FrontEnd y BackEnd de Vinted

Tecnologías informáticas de la web curso 20-21

Grupo 23
Daniel Zubieta Pascual 100346138
Adrián García Carrasco 100346192
Ignacio González Díaz Tendero 100346133
Sergio Cruzado Muñoz 100346032

Índice

Introducción

Para esta práctica se va a desarrollar una versión propia de la aplicación de venta en línea Vinted. En esta página los usuarios pueden comprar y vender productos, principalmente productos textiles de segunda mano. Para ello debemos implementar y desarrollar las distintas funcionalidades que ofrece Vinted. En este proyecto, nos centraremos principalmente en desarrollar el back-end del sitio web.

Para este proyecto se hará distinción entre dos perfiles muy distintos, el perfil de usuario y el de administrador.

El proyecto, como indica el enunciado, se divide en dos ficheros *.war* según los dos tipos de usuario que tiene la página: uno para el perfil de compradores y vendedores, y otro para el perfil del administrador.

Reparto de la implementación de las funcionalidades y documentación.

Daniel Zubieta Pascual	Usuario estándar: • Darse de alta en la aplicación • Modificación de información del perfil de usuario • Baja de la aplicación • Identificación Vendedor: • Dar de alta un producto • Consultar productos ofertados y vendidos • Modificar datos del producto • Dar de baja el producto Comprador • Búsqueda de productos • Búsqueda avanzada • Ver información de un producto • Añadir producto carrito de la compra • Eliminar producto del carrito de la compra • Compra • Consultar compras realizadas Usuario administrador • Modificar usuario • Dar de baja a un usuario • Modificar un producto • Dar de baja un producto
Adrián García Carrasco	Usuario estándar: • Darse de alta en la aplicación • Modificación de información del perfil de usuario

	• Baja de la aplicación • Identificación Usuario administrador • Modificar usuario • Dar de baja a un usuario Memoria
Sergio Cruzado Muñoz	Usuario estándar: - Dar de alta un producto - Consultar productos • Comunicación Usuario administrador • Comunicación HTML y CSS de las páginas web usadas
Nacho González Díaz Tendero	Usuario estándar: • Comunicación Usuario administrador: • Comunicación HTML y CSS de las páginas web usadas

Antes de explicar la implementación de cada funcionalidad, debemos hacer una serie de consideraciones:

- Entre las clases cuyo nombre figura DAO y VO corresponden a accesos JDBC a la base de datos.
- Las clases UserManager y ProductManager corresponden a accesos JPA a la base de datos.
  Al tener diferentes accesos sobre la misma base de datos, incurrimos en un polimorfismo, por ejemplo, entre UserVO y UserJPA. Como consecuencia del polimorfismo, ha sido necesario vaciar la caché de la base de datos entre los distintos accesos para evitar errores. En código:
  `em.getEntityManagerFactory().getCache().evictAll();`

Esta línea de código ha tenido que ser añadida a las clases productManager y userManager, ya que al hacer un acceso con JDBC, si previamente se había accedido usando JPA se queda la cache de JPA y da error con el acceso en JDBC. Esto disminuye un poco la eficiencia de la aplicación, pero al ser un proyecto pequeño y no enfocado al rendimiento, no hay problema.

Funcionalidades de usuario

Perfil:

Lo primero que debemos hacer es implementar un modelo vista-controlador, por lo que creamos la clase UserController que será nuestro controlador para todos los métodos de usuario. Así como la clase User.java donde quedan recogidos los atributos y métodos del usuario.

Darse de alta en la aplicación

Esta funcionalidad queda recogida en el servlet Register.java. En esta clase, dentro de un método post hasheamos la contraseña para dotar de seguridad al proceso, creamos un objeto usuario para capturar los parámetros del registro, llamamos al método `registerUser()` de la clase UserController, este a su vez llama al método `registerUser()` de la clase UserDao. Este método prepara y lanza una sentencia SQL (acceso JDBC) para registrar al usuario `”INSERT INTO USER (name, surname, address, email, passwd) VALUES (?,?,?,?,?)"`. Una vez terminado el registro en la base de datos, dentro de la clase Register.java redirigimos al usuario a la página principal.

Modificación de información del perfil de usuario

Esta funcionalidad queda recogida en el servlet Modify.java. En esta clase, dentro de un método post recogemos los datos del usuario a modificar dentro de un objeto UserVo, recogemos los nuevos parámetros que se van a modificar usando `request.getparameter()`, llamamos al método `modifyUser()` de la clase UserController, este a su vez llama al método `modifyUser()` de la clase UserDao. Este método prepara y lanza la sentencia SQL (acceso JDBC) contra la base de datos para modificar los datos del usuario, `"UPDATE user SET name=?, surname=?, email=?, passwd=?, address=? WHERE ID=?"`, en caso de no insertar nueva contraseña se presupone que la anterior no se modifica por lo que la sentencia en este caso es `"UPDATE user SET name=?, surname=?, email=?, address=? WHERE ID=?"`. Una vez se han modificado los datos volvemos a la clase Modify.java, creamos una sesión nueva para el usuario y redireccionamos al usuario.

Baja de la aplicación

Esta funcionalidad queda recogida en el servlet DeleteUser.java. En el método post llamamos al método `deleteUser()` de la clase UserController, este a su vez llama al método `deleteUser()` de la clase UserDao. Este método prepara y lanza la sentencia SQL (acceso JDBC) contra la base de datos para eliminar al usuario “`DELETE FROM user WHERE ID=?”`. Una vez se ha borrado al usuario, volvemos a la clase DeleteUser.java, invalidamos la sesión actual del usuario y lo redirigimos a la página principal.

Identificación

Esta funcionalidad queda recogida en el servlet Login.java. En el método post hasheamos la contraseña para que el sistema sea seguro, mandamos el hash y el correo, llamamos al método `loginUser()` de la clase UserController, este a su vez llama al método `loginUser()` de la clase UserDao, este método prepara y lanza la consulta SQL (acceso JDBC) contra la base de datos usando el correo y la contraseña y si es correcto, devuelve un objeto usuario “`SELECT * FROM user WHERE email=? and passwd=?"`. En el servlet Login.java, iniciamos sesión, el carrito de la compra y redirigimos a la página principal, en caso de error retornamos a la página de login para repetir el proceso.

Vendedor:

Lo primero que debemos hacer es implementar un modelo vista-controlador, por lo que creamos la clase ProductController que será nuestro controlador para todos los métodos de producto. Así como la clase ProductVO.java donde quedan recogidos los atributos de los productos, así como los métodos getters y setters.

Debemos destacar la creación del método auxiliar `getProductId()` que utilizaremos para obtener el id del producto y así poder recuperar los atributos asociados al producto.

Dar de alta un producto

Esta funcionalidad queda recogida en el servlet RegisterProduct.java. En el método post, debido al encoding multipart mediante el cual se envía la petición, cogemos la Part que corresponde a la imagen. Creamos el objeto ProductoVo para almacenar sus características y llamamos al método `registerProduct ()` de la clase ProductController, este a su vez llama al método `registerProduct()` de la clase ProductDao. Dentro de este método creamos un nuevo objeto conexión, preparamos la sentencia SQL (acceso JDBC), guardamos los valores para usarlos en la sentencia SQL, ejecutamos el comando SQL `"INSERT INTO PRODUCT (title, category, description, image, price, id_user) VALUES (?,?,?,?,?, ?)"` y devolvemos "True" si se ha ejecutado correctamente o "False" en caso contrario.

A continuación, en la clase RegisterProduct.java redirigimos a la página principal.

Consultar productos ofertados y vendidos

Esta funcionalidad está implementada en el servlet GetProductOwner.java.

En el método post llamamos al método `findByPersonId()` de la clase ProductManager y le pasamos el id del usuario del que queremos recuperar los productos. Este método nos devuelve una lista con los productos asociados al id del vendedor mediante una sentencia JPA. A continuación, redirigimos a la página `myProducts.jsp.`

Modificar datos del producto

Esta funcionalidad está implementada en el servlet ModifyProduct.java.

En el método post almacenamos temporalmente los nuevos parámetros de la modificación, creamos un objeto ProductVo, con esos parámetros y llamamos al método `modifyProduct()` de la clase ProductController, que a su vez llama al método `modifyUser()` de la clase ProductDao .

Para modificar un producto (método `modifyUser()`), nos creamos una nueva conexión, preparamos la consulta SQL (acceso JDBC) "`UPDATE product SET title=?, category=?, description=?, image=?, price=? WHERE ID=?`" y a continuación lanzamos la consulta y actualizamos los atributos del producto con los nuevos introducidos por el usuario

Dar de baja el producto

Esta funcionalidad queda recogida en el servlet DeleteProduct.java.

Para borrar un producto creamos una nueva conexión, almacenamos en un objeto ProductJPA el producto que deseamos borrar, que recuperamos usando el método `findProductId ()` de la clase ProductManager. Si el usuario tiene un producto con el id que se desea borrar llamamos al método `deleteProduct` () de la clase ProductManager que elimina el producto de la base de datos mediante una sentencia JPA. Si el producto ha sido borrado correctamente redireccionamos a `"myProducts?delete=success".`

Comprador

En este caso usaremos el modelo vista-controlador implementado para el apartado de vendedor. La clase SoldProductVO.java donde quedan recogidos los atributos de los productos, así como los métodos getters y setters.

Búsqueda de productos

Esta funcionalidad queda implementada dentro ddel servlet SimpleSearch.java

En el método post llamamos al método `simpleSearch()` de la clase productManager, este método crea y lanza una sentencia JPA con el título, descripción de busqueda contra la base de datos y devuelve los productos resultado de la consulta.

Una vez tenemos los productos, en la clase SimpleSerarch.java redireccionamos a `products.jsp.`

Búsqueda avanzada

Esta funcionalidad queda implementada dentro del servlet AdvancedSearch.java

En el método post llamamos al método `advancedSearch()` de la clase productManager con el título, descripción y categoría introducidos por el usuario. Este método crea y lanza una sentencia JPA con los parámetros título, descripción y categoría contra la base de datos y devuelve los productos resultado de la consulta.

Una vez tenemos los productos, en la clase advancedSerarch.java redireccionamos a `products.jsp.`

Ver información de un producto

Esta funcionalidad queda implementada en el servlet GetProducts.java.

En el método post llamamos al método `getProductId()` de la clase ProductController, este método llama a `getProductId()` de la clase ProductDao que lanza una sentencia SQL (acceso JDBC) en la base de datos para recuperar el producto con ese ID, almacena los datos recuperados en un nuevo objeto ProductVo.

En la clase GetProducts.java redireccionamos a `modifyProduct.jsp`

Añadir producto carrito de la compra

Esta funcionalidad queda implementada en el servlet AddCart.java

En el método post llamamos al método `findProductId()` de la clase productManager para obtener el id del producto mediante una sentencia JPA. A continuación, recuperamos el carrito de la compra (es una lista de productos) y añadimos a la lista el producto cuyo id hemos recuperado, seteamos el carrito para guardar los cambios y redireccionamos a products.jsp

Eliminar producto del carrito de la compra

Esta funcionalidad queda implementada en el servlet DeleteCart.java.

En el método post llamamos al método `findProductId()` de la clase productManager para obtener el id del producto mediante una sentencia JPA. A continuación, recuperamos el carrito de la compra (es una lista de productos) y buscamos en la lista el producto cuyo id hemos recuperado, una vez encontrado, lo eliminamos de la lista y seteamos el carrito para guardar los cambios. Por último, redireccionamos a products.jsp

Compra

Esta funcionalidad queda implementada en el servlet Checkout.java.

En el método post recuperamos el carrito (lista de productos) asociado al usuario, mientras queden productos en la lista los vamos recuperando y marcando como vendidos usando un bucle. Esto se hace creando en cada iteración un objeto SoldProductVo nuevo por cada elemento del carrito, una vez creado ese objeto se llama al método `addSoldProduct()` de la clase SoldProductDao que registra ese producto como nuevo producto vendido en la base de datos (acceso JDBC), a continuación, se llama al método `modifyProduct()` de la clase ProductController con el id del producto que se está marcando como vendido para que modifique el producto original en la base de datos.

Consultar compras realizadas

Esta funcionalidad queda implementada en el servlet MyPurchases.java.

En el método post llamamos al método `getMyPurchases()` de la clase productManager que lanza una sentencia JPA en la base de datos usando el id del usuario para recuperar sus productos comprados, estos productos son almacenados en una lista. Una vez realizado se redirecciona a `myPurchases.jsp.`

Comunicación

Para la implementación de la comunicación entre dos usuarios es necesario crear dos Servlets, uno para escribir mensajes (productor) y otro para leerlos (consumidor).

De acuerdo a nuestra implementación los mensajes no son persistentes una vez leídos, es decir, cuando un mensaje es consumido este desaparece si recargamos la página.

Comunicación con otros usuarios a través del chat

Envío de mensajes:

En el método `doPost()` del servlet MessageController obtenemos el id del emisor y el del receptor, atributos que usaremos para saber quién envía y quién recibe el mensaje, guardamos en un objeto mensaje el mensaje que se nos ha pasado como parámetro. A continuación, inicializamos la connectionFactory y la cola, inicializamos el productor de mensajes para que se encolen. Enviamos el mensaje y cerramos el productor y las colas.

Por último, redirigimos a `mensajes2.jsp`

Lectura de mensajes:

En el método `doPost()` del servlet ReadController creamos el buffer de mensajes, y recuperamos el id del usuario a cuyo chat queremos acceder.

A continuación, recuperamos la cola, la connectionFactory y el mensaje, a continuación, mostramos el mensaje, cerramos la conectionFactory y las colas y redirigimos a `mensajes2.jsp.`

Funcionalidades del administrador

Estas funcionalidades quedan recogidas dentro de p1_admin.war

Modificar usuario

Esta funcionalidad queda implementada en el servlet ModifyUser.java.

En el método post creamos un nuevo objeto usuario con los atributos introducidos por el administrador, a continuación, llamamos al método `modifyUser()` de la clase AdminController, que a su vez llama al método `modifyUser()` de la clase AdminDao. Este método prepara y lanza la siguiente sentencia SQL (acceso JDBC) `"UPDATE user SET name=?, surname=?, email=?, passwd=?, address=? WHERE ID=?"`, se presupone que en caso de que no se introduzca una nueva contraseña, no se modifica la previa `"UPDATE user SET name=?, surname=?, email=?, address=? WHERE ID=?"`. Estas sentencias actualizan los datos del usuario administrador en la base de datos.

Dar de baja a un usuario

Esta funcionalidad queda implementada en el servlet DeleteUser.java.

Esta funcionalidad queda recogida en la clase DeleteUser.java. En el método post creamos un nuevo objeto usuario y llamamos al método `deleteUser()` de la clase UserController, este a su vez llama al método `deleteUser()` de la clase UserDao. Este método lanza una sentencia JPA contra la base de datos para eliminar al usuario cuyo id hemos obtenido. Una vez se ha borrado al usuario, volvemos a la clase DeleteUser.java, invalidamos la sesión actual del usuario y lo redirigimos a la página principal.

Modificar un producto

Esta funcionalidad queda implementada en el servlet ModifyProduct.java.

En el método post almacenamos temporalmente los nuevos parámetros de la modificación, creamos un objeto ProductVo, con esos parámetros y llamamos al método `modifyProduct()` de la clase ProductController, que a su vez llama al método `modifyUser()` de la clase ProductDao .

Para modificar un producto (método `modifyUser()`), nos creamos una nueva conexión, preparamos la consulta SQL (acceso JDBC) “`UPDATE product SET title=?, category=?, description=?, image=?, price=? WHERE ID=?`” y a continuación lanzamos la consulta y actualizamos los atributos del producto con los nuevos introducidos por el usuario

Dar de baja un producto

Esta funcionalidad queda implementada en el servlet DeleteProduct.java.

Almacenamos en un objeto ProductJPA el producto que deseamos borrar, que recuperamos usando el método `findProductId ()` de la clase ProductManager. Si el usuario tiene un producto con el id que se desea borrar llamamos al método `deleteProduct` () de la clase ProductManager que elimina el producto de la base de datos mediante una sentencia JPA. Si el producto ha sido borrado correctamente redireccionamos a `"myProducts?delete=success".`

Comunicación con otros usuarios

Esta funcionalidad permite enviar mensajes únicamente entre usuarios administradores, para simplificar la comunicación entre administradores.

Envío de mensajes:

En el método `doPost()` del servlet MessageController obtenemos el id del emisor y el del receptor, atributos que usaremos para saber quién envía y quién recibe el mensaje, guardamos en un objeto mensaje el mensaje que se nos ha pasado como parámetro. A continuación, inicializamos la connectionFactory y la cola, inicializamos el productor de mensajes para que se encolen. Enviamos el mensaje y cerramos el productor y las colas.

Por último, redirigimos a `mensajes2.jsp`

Lectura de mensajes:

En el método `doPost()` del servlet ReadController creamos el buffer de mensajes, y recuperamos el id del usuario a cuyo chat queremos acceder.

A continuación, recuperamos la cola, la connectionFactory y el mensaje, a continuación, mostramos el mensaje, cerramos la conectionFactory y las colas y redirigimos a `mensajes2.jsp.`

Información de la base de datos

Nuestra base de datos consta de 3 tablas Usuario (user), productos vendidos (sold_products) y productos (product).

La tabla user está formada por los siguientes campos:

Campo	Descripción	Tipo:
Id	Identificador unívoco de usuario en el sistema.	Int
Name	Nombre del usuario.	Char
Surname	apellido del usuario.	Char
Email	Email del usuario	Char
Password	Contraseña del usuario	Char
Adress	Dirección del usuario	Char
Is_admin	Indicador de si es administrador o no.	Int

La tabla sold_products está formada por los siguientes campos:

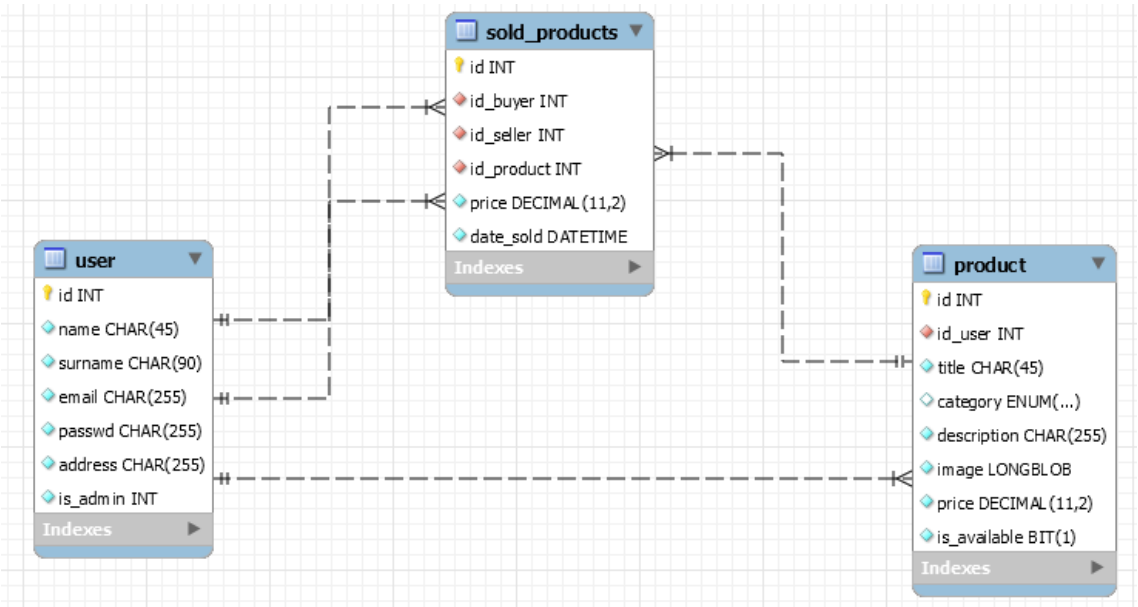
Campo	Descripción	Tipo:
Id	Id que identifica al producto que se ha vendido	Int
Id_buyer	Id que identifica al comprador	Int
Id_seller	Id que identifica al vendedor	Int
Id_product	Id que identifica al producto vendido en la tabla producto	Int

price	Precio por el que el producto ha sido vendido	Decimal
Date_sold	Fecha de la venta	Datetime

La tabla product está formada por los siguientes campos:

Campo	Descripción	Tipo:
Id	Id que identifica al producto	Int
Id_user	Id que identifica al vendedor	Int
title	Título del artículo	Char
category	Categoría del artículo	Enum
description	Descripción del artícuo	Char
image	Imagen del artículo	LongBL OB
price	Precio del artículo	Decimal
Is_avaiable	Indicador de si está disponible o no.	Bit

Una vez explicadas las tablas, el grafo relacional sería este:



Conclusiones

Esta práctica nos ha sido útil para afianzar los conocimientos vistos y aprendidos en clase. Para su realización hemos necesitado invertir una gran cantidad de tiempo, tanto para desarrollar el proyecto como para buscar la información necesaria para saber implementar sus funcionalidades ya que hay mucha información que sintetizar e implementar. Además, nos costó un poco hacer funcionar el entorno entero al principio y a mitad de la práctica fue

necesario realizar una migración completa del proyecto a eclipse ya que en Spring Tools Suite no llegaba a funcionar correctamente.
En general, nos ha resultado una práctica muy compleja de elaborar debido al alto número de elementos y recursos que se necesitan para crear una aplicación web tan completa y con tantas funcionalidades como la requeridas.