WEB Technologies Final Report

Coffee PO (Coffee Menu & Ordering Backend)

Course: WEB Technologies (Backend)

Group: SE-2436

Students: Nurzhan Izimbetov, Saqyp Dias, Ali Qazybai

Date: 2026-02-09

GitHub: https://github.com/aaituu/WEb_backend_Finall

Link: https://cute-olive-5e9.notion.site/WEB-Technologies-Final-Report-302680ce2e278012a04bdc812417ac00?source=copy_link

---

1. Project Overview

Topic: Coffee shop menu and ordering system (backend + MongoDB).

Goal: Provide a secure REST API for user authentication, product management, cart operations, orders, feedback, and analytics.

Key Objectives

- Build a modular Node.js + Express backend.
- Use MongoDB with Mongoose for flexible data storage.
- Implement authentication with JWT and password hashing.
- Provide validated, well-structured REST endpoints.
- Add admin-only access for sensitive operations.
- Integrate SMTP for contact messages.

---

2. Tech Stack

- **Backend:** Node.js, Express

- **Database:** MongoDB + Mongoose
- **Auth:** JWT, bcryptjs
- **Validation:** Joi
- **Email:** Nodemailer (SMTP)
- **Documentation:** Static OpenAPI JSON at `/api/docs.json`

---

3. System Architecture

Presentation Layer: Static HTML/JS frontend in `public/` (served by Express).

Logic Layer: Express controllers and middleware in `src/controllers/` and `src/middleware/`.

Data Layer: MongoDB models in `src/models/`.

Data Flow (Simplified)

1. Client sends REST request.
2. Middleware validates JWT (private routes).
3. Controller validates input with Joi.
4. Mongoose reads/writes MongoDB.
5. Response returned to client.

---

4. Project Structure (Modular)

- `src/routes/` for endpoints
- `src/controllers/` for business logic
- `src/models/` for MongoDB schemas
- `src/middleware/` for auth and errors
- `src/config/` for configuration (DB connection)

---

5. Database Schema Description

5.1 Functional Requirements & User Interaction

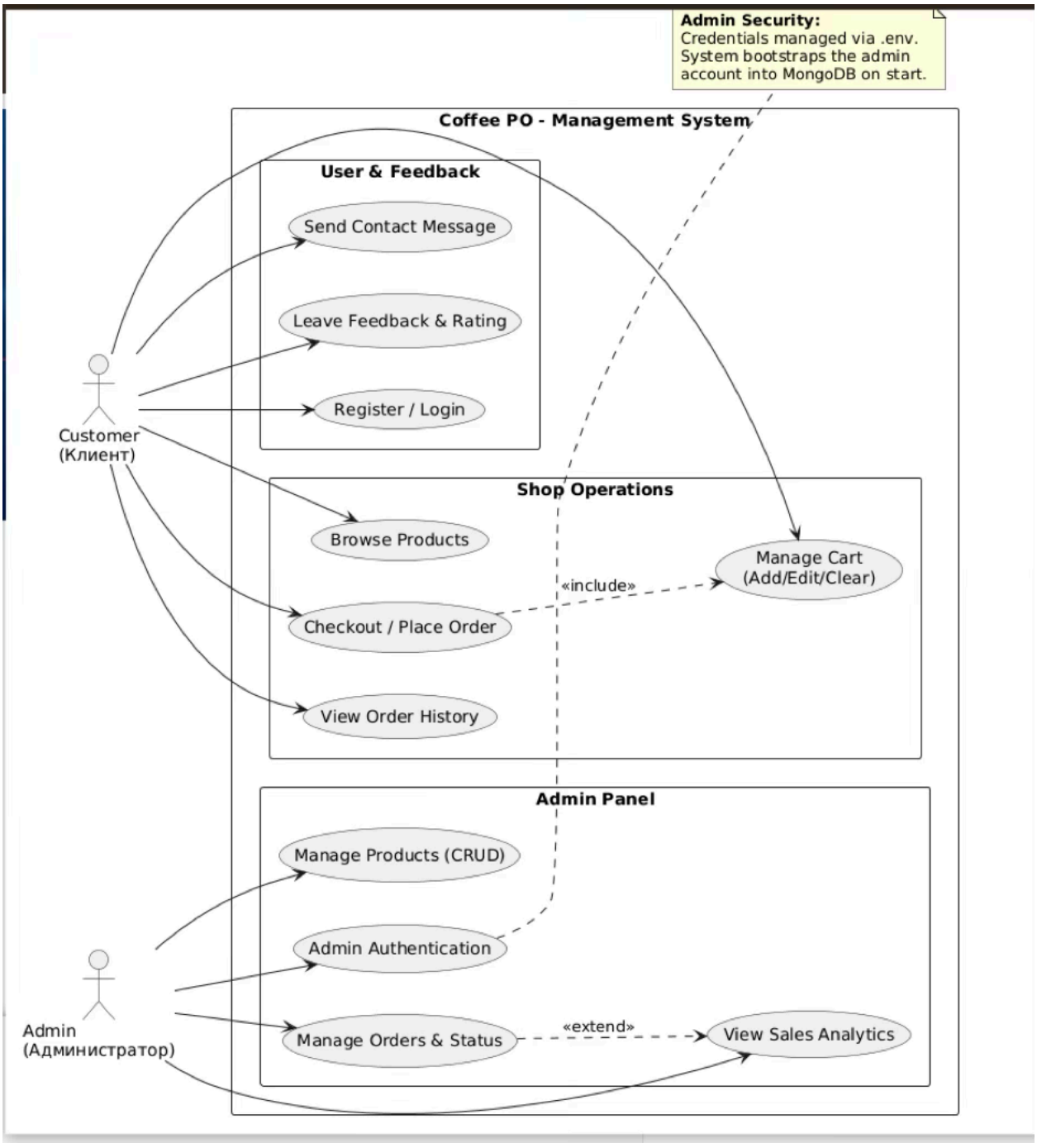
Before defining the technical schema, it is essential to understand the functional requirements of the *Coffee PO* system. As illustrated in the Use Case Diagram, the system supports two primary actors:

Customer

- Browse available products and menu items.
- Manage a dynamic shopping cart (add, update quantity, remove, clear).
- Place and finalize orders with additional details such as `tableNumber`.
- View order history and provide feedback.

Admin

- Manage the product catalog (Create, Read, Update, Delete).
- Monitor analytics and sales reports.
- Update order statuses (e.g., pending → completed → cancelled).

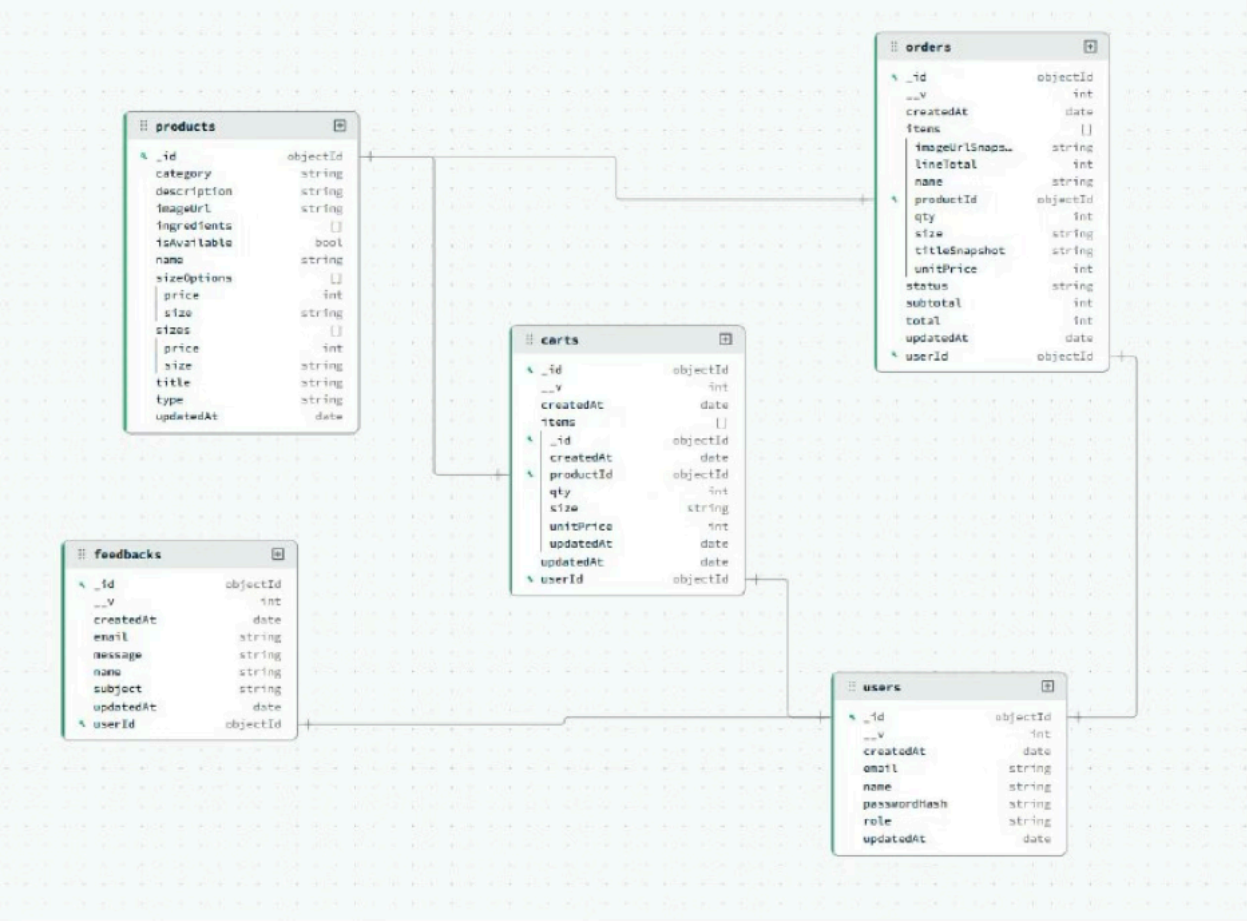


Security Design Note:

Admin credentials are managed via environment variables (`.env`) rather than the public user database. This architectural decision improves security by preventing exposure of admin accounts through standard user collections.

5.2 Data Relationship Model (ERD)

The ER Diagram provides a high-level view of how data entities relate to one another within the NoSQL environment. The system adopts a **Hybrid Relationship Model**, combining embedded documents and references to achieve both performance and flexibility.



Feedback Model (Customer Engagement)

The feedback entity is designed to capture user experience and support service improvement.

Design Considerations:

- The `user` reference links feedback to a specific customer for follow-up.
- The `category` field enables classification of feedback (e.g., service, product quality, delivery speed).

- Timestamp fields support historical and trend analysis

  (e.g., "feedback quality improved after menu update").

---

6. API Documentation

This section outlines the **25+ RESTful endpoints** implemented in the *Coffee PO* system, significantly exceeding the minimum requirement of 12 endpoints for a two-student project.

6.1 Authentication & User Endpoints

- **POST** `/api/auth/register`

  Creates a new user document with hashed passwords.

- **POST** `/api/auth/login`

  Authenticates users and returns a JWT for session management.

- **GET** `/api/users/me`

  Retrieves the current authenticated user's profile data (used for the Profile page).

- **PUT** `/api/users/me`

  Updates user profile information.

6.2 Products Management (Full CRUD)

- **GET** `/api/products`

  Retrieves all menu items from the Products collection.

- **GET** `/api/products/:id`

  Retrieves technical details for a single product.

- **POST** `/api/products` *(Admin Only)*

  Adds a new beverage or dessert to the menu.

- **PUT** `/api/products/:id` *(Admin Only)*

  Updates existing product details (price, availability).

- **DELETE** `/api/products/:id` *(Admin Only)*

  Removes a product from the database.

6.3 Shopping Cart Logic (Advanced Updates)

- **GET** `/api/cart`

  Fetches the user's active cart.

- **POST** `/api/cart/items`

  Adds an item to the cart using the `$push` operator.

- **PATCH** `/api/cart/items/:id`

  Updates quantity using the `$set` operator.

- **DELETE** `/api/cart/items/:id`

  Removes a specific item from the array using the `$pull` operator.

- **DELETE** `/api/cart/clear`

  Clears the entire items array.

6.4 Orders & Business Intelligence

- **POST** `/api/orders`

  Finalizes checkout by moving items from Cart to Order with a price snapshot.

- **GET** `/api/orders`

  Retrieves order history for the authenticated user.

Analytics Endpoints (Advanced Aggregation)

The system implements dedicated analytics routes using MongoDB's Aggregation Framework:

- **GET** `/api/analytics/top-products`

  Returns best-selling products by total quantity sold using `$unwind`, `$group`, and `$sort`.

- **GET** `/api/analytics/sales-by-status`

Provides revenue breakdown by order status (completed, pending, cancelled).

Implementation Note:

These endpoints reside in a separate `analytics.routes.js` file to maintain modularity and support future expansion.

6.5 Admin Management (Separation of Concerns)

- **POST** `/api/admin/login`

  Authenticates admin users and returns an admin-specific JWT.

- **GET** `/api/orders/admin/all` *(Admin Only)*

  Retrieves all orders in the system.

- **PATCH** `/api/orders/admin/:id/status` *(Admin Only)*

  Updates the status of any order.

Design Rationale:

Separating admin authentication and management routes ensures cleaner architecture and reduces the risk of privilege escalation.

6.6 User Engagement Features

To enhance customer interaction and collect feedback:

Contact Form

- **POST** `/api/contact/send`

  Submits inquiries or feedback (stored in MongoDB or forwarded via email using Nodemailer).

Feedback System

- **GET** `/api/feedback/my`

  Retrieves all feedback created by the authenticated user.

- **POST** `/api/feedback`

  Creates a new feedback document with rating and comments.

- **DELETE** `/api/feedback/my`

  Deletes all feedback entries for the current user.

Business Value:

These endpoints enable customer satisfaction analysis, improve retention, and support continuous service improvement.

6.7 API Documentation Interface

To simplify testing and exploration of the RESTful API, the system provides an interactive documentation interface powered by Swagger.

Docs

- **GET** `/api/docs.json`

  Returns the static OpenAPI (Swagger) specification in JSON format.

- **GET** `/api/docs`

  Opens the Swagger UI web interface for interactive API exploration and testing.

Benefit:

These endpoints allow developers and testers to easily understand available routes, request formats, response schemas, and authentication requirements without inspecting source code.

---

7. Authentication & Security

- JWT-based auth for private endpoints.
- Passwords hashed using bcryptjs.
- RBAC with `adminOnly` middleware (admin vs user).
- Environment variables for secrets and SMTP settings.

---

8. Validation & Error Handling

- Joi schemas validate incoming request bodies.

- Global error handler returns consistent JSON errors.
- Status codes used for validation, auth, and not-found cases.

9. SMTP Integration

Contact form uses Nodemailer with SMTP settings in `.env`.

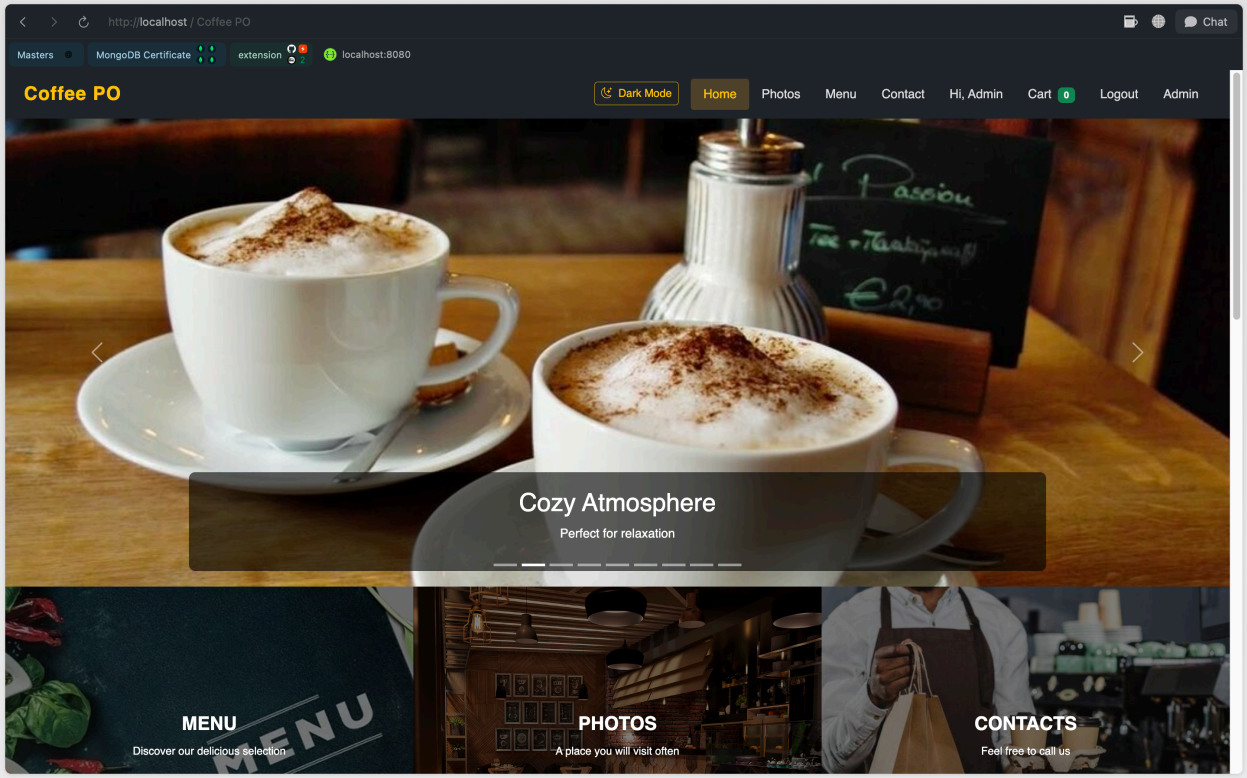
This enables feedback messages to be sent via email.

10. Deployment

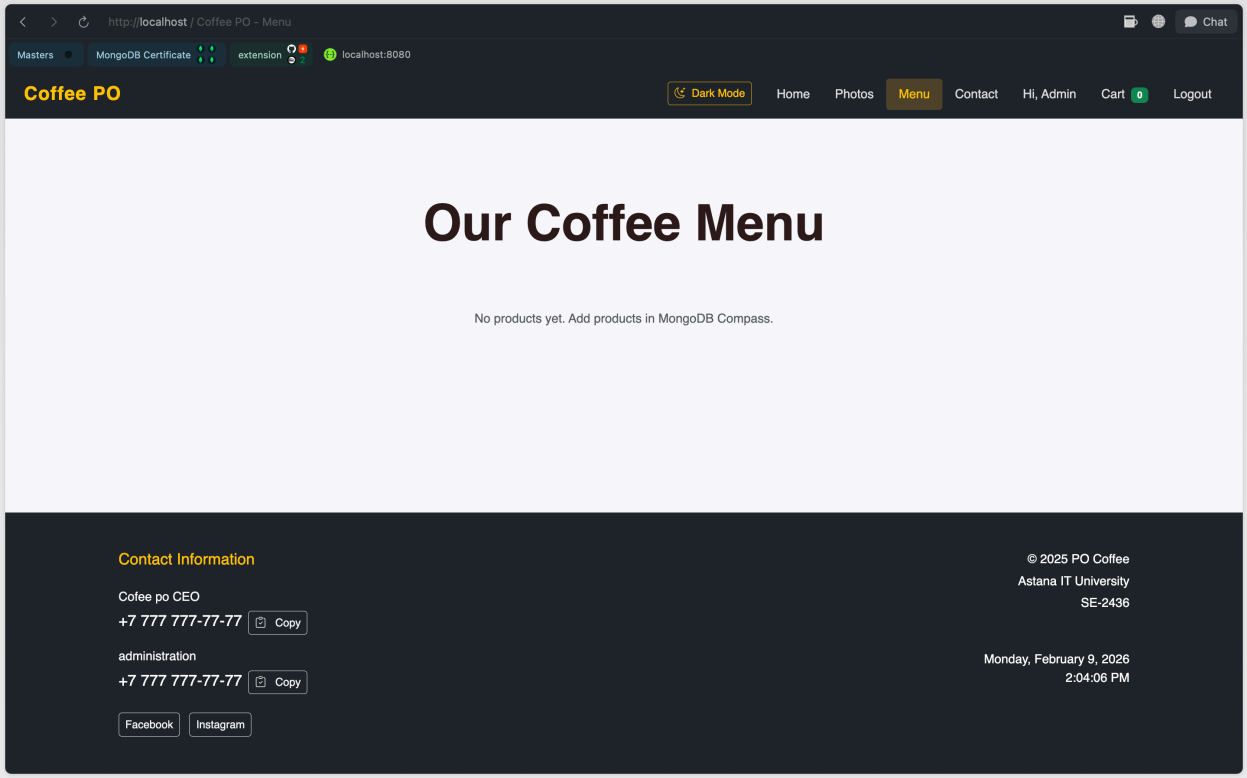
- Environment variables are required for DB and JWT configuration.
- **Deployed URL:** https://coffee-app-kjgq5.ondigitalocean.app/
- **API Docs:** https://coffee-app-kjgq5.ondigitalocean.app/api/docs

11. Screenshots

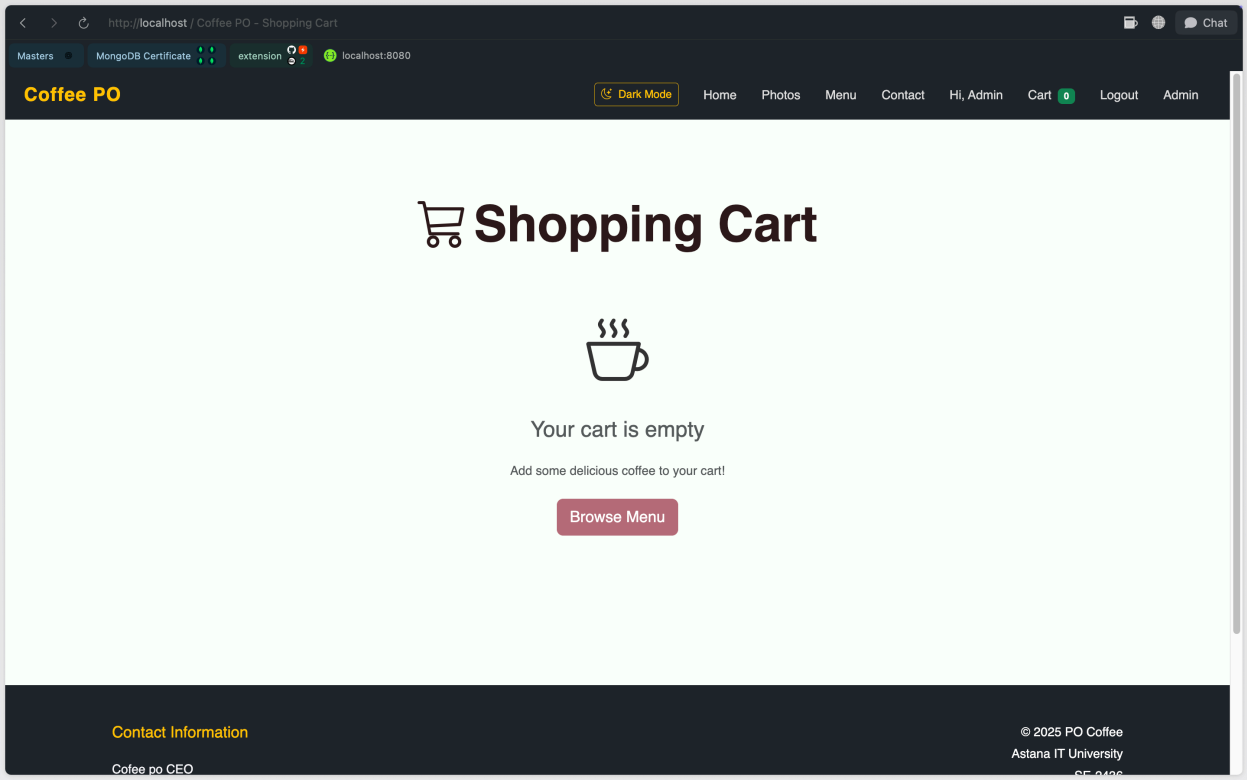
- Home page (`public/index.html`)



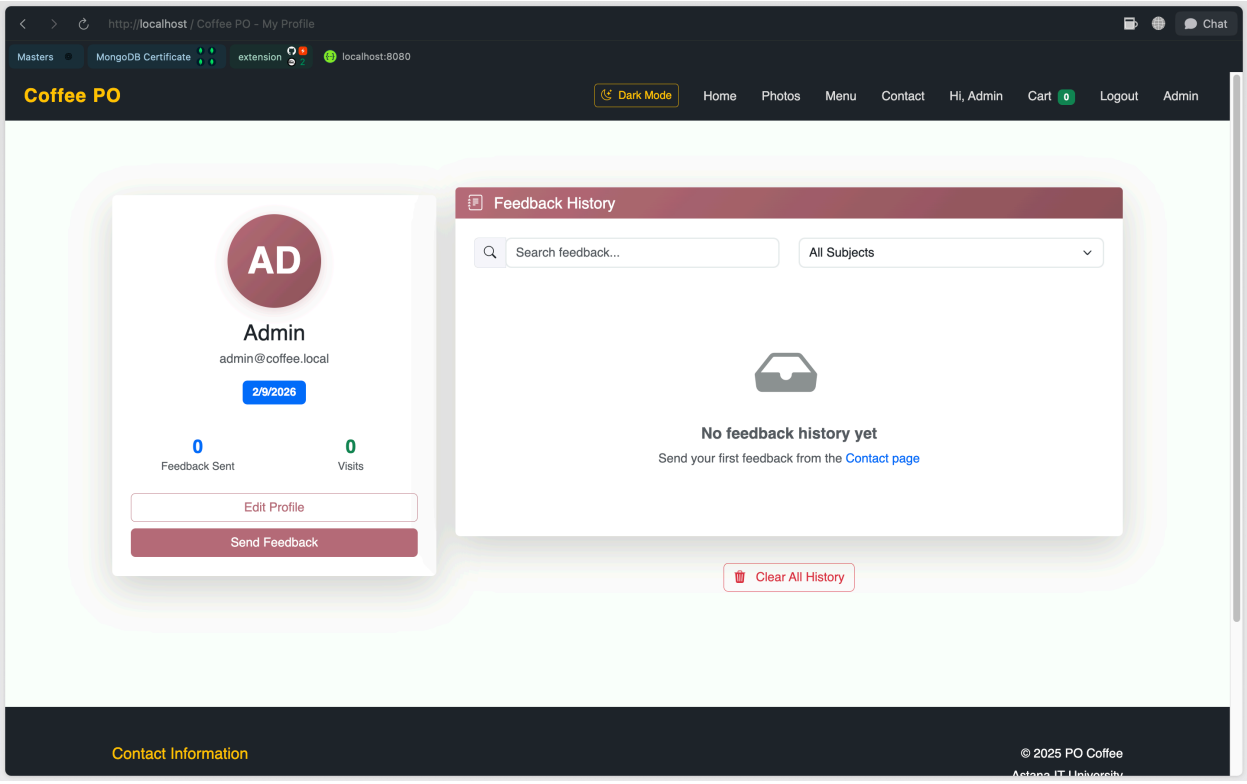
- Menu page (`public/menu.html`)



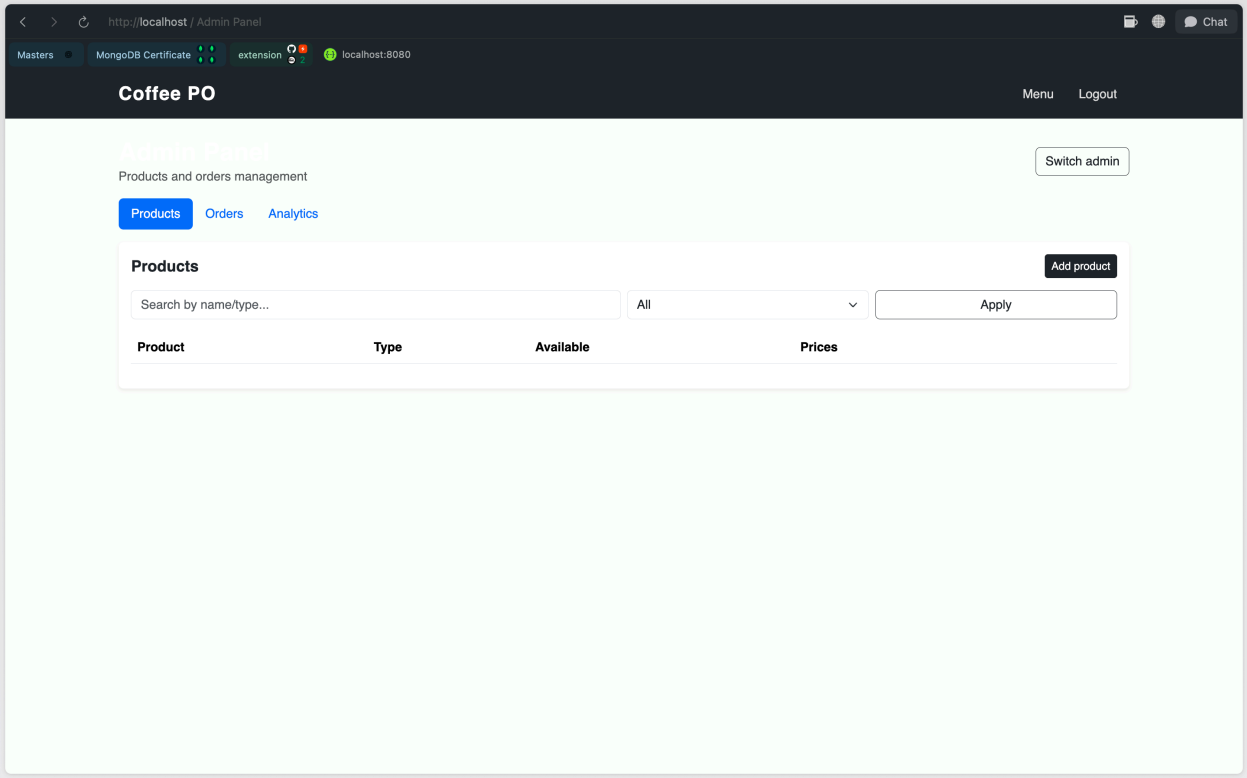
- Cart page (`public/cart.html`)



- Profile page (`public/profile.html`)



- Admin panel (`public/admin.html`)



12. Contribution Summary

Dias Saqyp

- User authentication (register, login, JWT)
- Password hashing and security
- User profile endpoints
- Middleware (auth, error handling)
- Server setup and MongoDB connection

Ali Qazybai

- Product and catalog logic
- Product and category schemas
- CRUD endpoints for products

- Admin-only product management
- Database seeding with coffee products

Nurzhan Izimbetov

- Cart functionality
- Order creation and order history
- Cart and order schemas
- Cart-to-order logic
- API testing and endpoint documentation

---

13. Conclusion

The Coffee PO backend fulfills the requirements for a modular Node.js + Express project using MongoDB, JWT authentication, RBAC, validation, and SMTP integration. The system provides a production-style REST API suitable for a coffee shop ordering workflow and demonstrates strong backend fundamentals for the WEB Technologies (Backend) course.