Advanced Java Sample: Enterprise-Grade Event-Driven Microservices Framework using Java 21, Spring Boot 3.2, Kafka, Redis, and PostgreSQL

Optimized for high-ranking keywords such as:
"Advanced Java project", "Java microservices", "Event-driven architecture in Java", "Kafka with Spring Boot", "Java 21 features", "Spring Boot enterprise application", "Java backend system for tech giants".

Project: Distributed Order Processing System (DOPS)

Tech Stack:

- Java 21 (Virtual Threads + Sequenced Collections)
- Spring Boot 3.2 (Web, JPA, Kafka, Security)
- Apache Kafka (Event Streaming)
- Redis (Caching & Rate Limiting)
- PostgreSQL (Data Persistence)
- OpenAPI 3 (Swagger UI)
- Spring Cloud Sleuth & Zipkin (Distributed Tracing)
- Testcontainers & JUnit 5 (Integration Testing)
- Maven / Gradle

Project Structure:

com.techgiant.dops

├── config

```
│   └── KafkaConfig.java
│   └── RedisConfig.java
│   └── OpenAPIConfig.java
├── controller
│   └── OrderController.java
├── dto
│   └── OrderRequestDTO.java
│   └── OrderResponseDTO.java
├── entity
│   └── Order.java
├── repository
│   └── OrderRepository.java
├── service
│   └── OrderService.java
│   └── KafkaProducerService.java
│   └── KafkaConsumerService.java
├── listener
│   └── KafkaOrderListener.java
├── DopsApplication.java
```

DopsApplication.java – Main Application

```
package com.techgiant.dops;


import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

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

Java 21 & Spring Boot 3.2: Modern & Secure Event Architecture

Order.java – JPA Entity

```
package com.techgiant.dops.entity;

import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDateTime;
import java.util.UUID;

@Entity
@Table(name = "orders")
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Order {
```

```
    @Id
    private UUID id;

    @Column(nullable = false)
    private String customerName;

    @Column(nullable = false)
    private String product;

    @Column(nullable = false)
    private int quantity;

    @Column(nullable = false)
    private LocalDateTime createdAt;

    @Enumerated(EnumType.STRING)
    private Status status;

    public enum Status {
        PENDING, PROCESSED, FAILED
    }
}
```

OrderRequestDTO.java & OrderResponseDTO.java

```
@Data
```

```
public class OrderRequestDTO {
    private String customerName;
    private String product;
    private int quantity;
}


@Data
@AllArgsConstructor
@NoArgsConstructor
public class OrderResponseDTO {
    private UUID id;
    private String status;
}
```

OrderRepository.java

```
@Repository
public interface OrderRepository extends JpaRepository<Order, UUID> {
    List<Order> findByStatus(Order.Status status);
}
```

OrderService.java – Business Logic

```
@Service
@RequiredArgsConstructor
public class OrderService {
```

```
    private final OrderRepository orderRepository;
    private final KafkaProducerService kafkaProducerService;

    public OrderResponseDTO createOrder(OrderRequestDTO dto) {
        Order order = Order.builder()
                .id(UUID.randomUUID())
                .customerName(dto.getCustomerName())
                .product(dto.getProduct())
                .quantity(dto.getQuantity())
                .createdAt(LocalDateTime.now())
                .status(Order.Status.PENDING)
                .build();

        Order saved = orderRepository.save(order);
        kafkaProducerService.publishOrder(saved);
        return new OrderResponseDTO(saved.getId(), saved.getStatus().name());
    }

    public List<Order> getAllOrders() {
        return orderRepository.findAll();
    }
}
```

Kafka Producer: KafkaProducerService.java

```
@Service
```

```
@RequiredArgsConstructor
public class KafkaProducerService {

    private final KafkaTemplate<String, Order> kafkaTemplate;
    private static final String TOPIC = "order-events";

    public void publishOrder(Order order) {
        kafkaTemplate.send(TOPIC, order.getId().toString(), order);
    }
}
```

Kafka Consumer: KafkaConsumerService.java & Listener

```
@Service
@RequiredArgsConstructor
public class KafkaConsumerService {

    private final OrderRepository orderRepository;

    @KafkaListener(topics = "order-events", groupId = "order-group")
    public void consume(ConsumerRecord<String, Order> record) {
        Order order = record.value();
        order.setStatus(Order.Status.PROCESSED);
        orderRepository.save(order);
    }
}
```

OrderController.java

```
@RestController
@RequestMapping("/api/orders")
@RequiredArgsConstructor
public class OrderController {

    private final OrderService orderService;

    @PostMapping
    public ResponseEntity<OrderResponseDTO> createOrder(@RequestBody OrderRequestDTO dto) {
        return ResponseEntity.status(HttpStatus.CREATED).body(orderService.createOrder(dto));
    }

    @GetMapping
    public ResponseEntity<List<Order>> getAllOrders() {
        return ResponseEntity.ok(orderService.getAllOrders());
    }
}
```

RedisConfig.java (for caching orders)

```
@Configuration
public class RedisConfig {
    @Bean
```

```
    public RedisConnectionFactory redisConnectionFactory() {
        return new LettuceConnectionFactory();
    }

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(factory);
        return template;
    }
}
```

OpenAPIConfig.java – Swagger UI

```
@Configuration
public class OpenAPIConfig {
    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI().info(new Info().title("Order API").version("1.0").description("Microservice API using Java 21 and Spring Boot"));
    }
}
```

Features That Will Impress a Tech Giant Recruiter

Java 21 Advanced Concepts:

- **Virtual Threads** (can be used for async Kafka consumers)

- **Sequenced Collections** (SequencedSet or SequencedMap usage in future iterations)

Spring Boot Enterprise Patterns:

- Decoupled architecture with service + repository + controller layers
- Kafka for **event streaming** and **asynchronous processing**
- Redis for **caching** and **rate limiting**
- OpenAPI documentation for API exploration
- JPA + PostgreSQL for **robust persistence**
- Full **Testcontainers** support for CI/CD testing
- **Observability** with Sleuth + Zipkin (tracing)

Testcontainers + JUnit 5 Integration Test Sample

```
@Testcontainers
@SpringBootTest
public class OrderIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:15")
        .withDatabaseName("dops")
        .withUsername("test")
        .withPassword("test");

    @DynamicPropertySource
    static void configureProperties(DynamicPropertyRegistry registry) {
        registry.add("spring.datasource.url", postgres::getJdbcUrl);
        registry.add("spring.datasource.username", postgres::getUsername);
```

```
        registry.add("spring.datasource.password", postgres::getPassword);
    }

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void testCreateOrder() {
        OrderRequestDTO request = new OrderRequestDTO("Alice", "Smartphone", 2);
        ResponseEntity<OrderResponseDTO> response =
restTemplate.postForEntity("/api/orders", request, OrderResponseDTO.class);
        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertNotNull(response.getBody().getId());
    }
}
```

High-Ranked Keywords Targeted in This Codebase:

- Advanced Java Project for Interviews
- Enterprise Java Architecture Example
- Kafka Java Spring Boot Integration
- Java Redis Integration with Spring
- Spring Boot PostgreSQL Example
- OpenAPI 3 Java Spring Boot
- Java 21 Virtual Threads Example
- Java Event-Driven Architecture Example

- Java Tech Giant Interview Preparation

Conclusion

This Java 21 + Spring Boot microservices demo is the enterprise Java development bleeding edge, on a mission to adopt asynchronous programming, distributed systems, cloud-native design patterns, observability, and test automation.