HUST UNIVESITY OF SCIENCE AND TECHNOLOGY
The School of Information and Communication Technology

Capstone Project

ITSS SOFTWARE DEVELOPMENT

Guidance Teacher: Nguyen Thi Thu Trang

Group : 05

Members:

Mai Hoang Duc	20215195
Ngo Nhu Dien	20194542
Cao Huy Dong	20210188
Nguyen Tho Dat	20215193
Tong Tran Minh Duc	20205147

Hanoi, 06, 2024

Table of Contents

1 Introduction

1.1 Objective

The objective of this Software Design Document (SDD) is to provide a detailed description of the design for the AIMS (An Internet Media Store) application. This document will serve as a comprehensive blueprint for the development team to implement the software and for the testing team to verify that the software operates as expected. The SDD also aids project managers and stakeholders in understanding the scope, design, and associated risks of the software development process..

1.2 Scope

The AIMS application will function as a 24/7 e-commerce platform allowing users to buy and sell physical media products such as books, CDs, LP records, and DVDs. The application will support up to 1,000 simultaneous users without significant performance degradation and can operate continuously for 300 hours without failure, with a recovery time of one hour after an incident.

The main features of the AIMS application include:

- Product management capabilities for product managers, allowing them to add, view, edit, or delete products.
- User management features for administrators, including creating, viewing, updating, and deleting user accounts, resetting passwords, and managing user roles.
- A user-friendly interface for customers to browse, search, and sort products, add items to a shopping cart, and complete purchases.
- Order processing functionality that includes inventory checks, delivery information setup, and integration with VNPay for payment processing.

The primary goal of the AIMS application is to provide a seamless and efficient e-commerce experience for both product managers and customers, facilitating the buying and selling of physical media products while ensuring robust performance and security.

1.3 Glossary

Term	**Explaination**
AIMS	The name of the e-commerce application being developed.

User	An individual who uses the AIMS application to buy or manage media products.
Product Manager	A user role responsible for adding, viewing, editing, or deleting media products within the AIMS application.
Administrator	A user role responsible for managing user accounts and system settings within the AIMS application.

Media Product	A virtual cart where customers can add media products for purchase.
Order	A confirmed purchase request made by a customer within the AIMS application.
Delivery Fee	The charge applied to an order based on the delivery location and weight of the products.
Rush Order	A delivery option allowing customers to receive their items within a prearranged timeframe of 2 hours for addresses within the inner city of Hanoi.
Inventory	The stock of media products available for sale within the AIMS application.
VAT	Value-Added Tax, a 10% tax applied to the sale of media products within the AIMS application.
VNPay	The payment gateway integrated with the AIMS application for processing credit card transactions.
Database	A collection of data that is organized and stored in a way that allows for efficient retrieval and manipulation.
UI	User Interface, the visual elements and design of the application through which users interact with the software.

1.4 References

CaseStudy - Google Drive

2 Overall Description

2.1 Survey

List of actors and description:

- User: The main actor in the application. Users interact with the system to browse and purchase media products, manage their shopping cart, and complete orders.
- Product Manager: A user responsible for managing media products within the application, including adding, viewing, editing, and deleting products.
- Administrator: A user responsible for managing the application's user accounts, roles, and settings, including creating, viewing, updating, and deleting user information.
- VNPay: The payment gateway system integrated into AIMS to facilitate secure online transactions. This actor handles the actual processing of payment requests and refunds..

2.2 Overall requirements

The following use case diagram illustrates the primary interactions between the actors and the system:


uc
AIMS Project
Place Order
<<extend>>
Place Rush Order
<<include>>
Pay Order
VNPAY
Cancel Order
CRUD Products
Product Manager
Browse Products
Customer
Manage Cart
Manage Orders
Login
Sign up

2.3 Assumption/Constraints/Risks

Assumptions

The AIMS project, a desktop e-commerce software, operates under several key assumptions:

- **Compatibility**: The software is expected to run on common desktop operating systems like Windows, macOS, and popular Linux distributions, suitable for home and small business hardware.
- **Internet Dependence**: A stable internet connection is required for browsing the catalog and making purchases using VNPay.
- **User Familiarity**: The system is designed for users with some experience in online shopping, offering an intuitive interface similar to common e-commerce platforms.
- **Content Licensing**: All media content available on AIMS is properly licensed from content providers, adhering to licensing and distribution agreements.
- **Hardware Requirements**: The software is assumed to have modest hardware requirements, making it compatible with most consumer-grade desktops and laptops.

Constraints

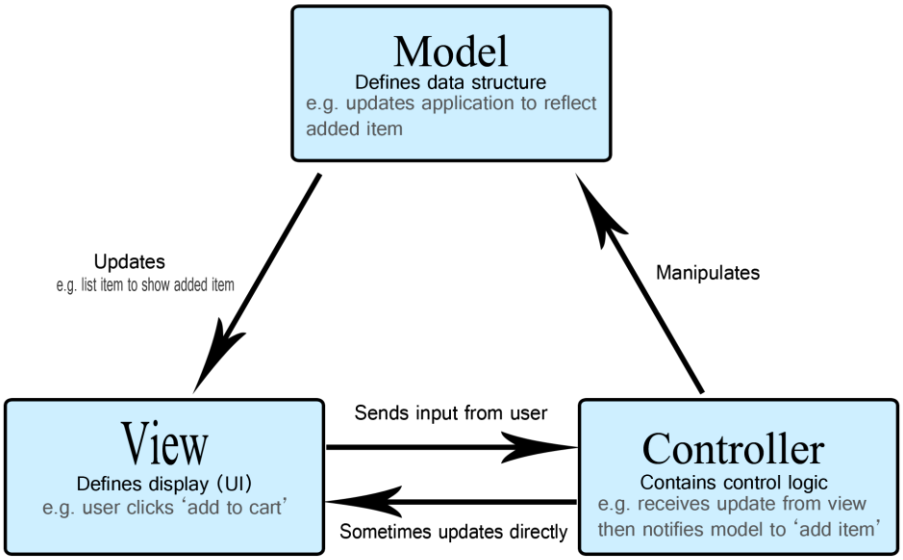
- **Internet Requirement for VNPay**: Users must have an active internet connection to access VNPay functionality.
- **Product Image Limitations**: Users can only select from predefined product images, not upload custom images, to ensure consistency in the user experience.

Risks

- **Unauthorized Account Access**: There is a risk of unauthorized access to user accounts. Mitigation measures include strong authentication and authorization controls, such as role-based permissions and activity logging.
- **Excessive User Interactions**: Rapid user interactions with VNPay could cause errors.

3 System Architecture and Architecture Design

3.1 Architectural Patterns

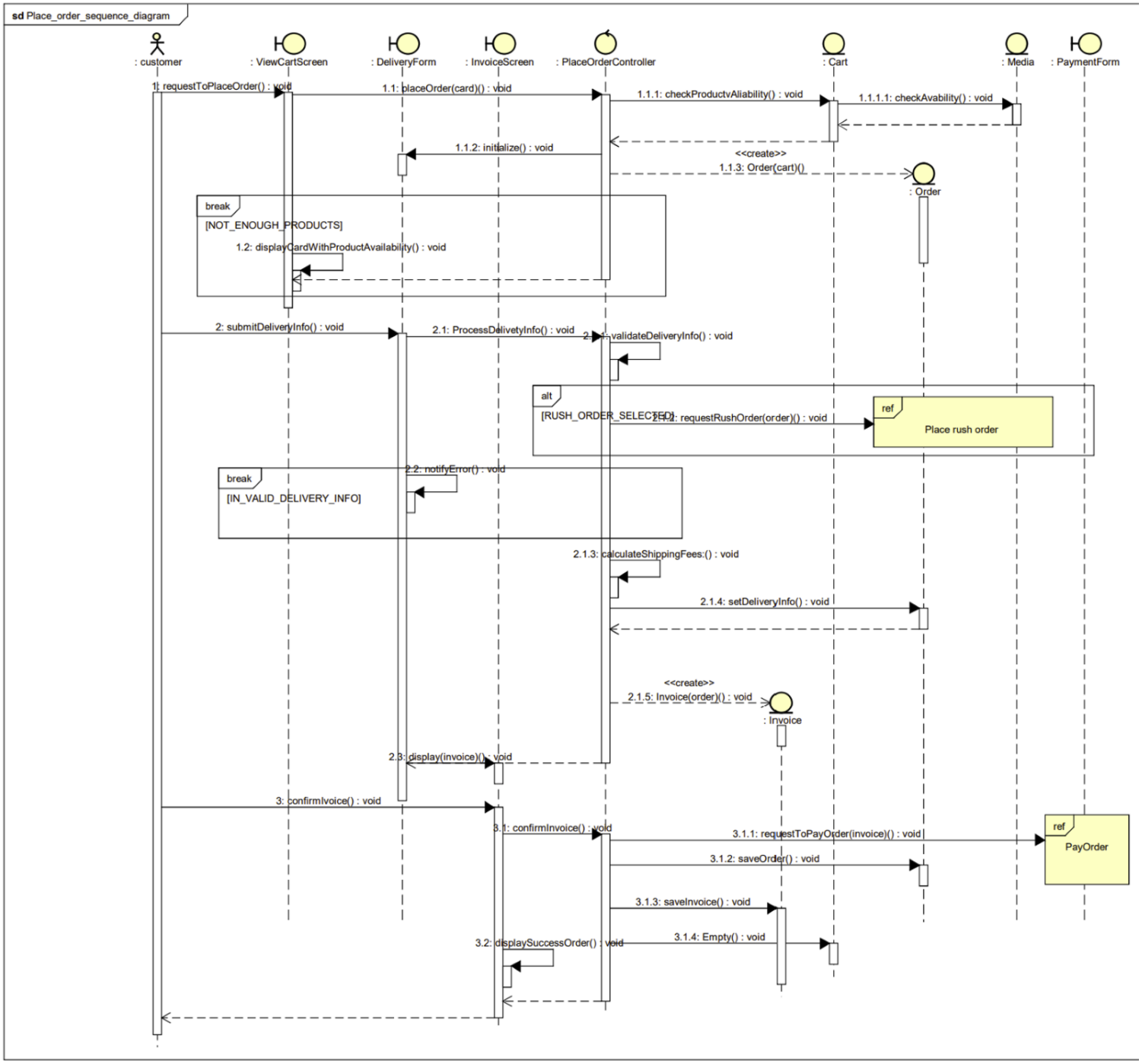


We used the Entity-Views-Controller (EVC) pattern, a variation of the Model-View-Controller (MVC) pattern. In this pattern:

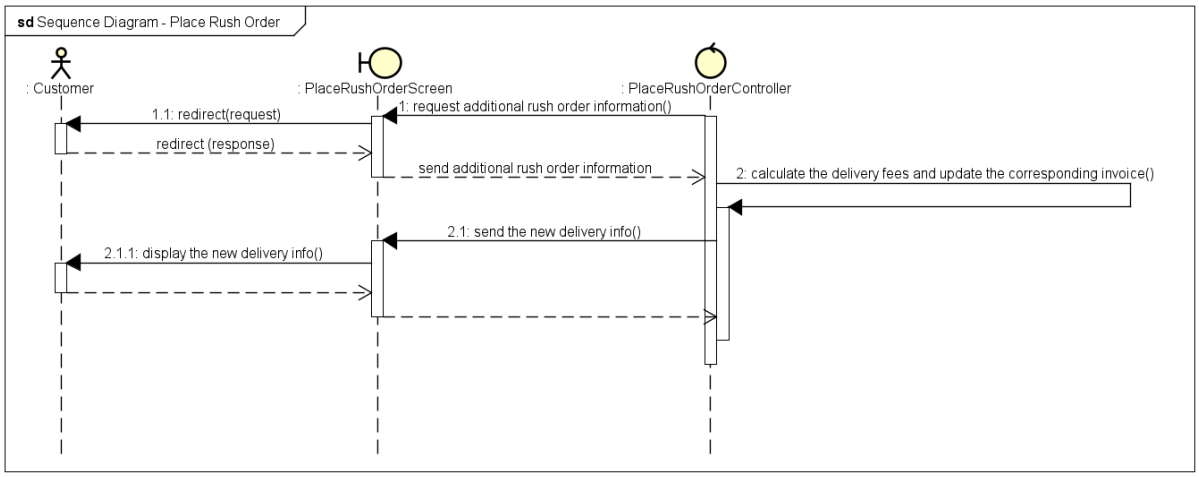
- **Entity**: Represents the core business logic and data of the application. Entities encapsulate the data and the rules that govern access to and updates of this data. They are similar to the Model in MVC.
- **Views**: Handle the presentation layer of the application. Views are responsible for displaying the data provided by the entities to the user. They ensure that the data is presented in a meaningful way, often formatting or transforming it for display. They are similar to the View in MVC.
- **Controller**: Manages the user input and interactions, updating the entities and views accordingly. Controllers handle the logic that determines how the application responds to user actions, typically by calling methods on entities and updating views. They are similar to the Controller in MVC.

3.2 Interaction Diagrams

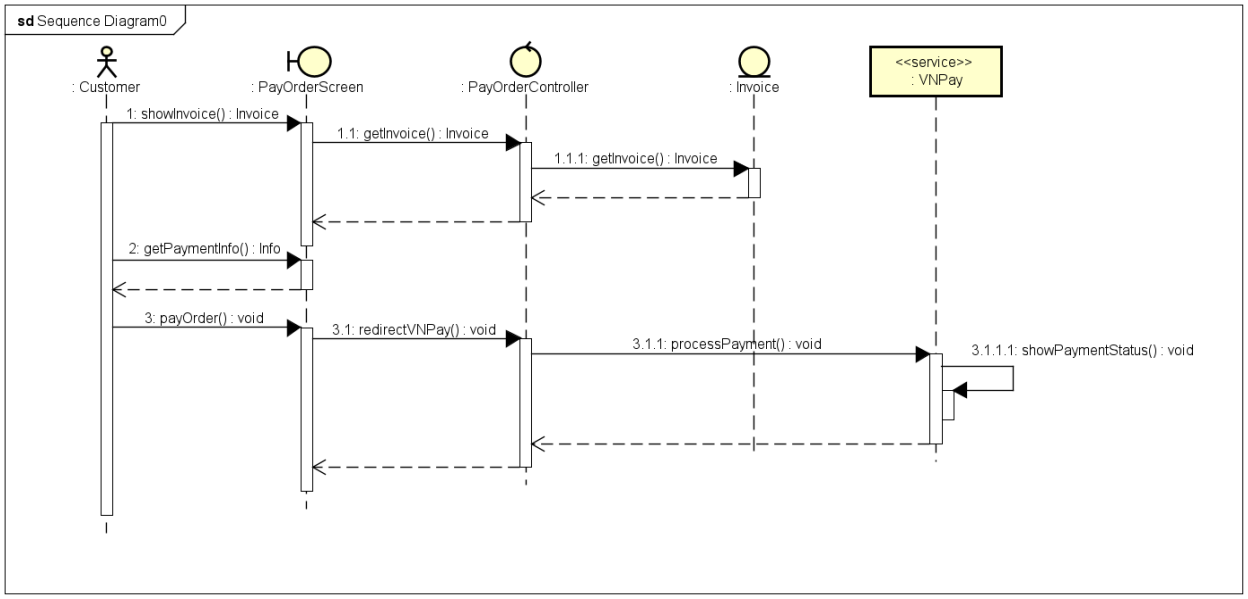
3.2.1 Place Order



3.2.2 Place Rush Order

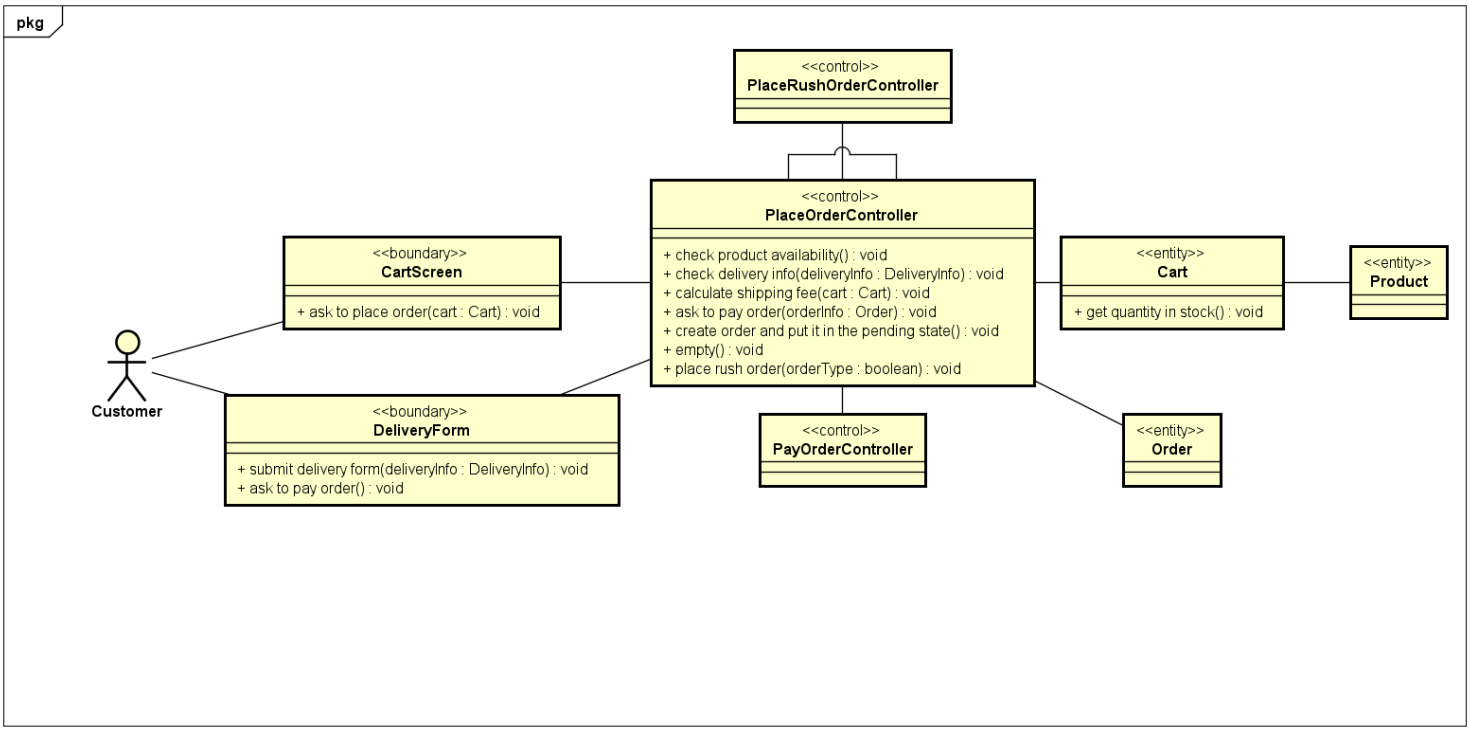


3.2.3 Pay Order

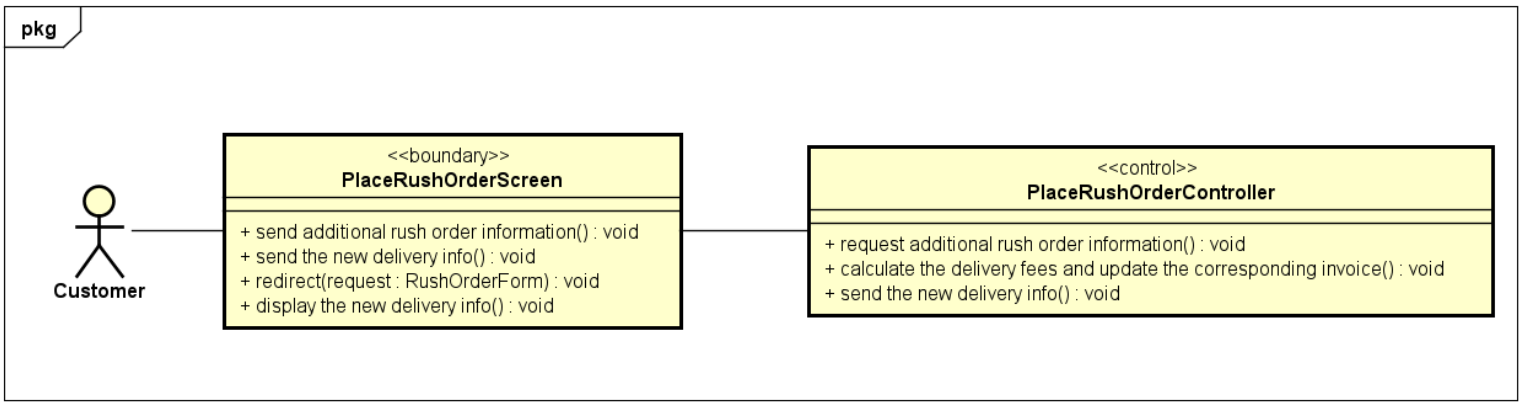


3.3 Analysis Class Diagrams

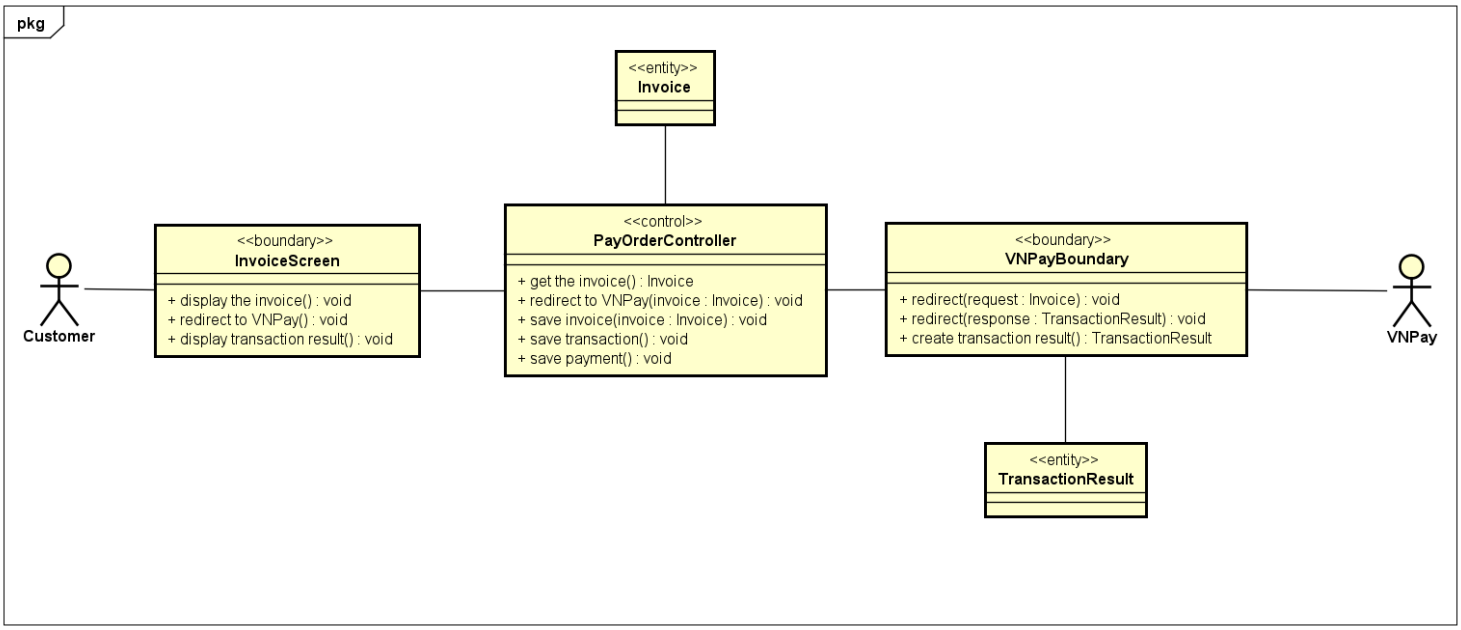
3.3.1 Place order



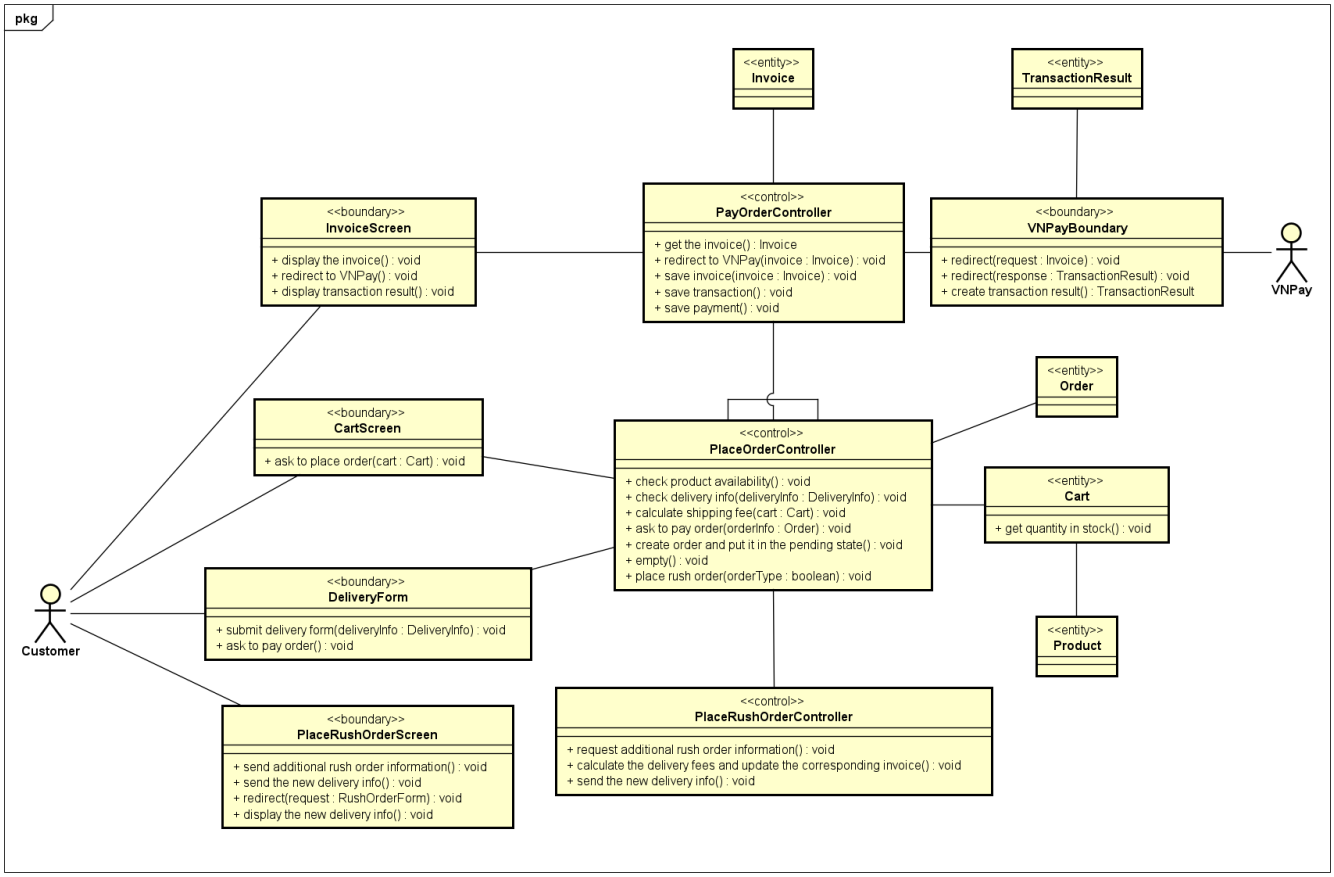
3.3.2 Place rush order



3.3.3 Pay order



3.4 Unified Analysis Class Diagram



3.5 Security Software Architecture

Given the scope of the project, a simple login/logout system has been implemented. Below is a detailed description of the security components and configuration:

Authentication:

- **User Authentication**: A basic login system is used to validate user identities before granting access to the system. Users must enter a valid username and password to log in. Passwords are stored securely in the SQLite database to prevent unauthorized access.
- **Session Management**: Upon successful login, a user session is created and maintained until the user logs out. Sessions are managed using unique session IDs to track user activities and ensure that each session is securely maintained.

4 Detailed Design

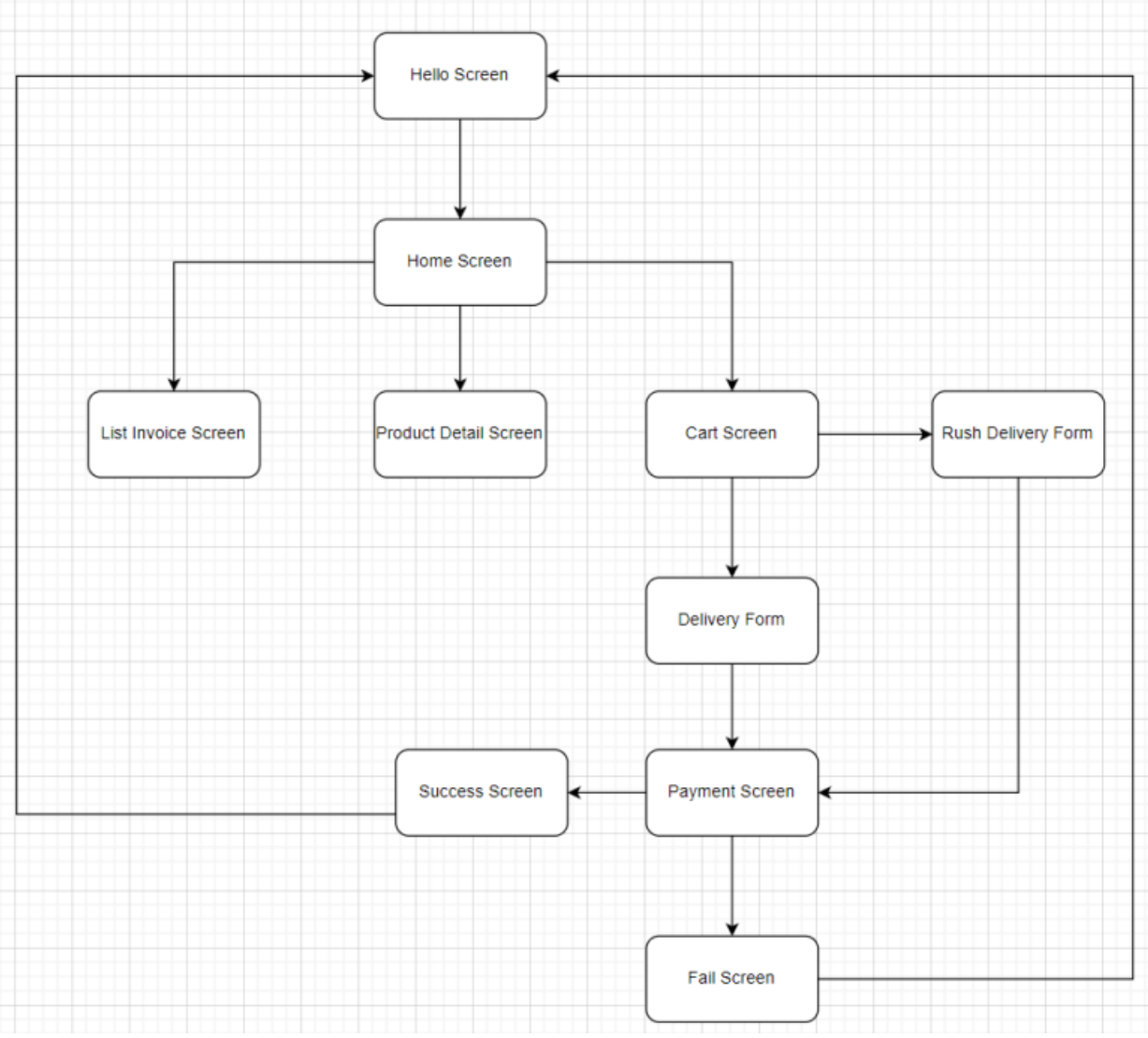
4.1 User Interface Design

4.1.1 Screen Configuration Standardization

Display:

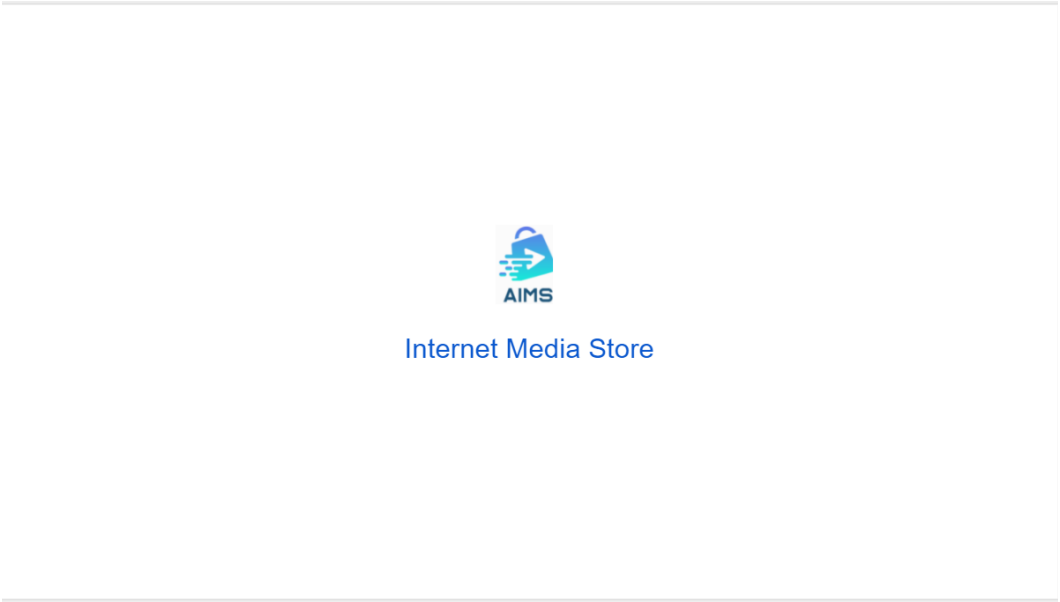
- Number of colors supported: 16,777,216 colors
- Resolution: 1366 x 768 pixels

4.1.2 Screen Transition Diagrams

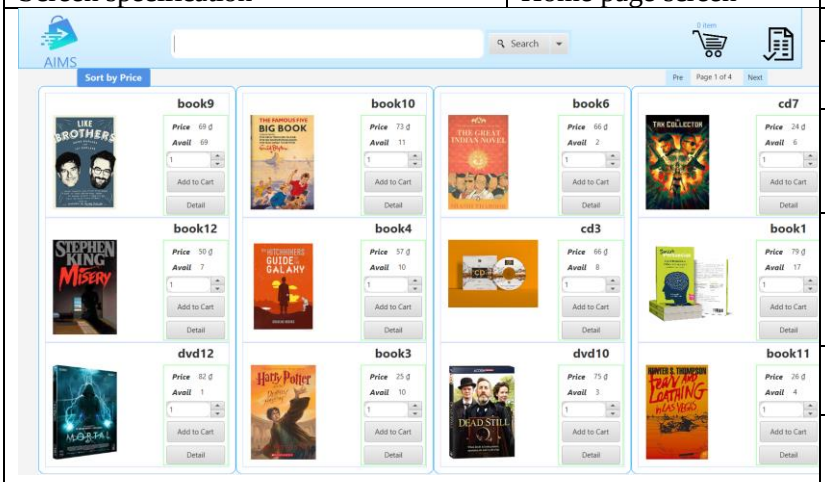


4.1.3 Screen Specifications

Hello Page



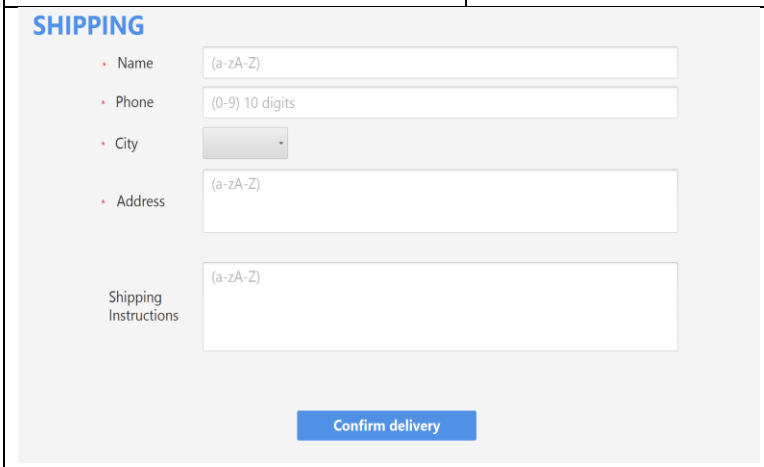
Home Page

AIMS Software		Date of creation		
Screen specification	Home page screen	15/6/2024		
		Control	Operator	Function
		Search Input Area	Type	Input Search Keyword
		Sort by button	Click	Sort media in a chosen order
		Search Button	Click	Search Media with input in search area
		Cart Button	Click	Display Cart Page
		Area for displaying Banner	Initial	Display Banner
		Area for displaying Media	Initial	Display Media
		Next Button	Click	Display next page
		Pre Button	Click	Display previous page

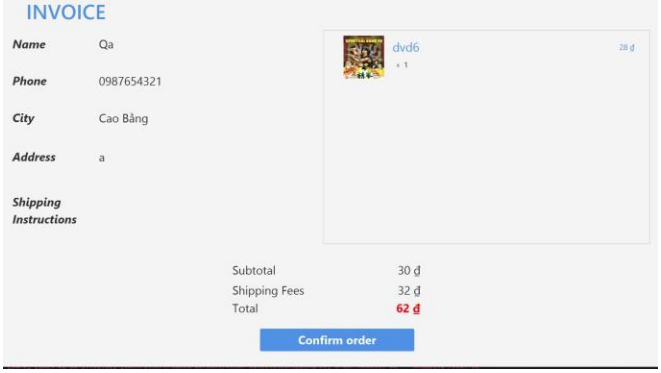
Cart Page

AIMS Software		Date of creation		
Screen specification	Cart screen	15/6/2024		
		Control	Operator	Function
		Area for displaying Price	Initial	Display price
		Area for displaying Media	Initial	Display Media
		Place order Button	Click	Display Delivery Form
		Delete Button	Click	Remove product from cart

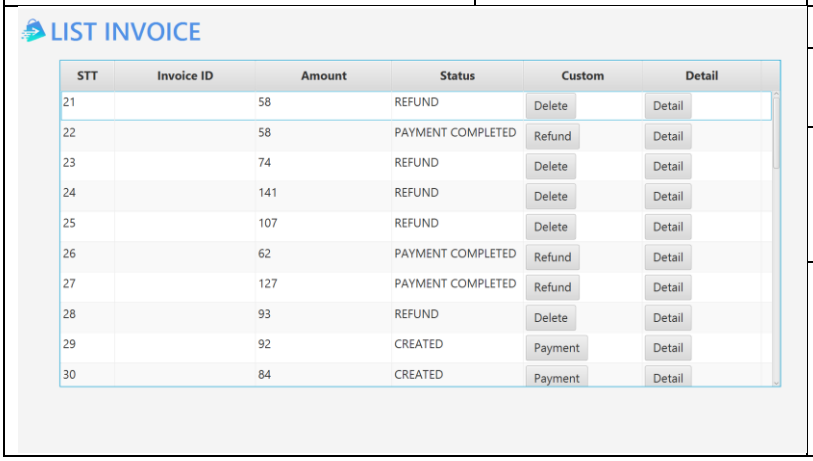
Delivery Information Form Page

AIMS Software		Date of creation		
Screen specification	Shipping screen	15/6/2024		
		Control	Operator	Function
		Name Input area	Type	Set name reciever
		Phone input area	Type	Set phone reciever
		Province Input area	Choose	Set province reciever
		Address Input area	Type	Set address reciever
		Instruction Input area	Type	Set instruction reciever
		Submit button	Click	Send form and display Rush Order Form Page or Invoice Page

Invoice Page

<table>
<tr><td colspan="2">AIMS Software</td><td>Date of creation</td><td colspan="2"></td></tr>
<tr><td colspan="2">Screen specification</td><td>Home page screen</td><td>3/6/2024</td><td colspan="2"></td></tr>
<tr><td colspan="2" rowspan="5"> 
</td><td>Control</td><td>Operator</td><td>Function</td></tr>
<tr><td>Area for displaying Price</td><td>Initial</td><td>Display price</td></tr>
<tr><td>Area for displaying Delivery Information</td><td>Initial</td><td>Display Delivery Information</td></tr>
<tr><td>Area for displaying Media</td><td>Initial</td><td>Display Media</td></tr>
<tr><td>Confirm order Button</td><td>Click</td><td>Display Payment Page</td></tr>
</table>

List Invoice Page

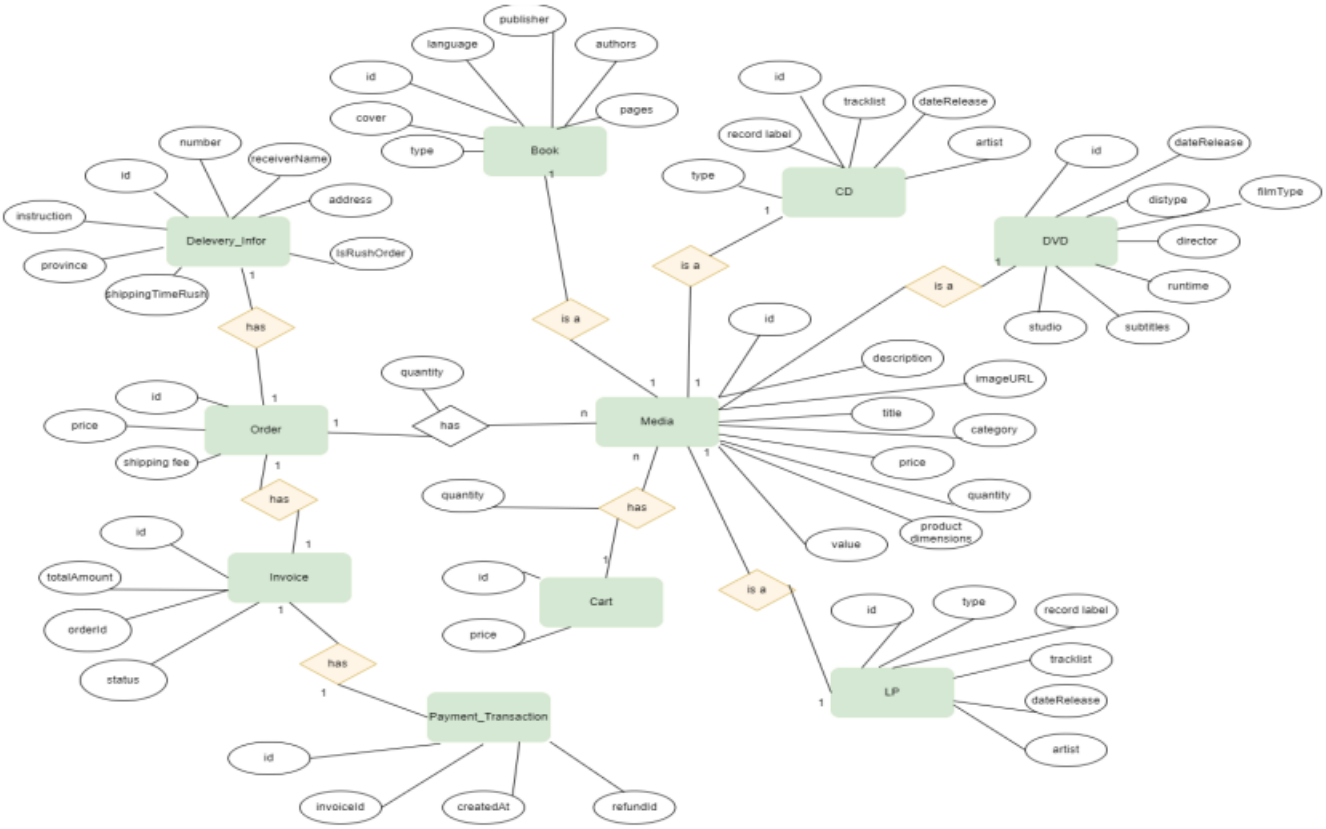
<table>
<tr><td colspan="2">AIMS Software</td><td>Date of creation</td><td colspan="2"></td></tr>
<tr><td colspan="2">Screen specification</td><td>Home page screen</td><td>12/6/2024</td><td colspan="2"></td></tr>
<tr><td colspan="2" rowspan="4"> 
</td><td>Control</td><td>Operator</td><td>Function</td></tr>
<tr><td>Delete Button</td><td>Click</td><td>DeleteInvoice</td></tr>
<tr><td>Refund Button</td><td>Click</td><td>RefundOrder</td></tr>
<tr><td>Payment Buton</td><td>Click</td><td>PayOrder</td></tr>
</table>

4.2 Data Modeling

4.2.1 Conceptual Data Modeling

Data modeling is the process of creating a visual representation of an entire information system or parts of it to communicate the connections between data points and structures. The goal is to illustrate the types of data used and stored within the system, the relationships between these data types, how the data can

be grouped and organized, as well as its formats and attributes.



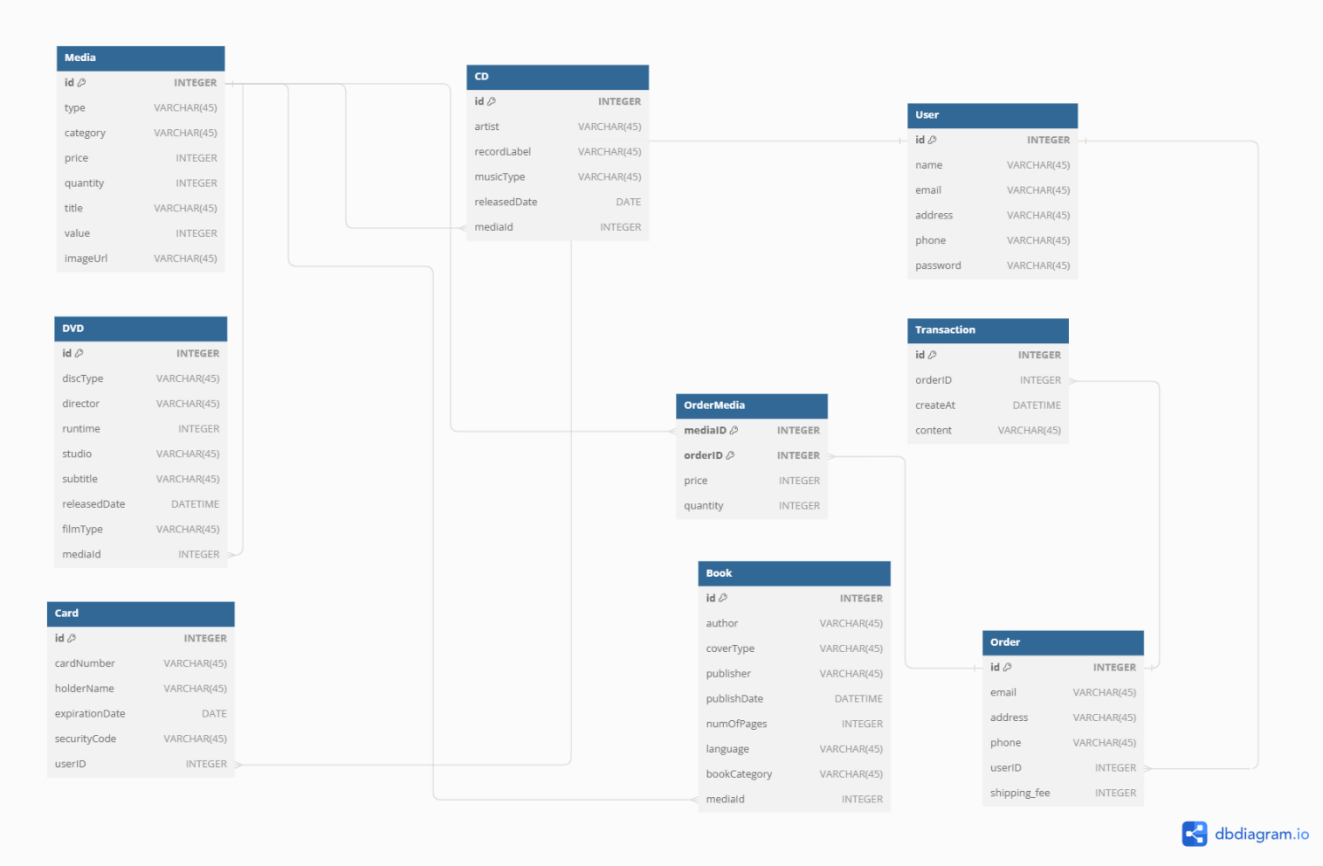
4.2.2 Database Design

4.2.2.1 *Database Management System*

To manage the information of the AIMS application, the system uses SQLite as the database management system.

SQLite is a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a popular choice for embedded database systems due to its simplicity, lightweight nature, and ease of setup. SQLite provides robust features such as support for various data types, data security, query optimization, user management, data backup, and recovery, along with scalability and customization capabilities.

In the AIMS application, SQLite is used to store information about products and their attributes, rental history, payment transactions, deposits, invoices, and more. With SQLite, the application can efficiently and securely search, add, modify, and delete information in the database. Additionally, the security features of SQLite help ensure the safety of user data.



dât

4.2.2.2 *Database Detail Design*

Table 1. Example of table design

- **Transaction**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	type			VARCHAR(45)	Yes	Type of media
3	category			VARCHAR(45)	Yes	Category of media
4	price			INTEGER	Yes	Price of media
5	quantity			INTEGER	Yes	Quantity of media
6	title			VARCHAR(45)	Yes	Title of media
7	value			INTEGER	Yes	Value of media
8	imageUrl			VARCHAR(45)	Yes	URL of media image

- **CD**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	artist			VARCHAR(45)	Yes	Artist of the CD
3	recordLabel			VARCHAR(45)	Yes	Record label of the CD
4	musicType			VARCHAR(45)	Yes	Type of music
5	releasedDate			DATE		Release date of the CD

- **Book**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	author			VARCHAR(45)	Yes	Author of the book
3	coverType			VARCHAR(45)	Yes	Cover type of the book
4	publisher			VARCHAR(45)	Yes	Publisher of the book
5	publishDate			DATETIME	Yes	Publishing date
6	numOfPages			INTEGER	Yes	Number of pages
7	language			VARCHAR(45)	Yes	Language of the book
8	bookCategory			VARCHAR(45)	Yes	Category of the book

- **User**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	name			VARCHAR(45)	Yes	User's name
3	email			VARCHAR(45)	Yes	User's email
4	address			VARCHAR(45)	Yes	User's address
5	phone			VARCHAR(45)	Yes	User's phone number
6	PASSWORD			VARCHAR(45)	Yes	User's password

- **DVD**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	discType			VARCHAR(45)	Yes	Type of DVD disc
3	director			VARCHAR(45)	Yes	Director of the DVD
4	runtime			INTEGER	Yes	Runtime in minutes
5	studio			VARCHAR(45)	Yes	Studio of the DVD
6	subtitle			VARCHAR(45)	Yes	Subtitle language
7	releasedDate			DATETIME		Release date of the DVD
8	filmType			VARCHAR(45)	Yes	Type of film

- **Order**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	email			VARCHAR(45)	Yes	User's email
3	address			VARCHAR(45)	Yes	User's address
4	phone			VARCHAR(45)	Yes	User's phone number
5	userID		√	INTEGER	Yes	ID of the user
6	shipping_fee			INTEGER	Yes	Shipping fee
#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment

- **OrderMedia**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	mediaID		√	INTEGER	Yes	ID of the media
2	orderID		√	INTEGER	Yes	ID of the order
3	price			INTEGER	Yes	Price of the media
4	quantity			INTEGER	Yes	Quantity of the media

- **Transaction**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	orderID		√	INTEGER	Yes	ID of the order
3	createAt			DATETIME	Yes	Creation timestamp
4	content			VARCHAR(45)	Yes	Content of transaction

- **Card**

#	Column Name	PK	FK	Data Type	Mandatory	Description
1	id	√		INTEGER	Yes	ID, auto increment
2	cardNumber			VARCHAR(45)	Yes	Card number
3	holderName			VARCHAR(45)	Yes	Name of card holder
4	expirationDate			DATE	Yes	Expiration date
5	securityCode			VARCHAR(45)	Yes	Security code
6	userID		√	INTEGER	Yes	ID of the user

-

- ***4.2.2.3 Database Detailed Script***

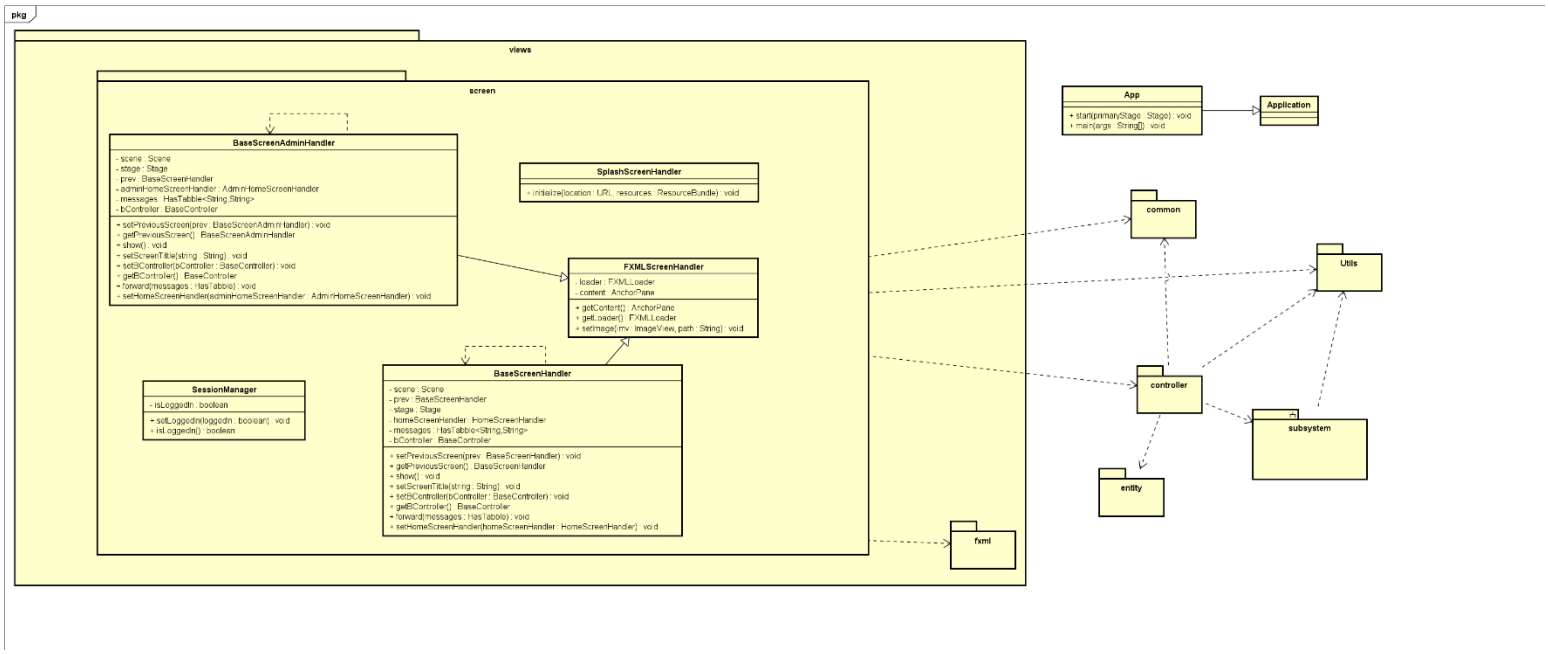
For SQL statements, access the corresponding AIMS.sql file in the source code.

4.3 Non-Database Management System Files

NONE

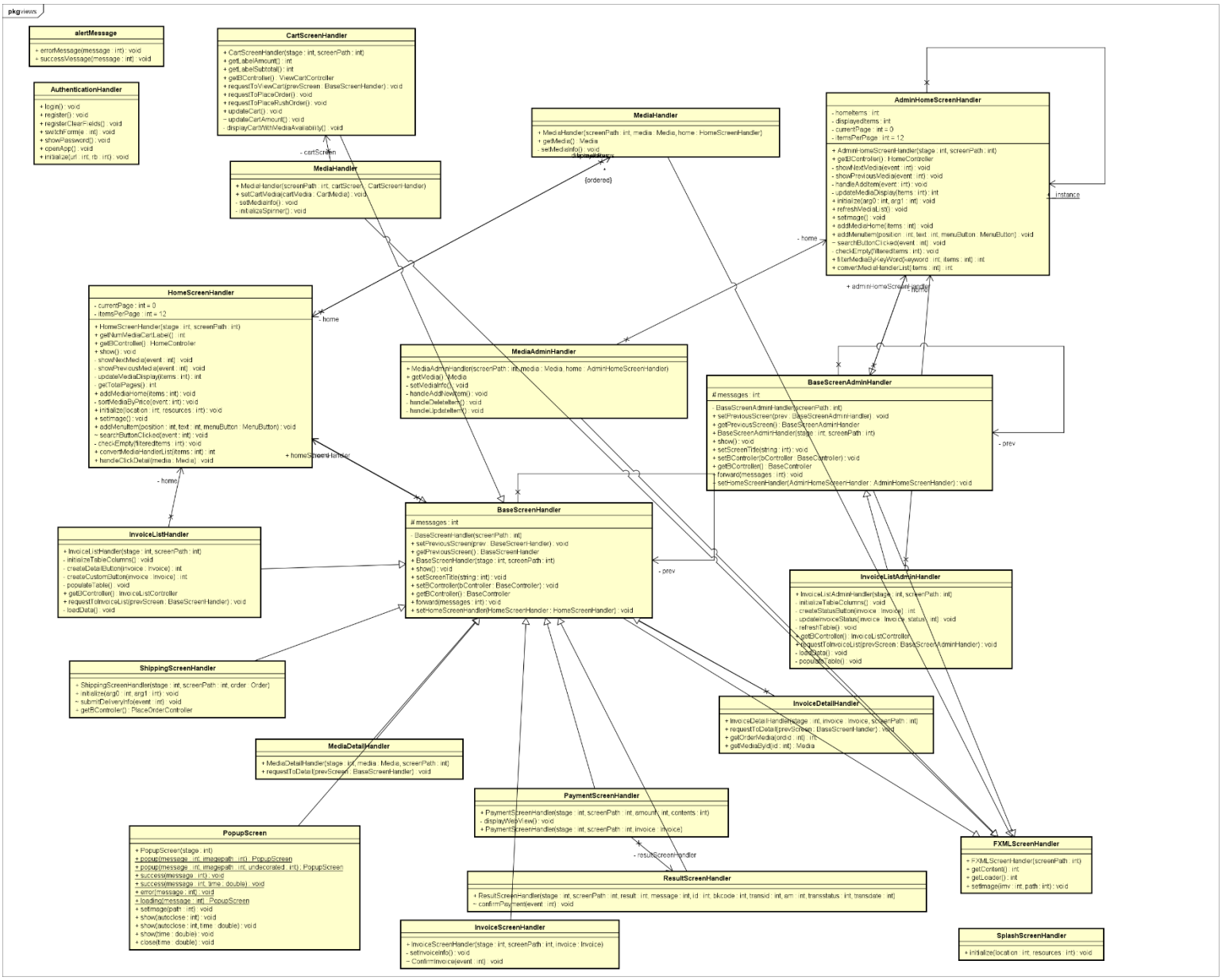
4.4 Class Design

4.4.1 General Class Diagram

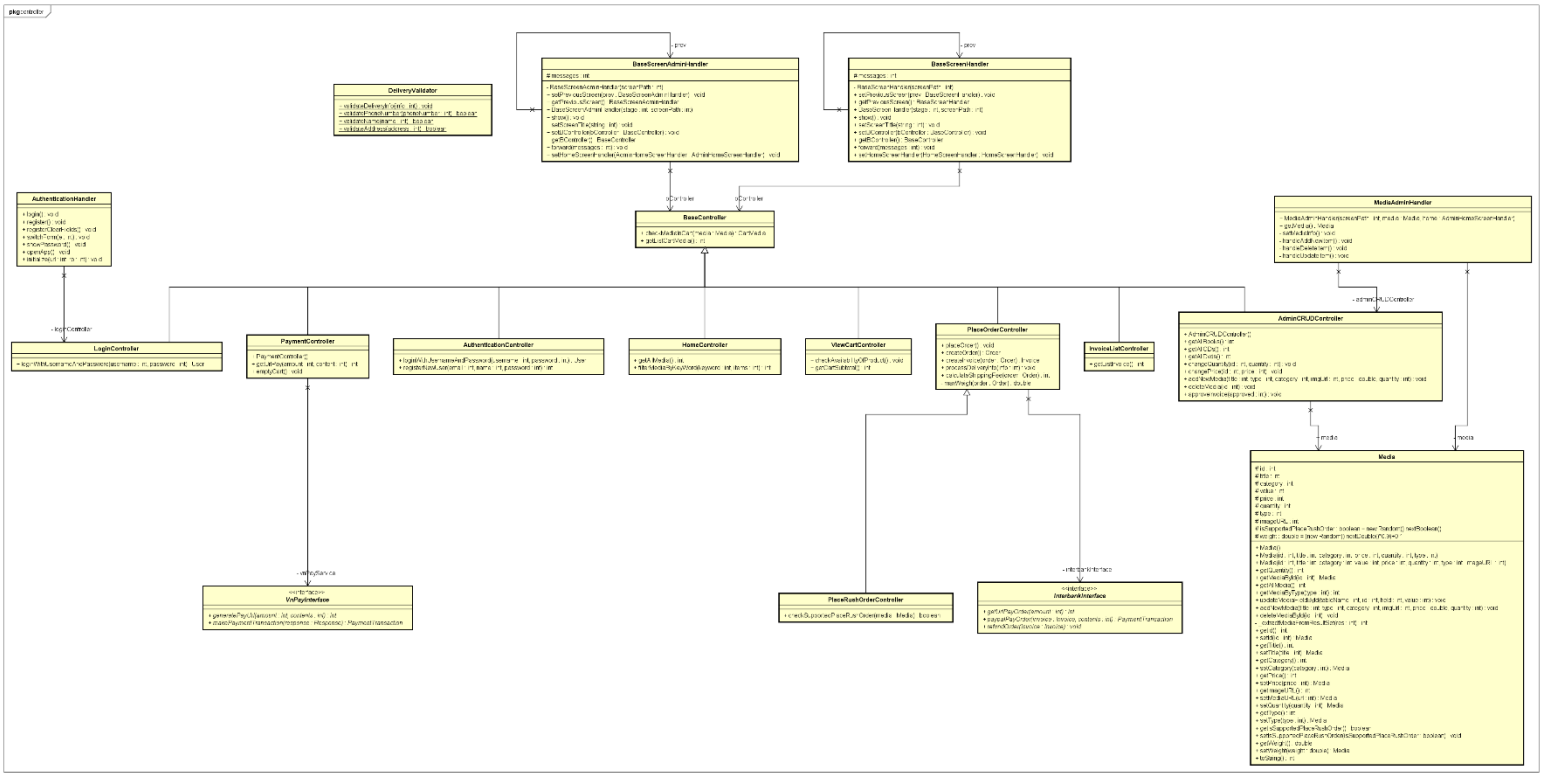


4.4.2 **Class Diagrams**

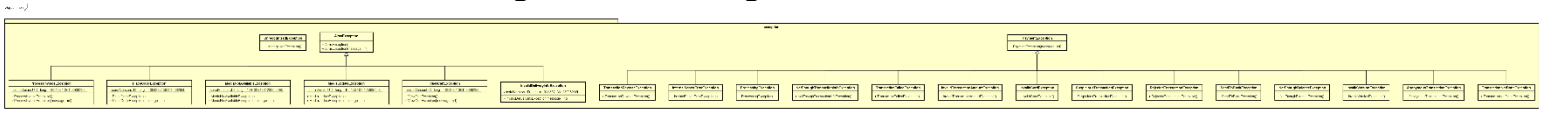
4.4.2.1 ***Class Diagram for Package Views***



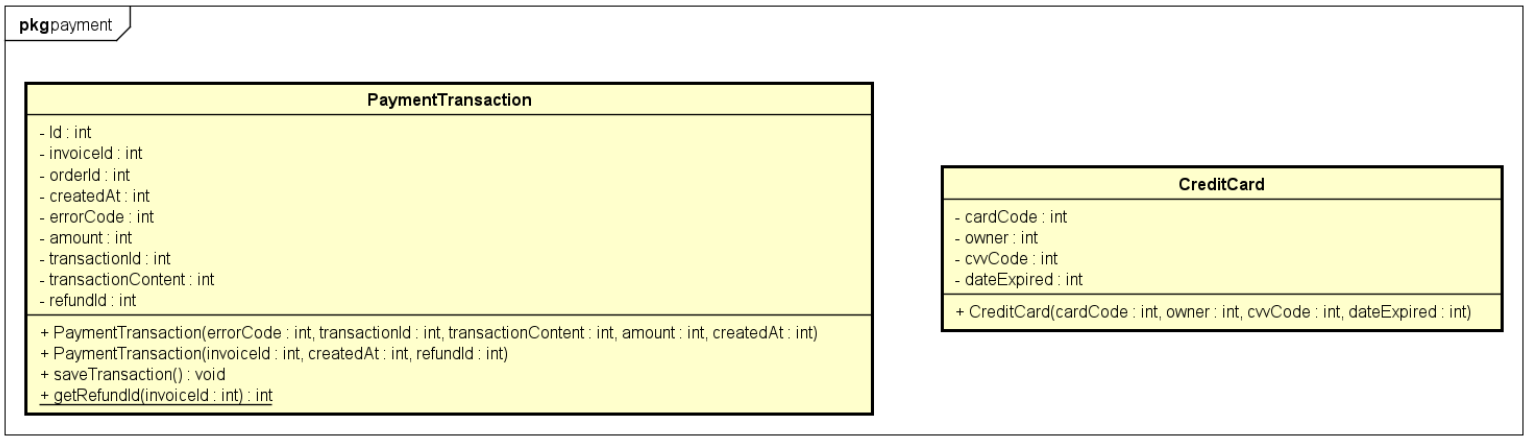
4.4.2.2 *Class Diagram for Package Controller*



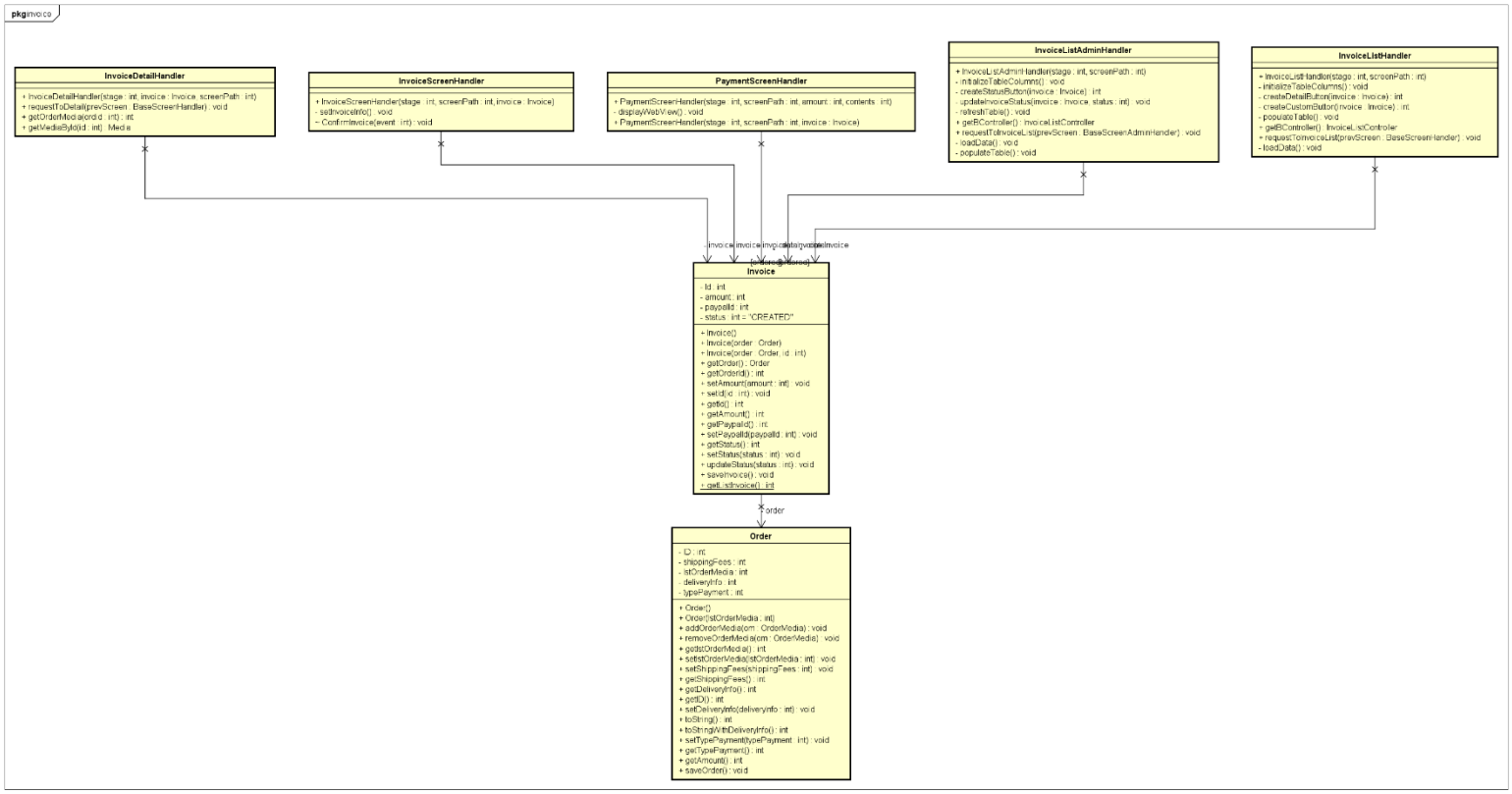
4.4.2.3 *Class Diagram for Package Common*



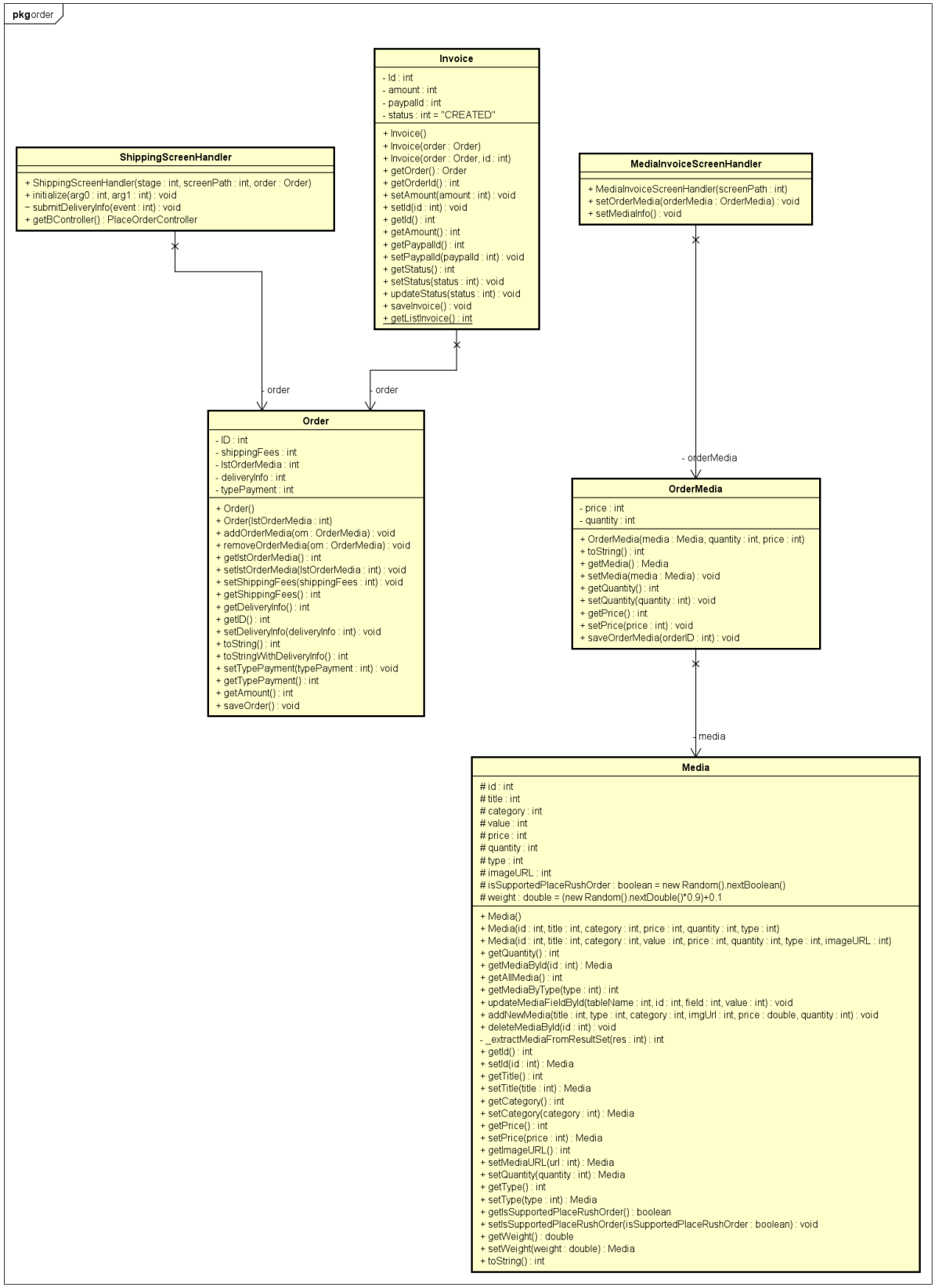
4.4.2.4 *Class Diagram for Package Entity-Payment*



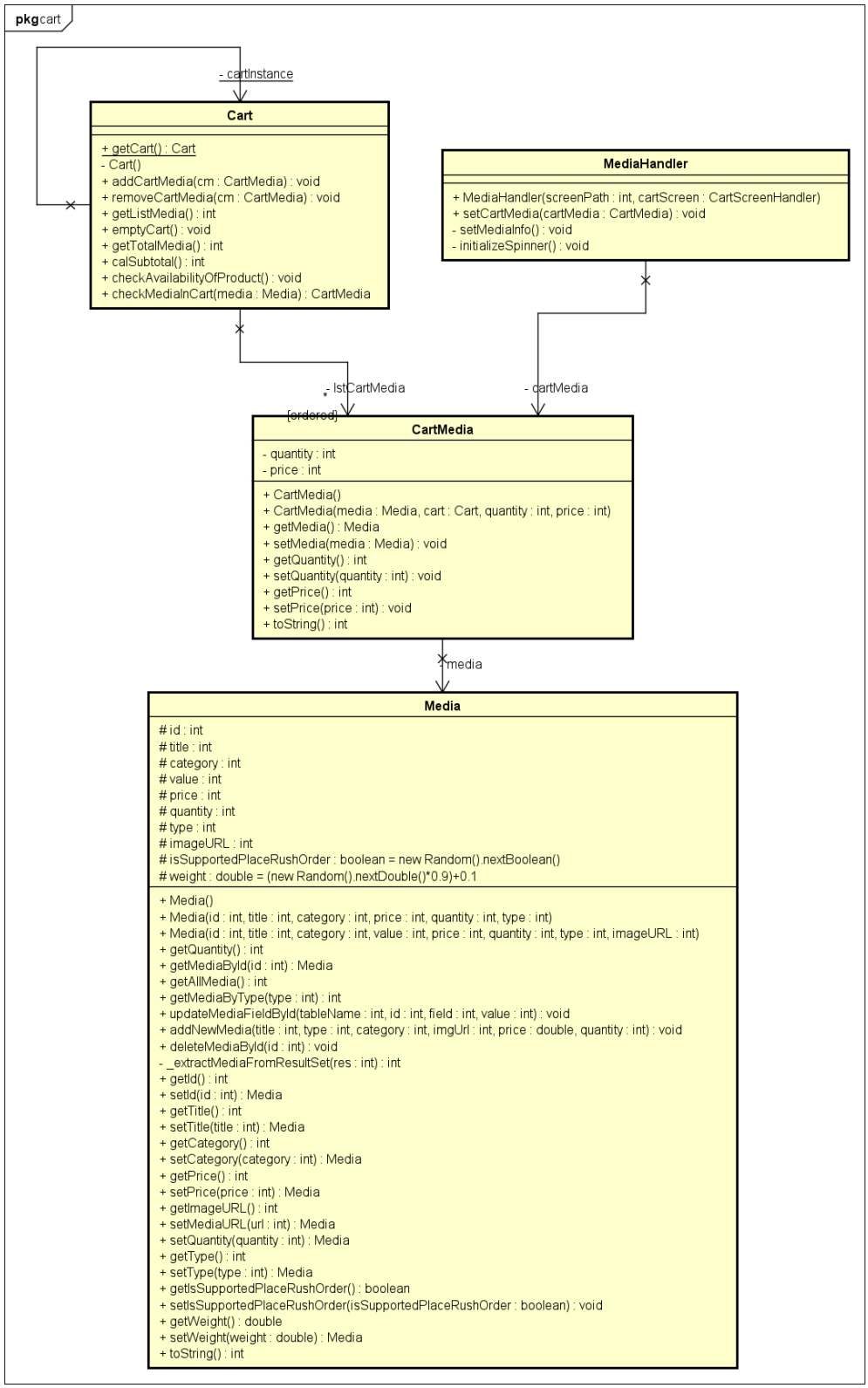
4.4.2.5 *Class Diagram for Package Entity - Invoice*



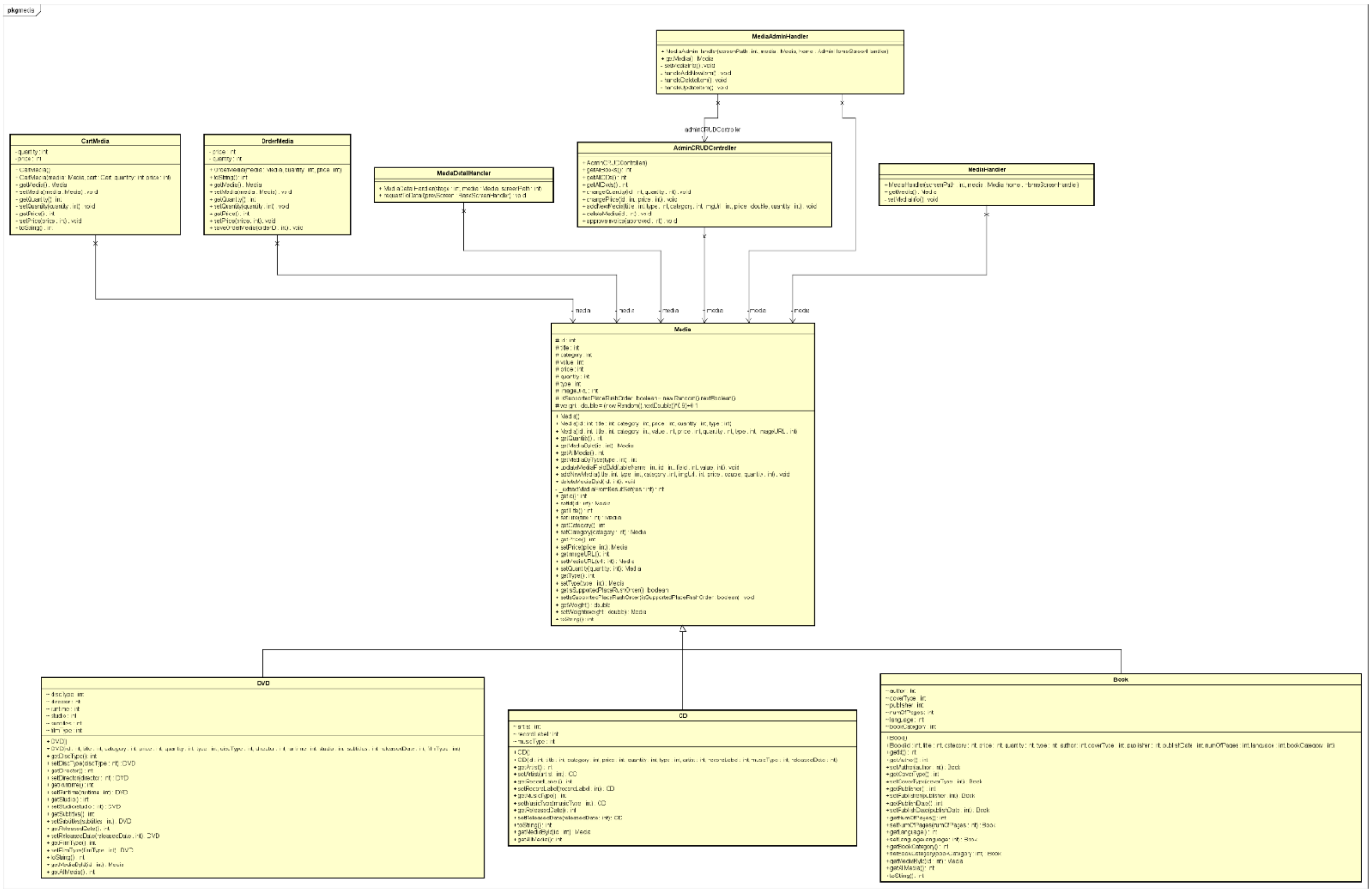
4.4.2.6 *Class Diagram for Package Entity – Order*



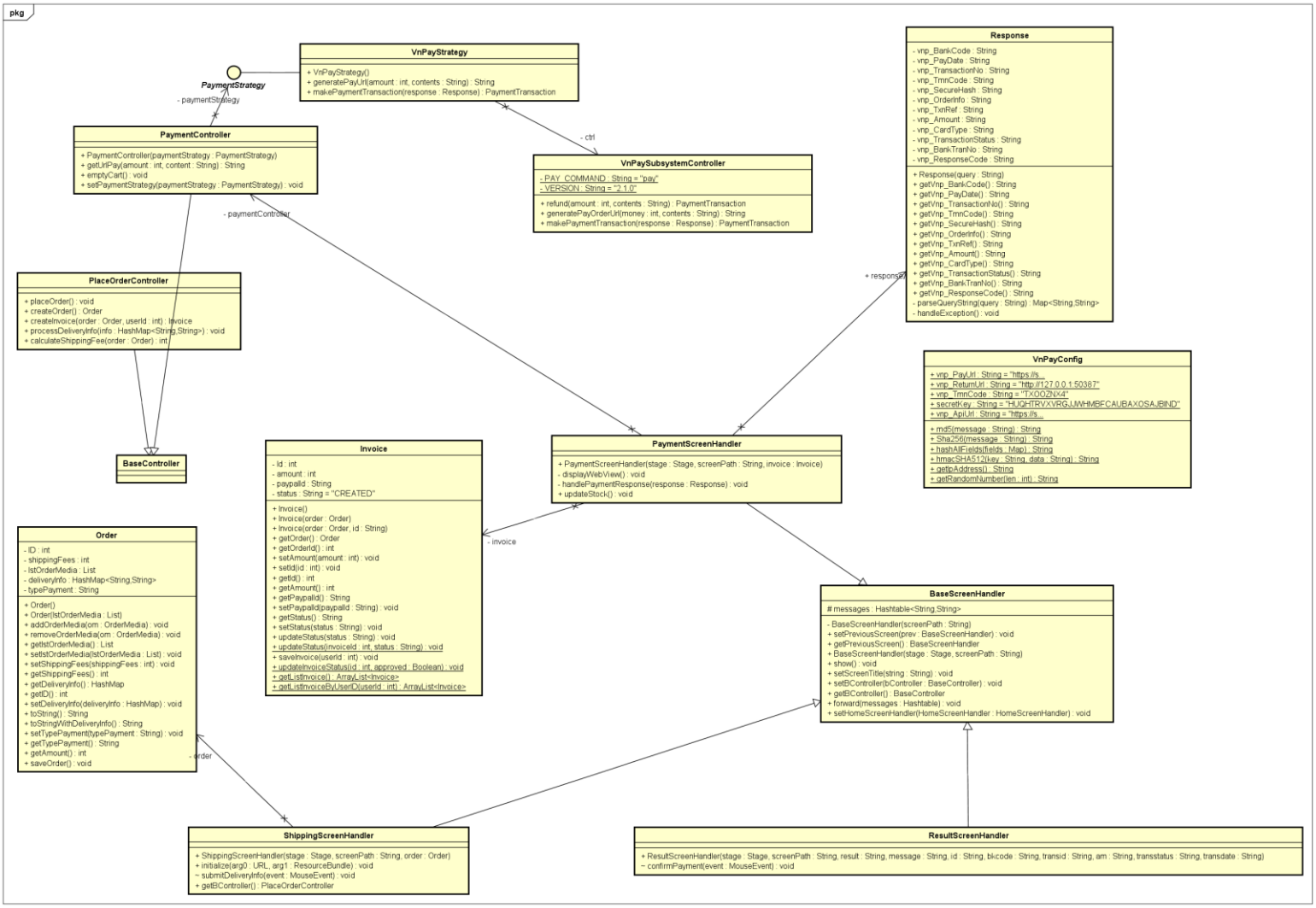
4.4.2.7 *Class Diagram for Package Entity – Cart*



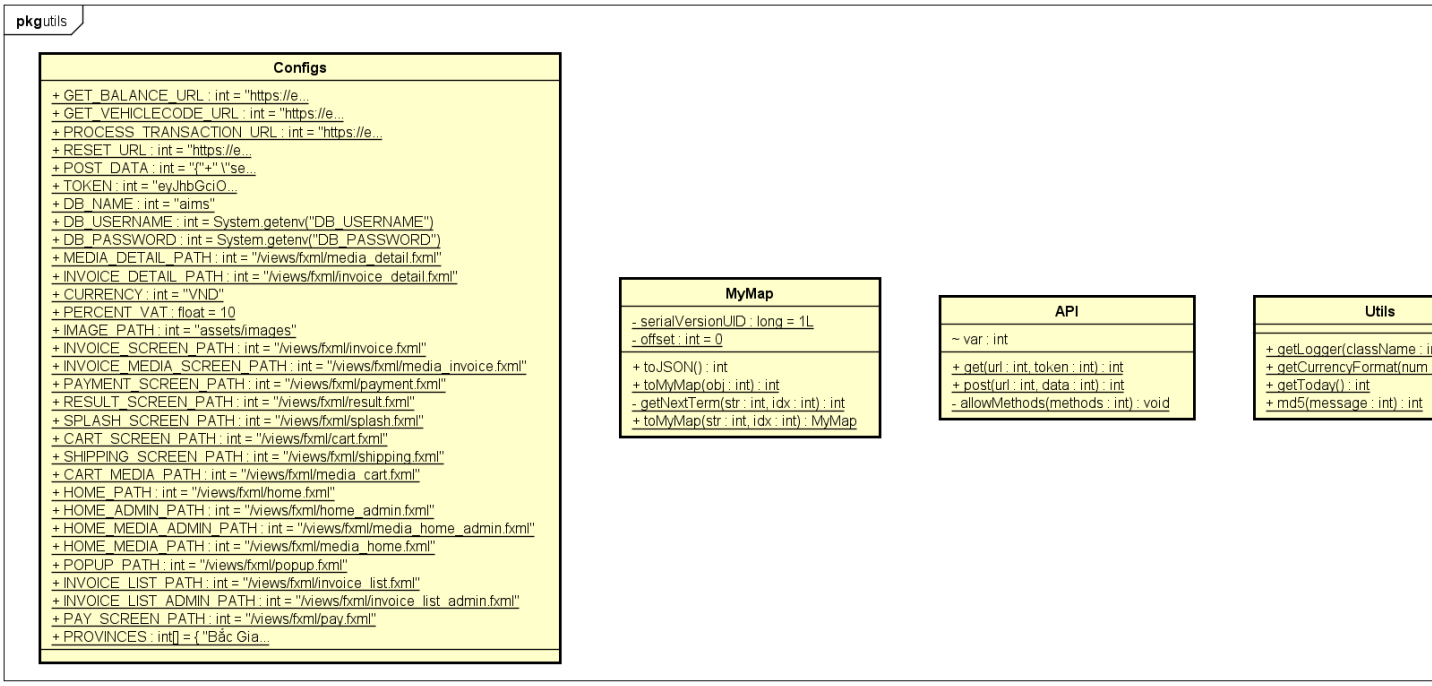
4.4.2.8 *Class Diagram for Package Entity - Media*



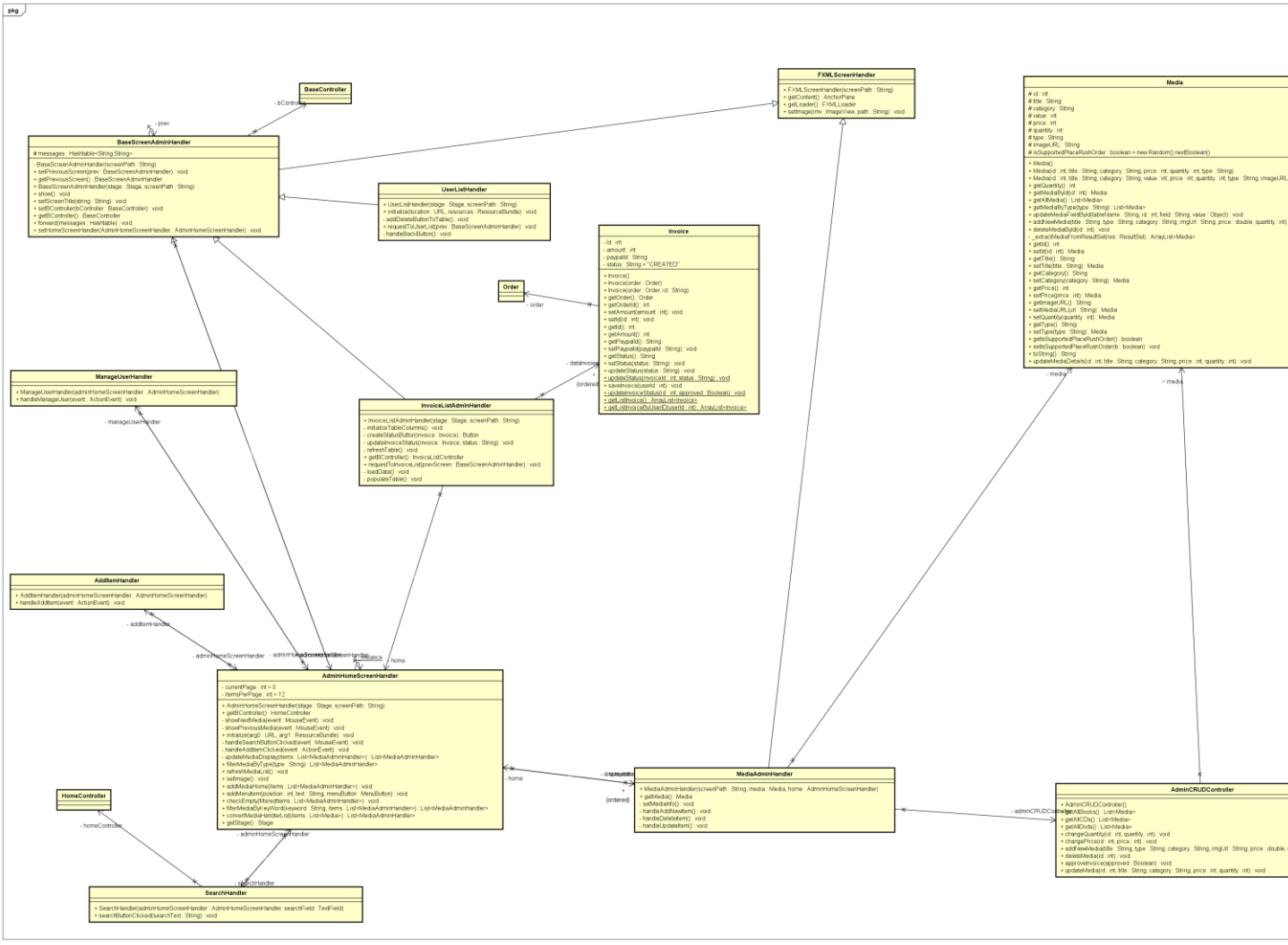
4.4.2.9 *Class Diagram for Package Subsystem*



4.4.2.10 *Class Diagram for Package Utlis*



4.4.2.11 *Class Diagram for package views.screen.home_admin*


pkg
BaseController
- bController
- prev
BaseScreenAdminHandler
messages : Hashtable<String,String>
- BaseScreenAdminHandler(screenPath : String)
+ setPreviousScreen(prev : BaseScreenAdminHandler) : void
+ getPreviousScreen() : BaseScreenAdminHandler
+ BaseScreenAdminHandler(stage : Stage, screenPath : String)
+ show() : void
+ setScreenTitle(string : String) : void
+ setBiController(biController : BaseController) : void
+ getBiController() : BaseController
+ forward(messages : Hashtable) : void
+ setHomeScreenHandler(AdminHomeScreenHandler : AdminHomeScreenHandler) : void
FXMLScreenHandler
+ FXMLScreenHandler(screenPath : String)
+ getContent() : AnchorPane
+ getLoader() : FXMLLoader
+ setImage(imv : ImageView, path : String) : void
Media
id : int
title : String
category : String
value : int
price : int
quantity : int
type : String
imageURL : String
isSupportedPlaceRushOrder : boolean = new Random().nextBoolean()
+ Media()
+ Media(id : int, title : String, category : String, price : int, quantity : int, type : String)
+ Media(id : int, title : String, category : String, value : int, price : int, quantity : int, type : String, imageURL
+ getQuantity() : int
+ getMediaById(id : int) : Media
+ getAllMedia() : List<Media>
+ getMediaByType(type : String) : List<Media>
+ updateMediaFieldById(tableName : String, id : int, field : String, value : Object) : void
+ addNewMedia(title : String, type : String, category : String, imgUrl : String, price : double, quantity : int)
+ deleteMediaById(id : int) : void
- _extractMediaFromResultSet(res : ResultSet) : ArrayList<Media>
+ getId() : int
+ setId(id : int) : Media
+ getTitle() : String
+ setTitle(title : String) : Media
+ getCategory() : String
+ setCategory(category : String) : Media
+ getPrice() : int
+ setPrice(price : int) : Media
+ getImageURL() : String
+ setMediaURL(url : String) : Media
+ setQuantity(quantity : int) : Media
+ getType() : String
+ setType(type : String) : Media
+ getIsSupportedPlaceRushOrder() : boolean
+ setIsSupportedPlaceRushOrder(b : boolean) : void
+ toString() : String
+ updateMediaDetails(id : int, title : String, category : String, price : int, quantity : int) : void
- media
- media
UserListHandler
+ UserListHandler(stage : Stage, screenPath : String)
+ initialize(location : URL, resources : ResourceBundle) : void
- addDeleteButtonToTable() : void
+ requestToUserList(prev : BaseScreenAdminHandler) : void
- handleBackButton() : void
Invoice
- id : int
- amount : int
- paypalId : String
- status : String = "CREATED"
+ Invoice()
+ Invoice(order : Order)
+ Invoice(order : Order, id : String)
+ getOrder() : Order
+ getOrderId() : int
+ setAmount(amount : int) : void
+ setId(id : int) : void
+ getId() : int
+ getAmount() : int
+ getPaypalId() : String
+ setPaypalId(paypalId : String) : void
+ getStatus() : String
+ setStatus(status : String) : void
+ updateStatus(status : String) : void
+ updateStatus(invoiceId : int, status : String) : void
+ saveInvoice(userId : int) : void
+ updateInvoiceStatus(id : int, approved : Boolean) : void
+ getListInvoice() : ArrayList<Invoice>
+ getListInvoiceByUserID(userId : int) : ArrayList<Invoice>
Order
- order
- dataInvoice
*
{ordered}
ManageUserHandler
+ ManageUserHandler(adminHomeScreenHandler : AdminHomeScreenHandler)
+ handleManageUser(event : ActionEvent) : void
- manageUserHandler
InvoiceListAdminHandler
+ InvoiceListAdminHandler(stage : Stage, screenPath : String)
- initializeTableColumns() : void
- createStatusButton(invoice : Invoice) : Button
- updateInvoiceStatus(invoice : Invoice, status : String) : void
- refreshTable() : void
+ getBiController() : InvoiceListController
+ requestToInvoiceList(prevScreen : BaseScreenAdminHandler) : void
- loadData() : void
- populateTable() : void
AddItemHandler
+ AddItemHandler(adminHomeScreenHandler : AdminHomeScreenHandler)
+ handleAddItem(event : ActionEvent) : void
- addItemHandler
- adminHomeScreenHandler
- adminHomeScreenHandler
- instance
- home
AdminHomeScreenHandler
- currentPage : int = 0
- itemsPerPage : int = 12
+ AdminHomeScreenHandler(stage : Stage, screenPath : String)
+ getBiController() : HomeController
- showNextMedia(event : MouseEvent) : void
- showPreviousMedia(event : MouseEvent) : void
+ initialize(arg0 : URL, arg1 : ResourceBundle) : void
- handleSearchButtonClicked(event : MouseEvent) : void
- handleAddItemClicked(event : ActionEvent) : void
- updateMediaDisplay(items : List<MediaAdminHandler>) : List<MediaAdminHandler>
+ filterMediaByType(type : String) : List<MediaAdminHandler>
+ refreshMediaList() : void
+ setImage() : void
+ addMediaHome(items : List<MediaAdminHandler>) : void
+ addMenuItem(position : int, text : String, menuButton : MenuButton) : void
+ checkEmpty(filteredItems : List<MediaAdminHandler>) : void
+ filterMediaByKeyWord(keyword : String, items : List<MediaAdminHandler>) : List<MediaAdminHandler>
+ convertMedia-handlerList(items : List<Media>) : List<MediaAdminHandler>
+ getStage() : Stage
- adminHomeScreenHandler
- home
- displayItem
*
{ordered}
MediaAdminHandler
+ MediaAdminHandler(screenPath : String, media : Media, home : AdminHomeScreenHandler)
+ getMedia() : Media
- setMediaInfo() : void
- handleAddNewItem() : void
- handleDeleteItem() : void
- handleUpdateItem() : void
- adminCRUDController
AdminCRUDController
+ AdminCRUDController()
+ getAllBooks() : List<Media>
+ getAllCDs() : List<Media>
+ getAllDvds() : List<Media>
+ changeQuantity(id : int, quantity : int) : void
+ changePrice(id : int, price : int) : void
+ addNewMedia(title : String, type : String, category : String, imgUrl : String, price : double,
+ deleteMedia(id : int) : void
+ approveInvoice(approved : Boolean) : void
+ updateMedia(id : int, title : String, category : String, price : int, quantity : int) : void
HomeController
- homeController
SearchHandler
+ SearchHandler(adminHomeScreenHandler : AdminHomeScreenHandler, searchField : TextField)
+ searchButtonClicked(searchText : String) : void
- searchHandler

5 Design Considerations

5.1 Goals and Guidelines

Goals:

- Performance Optimization:
  - Objective: Enhance the application's speed and responsiveness.
  - Implementation: Utilize efficient database queries, caching strategies, and minimize data processing and network overhead to ensure quick and smooth operation.
- Scalability:
  - Objective: Develop a modern, easy-to-use interface with JavaFX.
  - Implementation: Employ a modular architecture and scalable technologies like MySQL and adaptable cloud infrastructure to support growth.
- Security and Privacy:
  - Objective: Protect user data and comply with regulations.
  - Implementation: Focus on strong authentication and authorization protocols, implement thorough logging, and adopt best practices for data security and privacy.
- User-Friendly Interface:
  - Objective: Provide an intuitive and consistent user experience.
  - Implementation: Design a clear and navigable interface that enhances ease of use and user satisfaction.

Guidelines:

- Coding Standards:
  - Objective: Maintain high code quality and consistency.
  - Implementation: Adhere to standardized naming conventions, consistent formatting, and thorough documentation to facilitate maintenance and reduce errors.
- Modular Design:
  - Objective: Simplify management, testing, and updates.
  - Implementation: Structure the application into self-contained modules, allowing for easier handling and modification without impacting the entire system.
- Error Handling:
  - Objective: Ensure the application can handle and recover from errors effectively.
  - Implementation: Implement robust error handling mechanisms to provide meaningful feedback to users and developers and ensure the system can recover gracefully from issues.

5.2 Architectural Strategies

- **Java and JavaFX for Desktop Application Development**:

    - Java and JavaFX have been selected as the core technologies. Java's platform independence is a significant advantage, ensuring that the application can operate seamlessly across various operating systems such as Windows, macOS, and Linux without requiring any modifications. This capability aligns with the goal of broad accessibility and flexibility for the end-users.
    - JavaFX complements Java by offering a robust set of tools and UI components designed for creating modern, visually appealing, and responsive user interfaces specifically tailored for desktop environments. JavaFX's rich set of features, including advanced graphics and media libraries, allows for the development of an engaging and user-friendly interface. This choice ensures that the AIMS application not only functions well but also provides an intuitive and pleasant user experience.
- **Use of SQLite as the Embedded Database**: SQLite has been selected as the embedded database management system for the AIMS desktop application. Its lightweight nature and efficient performance make it an ideal choice for desktop applications that have moderate data storage and processing needs. SQLite's self-contained and serverless architecture simplifies the integration process and minimizes the application's footprint, making it a perfect fit for a desktop environment.
- **Design of Desktop User Interface**:
    - The user interface (UI) design for the AIMS desktop application is centered on enhancing usability and functionality. The design prioritizes clear navigation and the use of intuitive UI components, such as buttons, menus, and toolbars, which facilitate an efficient and user-friendly experience. Each aspect of the UI is structured to support the primary functionalities of the application, including product management, order processing, and user administration.
    - The UI is designed with a focus on streamlining user interactions and minimizing the learning curve. Consistent design patterns and visual cues are employed to guide users through the application, ensuring that they can perform their tasks efficiently and with ease. The aim is to create a cohesive and visually appealing interface that aligns with the functional requirements of the AIMS application.
- Make Use of Github:
    - GitHub will be utilized as the version control system and collaborative platform for the development of the AIMS desktop application. GitHub provides a comprehensive set of tools that facilitate effective source code management, enabling the team to track changes, manage versions, and collaborate seamlessly.
    - Using GitHub, the development team can implement a structured workflow, incorporating practices such as branching and merging, to ensure that development progresses smoothly and efficiently. GitHub's features for issue tracking, project management, and continuous integration further enhance the team's ability to manage the development process, address issues promptly, and maintain a high standard of code quality.

5.3 Coupling and Cohesion

Coupling:

- The system prioritizes low coupling to enhance modularity and ease of maintenance. It achieves this through a modular architecture that separates the user interface, backend logic, and data persistence into distinct modules. Each module can operate independently, making the system easier to update and manage.

```
public class ResultScreenHandler extends BaseScreenHandler {
    //Data Coupling
    public ResultScreenHandler(Stage stage, String screenPath, String result, String message, String id, String bkcode, String transid, String am, String transstatus, String
        super(stage, screenPath);
        resultLabel.setText(result);
        messageLabel.setText(message);
        invoiceid.setText(id);
        bankcode.setText(bkcode);
        transactionid.setText(transid);
        amount.setText(am);
        transactionstatus.setText(transstatus);
        String date =transdate.substring(beginIndex:6,endIndex:8) +"/"+transdate.substring(beginIndex:4,endIndex:6)+"/"+transdate.substring(beg...0,4)+ " at " + transdate.subs
        transactiondate.setText(date);
    }
```

Cohesion:

- To ensure high cohesion, each component is dedicated to a specific task. This focus improves the readability, maintainability, and reusability of the code. The Single Responsibility Principle is rigorously applied: UI controllers are responsible solely for user interactions, while service classes manage business logic.

```
//funtional cohesion
public class ViewCartController extends BaseController{

    /**
     * This method checks the available products in Cart
     * @throws SQLException
     * Data Coupling
     */
    public void checkAvailabilityOfProduct() throws SQLException{
        Cart.getCart().checkAvailabilityOfProduct();
    }

    /**
     * This method calculates the cart subtotal
     * @return subtotal
     * Data Coupling
     */
    public int getCartSubtotal(){
        int subtotal = Cart.getCart().calSubtotal();
        return subtotal;
    }

}
```

5.4 Design Principles

Ensuring the SOLID principles in the current design:

- Single Responsibility Principle(SRP): A class should have only one reason to change, meaning it should have only one job or responsibility. In the current design, this can be ensured by dividing the application into packages like Controller, View, Entity, and Data Access Layer, so each class has a single responsibility and performs only one specific function.
- Open/Closed Principle(OCP): Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. This means classes should be designed to allow their behavior to be extended without altering their source code, ensuring that new functionality can be added without changing existing code.
- Liskov Substitution Principle (LSP): Objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program. Subclasses should override the parent class methods in a way that does not break functionality from a client's point of view.
- Interface Segregation Principle (ISP): Clients should not be forced to depend on interfaces they do not use. This means designing small, specific interfaces rather than large, general-purpose ones, ensuring that implementing classes only need to be concerned with the methods that are of interest to them.

- Dependency Inversion Principle (DIP): High-level modules should not depend on low-level modules. Both should depend on abstractions. This principle emphasizes that classes should depend on interfaces or abstract classes rather than concrete classes and functions, promoting loose coupling and enhancing testability and maintainability.

For example, this method retrieves all Media objects from the database and returns them, which creates the violation of SOLID:

-VIOLATION OF Single Responsibility Principle (SRP):

+ This class not only controls the logic flow but also directly performs database queries to retrieve Media data.

+ This makes the class responsible for more tasks than necessary.

-VIOLATION OF Dependency Inversion Principle (DIP):

+ The HomeController class directly creates a Media object inside the method to retrieve data from the database.

+ This creates a tight coupling between HomeController and the Media class, which does not adhere to the dependency inversion principle.

```java
public class HomeController extends BaseController{

    /** Phương thức này lấy tất cả các đối tượng Media từ cơ sở dữ liệu và trả về chúng.
     * VI PHẠM NGUYÊN TẮC SOLID:
     * - VI PHẠM Single Responsibility Principle (Nguyên tắc đơn trách nhiệm):
     *   + Lớp này không chỉ điều khiển luồng logic mà còn trực tiếp thực hiện truy vấn cơ sở dữ liệu để lấy dữ liệu Media.
     *   + Điều này làm cho lớp này có nhiều trách nhiệm hơn cần thiết.
     *
     * - VI PHẠM Dependency Inversion Principle (Nguyên tắc đảo ngược phụ thuộc):
     *   + Lớp HomeController tạo trực tiếp một đối tượng Media bên trong phương thức để lấy dữ liệu từ cơ sở dữ liệu.
     *   + Điều này tạo ra mối liên kết cứng giữa HomeController và lớp Media, không tuân theo nguyên tắc đảo ngược phụ thuộc.
     *
     /**
     * this method gets all Media in DB and return back to home to display
     * @return List[Media]
     * @throws SQLException
     */

    public List<Media> getAllMedia() throws SQLException{
        // data coupling
        return new Media().getAllMedia();
    }
    public List<Media> filterMediaByKeyWord(String keyword, List<Media> items) {
        List<Media> filteredItems = new ArrayList<>();
        for (Media item : items) {
            if (item.getTitle().toLowerCase().contains(keyword)) {
                filteredItems.add(item);
            }
        }
        return filteredItems;
    }

}
```

5.5 Design Patterns

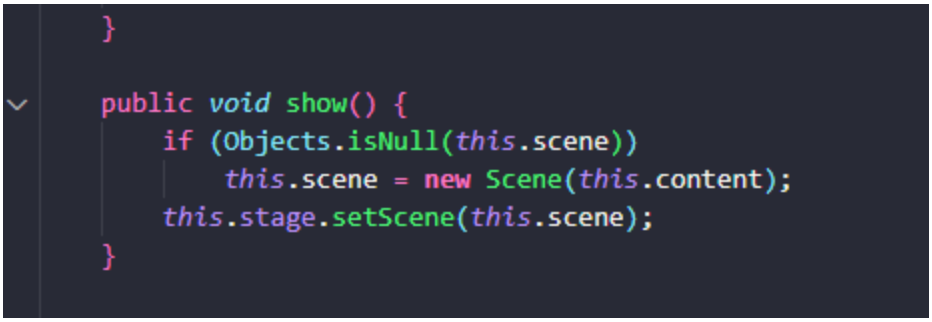
Strategy pattern:

- The Strategy pattern helps provide suggestions for a class to perform a task in multiple ways and extracts these algorithms into separate classes.

```
public class VnPaySubsystem implements VnPayInterface {
    private VnPaySubsystemController ctrl;

    public VnPaySubsystem() {
        this.ctrl = new VnPaySubsystemController();
    }

    public String generatePayUrl(int amount, String contents) {
        try {
            return ctrl.generatePayOrderUrl(amount, contents);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public PaymentTransaction makePaymentTransaction(Response response) throws ParseException {
            return ctrl.makePaymentTransaction(response);
    }
}
```

Singleton pattern:

- Singleton pattern ensures that a class has only one unique instance and provides a global way to access that instance.
- The pattern is used in the BaseScreenHandler class as the function below. The BaseScreenHandler class has an attribute stage to store the stage of the screen, and it is initially initialized with a null value. Additionally, this attribute is set as private to prevent it from being modified by other classes.



6 Work Assessment

	H. Đức	Diện	Đông	M.Đức	Đạt
Percentage	32.5	20	32.5	10	5