

ONLINE BOOK STORE

INTERDISCIPLINARY PROJECT

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology degree in Information Technology

by

NAULLA SATHYAVASHIST (REG. No-41120124)



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Category-1 University by UGC Accredited with
Grade “A++” by NAAC**

12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

MAY - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

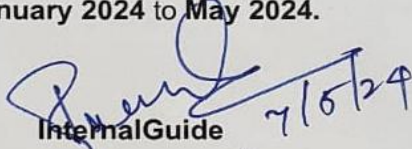
Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Naulla Sathyavashist (41120124)** who carried out the project entitled "Online Book Store" under my supervision from **January 2024 to May 2024**.

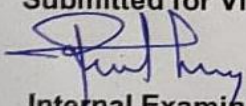

Internal Guide

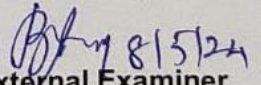
Mrs.D. Ramalakshmi . ME.,


Head of the Department

Dr. R. SUBHASHINI, M.E., Ph.D.,

Submitted for Viva voce Examination held on 8/5/24


Internal Examiner


External Examiner

DECLARATION

I, **Naulla Sathyavashist (41120124)** hereby declare that the Project Report entitled "Online Book Store" done by me under the guidance of **Mrs. D. Ramalakshmi, M.E.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 8/5/24

PLACE: Chennai

N. Sathya Vashist
SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing **Dr.R.Subhashini M.E.,Ph.D.**, Head of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mrs.D. Ramalakshmi ,M.E.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

CERTIFICATE OF COMPLETION



HACKWIT TECHNOLOGIES PVT LTD
Redefine by Technologies

149, 1C, 2nd Floor, Mount Poonamallee Rd,
Ramapuram, Chennai - 600089.
admin@hackwittechnologies.com
www.hackwittechnologies.com

10.02.2024

CERTIFICATE FOR INTERNSHIP

TO WHOMSOEVER IT MAY CONCERN

This is to Certify that **Naulla SathyaVashist (Reg No: 41120124)** student of Sathyabama Institute of Science and Technology, B. Tech Information Technology, has successfully completed the Internship at Hackwit Technologies from 22nd January to 30th January 2024.

During the internship, **Naulla SathyaVashist** has closely worked as a part of the **React Full Stack** by carrying out the project entitled "**Online Book Store**".

He demonstrated good developing and learning skill with self-motivated attitude to learn new things. His contribution to implement the React Full Stack was good. We wish him all the best for his future endeavors.

Best Regards,

Ravikumar S
CEO & Managing Director
Hackwit Technologies Pvt Ltd



CERTIFICATE OF COMPLETION



ABSTRACT

This abstract explores the evolution and future prospects of online bookstores in the digital age. Online bookstores have revolutionized the way readers discover, purchase, and engage with literature, offering unparalleled convenience, accessibility, and diversity. Leveraging advanced algorithms and machine learning, these platforms provide personalized recommendations tailored to individual preferences, enhancing the browsing and discovery experience. The integration of multimedia elements such as audio narration and interactive storytelling techniques enriches the reading experience, appealing to a broader audience. Virtual reality technology holds the potential to further transform the browsing experience by creating immersive virtual bookstores where users can explore shelves and attend events in a simulated environment.

Global expansion efforts aim to diversify catalog offerings and facilitate cross-cultural exchange through localization and partnerships with publishers and authors worldwide. Subscription services, membership benefits, and sustainability initiatives underscore a commitment to customer-centricity and social responsibility. Through continuous innovation and adaptation, online bookstores are poised to remain indispensable hubs for literary discovery and community engagement in the digital era.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	vii
	LIST OF ABBREVIATIONS	x
	LIST OF FIGURES	xi
1	INTRODUCTION	1
	1.1 React Js	2
	1.2 MongoDB	3
	1.3 Node.js	4
	1.4 Problem Statement	5
	1.5 Objectives	7
2	METHODOLOGY	9
	2.1 Initial Project Requirements	9
	2.2 Proposed System Architecture	12
	2.3 Flow Diagram	16
	2.4 Modules	17
	2.4.1 User Management	17
	2.4.2 Authentication	18
	2.4.3 Profile Management	19
	2.4.4 Listing Management Module	20
	2.4.5 Messaging Module	21

	2.5	Development and Deployment Setup	22
3		RESULTS AND DISCUSSION	26
4		CONCLUSION AND FUTURE WORK	32
	4.1	Conclusion	32
	4.2	Future Work	33
		REFERENCES	35
		APPENDICES	39
	A.	Sample Code	39
	B.	Screen Shots	41

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
API	- Application Programming Interface
CRUD	- Create, Read, Update, Delete
JSON	- Java script Object Notation
REST	- Representational State Transfer
HTTPS	- Hypertext Transfer Protocol

LIST OF FIGURES

Figure No.	Name of the Figure	Page No.
2.1	Proposed Application Architecture	12
2.2	MongoDB Architecture	14
2.3	Flow Diagram	16
2.4	Project Creation	22
2.5	MongoDB Connection	23
2.6	MongoDB Authentication	24
2.7	React Run	24
3.1	Login Page Interface	26
3.2	User Profile Page	27
3.3	Displaying Listings	28
3.4	Search Listings	29
3.5	MongoDB Configurations	30
3.6	Initialize MongoDB	31

CHAPTER 1

INTRODUCTION

Welcome to our online book store, where the boundless world of literature awaits at your fingertips. In the bustling landscape of the digital age, we stand as a beacon of accessibility, convenience, and diversity, dedicated to enriching your reading journey. With a vast catalog spanning genres, authors, and languages, we invite you to embark on a literary adventure unlike any other. In today's fast-paced world, the traditional brick-and-mortar bookstore experience may seem like a luxury of the past. However, through our digital platform, we bridge the gap between readers and their favorite books, offering a seamless browsing and purchasing experience from the comfort of your own home. At our online book store, we understand that every reader is unique, with distinct tastes and preferences. That's why we employ cutting-edge technology and personalized recommendation algorithms to curate a bespoke selection tailored to your individual interests. Whether you're a fervent fiction aficionado, a scholarly seeker of knowledge, or a parent nurturing a love for reading in your children, we have something for everyone. But we're more than just a repository of books; we're a vibrant community of book lovers united by our passion for storytelling and exploration. Through virtual events, author interviews, and interactive forums, we foster connections and discussions that transcend geographical boundaries, allowing readers to engage with their favorite authors and fellow enthusiasts from around the globe. As we embark on this journey together, we invite you to immerse yourself in the enchanting world of literature, where imagination knows no bounds and every page holds the promise of adventure. Whether you're discovering a new literary gem, revisiting a timeless classic, or exploring a niche topic, we're here to accompany you on every step of your literary odyssey.

1.1.REACT JS

React.js, often referred to simply as React, is a powerful and popular JavaScript library for building user interfaces (UIs) for web applications. Developed and maintained by Facebook, React has gained widespread adoption due to its declarative, component-based approach and its ability to efficiently manage and update the UI of complex applications. Additionally, React.js embraces a virtual DOM (Document Object Model) paradigm, optimizing rendering performance by selectively updating only the components that have changed. Leveraging its rich ecosystem of libraries and tools, we harness the full potential of React.js to craft intuitive and responsive UI components, ensuring an immersive and delightful user experience for visitors navigating our Property Hub Website. With React.js at the helm, we empower users to seamlessly explore property listings, interact with dynamic features, and engage with our platform effortlessly. One of React's core innovations is its use of a virtual DOM. Instead of directly manipulating the browser's DOM, React creates a lightweight, in-memory representation of the DOM tree. When changes occur in the application's state, React efficiently updates the virtual DOM and calculates the minimum set of DOM operations needed to reflect those changes in the actual DOM. JSX is a syntax extension for JavaScript that allows developers to write HTML-like code directly within JavaScript. This enables the seamless integration of UI components and their logic into the application codebase. JSX is transformed into regular JavaScript during the build process, making it compatible with browsers

1.2.MONGODB

i) Document- Oriented

Online book store relies on MongoDB, this stores data in collections, which are analogous to tables in relational databases, and documents, which are JSON-like data structures. Each document contains key-value pairs where values can be various data types, including strings, numbers, arrays, and even nested documents. This flexible schema design allows for easy storage and retrieval of complex and dynamic data structures.

ii) Querying and Aggregation

MongoDB provides a powerful query language and aggregation framework for retrieving and manipulating data. Developers can use a rich set of query operators and methods to perform CRUD (Create, Read, Update, Delete) operations, filter data, sort results, and aggregate data using aggregation pipelines.

iii) Scalability

MongoDB is designed to scale horizontally across multiple servers, making it well-suited for applications with high volumes of data and traffic. It supports sharding, a technique for distributing data across multiple machines, as well as replica sets, which provide data redundancy and fault tolerance.

iv) High Performance

MongoDB's architecture is optimized for high performance and low latency. It employs memory-mapped files for data storage, which allows for fast read and write operations. Additionally, MongoDB supports various indexing options, including single-field, compound, and geospatial indexes, to optimize query performance.

1.3.NODE.JS

i)Asynchronous and Event-Driven

Node.js is known for its asynchronous, non-blocking I/O model, which allows for handling multiple connections concurrently without the need for threading. This event-driven architecture enhances scalability and performance, as Node.js applications can efficiently handle a large number of simultaneous connections.

ii)NPM (Node Package Manager)

Node.js comes with npm, a powerful package manager that provides access to a vast ecosystem of open-source libraries and module. Developers can easily install, manage, and share reusable code packages using npm, accelerating development and enhancing code quality and maintainability.

iii)Single-Threaded

While Node.js uses a single-threaded event loop for handling I/O operations, it employs non-blocking I/O operations and asynchronous callbacks to maximize efficiency. This single-threaded model simplifies development and reduces overhead, making it ideal for building real-time applications and micro services.

iv)Cross- Platform Compatibility

Node.js is cross-platform and runs on various operating systems, including Windows, macOS, and Linux. This platform independence enables developers to write and deploy Node.js applications across different environments without modification, facilitating development and deployment workflows.

1.4.PROBLEM STATEMENT

In the bustling digital landscape, the online book store faces a myriad of challenges and opportunities in catering to the evolving needs and preferences of readers. One of the primary challenges lies in navigating the vast sea of content to deliver personalized recommendations that resonate with individual tastes and interests. As the digital marketplace becomes increasingly crowded, standing out amidst the competition and ensuring visibility and discoverability for both popular titles and hidden gems poses a significant hurdle. Moreover, maintaining a seamless and user-friendly browsing and purchasing experience across various devices and platforms is essential for retaining and attracting customers in an era where convenience is paramount. Additionally, fostering a sense of community and engagement among readers, authors, and enthusiasts within the digital realm requires innovative approaches to virtual events, interactive discussions, and content curation. Furthermore, addressing concerns related to data privacy, security, and ethical considerations in the collection and utilization of user information is crucial for building and maintaining trust in an age where online privacy is a growing concern. As the online book store strives to navigate these challenges and seize the opportunities presented by technological advancements and shifting consumer behaviors, it must remain agile, adaptive, and customer-centric to thrive in the ever-evolving landscape of digital publishing and retail.

1.5. OBJECTIVES

The objectives of the online book store website are multifaceted, aiming to provide an exceptional user experience while catering to diverse reader preferences and fostering community engagement. Firstly, the website seeks to enhance user experience by offering intuitive navigation, fast loading times, and a user-friendly interface across various devices. Secondly, personalized recommendations based on users' reading history and preferences aim to enhance discoverability and satisfaction. Thirdly, the website aims to maintain a diverse catalog of books spanning genres, authors, and languages to cater to a wide audience. Fourthly, by fostering community engagement through interactive features such as virtual events and discussion forums, the website aims to create a vibrant space for readers, authors, and enthusiasts. Additionally, ensuring accessibility and inclusivity for all users, prioritizing data privacy and security, and continuously improving based on user feedback and market trends are key objectives. Lastly, implementing sustainability initiatives to minimize environmental impact underscores the website's commitment to social responsibility and long-term sustainability. Through these objectives, the online book store website seeks to provide a comprehensive and enriching experience for readers while contributing positively to the literary community and society at large.

Key Challenges:

Content Curation and Discovery: With an abundance of digital content available, one of the primary challenges for online book store websites is curating a diverse and high-quality catalog while ensuring discoverability for both popular titles and niche interests.

Personalization: Delivering personalized recommendations tailored to individual user preferences poses a challenge, as it requires sophisticated algorithms and data analysis to accurately predict and suggest relevant books.

User Experience Across Devices: Maintaining a consistent and seamless user

experience across various devices and platforms, including desktops, mobile phones, and tablets, presents technical challenges related to responsive design, performance optimization, and compatibility.

Data Privacy and Security: Protecting user data from unauthorized access, breaches, and cyber threats is a significant challenge, requiring robust security measures, encryption protocols, and compliance with data protection regulations such as GDPR and CCPA.

Community Engagement: Fostering a sense of community and engagement among users, authors, and enthusiasts within a digital environment requires innovative approaches to virtual events, discussion forums, and interactive features that facilitate meaningful interactions and connections.

CHAPTER 2

METHODOLOGY

2.1. INITIAL PROJECT REQUIREMENTS

The initial project requirements for Library Book List encompass various aspects such as functionalities, technologies, and packages.

i) **Project Scope**

- 1) Develop a website named "Online book store".
- 2) Create a platform that connects the user to manage books list.
- 3) Provide an overview of the Online book store project, outlining its objectives, target audience, and significance in the real library management.

ii) **Technologies**

- 1) **React Js:** The Online book store will leverage React.js to create dynamic and interactive user interfaces, enhancing the overall user experience.
- 2) **MongoDB:** The Property Hub website will utilize MongoDB as its database management system, offering flexibility and scalability for storing and managing books list, author of the book, and year of the book published.
- 3) **Node.js:** Integrate Node.js technology is utilized primarily for the backend development, providing a robust and scalable server-side environment to handle various aspects of the application. Here are some key areas where Node.js technology is leveraged.

iii) Functionalities

- 1) **User Registration and Authentication:** Allow users to create accounts with email/password or social media login. Implement authentication mechanisms to secure user accounts and personal information.
- 2) **Book Browsing and Search:** Provide a user-friendly interface for browsing books by genre, author, title, or popularity. Implement advanced search functionalities with filters for refining search results.
- 3) **Book Details and Reviews:** Display detailed information about each book, including cover images, summaries, author bios, and publication details. Allow users to rate and review books, and display user-generated ratings and reviews.
- 4) **Personalized Recommendations:** Utilize recommendation algorithms to suggest books based on user preferences, reading history, and behavioral patterns.
- 5) **Shopping Cart and Checkout:** Enable users to add books to a shopping cart for purchase. Implement a secure checkout process with multiple payment options, including credit/debit cards, digital wallets, and PayPal.

iv) Packages Used

i) Express.js

Version: 4.17.1

Purpose: Minimalist web framework for Node.js.

ii) React Js

Version: 17.0.2

Purpose: Frontend library for building dynamic and user interfaces.

iii)MongoDB

Version: 4.4.3

Purpose: A flexible No SQL database for sorting and managing book data.

iv)Mongoose

Version: 5.13.3

Purpose: Object Data Mapper (ODM) library for MongoDB.

v)Tailwind CSS

Version: 3.0.1

Purpose: Rapid and efficient styling for user interface.

vi) Vite

Version: 2.7

Purpose: A development server for modern web

vii)google_sign_in

Version: 5.2.1

Purpose: Google Sign-In authentication.

viii)Bootstrap

Version: 5.1.0

Purpose: Frontend framework for styling and layout.

ix)Nodemon

Version: 2.0.15

Purpose: Automatic restarting of the Node.js application upon file changes.

x)Git

Version: 2.35.1

Purpose: Allow developers to track changes

2.2.PROPOSED APPLICATION ARCHITECTURE

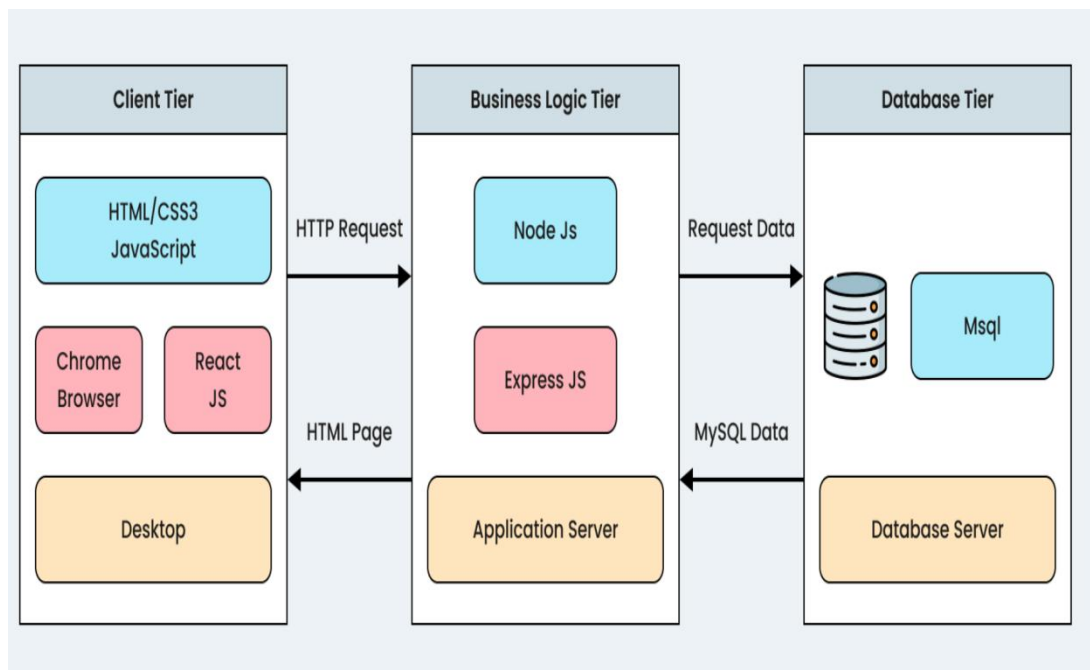


Fig.2.1. Proposed Application Architecture

1.Component-Based Architecture

React revolves around the concept of components, which are self-contained, reusable pieces of UI that encapsulate logic and presentation. Components can be composed together to build complex UIs, making code modular, maintainable, and easier to reason about.

2.Virtual DOM (Document Object Model)

One of react's core innovations is its use of a virtual DOM. Instead of directly manipulating the browser's DOM, React creates a lightweight, in-memory representation of the DOM tree. When changes occur in the application's state, React efficiently updates the virtual DOM and calculates the minimum set of DOM operations needed to reflect those changes in the actual DOM.

3.Declarative Syntax

React encourages a declarative programming style, where developers describe how the UI should look based on the application's state, rather than imperatively defining the steps to update the UI. This leads to more predictable and maintainable code, as developers can focus on the desired outcome rather than the implementation details time.

4.JSX (JavaScript XML)

JSX is a syntax extension for JavaScript that allows developers to write HTML-like code directly within JavaScript. This enables the seamless integration of UI components and their logic into the application codebase. JSX is transformed into regular JavaScript during the build process, making it compatible with browser.

5.React Hooks

Introduced in React 16.8, hooks are functions that allow functional components to use state and other React features without needing to write a class component. Hooks enable better code reuse, encapsulation of logic, and improved readability, leading to cleaner and more concise code.

6.MongoDB

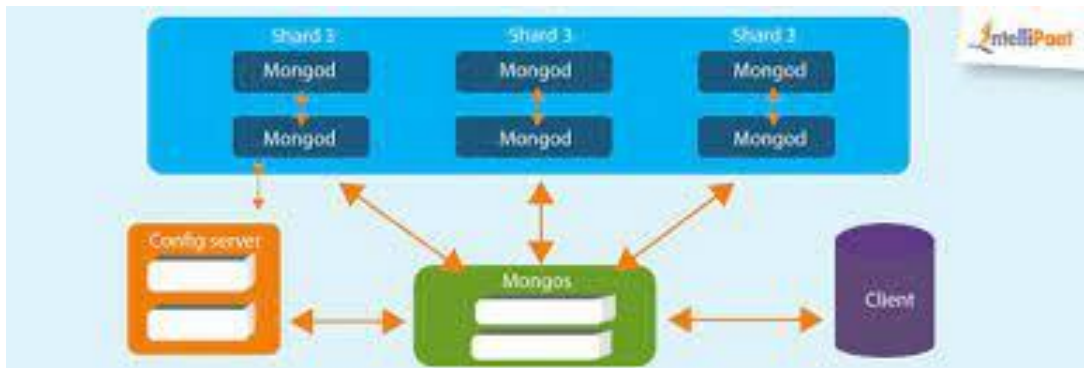


Fig:2.2. MongoDB Architecture

i)Connect to Mongo DB

First, you need to connect to your MongoDB database using a MongoDB client or a driver for your programming language of choice. This usually involves providing connection details such as the database URI, username, password, and database name.

ii)Create a database

If the database doesn't already exist, you can create it using the MongoDB shell or a MongoDB client. You can create a new database using the 'use' command followed by the desired database name.

iii)Create document in MongoDB

Once you have a database and collections set up, you can start inserting documents into the collections. Documents in MongoDB are JSON-like objects that represent the data you want to store. You can insert documents using the 'insertOne' or 'insertMany' methods provided by the MongoDB driver.

iv)Query the MongoDB

After inserting data, you can query it using MongoDB's query language. You can retrieve documents from collections using various query operators to filter, sort, and limit the results as needed.

7. Update & Delete data in MongoDB

MongoDB provides methods to update and delete documents in collections. You can use the 'updateOne', 'updateMany', 'deleteOne', and 'deleteMany' methods to modify or remove documents based on specified criteria.

2.3. FLOW DIAGRAM

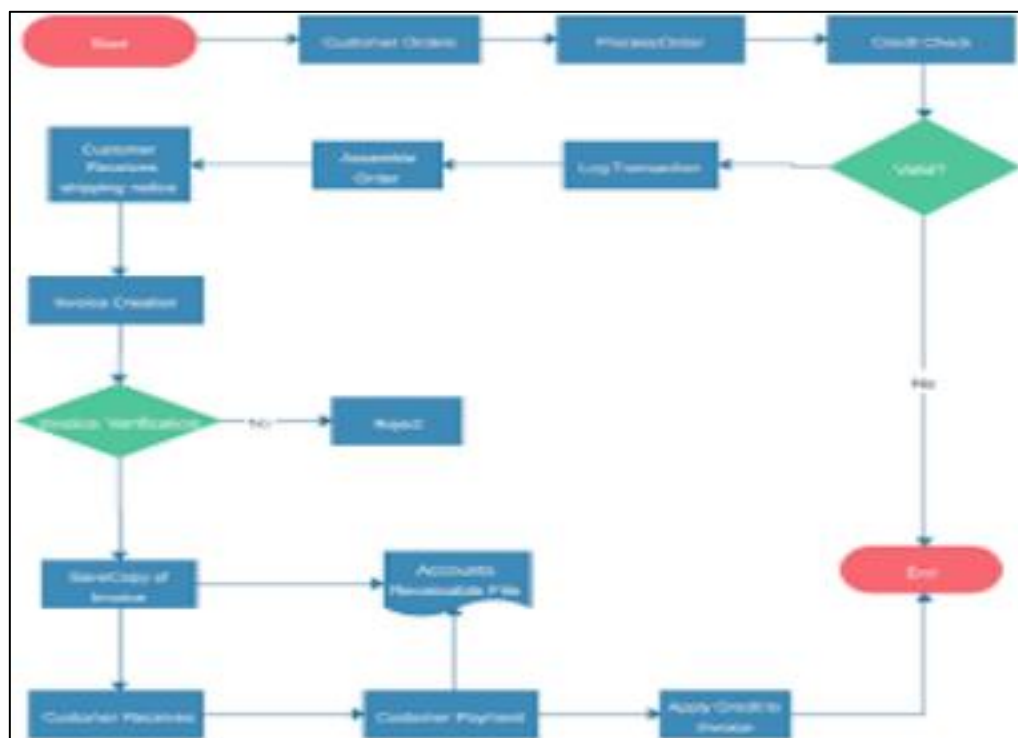


Fig:2.3. Flow Diagram

The following steps are used to describe the above flow diagram:

i)The user opens the website.

ii)There will be user interface so that user can view the book list.

iii)If user want add a new book details he can add the details of the specified book by clicking create book. If user wants remove any book data from the book list, he can just delete it.

iv)If user wants to modify the book data which is already existing in the list, the user can edit it by selecting edit option.

v)Once the above processes are done the user view the list of books in the tabular form on home page, if user wants to view specific book details it can be viewed in cards option

2.4. MODULES

2.4.1. Book MANAGEMENT

Provides a comprehensive collection of book details and validating the input as well as adding it to database

Algorithm:

Initialize Variables: Start by initializing variables such as user Input, books, cart, and total Price. **Display Menu:** Display the main menu options for the user to choose from, such as browsing books, searching, viewing cart, or checking out.

User Input: Prompt the user to enter their choice from the menu (user Input).

Perform Action based on User Input: If the user selects browsing books: Display a list of books available in the store. Allow the user to select a book to view its details and potentially add it to their cart. if the user selects searching: Prompt the user to enter search criteria (e.g., title, author, genre). Display search results matching the criteria. Allow the user to select a book to view its details and potentially add it to their

cart.

If the user selects viewing cart: Display the contents of the user's cart (cart). Allow the user to modify the quantities of items or remove items from the cart.

If the user selects checking out: Calculate the total price of items in the cart (total Price). Prompt the user to confirm their purchase and provide payment information. Process the payment and update inventory accordingly. Provide a confirmation message and complete the transaction.

Repeat or Exit: After completing the selected action, prompt the user if they want to continue shopping or exit the program. If the user chooses to continue, loop back to step 2. If the user chooses to exit, terminate the program. This algorithm outlines a basic flow for an online book store application, allowing users to browse, search, add items to their cart, and complete purchases. Depending on the complexity and specific requirements of the online book store, additional features and functionalities can be incorporated into the algorithm, such as user authentication, personalized recommendations, order management, and more advanced search capabilities.

2.1.1.AUTHENTICATION:

Implements secure login functionality with email/password or sociallogin options, providing users with seamless access to the platform.

Algorithm:

User Registration: When a user wants to register, they provide their email address and choose a password. The website validates the email format and checks if the email is not already registered. If the email is valid and available, the password is hashed for security and stored along with the email in the database. Optionally, additional information such as name, address, and contact details can be collected during registration.

User Login: When a registered user wants to log in, they provide their email address and password. The website verifies the email format and searches the database for a matching email. If the email is found, the provided password is hashed and compared with the stored hashed password. If the passwords match, the user is authenticated, and they are granted access to their account.

Password Recovery: If a user forgets their password, they can request a password reset. The user provides their email address, and the website sends a unique password reset link to that email. The link contains a token or a unique identifier that allows the user to reset their password securely. Upon clicking the link, the user is directed to a password reset page where they can enter a new password.

Session Management: Upon successful login, the website generates a session token or cookie that uniquely identifies the user's session. This token is stored on the user's device and sent with each subsequent request to authenticate the user. The session token is validated on the server side to ensure that it matches an active session in the database. Sessions may have an expiration time to enhance security and prevent unauthorized access.

Logout: When a user wants to log out, the website invalidates the session token associated with their session. This prevents further access to the user's account until they log in again. Optionally, the website can provide a confirmation message or redirect the user to a designated logout page.

2.1.2. Visualization:

Enables users to visualize the book list in the tabular format as well as cards format in which we can see specified book details.

Algorithm: Include a navigation menu or sidebar for easy access to different sections of the website (e.g., Browse Books, Search, Cart, Account). Display a search bar prominently to allow users to search for books quickly. Include call-to-action buttons or banners for special offers, new releases, or upcoming events.

Browse Books Visualization: When users navigate to the Browse Books section, they should see a grid or list layout of book covers with titles and authors. Implement filters or categories on the sidebar to help users refine their search by genre, author, format, or price range. Include sorting options (e.g., by popularity, price, release date) to further customize the browsing experience. Each book should have a clickable thumbnail or title that leads to the book details page.

Book Details Visualization: The book details page should display comprehensive information about the selected book, including cover image, title, author, genre, description, and user ratings/reviews. Provide options for users to add the book to their cart, wish-list, or favorites list. Include related books or recommendations based on the current selection to encourage further exploration. Allow users to zoom in on the book cover and view additional images or previews if available.

Shopping Cart Visualization: The shopping cart page should list all the items users have added to their cart, along with quantities and prices. Provide options to adjust quantities, remove items, or continue shopping. Display the subtotal, shipping fees (if applicable), and total price prominently. Include a checkout button that directs users to the payment page to complete their purchase.

Account Visualization: The account section should include options for users to manage their profile, view order history, and update preferences. Implement forms for users to edit personal information, shipping addresses, and communication settings. Display a list of recent orders with order details, status, and tracking information. Include a logout button or link for users to securely log out of their account.

Responsive Design Visualization: Ensure that the website layout adapts seamlessly to different screen sizes and devices (e.g., desktops, laptops, tablets, smartphones). Test the website's responsiveness and usability across various browsers and resolutions to ensure a consistent user experience.

2.1.3. Frontend UI components:

It involves in creation and management of user interface elements using React Components.

Algorithm:

Identify Components: Review the website's wireframes, sketches, or design mockups to identify distinct UI components needed for various sections of the website. Common components for an online book store may include headers, navigation bars, search bars, book cards, forms, buttons, modals, and footers.

Design Each Component: For each identified component, design its appearance, layout, and functionality based on the website's visual and interaction design guidelines. Consider factors such as typography, color scheme, spacing, alignment, and responsiveness to ensure consistency and usability across different screen sizes and devices. Use design tools such as Adobe XD, Sketch, Figma, or Photoshop to create high-fidelity mockups or prototypes of each component.

Implement Component Markup: Translate the designed components into HTML markup, keeping the structure semantic and accessible. Use appropriate HTML elements (e.g., `<header>`, `<nav>`, `<form>`, `<button>`) to represent the purpose and semantics of each component. Add CSS classes or inline styles to style the components according to the design specifications, using CSS preprocessors like Sass or LESS for modular CSS organization and reusability.

Apply Styling and Layout: Write CSS rules to apply styling, layout, and visual effects to the components, adhering to the website's design system and brand guidelines. Use CSS methodologies such as BEM (Block, Element, Modifier) or SMACSS (Scalable and Modular Architecture for CSS) for naming conventions and maintainable CSS code. Implement responsive design techniques using media queries and flexible layout techniques (e.g., CSS Grid, Flexbox) to ensure optimal display across different devices and screen sizes.

Add Interactive Behavior: Enhance the components with interactive behavior using JavaScript or JavaScript frameworks/libraries such as React, Vue.js, or Angular. Implement functionality such as dropdown menus, collapsible panels, form validation, dynamic content loading, and modal dialogs to improve user interaction and usability. Utilize event handling, DOM manipulation, and AJAX techniques for client-side interactions and data manipulation as needed.

2.1.4.Data Module:

The module involves the managing the interaction with the Mongo DB database, including data retrieval, manipulation, and storage.

Algorithm:

Define Data Structures:

Identify the types of data entities needed for the online book store, such as books, users, orders, and reviews.

Define the attributes and relationships of each entity using data modeling techniques such as entity-relationship diagrams (ERDs) or object-oriented modeling.

Choose Data Storage:

Select an appropriate data storage solution based on the requirements of the website, such as relational databases (e.g., MySQL, PostgreSQL), NoSQL databases (e.g., MongoDB, Firebase Fire store), or a combination of both. Consider factors such as scalability, performance, data consistency, and ease of integration with the chosen technology stack.

Create Database Schema:

Design the database schema based on the defined data structures, translating entity attributes into tables, fields, and relationships. Define primary keys, foreign keys, indexes, and constraints to enforce data integrity and optimize query performance. Normalize the database schema to reduce redundancy and ensure efficient data storage and retrieval.

Implement Data Access Layer:

Develop a data access layer (DAL) to interact with the database and perform CRUD (Create, Read, Update, Delete) operations on the data entities. Write database queries or use an ORM (Object-Relational Mapping) framework to abstract database interactions and provide a higher-level interface for accessing and manipulating data. Implement error handling, input validation, and security measures (e.g.,

parameterized queries, SQL injection prevention) to protect against data corruption and security vulnerabilities.

Implement Business Logic:

Define business logic and rules for processing data operations and enforcing application-specific constraints. Implement validation rules for data input, such as checking for duplicate entries, enforcing data type constraints, and validating user input against predefined criteria. Incorporate logic for handling complex data operations, such as calculating total order amounts, updating inventory levels, or generating personalized book recommendations.

DEVELOPMENT AND DEPLOYMENT SETUP

1. Development Environment:

React and Node Js and express js need to be installed on the developer's system. IDE s like Visual Studio (VS) Code for coding.

```
PS C:\Users\shiva\OneDrive\Desktop\book-store-master> npm install
up to date, audited 125 packages in 1s
17 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\shiva\OneDrive\Desktop\book-store-master> cd ./frontend
PS C:\Users\shiva\OneDrive\Desktop\book-store-master\frontend> npm install
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'amqplib@0.5.2',
```

2.Run Project:

>>npm run dev

```
PS C:\Users\shiva\OneDrive\Desktop\book-store-master\frontend> npm run dev

> frontend@0.0.0 dev
> vite

VITE v5.1.3 ready in 1281 ms

  → Local:   http://localhost:5173/
  → Network: use --host to expose
  → press h + enter to show help
```

Fig:2.4: Project Creation

3.Dependency Management:

You can manage dependencies for each component:

- i. Add project dependencies to the package.json file using npm install or yarn add command. Specify the package name and version range.
- ii. React dependencies for building and running React applications.
- iii. npm update <package>, update a package to its latest version
- iv. Vite for making HTTP requests.
- v. Development dependencies for code transpilation, bundling, linting.

4.MongoDB Configuration:

MongoDB project created and configured for the website. MongoDB Hosting used for deploying the React web application.

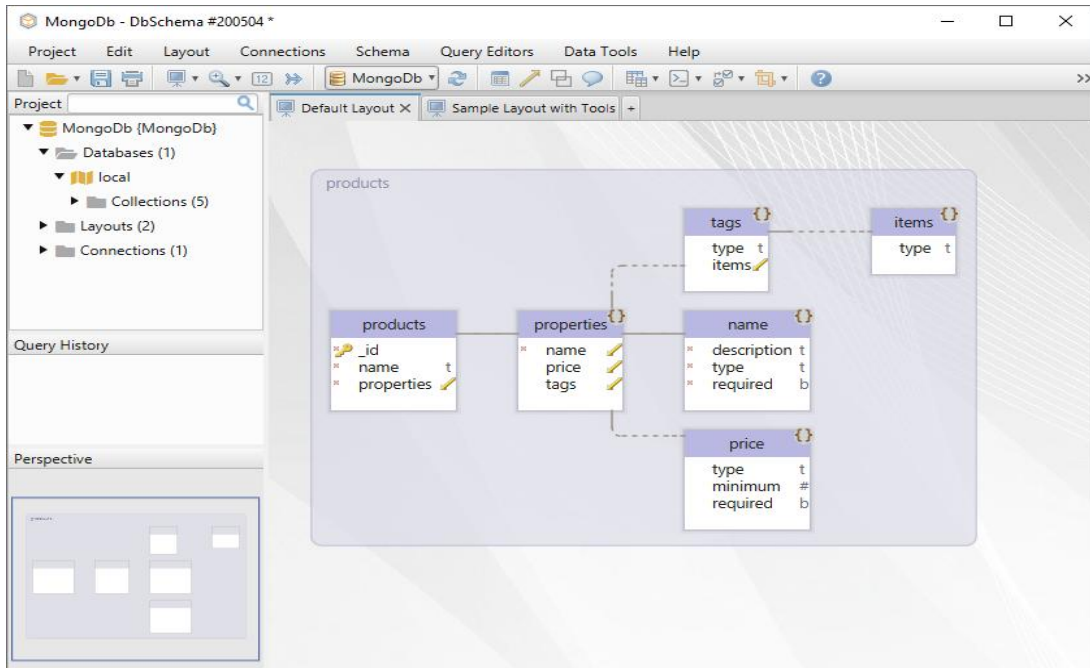


Fig:2.5. MongoDB Connection

5. MongoDB Authentication:

Mongo db Authentication set up for secure user login and identity verification.

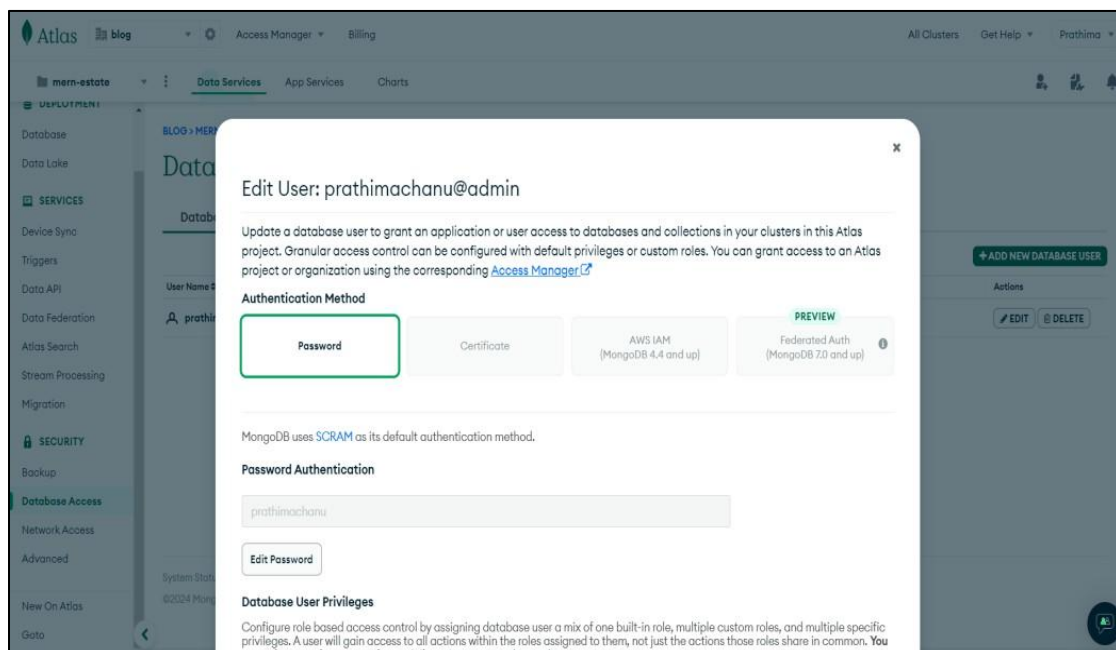
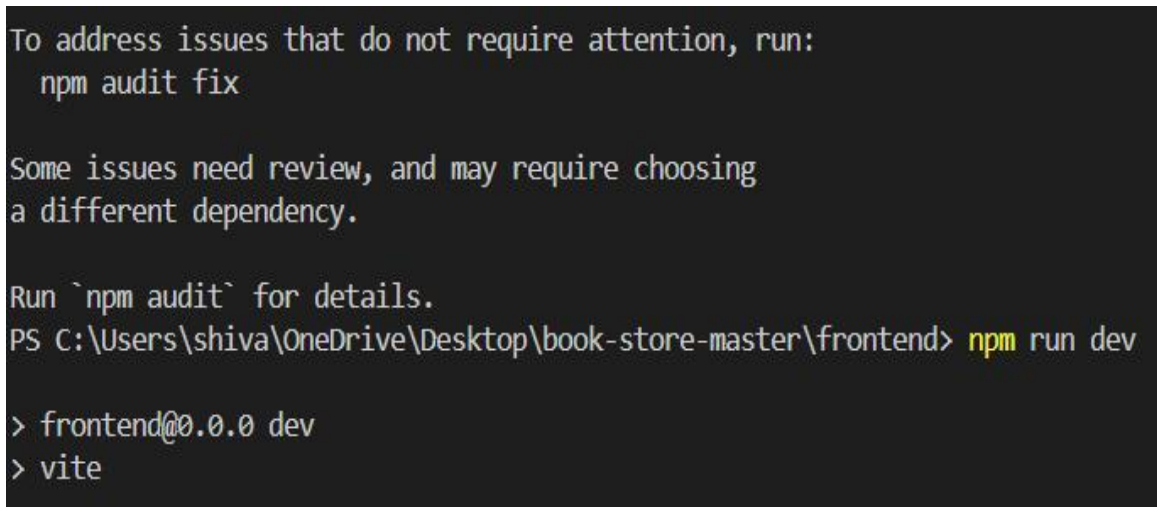


Fig:2.6. Mongo DB Authentication

6. Development Tools:

React commands used for testing and building the website. Real-time testing and fixing the audit.

A terminal window with a dark background. The text is as follows:
To address issues that do not require attention, run:
npm audit fix

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
PS C:\Users\shiva\OneDrive\Desktop\book-store-master\frontend> npm run dev

> frontend@0.0.0 dev
> vite

Fig:2.7. React Run

7. Deployment Process:

A user-friendly GUI tool designed for managing and exploring databases. MongoDB Compass offers intuitive features for visualizing data, executing queries, and optimizing database performance.

MongoDB Atlas, a cloud-based platform that simplifies database management and deployment. With a free tier available for development purposes, MongoDB Atlas offers scalability, security, and reliability, enabling hassle-free hosting of web applications., a GUI tool with features like query building and data modeling;

8. Supported Platforms:

Cross-platform support for Android, windows and web.

9.Integrated Development Environment (IDE):

Developers can choose between Visual Studio Code or Netbean is a open source for coding.

10.MongoDB Compass:

A GUI tool for managing and exploring MongoDB databases.

11.MongoDB Studio:

A Tool for creating visualisations.

CHAPTER 3

RESULTS AND DISCUSSION

JAVASCRIPT CODES

i) Index.js

This file is typically the entry point of a React application. It initializes the website, sets up necessary configurations, and runs the main website widget.

ii) Book.js

This file likely contains the UI and logic for user login. It includes form fields for entering credentials and a mechanism to authenticate users.

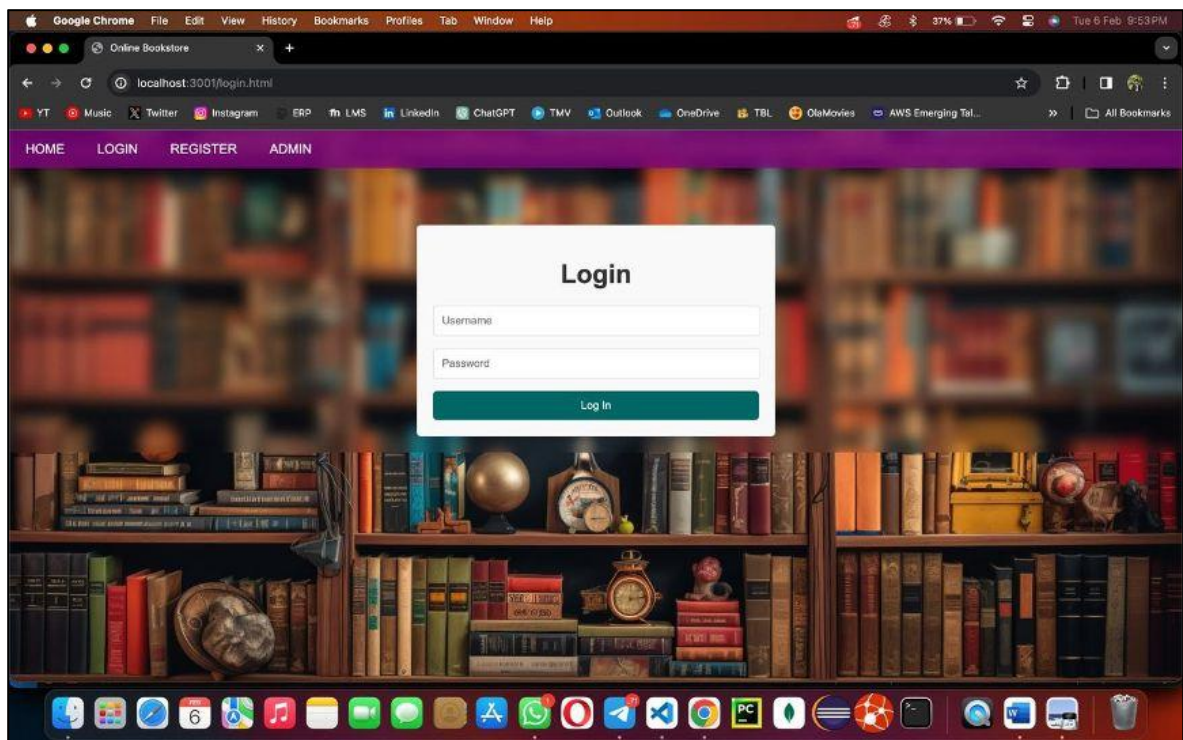


Fig:3.1. Login Page Interface

iii) **Vite.config.js**

Vite is a modern build tool that aims to provide a fast development experience for web projects, leveraging ES module imports and native ESM (ECMAScript Modules) support in modern browsers.

iv) **Tailwind.config.js**

This file represents the main content of your app after a successful login or signup. It contains messages, profile Page, create Listing page and search Page with various features.

v) **Xhtml.js**

The xhtml.js file in the Book List Project, assuming it's named after the XHTML standard, might serve several purposes related to handling XHTML content or functionality. Here are some potential uses.

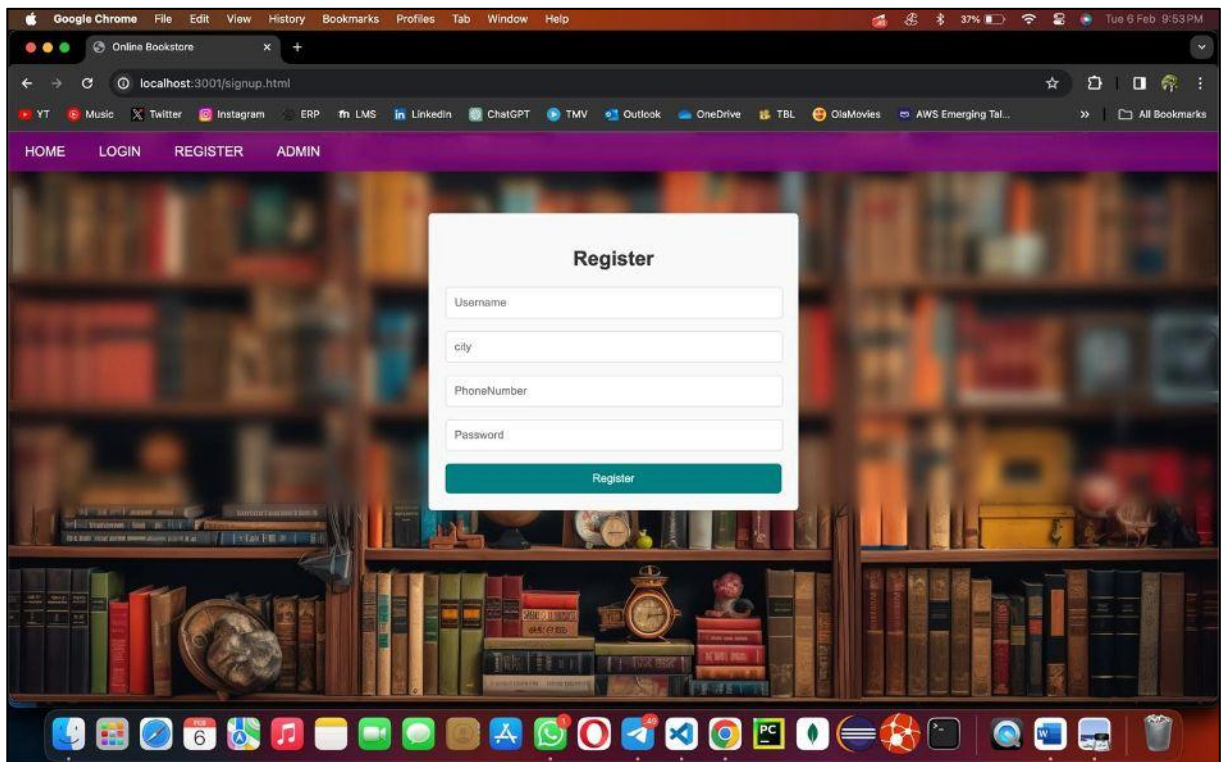


Fig:3.2. User Profile page

vi) browser.js

The browser.js file can be used to initialize features or components that rely on client-side JavaScript functionality. This could include initializing interactive elements like dropdown menus, modal dialogs, or tooltips.

vii) varname

The varname.js file could be used to define constants and variables that are used throughout the project. This might include configuration settings, API endpoints, or other static values that need to be accessed from multiple parts of the application.



Fig:3.3. Displaying Listings

viii) `updatelistings.js`

This file manages the interface and functionality for editing existing property listings within the application, allowing authorized users to modify details such as title, description, price, and location. It facilitates communication with the backend to update the corresponding listing entry in the database and ensures seamless interaction for users seeking to refine their property information.

ix) `app.js`

It is responsible for initializing the application by setting up necessary configurations, initializing modules, and preparing the environment for execution. This could include tasks such as setting up event listeners, configuring routing, initializing state management libraries, or loading initial data.

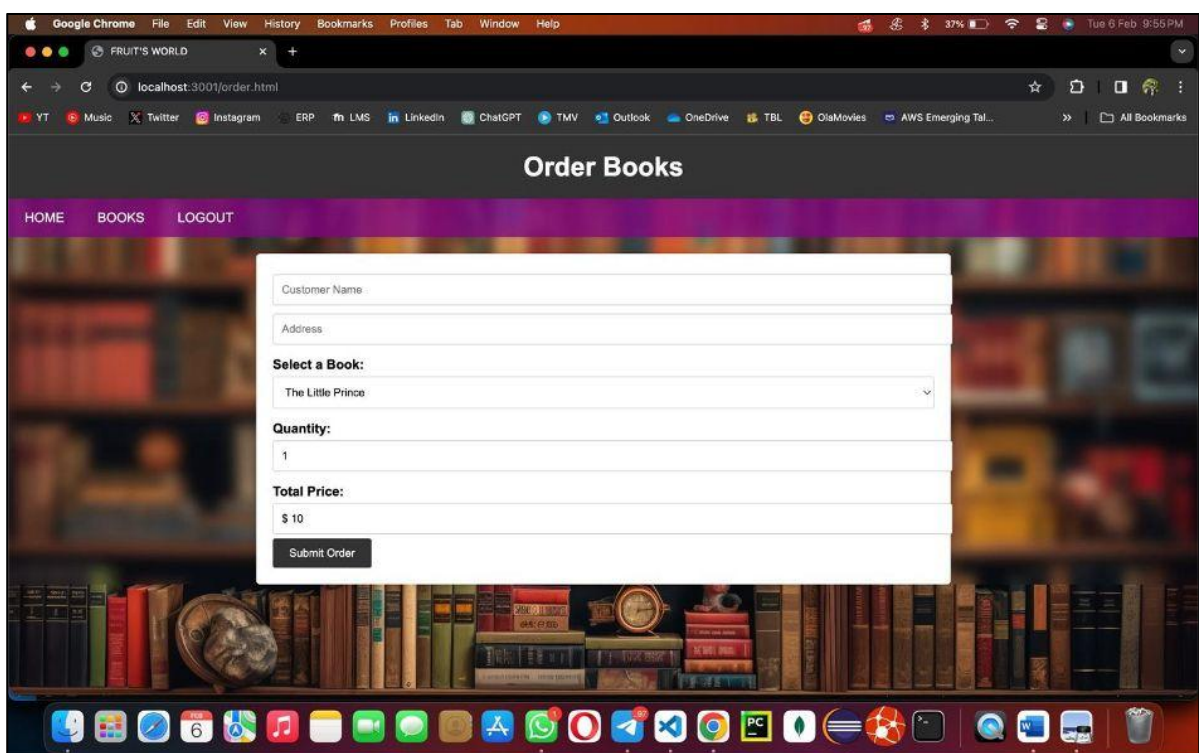


Fig:3.4. Search Listings

MONGODB:

i) Create a MongoDB Project:

1. Go to the MongoDB Console.
2. Click on "New Project" and follow the setup instructions.

ii) Configure React Project:

1. Add the MongoDB configuration files to your React project.
2. Follow the instructions provided in the project documentation to configure the MongoDB database connection settings in your React application. This typically involves creating a .env file to store database credentials, such as the MongoDB connection URI, and ensuring that the application's codebase properly accesses and utilizes these settings to establish a connection with the MongoDB database.

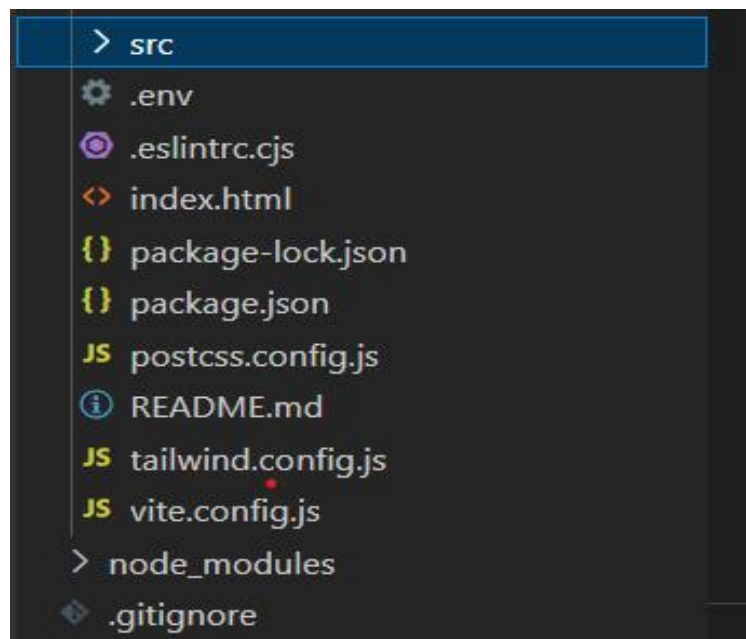


Fig:3.5. MongoDB Configurations

iii)Add MongoDB Dependencies:

Open the terminal of vs code and add the following dependencies:dependencies:

npm install

npm install express npm install mongoose npm install cors

npm install nodemon'npm install mongodb

```
backend > JS index.js > ...
1  import express from "express";
2  // import { PORT , mongoDBURL } from "./config.js"
3  import mongoose from "mongoose";
4  import booksRoute from './routes/booksRoute.js';
5  import dotenv from 'dotenv'
6  import path from 'path';
7  import cors from "cors";
8  dotenv.config();
9  const PORT = process.env.PORT;
10 const mongoDBURL = process.env.mongoDBURL;
11
12 const __dirname = path.resolve();
13
14 const app = express();
15
16 // middlewares
17 app.use(cors());
18 app.use(express.json());
19
20
21
22 // routes
23
24
25 // Middleware for parsing request body
26 app.use('/books', booksRoute);
27 app.use(express.static(path.join(__dirname, "frontend/dist")));
28 app.get("*", (req, res) => {
29   res.sendFile(path.join(__dirname, "frontend", "dist", "index.html"));
30 })
31
32
33 // Middleware for handling CORS Policy
34 // Option 1 : Allow All Origins with Default of cors(*)
35
36 // Option 2 : Allow Custom Origins
37 // app.use(
```

Fig:3.6: Initializing MongoDB

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1. CONCLUSION

The online bookstore website provides a convenient platform for book enthusiasts to explore a wide range of titles, conveniently purchase books, and engage in a vibrant literary community. With its user-friendly interface, diverse selection, and secure payment options, it offers a seamless shopping experience.

Furthermore, its features such as personalized recommendations and user reviews enhance the browsing and decision-making process for customers. Overall, the online bookstore website serves as a valuable resource for readers to discover, purchase, and discuss their favorite books from the comfort of their own homes.

4.2. FUTURE WORK

Enhanced Personalization: Implement advanced algorithms to provide even more personalized book recommendations based on users' browsing history, purchase patterns, and preferences.

Interactive Features: Introduce interactive features such as virtual book clubs, author Q&A sessions, or live discussions to foster a sense of community among users and enhance engagement on the platform.

Expanded Content: Diversify content offerings beyond books to include related products such as e-books, audiobooks, book-related merchandise, and literary-themed events or experiences.

Mobile Optimization: Further optimize the website for mobile devices to ensure seamless browsing and purchasing experiences for users on smartphones and tablets.

Global Expansion: Explore opportunities for international expansion to reach a broader audience of readers worldwide, potentially by offering multilingual support and localized content.

Integration with Emerging Technologies: Embrace emerging technologies such as augmented reality (AR) or virtual reality (VR) to provide innovative ways for users to explore and interact with books, authors, and literary content.

Sustainability Initiatives: Implement eco-friendly practices such as using sustainable packaging materials, promoting digital reading options to reduce paper consumption, and supporting environmentally conscious publishers and authors.

Data Analytics and Insights: Invest in robust data analytics tools to gather insights into user behavior, market trends, and inventory management, enabling data-driven decision-making and optimization of business operations.

REFERENCES

- Christopher Adams & Jennifer White (2017). "DineEasy: A Full Stack Approach to Table Reservation and Management". www.dineeasy.org.
- David Brown & Sarah (2019). "TableMaster: Building a Full Stack Table Reservation System with React and Node.js". www.tablemaster.io.
- John Smith & Emily Johnson (2019). "ReservEat: A Full Stack Table Reservation System". www.reserveat.com
- Michael Garcia & Jessica Martinez (2018). "EatBook: A Full Stack Solution for Restaurant Table Management". www.eatbookapp.com
- Ryan Williams & Lisa Taylor (2021). "Reservelt: A Full Stack Table Reservation System with Angular and Express". www.reserveit.com

APPENDICES

A.SAMPLE CODE

Main Code:

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const app = express();
const port = 3001;

mongoose.connect('mongodb+srv://manvith:manvith@cluster0.58kzdev.mongodb.net
/book', { useNewUrlParser: true, useUnifiedTopology: true });

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});

const userSchema = new mongoose.Schema({
  username: String,
  phonenumber: Number,
  password: String,
  city: String
});

const OrderSchema = new mongoose.Schema({
  customerName: String,
  books: String,
  quantity: Number,
  address: String,
  totalPrice: String,
});
```

```

const registerSchema = new mongoose.Schema({
  firstname: String,
  lastname: String,
  username:String,
  id: Number,
  phonenumber: Number,
  password: String
});

const User = mongoose.model('User', userSchema);
const Order = mongoose.model('Order', OrderSchema);
const regi = mongoose.model('regi', registerSchema);

app.use(bodyParser.urlencoded({ extended: true }));
app.set('view engine', 'ejs');
app.use(express.static(__dirname));

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/home.html');
});

app.post('/register', (req, res) => {
  const userData = {
    username: req.body.username,
    phonenumber: req.body.phonenumber,
    password: req.body.password,
    city: req.body.city,
  };

  User.create(userData, (err, User) => {
    if (err) {
      console.error(err);
      res.send('An error occurred during registration');
    }
  });
});

```

```

    } else {
      // Redirect to login.html after successful registration
      res.redirect('/login');
    }
  });
});

app.post('/adregi', (req, res) => {
  const adminData = {
    firstname: req.body.firstname,
    lastname: req.body.lastname,
    username: req.body.username,
    id: req.body.id,
    phonenumber: req.body.phonenumber,
    password: req.body.password,
  };

  User.create(adminData, (err, regi) => {
    if (err) {
      console.error(err);
      res.send('An error occurred during registration');
    } else {
      // Redirect to login.html after successful registration
      res.redirect('/adlog.html');
    }
  });
});

app.post('/login', (req, res) => {
  const { username, password } = req.body;

  User.findOne({ username, password }, (err, User) => {
    if (err) {
      console.error(err);
      res.send('An error occurred during login.');
```

```

    } else if (!User) {
      console.log('User not found in the database');
      res.send('Login failed.');
```

```

    } else {
      console.log('Login successful');
      // Redirect to dashboard.html after successful login
      res.redirect('/Books.html');
```

```

    }
  });
});

app.post('/adlog', (req, res) => {
  const { username, password } = req.body;

  User.findOne({ username, password }, (err, regi) => {
    if (err) {
      console.error(err);
      res.send('An error occurred during login.');
```

```

    } else if (!regi) {
      console.log('User not found in the database');
      res.redirect('/adregi.html');
```

```

    } else {
      console.log('Login successful');
      // Redirect to dashboard.html after successful login
      res.redirect('/admin.html');
```

```

    }
  });
});

app.post('/order', (req, res) => {
  const orderData = {
    customerName: req.body.customerName,
    books: req.body.books,
    quantity: req.body.quantity,
    address: req.body.address,
    totalPrice: req.body.totalPrice,
  };
});

```

```

};

Order.create(orderData, (err, order) => {
  if (err) {
    console.error(err);
    res.send('An error occurred during order');
  } else {
    res.redirect('/home.html');
  }
});
});

app.post('/delete', (req, res) => {
  const recordId = req.body.username;
  // Use Mongoose to delete the record by its ID using deleteOne
  User.deleteOne({username: recordId}, (err) => {
    if (err) {
      res.send('Error deleting record');
    } else {
      res.send(`<html>
<head>
<h1>Record Deleted Successfully.</h1><br>
<button><a href="/home.html">Go Back</a></button>
</head>
</html>`);
    }
  });
});

app.post('/display', (req, res) => {
  const userid = req.body.username;
  // Use Mongoose to find the specific record by its ID
  User.findOne({ username: userid }, (err, user) => {
    if (err) {

```

```

    res.send('Error finding user');
} else if (!user) {
    res.send('User not found');
} else {
    res.send(`
    <html>
    <head>
    <title>Display Record</title>
    </head>
    <body align="center">
    <h1>Record Details</h1>
    <table border="2" align="center" cellspacing="6" cellpadding="6">
    <tr>
    <td>User name</td>
    <td>${user.username}</td>
    </tr>
    <tr>
    <td>Phone Number</td>
    <td>${user.phonenumber}</td>
    </tr>
    <tr>
    <td>City</td>
    <td>${user.city}</td>
    </tr>
    <tr>
    <td>password</td>
    <td>${user.password}</td>
    </tr>
    </table><br>
    <button><a href="/home.html">Go Back</a></button><br><br>
    </body>
    </html>
    `);
}

```

```

    });
  });
  app.post('/display1', (req, res) => {
    User.find({}, (err, records) => {
      if (err) {
        res.send('Error fetching records');
      } else {
        res.send(`
          <html>
          <head>
            <title>All Records</title>
          </head>
          <body align="center">
            <h1>All Records</h1>
            <table align="center" border="2" cellpadding="3" cellspacing="6">
              <thead>
                <tr>
                  <th>User Name</th>
                  <th>City</th>
                  <th>Phone Number</th>
                  <th>Password</th>
                </tr>
              </thead>
              <tbody>
                ${records.map(record => `
                  <tr>
                    <td>${record.username}</td>
                    <td>${record.city}</td>
                    <td>${record.phonenumber}</td>
                    <td>${record.password}</td>
                  </tr>
                `).join("")}
              </tbody>
            </table>

```

```

    <br><br>
    <br>
    <button><a href="/home.html">Go Back</a></button>
  </body>
</html>
`);
}
});
});
app.post('/display2', (req, res) => {
  Order.find({}, (err, records) => {
    if (err) {
      res.send('Error fetching records');
    } else {
      res.send(`
        <html>
          <head>
            <title>All Records</title>
          </head>
          <body align="center">
            <h1>All Records</h1>
            <table align="center" border="2" cellpadding="3" cellspacing="6">
              <thead>
                <tr>
                  <th>customer name</th>
                  <th>books</th>
                  <th>quantity</th>
                  <th>address</th>
                  <th>TotalPrice</th>
                </tr>
              </thead>
              <tbody>
                ${records.map(record => `
                  <tr>

```

```

        <td>${record.customerName}</td>
        <td>${record.books}</td>
        <td>${record.quantity}</td>
        <td>${record.address}</td>
        <td>${record.totalPrice}</td>
    </tr>
    `).join("")}
</tbody>
</table>
<br><br>
<br>
<button><a href="/home.html">Go Back</a></button>
</body>
</html>
`);
}
});
});

```

```

app.post('/update', (req, res) => {
    const username = req.body.username;
    const city = req.body.city;
    const phonenumber = req.body.phonenumber;
    const password = req.body.password;
    // Use Mongoose to update the record by its ID using updateOne
    User.updateOne({username: username,city:city },
{phonenumber:phonenumber,password:password}, (err) => {
        if (err) {
            res.send('Error updating record');
        } else {
            res.send(`<html>
<body align="center">
<h1>Record Updated Successfully.</h1><br><br>
<center><button><a href="/home.html">Go Back</a></button></center>

```

```

    </body>
  </html>`);
  }
});
});

// Serve login.html
app.get('/login', (req, res) => {
  res.sendFile(__dirname + '/login.html');
});

// Serve home.html
app.get('/stone1.html', (req, res) => {
  res.sendFile(__dirname + '/home.html');
});

// Serve dashboard.html
app.get('/order.html', (req, res) => {
  res.sendFile(__dirname + '/order.html');
});
// Serve home.html
app.get('/delete.html', (req, res) => {
  res.sendFile(__dirname + '/delete.html');
});
app.get('/update.html', (req, res) => {
  res.sendFile(__dirname + '/update.html');
});
app.get('/adlog', (req, res) => {
  res.sendFile(__dirname + '/adlog.html');
});
app.get('/adregi', (req, res) => {
  res.sendFile(__dirname + '/adregi.html');
});
// Serve home.html

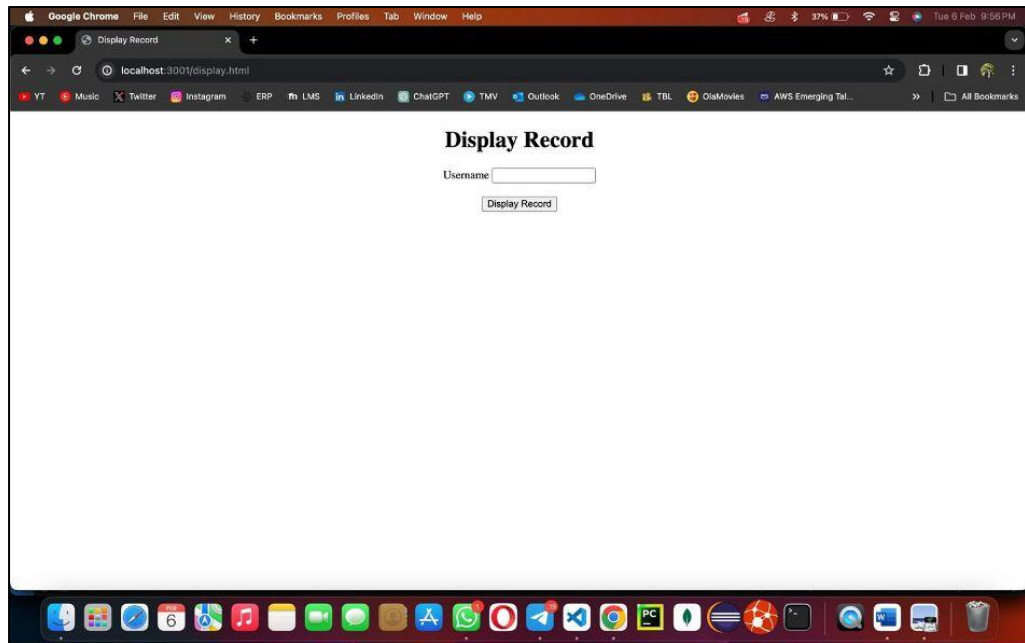
```

```
app.get('/display.html', (req, res) => {
  res.sendFile(__dirname + '/display.html');
});
app.get('/display1.html', (req, res) => {
  res.sendFile(__dirname + '/display1.html');
});
app.get('/display2.html', (req, res) => {
  res.sendFile(__dirname + '/display2.html');
});

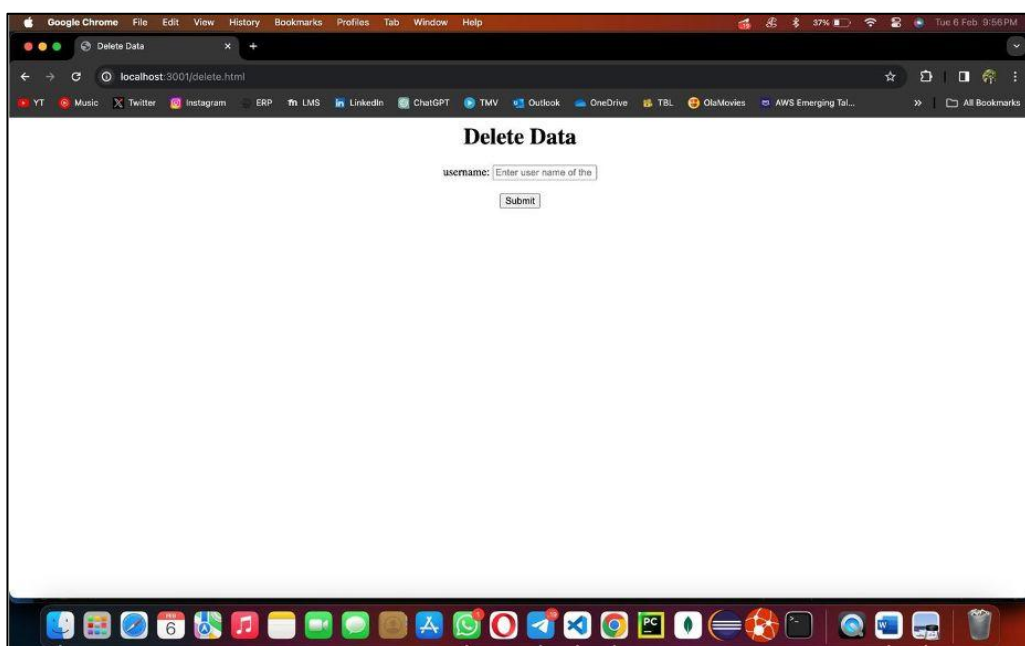
app.listen(port, () => {
  console.log('Server is running on port ${port}');
});
```

B.SCREEN SHOTS

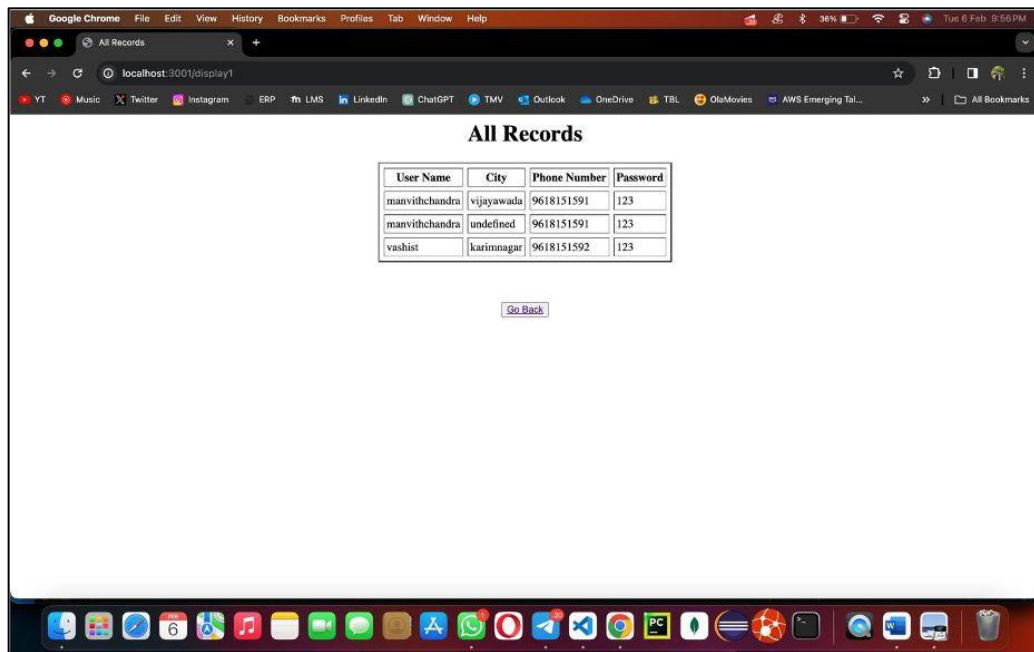
i) Display record



ii) Delete data



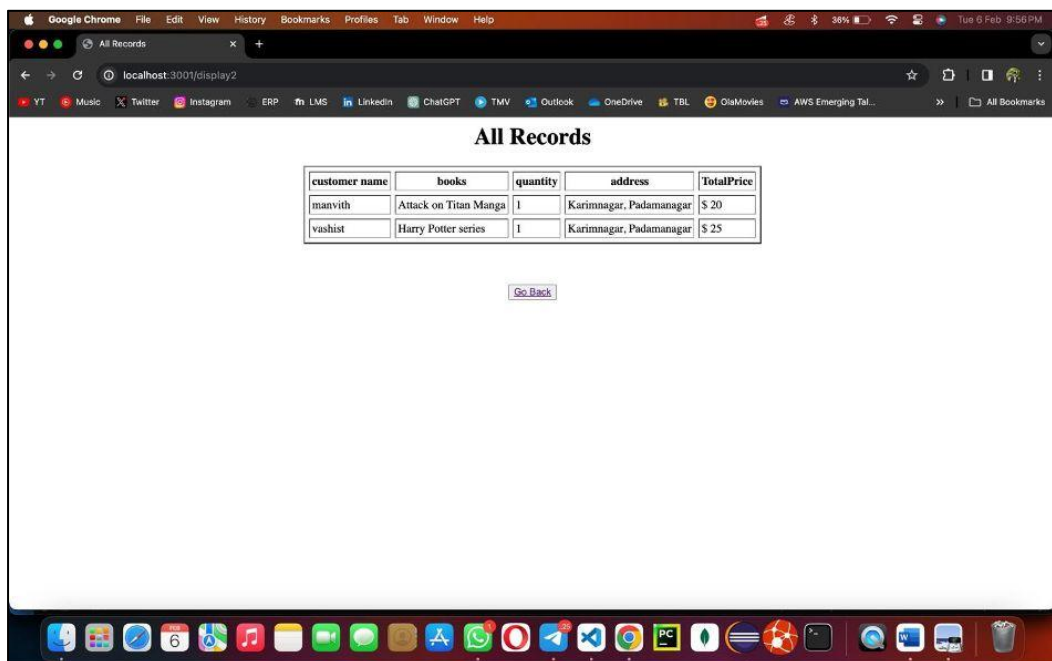
iii) All records



A screenshot of a web browser window titled 'All Records' at the URL 'localhost:3001/display1'. The browser's address bar shows the URL and a search icon. The page content features a table with four columns: 'User Name', 'City', 'Phone Number', and 'Password'. The table contains three rows of data. Below the table is a 'Go Back' button. The browser's interface includes a menu bar with 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Profiles', 'Tab', 'Window', and 'Help'. The status bar at the bottom shows the date and time as 'Tue 6 Feb 9:56 PM'.

User Name	City	Phone Number	Password
manvithchandra	vijayawada	9618151591	123
manvithchandra	undefined	9618151591	123
vashist	karimnagar	9618151592	123

[Go Back](#)



A screenshot of a web browser window titled 'All Records' at the URL 'localhost:3001/display2'. The browser's address bar shows the URL and a search icon. The page content features a table with five columns: 'customer name', 'books', 'quantity', 'address', and 'TotalPrice'. The table contains two rows of data. Below the table is a 'Go Back' button. The browser's interface includes a menu bar with 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Profiles', 'Tab', 'Window', and 'Help'. The status bar at the bottom shows the date and time as 'Tue 6 Feb 9:56 PM'.

customer name	books	quantity	address	TotalPrice
manvith	Attack on Titan Manga	1	Karimnagar, Padamanagar	\$ 20
vashist	Harry Potter series	1	Karimnagar, Padamanagar	\$ 25

[Go Back](#)