Linus Tabari Code Review

What went well:

1. The implementation is generally solid with good separation of concerns and proper use of Spring Security features.
2. The validation implementation effectively uses annotations for required fields, data formats, and clear error messages while differentiating between create and update operations and the program also handles exceptions gracefully with clear responses, proper logging, and security-aware defaults.
3. The GlobalExceptionHandler effectively centralizes error handling for validation, domain, and security exceptions, returning structured JSON responses with proper HTTP status codes and logging.
4. Your logging effectively tracks key operations with relevant metadata (IDs, user actions) using appropriate log levels and follows a consistent pattern throughout the services.
5. Your unit tests provide comprehensive coverage of service methods with proper mocking, clear organization, effective assertions, and thorough exception testing.
6. Your integration tests effectively use TestContainers to validate database operations, API flows, and Kafka events with comprehensive coverage of success/failure scenarios and proper security testing.

Areas for Improvement:

1. The exception handling could be enhanced by adding more custom exceptions (e.g., for business logic or rate-limiting), including request IDs for better traceability, standardizing error codes, and masking sensitive data in error responses
2. Transition to structured key-value logging (instead of concatenated strings), add correlation IDs for traceability, and enrich error logs with more context.
3. Consider reducing boilerplate with test data builders, improving verification precision, adding edge cases, enhancing readability, and expanding Kafka event validation.

Future Improvements:

1. Integrate log aggregation tools, add audit logging for sensitive actions, implement distributed tracing, and log business metrics for analytics.
2. Implement test data builders, Kafka consumer helpers, concurrent tests, and performance metrics to strengthen reliability and maintainability.
3. Distributed Tracing: Implement OpenTelemetry or Sleuth with Zipkin/Jaeger to track cross-service requests and debug latency issues.
4. Advanced Caching: Integrate Redis for high-traffic data (menus, restaurants) with Kafka-driven cache invalidation.

Overall, this microservice project demonstrates strong fundamentals in security, testing, and architecture, with well-structured code, comprehensive unit/integration tests, and proper containerized dependencies. The JWT-based authentication, TestContainers-powered integration tests, and clear separation of concerns reflect mature development practices.