PROJECT REPORT FOR SOFTWARE ENGINEERING

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY IN
COMPUTER SCIENCE
ENGINEERING
BY

Raaina Whalla (224083)
Priya Jyot (224082)
Preti Sharma (224081)

SUBMITTED TO

DR. SONIKA JINDAL

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
SHAHEED BHAGAT SINGH STATE UNIVERSITY FEROZEPUR,
152004 (PUNJAB)

Table OF CONTENTS

Introduction

A Library Management System (LMS) is a software application that simplifies and automates the operations of libraries. It is a complete system for managing library duties such as purchases, member management, monitoring, storing, and circulation. The primary objective of an LMS is to properly organize and manage the resources available in a library, making it easier for librarians to conduct everyday operations and create a user-friendly experience for users.

1.1 **Problem Statement**

Managing a library's resources efficiently is crucial for ensuring smooth operations and user satisfaction. Traditional methods of library management, such as manual record-keeping, are often prone to errors, time-consuming, and lead to inefficiencies like misplaced books, delays in updating inventory, and difficulties in tracking issued or overdue books. Users frequently face challenges in locating desired books, checking their availability, or renewing borrowed items. Similarly, librarians struggle with resource mismanagement due to the lack of a centralized system, making it hard to monitor book inventory, track overdue returns, or generate meaningful reports for decision-making. To address these challenges, a Library Management System aims to digitize library operations and provide an efficient, user-friendly platform for managing book inventory, automating workflows, and enhancing user access. This system will enable centralized tracking of books, automated notifications for due dates, and advanced search functionalities .

1.2 Solution:

A Library Management System (LMS) is a software solution designed to automate and streamline library operations, addressing inefficiencies in manual management. The system provides a centralized database for cataloging books, enabling quick search and real-time availability updates. It simplifies user account management by tracking borrowing history, overdue books, and fines, with automated notifications for due dates and penalties. Additionally, the LMS allows users to reserve books online, while librarians benefit from inventory tracking and report generation for better resource management. With features like barcode integration, secure access controls, and scalability, the LMS enhances the user experience, minimizes human errors, and ensures efficient library operations.

1.3 **Aims and Objectives**:

The objective of the Library Management System (LMS) project is to design and implement an efficient and user-friendly system that automates the various tasks associated with managing a library

Aims:

- **Streamline Library Operations**: Simplify and automate the processes of borrowing, returning, and cataloging books.
- **Enhance User Experience**: Provide users with an easy-to-use platform to search for,reserve and manage books.
- **Improve Efficiency**: Minimize manual errors and reduce the workload for library staff.
- **Enable Digital Transformation**: Transition from traditional methods to a more modern, digital system.

Objectives:

1. **Centralized Database**: Maintain a centralized database for storing book details, user information, and transaction history.
2. **Real-Time Tracking**: Enable real-time tracking of book availability and user activities.
3. **Search and Filter Options**: Allow users to search books by title, author.
4. **User Management**: Manage different user roles (e.g., students, staff, librarian) with appropriate access rights.
5. **Notification System**: Send reminders for due dates, overdue fines, and new book arrivals.
6. **Report Generatio1.4n**: Provide detailed reports on library usage, book circulation, and inventory management.
7. **Secure Access**: Implement secure login mechanisms for both users and administrators.
8. **Scalability**: Design the system to accommodate future growth in terms of users and library resources.

1.4 Target Audience:

The **target audience** of a project refers to the specific group of people or entities that the project is designed to serve or address. Here's how to identify and define the target audience based on different contexts:

For a Library Management System (LMS) Project:

1. **Students**:
   - Need a simple way to search, borrow, and return books.
   - Require reminders for due dates and late returns.
2. **Teachers and Academic Staff**:
   - Require access to academic resources, references, and research materials.
   - May need extended borrowing privileges or specialized content.
3. **Librarians and Library Staff**:
   - Use the system for book cataloging, inventory management, and user administration.
   - Require tools for generating reports and tracking library usage.

2. System Features:

1. User Management

- **User Roles**: Define roles such as students, teachers, librarians, and administrators.
- **User Registration and Login**: Secure login system with unique credentials.
- **Profile Management**: Allow users to update personal information and view borrowing history.

2. Book Management

- **Catalog Management**: Add, update, and delete book records.
- **Categorization**: Organize books by categories like genre, author, or subject.

- **Book Availability**: Display real-time availability status (available, reserved, issued)

3. Search and Filter

- **Search Functionality**: Enable users to search books by title, author, genre,

4. Borrowing and Returning

- **Borrowing System**: Track borrowed books with issue and return dates.
- **Renewal Option**: Allow users to extend the borrowing period if applicable.
- **Overdue Notifications**: Notify users about overdue books and penalties

5. Notifications

- **Email/SMS Alerts**: Send reminders for due dates, overdue fines, or reserved book availability.
- **Event Notifications**: Inform users about library events or updates.

6. Reports and Analytics

- **Usage Reports**: Generate reports on the most borrowed books, user activity, and library performance.
- **Inventory Reports**: Track the total number of books, damaged/lost items, and procurement needs.

7. Security and Access Control

- **Role-Based Access**: Restrict access to specific features based on user roles.
- **Data Encryption**: Protect user and book data from unauthorized access.

8. Librarian Features

- **System Configuration**: Allow librarians to manage system settings like fine policies.
- **Audit Logs**: Keep track of all system activities for accountability.

3. Feasibility Study

A feasibility study is a systematic analysis to determine whether a proposed project is viable. For a Library Management System (LMS), this study typically encompasses technical, operational, economic, and scheduling feasibility. Below is a detailed breakdown:

3.1 Technical Feasibility

- **Required Technology:**
  - Modern programming languages and tools like **REACT** can be used for the front-end.
  - Back-end technologies like **Node.js, Express.js**, and databases like **MongoDB** provide scalability and reliability.

- **Expertise Availability:**

  Developers, IT staff, and maintenance teams with relevant skills must be available to build and sustain the system.

Conclusion: LMS is technically feasible if the organization has access to the required technologies and expertise.

3.2. Operational Feasibility

- **User Requirements:**
  - The system must fulfill the needs of library staff and users by streamlining book cataloging, issuing, returns, and tracking overdue items.
- **Ease of Use:**
  - User-friendly interfaces should minimize training needs and enhance adoption.
- **Workflow Impact:**
  - The system should seamlessly integrate into the existing library processes, reducing manual efforts.

Conclusion: If the system enhances efficiency and is intuitive for users, it is operationally feasible.

3. 3 Economic Feasibility

- **Cost Estimation:**
  - **Development Costs:** Salary of developers, cost of software tools, and initial hardware setup.
  - **Operational Costs:** Hosting, maintenance, and potential updates.
  - **Savings:** Reductions in labor costs, paper usage, and error correction costs.
- **Benefits:**
  - Increased efficiency, better resource tracking, and improved user satisfaction.

- Revenue opportunities through features like paid membership or overdue fines tracking.

Conclusion: If the benefits outweigh the costs in the long run, the project is economically feasible.

4. Requirement Gathering:

Requirement gathering is a critical phase of a Library Management System (LMS) project, involving stakeholders and system users to identify and document the necessary functionalities, features, and constraints. Below is a structured outline of the requirements for an LMS:

1. Stakeholders Identification

- **Librarians/Library Staff:**
  - Manage books, members, and administrative tasks.
- **Students/Readers:**
  - Borrow, return, and search for books.
- **Library Administrators:**
  - Monitor system usage and generate reports.

5. Software Requirement Specification

A **Software Requirements Specification (SRS)** document outlines the functional and non-functional requirements, tools, and other key aspects of a software project. For your **Library Management System (LMS)**, here's a detailed breakdown of what to include:

5.1 Tools and technologies used:

Frontend (React + Tailwind CSS)

1. **UI Design and Styling**
   - **React:** React is a JavaScript library for building user interfaces (UIs). It was developed by Facebook and is widely used for creating dynamic, interactive, and efficient web applications. React is component-based, which allows developers to break down the UI into reusable and manageable pieces.
   - **Tailwind CSS**: **Tailwind CSS** is a **utility-first CSS framework** that provides a set of pre-defined classes to build modern, responsive, and

highly customizable user interfaces directly in your HTML. Unlike traditional CSS frameworks (e.g., Bootstrap), Tailwind focuses on utilities rather than predefined components, giving developers more control over the design.

- **DaisyUI:** **DaisyUI** is a **Tailwind CSS component library** that simplifies the process of building user interfaces by providing pre-designed and customizable components. It is built on top of **Tailwind CSS**, so you can still use Tailwind's utility-first classes while taking advantage of DaisyUI's ready-made components.

  .
- **React Hook Form**: For efficient form handling with validation.

2. **Testing**
   - **React Testing Library**: For UI testing with a focus on user interactions.
3. **Routing**
   - **React Router**: For client-side routing and navigation between pages (e.g., home, search, dashboard).

Backend Framework

- **Node.js: Node.js** is an open-source, cross-platform, server-side runtime environment that enables developers to build scalable and high-performance applications using JavaScript. It is built on Google Chrome's **V8 JavaScript engine**, which allows it to execute JavaScript code outside of a browser.

- **Express.js:** Express.js is a fast, minimal, and flexible web application framework for Node.js. It simplifies the process of building server-side applications by providing a robust set of features to handle HTTP requests, routing, middleware, and more. It is one of the most popular frameworks for Node.js, often referred to as the "de facto standard" for building web applications and APIs.

Database:

A **database** is an organized collection of data that is stored, managed, and retrieved electronically. It allows users and applications to efficiently store and access data in a structured way, often using a system known as a **Database Management System (DBMS)**. Databases are essential for modern applications, enabling dynamic content and functionality like user management, transactions, and analytics. The data base used in this project is Mongodb.

- **Mongodb**: MongoDB is a popular NoSQL database designed for storing and managing large volumes of unstructured or semi-structured data. Unlike traditional relational databases (RDBMS), MongoDB uses a document-oriented model, storing data in flexible, JSON-like structures called documents. This allows it to handle a wide variety of data types and adapt to evolving application needs without requiring a predefined schema.

5.2 Functional Requirements

These define what the system should do:

1 User Management

- Registration and authentication for librarians, students, and administrators.
- Roles and permissions:
  - Librarians can manage books and transactions.
  - Readers can search, reserve, and borrow books.

2 Book Management

- Add, update, or delete book details (title, author etc.).
- Manage copies of books to track inventory.

3 Book Search and Reservation

- Advanced search using filters like title, author..
- Display book availability (available/borrowed/reserved).
- Allow users to reserve books for pickup.

4 Issuing and Returning

- Set due dates and fines for overdue returns.
- Notifications for upcoming return deadlines.

5 Reporting and Analytics

- Generate reports on:
  - Issued/returned books.
  - Popular books.
  - Late returns and fines collected.
- Monitor usage trends.

6 Notification System

- Send email/SMS notifications for:

    - Reservation status updates.
    - Return reminders.
    - Overdue fines.

7 Security Features

- Data encryption for sensitive information.
- Secure login with multi-factor authentication (if needed).
- Role-based access control to restrict unauthorized actions.

5.3. Non-functional requirements:-

The definition for a non-functional requirement is that it essentially specifes how the system should behave and that it is a constraint upon the systems behavior. One could abo think of non- farctional requirements as quality attributes for of a system

1.Product requirements:-
Requirements which specify that the delivered product must behave in a particular way, eg execution speed, reliability etc.

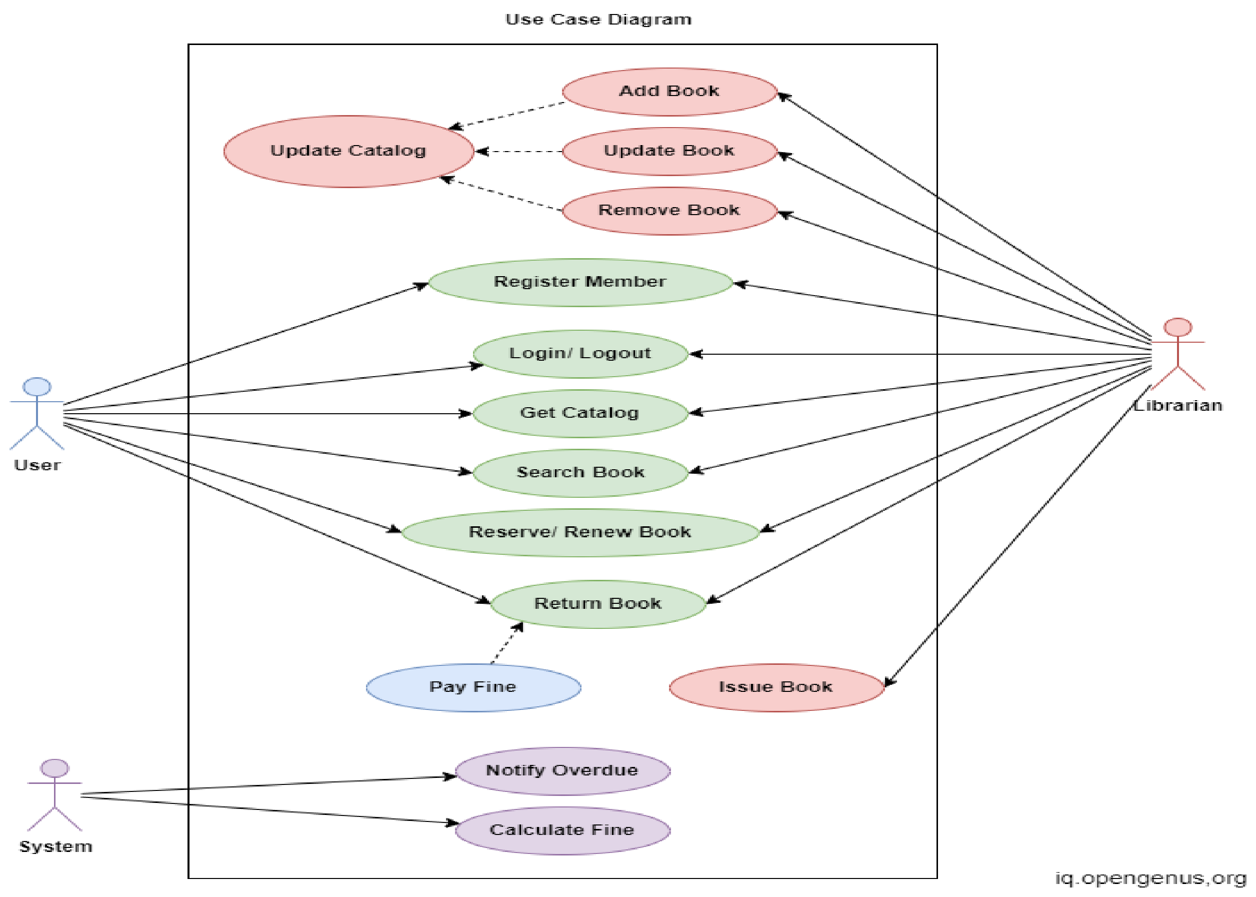
2.Availability Requirement:-
The system is avaibble 100% for the user and is used 24 hrs a day and 365 days year. The system shall be operational 24 hours a day and 7 days a week.

3. Efficiency Requirement:-
Mean Time to Repair (MTTR) Even if the system fuis, the system will be recovered back up within an hour or less.

4.Accuracy:-
The system should accurately provide real time information taking into consideration Various concurrency issues. The system shall provide 100% access rehability.

5.Reliability Requirement:-
The system has to be 100% rehable due to the importance of data and the damages that can be caused by incorrect or incomplete data. The sysaem will nan 7 days a week, 24 hours .

5.4 Use Case Diagram for Library management System
Explaination of Use Case diagram



Here, we will understand the designing use case diagram for the library management system. Some scenarios of the system are as follows :

1. User who registers himself as a new user initially is regarded as staff or student for the library system.
    - For the user to get registered as a new user, registration forms are available that is needed to be fulfilled by the user.
    - After registration, a library card is issued to the user by the librarian. On the library card, an ID is assigned to cardholder or user.
2. After getting the library card, a new book is requested by the user as per there requirement.
3. After, requesting, the desired book or the requested book is reserved by the user that means no other user can request for that book.
4. Now, the user can renew a book that means the user can get a new due date for the desired book if the user has renewed them.
5. If the user somehow forgets to return the book before the due date, then the user pays fine. Or if the user forgets to renew the book till the due date, then the book will be overdue and the user pays fine.

6. User can fill the feedback form available if they want to.
7. Librarian has a key role in this system. Librarian adds the records in the library database about each student or user every time issuing the book or returning the book, or paying fine.
8. Librarian also deletes the record of a particular student if the student leaves the college or passed out from the college. If the book no longer exists in the library, then the record of the particular book is also deleted.
9. Updating database is the important role of Librarian.

6. System Design OF Library Management System

The system design of a Library Management System (LMS) involves planning and organizing the architecture, components, and workflows required for managing library resources and services efficiently. Below is a high-level breakdown of the system design:

1.Architecture

- **Presentation Layer**: User Interface (web app, mobile app.
- **Database Layer**: Persistent storage of books, members and logs.

System Components:

- **Frontend**: React for UI.
- **Backend**: Node.js with Express.js for handling APIs.
- **Database**:
  - NoSQL: MongoDB (for flexibility in storing unstructured data like book categories).
- **Search Engine**: Elasticsearch (for fast book searches).
- **Authentication**: OAuth, JWT for secure login and role-based access.
- **Notification Service**: Email/SMS notifications for overdue books.

2.Logical Design:

The logical design of a Library Management System refers to the abstract representation of the system's structure, focusing on the relationships and flow of data between components without concerning itself with the technical implementation or physical deployment. It is primarily concerned with *what* the system does rather than *how* it does it

Key Components of a Logical Design for a Library Management System:

1. **Entities and Attributes**:
   - Define the main components of the system and their properties.
   - Examples:
     - **Book**: Book_ID, Title, Author, Publisher, ISBN, Genre, Availability_Status.
     - **User**: User_ID, Name, Contact_Details
     - **Librarian**: Librarian_ID, Name, Contact_Details.
     - **Transaction**: Transaction_ID, Book_ID, User_ID, Issue_Date, Due_Date, Return_Date.
2. **Relationships**:
   - Illustrate how entities interact with one another.
   - Examples:
     - A **User** can borrow multiple **Books**.
     - A **Book** can be issued to only one **User** at a time.
     - A **Librarian** manages transactions related to **Books** and **Users**.
3. **Processes**:
   - Detail the major operations performed by the system.
   - Examples:
     - **Book Issuance**: When a user borrows a book, the system checks availability, logs the transaction, and updates the status of the book.
     - **Book Return**: When a user returns a book, the system calculates fines (if applicable), updates the book's status, and logs the transaction.
     - **Catalog Management**: Adding, updating, or removing books from the system.
     - **User Management**: Adding, updating, or deleting user profiles.

7.2.1 E-R diagram For Library Management System

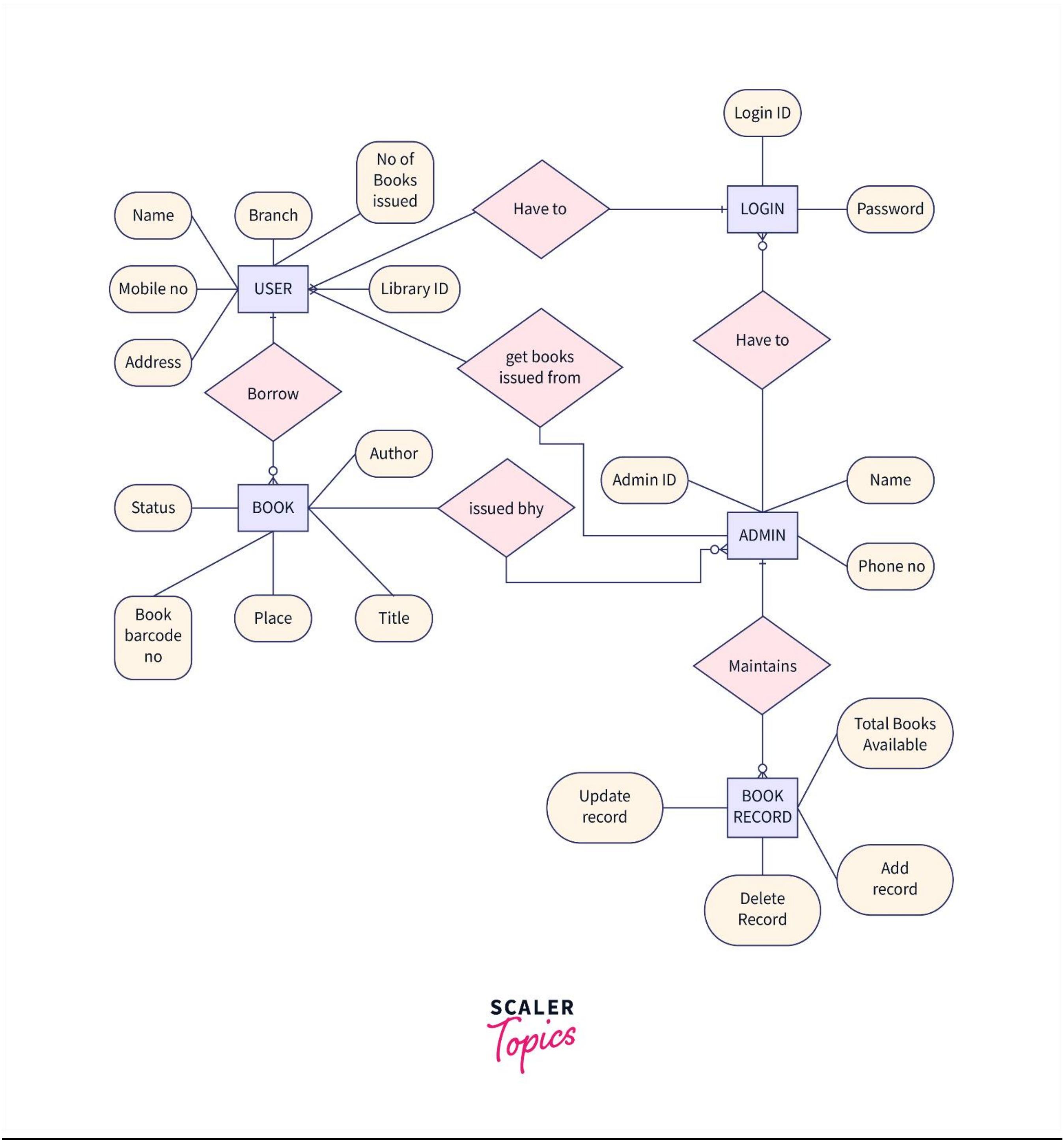
An **Entity-Relationship (ER) Diagram** is a graphical representation of the logical structure of a database. It is used to model the entities (objects), the relationships between them, and the attributes of the entities, providing a high-level blueprint for database design.

Components OF E-R Diagram

1. **Entities:**
   - Definition: Objects or concepts that represent data in the system.
   - Representation: Rectangles.
   - Types:
     - Strong Entity: Can exist independently (e.g., Book).
     - Weak Entity: Relies on a strong entity (e.g., Transaction).
2. **Attributes:**
   - Definition: Properties or characteristics of an entity or relationship.
   - Representation: Ovals connected to entities/relationships.
   - Types:
     - Key Attribute: Uniquely identifies an entity (e.g., User_ID).
     - Derived Attribute: Can be calculated (e.g., Fine).
     - Multivalued Attribute: Can have multiple values (e.g., Genres).
3. **Relationships:**
   - Definition: Associations between entities.
   - Representation: Diamonds.
   - Types:
     - One-to-One (1:1), One-to-Many (1:N), Many-to-Many (M:N).
4. **Primary Key**:
   - Unique identifier for each entity instance.
   - Example: Book_ID for Book.
5. **Foreign Key:**
   - Links one entity to another.
   - Example: User_ID in Transaction refers to User.
6. **Cardinality:**
   - Defines the number of instances of one entity related to another.
   - Types: 1:1, 1:N, M:N.

These components together create a visual representation of a system’s data structure and relationships, aiding in database design and development.

E-R Diagram For Library Management System



Explaination Of E-r Diagram

ER diagram of a Library Management System contains various entities and their attributes, which explains the overall structure and functioning of a library and ultimately helps in designing its database. A Library

Management System database keeps track of the following considerations –

- Both the Admin and users can log in to access the permitted functionalities in a Library database.
- The Admin maintains the book record by updating the availability status of books and can add or delete book records.
- Every Book has its own title, Price, availability status, author, and other details.
- Every user is registered with their library_id, name (first name, last name), Mobile No, email, address, and the number of books issued. The Admin keeps track of all the users.
- Every Admin is registered with their **admin_id, name (first name, last name), Mobile No, email, and address**.
- Users can issue and return books corresponding to which, they will have an issue date and return date respectively. A user can get the books issued from the Admin after successful login.

7.2.2 Data Flow Diagram For library Management System

A **Data Flow Diagram (DFD)** is a graphical representation of the flow of data within a system. It shows how data moves between processes, data stores, external entities, and the system itself. A DFD helps visualize how a system handles and processes data, making it easier to understand and analyze the system's functionality.

Components of a Data Flow Diagram:

1. **Processes**:
   - Represent actions or operations that transform data.
   - **Symbol**: Circles or rounded rectangles.
   - **Example**: Issue Book, Validate User.
2. **Data Flows**:
   - Represent the movement of data between processes, data stores, and external entities.
   - **Symbol**: Arrows.
   - **Example**: User Request, Book Details.
3. **Data Stores**:
   - Represent where data is stored for later use.

- **Symbol**: Open rectangles or parallel lines.
- **Example**: Books Database, Users Database.

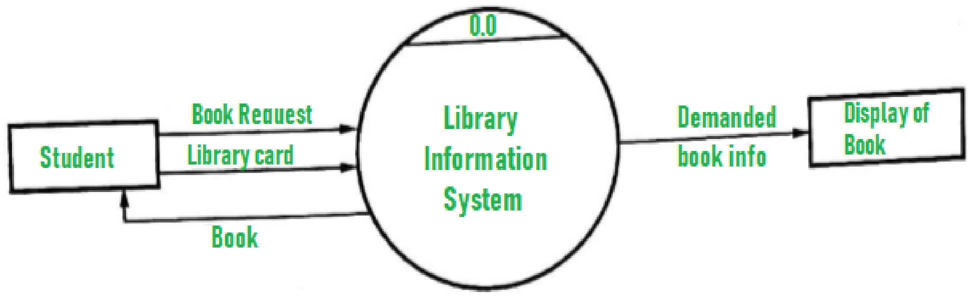
4. **External Entities**:
   - Represent sources or destinations of data outside the system.
   - **Symbol**: Squares or rectangles.
   - **Example**: Library Member, Published

Various Levels Of DFD

Level 0 Data Flow Diagram (DFD)

Level 0 is the highest-level Data Flow Diagram (DFD), which provides an overview of the entire system. It shows the major processes, data flows, and data stores in the system, without providing any details about the internal workings of these processes.

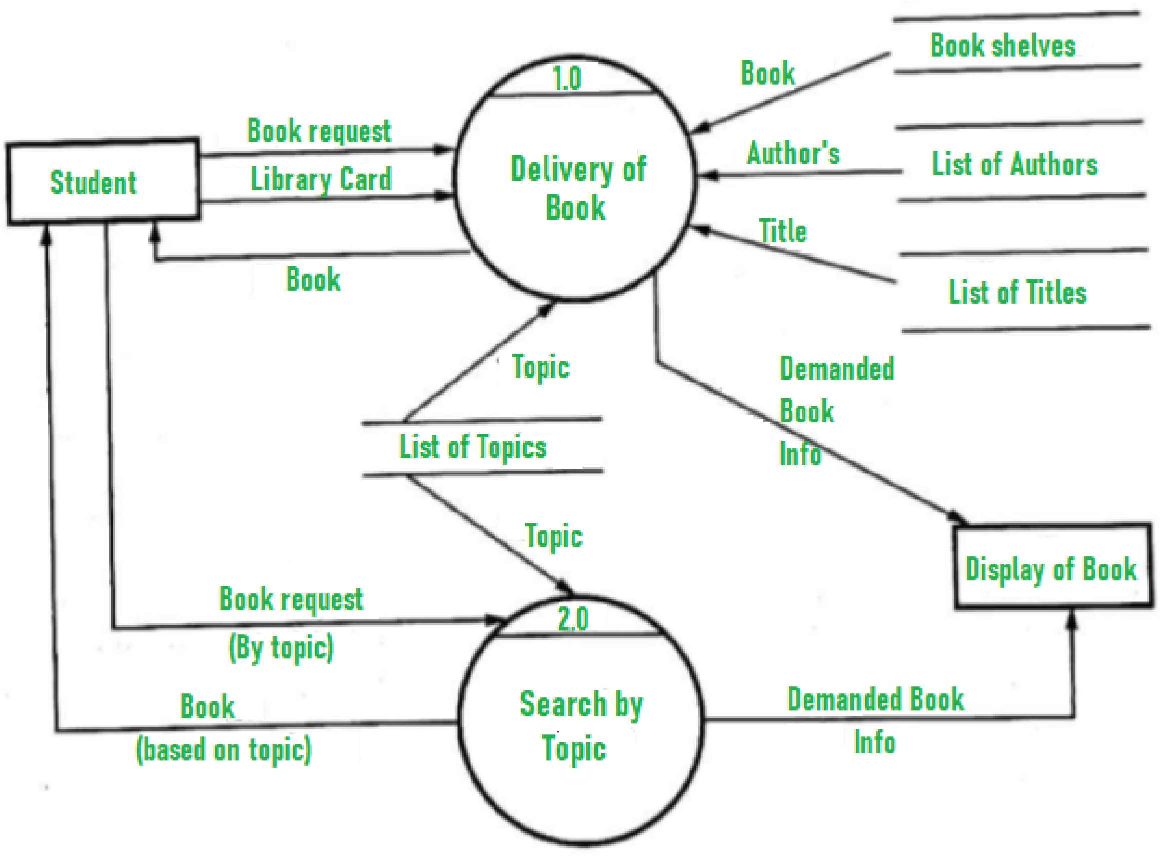
It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.



1-Level Data Flow Diagram (DFD)

1-Level provides a more detailed view of the system by breaking down the major processes identified in the level 0 Data Flow Diagram (DFD) into sub-processes. Each sub-process is depicted as a separate process on the level 1 Data Flow Diagram (DFD). The data flows and data stores associated with each sub-process are also shown.

In 1-level Data Flow Diagram (DFD), the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level Data Flow Diagram (DFD) into subprocesses.

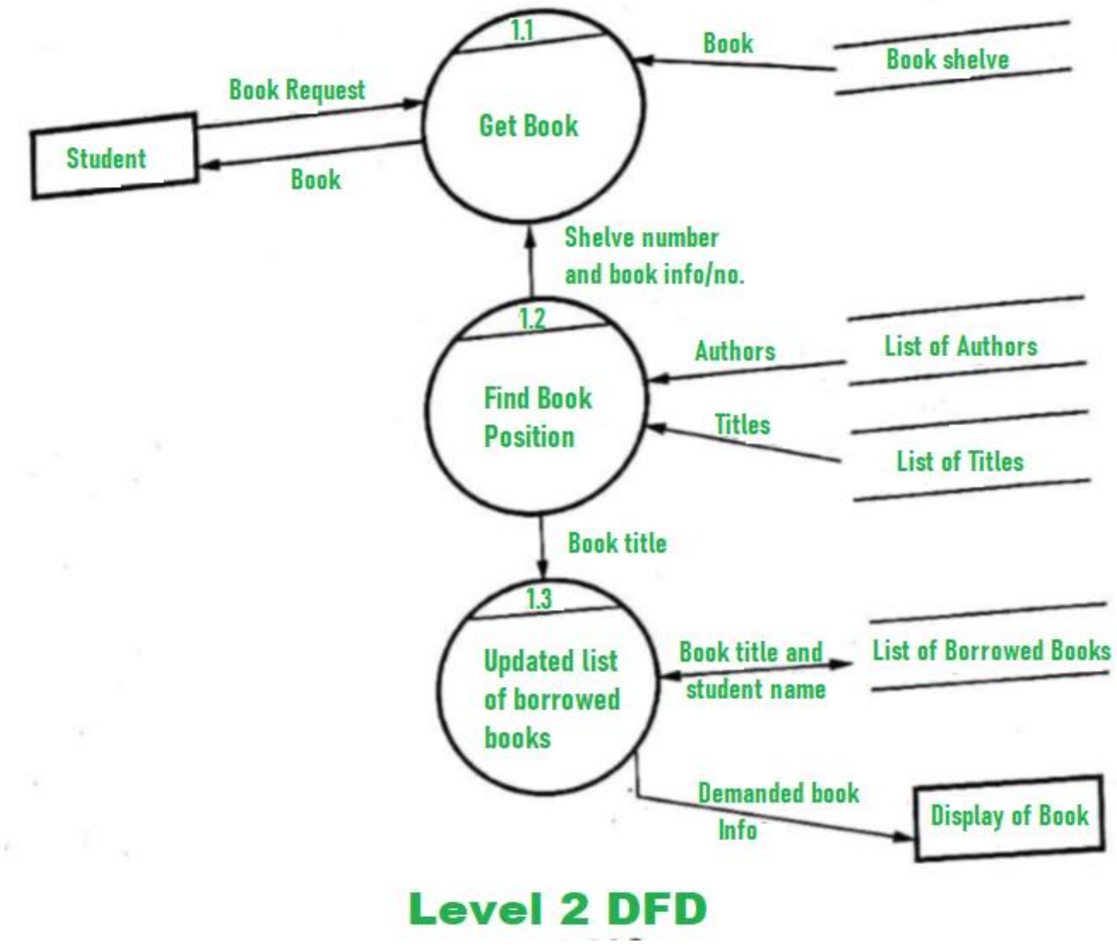


Level 1 DFD

2-Level Data Flow Diagram (DFD)

2-Level provides an even more detailed view of the system by breaking down the sub-processes identified in the level 1 Data Flow Diagram (DFD) into further sub-processes. Each sub-process is depicted as a separate process on the level 2 DFD. The data flows and data stores associated with each sub-process are also shown.

2-Level Data Flow Diagram (DFD) goes one step deeper into parts of 1-level DFD. It can be used to plan or record the specific/necessary detail about the system's functioning.



8. <u>Coding</u>

<u>Implementing Library Mangement System | Environment Creation:</u>

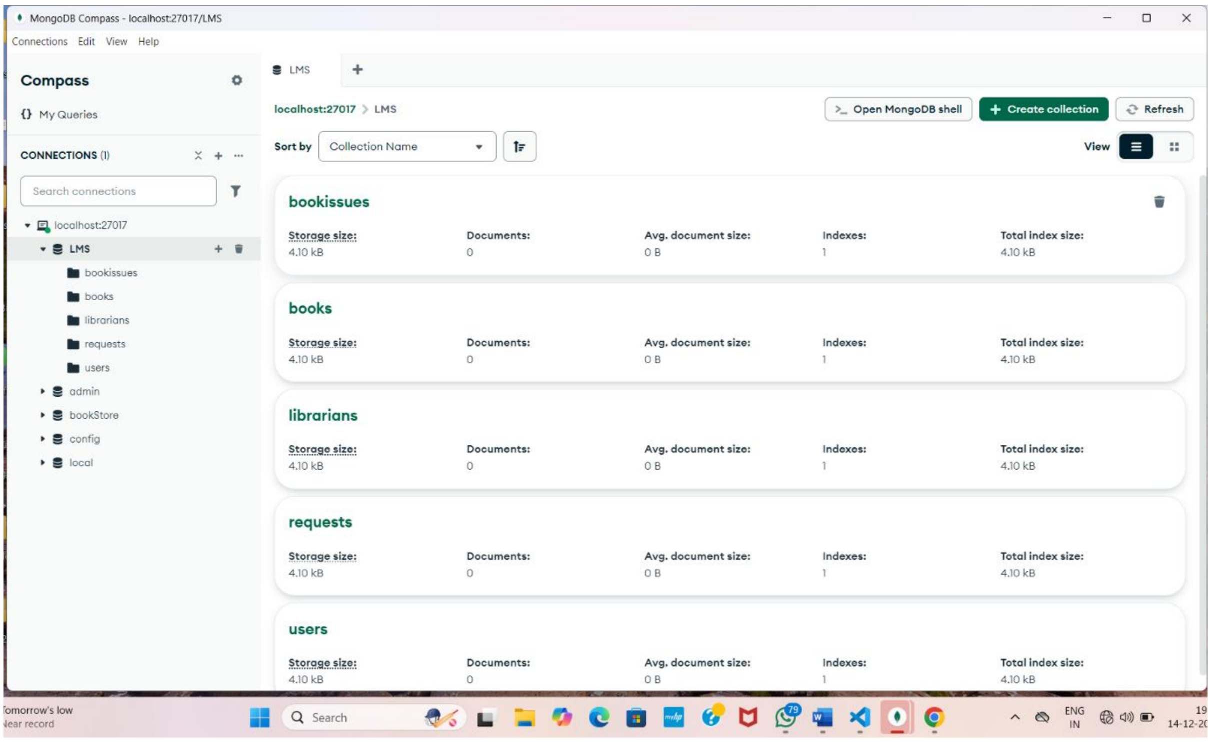
Required Softwares:

- VS Code (you can use any other suitable editor as well)
- Mongodb Compass(For storing data)
- Install Tailwind or download Tailwind extension on vscode.

After we downloaded the above required software now we will start creating our project . In the following article We will discuss about different different modules compiled with same category.
We will discuss it stepwise :

Implementing Library Mangement System | Database Creation:

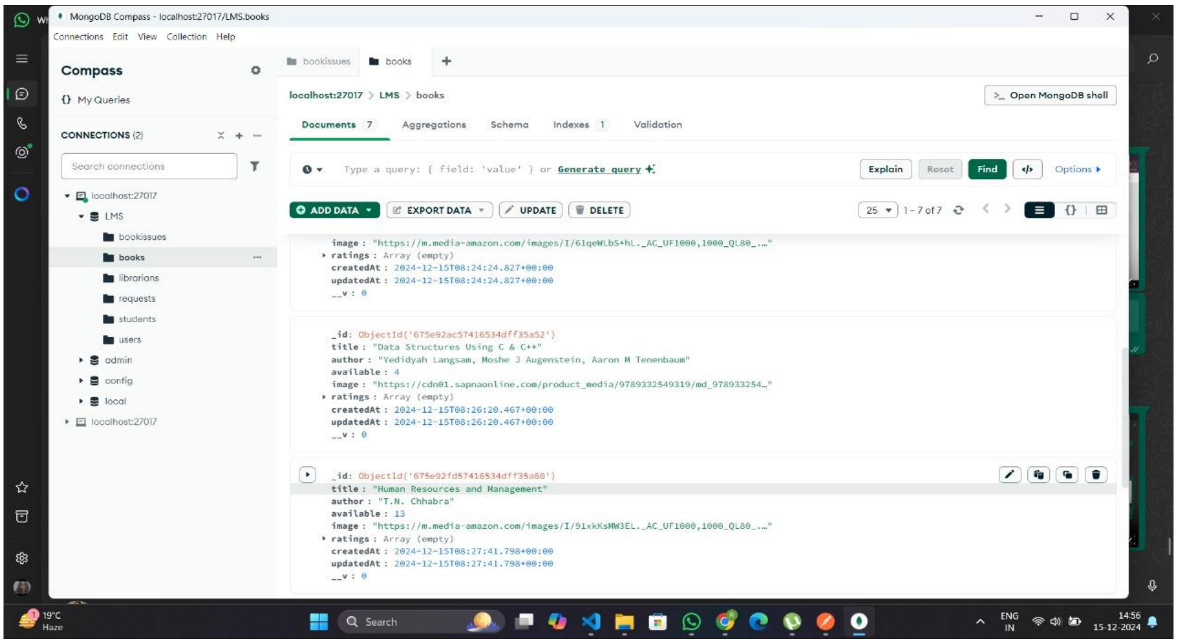
Database Used in this project:



MongoDB Compass - localhost:27017/LMS

localhost:27017 > LMS

	Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
bookissues	4.10 kB	0	0 B	1	4.10 kB
books	4.10 kB	0	0 B	1	4.10 kB
librarians	4.10 kB	0	0 B	1	4.10 kB
requests	4.10 kB	0	0 B	1	4.10 kB
users	4.10 kB	0	0 B	1	4.10 kB



MongoDB Compass - localhost:27017/LMS.books

localhost:27017 > LMS > books

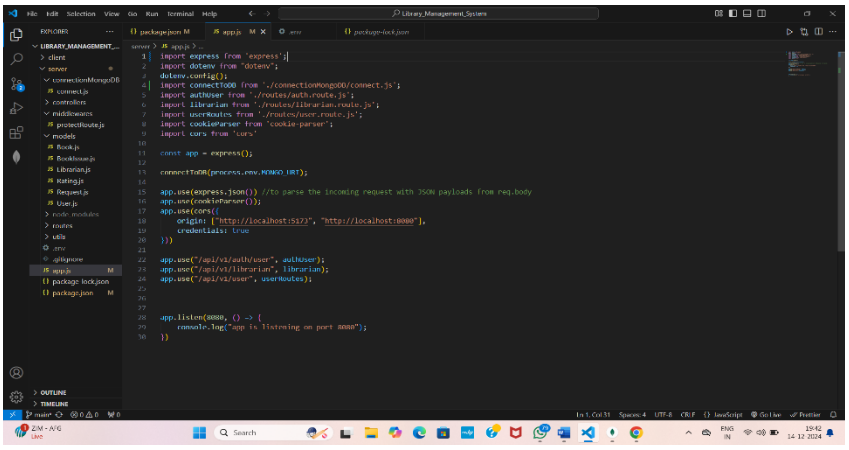
Documents 7 | 1 - 7 of 7

```
image : "https://m.media-amazon.com/images/I/61qeWLbS+hL._AC_UF1000,1000_QL80_..."
ratings : Array (empty)
createdAt : 2024-12-15T08:24:24.827+00:00
updatedAt : 2024-12-15T08:24:24.827+00:00
__v : 0

_id: ObjectId('675e92ac57416534dff35a52')
title : "Data Structures Using C & C++"
author : "Yedidyah Langsam, Moshe J Augenstein, Aaron M Tenenbaum"
available : 4
image : "https://cdn01.sapnaonline.com/product_media/9789332549319/md_978933254..."
ratings : Array (empty)
createdAt : 2024-12-15T08:26:20.467+00:00
updatedAt : 2024-12-15T08:26:20.467+00:00
__v : 0

_id: ObjectId('675e92fd57416534dff35a60')
title : "Human Resources and Management"
author : "T.N. Chhabra"
available : 13
image : "https://m.media-amazon.com/images/I/91xkKsMW3EL._AC_UF1000,1000_QL80_..."
ratings : Array (empty)
createdAt : 2024-12-15T08:27:41.798+00:00
updatedAt : 2024-12-15T08:27:41.798+00:00
__v : 0
```

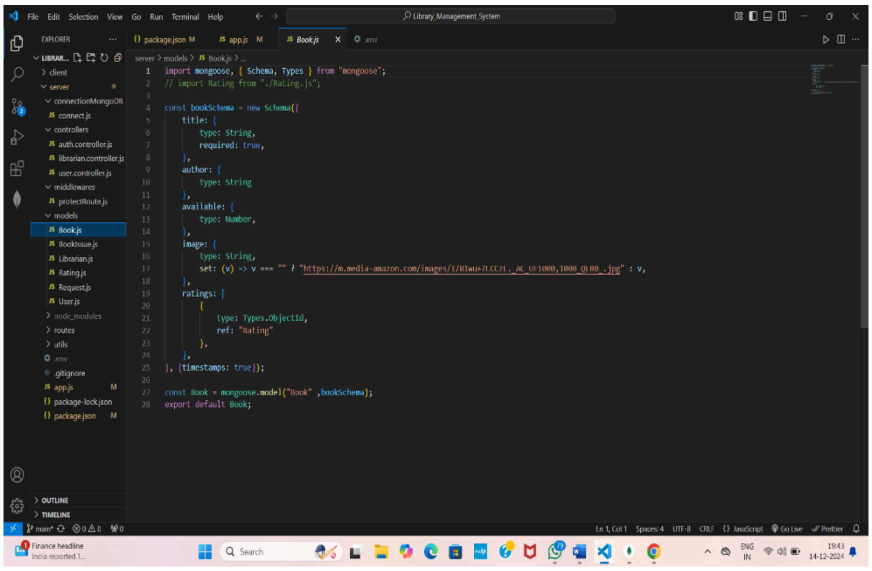
After creating the database we can now start building the frontend of our project.

Implementing Library Mangement System | Frontend and Backend Development:

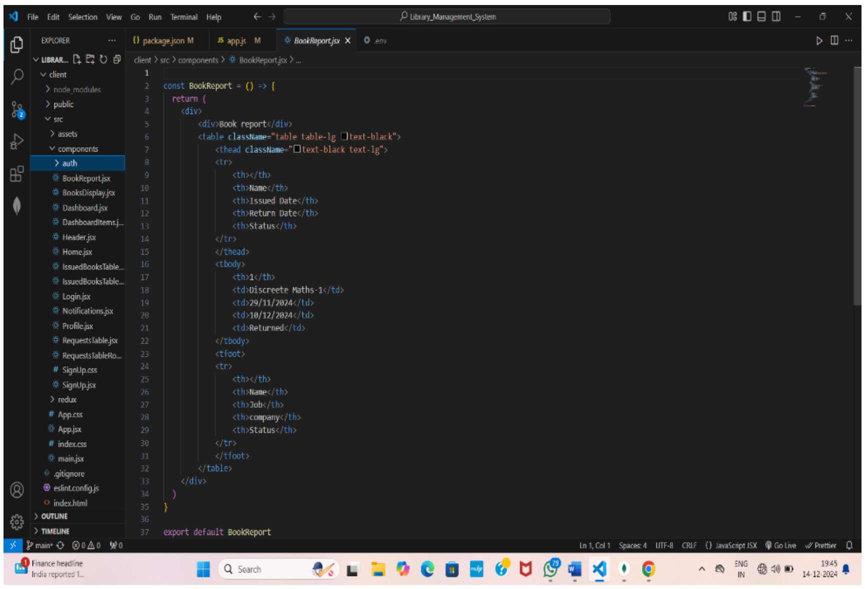
Now we are going to develop our frontend and backend part of the project in different modules.



```
import express from 'express';
import dotenv from "dotenv";
dotenv.config();
import connectToDB from './connectionMongoDB/connect.js';
import authUser from './routes/auth.route.js';
import librarian from './routes/librarian.route.js';
import userRoutes from './routes/user.route.js';
import cookieParser from 'cookie-parser';
import cors from 'cors'

const app = express();

connectToDB(process.env.MONGO_URI);

app.use(express.json()) //to parse the incoming request with JSON payloads from req.body
app.use(cookieParser());
app.use(cors({
    origin: ["http://localhost:5173", "http://localhost:8080"],
    credentials: true
}))

app.use("/api/v1/auth/user", authUser);
app.use("/api/v1/librarian", librarian);
app.use("/api/v1/user", userRoutes);



app.listen(8080, () => {
    console.log("app is listening on port 8080");
})
```

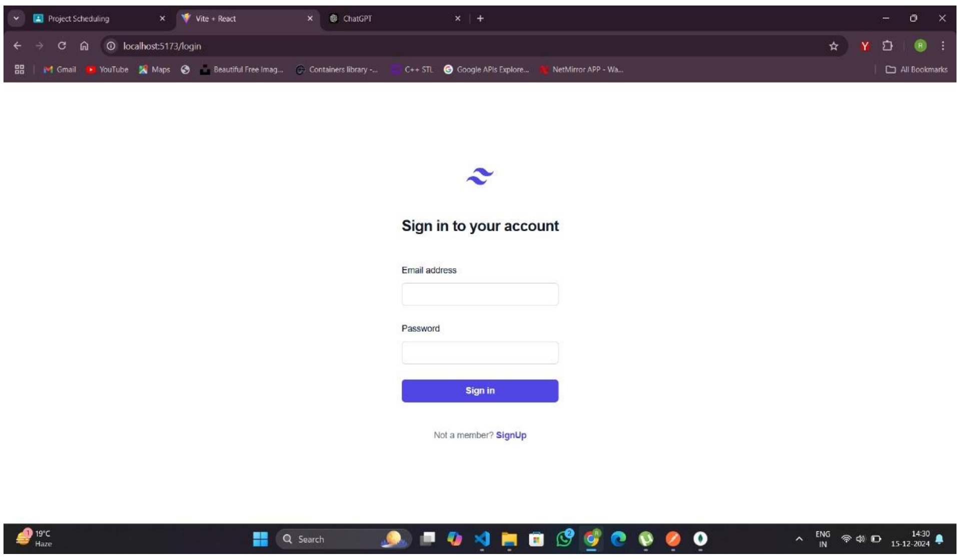


```
import mongoose, { Schema, Types } from "mongoose";
// import Rating from "./Rating.js";

const bookSchema = new Schema({
    title: {
        type: String,
        required: true,
    },
    author: {
        type: String
    },
    available: {
        type: Number,
    },
    image: {
        type: String,
        set: (v) => v === "" ? "https://m.media-amazon.com/images/I/81wu+7LCCzL._AC_UF1000,1000_QL80_.jpg" : v,
    },
    ratings: [
        {
            type: Types.ObjectId,
            ref: "Rating"
        },
    ],
}, {timestamps: true});

const Book = mongoose.model("Book" ,bookSchema);
export default Book;
```



```

const BookReport = () => {
  return (
    <div>
        <div>Book report</div>
        <table className="table table-lg text-black">
            <thead className="text-black text-lg">
            <tr>
                <th></th>
                <th>Name</th>
                <th>Issued Date</th>
                <th>Return Date</th>
                <th>Status</th>
            </tr>
            </thead>
            <tbody>
                <th>1</th>
                <td>Discreete Maths-1</td>
                <td>29/11/2024</td>
                <td>10/12/2024</td>
                <td>Returned</td>
            </tbody>
            <tfoot>
            <tr>
                <th></th>
                <th>Name</th>
                <th>Job</th>
                <th>company</th>
                <th>Status</th>
            </tr>
            </tfoot>
        </table>
    </div>
  )
}

export default BookReport
```

Output Screen :



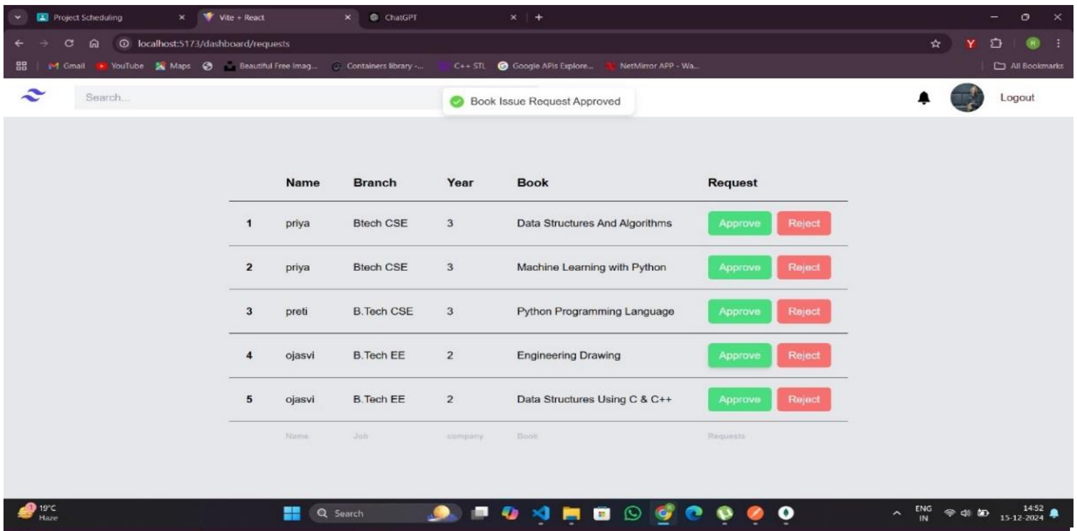
Sign in to your account

Email address

Password

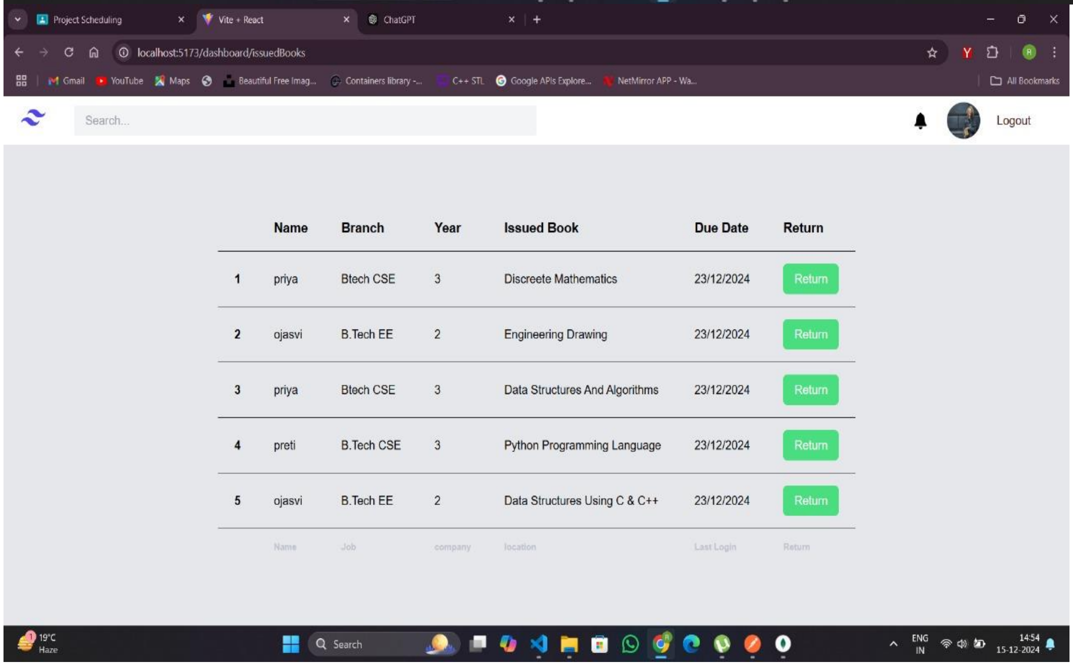
Sign in

Not a member? SignUp

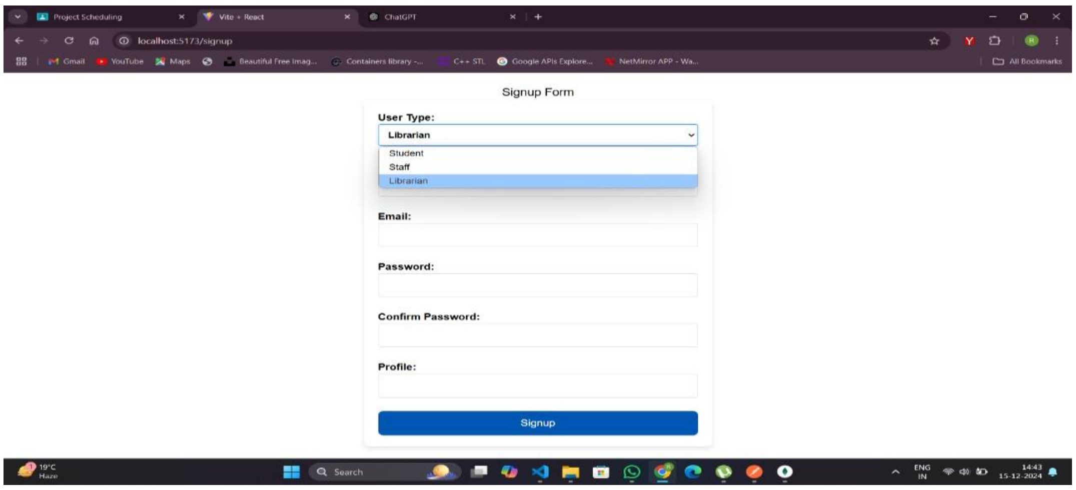


Book Issue Request Approved

	Name	Branch	Year	Book	Request
1	priya	Btech CSE	3	Data Structures And Algorithms	Approve Reject
2	priya	Btech CSE	3	Machine Learning with Python	Approve Reject
3	preti	B.Tech CSE	3	Python Programming Language	Approve Reject
4	ojasvi	B.Tech EE	2	Engineering Drawing	Approve Reject
5	ojasvi	B.Tech EE	2	Data Structures Using C & C++	Approve Reject
	Name	Job	company	Book	Requests



	Name	Branch	Year	Issued Book	Due Date	Return
1	priya	Btech CSE	3	Discreete Mathematics	23/12/2024	Return
2	ojasvi	B.Tech EE	2	Engineering Drawing	23/12/2024	Return
3	priya	Btech CSE	3	Data Structures And Algorithms	23/12/2024	Return
4	preti	B.Tech CSE	3	Python Programming Language	23/12/2024	Return
5	ojasvi	B.Tech EE	2	Data Structures Using C & C++	23/12/2024	Return
	Name	Job	company	location	Last Login	Return


Signup Form

User Type:

Librarian

Student
Staff
Librarian

Email:

Password:

Confirm Password:

Profile:

Signup

9.<u>Testing oF Library Mangement System</u>

Testing is a crucial phase in the development of a library management system (LMS) to ensure that it meets its intended requirements, functions correctly, and is free of bugs. Below are some key steps and considerations for the testing phase of a library management system:

1. Unit Testing:

- Test individual modules or components of the system in isolation to ensure they function as intended.
- Identify and fix any bugs or issues found at the module level.

2. Integration Testing:

- Verify that different modules and components of the LMS work together seamlessly.
- Test data flow and interactions between various parts of the system.

3. Functional Testing:

- Validate that the LMS performs its intended functions accurately and efficiently.
- Test basic functionalities such as adding, updating, and deleting books, managing user accounts, and generating reports.

4. User Interface (UI) Testing:

- Ensure that the user interface is user-friendly, intuitive, and visually appealing.
- Check for consistency in design elements and responsiveness across different devices.

5. Performance Testing:

- Assess the system's performance under normal and peak load conditions.
- Check response times, scalability, and overall system stability.

6. Security Testing:

- Identify and rectify any security vulnerabilities in the system.
- Ensure that user data is handled securely, and unauthorized access is prevented**.**

7. Usability Testing:

- Evaluate the LMS from an end-user perspective to ensure ease of use.
- Gather feedback on user interfaces, navigation, and overall user experience.

8. Compatibility Testing:

- Test the LMS on various browsers, operating systems, and devices to ensure cross-platform compatibility.

9. Regression Testing:

- Conduct tests to ensure that new changes or fixes do not negatively impact existing functionalities.
- Re-run previously executed test cases to verify the overall system stability.