📚SmartShop Documentation

Comprehensive documentation for the **SmartShop Microservices E-Commerce Platform**.

---

1. 📖Introduction

SmartShop is a distributed e-commerce platform built using a **microservices architecture**. It is designed for scalability, high performance, real-time payments, and secure authentication. Each service runs independently and communicates through REST APIs or Apache Kafka.

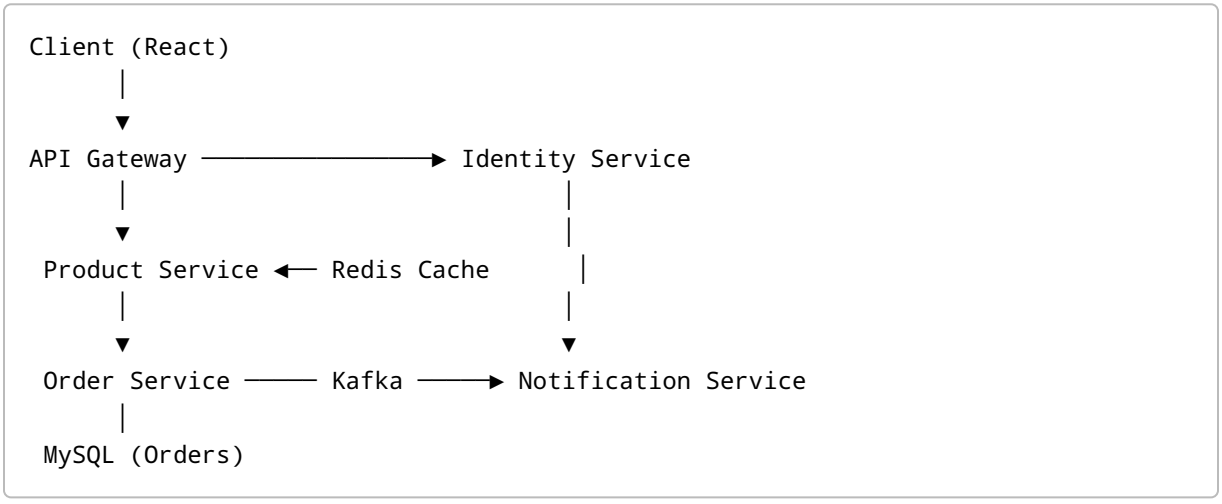
This documentation describes the system architecture, service-level details, endpoints, database schema, request flows, and deployment instructions.

---

2. 💣System Architecture

SmartShop consists of multiple independent components: - **React Frontend** (Client) - **API Gateway** (Routing & Security) - **Eureka Server** (Service Discovery) - **Microservices**: Identity, Product, Order, Notification - **Infrastructure**: MySQL, Kafka, Zookeeper, Redis

2.1 Architecture Diagram



---

3. 🧰Microservices Overview

3.1 Discovery Service (Eureka)

- Port: **8761**

- Purpose: Registers all microservices
- Tech: Netflix Eureka

3.2 API Gateway

- Port: **8080**
- Validates JWT tokens
- Routes traffic to internal services
- Tech: Spring Cloud Gateway (Reactive)

3.3 Identity Service

- Port: **8081**
- Features:
- User registration
- Login & JWT generation
- Password encryption using BCrypt
- Database: `smartshop_users`

3.4 Product Service

- Port: **8082**
- Features:
- Product creation (Admin only)
- Product listing
- Redis caching for high-speed reads
- Database: `smartshop_products`

3.5 Order Service

- Port: **8083**
- Features:
- Order creation
- Razorpay order initialization
- Kafka event publishing
- Database: `smartshop_orders`

3.6 Notification Service

- Port: **8084**
- Kafka consumer
- Logs pseudo-emails for placed orders

---

4. 📡Communication Flow

4.1 Authentication Flow

1. User enters login credentials.
2. Identity Service validates the user.
3. Generates JWT with roles & expiry.
4. API Gateway validates the token on each request.

4.2 Order Placement Flow

1. User places an order → API Gateway → Order Service
2. Order Service creates MySQL entry
3. Publishes `ORDER_PLACED` event to Kafka
4. Notification Service consumes event
5. Logs simulated email message

4.3 Payment Flow (Razorpay)

1. Backend generates Razorpay order ID
2. Frontend opens Razorpay checkout form
3. User completes test payment
4. Razorpay sends callback to frontend
5. Frontend sends verification request to backend
6. Order marked as **PAID**

---

5. 🗄️Database Schema

5.1 Users Table

Column	Type	Description
id	BIGINT	Primary key
name	VARCHAR	User name
email	VARCHAR	Unique email
password	VARCHAR	BCrypt hashed

5.2 Products Table

Column	Type
id	BIGINT
name	VARCHAR

Column	Type
price	DOUBLE
imageUrl	VARCHAR

5.3 Orders Table

Column	Type
id	BIGINT
userId	BIGINT
amount	DOUBLE
status	VARCHAR (PENDING/PAID)

---

6. 🔗REST API Endpoints

6.1 Identity Service

POST /auth/register
Registers new user.

POST /auth/login
Returns JWT token.

6.2 Product Service

GET /products
Returns list of products (cached).

POST /products/add (Admin only)
Adds new product.

6.3 Order Service

POST /orders/create
Creates a new order & initializes Razorpay.

POST /orders/verify
Verifies payment signature.

6.4 Notification Service

No public API. Listens to Kafka.

7. ⚙️Setup & Installation

Step 1 — Start Infrastructure

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

Step 2 — Create Databases

```
CREATE DATABASE smartshop_products;
CREATE DATABASE smartshop_orders;
CREATE DATABASE smartshop_users;
```

Step 3 — Start Backend Services

Run in this order: 1. Discovery Service 2. API Gateway 3. Identity Service 4. Product Service 5. Order Service 6. Notification Service

Step 4 — Start Frontend

```
cd smartshop-client
npm install
npm run dev
```

---

8. 🧪Testing

Functional Tests

- User registration/login
- Product listing with Redis cache
- Cart and checkout
- Razorpay payment flow
- Kafka event logging

Load Tests

- Apache JMeter recommended
- Product service performs **O(1)** reads with Redis

---

9. 🔮 Future Enhancements

- **Resilience4j circuit breakers**
- **Zipkin tracing**
- **Kubernetes deployment (K8s)**
- **Email/SMS real-world notifications**

---

10. Author

Documentation prepared for **SmartShop Microservices Portfolio Project**.