# TITLE

## SHO P EZ ECO MMERCE U SI NG

**ME RN STA CK**

# TEAM MEMBERS

1. Shoban babu
2. Shaik Imran
3. Shaik kaif
4. Sowmya

### Introduction to shopEZ:

.

What is shopEZ?

* A modern e-commerce platform.
* Designed for user-friendly shopping experience. Why MERN stack for development?
* Full-stack JavaScript solution.
* Efficient, scalable, and widely adopted for web apps.

### Introduction to MERN Stack:

MERN Components:

* MongoDB: NoSQL database for storing product and user data.
* Express.js: Back-end web framework.
* React.js: Front-end library for building user interfaces.
* Node.js: JavaScript runtime for building server-side applications.

It sounds like you're working on an eCommerce application for "Shop EZ." Here are some key elements you might consider for

the application, especially if you’re developing it with the MERN stack (MongoDB, Express, React, Node.js):

### Homepage

* Display trending products, categories, and special offers.
* Feature a user-friendly design with an intuitive layout for easy navigation.
* Include search and filtering options to help users find products quickly.

### Product Pages

* Detailed product descriptions, including images, specifications, reviews, and pricing.
* A related products section to encourage additional purchases.
* Add-to-cart and wish-list options.

### Shopping Cart and Checkout

* A shopping cart that allows users to adjust quantities and view the total price.
* A seamless checkout process with multiple payment options (credit/debit cards, digital wallets, etc.).
* Order summary and confirmation details.

### User Authentication

* Secure registration and login, possibly integrating social logins.
* Password management and profile settings for users to manage their account information.

### Order Management

* Allow users to track order status, view past orders, and initiate returns if necessary.
* Notifications via email or SMS for order confirmation, shipping, and delivery updates.

### Admin Dashboard

* Product and category management, including adding, editing, and removing items.
* Order and customer management to fulfill orders and respond to queries.
* Analytics to track sales, user activity, and other key metrics.

### Search and Filtering

* Advanced search capabilities and filters based on category, price, brand, etc.
* Sort products by relevance, price, popularity, and reviews.

### Review and Ratings

* Enable customers to leave feedback on products.
* Display average ratings and reviews on product pages.

### Responsive Design

* Ensure compatibility with both desktop and mobile devices for a smooth shopping experience.

### Security Features

* Data encryption, secure payment processing, and protection of sensitive information.
* Implement user role-based access control, particularly for the admin dashboard.

### Suggested Tech Stack:

* **Frontend**: React.js for user interface.
* **Backend**: Node.js and Express.js for server-side logic.
* **Database**: MongoDB for storing user data, product info, and orders.
* **Authentication**: JWT (JSON Web Token) for secure authentication.

### Possible Additional Features:

* **Personalized Recommendations**: Display recommended products based on browsing history.
* **Discount Codes and Offers**: Allow users to apply promo codes during checkout.
* **Wishlist and Favorites**: Users can save products for future reference.

Let me know if you need more detail on any specific feature, or if there's a part you want to dive deeper into!

### Architecture Diagram:

* + Show the diagrams that explains the interaction between React.js (Frontend), Express.js (Backend), and MongoDB (Database).

Explain the flow:

* + User interacts with React frontend.
  + React sends requests to the Node.js backend via Express.

Diagram showing the flow between:

* + Front-end (React.js)
  + Back-end (Express.js & Node.js)
  + Database (MongoDB)
  + Explanation of communication between components.
  + Data is stored/retrieved from MongoDB.

**FLOWCHART**

User Visits Website

|

User Logs In/Register

|

Browses Products

|

Adds Items to Cart

| Checks Out

|

Receives Confirmation

|

Views Order History

### Setup of MERN Stack Project

Start by creating your project directories and initializing your backend and frontend:

# Create directories

**mkdir** shopEZ

**cd** shopEZ

# Initialize backend

**mkdi**r backend && **cd** backend npm init -y

npm install express mongoose dotenv cors bcryptjs jsonwebtoken

# Initialize frontend

**cd** ..

npx create-react-app frontend

**cd** frontend

npm install axios react-router-dom

### Backend Setup: (Node.js + Express)

Backend Structure:

1. **backend/server.js**: Entry point.
2. **backend/models**: Contains data models for MongoDB.
3. **backend/routes**: Routes for different API endpoints. Backend Code for Product Model and API: **backend/models/Product.js**

Define the product model using Mongoose:

### backend/routes/productRoutes.js

Define API endpoints to interact with the product database:

### backend/server.js

**Define the product model using Mongoose:**

const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({ name: { type: String, required: true }, description: String,

price: { type: Number, required: true }, category: String,

stock: Number, imageUrl: String,

});

module.exports = mongoose.model('Product', productSchema);

### Define API endpoints to interact with the product database:

const express = require('express');

const Product = require('../models/Product'); const router = express.Router();

// Get all products

router.get('/', async (req, res) => { try {

const products = await Product.find();

res.json(products);

} catch (error) {

res.status(500).json({ message: error.message });

}

});

// Get a specific product router.get('/:id', async (req, res) => {

try {

const product = await Product.findById(req.params.id);

if (!product) return res.status(404).json({ message: 'Product not found' });

res.json(product);

} catch (error) {

res.status(500).json({ message: error.message });

}

});

module.exports = router;

### Initialize Express and connect to MongoDB:

const express = require('express');

const mongoose = require('mongoose'); const dotenv = require('dotenv');

const cors = require('cors');

const productRoutes = require('./routes/productRoutes');

dotenv.config();

const app = express(); app.use(cors()); app.use(express.json());

// MongoDB connection

mongoose.connect(process.env.MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log("MongoDB connected"))

.catch((error) => console.error(error));

// Routes

app.use('/api/products', productRoutes);

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

### User Authentication Flow:

const bcrypt = require('bcrypt');

const User = require('./models/User'); // Mongoose User model

app.post('/register', async (req, res) => {

const hashedPassword = await bcrypt.hash(req.body.password, 10); const user = new User({

username: req.body.username, password: hashedPassword

});

await user.save(); res.status(201).send('User registered');

});

### Frontend Setup: (React)

React Product Listing and Product Details Page**:**

In **frontend/src/App.js**, set up routing for the application:

Product List Component**:**

### frontend/src/components/ProductList.js

Product Details Component:

### frontend/src/components/ProductDetails.js

**React Product Listing and Product Details Page**:

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom'; import ProductList from './components/ProductList';

import ProductDetails from './components/ProductDetails';

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<ProductList />} />

<Route path="/product/:id" element={<ProductDetails />} />

</Routes>

</Router>

);

}

export default App;

### Product List Component:

import React, { useState, useEffect } from 'react'; import axios from 'axios';

import { Link } from 'react-router-dom';

const ProductList = () => {

const [products, setProducts] = useState([]); useEffect(() => {

const fetchProducts = async () => { try {

const response = await axios.get('http://localhost:5000/api/products');

setProducts(response.data) } catch (error) {

console.error('Error fetching products', error); } }; fetchProducts(); }, []);

return (

<div><h2>Product List</h2>

<div className="product-grid">

{products.map((product) => (

<div key={product.\_id} className="product-card">

<Link to={`/product/${product.\_id}`}>

<img src={product.imageUrl} alt={product.name} />

<h3>{product.name}</h3></Link>

<p>${product.price}</p>

</div> ))}

</div> </div>);}; export default ProductList;

### Product Details Component:

import React, { useState, useEffect } from 'react'; import axios from 'axios';

import { useParams } from 'react-router-dom'; const ProductDetails = () => {

const { id } = useParams();

const [product, setProduct] = useState(null); useEffect(() => {

const fetchProduct = async () => {

try {

const response = await axios.get(`http://localhost:5000/api/products/${id}`); setProduct(response.data);

} catch (error) {

console.error('Error fetching product details', error); } };

fetchProduct();}, [id]);

if (!product) return <div>Loading...</div>; return (

<div>

<img src={product.imageUrl} alt={product.name} />

<h2>{product.name}</h2>

<p>{product.description}</p>

<p>Price: ${product.price}</p>

</div>

);};

export default ProductDetails;

### Cart Functionality (React.js):

const Cart = () => {

const [cart, setCart] = useState([]);

const addToCart = (product) => { setCart([...cart, product]);

};

return (

<div>

<h2>Cart</h2>

{cart.length === 0 ? <p>Your cart is empty</p> : <ul>{cart.map(item => <li

key={item.id}>{item.name}</li>)}</ul>}

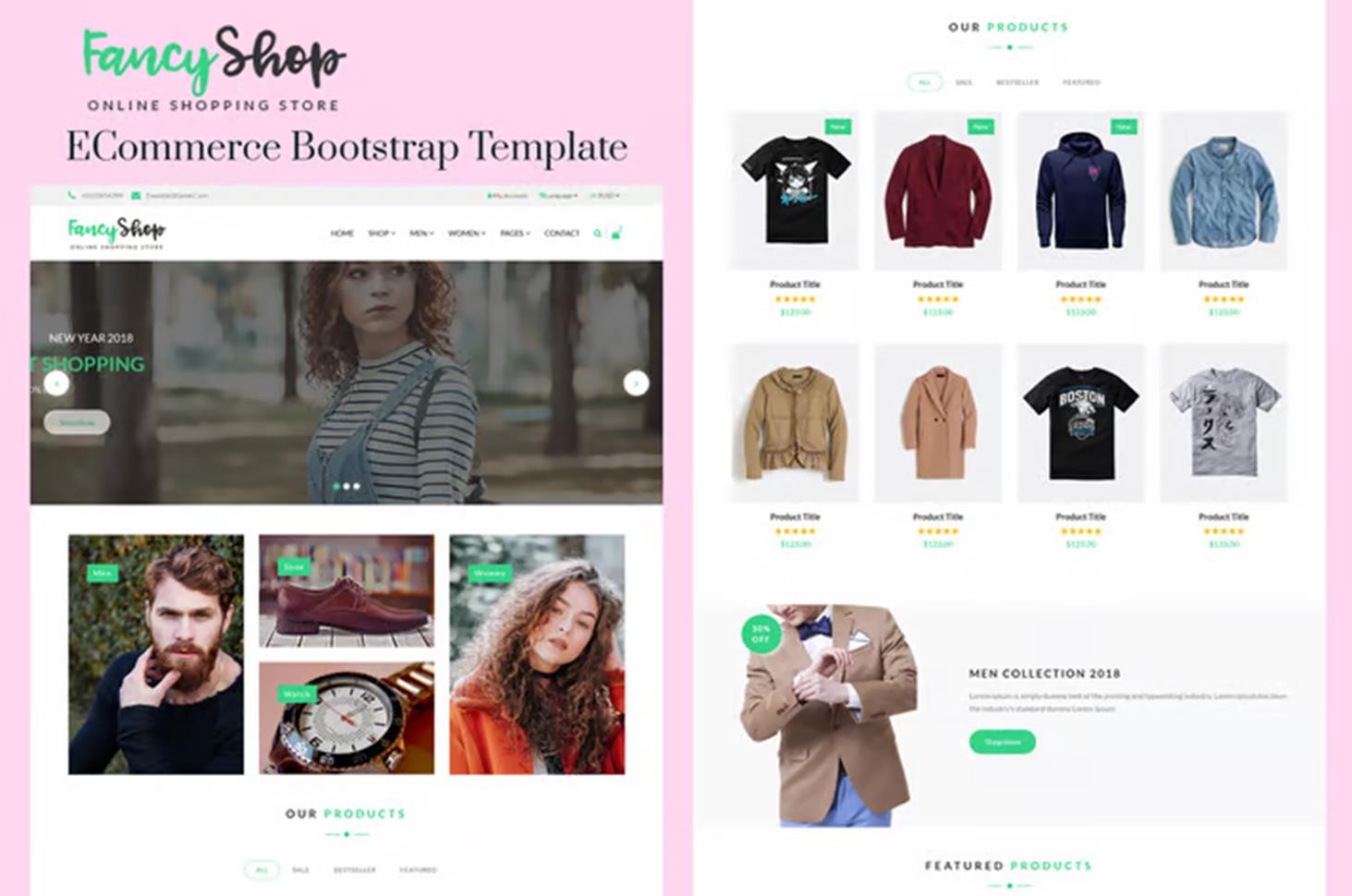
</div>

);

};

export default Cart;





### Running the Application

Start the Backend Server:

**cd** backend

### node server.js

Start the Frontend:

**cd** frontend

npm start

This setup will give you a basic eCommerce application with product listing and product details pages. Additional features like a shopping cart, user authentication, or admin dashboard would follow similar patterns but require more detailed backend logic and frontend components. Let me know if you'd like to expand on any part!

# Output/Demo

* Show screenshots or GIFs of:
* User logging in.
* Product listing.
* Adding items to cart.
* Checkout process.

# Merits:

1. Full-Stack JavaScript:

* Simplifies development by using a single language across the stack.

1. Scalability:

* MongoDB handles large amounts of data, making it easy to scale.

1. Performance:

* Node.js offers non-blocking I/O, enhancing speed and performance.

1. Active Community:

* Large community support and abundant resources for troubleshooting.

1. Component-Based Architecture:

* React’s component-based approach promotes reusability and easier maintenance.

# Demerits:

* 1. SEO Challenges:
  + React applications can face challenges in search engine optimization if not handled properly.
  1. Middleware Overhead:
  + Need for additional middleware for certain functionalities can complicate the architecture.
  1. Rapid Changes:
  + The fast evolution of libraries and frameworks can lead to frequent updates and changes in best practices.
  1. Initial Setup Time:
  + Initial setup and configuration can take time compared to simpler stacks…

## Conclusion and Future Enhancements:

* Summary of the shopEZ platform.
* Potential future features:
* Integration with payment gateways.
* AI-powered product recommendations.
* Improved user experience with real-time updates.
* Recap the architecture and functionality of shopEZ.
* Discuss potential future improvements:
* Integration with payment gateways.
* Recommendation engine with AI.
* Mobile app version.