

StoreFlow — A1.4 Shipping Engine & Fulfilment Workflows (Part 3)

16. Real-Time Event Hierarchy

Real-time behavior in StoreFlow is driven by a small, well-defined set of events.

Core domain events:

- OrderCreated
- OrderStatusUpdated
- ShippingStatusUpdated
- LoyaltyPointsUpdated (future)
- NotificationCreated (future general-purpose)

Characteristics:

- Events are domain-focused, not UI-focused.
- Contain minimal, structured payloads.
- Broadcast over two primary channel types:
 - store.{store_id}.*
 - customer.{public_id}.*

17. Event → Listener → Broadcast Pipeline

Laravel-style event pipeline:

1) Domain-level action occurs (e.g., `OrderService::createOrder`).

2) Domain event dispatched:

```
event(new OrderCreated($order));
```

3) Event listeners:

- Persist audit logs if needed.
- Prepare broadcastable payloads.

4) Broadcast layer:

- Implements `ShouldBroadcast`.
- Channel name is derived from `order.store_id` or `order.public_id`.
- Broadcast driver sends message to WebSocket server.

5) Frontend clients:

- Vue components subscribe to channels using Laravel Echo.
- When event received, Vuex/Pinia state or local component state is updated.

Benefits:

- Clear separation between domain logic and real-time delivery.
- Easy to add new consumers (e.g., mobile app) without changing domain logic.

18. Store Channels vs Customer Channels

Two primary channel namespaces:

Store Channels:

- `store.{store_id}.orders`
 - Receives:
 - OrderCreated
 - OrderStatusUpdated
 - ShippingStatusUpdated (if relevant to store staff)
- `store.{store_id}.notifications`
 - Receives:
 - Important internal alerts

Consumer:

- Dashboard (Operations & Management views).

Customer Channels:

- `customer.{public_id}.order`
 - Receives:
 - OrderStatusUpdated
 - ShippingStatusUpdated
- `customer.{public_id}.loyalty` (future)
 - Receives:
 - Loyalty point adjustments

Security:

- Store channels require authenticated merchant/staff user with store access.
- Customer channels use `public_id` which is a secure, non-guessable token.
- No internal numeric IDs exposed on customer channels.

19. Real-Time Sequence – Example: Order Lifecycle

TEXT SEQUENCE (STORE CHANNEL):

Customer → Storefront: Places order

→ Backend: Create order (pending)

→ Event: OrderCreated

→ Listener: Broadcast to store.{store_id}.orders

→ Dashboard: Operations panel renders new order card

Staff → Dashboard: Accepts order

→ Backend: Status = accepted

→ Event: OrderStatusUpdated

→ Listener: Broadcast to store.{store_id}.orders

→ Dashboard: Order card moves to 'Accepted' column

Staff → Dashboard: Marks Ready / Shipped

→ Backend: Status updated accordingly

→ Event: OrderStatusUpdated (+ ShippingStatusUpdated for shipping)

→ Listener: Broadcast to both store.{store_id}.orders and customer.{public_id}.order

→ Dashboard & storefront UIs update in real time.

20. Error Handling in Real-Time Layer

Possible real-time layer failures:

1) WebSocket server down:

- Events still dispatched and persisted if audit logging is enabled.
- Frontend gracefully falls back to manual refresh/polling.
- Operations panel will still work with reloads.

2) Broadcast exceptions:

- Exceptions during broadcast are logged.
- Do not roll back the underlying business transaction.
- Real-time failures must never prevent order creation or status updates.

3) Client-side issues:

- If Echo or WS connection fails, UI indicates connection error.
- Users can manually refresh to sync state.

Design Principle:

- Real-time is an enhancement, not a hard dependency for correctness.

21. Scaling the WebSocket Infrastructure

Scaling strategy:

Single Node (MVP):

- Laravel WebSockets running on same server as main app.
- Used for dev and early-stage production.

Multi-Node:

- Dedicated WebSocket pods/services behind load balancer.
- Redis pub/sub used so that events from any app node reach all WS nodes.
- Stateless WebSocket servers: no local session storage; rely on Redis.

Authentication:

- Private channel auth uses signed HTTP calls from frontend.
- Backend verifies user access to store or order before granting subscription.

Monitoring:

- Track:
 - active connections
 - messages per minute
 - dropped connections
- Setup alerts if WebSocket layer is down or heavily degraded.

22. Event Rate Limiting & Spam Protection

Risk:

- Misbehaving clients or bugs could produce excessive event dispatches (e.g., rapidly toggling statuses).

Mitigations:

- Domain-level throttling:
 - Prevent extremely rapid status changes on the same order.
- Event batching (future):
 - For non-critical updates, bundle multiple changes into a single event.
- Per-user rate limiting on key endpoints:
 - e.g., limit status change actions per minute.

Internal Safety:

- Ensure broadcast calls are idempotent where possible.
- Events should be small and cheap to serialize/deserialize.

23. Developer Experience & Extensibility

DX Considerations:

- Clear naming conventions for events and channels.
- Shared TypeScript or PHP interfaces for payloads (if used).
- Centralized mapping of:
 - domain events
 - broadcast events
 - subscribed channels

Extension Points:

- Third-party integrations can listen on the same event stream.
- Future admin UI may visualize event flows and real-time state.

Testing:

- Unit tests for:
 - domain event dispatch
 - policy checks for channel auth
- Integration tests using Laravel's broadcasting testing helpers.