# Spring Boot Microservices Case Studies

## Product Service

### 1. Product.java (Entity)

```
@Entity
public class Product {
   @Id
   private String id;
   private String name;
   private double price;
   private int stock;

   // Getters and setters
}
```

### 2. ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, String> {
}
```

### 3. ProductService.java

```
public interface ProductService {
   Product addProduct(Product product);
   Product getProduct(String id);
   List<Product> getAllProducts();
   String reduceStock(String id, int qty);
}
```

### 4. ProductServiceImpl.java

```
@Service
public class ProductServiceImpl implements ProductService {

   @Autowired
   private ProductRepository repository;

   @Override
   public Product addProduct(Product product) {
      return repository.save(product);
   }
```

```
    @Override
    public Product getProduct(String id) {
        return repository.findById(id).orElse(null);
    }

    @Override
    public List<Product> getAllProducts() {
        return repository.findAll();
    }

    @Override
    public String reduceStock(String id, int qty) {
        Product product = getProduct(id);
        if (product != null && product.getStock() >= qty) {
            product.setStock(product.getStock() - qty);
            repository.save(product);
            return "Stock updated";
        }
        return "Insufficient stock";
    }
}
```

## 5. ProductController.java

```
@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @PostMapping
    public Product addProduct(@RequestBody Product product) {
        return productService.addProduct(product);
    }

    @GetMapping("/{id}")
    public Product getProduct(@PathVariable String id) {
        return productService.getProduct(id);
    }

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    @PutMapping("/{id}/reduceStock")
    public String reduceStock(@PathVariable String id, @RequestParam int qty) {
        return productService.reduceStock(id, qty);
```

```
    }
}
```

### 6. SwaggerConfig.java (Optional)

```
@Configuration
public class SwaggerConfig {

    @Bean
    public OpenAPI apiInfo() {
        return new OpenAPI()
            .info(new Info()
            .title("Product Service API")
            .description("Product management microservice")
            .version("1.0"));
    }
}
```

## Order Service (with Circuit Breaker & Swagger)

### 1. Order.java (Entity)

```
@Entity
public class Order {
    @Id
    private String orderId;
    private String productId;
    private int quantity;
    private double totalAmount;

    // Getters and setters
}
```

### 2. Product.java (Model)

```
public class Product {
    private String id;
    private String name;
    private double price;
    private int stock;

    // Getters and setters
}
```

### 3. OrderRepository.java

```
public interface OrderRepository extends JpaRepository<Order, String> {
}
```

### 4. OrderService.java

```
public interface OrderService {
   String placeOrder(Order order);
}
```

### 5. OrderServiceImpl.java

```
@Service
public class OrderServiceImpl implements OrderService {

   @Autowired
   private OrderRepository orderRepository;

   @Autowired
   private RestTemplate restTemplate;

   private Map<String, Product> productCache = new HashMap<>();

   @Override
   @CircuitBreaker(name = "productService", fallbackMethod = "fallbackGetProduct")
   public String placeOrder(Order order) {
      Product product = restTemplate.getForObject("http://product-service/products/" +
order.getProductId(), Product.class);

      if (product == null || product.getStock() < order.getQuantity()) {
         return "Product not available or insufficient stock.";
      }

      double total = product.getPrice() * order.getQuantity();
      order.setTotalAmount(total);

      // Reduce stock
      restTemplate.put("http://product-service/products/" + order.getProductId() +
"/reduceStock?qty=" + order.getQuantity(), null);
      orderRepository.save(order);

      // cache product
      productCache.put(order.getProductId(), product);

      return "Order placed successfully with total: $" + total;
   }
```

```
    public String fallbackGetProduct(Order order, Throwable t) {
        Product cachedProduct = productCache.get(order.getProductId());
        if (cachedProduct != null && cachedProduct.getStock() >= order.getQuantity()) {
            double total = cachedProduct.getPrice() * order.getQuantity();
            order.setTotalAmount(total);
            orderRepository.save(order);
            return "Fallback: Order placed using cached product with total: $" + total;
        }
        return "Fallback: Product not available or insufficient stock (from cache).";
    }
}
```

## 6. OrderController.java

```
@RestController
@RequestMapping("/orders")
@Tag(name = "Order APIs", description = "Operations related to Orders")
public class OrderController {

    @Autowired
    private OrderService orderService;

    @PostMapping
    public ResponseEntity<String> placeOrder(@RequestBody Order order) {
        String result = orderService.placeOrder(order);
        return ResponseEntity.ok(result);
    }
}
```

## 7. SwaggerConfig.java (Optional)

```
@Configuration
public class SwaggerConfig {

    @Bean
    public OpenAPI apiInfo() {
        return new OpenAPI()
            .info(new Info()
            .title("Order Service API")
            .description("Handles order placement with Circuit Breaker")
            .version("1.0"));
    }
}
```

## 8. Application Class & RestTemplate Bean

```
@SpringBootApplication
```

```
@EnableEurekaClient
public class OrderServiceApplication {

   public static void main(String[] args) {
      SpringApplication.run(OrderServiceApplication.class, args);
   }

   @Bean
   public RestTemplate restTemplate() {
      return new RestTemplate();
   }
}
```

### 9. application.yml Sample

```
server:
 port: 8082

spring:
 application:
   name: order-service

eureka:
 client:
   service-url:
     defaultZone: http://localhost:8761/eureka

resilience4j:
 circuitbreaker:
   instances:
     productService:
       registerHealthIndicator: true
       slidingWindowSize: 5
       failureRateThreshold: 50
       waitDurationInOpenState: 10s
```

## Payment Service

### 1. Payment.java (Entity)

```
@Entity
public class Payment {
   @Id
   private String paymentId;
   private String orderId;
   private String paymentStatus;
   private double amount;
```

```
    // Getters and setters
}
```

### 2. PaymentRepository.java

```
public interface PaymentRepository extends JpaRepository<Payment, String> {
}
```

### 3. PaymentService.java

```
public interface PaymentService {
    Payment processPayment(Payment payment);
}
```

### 4. PaymentServiceImpl.java

```
@Service
public class PaymentServiceImpl implements PaymentService {

    @Autowired
    private PaymentRepository paymentRepository;

    @Override
    public Payment processPayment(Payment payment) {
        payment.setPaymentStatus("SUCCESS");
        return paymentRepository.save(payment);
    }
}
```

### 5. PaymentController.java

```
@RestController
@RequestMapping("/payments")
public class PaymentController {

    @Autowired
    private PaymentService paymentService;

    @PostMapping
    public ResponseEntity<Payment> makePayment(@RequestBody Payment payment) {
        return ResponseEntity.ok(paymentService.processPayment(payment));
    }
}
```

### 6. application.yml Sample

```
server:
  port: 8083

spring:
  application:
    name: payment-service

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
```

## Eureka Server

### 1. Main Application

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

### 2. application.yml

```
server:
  port: 8761

spring:
  application:
    name: eureka-server

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

## API Gateway (Spring Cloud Gateway)

### 1. Main Application

```
@SpringBootApplication
@EnableEurekaClient
public class ApiGatewayApplication {
   public static void main(String[] args) {
      SpringApplication.run(ApiGatewayApplication.class, args);
   }
}
```

### 2. application.yml

```
server:
 port: 8080

spring:
 application:
  name: api-gateway

 cloud:
  gateway:
   routes:
    - id: product-service
      uri: http://localhost:8081
      predicates:
       - Path=/products/**
    - id: order-service
      uri: http://localhost:8082
      predicates:
       - Path=/orders/**
    - id: payment-service
      uri: http://localhost:8083
      predicates:
       - Path=/payments/**

eureka:
 client:
  service-url:
   defaultZone: http://localhost:8761/eureka
```