# BATCH-ID  LTVIP2025TMID59619

**1.Introduction**

**• Project Title:**

Shopez: One-Stop Shop For Online Purchases

**• Team Members:**

* Ummadi Lakshmi Aravind Reddy– Team Leader
* Syamala Uma Maheswara Reddy– Team Member
* Tammisetty Lakshmi Prasanna Kumar– Team Member
* Surangi Gnanesh– Team Member

**2. Application Flow**

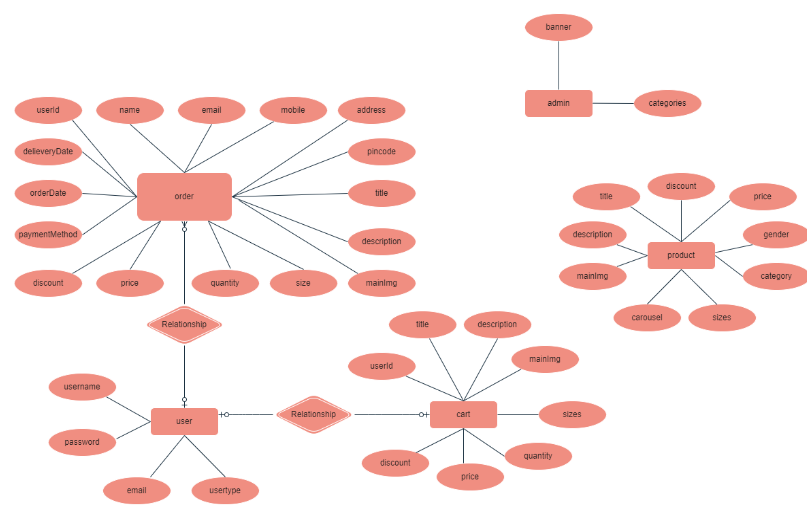
**• Purpose:**

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.

**• Features:**

* **User Registration & Login**: Users can sign up or log in securely using email and password. Authentication ensures personalized experiences and secure access.
* **Product Listing**: Display of products with images, names, prices, and brief descriptions.Pagination for large catalogs.
* **Smart Search & Filters:** Quickly find desired products using keyword search or filters like categories, price, or ratings. Enhances the user’s ability to find preferred products.
* **Quick Checkout Process:** Users can enter their name, delivery address, and payment info in a few steps. The simplified form minimizes ordering time.
* **Payment Integration:** Supports multiple payment methods like cards, wallets, or UPI. Transactions are processed securely and instantly.
* **Order Confirmation:** Once an order is placed, users receive an instant confirmation message. It ensures peace of mind and verifies successful placement.
* **Order tracking:** Users will be able to track their order status in real-time. This adds transparency and reduces uncertainty.
* **Secure Authentication :** User data is protected with secure login and session handling. Maintains platform integrity and user trust.

**3. Architecture**



**Fig: - Architecture**

**Frontend (React.js)**

The frontend is responsible for the visual representation and interaction of the system. It would include:

* **User Authentication:** Forms for login and registration with role-based access (customer, admin, restaurant owner).
* **Dashboard:**  Role-specific dashboards showing order history (user), food/menu management (owner), and analytics (admin).
* **Product Exploring:** Product exploring allows users to browse, view details, and discover items available in the application.
* **Cart Management:** Option to add/remove items, update quantity, and view total price before ordering.
* **Order placement:** A checkout page with options for delivery details, payment method, and discount application.

**Backend (Express.js & Node.js)**

The backend handles the server-side operations, including:

* **Authentication:** Verifies user credentials and grants access.
* **Authorization:** Determines the permissions for each user role.
* **Order Management:** Interacts Handles order creation, delivery date tracking, payment method processing, and status updates. details.
* **Business Logic:** Implements logic for price calculation (with discounts), products and order timings.
* **Products & Accessories:** Collection of main items and their complementary add-ons available for purchase in the application.
* **Data Validation & Security:** Ensures clean, secure data handling through input validation, hashing passwords, and secure API communication.

**Database (MongoDB)**

The database stores the persistent data of the system. It would typically include:

* **Users:** Collection to store user information including name, email, password, and user role (e.g., customer, admin).
* **Orders:** Collection to store order details such as order date, delivery date, payment method, price, discount, product details, and user address.
* **Cart:** Collection to temporarily store products selected by users before placing an order. Includes item quantity, price, and associated company.
* **Products:** Collection to store product details including title, description, price, discount, rating & product ID.
* **Admins**: Collection to manage admin-related content such as promotional banners and categories(e.g., Men, Women).

**4. Setup Instructions**

**• Prerequisites:**

* **Node.js and npm:**Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server side. It provides a scalable and efficient platform for building network applications.
* **Express.js:**  Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.
* **MongoDB:**MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.
* **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.

**• Installation:**

* React
* Bootstrap
* react-bootstrap
* bcryptjs
* express
* dotenv
* mongoose
* Nodemon
* Json
* MongoDB

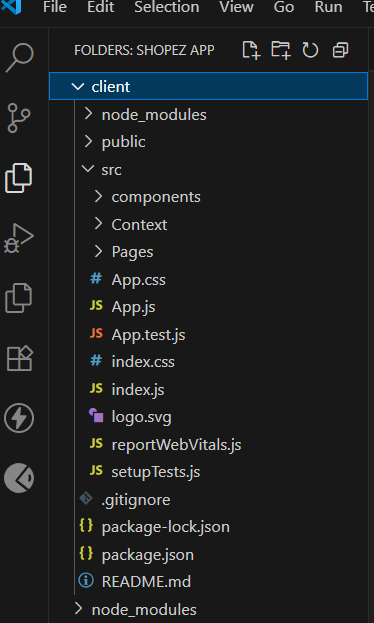
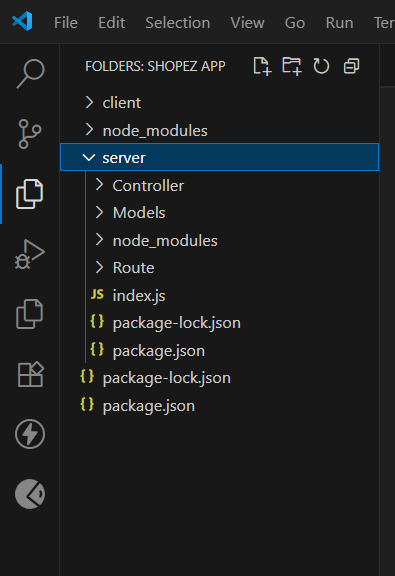
**5. Folder Structure**

**•Client:**

* **public**: contains all the npm packages installed for the React app.
* **src**: Main source folder.
  + **App.css:** The App.css file contains the styling rules that define the appearance and layout of components in a React application's App.js file.
  + **logo.svg:**  A simple and modern vector logo representing the brand.
  + **App.js:** App.js is the main component that sets up the structure and navigation of the application.
  + **Index.js**: Entry-point that renders the main React component (App) into the DOM.
  + **SetupTests.js:** setupTests.js is a configuration file that initializes the testing environment for project.
  + **reportWebVitals.js**: collects and reports performance metrics to help analyze and improve the application's user experience.
* **package.json**: Lists dependencies, scripts (like start, build), and metadata for the React app.
* **package-lock.json**: Exact version tree of installed dependencies—used to ensure consistent installs across machines.
* **README.md**: Documentation for your React project—how to install, run, or understand the app.

**• Server:**

* **config**: Configuration files, such as database and application settings.
* **controllers**: Logic for handling requests and responses.
* **models**: Database schemas and models.
* **routes**: Route definitions for the API endpoints.
* **middlewares**: Custom middleware functions.
* **schema**: Utility Contains Mongoose schema definitions for MongoDB collections (e.g., User, Book, etc.).
* **package.json**: Lists dependencies and scripts.
* **package-lock.json**: Exact version tree of installed dependencies—used to ensure consistent installs across machines.

  Fig: - Frontend Fig: - Backend

**6. Running the Application**

**Frontend:** npm start -in the client directory  **Backend:** node index.js- in the server directory.

**7. Application Flow**

    const PORT = 10000;

const express = require("express");

const app = express();

const mongoose = require("mongoose");

const jwt = require("jsonwebtoken");

// const multer = require("multer");

const path = require("path");

const cors = require("cors");

const { type } = require("os");

const { request } = require("http");

const { error } = require("console");

app.use(express.json()); //response will auto pass thorugh json

app.use(cors()); //react will connect using on the prot

app.use(

      cors({

          origin : "http://localhost:3000"

      })

)

const Users=mongoose.model('Users',{

      name:{

            type:String

      },

      password:{

            type:String

      },

      email:{

            type:String

      },

      cartData:{

            type:Object

      },

      date:{

            type:Date

      }

})

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

      let check=await Users.findOne({email:req.body.email});

      if(check){

            return res.status(400).json({success:false,errors:"existing users found with same username"})

      }

      let cart ={};

      for(let i=0;i<300;i++){

            cart[i]=0

      }

      const user=new Users({

      name:req.body.username,

      email:req.body.email,

      password:req.body.password,

      cartData:cart,

})

await user.save();

const data={

      user:{

            id:user.id

      }

}

const token=jwt.sign(data,'secret-ecom');

res.json({success:true,token})

})

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

let user=await Users.findOne({email:req.body.email});

if(user){

      const passCompare=req.body.password===user.password;

      if(passCompare){

            const data={

                  user :{

                        id:user.id

                  }

            }

            const token=jwt.sign(data,'secret\_ecom');

            res.json({success:true,token});

      }

      else{

            res.json({success:false,errors:"Wrong password"})

      }

}

else{

      res.json({success:false,errors:"Wrong email id"})

}

})

//Database connectio with mongo DB

mongoose.connect("mongodb+srv://prasanna:prasanna123@cluster0.p4jib1n.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"

)

app.listen(PORT,() => {

      try {

            console.log(`Connected to database at PORT : ${PORT}`)

      } catch (err) {

            console.error(`Error connecting to database, Error : ${err}`)

      }

})

**API Request/Response Formats**

- Request: JSON

- Response: JSON

**API Error Handling**

- Error Codes: 400, 404, 500

- Error Messages: Detailed error messages

**8. Authentication**

Explain how authentication and authorization are handled in the project.

Include details about tokens, sessions, or any other methods used.

• **Authentication:**

* User Registration: Users sign up with username, email, password, and role.
* Passwords: passwords are hashed and stored in MongoDB.
* Login: On valid credentials, a JWT token is generated and sent to the client.
* Token storage: Upon Token is saved on the frontend (e.g., localStorage) for future secure requests.

• **Authorization:**

* Role Embedding: Users are assigned roles (admin, doctor, patient).
* Access Control: Routes/features are restricted based on role (e.g., only Admins manage menus).
* Middleware Check: Role-based checks are performed before processing protected actions..

• **Token Management:**

* Token Generation: JWT is generated using user ID, role, and expiration time.
* Token Storage: JWT is stored in local storage on the client-side.
* Token Validation: JWT is validated on each request using middleware.

**9. User Interface**

• Provide screenshots showcasing login page and registration form.

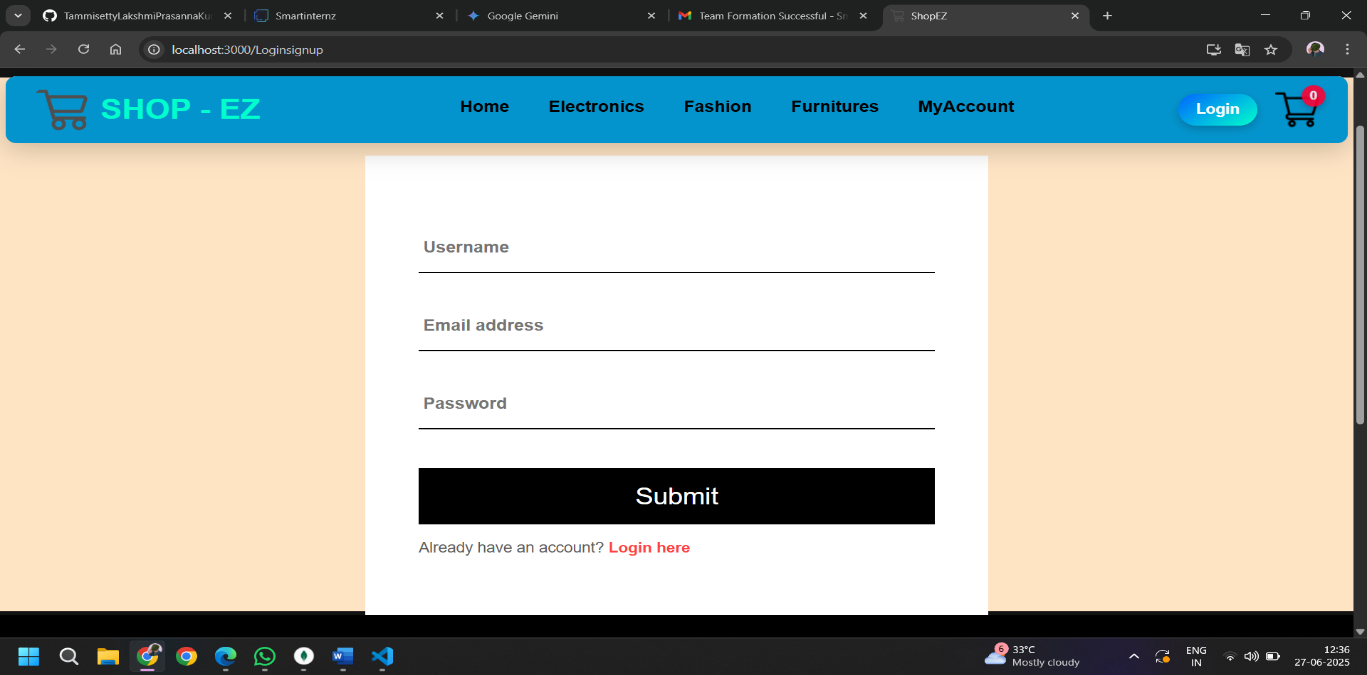
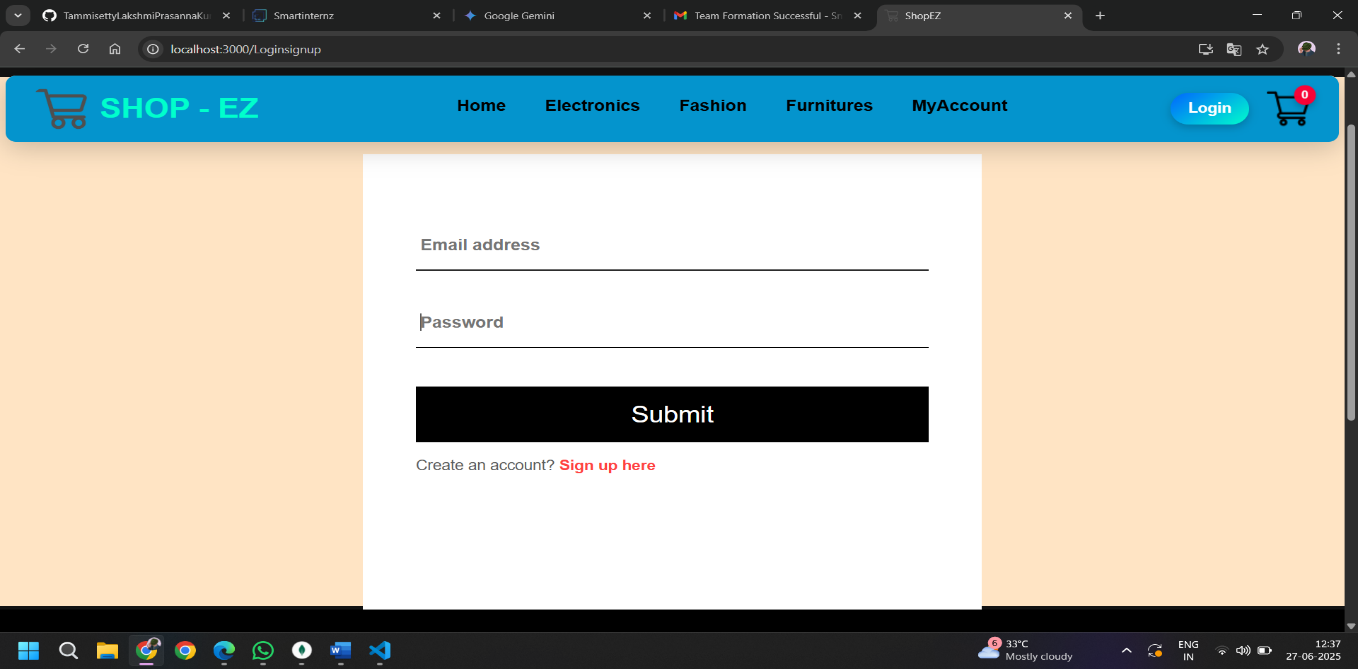


Fig: - Register Form

Fig: - Login Page

**10. Testing:**  Describe the testing strategy and tools used.

* **API Testing:** Use tools like Postman or Insomnia to test authentication (/register, /login) and protected routes with and without JWT tokens.
* **Unit Testing:** Use Jest or Mocha to write test cases for individual functions like password hashing, token generation, and role checks.
* **Integration Testing:** Test complete flows such as user registration to order placement using tools like Supertest with Express.
* **Frontend Testing:** Use React Testing Library or Jest to test form validations, component rendering, and user interactions.

**11. Screenshots or Demo**

**Project Structure**

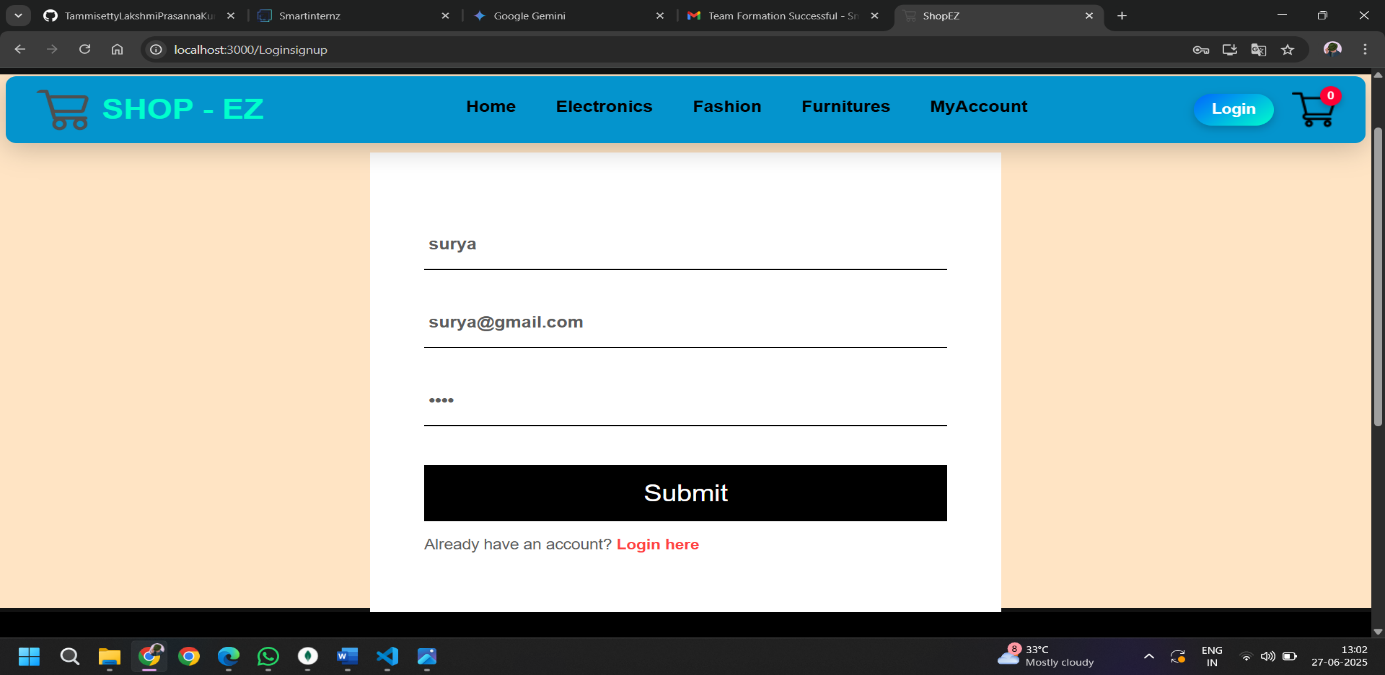


Fig: - Registration Form

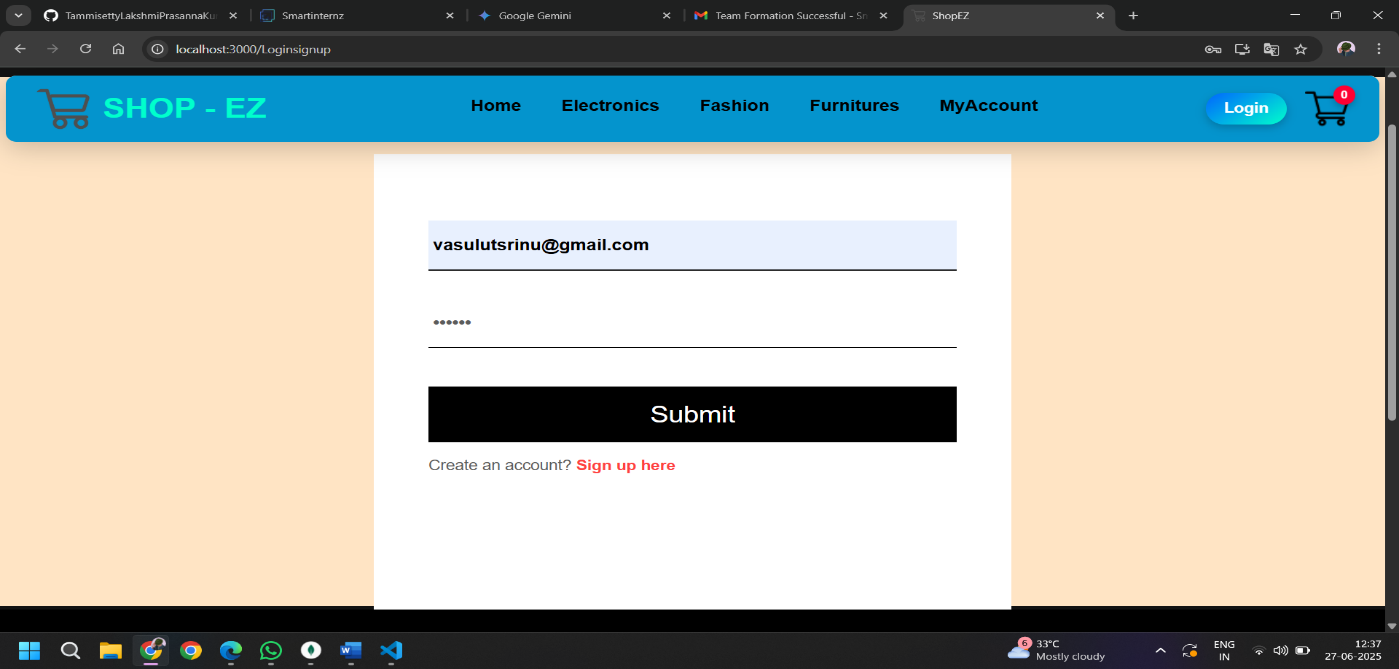


Fig: - Login Page

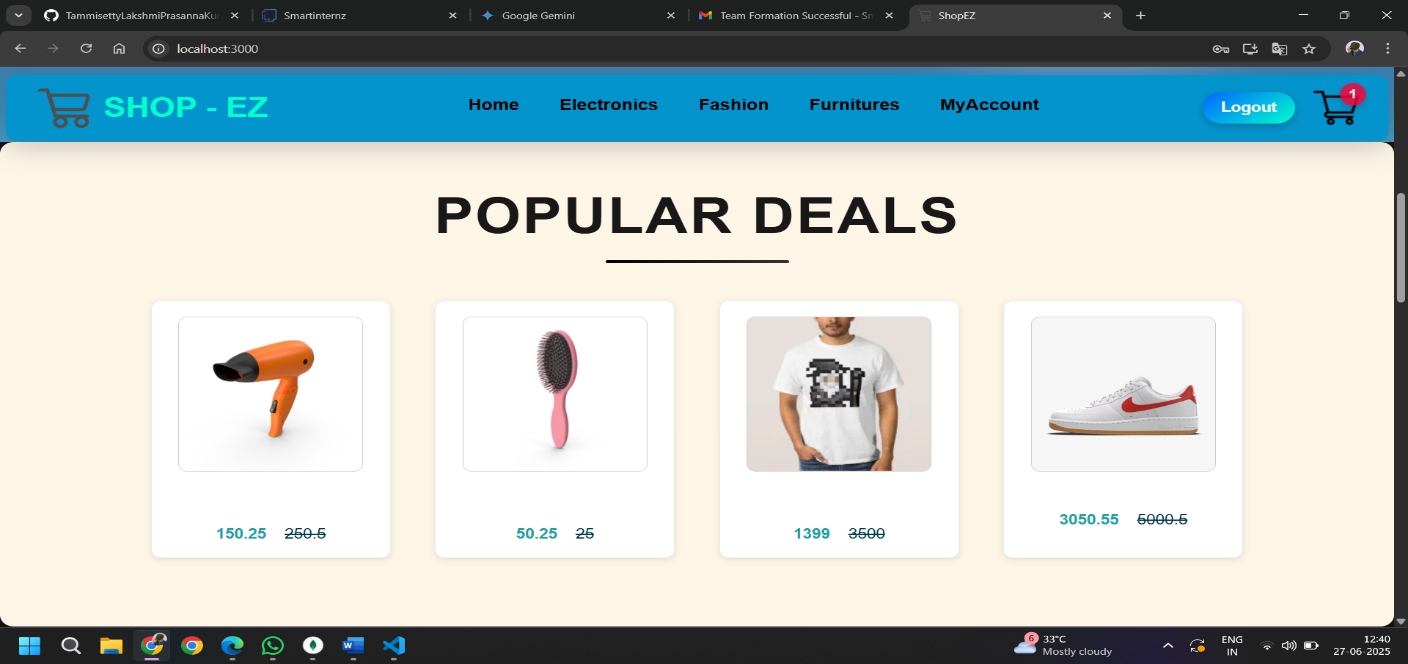


Fig: - User Interface

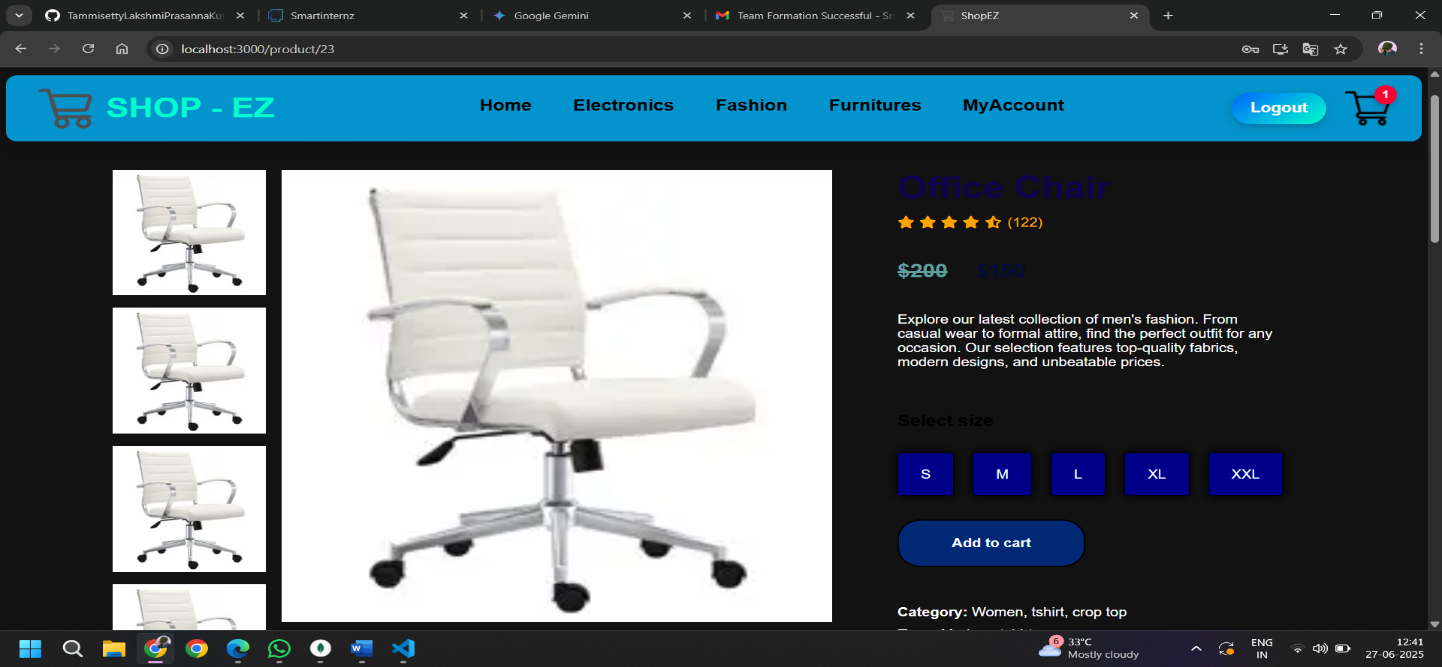


Fig: - Product Details

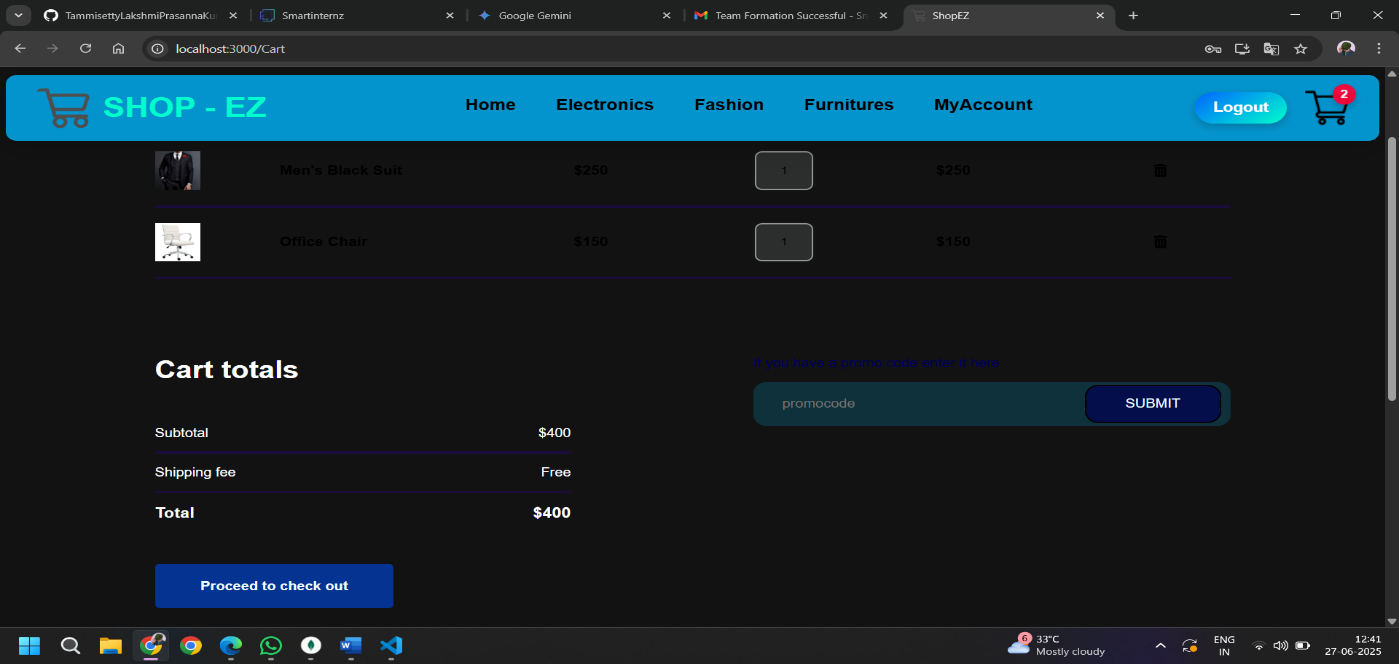


Fig: - Orders

**12. Known Issues**

Document any known bugs or issues that users or developers should be aware of.

**• Backend (Node.js):**

* Error handling: Uncaught exceptions may cause server crashes.
* MongoDB connection issues: Occasional disconnections require manual restart.
* API rate limiting: No rate limiting implemented, vulnerable to abuse.

**• Frontend (React):**

* Browser compatibility: Issues with older browser versions (IE, Safari).
* Mobile responsiveness: Layout issues on smaller screens.
* State management: Complex state updates may cause rendering errors.

**• Database (MongoDB):**

* Data consistency: Lack of transactions may lead to inconsistent data.
* Query optimization: Slow queries impact performance.
* Indexing: Missing indexes cause slow query execution.

**• Security:**

* Authentication: Weak password hashing (BCrypt).
* Authorization: Incomplete role-based access control.
* XSS protection: Insufficient input sanitization.

**13. Future Enhancements**

Outline potential future features or improvements that could be made to the project.

* Extend schema to include Orders, Cart, and User Roles (Admin, Delivery).
* Implement pagination and lazy loading to improve frontend and backend performance.
* Use React Query or SWR for efficient API calls and caching in the frontend.
* Secure inputs and hash passwords using bcrypt and input sanitization.
* Modularize codebase, set up Docker and CI/CD pipelines for smooth deployment.