Transactional Outbox Documentation

This service implements the Transactional Outbox pattern to guarantee reliable event publication while preserving atomicity between domain state changes and integration events. The design ensures at-least-once delivery with controlled retries and exponential backoff.

Storage Model

- Entity: Ticketing.Domain.Entities.OutboxEvent
- Fields:
  - EventType
  - EventData (JSON payload)
  - CreatedAt
  - IsProcessed
  - ProcessedAt
  - RetryCount
  - MaxRetries (default: 3)
  - NextRetryAt
  - LastError
- Persistence:
  - Exposed via DbSet<OutboxEvent> in TicketingDbContext
  - Table created and versioned through EF Core migrations

Write Path (Atomic with Business Transactions)

- Command handlers (e.g., ReservationCommandHandlers.CreateAsync) follow this flow:
  1. Apply domain changes (create/update aggregates).
  2. Call IOutboxService.SaveEventAsync(eventType, data) to enqueue an outbox event.
  3. Call SaveChangesAsync() once on the DbContext.
- This guarantees that domain changes and the outbox event are committed in the same transaction, eliminating partial failures

Dispatch Path (Background Publisher)

- Scheduler: Quartz job OutboxPublisherJob, triggered every 30 seconds.
- Flow:
  1. Load retryable events via IOutboxService.GetRetryableEventsAsync().
  2. Publish events using IEventPublisher.PublishAsync() (RabbitMQ implementation).
  3. On success:
     - Mark event as processed (IsProcessed = true)
     - Set ProcessedAt = UtcNow
  4. On failure:
     - Increment RetryCount
     - Record LastError

        - Schedule the next retry (NextRetryAt)

Retry Strategy & Backoff

Retry Selection Criteria

GetRetryableEventsAsync() returns events that:

- Are not processed
- Have RetryCount < MaxRetries
- Have NextRetryAt null or due (<= UtcNow)

Exponential Backoff

- On each publish failure:
    - RetryCount += 1
    - NextRetryAt = UtcNow + 2^RetryCount minutes
    - LastError captured for diagnostics

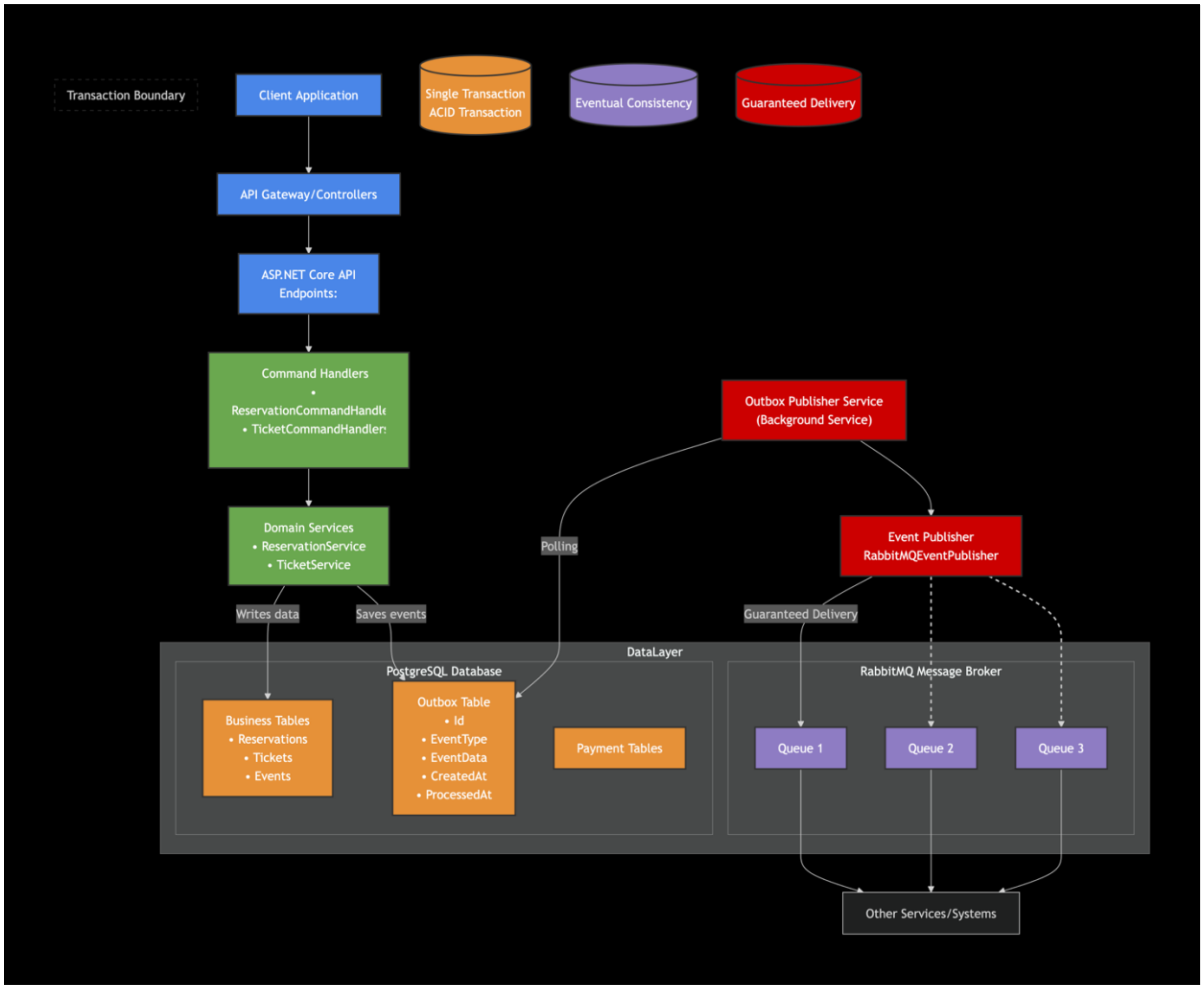
Stop Condition

- Events with RetryCount >= MaxRetries are no longer selected.
- Failed events remain in the outbox table for manual inspection or recovery.

Delivery Semantics

- The system provides at-least-once delivery from the outbox.
- Idempotency is expected downstream (publisher/consumers) to safely handle duplicate deliveries.

Outbox implementation diagram



Transaction Boundary
Client Application
Single Transaction ACID Transaction
Eventual Consistency
Guaranteed Delivery
API Gateway/Controllers
ASP.NET Core API Endpoints:
Command Handlers • ReservationCommandHandle • TicketCommandHandlers
Domain Services • ReservationService • TicketService
Outbox Publisher Service (Background Service)
Event Publisher RabbitMQEventPublisher
Polling
Writes data
Saves events
Guaranteed Delivery
DataLayer
PostgreSQL Database
Business Tables • Reservations • Tickets • Events
Outbox Table • Id • EventType • EventData • CreatedAt • ProcessedAt
Payment Tables
RabbitMQ Message Broker
Queue 1
Queue 2
Queue 3
Other Services/Systems