Project Documentation: Library Inventory Manager

Project Name: Library Inventory Manager

Author: Dominik Hoch

Contact: domik.hoch@gmail.com

Date: 2.1.2026

School: SPŠE Ječná

Project Type: School Project (Subject: Programming/PV)

---

1. Requirements Specification

The goal of this project was to create a database-driven application for managing a library inventory. The system allows librarians to manage books, authors, categories, and loans through a web interface.

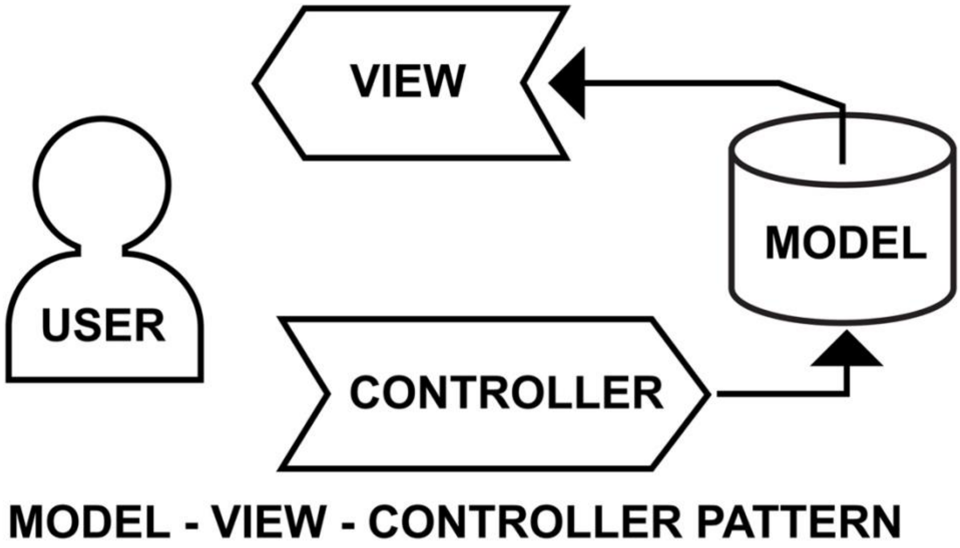
1.1 Functional Requirements

- **Book Management:** The system must allow creating, reading, updating, and deleting (CRUD) book records.
- **Complex Relations:** The system must handle **M:N relationships** between books and authors (one book can have multiple authors).
- **Transactions:** Critical operations (saving a book with authors, creating a loan) must be wrapped in database transactions to ensure data integrity.
- **Loans:** The system must allow assigning a book to a borrower and tracking its return status.
- **Reporting:** The system must generate a summary report aggregating data from at least three tables (using SQL Views).
- **Import:** The system must allow bulk import of book data via JSON files.

---

2. Application Architecture

The application follows the **MVC (Model-View-Controller)** architectural pattern and implements the **DAO (Data Access Object)** design pattern to separate business logic from database access.



2.1 Structural Description

- **Controller Layer (`app.py`):** Handles incoming HTTP requests, processes input data, calls the DAO layer, and renders HTML templates.
- **Data Access Layer (`dao/`):** Contains classes (`BookDao`, `ReportDao`) that encapsulate all SQL queries. This ensures that the rest of the application is not dependent on the specific database implementation.
- **Service Layer (`services/`):** Handles complex business logic not directly related to CRUD, such as parsing and validating JSON imports.
- **Presentation Layer (`templates/`):** HTML5 templates.

3. Application Behavior

The application behavior is event-driven based on user interaction via the web interface.

3.1 Use Case Example: Borrowing a Book

1. **User Action:** User clicks "Borrow" on a book in the main list.
2. **Controller:** The route `/borrow/<id>` is triggered.
3. **View:** A form is displayed asking for the borrower's name.
4. **Submission:** User submits the form.
5. **DAO Layer:** The `create_loan` method is called.
   - *Transaction Start.*
   - Insert record into `loans` table.
   - Update `books` table (set `is_borrowed` flag to 1).
   - *Commit Transaction.*
6. **Feedback:** The user is redirected to the dashboard with a success message.

---

4. Interfaces and Dependencies

The application relies on the following third-party libraries and interfaces:

- **Python 3.9+:** The core runtime environment.
- **Flask (2.x):** The web framework used for routing and template rendering.
- **MySQL Connector/Python:** The standard driver for connecting to MySQL databases.
- **Bootstrap 5:** CSS framework used via CDN for the user interface.

System Dependencies:

- **MySQL Server (8.0+):** Must be running and accessible via TCP/IP.

---

5. Legal and Licensing

- **License:** MIT License. The code is free to use for educational and personal purposes.
- **Copyright:** The source code is original work created by the author listed above for educational purposes.
- **Third-Party Rights:** Flask and MySQL Connector are used under their respective open-source licenses (BSD, GPL/FOSS).

---

6. Configuration

Configuration is strictly separated from the code and stored in a JSON file.

File: `config/db_config.json` **Format:**

JSON
```
{
  "host": "localhost",
  "user": "root",
  "password": "YOUR_PASSWORD",
  "database": "library_db"
}
```

- **host:** Database server address (usually localhost).
- **user/password:** Credentials for MySQL access.
- **database:** Name of the database schema (default: `library_db`).

---

7. Installation and Launch

For detailed instructions, please refer to the `README.md` file included in the project repository.

Brief Summary:

1. Install Python dependencies: `pip install -r requirements.txt`.
2. Import database structure: `mysql -u root -p < sql/install.sql`.
3. Configure credentials in `config/db_config.json`.
4. Run the application: `python app.py`.

---

8. Error Handling

The application handles common error states to prevent crashes:

- **Database Connection Failure:** If the DB is unreachable, the application catches `mysql.connector.Error` and displays a user-friendly error message instead of a stack trace.
- **Invalid Input:** If a user enters text into a numeric field (e.g., Price), the `ValueError` is caught, and a flash warning is shown.
- **Transaction Failure:** If an error occurs during a multi-step database operation, `conn.rollback()` is executed to return the database to a consistent state.
- **Import Errors:** Malformed JSON files trigger a validation error displayed to the user.

---

9. Testing and Validation

The application was validated using manual testing scenarios (Black Box Testing).

9.1 Test Results

- **CRUD Operations:** Validated. Books can be added, edited, and deleted. Changes persist in the database.
- **Transactions:** Validated. If adding an author fails during book creation, the book is not created (Atomicity confirmed).
- **M:N Relationships:** Validated. Multiple authors are correctly linked to a single book.
- **Import:** Validated. JSON files with correct structure are imported; invalid files are rejected.

10. Versions and Known Issues

- **Current Version:** 1.0.0
- **Known Issues:**
  - The application does not currently provide a web interface for creating new Authors or Categories. These must be pre-populated in the database or inserted via SQL.
  - No user authentication (login) is implemented; the system is open to all users on the local network.

11. Database Model (E-R Diagram)

The application utilizes a relational database (MySQL) with the following schema:

Tables:

1. **categories:** (`id`, `name`, `is_active`) - Categorization of books.
2. **authors:** (`id`, `name`) - List of authors.
3. **books:** (`id`, `title`, `price`, `condition`, `is_borrowed`, `category_id`) - Main entity.
4. **book_authors:** (`book_id`, `author_id`) - Junction table for M:N relationship.
5. **loans:** (`id`, `book_id`, `borrower_name`, `loan_date`, `returned_date`) - Transactional table for loans.

Views:

- `view_book_details`: Simplifies querying books with joined authors and categories.
- `view_library_stats`: Aggregates data for the reporting module.

12. Network and Hardware Configuration

- **Network:** The application runs on the `localhost` loopback interface (127.0.0.1) on port **5000**.
- **Database:** Connects to MySQL on the standard port **3306**.

- **Hardware Requirements:** Any standard PC capable of running Python 3 and MySQL Server. No special hardware required.

---

13. Import/Export Schema

The application supports importing data via **JSON**.

Import Rules:

- File type: `.json`
- Structure: List of objects `[{...}, {...}]`
- **Mandatory Fields:** `title` (String), `price` (Number).
- **Optional Fields:** `authors` (Array of Integers - IDs), `category_id` (Integer).

Example Import File:

JSON

```
[
  {
    "title": "The Great Gatsby",
    "price": 15.99,
    "category_id": 3,
    "authors": [1]
  }
]
```