COLLEGE CODE : 9222

COLLEGE NAME : THENI KAMMAVAR SANGAM COLLEGEOFTECHNOLOGY

DEPARTMENT : B.tech(IT)

STUDENT NM-ID : 3971DB2B07CAADD9434475F74066FF6C

ROLL NO : 23it015

DATE : 19-09-2025

Completed the project named as

Phase__ TECHNOLOGY PROJECT

NAME : library book management

SUBMITTED BY,

NAME : JEYASURYA K

MOBILE NO : 6374954151

Library Book Management System

Solution Design & Architecture

1. Tech Stack Selection

Choosing the right technologies ensures scalability, maintainability, and performance. Below is a recommended stack suitable for modern web applications:

- **Frontend:**
  - **Framework:** React.js
    Reason: React offers a component-based architecture, enabling reusable UI components and fast rendering via the virtual DOM.
  - **State Management:** Redux or React Context API
    Reason: To manage global state such as user authentication status, book lists, and issue statuses efficiently.
  - **Styling:** Tailwind CSS / Material-UI
    Reason: Provides utility-first or component-based CSS for rapid UI development and consistent design.
  - **HTTP Client:** Axios or Fetch API
    Reason: For interacting with backend REST APIs asynchronously.
- **Backend:**
  - **Runtime:** Node.js
    Reason: Asynchronous, event-driven environment suitable for I/O-bound operations like database queries.
  - **Framework:** Express.js
    Reason: Lightweight and flexible framework for building RESTful APIs.
  - **Authentication:** JWT (JSON Web Tokens)
    Reason: Stateless, scalable user authentication mechanism.
  - **Database ORM/ODM:** Mongoose (MongoDB) or Sequelize (SQL)
    Reason: Simplifies database operations and schema validation.
- **Database:**
  - **Primary Option:** MongoDB (NoSQL)
    Reason: Flexibility in schema design, easy to handle book metadata with varying fields.
  - **Alternative Option:** PostgreSQL (SQL)
    Reason: Strong relational support, ACID compliance, useful if complex joins are required.
- **DevOps:**
  - **Containerization:** Docker for consistent environments.
  - **CI/CD:** GitHub Actions or Jenkins to automate testing and deployment.
  - **Hosting:** AWS (EC2, RDS), Heroku, or Vercel for scalable cloud deployment.

---

2. UI Structure / API Schema Design

UI Structure

The UI is designed with usability and role-based access in mind.

Page/Component	Description	Access Role
Login/Register	User authentication	All users
Dashboard	Overview of borrowed books, due dates	Member, Librarian
Book Catalog	Search and browse books	All
Book Details	Detailed info with availability status	All
Issue Book	Interface to issue a book	Librarian, Admin
Return Book	Return process interface	Librarian, Admin
Add/Edit Book	Form to manage book entries	Admin
User Management	Manage library users	Admin

API Schema Design

Below is a sample of the core REST API endpoints:

HTTP Method	Endpoint	Description	Auth Required
POST	`/auth/register`	Register a new user	No
POST	`/auth/login`	User login and JWT generation	No
GET	`/books`	Retrieve list of books	Yes
GET	`/books/:id`	Get detailed book info	Yes
POST	`/books`	Add a new book	Admin
PUT	`/books/:id`	Update book info	Admin
DELETE	`/books/:id`	Remove book	Admin
POST	`/books/:id/issue`	Issue a book to a member	Librarian
POST	`/books/:id/return`	Return a borrowed book	Librarian
GET	`/users`	List all users	Admin
PUT	`/users/:id/role`	Change user role (Admin, Member)	Admin

Request & Response example for issuing a book:

- **Request:** `POST /books/12345/issue`
- `{`
- `  "userId": "67890"`
- `}`
- **Response:**
- `{`
- `  "message": "Book issued successfully",`
- `  "dueDate": "2025-10-10"`
- `}`

---

3. Data Handling Approach

- **Frontend State Management:**
  - Use **Redux** to manage complex state including authentication, book inventory, current loans, and notifications.
  - Asynchronous API calls are handled with middleware such as **redux-thunk** or **redux-saga**.
  - UI optimistically updates loan status for instant feedback while backend confirms changes.
- **Backend Data Processing:**
  - **MVC Pattern:**
    - **Models:** Define data schemas (User, Book, Transaction).
    - **Controllers:** Handle business logic (issue/return books, user roles).
    - **Routes:** Map HTTP endpoints to controller functions.
  - Use input validation libraries like **Joi** or **express-validator** to sanitize inputs.
  - Implement **middleware** for authentication, authorization, and error handling.
  - Log every issue/return transaction for auditability.
- **Database Schema Example:**

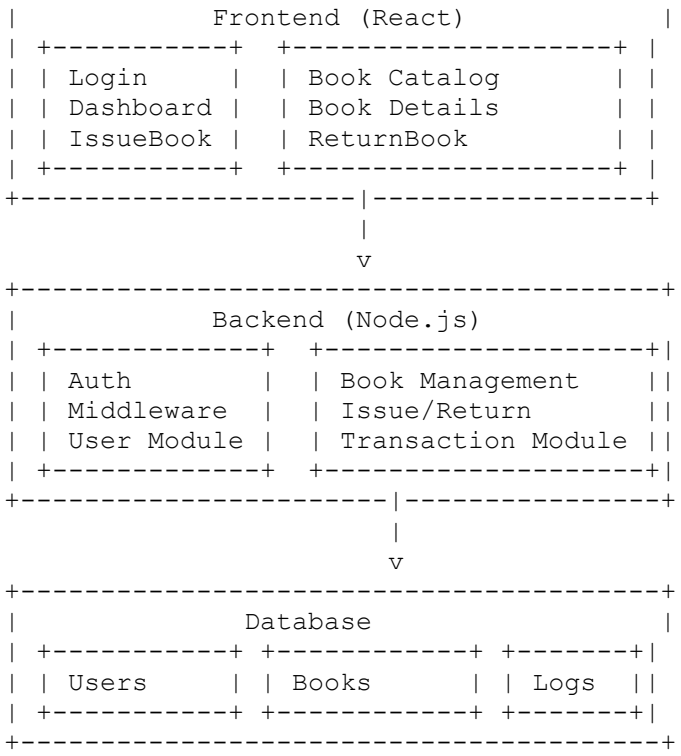
Collection/Table	Key Fields	Description
Users	`userId, name, email, role`	Members, Librarians, Admins
Books	`bookId, title, author, category, availabilityStatus`	Book inventory
Transactions	`transactionId, userId, bookId, issueDate, dueDate, returnDate`	Book issue/return logs

- **Data Consistency:**
  - Transactions are atomic to ensure no double issuing of the same book.
  - Status of book (available/issued) updated synchronously.

---

4. Component / Module Diagram

```
+--------------------------------------+
```



---

5. Basic Flow Diagram