🛍️Mini E-commerce API (Monolithic Architecture)

A full-featured **e-commerce backend** built with **Spring Boot**, designed for **Intern/Fresher portfolio projects**. This project integrates modern backend features — from authentication, caching, messaging, and real-time updates to cloud deployment and containerization.

🗜️Features Overview

🔐Authentication & Security

- JWT Authentication (Access + Refresh Token)
- OTP verification for registration and forgot password
- Login via Google & GitHub (OAuth2)
- Role-based Authorization (Spring Security + @PreAuthorize)
- Password reset via email (MailSender)

🧰Core Business Modules

- CRUD operations with DTO + MapStruct mapper
- RESTful API design & versioning
- Product management, order, cart, category, and payment modules
- Scheduler for automated tasks (fixed-rate / cron)
- Event listener (Spring ApplicationEventPublisher)
- WebSocket real-time notifications (order status, messages)
- Payment integration (VNPAY sandbox)

⚙️System & Infrastructure

- PostgreSQL with Flyway for migrations
- Redis caching (TTL + session management)
- RabbitMQ for async message queue (event-driven communication)
- Cloudinary integration for image uploads
- Dockerfile + docker-compose.yml (multi-container setup)
- Swagger API documentation (OpenAPI 3)
- Global Exception Handler + Validation
- Audit logging with SLF4J
- I18N (English & Vietnamese)
- Healthcheck & Actuator endpoints

🧱Architecture Overview

Type: Monolithic (modular structured)

Main Packages:

```
com.chubecommerce
 ├ auth/          → login, register, JWT, OTP
 ├ user/          → CRUD users, roles
 ├ product/       → CRUD product, category, image upload
 ├ order/         → cart, checkout, payment (VNPAY)
 ├ common/        → constants, exception, global handler
 ├ config/        → security, swagger, redis, etc.
 ├ scheduler/     → background tasks
 ├ event/         → listener & publisher
 ├ websocket/     → realtime notification
 └ util/          → helper utils
```

🧮Database Design (PostgreSQL)

Main tables: - **users**, **roles**, **refresh_tokens** - **products**, **categories**, **product_images** - **orders**, **order_items**, **payments** - **otp_verifications**, **audit_logs**

Each entity extends `BaseEntity` (id, created_at, updated_at, created_by, updated_by).

⚙️Tech Stack Breakdown

Technology	Purpose
Spring Boot 3.x	Core framework for building RESTful backend
Spring Security + JWT	Authentication and authorization management
Spring Data JPA (Hibernate)	ORM for database CRUD operations
PostgreSQL	Main relational database
Flyway	Database version control and migrations
Redis	Cache storage and temporary OTP/session management
RabbitMQ	Message broker for async event handling (payment success, email notifications...)
Cloudinary	Cloud storage for product images
Spring Mail (JavaMailSender)	Send emails for OTP, reset password, confirmation

Technology	Purpose
Swagger / OpenAPI 3	Interactive API documentation
WebSocket (STOMP)	Real-time notifications for orders and system messages
MapStruct	Object mapping (DTO ↔ Entity) for clean architecture
Lombok	Boilerplate code reduction (getters, setters, builders)
Docker & Docker Compose	Containerized local setup
SLF4J + Logback	Centralized logging and error tracing
Spring Scheduler	Background jobs (e.g. auto-cancel unpaid orders)
Spring Event Listener	Decoupled async logic (email sending, logging)
Actuator	Health check and monitoring endpoints
I18N ResourceBundle	Multilingual support (EN / VN)
VNPAY Integration	Online payment gateway

🕐 Use Case Flow Descriptions

User Registration + OTP Verification

1. User submits registration form → `POST /api/auth/register`
2. Backend creates temporary user record + generates OTP → sends email via `MailSender`
3. User enters OTP → `POST /api/auth/verify-otp`
4. Backend verifies OTP (stored in Redis or table `otp_verifications`)
5. Account activated → JWT access & refresh tokens returned.

🔑 Login + JWT Token Flow

1. User login with credentials → `POST /api/auth/login`
2. System validates user → issues `accessToken` + `refreshToken`
3. Access token used for all secured endpoints.
4. When expired → `POST /api/auth/refresh` to renew using refresh token.

🔐 Forgot Password + Reset

1. User submits email → system generates OTP → sends via MailSender.
2. User submits OTP + new password → verify OTP → update user password.

🛍️ Product Management

1. Admin CRUD products via `ProductController`.
2. Upload product images to Cloudinary.

3. Cache popular products using Redis (TTL 5–10 minutes).
4. List APIs paginated + searchable (criteria query).

🛒 Cart & Checkout Flow

1. User adds product to cart → saved in DB or Redis session.
2. Checkout → create Order & OrderItem records.
3. System triggers payment initiation (VNPAY URL generation).

💳 Payment (VNPAY)

1. User redirected to VNPAY sandbox URL.
2. After payment → callback to `/api/payments/vnpay/return`.
3. Verify signature + update Order status = `PAID`.
4. Publish `PaymentSuccessEvent` to RabbitMQ.
5. Consumer listens → send confirmation email & WebSocket notification.

🔔 Real-time Notification (WebSocket)

- Order status changes or admin broadcasts → pushed via STOMP.
- Example: "Your order #123 has been shipped."

🕑 Scheduler Jobs

- Auto-cancel unpaid orders after 30 minutes.
- Clear expired OTPs daily.
- Trigger daily report events to RabbitMQ.

📨 Event Listener Use Cases

- `PaymentSuccessEventListener` → send mail, update stats.
- `UserRegisteredEventListener` → log audit + send welcome mail.

Caching Strategy

- Redis for short-lived data (OTP, token blacklist, product cache).
- TTL ensures auto-expiration, improving performance.

📦 Deployment Flow

1. Dockerize app + services (PostgreSQL, Redis, RabbitMQ).
2. Deploy on **Render / Railway / AWS EC2**.
3. Use environment variables for secret configs.

🧮Local Development Setup

✂️Prerequisites

Make sure you have installed: - **Java 17+** - **Maven 3.9+** - **Docker & Docker Compose**

✒️Clone & Build

```
git clone https://github.com/<your-username>/mini-ecommerce-api.git
cd mini-ecommerce-api
mvn clean install
```

🛏️Configure Environment Variables

Create a file `.env` in the project root:

```
# Database
POSTGRES_DB=ecommerce_db
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres

# Redis
REDIS_HOST=redis
REDIS_PORT=6379

# RabbitMQ
RABBITMQ_HOST=rabbitmq
RABBITMQ_PORT=5672
RABBITMQ_USER=guest
RABBITMQ_PASSWORD=guest

# Cloudinary
CLOUDINARY_NAME=your_cloud_name
CLOUDINARY_API_KEY=your_api_key
CLOUDINARY_API_SECRET=your_api_secret

# JWT & OTP
JWT_SECRET=your_jwt_secret_key
OTP_EXPIRATION_MIN=5

# MailSender
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=your_email@gmail.com
MAIL_PASSWORD=your_app_password
```

```
# VNPAY
VNPAY_TMN_CODE=your_tmn_code
VNPAY_SECRET_KEY=your_secret_key
VNPAY_PAY_URL=https://sandbox.vnpayment.vn/paymentv2/vpcpay.html
VNPAY_RETURN_URL=http://localhost:8080/api/payments/vnpay/return
```

🌂Start Services

```
docker-compose up -d
```

This will start: - PostgreSQL - Redis - RabbitMQ

🌡️ Run Application

```
mvn spring-boot:run -Dspring.profiles.active=dev
```

Or via Docker Compose:

```
docker-compose up --build
```

🎒Access Swagger API Docs

👉http://localhost:8080/swagger-ui/index.html

📈Roadmap (Planned Enhancements)

- 👈Integration test with Testcontainers
- 👈CI/CD pipeline (GitHub Actions)
- 👈AWS S3 for file storage (alternative to Cloudinary)
- 👈Deployment with Render / Railway / AWS EC2

💻Author

Tran Quang Huy
FPT University | Backend Developer (Spring Boot)
📀chube.dev@gmail.com
👈LinkedIn | GitHub

😅*“Code clean, think in modules, deploy like a pro.”*