HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

AN INTERNET MEDIA STORE

Subject: ITSS SOFTWARE DEVELOPMENT

Group 17

Đinh Việt Quang - 20215235

Ngô Minh Quý - 20215238

Trịnh Diễm Quỳnh - 20210737

Hồ Nam Sơn - 20215239

Lê Phú Tài - 20210759

Hanoi, June 2024

Table of Contents

1 Introduction

The journey towards knowledge, art, and entertainment has always been and will continue to be an integral part of human life. However, life itself is not inherently easy. There will come a time when the fruits of creative labor may struggle to reach people, as artists and intellectuals may find it challenging to sustain themselves with adequate living standards.

Fortunately, in the age of the booming Internet and the Fourth Industrial Revolution, new opportunities have emerged for all of us. One such opportunity is the AIMS Project, an E-commerce system designed for purchasing media products.

1.1 Objective

The system enables customers to browse products, add items to their cart, proceed to checkout, make payments for orders, and manage their purchase history. The project aims to offer a hands-on experience for students to enhance their programming and software engineering skills, focusing particularly on web development, database design, and software architecture. Furthermore, the project aims to deepen students' understanding of the e-commerce industry, encompassing both business processes and technical aspects such as payment processing, inventory management, and order fulfillment.

1.2 Scope

The software product to be produced is the AIMS Software, which is an online platform for e-commerce systems, a comprehensive and dynamic platform designed to cater to diverse needs. The system will support various features and functionalities to provide a seamless user experience. It allows customers to order products and make payments, and for administrators and product managers to manage users, orders, and inventory. The system will support various features and functionalities to provide a seamless user experience.

Notably, by using the AIMS website, users can expect a user-friendly interface, intuitive navigation, and a robust course catalog that spans different expertise levels.

The AIMS Software will allow customers to browse and search for products, add products to their cart, view the invoice before payment, and make payments using a prepaid credit card. Customers will also be able to cancel their orders and receive refunds.

For administrators and product managers, the AIMS Software will provide a view for managing orders, including approving or rejecting pending orders, and updating inventory levels. The software will also enable the addition, deletion, and editing of products in the inventory.

The purpose of the AIMS Software application is to provide customers with a convenient and efficient means of ordering products, while enabling product managers to effectively manage orders and inventory. The relevant benefits include streamlined order processing, improved inventory management, and increased customer satisfaction. The objectives and goals are to create a user-friendly and reliable software system that meets the needs of both customers and administrators. For purchasing purposes, customers will have the option of using a credit card. The transaction will be processed by a third-party payment processing service called VNPay.

In summary, the AIMS website is intended to be a versatile and user-centric platform that not only provides top-notch service but also fosters a sense of community and adaptability akin to successful platforms in the near future.

1.3 Glossary

No	*Term*	*Explanation*	*Example*	*Note*
1	AIMS	AIMS stands for "Automated Inventory Management System". It is a software system designed to help businesses manage their inventory and streamline their operations		
2	E-commerce	E-commerce (electronic commerce) refers to the buying and selling of goods and services over the internet.		
3	Customer	A customer is a person or organization that purchases goods or services from a business.		
4	Credit Card	A credit card is a plastic card issued by a bank or financial services company that allows cardholders to borrow funds to purchase goods and services. The borrowed funds must be repaid with interest.		

No	*Term*	*Explanation*	*Example*	*Note*
5	CRUD	Four basic functions, namely Create, Retrieve, Update, Delete		
6	Use Case Analysis	A technique that aids in modeling the requirements of a software system. A well-crafted Use Case model will describe the system in the most intuitive and easy-to-understand way for all users and clients.		

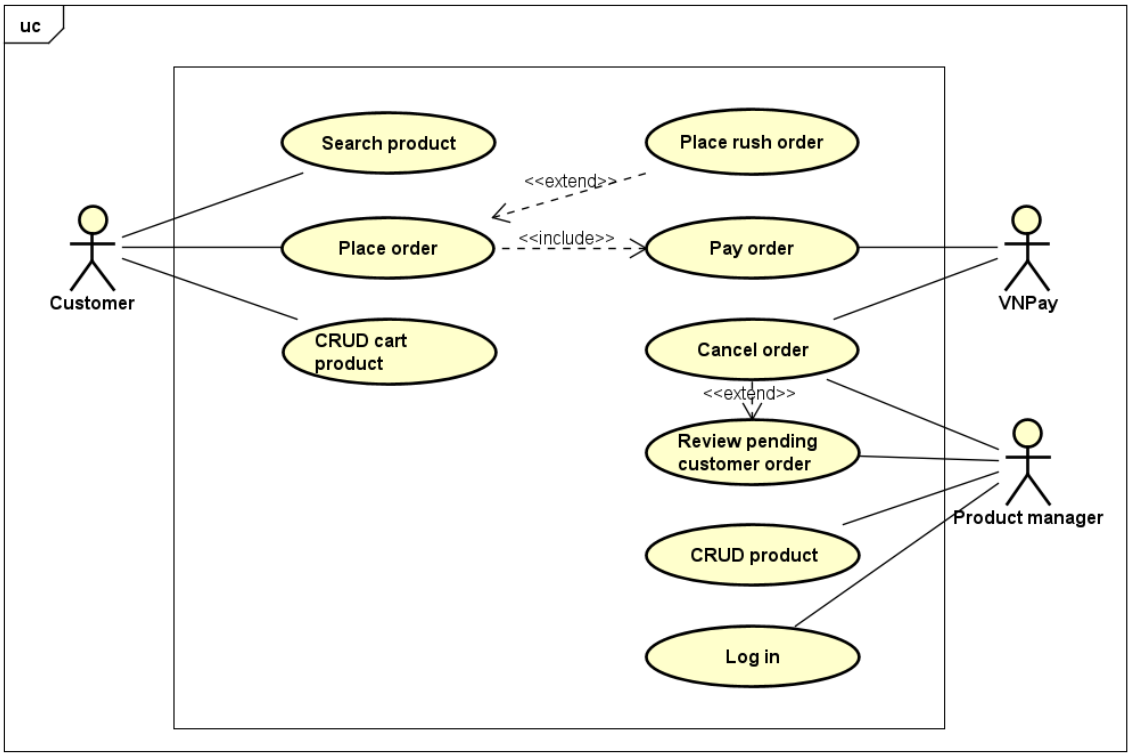
1.4 References

2 Overall Description

2.1 General Overview

The AIMS System involves three main actors interacting with the system including the customer, product manager and VNPay. Additionally, we have our own database to store information and data that is related to our system, as well as a subsystem to proceed payment transactions.

Use case diagram represents the interactions between actors and usecases. It represents the functional requirements of the system, showing the interaction between external and internal actors with the system. The below figure is the general use-case diagram for our design:



2.2 Assumptions/Constraints/Risks

2.2.1 Assumptions

- Related Software or Hardware:

- The AIMS platform will be built to operate seamlessly on common java application.
- The system will use a standard relational database management system (e.g., SQLite) for data storage.

- Operating Systems:

- The primary development and deployment environments will be Window-based, but the end-user application must be compatible with Windows and macOS as well.

- End-User Characteristics:

- Users are assumed to have a basic understanding of navigating internet-based stores and digital media.
- End-users will have stable internet connections and devices capable of streaming or downloading media content.

- Changes in Functionality:

- Future updates may include new media formats or integration with new payment systems, which must be accommodated by a flexible and modular system architecture.
- The platform should be designed to support scalability, allowing for feature expansion without significant rework.

2.2.2 Constraints

- Availability or Volatility of Resources:

- Development and operational resources, including skilled personnel and funding, may fluctuate.
- Dependency on third-party APIs for payment processing and media content delivery must be considered, with fallback plans in case of service disruption.

- Interoperability Requirements:

- Integration with third-party payment gateways (e.g., VNPay).
- Ability to interface with various content providers and media libraries.

- Licensing Requirements:

- Compliance with open-source licenses for any third-party libraries or tools used.
- Adherence to media licensing agreements for content distribution.

- Data Repository and Distribution Requirements:

- Centralized database for user data, transactions, and media inventory.
- Efficient content delivery network to distribute media content globally.

- Memory or Other Capacity Limitations:

- Server and database optimizations to handle large volumes of concurrent users and media content.
- Efficient caching mechanisms to reduce load times and server strain.

- Performance Requirements:

- Fast load times and high responsiveness, especially for media playback.
- System must support a large number of concurrent users without significant performance degradation.

- Other Requirements:

- Ongoing maintenance and support plans to deal with any emerging issues and ensure system longevity.

2.2.3 Risks

- Third-Party Risks:

- **Risk**: Dependency on third-party services for payment processing or content delivery.
- **Mitigation**: Establishing multiple vendor relationships and creating contingency plans.

- Resource Risks:

- **Risk**: Shortage of skilled developers or key personnel.
- **Mitigation**: Competitive hiring practices and investing in employee training and retention.

- Technical Risks:

- **Risk**: Scalability challenges as user base grows.
- **Mitigation**: Implementing scalable cloud infrastructure and load balancing.

3 System Architecture and Architecture Design

3.1 Architectural Patterns

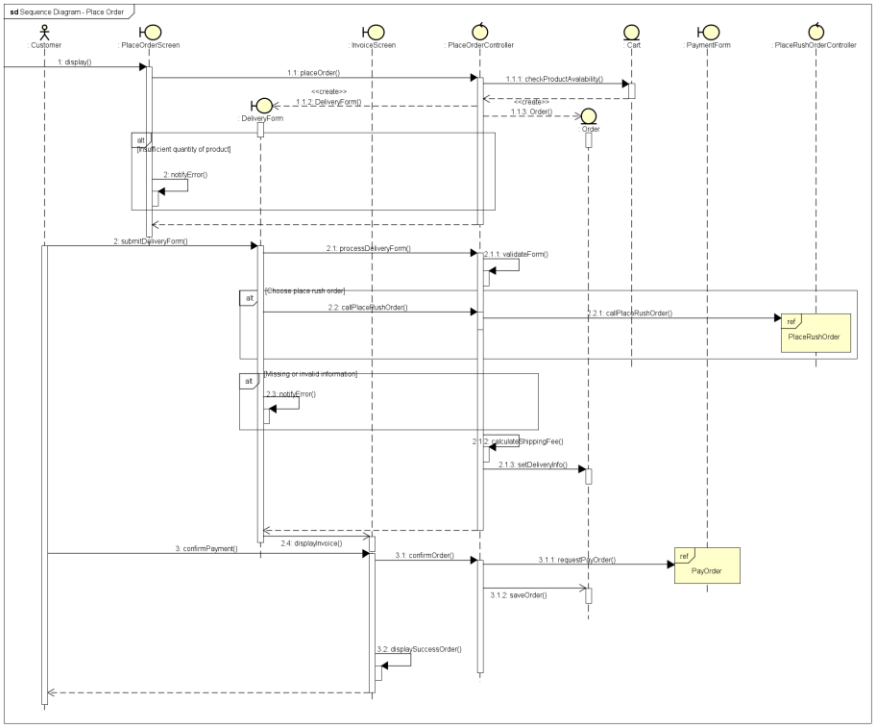
In this project, we have created the MVC project structure so that it is easier for member assignment. The MVC pattern separates the application into three main components: the Model, the View, and the Controller.

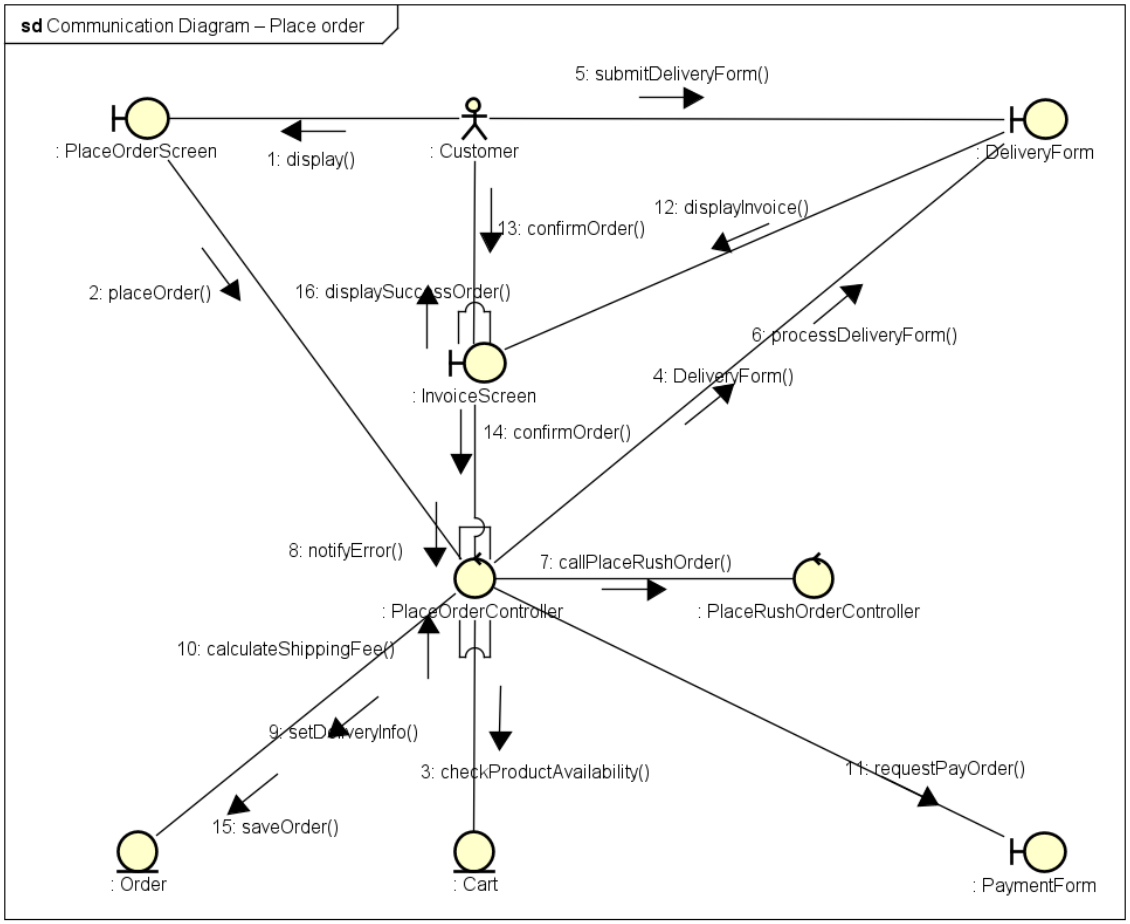
- Model: Manages the data and logic of the application.
- View – Services: Handles the user interface of the application.
- Controller: Acts as an intermediary between Model and View, processing user input and updating the Model and View accordingly.

There are several reasons why we chose MVC as the architectural patterns. Firstly, it divides the application into distinct sections, making it easier for us to manage and develop. Secondly, the clear separation between components enables the application to scale more efficiently. Each member can work on different parts of the application without interfering with each other. Additionally, components in the MVC pattern can be reused across different parts of the application. For example, the same Model Media in our design can be used with different Views such as home screen, cart screen…

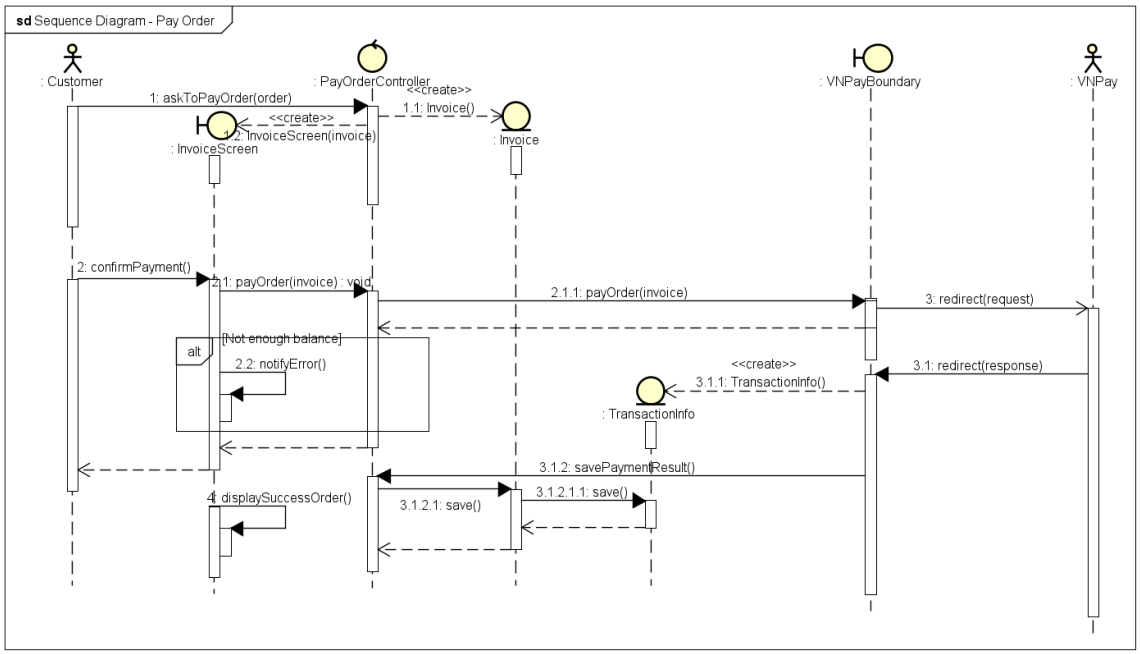
3.2 Interaction Diagrams

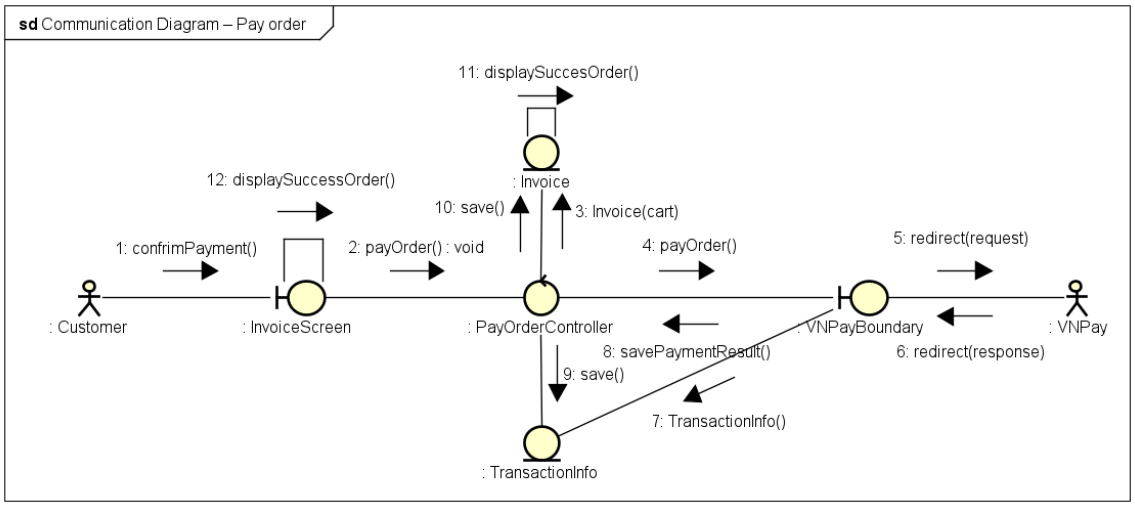
a. Place order



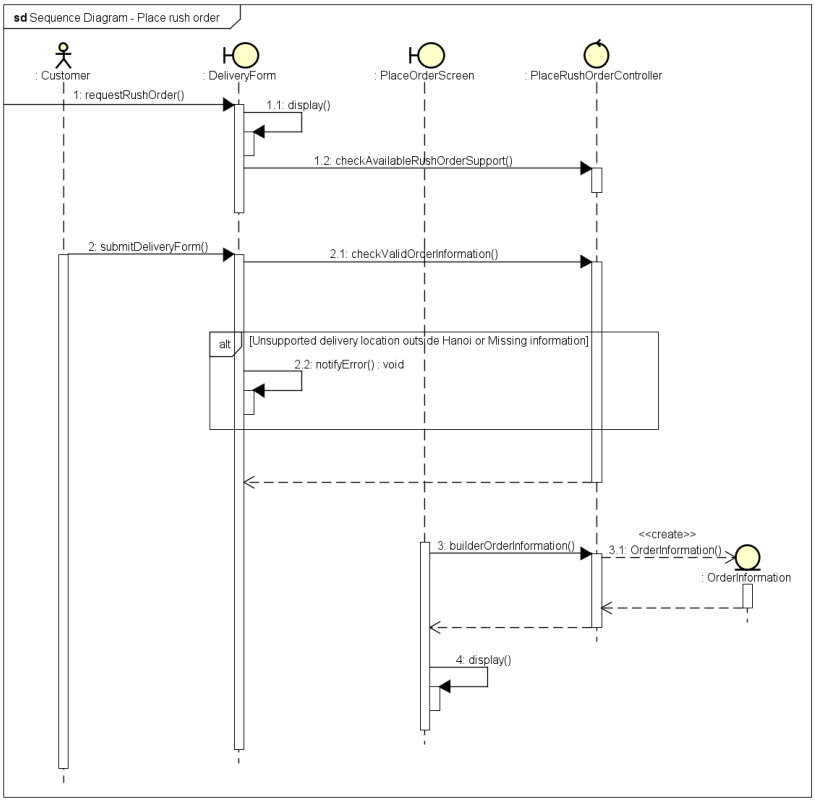

sd Communication Diagram – Place order
5: submitDeliveryForm()
: PlaceOrderScreen
1: display()
: Customer
: DeliveryForm
12: displayInvoice()
13: confirmOrder()
2: placeOrder()
16: displaySuccessOrder()
6: processDeliveryForm()
: InvoiceScreen
4: DeliveryForm()
14: confirmOrder()
8: notifyError()
7: callPlaceRushOrder()
: PlaceOrderController
: PlaceRushOrderController
10: calculateShippingFee()
9: setDeliveryInfo()
3: checkProductAvailability()
11: requestPayOrder()
15: saveOrder()
: Order
: Cart
: PaymentForm

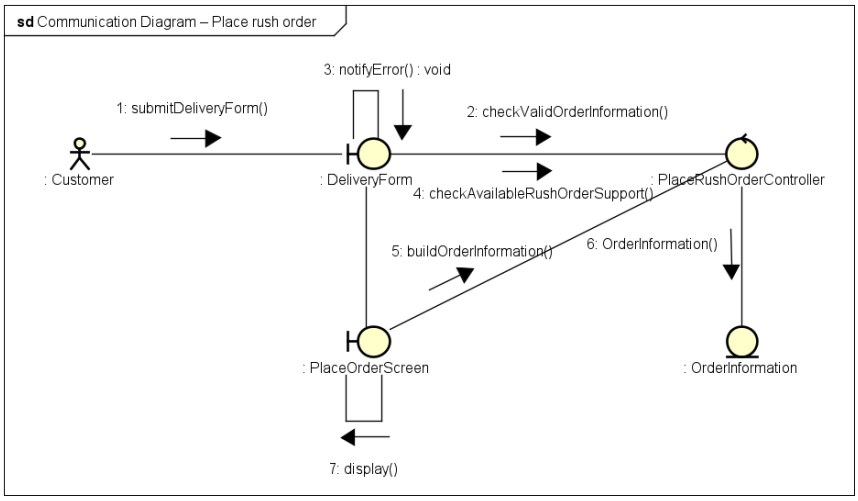
b. Pay order


sd Sequence Diagram - Pay Order
: Customer
: PayOrderController
: VNPayBoundary
: VNPay
1: askToPayOrder(order)
<<create>>
1.1: Invoice()
<<create>>
1.2: InvoiceScreen(invoice)
: InvoiceScreen
: Invoice
2: confirmPayment()
2.1: payOrder(invoice) : void
2.1.1: payOrder(invoice)
3: redirect(request)
alt
[Not enough balance]
2.2: notifyError()
<<create>>
3.1.1: TransactionInfo()
3.1: redirect(response)
: TransactionInfo
3.1.2: savePaymentResult()
4: displaySuccessOrder()
3.1.2.1: save()
3.1.2.1.1: save()

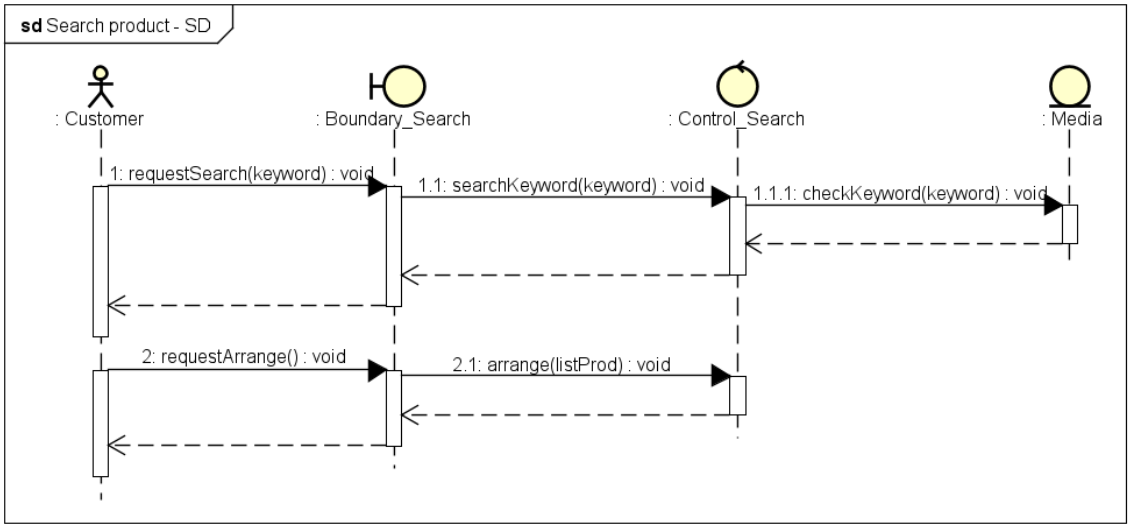


c. Place rush order

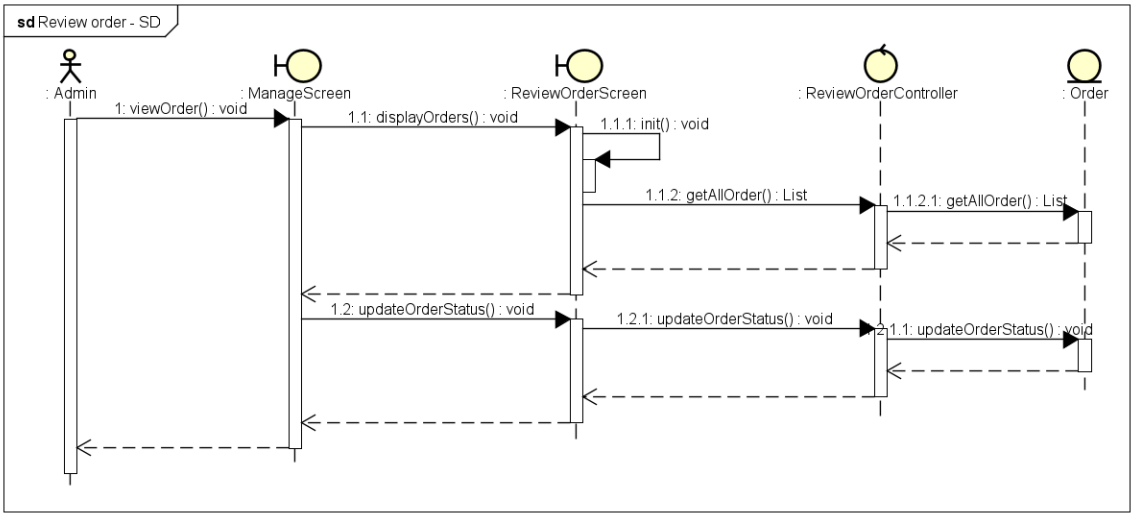




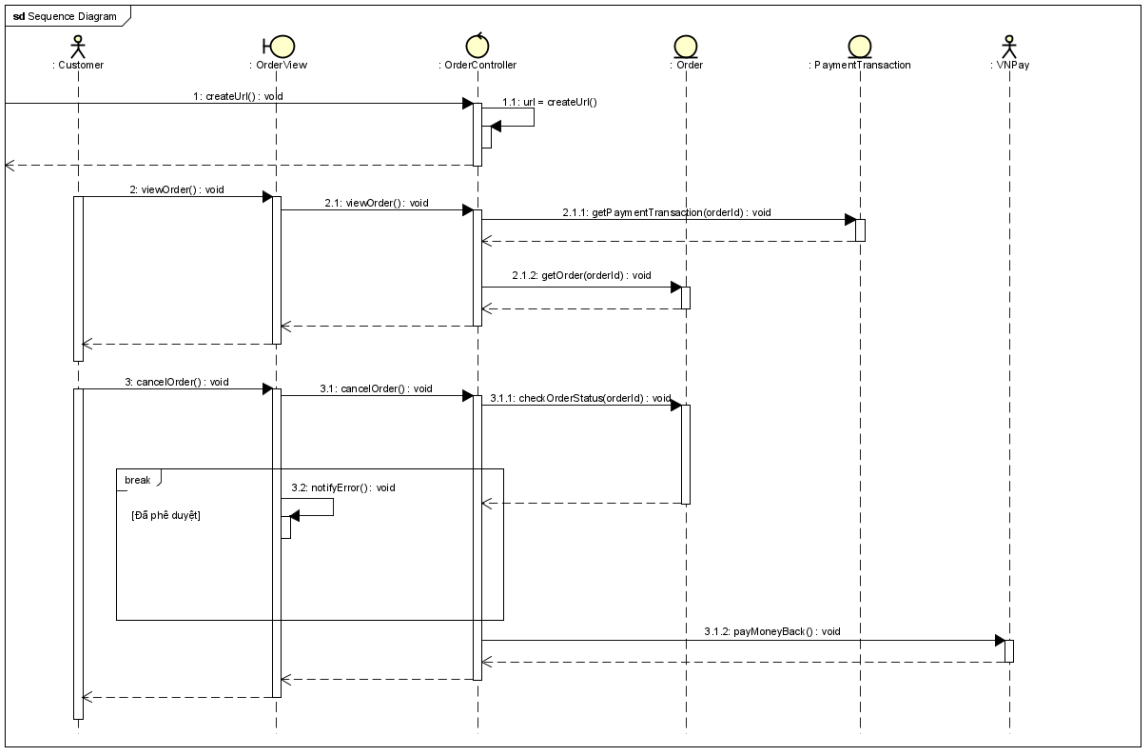
d. Search product



e. Review pending customer order

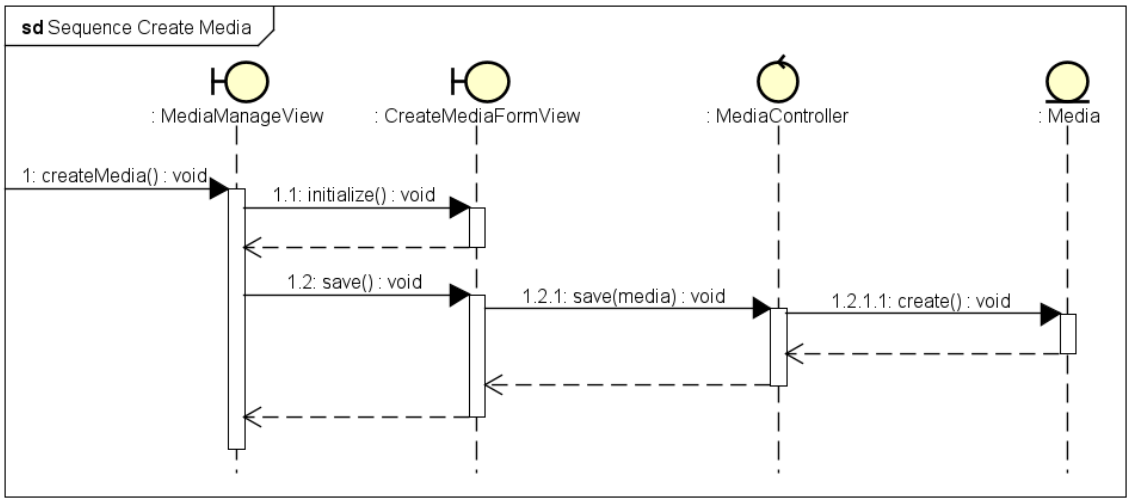


g. Cancel order

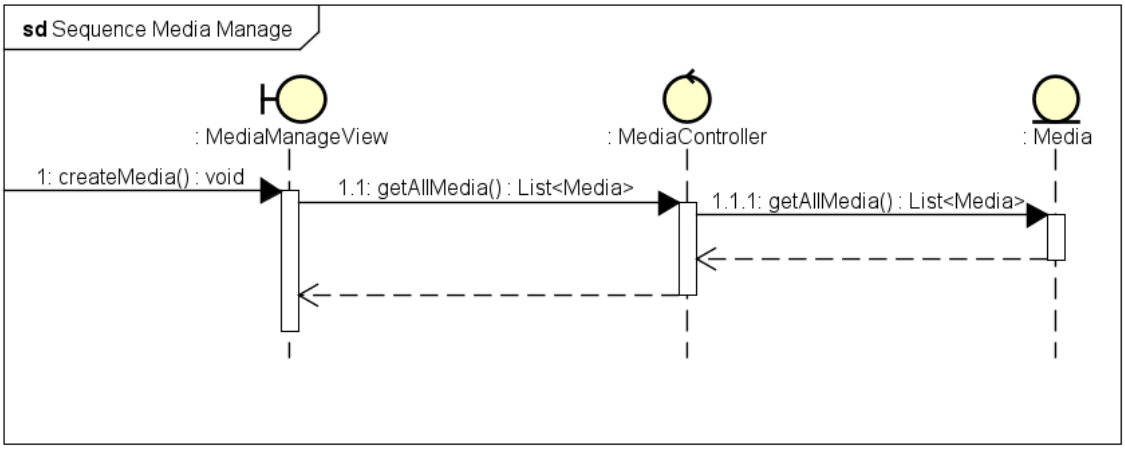


h. CRUD product

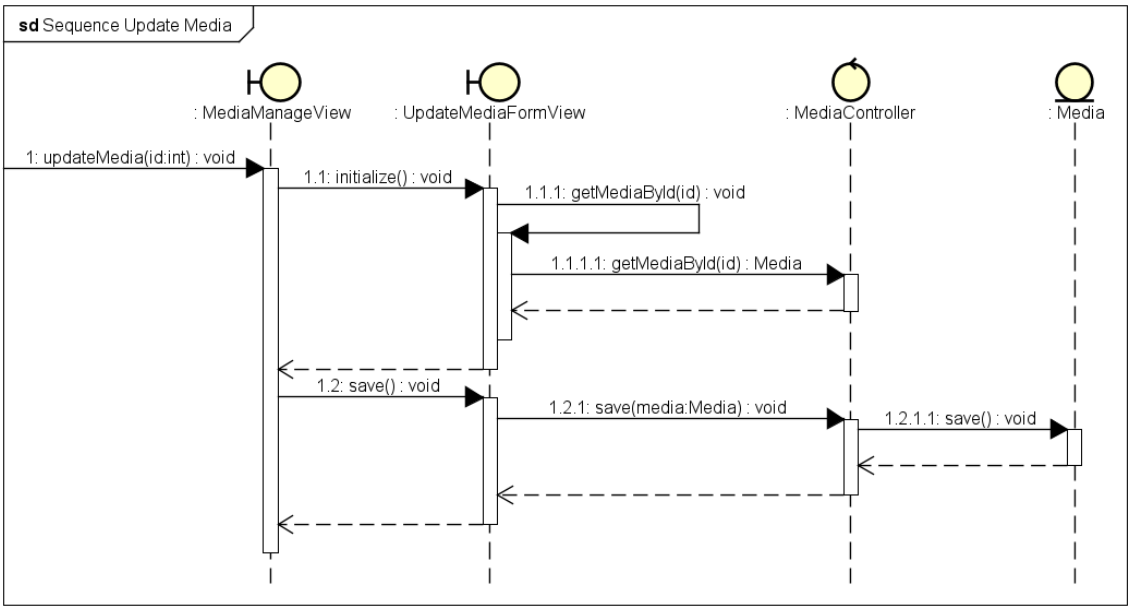
- Create Media



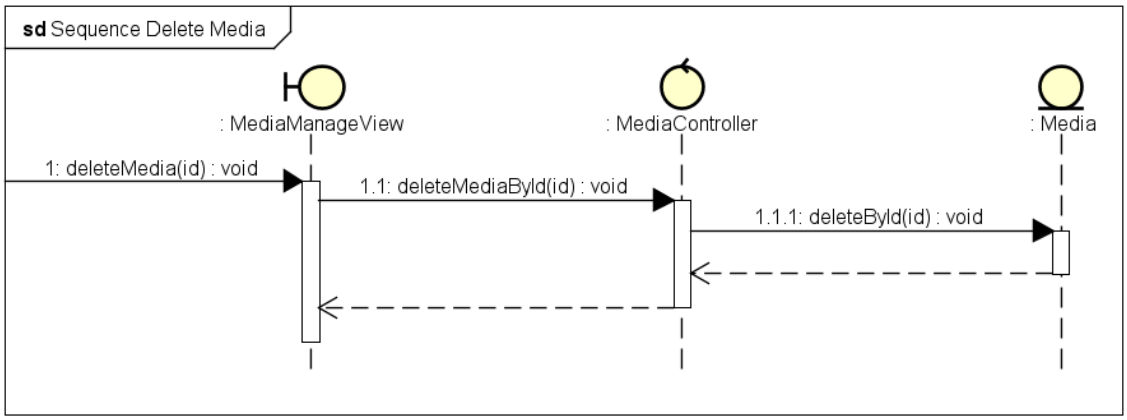
- Manage Media



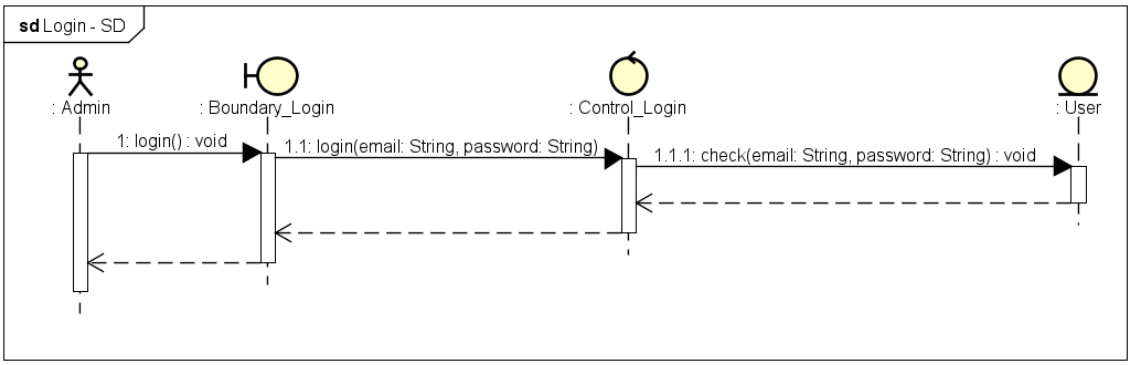
- Update Media



- Delete Media

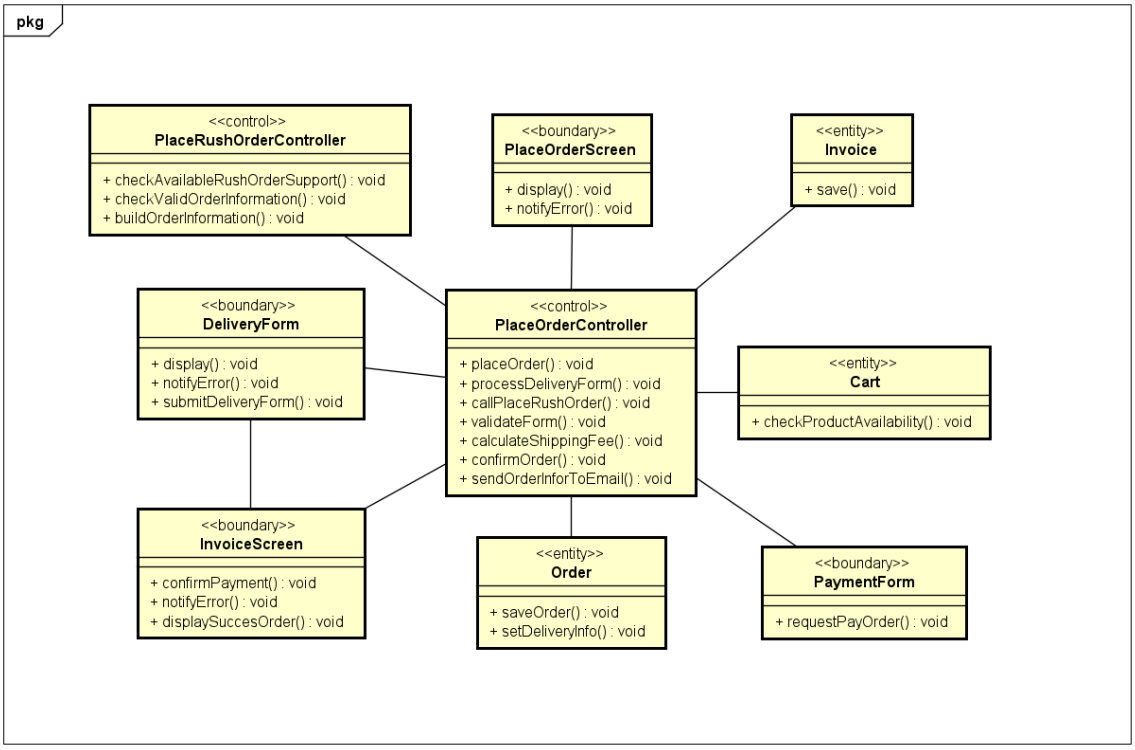


i. Log in

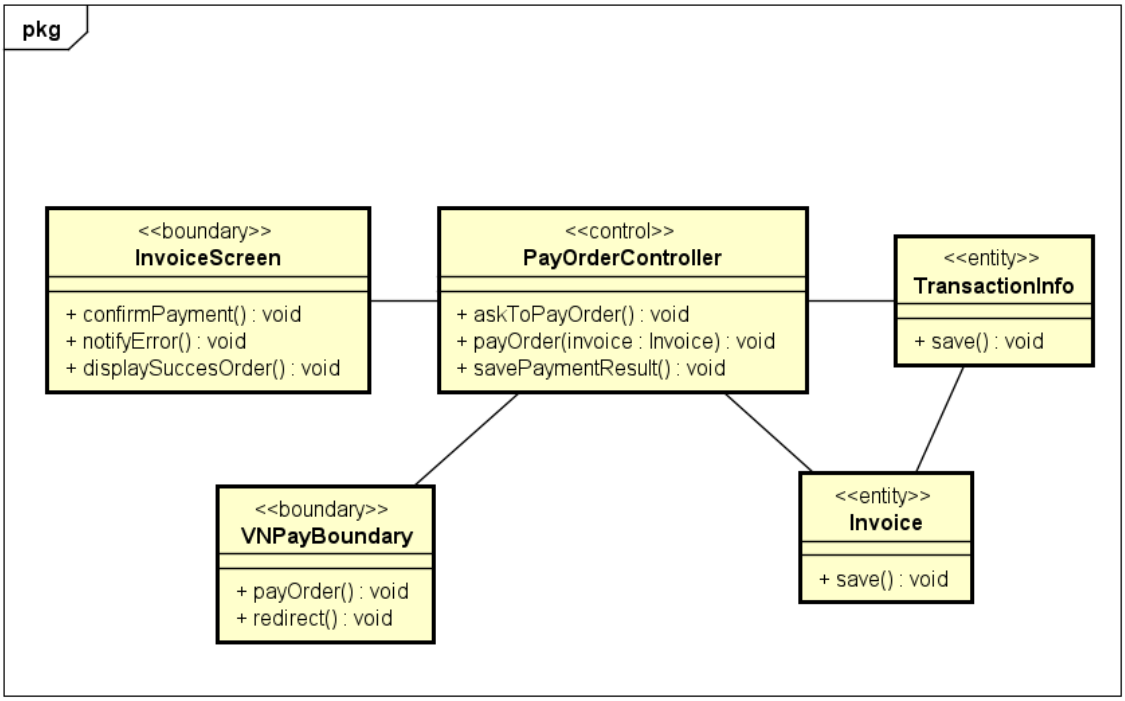


3.3 Analysis Class Diagrams

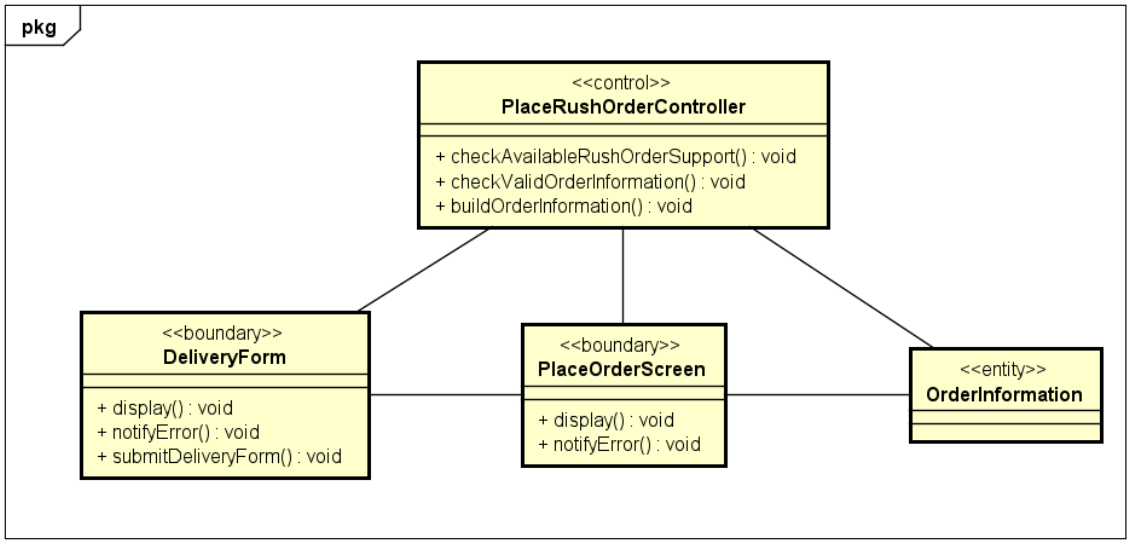
a. Place order



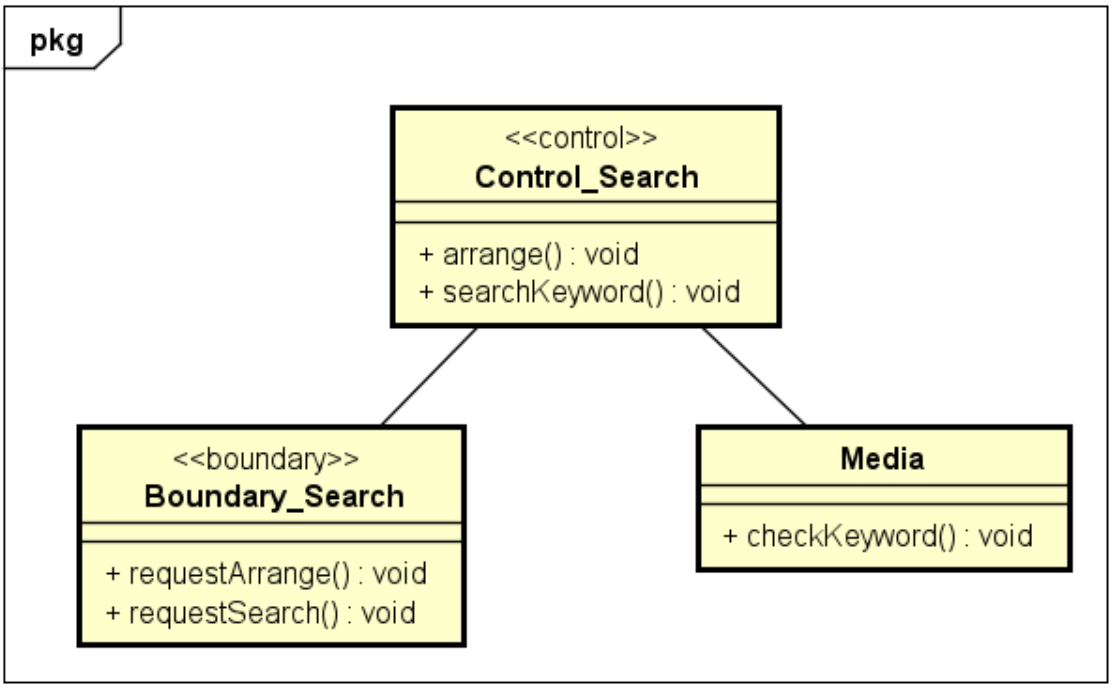
b. Pay order



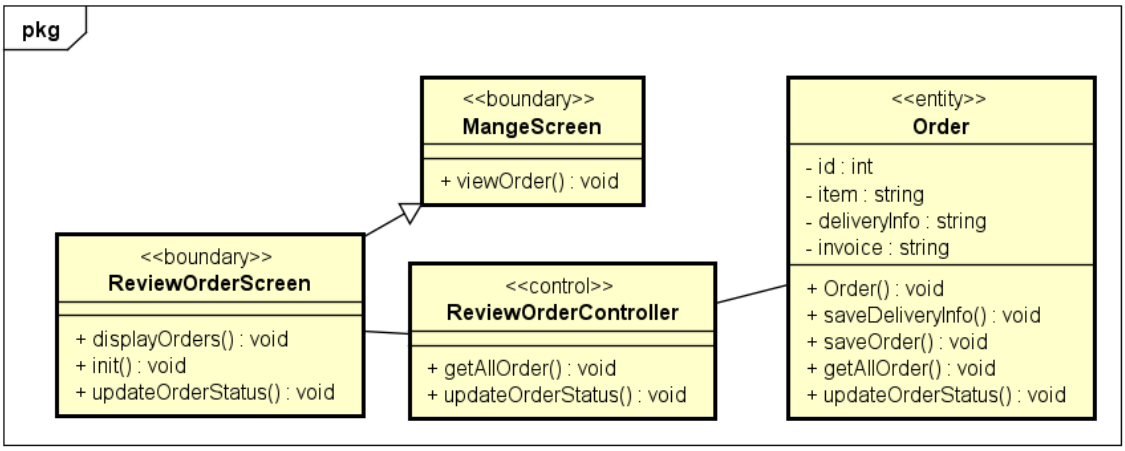
c. Place rush order



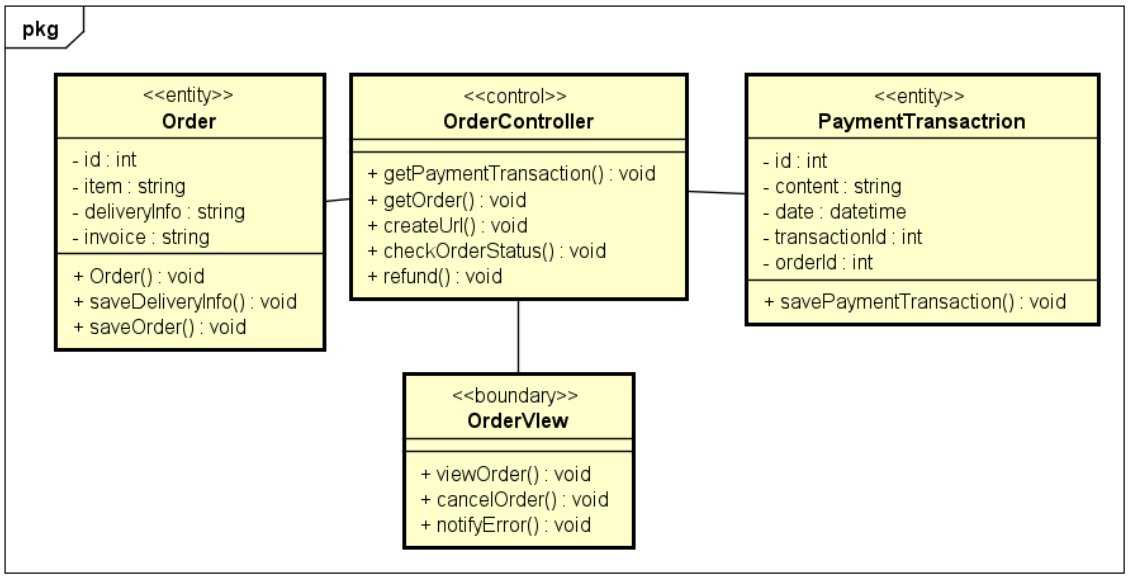
d. Search product



e. Review pending customer order

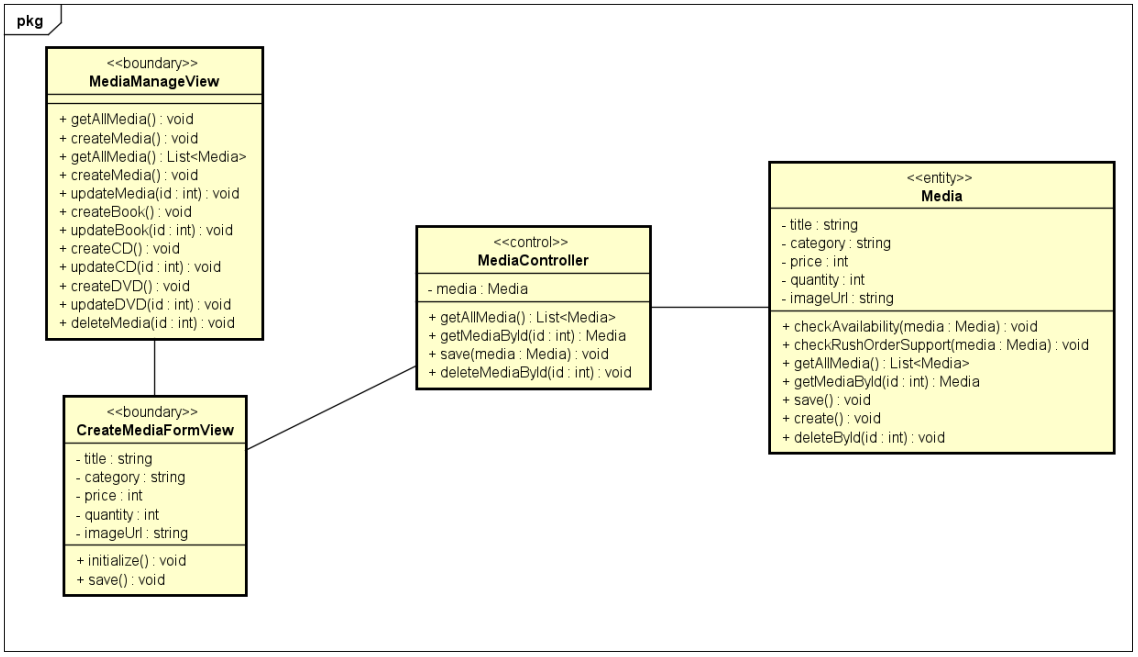


g. Cancel order

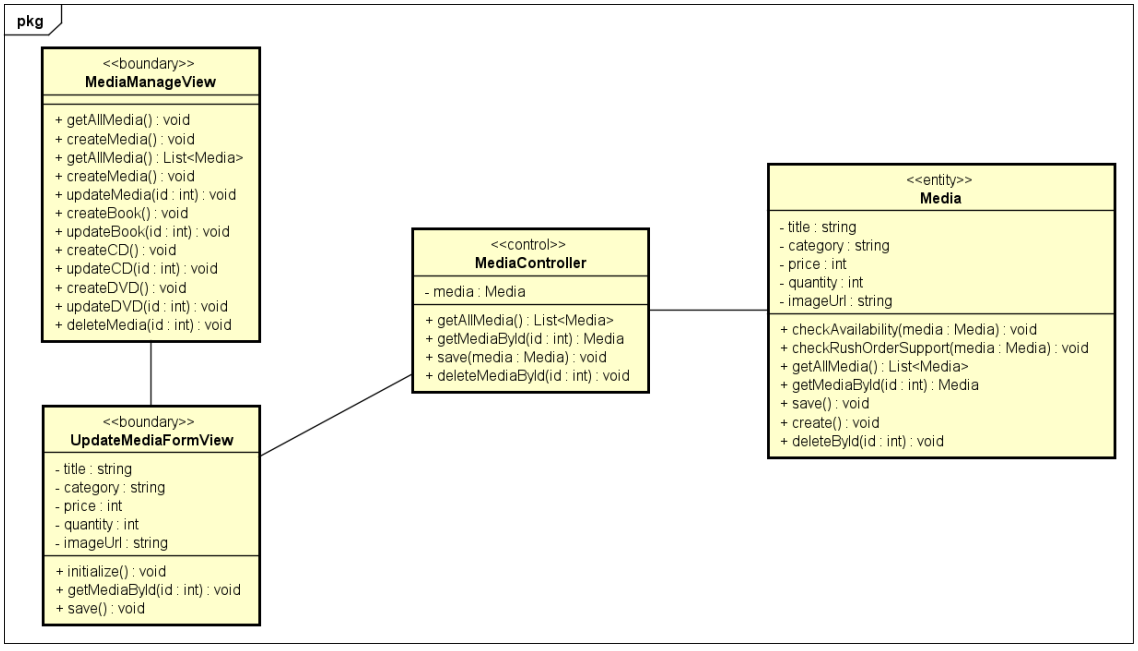


h. CRUD product

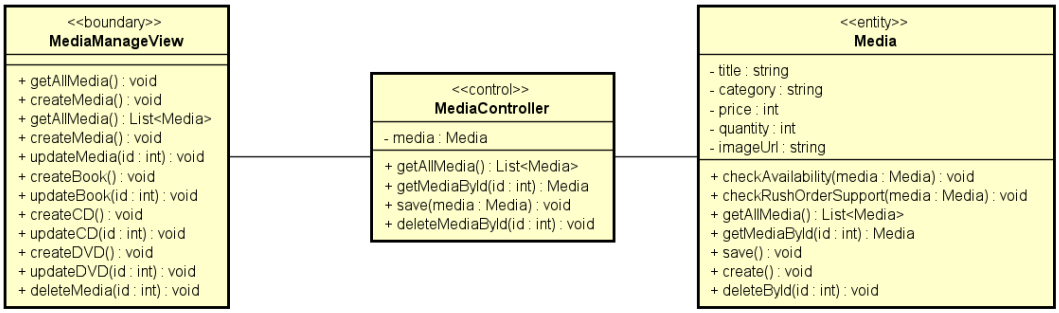
- Create Media



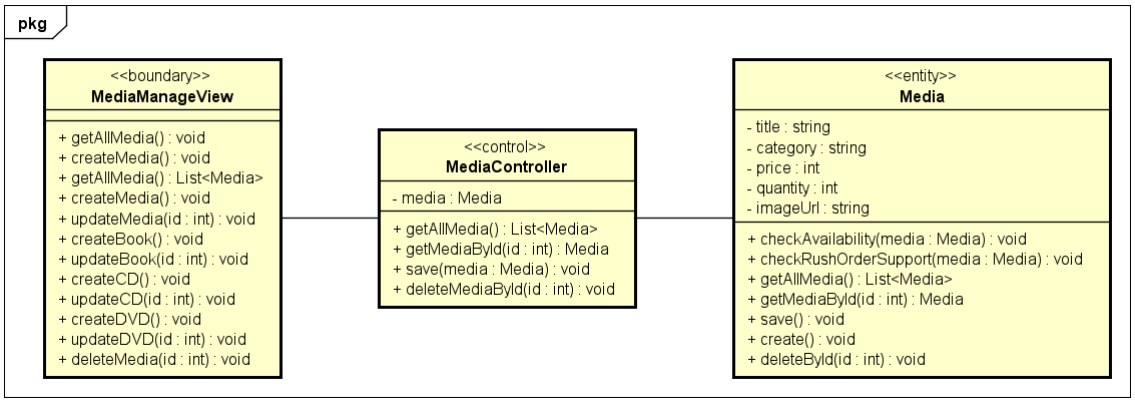
- Update Media



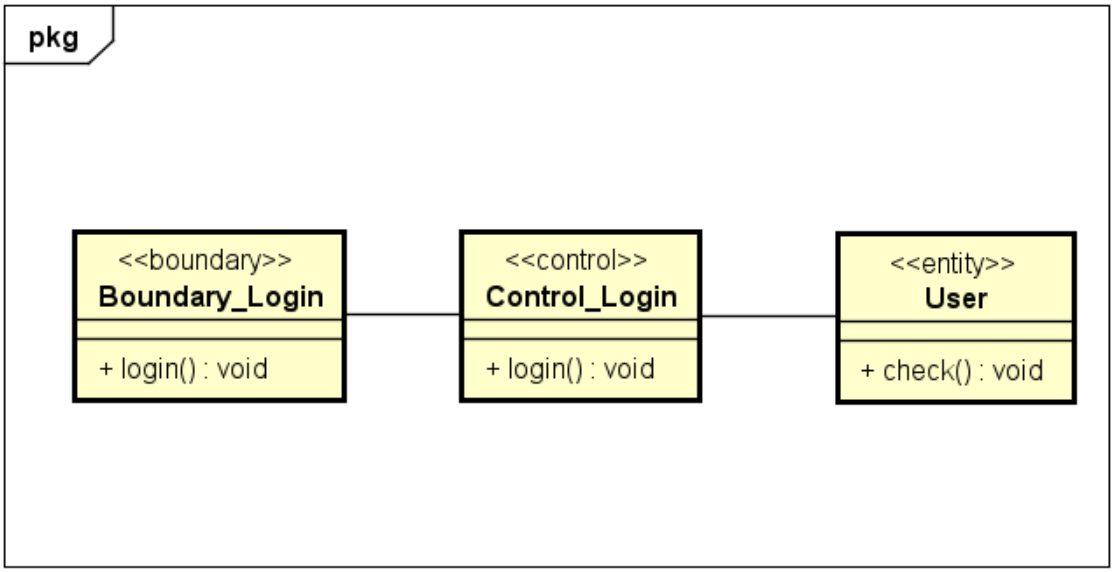
- View Media



- Delete Media



i. Log in



4 Detailed Design

4.1 *User Interface Design*

4.1.1 Screen Configuration Standardization

1. Display

Number of colors supported: 16,777,216 colors.

Resolution: 1366 × 768 pixels.

2. 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 in the middle.

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.

3. Control

Size of the text: medium size (mostly 24px).

Font: Arial (mostly), Newsreader for section text.

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

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 splash screen, and then the first screen (home screen) will appear.

4. Direct input from the keyboard

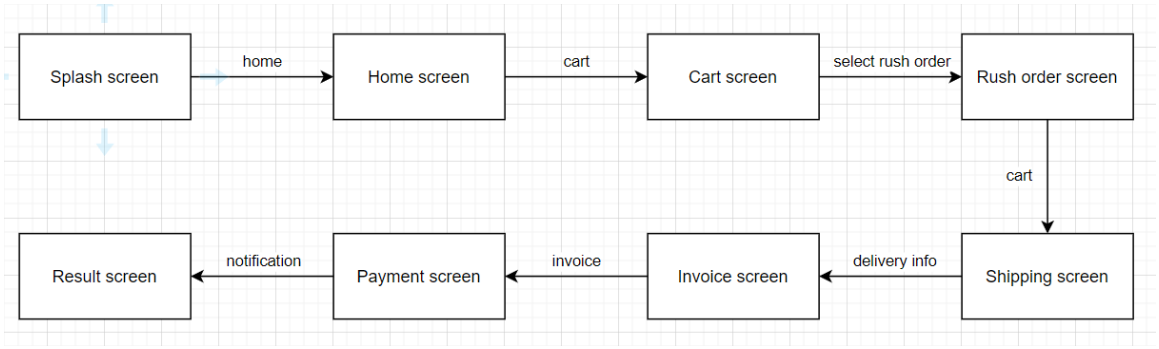
There will be no shortcuts. There are back buttons to move back to the previous screen. Also, there is the close button “X” located at the title bar to the right to close the screen.

5. Error

A message will be sent to notify the users what is the problem.

4.1.2 Screen Transition Diagrams

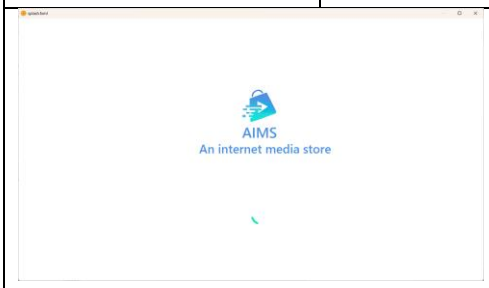
a. As a customer



b. As a manager

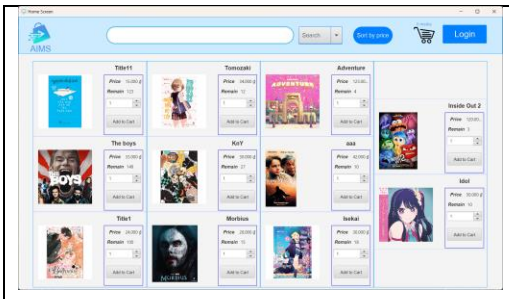
4.1.3 Screen Specifications

a. Splash screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Splash screen	6/6/2024			Dinh Viet Quang
		Control	Operation	Function	
		Area for displaying AIMS logo	Initial	Displaying AIMS logo and loading.gif	

b. Home screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Home screen	6/6/2024			Dinh Viet Quang

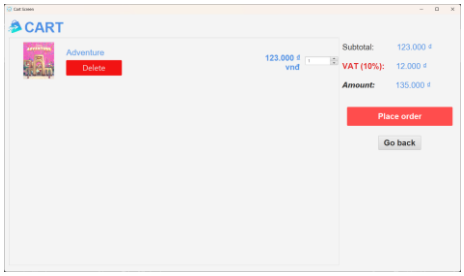


Control	Operation	Function
Area for displaying search content	Initial	Display the search content
Search button	Click	Search for items which have properties matching with search content
Area for displaying number of medias in cart and cart icon	Click	Navigate to cart screen
Area for displaying items in the shop	Initial	Display the items in the shop
Area for displaying item information	Initial	Display title, price, number of available products of that item and the amount of that item that user might want to add to cart
Arrow up button	Click	Increase the amount of item that user might want to add to cart
Arrow down button	Click	Increase the amount of item that user might decrease to add to cart
Add to cart button	Click	Add item to the cart

Screen name	Home screen			
Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Search content	500	Character	Black	
Amount of item	20	Numeral	Black	

c. Cart screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Cart screen	6/6/2024			Dinh Viet Quang

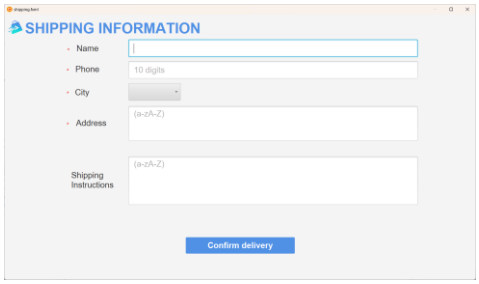


Control	Operation	Function
Area for displaying the subtotal	Initial	Display the subtotal
Area for display items in the cart	Initial	Display the media with the corresponding information
Place order button	Click	Display the Delivery Form
Delete button	Click	Remove the item from the cart
Back button	Click	Return to previous screen

Screen name	Cart screen			
Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Media title	50	Character	Blue	Left-justified
Price	20	Numeral	Blue	Right justified
Subtotal	20	Numeral	Blue	Left-justified

d. Shipping screen

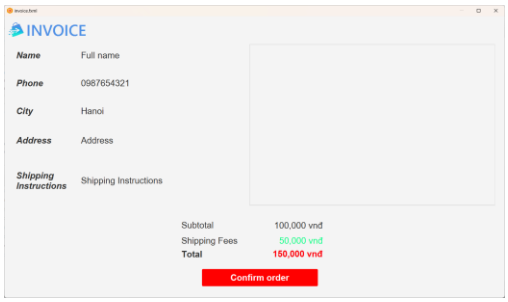
AIMS Software	Date of creation	Approved by	Reviewed by	Person in charge

Screen specification	Shipping screen	6/6/2024			Dinh Viet Quang
		Control	Operation	Function	
		Area for display customer information	Initial	Display some questions to request customer to fill	
		Confirm delivery	Click	Display the Invoice	

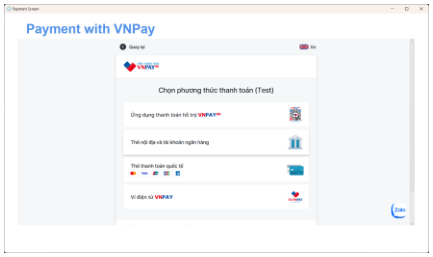
Screen name	Shipping screen			
Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Name	30	Character	Black	
Phone	10	Numeral	Black	
Province/City	50	Character	Black	
Address	50	Character	Black	
Shipping method	100	Character	Black	

e. Invoice screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Invoice screen	6/6/2024			Dinh Viet Quang
		Control	Operation	Function	

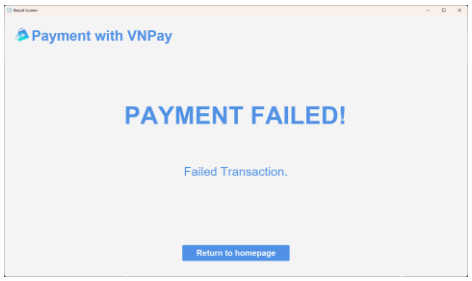
	Area for display order information	Initial	Display some order information
	Area for displaying the subtotal	Initial	Display the subtotal
	Confirm to Order button	Click	Display the payment

f. Payment screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Payment screen	6/6/2024			Dinh Viet Quang
		Control	Operation	Function	
		Area for display customer's card information	Initial	Display some question to request customer to fill	
		Choose payment method	Click	Display payment method that customer can choose (credit card is default)	
		Confirm payment	Click	Display the result	
		Back button	Click	Return to previous screen	

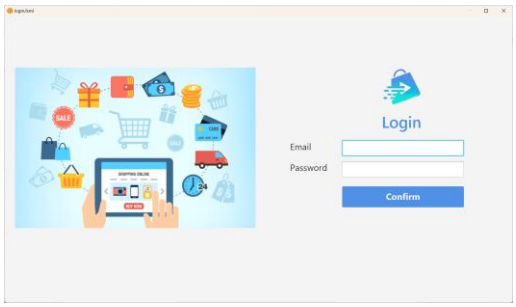
Screen name	Payment screen			
Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Card name	30	Character	Black	Left-justified
Card number	30	Numeral	Black	Left-justified
First name	30	Character	Black	Left-justified
Surname	30	Character	Black	Left-justified
Date	30	Date	Black	Left-justified

g. Result screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Result screen	6/6/2024			Dinh Viet Quang
		Control	Operation	Function	
		Area for displaying result	Initial	Display purchase result	
		Button	Click	Return to home screen	

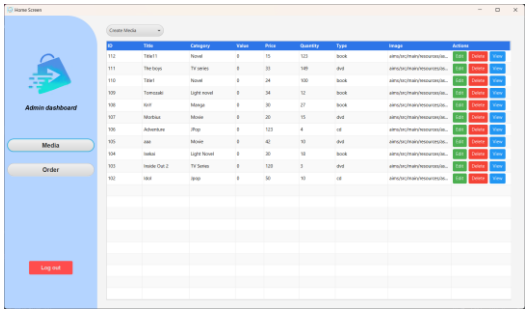
h. Login screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Login screen	6/6/2024			Dinh Viet Quang

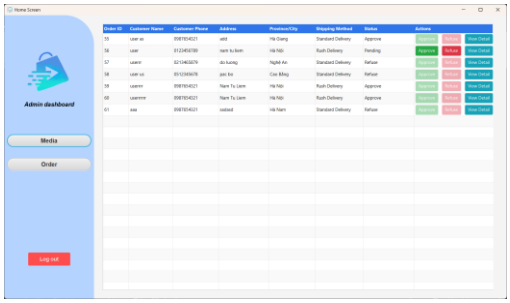
	Control	Operation	Function
	Area for display email and password	Initial	Display some fields to request manager to fill to access
	Confirm	Click	Display the Manage screen
	Logo	Click	Return to Home screen

i. Manage product screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Manage product screen	6/6/2024			Dinh Viet Quang

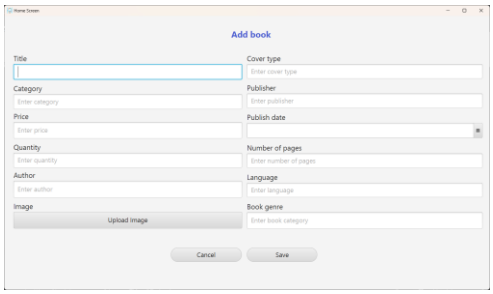
	Control	Operation	Function
	Media button	Click	Redirect to Manage product screen
	Order button	Click	Redirect to Manage order screen
	Logout button	Click	Log out from Manage screen
	Dashboard	Initial	Display media list
	Create button	Click	Redirect to create form
	Edit button	Click	Redirect to update form
	Delete button	Click	Delete the media out of the database
	View button	Click	View the media info

j. Manage order screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Manage order screen	6/6/2024			Dinh Viet Quang
		Control	Operation	Function	
		Media button	Click	Redirect to Manage product screen	
		Order button	Click	Redirect to Manage order screen	
		Logout button	Click	Log out from Manage screen	
		Dashboard	Initial	Display media list	
		Approve button	Click	Change the order status to approvement	
		Reject button	Click	Change the order status to rejection	
		View button	Click	View the order info	

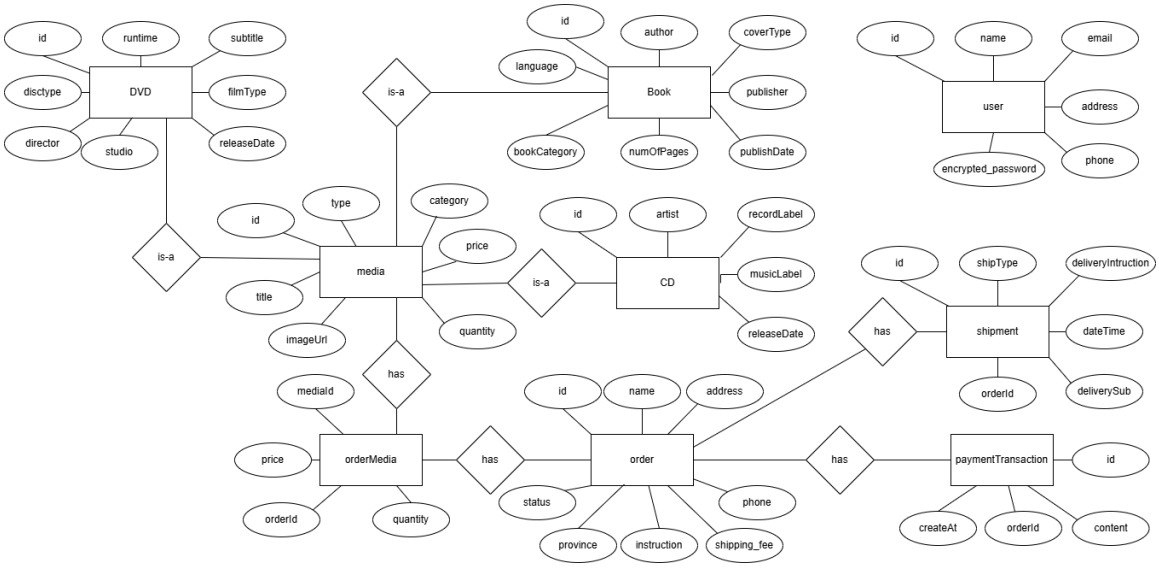
k. Create media screen

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Create media screen	6/6/2024			Dinh Viet Quang
		Control	Operation	Function	
		Area for display title screen	Initial	Display the form to create	

	Text field	Fill	Fill the media information
	Save button	Click	Save/Update the media
	Cancel button	Click	Cancel the process and return to the manage media screen

4.2 Data Modeling

4.2.1 Conceptual Data Modeling

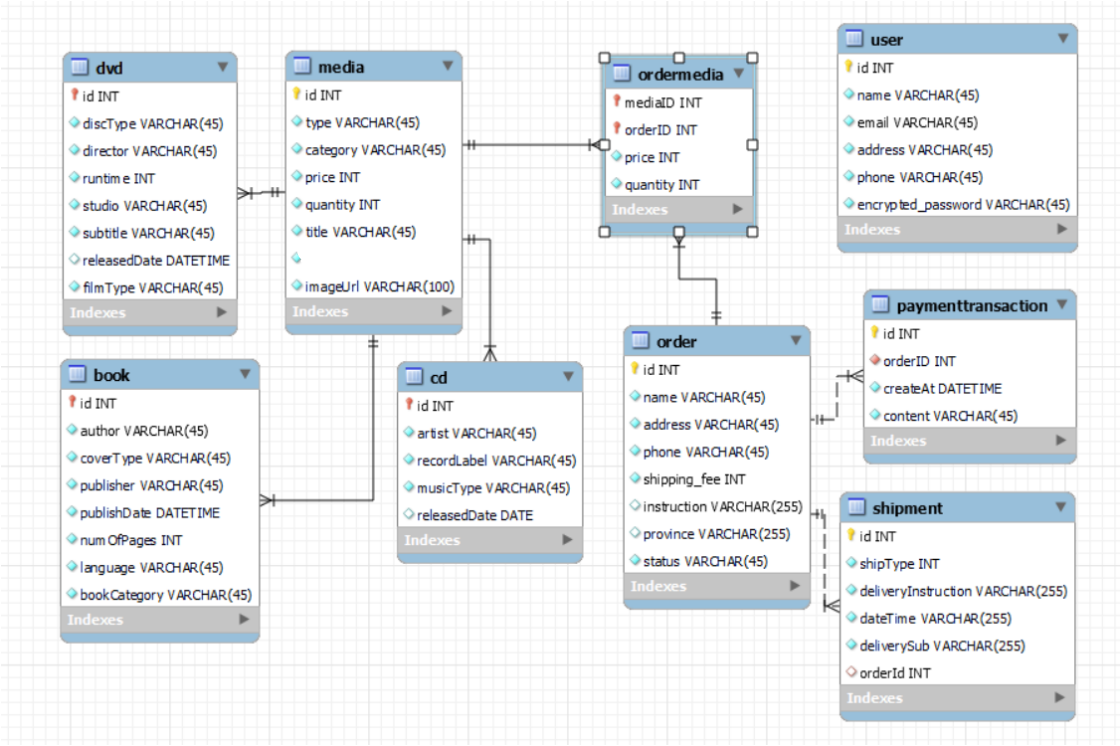


4.2.2 Database Design

4.2.2.1 Database Management System

We decided to use SQLite as our Database Management System.

4.2.2.2 Database Diagram



4.2.2.3 Database Detail Design

Table 1. Example of table design

a.DVD Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2			discType	VARCHAR(45)		Yes	Type of the disc
3			director	VARCHAR(45)			Director of the DVD
4			runtime	INT			Runtime in minutes
5			studio	VARCHAR(45)			Studio
6			subtitle	VARCHAR(45)			Subtitle
7			releasedDate	DATETIME			Release date
8			filmType	VARCHAR(45)			Type of the film

b. Book Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2			author	VARCHAR(45)		Yes	Author of the book

3			coverType	VARCHAR(45)		Yes	Type of the cover
4			publisher	VARCHAR(45)		Yes	Publisher
5			publishDate	DATETIME		Yes	Publish date
6			numOfPages	INT		Yes	Number of pages
7			language	VARCHAR(45)		Yes	Language
8			bookCategory	VARCHAR(45)		Yes	Category of the book

c. CD Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2			artist	VARCHAR(45)			Artist
3			recordLabel	VARCHAR(45)			Record label
4			musicType	VARCHAR(45)			Type of music
5			releasedDate	DATE			Release date

d. Media Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2			type	VARCHAR(45)			Type of media
3			category	VARCHAR(45)			Category of media
4			price	INT			Price
5			quantity	INT			Quantity available
6			title	VARCHAR(45)			Title of the media
7			imageUrl	VARCHAR(100)			URL of the media image

e. OrderMedia Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		mediaID	INT		Yes	ID of the media
2	✓		orderID	INT		Yes	ID of the order
3			price	INT			Price of the order media
4			quantity	INT			Quantity of the order media

f. User Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2			name	VARCHAR(45)			Name of the user
3			email	VARCHAR(45)			Email address of the user
4			address	VARCHAR(45)			Address of the user
5			phone	VARCHAR(45)			Phone number of the user
6			encrypted_password	VARCHAR(45)			Encrypted password

g. Order Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2			name	VARCHAR(45)			Name of the order
3			address	VARCHAR(45)			Address for the order
4			phone	VARCHAR(45)			Phone number for the order
5			shipping_fee	INT			Shipping fee
6			instruction	VARCHAR(255)			Instruction for the order
7			province	VARCHAR(255)			Province for the order
8			status	VARCHAR(45)			Status of the order

h. PaymentTransaction Schema

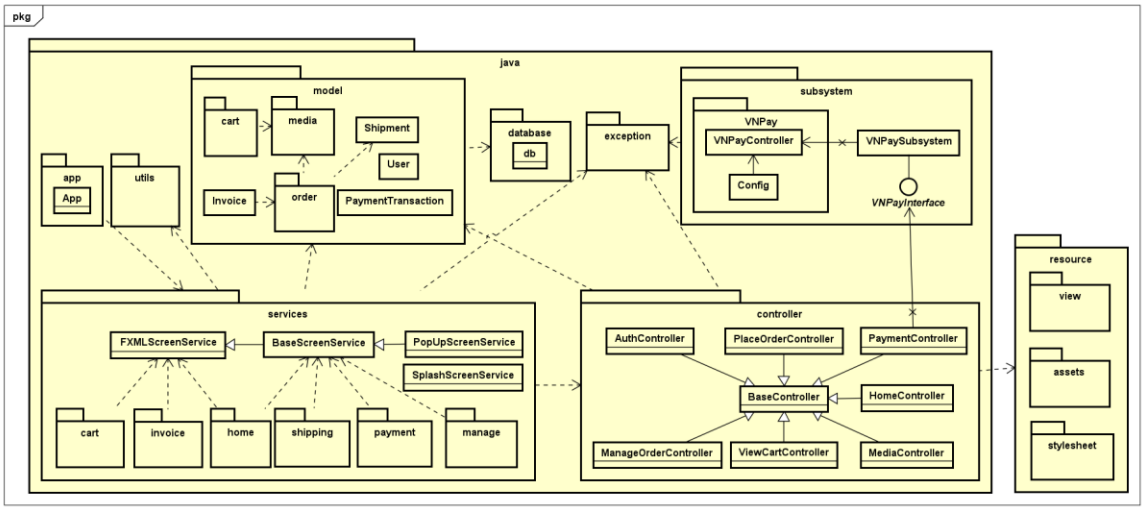
#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2		✓	orderID	INT		Yes	ID of the associated order
3			createAt	DATETIME			Creation date and time
4			content	VARCHAR(45)			Content of the transaction

i. Shipment Schema

#	*PK*	*FK*	*Column name*	*Data type*	*Default value*	*Mandatory*	*Description*
1	✓		id	INT		Yes	Unique identifier
2			shipType	INT			Type of shipment
3			deliveryInstruction	VARCHAR(255)			Delivery instruction
4			dateTime	VARCHAR(255)			Date and time of shipment
5			deliverySub	VARCHAR(255)			Delivery subtitle
6		✓	orderId	INT			ID of the associated order

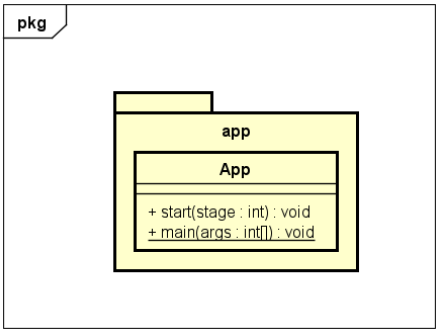
4.3 Class Design

4.3.1 General Class Diagram

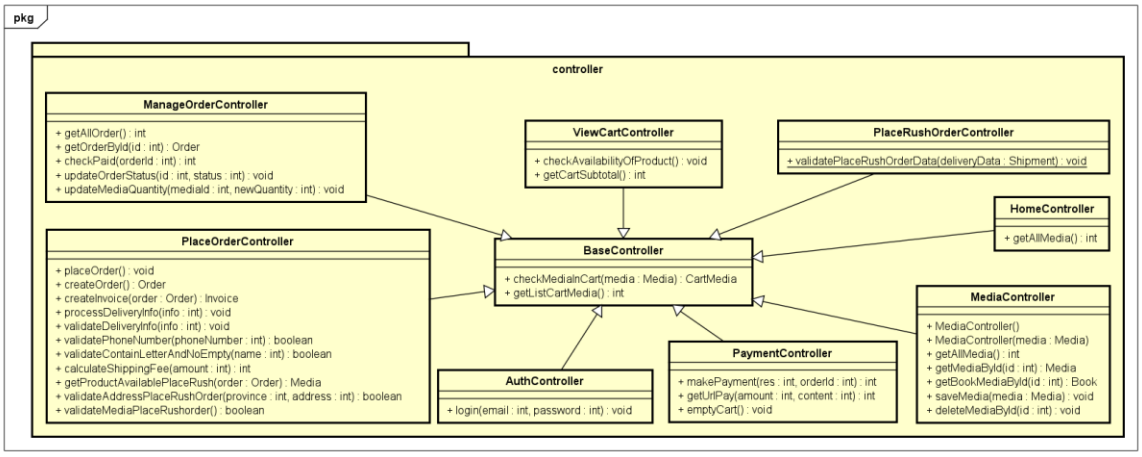


4.3.2 Class Diagrams

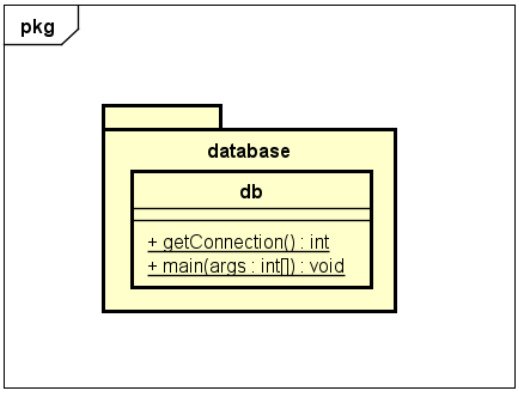
4.3.2.1 Class Diagram for Package app



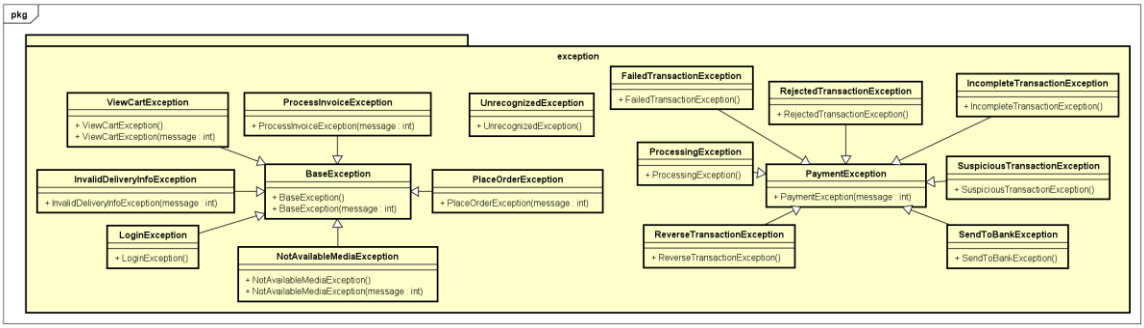
4.3.2.2 Class Diagram for Package controller



4.3.2.3 Class Diagram for Package database

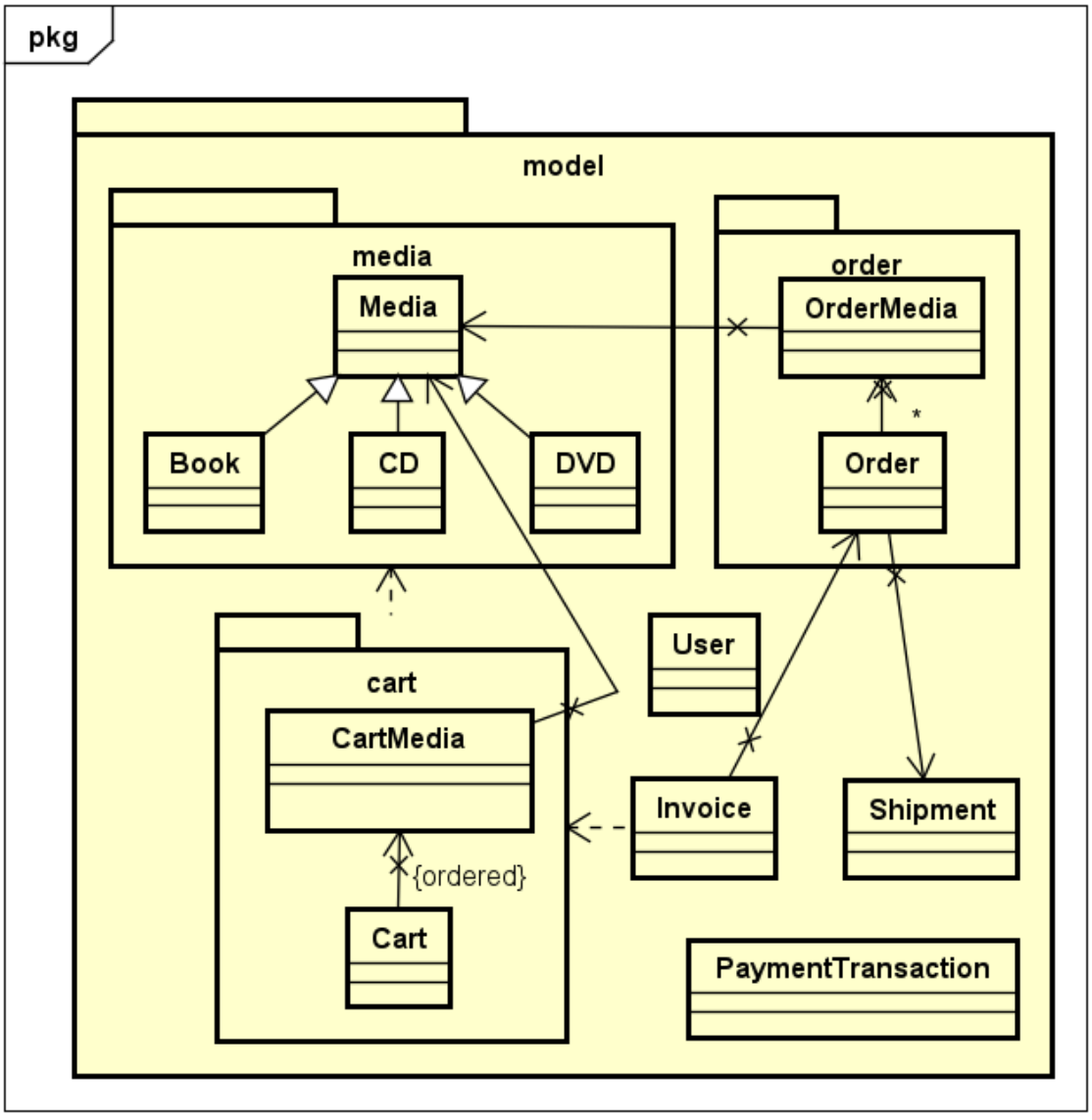


4.3.2.4 Class Diagram for Package exception

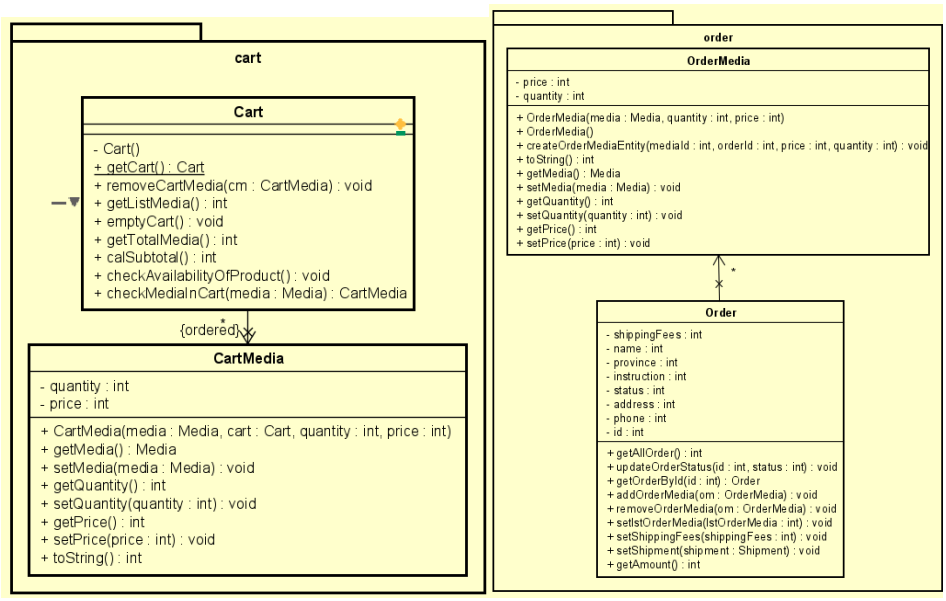


4.3.2.5 Class Diagram for Package model

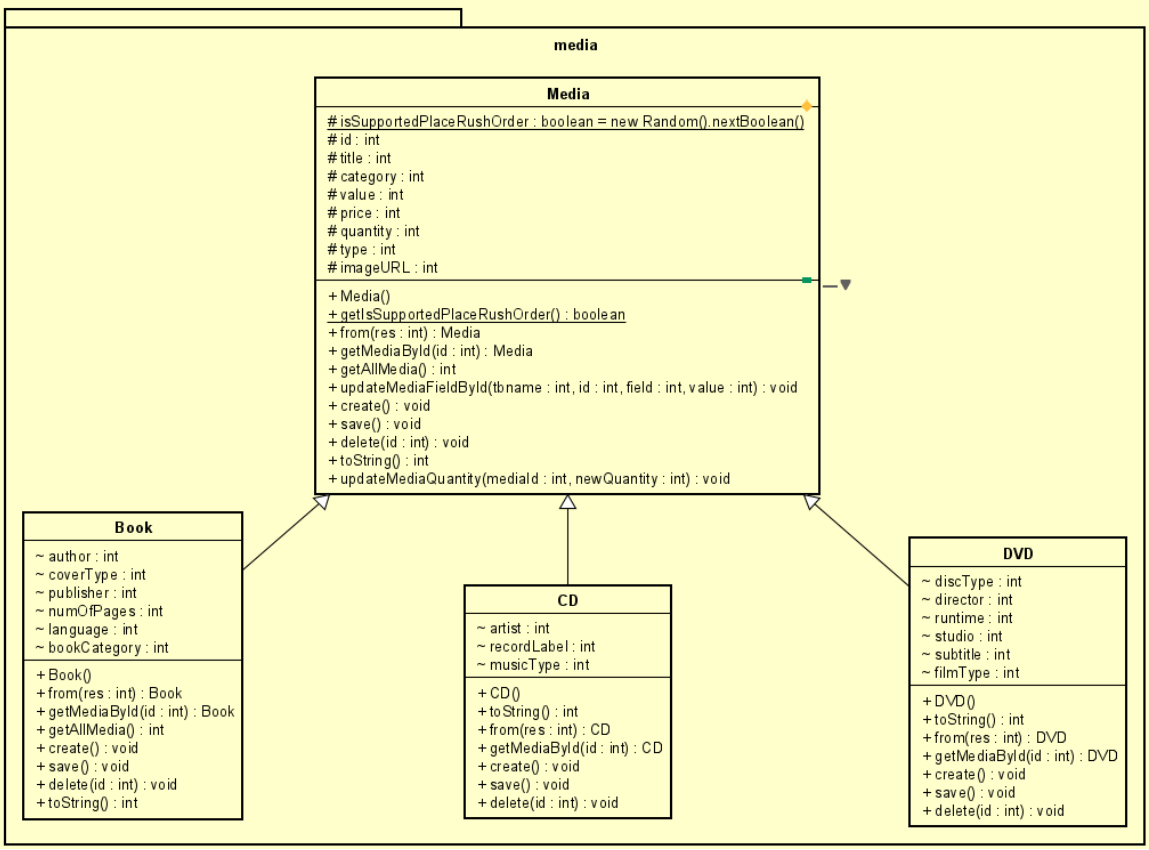
General model package class diagram



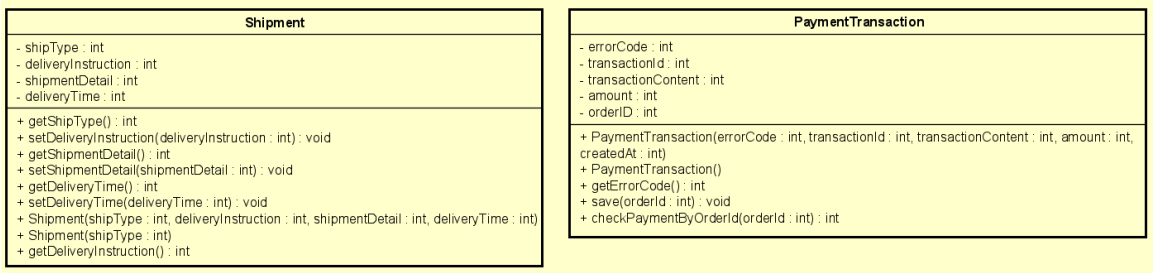
cart and order model



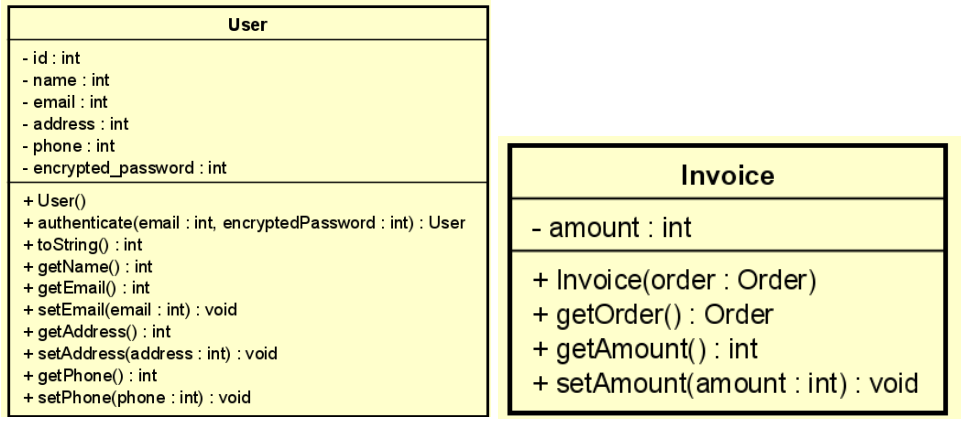
media package



shipment and paymentTransaction package

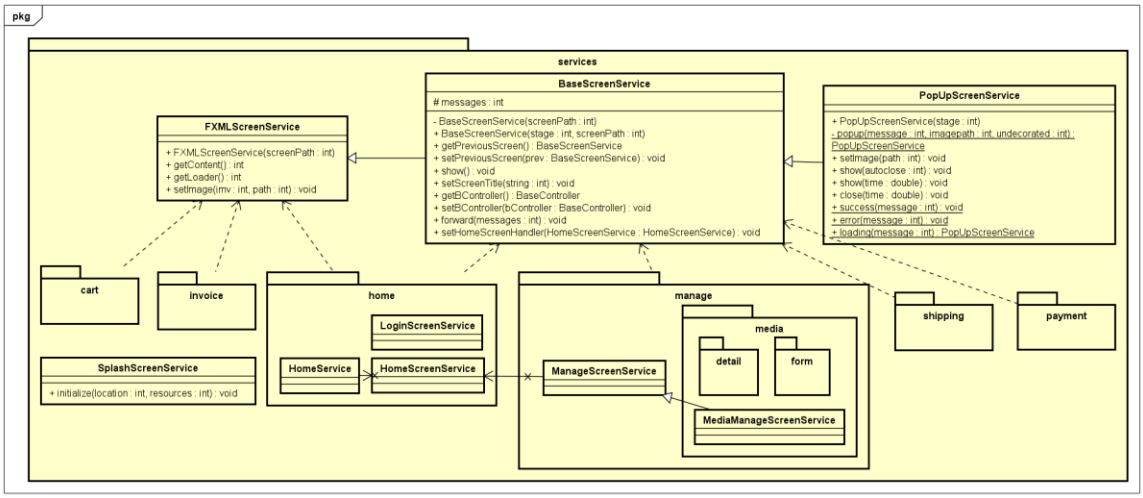


user and invoice package

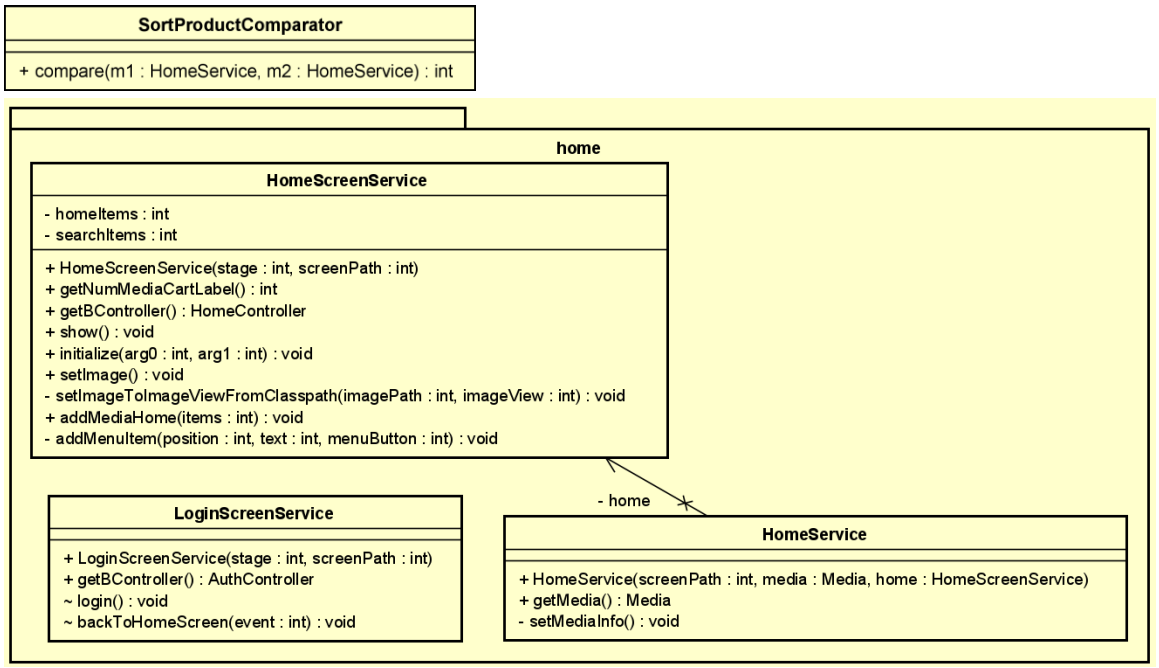


4.3.2.5 Class Diagram for Package services

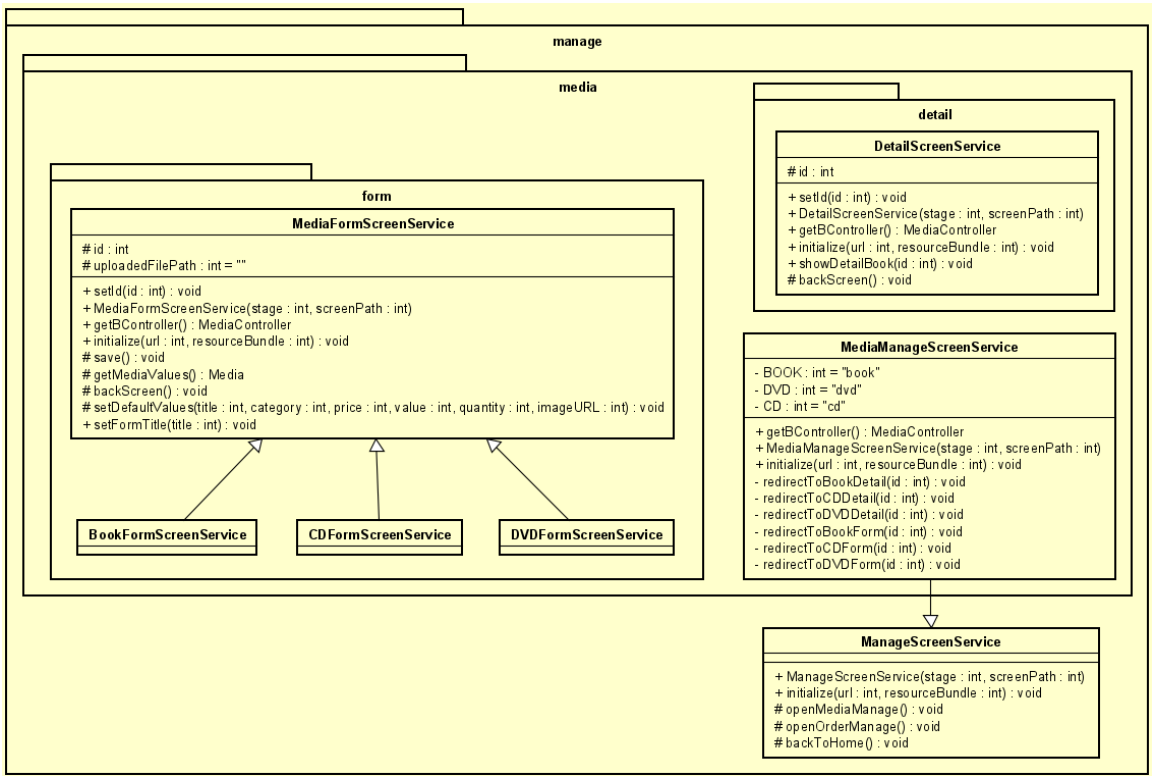
General package services class diagram



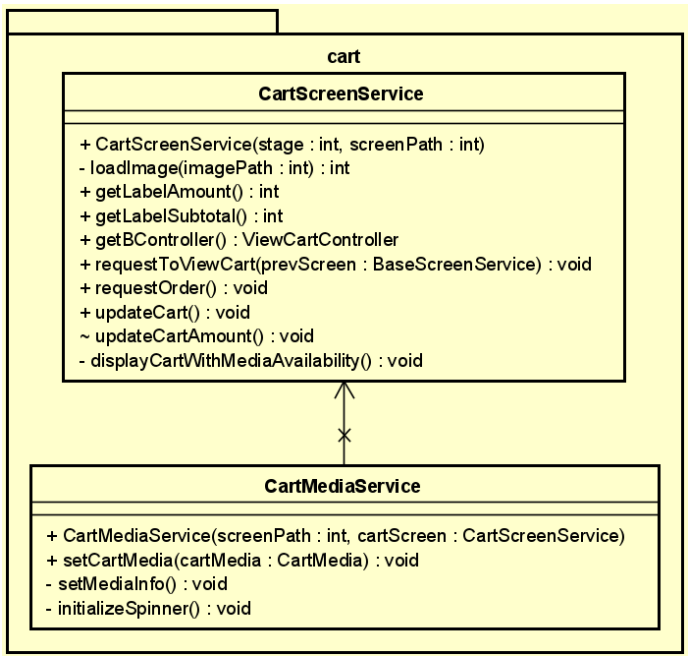
home package



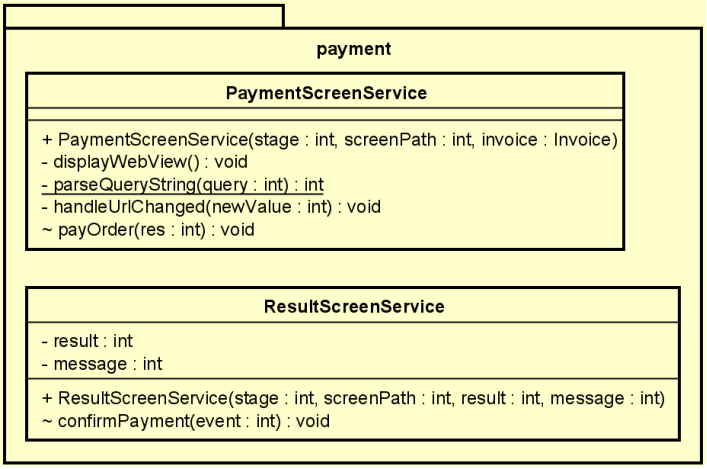
manage Package



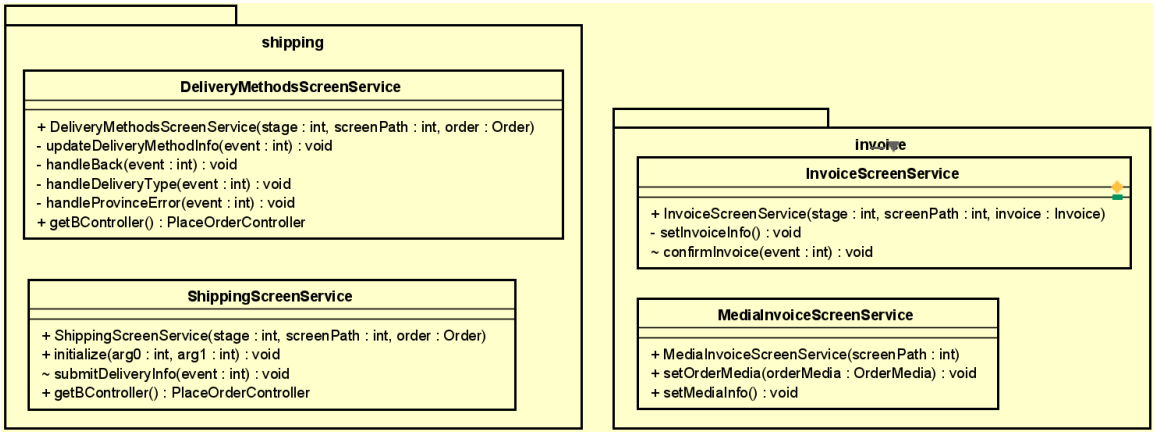
cart package



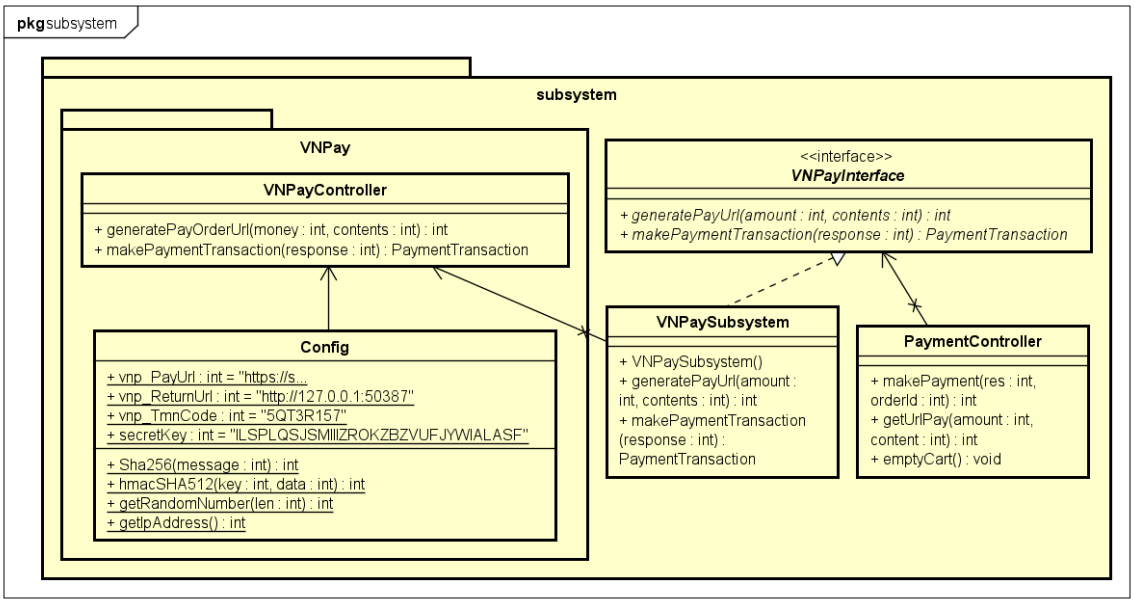
payment package



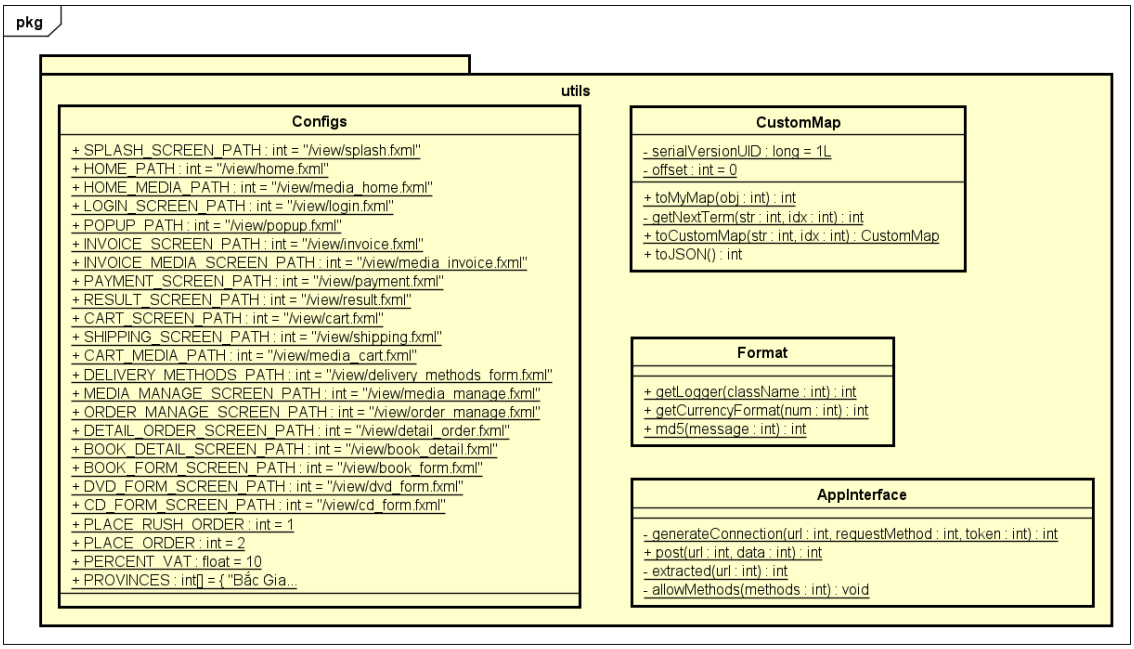
shipping and invoice package



4.3.2.6 Class Diagram for Package subsystem



4.3.2.7 Class Diagram for Package utils



4.3.3 Class Design

4.3.3.1 Class "BaseController"

Table 2. Example of operation design

#	*Name*	*Return type*	*Description (purpose)*

1	checkMediaInCart	CartMedia	Checks if a specific media item is in the cart.
2	getListCartMedia	int	Retrieves the list of media items in the cart.

Parameter Details

- **checkMediaInCart(media: Media): CartMedia**
  - **media**: The media item to check in the cart.
  - **Returns**: The corresponding CartMedia object if the media is in the cart, otherwise null.
- **getListCartMedia(): int**
  - **Returns**: The number of media items in the cart.

Exceptions

- **checkMediaInCart(media: Media): CartMedia**
  - **MediaNotFoundException** if the media item is not found in the cart.

Method

- **checkMediaInCart(media: Media): CartMedia**
  - **Purpose**: To determine if a given media item is already present in the cart and return its corresponding CartMedia object.
  - **Usage**: This method takes a Media object as a parameter and returns a CartMedia object if the media is found in the cart. If the media is not found, it may return null or throw a MediaNotFoundException.

4.3.3.2 Class "BaseScreenService"

Table 1. Attribute Design

#	Name	Data type	Default value	Description
1	messages	int	None	An integer representing the number of messages

Table 2. Operation Design

#	Name	Return type	Description (purpose)
1	BaseScreenService		Constructor that initializes the service with a screen path.

2	BaseScreenService		Constructor that initializes the service with a stage and a screen path.
3	getPreviousScreen	BaseScreenService	Retrieves the previous screen service instance.
4	setPreviousScreen	void	Sets the previous screen service instance.
5	show	void	Displays the screen.
6	setScreenTitle	void	Sets the screen title.
7	getBController	BaseController	Retrieves the base controller instance.
8	setBController	void	Sets the base controller instance.
9	forward	void	Forwards to the next screen with a specified number of messages.
10	setHomeScreenHandler	void	Sets the home screen handler with a specified HomeScreenService instance.

Parameter Details

- **BaseScreenService(screenPath: int)**
  - **screenPath**: The path to the screen to initialize the service.
  - **Returns**: An instance of BaseScreenService.
- **BaseScreenService(stage: int, screenPath: int)**
  - **stage**: The stage of the screen.
  - **screenPath**: The path to the screen to initialize the service.
  - **Returns**: An instance of BaseScreenService.
- **getPreviousScreen(): BaseScreenService**
  - **Returns**: The previous screen service instance.
- **setPreviousScreen(prev: BaseScreenService): void**
  - **prev**: The previous screen service instance to set.
  - **Returns**: None.

- **show(): void**
    - **Returns**: None.
- **setScreenTitle(title: string): void**
    - **title**: The title of the screen to set.
    - **Returns**: None.
- **getBController(): BaseController**
    - **Returns**: The base controller instance.
- **setBController(bController: BaseController): void**
    - **bController**: The base controller instance to set.
    - **Returns**: None.
- **forward(messages: int): void**
    - **messages**: The number of messages to forward.
    - **Returns**: None.
- **setHomeScreenHandler(homeScreenService: HomeScreenService): void**
    - **homeScreenService**: The home screen service instance to set as the handler.
    - **Returns**: None.

Exceptions

- **BaseScreenService(screenPath: int)**
    - No specific exceptions indicated.
- **BaseScreenService(stage: int, screenPath: int)**
    - No specific exceptions indicated.
- **getPreviousScreen(): BaseScreenService**
    - No specific exceptions indicated.
- **setPreviousScreen(prev: BaseScreenService): void**
    - No specific exceptions indicated.
- **show(): void**
    - No specific exceptions indicated.
- **setScreenTitle(title: string): void**

    - No specific exceptions indicated.
- **getBController(): BaseController**
    - No specific exceptions indicated.
- **setBController(bController: BaseController): void**
    - No specific exceptions indicated.
- **forward(messages: int): void**
    - No specific exceptions indicated.
- **setHomeScreenHandler(homeScreenService: HomeScreenService): void**
    - No specific exceptions indicated.

Method Details

BaseScreenService(screenPath: int)

- **Purpose**: To initialize the screen service with a specified screen path.
- **Usage**: This constructor takes an integer screen path as a parameter and initializes the BaseScreenService.

BaseScreenService(stage: int, screenPath: int)

- **Purpose**: To initialize the screen service with a specified stage and screen path.
- **Usage**: This constructor takes two integer parameters, stage and screen path, to initialize the BaseScreenService.

getPreviousScreen(): BaseScreenService

- **Purpose**: To retrieve the previous screen service instance.
- **Usage**: This method returns the BaseScreenService instance representing the previous screen.

setPreviousScreen(prev: BaseScreenService): void

- **Purpose**: To set the previous screen service instance.
- **Usage**: This method takes a BaseScreenService instance as a parameter and sets it as the previous screen.

show(): void

- **Purpose**: To display the screen.
- **Usage**: This method does not take any parameters and returns nothing. It simply displays the screen.

setScreenTitle(title: string): void

- **Purpose**: To set the screen title.
- **Usage**: This method takes a string title as a parameter and sets the screen title.

getBController(): BaseController

- **Purpose**: To retrieve the base controller instance.
- **Usage**: This method returns the BaseController instance associated with the screen service.

setBController(bController: BaseController): void

- **Purpose**: To set the base controller instance.
- **Usage**: This method takes a BaseController instance as a parameter and sets it as the base controller.

forward(messages: int): void

- **Purpose**: To forward to the next screen with a specified number of messages.
- **Usage**: This method takes an integer representing the number of messages and forwards to the next screen.

setHomeScreenHandler(homeScreenService: HomeScreenService): void

- **Purpose**: To set the home screen handler with a specified HomeScreenService instance.
- **Usage**: This method takes a HomeScreenService instance as a parameter and sets it as the home screen handler.

4.3.3.3 Class "Media"

Table 1. Attribute Design

#	Name	Data type	Default value	Description
1	id	int	None	The unique identifier for a media item
2	title	int	None	The title of the media item
3	category	int	None	The category the media item belongs to
4	value	int	None	The value of the media item

5	price	int	None	The price of the media item
6	quantity	int	None	The quantity available of the media item
7	type	int	None	The type of the media item
8	imageURL	int	None	The URL for the image of the media item
9	isSupportedPlaceRushOrder	boolean	new Random().nextBoolean()	Indicates if rush orders are supported for the item

Table 2. Operation Design

#	**Name**	**Return type**	**Description (purpose)**
1	Media		Constructor for the Media class
2	getIsSupportedPlaceRushOrder	boolean	Returns if rush orders are supported for this media item
3	from	Media	Creates a Media object from a resource
4	getMediaById	Media	Retrieves a Media object by its ID
5	getAllMedia	int	Retrieves all Media objects
6	updateMediaFieldById	void	Updates a specific field of a Media object by its ID
7	create	void	Creates a new Media object
8	save	void	Saves the Media object
9	delete	void	Deletes a Media object by its ID
10	toString	int	Returns a string representation of the Media object
11	updateMediaQuantity	void	Updates the quantity of a Media object by its ID

Parameter Details

- **from(res: int): Media**
  - **res**: An integer representing the resource.
  - **Returns**: A Media object created from the resource.
- **getMediaById(id: int): Media**
  - **id**: The unique identifier of the media item.
  - **Returns**: The Media object with the specified ID.
- **updateMediaFieldById(tbname: int, id: int, field: int, value: int): void**
  - **tbname**: The name of the table.
  - **id**: The ID of the media item.
  - **field**: The field to be updated.
  - **value**: The new value for the specified field.
  - **Returns**: None.
- **updateMediaQuantity(mediaId: int, newQuantity: int): void**
  - **mediaId**: The ID of the media item.
  - **newQuantity**: The new quantity to be set.
  - **Returns**: None.

Exceptions

- **from(res: int): Media**
  - **ResourceNotFoundException** if the resource does not exist.
- **getMediaById(id: int): Media**
  - **MediaNotFoundException** if no media with the specified ID is found.
- **updateMediaFieldById(tbname: int, id: int, field: int, value: int): void**
  - **FieldUpdateException** if the field cannot be updated.
- **updateMediaQuantity(mediaId: int, newQuantity: int): void**
  - **QuantityUpdateException** if the quantity cannot be updated.

Method Details

Media()

- **Purpose**: Constructor to initialize a Media object.

- **Usage**: This is used to create a new instance of a Media object.

getIsSupportedPlaceRushOrder(): boolean

- **Purpose**: To check if rush orders are supported for this media item.
- **Usage**: This method returns a boolean value indicating if rush orders are supported.

from(res: int): Media

- **Purpose**: To create a Media object from a specified resource.
- **Usage**: This method takes an integer resource and returns a Media object created from it.

getMediaById(id: int): Media

- **Purpose**: To retrieve a Media object by its unique identifier.
- **Usage**: This method takes an integer ID and returns the corresponding Media object.

updateMediaFieldById(tbname: int, id: int, field: int, value: int): void

- **Purpose**: To update a specific field of a Media object by its ID.
- **Usage**: This method takes the table name, media ID, field to be updated, and the new value for the field.

create(): void

- **Purpose**: To create a new Media object.
- **Usage**: This method initializes and creates a new media item.

save(): void

- **Purpose**: To save the current state of the Media object.
- **Usage**: This method saves any changes made to the Media object.

delete(id: int): void

- **Purpose**: To delete a Media object by its unique identifier.
- **Usage**: This method takes an ID and deletes the corresponding Media object.

toString(): int

- **Purpose**: To return a string representation of the Media object.
- **Usage**: This method returns an integer which is a string representation of the Media.

updateMediaQuantity(mediaId: int, newQuantity: int): void

- **Purpose**: To update the quantity of a Media object by its ID.
- **Usage**: This method takes the media ID and the new quantity to set for the media item.

5 Design Considerations

5.1 Goals and Guidelines

Objectives:

- Provide a user-friendly application.
- Provide an eye-catching interface and convenient experience for users.
- The response time for the system is 1 second at normal and 2 seconds during a peak load.

Strategies:

- Obligate the coding convention in Java, and OOP principles.
- Avoid hard-coding.
- Write comments for codes.
- Structure the documents for maintenance.

5.2 Architectural Strategies

Our goal is to reuse components effectively. Here are the tools and technologies we are using:

- Programming Language: Java
- Database: SQLite
- UML Tool: Astah
- GUI Builder: Scene Builder

We aim to minimize memory and space usage, reduce complexity to enhance response times, and improve overall performance. We also prioritize ease of maintenance. Looking ahead, we plan to update the system with new features, including admin capabilities for managing CRUD operations, generating statistics, and tracking profits.

5.3 Coupling and Cohesion

5.3.1. Coupling

5.3.1.1. Common coupling

This is the case where 2 modules share the same data, 2 global structures can be edited, accessed, or share the same source code blocks, which will violate common coupling. However, object-oriented programming does not have common data. All data belongs to class. Therefore, our project does not violate common coupling.

Related modules	Description	Improvement
No related module	Our modules are self-contained and do not rely on other modules to operate	No improvement

5.3.1.2. Control coupling

A module violates control coupling when it passes control parameters to other modules through method calls. This is not good because the called component will know the structure inside the calling component and when this structure is changed, the called component will have to change accordingly.

Related modules	Description	Improvement
No related module	Our methods are designed to carry out only one specific task, so no control coupling existed	No improvement

5.3.1.3. Stamp coupling

Two classes are considered to violate stamp coupling if one class sends a collection or an object as a parameter and only some portion of the data is used by the second class.

Related modules	Description	Improvement
Controller module	HomeController passes the Model object and ViewCartController passes the Cart object, but do not use all of its properties, this is still acceptable so that the method does not have too many parameters passed in, which will cause confusion.	No improvement

5.3.1.4. Data coupling

Two components are data coupled if there are homogeneous data items.

Related modules	Description	Improvement
Controller and srervices modules	Services need data to render GUI, which is acceptable	No improvement

5.3.2. Cohesion

5.3.2.1. Coincidental cohesion

Sub-components are placed in one component randomly.

Related modules	Description	Improvement
No module		No improvement

5.3.2.2. Logical cohesion

The components in a module are related to each other logically, not functionally.

Related modules	Description	Improvement
No module		No improvement

5.3.2.3. Temporal cohesion

Elements of a component are related by timing.

Related modules	Description	Improvement
App class	The App class performs the function of displaying Splash screen first and then displaying Home screen.	No improvement

5.3.2.4. Procedure cohesion

Components are placed in a module because they are closely related to each other in a certain order but are not functionally related to each other.

Related modules	Description	Improvement
No module		No improvement

5.3.2.5. Communicational cohesion

Components are placed in the same module because they operate on the same data

Related modules	Description	Improvement
MediaController, ViewCartController, ManageCartController	Having different methods of performing functions but they all operate on the same Order, Cart data.	No improvement

5.3.2.6. Sequential cohesion

In a module, the output of one component is the input of another.

Related modules	Description	Improvement
No module		No improvement

5.3.2.7. Information cohesion

Operations are independent (have their own input and output but they can operate on a common data set that is an attribute of that class).

Related modules	Description	Improvement
utils, database	All methods are to perform database queries or manipulate json string	No improvement

5.3.2.8. Functional cohesion

Each subcomponent performs a certain job and addresses the overall purpose of that component

Related modules	Description	Improvement
Most of the modules		No improvement

In our software design, we have identified some components that still exhibit Control Coupling issues. We have tried our best to resolve these problems to reduce coupling and increase cohesion. However, due to time constraints, we may not be able to resolve these issues before the announced deadline.

5.4 Design Principles

We design simple classes following SOLID principles, which means:

- Each class has a single responsibility or job.
- Software entities are open for extension but closed for modification.
- We also use interfaces classes, ensuring that subclasses can be substituted for their base classes.
- Specific interfaces are used as needed, instead of general-purpose interfaces that are not utilized.
- We organize classes with similar properties into the same package for easier management. This approach allows us to reuse source code and adapt to changing requirements efficiently.

5.4.1 Single Responsibility Principle

A class should have only one reason to change, meaning it should only have one job or responsibility. This helps to make the system more understandable, easier to maintain, and less prone to errors.

However, in the project, some classes in controller package such as ManageOrderController or PlaceOrderController violate this principle because they perform many different functions for managing order or placing order.

5.4.2 Open/Closed Principle

Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. This allows the behavior of the system to be extended without altering its source code, reducing the risk of introducing new bugs.

Currently there is no example satisfying this principle.

5.4.3 Liskov Substitution Principle

Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program. This ensures that a subclass can stand in for its superclass without the client code needing to know the difference, polymorphism.

The inheritance hierarchy in the Media class follows this principle because the subclasses of Media including CD, DVD, Book all perform the functions of the Media class.

5.4.4 Interface Segregation Principle

Instead of using an overly large interface with too many methods, we divide it into smaller subinterfaces with specific purposes. This approach prevents implementing classes from having to implement methods they do not actually need.

Our project does not violate this principle because no interface has too many methods.

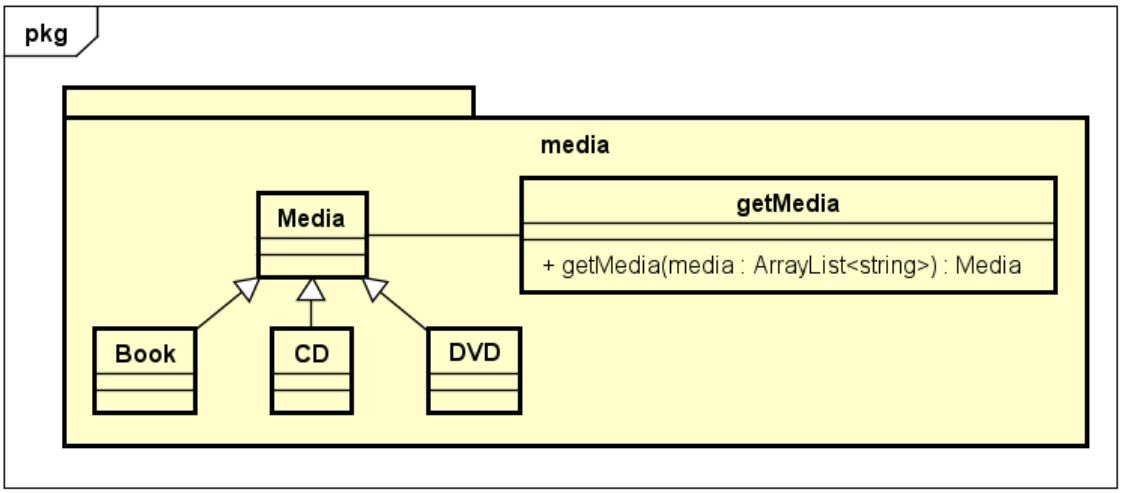
5.4.5 Dependency Inversion Principle

High-level modules should not depend on low-level modules. Both should depend on abstractions. Additionally, abstractions should not depend on details. Details should depend on abstractions.

Currently, screen display classes in services package depend quite strictly on several model classes, which leads to this principal violation.

5.5 Design Patterns

*** ***Factory (proposed)***



Useage: When a superclass (Model) with many subclasses exists and based on the input we need to return a subclass. This pattern helps us take the responsibility of an initialization class from the user (client) side to the Factory class.

Benefit:

- Reduce dependencies between modules
- Easy to add more subclasses in the future.
- Able to initialize objects but hides the logic of initialization. Users do not know the actual logic initialized below the factory method.

Member assignment

No	Member	Task	Percentage
1	Đinh Việt Quang	GUI, service - controller - exception class, UC search product, UC CRUD cart product	25%
2	Ngô Minh Quý	Service class, UC log in	15%
3	Trịnh Diễm Quỳnh	Database - execption class, UC cancel order, UC review order	20%
4	Hồ Nam Sơn	App - utils class, UC CRUD cart prouct	15%
5	Lê Phú Tài	Model - services class, UC place order, UC pay order, UC place rush order	25%