HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

An Internet Media Store

Subject: ITSS Software Development

Group 09

Le Xuan Hieu 20215201(Leader)

Nguyen Ha Hieu 20215202

Nguyen Manh Hieu 20215203

Nguyen Van Hieu 20215204

Pham Trung Hieu 20215205

Hanoi, June 2024

Table of Contents

1 Introduction

1.1 Objective

The purpose of this Software Design Document is providing a comprehensive understanding of the AIMS Project. The following subsections of the SDD should provide an overview of the entire AIMS

1.2 Scope

The software product to be produced is AIMS, a desktop e-commerce software.

AIMS Project is a desktop e-commerce software that operates 24/7, allowing new users to easily familiarize themselves. This software can serve up to 1,000 customers simultaneously without significantly reducing performance and can operate continuously for 300 hours without failure. Additionally, the software can resume normal operation within a maximum of 1 hour after an incident. The maximum response time of the software is 2 seconds under normal conditions or 5 seconds during peak hours.

1.3 Glossary

CRUD	Include: Create, Read, Update, Delete

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

AIMS is designed and implemented based on several core principles and strategies to ensure its effectiveness, scalability, security, and maintainability.

Principles:

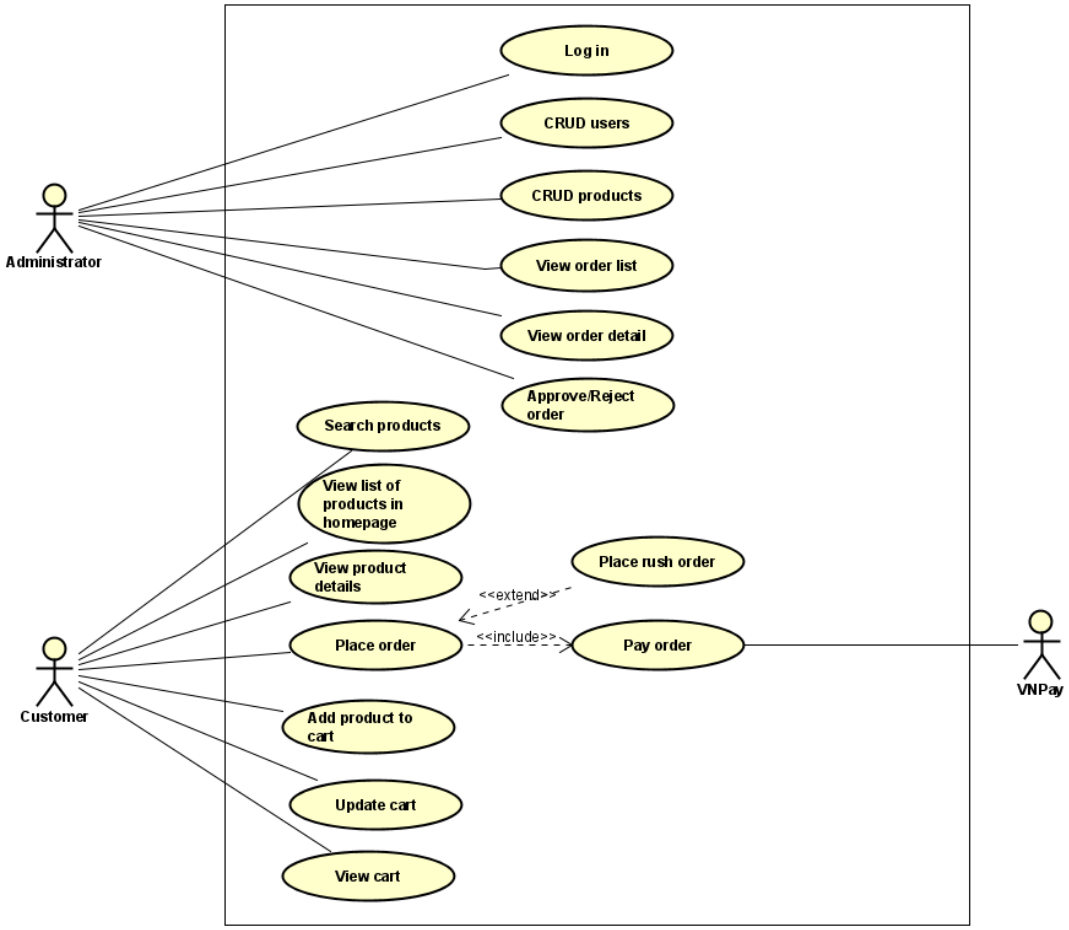
- Modularity and Reusability: AIMS is structured into modular components to promote reusability and facilitate easier maintenance and updates. Each module encapsulates specific functionalities, adhering to principles of object-oriented design.
- Scalability: The system is architected to handle increasing demands and data volumes. It employs scalable technologies and patterns such as microservices architecture to distribute workload efficiently across servers.
- Maintainability: The system is designed for ease of maintenance and extensibility. It utilizes clean, well-documented code, version control systems, and adheres to coding standards and best practices. Automated testing ensures early detection of issues.

Strategies:

Test-Driven Development (TDD): AIMS adopts TDD practices with comprehensive test suites covering unit tests, integration tests, and end-to-end tests. Testing frameworks like JUnit.

2.1 General Overview

- ***Overall requirements***



2.2 Assumptions/Constraints/Risks

2.2.1 Assumptions

- *E-commerce Users*
  The end user can be a customer shopping on an e-commerce platform, who uses the system to pay for their order.
- *Staff*
  Administrators use the system to manage orders, update goods quantities and handle issues related to payment transactions
  Possible Changes in Functionality
- Payment Service Changes
  The system may need to integrate with other payment services besides VNPay in the future. This may require extending or changing the IPayment interface to support new services.

- Database Expansion
  The system may require database expansion to store more information about customers, products or transactions.
- New feature
  The system can add new features such as installment support, promotions, or integration with shipping systems to provide delivery information.
- User interface
  The user interface may be updated or changed to improve the user experience, including changing the design or adding new functionality.

2.2.2 Constraints

Client Devices

The end-users might use a variety of client devices (desktops, laptops, tablets, smartphones). The system needs a responsive design to provide a consistent user experience across all devices. Testing must cover multiple device types and screen sizes.

Integration with Payment Gateways

The system must integrate seamlessly with various payment gateways (e.g., VNPay). This requires adherence to specific APIs and protocols provided by these services.

Data Storage

The choice between centralized and distributed storage impacts data consistency, performance, and availability. A centralized database might face latency issues, while distributed storage complicates data synchronization.

2.2.3 Risks

- Inadequate Access Control

Description: Insufficiently restrictive access controls can lead to unauthorized access to system functionalities or data.

Mitigation: Implement role-based access control (RBAC), conduct regular access reviews, and enforce the principle of least privilege.

- Performance Risks

Scalability Issues

Description: The system might not handle increased load or high traffic effectively, leading to slow response times or crashes.

Mitigation: Design for scalability by using load balancing, caching, and microservices architecture. Perform load testing to identify and address bottlenecks.

- Data Integrity Risks

  Data Loss or Corruption

  Description: Data might be lost or corrupted due to system failures, bugs, or malicious activities.

  Mitigation: Implement robust backup and recovery procedures, use transaction management to ensure data consistency, and perform regular integrity checks.

- Concurrency Issues

  Description: Concurrent access to data can lead to inconsistencies if not managed properly.

  Mitigation: Use proper transaction isolation levels, locking mechanisms, and concurrency control techniques.

- Incomplete Requirements

  Description: Incomplete or unclear requirements can lead to scope creep and misaligned system functionalities.

  Mitigation: Ensure thorough requirement gathering, involve stakeholders in the development process, and maintain clear and updated documentation.

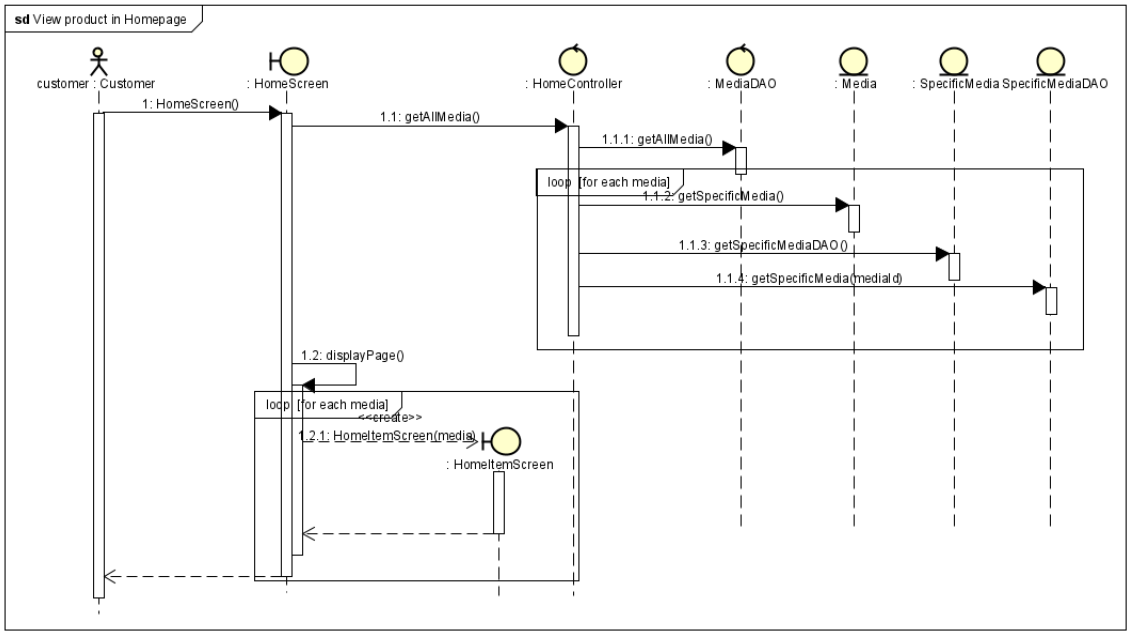
3 System Architecture and Architecture Design

3.1 Architectural Patterns

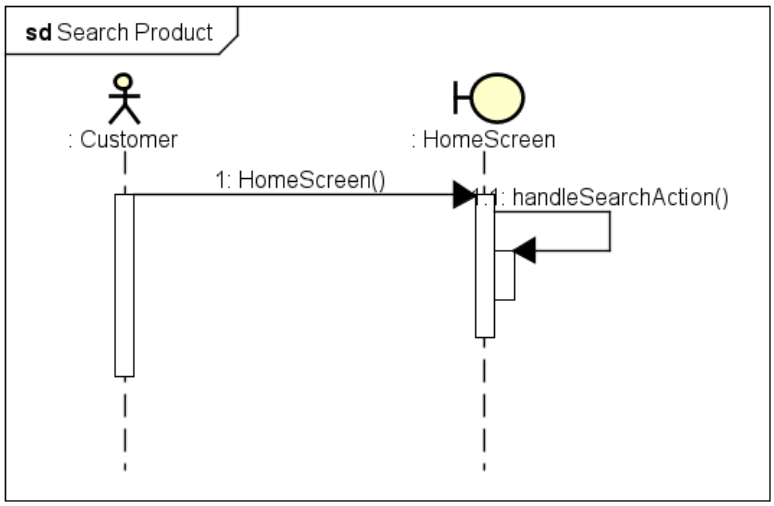
We chose the MVC (Model-View-Controller) model for our application to ensure a clear separation of concerns, enhancing maintainability, scalability, and testability. In our design, the View is responsible for rendering and handling FXML, allowing the UI to be modified independently of the business logic. The Controller coordinates actions and handles complex logic, serving as the intermediary between the View and the Model. The Model comprises entities that store data and interact with the database, ensuring that data management is decoupled from the user interface. This structure facilitates easier updates and maintenance, as changes in one component do not directly impact others.

3.2 Interaction Diagrams

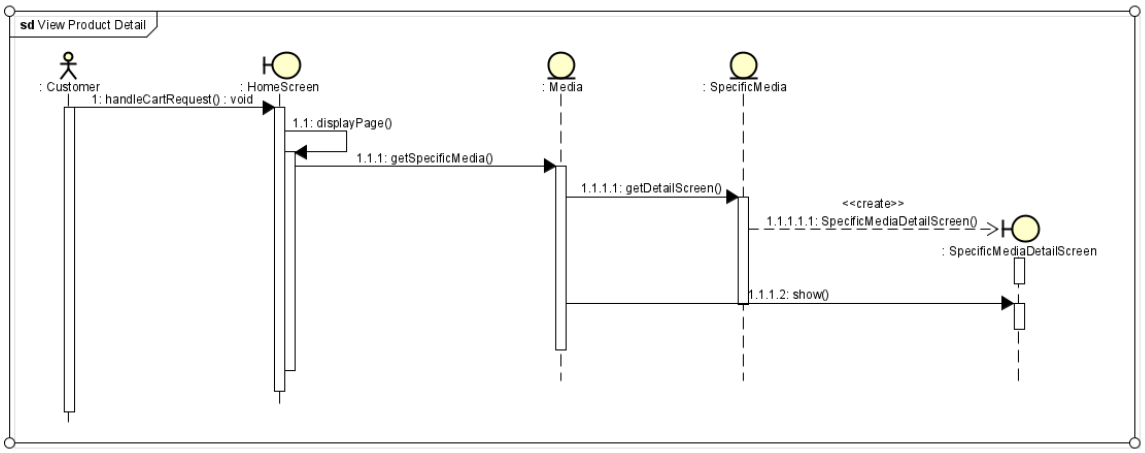
3.2.1. View list of products in homepage



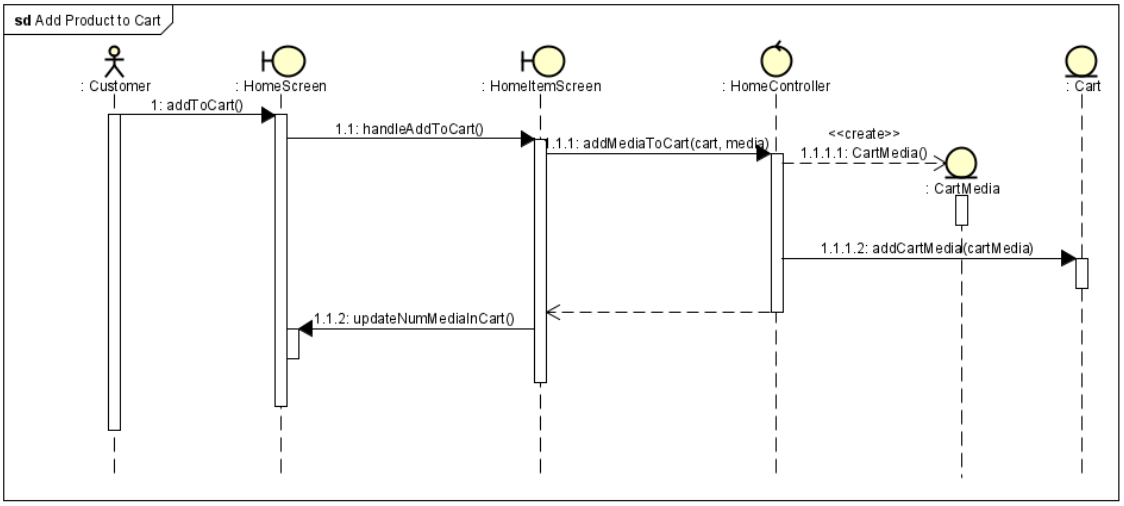
3.2.2. Search products



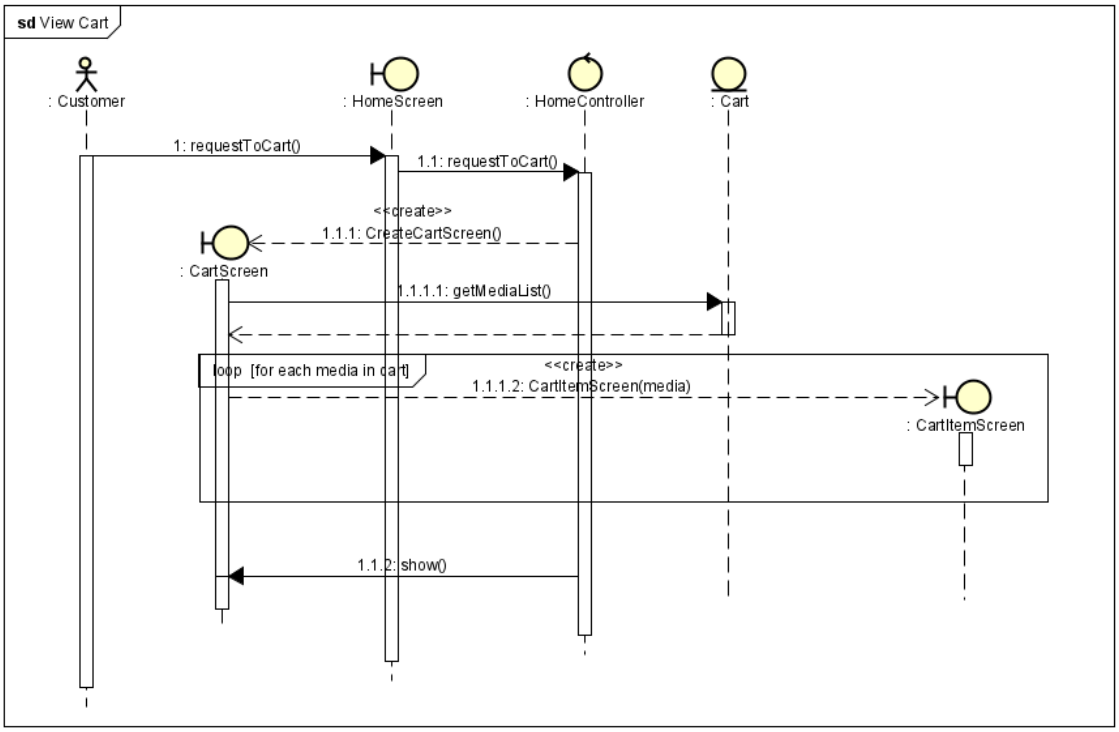
3.2.3. View product detail



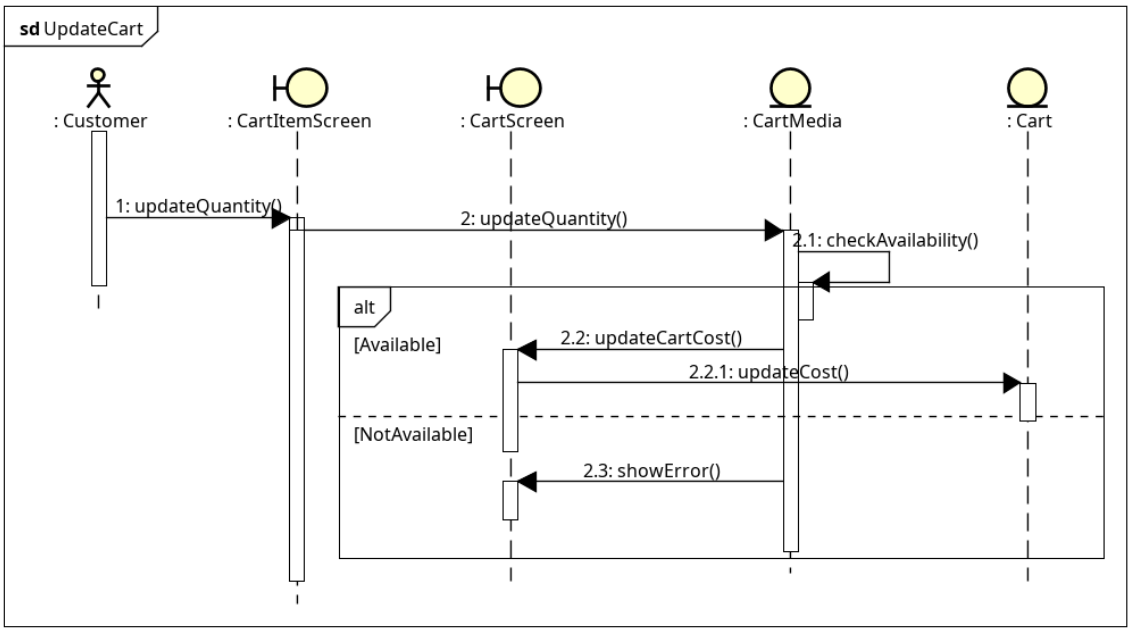
3.2.4. Add product to cart



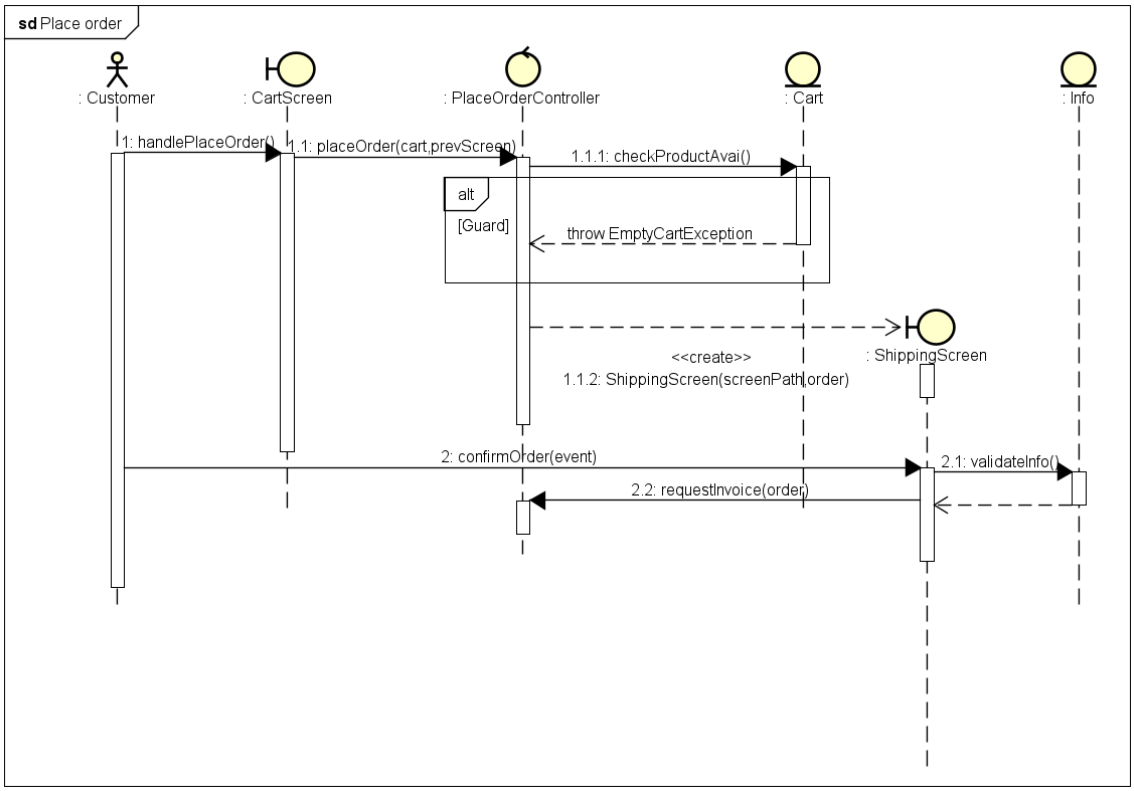
3.2.5. View cart



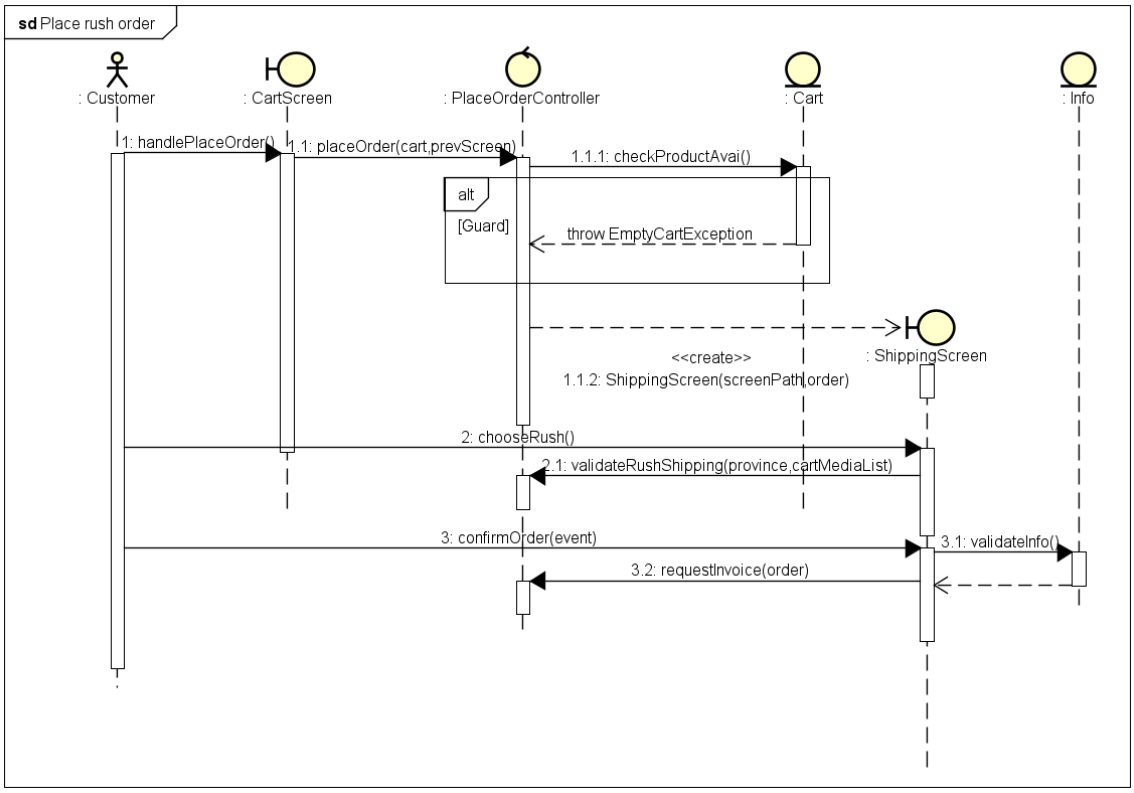
3.2.6. Update cart



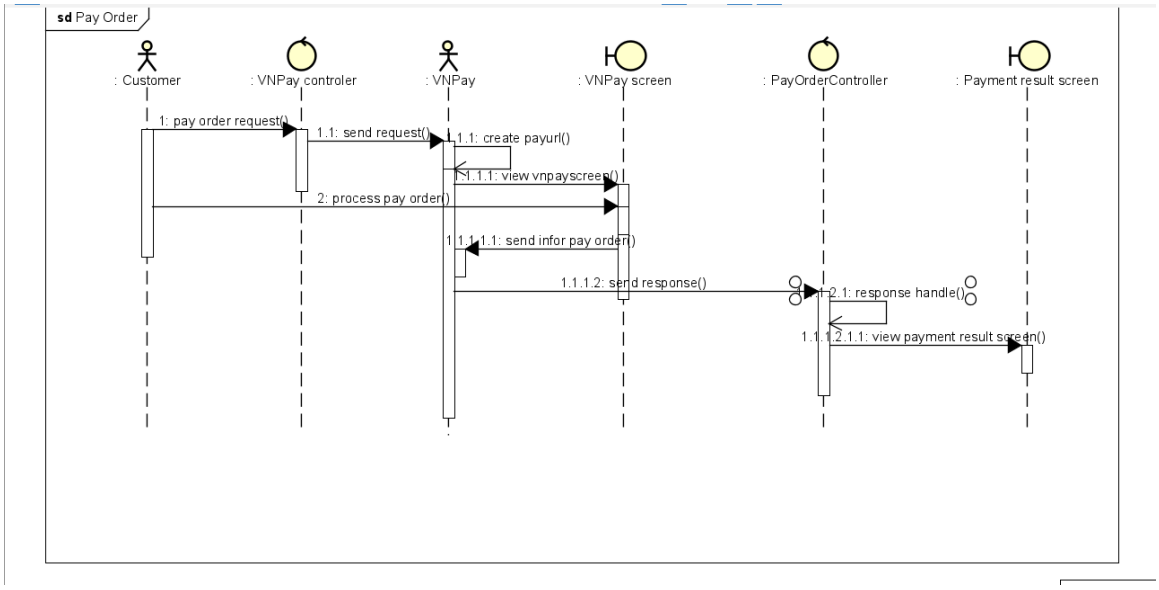
3.2.7. Place order



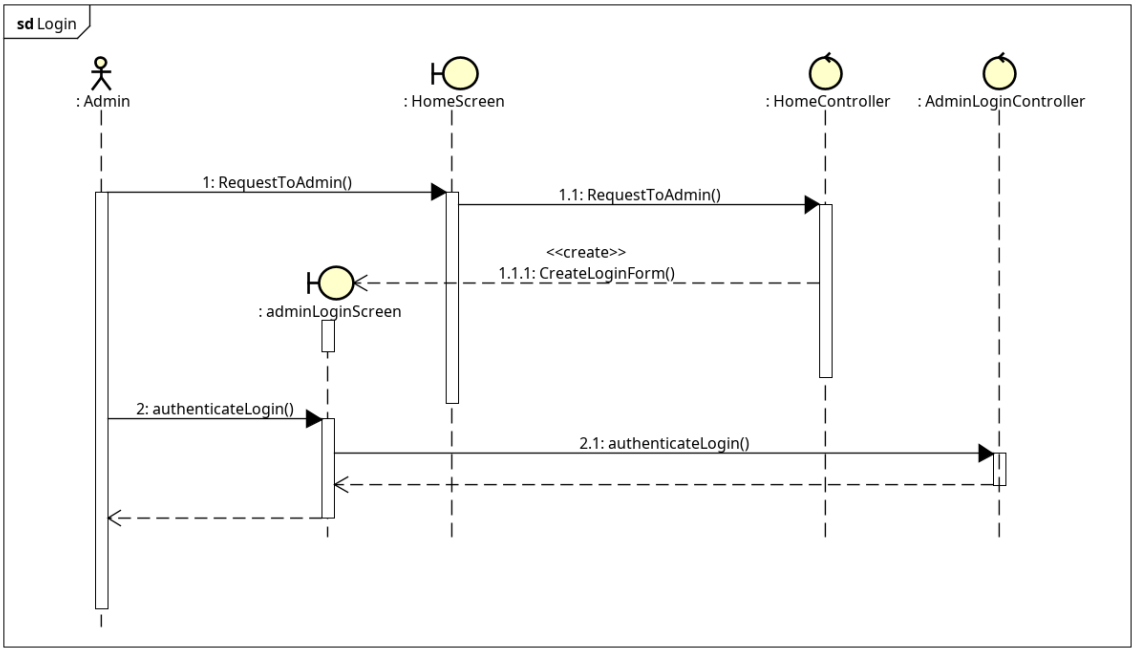
3.2.8. Place rush order



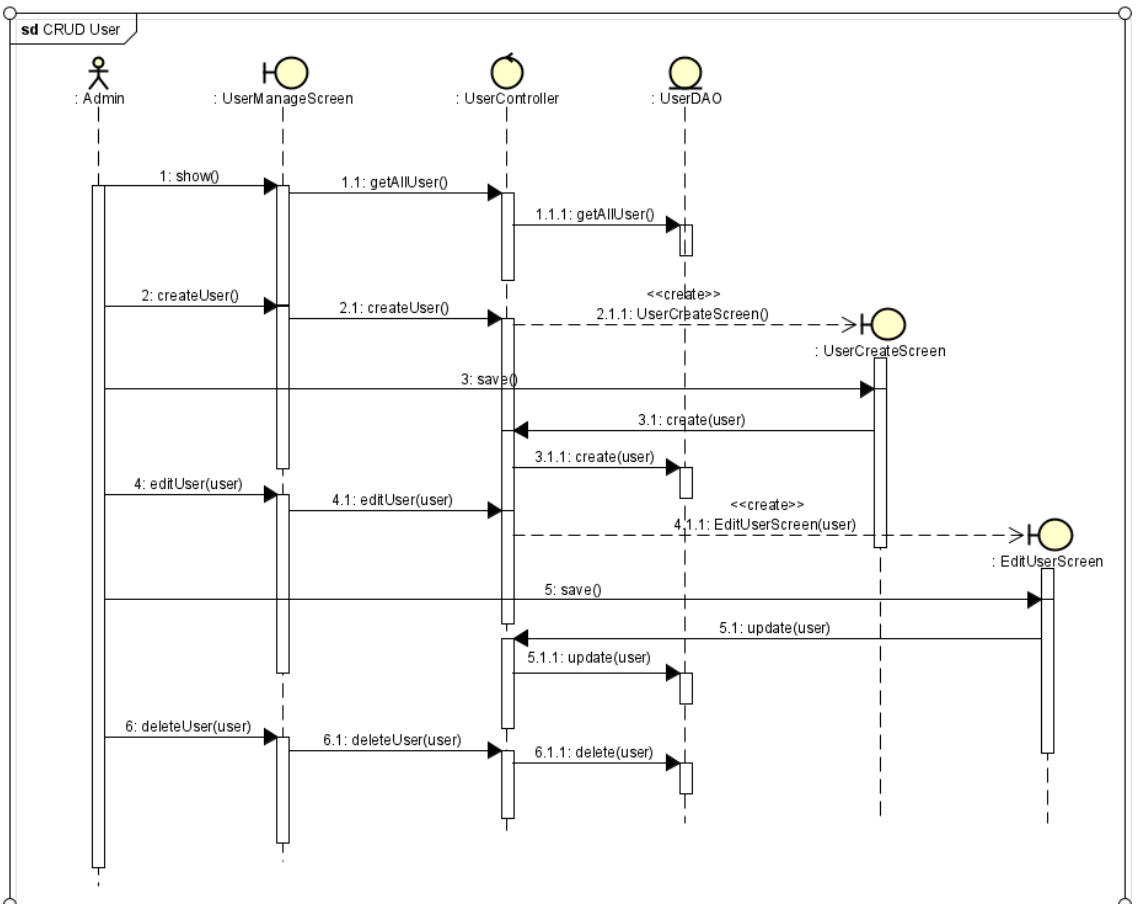
3.2.9. Pay order



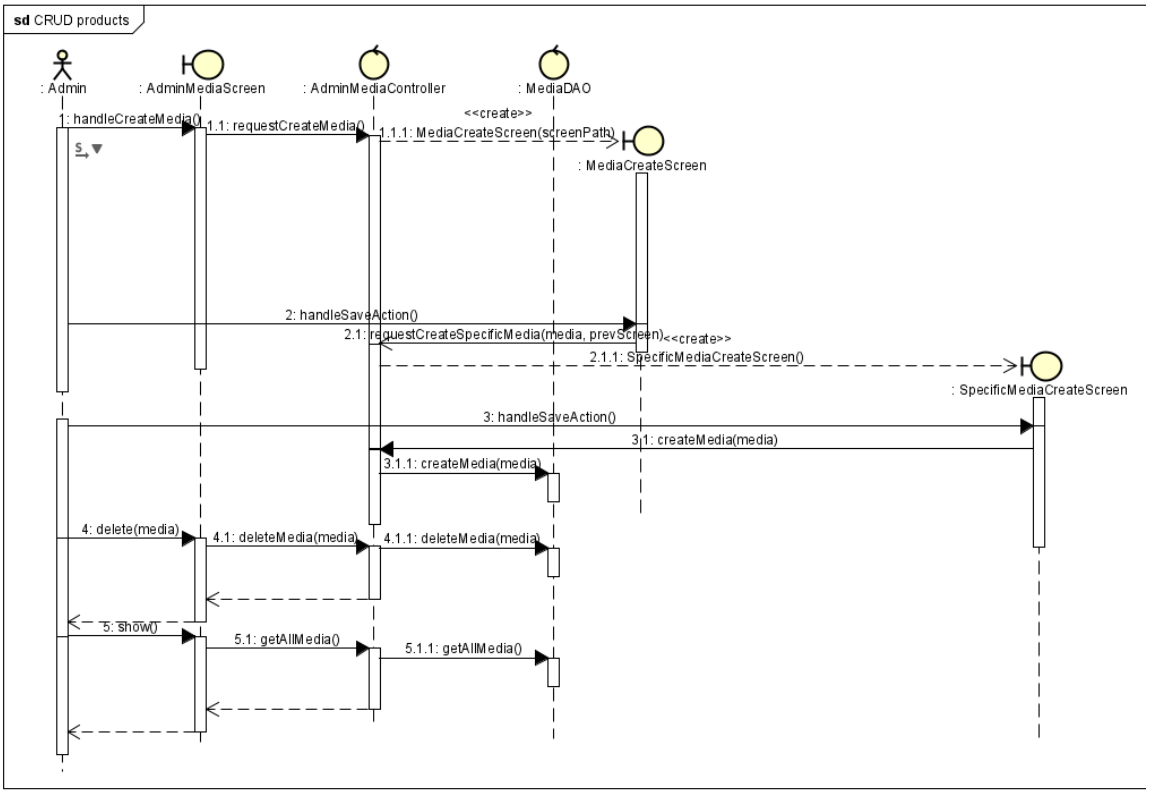
3.2.10. Log in



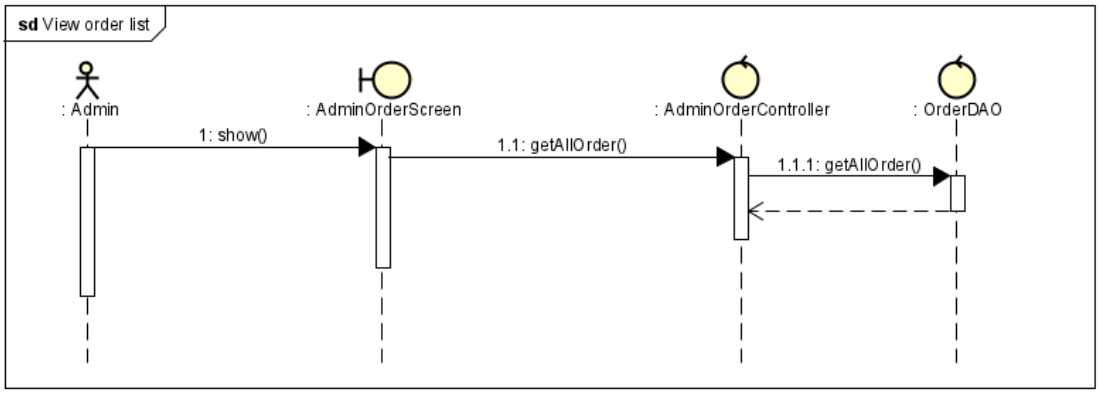
3.2.11. CRUD users



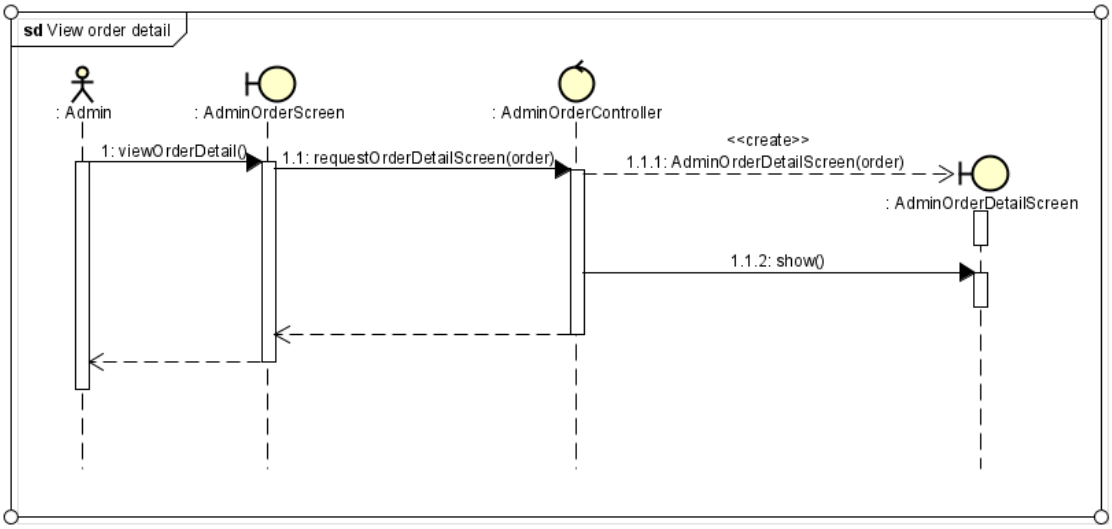
3.2.12. CRUD products



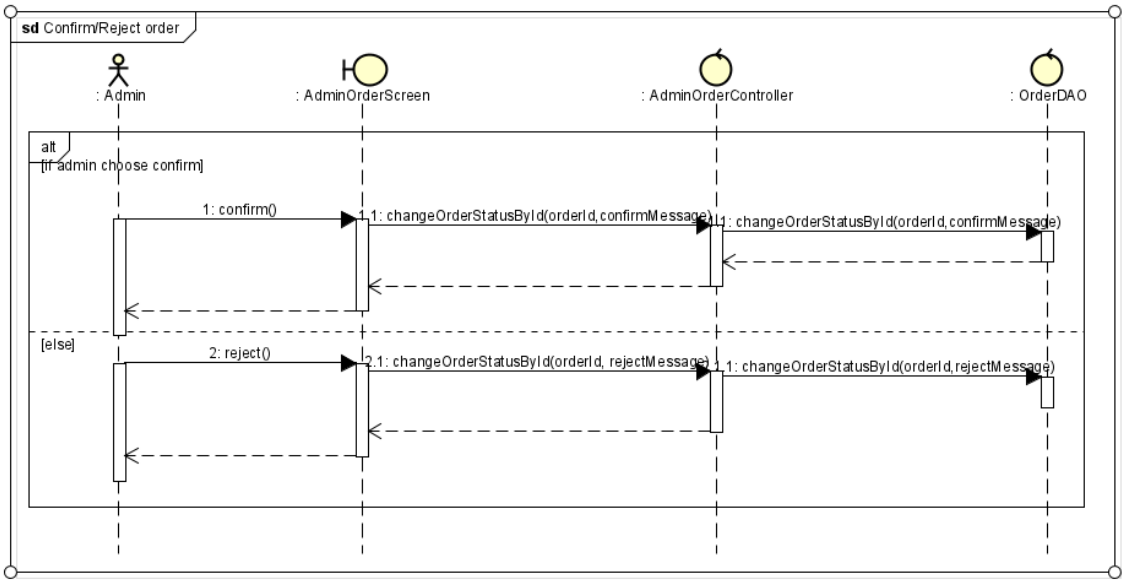
3.2.13. View order list



3.2.14. View order detail

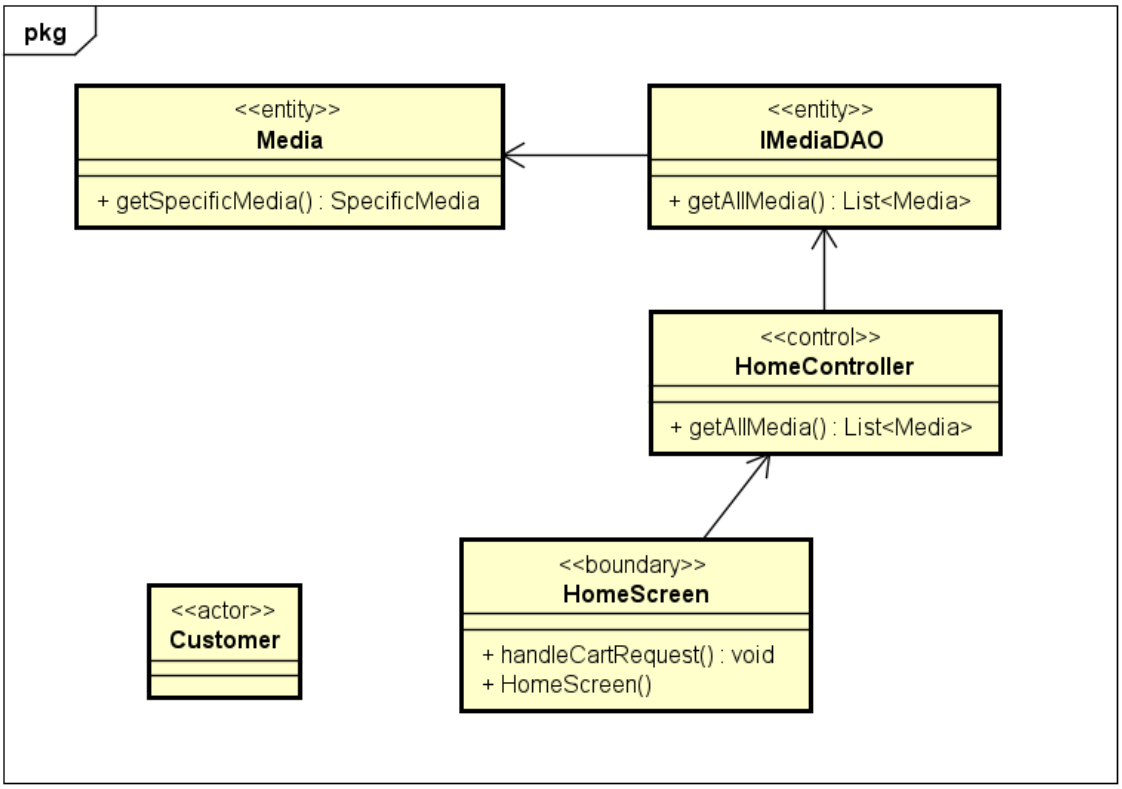


3.2.15. Approve/reject order

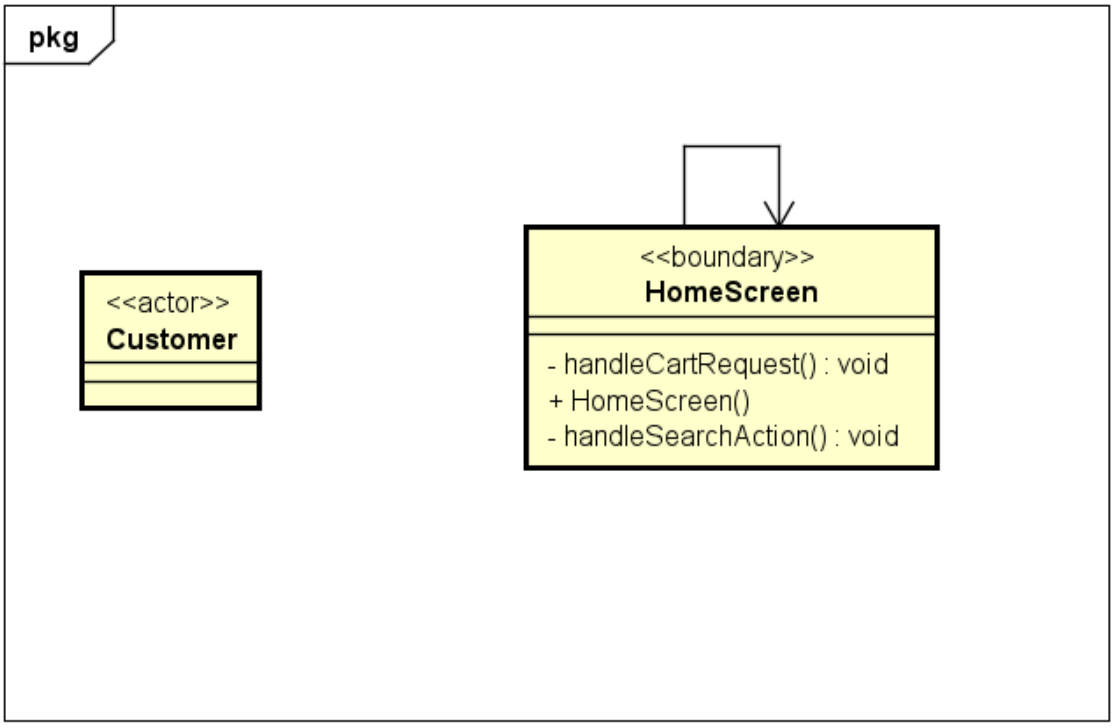


3.3 Analysis Class Diagrams

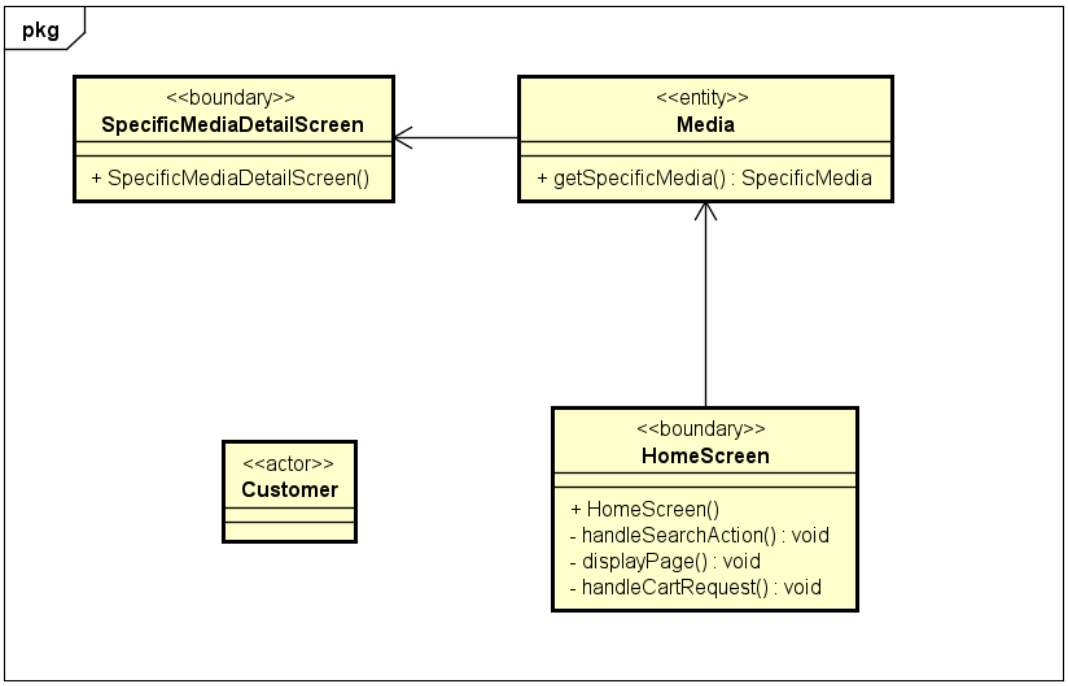
3.3.1 View Product in homepage



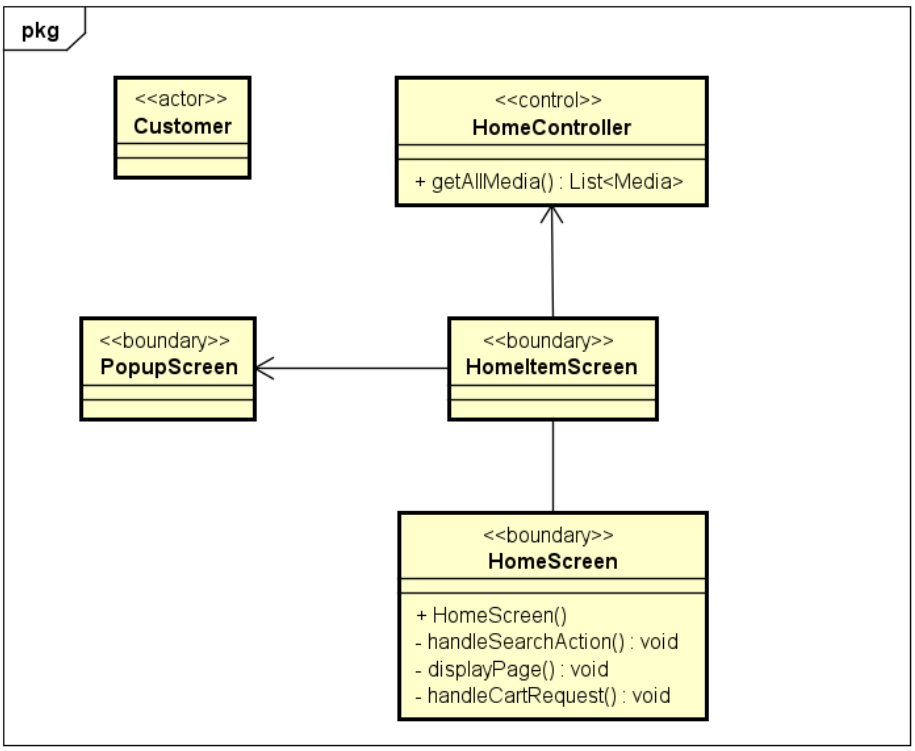
3.3.2 Search Product



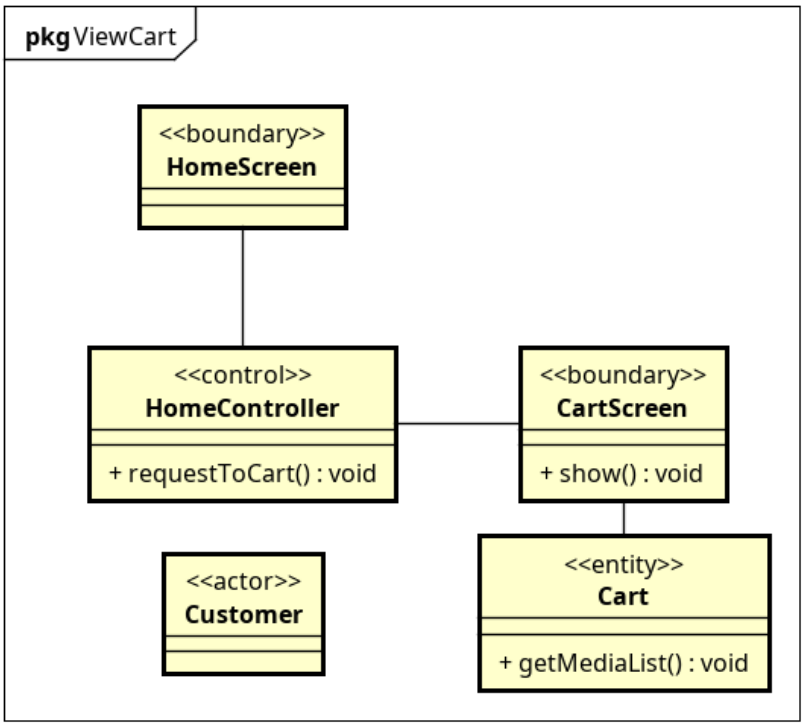
3.3.3 View Product details



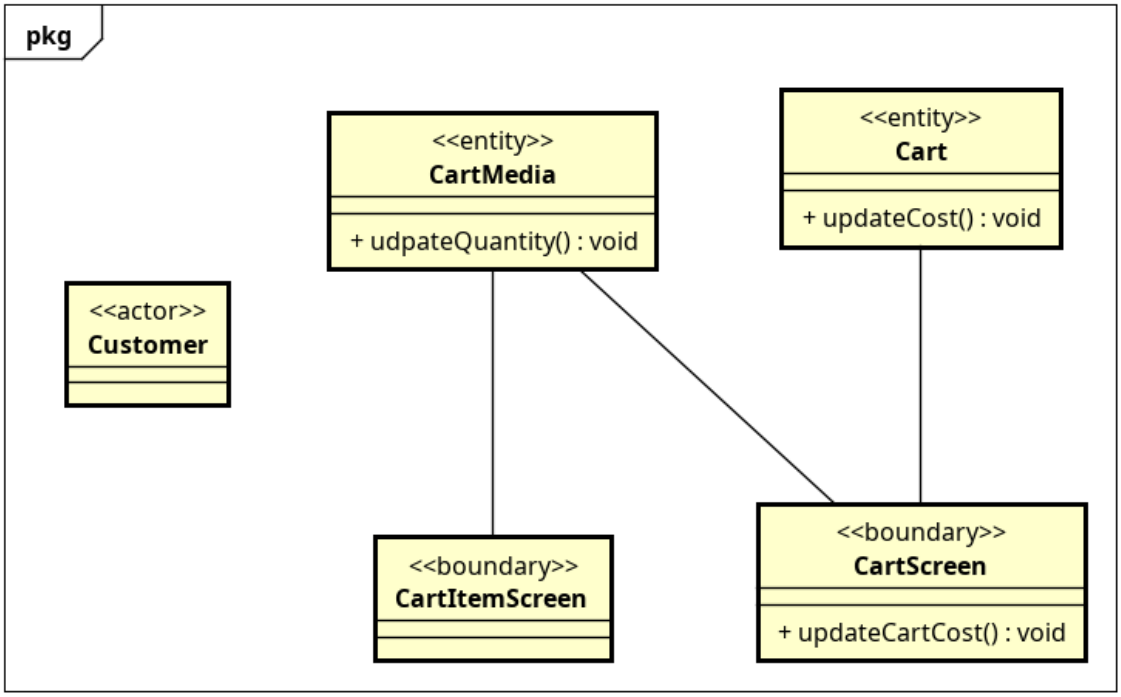
3.3.4 Add Product to Cart



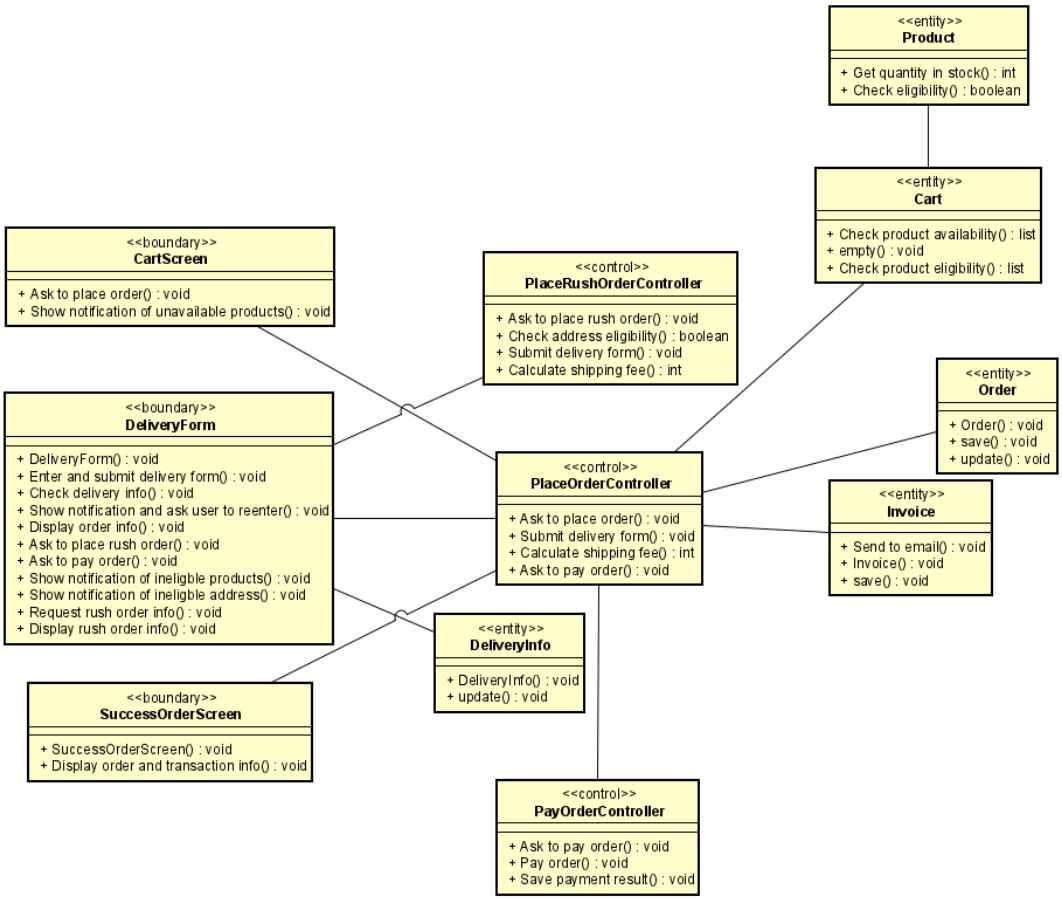
3.3.5 View Cart



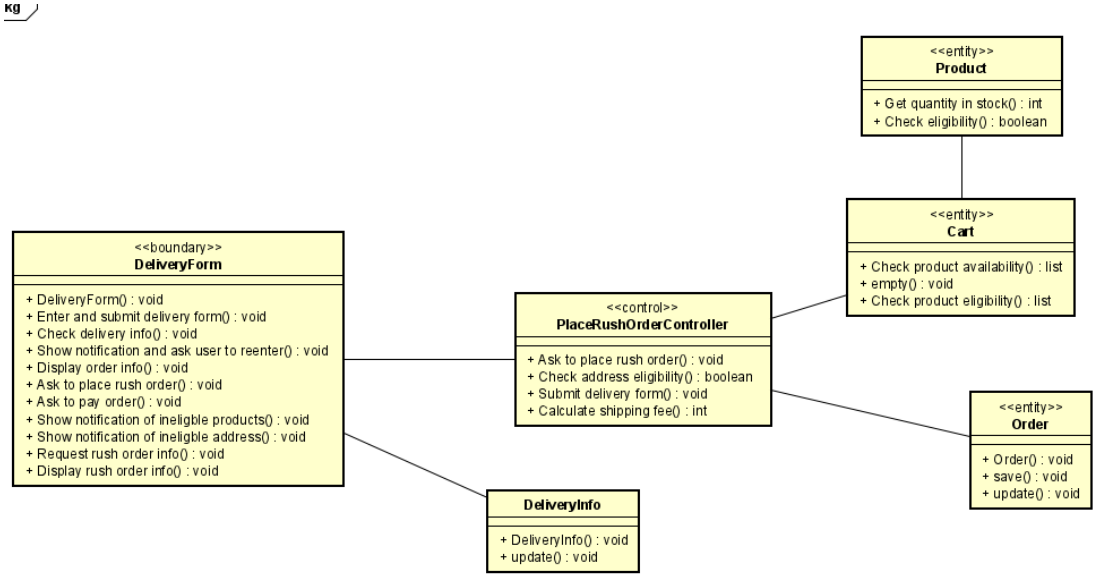
3.3.6 Update Cart



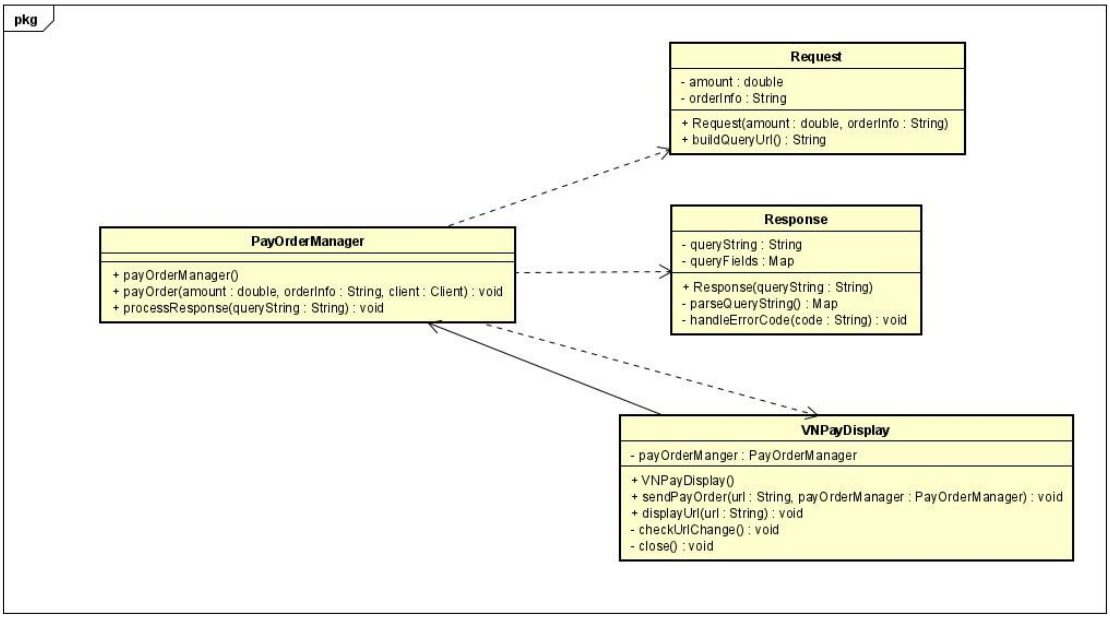
3.3.7 Place order



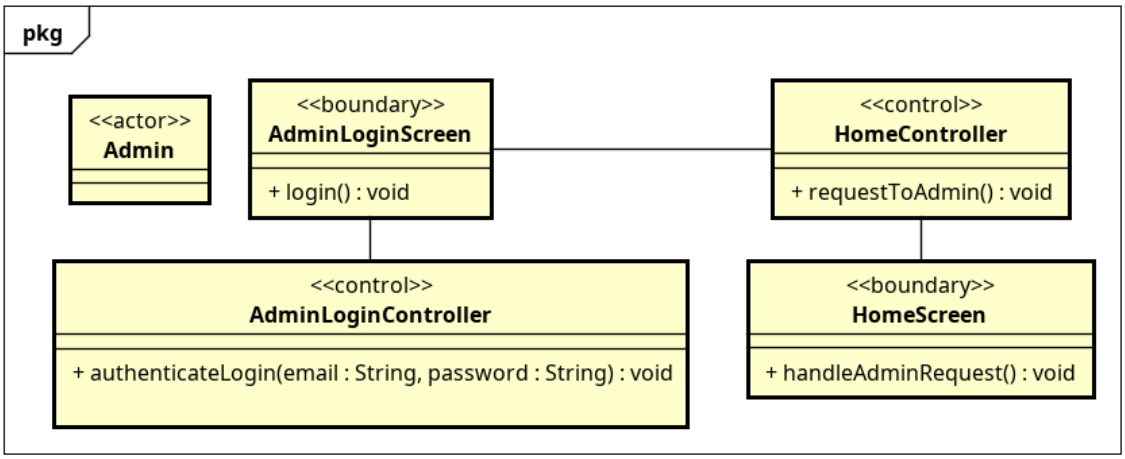
3.3.8 Place rush order



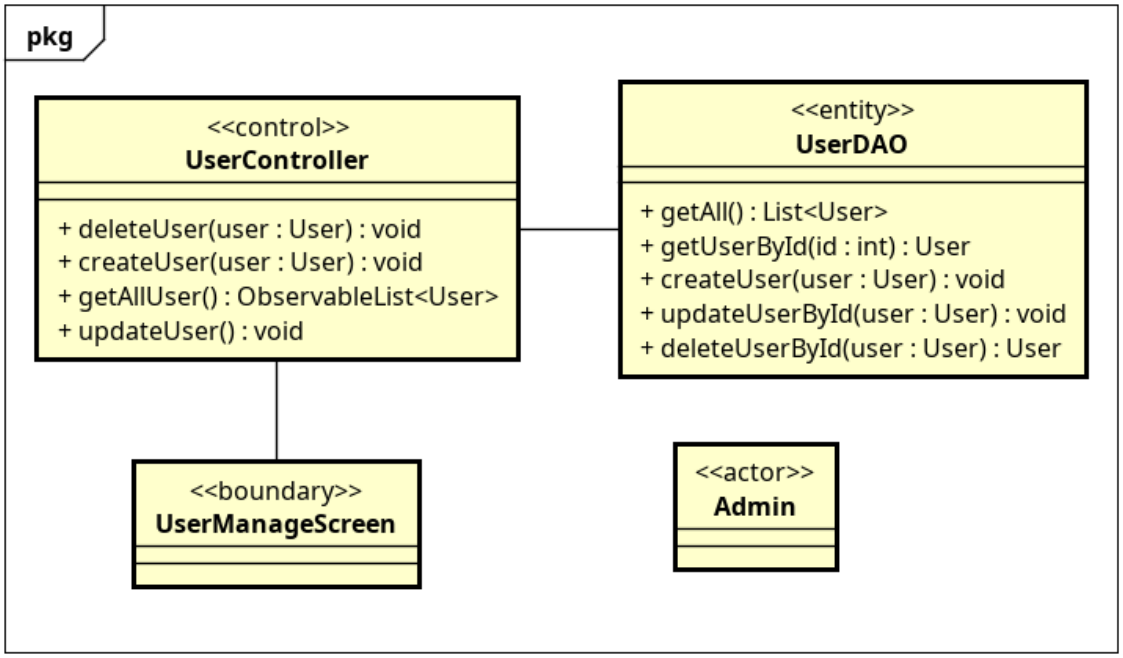
3.3.9 Pay order



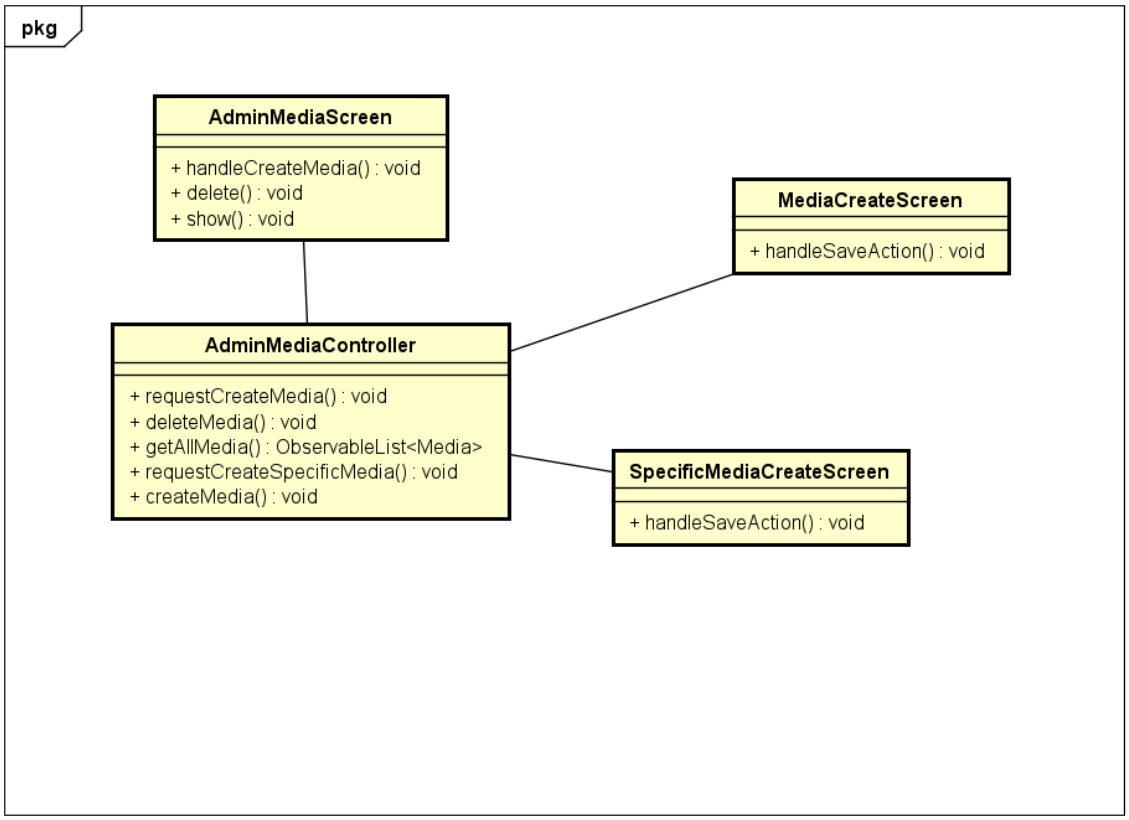
3.3.10 Log In



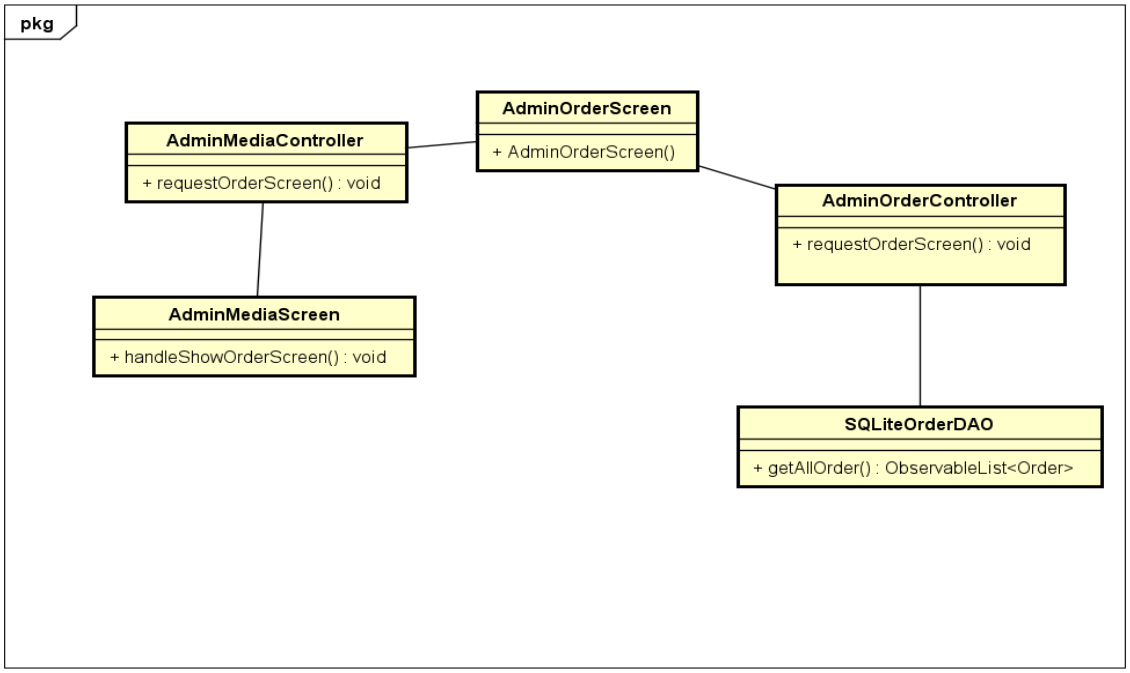
3.3.11 CRUD User



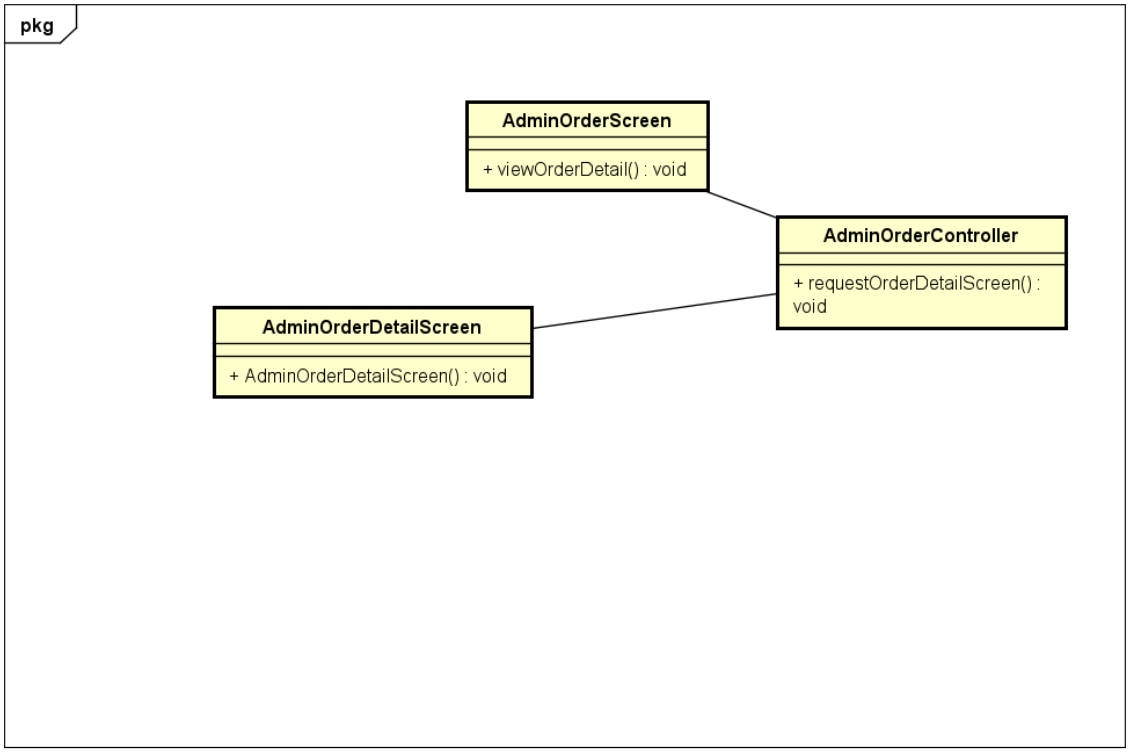
3.3.12 CRUD Products



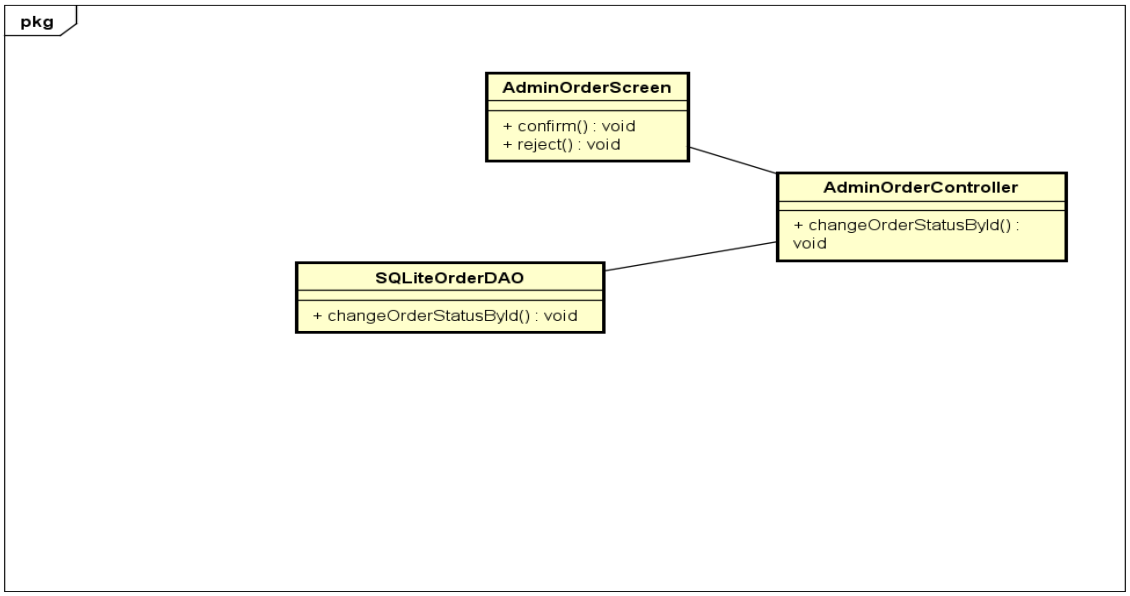
3.3.13 View order list



3.3.14 View order detail



3.3.15 Approve/Reject order



3.4 Unified Analysis Class Diagram

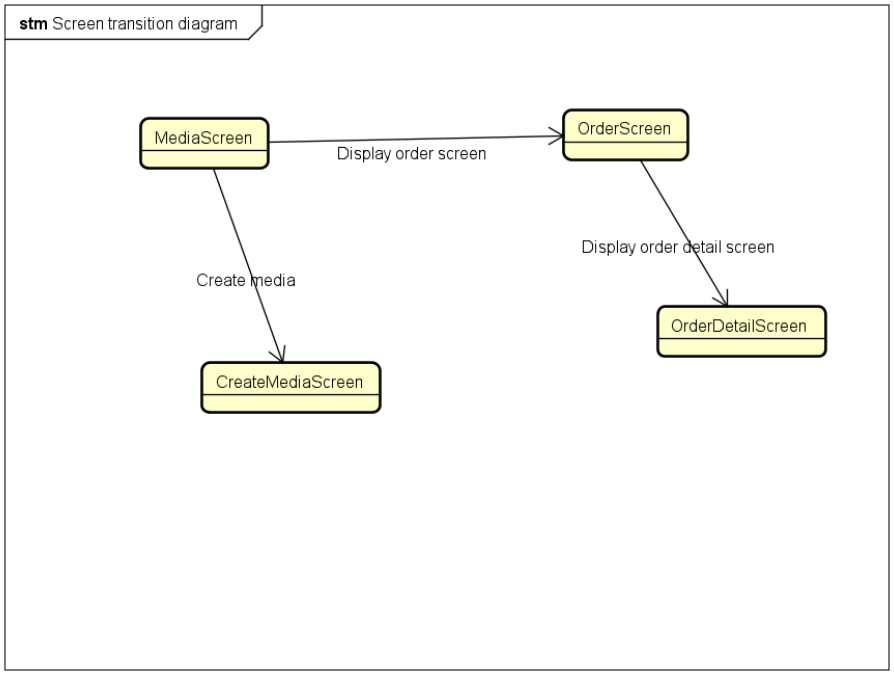
3.5 Security Software Architecture

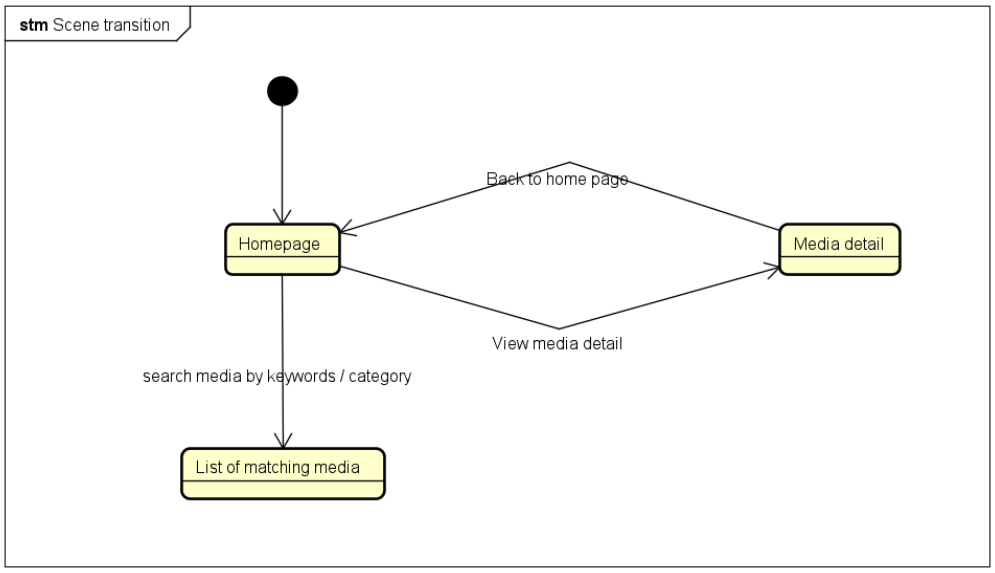
4 Detailed Design

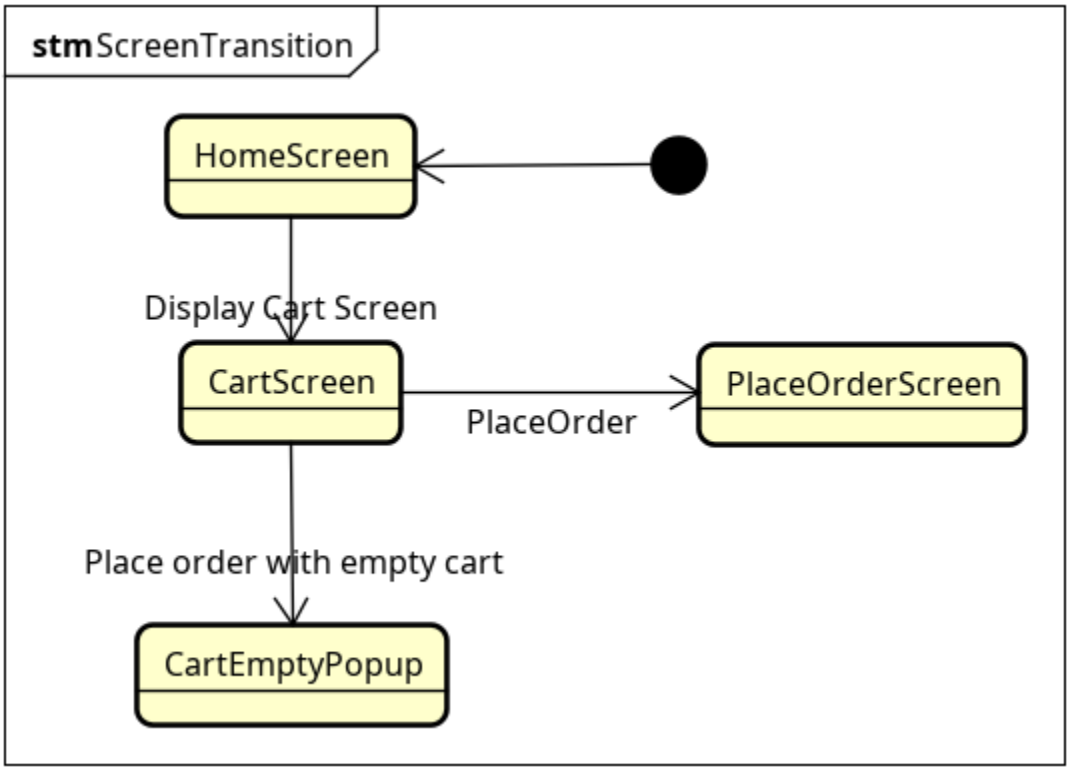
4.1 User Interface Design

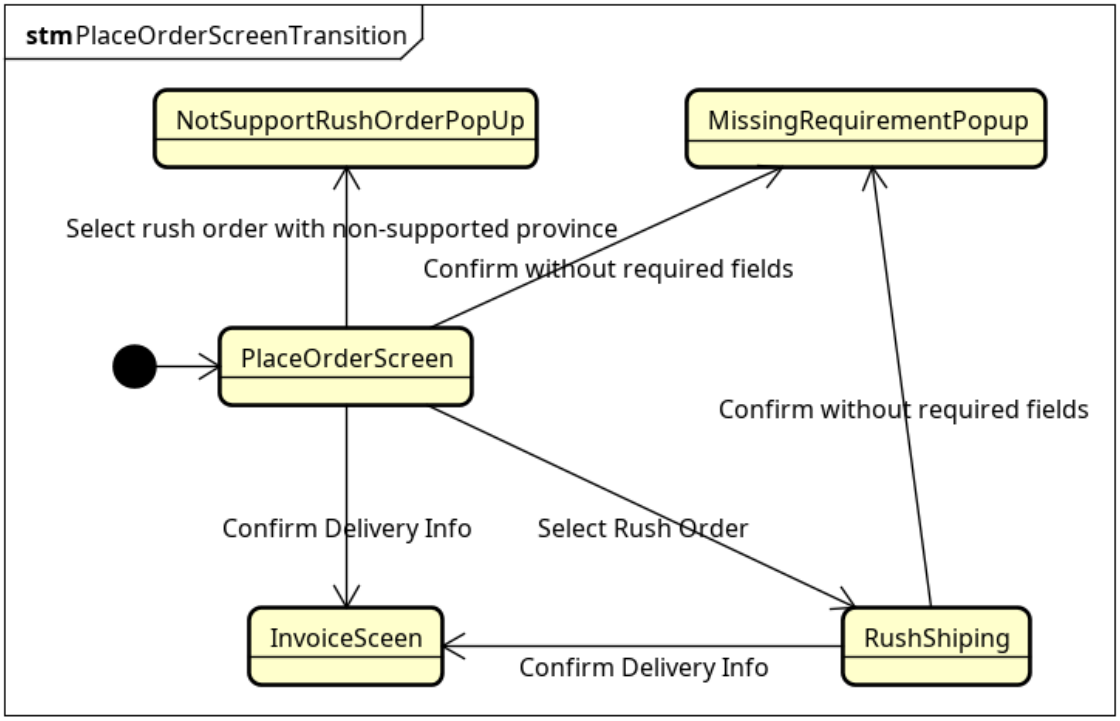
4.1.1 Screen Configuration Standardization

4.1.2 Screen Transition Diagrams


stm Screen transition diagram
MediaScreen
Display order screen
OrderScreen
Display order detail screen
Create media
OrderDetailScreen
CreateMediaScreen


stm Scene transition
Back to home page
Homepage
Media detail
View media detail
search media by keywords / category
List of matching media



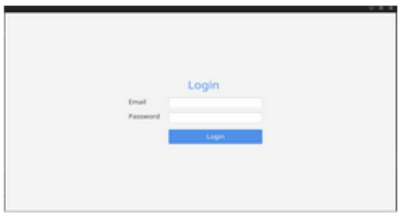


4.1.3 Screen Specifications

5.1.3.5 Cart Screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Cart Screen				Pham Trung Hieu
		Control	Operation	Function	
		Area for displaying products	initial	Display list of items in cart	
		Area for display cost	initial	Display calculated cost of items in cart	
		place order button	click	Place an order with items in cart	
		cart item spinner	click/input	update number of each item in cart	

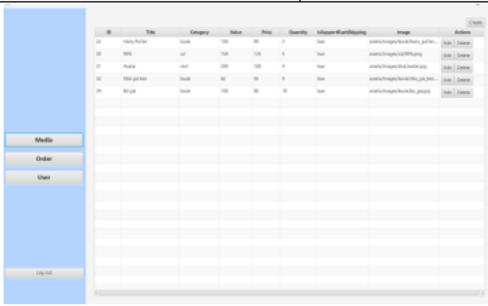
5.1.3.9 Login Screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Login Screen				Pham Trung Hieu
		Control	Operation	Function	
		Email input	enter text	Enter email	
		Password input	enter text	Enter password	
		Login button	click	Login user to system	

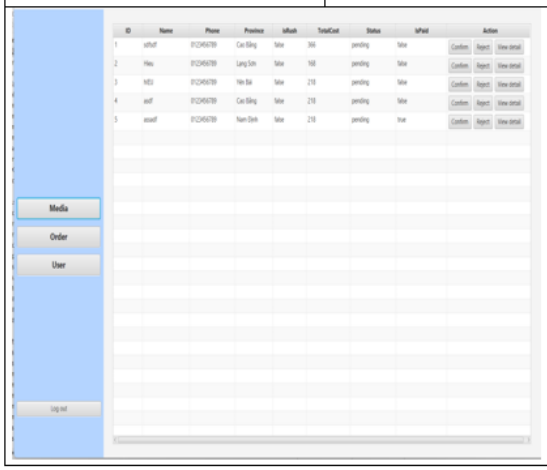
5.1.3.10 CRUD Users Screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Admin User Screen				Pham Trung Hieu
		Control	Operation	Function	
		Area for displaying users	initial	Display list of users	
		order button	click	Go to order screen	
		media button	click	Go to media screen	
		logout button	click	Log out admin account and back to home screen	
		user button	click	Go to current screen (user screen)	
		create button	click	Open create user screen	
		delete button	click	Delete current user	
		edit button	click	Open edit current user screen	

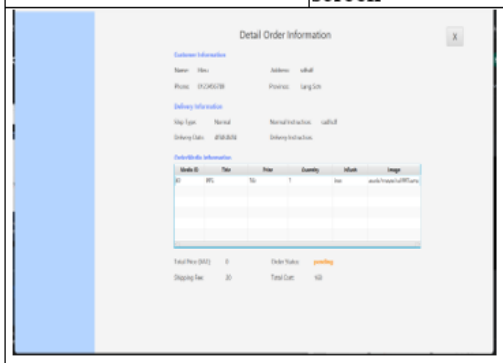
5.1.3.12 Products screen

AIMS Software		Date of creation	Approved by	Review by	Person in charge
Screen Specification	Product screen	5/4/2024			Nguyen Ha Hieu
		Control	Operation	Function	
		Area for displaying products	Initial	Display products	
		Order	Click	Go to order screen	
		User	Click	Go to user screen	
		Logout	Click	Logout	
		Create	Click	Go to create screen	

5.1.3.13 Order screen

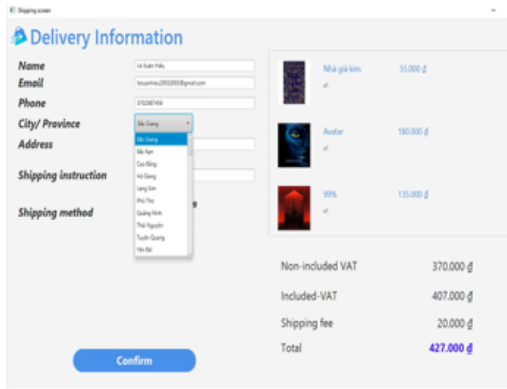
AIMS Software		Date of creation	Approved by	Review by	Person in charge
Screen Specification	Order screen	5/4/2024			Nguyen Ha Hieu
		Control	Operation	Function	
		Area for displaying orders	Initial	Display orders	
		Media	Click	Go to product_screen	
		User	Click	Go to user screen	
		Logout	Click	Logout	

5.1.3.14 Order detail screen

AIMS Software		Date of creation	Approved by	Review by	Person in charge
Screen Specification	Order detail screen	5/4/2024			Nguyen Ha Hieu
		Control	Operation	Function	
		Area for displaying order information	Initial	Display products	
		X	Click	Go back to order screen	

5.1.3.15 Shipping Screen

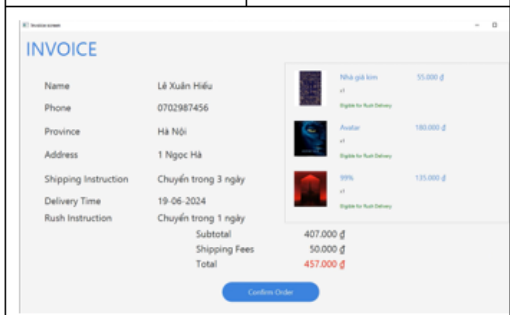
AIMS Software		Date of creation	Approved by	Reviewed by	Persion in charge
Screen specification	View shipping screen	18/6/2024			Nguyen Manh Hieu



Control	Operation	Function
Name input	Initial	Display the delivery information
Email input	Enter text	Enter email
Phone input	Enter text	Enter phone number
City/Province input	Choose list box	Choose province/city
Shipping method input	Choose tick box	Choose shipping method
Shipping instruction	Enter text	Enter shipping instruction
Confirm button	Click	Display the invoice screen
Area for display items in order	Initial	Display the media with the corresponding information

5.1.3.16 Invoice Screen

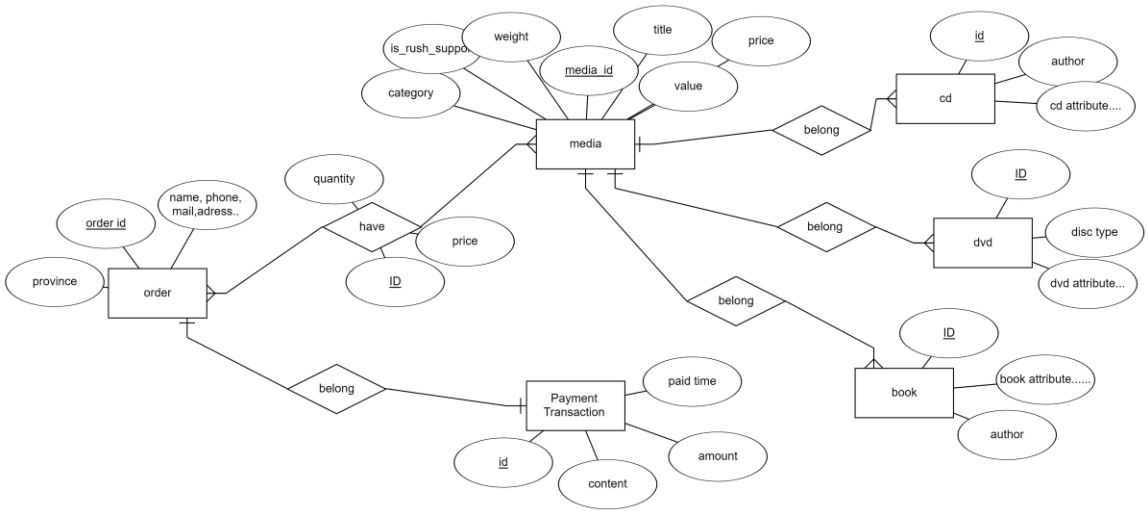
AIMS Software		Date of creation	Approved by	Reviewed by	Persion in charge
Screen specification	View invoice screen	18/6/2024			Nguyen Manh Hieu



Control	Operation	Function
Area to display delivery info	Initial	Display the delivery information
Area to display list item in order	Initial	Display the list item
Area to display total payment money	Initial	Display payment monney
Button confirm order	Click	Display the vnpay screen

4.2 Data Modeling

4.2.1 Conceptual Data Modeling



4.2.2 Database Design

4.2.2.1 Database Management System

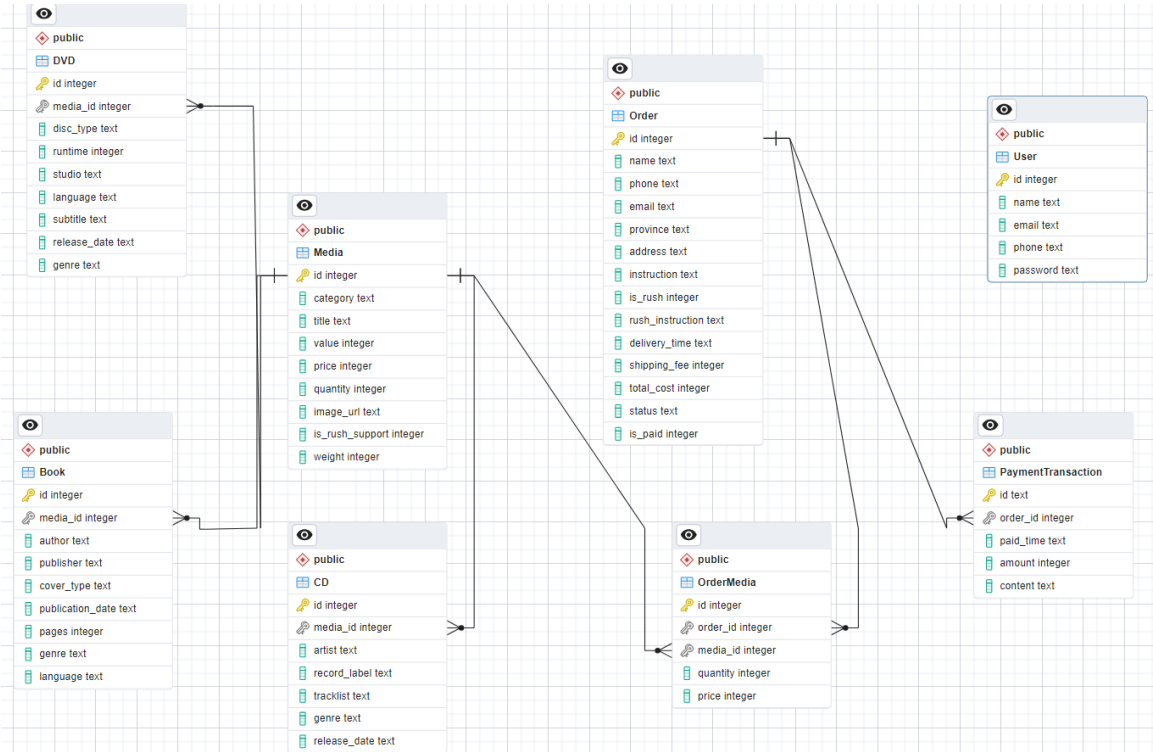
A Database Management System (DBMS) is software that interacts with end users, applications, and the database itself to capture and analyze data. The DBMS manages the data, the database engine, and the database schema, facilitating the processes of defining, creating, querying, updating, and administering databases.

A Database Management System (DBMS) is software that interacts with end users, applications, and the database itself to capture and analyze data. The DBMS manages the data, the database engine, and the database schema, facilitating the processes of defining, creating, querying, updating, and administering databases.

Some key Decisions of a DBMS:

- Data Storage and Management:
- Backup and Recovery
- Data Independence

4.2.2.2 Database Diagram



4.2.2.3 Database Detail Design

4.2.2.3.1 Physical data model

1. **Table User**

STT	PK	FK	Column Name	Data Type	Mandatory	Description
1	x		id	INT	Yes	ID, auto increment
2			name	VARCHAR (45)	Yes	User name
3			email	VARCHAR (45)	Yes	User email
4			password	VARCHAR (45)	Yes	User address

5			phone	VARCHAR (45)	Yes	User contact number

2. Table Media

STT	**PK**	**FK**	**Column Name**	**Data Type**	**Mandatory**	**Description**
1	x		id	INT	Yes	ID, auto increment
2			price	INT	Yes	Current price
3			category	VARCHAR(45)	Yes	Media category (story, pop, adventure)
4			value	INT	Yes	Product value
5			quantity	INT	Yes	Number of products
6			title	VARCHAR(45)	Yes	Product name
7			Image_url	VARCHAR(45)	No	Product image path
8			is_rush_support	INT	Yes	Rush (yes/no)
9			weight	INT	No	weight

3.Table CD

STT	PK	FK	Column Name	Data Type	Mandatory	Description
1	x		id	INT	Yes	ID, auto increment
2		X	media_id	INT	YES	ID from MEDIA table
3			artist	VARCHAR(45)	Yes	Artist's name
4			recordLabel	VARCHAR(45)	Yes	Record Label
5			musicType	VARCHAR(45)	Yes	Music genres
6			releasedDate	DATETIME	No	Release Date

4.Table Book

STT	PK	FK	Column Name	Data Type	Mandatory	Description
1	x		id	INT	Yes	ID, auto increment
2		x	media_id	INT	YES	ID from Media table
3			author	VARCHAR(45)	Yes	Author's name
4			coverType	VARCHAR(45)	Yes	Cover Type

5			numberOfPages	INT	Yes	Page number
6			language	VARCHAR(45)	Yes	Language
7			genre	VARCHAR(45)	Yes	Type of Book
8			publisher	VARCHAR(45)	Yes	Publishing house
9			publishDate	DATETIME	Yes	Date of publishing

5. Table DVD

STT	**PK**	**FK**	**Column Name**	**Data Type**	**Mandatory**	**Description**
1	x		id	INT	Yes	ID, auto increment
2		x	media_id	INT	Yes	ID from Media table
3			runtime	INT	Yes	time of DVD to
4			Disc_type	VARCHAR(45)	Yes	Disc Type
5			subtitle	VARCHAR(45)	Yes	Subtitles

6			studio	VARCHAR(45)	Yes	Manufacturer
7			language	VARCHAR(45)	Yes	Language
8			genre	VARCHAR(45)	Yes	Type of DVD
9			Released_date	DATETIME	No	Release Date

6. Table OrderMedia

STT	PK	FK	Column Name	Data Type	Mandatory	Description
1	x		id	INT	Yes	ID
2		x	Order_id	INT	Yes	ID of order
3		X	Media_id	INT	Yes	ID from media table
4			Quantity	INT	Yes	Number
5			Price	INT	Yes	Number

7. Table Payment Transaction

STT	PK	FK	Column Name	Data Type	Mandatory	Description
1	x		id	INT	Yes	ID
2		x	Order_id	INT	Yes	ID of order
3			Paid_time	VARCHAR(45)	Yes	Time pay order

4			Amount	INT	Yes	Number
5			Content	VARCHAR(45)	Yes	Content about Order_id

4.2.2.3.2 Database Script from Logical Data Model -- Table: Media

```
CREATE TABLE IF NOT EXISTS "Media" (
"id"	INTEGER,
"category"	TEXT NOT NULL,
"title"	TEXT NOT NULL,
"value"INTEGER NOT NULL,
"price" INTEGER NOT NULL,
"quantity"	INTEGER NOT NULL,
"image_url"	TEXT NOT NULL,
"is_rush_support"	INTEGER NOT NULL,
"weight"	INTEGER DEFAULT 0,
PRIMARY KEY("id" AUTOINCREMENT)
);

-- Table: DVD
CREATE TABLE IF NOT EXISTS "DVD" (
"id"	INTEGER,
"media_id"	INTEGER NOT NULL,
"disc_type"	TEXT NOT NULL,
"runtime"	INTEGER NOT NULL,
"studio"	TEXT NOT NULL,
"language"	TEXT NOT NULL,
```

```
"subtitle"        TEXT,
"release_date" TEXT,
"genre"           TEXT,
FOREIGN KEY("media_id") REFERENCES "Media"("id"),
PRIMARY KEY("id" AUTOINCREMENT)
);

-- Table: CD
CREATE TABLE IF NOT EXISTS "CD" (
"id"     INTEGER,
"media_id"     INTEGER NOT NULL,
"artist" TEXT NOT NULL,
"record_label" TEXT NOT NULL,
"tracklist"       TEXT NOT NULL,
"genre"           TEXT NOT NULL,
"release_date" TEXT,
FOREIGN KEY("media_id") REFERENCES "Media"("id"),
PRIMARY KEY("id" AUTOINCREMENT)
);

-- Table: Book
CREATE TABLE IF NOT EXISTS "Book" (
"id"     INTEGER,
"media_id"     INTEGER NOT NULL,
"author"          TEXT NOT NULL,
"publisher"     TEXT NOT NULL,
"cover_type"   TEXT NOT NULL,
"publication_date"      TEXT NOT NULL,
"pages"           INTEGER,
```

```
"genre"         TEXT,
"language"      TEXT,
FOREIGN KEY("media_id") REFERENCES "Media"("id"),
PRIMARY KEY("id" AUTOINCREMENT)
);
CREATE TABLE IF NOT EXISTS "Order" (
"id"     INTEGER,
"name"TEXT NOT NULL,
"phone"         TEXT NOT NULL,
"email"         TEXT NOT NULL,
"province"      TEXT NOT NULL,
"address"       TEXT NOT NULL,
"instruction"   TEXT NOT NULL,
"is_rush"       INTEGER NOT NULL,
"rush_instruction"      TEXT,
"delivery_time"         TEXT,
"shipping_fee"INTEGER NOT NULL,
"total_cost"    INTEGER NOT NULL,
"status"        TEXT NOT NULL DEFAULT 'pending',
"is_paid"       INTEGER NOT NULL DEFAULT 0,
PRIMARY KEY("id" AUTOINCREMENT)
);
CREATE TABLE IF NOT EXISTS "OrderMedia" (
"id"     INTEGER,
"order_id"      INTEGER NOT NULL,
"media_id"      INTEGER NOT NULL,
"quantity"      INTEGER NOT NULL,
"price" INTEGER NOT NULL,
FOREIGN KEY("order_id") REFERENCES "Order"("id"),
```

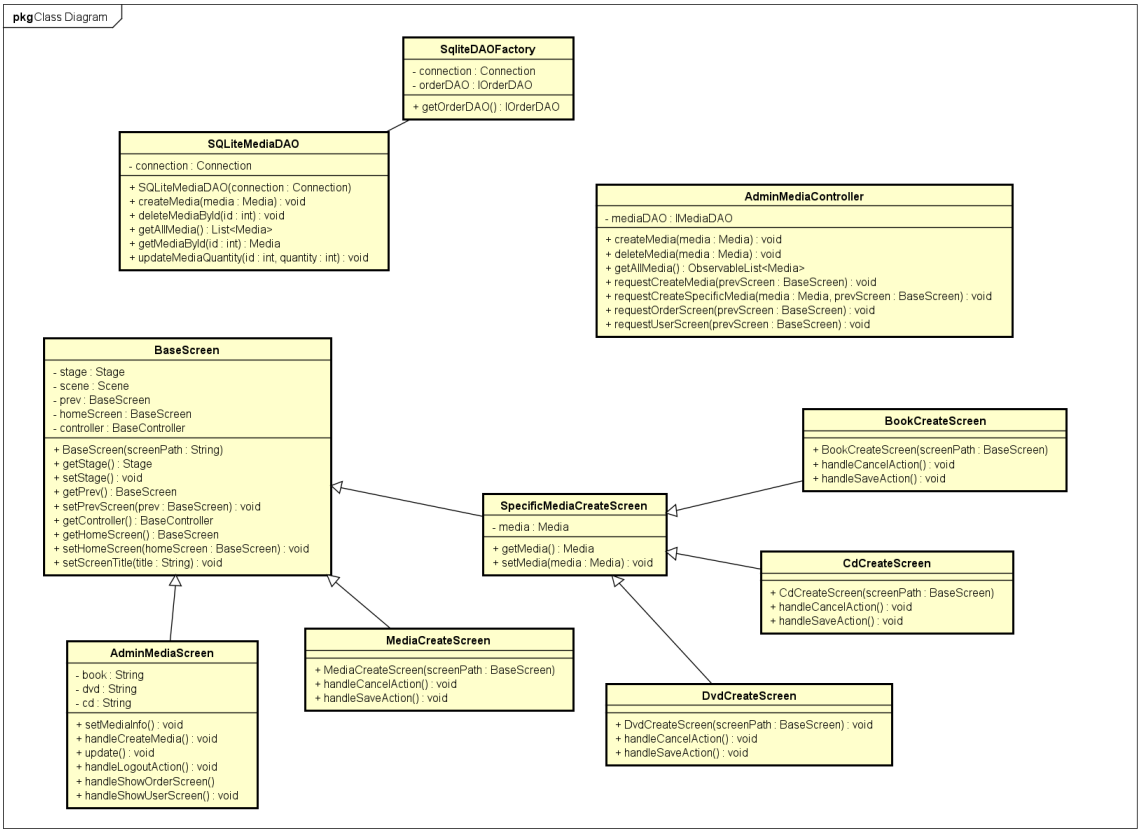
```
FOREIGN KEY("media_id") REFERENCES "Media"("id"),
PRIMARY KEY("id" AUTOINCREMENT)
);
-- Table: PaymentTransaction
CREATE TABLE IF NOT EXISTS "PaymentTransaction" (
"id"     TEXT,
"order_id"      INTEGER NOT NULL,
"paid_time"     TEXT NOT NULL,
"amount"        INTEGER NOT NULL,
"content"       TEXT NOT NULL,
PRIMARY KEY("id"),
FOREIGN KEY("order_id") REFERENCES "Order"("id")
);
-- Table: User
CREATE TABLE IF NOT EXISTS "User" (
"id"     INTEGER,
"name"TEXT NOT NULL,
"email"         TEXT NOT NULL,
"phone"         TEXT NOT NULL,
"password"      TEXT,
PRIMARY KEY("id")
);
COMMIT TRANSACTION;
PRAGMA foreign_keys = on;
```

4.3 Class Design

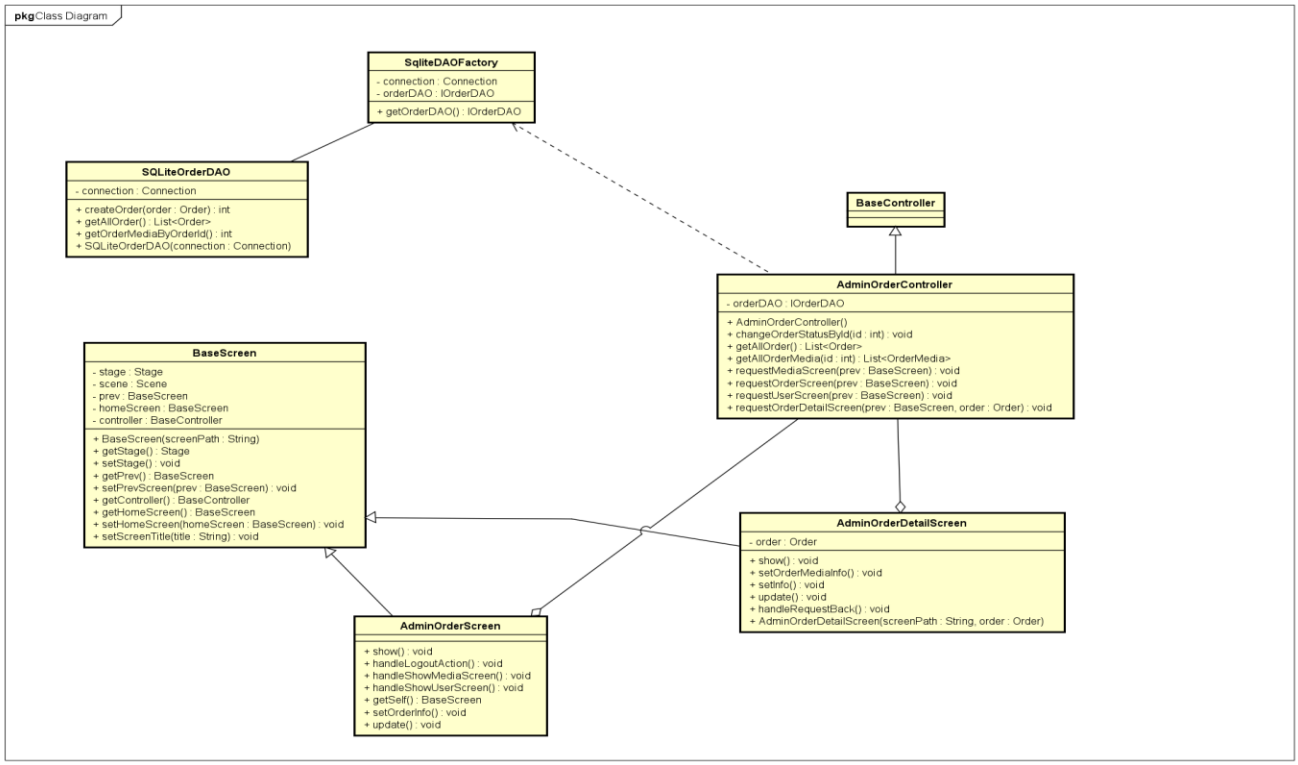
4.3.1 General Class Diagram

4.3.2 Class Diagrams

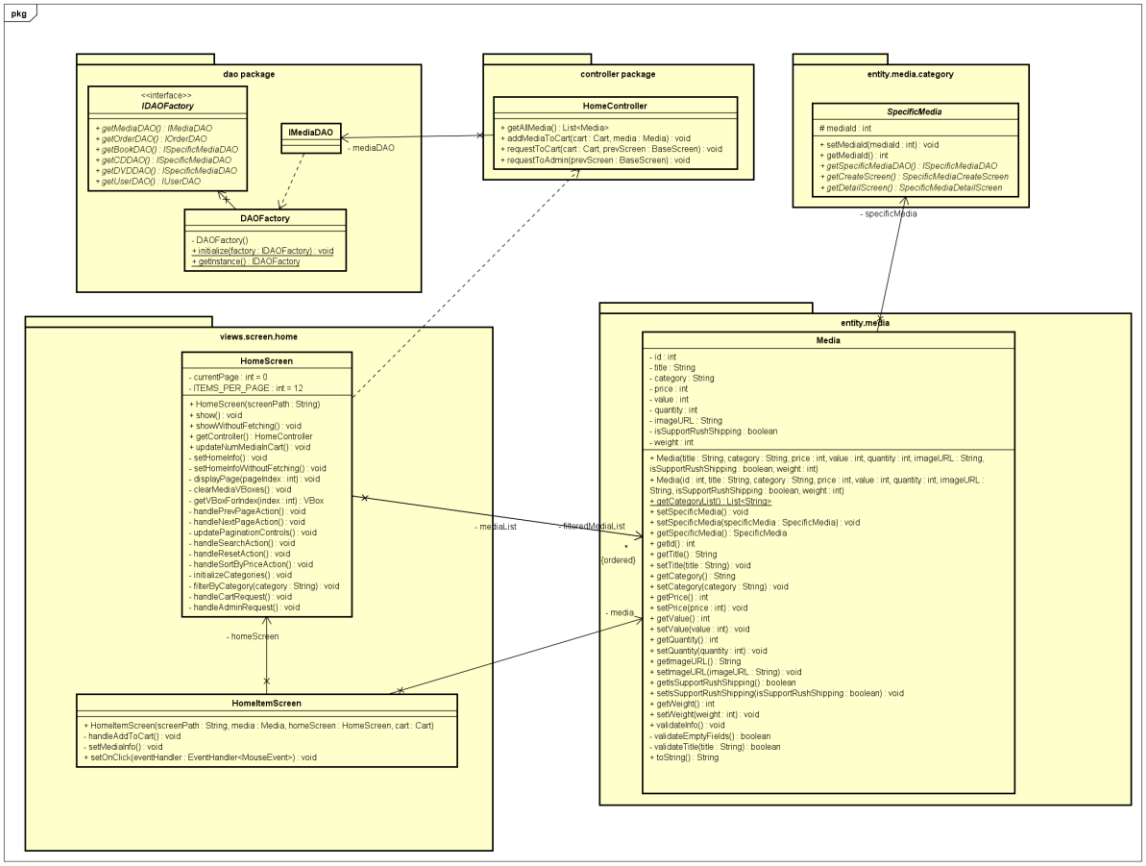
4.3.2.1 Class Diagram for Media Management



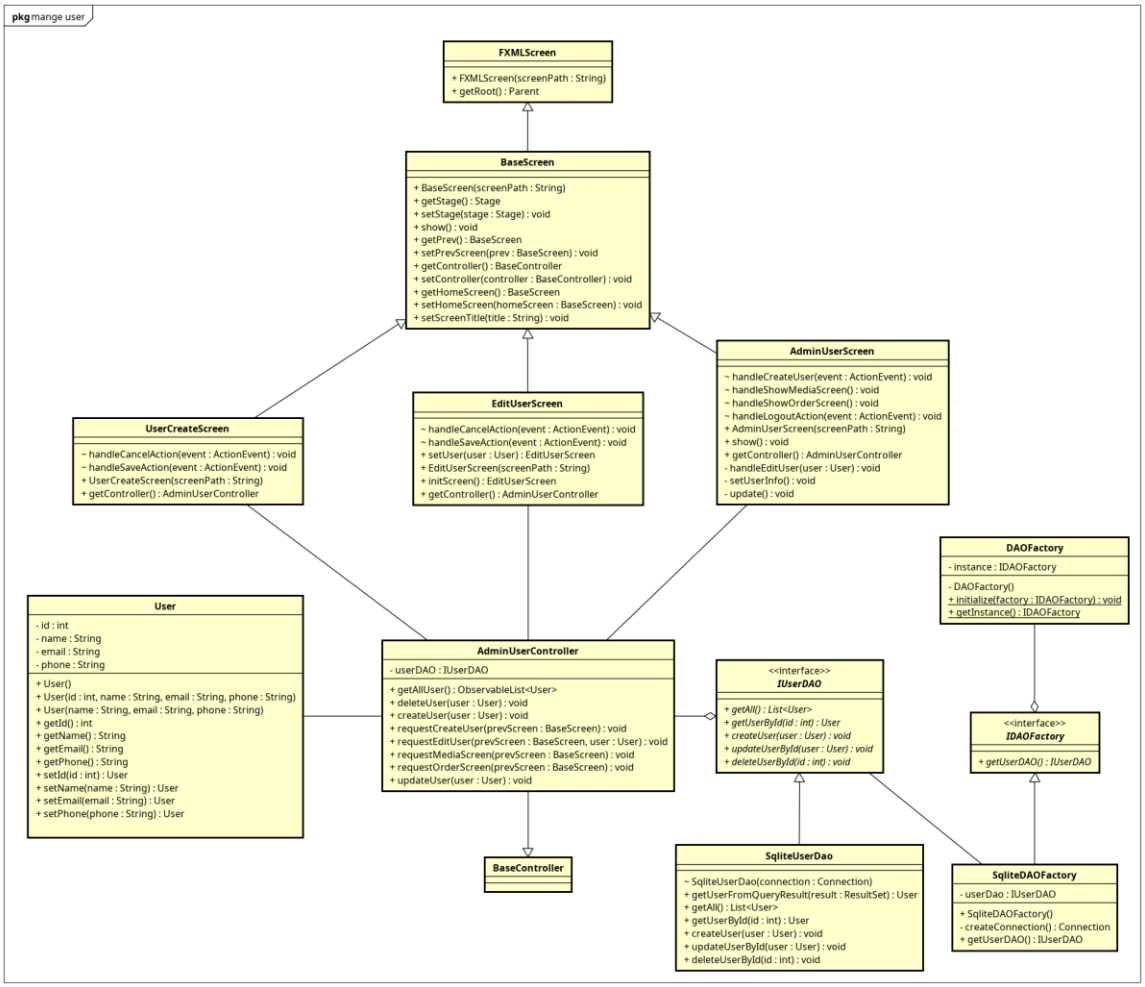
4.3.2.2 Class Diagram for Order management



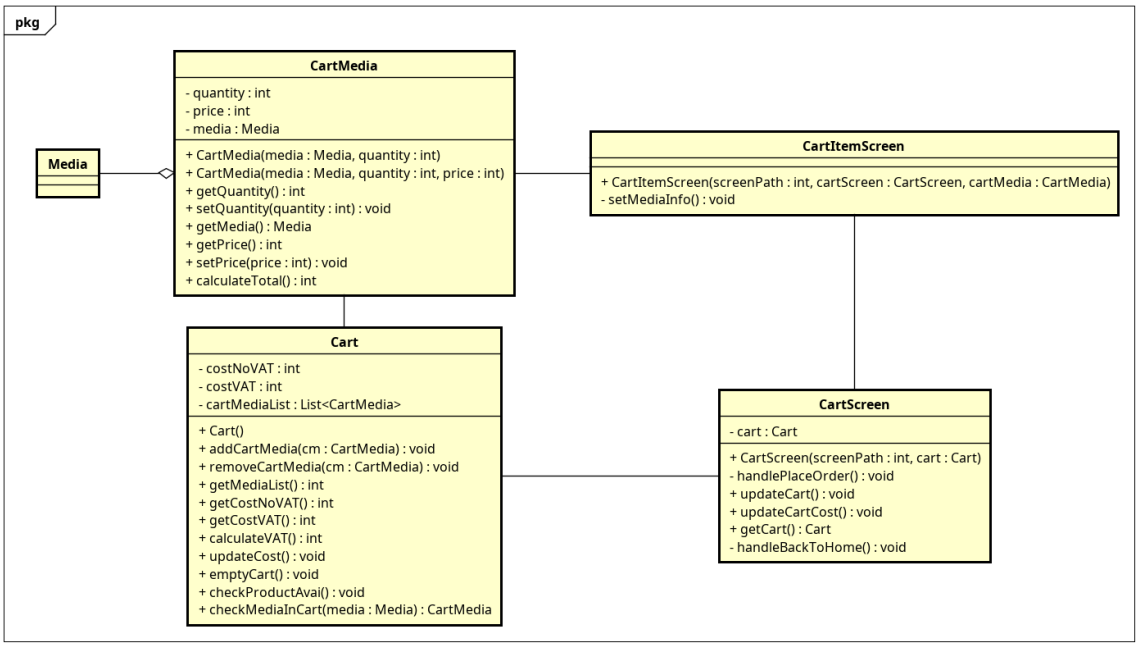
4.3.2.3 Class Diagram for Home Page



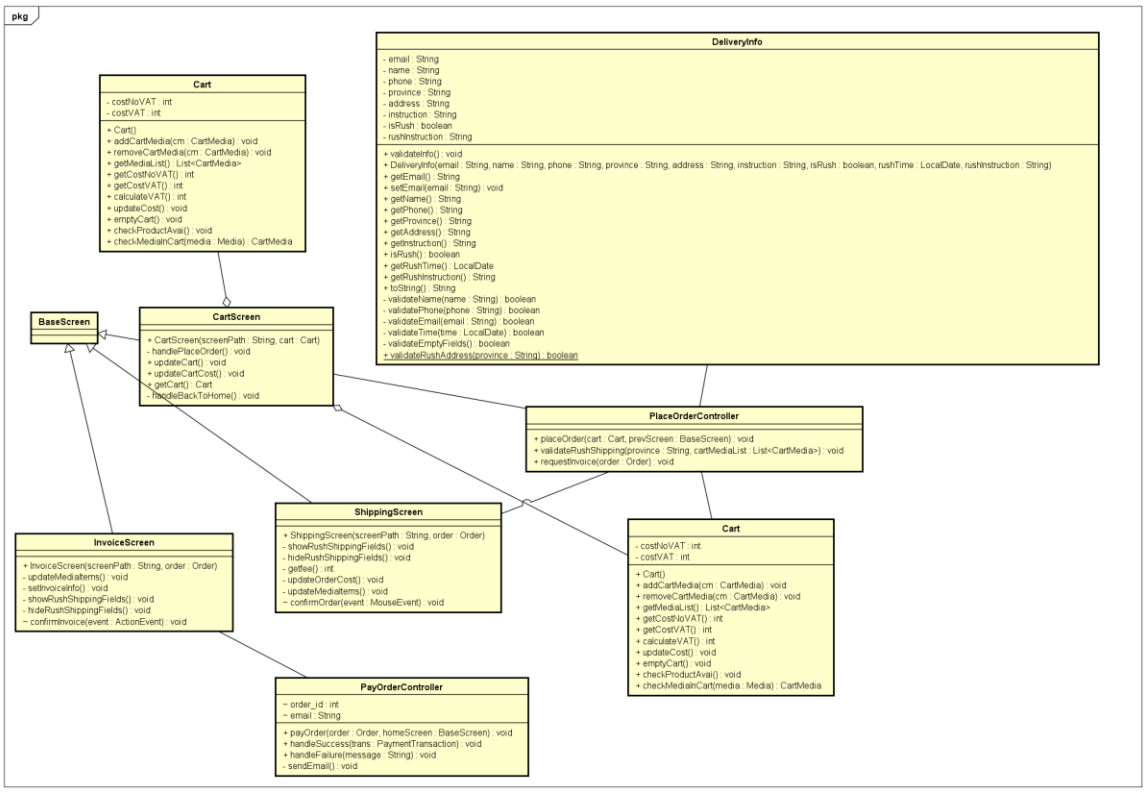
4.3.2.4 Class Diagram for User management



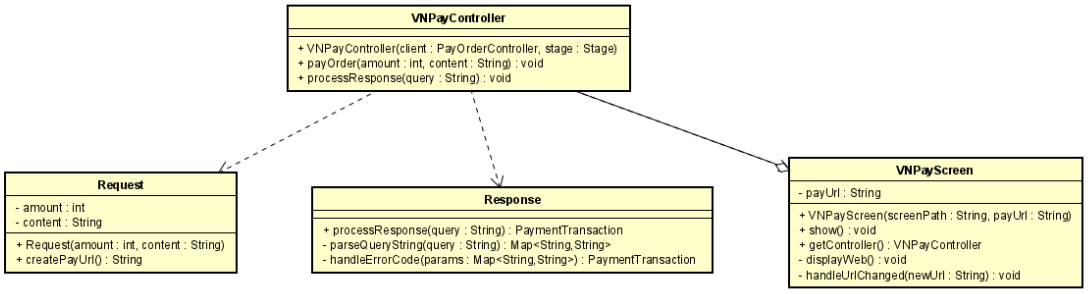
4.3.2.5 Class Diagram for Cart



4.3.2.6 Class Diagram for Place order



4.3.2.7 Class Diagram for Subsystem



4.3.3 Class Design

5 Design Considerations

5.1 Goals and Guidelines

- **Goals**

1.Performance Optimization

Prioritize speed and responsiveness over memory usage. Fast load times and quick responses are critical for enhancing the user experience, especially in an e-commerce application where users expect to browse and purchase items efficiently.

2.Scalability

Design the system to handle increasing loads of data and traffic without significant performance degradation. The system should accommodate growth in user base and data volume without requiring a complete redesign, ensuring long-term viability and cost-effectiveness.

3.Security and Data Privacy

Implement robust security measures to protect user data and ensure compliance with data privacy regulations. Protecting user data builds trust and ensures compliance with legal requirements like GDPR and CCPA, which is essential for maintaining the application's reputation and legal standing.

- **Guidelines**

1.Follow established coding standards such as naming conventions, code structuring, and documentation practices (we use Java Coding Conventions). The reason is that Consistent coding practices improve code readability, maintainability, and reduce the likelihood of errors, facilitating easier collaboration among us.

2.We follow the MVC coding struction . This makes our project clearly between the user interface and logic operations. With the MVC model, our work has been divided easily.

3.Modularization is separated clearly so that the system where each module is responsible for a specific functionality (e.g., user management, product catalog, order processing, payment management).

5.2 Coupling and cohesion

5.3 Design Principles

Our design is quite follow the SOLID principles .

•Single responsibility principle: we divided our project into the separated module, each contain the separated class that do the specific task. If there are new changing requirements in the future, the extension will be more easily apply on this system. Take the entity, for example, this package refers to the entity's definition in the database. Each class in the entity package defines one object in the database. The specific operations that can do by this object is contained in its class. It means that in the future, if there are any changes like adding some operation to the entity, we only need to create a new implementation method for this object. This ensures the single responsibility principle.

•Open close responsibility principle: the operation of querying to the database is implemented as data access object. We use DAO design pattern, so that all the actor operation is implemented in DAO object that follow the action interface. The reason for doing this design is that in the future, maybe we want to use another database management system, this change can be met by creating a new DAO object that implements the actor interface without modifying the code. This way helps us ensure the open close principle.

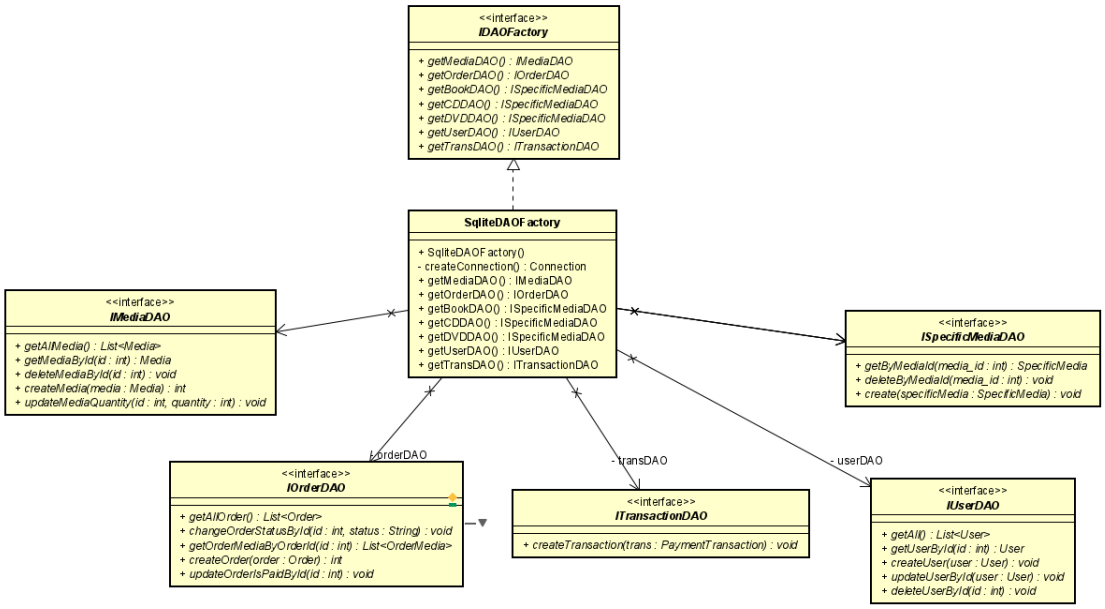
•Liskov substitution principle: The subclass can be substitution its parents. All the derived classes substitute the base class because there are many derived classes that extend from base classes without overriding the inherited method. It means that the derived classes have the same methods as the base classes so that it can substitude the base class.

•Interface segregation principle: our design makes sure that all the client class do not implement the interface that contain some unuse method. It means that every class of the system if implemented from another interface will use all the methods that have defined in this interface. Hence, in the future, if there are any new class that need to implement new methods, we can easily merge all of them for easily manage these kinds of interface.

•Dependency inversion principle: in the controller package, we create the basecontroller class, and all the higher controller like homeController, cartController.. Are dependent on this lower class, this makes our system easier to extend. Take the extend of cart function for example, in the future if the system wants to have some function that related to the management refund invoice, we can create new controller that inherit from the base controller with default action like set the screen, set controller for this screen. This is also the way that we ensure the open close principle.

5.4 Design Patterns

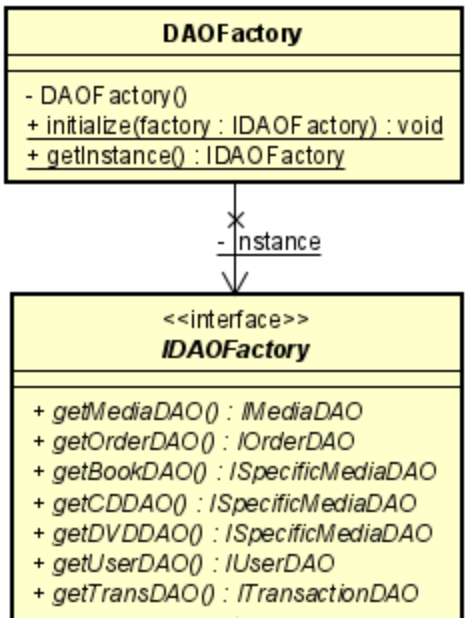
a) Abstract factory design pattern



The abstract factory design pattern is particularly useful in this architecture because it provides a way to encapsulate a group of related factory methods without specifying their concrete classes. By using the IDAOFactory interface, the system allows for easy integration of different database systems in the future. For example, if there's a need to switch from SQLite to MySQL, I can simply implement a new factory class, such as MySqlDAOFactory, that adheres to the IDAOFactory interface. This new factory will return DAO instances that interact with the MySQL database instead of SQLite, ensuring minimal changes to the existing codebase and promoting scalability and flexibility.

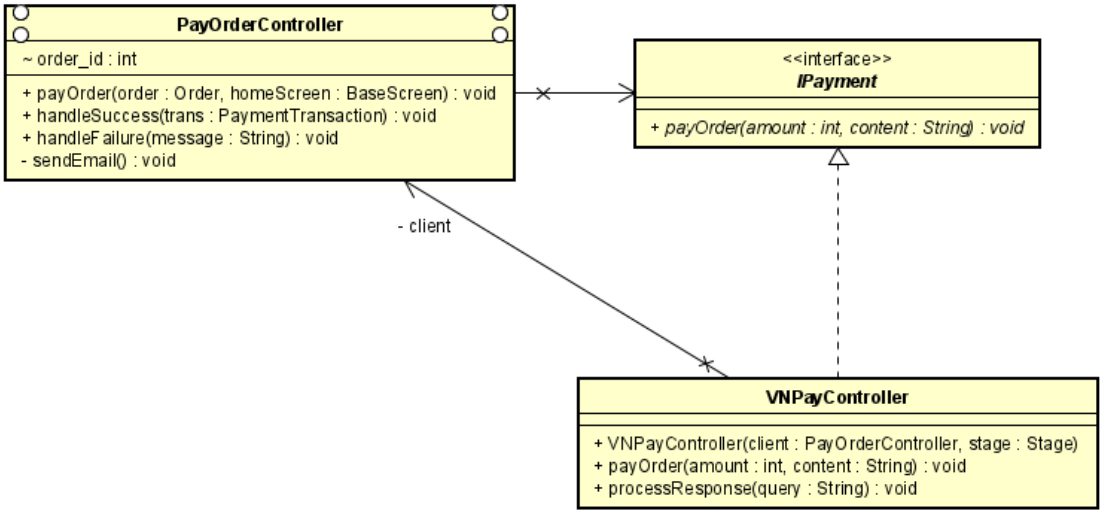
The choice of an abstract factory over a simple factory is driven by the need to produce families of related objects. A simple factory would only allow for the creation of one type of object, whereas the abstract factory pattern can produce a set of related objects (e.g., media DAOs, order DAOs, user DAOs, etc.). This pattern enhances the maintainability and extensibility of the code, as the DAOs can be easily replaced or extended by altering only the factory implementations, without affecting the core logic that depends on these DAOs. Additionally, this approach adheres to the Open/Closed Principle, one of the SOLID principles, which states that software entities should be open for extension but closed for modification.

b) Singleton pattern



DAOFactory is a singleton object that implements IDAOFactory. We set the DAOFactory to be a specific type, like SqliteDAOFactory and MySqlDAOFactory at the time of initializing the application. There are 2 reasons why we use this pattern. First, we want the factory to act like a global object so that we could access everywhere without passing through many classes. Secondly, we want the database we interact with is consistent while the app is running.

c) Strategy pattern



In the future, we intend to integrate other payment methods like Paypal, ViettelPay,… That is why we use the strategy pattern. By implementing new payment method and

setting it in PayOrderController via Ipayment interface, we could easily switch between strategies without modyfing PayOrderController.