18-3-2018

Iván González Mahagamage

Universisdad de oviedo

Entregable 1

Sistemas distribuidos e internet

**Autores**: **Iván González Mahagamage**

Estudiante de Ingeniería Informática del Software en la Universidad de

**Fecha**: 18 de marzo de 2018

**Versión:** V1

Contenido

[Implementación 4](#_Toc512465136)

[0. Aspectos comunes 4](#_Toc512465137)

[0.1. Entidades de la aplicación. 4](#_Toc512465138)

[0.2. Creación automática de datos. 6](#_Toc512465139)

[0.3. Creación del log de actividad. 7](#_Toc512465140)

[0.4. Proceso de internacionalización. 9](#_Toc512465141)

[0.5. Externacionalización de las consultas. 9](#_Toc512465142)

[0.6. Roles de la aplicación 10](#_Toc512465143)

[0.7. Definición de permisos 11](#_Toc512465144)

[0.8. Utilidades 11](#_Toc512465145)

[1. Público: registrarse como usuario 12](#_Toc512465146)

[2. Público: iniciar sesión 17](#_Toc512465147)

[3. Usuario registrado: listar todos los usuarios de la aplicación 19](#_Toc512465148)

[4. Usuario registrado: buscar entre todos los usuarios de la aplicación 21](#_Toc512465149)

[5. Usuario registrado: enviar una invitación de amistad a un usuario 22](#_Toc512465150)

[6. Usuario registrado: listar las invitaciones de amistad recibidas 24](#_Toc512465151)

[7. Usuario registrado: aceptar una invitación recibida 25](#_Toc512465152)

[8. Usuario registrado: listar los usuarios amigos 27](#_Toc512465153)

[9. Usuario registrado: crear una nueva publicación 29](#_Toc512465154)

[10. Usuario registrado: listar mis publicaciones 31](#_Toc512465155)

[11. Usuario registrado: listar las publicaciones de un usuario amigo 32](#_Toc512465156)

[12. Usuario registrado: crear una publicación con una foto adjunta 34](#_Toc512465157)

[13. Público: iniciar sesión como administrador 36](#_Toc512465158)

[14. Consola de administración: listar todos los usuarios de la aplicación 38](#_Toc512465159)

[15. Consola de administración: Consola de administración: eliminar usuario 40](#_Toc512465160)

[Prueba Unitarias 41](#_Toc512465161)

[1.1. Registro de Usuario con datos válidos. 41](#_Toc512465162)

[1.2. Registro de Usuario con datos inválidos (repetición de contraseña invalida). 41](#_Toc512465163)

[2.1. Inicio de sesión con datos válidos. 42](#_Toc512465165)

[2.2. Inicio de sesión con datos inválidos. 42](#_Toc512465166)

[3.1. Acceso al listado de usuarios desde un usuario en sesión. 43](#_Toc512465168)

[3.2. Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión. 43](#_Toc512465169)

[4.1. Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión. 44](#_Toc512465171)

[4.2. Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado. 44](#_Toc512465172)

[5.1. Enviar una invitación de amistad a un usuario de forma valida. 45](#_Toc512465174)

[5.2. Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente. 46](#_Toc512465175)

[6.1. Listar las invitaciones recibidas por un usuario. 47](#_Toc512465177)

[7.1. Aceptar una invitación recibida. 47](#_Toc512465179)

[8.1. Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo. 48](#_Toc512465181)

[9.1. Crear una publicación con datos válidos. 48](#_Toc512465183)

[10.1. Acceso al listado de publicaciones desde un usuario en sesión. 49](#_Toc512465185)

[11.1. Listar las publicaciones de un usuario amigo. 49](#_Toc512465187)

[11.2. Utilizando un acceso vía URL tratar de listar las publicaciones de un usuario que no sea amigo del usuario identificado en sesión. 50](#_Toc512465188)

[12.1. Crear una publicación con datos válidos y una foto adjunta. 51](#_Toc512465190)

[12.2. Crear una publicación con datos válidos y sin una foto adjunta. 52](#_Toc512465191)

[13.1. Inicio de sesión como administrador con datos válidos. 52](#_Toc512465193)

[13.2. Inicio de sesión como administrador con datos inválidos. 53](#_Toc512465194)

[14.1. Desde un usuario identificado en sesión como administrador listar a todos los usuarios de la aplicación. 53](#_Toc512465196)

[15.1. Desde un usuario identificado en sesión como administrador eliminar un usuario existente en la aplicación. 54](#_Toc512465198)

[15.2. Intento de acceso vía URL al borrado de un usuario existente en la aplicación. 54](#_Toc512465199)

[Otros aspectos 55](#_Toc512465200)

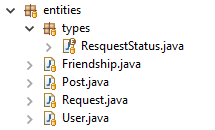
[1. Usuario registrado: bloquear invitaciones de amistad de un usuario 55](#_Toc512465201)

# Implementación

## Aspectos comunes

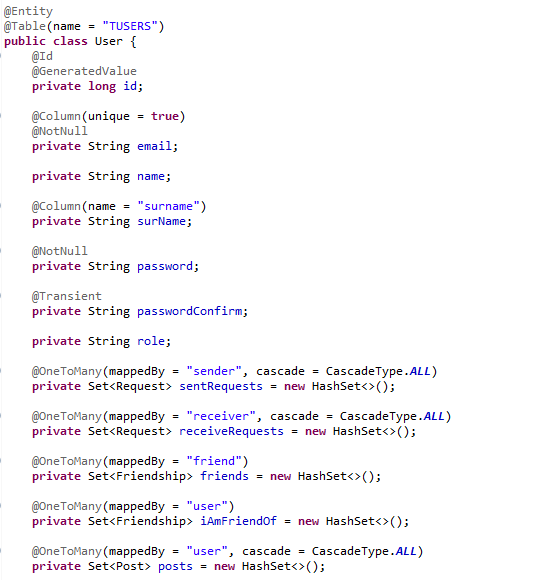
### Entidades de la aplicación.

Para la realización de la aplicación se ha diseñado una serie clases de objeto dentro del paquete “entities”.



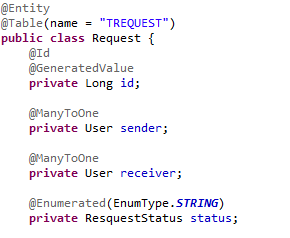
Todos ellas están configuradas con la tecnología JPA para manejar la base de datos de manera automática. Las clases creadas son:

* User: simula los usuarios que están dentro de la aplicación.



La información del usuario que se guarda es su email, es único y se usara para su identificación, nombre completo y contraseña. Además de las referencias a los otros objetos de la aplicación.

* Request: simula las peticiones de amistad que se envían entre sí.



En este objeto además de guardar una referencia al usuario que manda la petición y el que la recibe, se guarda el estado de esta, que se indicara con enumerable. Este puede ser enviada, aceptada o bloqueada.

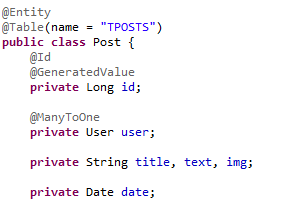


* Friendship: simula la amistad que existe entre los diferentes usuarios.



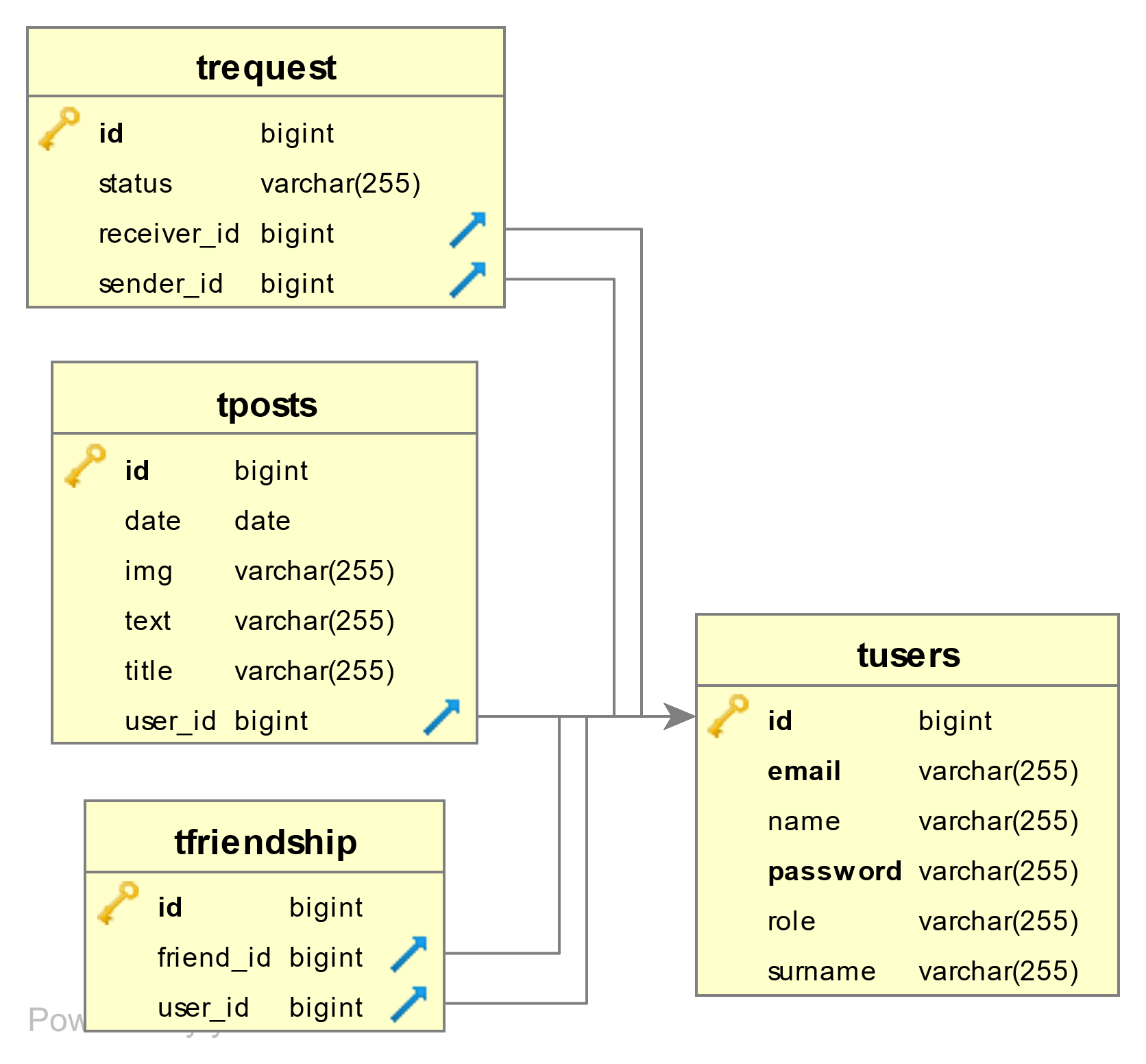
Esta clase ha sido creada debido a los problemas entre el mapeador y la paginación de lista, que se explicará más adelante en el documento. La manera elegante de crear las amistades hubiera sido en el atributo “friends” de la clase “User” haber puesto la etiqueta “ManyToMany” pero a la hora de hacer la paginación de resultados la aplicación fallaba, cuando había más de una página. Por este motivo se optó por crear esta clase

* Post: publicaciones que realizan los usuarios.



En esta clase se guarda una referencia al usuario que ha creado la publicación, el título y el contenido se guardan directamente, pero se guarda la referencia a la imagen, no la misma en sí, pero esto ya se detallara más adelante.

La base de datos resultante es la siguiente:



También hay que decir que cada una de estas clases tiene un controlador, un servicio y repositorio.

### Creación automática de datos.

Para tener datos para probar la aplicación se ha creado la clase “InsertSampleDataService” que tiene tres funciones:

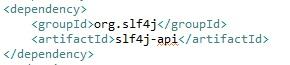
* Crea un usuario Admin: [ivangonzalezmahagamage@gmail.com](mailto:ivangonzalezmahagamage@gmail.com).
* Crea un usuario User: [igm1990@hotmail.com](mailto:igm1990@hotmail.com)
* Genera el número de usuarios aleatorios que le indicamos.

A continuación, añado un fragmento de esta clase en la que se muestra como realiza estas opciones en el “@PostContruct”

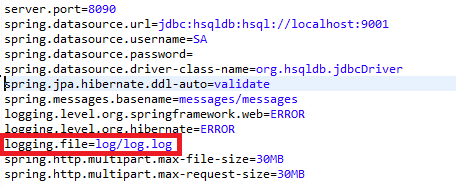


### Creación del log de actividad.

Como se indica en el enunciado, se creado una clase denominada “LogService” para crear un log de todos los sucesos que ocurren en la aplicación. Para ello se añadido la siguiente dependencia al archivo “pom.xml”.



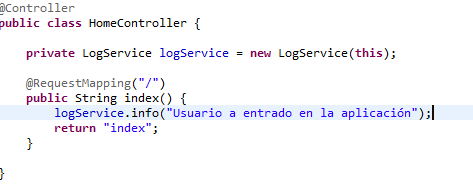
También en el fichero “aplicaction.properties” indicamos el fichero donde se guardará el log.



La implementación de esta clase es la siguiente:

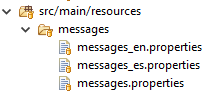


Se optado por esta manera para hacerla dinámica, únicamente tenemos que instanciarla en las clases que queremos registrar su actividad pasándole como parámetro “this” en su constructor. Un ejemplo:

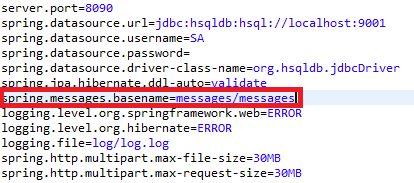


### Proceso de internacionalización.

Para el proceso de internacionalización se han creado los ficheros .messages como se indicó en clase únicamente con una ligera variación, su localización. En mi caso, los he colocado dentro de un directorio denominado “messages” dentro del directorio “resources”

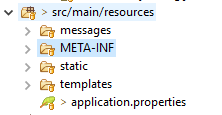


Para que la aplicación los reconozca, he tenido que hacer la siguiente modificación en el archivo “aplicaction.properties”.



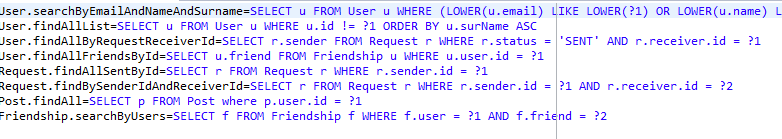
### Externacionalización de las consultas.

Este apartado no se ha dado en clase pero yo he decido externalizar las consultas tal como nos enseñaron en la asignatura de “Repositorios de la Información”. Para ello he creado un directorio denominado “META-INF” dentro del directorio “resource”.



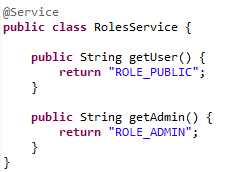
Dentro de este he creado un fichero denominado “jpa-named-queries.properties”, en el cual he añadido las diferente consultas que se usan en la aplicación. Para que esta funcione correctamente, se debe seguir la siguiente estructura:

* Nombre de la clase que devuelve la consulta.
* Nombre del método dentro del repositorio que referencia la consulta.
* Símbolo “=”
* Consulta



### Roles de la aplicación

He establecido dos roles para la aplicación “ROLE\_USER” para usuario sin privilegios especiales y “ROLE\_ADMIN” para usuario con privilegios de administrador. Esto se realizado mediante la creación de la clase “RolesService”.



### Definición de permisos

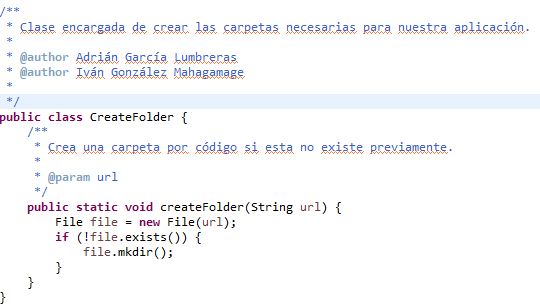
Para esta aplicación hemos definidos los siguientes permisos en la clase “WebSecurityConfig”.



Definimos que los css, imágenes, scripts, las pagina de inicio, registrase, login, y login del administrador pueda acceder cualquier usuario. También definimos que las url que sean de administrador solo pueden acceder los usuarios que tengan el rol “ADMIN\_USER”

### Utilidades

He reutilizado una clase denominada “createFolder” para crear directorios de manera dinámica para el log de la aplicación y el guardado de las imágenes subidas por los usuarios.



## Público: registrarse como usuario

Primero se ha creado una vista para este propósito denominada “signup.html”. De esta vista solo voy destacar ciertos aspectos ya que es demasiado código para ponerlo en este documento.

* Al formulario le insertamos un object para que, si el registro no es válido, guarde los datos que ha introducido el usuario a excepción de las contraseñas para evitar que los tenga que volver a introducir.



* A parte de introducir la internacionalización en todas las vistas, he creado un archivo en JavaScript para incluir validaciones por el lado del cliente además de incluir el atributo placeholder.



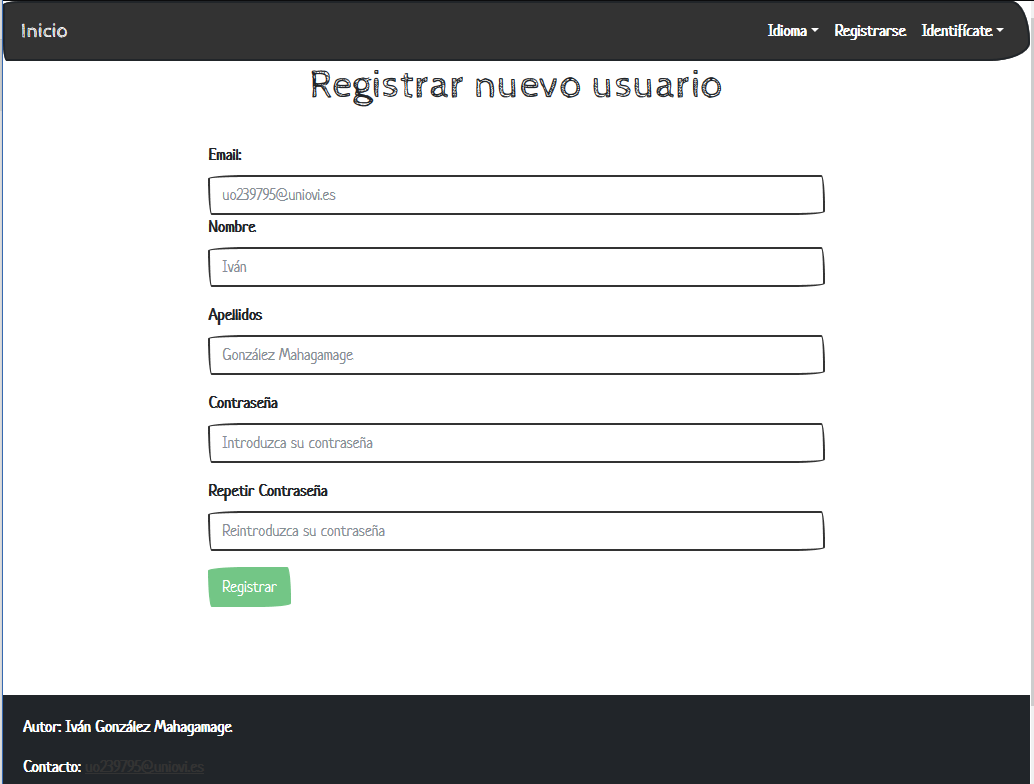
* El archivo JavaScript denominado anterior es “checkForm.js”.”



Comprueba que las dos contraseñas que introduce el usuario son iguales y hasta que el usuario no introduce todos los campos del formulario desactiva el botón de “Enviar”.

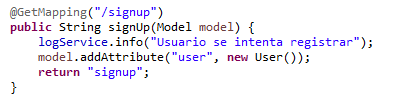


El resultado final de estos archivos es:

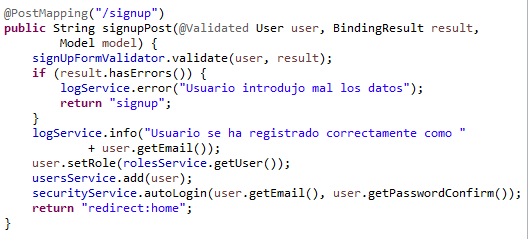


Para que la vista se muestre correctamente debemos incluir en “UserController” los siguientes métodos:

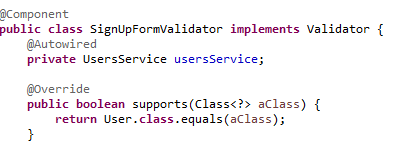
* Un método para recoja la petición Get para mostrar la vista.



* Un método para recoja la petición Post y procese el formulario. Al hacer el autologin usamos el parámetro “passwordConfirm” ya que este método necesita la contraseña sin codificar para funcionar correctamente.

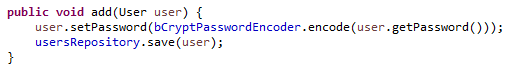


En este método incluimos una llamada a un validador para comprobarlo en lado del servidor.





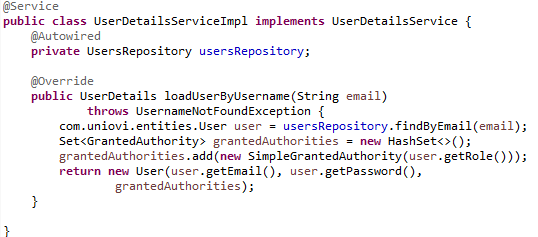
Si supera las comprobaciones se le asigna el rol “ROLE\_USER” y se llama a “UserService” para que lo añada a la base de datos y encripte su contraseña.



Una vez finalizado este proceso, la aplicación loguea automáticamente al usuario para que comience a usar la aplicación. Para ello también debemos implementar la clase “SecurityService”.



A su vez, esta clase llama a “UserDetailsServiceImpl”.



## Público: iniciar sesión

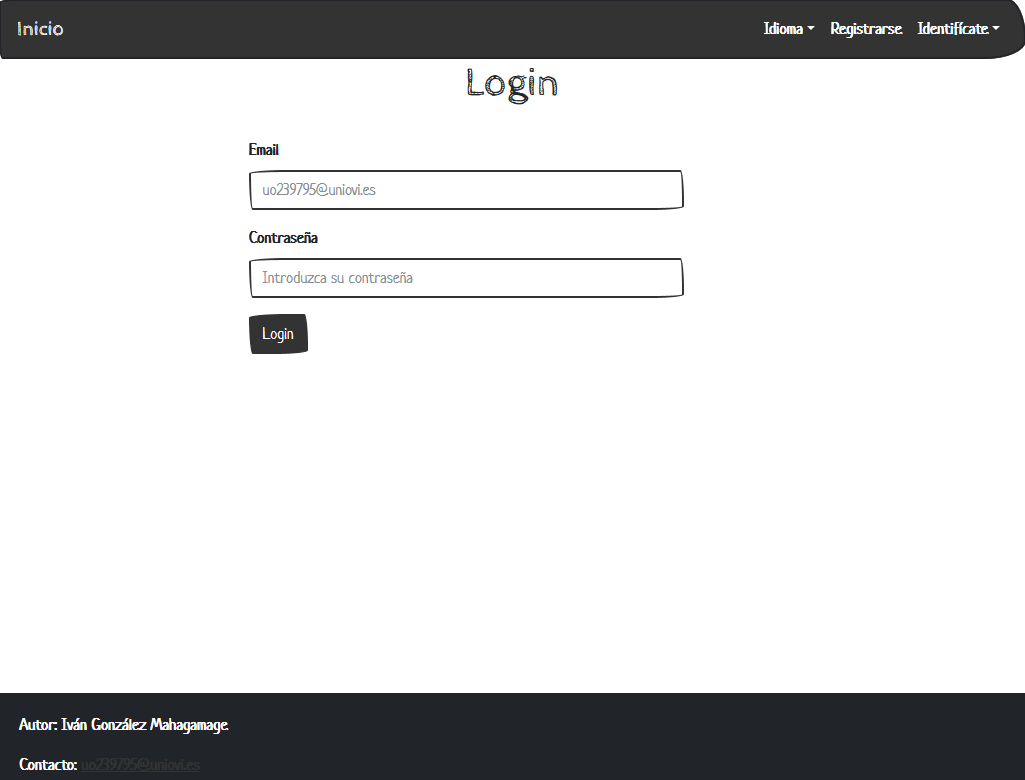
Se ha implementado la vista “login.html” para que el usuario pueda iniciar sesión.

Hay destacar un punto fundamental para que funcione correctamente esta vista.

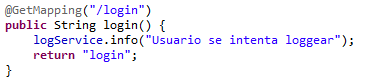
* Los inputs que se usan deben tener como atributo name “username” y “password” respectivamente para que los reconozca Spring Security sin problemas.



El resultado es el siguiente:



Se declarado un nuevo método “login()” en “UserController” para mostrar esta vista.



Cabe destacar que como se usa Spring Security no hay que implementar la petición post de la vista.

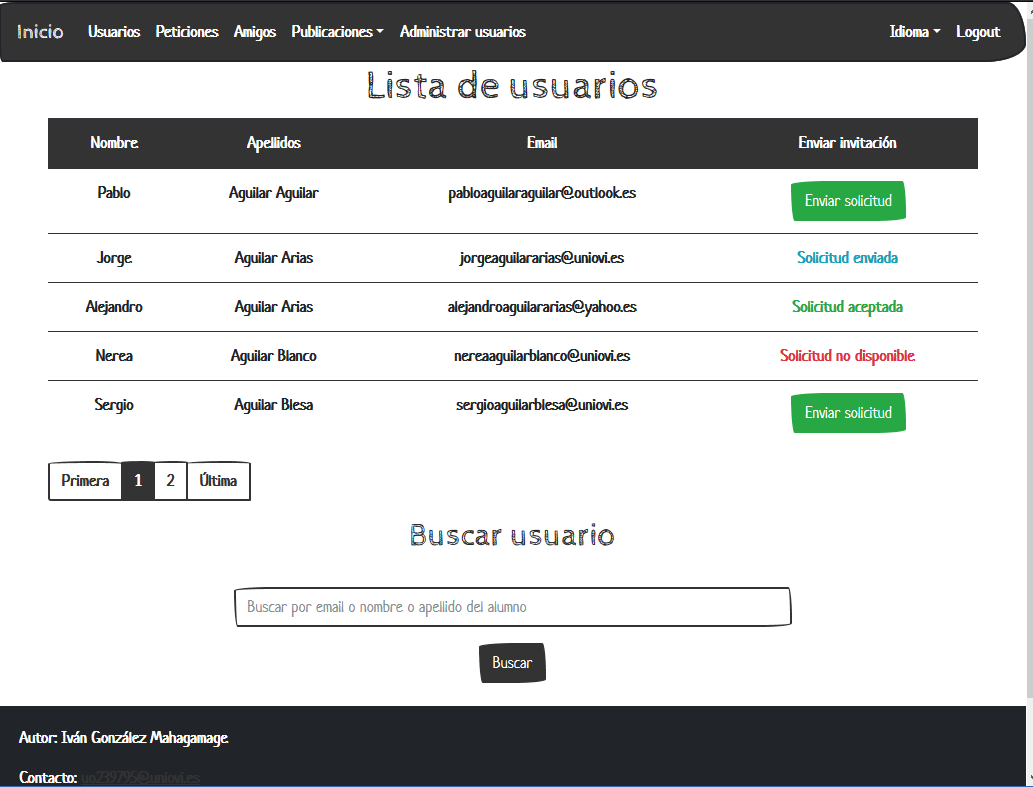
## Usuario registrado: listar todos los usuarios de la aplicación

Se ha creado la vista “list.html” dentro del directorio “users”.

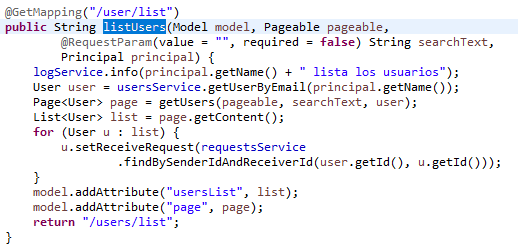


En esta vista hay elementos pertenecientes a otros casos de usos que se explicarán más adelante.

El resultado es el siguiente:

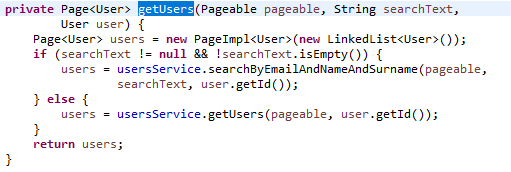


Se declarado un nuevo método “listUsers()” en “UserController” para mostrar esta vista.

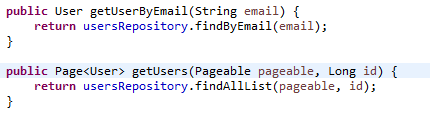


El bucle for que hay en el método se explicara en el caso de uso 5.

También se ha creado el método “getUsers()” en esta clase para diferenciar cuando se muestran todos los usuarios o se desea filtrar, en este caso de uso e ejecutaría la opción del else.



Para realizar estas búsquedas se han implementado los siguientes métodos en “UserServicies”



Y en la clase “UsersRepository” también se han añadido nuevos métodos.



Cuyas consultas son:

* User.findAllList=SELECT u FROM User u WHERE u.id != ?1 ORDER BY u.surName ASC

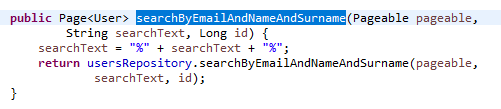
## Usuario registrado: buscar entre todos los usuarios de la aplicación

Para hacer este caso de uso se ha utilizado la vista “list.html” del directorio “users”, para ser más específicos el siguiente fragmento.



Cuando se ejecuta este formulario llama al método “listUsers()” en “UserController” pero indicando el parámetro “searchText” para realizar el filtro en el método “getUsers()”.

En “UsersService” se ha implementado un nuevo método “searchByEmailAndNameAndSurname” para realizar la nueva búsqueda.



Y en “UsersRepository” también se ha añadido un nuevo método “searchByEmailAndNameAndSurname”.

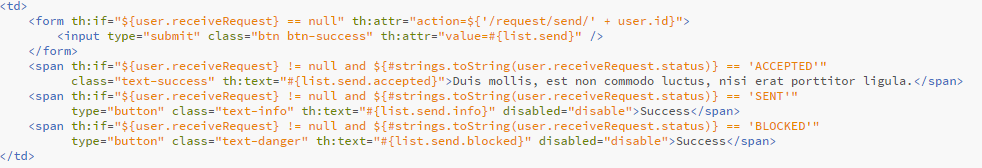


Cuya consulta es:

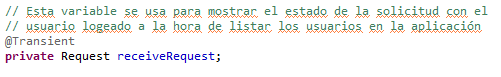
* User.searchByEmailAndNameAndSurname=SELECT u FROM User u WHERE (LOWER(u.email) LIKE LOWER(?1) OR LOWER(u.name) LIKE LOWER(?1) OR LOWER(u.surName) LIKE LOWER(?1)) AND u.id != ?2 ORDER BY u.surName ASC

## Usuario registrado: enviar una invitación de amistad a un usuario

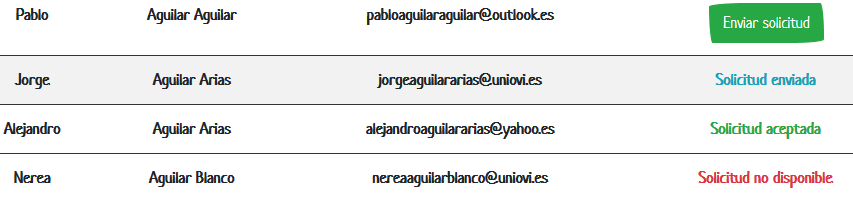
Para hacer este caso de uso se ha utilizado la vista “list.html” del directorio “users”, para ser más específicos el siguiente fragmento.



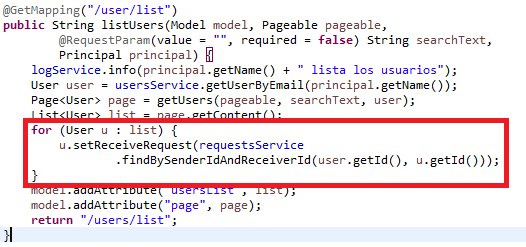
Como en thymleaf es difícil iterar en una lista de objetos contenida dentro de un objeto, he decidido hacer la comprobación en java. Para ello he añadido un atributo transient a los usuarios denominado “receiveRequest”.



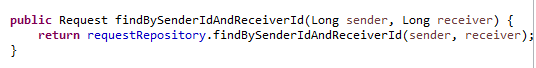
En este objeto se carga la petición de amistad entre el usuario que está en sesión y el este usuario, si existe. Así en thymleaf solo cambiamos lo que se muestra dependiendo de este atributo.



Como este atributo es null cuando se crean los objetos, a la hora de llamar al controlador se inicializa.



También hay que modificar “UsersServices” para añadir un nuevo método que devuleva las peticiones en “RequestServices”.



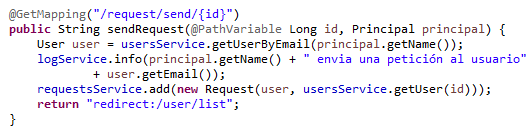
Y por consecuencia también hay que modificar “RequestRepository”.



Cuya consulta es:

* Request.findBySenderIdAndReceiverId=SELECT r FROM Request r WHERE r.sender.id = ?1 AND r.receiver.id = ?2

Una vez configurada la lista, cuando el usuario envia el formulario para enviar una petición, es decir pulsa el botón “Enviar solicitud” se llama a “RequestController” con el siguiente método.



Y a su vez a “RequestService” con el siguiente método.

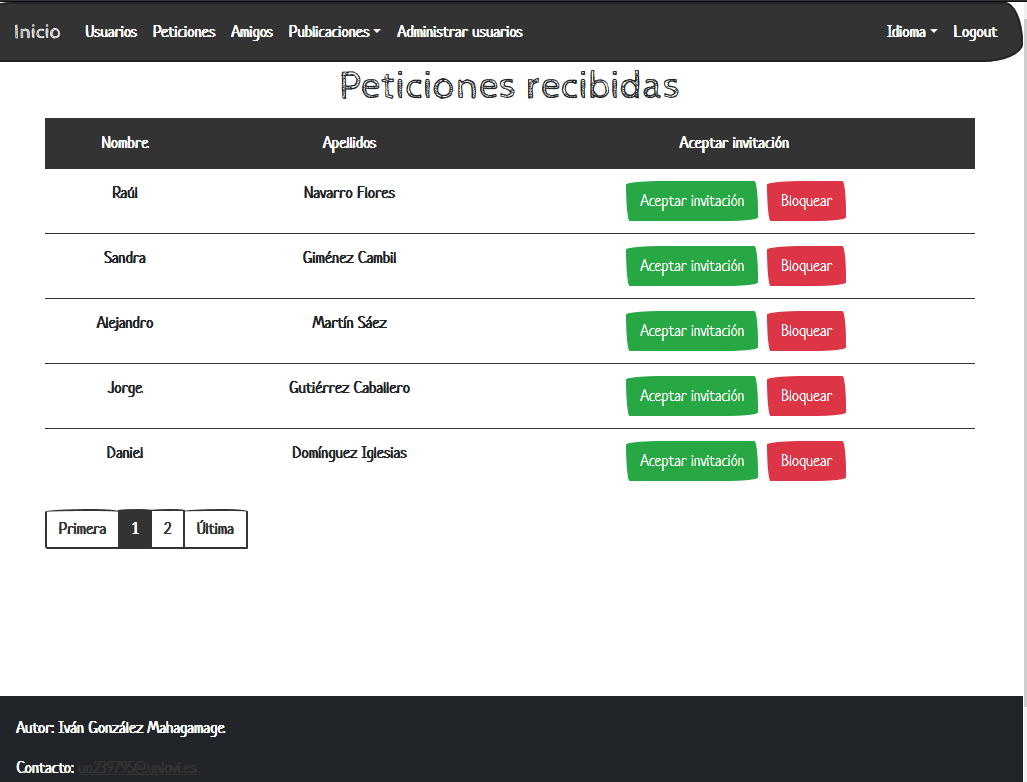


## Usuario registrado: listar las invitaciones de amistad recibidas

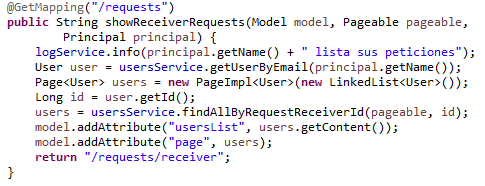
Para listar todas las invitaciones se ha creado una nueva vista denominada “receiver.html” dentro del directorio “requests”.



Dicha vista se mostraría de la siguiente forma.



Para mostrar esta vista, se ha creado un nuevo método dentro de “RequestReceiver”.



En “RequestService” se creado un nuevo método para cargar estas peticiones.



Y en “RequestRepository” también se ha añadido un nuevo método para realizar la nueva consulta.



Cuya consulta es:

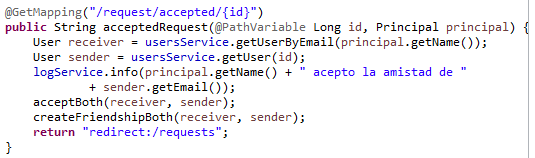
* User.findAllByRequestReceiverId=SELECT r.sender FROM Request r WHERE r.status = 'SENT' AND r.receiver.id = ?1

## Usuario registrado: aceptar una invitación recibida

Para aceptar una invitación existe el siguiente formulario en la vista “receiver.html”.

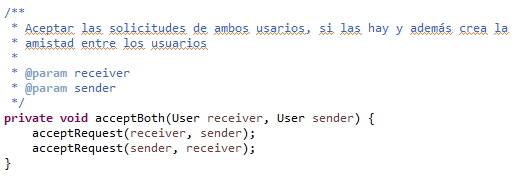


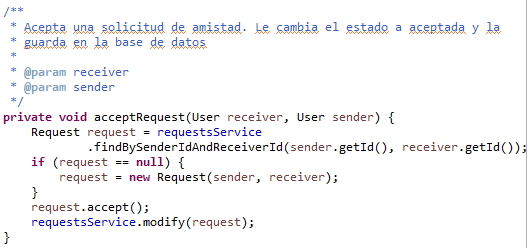
Este formulario llama a “RequestController”. Este controlador acepta la solicitud en varios pasos.

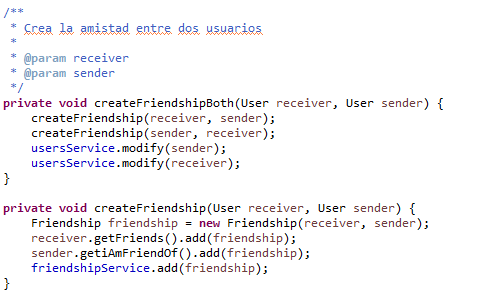


1. Comprueba las peticiones que existen entre los dos usuarios.
   1. Si existen en ambas direcciones, las marca como aceptadas.
   2. Si no existe una de ellas, la crea y la marca como aceptada. Esta solución no es la más optima, pero como no se especifica en el enunciado de la práctica he optado por esta vía ya que es la más rápida.
2. Actualiza/añade estas solicitudes.
3. Crea la nueva amistad en la base de datos en ambas direcciones.

Para realizar estas acciones se han creado los siguientes métodos en “RequestController”.



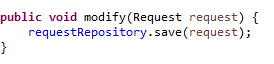




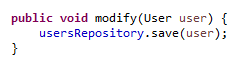
Dentro de la entidad “Request” creamos el método “accept”



En la clase “RequestService” creamos el método “modify()”:



Y para finalizar creamos el método “modify()” en “UsersService”.

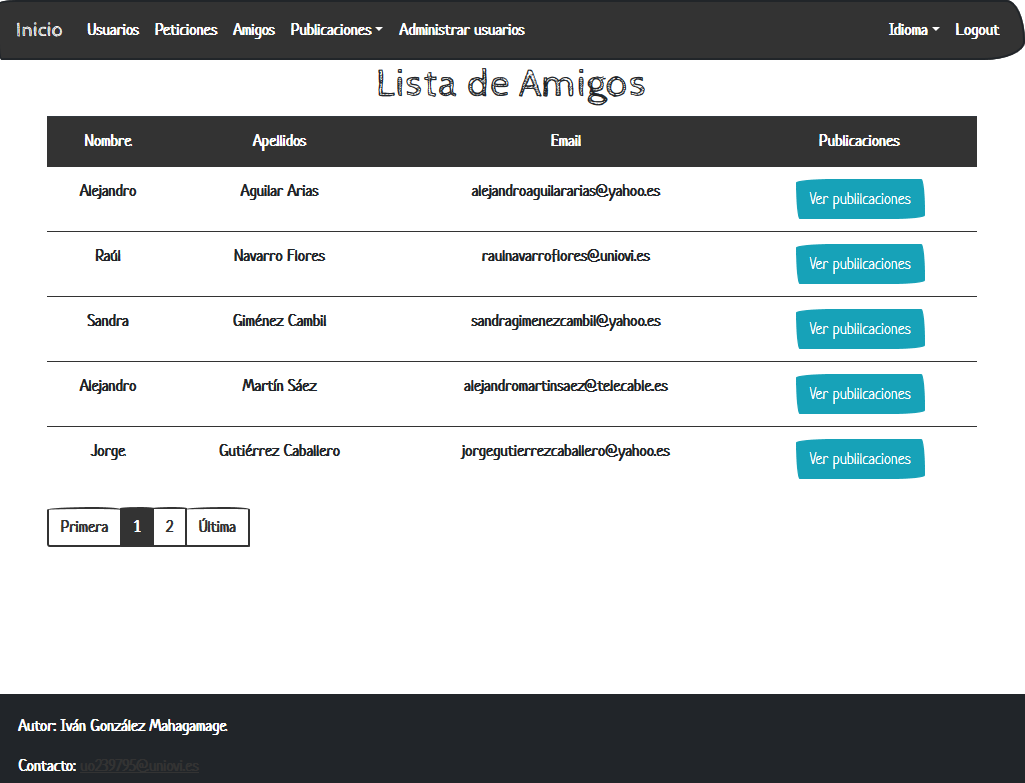


## Usuario registrado: listar los usuarios amigos

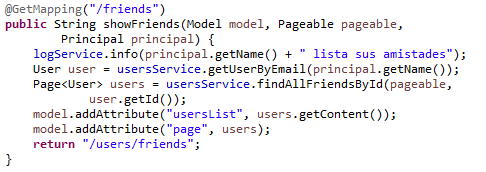
Para ver los amigos de un usuario se ha creado la vista “friends.html” dentro del directorio “users”.



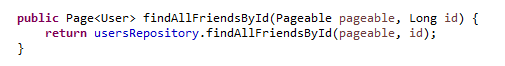
El resultado sería:



Para mostrar esta vista, se ha añadido el método “showFriends()” en “UserController”.



En “UsersServicies” se ha añadido el método “findAllFriendsById()”.



Y en “UsersRepository” se ha añadido el método “findAllFriendsById()”.



Cuya consulta es:

* User.findAllFriendsById=SELECT u.friend FROM Friendship u WHERE u.user.id = ?1

## Usuario registrado: crear una nueva publicación

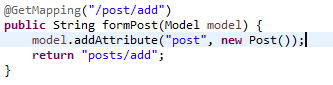
Para que el usuario cree nuevas publicaciones se ha creado la vista “add.html” dentro del directorio “posts”.



En esta vista ya están incluidos elementos del caso de uso 12 que se explicarán más adelante.  
El resultado de este html es el siguiente:



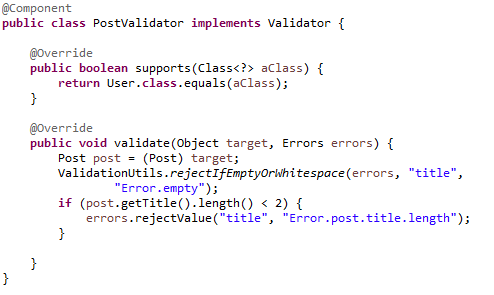
Para mostrar esta vista hay que implementar un nuevo método “formPost()” en “PostController”.



Y para procesar el formulario de creación hay que crear otro método en “PostController”.



Obviando la parte de imágenes, este método llama primero un validador para comprobar los datos introducidos por el usuario. Esta clase es “PostValidator”.



Una vez validado se guarda llamando a “PostService” con el método “add()”.



## Usuario registrado: listar mis publicaciones

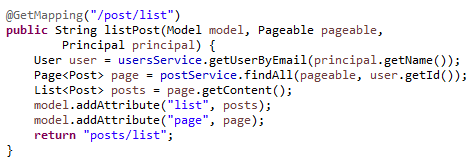
Para listar las publicaciones del usuario se ha implementado la vista “list.html” en el directorio “posts”.



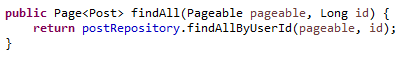
Y se mostraría de la siguiente forma.



Para ello, se ha implementado el método “listPost” en “PostController”



En “PostService” se ha implementado el método “findAll”.



Y en “PostRepository” el método “”.



Cuya consulta es:

* Post.findAllByUserId=SELECT p FROM Post p where p.user.id = ?1 ORDER BY p.date ASC

## Usuario registrado: listar las publicaciones de un usuario amigo

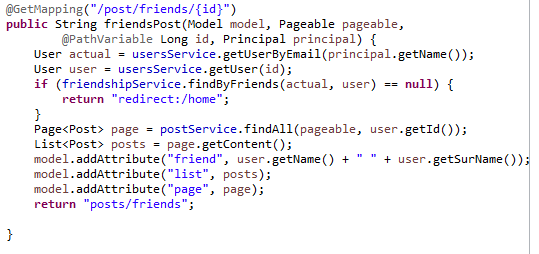
Para hacer este caso de uso, he creado unas vista denominada “friends.html” en la carpeta “posts”. Esta vista poco difiere a la de listar las propias publicaciones, únicamente se indica en el título de la página a quien pertenece estas publicaciones.



El resultado es:



En “PostController” añadimos el siguiente método.



En este método comprobamos que los dos usuarios tienen amistad, en caso contrario devolvemos al usuario a la página principal.

## Usuario registrado: crear una publicación con una foto adjunta

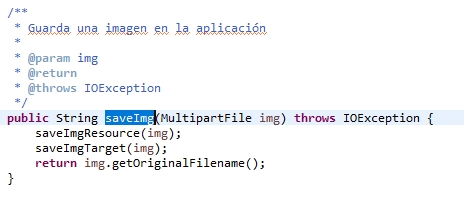
La vista usada para este caso es la misma que para el caso 9, voy a destacar el elemento que he añadido para que funcione.

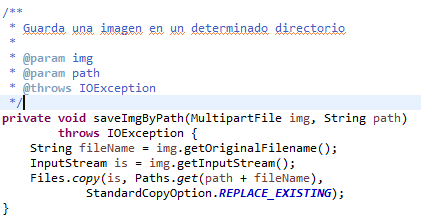


En “PostController” comprobamos que la imagen es válida.



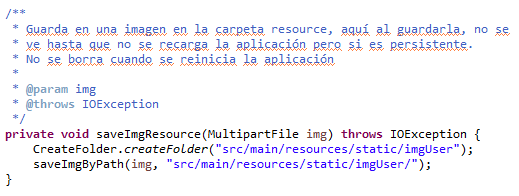
Despues en “PostService” guardamos el post con la imagen.



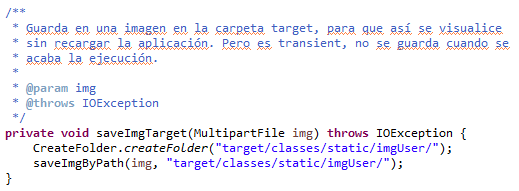


La imagen se guarda en dos sitios.

1. En la carpeta “resource” para que sea persistente pero no se cargara la imagen de este directorio hasta que no se reinicie o refresque el proyecto.



1. En la carpeta “target” para que se muestre la imagen sin recargar la aplicación, pero esta carpeta se destruye cuando se recarga, no es persistente

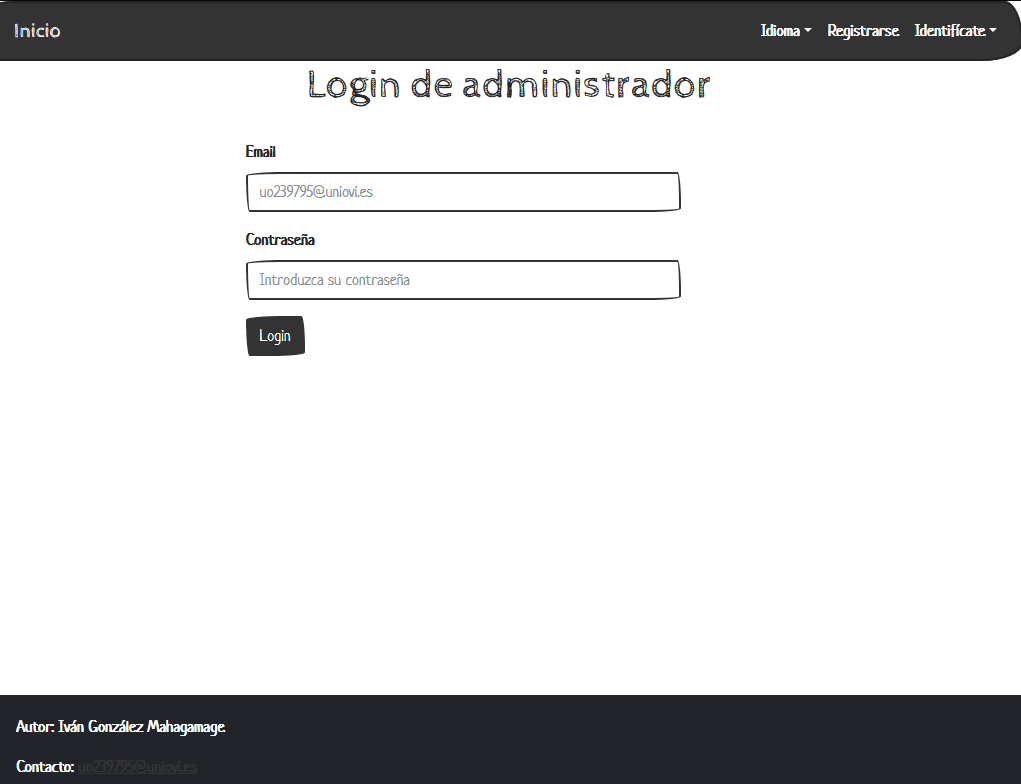


## Público: iniciar sesión como administrador

Para hacer este caso de uso, se creado una vista denominada “adminLogin.html”.

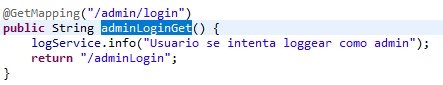


Cuya visualización es.



En “UserController” se han creado dos métodos para que funcione esta vista.

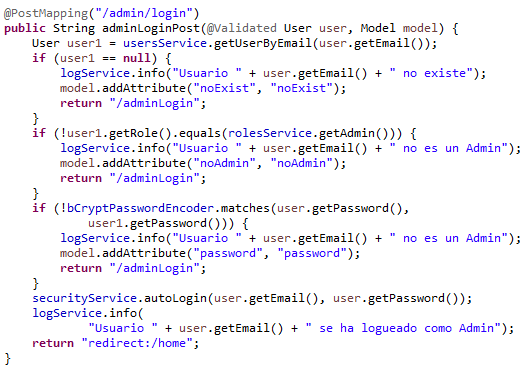
* Un método para mostrar la vista.



* Un método para comprobar el formulario.

1. Comprueba si el usuario existe.
2. Comprueba que el usuario es un administrador.
3. Comprueba la contraseña sea correcta.

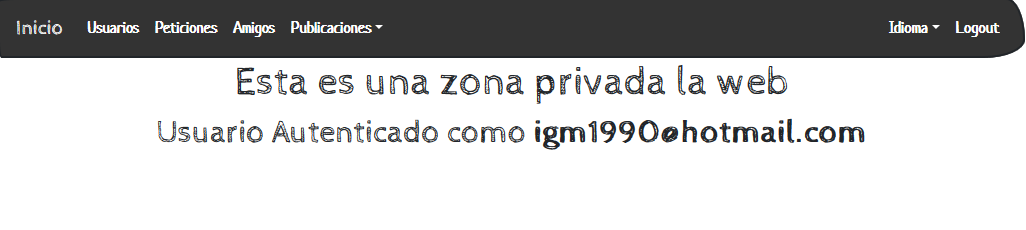
Si todas las comprobaciones son correcta logue al usuario, usando el autologin de Spring Security y lo lleva a la pantalla de “home”



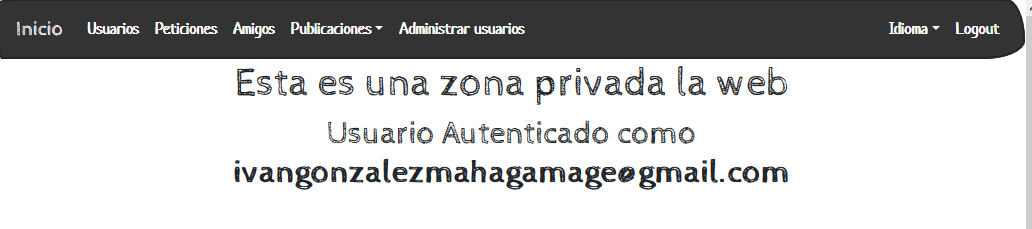
## Consola de administración: listar todos los usuarios de la aplicación

Para mostrar esta lista se ha creado una nueva opción en el nav solo visible para los administradores.

* Usuario.



* Administrador.



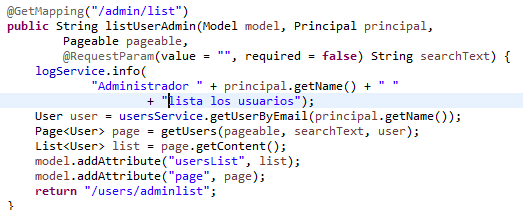
Después se ha creado la vista “adminlist.html”.



La cual se muestra de la siguiente forma.



En la clase “UserController” se añadido el método “”.

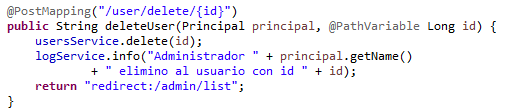


## Consola de administración: Consola de administración: eliminar usuario

Para este caso se ha usado la misma vista que en el caso 14. Voy a hacer hincapié en el elemento de esta vista usado para este caso.



Este formulario llama al método “deleteUser()” de la clase “UserController”.

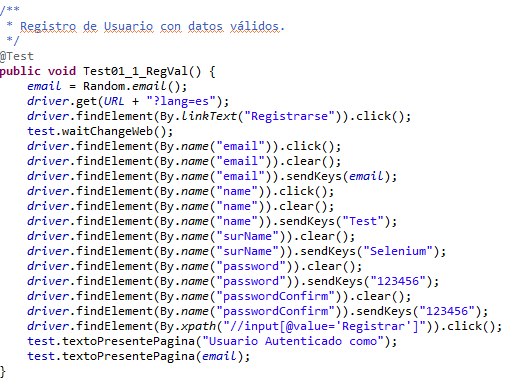


Este a su vez llama a “delete()” de la clase “UserService”

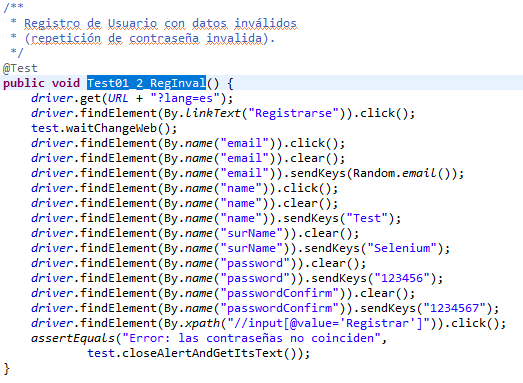


# Prueba Unitarias

## Registro de Usuario con datos válidos.

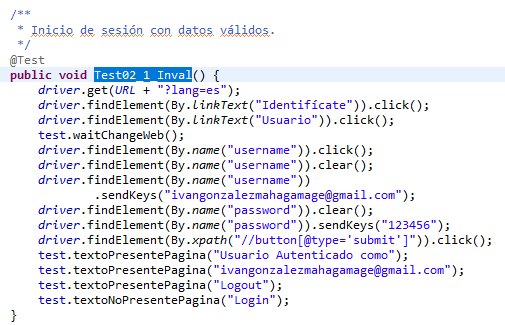


## Registro de Usuario con datos inválidos (repetición de contraseña invalida).

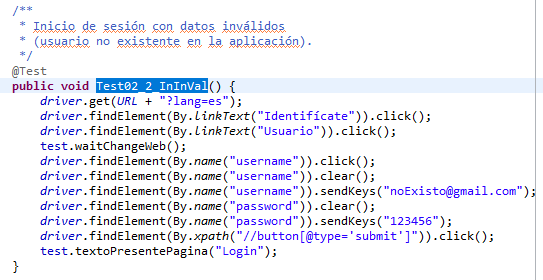




## Inicio de sesión con datos válidos.

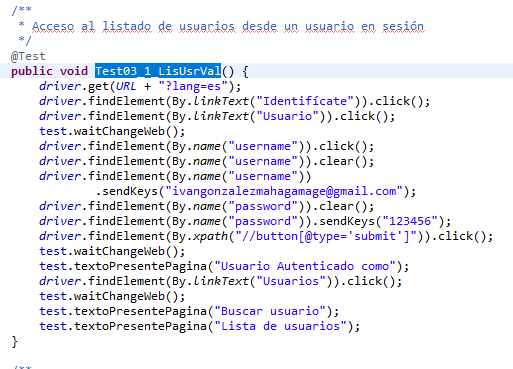


## Inicio de sesión con datos inválidos.

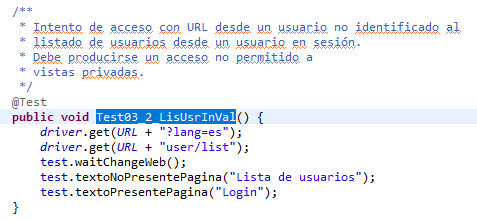




## Acceso al listado de usuarios desde un usuario en sesión.

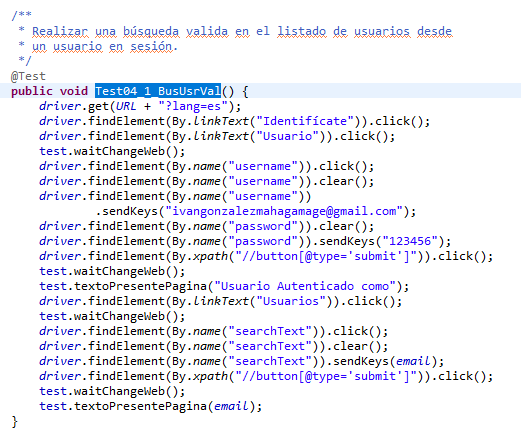


## Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión.

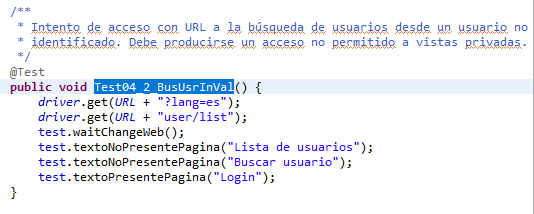




## Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión.

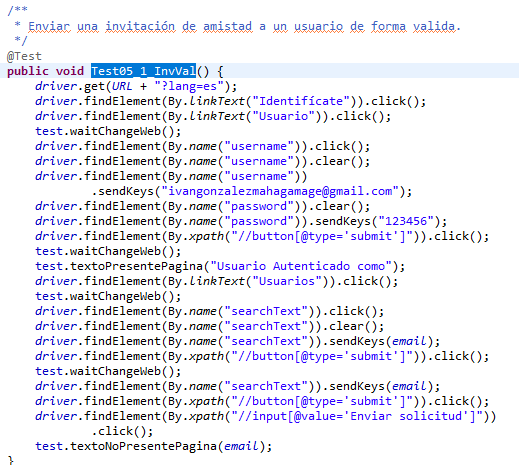


## Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado.

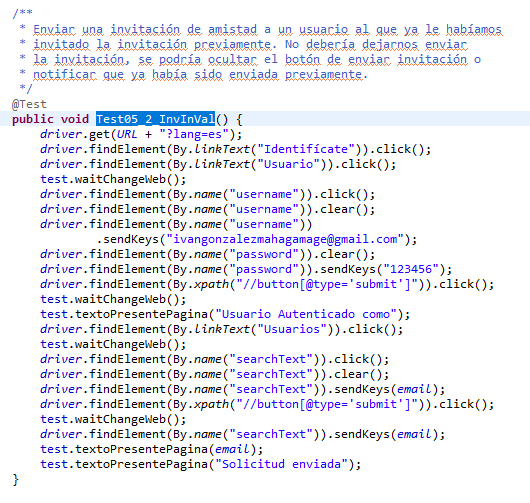




## Enviar una invitación de amistad a un usuario de forma valida.

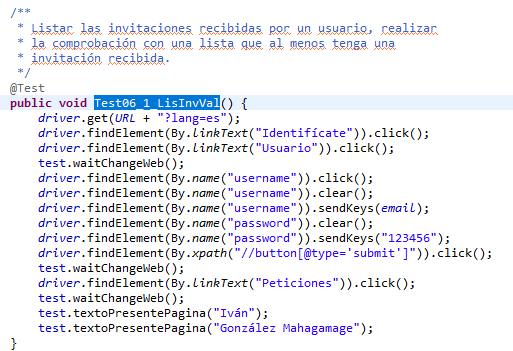


## Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente.



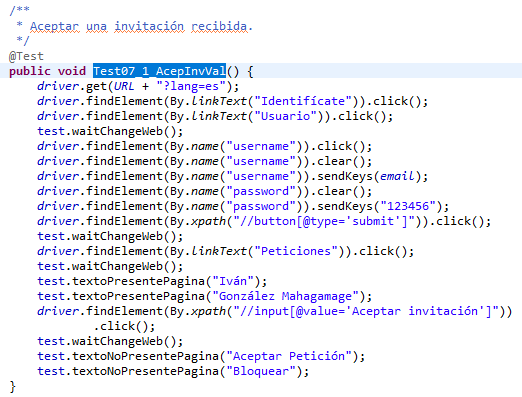


## Listar las invitaciones recibidas por un usuario.



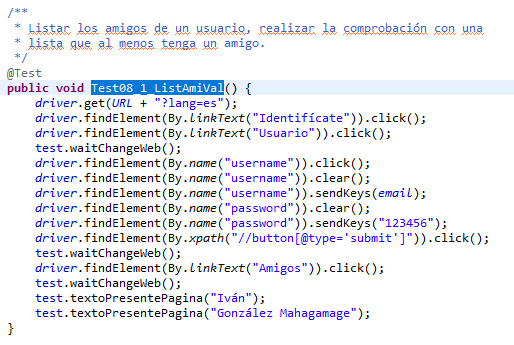


## Aceptar una invitación recibida.



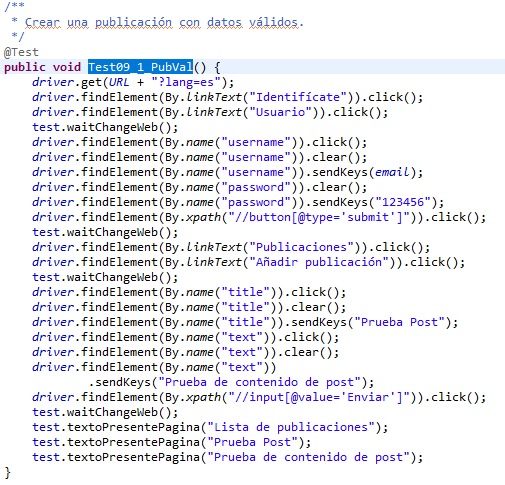


## Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo.



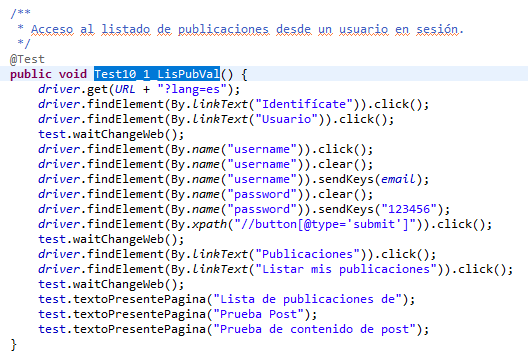


## Crear una publicación con datos válidos.





## Acceso al listado de publicaciones desde un usuario en sesión.

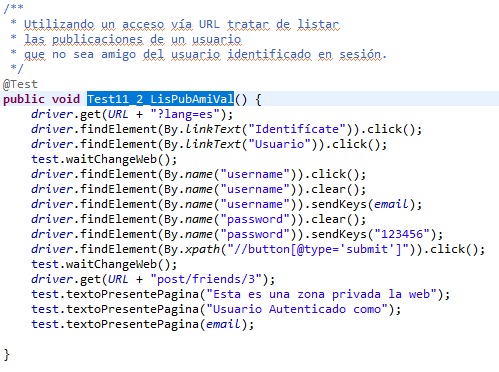




## Listar las publicaciones de un usuario amigo.

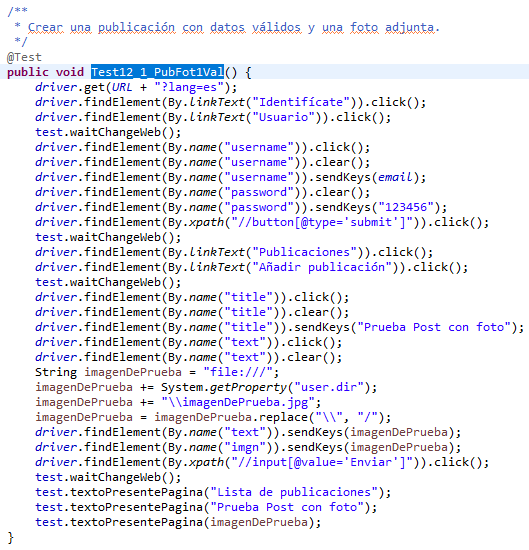


## Utilizando un acceso vía URL tratar de listar las publicaciones de un usuario que no sea amigo del usuario identificado en sesión.

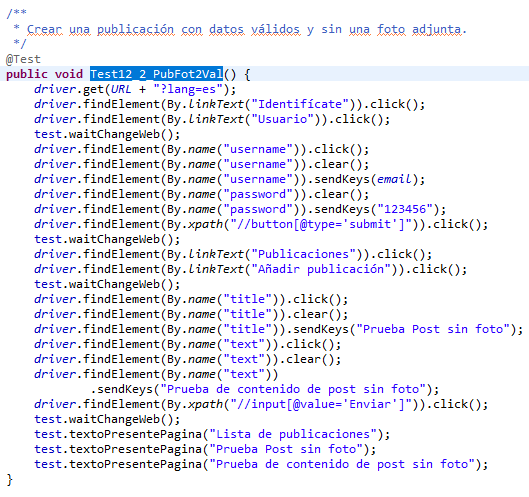




## Crear una publicación con datos válidos y una foto adjunta.

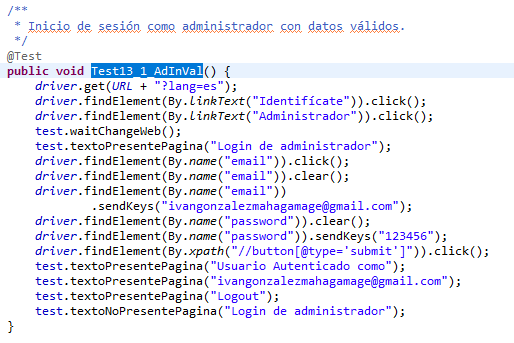


## Crear una publicación con datos válidos y sin una foto adjunta.

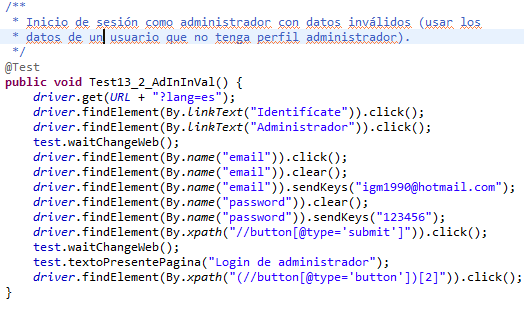




## Inicio de sesión como administrador con datos válidos.

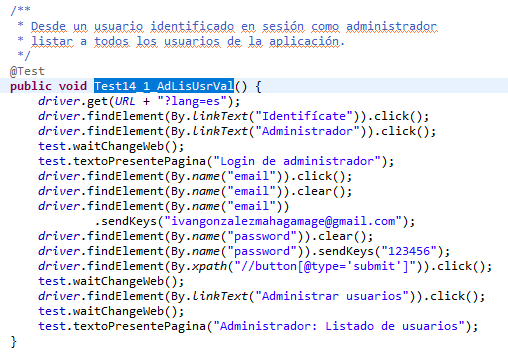


## Inicio de sesión como administrador con datos inválidos.



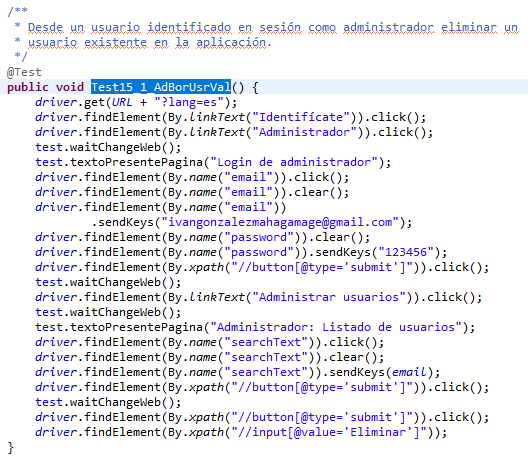


## Desde un usuario identificado en sesión como administrador listar a todos los usuarios de la aplicación.

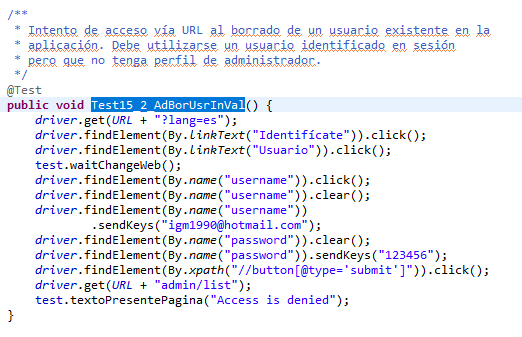




## Desde un usuario identificado en sesión como administrador eliminar un usuario existente en la aplicación.



## Intento de acceso vía URL al borrado de un usuario existente en la aplicación.



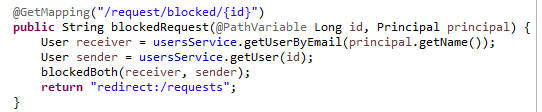
# Otros aspectos

## Usuario registrado: bloquear invitaciones de amistad de un usuario

Este caso de uso no está dentro de la especificación, pero lo implemente después de lo comentado en la tutoría grupal. Para ello, en la vista usada en el caso 7 se añadió un formulario adicional.



En “PostController” creamos un método que ejecute el formulario.



Después dentro de “PostService” añadimos los siguiente métodos.

