Progrmación Avanzada
Sitio Web de Subastas

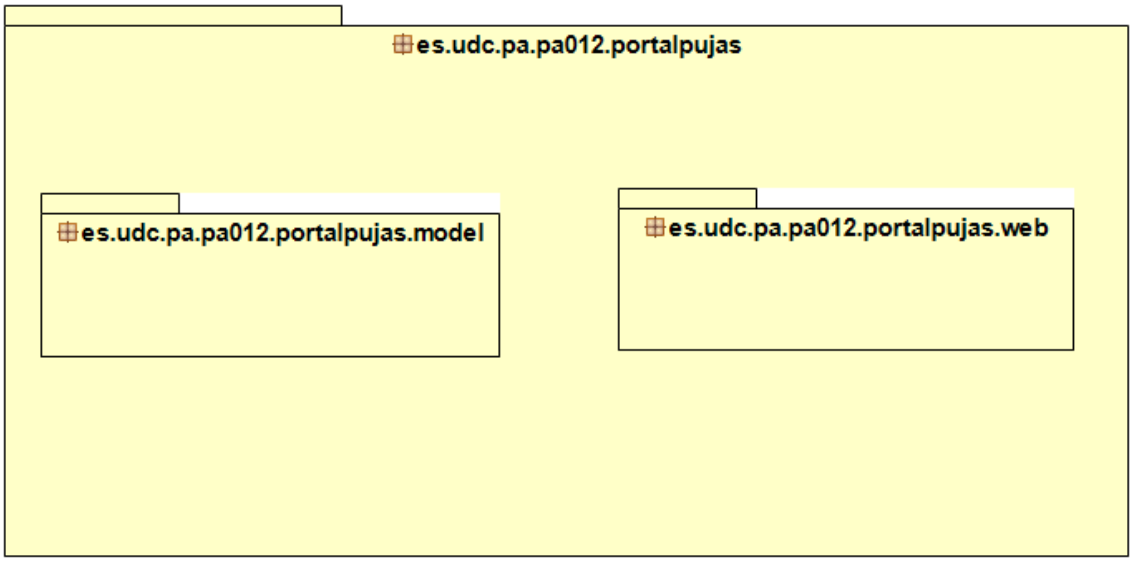
7 de junio de 2015

Castro Fernández, Laura
Zas Vázquez, Christopher

Índice

1. Arquitectura global

La aplicación se ha dividido en varios paquetes. En la siguiente imagen podemos ver los paquetes de más alto nivel que tenemos



- `es.udc.es.pa.pa012.portalpujas` Paquete principal del proyecto, contiene a los dos principales paquetes del sistema:

  `es.udc.pa.p012.portalpujas.model` Paquete correspondiente a la capa modelo, contiene las clases persistentes así como los servicios que se han implementado.

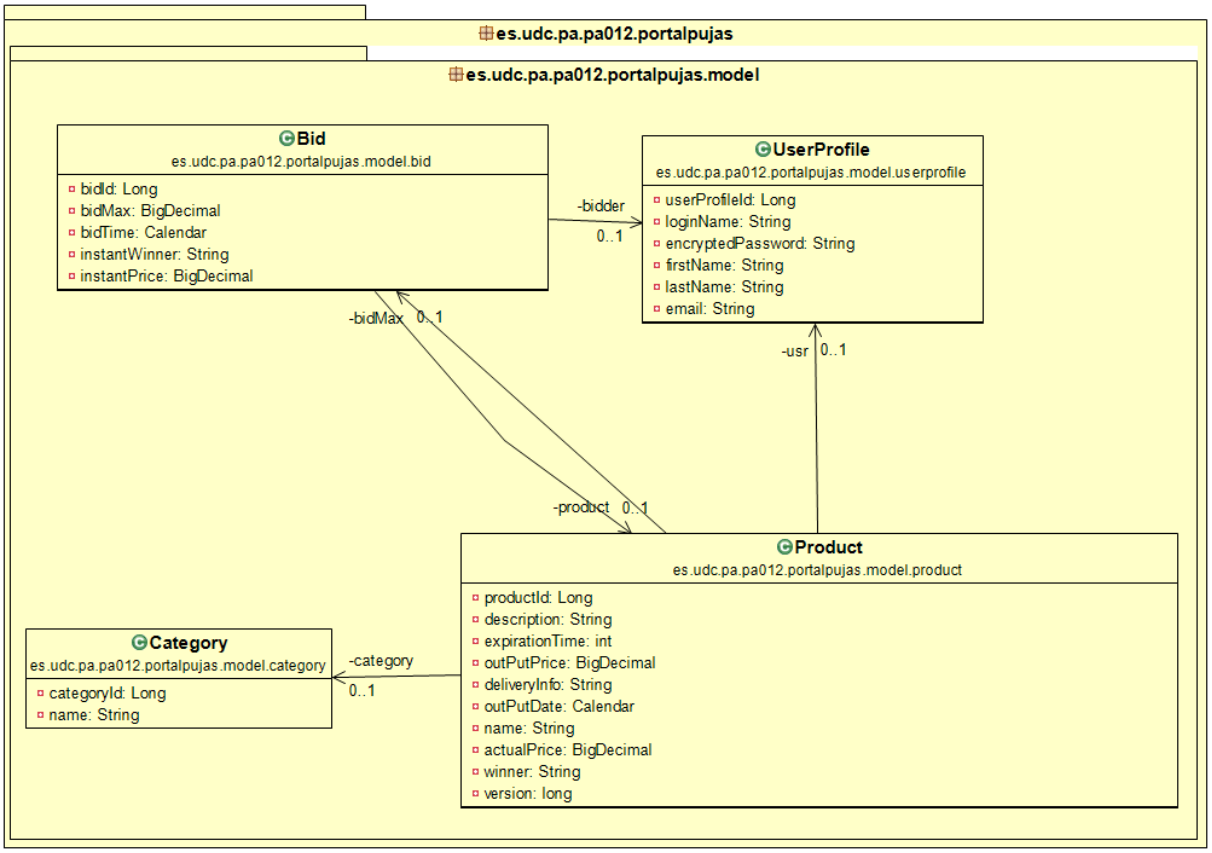
  `es.udc.pa.pa012.portalpujas.web` Contiene las clases necesarias para la implementación de la Capa Web del sistema.

2. Modelo

El modelo, o Capa Modelo es el onjunto de clases que implementan la lógica de negocio de los casos de uso de la aplicación. A continucación mostraremos las entidades persistentes que se han diseñao así como los diferentes servicios implementados.

2.1. Clases Persistentes

El siguinete diagrama muestra las entidades que contiene nuestro sistema.



Se puede apreciar que se han implementado cuatro entidades persistenntes. *Categoría(category)*, *Puja(bid)*, *Producto(product)* y *Usuario(userprofile)*.
Se ha establecido una relación Product-Category dado que un producto siempre pertenece a una categoría, del mismo modo existe un realción Product-UserProfle ya que un producto siempre tiene asociado un usuario. Además, existe la relación Bid-UserProfile para tener información del usuario que ha llevado a cabo una puja. Por último existen las relaciones Product-Bid para poder tener información de las pujas de un producto, tales como quén ha pujado y cuánto o simplemente cual es la puja más alta de un producto.

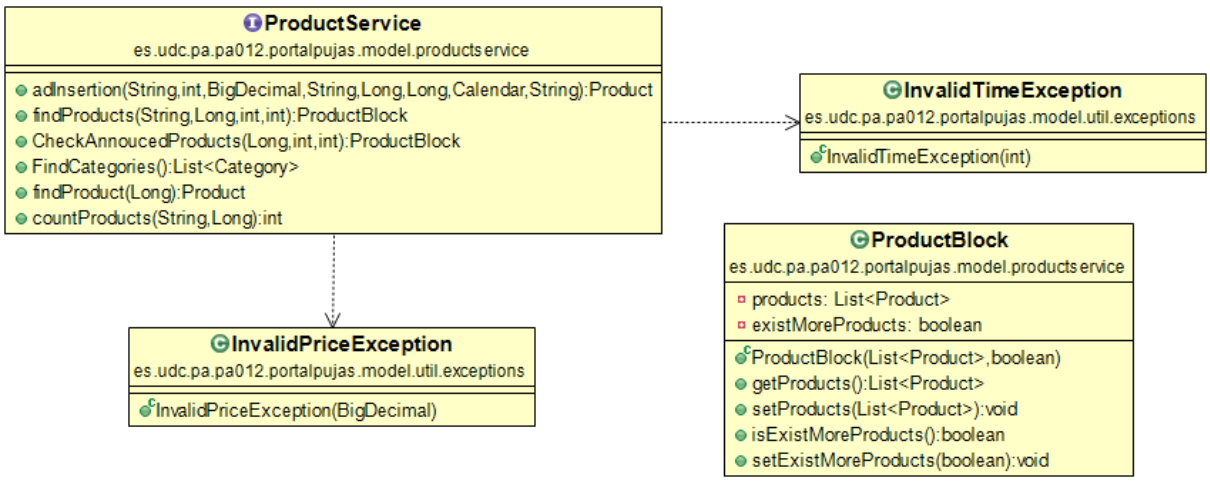
2.2. Intrfaces de los servicios del modelo

En la aplicación existen los siguientes servicios:

- `productService`
- `bidService`
- `userService`

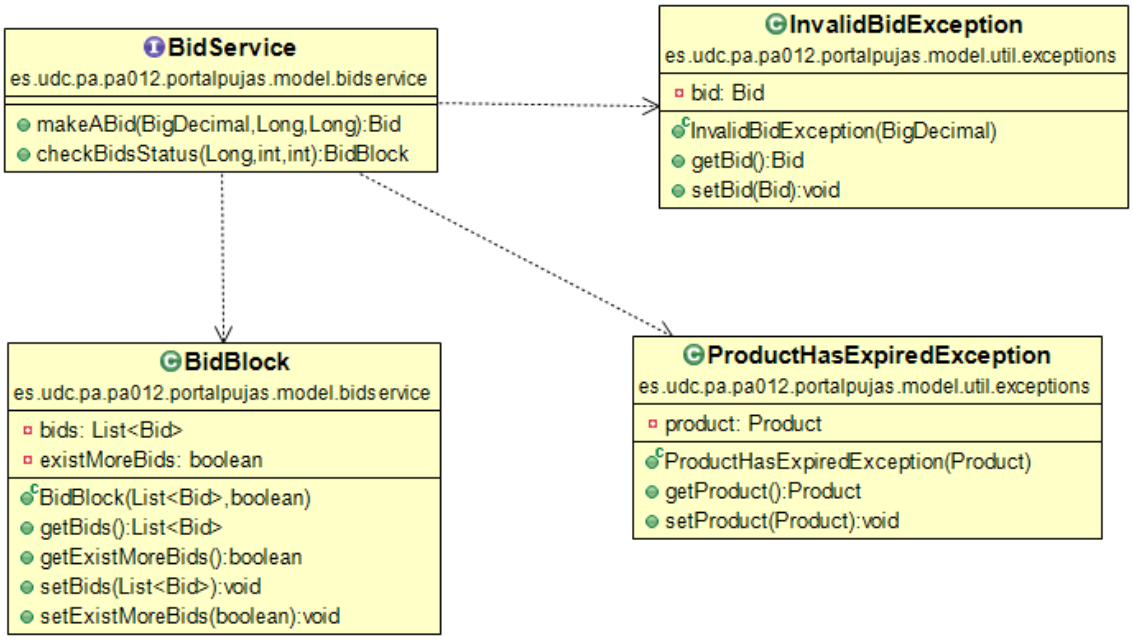
Se ha optado por agrupar los casos de uso más ligados a los productos, como son la creación de los mismos o la visualización de la nformación de cada uno, en el servicio `productService`. Los casos de uso relacionados con las pujas, como pueden ser, realziar una puja o obtener un histórico de las mismas se han implementado dentro del servicio `bidService`. La gestión de los usuarios del sistema se encuentra en el `userService`.
A continuación se mostrarán los diagramas correspondientes a casa servicio, mostrando las interfaces de los mismos.

Diagrama del servicio `productService`



En el diagrama se pueden apreciar las operacioens que lleva a cabo este servicio. Además se puede ver que se han creado dos excepciones. Una para controlar que el tiempo de expiración de un produco que se pone a la venta sea válido y otra para controlar que el precio sea correcto. Además se ha creado la case `ProductBlock` la cual no es una clase persistente, sino una clase que representa los datos de salida de algún caso de uso de este servicio.

Dagraama del servicio `bidService`

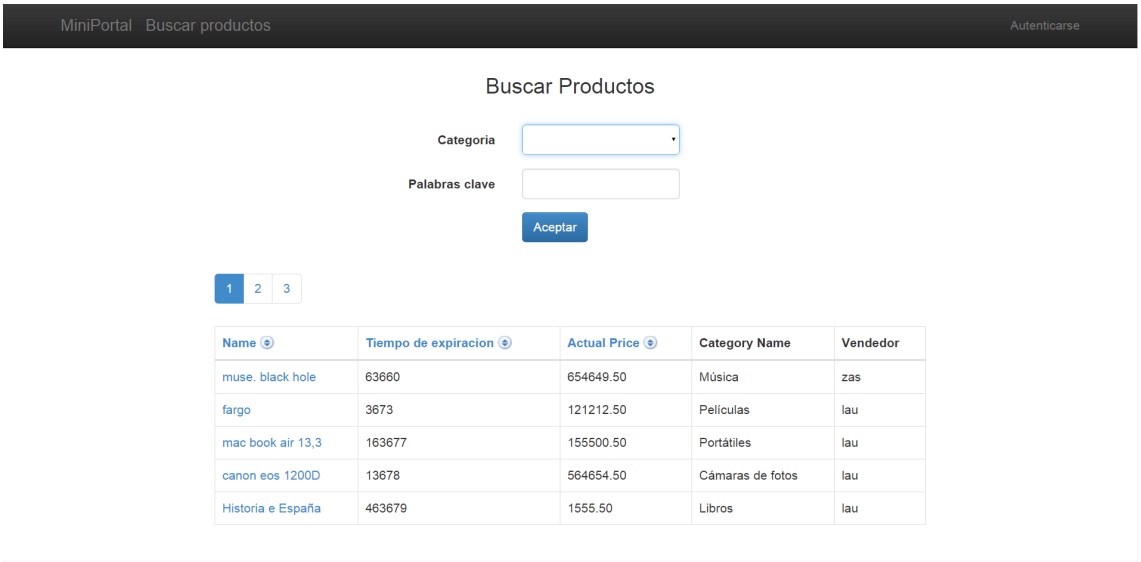


Este servicio hace uso de dos expciones, `ProductHasExpiredException` la cual se lanza cuando un usuario intenta pujar por un producto y este ha espirado. Por otro lado `InvalidBidException` se obtiene cuando el valor de una determinada puja no es el correcto, por ejemplo, cuando es mejor que el precio. Al igual que en el anterior servicio, se ha creado la clase `BidBlock` para representar los datos de salida del caso de uso que muestra información de las pujas, `checkBidStatus`.

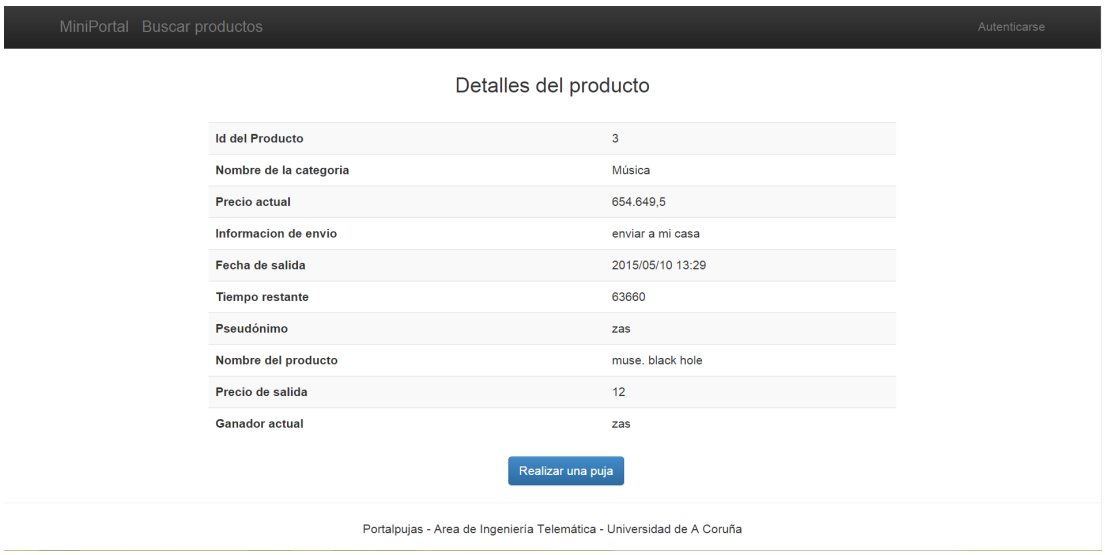
3. Interfaz gráfica

Todo lo descrito anteriormente se ha uilizado para la implementación de la Capa Web. Veámos una serie de capturas de la interfaz y una breve descripción de las funcionalidades de la misma.

Una de las funcionalidades que se ofrece es la búsqueda de productos. Un usuario autenticado o no puede buscar todos los existentes o realizar una búsqueda más exhaustiva por palabras clave que contiene el nombre del producto o por la categoría a la que pertenece. Se ha usado el componente *Grid* de tapestry para realizar la paginación del resltado de la búsqueda. La siguiente imagen muestra como ha quedado el diseño de esta búsqueda.

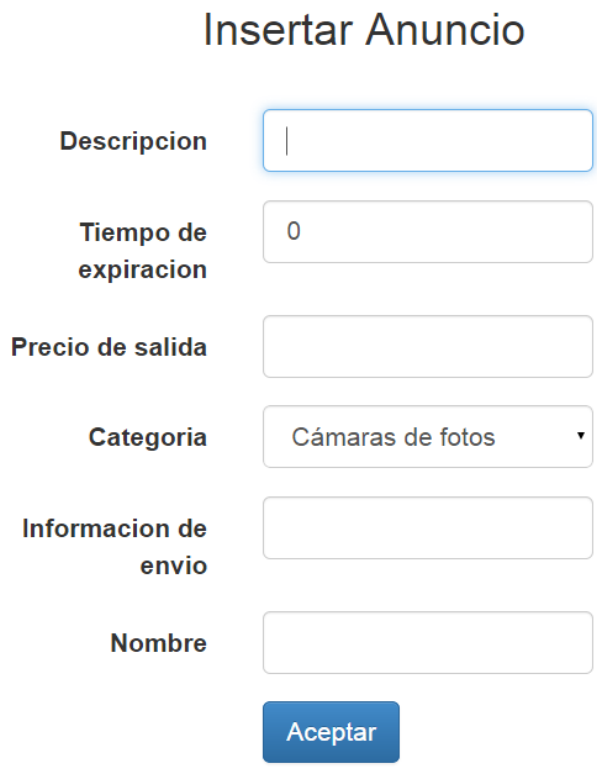


Con el resultado de una búsqueda se puede acceder a la información de un producto haciendo click sobre su nombre, la información mostrada se puede ver en la siguiente imagen

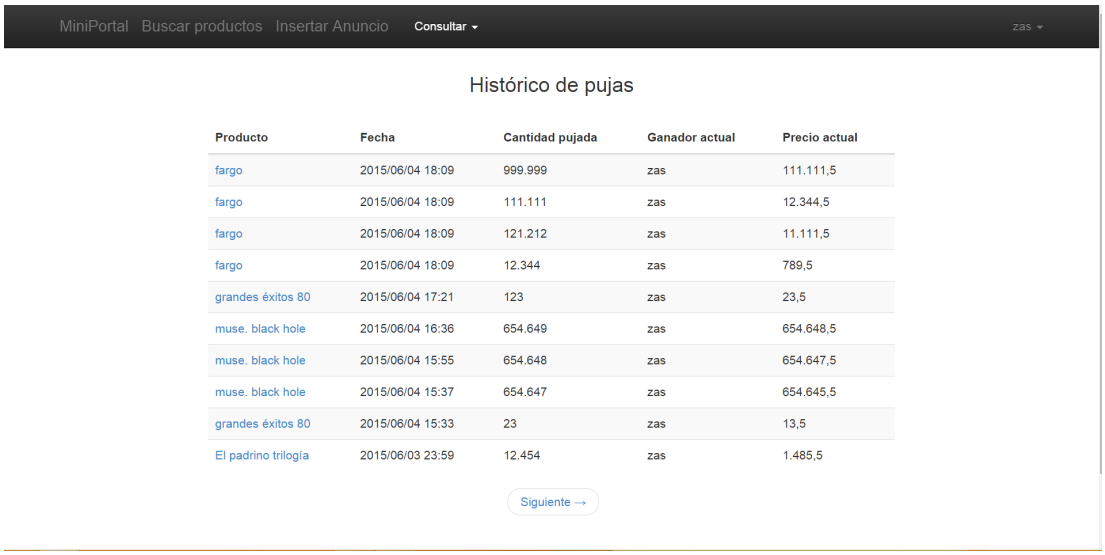


En esta página, si el producto no ha caucado, se muestra un botón para poder pujar por el producto. Sólo pueden pujar por los productos usuarios autenticados. El formulario de puja se muestra a continuación

Un usuario registrado puede dar de alta productos, la información que se le pide se muestra en la siguiente imagen que sería el formulario que la aplicación le muestra para dar uno de alta.



Además de estas funcionalidades un usuario autenticado podrá ver los productos por los que ha pujado a lo largo del tiempo así como los productos que ha anunciado si es el caso. Para llevar a cabo la paginación en el caso de que se devuelva un número elevado de productos o pujas se ha optado por usar el patrón *Page by Page Iterator* con el cual el resultado se mostrarán en rangos de 10 en 10 y con enlaces *Anterior/Siguiente* para que un usuario pueda recorrer los productos o pujas comodamente bajo su criterio. A continuación se muestra una imagen de un histórico de las pujas de un usuario.



4. Trabajos tutelados

Se ha implementado AJAX en la aplicación. Para la implementación se ha usado el soporte AJAX que ofrece Tapestry. Tapestry hace uso de *Zonas* pare realizar actualizaciones parciales de la página, para ello proporciona un componente `Zone`. Una zona se puede actualizar a traves de un *EventLink ActionLink* o medianto un formulario. Al hacer clic en ese vínculo invocará un método de controlador de eventos en el servidor de forma normal, excepto que el valor de retorno del método controlador de eventos se utiliza para enviar una respuesta parcial de la página al cliente, y el contenido de esa respuesta se utiliza para actualizar la Zona .
En la página de búsqueda de productos se ha usado este enfoque para recargar la zona que contiene el grid con el resultado de la búsqueda. Veámos cómo sería el código:

```
1
2 <t:zone t:id="formZone" id="formZone">
3     <t:grid >
4         [...]
5     </t:grid>
6 </t:zone>
```

En la clase java asociada a esa página se ha declarado un componente Zone con el nombre `formZone` el método que actualiza esta zona sería el siguiente:

```
1
2   void onValidateFromSearchForm() {
3     if (request.isXHR()) {
4       ajaxResponseRenderer.addRender(formZone);
5     }
6   }
```

`searchForm` es el componente asociado al formulario de búsqueda, si el formulario no contiene errores el componente `AjaxResponseRenderer` indicará las zonas que se tienen que actualizar, en este caso, las zonas cuyo id sea `formZone`

Este enfoque se ha seguido para introducir AJAX en la paginación de los históricos de productos y pujas de un usuario. Así como en la tabla con información que se muestra en el formulario de pujar. Si la información de esa tabla varía, por ejemplo, si nos muestra como ganador a otro usuario pero al pujar nosotros pasamos al ser el ganador, AJAX actualizará esa tabla.

5. Compilación e instalación de la aplicación

Antes de nada arrancaremos la BD con el comando **mysqld**

Para la arrancar la aaplicación desde **Jetty** nos situamos en el directorio de la aplicación y ejecutamos **mvn sql:execute install**, finalemente ejecutamos **mvn jetty:run**.
Abrimos un navegador y accedemos a **http://localhost:9090/portalpujas/**.

Si queremos ejecutar la aplciaciós desde **Tomcat**, suponiendo que éste ya está configurado, debemos primero crear el archivo WAR, con el comando **mvn package**. Este archivo lo copiaremos al directorio /webapp del Tomcat. Despues nos situamos en el directorio /bin y ejecutamos el script startup.sh. Por ultimo accedemos desde el navegador a la URL: **http://localhost:8080/portalpujas/**.