MERN STACK POWERED BY MONGODB

SHOPEZ:E-Commerce Application

A PROJECT REPORT

Submitted by

JANANI M A	**113321104030**
MANCHU PALLAVI	**113321104055**
NIKSHITHA PRINCEE A	**113321104067**
REKHA R D	**113321104081**
UKKISALA LAKSHMI	**113321104104**

BACHELOR OF ENGINEERING

COMPUTER SCIENCE AND ENGINEERING

VELAMMAL INSTITUTE OF TECHNOLOGY

CHENNAI 601 204

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **“SHOPEZ:E-Commerce Application”** is the Bonafide work of **“JANANI M A -113321104030, MANCHU PALLAVI-113321104055, NIKSHITHA PRINCEEE A-113321104067, REKHA R D-113321104081,UKKISALA LAKSHMI-113321104104”** who carried out the project work under my supervision .

SIGNATURE
Dr.V.P.Gladis Pushaparathi,
Professor,
Head of the Department,
Computer Science and Engineering,
Velammal Institute of Technology,
Velammal Knowlegde Park,
Panchetti, Chennai-601 204.

SIGNATURE
Mrs.Joice Ruby J
Assistant Professor,
NM Coordinator,
Computer Science and Engineering,
Velammal Institute of Technology,
Velammal Knowlegde Park,
Panchetti, Chennai-601 204.

ACKNOWLEDGEMENT

We are personally indebted to many who had helped me during the course of this project work. My deepest gratitude to the God Almighty.

We are greatly and profoundly thankful to our beloved Chairman **Thiru.M.V.Muthuramalingam** for facilitating us with this opportunity. My sincere thanks to our respected Director **Thiru.M.V.M Sasi Kumar** for his consent to take up this project work and make it great success

We are also thankful to our Advisor **Shri.K.Razak, M.Vaasu** and our Principal **Dr.N.Balaji** and our Vice Principal **Dr.S.Soundararajan** for their never ending encouragement which accelerates us towards innovation.

We are extremely thankful to our Head of the Department **Dr.V.P.Gladis Pushaparathi**, Internship coordinator **Mrs. Joice Ruby J** for their valuable teachings and suggestions.

The Acknowledgement would be incomplete if we would not mention a word of thanks to my Parents, Teaching and Non-Teaching Staffs, Administrative Staffs, Friends who had motivated and lent their support throughout the project.

Finally, we thank all those who directly or indirectly contributed to the successful completion of this project .Your contributions have been a vital part of our success

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

ShopEZ is a cutting-edge e-commerce application designed to provide a seamless and engaging online shopping experience. Developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js), this project demonstrates the integration of modern web technologies to build a fully functional and scalable e-commerce platform.

At its core, ShopEZ focuses on delivering an intuitive user experience, enabling users to explore and purchase products with ease. The application supports secure user authentication using JSON Web Tokens (JWT), ensuring that customer data remains safe and protected. With role-based access control, administrators can efficiently manage products, categories, and orders, while customers enjoy features like personalized profiles and dynamic shopping carts.

ShopEZ features a well-organized product catalog with advanced search and filter capabilities, making it easy for users to discover items based on their preferences. The platform supports essential e-commerce functionality such as order placement, real-time inventory updates, and a streamlined checkout process.

On the backend, MongoDB serves as a robust database for managing product information, user data, and order details. Express.js and Node.js provide a fast and scalable API layer, while React.js powers a responsive and interactive frontend interface.

By implementing ShopEZ, developers gain hands-on experience in building a full-stack application that combines functionality, performance, and design into a modern e-commerce solution.

CHAPTER 2

PROJECT OVERVIEW

Our basic ShopEZ app! Our app is designed to provide a seamless online shopping experience for customers, making it convenient for them to explore and purchase a wide range of products. Whether you are a tech enthusiast, a fashionista, our app has something for everyone.

PURPOSE AND GOALS

The primary purpose of the ShopEZ project is to create a modern, scalable, and user-friendly e-commerce application that demonstrates the seamless integration of the MERN (MongoDB, Express.js, React.js, Node.js) stack. It aims to provide an end-to-end solution for online shopping, showcasing the capabilities of full-stack development while addressing the needs of both customers and administrators.

For customers, the goal is to deliver a smooth and engaging shopping experience. This includes intuitive navigation, advanced product search and filtering, secure account management, and a simple checkout process. ShopEZ emphasizes user-centric design, ensuring that customers can effortlessly browse products, manage their carts, and place orders in a responsive and visually appealing interface.

For administrators, the application offers robust tools for managing products, categories, and customer orders. The admin dashboard ensures efficient inventory management and order tracking, facilitating smooth business operations.

From a development perspective, ShopEZ aims to demonstrate best practices in web application architecture, focusing on performance, scalability, and maintainability. By leveraging MongoDB for database management, Express.js and Node.js for backend services, and React.js for a dynamic frontend, the project highlights the strengths of the MERN stack in building modern web applications.

Overall, ShopEZ serves as both a practical e-commerce platform and a comprehensive learning experience in full-stack development.

.FEATURES

1. **User Registration and Authentication**:
   - Secure sign-up and login process using JWT authentication.
   - Social media login options for faster onboarding.
2. **Product Browsing and Search**:
   - Explore items by category, such as Fruits, Vegetables, Beverages, and more.
   - Advanced search functionality with filters for price range, ratings, and brands.
3. **Cart and Wishlist**:
   - Add items to the shopping cart for immediate purchase.
   - Save favorite items to the wishlist for future purchases.
4. **Checkout and Payments**:
   - Multiple payment options, including credit/debit cards, UPI, and digital wallets.
   - Integration with payment gateways like PayPal and Razorpay for secure transactions.
5. **Order Management**:
   - Real-time order tracking from placement to delivery.
   - Notifications for order status updates.
6. **Admin Dashboard**:
   - Tools for adding, updating, or removing products.
   - Monitor user activity and manage orders effectively.
7. **Data Analytics**:
   - Visual insights for admins, including sales trends, inventory usage, and customer preferences.
8. **Mobile Responsiveness**:
   - Fully responsive design, ensuring seamless usability across desktops, tablets, and smartphones.

Unique Selling Points:

- **Personalized Experience**: AI-based recommendation engine for suggesting products based on user behavior.
- **User-Friendly Interface**: Intuitive design with easy navigation for all age groups.
- **Efficiency**: Swift loading times and minimal latency to enhance user experience.
- **Security**: End-to-end encryption for secure user data handling and payment processing.

Target Audience:

- Busy professionals seeking convenience in shopping.
- Retail grocery stores aiming to digitize their business operations. Customers in urban and semi-urban areas with internet access and mobile devices

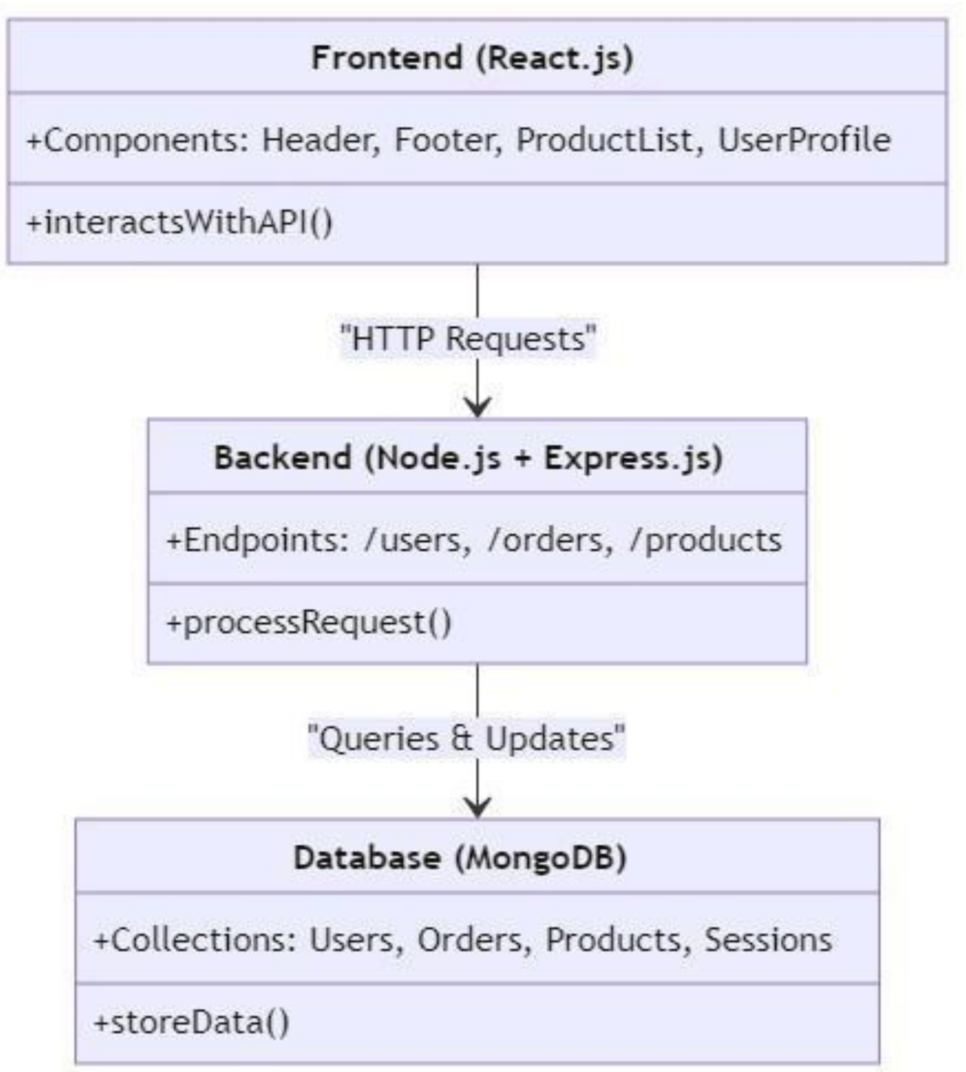
Business Benefits:

- Expanded market reach for stores.
- Improved inventory and order management through centralized dashboards.
- Enhanced customer satisfaction leading to repeat business.

CHAPTER 3

ARCHITECTURE

The architecture of the ShopEZ E-commerce App is designed to be scalable, efficient, and user-friendly. It follows a **MERN (MongoDB, Express.js, React.js, Node.js)** stack to create a full-stack application with a focus on modularity and performance. The architecture is divided into three main components: **Frontend**, **Backend**, and **Database**, with seamless communication between them.

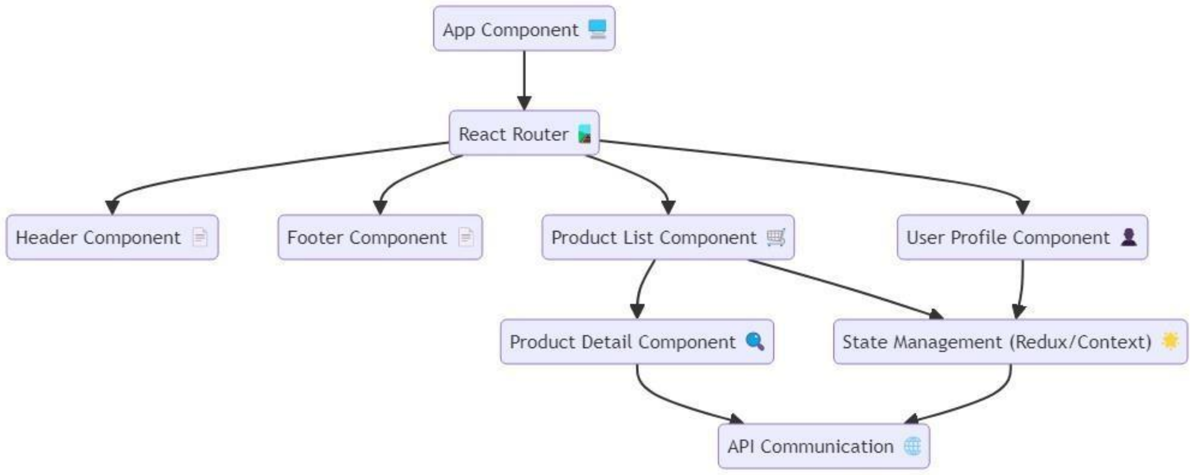

Frontend (React.js)
+Components: Header, Footer, ProductList, UserProfile
+interactsWithAPI()
"HTTP Requests"
Backend (Node.js + Express.js)
+Endpoints: /users, /orders, /products
+processRequest()
"Queries & Updates"
Database (MongoDB)
+Collections: Users, Orders, Products, Sessions
+storeData()

FRONT END

The frontend is built using **React.js**, a JavaScript library known for its high performance and reusable components. The design emphasizes a responsive and intuitive user interface for easy navigation

Key Features:

- **Component-Based Design**:
  Each UI element, such as the product list, shopping cart, and user profile, is implemented as an independent, reusable component.
  - Example Components:
    - ProductCard for displaying individual products.
    - Cart for managing items added by users.
    - Navbar for seamless navigation across pages.
- **State Management**:
  - **Redux** or React Context API is used to manage global states, such as user authentication, cart data, and order status.
  - Ensures smooth data flow across components without redundancy.
- **Routing**:
  - **React Router** is used for efficient navigation between pages (e.g., Home, Product Details, Checkout).
  - Implements dynamic routes for viewing individual product details.
- **Styling and Responsiveness**:
  - CSS frameworks like **Bootstrap** and custom SCSS are used for styling.
  - Ensures mobile-first design for optimal performance across devices.

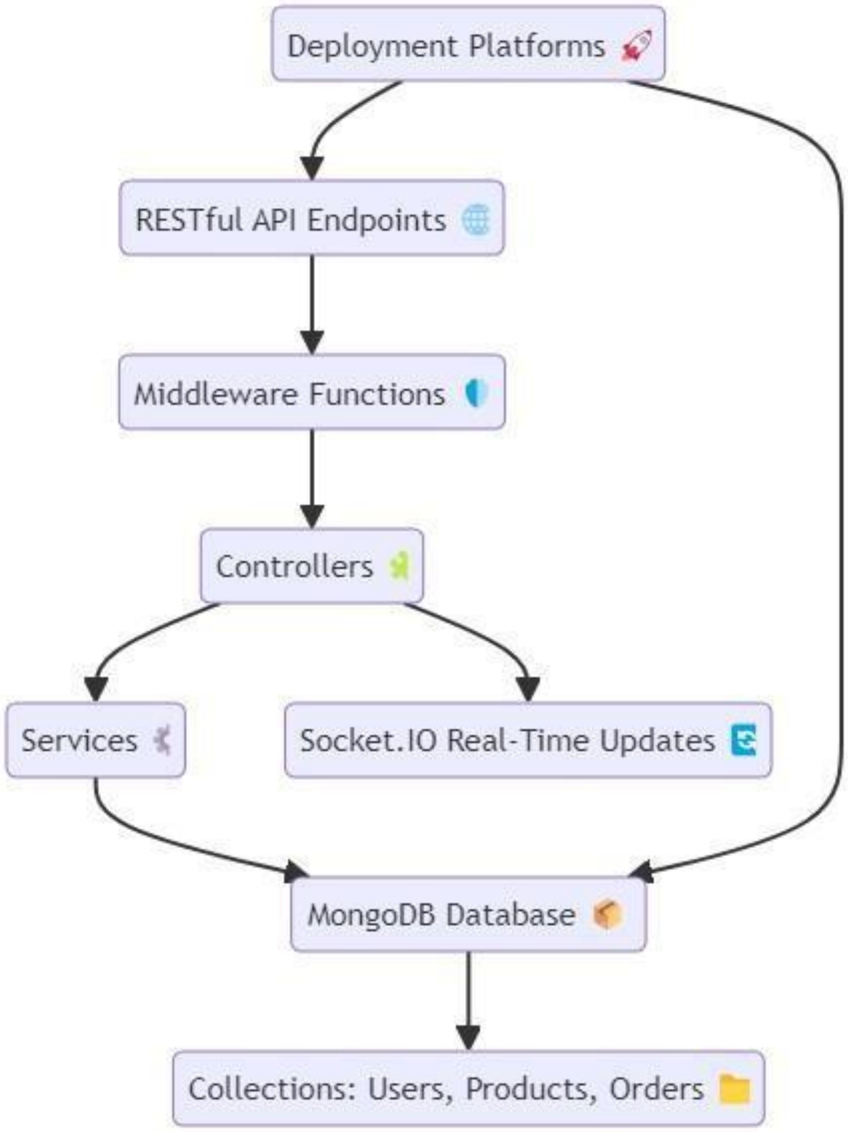


BACK END

The backend is powered by **Node.js** and **Express.js**, providing a robust server-side framework to handle business logic, data management, and communication with the frontend.

Key Features:

- **API-Driven Design**:
  RESTful APIs are designed for each functionality, enabling seamless interaction between the frontend and backend.
  - Example Endpoints:
    - POST /api/register for user registration.
    - GET /api/products for fetching all products.
    - POST /api/orders for placing an order.
- **Middleware**:
  - Middleware functions are used to handle tasks like authentication, request validation, and error handling.
  - Example: A JWT-based middleware ensures that only authenticated users can access protected routes.
- **Scalability**:
  - Modular structure with separate folders for routes, controllers, and services.
  - This separation of concerns makes it easy to scale the application by adding new features.
- **Real-Time Updates**:
  - Integrates **Socket.IO** for real-time notifications, such as order status changes or inventory updates



DATABASE

The database layer uses **MongoDB**, a NoSQL database known for its flexibility and scalability, to store and manage application data.

Key Features:

- **Schema Design**:
  MongoDB's flexible schema design allows for quick adjustments as new requirements emerge. The main collections include:
  - **Users**: Stores user profiles, authentication details, and order history.
  - **Products**: Stores product details, stock levels, and pricing.
  - **Orders**: Tracks order placement, status, and payment information.

- **Relationships**:
    - Logical relationships between collections (e.g., linking orders to users and products using ObjectIDs).
- **Indexing**:
  Indexes are created on fields like productName and category to optimize search queries.
- **Scalability**:
    - MongoDB's sharding feature can be leveraged to handle large datasets and concurrent users.

Integration and Communication

- **Frontend-Backend Communication**:
    - The frontend communicates with the backend through RESTful APIs.
    - Asynchronous requests are handled using **Axios** or **Fetch API**, ensuring non-blocking operations.
- **Backend-Database Communication**:
    - The backend uses **Mongoose**, an Object Data Modeling (ODM) library for MongoDB, to interact with the database.
    - Mongoose schemas enforce data consistency and validation at the application level.

Security

Security measures are implemented at every layer of the architecture to ensure the safety of user data and transactions:

- **Frontend**:
    - Input validation using React hooks and libraries like Formik or Yup.
- **Backend**:
    - JWT-based authentication and role-based access control. ○ Rate-limiting middleware to prevent brute force attacks.
- **Database**:
    - Data encryption for sensitive fields like passwords (hashed using bcrypt).

Deployment

The architecture supports smooth deployment to cloud platforms like **AWS**, **Heroku**, or **Vercel**. Key components:

- **Frontend**: Deployed on **Netlify** or **Vercel**.
- **Backend**: Deployed on **AWS EC2** or **Heroku**.

Database: Hosted on **MongoDB Atlas**, a fully managed cloud database service

Technology Stack:

Layer	**Technology**
Frontend	React.js, Redux, Bootstrap
Backend	Node.js, Express.js
Database	MongoDB, Mongoose
State Management	Redux or Context API
Authentication	JWT
Deployment	AWS, Heroku, MongoDB Atlas

CHAPTER 4

SETUP INSTRUCTIONS

The following detailed setup instructions will guide you through the process of installing and running the ShopEZ App locally on your system.

PREREQUISITIES

1. **Node.js** (v14 or later):
   - Download and install from Node.js Official Website.
   - Verify installation by running:

```
node -v npm

-v
```

2. **MongoDB** (v4.4 or later):
   - Download and install MongoDB from MongoDB Official Website. ○ Start MongoDB service:

```
mongod
```

3. **Git**:
   - Download and install Git from Git Official Website.
   - Verify installation by running:

```
git --version
```

4. **Code Editor**:
   - Use a code editor such as **Visual Studio Code** (recommended).

5. **Browser**:
   - A modern browser like Google Chrome or Mozilla Firefox.

Cloning the Repository

1. Open a terminal and navigate to the directory where you want to clone the project.
2. Clone the repository:
   Git clone : https://github.com/REKHA-RD6/E-commerce-Aplication.git

3.Navigate to the project directory:

```
cd grocery-web-app
```

Environment Variables

1. Create a .env file in both the **client** and **server** directories.
2. Add the following environment variables in the respective .env files:

Client (/client/.env):

```
REACT_APP_API_URL=http://localhost:5000/api
```

Server (/server/.env):

```
PORT=5000

MONGO_URI=mongodb://localhost:27017/grocery-app

JWT_SECRET=your_jwt_secret_key
```

INSTALLATION

Install the required dependencies for both the client (frontend) and server (backend).

1. **Frontend**:

Navigate to the client directory:

```
cd client
```

Install dependencies using npm:

```
npm install
```

2. **Backend**:

Navigate to the server directory:

```
cd ../server
```

Install dependencies using npm:

```
npm install
```

CHAPTER 5

RUNNING THE APPLICATION

1.Start the MongoDB Service:

○ Ensure the MongoDB service is running before starting the application.

```
mongod
```

2.Start the Backend Server:

Navigate to the server directory:

```
cd server
```

Start the backend server:

```
npm start
```

○ The backend server will run at http://localhost:5000.

3.Start the Frontend Server:

Open a new terminal and navigate to the client directory:

```
cd client
```

Start the frontend development server:

```
npm start
```

○ The frontend will run at http://localhost:3000.

Verify the Setup

1. Open a browser and navigate to http://localhost:3000 to access the frontend of the Grocery Web App.
2. Test key functionalities:
   - ○ User registration and login.
   - ○ Product browsing and adding items to the cart.

Admin functionalities if applicable .

ADDITIONAL NOTES :

Data Seeding:

If the application requires initial data for products or users, run a seed script (if provided) in the backend directory:

```
npm run seed
```

1. **Using External APIs**:
   If the project integrates external APIs (e.g., payment gateways), ensure their keys are added to the .env file.
2. **Debugging**:
   - Use the terminal to monitor logs from the frontend and backend for troubleshooting errors.

Use browser developer tools for inspecting frontend issues

1. **Product Catalog:** Pre-load a diverse range of grocery products, categorized by type (e.g., fruits, vegetables, dairy, etc.), along with essential attributes like product name, description, price, SKU, and stock status. This ensures users have a rich selection when they first interact with the app.
2. **User Accounts:** Seed sample user data for testing purposes, including user profiles, account details, and order history. This helps ensure that user authentication and account management features are functioning correctly.
3. **Promotions and Offers:** Include default promotional offers, discounts, and loyalty points information so users can access deals immediately upon signup. This is essential for testing notification systems and discount applications.
4. **Categories and Subcategories:** Populate predefined categories (e.g., fruits, vegetables, snacks) with relevant products. This allows users to filter and search effectively from day one.
5. **Stock Levels and Availability:** Seed data reflecting initial stock levels for products, enabling real-time updates for users and testing the stock tracking and notification system.
6. **Payment Methods:** Configure a set of sample payment methods and payment gateway test data for the initial stages of development and testing.

CHAPTER 6

API DOCUMENTATION

This document outlines the key API endpoints for the **ShopEZ: E-commerce App**, which allows users to browse products, manage their cart, place orders, and track delivery. The API is RESTful, with JSON responses.

Base URL arduino
https://shopEZapp.com/api

Authentication

All requests require an authentication token. The token is provided upon successful login and should be included in the Authorization header for every request.

Authorization Header
makefile
Authorization: Bearer <access_token>

Endpoints

1. User Registration

POST /auth/register

- **Description:** Create a new user account.
- **Request Body:**

  json

  {

"name": "John Doe",
"email":"john.doe@example
.com",

"password":"securepassword"

}

- **Response:**

  {

  "message":"User

  registered successfully”

  "user": { "id": "userId",

  "email":”john.doe@example

  .com" }

  }

2. **user Login**

POST /auth/login

- **Description:** Login and retrieve authentication token.
- **Request Body:**

  json

  ```
  {
    "email": "john.doe@example.com",
    "password": "securepassword"
  }
  ```

- **Response:**

  json

```
{
  "token":”JWT_TOKEN”,
  "token": "{“id”:"userId”,”name”:”John doe”}
}
```

2. Product Management

GET /products

- **Description:** Retrieve a list of all products.
- **Query Parameters:**
  - category (optional) – Filter products by category.
  - search (optional) – Search products by name or description.

Response: json

```
[
 {
   "id": "101",
   "name": "Apple",
   "category": "Fruits",
   "price": 1.99,
   "stock": 50,
   "image_url": "https://example.com/apple.jpg"
 },
 {
   "id": "102",
   "name": "Banana",
   "category": "Fruits",
   "price": 1.49,
   "stock": 20,
   "image_url": "https://example.com/banana.jpg"
 }
]
```

2.Get Product by ID

GET /products/{id}

- **Description:** Get details of a specific product.

Response: json

```
{
  "id": "101",
  "name": "Apple",
  "category": "Fruits",
  "price": 1.99,
  "stock": 50,
  "description": "Fresh red apples from local farms.",
  "image_url": "https://example.com/apple.jpg"
}
```

GET /categories

- **Description:** Retrieve all product categories.
- **Response:** json

```
[
  "Fruits",
  "Vegetables",
  "Dairy",
  "Shoes",
  "Books"
]
```

2. Shopping Cart

GET /cart

- **Description:** Retrieve the current user's shoppingcart.

Response:

json

```
{
  "items": [
    {
      "product_id": "101",
      "name": pen",
      "quantity": 1,
      "price": 10
    },
    {
      "product_id": "102",
      "name": "Ball",
      "quantity": 2,
      "price": 100
    }
  ],
  "total_price": 110
}
```

POST /cart

- **Description:**

  Add a product to the cart.

  Request Body:

  json

  ```
  {
    "product_id": "101",
    "quantity": 2
  }
  ```

- **Response:**

  json

  ```
  {
    "message": "Product added to cart",
  ```

```
 "cart": {

   "items": [
     {
       "product_id": "101",
       "name": "pen",
       "quantity": 3,
       "price": 100
     }
   ],
   "total_price": 310
 }
}
```

DELETE /cart/{product_id}

- **Description:** Remove a product from the cart.

Response:

json

```
{
  "message": "Product removed from cart",
  "cart": {
    "items": [
      {
        "product_id": "102",
        "name": "Ball",
        "quantity": 2,
        "price": 100
      }
    ],
    "total_price": 110
  }
}
```

3. Order Management

POST /orders

- **Description:** Place a new order.

  Request Body:

  ```
  json
  {
    "address": "123 Main St, City, Country",
    "payment_method": "credit_card",
    "cart_id": "abc123"
  }
  ```

- **Response:** json

  ```
  {
    "message": "Order placed successfully",
    "order_id": "order123",
    "status": "pending"
  }
  ```

GET /orders/{order_id}

 Description: Retrieve order

details.

Response:

json

```
{
  "order_id": "order123",
  "status": "pending",
  "items": [
    {
      "product_id": "101",
      "name": "pen",
      "quantity": 1,
      "price": 10
    }
```

```
],
"total_price": 50,
"shipping_address": "123 Main St, City, Country" }
```

GET /orders

- **Description:** Retrieve all orders for the authenticated user.
- **Response:**

```
json
[
  {
    "order_id": "order123",
    "status": "shipped",
    "total_price": 50,
    "date": "2024-11-20T14:30:00Z"
  }
]
```

4. Stock and Notification System

GET /stock/{product_id}

- **Description:** Get real-time stock availability for a product.
- **Response:**

```
json
{
  "product_id": "101",
  "stock": 50
}
```

POST /notifications

- **Description:** Subscribe to notifications for deals, promotions, or stock updates.
- **Request Body:**

```
json
{
  "product_id": "101",
  "notification_type": "stock_update"
}
```

- **Response:**

  json

  ```
  {
    "message": "Subscribed to stock updates",
    "subscription_id": "sub123"
  }
  ```

Error Handling

All API responses include an error code and message in the event of a failure.

Example Error Response:

json

```
{
"error
": {
    "code": "400",
    "message": "Product not found"
  }
}
```

Rate Limiting

- API requests are rate-limited to prevent abuse. The limits are as follows: ○ **500 requests per minute** per user ○ If exceeded, you will receive a 429 Too Many Requests error.

CHAPTER 7

AUTHENTICATION & AUTHORIZATION

Authentication is a critical feature in the Grocery Web App as it ensures secure and personalized access to the platform. Here's a detailed breakdown of the authentication process: 1. **User Registration (Sign-Up):**

- Users are required to create an account with personal details such as name, email, phone number, and a secure password. ○ Validation checks should be in place to ensure the email is unique and the password meets security requirements (e.g., minimum length, special characters).
- An email verification process can be implemented to ensure the legitimacy of the user's email address.

2. **Login:**
   - Users can log in using their registered email and password. ○ A "Remember Me" feature can be added for a more convenient login experience on returning visits.
   - Multi-Factor Authentication (MFA) can be integrated for enhanced security, prompting users for an additional verification step (e.g., SMS code or email link).
3. **Password Recovery:**
   - A "Forgot Password" option allows users to reset their password through a secure process, typically involving sending a password reset link to the registered email. ○ Security questions or additional identity verification can be added to ensure the user is the rightful account holder.
4. **Session Management:**
   - After successful login, users are assigned a session token, stored securely (e.g., in cookies or local storage), allowing them to remain logged in while browsing the site. ○ Session expiry should be managed to automatically log out inactive users, improving security.
5. **Profile Management:**
   - Users can update their personal details, such as name, address, phone number, and payment information through their account settings.

    - Password changes can also be handled through this section with reauthentication for security.

6. **Role-Based Authentication:**
    - Different user roles, such as **Admin**, **Customer**, or **Seller**, can be defined with varying levels of access.
        - **Admins** manage the backend, approve listings, handle inventory, and monitor orders.
        - **Customers** can browse products, place orders, and view order history.
        - **Sellers** can add products, track orders, and manage their inventory.
7. **Social Media Login:**
    - Integrating options like Google, Facebook, or other OAuth-based login methods can provide an easier and faster authentication process for users.
8. **Security Measures:**
    - Use encryption protocols like HTTPS to ensure the security of user credentials and sensitive information.
    - Implement rate limiting, CAPTCHA, and other mechanisms to prevent brute-force attacks and unauthorized access.

CHAPTER 8

CODING

1.CLIENT

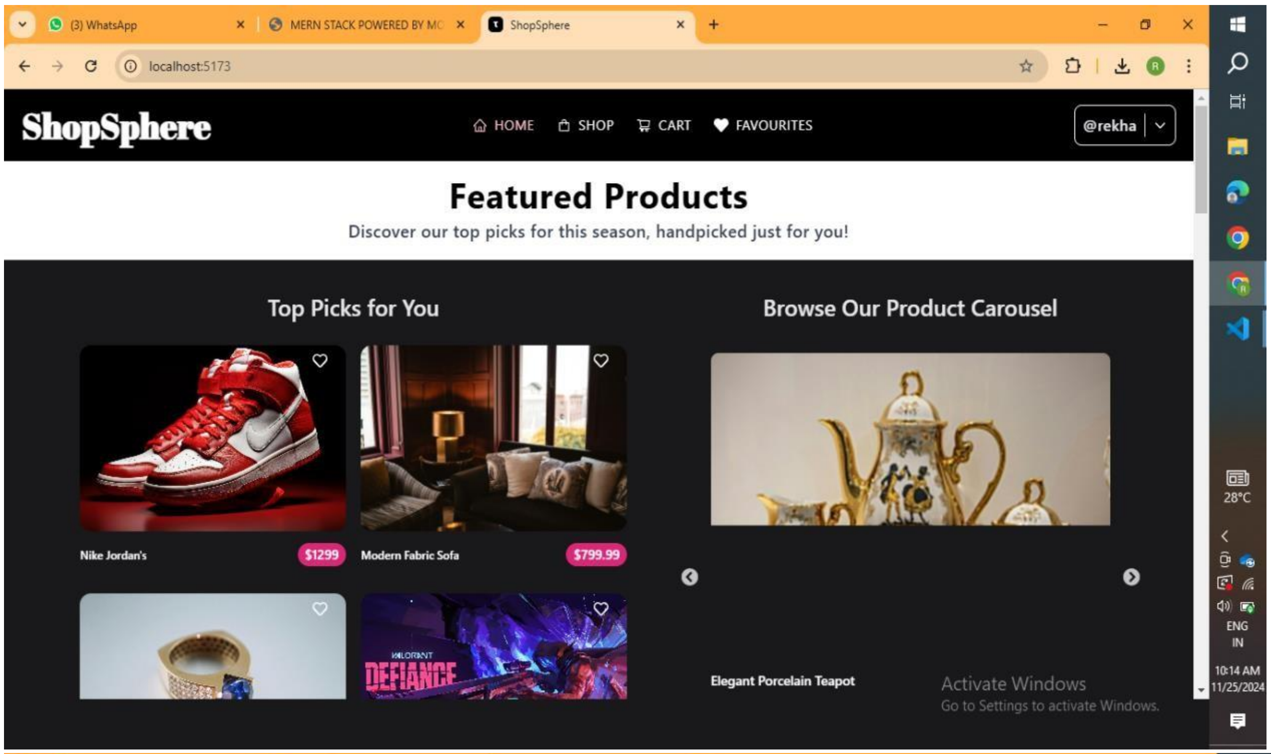
```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-
warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "@hookform/resolvers": "^3.9.0",
    "@reduxjs/toolkit": "^2.2.6",
    "apexcharts": "^3.51.0",
    "axios": "^1.7.2",
    "lucide-react": "^0.445.0",
    "moment": "^2.30.1",
    "prettier": "^3.3.3",
    "react": "^18.3.1",
    "react-apexcharts": "^1.4.1",
    "react-dom": "^18.3.1",
    "react-hook-form": "^7.53.0",
    "react-icons": "^5.2.1",
    "react-razorpay": "^3.0.1",
    "react-redux": "^9.1.2",
    "react-router": "^6.25.1",
    "react-router-dom": "^6.25.1",
    "react-slick": "^0.30.2",
    "react-toastify": "^10.0.5",
    "slick-carousel": "^1.8.1",
    "zod": "^3.23.8"
  },
  "devDependencies": {
    "@types/react": "^18.3.3",
    "@types/react-dom": "^18.3.0",
    "@vitejs/plugin-react": "^4.3.1",
    "autoprefixer": "^10.4.19",
    "eslint": "^8.57.0",
    "eslint-plugin-react": "^7.34.3",
    "eslint-plugin-react-hooks": "^4.6.2",
    "eslint-plugin-react-refresh": "^0.4.7",
    "postcss": "^8.4.40",
    "tailwindcss": "^3.4.6",
    "vite": "^5.3.4"
  }
}
```

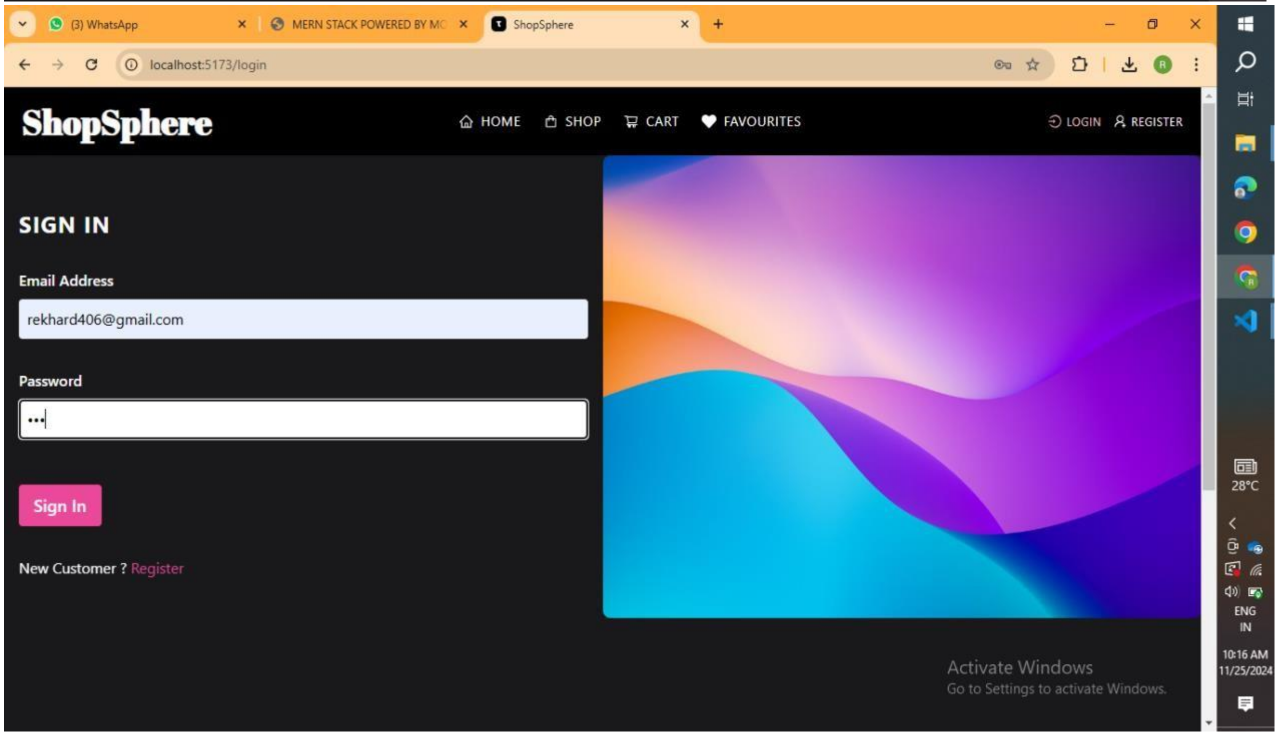
2.SERVER

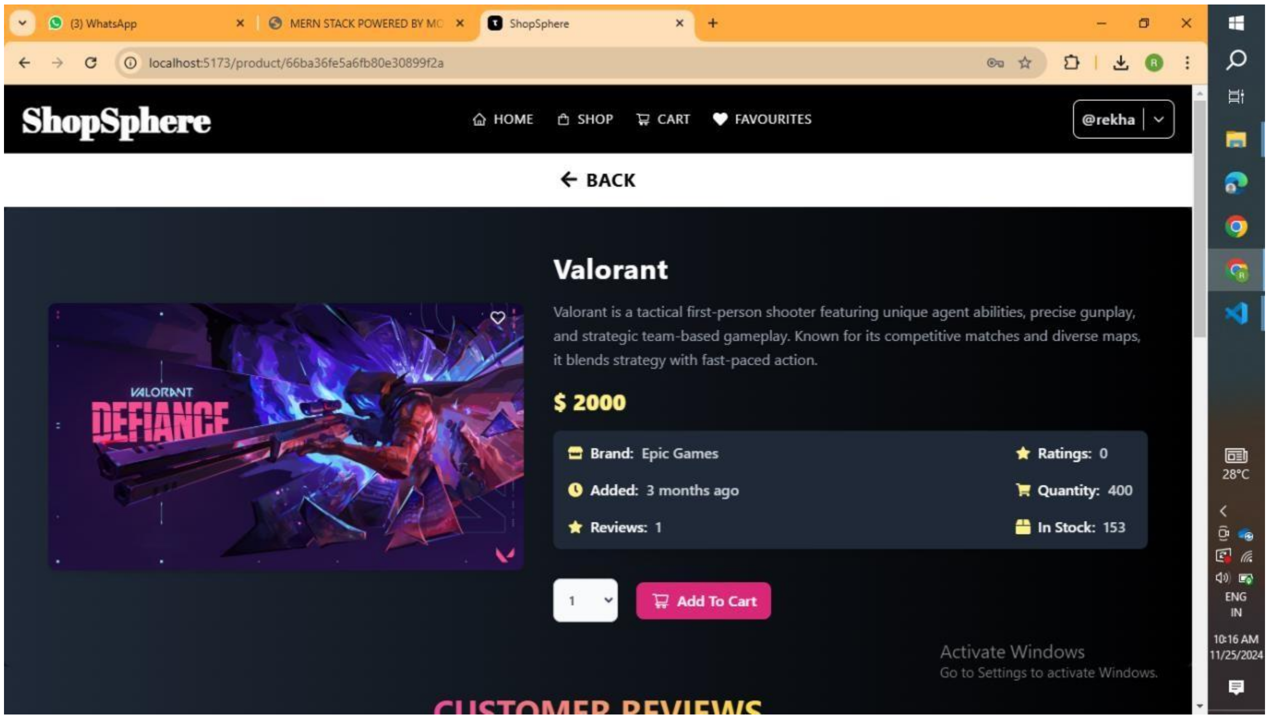
```
  {
  "name": "thrifty",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "node index.js",
    "backend": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cloudinary": "^2.5.0",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.21.0",
    "express-async-handler": "^1.2.0",
    "express-formidable": "^1.2.0",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.5.1",
    "multer": "^1.4.5-lts.1",
    "nodemon": "^3.1.4",
    "razorpay": "^2.9.4"
  },
  "devDependencies": {
    "prettier": "^3.3.3"
  }
}
```

CHAPTER 9

SCREENSHOTS







Valorant

Valorant is a tactical first-person shooter featuring unique agent abilities, precise gunplay, and strategic team-based gameplay. Known for its competitive matches and diverse maps, it blends strategy with fast-paced action.

$ 2000

Brand: Epic Games
Added: 3 months ago
Reviews: 1
Ratings: 0
Quantity: 400
In Stock: 153

DEMO VIDEO LINK:

https://drive.google.com/file/d/1-891bpkDzDon6fnf7fQ_RbrbzStHSFNd/view?usp=drivesdk

PROJECT GITHUB LINK:

https://github.com/REKHA-RD6/E-commerce-Aplication

CHAPTER 10

KNOWN ISSUES

While developing the ShopEZ e-commerce App, there are several potential or known issues that might arise. These can range from UI glitches to functionality problems. Below are common issues and challenges that may need to be addressed during development, testing, or post-launch.

1. Limited Payment Gateway Integration

- **Issue:** The project currently lacks integration with a real-world payment gateway. Users cannot make actual transactions, and the checkout process is simulated.
- **Impact:** Limits usability in real-world scenarios.
- **Potential Fix:** Integrate APIs from platforms like PayPal, Stripe, or Razorpay for secure online payments.

2. Lack of Performance Optimization

- **Issue:** The application may experience slow performance with large datasets due to basic MongoDB queries and a lack of caching mechanisms.
- **Impact:** Potential latency when fetching or filtering large numbers of products.
- **Potential Fix:** Implement indexing for database fields and use caching solutions like Redis.

3. Limited Scalability of User Roles

- **Issue:** Role-based access control (RBAC) is basic, supporting only "admin" and "customer" roles.
- **Impact:** Extending roles for vendors or delivery personnel may require significant refactoring.
- **Potential Fix:** Use a more flexible RBAC model with role hierarchies and permissions.

4. Frontend Responsiveness on Certain Devices

- **Issue:** The React frontend, while responsive, may not render perfectly on all screen sizes, particularly older devices or uncommon resolutions.
- **Impact:** Poor user experience for a segment of users.
- **Potential Fix:** Conduct extensive cross-device testing and refine CSS breakpoints.

5. Minimal Order Tracking System

- **Issue:** The current order status feature supports only a few states (e.g., Pending, Shipped).
- **Impact:** Insufficient for real-world e-commerce needs, such as returns or cancellations.
- **Potential Fix:** Expand order management features to include a complete lifecycle, including return, refund, and cancellation processes.

6. Basic Search and Filter Functionality

- **Issue:** Search is limited to product names, and filters only support predefined categories.
- **Impact:** Reduces usability for customers seeking advanced search options.
- **Potential Fix:** Implement full-text search and additional filters for price range, ratings, and tags.

7. Security Enhancements Required

- **Issue:** While JWT is used for authentication, the app lacks advanced security features like rate limiting, input sanitization, or HTTPS enforcement.
- **Impact:** Vulnerable to attacks such as brute force, SQL injection, or data interception.
- **Potential Fix:** Add middleware for rate limiting, validate user inputs, and enforce HTTPS for all communications.

8. No Email or Notification System

- **Issue:** The application does not send emails or notifications for order confirmations or updates.
- **Impact:** Users and admins must rely on manual tracking of orders.

- **Potential Fix:** Integrate email services like SendGrid or Nodemailer and add push notification capabilities.

CHAPTER 11

FUTURE ENHANCEMENT

1. **Payment Gateway Integration**

- **Description:** Add real-world payment gateway integrations like PayPal, Stripe, or Razorpay to support secure online transactions.
- **Impact:** Enables real monetary transactions, improving the platform's usability for production.

2. Advanced Search and Filters

- **Description:** Enhance search capabilities to include full-text search and add filters for price range, product ratings, brand, and availability.
- **Impact:** Provides a more tailored shopping experience, increasing user satisfaction.

3. Role-Based User Expansion

- **Description:** Extend the role-based access control (RBAC) system to include vendors, delivery personnel, and customer support roles.
- **Impact:** Broadens the platform's utility, making it suitable for multi-role operations.

4. Enhanced Order Tracking

- **Description:** Implement a comprehensive order tracking system, including shipment tracking, returns, refunds, and cancellations.
- **Impact:** Improves customer experience and trust in the platform.

5. Notifications and Email Alerts

- **Description:** Integrate email and push notifications for order confirmations, shipping updates, and promotional campaigns.

- **Impact:** Keeps users engaged and informed, leading to better customer retention.

6. AI-Powered Recommendations

- **Description:** Introduce machine learning algorithms to provide personalized product recommendations based on user behavior and purchase history.
- **Impact:** Boosts sales and enhances the user experience.

7. Multi-Language and Currency Support

- **Description:** Add multi-language capabilities and support for multiple currencies to cater to a global audience.
- **Impact:** Expands the platform's reach and usability across diverse regions.

8. Progressive Web Application (PWA)

- **Description:** Convert the platform into a Progressive Web Application for offline access and better performance on mobile devices.
- **Impact:** Increases accessibility and improves the mobile shopping experience

9. Social Media Integration

- **Description:** Allow users to log in via social media platforms and share products directly on their social profiles.
- **Impact:** Enhances user convenience and promotes the platform through social sharing.

10. Reviews and Ratings System

- **Description:** Add a feature for customers to leave reviews and rate products.
- **Impact:** Builds trust among users and provides valuable feedback to sellers.