**E-COMMERCE SHOPPING WEBSITE**

**Anu Stores Tec and Services**

A Comprehensive Report on Full-Stack Web Application Development

Submitted By:

Name:Annu

Ghno: GH1039469

Sub mitted To:

Prof:AlirezanMahmoud

Course: M607 Computer Science Application Lab

Git Hub Website Link:

Abstract

This report documents the complete development journey of "Anu Stores Tec and Services," a modern e-commerce platform built from scratch. Throughout this project, I gained hands-on experience with full-stack web development, working with both frontend and backend technologies. The application features user authentication, product management, shopping cart functionality, payment processing, and an admin dashboard. What started as a basic concept evolved into a sophisticated platform with advanced features like multi-language support, AI chatbot assistance, and real-time order tracking. This report walks through the entire development process, from initial planning to final implementation, including the challenges I faced and how I overcame them.

INTRODUCTION

Online shopping has completely changed how we buy things. I remember when my family had to drive to multiple stores just to find what we needed. Now, everything is available at our fingertips. This observation inspired me to build my own e-commerce platform to understand what goes into creating these websites that we use every day.

When I started this project, I wanted to create something practical that real people could actually use. Not just another demo website with fake buttons, but a complete system where users could browse products, add items to their cart, make payments, and track their orders. I also wanted to include an admin side because I realized that managing an online store is just as important as the customer-facing features.

The project turned out to be much more complex than I initially thought. There were moments of frustration when things didn't work, but also moments of excitement when I finally got a feature working after hours of debugging. Through this journey, I learned that building a real application is very different from following tutorials. You have to make decisions, solve problems that nobody has written about, and sometimes start over when your approach doesn't work.

PROJECT OBJECTIVES

When I started planning this project, I sat down and wrote out what I wanted to achieve. Here's what I aimed to accomplish:

**Primary Goals:**

The main goal was to create a fully functional e-commerce website where users could actually shop. I wanted real user accounts, a working shopping cart, and the ability to process payments. It had to feel like a real online store, not just a prototype.

On the technical side, I wanted to learn full-stack development properly. That meant working with React for the frontend, Node.js and Express for the backend, and MongoDB for the database. I had used these technologies individually before, but never all together in one complete project.

**Learning Objectives:**

I wanted to understand how authentication really works. Not just copying code from Stack Overflow, but actually understanding JWT tokens, password hashing, and session management. Security was important to me because I've read about data breaches and wanted to implement things correctly.

Database design was another area I wanted to improve. I needed to figure out how to structure data for users, products, and orders in a way that made sense and performed well. This meant learning about relationships between different collections in MongoDB.

**Advanced Features:**

As the project progressed, I became more ambitious. I added features like multi-language support because I wanted the site to be accessible to people who don't speak English. The AI chatbot was something I thought would really help users get quick answers to common questions without waiting for customer support.

The order tracking feature with Google Maps integration was particularly exciting to implement. I wanted users to see exactly where their order was in the delivery process, just like the big e-commerce sites do.

LITERATURE REVIEW AND RESEARCH

Before diving into coding, I spent time researching how modern e-commerce websites work. I looked at popular sites like Amazon, Shopify stores, and various other online retailers to understand common patterns and features.

**Industry Standards:**

I noticed that most successful e-commerce sites have similar core features - user authentication, product catalogs with good search, shopping carts that remember your items, and secure checkout processes. This helped me understand what users expect from an online shopping experience.

I also researched best practices for web security. This was crucial because handling user data and payments is a serious responsibility. I learned about OWASP security guidelines, common vulnerabilities, and how to prevent them.

**Technology Stack Selection:**

Choosing the right technologies took some research. I decided on the MERN stack (MongoDB, Express, React, Node.js) because it's widely used in the industry and uses JavaScript throughout, which meant I didn't have to switch between different programming languages.

React made sense for the frontend because it's component-based, which means I could build reusable pieces of UI. This saved a lot of time as the project grew. Node.js and Express were perfect for the backend because they're fast and there's a huge ecosystem of packages available.

MongoDB was chosen for the database because it works well with JavaScript objects and offers flexibility. Since I wasn't entirely sure how my data structure would evolve, MongoDB's schema flexibility was helpful during development.

**Payment Processing Research:**

Integrating payments was something I was nervous about. After researching different options, I implemented both PayPal and a credit card payment system. I learned about PCI compliance, why you should never store credit card details yourself, and how to use payment gateways safely.

SYSTEM DESIGN AND ARCHITECTURE

The architecture of this application follows a clear separation between frontend and backend, which is a pattern I learned is called client-server architecture.

**Frontend Architecture:**

The React frontend is organized into several layers. At the top level, I have the main App component that handles routing using React Router. This decides which page to show based on the URL.

I used React Context API for state management. Initially, I considered Redux, but Context API turned out to be sufficient for this project and much simpler to implement. I created separate contexts for authentication, shopping cart, wishlist, language preferences, and theme settings. This way, different parts of the app could access shared data without passing props through multiple levels.

The component structure follows a logical hierarchy. I have page components for major sections like HomePage, ProductPage, CartPage, and CheckoutPage. Then there are smaller reusable components like Navbar, Footer, SearchBar, and ProductCard. Breaking things down into small components made the code much more manageable.

**Backend Architecture:**

The backend follows the MVC pattern - Models, Views (which in this case are JSON responses), and Controllers. The server.js file is the entry point that sets up Express, connects to MongoDB, and registers all the routes.

I organized routes into logical groups: auth routes for login and registration, product routes for managing products, order routes for handling purchases, and admin routes for management functions. Each route points to controller functions that contain the actual business logic.

Middleware plays a crucial role in the backend. I created authentication middleware that checks if a user is logged in by verifying their JWT token. There's also admin middleware that ensures only admin users can access certain endpoints.

**Database Design:**

The MongoDB database has three main collections: Users, Products, and Orders.

The User model stores account information including name, email, hashed password, role (user or admin), and address details. I made sure to never store passwords in plain text - they're always hashed using bcryptjs before saving.

The Product model contains everything needed to display and sell products: name, description, price, category, image URL, stock count, and rating information. I included fields for tracking inventory because I wanted the system to handle stock management properly.

The Order model is the most complex. It references the user who placed the order, contains an array of order items (each with product details, quantity, and price), shipping address, payment information, and various status flags like isPaid and isDelivered. I designed it to store a snapshot of product information at the time of purchase, so if product details change later, historical orders remain accurate.

IMPLEMENTATION DETAILS

The actual coding process was iterative. I didn't build everything at once. Instead, I started with the basics and gradually added features.

Phase 1: Setting Up the Foundation

I began by setting up the project structure. First, I initialized a Node.js project for the backend and created a React app for the frontend. Getting the development environment right was important - I set up scripts in package.json to run both servers simultaneously during development.

Connecting to MongoDB was my first real challenge. Initially, I tried using a local MongoDB instance, but I ran into issues with installation on my system. I switched to MongoDB Atlas, which provides a free cloud database. This actually worked out better because it meant my database was accessible from anywhere.

Phase 2: User Authentication System

Building the authentication system taught me a lot about security. I implemented registration where users provide their name, email, and password. The password gets hashed using bcryptjs before storing it in the database. This means even if someone accessed the database, they couldn't read the actual passwords.

For login, I compare the provided password with the stored hash. If they match, I generate a JWT token that includes the user's ID and role. This token is sent to the frontend, which stores it in localStorage and includes it in subsequent requests to prove the user is authenticated.

I created a PrivateRoute component in React that checks if a user is logged in before allowing access to certain pages. This prevents unauthorized users from accessing the cart, checkout, or profile pages.

Phase 3: Product Management

I created a seed script to populate the database with sample products. This was essential for testing. The products span different categories - electronics, clothing, sports items, and more. Each product has a name, description, price, category, image URL, and stock quantity.

The frontend displays products in a grid layout. I implemented a ProductCard component that shows the product image, name, price, and an "Add to Cart" button. Clicking on a product takes you to a detailed product page with more information and reviews.

For the admin side, I built interfaces to create, update, and delete products. Initially, I had a separate form for adding and editing, but I refactored it into a single reusable form that works for both operations.

Phase 4: Shopping Cart Implementation

The shopping cart was trickier than I expected. I wanted it to persist even if the user closed their browser, so I stored cart data in localStorage in addition to React state. When a user adds a product to their cart, it gets saved both in the CartContext state and localStorage.

I implemented quantity controls with plus and minus buttons. There's also validation to prevent users from adding more items than available in stock. The cart displays the total price, which updates immediately when quantities change.

One bug I had to fix was with duplicate items. Initially, adding the same product twice created two separate entries in the cart. I had to modify the logic to check if a product already exists in the cart and just increment its quantity instead.

Phase 5: Checkout and Payment Process

The checkout page collects the shipping address from the user. I added form validation to ensure all required fields are filled out correctly. Users can choose between PayPal and credit card payment.

For credit card payments, I implemented a test mode that validates the card format and expiry date without actually processing real payments. The card number field automatically formats the input with spaces every four digits, which was a nice touch that improved user experience.

The PayPal integration required getting API credentials from the PayPal Developer portal. I used their JavaScript SDK to render the PayPal button. When payment is successful, the order is created in the database with all the relevant details.

Phase 6: Order Management

After a successful payment, users are redirected to an order confirmation page showing their order details. I created an orders page where users can view all their past orders. Each order shows the items purchased, total amount, payment status, and delivery status.

The order tracking feature was one of the more complex implementations. It uses Google Maps API to show a visual representation of the order's journey from the warehouse to the delivery location. I had to learn about the Google Maps JavaScript API and how to add custom markers for different stages of delivery.

Phase 7: Admin Dashboard

The admin dashboard provides an overview of the store's performance. It displays statistics like total sales, number of orders, revenue, and user count. I used MongoDB aggregation queries to calculate these metrics from the database.

Admins can view and manage all orders, update order status, and see which orders need attention. The product management section allows full CRUD operations - creating new products, editing existing ones, and removing products from the catalog.

I also added a user management section where admins can view all registered users and their roles. This could be expanded to allow banning users or changing their roles if needed.

Phase 8: Advanced Features

**Once the core functionality was working, I added features to make the platform stand out:**

The language switcher was implemented using React Context. I created translation objects for English, Spanish, and French. Every text element in the UI references these translation keys instead of hardcoded strings. When the user changes the language, the entire interface updates immediately.

The AI chatbot was a fun addition. It's not connected to a real AI service yet, but it has predefined responses to common questions about products, orders, shipping, and returns. The chatbot interface has a clean design with message bubbles, typing indicators, and smooth animations.

The API key gate adds an extra security layer. Before accessing the site, users must enter a valid API key. This creates a sense of exclusivity and could be useful for private sales or beta testing. The key is stored in sessionStorage with an expiration time.

I spent time on the visual design too. The site has a modern, futuristic theme with a dark color scheme, gradient effects, and smooth animations. The starfield background creates an immersive experience. I used CSS animations for hover effects, loading states, and page transitions.

CHALLENGES FACED AND SOLUTIONS

**No project is without challenges. Here are some of the major obstacles I encountered:**

**CORS Issues:**

When I first tried to connect the frontend to the backend, I got CORS errors. The browser was blocking requests from localhost:3000 (frontend) to localhost:5000 (backend). I learned about CORS policy and fixed it by installing and configuring the cors package in Express. I had to specify which origins are allowed and which HTTP methods are permitted.

**State Management Confusion:**

Managing state across multiple components was confusing initially. Props drilling (passing data through many component levels) became messy quickly. That's when I learned about Context API properly. It took some refactoring to move shared state into contexts, but it made the code much cleaner and easier to maintain.

**Payment Integration Complexity:**

Integrating PayPal was challenging because the documentation assumed a lot of prior knowledge. I had to read through multiple guides and watch tutorial videos to understand the flow. The sandbox environment helped a lot for testing without using real money. Getting the webhooks to work properly for payment confirmation took several attempts.

**Image Handling:**

Initially, I hardcoded image URLs for products, which wasn't scalable. I wanted to implement image upload functionality where admins could upload product images. I researched different approaches - storing images in the database, using cloud storage services like AWS S3 or Cloudinary. Due to time constraints, I settled on using URL references for now, but this is something I want to improve.

**Responsive Design:**

Making the site look good on different screen sizes was time-consuming. I used CSS media queries extensively and tested on various device sizes using browser developer tools. The cart page and product grid required special attention to ensure they worked well on mobile devices.

**Authentication Token Management:**

Handling token expiration gracefully was tricky. I had to implement logic to check if a token is expired before making API requests. If expired, the user is logged out and redirected to the login page. I also added token refresh functionality to extend sessions for active users.

**Database Relationships:**

Understanding how to structure related data in MongoDB took some learning. Unlike SQL databases where you define relationships explicitly, MongoDB requires you to think about embedding vs. referencing. For example, should I embed product details in orders or just reference the product ID? I learned that embedding is better for data that should remain unchanged (like order history) while referencing works for data that might change (like product inventory).

TESTING AND DEBUGGING

Testing was an ongoing process throughout development. I didn't write formal unit tests initially, which is something I regret now. Most of my testing was manual.

**Functionality Testing:**

I tested each feature as I built it. For user registration, I tried creating accounts with various inputs - valid emails, invalid emails, duplicate emails, weak passwords, etc. This helped me add proper validation and error messages.

For the shopping cart, I tested edge cases like adding the same item multiple times, changing quantities, removing items, and checking if the total calculated correctly. I also tested what happens if a product goes out of stock while it's in someone's cart.

The checkout process required thorough testing because it involves multiple steps and payment. I tested both PayPal and credit card payment flows repeatedly to ensure orders were created correctly with all the necessary information.

**Browser Compatibility:**

I tested the site on different browsers - Chrome, Firefox, Edge, and Safari. Most things worked consistently, but I did find some CSS inconsistencies that needed browser-specific fixes. For example, the date input fields looked different across browsers, so I had to add custom styling.

**Responsive Testing:**

Using Chrome DevTools, I tested the site on various simulated devices - iPhone, iPad, Samsung phones, etc. I discovered several layout issues that only appeared on smaller screens. The navbar needed a complete redesign for mobile, which led me to implement a hamburger menu.

**Performance Testing:**

I used the browser's Network tab to check how fast pages loaded and how much data was being transferred. I noticed that product images were large and slowing down the page. I learned about image optimization and started using compressed images with appropriate dimensions.

I also checked the MongoDB queries to ensure they were efficient. Some queries were fetching unnecessary fields, so I modified them to select only needed data.

**Debugging Tools:**

React Developer Tools helped immensely for debugging frontend issues. I could inspect component props and state in real-time, which made finding bugs much faster. For the backend, I used console.log extensively (though I know this isn't best practice). I also used Postman to test API endpoints independently from the frontend.

RESULTS AND ACHIEVEMENTS

After several weeks of development, the project reached a stage where all core features are working reliably. Here's what was accomplished:

**Functional Deliverables:**

A complete e-commerce platform where users can browse products, add items to their cart, proceed through checkout, make payments, and track their orders. The admin dashboard provides full control over products, orders, and users. The authentication system is secure with proper password hashing and token-based sessions.

The site is fully responsive and works well on devices of all sizes. The multi-language support makes it accessible to non-English speakers. The AI chatbot provides instant help for common questions. The order tracking with maps gives users visibility into their delivery status.

**Technical Achievements:**

Successfully implemented a full MERN stack application with proper separation of concerns. The code is organized in a maintainable way with reusable components and clear folder structure. The API follows RESTful principles with appropriate HTTP methods and status codes.

Integrated multiple external services - PayPal for payments and Google Maps for location features. Implemented secure authentication using industry-standard practices. Created a database schema that efficiently handles relationships between users, products, and orders.

**Personal Growth:**

This project significantly improved my web development skills. I gained confidence working with React and building complex UIs. I learned how to structure a Node.js backend application properly with routes, controllers, and middleware. My understanding of databases improved dramatically.

I also learned important soft skills like project planning, time management, and problem-solving. When facing bugs or challenges, I learned to break problems down, research solutions, and try different approaches until something worked.

**Areas for Improvement:**

While I'm proud of what I built, there are areas that could be better. The code could use more comments and documentation. I should have written automated tests from the beginning instead of relying on manual testing. Some functions are too long and could be refactored into smaller, more focused functions.

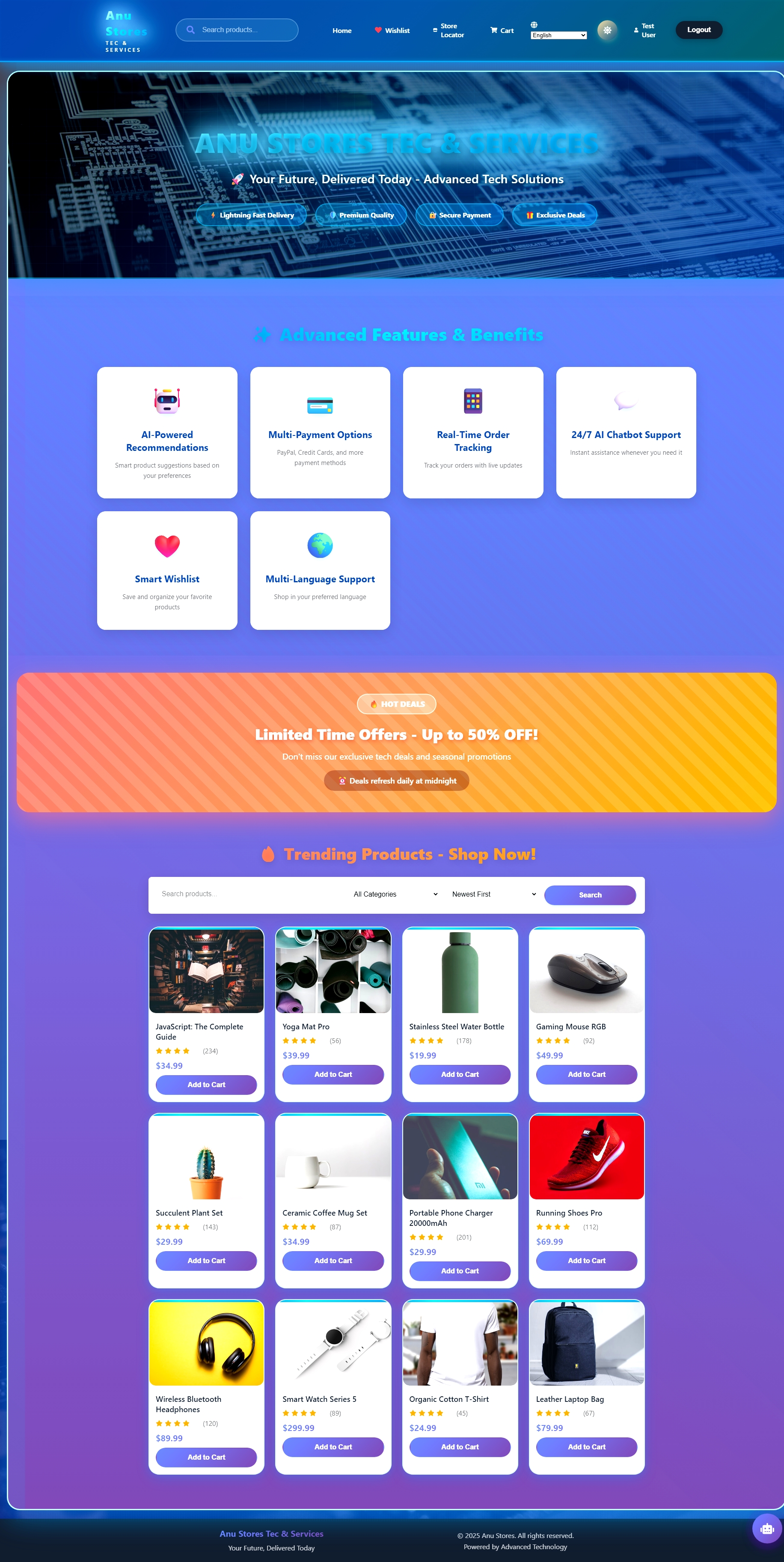
The error handling could be more comprehensive. Currently, some errors just log to the console without providing meaningful feedback to users. The UI could be more accessible - I didn't follow all WCAG guidelines for accessibility.

SCREENSHOTS AND VISUAL DOCUMENTATION

**Below are screenshots demonstrating the key features and interfaces of the application:**

1. Homepage and Product Display

The homepage serves as the entry point where users first see available products. The layout features a clean grid displaying products with images, names, prices, and quick action buttons. The navigation bar at the top provides easy access to different sections. The design uses a modern dark theme with attractive gradients and animations.



2. API Key Authentication Gate

A screenshot of a login form

AI-generated content may be incorrect.

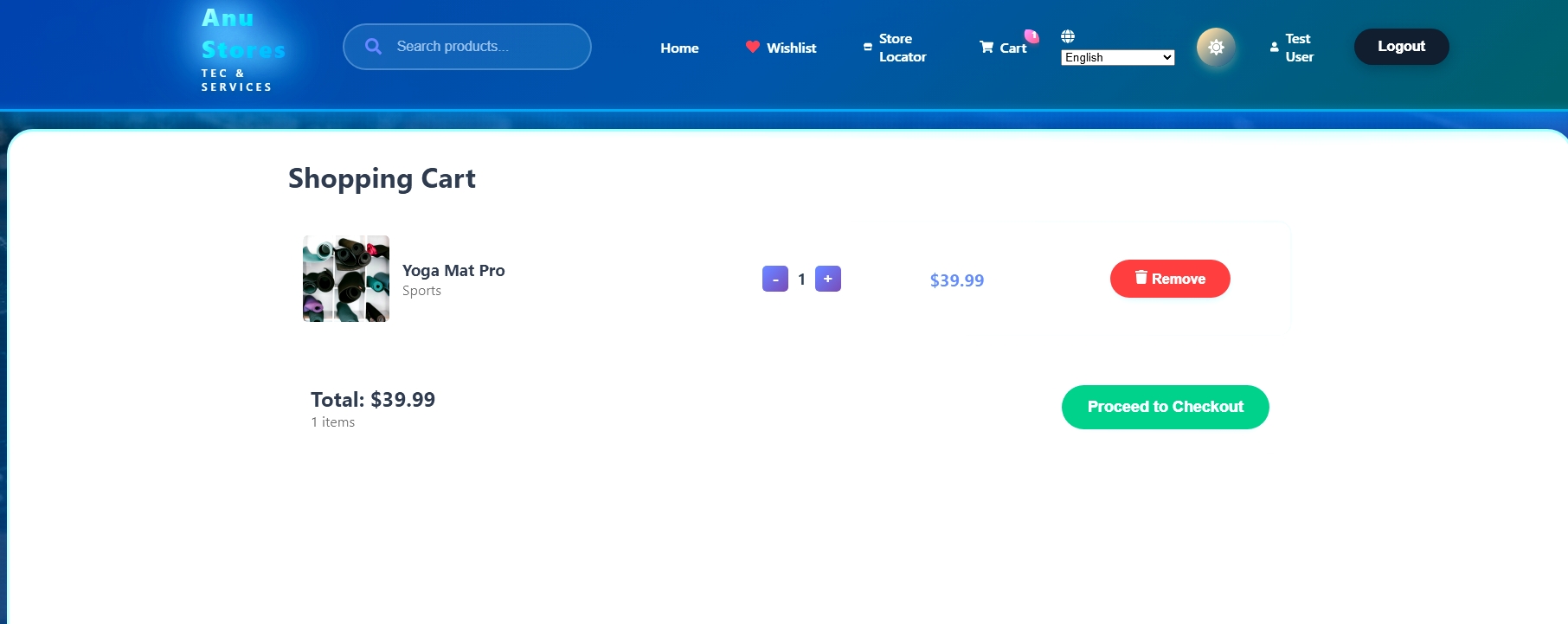
Before accessing the main application, users encounter a security gate requiring an API key. This screen features an animated starfield background that creates an immersive first impression. Users enter the demo key "ANU2025TECH" to gain access. The session persists for 24 hours before requiring re-authentication.

3. Product Details Page

A screenshot of a computer

AI-generated content may be incorrect.

Clicking on any product leads to a detailed view showing comprehensive information including description, specifications, price, and availability. Users can select quantity and add items directly to their cart from this page. The page also shows product ratings and reviews which helps users make informed decisions.

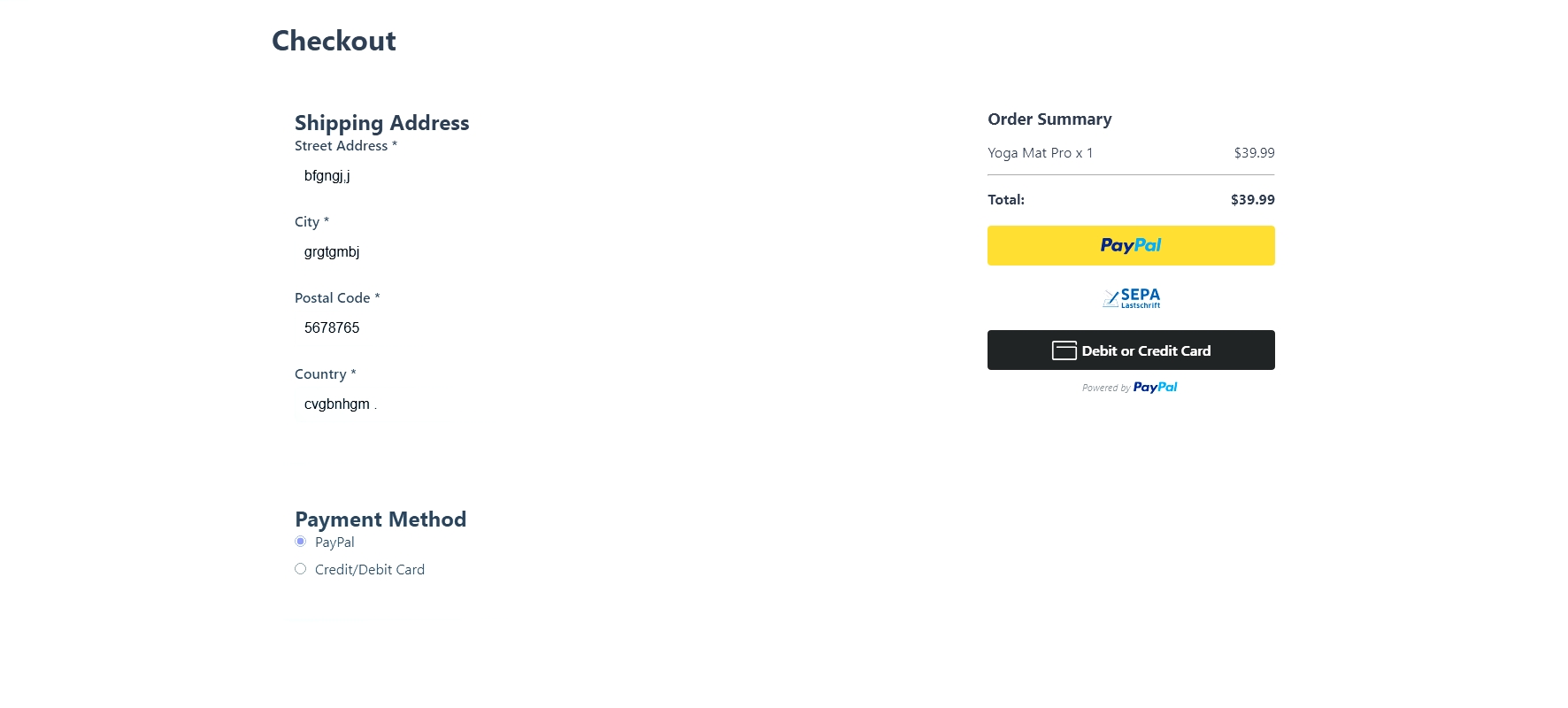


4. Shopping Cart Interface

The cart page displays all items selected for purchase with options to adjust quantities or remove items. The interface clearly shows individual item prices and the calculated total. Users can see real-time updates as they modify quantities. The prominent checkout button guides users to the next step in their purchase journey.

5. User Authentication Pages

The login and registration forms provide secure access to user accounts. Forms include validation to ensure proper data entry. Error messages appear when credentials are incorrect or fields are incomplete. The design maintains consistency with the overall site aesthetic while keeping the focus on form functionality.



6. Checkout and Payment Process

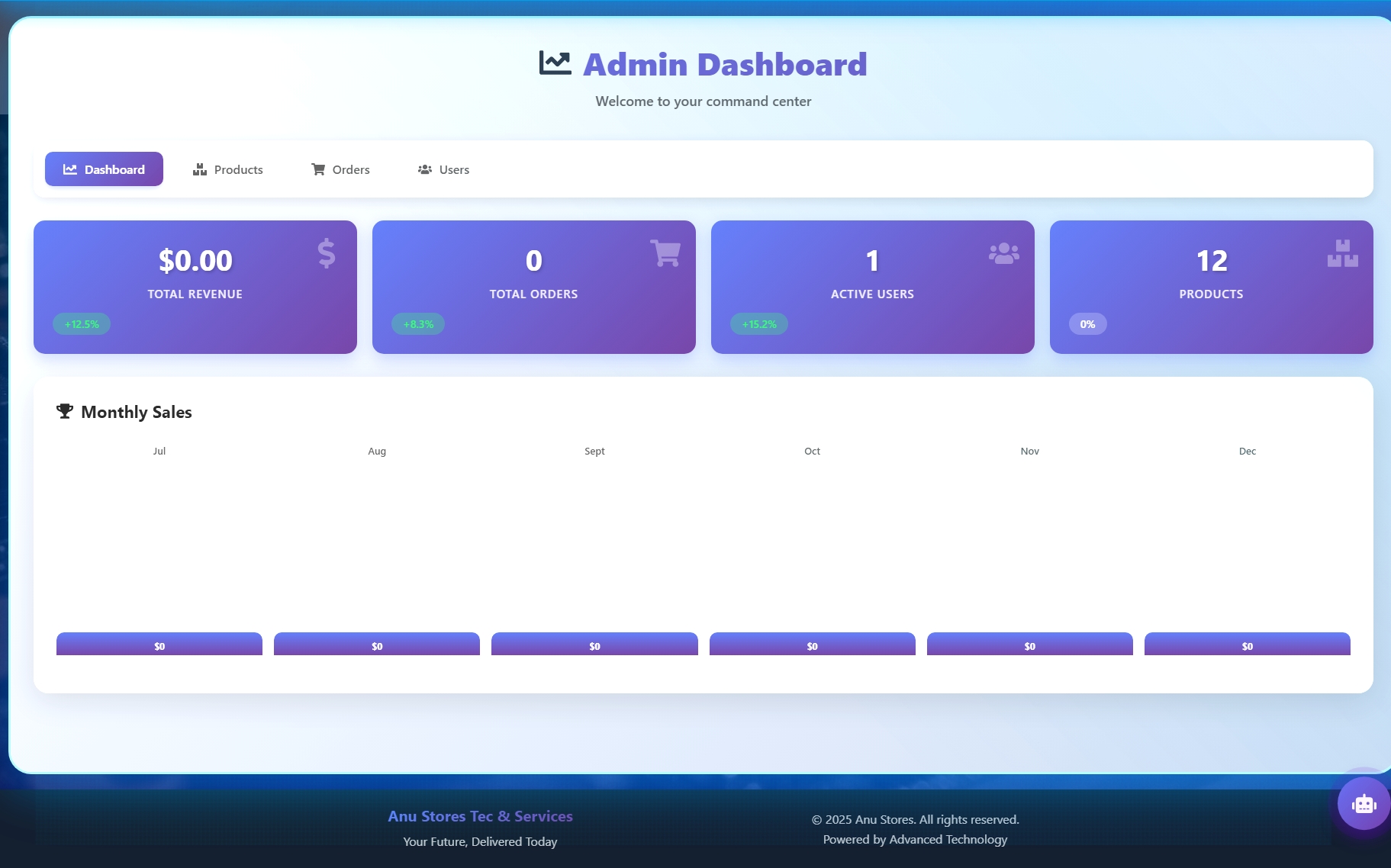
The checkout page collects shipping address information through a well-organized form. Users then choose their preferred payment method - either PayPal or credit card. The credit card form includes validation for card numbers, expiry dates, and CVV codes. The interface provides clear feedback at each step of the process.

7. Order Confirmation

After successful payment, users receive immediate confirmation with complete order details. The page displays purchased items, quantities, prices, shipping address, and payment information. A unique order ID allows users to reference their purchase. The interface provides clear next steps for tracking the order.

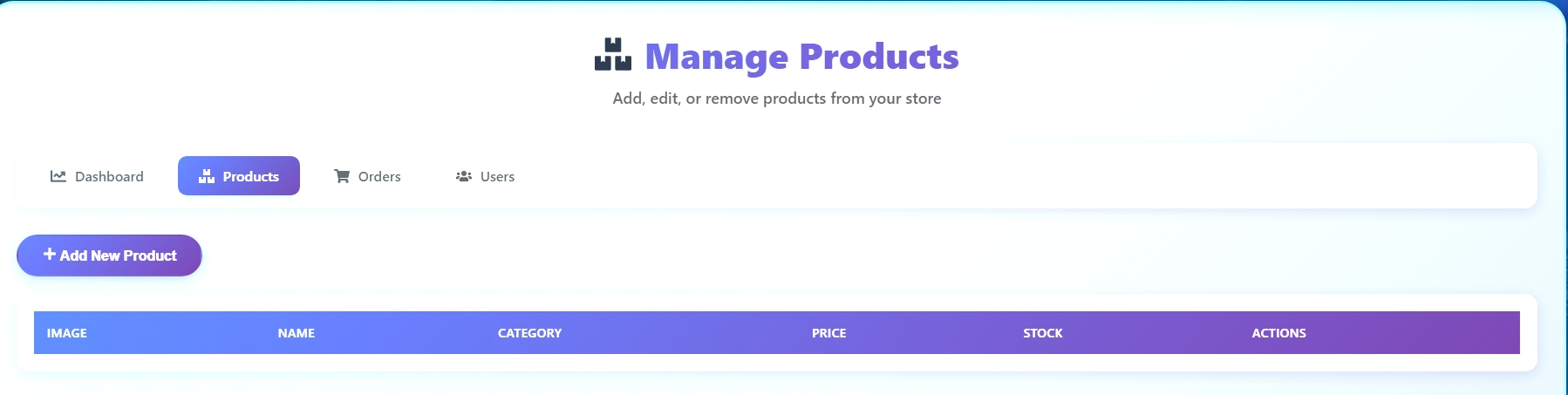
8. Order Tracking with Maps

This feature shows the real-time location and status of orders using Google Maps integration. Visual markers indicate the warehouse location, current position, and delivery destination. A progress indicator shows which stage the order is in - processing, shipped, out for delivery, or delivered. This transparency builds trust with customers.



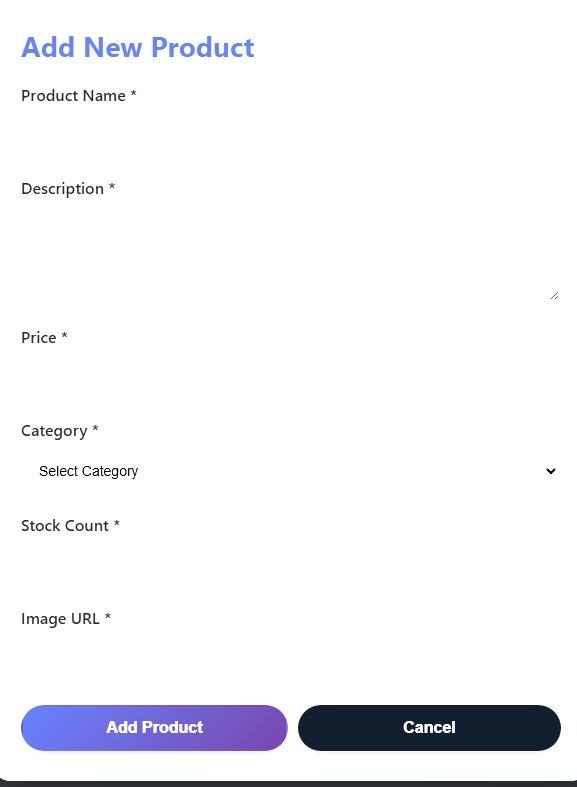
9. Admin Dashboard Overview

The admin interface provides a comprehensive view of store operations. Statistics display total sales, revenue, number of orders, and user counts. Graphs and charts visualize sales trends over time. Quick access buttons lead to product management, order management, and user management sections.



10. Product Management Interface

Admins can view all products in a table format with options to edit or delete each item. The interface includes an "Add New Product" button that opens a form for creating products. Admins can update product details, adjust prices, manage inventory, and upload images. The interface provides instant feedback when changes are saved.



11. AI Chatbot Assistant

A screenshot of a chatbot

AI-generated content may be incorrect.

A floating chatbot widget appears in the bottom-right corner of every page. Users can click it to ask questions about products, orders, shipping, or general inquiries. The bot provides predefined but helpful responses to common questions. The interface includes typing indicators and smooth animations that create a responsive feel.



FUTURE ENHANCEMENTS

While the current version is fully functional, there are many features I'd like to add in the future:

**Enhanced Features:**

Product reviews and ratings where users can leave feedback on items they've purchased would add social proof. A wishlist feature is partially implemented but could be expanded. Advanced search with filters for price range, category, ratings, and availability would improve product discovery.

Email notifications for order confirmations, shipping updates, and promotional offers would keep users informed. Push notifications for mobile users could alert them about order status changes or special deals.

**Technical Improvements:**

Implementing automated testing with tools like Jest for unit tests and Cypress for end-to-end tests would make the application more reliable. Better error handling with proper error boundaries in React and comprehensive try-catch blocks in the backend would improve stability.

Adding image upload functionality with cloud storage integration would make the admin experience better. Implementing a proper content management system for managing site content would be valuable.

Performance optimization through code splitting, lazy loading, and caching strategies would make the site faster. Server-side rendering or static generation with Next.js could improve initial load times and SEO.

**Business Features:**

Implementing a recommendation system that suggests products based on browsing history and purchases would increase sales. Discount codes and promotional campaigns would help with marketing efforts. Integration with inventory management systems would automate stock tracking.

Multiple payment gateway options beyond PayPal would give users more flexibility. Support for different currencies and international shipping would expand the potential market.

**Security Enhancements:**

Two-factor authentication for user accounts would add an extra security layer. Rate limiting on API endpoints would prevent abuse. More comprehensive input validation and sanitization would protect against injection attacks.

Regular security audits and penetration testing would help identify vulnerabilities. Implementing proper logging and monitoring would help detect suspicious activities.

LESSONS LEARNED

**This project taught me valuable lessons that go beyond technical skills:**

**Planning Matters:**

I learned that spending time on planning before coding saves time overall. Initially, I jumped straight into coding without a clear plan, which led to frequent refactoring. Creating a basic design document and database schema first would have been more efficient.

**Break Problems Down:**

Complex features become manageable when broken into small steps. When implementing the checkout process seemed overwhelming, I divided it into stages - cart validation, address form, payment selection, payment processing, and order creation. Tackling one piece at a time made it achievable.

**User Experience is Critical:**

Building features is one thing, but making them user-friendly is another. I learned to think from the user's perspective - clear error messages, loading indicators, intuitive navigation, and responsive feedback make a huge difference in how people perceive the application.

CONCLUSION

Building this e-commerce platform from scratch was one of the most challenging and rewarding experiences of my learning journey. What started as a course project became a comprehensive application that taught me full-stack development in a practical, hands-on way.

The project forced me to learn technologies deeply rather than superficially. I couldn't just follow a tutorial because I had to make decisions about architecture, solve unique problems, and integrate multiple systems. This real-world experience is invaluable and very different from classroom learning.

I'm proud of what I accomplished - a fully functional online store with features comparable to professional e-commerce sites. Users can shop, pay, and track orders. Admins can manage the entire operation through a clean dashboard. The code is organized and maintainable.

More importantly, I learned how to learn. When I encountered problems I couldn't solve immediately, I developed strategies for finding solutions - reading documentation carefully, breaking problems down, experimenting with different approaches, and knowing when to seek help from the developer community.

This project showed me that I'm capable of building complex applications independently. It gave me confidence in my abilities and a clearer sense of what I want to learn next. I understand both my strengths and the areas where I need to improve.

The experience also highlighted the importance of soft skills in development. Communication, time management, perseverance, and problem-solving are just as important as coding ability. Real projects don't go smoothly - there are setbacks, bugs, and moments of frustration. Learning to push through those moments is part of becoming a developer.

Looking ahead, I'm excited to continue improving this project and applying what I learned to new challenges. The fundamentals I gained - working with databases, building APIs, creating React interfaces, handling authentication, integrating external services - are transferable to many other applications.

This project transformed me from someone who could follow tutorials to someone who can build functional applications independently. That's the real achievement, more valuable than the code itself.

TECHNICAL SPECIFICATIONS

**Frontend Technologies:**

- React.js (v18.2.0) for UI components and user interface

- React Router (v6.x) for client-side routing and navigation

- Context API for state management across components

- Axios for HTTP requests to backend API

- CSS3 with custom animations and responsive design

- React Icons library for consistent iconography

**Backend Technologies:**

- Node.js (v14+) runtime environment

- Express.js (v4.18) web application framework

- MongoDB (v5.x) NoSQL database

- Mongoose (v6.x) for object data modeling

- JWT (jsonwebtoken) for authentication tokens

- Bcryptjs for password hashing and security

**External Services:**

- PayPal SDK for payment processing

- Google Maps JavaScript API for location features

- MongoDB Atlas for cloud database hosting

**Development Tools:**

- npm for package management

- Concurrently for running multiple servers

- Nodemon for automatic server restart during development

- Git for version control

- VS Code as primary code editor

**Security Implementations:**

- Password hashing before database storage

- JWT token-based authentication

- HTTP-only cookies for sensitive data

- Input validation and sanitization

- CORS configuration for cross-origin requests

- Role-based access control for admin routes

**Official Documentation:**

- React Documentation (react.dev)

- Express.js Guide (expressjs.com)

- MongoDB Manual (mongodb.com/docs)

- Mongoose Documentation (mongoosejs.com)

- PayPal Developer Documentation

- Google Maps Platform Documentation

**Online Learning Platforms:**

- MDN Web Docs for HTML, CSS, and JavaScript reference

- Stack Overflow for problem-solving and community support

- YouTube tutorials for complex concepts

- GitHub repositories for code exampl

**Project Repository Structure:**

**The complete project is organized with the following structure:**

- /client directory contains all React frontend code

- /server directory contains Express backend code

- /models define MongoDB schemas

- /routes define API endpoints

- /controllers contain business logic

- /middleware handles authentication and validation

- /context manages React state

- /components houses reusable UI components

- /pages contains main page components

**Environment Configuration:**

**The project requires specific environment variables:**

- MONGODB\_URI for database connection

- JWT\_SECRET for token generation

- PAYPAL\_CLIENT\_ID for payment processing

- GOOGLE\_MAPS\_API\_KEY for map features

- PORT for server configuration