**FULL STACK DEVELOPMENT WITH MERN PROJECT DOCUMENTATION**

**1. Introduction**

**Project Title** : ShopEZ – One-Stop Shop for Online Purchases

**Team id** : LTVIP2025TMID55709

**Team Members:**

1. **Team Leader:**

Vulli Yuthika – Full Stack Developer & Project Coordinator  
Responsible for overall planning, coordination, GitHub management, and integration of frontend and backend.

1. **Team Member**:

Veeroji Kiran– Frontend Developer  
Works on the React-based UI, handles component design, page routing, and user interactions.

1. **Team Member**:

Varshitha Talasila– Backend Developer  
Builds RESTful APIs using Node.js and Express.js, manages authentication and server logic.

1. **Team Member:**

Velchuri Mahesh– Database Administrator  
Designs and manages MongoDB schemas, handles CRUD operations and ensures data consistency.

**2. Project Overview**

**• Purpose:**

To provide a seamless online shopping experience where users can browse, search, and purchase products, with an intuitive interface and secure order handling.

**• Features:**

* Browse and filter products by category, price, and popularity
* Add items to cart and checkout securely
* User authentication and order tracking
* Admin dashboard to manage products, categories, and orders
* Email confirmation upon successful purchase

**3. Architecture**

**• Frontend:**

Developed using **React.js** with modular components and dynamic routing using **React Router**. Styled using **Tailwind CSS** or **Material-UI** for responsive design.

**Components:**

* **ProductList.js**: Displays product catalog with filters
* **ProductDetails.js**: Shows detailed view of a selected product
* **Cart.js**: Displays current items in the user's shopping cart
* **Checkout.js**: Collects payment and address details
* **AdminPanel.js**: Interface for managing products and orders

**State Management:**

* **Redux Toolkit** or **Context API** used for maintaining global state such as cart items, user session, and product data

**• Backend:**

Powered by **Node.js** and **Express.js**, exposing a RESTful API to manage users, products, carts, and orders.

**API Routes:**

javascript

CopyEdit

// Product Routes

app.get('/api/products', productController.getAll);

app.get('/api/products/:id', productController.getOne);

app.post('/api/products', adminMiddleware, productController.create);

// Cart & Order Routes

app.post('/api/cart', authMiddleware, cartController.addToCart);

app.post('/api/orders', authMiddleware, orderController.create);

**Middleware:**

* **JWT Authentication** for secure login and access control
* **Rate Limiting** and **CORS Handling** for security and performance
* **Validation** of input data using libraries like express-validator

**• Database:**

**MongoDB** is used to store user accounts, product details, orders, and cart data. Managed with **Mongoose** for schema modeling and validation.

**Schemas:**

javascript

CopyEdit

// User Schema

const UserSchema = new Schema({

email: { type: String, unique: true },

password: String,

cart: [{ type: Schema.Types.ObjectId, ref: 'Product' }],

orders: [{ type: Schema.Types.ObjectId, ref: 'Order' }]

});

// Product Schema

const ProductSchema = new Schema({

name: String,

category: String,

price: Number,

stock: Number,

description: String,

imageUrl: String

});

// Order Schema

const OrderSchema = new Schema({

user: { type: Schema.Types.ObjectId, ref: 'User' },

products: [{ product: Schema.Types.ObjectId, quantity: Number }],

totalAmount: Number,

status: { type: String, default: 'Pending' },

createdAt: { type: Date, default: Date.now }

});

**4. Setup Instructions**

**Prerequisites:**

* Node.js >= 18
* MongoDB installed and running
* npm or yarn

**Installation:**

* + git clone https://github.com/your-username/FlightFinder.git
  + cd FlightFinder
  + cd server
  + npm install
  + cd ../client
  + npm install

**Environment Variables:**  
Create a .env file in the /server folder with:

PORT=3000

MONGODB URI-mongodb+srv://nehapriya:<db\_password>@cluster0.zghkpqk.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

**5. Folder Structure**

* **Client:**

/client

├── /src

│ ├── /components

│ ├── /pages

│ ├── /api

│ ├── App.js

│ └── index.js

* Server:

/server

├── /controllers

├── /routes

├── /models

├── server.js

└── .env

**5.Running the Application**

**Frontend:**

* cd client
* npm start

**Backend:**

* cd server
* npm start

**7. API Documentation**

**• POST /api/auth/register**

Registers a new user.  
**Request Body:**

json

CopyEdit

{ "name": "John Doe", "email": "john@example.com", "password": "securePass123" }

**Response:**

json

CopyEdit

{ "success": true, "token": "<jwt-token>" }

**• POST /api/auth/login**

Logs in an existing user.  
**Request Body:**

json

CopyEdit

{ "email": "john@example.com", "password": "securePass123" }

**Response:**

json

CopyEdit

{ "success": true, "token": "<jwt-token>" }

**• GET /api/products**

Retrieves a list of all available products.  
**Response:**

json

CopyEdit

[

{

"\_id": "1",

"name": "Wireless Mouse",

"price": 999,

"category": "Electronics",

"imageUrl": "/images/mouse.jpg"

},

...

]

**• POST /api/cart**

Adds a product to the user’s cart (requires login).  
**Request Body:**

json

CopyEdit

{ "productId": "1", "quantity": 2 }

**Response:**

json

CopyEdit

{ "success": true, "message": "Product added to cart" }

**• POST /api/orders**

Places an order for the logged-in user.  
**Request Body:**

json

CopyEdit

{ "cartItems": [ { "productId": "1", "quantity": 2 } ] }

**Response:**

json

CopyEdit

{ "success": true, "orderId": "ORD123456" }

**8. Authentication**

* **JWT (JSON Web Token)** is used to authenticate users.
* Tokens are securely stored in **HTTP-only cookies** to prevent XSS attacks.
* Protected API routes (like /api/orders, /api/cart) **validate the JWT** on every request.
* Admin-specific routes require an **admin role** to access product management endpoints.

**9. User Interface**

**• Home Page:**

* Displays a featured product section and main categories
* Search bar for quickly locating items
* User login/register buttons in navbar

**• Product Listing Page:**

* Grid of products with filtering (category, price range, rating)
* Option to add items directly to cart

**• Cart Page:**

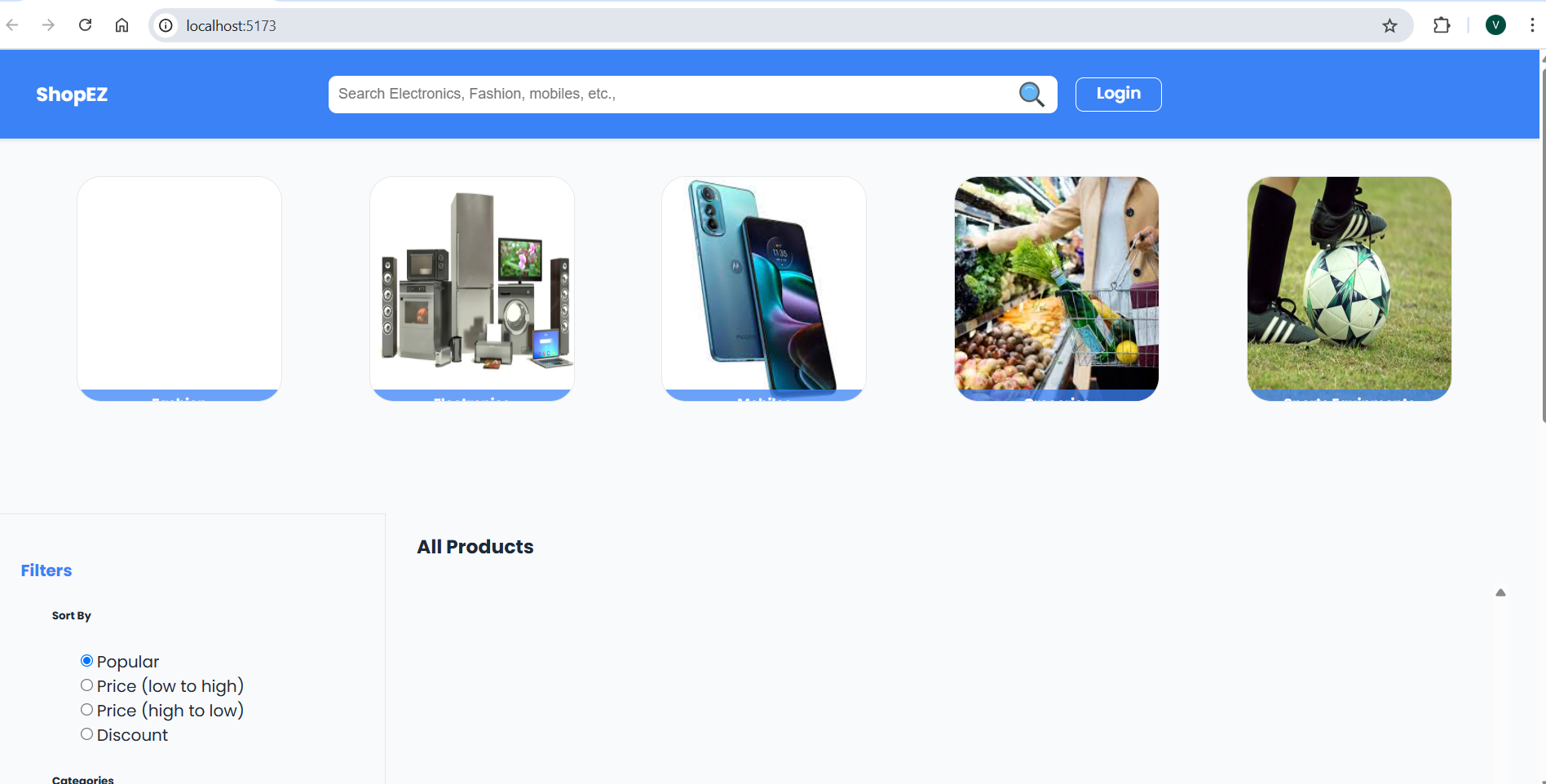
* Shows added items with quantity and price
* Options to update quantity or remove item

**• Checkout Page:**

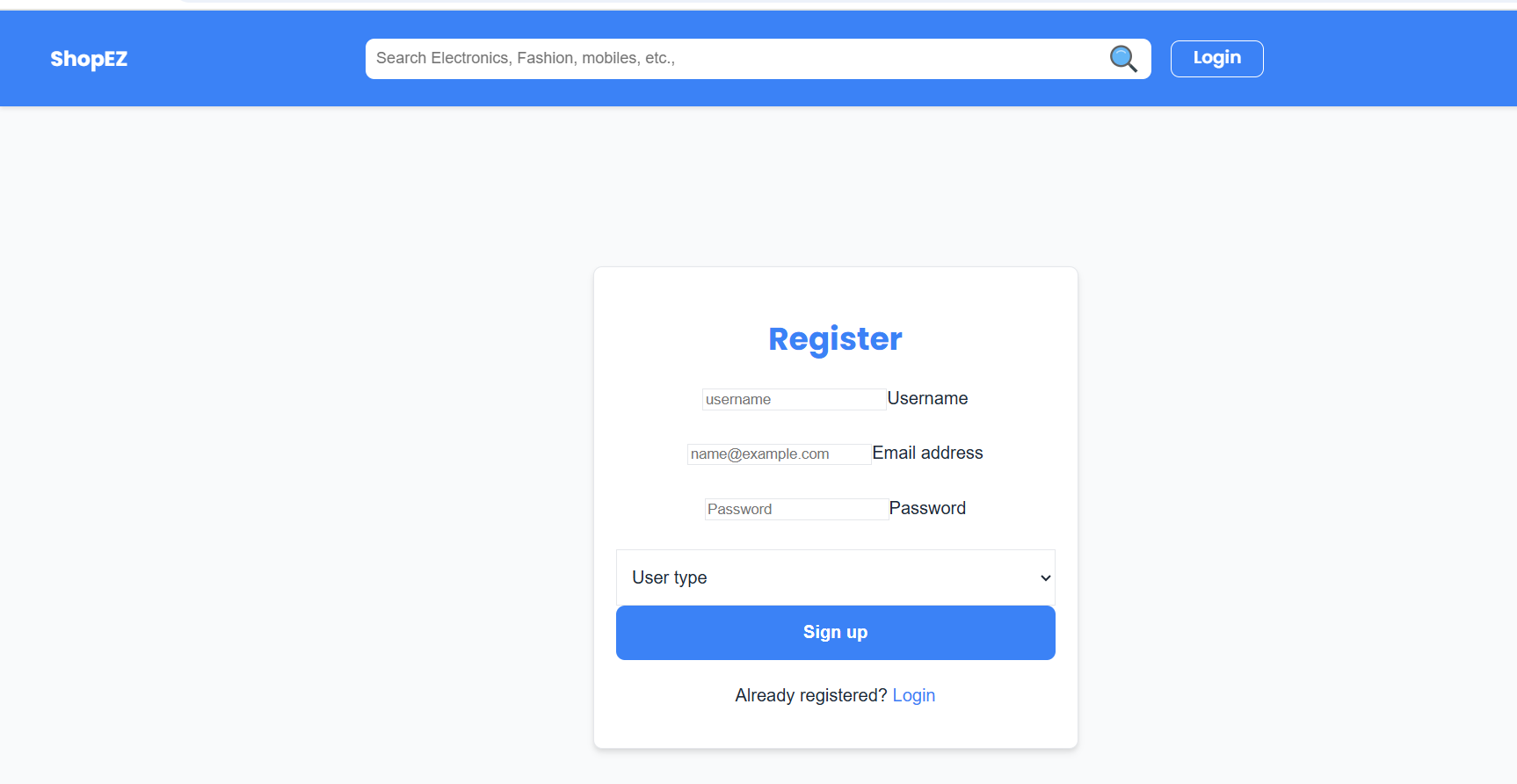
* Form for entering shipping and payment details
* Order summary and confirmation

**• Admin Panel:**

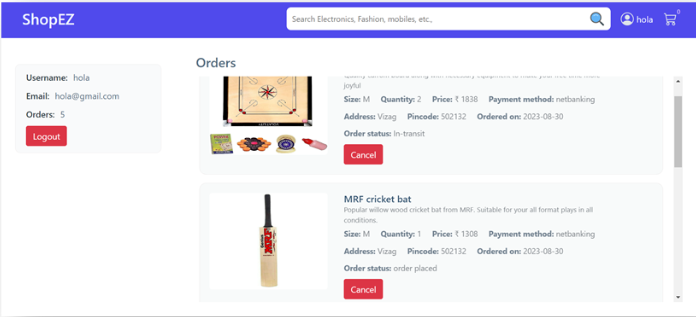
* Dashboard with total sales, order history, and product inventory
* Forms for adding/editing products

****

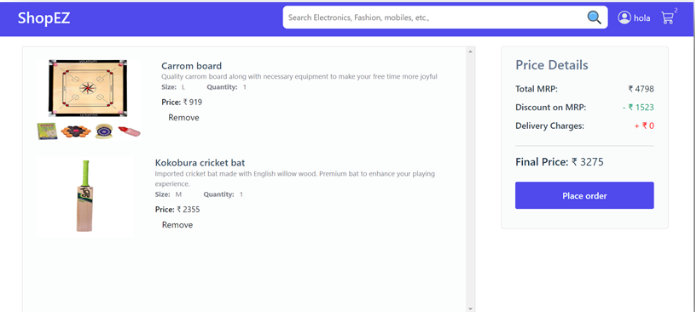
**❖ Registration**

****

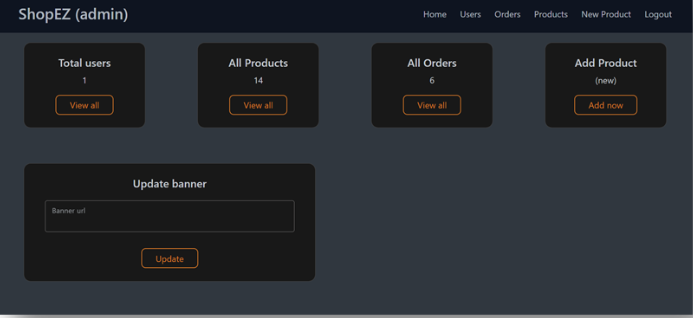
**❖ Dashboard**

****

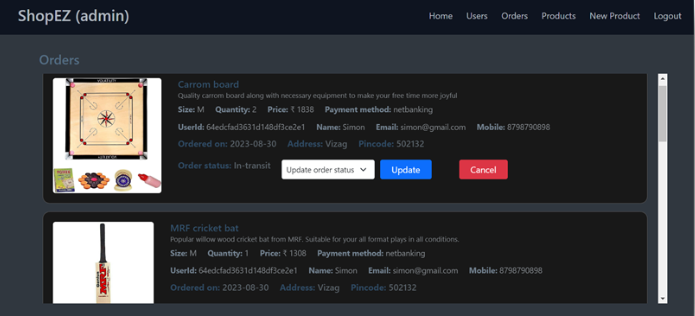
**❖ Cat**

****

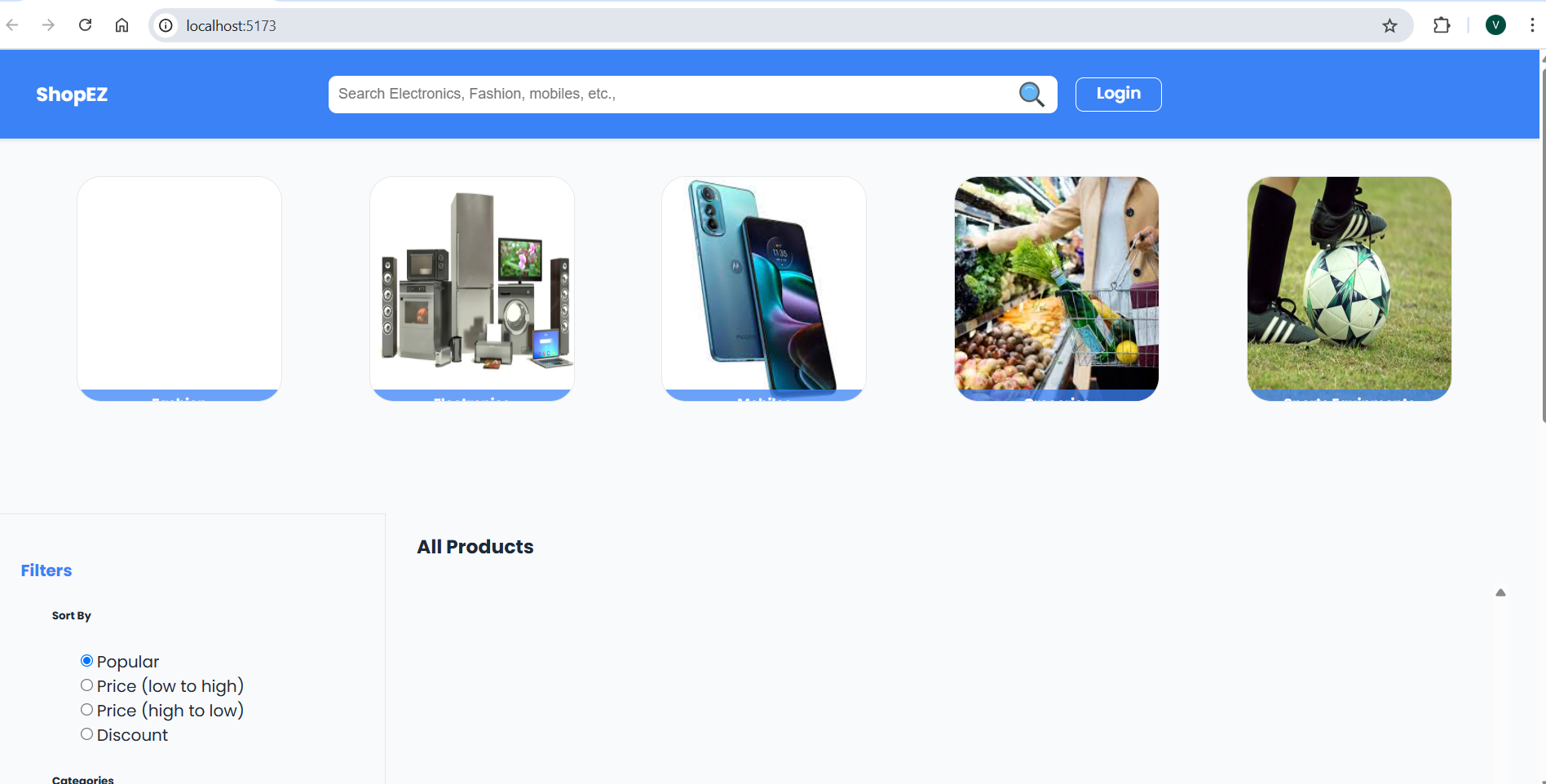
**❖ Admin dashboard**

****

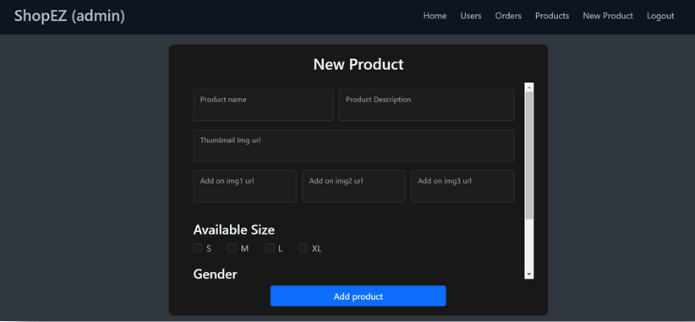
**❖ All Orders**

****

**❖All Products**

****

**❖New Product Page**

****

**10. Testing**

* **Manual Testing:**  
  All backend API endpoints were tested using **Postman** to verify request/response structure, authorization, and error handling.
* **Frontend Testing:**  
  Basic component-level testing implemented using **Jest** and **React Testing Library**.  
  Tests cover form validations, cart interactions, and conditional rendering.

**11. Github**

**Link**

https://github.com/varshitha3121/shopesy

**12. Known Issues**

* **Password recovery** functionality is not yet implemented.
* **Mobile and tablet responsiveness** is inconsistent on certain screen sizes.
* **Product image compression** is not optimized for large uploads.

**13. Future Enhancements**

* **Online Payment Integration:**  
  Integration with payment gateways like **Razorpay** or **Stripe** for smooth transaction handling.
* **Role-Based Access Control:**  
  Differentiated admin and user permissions for better management and security.
* **Inventory Alerts & Notifications:**  
  Real-time alerts for low stock or successful order placement via email/SMS.
* **Progressive Web App (PWA):**  
  Offline capabilities and push notifications for improved user engagement.