Biblionex

System Design Document

September 12, 2024

Contents

1 Introduction **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Definitions, Acronyms, and Abbreviations . . . 1

2 System Overview **2**
- 2.1 System Context . . . 2
- 2.2 Objectives and Goals . . . 2
- 2.3 Stakeholders . . . 2
- 2.4 High-Level Functionality . . . 2
  - 2.4.1 Librarian Functions . . . 2
  - 2.4.2 Patron Functions . . . 3
- 2.5 Constraints and Assumptions . . . 3
- 2.6 Use Case Diagram . . . 4

3 System Architecture **5**
- 3.1 MVC Pattern . . . 5
  - 3.1.1 Model . . . 5
  - 3.1.2 View . . . 5
  - 3.1.3 Controller . . . 5
- 3.2 Observer Pattern . . . 5
  - 3.2.1 Implementation . . . 6

4 Detailed Design **7**
- 4.1 Class Diagram . . . 7
- 4.2 Sequence Diagrams . . . 9
- 4.3 State Diagrams . . . 11
- 4.4 Activity Diagrams . . . 11
- 4.5 Deployment Diagram . . . 12
  - 4.5.1 Two-Tier Architecture . . . 12
  - 4.5.2 Three-Tier Architecture . . . 13

References **14**

List of Figures

2.1 Use Case Diagram 4
4.1 First half of the class diagram for the model package 7
4.2 Second half of the class diagram for the model package 8
4.3 Sequence diagram when patron pays a fine 9
4.4 Sequence diagram when librarian creates a new vendor 10
4.5 Sequence diagram for checkin process 10
4.6 State diagram for login 11
4.7 Activity diagram for checkout process 11
4.8 Deployment diagram for a 2-tier architecture 12
4.9 Deployment diagram for a 3-tier architecture 13

1 | Introduction

1.1 | Purpose

The purpose of this System Design Document (SDD) is to describe the architecture and design of a Library Management System (LMS). This document provides a comprehensive overview of the system's structure, components, and interactions, serving as a blueprint for developers to implement the system. For access to the source code, ongoing updates, and additional documentation, please visit the project's GitHub repository (creme332, 2024).

1.2 | Scope

This document covers the design and architecture of the LMS, including high-level architectural designs, detailed designs of system components, and data models. It encompasses all modules of the LMS, such as user management, book cataloging, borrowing and returning books, and reporting.

1.3 | Definitions, Acronyms, and Abbreviations

- Check-Out: The process of borrowing materials from the library. When a patron checks out an item, it is recorded in the system, and the item is temporarily removed from the library's available inventory.
- Check-In: The process of returning borrowed materials to the library. The system updates the inventory to reflect that the item is now available for other patrons.
- LMS: Library Management System
- UML: Unified Modeling Language
- DBMS: Database Management System
- MVC: Model-View-Controller
- JVM: Java Virtual Machine

2 | System Overview

2.1 | System Context

The LMS operates within a physical library environment. It facilitates the management of library resources catering to the needs of two primary stakeholders: patrons and librarians.

2.2 | Objectives and Goals

The primary objectives of the LMS are to streamline library operations, enhance user experience, and provide robust reporting capabilities. The system aims to automate book cataloging, borrowing, and returning processes while ensuring data integrity and security.

2.3 | Stakeholders

Key stakeholders include:

- Patrons: Need to search for books, borrow and return books.
- Librarians: Need to manage book cataloging, handle user accounts, and generate reports on library activities.

2.4 | High-Level Functionality

This section outlines the primary functions and features of the LMS, detailing how it supports the needs of both librarians and patrons. The functionality is designed to ensure efficient library operations and a seamless user experience.

2.4.1 | Librarian Functions

- **User Authentication and Management:**
  - **Log In:** Librarians can log into the system using their credentials to access administrative functions.
  - **Log Out:** Librarians can securely log out of the system when their session ends.
- **Library Catalog Management:**
  - **Add Materials:** Librarians can add new materials to the library catalog, including books, journals, and other resources.
  - **Update Materials:** Existing materials in the catalog can be updated with new information or corrections.
  - **Delete Materials:** Materials that are no longer part of the library's collection can be removed from the catalog.
  - **Search Catalog:** Librarians can search the library catalog to find specific materials using various search criteria.
- **Patron Management:**
  - **Create Patron Accounts:** Librarians can create new accounts for patrons, including entering relevant personal information.
  - **Search Patrons:** Librarians can search for existing patron accounts by email or name to manage account details and resolve issues.
- **Material Transactions:**
  - **Check In Materials:** Librarians can process returned materials, updating their status and availability.
  - **Check Out Materials:** Librarians can check out materials to patrons, updating their status in the catalog.
  - **Renew Materials:** Librarians can extend the loan period for materials upon patron request.

- **Material Ordering:**
  - **Order New Materials:** Librarians can place orders for new materials to be added to the library's collection.
- **Reporting:**
  - **Generate Reports:** Librarians can generate various reports on library operations and usage patterns, including inventory reports, borrowing statistics, and overdue items.

2.4.2 | Patron Functions

- **User Authentication and Account Management:**
  - **Log In:** Patrons can log into the system using their credentials to access personal services and manage their accounts.
  - **Log Out:** Patrons can securely log out of the system when their session ends.
  - **Change Account Information:** Patrons can update their account details, such as contact information and preferences.
  - **Change Password:** Patrons can change their account password for security purposes.
- **Catalog Interaction:**
  - **Browse Catalog:** Patrons can browse the library catalog to explore available materials.
  - **Search for Items:** Patrons can search for specific items using keywords, titles, authors, or categories.
  - **View Material Details:** Patrons can view detailed information and availability status for materials in the catalog.
- **Material Transactions:**
  - **Borrow Materials:** Patrons can borrow available materials from the library.
  - **Renew Loans:** Patrons can renew existing loans to extend the borrowing period.
  - **View Loan History:** Patrons can view their borrowing history, including past and current loans.
- **Account Management:**
  - **Manage Account Details:** Patrons can manage their account details, including contact information and preferences.
  - **Pay Outstanding Fines:** Patrons can pay any outstanding fines or fees associated with their account.

2.5 | Constraints and Assumptions

- **Physical Materials:** The LMS handles only physical materials. The system does not support digital resources or e-books.
- **Checkout Process:** Patrons cannot check out materials directly through the LMS. To borrow materials, patrons must visit the library in person and complete the checkout process with the assistance of a librarian.
- **Tracked Materials:** The system tracks only specific types of materials: books, videos, and journals. Other types of materials or resources are not managed by the LMS.

2.6 | Use Case Diagram

Figure 2.1: Use Case Diagram

This use case diagram illustrates the interactions between different user roles and the LMS, **with the assumption that all users are already authenticated**. As such, the diagram focuses solely on the core functionalities of the system, excluding any login or logout processes.
The term "manage" refers to performing CRUD (Create, Read, Update, Delete) operations on various resources. For example, "Manage material" involves adding new materials to the system, viewing or searching for material details, updating existing material information, and removing materials from the library database.

3 | System Architecture

3.1 | MVC Pattern

The LMS adopts MVC architectural pattern to separate concerns and enhance modularity. The MVC pattern divides the application into three main components:

3.1.1 | Model

- **Definition:** The Model represents the core data and business logic of the LMS. It manages the data related to books, patrons, transactions, and other resources.
- **Responsibilities:**
  - Maintain and manage the state of the application data.
  - Define and enforce business rules, such as loan periods and fines.
  - Provide data to the View and respond to updates from the Controller.
- **Example:** In the LMS, the Model includes classes like Librarian, Patron, and Loan, which handle operations such as check-in and checkout.

3.1.2 | View

- **Definition:** The View is responsible for presenting data to the user and rendering the user interface. It displays information from the Model and allows user interaction.
- **Responsibilities:**
  - Render data in a user-friendly format.
  - Provide interface elements like search forms, catalogs, and user profiles.
  - Relay user inputs to the Controller for processing.
- **Example:** The View in the LMS includes the interfaces for browsing the catalog, managing user accounts, and processing transactions.

3.1.3 | Controller

- **Definition:** The Controller serves as an intermediary between the Model and the View. It processes user input, updates the Model, and refreshes the View.
- **Responsibilities:**
  - Handle user actions and input from the View.
  - Update the Model based on user interactions, such as checking out a book or updating account details.
  - Ensure the View is updated to reflect changes in the Model.
- **Example:** In the LMS, the Controller handles operations like processing a book checkout request and managing user registrations.

3.2 | Observer Pattern

In the LMS, the Observer pattern is used to manage the global state of the application and handle screen transitions seamlessly. This pattern ensures that changes in the application's state are effectively communicated to the component responsible for handling UI updates.

3.2.1 | Implementation

We implement the **Observer Pattern** using Java's built-in **JavaBeans** mechanism. JavaBeans naturally support property change notifications, which align well with the Observer Pattern. This simplifies our implementation of the pattern, as JavaBeans provide ready-to-use features like event firing and listener registration.

- **PropertyChangeListener** acts as the observer, where components can register to listen for changes.
- **PropertyChangeSupport** manages the list of observers and handles notifying them when a property in the JavaBean (subject) changes.

The `AppState` class serves as the subject in the Observer pattern. It maintains the global application state, such as the currently displayed screen. It provides methods for observers to subscribe to changes and notifies them when the state changes.
The `FrameController` class acts as an observer of the `AppState` class. It listens for changes in the `screen` variable within `AppState` and executes the logic needed to switch screens based on the updated state.
Interaction

- **State Change**: `AppState` updates the screen variable.
- **Notification**: `AppState` notifies `FrameController` of the change.
- **UI Update**: `FrameController` adjusts the user interface to display the new screen.

Benefits

- **Decoupling**: Separates the state management (`AppState`) from the UI logic (`FrameController`), promoting a modular design.
- **Flexibility**: Allows the addition of new observers or changes to existing ones without altering the core state management class.
- **Maintainability**: Simplifies the process of updating the UI in response to state changes, making the system easier to maintain and extend.

4 | Detailed Design

This section provides an in-depth look at the system's design, including various UML diagrams that describe the architecture and interactions within the LMS. All diagrams presented in this section adhere to the UML 2.5.1 specification (OMG, 2017), ensuring consistency and clarity in the representation of the system's components and interactions.

4.1 | Class Diagram

The class diagrams in this section illustrate the primary classes and their relationships within the LMS. To maintain clarity and focus on the core functionalities, basic methods such as constructors, getters, setters, and some database access methods have been purposefully excluded.

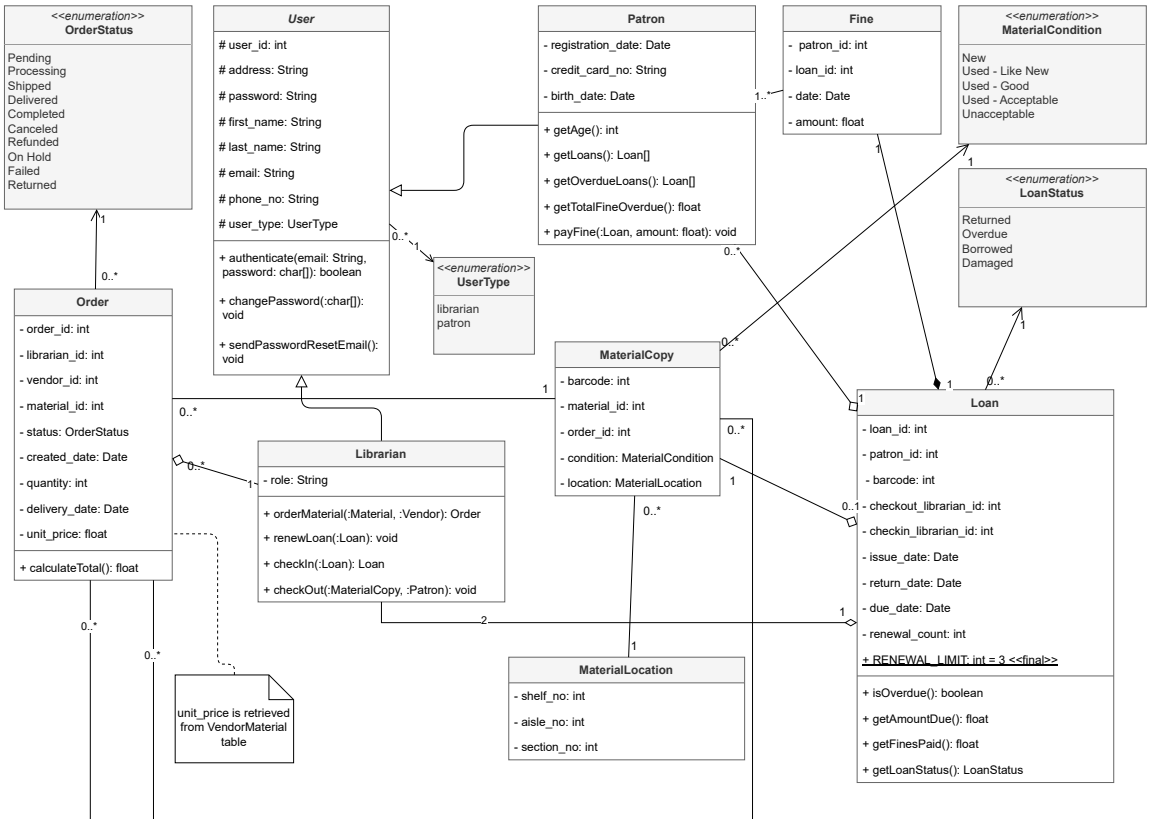


Figure 4.1: First half of the class diagram for the model package

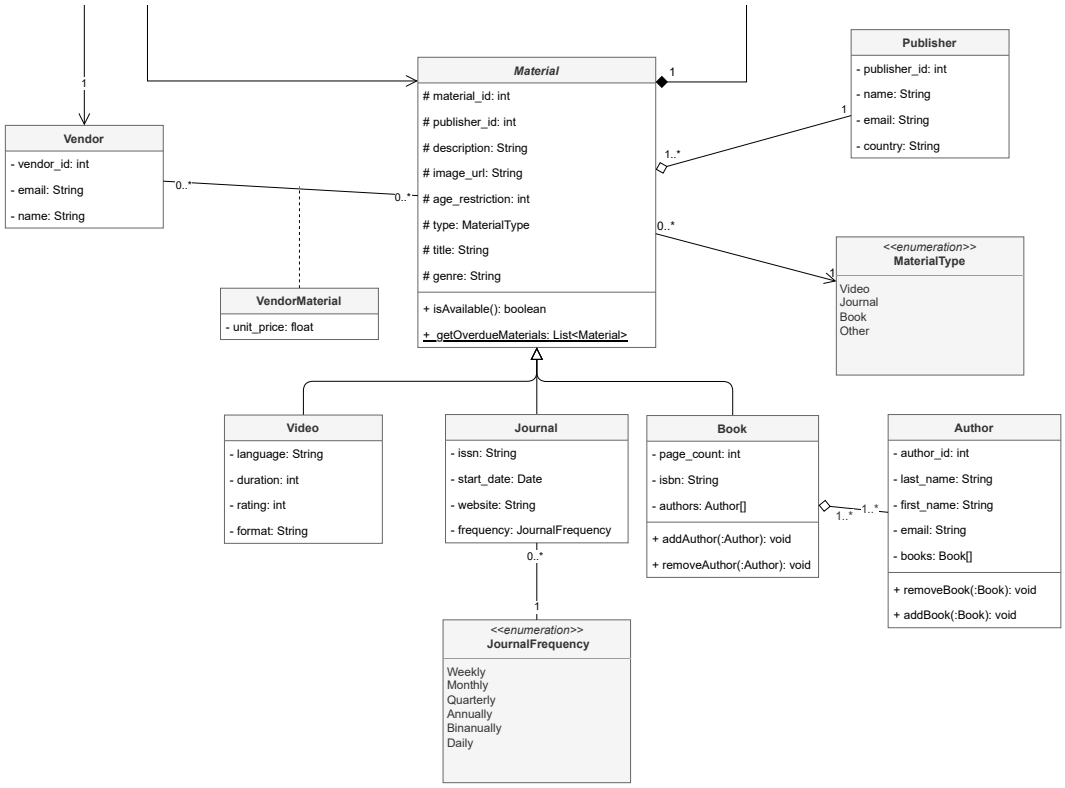


Figure 4.2: Second half of the class diagram for the model package

4.2 | Sequence Diagrams

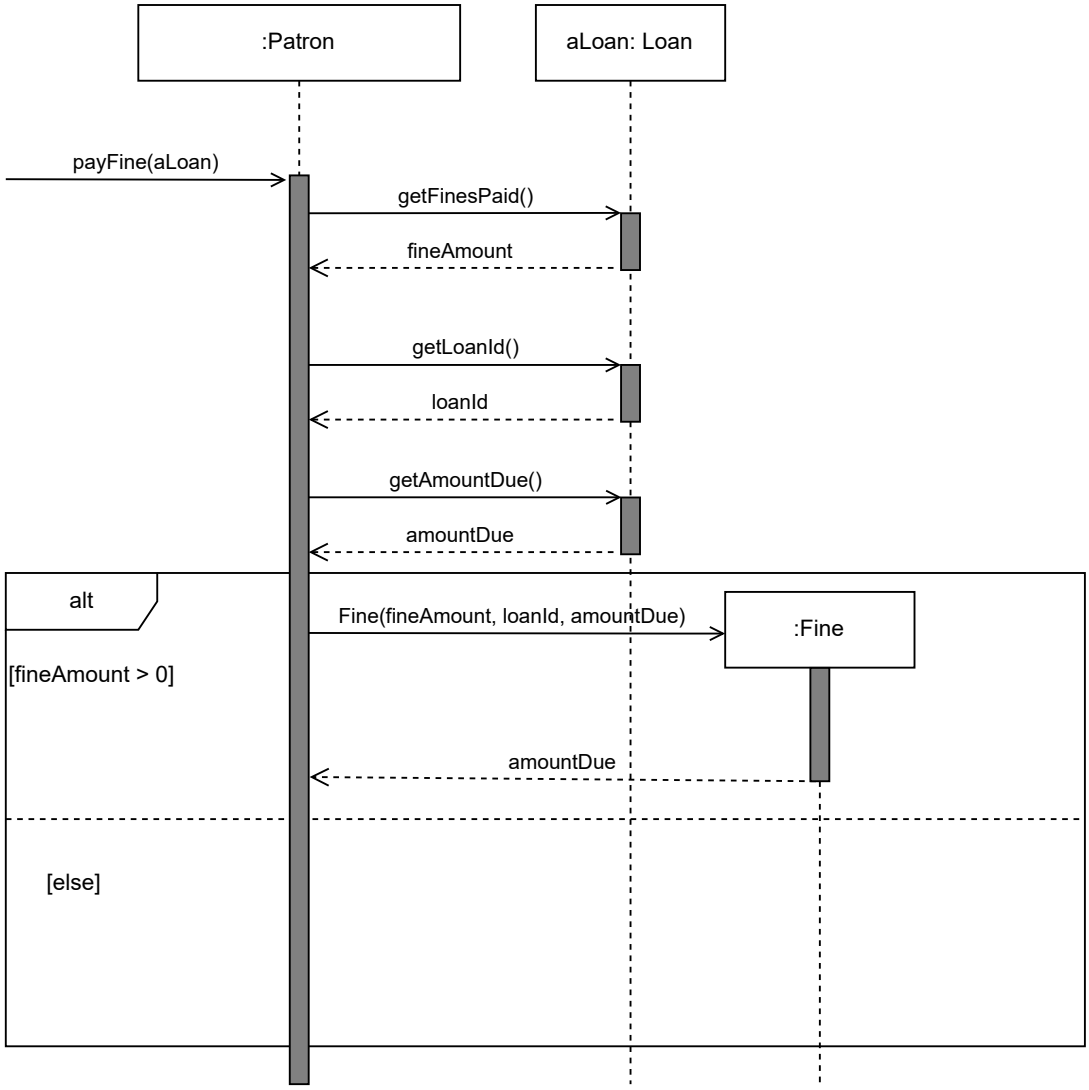


Figure 4.3: Sequence diagram when patron pays a fine

The sequence diagram above details the interactions involved when a patron pays a fine for an overdue loan. The process begins with the invocation of the `payFine` method, which is called with the specific material loan as its parameter. The system then calculates the fine amount and the total amount due. If the total amount is valid, a new fine record is created in the database.

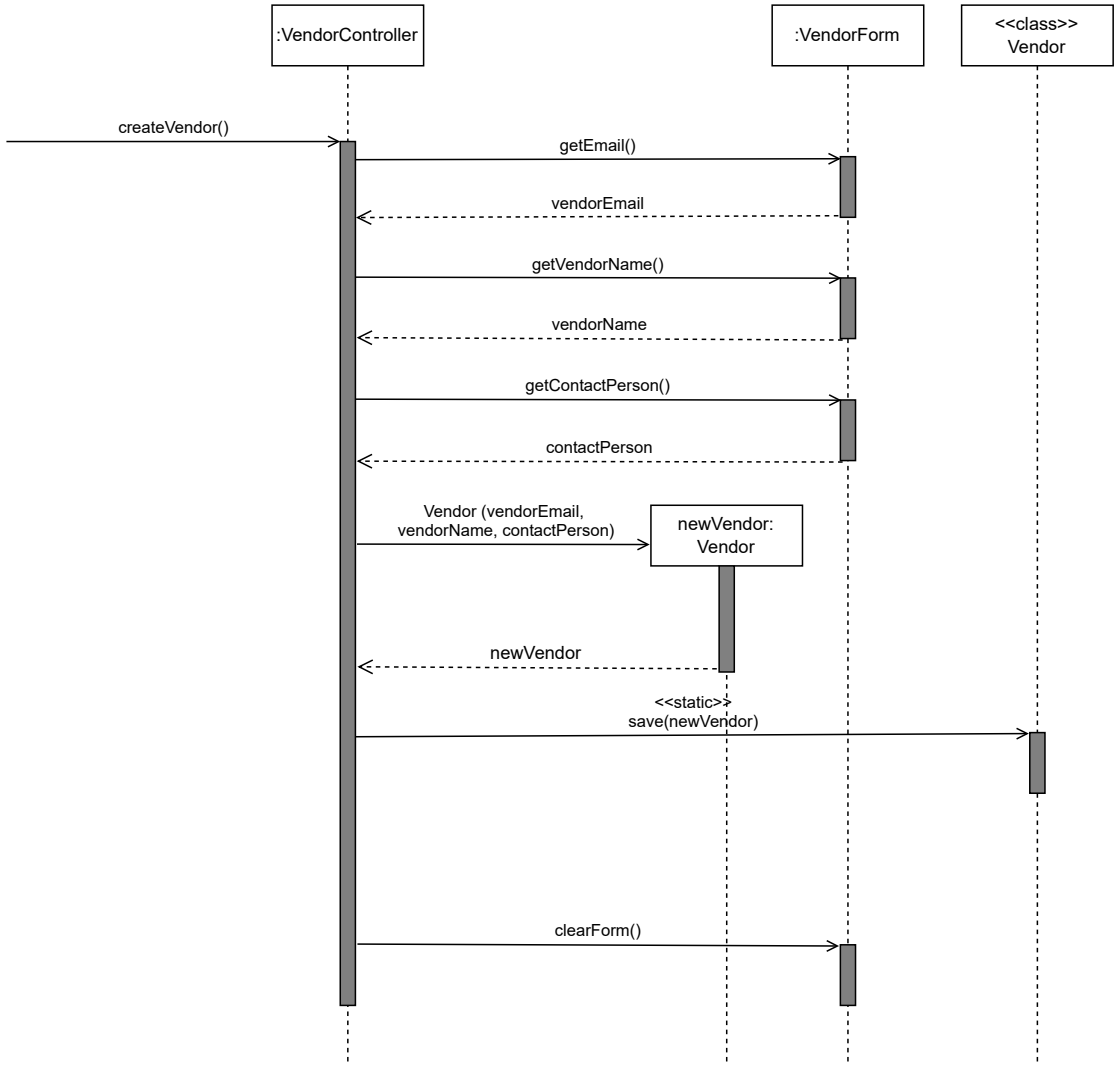


Figure 4.4: Sequence diagram when librarian creates a new vendor

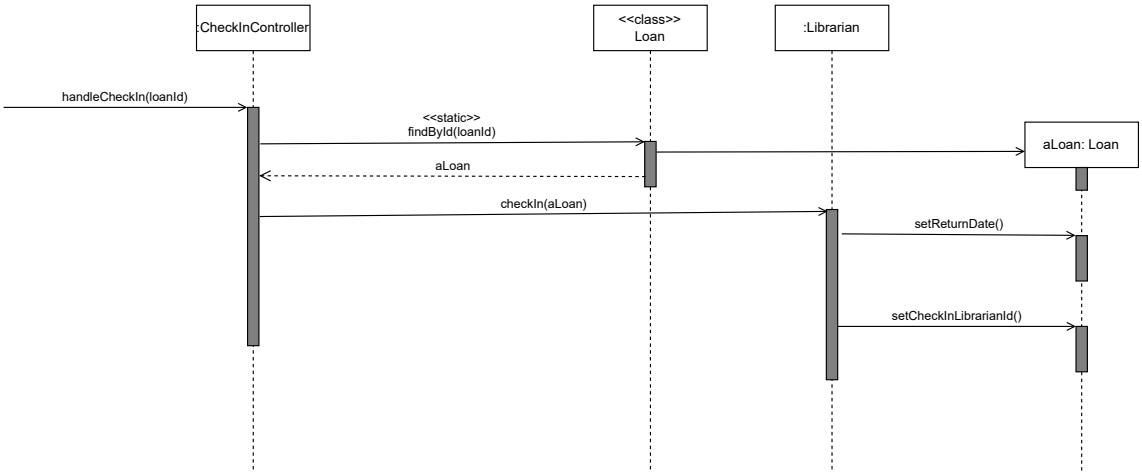


Figure 4.5: Sequence diagram for checkin process

4.3 | State Diagrams

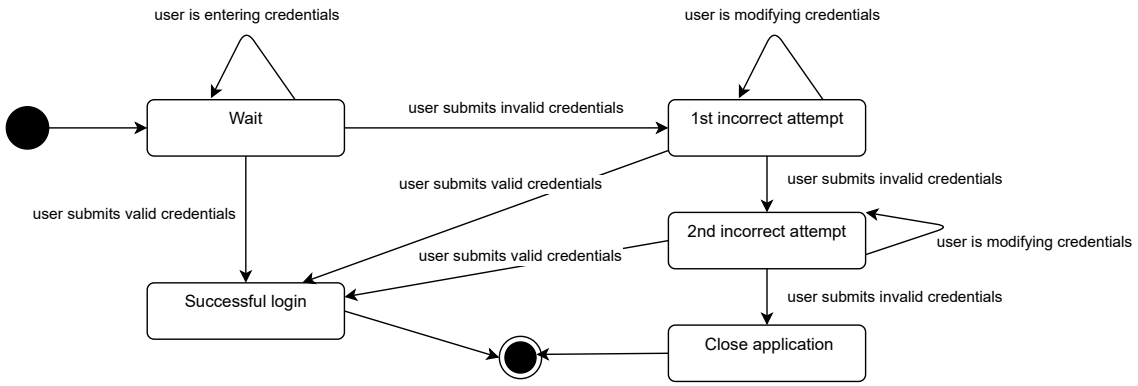


Figure 4.6: State diagram for login

The state diagram illustrates the login process, detailing the flow of user authentication. The user is allowed up to three attempts to enter their credentials. If the credentials are incorrect on the third attempt, the application will terminate the login session.

4.4 | Activity Diagrams

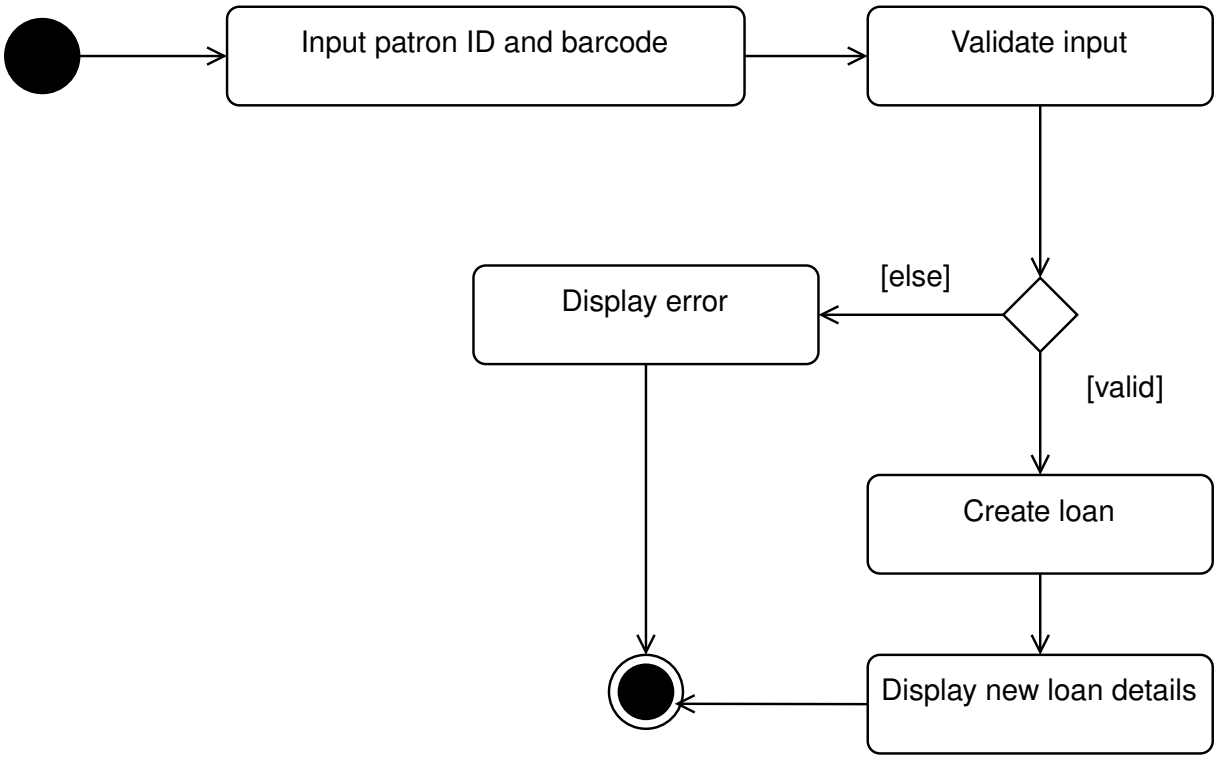


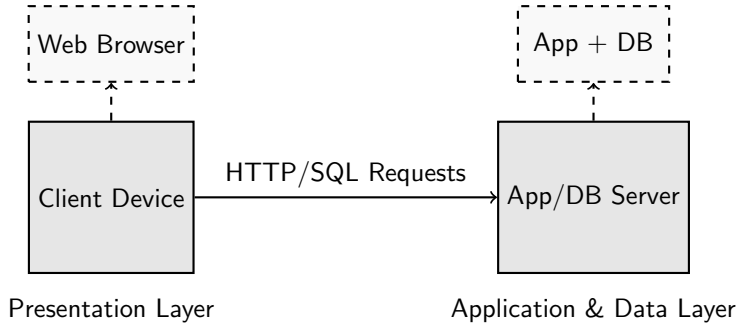
Figure 4.7: Activity diagram for checkout process

This activity diagram depicts the checkout workflow for materials. The process starts when the librarian inputs both the patron's ID and the material's ID. The system then validates this information and creates a loan record in the database if the details are correct. If the details are invalid, an error message is displayed.

4.5 | Deployment Diagram

In this section, we will display and describe two possible deployment architectures for the Library Management System: a 2-tier architecture and a 3-tier architecture (IBM, no date). Each architecture has its own benefits depending on scalability, complexity, and performance requirements.

4.5.1 | Two-Tier Architecture



The 2-tier architecture consists of:

- Client/Presentation Layer: Where the user interacts with the system.
- Database/Application Layer: A single server hosts both the business logic (application logic) and the database.

The communication happens directly between the client and the server, which contains both the logic and the database.

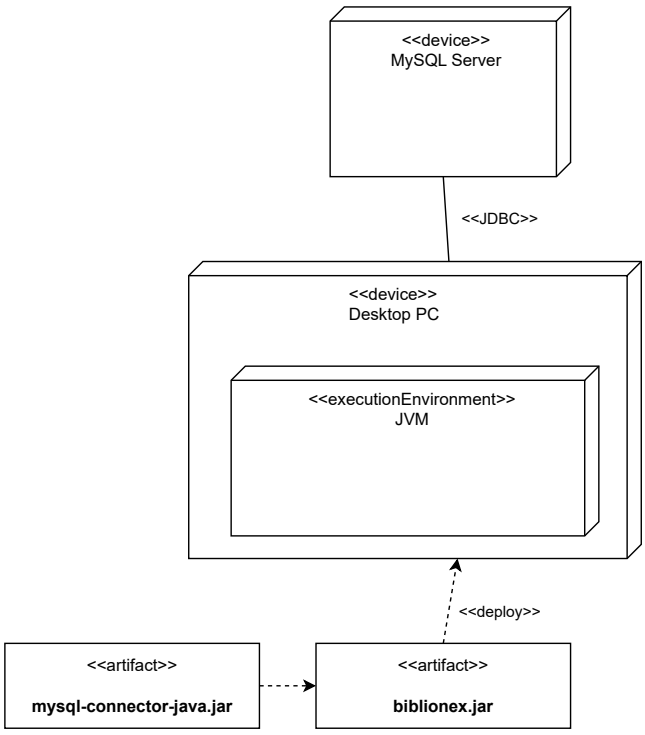
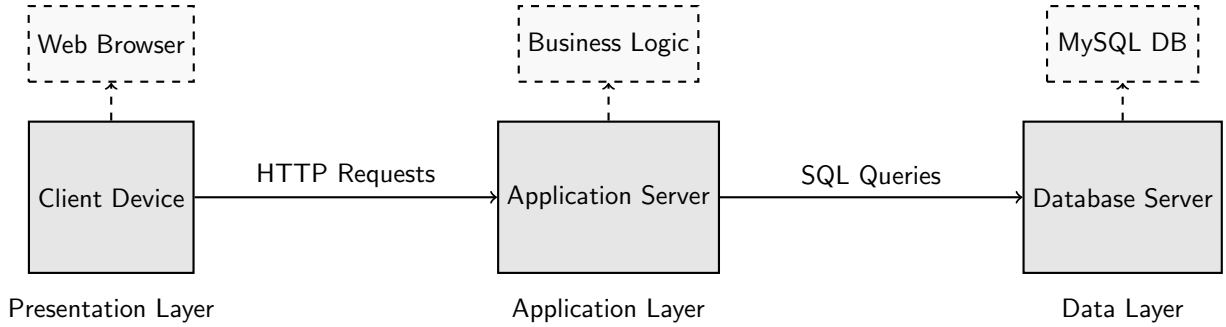


Figure 4.8: Deployment diagram for a 2-tier architecture

This deployment diagram illustrates the physical architecture of the LMS, showing how components are deployed across different hardware nodes in the production environment. The application runs on a computer with a Java Virtual Machine (JVM) to execute the JAR files, and it communicates with a separate MySQL Server for database operations.

4.5.2 | Three-Tier Architecture



In a 3-tier architecture, the system is divided into:

- Client/Presentation Layer: The user interface (UI).
- Application Layer: A separate server handles the business logic.
- Data Layer: A dedicated database server manages data storage and queries.

The communication flows from the client to the application server and then to the database server.

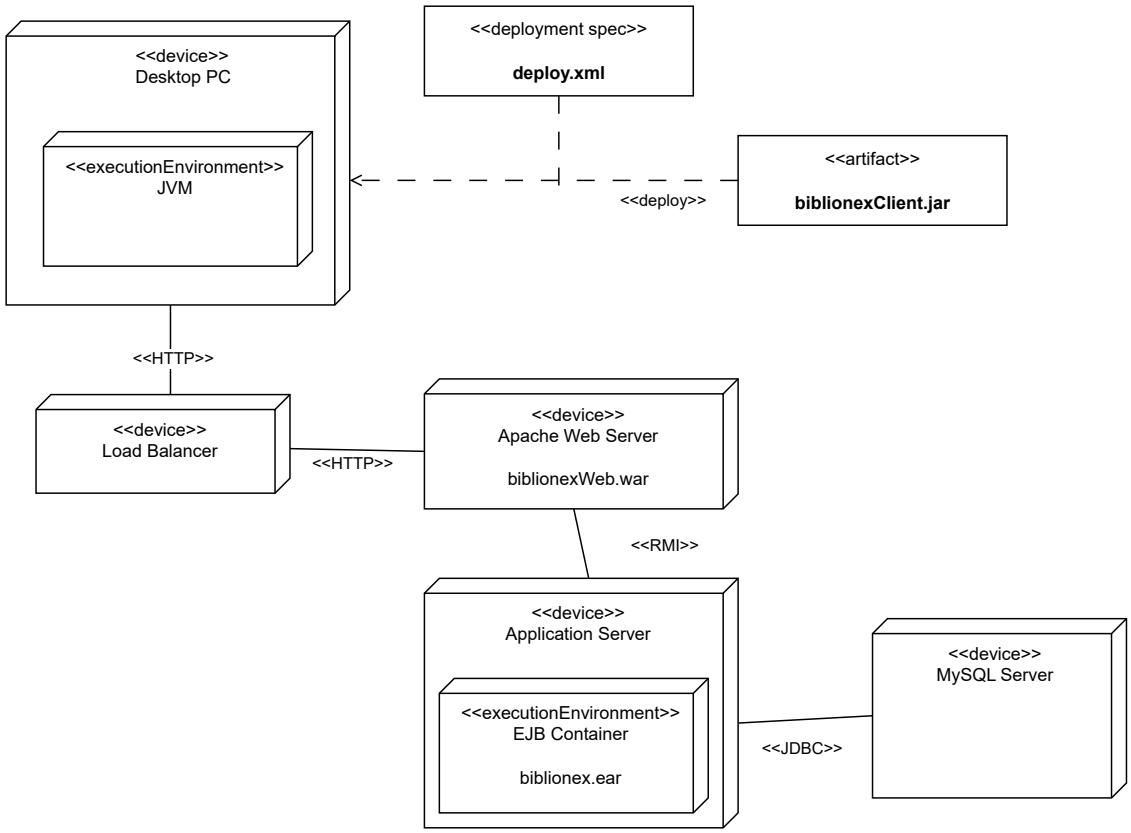


Figure 4.9: Deployment diagram for a 3-tier architecture

References

creme332 (2024). *biblionex*. Available at: https://github.com/creme332/biblionex [Accessed on 08/04/2024].

IBM (no date). *What is three-tier architecture?* Available at: https://www.ibm.com/topics/three-tier-architecture [Accessed on 09/10/2024].

OMG (2017). *OMG® Unified Modeling Language® (OMG UML®)*. Available at: https://www.omg.org/spec/UML/2.5.1/PDF [Accessed on 08/04/2024].