Project Report : Library System Management

Group Members

N°	NOM ET PRENOM	MATRICULES	FUNCTIONS
1	PETNOU HONLUE FREDERIC ARMEL	22G00362	Project manager, developer, desiner, QA…
2	DJIFACK ASSONGTIA DARIL VIANY	22G00438	Developer , QA, Analyst
3	KAMDEU YOUMBISSIE CHRETIEN ROSTINI	22G00173	Designer , Developer, QA,Analyst
4	AGASSEM SANDA FLORENTIN	24G1072	Analyst
5	MAMOUDOU BAOURO	24G01105	Analyst
6	FOKAM DE LOKA WILFRIED	22G00138	Designer
7	KAJE BOUKEM WARREN CAPONNE	24G01094	
8	JEUDIEU TSAGUE AUDREY DANIELLE	22G00167	
9	KENFACK RONYL VESIAN	22G00186	
10	NGAMBI DIKOUME ALMA DANIELA	22G00301	

I- Background and Objective

The goal of this project is to develop a Java application to manage the day-to-day operations of a small library. The application will allow you to register books, manage borrowing and returns, and view the book inventory.

II- Technical Requirements

- **Language :** Java (version 8 or higher).
- **Methodology :** Approach object- oriented .
- **Tools :** A IDE like Netbeans , Eclipse Or IntelliJ IDEA.
- **Documentation:** Each group must provide a report explaining the architecture, technical choices, and a user manual.
- **Versioning (Optional) :** To use Git For THE follow up of the modifications And there collaboration.

III- Presentation General

- **Application Name:** Library Manager
- **Technologies:** java, javaFX , SQLite, JDBC
- **Architecture:** MVC (Model- view- controller)

IV- Design phase

1. Needs analysis

Management of Books

- **Addition of a book :** To input THE information of base (title, author, ISBN, year of publication).
- **Deletion and modification:** Allow updating or deleting an existing book.
- **Consultation of the inventory :** Display there list complete of the books available.

Management of Loans

- **Loan of a book :** Save the loan by A user (name, borrowing date).
  - **Back of book :** Save the return and put has day the status of the book.

- **Historical :** To consult the history of the loans For each book or user.

Interface User Console

- A menu main allowing of navigate between THE features (management books, loan management, consultation).
- Management of the entries user with of the validations of seizure.
  - Data Persistence (Optional for beginners)
    To use of the files text Or A format simple (by example, CSV) For to safeguard And restore THE data between several executions of the application.

2. MVC Architecture Design

```
LibraryManager/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com.example.librarymanager/
│   │   │       ├── App.java (classe principale)
│   │   │       ├── Controllers/
│   │   │       ├── Database/
│   │   │       │   ├── BooksTable.java
│   │   │       │   ├── CategoryTable.java
│   │   │       │   ├── LoanTable.java
│   │   │       │   ├── Repository.java
│   │   │       │   └── UserTable.java
│   │   │       ├── Models/
│   │   │       ├── utils/
│   │   │       ├── views/
│   │   │       │   └── ViewFactory.java
│   │   │       └── module-info.java
│   │   └── resources/
│   │       ├── FXML/ (fichiers FXML pour les vues)
│   │       ├── language/ (fichiers de localisation)
│   │       ├── style/ (fichiers CSS)
│   │       ├── DB_Schema.sql
│   │       └── library.db (base de données SQLite)
├── target/
├── pom.xml (configuration Maven)
├── mvnw
├── mvnw.cmd
├── README.md
└── .gitignore
```

- Controllers /: management of user interactions
- Models/: entity profession (Books, User, Loan, Category, etc.)
- Views /: Fxml interfaces
- Databases /: CRUD operations and table management
- Utils / : Utility Classes (eg: Currentuser for session management)

3. Database modeling

The database was initially modeled from the classes in the class diagram but was later updated to meet development needs .

a- table books

- book_id : unique identifier (primary key, auto-incrementing)
- title : title of the book (required)
- author : author (required)
- isbn : isbn number (unique)
- description: book description
- category_id : reference to the category (foreign key)
- published_year : year of publication
- copies_total : total number of copies (>0)
- copies_available : number of copies available (≥0)
- created_at : creation date
- image_path : image path

b- table category (categories

- category_id : unique identifier (primary key, auto-incrementing)
- name : category name

c- table loans

- loan_id : unique identifier (primary key, auto-incrementing)
- book_id : reference to the book (foreign key)
- user_id : reference to the user (foreign key)
- borrowed_at : borrowing date
- due_at : expected return date
- returned_at : effective return date
- status : loan status (' ongoing ', ' returned ', ' overdue ')
- number_of_book : number of books borrowed

d- users table

- user_id : unique identifier (primary key, auto-incrementing)
- username : username (unique)
- password_hash : hashed password
- role : role ('admin' or ' member ')
- created_at : creation date
- phone: phone number
- birthdate : date of birth
- gender : gender ('male' or ' female ')
- address : address
- email: email (unique)

V- Implementation phase

4. Development of data models

The data models were extracted from the UML class diagram, so we have the following models :

- Books.java
- Attributes: book_id , title, author, isbn , category_id , published_year , copies_total , copies_available , image_path , description

- User.java
- Attributes: user_id , fullName , email, password, role, phone, birthdate, address, gender

- Loan.java
- Attributes: loan_id , book_id , user_id , borrowed_at , due_at , returned_at , status, book_name , user_name , numberOfBook

- Category.java
- Attributes: category_id , name

- Notification.java
- Attributes : message, date

- Model.java
  - Singleton class managing ViewFactory

- History.java
- Planned for future implementation

5. Creation of user interfaces

The interfaces were created using scenebuilder which is software that allows you to create fxml views for free so we have the following fxml files :

Authentication :
- Login.fxml : Login interface
- Registration.fxml : New user registration

Main Interfaces:
- Dashboard.fxml : Main application dashboard
- Sidebar.fxml : Navigation sidebar component
- Dash.fxml : Dashboard content

Resource Management:
- Books.fxml : Book management interface
- BookCard.fxml : Individual book display component
- BookDetails.fxml : Detailed book information
- EditBook.fxml : Book editing interface
- Users.fxml : User management interface
- Borrows.fxml : Loan management interface
- BorrowsCard.fxml : Individual loan display component
- ConfirmBorrow.fxml : Loan confirmation interface

Support:
- Help.fxml : Help and contact information

6. Controller Implementations

For each fxml view above, a controller of the same name is associated to manage the communication between the view and the database .

7. Database integration

- Repository<T> interface defining standard CRUD operations
- Dedicated classes per entity (BooksTable , UserTable , etc.)
- Queries prepared for security
- Transactions to maintain data integrity

VI- Difficulties Meetings

8. Techniques

- Conflicts during pushes:
  Concurrent modifications to the same files (eg: DatabaseUtils.java, controllers) generated frequent conflicts.

Solutions:
- Adoption of a feature branch strategy
- Code review via pull requests
- Use git rebase for a clean linear history
- Distribution of tasks:

The initial distribution of work lacked clarity.
Solutions:
- A numbered task list was sent at the beginning of each sprint; each member chose a number
- Communication via WhatsApp for tasks and coordination
- Daily or one-off meetings on Google Meet or in dedicated WhatsApp groups

Implementation of the MVC Pattern
- Confusion of responsibilities:

Business logic sometimes included in controllers
Solutions:
- Refactoring to clarify the role of classes (model vs controller)
- Centralization of global information in CurrentUser
- Use of clear structures to separate data access and business logic
- Dependencies between controllers:

Sharing data between views (e.g. login status, item selection) Solutions:

- Dependencies managed via static methods or access to static attributes
- CurrentUser played the role of global session

9. Functional

Communication between Interface Components

- Unsynchronized updates:

Changes in one view were not reflected elsewhere.

Solutions:

- Using the CurrentUser utility class to store the active user
- Static methods and attributes in controllers to share state or access views
- Explicit refreshes triggered as needed in the relevant views

VII- Prospects for improvement

- Automated tests: integration of unit tests (JUnit) and integration (TestFX)
- Improved Git workflow: adoption of GitFlow to standardize the development cycle
- Monitoring and logs: adding activity logs with Log4j to diagnose anomalies
- New features planned:
- Book reservations
- Email notifications
- Mobile interface
- Data export
- Management of fines

VIII- Conclusion

The Library Manager project has enabled the construction of a robust, scalable, and user-oriented application. The challenges encountered were overcome through rigorous organization, effective use of MVC architecture, and simple but effective collaborative tools (Git, WhatsApp, Google Meet). The system remains extensible and provides a solid foundation for future team development.