HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

Version 1.3

AIMS - AN ONLINE MEDIA STORE
ITSS SOFTWARE DEVELOPMENT

Group 10

Nguyễn Trọng Huy – 20210451

Dương Văn Hữu – 20215210

Nguyễn Chấn Hưng – 20215209

Đặng Việt Hoàng – 20215206

Đặng Ngọc Huy - 20200270

Hanoi, June 2024

Table of Contents

List of Figures

List of Tables

1. Introduction

The Software Design Document (SDD) serves as a comprehensive guide for the software development process, providing detailed descriptions of the architecture, components, interfaces, and data for a software system. This document outlines the design decisions and approaches used to achieve the project requirements, ensuring that all stakeholders have a clear understanding of the system's structure and functionality. The SDD encompasses various subsections that detail the overall system architecture, module design, data design, and interface design, among others. By offering a thorough overview, the SDD aims to facilitate communication among team members, enhance system maintainability, and support future development and enhancements.

1.1. Objective

The purpose of this Software Design Document (SDD) is to provide a detailed blueprint of the software architecture, design decisions, and implementation plans for the development team. It serves as a crucial reference that ensures consistency, clarity, and alignment throughout the software development lifecycle. The intended audience for this SDD includes software developers, system architects, project managers, quality assurance testers, and other stakeholders involved in the project. By clearly documenting the design specifications, this SDD aims to facilitate effective communication, streamline the development process, and ensure that the final software product meets the specified requirements and standards.

1.2. Scope

This AIMS – ‘An Internet Media Store’ software is developed to be a desktop platform e-commerce software, which helps users to order media products on the Internet, and the store managers, at the same time, are easier to manage their store as well as the orders.

This program is capable of catering to 1,000 clients concurrently with minimal impact on performance and can run uninterrupted for 300 hours without any issues. Moreover, it can return to regular functioning within a maximum of 1 hour following an incident. The software's response time ranges from 2 seconds under typical circumstances to 5 seconds during periods of peak activity.

In AIMS, customers can not only search for products, but also sort products as they desire, they can place an order or rush order for necessary cases. AIMS is supported for VNPay transactions; thus, customers can easily pay for their order. Moreover, customers can review their order and modify any information during the processing order stage. While shopkeepers can manage their store by managing products directly in the system. They,

meanwhile, can process the orders of the customers. For administrators, they are capable of managing users and privilege problems of users.

Besides, for a desktop website, the need for a graphical user interface is also under consideration, which can meet the requirements of end users and enhance the experience of users. Throughout the development stage, every document is also recorded for the future maintenance and upgrading. We keep our focus on every stage to supervise the timeline of the client provided and the quality the software may deliver. If any change is made, our team will adapt quickly to revise our work.

1.3. Glossary

Table 1 Glossary

No	Term	Explanation	Example	Note
1	session	A session is a temporary period of interaction or engagement between a user and a system, during which the user accesses and interacts with software or a website, and the system maintains relevant settings and information.	software session	

2	VAT (Value-added tax)	It is a type of consumption tax that is levied on the value added to goods and services at each stage of production or distribution. VAT is typically implemented as a percentage of the final selling price of a product or service, and it is collected by businesses on behalf of the government.		
3	API (Application Programming Protocol)	API is a set of rules, protocols, and tools that allows different software apps to communicate with each other.	VNPay API	AIMS connects to the API of VNPay for transactions.
4	Payment gateway	A technology service that facilitates the secure transmission of payment information between a merchant's website or application and the financial institutions involved in		

		processing payment transactions.		
5	GUI (Graphical user interface)	Refers to the visual elements and interactive components of a software application that allow users to interact with the system using graphical icons, buttons, menus, and windows.		
6	Credit card	A credit card is a payment card issued by a financial institution, such as a bank or credit union, that allows cardholders to borrow funds up to a predetermined limit to make purchases or pay for goods and services.		AIMS currently supports for credit card payment through VNPay
7	Authentication	Authentication is the process of verifying the identity of a user, device, or system		

		attempting to access a resource or service.		
8	Response time	Response time refers to the amount of time it takes for a system to respond to a user's request or input.		

1.4. References

[1] Centers for Medicare & Medicaid Services, "System Design Document Template," [Online]. Available: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SystemDesignDocument.docx.

2. Overall Description

In this section, we deliver the main points in the SDD document, which include the overview and technical-view assumptions, constraints and risks for AIMS - An Online Media Store project.

2.1. General Overview

The system context and basic design approach focus on creating a robust, scalable, and maintainable software solution. The system architecture follows a modular design, allowing for easy updates and integration with other systems. The software architecture employs a layered approach, separating concerns and ensuring clear interfaces between components. Design goals include high performance, security, and user-friendly interfaces. High-level context diagrams, such as general use case diagrams and activity diagrams, provide an overview of system interactions and workflows. Any updates to these diagrams reflect recent changes in requirements or understanding, ensuring the design remains aligned with current project needs.

2.2. Assumptions/Constraints/Risks

2.2.1. Assumptions

The design of the system is based on several key assumptions. It is assumed that the system will operate within a stable hardware environment, utilizing specific operating systems as outlined in the project requirements. The end-users are expected to have a basic level of technical proficiency, and any future changes in functionality will be minimal and manageable. Additionally, the system relies on the integration with related software components, which are assumed to be available and compatible as specified.

2.2.2. Constraints

Several global constraints impact the design of the system. The hardware environment imposes limitations on processing power and memory, requiring optimization of software performance. The end-user environment demands a user-friendly interface accessible on various devices. Resource availability and potential volatility necessitate a design that can handle fluctuations efficiently. Compliance with industry standards and interoperability with existing systems are mandatory, influencing interface and protocol design. Licensing requirements must be adhered to, affecting software choices and costs. The data repository needs to ensure secure and efficient storage and distribution of information. Security requirements, including data protection and user privacy, are paramount. Performance requirements dictate the need for fast and reliable system

responses. Network communications must support seamless data exchange, and rigorous verification and validation processes are necessary to maintain high quality and reliability. All these constraints collectively shape the system's architecture and implementation.

2.2.3. Risks

The system design is subject to various risks, including potential security vulnerabilities, resource limitations, and integration challenges. Security risks involve unauthorized access and data breaches, mitigated by implementing robust encryption and authentication mechanisms. Resource limitations, such as processing power and memory, are addressed through efficient coding practices and performance optimization. Integration challenges with existing systems and third-party components are managed by thorough testing and adherence to interoperability standards. Additionally, potential changes in user requirements or operating environments pose risks, which are mitigated by maintaining a flexible and modular design that can adapt to evolving needs. Regular risk assessments and mitigation strategies ensure that potential issues are identified and addressed proactively.

3. System Architecture and Architecture Design

In this section, we propose the general overview of System Architecture Design. This starts from defining the Architectural Patterns, then we design the interaction diagram using sequence diagrams for the proposed architectural patterns. From the interaction diagrams, we can have the analysis class diagrams, which is the fundamental steps for the detailed-level class diagrams for implementation.

3.1. Architectural Patterns

The Entity-Control-Boundary (ECB) architecture design pattern is a software design approach that helps in organizing and structuring code to enhance maintainability, scalability, and clarity. This pattern divides the system into three primary types of components: entities, controls, and boundaries.

Entities represent the core business logic and data of the application. These components encapsulate the main data structures and the operations directly related to the core functionality of the system. Entities are typically designed to be reusable and independent of the specific use cases. They model real-world objects and are responsible for managing and maintaining their state and behavior. For example, in a banking application, entities might include classes such as *Account*, *Customer*, and *Transaction*.

Controls are responsible for coordinating the flow of information between entities and boundaries. They manage the application's use cases, orchestrating the interactions between entities and ensuring the correct sequence of operations. Controls contain the logic that drives the application's processes, ensuring that user inputs are properly processed and that the necessary business rules are applied. For instance, a *PlaceOrder* control might handle the logic for executing a financial transaction, validating inputs, and updating account balances accordingly.

Boundaries define the interfaces through which the system interacts with external actors, such as users, other systems, or external services. They serve as the point of communication between the application and its environment, encapsulating the specifics of input and output mechanisms. Boundaries translate user actions into requests that the control components can handle and present the results back to the user. Examples of boundaries include user interface components like forms and buttons, web service interfaces, or API endpoints, which serve as subsystems in interface design.

We select this architecture pattern because many advantages it delivers:

- **Separation of Concerns**: By clearly dividing the system into entities, controls, and boundaries, the ECB pattern ensures a clean separation of concerns, making the codebase more modular and easier to understand.

- **Maintainability**: Changes in one part of the system (e.g., user interface changes) do not directly impact the core business logic encapsulated in entities, thereby reducing the risk of introducing bugs when making modifications.

- **Scalability**: The modular nature of the ECB pattern facilitates scalability. New features can be added by creating new controls and boundaries without altering existing entities.

- **Testability**: The clear separation between entities, controls, and boundaries enhances testability. Business logic can be tested independently of the user interface, and mock boundaries can be used to simulate interactions during testing.

3.2. Interaction Diagrams

3.2.1 Place order

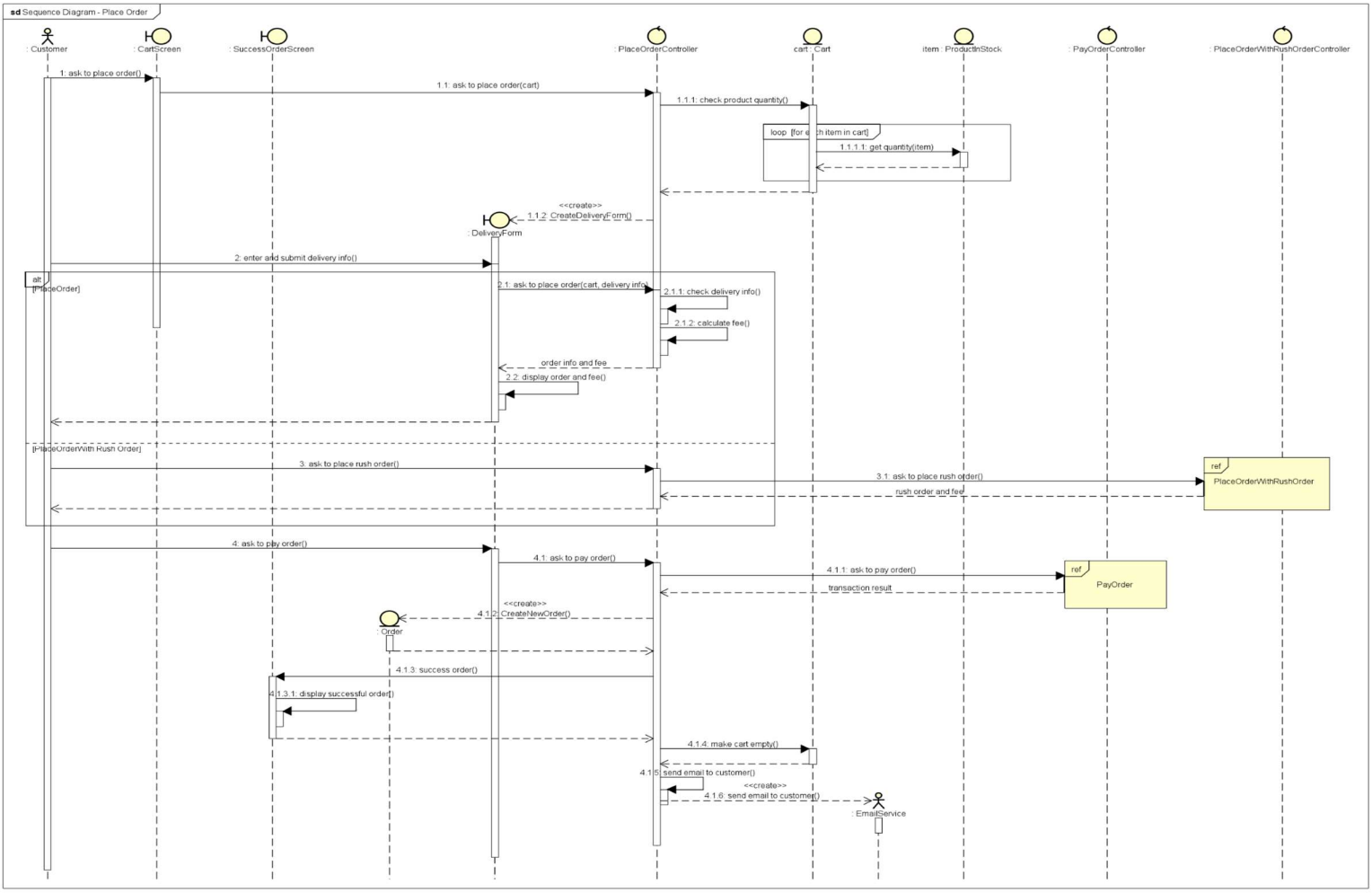


Figure 1 Place order sequence diagram

3.2.2 Place rush order

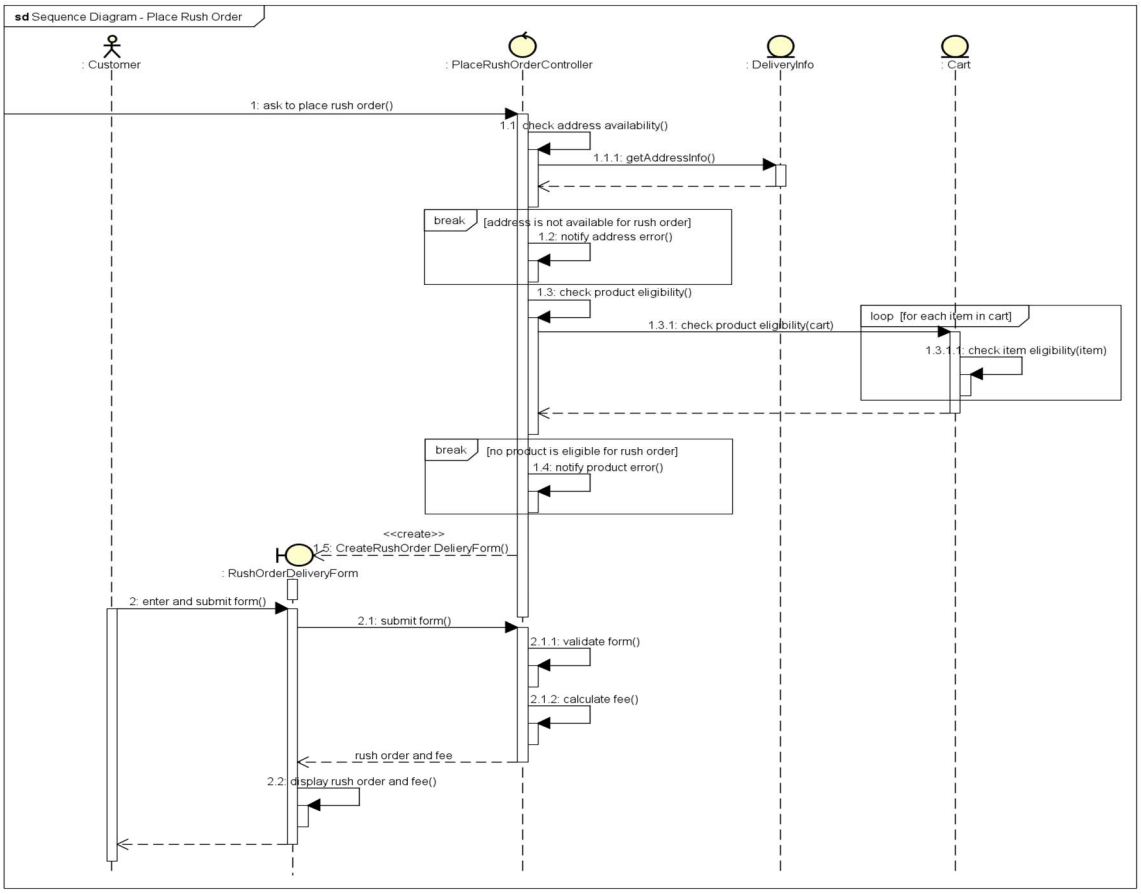


Figure 2 Place rush order sequence diagram

3.2.3 Pay order

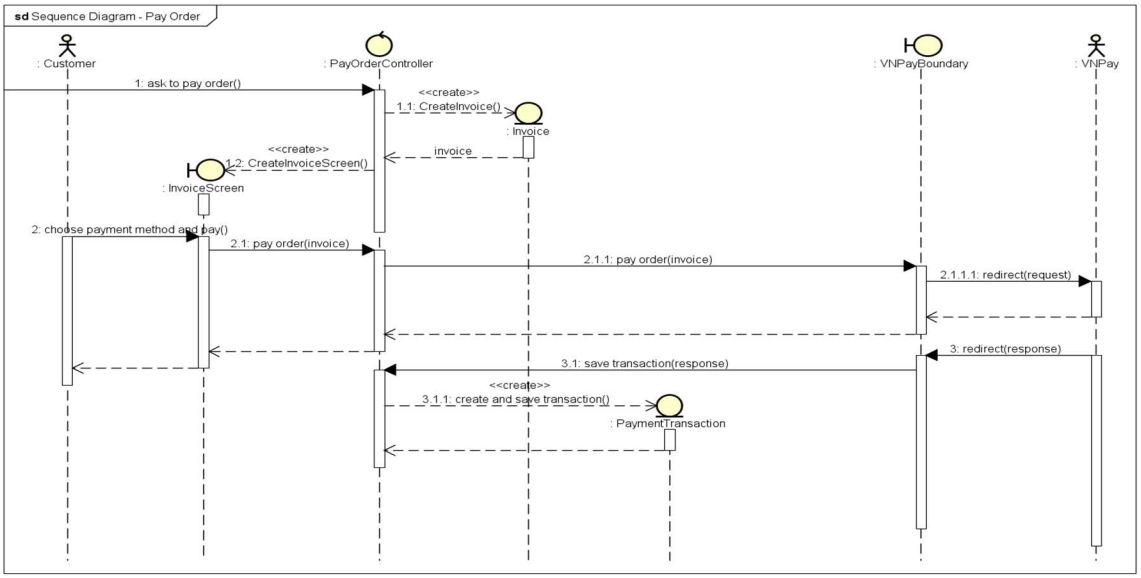


Figure 3 Pay order sequence diagram

3.2.4 Cancel order

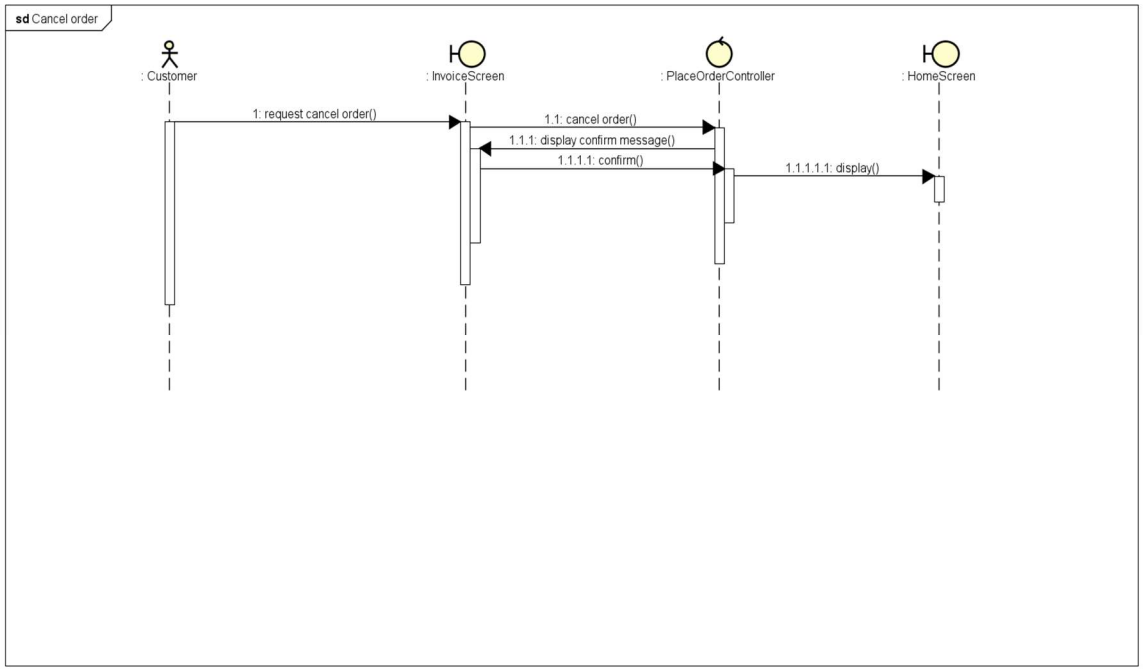


Figure 4 Cancel order sequence diagram

3.2.5 Review order

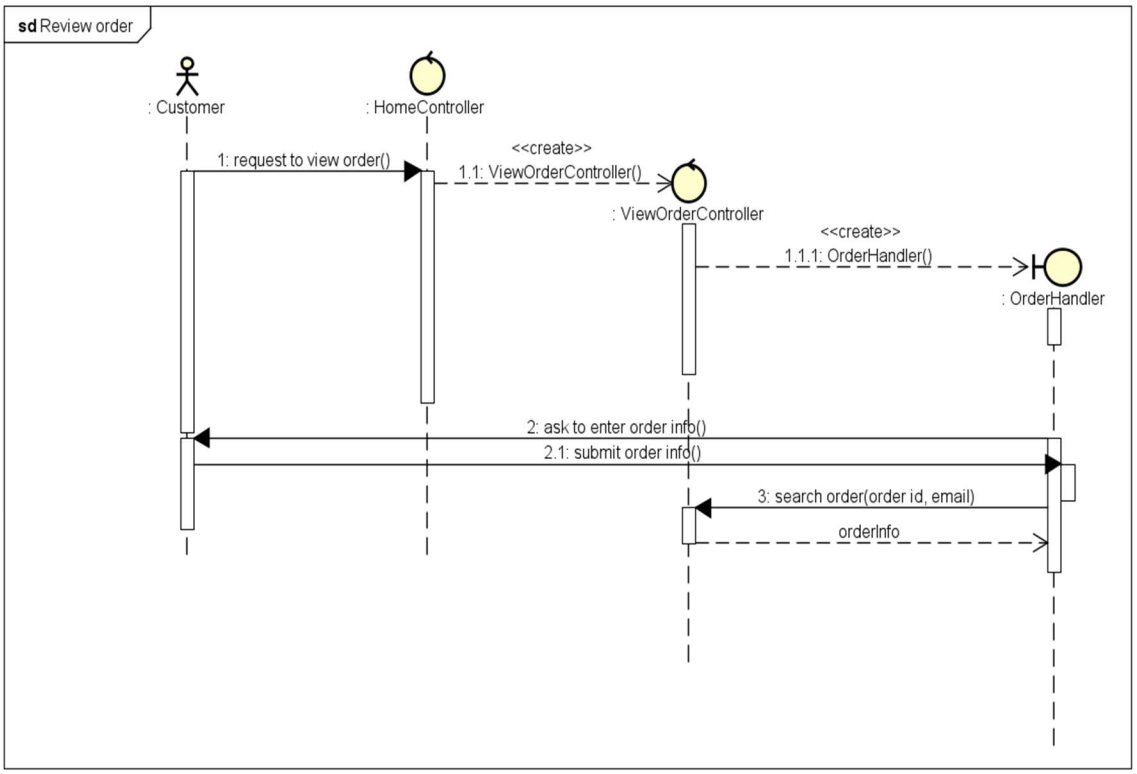


Figure 5 Review order sequence diagram

3.2.6 CRUD products in cart

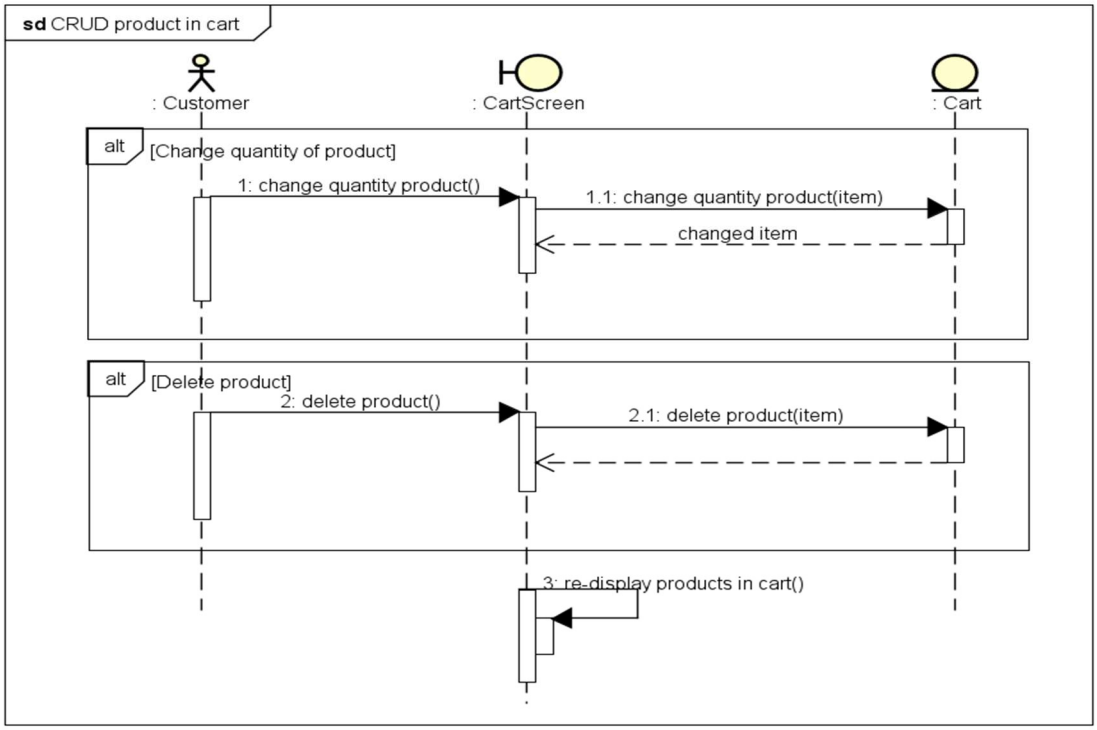


Figure 6 CRUD products in cart sequence diagram

3.2.7 Search for products

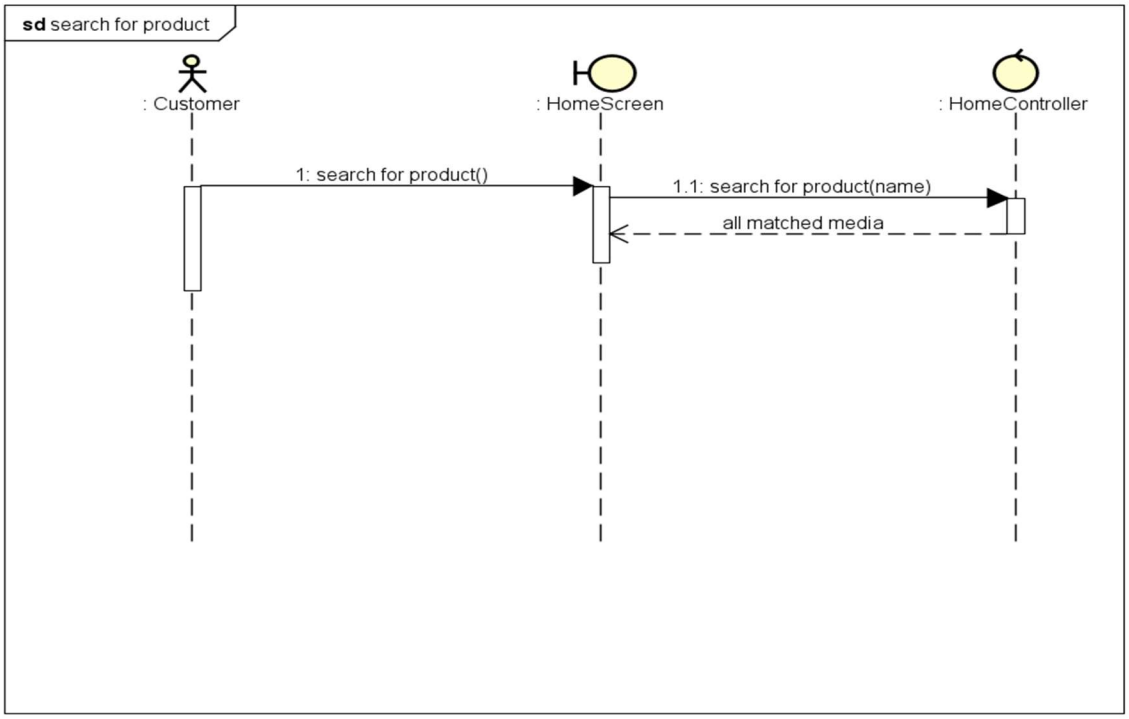


Figure 7 Search for products sequence diagram

3.2.8 Sort products by price

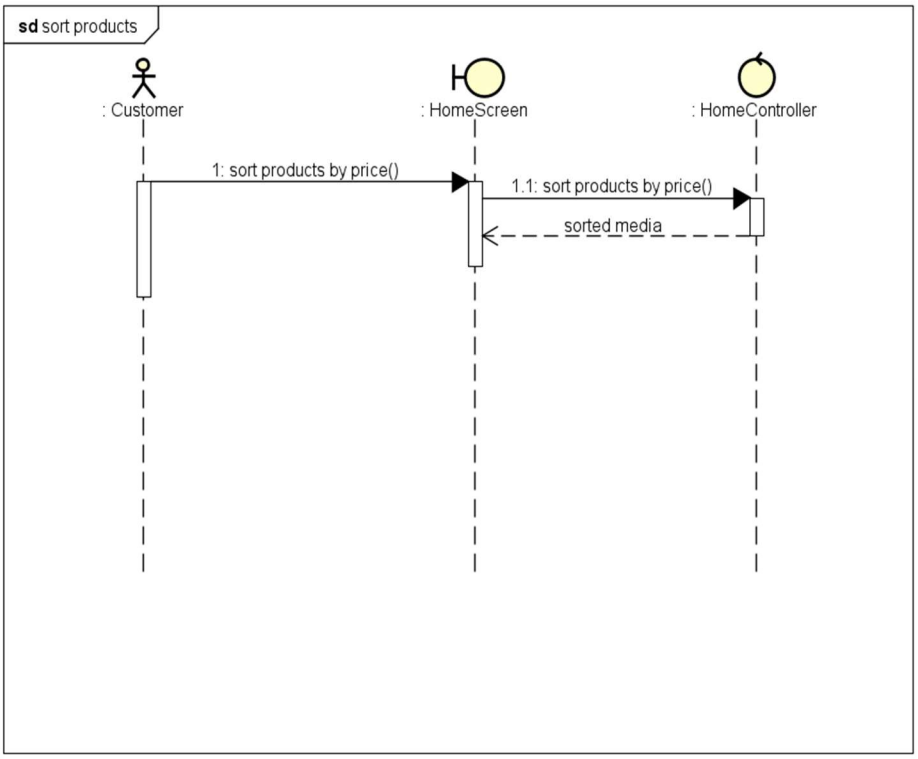


Figure 8 Sort products by price sequence diagram

3.2.9 Log in

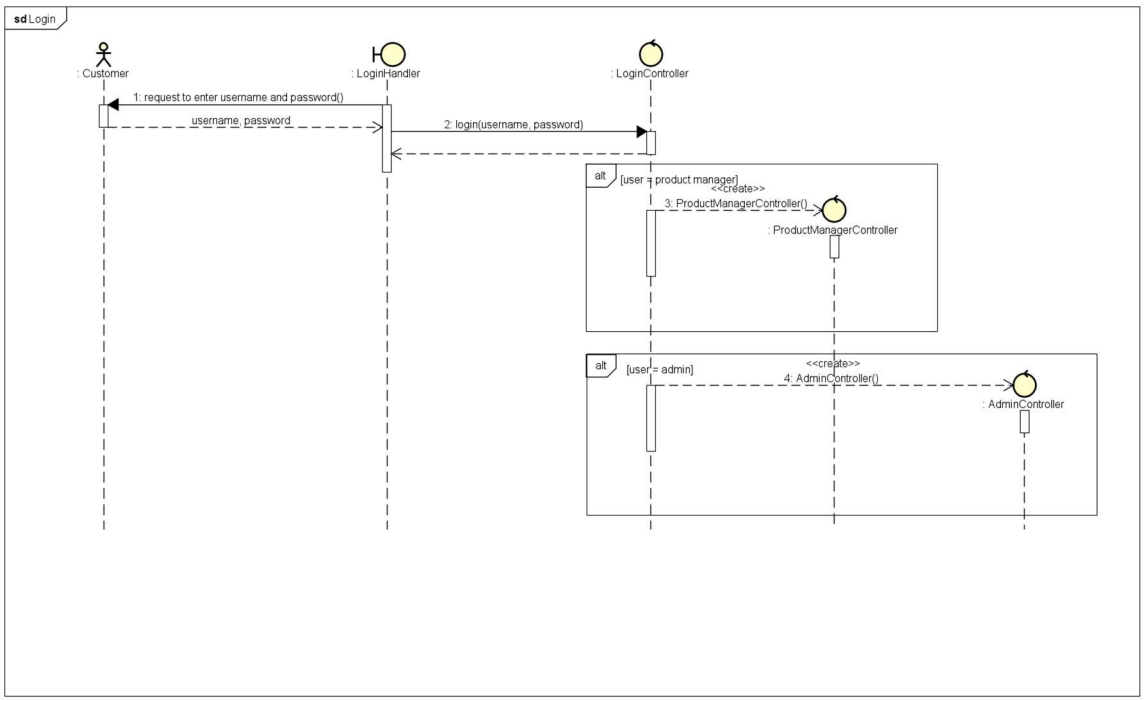


Figure 9 Login sequence diagram

3.2.10 CRUD products

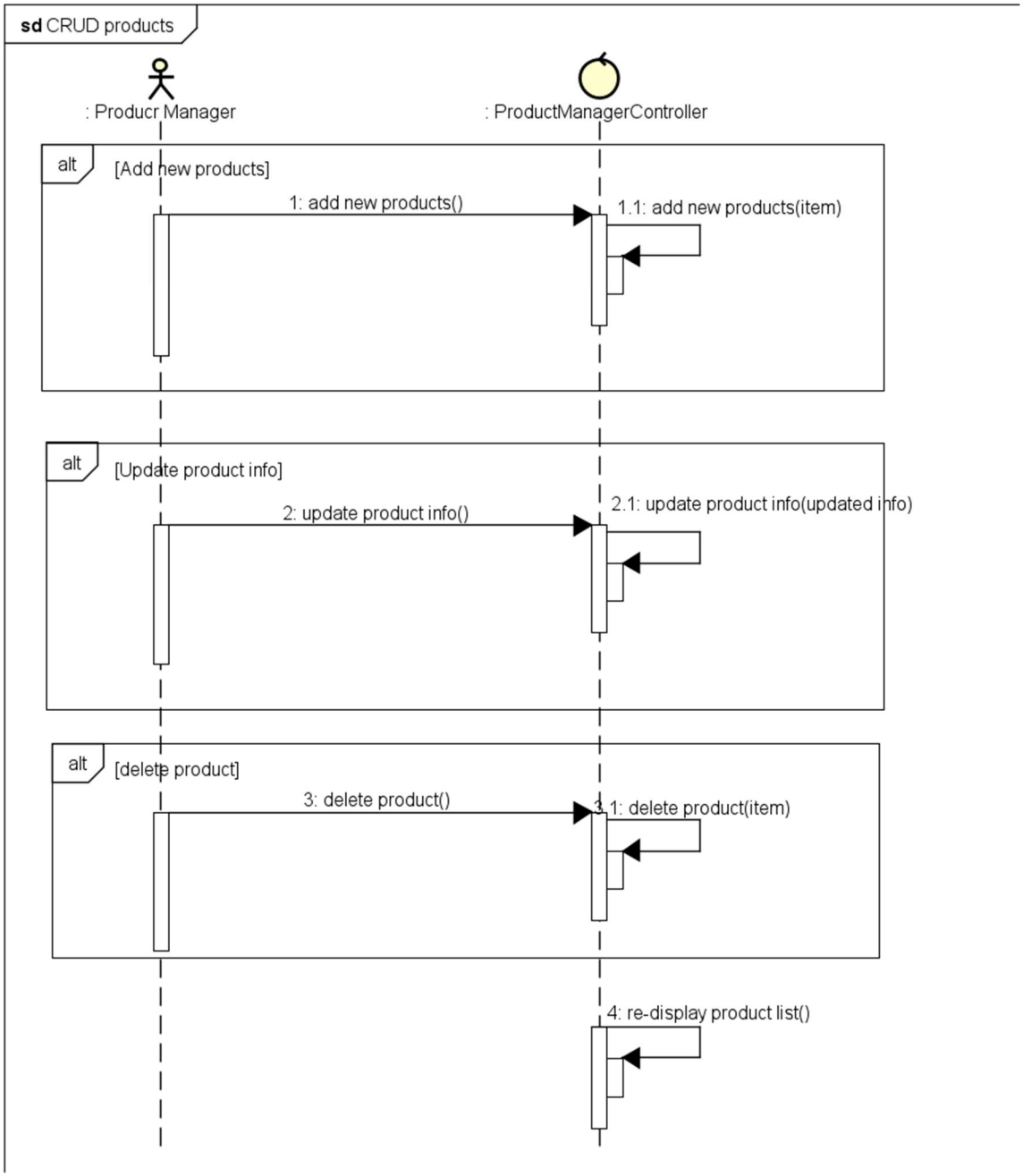


Figure 10 CRUD products sequence diagram

3.2.11 CRUD users

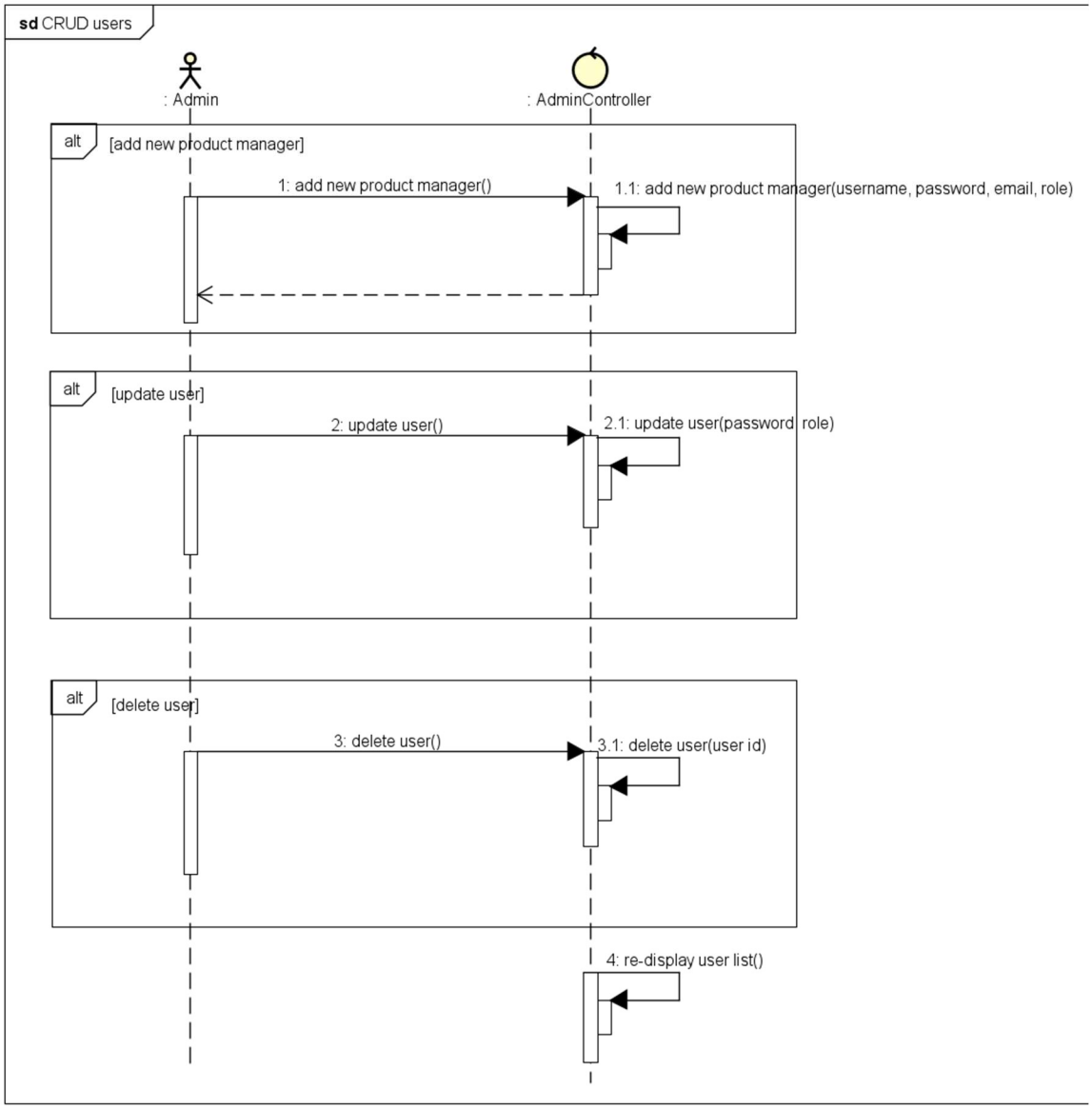


Figure 11 CRUD users sequence diagram

3.2.12 Change password

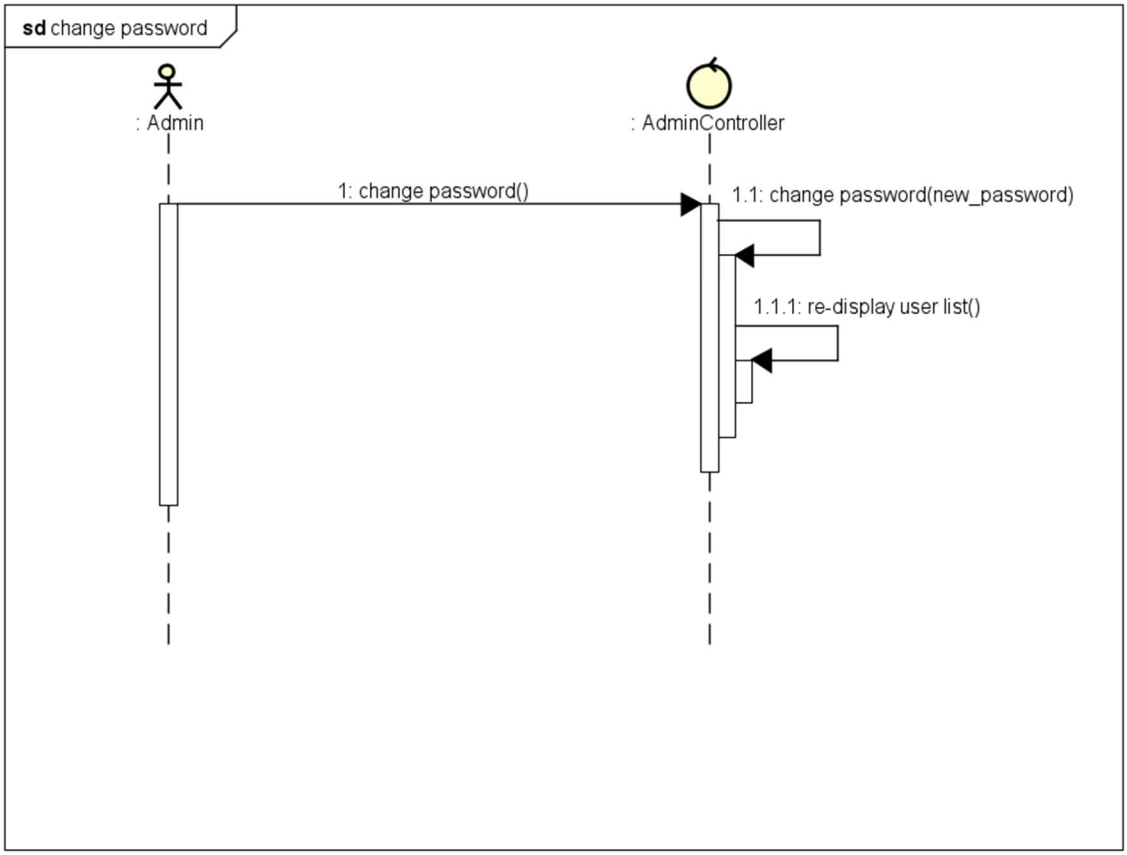


Figure 12 Change password sequence diagram

3.3. Analysis Class Diagrams

3.3.1 Package Entity

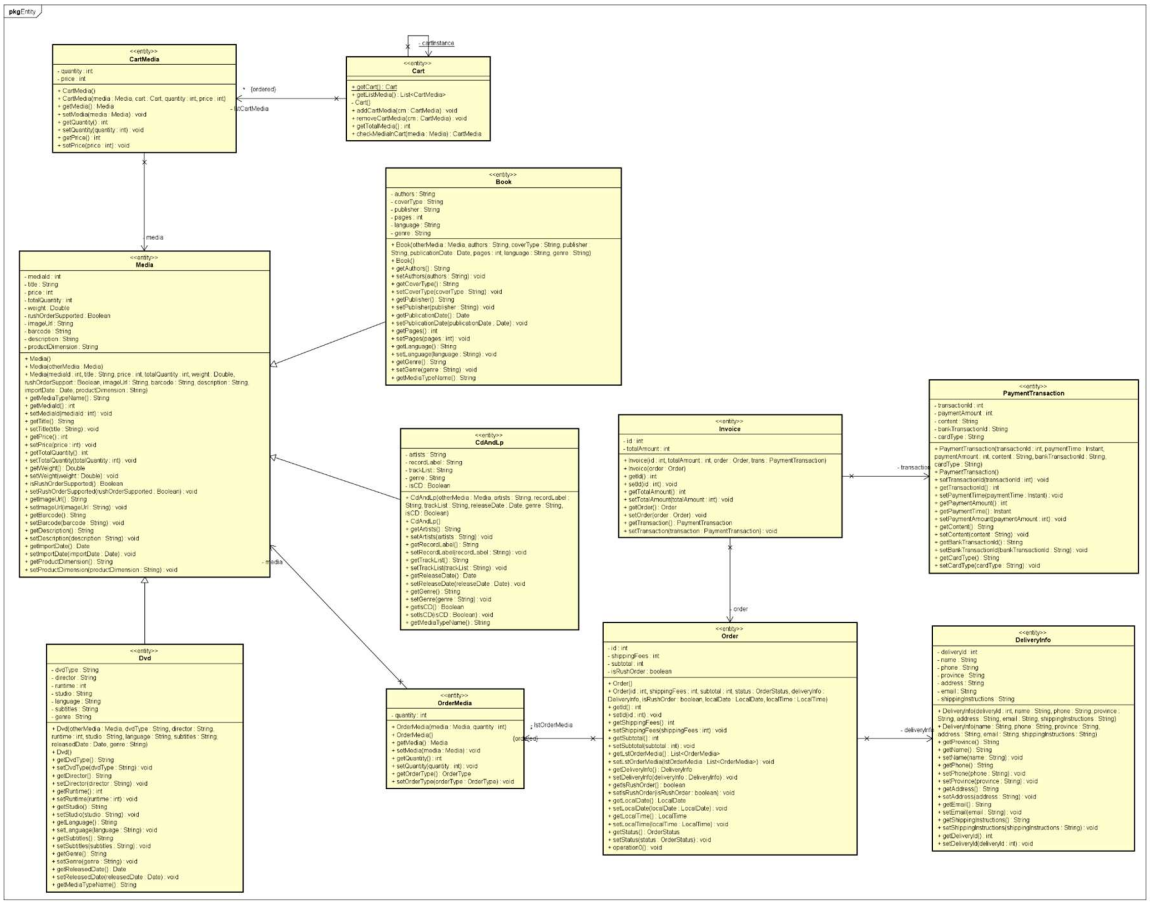


Figure 13 Analysis entity class diagram

3.3.2 Package Controller

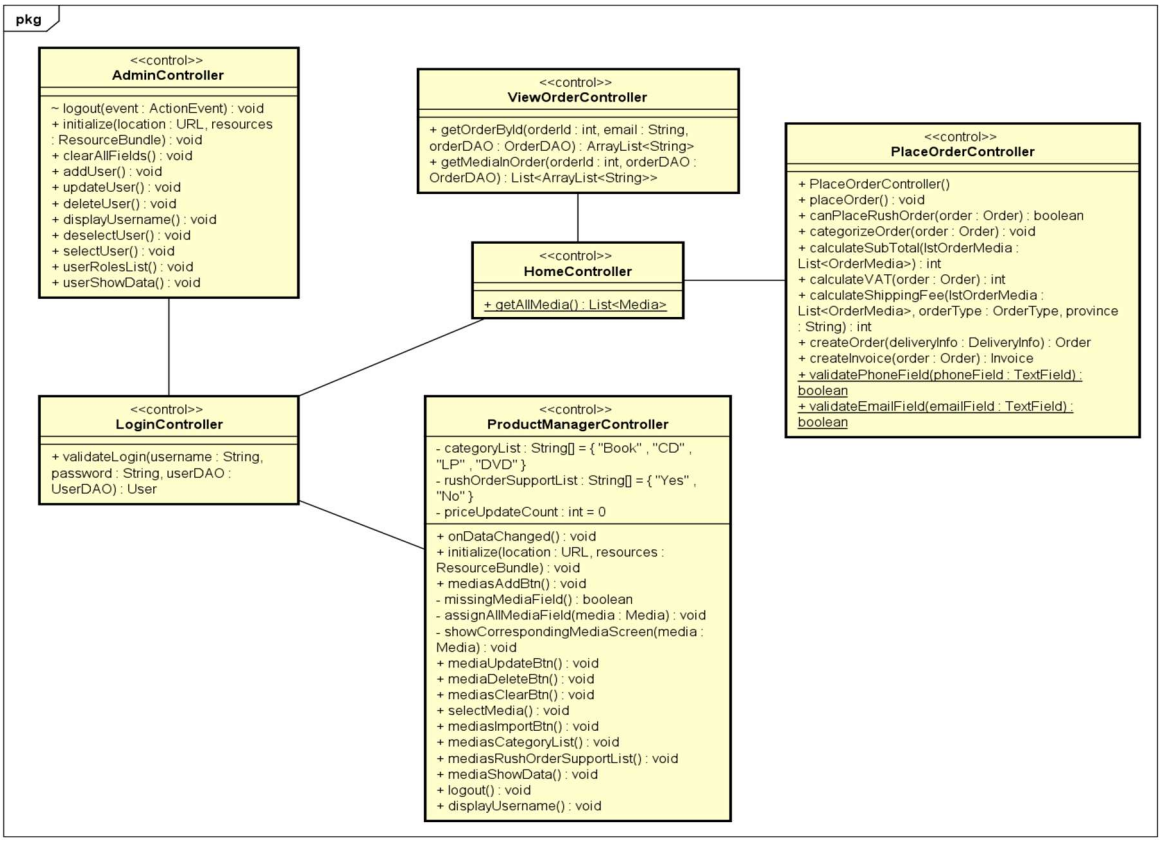


Figure 14 Analysis controller class diagram

3.3.3 Package Boundary

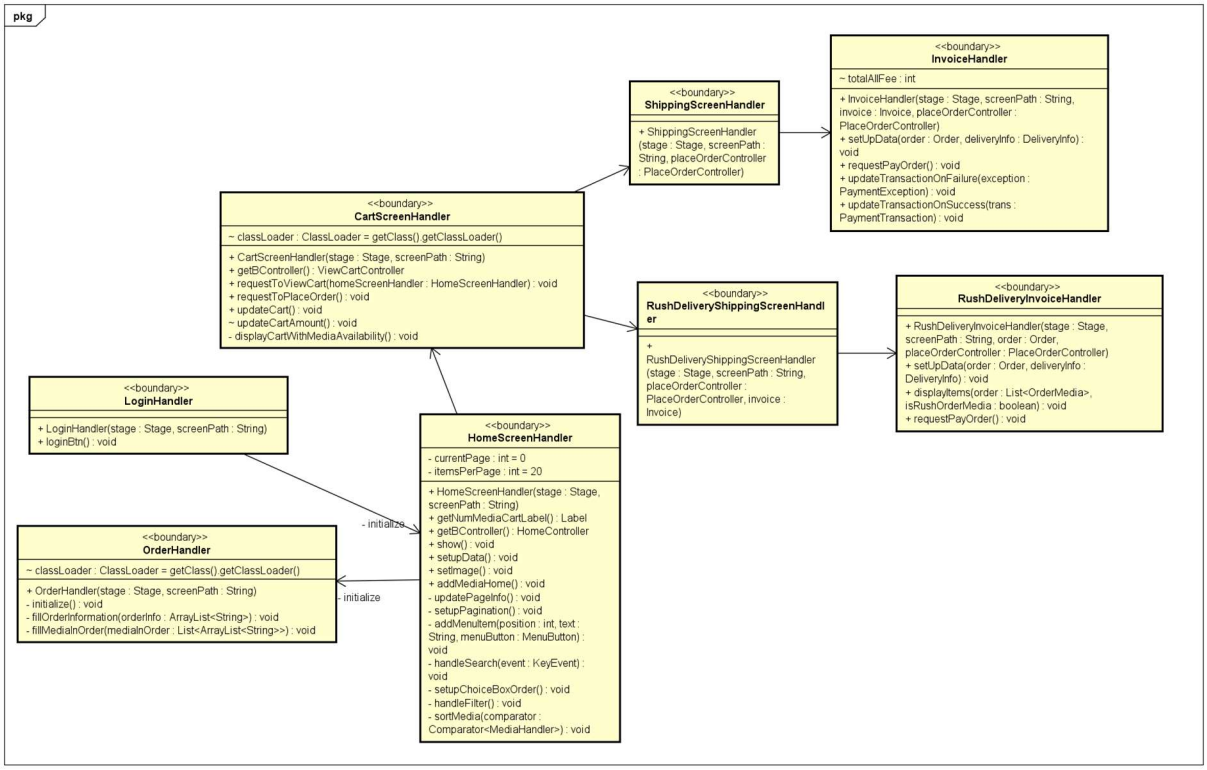


Figure 15 Analysis boundary class diagram

3.4. Unified Analysis Class Diagram

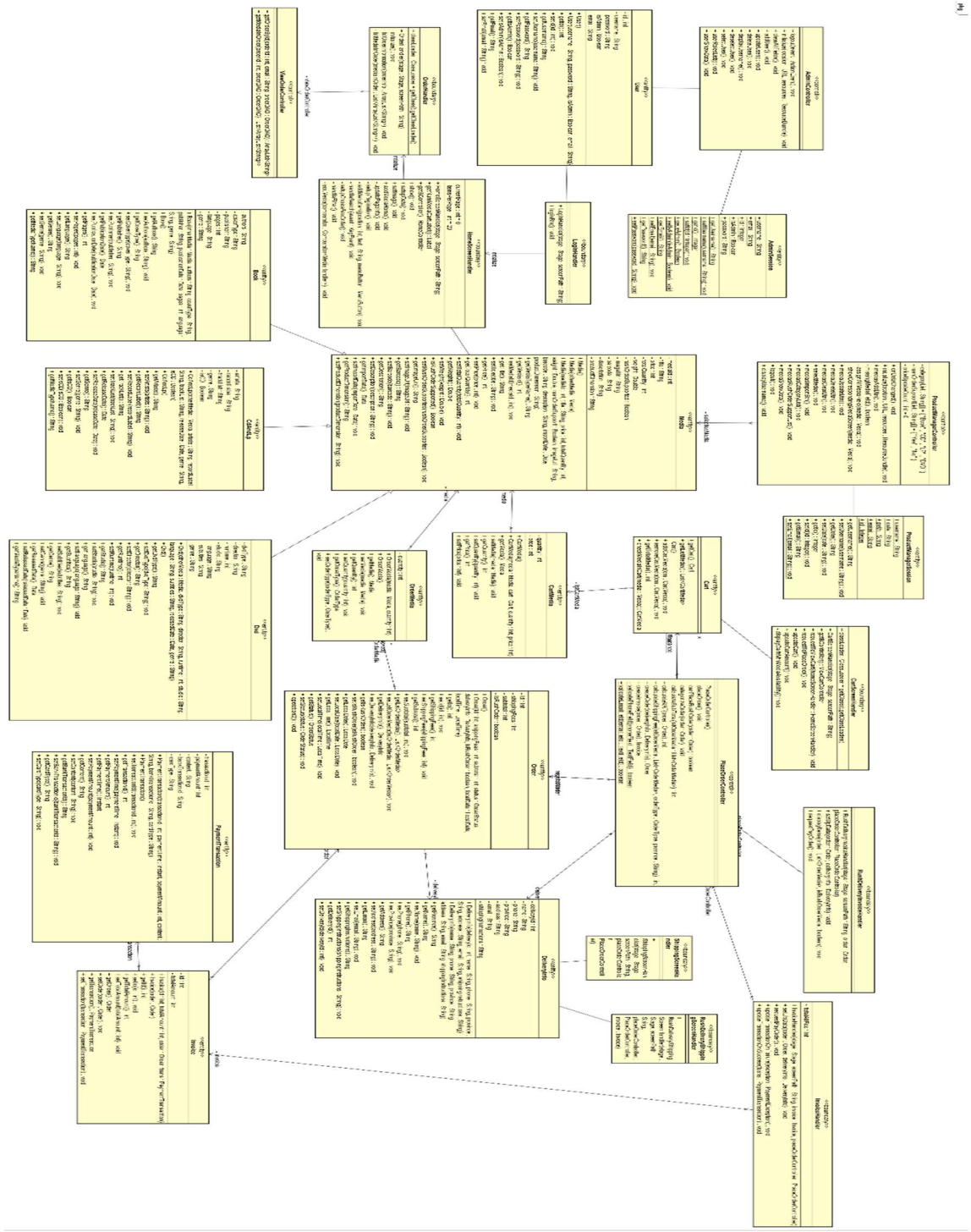


Figure 16 Unified analysis class diagram

4. Detailed Design

4.1. User Interface Design

4.1.1. Screen Configuration Standardization

1. Description

This document describes the configuration of the GUI of AIMS software for use case "Place Order" and all of the use cases related to it.

2. Display

Number of colors supported: 16,777,216 colors

Resolution: 1366 × 768 pixels

3. Screen

Location of standard buttons: At the bottom (vertically) and in the middle (horizontally) of the frame.

Location of the messages: Starting from the top vertically and in the middle horizontally of the frame down to the bottom.

Display of the screen title: The title is located at the top of the frame under the header.

Consistency in expression of alphanumeric numbers: comma for separator of thousand while strings only consist of characters, digits, commas, dots, spaces, underscores, and hyphen symbol.

Header: located right at the top (vertically) of the screen with the width is fit to the parent.

Footer: located right at the bottom (vertically) of the screen with the width is fit to the parent.

4. Control

Size of the text: medium size (at most 24px). Font: Arial. Color: #000000.

Background of standard button: background color: #c82905.

Input check process: Should check if it is empty or not. Next, check if the input is in the correct format or not.

Hover: hand sign.

Sequence of moving the focus: There will be no stack frames. Each screen will be separated. However, the manual is considered a popup message, as the main screen cannot

be operated while the manual screen is shown. After the opening screen, the app will start with a splash screen, and then the first screen (home screen) will appear.

5. Error

A message will be given to notify the users what is the problem with a pop-up.

6. Direct input from the keyboard

There will be no shortcuts. There are back buttons to move back to the previous screen. Also, there is the close button “X” located at the title bar to the right to close the screen.

4.1.2. Screen Transition Diagrams

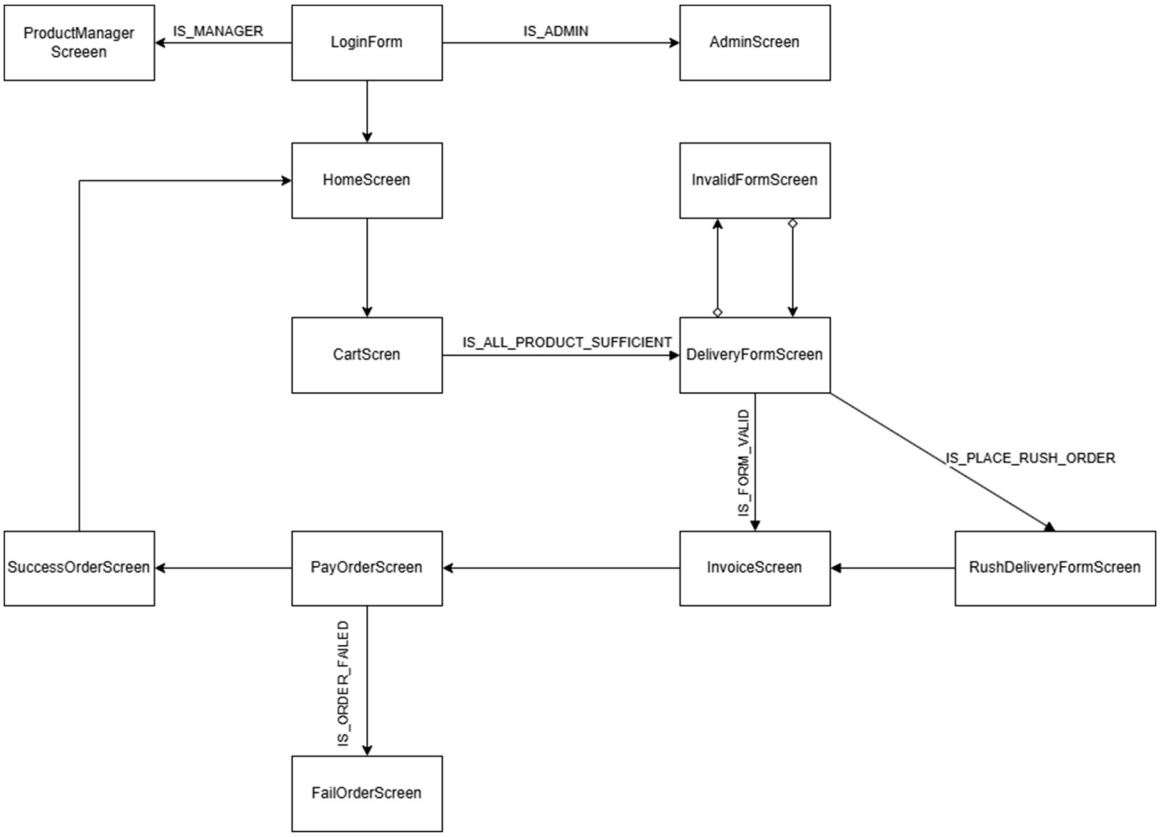


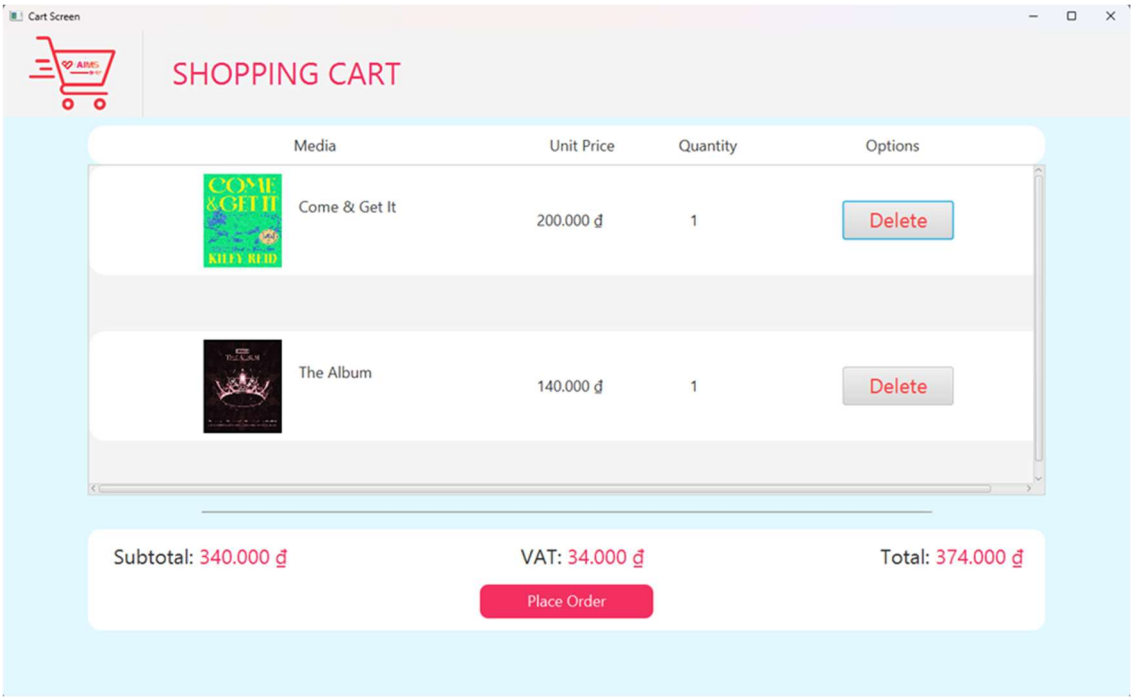
Figure 17 Screen transition diagram

4.1.3. Screen Specifications

Table 2 Cart Screen

AIMS SOFTWARE			
Screen specification		CartScreen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**

07/04/2024			Group 10

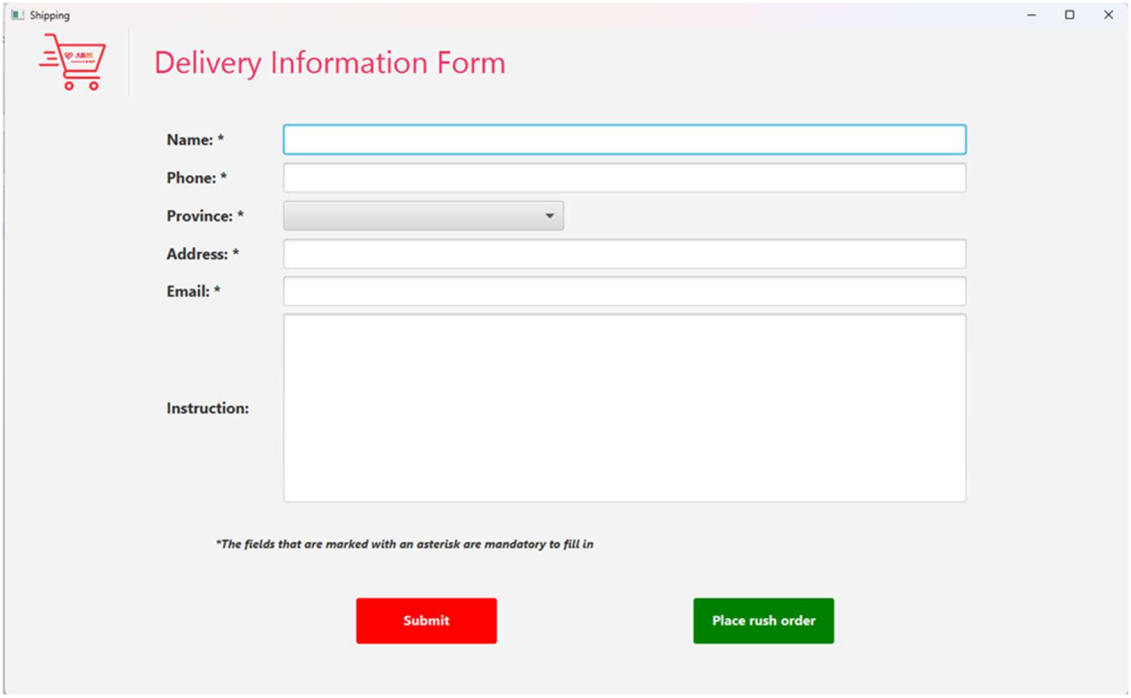


Control	Operation	Function
Area for display the list of products in the cart	Initialization	Display products in the cart.
Quantity spinner	Select	Display a product quantity in the cart.
Remove button	Click	Remove the product from the cart
Area for displaying subtotal and total	Initialization	Display the amount that has to paid
Button for place order	Click	Move to DeliveryFormScreen

Table 3 Delivery Form Screen

AIMS SOFTWARE			
Screen specification		DeliveryFormScreen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**

07/04/2024			Group 10

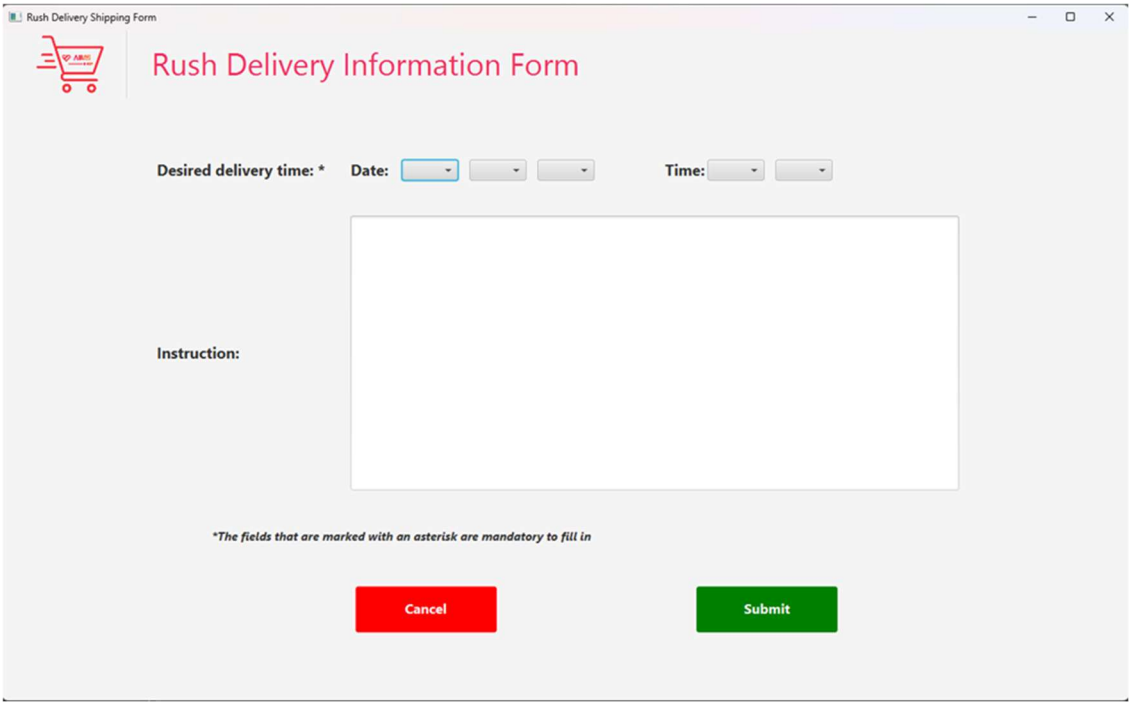


Control	Operation	Function
Area for delivery form display	Initialization	Display the form for delivery
“Full name” attribute	Input	Allow user to input name
“Phone number” attribute	Input	Allow user to input phone number
“Province” attribute	Select	Allow user to select province
“Address” attribute	Input	Allow user to input address
“Shipping instruction” attribute	Input	Allow user to input shipping instruction
“Continue” button	Click	Move to InvoiceScreen
“Place Rush Order” button	Click	Move to RushDeliveryForm

Table 4 Rush Delivery Form Screen

AIMS SOFTWARE	
Screen specification	RushDeliveryFormScreen

Date of creation	Approved by	Reviewed by	Person in charge
07/04/2024			Group 10

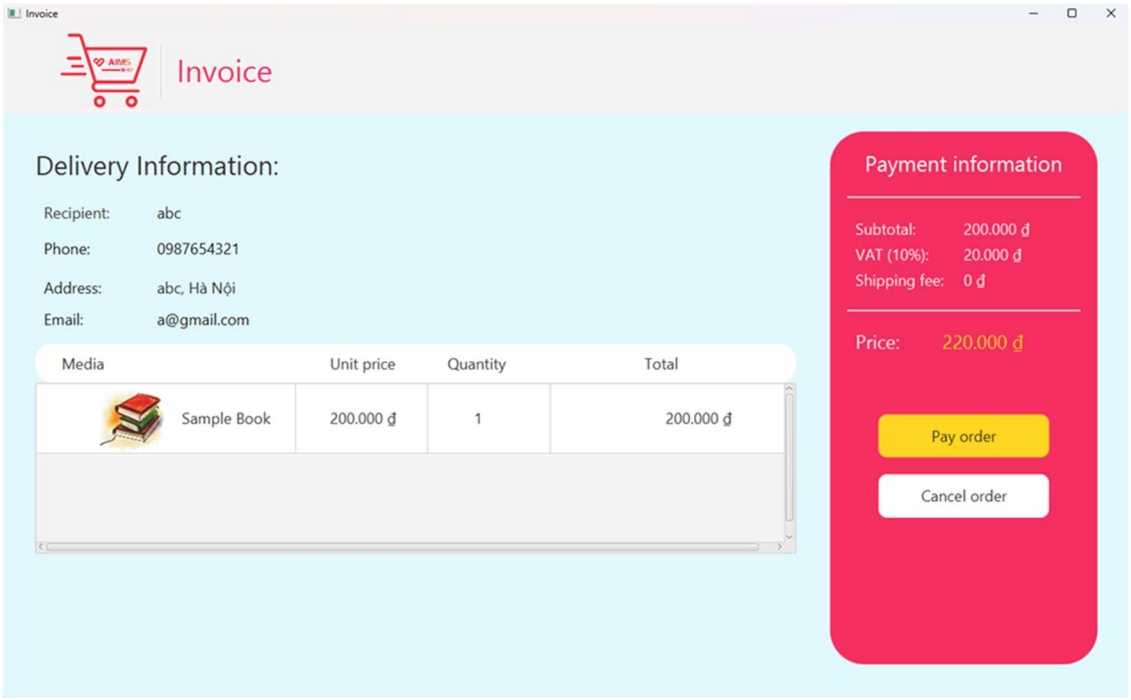


Control	Operation	Function
Area for rush delivery form display	Initialization	Display rush delivery form with one more attribute
“Rush delivery time”	Select	Select the date for rush time
“Continue”	Click	Move to InvoiceScreen

* The other control in this form is inherited from the previous DeliveryFormScreen.

Table 5 Invoice Screen

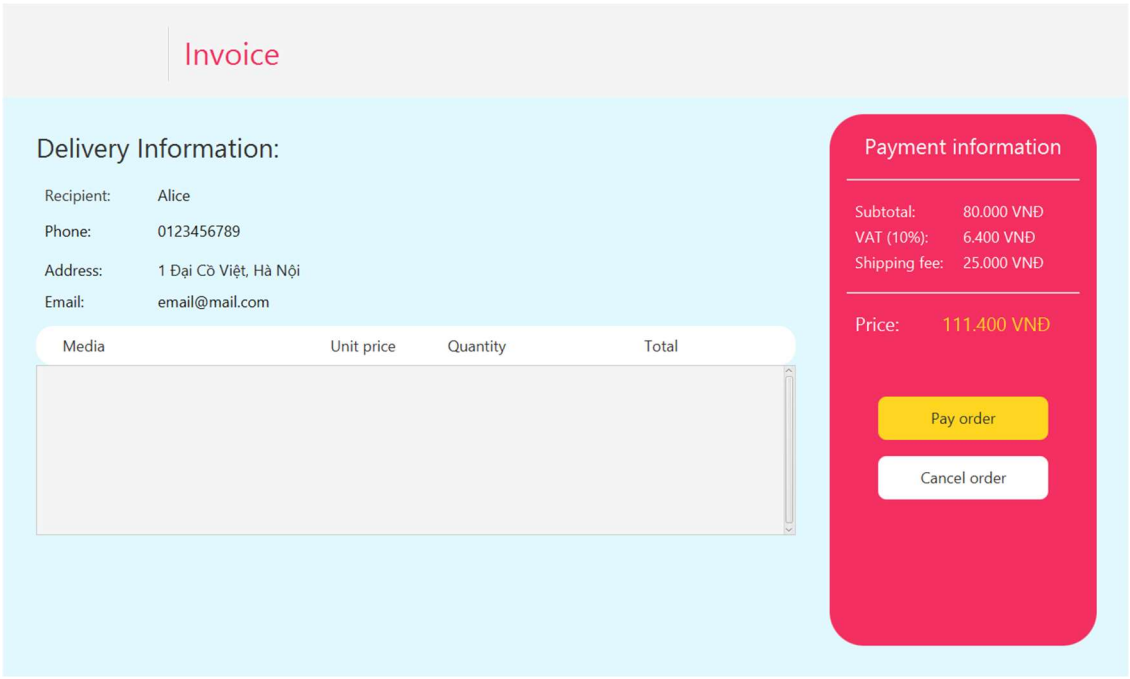
AIMS SOFTWARE			
Screen specification		InvoiceScreen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10



Control	Operation	Function
Area for list of products in the order	Initialization	Display the products in the order
Delivery Information	Initialization	Display the delivery form information from the customer
Area for total fee	Initialization	Display the amount, shipping fee and total must pay
“Pay the invoice” button	Click	Move to PayOrderScreen.

Table 6 Pay Order Form

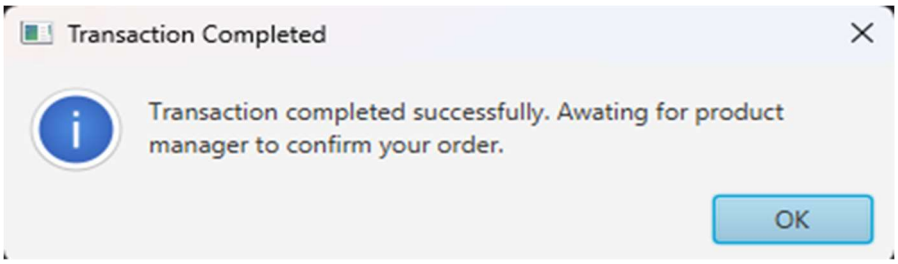
AIMS SOFTWARE			
Screen specification		PayOrderForm	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10



Control	Operation	Function
Area for total fee display	Initialization	Display amount the customer must pay.
"Cardholder name"	Input	Allow input name in the card.
"Card number"	Input	Allow the input number of the card.
"Expired date"	Input	Allow to input the expiration date in the card.
"Payment method"	Radio	Default credit card.
"Continue with VNPAY"	Click	Move to the boundary of VNPAY to process the transaction.

Table 7 Success Order Screen

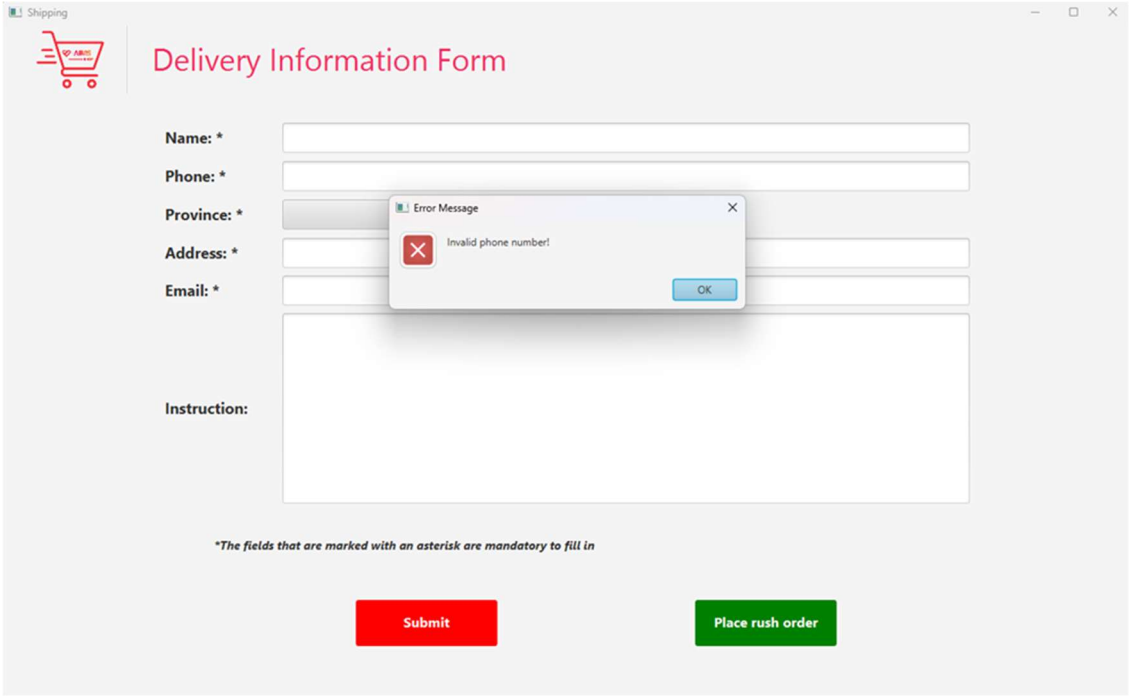
AIMS SOFTWARE			
Screen specification		SuccessOrderScreen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10



Control	Operation	Function
Area to display the status of order	Initialization	Display the status of the order and the information of the transaction.
“Ok” button	Click	Move to HomeScreen.

Table 8 Invalid Form Screen

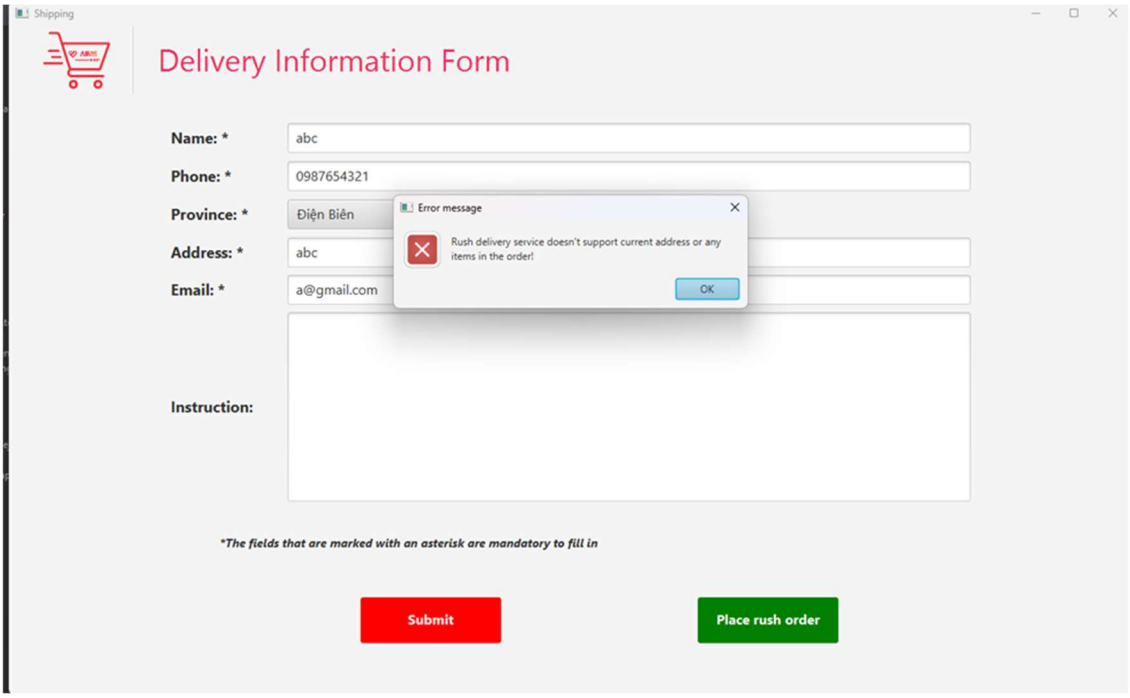
AIMS SOFTWARE			
Screen specification		InvalidFormScreen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10



Control	Operation	Function
Message	Initialization	Display the error content
“OK” button	Click	Move to the previous screen

Table 9 Invalid Rush Order Screen

AIMS SOFTWARE			
Screen specification		InvalidFormScreen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10



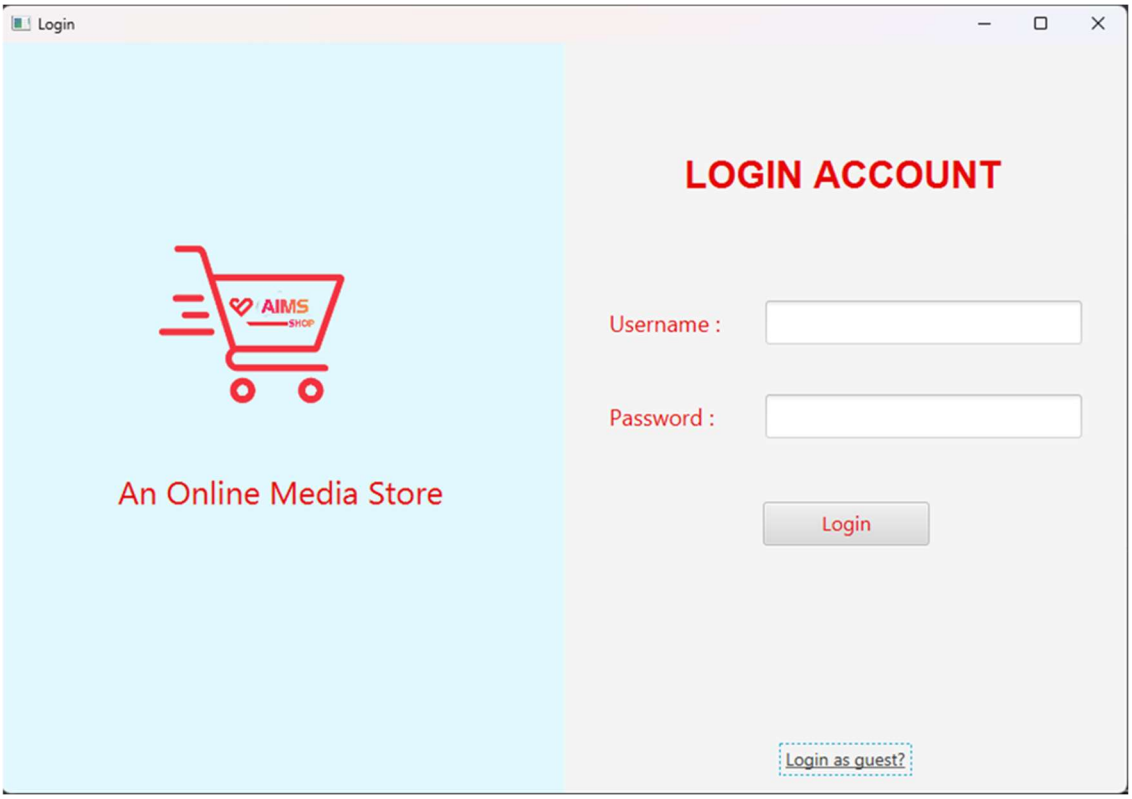
Control	Operation	Function
Message	Initialization	Display the error content
“OK” button	Click	Move to the previous screen

Table 10 Login Screen

AIMS SOFTWARE

Screen specification	LoginScreen

Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10

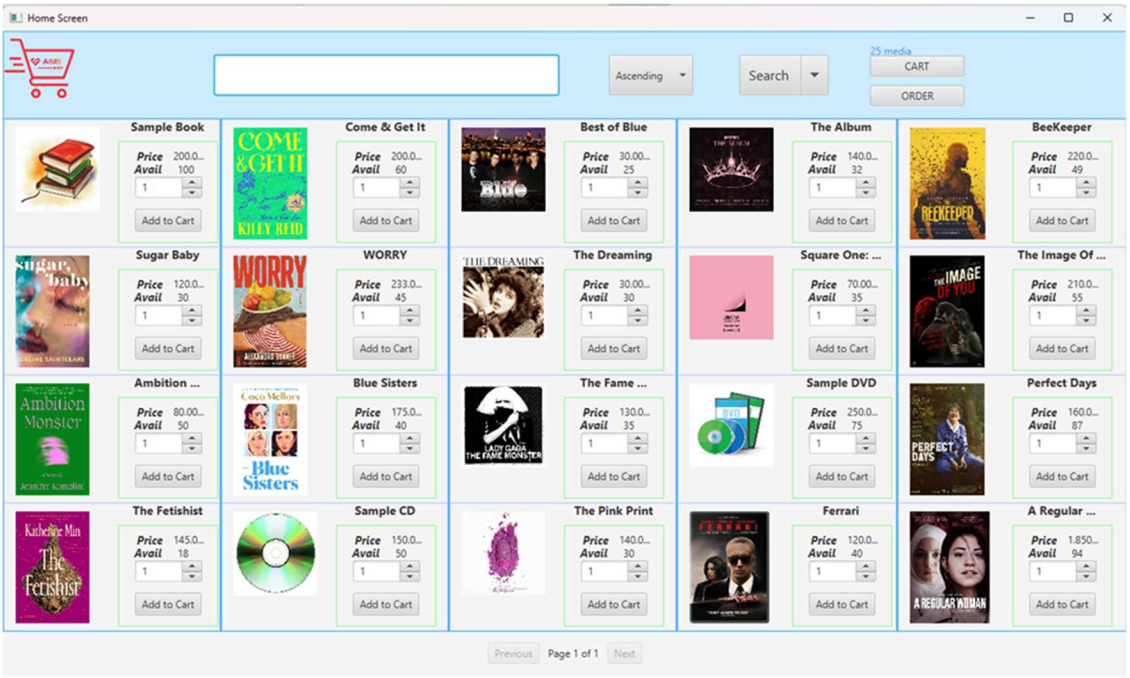


Control	**Operation**	**Function**
Button Login	Click	Login
“Username” attribute	Input	Allow user to input name
“Password” attribute	Input	Allow user to input password
Login as guest	Click	Login as guest

Table 11 Home Screen

AIMS SOFTWARE	
Screen specification	HomeScreen

Date of creation	Approved by	Reviewed by	Person in charge
07/04/2024			Group 10

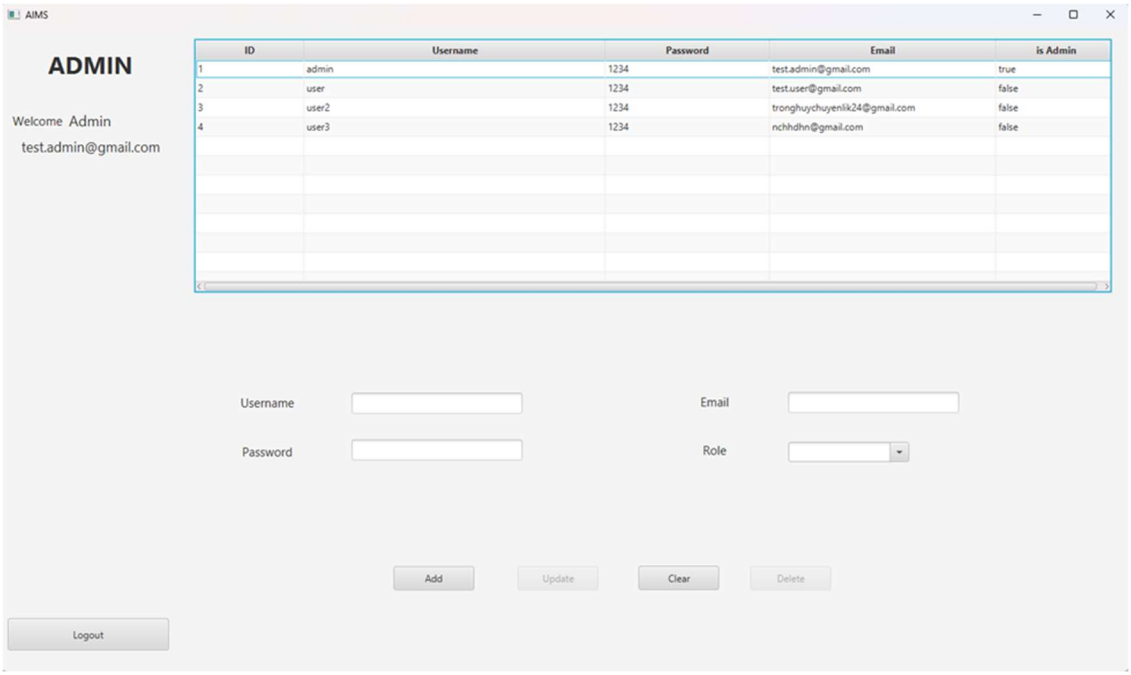


Control	Operation	Function
Area for display the list of products	Initialization	Display products.
Quantity spinner	Select	Display a product quantity in the cart.
Button Add to cart	Click	Add product to cart
Button Cart	Click	View cartScreen
Button Order	Click	View Order
Button search	Click	Search product
Button Ascending	Click	Sort product with price

Table 12 Admin Screen

AIMS SOFTWARE	
Screen specification	AdminScreen

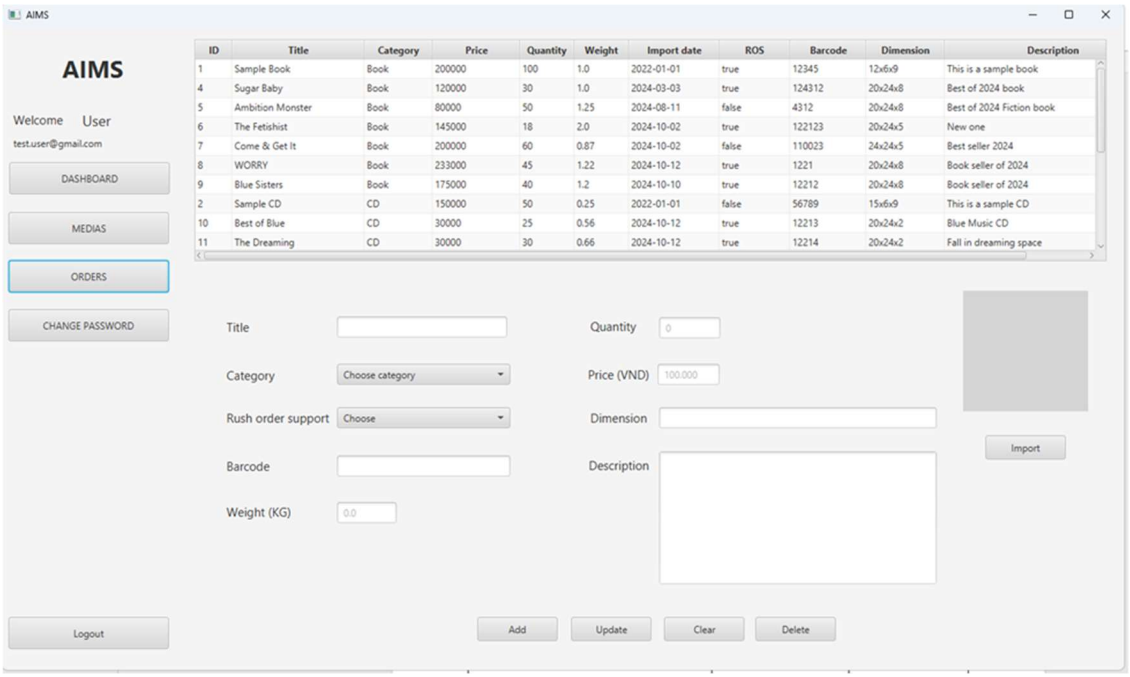
Date of creation	Approved by	Reviewed by	Person in charge
07/04/2024			Group 10



Control	Operation	Function
Area for display the list of account	Initialization	Display account.
Button Add	Click	Add Account
Button Update	Click	Update Account
Button Delete	Click	Delete Account
Button Logout	Click	Logout
“Username” attribute	Input	Allow user to input user
“Password” attribute	Input	Allow user to input password
“Email” attribute	Input	Allow user to input email
“Role” attribute	Input	Allow user to input role

Table 13 Product Manager Screen

AIMS SOFTWARE			
Screen specification		ProductmanagerScreen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10

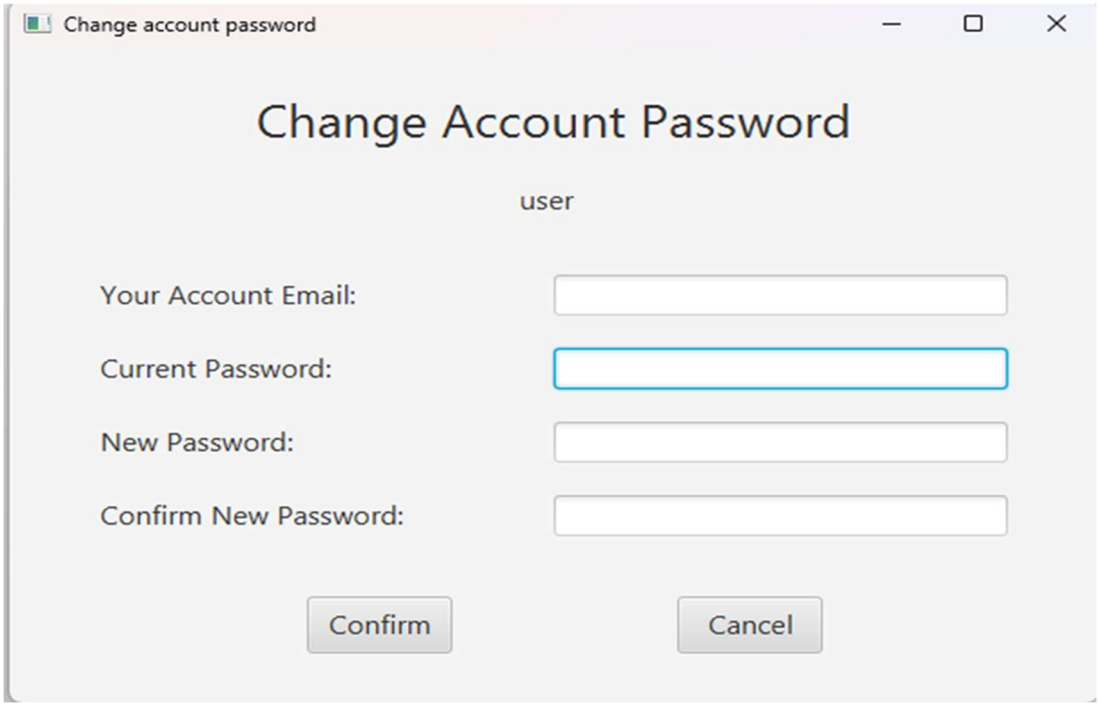


Control	Operation	Function
Area for display the list of account	Initialization	Display account.
Button Add	Click	Add media
Button Update	Click	Update media
Button Delete	Click	Delete media
Button Logout	Click	Logout
“Title” attribute	Input	Allow user to input title
“quantity” attribute	Input	Allow user to input quantity

“category” attribute	Input	Allow user to input category
“price” attribute	Input	Allow user to input price
“Rush order support” attribute	Input	Allow user to input rush order support
“Rush order support” attribute	Input	Allow user to input rush order support
“Dimension” attribute	Input	Allow user to input dimension
“Barcode” attribute	Input	Allow user to input Barcode
“Description” attribute	Input	Allow user to input Description
“Weight” attribute	Input	Allow user to input Description
Button import	Click	Import image

Table 14 Change Account Password Screen

AIMS SOFTWARE			
Screen specification		Change Account Password Screen	
Date of creation	**Approved by**	**Reviewed by**	**Person in charge**
07/04/2024			Group 10



Control	Operation	Function
Area for display the list of account	Initialization	Display account.
Button Confirm	Click	Update password account
Button Cancel	Click	Close the change account password screen
“Your Account Email” attribute	Input	Allow user to input email
“Current Password” attribute	Input	Allow user to input current password
“New Password” attribute	Input	Allow user to input new password
“Confirm New Password” attribute	Input	Allow user to input Confirm New Password

4.2. Data Modeling

4.2.1. Conceptual Data Modeling

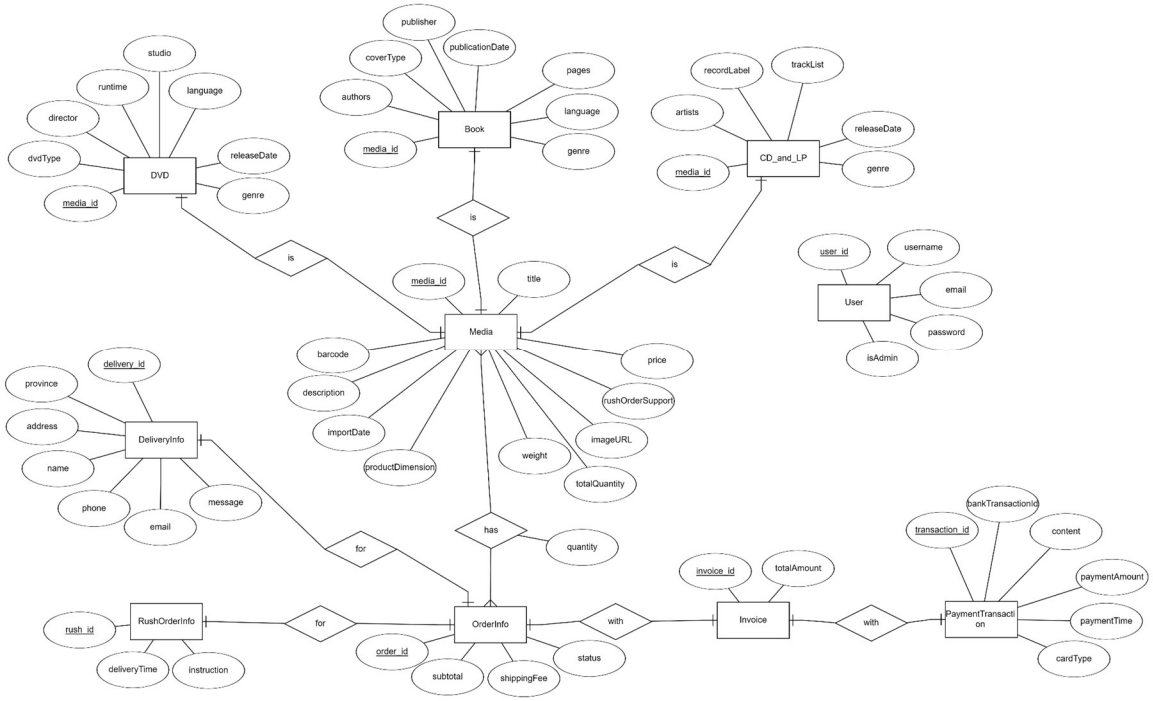


Figure 18 ERD diagram

4.2.2. Database Design

4.2.2.1. Database Management System

The decision for the Database Management System (DBMS) for this project is to use MySQL. MySQL is a widely used, open-source relational database management system known for its reliability, scalability, and ease of use. It supports a wide range of applications, from small-scale projects to large-scale enterprise systems, making it a versatile choice for our needs.

MySQL offers robust performance and security features, ensuring that data is stored efficiently and can be retrieved quickly, even under high load conditions. It supports various storage engines, including InnoDB, which provides ACID (Atomicity, Consistency, Isolation, Durability) compliance for reliable transaction processing. MySQL also offers strong data protection mechanisms, including data encryption, user authentication, and access control.

Additionally, MySQL integrates well with numerous programming languages and platforms, making it highly adaptable to different development environments. Its extensive documentation and active community support make it easier to troubleshoot issues and

implement best practices. Overall, MySQL's proven track record and comprehensive features make it the ideal choice for managing our database requirements.

4.2.2.2. Database Diagram

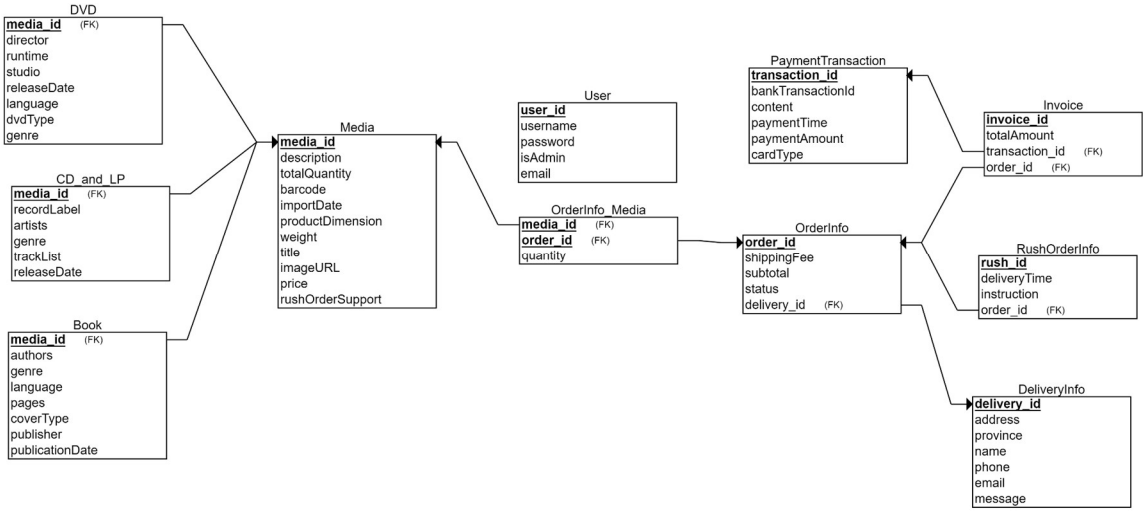


Figure 19 Database schema

4.2.2.3. Database Detail Design

Table 15 Media Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x		media_id	integer		yes	increment
2			description	varchar		yes	
3			totalQuantity	integer		yes	
4			barcode	varchar		yes	
5			importDate	date		yes	
6			productDimension	varchar		yes	
7			weight	real		yes	
8			title	varchar		yes	
9			imageURL	varchar		yes	
10			price	integer		yes	
11			rushOrderSupport	boolean		yes	

Table 16 DVD Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*

1	x	x	media_id	integer		yes	increment
2			director	varchar		yes	
3			runtime	integer		yes	
4			studio	varchar		yes	
5			releaseDate	date		yes	
6			language	varchar		yes	
7			DvdType	varchar		yes	
8			genre	varchar		yes	

Table 17 CD_and_LP Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x	x	media_id	integer		yes	increment
2			recordLabel	varchar		yes	
3			artists	varchar		yes	
4			genre	varchar		yes	
5			trackList	varchar		yes	
6			releaseDate	date		yes	

Table 18 Book Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x	x	media_id	integer		yes	increment
2			authors	varchar		yes	
3			genre	varchar		yes	
4			language	varchar		yes	
5			pages	int		yes	
6			coverType	varchar		yes	
7			publisher	varchar		yes	
8			publicationDate	date		yes	

Table 19 User Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*

1	x		user_id	integer		yes	increment
2			username	varchar		yes	
3			password	varchar		yes	
4			email	varchar		yes	
5			isAdmin	boolean		yes	

Table 20 DeliveryInfo Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x		delivery_id	integer		yes	increment
2			address	varchar		yes	
3			province	varchar		yes	
4			email	varchar		yes	
5			name	varchar		yes	
6			phone	varchar		yes	
7			messages	varchar		no	

Table 21 OrderInfo Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x		order_id	integer		yes	increment
2			status	varchar		yes	
3			subtotal	integer		yes	
4			shippingFee	integer		yes	
5		x	delivery_id	integer		yes	

Table 22 RushOrderInfo Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x		rush_id	integer		yes	increment
2		x	order_id	integer		yes	
3			instruction	varchar		no	
4			deliveryTime	datetime		yes	

Table 23 OrderInfo_Media Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1		x	order_id	integer		yes	
2		x	media_id	integer		yes	
3			quantity	integer		yes	

Table 24 Payment Transaction Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x		transaction_id	integer		yes	increment
2			bankTransactionId	integer		yes	
3			quantity	integer		yes	
4			content	varchar		yes	
5			paymentTime	datetime		yes	
6			paymentAmount	integer		yes	
7			cardType	varchar			

Table 25 Invoice Table Details

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	x		invoice_id	integer		yes	increment
2		x	transaction_id	integer		yes	
3		x	order_id	integer		yes	
			totalAmount	integer		yes	

INDEX

-- Index: delivery_id_index

CREATE INDEX delivery_id_index ON OrderInfo (delivery_id);

-- Index: transaction_id_index, order_id_index

CREATE INDEX invoice_transaction_id_index ON Invoice (transaction_id);

CREATE INDEX invoice_order_id_index ON Invoice (order_id);

-- Index: order_id_index

CREATE INDEX rush_order_id_index ON RushOrderInfo (order_id);

4.3. Class Design

4.3.1. General Class Diagram

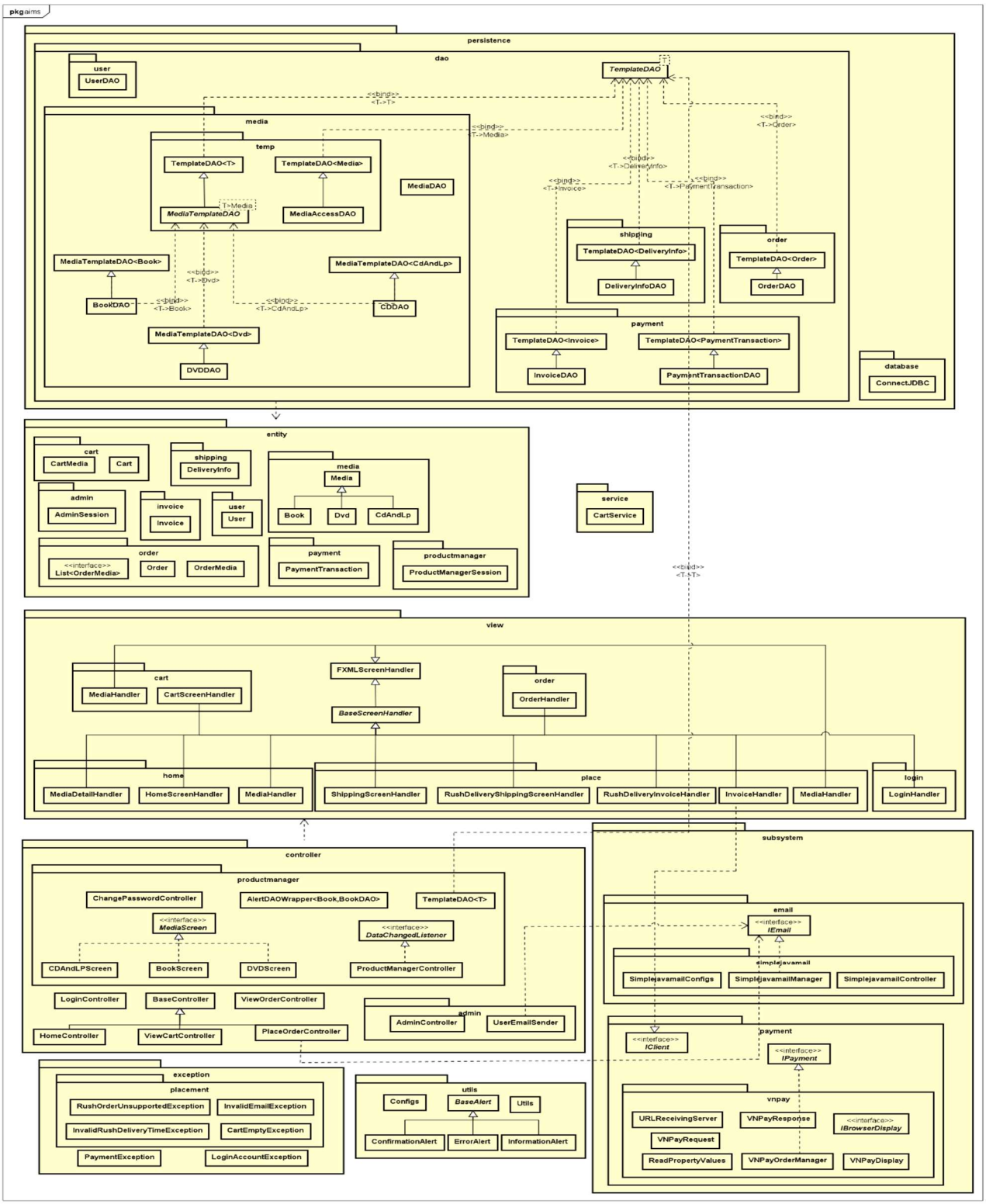


Figure 20 General Class Diagram

4.3.2. Class Diagrams

4.3.2.1. Class Diagram for Controller

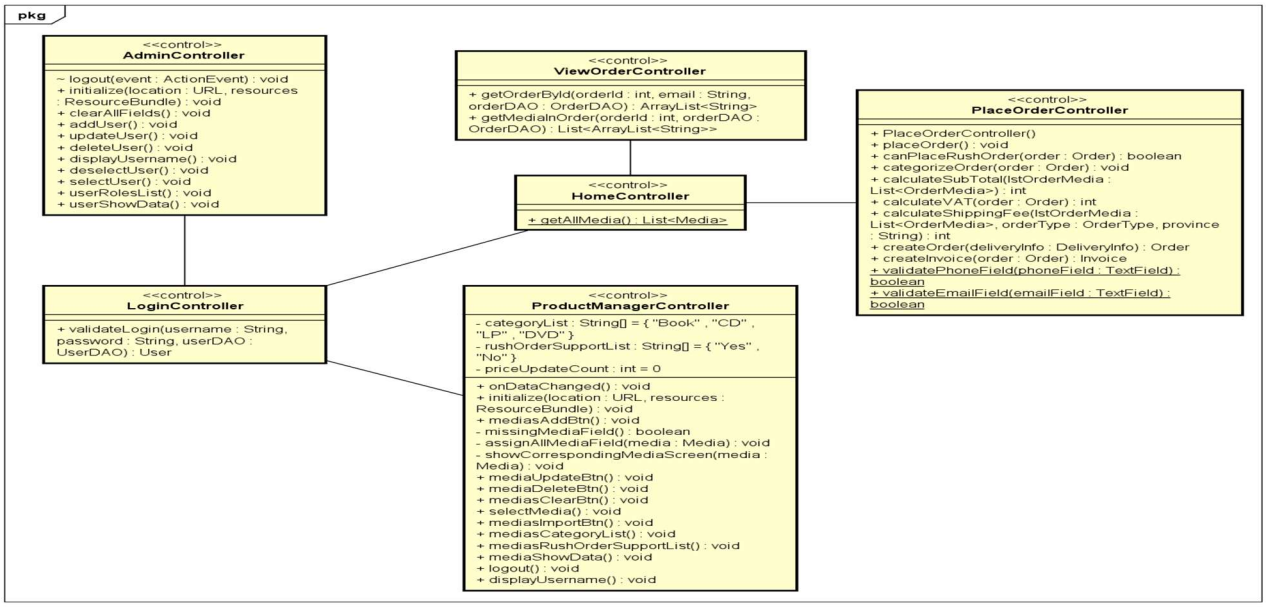


Figure 21 Class diagram for Controller

4.3.2.2. Class Diagram for Entity

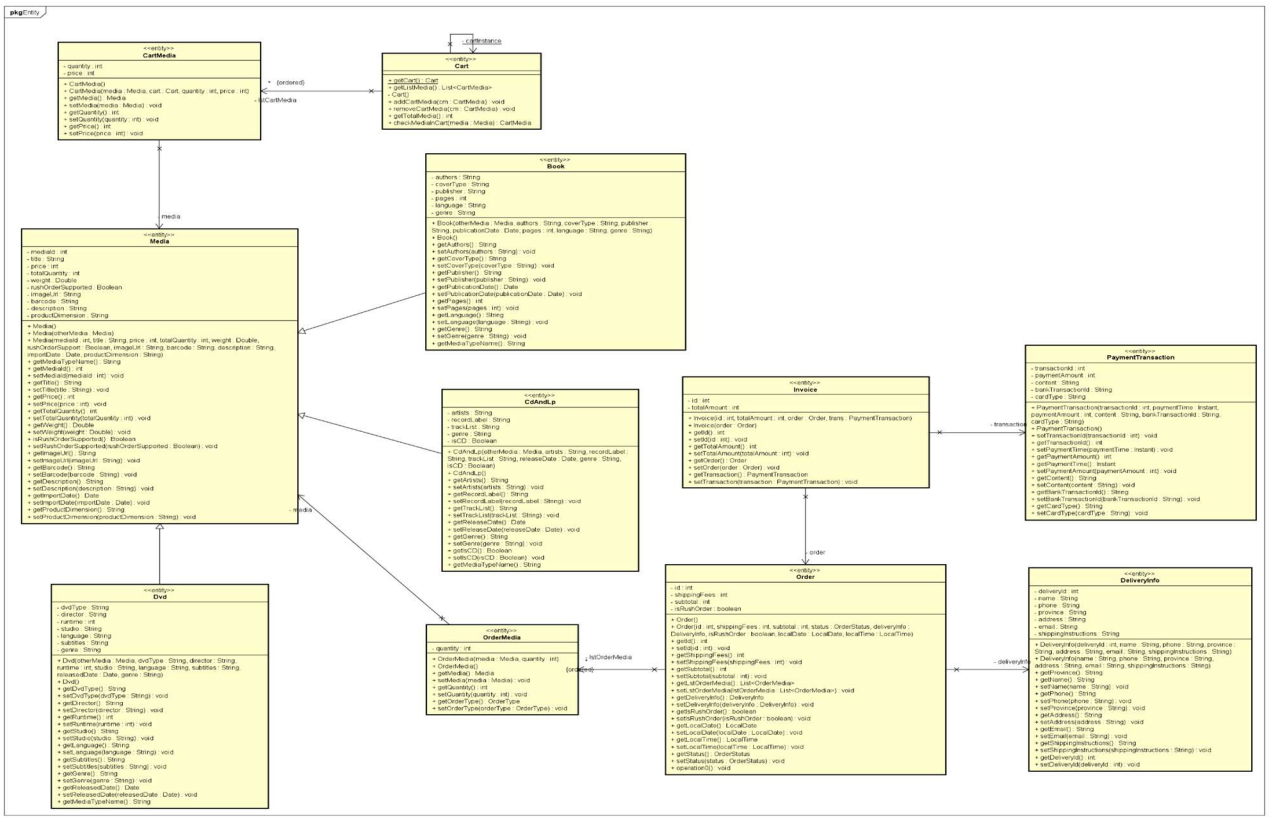


Figure 22 Class diagram for Entity

4.3.2.3. Class Diagram for Boundary

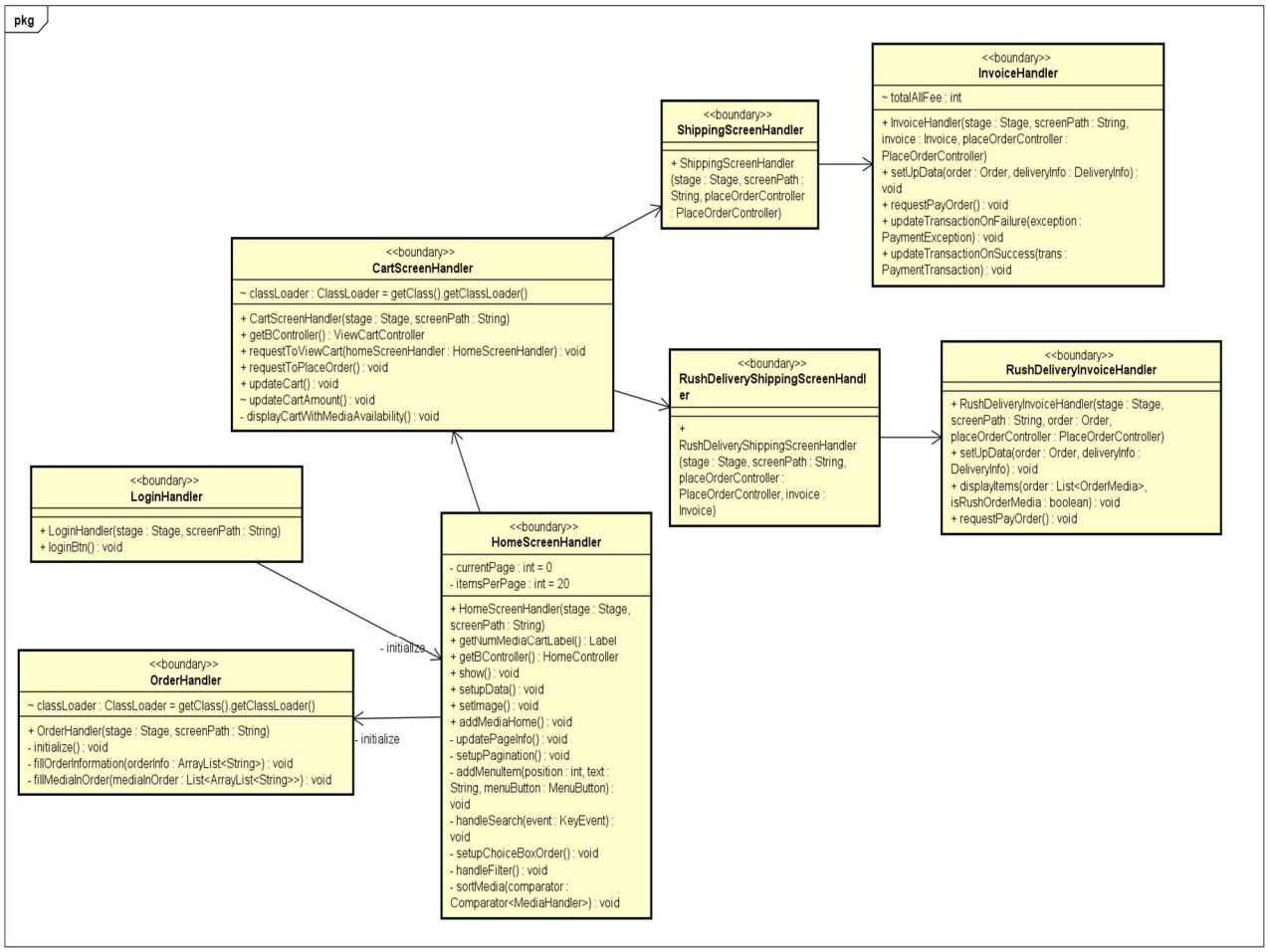


Figure 23 Class diagram for Boundary

4.3.3. Class Diagram for SubSystem

4.3.3.1. Email SubSystem

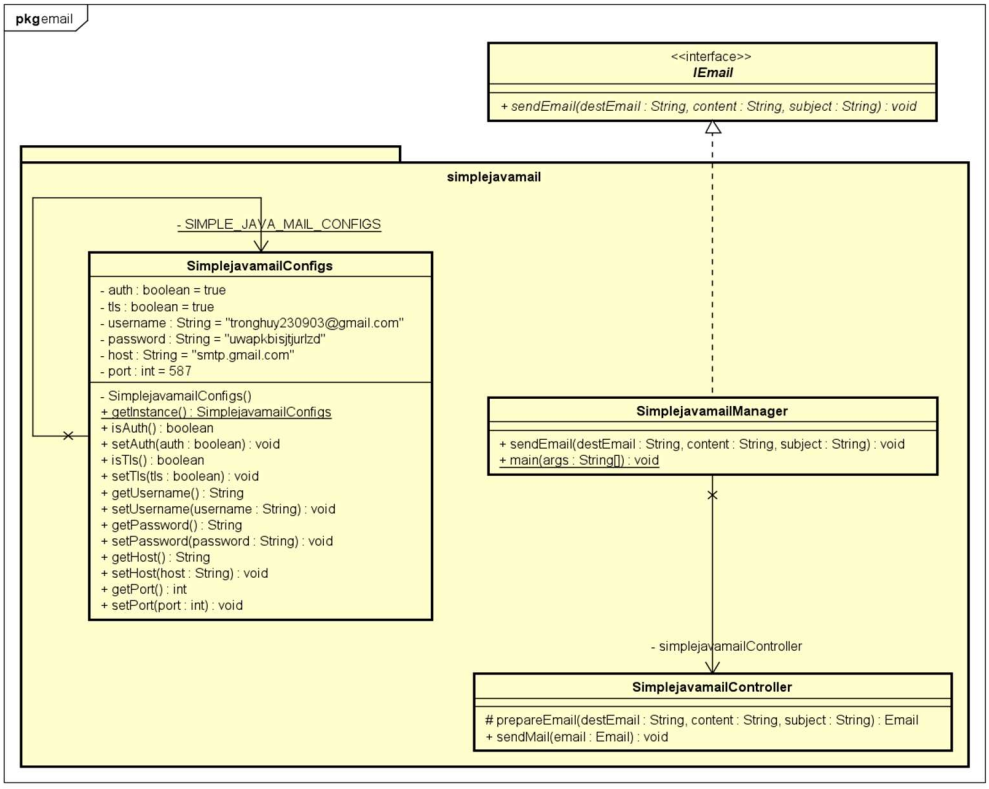


Figure 24 Email Subsystem Design

4.3.3.2. Payment SubSystem

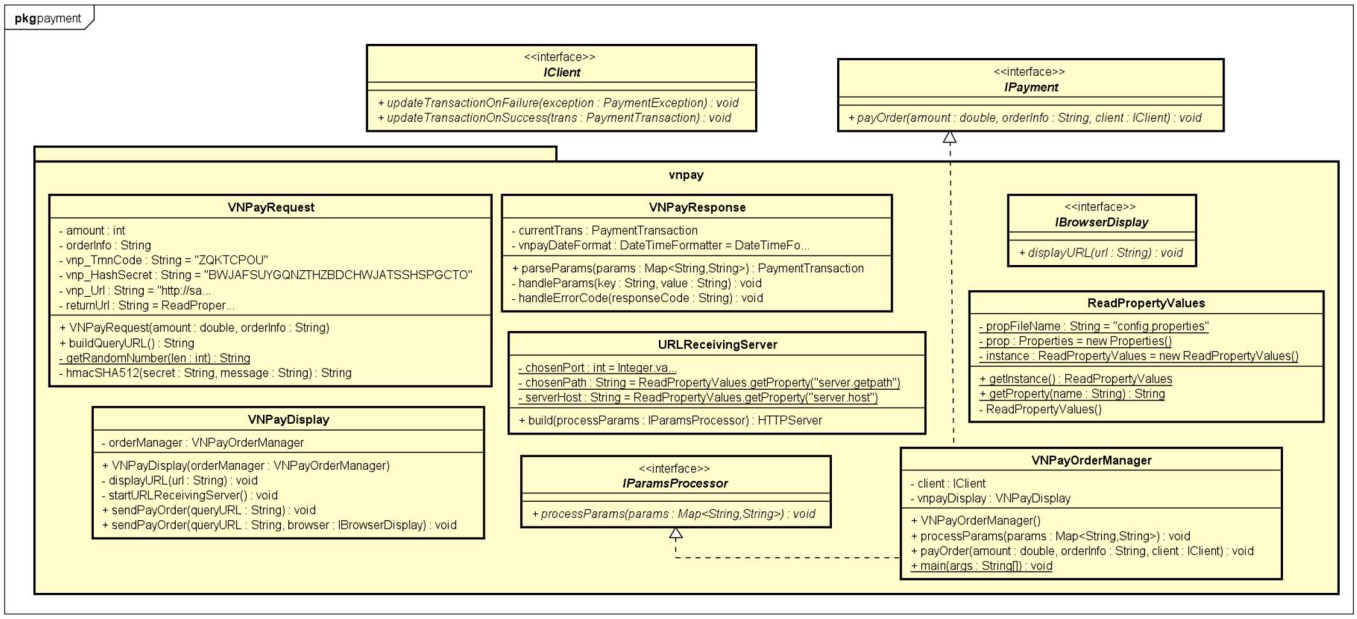


Figure 25 Payment Subsystem Design

4.4. Class Design Details

4.4.1.1. Class "Controller"

- Methods for AdminController

- **Method 1:**
  - **Name:** logout(event)
  - **Return type:** void
  - **Description:** Logs out the user
- **Method 2:**
  - **Name:** initialize(location, resources)
  - **Return type:** void
  - **Description:** Initializes the controller
- **Method 3:**
  - **Name:** ResourceBundle
  - **Return type:** void
  - **Description:** ResourceBundle for localization
- **Method 4:**
  - **Name:** addUser()
  - **Return type:** void
  - **Description:** Adds a user
- **Method 5:**
  - **Name:** updateUser()
  - **Return type:** void
  - **Description:** Updates a user
- **Method 6:**
  - **Name:** deleteUser()
  - **Return type:** void
  - **Description:** Deletes a user
- **Method 7:**
  - **Name:** displayUsername()
  - **Return type:** void
  - **Description:** Displays the username
- **Method 8:**
  - **Name:** selectUser()
  - **Return type:** void
  - **Description:** Selects a user
- **Method 9:**
  - **Name:** userRoles()

    - **Return type:** void
    - **Description:** Manages user roles
- **Method 10:**
    - **Name:** userShowData()
    - **Return type:** void
    - **Description:** Shows user data
- **Methods for LoginController**
- **Method 11:**
    - **Name:** validateLogin(username, password)
    - **Return type:** User
    - **Description:** Validates login credentials
- **Methods for HomeController**
- **Method 12:**
    - **Name:** getAllMedia()
    - **Return type:** List<Media>
    - **Description:** Retrieves all media items
- **Methods for PlaceOrderController**
- **Method 13:**
    - **Name:** PlaceOrderController()
    - **Return type:** Constructor
    - **Description:** Constructor method
- **Method 14:**
    - **Name:** placeOrder()
    - **Return type:** void
    - **Description:** Places an order
- **Method 15:**
    - **Name:** canPlaceRushOrder(order)
    - **Return type:** boolean
    - **Description:** Checks if a rush order can be placed
- **Method 16:**
    - **Name:** calculateOrderSubtotal(order)
    - **Return type:** void
    - **Description:** Calculates the order subtotal
- **Method 17:**
    - **Name:** listOrderMethods()
    - **Return type:** List<String>
    - **Description:** Lists the order methods
- **Method 18:**
    - **Name:** calculateVAT(order)

    - **Return type:** void
    - **Description:** Calculates VAT for the order
- **Method 19:**
    - **Name:** calculateShipping(order)
    - **Return type:** void
    - **Description:** Calculates shipping cost for the order
- **Method 20:**
    - **Name:** createOrder(deliveryInfo)
    - **Return type:** Invoice
    - **Description:** Creates an order with delivery information
- **Method 21:**
    - **Name:** validatePhoneField(phoneField)
    - **Return type:** boolean
    - **Description:** Validates the phone field input
- **Method 22:**
    - **Name:** validateEmailField(emailField)
    - **Return type:** boolean
    - **Description:** Validates the email field input
- **Methods for ProductManagerController**
- **Method 23:**
    - **Name:** ProductManagerController()
    - **Return type:** Constructor
    - **Description:** Constructor method
- **Method 24:**
    - **Name:** initialize(location, resources)
    - **Return type:** void
    - **Description:** Initializes the controller
- **Method 25:**
    - **Name:** ResourceBundle
    - **Return type:** void
    - **Description:** ResourceBundle for localization
- **Method 26:**
    - **Name:** missingField(fieldName)
    - **Return type:** boolean
    - **Description:** Checks if a required field is missing
- **Method 27:**
    - **Name:** missingMediaField(media)
    - **Return type:** void
    - **Description:** Checks if media field is missing

- **Method 28:**
    - **Name:** showMediaScreen(media)
    - **Return type:** void
    - **Description:** Displays the media screen
- **Method 29:**
    - **Name:** mediaUpdated()
    - **Return type:** void
    - **Description:** Updates media information
- **Method 30:**
    - **Name:** mediaDeleted()
    - **Return type:** void
    - **Description:** Deletes media information
- **Method 31:**
    - **Name:** mediaSelected()
    - **Return type:** void
    - **Description:** Selects media information
- **Method 32:**
    - **Name:** mediaRushOrderSupportList()
    - **Return type:** void
    - **Description:** Lists media with rush order support
- **Method 33:**
    - **Name:** displayUsername()
    - **Return type:** void
    - **Description:** Displays the username
- **Methods for ViewOrderController**
- **Method 34:**
    - **Name:** getOrderById(orderId, email)
    - **Return type:** Order
    - **Description:** Retrieves an order by ID
- **Method 35:**
    - **Name:** getOrderDAO()
    - **Return type:** ArrayList<String>
    - **Description:** Gets the Order Data Access Object
- **Method 36:**
    - **Name:** getMediaInOrder(orderId)
    - **Return type:** List<ArrayList<String>>
    - **Description:** Retrieves media items in the order

4.4.1.2. Class "Entity"

- **CartScreenHandler**
  - Attributes:
    - stage
    - screenPath
    - cartController
  - Methods:
    - CartScreenHandler(stage, screenPath): Constructor for CartScreenHandler
    - requestToViewCart(homeScreenHandler): Handles the request to view the cart
    - requestToPlaceOrder(homeScreenHandler): Handles the request to place an order
    - initialize(): Initializes the CartScreenHandler
    - updateCart(): Updates the cart
    - displayCartMedia(mediaHandlerList): Displays the cart media
- **LoginHandler**
  - Attributes:
    - stage
    - screenPath
  - Methods:
    - LoginHandler(stage, screenPath): Constructor for LoginHandler
    - login(): Handles the login process
- **OrderHandler**
  - Attributes:
    - stage
    - screenPath
  - Methods:
    - OrderHandler(stage, screenPath): Constructor for OrderHandler
    - getOrderInformation(orderId): Retrieves order information
    - viewOrderDetails(orderId): Views order details
- **HomeScreenHandler**
  - Attributes:
    - stage
    - screenPath
    - user
    - mediaController
    - orderController
    - userController

- Methods:
    - HomeScreenHandler(stage, screenPath, user): Constructor for HomeScreenHandler
    - initialize(): Initializes the HomeScreenHandler
    - logout(): Handles user logout
    - goToCart(): Navigates to the cart screen
    - goToOrderHistory(): Navigates to the order history screen
    - goToUserProfile(): Navigates to the user profile screen
    - viewCart(): Displays the cart
    - viewMedia(media): Displays the media details
    - searchMedia(keyword): Searches for media based on the keyword

4.4.1.3. Class "BoundPackage"

- **CartScreenHandler**
    - Attributes: classLoader, prevStage, cartController
    - Methods:
        - CartScreenHandler(stage, screenPath): Constructor for CartScreenHandler
        - requestToViewCart(homeScreenHandler): Handles the request to view the cart
        - requestToPlaceOrder(homeScreenHandler): Handles the request to place an order
        - initialize(): Initializes the CartScreenHandler
        - updateCart(): Updates the cart
        - displayCartMedia(mediaHandlerList): Displays the cart media
- **LoginHandler**
    - Attributes: classLoader
    - Methods:
        - LoginHandler(stage, screenPath): Constructor for LoginHandler
        - login(): Handles the login process
- **OrderHandler**
    - Attributes: classLoader
    - Methods:
        - OrderHandler(stage, screenPath): Constructor for OrderHandler
        - getOrderInformation(orderId): Retrieves order information
        - viewOrderDetails(orderId): Views order details
- **HomeScreenHandler**
    - Attributes: user, mediaController, orderController, userController
    - Methods:

        - HomeScreenHandler(stage, screenPath, user): Constructor for HomeScreenHandler
        - initialize(): Initializes the HomeScreenHandler
        - logout(): Handles user logout
        - goToCart(): Navigates to the cart screen
        - goToOrderHistory(): Navigates to the order history screen
        - goToUserProfile(): Navigates to the user profile screen
        - viewCart(): Displays the cart
        - viewMedia(media): Displays the media details
        - searchMedia(keyword): Searches for media based on the keyword
- **ShippingScreenHandler**
-
    - Attributes: placeOrderController
    - Methods:
        - ShippingScreenHandler(stage, screenPath, placeOrderController):Constructor for ShippingScreenHandler
        - toConfirmOrder(): Navigates to the confirm order screen
- **RunDeliveryScreenHandler**
    - Attributes: placeOrderController, invoice
    - Methods:
        - RunDeliveryScreenHandler(stage, screenPath, placeOrderController, invoice): Constructor for RunDeliveryScreenHandler
        - viewOrderDetails(): Views order details
        - confirmOrder(): Confirms the order
        - cancelOrder(): Cancels the order
        - initialize(): Initializes the RunDeliveryScreenHandler
- **RunDeliveryShippingScreenHandler**
    - Attributes: placeOrderController, invoice
    - Methods:
        - RunDeliveryShippingScreenHandler(stage, screenPath, placeOrderController, invoice): Constructor for RunDeliveryShippingScreenHandler
        - confirmOrder(): Confirms the order
        - cancelOrder(): Cancels the order
        - initialize(): Initializes the RunDeliveryShippingScreenHandler

4.4.1.4. Class “Payment”

- **CartScreenHandler**
    - Attributes: classLoader, prevStage, cartController

    - Methods:
        - CartScreenHandler(stage, screenPath): Constructor for CartScreenHandler
        - requestToViewCart(homeScreenHandler): Handles the request to view the cart
        - requestToPlaceOrder(homeScreenHandler): Handles the request to place an order
        - initialize(): Initializes the CartScreenHandler
        - updateCart(): Updates the cart
        - displayCartMedia(mediaHandlerList): Displays the cart media
- **LoginHandler**
    - Attributes: classLoader
    - Methods:
        - LoginHandler(stage, screenPath): Constructor for LoginHandler
        - login(): Handles the login process
- **OrderHandler**
    - Attributes: classLoader
    - Methods:
        - OrderHandler(stage, screenPath): Constructor for OrderHandler
        - getOrderInformation(orderId): Retrieves order information
        - viewOrderDetails(orderId): Views order details
- **HomeScreenHandler**
    - Attributes: user, mediaController, orderController, userController
    - Methods:
        - HomeScreenHandler(stage, screenPath, user): Constructor for HomeScreenHandler
        - initialize(): Initializes the HomeScreenHandler
        - logout(): Handles user logout
        - goToCart(): Navigates to the cart screen
        - goToOrderHistory(): Navigates to the order history screen
        - goToUserProfile(): Navigates to the user profile screen
        - viewCart(): Displays the cart
        - viewMedia(media): Displays the media details
        - searchMedia(keyword): Searches for media based on the keyword

5. Design Considerations

5.1. Goals and Guidelines

The design of the system and its software is driven by several overarching goals, guidelines, principles, and priorities aimed at achieving a balance between performance, usability, maintainability, and security.

Goals

- Performance Optimization: The system is designed with an emphasis on speed and responsiveness. This priority ensures that user interactions are processed quickly, enhancing the overall user experience. Performance optimization includes efficient database queries, caching mechanisms, and minimizing the overhead in data processing and network communication.

- Scalability: The system architecture is designed to scale efficiently to accommodate growing numbers of users and data. This involves modular design, use of scalable technologies like MySQL for the database, and cloud infrastructure that can expand as needed.

- Security and Privacy: Ensuring the security and privacy of user data is a top priority. The system incorporates strong encryption protocols, robust authentication and authorization mechanisms, and comprehensive logging and auditing practices to protect sensitive information and comply with relevant regulations.

- User-Friendly Interface: The system aims to provide an intuitive and consistent user interface. This goal ensures that users can easily navigate and interact with the system, reducing the learning curve and improving overall satisfaction.

Guidelines

- Coding Standards: Adherence to coding guidelines and conventions is essential for maintaining code quality and readability. The use of standardized naming conventions, consistent code formatting, and thorough documentation helps in reducing errors and facilitates easier code reviews and maintenance.

- Modular Design: The system follows a modular design approach, where functionality is divided into discrete, self-contained modules. This guideline aids in isolating different parts of the system, making it easier to manage, test, and update individual components without affecting the overall system.

- Error Handling: Implementing robust error handling mechanisms ensures that the system can gracefully recover from unexpected issues and provide meaningful feedback to

users and developers. This includes logging errors for diagnostics and providing user-friendly error messages.

5.2. Architectural Strategies

Programming Language and Frameworks

The system is developed using Java as the primary programming language, with JavaFX for building the graphical user interface (GUI). This combination leverages Java's robustness and platform independence along with JavaFX's rich set of UI components and capabilities for creating modern, responsive, and user-friendly desktop applications. The project is managed using Maven, which facilitates dependency management, builds automation, and overall project management.

Database Management System

As previously mentioned, the system uses MySQL as the DBMS. MySQL's reliability, performance, and strong support for ACID transactions make it suitable for our needs. Additionally, MySQL's compatibility with a wide range of applications and its robust security features align with our security and scalability goals.

User Interface Paradigms

The user interface follows a Rich Client Application paradigm using JavaFX. This approach provides a rich and interactive user experience with dynamic content updates and a responsive design. JavaFX allows for the creation of sophisticated GUI components, ensuring that the application is both functional and visually appealing. The design adheres to usability and accessibility principles to cater to a broad user base.

Error Detection and Recovery

The system incorporates comprehensive error detection and recovery mechanisms. In Java, try-catch blocks are used extensively to handle exceptions gracefully, providing meaningful feedback to users and logging errors for developers to review. JavaFX's built-in error handling capabilities ensure that UI-related errors do not crash the entire application, maintaining a smooth user experience.

Testing with JUnit

The system employs JUnit for unit testing, ensuring the reliability and correctness of the codebase. JUnit is an open-source testing framework for Java that provides an efficient way to write and run repeatable tests. The use of JUnit offers several benefits:

- Automated Testing: JUnit allows developers to write test cases that can be automatically executed, ensuring that the code functions as expected without manual intervention.

- Test Coverage: By writing comprehensive unit tests, the system can achieve high test coverage, identifying potential bugs and issues early in the development cycle.

Version Control with GitHub and 'Feature/Release' Workflow

The system uses Git for version control, hosted on GitHub to manage the source code efficiently and collaboratively. GitHub serves as a central repository where developers can push their changes, ensuring everyone has access to the latest version of the codebase. The version control process involves creating branches for new features or bug fixes, making and committing changes locally, and then pushing these changes to GitHub. Pull requests (PRs) facilitate code review and discussions before changes are merged into the main branch, maintaining high code quality and consistency.

The project follows a 'feature/release' workflow to streamline development and deployment. For new features or fixes, developers create a dedicated branch (feature/feature-name) from the main branch. Once development is complete, a PR is submitted for peer review. Upon approval, the feature branch is merged back into the main branch. For releases, a release/release-version branch is created, incorporating all approved features and fixes. This branch undergoes thorough testing and bug fixing before being merged into the main branch and tagged for release. This workflow ensures organized development, efficient collaboration, and smooth, stable releases.

5.3. Coupling and Cohesion

Coupling

The system is designed to achieve low coupling between its components. Low coupling is essential for enhancing modularity, making the system easier to maintain, extend, and test. Evidence of low coupling in the design includes:

Modular Architecture: The use of distinct modules for different functionalities (e.g., UI with JavaFX, backend logic with Java, and data persistence with MySQL) ensures that changes in one module have minimal impact on others.

Interfaces and Abstraction: Key components interact through well-defined interfaces and abstract classes, reducing direct dependencies. For example, data access objects (DAOs) interact with the database via Hibernate, which abstracts the underlying database operations.

Cohesion

The design aims for high cohesion within each component, ensuring that each module or class is focused on a single task or closely related tasks. High cohesion improves code readability, maintainability, and reusability. Evidence of high cohesion includes:

Single Responsibility Principle: Classes and methods are designed to have a single responsibility. For instance, UI controllers in JavaFX handle user interactions, while service classes encapsulate business logic.

Consistent Design Patterns: The use of design patterns like Entity - Boundary - Controller (EBC) ensures that the system's structure promotes high cohesion.

5.4. Design Patterns

1. Singleton design pattern

```
public class Cart {  19 usages  NTHuyne +1
    private final List<CartMedia> lstCartMedia;  6 usages
    private static Cart cartInstance;  3 usages

    public static Cart getCart(){  hwangisgone +1
        if(cartInstance == null) {
            cartInstance = new Cart();
        }
        return cartInstance;
    }

    public List<CartMedia> getListMedia() { return lstCartMedia; }

    private Cart() { this.lstCartMedia = new ArrayList<>(); }

    public void addCartMedia(CartMedia cm) { lstCartMedia.add(cm); }

    public void removeCartMedia(CartMedia cm) { lstCartMedia.remove(cm); }

    public int getTotalMedia(){  3 usages  NTHuyne
        int total = 0;
        for (CartMedia obj : lstCartMedia) {
            total += obj.getQuantity();
        }
        return total;
    }

    public CartMedia checkMediaInCart(Media media){  1 usage  NTHuyne +1
        for (CartMedia cartMedia : lstCartMedia) {
            if (cartMedia.getMedia().getMediaId() == media.getMediaId()) {
                return cartMedia;
            }
        }
        return null;
    }
}
```

Figure 26 Singleton for Cart class

Figure 27 Singleton for SimplejavamailConfig class

The Singleton design pattern ensures a class has only one instance while providing a global point of access to that instance. This is particularly beneficial in scenarios where a single object is needed to coordinate actions across a system, such as logging, configuration management, or managing connections to a resource like a database. By controlling the instantiation process, Singletons help in reducing memory overhead and potential conflicts caused by having multiple instances. They also promote consistent access and can simplify debugging and maintenance by centralizing the control logic.

2. Template Method Design Pattern

```java
// Utilizing Template Design Pattern
public abstract class TemplateDAO<T> {  16 usages  9 inheritors  hwangisgone
    protected final Connection connection;

    public TemplateDAO() { this.connection = ConnectJDBC.getConnection(); }

    public TemplateDAO(Connection conn) { this.connection = conn; }

    public abstract String getDaoName();  8 implementations  hwangisgone

    // Override these methods
    protected String getAllQuery() throws SQLException {  2 usages  5 overrides  hwangisgone
        throw new SQLException("Unimplemented GetAll for " + getDaoName() +" DAO");
    }

    protected String getByIdQuery() throws SQLException {  1 usage  7 overrides  hwangisgone
        throw new SQLException("Unimplemented GetById for " + getDaoName() +" DAO");
    }

    protected String deleteQuery() throws SQLException {  1 usage  3 overrides  hwangisgone
        throw new SQLException("Unimplemented Delete for " + getDaoName() +" DAO");
    }

    // For getAll and getById
    protected T createItemFromResultSet(ResultSet res) throws SQLException {  6 usages  8 overrides  hwangisgone
        System.err.println("Cannot create item for " + getDaoName() +" DAO");
        throw new SQLException("createItemFromResultSet is not implemented");
    }

    protected String addQuery() throws SQLException {  1 usage  8 overrides  hwangisgone
        throw new SQLException("Unimplemented Add for " + getDaoName() +" DAO");
    }
    protected void addParams(PreparedStatement stmt, T item) throws SQLException {  1 usage  8 overrides  hwangisgone
        throw new SQLException("Unimplemented Add for " + getDaoName() +" DAO");
    }

    protected String updateQuery() throws SQLException {  1 usage  4 overrides  hwangisgone
        throw new SQLException("Unimplemented Update for " + getDaoName() +" DAO");
    }
    protected void updateParams(PreparedStatement stmt, T item) throws SQLException {  1 usage  4 overrides  hwangisgone
        throw new SQLException("Unimplemented Update for " + getDaoName() +" DAO");
    }
```

Figure 28 Template method design pattern for DAO

The Template Method design pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. This pattern lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. It promotes code reuse and enables a high-level outline of an operation while allowing subclasses to refine or extend specific parts of the operation. By encapsulating invariant parts of an algorithm and allowing variation only in specific parts, it enhances flexibility and ensures consistent execution across different implementations. This pattern is particularly useful in scenarios where multiple classes share the same algorithm but differ in specific steps.

3. Strategy Design Pattern

```
package com.hust.ict.aims.subsystem.payment;

import ...

public interface IClient {  6 usages  1 implementation  hwangisgone
    void updateTransactionOnFailure(PaymentException exception);  1 usage  1 implementation  hwangisgone
    void updateTransactionOnSuccess(PaymentTransaction trans);  1 usage  1 implementation  hwangisgone
}
```

Figure 29 Strategy design pattern