1. Introduction

Project Title

ShopEZ – One Stop Shop For Online Purchases

Team Id : LTVIP2026TMIDS84991

Team Members

- **Nakka Sharmila** – Frontend Development & UI Design
- **Pydipala Sai Akash**– Backend Development & API Development
- **NeeruKonda Prasanna** – Database Design & Testing
- **Rasmitha Lekha Rambha** – Database Design & Testing

Introduction

In today's digital era, online shopping has become an essential part of everyday life, with consumers expecting fast, secure, and convenient access to products from the comfort of their homes. Traditional shopping methods often involve physical store visits, limited product availability, time constraints, and a lack of real-time order tracking, which creates inconvenience for customers and operational inefficiencies for business owners. To address these challenges, ShopEZ – Smart Online Shopping Platform was developed as a full-stack web application using the MERN stack (MongoDB, Express.js, React.js, and Node.js) with the objective of simplifying and digitalizing the online shopping experience through a centralized, responsive, and user-friendly platform. ShopEZ allows customers to browse products, view detailed descriptions, add items to the cart, and place orders seamlessly, while a demo payment flow simulates real-time transactions and enables users to track order details efficiently. The application ensures smooth navigation, quick product access, and a modern user interface to enhance user experience. Additionally, ShopEZ provides a dedicated admin dashboard that allows administrators to add, update, or remove products and monitor total users, total products, and total orders through real-time statistics, thereby improving operational efficiency and simplifying business management. The system follows a client-server architecture and uses RESTful APIs for structured communication between the frontend and backend, along with secure authentication and role-based access control mechanisms to protect user data and restrict access to administrative functionalities. Overall, ShopEZ aims to make online shopping more accessible, efficient, reliable, and secure for both customers and administrators while demonstrating the practical implementation of full-stack development using the MERN technology stack.

.

2. Project Overview

Purpose

The primary purpose of **ShopEZ Smart Online Shopping Platform** is to provide a digital solution for managing online product browsing, purchasing, and order management efficiently and securely. Traditional shopping methods often involve physical store visits, limited product availability, billing delays, and lack of real-time order tracking. These challenges create inconvenience for customers and operational inefficiencies for business owners. This project aims to eliminate these limitations by offering an automated, user-friendly, and centralized web-based e-commerce platform. The main goals of the project are:

- To simplify the online shopping process for customers.
- To provide a seamless product browsing and cart management experience.
- To enable secure user registration and authentication.
- To allow administrators to manage products, users, and orders efficiently.
- To provide real-time updates on order status and platform statistics.
- To develop a scalable and maintainable e-commerce system using the MERN stack.

By integrating modern web technologies, ShopEZ ensures faster transactions, improved accuracy in order management, enhanced data security, and an intuitive user experience for both customers and administrators. The system is designed to be responsive, efficient, and scalable, making it suitable for small to medium-sized online businesses.

Features

The ShopEZ application provides multiple features based on user roles (Customer and Admin), along with essential technical features to ensure smooth and secure platform functionality.

1. User (Customer) Features

- User Registration and Login using secure authentication.
- Browse and view all available products with detailed descriptions and pricing.
- Search and filter products based on category, price, or preferences.
- Add products to cart and manage cart items (increase/decrease quantity or remove items).
- Place orders through a seamless checkout process.
- Experience a demo payment flow to simulate secure online transactions.
- View order confirmation and order details after successful checkout.
- Track order history and view previous purchases.
- Logout securely from the system.

2. Admin Features

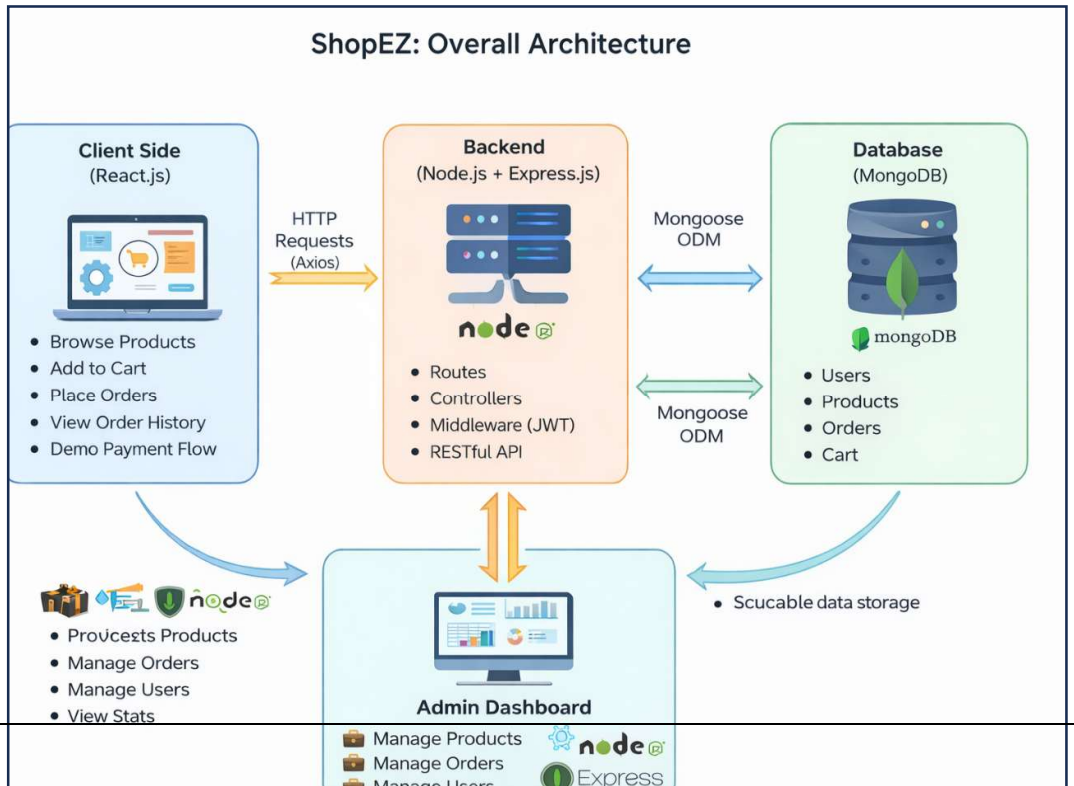
- Secure Admin Login with role-based access control.
- Access a dedicated Admin Dashboard with real-time statistics (Total Products, Total Orders, Total Users).
- Add new products with product details such as name, price, description, and image.
- Update or delete existing products.
- View and manage all customer orders.
- View and manage registered users.
- Monitor overall platform activity and performance.
- Maintain system integrity and enforce platform policies.

3. Technical Features

- RESTful API communication between frontend and backend.
- JWT-based authentication and authorization for secure access.
- Secure password hashing using bcrypt.
- MongoDB database for scalable and efficient data storage.
- Responsive User Interface built using React and modern CSS frameworks.
- Real-time data updates using state management without full page reload.
- Client-server architecture for structured application flow.

.

3. Architecture



Frontend Architecture (React.js)

The frontend of **ShopEZ Smart Online Shopping Platform** is developed using React.js following a component-based architecture. This modular approach ensures reusability, scalability, and easy maintenance of UI components.

Architecture Design

- The UI is divided into reusable components such as Navbar, Product Cards, Cart Component, Admin Dashboard, and Forms.
- React Router is used for seamless navigation between pages without refreshing the browser.
- Axios is used to communicate with backend RESTful APIs.
- CSS and modern UI styling techniques are used to build a responsive and professional interface.
- Role-based rendering ensures separate interfaces for Customer and Admin users.

Frontend Flow

1. User interacts with UI components (buttons, forms, cart actions).
2. Axios sends HTTP requests to backend API endpoints.
3. Backend processes the request and returns a response.
4. React updates the UI dynamically using state management (useState, useEffect).

Key Frontend Modules

- Authentication Pages (Login / Register)
- Home Page (Product Listing)
- Product Details Page
- Cart Page
- Checkout & Demo Payment Page
- Order History Page
- Admin Dashboard
- Manage Products Page
- Manage Users Page
- View Orders Page

The frontend ensures smooth user experience through real-time updates and dynamic rendering without page reloads.

Backend Architecture (Node.js + Express.js)

The backend of ShopEZ is developed using Node.js with the Express.js framework. It handles business logic, authentication, order processing, and database communication.

Architectural Structure

- The backend follows the MVC (Model-View-Controller) pattern:
- Routes – Define API endpoints.
- Controllers – Handle request and response logic.
- Models – Define MongoDB schema using Mongoose.
- Middleware – Handle JWT authentication and role authorization.

Backend Flow

Client Request → Route → Middleware (JWT Verification) → Controller → Database → Response

Key Backend Features

- RESTful API design
- JWT-based Authentication
- Password hashing using bcrypt
- Role-based authorization (Admin / Customer)
- Order management logic
- Cart handling logic
- Error handling middleware

This structured backend ensures scalability, maintainability, and secure data handling.

Database Architecture (MongoDB)

ShopEZ uses MongoDB as a NoSQL database for storing user, product, cart, and order data. Mongoose is used as an ODM to define schemas and interact with the database.

Main Collections

1. Users Collection

Stores registered users and admins.

Fields:

- id
- name
- email

- password (hashed)
- role (admin / customer)
- timestamps

2. Products Collection

Stores product details available in the platform.

Fields:

- name
- description
- price
- image
- category
- stock
- timestamps

3. Orders Collection

Stores customer order information.

Fields:

- id
- userId (reference to Users)
- products (array of product references)
- totalAmount
- paymentStatus (success / pending)
- orderStatus (placed / shipped / delivered)
- createdAt

4. Cart Collection

Stores cart data temporarily.

Fields:

- userId
- products
- quantity

Database Interaction

Controllers use Mongoose methods such as:

- find()
- findById()
- save()
- updateOne()
- deleteOne()
- Relationships are maintained using ObjectId references between collections.

MongoDB ensures:

- Flexible schema design
- Scalability
- High performance
- Efficient data retrieval

4. Setup Instructions

This section explains the necessary software requirements and step-by-step instructions to set up the ShopEZ Smart Online Shopping Platform application on a local system.

.• Prerequisites

Before running the project, ensure the following software and tools are installed:

1. Software Requirements

- Node.js (v16 or higher)
- npm (comes with Node.js)
- MongoDB Atlas account (for cloud database connection) or Local MongoDB installation
- Git (for cloning the repository)
- Code Editor (e.g., Visual Studio Code)

2. Optional Tools

- Postman (for API testing)
- MongoDB Compass (for database visualization)

Installation

Follow the steps below to set up the project locally

Step 1: Clone the Repository

Open terminal or command prompt and run:

```
git clone <repository-link>
```

Navigate into the project directory

```
cd shopez
```

Step 2: Install Backend Dependencies

Navigate to the server folder:

```
cd server
```

Install required dependencies:

```
npm install
```

This installs packages such as:

- express
- mongoose
- jsonwebtoken
- bcryptjs
- cors
- dotenv

Step 3: Configure Environment Variables

Inside the **server** folder, create a `.env` file and add the following:

```
PORT=5000
MONGO_URI=your_mongodb_connection_string
JWT_SECRET=your_secret_key
```

Replace:

- `your_mongodb_connection_string` with your MongoDB Atlas connection string.
- `your_secret_key` with a secure random string.

Step 4: Install Frontend Dependencies

Open a new terminal and navigate to the client folder:

```
cd client
```

Install frontend dependencies:

```
npm install
```

This installs packages such as:

- react
- react-router-dom
- axios
- bootstrap

Step 5: Start the Application

Start Backend Server:

```
cd server
npm start
```

Backend runs on:

```
http://localhost:5000
```

Start Frontend Application:

```
cd client
npm start
```

Frontend runs on:

```
http://localhost:3000
```

After completing these steps, the application will be successfully running on your local system.

5. Folder Structure

The ShopEZ project follows a modular and well-organized folder structure to ensure scalability, readability, and ease of maintenance. The application is divided into two main parts:
Client (Frontend – React.js)
Server (Backend – Node.js + Express.js)

Client (React Frontend)

The frontend is built using React.js and follows a component-based architecture. The folder structure ensures proper separation between reusable components, pages, routing, and utilities.

Client Folder Structure

```
client/
│
├── public/
│   └── index.html
│
├── src/
│   ├── components/
│   ├── pages/
│   ├── admin/
│   ├── context/ (or redux if used)
│   ├── hooks/
│   ├── utils/
│   ├── assets/
│   ├── App.js
│   └── index.js
│
└── package.json
```

Description of Important Folders

1. public/
Contains static files such as `index.html` which acts as the root HTML file.

2. src/
Main working directory of the React application.

- **components/**

  Reusable UI components such as:

- Navbar
- ProductCard
- CartItem
- Footer
- Loader
- ProtectedRoute

- **pages/**
  Contains main customer pages:

- Home Page (Product Listing)
- Product Details Page
- Cart Page
- Checkout Page
- Payment Success Page
- Login Page
- Register Page
- Order History Page

- admin/

Contains admin-specific pages:

- AdminDashboard
- AddProduct Page
- ManageProducts Page
- ManageUsers Page
- ViewOrders Page
- ViewCarts Page
- **hooks/**
  Custom React hooks for reusable logic.
- **utils/**
  Utility functions such as:
  - API configuration
  - Helper functions
- **assets/**
  Images, icons, and static resources.
- **App.js**
  Main component that handles routing using React Router.
- **index.js**
  Entry point of the React application.

This structured frontend ensures modular design and maintainability.

• Server (Node.js Backend)

The backend follows an MVC (Model-View-Controller) pattern to organize logic efficiently.

Server Folder Structure

```
server/
|
```

```
├── config/
├── controllers/
├── models/
├── routes/
├── middleware/
├── utils/
├── uploads/ (if file upload is used)
├── server.js
└── package.json
```

Description of Important Folders

1. config/
Contains configuration files such as:

- Database connection setup

2. controllers/
Contains business logic for handling requests and responses.
Examples:

- userController.js
- authController.js
- productController.js
- orderController.js
- adminController.js
- cartController.js

3. models/
Defines MongoDB schemas using Mongoose.
Examples:

- UserModel.js
- ProductModel.js
- OrderModel.js
- CartModel.js

4. routes/
Defines API endpoints and connects them to controllers.
Examples:

- authRoutes.js
- productRoutes.js
- orderRoutes.js
- adminRoutes.js
- cartRoutes.js

5. middleware/
Contains custom middleware such as:

- JWT authentication middleware
- Role-based authorization
- Error handling middleware

6. utils/
Utility functions such as:

- Error handling classes
- Async error wrapper

7. uploads/ (if implemented)
Stores uploaded documents such as medical files.

8. server.js
Main entry point of the backend application.
Responsible for:

- Initializing Express app
- Connecting to MongoDB
- Registering routes
- Starting the server

6. Running the Application

This section explains how to start and run the **DocSpot: Seamless Appointment Booking for Health** application locally after completing the installation and setup process.

The application consists of two main parts:

- Frontend (React.js)
- Backend (Node.js + Express.js)

Both servers must be running simultaneously for the application to function properly.

• Starting the Backend Server

1. Open a terminal.
2. Navigate to the server directory:

```
cd server
```

3. Start the backend server:

```
npm start
```

If configured correctly, the backend will run on:

```
http://localhost:5000
```

The backend server is responsible for:

- Handling API requests
- Connecting to MongoDB
- Managing authentication and authorization

• Starting the Frontend Server

1. Open a new terminal window.
2. Navigate to the client directory:

```
cd client
```

3. Start the React application:

```
npm start
```

The frontend application will run on:

```
http://localhost:3000
```

The frontend is responsible for:

- Displaying the user interface
- Communicating with backend APIs
- Handling user interactions

• Running Both Servers Simultaneously

To ensure the application works correctly:

- Backend must be running on port 5000
- Frontend must be running on port 3000

The frontend communicates with the backend through API calls (e.g., `http://localhost:5000/api/...`).

Once both servers are running successfully, users can access the application through their web browser at:

```
http://localhost:3000
```

7. API Documentation

The backend of ShopEZ: Smart Online Shopping Platform exposes RESTful APIs to handle authentication, product management, cart operations, orders, payments, and admin operations.

Base URL (Local Development):

```
http://localhost:5000/api
```

1️⃣ Authentication APIs

1. Register User

Endpoint:

```
POST /api/users/register
```

Description:
Registers a new user (Customer or Admin).

Request Body:

```
{
  "name": "John Doe",
  "email": "john@gmail.com",
  "password": "123456",
  "role": "customer"
}
```

Response (Success):

```
{
  "message": "User registered successfully",
  "token": "jwt_token_here"
}
```

2. Login User

Endpoint:

```
POST /api/users/login
```

Description:
Authenticates user and returns JWT token.

Request Body:

```
{
  "email": "john@gmail.com",
  "password": "123456"
}
```

Response (Success):

```
{
  "message": "Login successful",
  "token": "jwt_token_here",
  "user": {
    "id": "12345",
    "role": "customer"
  }
}
```

2️ Product APIs

1. Add Product (Admin Only)

Endpoint: POST /api/products

Description: Admin adds a new product to the store.

Headers: Authorization: Bearer <admin_token>

Request Body:

```
{

  "name": "Wireless Headphones",

  "price": 1999,

  "category": "Electronics",

  "stock": 50,

  "description": "High quality Bluetooth headphones"
```

}**Response:**

```
{
  "message": "Product added successfully",

}
```

2. Get All Products

Endpoint:

```
GET /api/products
```

Description:
Returns list of available products.

Response:

```
[
  {
    "name": "Wireless Headphones",
    "price": 1999,
    "category": "Electronics",
     "stock": 50
  }
]
```

3. Update Product (Admin Only)

Endpoint:

PUT /api/products/:id

Description: Admin updates product details.

Headers:

```
Authorization: Bearer <admin_token>
```

Response:

```
{
  "message": "Product updated successfully"
}
```

4. Delete Product (Admin Only)

Endpoint:
DELETE /api/products/:id

Description: Admin removes a product from store.
Headers: Authorization: Bearer <admin_token>

```
{
 "message": "Product deleted successfully"
}
```

3 Cart APIs

1. Add to Cart

Endpoint: POST /api/cart

Description: Customer adds product to cart.

Headers: Authorization: Bearer <customer_token>

Request Body:

{ "productId": "product_id_here", "quantity": 2 }

Response:

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

2. Get User Cart

Endpoint: GET /api/cart

Description: Returns logged-in user's cart items.

Headers: Authorization: Bearer <customer_token>

Response:

[{ "productName": "Wireless Headphones", "quantity": 2, "price": 1999 }]

3. Remove from Cart

Endpoint: DELETE /api/cart/:id

Description: Removes item from cart.

Headers: Authorization: Bearer <customer_token>

Response:

{ "message": "Item removed from cart" }

4. Order APIs

1. Place Order

Endpoint: POST /api/orders

Description: Customer places order for items in cart.

Headers: Authorization: Bearer <customer_token>

Request Body:

{ "shippingAddress": "Hyderabad, Telangana", "paymentMethod": "COD" }

Response:

{ "message": "Order placed successfully", "status": "pending" }

2. Get User Orders

   Endpoint: GET /api/orders/myorders

Description: Returns all orders of logged-in customer.

Headers: Authorization: Bearer <customer_token>

Response:

[{ "orderId": "12345", "totalAmount": 3998, "status": "shipped" }]

3. Update Order Status (Admin Only)

   Endpoint: PUT /api/orders/:id

Description: Admin updates order status (Processing / Shipped / Delivered / Cancelled).

Headers: Authorization: Bearer <admin_token>

Request Body:

{ "status": "shipped" }

Response:

{ "message": "Order status updated successfully" }

5 Admin APIs

1. Get All Users

   Endpoint: GET /api/admin/users

Description: Returns list of all registered users.

Headers: Authorization: Bearer <admin_token>

2. Get All Orders

   Endpoint: GET /api/admin/orders

Description: Returns all orders placed on the platform.

Headers: Authorization: Bearer <admin_token>

Authentication Requirement

All protected routes require:

Authorization: Bearer <JWT_TOKEN>

JWT token is generated during login and must be included in request headers for secure access.

8. Authentication

Authentication and authorization in **DocSpot: Seamless Appointment Booking for Health** are implemented using **JWT (JSON Web Token)** and role-based access control to ensure secure access to the system.

The system verifies user identity during login and restricts access to specific features based on user roles (Admin, Doctor, Patient).

• Authentication Mechanism

1 User Registration

- When a new user registers, their password is encrypted using **bcrypt** before storing it in the database.
- The encrypted (hashed) password ensures that even if the database is compromised, the actual password cannot be retrieved.

Example process:

1. User enters password.
2. Backend hashes password using bcrypt.
3. Hashed password is stored in MongoDB.

2 User Login

- During login, the system:
    1. Verifies the email.
    2. Compares the entered password with the hashed password using bcrypt.
    3. If valid, generates a JWT token.

• JWT (JSON Web Token)

JWT is used for stateless authentication.

Token Generation

- After successful login, the backend generates a JWT token.
- The token contains:
  - User ID
  - User Role
  - Expiry time

Example Token Payload:

```
{
  "id": "user_id_here",
  "role": "customer",
  "exp": "expiry_time"
}
```

The token is signed using a secret key stored in the `.env` file:

```
JWT_SECRET=your_secret_key
```

Token Storage

- The generated JWT token is sent to the frontend.
- It is stored in **localStorage** (or sessionStorage).
- The token is attached to future API requests in the Authorization header:

```
Authorization: Bearer <JWT_TOKEN>
This ensures only authenticated users can access protected routes.
```

• Authorization Mechanism (Role-Based Access Control)

Authorization determines what actions a user can perform.

The system supports three roles:

1 Customer

- Browse products
- Add products to cart

- Update cart items
- Place orders
- View order history

2️ Admin

- View assigned appointments
- Add new products
- Update product details
- Delete products
- View all users
- View and manage all orders
- Monitor overall system statistics

Middleware Implementation

A custom authentication middleware is used in the backend:

1. Extracts JWT from request header.
2. Verifies token using the secret key.
3. Decodes user information.
4. Attaches user details to request object.
5. Allows access if valid; otherwise returns unauthorized error.

Example Flow:

Client Request → JWT Middleware → Role Check → Controller → Response

• Session Handling

The application uses **stateless authentication**.

- No server-side sessions are stored.
- Each request must include a valid JWT.
- This improves scalability and performance.

• Security Measures

- Password hashing using bcrypt
- Secure JWT signing
- Environment variables for sensitive data
- Role-based route protection
- Unauthorized access returns HTTP 401 or 403

This authentication and authorization system ensures:

- Secure user identity verification

- Restricted access to sensitive features
- Data privacy and system integrity
- Scalable and maintainable security architecture

9. User Interface

The User Interface (UI) of ShopEZ Smart Online Shopping Platform is designed to be responsive, user-friendly, and intuitive. The frontend is developed using React.js with Bootstrap (or CSS) to ensure a modern, clean, and interactive design.

The UI is divided based on user roles: Customer and Admin. Each role has a dedicated dashboard with relevant features.

• Login Page

Description:
The login page allows registered users (Customer or Admin) to securely log into the system using their email and password.

UI Elements:

- Email input field
- Password input field
- Login button
- Link to registration page
- Error message display (if invalid credentials)

📸 *Insert Screenshot: Login Page*

• Registration Page

Description:
New users can create an account by entering their details.

UI Elements:

- Name field
- Email field
- Password field
- Role selection (User/Admin)
- Register button

📸 *Insert Screenshot: Registration Page*

•Home Page (Product Listing Page)

Description:
After login, customers are redirected to the Home Page where they can browse all available products.

UI Features:

- Product cards with:
- Product Image
- Product Name
- Product Price
- Short Description
- "View Details" button
- "Add to Cart" button
- Navigation bar with Cart and Logout options

📸 *Insert Screenshot: Patient Dashboard*

• Product Details Page

Description:
Displays detailed information about a selected product.

UI Elements:

- Date selection
- Large product image
- Product name
- Product description
- Price
- Add to Cart button

📸 *Insert Screenshot: Appointment Booking Form*

• Cart Page

Description: Displays all products added to the cart before checkout.

UI Features:

• List of selected products

• Product name and price

• Quantity update option

• Remove item button

• Total price calculation

• Proceed to Checkout button

- **Checkout / Order Page**
- Description: Allows customers to confirm and place their order.

  UI Elements:

  • Order summary

  • Total amount

  • Demo payment button (if implemented)

  • Place Order button

  • Success message after order placement

- **Order History Page**

Description: Customers can view all their previous orders.

UI Features:

• Order ID

• Ordered products

• Order date

• Total amount

• Order status (Placed / Shipped / Delivered – if implemented)

- **Admin Dashboard**

Description: Admin oversees and manages the entire ShopEZ platform.

UI Features:

- Dashboard cards displaying:
- Total Users
- Total Products
- Total Orders
- Navigation sidebar
- Quick access to product management

📸 Insert Screenshot: Admin Dashboard

• Add Product Page (Admin)

Description: Admin can add new products to the system.

UI Elements:

• Product name field

• Description field

• Price field

• Category field (if implemented)

• Image upload option

• Submit button

📸 Insert Screenshot: Add Product Page

• Manage Products Page (Admin)

Description: Admin can update or delete existing products.

UI Features:

• Product list table

• Edit button

• Delete button

• Real-time updates after modification

📸 Insert Screenshot: Manage Products Page

• UI Characteristics

Fully responsive design for different screen sizes

• Clean and minimal layout

• Smooth navigation using React Router

• Role-based dashboard rendering

• Real-time cart updates without page refresh

• Interactive buttons and alerts

• Secure access to admin pages

• User-friendly checkout experience

10. Testing

Testing is an essential phase in the development of ShopEZ Smart Online Shopping Platform to ensure reliability, functionality, security, and performance. The application was tested at multiple levels, including frontend, backend, and database interactions.

Testing Strategy

The project follows a **manual and functional testing strategy** to validate system behavior and ensure all features work as expected.

The testing process includes:

- Unit Testing (individual components and APIs)
- Integration Testing (frontend-backend interaction)
- Authentication & Authorization Testing
- Role-Based Access Testing
- CRUD Operation Testing
- Error Handling Testing

• Frontend Testing

Frontend testing ensures that all UI components render correctly and user interactions function properly.

Tested Scenarios:

- Login and Registration form validation
- Navigation between pages using React Router
- Role-based dashboard rendering (Customer/Admin)
- Product listing and display functionality
- Add to Cart and Remove from Cart workflow
- Cart total calculation
- Checkout and order placement workflow
- Display of success/error messages
- Real-time cart updates without page refresh

Tools Used:

- Browser Developer Tools (Chrome DevTools)
- Manual UI interaction testing

• Backend Testing

Backend testing ensures APIs return correct responses and handle data securely.

Tested Scenarios:

- User registration and login functionality
- JWT token generation and verification
- Product CRUD operations (Add, Update, Delete)
- Cart management APIs
- Order placement and retrieval APIs
- Admin-only route protection
- Unauthorized access handling
- Error responses for invalid inputs

Tools Used:

- Postman (for API testing)
- Console logs for debugging

• Database Testing

Database testing verifies correct storage and retrieval of data.

Tested Scenarios:

- Proper creation of user records
- Password hashing verification
- Product data insertion and updates
- Cart item storage and removal

- Order data consistency
- Relationship validation between Users, Products, Cart, and Orders collections

Tools Used:

- MongoDB Atlas Dashboard
- MongoDB Compass (optional)

• Authentication & Authorization Testing

Special attention was given to security-related testing.

Verified:

- Password encryption using bcrypt
- JWT token generation after login
- Protected routes requiring valid tokens
- Role-based restrictions (Admin,Customer)
- Proper error handling for expired or invalid tokens

• Integration Testing

Integration testing ensures that frontend and backend communicate correctly.

Verified:

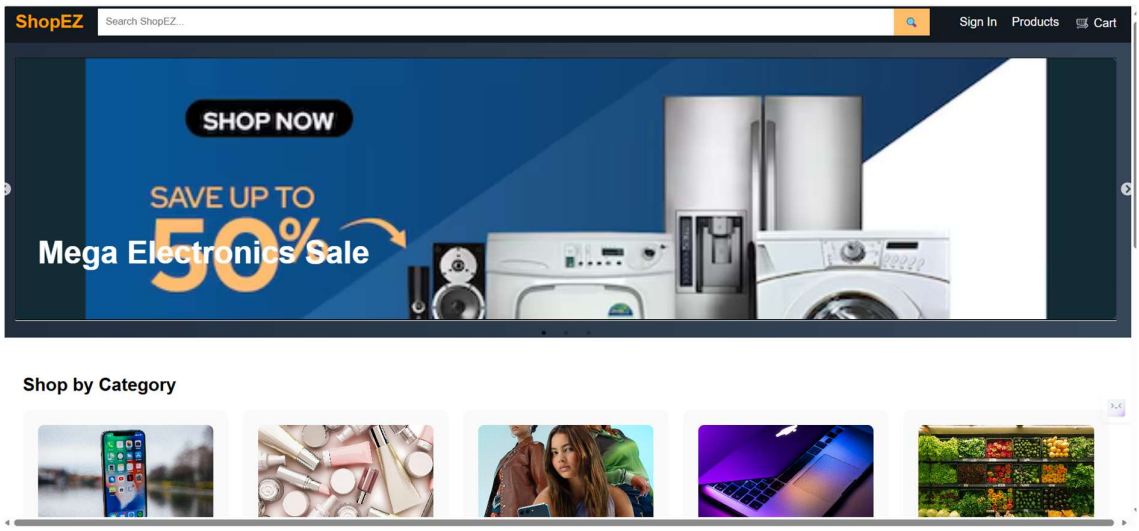
- Axios API calls returning correct responses
- Real-time updates in UI after backend response
- Proper handling of server errors
- Data consistency between client and server

• Testing Outcomes

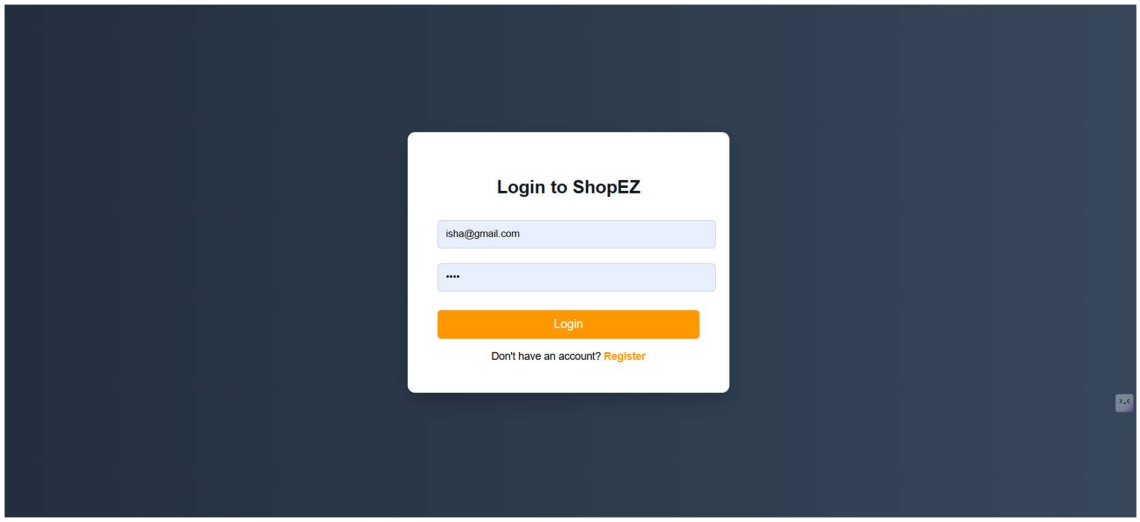
- All core functionalities were successfully validated.
- Role-based access control worked correctly.
- Product management system functioned without errors.
- Cart and order workflow operated smoothly.
- Security mechanisms prevented unauthorized access.
- The application demonstrated stable performance during testing.

11. Screenshots or Demo

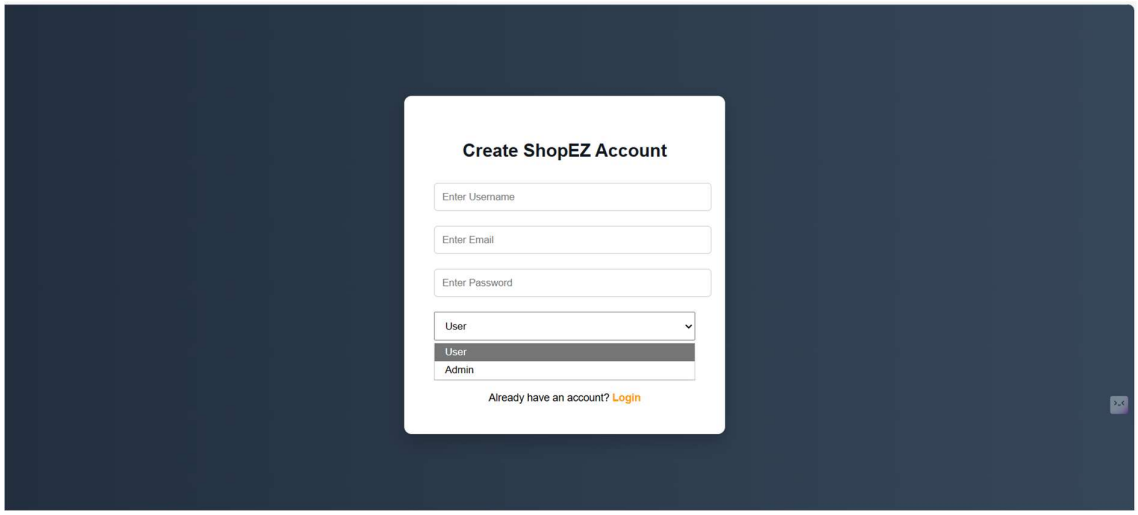
Landing Page



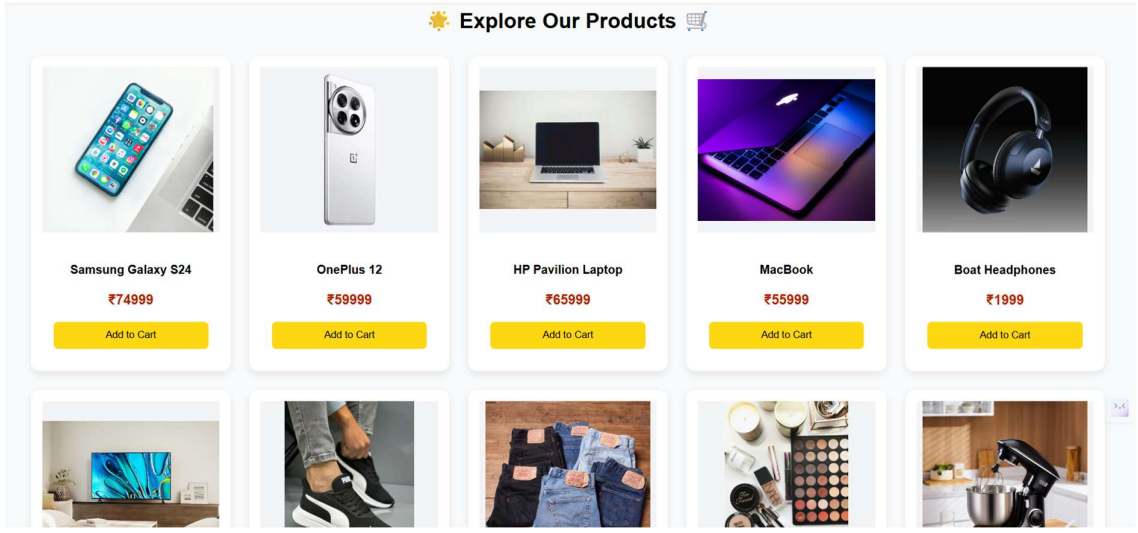
Login page



Registration Page

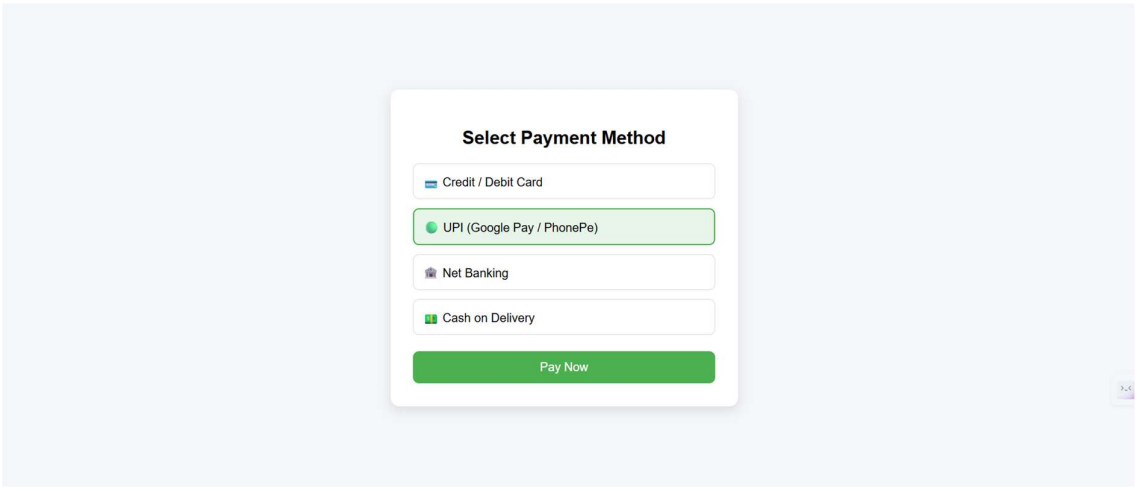


Product Details Page

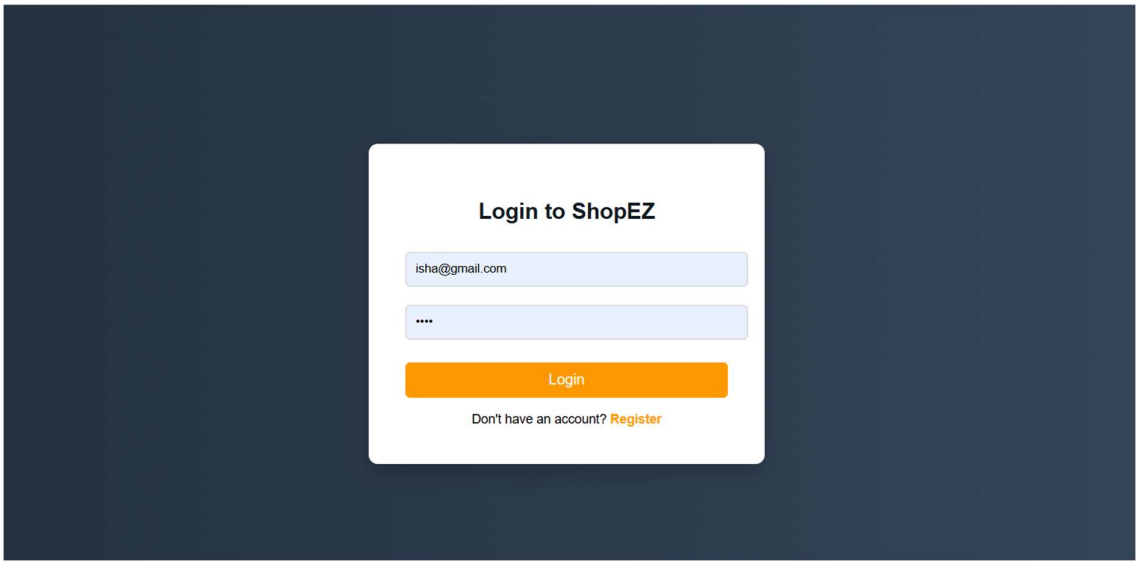


Cart Page

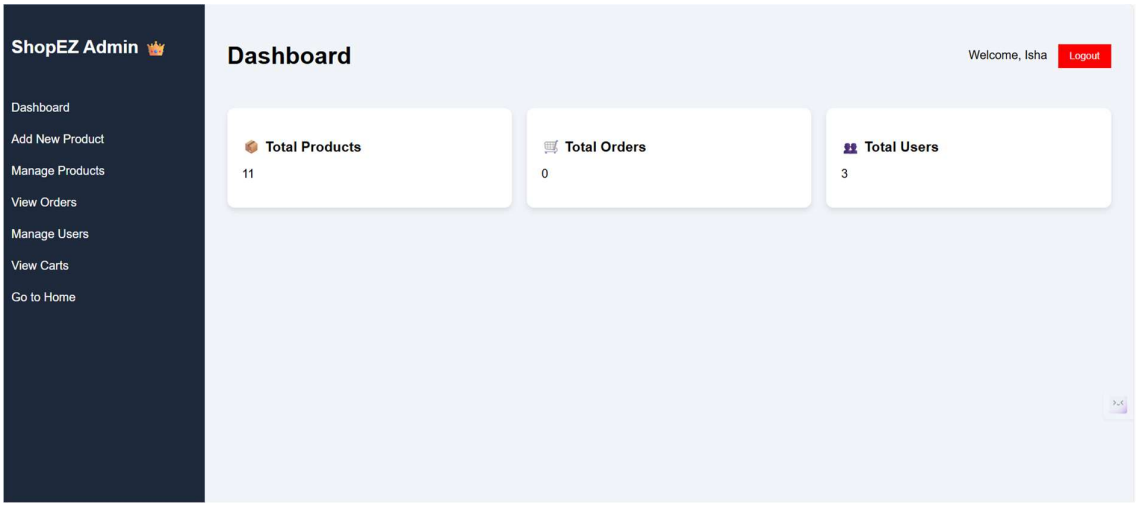
Checkout page



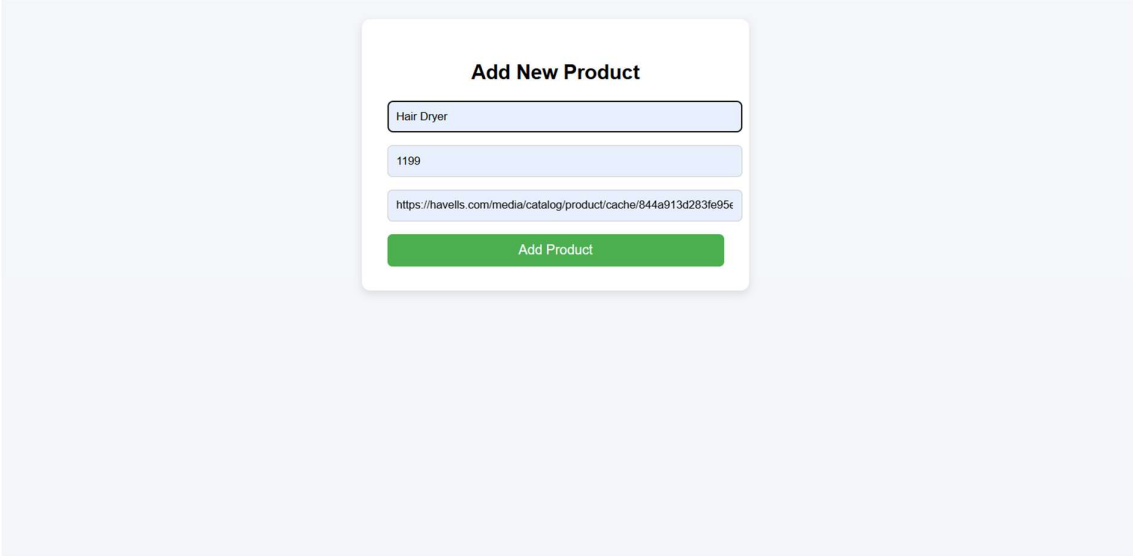
Admin Login Page



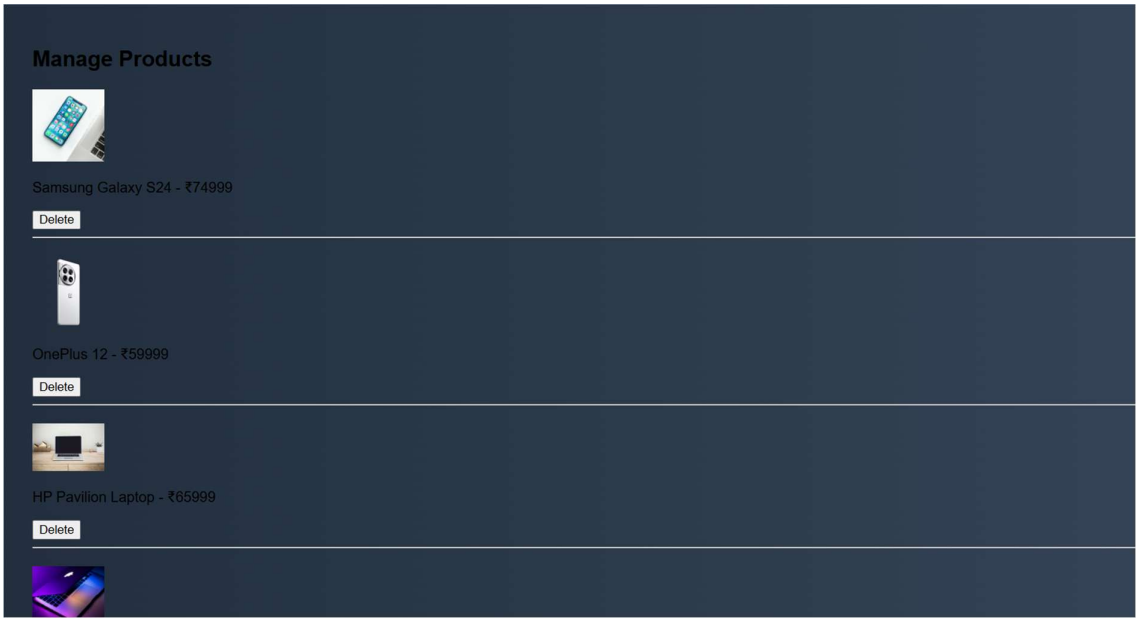
Admin Dashboard



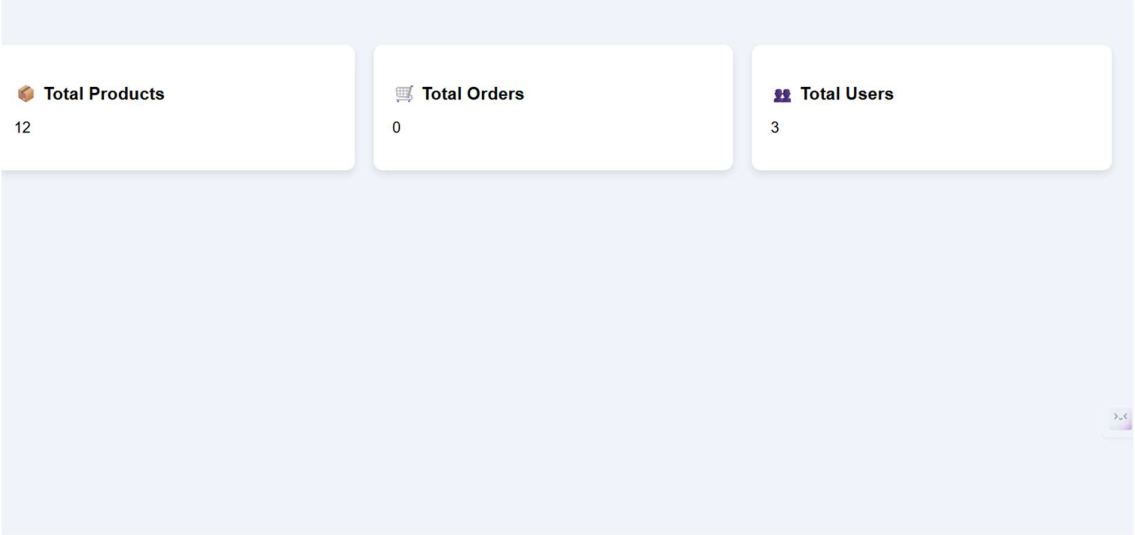
Add New Product Page



Manage Products Page



Admin Statistics Overview (Total Users / Products / Orders)



12. Known Issues

While ShopEZ Smart Online Shopping Platform is fully functional and stable for academic and small-scale deployment purposes, the following known issues and limitations have been identified. These should be considered for future improvements.

Limited Real-Time Notifications

The system does not currently provide real-time push notifications.
Users must manually refresh the page to see updated cart or order status.

No live notification system is implemented for order updates.

No Email or SMS Alerts

Order confirmations, cancellations, or status updates are not sent via email or SMS.
All order updates are visible only within the application dashboard.
No external communication system is integrated.

Demo Payment Only (No Real Payment Gateway)

The application includes only a demo payment success flow.
No real payment gateway integration (such as Razorpay or Stripe).
Transactions are simulated for demonstration purposes only.
No real-time payment verification system

Basic Error Handling Messages

Some backend validation errors may return generic messages instead of detailed, user-friendly explanations.

• Limited frontend validation messages.

• Advanced centralized error handling is not fully implemented.

Limited Scalability Optimization

The current system does not include:

- Load balancing
- Caching mechanisms
- Performance optimization for high traffic

This may affect performance under heavy user load.

Manual Testing Only

The project currently relies on manual testing.
Automated testing frameworks such as Jest or Mocha are not implemented.

Image Storage Limitation

If product image upload is implemented:
• Images may be stored locally in the server folder.
• Cloud storage integration (e.g., AWS S3, Cloudinary) is not implemented.
• No advanced image compression or optimization.

UI Improvements Needed for Mobile Optimization

Although responsive, certain layouts may require

Better alignment on smaller devices

Improved spacing and font scaling

Enhanced mobile navigation experience

These known issues do not affect the core shopping functionality (product browsing, cart, order placement, admin management) but highlight areas that can be enhanced to improve scalability, security, performance, and overall user experience.

13. Future Enhancements

To improve scalability, functionality, and user experience, several enhancements can be implemented in future versions of ShopEZ Smart Online Shopping Platform.

Online Payment Integration

Integrate secure payment gateways such as:
• Razorpay
• Stripe
• Credit/Debit Card Payments
• UPI & Net Banking

This will allow customers to make real-time secure payments during checkout.

Email and SMS Notifications

Implement automated notifications for:

- Order confirmation
- Payment confirmation
- Order shipped/delivered updates
- Account registration confirmation
- Admin alerts for new orders

This can be achieved using services like email APIs or SMS gateways.

Product Rating and Review System

Allow customers to:
Rate products
Write reviews
View product ratings before purchase
This improves transparency, trust, and decision-making for customers.

Advanced Search and Filtering

Enhance product browsing by adding:

• Category-based filtering

• Price range filtering

• Search by product name

• Sorting (Low to High / High to Low)

• Stock availability filtering

This improves user experience and shopping efficiency.

Admin Analytics Dashboard

Develop a detailed analytics panel for administrators showing:

- Total revenue overview
- Monthly sales trends
- Top-selling products
- User growth statistics
- Order status distribution

This helps in business decision-making and performance tracking.

Inventory Management System

Implement advanced inventory tracking:
• Automatic stock deduction after order placement
• Low stock alerts
• Out-of-stock product management
• Inventory reports
This improves operational efficiency.

Mobile Application Development

Develop a cross-platform mobile application using:

• React Native or

• Flutter

This would increase accessibility, engagement, and business reach.

Cloud Image Storage Integration

Integrate cloud storage services such as:

• Cloudinary

• AWS S3

Benefits:

• Scalable image hosting

• Improved performance

• Reduced server storage dependency

• Better image optimization

Automated Testing Implementation

Introduce automated testing using:

- Jest (Frontend testing)
- Mocha/Chai (Backend testing)

This would improve code reliability and maintainability.

Token Refresh Mechanism

Implement refresh tokens to:

- Improve user session management
- Prevent frequent login interruptions

Order Tracking System

Add real-time order tracking with stages such as:

• Order Placed

• Processing

• Shipped

• Out for Delivery

• Delivered

This improves transparency and customer satisfaction.

These enhancements would transform ShopEZ into a scalable, production-ready e-commerce platform capable of handling large user bases and real-world business operations efficiently and securely.