Advanced Java Sample Project: Reactive Microservices for Real-Time Inventory Management

High-Ranking Java Keywords Included

- Advanced Java Architecture
- Spring WebFlux Reactive Programming
- Hexagonal Microservices in Java
- Event Sourcing with Kafka
- CQRS in Java Microservices
- Reactive Redis Caching
- OpenTelemetry Distributed Tracing
- Dockerized Java Microservice
- JWT Authentication in Spring WebFlux
- Java 17 Functional Reactive Streams

Project Summary

We are building a **real-time, reactive inventory management system** using **advanced enterprise-level Java**. This system supports non-blocking operations with **Spring WebFlux**, allows **reactive event sourcing and CQRS**, integrates **Kafka as an event log**, and supports **JWT security and Redis caching** — all while adhering to **clean architecture principles**.

Project Structure (Hexagonal Architecture)

inventory-system/

├── application/

```
│   ├── service/
│   └── port/in/
│   └── port/out/
├── domain/
│   └── model/
│   └── event/
│   └── command/
├── infrastructure/
│   ├── kafka/
│   ├── redis/
│   ├── repository/
│   ├── config/
│   └── web/
├── bootstrap/
│   └── InventoryApplication.java
└── docker/
```

Dependencies (build.gradle.kts)

```
plugins {
    id("org.springframework.boot") version "3.1.0"
    id("io.spring.dependency-management") version "1.1.0"
    kotlin("jvm") version "1.8.20"
    kotlin("plugin.spring") version "1.8.20"
}
```

```
dependencies {
    implementation("org.springframework.boot:spring-boot-starter-webflux")
    implementation("org.springframework.boot:spring-boot-starter-data-redis-reactive")
    implementation("org.springframework.kafka:spring-kafka")
    implementation("io.projectreactor.kafka:reactor-kafka:1.3.17")
    implementation("org.springframework.boot:spring-boot-starter-security")
    implementation("io.jsonwebtoken:jjwt-api:0.11.5")
    implementation("io.opentelemetry.instrumentation:opentelemetry-spring-boot-starter:2.0.0-alpha")
    implementation("org.mapstruct:mapstruct:1.5.5.Final")
    kapt("org.mapstruct:mapstruct-processor:1.5.5.Final")
}
```

Key Domain Classes

InventoryItem.java

```
public record InventoryItem(UUID id, String sku, String name, int quantity) {
}
```

InventoryEvent.java

```
public sealed interface InventoryEvent permits ItemAdded, ItemRemoved {
    UUID itemId();
    Instant occurredOn();
}


public record ItemAdded(UUID itemId, int quantity, Instant occurredOn) implements InventoryEvent {}
```

```java
public record ItemRemoved(UUID itemId, int quantity, Instant occurredOn) implements InventoryEvent {}
```

Command Handlers (CQRS)

AddItemCommand.java

```java
public record AddItemCommand(UUID itemId, String sku, String name, int quantity) {}
```

AddItemHandler.java

```java
@Component
public class AddItemHandler {

    private final InventoryEventStore eventStore;

    public AddItemHandler(InventoryEventStore eventStore) {
        this.eventStore = eventStore;
    }

    public Mono<Void> handle(AddItemCommand command) {
        InventoryEvent event = new ItemAdded(command.itemId(), command.quantity(), Instant.now());
        return eventStore.save(event);
    }
}
```

Reactive Event Store

InventoryEventStore.java

```
public interface InventoryEventStore {
    Mono<Void> save(InventoryEvent event);
    Flux<InventoryEvent> findByItemId(UUID itemId);
}
```

Kafka Event Publisher

```
@Component
public class KafkaInventoryEventStore implements InventoryEventStore {

    private final KafkaSender<String, InventoryEvent> kafkaSender;

    public KafkaInventoryEventStore(KafkaSender<String, InventoryEvent> kafkaSender) {
        this.kafkaSender = kafkaSender;
    }

    @Override
    public Mono<Void> save(InventoryEvent event) {
        SenderRecord<String, InventoryEvent, UUID> record = SenderRecord.create(
            new ProducerRecord<>("inventory-events", event.itemId().toString(), event),
            event.itemId()
        );
        return kafkaSender.send(Mono.just(record)).then();
    }
    @Override
    public Flux<InventoryEvent> findByItemId(UUID itemId) {
        // In event sourcing, you would replay all events for an item
        return Flux.empty(); // Simulated — real impl would consume from Kafka
    }
}
```

JWT Security Configuration

SecurityConfig.java

```
@EnableWebFluxSecurity
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain securityFilterChain(ServerHttpSecurity http) {
        return http
            .csrf().disable()
            .authorizeExchange()
                .pathMatchers("/api/**").authenticated()
                .anyExchange().permitAll()
            .and()
            .oauth2ResourceServer()
                .jwt()
            .and().and()
            .build();
    }
}
```

Web Controller (Reactive Endpoint)

```
@RestController

@RequestMapping("/api/inventory")

public class InventoryController {



    private final AddItemHandler addItemHandler;



    public InventoryController(AddItemHandler addItemHandler) {

        this.addItemHandler = addItemHandler;

    }



    @PostMapping

    public Mono<ResponseEntity<Void>> addItem(@RequestBody AddItemCommand
command) {

        return addItemHandler.handle(command)

                .thenReturn(ResponseEntity.status(HttpStatus.CREATED).build());

    }

}
```

Redis Reactive Caching Layer

```
@Service

public class InventoryCache {
    private final ReactiveRedisTemplate<String, InventoryItem> redisTemplate;

    public InventoryCache(ReactiveRedisTemplate<String, InventoryItem> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    public Mono<InventoryItem> get(String sku) {
        return redisTemplate.opsForValue().get(sku);
    }

    public Mono<Boolean> set(String sku, InventoryItem item) {
        return redisTemplate.opsForValue().set(sku, item);
    }
}
```

OpenTelemetry Integration

```
# application.yml
otel:
  tracing:
    enabled: true
    exporter: otlp
    endpoint: http://localhost:4317
```

OpenTelemetryConfig.java

```
@Configuration
public class OpenTelemetryConfig {
    @PostConstruct
    public void init() {
        OpenTelemetrySdk.builder()
            .setTracerProvider(SdkTracerProvider.builder().build())
            .buildAndRegisterGlobal();
    }
}
```

Dockerfile

```
FROM eclipse-temurin:17-jdk
WORKDIR /app
COPY build/libs/inventory-system.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Advanced Java Concepts Demonstrated

- **Reactive Streams API** via **Project Reactor**
- **Spring WebFlux** non-blocking I/O
- **Hexagonal Architecture (Ports & Adapters)**
- **CQRS** for separating reads and writes
- **Event Sourcing** with Kafka and reactive event store
- **JWT-based Stateless Authentication**
- **Redis-based caching** using ReactiveRedisTemplate
- **OpenTelemetry instrumentation** for tracing and observability

- **Java 17 Records & Sealed Classes**
- **Fully Dockerized Microservice**