Outbox Documentation

Transactional Outbox Implementation

The Transactional Outbox Pattern ensures reliable publication of domain events in the FarmersHaulShare modular monolith. When a business action occurs (e.g., a farmer posts a batch), domain events (e.g., BatchPosted) are saved to an **OutboxMessages** table in the **same database transaction** as the aggregate changes. This guarantees **no events are lost** if the app crashes before publishing.

A background worker polls the Outbox and publishes events to RabbitMQ, allowing eventual consistency across modules (e.g., batch posting triggers pricing, haul scheduling, and notifications).

Key Components

1. **OutboxMessage Entity** (in SharedKernel):
   - ➢ Stores Id, EventType (assembly-qualified for deserialization), Payload (JSON), OccurredOn, ProcessedOn, Status (Pending/Published/Failed), RetryCount.
   - ➢ Uses a constructor that automatically includes assembly name for reliable deserialization.
2. **Transactional Saving** (in DbContext, e.g., BatchPostingAndGroupingDbContext.cs):
   - ➢ Override SaveChangesAsync to intercept entities implementing IHaveDomainEvents.
   - ➢ Collects raised events and adds them to OutboxMessages in the same transaction.
   - ➢ Clears events from aggregates after saving.
3. **Publisher** (OutboxPublisherJob.cs in CompositionRoot):
   - ➢ Quartz.NET job runs every 10 seconds.
   - ➢ Polls pending/failed events (RetryCount < 5).
   - ➢ Deserializes payload using assembly-qualified type.
   - ➢ Publishes via MassTransit to RabbitMQ.
   - ➢ Marks as Published on success, or increments RetryCount on failure.
   - ➢ Saves changes transactionally.
4. **Configuration** (in Program.cs):
   - ➢ Registers Quartz job and trigger.
   - ➢ Adds MassTransit with RabbitMQ host.
   - ➢ DbContext registered with Postgres connection.

How Retries Are Handled

- ➢ If RabbitMQ is down: Job catches exception, marks event as Failed, increments RetryCount.

- Retries up to 5 times (configurable).
- Events stay in Outbox until published — no loss.
- After max retries, events remain Failed (manual intervention possible via DB).

Reliability Benefits

- **Atomicity**: Events saved with data — crash-proof.
- **No loss**: Events stay in DB until published.
- **Eventual consistency**: Modules react asynchronously.
- **Prepares for microservices**: Per-module Outbox.

This implementation ensures FarmersHaulShare's core events (e.g., BatchPosted → HaulShareCreated → PriceCalculated) are reliable, supporting fair pricing, grouping, and notifications without data loss.