

E-Commerce Platform for High-End Tech Products Final Project Report

-Super NiuBi Market – Ultimate Tech Shopping Platform

Members:

Zheng JieRong

(Team Leader, Full-stack Development, Backend API & Authentication) |

Wu YuHan

(Frontend Development, UI/UX Design, Cart & Checkout Implementation) |

Labor Division and Contribution Percentage

As the two developers of this project, we (Zheng JieRong and Wu Yuhan) collaborated closely, dividing tasks based on our strengths while ensuring overlap for quality assurance. Zheng JieRong focused on the backend and core logic, while Wu Yuhan handled the frontend and user-facing features. We conducted joint code reviews and **testing**.

ZhengJieRong:

Main Modules & Tasks: Backend architecture (Express.js, API routes, JWT authentication, data storage with JSON files), seller dashboard backend, order management, product upload with Multer, API wrapper (api.js), security features, deployment and testing

WuYuHan:

Main Modules & Tasks: Frontend pages (HTML/CSS/JS for all views: index, products, detail, cart, checkout, login/register, help centers), UI design with cyberpunk theme, responsive layout, client-side logic (cart updates, image gallery, toast notifications), integration with backend APIs

We both contributed to documentation, debugging, and the overall project integration, with Zheng JieRong taking a slight lead due to the complexity of backend security and data persistence.

1. Introduction

We (Zheng JieRong and Wu Yuhao) developed Super NiuBi Market as our final project for the Web Application Development course. This e-commerce platform targets tech enthusiasts, particularly gamers and professionals seeking high-end peripherals like mechanical keyboards, gaming mice, and headsets. The name "NiuBi" (a playful Chinese slang for "awesome") embodies our vision of a cutting-edge, user-friendly site with a cyberpunk aesthetic.

Business Overview:

- Target Audience: Young adults in Hong Kong and the Greater Bay Area interested in premium tech gadgets.
- Model: Hybrid B2C/C2C – buyers shop from seller-listed products; sellers manage their inventory via a dashboard.
- Unique Selling Points: Dark-themed UI with cyan accents for a futuristic feel, role-based access (buyer/seller), simulated payments, and persistent data via a custom backend.
- Project Goals: Demonstrate full-stack skills, from frontend interactivity to backend API security, while ensuring a seamless user experience.

From our perspective as developers, the project challenged us to integrate frontend responsiveness with backend reliability, emphasizing secure authentication and data handling without a full database.

2. Listing the Functional Requirements Implemented

Part1:

We implemented the following key features, prioritizing user experience and security:

F01

Feature: User registration for buyers and sellers (with role-specific fields like business license for sellers)

Remarks: Toggleable forms in register.html; backend validation in auth.js

F02

Feature: Secure login/logout with JWT tokens and auto-redirect on session expiry

Remarks: Handled in login.html and api.js; bcrypt for password hashing

F03

Feature: Product browsing with filters (category, price range, sorting by price/sales)

Remarks: Implemented in products.html; dynamic rendering via JS

F04

Feature: Product detail view with image gallery, thumbnails, and navigation arrows

Remarks: product_detail.html; supports multiple images from backend

F05

Feature: User registration for buyers and sellers (with role-specific fields like business license for sellers)

Remarks: Toggleable forms in register.html; backend validation in auth.js

F06

Feature: Cart management: update quantity, remove items, real-time subtotal calculation

Remarks: jQuery event handlers in cart.html; PUT /user/cart; cart.html; server-side storage in users.json

F07

Feature: Checkout process with address input, delivery options, and simulated WeChat/Alipay payment via QR codes

Remarks: checkout.html; POST /orders/place-order clears cart

F08

Feature: Seller dashboard for product upload, editing, inventory management, and status toggling (publish/unpublish)

Remarks: seller_dashboard.html; Multer for image uploads in products.js

F09

Feature: Order history viewing for buyers

Remarks: Integrated in user profile (via /orders GET)

F10

Feature: Feedback submission form with backend storage

Remarks: help_feedback.html; POST /feedback

F11

Feature: Help center pages (FAQ, service policy, after-sales, feedback)

Remarks: Static content with dynamic elements like cart count

F12

Feature: Responsive design across devices

Remarks: Media queries in CSS; tested on mobile/desktop

F13

Feature: About us page with platform introduction

Remarks: about.html

Part 2:

The first time I saw it is on Tiktok and knew it was Due to Python implementation. But I searched many different knowledge from Github, BiliBili, and ask others for help. Finally, I developed a tremendous Christmas Tree as you can see here by HTML, CSS, JS, which we learnt in this semester.

F14: Feature: Christmas Tree

Core Concept

This is a browser-based 3D animation that uses thousands of tiny points (particles) to create a complete Christmas scene—tree, heart, water, and stars—without using traditional 3D models.

Building Blocks

1. Particle System Foundation

- Everything you see consists of individual dots called "particles"
- Each particle has position, color, and size properties
- Thousands of particles combine to form recognizable shapes
- This approach creates a magical, sparkling effect

2. Mathematical Shapes

- **The Heart:** Created using a mathematical formula that plots points along a classic heart curve

- **The Tree:** Built as 7 concentric circular layers that widen toward the bottom
- **Water Ripples:** Particles arranged in circles that move with wave patterns
- **Stars:** Randomly scattered particles with varying brightness

Visual Design Logic

Color Scheme

- **Tree:** 60% pink, 40% white particles create a festive candy-like appearance
- **Heart:** Soft pink (#edcad1) with brighter centers, darker edges for depth
- **Ground:** Pure white particles that look like water or snow reflections
- **Stars:** White with random brightness variations (60-100%)

Spatial Arrangement

- Tree stands in the center
- Heart sits on top as a tree-topper
- Water ripples spread outward on the ground
- Stars fill the distant background

Animation Mechanics

1. Tree Rotation

- The entire tree (including heart) slowly rotates
- Creates a "display stand" effect where you can see all sides
- Rotation speed: about 0.3 degrees per frame

2. Water Wave Effect

- Uses a sine wave mathematical function
- Waves start from the center and travel outward
- Particles move up and down based on their distance from center
- Creates the illusion of expanding ripples

3. Real-time Updates

- The computer redraws the entire scene 60 times per second
- Each frame calculates new particle positions

- This creates smooth, continuous motion

Audio Interaction

User Activation

- Modern browsers block automatic audio playback
- Music only starts after user clicks or presses a key
- This respects browser policies and user preferences

Volume Control

- Top-right button toggles sound on/off
- Visual feedback shows muted state
- Music loops continuously for atmosphere

Adaptive Display

Responsive Design

- Automatically adjusts to any screen size
- Maintains correct proportions when resizing
- Works on desktop and mobile devices

Performance Optimization

Efficient Rendering

- Uses advanced geometry techniques for smooth performance
- Controls particle count to balance beauty and speed
- Optimizes transparent particle rendering

Lightweight Approach

- No heavy 3D models—only mathematical calculations
- Single audio file for background music
- Clean code structure for reliable operation

Creative Innovation

1. **Pure Particle Art:** Traditional scenes use solid shapes; this uses only dots
2. **Mathematical Beauty:** Formulas create organic, pleasing shapes

3. **Layered Depth:** Foreground, middle ground, and background create 3D space
4. **Unified Style:** All elements share the same particle aesthetic
5. **Interactive Elements:** User controls both viewing angle and audio

Overall Effect

The combination of mathematical precision, particle effects, and gentle animation creates a dreamlike Christmas scene that feels both magical and technically sophisticated. The scene invites viewers to appreciate how simple elements (dots) can combine to create complex beauty through programming and mathematics.

3. Design Features

3.1 Web Structure

Our site follows a modular structure for maintainability:

— index.html	→ Homepage with carousel and featured products
— products.html	→ Product listing with search/filters
— product_detail.html	→ Detailed product view
— cart.html	→ Shopping cart management
— checkout.html	→ Order checkout and payment simulation
— login.html	→ Unified login for buyers/sellers
— register.html	→ Role-based registration
— seller_dashboard.html	→ Seller management interface
— help_faq.html, help_policy.html, etc.	→ Help center sections
— about.html	→ Platform overview
— js/	
— api.js	→ API client with auth handling
— common.js	→ Shared utilities (e.g., cart count)
— script.js	→ General scripts
— style.css	→ Global styling

└─ uploads/ → Server-hosted product images

Backend:

backend/

└─ server.js → Express app entry

└─ routes/ → API routes (auth.js, user.js, products.js, order.js)

└─ data/ → JSON files (users.json, products.json, orders.json)

3.2 UI/UX Explanation

- Frontend Analysis: The UI adopts a dark cyberpunk theme (#101010 background, #00FFFF accents) for immersion. We used sticky headers for navigation, toast notifications for feedback (e.g., "Added to cart"), and responsive grids (flexbox/grid) for product displays. Processing involves jQuery for DOM manipulation and event handling, ensuring smooth interactions like quantity updates without page reloads. Challenges included syncing cart badges across pages – solved via async API calls in common.js.
- Backend Analysis: The backend uses Express for routing, with JWT for authentication (protecting routes like /user/cart). Data is stored in JSON files for simplicity, with custom Store class for CRUD operations. Processing flow: Requests hit api.js (frontend wrapper) → Express middleware verifies token → Route handlers update JSON. Security: bcrypt hashing, UUID for IDs, Multer for secure file uploads. We focused on error handling (e.g., 401 redirects) to prevent data leak

3.3 Class Diagram (Simplified)

Class: User

- id: string

- name: string

- email: string

- password: string (hashed)

- role: 'user' | 'seller'

- cart: Array<CartItem>

- Methods: register(), login()

Class: Product

- id: string

- name: string

- price: number

- images: Array<Image>

- sellerId: string

- Methods: upload(), updateStatus()

Class: Order

- id: string

- userId: string

- items: Array<OrderItem>

- status: 'pending' | 'paid'

- Methods: placeOrder(), getHistory()

Associations:

- User 1:* Product (sellers own products)

- User 1:* Order (buyers place orders)

- Product 1:* CartItem (in carts)

4. Description of Implementation

From our developer perspective, we built the site iteratively: starting with backend APIs, then frontend integration.

- **Technologies Used:**

- Frontend: HTML5, CSS3 (with media queries), JavaScript (ES6+), jQuery 3.5.1 (for DOM/events).
- Backend: Node.js v20+, Express v4.22, bcryptjs (hashing), jsonwebtoken (JWT), multer (uploads), uuid (IDs).
- Tools: VS Code, Nodemon (dev), Postman (API testing). No frameworks like

React/Vue to focus on vanilla skills.

- 3rd Party: jQuery CDN, Placeholder images for testing.

- **High-Level Code Walkthroughs:**

1.

Authentication (auth.js – Zheng JieRong): Registers users with role checks; hashes passwords; issues JWT. Login verifies hash and returns token/user.

// Register excerpt

```
const hashedPassword = await bcrypt.hash(password, 10);
```

```
const user = { id: uuidv4(), ... }; // Role-specific fields
```

```
await userStore.push(user);
```

```
const token = jwt.sign({ userId: user.id, role: user.role }, JWT_SECRET);
```

2.

Cart Management (cart.html & user.js – Wu Yuhang): Fetches cart via GET /user/cart; updates with PUT; adds via POST. Real-time rendering with jQuery.

// Render cart

```
function renderCart() {
```

```
  let subtotal = 0;
```

```
  cartData.forEach(item => {
```

```
    const row = `<tr>...</tr>`;
```

```
    subtotal += item.price * item.quantity;
```

```
  });
```

```
}
```

3. **Product Upload (seller_dashboard.html & products.js – Zheng JieRong):** FormData for images; Multer saves to /uploads; stores URL in products.json.

// Upload handler

```
router.post('/', upload.single('image'), async (req, res) => {
```

```
  const product = { id: uuidv4(), images: [{ url: `/uploads/${req.file.filename}` }], ... };
```

```
  await productStore.push(product);
```

});

4. Payment Simulation (checkout.html – Wu Yuhan): Generates QR based on method; simulates success after "scan".

5. Screenshots of Web Prototype

(As developers, we have captured screenshots in a logical flow: from homepage browsing to purchase completion, and seller operations. Please insert them here as per the navigation sequence below. We recommend high-resolution images with annotations.)

1. Homepage (index.html) – Carousel and featured products.
2. Product List (products.html) – Filters applied.
3. Product Detail (product_detail.html) – Gallery view.
4. Shopping Cart (cart.html) – Items with totals.
5. Checkout (checkout.html) – Form and QR code.
6. Login/Register – Toggle views.
7. Seller Dashboard – Upload form and inventory.
8. Help Center – FAQ page.
9. Mobile Responsive Views – Key pages on phone.

6. List of References

1. Express.js Documentation. <https://expressjs.com/> (Accessed Nov 2025).
2. JSON Web Tokens Introduction. <https://jwt.io/introduction> (Accessed Nov 2025).
3. Multer File Upload Guide. <https://www.npmjs.com/package/multer> (Accessed Nov 2025).
4. jQuery API Documentation. <https://api.jquery.com/> (Accessed Nov 2025).
5. MDN Web Docs – Fetch API. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (Accessed Nov 2025).
6. Cyberpunk UI Design Inspiration. Dribbble.com (various posts, 2023-2025).
7. Node.js Best Practices. <https://github.com/goldbergonyi/nodebestpractices> (Accessed Nov 2025).

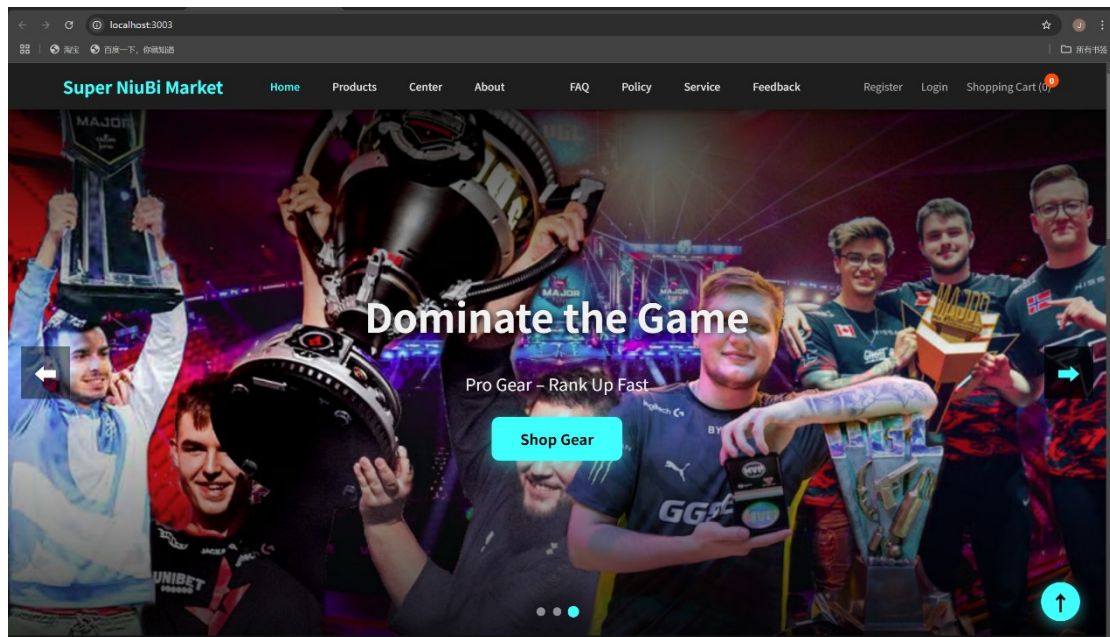
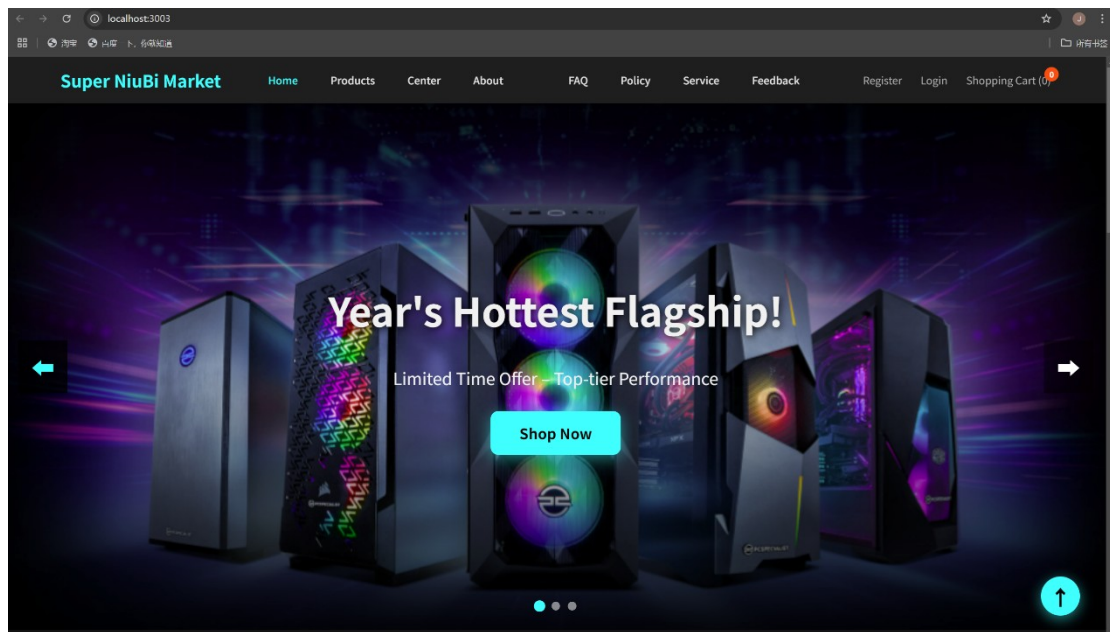
8. Hanumanth, L. (2021) HTML and CSS: Design and Build Websites, Kindle Unlimited.
9. John Wiley & Duckett, S. (2022) JavaScript and JQuery: Interactive Front-End Web Development, 1st Edition, John Wiley & Sons.
10. Ruvalcaba, Z. Delamater, M. (2017) Murach's JavaScript and jQuery, 3rd Edition, Mike Murach & Associate.
11. Bojinov, V. (2018) RESTful Web API Design with Node.js 10, 3rd Edition, Packt Publishing. Syed, B. (2014) Beginning Node.js, 1st Edition, Apress.
12. Herron, D. (2018) Node.js Web Development, 4th Edition, Packt Publishing.
13. Learn web development. MDN. (2022). Retrieved November 14, 2022, from <https://developer.mozilla.org/en-US/docs/Learn>

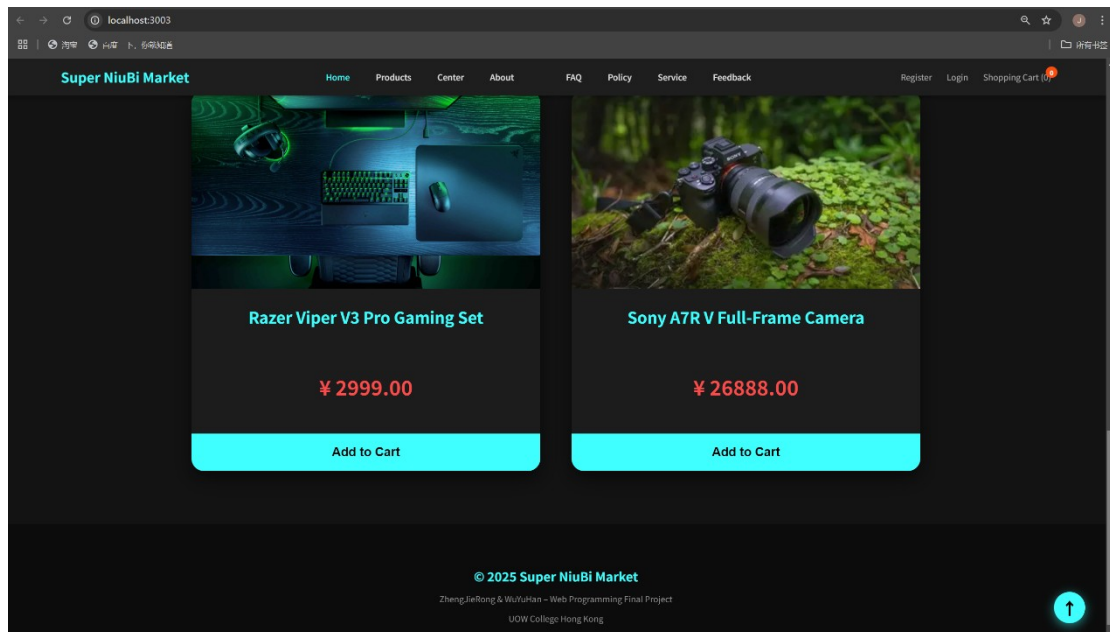
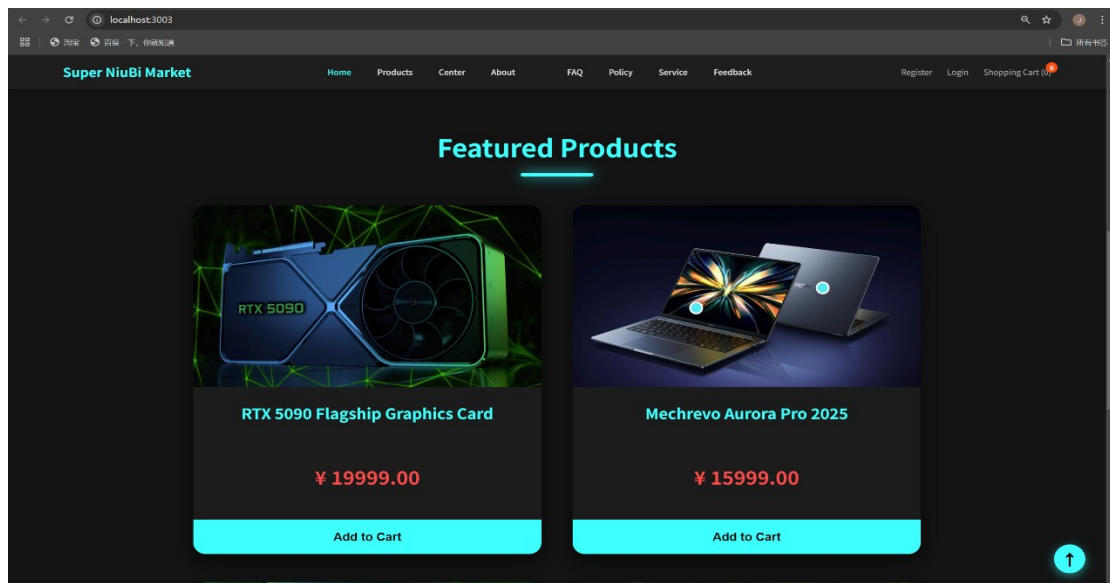
Declaration

We, Zheng JieRong and Wu Yuhan, declare that this project is our original work, developed collaboratively as UOW College Hong Kong Year 2 students. All external sources are cited.

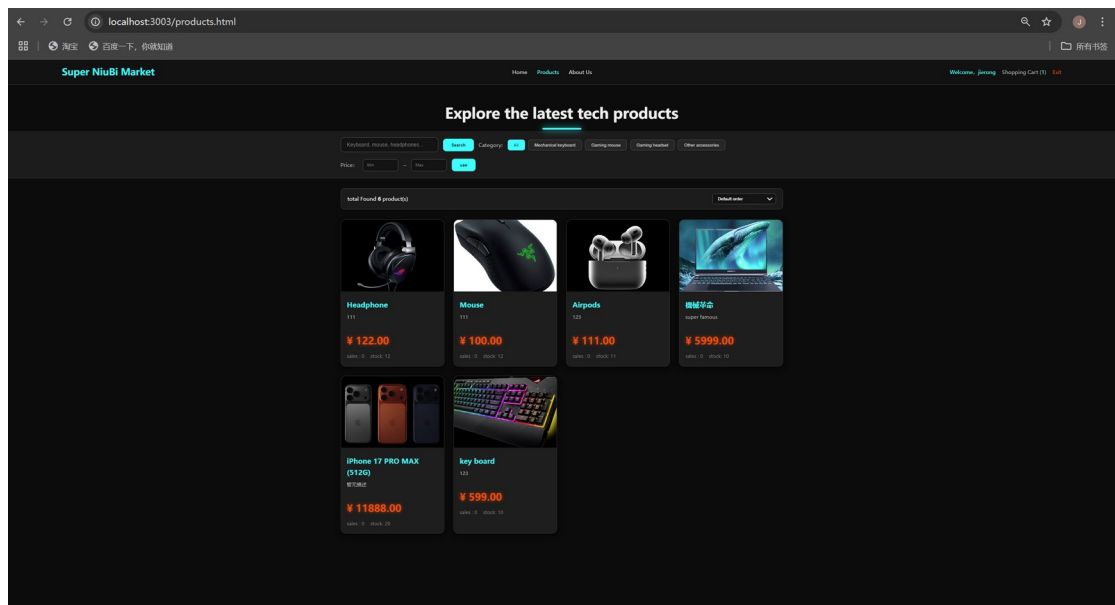
Blow is Demo (Screenshot):

Home Page:

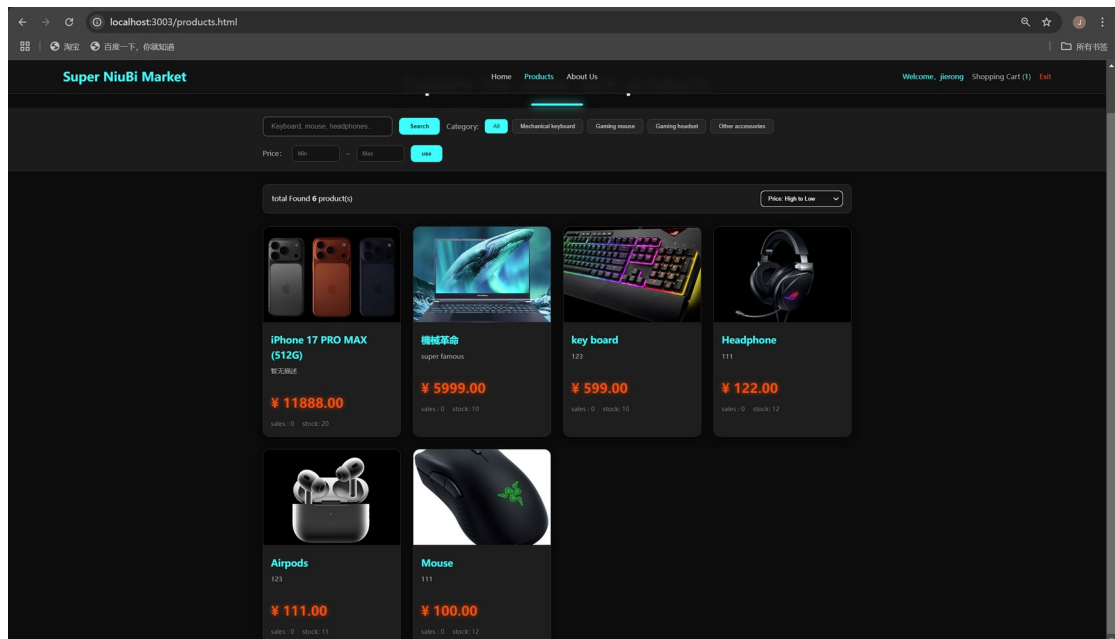
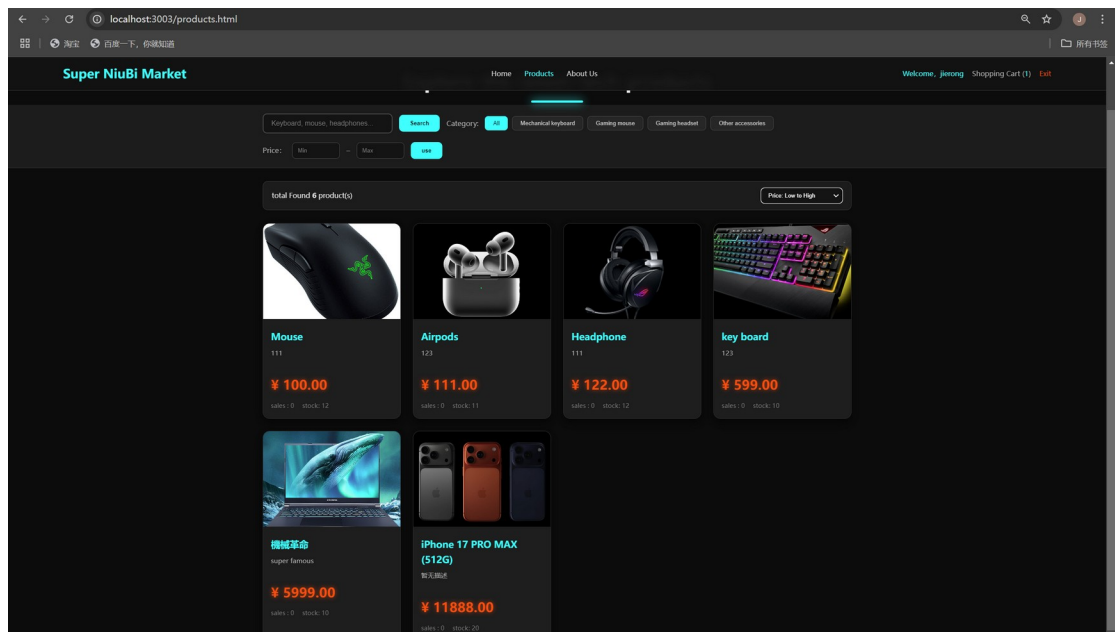


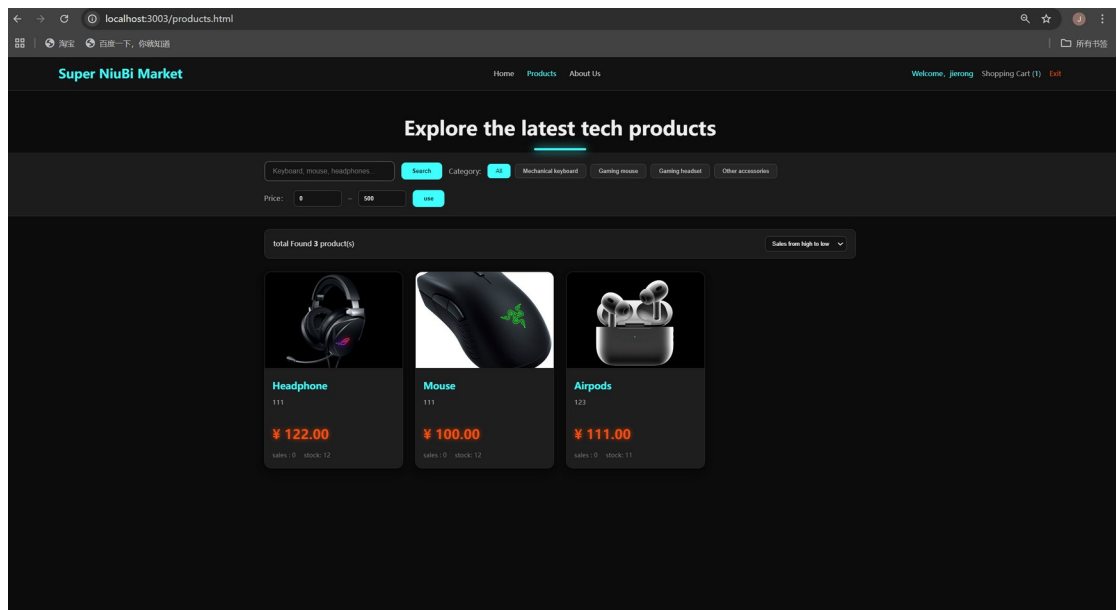
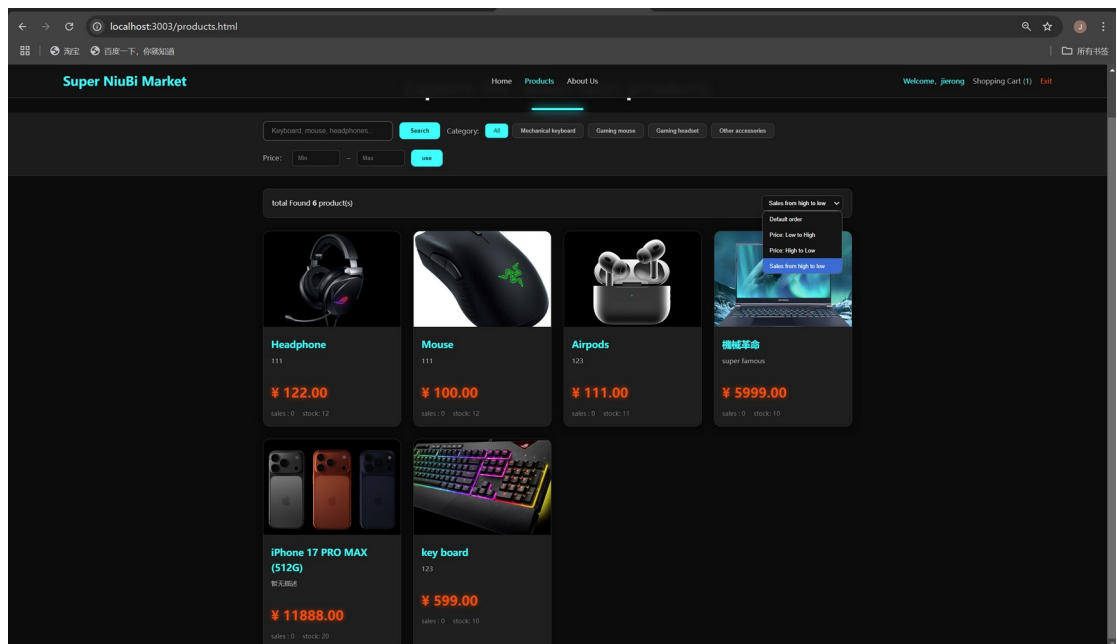


Products Page:

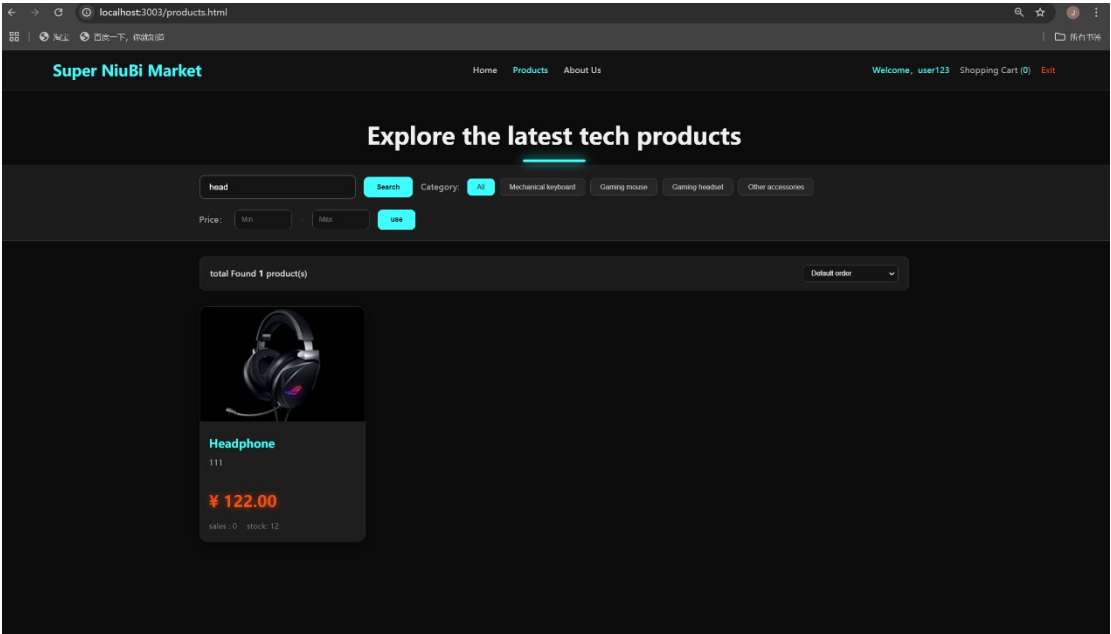


Sorting / Order:

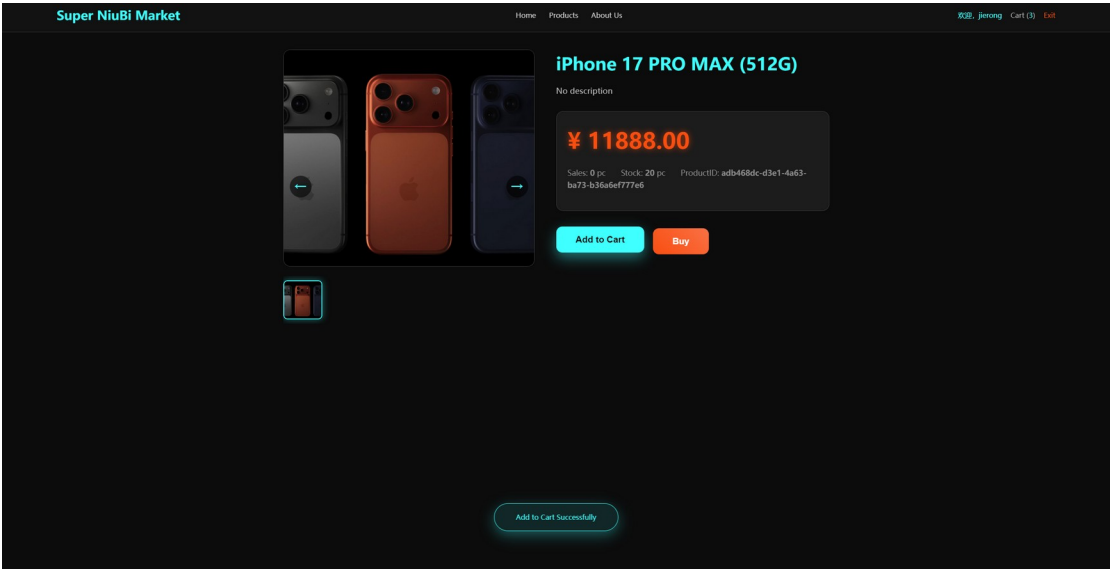




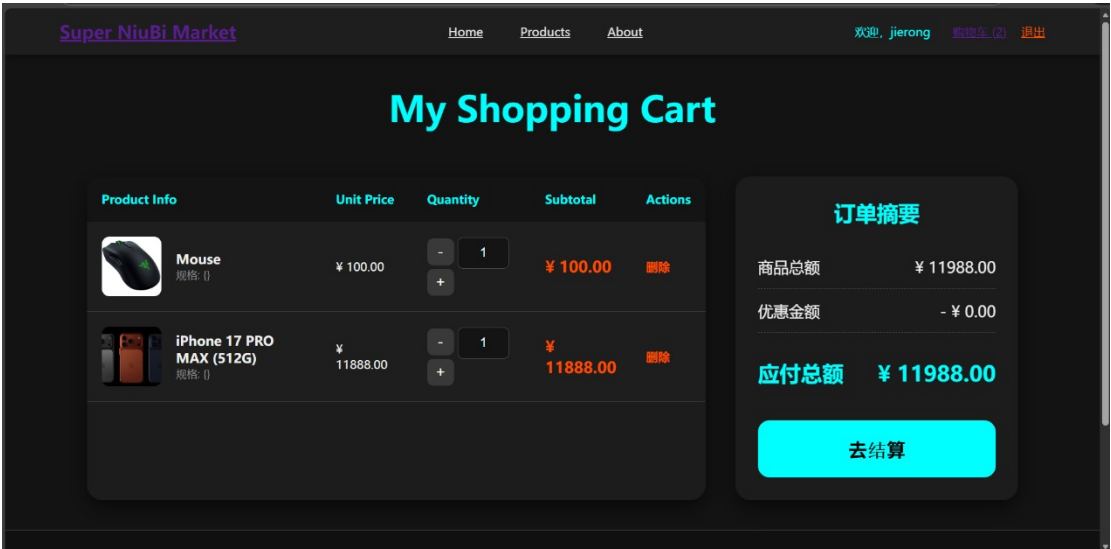
Searching:



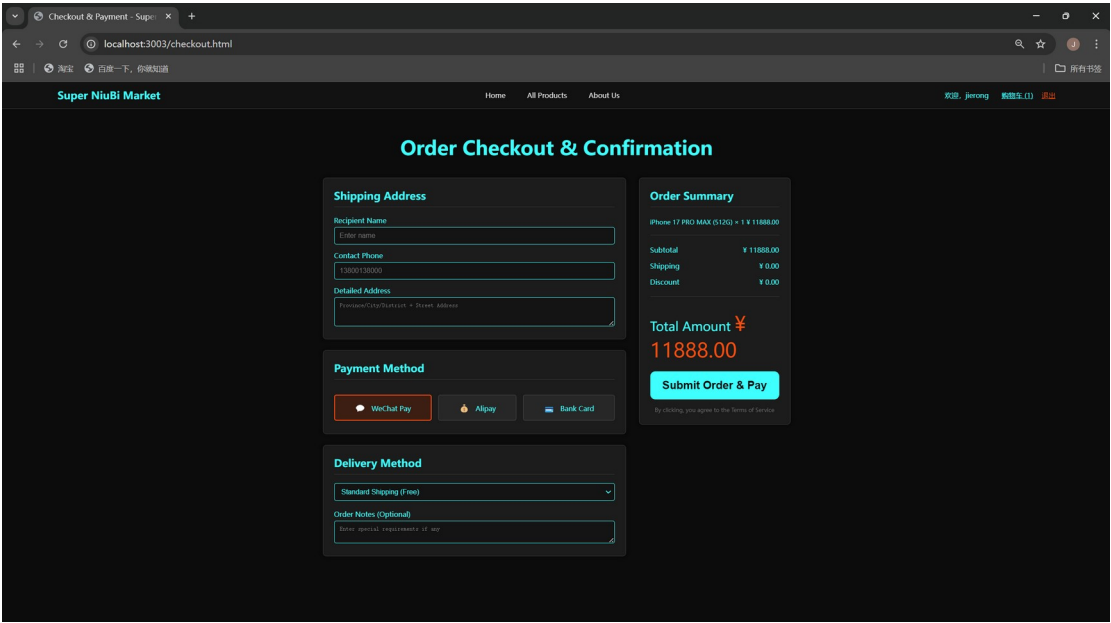
Products Detail Page:

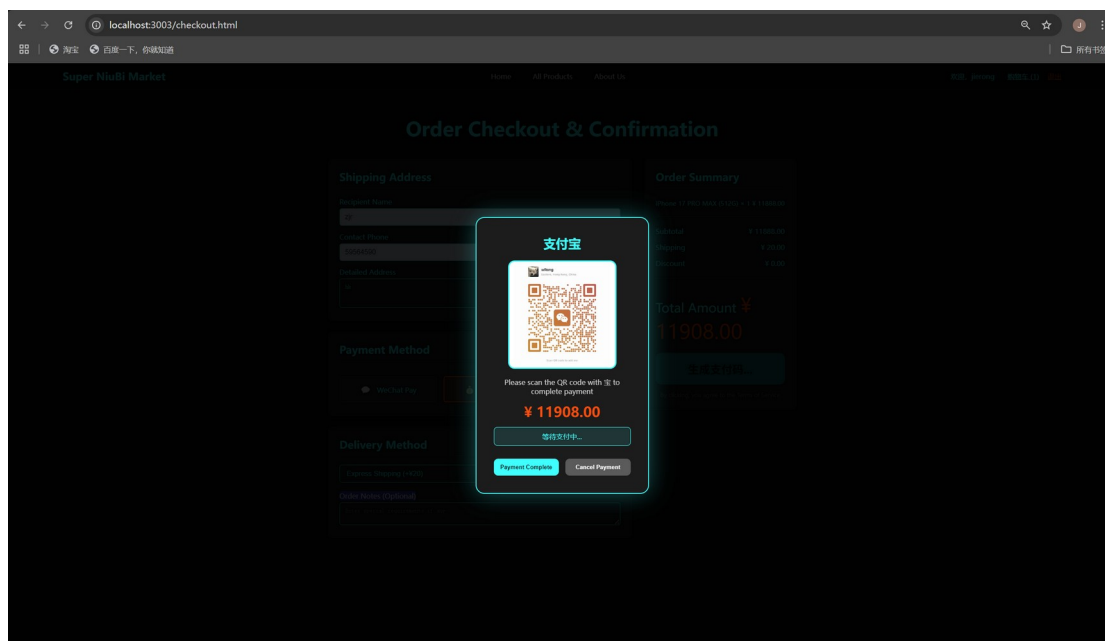


Cart Page:

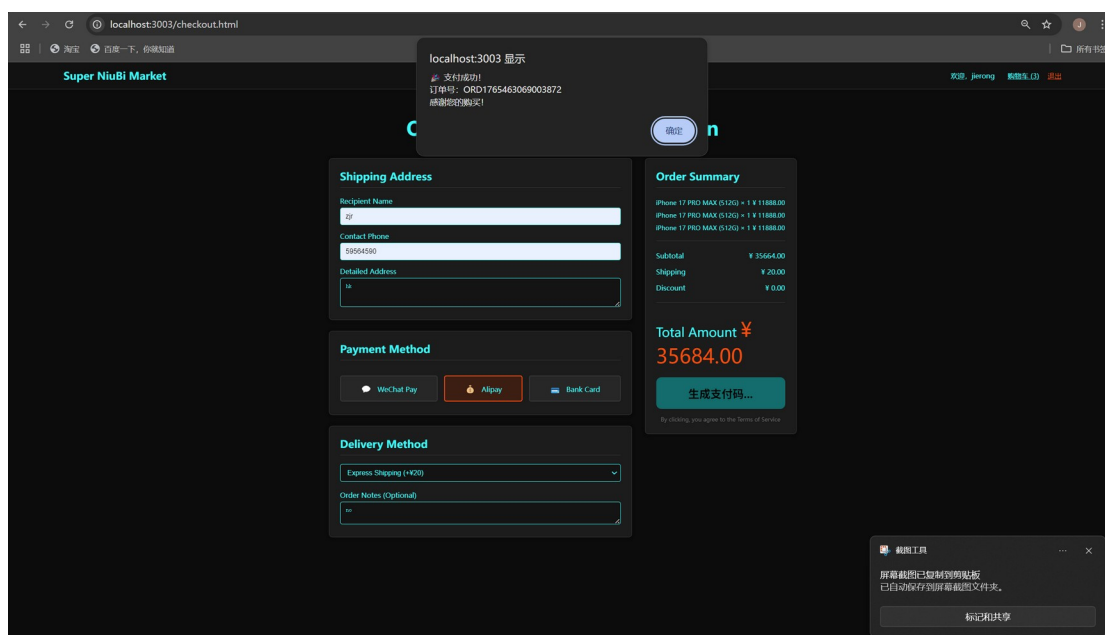


Checkout Page:

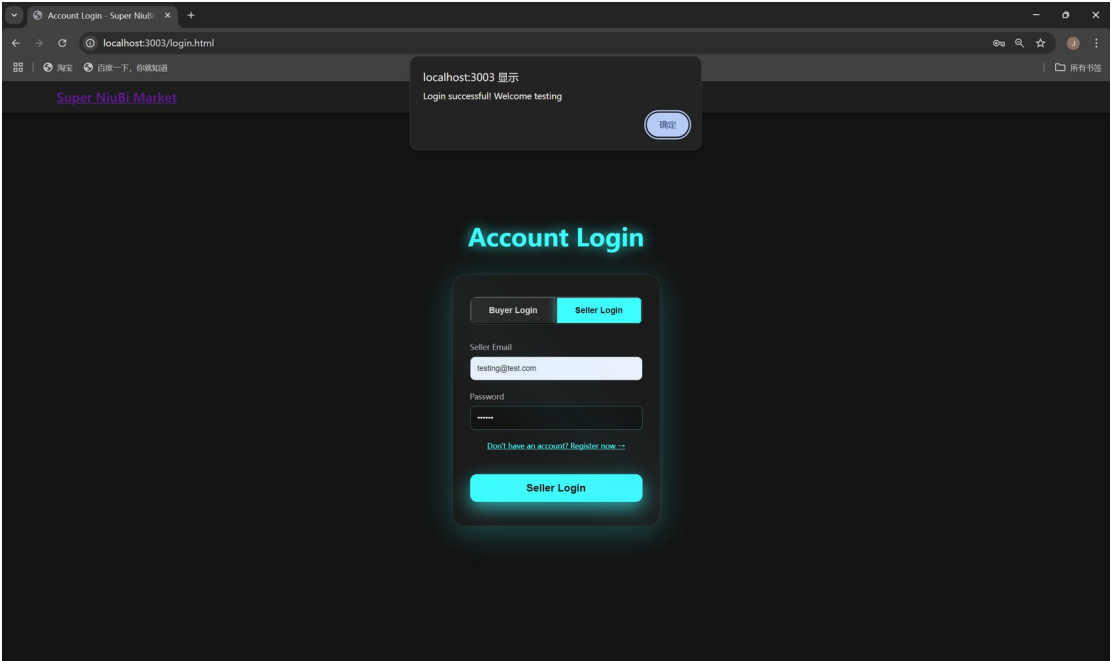




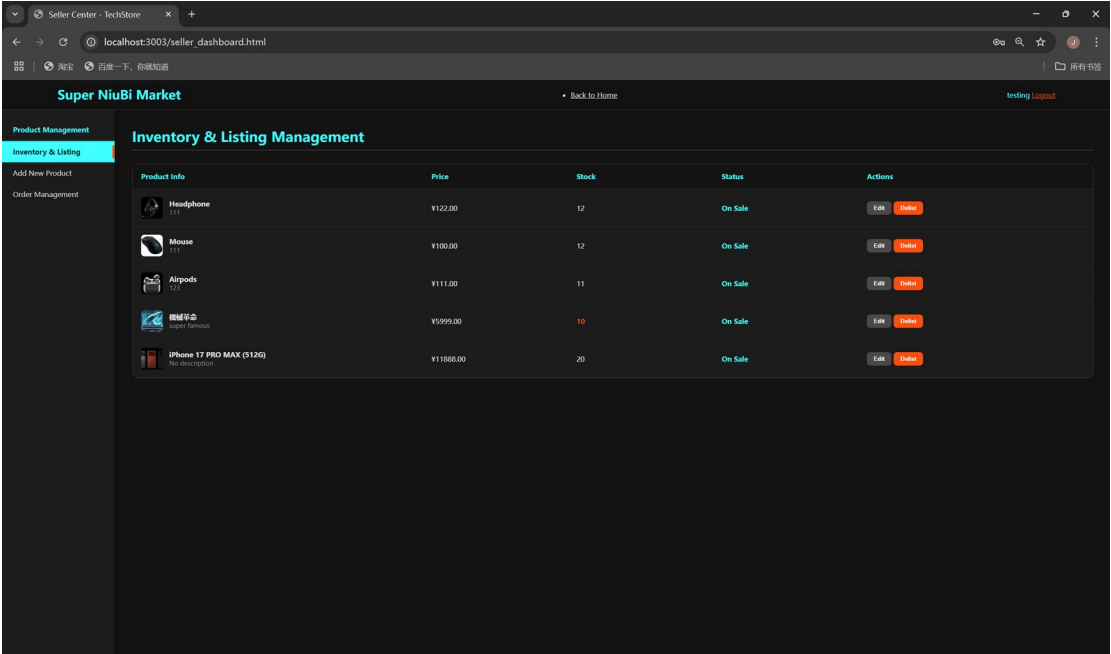
Pay Successfully Alert Page:



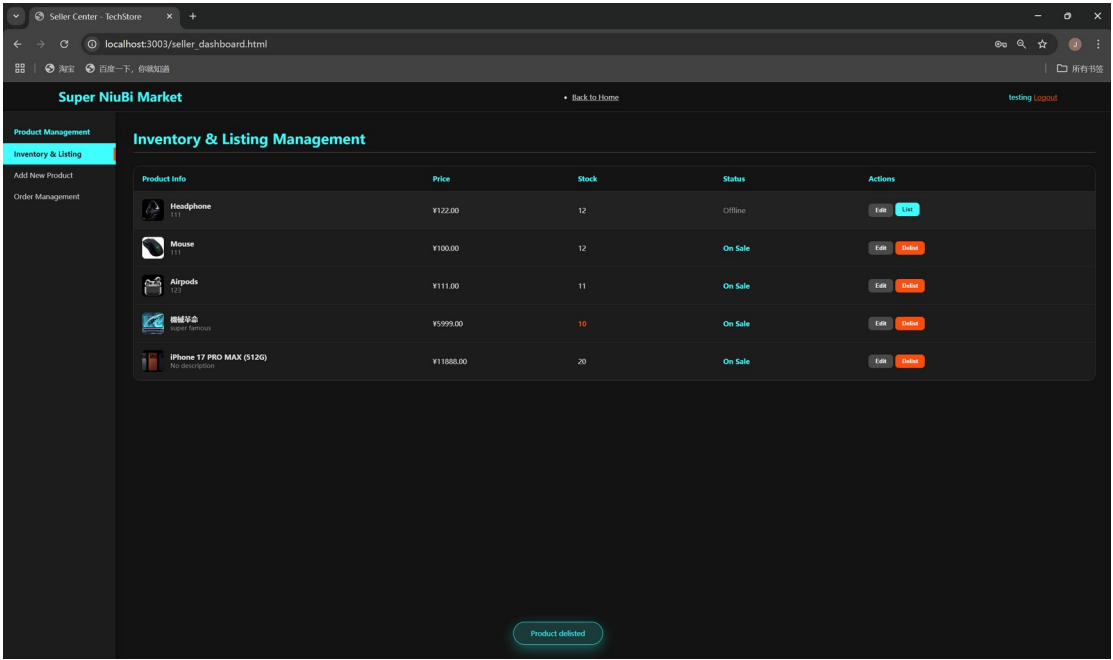
Merchant Login Page:



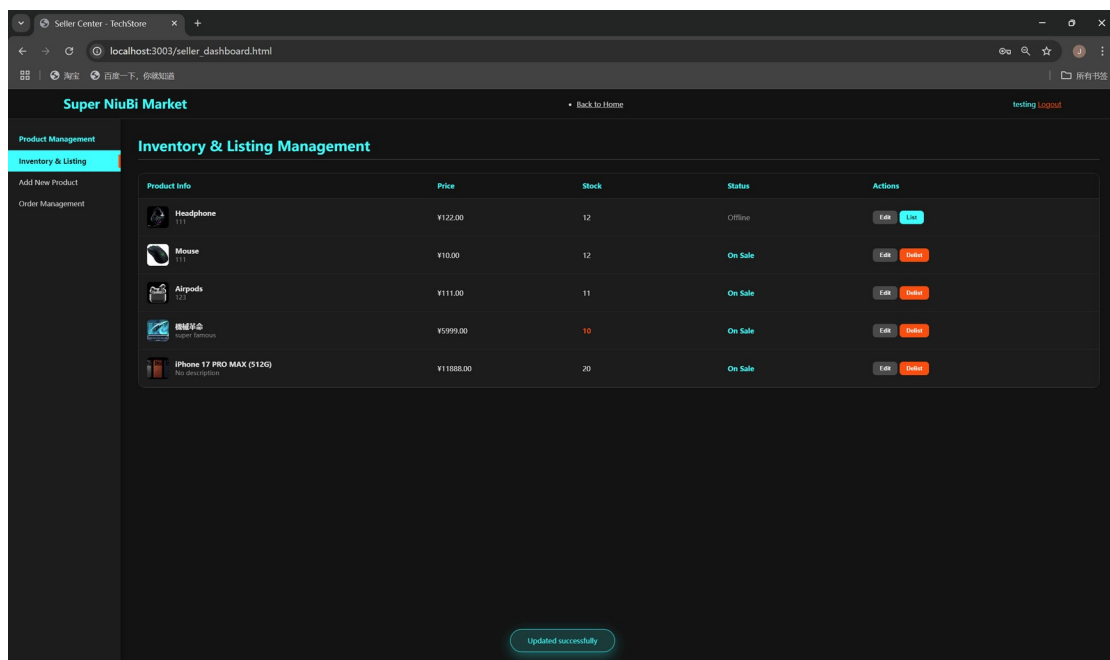
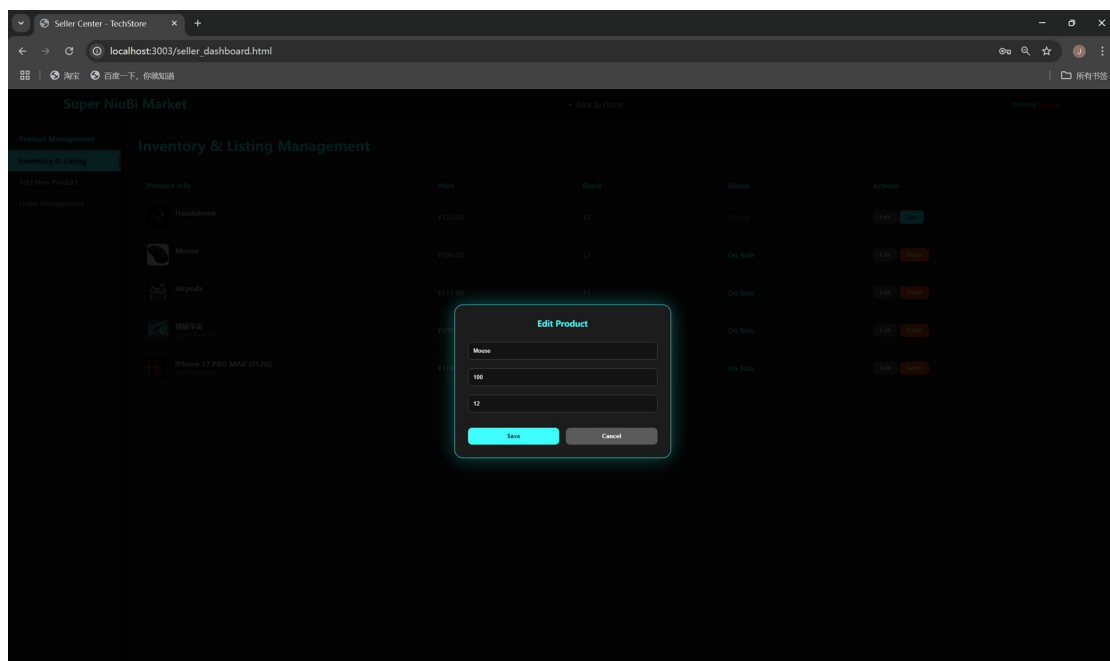
Store Products Page:



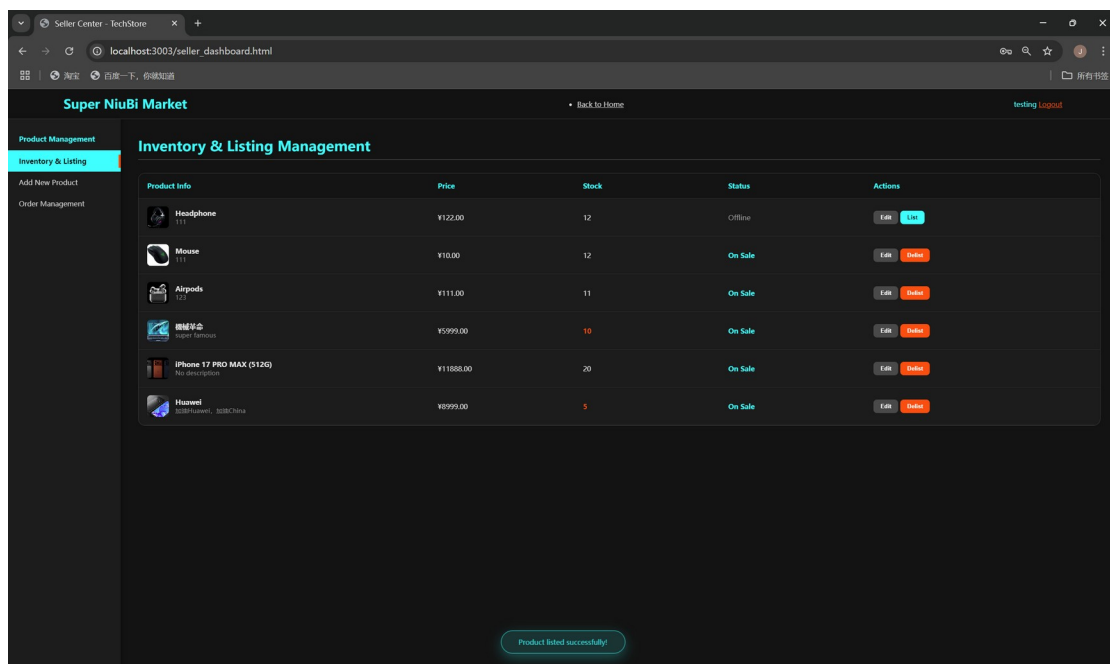
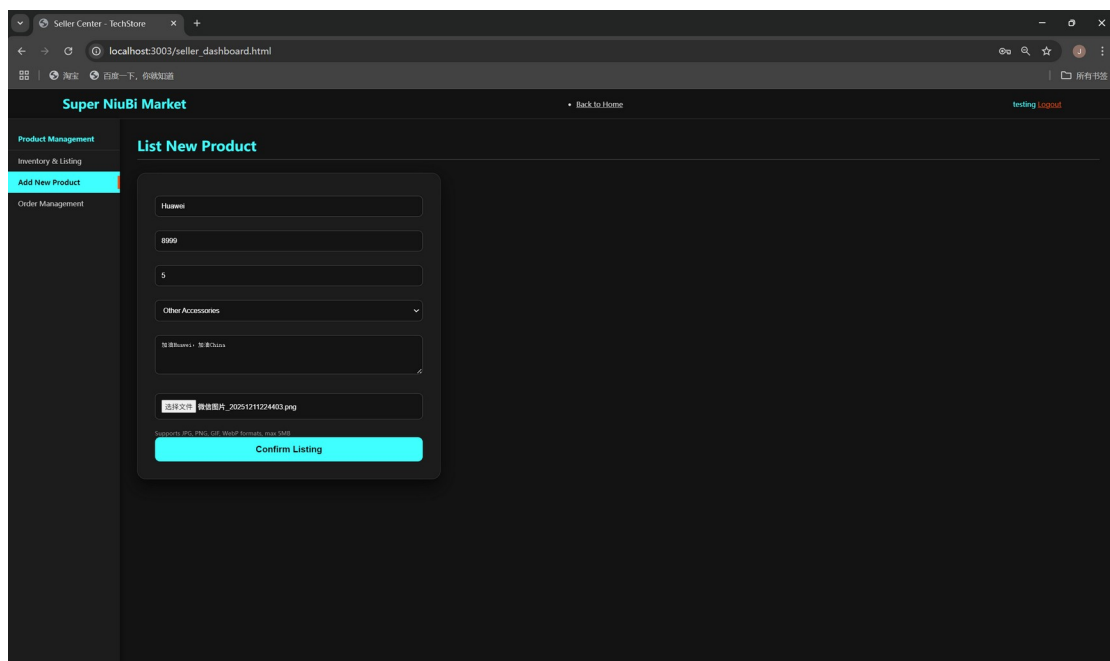
Delete Product:



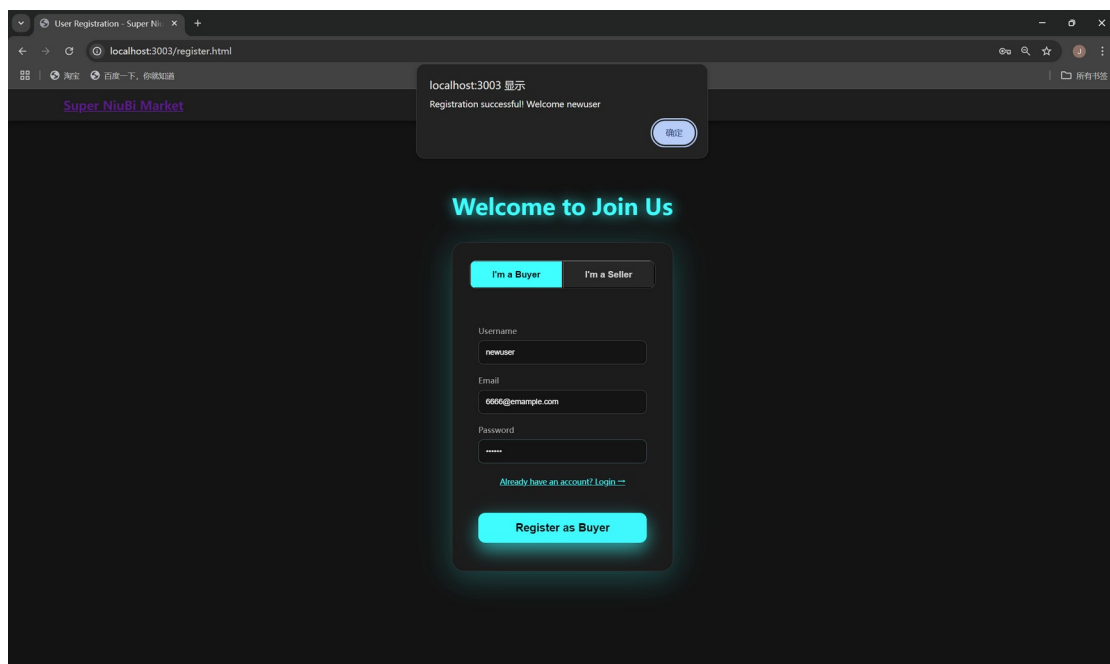
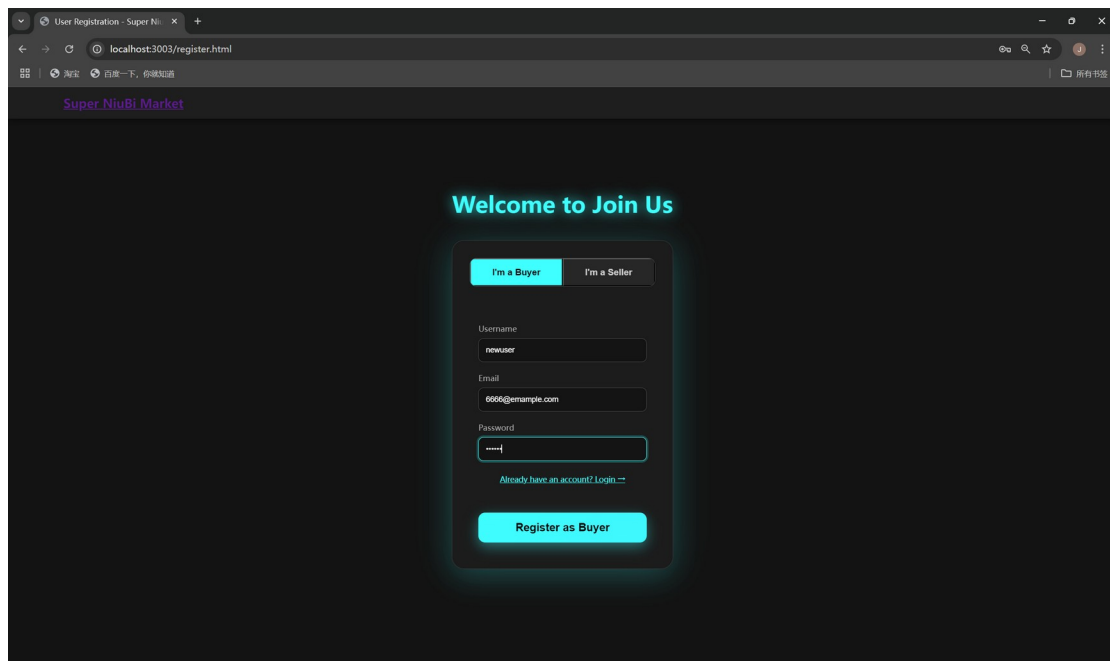
Edit Product Information:



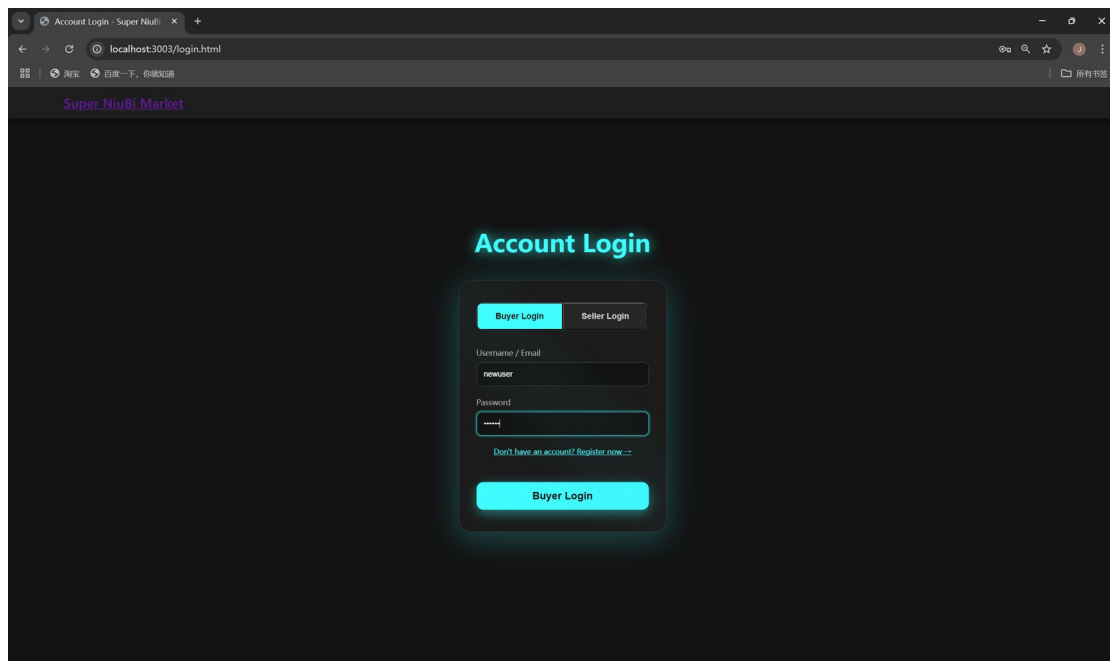
Add New Product Page:



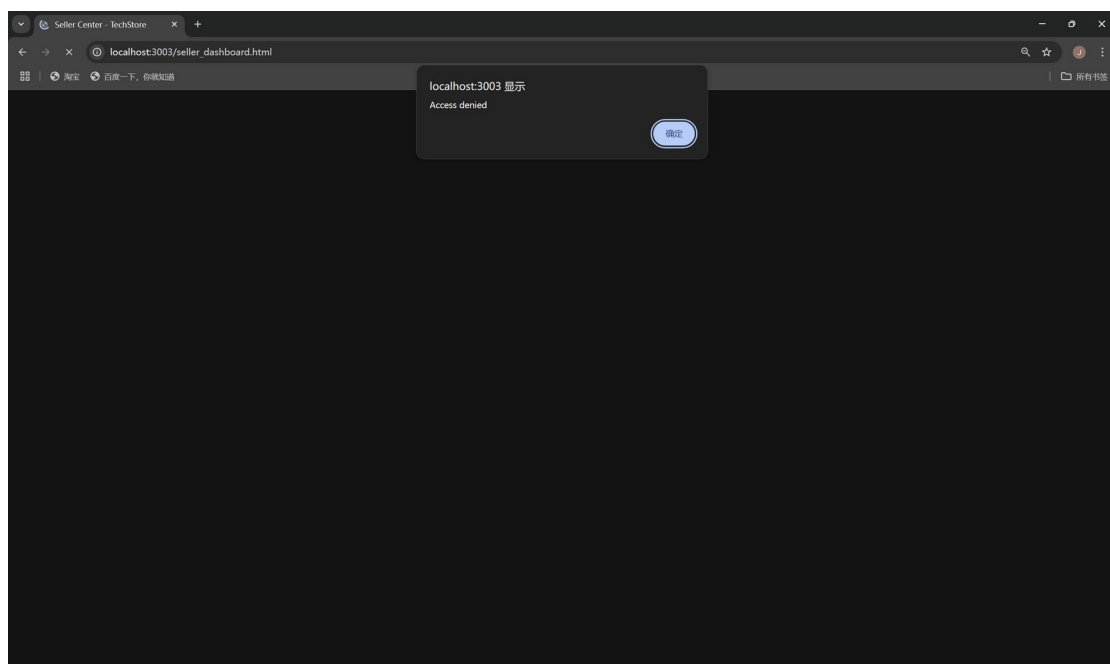
Buyer Register Page:



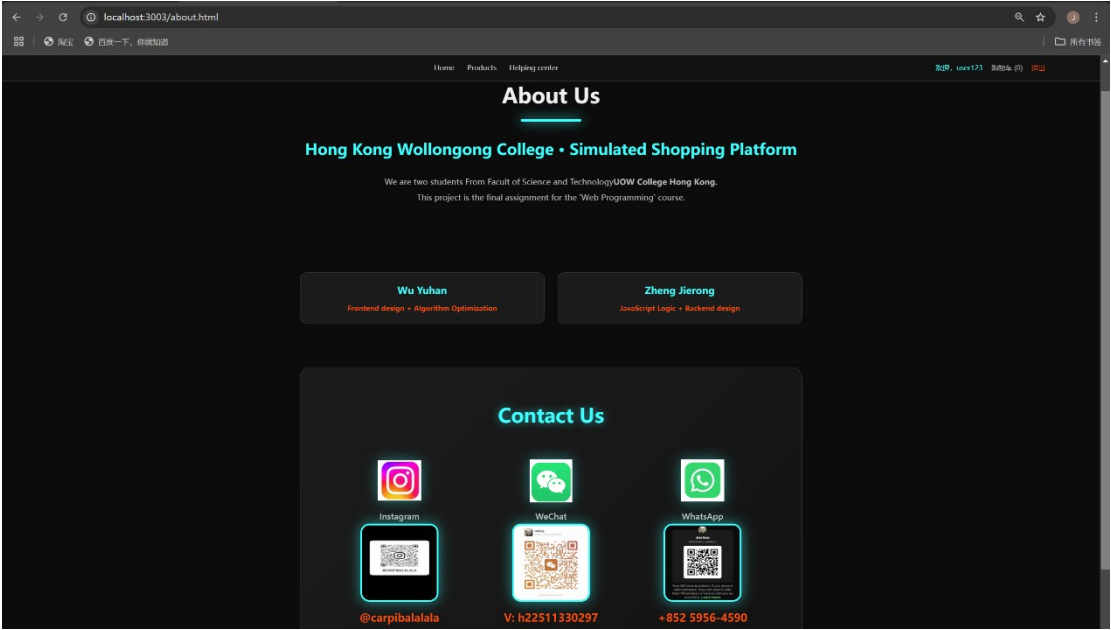
Buyer Login Page:



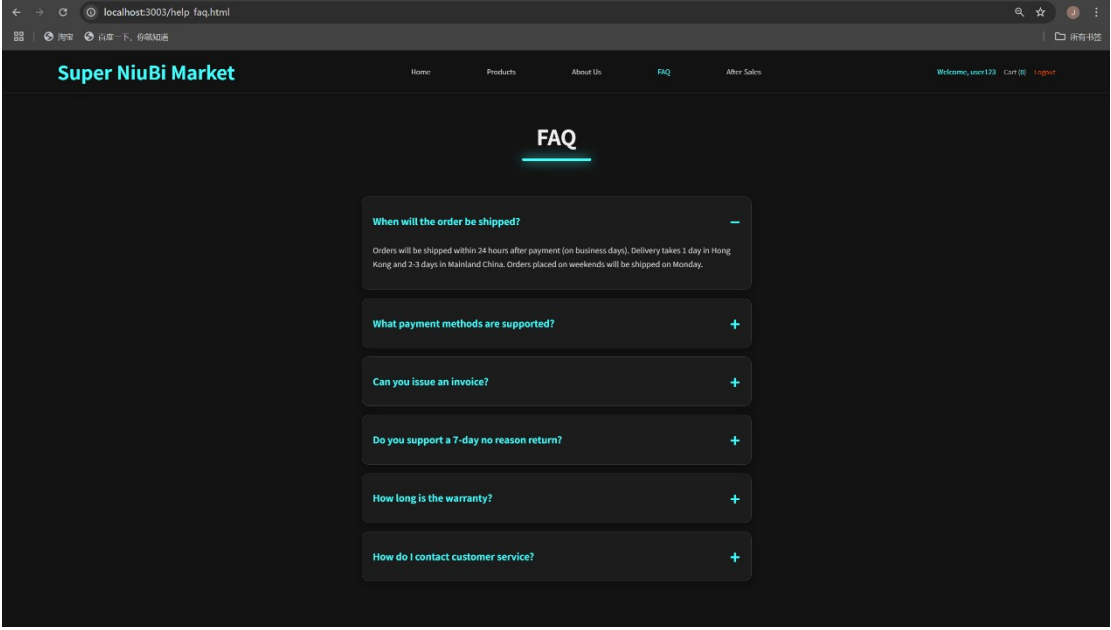
If you are user, you can't enter Merchant Center



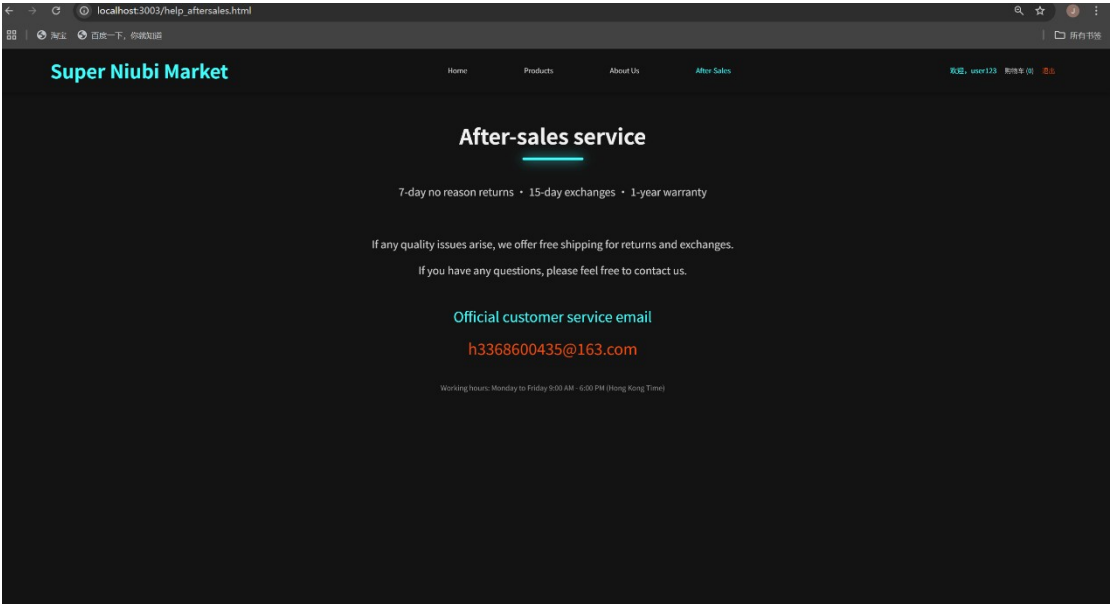
About Us Page:



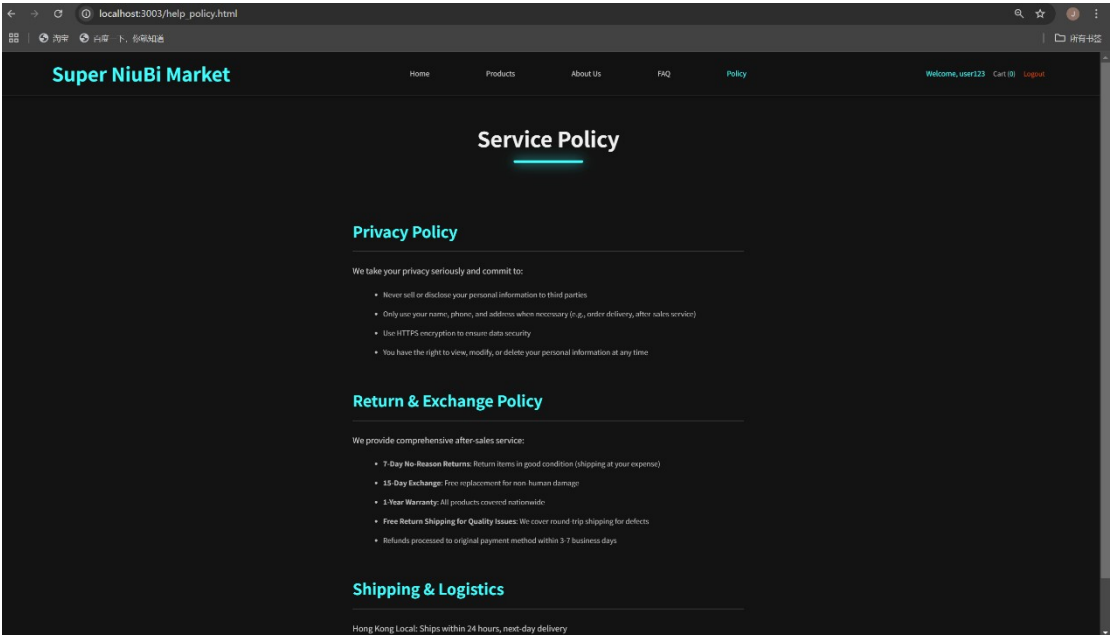
FAQ Page:



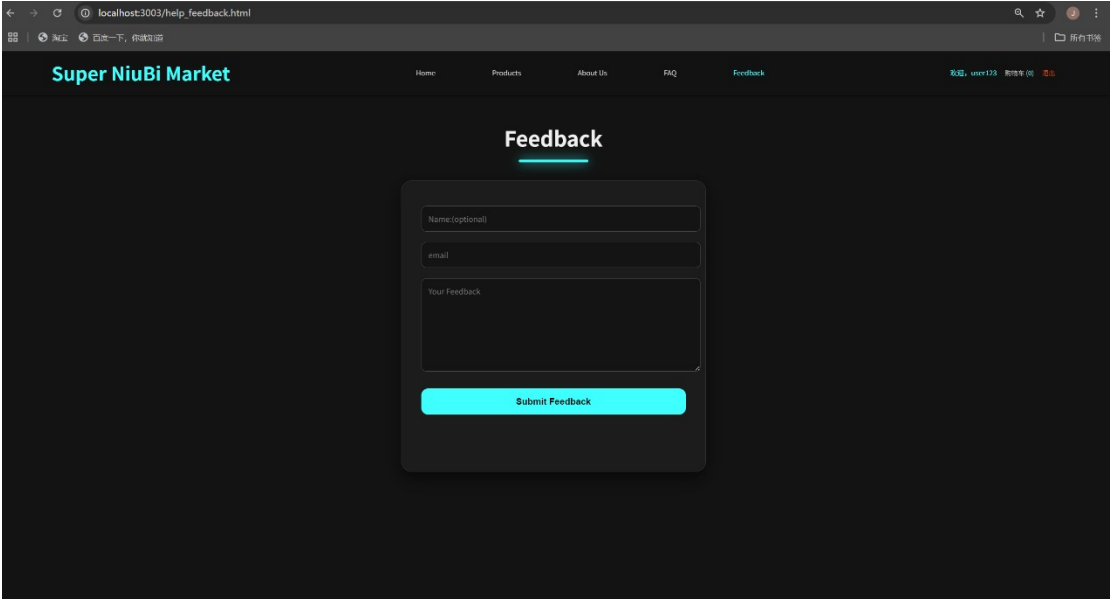
After-Sales Service Page:



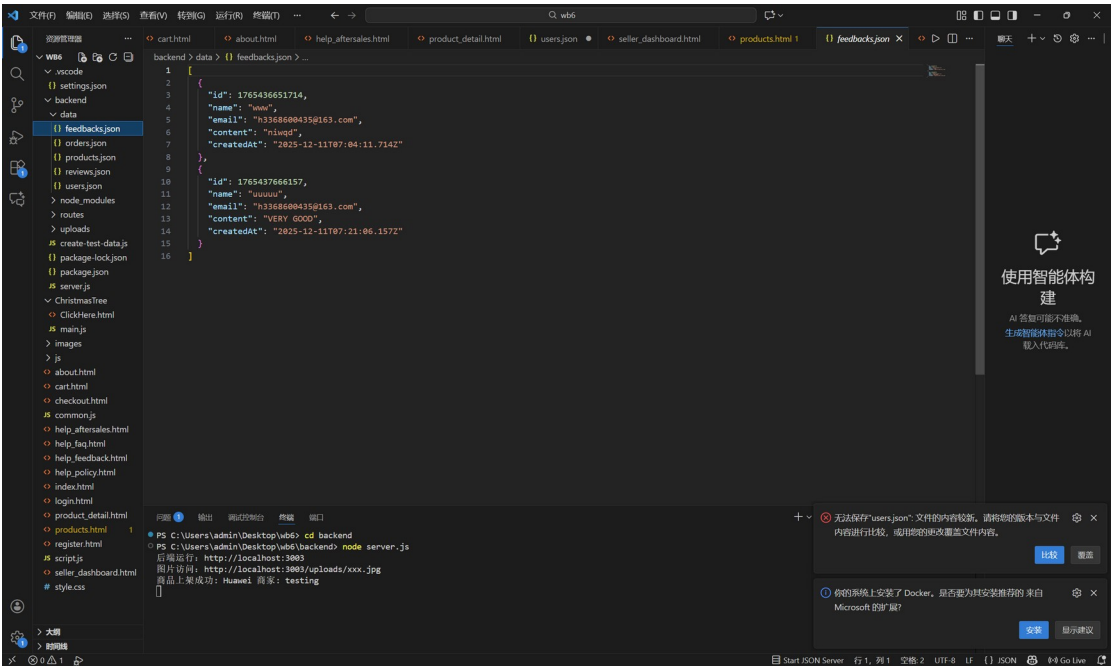
Policy Page:



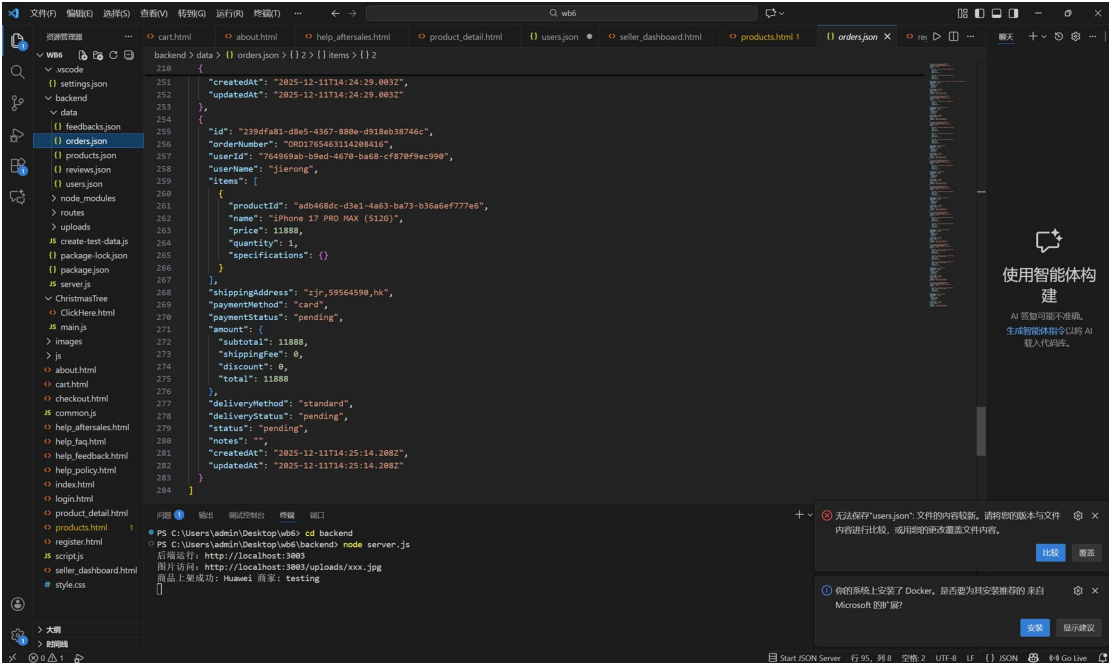
Feedback Page:



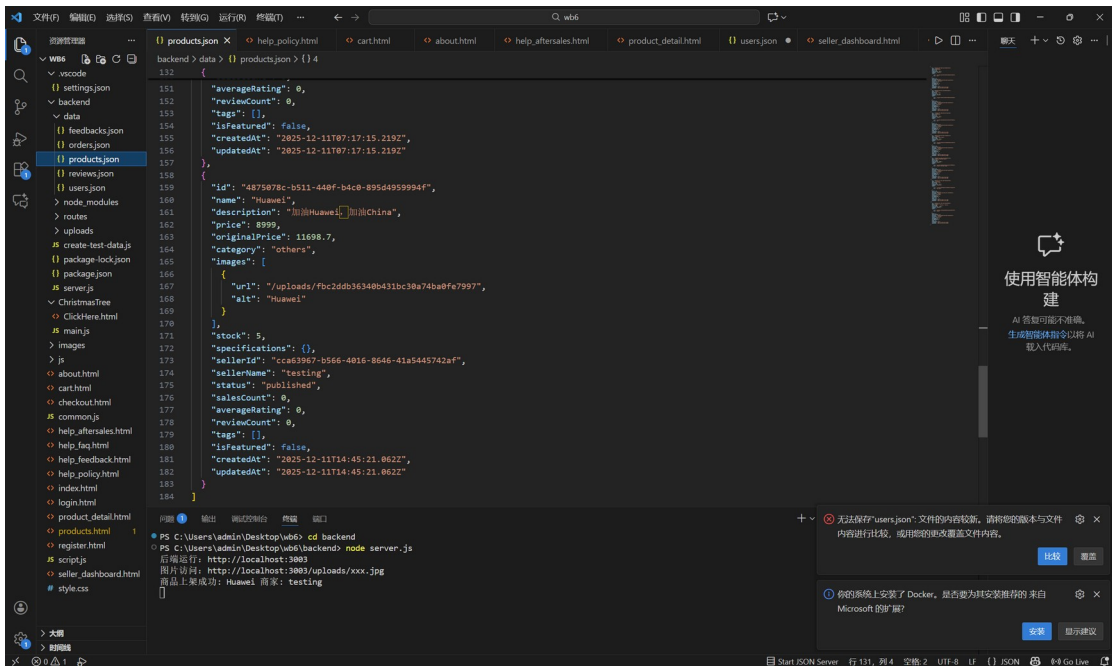
In vsCode, you can see Feedback in json file



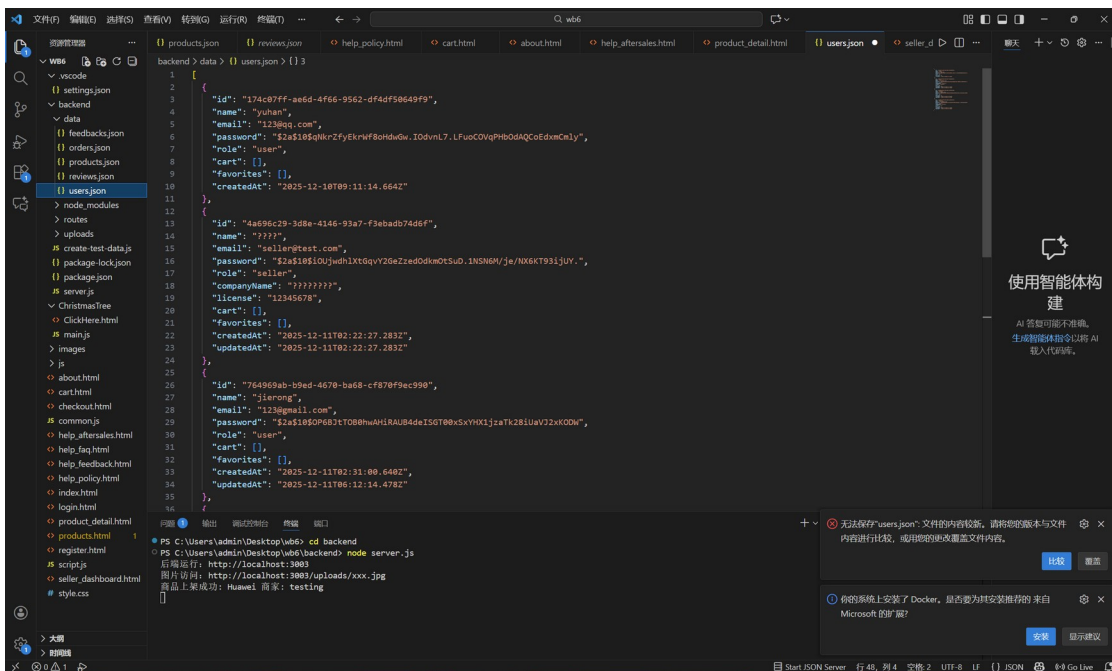
Record Order information:



Record Products information:



Record Users' Information:



Merry Christmas!

