Library Management System for Group-Based Student Borrowing

Temitope Oyemade

https://temitopejoshua.github.io/db-personal-app/

Department of Computer Science

Bowie State University

1. Executive Summary

This document outlines the design, development, and deployment of a Library Management System tailored to manage book borrowing and return processes in a classroom setting. The system addresses the needs of 78 library books, a class of students categorized into groups, and their specific borrowing preferences. It incorporates features such as borrowing status, return tracking, group membership identification, and book ratings.

2. Problem Definition

The challenge is to design a database system for a class with students divided into four groups (A-D), each consisting of three members. Each student has three preferred books from a longlist of 78 books. The system should manage book borrowing, return dates, and due notifications. The database must allow searching by student name to display borrowed books and corresponding due dates. Additional features include average book rating calculations and listing books by rating performance.

3. Requirements
   - Functional Requirements:
     - Book Management: Create and manage a database of 78 books with relevant details (title, author, rating, etc).
     - Student Management: Store student information, including names, groups, and book preferences.
     - Borrowing Records: Track borrowing status, due dates, and return information.
     - Queries: Users can search by student name, group membership, and book availability.
     - Reporting: Display books with above and below-average ratings.
   - Non-Functional Requirements:
     - Usability: User-friendly interface for queries.
     - Accessibility: Hosted on GitHub Pages.

4. Proposed Solution

We propose a Library Management System with the following features:

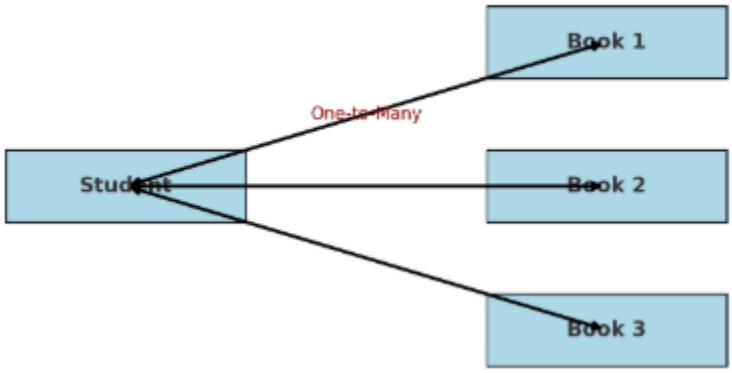
- A relational database to manage books, students, borrowing transactions, and group memberships.
- A user interface (UI) for querying borrowing details and viewing group information.
- A backend system to calculate average ratings and display books above or below the average.
- A method to dynamically update and query the database.

The system will be developed using Java for the backend, MySQL for the database, and a web interface hosted on GitHub Pages.

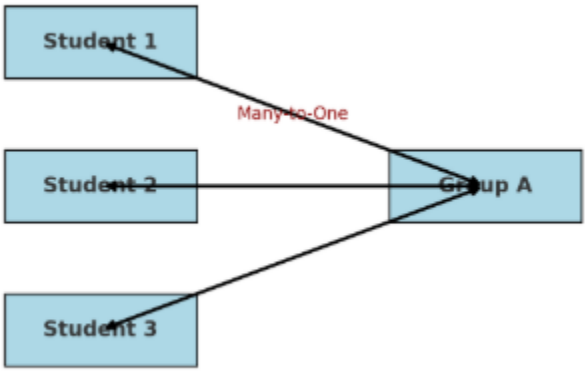
5. Design and Methodologies

5.1. Database Design

- Entities:
    - Books (BookID, Title, Author, Rating, BorrowedStatus)
    - Students (StudentID, FirstName, LastName, Group, PreferredBooks)
    - Borrowing (TransactionID, BookID, StudentID, BorrowDate, ReturnDate)
    - Groups Table: (group_id, group_name)
- Relationships and Entity-Relationship Diagram (ERD):
    - One-to-Many: A student can borrow multiple books.



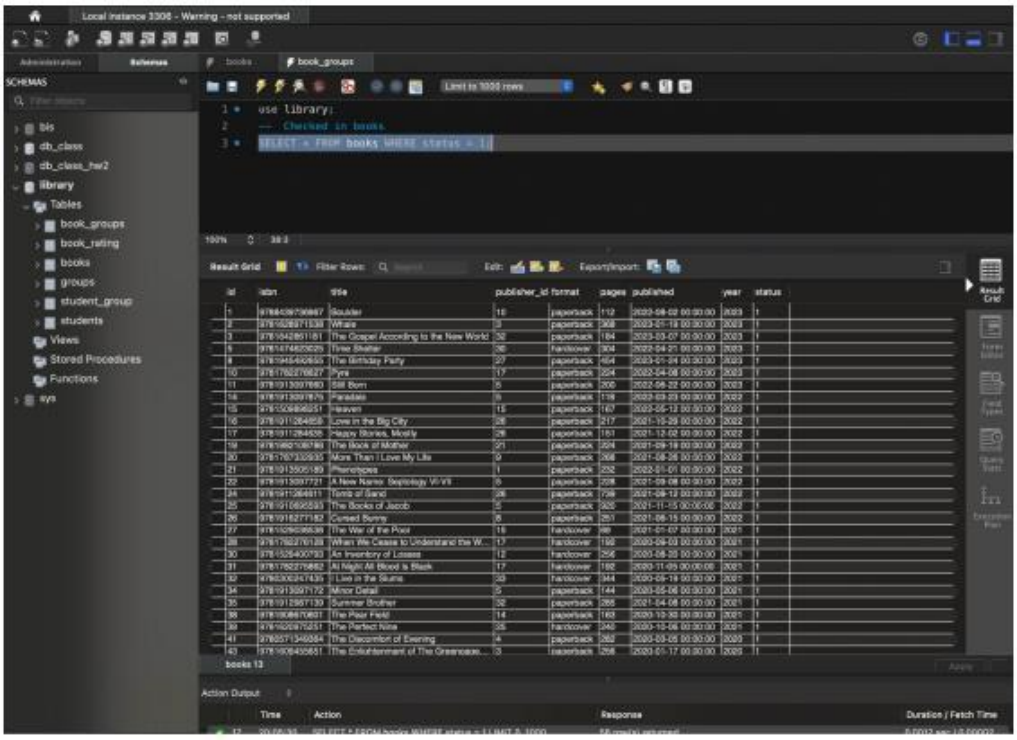
    - Many-to-One: Multiple students (3) belong to a group.



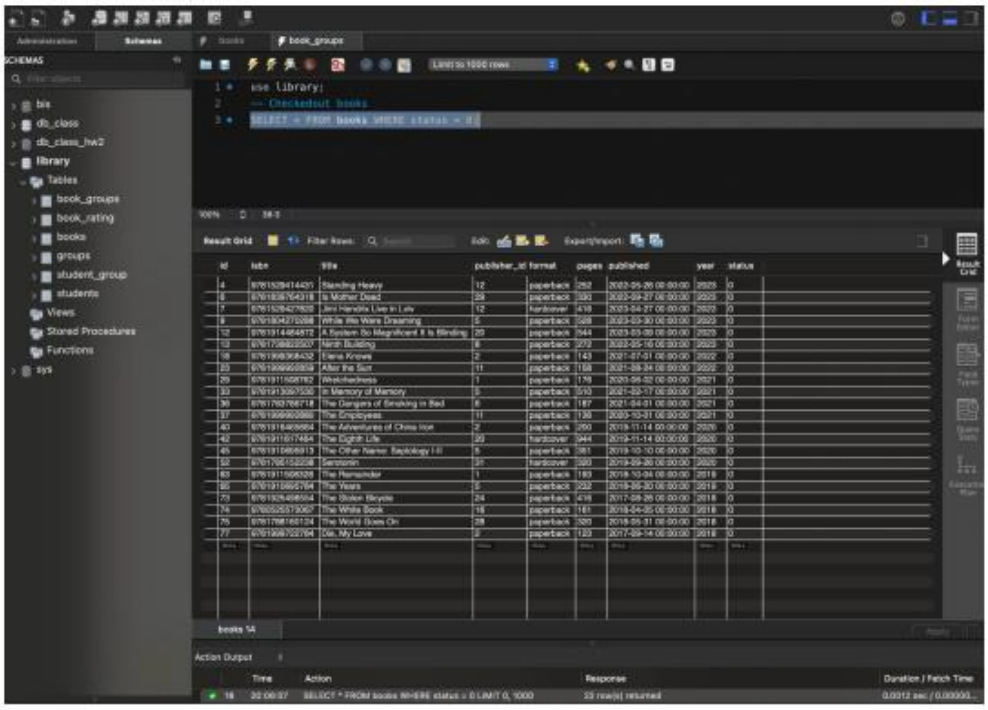
Normalization: Ensuring no data redundancy and efficient querying by dividing the database into normalized tables.

- SQL Queries:
    - Query for borrowed books and return status:

SELECT * FROM books WHERE status = 1;

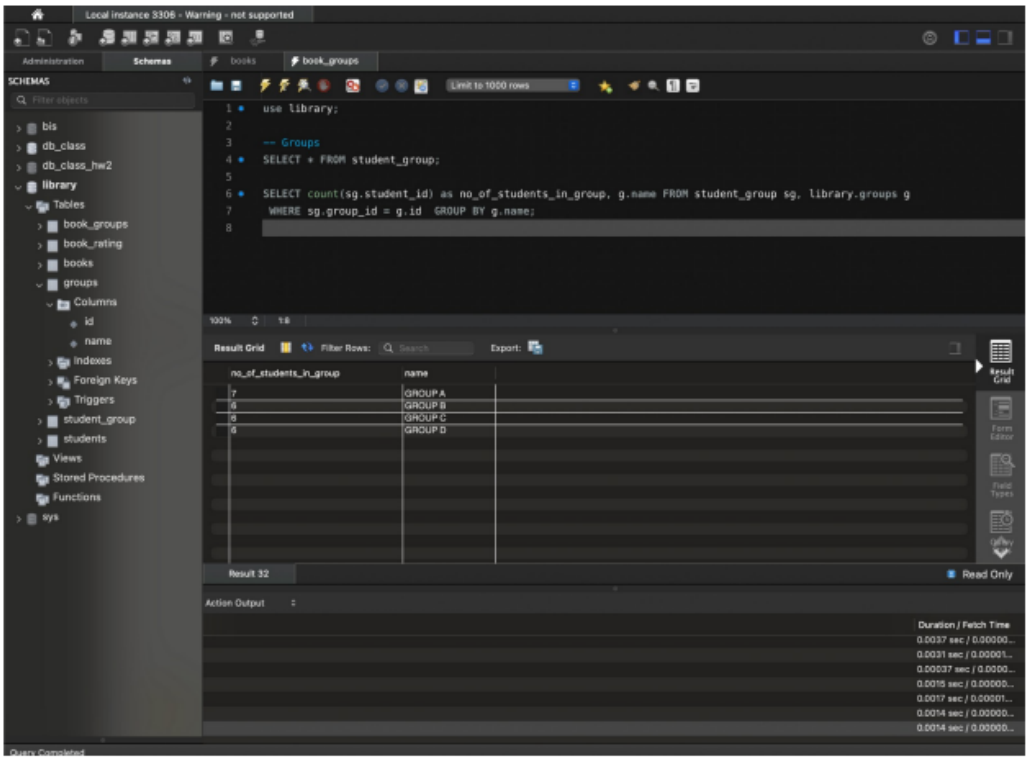


SELECT * FROM books WHERE status = 0;

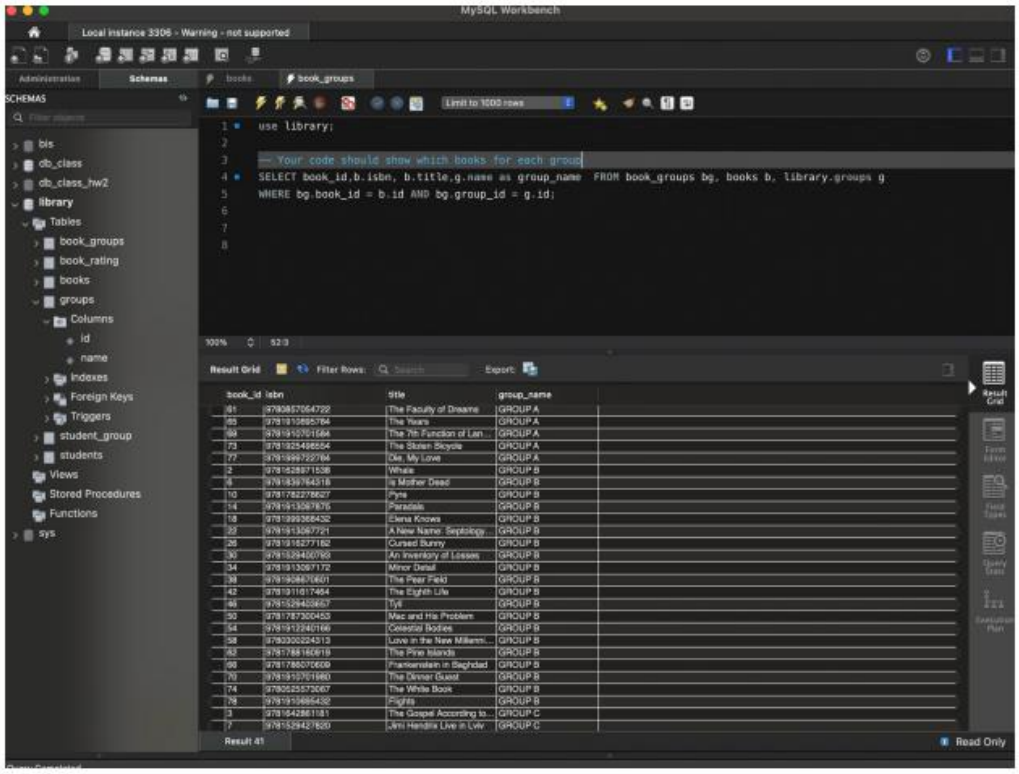


- Query for group affiliations and preferred books.

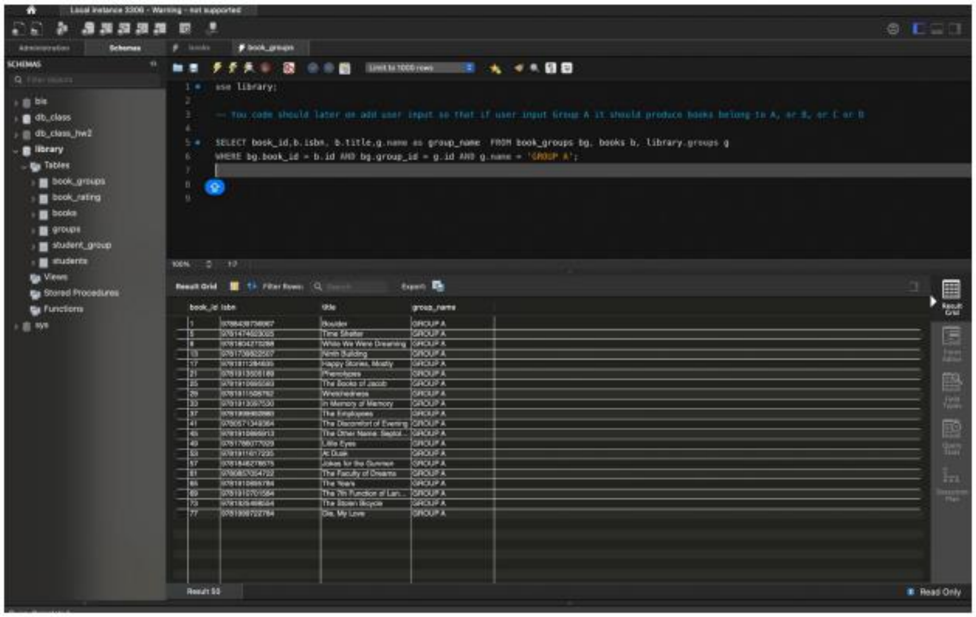
```
CREATE TABLE `groups` (
 `id` int NOT NULL AUTO_INCREMENT,
 `name` varchar(45) DEFAULT NULL,
 PRIMARY KEY (`id`),
 UNIQUE KEY `id_UNIQUE` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
SELECT count(sg.student_id) as no_of_students_in_group, g.name FROM student_group sg,
library.groups g
 WHERE sg.group_id = g.id GROUP BY g.name;
```

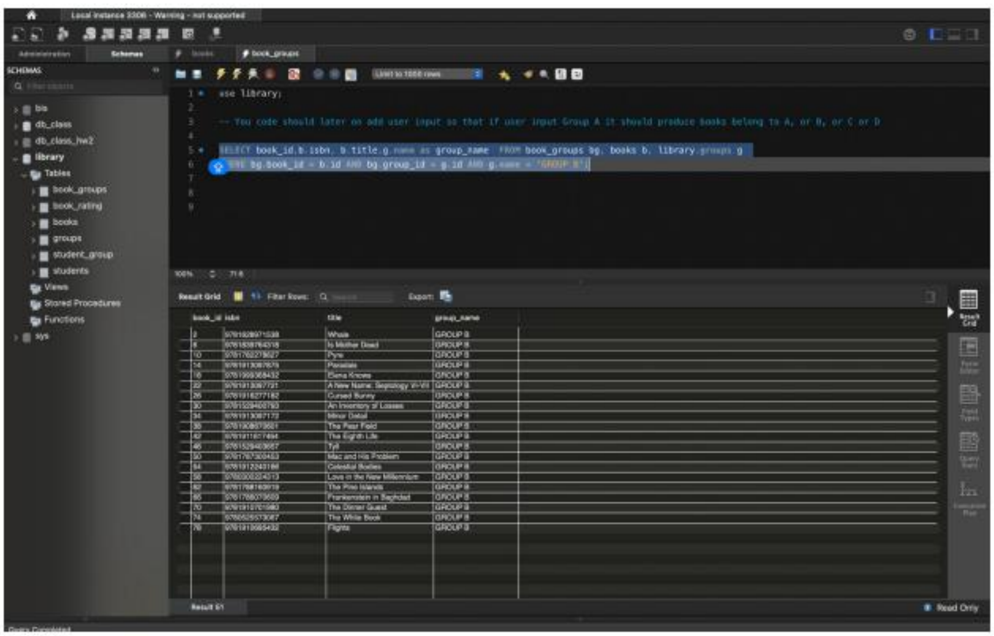


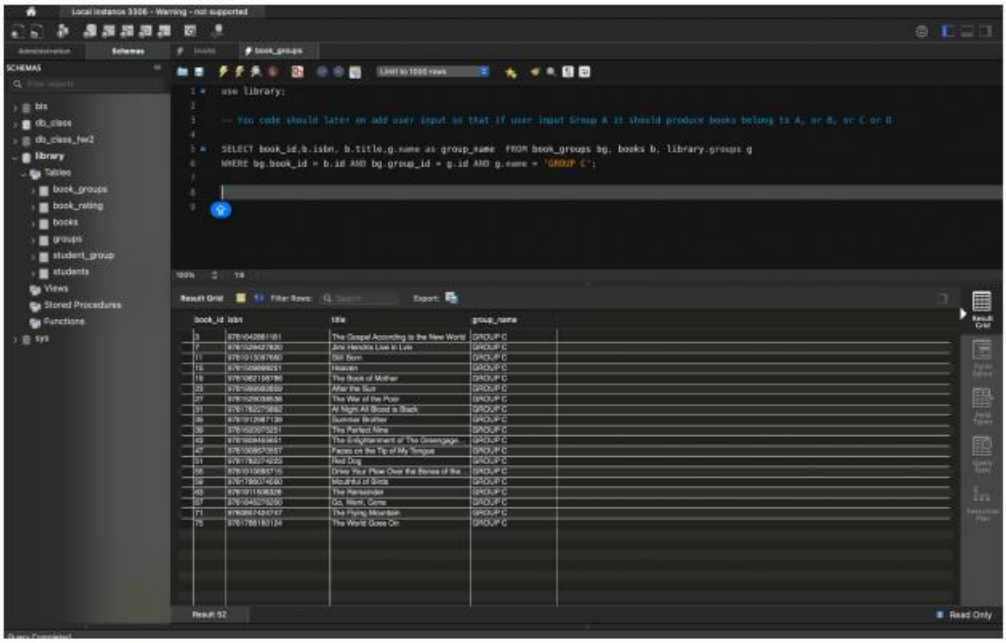
SELECT book_id,b.isbn, b.title,g.name as group_name FROM book_groups bg, books b, library.groups g
WHERE bg.book_id = b.id AND bg.group_id = g.id;

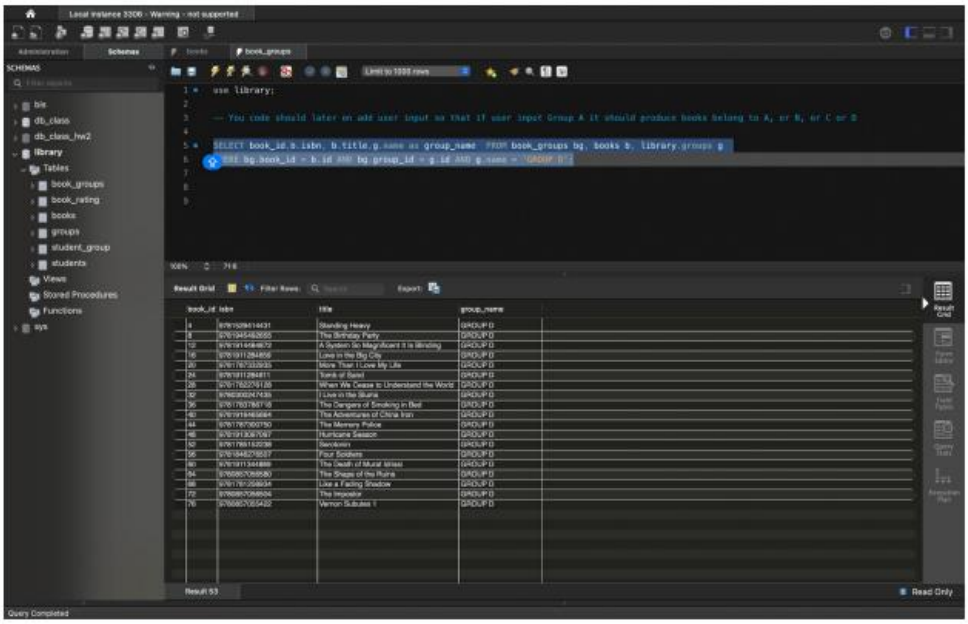


SELECT book_id,b.isbn, b.title,g.name as group_name FROM book_groups bg, books b, library.groups g
WHERE bg.book_id = b.id AND bg.group_id = g.id AND g.name = 'GROUP A';


Local instance 3306 - Warning - not supported
book_groups
SCHEMAS
library
Tables
book_groups
book_rating
books
groups
student_group
students
Views
Stored Procedures
Functions
sys
use library;
-- You code should later on add user input so that if user input Group A it should produce books belong to A, or B, or C or D
SELECT book_id,b.isbn, b.title,g.name as group_name FROM book_groups bg, books b, library.groups g
WHERE bg.book_id = b.id AND bg.group_id = g.id AND g.name = 'GROUP A';
Result Grid
book_id
isbn
title
group_name
Read Only

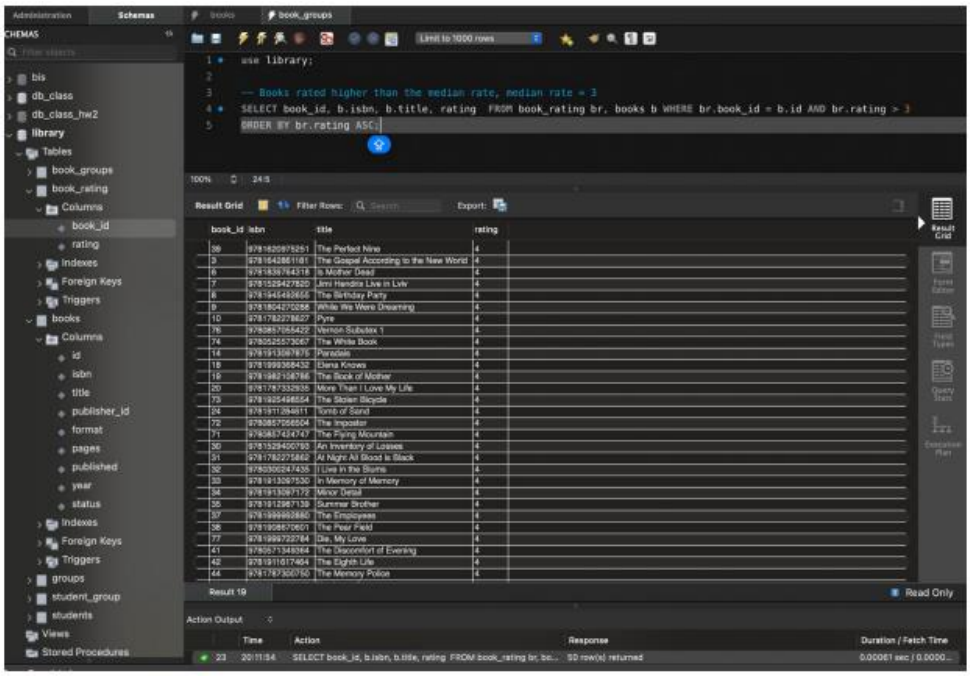

Local instance 3306 - Warning - not supported
book_groups
SCHEMAS
library
Tables
book_groups
book_rating
books
groups
student_group
students
Views
Stored Procedures
Functions
sys
use library;
-- You code should later on add user input so that if user input Group A it should produce books belong to A, or B, or C or D
Result Grid
book_id
isbn
title
group_name
Read Only



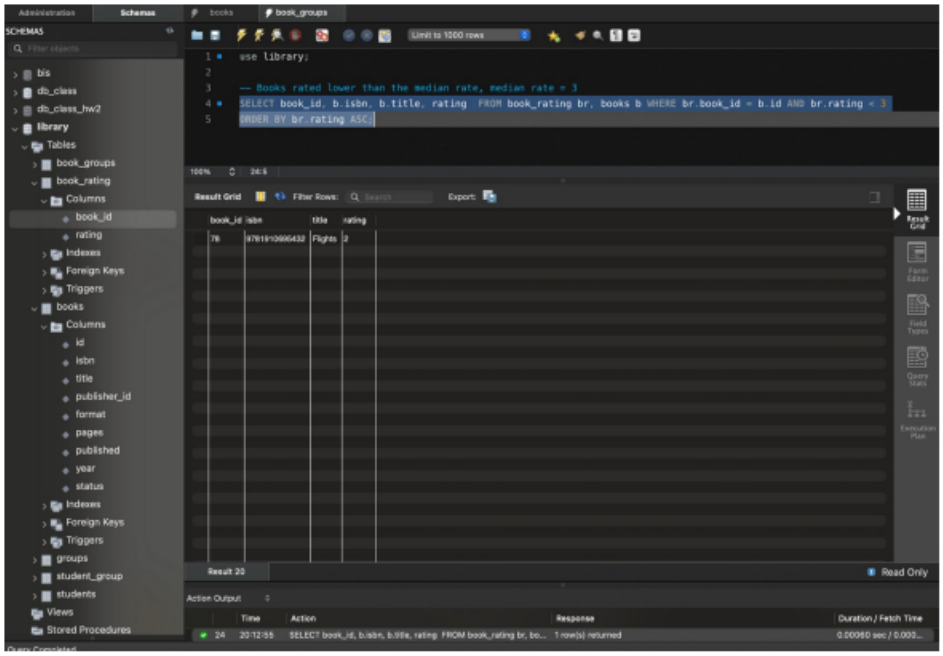


- Query for book ratings above or below average.

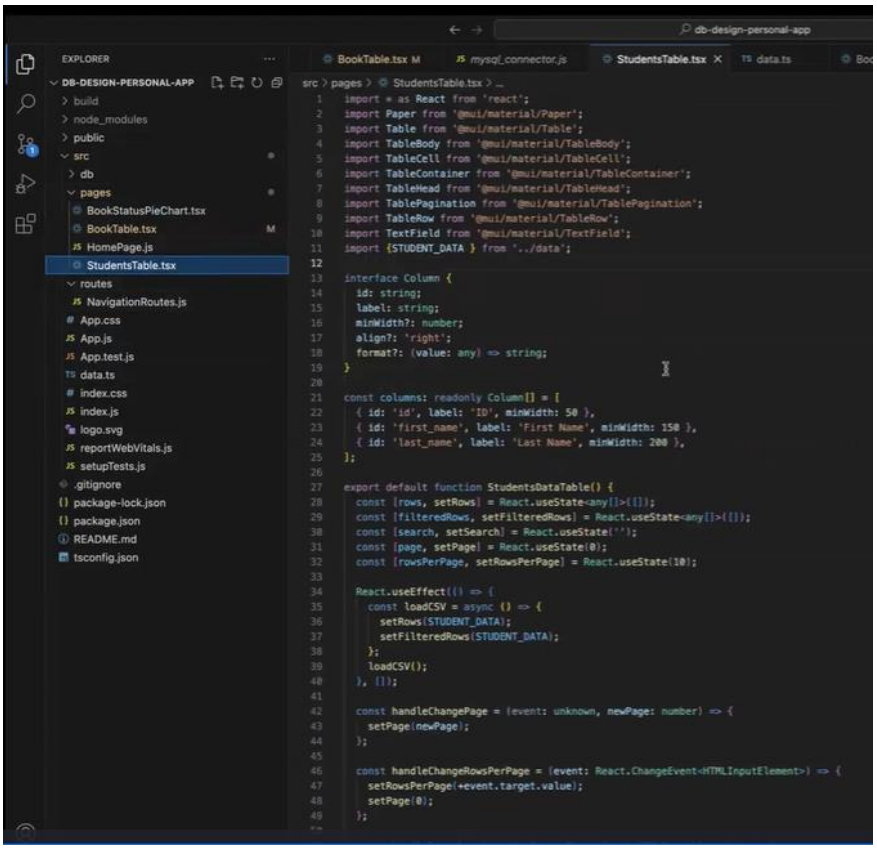
Books rated higher than the median rate, median rate = 3
SELECT b.id,b.isbn, b.title, rating FROM book_rating br, books b WHERE br.book_id = b.id AND br.rating > 3;

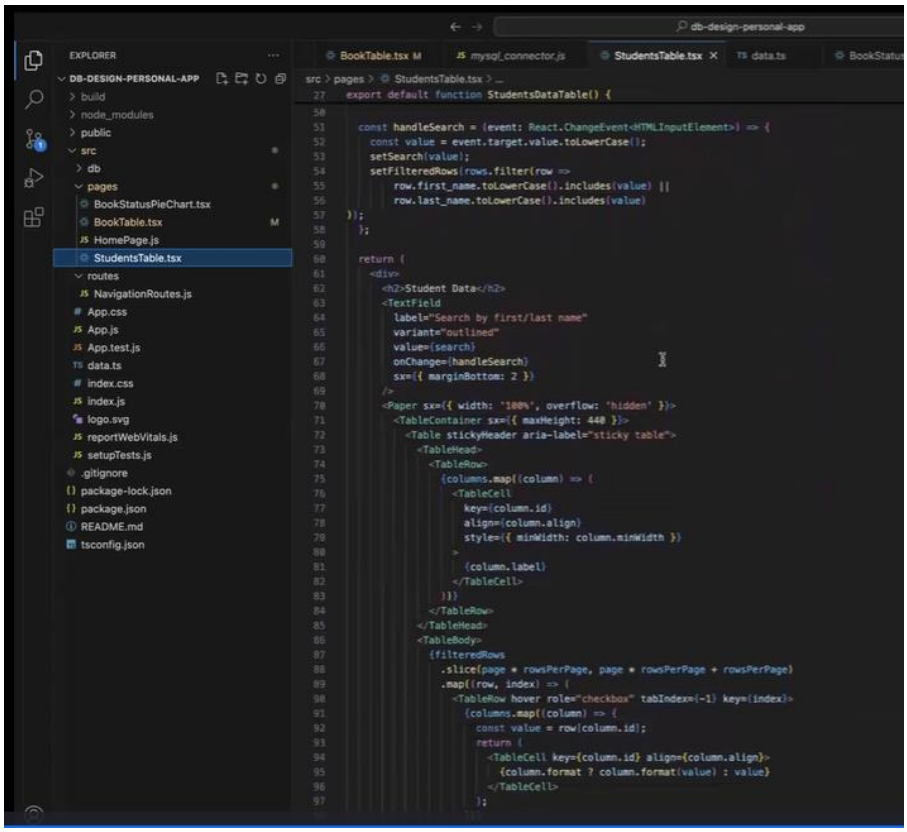


Books rated lower than the median rate, median rate = 3 SELECT b.id,b.isbn, b.title, rating FROM book_rating br, books b WHERE br.book_id = b.id AND br.rating < 3;



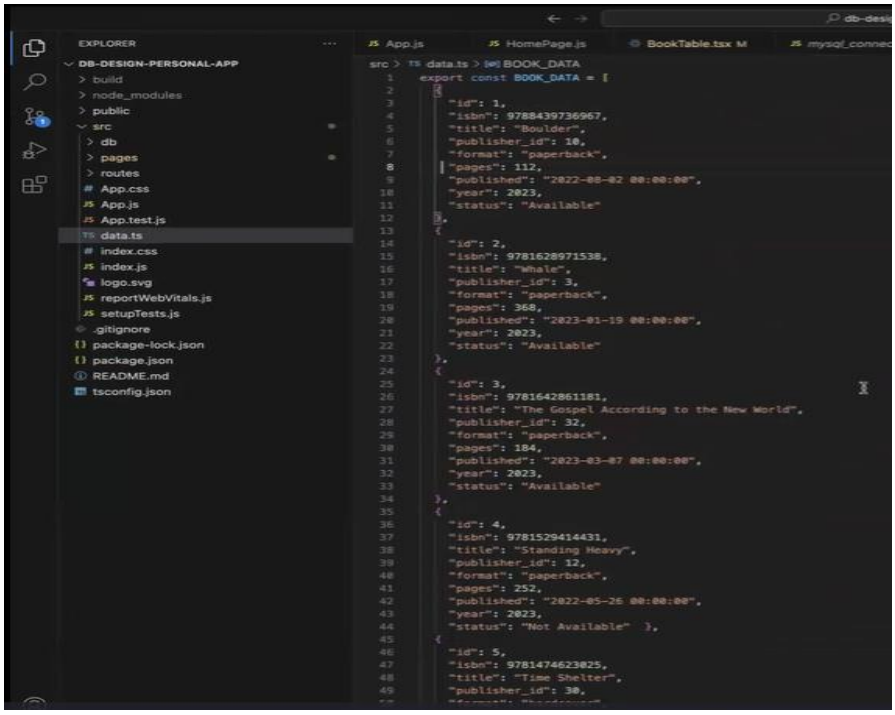
- Sequelize models: These are models to import the codes from MySQL database
  - Student data Model:


db-design-personal-app
EXPLORER
DB-DESIGN-PERSONAL-APP
build
node_modules
public
src
db
pages
BookStatusPieChart.tsx
BookTable.tsx M
HomePage.js
StudentsTable.tsx
routes
NavigationRoutes.js
App.css
App.js
App.test.js
data.ts
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
tsconfig.json
BookTable.tsx M
mysql_connector.js
StudentsTable.tsx
data.ts
src > pages > StudentsTable.tsx > ...
import * as React from 'react';
import Paper from '@mui/material/Paper';
import Table from '@mui/material/Table';
import TableBody from '@mui/material/TableBody';
import TableCell from '@mui/material/TableCell';
import TableContainer from '@mui/material/TableContainer';
import TableHead from '@mui/material/TableHead';
import TablePagination from '@mui/material/TablePagination';
import TableRow from '@mui/material/TableRow';
import TextField from '@mui/material/TextField';
import {STUDENT_DATA } from '../data';
interface Column {
id: string;
label: string;
minWidth?: number;
align?: 'right';
format?: (value: any) => string;
}
const columns: readonly Column[] = [
{ id: 'id', label: 'ID', minWidth: 50 },
{ id: 'first_name', label: 'First Name', minWidth: 150 },
{ id: 'last_name', label: 'Last Name', minWidth: 200 },
];
export default function StudentsDataTable() {
const [rows, setRows] = React.useState<any[]>([]);
const [filteredRows, setFilteredRows] = React.useState<any[]>([]);
const [search, setSearch] = React.useState('');
const [page, setPage] = React.useState(0);
const [rowsPerPage, setRowsPerPage] = React.useState(10);
React.useEffect(() => {
const loadCSV = async () => {
setRows(STUDENT_DATA);
setFilteredRows(STUDENT_DATA);
};
loadCSV();
}, []);
const handleChangePage = (event: unknown, newPage: number) => {
setPage(newPage);
};
const handleChangeRowsPerPage = (event: React.ChangeEvent<HTMLInputElement>) => {
setRowsPerPage(+event.target.value);
setPage(0);
};

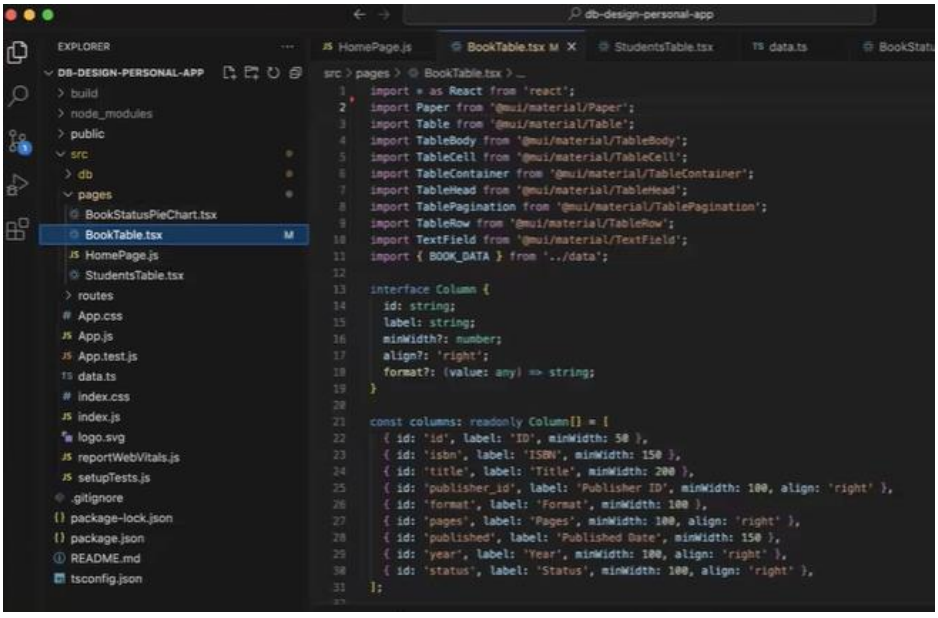

db-design-personal-app
EXPLORER
DB-DESIGN-PERSONAL-APP
build
node_modules
public
src
db
pages
BookStatusPieChart.tsx
BookTable.tsx M
HomePage.js
StudentsTable.tsx
routes
NavigationRoutes.js
App.css
App.js
App.test.js
data.ts
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
tsconfig.json
BookTable.tsx M
mysql_connector.js
StudentsTable.tsx
data.ts
src > pages > StudentsTable.tsx > ...
export default function StudentsDataTable() {
const handleSearch = (event: React.ChangeEvent<HTMLInputElement>) => {
const value = event.target.value.toLowerCase();
setSearch(value);
setFilteredRows(rows.filter(row =>
row.first_name.toLowerCase().includes(value) ||
row.last_name.toLowerCase().includes(value)
));
};
return (

<h2>Student Data</h2>
<TextField
label="Search by first/last name"
variant="outlined"
value={search}
onChange={handleSearch}
sx={{ marginBottom: 2 }}
/>
<Paper sx={{ width: '100%', overflow: 'hidden' }}>
<TableContainer sx={{ maxHeight: 440 }}>
<Table stickyHeader aria-label="sticky table">
<TableHead>
<TableRow>
{columns.map((column) => (
<TableCell
key={column.id}
align={column.align}
style={{ minWidth: column.minWidth }}
>
{column.label}
</TableCell>
))}
</TableRow>
</TableHead>
<TableBody>
{filteredRows
.slice(page * rowsPerPage, page * rowsPerPage + rowsPerPage)
.map((row, index) => {
<TableRow hover role="checkbox" tabIndex={-1} key={index}>
{columns.map((column) => {
const value = row[column.id];
return (
<TableCell key={column.id} align={column.align}>
{column.format ? column.format(value) : value}
</TableCell>
);

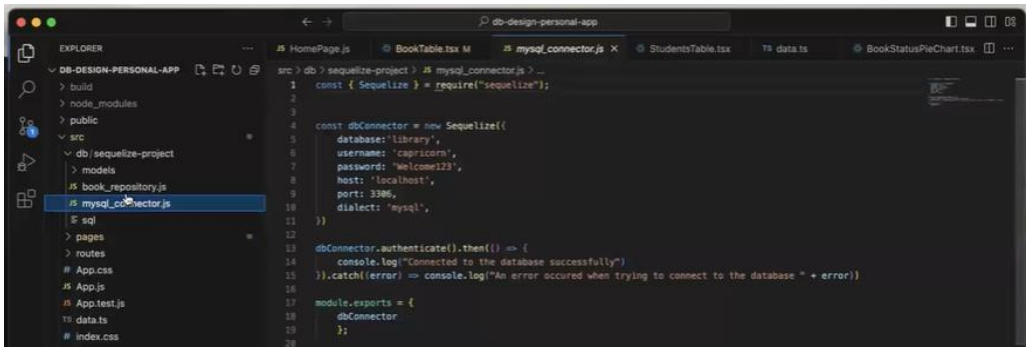
- Book data Model:



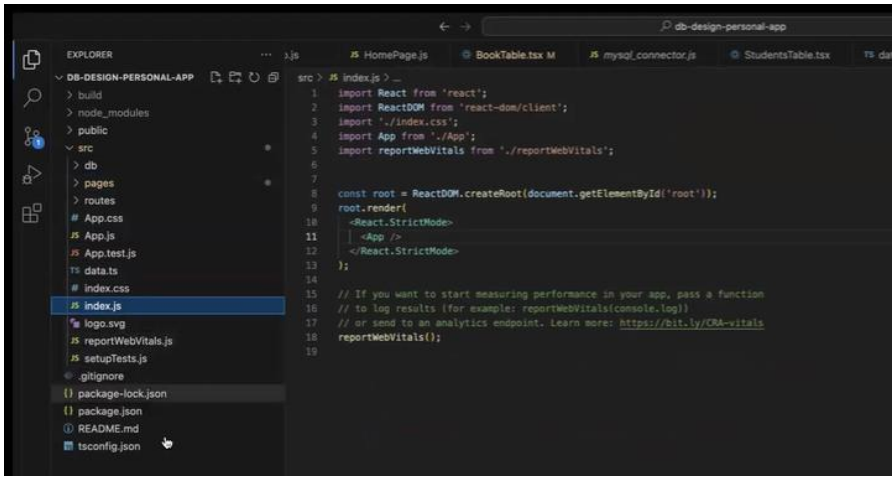
- Book Table Model:



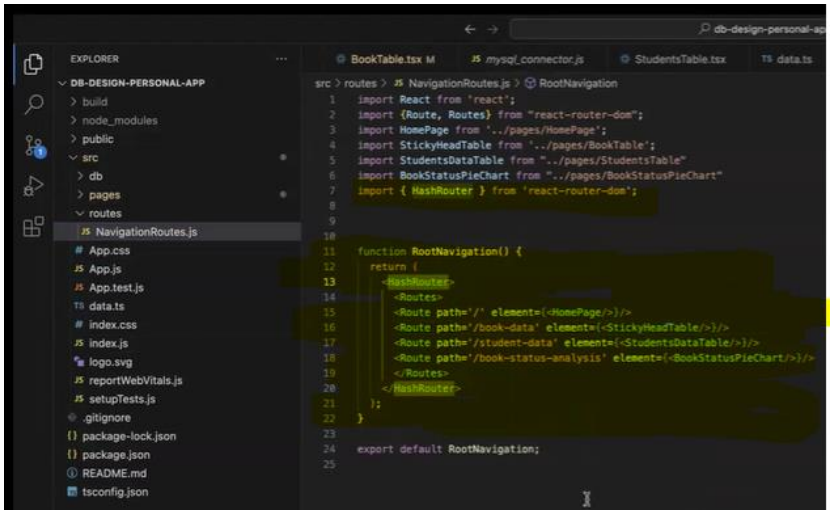
- Database Connector:



- Routers: These were used to connect the front-end our backend database
    - React: The front end was built with React

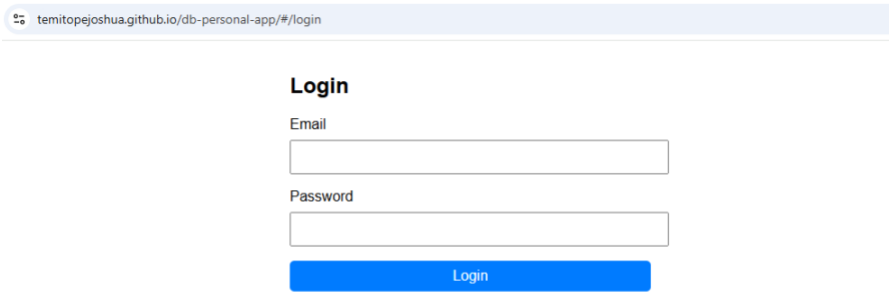


    - HashRoute: HashRoute was imported for connectivity with GitHub

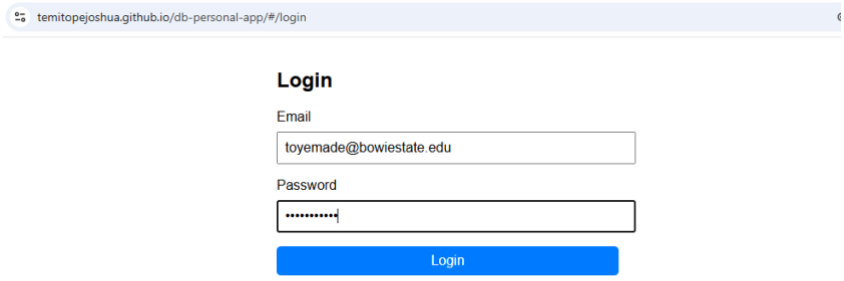


5.2. Front-end Demonstration

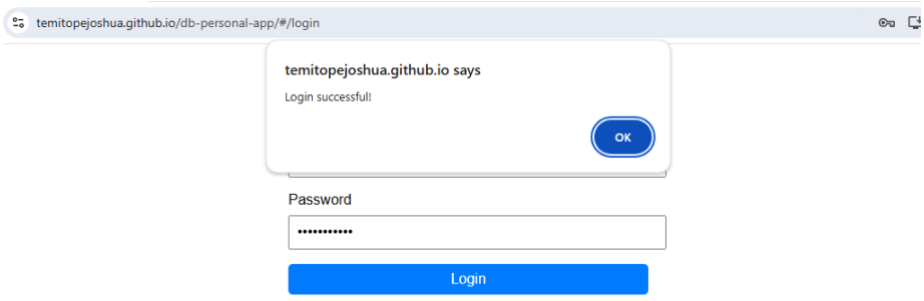
Click the link https://temitopejoshua.github.io/db-personal-app/ to access the front-end of the database through GitHub.



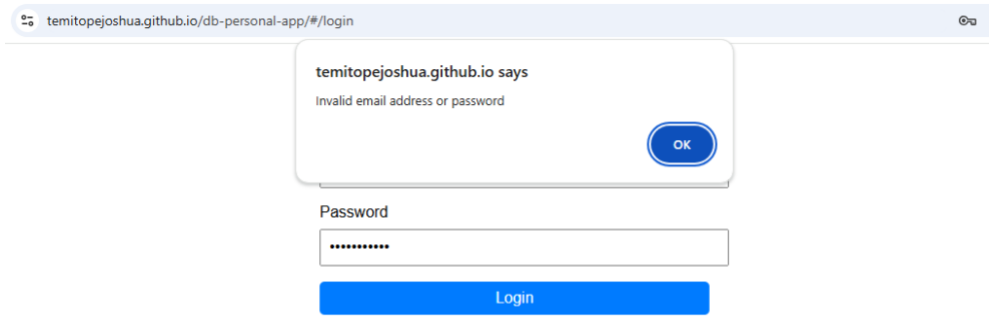
Enter email and password



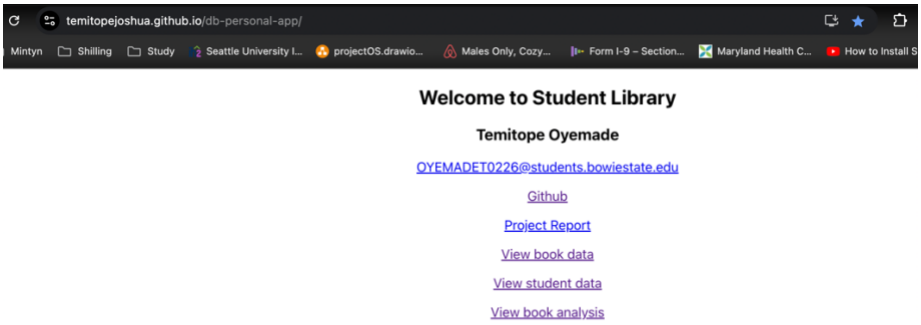
If the correct email and password are entered, access will be granted.



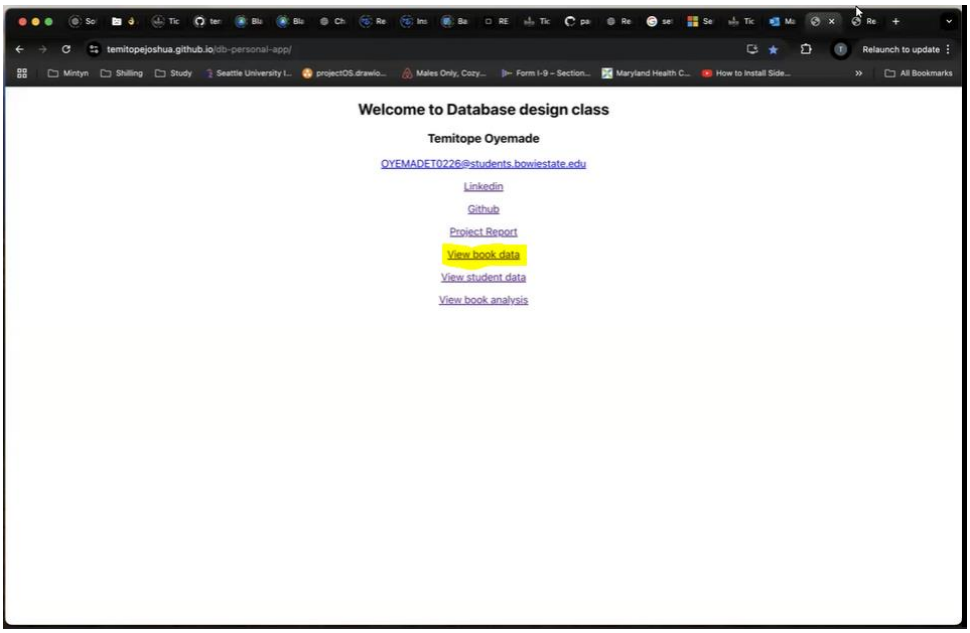
Otherwise, access will not be granted.



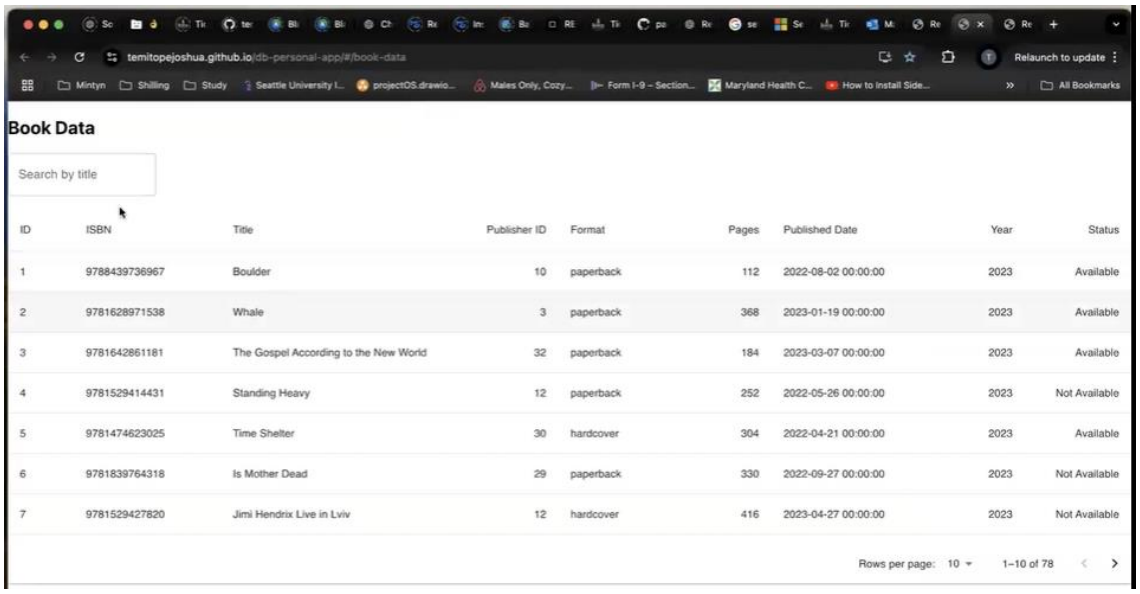
Once access is granted take any of the actions below to explore the page.



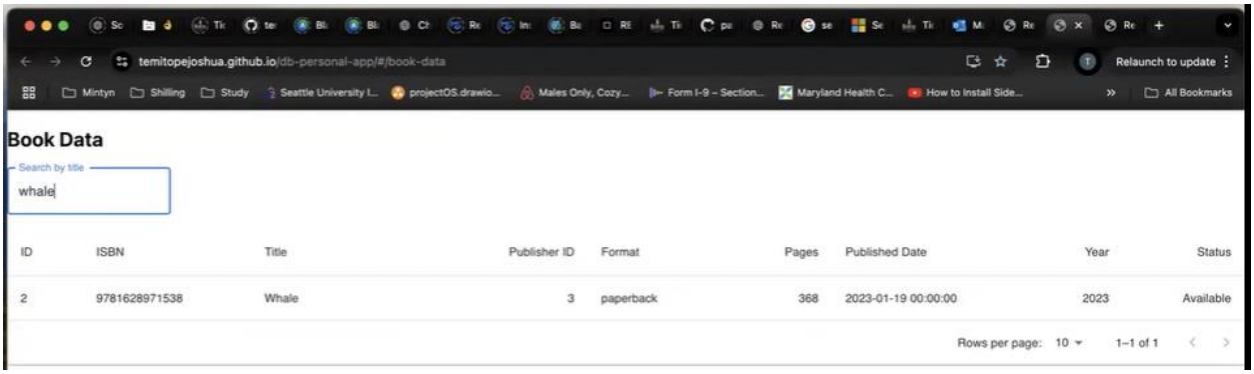
On the website, click on “View book data” to view the book data

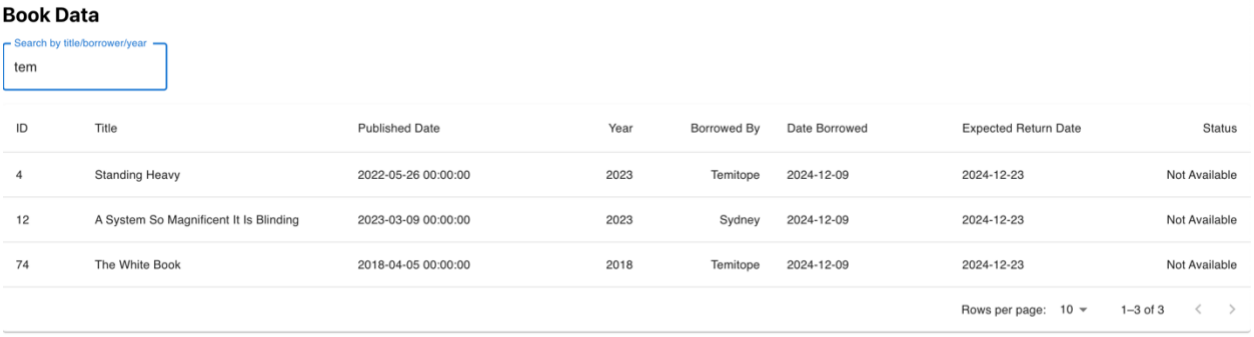


Webpage displaying Book Data

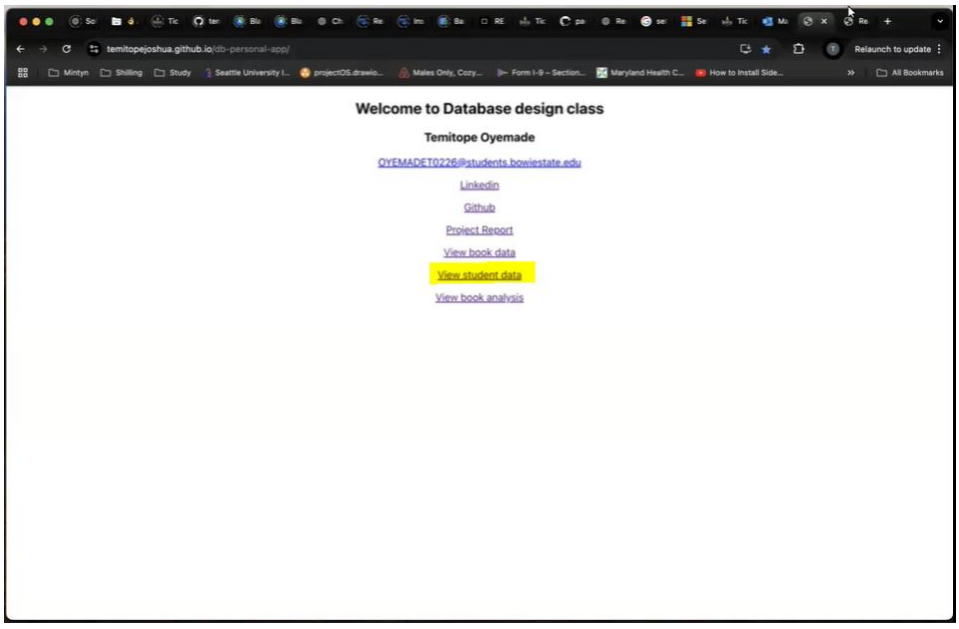


Search by Title/Borrower/Year – for example Whale, or Paradais or Temitope– as you begin to type, it begins to filter out your search

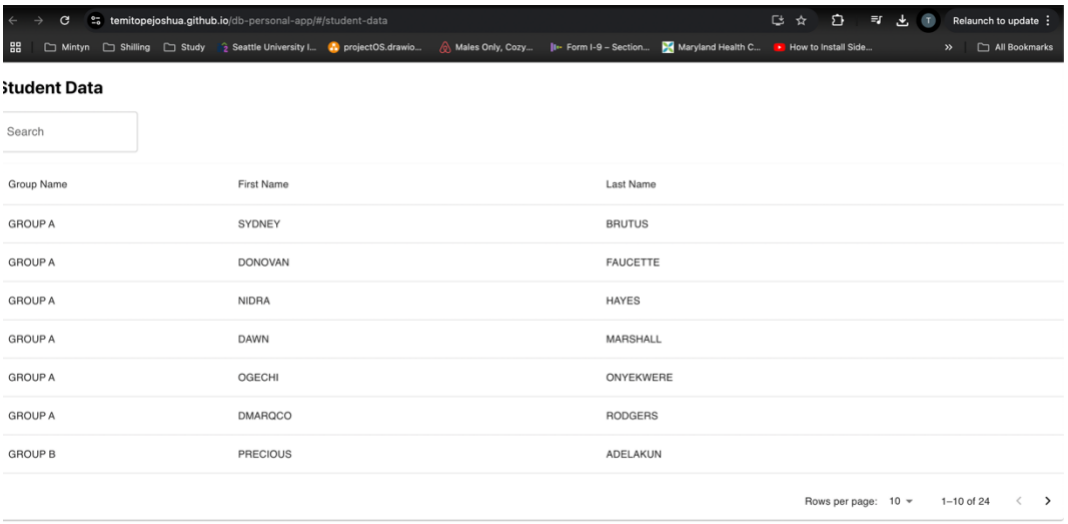




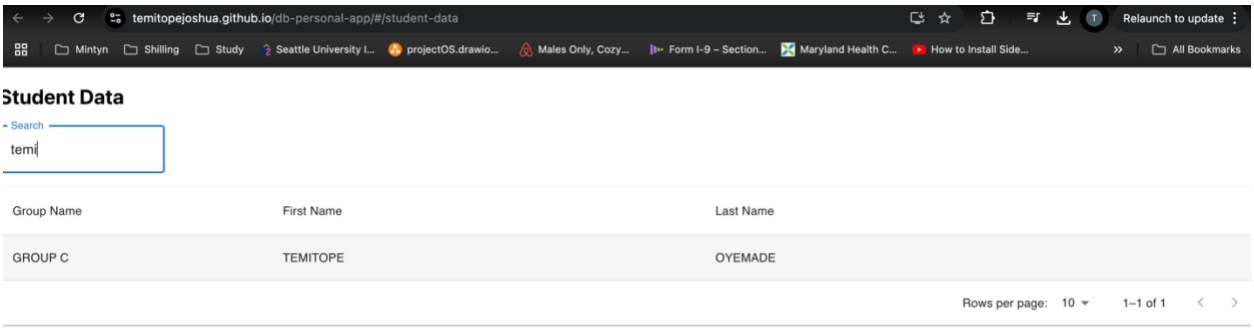
On the website, Click on “View student data” to view student data

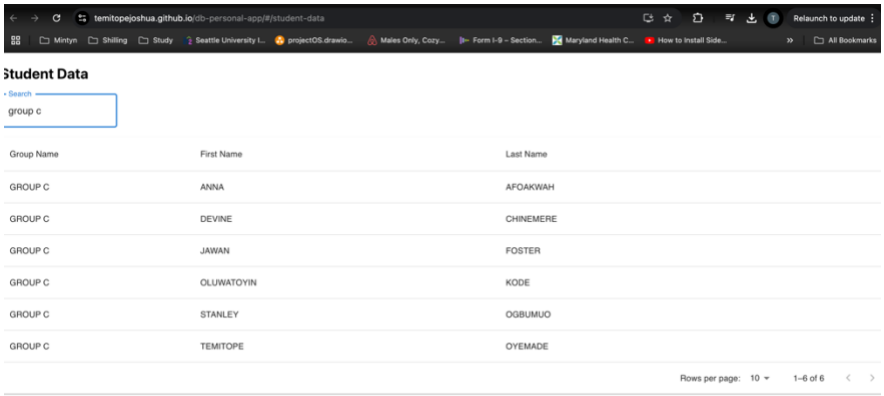


Webpage displaying Student Data and the group they belong to, you can view students that belong to a group by simply typing the group name in the search box.

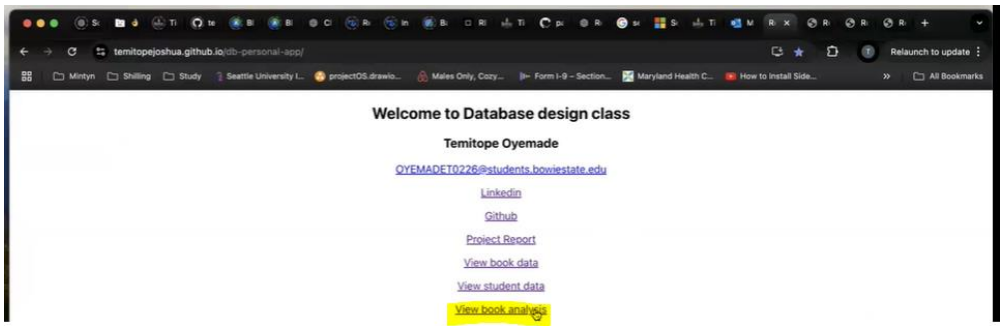


Users can search student data by first, last name or group name. For example as you begin to type “te” for Temitope all the name that have “te” are displayed or “ja” for Jared, all the name with “ja” are displayed.

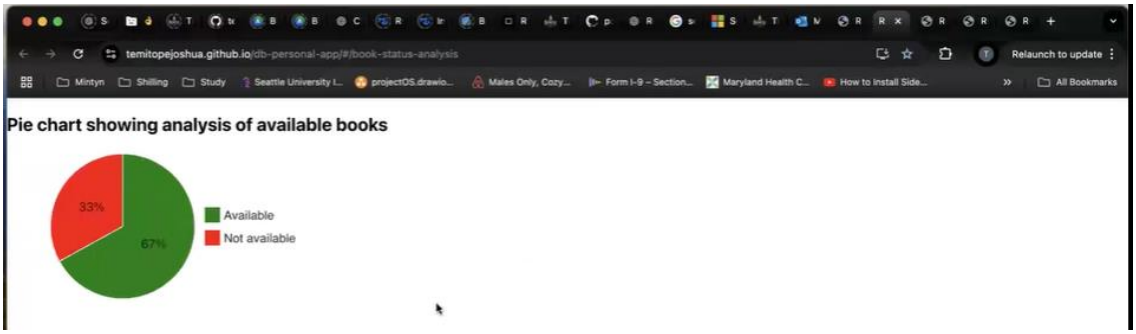




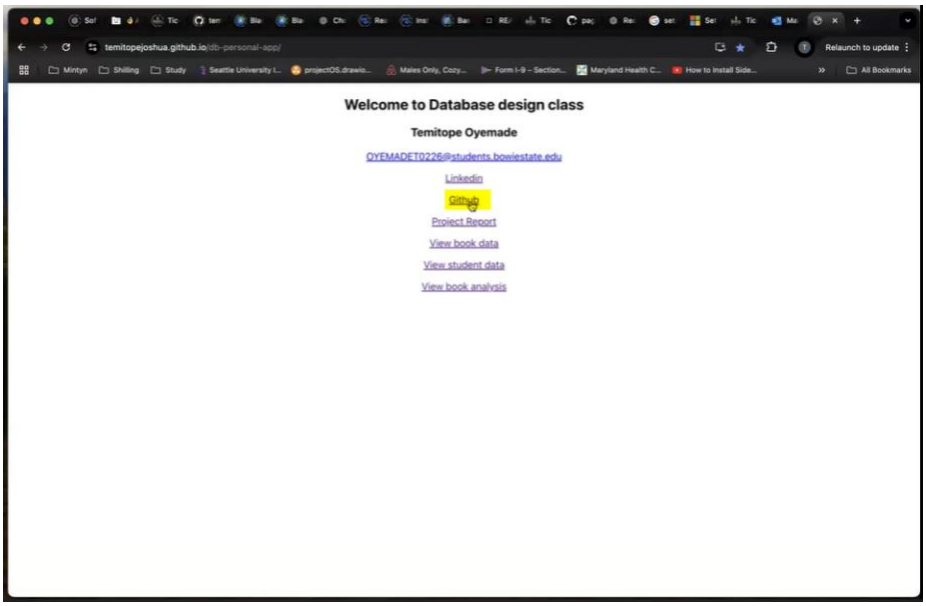
On the front end, Click on “View book analysis” to view the data analysis of book availability.



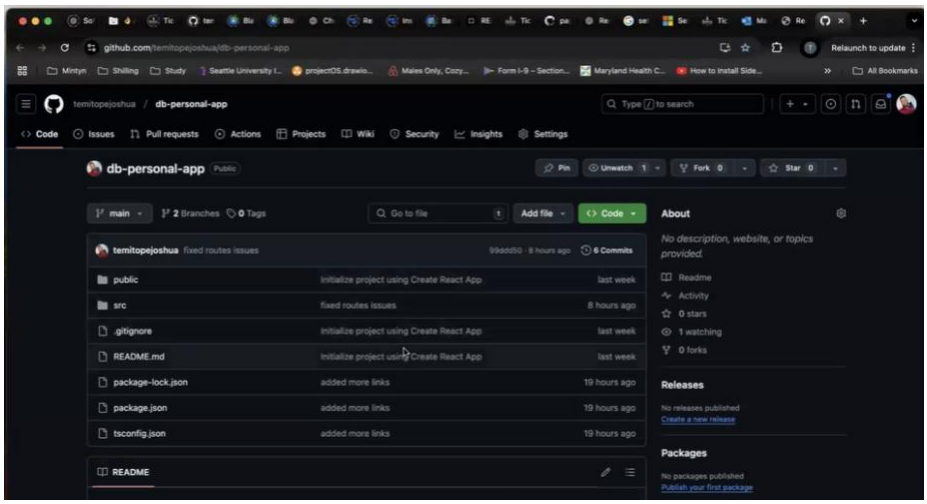
The pie chart shows a quick view of the percentage of books that are available and unavailable.



On the website, Click on “GitHub” to view the GitHub Page.



Website displaying GitHub Page with all the project packages and codes



5.3. Tools and Technologies

- Programming Language:
    - Java: Chosen for its platform independence, strong database connectivity, and robust features, Java was used to build the core application logic for interactive database management.
- Backend Framework:
    - Node.js: Utilized for its lightweight, event-driven architecture, Node.js efficiently handled concurrent requests and facilitated dynamic interactions between the client and the database.
- ORM (Object-Relational Mapping):
    - Sequelize: This ORM simplified database operations with MySQL, providing an abstraction layer over raw SQL queries. This reduced development time, enabled clean code, and improved maintainability.
- Database:
    - MySQL: Selected for its reliability, scalability, and support for SQL standards, MySQL efficiently managed the relational data needed for the project. Its robust querying capabilities allowed seamless handling of borrowing and returning operations.
- Web Hosting:
    - GitHub Pages: Used to host project documentation and static resources, ensuring easy public access, version control, and collaborative development.
- Frontend Framework:
    - React with HashRouter: React enabled the development of a dynamic and responsive user interface, while HashRouter provided seamless navigation with GitHub.

6. System Features

6.1. Borrowing Management

- Specify the name of the student.
- Display borrowed books, borrower details, and return dates.
- Dynamically update the borrowing and return status.

6.2. Rating Analysis

- Show books rated higher or lower than the average rating.
- Allow users to input new ratings to update average dynamically.

6.3. Group Management

- Display students in groups (A, B, C, D).
- Show books borrowed by students in each group.
- Allow querying of a student's group based on their name.

7. Conclusion and Recommendations

The Library Book Borrowing Management System successfully manages book inventory, student assignments, and borrowing activities while enabling meaningful queries and reports. Future improvements could include automated notifications for due dates and overdue penalties.

Appendices

Source Code Repository Link: [https://github.com/temitopejoshua/db-personal-app]

Github Page: https://temitopejoshua.github.io/db-personal-app/