# ShopEZ:
# E-commerce Application

Full Stack Development with MERN

**Team Lead:Nihalika Kumari**
**Team ID: SWTID1743607402**

# 1. Introduction

**Project Title:** ShopEZ: E-commerce Application

ShopEZ is your one-stop destination for effortless online shopping. With a user-friendly interface and a comprehensive product catalog, finding the perfect items has never been easier. Seamlessly navigate through detailed product descriptions, customer reviews, and available discounts to make informed decisions. Enjoy a secure checkout process and receive instant order confirmation. For sellers, our robust dashboard provides efficient order management and insightful analytics to drive business growth. Experience the future of online shopping with ShopEZ today.

- Seamless Checkout Process
- Effortless Product Discovery
- Personalized Shopping Experience
- Efficient Order Management for Sellers
- Insightful Analytics for Business Growth

**Team Members:**

1. **Nihalika Kumari(Team Lead)**
2. **Rishav Raj**
3. **Asmita Sakhare**
4. **Vedika Vivek Gangil**

# 2. Project Overview

### 2.1 Purpose

ShopEZ aims to revolutionize the online shopping experience by offering a user-friendly, intuitive platform that caters to both buyers and sellers. For customers, ShopEZ provides a personalized, efficient, and secure way to discover and purchase products, while for sellers, it offers robust tools for order management and data-driven business growth. The goal is to simplify online shopping, making it faster, more enjoyable, and stress-free.

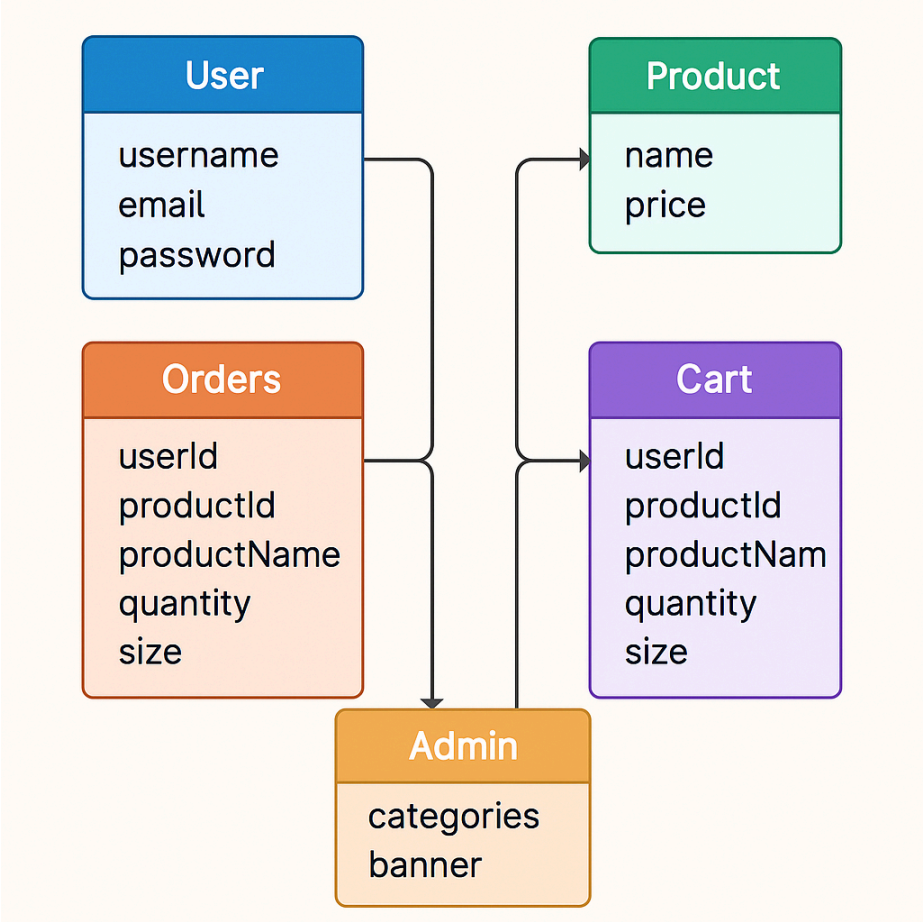
### 2.2 Features

1. **Effortless Product Discovery :**Advanced filtering and search options to help users find products quickly based on category, style, budget, and more.

2. **Personalized Shopping Experience :**Smart recommendations based on browsing history, user preferences, and previous purchases.
3. **Seamless Checkout Process:** Quick and secure checkout with multiple payment options and instant order confirmation.
4. **Efficient Order Management for Sellers:** Seller dashboard for real-time order tracking, fulfillment updates, and customer notifications.
5. **Insightful Analytics for Business Growth:** Visual reports and performance metrics to help sellers understand trends, customer behavior, and optimize their offerings.
6. **User & Admin Authentication:** Secure login/signup for users and role-based access for admin operations.
7. **Integrated Backend & Database:** APIs for Users, Products, Orders, and Admin actions, backed by a structured database storing all necessary collections.

## 3. Architecture

### Database Architecture



1. User Collection (User)
   - Fields:
     - username
     - email
     - password
   - Purpose: Stores user registration and login information.
   - Relationships:
     - Referenced by both Orders and Cart schemas through userId.

2. Product Collection (Product)
   - Fields:
     - name
     - price
   - Purpose: Stores information about each product available on the platform.
   - Relationships:
     - Referenced by both Orders and Cart schemas through productId.

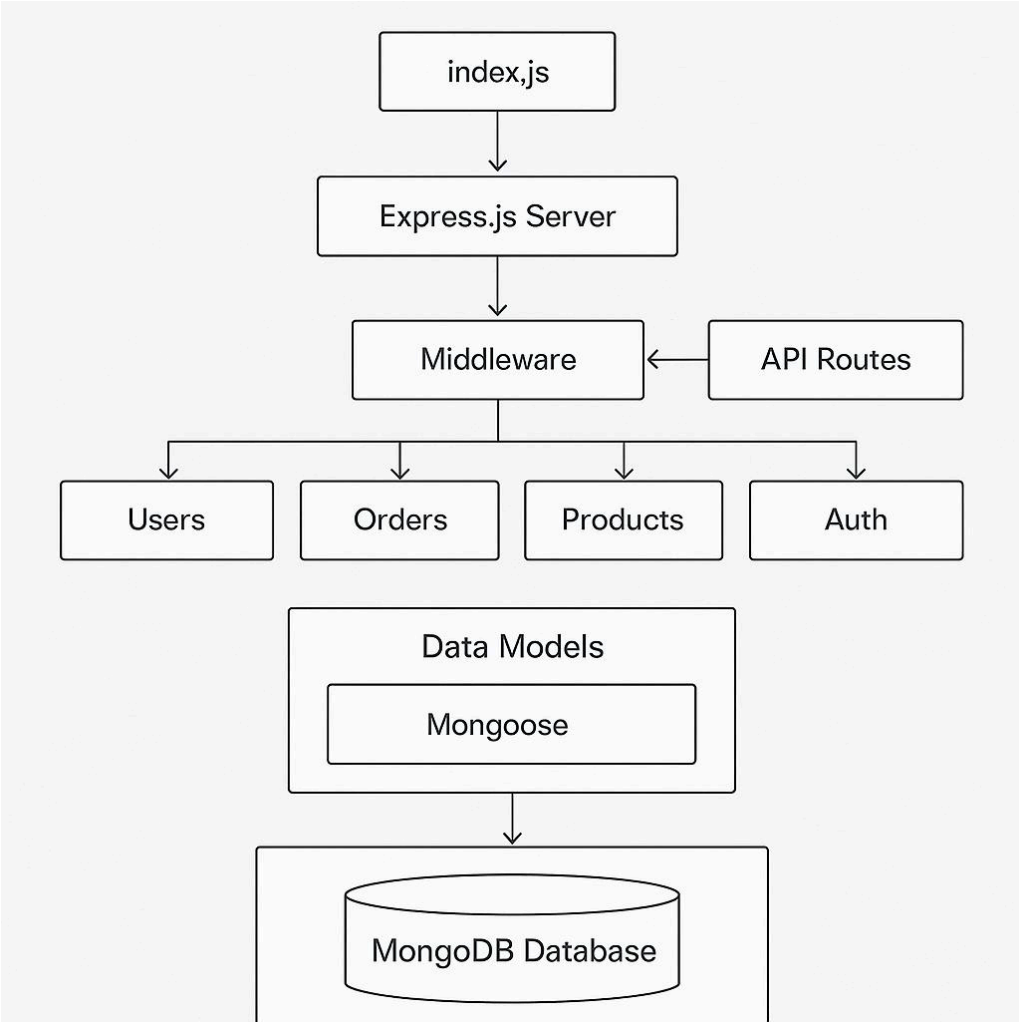
3. Orders Collection (Orders)
   - Fields:
     - userId (reference to User)
     - productId (reference to Product)
     - productName
     - quantity
     - size
   - Purpose: Records completed orders placed by users.
   - Relationships:
     - References User and Product to link the order to who made it and what was ordered.

4. Cart Collection (Cart)
   - Fields:
     - userId (reference to User)
     - productId (reference to Product)
     - productName
     - quantity
     - size
   - Purpose: Tracks products added to a user's cart before checkout.
   - Relationships:

    - Also references User and Product collections.

5. Admin Collection (Admin)
   - Fields:
     - categories
     - banner
   - Purpose: Stores administrative data like product categories and homepage banners.
   - Relationships:
     - Standalone, no foreign keys; but it's crucial for front-end display and category organization.

**Backend Architecture**




1. index.js
   - This is the entry point of the backend.

- It initializes the Express server and connects to the MongoDB database.

2. Express.js Server
   - Handles incoming HTTP requests and sends responses.
   - Sets up middleware and API routing.

3. Middleware
   - Code that runs between the request and response.
   - Common uses: logging, CORS handling, JSON parsing, authentication checks.

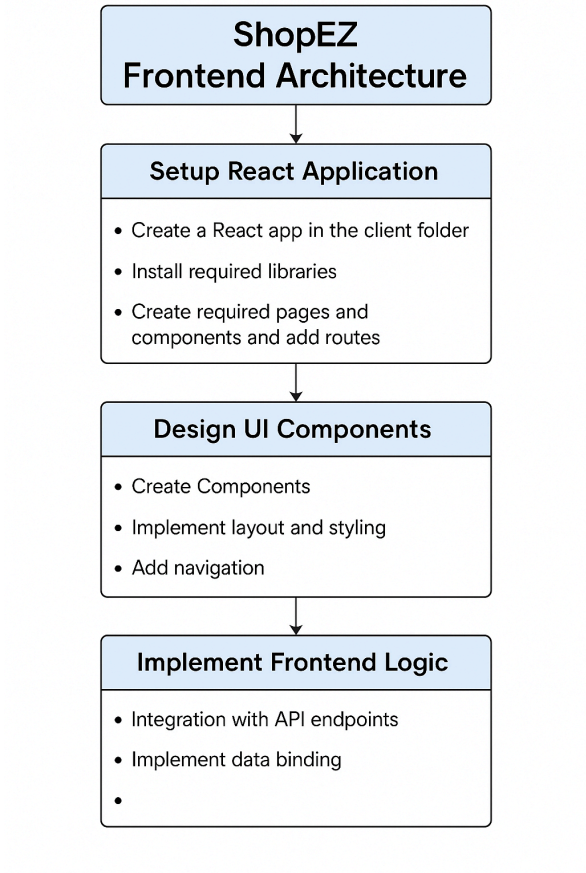
4. API Routes
   - Define endpoints like /users, /orders, /products, /auth.
   - Each endpoint maps to specific controller logic.

5. Users, Orders, Products, Auth
   - Modular route handlers.
   - Example: Users route handles registration/login; Orders handles placing orders.

6. Data Models (Mongoose)
   - Mongoose defines schemas (blueprints) for MongoDB documents.
   - Ensures structured interaction with the database.

7. MongoDB Database
   - Stores persistent data (users, products, carts, etc.).
   - NoSQL, schema-flexible, works well with Mongoose.

**Frontend Architecture**




1. Setup React Application

This is the foundation phase:

- Create a React app in the client folder
  Initialize your project using a command like npx create-react-app client.
- Install required libraries
  E.g., React Router, Axios, Tailwind CSS, Redux, etc.
- Create required pages and components and add routes
  Define pages (e.g., Home, Product, Cart, Profile) and configure routing using react-router-dom.

2. Design UI Components

This is the interface design phase:

- Create Components
  Build reusable elements like Navbar, ProductCard, CartItem, etc.

- Implement layout and styling
  Use CSS, SCSS, or frameworks like Tailwind or Bootstrap for design consistency.
- Add navigation
  Add <Link> or <NavLink> components to navigate between pages smoothly.

3. Implement Frontend Logic

This is the functionality phase:

- Integration with API endpoints
  Use Axios or Fetch to connect frontend to backend APIs (login, product fetch, order creation, etc.).
- Implement data binding
  Use useState, useEffect, or Context/Redux to reflect real-time data in the UI—e.g., cart updates or login status.

## 4. Set Up Instructions

To set up and run the **ShopEZ** e-commerce application built with the **MERN** stack (MongoDB, Express.js, React.js, Node.js), ensure the following software and tools are installed:

### 4.1 Software Dependencies

- **Node.js & npm** (Server-side JS runtime & package manager)
  Download Node.js
  Installation Instructions

- **MongoDB** (NoSQL database for storing products, users, orders)
  Download MongoDB
  Installation Instructions
- **Express.js** (Web framework for Node.js)
  Install via terminal: npm install express
- **React.js** (Frontend library for UI components)
  React Setup Guide
- **Mongoose** (ODM for MongoDB in Node.js)
  Used to perform CRUD operations with ease
  Tutorial to Connect Node.js with MongoDB
- **Git** (Version control system)
  Download Git

- **Code Editor / IDE**

  Visual Studio Code

  Sublime Text

  WebStorm

### 4.2 Installation

Follow the steps below to clone and set up the ShopEZ application locally:

**1. Clone the Repository**

Open terminal and run:

git clone https://github.com/nihalikakumari/ShopEZ-E-commerce-Application.git

**2. Navigate to the Project Folder**

cd ShopEZ-Ecommerce-App-MERN

**3. Install Dependencies**

npm install

**4. Set Up Environment Variables**

Create a .env file in the root directory and include the following (example):

PORT=3000

MONGO_URI=mongodb://localhost:27017/shopez

JWT_SECRET=your_jwt_secret_key

Replace MONGO_URI with your actual local or cloud MongoDB URI.

**5. Start the Development Server**

npm run dev

or

npm start

**6. Access the Application**

Open your browser and navigate to:
http://localhost:3000

## 5. Folder Structure

Project Root (/shop-ez)

```
/shop-ez
├── client/      ← React Frontend
├── server/      ← Express Backend
├── .gitignore
├── package.json (optional: root-level for mono-repo setup)
└── README.md
```

Client: React Frontend Structure (/client)

```
/client
├── public/
│   └── index.html
├── src/
│   ├── assets/           ← Images, icons, etc.
│   ├── components/       ← Reusable UI components (Navbar, Footer, etc.)
│   ├── pages/            ← Page views (Home, Login, ProductList, etc.)
│   ├── routes/           ← React Router DOM routes
│   ├── services/         ← API service files (axios calls to backend)
│   ├── utils/            ← Helper functions
│   ├── context/          ← Context API for state (auth, cart, etc.)
│   ├── App.js            ← Root component
│   ├── index.js          ← Entry point
│   └── styles/           ← Global styling (CSS/SCSS/Tailwind config)
├── .env                  ← Environment variables
├── package.json
└── README.md
```

Server: Node.js + Express Backend Structure (/server)

```
/server
├── config/
│   └── db.js                ← MongoDB connection setup
├── controllers/             ← Route logic (e.g., userController.js)
│   ├── authController.js
│   ├── productController.js
│   └── orderController.js
├── middleware/
│   ├── auth.js              ← Auth middleware (JWT verification)
│   └── errorHandler.js      ← Centralized error handling
├── models/                  ← Mongoose schemas
│   ├── User.js
│   ├── Product.js
│   └── Order.js
├── routes/                  ← Express route handlers
│   ├── authRoutes.js
│   ├── productRoutes.js
│   └── orderRoutes.js
├── utils/
│   └── validators.js        ← Input validation, helpers
├── .env                     ← Environment variables (Mongo URI, JWT secret)
├── index.js                 ← Entry point (starts the Express server)
├── package.json
└── README.md
```

- The **React app** communicates with the Express backend using **Axios** via service files in client/src/services.
- Authentication can be handled using **JWT tokens**, stored in localStorage or cookies.
- Use **React Router** for client-side navigation, and **Redux** or **Context API** if global state management is needed.
- Consider using **Tailwind CSS** or **styled-components** for styling consistency.

## 6. Running the Application

- Frontend: npm start in the client directory.
- Backend: npm start in the server directory.

## 7. API Documentation

#### Authentication

- Use JWT token in Authorization: Bearer <token> header.
- Endpoints mostly require authentication; admin-only actions are clearly marked.

#### Auth Endpoints

- **Login**: /auth/login (POST) – Returns user data + JWT token.
- **Get Profile**: /auth/profile (GET) – Returns user profile data.

#### User Management

- **Register**: /users (POST)
- **Update Profile**: /users/profile (PUT)
- **Get All Users (Admin)**: /users (GET)
- **Delete User (Admin)**: /users/:id (DELETE)

#### Products

- **List Products**: /products (GET) with filters for search, category, price, etc.
- **Get Product**: /products/:id (GET)
- **Create/Update/Delete Product (Admin)**: /products (POST), /products/:id (PUT/DELETE)
- **Add Review**: /products/:id/reviews (POST)

- **Top / Featured / New / Sale Products**: /products/top, /products/featured, /products/new, /products/sale

**Orders**

- **Create Order**: /orders (POST)
- **Get Order by ID**: /orders/:id (GET)

## 8. Authentication

All protected routes require a JWT token in the Authorization header:

Authorization: Bearer <token>

The token is obtained during login or registration and should be included in all subsequent requests to protected endpoints.

**Error Responses**

All API endpoints return appropriate HTTP status codes:

- `200 OK`: Request succeeded

- `201 Created`: Resource created successfully

- `400 Bad Request`: Invalid request parameters

- `401 Unauthorized`: Authentication required or failed

- `404 Not Found`: Resource not found

- `500 Internal Server Error`: Server error

Error responses include a message field explaining the error:

{

"message": "Error message details"

}

```
```

**Testing the API**

You can test the API using tools like Postman or curl. Here are some example requests:

**Register a new user**

```
curl -X POST http://localhost:5000/api/users \
  -H "Content-Type: application/json" \
  -d '{"username": "testuser", "email": "test@example.com", "password": "password123"}'
```

**Login**

```
curl -X POST http://localhost:5000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email": "test@example.com", "password": "password123"}'
```

**Get products**

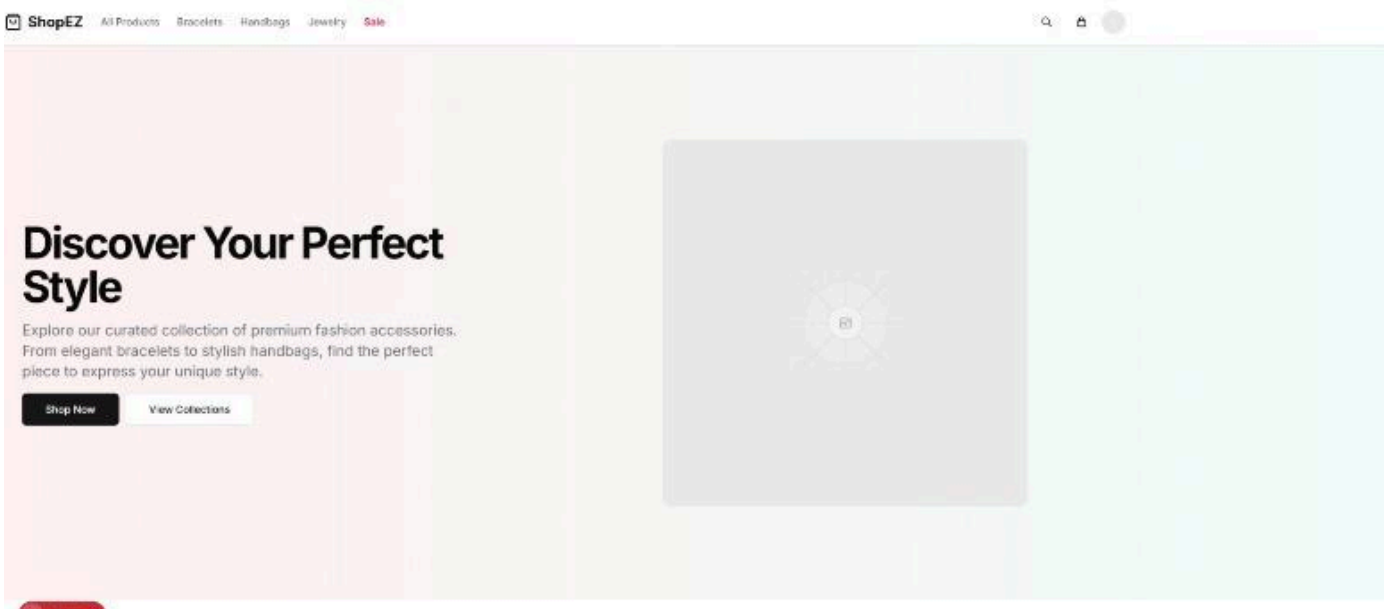
```
curl -X GET http://localhost:5000/api/products
```

**Create an order (authenticated)**

```
curl -X POST http://localhost:5000/api/orders \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -d '{
    "orderItems": [
      {
        "name": "Crystal Charm Bracelet",
        "quantity": 1,
        "image": "/placeholder.svg",
        "price": 89.99,
```

```
      "product": "PRODUCT_ID"
    }
  ],
  "shippingAddress": {
    "address": "123 Main St",
    "city": "Boston",
    "postalCode": "02108",
    "country": "USA"
  },
  "paymentMethod": "PayPal",
  "itemsPrice": 89.99,
  "taxPrice": 9.00,
  "shippingPrice": 0,
  "totalPrice": 98.99
}'
```

**9. User Interface**




## Featured Products

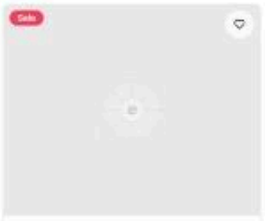
Discover our most popular items and latest arrivals


Bracelets

**Crystal Charm Bracelet**

$89.99

Add to Cart




Handbags

**Leather Crossbody Bag**

$129.99 ~~$160.00~~

Add to Cart


Jewelry

**Pearl Drop Earrings**

$69.99

Add to Cart



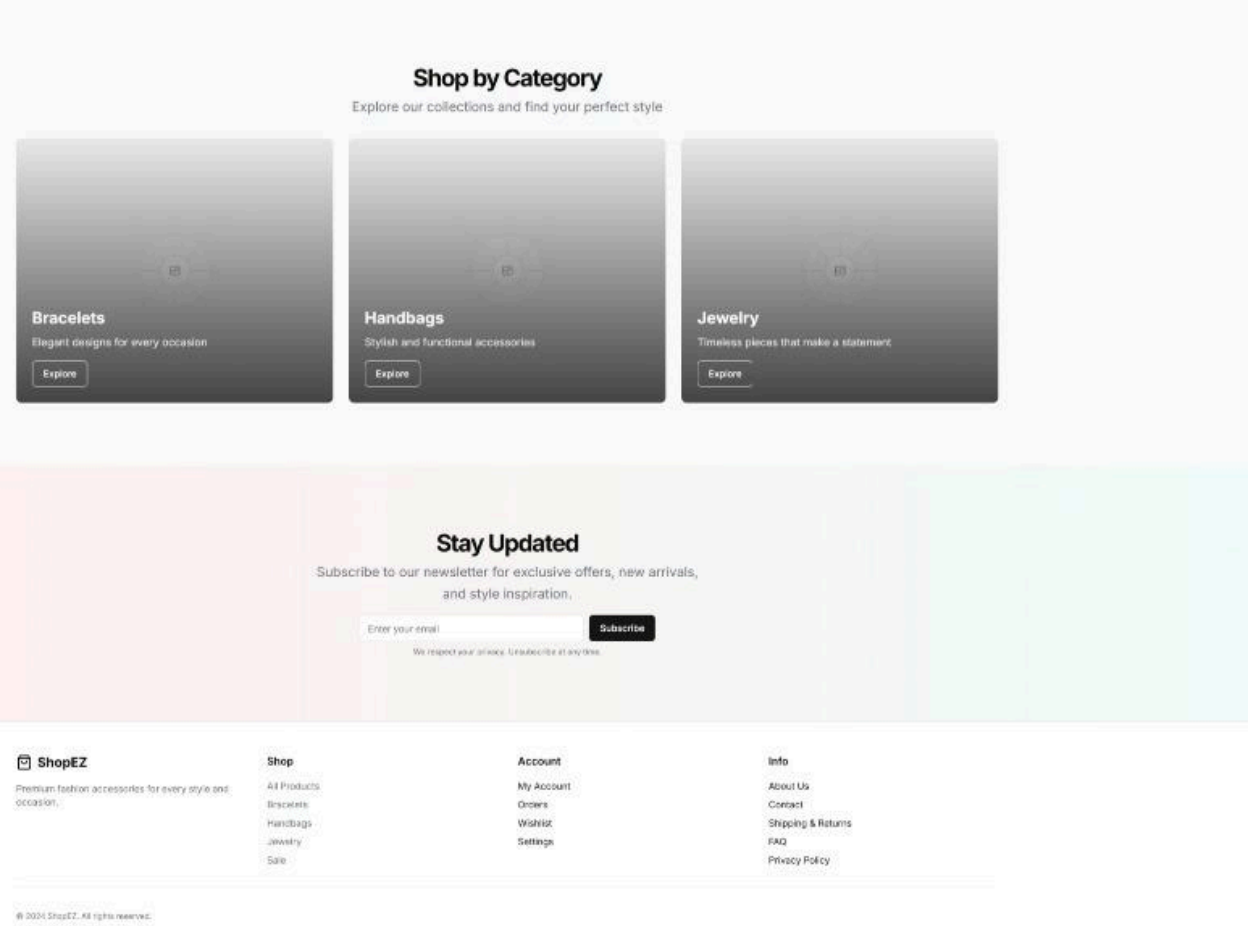
Jewelry

**Gold Chain Necklace**

$99.99

Add to Cart

View All Products




## 10. Testing

### Tools Used:

| **Layer** | **Tool** | **Purpose** |
|---|---|---|
| Backend | **Jest** | JavaScript testing framework for unit & integration tests |
| Backend | **Supertest** | For testing Express routes & APIs |

| | | |
|---|---|---|
| Frontend | **React Testing Library** | Testing UI components in isolation |
| Frontend | **Jest** | Underlying test runner for React Testing Library |
| Manual Tests | Browser/Postman | Manual end-to-end testing of flows & APIs |

**Backend Testing Strategy**

- **Unit Tests:**
  Test individual functions such as controllers and utilities (e.g., hashing passwords, calculating totals).
- **Integration Tests:**
  Test end-to-end API endpoints with an in-memory MongoDB database using Supertest.

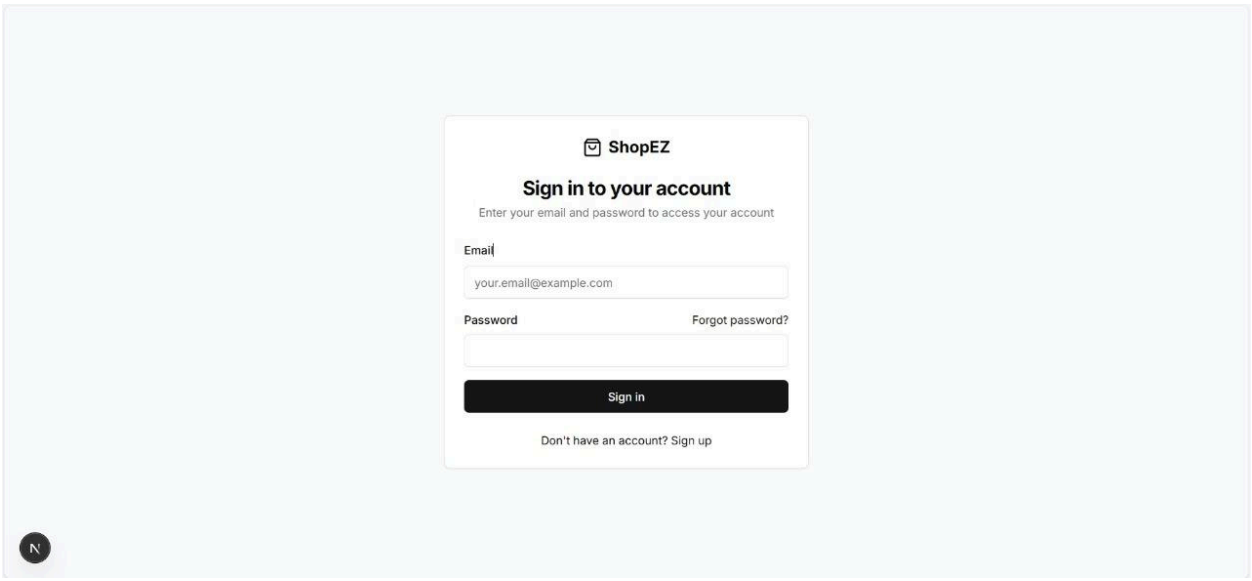
**Frontend Testing Strategy**

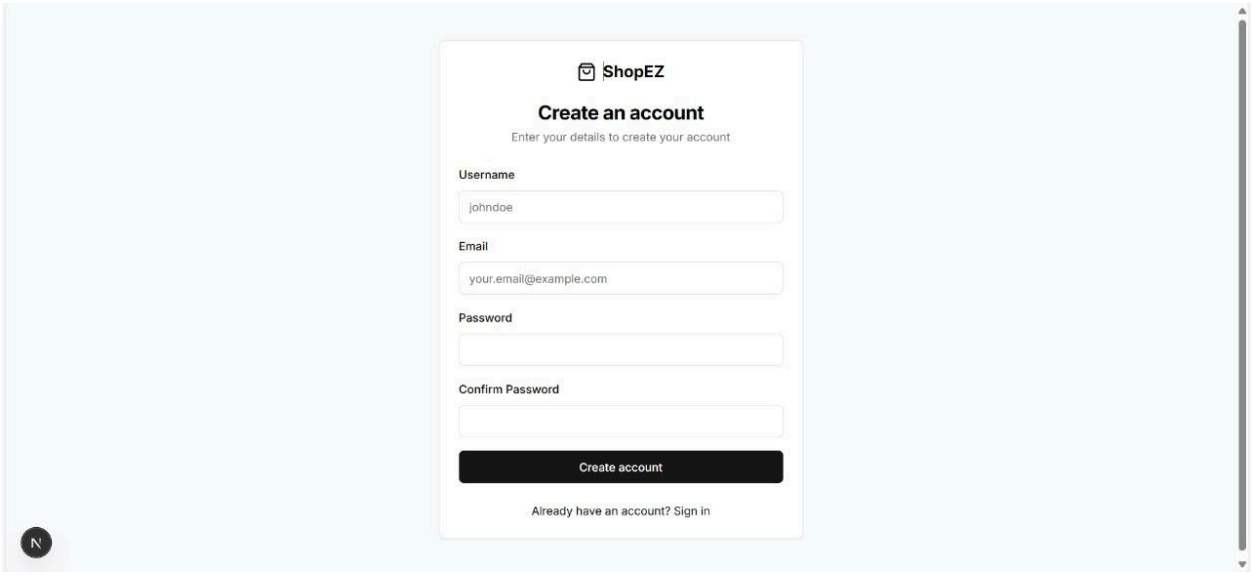
- **Component Tests:**
  Use React Testing Library to verify that components render correctly and interact as expected.

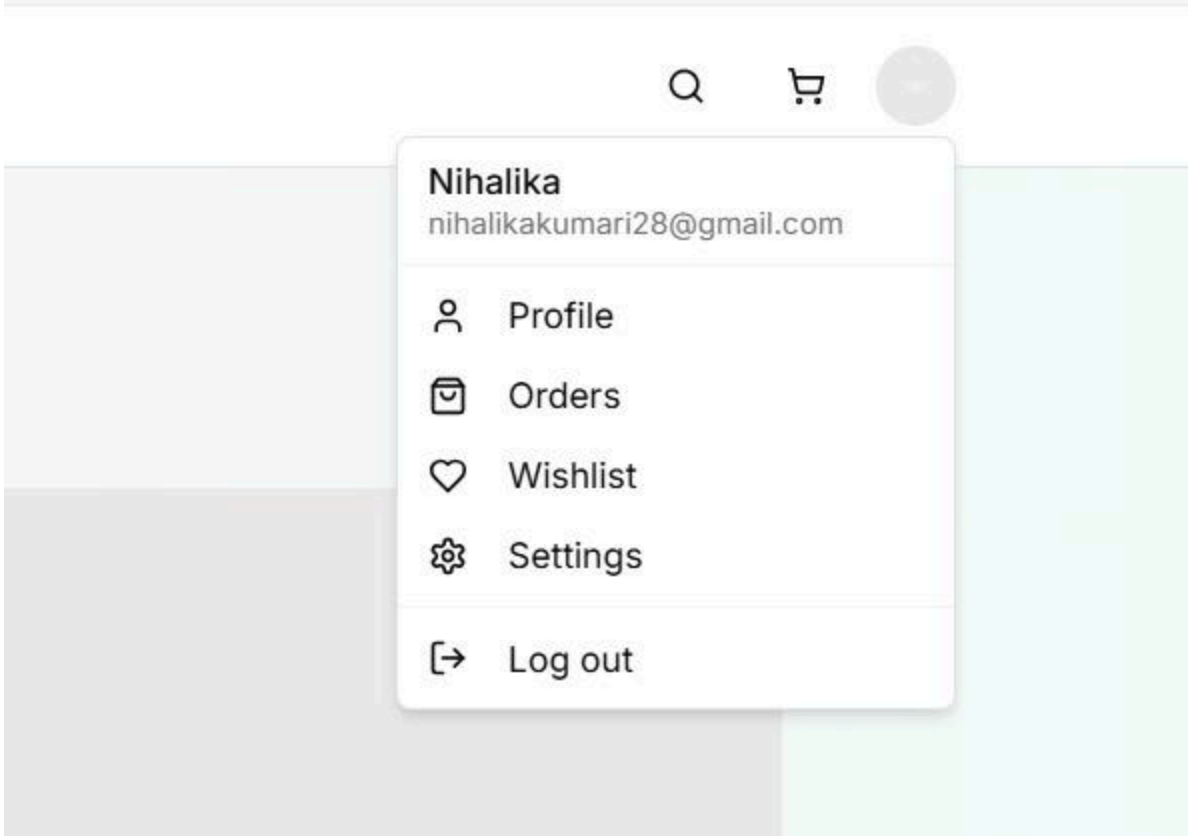
## 11. Screenshots or Demo

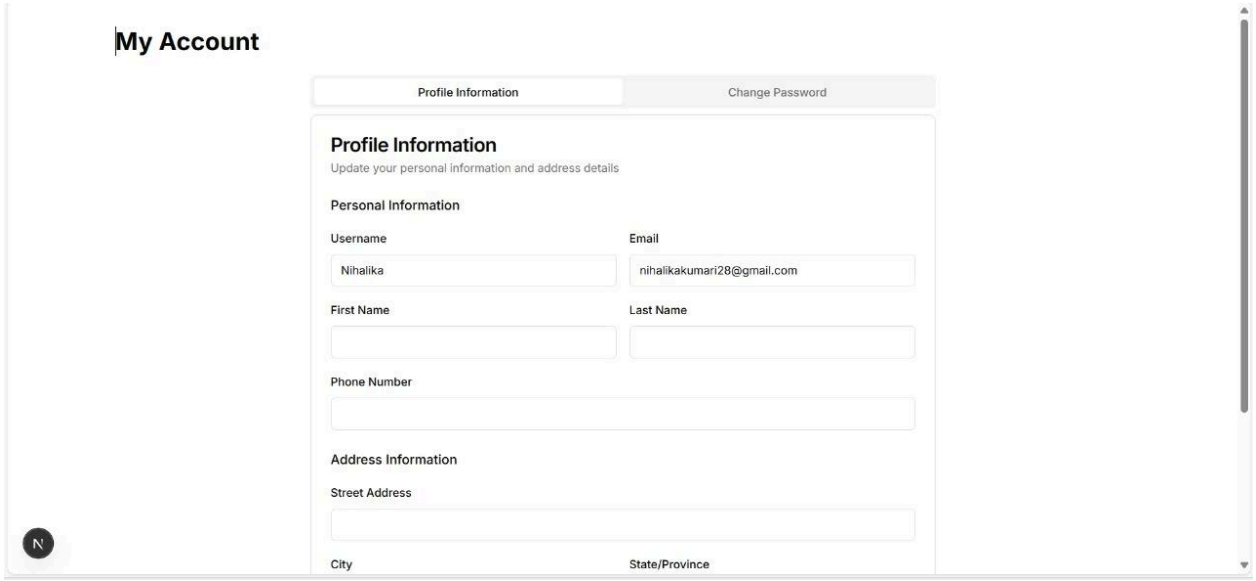
Demo link: https://drive.google.com/file/d/1pkRUoN_CfH2FfPHobnhE3ysJmVEUz2TP/view?usp=sharing

Screenshots:

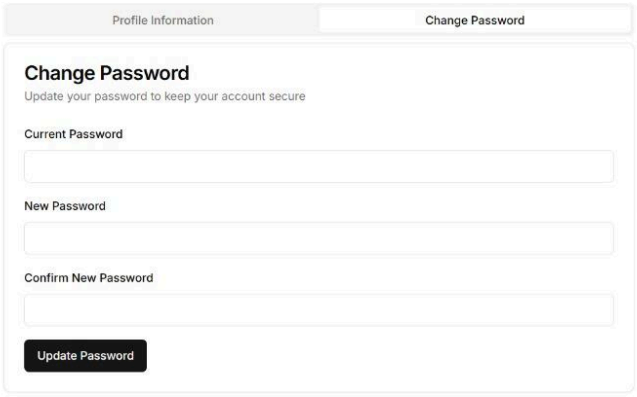

ShopEZ
Sign in to your account
Enter your email and password to access your account
Email
your.email@example.com
Password
Forgot password?
Sign in
Don't have an account? Sign up

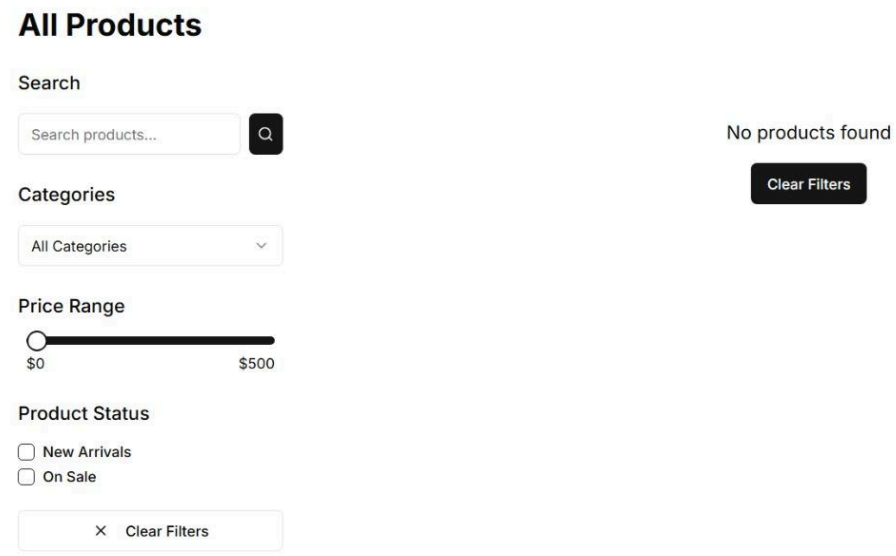

ShopEZ
Create an account
Enter your details to create your account
Username
johndoe
Email
your.email@example.com
Password
Confirm Password
Create account
Already have an account? Sign in


Nihalika
nihalikakumari28@gmail.com
Profile
Orders
Wishlist
Settings
Log out


My Account
Profile Information
Change Password
Profile Information
Update your personal information and address details
Personal Information
Username
Nihalika
Email
nihalikakumari28@gmail.com
First Name
Last Name
Phone Number
Address Information
Street Address
City
State/Province

## My Account


Profile Information
Change Password
Change Password
Update your password to keep your account secure
Current Password
New Password
Confirm New Password
Update Password





## 12. Known Issues

- Order Confirmation Delay
    - Issue: Order confirmation page occasionally delays due to backend processing time or MongoDB write latency.
    - Impact: UI may appear unresponsive after placing an order.
    - Workaround: Refreshing shows the updated order.
- Role-Based Access Gaps
    - Issue: Some admin routes do not fully enforce role-based access.

    - Impact: Authenticated users may access restricted endpoints if checks are not comprehensive.
    - Planned Fix: Strengthen middleware and add route-level guards.
- Mobile UI Optimization
    - Issue: Some frontend components may not be fully responsive on smaller devices.
    - Impact: UI elements may overlap or be misaligned on mobile.
    - Planned Fix: Responsive design updates in upcoming UI refactor.

**13. Future Enhancements**

1. Enhanced User Experience
    - Add product filters (price, rating, brand, etc.)
    - Implement search suggestions/autocomplete
    - Add product reviews and ratings
    - Enable wishlist functionality

2. Progressive Web App (PWA)
    - Make ShopEZ installable on mobile
    - Add offline support for browsing cached products

3. Secure Payment Gateway Integration
    - Integrate Razorpay, Stripe, or PayPal for online payments
    - Add payment success/failure tracking

4. Role-Based Access Control
    - Separate admin and user privileges
    - Create moderator roles for reviewing product listings/reviews

5. Order Tracking & Notifications
    - Add shipment tracking feature
    - Send email/SMS/Push notifications on order status changes
6. Analytics Dashboard
    - For admins to track sales, user growth, and product popularity
    - Include visual graphs and stats

7. AI-Powered Recommendations
    - Suggest products based on user history or similar user behavior

8. Multilingual & Currency Support
    - Support for multiple languages and region-based pricing