

# Shopez – MERN Full Stack E-Commerce

## 1. Introduction

**Project Title:** Shoppez – MERN Full Stack E-Commerce

Shopez is a full-stack e-commerce web application developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js). The project provides a complete online shopping platform with secure authentication, product browsing, cart management, order processing, and an admin dashboard.

## 2. Project Overview

The Shoppez application follows a modular and scalable technical architecture to ensure high performance, security, and ease of maintenance

## 3. Technical Architecture

The Shoppez application follows a scalable and modular architecture to ensure performance, security, and maintainability. The system is divided into frontend, backend, API services, and database layers.

### User Interface (UI) :

The frontend is built using React.js and styled with Tailwind CSS. Users can browse products, search by category, view product details, add items to cart, and place orders. The UI is fully responsive and optimized for all devices.

### Web Server :

The backend server is built using Node.js and Express.js to handle HTTP requests and business logic.

### Authentication Service

Authentication is implemented using JWT (JSON Web Tokens). Passwords are encrypted using bcrypt, ensuring secure access.

### Database :

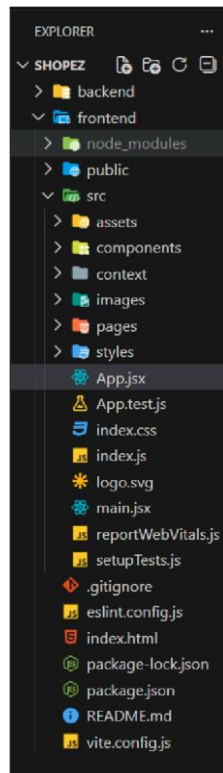
MongoDB stores user information, product details, cart items, and order data.

## 4. Features

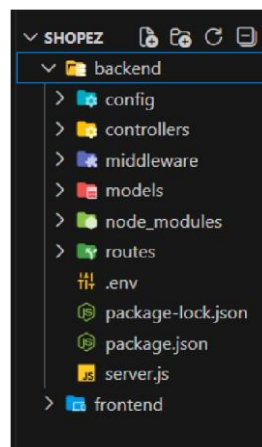
- User registration and secure login
- Product browsing with category filtering
- Add to cart and checkout functionality
- Order placement and order tracking
- Admin dashboard for managing products and orders
- Role-based authentication
- Responsive UI design

## 5.Folder Structure

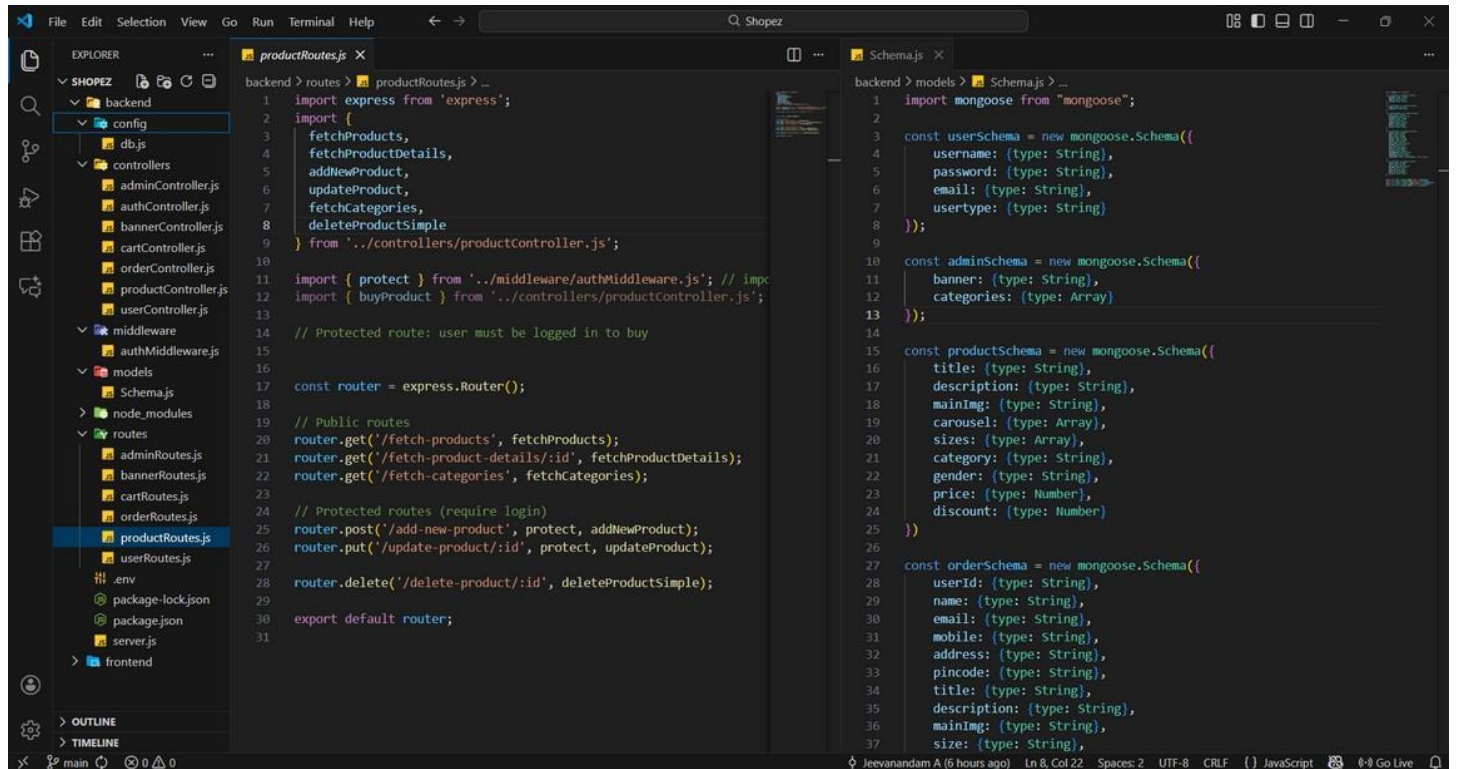
**Client:** React frontend.



- **Server:** Node.js backend.



## 6.API Documentation



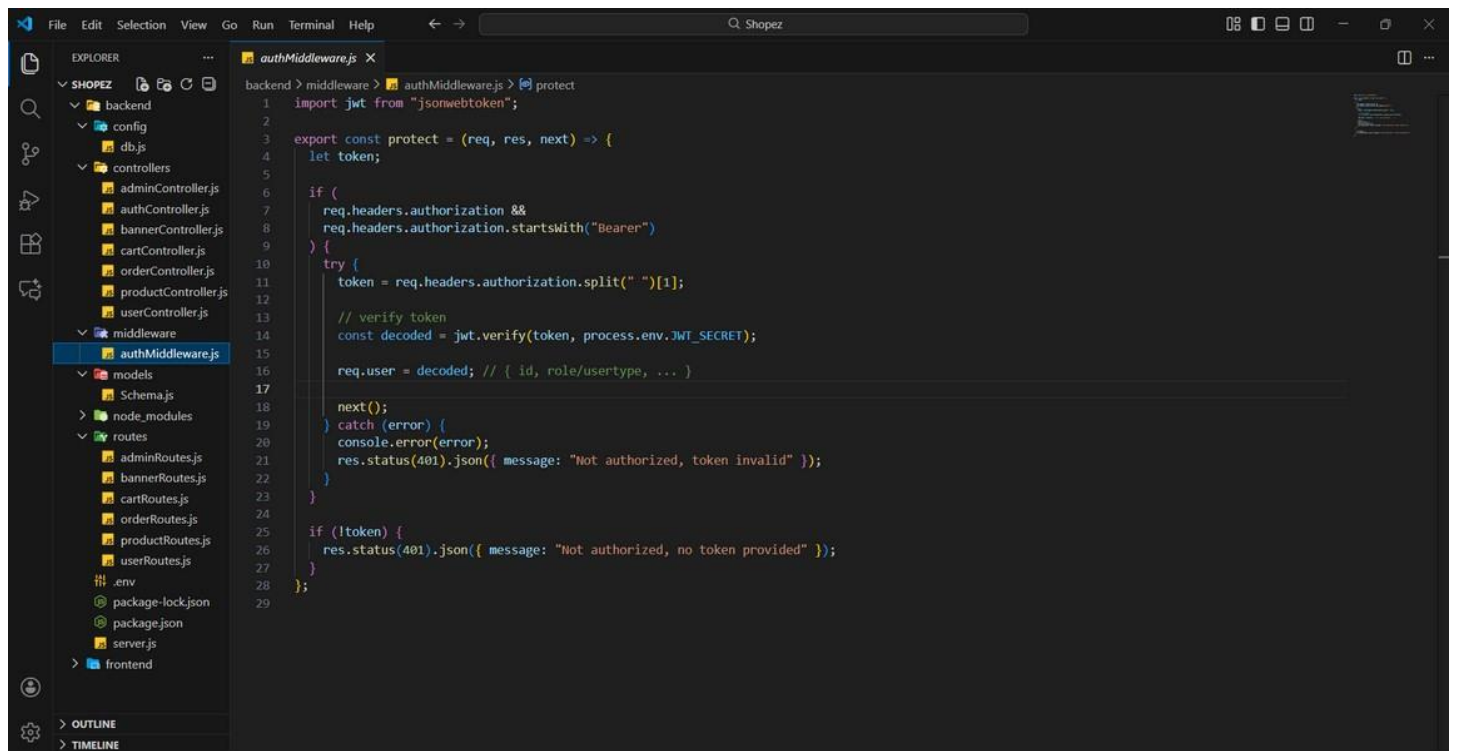
The screenshot shows a VS Code editor with two files open: `productRoutes.js` and `Schema.js`. The `productRoutes.js` file defines routes for a product management system, including endpoints for fetching products, details, categories, adding new products, updating products, and deleting products. The `Schema.js` file defines Mongoose schemas for users, admin, products, and orders. The Explorer panel on the left shows the project structure, including `backend`, `config`, `controllers`, `middleware`, `models`, `node_modules`, `routes`, and `frontend`.

```
productRoutes.js
1 import express from 'express';
2 import {
3   fetchProducts,
4   fetchProductDetails,
5   addNewProduct,
6   updateProduct,
7   fetchCategories,
8   deleteProductSimple
9 } from '../controllers/productController.js';
10
11 import { protect } from '../middleware/authMiddleware.js'; // import
12 import { buyProduct } from '../controllers/productController.js';
13
14 // Protected route: user must be logged in to buy
15
16 const router = express.Router();
17
18 // Public routes
19 router.get('/fetch-products', fetchProducts);
20 router.get('/fetch-product-details/:id', fetchProductDetails);
21 router.get('/fetch-categories', fetchCategories);
22
23 // Protected routes (require login)
24 router.post('/add-new-product', protect, addNewProduct);
25 router.put('/update-product/:id', protect, updateProduct);
26 router.delete('/delete-product/:id', deleteProductSimple);
27
28 export default router;
```

```
Schema.js
1 import mongoose from "mongoose";
2
3 const userSchema = new mongoose.Schema({
4   username: {type: String},
5   password: {type: String},
6   email: {type: String},
7   usertype: {type: String}
8 });
9
10 const adminSchema = new mongoose.Schema({
11   banner: {type: String},
12   categories: {type: Array}
13 });
14
15 const productSchema = new mongoose.Schema({
16   title: {type: String},
17   description: {type: String},
18   mainImg: {type: String},
19   carousel: {type: Array},
20   sizes: {type: Array},
21   category: {type: String},
22   gender: {type: String},
23   price: {type: Number},
24   discount: {type: Number}
25 });
26
27 const orderSchema = new mongoose.Schema({
28   userId: {type: String},
29   name: {type: String},
30   email: {type: String},
31   mobile: {type: String},
32   address: {type: String},
33   pincode: {type: String},
34   title: {type: String},
35   description: {type: String},
36   mainImg: {type: String},
37   size: {type: String},
38 });
```

## 7.Authentication

Authentication uses JWT stored in cookies for secure sessions. Authorization ensures only logged-in users can access protected routes while admin routes require an admin role. Passwords are securely hashed using bcrypt. Middleware validates tokens, checks user roles, and prevents unauthorized access.

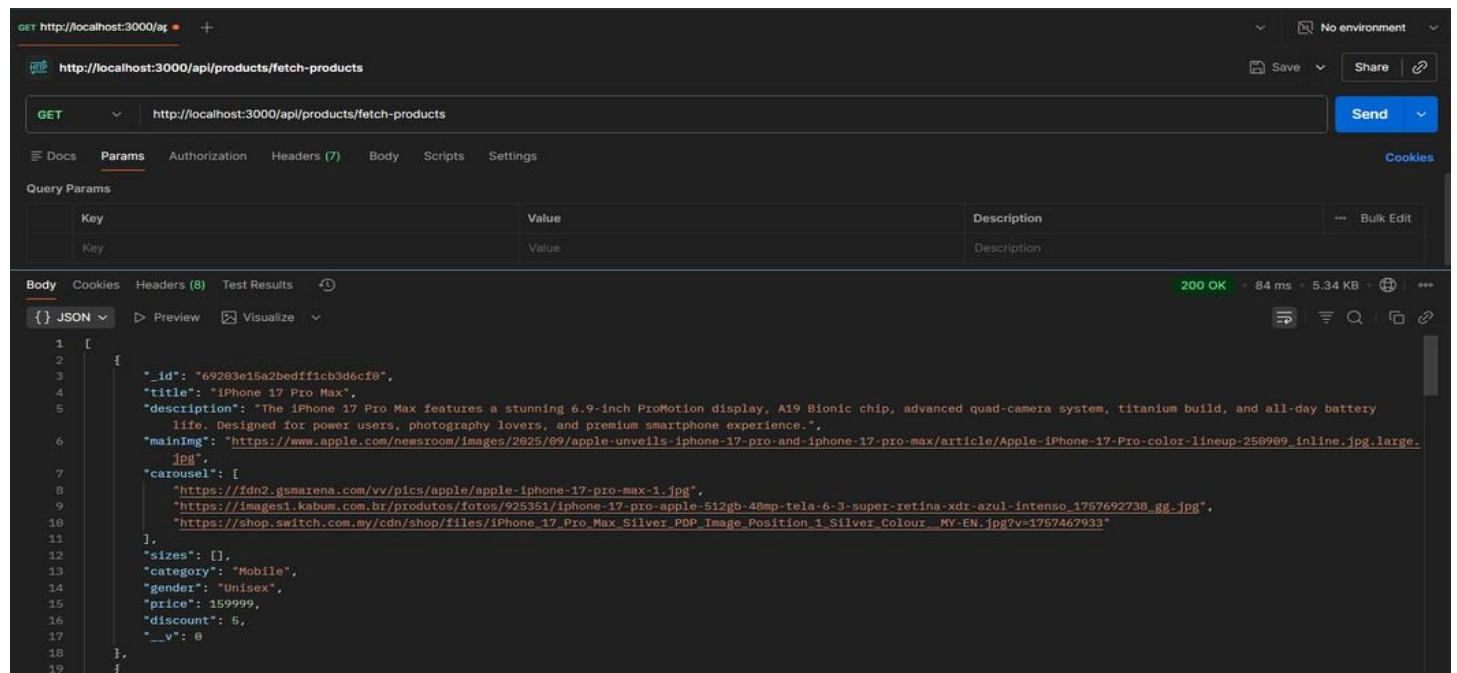


The screenshot shows a VS Code editor with the `authMiddleware.js` file open. The file defines a `protect` middleware function that checks for a valid JWT token in the request headers. If the token is valid, it decodes it and sets the user information on the request object. If the token is invalid or missing, it returns a 401 status with an error message. The Explorer panel on the left shows the project structure, including `backend`, `config`, `controllers`, `middleware`, `models`, `node_modules`, `routes`, and `frontend`.

```
authMiddleware.js
1 import jwt from "jsonwebtoken";
2
3 export const protect = (req, res, next) => {
4   let token;
5
6   if (
7     req.headers.authorization &&
8     req.headers.authorization.startsWith("Bearer")
9   ) {
10     try {
11       token = req.headers.authorization.split(" ")[1];
12
13       // verify token
14       const decoded = jwt.verify(token, process.env.JWT_SECRET);
15
16       req.user = decoded; // { id, role/usertype, ... }
17
18       next();
19     } catch (error) {
20       console.error(error);
21       res.status(401).json({ message: "Not authorized, token invalid" });
22     }
23   }
24
25   if (!token) {
26     res.status(401).json({ message: "Not authorized, no token provided" });
27   }
28 };
29
```

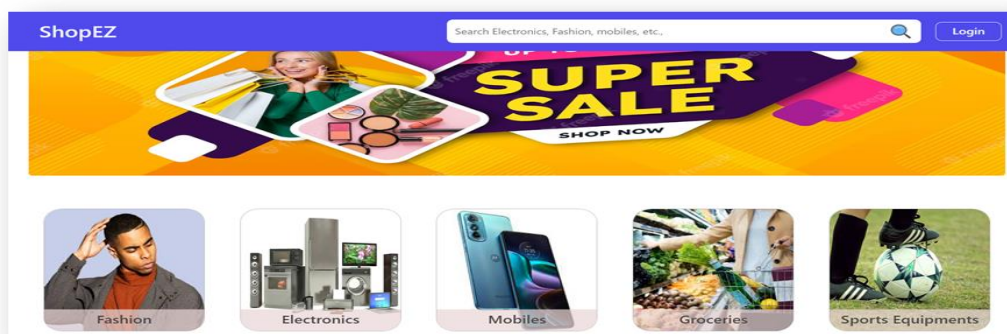
## 8. Testing

Testing is done using Postman for backend APIs and browser dev tools for frontend validation. Console logs, error boundaries, and custom middleware help identify issues. Manual testing ensures function reliability and user experience.

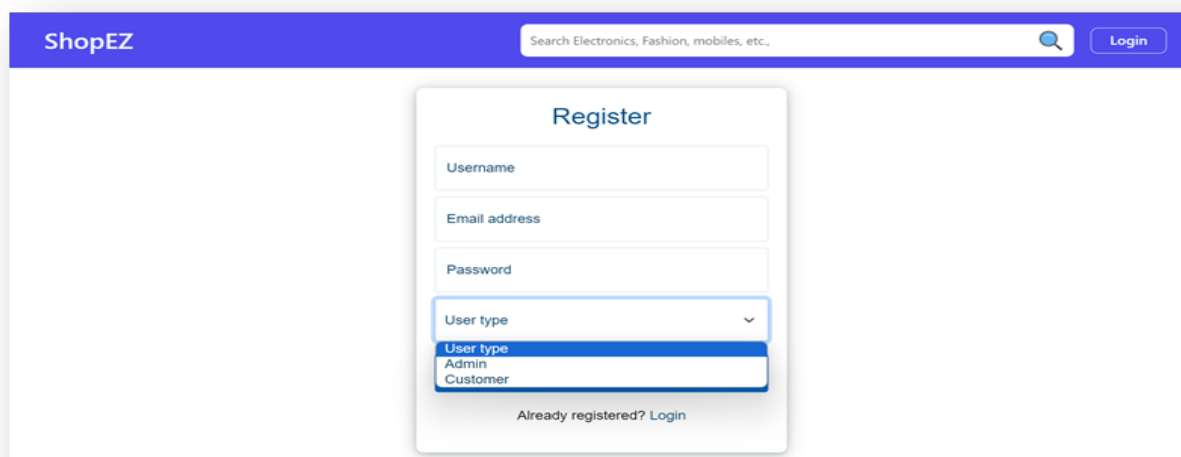


## 9. Screenshots or Demo

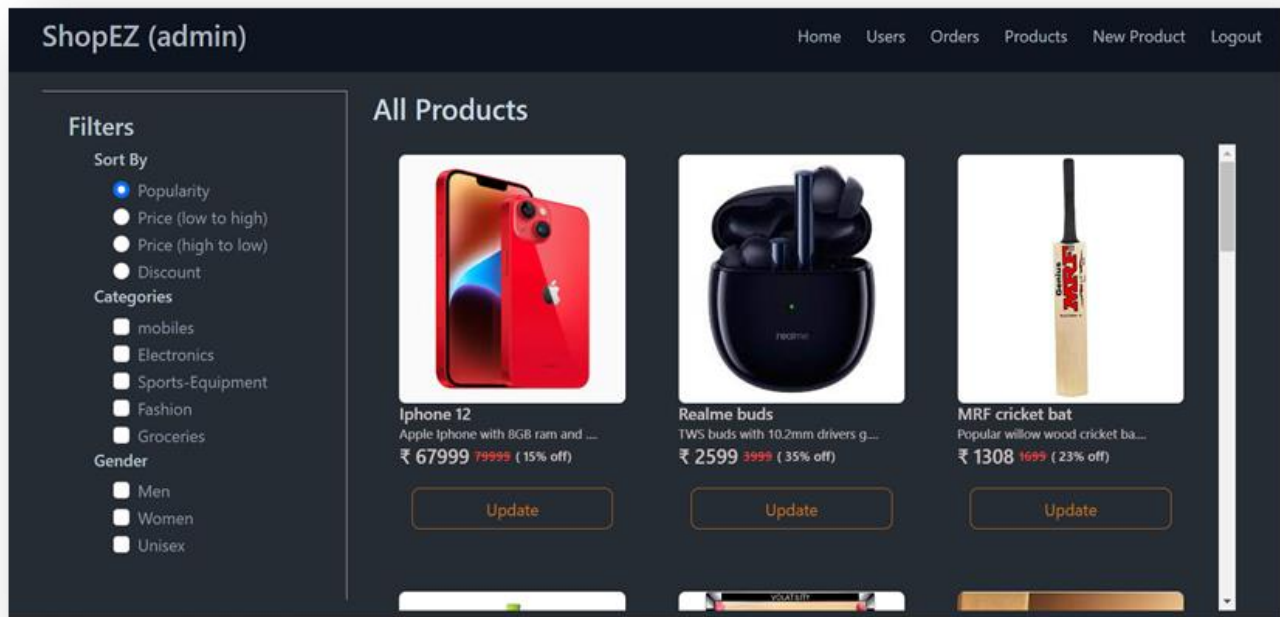
### Admin dashboard



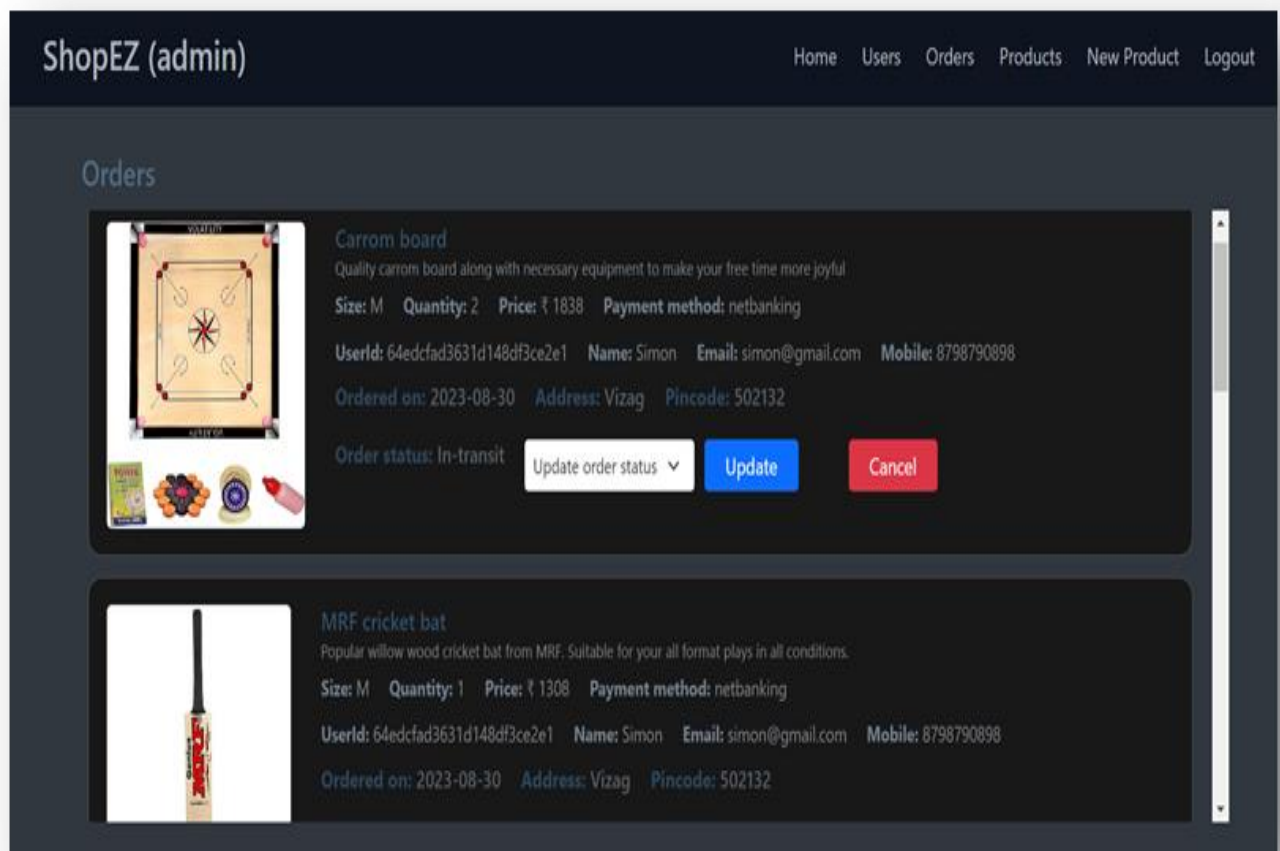
### Authentication Page



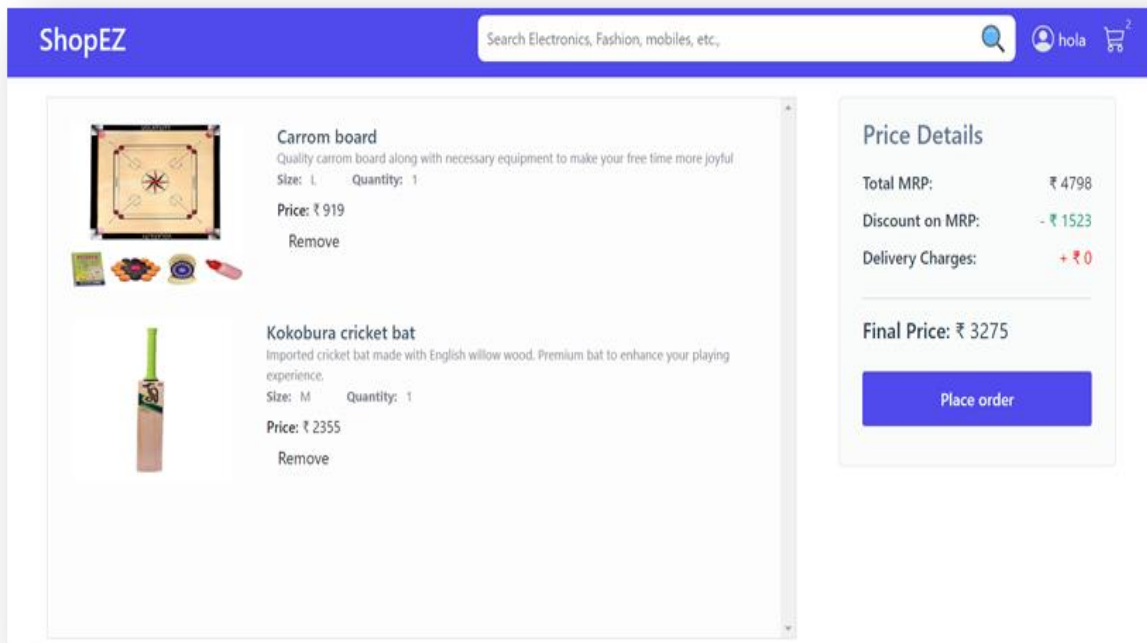
## All Products Page



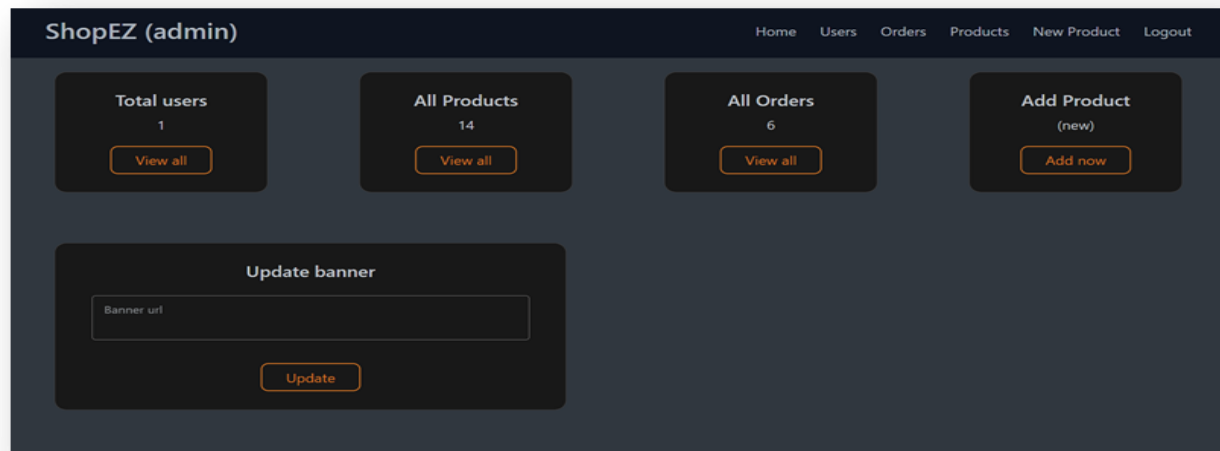
## Cart Page



## Orders Page



## Admin page



## 10.Known Issues

- Payment gateway not integrated yet
- Email notifications pending
- Pagination for admin dashboard can be improved
- UI optimization needed for very small screens

## 11.Future Enhancements

- Add online payment gateway (Razorpay/Stripe)
- Add wish list feature
- Add delivery tracking
- Add analytics dashboard for admin
- Add email + SMS notifications
- Add product comparison
- Add AI-based product recommendations

## 12.Conclusion

Shopez demonstrates a real-world implementation of a MERN stack e-commerce application. The project highlights secure authentication, responsive UI design, and efficient backend architecture, making it suitable for academic submission.