Backend System Design — E-commerce APIs

1. Introduction

This document describes the backend system design of an e-commerce API service built with **Node.js + Express** and **MongoDB**. It provides RESTful endpoints for managing users, products, categories, carts, and orders. The backend is **API-only** (no UI) and intended for portfolio demonstration.

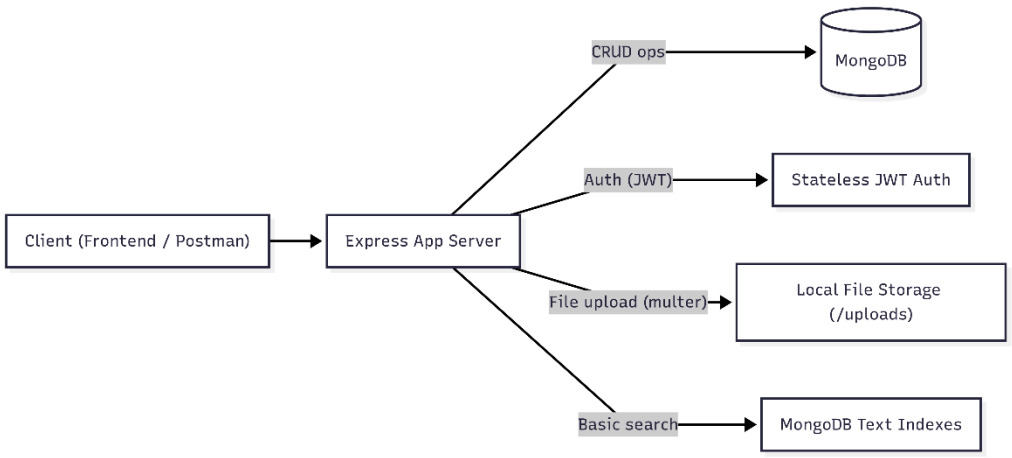
---

2. Current Implementation

2.1 Tech Stack

- **Runtime:** Node.js
- **Framework:** Express.js
- **Database**: MongoDB (Atlas)
- **Authentication:** JWT (Bearer tokens)
- **File Uploads:** Multer (stored locally in /public for now)
- **Logging:** Morgan middleware
- **CORS:** Enabled for frontend (`http://localhost:5000`)
- **Custom Middleware**:
  - `authJwt` → JWT verification
  - `LoggingMW` → request logging
  - `ErrorMW` → centralized error handler

2.2 Architecture (Implemented)



2.3 Future Enhancements

- **Redis** → caching, session store, rate limiting
- **RabbitMQ/Kafka** → async background jobs (emails, inventory updates)
- **S3-compatible storage** → scalable image/file storage
- **Meilisearch/Elasticsearch** → advanced search

---

3. API Endpoints

3.1 Cart (Users)

- `POST /cart/add` → add item to cart
- `POST /cart/remove` → remove item from cart
- `PUT /cart/update` → update item quantity
- `GET /cart/` → get current user's cart
- `DELETE /cart/clear` → clear cart

3.2 Users

- `PUT /admin/:id` → set user as admin (admin)
- `PUT /user/:id` → update user (admin or profile's owner)
- `POST /user/register` → user registration
- `POST /user/login` → user login
- `POST /user/logout` → user logout
- `GET /user/` → list all users (admin)
- `GET /user/:id` → get user by ID (admin)
- `DELETE /user/:id` → delete user (admin)

3.3 Categories

- `POST /categories/` → add category (admin)
- `GET /categories/:id` → get category by ID
- `GET /categories/` → list all categories
- `PUT /categories/:id` → update category (admin)
- `DELETE /categories/:id` → delete category (admin)

3.4 Orders

- `POST /orders/` → place order (user)
- `GET /orders/` → list all orders (admin)
- `GET /orders/user/:id` → get orders by user (admin or profile's owner)
- `GET /orders/:id` → get order by ID (admin or profile's owner)

- `PATCH /orders/:id` → update order's status (admin)
- `DELETE /orders/:id` → delete order (admin or profile's owner)

3.5 Products

- `POST /products/` → create product (admin, with image upload)
- `GET /products/get/count` → count products
- `GET /products/` → list all products
- `GET /products/:id` → get product by ID
- `PUT /products/images/:id` → update product images (admin)
- `PUT /products/:id` → update product (admin)
- `DELETE /products/:id` → delete product (admin)

---

4. API Specification (OpenAPI 3.0)

• A full machine-readable spec is provided in openapi.yaml at the root of the repo.

• This spec describes all endpoints, request/response schemas, and auth requirements.

• You can:

- Import it into Postman (File → Import → openapi.yaml) to auto-generate a collection.
- Load it in Swagger UI for an interactive API explorer.

---

5. Data Model

- **User**: { id, name, email, password, phone, isAdmin, street, apartment, city, country, createdAt }
- **Product**: { id, name, description, price, categoryId, image(s), stock, rating, dateCreated }
- **Category**: { id, name, icon }
- **Order**: { id, userId, items, shippingAddress1, shippingAddress2, phone, totalPrice, status, dateOrdered }
- **Cart**: { id, userId, items: [{ productId, quantity }] }

---

6. Security

- **Authentication**: JWT-based, attached in `Authorization` header
- **Authorization**: middleware checks (`checkAdmin`, `checkAuth`, `checkNonAdminUser`)

- **Validation**: middleware for request body (e.g., `validateAddToCart`, `validator` for users and products)

---

7. Limitations

- **File Storage**: Images stored locally (`/uploads`); no CDN/S3 integration.
- **Search**: Limited to MongoDB `$text` queries (basic relevance only)
- **Scalability**: Single Express instance; no load balancing, Redis cache, or queues
- **Transactions**: MongoDB transactions not yet implemented for multi-step operations
- **Testing**: No automated test suite (manual Postman testing only)

---

8. Deployment Notes

- Environment variables handled via `.env` (`DB_URI`, `API_URL`, `PORT`)
- Static assets served from `/public`
- Dev logging via `morgan('dev')`
- Example dev startup:

```
NODE_ENV=development PORT=3000 API_URL=/api/v1

DB_URI=mongodb+srv://<username>:<password>@cluster0.xmxu3yc.mongodb.net/onlineShopDB?retryWrites=true&w=majority&appName=Cluster0
```

---

9. Portfolio Value

- Shows **full-stack API design** (auth, CRUD, validation, middleware)
- Demonstrates **scalable patterns** (JWT stateless auth, separation of concerns)
- Highlights **awareness of missing production features** (queues, caching, cloud storage)
- Clear system design document + OpenAPI + architecture diagram → recruiter-ready