ShopEZ: E-commerce Application

Naan Mudhalvan Project

Team Members	NM ID
Josan George	**D923AA0CB7B196088D54FIE48A1999BF**
Madaesh D	**3FB300AA214212582C24C52A04689CF6**
Mohammed Zohair	**F0398EA737CA9E4BAC746E441C34FC68**
Keerthika I	**ED1446581009535987E4D54733933C77**

Table of Contents

ABSTRACT:

ShopEZ is a versatile e-commerce application built using the **MERN stack (MongoDB, Express, React, Node.js)** designed to deliver a smooth and intuitive online shopping experience. It serves as a one-stop destination for customers seeking a wide range of products, providing effortless navigation through product catalogs, detailed descriptions, and personalized recommendations.

Through a client-server architecture, **ShopEZ** enables seamless product discovery, secure user authentication, efficient cart and order management, and a reliable payment processing system. It offers features that support a personalized and engaging shopping experience, including user-friendly filtering, a streamlined checkout process, and instant order confirmations. For sellers, **ShopEZ** provides a robust dashboard that simplifies order management, monitors customer interactions, and offers analytics to support business growth.

This documentation outlines the requirements, architecture, features, and implementation steps needed to develop **ShopEZ using MERN**, aiming to set a new standard in online shopping experiences for both buyers and sellers.

Keywords: E-commerce, MERN Stack, Online Shopping, Secure Checkout, Seller Dashboard

1. INTRODUCTION

In recent years, the demand for online shopping has surged, driven by a need for convenient, efficient, and personalized shopping experiences. **ShopEZ** is an e-commerce platform designed to meet this demand, offering a user-friendly environment where customers can explore products, view details, and make purchases with ease. Built using the MERN stack, ShopEZ leverages MongoDB for data storage, Express.js for server functionality, React for a dynamic front-end, and Node.js for back-end operations, ensuring a scalable and efficient shopping experience.

ShopEZ emphasizes accessibility, intuitive navigation, and personalized recommendations, making it easy for customers of all technical backgrounds to browse and purchase products. Core features include secure checkout, real-time order updates, and personalized suggestions. Additionally, ShopEZ provides sellers with a comprehensive dashboard that enables efficient order management, customer insights, and analytics to drive growth. This documentation outlines the technical architecture, core functionalities, and workflows in ShopEZ, highlighting how it combines modern technologies to create an engaging and seamless online shopping experience.

1.1. PURPOSE

The purpose of the **ShopEZ** e-commerce platform is:

- To provide a centralized online platform for convenient, personalized shopping experiences.
- To enable users to browse a wide range of products, view detailed descriptions, and make purchases securely.
- To offer sellers a robust dashboard for managing product listings, tracking orders, and accessing customer analytics to support business growth.

- To ensure a seamless, secure checkout process, protecting users' personal and payment information.
- To create a scalable, efficient solution for both customers and sellers, designed to enhance the e-commerce experience.

1.2. SCOPE

The scope of the ShopEZ platform includes the development, deployment, and maintenance of a comprehensive e-commerce system using the MERN stack. Key components and features within this scope include:

- **User Management** : Supports user registration, login, profile management, and secure session handling.
- **Product Catalog Management** : Allows sellers to create, update, organize, and publish product listings with details like images, descriptions, prices, and categories.
- **Personalized Recommendation** : Provides customers with product suggestions based on their browsing history and preferences.
- **Shopping Cart & Checkout :** Enables users to add items to a cart, enter shipping details, and proceed with secure payment options.
- **Order Tracking** : Allows users to track order status from placement to delivery and receive instant order confirmations.
- **Payment Processing** : Integrates with payment gateways to facilitate secure transactions.
- **Cross-Device Accessibility** : Ensures compatibility across devices (desktops, tablets, and smartphones) for seamless access from any location.
- **Admin Dashboard** : Provides an administrative dashboard for managing users, products, and sales analytics.
- **Seller Dashboard** : Includes features for sellers to monitor orders, track customer interactions, and view sales metrics.

1.3. PROJECT OBJECTIVE

The objective of the **ShopEZ** project is to develop a versatile and user-friendly e-commerce platform using the MERN stack. This platform aims to deliver an enjoyable shopping experience for users and a powerful management system for sellers by incorporating essential e-commerce functionalities, including secure checkout, order tracking, product recommendations, and sales analytics.

---

2. SYSTEM REQUIREMENTS

2.1. HARDWARE

- **Operating System**: Windows 8 or higher
- **RAM**: 4 GB minimum (8 GB recommended for optimal performance)

2.2. SOFTWARE

- **Node.js**: LTS version for both back-end and front-end development
- **MongoDB**: For database management, using either MongoDB Atlas or a local instance
- **React.js**: Front-end framework for building the user interface
- **Express.js**: Back-end framework for handling server operations and APIs
- **Git**: For version control and collaboration
- **Code Editor**: e.g., Visual Studio Code for development
- **Web Browsers**: At least two browsers installed for testing compatibility (e.g., Chrome and Firefox)

2.3. NETWORK

- **Bandwidth**: 30 Mbps for a smooth online experience and faster testing.

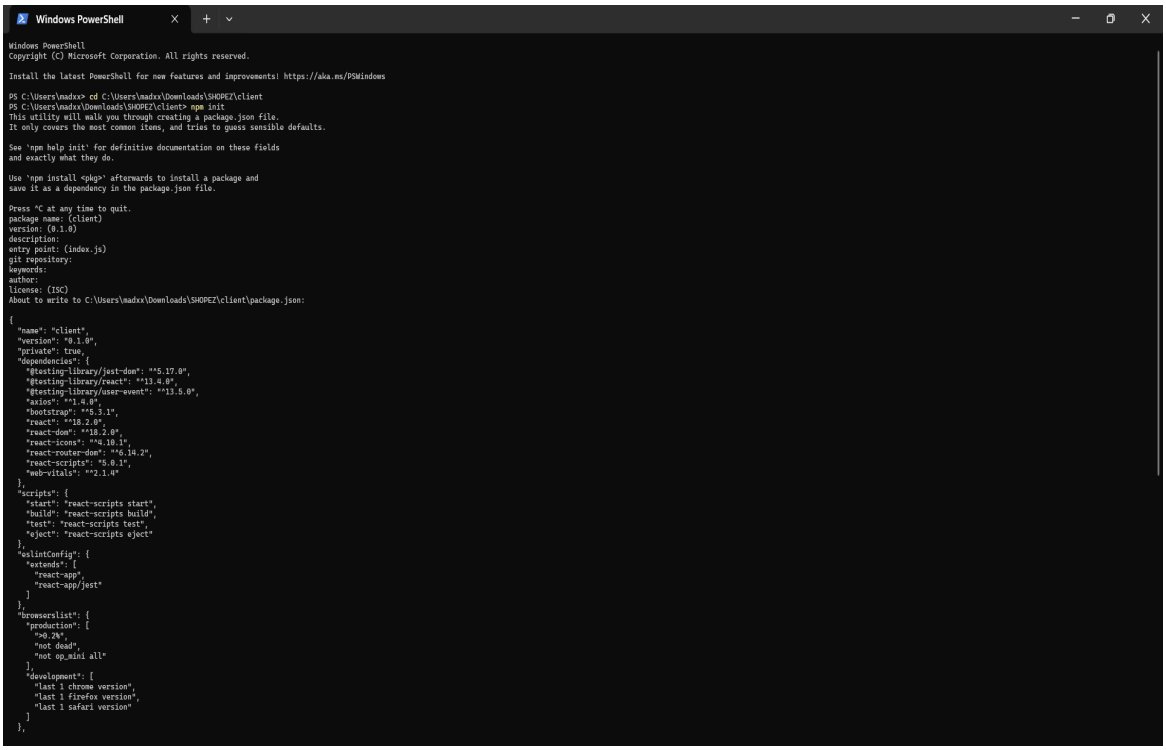
---

3. PRE-REQUISITES

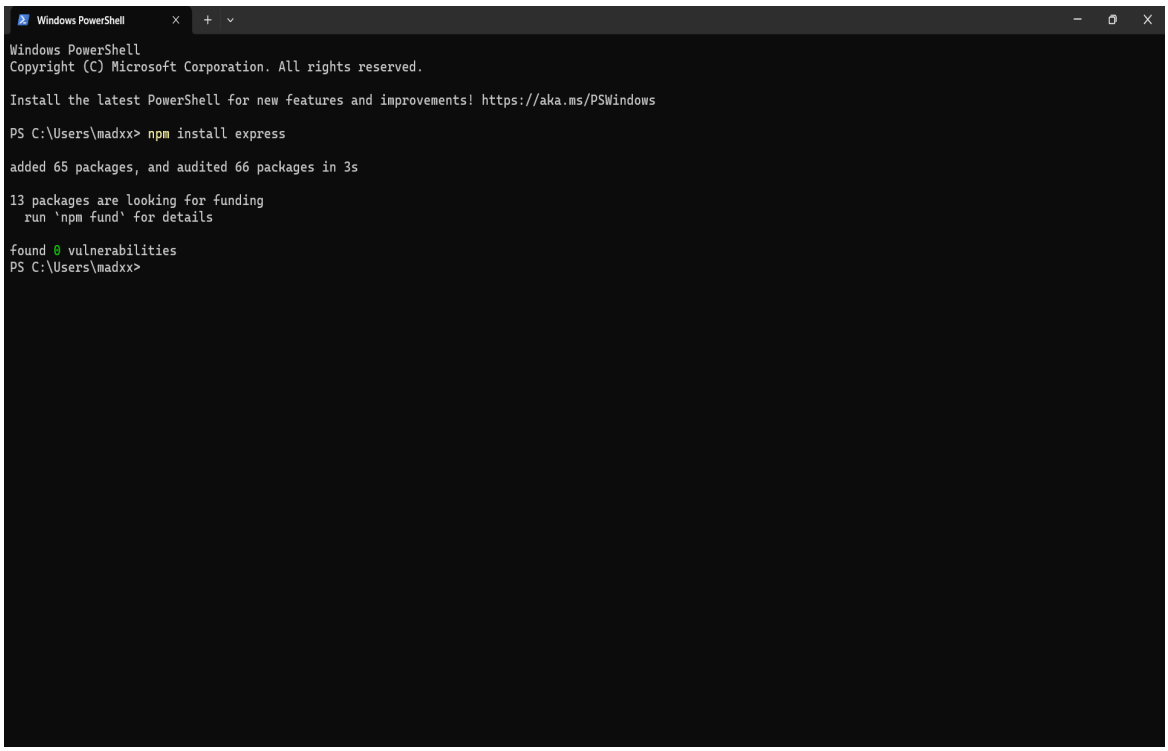
To develop the **ShopEZ** e-commerce application using Node.js, Express.js, MongoDB, and React.js, here are the essential prerequisites:

Key Prerequisites

- **Node.js and npm**:
  Node.js is a runtime environment that enables JavaScript to run server-side, offering scalability and efficiency for network applications.
  - **Download**: Node.js
  - **Installation Instructions**: Node.js Installation Guide
  - Run npm init to set up the project dependencies.

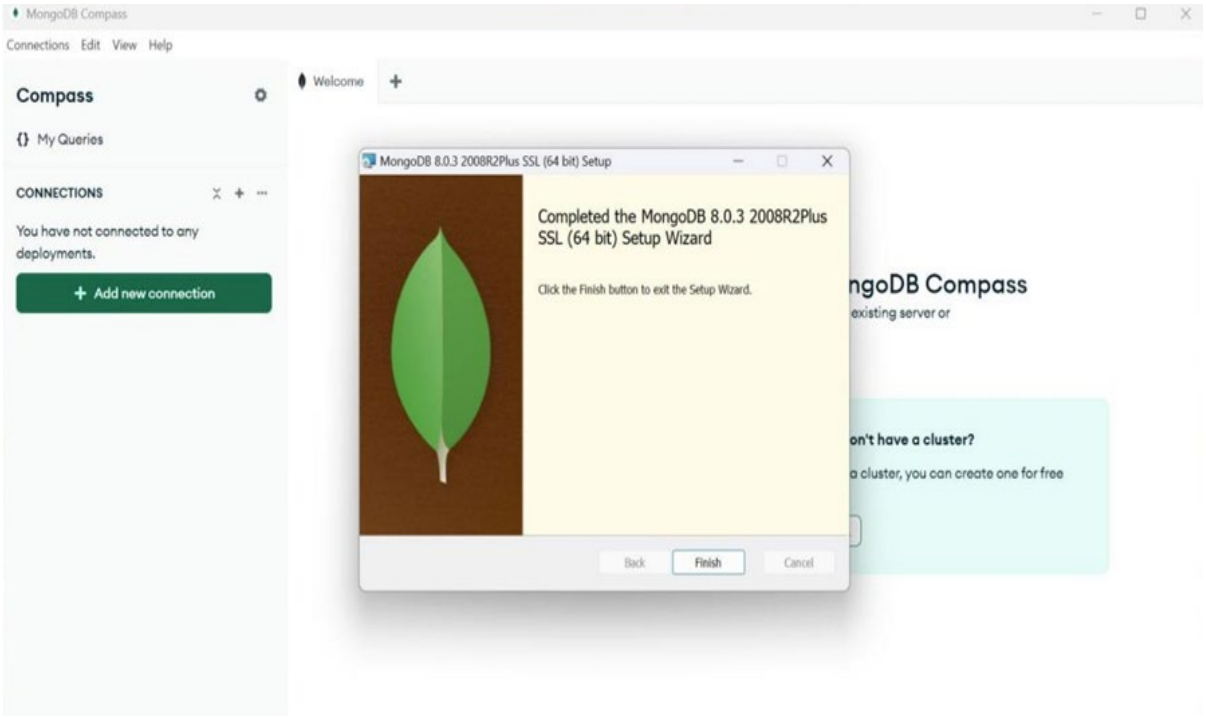


- **Express.js**:
  Express.js is a lightweight and efficient web framework for Node.js that simplifies the creation of APIs and web applications, supporting routing, middleware, and a modular structure.
    - **Installation**: Open your terminal and run: npm install express



- **MongoDB**:
  MongoDB is a NoSQL database ideal for handling diverse data types, offering scalability and flexibility with JSON-like storage. It's highly compatible with Node.js, making it suitable for large-scale applications.
    - **Download**: [MongoDB Community Server]

- **Installation Instructions**: MongoDB Installation Guide



- **React.js**:
  React.js is a widely used JavaScript library for building interactive user interfaces. It enables reusable UI components and dynamic web applications.
  - **Installation Guide**: React.js Installation
- **HTML, CSS, and JavaScript**:
  Basic knowledge of HTML, CSS, and JavaScript is essential for structuring, styling, and enabling interactivity on the client-side.
- **Database Connectivity**:
  Use a MongoDB driver or an ODM like Mongoose to connect the Node.js server with MongoDB for CRUD operations.
- **Additional UI Libraries**:
  Utilize Material-UI and Bootstrap for enhanced styling and responsive components.

Install dependencies:

Client:

cd client
npm install

Server:

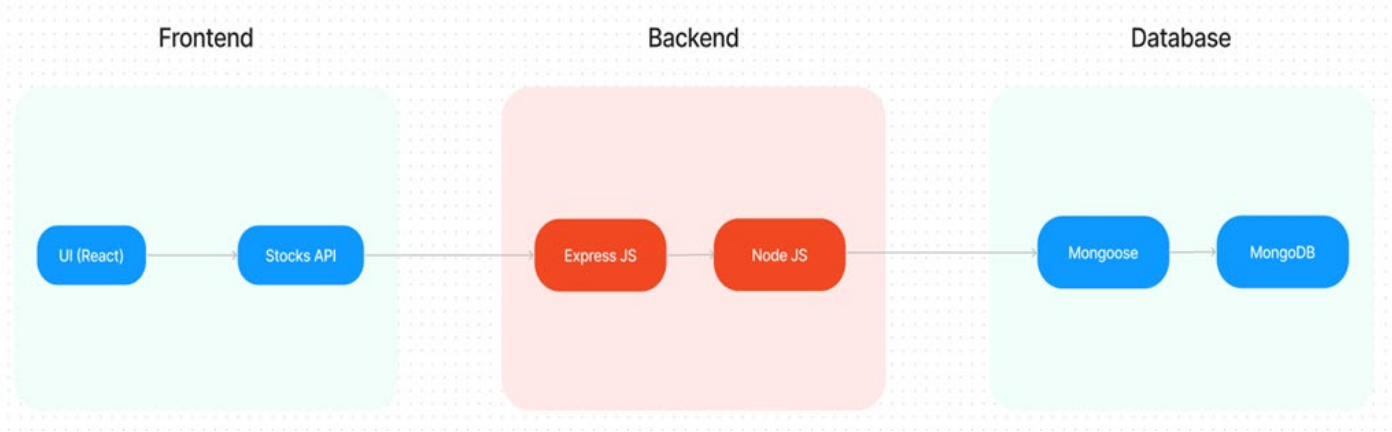
cd server
npm install

Start the Development Server

- To launch the development server, execute:
  npm start
- Access the **ShopEZ** app at http://localhost:3000 by default.

---

4. ARCHITECTURE



4.1. TECHNICAL ARCHITECTURE

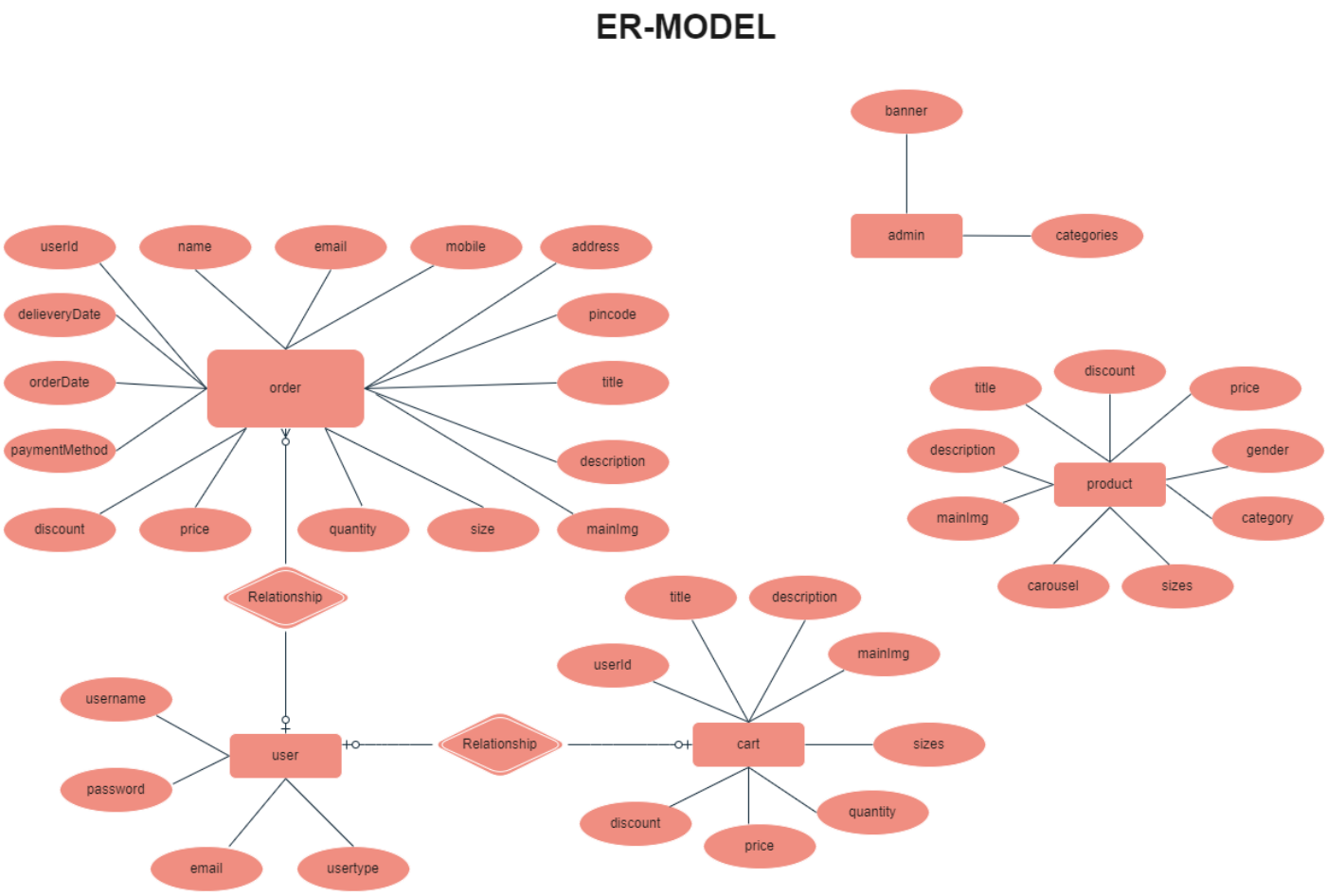
The technical architecture of the **ShopEZ** application follows a client-server model, with the front-end serving as the client and the back-end as the server. This structure facilitates efficient data flow and responsive interactions for users browsing and purchasing products.

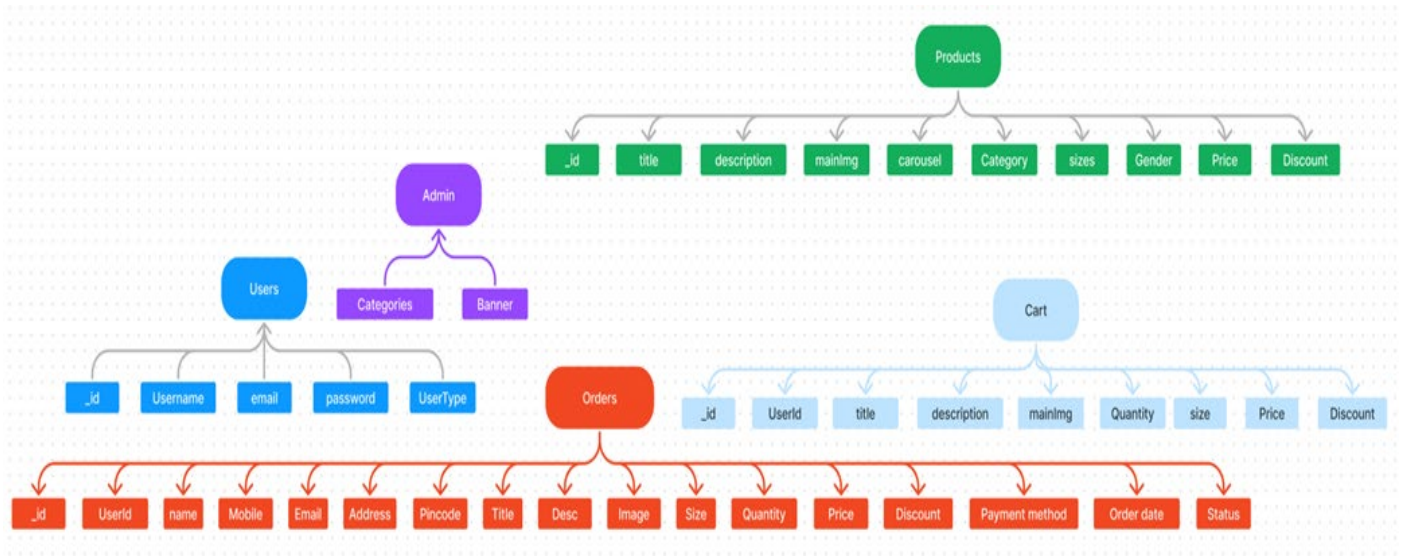
- **Client**:
  The Client, developed in React, handles user interface elements and product presentation. Using libraries like Axios, the front-end communicates seamlessly with the back-end through RESTful APIs. Additionally, the frontend incorporates Material-UI and Bootstrap to create a polished, responsive UI that enhances the user experience across various devices.
- **Server**:
  The server, built with Express.js, manages server-side logic and database interactions. MongoDB is used for data storage, offering a flexible, scalable solution for handling structured and unstructured data, including user details, product information, and order records.
- **Data Exchange**:
  RESTful APIs facilitate communication between the front-end and back-end, ensuring smooth data retrieval and updates. This modular approach also enables flexibility in expanding or modifying the system in the future.
- **Authentication and Security**:
  JWT (JSON Web Tokens) are implemented to secure user sessions, ensuring role-based access for customers, sellers, and admins. This token-based authentication ensures that only authorized

users can access restricted actions, such as managing product listings or viewing sensitive order details.

- **Real-Time Updates**:
  The platform provides real-time notifications, such as instant order confirmations and status updates, enhancing user engagement and transparency. For sellers, the dashboard includes analytics on sales trends and customer behavior, supporting data-driven decisions.
- **Third-Party Integrations**:
  Payment gateways are integrated for secure transactions, supporting multiple payment methods for a convenient checkout process. Additionally, analytics tools are incorporated into the admin dashboard to visualize metrics on product engagement, sales performance, and customer demographics.

5. ER-DIAGRAM


ER-MODEL
banner
admin
categories
userId
name
email
mobile
address
delieveryDate
pincode
orderDate
order
title
paymentMethod
description
discount
price
quantity
size
mainImg
title
discount
price
description
gender
product
mainImg
category
carousel
sizes
Relationship
title
description
userId
mainImg
username
user
Relationship
cart
sizes
password
discount
quantity
price
email
usertype


Products
_id
title
description
mainImg
carousel
Category
sizes
Gender
Price
Discount
Admin
Users
Categories
Banner
Cart
_id
Username
email
password
UserType
Orders
_id
UserId
title
description
mainImg
Quantity
size
Price
Discount
_id
UserId
name
Mobile
Email
Address
Pincode
Title
Desc
Image
Size
Quantity
Price
Discount
Payment method
Order date
Status

The **ShopEZ** platform contains several key entities within its database, representing different collections and their attributes. The **primary entities** in the **ShopEZ** database are:

a. USERS Entity

- **Represents**: The registered users on the ShopEZ platform, including both customers and admins.
- **Attributes**:
    - **userId:** Unique identifier for each user (primary key).
    - **username:** Username associated with the user account.
    - **email:** User's email address.
    - **password:** Encrypted password for secure login.
    - **userType:** Defines the user role (e.g., customer, admin).
    - **name:** Full name of the user.
    - **mobile:** Contact number.
    - **address:** Physical address of the user.
    - **pincode:** Area pincode for delivery purposes.

b. PRODUCTS Entity

- **Represents**: The various products available for sale on ShopEZ.
- **Attributes**:
    - **productId:** Unique identifier for each product.
    - **title:** Product title or name.
    - **description:** Brief description of the product.
    - **mainImg:** Main image URL of the product.
    - **carousel:** Additional images for product display.
    - **category:** Category the product belongs to (e.g., electronics, fashion).
    - **sizes:** Available sizes for the product (if applicable).
    - **gender:** Target audience for the product (e.g., men, women).
    - **price:** Price of the product.
    - **discount:** Discount applied to the product, if any.

c. CART Entity

- **Represents**: The shopping cart for each user, storing items added by the user for potential purchase.
- **Attributes**:
    - **cartId:** Unique identifier for each cart item.
    - **userId:** References the user who added the item.
    - **title:** Title of the product added to the cart.
    - **description:** Brief description of the cart item.
    - **mainImg:** Main image URL of the product in the cart.
    - **quantity:** Quantity of the product in the cart.
    - **size:** Selected size of the product (if applicable).
    - **price:** Price of the product.
    - **discount:** Discount on the product, if any.

d. ORDERS Entity

- **Represents**: Orders placed by users on ShopEZ.
- **Attributes**:
    - **orderId:** Unique identifier for each order.
    - **userId:** References the user who placed the order.
    - **name:** Name of the customer placing the order.
    - **mobile:** Contact number of the customer.
    - **email:** Email address of the customer.
    - **address:** Shipping address for the order.
    - **pincode:** Area pincode for delivery.
    - **title:** Title of the ordered product.
    - **description:** Description of the ordered product.
    - **mainImg:** Image URL of the ordered product.
    - **size:** Size of the product ordered.
    - **quantity:** Quantity of the product ordered.
    - **price:** Price of the ordered product.
    - **discount:** Discount applied on the order.

- **paymentMethod:** Payment method chosen for the order.
- **orderDate:** Date the order was placed.
- **status:** Current status of the order (e.g., pending, shipped, delivered).

e. ADMIN Entity

- **Represents**: The administrative functions and configurations for the platform.
- **Attributes**:
  - **adminId:** Unique identifier for the admin.
  - **categories:** List of product categories managed by the admin.
  - **banner:** Banner images used on the ShopEZ platform for promotional purposes.

Summary of Relationships:

- **Users and Orders**: Each user can place multiple orders, with each order containing details on the products, payment, and shipping information.
- **Users and Cart**: Users can add products to their cart for potential purchase, with each cart linked to a unique user.
- **Admin and Products**: Admins manage the catalog of products available on ShopEZ, including adding or modifying listings and categories.

---

6. PROJECT STRUCTURE

The **ShopEZ** e-commerce application is structured into **frontend** and **backend** directories, following a modular approach for better scalability and maintainability. This structure organizes components by functionality, supports role-based features, and simplifies code management for both developers and administrators.

Frontend Structure

The frontend is built using **React.js** and is housed within the client folder. This directory contains all the files and components required for the user interface of ShopEZ.

Root Folders and Files

- **client**: The main folder containing all files and folders for the frontend.
- **node_modules**: Stores all the dependencies and modules installed via npm (Node Package Manager). This folder is generated automatically when dependencies are installed.
- **public**: Holds static files like index.html (the main HTML file for the React app), images, and other assets that are accessible directly by the browser. This folder is typically used for assets that don't change often.
- **src**: This is the main source folder for React components, application logic, and styles. All core components, pages, and configuration files are organized within this directory.

Folders and Files Inside src

1. **components**: Contains reusable React components. The components folder is divided into subfolders based on functionality or user roles, ensuring role-based feature encapsulation for better management.
    - **admin**:
        - **AdminHome.jsx:** The main dashboard or homepage component for admin users, displaying an overview of the admin functionalities.
        - **AllProducts.jsx:** Lists all products on the platform, allowing admins to view, edit, or delete listings.
    - **common**: Contains shared components accessible to all users regardless of their role.
        - **AllProducts.jsx:** A component that displays a list of all available products, which can be accessed by users, guests, and admins.
        - **AxiosInstance.jsx:** Configures Axios for making HTTP requests, likely setting up base URLs or authorization headers for consistent API calls.
        - **Dashboard.jsx:** A general dashboard component that can serve as a main page for logged-in users, displaying relevant information.
        - **Home.jsx:** The landing page component for the application, providing a welcoming interface with key highlights.
        - **Login.jsx:** The login component for user authentication, enabling users to securely access their accounts.
        - **NavBar.jsx:** A navigation bar that provides links to different parts of the application, adapting based on the user's role.
        - **Register.jsx:** The registration component for new users to sign up and create an account on the platform.
        - **UserHome.jsx:** The main dashboard for general users, providing access to their personalized shopping experience.

- **user**: Contains subfolders for components specific to each user role.
    - **customer**:
        - **ProductDetail.jsx:** Displays detailed information about a selected product, including images, descriptions, and purchase options.
        - **Cart.jsx:** Displays items in the user's shopping cart, allowing them to review and proceed to checkout.
        - **Profile.jsx:** Shows the user's profile details, including personal information and order history.
    - **seller**:
        - **AddProduct.jsx:** A form component that allows sellers to add new products to their inventory.
        - **SellerDashboard.jsx:** The main dashboard for sellers, providing an overview of sales metrics, product listings, and customer interactions.

2. **context**: Typically used for managing global state across components, such as user authentication status or shopping cart data.
3. **images**: Stores image assets used throughout the application, such as logos, icons, and product images.
4. **pages**: Contains page-level components for different routes in the application.
    - **ProductListing.jsx:** Lists all products available for purchase.
    - **Checkout.jsx:** Manages the checkout process, including payment and shipping details.
    - **OrderConfirmation.jsx:** Confirms the order after checkout and provides order details.
5. **styles**: Defines global CSS styles and custom styling for components, ensuring a cohesive design across the application.

Core Files in src

- **App.css**: Contains global styles for the entire frontend application, defining the look and feel of the interface.
- **App.jsx**: The root component of the application, which typically contains routes to various pages and loads other major components.
- **index.js**: The main entry point for the React application. It renders the React app into the main HTML document (index.html in the public folder).

Backend Structure

The backend is built using **Node.js** and **Express.js** to handle server-side logic, data management, and API requests. This part of the project is contained within the server folder.

Folder and File Structure

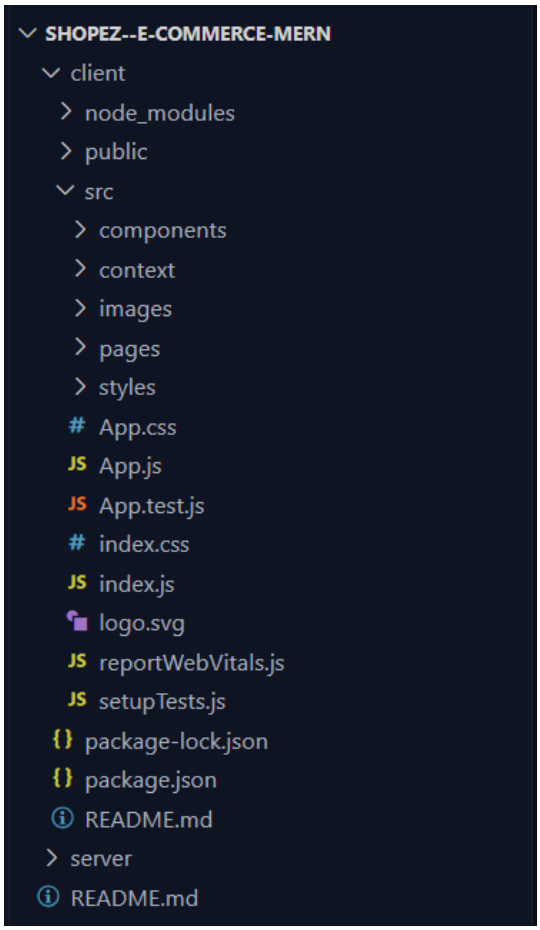
1. **config**: Holds configuration files necessary for the backend setup.
    - **.env:** Stores sensitive environment variables, such as database credentials, API keys, and other configuration settings that should not be hard-coded.
2. **controllers**: Contains server-side logic for handling requests and processing data.
    - **adminController.js**: Manages admin-specific actions, such as adding products, managing categories, and viewing order history.
    - **userController.js**: Handles user-related actions, such as user registration, login, browsing products, and placing orders.
3. **middlewares**: Contains middleware functions that are applied to specific routes for handling authentication and authorization.
    - **authMiddleware.js**: Verifies if users are authenticated and authorized to access certain routes. It ensures that only logged-in users can access restricted areas.
4. **routers**: Defines API routes for each user role and functionality, mapping each endpoint to specific controller functions.
    - **adminRoutes.js**: Defines API endpoints for admin functionalities, such as managing products and orders, and links them to the adminController.js functions.
    - **userRoutes.js**: Defines API endpoints for user functionalities, such as registration, login, product browsing, and cart management.
5. **schemas**: Contains MongoDB models (schemas) that define the structure of the collections in the database.
    - **userModel.js**: Defines the user schema, including fields like username, email, password, and userType (e.g., customer or admin).
    - **productModel.js**: Defines the product schema, including fields like title, description, price, category, and images.
    - **orderModel.js**: Defines the order schema, including fields such as userId, productId, quantity, price, paymentMethod, and status.
    - **cartModel.js**: Defines the schema for items added to the cart by users, linked to userId and product details.
6. **uploads**: Stores files uploaded by users, such as product images or profile pictures. This folder ensures uploaded files are organized and accessible to the server when required.

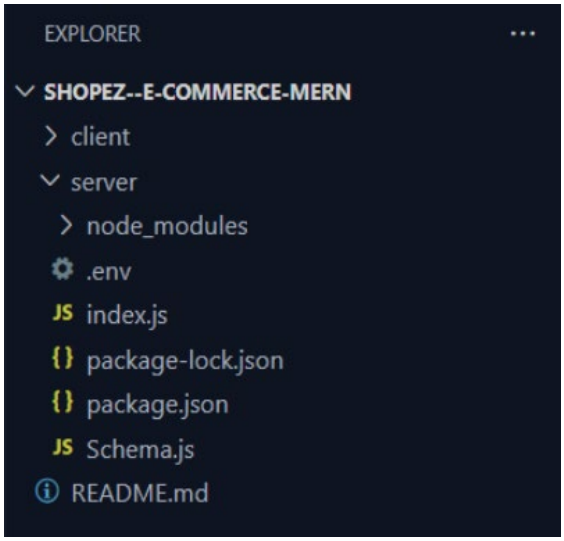
Environment and Configuration Files

- **.env**: Contains environment variables for the backend, such as database URIs and API keys. This file is essential for keeping sensitive data secure and separate from the codebase.
- **.gitignore**: Specifies files and directories that should be ignored by Git, such as node_modules, .env, and other files that don't need to be tracked in version control.

Entry Points and Package Files

- **index.js**: The main entry point for the backend server. It initializes the server, connects to the MongoDB database, sets up middleware, and loads routes for handling API requests.
- **package.json**: Contains metadata for the backend project, including dependencies, scripts, and project information. It manages the backend libraries and tools required to run the server.
- **package-lock.json**: Ensures consistency by locking the exact versions of dependencies installed in node_modules.





Supporting Files and Documentation

- **README.md**: Contains documentation for the project, including setup instructions, usage guidelines, and general information about the platform. This file is essential for helping developers and contributors understand and work with the codebase.

- **vite.config.js**: The configuration file for Vite, a fast build tool and development server for the frontend. This file allows customization of Vite's behavior, such as setting up plugins, defining alias paths, and configuring development and production environments.

7. APPLICATION FLOW

The **ShopEZ** application is designed to accommodate three primary user roles: **Customer**, **Seller (Restaurant)**, and **Admin**. Each role has specific responsibilities and permissions, managed through dedicated user flows and API endpoints. Below is an overview of the application flow for each role.

1. Customer Flow

- **Account Registration**:
    - Users begin by registering for an account on the **ShopEZ** platform.
    - During registration, they provide essential details such as email, password, and contact information.
- **Login**:
    - After registering, users can log in using their credentials (email and password).
    - Upon successful login, users are directed to the home page, where they can browse available products.
- **Product Browsing**:
    - Once logged in, users can view the full catalog of products available on the platform.
    - They can use filtering options, such as category, price, or brand, to narrow down their choices and find specific items.
- **Add to Cart**:
    - Users can add products to their cart. Each cart item includes details such as product title, price, quantity, and any applicable discounts.
    - Users can review and modify the items in their cart before proceeding to checkout.
- **Checkout Process**:
    - Users enter their shipping address and payment details during the checkout process.
    - They can choose from various payment methods, such as credit card, debit card, or digital wallets, for completing their purchase.
- **Order Confirmation & Tracking**:
    - After placing an order, users receive an order confirmation with details like order ID, estimated delivery time, and contact information for customer support.
    - Users can track their order status from their profile section, where they can view order history and order details.

2. Seller Flow (Restaurant)

- **Authentication**:
    - Sellers, such as restaurants, authenticate by logging in with their credentials.
    - New sellers may need to register and provide verification information before gaining access to the platform.
- **Admin Approval**:
    - After logging in, sellers need to obtain approval from the admin to start listing products.
    - Admins review seller accounts and grant them permission to add and manage product listings.
- **Product Management**:
    - Once approved, sellers can add, edit, or delete food items from their product catalog.
    - Each product listing includes details like title, description, price, and images. Sellers can also categorize items based on cuisine type or dietary preferences.
- **Order Management**:
    - Sellers can view incoming orders and prepare items for delivery or pickup.
    - They can update order statuses to reflect preparation, out-for-delivery, or completed stages, allowing customers to track progress in real-time.

3. Admin Flow

- **Login & Dashboard Access**:
    - Admins log in with their credentials and are directed to the Admin Dashboard upon successful authentication.
    - The dashboard provides an overview of platform activities, including user registrations, product listings, and recent orders.
- **User Management**:
    - Admins can view the list of all registered users, including customers and sellers.
    - They have the ability to approve or reject new seller accounts, monitor user activity, and manage user profiles as needed.
- **Product and Order Oversight**:
    - Admins can access the entire catalog of products listed on the platform, including those managed by sellers.
    - They have the authority to add, edit, or remove product listings, ensuring that all products meet platform standards.
    - Admins can also view order history, track order statuses, and resolve issues that may arise during order processing

- **Platform Management**:
    - Admins can monitor all activities within the application, including user engagement, order volumes, and sales data.
    - They ensure that platform operations run smoothly and address any technical or logistical issues that may arise.
    - Admins have the power to update platform settings, adjust fees, and oversee security protocols to protect user data and maintain the integrity of the application.

---

8. PROJECT SETUP & CONFIGURATION

To set up the **ShopEZ** e-commerce application, the project is organized into two main folders: **client** for the frontend and **server** for the backend. Each folder has specific dependencies and configurations necessary for development.

Folder Setup

- **client**: Contains all files related to the frontend of the application, built with React.
- **server**: Contains all files related to the backend, built with Node.js and Express.

8.1. FRONT-END DEVELOPMENT

The frontend is developed using **React** along with a variety of libraries to create an engaging, responsive, and user-friendly interface.

Required Dependencies

The following dependencies are essential for the frontend setup:

- **React**: Core library for building user interfaces.
- **Bootstrap** and **React-Bootstrap**: CSS frameworks for responsive design and styled components.
- **Material-UI** and **Ant Design (Antd)**: UI component libraries for enhanced styling and components.
- **Axios**: For making HTTP requests to communicate with the backend API.
- **MDB React UI Kit**: An additional UI library for more component options and styling.
- **React Router Dom**: For managing routing within the application.
- **React Icons**: Provides access to popular icon sets to enhance the UI.

Installation and Setup

These dependencies are installed in the **client** folder. Additionally, configuration files for development, such as package.json, will handle scripts for starting, building, and testing the frontend. Once installed, these libraries facilitate smooth development and help build a cohesive, interactive user experience.

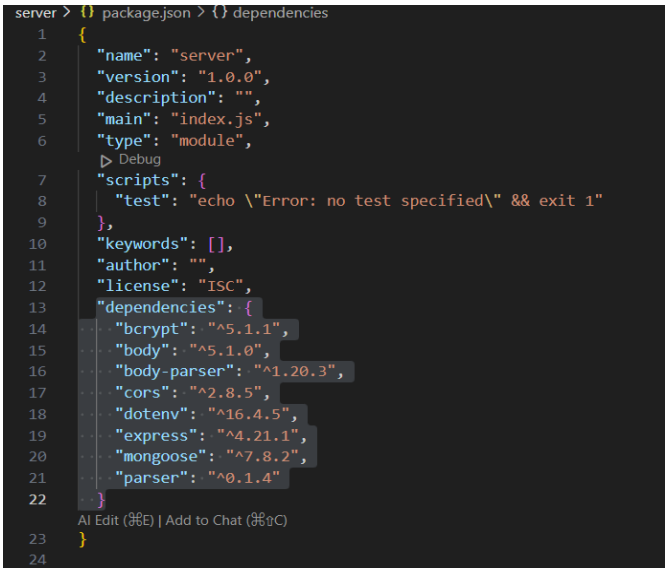
8.2. BACK-END DEVELOPMENT

The backend, built using **Node.js** and **Express.js**, handles server-side logic, authentication, and database interactions.

Required Dependencies

The backend requires the following packages:

- **Express**: Web framework for handling API routes and server requests.
- **Mongoose**: An ODM library for managing MongoDB data models.
- **JWT (jsonwebtoken)**: For implementing secure token-based authentication.
- **CORS**: Allows cross-origin requests between the frontend and backend.
- **bcrypt**: For hashing passwords, enhancing user data security.
- **body-parser**: Parses incoming request data, making it accessible in the server.
- **dotenv**: Loads environment variables from a .env file for secure configuration.
- **Multer**: Handles file uploads, such as product images.



Configuration

- **Environment Variables**: Sensitive information like the database connection string, port number, and JWT secret key are stored in a .env file. This keeps credentials and configuration separate from the codebase, ensuring better security.
- **Server Setup**: Middleware such as CORS and body-parser are configured to handle cross-origin requests and parse incoming data.
- **Authentication Middleware**: Authentication is managed by a custom middleware that verifies JWT tokens, protecting restricted routes and ensuring that only authorized users can access certain areas of the backend.

8.3. DATABASE DEVELOPMENT

MongoDB is used for data storage, with **Mongoose** as the ODM for schema management and interactions.

Database Configuration

- **Mongoose Integration**: MongoDB is connected to the backend through Mongoose, which simplifies schema creation, data validation, and database operations.
- **Model Management**: All database schemas, such as for users, products, and orders, are stored in a dedicated models folder. This organization ensures that each entity in the database has a well-defined structure, improving data integrity and simplifying CRUD operations.

---

9. PROJECT IMPLEMENTATION & EXECUTION

The **ShopEZ** application is implemented in two main sections: the **client** (frontend) and the **server** (backend). Each section is organized with specific folders and files, following a modular approach for efficient development, maintenance, and scalability.

9.1. CLIENT IMPLEMENTATION

Client Folder Structure

client

- **client > public**
  - client > public > index.html
- **client > src**
  - client > src > App.js
  - client > src > App.css
  - client > src > index.js
  - client > src > index.css
  - client > src > reportWebVitals.js
  - client > src > setupTests.js
  - client > src > App.test.js
  - **client > src > components**
    - client > src > components > FlashSale.jsx
    - client > src > components > Footer.jsx
    - client > src > components > Login.jsx
    - client > src > components > Navbar.jsx
    - client > src > components > Products.jsx
    - client > src > components > Register.jsx
  - **client > src > context**
    - client > src > context > GeneralContext.js
  - **client > src > pages**
    - **client > src > pages > admin**
      - client > src > pages > admin > Admin.jsx
      - client > src > pages > admin > AllOrders.jsx

        - client > src > pages > admin > AllProducts.jsx
        - client > src > pages > admin > AllUsers.jsx
        - client > src > pages > admin > NewProduct.jsx
        - client > src > pages > admin > UpdateProduct.jsx
    - **client > src > pages > customer**
        - client > src > pages > customer > Cart.jsx
        - client > src > pages > customer > CategoryProducts.jsx
        - client > src > pages > customer > IndividualProduct.jsx
        - client > src > pages > customer > Profile.jsx
        - client > src > pages > customer > Authentication.jsx
        - client > src > pages > customer > Home.jsx
- **client > src > styles**
    - client > src > styles > Admin.css
    - client > src > styles > AllOrders.css
    - client > src > styles > AllProducts.css
    - client > src > styles > AllUsers.css
    - client > src > styles > Authentication.css
    - client > src > styles > Cart.css
    - client > src > styles > CategoryProducts.css
    - client > src > styles > CheckOutPage.css
    - client > src > styles > FlashSale.css
    - client > src > styles > Footer.css
    - client > src > styles > Home.css
    - client > src > styles > IndividualProduct.css
    - client > src > styles > Navbar.css
    - client > src > styles > NewProducts.css
    - client > src > styles > Products.css
    - client > src > styles > Profile.css

- **client > package.json**
- **client > package-lock.json**
- **client > .gitignore**

9.2. SERVER IMPLEMENTATION

Server Folder Structure

server

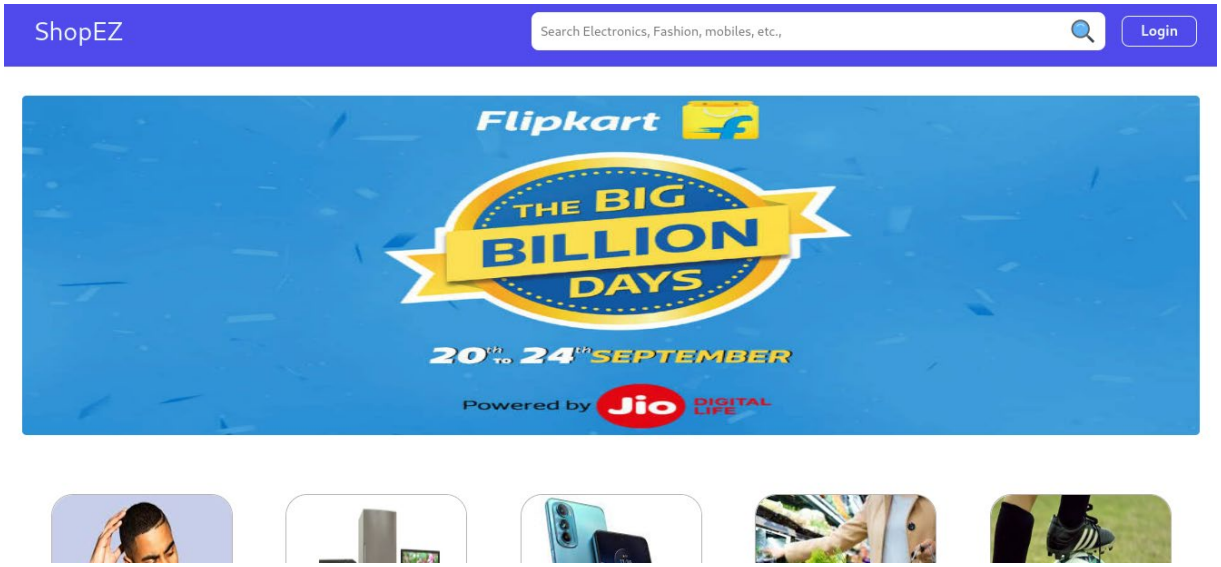
- **server > .env**
- **server > index.js**
- **server > Schema.js**
- **server > package.json**
- **server > package-lock.json**
- **server > .gitignore**

9.3. SCREENSHOTS

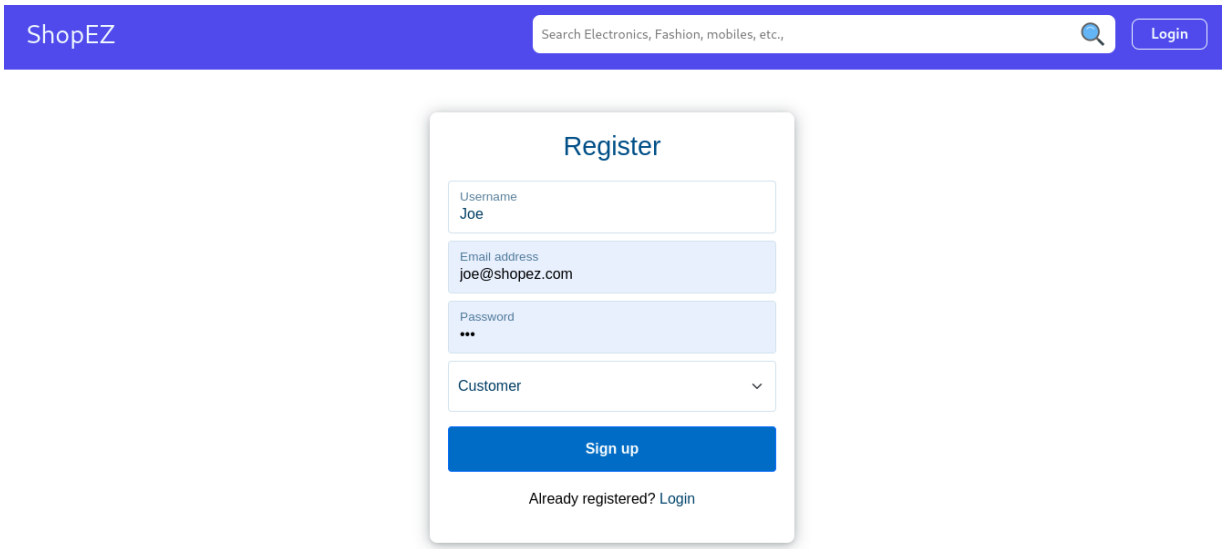
The screenshots provided below showcase key pages in the **ShopEZ** application, illustrating the user and admin flows. These screenshots cover critical aspects such as account registration, product browsing, cart management, and the admin dashboard.

User Flow Screenshots

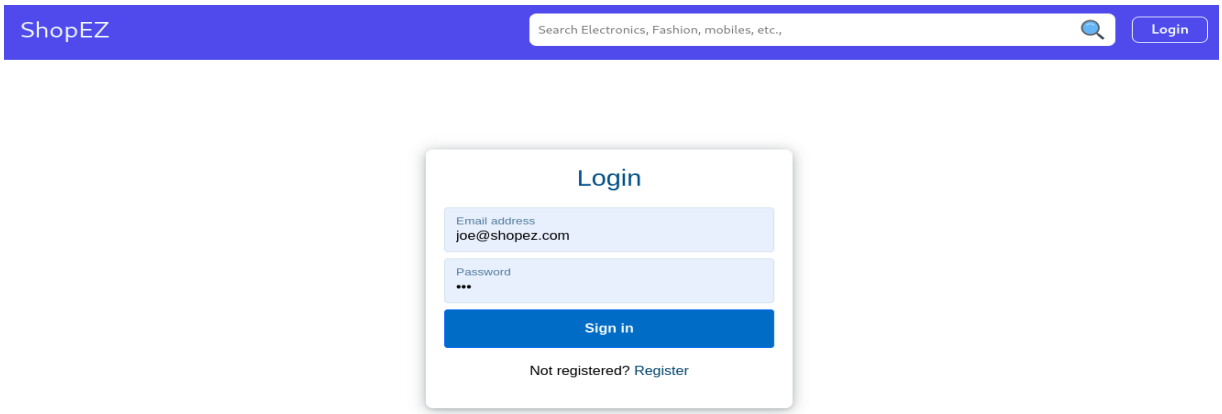
1. **Landing Page**
   The starting point for all users, showcasing the platform's offerings and featured products.



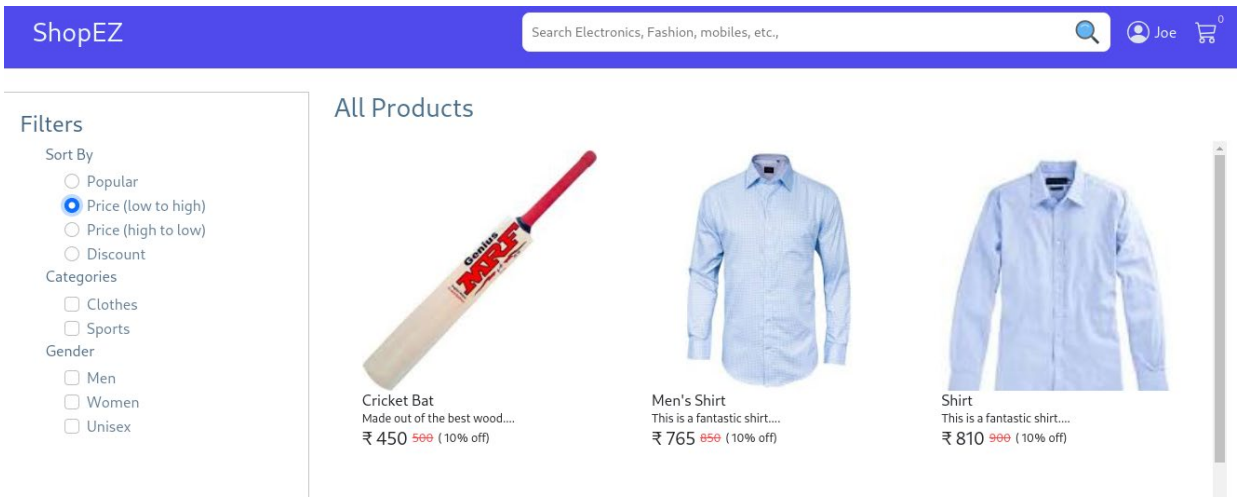
2. **Register Page**
   Users can create a new account by entering essential details such as name, email, and password.



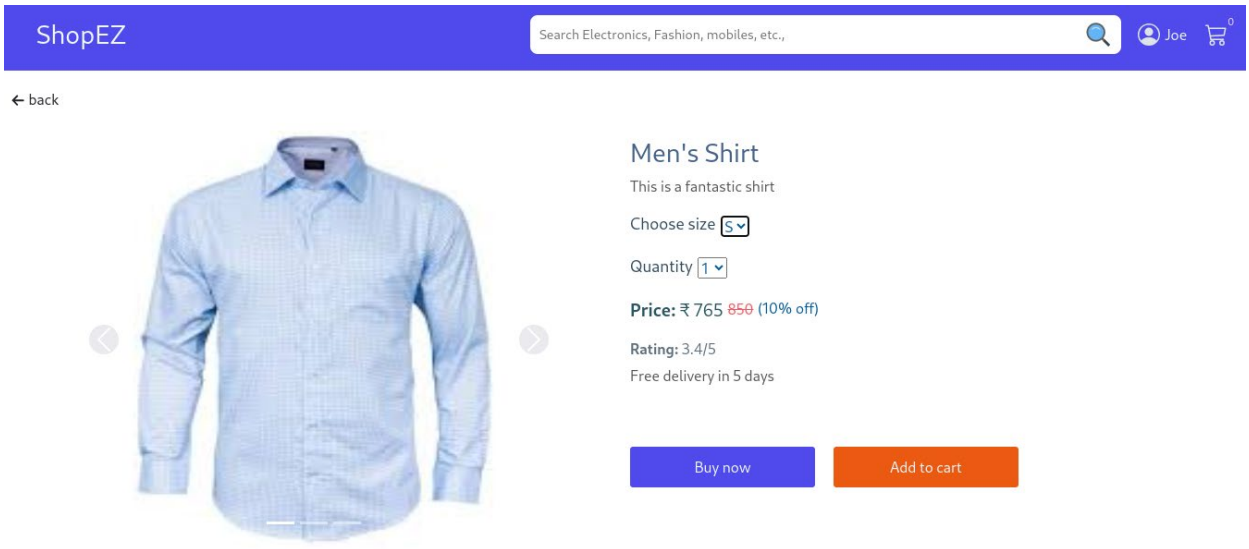
3. **Login Page**
   Existing users can log in using their credentials to access their personalized shopping experience.



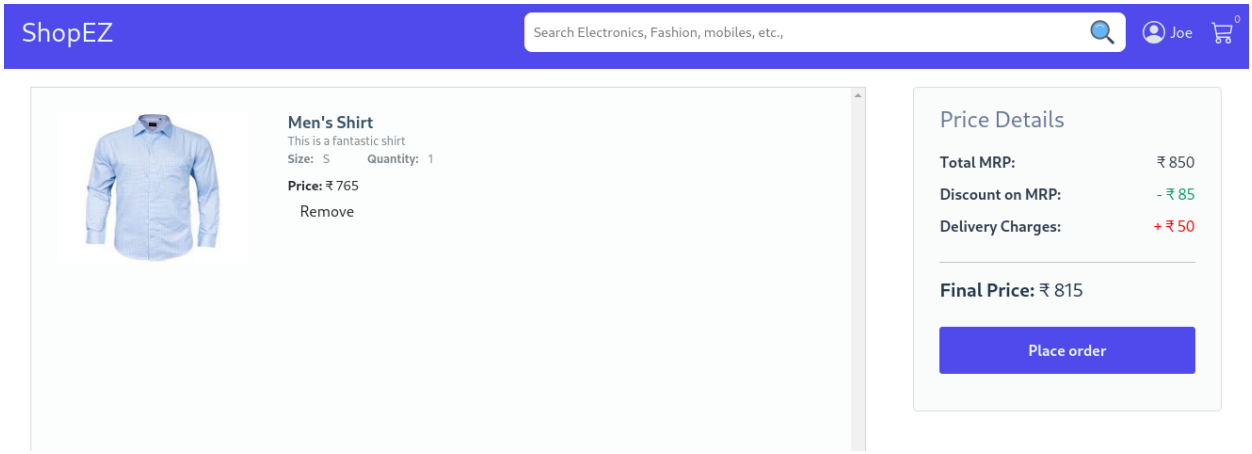
4. **Product Listing Page**
   Displays available products, with options to filter by category, price, and more, allowing users to browse the platform's offerings.



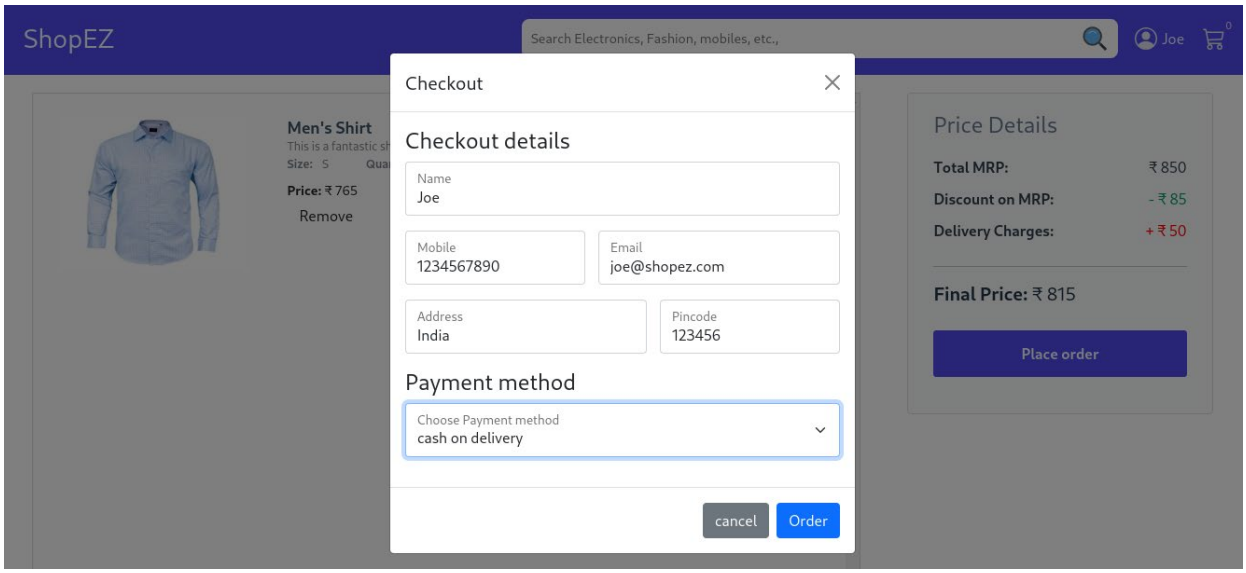
5. **Individual Product Page**
   Shows detailed information about a specific product, including description, price, and available sizes or colors.



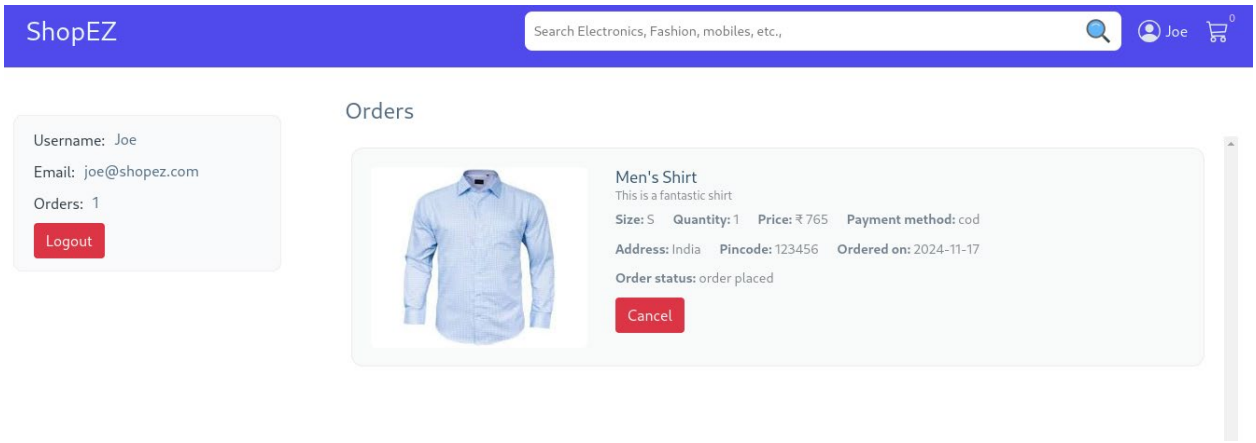
6. **Cart Page**
   Users can review the items in their cart, adjust quantities, and proceed to checkout.



7. **Checkout Page**
   Allows users to enter their shipping address and payment details to complete their purchase.

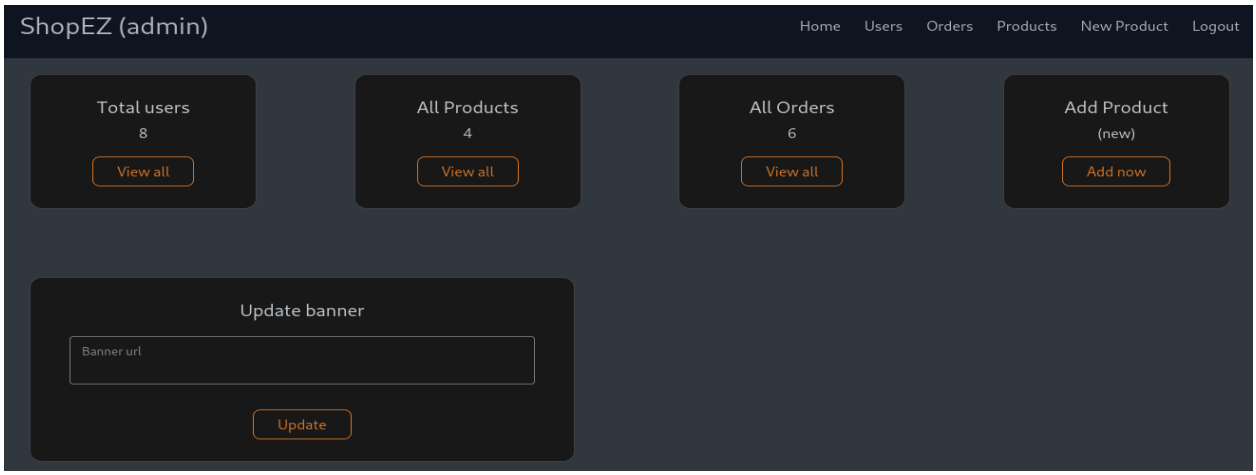


8. **User Profile Page**
   Provides access to user-specific information, including order history, account settings, and contact details.

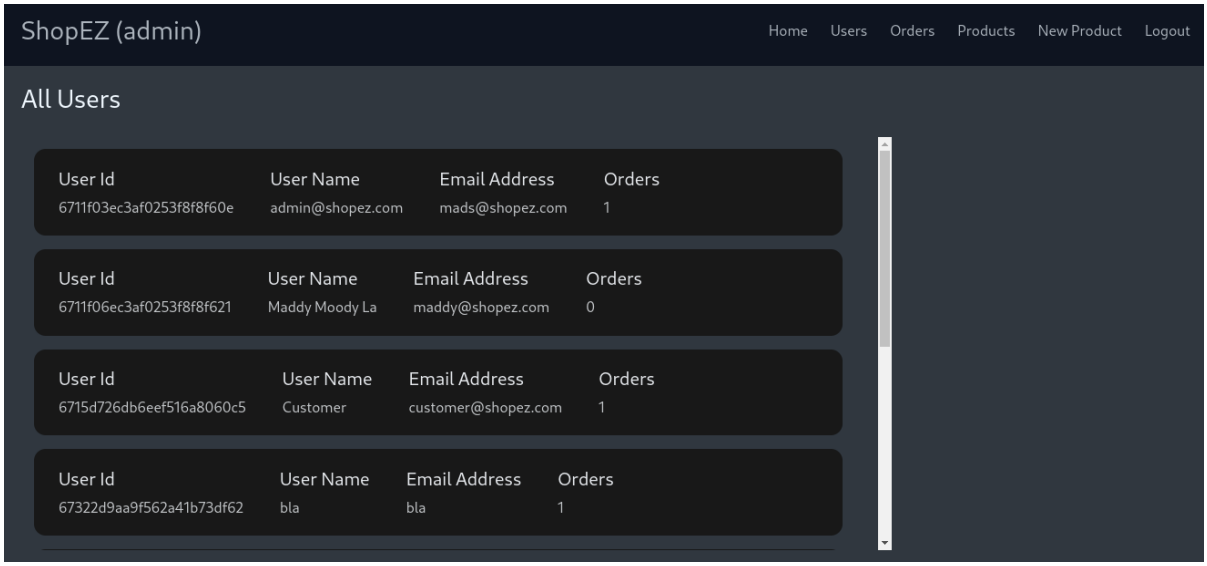


Admin Flow Screenshots

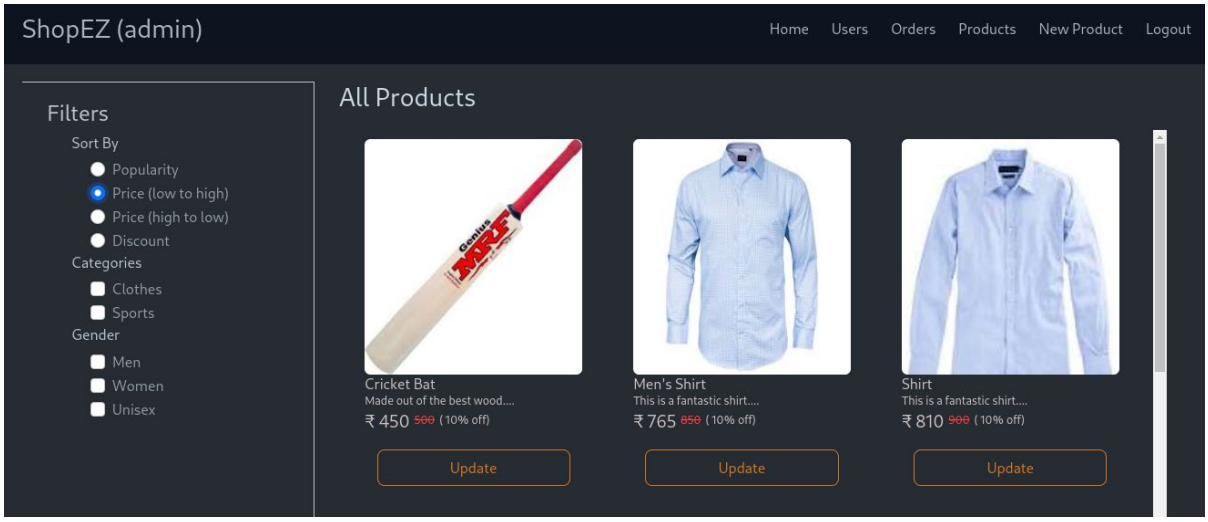
1. **Admin Dashboard**
   The main control panel for admins, offering an overview of platform activities and access to various management features.



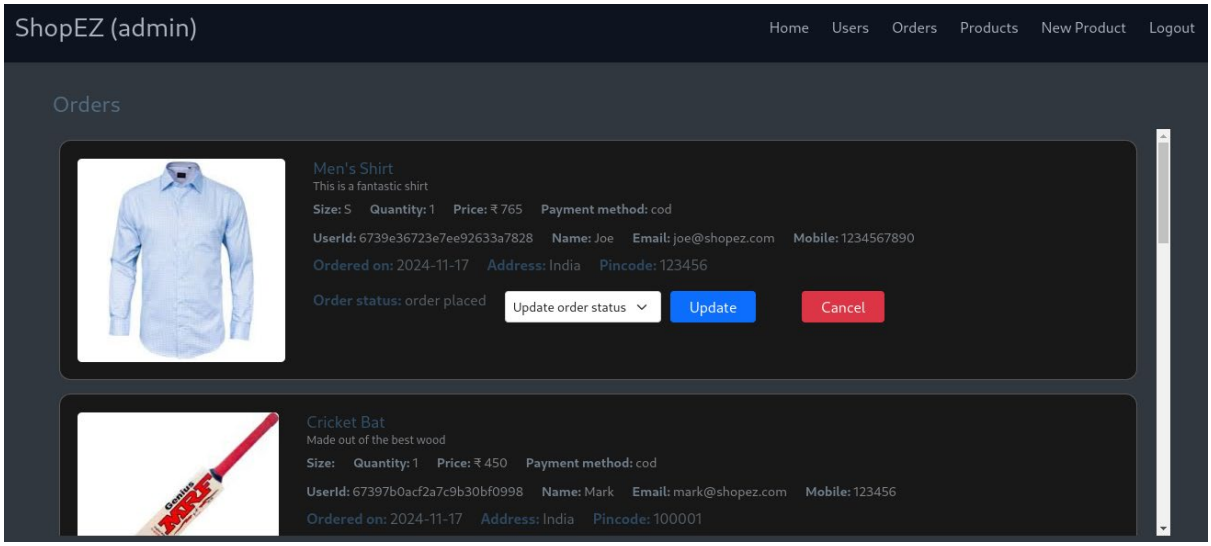
2. **All Users Page**
   Displays a list of all registered users, with options to view details and manage user accounts.



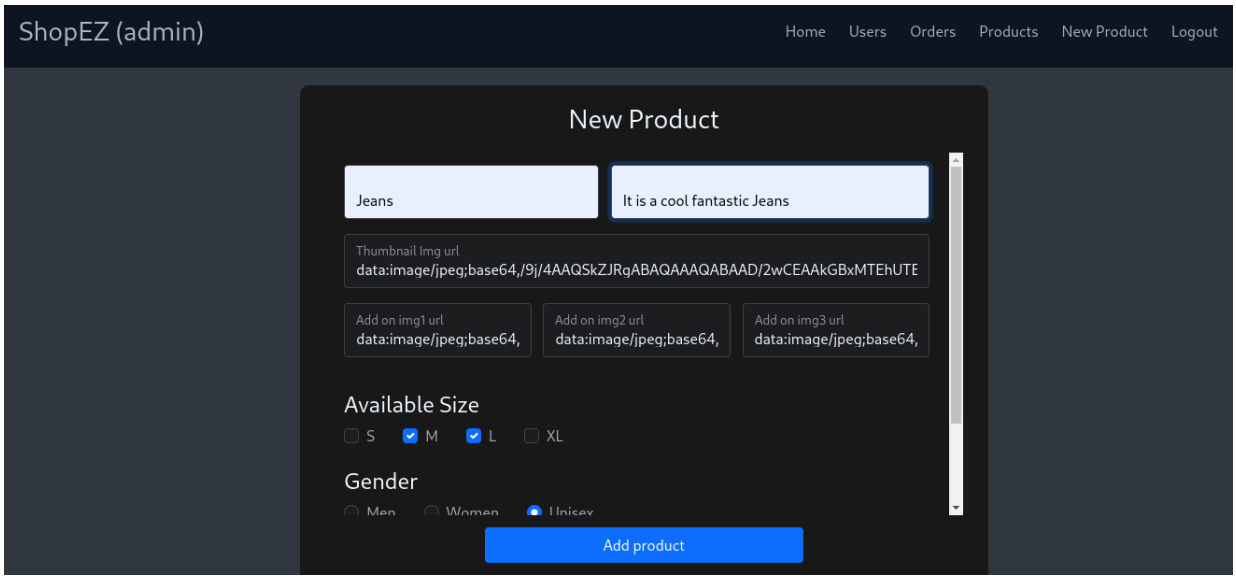
3. **All Products Page**
   Admins can view, add, edit, or delete products listed on the platform.



4. **All Orders Page**
   Shows all orders placed by users, including details on order status, customer information, and product lists.



5. **New Product Page**
   Allows admins to add new products to the platform, specifying details like title, description, price, and images.



10. CONCLUSION:

ShopEZ is a comprehensive e-commerce platform designed to provide a seamless and intuitive shopping experience for users, while also offering a robust management system for sellers and administrators. Built on the MERN stack (MongoDB, Express, React, Node.js), **ShopEZ** ensures scalability, security, and efficiency in handling a wide range of products, user interactions, and administrative tasks.

The platform successfully caters to the needs of three primary user roles: customers, sellers, and admins. Customers enjoy a streamlined shopping journey, from browsing products to completing secure checkouts. Sellers benefit from a dedicated dashboard that simplifies product management and order tracking, enabling them to scale their businesses effectively. Admins have a powerful set of tools to oversee platform activities, manage user accounts, and maintain product listings, ensuring a smooth and secure shopping environment.

ShopEZ demonstrates the potential of modern web technologies to create an engaging, scalable, and user-centric online marketplace. Through thoughtful design, modular development, and a focus on user experience, **ShopEZ** is positioned as a reliable and efficient solution for e-commerce, adaptable to future growth and enhancements.

In conclusion, **ShopEZ** not only meets the demands of today's e-commerce landscape but also sets a foundation for expansion, aiming to continue improving the shopping experience and management capabilities for users and businesses alike.