UNIVERSIDADE DA CORUÑA

MEMORIA DE LA PRÁCTICA DE PROGRAMACIÓN AVANZADA

Grupo pa008

Javier Lozano Gadín

javier.lozano@udc.es

Álvaro Balirac Seijas

alvaro.balirac@udc.es

1. Arquitectura global

Para la capa modelo se establece un paquete principal llamado model con los siguientes subpaquetes entre otros:

- ***es.udc.pa.pa008.practicapa.model.bid:***

  Este paquete contiene las clases relacionadas con la entidad Bid y su DAO.

- ***es.udc.pa.pa008.practicapa.model.bidservice:***

Este paquete contiene tanto las clases del servicio relacionadas con Bid como las excepciones personalizadas usadas en dichos servicios.

- ***es.udc.pa.pa008.practicapa.model.category***:

  Este paquete contiene las clases relacionadas con la entidad Category y su DAO.

- ***es.udc.pa.pa008.practicapa.model.categoryservice***

  Este paquete contiene las clases del servicio relacionadas con Category.

- ***es.udc.pa.pa008.practicapa.model.product:***

  Este paquete contiene las clases relacionadas con la entidad Product y su DAO.

- ***es.udc.pa.pa008.practicapa.model.productservice***

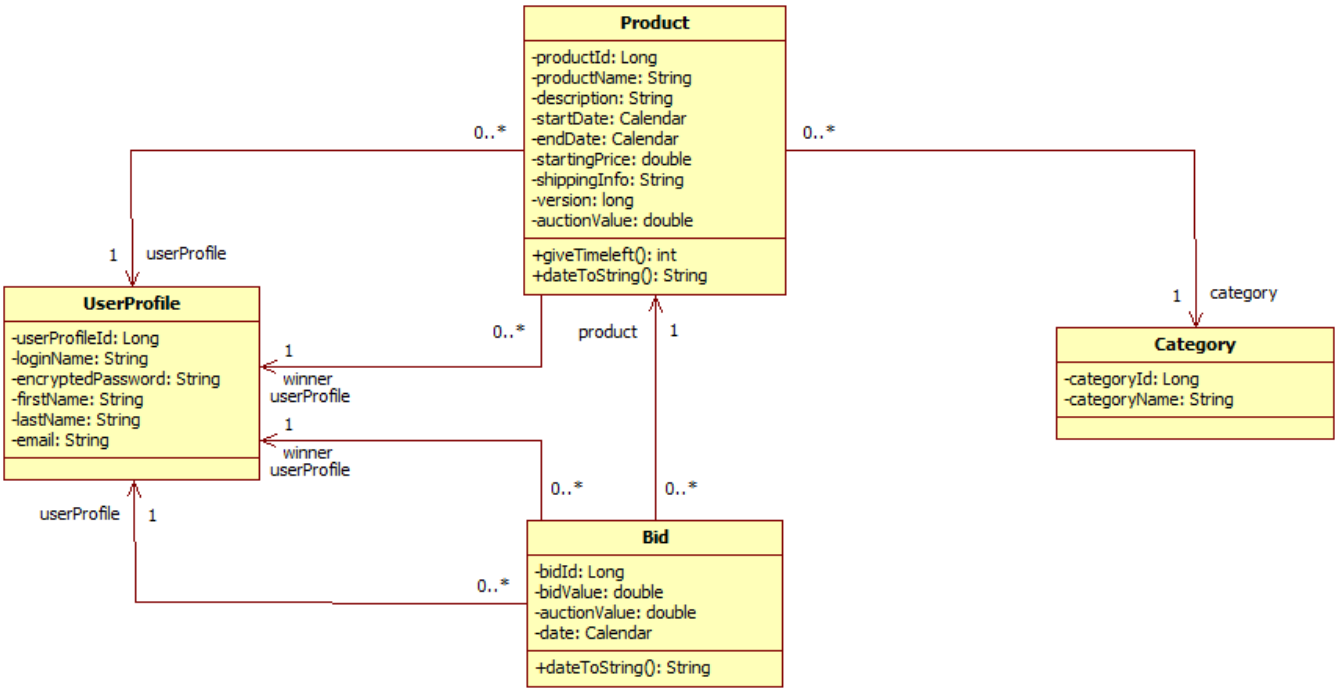
  Este paquete contiene las clases del servicio relacionadas con Product.

Para la capa web se ha establecido un paquete principal llamado web, con los paquetes ***components***, ***pages***, ***services*** y ***util***.

Como subpaquetes de pages se encuentran ***bid***, ***product*** y ***user*** que contienen las clases de las páginas relacionadas con dichas entidades.

2. Modelo

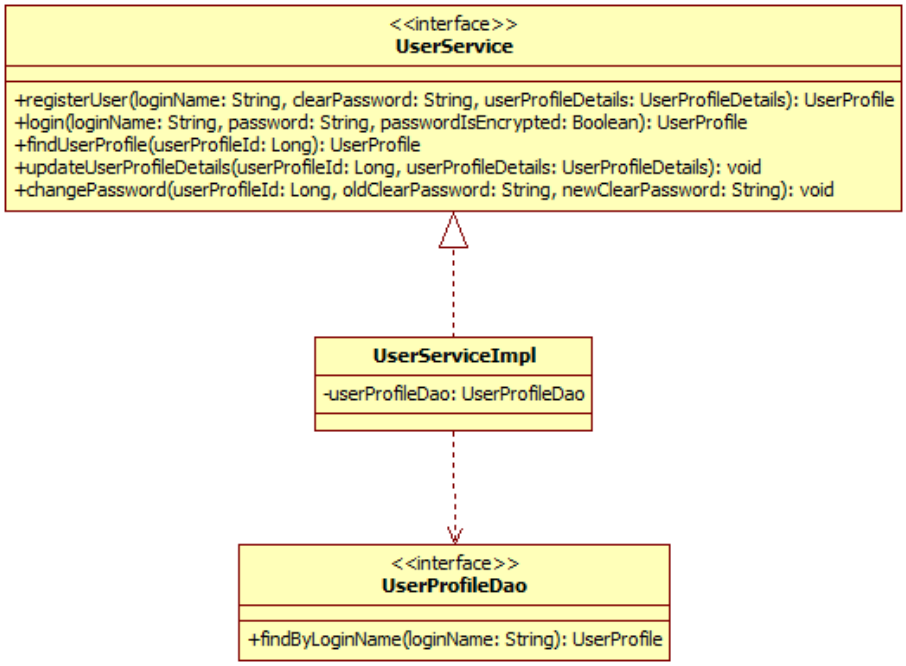
2.1. Clases persistentes



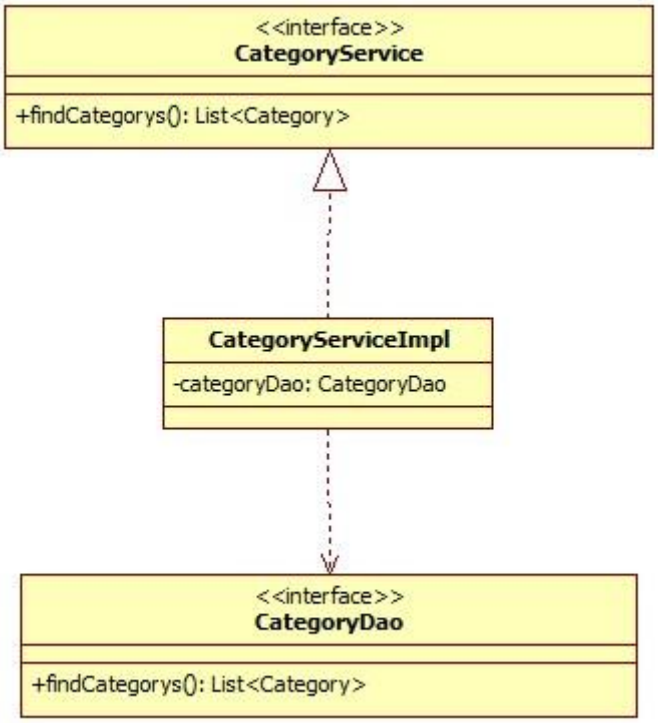
2.2. Interfaces de los servicios ofrecidos por el modelo

A la hora de agrupar los distintos servicios en interfaces hemos dividido dichas fachadas en ***userService***, ***bidService***, ***categoryService*** y ***productService***, englobando dentro de cada una de ellas los servicios que hacen mayor uso de cada una de esas entidades.

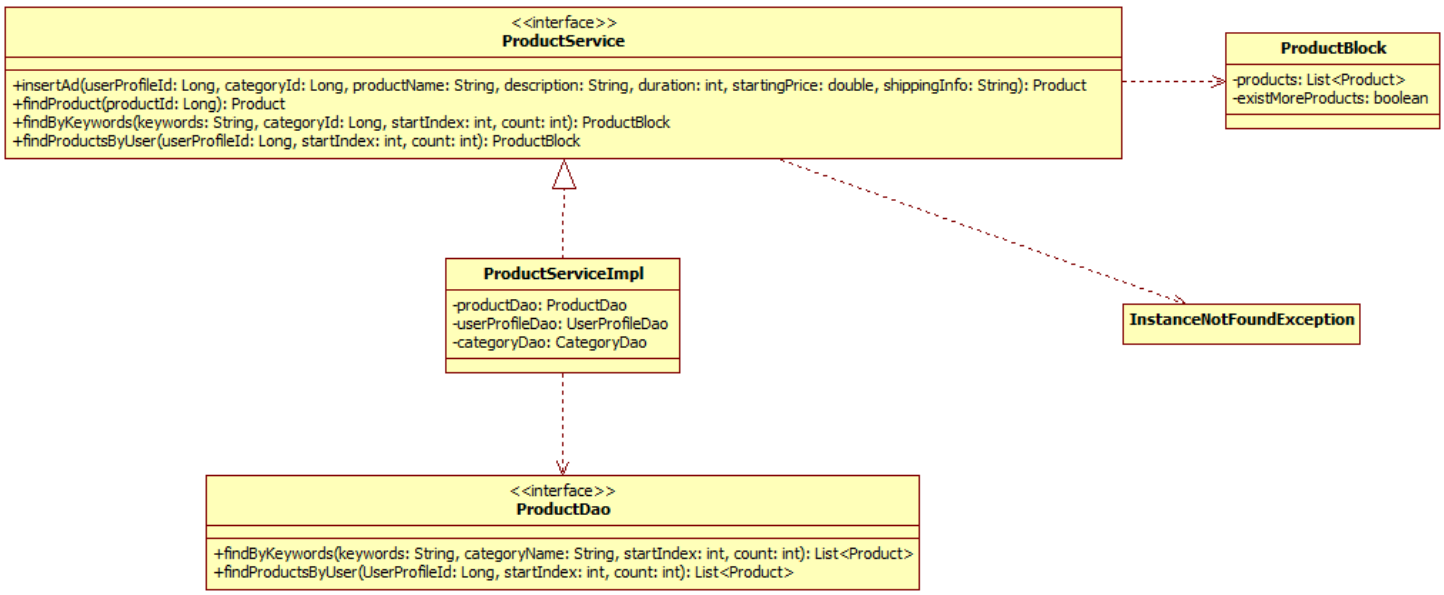
2.2.1. API de UserService



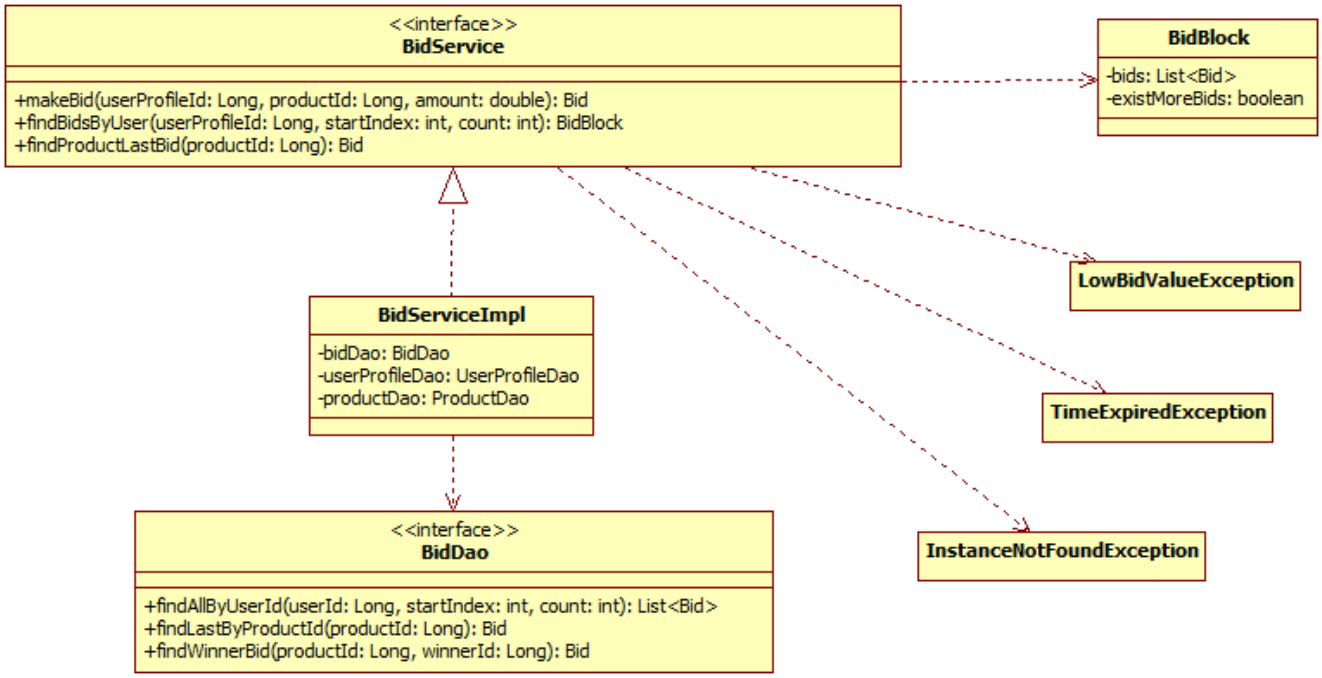
2.2.2. API de CategoryService



2.2.3. API de ProductService



2.2.4. API de BidService



2.3. Otros aspectos

En el campo “proName” de la tabla Product, se ha establecido el charset *latin1_spanish_ci* para que no sea case sensitive, es decir, que en las búsquedas no distinga entre mayúsculas y minúsculas.

En la entidad Product usamos la anotación @BatchSize para mejorar el rendimiento a la hora de lanzar consultas a la BD al ver productos en viewProducts y al consultar el historial de pujas. En ambos casos se consigue realizar menos consultas al intentar obtener datos de product que no sean su ID, ya que si hay varios proxies sin inicializar de product se traen de BD los 3 primeros productos.

En la entidad Product se ha determinado que es más eficiente incluir en esta los atributos auctionValue y winner para reducir el número de consultas a la BD. Por ejemplo a la hora de ver los detalles de un producto, de esta manera no hará falta hacer una consulta a la última puja para saber el ganador o el valor de la subasta.

3. Interfaz gráfica

Para la interfaz gráfica se ha hecho un uso intensivo del framework para html, css y js Bootstrap, usando siempre los estilos predefinidos en Bootstrap. Los enlaces a los historiales de pujas y productos han sido emplazados en el menú desplegable de usuario y los enlaces a la búsqueda de productos e inserción de anuncios están en el menú horizontal principal.

4. Trabajos tutelados

1.1. AJAX

En primer lugar, hemos estimado que el lugar más idóneo para aplicar esta técnica es en el valor de la subasta en *productDetails*. Al hacer click en el enlace con id *refreshZone* se hace una petición XmlHttpRequest y se sustituye la información de la zona *ajaxAuctionValue*.

Las llamadas AJAX han sido realizadas usando las funcionalidades de Tapestry.

1.2. Pruebas funcionales contra la interfaz Web

En primer lugar, se ha creado un nuevo proyecto Maven llamado *pruebasinterfaz* en el que vamos a guardar los casos de uso comprobados. Creamos un *pom.xml* en el que se insertan las dependencias de JUnit y Selenium.

Ya tenemos la base para comenzar, asique creamos la clase ***AutenticationTest*** que albergará la comprobación de la funcionalidad de autenticar un usuario y la clase ***makeBidTest*** en la que se encuentra la comprobación de realizar una puja con acciones posteriores.

En los dos Test anteriores existe una estructura muy similar:

1. Se crea una instancia del driver del navegador que vamos a usar para la prueba (en nuestro caso Firefox)
2. Se proporciona al driver una ruta en la que se cargará la aplicación.
3. Se capturan los objetos pertinentes, se introducen valores, comprueban condiciones o avanza en las páginas con las funciones disponibles de WebDriver.
4. Se comprueban las condiciones necesarias mediante el *asserEquals.*
5. Una vez finalizadas las comprobaciones, se cierra el navegador.

5. Compilación e instalación de la aplicación

Consideraciones a tener en cuenta para ejecutar la práctica en cualquier máquina independientemente del sistema operativo instalado:

- Las BBDD deben llamarse pojo y pojotest, con usuario pojo y contraseña pojo.
- MySQL arrancará en el puerto por defecto.

1. Arrancar la BD MySQL
2. Compilar aplicación con **mvn sql:execute install**
3. Generar el WAR de la aplicación web con **mvn package**
4. Copiar el WAR al directorio de TomCat con **cp <nombreProyecto>.war <<Tomcat>>/webapps**
5. Lanzar el servidor Tomcat con **<<Tomcat>>/bin/startup.sh**

6. Problemas conocidos

Al añadir los datos iniciales desde la página datagen, las pujas se añaden todas de golpe, y aunque dichas pujas se han realizado correctamente, no se muestran en el orden correcto por fecha en el historial. Esto no pasa si se hacen pujas directamente desde la página makeBid.