

Full Stack Development with MERN

Project Documentation

INTRODUCTION

- PROJECT TITLE:

SHOP EZ: One Shop Stop For Online Purchasing

TEAM MEMBERS:

- DAGGULA. ALEKYA (ROLE: FRONTEND, IMPLEMENTATION, EXECUTION)
- TURAKA. VIJAY (ROLE: FRONTEND, BACKEND, IMPLEMENTATION)
- ALLA. BALA VENKATA TRIPURANTHA KESAVA REDDY (ROLE: BACKEND)
- ATCHI. AJAY BABU (ROLE: FRONTEND)
- RAYALA. RAVI KISHORE (ROLE: BACKEND)

PROJECT OVERVIEW

PURPOSE:

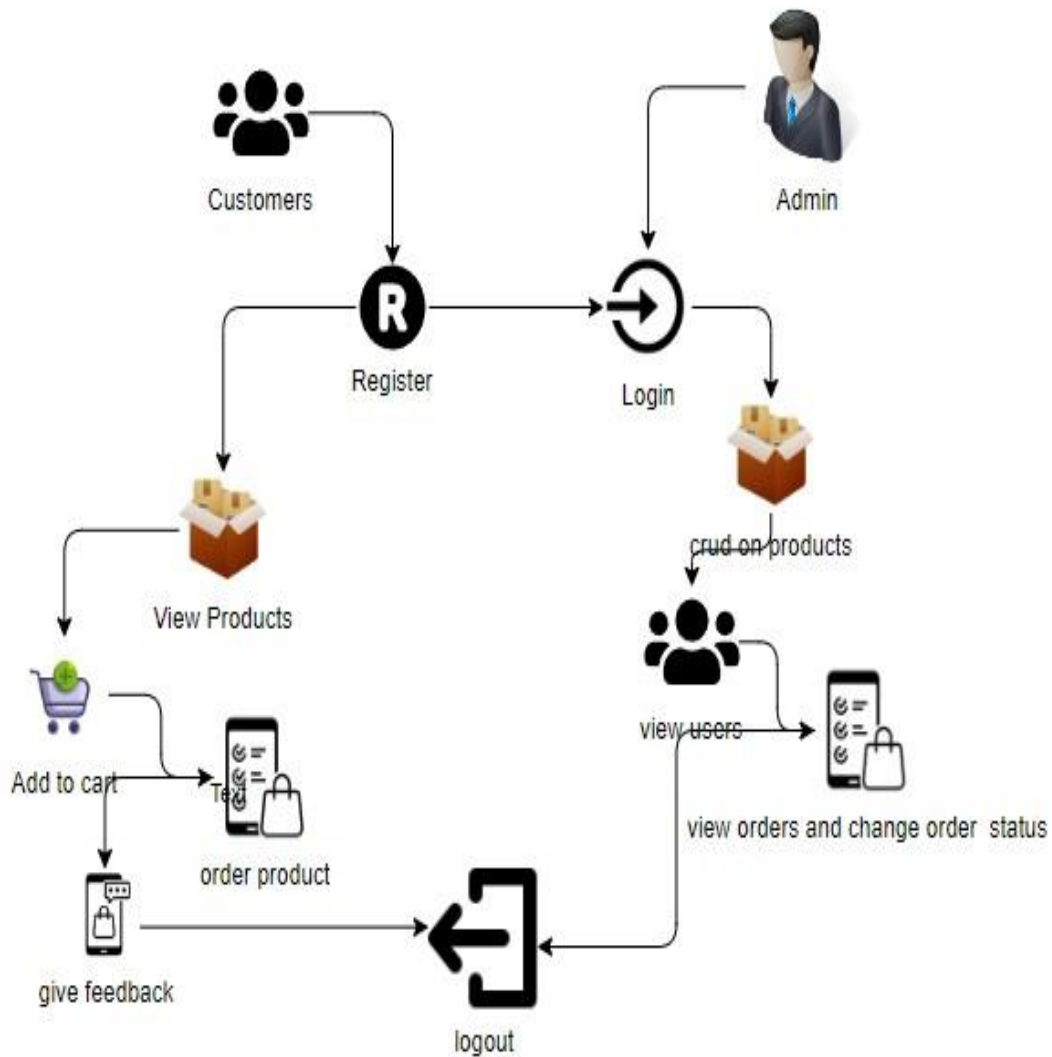
“Shop Smart” is a full-stack e-commerce application built with the MERN stack (MongoDB, Express.js, React, Node.js) that serves several important purposes. It acts as an online store where users can explore products, add items to their cart, and complete purchases, benefiting both buyers and sellers in the digital marketplace. This project allows developers to refine their skills in database management, server-side and client-side development, and user authentication. Additionally, creating a responsive design enhances UI/UX

capabilities, while developing a RESTful API deepens understanding of clientserver architecture and data management. Hosting the application on platforms like Heroku or AWS provides valuable experience in deployment and managing production environments. Ultimately, "Shop Smart" not only meets the real-world need for small businesses to establish an affordable online presence but also serves as a significant learning opportunity for aspiring developers.

FEATURES:

1. User Authentication
2. Product Browsing
3. Shopping Cart Functionality
4. Secure Checkout Process
5. Responsive Design
6. Order Management
7. Admin Dashboard
8. Product Reviews and Ratings
9. Search and Filter Options
10. Wishlist Functionality
11. Payment Integration
12. Real-time Notifications

ARCHITECTURE



FRONTEND DEVELOPMENT

1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

2.Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

3.Implement frontend logic:

- Integration with API endpoints.
- Implement data binding

BACKEND DEVOLOPMENT

1. Setup express server:

- Create index.js file.
- Create an express server on your desired port number.
- Define API's

2. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for admin, users, products, orders and other relevant data.

3. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

4. Define API Routes:

- Create separate route files for different API functionalities such as users, orders, and authentication.
- Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

5. Implement Data Models:

- Define Mongoose schemas for the different data entities like products, users, and orders.
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

7. Handle new products and Orders:

- Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.
- Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

- Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.
- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

DATABASE

Database Development Create

database in cloud

- Install Mongoose.
- Create database connection.
- Open your web browser and navigate to <http://localhost:3000>.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

Schema use-case:

1. User Schema:

- Schema: user schema
- Model: 'User'
- The User schema represents the user data and includes fields such as username, email, and password.
- It is used to store user information for registration and authentication purposes.
- The email field is marked as unique to ensure that each user has a unique email address

2. Product Schema:

- Schema: product schema
- Model: 'Product'
- The Product schema represents the data of all the products in the platform.
- It is used to store information about the product details, which will later be useful for ordering .

3. Orders Schema:

- Schema: orders schema
- Model: 'Orders'
- The Orders schema represents the orders data and includes fields such as user id, product Id, product name, quantity, size, order date, etc.,
- It is used to store information about the orders made by users.
- The user Id field is a reference to the user who made the order.

4. Cart Schema:

- Schema: cart schema
- Model: 'Cart'

- The Cart schema represents the cart data and includes fields such as userId, product id, product name, quantity, size, order date, etc.,
- It is used to store information about the products added to the cart by users.
- The user Id field is a reference to the user who has the product in cart.

5. Admin Schema:

- Schema: adminSchema
- Model: 'Admin
- The admin schema has essential data such as categories, banner.

SETUP INSTRUCTION

Pre-requisites

To develop a full-stack e-commerce app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm:

Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions:
<https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide:

<https://reactjs.org/docs/create-a-new-react-app.html>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

To Connect the Database with Node JS go through the below provided link: •

Link: <https://www.section.io/engineering-education/nodejs-mongoosejsmongodb/>

To run the existing ShopEZ App project downloaded from github: Follow below steps:

Clone the repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the e-commerce app. •
Execute the following command to clone the repository:

Drive

Link:

https://drive.google.com/drive/folders/1QnZb2_S3rrupyv1hm8Kok2FKBqTwTebc?usp=drive_link

Install Dependencies:

- Navigate into the cloned repository directory:

cd ShopEZ—e-commerce-App-MERN

- Install the required dependencies by running the following command: **npm install**

Start the Development Server:

- To start the development server, execute the following command:

npm run dev or npm run start

- The e-commerce app will be accessible at <http://localhost:3000> by default. You can change the port configuration in the .env file if needed.

Access the App:

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the ShopEZ app on your local machine. You can now proceed with further customization, development, and testing as needed.

INSTALLATION

NODE INSTALLATION:

Setting up the Node Development Environment

The Node can be installed in multiple ways on a computer. The approach used by you depends on the existing development environment in the system. There are different package installers for different environments. You can install Node by grabbing a copy of the source code and compiling the application. Another way of installing Node is by cloning the GIT repository in all the three environments and then installing it on the system.

Installing Node On Windows (WINDOWS 10):

You have to follow the following steps to install the Node.js on your Windows :

Step-1: Downloading the Node.js '.msi' installer.

The first step to install Node.js on windows is to download the installer. Visit the official Node.js website i.e) <https://nodejs.org/en/download/> and download the

.msi file according to your system environment (32-bit & 64-bit). An MSI installer will be downloaded on your system.

Step-2: Running the Node.js installer.

Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:- □ Double click on the .msi installer.

The Node.js Setup wizard will open.

- Welcome To Node.js Setup Wizard.

Select “Next”

-
- After clicking “Next”, End-User License Agreement (EULA) will open.

Check “I accept the terms in the License Agreement”

Select “Next”

-
- Destination Folder

Set the Destination Folder where you want to install Node.js & Select “Next”

-
- Custom Setup

Select “Next”

-
- Ready to Install Node.js.

The installer may prompt you to “install tools for native modules”. ***Select “Install”***

- Installing Node.js.

Do not close or cancel the installer until the install is complete

□ Complete the Node.js Setup Wizard. ***Click “Finish”***

Step 3: Verify that Node.js was properly installed or not.

To check that node.js was completely installed on your system or not, you can run the following command in your command prompt or Windows Powershell and test it:-

```
C:\Users\Admin> node -v
```

If node.js was completely installed on your system, the command prompt will print the version of the node.js installed.

Express Installation:

Assuming you've already installed [Node.js](#), create a directory to hold your application, and make that your working directory.

```
$ mkdir myapp
```

```
$ cd myapp
```

Use the npm init command to create a package.json file for your application.

For more information on how package.json works, see [Specifics of npm's package.json handling](#).

```
$ npm init
```

This command prompts you for a number of things, such as the name and version of your application. For now, you can simply hit RETURN to accept the defaults for most of them, with the following exception:

entry point: (index.js)

Enter app.js, or whatever you want the name of the main file to be. If you want it to be index.js, hit RETURN to accept the suggested default file name.

Now install Express in the myapp directory and save it in the dependencies list. For example:

```
$ npm install express
```

To install Express temporarily and not add it to the dependencies list:

```
$ npm install express --no-save
```

By default with version npm 5.0+ npm install adds the module to the dependencies list in the package.json file; with earlier versions of npm, you must specify the --save option explicitly. Then, afterwards, running npm install in the app directory will automatically install modules in the dependencies list.

React Installation:

```
npm install -g create-react-app
```

Installation will take few seconds

It will globally install react app for you. To check everything went well run the command

```
create-react-app --version
```

version 4.0.3

If everything went well it will give you the installed version of react app

Step 4: Now Create a new folder where you want to make your react app using the below command:

```
mkdir newfolder
```

Note: The *newfolder* in the above command is the name of the folder and can be anything.

Move inside the same folder using the below command:

```
cd newfolder (your folder name)
```

Step 5: Now inside this folder run the command → create-react-app

```
reactfirst YOUR_APP_NAME
```

It will take some time to install the required dependencies

NOTE: Due to npm naming restrictions, names can no longer contain capital letters, thus type your app's name in lowercase.

Step 6: Now open the IDE of your choice for eg. Visual studio code and open the folder where you have installed the react app **newfolder** (in the above example) inside the folder you will see your app's name **reactapp** (In our example). Use the terminal and move inside your app name folder. Use command **cd reactapp** (your app name)

Step 7: To start your app run the below command :

```
npm start
```

Once you run the above command a new tab will open in your browser showing React logo as shown below :

Congratulation you have successfully installed the react-app and are ready to build awesome websites and app.

ABOUT MERN:

It seems there might be a slight confusion. The MERN stack traditionally refers to MongoDB, Express.js, React, and Node.js, with MongoDB serving as the NoSQL database. If you are specifically using MySQL as your database, it's often referred to as the MEAN stack, where the "M" stands for MySQL.

MySQL is a widely-used relational database management system that supports the structured query language (SQL). It provides a robust and scalable solution for handling structured data, making it suitable for various applications. MySQL Workbench, a visual database design tool, is commonly used to interact with and

manage MySQL databases. It offers features such as SQL development, database design, and administration tools in a unified environment.

Integrating MySQL into your stack involves configuring the backend (Node.js and Express) to communicate with the MySQL database, while the frontend (React) interacts with the backend through APIs. This MEAN stack can provide a powerful and flexible foundation for building dynamic and data-driven web applications, leveraging the strengths of both MySQL and the JavaScript-based technologies.

Installing visual studio code

Steps to installing visual studio code:

Step 1: Head over to the website <https://code.visualstudio.com/download>

Step2:choose the option as per your choice.

Step 3: Once download completed,open it and your screen will appear like this

Step 4: click next until you saw the below page

Step 4: After installation your screen will appear like this ,click finish and continue.

FOLDER STRUCTURE

Client (React Frontend) scss

Copy code client/

```
├── public/
|   ├── index.html
|   ├── favicon.ico
|   └── assets/      // Static assets like images, fonts, etc.
├── src/
|   ├── components/  // Reusable components (e.g., Navbar, Footer)
|   ├── pages/       // Page components (e.g., Home, ProductDetail, Cart)
|   ├── contexts/    // Context API for state management
|   ├── hooks/       // Custom hooks
|   ├── services/    // API service functions (e.g., axios calls)
|   ├── styles/      // Global and component-specific styles (CSS or SCSS)
|   ├── utils/       // Utility functions (e.g., formatters, validators)
|   ├── App.js       // Main app component
|   ├── index.js     // Entry point for React
|   └── routes.js    // Route definitions for React Router
├── package.json     // Client dependencies and scripts
└── .env             // Environment variables for the client
```

Server (Node.js Backend) arduino Copy code server/

```
├── config/          // Configuration files (e.g., database connection, environment variables)
├── controllers/     // Business logic (e.g., userController, productController)
├── middleware/      // Custom middleware (e.g., authentication checks)
└── models/          // Mongoose models (e.g., User, Product, Order)
```

```
└─ routes/      // Express route definitions (e.g., userRoutes, productRoutes)
└─ services/    // Additional service functions (e.g., email service)
└─ utils/       // Utility functions (e.g., error handling, response formatting)
└─ .env         // Environment variables for the server
└─ server.js    // Main server file
└─ package.json // Server dependencies and scripts
└─ README.md    // Project documentation
```

Overview

- **Client:** The React frontend consists of components, pages, and styles, organized to facilitate reusability and separation of concerns. Contexts and hooks help manage state and side effects.
- **Server:** The Node.js backend is organized into controllers for handling requests, models for database schemas, and routes for defining API endpoints. Middleware can handle tasks like authentication.

RUNNING THE APPLICATION

1. **Navigate to the Client Directory and Start the Frontend:**

```
cd client
```

```
npm install    # Install dependencies (if you haven't already)
npm start     # Start the React frontend
```

2. **Navigate to the Server Directory and Start the Backend:**

```
cd ../server    # Move back to the server directory
npm install     # Install dependencies (if you haven't already)
npm start       # Start the Node.js backend
```

Summary of Commands

□ **Frontend:**

```
cd client
npm install
npm start
```

□ **Backend:**

```
cd ../server
```

API DOCUMENTATION

User Endpoints

1.1 Register a New User

- **Endpoint:** POST /api/users/register

- **Request Body:**

```
{  
  "username": "string",  
  "email": "string",  
  "password": "string"  
}
```

- **Response:**

- **Success (201):**

```
{  
  "message": "User registered successfully",  
  "user": {  
    "id": "user_id",  
    "username": "string",  
    "email": "string"  
  }  
}
```

- **Error (400):**

```
{  
  "error": "User already exists"  
}
```

1.2 User Login

- **Endpoint:** POST /api/users/login □

Request Body:

```
{  
  "email": "string",  
  "password": "string"  
}
```

- **Response:** ○ **Success (200):**

```
{  
  "message": "Login successful",  
  "token": "jwt_token"  
}
```

- **Error (401):**

```
{  
  "error": "Invalid credentials"  
}
```

1.3 Get User Profile

- **Endpoint:** GET /api/users/profile □

Headers: ○ Authorization:

Bearer <token> □ **Response:** ○

Success (200):

```
{  
  "id": "user_id",  
  "username": "string",
```

```
"email": "string"
}
```

○ **Error (401):**

```
{
  "error": "Unauthorized"
}
```

Product Endpoints

2.1 Get All Products

- **Endpoint:** GET /api/products □

Response: ○ **Success (200):**

```
{
  "id": "product_id",
  "name": "string",
  "price": "number",
  "description": "string",
  "image": "url"
},
```

2.2 Get Product by ID □ **Endpoint:**

GET /api/products/:id □

Response: ○ **Success (200):**

```
{
  "id": "product_id",
  "name": "string",
  "price": "number",
  "description": "string",
}
```

```
"image": "url"
}
```

- **Error (404):**

```
{
  "error": "Product not found"
}
```

2.3 Create a New Product (Admin Only)

- **Endpoint:** POST /api/products □

Headers: ◦ Authorization:

Bearer <token> □ **Request Body:**

json

Copy code

```
{
  "name": "string",
  "price": "number",
  "description": "string",
  "image": "url"
}
```

- **Response:**

- **Success (201):**

```
{
  "message": "Product created successfully",
  "product": {
    "id": "product_id",
    "name": "string",
    "price": "number",
    "description": "string",
    "image": "url"
  }
}
```

Cart Endpoints

3.1 Get User Cart

- **Endpoint:** GET /api/cart □
Headers: ○ Authorization:
Bearer <token> □ **Response:** ○
Success (200):

```
{
  "productId": "product_id",
  "quantity": "number"
}
```

- **Endpoint:**

□ **Headers:**

○

□

3.2 Add Product to Cart POST

/api/cart

Authorization: Bearer <token>

Request Body:

```
{  
  "productId": "product_id",  
  "quantity": "number"  
}
```

- **Response:**

- **Success (201):**

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

3.3 Remove Product from Cart

- **Endpoint:** DELETE

/api/cart/:productId □ **Headers:** ○

Authorization: Bearer <token> □

Response:

- **Success (200):**

```
{  
  "message": "Product removed from cart"  
}
```

□ **Endpoint:**

□ **Headers:**

○

□

Order Endpoints

4.1 Create an Order

POST /api/orders

Authorization: Bearer <token>

Request Body:

```
{
  "cartItems": [
    {
      "productId": "product_id",
      "quantity": "number"
    }
  ],
  "totalAmount": "number",
  "shippingAddress": "string"
}
```

□ **Response:** ○ **Success**

(201):

```
{
  "message": "Order placed successfully",
  "order": {
    "id": "order_id",
    "totalAmount": "number",
  }
}
```

□ **Endpoint:**

□ **Headers:**

○

□

"status": "string"

}

}

4.2 Get User Orders GET

/api/orders

□ **Endpoint:**

□ **Headers:**

○

□

Authorization: Bearer <token>

Response:

○ **Success (200):**

```
{  
  "id": "order_id",  
  "totalAmount": "number",  
  "status": "string",  
  "createdAt": "date"  
},
```

4.3 Get Order by ID □ **Endpoint:** GET

/api/orders/:id □ **Headers:** ○

Authorization: Bearer <token> □

Response:

○ **Success (200):**

```
{  
  "id": "order_id",  
  "totalAmount": "number",  
  "status": "string",  
  "items": [  
    {  
      "productId": "product_id",  
      "quantity": "number" "shippingAddress": "string",  
      "createdAt": "date"  
    }  
  ]  
}
```

AUTHENTICATION

1. User Registration:

- Users can register by providing a username, email, and password. ○ Passwords are securely hashed using libraries like bcrypt before being stored in the database. This ensures that even if the database is compromised, the passwords remain secure.

2. User Login:

- During login, users provide their email and password. ○ The server checks the credentials against the database. If the email exists and the password matches the hashed version, the user is authenticated.

3. JWT (JSON Web Tokens):

- Upon successful authentication, the server generates a JWT. This token contains user information (such as user ID) and is signed with a secret key. ○ The token is sent back to the client and is stored (typically in localStorage or sessionStorage).
- The JWT allows the server to verify the user's identity on subsequent requests without needing to re-authenticate.

Authorization

1. Role-Based Access Control (RBAC):

- Users can have different roles (e.g., user, admin). The role information can be included in the JWT payload.
- The server checks the user's role when accessing specific endpoints (e.g., admin endpoints for managing products).

2. Token Verification Middleware:

- For protected routes, a middleware function checks for the presence of the JWT in the request headers.

- The token is verified using the secret key. If valid, the middleware extracts the user information from the token and attaches it to the request object, allowing access to the endpoint.
- If the token is missing or invalid, a 401 Unauthorized response is sent.

3. Session Management:

- While the project primarily uses JWT for stateless authentication, sessions can also be implemented if needed (e.g., using sessions with a session store).
- However, JWTs are preferred in this case for their scalability, especially in a distributed environment.

Token Expiration and Refresh

□ Token Expiration:

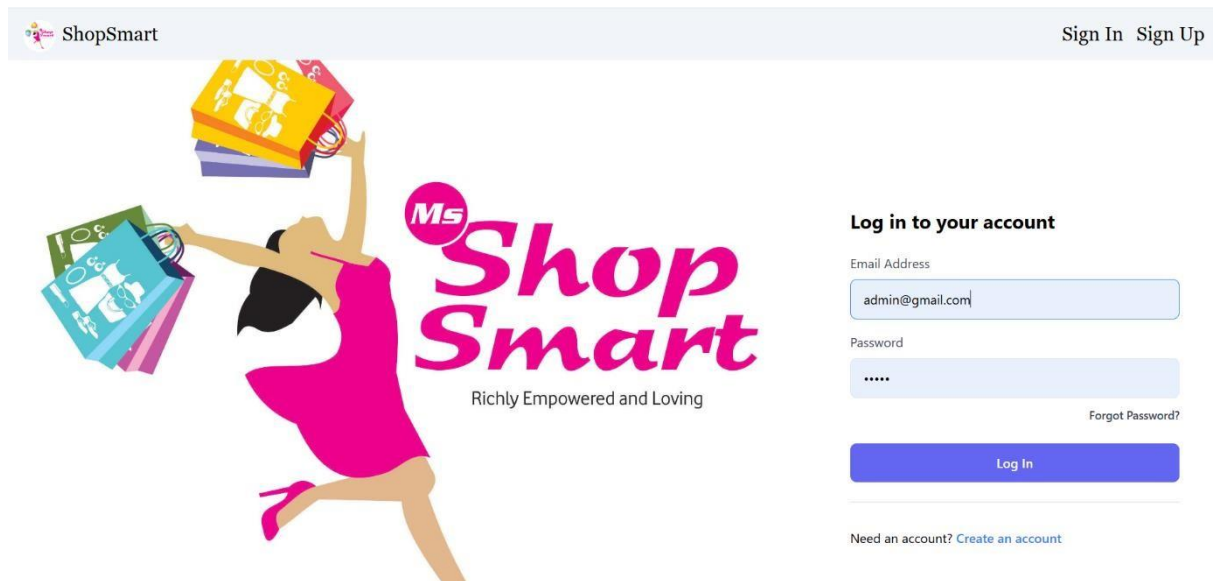
- The JWT has an expiration time (e.g., 1 hour) to limit its validity. After expiration, users need to log in again to obtain a new token.

□ Refresh Tokens (Optional):

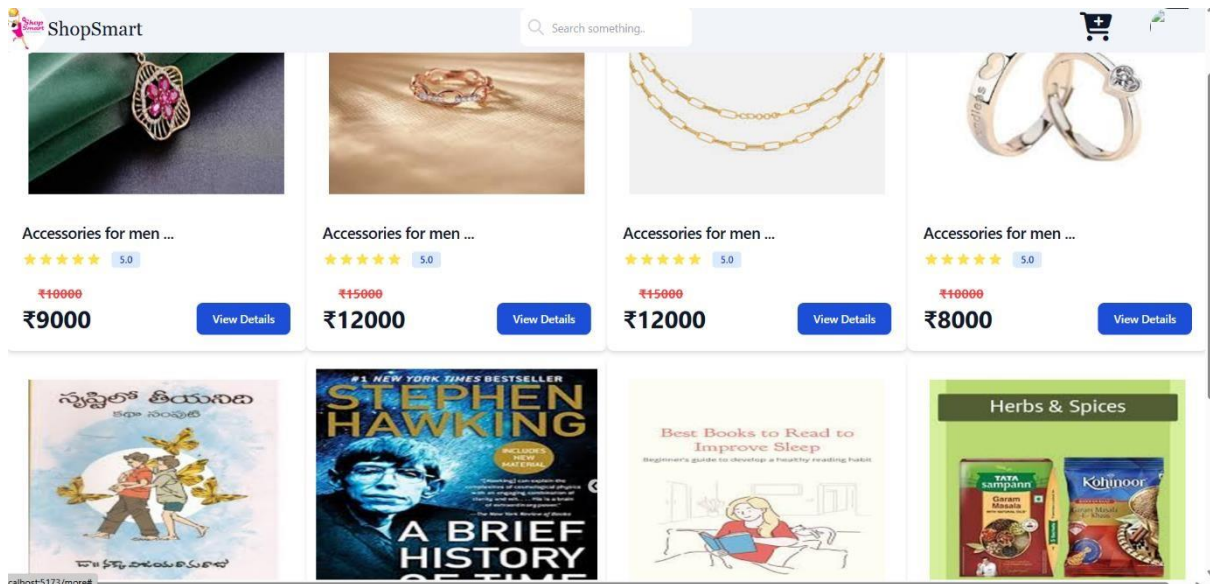
- If implementing refresh tokens, the server can issue a long-lived refresh token alongside the access token.
- The refresh token can be used to obtain a new access token without requiring the user to log in again.

USER INTREFACE

Home Page:



Customer View Product Page:



Add To Cart:



Your Cart



kids ware for
boys

Clothing and
Apparel

- 1 +

₹2500



kids ware for
girls

Clothing and
Apparel

- 1 +

₹2000



Accessories
men and
women

Jewelry and
Accessories

- 1 +

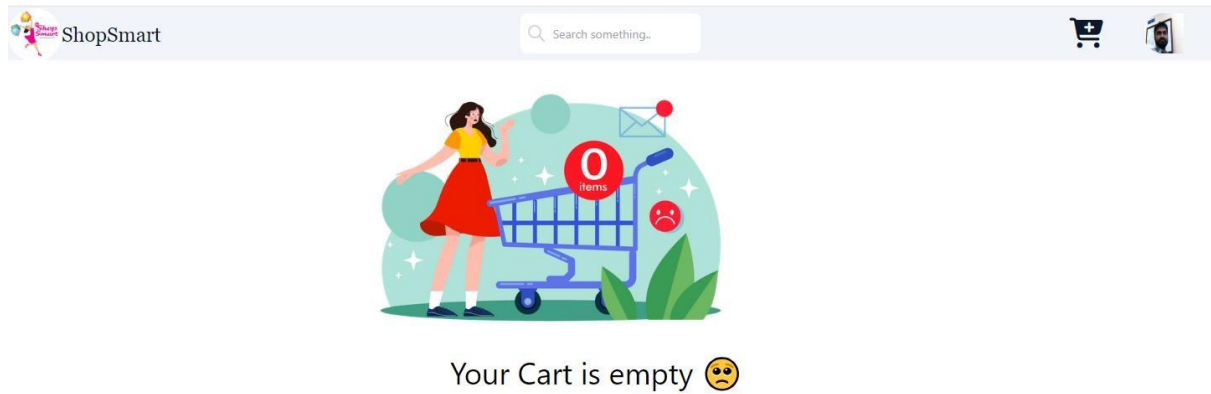
₹12000



Subtotal

₹16500

Cart Page:



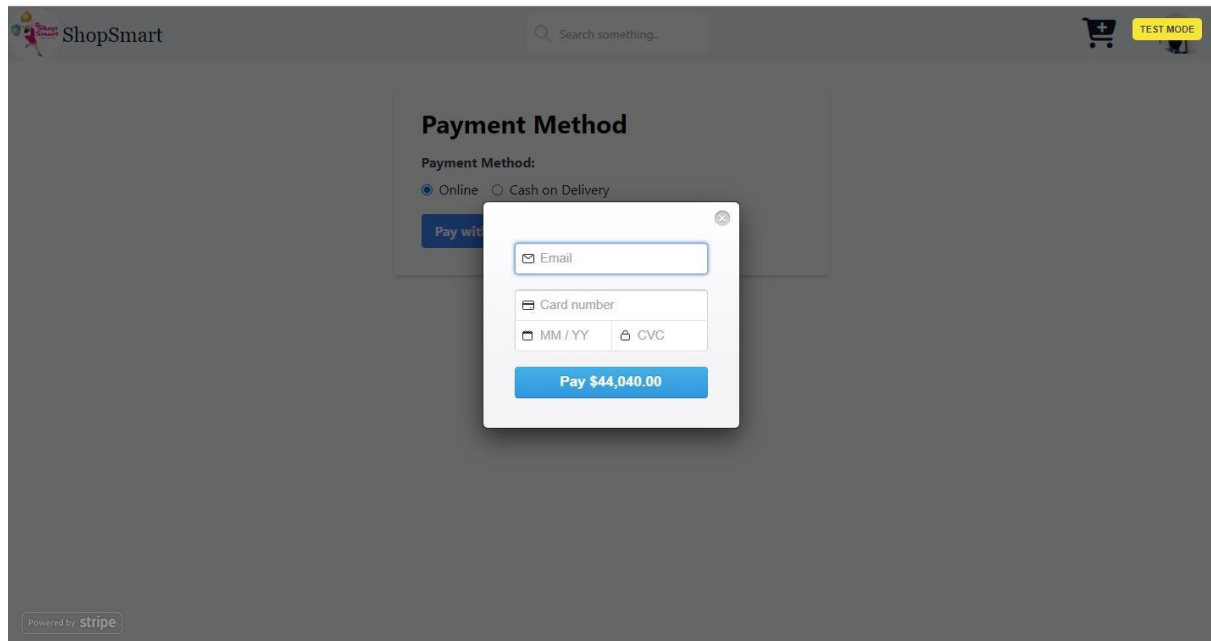
Shipping Info Page:

The screenshot shows the ShopSmart website's shipping information page. The header is identical to the cart page, featuring the ShopSmart logo, a search bar, and icons for a shopping cart and a user profile. The main content area is titled "Shipping Info" with the subtitle "Enter the corret Address". Below this, there is a form with the following fields:

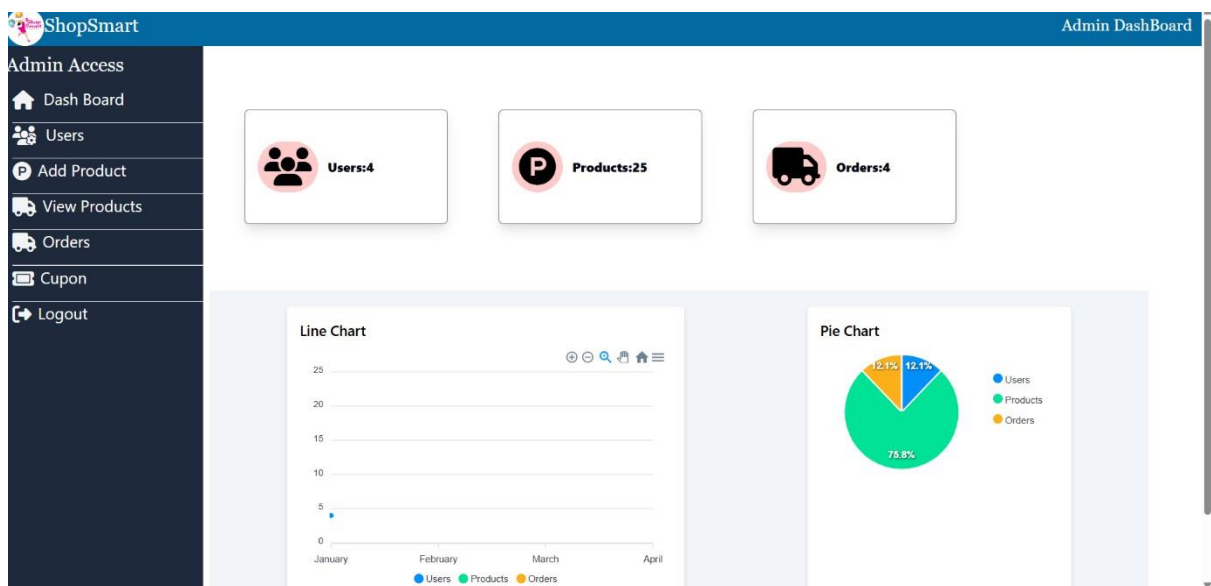
- Full Name:
- Email Address:
- Address / Street:
- Mobile Number:
- City:
- Country / region:
- State / province:
- Zipcode:

A blue "Next" button is located at the bottom right of the form.

Payment Page:



Admin Dashboard:




Add Product Page:

The screenshot displays the ShopSmart Admin Dashboard. The top navigation bar is blue with the ShopSmart logo on the left and 'Admin DashBoard' on the right. A dark blue sidebar on the left lists 'Admin Access' options: Dash Board, Users, Add Product, View Products, Orders, Feedback, Cupon, and Logout. The main content area is white and features the 'Add New Product' form. This form includes fields for Product Name, Description, Original Price, Price, Category, and Stock. It also has a file upload section for images with a 'Choose Files' button and a 'No file chosen' status. A blue 'Add Product' button is at the bottom of the form.

TESTING

- Unit Testing: Write and execute unit tests to validate the functionality of individual components and modules.
 - Integration Testing: Test the integration between frontend and backend components to ensure proper communication and data exchange.
 - End-to-End Testing: Conduct end-to-end testing to validate the entire workflow, including user registration, product browsing, adding items to the cart, checkout process, and order confirmation.
 - Performance Testing: Measure and optimize the performance of the e-commerce platform by analyzing response times, server load, and database performance under various traffic conditions.

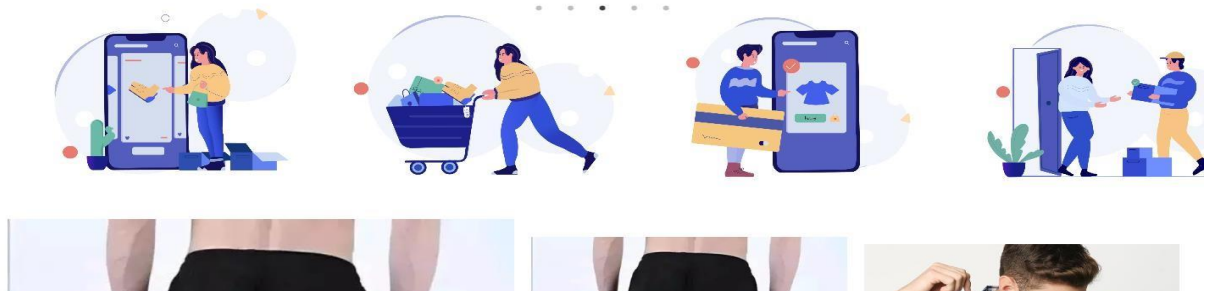
SCREENSHORTS OR DEMO

ShopSmarts

[Sign In](#) [Sign Up](#)



acer
Acer NITRO Gaming Monitors
From ₹13,599
Up to 180Hz Refresh Rate | 1 ms
Just Launched | Shop Now



KNOWN ISSUES

Delayed Cart Updates

Description:

Issue: When users add or remove products from their cart, the changes sometimes do not reflect immediately. Users may have to refresh the page to see an updated cart total or product list.

Impact: Medium — May cause confusion when adjusting product quantities or prices.

Workaround: Users can manually refresh the page to see the correct cart contents.

Status:

In Progress — The issue is being investigated, and performance improvements for real-time cart updates are planned for a future patch.

Session Timeout After Prolonged Inactivity

Description:

Issue: Users who remain inactive for an extended period are automatically logged out without a warning message, losing any unsaved progress in their cart or checkout process.

Impact: High — May result in frustration for users who lose their session during checkout.

Workaround: Users should try to complete their purchases without extended breaks, or re-login to resume their session.

Status:

Planned Fix — A session expiration warning will be implemented to alert users before they are logged out.

. Inconsistent Product Filter Behavior

Description:

Issue: On the product listing page, some filter combinations (e.g., filtering by both brand and price range) do not return the expected results. Certain filters may reset or clear unexpectedly.

Impact: Medium — May limit users' ability to refine search results, leading to a less efficient shopping experience.

Workaround: Users can clear all filters and re-apply them one at a time to avoid conflicts.

Status:

Under Review — The development team is testing filter behavior for consistency and is working on a fix.

FUTURE ENHANCEMENT

Description:

Implement a recommendation engine that suggests products to users based on their browsing history, past purchases, and user behavior.

Benefits:

Increases sales by presenting relevant products to customers.

Improves user engagement and encourages repeat visits.

Implementation:

Use machine learning algorithms to analyze customer behavior and provide tailored product suggestions on the homepage and product pages.

Advanced Search with Natural Language Processing (NLP)

Description:

Enhance the search functionality by incorporating NLP techniques, allowing users to search in a more intuitive, conversational manner (e.g., "Show me affordable smartphones under \$500").

Benefits:

Provides a smoother and more user-friendly search experience.

Increases accuracy in delivering relevant search results.

Implementation:

Integrate NLP models to understand complex queries and improve filtering capabilities.

3. Multi-Currency and Multi-Language Support

Description:

Offer multi-currency pricing and the ability to switch between different languages to support global customers.

Benefits:

Attracts international customers by offering a localized shopping experience.

Increases accessibility for non-English-speaking users.

Implementation:

Add currency conversion based on real-time exchange rates.

Enable language switching and translate key content like product descriptions and checkout instructions.