

K.R. MANGALAM UNIVERSITY

School of Engineering and Technology



Project Report On

BOOK BAZAR

Group Code ---- Y1-2024-25-G298

Academic Year: 2024–2025

Under the Guidance of :

Ms. Kriti Sharma

Submitted By:

Neha Chaudhary: 2401560029

Shristi Kumari : 2401560066

Class : MCA

CERTIFICATE

This is to certify that the project titled “**Book Bazar**” submitted by *Neha Chaudhary(2401560028)*, *Shristi Kumari(2401560066)* in partial fulfillment of the requirements for the award of the degree of **Master of Computer Applications** from **K.R. Mangalam University** is a bona fide record of work carried out by them during the academic year **2024–2025** under my guidance.

Ms.Kriti Sharma

Guide

Head of Department

Sign_____

INDEX

1.	Abstract	Page No.
2.	Introduction (description of broad topic)	01
3.	Motivation	02
4.	Literature Review	04
5.	Gap Analysis	06
6.	Problem Statement	07
7.	Objectives	08
8.	Methodology	09
9.	Tools/platform Used	11
10.	Implementation	14
11.	Complete Implementation	16
12.	Results And Discussion	81
13.	Conclusion & Future Work	85
14.	References	87

ABSTRACT

The **Book Bazar** project is an innovative full-stack web application designed to create a dynamic marketplace for users to buy and sell pre-owned books. Leveraging the power of the MERN stack (MongoDB, Express.js, React.js, and Node.js), the platform aims to foster a community-driven solution for book lovers, promoting sustainable practices by facilitating the reuse of books. The application features a user-friendly interface, allowing sellers to list books with detailed descriptions and buyers to browse through a categorized and searchable catalog of available books.

Key functionalities include secure user authentication, a robust cart management system, and an efficient checkout process. Sellers have access to tools for managing inventory, while buyers can save favourite books for later purchase. The backend ensures reliable data management, leveraging MongoDB for efficient storage and retrieval of book data and user information.

A standout feature of the project is the integration of a **Rasa-powered chatbot**, designed to enhance user interaction and support. This conversational AI assistant addresses user queries, provides guidance on navigation, helps resolve issues, and simplifies onboarding for new users. By enabling 24/7 real-time support, the chatbot significantly improves user experience and engagement.

The project incorporates responsive design principles to ensure seamless accessibility across various devices and platforms. It also prioritizes security through features like encrypted data transmission, robust authentication protocols, and secure API endpoints.

In addition to its technical achievements, Book Bazar addresses real-world challenges, such as reducing the cost of acquiring books and promoting environmental sustainability by reusing resources. It demonstrates the practical application of full-stack development principles and conversational AI to deliver a solution that is both user-centric and scalable.

The project not only highlights the potential of modern web technologies but also serves as a blueprint for combining intelligent automation with robust software development practices to meet diverse user needs in an evolving digital landscape.

KEYWORDS: *BookBazar, MERN, Rasa Chatbot*

Introduction

The demand for an accessible, sustainable, and community-driven platform for buying and selling books has grown in recent years. Whether for academic, leisure, or professional purposes, books remain an essential resource, yet the cost of new books can be prohibitive for many. Additionally, many used books often go unused, contributing to waste. The **Book Bazar** project addresses these challenges by creating a full-stack web application that connects buyers and sellers of pre-owned books, fostering a vibrant marketplace that is economically and environmentally beneficial.

Book Bazar is built using the **MERN stack** (MongoDB, Express.js, React.js, and Node.js), one of the most popular technology stacks for modern web development. The platform is designed to be scalable, secure, and user-friendly, offering features like account creation, secure login, book listings, and a streamlined purchase process. Sellers can list their books with essential details like title, author, price, and condition, while buyers can browse or search for books, add them to a cart, and complete purchases efficiently.

A key differentiator of Book Bazar is the integration of a **Rasa-powered chatbot**, which adds an intelligent conversational interface to the platform. The chatbot offers real-time assistance to users, guiding them through the platform, answering common questions, and helping resolve issues. This ensures a more personalized and engaging user experience, making the platform accessible even to less tech-savvy individuals.

The project adopts a modular design, ensuring flexibility and maintainability, and employs modern best practices in web development, including:

- **Responsive Design:** Optimizing the platform for use on mobile devices, tablets, and desktops.
- **Secure APIs:** Ensuring data security during interactions between the frontend and backend.
- **Robust Database Management:** Using MongoDB for efficient and scalable data storage.

The primary objective of this project is to create a seamless, user-centric platform that meets the diverse needs of book enthusiasts while promoting sustainability. Furthermore, by incorporating advanced conversational AI,

Book Bazar demonstrates the practical application of emerging technologies in enhancing user experience and engagement.

This project report delves into the technical, functional, and design aspects of Book Bazar, highlighting its development process, features, challenges, and future potential as a scalable and innovative solution in the e-commerce domain.

MOTIVATION

The idea for **Book Bazar** stems from a desire to address several challenges faced by book enthusiasts, students, and professionals in their pursuit of affordable and accessible books. Books, whether academic or recreational, play a vital role in education and personal growth. However, new books can be expensive, and many individuals, particularly students, may struggle to afford them. On the other hand, countless used books remain underutilized, gathering dust on shelves, when they could be valuable resources for others.

This project is driven by a commitment to creating a platform that bridges the gap between buyers and sellers of pre-owned books, fostering a community where books are not only more accessible but also reused sustainably. By providing a marketplace for second-hand books, Book Bazar encourages an environmentally conscious approach, reducing the demand for new print runs and minimizing waste.

The integration of a **Rasa-powered chatbot** further enhances the project's motivation by addressing the need for user-friendly assistance in navigating digital platforms. Many users, particularly those unfamiliar with technology, often face challenges when using e-commerce websites. By incorporating conversational AI, Book Bazar offers real-time support, ensuring that every user, regardless of technical expertise, can easily interact with the platform.

Key motivations behind the project include:

- **Affordability:** To make books more accessible to everyone by offering a platform for purchasing second-hand books at reduced prices
- **Sustainability:** To promote the reuse of books, thereby contributing to environmental conservation by reducing waste and the carbon footprint of new book production.
- **Convenience:** To provide a seamless and user-friendly platform where users can easily browse, buy, or sell books without technical hurdles.

- **Innovation:** To demonstrate the practical application of the MERN stack and conversational AI technologies, showcasing their potential to solve real-world problems effectively.
- **Community Building:** To create a space where book enthusiasts can connect and share resources, fostering a culture of learning and sustainability.

By addressing these key issues, Book Bazar aspires to make a meaningful impact on how books are bought and sold, creating a sustainable and inclusive ecosystem for knowledge exchange. This project is not only a testament to the possibilities of modern web development but also a step towards building a more resource-efficient and connected world.

LITERATURE REVIEW

The development of Book Bazar, a full-stack MERN (MongoDB, Express.js, React.js, Node.js) application with an integrated Rasa chatbot, is influenced by several technological advancements and existing research in the fields of web development, e-commerce platforms, and conversational AI. This literature review explores the foundational principles, existing solutions, and technological innovations that guided the project.

1. **Web Development Using MERN Stack:** - The MERN stack is widely regarded as a powerful framework for developing dynamic and scalable web applications. Research emphasizes the advantages of using React.js for building responsive user interfaces, MongoDB for handling large datasets efficiently, and Node.js for enabling asynchronous server-side operations (Kumar & Singh, 2021). By leveraging the MERN stack, Book Bazar achieves seamless interaction between the front-end and back-end, ensuring high performance and ease of maintenance.
2. **User Engagement in Marketplaces:** - User engagement is critical for the success of online marketplaces. Gamification, personalization, and customer support have been identified as key factors that enhance engagement (Chen et al., 2019). The integration of a Rasa chatbot in Book Bazar addresses the need for real-time customer support, providing instant responses to queries about book listings, account management, and transaction status.
3. **Conversational AI with Rasa:** - Rasa is an open-source conversational AI platform that enables developers to create intelligent chatbots using natural language understanding (NLU) and dialogue management. Literature on Rasa highlights its flexibility and effectiveness in providing tailored conversational experiences (García et al., 2020). By integrating Rasa, Book Bazar enhances user interactions, reduces customer support costs, and ensures accessibility for users unfamiliar with the platform's functionalities.
4. **Sustainability and Circular Economy:** - Recycling and reusing goods, including books, align with the principles of sustainability and circular economies. Research shows that initiatives encouraging the reuse of

resources contribute significantly to reducing waste and environmental impact (Geiss Doerfer et al., 2017). Book Bazar promotes sustainability by facilitating the exchange of pre-owned books, making it a platform with both environmental and economic benefits.

5. Challenges in Marketplace Development: - Developing a secure and reliable online marketplace poses several challenges, such as ensuring data privacy, preventing fraudulent transactions, and maintaining system uptime. Studies underline the importance of incorporating features like secure authentication, encrypted communication, and robust error handling in platform design (Choudhury et al., 2018). These principles are implemented in Book Bazar to ensure a secure and trustworthy user experience.

Conclusion

The literature reviewed provides a strong foundation for the design and development of Book Bazar. By drawing on best practices in web development, conversational AI, and e-commerce, this project aims to address the limitations of existing solutions while creating a platform that is both innovative and impactful. The integration of a Rasa chatbot further enhances its usability, making Book Bazar a standout application in the domain of online book exchanges

GAP ANALYSIS

The development of **Book Bazar**, a full-stack MERN application integrating a Rasa chatbot, is driven by the identification of specific gaps in existing solutions for book buying and selling platforms. This section analyzes these gaps to highlight the unique value that Book Bazar brings to the market.

1. Limited Focus on Niche Markets: - Large e-commerce platforms like Amazon and eBay cater to a wide variety of goods but lack dedicated support for niche markets, such as the pre-owned book market. Smaller platforms exist but often lack scalability and user-friendly features.
A dedicated marketplace that focuses exclusively on the needs of book buyers and sellers, providing features like categorized search, detailed book information, and community interaction.
2. Insufficient User Engagement and Interaction: - Current platforms often fail to engage users effectively, relying heavily on static interfaces with limited support for interaction or assistance.
Integration of a **Rasa chatbot** to provide real-time assistance, answering user queries about transactions, account setup, and book listings, thereby enhancing user experience and retention.
3. Complex User Interfaces in Existing Platforms: - Many platforms prioritize new book sales over promoting the reuse of pre-owned books, despite the growing demand for sustainable consumption.
By facilitating the buying and selling of used books, the platform aligns with sustainability goals, reducing the demand for new books and promoting environmentally friendly practices.

PROBLEM STATEMENT

The book market, particularly for pre-owned books, is fragmented and underdeveloped, leaving buyers and sellers without an effective, specialized platform. Existing e-commerce solutions either lack focus on niche markets like used books or fail to provide tailored functionalities for seamless transactions.

- Buyers struggle to find platforms specifically catering to affordable, second-hand books.
- Existing platforms lack features such as real-time assistance, efficient book categorization, and secure transactions specific to book trading.
- Overly complex interfaces, lack of personalization, and inadequate support for non-technical users hinder effective interaction.
- A gap in promoting sustainability by reusing books, contributing to increased demand for new prints and resource wastage.
- Users often face difficulties navigating platforms without real-time support, which can lead to frustration and transaction abandonment.

OBJECTIVES

The objectives of the **Book Bazar** project are as follows:

- Develop a dedicated platform for users to buy, sell, and exchange books, focusing on both new and pre-owned collections.
- Integrate a **Rasa chatbot** to provide instant support for user queries, navigation, and transactions.
- Encourage the reuse of books by facilitating the sale of second-hand books, reducing environmental impact.
- Implement features like user authentication, secure payment gateways, and transaction tracking for safe and reliable usage.
- Introduce smart categorization, advanced search filters, and personalized recommendations to enhance book discovery for users.
- Leverage the **MERN stack** (MongoDB, Express.js, React.js, Node.js) to ensure the application can handle growing user traffic and data efficiently.
- Provide features such as reviews, ratings, and feedback options to build a community of engaged book enthusiasts.
- Develop a responsive design to ensure a consistent and user-friendly experience across desktop and mobile platforms.
- Allow sellers to list books easily, track sales, and manage inventory through a simplified dashboard.
- Offer tools for sellers and administrators to analyze sales trends, user behaviour, and platform performance for continuous improvement.

Methodology

The development of the **Book Bazar** platform follows a systematic methodology to ensure a structured, efficient, and user-focused solution. The process involves several key stages, as outlined below:

1. Requirement Analysis

- Conduct surveys and interviews with potential users to gather insights into their challenges and expectations.
- Analyse existing book-selling platforms to identify shortcomings and opportunities for innovation.

2. System Design and Architecture

- Use **React.js** to create an intuitive and dynamic user interface with components like Home, About Us, Book Listings, Cart, and Profile pages.
- Implement responsive design to ensure compatibility across devices.
- Develop a robust backend using **Node.js** and **Express.js** to handle API requests, authentication, and database operations.
- Use **MongoDB** for managing user data, book listings, and transactional records, ensuring scalability and efficiency.
- Configure the **Rasa chatbot** to address user queries, assist with navigation, and provide recommendations.

3. Development Process

- Divide the development into sprints, focusing on specific features or modules in each sprint.
- Build features like book listing, user registration, search filters, and transaction management
- Train the Rasa chatbot with common user queries and intent detection using a well-structured knowledge base.

4. Testing

- Test individual components of the MERN stack for functionality.
- Verify the seamless interaction between the frontend, backend, and database
- Conduct user testing sessions to evaluate the platform's ease of use and overall experience.

- Test the Rasa chatbot for accurate responses and handling of edge cases.

5. Deployment

- Host the application on a cloud platform (e.g., AWS, Vercel, or Heroku) for accessibility.
- Use **MongoDB Atlas** for managing the database in the cloud
- Deploy the Rasa chatbot as a microservice to ensure scalability and reliability.

6. Maintenance and Updates

- Monitor platform performance and user feedback to address bugs and improve features.
- Regularly update the Rasa chatbot with new queries and enhanced responses based on user interactions.
- Scale the backend and database as the user base grows.

Details of tools, software, and equipment utilized.

PLATFORM USED

The **Book Bazar** project leverages a wide range of tools, software, and technologies to create a seamless, interactive, and scalable platform. Below are the details of the major tools and software used in the development process:

1. Frontend Development:

- A JavaScript library used to build the user interface, offering high performance using virtual DOM. React enables the creation of reusable UI components and efficient state management.
- HTML for structuring web content, and CSS for styling and layout. CSS ensures responsiveness and an adaptive design suitable for various screen sizes.
- A utility-first CSS framework used to design responsive and modern UI elements quickly. It provides pre-built classes for styling without writing custom CSS
- A library for handling routing within a React application. It allows smooth navigation between different pages (Home, About Us, Book Listings, Profile, Cart) without reloading the entire page.
- A promise-based HTTP client used for making API calls to the backend server, fetching data, and handling user requests.

2. Backend Development:

- A JavaScript runtime built on Chrome's V8 engine, enabling the development of scalable server-side applications using JavaScript. It supports asynchronous event-driven architecture, making it suitable for I/O-heavy applications.
- A minimalist web framework for Node.js, used for setting up routing, middleware, and API endpoints. It simplifies backend development by providing easy ways to handle HTTP requests.
- Used for authentication and securing API routes. JWT enables secure transmission of user data between the client and server by encoding the payload.

3. Database:

- A NoSQL database used for storing user data, book listings, transactions, and other dynamic content. MongoDB's flexible

schema is ideal for the unstructured data that might evolve over time, such as user preferences and book details.

- A fully managed cloud database service for MongoDB, offering automatic scaling, backups, and high availability, ensuring the database is secure and performs well in production.

4. Chatbot Integration:

- An open-source machine learning framework used for building conversational AI. Rasa handles natural language understanding (NLU) and dialogue management to create intelligent chatbots capable of engaging users with dynamic responses.
- A tool to improve and test the Rasa chatbot by collecting real conversations, enhancing model training, and improving user intent recognition.

5. Version Control and Collaboration:

- A distributed version control system used to manage the project's source code, track changes, and collaborate effectively among developers.
- A platform for hosting and sharing Git repositories, allowing developers to collaborate, manage pull requests, and deploy code efficiently.
- A CI/CD tool for automating the testing and deployment process. It helps to build, test, and deploy the application automatically when changes are made to the code.

6. Testing:

- A testing framework used to write unit tests for React components and backend functions. It supports mocking, snapshot testing, and asynchronous code testing.
- A testing tool used to send HTTP requests and assert responses for the backend API routes built using Express.
- A testing framework for end-to-end testing of web applications, ensuring that the UI and user flows are functioning as expected.
- A cloud platform as a service (PaaS) that enables easy deployment of Node.js applications. It simplifies the deployment and scaling process by providing integrated databases and easy configuration management.

7. Equipment:

- A personal computer or server with adequate specifications (e.g., 8 GB+ RAM, modern processor) to handle development tasks and testing of the application.
- A stable internet connection is necessary for accessing development tools, deploying the application, and running cloud services.

IMPLEMENTATION

The implementation of the Book Bazar project involved several phases, starting from designing the user interface (UI) to integrating features such as user authentication, book management, and chatbot integration. Below is a detailed explanation of how the project was implemented:

1. Frontend Implementation (React.js)

The frontend of **Book Bazar** was developed using **React.js**, a popular JavaScript library for building user interfaces. The main goal was to create an intuitive and responsive user experience.

Components and Pages:

- **Home Page:** The home page displays a list of books for sale, promotional banners, and quick navigation to other sections like the cart and user profile.
- **Book Listings:** Users can browse, filter, and search for books available for purchase. Each book listing includes a title, description, price, and an option to add the book to the cart.
- **Cart Page:** This page allows users to view items they've added to their cart, change quantities, and proceed to checkout.
- **Profile Page:** The profile page stores user details, including their address and order history.
- **Login/Signup Pages:** Users can sign up and log in using their credentials (email and password). The frontend uses React's controlled components for managing form data.

2. Backend Implementation (Node.js + Express.js)

The backend was built using **Node.js** and **Express.js**, creating RESTful APIs to handle user authentication, book listings, cart management, and order placement.

- During **signup**, the backend hashes the password using **bcrypt.js** before storing it in the database. After successful signup, a JWT token is created and sent to the frontend.
- During **login**, the entered password is compared with the hashed password stored in the database. If the login is successful, a JWT token is issued.

- POST /api/v1/sign-up: User registration, including hashing of the password and creation of a new user in the database.
- POST /api/v1/login: User login, verification of credentials, and generation of a JWT token.
- GET /api/v1/books: Fetches a list of books from the database, including details such as title, author, description, and price.
- POST /api/v1/cart: Adds items to the user's cart.
- GET /api/v1/cart: Retrieves the current user's cart items.
- POST /api/v1/order: Creates a new order from the user's cart and processes the payment
- GET /api/v1/order/:id: Fetches order details by ID.

3. Rasa Chatbot Integration

A key feature added to the **Book Bazar** project is the **Rasa Chatbot**, which was integrated to assist users in finding books, answering FAQs, and tracking orders.

- Rasa Server: The Rasa chatbot was deployed as a separate service, hosted on a cloud platform (e.g., Heroku).
- Rasa Training: The chatbot was trained using intents related to book searches, order tracking, and general inquiries.

COMPLETE IMPLEMENTATION

Backend

User.js

```
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema({
  username:
    { type: String,
      required: true,
      unique: true,
    },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password:
    { type: String,
      required: true,
    },
  address:
    { type:
      String,
      required: true,
    },
  avatar: {
    type: String,
```

```

default:
"https://i.pinimg.com/736x/93/e8/d0/93e8d0313894ff752ef1c6970116bad6.jpg"
},
role: {
  type: String,
  default: "user",
  enum: ["user", "admin"]
},
favourites: [
  {
    type: mongoose.Types.ObjectId,
    ref: "books",
  },
],
cart: [
  {
    type: mongoose.Types.ObjectId,
    ref: "books",
  },
],
orders: [
  {
    type: mongoose.Types.ObjectId,
    ref: "order",
  },
],

```

```
},  
{ timestamps: true }  
);  
module.exports = mongoose.model("User", userSchema);
```

Book.js

```
const mongoose = require("mongoose");  
const bookSchema = new mongoose.Schema(  
  {  
    title: {  
      type: String,  
      required: true,  
    },  
    author: {  
      type: String,  
      required: true,  
    },  
    price: {  
      type: Number,  
      required: true,  
    },  
    desc: {  
      type: String,  
      required: true,
```

```
},
language:
  { type: String,
    required: true,
  },
bookType:
  { type: String,
    required: true,
    enum: [
      "Hardcover",
      "Paperback",
      "E-Book",
      "Audiobook",
      "Graphic Novel",
      "Textbook",
      "Comic Book",
      "Magazine",
      "Journal"
    ],
  },
condition:
  { type:
    String,
    required: true,
    enum: ["New", "Like New", "Good", "Acceptable"],
  },
url: {
```



```

        type: String,
        required: true,
      },
    },
    { timestamps: true }
  );
module.exports = mongoose.model("Book", bookSchema);

```

Order.js

```

const mongoose = require('mongoose');

const orderSchema = new
  mongoose.Schema({ user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  book: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Book',
    required: true
  },
  quantity: {
    type: Number,
    required: true,

```

```

        default: 1,
        min: 1
    },
    orderDate:
        { type: Date,
          default: Date.now
        },
    status: {
        type: String,
        enum: ['pending', 'confirmed', 'shipped', 'delivered'],
        default: 'pending'
    }
}, { timestamps: true });

```

```

module.exports = mongoose.model('Order', orderSchema);

```

Server.js file

```

const express = require("express");
const app = express();
const cors = require("cors");

require("dotenv").config(); require('./conn/
conn');
const user = require("./routers/user")

```

```

const Books= require("./routers/book")
const Favourite = require("./routers/favourite")
const Cart = require("./routers/cart")
const Order = require("./routers/order")

app.use(cors());
app.use(express.json());

//routes app.use("/api/
v1", user); app.use("/api/
v1", Books);
app.use("/api/v1", Favourite);
app.use("/api/v1", Cart);
app.use("/api/v1", Order);
// Creating port
app.listen(process.env.PORT, () => {
  console.log(` server started at port ${process.env.PORT}`);
});

```

Routers User.js

```

const router = require("express").Router();
const User = require("../models/user");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const {authenticateToken} = require("./userAuth");

```

```

// Sign-up route
router.post("/sign-up", async (req, res) =>
  { try {
    const { username, email, password, address } = req.body;

    // Validate required fields
    if (!username || !email || !password || !address) {
      return res.status(400).json({ message: "All fields are required" });
    }

    // Validate email format
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
      return res.status(400).json({ message: "Invalid email format" });
    }

    // Check if username already exists
    const existingUsername = await User.findOne({ username });
    if (existingUsername) {
      return res.status(400).json({ message: "Username already exists" });
    }

    // Check if email already exists
    const existingEmail = await User.findOne({ email });
    if (existingEmail) {

```

```

        return res.status(400).json({ message: "Email already exists" });
    }

    // Validate password length
    if (password.length <= 5) {
        return res.status(400).json({ message: "Password length should be greater
than 5 characters" });
    }

    // Hash the password
    const hashedPassword = await bcrypt.hash(password, 10);

    // Create and save the user
    const newUser = new User({
        username,
        email,
        password: hashedPassword,
        address,
    });

    await newUser.save();
    return res.status(200).json({ message: "Signup successful" });

} catch (error) {
    res.status(500).json({ message: "Internal server error", error: error.message
});
}

```

```

    }
  });

// Sign-in route
router.post("/sign-in", async (req, res) =>
  { try {
    const { email, password } = req.body;

    // Validate input
    if (!email || !password) {
      return res.status(400).json({ message: "Email and password are required"
});
    }

    // Find user by email
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: "Invalid email or password" });
    }

    // Validate password
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      return res.status(400).json({ message: "Invalid email or password" });
    }
  }

```

```

// Generate JWT token
const token = jwt.sign(
  {
    id: user._id,
    username: user.username,
    role: user.role,
  },
  "bookbazar123", // Use a strong secret key
  { expiresIn: "200d" }
);

// Respond with user data and token
res.status(200).json({
  message: "Sign-in successful",
  user: {
    id: user._id,
    username: user.username,
    email: user.email,
    role: user.role,
  },
  token,
});

} catch (error) {
  res.status(500).json({ message: "Internal server error", error: error.message
});

```

```

    }
  });

//get-user-information
router.get("/get-user-information", authenticateToken, async (req, res)
=>{ try {
    const { id } = req.headers;
    const data = await User.findById(id).select("-password");
    return res.status(200).json(data);

  } catch (error){
    res.status(500).json({ message: "Internal server error" });
  }
});

//update address
router.put("/update-address", authenticateToken, async (req, res)
=>{ try{
    const { id } = req.headers;
    const { address } = req.body;
    await User.findByIdAndUpdate(id, { address: address });
    return res.status(200).json({ message: "Address updated Successfully" });
  } catch (error){
    res.status(500).json({ message: "Internal server error" });
  }
});

module.exports = router;

```


Routes Book.js File

```
const router = require("express").Router();
const User = require("../models/user");
const jwt = require("jsonwebtoken");
const Book = require("../models/book");
const { authenticateToken } = require("../userAuth");

// Add book -- Admin only
router.post("/add-book", authenticateToken, async (req, res) =>
  { try {
    const { id } = req.headers;
    const user = await User.findById(id);
    const book = new Book({
      title: req.body.title,
      author: req.body.author,
      price: req.body.price,
      desc: req.body.desc,
      language: req.body.language,
      bookType: req.body.bookType,
      condition: req.body.condition,
      url: req.body.url,
    });

    await book.save();

    res.status(200).json({ message: "Book added successfully", book });
```

```
    } catch (error) {  
        res.status(500).json({ message: "Internal server error", error: error.message  
});  
    }  
});
```

// update-Books

```
router.put("/update-book", authenticateToken, async (req, res) =>
```

```
    { try {  
        const { bookid } = req.headers;  
  
        await Book.findByIdAndUpdate(bookid, {  
            title: req.body.title,  
            author: req.body.author,  
            price: req.body.price,  
            desc: req.body.desc,  
            language: req.body.language,  
            bookType: req.body.bookType,  
            condition: req.body.condition,  
            url: req.body.url,  
        });
```

```
        // if (!updatedBook) {
```

```
            // return res.status(404).json({ message: "Book not found!" });
```

```
        // }
```

```

    return res.status(200).json({
      message: "Book Updated successfully!",
    });

  } catch (error) {
    res.status(500).json({ message: "Internal server error", error: error.message });
  }
});

```

//delete book

```

router.delete("/delete-book",authenticateToken, async (req, res)
=>{ try{
  const {bookid } = req.headers;
  await Book.findByIdAndDelete(bookid);
  return res.status(200).json({
    message: "Book deleted successfully!",
  });
}catch (error){
  return res.status(500).json({ message: "An error occurred"});
}
});

```

//get all books

```

router.get("/get-all-books", async (req, res) =>{

```

```

try{
  const books = await Book.find().sort({ createdAt: -1});
  return res.json({
    status: "Success",
    data: books,
  });
} catch (error){
  return res.status(500).json({ message: "An error occurred"});
}
});

```

//get recently added books

```

router.get("/get-recent-books", async (req, res)
=>{ try{
  const books = await Book.find().sort({ createdAt: -1}).limit(8);
  return res.json({
    status: "Success",
    data: books,
  });
} catch (error){
  return res.status(500).json({ message: "An error occurred"});
}
});

```

//get book by id

```

router.get("/get-book-by-id/:id", async (req, res) =>{

```

```

try{
  const {id } = req.params;
  const book = await Book.findById(id);
  return res.json({
    status: "Success",
    data: book,
  });
} catch (error){
  return res.status(500).json({ message: "An error occurred"});
}
});

module.exports = router;

```

Routes Cart.js File

```

const router = require("express").Router();
const User = require("../models/user");
const Book = require("../models/book");
const Order = require("../models/order")
const { authenticateToken } = require("../userAuth");

// Add to Cart
router.put("/add-to-cart", authenticateToken, async (req, res) =>
  { try {

```

```

const { bookid, id } = req.headers;

const userData = await User.findById(id);
if (!userData) {
  return
    res.status(404).json({ status:
      "Error",
      message: "User not found",
    });
}

// Check if the book is already in the cart
const isBookInCart = userData.cart.includes(bookid);
if (isBookInCart) {
  return
    res.status(200).json({ status:
      "Success",
      message: "Book is already in the cart",
    });
}

// Add the book to the cart
await User.findByIdAndUpdate(id, {
  $push: { cart: bookid },
});

return

```

```
res.status(200).json({ status  
: "Success",
```

```

        message: "Book added to cart successfully",
    });
} catch (error)
{ console.error(error);
return
    res.status(500).json({ status
        : "Error",
        message: "An error occurred while adding the book to cart",
        error: error.message,
    });
}
});

//remove to cart
router.put("/remove-from-cart", authenticateToken, async (req, res) =>
{ try {
    const { bookid, id } = req.headers;

    const userData = await User.findById(id);
    if (!userData) {
        return
            res.status(404).json({ status:
                "Error",
                message: "User not found",
            });
    }
}

```



```
// Check if the book is in the cart
```

```

const isBookInCart = userData.cart.includes(bookid);
if (!isBookInCart) {
  return
    res.status(400).json({ status:
      "Error",
      message: "Book is not in the cart",
    });
}

// Remove the book from the cart
await User.findByIdAndUpdate(id, {
  $pull: { cart: bookid },
});

return
  res.status(200).json({ status
    : "Success",
    message: "Book removed from cart successfully",
  });
} catch (error)
{ console.error(error);
  return
    res.status(500).json({ status
      : "Error",
      message: "An error occurred while removing the book from the cart",
      error: error.message,
    });
}

```

```
}  
});
```

```

// Get User Cart
router.get("/get-user-cart", authenticateToken, async (req, res) =>
  { try {
    const { id } = req.headers;

    const userData = await User.findById(id)
      .populate({ pa
        th: 'cart',
        model: 'Book',
        select: 'title author price description imageUrl' // Select the fields you
want to return
      });

    if (!userData) {
      return
        res.status(404).json({ status:
          "Error",
          message: "User not found",
        });
    }

    // Reverse the cart order to show newest items first
    const reversedCart = [...userData.cart].reverse();

    return
      res.status(200).json({ status
        : "Success",

```



```

        message: "Cart retrieved successfully",
        data: reversedCart,
    });
} catch (error)
{ console.error(error);
return
    res.status(500).json({ status
    : "Error",
    message: "An error occurred while retrieving the cart",
    error: error.message,
    });
}
});
module.exports = router;

```

Routes Order.js File

```

const router = require("express").Router();
const User = require("../models/user");
const Order = require("../models/order");
const Book = require("../models/book");
const { authenticateToken } = require("../userAuth");

// Place Order Route
router.post("/place-order", authenticateToken, async (req, res) =>
    { try {

```

```

const { id } = req.headers;
const { order } = req.body;

// Validate input
if (!Array.isArray(order) || order.length === 0)
  { return res.status(400).json({
    status: "Error",
    message: "Invalid order data provided.",
  });
}

// Find user to verify existence
const user = await User.findById(id);
if (!user) {
  return
    res.status(404).json({ status:
      "Error",
      message: "User not found",
    });
}

const orderResults = [];

// Process each order item
for (const orderData of order) {
  // Create new order
  const newOrder = new Order({

```

```

        user: id,
        book: orderData._id,
        quantity: orderData.quantity || 1,
        status: 'pending',
        orderDate: new Date()
    });

    const savedOrder = await newOrder.save();
    orderResults.push(savedOrder._id);

    // Update user's orders and cart
    await User.findByIdAndUpdate(id, {
        $push: { orders: savedOrder._id },
        $pull: { cart: orderData._id }
    });
}

return
    res.status(200).json({ status
        : "Success",
        message: "Order placed successfully!",
        orderIds: orderResults
    });

} catch (error) {
    console.error("Error while placing order:", error);
    return res.status(500).json({

```



```

        status: "Error",
        message: "An error occurred while placing the order.",
        error: error.message,
    });
}
});

```

// Get Order History of Particular User

```

router.get("/get-order-history", authenticateToken, async (req, res) =>
{ try {
    const { id } = req.headers;

    const userData = await
        User.findById(id).populate({ path: "orders",
        populate:
            { path:
                "book",
                select: "title author price description imageUrl" // Select specific fields
            }
        });

    if (!userData) {
        return
            res.status(404).json({ status:
                "Error",
                message: "User not found"
            });
    }
} catch (error) {
    res.status(500).json({ status: "Error", message: error.message });
}
});

```

```
});  
}
```

```

    const orderData = userData.orders.reverse();
    return res.status(200).json({
      status: "Success",
      data: orderData,
    });

  } catch (error) {
    console.error("Error while fetching order history:", error);
    return res.status(500).json({
      status: "Error",
      message: "An error occurred while fetching the order history",
      error: error.message,
    });
  }
});

// Get All Orders (Admin)
router.get("/get-all-orders", authenticateToken, async (req, res) =>
{ try {
  const orders = await Order.find()
    .populate({ path
      : "book",
      select: "title author price imageUrl"
    })
    .populate({

```

```

        path: "user",
        select: "username email address"
    })
    .sort({ createdAt: -1 });

    return
    res.status(200).json({ status
    : "Success",
    data: orders
    });

} catch (error) {
    console.error("Error while fetching all orders:", error);
    return res.status(500).json({
        status: "Error",
        message: "An error occurred while fetching orders",
        error: error.message,
    });
}
});

// Update Order Status (Admin)
router.put("/update-status/:id", authenticateToken, async (req, res) =>
{ try {
    const { id } = req.params;
    const { status } = req.body;

```

```
if (!status) {  
  return  
    res.status(400).json({ status:  
      "Error",  
      message: "Status is required"  
    });  
}
```

```
const updatedOrder = await  
  Order.findByIdAndUpdate( id,  
    { status },  
    { new: true }  
  );
```

```
if (!updatedOrder) {  
  return  
    res.status(404).json({ status:  
      "Error",  
      message: "Order not found"  
    });  
}
```

```
return  
  res.status(200).json({ status  
    : "Success",  
    message: "Status Updated Successfully",  
    order: updatedOrder
```

```
});
```

```

    } catch (error) {
      console.error("Error while updating order status:", error);
      return res.status(500).json({
        status: "Error",
        message: "An error occurred while updating the order status",
        error: error.message,
      });
    }
  });
});

module.exports = router;

```

Route Favourite.js File

```

const router = require("express").Router();
const User = require("../models/user");
const Book = require("../models/book");
const { authenticateToken } = require("../userAuth");

// Add book to favourites
router.put("/add-book-to-favourite", authenticateToken, async (req, res) =>
  { try {
    const { bookid, id } = req.headers;
    const userData = await User.findById(id);

```



```

    // Check if the book is already in favourites
    const isBookFavourite = userData.favourites.includes(bookid);
    if (isBookFavourite) {
        return res.status(200).json({ message: "Book is already in favourites." });
    }

    await User.findByIdAndUpdate(id, { $push: { favourites: bookid } });
    return res.status(200).json({ message: "Book added to favourites successfully."
});

} catch (error) {
    // Log the error and send a detailed response
    console.error(error);
    res.status(500).json({ message: "Internal server error", error: error.message
});
}
});

// Remove books from Favourite
router.put("/remove-book-from-favourite", authenticateToken, async (req, res) =>
{
    try {
        const { bookid, id } = req.headers;
        const userData = await User.findById(id);

        // Check if the book is in favourites
        const isBookFavourite = userData.favourites.includes(bookid);

```



```

    if (!isBookFavourite) {
        return res.status(400).json({ message: "Book is not in favourites." });
    }

    // Remove the book from favourites
    await User.findByIdAndUpdate(id, { $pull: { favourites: bookid } });

    return res.status(200).json({ message: "Book removed from favourites successfully." });
} catch (error) {
    // Log the error and send a detailed response
    console.error(error);
    res.status(500).json({ message: "Internal server error", error: error.message
});
}
});

// Get Favourite books of a particular user
router.get("/get-favourite-books", authenticateToken, async (req, res) =>
{ try {
    const { id } = req.headers;

    const userData = await User.findById(id)
        .populate({
            path: 'favourites',
            model: 'Book',

```

```

        select: 'title author price description imageUrl'
    });

    if (!userData) {
        return
        res.status(404).json({ status:
            "Error",
            message: "User not found",
        });
    }

    return
    res.status(200).json({ status
        : "Success",
        data: userData.favourites,
    });
} catch (error)
{ console.error(error);
    return
    res.status(500).json({ status
        : "Error",
        message: "An error occurred while retrieving favourite books",
        error: error.message,
    });
}
});

```

```
module.exports = router;
```

FRONTEND

App.jsx File

```
import React, { useEffect } from "react";

import { Routes, Route } from "react-router-dom";

import { useDispatch, useSelector } from "react-redux";

import { authActions } from "../Store/auth";


// Pages and Components

import Home from "../Pages/Home";

import Navbar from "../Components/Navbar/Navbar";

import Footer from "../Components/Footer/Footer";

import SaleBook from "../Components/SaleBook/SaleBook";

import AllBooks from "../Pages/AllBooks";

import Login from "../Pages/Login";

import SignUp from "../Pages/SignUp";

import Cart from "../Pages/Cart";

import Profile from "../Pages/Profile";

import AboutUs from "../Pages/AboutUs";

import ViewBookDetails from "../Components/ViewBookDetails/ViewBookDetails";


const App = () => {

  const dispatch = useDispatch();

  const role = useSelector((state) => state.auth.role);


  // Persist login state from localStorage
```

```

useEffect(() =>
  { if (
    localStorage.getItem("id") &&
    localStorage.getItem("token") &&
    localStorage.getItem("role")
  ) {
    dispatch(authActions.login());
    dispatch(authActions.changeRole(localStorage.getItem("role")));
  }
}, [dispatch]); // Added `dispatch` to dependencies

return (
  <div id="root" className="flex flex-col min-h-screen">
    <Navbar />
    <main className="flex-grow">
      <Routes>
        <Route exact path="/" element={<Home />} />
        <Route path="/all-books" element={<AllBooks />} />
        <Route path="/about-us" element={<AboutUs />} />
        <Route path="/cart" element={<Cart />} /> { /* Fixed path case */}
        <Route path="/profile" element={<Profile />} /> { /* Fixed path case */}
        <Route path="/signup" element={<SignUp />} />
        <Route path="/login" element={<Login />} />
        <Route path="/view-book-details/:id" element={<ViewBookDetails />} />
        <Route path="/sale-books" element={<SaleBook />} /> { /* Fixed path case
*/}

```

```

        <Route path="*" element={<div>404 - Page Not Found</div>} /> {/*
Fallback route */}

    </Routes>

</main>

<Footer />

</div>

);

};

export default App;

```

Main.jsx File

```

import React from 'react';
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import './index.css';
import App from './App.jsx';
import { BrowserRouter as Router } from 'react-router-dom';
import { Provider } from "react-redux";
import store from "./Store/index.js";

createRoot(document.getElementById('root')).render(

  <StrictMode>

    <Router>

      <Provider store={store}>

        <App />

```



```
    </Provider>

    </Router>

  </StrictMode>

);
```

Components

Navbar

```
import React, { useState } from "react";
import { useSelector } from "react-redux";
import { Link } from "react-router-dom";

const Navbar = () => {
  const [isMenuOpen, setIsMenuOpen] = useState(false); // State to handle menu
  toggle

  const links = [
    {
      title: "Home",
      link: "/",
    },
    {
      title: "About Us",
      link: "/about-us",
    },
    {
      title: "All Books",
```

```

    link: "/all-books",
  },
  {
    title: "Cart",
    link: "/cart",
  },
  {
    title: "Profile",
    link: "/Profile",
  },
];

const isLoggedIn = useSelector((state) => state.auth.isLoggedIn );
if (isLoggedIn === false){
  links.splice(2,3);
}

return (
  <div className="flex bg-zinc-800 text-white px-8 py-4 items-center justify-between">
    <Link to="/" className="flex items-center">
      
      <h1 className="text-2xl font-semibold">BookBazar</h1>

```

</Link>

{/* Desktop View: Nav links and Login/Signup buttons */}

<div className="hidden md:flex items-center gap-4">

<div className="flex gap-5">

{links.map((item, i) => (

<Link

to={item.link}

className="hover:text-blue-500 transition-all duration-300"

key={i}

>

{item.title}

</Link>

)))

</div>

<div className="flex gap-4">

<Link to="/

LogIn"

className="px-2 py-1 border border-blue-500 hover:bg-white hover:text-zinc-800 transition-all duration-300"

>

LogIn

</Link>

<Link to="/

SignUp"

```
        className="px-2 py-1 bg-blue-500 rounded hover:bg-white hover:text-zinc-800 transition-all duration-300"
```

```
>
```

```
    SignUp
```

```
</Link>
```

```
</div>
```

```
</div>
```

```
{/* Mobile View: Hamburger Menu */}
```

```
<div className="md:hidden flex items-center">
```

```
<button
```

```
  className="text-white"
```

```
  onClick={() => setIsMenuOpen(!isMenuOpen)}
```

```
>
```

```
<svg xmlns="http://www.w3.org/2000/"
```

```
  svg" fill="none"
```

```
  viewBox="0 0 24 24"
```

```
  stroke="currentColor"
```

```
  className="w-6 h-6"
```

```
>
```

```
<path
```

```
  strokeLinecap="round"
```

```
  strokeLinejoin="round"
```

```
  strokeWidth="2"
```

```
  d="M4 6h16M4 12h16M4 18h16"
```

```

    />
  </svg>
</button>
</div>

{/* Mobile Menu */}
{isMenuOpen && (
  <div className="absolute top-16 left-0 w-full bg-zinc-800 text-white px-6 py-4 md:hidden">
    <div className="flex flex-col items-center gap-4">
      {links.map((item, i) => (
        <Link
          to={item.link}
          className="hover:text-blue-500 transition-all duration-300"
          key={i}
          onClick={() => setIsMenuOpen(false)} // Close menu on link click
        >
          {item.title}
        </Link>
      ))}
    <div className="flex flex-col gap-4">
      <Link to="/"
        LogIn"
        className="px-2 py-1 border border-blue-500 hover:bg-white hover:text-zinc-800 transition-all duration-300"
        onClick={() => setIsMenuOpen(false)}

```

```

    >
      LogIn
    </Link>
    <Link to="/"
      SignUp"
      className="px-2 py-1 bg-blue-500 rounded hover:bg-white hover:text-
zinc-800 transition-all duration-300"
      onClick={() => setIsMenuOpen(false)}
    >
      SignUp
    </Link>
  </div>
</div>
</div>
)}
</div>
);
};

```

```
export default Navbar;
```

Footer.jsx File

```
import React from "react";
```

```

const Footer = () => {
  const quickLinks = [

```

```

    { title: "Home", link: "/" },
    { title: "About Us", link: "/about-us" },
    { title: "All Books", link: "/all-books" },
    { title: "Contact Us", link: "/contact-us" },
    { title: "FAQ", link: "/faq" },
  ];

```

```

const policyLinks = [
  { title: "Privacy Policy", link: "/privacy-policy" },
  { title: "Terms of Service", link: "/terms-of-service" },
  { title: "Refund Policy", link: "/refund-policy" },
  { title: "Shipping Info", link: "/shipping-info" },
];

```

```

return (
  <div className="bg-blue-200 h-screen flex flex-col">
    <footer className="bg-gradient-to-b from-blue-900 to-blue-950 text-white mt-
auto">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-12">
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8">
          { /* Logo and Description */ }
          <div className="space-y-4">
            <div className="flex items-center space-x-3">
              

    <h2 className="text-2xl font-bold bg-gradient-to-r from-white to-gray-
300 text-transparent bg-clip-text">
        BookBazar
    </h2>
</div>

<p className="text-gray-300 text-sm leading-relaxed">
    Discover a world of second-hand books, save money, and connect with
fellow book lovers.

    Buy, sell, and share your favorite reads!
</p>
</div>

{ /* Quick Links */ }
<div className="space-y-4">
    <h3 className="text-lg font-semibold text-white">Quick Links</h3>
    <ul className="space-y-2">
        {quickLinks.map((item, index) => (
            <li key={index}>
                <a
                    href={item.link}
                    className="text-gray-300 hover:text-white transition-colors duration-
300 flex items-center space-x-2"
                >
                    <span className="h-1 w-1 bg-blue-400 rounded-full"></span>
                    <span>{item.title}</span>

```



```

        </a>
      </li>
    )})
  </ul>
</div>

```

```

{ /* Policies */ }

<div className="space-y-4">
  <h3 className="text-lg font-semibold text-white">Our Policies</h3>
  <ul className="space-y-2">
    {policyLinks.map((item, index) => (
      <li key={index}>
        <a
          href={item.link}
          className="text-gray-300 hover:text-white transition-colors duration-
300 flex items-center space-x-2"
        >
          <span className="h-1 w-1 bg-blue-400 rounded-full"></span>
          <span>{item.title}</span>
        </a>
      </li>
    )})
  </ul>
</div>

```

```

{ /* Newsletter */ }

```

```

<div className="space-y-4">
  <h3 className="text-lg font-semibold text-white">Stay Updated</h3>
  <p className="text-sm text-gray-300">
    Subscribe to our newsletter for the latest updates and deals.
  </p>
  <div className="flex flex-col space-y-2">
    <input
      type="email"
      placeholder="Enter your email"
      className="px-4 py-2 bg-blue-800/50 rounded-lg focus:outline-none
focus:ring-2
      focus:ring-blue-400 text-white placeholder-gray-400"
    />
    <button className="px-4 py-2 bg-blue-500 text-white rounded-lg
hover:bg-blue-600
      transition-colors duration-300 transform hover:scale-105">
      Subscribe
    </button>
  </div>

  { /* Contact Info */ }
  <div className="pt-4 space-y-2">
    <p className="text-sm text-gray-300">
      <span className="font-semibold text-white">Email:</span>
support@bookbazar.com
    </p>
    <p className="text-sm text-gray-300">

```

```
7644808208      <span className="font-semibold text-white">Phone:</span> +91
```

```
      </p>
    </div>
  </div>
</div>
```

```
{/* Bottom Section */}

<div className="mt-12 pt-8 border-t border-blue-800">

  <div className="flex flex-col md:flex-row justify-between items-center
space-y-4 md:space-y-0">

    <p className="text-sm text-gray-400">

      © {new Date().getFullYear()} BookBazar. All rights reserved.

    </p>
```

```
{/* Social Links */}

<div className="flex space-x-6">

  {[ 'Facebook', 'Twitter', 'Instagram', 'LinkedIn' ].map((social) => (

    <a

      key={social}

      href="#"

      className="text-gray-400 hover:text-white transition-colors duration-

300

        transform hover:scale-110"

    >

      <span className="sr-only">{social}</span>

      <i className={ ` fab fa-${social.toLowerCase()} ` }></i>
```



```
</h1>
```

```
<p className="text-lg lg:text-xl text-gray-700 max-w-lg">
```

Join a community of book enthusiasts! Find incredible deals on pre-loved books, or sell your collection with ease. Your next story awaits.

```
</p>
```

```
<div className="mt-6 flex flex-col sm:flex-row gap-4">
```

```
<Link
```

```
to="/all-books"
```

```
className="bg-blue-600 text-white text-lg font-semibold px-8 py-4 rounded-lg hover:bg-blue-700 transition-all shadow-lg"
```

```
>
```

Buy Books

```
</Link>
```

```
<Link
```

```
to="/sale-books"
```

```
className="bg-green-500 text-white text-lg font-semibold px-8 py-4 rounded-lg hover:bg-green-600 transition-all shadow-lg"
```

```
>
```

Sell Books

```
</Link>
```

```
</div>
```

```
</div>
```

```
{/* Right Section - Image */}
```

```
<div className="w-full lg:w-1/2 flex items-center justify-center mt-10 lg:mt-0 relative">
```

```

    <div className="absolute top-0 left-0 w-full h-full bg-gradient-to-b from-blue-300 to-transparent opacity-30 rounded-lg"></div>

    
  </div>
</div>

);
};

export default Hero;

```

RecentlyAdded.jsx

```

import React, { useEffect, useState } from "react";
import axios from "axios";
import BookCard from "../BookCard/BookCard";
import Loader from "../Loader/Loader";

const RecentlyAdded = () => {
  const [data, setData] = useState([]); // Initialize with an empty array
  const [loading, setLoading] = useState(true); // Track loading state

  useEffect(() => {

```

```

const fetch = async () => {
  try {
    const response = await axios.get( "http://
      localhost:3000/api/v1/get-recent-books"
    );
    setData(response.data.data || []); // Fallback to an empty array
  } catch (error) {
    console.error("Error fetching recently added books:", error);
  } finally {
    setLoading(false); // Ensure loader stops
  }
};

fetch();
}, []);

return (
  <div
    className="min-h-screen bg-gradient-to-b from-gray-100 via-gray-200 to-
gray-300 px-6 py-12"
    style={{
      backgroundImage: `url('https://www.transparenttextures.com/patterns/white-
diamond.png')`,
      backgroundSize: "cover",
    }}
  >
    <h4 className="text-4xl font-extrabold text-gray-900 mb-12 text-center">

```

Recently Added Books

</h4>

```
{/* Loader */}
```

```
{loading && (
```

```
  <div className="flex items-center justify-center my-12">
```

```
    <Loader />
```

```
  </div>
```

```
)}
```

```
{/* No Data Message */}
```

```
{!loading && data.length === 0 && (
```

```
  <p className="text-center text-gray-500 my-12 text-lg">
```

```
    No books added recently. Please check back later!
```

```
  </p>
```

```
)}
```

```
{/* Books Grid */}
```

```
  <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4  
gap-8">
```

```
    {data.map((item, i) => (
```

```
      <div
```

```
        key={i}
```

```
        className="bg-white rounded-lg shadow-lg hover:shadow-2xl transition-  
shadow duration-300 p-4"
```

```
      >
```



```

        <BookCard data={item} />
      </div>
    )}
  </div>
</div>
);
};

```

```
export default RecentlyAdded;
```

BookCard.jsx

```

import React from "react";
import { Link } from "react-router-dom";

const BookCard = ({ data }) => {
  // Dummy image if no image URL is provided
  const defaultImage = "https://via.placeholder.com/150x200?text=No+Image";

  return (
    <>
      <Link to={` /view-book-details/${data._id}`} >
        <div className="bg-white rounded-lg shadow-lg overflow-hidden transform
transition-transform duration-300 hover:scale-105">
          { /* Book Image */ }
          <div className="h-64 bg-gray-100">
            <img

```

```

        src={data.url || defaultImage}
        alt={data.title || "Book Cover"}
        className="w-full h-full object-cover"
      />
    </div>

    { /* Book Details */ }
    <div className="p-4 space-y-2">
      <h2 className="text-lg font-bold text-gray-800 truncate">
        {data.title || "Untitled"}
      </h2>
      <p className="text-sm text-gray-600">by {data.author || "Unknown
Author"}</p>
      <p className="text-xl font-semibold text-blue-500">
        ₹{data.price ? data.price.toFixed(2) : "N/A"}
      </p>
    </div>
  </div>
</Link>
</>
);
};
export default BookCard;

```

SaleBook.jsx

```
import React, { useState } from "react";
```

```

import axios from "axios";

const SaleBook = () => {
  const [formData, setFormData] =
    useState({ title: "",
      author: "",
      price: "",
      desc: "",
      language: "",
      bookType: "",
      condition: "",
      url: "",
    });

  const [loading, setLoading] = useState(false);
  const [message, setMessage] = useState("");

  // Handle form input changes
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  // Handle form submission
  const handleSubmit = async (e) =>
    { e.preventDefault();

```

```

setLoading(true);
setMessage("");

try {
  const token = localStorage.getItem("token");
  const userId = localStorage.getItem("userId");

  const response = await
    axios.post( "http://localhost:3000/api/
v1/add-book", formData,
    {
      headers: {
        Authorization: `Bearer ${token}`,
        id: userId,
      },
    }
  );

  setMessage(response.data.message);
  setFormData({
    title: "",
    author: "",
    price: "",
    desc: "",
    language: "",
    bookType: "",
  });
}

```

```

        condition: "",
        url: "",
    });
} catch (error) {
    console.error("Error adding book:", error);
    setMessage(error.response?.data?.message || "Failed to add the book.");
} finally
    { setLoading(false)
    ;
    }
};

return (
    <div className="min-h-screen bg-gradient-to-b from-blue-100 to-blue-200 py-
12 px-6 flex items-center justify-center">
        <div className="bg-white shadow-lg rounded-lg p-8 max-w-lg w-full">
            <h2 className="text-2xl font-bold text-gray-800 text-center mb-6">
                Add a Book for Sale
            </h2>
            {message && (
                <p
                    className={`text-center mb-4 $
                        { message.includes("successfully")
                            ? "text-green-500"
                            : "text-red-500"
                        }` }
                >

```

```

    {message}
  </p>
)}
<form onSubmit={handleSubmit} className="space-y-4">
  <input
    type="text"
    name="title"
    value={formData.title}
    onChange={handleChange}
    placeholder="Book Title"
    className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
    required
  />
  <input
    type="text"
    name="author"
    value={formData.author}
    onChange={handleChange}
    placeholder="Author"
    className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
    required
  />
  <input
    type="number"

```

```

    name="price"
    value={formData.price}
    onChange={handleChange}
    placeholder="Price (in USD)"
    className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
    required
  />
  <textarea
    name="desc"
    value={formData.desc}
    onChange={handleChange}
    placeholder="Description"
    rows="4"
    className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
    required
  ></textarea>
  <input
    type="text"
    name="language"
    value={formData.language}
    onChange={handleChange}
    placeholder="Language"
    className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
    required

```

```

/>
<input
  type="text"
  name="bookType"
  value={formData.bookType}
  onChange={handleChange}
  placeholder="Book Type (e.g., Hardcover, Paperback)"
  className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
  required
/>
<input
  type="text"
  name="condition"
  value={formData.condition}
  onChange={handleChange}
  placeholder="Condition (e.g., New, Like New, Used)"
  className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
  required
/>
<input
  type="text"
  name="url"
  value={formData.url}
  onChange={handleChange}

```



```

        placeholder="Image URL"

        className="w-full p-3 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"

    />

    <button

        type="submit"

        className="w-full bg-blue-500 hover:bg-blue-600 text-white p-3 rounded
transition-all duration-300"

        disabled={loading}

    >

        {loading ? "Adding..." : "Add Book"}

    </button>

</form>

</div>

</div>

);

};

export default SaleBook;

```

Pages

AllBooks

```

import React, { useEffect, useState } from "react";

import axios from "axios";

import Loader from "../Components/Loader/Loader";

import BookCard from "../Components/BookCard/BookCard";

```

```

const AllBooks = () => {
  const [data, setData] = useState([]); // Initialize with an empty array
  const [loading, setLoading] = useState(true); // Track loading state

  useEffect(() => {
    const fetchBooks = async () =>
    { try {
      const response = await axios.get("http://localhost:3000/api/v1/get-all-books");
      setData(response.data.data || []); // Fallback to an empty array
    } catch (error) {
      console.error("Error fetching books:", error);
    } finally {
      setLoading(false); // Ensure loader stops
    }
  };
  fetchBooks();
}, []);

  return (
    <div className="min-h-screen px-6 py-8 bg-gradient-to-b from-blue-100 via-white to-blue-100">
      <h4 className="text-4xl font-extrabold text-blue-800 text-center mb-12">
        All Books
      </h4>

```

```

    {loading ? (
      <div className="flex items-center justify-center my-12">
        <Loader />
      </div>
    ) : data.length === 0 ? (
      <p className="text-center text-gray-600 my-12 text-lg">
        No books found. Please check back later!
      </p>
    ) : (
      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4
gap-8">
        {data.map((item, i) => (
          <div
            key={i}
            className="transform transition-transform duration-300 hover:scale-105"
          >
            <BookCard data={item} />
          </div>
        ))}
      </div>
    )}
  </div>
);
};

export default AllBooks;

```

Home.jsx

```
import React from "react";
import Hero from "../Components/Home/Hero";
import RecentlyAdded from "../Components/Home/RecentlyAdded";
// import Footer from "../Components/Footer/Footer";

const Home = () =>
  { return (
    <div className="bg-blue-200 h-screen flex flex-col">
      <main className="flex-grow">
        <Hero />
        <RecentlyAdded />
      </main>
      { /* <Footer/> */ }
    </div>
  );
};

export default Home;
```

Package.json

```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
```

```

"type": "module",
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "lint": "eslint .",
  "preview": "vite preview"
},
"dependencies": {
  "@reduxjs/
  toolkit": "^2.3.0", "axios":
  "^1.7.7",
  "react": "^18.3.1",
  "react-dom": "^18.3.1",
  "react-icons": "^5.3.0",
  "react-redux": "^9.1.2",
  "react-router-dom": "^6.27.0"
},
"devDependencies":
{ "@eslint/js": "^9.13.0",
"@types/react": "^18.3.12",
"@types/react-dom": "^18.3.1",
"@vitejs/plugin-react": "^4.3.3",
"autoprefixer": "^10.4.20",
"eslint": "^9.13.0",
"eslint-plugin-react": "^7.37.2",
"eslint-plugin-react-hooks": "^5.0.0",
"eslint-plugin-react-refresh": "^0.4.14",

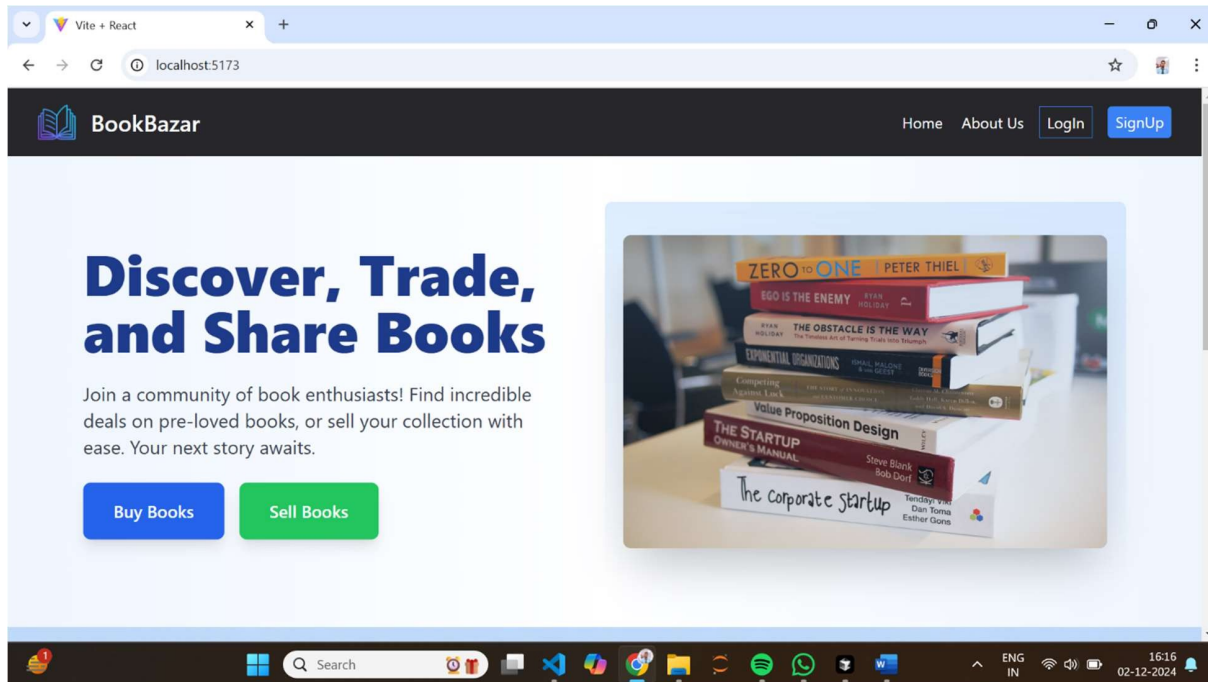
```

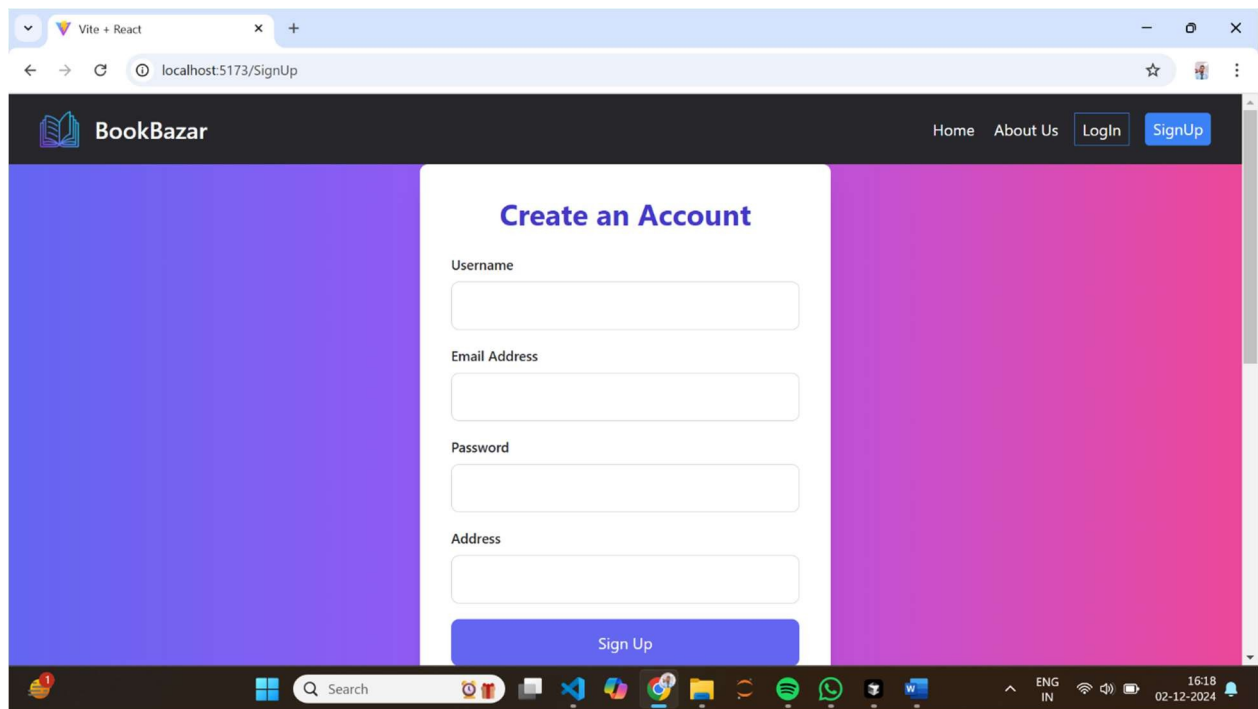
```
"globals": "^15.11.0",  
"postcss": "^8.4.47",  
"tailwindcss": "^3.4.14",  
"vite": "^5.4.10"  
}  
}
```

RESULTS AND DISCUSSIONS

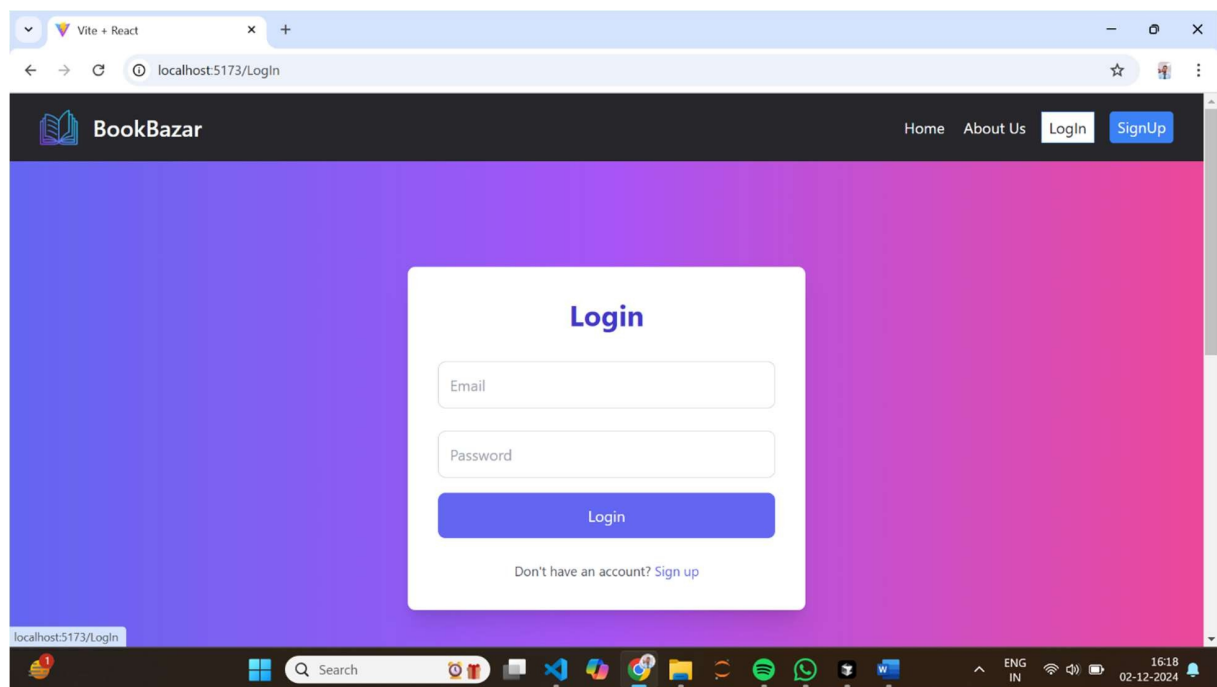
THE GUI:

Home Page

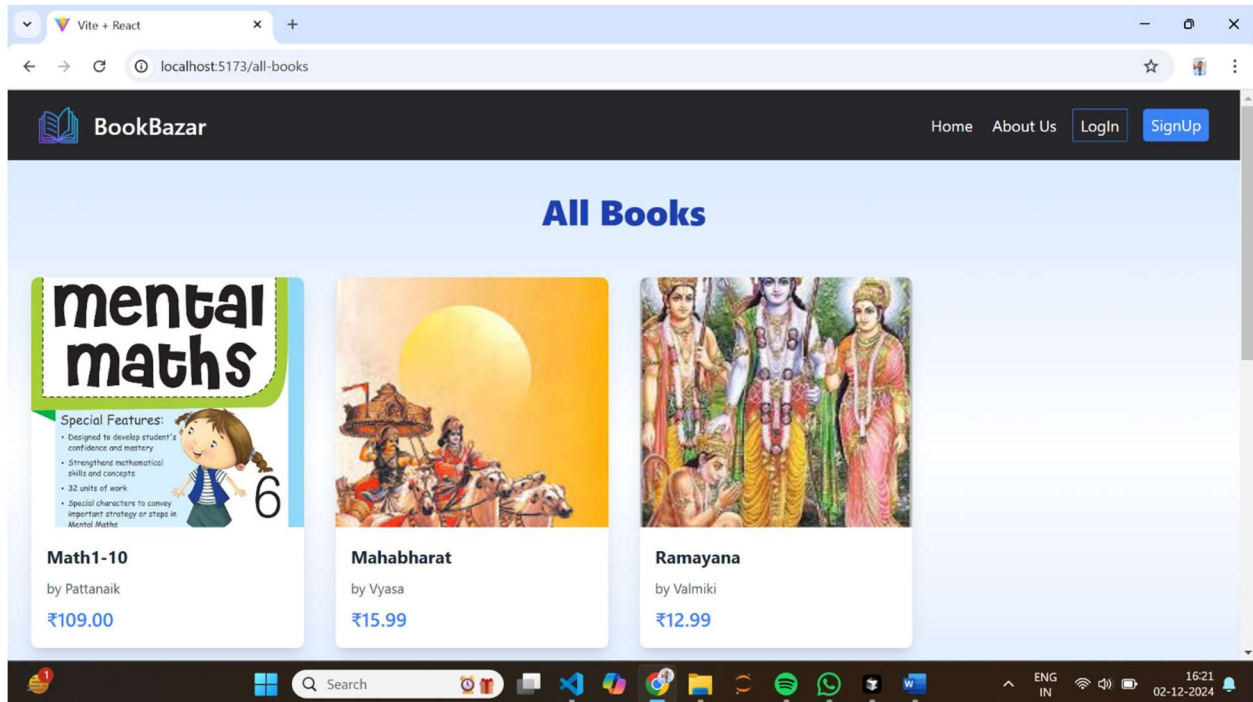




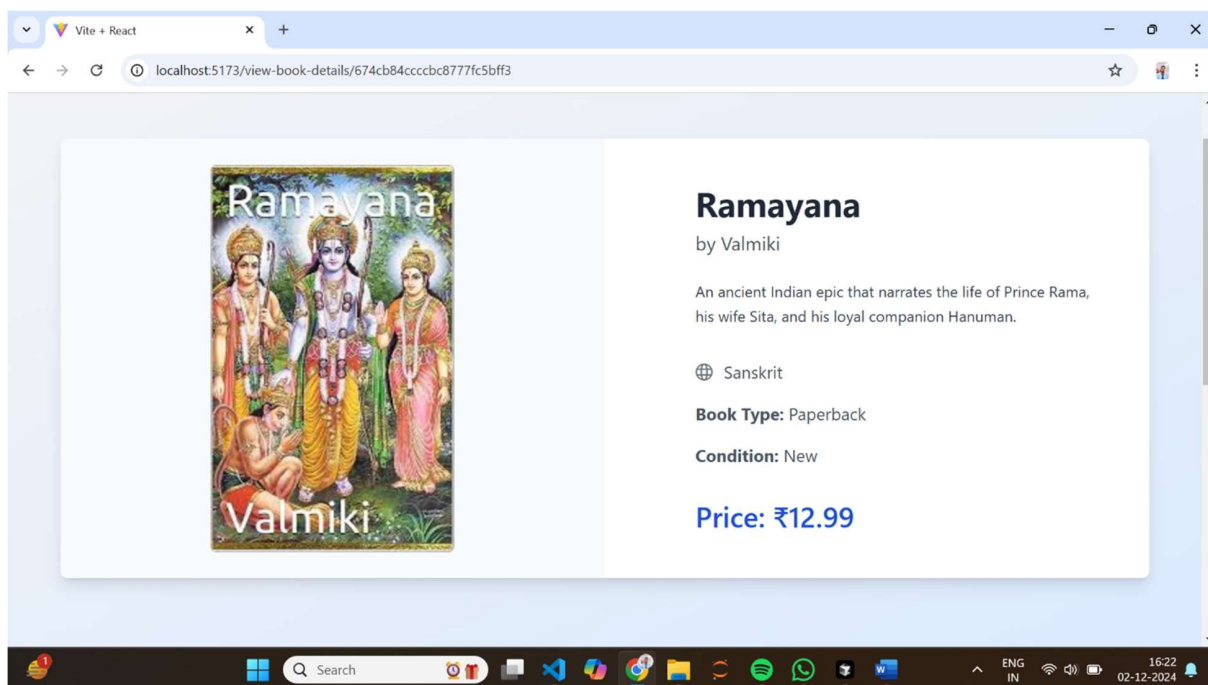
Login Page



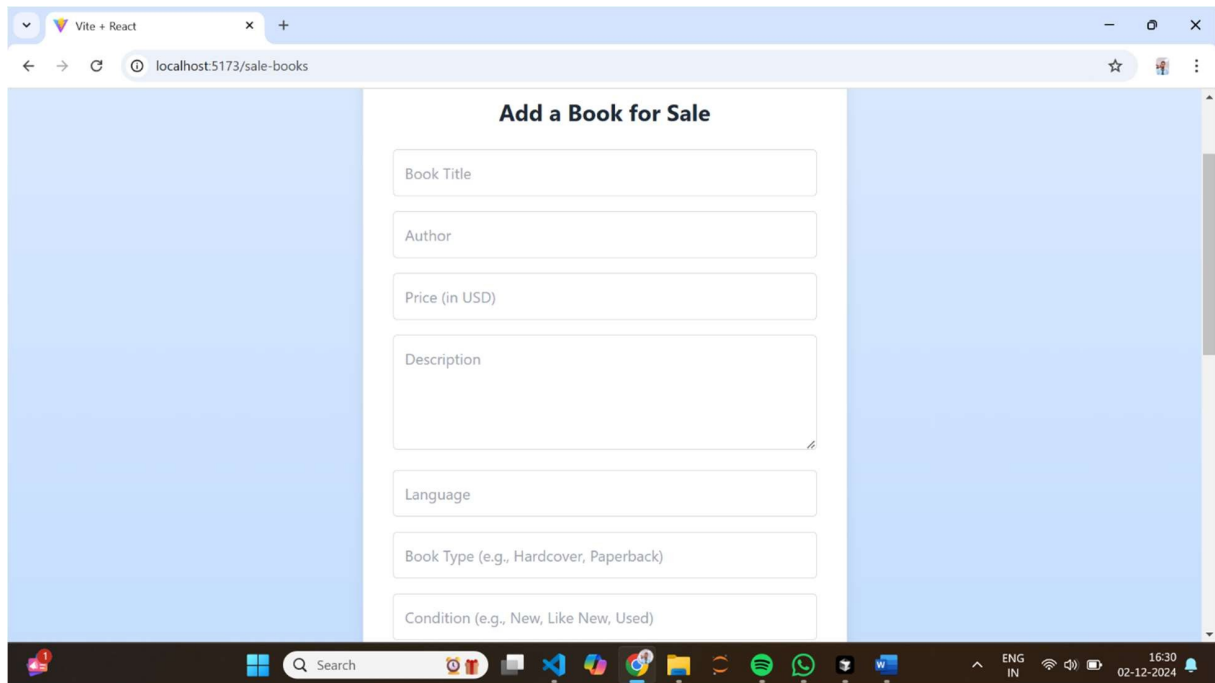
All Books Pages



Book Details Pages



Sale Book Pages



The screenshot shows a web browser window with the address bar displaying 'localhost:5173/sale-books'. The page title is 'Vite + React'. The main content area features a form titled 'Add a Book for Sale' with the following fields:

- Book Title
- Author
- Price (in USD)
- Description
- Language
- Book Type (e.g., Hardcover, Paperback)
- Condition (e.g., New, Like New, Used)

The Windows taskbar at the bottom shows the time as 16:30 on 02-12-2024, with system settings for ENG IN, Wi-Fi, and battery level.

FUTURE WORK

The **Book Bazar** platform has immense potential for growth and enhancement. The following points highlight the future scope of the project:

- Implementing a machine learning-based recommendation system to suggest books to users based on their past searches, purchases, and preferences.
- Expanding the chatbot functionality to include real-time human chat support for resolving complex queries.
- Incorporating AI-based image recognition to verify the condition of books uploaded for sale.
- Allowing sellers to set dynamic pricing based on market demand and supply trends.
- Developing a mobile app version of **Book Bazar** for Android and iOS users to provide a seamless experience across devices.
- Adding a dark mode feature to improve user comfort during nighttime browsing.
- Collaborating with book publishers to allow users to purchase new books directly through the platform.
- Supporting multiple languages and currencies to make the platform accessible globally.

CONCLUSION

The **Book Bazar** project demonstrates the successful integration of modern web development technologies to create a dynamic platform where users can buy and sell books with ease. By leveraging the MERN stack, the project ensures a seamless user experience through a robust frontend, efficient backend, and a secure database system. The incorporation of a Rasa chatbot further enhances user interaction, providing real-time assistance and improving platform accessibility.

Throughout the development process, challenges such as ensuring secure data handling, optimizing user flows, and integrating advanced features were systematically addressed, resulting in a scalable and user-friendly application.

This project not only solves the problem of connecting book buyers and sellers but also introduces innovative features like personalized recommendations and interactive chat support. It bridges the gap between traditional book exchanges and modern e-commerce platforms, making it highly relevant in today's digital era.

The work done so far lays a solid foundation for future enhancements, including AI-driven functionalities, mobile app development, and internationalization. As the platform evolves, it has the potential to become a comprehensive marketplace for all book-related needs, fostering a vibrant community of readers and sellers.

In conclusion, **Book Bazar** stands as a testament to the power of full-stack development and collaborative innovation, addressing a meaningful need in a digital-first world while leaving room for continual growth and improvement

REFERENCES

1. **ReactJS Official Documentation:** Comprehensive guide on building user interfaces using ReactJS. <https://reactjs.org/docs/getting-started.html>
2. **Node.js Documentation:** Insight into server-side JavaScript runtime environment for building scalable applications. <https://nodejs.org/en/docs>
3. **Express.js Guide:** Tutorials and resources for using the Express framework in web applications. <https://expressjs.com/>
4. **MongoDB Official Documentation:** Reference material for managing and interacting with MongoDB databases. <https://www.mongodb.com/docs/>
5. **MERN Stack Tutorials:** Step-by-step tutorials for developing full-stack applications using MongoDB, Express, React, and Node.js. <https://www.mongodb.com/mern-stack>
6. **Rasa Documentation:** Guidelines and tutorials for creating AI-powered conversational agents. <https://rasa.com/docs/>
7. **TailwindCSS Documentation:** Resources for implementing modern and responsive UI designs with TailwindCSS. <https://tailwindcss.com/docs>
8. **OpenAI Rasa Chatbot Integration:** Use cases and examples of integrating Rasa with full-stack applications. <https://blog.rasa.com/>

