HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

Version 1.3

AIMS - AN ONLINE MEDIA STORE

ITSS SOFTWARE DEVELOPMENT

**Group 08**

Vũ Quyền Gia Linh - 20225734

Chu Vĩnh Khang - 20225863

Cao Quang Huy - 20225724

Nguyễn Đức Nghĩa - 20225895

Nguyễn Trần Việt Bách - 20198206

*Hanoi, June 2025*

**Table of Contents**

[1. Introduction 7](#_Toc201710830)

[1.1. Objective 7](#_Toc201710831)

[1.2. Scope 7](#_Toc201710832)

[1.3. Glossary 8](#_Toc201710833)

[1.4. References 11](#_Toc201710834)

[2. Overall Description 12](#_Toc201710835)

[2.1. General Overview 12](#_Toc201710836)

[2.2. Assumptions/Constraints/Risks 12](#_Toc201710837)

[2.2.1. Assumptions 12](#_Toc201710838)

[2.2.2. Constraints 12](#_Toc201710839)

[2.2.3. Risks 13](#_Toc201710840)

[3. System Architecture and Architecture Design 14](#_Toc201710841)

[3.1. Architectural Patterns 14](#_Toc201710842)

[3.2. Interaction Diagrams 15](#_Toc201710843)

[3.3. Analysis Class Diagrams 16](#_Toc201710844)

[3.4. Unified Analysis Class Diagram 16](#_Toc201710845)

[4. Detailed Design 17](#_Toc201710846)

[4.1. User Interface Design 17](#_Toc201710847)

[4.1.1. Screen Configuration Standardization 17](#_Toc201710848)

[4.1.2. Screen Transition Diagrams 18](#_Toc201710849)

[4.1.3. Screen Specifications 18](#_Toc201710850)

[4.2. Data Modeling 31](#_Toc201710851)

[4.2.1. Conceptual Data Modeling 31](#_Toc201710852)

[4.2.2. Database Design 31](#_Toc201710853)

[4.3. Class Design 35](#_Toc201710854)

[4.3.1. General Class Diagram 35](#_Toc201710855)

[4.3.2. Class Diagrams 35](#_Toc201710856)

[4.3.3. Class Diagram for SubSystem 36](#_Toc201710857)

[4.3.3.1.Email SubSystem 36](#_Toc201710858)

[4.3.3.2. Payment SubSystem 36](#_Toc201710859)

[5. Design Considerations 37](#_Toc201710860)

[5.1. Goals and Guidelines 37](#_Toc201710861)

[5.2. Architectural Strategies 37](#_Toc201710862)

[5.3. Coupling and Cohesion 39](#_Toc201710863)

[5.4. Design Patterns 40](#_Toc201710864)

**List of Figures**

[Figure 1 Place order sequence diagram 15](#_Toc169638719)

[Figure 2 Place rush order sequence diagram 16](#_Toc169638720)

[Figure 3 Pay order sequence diagram 16](#_Toc169638721)

[Figure 4 Cancel order sequence diagram 17](#_Toc169638722)

[Figure 5 Review order sequence diagram 17](#_Toc169638723)

[Figure 6 CRUD products in cart sequence diagram 18](#_Toc169638724)

[Figure 7 Search for products sequence diagram 18](#_Toc169638725)

[Figure 8 Sort products by price sequence diagram 19](#_Toc169638726)

[Figure 9 Login sequence diagram 19](#_Toc169638727)

[Figure 10 CRUD products sequence diagram 20](#_Toc169638728)

[Figure 11 CRUD users sequence diagram 21](#_Toc169638729)

[Figure 12 Change password sequence diagram 22](#_Toc169638730)

[Figure 13 Analysis entity class diagram 23](#_Toc169638731)

[Figure 14 Analysis controller class diagram 24](#_Toc169638732)

[Figure 15 Analysis boundary class diagram 25](#_Toc169638733)

[Figure 16 Unified analysis class diagram 26](#_Toc169638734)

[Figure 17 Screen transition diagram 28](#_Toc169638735)

[Figure 18 ERD diagram 42](#_Toc169638736)

[Figure 19 Database schema 43](#_Toc169638737)

[Figure 20 General Class Diagram 47](#_Toc169638738)

[Figure 21 Class diagram for Controller 48](#_Toc169638739)

[Figure 22 Class diagram for Entity 48](#_Toc169638740)

[Figure 23 Class diagram for Boundary 49](#_Toc169638741)

[Figure 24 Email Subsystem Design 50](#_Toc169638742)

[Figure 25 Payment Subsystem Design 50](#_Toc169638743)

[Figure 26 Singleton for Cart class 62](#_Toc169638744)

[Figure 27 Singleton for SimplejavamailConfig class 63](#_Toc169638745)

[Figure 28 Template method design pattern for DAO 64](#_Toc169638746)

[Figure 29 Strategy design pattern 65](#_Toc169638747)

**List of Tables**

[Table 1 Glossary 8](#_Toc169635068)

[Table 2 Cart Screen 28](#_Toc169635069)

[Table 3 Delivery Form Screen 29](#_Toc169635070)

[Table 4 Rush Delivery Form Screen 30](#_Toc169635071)

[Table 5 Invoice Screen 31](#_Toc169635072)

[Table 6 Pay Order Form 32](#_Toc169635073)

[Table 7 Success Order Screen 33](#_Toc169635074)

[Table 8 Invalid Form Screen 34](#_Toc169635075)

[Table 9 Invalid Rush Order Screen 35](#_Toc169635076)

[Table 10 Login Screen 35](#_Toc169635077)

[Table 11 Home Screen 36](#_Toc169635078)

[Table 12 Admin Screen 37](#_Toc169635079)

[Table 13 Product Manager Screen 39](#_Toc169635080)

[Table 14 Change Account Password Screen 40](#_Toc169635081)

[Table 15 Media Table Details 43](#_Toc169635082)

[Table 16 DVD Table Details 43](#_Toc169635083)

[Table 17 CD\_and\_LP Table Details 44](#_Toc169635084)

[Table 18 Book Table Details 44](#_Toc169635085)

[Table 19 User Table Details 44](#_Toc169635086)

[Table 20 DeliveryInfo Table Details 45](#_Toc169635087)

[Table 21 OrderInfo Table Details 45](#_Toc169635088)

[Table 22 RushOrderInfo Table Details 45](#_Toc169635089)

[Table 23 OrderInfo\_Media Table Details 46](#_Toc169635090)

[Table 24 Payment Transaction Table Details 46](#_Toc169635091)

[Table 25 Invoice Table Details 46](#_Toc169635092)

# Introduction

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

## Objective

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

## Scope

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

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

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

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

## Glossary

Table Glossary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Term** | **Explanation** | **Example** | **Note** |
| 1 | Session | A session is a temporary period of interaction or engagement between a user and a system, during which the user accesses and interacts with software or a website, and the system maintains relevant settings and information. | software session |  |
| 2 | VAT  (Value-added tax) | It is a type of consumption tax that is levied on the value added to goods and services at each stage of production or distribution. VAT is typically implemented as a percentage of the final selling price of a product or service, and it is collected by businesses on behalf of the government. |  |  |
| 3 | API  (Application Programming Protocol) | API is a set of rules, protocols, and tools that allows different software apps to communicate with each other. | VNPay API | AIMS connects to the API of VNPay for transactions. |
| 4 | Payment gateway | A technology service that facilitates the secure transmission of payment information between a merchant's website or application and the financial institutions involved in processing payment transactions. |  |  |
| 5 | GUI  (Graphical user interface) | Refers to the visual elements and interactive components of a software application that allow users to interact with the system using graphical icons, buttons, menus, and windows. |  |  |
| 6 | Credit card | A credit card is a payment card issued by a financial institution, such as a bank or credit union, that allows cardholders to borrow funds up to a predetermined limit to make purchases or pay for goods and services. |  | AIMS currently supports for credit card payment through VNPay |
| 7 | Authentication | Authentication is the process of verifying the identity of a user, device, or system attempting to access a resource or service. |  |  |
| 8 | Response time | Response time refers to the amount of time it takes for a system to respond to a user's request or input. |  |  |

## References

# Overall Description

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

## General Overview

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

## Assumptions/Constraints/Risks

### Assumptions

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

### Constraints

Several global constraints impact the design of the system. The hardware environment imposes limitations on processing power and memory, requiring optimization of software performance. The end-user environment demands a user-friendly interface accessible on various devices. Resource availability and potential volatility necessitate a design that can handle fluctuations efficiently. Compliance with industry standards and interoperability with existing systems are mandatory, influencing interface and protocol design. Licensing requirements must be adhered to, affecting software choices and costs. The data repository needs to ensure secure and efficient storage and distribution of information. Security requirements, including data protection and user privacy, are paramount. Performance requirements dictate the need for fast and reliable system responses. Network communications must support seamless data exchange, and rigorous verification and validation processes are necessary to maintain high quality and reliability. All these constraints collectively shape the system’s architecture and implementation.

### Risks

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

# System Architecture and Architecture Design

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

## Architectural Patterns

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

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

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

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

We select this architecture pattern because many advantages it delivers:

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

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

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

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

## Interaction Diagrams

3.2.1 Place order

Figure 1 Place order sequence diagram

3.2.2 Place rush order

Figure 2 Place rush order sequence diagram

3.2.3 Pay order

Figure 3 Pay order sequence diagram

3.2.4 Cancel order

Figure 4 Cancel order sequence diagram

3.2.5 Review order

Figure 5 Review order sequence diagram

3.2.6 CRUD products in cart

Figure 6 CRUD products in cart sequence diagram

3.2.7 Search for products

Figure 7 Search for products sequence diagram

3.2.8 Sort products by price

Figure 8 Sort products by price sequence diagram

3.2.9 Log in

Figure 9 Login sequence diagram

3.2.10 CRUD products

Figure 10 CRUD products sequence diagram

3.2.11 CRUD users

Figure 11 CRUD users sequence diagram

3.2.12 Change password

Figure 12 Change password sequence diagram

## Analysis Class Diagrams

3.3.1 Package Entity

Figure 13 Analysis entity class diagram

3.3.2 Package Controller

Figure 14 Analysis controller class diagram

3.3.3 Package Boundary

Figure 15 Analysis boundary class diagram

## Unified Analysis Class Diagram

Figure 16 Unified analysis class diagram

# Detailed Design

## User Interface Design

The user interface (UI) of the AIMS application is designed to be intuitive, consistent, and user-friendly, ensuring a seamless experience for all users. The design follows modern UI principles, focusing on clarity, accessibility, and responsiveness across different screens and user roles.

### Screen Configuration Standardization

1. Description

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

2. Window Size and Layout

Each main window is initialized with a consistent size (e.g., 1500x800 pixels for the main frame) and is centered on the user’s screen. Layout managers are used to organize components logically and responsively.

3. Navigation and Toolbars

A standardized toolbar is present at the top of each main screen, providing quick access to primary functions such as Home, Cart, Orders, Admin, Login, and Profile. The toolbar layout and button arrangement remain consistent across all screens.

4. Component Alignment and Spacing

All UI components (labels, buttons, text fields, tables) are aligned and spaced uniformly using layout managers and padding. This ensures visual balance and prevents clutter.

5. Font and Color Scheme

The application uses a consistent font family and standardized font sizes for headers, labels, and content. The color scheme is chosen to provide sufficient contrast and readability, with standard button backgrounds and highlight colors for interactive elements.

6. Button and Input Field Standards

Buttons and input fields maintain a uniform size, background color, and border style. Standard Swing components are used, and customizations are applied consistently.

7. Dialog and Popup Consistency

All dialog boxes (e.g., error messages, confirmations, input dialogs) follow a unified style, with clear titles, messages, and action buttons

8. Responsiveness

The UI is designed to adapt gracefully to different screen sizes and resolutions, ensuring usability on various devices.

By enforcing these screen configuration standards, the application delivers a professional and predictable user experience, making it easier for users to navigate and interact with the system.

### Screen Transition Diagrams

Figure 17 Screen transition diagram

### Screen Specifications

Table Cart Screen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | |
| **Screen specification** | | | CartFrame | |
| **Date of creation** | **Approved by** | | **Reviewed by** | **Person in charge** |
| 23/06/2025 |  | |  | Group 08 |
|  | | | | |
| **Control** | | **Operation** | | **Function** |
| Product List Table | | View | | Displays all products currently in the user's cart, including quantity, price, and subtotal for each item. |
| Quantity Field | | Edit | | Allows the user to change the quantity of each product in the cart. |
| Update Button | | Click | | Updates the cart with the new quantities entered by the user |
| Delete Button | | Click | | Removes the selected product from the cart. |
| Total Price Label | | View | | Shows the total price of all items in the cart. |
| Checkout Button | | Click | | Proceeds to the checkout process, where the user enters shipping and payment information. |
| Continue Shopping Button | | Click | | Returns the user to the product browsing screen. |
| Clear Cart Button | | Click | | Removes all items from the cart. |
| Back Button | | Click | | Returns to the previous screen. |

Table Shipping Information Dialog

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | | |
| **Screen specification** | | | ShippingInformationDialog | | |
| **Date of creation** | **Approved by** | | **Reviewed by** | | **Person in charge** |
| 23/06/2025 |  | |  | | Group 08 |
|  | | | | | |
| **Control** | | **Operation** | | **Function** | |
| Recipient Name (Text Field) | | Input | | User enters the name of the recipient for the order. | |
| Email (Text Field) | | Input | | User enters the recipient’s email address (validated for correct format). | |
| Phone Number (Text Field) | | Input | | User enters the recipient’s phone number (validated for correct format). | |
| Delivery Address (Text Field) | | Input | | User enters the full delivery address. | |
| Province/City (Text Field) | | Input | | User enters the province or city for delivery. | |
| Rush Delivery (Checkbox) | | Select | | User can select this option to request rush (express) delivery. | |
| Rush Delivery Time (Text Field) | | Input | | If rush delivery is selected, user can specify the desired delivery time (format: yyyy-MM-dd HH:mm). | |
| OK Button | | Click | | Confirms and submits the shipping information. | |
| Cancel Button | | Click | | Cancels the operation and closes the dialog | |

Table Main Product Browsing Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | | |
| **Screen specification** | | | MainFrame | | |
| **Date of creation** | **Approved by** | | **Reviewed by** | | **Person in charge** |
| 23/06/2025 |  | |  | | Group 08 |
|  | | | | | |
| **Control** | | **Operation** | | **Function** | |
| Home Button | | Click | | Returns to the home/main screen. | |
| Cart Button | | Click | | Opens the user's shopping cart. | |
| Store Name Label | | View | | Displays the application/store name. | |
| Search Title Field | | Input | | Allows the user to search for products by title. | |
| Category ComboBox | | Select | | Filters products by category (e.g., Book, CD, DVD, etc.). | |
| Rush Delivery ComboBox | | Select | | Filters products by rush delivery eligibility (All, Rush Eligible, Non-Rush). | |
| Sort Type ComboBox | | Select | | Allows sorting products by title or price. | |
| Sort Order ComboBox | | Select | | Allows sorting in ascending or descending order. | |
| Search Button | | Click | | Applies the search and filter criteria to the product list. | |
| Reset Button | | Click | | Clears all search and filter fields, showing all products. | |
| Login Button | | Click | | Opens the login dialog for user authentication. | |
| Sign Up Button | | Click | | Opens the sign-up dialog for new user registration. | |
| Session Label | | View | | Displays the current session ID or user status. | |
| Product Panel (Grid) | | View | | Shows a grid of product cards, each displaying product details (title, category, price, stock, delivery type). | |
| Detail Button (per product) | | Click | | Opens a detailed view of the selected product. | |
| Add to Cart Button (per product) | | Click | | Adds the selected product to the user's cart. | |
| Previous Button | | Click | | Navigates to the previous page of products. | |
| Page Label | | View | | Displays the current page number. | |
| Next Button | | Click | | Navigates to the next page of products. | |

Table Log In Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | | |
| **Screen specification** | | | LoginFrame | | |
| **Date of creation** | **Approved by** | | **Reviewed by** | | **Person in charge** |
| 23/06/2025 |  | |  | | Group 08 |
|  | | | | | |
| **Control** | | **Operation** | | **Function** | |
| Username Field | | Input | | User enters their username. | |
| Password Field | | Input | | User enters their password (masked input). | |
| Home Button | | Click | | Returns the user to the main/home screen without logging in. | |
| Login Button | | Click | | Submits the entered credentials to authenticate and log in the user. | |

Table Sign Up Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | | |
| **Screen specification** | | | SignUpFrame | | |
| **Date of creation** | **Approved by** | | **Reviewed by** | | **Person in charge** |
| 23/06/2025 |  | |  | | Group 08 |
|  | | | | | |
| **Control** | | **Operation** | | **Function** | |
| Username Field | | Input | | User enters their desired username. | |
| Password Field | | Input | | User enters their desired password (masked input). | |
| Email Field | | Input | | User enters their email address (validated for correct format). | |
| Home Button | | Click | | Returns the user to the main/home screen without signing up. | |
| Sign Up Button | | Click | | Submits the entered information to create a new user account. | |

Table Profile Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | | |
| **Screen specification** | | | ProfileFrame | | |
| **Date of creation** | **Approved by** | | **Reviewed by** | | **Person in charge** |
| 23/06/2025 |  | |  | | Group 08 |
|  | | | | | |
| **Control** | | **Operation** | | **Function** | |
| Username Field | | View/Edit | | Displays the current username (may be editable or read-only depending on implementation). | |
| Email Field | | View/Edit | | Displays and allows editing of the user's email address. | |
| New Password Field | | Input | | Allows the user to enter a new password (optional; if left blank, password is unchanged). | |
| Save Button | | Click | | Saves the updated profile information (username, email, and/or password). | |
| Cancel Button | | Click | | Cancels the operation and closes the profile screen without saving changes. | |

Table Admin Panel – User Management Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | | |
| **Screen specification** | | | AdminFrame (User Management) | | |
| **Date of creation** | **Approved by** | | **Reviewed by** | | **Person in charge** |
| 23/06/2025 |  | |  | | Group 08 |
|  | | | | | |
| **Control** | | **Operation** | | **Function** | |
| User Table | | View/Select | | Displays a list of all users with columns: ID, Username, Email, Role, Blocked status. Allows row selection for editing. | |
| Add User Button | | Click | | Opens a dialog to add a new user to the system. | |
| Edit Selected Button | | Click | | Opens a dialog to edit the details of the selected user in the table. | |

Table Edit Product Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | | |
| **Screen specification** | | | AddProductFrame | | |
| **Date of creation** | **Approved by** | | **Reviewed by** | | **Person in charge** |
| 23/06/2025 |  | |  | | Group 08 |
|  | | | | | |
| **Control** | | **Operation** | | **Function** | |
| Title Field | | Input | | Enter or edit the product title. | |
| Category Dropdown | | Select | | Select the product category (e.g., BOOK, CD, LP\_RECORD, DVD). | |
| Price Field | | Input | | Enter or edit the product price (VND). | |
| Value Field | | Input | | Enter or edit the product value (VND, before VAT). | |
| Stock Quantity Field | | Input | | Enter or edit the available stock quantity. | |
| Weight Field | | Input | | Enter or edit the product weight (kg). | |
| Rush Eligible Checkbox | | Select | | Mark if the product is eligible for rush delivery. | |

Note:

Fields and sections may change dynamically based on the selected category (e.g., Book, CD, DVD, etc.).

Input validation is performed for required fields and correct data formats (e.g., date, numbers).

The screen is used for both adding new products and editing existing products.

Table Order Management Screen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | |
| **Screen specification** | | | OrderFrame | |
| **Date of creation** | **Approved by** | | **Reviewed by** | **Person in charge** |
| 23/06/2025 |  | |  | Group 08 |
|  | | | | |
| **Control** | | **Operation** | | **Function** |
| Order Table | | View/Select | | Displays a list of all orders with columns: ID, Recipient, Email, Address, Status, and Total. Allows row selection for further actions. |
| Change Status Button | | Click | | Opens a dialog or enables editing to change the status of the selected order (e.g., Pending, Approved, Rejected, Canceled). |
| View Detail Button | | Click | | Opens a detailed view of the selected order, showing all order items and additional information. |

Table Edit Order Status Dialog

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AIMS SOFTWARE** | | | | |
| **Screen specification** | | | EditOrderStatusDialog | |
| **Date of creation** | **Approved by** | | **Reviewed by** | **Person in charge** |
| 23/06/2025 |  | |  | Group 08 |
|  | | | | |
| **Control** | | **Operation** | | **Function** |
| Status Dropdown | | Select | | Allows the user to select a new status for the order (e.g., Pending, Approved, Rejected, Canceled). |
| OK Button | | Click | | Confirms the status change and updates the order in the system. |
| Cancel Button | | Click | | Cancels the operation and closes the dialog without making changes. |

## Data Modeling

### Conceptual Data Modeling

Figure 18 ERD diagram

### Database Design

#### Database Management System

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

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

Additionally, PostgreSQL integrates well with numerous programming languages and platforms, making it highly adaptable to different development environments. Its extensive documentation and active community support make it easier to troubleshoot issues and implement best practices. Overall, PostgreSQL’s proven track record and comprehensive features make it the ideal choice for managing our database requirements.

#### Relational Schema

Figure 19 Database schema

#### Database Detail Design

Table Products Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | product\_id | integer | nextval | yes | increment |
| 2 |  |  | title | varchar |  | yes |  |
| 3 |  |  | category | varchar |  | yes |  |
| 4 |  |  | value | numeric |  | yes |  |
| 5 |  |  | price | numeric |  | yes |  |
| 6 |  |  | stock\_quantity | integer |  | yes |  |
| 7 |  |  | is\_rush\_eligible | boolean | false |  |  |
| 8 |  |  | weight | numeric |  | yes |  |
| 9 |  |  | created\_at | timestamp | CURRENT\_TIMESTAMP |  |  |
| 10 |  |  | updated\_at | timestamp | CURRENT\_TIMESTAMP |  |  |
| 11 |  |  | authors | varchar |  |  |  |
| 12 |  |  | cover\_type | varchar |  |  |  |
| 13 |  |  | publisher | varchar |  |  |  |
| 14 |  |  | publication\_date | date |  |  |  |
| 15 |  |  | num\_pages | integer |  |  |  |
| 16 |  |  | book\_language | varchar |  |  |  |
| 17 |  |  | book\_genre | varchar |  |  |  |
| 18 |  |  | artists | varchar |  |  |  |
| 19 |  |  | record\_label | varchar |  |  |  |
| 20 |  |  | tracklist | varchar |  |  |  |
| 21 |  |  | music\_genre | varchar |  |  |  |
| 22 |  |  | release\_date | date |  |  |  |
| 23 |  |  | disc\_type | varchar |  |  |  |
| 24 |  |  | director | varchar |  |  |  |
| 25 |  |  | runtime | integer |  |  |  |
| 26 |  |  | studio | varchar |  |  |  |
| 27 |  |  | dvd\_language | varchar |  |  |  |
| 28 |  |  | subtitles | varchar |  |  |  |
| 29 |  |  | available | varchar | 'yes'::character varying |  |  |

Table Product History Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | history\_id | integer | nextval | yes | increment |
| 2 |  | x | product\_id | integer |  | yes |  |
| 3 |  |  | operation | varchar |  | yes |  |
| 4 |  |  | description | text |  |  |  |
| 5 |  |  | user\_id | integer |  | yes |  |
| 6 |  |  | created\_at | timestamp | CURRENT\_TIMESTAMP |  |  |

Table Cart Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | cart\_id | integer | nextval | yes | increment |
| 2 |  |  | session\_id | varchar |  | yes |  |
| 3 |  |  | created\_at | timestamp | CURRENT\_TIMESTAMP |  |  |

Table Cart Items Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | cart\_item\_id | integer | nextval | yes | increment |
| 2 |  |  | cart\_id | integer |  | yes |  |
| 3 |  |  | product\_id | integer |  | yes |  |
| 4 |  |  | quantity | integer |  | yes |  |

Table User Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | user\_id | integer |  | yes | increment |
| 2 |  |  | username | varchar |  | yes |  |
| 3 |  |  | password | varchar |  | yes |  |
| 4 |  |  | email | varchar |  | yes |  |
| 5 |  |  | isAdmin | boolean |  | yes |  |

Table Orders Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | order\_id | integer | nextval | yes | increment |
| 2 |  | x | user\_id | integer |  |  |  |
| 3 |  |  | recipient\_name | varchar |  | yes |  |
| 4 |  |  | email | varchar |  | yes |  |
| 5 |  |  | phone\_number | varchar |  | yes |  |
| 6 |  |  | delivery\_address | varchar |  | yes |  |
| 7 |  |  | province\_city | varchar |  |  |  |
| 8 |  |  | delivery\_fee | numeric |  |  |  |
| 9 |  |  | is\_rush\_delivery | boolean | false |  |  |
| 10 |  |  | rush\_delivery\_time | timestamp |  |  |  |
| 11 |  |  | total\_amount | numeric |  | yes |  |
| 12 |  |  | status | varchar |  | yes |  |
| 13 |  |  | created\_at | timestamp | CURRENT\_TIMESTAMP |  |  |

Table Order Items Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | order\_item\_id | integer | nextval | yes | increment |
| 2 |  | x | order\_id | integer |  | yes |  |
| 3 |  |  | product\_id | integer |  | yes |  |
| 4 |  |  | quantity | integer |  | yes |  |
| 5 |  |  | price | numeric |  | yes |  |

Table Transaction Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | transaction\_id | integer | nextval | yes | increment |
| 2 |  | x | order\_id | integer |  | yes |  |
| 3 |  |  | transaction\_vnpay\_id | varchar |  | yes |  |
| 4 |  |  | amount | numeric |  | yes |  |
| 5 |  |  | content | varchar |  |  |  |
| 6 |  |  | created\_at | timestamp | CURRENT\_TIMESTAMP |  |  |

Table Users Table Details

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *#* | *PK* | *FK* | *Column name* | *Data type* | *Default value* | *Mandatory* | *Description* |
| 1 | x |  | user\_id | integer | nextval | yes |  |
| 2 |  |  | username | varchar |  | yes |  |
| 3 |  |  | password | varchar |  | yes |  |
| 4 |  |  | email | varchar |  | yes |  |
|  |  |  | role | varchar |  | yes |  |
|  |  |  | is\_blocked | boolean | false |  |  |
|  |  |  | created\_at | timestamp | CURRENT\_TIMESTAMP |  |  |

**INDEX**

*-***- Index: product\_id\_index**

CREATE INDEX product\_id\_index ON products (product\_id);

**-- Index: transaction\_id\_index, order\_id\_index**

CREATE INDEX transaction\_id\_index ON Transaction (transaction\_id);

CREATE INDEX order\_id\_index ON Orders (order\_id);

## Class Design

### General Class Diagram

Figure 20 General Class Diagram

### Class Diagrams

#### Class Diagram for Controller

Figure 21 Class diagram for Controller

#### Class Diagram for Entity

Figure 22 Class diagram for Entity

* + - 1. ***Class Diagram for Boundary***

Figure 23 Class diagram for Boundary

### Class Diagram for SubSystem

### 4.3.3.1.Email SubSystem

Figure 24 Email Subsystem Design

### 4.3.3.2. Payment SubSystem

Figure 25 Payment Subsystem Design

# Design Considerations

## Goals and Guidelines

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

**Goals**

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

- Scalability: The system architecture is designed to scale efficiently to accommodate growing numbers of users and data. This involves modular design, use of scalable technologies like PostgreSQL for the database.

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

**Guidelines**

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

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

- Error Handling: Implementing robust error handling mechanisms ensures that the system can gracefully recover from unexpected issues and provide meaningful feedback to users and developers. This includes logging errors for diagnostics and providing user-friendly error messages.

## Architectural Strategies

**Programming Language and Frameworks**

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

**Database Management System**

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

**User Interface Paradigms**

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

**Error Detection and Recovery**

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

**Testing with JUnit**

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

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

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

**Version Control with GitHub and 'Feature/Release' Workflow**

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

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

## Coupling and Cohesion

**Coupling**

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

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

**Cohesion**

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

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

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

## Design Patterns

**1. Singleton Design Pattern**

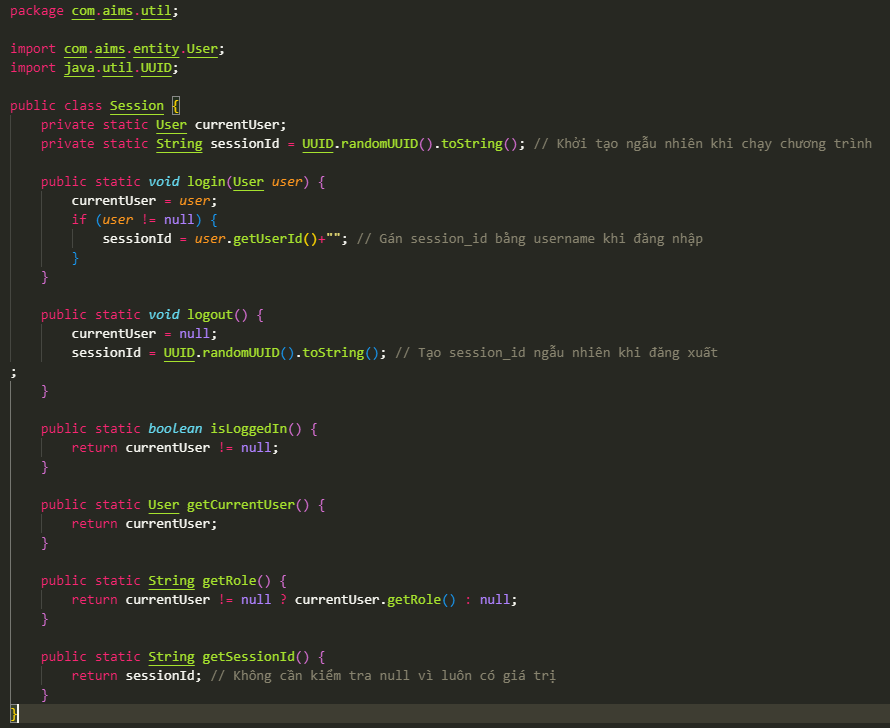


Figure 26 Singleton for Session

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

**2. DAO Design Pattern (Data Access Object)**

Encapsulates all access to the data source. The DAO manages the connection with the data source to obtain and store data. Separates low-level data accessing API or operations from high-level business services.

**2. DAO Design Pattern (Data Access Object)**

Encapsulates all access to the data source. The DAO manages the connection with the data source to obtain and store data. Separates low-level data accessing API or operations from high-level business services.

**3. MVC Design Pattern (Model-View-Controller) or ECB Design Pattern (Entity-Controller-Boundary)**

Separates the application into three interconnected components: Model: Represents the data and business logic. View: Handles the UI and presentation. Controller: Handles the input, processes it (using Model), and updates the View. Improves code organization, maintainability, and scalability.

4. Factory Design Pattern (partially, in UI creation)

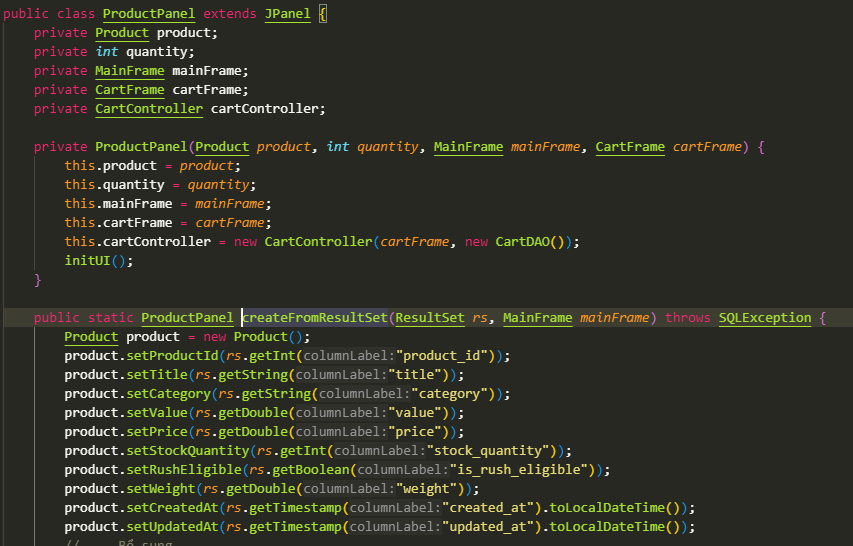


Figure 27 Factory for ProductPanel

Methods like ProductPanel.createFromResultSet(...) act as a simple factory for creating ProductPanel objects from database results. Encapsulates object creation logic. Centralizes and simplifies object creation, especially when it involves complex setup.