



WEB TECHNOLOGIES

**ASSIGNMENT-2**

**LIBRARAY MANAGEMENT SYSTEM**

**NAME : A. SATHWIKA ROLL NO : 160122749304 SECTION : C5**

# DEPARTMENT : COMPUTER ENGINEERING AND TECHNOLOGY SEMESTER : IV

**SUBMITTED TO : N.SUJATHA GUPTA MAM .**

## INTRODUCTION :

**1. Motivation:**

Libraries play a fundamental role in facilitating knowledge dissemination and academic pursuits within educational institutions and communities. However, traditional library management systems often face significant challenges in terms of efficiency, accessibility, and user experience. The motivation behind developing a modern library management system stems from the need to address these challenges and enhance the overall effectiveness of library operations.

The primary motivations for this project include:

* **Enhancing Efficiency**: Automating manual processes such as book cataloging, inventory management, and user interactions to streamline library operations and improve resource utilization.
* **Improving User Experience**: Providing students and library administrators with a user-friendly platform that enables seamless book search, borrowing, and fine management from any location.
* **Embracing Technology**: Leveraging modern technologies such as MongoDB, Node.js, and HTML/CSS to modernize library services and adapt to evolving user expectations in the digital era.

## SYSTEM SPECIFICATIONS:

The proposed library management system will be developed using a stack of technologies that include MongoDB, Node.js, Express.js, HTML/CSS for front-end development, and Handlebars (hbs) as the template engine for server-side rendering. The system specifications include:

1.Front End Development 2.DataBase Management 3.Backend Development

## Database Management:

* + Utilization of MongoDB for storing and managing library resources, user data, and transactional information.
* Implementation of Mongoose as the ODM (Object Data Modeling) library to define schemas and interact with the MongoDB database.
* Implementation of indexes in MongoDB to optimize query performance and

ensure fast retrieval of data.

* Use of MongoDB's aggregation framework to perform complex data analysis and

reporting.

## Back-End Development:

* + Development of the server-side application using Node.js and Express.js.
  + Creation of RESTful APIs to handle CRUD operations for managing books, user accounts, borrowing transactions, and fine calculations.
  + Integration of authentication middleware to secure routes and manage user sessions.
* Implementation of JWT (JSON Web Tokens) for stateless authentication to

enhance security and scalability.

* Use of Express.js middleware to handle errors, log activities, and parse incoming

requests.

## Front-End Development:

* + Implementation of responsive and user-friendly interfaces using HTML, CSS, and JavaScript.
  + Use of Handlebars (hbs) for server-side rendering of dynamic content based on data retrieved from the back-end.
* Utilization of CSS frameworks like Bootstrap or Tailwind CSS to ensure consistent

design and improve development efficiency.

## Front-End Development :

The front-end of the library management system is developed using HTML, CSS, and JavaScript. Handlebars (hbs) is utilized as the template engine for server-side rendering, allowing dynamic content generation based on data retrieved from the back-end. AJAX (Asynchronous JavaScript and XML) is leveraged for asynchronous data retrieval, enhancing the responsiveness of the user interface.

## HTML CODE:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<link rel="stylesheet" href="adminstd.css">

<link href='https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css' rel='stylesheet'>

</head>

<body>

<div class="ad1">

<form action="/adminstd" method="post">

<div class="ad">

<h1>Admin Login</h1>

<label>Username</label>

<br>

<input type="text" name="name" required>

<br>

<label>Password</label>

<br>

<input type="password" name="password" required>

<br>

<div id="adm-Msg">{{ admMsg }}</div>

<button>login</button>

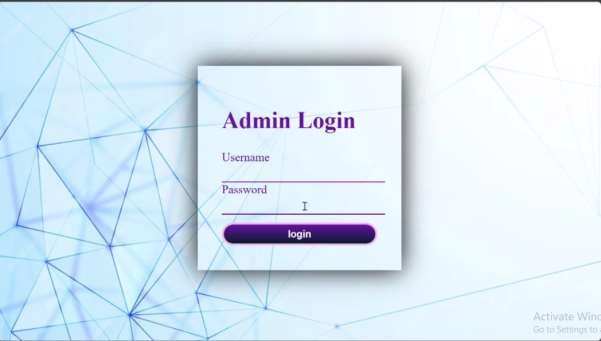
</div>

</form>

</div>

</body>

</html>



## CSS:

button{

position: relative; width: 100%; height: 45px;

background: transparent; border: 4px solid #F8BDEB; outline: none;

border-radius: 40px; cursor: pointer; font-size: 20px; color: #fff;

font-weight: 600;

z-index: 1; overflow: hidden; margin-top: 1rem;

}

button::before{ content:''; position: absolute; top: -100%;

left: 0;

width: 100%;

height: 300%;

background: linear-gradient(#081b29, #6420AA, #081b29,#6420AA); z-index: -1;

transition: .5s;

}

button:hover::before{ top: 0;

}

#adm-Msg{

color: red;

font-size: 1.3rem; margin-left: 4rem;

}

**JavaScript Functions for Library Management System:**

The provided JavaScript code encompasses a set of functions crucial for the front-end functionality of a library management system. The `toggleMenu()` function facilitates toggling the visibility of menu options, enhancing user navigation within the system. `searchBooks()` asynchronously handles user queries, fetching and displaying search results dynamically while also providing suggestions when the search input is empty. The `fetchSuggestions()` function focuses solely on retrieving and displaying book suggestions to aid users in their search. Additionally, category-specific functions like `bookNowN`, `bookNowM`, `bookNowJ`, and `bookNowP` cater to different sections of the library, fetching and displaying relevant books based on user interaction. These functions collectively enhance the system's usability by providing real-time updates and interactive content management, ultimately improving the user experience of the library management interface.

## CODE:

function toggleMenu() {

var menuOptions = document.getElementById("menuOptions"); if (menuOptions.style.display === "block") {

menuOptions.style.display = "none";

} else {

menuOptions.style.display = "block";

}

}

async function searchBooks() {

const query = document.getElementById('searchInput').value.trim(); const searchResultsDiv = document.getElementById('searchResults'); searchResultsDiv.innerHTML = ''; // Clear previous search results

if (query === '') {

fetchSuggestions(); // Fetch and display suggestions searchResultsDiv.textContent = 'Please enter a book name.'; return;

}

try {

const response = await fetch(`/search?q=${encodeURIComponent(query)}`);

const books = await response.json();

const suggestionsDiv = document.getElementById('suggestions'); suggestionsDiv.innerHTML = ''; // Clear suggestions

books.forEach(book => {

const bookDiv = document.createElement('div'); bookDiv.classList.add('book'); // Add book class bookDiv.innerHTML = `

<img src="${book.image}" alt="${book.Title}" style="max-width:

200px;">

});

<h3>${book.Title}</h3>

<p>Author: ${book.Author}</p>

<p>Genre: ${book.Genre}</p>

<p>Available: ${book.Available}</p>

<p>location: ${book.location}</p>

`; suggestionsDiv.appendChild(bookDiv);

} catch (error) {

console.error('Error fetching search results:', error); searchResultsDiv.textContent = 'An error occurred while fetching

search results.';

}

}

async function fetchSuggestions() { try {

const response = await fetch('/suggestions'); const suggestions = await response.json();

const suggestionsDiv = document.getElementById('suggestions'); suggestions.forEach(book => {

const bookDiv = document.createElement('div'); bookDiv.classList.add('book'); bookDiv.innerHTML = `

<img src="${book.image}" alt="${book.Title}" style="max-width:

200px;">

<h3>${book.Title}</h3>

<p>Author: ${book.Author}</p>

<p>Genre: ${book.Genre}</p>

<p>Available: ${book.Available}</p>

<p>location: ${book.location}</p>

`;

suggestionsDiv.appendChild(bookDiv);

});

} catch (error) {

console.error('Error fetching suggestions:', error);

}

}

fetchSuggestions();

async function bookNowN() { try {

const response = await fetch('newbook1'); const books = await response.json();

const suggestionsDiv = document.getElementById('suggestions'); suggestionsDiv.innerHTML = ''; // Clear suggestions

books.forEach(book => {

const bookDiv = document.createElement('div'); bookDiv.classList.add('book'); // Add book class bookDiv.innerHTML = `

<img src="${book.image}" alt="${book.Title}" style="max-width:

200px;">

});

<h3>${book.Title}</h3>

<p>Author: ${book.Author}</p>

<p>Genre: ${book.Genre}</p>

<p>Available: ${book.Available}</p>

<p>location: ${book.location}</p>

`; suggestionsDiv.appendChild(bookDiv);

} catch (error) {

console.error('Error fetching search results:', error); searchResultsDiv.textContent = 'An error occurred while fetching

search results.';

}

}

async function bookNowM() { try {

const response = await fetch('magzine');

const books = await response.json();

const suggestionsDiv = document.getElementById('suggestions'); suggestionsDiv.innerHTML = ''; // Clear suggestions

books.forEach(book => {

const bookDiv = document.createElement('div'); bookDiv.classList.add('book'); // Add book class bookDiv.innerHTML = `

<img src="${book.image}" alt="${book.Title}" style="max-width:

200px;">

});

<h3>${book.Title}</h3>

<p>Author: ${book.Author}</p>

<p>Genre: ${book.Genre}</p>

<p>Available: ${book.Available}</p>

<p>location: ${book.location}</p>

`; suggestionsDiv.appendChild(bookDiv);

} catch (error) {

console.error('Error fetching search results:', error); searchResultsDiv.textContent = 'An error occurred while fetching

search results.';

}

}

async function bookNowJ() { try {

const response = await fetch('journal1'); const books = await response.json();

const suggestionsDiv = document.getElementById('suggestions'); suggestionsDiv.innerHTML = ''; // Clear suggestions

books.forEach(book => {

const bookDiv = document.createElement('div'); bookDiv.classList.add('book'); // Add book class bookDiv.innerHTML = `

<img src="${book.image}" alt="${book.Title}" style="max-

width: 200px;">

<h3>${book.Title}</h3>

<p>Author: ${book.Author}</p>

<p>Genre: ${book.Genre}</p>

<p>Available: ${book.Available}</p>

<p>location: ${book.location}</p>

`;

suggestionsDiv.appendChild(bookDiv);

});

} catch (error) {

console.error('Error fetching search results:', error); searchResultsDiv.textContent = 'An error occurred while fetching

search results.';

}

}

async function bookNowP() {

try {

const response = await fetch('project1'); const books = await response.json();

const suggestionsDiv = document.getElementById('suggestions'); suggestionsDiv.innerHTML = ''; // Clear suggestions

books.forEach(book => {

const bookDiv = document.createElement('div'); bookDiv.classList.add('book'); // Add book class bookDiv.innerHTML = `

<img src="${book.image}" alt="${book.Title}" style="max-width: 200px;">

<h3>${book.Title}</h3>

<p>Author: ${book.Author}</p>

<p>Genre: ${book.Genre}</p>

<p>Available: ${book.Available}</p>

<p>location: ${book.location}</p>

`; suggestionsDiv.appendChild(bookDiv);

});

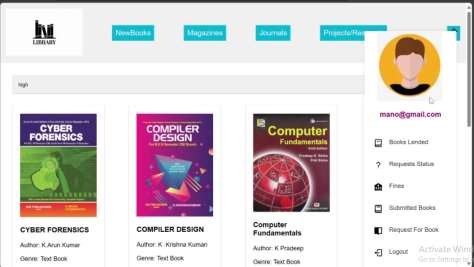
} catch (error) {

console.error('Error fetching search results:', error); searchResultsDiv.textContent = 'An error occurred while fetching

search results.';

}

}



## HOME PAGE USING REACT JS :

The provided code is a React component named `AdminPage` designed for the admin interface of a library management system. The component uses React hooks for state management, such as `useState`, to handle various interactive elements. The `menuOpen` state controls the visibility of the side menu, while `displayName` stores the name of the logged-in user. The `searchQuery`, `searchResults`, and `suggestions` states manage the search functionality and display book suggestions. Event handlers like `toggleMenu`,

`searchBooks`, and category-specific functions (`bookNowN`, `bookNowM`, `bookNowJ`,

`bookNowP`) manage user interactions, such as toggling the menu and initiating book searches.

The JSX structure comprises a header with a logo and buttons for navigating different book categories, such as New Books, Magazines, Journals, and Projects/Research. The header also includes a menu icon that, when clicked, toggles the side menu. The side menu displays the user profile and navigation links for various admin tasks, including modifying book status, adding new books, viewing requests, confirming submissions, checking fines, and logging out. The main section features a search bar for inputting search queries and viewing results, with a dedicated area for displaying book suggestions. This setup provides a foundational interface for library administrators, emphasizing user interaction and state management.

# CODE:

import React, { useState } from 'react'; import './homeadm.css';

import 'boxicons';

const AdminPage = () => {

const [menuOpen, setMenuOpen] = useState(false);

const [displayName, setDisplayName] = useState('Admin'); // You can set the initial state or fetch it from an API

const [searchQuery, setSearchQuery] = useState(''); const [searchResults, setSearchResults] = useState([]); const [suggestions, setSuggestions] = useState([]);

const toggleMenu = () => { setMenuOpen(!menuOpen);

};

const searchBooks = () => {

// Implement your search logic here

setSearchResults(['Book1', 'Book2', 'Book3']); // Example result

};

const bookNowN = () => {

// Handle new books click

};

const bookNowM = () => {

// Handle magazines click

};

const bookNowJ = () => {

// Handle journals click

};

const bookNowP = () => {

// Handle projects/research click

};

return (

<div>

<header>

<div className="logo">

<img

src="https://tse2.mm.bing.net/th?id=OIP.BjEx5m0e19xmMGEGuvySGgHaEc &pid=Api&P=0&h=180"

alt="Library Logo"

/>

</div>

<div className="NewBooks">

<button type="button" onClick={bookNowN}>NewBooks</button>

</div>

<div className="magzines">

<button type="button" onClick={bookNowM}>Magazines</button>

</div>

<div className="Journals">

<button type="button" onClick={bookNowJ}>Journals</button>

</div>

<div className="ProjectsResearch">

<button type="button" onClick={bookNowP}>Projects/Research</button>

</div>

<div className="menu">

<span className="menu-icon" onClick={toggleMenu}>

<box-icon name='menu' animation='spin' color='rgba(0,0,0,0.97)'></box-icon>

</span>

{menuOpen && (

<div className="menu-options" id="menuOptions">

<div className="profile">

<img

src="https://tse2.mm.bing.net/th?id=OIP.Z306v3XdxhOaxBFGfHku 7wHaHw&pid=Api&P=0&h=180"

alt="Profile"

/>

<br />

<h3><span id="display-name">{displayName}</span></h3>

</div>

<ul>

<li><box-icon name='book'></box-icon><a href="/modify- books">Modify Books Status</a></li>

<li><box-icon type='solid' name='book-add'></box-icon><a href="/Add-Books">Add New Books</a></li>

<li><box-icon name='question-mark' color='#101011'></box- icon><a href="Requests">Requests</a></li>

<li><box-icon name='book'></box-icon><a href="/Submissions">Confirm Submissions</a></li>

<li><box-icon name='coin-stack' type='solid' color='#101011'></box-icon><a href="AdmFines">Students With Fines</a></li>

<li><box-icon type='solid' name='book-open'></box-icon><a href="/NewBookRequests">New Requests</a></li>

<li><box-icon name='log-out' color='#101011'></box-icon><a href="/logout">Logout</a></li>

</ul>

</div>

)}

</div>

</header>

<main>

<div className="search-head">

<div className="search-bar">

<input

type="text" id="searchInput" placeholder="Search..." value={searchQuery}

onChange={(e) => setSearchQuery(e.target.value)}

/>

<button onClick={searchBooks}>Search</button>

<div id="searchResults">

{searchResults.map((result, index) => (

<div key={index}>{result}</div>

))}

</div>

</div>

</div>

<div className="book-box">

<div id="suggestions">

{suggestions.map((suggestion, index) => (

<div key={index}>{suggestion}</div>

))}

</div>

</div>

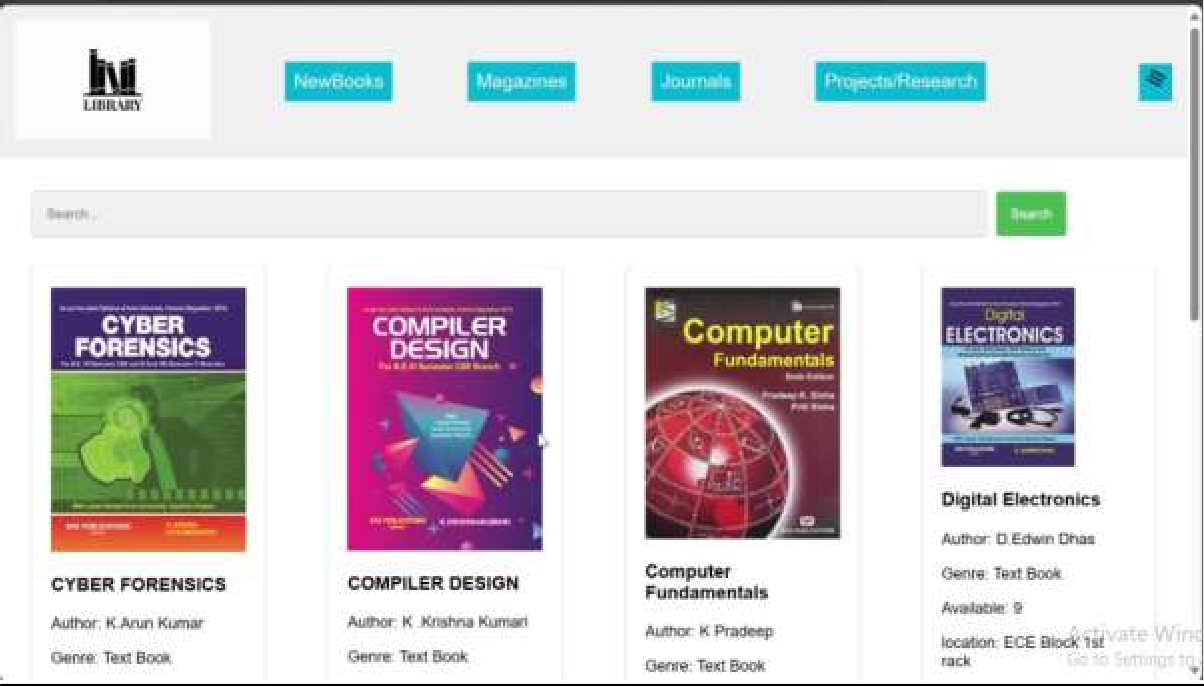
</main>

</div>

);

};

export default AdminPage;



**Library Management System Backend Using Node & Express.js:**

This code sets up an Express.js server that handles user signup and login functionalities. It uses MongoDB for storing user data, Handlebars for rendering HTML templates, and middleware

**express-session**

**multer**

like management is

**body-parser**

for parsing request bodies and implemented using the

for handling file uploads. The session middleware, which helps in

maintaining user sessions across different routes. The server renders various pages (**start**, **signup**, **adminstd**, **homestd**) based on user interactions and manages authentication to secure user access.

The library management system is built upon a client-server architecture using Node.js and Express.js. The server-side application handles HTTP requests from the client (web browser) and interacts with the MongoDB database to manage library resources and user data.

const express = require("express") const multer = require('multer') const mongoose = require('mongoose')

const { MongoClient, ObjectID } = require('mongodb'); const app = express()

const session = require('express-session') const path = require("path")

const hbs = require("hbs")

const collection = require("./mongodb")

const RequestNew = require("./requestForNew") const collect = require("./mongodb1")

const books = require("./mongodb2") const StdData = require("./StudentData") const SubmitData = require("./Submit") const BookFines = require("./Fines")

const publicpath = path.join( dirname, '../public') const templatepath = path.join( dirname, '../templates') const keypath = path.join( dirname, '../keys')

const upload = multer({ dest: 'uploads/' }) const bodyParser = require('body-parser')

app.use(bodyParser.json()); app.use(session({

secret: 'secret', resave: false, saveUninitialized: true

}));

console.log(templatepath) console.log(keypath) app.use(express.json()) app.set("view engine", "hbs") app.set("views", publicpath)

app.use(express.urlencoded({ extended: false })) app.use(express.static(templatepath)) app.use(express.static(keypath)) app.get("/start", (req, res) => {

res.render("start")

})

app.get("/", (req, res) => { res.render("start")

})

app.get("/signup", (req, res) => {

res.render("signup")

})

app.get("/adminstd", (req, res) => { res.render("adminstd")

})

app.post("/signup", async (req, res) => { const data = {

name: req.body.name, password: req.body.password

}

await collection.insertMany([data]) res.render('signup')

})

app.post("/login", async (req, res) => {

try {

const check = await collection.findOne({ name: req.body.name })

if (check.password === req.body.password) { req.session.loginName = req.body.name; res.render('homestd', { displayName: check.name })

}

else {

res.render('signup', { loginMsg: "wrong password" })

}

}

catch {

res.render('signup', { loginMsg: "wrong details" })

}

})

**MongoDB Connection and Schema Definition for Library Management System:**

This code effectively sets up a MongoDB connection and defines a schema for managing book data within a library system. It handles connection success and failure scenarios and provides a structured way to store and retrieve book information, including images, titles,

authors, genres, availability, and locations. The model is made available for use in

**books**

other modules, facilitating database operations such as creating, reading, updating, and deleting book records.

MongoDB is employed as the database management system, providing a NoSQL solution for flexible and scalable data storage. The database schema is designed to accommodate collections for books, users, transactions, and fines. Mongoose, an ODM (Object Data Modeling) library for MongoDB, facilitates schema definition and database operations.

**CODE:**

const mongoose=require("mongoose")

mongoose.connect("mongodb://localhost:27017/MINIPROJECT\_39")

.then(()=>{

console.log("mongodb connected");

})

.catch(()=>{

console.log("failed to connected");

})

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'MongoDB connection error:'));

const AddedSchema=new mongoose.Schema({ image:{

type:String, required:true

},

Title:{

type:String, required:true

},

Author:{

type:String, required:true

},

Genre:{

type:String,

required:true

},

Available:{

type:Number, required:true

},

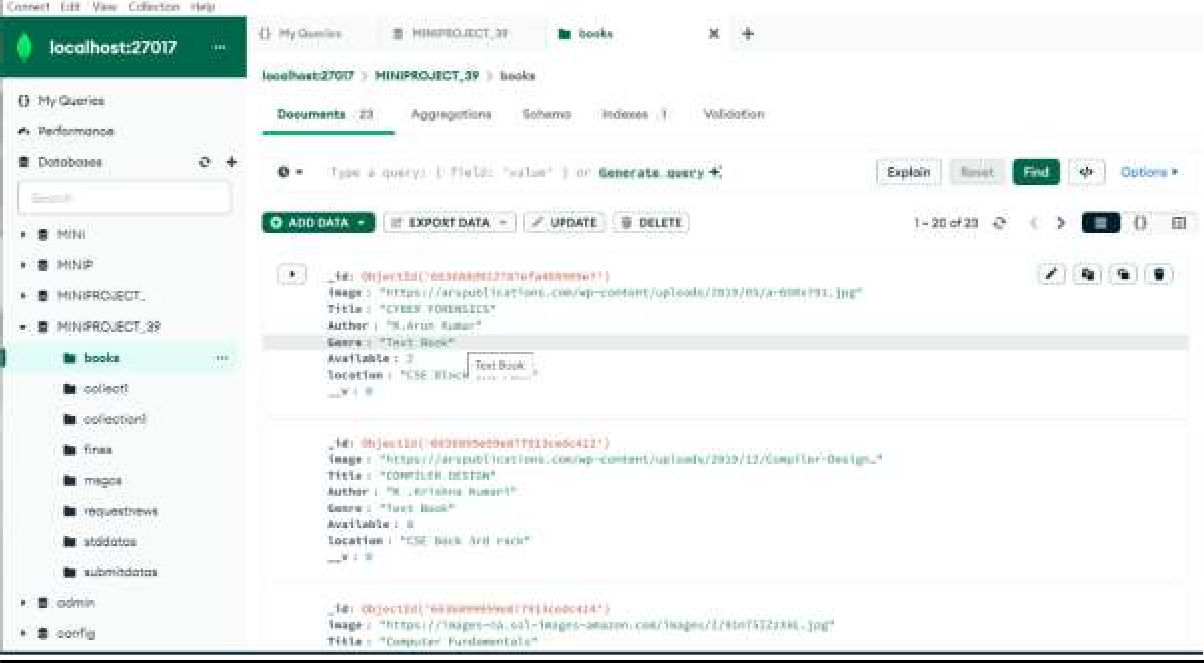
location:{

type:String, required:true

}

})

const books=new mongoose.model("books",AddedSchema) module.exports=books



## CONCLUSION :

In conclusion, the implementation of the library management system, with its intricate blend of backend and frontend functionalities, stands as a testament to the efficiency and effectiveness of modern technological solutions in streamlining library operations. Through meticulous database management, facilitated by MongoDB and Mongoose, the system adeptly stores and organizes vast repositories of book data, ensuring seamless retrieval and manipulation. Concurrently, the Express.js framework orchestrates the backend operations, facilitating smooth communication between the server and the client-side interface.

The frontend, powered by React.js, provides a visually appealing and intuitively navigable user experience, enhancing user engagement and satisfaction. Through features like real-time search functionality, category-specific book browsing, and dynamic content updates, the system caters to the diverse needs of both library administrators and patrons alike. As libraries continue to evolve in the digital age, this sophisticated management system serves as a cornerstone for modernizing library services, fostering greater accessibility, efficiency, and convenience for all stakeholders involved.