Group Two Project Documentation

PROJECT NAME: BLIND DATE WITH A BOOK

Group members: Debbie Richford, Allize Renae, Rachel Fuller, Janine O'Connor, Maisie Pepper, Kaveyah Ravikumar, Nemi Imoh

INTRODUCTION

Aims and Objectives

As a group of keen readers, we discussed how we could utalise technology to help a book club organiser decide what book to read next and manage their choices. We came up with 'Blind Date with a Book' - a command-line app that helps coordinators choose books by genre and manage reading timelines. It solves a real-world problem of what to read next, as well as how to keep track of the book club's reading deadlines and reading history.

The app uses the **Open Library API** to generate a book choice, stores the club's reading history in an SQL database, and automatically sets a reading deadline.

Roadmap of Report

This report outlines the project background, system specifications and design, implementation and testing strategy, and concludes with an evaluation and potential improvements.

BACKGROUND

The idea for our project was inspired by a fundraiser called *Blind Date with a Book*, organised by one of our group members for a literacy charity, where books were wrapped, sorted into genres, and sold as surprise reads.

From here, we all voted as a team and decided we wanted to build a Book Club App designed to streamline random weekly book selections, track deadlines, and maintain a record of past choices.

We chose the Open Library API because it provides free access to a large catalogue of books, is well-documented, and was straightforward to integrate.

How the App Runs

When the app is launched, the user is greeted with a welcome message and introduced to the available options.

The user is presented with three main menu options: A. Choose New Book / B. View Current Book and Deadline. / C. View Book History.

If the user chooses *Choose New Book*, they are presented with four possible genre options (Horror, Comedy, Romance or Random). The app fetches a book from the Open Library API based on the chosen genre, or it fetches a random book if that option is selected. Key book details, including title, author, and genre, are then displayed. The book data is then stored in the SQL database along with an assigned date and a reading deadline.

If the user selects *View Current Book and Deadline*, the app displays the book currently selected by the club, including its title, author, genre, and the deadline for completion.

If the user selects *View Book History,* the app shows a list of all previously selected books along with their assigned dates, deadlines, and genres.

This app simplifies book club management by automating book selection, tracking reading deadlines, and maintaining a historical archive of past choices.

API: https://openlibrary.org/developers/api

SPECIFICATIONS AND DESIGN

Functional Requirements

These are the core features the app is designed to deliver:

1. User Input and Validation:
    - Provide an in-terminal app that allows users to get a new book, view the current book, view the reading deadline, or view book history.
    - Validate user inputs to prevent errors from invalid choices.

2. Selection of a New Book:
    - Allow users to pick one of the available genres or select a random book.
    - Fetch book details (title, author, genre) from the Open Library API.
    - Assign a date and reading deadline to the chosen book.

3. Book Tracking
   - When a new book is returned to the user, the book details (title, author, genre, assigned date, deadline) are automatically saved in an SQL database.
   - Users can view the current book and reading deadline, or view book history.

4. API Integration
   - Use Flask to handle backend logic and manage communication between the app, API, and database.
   - In addition to retrieving data from the Open Library API, the app creates a custom-built API with endpoints that will communicate with the SQL database.

Non-Functional Requirements

- Performance: API calls and database queries to return quick responses.
- Reliability: accurate data is stored in and retrieved from the SQL database.
- Usability: The interface is clear and simple for users.
- Error Handling: The system handles invalid inputs, API failures, and database errors.
- Scalability: The modular design and OOP principles make the app extendable.

DESIGN AND ARCHITECTURE

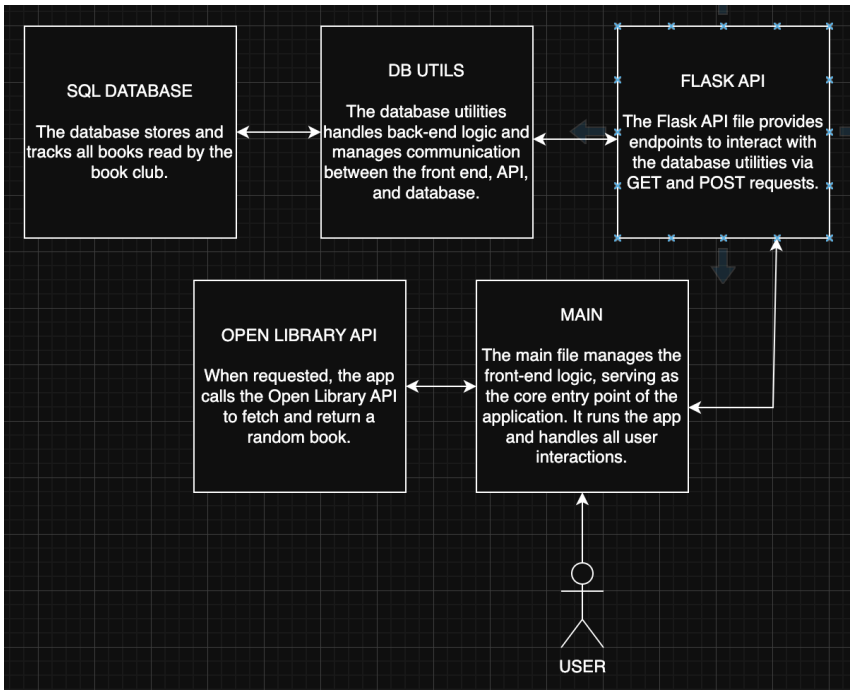
Code was organised into multiple files to allow for better maintenance and testing, and to make the system more scalable. Using a modular approach with key OOP principles made the code cleaner, more efficient, and easier to collaborate on.

Key files:

- SQL files:
  - **database_setup.sql** - establishes the LibraryDB database and book_tracker table
- Python files:
  - **main.py -** runs the app and interacts with the user.
  - **UI.py** – user-facing code; contains *UIClass* and *UIDatabaseClass*.
  - **flask_api.py -** Flask endpoints, connects to the SQL database
  - **book.py -** contains the *Book* class.
  - **book_api.py -** interacts with Open Library API.
  - **db_utils.py -** interface layer for database operations.
  - **deadline.py -** date functionality.
  - **test.py** - unit tests to ensure the reliability of the app.
  - **config.py** - stores database credentials.

- Other files:
    1. **requirements.txt** - lists third-party libraries needed for the project to run.
    2. **README.md -** provides details of the project and how to run the app.
    3. **.gitignore** - tells Git which files and folders to ignore.

Architecture Diagram



(*System architecture showing the flow between the user, Flask API, Open Library API, and SQL database.)*

IMPLEMENTATION AND EXECUTION

Development Approach and Team Member Roles

We created a list of all the tasks required to bring the system to life and allocated responsibilities based on individual strengths and areas of expertise.

Allize	Main.py - app execution, Book.py (Book class), GitHub Coordination, Project Manager, Project Outline & Documentation, Presentation Slides
Debbie	UI.py (UI class), Project Outline & Documentation
Rachel	UI.py (UI class), Project Outline & Documentation, Unit testing
Maisie	API integration from Open Library (book_api.py), API integration with Flask and SQL endpoints (flask_api.py), Unit testing, Documentation, README.md
Kav	SQL schema
Nemi	db_utils.py

Allize	Main.py - app execution, Book.py (Book class), GitHub Coordination, Project Manager, Project Outline & Documentation, Presentation Slides
Debbie	UI.py (UI class), Project Outline & Documentation
Rachel	UI.py (UI class), Project Outline & Documentation, Unit testing
Maisie	API integration from Open Library (book_api.py), API integration with Flask and SQL endpoints (flask_api.py), Unit testing, Documentation, README.md
Kav	SQL schema
Janine	deadline.py
Everyone	Requirements.txt, Testing – Manual Testing, Unit Testing, UAT Testing and Regression Testing Code Reviews and Debugging Project Documentation Presentation Slides

Tools and Libraries

- **Database**: Implemented using SQL.
- **Python modules**: Used DateTime, Random, JSON, Requests, Flask, MySQL Connector, and Itertools.
- **API integration**: Extended functionality with the **Open Library API**.
- **Testing**: Combined unit tests, manual checks, and user acceptance testing.
- **Style**: OOP for complex files, procedural for simpler routines, exception handling for robustness.

Agile Development

We incorporated agile elements throughout development:

- Iterative approach - scaling back scope to ensure delivery of a core working system.
- Mob programming - used to resolve UI-function integration issues.
- Code reviews - performed on pull requests in GitHub.
- Continuous integration - feature branches were merged into *main* regularly.

Implementation Process

Collaboration shaped every stage of our project. At the start, everyone contributed ideas, and after open discussion, we voted on the direction to take, which led to *Blind Date With A Book*. Regular catch-ups helped us stay aligned, and mob programming sessions supported debugging and strengthened teamwork.

Challenges included balancing workload fairly, scheduling across different commitments, and handling Git merge conflicts. Once we adopted feature branches and strict pull request reviews, our workflow improved.

We also scaled back our original, overly ambitious design to focus on a smaller but functional version. This adjustment allowed us to meet the deadline with a working core system. Features we postponed were documented as future enhancements.

TESTING AND EVALUATION

Our testing strategy combined automated unit tests, manual functional tests, user acceptance testing, and regression testing. Each module was tested individually before being integrated and tested as part of the main system.

Tests Performed

- Connection to the database.
- Successful API call to the Open Library API.
- Randomly return one book from the Open Library API.
- Adding a new book to the database.
- Viewing the current book and the current deadline.
- Viewing previously read books.
- OOP classes that generate objects with correct properties/methods.
- User interaction, flow, and input validation.

System Limitations

Our testing showed that the API occasionally returned books that were not in English, despite the programme specifying English-language books only. We discovered this is due to some books being incorrectly labelled in the Open Library database. This issue is something we would look to rectify in future versions of the app.

There is currently a chance that a book would be suggested to the user that they have already read. Future versions could include logic to check the database first, and only return titles not already in the database.

Evaluation

Testing confirmed that the app met all functional and non-functional requirements. The app was responsive, reliable and user-friendly, with accurate error handling. Limitations were identified but could be addressed in future versions.

CONCLUSION

Overview

As a team, we successfully built a working in-terminal app that delivers our vision for the project. The app fulfills all of the identified functional requirements (user input, book selection, tracking and API integration).

It also successfully fulfills the non-functional criteria we identified (performance, reliability, usability, error handling and scalability). For example, the user welcome menu asks the user to choose from A, B, C. If they type a different letter, or no letter at all, they are asked to choose again. As another example, we wanted the call to the Open Library API to return accurate data and not take too long for the user, therefor if the call takes too long it returns an error message to the user.

As a team, our biggest challenge was coordinating roles and schedules, and finding suitable times we could all meet together to discuss the project, but the experience strengthened our teamwork and problem-solving.

Future Improvements

Future enhancements could include:

- Expanding the choice of genres the user can choose from (currently set at 3 or a random book).
- Allowing custom deadlines (currently set to one week).
- Registering book club members, including an additional table in the SQL file to record member details.
- Option to have members sent details of the chosen book and reminders as the deadline gets closer.
- Member voting on genres.
- Book ratings and reviews.
- Option to get audiobooks, not just print books.
- Preventing duplicate book selections and giving book language choices.

Thank you for reading! We hope you enjoy using the Blind Date with A Book app!