

---

# Order Reconciliation System Architecture Overview

+

•

○

Operational considerations, technological limits, and functional workflows

# Solution Map (Projects & Boundaries)

- • API & Edge: OrderProcessingSystem.API
- • Auth/Security: OrderProcessingSystem.Authentication
- • Domain/Core: OrderProcessingSystem.Core
- • Contracts/DTOs: OrderProcessingSystem.Contracts
- • Data/Persistence: OrderProcessingSystem.Data, OrderDatabase
- • Caching: OrderProcessingSystem.Cache
- • Events/Integration: OrderProcessingSystem.Events
- • Infra/Cross-cutting: OrderProcessingSystem.Infrastructure, OrderProcessingSystem.Utilities
- • UI: OrderProcessingSystem.UI
- • Quality: OrderProcessingSystem.Tests
- • Storage Samples: BlobStorageSimulation/\_Sample
- • Dev Convenience: Scripts-MacOS, Scripts-Windows

# End-to-End Flow



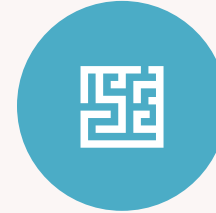
- ORDER CAPTURE (UI → API): CUSTOMER SUBMITS AN ORDER



- VALIDATION & MAPPING: API VALIDATES AND MAPS DTOS TO DOMAIN



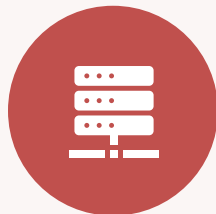
- BUSINESS RULES: TOTALS, INVENTORY/ELIGIBILITY IN CORE



- PERSISTENCE: TRANSACTIONAL COMMIT (ORDER, ORDERITEMS)



- CACHE WARM-UP: PRECOMPUTE HOT READ VIEWS




- INTEGRATION EVENTS: PUBLISH ORDERCREATED, ETC.



- USER FEEDBACK: UI SEES STATUS TRANSITIONS

# API Layer (Edge)

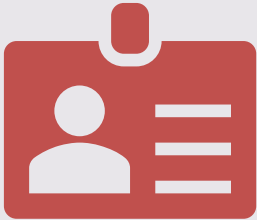
- Responsibilities: input validation, versioned endpoints, problem details
- 

- Orchestration: invoke domain services, repositories, cache
- 

- Security hooks: authentication/authorization middleware
- 

- Observability: consistent logs and correlation IDs

# Authentication & Authorization



- Auth module encapsulates token validation/issuance



- Role/Policy checks at controllers/endpoints



- Secrets/keys via configuration; domain stays pure

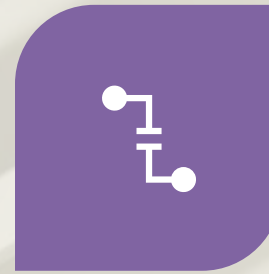
# Data & Persistence



- REPOSITORY ABSTRACTIONS + DB CONTEXT CONFIGURATION



- ORDERDATABASE SCRIPTS FOR SCHEMA AND LOCAL BOOTSTRAPPING

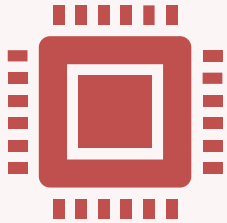


- AGGREGATE-ORIENTED UPDATES AND TRANSACTION BOUNDARIES

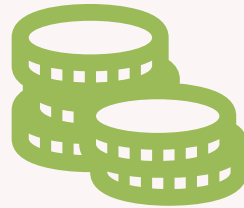


- MIGRATIONS READINESS FOR ITERATIVE SCHEMA EVOLUTION

# Caching Strategy



- ICache abstractions allowing in-memory/Redis plug-in



- Use cases: order summaries, idempotency tokens, lookups



- Consistency: write-through + targeted invalidation on events

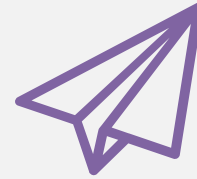
# Events & Integration (Async)



- EVENT CONTRACTS: ORDERCREATED, PAYMENTCAPTURED, ORDERFULFILLED



- DECOUPLING: RETRIES, AUDITABILITY, EVENTUAL CONSISTENCY



- OUTBOX-FRIENDLY: PUBLISH AFTER SUCCESSFUL COMMIT



# Background Processing

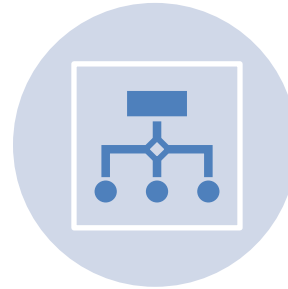
---

- IHostedService-style workers for outbox relays and retries
- 
- Blob workflows via BlobStorageSimulation (e.g., invoices)
- 
- Reliability: exponential backoff, poison-message parking, idempotent handlers

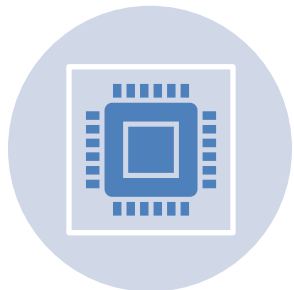
# UI, Testing & Dev Experience



- UI consumes API for capture and tracking views



- Tests: unit + integration for domain and API contracts



- Dev scripts: macOS/Windows one-liners for setup/run



- Next steps: real message bus, distributed cache, CI/CD, full observability