Factory Layer Documentation

Overview

The factory layer is designed to handle the creation and transformation of domain entities. In this project, it ensures consistent logic when generating objects such as `SoldPiece` or `DonatedPiece` from raw database records and vice versa.

Factories centralize instantiation logic, allowing the system to: - **Maintain consistency** when constructing domain objects. - **Encapsulate creation rules**, reducing duplication across the codebase. - **Isolate transformation logic** between the data and domain layers.

Piece Factory → PieceFactory

Purpose

The `PieceFactory` class is responsible for constructing `Piece` domain objects from raw database records and converting `Piece` instances back into plain objects suitable for database operations. It automatically determines whether a record should be treated as a `SoldPiece` or `DonatedPiece` based on its attributes (specifically, the presence of a `price`).

Responsibilities

- Generate `SoldPiece` or `DonatedPiece` instances dynamically.
- Provide a unified interface to handle both donation and sale pieces.
- Transform `Piece` objects into data transfer objects (DTOs) for insertion or updating in Supabase.
- Enforce consistent construction rules across the data and domain layers.

Benefits

- **Abstraction:** The logic for differentiating between `SoldPiece` and `DonatedPiece` is centralized.
- **Consistency:** Guarantees that every `Piece` object is created with valid, structured data.
- **Maintainability:** Simplifies the process of adding new subtypes (e.g., `RentedPiece`) in the future.
- **Reusability:** Both repositories and services can use the same factory without duplicating object creation logic.
- **Error Reduction:** Minimizes incorrect instantiation of objects by ensuring domain integrity.

Methods Summary

Method	Return Type	Description	Example
makePiece(item)	Piece	Creates a `SoldPiece` or `DonatedPiece` based on the provided record	`factory.makePiece(dbRecord)`
toDTO(piece)	Record<string, any>	Converts a `Piece` domain object into a DTO for database storage	`factory.toDTO(piece)`

Justification for Factory Pattern

The factory pattern is used to encapsulate complex object creation logic and promote a clean separation between raw data and domain entities. By centralizing construction, the application gains: - **Simplified creation:** Avoids scattered `new` calls throughout the code. - **Reduced coupling:** Domain logic doesn't depend on how objects are instantiated. - **Better testability:** Creation rules can be tested independently from repositories. - **Consistency across layers:** Ensures that domain models always follow the same initialization logic, regardless of source.

Example Usage

```
const factory = new PieceFactory();

// Convert a database record into a domain object
const piece = factory.makePiece({
  id: "1",
  name: "Blue Shirt",
  category: "SHIRT",
  price: 20,
  condition: "LIKE_NEW",
  images: [],
  user_id: "user123"
});
console.log(piece instanceof SoldPiece); // true

// Convert a domain object back to a DTO for saving
const dto = factory.toDTO(piece);
console.log(dto);
```