

# - PRÁCTICA 1 -

Guillermo Facundo Colunga, Pablo Menéndez Suárez.

## PROFESORADO:

Edward Rolando Núñez Valdez - [nunezedward@uniovi.es](mailto:nunezedward@uniovi.es)

---

**Resumen.** En este documento se muestra la metodología de trabajo empleada durante la práctica, los casos de uso implementados en la misma y los tests unitarios de selenium incluidos para la correcta ejecución de pruebas de aceptación sobre el proyecto suministrado.

**Palabras clave:** spring boot, jpa, sistemas distribuidos, selenium, maven.

---

## Introducción

Esta práctica se enmarca en el uso de Spring Boot para la realización de una pequeña aplicación distribuida que representa el dominio de una red social. Para ello se utilizará el IDE STS para el desarrollo de Spring Boot y Maven como gestor de dependencias.

## Metodología de trabajo

Para la realización de esta práctica ambos integrantes del grupo decidimos usar algún sistema para el control de versiones de nuestro código así como para poder trabajar sin problemas de forma totalmente independiente. Por ello usamos GitHub como plataforma para alojar nuestro código. De la misma forma y para automatizar las pruebas que se realizaban sobre el código configuramos un sistema de integración continua sobre la plataforma circle-ci. En dicha plataforma, de forma automática, cada vez que se realiza una Pull Request sobre el repositorio central en GitHub se despliega una base de datos completamente vacía y a continuación se procede a correr los tests unitarios de las entidades así como los tests de integración que se aplican a la capa de persistencia. Posteriormente se comprueba que la aplicación entera puede ser desplegada sin problemas.

De la misma forma se ha utilizado la librería lombok para añadir funcionalidades a java que no trae de serie, como la inclusión de etiquetas @Getter @Setter o @Slf4j que permite usar el log sin tener que realizar la instanciación explícita.

## Casos de uso implementados

De los 15 casos de uso propuestos en el documento original de la práctica se han implementado los 8 obligatorios. A continuación se describe su implementación.

### C.U.1: Registrarse como usuario

#### Descripción:

Permitir a los usuarios registrarse en la aplicación aportando un nombre, un email (que no podrá ser el mismo que otro previamente aportado) y una contraseña (que tendrá que repetir para confirmar que la contraseña introducida es la deseada por el usuario).

#### Paquetes afectados:

`com.uniovi.controllers`, `com.uniovi.entities.User`,  
`com.uniovi.services.SecurityService`, `com.uniovi.services.UsersService`,  
`com.uniovi.validators.SignUpFormValidator`.

#### Implementación:

Para poder permitir registrar un usuario lo primero que tenemos que hacer es definir un elemento que controle esa acción. Por lo tanto se definen una serie de controladores para recibir y transmitir la intencionalidad de un usuario de registrarse a través de la ruta `/signup`. Dichos controladores serán también los encargados de gestionar la validación del formulario de registro. Y en caso de registro realizarán un *autologin* del usuario en el sistema.

Por supuesto para su implementación total hace falta crear las plantillas de thymeleaf así como una documento específico `signup.html` que contenga el formulario para el registro.

#### Gestión de errores:

En el caso de que un usuario que se esté registrando introduzca algún valor en algún campo que no cumpla con las condiciones de validación se le mostrará un aviso en el propio formulario y se impedirá su registro.

### C.U.2: Iniciar sesión

#### Descripción:

Existirá un formulario de inicio de sesión en que el usuario pueda suministrar su email y contraseña. Si dichos datos de autenticación son válidos y la relación se encuentra registrada en el sistema se permitirá el acceso al usuario llevándolo a la lista general de usuarios de la aplicación.

#### Paquetes afectados:

`com.uniovi.controllers`, `com.uniovi.entities.User`,  
`com.uniovi.services.SecurityService`, `com.uniovi.services.UsersService`,  
`com.uniovi.services.UserDetailsService`, `com.uniovi.WebSecurityConfig`.

#### Implementación:

Se empieza por crear una configuración de seguridad web para la aplicación, para ello se crea la clase `WebSecurityConfig.java` donde se especifica la política de seguridad que se aplicará a cada ruta. A continuación se realiza un controlador que requiera el mapeo de `/login` a la página correspondiente. Y finalmente en la página correspondiente se delega en spring boot y thymeleaf para que realicen la autenticación junto a los servicios que se proveen.

#### Gestión de errores:

En caso de que la relación de datos provistos no coincida con ningún registro de nuestra aplicación el usuario no podrá acceder a las páginas privadas.

#### C.U.3: Listar todos los usuarios de la aplicación

##### Descripción:

A través de un enlace en la barra de navegación se permite a los usuarios autenticados acceder a una lista general de todos los usuarios de la aplicación. Dicha lista se encuentra paginada y muestra sólo 5 usuarios cada vez.

##### Paquetes afectados:

`com.uniovi.controllers`, `com.uniovi.entities.User`,  
`com.uniovi.services.UserService`.

##### Implementación:

En este caso se crea otro controlador para escuchar las peticiones enviadas a `/user/list` que será la ruta donde los usuarios autenticados puedan ver la lista general de usuarios. Dicho controlador cogerá el modelo, un objeto de paginación y un texto opcional de búsqueda (Se hablará de el en C.U.4). Para mostrar la lista de usuarios el controlador delegará en el servicio de usuarios el buscar la página de usuarios correspondiente para mostrarlo posteriormente en el modelo.

#### C.U.4: Buscar entre todos los usuarios de la aplicación

##### Descripción:

Para el caso de uso anterior (C.U.3) se incluye una barra de búsqueda que permite filtrar los registros de usuarios mostrados en C.U.3 por nombre y email. Esto es, para la búsqueda "alba" devolverá todos los usuarios que en el nombre o en el correo tienen "alaba".

##### Paquetes afectados:

`com.uniovi.controllers`, `com.uniovi.services.UserService`,

##### Implementación:

La implementación de este caso de uso extiende la del anterior ya que se basa en el tercer parámetro del controlador que mapea a `/user/list`. Lo que realmente hace es pasar el parámetro al servicio de usuarios que realiza una query sobre el repositorio de usuarios, permitiendo así filtrar por nombre e email.

#### C.U.5: Enviar una petición de amistad

##### Descripción:

Junto a la información de cada usuario procedente de C.U.3 se comprará un enlace que permita enviar una petición de amistad a dicho usuario.

**Paquetes afectados:**

`com.uniovi.controllers`, `com.uniovi.entities.User`.

**Implementación:**

Para implementar este caso de uso crearemos un enlace en la plantilla de thymeleaf que sea `/user/addFriendRequest`. Dicho enlace estará mapeado con un controlador que pedirá el mapeo para `/user/addFriendRequest/{id}`, a continuación buscará el id en el repositorio de usuarios a través del servicio correspondiente y le añadirá una petición de amistad del usuario actual. Por supuesto, para poder añadir la petición al usuario este tendrá que tener una relación mapeada que consistirá en un set de usuarios.

**C.U.6: Listar las peticiones de amistad recibidas.****Descripción:**

Para el usuario en sesión se mostrará la opción de ver las peticiones de amistad recibidas. Dichas peticiones se mostrarán en una lista que implementará un sistema de paginación con 5 peticiones por página.

**Paquetes afectados:**

`com.uniovi.controllers`, `com.uniovi.services.UserService`,  
`com.uniovi.entities.User`.

**Implementación:**

Para mostrar las peticiones se creará un controlador que requiera el mapeo correspondiente a `/user/peticiones`. Dicho controlador obtendrá las peticiones de amistad del usuario actual como una sucesión de páginas que poder mostrar en la tabla.

**C.U.7: Aceptar una invitación recibida****Descripción:**

Para la lista de invitaciones de amistad mostrada en el C.U.6 se mostrará un botón por cada tupla que permita al usuario en sesión aceptar las peticiones de amistad. Una vez una petición de amistad es aceptada esta debe desaparecer de la lista de peticiones para aparecer en la de amistades. Una vez aceptada ambos usuarios son amigos.

**Paquetes afectados:**

`com.uniovi.controllers`, `com.uniovi.services.UserService`,  
`com.uniovi.entities.User`.

**Implementación:**

Para implementar este caso de uso se extiende el caso de uso anterior añadiendo un controlador que requerirá el mapeo a `/user/acceptFriendRequest/{id}`, donde id será el id del usuario a quien aceptaremos la petición de amistad. El método de aceptar la petición de amistad se encuentra implementado dentro de cualquier usuario y lo que hace es, en una transacción, elimina la petición de amistad y añade a A como amigo de B y a B como amigo de A.

**Gestión de errores:**

Debido a la necesidad de hacer la transacción atómica se usa la etiqueta `@Transactional`

**C.U.8: Listar los usuarios amigos****Descripción:**

Para el usuario en sesión se mostrará un enlace que permitirá acceder a una vista que muestre una lista paginada de los usuarios amigos.


**Paquetes afectados:**

`com.uniovi.controllers`, `com.uniovi.services.UsersService`.

**Implementación:**

Este caso de uso es muy similar a los anteriores de listar y por lo tanto se emplearán las mismas técnicas. Se creará un receptor de la petición sobre `/user/amigos`. Dicho controlador se encargará de añadir al modelo la vista paginada de amigos y posteriormente re-dirigir a la vista correspondiente. En dicha vista se mostrará la lista de amigos haciendo uso de `thmleaf`.

## Catálogo de pruebas unitarias

 **ATENCIÓN.** Los tests se han desarrollado desde la plataforma macOS, por lo tanto se encuentran configurados para dicha plataforma. Si se quisieran ejecutar desde una plataforma distinta se tendrían que configurar los tests para que arrancaran el navegador firefox en la ruta deseada.

Código Prueba	Nombre	Descripción
PR01_1	RegVal	Realiza un registro de usuario con datos que son aceptados por el validador.
PR01_2	RegInval	Realiza un registro de usuario insertando una repetición de contraseña diferente de la contraseña, con lo que el validador de los datos deberá de indicar que la contraseña introducida y su repetición no coinciden.
PR02_1	InVal	Se realiza un inicio de sesión con datos válidos. Con lo que se espera que la aplicación nos deje acceder a las zonas privadas.
PR02_2	InInVal	Se realiza un inicio de sesión con datos NO válidos. Con lo que se espera que la aplicación NO nos deje acceder a las zonas privadas, en cambio se nos pedirá de nuevo que introduzcamos el email y la contraseña.
PR03_1	LisUsrVal	Se accede al listado de usuarios desde un usuario en sesión.
PR03_2	LisUsrInVal	Se intenta acceder directamente a través de la url al listado de usuarios de la aplicación sin estar autenticados en la misma.
PR04_1	BusUrsVal	Se comprueba que se puede buscar en la lista general de usuarios.
PR04_2	BusUrsInVal	Se intenta acceder directamente a través de la URL a la búsqueda de usuarios, sin que estemos autenticados en la aplicación.

PR05_1	InvVal	Se envía una invitación de amistad al usuario con email <u>p6@hotmail.com</u>
PR05_2	InvInVal	Se intenta enviar una petición de amistad al mismo usuario, <u>p6@hotmail.com</u> , pero como a este usuario ya se la hemos enviado la aplicación no nos dejará.
PR06_1	ListInvVal	Se comprueba que un usuario autenticado en la aplicación puede acceder a la lista de invitaciones de amistad pendientes de aceptar y que dicha lista tiene al menos un elemento.
PR07_1	AceplInvVal	Desde un usuario autenticado se accede a la listad e invitaciones de amistad y se acepta una.
PR08_1	ListAmiVal	Se comprueba que desde un usuario autenticado en la aplicación se puede acceder a la lista de amigos y que dicha lista tiene al menos un elemento.