SDI – Sistemas Distribuidos e Internet

PRÁCTICA 1 DE ENTREGA – SPRING

INFORME

Grupo 601-610

URL AWS	**http://ec2-35-180-43-52.eu-west-3.compute.amazonaws.com:8090**
Nombre1:	**Othmane**
Apellidos1:	**Bakhtaoui**
Email1:	**UO259323@uniovi.es**
Cód. ID GIT	**601**
URL GIT	**github.com/othub/sdi-entrega1-601-610/**
Nombre2:	**Vladislav**
Apellidos2:	**Stelmakh**
Email2:	**UO257580@uniovi.es**
Cód. ID GIT	**610**

Índice

Introducción

El trabajo entregado ha consistido en la elaboración de una aplicación/página web inspirada en el tan famoso "Wallapop", una aplicación desarrollada con el fin de poder hacer compras y ventas de manera online de cualquier tipo de producto, existiendo a su vez una interacción por medio de mensajes entre vendedor y comprador.

Seguridad

A ambos usuarios se les permite el acceso mediante una autenticación previa, ya que sin ella no se puede acceder a la aplicación, ya sea identificándose, debido a que ya tienen una cuenta creada, o sin embargo, registrándose como nuevos usuarios.

Hay distintos tipos de usuarios, diferenciados por su **ROLE -> ROLE_STANDARD (usuarios normales) y ROLE_ADMIN (administrador)**. Dependiendo del rol que tengas, puedes visualizar y realizar unas acciones u otras. Si tienes rol de administrador solamente puedes gestionar usuarios, y si tienes rol standard, puedes gestionar tus productos, tus mensajes…

Finalmente, cabe destacar que no se puede registrar el correo admin@email.com ya que es propio al administrador.

Productos

Estos pueden anunciar sus productos de manera gratuita **"Gestión de ofertas" -> "Agregar ofertas",** o si lo desean, pueden destacar cualquiera de sus productos aportando una cantidad de 20€ por ello. También pueden ver sus ofertas personales **"Gestión de ofertas" -> "Ver ofertas personales"**, desde donde también podrán destacar una oferta, si es que no han querido hacerlo previamente, o borrar alguna de sus ofertas. Por último, también podrán ver sus ofertas compradas **"Gestión de ofertas" -> "Ofertas compradas"**.

Todos los productos serán mostrados en la ventana principal, y en diversas páginas, gracias a la implementación de la paginación, tras haber accedido con sus datos personales a la aplicación, los cuáles le serán mostrados en esta vista. Los usuarios podrán ver toda la información de dichos productos; nombre del producto, una pequeña descripción, el precio de dicho producto y la fecha en la que este ha sido anunciado. Es en este aparatado donde podrán buscar en el "navegador" ofrecido y comprar los productos que deseen, y en el que además, verán que existen una serie de productos destacados, que en este caso serán los primeros en aparecer en la lista, ya que esta es la ventaja que se les ofrece a aquellas personas que quieran destacar sus productos.

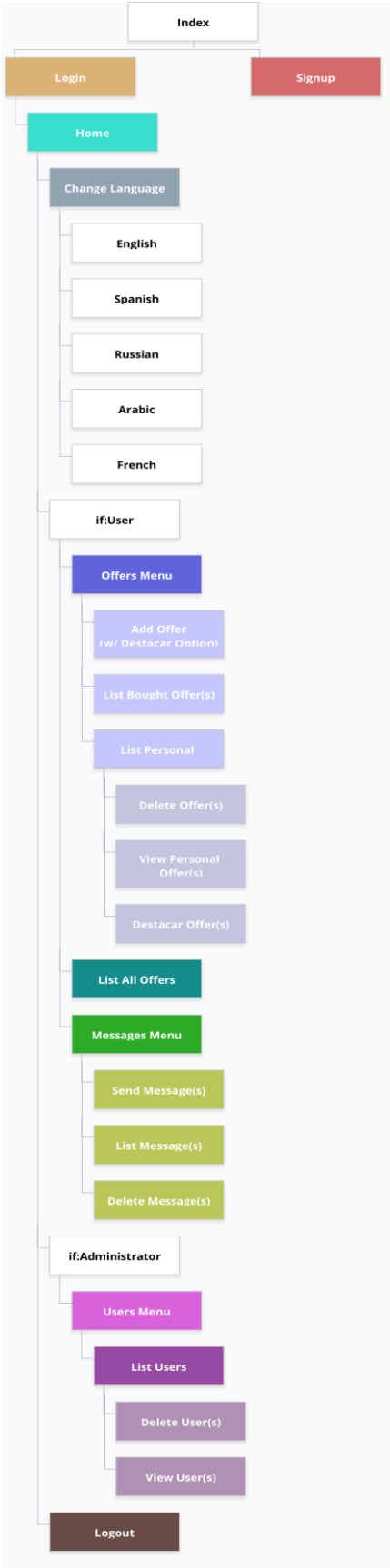
Mensajes

Por otro lado, si un usuario desea mandarle un mensaje a un vendedor para poder preguntarle algo acerca de un producto, puede hacerlo en la barra de navegación superior **"Mensajes" -> "Enviar mensajes"**, donde a continuación podrá elegir el producto por el que quiere preguntar y el mensaje que desea enviar. También le damos la posibilidad de que pueda ver todos sus mensajes **"Mensajes" -> "Ver conversaciones"** y que además pueda borrarlos si así lo desea **"Mensajes" -> "Borrar mensajes"**.

Internacionalización

Finalmente, podemos acabar comentando que todos los usuarios, en la barra de navegación superior, poseen tres opciones más a su derecha; pueden ver el saldo que tienen en su cuenta, pueden desconectarse de la sesión, y además pueden cambiar el idioma de la página, gracias a la implementación que hemos hecho de la internacionalización de la aplicación; les hemos ofrecido a los usuarios la posibilidad de poder ver e iterar por la aplicación en 5 idiomas diferentes (**inglés, español, ruso, francés y árabe**).

Mapa de navegación



Aspectos técnicos y de diseño relevantes

En cuanto a la **parte técnica** y de diseño de la aplicación, cabe destacar que hemos seguido lo visto en clase, y hemos aplicado el modelo MVC (Modelo Vista Controlador), un patrón de arquitectura software, que separa los **datos** y la **lógica de negocio** de una aplicación de su **representación** y **el modelo encargado de gestionar los eventos y las comunicaciones.** Para ello MVC propone la construcción de tres **componentes** distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por el otro lado para la interacción del usuario, todo esto mediante la **reutilización del código** y la **separación de conceptos**.

Podemos añadir que hacemos uso de **servicios**, **repositorios**, **controladores**… Por ejemplo:

Para el apartado de los productos hemos usado **OffersRepository**, donde hemos definido métodos que nos permiten obtener todos los productos, los productos de un usuario, buscar productos por su nombre, actualizar la disponibilidad de estos, o hacer que estén destacados, o incluso "organizarlos" para poder ver en la lista de productos, primero los productos destacados.

También hemos hecho uso de **OffersService** donde hemos definido los métodos que nos han sido necesarios para obtener, añadir y borrar ofertas… obtener las ofertas asociadas a mensajes en concretos, ofertas asociadas a usuarios, ofertas disponibles y actualización de ofertas al estado de destacadas.

Finalmente, en cuanto a **OffersController**, lo hemos utilizado para la implementación de los **@RequestMapping**, los cuáles nos permitían obtener ofertas, añadirlas, destacarlas, borrarlas, listarlas y hacer que estuviesen disponibles.

Algo a destacar sería que disponemos de comentarios que nos dicen que hace cada método, y sobretodo que hemos implementado el borrado múltiple.

En cuanto a la **parte de diseño**, podemos hablar de la organización en diversas carpetas y subcarpetas que tenemos.

- En la carpeta **static:**
  - Tenemos los **.css**; uno para la personalización y otro para los errores.
  - Disponemos de otra carpeta **img** donde almacenamos las imágenes de los idiomas y de los logos.
  - Finalmente tenemos una carpete **script** donde tenemos un javascript para el cambio de idioma.
- En la carpeta **templates**:
  - Tenemos los **html** de las páginas de **inicio**, **login**, **singup** e **index**.
  - Y diversas subcarpetas:
    - **Error:** para los errores que podemos obtener al intentar comprar o borrar un producto cuándo no podemos, o al efectuar el pago para destacarlo sin tener dinero.
    - **Fragments:** donde tenemos la cabecera, la barra de navegación, el pie de página y la paginación
    - **Message:** donde tenemos los archivos para añadir mensajes, borrarlos y obtener una lista.
    - **Offer:** donde tenemos los archivos para añadir una oferta, ver las ofertas compradas y listar las que tenemos.
    - **User:** donde tenemos la lista de usuarios, que nos permite el borrado de estos.

Finalmente, tenemos todos los archivos de propiedades de mensajes, donde almacenamos todos los mensajes de cada idioma que ofrecemos para la aplicación.

Un comentario sobre los casos de mensajes :

En **message/add**: el usuario selecciona a través del **<select>** la oferta que quiere y le pasa un mensaje y se actualiza la tabla de mensajes enviado en la propia vista.

En **message/delete**, el usuario selecciona los mensajes multiples que quiere borrar.

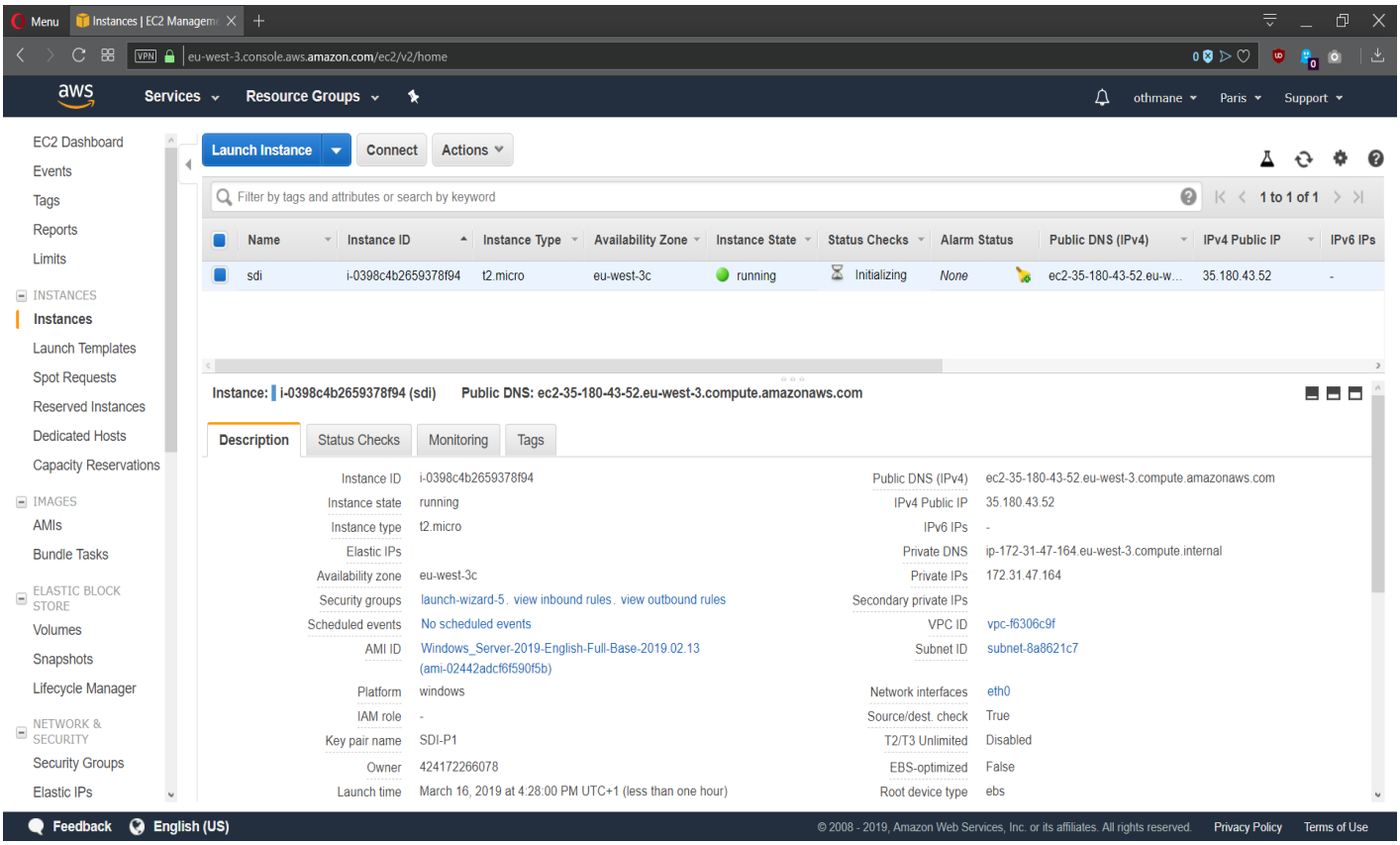
En **message/list** : queríamos hacer listar conversaciones agrupándolo por ofertas pero nos salieron muchos errores. Al final, decidimos que aparecen los mensajes en si ordenándolos por tiempo de envío y la oferta.

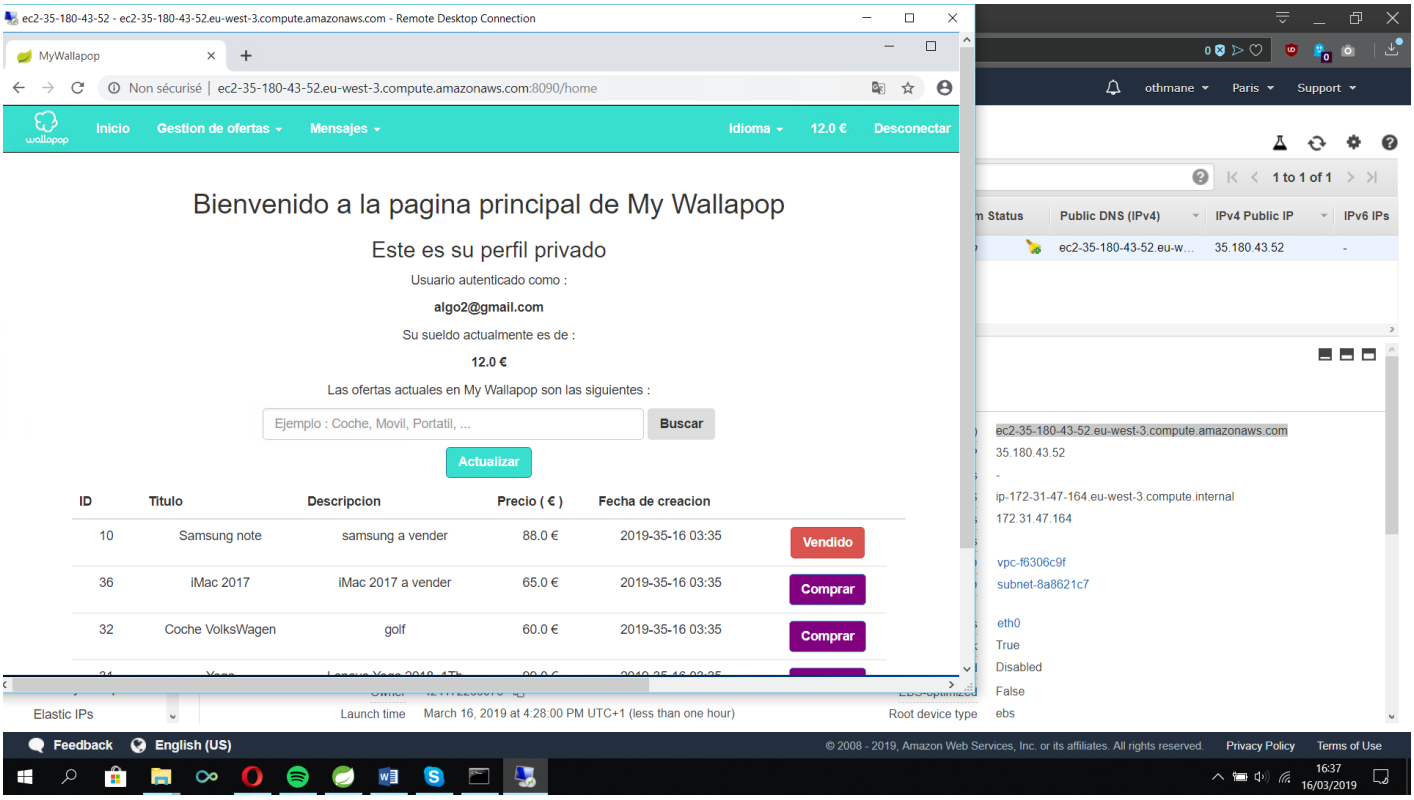
Un comentario sobre los **botones** de **comprar y destacar** :

Normalmente al hacer click en estos botones, se debe redirigir a una página de error como en el caso de borrar el usuario administrador como admin (**error/deletion**), pero no sabemos porque no lo hace. Al final, al hacer click, y no tenemos saldo suficiente o queremos comprar nuestra propia oferta, actualiza la página y todo es correcto.

Información necesaria para el despliegue y ejecución

➔ Prueba de funcionalidad en Amazon AWS EC2





➔ Comentarios :

La aplicación funciona y hace lo que tiene que hacer en el puerto **8090**.

Para los tests, son dependientes entre si.(más info en el fichero **excel**).

Los tests están en la clase (**MyWallapopAppTests.java**) y todos están en verde y tienen el **@PostConstruct comentado y detallado** y se pueden ejecutar normalmente.

Y como extra para no recorrer las más de 1000 lineas de codigo de prueba, les hemos proporcionado los tests individualmente para mirar cada uno de detalle. Y si quieren ejecutarlos todos, están agrupados en **Alltests.java y si se quiere ejecutar alguno de ellos en el caso de ya ejecutando MyWallapopAppTests.java** mejor parar la app y volver a ejecutarla para que no haya conflictos. **Esto es opcional** porque son las mismas pruebas que en **MyWallapopAppTests.java** pero cada caso de prueba en una clase**.**