Web SPA de Subastas

Práctica de Programación Avanzada

3º Curso – Ingeniería del Software

Curso 2018-2019

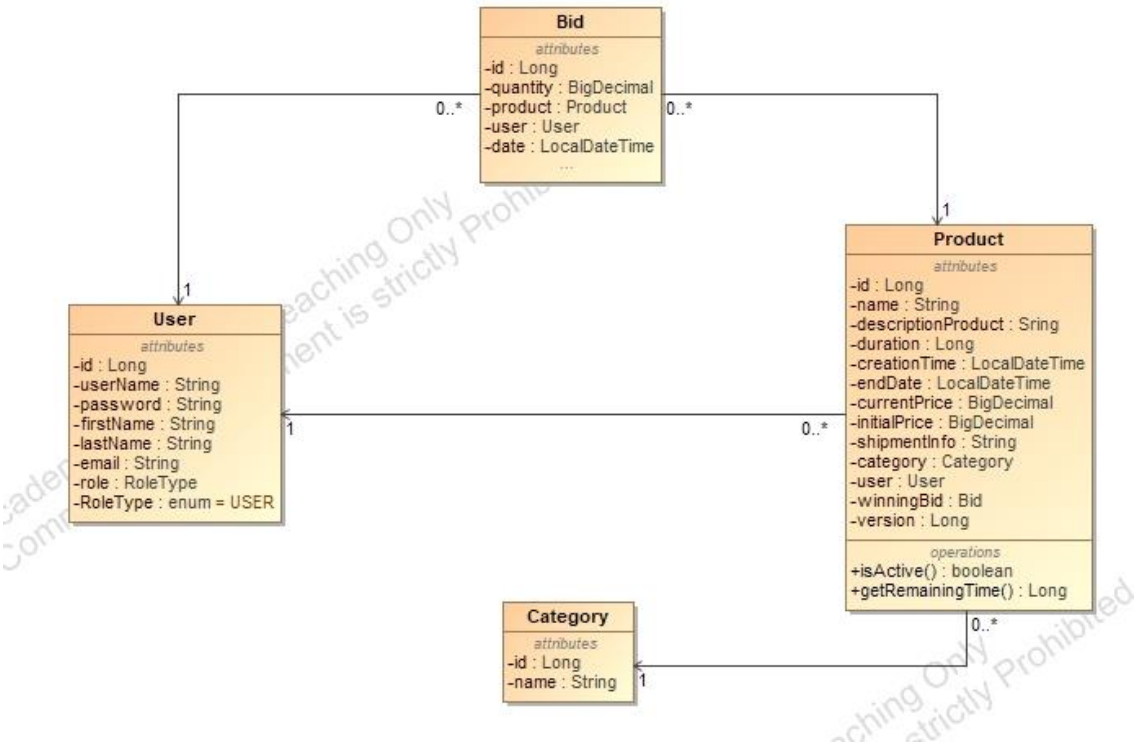
Nombre	**Login**
Regueiro Teijido, Pablo	pablo.regueiro.teijido
Álvarez Gurdiel, Julia	julia.gurdiel
Filgueiras Rilo, Juan	juan.filgueiras.rilo

ÍNDICE

1. BACKEND

1.1. CAPA DE ACCESO A DATOS

Para cumplir con el acometido de la práctica, se han creado distintas entidades con sus correspondientes atributos, tal y como se muestran en el siguiente diagrama:



Podemos diferencia cuatro entidades distintas:

- **User:** Comprende los datos relacionados con el perfil de usuario que interactúa con la aplicación web. Contiene un identificador propio para cada usuario, así como el nombre completo, un correo identificativo y una contraseña asociada. A su vez, un usuario puede tener diferentes roles, dependiendo de la finalidad con la que utilice la aplicación, es decir, ya sea para participar en subastas para la adquisición de productos, o para promocionar productos para su venta.
  Como se puede comprobar en el diagrama, un usuario puede realizar múltiples pujas, y también puede promocionar múltiples productos.

- **Bid:** Contiene todos los atributos relacionados a las pujas que se pueden realizar sobre un producto de un cliente. Entre ellos podemos diferenciar un identificador de la puja, así como el producto sobre el cual se realiza la misma, el usuario que la realiza y la fecha en la que se produce.
  Además, en el diagrama se visualiza que un producto puede contener diferentes pujas.

- **Category:** Esta entidad contiene un identificador para cada una de las categorías de los distintos productos que se encuentran en la aplicación, así como el nombre de cada una de ellas.
  Una categoría puede tener múltiples productos, pero un producto sólo puede pertenecer a una categoría.

- **Product:** Es la entidad principal de la aplicación, ya que la finalidad de la misma consiste en la venta de productos a través de subastas de los distintos usuarios que pertenecen a la misma. Contiene un identificador para el producto, un nombre para el mismo con la descripción de sus funcionalidades, y con respecto a las pujas que se realicen sobre él, tiene una fecha de inicio de la subasta y una fecha de fin, un precio de salida y el precio en el que se encuentra actualmente la subasta, y la puja que va ganando actualmente.
  También dispone de un atributo que identifica la categoría a la que pertenece, y el usuario propietario de ese producto.

1.2. CAPA LÓGICA DE NEGOCIO

❖ **AuctionService:** Interfaz que contiene las diferentes operaciones que se pueden realizar con respecto a la subasta de un producto.

Por un lado, se pueden crear nuevas pujas a partir del identificador del producto que se desea adquirir. Por otro lado, se puede obtener una lista con todas las pujas realizadas por un usuario concreto, a través del identificador de ese usuario.

❖ **ProductService:** Interfaz que contiene los casos de uso relacionados con los productos de los usuarios.

A partir de este servicio se puede añadir un nuevo producto a la aplicación, indicando los datos mencionados anteriormente.

También se pueden buscar productos a través de palabras clave del nombre del producto o de la categoría. A su vez, se puede obtener la lista de todas las categorías de los distintos productos que se encuentran almacenados en la aplicación.

- **UserService:** Interfaz que contiene las distintas operaciones que un usuario puede realizar en la aplicación relacionadas con la identificación y realización de operaciones según los privilegios de los que disponga.

  Por ello, mediante este servicio, un usuario podrá crear una nueva cuenta en la aplicación, así como identificarse mediante un nombre de usuario y una contraseña aportadas en la creación del mismo. También es posible realizar modificaciones en el perfil del usuario, aportando el nombre y el correo electrónico con el que se creó la cuenta.

  Por último, para aumentar la seguridad de la aplicación, también es posible realizar cambios en la contraseña.

1.3 CAPA SERVICIOS REST

- **AuctionController:** Mediante la capa de servicios REST, se realiza un enlace interno entre la capa de servicios y la capa de acceso a servicios, enlazando cada una de las interacciones del usuario realizadas en el interfaz de usuario con su correspondiente caso de uso en la lógica de negocio, gracias a estos controladores.

  A través de este controlador será posible tratar los casos de uso relacionados con las pujas que los distintos usuarios del sistema hayan realizado sobre los productos que otros usuarios hayan creado en la misma.

  Por otro lado, también es posible recuperar todas las pujas que ha realizado un mismo usuario.

- **ProductController:** A través de este controlador se tratarán todas las operaciones relacionadas con los productos introducidos en la aplicación.

  Por ello, un usuario puede ver un menú desplegable con todas las categorías de productos creados para poder realizar una búsqueda de los productos en los que está interesado.

  Una vez encontrado el producto deseado, es posible visualizar el contenido en detalle del mismo, con el caso de uso de obtener el detalle de un producto.

  Además de poder visualizar los productos creados por el propio usuario, es posible crear nuevos productos para poder anunciarlos en la aplicación con la finalidad de venderlos mediante una subasta en la que pujan el resto de usuarios.

- **UserController:** El controlador relacionado con las operaciones del usuario será el encargado de realizar las conversiones necesarias de código JSON a Dto y viceversa que permitan completar las operaciones de creación o modificación de los datos asociados a una cuenta de un usuario.

  También es el módulo encargado de generar un token, que es el elemento que permite identificar de manera unívoca a un usuario aportando seguridad a la aplicación, pues el token es un componente que no puede ser simulado por una persona que intente acceder de manera malintencionada suplantando una identidad.

2. FRONTEND

- **Auction:** Este módulo de la capa frontal de la aplicación tiene como acometido encapsular el componente que contiene el botón de crear una puja sobre un producto creado de la aplicación de otro usuario, ya que un usuario no puede realizar pujas sobre un producto creado por él mismo.

  Esta operación es posible realizarla aportando el identificador del usuario que va a realizar la puja, el identificador del producto sobre el que quiere pujar, y el precio que quiere pujar, teniendo en cuenta que podrá visualizar el precio actual de la puja más alta realizada sobre ese producto, y que deberá ser mayor a la misma para que sea considerada como válida por el sistema.

- **Catalog:** Este módulo contiene diferentes componentes de la interfaz de la aplicación que realizan varios de los casos de uso implementados.

  Por un lado, en la cabecera de la interfaz se visualiza el botón “Buscar”, el cual permitirá ejecutar el caso de uso de buscar los productos guardados en el sistema, siendo posible realizar un filtro mediante palabras clave que contenga el producto, o la categoría del mismo.

  Una vez accedido al conjunto de los productos devueltos por la búsqueda, un usuario podrá visualizar el detalle de cada uno de los productos, con lo que se renderiza la página mostrándose el componente del detalle de los productos.

  Por defecto en la aplicación se muestran 10 productos, por lo que en caso de que haya más añadidos en la base de datos, deberán ser mostrados en distintas páginas, por lo que en la parte inferior del cuerpo del sistema se encuentran dos botones (“Anterior” y “Siguiente”), que forman el componente de los resultados de la búsqueda de los productos y estarán ctivados o desactivados para su uso en función de si hay más productos que visualizar o no.

- **User:** Este módulo contiene los diferentes estados de los componentes asociados a las cuentas de los usuarios creados en el sistema.

  Entre ellos, contiene los botones mostrados en el menú desplegable de la parte derecha de la cabecera mediante los cuales es posible actualizar el perfil o cambiar la contraseña de una cuenta.

  También es posible en este módulo crear un nuevo usuario, es decir, registrar un usuario y autenticarse una vez creado.

3. TRABAJOS TUTELADOS

3.1 BACKEND: BATCHSIZE

Tras entender el problema de las n+1 consultas planteado, se observó que este proyecto se ve afectado por él cuando un usuario quiere acceder a sus pujas realizadas, de manera que será necesario realizar una consulta para recuperar todas las pujas, y una consulta adicional por producto para obtener los nombres de cada uno de los productos que se quieran visualizar en detalle. Este problema también se produce en el caso de uso de obtener los productos de un usuario, ya que este indica la puja ganadora del mismo si es que la tiene, y de cada puja ganadora se obtiene el email del usuario que realizó esa puja, por lo que también se realizaría una consulta por cada una de las pujas ganadoras por producto, y una consulta a mayores por cada uno de los usuarios que proporcionan el correo de contacto.

Para resolver el problema en cuestión, existen diferentes posibilidades, entre las cuales se plantea la modificación del tipo de ejecución, cambiándola de “Lazy” a “Eager” en el parámetro FetchType. Esta solución no es óptima ni la más correcta ya que puede ocasionar el problema del producto cartesiano, por lo que ha sido desechada.

La solución elegida consiste en utilizar el algoritmo *batch fetching*, que consiste en optimizar el número de consultas realizadas, es decir, realizando un único SELECT se obtienen, a partir de los proxies, todos los datos de los productos en vez de obtener cada entidad con un SELECT independiente. De esta manera, basándose en la utilización de la entidad persistente y conociendo la existencia de varias instancias en la misma, es posible realizar esta optimización de número de consultas con el algoritmo mencionado.

Basándose en este concepto, se utiliza la notación @BatchSize en la entidad del producto indicándole el tamaño de entidades que se quieren cargar con el primer SELECT que se produzca al buscar el nombre de un producto. Por defecto se ha determinado una cantidad de 10 elementos, ya que es el número de productos que permitimos mostrar por cada página en el caso de uso desarrollado.

Este algoritmo también es llamado *blind-guess optimization* ya que se desconoce el número de proxies no inicializados en la entidad Product.

Esta solución es óptima y permite reducir el número de consultas realizadas de n+1 consultas a n+1/10 consultas, por lo que optimizará el rendimiento de la aplicación.

3.2 FRONTEND: SNAPSHOT TESTING

Tras la realización del frontend, se propone este trabajo para realizar el desarrollo de las pruebas automatizadas del frontend con la finalidad de comprobar las renderizaciones de los distintos componentes contenidos en la aplicación. Para ello, se realizará el "snapshot testing" del componente FindProductsResult, donde se verifican las renderizaciones del componente que muestra los resultados de búsqueda tanto en el caso de recibir un resultado de búsqueda vacío, como con al menos un producto en su interior.

Para implementar los dos casos de prueba mencionados, se han abordado una serie de problemas que son los siguientes:

El primer problema a abordar, común a ambos casos de prueba, surge con el componente que la aplicación exporta por defecto en la aplicación, ya que este incluye los decoradores de Redux, que no son necesarios para realizar los testeos indicados, pues solamente se quieren comprobar los renderizados a nivel de componente.

Para solucionar este inconveniente, se ha exportado directamente el componente sin los decoradores mencionados anteriormente, para importarlo en el fichero de test e inyectar de manera manual los elementos que sean necesarios para realizar las pruebas pertinentes del renderizado del componente sin tener en cuenta la gestión de eventos y el store de Redux.

En el caso de uso en el que el resultado de búsqueda sea vacío, es necesario mostrar un mensaje indicando lo ocurrido, pero tal y como se ha implementado la práctica ocurre que el *FormattedMessage* que se usa para mostrar el mensaje de aviso de que no hay productos requiera del IntlProvider, encargado de proporcionar los elementos necesarios para la internacionalización del mensaje (locale y messages). Para solventarlo, se ha importado todo lo respectivo al módulo de internacionalización en el cual figuran el locale y los mensajes, y se les ha inyectado mediante el IntlProvider a nivel de componente en este caso de test.

Por otro lado, en el caso de uso en el que el resultado de búsqueda de los productos es mayor o igual que uno, es necesario crear un mock con jest para los componentes Product y Pager, ya que son elementos que no se quieren incluir en los test de prueba, siendo componentes anidados que se deberían testear aparte en su respectivo fichero dedicado al "snapshot testing". Como solo se quiere comprobar que el producto renderiza lo esperado y no lo que está anidado dentro de éste, se ha acudido a la creación de mocks para ambos componentes.

4. PROBLEMAS CONOCIDOS

No se han detectado errores relevantes en la práctica realizada.