BookWorm Final Report

Members:

Yashik Dhanaraj, Aditi Jorapur, Sharanya Udupa

****Sharanya's commits aren't showing up but she committed****

- **She got a new laptop and did not set up her github username, but since the repo was public, she committed anonymously (name will show up but not username). You are able to view this using commit history**

Usernames:

dyashik, aditijorapur, sharanya2003 + Anonymous

Project Introduction

BookWorm is a library management system crafted for librarians allowing a seamless and organized way to execute library operations. Librarians get the opportunity to add, edit, and delete books within the system, ensuring an up-to-date collection of the catalog. They also have the ability to manage user accounts including tracking the borrowing and returning of books.

Objective

The key objectives for this library management web application are:

<u>Catalog Management</u>

The system will allow librarians to add, edit and delete book records including details like title and author. It will support searching and filtering of books in the catalog by various parameters like title, author, and publication year The system will also track the number of copies available for each book.

<u>User Management</u>

The application will enable adding, updating and deleting user accounts. Details like name, email, phone number, address that can be stored for each user account.

<u>Circulation Management</u>

Users will be able to check book availability in the system catalog and reserve books. The application will facilitate users borrowing books by allowing them to check-out items. It will track critical circulation details like checkout dates and due dates for loans.

<u>Database</u>

On the backend, SQLAlchemy will be leveraged with a MySQL database, defining clear entities and relationships between them like Books, Users, Loans etc. Data validation, constraints, indexes and optimizations will be implemented to ensure robustness and performance.

UI/UX

The web interface for both librarians and general users will be implemented using HTML/CSS focused on maximizing ease of use. Users will be able to intuitively search, browse, reserve and check-out items. Custom interfaces will cater specifically to librarian workflows vs regular user needs.

The completed application will deliver a full-featured, modern library management system built on Python, SQL and JavaScript.

Project High-Level Design

The library management system employs a client-server model with a Flask frontend and Python backend. The key high-level components are:

Architecture

The library management system will leverage a modular microservices architecture consisting of independent web, api and database services which can scale independently. All services will be containerized using Docker for portability across environments.

Frontend Web Application

The client-facing frontend will be developed as a multi page application based on HTML, which provides improved responsiveness and minimizes page reloads. State management will be handled centrally using the React Context API making state changes predictable. The UI will incorporate the Bootstrap component library for layouts, navigation and data visualization yielding consistency across screen sizes. Core UI functionalities like searching, reserving and managing loans will be developed as reusable components accelerating development.

Backend using Python

Backend was written using Python, which contained all the core application logic, database models and interactions, route handlers, and integration points with the front end. Specifically, the Python backend code included:

1. Database models defined using SQLAlchemy that structured the data and relationships between tables. Models created included Book, Author, User, Loan, and Address.
2. CRUD routes built with Flask that handled creation, reading, updating, and deletion of records for each database model. These routes accessed and manipulated the database based

We tested the backend aspect of the application using SQLite3 Editor extension, which allowed us to view if the database was updated correctly, or if it required any further testing using subqueries.

Key Libraries

Flask: The Flask Python web framework provides the core backend server framework. Flask allows defining REST API endpoints and routes easily to expose backend functionality for front-end consumption along with integrated support for plugins.

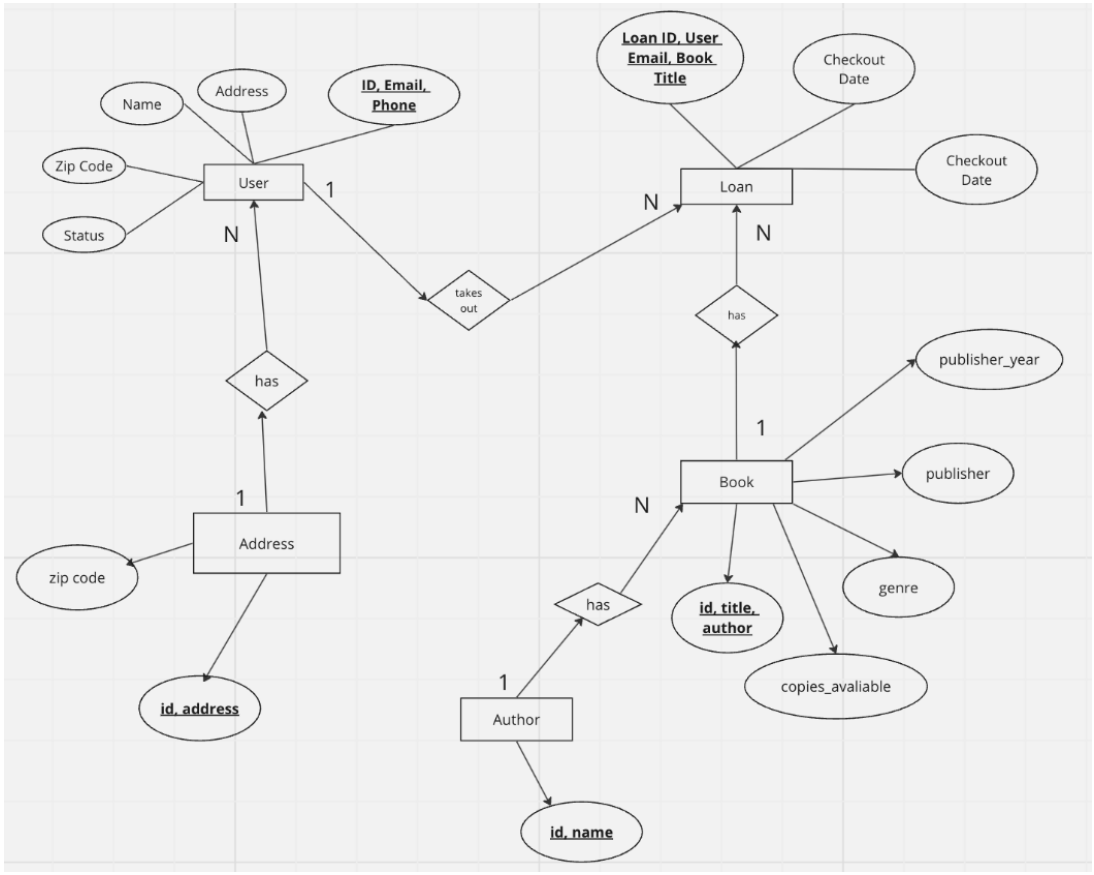
SQLAlchemy: SQLAlchemy is the ORM library that enables querying, updating and managing the MySQL database using Python without needing to write raw SQL. SQLAlchemy provides

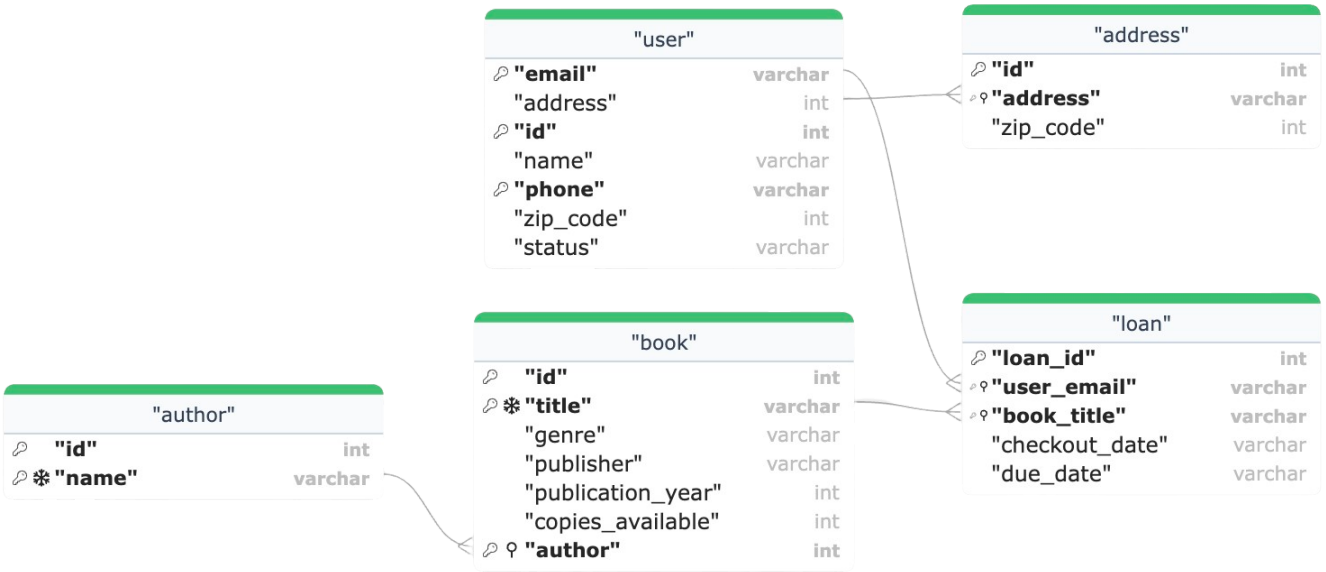
database abstraction through Python model classes mapped to database entities to retain optimizations.

HTML: Hypertext Markup Language will provide the basic building blocks and content structure for the user interface. HTML elements like headers, paragraphs, forms, tables and more will encapsulate information to be rendered on the frontend. HTML pages will be generated by the Flask backend by injecting model data from the SQLAlchemy into predefined templates.

Database Design

ER Diagram





Relationships

Author to Book (One-to-Many): The author table's author column links to the author column in the book table, establishing a one-to-many relationship between an author and their books

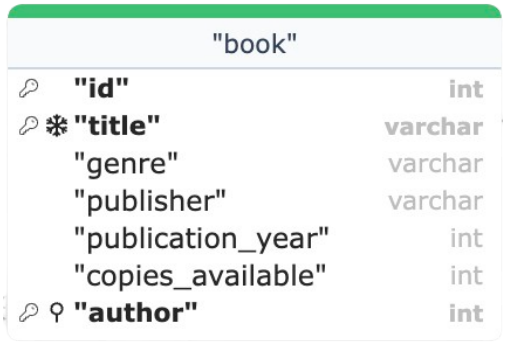
User to Loan (One-to-Many): The user_email column in the loan table references the email column in the user table, indicating that one user can have multiple loans

Loan to Book (Many-to-One): The title column in the book table is referenced by the book_title column in the loan table, indicating that many loans can be linked to a single book

Address to User (One-to-Many): An address can be linked to multiple users. For instance, a shared household where different family members or occupants share the same address.

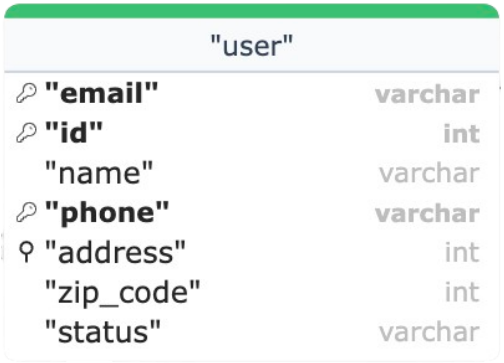
Normalization of Tables

Book Table



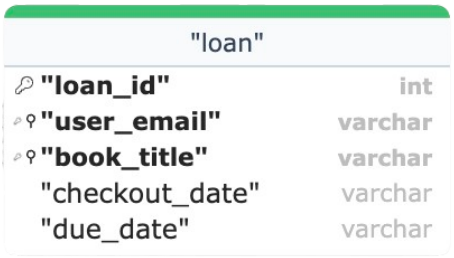
The book table separates information about books into distinct entities to minimize the redundancy and dependency of non-key attributes. 1NF is already assumed because there are no repeating values or groups of values in a column. There is no partial dependency so the Book Table is in 2NF. There is no transitive dependency in the Book Table so the table is in 3NF.

User Table

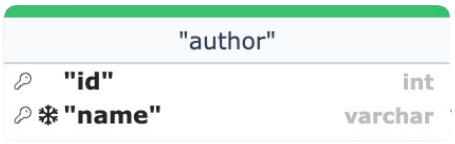


The user table separates information about users into distinct entities to minimize the redundancy and dependency of non-key attributes. 1NF is already assumed because there are no repeating values or groups of values in a column. There is no partial dependency so the User Table is in 2NF. There is no transitive dependency in the User Table so the table is in 3NF.

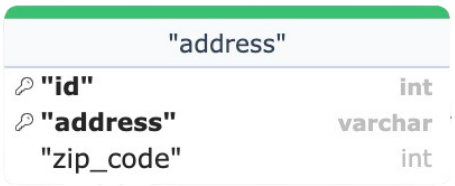
Loans Table



The loans table has the ability to check the same book out multiple times which makes every column dependent on the last. 1NF is the basic assumption and because of these dependencies, the loans table is in 1NF.



The "author" table uses Third Normal Form (3NF) in its design. It avoids data redundancy and dependency issues. Each attribute within the table contains atomic values, ensuring no repeating groups exist. The primary key ("id", “name”) uniquely identifies each author, and attributes like "name" depend solely on this key, preventing partial dependencies. Importantly, there are no transitive dependencies; all non-prime attributes relate directly to the primary key.

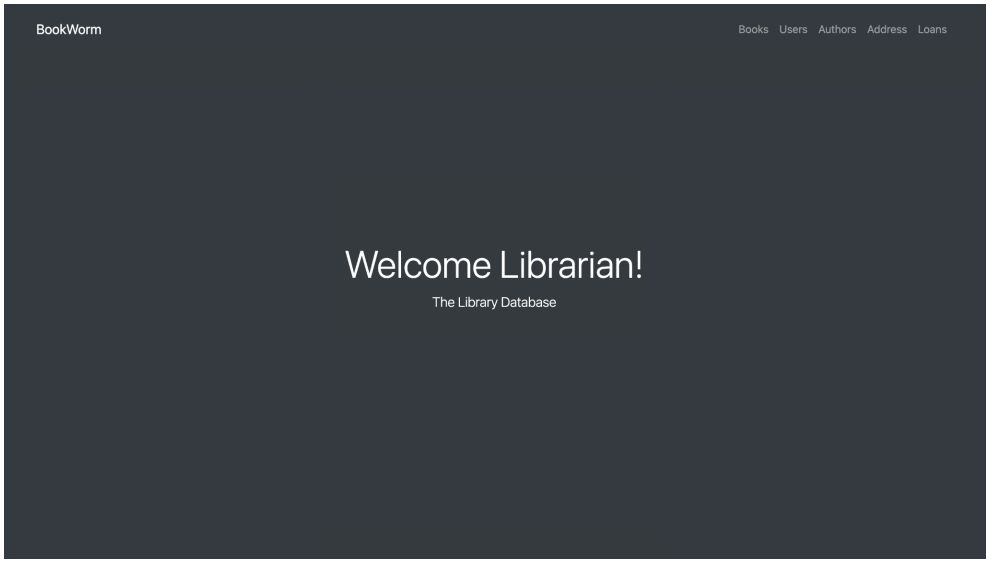


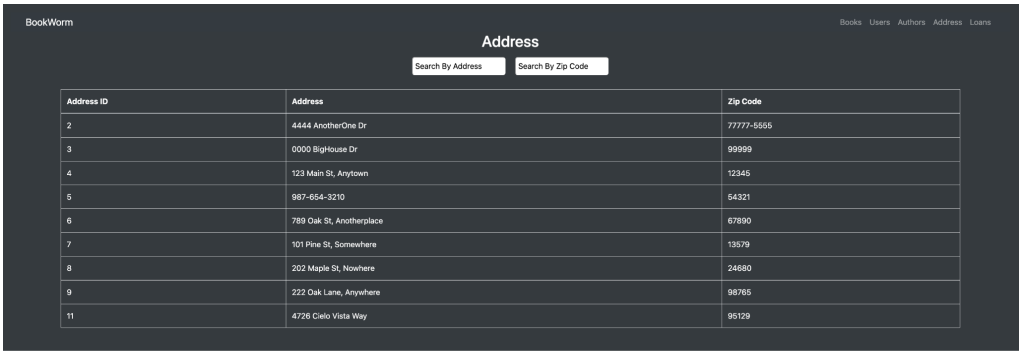
The "address" table uses Third Normal Form (3NF), a structured schema without data redundancy or dependency anomalies. Each attribute within the table holds atomic values, eliminating any repeating groups or arrays. The primary key ("id", “address”) uniquely identifies each address entry, and all other attributes, such as "zip_code," solely depend on this key, ensuring no partial dependencies exist. There are no transitive dependencies, as all non-prime attributes directly relate to the primary key.

Results

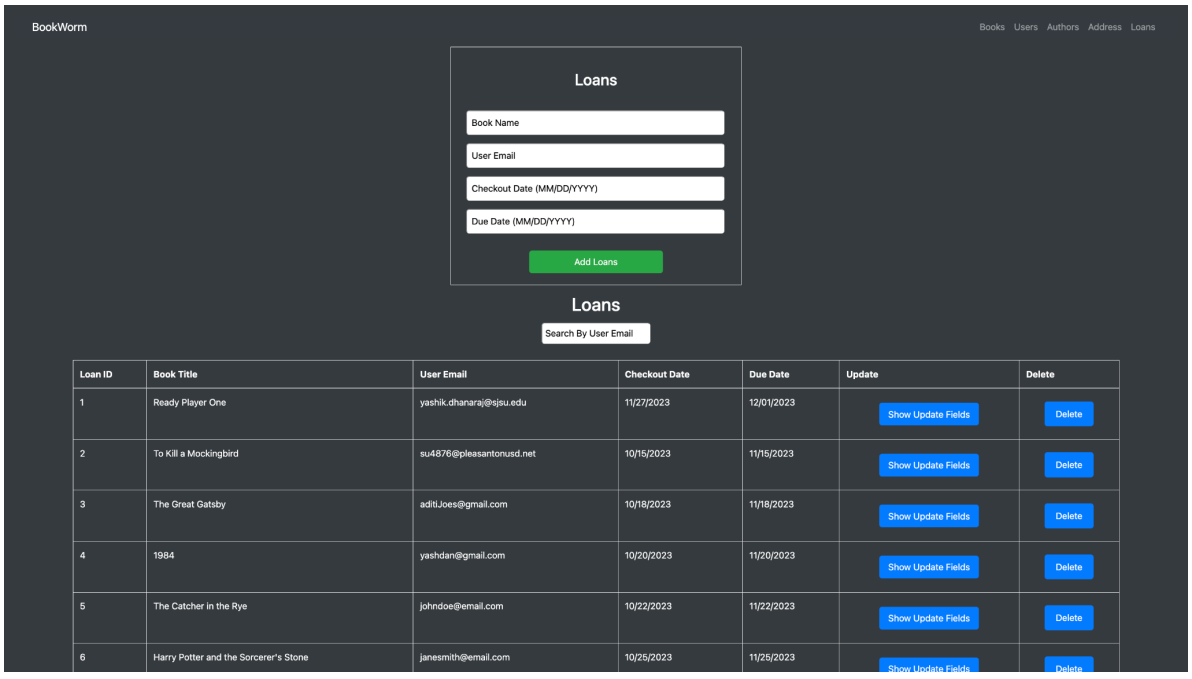
BookWorm is an intuitive CRUD Python web application built with Flask that empowers librarians to seamlessly manage a library's catalog of books, authors, authors, and circulation data across user-friendly forms linked to an underlying SQL database (using the SQLAlchemy toolkit). Librarians have the ability to add/delete/edit users, books, addresses, loans, and authors. Our application handles integrity errors and successfully adds items with alerts. It checks for a unique phone number & email when adding a library user. It also checks the Books table to verify there are enough copies of a book when creating a loan (checking a book out) and decrements the amount of the specified book in the Book table. When a user with the address does not exist anymore, the value from the Address table automatically gets deleted. Our application has the ability to search through any table created along with an advanced search option for the Address table.

Final Deliverables

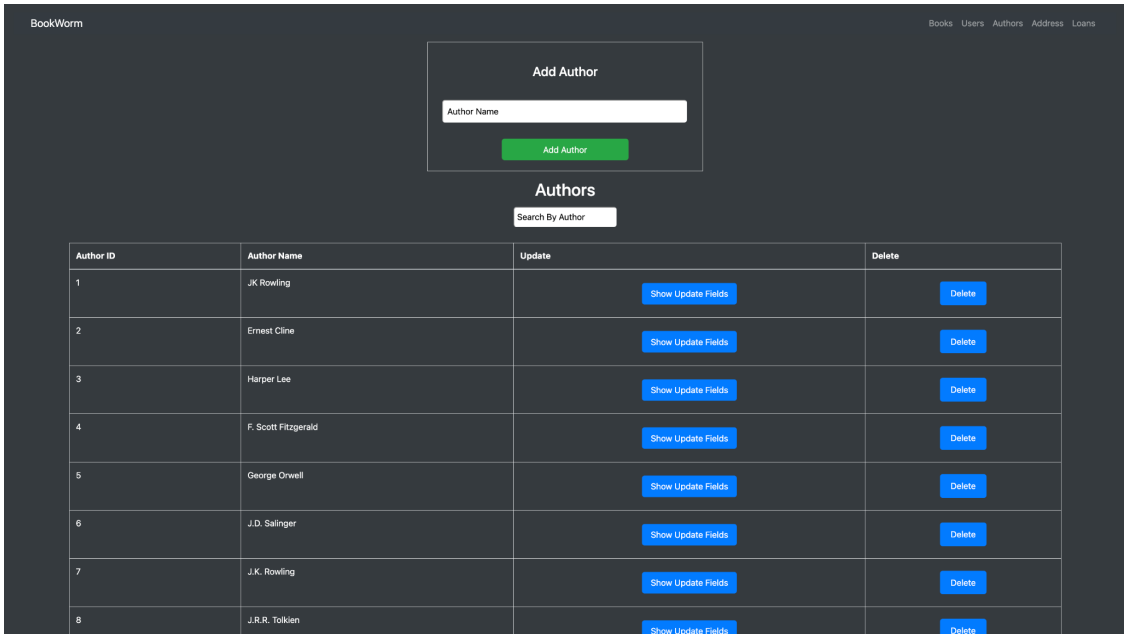


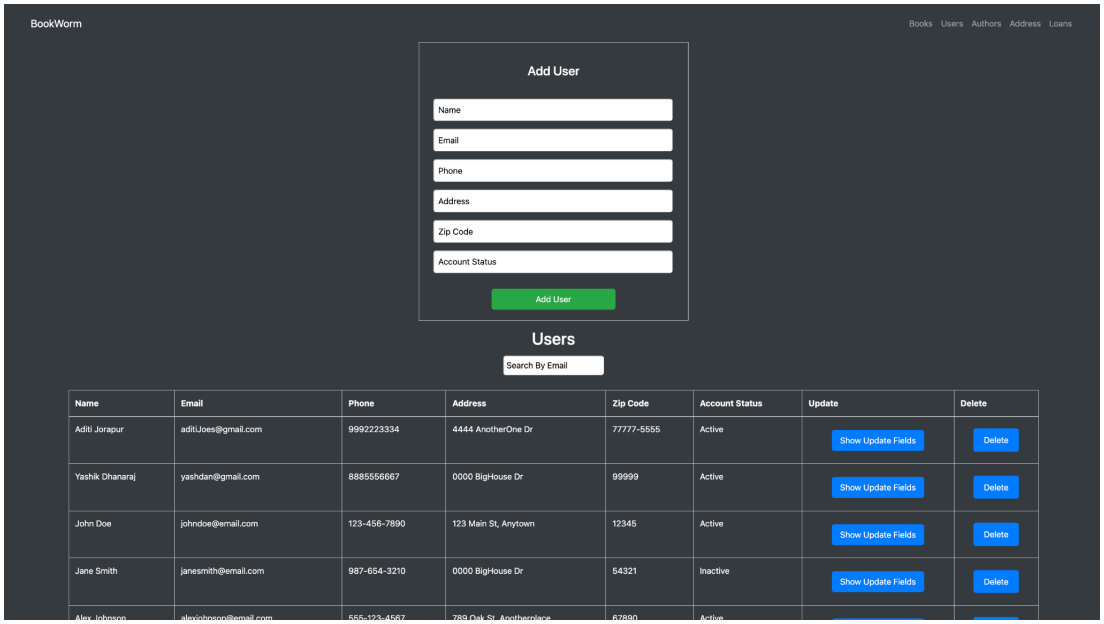


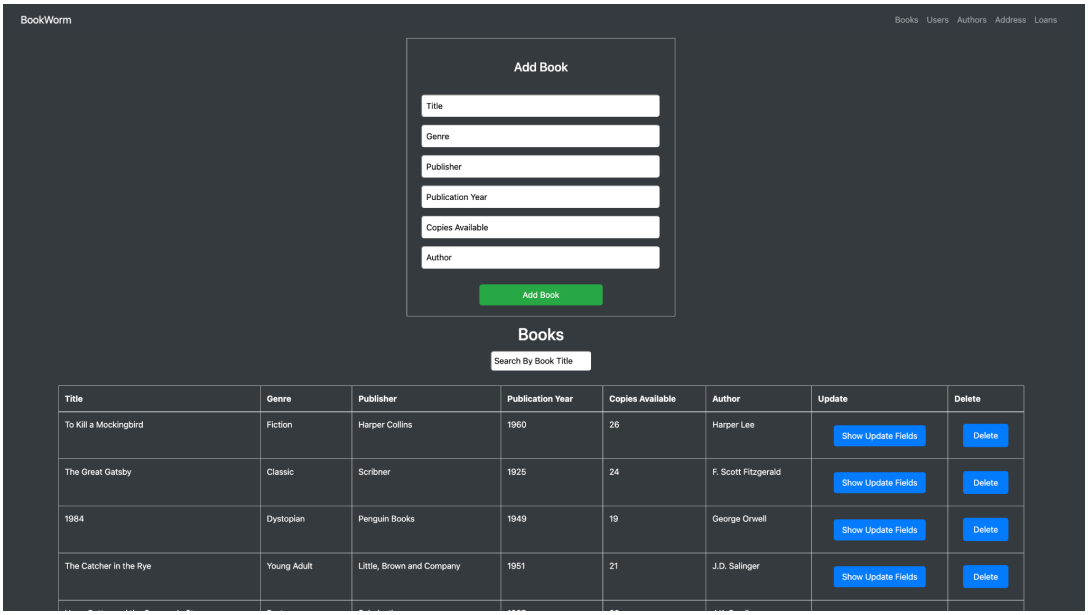
Address ID	Address	Zip Code
2	4444 AnotherOne Dr	77777-5555
3	0000 BigHouse Dr	99999
4	123 Main St, Anytown	12345
5	987-654-3210	54321
6	789 Oak St, Anotherplace	67890
7	101 Pine St, Somewhere	13579
8	202 Maple St, Nowhere	24680
9	222 Oak Lane, Anywhere	98765
11	4726 Cielo Vista Way	95129



Loan ID	Book Title	User Email	Checkout Date	Due Date	Update	Delete
1	Ready Player One	yashik.dhanaraj@sjsu.edu	11/27/2023	12/01/2023	Show Update Fields	Delete
2	To Kill a Mockingbird	su4876@pleasantonusd.net	10/15/2023	11/15/2023	Show Update Fields	Delete
3	The Great Gatsby	aditiJoes@gmail.com	10/18/2023	11/18/2023	Show Update Fields	Delete
4	1984	yashdan@gmail.com	10/20/2023	11/20/2023	Show Update Fields	Delete
5	The Catcher in the Rye	johndoe@email.com	10/22/2023	11/22/2023	Show Update Fields	Delete
6	Harry Potter and the Sorcerer's Stone	janesmith@email.com	10/25/2023	11/25/2023	Show Update Fields	Delete


BookWorm
Books Users Authors Address Loans
Add Author
Author Name
Add Author
Authors
Search By Author
Author ID
Author Name
Update
Delete
1
JK Rowling
Show Update Fields
Delete
2
Ernest Cline
Show Update Fields
Delete
3
Harper Lee
Show Update Fields
Delete
4
F. Scott Fitzgerald
Show Update Fields
Delete
5
George Orwell
Show Update Fields
Delete
6
J.D. Salinger
Show Update Fields
Delete
7
J.K. Rowling
Show Update Fields
Delete
8
J.R.R. Tolkien
Show Update Fields
Delete





Contribution/Work done by each team member

All three members of the team worked together to plan the tech stack and the tables that were to be created. Yashik worked on creating the Users table and the Books table. Aditi worked on the Authors and Addresses table. Sharanya worked on the Loans table and the front end portion of the application. All three worked on creating the presentation and the final report for the project.

References/ Any additional sources?

- **CMPE 131 Notes**: 09_Databases.pdf
- **Flask API:** https://flask.palletsprojects.com/en/3.0.x/
- **SQLAlchemy API**: https://www.sqlalchemy.org
- **Bootstrap Documentation**: https://getbootstrap.com/docs/5.3/getting-started/introduction/

● GitHub Repository with source code and project report
https://github.com/dyashik/157Project.git
● A Readme in the GitHub repository having instructions to run your application (The application should be runnable without errors)
https://github.com/dyashik/157Project/blob/main/README.md