# Full Stack Development with Mern


## 1.Introduction

- **Project Title :** One-Stop Shop For Online Purchases

- **Team Members**

| | | |
|---|---|---|
| Team ID | : | LTVIP2025TMID44047 |
| Team Leader | : | Diddi Sri Akshitha |
| Team M ember | : | Veera Ragini |
| Team Member | : | Madasi Nihitha |
| Team Member | : | Shaik Sameer |

# 2.Project Overview

- **Purpose :**

The purpose of this project is to develop a comprehensive and user-friendly online platform that serves as a **one-stop shop** for purchasing a wide variety of products across multiple categories such as electronics, fashion, groceries, home essentials, and more. The main goal is to **streamline the shopping experience** by integrating multiple vendors and product types into a single, easy-to-navigate website or mobile app.

**Key Goals:**

- Provide a **centralized platform** for all types of online shopping needs.
- Ensure **convenient and secure payment options**.
- Offer **personalized product recommendations** using customer preferences and purchase history.
- Enable **efficient search, filtering, and comparison** tools.
- Maintain **real-time inventory and order tracking**.
- Build a **loyal customer base** through rewards, discounts, and responsive customer service.

This platform aims to save time and effort for users by eliminating the need to visit multiple websites, making online shopping **faster, easier, and more reliable**.

- **Features :**

1. **Multi-Category Product Listings**
    - Wide range of products including electronics, clothing, groceries, home goods, and more.
2. **Advanced Search and Filters**
    - Keyword search, category filters, price range, brand, ratings, and availability options.
3. **Personalized Recommendations**
    - AI-based suggestions based on browsing history, purchase behavior, and preferences
4. **Secure User Accounts**
    - User registration, profile management, order history, and wishlist functionality.
5. **Multiple Payment Options**
    - Credit/debit cards, UPI, net banking, mobile wallets, and cash on delivery.

# 3.Architecture

- **Frontend :**

1. **Component-Based Structure**:
   The UI is built using reusable components like Header, ProductList, Cart, and Checkout.
2. **Routing**:
   React Router handles navigation between pages like Home, Products, Cart, and Orders.
3. **State Management**:
   Uses **React Context API** or **Redux** to manage global states like cart items and user login status.
4. **API Integration**:
   Uses Axios or Fetch to communicate with backend services (e.g., to get product data or place orders).
5. **UI Library**:
   Can use **Tailwind CSS**, **Bootstrap**, or **Material-UI** for styling and responsive design.
6. **Authentication**:
   Login and signup handled with form components and token storage (e.g., in localStorage).
7. **Performance Optimization**:
   Code-splitting and lazy loading are used to load components only when needed.
   This makes the frontend fast, user-friendly, and easy to maintain.

- **Backend :**

1. **Server Setup**:
   Built with **Node.js** and **Express.js** to handle HTTP requests and responses.
2. **Routing**:
   Uses Express routers to manage endpoints like /products, /users, /orders, and /cart.
3. **Database**:
   Connects to a database like **MongoDB** (using Mongoose) or **MySQL** to store user, product, and order data.
4. **Authentication**:
   Uses **JWT (JSON Web Tokens)** for user login, signup, and secure route access.
5. **API Layer**:
   RESTful APIs are created for frontend to perform actions like fetching products, adding to cart, and placing orders.
6. **Middleware**:
   Includes middleware for error handling, request validation, logging, and security (like CORS, Helmet).
7. **File Uploads**:
   Supports image or document uploads using multer if needed (e.g., product images).
8. **Environment Management**:
   Configuration managed using .env files (e.g., port number, database URL, secret keys).

This backend setup supports a scalable and secure e-commerce platform.

- **Database :**

### 📦 1. Users Collection

Stores customer info.

{

name: String,

email: String,

password: String,

address: String,

isAdmin: Boolean

}

- **Use:** Signup, login, update profile.

### 🛍️ 2. Products Collection

Stores product details.

{

name: String,

description: String,

price: Number,

stock: Number,

category: String,

imageUrl: String

}

- **Use:** Show product list, search, manage stock.

### 📦 3. Orders Collection

Stores customer orders.

{

userId: ObjectId,

products: [{ productId: ObjectId, quantity: Number }],

totalAmount: Number,

status: String

}

- **Use:** Place order, track order, update status.

### 🛒 4. Cart Collection *(optional)*

Stores items added to cart.

{

userId: ObjectId,

items: [{ productId: ObjectId, quantity: Number }]

}

- **Use:** Add, remove, or update cart items.

### 🔁 MongoDB Interactions

- **Add Data:** Model.create(data)
- **Read Data:** Model.find(), Model.findById(id)
- **Update Data:** Model.updateOne()
- **Delete Data:** Model.deleteOne()

**This schema helps manage users, products, orders, and carts in a simple online shopping system.**

## 4.Setup Instructions

- **Prerequisites:**

1. **Node.js** – Runtime for running JavaScript on the server
2. **Express.js** – Web framework for Node.js

3. **MongoDB** – NoSQL database for storing data
4. **Mongoose** – ODM for MongoDB to interact easily
5. **React.js** – Frontend JavaScript library
6. **Axios** – For making HTTP requests from frontend
7. **JWT (jsonwebtoken)** – For authentication
8. **bcryptjs** – For password hashing
9. **Cors** – To handle cross-origin requests
10. **Dotenv** – To manage environment variables
11. **Nodemon** – For automatic server restarts during development

These tools are needed to build and run the one-stop shop application.

- **Installation :**

  - **1. Clone the Project**
    git clone https://github.com/your-username/your-repo.git
    cd your-repo
  - **2. Install Backend Dependencies**
    cd backend
    npm install
  - **3. Install Frontend Dependencies**
    cd ../frontend
    npm install
  - **4. Set Up Environment Variables**

  ✅ **In backend/.env file:**

  PORT=5000
  MONGO_URI=your_mongodb_connection_string
  JWT_SECRET=your_jwt_secret_key

  ✅ **In frontend/.env file:**

  REACT_APP_API_URL=http://localhost:5000

  - **5. Run the Backend Server**
    cd backend
    npm start
  - **6. Run the Frontend App**
    cd ../frontend
    npm start

  Now your app should be running locally! 🎉

## 5.Folder Structure

- **Client :**

  🗀 **1. src/ – Main source folder**

  Contains all the frontend code.

**2. components/**

Reusable UI parts

- Header.js
- ProductCard.js
- CartItem.js
- Footer.js

**3. pages/**

Main pages of the app

- Home.js – Product listings
- ProductDetails.js – Single product view
- Cart.js – Shopping cart
- Login.js / Register.js
- Checkout.js
- Orders.js

**4. context/ or store/**

Handles global state (e.g., cart, user)

**5. services/**

Handles API calls using Axios

- productService.js
- userService.js

**6. App.js**

Main component with routing setup

**7. index.js**

Entry point – renders the app

This structure keeps the code clean, modular, and easy to manage.

- **Server :**

**1. server/ or backend/**

Main backend folder

**2. routes/**

Defines API routes

- userRoutes.js
- productRoutes.js
- orderRoutes.js
- cartRoutes.js

**3. controllers/**

Handles logic for each route

- userController.js
- productController.js

**4. models/**

Defines MongoDB schemas using Mongoose

- User.js
- Product.js
- Order.js

**5. middleware/**

Handles auth and error checking

- authMiddleware.js
- errorHandler.js

**6. config/**

Database connection and environment config

- db.js

**7. server.js**

Entry point – sets up Express app and runs the server.

This structure makes the backend organized, modular, and easy to maintain.

## 6.Running the Application

- **Frontend :**

1. Open terminal
2. Go to the frontend/client folder:
   `cd frontend`
3. Start the React app:
   `npm start`

✅ The app will open in your browser at http://localhost:3000 by default.

- **Backend :**

1. Open terminal
2. Go to the backend/server folder:
   `cd backend`
3. Start the server:
   `npm start`

✅ The server will run on http://localhost:5000 (or the port set in .env).

## 7.API Documentation

- **Document all endpoints exposed by backend**

**User Endpoints**

- POST /api/users/register – Register a new user
- POST /api/users/login – Login user
- GET /api/users/profile – Get user profile (auth required)

🔹 **Product Endpoints**

- GET /api/products – Get all products
- GET /api/products/:id – Get product by ID
- POST /api/products – Add new product (admin)
- PUT /api/products/:id – Update product (admin)
- DELETE /api/products/:id – Delete product (admin)

🔸 **Cart Endpoints**

- GET /api/cart – Get user cart
- POST /api/cart – Add item to cart
- PUT /api/cart – Update cart item quantity
- DELETE /api/cart/:id – Remove item from cart

🔸 **Order Endpoints**

- POST /api/orders – Place a new order
- GET /api/orders – Get user's orders
- GET /api/orders/:id – Get order by ID
- PUT /api/orders/:id – Update order status (admin)

These endpoints allow full user, product, cart, and order management.

- **Include request methods,parametera and examples**

👤 **User Endpoints**

🔹 **POST /api/users/register**

**Creates a new user**

**Body:**

```
{ "name": "Alice", "email": "alice@example.com", "password": "123456" }
```

**Response:**

```
{ "message": "User registered", "token": "abc123" }
```

🔹 **POST /api/users/login**

**User login**

**Body:**

```
{ "email": "alice@example.com", "password": "123456" }
```

**Response:**

```
{ "message": "Login successful", "token": "abc123" }
```

🔹 **GET /api/users/profile *( 🔐 Auth required)***

**Get logged-in user info**

**Headers:** Authorization: Bearer token

**Response:**

```
{ "name": "Alice", "email": "alice@example.com" }
```

🛍️ **Product Endpoints**

🔹 **GET /api/products**

**List all products**

**Response:**

```
[
 { "name": "Laptop", "price": 50000 },
```

{ "name": "Phone", "price": 20000 }
]

🔹 **GET /api/products/:id**
**Get one product by ID**
**Response:**
{ "name": "Laptop", "price": 50000 }

🔹 **POST /api/products ( 🔐 *Admin only)***
**Add a product**
**Body:**
{ "name": "Tablet", "price": 15000, "stock": 20 }
**Response:**
{ "message": "Product added" }

🔹 **PUT /api/products/:id ( 🔐 *Admin only)***
**Update a product**
**Body:**
{ "price": 14000 }
**Response:**
{ "message": "Product updated" }

🔹 **DELETE /api/products/:id ( 🔐 *Admin only)***
**Delete a product**
**Response:**
{ "message": "Product deleted" }

🛒 **Cart Endpoints**

🔹 **GET /api/cart ( 🔐 *Auth required)***
**Get user's cart**
**Response:**
{ "items": [{ "productId": "123", "quantity": 2 }] }

🔹 **POST /api/cart**
**Add item to cart**
**Body:**
{ "productId": "123", "quantity": 1 }
**Response:**
{ "message": "Item added to cart" }

🔹 **PUT /api/cart**
**Update item quantity**
**Body:**
{ "productId": "123", "quantity": 3 }
**Response:**
{ "message": "Cart updated" }

🔹 **DELETE /api/cart/:id**
**Remove item from cart**
**Response:**
{ "message": "Item removed from cart" }

📦 **Order Endpoints**

🔹 **POST /api/orders**

**Place a new order**
**Body:**
```
{
 "items": [{ "productId": "123", "quantity": 2 }],
 "total": 1000
}
```
**Response:**
```
{ "message": "Order placed", "orderId": "ord001" }
```
🔹 **GET /api/orders *( 🔐 Auth required)***
**Get user orders Response:**
```
[
 { "orderId": "ord001", "total": 1000, "status": "Pending" }
]
```
🔹 **GET /api/orders/:id**
**Get order by IDResponse:**
```
{ "orderId": "ord001", "status": "Shipped" }
```
🔹 **PUT /api/orders/:id *( 🔐 Admin only)***
**Update order status**
**Body:**
```
{ "status": "Delivered" }
```
**Response:**
```
{ "message": "Order status updated" }
```
✅ These endpoints allow the frontend to perform all operations: register, login, browse products, manage cart, and place/view orders.

## 8.Authentication

- **Explain how authentication and authorization are handled in the project.**

🔐 **Authentication (Login & Verify User)**

1. **User logs in** with email & password.
2. Server **verifies credentials** and creates a **JWT (JSON Web Token)**.
3. The token is sent to the frontend and stored in **localStorage** or **cookies**.

✅ **Authorization (Protect Routes)**

1. For protected routes (like /api/orders, /api/users/profile), the token is sent in the **Authorization header**:
2. Authorization: Bearer <token>
3. Backend middleware **verifies the token**.
4. If valid, it allows access; if not, it blocks the request.

🛡️ **Admin Check**

Admin-only routes (e.g., adding/deleting products) check:

if (user.isAdmin) { allow } else { deny }

**Tools Used:**

- jsonwebtoken – to create/verify tokens
- bcryptjs – to hash passwords
- Middleware – to protect routes

This ensures secure login and restricted access to sensitive features.

- **Include details about tokens,sessions or any other methods used**

🔐 **1. Tokens (JWT - JSON Web Token)**

- After login or registration, a **JWT token** is created.
- The token includes user ID and role (admin/user).
- It is sent to the client and stored in **localStorage** or **cookies**.

📜 **2. Using the Token**

- For protected routes, the client sends the token in the **Authorization header**:
- Authorization: Bearer <token>
- The backend uses jsonwebtoken to **verify** the token.

🛡️ **3. Middleware for Protection**

- A custom middleware checks if the token is valid.
- If valid → user is allowed access.
- If invalid → request is denied.

🔑 **4. Password Security**

- Passwords are hashed using bcryptjs before storing in the database.

🚫 **5. Sessions**

- **Not used.**

  Stateless JWT tokens are used instead of server-side sessions.

✅ **Summary**

- **Authentication** = Login with email/password → receive JWT
- **Authorization** = Use token to access secure routes
- No session storage — everything is handled with **secure tokens**.

## 9.User Interface

- **GIFs showcasing different UI features**

🎞️ **Suggested GIFs (Optional)**

- **Product being added to cart**
- **Login and redirect to dashboard**
- **Placing an order**
- **Admin editing a product**

## 10.Testing

- **Describe the testing strategy and tools used**

  ✅ **1. Testing Strategy**

  - **Unit Testing**: Test individual functions (e.g., product price calculation).
  - **API Testing**: Test backend routes like /api/products, /api/users/login.
  - **Integration Testing**: Test combined components (e.g., placing an order).
  - **UI Testing**: Check if buttons, forms, and navigation work correctly.

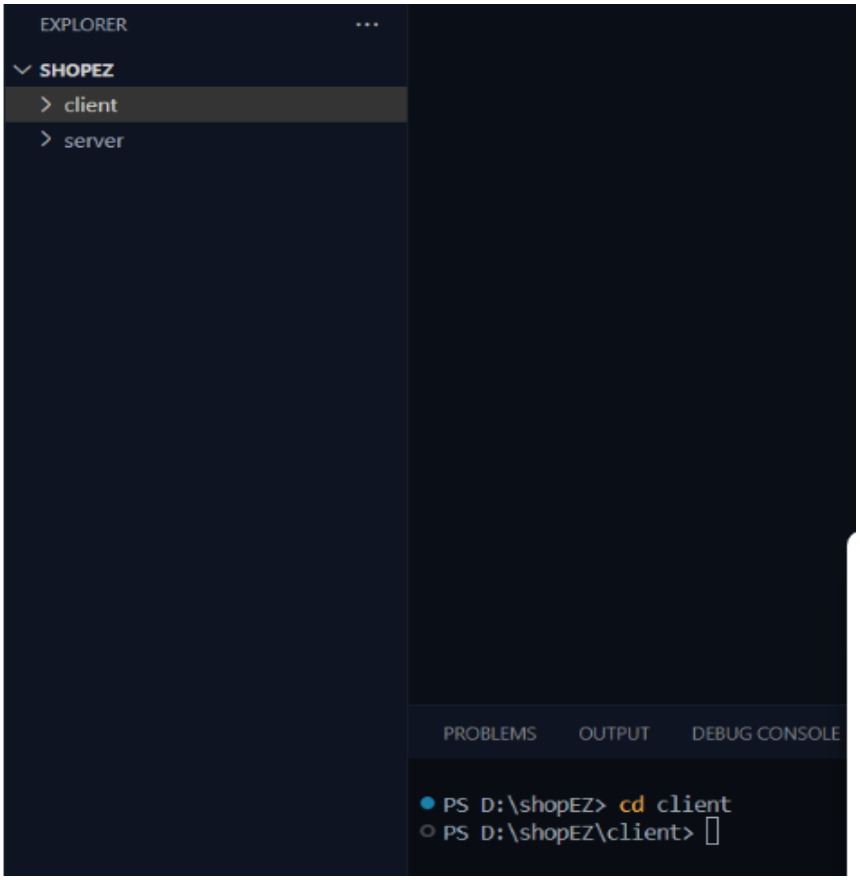
  🛠️ **2. Tools Used**

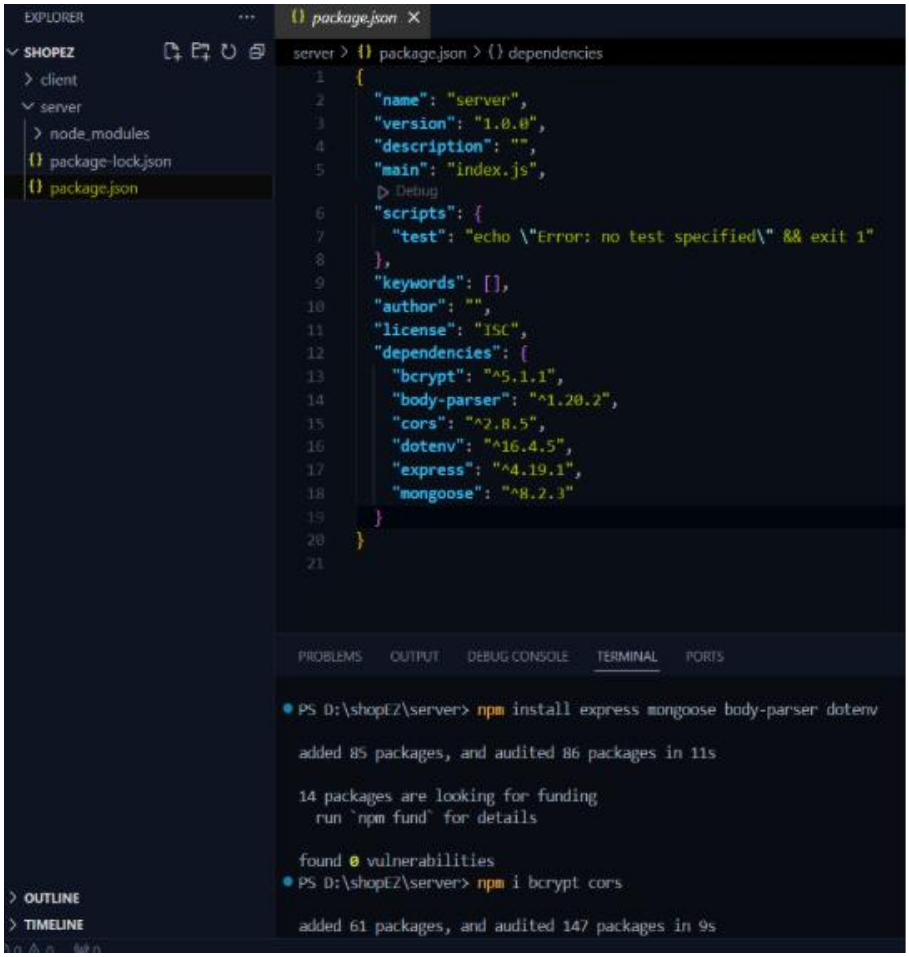
  - **Jest** – For JavaScript unit testing
  - **Supertest** – For testing Express.js API routes
  - **React Testing Library** – For testing React components
  - **Postman** – For manual API testing
  - **Cypress** *(optional)* – For end-to-end UI testing

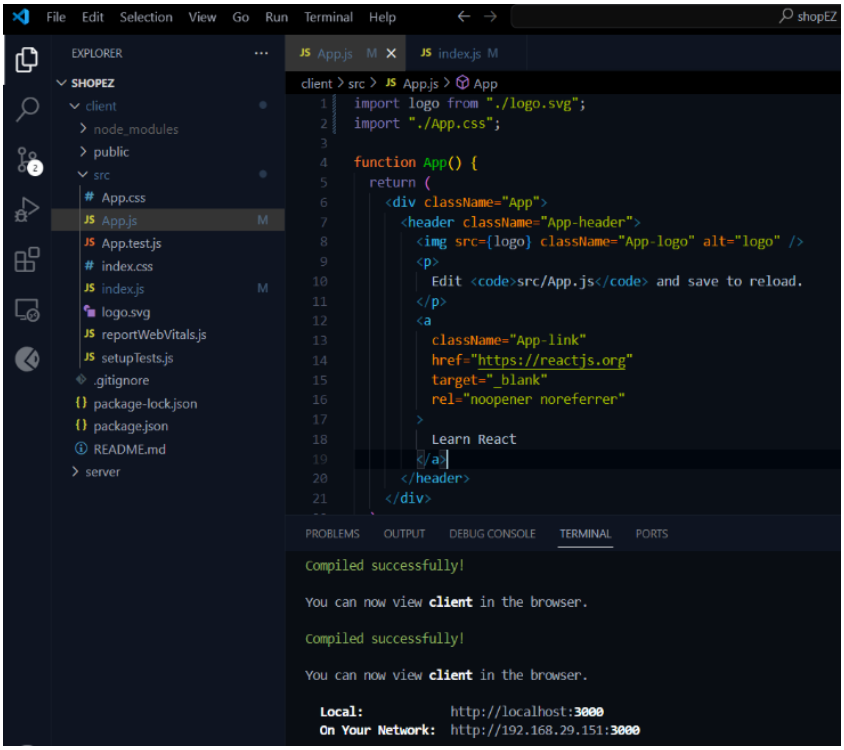
  This approach ensures that the app works correctly at all levels — backend, frontend, and overall user flow.

## 11.Screenshots or demo

- **Provide screenshots :**




EXPLORER
SHOPEZ
client
server
node_modules
package-lock.json
package.json
package.json
server > package.json > dependencies
{
"name": "server",
"version": "1.0.0",
"description": "",
"main": "index.js",
Debug
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
"bcrypt": "^5.1.1",
"body-parser": "^1.20.2",
"cors": "^2.8.5",
"dotenv": "^16.4.5",
"express": "^4.19.1",
"mongoose": "^8.2.3"
}
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS D:\shopEZ\server> npm install express mongoose body-parser dotenv
added 85 packages, and audited 86 packages in 11s
14 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
PS D:\shopEZ\server> npm i bcrypt cors
added 61 packages, and audited 147 packages in 9s
OUTLINE
TIMELINE


File Edit Selection View Go Run Terminal Help
shopEZ
EXPLORER
SHOPEZ
client
node_modules
public
src
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
server
App.js M
index.js M
client > src > App.js > App
import logo from "./logo.svg";
import "./App.css";
function App() {
return (

<header className="App-header">
<img src={logo} className="App-logo" alt="logo" />

Edit <code>src/App.js</code> and save to reload.

<a
className="App-link"
href="https://reactjs.org"
target="_blank"
rel="noopener noreferrer"
>
Learn React
</a>
</header>

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Compiled successfully!
You can now view client in the browser.
Compiled successfully!
You can now view client in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.29.151:3000

## 12.Known Issues

- **Document any known bugs :**

1. ⚠️ **Cart not updating in real-time**
    - Cart item count may not refresh until page reload.
2. ⚠️ **Product image upload (Admin)**
    - Image preview may fail if file size is too large.
3. ⚠️ **Token expiration not handled**
    - User may stay logged in even after token expires.
4. ⚠️ **Slow response on large product lists**
    - No pagination or lazy loading implemented yet.
5. ⚠️ **Mobile responsiveness issues**
    - Some UI elements may break on very small screens.

✅ These issues can be fixed in future updates. Developers should test carefully and log any new bugs.

## 13.Future Enhancements

- **Outline potential future features :**

1. 🌐 **Search & Filter Options**
    - Add product filtering by price, category, and rating.
2. 🛡️ **Token Expiry Handling**
    - Auto logout or refresh token on expiry.
3. 📱 **Full Mobile Responsiveness**
    - Optimize UI for all device sizes.
4. 📦 **Order Tracking System**
    - Show delivery status updates to users.
5. 🌟 **Product Ratings & Reviews**
    - Let users rate and review products.
6. 🧾 **Invoice Download**
    - Generate and download PDF invoices.
7. 🛍️ **Wishlist Feature**
    - Save products for later purchase.
8. 📊 **Admin Dashboard with Analytics**
    - Show sales stats, best-selling products, and user activity.

9. 🔔 **Email Notifications**
    - Send emails for order confirmations and status updates.
10. 💳 **Online Payment Integration**

- Add gateways like Razorpay, Stripe, or PayPal.

These features can enhance user experience and make the app more powerful and complete.