Concept Refactoring: Multi-Entity Evolution

Context

Our domain is **discovery-only**—the app makes items visible and helps people contact each other; all exchanges happen offline. Modeling choices below tighten boundaries, add invariants, and reduce ambiguity.

R1) Listing → Piece (Aggregate Root)

Original (M1)

`Listing` mixed the physical garment and its on-platform post.

Refactored (M2)

`Piece` becomes the **aggregate root** that governs lifecycle (`Active ⇄ Reserved → Closed`), reviews, condition rating, images, and reservation. A "listing" is just the visibility event/view (e.g., `ListingPublished`). All mutations go through the root.

Motivation (insight)

Observed events are about a **garment entering/leaving visibility** and the exchange concluding offline—so the stable identity is the garment, not the post.

Key invariants

≥1 image at publish; reviews belong to exactly one Piece; legal transitions only; edits blocked when Closed.

Impact

Clearer lifecycle, stronger consistency, future-proofing for re-posting/history.

R2) Reviews Ownership: Profile view ← Piece source

Original (M1)

Reviews were conceptually tied to profiles/listings interchangeably.

Refactored (M2)

`Review` is a **child of Piece**; profiles show **read-only projections** (avg + count). Profile owners cannot mutate reviews.

Motivation

Trust artifacts arise from specific item exchanges; attaching them to the item prevents cross-linking or self-editing.

Impact

Integrity of reputation; simpler aggregation; fewer ambiguity bugs.

R3) Account/Auth vs User Profile

Original (M1)

“User” combined auth identity with public profile.

Refactored (M2)

Keep auth in the platform layer; `UserProfile` becomes a separate aggregate for editable public info (name, picture, location, bio), saved listings, and listing-scoped message threads.

Motivation

Profiles change frequently and are user-owned; reviews/ratings remain external projections (see R2).

Impact

Cleaner permissions, clearer write boundaries, easier caching.

R4) Interest & Messaging as Listing-Scoped Threads

Original (M1)

“Contact seller” was a generic action with no stable conceptual anchor.

Refactored (M2)

Define **Interest** (the first contact or save) and bind **MessageThread** to the listing/piece context; notifications and messages are grouped **by listing**.

Motivation

Your domain events already distinguish **Interest Expressed**; grouping threads by listing mirrors real conversations and reduces orphaned chats.

Impact

Traceable contact flows; simpler moderation; better UX.

R5) Status Machine Made Explicit

Original (M1)

Status semantics were implied, edits sometimes allowed after closure.

Refactored (M2)

Enforce `Active ⇄ Reserved → Closed` (no reopen). Close disables edits. Reserve only from Active;

Close only from Active/Reserved.

Motivation

Matches offline hand-off and prevents “zombie” edits on closed items.

Impact

Consistent UI states; fewer race conditions; cleaner audit trail.

R6) Category/Type as Validated Attribute (not a mini-taxonomy inside the item)

Original (M1)

Ad-hoc category choices risked misclassification.

Refactored (M2)

`categorize(Piece, Type)` validates against allowed types; does not affect visibility; remains an attribute under the Piece root.

Motivation

Better search/filter quality without coupling to listing persistence.

Impact

Higher precision for discovery; simpler index strategies.

R7) “Transaction” removed (Discovery-Only Clarification)

Original (M1)

Early vocabulary hinted at in-app transactions.

Refactored (M2)

Interfaces and flows explicitly avoid payments; the app facilitates visibility, search/filter, save, and contact only; exchange and payment occur **outside** the platform.

Motivation

Aligns model with real behavior and reduces scope/risks.

Impact

Leaner model; fewer legal/security constraints; clearer acceptance tests.

R8) Function Signatures aligned to events/guards

Original (M1)

Functions loosely implied effects.

Refactored (M2)

Signatures encode inputs, outputs, pre/postconditions, and guard the root (e.g., `publishListing(Piece,Seller,Locale) → ListingPublished`, `expressInterest(Listing,Buyer) → InterestExpressed`, `closeListing(Listing,Seller) → ListingClosed`).

Motivation

Make invariants enforceable and testable.

Impact

Deterministic behaviors; acceptance tests map 1-to-1 with domain events.

Measurable Improvements

- **Model clarity:** single write point per aggregate (Piece/Profile); reviews provenance unambiguous.
- **Conceptual separation:** UI artifacts (cards/listings) vs domain entities (Piece) vs projections (profile ratings).
- **Testability:** preconditions/postconditions codified in signatures and status machine.

Reflection

These refactorings aren't cosmetic. They reflect deeper understanding of how discovery, contact, and offline exchange actually work in our setting, and they encode that understanding with DDD boundaries plus simple, enforceable invariants.