Real-Time Event-Driven Reactive Microservice for Fraud Detection

---

Technologies and High-Ranking Keywords

Java 17, Spring Boot 3.x, Spring WebFlux, Reactive Streams, Project Reactor, Redis, PostgreSQL, Kafka, JWT, OAuth2, Resilience4j, OpenAPI, Hexagonal Architecture, Clean Code, Builder Pattern, Event-Driven Architecture, Enterprise Integration Pattern, CI/CD, High Availability, Scalable Microservices.

Directory Structure

fraud-detection-service/

├── application/

│ ├── config/

│ ├── controller/

│ ├── dto/

│ └── service/

├── domain/

│ ├── model/

│ └── service/

├── infrastructure/

│ ├── kafka/

│ ├── redis/

│ └── repository/

├── common/

│ └── exception/

├── FraudDetectionApplication.java

---

1. FraudDetectionApplication.java

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

---

2. Security Configuration with JWT and OAuth2

```
@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain securityFilterChain(ServerHttpSecurity http) {
        return http
            .csrf().disable()
            .authorizeExchange()
                .pathMatchers("/api/v1/health", "/swagger-ui.html").permitAll()
                .anyExchange().authenticated()
            .and()
```

```
            .oauth2ResourceServer(ServerHttpSecurity.OAuth2ResourceServerSpec::jwt)
            .build();
    }
}
```

---

3. DTO with Builder Pattern

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class TransactionEventDTO {
    private String transactionId;
    private String userId;
    private BigDecimal amount;
    private String location;
    private LocalDateTime timestamp;
}
```

---

4. Domain Service (Clean Architecture)

```
@Service
@RequiredArgsConstructor
public class FraudAnalysisService {

    private final RiskScoringStrategyFactory strategyFactory;
```

```
    public Mono<Boolean> isFraudulent(TransactionEventDTO event) {
        RiskScoringStrategy strategy = strategyFactory.getStrategy(event);
        double score = strategy.evaluate(event);
        return Mono.just(score > 0.85);
    }
}
```

---

5. Strategy Pattern for Fraud Scoring

```
public interface RiskScoringStrategy {
    double evaluate(TransactionEventDTO event);
}

@Component
public class HighAmountStrategy implements RiskScoringStrategy {
    public double evaluate(TransactionEventDTO event) {
        return event.getAmount().compareTo(BigDecimal.valueOf(5000)) > 0 ? 0.9 : 0.1;
    }
}

@Component
public class LocationAnomalyStrategy implements RiskScoringStrategy {
    public double evaluate(TransactionEventDTO event) {
        return event.getLocation().equalsIgnoreCase("unknown") ? 0.95 : 0.05;
    }
}
```

Factory

```
@Component
@RequiredArgsConstructor
public class RiskScoringStrategyFactory {
    private final List<RiskScoringStrategy> strategies;

    public RiskScoringStrategy getStrategy(TransactionEventDTO event) {
        // Dynamic evaluation logic
        return event.getAmount().compareTo(BigDecimal.valueOf(5000)) > 0
            ? new HighAmountStrategy()
            : new LocationAnomalyStrategy();
    }
}
```

---

6. Reactive REST Controller (Spring WebFlux)

```
@RestController
@RequestMapping("/api/v1/fraud")
@RequiredArgsConstructor
public class FraudDetectionController {

    private final FraudAnalysisService fraudService;

    @PostMapping("/analyze")
    public Mono<ResponseEntity<Map<String, Object>>> analyze(@RequestBody
TransactionEventDTO dto) {
```

```
        return fraudService.isFraudulent(dto)
                .map(isFraud -> Map.of("fraudulent", isFraud))
                .map(body -> ResponseEntity.ok().body(body));
    }
}
```

---

7. Kafka Listener for Event-Driven Architecture

```
@Component
@RequiredArgsConstructor
public class TransactionEventListener {

    private final FraudAnalysisService fraudService;

    @KafkaListener(topics = "transaction-events", groupId = "fraud-detector")
    public void onEvent(String payload) {
        TransactionEventDTO event = new Gson().fromJson(payload,
TransactionEventDTO.class);
        fraudService.isFraudulent(event)
            .subscribe(isFraud -> {
                if (isFraud) {
                    log.warn("FRAUD DETECTED: {}", event);
                    // Call alerting system or blacklist service
                }
            });
    }
```

```
}
```

---

8. Redis Caching (Reactive)

```
@Service
@RequiredArgsConstructor
public class RiskCacheService {

    private final ReactiveRedisTemplate<String, TransactionEventDTO> redisTemplate;

    public Mono<Void> cacheRiskScore(TransactionEventDTO event, double score) {
        return redisTemplate.opsForValue()
                .set("risk:" + event.getTransactionId(), event, Duration.ofMinutes(15))
                .then();
    }

    public Mono<TransactionEventDTO> getCachedRisk(String transactionId) {
        return redisTemplate.opsForValue().get("risk:" + transactionId);
    }
}
```

---

9. PostgreSQL Repository (Reactive)

```
@Repository
public interface ReactiveTransactionRepository extends
ReactiveCrudRepository<TransactionEvent, String> {
    Flux<TransactionEvent> findByUserId(String userId);
```

```
}
```

---

10. Global Exception Handler

```
@RestControllerAdvice
public class GlobalErrorHandler {

    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<String> handleRuntime(RuntimeException ex) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Unexpected error: " + ex.getMessage());
    }
}
```

---

11. Swagger/OpenAPI Configuration

```
@Configuration
public class OpenAPIConfig {

    @Bean
    public OpenAPI springShopOpenAPI() {
        return new OpenAPI()
            .info(new Info().title("Fraud Detection API")
            .description("Reactive Fraud Detection using Spring WebFlux")
            .version("v1.0"));
    }
}
```

12. Reactive Unit Tests with WebTestClient

```
@WebFluxTest(controllers = FraudDetectionController.class)
public class FraudDetectionControllerTest {

    @Autowired
    private WebTestClient webTestClient;

    @MockBean
    private FraudAnalysisService fraudService;

    @Test
    void testAnalyzeFraud() {
        TransactionEventDTO dto = TransactionEventDTO.builder()
                .transactionId("tx123")
                .userId("u456")
                .amount(BigDecimal.valueOf(9000))
                .location("US")
                .timestamp(LocalDateTime.now())
                .build();

        when(fraudService.isFraudulent(any())).thenReturn(Mono.just(true));

        webTestClient.post()
            .uri("/api/v1/fraud/analyze")
```

```
            .bodyValue(dto)
            .exchange()
            .expectStatus().isOk()
            .expectBody()
            .jsonPath("$.fraudulent").isEqualTo(true);
    }
}
```

---

13. application.yml Configuration

```
spring:
  application:
    name: fraud-detection-service
  r2dbc:
    url: r2dbc:postgresql://localhost:5432/frauddb
    username: postgres
    password: postgres
  kafka:
    bootstrap-servers: localhost:9092
    consumer:
      group-id: fraud-detector
  redis:
    host: localhost
    port: 6379

server:
```

```
  port: 8080

logging:
  level:
    root: INFO
```

---

14. Key Design Decisions

Decision	Reason
Reactive stack (WebFlux)	High throughput, non-blocking, perfect for real-time data
Hexagonal Architecture	Maintainability, testability, separation of concerns
Kafka	Event-driven architecture, asynchronous decoupled communication
Redis	Caching for performance & rate-limiting
Strategy Pattern	Extensible and testable fraud scoring logic
JWT + OAuth2	Secure API integration in multi-tenant environments

---

15. High-Value Keywords Embedded in Context

- **Enterprise Java for Scalable Microservices**
- **Reactive Spring Boot with Project Reactor**
- **JWT OAuth2 Secure REST API**
- **Real-Time Kafka Event Processing**
- **Redis Caching for Low Latency Applications**
- **PostgreSQL with R2DBC for Reactive Persistence**
- **Clean Code with Hexagonal Architecture**
- **Risk Detection Algorithms using Strategy Pattern**

- **High Availability Java Application Design**
- **Unit Testing with WebTestClient and Mockito**