## Comunicación y estructura de desarrollo

Para la realización del proyecto se ha realizado un repositorio en git que se ha dividido en dos branches (ramas).

- Rama master: La rama master sirve para realizar modificaciones finales, versiones más antiguas y menos modificadas pero siempre totalmente funcionales.
- Rama develop: La rama develop es donde se realizan todos los cambios. Cada cambio realizado por un miembro de alguno de los dos equipos de proyecto se ha ido subiendo a git, siempre y cuando fuese ya un cambio completo (una funcionalidad).

La estructura de desarrollo se ha realizado, primero dividiendo el grupo en dos equipos y separando así la realización de la base de datos de la documentación relacionada con la arquitectura del sistema (Diagramas de clases y secuencia) en la primera parte de esta entrega.

En la segunda parte de la entrega, se ha reasignado uno de los miembros del equipo B al equipo A y se han preparado dos nuevas tareas. El equipo A ha realizado la codificación del front-end y una pequeña parte del back-end, mientras que el equipo B se ha encargado de la arquitectura del sistema y el el resto del back-end.

## Arquitectura de Desarrollo

La arquitectura de desarrollo del sistema sigue tal cual el patrón arquitectónico Modelo-Vista-Controlador (MVC). En este tipo de arquitectura se separan los datos, la lógica de negocio y la interfaz de usuario. Este patrón y esta separación hace más reutilizables los componentes de cada una de las partes sin tener que tocar otros. Por ejemplo, es posible modificar las vistas o la lógica de los datos sin necesidad de tocar el modelo (las entidades y la interacción con la base de datos).

### Capa de acceso a datos

Esta capa se puede incluir o no dentro del modelo, dependiendo de lo detallada que sea la implementación del MVC. En este caso hemos creado un namespace aparte “database” que se ocupa del acceso a los datos.

Para el acceso a datos hemos implementado un patrón DAO (Data access object), que está representado en el diagrama de clases:

- **Interfaz DAO:** Define una interfaz para el acceso a datos. En el caso de que en determinado momento se quiera cambiar la base de datos MySQL por ficheros XML, por ejemplo, solo habría que añadir una nueva clase que implemente a esta con los mismos nombres en los métodos. Nunca sería necesario hacer por tanto modificaciones en la

vista o en la lógica (controladores), ya que aunque la parte de persistencia ha cambiado la interfaz y métodos a los que accede siempre son los mismos.

- **Clase abstracta SQLDAO:** Aquí se realiza el código común de todo el acceso a base de datos, es decir, el que compartirán todas las clases concretas que hereden de esta (BiddingDAO para subasta o ProductDAO para producto).
- **Clases DAO concretas:** Aquí se implementa código concreto para una clase concreta. En nuestro caso estas clases llaman solo al constructor de la clase abstracta, ya que se ha implementado de forma que en esta se genere código reutilizable para todas las clases de abajo.

### Modelo

Los modelos se corresponden con las entidades que se muestran en el diagrama de clases. Los modelos representan a una entidad real (Usuario, Administrador, Producto…), y se ocupan de interactuar con la capa de acceso a datos.

El controlador se ocupa de interactuar con los modelos y estos contienen métodos para realizar operaciones con la base de datos, ya sea modificándola u obteniendo datos de esta.

Entre las características del modelo en el diagrama de clases es importante hablar de otras dos clases (DAOFactory y Model).

DAOFactory es una clase del paquete de acceso a datos que sirve para crear DAOs en función de lo que se necesite. Un ejemplo sería cuando se añade una nueva subasta, para añadirla a la base de datos hay que crear un nuevo DAO y para esto se puede pasar como parámetro el nombre de la propia entidad “subasta”. Consideremos que se añade una nueva subasta (clase Bidding), pasando al DAOFactory el nombre de esta clase este creará, en base al nombre de esta entidad un SQLBiddingDAO, si en vez de eso se pasase “Product” se crearía un SQLProductDAO. Este fenómeno se llama Reflexión (modifica en tiempo de ejecución el comportamiento del programa en base al nombre de la entidad, y el patrón de creación es una factoría, que crea nuevos objetos dependiendo de un parámetro que se le pase.

Model es una clase abstracta con las operaciones comunes de todos los modelos, es decir, el contacto con la capa de acceso a datos para almacenar, borrar y validar.

### Controlador

Dentro del paquete de controladores se incluye la lógica de negocio de cada clase concreta (un controlador por entidad). Los controladores se encargan de abstraer las vistas de los modelos, gestionando su interacción con ellos y haciéndolos así más reutilizables (el cambio en un modelo no afecta a las vistas y viceversa). Los controladores disponen de una clase abstracta común para compartir algunos métodos para todos los controladores.

Dentro del paquete de controladores se contiene también una clase Router que se encarga, según la petición, de elegir el controlador concreto que se va a usar, crearlo y llamar a la acción (método) correspondiente.

### Vista

La clase View se asocia con las distintas plantillas y carga los archivos correspondientes a cada vista según la acción definida por el controlador asociado a dicha vista.

## Sistema de comunicación y control de cambios

### Diagrama de casos de uso

Se ha trabajado sobre la ET1 considerando ciertas modificaciones desde el inicio en el prototipado falso. Se ha modificado un pequeño detalle en el diagrama de casos de uso en relación a las tareas del usuario, que consideramos que no debe modificar ni eliminar subastas. En caso de que alguien puje no tendría lógica que sea posible que el usuario pueda eliminar el producto de la subasta. Quizás se podría considerar también que se eliminase si no hay pujas, pero no hemos entrado a ese nivel de detalle.

### Prototipado falso

El grupo A se ha ocupado de retocar el prototipado según iban avanzando las vistas, aunque la versión final se ha dejado para la última semana, ya que según iba avanzando el sistema funcional se iban encontrando nuevos errores de prototipado, por lo cual no tenía sentido realizar excesivos cambios en ese punto y tampoco en estado y transiciones. El prototipado se ha modificado en varias ocasiones pero no se ha realizado una versión final de todo hasta el último día, debido a lo comentado.

Considerando estos problemas con el prototipado, se ha realizado la nueva versión de estados y transiciones considerando el último prototipado solo una vez, ya que no consideramos productivo en este caso gastar recursos en hacer modificaciones continuas de ese punto. Hemos ido modificando solo pequeños detalles, dándole así un poco más ágil al desarrollo.

## Pequeños detalles del diagrama entidad-relación

- Análisis de requisitos
    - Eliminada la posibilidad de modificación de pujas.
    - Eliminada la posibilidad de que el usuario elimine sus pujas. Solo el administrador debe poder eliminarlas. (*no se especifica eliminacion de subastas)
- Diagrama de casos de uso.
    - Se ha eliminado la posibilidad de modificar pujas.
    - El usuario ya no puede eliminar pujas, en cambio el administrador sí.
- Descripción de casos de uso
    - Eliminación de “Modificar Pujas”
    - Ahora solo el administrador puede eliminar pujas.
    - Modificada posibilidad de cambiar porcentajes al administrador desde vista usuario, ahora solo es posible realizarla desde la vista administrador.
- Diagrama entidad-relación
    - Se creó una entidad Calificación relacionada con los productos y los usuarios con los atributos idCalificacion, puntuacion y comentario.
    - Se eliminaron los atributos cantidad y tipo de la entidad Producto.
    - Se añadió el atributo estado a la entidad Producto.
    - Se cambió la cardinalidad de la relación Producto-Subasta a (0,1).
    - Se eliminaron los atributos nombre, tiempoLímite, descripción y fecha de la entidad Subasta.
    - La entidad Puja se transformó en entidad débil para facilitar la implementación.
    - Se eliminaron los atributos fecha, hora y secuencia de la entidad Puja y se añadieron idPuja y fechaPuja.
    - Se añadió la relación Puja - Pago (1,N)
    - Se añadieron los atributos: cuentaPaypal, numTarjeta, metodoPago y comision a la entidad Pago
    - Se creó la entidad Compra relacionada con Venta, Pago y Usuario con los atributos idCompra, fechaCompra y cantidad.
    - Se cambio la cardinalidad de la relación Producto Venta a (0,1).
    - Se añadió el atributo stock a la entidad Venta.
    - Se eliminaron los atributos: cuentaPaypal, cuentaBancaria, fechaNacimiento y movil de la entidad Producto.
    - El atributo telefono de la entidad Producto pasó a ser simple.
    - Se creó una relación Usuario - Puja (0,N)
- Prototipado falso
    - Modificados los menús laterales con añadidos.
    - Se eliminaron las pantallas de error y confirmación para que simplemente, en caso de error, vuelvan a sí mismas (en la implementación vuelven con un mensaje que solo aparecerá una vez.

    - En la pantalla principal antes salían subastas y productos mezclados, ahora hay una lista para cada uno.
    - En la pantalla principal de usuario logeado, pasa de mostrar productos basados en preferencias a los nuevos productos en venta y subasta, igual que si no estuvieses registrado.
    -
- Diagrama de estados.
- Diagrama de transiciones.

### Sistema de comunicación entre los miembros del grupo

Para la comunicación entre los dos equipos se ha usado principalmente un grupo de Whatsapp, que ha servido especialmente para formalizar reuniones continuas así como acordar fechas y horas para realizar partes del trabajo entre los miembros. Para noticias o información larga y más detallada se ha hecho uso del correo electrónico.