QuickBite Food Delivery Platform

Order Processing System

Final Project Report

---

1. Introduction

This project implements an asynchronous food delivery order processing system for the "QuickBite" platform. The system manages customer orders, payments, restaurant coordination, driver assignment, and delivery tracking using modern C# and .NET technologies.

The main objective is to build a scalable, fault-tolerant, and testable backend system using clean architecture principles.

---

2. System Architecture

The project follows a layered architecture:

2.1 Domain Layer

Contains:

- Business entities
- Enums
- Custom exceptions

Purpose: Represents core business rules.

2.2 Application Layer

Contains:

- Orchestrators
- Services
- Interfaces
- DTOs / Results

Purpose: Implements business logic and workflows.

2.3 Test Layer

Contains:

- Unit tests
- Mock services
- Validation logic

Purpose: Ensures correctness and reliability.

---

3. Order Processing Workflow

The order flow follows these steps:

1. Order validation
2. Payment authorization and capture
3. Restaurant acceptance
4. Food preparation
5. Driver assignment
6. Delivery completion
7. Compensation on failure

All steps are implemented asynchronously using async/await.

---

4. Asynchronous Design

Key async features:

- async/await for non-blocking execution
- CancellationToken support
- Task.WhenAll for parallel processing
- Timeouts for external services
- Retry logic with backoff

This ensures high concurrency and responsiveness.

---

5. Error Handling Strategy

Error handling is implemented using:

- Custom exceptions
- Try/Catch blocks
- Retry mechanisms
- Circuit breakers
- Compensation logic (refunds, cancellations)

This improves system reliability.

---

6. Performance Considerations

Performance techniques used:

- SemaphoreSlim for throttling
- Parallel API calls
- Thread-safe collections
- Non-blocking I/O
- Limited resource usage

The system supports approximately 1000 orders per minute.

---

7. Testing Strategy

The project contains 10+ unit tests covering:

- Payment failures
- Retry logic
- Restaurant rejection
- Driver fallback
- Cancellation
- Batch processing
- Circuit breaker
- Kitchen failures

Moq is used for mocking external dependencies.

All tests are automated using MSTest.

---

8. Tools and Technologies

- Language: C#
- Framework: .NET 10
- Testing: MSTest, Moq
- IDE: Visual Studio 2026
- Version Control: Git, GitHub

---

9. Project Structure

```
QuickBite.Domain
QuickBite.Application
QuickBite.Tests
```

Each layer is independent and loosely coupled.

---

10. Conclusion

The QuickBite system successfully implements a scalable and testable order processing platform. The project demonstrates professional software engineering practices including clean architecture, async programming, unit testing, and error handling.