

E-commerce Website Development Report

Kevin Zhang,Adel Leshob,Hugo Dos Reis



June 1, 2025

Seoul National University of Science and Technology (SeoulTech)

232 Gongneung-ro, Nowon-gu, Seoul, South Korea

| **Section** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Page** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1. Project Overview** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **2** |
| **1.1 Context and Objectives** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **2** |
| **1.2 Scope and Expected Outcomes** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **2** |
| **2. Architecture and Design** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **3-5** |
| **2.1 Architecture Overview** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **3-4** |
| **2.2 Design** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **4-5** |
| **3. Key Features and Technologies** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **6-7** |
| **3.1 Main Functionalities** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **6** |
| **3.2 Technologies and Tools** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **7** |
| **4. Challenges and Solutions** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **8** |
| **4.1 Problems Encountered** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **8** |
| **4.2 Solutions Implemented** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **8** |
| **5. Individual Contributions** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **9-11** |
| **5.1 Team Member 1: Kevin Zhang** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **9** |
| **5.2 Team Member 2: Adel leshob** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **10** |
| **5.3 Team Member 3 Hugo Dos Res** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **11** |
| **6. Conclusion and Future Perspectives** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **12** |

**1. Project Overview**

**1.1 Context and Objectives**

This project involved designing and developing a fully functional e-commerce website as a group assignment. The main goal was to create a secure and user-friendly online shopping platform using Node.js and the Express framework, with SQLite as the database for persistent data storage. The core objective was to apply and integrate all the web development concepts learned throughout the course into a single, practical project, including modular code design, secure user authentication, and session management.

**1.2 Scope and Expected Outcomes**

The application aims to replicate core e-commerce functionalities expected in a real-world site. By the project’s completion, the team delivered a working prototype enabling users to:

* Register and log in securely
* Browse and search products
* Add, update, and remove items in a shopping cart
* Complete a simulated checkout process
* Persistently manage user and product data
* Fully functional admin panel to manage the website

This project also provided valuable experience in collaborative development, integrating front-end and back-end technologies, and implementing security measures in web applications.

**2. Architecture and Design**

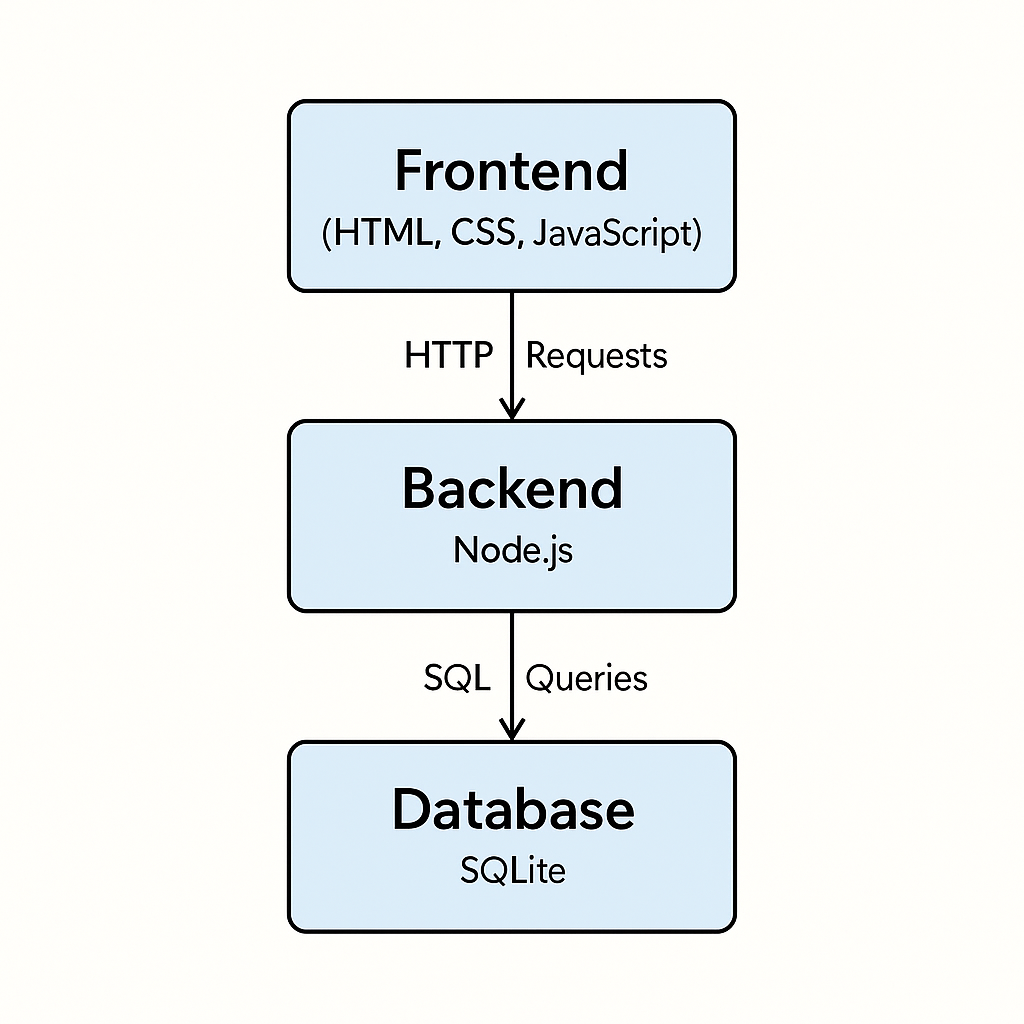
**2.1 Architecture Overview**

The e-commerce project is structured following a classic client-server architecture comprising:

* **Frontend (Client):** User interface built with HTML, CSS, and JavaScript, enabling browsing, searching, cart management, and user authentication.
* **Backend (Server):** Server-side application responsible for business logic, user management, product handling, order processing, and database communication.
* **Database:** Relational storage (sqlLite) for product, user, order, and session data.

This layered architecture ensures clear separation of concerns, improves maintainability, and facilitates future scalability.

**Image: System Architecture**

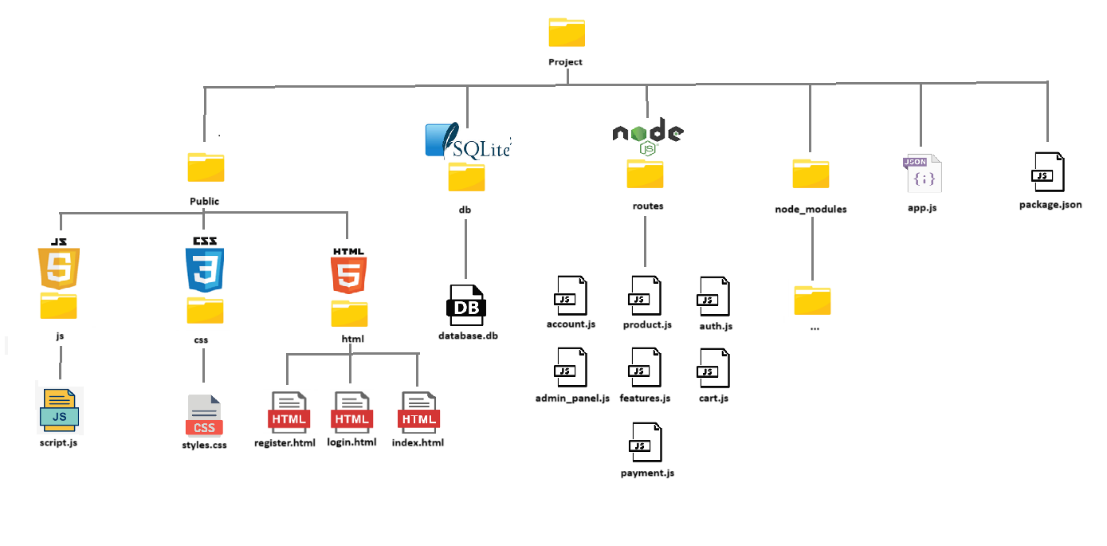


This diagram shows the main parts of the e-commerce app and how they work together. The front end talks to the back end, which handles user actions and connects to the database to store data.

**The Importance of Folder Structure**

A well-structured folder system is essential for keeping your project organized, readable, and easy to maintain. It helps teams collaborate efficiently, simplifies debugging, and ensures scalability as the project grows.

**Image: Files Architecture**

****

This structure shows how files are organized in our e-commerce website project. It separates code, assets, and configuration to keep the project clean and easy to manage.

**2.2 Design**

**About the E-commerce Website**

The e-commerce website features a clean and modern design with a homepage showcasing featured products and promotions. Users can browse a categorized product catalog, manage their cart, and complete purchases through a secure checkout. An admin panel allows product and order management, making the platform functional for both customers and administrators.

**Image: Website Overview**

**A screenshot of a video game

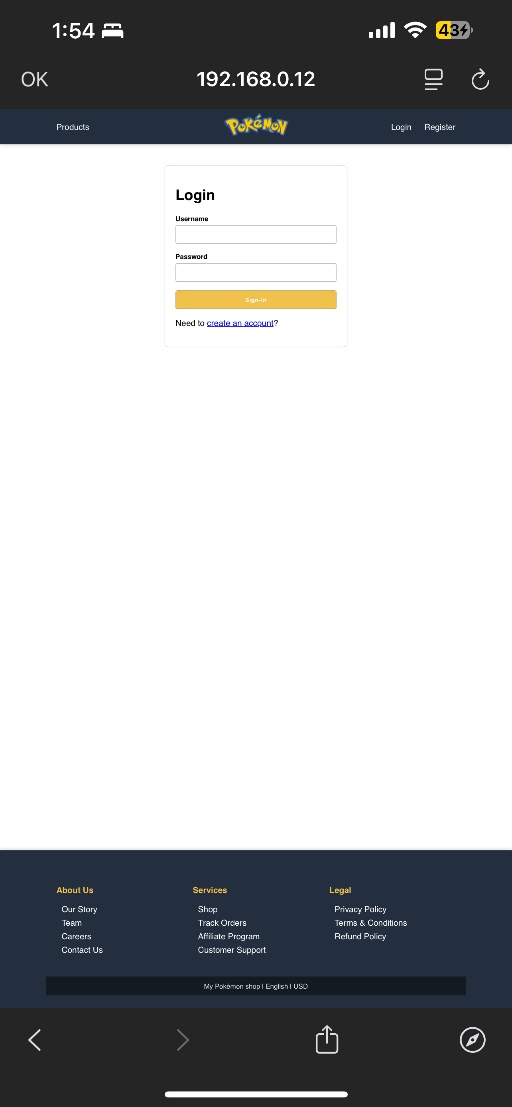
AI-generated content may be incorrect.**

A screenshot of a video game

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Une image contenant texte, capture d’écran, logiciel, Page web

Le contenu généré par l’IA peut être incorrect.

**3. Key Features and Technologies**

**3.1 Main Functionalities**

This section highlights the main features of the e-commerce application, and the key technologies used to build it. The focus is on usability, security, and performance.

**The main features of the application are:**

* **Homepage**: A landing page showcasing featured products, and a clear navigation menu.
* **Product Catalog**: Displays products with images, descriptions, and prices. Includes search and filtering options, and product categorization.
* **Individual Product Page:** Display the selected product with image, description, and price.
* **Shopping Cart**: Allows users to add, remove, and update items in their cart. Automatically calculates the total cost including taxes and shipping.
* **User Authentication**: Supports user registration, login, and profile management.
* **Checkout Process**: Provides a secure checkout process integrated with a mock payment gateway.
* **Admin Panel**: Enables management of products, user accounts, and orders.

**Additional development goals included:**

* Writing clean, modular, and maintainable code.
* Implementing secure user authentication with password hashing using bcrypt.
* Managing user and product data through the SQLite database.
* Creating a responsive user interface for enhanced user experience.

**3.2 Technologies and Tools**

For the development of the e-commerce website, the following technologies and tools were used to ensure an efficient application:

* **JavaScript**: The core scripting language used to implement dynamic behavior and interactivity on the client side.
* **Node.js**: A JavaScript runtime environment used to build the server-side of the application, enabling fast and scalable back-end development.
* **Express**: A lightweight and flexible web framework for Node.js, used to simplify the creation of server-side logic, handle routing, and manage HTTP requests and responses efficiently.
* **Bcrypt**: A password-hashing library used to securely store user passwords by applying a one-way encryption algorithm, which includes salting and multiple hashing rounds to protect against brute-force and rainbow table attacks.
* **HTML & CSS**: The foundational technologies for structuring and styling the web pages, ensuring a responsive and visually appealing user interface.
* **SQLite**: A lightweight, file-based relational database management system used for storing product data, user information, and transaction records.
* **GitHub**: The version control platform used to manage the project's source code, track changes, and facilitate collaboration through repositories.

These technologies were selected for their practicality and reliability, enabling efficient development and solid performance.

**4. Challenges and Solutions**

|  |
| --- |
| **4.1 Problems Encountered** |
| Understanding the Express.js Framework  Express.js is a powerful and flexible Node.js framework. However, at the beginning, we struggled to understand how middleware, routing, and asynchronous code worked. Features like request handling, session management, and dynamic HTML generation were unfamiliar, making the learning curve steep.  Using SQLite and SQL Queries  While SQLite is lightweight and easy to set up, integrating it with Node.js required careful handling. We encountered issues with asynchronous data access, and managing complex queries across multiple related tables—such as joining users, products, and orders—often led to unexpected bugs or performance problems.  Implementing CSS for a Good Design  Designing a responsive and modern-looking interface was not straightforward. We had to deal with layout inconsistencies, mobile responsiveness, and ensuring a unified look between different pages. In addition, balancing visual appeal with accessibility (like button sizes, contrast, and readability) took several iterations.  Maintaining Modular and Reusable Code  As the project grew, keeping the code modular and avoiding repetition became more difficult. We often found ourselves copying similar logic between routes or pages, which made debugging and updating features more error-prone.  Handling User Session and Access Control  Implementing proper access control, especially distinguishing between regular users and admins, proved to be more complex than expected. We had to ensure that users could only access their own data while admins could manage everything securely.  **4.2 Solutions Implemented** |

Learning Express Step by Step

We followed a hands-on learning approach—starting with basic routes, then gradually integrating middleware like body-parser, session management, and static file serving. Reading the official documentation and experimenting with small code examples helped build our understanding.

Writing and Testing SQL Queries Thoroughly

To reduce bugs, we wrote queries incrementally and tested them in isolation using database browsers or simple Node.js scripts. We also modularized database access functions to keep our routes clean and easier to debug.

Improving CSS Styling with Iterative Design

We used a mobile-first approach and incorporated responsive techniques like flexbox and media queries. Color schemes and layout spacing were refined through user feedback and design inspiration from real e-commerce sites. We also used consistent button styles, shadows, and hover effects to improve interactivity.

Modularizing Code and Separating Concerns

We refactored repetitive logic into helper functions and structured our routes across different files. Middleware functions were reused for things like authentication checks, and layout components like headers and footers were kept in shared code.

Securing Access Control and Sessions

We implemented session-based authentication and role-based access checks using middleware. Admin routes required extra checks to prevent unauthorized access, and we ensured user-specific data like carts and orders were only accessible by the correct user.

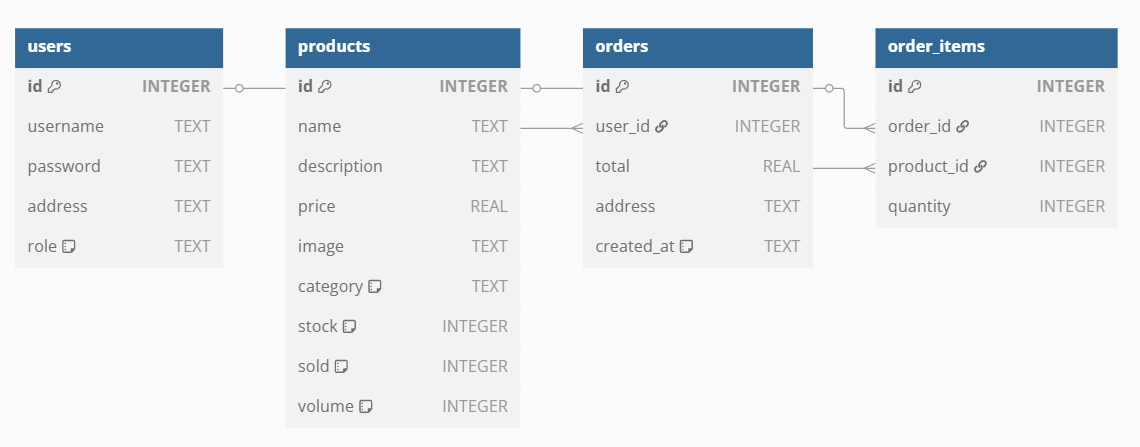
|  |
| --- |
| **5. Individual Contributions** |
| **5.1 Team Member 1: Kevin Zhang** |

Kevin Zhang was responsible for back-end development, focusing on the database layer of the e-commerce platform. He designed and implemented the database schema using SQLite, structuring product details, user data, shopping cart contents, and order history.

He integrated back-end logic with the database through efficient SQL queries for CRUD operations. Additionally, Kevin developed a fully functional admin panel to manage products, users, and orders directly from the database.

**Database:**

**Image: DataBase Overview**



The diagram above shows the structure of the database used in the application. It has been designed to manage a simple system for handling users, products, and orders.

**Table Descriptions**

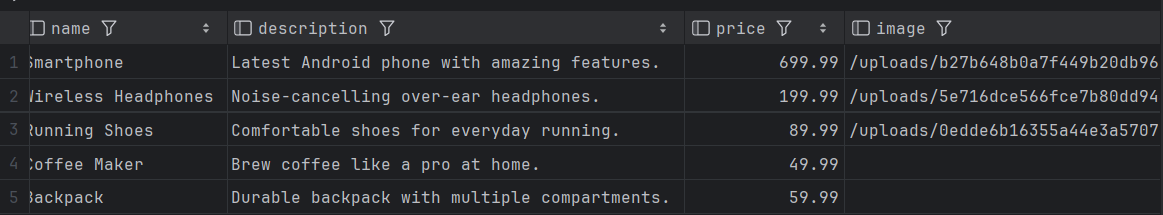
* **users**: Contains information about registered users, including their username, password, address, and role (with a default value of "user").
* **products**: Stores available products along with their details such as name, description, price, image, category, stock quantity, number of units sold, and volume.
* **orders**: Represents orders placed by users. Each record includes the total amount, delivery address, and a creation timestamp.
* **order\_items**: Intermediate table linking orders to products, specifying the quantity of each product in a given order.

**Relationships Between Tables**

* A user can place multiple orders.
* An order can contain multiple products, and each product can appear in multiple orders. This many-to-many relationship is managed via the order\_items table.

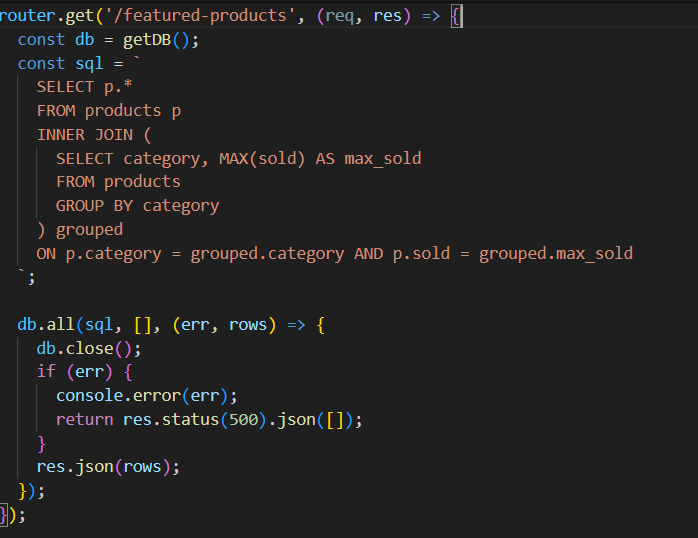
This relational design supports clear data organization and efficient handling of CRUD operations within the application.

**Image: DataBase exemple**



Here you can see an example of what the database looks like and the type of information it contains.  
It stores all the items sold on the website, including the product name, price, and image URL.

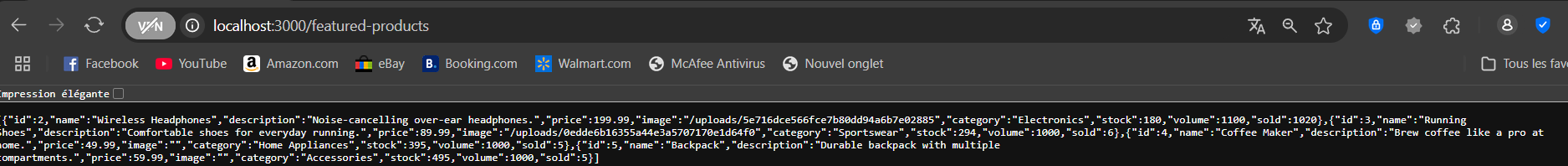
**Image: Code snippet**



For example, this code retrieves data from the database to find the most popular product in each category. The result is stored in a variable and then sent as a JSON response.

In this case, the route is called /featured-products. So, if you go to your browser and visit the following URL:

**Image:** <http://localhost:3000/featured-products>



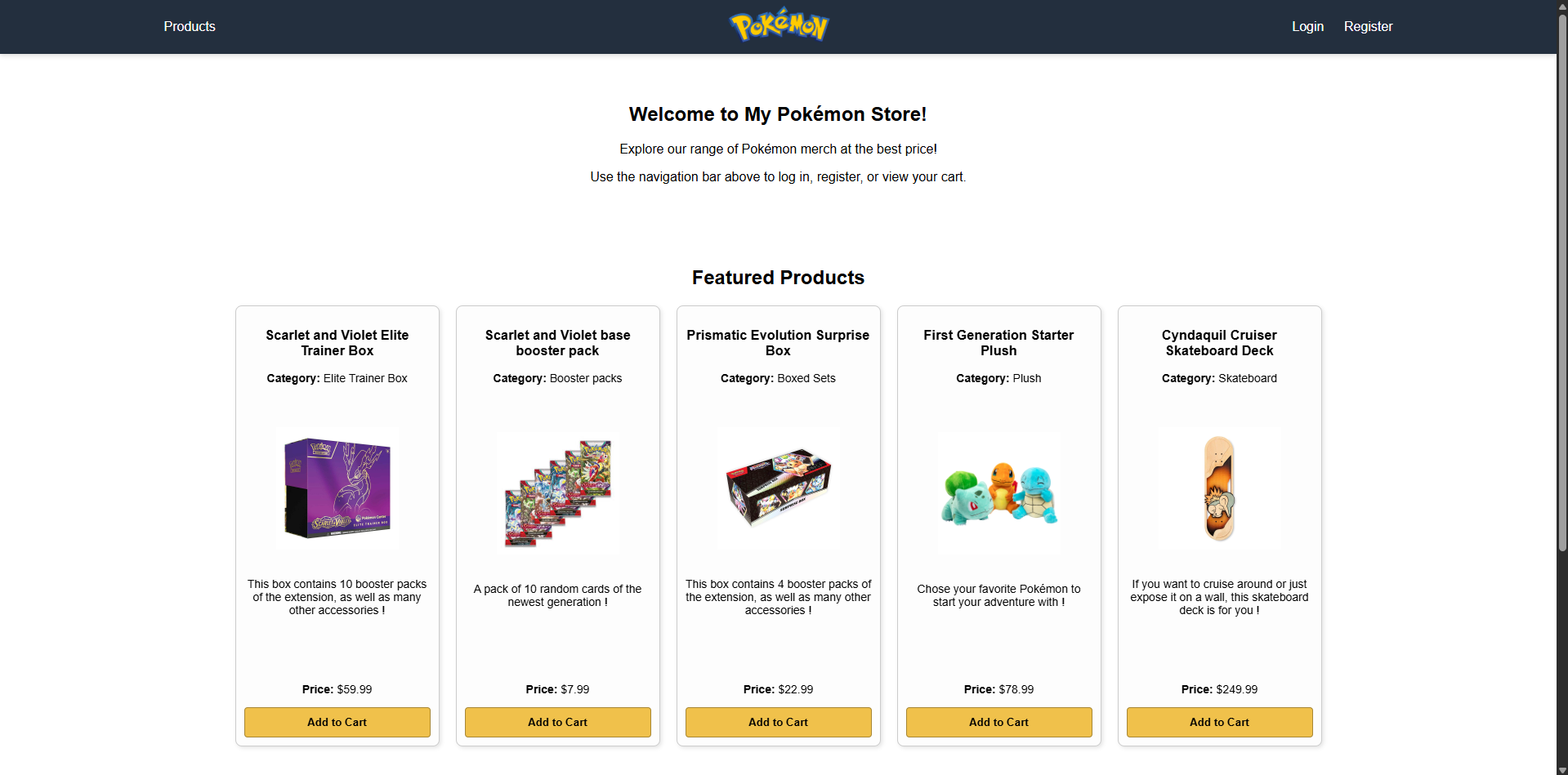
you'll see the JSON data directly in the browser.

This JSON can then be used however you want, such as fetching it from the frontend using JavaScript to display the featured products on your website.

**5.2 Team Member 2: Adel Leshob**

Adel Leshob focused on front-end development using HTML, CSS, and JavaScript to create a user-friendly and responsive interface. His work ensured that visitors could navigate the site smoothly across different devices.

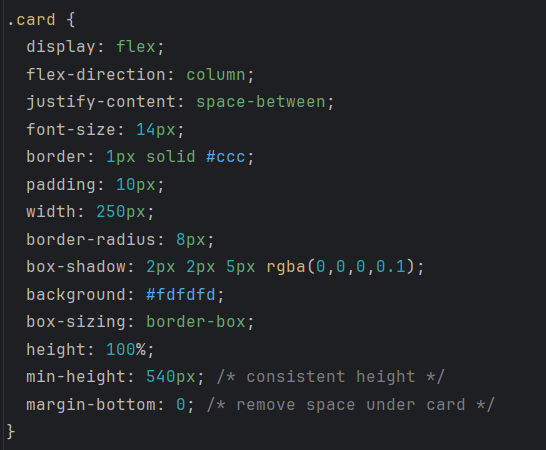
**Image: HomePage view**

****

The image above shows the homepage as designed by Adel. It features a clear layout with easy access to the product catalog, shopping cart, and other key sections, following common e-commerce design principles.

Adel also implemented essential components like the product catalog, shopping cart, and checkout forms. The code snippet below demonstrates how these elements were built to interact with dynamic data, ensuring that product information and user details are displayed accurately.

**Image: Code snippet**

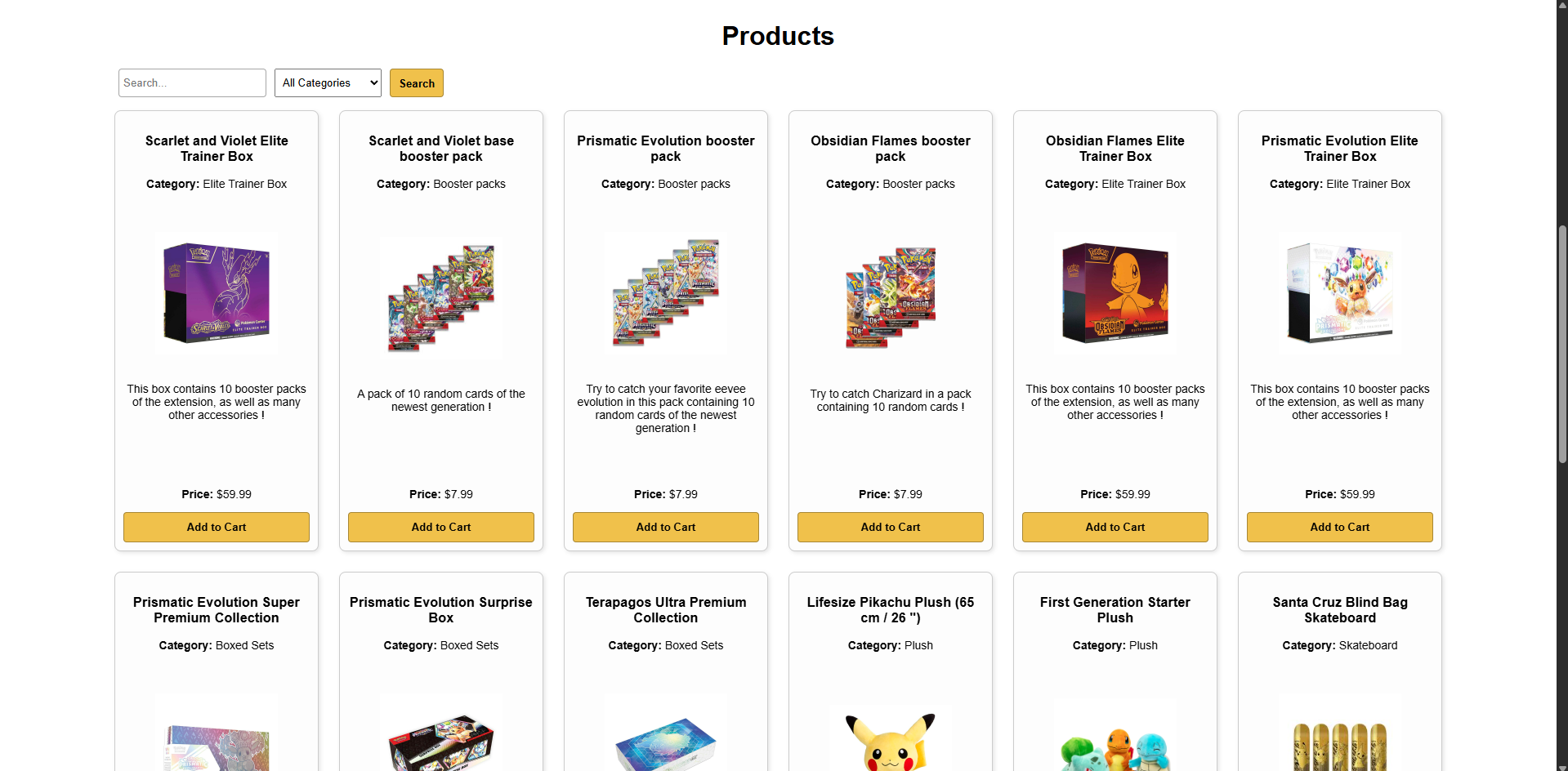
Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

This snippet highlights the use of CSS and HTML to render dynamic content and manage user interactions, contributing to an intuitive and visually appealing experience.

Adel also implemented both the products listing page and the individual product detail page, allowing users to browse all available items and view specific information about each product.

**Image: Products page**

****

The image above shows a typical example of Products page.

**Image: Individual product page**

**A screenshot of a video game

AI-generated content may be incorrect.**

The image above shows a typical example of an individual product page.

|  |
| --- |
| **5.3 Team Member 3: Hugo Dos Reis** |

Hugo dos Reis was responsible for routing and integrating website components, using Express.js to manage API endpoints and connect the front-end with the back-end. He ensured features — from viewing products to placing orders — were accessible through properly structured routes. Hugo also maintained user session logic, keeping users connected throughout their experience.

Beyond routing, he managed the website’s security layer by implementing password hashing to securely store user credentials, a crucial step for safeguarding data and following best practices. Basic protections like input validation and session control were also applied to enhance the application’s integrity.

Hugo focused on core features, especially user authentication and account management.

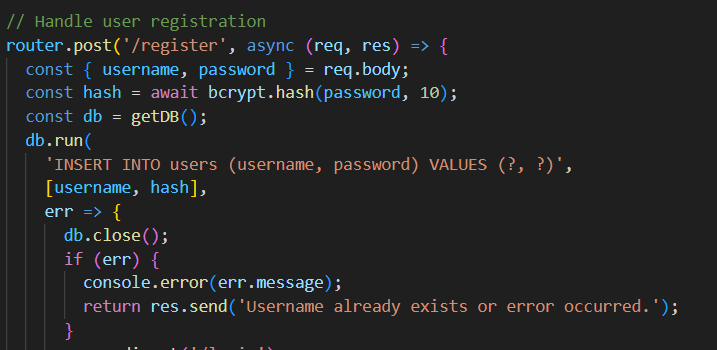
|  |
| --- |
|  |

**Password Security:**

We use **bcrypt** in Node.js to securely hash user passwords before storing them. Bcrypt adds salt and applies a strong hashing algorithm, protecting passwords from being stored in plain text.

The /register post will handle retrieving the password from the HTML form and then hashing it to add the hash to the database using bcrypt.

**Image: Password hashing Code**

****

In this code snippet, you can see how the password is hashed asynchronously using bcrypt’s hash method with a specified salt round value. Once the password is hashed, it is ready to be saved securely to the database.

**Image: Password hashed**

A screenshot of a computer

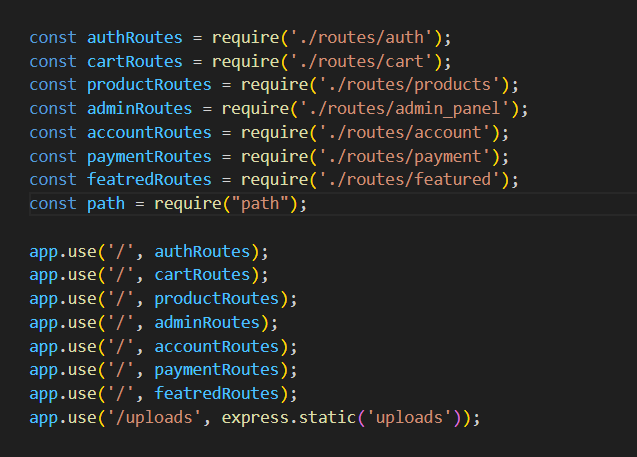
AI-generated content may be incorrect.

The final image illustrates an example of the hashed password stored in the database. Notice that it appears as a long, complex string that cannot be reversed to retrieve the original password, providing strong protection for user credentials.

**Routing:**

Routing is a fundamental part of the web application, as it defines how requests from users are handled and which responses are sent back. In our project, the main routing control is centralized in the app.js file. This file acts as the entry point for all HTTP requests, directing them to the appropriate handlers based on the requested URL and method.

**Image: Routing and Connectivity Code**

****

The image above shows a typical example of the routing setup in app.js. Here, Express.js is used to define routes that connect the front-end interface with the back-end logic. Each route corresponds to a specific feature, such as viewing products, managing user sessions, or processing orders.

While app.js manages the main routing setup, individual route handling is split across different files. Each module handles routes specific to its function, such as user authentication or product management. This modular structure keeps the code organized and simplifies development by isolating related routes.

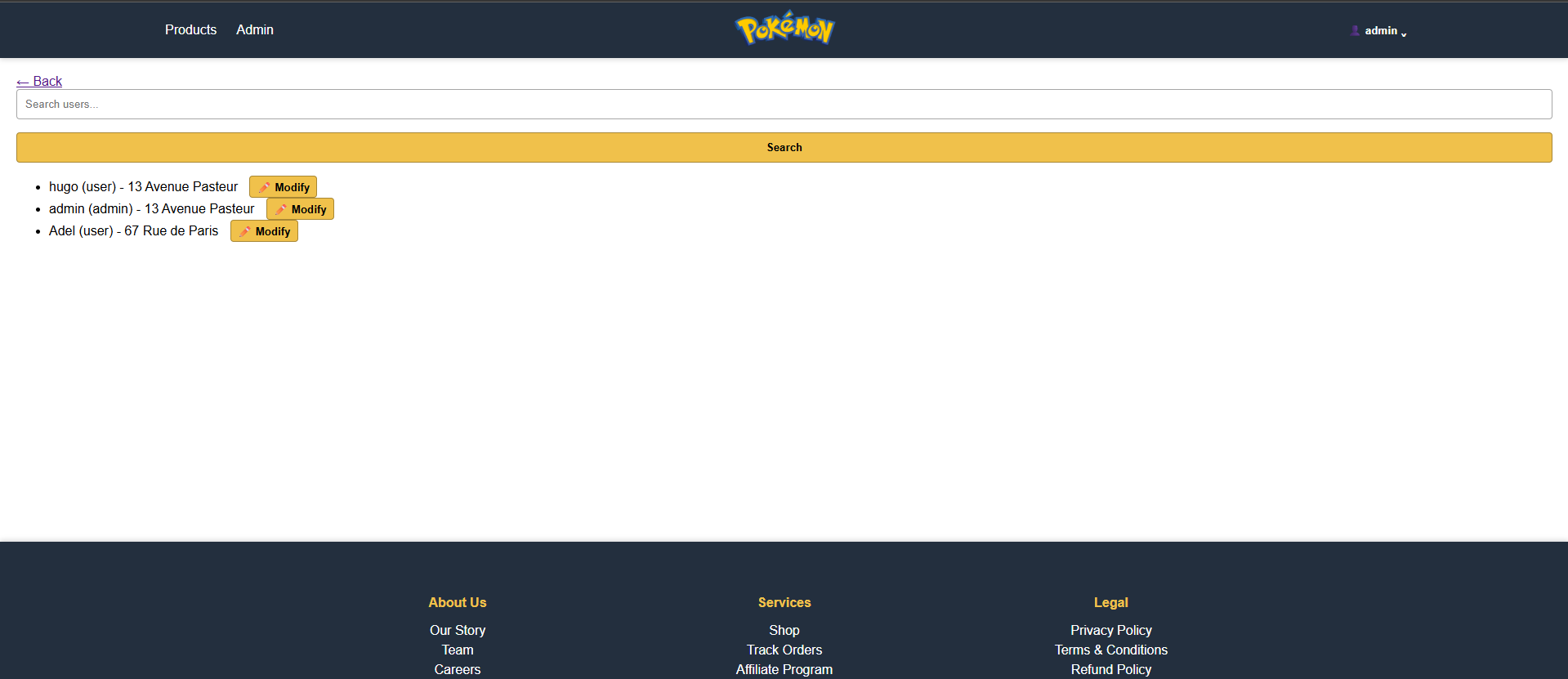
Admin panel:

Our admin panel provides comprehensive control over the website’s internal data and operations. Designed exclusively for administrators, it allows secure management of the three key entities of the system: **users**, **products**, and **orders**.

* **User Management**: Admins can view a full list of users, search by username, and access detailed profiles. Each user's information — including username, address, and role (user or admin) — can be edited directly. This is useful for correcting input errors, promoting users to admins, or maintaining account data.
* **Product Management**: Admins have the ability to search, create, update, and delete product listings. The interface allows updates to product name, price, category, stock, and image. A visual stock-to-volume indicator helps monitor product availability, enhancing inventory awareness. The library multer lets administrators upload images in a secure way for their products.
* **Order Management**: All orders are listed with associated users and timestamps. Admins can edit product quantities within orders, remove specific products, or delete entire orders. This level of control is useful for handling errors or fraudulent activities.

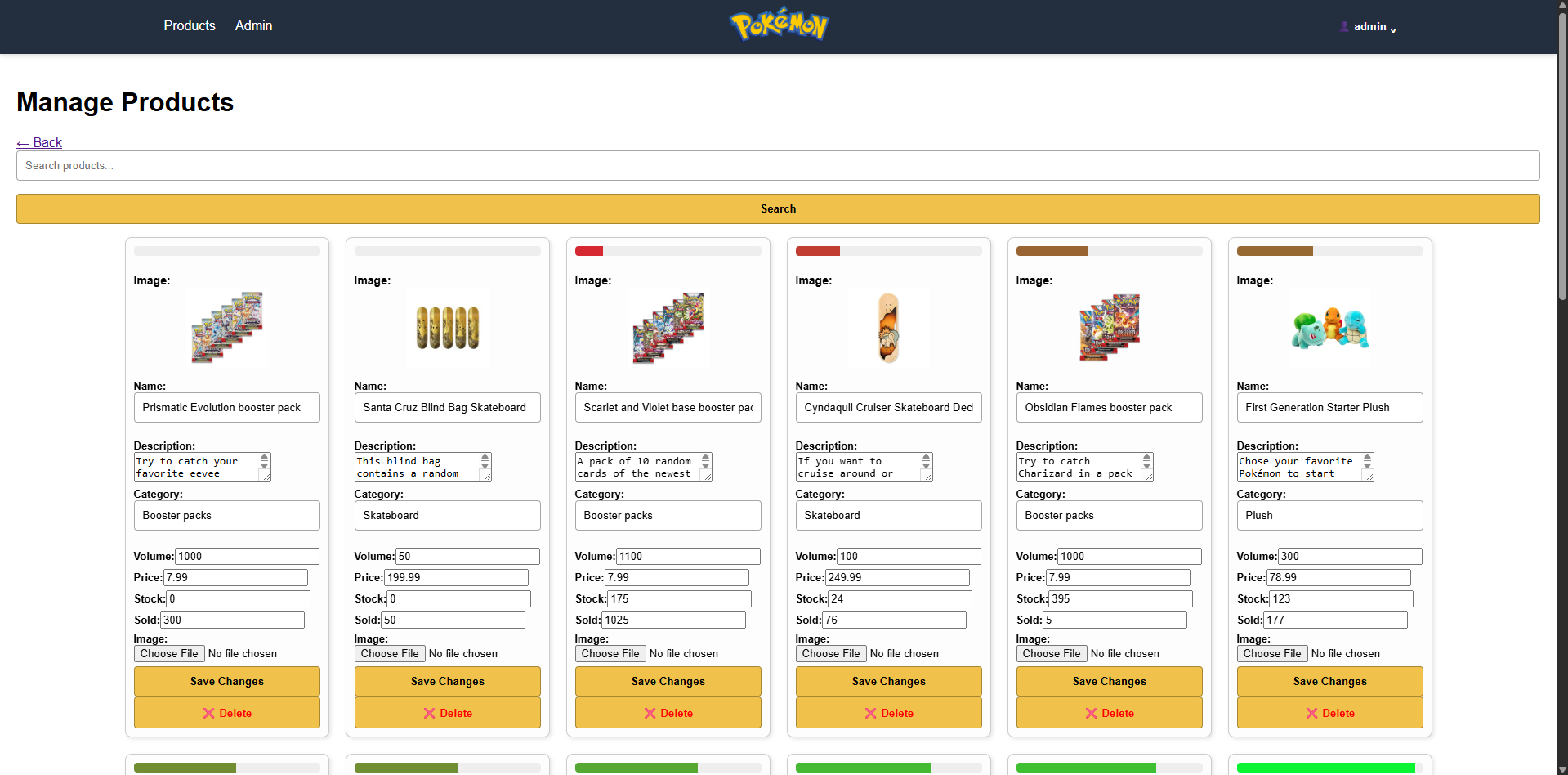
Overall, the admin panel acts as the **central hub** for maintaining data integrity, responding to customer issues, and managing the evolving content of the site. It's secured with session-based access control to prevent unauthorized use.

Image: User management



The image above shows a typical example of the user management interface. Admins can search for and select a user to view or modify their information.

Image: Products management and creation



The image above shows a typical example of Products management and creation. Admins can search for and modify products their data, admins can also create new products.

Image: Orders management

A screenshot of a computer

AI-generated content may be incorrect.

The image above shows a typical example of Orders management. Admins can search for, modify and delete orders.

|  |
| --- |
| **6. Conclusion and Future Perspectives** |

This e-commerce project was a good opportunity to practice full-stack web development using tools like Node.js, Express, JavaScript, SQLite, HTML, and CSS. The website includes important features such as a product catalog, shopping cart, user login, and a basic checkout process.

We built a simple and functional database, and the code works as intended, even if it could be improved for readability and organization. The design is basic but usable, and some adjustments may be needed for better responsiveness on all screen sizes.

Main features like product categories, shopping cart updates, administration panel, and user accounts were implemented successfully. A test payment system was used, but the code is ready to connect to a real one in the future.

This project can be improved by adding stronger security, a better design, and more advanced features if needed.

Future improvements may include:

* Migrating to a more robust database solution for better scalability.
* Adding real-time features such as order tracking or live chat support.
* A real payment process and interface instead of a mock one

Overall, this project serves as a strong foundation for building a professional e-commerce platform and demonstrates the practical application of modern web development practices.