# Library Management System - Architecture Documentation

## 1. System Overview

### 1.1 Purpose

The Library Management System is a JavaFX-based application designed to manage library operations including book inventory, user management, and transaction processing.

### 1.2 System Architecture

The system follows a Model-View-Controller (MVC) architecture pattern with the following layers: - Presentation Layer (JavaFX) - Business Logic Layer (Services) - Data Access Layer (Models) - Database Layer (MySQL)

## 2. Architecture Design

### 2.1 High-Level Architecture

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ │ │ │ │ │  
│ Presentation │────▶│ Business Logic │────▶│ Data Access │  
│ (JavaFX) │ │ (Services) │ │ (Models) │  
│ │ │ │ │ │  
└─────────────────┘ └─────────────────┘ └────────┬────────┘  
 │  
 ▼  
 ┌─────────────────┐  
 │ │  
 │ Database │  
 │ (MySQL) │  
 │ │  
 └─────────────────┘

### 2.2 Key Components

1. **Models**
   * User
   * Book
   * Transaction
   * BaseModel
2. **Controllers**
   * LibrarianDashboardController
   * User Management Controllers
   * Book Management Controllers
3. **Services**
   * AuthenticationService
   * BookService
   * TransactionService
   * UserService
4. **Utilities**
   * DatabaseConnection
   * Logger
   * Validator

## 3. Component Details

### 3.1 Models

#### User Model

public class User extends BaseModel {  
 private String username;  
 private String password;  
 private String role;  
 private String email;  
 private String phone;  
 // Getters and Setters  
}

#### Book Model

public class Book extends BaseModel {  
 private String title;  
 private String author;  
 private String isbn;  
 private int quantity;  
 private String category;  
 // Getters and Setters  
}

#### Transaction Model

public class Transaction extends BaseModel {  
 private int userId;  
 private int bookId;  
 private Date issueDate;  
 private Date returnDate;  
 private String status;  
 // Getters and Setters  
}

### 3.2 Controllers

#### LibrarianDashboardController

* Manages the main librarian interface
* Handles book and user management
* Processes transactions
* Generates reports

### 3.3 Services

#### AuthenticationService

* Handles user authentication
* Manages session information
* Controls access permissions

#### BookService

* Manages book inventory
* Handles book check-in/check-out
* Processes book reservations

#### TransactionService

* Manages book transactions
* Tracks due dates
* Handles fines and penalties

## 4. Database Schema

### 4.1 Tables

#### users

CREATE TABLE users (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 username VARCHAR(50) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL,  
 role VARCHAR(20) NOT NULL,  
 email VARCHAR(100),  
 phone VARCHAR(20),  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP  
);

#### books

CREATE TABLE books (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 title VARCHAR(255) NOT NULL,  
 author VARCHAR(100) NOT NULL,  
 isbn VARCHAR(20) UNIQUE NOT NULL,  
 quantity INT NOT NULL,  
 category VARCHAR(50),  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP  
);

#### transactions

CREATE TABLE transactions (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT NOT NULL,  
 book\_id INT NOT NULL,  
 issue\_date DATE NOT NULL,  
 return\_date DATE,  
 status VARCHAR(20) NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  
 FOREIGN KEY (user\_id) REFERENCES users(id),  
 FOREIGN KEY (book\_id) REFERENCES books(id)  
);

## 5. Implementation Details

### 5.1 Dependencies

* JavaFX 17
* MySQL Connector/J
* Maven for dependency management

### 5.2 Configuration

* Database connection settings in application.properties
* Logging configuration
* Security settings

### 5.3 Build and Deployment

1. Clone the repository
2. Configure database settings
3. Run mvn clean install
4. Execute the generated JAR file

## 6. Security Considerations

### 6.1 Authentication

* Password hashing using BCrypt
* Session management
* Role-based access control

### 6.2 Data Protection

* Input validation
* SQL injection prevention
* XSS protection

## 7. Error Handling

### 7.1 Exception Handling

* Custom exception classes
* Global exception handler
* Logging mechanism

### 7.2 Validation

* Input validation
* Business rule validation
* Database constraint validation

## 8. Testing Strategy

### 8.1 Unit Testing

* JUnit tests for services
* Mock objects for dependencies
* Test coverage reporting

### 8.2 Integration Testing

* Database integration tests
* API endpoint tests
* End-to-end workflow tests

## 9. Future Enhancements

### 9.1 Planned Features

* Mobile application support
* Advanced reporting system
* Integration with external library systems

### 9.2 Performance Optimization

* Query optimization
* Caching implementation
* Load balancing

## 10. Detailed Component Specifications

### 10.1 User Management System

#### User Roles and Permissions

public enum UserRole {  
 ADMIN("Full system access"),  
 LIBRARIAN("Book and user management"),  
 MEMBER("Book borrowing and viewing");  
  
 private final String description;  
 // Constructor and getter  
}

#### User Authentication Flow

1. User enters credentials
2. System validates against database
3. Session token generated
4. Role-based permissions assigned
5. Session timeout after 30 minutes

### 10.2 Book Management System

#### Book Categories

public enum BookCategory {  
 FICTION("Fiction books"),  
 NON\_FICTION("Non-fiction books"),  
 REFERENCE("Reference materials"),  
 PERIODICAL("Magazines and journals"),  
 CHILDREN("Children's books");  
   
 private final String description;  
 // Constructor and getter  
}

#### Book Status Tracking

* Available
* Checked Out
* Reserved
* Lost
* Damaged

### 10.3 Transaction Management

#### Fine Calculation

public class FineCalculator {  
 private static final double DAILY\_FINE\_RATE = 0.50;  
 private static final int GRACE\_PERIOD\_DAYS = 7;  
   
 public double calculateFine(Date dueDate, Date returnDate) {  
 // Implementation details  
 }  
}

#### Transaction States

public enum TransactionStatus {  
 PENDING("Transaction initiated"),  
 COMPLETED("Successfully processed"),  
 OVERDUE("Past due date"),  
 CANCELLED("Transaction cancelled"),  
 LOST("Book reported lost");  
   
 private final String description;  
 // Constructor and getter  
}

## 11. API Documentation

### 11.1 REST Endpoints

#### User Management

POST /api/users/register  
GET /api/users/{id}  
PUT /api/users/{id}  
DELETE /api/users/{id}

#### Book Management

GET /api/books  
POST /api/books  
GET /api/books/{id}  
PUT /api/books/{id}  
DELETE /api/books/{id}

#### Transaction Management

POST /api/transactions/checkout  
POST /api/transactions/return  
GET /api/transactions/user/{userId}

### 11.2 Request/Response Formats

#### User Registration Request

{  
 "username": "string",  
 "password": "string",  
 "email": "string",  
 "phone": "string",  
 "role": "string"  
}

#### Book Checkout Request

{  
 "userId": "integer",  
 "bookId": "integer",  
 "issueDate": "date",  
 "returnDate": "date"  
}

## 12. Performance Optimization

### 12.1 Database Optimization

* Indexing strategy
* Query optimization
* Connection pooling
* Caching implementation

### 12.2 Application Optimization

* Lazy loading
* Batch processing
* Asynchronous operations
* Memory management

## 13. Monitoring and Logging

### 13.1 Logging Strategy

public class SystemLogger {  
 private static final Logger logger = Logger.getLogger(SystemLogger.class);  
   
 public void logTransaction(String message) {  
 logger.info("Transaction: " + message);  
 }  
   
 public void logError(String message, Exception e) {  
 logger.error("Error: " + message, e);  
 }  
}

### 13.2 Monitoring Metrics

* Response times
* Error rates
* Database performance
* User activity

## 14. Deployment Architecture

### 14.1 System Requirements

* Java 17 or higher
* MySQL 8.0 or higher
* 4GB RAM minimum
* 10GB disk space

### 14.2 Deployment Steps

1. Database setup
2. Application server configuration
3. Load balancer setup
4. Monitoring tools installation
5. Backup configuration

## 15. Maintenance Procedures

### 15.1 Regular Maintenance

* Database backup
* Log rotation
* Performance monitoring
* Security updates

### 15.2 Emergency Procedures

* System recovery
* Data restoration
* Incident response
* Communication plan