Code Review Report

Project: Restaurant Service

Reviewer: Winful Kofi Taylor

Reviewee: Yaw Frimpong

Date: July 11, 2025

What Was Done Right

Clean Architecture & Layer Separation

- Controllers, services, DTOs, and exceptions are well-separated.
- Use of annotations like @Service, @RestController promotes clarity.

Validation & Input Integrity

- DTOs are annotated with @NotBlank, @Email, @Pattern, etc.
- Compact constructor normalizes data (trim, lowercase, format).

Global Exception Handling

- Centralized via @RestControllerAdvice.
- Handles validation errors, custom exceptions, and general fallback.

Observability & Logging

- SLF4J + MDC used extensively.
- Logs include contextual metadata (operation, IDs).

Caching

- Efficient use of @Cacheable and @CacheEvict.
- Independent caching for performance.

Resilience & Retry Strategies

- Uses @RateLimiter and @Retry.
- Fallback methods for graceful failure handling.

Security Integration

- Role-based access via @PreAuthorize.
- Authenticated user info used securely in logic.

Testing

- Unit tests cover positive and negative service paths.
- Integration tests use TestContainers and MockMvc.

Areas for Improvement

Service vs Async Service Duplication

- Duplicate logic should be refactored using delegation.

Missing Tests

- No tests for AsyncRestaurantService, menu controllers, exception scenarios.

Global Exception - Fallback Logic

- Avoid revealing stack trace based on username. Use environment profile instead.

Hardcoded Role Strings

- Use constants or enums to avoid typos and improve maintainability.

Event Publisher Contracts

- Document event behavior and add test coverage for listeners.

Cache Inconsistency Risk

- Ensure consistent cache key names across operations.

Summary

Overall, this is a robust, secure, and well-structured codebase following modern Spring Boot microservice patterns. Minor improvements in test coverage and code reuse will further enhance its reliability and maintainability.