

Documentación Social Meli

Hernán Tonutti

Modelos

El modelado de la aplicación se realizó diferenciando dos roles claves que debían interactuar: compradores (*Buyers*) y vendedores (*Sellers*). Estos comparten cierta base como es el ID de usuario y el nombre, por lo que se creó una clase padre abstracta de la cual ambas heredan, llamada Usuario (*User*). Además esta clase padre define un método abstracto para poder obtener el tipo de usuario al cual se está accediendo (con el objetivo de evitar el uso luego de *instanceof* por ser una mala práctica dentro de POO), según lo definido en el enumerador de tipo de usuario (*UserType*).

En cuanto de las diferencia de las clases relacionadas a los roles, los compradores contienen una lista con los IDs de los usuarios a los cuales siguen y métodos para realizar acciones sobre esta: agregar, borrar, obtener cantidad y obtener la lista de IDs. Además se define la sobre-escritura del método que obtiene el tipo de usuario. De forma análoga, cada vendedor contiene una lista con los seguidores, los métodos para accionar sobre esta y la definición del tipo de usuario, pero agrega una lista de IDs de los post que contiene junto a acciones sobre esta.

Por último, dentro de las clases principales del modelo se encuentra también la publicación (*Post*), la cual contiene todos los atributos que caracterizan esta según la consigna, y dos constructores personalizados que permiten un mapeo rápido entre la clase del modelo y los DTO de ingreso que se nombrarán a continuación.

Por cada ingreso o egreso de datos del controlador se realizó un DTO para lograr la mayor independencia de los contratos. Estos se ordenaron en dos carpetas para distinguir cuales se referían a ingreso y cuáles a egreso. Además algunas partes repetitivas (como los detalles de los post y los IDs y nombres de los usuarios) se separaron en otras clases de DTOs internos que permitieron la re-utilización como atributos de otros DTOs.

Repositorio

Toda la información se centralizó en un único repositorio (*SocialRepository*), tanto para las publicaciones como para los usuarios. Se emplearon mapas para cada uno de estos entre los IDs y los objetos de la clase en sí, con el objetivo de mantener la

unicidad de cada ID y detectar accesos a IDs faltantes o posibles duplicaciones. Se destaca también el uso de un vector para todos los usuarios sin diferencia entre compradores y vendedores, lo cual fue permitido gracias a la herencia de la misma clase padre. Se brindan métodos para el acceso a la lista de valores de los mapas, para comprobar la existencia de los IDs de publicaciones y usuarios, y para la obtención, agregado y borrado de estos IDs.

Servicio

Al igual que el repositorio, este es único para todas las funciones (*SocialService*), por cada uno de las historia de usuario existe una función para obtener lo solicitado. Se destacan tres funciones privadas que se crearon para saciar procedimientos repetitivos de las demás. El más importante de estas funciones privadas es el validador de ID de usuario, el cual a través del paso del ID más del rol que se quiere validar (Buyer o Seller) comprueba la existencia y que el usuario contenga el rol buscado, en caso contrario arroja excepciones personalizadas para id no encontrado (*NotFoundException*) y para el rol incorrecto (*WrongTypeException*). Luego en las funciones privadas está la creación de las publicaciones, la cual valida la existencia ya del post arrojando una excepción (*DuplicateException*) en caso afirmativo. Por último, se encuentra la función que permite ordenar usuarios por nombre, pasándole solamente la lista de usuarios a ordenar y el orden que quiere (ascendente o descendente).

En todas las funciones realizadas se aplicaron expresiones lambda en medida de que fueran posibles aplicarlas. En esta capa además se encargó todo el mapeo entre las clases de los modelos y los DTOs.

Controlador

Se realizaron dos controladores diferentes en base a los endpoints que se requerían: uno basado en funcionalidades de los usuario (*UserController*) con el prefijo “/users” para todos sus endpoints, y otro para las publicaciones (*ProductController*) con el prefijo “/products”. Ambos controladores poseen un controlador de excepciones en común, el cual se encarga de devolver los *HTTP status* y mensajes correctos en caso de que alguna excepción suceda.

Casi todos los endpoints devuelven *ResponseEntity* con su *HTTP status* y *body* correspondiente, salvo algunos post que solamente devuelven un *HTTP status* indicando el éxito/fracaso de la operación.

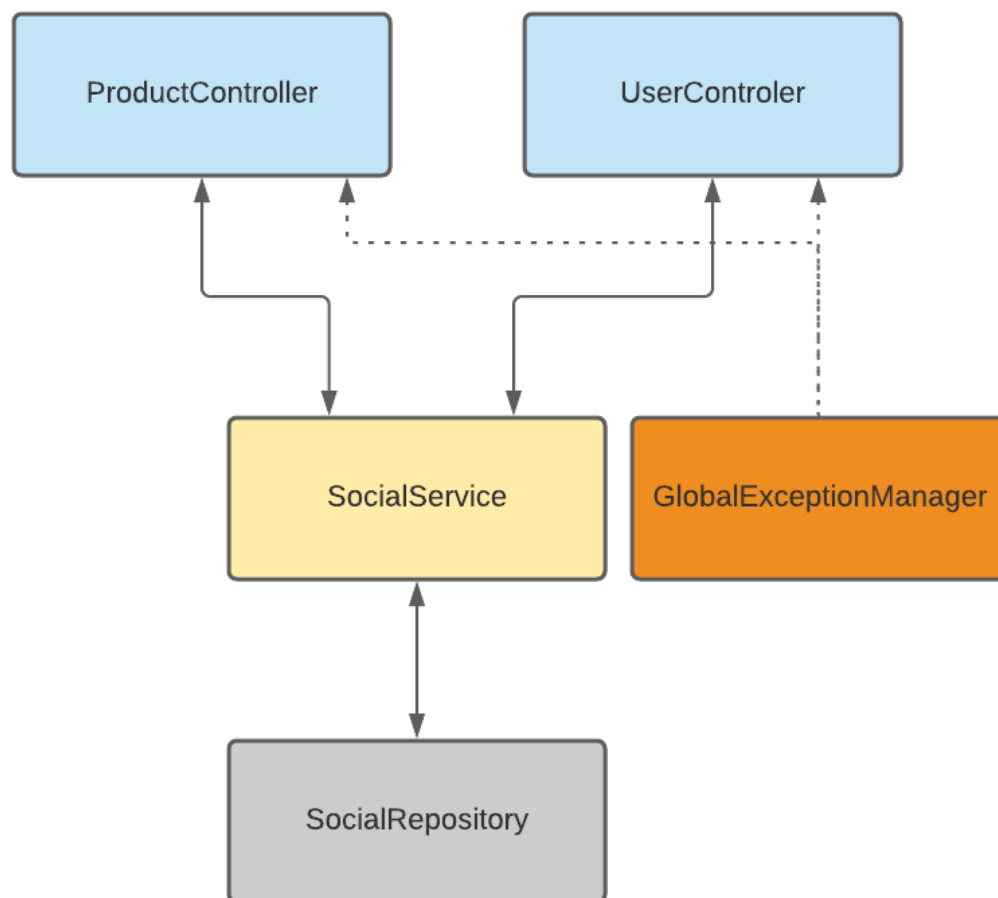
Cuestiones generales

En cada una de las clases creadas se conservó el encapsulamiento de sus atributos, brindando métodos en caso de necesitarse el tomar o definir los valores de alguno en partícula o funcionalidades que los afecten como en el caso de las listas. Además para el servicio y repositorio se implementaron interfaces y la inyección de dependencias correspondientes a estas.

Al centralizarse el guardado de los datos y también conservar la consistencia en cuanto a seguidores, seguidos y publicaciones de los vendedores, las operaciones que impliquen altas o bajas (las modificaciones solo afectarían si cambiaría el ID de alguno, para lo cuál no hay funcionalidades existentes) se realizaron en bloques transaccionales conteniendo el modificación de las listas de IDs en compradores o vendedores y los cambios en los mapas del repositorio.

Diagrama

A modo de resumen general se puede observar en el siguiente diagrama la estructura de las capas:



También se adjunta un archivo anexo en la misma carpeta de este documento junto a un diagrama especificando el modelado y los DTOs implementados junto a sus relaciones.