Product Service Development Guide

Development Environment Setup

This guide provides instructions for setting up your development environment for the Product Service microservice. Following these steps will help you establish a consistent development workflow that matches the twelve-factor methodology.

Prerequisites

Ensure you have the following tools installed:

- **JDK 17+** - Required for Java development
- **Maven 3.8+** - For dependency management and building
- **Docker and Docker Compose** - For containerized development
- **Git** - For version control
- **IDE** - IntelliJ IDEA (recommended), Eclipse, or VS Code with Java extensions
- **Postman/Insomnia** - For API testing

Local Development Setup

1. Clone the Repository

```
git clone https://github.com/your-org/ecommerce-microservices.git
cd ecommerce-microservices/product-service
```

2. Configure IDE

For IntelliJ IDEA:

1. Open the project: File → Open → Select the `product-service` directory
2. Import Maven project when prompted
3. Set Java 17 as the SDK: File → Project Structure → Project → SDK
4. Install Lombok plugin if not already installed

3. Build the Project

```
mvn clean install
```

4. Development Modes

Mode 1: Standalone Development with H2 Database

This mode doesn't require external dependencies and uses an in-memory H2 database.

1. Run the application with the default profile:

   ```
   mvn spring-boot:run
   ```

2. The application will start on port 8080 with the context path `/api`
   - Access the API at: http://localhost:8080/api
   - Access H2 console at: http://localhost:8080/api/h2-console (JDBC URL: `jdbc:h2:mem:productdb`)

Mode 2: Development with PostgreSQL

This mode uses a PostgreSQL database for persistence.

1. Start PostgreSQL:

   ```
   docker run --name postgres -e POSTGRES_USER=postgres -e
   POSTGRES_PASSWORD=postgres -e POSTGRES_DB=product_db -p 5432:5432 -d
   postgres:13
   ```

2. Run the application with the postgres profile:

   ```
   mvn spring-boot:run -Dspring-boot.run.profiles=postgres
   ```

Mode 3: Development with Full Stack

This mode starts the entire stack including PostgreSQL, Kafka, and Zookeeper.

1. Start all services using Docker Compose:

   ```
   docker-compose up -d db kafka zookeeper kafka-ui
   ```

2. Run the application with the postgres profile:

   ```
   mvn spring-boot:run -Dspring-boot.run.profiles=postgres
   ```

5. Development Database Management

Database Migrations

The project uses Flyway for database migrations:

1. Migration scripts are located in `src/main/resources/db/migration`
2. Following naming convention: `V{version}__{description}.sql`

To create a new migration:

1. Create a new SQL file in the migrations directory
2. Follow the naming convention (e.g., `V2__add_product_category.sql`)
3. Write your SQL statements

Generate Test Data

To load test data for development:

```
mvn spring-boot:run -Dspring-boot.run.profiles=postgres,dev-data
```

Code Quality Standards

Code Style

The project follows Google Java Style Guide. To check and apply formatting:

```
# Check style
mvn checkstyle:check

# Format code (if you have the formatter plugin)
mvn git-code-format:format-code
```

Testing Strategy

1. **Unit Tests**: Target individual classes with mocked dependencies
   - Located in `src/test/java`
   - Naming convention: `*Test.java`
   - Run with: `mvn test`
2. **Integration Tests**: Test component interactions
   - Located in `src/test/java`
   - Naming convention: `*IT.java`
   - Run with: `mvn verify`
3. **API Tests**: Test REST endpoints
   - Located in `src/test/java`
   - Naming convention: `*ApiTest.java`
   - Run with: `mvn verify`

Debugging

Remote Debugging

To enable remote debugging:

```
mvn spring-boot:run -Dspring-boot.run.jvmArguments="-Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005"
```

Then connect your IDE to port 5005.

Debugging in Docker

To debug the application running in Docker:

1. Update the docker-compose.yml file:

```
product-service:
  environment:
    - JAVA_TOOL_OPTIONS=-
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005
  ports:
    - "8080:8080"
    - "5005:5005"
```

2. Start the containers:

```
docker-compose up
```

3. Connect your IDE to port 5005.

Kafka Development

Viewing Kafka Topics and Messages

1. Start the Kafka UI:

```
docker-compose up -d kafka-ui
```

2. Access the Kafka UI at http://localhost:8090

Creating and Testing Kafka Events

Use the provided test utilities to publish sample events:

```
// Example from src/test/java/.../util/KafkaTestUtil.java
@Autowired
private KafkaTemplate<String, ProductEvent> kafkaTemplate;

public void sendTestProductCreatedEvent(Long productId) {
    ProductEvent event = new ProductEvent(
        productId,
        ProductEventType.CREATED,
        ZonedDateTime.now()
    );
```

```
        kafkaTemplate.send("product-created", event);
    }
```

API Documentation

The API documentation is available via Swagger UI:

1. Start the application
2. Access Swagger UI at http://localhost:8080/api/swagger-ui.html

Development Workflow

1. **Pull the latest changes**:

   ```
   git pull origin main
   ```

2. **Create a feature branch**:

   ```
   git checkout -b feature/your-feature-name
   ```

3. **Implement your changes with tests**

4. **Verify all tests pass**:

   ```
   mvn verify
   ```

5. **Submit a pull request** for code review

Troubleshooting

Common Issues

1. **Port conflicts**: If port 8080 is already in use, you can change it:

```
mvn spring-boot:run -Dspring-boot.run.arguments=--server.port=8081
```

2. **Database connection issues**: Verify PostgreSQL is running:

```
docker ps | grep postgres
```

3. **Kafka connectivity issues**: Check if Kafka is running:

```
docker ps | grep kafka
```