

StoreFlow — A1.1 Architecture & System Design (Part 3)

18. Event Lifecycle Architecture

Events drive real-time functionality.

Core domain events:

- OrderCreated
- OrderStatusUpdated
- ShippingStatusUpdated
- ProductUpdated
- LoyaltyPointsUpdated

Event flow:

1. Domain action triggers event.
2. Event dispatched to queue.
3. Listener broadcasts via WebSockets.
4. Dashboard/storefront clients update view.

Consistency:

- Events must not mutate data.
- All mutations occur before event dispatch.

19. WebSocket Real-Time Workflows

Channels:

`store.{id}.orders`

`merchant.{id}.notifications`

Workflows:

Order Placed → broadcast to store channel → dashboard updates instantly.

Order Status Change → partial update broadcast → detail panel updates.

Client behavior:

- Reconnect logic with exponential backoff.
- Offline detection.
- Optimistic UI optional for staff actions.

20. Data Integrity & Consistency Model

Approach:

- Strong consistency for write operations.
- Eventual consistency for UI updates.
- DB transactions wrap core writes.
- Prevent cross-store mutations with enforced scopes.

Anti-corruption layer:

- Repositories ensure only valid domain inputs allowed.
- Validation + sanitization at service boundaries.

21. Database Normalization Strategy

Normalization:

- Fully normalized schema (3NF).
- Avoids repeated data for multi-store.

Exceptions:

- Denormalize shipping rate lookups for performance.
- Cache shipping zones in Redis per store.

Constraints:

- FK constraints always enabled.
- ON DELETE CASCADE for dependent rows (product options, etc.).

22. Error Handling Architecture

Unified error handling pipeline:

- Laravel exception handler maps domain errors to correct HTTP codes.
- Dashboard errors handled via Inertia error pages.
- Storefront gets clean JSON for checkout/shipping.

Error categories:

- Tenant security violations
- Validation errors
- Payment errors (future)
- Fulfilment logic errors

23. Request Pipeline for Multi-Store Operations

Pipeline:

1. Auth middleware loads user.
2. Tenant middleware loads merchant + store.
3. Policy middleware checks permissions.
4. Controller delegates to service.
5. Service performs mutations + events.
6. Response returned with Inertia/JSON.

Guarantees:

- Every request bound to one store.
- No cross-store access allowed.

24. Asset Pipeline & Caching

Assets:

- Vite handles bundling.
- Cache-busted URLs on production.

Caching Layers:

- Product listings
- Storefront configs
- Shipping zones
- Frequently accessed customer profiles

Expiry:

- 5–30 minutes depending on domain.

25. Observability Model

Metrics:

- Queue length
- Websocket uptime
- Order rates per hour
- Shipping quote latency

Logs:

- Structured JSON for ingestion
- Tagged logs by merchant_id + store_id

Tracing:

- Optional OpenTelemetry integration

26. Horizontal Scaling Architecture

Scaling App Layer:

- Stateless PHP containers
- Session stored in Redis
- Horizontally scalable using load balancers

Scaling WebSockets:

- Multiple websocket servers behind LB
- Redis pub/sub for distributing messages

Scaling Queue Workers:

- Worker pools scaled independently
- Horizon recommended

27. Load Balancing & Failover

Load balancing:

- Nginx/ELB distributes traffic across app nodes
- Sticky sessions NOT required with Redis session store

Failover:

- Standby DB replica
- TTL-based failover routing
- Queue failover to backup Redis cluster

28. Security Hardening

Security layers:

- Strict CORS
- Content Security Policy
- HTTPS enforcement
- Session hijack prevention
- Rate limiting
- Brute-force login logout
- Periodic password rotation for staff (optional)

29. Future Architecture Extensions

Planned enhancements:

- Plugin system for merchants
- Machine-learning product recommendation engine
- Advanced analytics dashboard
- Automated fraud detection for orders
- Multi-language storefronts
- Global CDN caching for product images