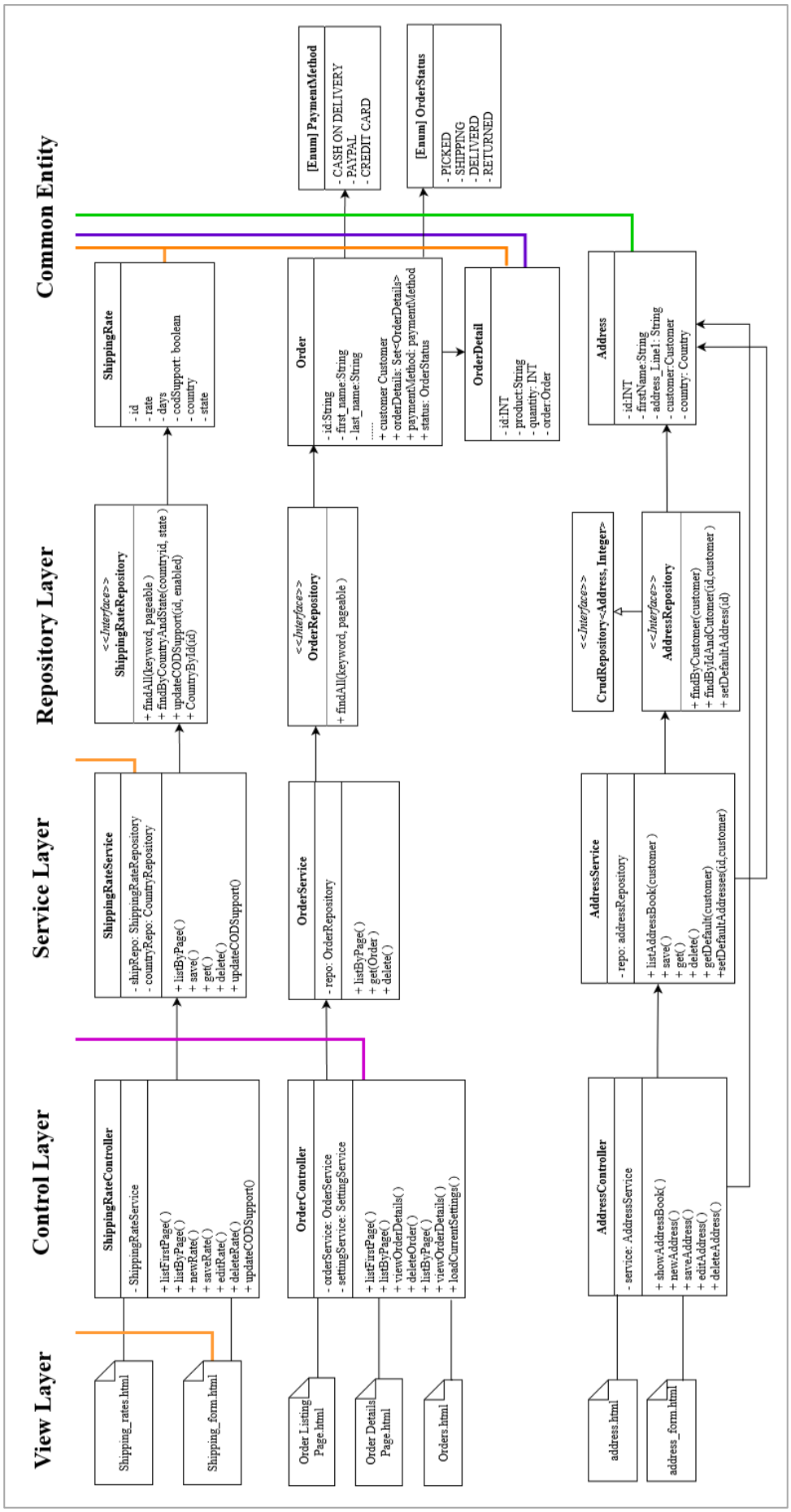
D1 Figure 3.5.1: Admin Application



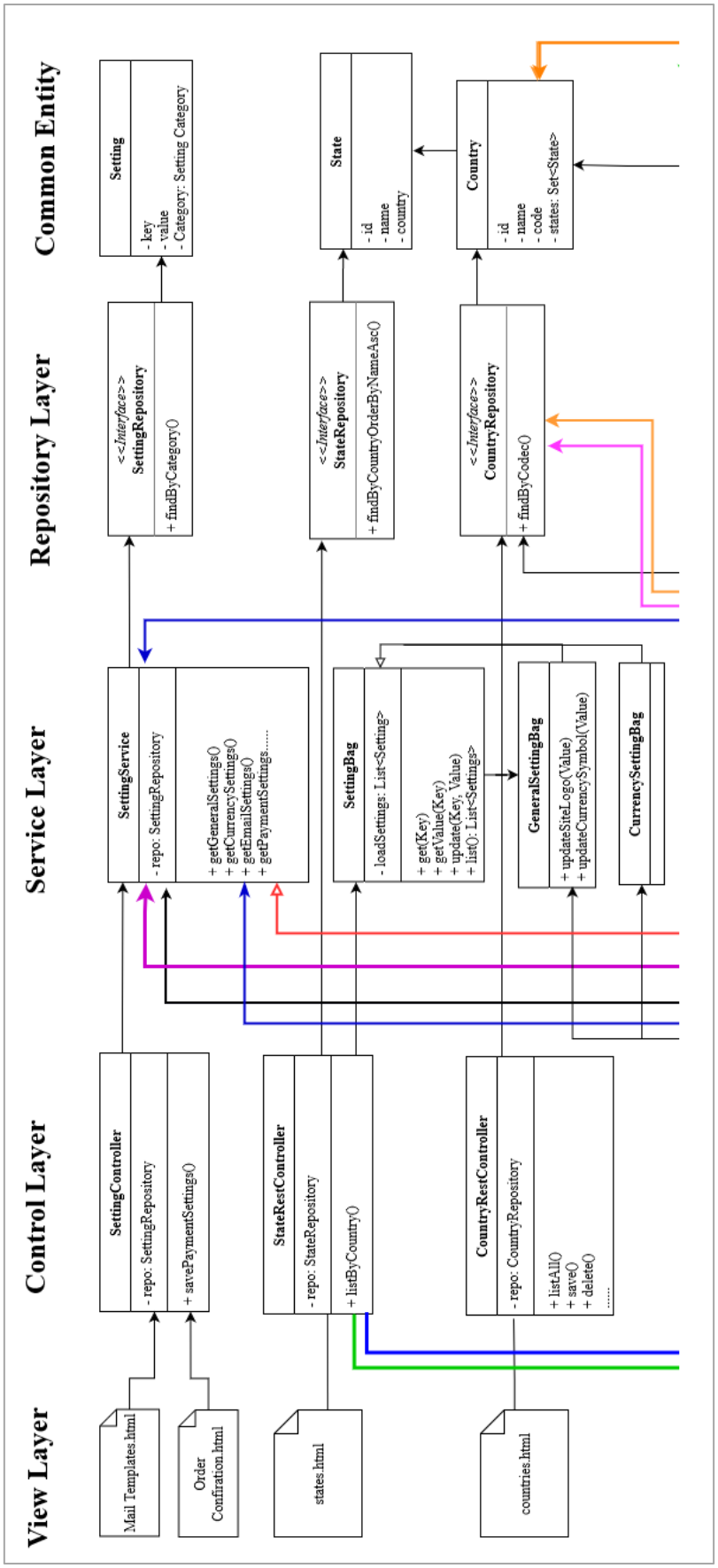

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
Category
- id
- name
- alias
- image
- enabled
@OneToOne
- parent_id
@OneToMany
-children: Set<Category>
Brand
- id
- name
- logo
@ManyToMany
@JoinTable
- categories: Set<Category>
many-to-many
UniDirectional
<<Interface>>
CategoryRepository
+ field1: Type
+ field2: Type
+ findAll()
+ findByName(name)
+ findByAlias(alias)
<<Interface>>
BrandRepository
+ field1: Type
+ field2: Type
+ findAll()
+ findByName(name)
CategoryService
- catRepo: CategoryRepository
+ listAll()
+ save()
+ get()
+ delete()
.....
CategoryCSVExporter
- listCategory: List<Category>
+ export()
BrandService
- repo:BrandRepository
+ listAll()
+ save()
+ get()
+ delete()
.....
BrandCSVExporter
- listBrand: List<Brand>
+ export()
CategoryController
- service: CategoryService
+ listAll()
+ newCategory()
+ saveCategory()
+ editCategory()
+ deleteCategory()
.....
CategoryRestController
- service: CategoryService
+ checkUnique(name, alias)
.....
BrandController
- service: BrandService
- catService:CategoryService
+ listAll()
+ newBrand()
+ saveBrand()
+ editBrand()
+ deleteBrand()
.....
BrandRestController
- service: BrandService
+ checkUnique(id, name)
.....
categories.html
category_form.html
brands.html
brand_form.html

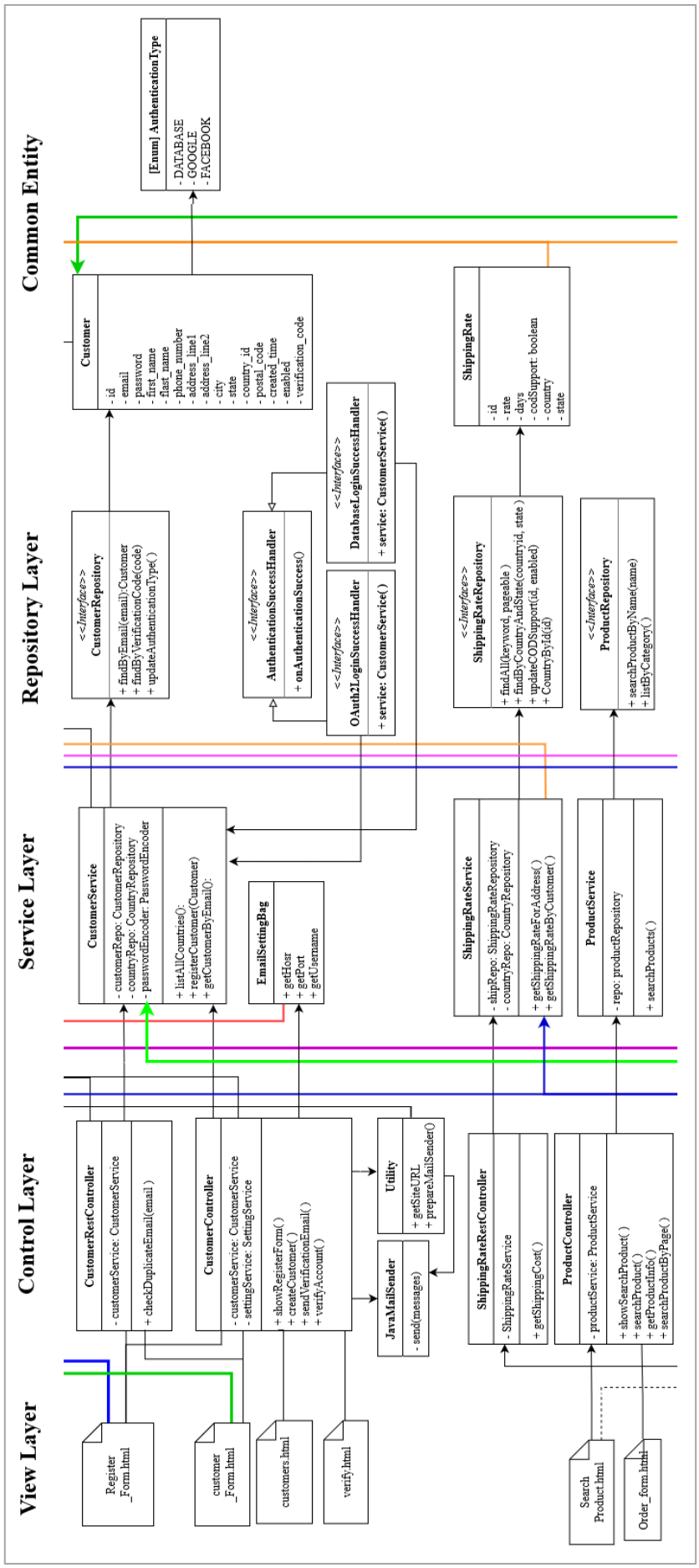

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
products.html
product_form.html
products_by_category.html
product_details.html
search_result.html
index.html
ProductController
- productService: ProductService
- brandService: BrandService
- catService:CategoryService
+ listAll()
+ newProduct()
+ saveProduct()
+ editProduct()
+ deleteProduct()
.....
ProductRestController
- service: ProductService
+ checkUnique(id, name)
ProductController
- productService: ProductService
- catService:CategoryService
+ viewProductDetails()
+ viewCategory()
+ search()
.....
MainController
- catService:CategoryService
+ viewHomePage()
.....
ProductService
- repo:ProductRepository
+ listAll()
+ save()
+ get()
+ delete()
.....
ProductService
- repo:ProductRepository
+ listByCategory()
+ getProduct()
+ search()
.....
CategoryService
- catRepo: CategoryRepository
+ listNoChildrenCategories()
+ getCategoryParents()
+ getCategory()
.....
<<Interface>>
ProductRepository
+ field1: Type
+ field2: Type
+ findAll()
+ findByName(name)
<<Interface>>
ProductRepository
+ field1: Type
+ field2: Type
+ listByCategory()
+ findByAlias()
+ search()
<<Interface>>
CategoryRepository
+ findAllEnabled(): List<Category>
Product
- id
- name
- shortDescription
- fullDescription
- price
- images
- details
- discountPercentage
- cost
@ManyToOne
- brand: Set<Brand>
@ManyToOne
- category: Set<Category>
ProductImage
- id
- name
ProductDetails
- id
- name
- value

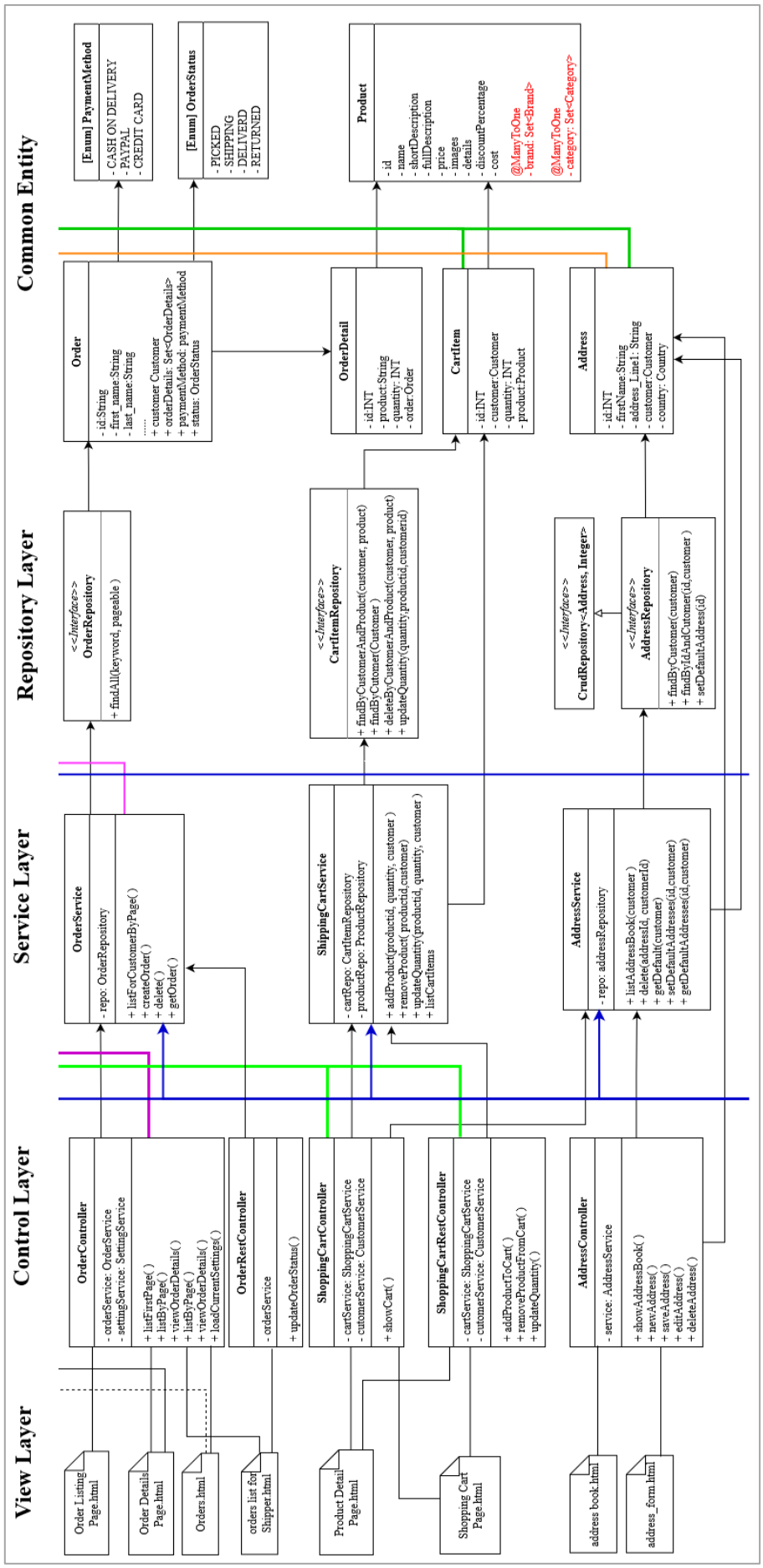

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
SettingController
- service:SettingService
- currencyRepo: CurrencyRepository
+ listAll()
+ saveGeneralSettings()
+ saveMailServerSettings()
.....
StateRestController
- repo: StateRepository
+ save()
+ delete()
.....
CountryRestController
- repo: CountryRepository
+ listAll()
+ save()
+ delete()
.....
CustomerController
- customerService: CustomerService
- settingService: SettingService
+ listFirstPage( )
+ listByPage( )
+ updateCustomerEnabledStatus( )
+ viewCustomer()
+ editCustomer( )
+ saveCustomer( )
+ deleteCustomer( )
SettingService
- repo: SettingRepository
+ getAllSettings()
+ saveAll()
+ getGeneralSettings()
+ getCurrencySettings()
+ getMailServerSettings()
.....
SettingBag
- loadSettings: List<Setting>
+ get(Key)
+ getValue(Key)
+ update(Key, Value)
+ list(): List<Setting>
GeneralSettingBag
+ updateSiteLogo(Value)
+ updateCurrencySymbol(Value)
CustomerService
- customerRepo: CustomerRepository
- countryRepo: CountryRepository
- passwordEncoder: PasswordEncoder
+ listAllCountries();
+ registerCustomer(Customer)
+ isEmailUnique(email)
+ verify(code)
+ listByPage(): Page<Customer>
+ updateCustomerEnabledStatus(id, enabled)
+ get(id): Customer
+ save(Customer)
+ delete(id)
<<Interface>>
CurrencyRepository
+ findAllByOrderByNameAsc()
<<Interface>>
SettingRepository
+ findByCategory()
<<Interface>>
StateRepository
+ findByCountryOrderByNameAsc()
<<Interface>>
CountryRepository
+ findAllByOrderByNameAsc()
<<Interface>>
CustomerRepository
+ findByEmail(email):Customer
+ findByVerificationCode(code)
+ findAll( ): Page<Customer>
+ updateEnabledStatus( )
+ countById(id): Long
+ enabled(id)
Currency
- id
- name
- symbol
- code
Setting
- key
- value
- Category: Setting Category
State
- id
- name
- country
Country
- id
- name
- code
- states: Set<State>
Customer
- id
- email
- password
- first_name
- last_name
- phone_number
- address_line1
- address_line2
- city
- state
- country_id
- postal_code
- created_time
- enabled
- verification_code
[Enum] Setting Category
- GENERAL
- CURRENCY
- MAIL_SERVER
[Enum] AuthenticationType
- DATABASE
- GOOGLE
- FACEBOOK
settings.html
states.html
countries.html
customers.html
customer_detail_model.html
verify.html

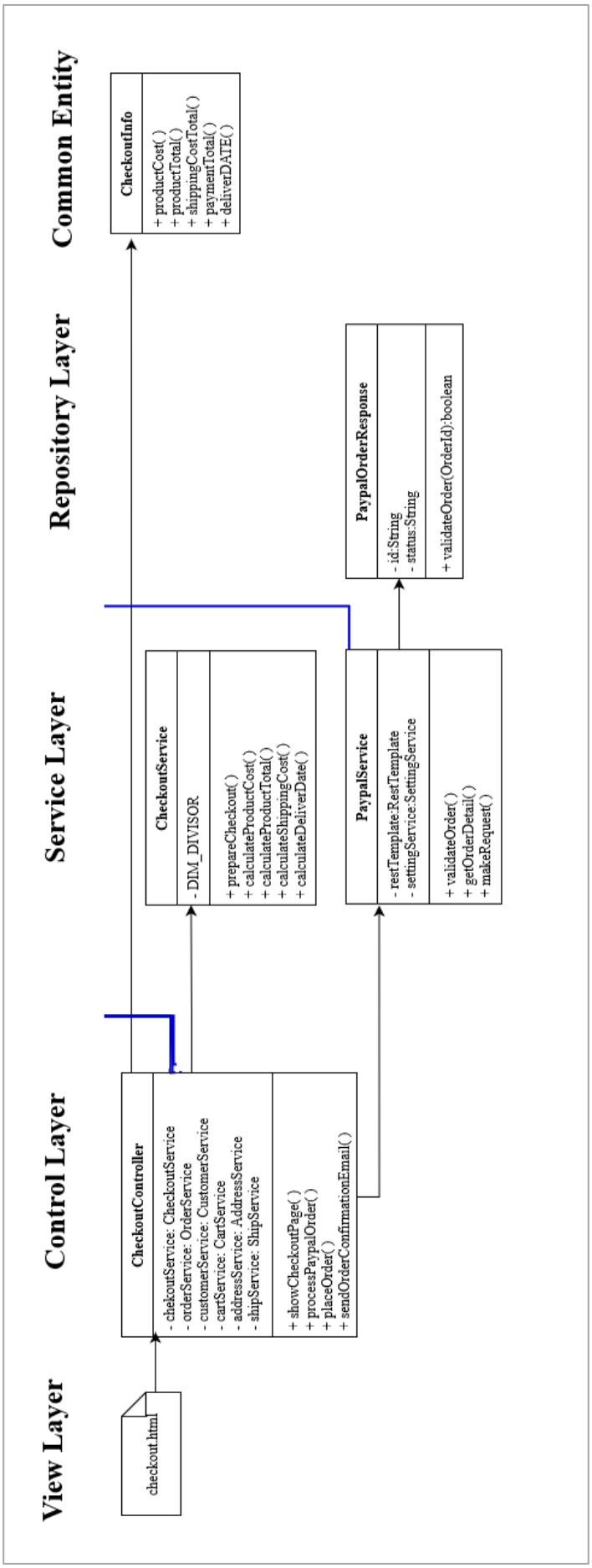

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
Shipping_rates.html
Shipping_form.html
Order Listing Page.html
Order Details Page.html
Orders.html
address.html
address_form.html
ShippingRateController
- ShippingRateService
+ listFirstPage( )
+ listByPage( )
+ newRate( )
+ saveRate( )
+ editRate( )
+ deleteRate( )
+ updateCODSupport()
OrderController
- orderService: OrderService
- settingService: SettingService
+ listFirstPage( )
+ listByPage( )
+ viewOrderDetails( )
+ deleteOrder( )
+ listByPage( )
+ viewOrderDetails( )
+ loadCurrentSettings( )
AddressController
- service: AddressService
+ showAddressBook( )
+ newAddress( )
+ saveAddress( )
+ editAddress( )
+ deleteAddress( )
ShippingRateService
- shipRepo: ShippingRateRepository
- countryRepo: CountryRepository
+ listByPage( )
+ save( )
+ get( )
+ delete( )
+ updateCODSupport()
OrderService
- repo: OrderRepository
+ listByPage( )
+ get(Order)
+ delete( )
AddressService
- repo: addressRepository
+ listAddressBook(customer)
+ save( )
+ get( )
+ delete( )
+ getDefault(customer)
+setDefaultAddresses(id,customer)
<<Interface>>
ShippingRateRepository
+ findAll(keyword, pageable)
+ findByCountryAndState(countryid, state)
+ updateCODSupport(id, enabled)
+ CountryById(id)
<<Interface>>
OrderRepository
+ findAll(keyword, pageable)
<<Interface>>
CrudRepository<Address, Integer>
<<Interface>>
AddressRepository
+ findByCustomer(customer)
+ findByIdAndCustomer(id,customer)
+ setDefaultAddress(id)
ShippingRate
- id
- rate
- days
- codSupport: boolean
- country
- state
Order
- id:String
- first_name:String
- last_name:String
.....
+ customer Customer
+ orderDetails: Set<OrderDetails>
+ paymentMethod: paymentMethod
+ status: OrderStatus
OrderDetail
- id:INT
- product:String
- quantity: INT
- order.Order
Address
- id:INT
- firstName:String
- address_Line1: String
- customer.Customer
- country: Country
[Enum] PaymentMethod
- CASH ON DELIVERY
- PAYPAL
- CREDIT CARD
[Enum] OrderStatus
- PICKED
- SHIPPING
- DELIVERD
- RETURNED

D2 Figure 3.5.2: Shopping Application



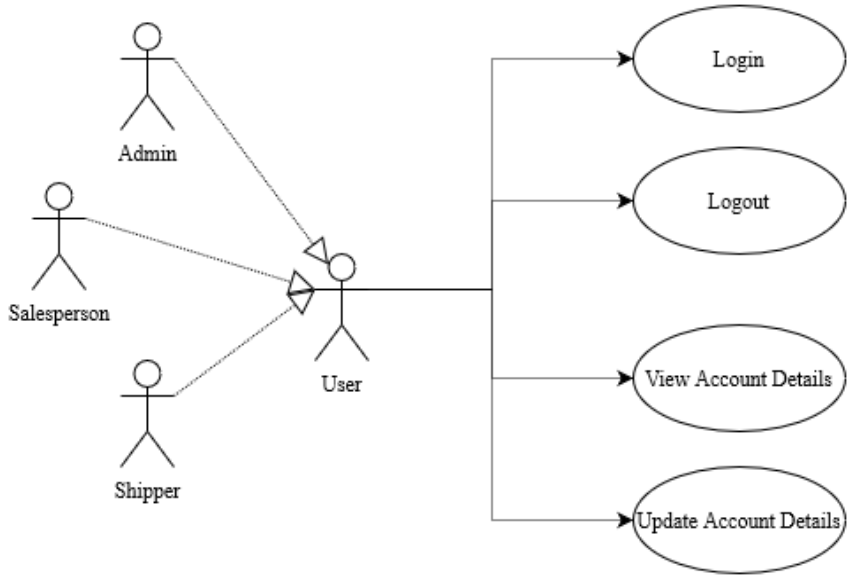

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
Register_Form.html
customer_Form.html
customers.html
verify.html
Search Product.html
Order_form.html
CustomerRestController
- customerService: CustomerService
+ checkDuplicateEmail(email)
CustomerController
- customerService: CustomerService
- settingService: SettingService
+ showRegisterForm( )
+ createCustomer( )
+ sendVerificationEmail( )
+ verifyAccount( )
Utility
+ getSiteURL
+ prepareMailSender()
JavaMailSender
- send(messages)
ShippingRateRestController
- ShippingRateService
+ getShippingCost( )
ProductController
- productService: ProductService
+ showSearchProduct( )
+ searchProduct( )
+ getProductInfo( )
+ searchProductByPage( )
CustomerService
- customerRepo: CustomerRepository
- countryRepo: CountryRepository
- passwordEncoder: PasswordEncoder
+ listAllCountries():
+ registerCustomer(Customer)
+ getCustomerByEmail():
EmailSettingBag
+ getHost
+ getPort
+ getUsername
ShippingRateService
- shipRepo: ShippingRateRepository
- countryRepo: CountryRepository
+ getShippingRateForAddress( )
+ getShippingRateByCustomer( )
ProductService
- repo: productRepository
+ searchProducts( )
<<Interface>>
CustomerRepository
+ findByEmail(email):Customer
+ findByVerificationCode(code)
+ updateAuthenticationType( )
<<Interface>>
OAuth2LoginSuccessHandler
+ service: CustomerService( )
<<Interface>>
AuthenticationSuccessHandler
+ onAuthenticationSuccess()
<<Interface>>
DatabaseLoginSuccessHandler
+ service: CustomerService( )
<<Interface>>
ShippingRateRepository
+ findAll(keyword, pageable)
+ findByCountryAndState(countryid, state)
+ updateCODSupport(id, enabled)
+ CountryById(id)
<<Interface>>
ProductRepository
+ searchProductByName(name)
+ listByCategory( )
Customer
- id
- email
- password
- first_name
- last_name
- phone_number
- address_line1
- address_line2
- city
- state
- country_id
- postal_code
- created_time
- enabled
- verification_code
[Enum] AuthenticationType
- DATABASE
- GOOGLE
- FACEBOOK
ShippingRate
- id
- rate
- days
- codSupport: boolean
- country
- state

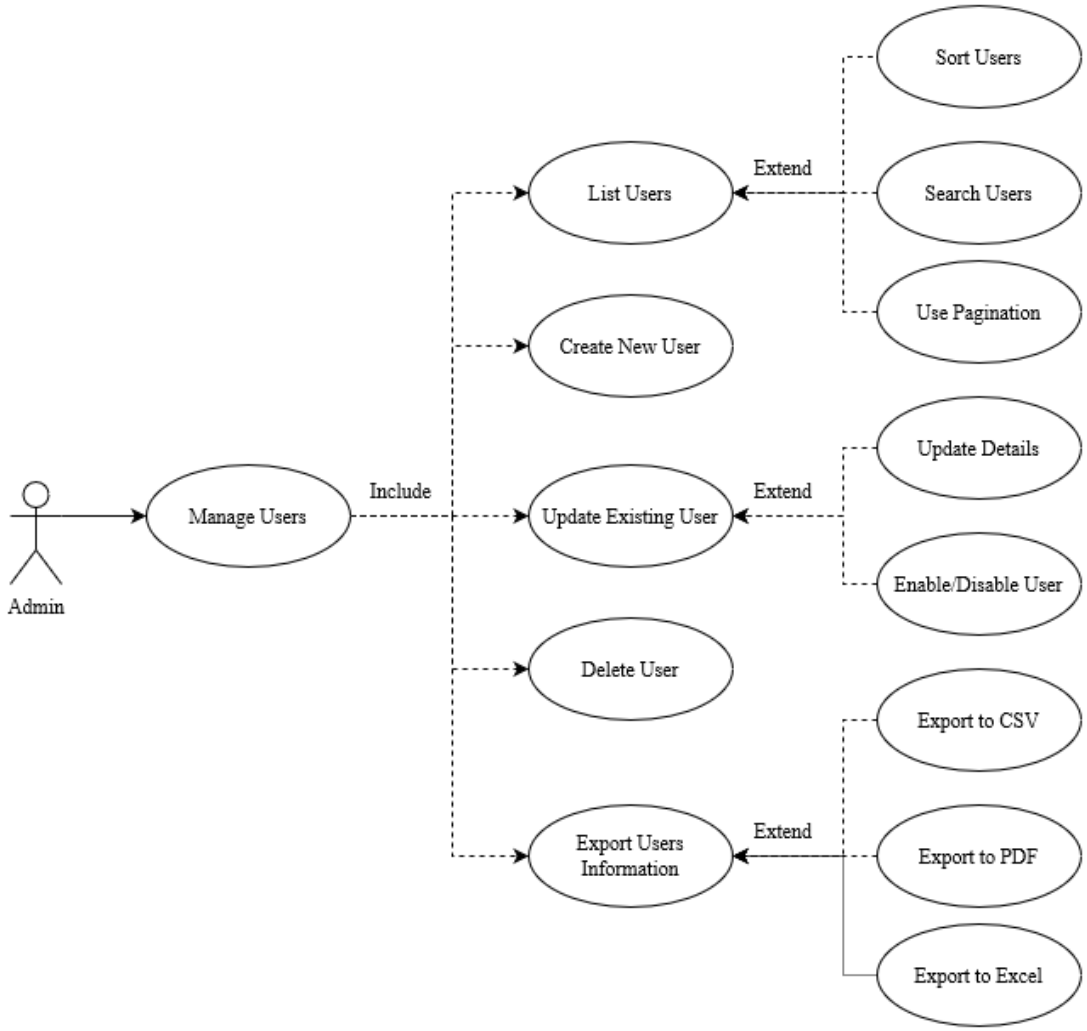

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
Order Listing Page.html
Order Details Page.html
Orders.html
orders list for Shipper.html
Product Detail Page.html
Shopping Cart Page.html
address book.html
address_form.html
OrderController
- orderService: OrderService
- settingService: SettingService
+ listFirstPage( )
+ listByPage( )
+ viewOrderDetails( )
+ listByPage( )
+ viewOrderDetails( )
+ loadCurrentSettings( )
OrderRestController
- orderService
+ updateOrderStatus( )
ShoppingCartController
- cartService: ShoppingCartService
- cutomerService: CustomerService
+ showCart( )
ShoppingCartRestController
- cartService: ShoppingCartService
- cutomerService: CustomerService
+ addProductToCart( )
+ removeProductFromCart( )
+ updateQuantity( )
AddressController
- service: AddressService
+ showAddressBook( )
+ newAddress( )
+ saveAddress( )
+ editAddress( )
+ deleteAddress( )
OrderService
- repo: OrderRepository
+ listForCustomerByPage( )
+ createOrder( )
+ delete( )
+ getOrder( )
ShippingCartService
- cartRepo: CartItemRepository
- productRepo: ProductRepository
+ addProduct(productid, quantity, customer)
+ removeProduct(productid,customer)
+ updateQuantity(productid, quantity, customer)
+ listCartItems
AddressService
- repo: addressRepository
+ listAddressBook(customer)
+ delete(addressid, customerid)
+ getDefault(customer)
+ setDefaultAddresses(id,customer)
+ getDefaultAddresses(id,customer)
<<Interface>>
OrderRepository
+ findAll(keyword, pageable)
<<Interface>>
CartItemRepository
+ findByCustomerAndProduct(customer, product)
+ findByCutomer(Customer)
+ deleteByCustomerAndProduct(customer, product)
+ updateQuantity(quantity,productid,customerid)
<<Interface>>
CrudRepository<Address, Integer>
<<Interface>>
AddressRepository
+ findByCustomer(customer)
+ findByIdAndCutomer(id,customer)
+ setDefaultAddress(id)
Order
- id:String
- first_name:String
- last_name:String
.....
+ customer Customer
+ orderDetails: Set<OrderDetails>
+ paymentMethod: paymentMethod
+ status: OrderStatus
OrderDetail
- id:INT
- product:String
- quantity: INT
- order:Order
CartItem
- id:INT
- customer:Customer
- quantity: INT
- product:Product
Address
- id:INT
- firstName:String
- address_Line1: String
- customer:Customer
- country: Country
[Enum] PaymentMethod
- CASH ON DELIVERY
- PAYPAL
- CREDIT CARD
[Enum] OrderStatus
- PICKED
- SHIPPING
- DELIVERD
- RETURNED
Product
- id
- name
- shortDescription
- fullDescription
- price
- images
- details
- discountPercentage
- cost
@ManyToOne
- brand: Set<Brand>
@ManyToOne
- category: Set<Category>

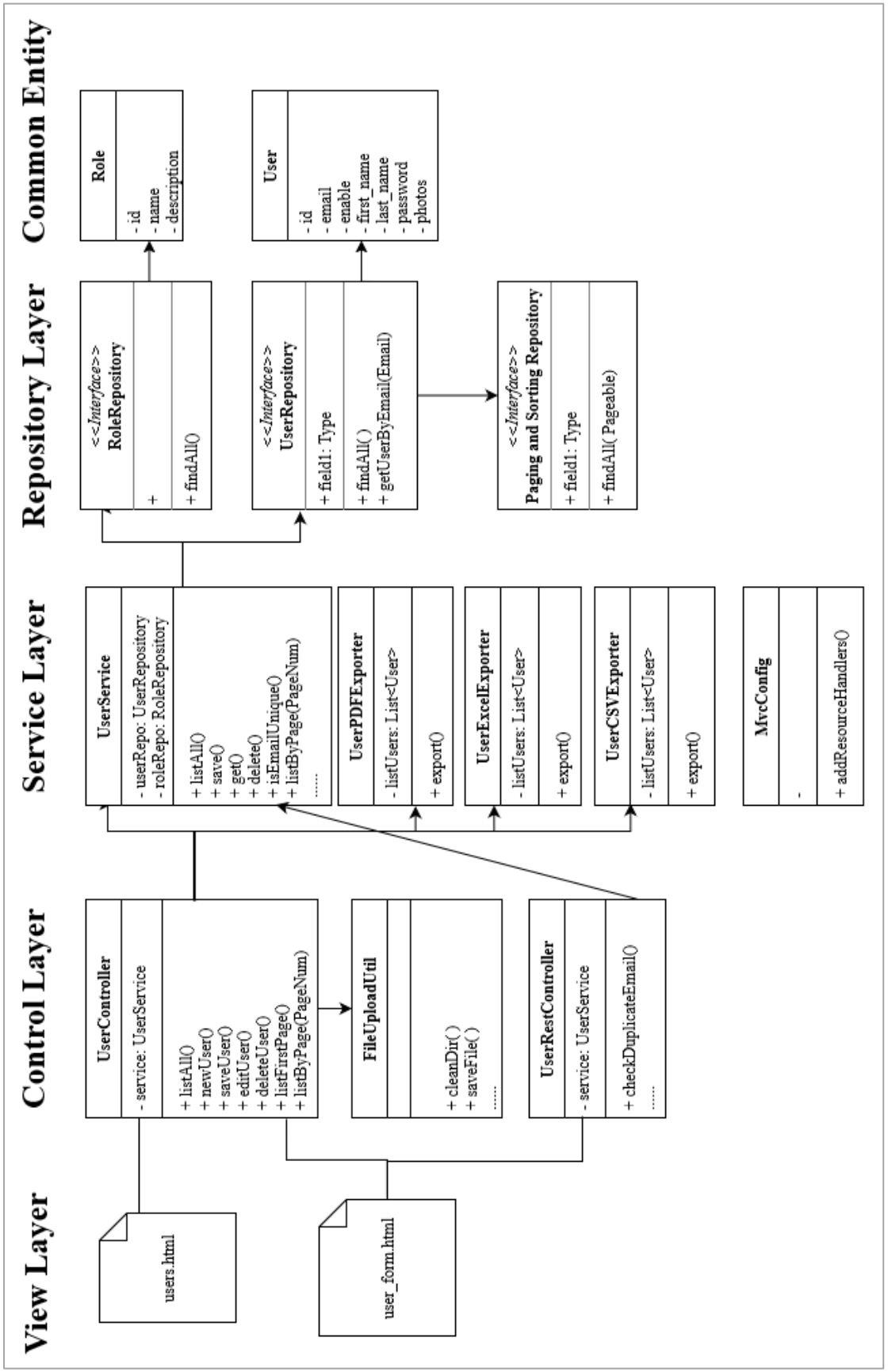

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
checkout.html
CheckoutController
- chekoutService: CheckoutService
- orderService: OrderService
- customerService: CustomerService
- cartService: CartService
- addressService: AddressService
- shipService: ShipService
+ showCheckoutPage( )
+ processPaypalOrder( )
+ placeOrder( )
+ sendOrderConfirmationEmail( )
CheckoutService
- DIM_DIVISOR
+ prepareCheckout( )
+ calculateProductCost( )
+ calculateProductTotal( )
+ calculateShippingCost( )
+ calculateDeliverDate( )
PaypalService
- restTemplate:RestTemplate
- settingService:SettingService
+ validateOrder( )
+ getOrderDetail( )
+ makeRequest( )
PaypalOrderResponse
- id:String
- status:String
+ validateOrder(OrderId):boolean
CheckoutInfo
+ productCost( )
+ productTotal( )
+ shippingCostTotal( )
+ paymentTotal( )
+ deliverDATE( )

E: Module UML Diagram

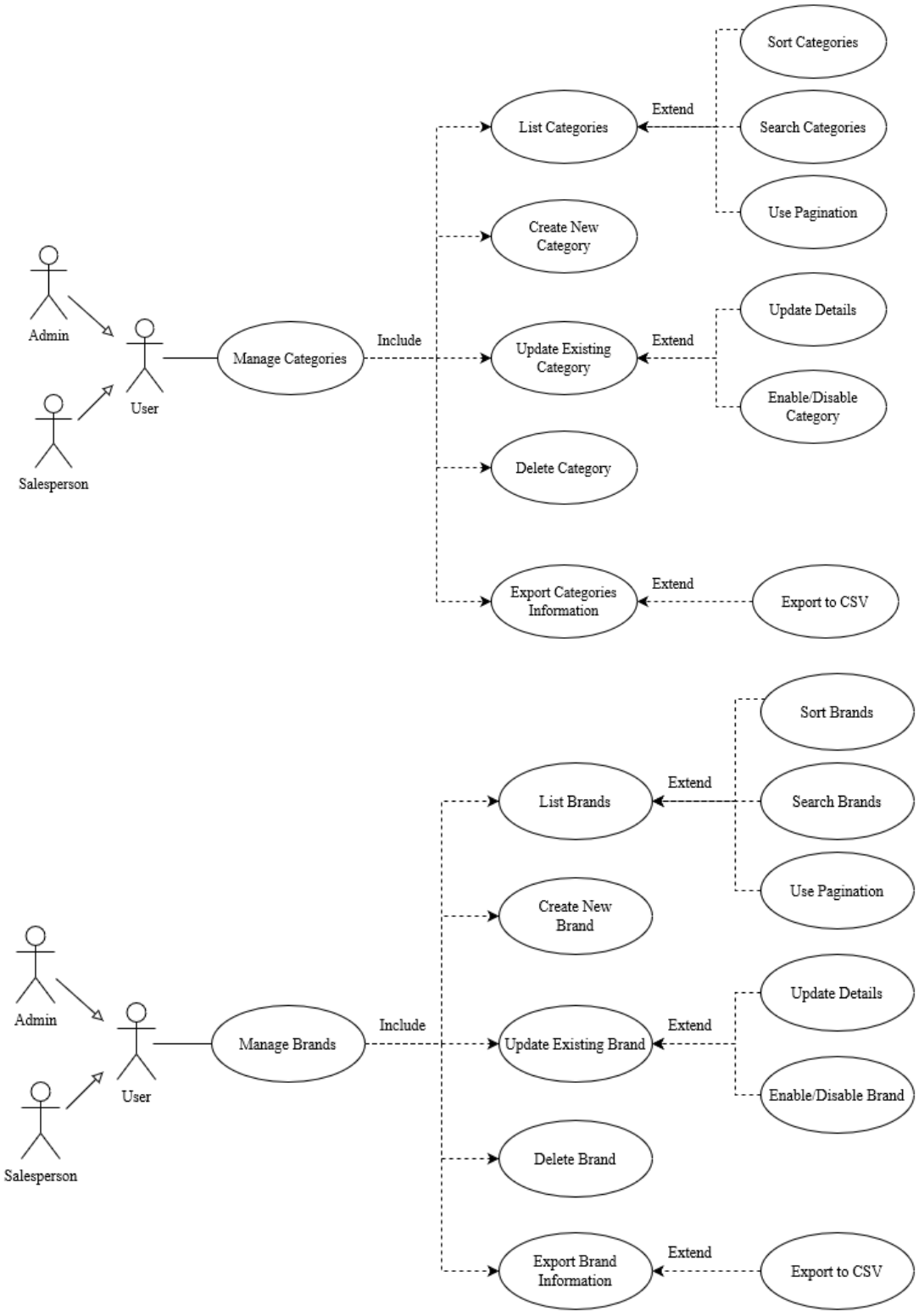
E1 User Module

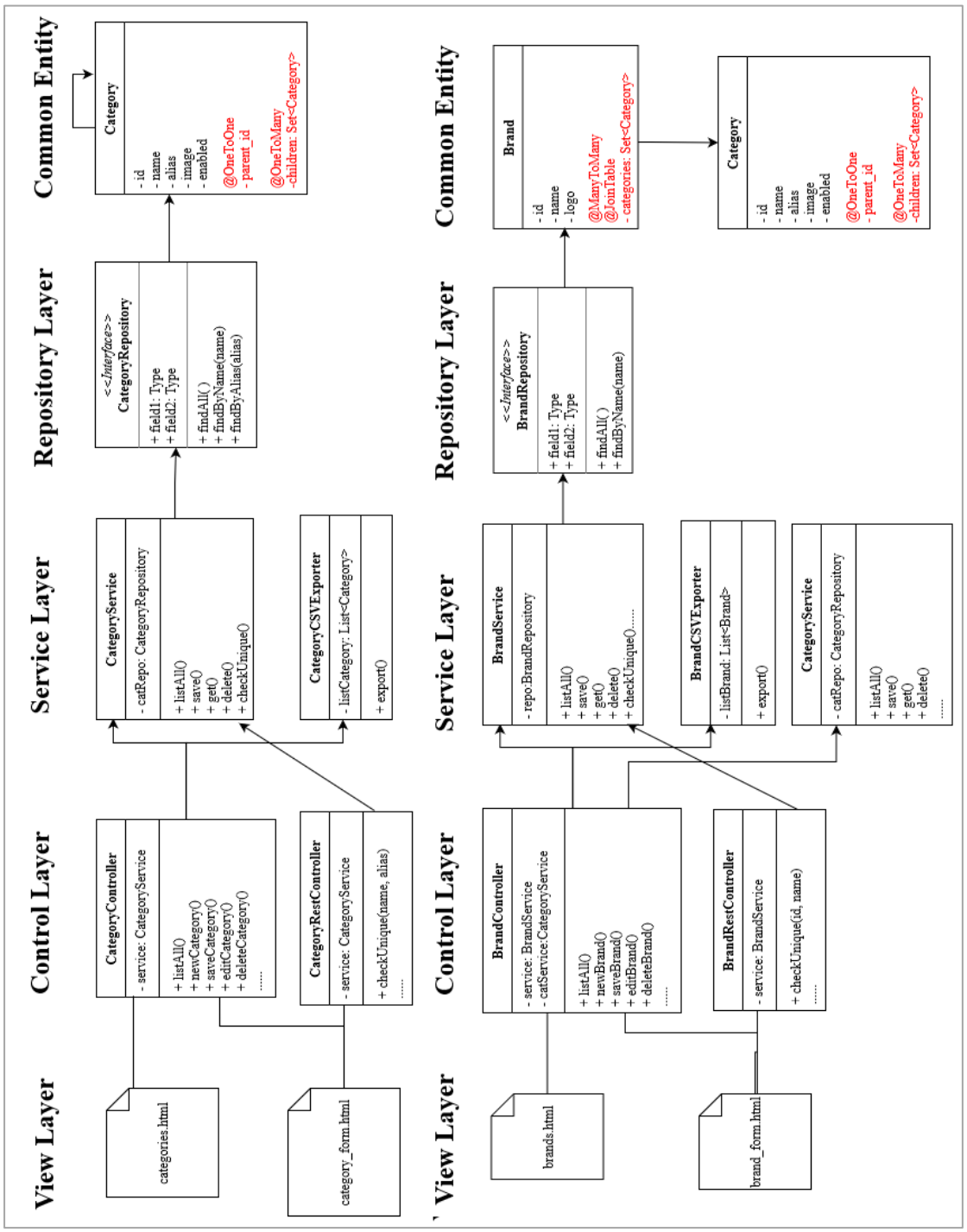




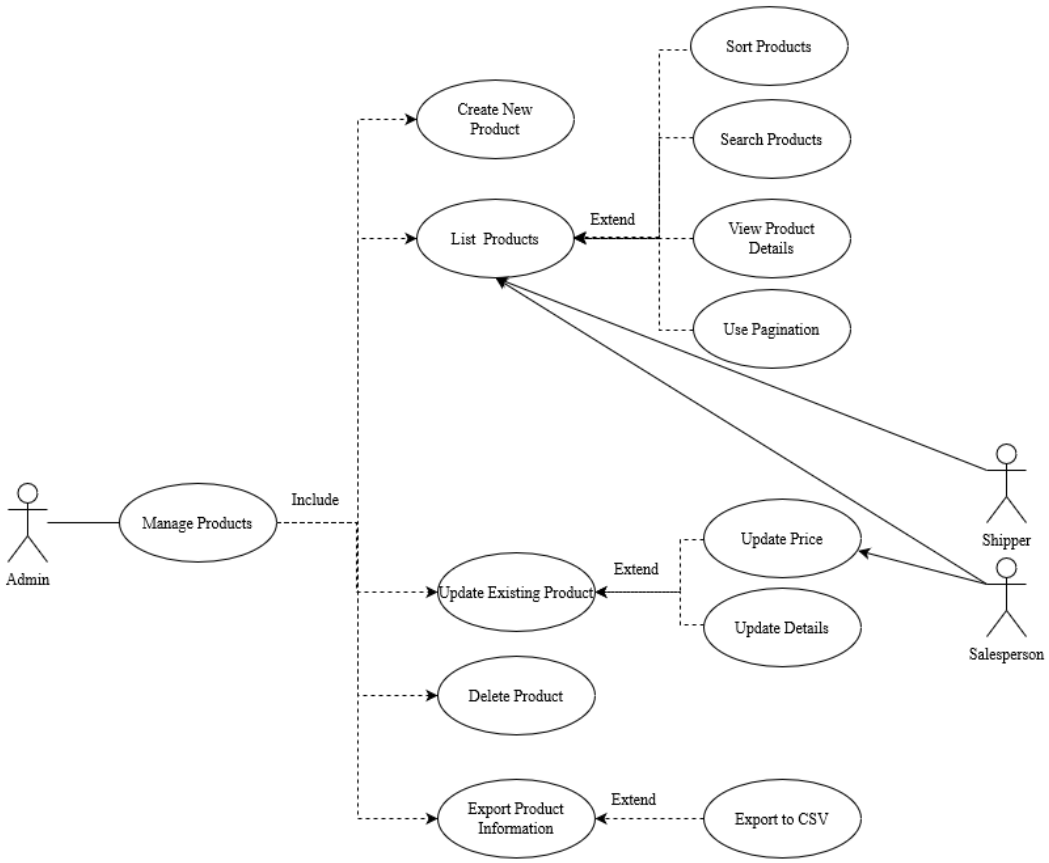

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
users.html
user_form.html
UserController
- service: UserService
+ listAll()
+ newUser()
+ saveUser()
+ editUser()
+ deleteUser()
+ listFirstPage()
+ listByPage(PageNum)
FileUploadUtil
+ cleanDir( )
+ saveFile( )
.....
UserRestController
- service: UserService
+ checkDuplicateEmail()
.....
UserService
- userRepo: UserRepository
- roleRepo: RoleRepository
+ listAll()
+ save()
+ get()
+ delete()
+ isEmailUnique()
+ listByPage(PageNum)
.....
UserPDFExporter
- listUsers: List<User>
+ export()
UserExcelExporter
- listUsers: List<User>
+ export()
UserCSVExporter
- listUsers: List<User>
+ export()
MvcConfig
-
+ addResourceHandlers()
<<Interface>>
RoleRepository
+
+ findAll()
<<Interface>>
UserRepository
+ field1: Type
+ findAll( )
+ getUserByEmail(Email)
<<Interface>>
Paging and Sorting Repository
+ field1: Type
+ findAll(Pageable)
Role
- id
- name
- description
User
- id
- email
- enable
- first_name
- last_name
- password
- photos

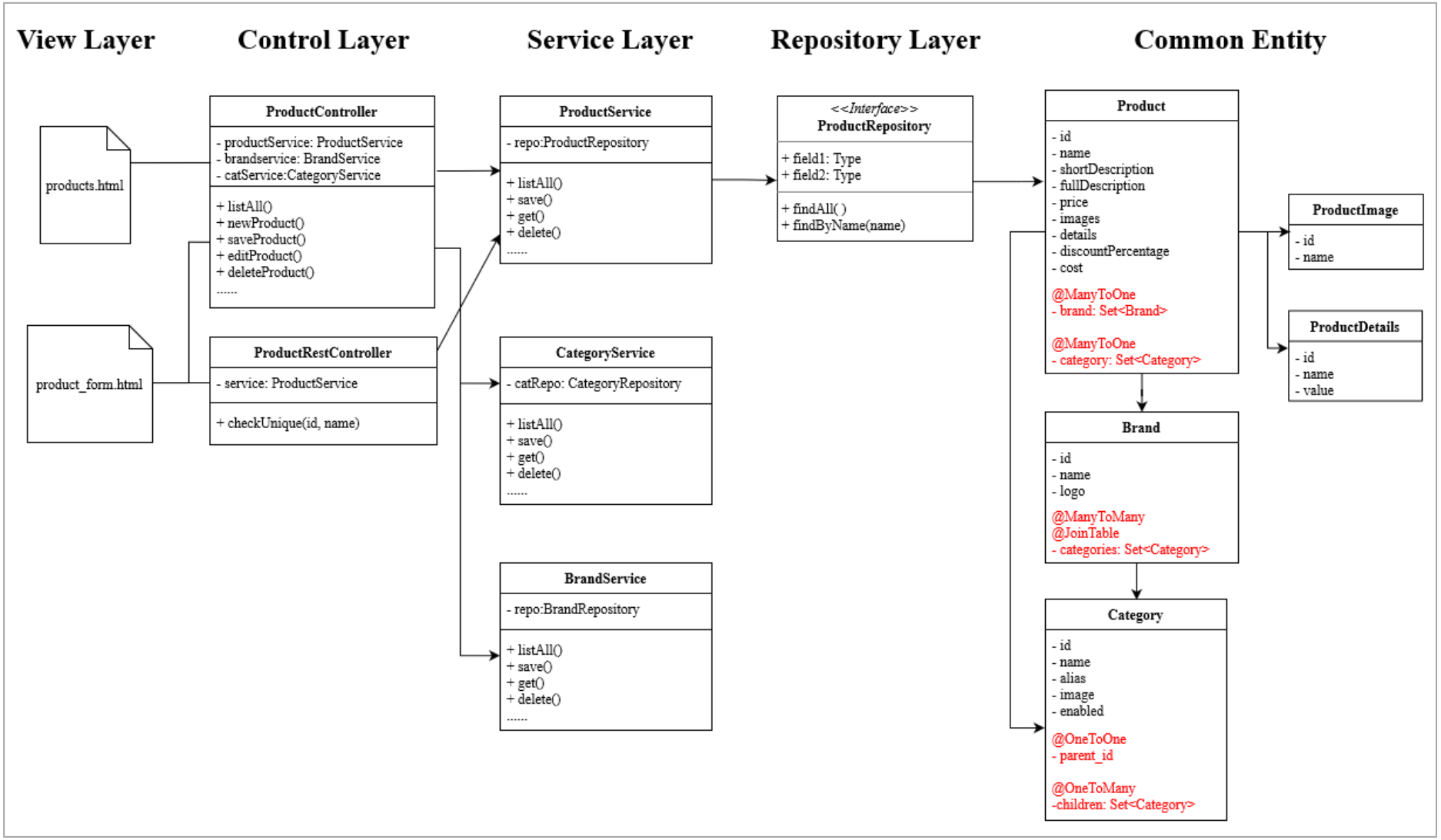
E2 Category and Brand Module

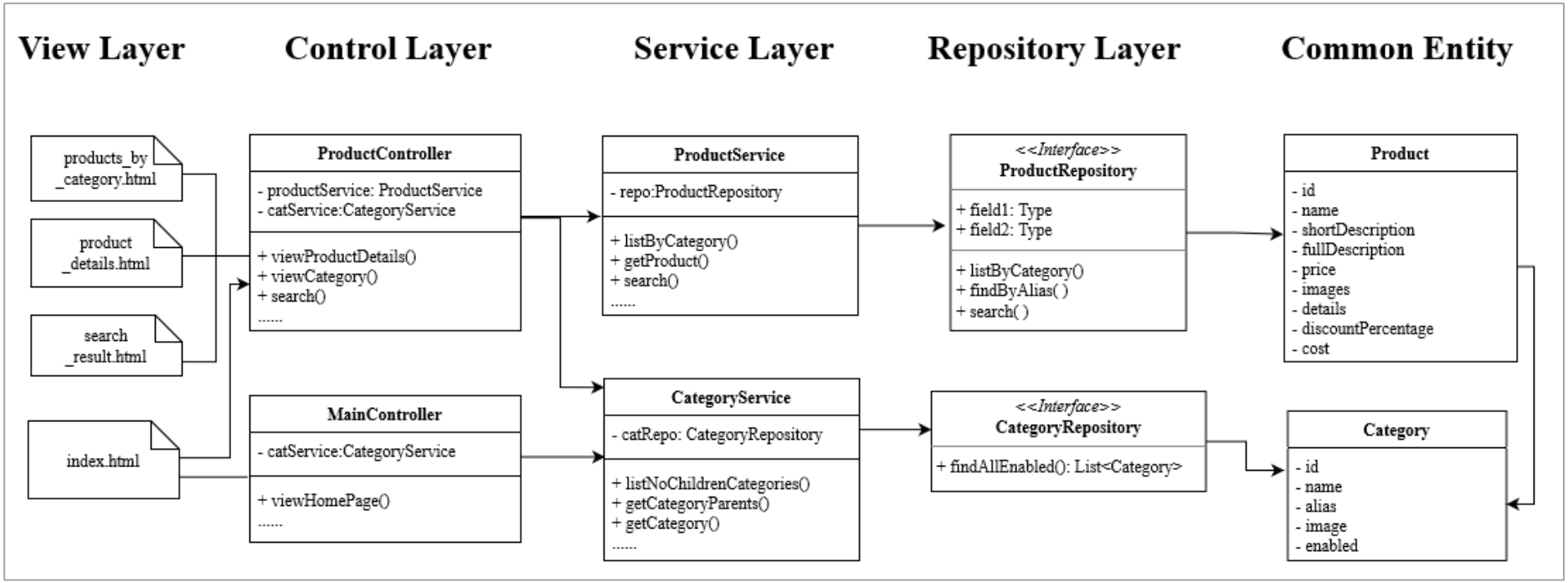



View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
categories.html
CategoryController
- service: CategoryService
+ listAll()
+ newCategory()
+ saveCategory()
+ editCategory()
+ deleteCategory()
.....
CategoryService
- catRepo: CategoryRepository
+ listAll()
+ save()
+ get()
+ delete()
+ checkUnique()
<<Interface>>
CategoryRepository
+ field1: Type
+ field2: Type
+ findAll()
+ findByName(name)
+ findByAlias(alias)
Category
- id
- name
- alias
- image
- enabled
@OneToOne
- parent_id
@OneToMany
-children: Set<Category>
category_form.html
CategoryRestController
- service: CategoryService
+ checkUnique(name, alias)
.....
CategoryCSVExporter
- listCategory: List<Category>
+ export()
brands.html
BrandController
- service: BrandService
- catService:CategoryService
+ listAll()
+ newBrand()
+ saveBrand()
+ editBrand()
+ deleteBrand()
.....
BrandService
- repo:BrandRepository
+ listAll()
+ save()
+ get()
+ delete()
+ checkUnique().....
<<Interface>>
BrandRepository
+ field1: Type
+ field2: Type
+ findAll()
+ findByName(name)
Brand
- id
- name
- logo
@ManyToMany
@JoinTable
- categories: Set<Category>
brand_form.html
BrandRestController
- service: BrandService
+ checkUnique(id, name)
.....
BrandCSVExporter
- listBrand: List<Brand>
+ export()
CategoryService
- catRepo: CategoryRepository
+ listAll()
+ save()
+ get()
+ delete()
.....
Category
- id
- name
- alias
- image
- enabled
@OneToOne
- parent_id
@OneToMany
-children: Set<Category>

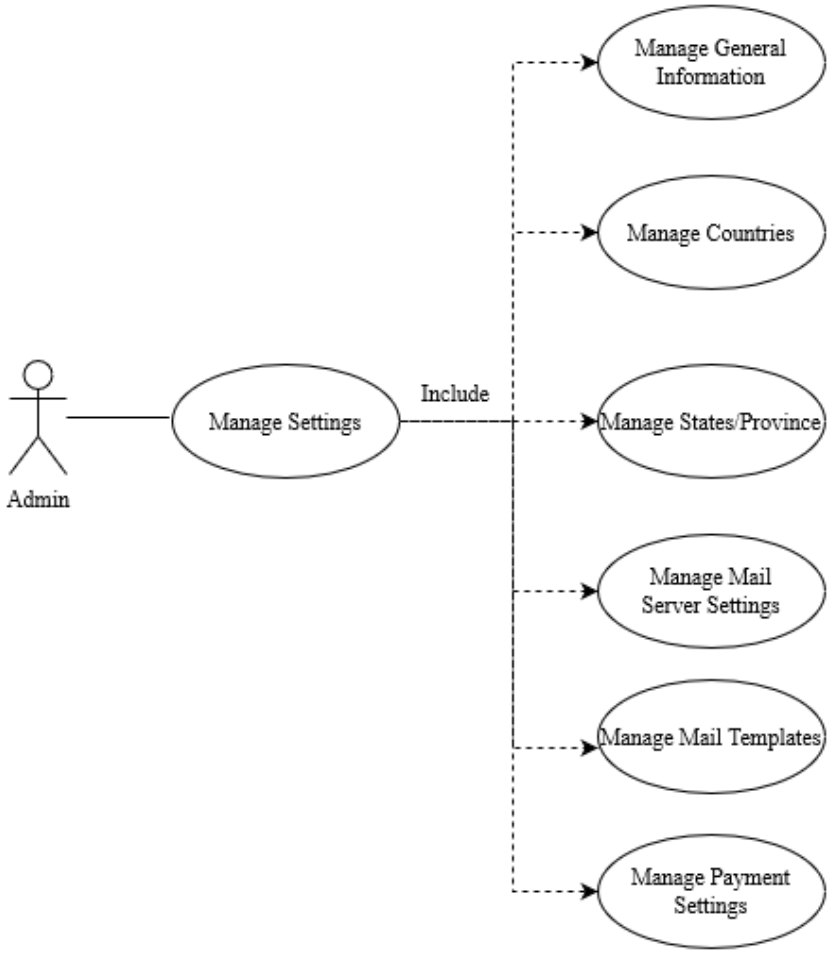
E3 Product Module

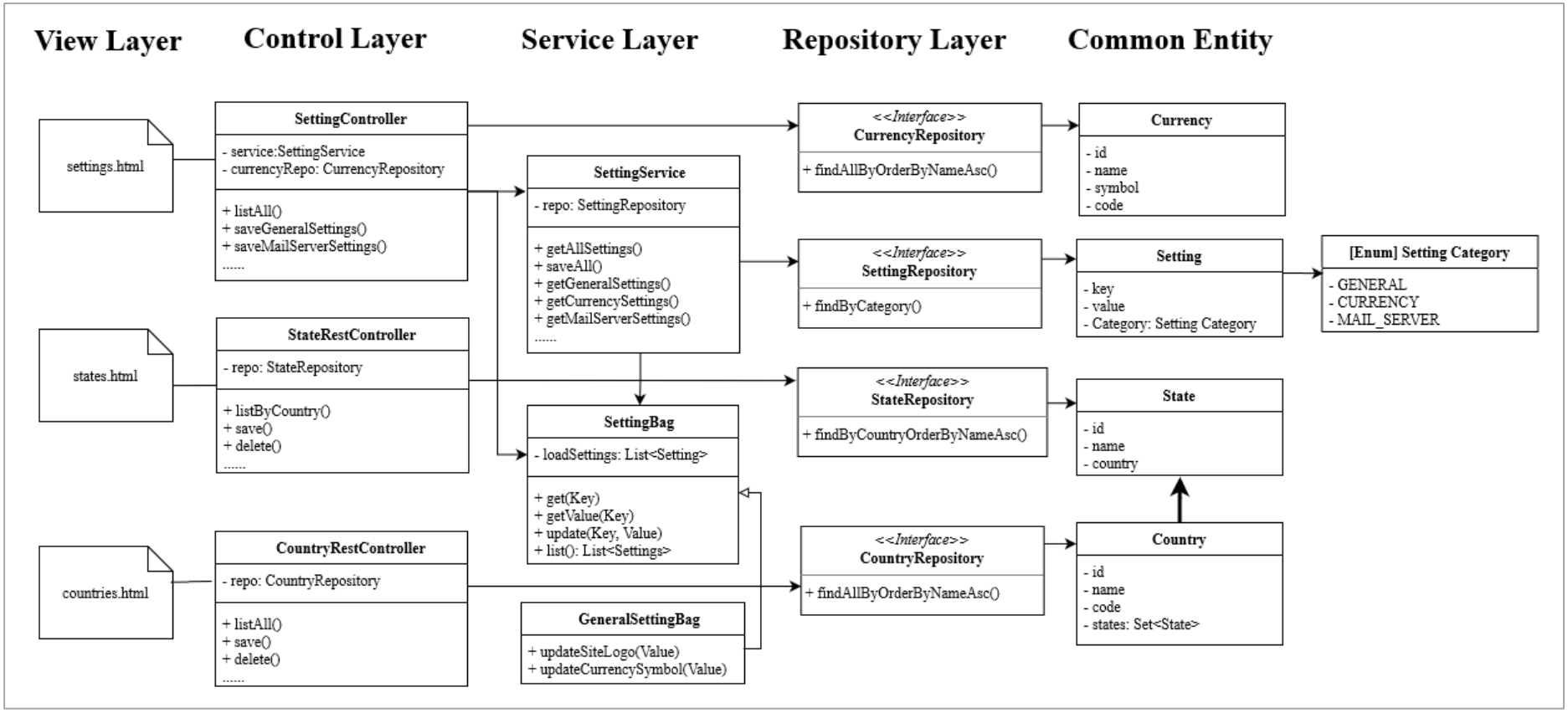



View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
products.html
product_form.html
ProductController
- productService: ProductService
- brandservice: BrandService
- catService:CategoryService
+ listAll()
+ newProduct()
+ saveProduct()
+ editProduct()
+ deleteProduct()
.....
ProductRestController
- service: ProductService
+ checkUnique(id, name)
ProductService
- repo:ProductRepository
+ listAll()
+ save()
+ get()
+ delete()
.....
CategoryService
- catRepo: CategoryRepository
+ listAll()
+ save()
+ get()
+ delete()
.....
BrandService
- repo:BrandRepository
+ listAll()
+ save()
+ get()
+ delete()
.....
<<Interface>>
ProductRepository
+ field1: Type
+ field2: Type
+ findAll( )
+ findByName(name)
Product
- id
- name
- shortDescription
- fullDescription
- price
- images
- details
- discountPercentage
- cost
@ManyToOne
- brand: Set<Brand>
@ManyToOne
- category: Set<Category>
ProductImage
- id
- name
ProductDetails
- id
- name
- value
Brand
- id
- name
- logo
@ManyToMany
@JoinTable
- categories: Set<Category>
Category
- id
- name
- alias
- image
- enabled
@OneToOne
- parent_id
@OneToMany
-children: Set<Category>

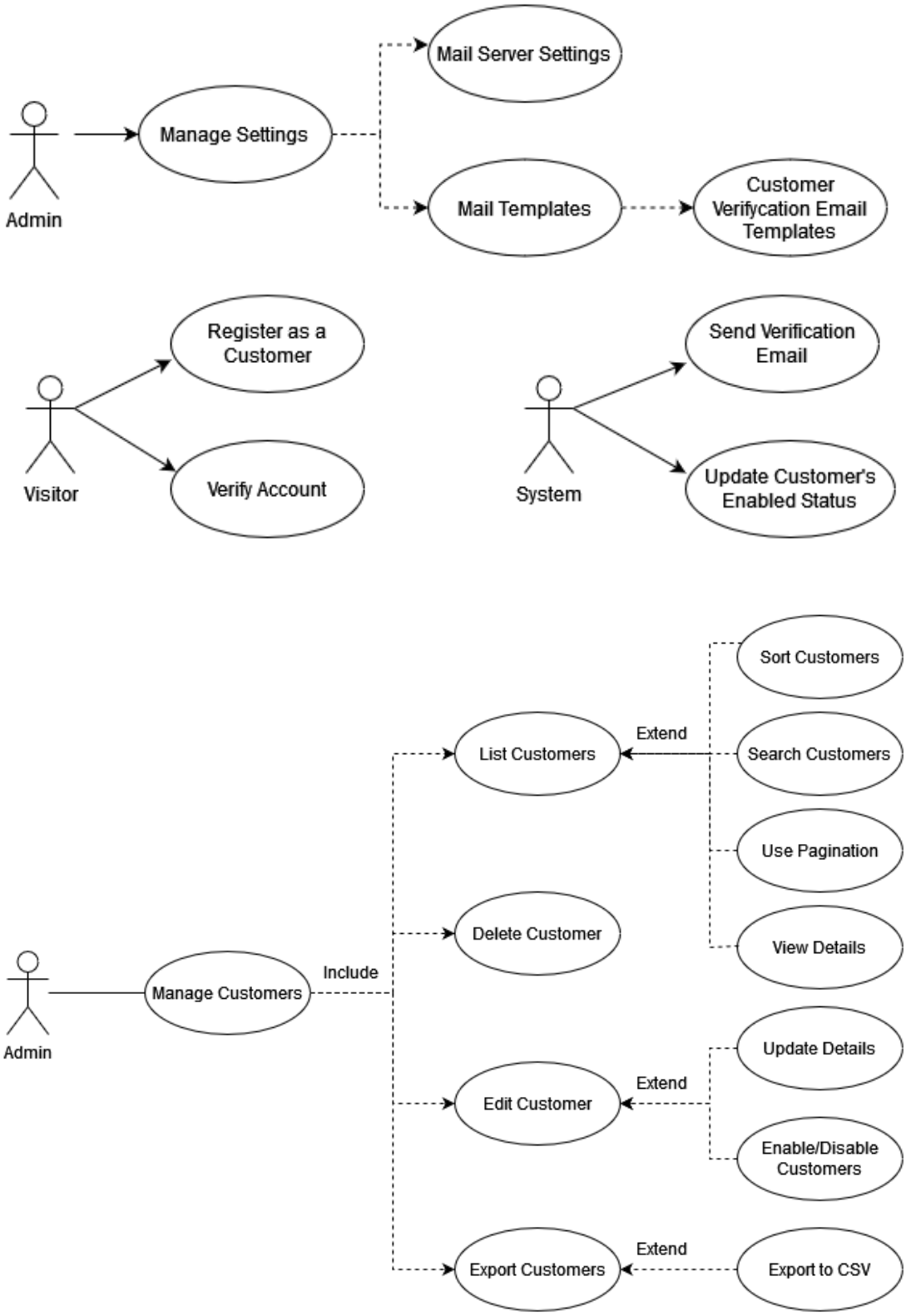

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
products_by _category.html
product _details.html
search _result.html
index.html
ProductController
- productService: ProductService
- catService:CategoryService
+ viewProductDetails()
+ viewCategory()
+ search()
.....
MainController
- catService:CategoryService
+ viewHomePage()
.....
ProductService
- repo:ProductRepository
+ listByCategory()
+ getProduct()
+ search()
.....
CategoryService
- catRepo: CategoryRepository
+ listNoChildrenCategories()
+ getCategoryParents()
+ getCategory()
.....
<<Interface>>
ProductRepository
+ field1: Type
+ field2: Type
+ listByCategory()
+ findByAlias( )
+ search( )
<<Interface>>
CategoryRepository
+ findAllEnabled(): List<Category>
Product
- id
- name
- shortDescription
- fullDescription
- price
- images
- details
- discountPercentage
- cost
Category
- id
- name
- alias
- image
- enabled

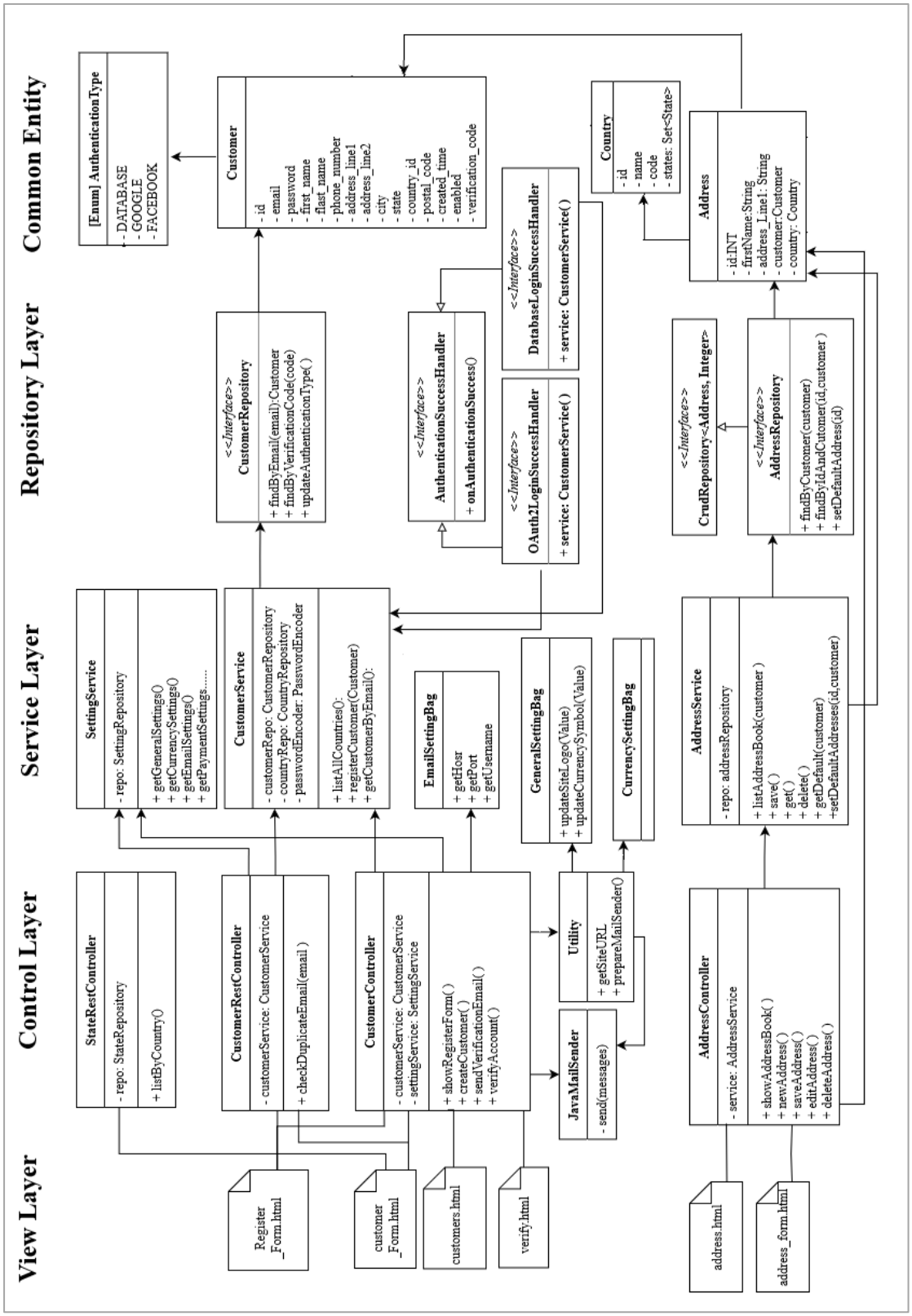
E4 Setting Module

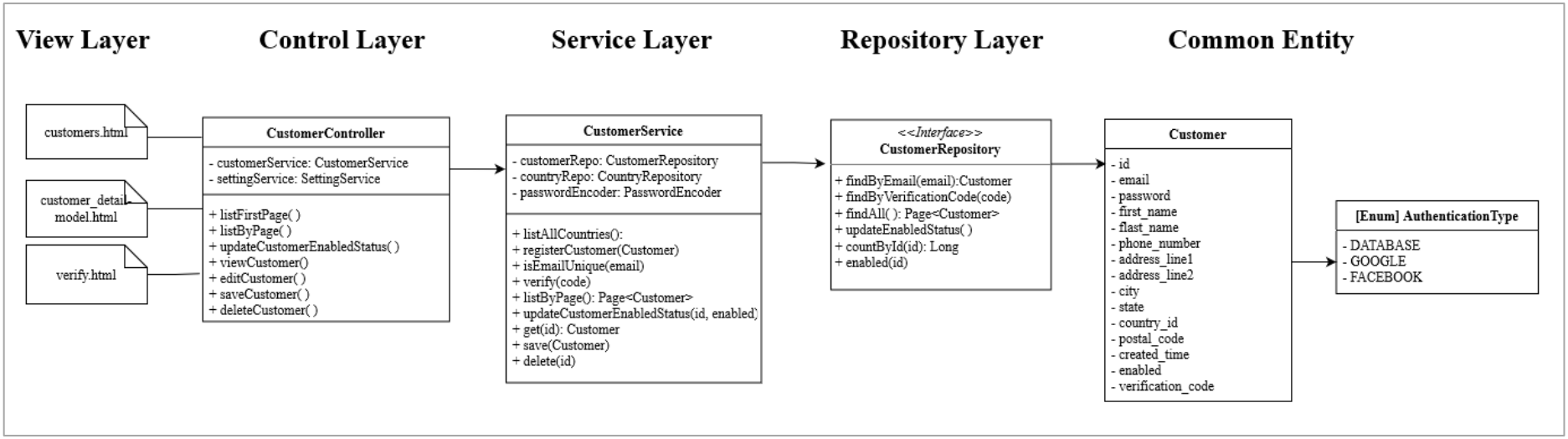



View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
settings.html
states.html
countries.html
SettingController
- service:SettingService
- currencyRepo: CurrencyRepository
+ listAll()
+ saveGeneralSettings()
+ saveMailServerSettings()
.....
StateRestController
- repo: StateRepository
+ listByCountry()
+ save()
+ delete()
.....
CountryRestController
- repo: CountryRepository
+ listAll()
+ save()
+ delete()
.....
SettingService
- repo: SettingRepository
+ getAllSettings()
+ saveAll()
+ getGeneralSettings()
+ getCurrencySettings()
+ getMailServerSettings()
.....
SettingBag
- loadSettings: List<Setting>
+ get(Key)
+ getValue(Key)
+ update(Key, Value)
+ list(): List<Settings>
GeneralSettingBag
+ updateSiteLogo(Value)
+ updateCurrencySymbol(Value)
<<Interface>>
CurrencyRepository
+ findAllByOrderByNameAsc()
<<Interface>>
SettingRepository
+ findByCategory()
<<Interface>>
StateRepository
+ findByCountryOrderByNameAsc()
<<Interface>>
CountryRepository
+ findAllByOrderByNameAsc()
Currency
- id
- name
- symbol
- code
Setting
- key
- value
- Category: Setting Category
[Enum] Setting Category
- GENERAL
- CURRENCY
- MAIL_SERVER
State
- id
- name
- country
Country
- id
- name
- code
- states: Set<State>

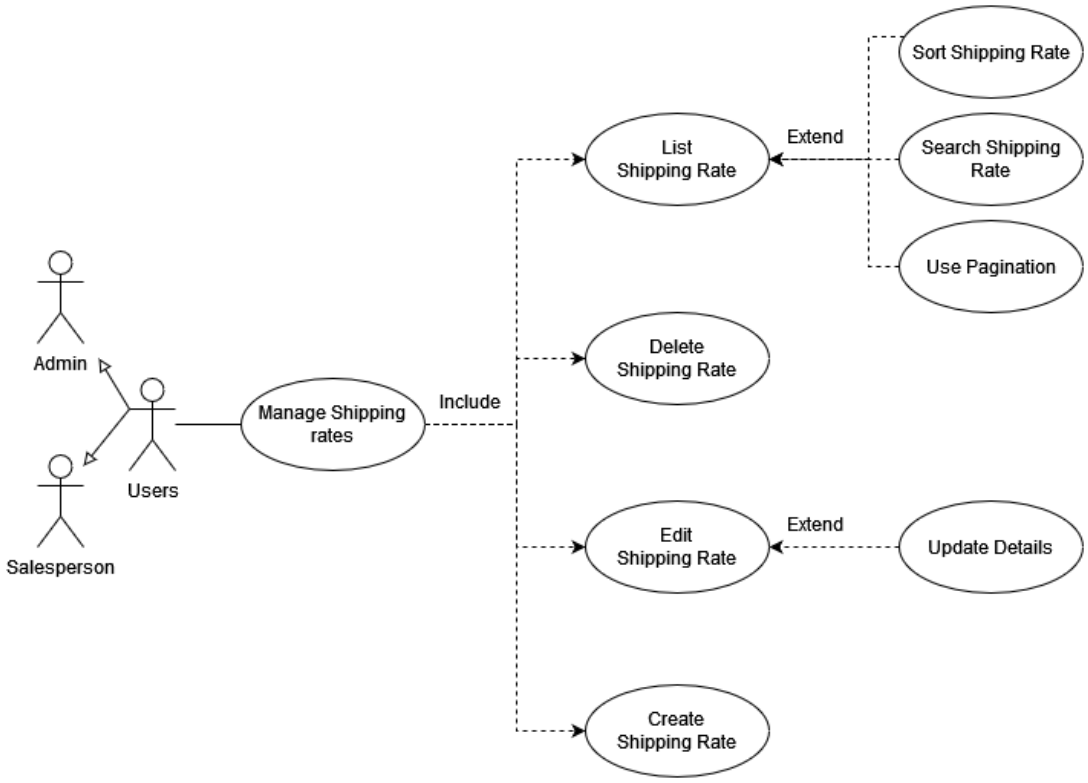
E5 Customer Module

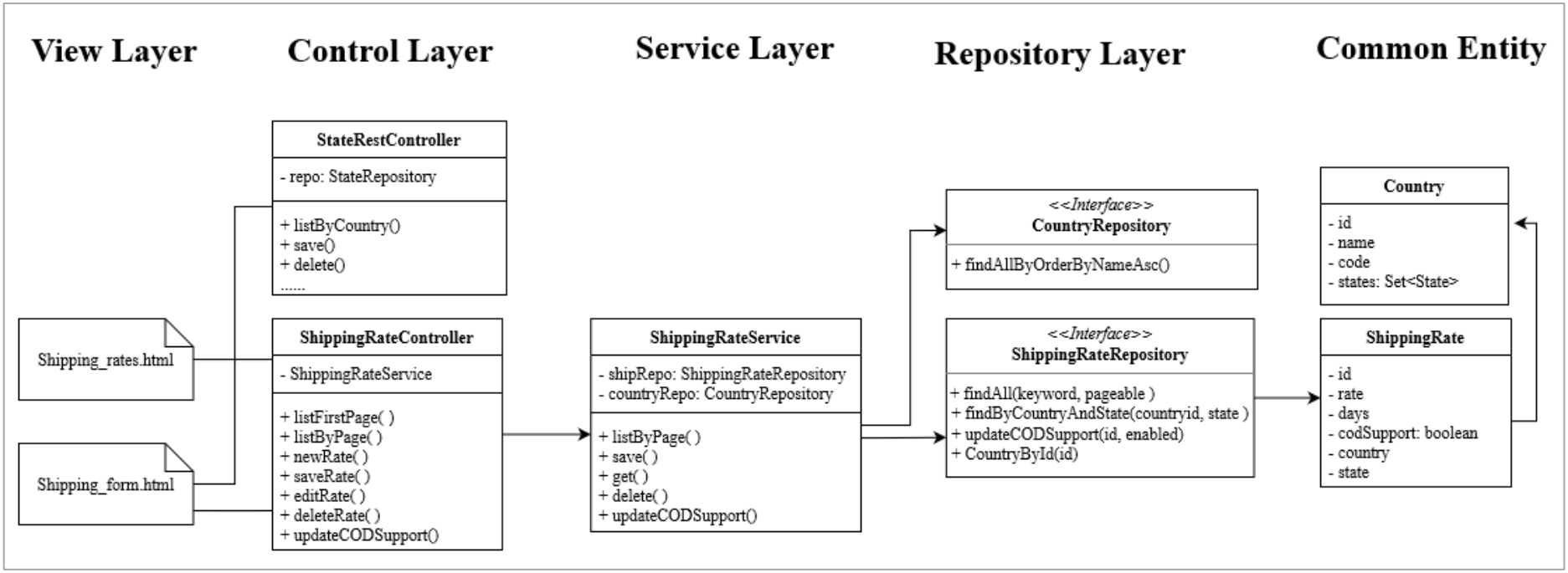



View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
[Enum] AuthenticationType
- DATABASE
- GOOGLE
- FACEBOOK
Customer
- id
- email
- password
- first_name
- last_name
- phone_number
- address_line1
- address_line2
- city
- state
- country_id
- postal_code
- created_time
- enabled
- verification_code
Country
- id
- name
- code
- states: Set<State>
Address
- id:INT
- firstName:String
- address_Line1: String
- customer:Customer
- country: Country
SettingService
- repo: SettingRepository
+ getGeneralSettings()
+ getCurrencySettings()
+ getEmailSettings()
+ getPaymentSettings.....
CustomerService
- customerRepo: CustomerRepository
- countryRepo: CountryRepository
- passwordEncoder: PasswordEncoder
+ listAllCountries():
+ registerCustomer(Customer)
+ getCustomerByEmail():
EmailSettingBag
+ getHost
+ getPort
+ getUsername
GeneralSettingBag
+ updateSiteLogo(Value)
+ updateCurrencySymbol(Value)
CurrencySettingBag
AddressService
- repo: addressRepository
+ listAddressBook(customer)
+ save( )
+ get( )
+ delete( )
+ getDefault(customer)
+setDefaultAddresses(id,customer)
<<Interface>>
CustomerRepository
+ findByEmail(email):Customer
+ findByVerificationCode(code)
+ updateAuthenticationType( )
<<Interface>>
AuthenticationSuccessHandler
+ onAuthenticationSuccess()
<<Interface>>
DatabaseLoginSuccessHandler
+ service: CustomerService( )
<<Interface>>
OAuth2LoginSuccessHandler
+ service: CustomerService( )
<<Interface>>
CrudRepository<Address, Integer>
<<Interface>>
AddressRepository
+ findByCustomer(customer)
+ findByIdAndCutomer(id,customer)
+ setDefaultAddress(id)
StateRestController
- repo: StateRepository
+ listByCountry()
CustomerRestController
- customerService: CustomerService
+ checkDuplicateEmail(email)
CustomerController
- customerService: CustomerService
- settingService: SettingService
+ showRegisterForm( )
+ createCustomer( )
+ sendVerificationEmail( )
+ verify.Account( )
Utility
+ getSiteURL
+ prepareMailSender()
JavaMailSender
- send(messages)
AddressController
- service: AddressService
+ showAddressBook( )
+ newAddress( )
+ saveAddress( )
+ editAddress( )
+ deleteAddress( )
Register_Form.html
customer_Form.html
customers.html
verify.html
address.html
address_form.html

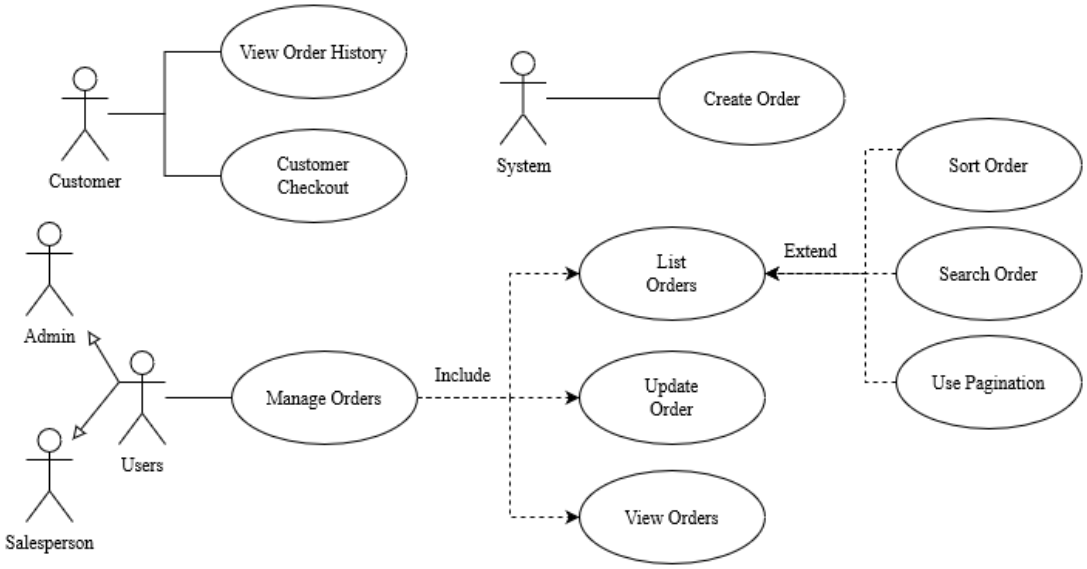

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
customers.html
customer_detail-model.html
verify.html
CustomerController
- customerService: CustomerService
- settingService: SettingService
+ listFirstPage( )
+ listByPage( )
+ updateCustomerEnabledStatus( )
+ viewCustomer()
+ editCustomer( )
+ saveCustomer( )
+ deleteCustomer( )
CustomerService
- customerRepo: CustomerRepository
- countryRepo: CountryRepository
- passwordEncoder: PasswordEncoder
+ listAllCountries():
+ registerCustomer(Customer)
+ isEmailUnique(email)
+ verify(code)
+ listByPage(): Page<Customer>
+ updateCustomerEnabledStatus(id, enabled)
+ get(id): Customer
+ save(Customer)
+ delete(id)
<<Interface>>
CustomerRepository
+ findByEmail(email):Customer
+ findByVerificationCode(code)
+ findAll( ): Page<Customer>
+ updateEnabledStatus( )
+ countById(id): Long
+ enabled(id)
Customer
- id
- email
- password
- first_name
- flast_name
- phone_number
- address_line1
- address_line2
- city
- state
- country_id
- postal_code
- created_time
- enabled
- verification_code
[Enum] AuthenticationType
- DATABASE
- GOOGLE
- FACEBOOK

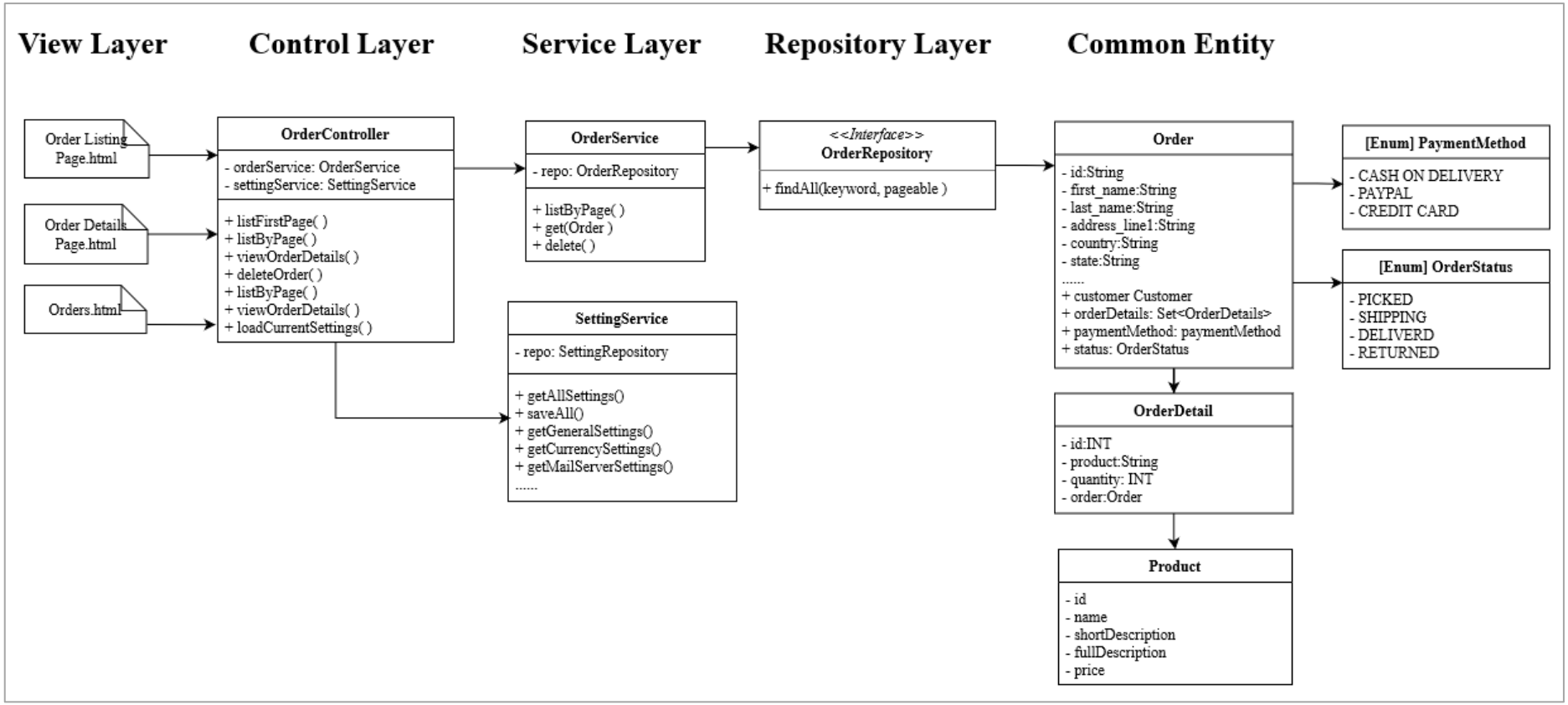
E6 Shipping Rate Module



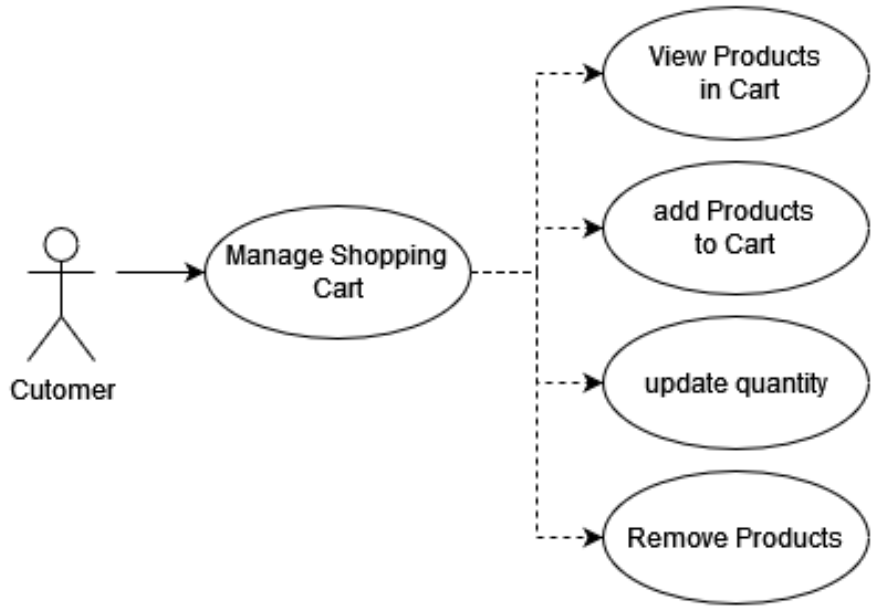

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
StateRestController
- repo: StateRepository
+ listByCountry()
+ save()
+ delete()
.....
ShippingRateController
- ShippingRateService
+ listFirstPage( )
+ listByPage( )
+ newRate( )
+ saveRate( )
+ editRate( )
+ deleteRate( )
+ updateCODSupport()
Shipping_rates.html
Shipping_form.html
ShippingRateService
- shipRepo: ShippingRateRepository
- countryRepo: CountryRepository
+ listByPage( )
+ save( )
+ get( )
+ delete( )
+ updateCODSupport()
<<Interface>>
CountryRepository
+ findAllByOrderByNameAsc()
<<Interface>>
ShippingRateRepository
+ findAll(keyword, pageable)
+ findByCountryAndState(countryid, state)
+ updateCODSupport(id, enabled)
+ CountryById(id)
Country
- id
- name
- code
- states: Set<State>
ShippingRate
- id
- rate
- days
- codSupport: boolean
- country
- state

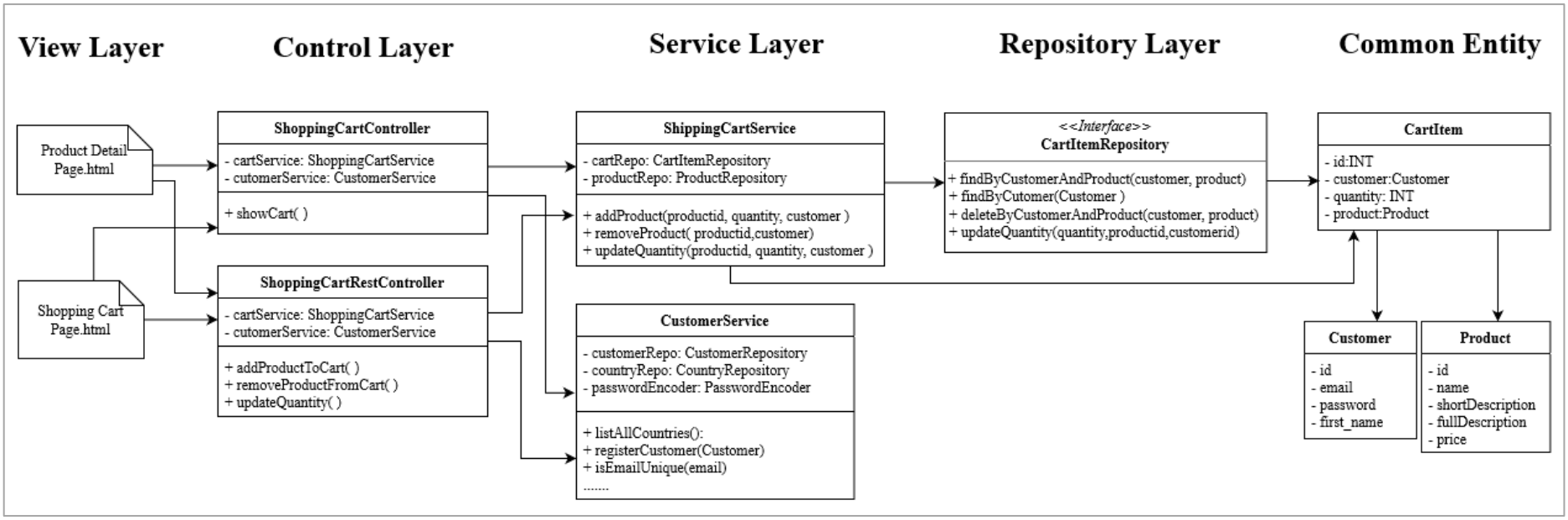
E7 Order Module



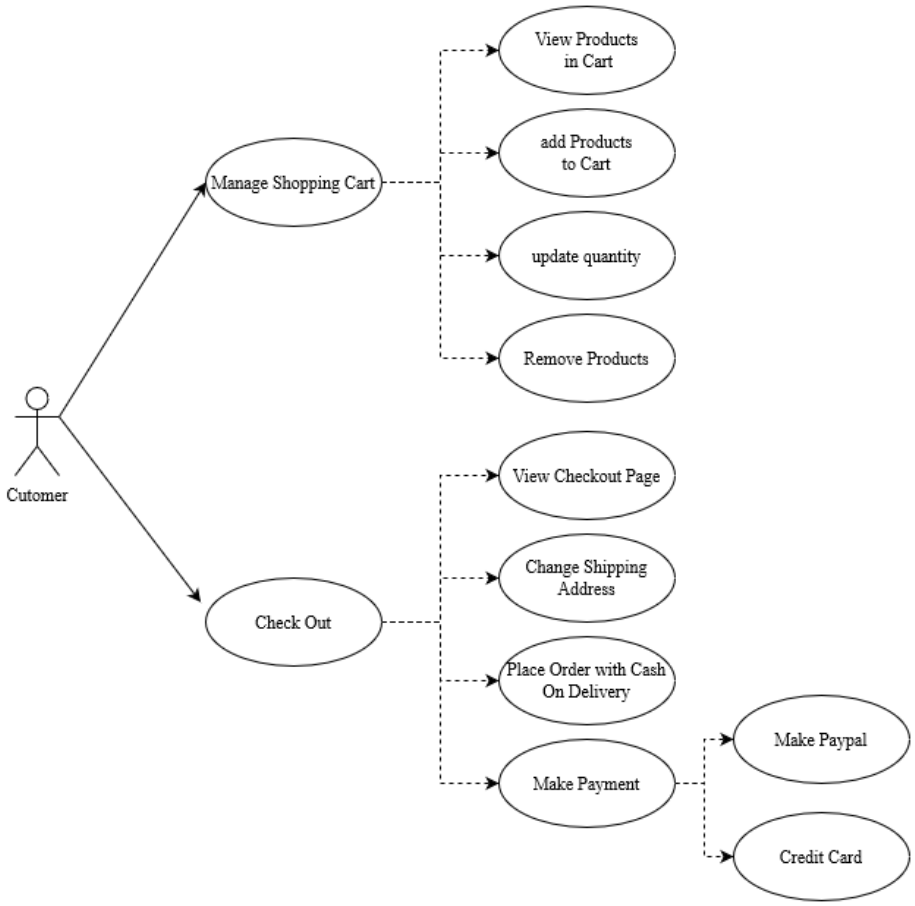

View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
Order Listing Page.html
Order Details Page.html
Orders.html
OrderController
- orderService: OrderService
- settingService: SettingService
+ listFirstPage( )
+ listByPage( )
+ viewOrderDetails( )
+ deleteOrder( )
+ listByPage( )
+ viewOrderDetails( )
+ loadCurrentSettings( )
OrderService
- repo: OrderRepository
+ listByPage( )
+ get(Order)
+ delete( )
SettingService
- repo: SettingRepository
+ getAllSettings()
+ saveAll()
+ getGeneralSettings()
+ getCurrencySettings()
+ getMailServerSettings()
.....
<<Interface>>
OrderRepository
+ findAll(keyword, pageable)
Order
- id:String
- first_name:String
- last_name:String
- address_line1:String
- country:String
- state:String
.....
+ customer Customer
+ orderDetails: Set<OrderDetails>
+ paymentMethod: paymentMethod
+ status: OrderStatus
[Enum] PaymentMethod
- CASH ON DELIVERY
- PAYPAL
- CREDIT CARD
[Enum] OrderStatus
- PICKED
- SHIPPING
- DELIVERD
- RETURNED
OrderDetail
- id:INT
- product:String
- quantity: INT
- order:Order
Product
- id
- name
- shortDescription
- fullDescription
- price

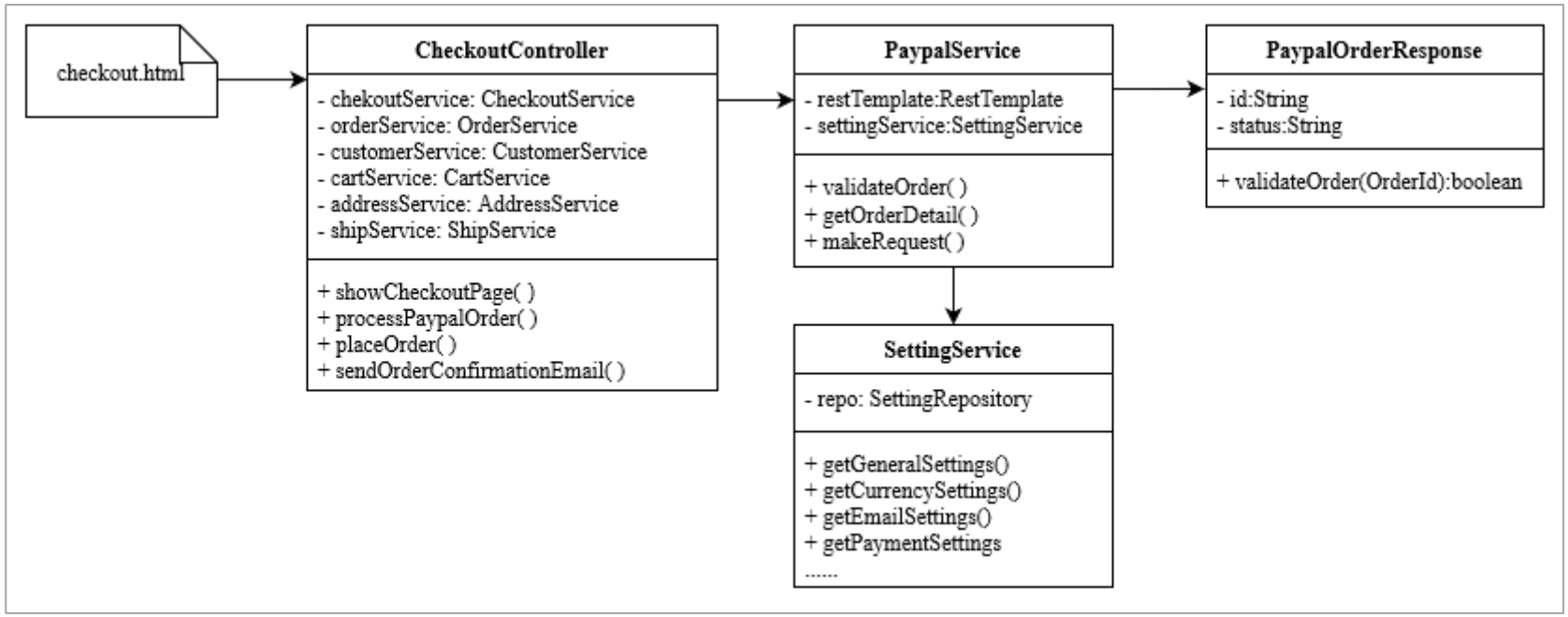
E8 Shopping Cart Module

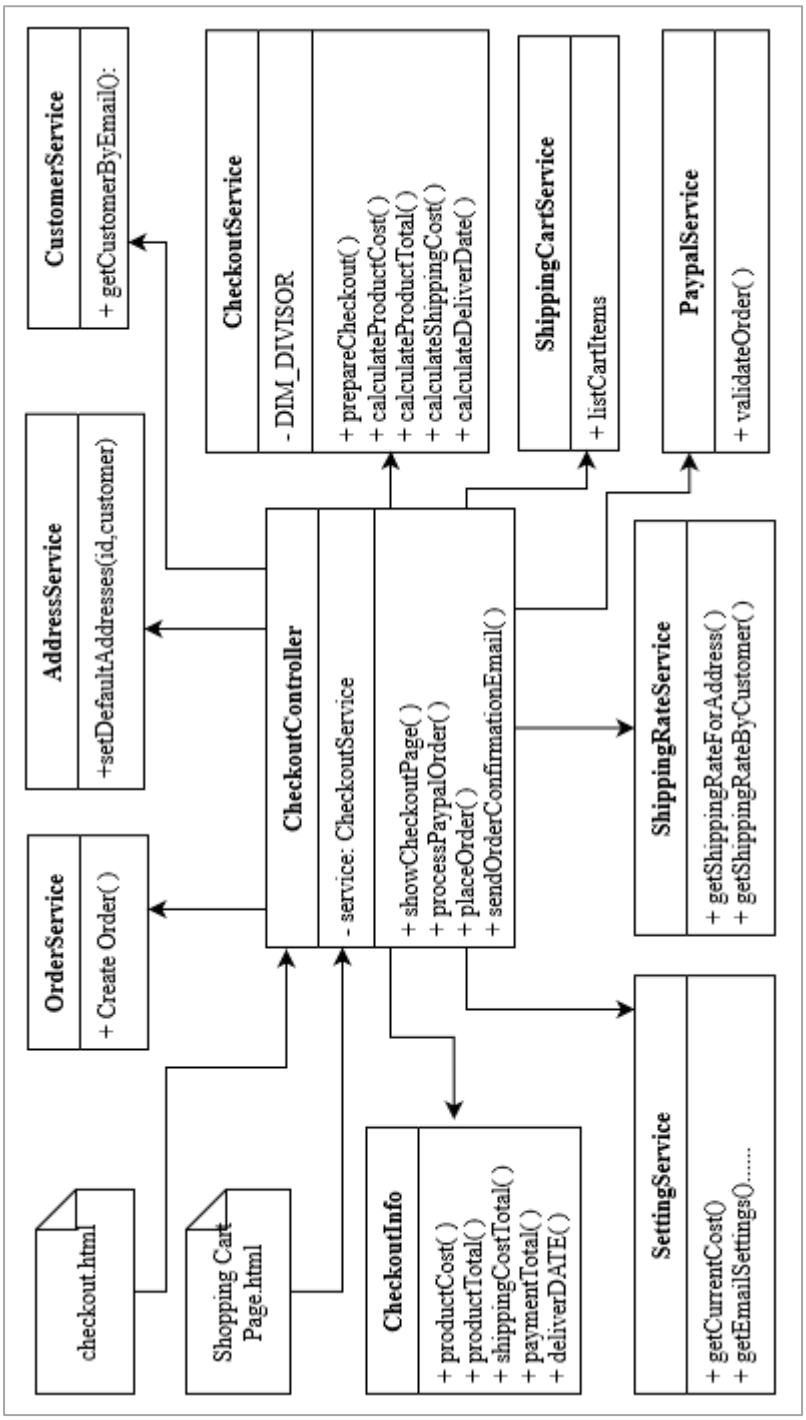



View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
Product Detail Page.html
Shopping Cart Page.html
ShoppingCartController
- cartService: ShoppingCartService
- cutomerService: CustomerService
+ showCart( )
ShoppingCartRestController
- cartService: ShoppingCartService
- cutomerService: CustomerService
+ addProductToCart( )
+ removeProductFromCart( )
+ updateQuantity( )
ShippingCartService
- cartRepo: CartItemRepository
- productRepo: ProductRepository
+ addProduct(productid, quantity, customer)
+ removeProduct(productid,customer)
+ updateQuantity(productid, quantity, customer)
CustomerService
- customerRepo: CustomerRepository
- countryRepo: CountryRepository
- passwordEncoder: PasswordEncoder
+ listAllCountries():
+ registerCustomer(Customer)
+ isEmailUnique(email)
.....
<<Interface>>
CartItemRepository
+ findByCustomerAndProduct(customer, product)
+ findByCutomer(Customer)
+ deleteByCustomerAndProduct(customer, product)
+ updateQuantity(quantity,productid,customerid)
CartItem
- id:INT
- customer:Customer
- quantity: INT
- product:Product
Customer
- id
- email
- password
- first_name
Product
- id
- name
- shortDescription
- fullDescription
- price

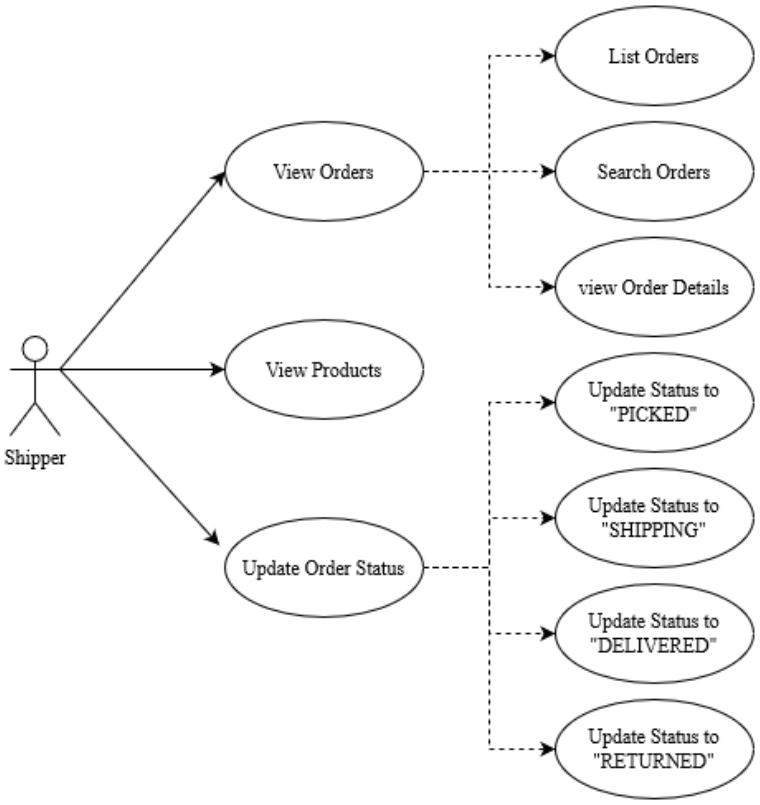
E9 Checkout Module

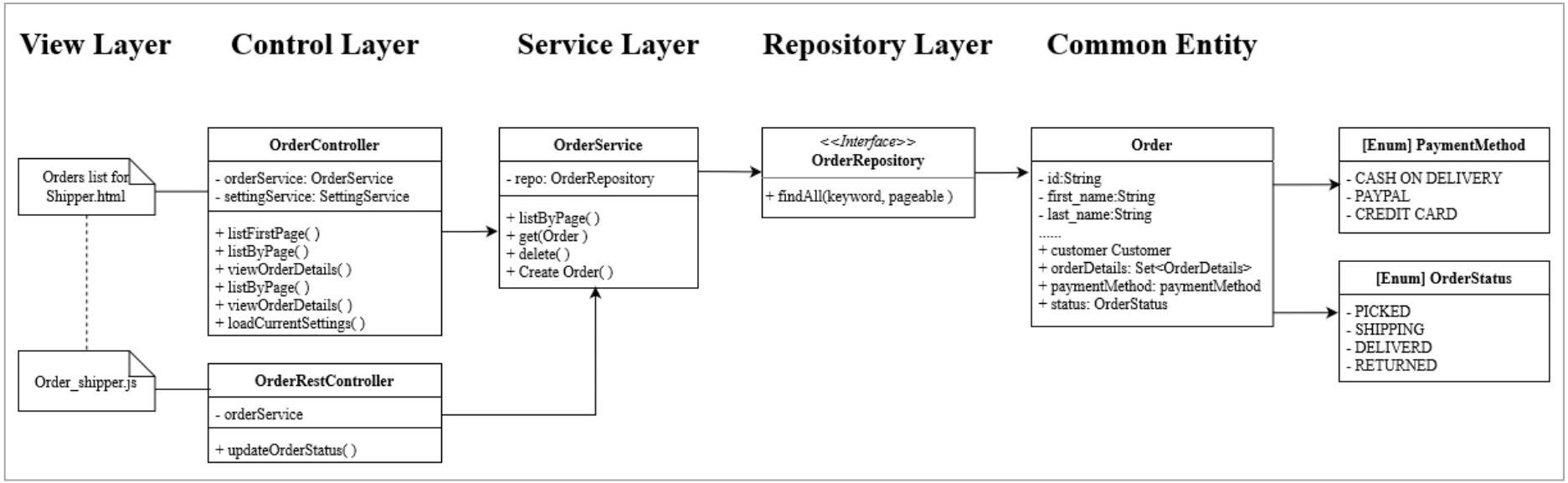



checkout.html
CheckoutController
- chekoutService: CheckoutService
- orderService: OrderService
- customerService: CustomerService
- cartService: CartService
- addressService: AddressService
- shipService: ShipService
+ showCheckoutPage( )
+ processPaypalOrder( )
+ placeOrder( )
+ sendOrderConfirmationEmail( )
PaypalService
- restTemplate:RestTemplate
- settingService:SettingService
+ validateOrder( )
+ getOrderDetail( )
+ makeRequest( )
SettingService
- repo: SettingRepository
+ getGeneralSettings()
+ getCurrencySettings()
+ getEmailSettings()
+ getPaymentSettings
.....
PaypalOrderResponse
- id:String
- status:String
+ validateOrder(OrderId):boolean


CustomerService
+ getCustomerByEmail():
AddressService
+setDefaultAddresses(id,customer)
CheckoutService
- DIM_DIVISOR
+ prepareCheckout( )
+ calculateProductCost( )
+ calculateProductTotal( )
+ calculateShippingCost( )
+ calculateDeliverDate( )
ShippingCartService
+ listCartItems
PaypalService
+ validateOrder( )
OrderService
+ Create Order( )
CheckoutController
- service: CheckoutService
+ showCheckoutPage( )
+ processPaypalOrder( )
+ placeOrder( )
+ sendOrderConfirmationEmail( )
ShippingRateService
+ getShippingRateForAddress( )
+ getShippingRateByCustomer( )
checkout.html
Shopping Cart Page.html
CheckoutInfo
+ productCost( )
+ productTotal( )
+ shippingCostTotal( )
+ paymentTotal( )
+ deliverDATE( )
SettingService
+ getCurrentCost()
+ getEmailSettings()....

E10 Shipper Module




View Layer
Control Layer
Service Layer
Repository Layer
Common Entity
Orders list for Shipper.html
Order_shipper.js
OrderController
- orderService: OrderService
- settingService: SettingService
+ listFirstPage( )
+ listByPage( )
+ viewOrderDetails( )
+ listByPage( )
+ viewOrderDetails( )
+ loadCurrentSettings( )
OrderRestController
- orderService
+ updateOrderStatus( )
OrderService
- repo: OrderRepository
+ listByPage( )
+ get(Order)
+ delete( )
+ Create Order( )
<<Interface>>
OrderRepository
+ findAll(keyword, pageable)
Order
- id:String
- first_name:String
- last_name:String
.....
+ customer Customer
+ orderDetails: Set<OrderDetails>
+ paymentMethod: paymentMethod
+ status: OrderStatus
[Enum] PaymentMethod
- CASH ON DELIVERY
- PAYPAL
- CREDIT CARD
[Enum] OrderStatus
- PICKED
- SHIPPING
- DELIVERD
- RETURNED

F: Validation and Testing Record

Figure 4.6

Test Case ID	Test Description	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
B-1-1	Verify Login with Valid E-mail and Password	1.Enter valid E-mail and Password.2.Click "login" button.	E-mail:cxj164@bham.ac.uk Password:123456789	User can login and enter the system.	Login succesful	Pass
B-1-2	Verify Login with Invalid E-mail and Password	1.Enter invalid E-mail and Password.2.Click"login" button.	E-mail:cxj164@gmail.com Password:123456789	User can not login.	Login rejected	Pass
B-2-1	View Control Panel Module Information	1.Click every navigation bar menue items		The system should display correct item lists	The system displayed correct lists of items.	Pass
B-2-2	Add new items to each module	1.Click the"Add New" Button		The system should show correct blank froms.	The system showed correct blank froms.	Pass
B-2-3	Export CSV/EXCEL/PDF	1.Click the export icons under the menu bar.		Files should be auto downloaded.	Files were auto downloaded.	Pass
B-2-4	Upload images for users, products. brands, categories. Site logo	1.Click "choose file"or "upload image" button.2.Click "save" button.		The images should be displayafter uploaded. By saving and returning to the list form, old images should be replaced by the new one.	New images were displayed in the frontend.	Pass
B-2-5	Update products description.	1.Select a product and click "edit" button. 2. Modify the description in the rich text tool box. 3. click "save".		Product description should be changed.	Product description changed.	Pass
B-2-6	Search a exist product	1.Type a procuts name in the filter text area. 2.Click "search" button.	Type "Camera"	Products should be displayed if the name including the key word.	All camera related products were displayed.	Pass
B-2-7	Search a non-exist product	1.Type a procuts name in the filter text area. 2.Click "search" button.	Type "Sands"	The web page should indicate no such product.	A notification of "no such product" displayed.	Pass
B-3-1	Load country List	1.Click"Refresh Countries"		Country list should be displayed.	Country list displayed.	Pass
B-3-2	Load State List	1.Click"Refresh Countries"		State list under a certain country should be displayed.	State list displayed.	Pass

Test Case ID	Test Description	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
F-1-1	Customer Register	1.Click Register. 2.Fill in the register form and submit.	E-mail: jiangchufeng1@gmail.com Password:123456789	Customer should receive an verification E-mail	An verification E-mail received.	Pass
F-1-2	Customer verification	1.Click the verify link in the verification E-mail		Webpage redirect to the website and a notification of successful registered will be displayed.	Verify link was working, and the website inditatied successfully registered.	Pass
F-2-1	Customer valid account login	1.Click "login" in the navigation bar. 2.Input E-mail and password in the login form. 3. Click "login" button.	E-mail: jiangchufeng1@gmail.com Password:123456789	Cutomer login successful	Login succesful	Pass
F-2-2	Customer invalid account login	1.Click "login" in the navigation bar. 2.Input E-mail and password in the login form. 3. Click "login" button.	E-mail: jiangchu123@gmail.com Password:123456789	Cutomer login reject	Login rejected	Pass
F-2-3	Cutomer login using Google account	1.Click "Login" in the navigation bar. 2.Click "Google Login".	Personal Google Account	Google login window should pop up. After input google E-mail and password, the webpage should redirected to the shopping website.	Login succesful	Pass
F-2-4	Cutomer login using Facebook account	1.Click "Login" in the navigation bar. 2.Click " Facebook Login".	Personal Facebook Account	Facebook login window should pop up. After input Facebook E-mail or phone number and password, the webpage should redirected to the shopping website.	Login succesful	Pass

Test Case ID	Test Description	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
F-3-1	Add products to the shopping cart	1. Click "Add to Cart" button in the product detail display webpage	Add Canon Camera to the Shopping Cart	Products should be displayed in the Shopping Cart.	Canon Camera was in the Shopping Cart	Pass
F-3-2	Checkout using Cash On delivery	1.Click "Checkout". 2. Choose "COD" in the payment method column.		The webpage should indicate the order has been placed. An order confirmation should be sent to the customer's account.	The webpage showed the order has been placed. A order confirmation received in the E-mail inbox.	Pass
F-3-3	Checkout using PayPal	1.Click "Checkout". 2. Choose "PayPal" in the payment method column.	Personal Paypal Account	PayPal login window should pop up. After make Payment, the webpage should indicate the order has been placed, and an order confirmation should be sent to the customer's account.	PayPal Login window pop up. However, customer cannot login due to unknow reason.	Fail
F-3-4	Checkout using Credit Card	1.Click "Checkout". 2. Choose "Credit Card" in the payment method column.	Personal Credit Card Information	Credit Card information form window should be diaplayed. After complete that form and submit, the webpage should indicate the order has been placed and an order confirmation should be sent to the customer's account.	Credit Card information form displayed. After submit the credit card information, a notification of "order has been placed" came out. The order confirmation has been received.	Pass