HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

Version 1.3

<Project Name>

Subject: <Name of subject>

<Group Number>

<List of participants>

*Hanoi,* *<month, year>*

Table of Contents

Table of Contents 1

1 Introduction 5

1.1 Objective 5

1.2 Scope 5

1.3 Glossary 5

1.4 References 5

2 Overall Description 7

2.1 General Overview 7

2.2 Assumptions/Constraints/Risks 7

2.2.1 Assumptions 7

2.2.2 Constraints 7

2.2.3 Risks 8

3 System Architecture and Architecture Design 9

3.1 Architectural Patterns 9

3.2 Interaction Diagrams 9

3.3 Analysis Class Diagrams 9

3.4 Unified Analysis Class Diagram 9

3.5 Security Software Architecture 9

4 Detailed Design 10

4.1 User Interface Design 10

4.1.1 Screen Configuration Standardization 10

4.1.2 Screen Transition Diagrams 10

4.1.3 Screen Specifications 10

4.2 Data Modeling 10

4.2.1 Conceptual Data Modeling 10

4.2.2 Database Design 10

4.3 Non-Database Management System Files 11

4.4 Class Design 11

4.4.1 General Class Diagram 11

4.4.2 Class Diagrams 11

4.4.3 Class Design 11

5 Design Considerations 13

5.1 Goals and Guidelines 13

5.2 Architectural Strategies 13

5.3 Coupling and Cohesion 14

5.4 Design Principles 14

5.5 Design Patterns 14

# 1. Introduction

## Objective

The purpose of this document is to present a detailed description of the designs of the AIMS Project, an Internet Media Store. This document is intended for the programming group in Team 14, to use the designs as guidelines to implement the project. Additionally, this document is for reviewing by Prof. Nguyen Thi Thu Trang and her assistants, as a part of the course. The document could also be used for designers who aim to upgrade or modify the design of the system

## Scope

The AIMS Project software’s aim, as its name suggests, is to provide an online media store service to customers. In this software, users will have the ability to search for products and see detailed information about them. Users can view, add, and manage media products such as books, CDs, LP records, and DVDs. Key functions of this system include managing product inventory, facilitating customer orders, and handling payments.

The software will support the execution of payment transactions through VNPay, which will handle validating, adding money, and deducting money processes.

This document gives a detailed description of the software architecture of the AIMS system. It specifies the structure and design of some of the modules discussed in the SRS. It also provides sequential and activity diagrams for some use cases. The class diagrams show how the programming team would implement the specific modules.

## Glossary

|  |  |
| --- | --- |
| Term | Explain |
| ITSS | Information Technology Systems and Services |
| RAM | Random Access Memory |
| IDE | Integrated Development Environment |
| JRE | Java Runtime Environment |

## References

# Overall Description

## General Overview

The AIMS Project is a simulator for an Internet media store. This is a desktop software that runs on the Java Runtime Environment. This software is designed using the MVC (Model-View-Controller) architectural design model.

## Assumptions/Constraints/Risks

### Assumptions

The design described in this document is based on simulation requirements instead of reality requirements. Therefore, functions related to authentication and personal customer information are neglected.

This software is designed to run on any operating system with Java Runtime Environment.

### Constraints

The system is built accessible only through personal computer. The system is implemented using Java and uses JavaFX for making user interfaces.

For storing data, SQLite will be used.

### Risks

Data security is critically important as data breaches have the potential to compromise sensitive user information. These breaches pose significant risks to individuals and organizations alike, potentially leading to unauthorized access, identity theft, financial fraud, and reputational damage. Implementing robust data security measures, such as encryption, access controls, regular security audits, and employee training, is essential to mitigate these risks and safeguard confidential data. Proactive measures are crucial in today's interconnected digital landscape to ensure the integrity, confidentiality, and availability of sensitive information

# System Architecture and Architecture Design

## Architectural Patterns

Model-View-Controller (MVC) is the chosen design pattern. The concept of MVC which used in this project can be described by the image below



MVC has its own advantages:

Simultaneous development - Multiple developers can work simultaneously on the model, controller and views.

[High cohesion](https://en.wikipedia.org/wiki/Cohesion_(computer_science)) - MVC enables logical grouping of related actions on a controller together.

[Low coupling](https://en.wikipedia.org/wiki/Loose_coupling) - The very nature of the MVC framework is such that there is low coupling among models, views or controllers.

Ease of modification - Because of the separation, future development or modification is easier, and also the scalability.

## Interaction Diagrams

### General use case

A diagram of a product

Description automatically generated

### Place order

A diagram of a project

Description automatically generated

### Pay order

A diagram of a computer program

Description automatically generated with medium confidence

### VNPay Pay order

A diagram of a flowchart

Description automatically generated

### Search media

A diagram of a diagram

Description automatically generated

### Get all media

A diagram of a computer program

Description automatically generated

### Create media

A diagram of a diagram

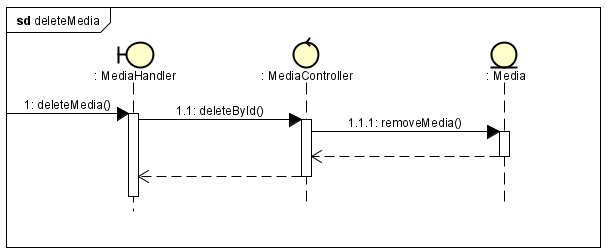
Description automatically generated

### Update media

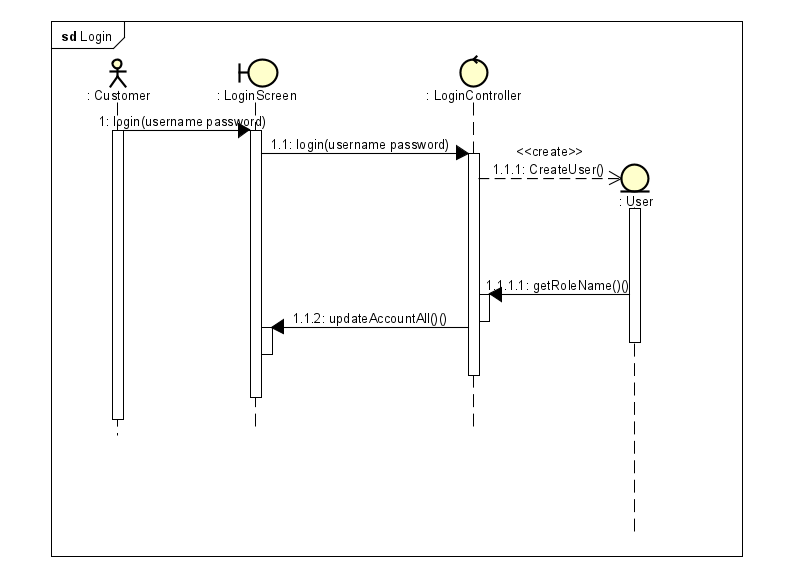
A diagram of a computer program

Description automatically generated

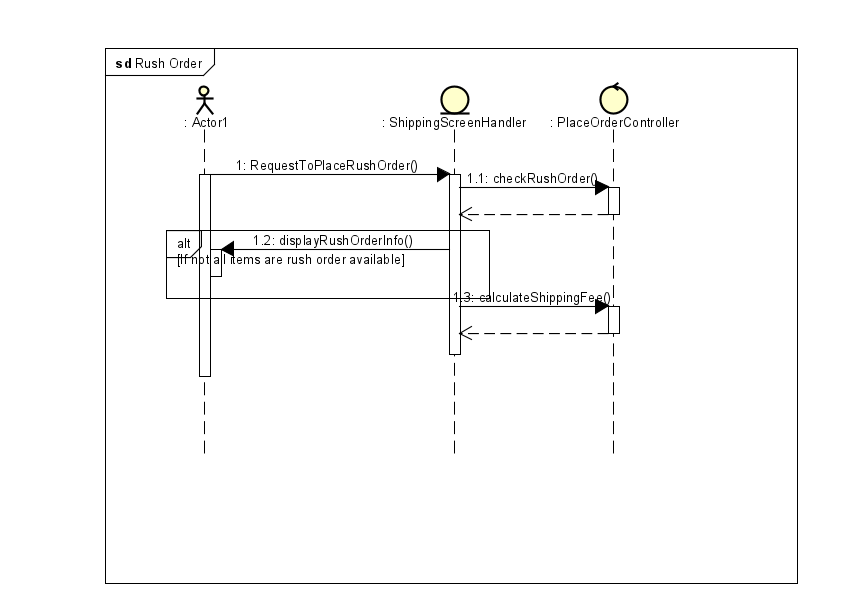
### Delete media



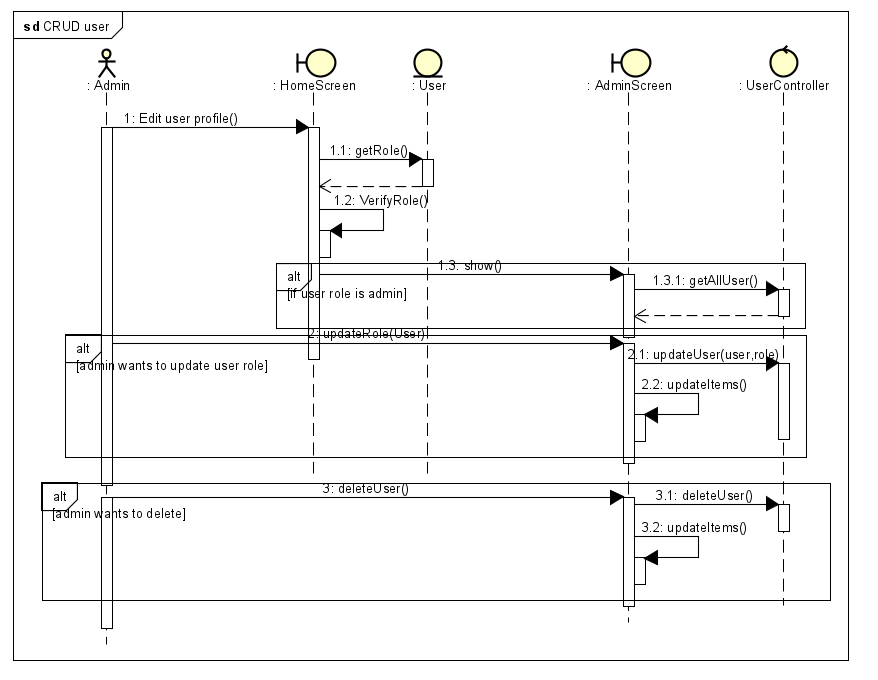
### Admin/ Manager login



### Rush Order



### Crud user



## Analysis Class Diagrams

## Unified Analysis Class Diagram

## Security Software Architecture

*<Describe the software components and configuration supporting the security and privacy of the system. Specify the architecture for (1) authentication to validate user identity before allowing access to the system;(2) authorization of users to perform functional activity once logged into the system, (3) encryption protocol to support the business risks and the nature of information, and (4) logging and auditing design, if required.>*

# Detailed Design

## User Interface Design

*<Suppose that you design a Graphical User Interface (GUI)>*

### Screen Configuration Standardization

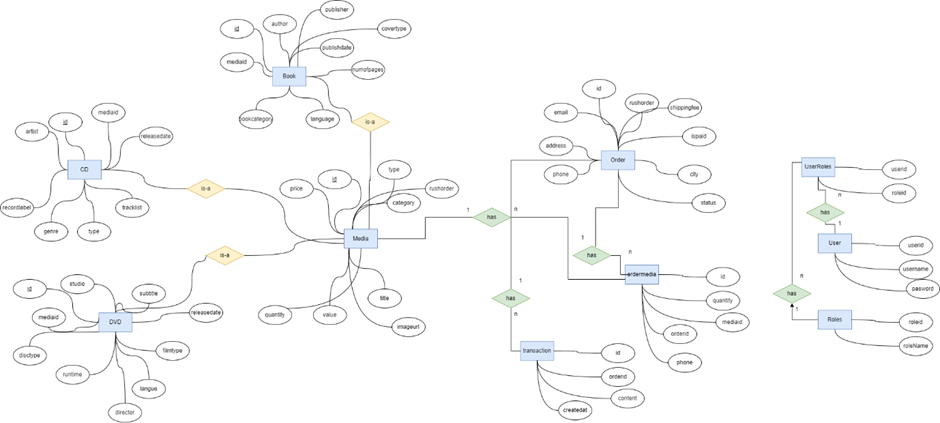
### Screen Transition Diagrams

### Screen Specifications

*<Screen images should be included in the screen specifications>*

## Data Modeling

### Conceptual Data Modeling



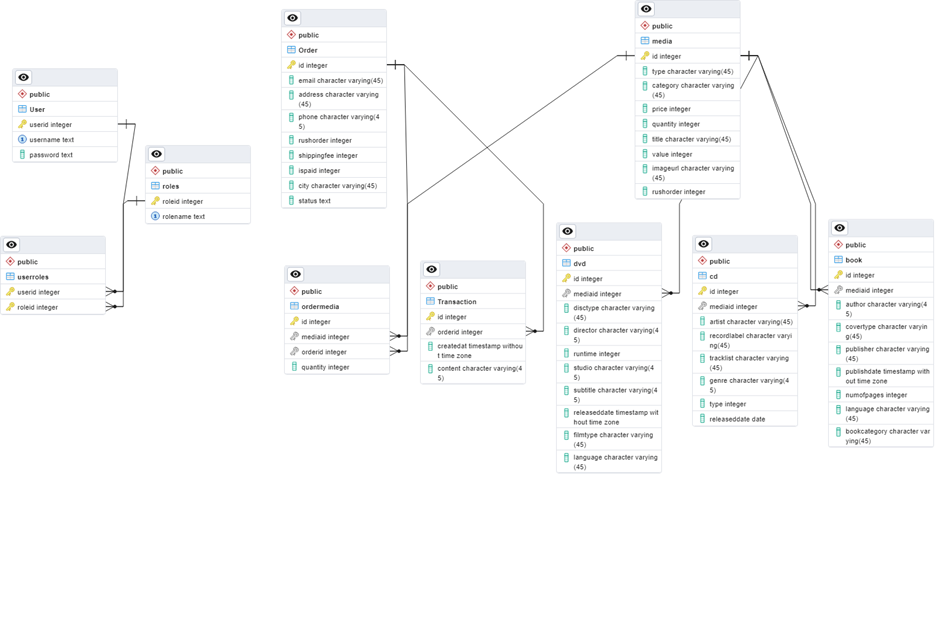
### Database Design

#### Database Management System

Database Management System: SQLite

SQLite is a lightweight, serverless, self-contained, and open-source relational database management system (RDBMS).

#### Database Diagram



#### Database Diagram

#### Media

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | Int | Yes | integer |
| 2 |  |  | type | String | Yes | Media type, e.g., Book, CD, DVD … |
| 3 |  |  | category | String | Yes | Media category as story, adventure, pop,.. |
| 4 |  |  | price | Int | Yes | The price of media |
| 5 |  |  | quantity | Int | Yes | The quantity of media |
| 6 |  |  | imageurl | String | Yes | The Image of media |
| 7 |  |  | rushorder | Int | Yes | Can rush order ? |

*Book*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | Int | Yes | integer |
| 2 |  | X | mediaid | Int | Yes | ID, same as ID of Media of which type is Book |
| 3 |  |  | author | String | No | Author |
| 4 |  |  | covertype | String | No | The type of book |
| 5 |  |  | publisher | String | No | Publisher |
| 6 |  |  | numofpages | Int | No | The number of pages of book |
| 7 |  |  | publishdate | Time | No | The day book publish |
| 8 |  |  | language | String | No | Language |
| 9 |  |  | bookcategory | String | No | Category of book |

*CD*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | Int | Yes | integer |
| 2 |  | X | mediaid | Int | Yes | ID, same as ID of Media of which type is CD |
| 3 |  |  | artist | String | No | Artist |
| 4 |  |  | recordlabel | String | No | The label of record |
| 5 |  |  | tracklist | String | No | Tracklist |
| 6 |  |  | genre | String | No | Genre |
| 7 |  |  | type | String | No | Type |
| 8 |  |  | releasedate | String | No | The day CD released |

*DVD*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | Int | Yes | integer |
| 2 |  | X | mediaid | Int | Yes | ID, same as ID of Media of which type is DVD |
| 3 |  |  | disctype | String | No | Disc type as Blu-ray, HD-DVD |
| 4 |  |  | director | String | No | Director |
| 5 |  |  | runtime | Int | No | Runtime |
| 6 |  |  | studio | String | No | Studio |
| 7 |  |  | subtitle | String | No | Subtitles |
| 8 |  |  | releaseddate | String | No | Release date |
| 9 |  |  | filmtype | String | No | Film type |
| 10 |  |  | language | String | No | Languge |

*Transction*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | Int | Yes | integer |
| 2 |  | X | orderid | Int | Yes | Order Id |
| 3 |  |  | createat | Time | Yes | The time order |
| 4 |  |  | content | String | No | Content |

*Ordermedia*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | Int | Yes | UUID |
| 2 |  | X | mediaid | Int | Yes | Media ID |
| 3 |  | X | orderid | Int | Yes | Order ID |
| 4 |  |  | quantity | Int | Yes | The number of media in order |

*Order*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | UUID | Yes | UUID |
| 2 |  |  | email | String | Yes | Email |
| 3 |  |  | address | String | Yes | Delivery address |
| 4 |  |  | phone | String | Yes | Delivery Phone |
| 5 |  |  | rushorder | INT | Yes | Can rush order? |
| 6 |  |  | shippingfee | INT | Yes | Shipping fees |
| 7 |  |  | ispaid | INT | Yes | Pay or not? |
| 8 |  |  | city | String | Yes | City |
| 9 |  |  | status | INT | Yes | Rejected or Apporved |

*User*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | UUID | Yes | UUID |
| 2 |  |  | username | String | Yes | Username |
| 3 |  |  | password | String | No | Password |

*UserRoles*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X |  | id | UUID | Yes | UUID |
| 2 |  | X | userid | Int | Yes | User ID |
| 3 |  | X | roleid | Int | Yes | Role ID |

*Roles*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | PK | FK | Column Name | Data Type | Mandatory | Description |
| 1 | X | X | roleid | Int | Yes | Role id |
| 2 |  |  | roleName | String | Yes | Role Name as user, admin or manager |

## Non-Database Management System Files

*<Provide the detailed description of all non-DBMS files if any and include a narrative description of the usage of each file that identifies if the file is used for input, output, or both, and if the file is a temporary file. Also provide an indication of which modules read and write the file and include file structures (refer to the data dictionary). As appropriate, the file structure information should include the following:*

*• Record structures, record keys or indexes, and data elements referenced within the records*

*• Record length (fixed or maximum variable length) and blocking factors*

*• Access method (e.g., index sequential, virtual sequential, random access, etc.)*

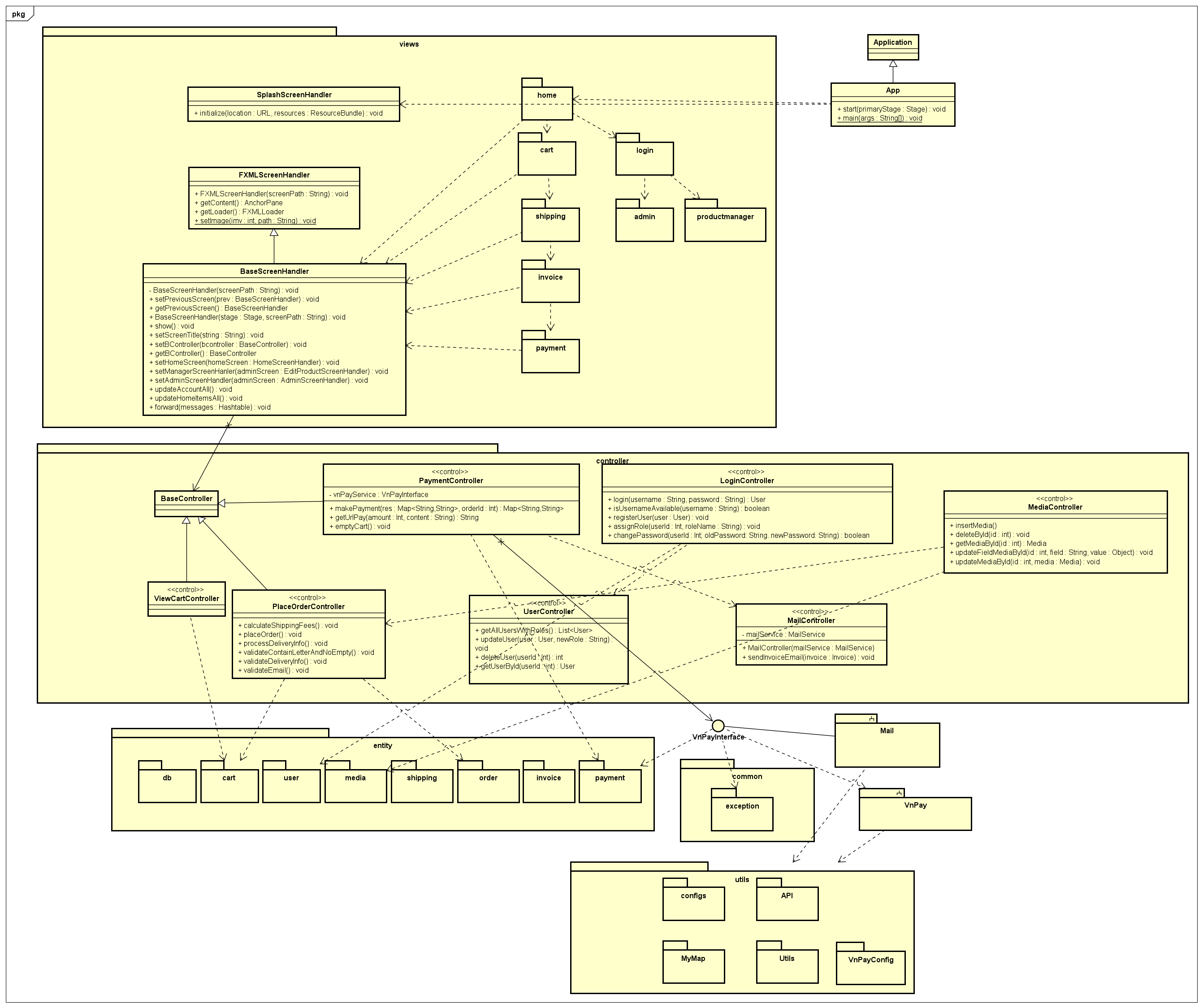
*• Estimate of the file size or volume of data within the file, including overhead resulting from file access methods*

*• Definition of the update frequency of the file (If the file is part of an online transaction-based system, provide the estimated number of transactions per unit of time, and the statistical mean, mode, and distribution of those transactions.)*

*• Backup and recovery specifications>*

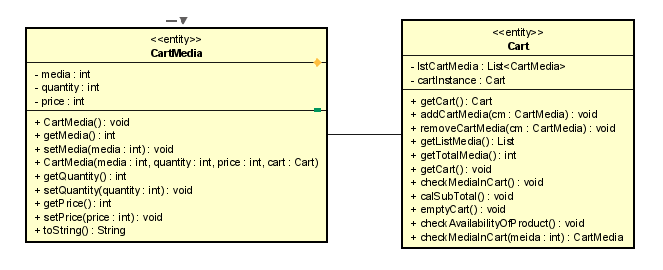
## Class Design

### General Class Diagram

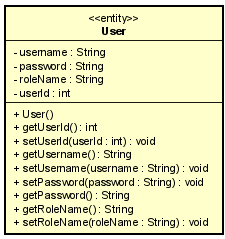


### Class Diagrams

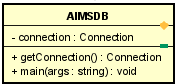
#### Class Diagram for Package Cart



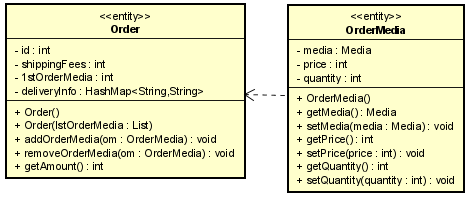
#### Class Diagram for Package User



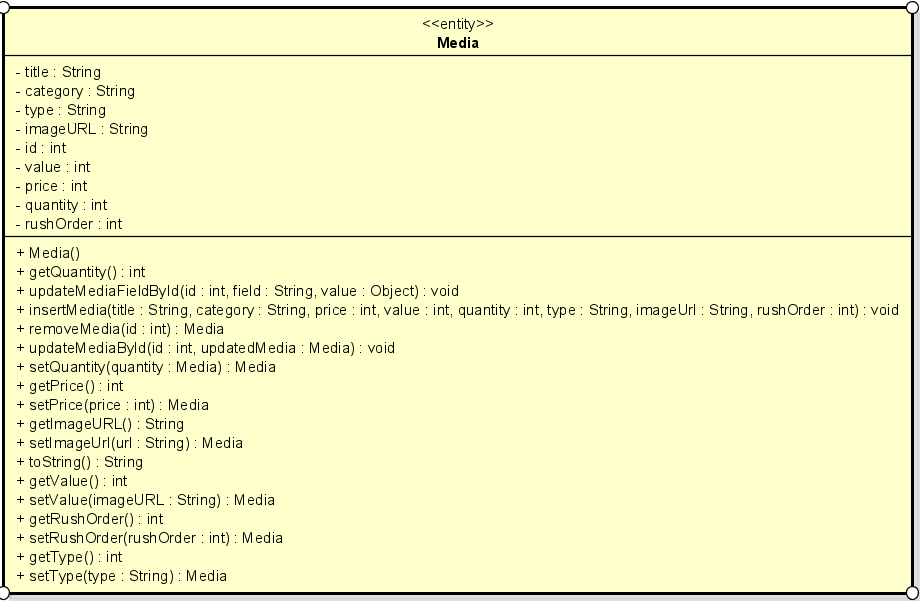
#### Class Diagram for Package DB



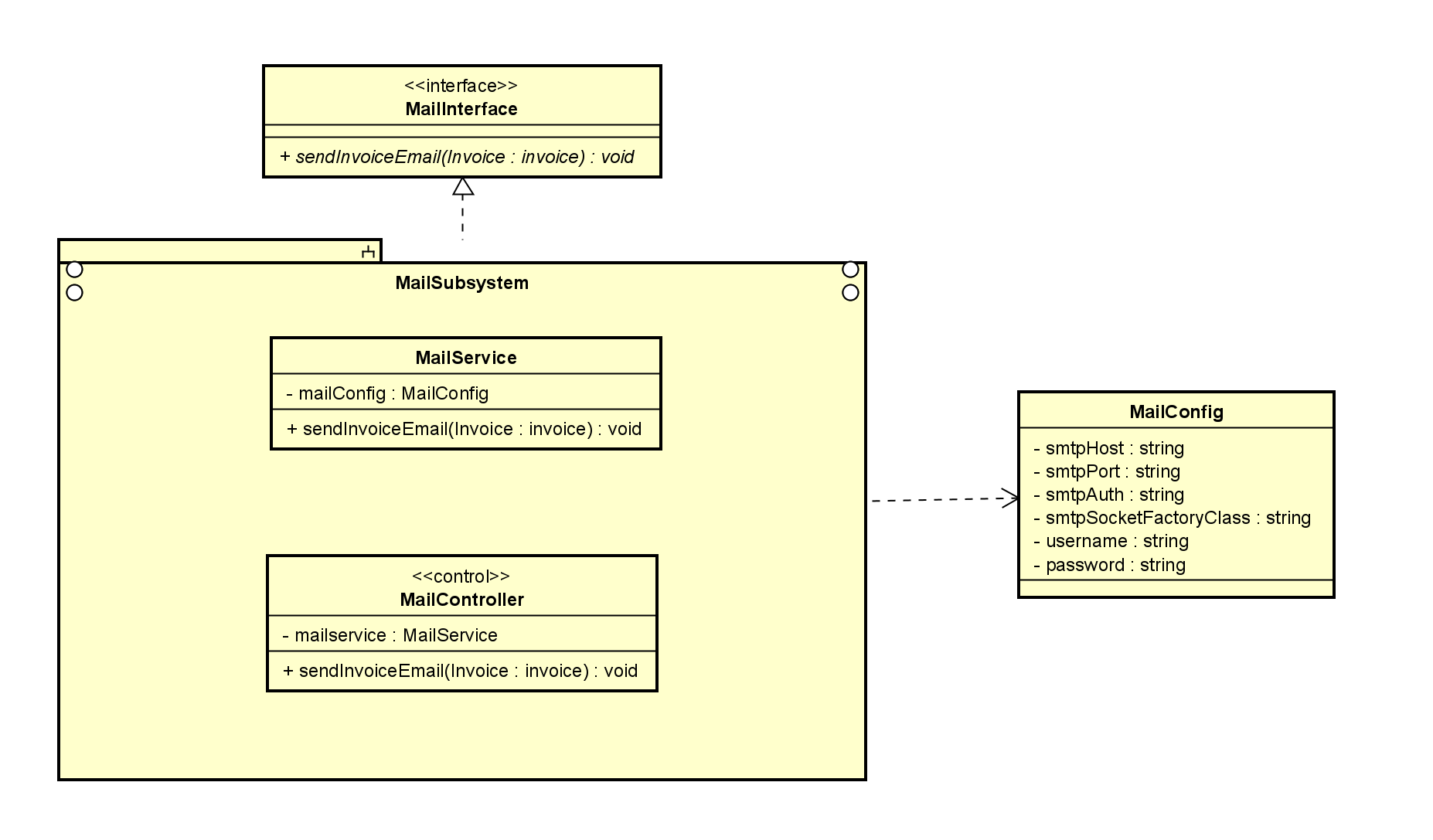
#### Class Diagram for Package Order



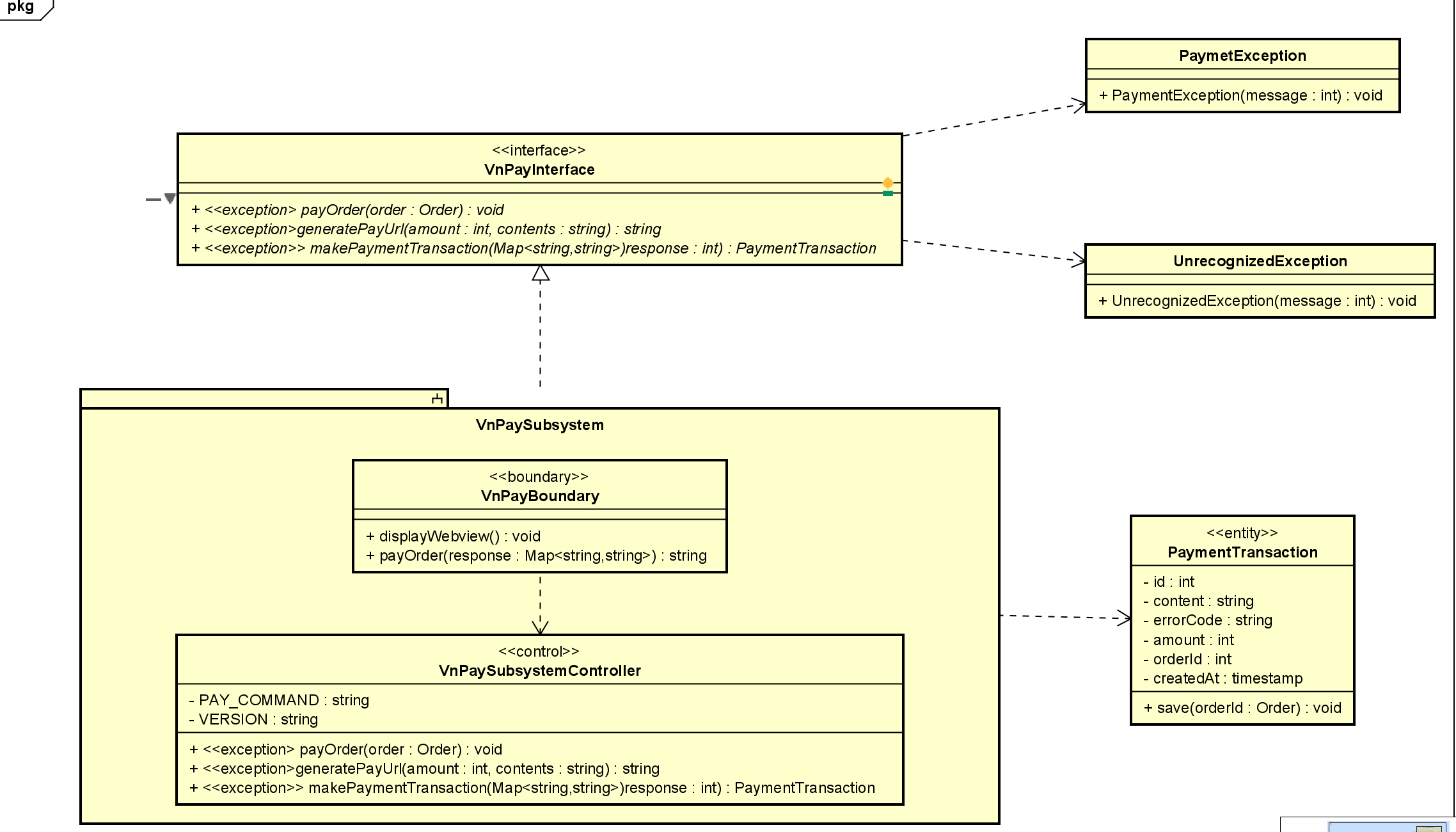
#### Class Diagram for Package Media



#### Class Diagram for Subsystem Mail



#### Class Diagram for Subsystem VnPay



# Design Considerations

## Goals and Guidelines

Goals:

* Performance Optimization:
  + Speed over Memory: The system is designed to prioritize execution speed over memory usage. This goal is essential for applications requiring real-time processing where delays can significantly impact user experience or system functionality.
  + Reason for Desirability: Prioritizing speed ensures that the system can handle high loads and provide immediate responses, which is critical for maintaining user satisfaction and operational efficiency.

Guidelines:

* **Coding Conventions:**
  + **Consistent Naming Conventions**: All variables, functions, and classes will follow a consistent naming convention
  + Commenting and Documentation: Every module, function, and significant code block will include comments explaining its purpose and logic.
* Security guidelines: Implement robust authentication mechanisms and role-based access control to ensure that users can only access resources they are authorized to use.

## Architectural Strategies

Design architecture: MVC – the reason has been stated above.

Design principles: the design will follow the SOLID principles. These principles set has long history and, via practical usages, proved as effective.

Database design: the software will use SQLite as the DBMS.

Framework to be used to develop the software will be JavaFX.

## Coupling and Cohesion

1. Coupling

* Loose coupling: The project follows a loosely coupled design in some aspects. For instance, the use of the [BaseScreenHandler](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) class from which [EditProductScreenHandler](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [CartScreenHandler](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), and [HomeScreenHandler](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) inherit suggests a form of polymorphism. This inheritance strategy allows for a reduction in coupling as changes in the base class do not necessarily require changes in the derived classes, provided the interface remains consistent.
* Evidence of Tight Coupling: However, there are indications of tight coupling as well. For example, the EditProductScreenHandler class directly interacts with controllers such as HomeController and ViewCartController. This direct interaction suggests that changes in the controller classes could necessitate changes in the EditProductScreenHandler class, indicating a higher level of coupling.

1. Cohesion:

* High Cohesion: The project exhibits high cohesion within its components. For example, the CartScreenHandler class is focused on handling the cart screen's UI and interactions, such as viewing the cart and placing orders. This focus on a single aspect of the application suggests that the class has a high level of cohesion.
* Functional Cohesion: The classes such as EditProductScreenHandler, CartScreenHandler, and HomeScreenHandler are examples of functional cohesion, where the elements of the module are grouped because they all contribute to a single well-defined task. Each handler class is responsible for managing the UI and the interactions of their respective screens.

## Design Principles

1. Single Responsibility Principle (SRP)

Each class in the design has a clearly defined responsibility. For example, [PaymentTransaction](http://vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is focused on representing a payment transaction's data, while [PaymentController](http://vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) handles the business logic related to payment processing. This separation of concerns ensures that each class is only responsible for one aspect of the system.

1. Open/Closed Principle (OCP)

Your system's design allows for the extension of behavior without modifying existing code. For instance, the payment processing system interacts with an VNPayInterface, which can be implemented by different subsystems like [VnPaySubsystem](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) . This design allows for the addition of new payment subsystems without changing the core payment processing logic.

1. Liskov Substitution Principle (LSP)

The use of interfaces like VNPayInterface suggests that subclasses (e.g., VnPaySubsystem) can be substituted for their parent class/interface without affecting the correctness of the program. This principle is fundamental for achieving polymorphism in object-oriented design.

1. Dependency Inversion Principle (DIP)

High-level modules such as PaymentController do not depend on low-level modules like VnPaySubsystem directly but rather on abstractions like VNPayInterface. This reduces the coupling between the payment processing logic and the specific payment gateways, making the system more flexible and easier to maintain.

1. Interface Segregation Principle (ISP)

Although not directly observable from the provided excerpts, the principle of designing fine-grained interfaces that are client-specific is generally followed in well-designed systems. For example, the separation of payment processing and transaction data handling suggests adherence to ISP, ensuring that classes only depend on the interfaces that they use.

## Design Patterns

*<Do you use any design patterns for your design? If yes, describe detailly why you use those design patterns? Describe in detail on the solutions and how to implement each design pattern>*