Supple Design Refactor for the Pieces Feature

Overview

This document provides a full explanation of the Supple Design improvements applied to the **Pieces** feature of the Hand-me-down Clothing project. The goal of these refactors was to:

- Increase maintainability and flexibility of the code
- Reduce duplication
- Improve intent-revealing design
- Decouple domain logic from infrastructure
- Ensure reliable parsing, formatting, and filtering of enums and attributes
- Strengthen documentation for future contributors

These changes support easier future extension, safer modifications, and clearer onboarding for new developers.

---

1. Refactored Architecture

1.1 Improved Enum Parsing via `parseEnum` Helper

Previously, the `PieceFactory` contained five nearly identical parsing methods for Category, Gender, Size, Condition, and Status. This violated supple design principles by hiding domain intent behind repetition.

New Design

We introduced a single **generic enum parser**: - Eliminates duplication - Supports both numeric and string inputs - Gracefully handles numeric strings (e.g., "2") - Throws descriptive validation errors - Makes each parseX method a thin, intention-revealing wrapper

This improves maintainability and reduces future errors.

1.2 Consistent DTO Construction with Optional ID Removal

The factory `toDTO()` method still includes `id`, ensuring full domain → DTO mapping for updates or

migrations.

However, during **piece creation**, `id` must be excluded so Supabase can auto-generate it. Now the repository removes `dto.id` specifically in the `createPiece` method.

This achieves: - Consistency (factory always outputs full DTO) - Safety (repository removes ID only when necessary) - Clarity (responsibility boundaries respected)

1.3 Enhanced Validation Logic

The previous validation incorrectly checked for `length < 0` and lacked meaningful error messages.

The updated validator:

- Verifies non-empty name, brand, and user_id
- Ensures positive ID for updates
- Throws a consolidated, descriptive error listing missing fields

This results in predictable behavior and clearer debugging.

1.4 Repository Now Uses Dependency Injection

Instead of hard-coding:

```
private supabase = createClient();
```

The repository now receives a Supabase client via constructor injection.

Benefits:

- Fully testable repository with mock clients
- Lower coupling to infrastructure
- More consistent with clean architecture

1.5 Unified Filtering System (Including Status)

The previous filtering code: - Contained a typo (`" category"`) - Treated `status` differently from other enums - Contained repetitive logic

New filtering system:

- Adds `status` to the `searchFields` list
- Makes all filterable fields behave consistently
- Uses `ilike()` for partial, case-insensitive enum matching
- Removes special branching for status
- Clean, predictable behavior for all filters

This creates a future-proof design where adding a new searchable field only requires updating one list.

1.6 Improved Formatting for All Enums

The previous formatting methods returned `.toString()`, which was not suitable for UI representation.

New implementation includes:

- Enum → string lookup
- A reusable `pretty()` helper
- Automatic conversion from ENUM_STYLE to "Enum Style"

This greatly improves UI readability while keeping formatting logic in the domain layer.

---

2. Updated Domain Models

Piece

The Piece model now: - Includes improved formatting helpers - Uses enum lookups for human-readable formatting - Supports future extensions thanks to consistent design

SoldPiece / DonatedPiece

Continue to extend Piece without modification. Refactors ensure future extensions remain decoupled.

---

3. Updated Repository Responsibilities

3.1 `createPiece()`

Now safely removes IDs before insertion and guarantees proper enum formatting.

3.2 `filterPieces()`

Unified, cleaner filtering logic ensuring: - less coupling - fewer branching errors - better extensibility

3.3 `deletePiece()`

Now provides robust success/failure reporting.

---

4. How This Supports Supple Design

The refactors align with core Supple Design principles:

Intention-Revealing Interfaces

Simplified parsing and filtering interfaces communicate intent clearly.

Low Coupling

Supabase dependency injection dramatically reduces coupling.

High Cohesion

Each class now handles its domain-specific responsibilities cleanly.

Declarative Design

Filtering rules are now declared in a centralized list, not embedded in imperative branches.

Side-Effect-Free Behavior

Enum parsing and formatting are deterministic and well-scoped.

Meaningful Names

Methods like `parseEnum`, `pretty`, and unified filtering logic make the code expressive.

5. Recommendations for Future Work

- Add a `FilterSpecification` class to avoid further logic accumulation in the repository
- Introduce unit tests for all new parsing, formatting, and filtering logic
- Extend supple design patterns into Seller, Buyer, and Review models
- Add optional caching for expensive filtering operations

6. Conclusion

These changes significantly improve the clarity, safety, and adaptability of the Pieces feature. By embracing supple design principles, the system is now more flexible for new features, easier for contributors to understand, and more robust against edge cases.

The new structure positions the project for long-term maintainability and a more scalable future.