Peer Review Feedback

Project: ByteBites Restaurant Microservices | **Peer-Code:** Patrick Appiah

Strengths (Kudos!)

1. **Excellent project structure** – Clean modularization and separation of concerns.
2. **Great API mapping** and versioning– Consistent and logical endpoint design.
3. **Excellent use of records as DTOs** – Well-integrated and secure.
4. **Caching usage** – Smart performance optimization.
5. **Use of bootstrap properties**: Spring Cloud features that require early bootstrapping
6. **Centralized security:** Reduces code repetition
7. **Audit Log :** A plus

Areas for Improvement

1. Code Quality & Best Practices

- **Comments vs. Self-Documenting Code**:
    - Avoid commented dependencies (e.g., in auth-service/pom.xml).
    - Follow AmaliTech's practice of letting code explain itself (comments and documentations only for complex logic).
- **Boilerplate Reduction**:
    - Use **Lombok** (e.g., @Data, @Builder) to minimize repetitive code (getters/setters, constructors).
- **Exception Handling**:
    - Let the **global exception handler** manage all exceptions. Avoid manual handling unless absolutely necessary.
    - **Security**: In auth-service security config, add AccessDeniedException handling alongside AuthenticationEntryPoint.

2. Design & Architecture

- **Shared DTOs/Configs**:
    - **Duplicate DTOs** (e.g., RestaurantDTO in order-service and restaurant-service) should be moved to shared-lib.
    - **Centralize configurations**:
        - RabbitMQ settings, server ports, and management props should live in the **centralized config service**.
        - **Security Risk**: server-config-repo must be **private** (public exposure compromises security).

- **CRUD Gaps & Priorities**:
    - Add **update/delete operations** for User, Restaurant, and MenuItem entities (critical for maintenance). More important and necessary than OAuth2 implementation

3. Security & Validation

- **Token Implementation**:
    - **Refresh Token**: If not needed, remove the placeholder code; otherwise, implement it fully.
- **Role Management**:
    - Refactor userRole into a **separate** enum (single-responsibility principle like orderstatus).

4. Modernization & Maintenance

- **Feign Resilience**:
    - Add **fallback methods** to OpenFeign clients for fault tolerance.
- **API URLs**:
    - Simplify @RequestMapping (e.g., /api/v1/restaurants in MenuController with method-level paths).

5. **Missing implementations**

- No circuit breaker
- No distributed tracing

Actionable Summary

Priority	Issue	Suggested Fix
High	Public server-config-repo	Make repository **private** immediately.
High	Missing CRUD ops	Implement update/delete for User, Restaurant, MenuItem.
Medium	Centralized configs	Move RabbitMQ, ports, etc., to config service.
Low	Lombok adoption	Add annotations to reduce boilerplate.

Overall: Solid foundation with clear opportunities to align with best practices. Great work!