CODE REVIEW

TRAINEE: MICHAEL OKYERE ADAMS

What Was Done Right

- **Excellent microservices architecture** with proper service decomposition - Auth Service, Restaurant Service, and Order Service are well-separated by domain boundaries.
- **Solid implementation of enterprise patterns** including Service Registry (Eureka), API Gateway, and Circuit Breaker (Resilience4j) for fault tolerance.
- **Proper asynchronous communication** using Apache Kafka for event-driven architecture enabling loose coupling between services.
- **Good security implementation** with JWT-based authentication and proper authorization flow through the API Gateway.
- **Well-structured service communication** with synchronous REST calls for immediate data needs and asynchronous messaging for events.
- **Clear service boundaries** with each microservice having its own domain models and DTOs for data transfer.

Areas for Improvement

- **Consider using Java Records** for DTOs instead of traditional classes to improve immutability and reduce boilerplate code.
- **Use a logging framework** like SLF4J with Logback instead of System.out.println for better log management and configuration.
- **Implement health checks** using Spring Boot Actuator for better monitoring and service discovery integration.
- **Add API versioning strategy** to ensure backward compatibility as services evolve.
- **Add comprehensive testing** including unit tests, integration tests, and contract testing between services.

Overall Assessment

- This is a **well-architected microservices platform** that demonstrates solid understanding of distributed systems principles. Your implementation showcases:
- **i. Strong Architectural Foundation:** Proper service decomposition with clear boundaries and responsibilities across Auth, Restaurant, and Order services.
- **ii. Enterprise-Grade Patterns:** Effective implementation of Service Discovery, API Gateway, Circuit Breaker, and Event-Driven Architecture.
- **iii. Modern Technology Stack:** Excellent use of Spring Cloud ecosystem with proper Kafka integration for asynchronous messaging.
- **Solid microservices implementation** that follows industry best practices. The service-to-service communication patterns and data flow are well-designed. Applying the suggested improvements would enhance the robustness and maintainability of the platform.