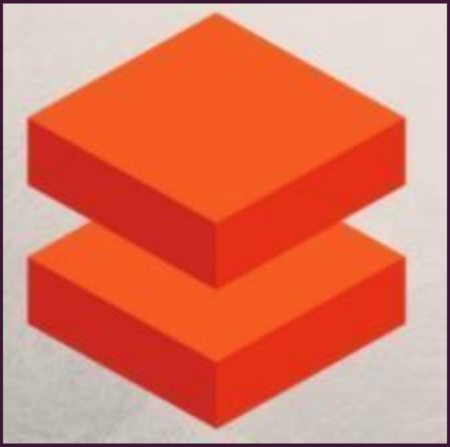
Capgemini



# Spring Boot Microservices- MASTER CLASS: Participant Guide

## Document Revision History

| Document Revision History Date | Revision No. | Author | Summary of Changes |
|---|---|---|---|
| 07-07-2023 | 1.0 | Alka Jhanwar | Version created for SME/BU Approval. |

Table of Contents

## Prerequisites:

1. Participants must have completed following pathways min 70% OR post assessments with min 50% score.
   https://degreed.com/pathway/k9ogoed1px?orgsso=capgemini

## Introduction:

In this masterclass workshop you will implement **E-commerce Application using** microservices architecture with Spring Boot.
Our goal is to develop a microservices-based e-commerce application using Spring Boot. The application will consist of three core microservices: Product Service, Order Service, and Payment Service. These microservices will handle the product catalog management, order processing, and payment integration aspects of the application. However, for the purpose of this problem statement, we will exclude the authentication and cart management features.

### Business Requirements:

**Product Catalog Service:** Maintain a catalog of products with details such as name, description, price, and availability.
**Order Service:** Handle order placement, order processing, and integration with payment gateways.
**Payment Service:** Manage payment processing.

### Solution Architecture:

The following components are involved in the architecture:

1. **API Gateway:** Acts as the entry point for clients and provides routing and load balancing functionalities. It forwards requests to the appropriate microservices.
2. **Service Registry:** Stores the registration information of all microservices, enabling service discovery and dynamic routing.
3. **Cloud Config Server**: Spring Cloud Config provides support for externalized configuration in a distributed system. With the Config Server, you have a central place to manage external properties for applications across all environments.
4. **Database:** Each microservice may have its own database to manage its specific data, ensuring loose coupling between services.
5. **Circuit Breaker Implementation (Using Resilience4j):** Use of the Circuit Breaker pattern can allow a microservice to continue operating when a related service fails, preventing the failure from cascading and giving the failing service time to recover.
6. Centralized logging System using ELK (Elasticsearch, Logstash and Kibana )

## Implementation:

Each microservice will be developed as a separate Spring Boot application, allowing independent development and deployment. Here's a brief overview of the implementation details:

The architecture for the e-commerce platform consists of the following microservices:

1. **Product Catalog Service:**
   - Responsible for managing product information.
   - Customers can retrieve product information, search for products based on various criteria, and view product details.

- Exposes REST APIs to retrieve product details, search for products, and manage product metadata.
- Stores product data in a dedicated database.

2. **Order Service:**
   - Handles the order placement and processing workflow.
   - Customers can create new orders by adding products to their order using the Order Service's APIs.
   - The Order Service calculates the total price based on the selected products and validates the availability of products (Communicates with product service).
   - If the products are available, the Order Service creates a new order with the associated products and generates an order ID.
   - Exposes REST APIs to create orders, update order status, and retrieve order information.
   - Communicates with the Product Service to update the quantity of the order.
   - Communicate with Payment service for payment processing
   - Stores order information in a dedicated database.

3. **Payment Service**
   - Handles payment processing.
   - The Payment Service updates the order status as paid and generates a transaction id.
   - Stores payment information in a dedicated database.

**Overall Flow:**

- Customer browses the product catalogue and selects products of interest.
- Customer adds selected products to the order using the Order Service's APIs.
- The Order Service calculates the total price and validates the availability of products.
- If the products are available, the Order Service creates a new order and generates an order ID.
- Customer proceeds to the payment stage, providing payment details.
- The Payment Service securely processes the payment and updates the order status as confirmed.

**Database Structure (For reference)**

**Product Table:**

| Column Name | Data Type | Description |
|---|---|---|
| Product_id | Primary Key(Number) | Unique identifier for a product |
| Name | VARCHAR | Name of the product |
| Price | Number | Price of the product |
| Category | VARCHAR | Category of the product |
| Quantity | Number | |

**Order Table:**

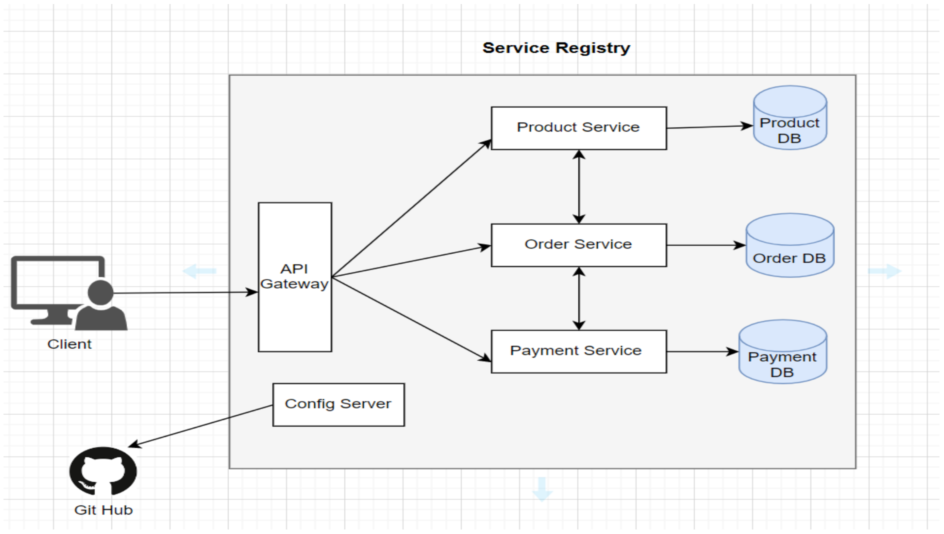
| Column Name | Data Type | Description |
|---|---|---|
| order_id | Primary Key | Unique identifier for an order |
| customer_name | VARCHAR | Name of the customer placing the order |
| customer_email | VARCHAR | Email of the customer placing the order |
| total_price | Number | Total price of the order |
| status | VARCHAR | Status of the order (e.g., pending, confirmed, delivered) |

| | | |
|---|---|---|
| order_date | Date | Date of order creation |
| product_id | TIMESTAMP | Timestamp of last update |
| order_quantity | Number | No of product ordered |

**Payment Table:**

| Column Name | Data Type | Description |
|---|---|---|
| payment_id | Primary Key(Number) | Unique identifier for a payment |
| order_id | Foreign Key | Reference to the order associated with the payment |
| amount | Number | Amount paid for the order |
| payment_status | VARCHAR | Status of the payment (e.g., pending, completed) |
| payment_date | Date | Date of payment completion |

**Application Architecture :**



Note: While authentication and cart management are excluded from this problem statement, they are important components of a comprehensive e-commerce application. Consider adding these features as future enhancements to improve user experience, security, and functionality.