

ShopEZ: E-commerce Application

Team ID : LTVIP2025TMID24654

Team Members :

Rajulapati Naga Phanindra Kumar

Puritipati Yaswanth Reddy

Punugupati Mounika

Putti Anusha

INTRODUCTION

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

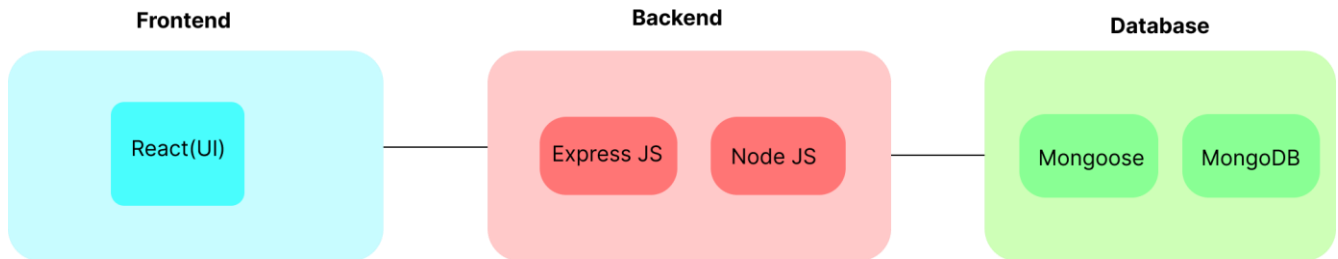
Personalized Shopping Experience

Efficient Order Management for Sellers

Insightful Analytics for Business Growth

PROJECT OVERVIEW

TECHNICAL ARCHITECTURE:

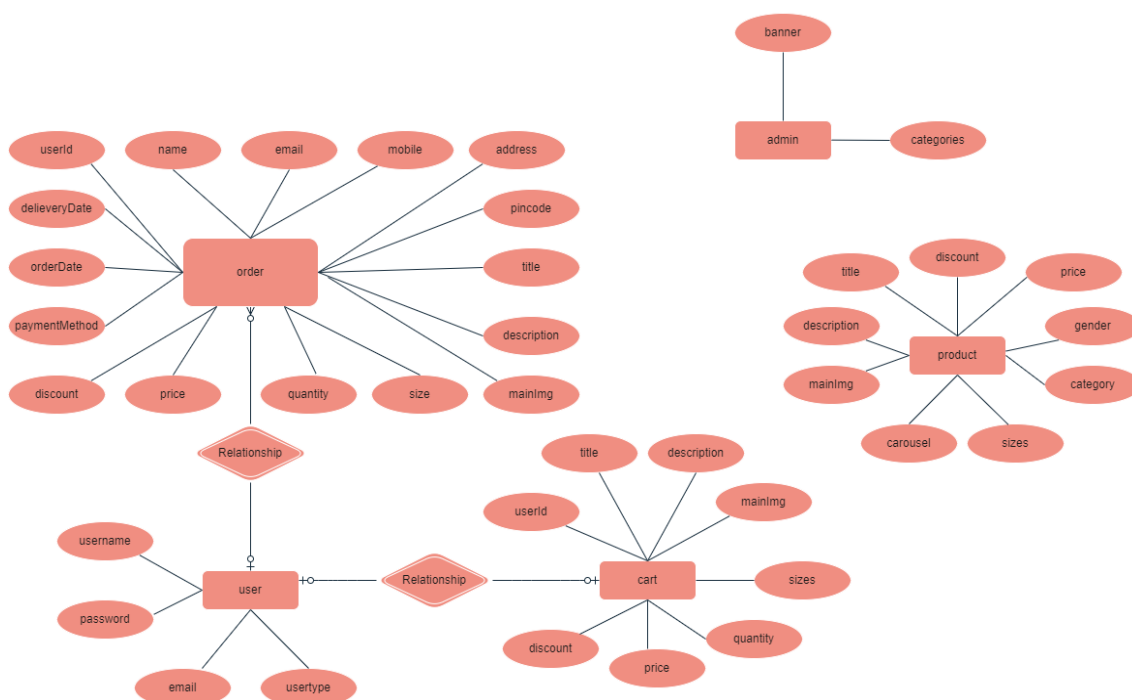


In this architecture diagram:

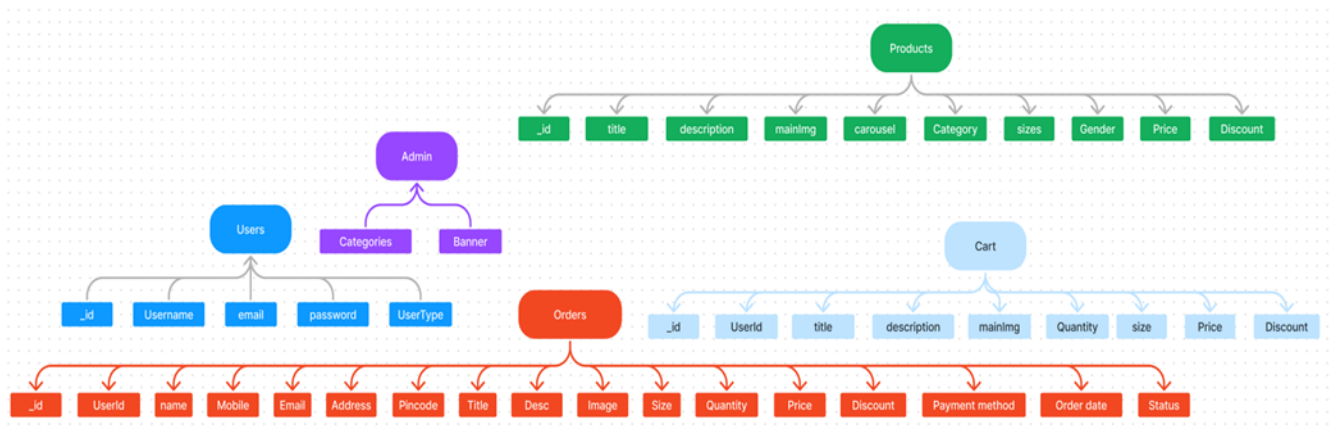
- The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Cart, Products, Profile, Admin dashboard, etc.,
- The backend is represented by the "Backend" section, consisting of API endpoints for Users, Orders, Products, etc., It also includes Admin Authentication and an Admin Dashboard.
- The Database section represents the database that stores collections for Users, cart, Orders and Product.

ER DIAGRAM:

ER-MODEL



- The Database section represents the database that stores collections for Users, Admin, Cart, Orders and products.



The ShopEZ ER-diagram represents the entities and relationships involved in an e-commerce system. It illustrates how users, products, cart, and orders are interconnected. Here is a breakdown of the entities and their relationships:

USER: Represents the individuals or entities who are registered in the platform.

Admin: Represents a collection with important details such as Banner image and Categories.

Products: Represents a collection of all the products available in the platform.

Cart: This collection stores all the products that are added to the cart by users. Here, the elements in the cart are differentiated by the user Id.

Orders: This collection stores all the orders that are made by the users in the platform.

PURPOSE:

Shopez is a full-featured online shopping platform that allows users to easily register, browse products, and place orders. It aims to deliver a secure, seamless, and fast shopping experience with administrative controls for product, user, and order management.

FEATURES:

1. **Comprehensive Product Catalog:** ShopEZ boasts an extensive catalog of products, offering a diverse range of items and options for shoppers. You can effortlessly explore and discover various products, complete with detailed descriptions, customer reviews, pricing, and available discounts, to find the perfect items for your needs.
2. **Shop Now Button:** Each product listing features a convenient "Shop Now" button. When you find a product that aligns with your preferences, simply click on the button to initiate the purchasing process.
3. **Order Details Page:** Upon clicking the "Shop Now" button, you will be directed to an order details page. Here, you can provide relevant information such as your shipping address, preferred payment method, and any specific product requirements.
4. **Secure and Efficient Checkout Process:** ShopEZ guarantees a secure and efficient checkout process. Your personal information will be handled with the utmost security, and we strive to make the purchasing process as swift and trouble-free as possible.
5. **Order Confirmation and Details:** After successfully placing an order, you will receive a confirmation notification. Subsequently, you will be directed to an order details page, where you can review all pertinent information about your order, including shipping details, payment method, and any specific product requests you specified.

In addition to these user-centric features, ShopEZ provides a robust seller dashboard, offering sellers an array of functionalities to efficiently manage their products and sales. With the seller dashboard, sellers can add and oversee multiple product listings, view order history, monitor customer activity, and access order details for all purchases.

ShopEZ is designed to elevate your online shopping experience by providing a seamless and user-friendly way to discover and purchase products. With our efficient checkout process, comprehensive product catalog, and robust seller dashboard, we ensure a convenient and enjoyable online shopping experience for both shoppers and sellers alike.

PREREQUISITES:

To develop a full-stack e-commerce app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm:

Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide:

<https://reactjs.org/docs/create-a-new-react-app.html>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

To Connect the Database with Node JS go through the below provided link: •

Link: <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

To run the existing ShopEZ App project downloaded from github: Follow below steps:

Clone the repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the e-commerce app.
- Execute the following command to clone the repository:

Git clone: <https://github.com/yaswanthpuritipati/ShopEZ-e-commerce-MERN>

Install Dependencies:

- Navigate into the cloned repository directory:
cd ShopEZ-e-commerce-MERN
- Install the required dependencies by running the following command: **npm install**

Start the Development Server:

- To start the development server, execute the following command:
npm run dev or npm run start
- The e-commerce app will be accessible at <http://localhost:3000> by default. You can change the port configuration in the .env file if needed.

Access the App:

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the ShopEZ app on your local machine. You can now proceed with further customization, development, and testing as needed.

USER & ADMIN FLOW:

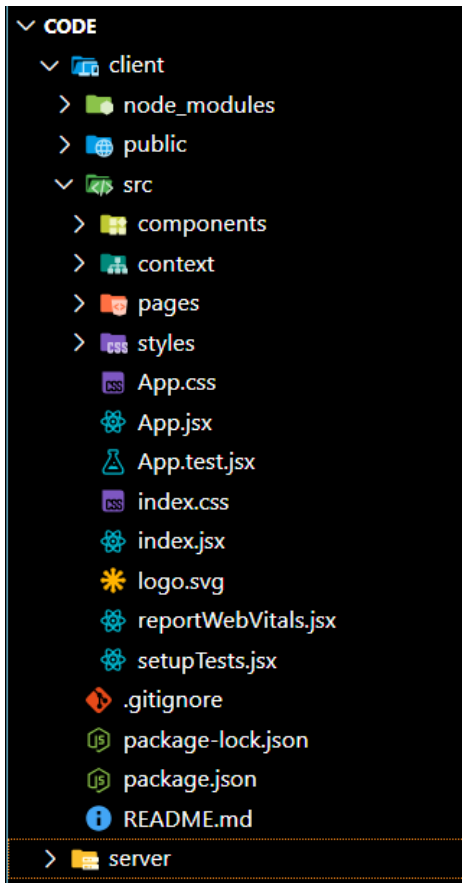
1. User Flow:

- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available products in the platform.
- Users can add the products they wish to their carts and order.
- They can then proceed by entering address and payment details.
- After ordering, they can check them in the profile section.

2. Admin Flow:

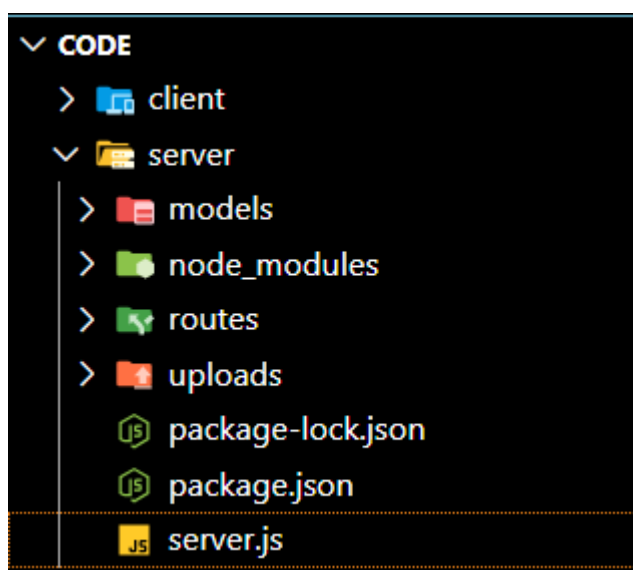
- Admins start by logging in with their credentials.
- Once logged in, they are directed to the Admin Dashboard.
- Admins can access the users list, products, orders, etc.,

PROJECT STRUCTURE:



This structure assumes a React app and follows a modular approach. Here's a brief explanation of the main directories and files:

- src/components: Contains components related to the application such as, register, login, home, etc.,
- src/pages has the files for all the pages in the application.



PROJECT SETUP AND CONFIGURATION:

Install required tools and software:

- Node.js.

Reference Article: <https://www.geeksforgeeks.org/installation-of-node-js-on-windows/>

- Git.

Reference Article: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

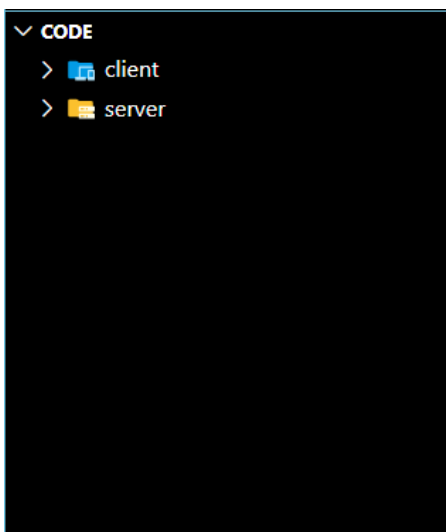
Create project folders and files:

- Client folders.
- Server folders

Referral Video Link:

https://drive.google.com/file/d/1uSMbPIAR6rfAEMcb_nLZAZd5QIjTpnYQ/view?usp=sharing

Referral Image:



DATABASE DEVELOPMENT:

Create database in cloud video link:-

<https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view>

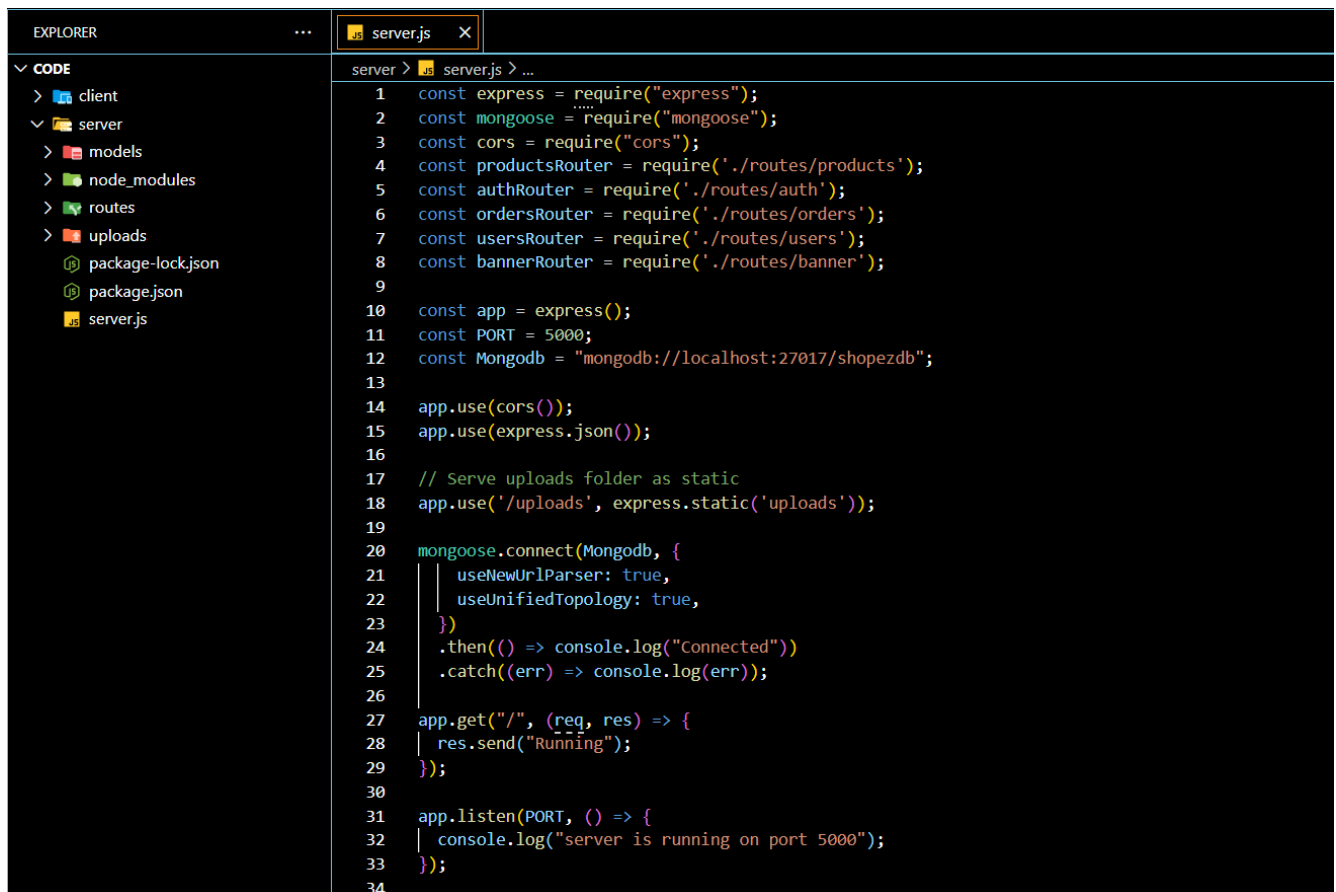
- Install Mongoose.
- Create database connection.

Reference Video of connect node with mongoDB database:

https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoy2Ag/view?usp=sharing

Reference Article: <https://www.mongodb.com/docs/atlas/tutorial/connect-to-your-cluster/>

Reference Image:



```
1  const express = require("express");
2  const mongoose = require("mongoose");
3  const cors = require("cors");
4  const productsRouter = require('./routes/products');
5  const authRouter = require('./routes/auth');
6  const ordersRouter = require('./routes/orders');
7  const usersRouter = require('./routes/users');
8  const bannerRouter = require('./routes/banner');
9
10 const app = express();
11 const PORT = 5000;
12 const MongoDB = "mongodb://localhost:27017/shopezdb";
13
14 app.use(cors());
15 app.use(express.json());
16
17 // Serve uploads folder as static
18 app.use('/uploads', express.static('uploads'));
19
20 mongoose.connect(Mongodb, {
21   useNewUrlParser: true,
22   useUnifiedTopology: true,
23 });
24 .then(() => console.log("Connected"))
25 .catch((err) => console.log(err));
26
27 app.get("/", (req, res) => {
28   res.send("Running");
29 });
30
31 app.listen(PORT, () => {
32   console.log("server is running on port 5000");
33 });
34
```

Model use-case:

1. User Model:

- Model: 'User'

- The User schema represents the user data and includes fields such as username, email, and password.
- It is used to store user information for registration and authentication purposes.
 - The email field is marked as unique to ensure that each user has a unique email address

2. Product Model:

- Model: 'Product'
- The Product schema represents the data of all the products in the platform.
- It is used to store information about the product details, which will later be useful for ordering .

3. Order Model:

- Model: 'Order'
- The Orders schema represents the orders data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,
- It is used to store information about the orders made by users.
- The user Id field is a reference to the user who made the order.

Code Explanation:

Models:

Now let us define the required schemas



```

1  const mongoose = require('mongoose');
2
3  const userSchema = new mongoose.Schema({
4    name: { type: String, required: true },
5    email: { type: String, required: true, unique: true },
6    password: { type: String, required: true },
7    isAdmin: { type: Boolean, default: false },
8    city: { type: String },
9    paymentMethod: { type: String, enum: ['Online', 'Cash on Delivery'] },
10   createdAt: { type: Date, default: Date.now },
11  });
12
13  module.exports = mongoose.model('User', userSchema);
  
```

```
Product.js x
server > models > Product.js > ...
1 const mongoose = require('mongoose');
2
3 const productSchema = new mongoose.Schema({
4   name: { type: String, required: true },
5   price: { type: Number, required: true },
6   discountedPrice: { type: Number },
7   image: { type: String, required: true },
8   description: { type: String, required: true },
9   category: { type: String, required: true },
10  createdAt: { type: Date, default: Date.now },
11 });
12
13 module.exports = mongoose.model('Product', productSchema);

server > models > Order.js > ...
1 const mongoose = require('mongoose');
2
3 const orderSchema = new mongoose.Schema({
4   user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
5   products: [
6     {
7       product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },
8       quantity: { type: Number, required: true, default: 1 },
9     }
10  ],
11   total: { type: Number, required: true },
12   city: { type: String, required: true },
13   paymentMethod: { type: String, enum: ['Online', 'Cash on Delivery'], required: true },
14   status: { type: String, default: 'Pending' },
15   createdAt: { type: Date, default: Date.now },
16 });
17
18 module.exports = mongoose.model('Order', orderSchema);
```

BACKEND DEVELOPMENT:

Setup express server:

- Create index.js file.
- Create an express server on your desired port number.
- Define API's

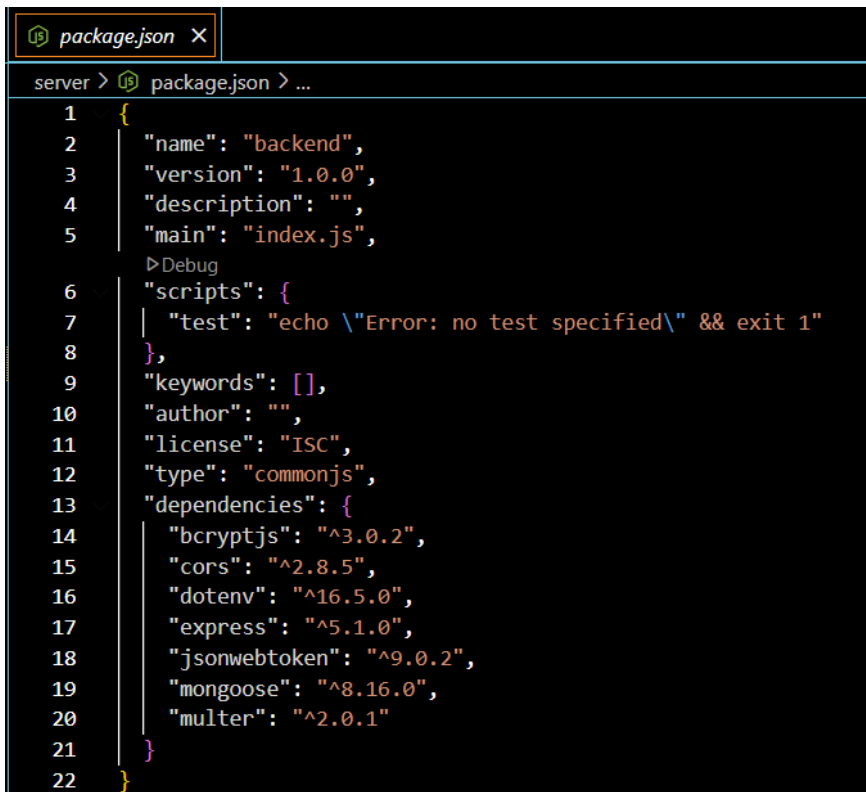
Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRio2qXS/view?usp=sharing

Set Up Project Structure:

- Create a new directory for your project and set up a package.json file using the npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.

Reference Video: <https://drive.google.com/file/d/19df7NU-gQK3DO6wr7ooAfJYIQwnemZoF/view?usp=sharing>

Reference Images:



```
server > package.json > ...
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "type": "commonjs",
13   "dependencies": {
14     "bcryptjs": "^3.0.2",
15     "cors": "^2.8.5",
16     "dotenv": "^16.5.0",
17     "express": "^5.1.0",
18     "jsonwebtoken": "^9.0.2",
19     "mongoose": "^8.16.0",
20     "multer": "^2.0.1"
21   }
22 }
```

2. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for admin, users, products, orders and other relevant data.

3. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies

and cors for handling cross-origin requests.

4. Define API Routes:

- Create separate route files for different API functionalities such as users, orders, and authentication.
- Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

5. Implement Data Models:

- Define Mongoose schemas for the different data entities like products, users, and orders.
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

7. Handle new products and Orders:

- Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.
- Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

- Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.

- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

WEB DEVELOPMENT:

1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

2.Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

3.Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Reference Video Link:

<https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?usp=sharing>

Reference Article Link:

https://www.w3schools.com/react/react_getstarted.asp

Reference Image:

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure with folders like 'client', 'node_modules', 'public', and 'src'. The 'App.jsx' file is selected in the 'src' folder. The main editor area shows the code for 'App.jsx', which defines a function 'App()' that returns a React Router configuration. The configuration includes providers for authentication and cart, and a series of routes for different pages like Home, Register, Login, Cart, Orders, Product Details, Category Page, Admin Dashboard, Admin Products, and Product Form. Below the code editor, the 'TERMINAL' tab is active, showing the output of the build process. It states 'Compiled successfully!', provides the local and network URLs for viewing the frontend, and notes that the development build is not optimized. It also mentions 'webpack compiled successfully'.

```
22
23 function App() {
24   return (
25     <Router>
26       <AuthProvider>
27       <CartProvider>
28       <Header />
29       <Routes>
30         <Route path="/" element={<Home />} />
31         <Route path="/register" element={<Register />} />
32         <Route path="/login" element={<Login />} />
33         <Route path="/cart" element={<Cart />} />
34         <Route path="/orders" element={<Orders />} />
35         <Route path="/orders/:id" element={<OrderDetails />} />
36         <Route path="/product/:id" element={<ProductDetails />} />
37         <Route path="/category/:category" element={<CategoryPage />} />
38         <Route path="/admin" element={<AdminDashboard />} />
39         <Route path="/admin/products" element={<AdminProducts />} />
40         <Route path="/admin/products/new" element={<ProductForm />} />
41         <Route path="/admin/products/:id/edit" element={<ProductForm />} />

```

Compiled successfully!

You can now view **frontend** in the browser.

Local: http://localhost:3000
On Your Network: http://192.168.52.44:3000

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled **successfully**

PROJECT IMPLEMENTATION & EXECUTION:

User Authentication:

- **Backend**

Now, here we define the functions to handle http requests from the client for authentication.


```
authjs x
server > routes > authjs > router.post('/register') callback > error
28
29 // Login
30 router.post('/login', async (req, res) => {
31   try {
32     const { email, password } = req.body;
33     const user = await User.findOne({ email });
34     if (!user) return res.status(400).json({ error: 'Invalid credentials' });
35     const match = await bcrypt.compare(password, user.password);
36     if (!match) return res.status(400).json({ error: 'Invalid credentials' });
37     const token = jwt.sign({ id: user._id, isAdmin: user.isAdmin }, JWT_SECRET, { expiresIn: '7d' });
38     res.json({ token, user: { id: user._id, name: user.name, email: user.email, isAdmin: user.isAdmin } });
39   } catch (err) {
40     res.status(500).json({ error: err.message });
41   }
42 });
43
44 module.exports = router;
```

• Frontend

Login:

```
Login.jsx x
client > src > pages > Login.jsx > ...
8 const Login = () => {
9   const [form, setForm] = useState({ email: '', password: '' });
10  const [loading, setLoading] = useState(false);
11  const [error, setError] = useState(null);
12  const navigate = useNavigate();
13  const { login } = useAuth();
14
15  const handleChange = (e) => {
16    setForm({ ...form, [e.target.name]: e.target.value });
17  };
18
19  const handleSubmit = (e) => {
20    e.preventDefault();
21    setLoading(true);
22    setError(null);
23    axios.post('http://localhost:5000/api/auth/login', form)
24      .then(res => {
25        login(res.data.user, res.data.token);
26        setLoading(false);
27        navigate('/'); // Redirect to home or dashboard
28      })
29      .catch((err) => {
30        setError(err.response?.data?.error || 'Login failed');
31        setLoading(false);
32      });
33  };
34 }
```

Register:

```
Register.jsx X
client > src > pages > Register.jsx > Register
7   const Register = () => {
8     const [form, setForm] = useState({ name: '', email: '', password: '' });
9     const [loading, setLoading] = useState(false);
10    const [error, setError] = useState(null);
11    const [success, setSuccess] = useState(false);
12
13    const handleChange = (e) => {
14      setForm({ ...form, [e.target.name]: e.target.value });
15    };
16
17    const handleSubmit = (e) => {
18      e.preventDefault();
19      setLoading(true);
20      setError(null);
21      setSuccess(false);
22      axios.post('http://localhost:5000/api/auth/register', form)
23        .then(() => {
24          setSuccess(true);
25          setLoading(false);
26          setForm({ name: '', email: '', password: '' });
27        })
28        .catch((err) => {
29          setError(err.response?.data?.error || 'Registration failed');
30          setLoading(false);
31        });
32    };
33  };
34  }
```

logout:

```
36   const logout = () => {
37     localStorage.removeItem('user');
38     localStorage.removeItem('token');
39     setUser(null);
40   };
41 }
```

All Products (User):

In the home page, we'll fetch all the products available in the platform along with the filters.

Fetching products:

```
ProductDetails.jsx X
client > src > pages > ProductDetails.jsx > ...
8  const ProductDetails = () => {
9    const { id } = useParams();
10   const { addToCart } = useCart();
11   const [product, setProduct] = useState(null);
12   const [loading, setLoading] = useState(true);
13   const [error, setError] = useState(null);
14   const [added, setAdded] = useState(false);
15   const [userProfile, setUserProfile] = useState(null);
16
17   useEffect(() => {
18     axios.get(`http://localhost:5000/api/products/${id}`)
19       .then(res => {
20         setProduct(res.data);
21         setLoading(false);
22       })
23       .catch(err => {
24         setError('Product not found.');
```

In the backend, we fetch all the products and then filter them on the client side.

```
JS products.js X
server > routes > JS products.js > ...
20
27  // GET all products
28  router.get('/', async (req, res) => {
29    try {
30      const products = await Product.find(filter).sort(sort);
31      res.json(products);
32    } catch (err) {
33      res.status(500).json({ error: err.message });
34    }
35  });
36
```

Filtering products:

client > src > components > SearchAndFilters.jsx > [1] SearchAndFilters

```
1  import React, { useState, useEffect } from 'react';
2  import '../styles/SearchAndFilters.css';
3
4  const SearchAndFilters = ({ onFiltersChange, initialFilters = {} }) => {
5    const [filters, setFilters] = useState({
6      search: '',
7      category: '',
8      minPrice: '',
9      maxPrice: '',
10     sortBy: '',
11     ...initialFilters
12   });
13
14   const [showAdvancedFilters, setShowAdvancedFilters] = useState(false);
15   const [categories, setCategories] = useState([]);
16
17   // Fetch unique categories from products
18   useEffect(() => {
19     fetch('http://localhost:5000/api/products')
20       .then(res => res.json())
21       .then(products => {
22         const uniqueCategories = [...new Set(products.map(p => p.category))].filter(Boolean);
23         setCategories(uniqueCategories);
24       })
25       .catch(err => console.error('Error fetching categories:', err));
26   }, []);
27
28   // Debounced search effect
29   useEffect(() => {
30     const timeoutId = setTimeout(() => {
31       onFiltersChange(filters);
32     }, 300); // 300ms delay
33
34     return () => clearTimeout(timeoutId);
35   }, [filters, onFiltersChange]);
36
37   const handleFilterChange = (key, value) => {
38     setFilters(prev => ({ ...prev, [key]: value }));
39   };
40
41   const clearFilters = () => {
42     const clearedFilters = {
43       search: '',
44       category: '',
45       minPrice: '',
46       maxPrice: '',
47       sortBy: ''
48     };
49     setFilters(clearedFilters);
50     onFiltersChange(clearedFilters);
51   };
```

Add product to cart:

Here, we can add the product to the cart or can buy directly.

```
ProductDetails.jsx
client > src > pages > ProductDetails.jsx > ...
8  const ProductDetails = () => {
45
46    const handleAddToCart = () => {
47      addToCart(product);
48      setAdded(true);
49      setTimeout(() => setAdded(false), 1500);
50    };
51
52    const handlePlaceOrder = async () => {
53      let city = userProfile?.city;
54      let paymentMethod = userProfile?.paymentMethod;
55      const token = localStorage.getItem('token');
56      if (!city || !paymentMethod) {
57        city = window.prompt('Enter your city:');
58        if (!city) return;
59        paymentMethod = window.prompt('Enter payment method (Online or Cash on Delivery):');
60        if (!paymentMethod || ![ 'Online', 'Cash on Delivery' ].includes(paymentMethod)) {
61          alert('Invalid payment method. ');
62          return;
63        }
64        // Save to backend
65        try {
66          await axios.patch('http://localhost:5000/api/users/me', { city, paymentMethod }, {
67            headers: { Authorization: `Bearer ${token}` },
68          });
69          setUserProfile((prev) => ({ ...prev, city, paymentMethod }));
70        } catch (err) {
71          alert('Failed to save user details. ');
72          return;
73        }
74      }
75      try {
76        await axios.post('http://localhost:5000/api/orders', {
77          products: [{ product: product._id || product.id, quantity: 1 }],
78          total: product.discountedPrice || product.price,
79          city,
80          paymentMethod,
81        }, {
82          headers: { Authorization: `Bearer ${token}` },
83        });
84        alert('Order placed successfully!');
85      } catch (err) {
86        alert(err.response?.data?.error || 'Order failed');
87      }
88    };

```

· Backend: In the backend, if we want to buy, then with the address and payment method, we process buying. If we need to add the product to the cart, then we add the product details along with the user Id to the cart collection.

Buy product:

```
orders.js x
server > routes > orders.js > ...
33 // Place order (user)
34 router.post('/', auth, async (req, res) => {
35   try {
36     const { products, total, city, paymentMethod } = req.body;
37     if (!products || !Array.isArray(products) || products.length === 0) {
38       return res.status(400).json({ error: 'No products' });
39     }
40     if (!city || !paymentMethod) {
41       return res.status(400).json({ error: 'City and payment method are required' });
42     }
43     const order = new Order({
44       user: req.user.id,
45       products,
46       total,
47       city,
48       paymentMethod,
49     });
50     await order.save();
51     res.status(201).json(order);
52   } catch (err) {
53     res.status(500).json({ error: err.message });
54   }
55 });
56
```

Add product to cart:

```
server.js x
server > server.js > ...
42
43 app.post('/', async (req, res) => {
44   try {
45     const { name, price, image, description, category } = req.body;
46     const product = new Product({ name, price, image, description, category });
47     await product.save();
48     res.status(201).json(product);
49   } catch (err) {
50     res.status(400).json({ error: err.message });
51   }
52 });
```

Order products:

Now, from the cart, let's place the order

- Frontend

```

Cart.jsx
client > src > pages > Cart.jsx > ...
8   const Cart = () => {
9
10  }
11
12  const handlePlaceOrder = async () => {
13    let city = userProfile?.city;
14    let paymentMethod = userProfile?.paymentMethod;
15    const token = localStorage.getItem('token');
16    if (!city || !paymentMethod) {
17      city = window.prompt('Enter your city:');
18      if (!city) return;
19      paymentMethod = window.prompt('Enter payment method (Online or Cash on Delivery):');
20      if (!paymentMethod || ![ 'Online', 'Cash on Delivery' ].includes(paymentMethod)) {
21        alert('Invalid payment method. ');
22        return;
23      }
24    }
25    // Save to backend
26    try {
27      await axios.patch('http://localhost:5000/api/users/me', { city, paymentMethod }, {
28        headers: { Authorization: `Bearer ${token}` },
29      });
30      setUserProfile((prev) => ({ ...prev, city, paymentMethod }));
31    } catch (err) {
32      alert('Failed to save user details. ');
33      return;
34    }
35  }
36
37  setLoading(true);
38  setError(null);
39  setSuccess(false);
40  try {
41    await axios.post(
42      'http://localhost:5000/api/orders',
43      {
44        products: cartItems.map(item => ({ product: item.id || item.id, quantity: item.quantity })),
45        total,
46        city,
47        paymentMethod,
48      },
49      {
50        headers: { Authorization: `Bearer ${token}` },
51      }
52    );
53    setSuccess(true);
54    setLoading(false);
55    alert('Order placed successfully!');
56    window.location.reload();
57  } catch (err) {
58    setError(err.response?.data?.error || 'Order failed');
59    setLoading(false);
60    alert(err.response?.data?.error || 'Order failed');
61  }
62  };

```

- Backend

In the backend, on receiving the request from the client, we then place the order for the products in the cart with the specific user Id.

```

orders.js
server > routes > orders.js > ...
33 // Place order (user)
34 router.post('/', auth, async (req, res) => {
35   try {
36     const { products, total, city, paymentMethod } = req.body;
37     if (!products || !Array.isArray(products) || products.length === 0) {
38       return res.status(400).json({ error: 'No products' });
39     }
40     if (!city || !paymentMethod) {
41       return res.status(400).json({ error: 'City and payment method are required' });
42     }
43     const order = new Order({
44       user: req.user.id,
45       products,
46       total,
47       city,
48       paymentMethod,
49     });
50     await order.save();
51     res.status(201).json(order);
52   } catch (err) {
53     res.status(500).json({ error: err.message });
54   }
55 });
56

```

Add new product:

Here, in the admin dashboard, we will add a new product.

o Frontend:

```
ProductForm.jsx X
client > src > pages > ProductForm.jsx > ProductForm > handleChange
7   const initialState = {
8     name: '',
9     price: '',
10    image: '',
11    description: '',
12    category: '',
13  };
14
15  const ProductForm = () => {
16    const { id } = useParams();
17    const isEdit = Boolean(id);
18    const [form, setForm] = useState(initialState);
19    const [imageFile, setImageFile] = useState(null);
20    const [loading, setLoading] = useState(false);
21    const [error, setError] = useState(null);
22    const navigate = useNavigate();
23
24    useEffect(() => {
25      if (isEdit) {
26        setLoading(true);
27        axios.get(`http://localhost:5000/api/products/${id}`)
28          .then(res => {
29            setForm({
30              name: res.data.name,
31              price: res.data.price,
32              image: res.data.image,
33              description: res.data.description,
34              category: res.data.category,
35            });
36            setLoading(false);
37          })
38          .catch(() => {
39            setError('Error loading product');
40            setLoading(false);
41          });
42      }
43    }, [id, isEdit]);
```

o Backend:

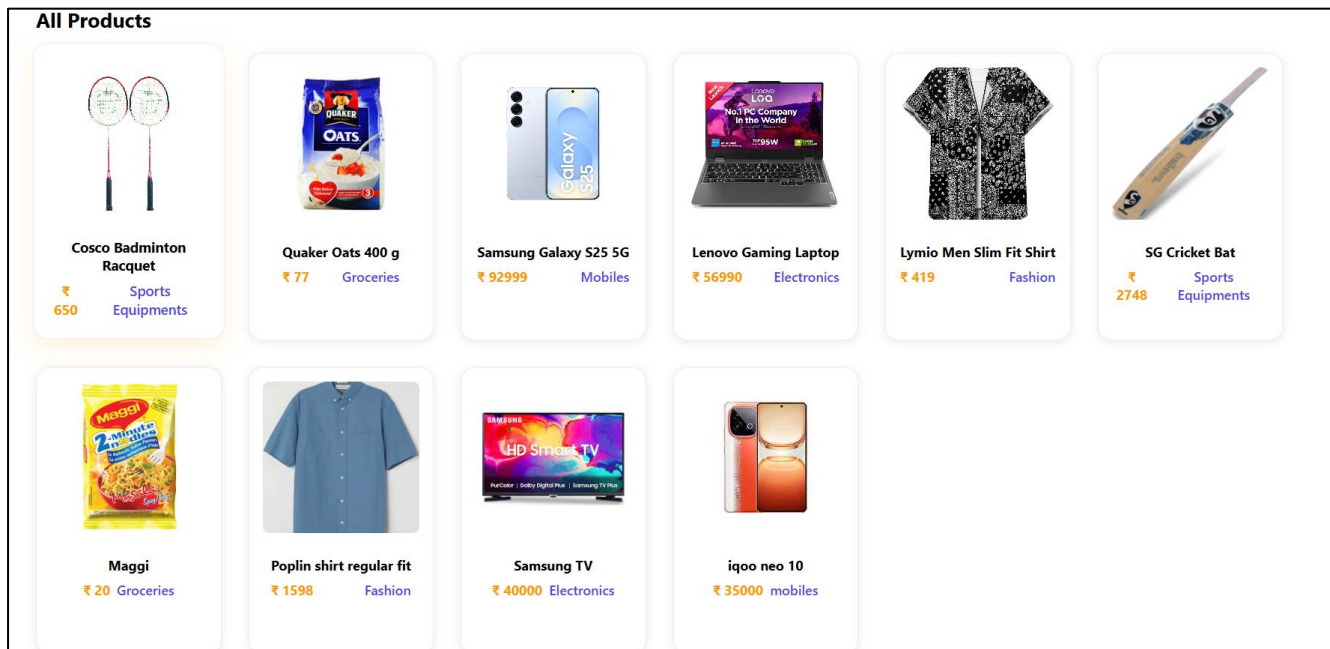
```
products.js X
server > routes > products.js > router.get("/") callback
92
93  // POST create new product (with image upload)
94  router.post('/', upload.single('image'), async (req, res) => {
95    try {
96      const { name, price, description, category } = req.body;
97      let imageUrl = req.body.image;
98      if (req.file) {
99        imageUrl = '/uploads/' + req.file.filename;
100      }
101      const product = new Product({ name, price, image: imageUrl, description, category });
102      await product.save();
103      res.status(201).json(product);
104    } catch (err) {
105      res.status(400).json({ error: err.message });
106    }
107  });
```


Demo UI images:




- Landing page



- Products



· Authentication




ShopEZ  [Login](#)  

[< Back](#)

Register

[Register](#)

Already have an account? [Login](#)

ShopEZ  [Login](#)  

[< Back](#)

Login

[Login](#)

Don't have an account? [Register](#)

User Orders

ShopEZ

Search Electronics, Fashion, mobiles, etc.,

Q

Hi, user

My Orders

Logout

← Back

Your Orders

Order ID: 685e2e6aabd5504254bb8248

Date: 6/27/2025, 11:08:50 AM

Status: Pending

Total: ₹ 1598

Products:

• Poplin shirt regular fit x 1

Cancel Order

Order ID: 685e2e8babd5504254bb8256

Date: 6/27/2025, 11:09:23 AM

Status: Pending

Total: ₹ 2748

Products:

• SG Cricket Bat x 1

Cancel Order

Cart

ShopEZ

Search Electronics, Fashion, mobiles, etc.,

Q



Hi, user

My Orders

Logout

← Back

Your Cart

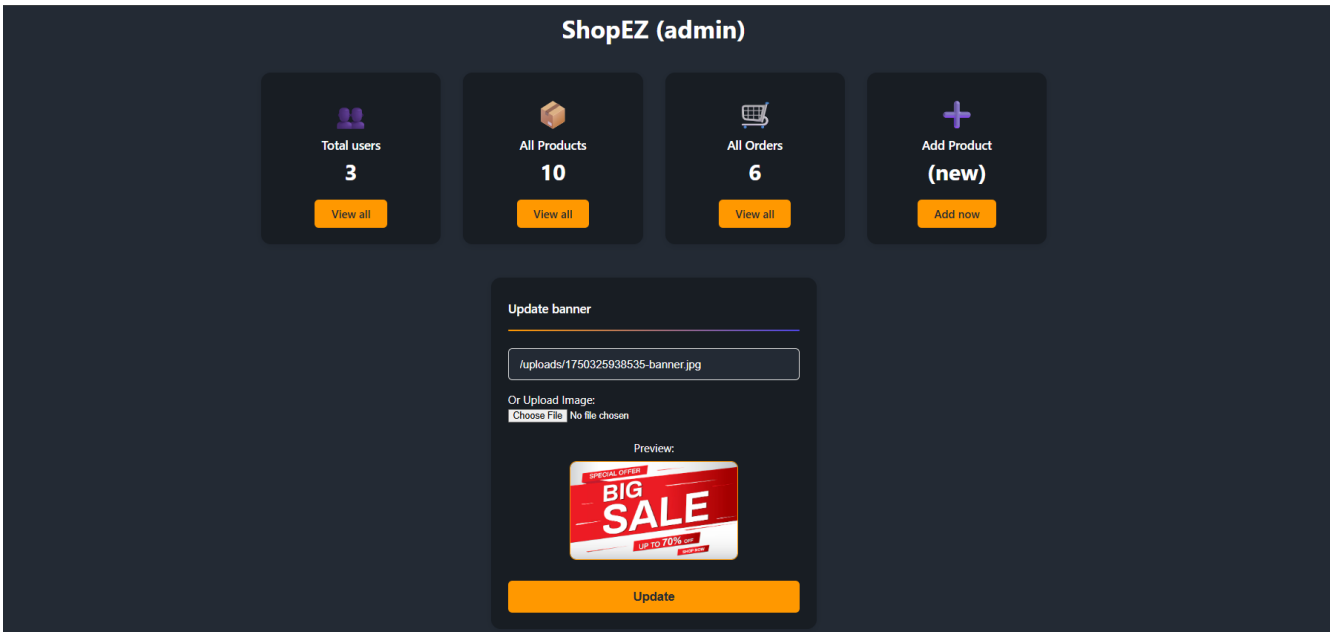
Product	Price	Quantity	Total	
 Lenovo Gaming Laptop	₹ 56990	1	₹ 56990	<div>Remove</div>
 iqoo neo 10	₹ 35000	1	₹ 35000	<div>Remove</div>

Total: ₹ 91990

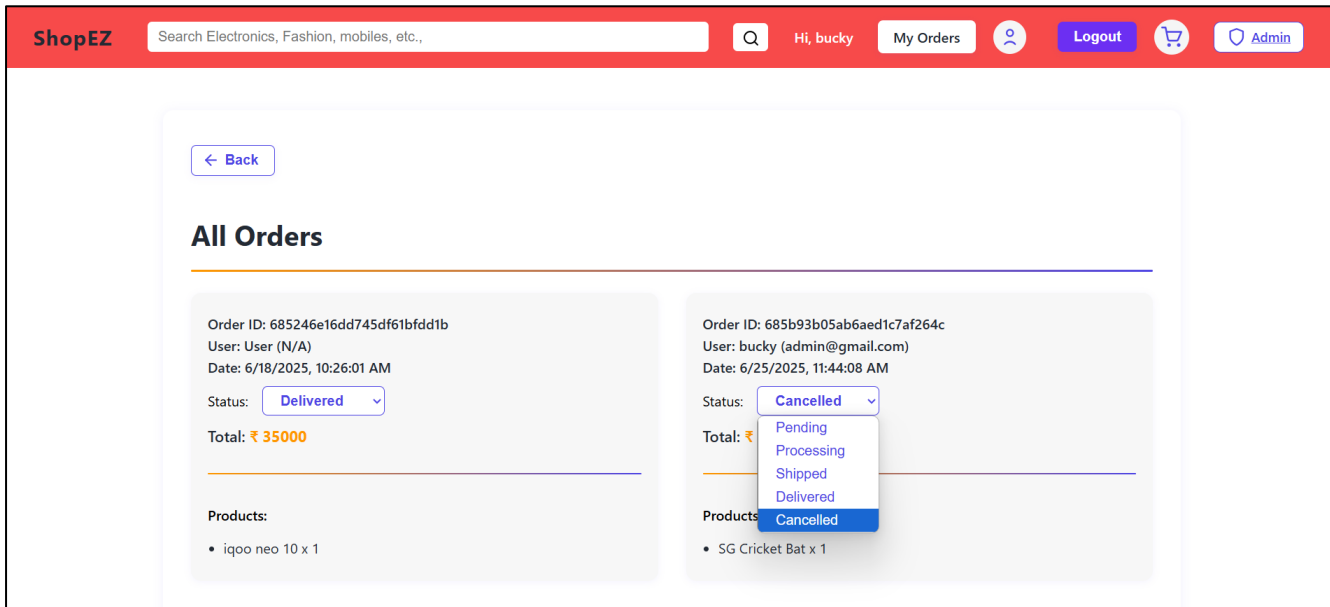
Change Address/Payment

Place Order

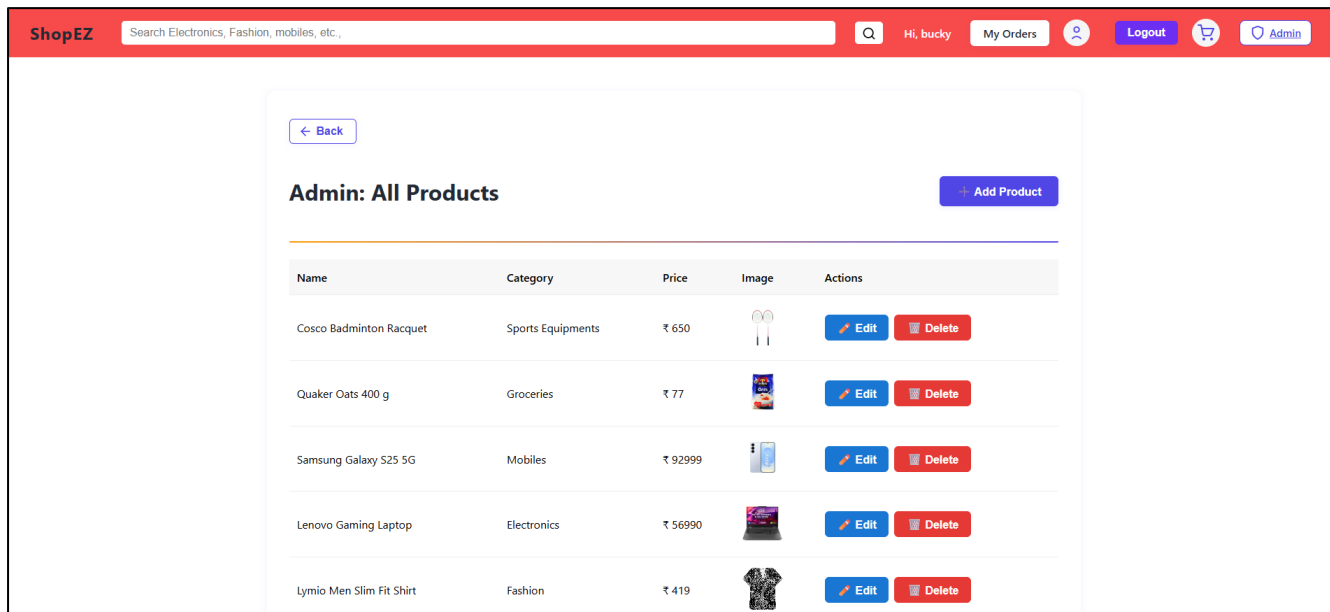
Admin dashboard



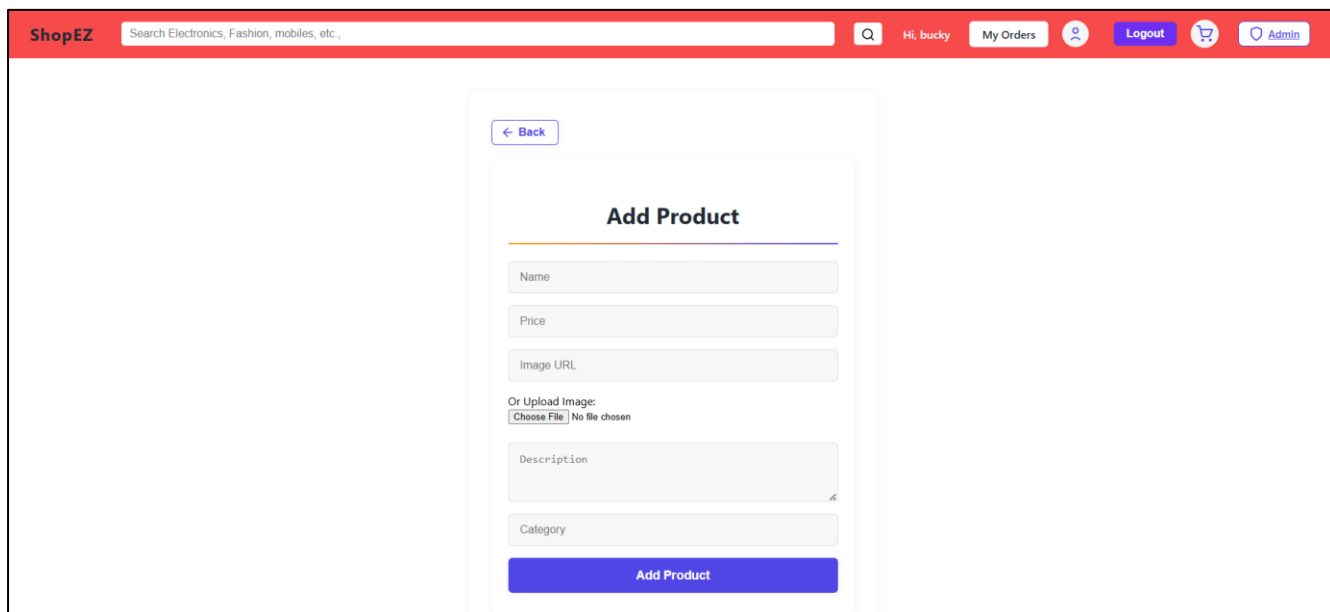
All Orders



All Products



New Product Page



For any further doubts or help, please consider the drive,

<https://drive.google.com/drive/folders/13GfDZKAYzKzK9pwMStaaVUEgY3kNY9CO?usp=sharing>

The demo of the app is available at:

https://drive.google.com/file/d/1LXRrIHypSGso5S6fwyv3yYW7xmyQsm_/view?usp=sharing

TESTING

Manual testing done for all flows: register, login, cart, order, admin panel

Testing Scope

Features and Functionalities to be Tested:

- User registration and login
- Product browsing and search
- Cart management
- Order placement and order history
- Admin product/user/order management

User Stories/Requirements to be Tested:

- USN-1: User registration
- USN-2: Registration confirmation
- USN-3: User login
- USN-4: Product browsing/search
- USN-5: Cart and order placement
- USN-6: Admin management

KNOWN ISSUES

- Product images might not appear if uploads folder is empty or missing
- Basic error messages need better UX design
- No support for password reset yet

FUTURE ENHANCEMENTS

- Integrate Stripe or Razorpay for payment gateway
- Allow product reviews and star ratings
- Enable email notifications on order status
- Multi-language support
- Add accessibility features and dark mode