TEST SYMFONY GRUPO TODO

Reporte técnico

Jorge Oscar Gianotti

Resumen

Se presenta este documento a modo de reporte técnico en el proceso de selección de desarrollador Symfony para la empresa Grupo Todo.

Se explica en detalle cada etapa en el desarrollo de la solución del ejercicio propuesto.

Se fundamentan posibles mejoras sobre la solución implementada.

INTRODUCCIÓN

Para la resolución del ejercicio propuesto se tomó como premisa alterar lo menos posible el desarrollo inicial propuesto adaptando las nuevas funcionalidades al código existente y al esquema de base de datos inicial.

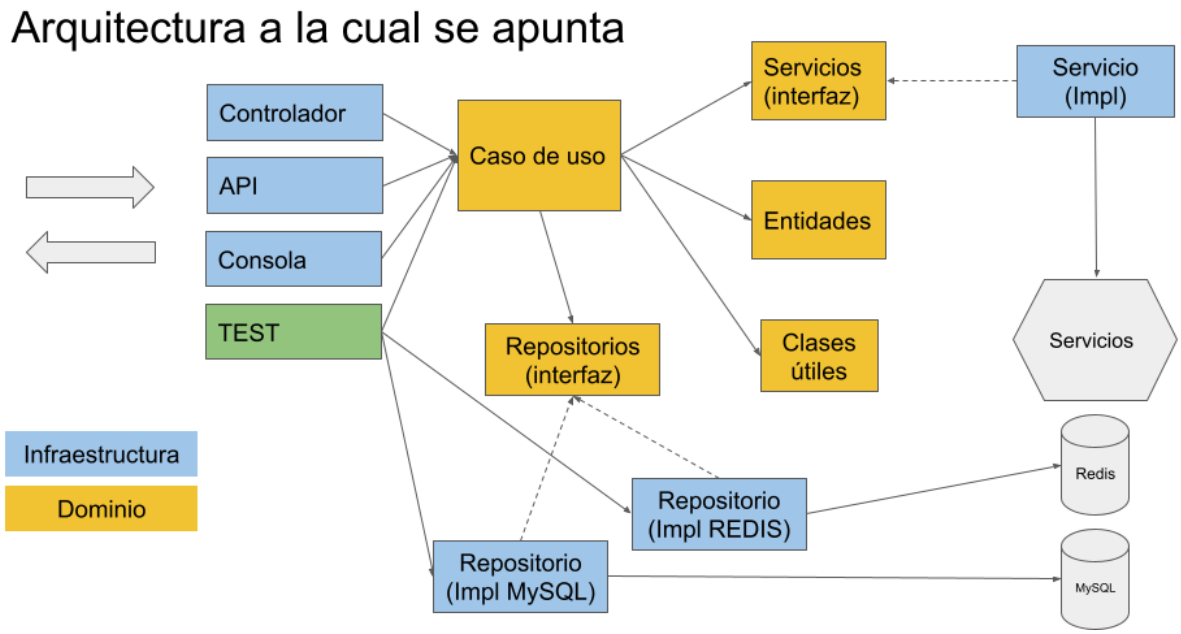
El desarrollo del ejercicio se realizó siguiendo la técnica *Test Driven Development (TDD)* creando un test unitario por caso de uso. Si bien no todos los casos de uso fueron implementados con un test, la mayoría si están cubiertos principalmente los ABM de Producto y Categoría.

Para poder realizar tests unitarios sobre cada caso de uso en la aplicación que fueran útiles y rápidos, se aplicó el concepto de separación de capas tomado de los patrones de

arquitectura en capas. Es así que podemos encontrar una primera capa de controladores cuya función será gestionar los protocolos de comunicación con el exterior. Luego se encuentra una capa de Dominio donde se destaca la creación un clase específica por caso de uso y el uso de interfaces para la comunicación con con todos los servicios externos. Por último tenemos una capa de servicios donde encontramos las clases concretas que implementan las interfaces de los servicios presentados en la capa de dominio.

Con esta división bien marcada cada test apuntará directamente al caso de uso sin pasar por el controlador y probará el comportamiento en relación a los servicios externos sin hacer llamadas reales a esos servicios.

De esta forma tenémos tests unitarios que validan de forma eficiente los casos de uso de nuestra aplicación.



ETAPAS DE IMPLEMENTACIÓN

Cada etapa se corresponde con la implementación de un único requerimiento.

Despliegue de servicios con Docker. La primera tarea realizada fue la incorporación de Docker y Docker Compose como herramientas de despliegue de los servicios utilizados por la aplicación. En este caso se definen dos servicios, un servicio de base de datos MySQL y un servicio para el servidor web Apache que incorpora PHP 7.4. Se incorpora además el uso de la herramienta Make como gestor de tareas que agrupa los comandos utilizados y facilita el despliegue de los servicios.

Aunque no sea una incorporación necesaria en la solución del ejercicio, es interesante como herramienta en el desarrollo dado que permite definir un entorno común para todas las personas que trabajen sobre el proyecto. La definición de este entorno puede cambiar fácilmente acompañando la evolución del proyecto.

Autorización de acceso. El primer requerimiento que presenta el ejercicio consiste en la incorporación del control de acceso solo para un usuario con contraseña. Parte de este requerimiento se encontraba implementado, donde se observa el modelo *User* ya creado y la tabla asociada en la base de datos con la incorporación de los datos de un primer usuario.

Dado que esta es una tarea común y recurrente en la mayoría de los proyectos, se suelen proveer herramientas para la creación automática del código asociado a esta tarea. En este caso se utilizó el comando *make:auth* del paquete make el cual provee la creación automática del código dentro de la estructura del framework Symfony.

ABMs de Productos y Categorías. Se implementó las tareas comunes de alta, baja y modificación. Las clases encargadas de atender estos casos de uso se ubican dentro del directorio *Application*.

Menú de categorías. Para mostrar un menú jerárquico de categorías, a la entidad Categoría se le agregó la posibilidad de registrar una categoría padre, pudiendo formar un árbol de categorías multinivel. Para gestionar ese árbol de categorías se agregó el paquete *BlueM/Tree* que nos permite organizar un conjunto de datos jerárquicos y mostrar esa relación de dependencia fácilmente haciendo uso del campo ***parent*** definido en la clase Categoría.

Breadcrumb mostrando la ruta de categorías de un producto. Para la creación del ***breadcrumb*** se utilizó un enfoque de ***read model***, o sea que el modelo está optimizado para las lecturas. Para crear un *breadcrumb* que muestre la ruta completa de la categoría asociada a un producto es necesario recorrer el árbol de categorías hacia arriba hasta formar la ruta completa. Al ser una estructura multinivel sin un límite esta tarea podría terminar con una gran cantidad de consultas a la base de datos volviéndose un proceso muy lento.

Como en este caso la creación o modificación de categorías será mucho menor que las lecturas que se van a realizar es conveniente guardar el dato ya procesado para que la lectura sea rápida. Es así que cada categoría registra en un campo de texto la ruta completa que se va a mostrar.

Para el proceso de modificación de una categoría se utiliza eventos de dominio haciendo uso del paquete symfony/messenger lanzando un mensaje o evento cuando una categoría es modificada para que se actualicen las rutas generadas tanto de la propia categoría como de todas sus categorías hijas manteniendo la consistencia de esa información. El uso de eventos es una

herramienta útil para desacoplar tareas relacionadas resultando en una base de código fácil de mantener.

POSIBLES MEJORAS

Una primera mejora interesante sería el agregado de tests funcionales que validen una funcionalidad específica completa pasando por el controlador que atiende ese caso de uso y los servicios reales. Este tipo de tests deben existir en poca cantidad y solo para funcionalidades críticas en relación al usuario o cliente.

Se puede mejorar la estructura de directorios para presentar una organización similar a la organización que se presenta en la arquitectura. Así se pueden encontrar los directorios de ***Dominio*** y ***Servicios***, con subdirectorios por recurso, donde se agrupan todas las clases asociadas a cada capa. Por ejemplo: Entidades, Casos de uso y Repositorios (Interfaces) se ubican dentro del directorio de Dominio y Repositorios (Implementación) en el directorio de Servicios.

También es interesante la posibilidad de agregar ***Value Objects*** o sea objetos sin identidad solo modelan un valor y su comportamiento asociado permitiendo agregar validaciones asociadas a los datos en un mismo objeto. En este desarrollo claramente se podría haber aplicado al dato que almacena la ruta completa de una categoría. Así en el momento de la creación de ese objeto se aplica la validación del formato que debe tener ese string mostrando los nombres de las categorías separados por una barra inclinada.

Por último también es interesante poder agregar identificadores únicos para las entidades que no fueran autogenerados por la base de datos. Se podría haber usado el formato de identificador **UUID** que siempre nos dará un identificador global único. La finalidad de esto es tener siempre en todo momento una instancia válida del objeto. Desde el momento de la creación no es

necesario esperar un identificador devuelto por la base de datos porque ya lo tendremos junto al resto de los datos. Esto es especialmente útil en sistemas que tienen desarrollos separados para *Frontend* y *Backend*.

Un extra necesario es mejorar la automatización del despliegue del entorno de desarrollo con Docker sobre todo en la creación de las bases de datos. Esto se podría hacer fácilmente con la técnica de ***docker in docker*** donde hay un contenedor que coordina las tareas sobre los demás contenedores.