myWallapop

Yago García Rodríguez - UO257178 - IDGIT: 303



Daniel Fernández Romeral - UO UO257461 - IDGIT:302

Indice

myWallapop 1

Indice 2

myWallapop 3

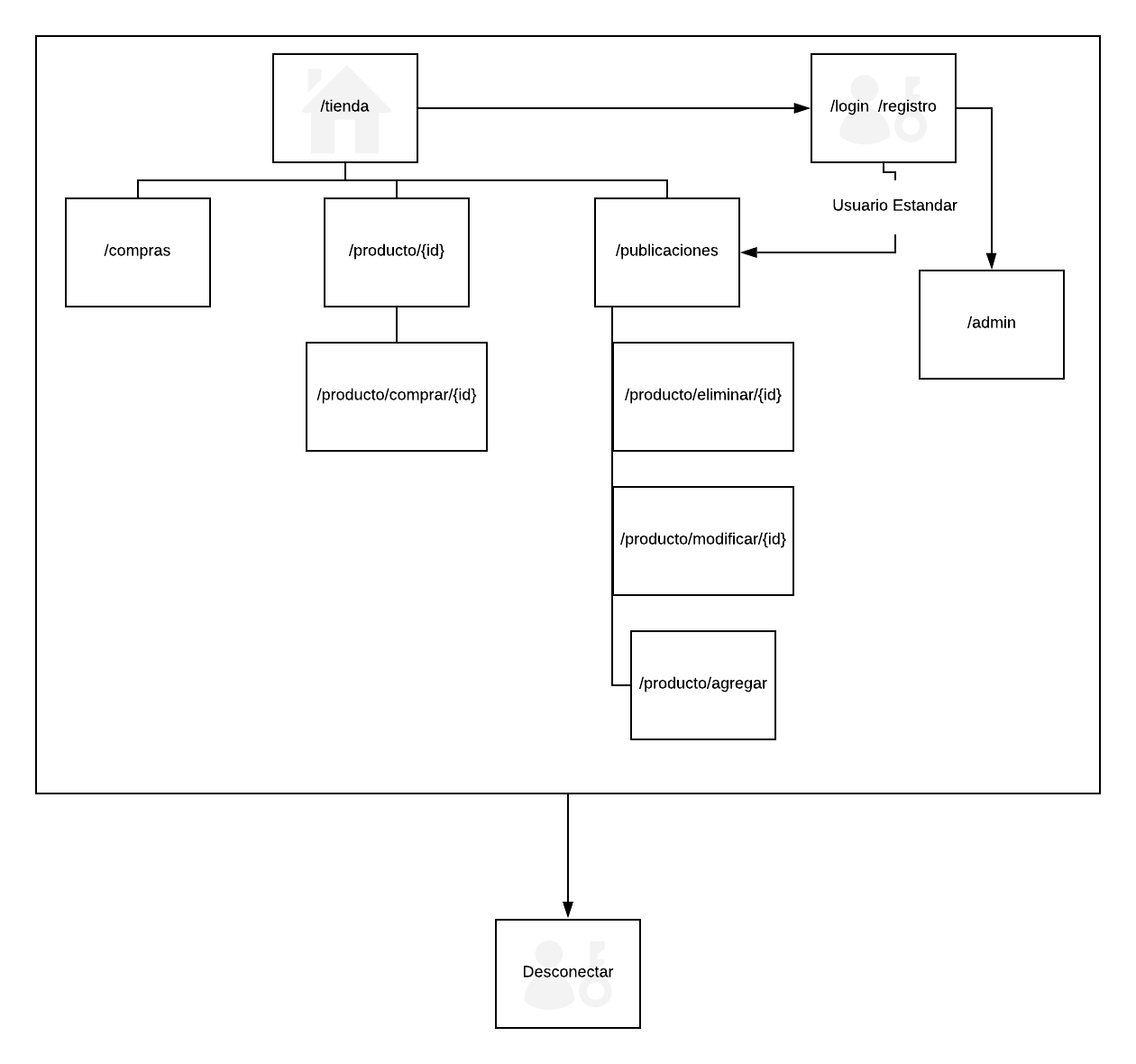
Casos de Uso 4

Pruebas Unitarias: 5

Anotaciones: 7

Problemas y soluciones: 10

myWallapop

Mapa de Navegación

Casos de Uso

Se han implementado todos los casos de uso obligatorios de la practica, tanto los de la parte de Node, REST y Api REST JQuery.

**PARTE1 – APLICACIÓN WEB**

1. PÚBLICO: REGISTRARSE COMO USUARIO . - Hecho

2. USUARIO REGISTRADO: INICIO DE SESIÓN . - Hecho

3. USUARIO REGISTRADO: FIN DE SESIÓN . - Hecho

4. ADMINISTRADOR: LISTADO DE USUARIOS . - Hecho

5. ADMINISTRADOR: BORRADO MÚLTIPLE DE USUARIOS. . - Hecho

6. USUARIO: DAR DE ALTA UNA NUEVA OFERTA . - Hecho

7. USUARIO: LISTADO DE OFERTAS PROPIAS . - Hecho

8. USUARIO: DAR DE BAJA UNA OFERTA . - Hecho

9. USUARIO: BUSCAR OFERTAS . - Hecho

10. USUARIO: COMPRAR UNA OFERTA . - Hecho

11. USUARIO: VER EL LISTADO DE OFERTAS COMPRADAS. - Hecho

12. OPTATIVO: MARCAR UNA OFERTA COMO DESTACADA. - Sin hacer

**PARTE 2A – IMPLEMENTACIÓN API REST**

1. IDENTIFICARSE CON USUARIO VÍA TOKEN. - Hecho

2. USUARIO: MOSTRAR LISTADO DE OFERTAS DISPONIBLES (DE OTROS USUARIOS). - Hecho

3. USUARIO: ENVIAR MENSAJES A UNA OFERTA . - Hecho

4. USUARIO: MOSTRAR MENSAJES DE UNA CONVERSACIÓN. - Hecho

5. OPTATIVO: MARCAR MENSAJE COMO LEÍDO . - Sin hacer

6. OPTATIVO: ELIMINAR UNA CONVERSACIÓN . - Hecho

**PARTE 2B – CLIENTE REST – APLICACIÓN WEB CON JQUERY**

7. AUTENTICACIÓN DEL USUARIO. - Hecho

8. MOSTRAR LISTADO DE OFERTAS DISPONIBLES - Hecho

10. ENVIAR Y VISUALIZAR MENSAJES A UNA OFERTA. - Hecho

10. OPTATIVO: VER EL LISTADO DE CONVERSACIONES . - Sin hacer

11. OPTATIVO: ELIMINAR UNA CONVERSACIÓN. - Hecho

12. OPTATIVO: MARCAR MENSAJES COMO LEÍDOS DE FORMA AUTOMÁTICA . - Sin hacer

13. OPTATIVO: MOSTRAR EL NÚMERO DE MENSAJES SIN LEER. - Sin hacer.

Pruebas Unitarias:

Se han realizado todas las pruebas para las funcionalidades marcadas como “Hecho” en el apartado anterior. Las pruebas requieren que la base de datos este vacía antes de iniciar la ejecución de test, ya que la mayoría de los mismos requieren de numero de objetos en base de datos y si no es el numero exacto, estos, fallarian. Si por alguna razón fallan los test, rogamos que se pongan en contacto con nosotros para vaciar la base de datos.

La mayoría de los test excepto los realizados para el cliente JQuery son una adaptación de los realizados con anterioridad para la actividad de Spring. Algunos de los test usan XPath absolutos ya que nos resultaba más rápido que utilizar los métodos de utilidades entregados. La extensión que hemos utilizado para determinar la ruta del XPath es “xPath Finder” disponible para la mayoría de navegadores.

Como no se puedo hacer el listado de conversaciones abiertas, en vez de eso se añadió un botón para borrar la conversación dentro de la propia conversación, y se comprobó que funcionara mediante pruebas unitarias.

**Lista de Tests:**

[Prueba1] Registro de Usuario con datos válidos.

[Prueba2] Registro de Usuario con datos inválidos (email vacío, repetición de contraseña inválida).

[Prueba3] Registro de Usuario con email existente.

[Prueba4] Inicio de sesión con datos válidos.

[Prueba5] Inicio de sesión con datos inválidos (email existente, pero contraseña incorrecta).

[Prueba6] Inicio de sesión con datos válidos (campo email o contraseña vacíos).

[Prueba7] Inicio de sesión con datos inválidos (email no existente en la aplicación).

[Prueba8] Hacer click en la opción de salir de sesión y comprobar que se redirige a la página de inicio de sesión (Login).

[Prueba9] Comprobar que el botón cerrar sesión no está visible si el usuario no está autenticado.

[Prueba10] Mostrar el listado de usuarios y comprobar que se muestran todos los que existen en el sistema.

[Prueba11] Ir a la lista de usuarios, borrar el primer usuario de la lista, comprobar que la lista se actualiza y dicho usuario desaparece.

[Prueba12] Ir a la lista de usuarios, borrar el último usuario de la lista, comprobar que la lista se actualiza y dicho usuario desaparece.

[Prueba13] Ir a la lista de usuarios, borrar 3 usuarios, comprobar que la lista se actualiza y dichos usuarios desaparecen.

[Prueba14] Ir al formulario de alta de oferta, rellenarla con datos válidos y pulsar el botón Submit. Comprobar que la oferta sale en el listado de ofertas de dicho usuario.

[Prueba15] Ir al formulario de alta de oferta, rellenarla con datos inválidos (campo título vacío) y pulsar el botón Submit. Comprobar que se muestra el mensaje de campo obligatorio.

[Prueba16] Mostrar el listado de ofertas para dicho usuario y comprobar que se muestran todas los que existen para este usuario.

[Prueba17] Ir a la lista de ofertas, borrar la primera oferta de la lista, comprobar que la lista se actualiza y que la oferta desaparece.

[Prueba18] Ir a la lista de ofertas, borrar la última oferta de la lista, comprobar que la lista se actualiza y que la oferta desaparece.

[Prueba19] Hacer una búsqueda con el campo vacío y comprobar que se muestra la página que corresponde con el listado de las ofertas existentes en el sistema

[Prueba20] Hacer una búsqueda escribiendo en el campo un texto que no exista y comprobar que se muestra la página que corresponde, con la lista de ofertas vacía.

[Prueba21] Hacer una búsqueda escribiendo en el campo un texto en minúscula o mayúscula y comprobar que se muestra la página que corresponde, con la lista de ofertas que contengan dicho texto, independientemente que el título esté almacenado en minúsculas o mayúscula.

[Prueba22] Sobre una búsqueda determinada (a elección de desarrollador), comprar una oferta que deja un saldo positivo en el contador del comprobador. Y comprobar que el contador se actualiza correctamente en la vista del comprador.

[Prueba23] Sobre una búsqueda determinada (a elección de desarrollador), comprar una oferta que deja un saldo 0 en el contador del comprobador. Y comprobar que el contador se actualiza correctamente en la vista del comprador.

[Prueba24] Sobre una búsqueda determinada (a elección de desarrollador), intentar comprar una oferta que esté por encima de saldo disponible del comprador. Y comprobar que se muestra el mensaje de saldo no suficiente.

[Prueba25] Ir a la opción de ofertas compradas del usuario y mostrar la lista. Comprobar que aparecen las ofertas que deben aparecer.

**Rest:**

[Prueba29] Inicio de sesión con datos válidos.

[Prueba30] Inicio de sesión con datos inválidos (email existente, pero contraseña incorrecta).

[Prueba31] Inicio de sesión con datos válidos (campo email o contraseña vacíos).

[Prueba33] Sobre una búsqueda determinada de ofertas (a elección de desarrollador), enviar un mensaje a una oferta concreta. Se abriría dicha conversación por primera vez. Comprobar que el mensaje aparece en el listado de mensajes.

[Prueba34] Sobre el listado de conversaciones enviar un mensaje a una conversación ya abierta. Comprobar que el mensaje aparece en el listado de mensajes.

[Prueba36] Sobre el listado de conversaciones ya abiertas. Pinchar el enlace Eliminar de la primera y comprobar que el listado se actualiza correctamente.

[Prueba37] Sobre el listado de conversaciones ya abiertas. Pinchar el enlace Eliminar de la última y comprobar que el listado se actualiza correctamente.

Anotaciones:

**Estructura de un Producto en la base de datos:**

{

nombre:,

descripcion:,

precio:,

vendedor: ,

imagen:,

fecha:

}

**Estructura de un usuario:**

{

email:,

nombre:,

apellidos:,

balance:,

password: seguro

}

**Estructura de una compra:**

{

usuario :,

productoId :

}

**Estructura de un mensaje:**

{

oferta: oferta,

usuario:,

conversacion : []

}

**Estructura de una conversación:**

{

texto:,

autor:,

time:,

leido:

}

Apis y parámetros:

Post: api/autentificar

Body:{email, password}

Get: api/producto

Header:{token}

Get: api/misproducto

Header:{token}

Get: api/idconversacion

Header:{token,id\_oferta,user}

Post: api/mensaje

Header:{token,id\_oferta }

Body:{mensaje,to}

Get: api/conversacion

Header:{token,id\_oferta }

Get: api/misconversaciones

Header:{token}

Get: api/mensajes

Header:{token,id\_oferta}

Get: api/eliminar

Header:{token,id\_conversacion}

Get: api/leido

Header:{token,id\_conversacion}

Problemas y soluciones:

Inicialmente las portadas de los productos usaban su propio router y se enviaban a una de las carpetas publicas. Debido a que algunos test requieren publicar ofertas y no encontramos la manera de subir un archivo a través de los test, decidimos retirar la portada. Finalmente la añadimos como un campo de texto donde insertar una url hacia una imagen de internet.

La aplicación tiene alguna funcionalidad extra como por ejemplo la de modificar coversaciones.

El usuario administrador del sistema es: "[admin@admin.com](mailto:admin@admin.com)" con contraseña: “admin”.

Tambien existen diversos usuarios regulares. Como :

“[prueba@prueba.com](mailto:prueba@prueba.com)" con contraseña “prueba”.

“[1@prueba.com](mailto:1@prueba.com)" con contraseña “1”.

“[12@prueba.com](mailto:1@prueba.com)" con contraseña “12”

“[123@prueba.com](mailto:1@prueba.com)" con contraseña “123”

“[1234@prueba.com](mailto:1@prueba.com)" con contraseña “1234”

….

Estos usuarios son creados al ejecutar las pruebas, la base de datos inicialmente está vacía para poder ejecutar las pruebas.

Para poder hacer bien el cliente del chat, se añadieron nuevos servicios Rest, como idconversacion, que devuelve el id de la conversación, o conversación, que devuelve los usuarios que han hablado a una oferta dada y misconversacions, que devuelve las ofertas a las que ha hablado el usuario.

Estos dos últimos son usados para listar las conversaciones en el cliente, funcionan perfectamente, pero en el cliente no se consiguió mostrarlas bien, debido a una serie de consultas en Ajax y unos bucles for no se pudo encontrar porque fallaba.

Para marcar los mensajes como leído se implementó el servicio leido, pero al tener que modificar un campo dentro de un objeto dentro de un array dentro de un objeto, no fuimos capaces de lograr que funcionara. El código que hay actualmente, según la documentación, debería funcionar, pero no lo hace.

Para que el dueño de una oferta pudiera responder a los mensajes usando la api, se usa un campo opcional en mensaje, llamado to. Este campo solo se necesita llenar si es el dueño de la oferta, para especificar a quien estas respondiendo. Si un cliente lo rellenase, no afectaría en nada, puesto que por una serie de ifs, el campo se llenaría con su usuario, por lo que no tendría ningún impacto.

En el widget-mensajes, existe un error por el cual, aunque se envié el mensaje correctamente y se suba a la base de datos, el método seguía concluyendo como error y cerrando la sesión. Para solucionar esto, simplemente se cambió para que recargase los mensajes.