**Full Stack Development with MERN**

**Project Documentation format**

**1. Introduction**

**Project Title:** Shopez e-commerce app – Mern

**Team Members**

* Sandeep S.K- Frontend
* Sathish Kumar R - Backend
* Udhaya Suriya D - Backend
* Sabari R – Frontend

**2. Project Overview**

* **Purpose:** The purpose of the Shopez e-commerce app is to provide a convenient, secure, and user-friendly platform for seamless online shopping and selling. It aims to enhance the buying experience, efficient payment and delivery options, while empowering sellers with tools to manage and grow their businesses.
* **Features:** The Shopez e-commerce app is designed to offer a comprehensive and seamless shopping experience with its array of features. It boasts a user-friendly interface and advanced product search with filters for easy navigation and quick discovery. The app supports favorite functionality, and a seamless checkout process with multiple payment options. Sellers benefit from a dedicated dashboard to manage inventory and sales, complemented by dynamic discounts and promotions to drive sales.

**3. Architecture**

* **Frontend:** React
* **Backend:** Node.js

**Database:** The database used in this Shopez e-commerence' website is **MongoDB**. Security is ensured with SSL/TLS encryption and role-based access. Payment integrations (RazorPay), push notifications (Firebase), and analytics (Google Analytics) are included.

**4. Setup Instructions**

To develop a Shopez e-commerce web application with the specified tech stack, these are the key prerequisites:

* **Frontend Setup:**
  + Clone the repo: git clone https://github.com/yourrepo/shopez-app.git
  + Install dependencies: npm install
  + Run the app: npm run dev
* **Backend Setup:**
  + Clone the repo: git clone https://github.com/yourrepo/shopez-backend.git
  + Install dependencies: npm install
  + Set up the database (MongoDB)
  + Configure environment variables (DB, RazorPay).
  + Start the backend: npm start

**5. Folder Structure**

* **Frontend:** The **frontend** folder contains components, pages, services, and styles, with React managing the UI and Redux for API calls.
* **Backend:** The **backend** folder includes configurations, models, controllers, routes, and middleware for handling database, authentication, and business logic.

**6. Running the Application**

* **Frontend:**
  + Clone the repo: git clone https://github.com/yourrepo/shopez-app.git
  + Install dependencies: npm install
  + Run the app:
    - npm run dev
* **Backend (API):**
  + Clone the repo: git clone https://github.com/yourrepo/shopez-backend.git
  + Install dependencies: npm install
  + Configure .env file with DB URI and API keys.
  + Start the server: npm start
* **Database:** Set up MongoDB

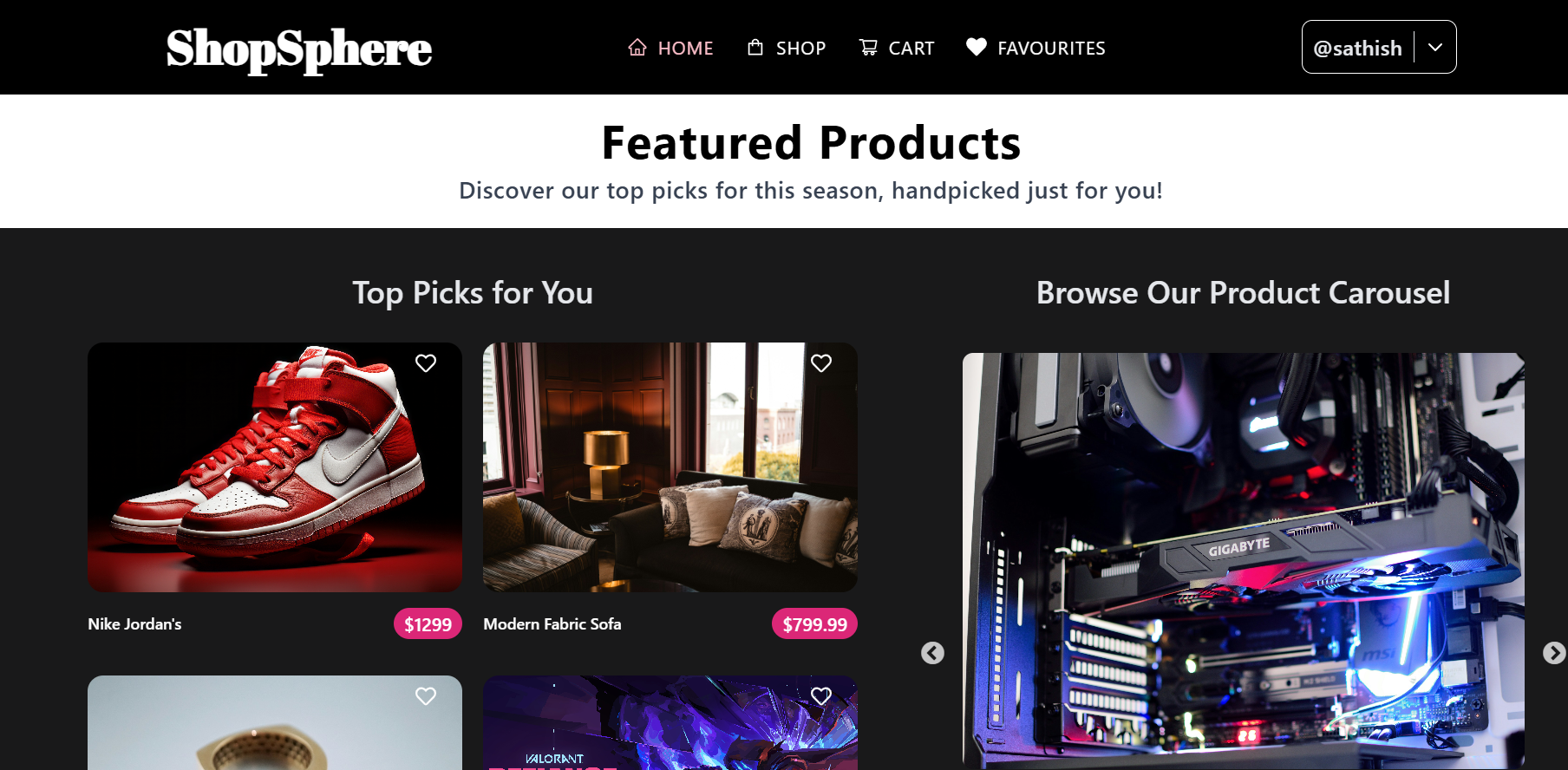
**7. API Documentation**

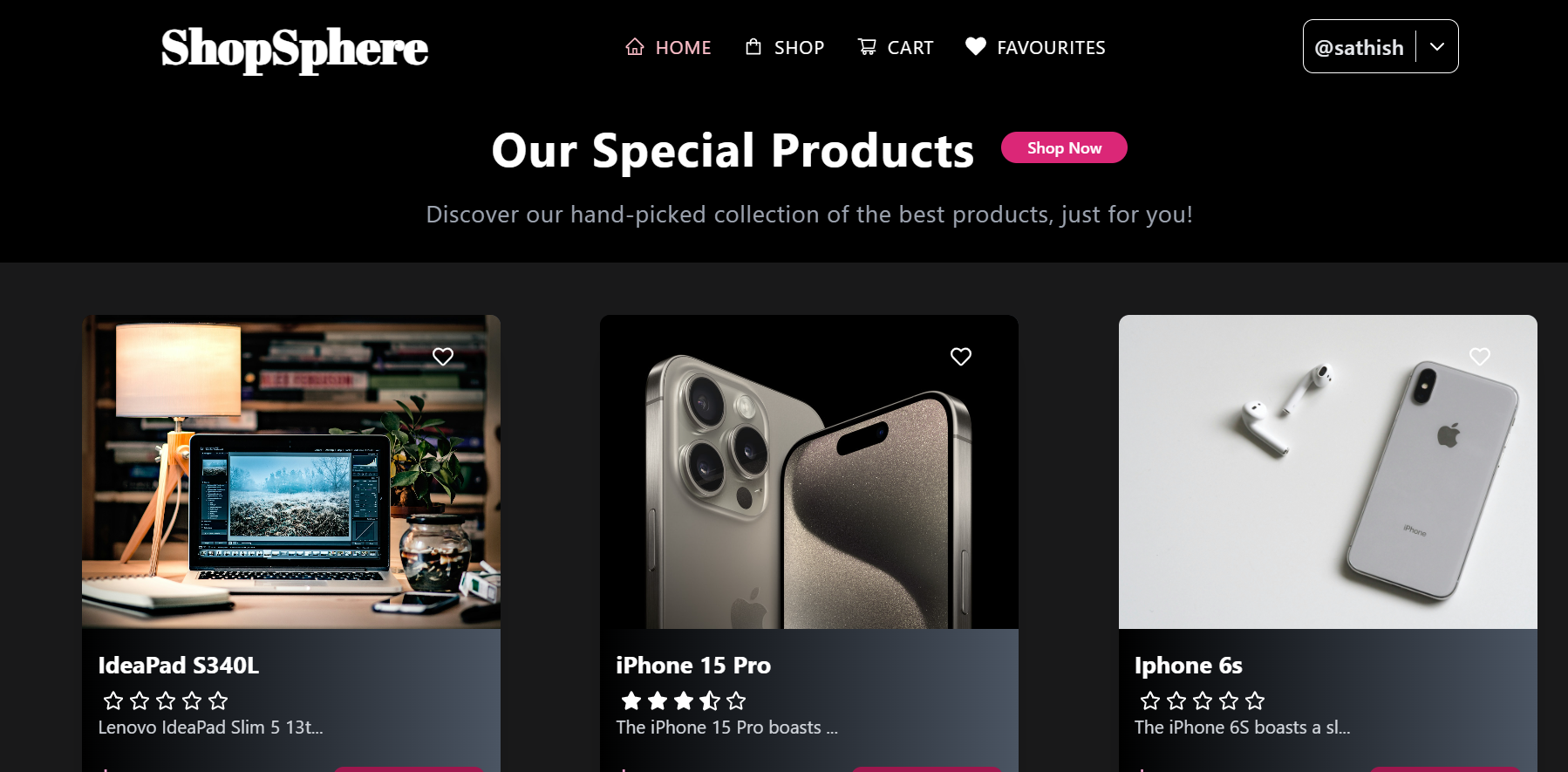
* The Shopez API provides a set of endpoints to manage user authentication, products, orders, and payments. For authentication, users can log in with POST /auth/login to get a JWT token, or register with POST /auth/register. The product-related endpoints allow users to view products (GET /products), get product details (GET /products/:id), and admins to add, update, or delete products (POST /products, PUT /products/:id, DELETE /products/:id). Orders can be placed via POST /orders, checked for status with GET /orders/:id, or canceled using DELETE /orders/:id. Payment integration includes initiating payments with POST /payments/stripe and confirming payments via POST /payments/confirm. Error handling includes standard HTTP responses such as 401 Unauthorized, 404 Not Found, and 500 Internal Server Error.

**8. Authentication**

* **Login**: POST /auth/login – Authenticates a user and returns a JWT token.
* Request: { "email": "user@example.com", "password": "password123" }
* Response: { "token": "jwt\_token\_here" }
* **Register**: POST /auth/register – Registers a new user.
* Request: { "email": "user@example.com", "password": "password123", "name": "John Doe" }
* Response: { "message": "User registered successfully" }
* **Validate Token**: GET /auth/validate-token – Validates JWT token for protected routes.
* Header: Authorization: Bearer <jwt\_token>
* Response: { "message": "Token is valid" }
* **Password Reset**: POST /auth/reset-password – Sends a password reset link to the user's email.
  + Request: { "email": "user@example.com" }
  + Response: { "message": "Password reset email sent" }

**9.User Interface**

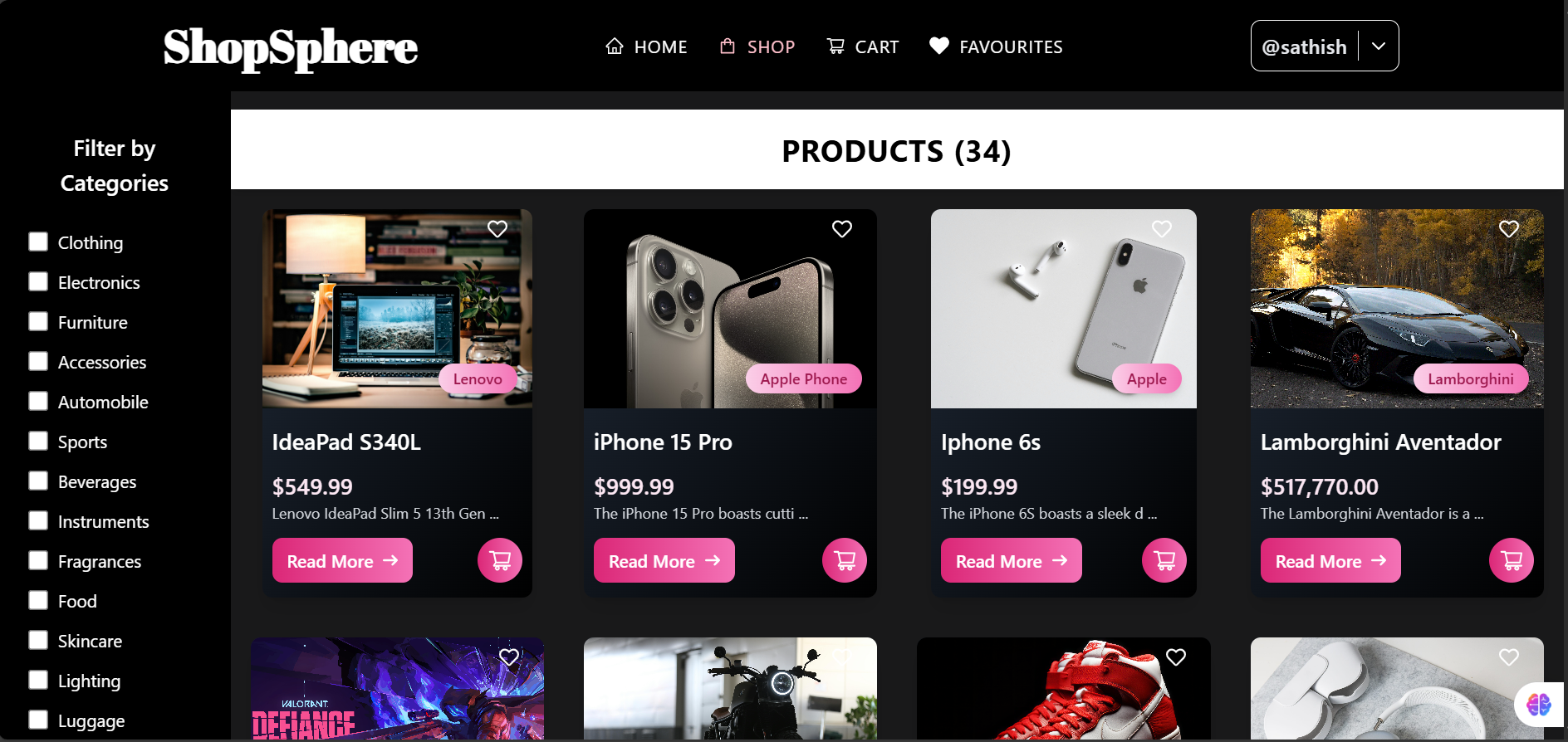


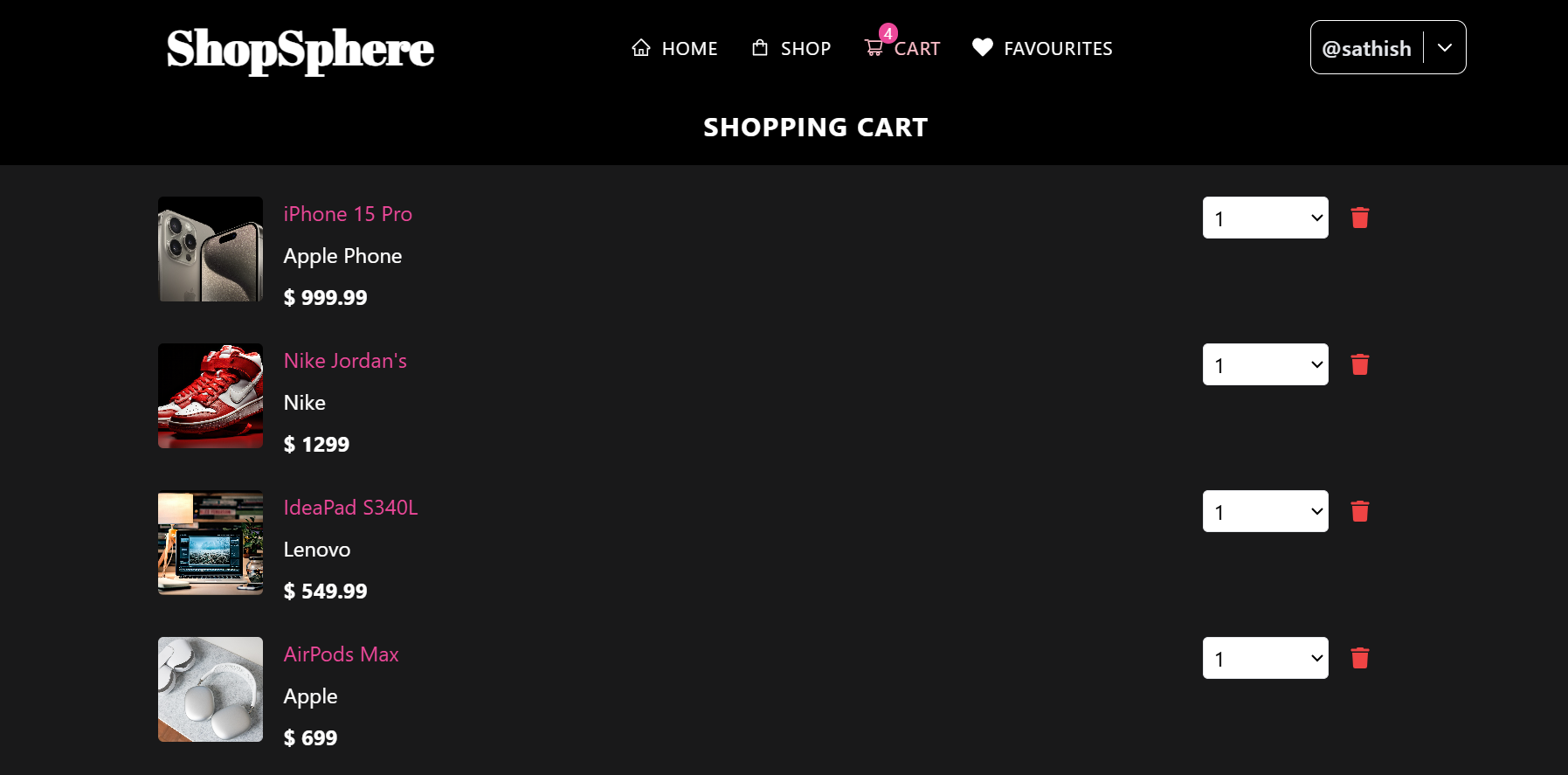


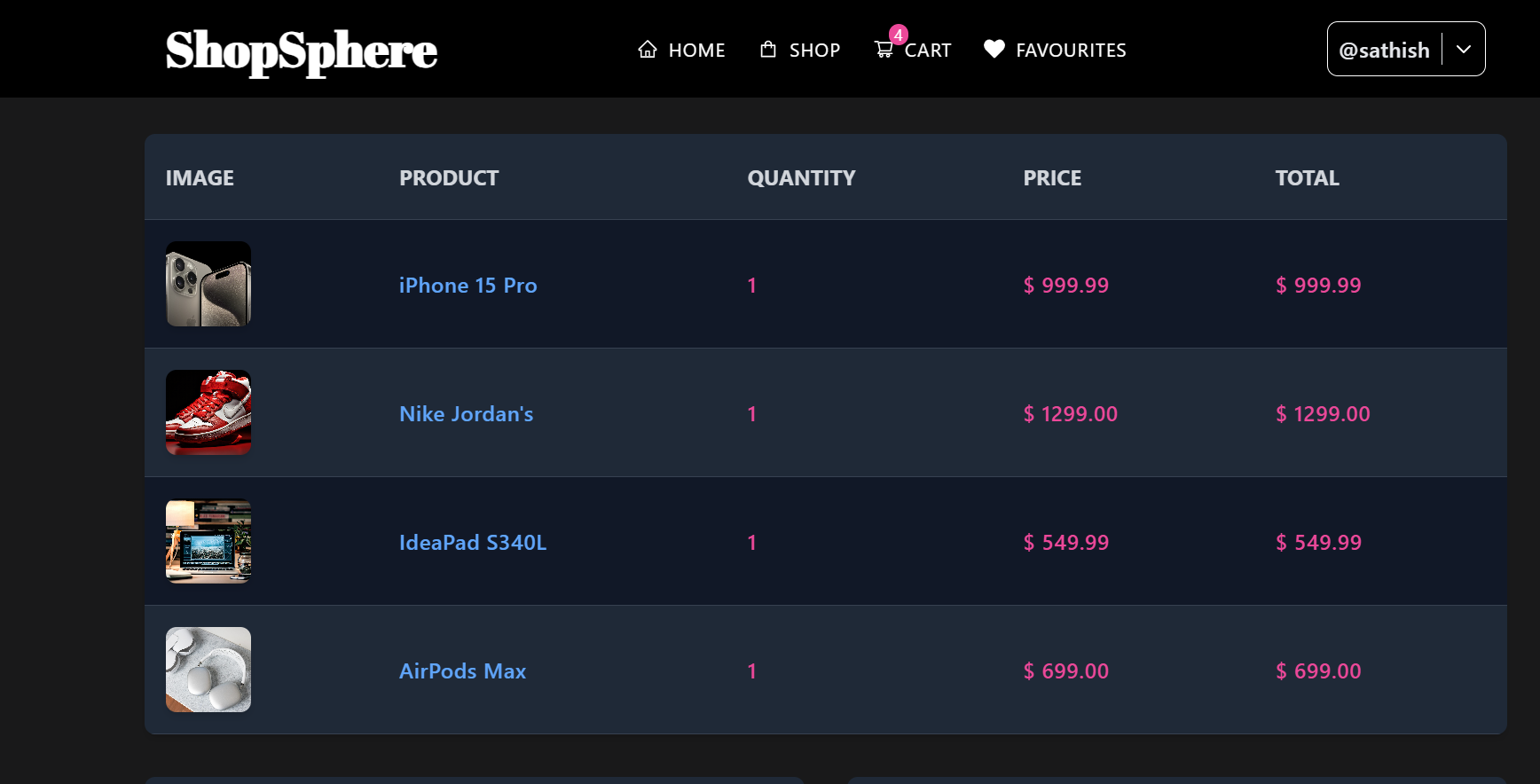
**10. Testing**

* Testing the Shopez app involves several steps for both the frontend and backend. For the frontend, manual testing ensures features like login, browsing, and checkout work smoothly on a device or emulator. Automated tests using Jest for unit tests. On the backend, manual testing through Postman checks API endpoints like login, product retrieval, and order management, while Jest with Supertest is used for automated API tests. Database testing involves validating CRUD operations using Mongoose (MongoDB). End-to-end testing tests the app's overall flow, ensuring processes like adding products to the cart and completing orders are seamless. Finally, performance testing ensures the app and APIs are optimized for speed and scalability.

**11. Screenshots or Demo**



****

****

**12. Known Issues**

1. **Login/Authentication:** Delays or failures on slow networks. Solution: Improve error handling and optimize token refresh**.**
2. **Product Images:** Slow loading or display issues. Solution: Optimize images and use caching.
3. **Cart Syncing:** Cart items may not sync across devices. Solution: Enhance session management and persistence.
4. **Checkout Issues:** Payment gateway errors (Stripe). Solution: Investigate API connections and improve feedback.
5. **App Crashes**: Occasional crashes on older devices. Solution: Optimize memory usage and fix leaks.
6. **Push Notifications:** Delivery/sync issues in certain regions. Solution: Review and adjust service integration.
7. **UI/UX**: Display issues on different screen sizes. Solution: Implement responsive design.
8. **Backend Performance**: Slow API responses during high traffic. Solution: Scale backend and optimize queries.

**13.Future Enhancements**

* **Advanced Search & Filters:** More search options like price, brand, and ratings.
* **Multi-Language Support:** Support for multiple languages.
* **Personalized Recommendations:** AI-based product suggestions.
* **Improved Payment Integration:** Add PayPal, Apple Pay, Google Pay.
* **Push Notifications:** Enhanced notifications for orders and offers.
* **User Reviews & Ratings**: Enable product reviews and ratings.
* **Analytics Dashboard**: Admin dashboard for detailed sales and customer insights.
* **AR Integration:** Visualize products in the real world before purchase**.**
* **Subscription Model:** Option for recurring orders (e.g., groceries).
* **Internationalization:** Support for international shipping and currency**.**