UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Implementación de un sistema de acceso seguro a web con JEE

MEMORIA PRESENTADA POR:

Juan Francisco Blasco Jordá

GRADO DE INGENIERÍA INFORMÁTICA

Convocatoria de defensa: Febrero-Marzo de 2018

Tutor: Pau Micó

Resumen

En este proyecto se realizará un estado del arte al respecto de las tecnologías de securización del acceso web. Además, se deberá implementar utilizando el framework Java Enterprise Edition, un ejemplo de aplicación web en la que el usuario pueda:

- Registrarse por vez primera para dar de alta la dupla usuario-contraseña.
- Acceder a la aplicación a partir de la dupla usuario-contraseña, si ya ha sido registrado anteriormente como usuario.

El acceso a la aplicación debe ser seguro (la información se envía cifrada). Además, el diseño de la base de datos permitirá que las contraseñas se almacenen de forma cifrada.

Este TFG comprenderá las fases de:

- estado del arte de sistema de acceso seguro a aplicaciones web
- implementación en JEE.

Palabras clave: JEE, GlassFish, SSL, Seguridad, Registro

Summary

This project will go through the state of art of securitization technologies of web access. Furthermore, an example of web application must be implemented. In this application the user will be able to:

- Register a user-password dupla for the first time.
- Access to the application using that user-password dupla, if it has been previously registered as user.

The access must be secure (information is sended encrypted). Also, the database design will allow the passwords to be stored with their encryption.

This project will include the phases of:

- State of art of the secure access system to web applications.
- Implementation in JEE

Keywords: JEE, GlassFish, SSL, Security, Register.

Contenidos

1 Introducción

¿Qué es lo que lleva a un usuario a confiar en una aplicación web? En primer lugar, la calidad y utilidad de la propia aplicación, pero también juega un papel importante la seguridad de la misma. ¿A qué se debe? Generalmente a la cantidad de datos de carácter personal que se pueden llegar a manejar, y que podrían quedar expuestos.

1.1 Antecedentes

En estos momentos hay entornos de desarrollo como NetBeans, con el que se ha trabajado a lo largo de la carrera, que ya ofrecen cierto nivel de seguridad, puesto que conectan de forma segura los sitios web desarrollados con las bases de datos. Sin embargo, deja sin cifrar la conexión con el usuario (entre otras cosas). Para entender las carencias que presentan estos entornos que permiten (en cierto grado) la automatización de la creación de aplicaciones web, hay que realizar un repaso de aquello que convierte un sitio web en "seguro". Posteriormente se trabajará en la implementación de aquellos elementos que se consideren importantes en un servidor GlassFish.

Una rápida búsqueda pone de manifiesto la falta de documentación y de unanimidad en cuanto al proceso para conseguir conexiones seguras con certificados válidos en un servidor de Glassfish, así que este documento podrá servir de guía una vez finalizada la investigación.

1.2 Objetivos

El primero y más necesario: la conexión segura al servidor GlassFish, que es la implementación oficial de JEE. Esto requerirá de la instalación de los certificados necesarios.

- Realizar un estudio del estado del arte en cuanto a seguridad en las conexiones a sitios web se refiere.
- Comprobar estrategias sobre el almacenaje en base de datos de credenciales de usuario.
- Investigar sobre las posibilidades que nos ofrece GlassFish para la implementación de los puntos anteriores.
- Implementar un sitio web sencillo que cumpla con lo anteriormente comentado.

1.3 Requerimientos

El objetivo de este trabajo es ofrecer una conexión segura al servidor GlassFish para el intercambio de datos, y que el almacenamiento de las credenciales del usuario no supongan un riesgo.

Para el primer punto se utilizarán certificados generados por la herramienta keytool, ofrecida por Java. Con ella se crearán tres pares de claves y tres certificados, que equivaldrán a una Autoridad de Certificación, una Autoridad Intermedia y un usuario final. Estos se firmarán utilizando el algoritmo de hash SHA-256 para cumplir con lo exigido por los navegadores modernos. El certificado intermedio, el de usuario final y el par de claves del mismo deberán estar instalados en el servidor GlassFish, y el certificado raíz deberá ser importado como Entidad de Certificación Raíz de Confianza.

Para el almacenamiento de los datos del usuario se hará uso de una base de datos MySQL. GlassFish estará configurado para validar contraseñas que estén almacenadas como hash, y la aplicación registrará a los usuarios utilizando la misma técnica.

La gestión de la seguridad estará en manos de un Security Realm, que controlará el acceso de usuarios no identificados a zonas restringidas.

2 Anteproyecto

2.1 Estado del arte

El primer paso consiste en realizar un análisis de aquello en lo que se basa la seguridad, a qué se enfrentan las aplicaciones web. También se dará un repaso a la la evolución de algunas de las herramientas (como los algoritmos utilizados para proteger una conexión). Como punto más importante, el uso de SSL/TLS para la protección de las conexiones.

Posteriormente se hablará de cómo Java en concreto se enfrenta a las fallas de seguridad más típicas y las herramientas y frameworks que están a su disposición.

2.1.1 Análisis

En este apartado se van a estudiar las posibilidades de explotación de vulnerabilidades, las bases para conseguir la seguridad en las conexiones y cuál es el panorama actual en lo que a amenazas se refiere.

2.1.1.1 SSL/TLS

La señal que indica que un sitio web está utilizando un certificado digital es la presencia del "https" en la dirección. Https es el protocolo basado en http que hace uso de un cifrado basado en SSL/TLS, que asegura principalmente la privacidad y la integridad de datos mediante el uso de certificados digitales y claves asimétricas[1]. A continuación se van a dar los detalles sobre el establecimiento de conexión y la comprobación de identidad .

¿Cuál es el proceso que se lleva a cabo para establecer una conexión inicial y establecer un canal de comunicación privado y único? El "handshake" o "apretón de manos" entre el cliente y el servidor. Este se puede dividir en una serie de pasos:

- El cliente envía un mensaje "client hello", que incluye información criptográfica. Esta información contiene, entre otras, la versión SSL o TLS, o las versiones CipherSuites[2]

[1] Indica la existencia de dos claves: una pública y una privada. Aquello que se ha cifrado con la clave privada solamente puede descifrarse con la pública, y viceversa.
[2] Set de algoritmos criptográficos, especifica un algoritmo para cada una de las siguientes tareas: intercambio de claves, cifrado masivo y autenticación de mensaje.

soportadas, en orden de preferencia. También se envía una cadena de bytes aleatorios, que se utilizarán en procesos subsiguientes.

- El servidor responde con un mensaje "server hello", que contiene la CipherSuite elegida por el servidor de entre las indicadas por el cliente, un identificador de sesión y otra cadena de bytes aleatorios. Junto a este mensaje envía el/los certificado/s digital/es firmado/s. En caso de necesitarlo, solicitará al cliente un certificado de cliente.

- El cliente verifica el certificado digital. El proceso de verificación está explicado más adelante.

- El cliente envía la cadena de bytes aleatorios que permitirá calcular la clave secreta para cifrar los siguientes intercambios de mensajes de datos. Esta clave será cifrada con la clave pública del servidor (de forma que solamente este último la pueda descifrar, puesto que es quien posee la clave privada).

- En caso de haberse solicitado un certificado de cliente, el cliente envía una cadena de bytes aleatorios, cifrados con la clave privada del propio cliente. Todo esto se enviaría junto al certificado digital del cliente.

- El servidor verifica el certificado digital.

- El cliente envía un mensaje de finalización, cifrado con la clave secreta. Esto es indicativo de que el proceso de "handshake" ha llegado a su fin por parte del cliente.

- Mientras dure la sesión SSL o TLS, los mensajes que se intercambien entre el cliente y el servidor estarán cifrados de forma simétrica[3] con la clave secreta calculada anteriormente. Ello garantiza la privacidad del mensaje, puesto que aunque sea interceptado, no puede ser descifrado.

El problema ahora es, si la clave pública del servidor es, como bien se indica, "pública", ¿qué impide a un atacante incorporarla en su propio certificado? La respuesta es: nada. Es más, podría realizarse todo el proceso de handshake sin ser el servidor real. El problema vendría a la

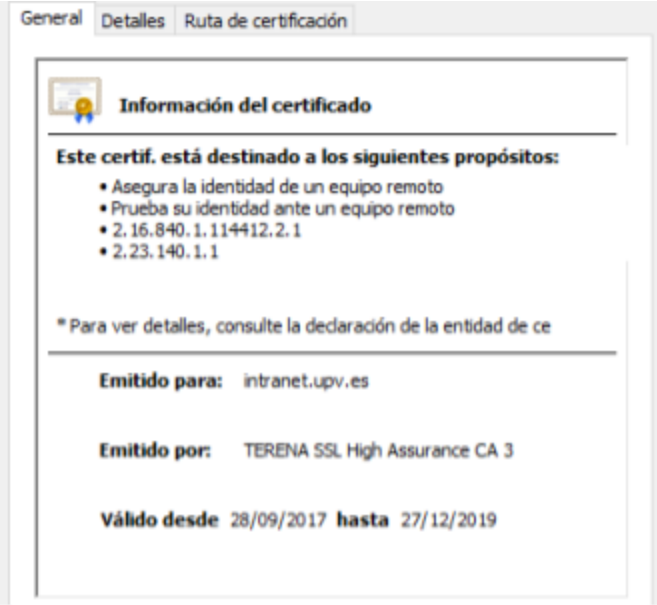
[3] Esto significa que solamente existe una clave, que sirve tanto para cifrar como para descifrar los mensajes.

hora de descifrar el mensaje que contiene la clave secreta que se va a utilizar a lo largo de la sesión: se envía cifrada con la clave pública del servidor original, y para descifrarla se necesita la clave privada. Además, todavía es necesario verificar que el certificado sea válido, y que se haya emitido para el servidor que lo presenta.

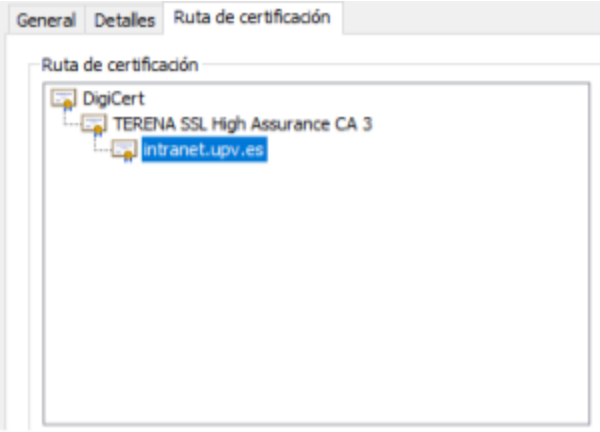
Para ello primero hay que preguntarse: ¿Quién puede firmar un certificado? La respuesta es: cualquiera. Cualquier persona/entidad puede generar un certificado y utilizarlo para firmar otros certificados. Pero los navegadores desde los que se contactan los servidores no confían en cualquier persona/entidad, sino solamente en las Autoridades Certificadoras (AC en adelante) y sus respectivos certificados. Estos tienen dos características que los diferencia del resto. Primero: están firmados por la misma entidad que los emite; y segundo: el servidor no necesita enviarlos, puesto que ya están en el depósito de certificados raíz de confianza del navegador. Las AC, sin embargo, no suelen firmar certificados finales (los que contendrían la información de los servidores junto a su clave pública), sino que delegan en autoridades intermedias. Estas autoridades intermedias tendrán su propio certificado, junto a una clave privada, firmado por la AC, y lo utilizarán para firmar los certificados finales (también podrían firmar certificados para otras autoridades intermedias).

Este conjunto de certificados termina formando una cadena, conocida como "cadena de confianza". El cliente que los recibe debe comprobar que la cadena no se rompa en ningún momento, puesto que eso supondría que el servidor les está entregando un certificado no válido, bien sea porque ha pasado la fecha de expiración o por cualquier otro motivo. Esto no impediría el intercambio cifrado de datos, pero el cliente no tendría forma de saber si el servidor con el que está contactando es quien dice ser.

La comprobación será explicada mediante un ejemplo real: el conjunto de certificados que se presentan al visitar la página de la Intranet de la UPV: https://intranet.upv.es

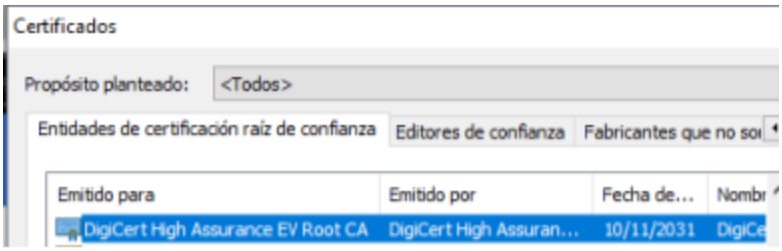


Certificado presentado.



Cadena de certificados.

En la primera imagen podemos ver quién ha emitido el certificado final de intranet.upv.es, para qué propósitos ha sido emitido y su validez. En la segunda podemos ver la cadena de certificados, siendo el de DigiCert (este es su nombre descriptivo) el certificado raíz. Si consultamos el depósito de certificados raíz de confianza del navegador, lo encontraremos ahí:



Certificado raíz

El primer paso de la comprobación pasa por revisar si se dispone de todos los certificados involucrados en la cadena. Salvo el certificado raíz, el servidor debería haberlos enviado todos. Si falta alguno, no se sigue con la comprobación, no hay forma de verificar la cadena de confianza.

Si se dispone de todos los certificados, el segundo paso es comprobar la validez del certificado final. Para ello se utiliza la "firma" que presenta. Esta firma es el resultado de aplicarle una función de hash (en el certificado se indica cuál se ha aplicado) al certificado y posteriormente cifrar el resultado con la clave privada de la autoridad emisora (TERENA SSL en este caso). El cliente deberá comprobar que el resultado de descifrar la firma del certificado final (intranet.upv.es) con la clave pública del certificado de TERENA SSL coincide con el resultado de aplicar una función hash al certificado final. En caso de coincidencia, podemos tener la seguridad de que es TERENA SSL quien ha firmado ese certificado, puesto que solamente esa entidad dispone de la clave privada para tal propósito.

El tercer paso sería repetir el segundo, pero en este caso para comprobar la validez del certificado de TERENA SSL. Este paso se repetiría hasta llegar al certificado raíz, que no hay que comprobar porque está emitido por una AC raíz de confianza.

El protocolo SSL/TLS provee ya de una seguridad considerable, aunque no tiene por qué ser suficiente. El éxito del SSL dependerá en gran parte de las funciones hash que se empleen durante el proceso de firma de los certificados.

También es habitual que en una aplicación web el cliente pueda registrarse, de forma que el servidor necesite almacenar unos credenciales del propio cliente (generalmente una dupla

usuario-contraseña). Y, aunque la comunicación se esté realizando de forma segura con la base de datos, eso no significa que haya que despreocuparse de cómo se guardan las credenciales.

2.1.1.2 Protección de las contraseñas

Las contraseñas no deben ser guardadas como texto plano, puesto que hasta un simple descuido que permita una lectura de la base de datos expondría las duplas email-contraseña (o cualquiera que sean los datos que asociemos al usuario). Esto entraña un peligro más allá del alcance de nuestra aplicación: en el 2013 un estudio[4] puso de manifiesto que la mitad de los usuarios de internet del Reino Unido usaban la misma contraseña para los sitios en los que se registraban. Esto deja un dilema: ¿de qué forma se deberían almacenar las contraseñas?

Una primera solución que puede acudir a la mente pasa por realizar un cifrado de las mismas. Solucionaría el problema principal: un simple vistazo a la base de datos no sería suficiente como para exponer todas las contraseñas sin más. Sin embargo, el cifrado requiere de una (o dos, en caso de que sea asimétrico) claves, y, por tanto, es reversible. No parece que sea la mejor solución.

La manera más correcta es una que ya se ha comentado anteriormente cuando se ha hablado de los certificados digitales: el hasheo. Esta técnica convierte las contraseñas en cadenas irreconocibles, y no existe forma de, a partir de esas cadenas, recuperar el contenido original.

2.1.1.3 Sobre las funciones hash

Se ha hablado en puntos anteriores del "hasheo", sin embargo puede no estar clara su objetivo. Una función de hash recibirá cualquier dato (ya sea una cadena de texto, un entero, un array) y devolverá una cadena de texto con una cantidad predefinida (generalmente) de caracteres. Esta cadena resultante se conoce como "digest". El "hasheo" se diferencia de un cifrado en que no es posible recuperar el dato original.

[4] https://www.ofcom.org.uk/about-ofcom/latest/media/media-releases/2013/uk-adults-taking-online-password-security-risks

Su utilidad radica en la posibilidad de comprobar la integridad de una serie de datos, puesto que la posibilidad de que dos entradas distintas produzcan la misma salida es muy escasa (pero no inexistente). Que dos entradas produzcan la misma salida se considera una colisión. Las funciones de hash han ido evolucionando precisamente para reducir las posibilidades de que se produzcan colisiones.

Las funciones más conocidas y utilizadas son las que siguen:

- MD5: a día de hoy su uso ha quedado reducido a la comprobación de integridad de archivos, debido a su antigüedad (1991) y a que ya se encontraron colisiones en el 1996, poniendo en entredicho su seguridad.

- SHA-1: desarrollada por la Agencia de Seguridad Nacional (NSA), ha sido ampliamente utilizada durante mucho tiempo, sin embargo, ya se han encontrado colisiones y lleva un tiempo sin ser la recomendada para temas relacionados con la seguridad. Diversas compañías, entre ellas Microsoft, han empezado a rechazar certificados que han sido firmados utilizando esta función. (Bibliografía: SHA-1 deprecation countdown). De momento el tratamiento que reciben estos certificados es el de mostrar avisos indicando que esa firma se ha generado utilizando una función que ya no es segura. Esta función produce una salida de 40 dígitos hexadecimales (esta suele ser su representación más utilizada)

- SHA-2: es el heredero de SHA-1, estando compuesto por una serie de funciones para generar cadenas de diferentes longitudes. Estamos hablando de un conjunto de seis: SHA-224, SHA-256, SHA-384, SHA-512/224 y SHA-512/256. Dan lugar a salidas de números hexadecimales de entre 56 y 128 dígitos. Esta longitud reduce considerablemente la posibilidad de hallar colisiones.

- SHA-3: lanzado en 2015, es la versión más nueva. Su propósito de momento no es sustituir a SHA-2, sino que cumple con la función de hacer de salvoconducto en caso de que sea requerido.

2.1.1.4 Ataques/vulnerabilidades comunes en aplicaciones web

Es necesario conocer a qué se expone una aplicación web, y en qué medida se ven afectadas aquellas basadas en JEE. La Open Web Application Security Project (OWASP en adelante) es una fundación sin ánimo de lucro que recoge un conjunto de buenas prácticas en lo que a seguridad se refiere y realiza estudios reuniendo las vulnerabilidades más comunes.

Entre ellas se encuentra la inyección de SQL, que puede darse cuando se utilizan valores como parte de un comando SQL. Un ejemplo de ello es el que sigue:

```
String query = "SELECT * FROM user WHERE email  = " + email + "and password
= " + password;
```

Siendo "email" y "password" variables con valores obtenidos a partir de entradas de usuario. Un usuario malicioso podría ingresar como contraseña " ; DROP TABLE user", con lo que se eliminaría la tabla "user". Esto se debe evitar a toda costa. En su lugar hay que utilizar "PreparedStatement", que toma los valores de entrada única y exclusivamente como parámetros, sin permitir que se conviertan en código SQL válido. La consulta anterior tendría esta forma utilizando los "PreparedStatement":

```
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM user WHERE
email = ? AND password = ?");
stmt.setString(1, email);
stmt.setString(2, password);
```

En este caso las dos variables serían tomadas únicamente como parámetro y nunca como código válido.

Otro de los grandes problemas que se mantiene arriba en las listas de la OWASP es el control de sesiones y la pérdida de autenticación, que puede terminar en un usuario malicioso obteniendo el acceso de otro. Esto se puede producir interceptando el id de sesión de otro usuario, obteniendo así su sesión y actuando como si fuera ese otro usuario. Su prevención pasa por realizar todas las comunicaciones que se realicen una vez un usuario se ha autenticado de forma cifrada (sobre SSL), de forma que aunque se intercepte una petición no se puede obtener ese id.

El siguiente consiste en la exposición de datos sensibles: los atacantes roban datos del servidor, de la base de datos o del tráfico que se genera cuando no está debidamente encriptado. Aunque se diera el caso de que un usuario obtuviera acceso a una base de datos de una aplicación, no debería poder ver los datos sensibles (como las contraseñas) como texto plano, y aunque interceptara una comunicación, tampoco debería tener acceso al contenido de esta. La solución a lo primero consiste en aplicar un hash a esos datos y para lo segundo es necesario encriptar la conexión (nuevamente mediante SSL).

Siguiendo al anterior, encontramos el de pérdida de control de acceso. Se da cuando se deja sin controlar un acceso a la aplicación: en todo aquello que una aplicación ponga a disponibilidad de un usuario externo debe haber algún tipo de comprobación. La política recomendada para estos casos consiste en denegar todo, y solamente permitir lo mínimo. Un ejemplo de mala configuración sería que un usuario no identificado o sin permisos pudiera acceder a un recurso reservado a administradores simplemente alterando la URL. Para evitarlo, se pueden utilizar las anotaciones indicadas en el punto 2.1.2.3 Seguridad en Java EE para controlar esos casos.

A continuación se encuentra uno que, según la OWASP, "es la segunda vulnerabilidad más frecuente en OWASP Top 10, y se encuentra en alrededor de dos tercios de todas las aplicaciones". Se trata del Cross-Site Scripting (XSS en adelante). Se da cuando la aplicación no revisa ni valida los datos introducidos por los usuarios, y después los incluye en una respuesta HTTP o en el HTML sin haberlo escapado previamente. Escapar una cadena de texto consiste en sustituir los signos potencialmente peligrosos (como "<" o "&") por (por ejemplo) sus equivalentes como entidad HTML ("<" y "&amp" correspondientemente). Una de las formas más sencillas de evitar este problema consiste en comparar contra expresiones regulares todo aquello que el usuario introduce en los formularios. Y a la hora de mostrarlo, utilizar funciones que eliminen o sustituyan todos los caracteres potencialmente peligrosos.

2.1.2 Sobre Java EE y frameworks alternativos

2.1.2.1 Java EE en la actualidad

Esta plataforma ha pasado mucho tiempo sin recibir ninguna actualización (Java EE 7 fue lanzado en 2013) por parte de Oracle. No fue hasta finales de 2016 que se habló de una posibilidad real de lanzamiento de una nueva versión, y fue a finales de septiembre de 2017 cuando ésta vio la luz. No es, sin embargo, una actualización al uso. Java EE 8 es la primera parte de la gran actualización que venían anunciando, que preveía grandes revisiones a las especificaciones relacionadas con la seguridad.

Esta prisa por lanzar una nueva versión ha venido dada por el hecho de que Oracle, la compañía que se encargaba de mantener esta plataforma, ha decidido delegar la tarea en la Eclipse Foundation. El objetivo es agilizar todo el proceso de actualización, puesto que la Eclipse Foundation está basada en el código abierto y en la colaboración de los usuarios. El nombre elegido para este proyecto sería el siguiente: EE4J, Eclipse Enterprise for Java Project.

La fundación se ha propuesto una serie de marcas para determinar el éxito del proyecto, entre las que está la de tener una comunidad fuerte de desarrolladores, proveedores y usuarios apoyando esta tecnología. Su principal objetivo: una transición rápida de las tecnologías de Java EE 8 al proyecto.

Es entendible, por lo tanto, que esta plataforma va a sufrir muchos cambios en los próximos meses.

2.1.2.2 Qué ofrece Java EE

En visto de la velocidad a la que se va a mover ahora mismo, solamente se van a hablar de aquellos que son estables. Por tanto, solamente se van a tener en cuenta aquellas posibilidades que existen en la versión 7.

A día de hoy, esta plataforma permite crear aplicaciones web de gran complejidad, y dotarlas de la seguridad necesaria para ello. Para este desarrollo se divide el desarrollo de la arquitectura en una serie de capas, con la intención de poder organizar y dividir el código. La división más usual es la que sigue:

- Capa de presentación: aquello con lo que el usuario final interactúa, aquello que se ve. Aquí se utilizarán tecnologías genéricas, como HTML y CSS; y otras que pertenecen al framework de Java, como JavaServer Faces, destinada al desarrollo de interfaces de usuario y a la conexión de las mismas con la capa de negocio, ofreciendo componentes y control sobre los mismos. También están las JavaServer Pages, que serían el equivalente Java a PHP. Permitiría la generación de páginas dinámicas. Las versiones más modernas previenen el XSS de forma automática, escapando los caracteres potencialmente peligrosos al mostrar contenido al usuario.

- Capa de negocio/servicios: es la que llevaría la lógica de la aplicación, la que trabajaría con datos y entradas de usuario, y que pondría en contacto a los mismos. En esta capa entrarían objetos planos de Java y las JavaBeans, los componentes reutilizables y manipulables visualmente. También se encontrarían las Enterprise JavaBeans (EJB en adelante), responsables de ofrecer, entre otras cosas, seguridad, de la que se hablará en el siguiente punto. Estas EJB permiten integrar la lógica de negocio poniendo en contacto las vistas con los datos.

- Capa de datos: se encarga de hacerse y trabajar con los datos necesarios para el funcionamiento de la aplicación. Aquí se encontrarán las entidades que reflejarán la estructura de las tablas de la base de datos. Todas aquellas acciones que queramos ver reflejadas en la base de datos deberán estar programadas en estas entidades. La Java Persistence Api permitirá actuar contra la base de datos pero de una manera orientada a objetos. Una de sus principales ventajas es que no requiere escribir consultas SQL para tratar con los datos (al menos para realizar tareas básicas).

- Capa de integración: es la que proporciona los medios para acceder los orígenes de los datos, ya sean bases de datos u otros servicios. En esta capa se encuentra la Java Database Connectivity (JDBC en adelante), que permite la ejecución de operaciones sobre bases de datos independientemente de sus sistema internos. Puede ser configurada directamente en el servidor.

2.1.2.3 Seguridad en Java EE

En su especificación se encuentran una serie de funcionalidades destinadas a autenticación, autorización y seguridad de transporte. También encontramos una serie de conceptos muy específicos de Java EE, de los que es necesario hablar para entender cómo se define la seguridad:

- Grupo: sirve para diferenciar y clasificar usuarios. Típicamente se situará a los usuarios que compartan características en un mismo grupo. Se pueden definir tanto en la aplicación como de forma externa (cuando se utiliza un proveedor distinto para identidad, como una base de datos).
- Rol: al igual que los grupos, su utilidad es la de clasificar usuarios. La diferencia: en el rol se especifican los permisos de acceso a los recursos de la aplicación. Posteriormente, cada grupo se mapea en uno o más roles. Estos roles estarán definidos en la propia aplicación.

Este mapeo se define en los descriptores específicos del servidor (glassfish-web.xml en el caso de un servidor GlassFish). El mapeo se realizará dentro de una "security constraint", donde se especifica si determinados recursos solamente tienen acceso desde un determinado rol, o si solamente se tiene acceso utilizando determinados métodos HTTP. También es posible indicar cómo se desea que se realicen los intercambios de datos, si se prefiere hacerlo de forma segura (vía HTTPS) o no.

Estos roles también pueden ser utilizados para determinar si se tiene acceso o no a determinados métodos de la EJB, ofreciendo más granularidad a la hora de definir los permisos de la aplicación.Para ello, primero se deben declarar en la clase:

```
@DeclareRoles({"Administrator", "Manager", "Employee"})
public class Calculator { ... }
```

Y después indicar cuáles de ellos pueden acceder a los métodos de la EJB

```
@RolesAllowed("Administrator")
public void setNewRate(int rate) { ... }
```

Los datos para la autenticación del cliente podrán venir de cuatro orígenes:

- BASIC: el cliente enviaría sus credenciales a través de un header HTTP.
- DIGEST: igual que en BASIC, salvo que en lugar de la contraseña se envía su "digest".
- CLIENT-CERT: tanto el cliente como el servidor exigen certificados digitales para comprobar la identidad. Requiere que la conexión se produzca a través de HTTPS.
- FORM: la aplicación debe solicitar los datos al cliente, y entregárselos al servidor. Se debe especificar la página que recogerá estos datos.

Una vez se tienen los datos, hay que hacer saber al servidor la ubicación del almacén de credenciales, para que pueda saber si se le da acceso al usuario o no, y en caso de que se lo dé, en qué grupo estaría. En Glassfish se indica en un "realm". Este debe poseer la información necesaria para conectarse al origen de datos. Su nombre deberá aparecer en el descriptor de la aplicación.

En cuanto a herramientas para la generación de certificados se refiere, Java pone a nuestra disposición keytool. Permite la gestión de claves, generación de certificados y firma de los mismos, además del manejo y creación de los almacenes de claves jks, utilizados por el servidor de Glassfish.

2.1.2.4 Apache Shiro

Se trata de un framework que hace hincapié en la seguridad y en la facilidad de uso. Puede realizar tareas de autenticación, autorización, criptografía y gestión de sesiones.

Su origen está en JSecurity, creado en el 2004 cuando se buscaba un framework de seguridad que operara bien a nivel de aplicación. En 2008 quedó en manos de Apache, que le cambió el nombre y lo lanzó en el 2010.

En lo que a autenticación se refiere, hay que remarcar un par de características interesantes:

- Cualquier cosa que se vaya a hacer está basada en el usuario que lo ejecuta, al que llama "Subject".

- Tiene un "recuérdame" integrado, de forma que recordará a los usuarios que vuelvan a la aplicación.

2.1.2.5 Spring security

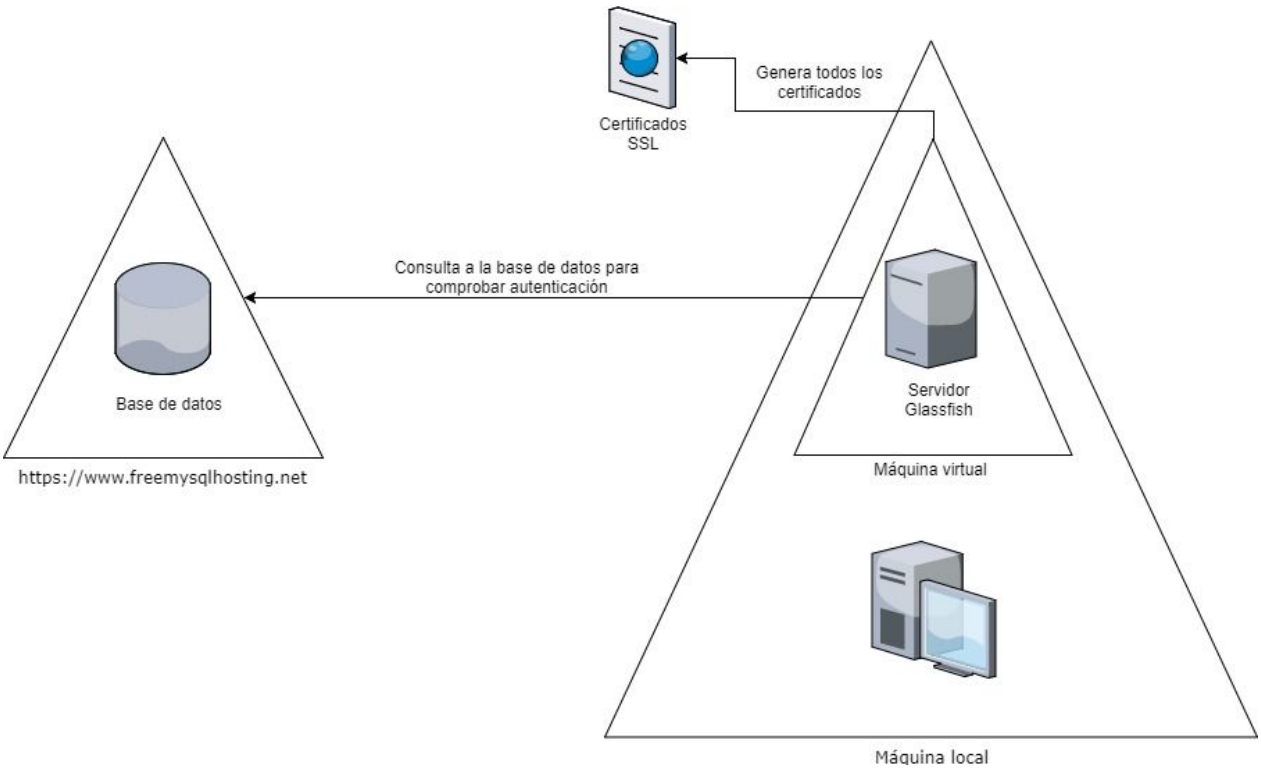
Ofrece una serie de servicios especializados en la seguridad. En este caso el punto más remarcable es la adaptabilidad que tiene para cumplir con necesidades específicas. Su objetivo principal es proporcionar el máximo soporte a los usuarios de Spring Framework, que es un framework de desarrollo para Java EE. Se puede integrar con una gran variedad de modelos de autenticación.

2.1.3 Conclusiones

En resumen: las amenazas son muchas y las soluciones a las mismas no siempre son sencillas, aunque las herramientas disponibles para su paliación son muchas. De todas formas, es necesario saber cómo resolvería Java por su cuenta todas estas máculas, sin recurrir a frameworks externos. En base a ello se plantean a continuación las propuestas.

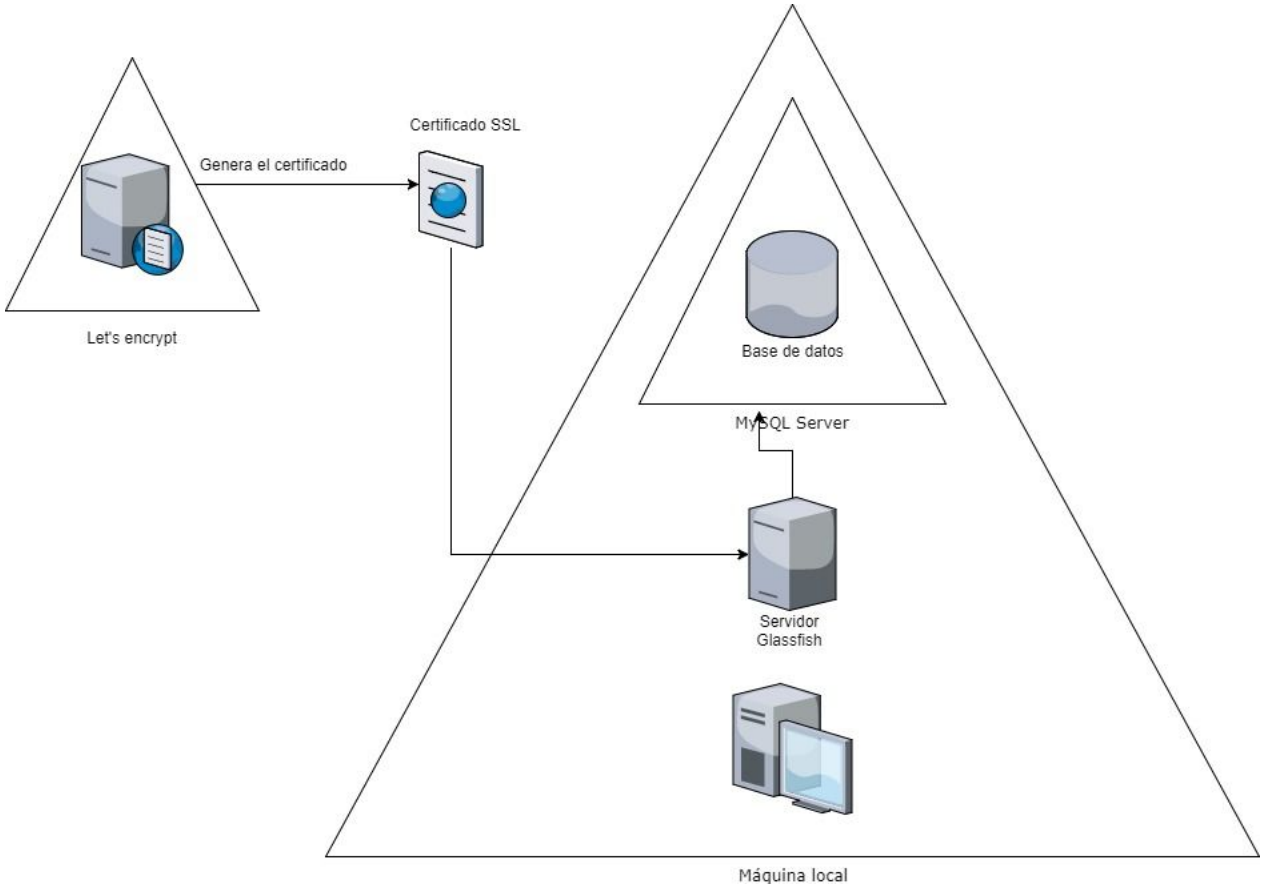
2.2 Estudio de propuestas

2.2.1 Propuesta 1



Esta propuesta pasa por utilizar únicamente las herramientas que Java ofrece para todo lo relacionado con autenticación. Toda la información de los usuarios quedaría en manos de una base de datos externa MySQL. En cuanto a los certificados, se generaría toda la cadena de confianza de certificados en la misma máquina, sin recurrir a AC externas, mediante la herramienta keytool. El servidor Glassfish estaría instalado en una máquina virtual basada en Ubuntu. Esto serviría para entender mejor el proceso de firmado y el funcionamiento de la cadena de confianza.

2.2.2 Propuesta 2



En esta se utilizaría Spring Security para todo lo relacionado con el tema de la autenticación y registro. El registro de usuarios se realizaría igualmente en una base de datos MySQL. En cuanto a certificados, se haría uso de la AC LetsEncrypt, que ofrece certificados gratuitos. La única contrapartida es que esta modalidad solamente permite generar certificados con 90 días de validez.

2.3 Justificación

2.3.1 Estimación de recursos

La segunda propuesta es menos viable, puesto que requeriría disponer de un sitio web previamente, para la emisión del certificado. Las entidades de certificación no emiten certificados para entornos locales.

El hosting de bases de datos no supone un problema, puesto que existen sitio web que lo hacen de forma gratuita (aunque con limitaciones). Dos ejemplos de ellos son los que siguen:

- Free MySQL Hosting, disponible aquí. Completamente funcional, en unos pocos minutos está lista para utilizar y ofrece 5MB de espacio. Lo único que solicitan a cambio es que semanalmente se entre en un enlace que envían por correo electrónico para renovar la suscripción gratuita. En caso de no hacerlo, eliminan el acceso gratuito, y para poder seguir haciendo uso es necesario contratar uno de los planes de pago. El principal problema es que no permite la creación de usuarios con permisos específicos, solamente existe el usuario administrador. Convendría tener la capacidad de tener un usuario con limitaciones.
- 000webhost, disponible aquí. Ofrecen almacenamiento de forma gratuita junto a un sitio web, pero solamente accesible desde ese sitio web, con lo que queda descartado inmediatamente. El servidor Glassfish no tendría acceso.

En cuanto al hosting del servidor Glassfish, no existe una opción específica gratuita:

- A2 Hosting, disponible aquí: ofrece un plan básico desde 5 dólares/mes.
- Anw Hosting, disponible aquí: además de ofrecer el servidor, incluyen también por 79€ al año un dominio web y una base de datos MySQL.

Son opciones interesantes, pero no ofrecen el control que se tendría de instalar el servidor en una máquina virtual local.

2.3.2 Valor educativo

En la primera propuesta es necesario tener un conocimiento más profundo en cuanto a generación de certificados se refiere, ya que no se dejaría en manos de otra entidad la firma del mismo. También es más útil de cara al futuro, cuando se lancen actualizaciones de Java EE. Serviría como punto de comparación entre el antes y el después de la cesión del desarrollo a la Eclipse Foundation.

Esto es algo que no valdría con la segunda, puesto que el desarrollo de Spring Security no tiene nada que ver ni con Oracle ni con Eclipse Foundation. Además,la gestión del certificado quedaría en manos de otra entidad.

2.3.3 Propuesta final

En vista de que la primera aporta más valor educativo, requiriendo una mayor comprensión del protocolo SSL, y está basada únicamente en lo que Java ofrece sin depender de terceros, va a pasar a ser la propuesta final, cuya implementación está explicada a continuación.

3 Implementación

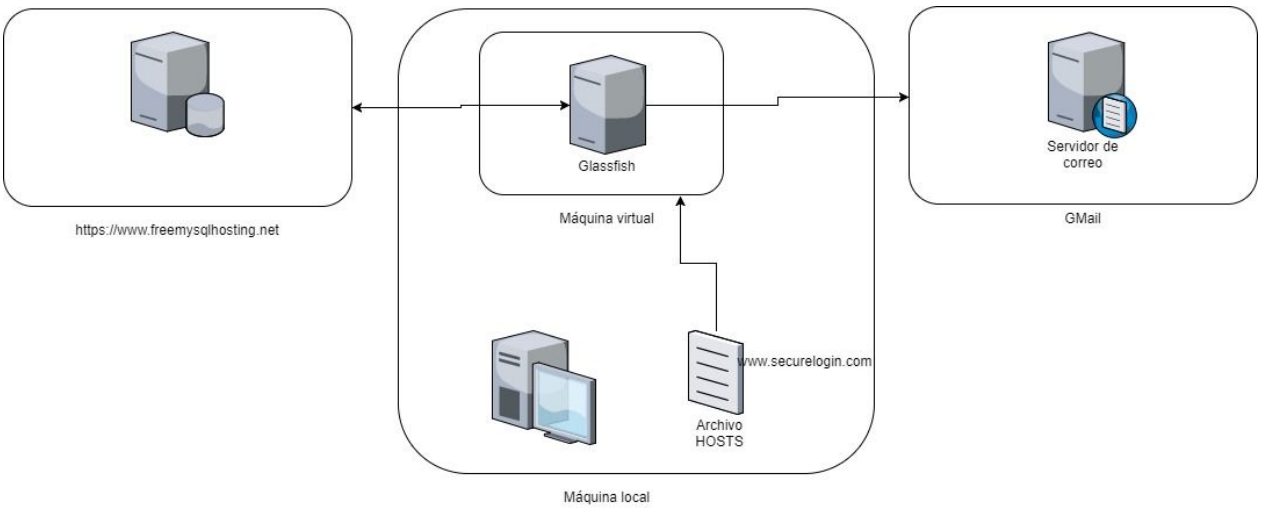
3.1 Entorno de desarrollo

Para el desarrollo, se va a utilizar:

- NetBeans IDE, versión 8.2, disponible para descargar en el siguiente enlace: https://netbeans.org/downloads/
- GlassFish server, versión 4.1.2. Esta versión de GlassFish implementa Java EE, versión 7.
- Mysql server, versión 5.7.18, disponible en el siguiente enlace: https://dev.mysql.com/downloads/windows/installer/5.7.html. Se han instalado los siguientes productos:
    - MySQL Workbench, versión 6.3.9.
- Java Development Kit, versión 8, disponible para descargar en el siguiente enlace: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Oracle VirtualBox, versión 5.2.6, disponible en el siguiente enlace: https://www.virtualbox.org/wiki/Downloads
- Ubuntu 16.04 LTS de 64 bits, disponible en el siguiente enlace: https://www.ubuntu.com/download/desktop

3.2 Implementación práctica

3.2.1 Detalles



El gráfico representa la arquitectura a la que deberá responder la aplicación. A continuación se desarrollará de forma más profunda el proceso de implementación, que pasará por estos pasos:

- Instalación de la máquina virtual de Java.
- Instalación del servidor Glassfish.
- Creación e instalación de certificados.
- Conexión del servidor con la base de datos.
- Creación del realm para la autenticación.
- Conexión con el servidor de correo.

Para empezar: en la máquina local se encontrará la máquina virtual que contendrá el servidor de Glassfish. Más adelante se explicará cómo instalar el servidor en sí y todo lo necesario para que funcione correctamente. Seguidamente, se deberá modificar el archivo HOSTS. Este es un archivo de sistema que contiene duplas nombre de dominio-dirección IP. El sistema utiliza estas entradas para resolver los nombres de dominios que, por ejemplo, se introducen en la barra de direcciones del navegador. No es el único método disponible, pero en este caso sirve para indicar a qué IP deberá consultar nuestro sistema cuando se introduzca la URL del sitio web a desarrollar en el navegador.

Siguiendo con el esquema, hay que explicar la base de datos, que pasa por el registro en www.freemysqlhosting.net para conseguir unas credenciales que serán utilizadas más adelante en la configuración del servidor Glassfish con el propósito de darle acceso a la base de datos, y por tanto a los datos necesarios para autenticar a los usuarios. La configuración de la conexión se explicará a lo largo de los siguientes puntos.

Por otro lado se encuentra el servidor de correo de Gmail. Glassfish no integra este servicio, por lo que es necesario recurrir a uno externo. Este servicio se utilizará tanto para confirmar a los usuarios registrados como para ofrecer la posibilidad de cambiar la contraseña en caso de que sea necesario. Será necesario registrar una cuenta, y posteriormente configurar el acceso desde Glassfish.

Para terminar, hay que recordar que todas las comunicaciones que se llevarán a cabo entre el cliente y la aplicación deben estar cifradas, por lo que será necesario habilitar y configurar el SSL en el servidor, lo cual pasa por la creación e instalación de todos los certificados necesarios para el correcto funcionamiento.

3.2.2 Instalación y configuración del servidor y los servicios necesarios

3.2.2.1 Instalación de la máquina virtual de Java

El servidor de Glassfish requiere de la presencia de la máquina virtual de Java en el sistema en el que se va a ejecutar, para poder ejecutar las aplicaciones web basadas en Java que se desarrollen. Para ello es necesario añadir el repositorio de Oracle, con lo que habrá que introducir los siguientes comandos en una consola:

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
```

Con esto ya es posible instalar Java en el sistema:

```
sudo apt-get install oracle-java8-installer
```

Y añadirlo como variable de sistema para que Glassfish tenga acceso a la instalación, editando el archivo /etc/bash.bashrc para que incluya las siguientes líneas:

```
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0
export PATH=$PATH:$JAVA_HOME/bin:$GLASSFISH_HOME/bin
```

3.2.2.2 Instalación del servidor Glassfish

Nuevamente habrá que hacer uso del terminal para introducir los siguientes comandos:

```
wget download.java.net/glassfish/4.1.2/release/glassfish-4.1.2.zip
apt-get install unzip
unzip glassfish-4.0.zip -d /opt
```

Unzip es necesario en caso de que no esté instalado un programa que gestione archivos comprimidos. Esa secuencia de comandos instalará el servidor bajo la carpeta /opt. Además, resultará conveniente una referencia a la carpeta de instalación de glassfish en el archivo /etc/bash.bashrc para poder hacer uso de sus comandos en cualquier ubicación:

```
export GLASSFISH_HOME=/otp/glassfish/bin
```

Una vez seguidos estos pasos, se puede iniciar el servidor:

```
asadmin start-domain
```

A continuación, es recomendable cambiar la contraseña a través de la cual se accede a la consola de administración:

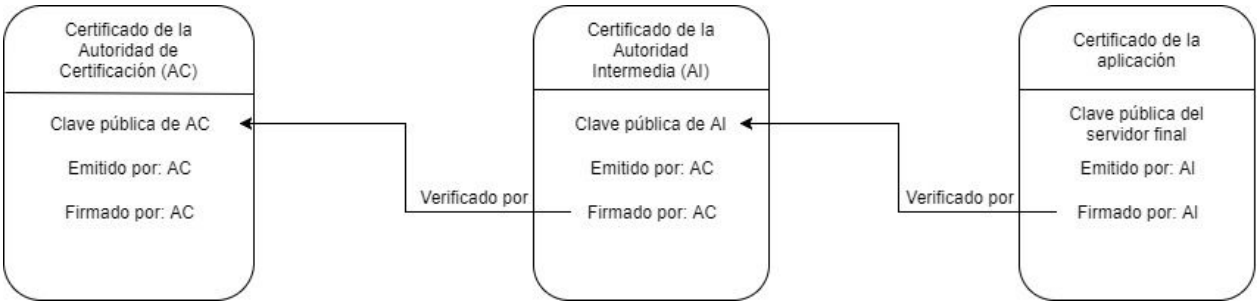
```
asadmin change-admin-password
```

Y habilitar el acceso desde el navegador (a través del puerto 4848):

```
asadmin enable-secure-admin
```

Una vez introducidos todos estos comandos, es necesario reiniciar el servidor de glassfish. A partir de este momento es posible configurarlo.

3.2.2.3 Creación e instalación de certificados



Los certificados a generar seguirán esta estructura, explicada en el punto 2.1.1.1 SSL/TLS, donde existe la figura de Autoridad Certificadora, Autoridad Intermedia y el certificado final.

El primer certificado que debe crearse es el que corresponderá a la Autoridad Certificadora Raíz (CAFran). Para ello, primero se debe generar un par de claves:

```
Keytool -genkeypair -alias ac -keyalg RSA -keystore ac.jks -keysize 2048
-storepass contra -sigalg SHA256withRSA
```

```
¿Cuáles son su nombre y su apellido?
  [Unknown]:  ac.securelogin.com
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  tfg
¿Cuál es el nombre de su organización?
  [Unknown]:  tfg
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  Alcoy
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:  Alicante
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  ES
¿Es correcto CN=ac.securelogin.com, OU=tfg, O=tfg, L=Alcoy, ST=Alicante, C=ES?
  [no]:  si

Introduzca la contraseña de clave para <ac>
        (INTRO si es la misma contraseña que la del almacén de claves):
```

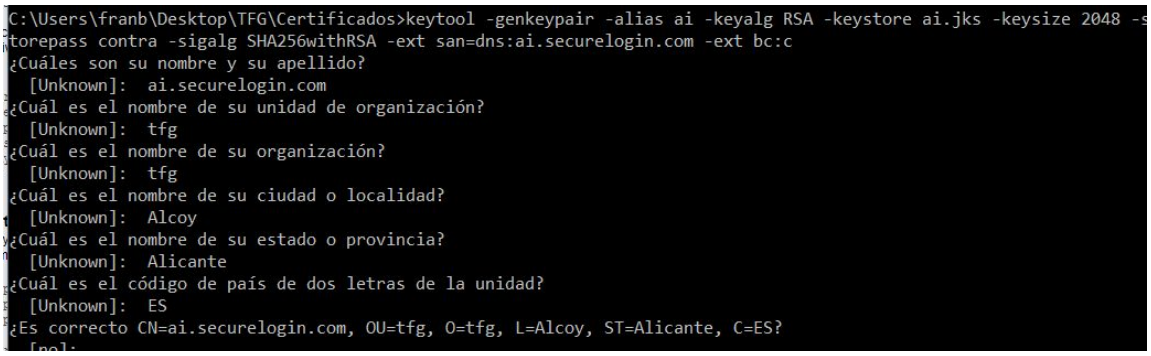
Esto ha generado un archivo ac.jks (un almacén de claves) en el que están almacenadas las claves pública y privada para la Autoridad Certificadora Raíz, bajo el alias indicado. El almacén está protegido con la contraseña "contra", al igual que las claves. También se ha especificado que el algoritmo para firmar utilice SHA256. El siguiente paso consiste en generar un certificado a partir de este par de claves:

```
Keytool -exportcert -alias ac -file ac.cer -keystore ac.jks
```

Este certificado está firmado con la clave privada de la Autoridad Certificadora Raíz, y es el que se debe importar en el navegador como "Entidad de certificación raíz de confianza". Al comando se le debe indicar el almacén de claves en el que está el par y el alias del mismo.

El siguiente paso es crear un certificado firmado por el anterior para la Autoridad Intermedia. Primero hay que generar su par de claves:

```
Keytool -genkeypair -alias ai -keyalg RSA -keystore ai.jks -keysize 2048
-storepass contra -sigalg SHA256withRSA -ext bc:c
```



El parámetro "-ext bc:c" se incluye para permitir que el certificado resultante pueda emitir otros certificados. El siguiente paso no es extraer el certificado, puesto que es necesario que esté firmado por el de la Autoridad Raíz. Hay que generar una solicitud de certificado (para la solicitud hay que utilizar la extensión de archivo csr):

```
keytool -certreq -keystore ai.jks -storepass contra -alias ai -file ai.csr
```

Y, posteriormente, firmarla con la clave privada de la Autoridad Raíz:

```
keytool -gencert -keystore ac.jks -storepass contra -alias ac -infile ai.csr
-outfile ai.cer
```

Si se solicita la información de este certificado, se puede ver lo siguiente:

```
keytool -printcert -file ai.cer
```

```
Propietario: CN=ai.securelogin.com, OU=tfg, O=tfg, L=Alcoy, ST=Alicante, C=ES
Emisor: CN=ac.securelogin.com, OU=tfg, O=tfg, L=Alcoy, ST=Alicante, C=ES
Número de serie: 7b3cc5fa
Válido desde: Sat Nov 04 23:02:25 CET 2017 hasta: Fri Feb 02 23:02:25 CET 2018
Huellas digitales del Certificado:
         MD5: E5:49:2E:81:42:ED:06:CF:FF:95:D4:8F:55:11:46:06
         SHA1: CF:DE:AA:EA:5C:7A:27:96:97:95:91:0E:B2:5C:BD:30:94:2D:3B:A8
         SHA256: B9:20:24:81:C1:99:69:99:29:ED:A2:DE:ED:A8:BC:C8:A2:79:40:F3:02:FB:7A:E1:47:F4:E5:ED:77:2B:DF:DA
         Nombre del Algoritmo de Firma: SHA256withRSA
         Versión: 3

Extensiones:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: CF F0 A6 80 1B 38 F1 19   DD 3E 99 8E 67 6B 85 6D  .....8...>..gk.m
0010: EC 51 3B 1F                                        .Q;.
]
]

#2: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C0 46 7A 64 CD B1 41 58   8E FD DD 57 33 CF 02 9F  .Fzd..AX...W3...
0010: 72 D0 35 4E                                        r.5N
]
]
```

Se puede observar que el emisor es la Autoridad Raíz, y que ha sido emitido para la Autoridad Intermedia.

Por último, el certificado final. El proceso es el mismo. Se genera el par de claves:

```
Keytool -genkeypair -alias cf -keyalg RSA -keystore cf.jks -keysize 2048
-storepass contra -sigalg SHA256withRSA
```

```
¿Cuáles son su nombre y su apellido?
  [Unknown]:  www.securelogin.com
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  tfg
¿Cuál es el nombre de su organización?
  [Unknown]:  tfg
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  Alcoy
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:  Alicante
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  ES
¿Es correcto CN=www.securelogin.com, OU=tfg, O=tfg, L=Alcoy, ST=Alicante, C=ES?
  [no]:  si

Introduzca la contraseña de clave para <cf>
        (INTRO si es la misma contraseña que la del almacén de claves):
```

Se crea la solicitud:

```
Keytool -certreq -keystore cf.jks -storepass contra -alias cf -file cf.csr
```

Y se firma:

```
Keytool -gencert -keystore ai.jks -storepass contra -alias ai infile cf.csr
-outfile cf.cer
```

Tras generar los certificados, es necesario instalarlos en el servidor. Para ello, tomaremos los archivos "cf.cer", "cf.jks", "ai.cer" y "ac.cer" y los copiaremos a la siguiente carpeta (para facilitar el trabajo): ruta_instalacion_glassfish/glassfish/domains/domain1/config. Hay dos archivos que van a ser modificados en esta carpeta: el almacén de claves "keystore.jks" y "cacerts.jks". El primero incluye los pares claves que GlassFish incorpora por defecto, y el segundo contiene una serie de certificados raíz de confianza. El primer paso será importar el par de claves

```
keytool -importkeystore -srckeystore cf.jks -destkeystore keystore.jks
```

Para que el servidor la pueda utilizar, habrá que cambiar su contraseña para que coincida con la del almacén de claves, que es también la contraseña maestra del servidor (por defecto: changeit):

```
keytool -keypasswd -new changeit -alias cf -keystore keystore.jks
```

A continuación se deberá importar el certificado de la Autoridad Raíz, tanto al archivo cacerts.jks como al keystore.jks:

```
keytool -import -trustcacerts -keystore cacerts.jks -file ac.cer -alias ac
keytool -import -trustcacerts -keystore keystore.jks -file ac.cer -alias ac
```

En ambos casos solicitará confirmación, puesto que no es un certificado emitido por una autoridad certificadora que tenga almacenada.

Seguidamente se importará el correspondiente a la Autoridad Intermedia de la misma forma (sustituyendo el término “ac” por “ai”), y para terminar se repite el proceso para el certificado final, con la diferencia de que este último solamente estaría en el archivo “keystore.jks”.

Después de este proceso es necesario indicarle a GlassFish qué certificado y clave debe utilizar. Para ello hay que abrir la consola de administración del servidor y buscar en la barra lateral la pantalla de configuración del “http-listener-2” (es el que corresponde por defecto a las conexiones https), está situada en: Configuration - server-config - HTTP service - Http listeners - http-listeners-2. Una vez en la pantalla adecuada, hay que dirigirse a la pestaña de “SSL” y ahí indicar primero el alias de la clave/certificado y el archivo en el que se encuentra, sustituyendo el que viene por defecto (s1as). Para finalizar, será necesario dirigirse al archivo “domain.xml” (situado en la misma carpeta que los archivos cacerts.jks y keystore.jks) y sustituir todas las ocurrencias que aparezcan de “s1as” por el nombre del alias del certificado y el par de claves finales (“cf” en este caso). Se reinicia el servidor y el proceso de instalación de certificados ha llegado a su fin.

3.2.2.3 Almacén de credenciales

Las credenciales de los usuarios quedarán guardadas en una base de datos MySQL, como se ha indicado anteriormente. La base se llamará “securelogin”, y tendrá dos tablas: “usuarios” (con los campos “email” y “password”) y “usuarios_grupos” (campos “email” y “grupo”). Puesto que de las contraseñas se guardará un Digest SHA256, que produce una cadena de 64 caracteres, será necesario que el campo “password” sea una cadena (varchar) de 64 caracteres. [Anexo, 8.2, código SQL].

3.2.2.4 Conexión del servidor con la base de datos

Primero es necesario conseguir una cuenta en www.freemysqlhosting.net. El proceso de registro es sencillo y rápido, solamente es necesario una dirección de email y una contraseña. Se solicita completar el proceso de registro mediante un enlace enviado por correo electrónico. Cuando esté todo listo, se enviará a la dirección indicada un correo con los datos necesarios para conectarse a la base de datos, además de un panel para gestionarla.

Hi

Your account number is: 265507

Your new database is now ready to use.

To connect to your database use these details

Server: sql11.freemysqlhosting.net
Name: sql[redacted]
Username: sql1[redacted]
Password: j[redacted]
Port number: 3306

Para poder conectar con la base de datos, primero hay que descargar el conector que ofrece MySQL, disponible en esta página: https://dev.mysql.com/downloads/connector/j/ . Una vez se ha descargado y descomprimido, hay que situar el archivo jar en la siguiente carpeta: ruta de instalación de GlassFish - glassfish - domains - dominio que se esté utilizando - lib - ext.

El siguiente paso consiste en la creación de una JDBC Connection Pool. Para ello, es necesaria la consola de administración. La pantalla para crearlas se encuentra en: Resources - JDBC - JDBC Connection Pools. Una vez ahí, se le da a "new" y se rellenan los campos [Anexo, 8.1, Creación de JDBC Connection Pool]:

New JDBC Connection Pool (Step 1 of 2)

Identify the general settings for the connection pool.

General Settings

Pool Name: *	MySQLPool
Resource Type:	javax.sql.DataSource Must be specified if the datasource class implements more than 1 of the interface.
Database Driver Vendor:	MySql Select or enter a database driver vendor
Introspect:	Enabled If enabled, data source or driver implementation class names will enable introspection.

Una vez creada, se indican los datos de conexión a base de datos. Posteriormente, habría que crear un JDBC Resource que se utilice la MySQLPool. Esto se puede realizar en: Resources -

JDBC - JDBC Resources. Solamente habría que indicar el nombre JNDI (“bbddtfg”) y la Resource Pool a la que debe apuntar (“MySQLPool”).

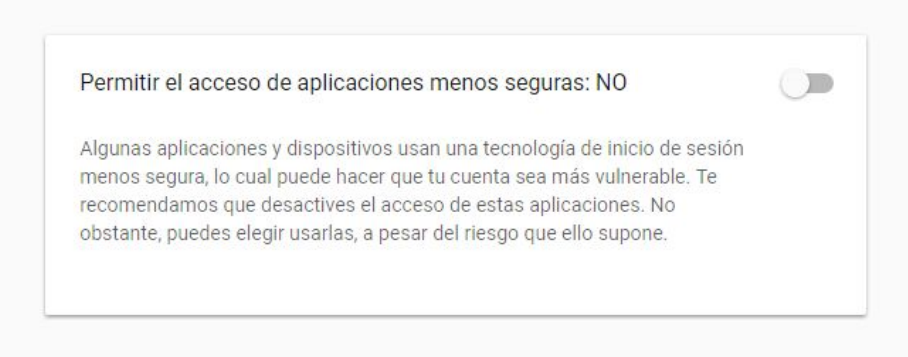
3.2.2.5 Creación del realm

Una vez disponible la conexión a la base de datos, se debe indicar la información necesaria para autenticar a los usuarios contra la base. Esta tarea se puede realizar yendo a la pantalla correspondiente, situada en: Configuration - server-config - Security - Realms - New. Se deben rellenar los datos de forma que coincida con lo que se ha creado en la base de datos:

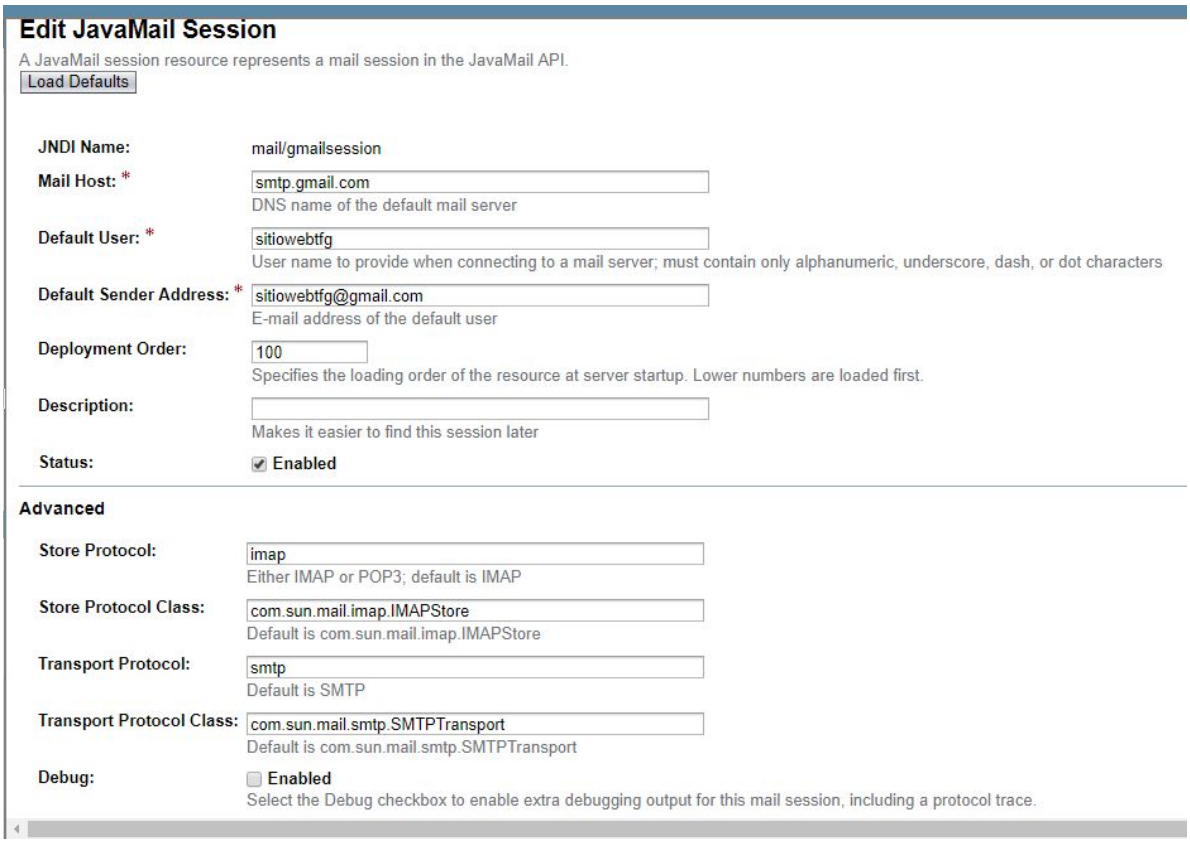
Se debe indicar el “Digest Algorithm” como SHA-256, puesto que es como se van a guardar las contraseñas.

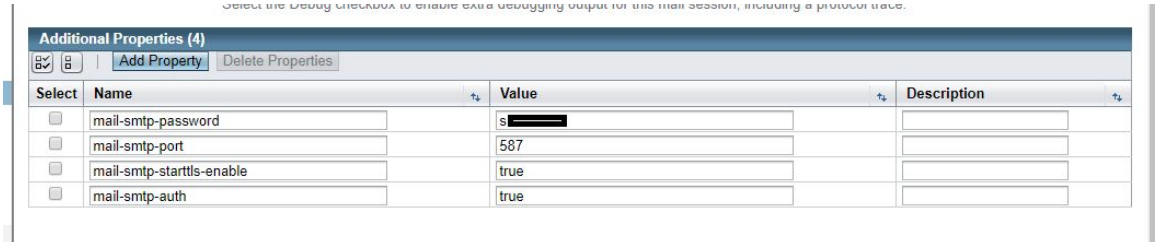
3.2.2.6 Conexión con el servidor de correo

El primer paso consiste en crear una cuenta en Gmail. Para ello es necesario dirigirse a https://www.google.com/gmail/ e indicar los datos necesarios. Una vez creada, será necesario dirigirse a https://myaccount.google.com/intro/security y activar la siguiente opción:



Así Google permitirá a Glassfish hacer uso de los servicios de envío de correos.
Posteriormente habrá que indicarle a Glassfish las credenciales para hacer uso de este servicio:

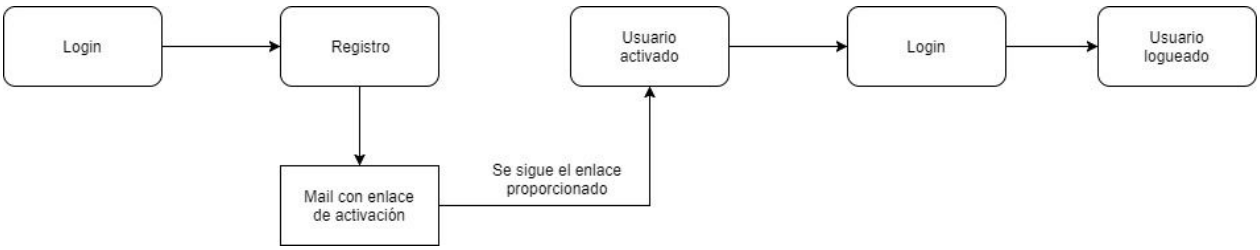




Y una vez configurados los servicios, se puede pasar a hablar del desarrollo de la aplicación.

3.2.3 La aplicación. El proyecto

Una vez Glassfish tiene acceso a todos los recursos necesarios, se puede desarrollar la aplicación haciendo uso de ellos. El flujo para registrarse en la aplicación será el que sigue:



3.2.3.1 Creación del proyecto y archivos de configuración

Para la aplicación se creará en NetBeans un proyecto Maven Web Application.

Hay tres archivos de configuración importantes:

- Glassfish-web.xml: el descriptor de GlassFish. En él se definirá el mapeo grupos(proporcionado por el realm)-roles. Situado en Web pages - WEB-INF. [Anexo, 8.2, glassfish-web.xml]
- Web.xml: el descriptor de la aplicación web. Se ha utilizado para definir los aspectos de seguridad: las limitaciones de los roles, los roles que existirán, la forma de autenticación y la forma de intercambio de datos. Se especifica que todas las conexiones funcionen bajo https, en una restricción que afecte a todas las páginas (*). No se debe especificar un tipo de llamada HTTP concreta, se debe aplicar a todas. Situado en la misma carpeta.[Anexo, 8.2, web.xml]
- Persistence.xml: la unidad de persistencia. Se especifica la forma de conexión a la base de datos, y el JDBC Resource al que debe apuntar. También se referenciarán las clases/entidades que representen a las tablas de la base de datos, una vez creadas. Ubicación: src/main/resources/META-INF. [Anexo, 8.2, persistence.xml]

3.2.3.2 Entidades para la autenticación y procesos de usuario

Se deben crear dos clases de entidad: la de usuarios y la de grupos. La de usuario tendrá además una consulta que permitirá saber si ya hay una entrada en la base de datos con ese email. Estarán situadas en el paquete “entity”. Junto a ellas se crearán otras dos: una dedicada a las solicitudes de activación de una cuenta nueva y otra a la de las peticiones para cambiar la contraseña. [Anexo, 8.2, User.java, Group.Java, Activation.Java, PasswordRestore.Java].

Un usuario tendrá a su disposición varios procesos para interactuar con su cuenta:

- Registro: Permitirá crear una cuenta, con un nombre y una contraseña. Esta contraseña deberá tener un mínimo de seis caracteres, una mayúscula, una minúscula y una cifra. Una vez creada, no podrá iniciar sesión con ella. Para poder hacerlo, deberá seguir un enlace enviado a la dirección de correo utilizada en el registro. Una vez activada la

cuenta, podrá iniciar sesión normalmente. Los usuarios que todavía no han activado su cuenta se almacenan en la base de datos con un grupo que no es el de los usuarios activados, de forma que se pueda hacer la distinción a la hora de iniciar sesión:

```
if (request.isUserInRole("pendingactivation")) {

    ...

} else if (request.isUserInRole("users")) {

    ...

} else {

    ...

}
```

- Restablecimiento de contraseña: se creará en la base de datos una entrada con el usuario, un parámetro generado aleatoriamente para la URL y una fecha y hora de caducidad dos horas posterior a la del momento de la creación. El parámetro se utilizará en el enlace para comprobar si existe una petición y utilizarla para modificar la contraseña. Una vez pasado el periodo de validez, el enlace quedará inútil.

3.2.3.3 UserEJB

Esta clase es la que pondrá en contacto la vista con los datos, permitiendo la creación de un usuario a partir de unos datos de entrada. También llevará a cabo la gestión de la activación y del restablecimiento de la contraseña. En esta clase se hace referencia a otra, "AuthenticationUtils", que contendrá una función para hashear las contraseñas cuando se crean los usuarios. También hará referencia a "MailUtils", que permitirá enviar correos a las direcciones de email facilitadas por los usuarios para iniciar el proceso de activación y de restablecimiento de contraseña. [Anexo, 8.2, UsuarioEJB.java, AuthenticationUtils.java]

3.2.3.4 La aplicación después del registro

Se trata de una aplicación sencilla, que permitirá crear, modificar y eliminar notas, todo ello sin abandonar la página principal. Todo el trabajo quedará reflejado en la base de datos.
Esta página se encontrará fuera del alcance de usuarios que no hayan iniciado sesión y de aquellos que no hayan activado su cuenta.

3.3. Pruebas

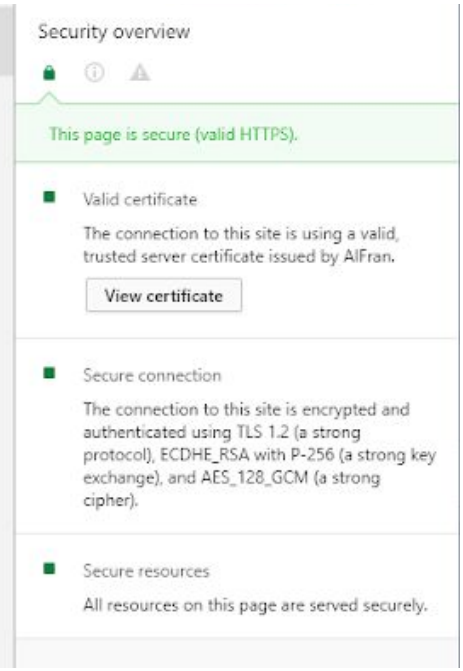
3.3.1 SSL

Es necesario comprobar que la aplicación redireccione automáticamente al puerto seguro y que proporcione los certificados necesarios.

Esta dirección lleva directamente al sitio https:

Y proporciona la cadena de certificados para realizar las verificaciones:

Cuando se comprueba la seguridad del sitio web, se obtienen los siguiente mensajes:



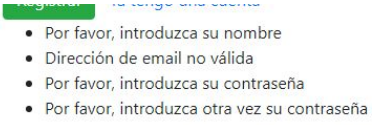
Se puede observar que la conexión es segura y que se dispone de confidencialidad para el intercambio de datos. Esto cumple con uno de los objetivos principales: la conexión segura.

3.3.2 El proceso de registro

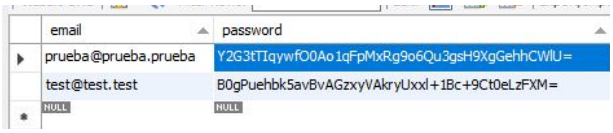
Para registrarse, hay que dirigirse a la página destinada a ello:

Esta página comprobará todos los datos introducidos, verifica que se introduce un email válido y que la contraseña satisface las condiciones anteriormente definidas (mínimo seis caracteres,

una mayúscula, una minúscula y una cifra). También se comprueba si la dirección introducida se ha utilizado para registrar otra cuenta. En caso contrario, muestra los siguientes errores:



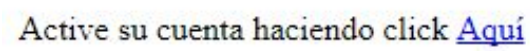
Una vez completado el registro, se puede comprobar en la base de datos que las contraseñas no se están guardando tal y como se introducen, sino que se ha almacenado el digest generado:



Esto cumple con otro de los objetivos.

Tras esto, el proceso de registro continúa con un email enviado a la dirección de correo indicado, con el enlace destinado a activar la cuenta. En caso de no activarla e intentar iniciar sesión con las credenciales correctas, se observa el siguiente mensaje:

En la dirección de correo indicada se encuentra este mensaje:



Y tras seguir el enlace, se llega a la siguiente pantalla:

Tras estos pasos, la cuenta está activada y lista para iniciar sesión.

3.3.3 El restablecimiento de la contraseña

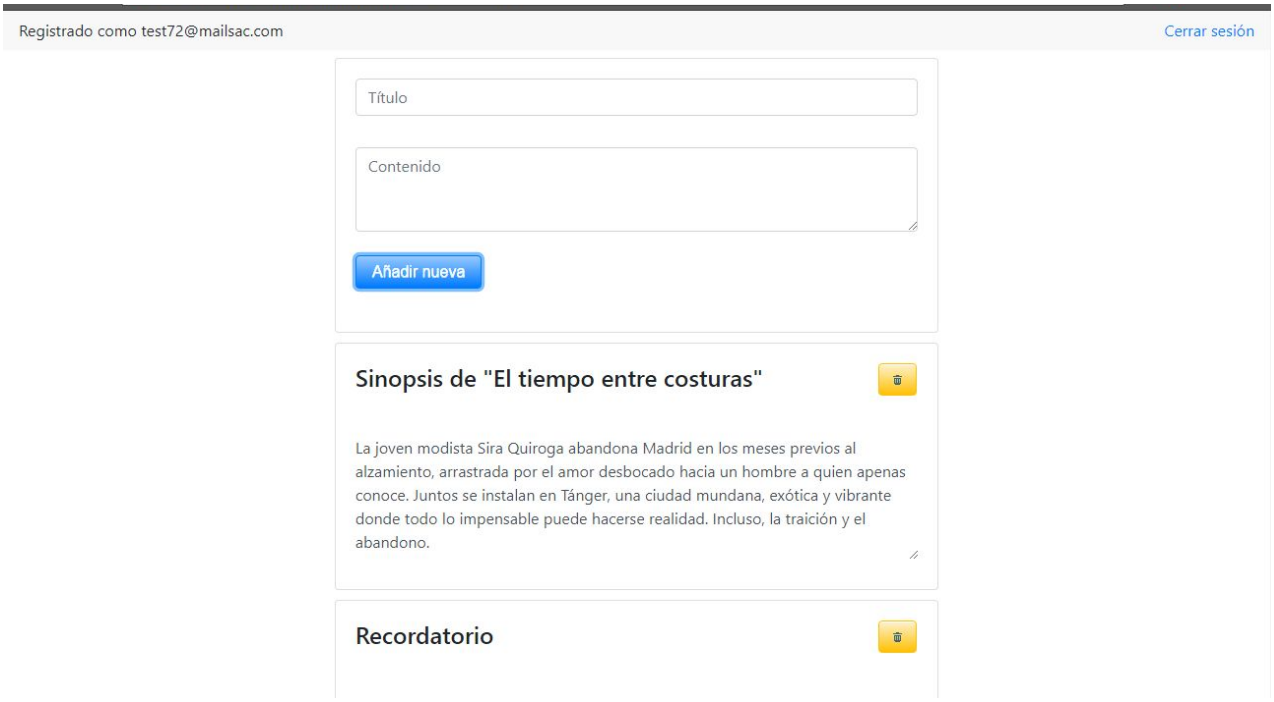
En caso de necesidad, es posible solicitar un restablecimiento de contraseña, indicando la dirección de correo electrónico al que debe llegar la solicitud con el enlace necesario. Cuando se sigue el enlace, se llega a esta pantalla:

Esta pantalla solamente será dejará de ser útil en cualquiera de estos dos casos:

- Si se ha utilizado ese enlace previamente para restablecer la contraseña.
- Si se han superado las dos horas de validez que tiene.

3.3.4 La aplicación

Una vez iniciada la sesión, la aplicación permite guardar notas, editarlas y eliminarlas, sin abandonar la página principal y sin necesidad de recargar:



4 Resultados

La aplicación cumple con los objetivos planteados al inicio del documento: la posibilidad de registrar usuarios guardando sus contraseñas de forma cifrada y el intercambio seguro de los datos.

El resultado de ello es una aplicación web que impide en todos los casos que un usuario externo pueda hacerse con los datos intercambiados entre un usuario registrado y el servidor, puesto que el intercambio se realiza sobre SSL.

También está prevenido uno de los problemas más importantes descritos 2.1.1.4: la inyección SQL. Ninguna de las consultas utilizadas es susceptible de ser utilizada para ese objetivo, al no utilizar ningún parámetro introducido por los usuarios como parte del código SQL.

5 Conclusiones

5.1 Conclusiones personales

Tras el estudio y la implementación, ha quedado claro que la seguridad en la web es algo tan necesario como delicado de implementar. Sin embargo, la recompensa es valiosa.

La investigación ha dado como resultado una guía para desarrollar incluso aplicaciones a escala empresarial, de una forma que no pongan en peligro aquellos datos intercambiados.

Asimismo, se ha dejado ver la potencia que puede llegar a tener el sistema de autenticación de Java Enterprise Edition, y el nivel de granularidad que ofrece es increíble. El poder especificar roles y permisos a nivel hasta de usuario da un control inmenso sobre el acceso a la aplicación.

También hay que remarcar la decisión de Oracle de ceder el desarrollo a la Eclipse Foundation. Las posibilidades que ofrece este framework son numerosas, y que su desarrollo esté en manos de una comunidad de desarrolladores como la de esta fundación no hace más que aumentar su valor.

5.2 Futuras líneas de desarrollo

Una vez esté completada la actualización del framework, sería interesante realizar el mismo desarrollo que en este proyecto, pero con las nuevas herramientas que estén disponibles, haciendo hincapié en las diferencias encontradas.

También se puede realizar utilizando las alternativas mencionadas en el proyecto: Spring Security y Apache Shiro.

De la misma forma se pueden investigar nuevas formas de generar hashes de contraseña, como bcryp o PBKDF2.

Existe por otro lado la posibilidad de ampliar la funcionalidad de la aplicación: compartir notas entre usuarios, añadir imágenes, personalizar el estilo de las notas por separado o reordenarlas.

6 Bibliografia

- D. Kolar, J. Branam, J. Rubinoff, *"Securing a Web Application in NetBeans IDE"*, NetBeans.org, online
- NetBeans.org, *"Building Secure Enterprise Beans in Java EE"*, online
- Solingest, *"Almacenar contraseñas en MySQL"*, online
- Apache SHIRO, online
- Web Application Security Fundamentals, online
- Herramienta para crear los diagramas: draw.io

SSL/TLS

- Secure Web Communications, online
- What makes a site secure?, online
- What is HTTPS?, online
- Certificate authority, online
- Certificate Authorities & trust hierarchies, online
- Public key infrastructure, online
- HTTP to HTTPS, What is a HTTPS Certificate? Online
- How does HTTPS actually work? Online
- An overview of the SSL or TLS handshake, online
- Cipher Suites in TLS/SSL, online
- Secure Web Communications, online
- The First Few Milliseconds of an HTTPS Connection, online
- El Cifrado Web (SSL/TSL), por Dante Odín Ramírez López y Carmina Cecilia Espinosa Madrigal, online
- Self-Signed, Root CA and Intermediate CA Certificates, por Kaushal Kumar Panday, online

Contraseñas

- Serious Security: How to store your users' passwords safely, online
- UK adults taking online password security risks, online
- Why passwords should be hashed, online

Hash

- Handbook of Applied Cryptography, por Alfred J. Menezes, Paul C. van Oorschot y Scott A. Vanstone, editorial: CRC Press. También disponible online
- Announcing the first SHA1 collision, online
- SHA-1 deprecation countdown, online
- Gradually Sunsetting SHA-1, online
- SHA-3: El nuevo estándar de Hash aprobado por el NIST, online
- Announcing Approval of Federal Information Processing Standard (FIPS) 202, SHA-3 Standard, online

Java EE en la actualidad

- Java EE 8 Overview, online
- The Java EE Tutorial, Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito,
- Java EE Development with Eclipse, Ram Kulkarni. Editorial: Packt Publishing.
- What is tackled in the Java EE Security API, online
- Revamping the Security API in Java EE 8, online
- Con7631 Finally, the Java EE Security API JSR 375, online
- Light at the End of the Long Tunnel for Java EE 8, online
- Oracle sigue liberando Java EE, ahora bajo la Eclipse Foundation, online
- Introducing EE4J: Java EE takes first steps into Eclipse Foundation, online
- The Eclipse Enterprise for Java Project Top Level Project Charter, online

Qué ofrece Java EE

- Java Authentication and Authorization Service (JAAS) Reference Guide, online
- Java EE whitepaper, online
- Layers of a Standard Enterprise Application, online
- Layered Architecture is Good, online
- JavaServer Faces Overview, online
- JavaServer Pages Overview, online
- Introducción a la tecnología EJB, online

- JPA (Java Persistence Api), online
- Java SE Technologies - Database, online

Seguridad en Java EE

- Security in Java EE Applications, online
- Building Secure Enterprise Beans in Java EE, online
- Using the JDBC Realm for User Authentication, online
- keytool - Key and Certificate Management Tool, online

Apache shiro

- Página web, online

Spring security

- Página web, online

Ataques comunes a aplicaciones web

- The 10 Most Important Security Controls Missing in JavaEE, online
- The Ten Most Critical Web Application Security Risks, online
- The ten most critical web application security vulnerabilities for java enterprise applications (2007), online

7 Acrónimos

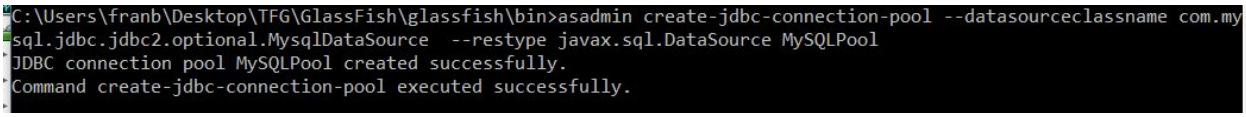
- AC: Autoridad Certificadora
- EJB: Enterprise JavaBean
- Java EE: Java Enterprise Edition
- JDBC: Java Database Connection
- JSR: Java Specification Request
- JAAS: Java Authentication and Authorization Service
- JASPIC: Java Authentication Service Provider Interface for Containers
- JACC: Java Authorization Contract for Containers
- PKI: Public Key Infrastructure (Infraestructura de Clave Pública)
- SSL: Secure Sockets Layer
- TLS: Transport Layer Security
- XSS: Cross-Site Scripting

8 Anexos

8.1 Resolución de problemas

Creación de JDBC Connection Pool

Si el servidor da error al crear una Connection Pool desde la consola de administración, se puede crear primero desde línea de comandos y después ya se podría configurar de forma gráfica. Para ello, se puede ejecutar el comando que sigue:



Tras esto, hay que reiniciar el servidor. Si no aparece, hay que darle al botón “New” y aparecerá. Otra opción es definirla a mano en el archivo domain.xml, situado en la carpeta “config” del dominio que se está utilizando.

Utilización de las herramientas de Java.

Al utilizar la herramienta de *keytool* se pueden producir un error similar al siguiente:

```
error de herramienta de claves: java.util.IllegalFormatConversionException:
d != java.lang.String
```

Para poder utilizar normalmente la herramienta se debe pasar un parámetro más a todas las introducciones:

```
-J-Duser.language=en
```

8.2 Códigos

Código SQL

Tabla usuarios:

```
CREATE TABLE `usuarios` (
  `identificador` int(11) NOT NULL AUTO_INCREMENT,
  `email` varchar(255) NOT NULL,
  `contrasena` varchar(64) NOT NULL,
  `nombre` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`identificador`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=latin1;
```

Tabla grupos_usuarios:

```
CREATE TABLE `grupos_usuarios` (
  `email` varchar(255) NOT NULL,
  `grupo` varchar(45) NOT NULL,
  PRIMARY KEY (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Tabla activacion_usuario:

```
CREATE TABLE `activacion_usuario` (
  `identificador` int(11) NOT NULL AUTO_INCREMENT,
  `fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `email` varchar(255) NOT NULL,
  `parametro` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`identificador`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1;
```

Tabla reset_contrasena:

```
CREATE TABLE `reset_contrasena` (
  `identificador` int(11) NOT NULL AUTO_INCREMENT,
  `fecha` datetime NOT NULL,
  `email` varchar(255) NOT NULL,
  `parametro` varchar(64) NOT NULL,
  PRIMARY KEY (`identificador`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;
```

Tabla notas:

```
CREATE TABLE `notas` (
  `identificador` int(11) NOT NULL AUTO_INCREMENT,
  `idusuario` int(11) NOT NULL,
  `titulo` varchar(64) NOT NULL,
  `contenido` longtext,
  PRIMARY KEY (`identificador`),
  KEY `idusuario_usuario_idx` (`idusuario`),
  CONSTRAINT `idusuario_usuario` FOREIGN KEY (`idusuario`) REFERENCES
`usuarios` (`identificador`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=60 DEFAULT CHARSET=latin1;
```

Archivos de configuración del proyecto:

Glassfish-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish
Application Server 3.1 Servlet 3.0//EN"
"http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app error-url="">
    <context-root>/</context-root>
    <security-role-mapping>
        <role-name>pendingactivation</role-name>
        <group-name>pendienteactivacion</group-name>
    </security-role-mapping>
    <security-role-mapping>
        <role-name>users</role-name>
        <group-name>usuario</group-name>
    </security-role-mapping>
    <class-loader delegate="true"/>
    <jsp-config>
        <property name="keepgenerated" value="true">
            <description>Keep a copy of the generated servlet class'
java code.</description>
        </property>
    </jsp-config>
    <parameter-encoding default-charset="UTF-8"/>
</glassfish-web-app>
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <display-name>Login seguro</display-name>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
```

```
    </context-param>
    <context-param>
        <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
        <param-value>.xhtml</param-value>
    </context-param>
    <mime-mapping>
        <extension>eot</extension>
        <mime-type>application/vnd.ms-fontobject</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>otf</extension>
        <mime-type>font/opentype</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>ttf</extension>
        <mime-type>application/x-font-ttf</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>woff</extension>
        <mime-type>application/x-font-woff</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>svg</extension>
        <mime-type>image/svg+xml</mime-type>
    </mime-mapping>
    <context-param>
        <param-name>primefaces.FONT_AWESOME</param-name>
        <param-value>true</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>

<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>ActivationServlet</servlet-name>
```

```
<servlet-class>net.tfg.servlets.ActivationServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ActivationServlet</servlet-name>
        <url-pattern>/activation</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            300
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>signin.xhtml</welcome-file>
    </welcome-file-list>

    <!-- SECURITY -->
    <login-config>
        <auth-method>FORM</auth-method>
        <realm-name>jdbcRealm</realm-name>
        <form-login-config>
            <form-login-page>/signin.xhtml</form-login-page>
            <form-error-page>/signin.xhtml</form-error-page>
        </form-login-config>
    </login-config>
    <security-role>
        <description/>
        <role-name>users</role-name>
    </security-role>
    <security-role>
        <description/>
        <role-name>pendingactivation</role-name>
    </security-role>
```

```
<security-constraint>
    <display-name>Restricted to users</display-name>
    <web-resource-collection>
        <web-resource-name>Restricted Access</web-resource-name>
        <url-pattern>/user/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>users</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
<security-constraint>
    <display-name>Restricted to non active users</display-name>
    <web-resource-collection>
        <web-resource-name>Restricted Access</web-resource-name>
        <url-pattern>/pendingactivation/*</url-pattern>

    </web-resource-collection>
    <auth-constraint>
        <role-name>pendingactivation</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
<security-constraint>
    <display-name>General</display-name>
    <web-resource-collection>
        <web-resource-name>General</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

```
</web-app>
```

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="accountsPU" transaction-type="JTA">

<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>jdbc/bbddtfg</jta-data-source>
        <class>net.tfg.entity.Group</class>
        <class>net.tfg.entity.User</class>
        <class>net.tfg.entity.Activation</class>
        <class>net.tfg.entity.PasswordRestore</class>
        <class>net.tfg.entity.Notes</class>
        <properties>
            <property name="eclipselink.logging.level" value="INFO"/>
        </properties>
    </persistence-unit>
</persistence>
```

Clases de entidad para la base de datos:

User.java:

```
package net.tfg.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@NamedQueries({
    @NamedQuery(name = "findUserById", query = "SELECT u FROM User u
WHERE u.email = :email")
})
@Table(name = "usuarios")
public class User implements Serializable {

    @Id
    @Column (name="identificador", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer identifier;


    @Column(name = "email", nullable = false, length = 255)
    private String email;

    @Column(name = "contrasena", nullable = false, length = 64)
    private String password;

    @Column(name = "nombre", nullable = false, length = 30)
```

```
    private String name;

    public User() {
    }

    public User(String email, String password, String name) {
        this.email = email;
        this.password = password;
        this.name = name;
    }

    public Integer getIdentifier() {
        return identifier;
    }

    public void setIdentifier(Integer identifier) {
        this.identifier = identifier;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getName() {
        return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Group.java:

```
package net.tfg.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@NamedQueries({
    @NamedQuery(name = "findUserGroup", query = "SELECT g FROM Group
g WHERE g.email = :email")
})
@Table(name = "grupos_usuarios")
public class Group implements Serializable {

    public static final String USERS_GROUP = "usuario";
    public static final String PENDING_ACTIVATION =
"pendienteactivacion";

    @Id
    @Column(name = "email", nullable = false, length = 255)
    private String email;
```

```
    @Column(name = "grupo", nullable = false, length = 32)
    private String groupname;

    public Group() {
    }

    public Group(String email, String groupname) {
        this.email = email;
        this.groupname = groupname;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getGroupname() {
        return groupname;
    }

    public void setGroupname(String groupname) {
        this.groupname = groupname;
    }

}
```

Activation.java:

```
package net.tfg.entity;

import java.io.Serializable;
import java.sql.Timestamp;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@NamedQueries({
    @NamedQuery(name = "findActivationByParameter", query = "SELECT a
FROM Activation a WHERE a.parameter = :parameter"),
    @NamedQuery(name = "deleteActivationById", query = "DELETE FROM
Activation a WHERE a.id = :id")
})
@Table(name = "activacion_usuario")
public class Activation implements Serializable {

    @Id
    @Column(name = "identificador", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "fecha", nullable = false)
    private java.sql.Timestamp date;

    @Column(name = "email", nullable = false)
    private String email;

    @Column(name = "parametro", nullable = false, length = 64)
    private String parameter;

    public Activation(String email) {
        this.date = new Timestamp(System.currentTimeMillis());
        this.email = email;
    }

    public Activation() {
        this.date = new Timestamp(System.currentTimeMillis());
```

```
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Timestamp getDate() {
        return date;
    }

    public void setDate(Timestamp date) {
        this.date = date;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getParameter() {
        return parameter;
    }

    public void setParameter(String parameter) {
        this.parameter = parameter;
    }

}
```

PasswordRestore.java:

```
package net.tfg.entity;

import java.io.Serializable;
import java.sql.Timestamp;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@NamedQueries({
    @NamedQuery(name = "findResetByParameter", query = "SELECT p FROM
PasswordRestore p WHERE p.parameter = :parameter"),
    @NamedQuery(name = "findResetByEmail", query = "SELECT p FROM
PasswordRestore p WHERE p.email = :email"),
    @NamedQuery(name = "deleteResetById", query = "DELETE FROM
PasswordRestore p WHERE p.id = :id")
})
@Table(name = "reset_contrasena")
public class PasswordRestore implements Serializable{
    @Id
    @Column(name = "identificador", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "fecha", nullable = false)
    private java.sql.Timestamp date;

    @Column(name = "email", nullable = false)
    private String email;
```

```
    //private
    @Column(name = "parametro", nullable = false, length = 64)
    private String parameter;

    public static final String PARAMETER = "param";

    public PasswordRestore(String email) {
        this.date = new Timestamp(System.currentTimeMillis());
        //Dos horas
        this.date.setTime(this.date.getTime() + (((2*60)*60)*1000));
        this.email = email;
    }

    public PasswordRestore() {
        this.date = new Timestamp(System.currentTimeMillis());
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Timestamp getDate() {
        return date;
    }

    public void setDate(Timestamp date) {
        this.date = date;
    }

    public String getEmail() {
        return email;
    }
```

```
    public void setEmail(String email) {
        this.email = email;
    }

    public String getParameter() {
        return parameter;
    }

    public void setParameter(String parameter) {
        this.parameter = parameter;
    }

}
```

Notes.java:

```
package net.tfg.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;

@Entity
@Table(name = "notas")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "findByIdentificador", query = "SELECT n FROM
Notes n WHERE n.id = :id")
    , @NamedQuery(name = "findByUser", query = "SELECT n FROM Notes n
WHERE n.userid = :userid")})
```

```java
public class Notes implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "identificador", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "idusuario")
    private Integer userid;

    @Column(name = "titulo", nullable = false)
    private String title;

    @Column(name = "contenido")
    private String content;

    public Notes() {
    }

    public Notes(String title, String content) {
        this.title = title;
        this.content = content;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getUserid() {
        return userid;
    }
```

```
    public void setUserid(Integer userid) {
        this.userid = userid;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

}
```

Clases EJB:

UserEJB.java:

```
package net.tfg.ejb;

import java.util.logging.Level;
import java.util.logging.Logger;
import java.net.URLEncoder;
import java.sql.Timestamp;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;

import net.tfg.entity.Group;
import net.tfg.entity.User;
import net.tfg.entity.Activation;
import net.tfg.entity.PasswordRestore;
import net.tfg.utils.AuthenticationUtils;
import net.tfg.utils.MailUtils;

@Stateless
public class UserEJB {

    @PersistenceContext(unitName = "accountsPU")
    private EntityManager em;

    public User createUser(User user) {
        try {

user.setPassword(AuthenticationUtils.encodeSHA256(user.getPassword())
);
        } catch (Exception e) {
            Logger.getLogger(getClass().getName()).log(Level.SEVERE,
null, e);
            e.printStackTrace();
```

```
        }

        Group group = new Group();
        group.setEmail(user.getEmail());
        group.setGroupname(Group.PENDING_ACTIVATION);

        Activation activation = new Activation(user.getEmail());
        try {

activation.setParameter(AuthenticationUtils.encodeSHA256(user.getEmai
l(), activation.getDate()));
        } catch (Exception e) {
            Logger.getLogger(getClass().getName()).log(Level.SEVERE,
null, e);
            e.printStackTrace();
        }

        MailUtils mailUtils = new MailUtils();
        try {

mailUtils.SendActivacionMail(URLEncoder.encode(activation.getParamete
r(), "UTF-8"), user.getEmail());
        } catch (Exception e) {
        }

        em.persist(user);
        em.persist(group);
        em.persist(activation);

        return user;
    }

    public User findUserById(String id) {
        TypedQuery<User> query = em.createNamedQuery("findUserById",
User.class);
        query.setParameter("email", id);
        User user = null;
```

```
        try {
            user = query.getSingleResult();
        } catch (Exception e) {
        }
        return user;
    }

    public Group findUserGroup(String id) {
        TypedQuery<Group> query =
em.createNamedQuery("findUserGroup", Group.class);
        query.setParameter("email", id);
        Group group = null;
        try {
            group = query.getSingleResult();
        } catch (Exception e) {

        }
        return group;
    }

    public Activation findUserActivation(String id) {
        TypedQuery<Activation> query =
em.createNamedQuery("findActivationByParameter", Activation.class);
        query.setParameter("parameter", id);
        Activation activation = null;
        try {
            activation = query.getSingleResult();
        } catch (Exception e) {
        }
        return activation;
    }

    public PasswordRestore findUserReset(String id) {
        TypedQuery<PasswordRestore> query =
em.createNamedQuery("findResetByParameter", PasswordRestore.class);
        query.setParameter("parameter", id);
        PasswordRestore pRestore = null;
```

```
        try {
            pRestore = query.getSingleResult();
        } catch (Exception e) {
        }
        return pRestore;
    }

    public void checkAndDeletePasswordRestore(String email) {
        TypedQuery<PasswordRestore> query =
em.createNamedQuery("findResetByEmail", PasswordRestore.class);
        query.setParameter("email", email);
        PasswordRestore prestore = null;
        try {
            prestore = query.getSingleResult();
        } catch (Exception e) {
        }
        if (prestore != null) {
            TypedQuery<PasswordRestore> deletequery =
em.createNamedQuery("deleteResetById", PasswordRestore.class);
            deletequery.setParameter("id", prestore.getId());
            try {
                deletequery.executeUpdate();
            } catch (Exception e) {
            }
        }

    }

    public void activateUser(Activation activation) {
        Group group = findUserGroup(activation.getEmail());
        group.setGroupname(Group.USERS_GROUP);
        em.merge(group);

        TypedQuery<Activation> query =
em.createNamedQuery("deleteActivationById", Activation.class);
        query.setParameter("id", activation.getId());
        try {
```

```
            query.executeUpdate();
        } catch (Exception e) {
        }
    }

    public void resetPasswordRequest(User user) {
        PasswordRestore passwordRestore;
        checkAndDeletePasswordRestore(user.getEmail());

        passwordRestore = new PasswordRestore(user.getEmail());

        try {

passwordRestore.setParameter(AuthenticationUtils.encodeSHA256(user.ge
tEmail(), passwordRestore.getDate()));
        } catch (Exception e) {
            Logger.getLogger(getClass().getName()).log(Level.SEVERE,
null, e);
            e.printStackTrace();
        }

        MailUtils mailUtils;
        mailUtils = new MailUtils();
        try {

mailUtils.SendResetPasswordMail(URLEncoder.encode(passwordRestore.get
Parameter(), "UTF-8"), user.getEmail());
        } catch (Exception e) {
        }
        em.persist(passwordRestore);

    }

    public boolean resetPassword(String resetString, String password)
{
        PasswordRestore pRestore = null;
        pRestore = findUserReset(resetString);
```

```
        if (pRestore != null) {

            if (pRestore.getDate().before(new
Timestamp(System.currentTimeMillis()))) {
                em.merge(pRestore);
                em.remove(pRestore);
                return false;
            }
            User user;
            user = findUserById(pRestore.getEmail());

            try {

user.setPassword(AuthenticationUtils.encodeSHA256(password));
            } catch (Exception e) {

Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, e);
                e.printStackTrace();
            }

            TypedQuery<PasswordRestore> deletequery =
em.createNamedQuery("deleteResetById", PasswordRestore.class);
            deletequery.setParameter("id", pRestore.getId());
            try {
                deletequery.executeUpdate();
            } catch (Exception e) {
            }

            em.merge(user);
            return true;
        } else {
            return false;
        }
    }
}
```

NotesEJB.java:

```
package net.tfg.ejb;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.ejb.Stateless;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import net.tfg.entity.Notes;
import net.tfg.entity.User;

@Stateless
public class NotesEJB {

    @PersistenceContext(unitName = "accountsPU")
    private EntityManager em;

    public void saveNote(Notes note) {
        note.setUserid(getUserId());
        if (note.getId() == null) {
            em.persist(note);
        } else {
            em.merge(note);
        }
    }

    public void deleteNote(Notes note) {
        if (!em.contains(note)) {
            note = em.merge(note);
        }
        em.remove(note);
    }
```

```
    public List<Notes> getList() {
        List<Notes> notes;
        notes = new ArrayList<Notes>();

        TypedQuery<Notes> query = em.createNamedQuery("findByUser",
Notes.class);
        query.setParameter("userid", getUserId());

        try {
            notes = query.getResultList();
        } catch (Exception e) {
        }

        return notes;
    }

    protected EntityManager getEntityManager() {
        return em;
    }

    private Integer getUserId() {
        ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();
        Map<String, Object> sessionMap =
externalContext.getSessionMap();
        User user = (User) sessionMap.get("User");
        return user.getIdentifier();
    }

}
```

Servlets:

ActivationServlet.java

```
package net.tfg.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.net.URLDecoder;
import net.tfg.ejb.UserEJB;
import net.tfg.entity.Activation;

public class ActivationServlet extends HttpServlet {

    @Inject
    private UserEJB userEJB;

    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {

        String paramName = "activate";
        String activationString;
        activationString =
URLDecoder.decode(request.getParameter(paramName), "UTF-8");
        activationString = activationString.replace(" ", "+");
        Activation activation =
userEJB.findUserActivation(activationString);

        if (activation != null) {
            userEJB.activateUser(activation);
            response.setContentType("text/html;charset=UTF-8");
            try (PrintWriter out = response.getWriter()) {
```

```
                out.println("<!DOCTYPE html>");
                out.println("<html>");
                out.println("<head>");
                out.println("<title>Activación de cuenta</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("Su cuenta ha sido activada. Haga click
<a href='./'>aquí</a> para ir a la pantalla de login");
                out.println("</body>");
                out.println("</html>");
            }
        } else {
            try (PrintWriter out = response.getWriter()) {
                out.println("<!DOCTYPE html>");
                out.println("<html>");
                out.println("<head>");
                out.println("<title>Página no encontrada</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("</body>");
                out.println("</html>");
            }
        }
    }

    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
```

```
    }

    @Override
    public String getServletInfo() {
        return "Short description";
    }

}
```

Utilidades:

AuthenticationUtils.java:

```
package net.tfg.utils;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

import javax.xml.bind.DatatypeConverter;

public final class AuthenticationUtils {

    public static String encodeSHA256(String password)
            throws UnsupportedEncodingException,
NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(password.getBytes("UTF-8"));
        byte[] digest = md.digest();
        return
DatatypeConverter.printBase64Binary(digest).toString();
    }

    public static String encodeSHA256(String email, Timestamp date)
            throws UnsupportedEncodingException,
NoSuchAlgorithmException {
        return encodeSHA256(email.concat(date.toString()));
    }

}
```

MailUtils.java:

```
package net.tfg.utils;

import javax.mail.Message;
```

```
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import net.tfg.entity.PasswordRestore;

public class MailUtils {

    public MailUtils() {
    }

    public void SendActivacionMail(String cadena, String mail) throws
NamingException {
        InitialContext ctx = new InitialContext();
        Session session
                = (Session) ctx.lookup("mail/gmailsession");

        Message msg;
        msg = new MimeMessage(session);

        try {
            msg.setSubject("Link de activación de cuenta");
            msg.setRecipient(Message.RecipientType.TO, new
InternetAddress(mail));
            msg.setContent("Active su cuenta haciendo click <a
href='https://www.securelogin.com/activation?activate=" + cadena + "'
target='__blank'>Aquí</a>", "text/html");
            Transport.send(msg);
        } catch (Exception e) {

System.out.println("net.tfg.utils.MailUtils.SendActivacionMail()");
            System.out.println(e.toString());
        }

    }
```

```
    public void SendResetPasswordMail(String cadena, String mail)
throws NamingException {
        InitialContext ctx = new InitialContext();
        Session session
                = (Session) ctx.lookup("mail/gmailsession");

        Message msg;
        msg = new MimeMessage(session);

        try {
            msg.setSubject("Link para cambiar su contraseña");
            msg.setRecipient(Message.RecipientType.TO, new
InternetAddress(mail));
            msg.setContent("Haga click <a
href='https://www.securelogin.com/resetpassword.xhtml?" +
PasswordRestore.PARAMETER + "=" + cadena + "'
target='__blank'>aquí</a> para crear una nueva contraseña.",
"text/html");
            Transport.send(msg);
        } catch (Exception e) {

System.out.println("net.tfg.utils.MailUtils.SendResetPasswordMail()")
;
            System.out.println(e.toString());
        }

    }

}
```

MailValidator.java:

```
package net.tfg.validators;

import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

```
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("emailValidator")
public class EmailValidator implements Validator {

    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-]+(\\."
            + "[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*"
            + "(\\.[A-Za-z]{2,})$";

    private Pattern pattern;
    private Matcher matcher;

    public EmailValidator() {
        pattern = Pattern.compile(EMAIL_PATTERN);
    }

    @Override
    public void validate(FacesContext context, UIComponent component,
Object value) throws ValidatorException {
        matcher = pattern.matcher(value.toString());
        if (!matcher.matches()) {
            throw new ValidatorException(new
FacesMessage(FacesMessage.SEVERITY_ERROR, "Invalid e-mail address",
null));
        }
    }

}
```

Clases de las vistas (managedBeans):

LoginView.java:

```
package net.tfg.managedbeans;

import java.io.Serializable;
import java.security.Principal;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import net.tfg.ejb.UserEJB;
import net.tfg.entity.User;

@ManagedBean
@SessionScoped
public class LoginView implements Serializable {

    private static final long serialVersionUID =
3254181235309041386L;

    private static Logger log =
Logger.getLogger(LoginView.class.getName());

    @Inject
    private UserEJB userEJB;
```

```
    private String email;
    private String password;

    private User user;

    public String login() {
        FacesContext context = FacesContext.getCurrentInstance();
        HttpServletRequest request = (HttpServletRequest)
context.getExternalContext().getRequest();

        try {
            request.login(email, password);
        } catch (ServletException e) {
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_ERROR, "Las credenciales
introducidas no son correctas.", null));
            return "signin";
        }

        if (request.isUserInRole("pendingactivation")) {
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Esta cuenta requiere ser
activada. Consulte su correo.", null));
            this.user = null;

            try {
                request.logout();
            } catch (Exception e) {
            }

            ((HttpSession)
context.getExternalContext().getSession(false)).invalidate();
            return "signin";
        } else if (request.isUserInRole("users")) {
            Principal principal = request.getUserPrincipal();
            this.user = userEJB.findUserById(principal.getName());
```

```
            log.info("Authentication done for user: " +
principal.getName());

            ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();
            Map<String, Object> sessionMap =
externalContext.getSessionMap();
            sessionMap.put("User", user);
            return "/user/home?faces-redirect=true";
        } else {
            try {
                request.logout();
            } catch (Exception e) {
            }
            return "signin";
        }

    }

    public String logout() {
        FacesContext context = FacesContext.getCurrentInstance();
        HttpServletRequest request = (HttpServletRequest)
context.getExternalContext().getRequest();

        try {
            this.user = null;
            request.logout();
            // clear the session
            ((HttpSession)
context.getExternalContext().getSession(false)).invalidate();
        } catch (ServletException e) {
            log.log(Level.SEVERE, "Failed to logout user!", e);
        }

        return "/signin?faces-redirect=true";
    }
```

```
    public String loaded() {
        FacesContext context = FacesContext.getCurrentInstance();
        HttpServletRequest request = (HttpServletRequest)
context.getExternalContext().getRequest();
        Principal principal = request.getUserPrincipal();

        if (principal == null) {
            return "";
        } else {
            return "/user/home?faces-redirect=true";
        }
    }

    public User getAuthenticatedUser() {
        return user;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

RegisterView.java:

```
package net.tfg.managedbeans;

import java.io.Serializable;
import java.util.logging.Logger;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.event.ComponentSystemEvent;
import javax.inject.Inject;

import net.tfg.ejb.UserEJB;
import net.tfg.entity.User;

@ManagedBean
@SessionScoped
public class RegisterView implements Serializable {

    private static final long serialVersionUID =
1685823449195612778L;

    private static Logger log =
Logger.getLogger(RegisterView.class.getName());

    @Inject
    private UserEJB userEJB;

    private String name;
    private String email;
    private String password;
    private String confirmPassword;
```

```
    public void validatePassword(ComponentSystemEvent event) {

        FacesContext facesContext =
FacesContext.getCurrentInstance();

        UIComponent components = event.getComponent();

        UIInput uiInputPassword = (UIInput)
components.findComponent("password");
        String password = uiInputPassword.getLocalValue() == null ?
"" : uiInputPassword.getLocalValue().toString();
        String passwordId = uiInputPassword.getClientId();

        UIInput uiInputConfirmPassword = (UIInput)
components.findComponent("confirmpassword");
        String confirmPassword =
uiInputConfirmPassword.getLocalValue() == null ? ""
                : uiInputConfirmPassword.getLocalValue().toString();

        if (password.isEmpty() || confirmPassword.isEmpty()) {
            return;
        }

        if (!password.equals(confirmPassword)) {
            FacesMessage msg = new FacesMessage("Las contraseñas no
coinciden");
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);
            facesContext.addMessage(null, msg);
            facesContext.renderResponse();
        }
        UIInput uiInputEmail = (UIInput)
components.findComponent("email");
        String email = uiInputEmail.getLocalValue() == null ? "" :
uiInputEmail.getLocalValue().toString();
        String emailId = uiInputEmail.getClientId();
        if (userEJB.findUserById(email) != null) {
            FacesMessage msg = new FacesMessage("Ya existe un usuario
```

```
con esta cuenta de correo.");
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);
            facesContext.addMessage(null, msg);
            facesContext.renderResponse();
        }

    }

    public String register() {
        User user = new User(email, password, name);
        userEJB.createUser(user);
        log.info("New user created with e-mail: " + email + " and
name: " + name);
        return "regdone";
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
```

```
        this.password = password;
    }

    public String getConfirmPassword() {
        return confirmPassword;
    }

    public void setConfirmPassword(String confirmPassword) {
        this.confirmPassword = confirmPassword;
    }
}
```

ForgotView.java:

```
package net.tfg.managedbeans;

import java.io.IOException;
import java.io.Serializable;
import java.net.URLDecoder;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.event.ComponentSystemEvent;
import javax.inject.Inject;
import javax.servlet.http.HttpServletRequest;
import net.tfg.ejb.UserEJB;
import net.tfg.entity.PasswordRestore;
import net.tfg.entity.User;

@ManagedBean
@SessionScoped
public class ForgotView implements Serializable {
```

```
    @Inject
    private UserEJB userEJB;

    private String email;

    private String password;
    private String confirmPassword;
    private String parameterString;

    public String reset() {
        FacesContext context = FacesContext.getCurrentInstance();

        User user;
        user = userEJB.findUserById(email);
        if (user != null) {
            userEJB.resetPasswordRequest(user);
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Se ha enviado un correo con
el link a esa dirección. ", null));
        } else {
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_WARN, "No existe una cuenta con
esa dirección asociada.", null));
        }
        return "";
    }

    public String resetPassword(String param)
            throws IOException {
        FacesContext context = FacesContext.getCurrentInstance();
        HttpServletRequest request = (HttpServletRequest)
FacesContext.getCurrentInstance().getExternalContext().getRequest();
        String resetString =
request.getParameter(PasswordRestore.PARAMETER);
        resetString = URLDecoder.decode(resetString, "UTF-8");
        resetString = resetString.replace(" ", "+");
```

```
        boolean result;
        result = userEJB.resetPassword(resetString, getPassword());

        if (result) {
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Se ha modificado la
contraseña", null));
        } else {
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_WARN, "Ha ocurrido un error. El
enlace no es válido, o bien ya ha sido utilizado.", null));
        }

        return "";
    }

    public void validatePassword(ComponentSystemEvent event) {

        FacesContext facesContext =
FacesContext.getCurrentInstance();

        UIComponent components = event.getComponent();

        UIInput uiInputPassword = (UIInput)
components.findComponent("password");
        String password = uiInputPassword.getLocalValue() == null ?
"" : uiInputPassword.getLocalValue().toString();
        String passwordId = uiInputPassword.getClientId();

        UIInput uiInputConfirmPassword = (UIInput)
components.findComponent("confirmpassword");
        String confirmPassword =
uiInputConfirmPassword.getLocalValue() == null ? ""
                : uiInputConfirmPassword.getLocalValue().toString();

        if (password.isEmpty() || confirmPassword.isEmpty()) {
            return;
```

```
        }

        if (!password.equals(confirmPassword)) {
            FacesMessage msg = new FacesMessage("Las contraseñas no
coinciden");
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);
            facesContext.addMessage(null, msg);
            facesContext.renderResponse();
        }
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getConfirmPassword() {
        return confirmPassword;
    }

    public void setConfirmPassword(String confirmPassword) {
        this.confirmPassword = confirmPassword;
    }

    public String getParameterString() {
        return parameterString;
```

```
    }

    public void setParameterString(String parameterString) {
        this.parameterString = parameterString;
    }

}
```

NotesBean.java:

```
package net.tfg.managedbeans;

import java.io.Serializable;
import java.util.Collections;
import java.util.List;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import javax.faces.context.FacesContext;
import javax.inject.Inject;
import net.tfg.ejb.NotesEJB;
import net.tfg.entity.Notes;

@ManagedBean
@ViewScoped
public class NotesBean implements Serializable {

    @Inject
    private NotesEJB notesEJB;

    private List<Notes> notesList;

    private String title;
    private String content;

    public NotesBean() {
    }
```

```
    public List<Notes> getNotesList() {
        notesList = notesEJB.getList();
        Collections.sort(notesList, (Notes note1, Notes note2) ->
note2.getId() - note1.getId());
        return notesList;
    }

    public void deleteNote(Notes note) {
        notesEJB.deleteNote(note);
    }

    public void saveNote() {
        if (((title == null || content == null) || title.equals(""))
|| content.equals("")) {
            FacesContext context = FacesContext.getCurrentInstance();
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Es necesario rellenar los
dos campos", null));
        } else {
            Notes note = new Notes(title, content);
            notesEJB.saveNote(note);
            this.setContent(null);
            this.setTitle(null);
        }

    }

    public void saveNote(Notes note) {
        notesEJB.saveNote(note);
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
```

```
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

}
```

Vistas

Signin.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough">
    <h:head>
        <title>Iniciar sesión</title>
        <h:outputStylesheet library="css" name="bootstrap.min.css" />
        <h:outputScript name="bootstrap.min.js" library="js" />
    </h:head>
    <h:body>
        <div class="container" style="padding-top: 25px;">
            <h:form class="form-horizontal">
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <h2>Iniciar sesión</h2>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
                            <label class="sr-only"
for="email">Email</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputText id="email"
value="#{loginView.email}" required="true" requiredMessage="Por
favor, introduzca su email" class="form-control"
a:placeholder="nombre@ejemplo.com" a:autofocus="true"></h:inputText>
                            </div>
                        </div>
                    </div>
```

```
                </div>
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
                            <label class="sr-only"
for="email">Contraseña</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputSecret id="password"
value="#{loginView.password}" required="true" requiredMessage="Por
favor, introduzca su contraseña" class="form-control"
a:placeholder="Contraseña" ></h:inputSecret>
                            </div>
                        </div>
                    </div>
                </div>
                <div class="row" style="padding-top: 1rem">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <h:commandButton class="btn btn-success"
action="#{loginView.login}" value="Entrar"></h:commandButton>
                        <span class="float-right">
                            <h:link class="btn btn-link" value="Crear
cuenta" outcome="register" />
                            <h:link class="btn btn-link" value="He
olvidado mi contraseña" outcome="forgotpassword" />
                        </span>
                        <h:messages ></h:messages>
                    </div>
                </div>
            </h:form>
        </div>
    </h:body>
</html>
```

Register.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough">
    <h:head>
        <title>register</title>
        <h:outputStylesheet library="css" name="bootstrap.min.css" />
        <h:outputScript name="bootstrap.min.js" library="js" />
    </h:head>
    <h:body>
        <div class="container" style="padding-top: 25px;">
            <h:form class="form-horizontal">
                <f:event listener="#{registerView.validatePassword}"
type="postValidate" />

                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <h2>Registrar una cuenta nueva</h2>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
                            <label class="sr-only"
for="email">Nombre</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputText id="name"
value="#{registerView.name}" required="true" requiredMessage="Por
favor, introduzca su nombre" class="form-control"
a:placeholder="Nombre" a:autofocus="true"></h:inputText>
```

```
                            </div>
                        </div>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
                            <label class="sr-only"
for="email">Email</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputText id="email"
value="#{registerView.email}" required="true" requiredMessage="Por
favor, introduzca su email" class="form-control"
a:placeholder="nombre@ejemplo.com">
                                    <f:validator
validatorId="emailValidator" />
                                </h:inputText>
                            </div>
                        </div>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
                            <label class="sr-only"
for="email">Contraseña</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputSecret id="password"
value="#{registerView.password}" required="true" requiredMessage="Por
favor, introduzca su contraseña" class="form-control"
a:placeholder="Contraseña" validatorMessage="La contraseña debe
contener como mínimo un caracter en minúscula, uno en mayúscula, una
cifra y un 6 caracteres al menos.">
```

```
                                    <f:validateRegex
pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,20})" />
                                </h:inputSecret>
                            </div>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-md-3"></div>
                        <div class="col-md-6">
                            <div class="form-group has-danger">
                                <label class="sr-only"
for="confirmpassword">Confirme su contraseña</label>
                                <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                    <h:inputSecret id="confirmpassword"
value="#{registerView.confirmPassword}" required="true"
requiredMessage="Por favor, introduzca otra vez su contraseña"
class="form-control" a:placeholder="Confirme su contraseña"
validatorMessage="La contraseña debe contener como mínimo un caracter
en minúscula, uno en mayúscula, una cifra y un 6 caracteres al
menos.">
                                    <f:validateRegex
pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,20})" />
                                </h:inputSecret>
                            </div>
                        </div>
                    </div>

                    <div class="row" style="padding-top: 1rem">
                        <div class="col-md-3"></div>
                        <div class="col-md-6">
                            <h:commandButton class="btn btn-success"
action="#{registerView.register}"
value="Registrar"></h:commandButton>
                            <h:link class="btn btn-link" value="Ya tengo
```

```
una cuenta" outcome="signin" /><br></br>
                        <h:messages ></h:messages>
                    </div>
                </div>
            </h:form>
        </div>
    </h:body>
</html>
```

Regdone.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
          xmlns:h="http://java.sun.com/jsf/html" >
<h:head>
     <title>Registration done</title>
       <h:outputStylesheet library="css" name="bootstrap.min.css" />
</h:head>
<h:body>
     <h3>Your account has been successfully created</h3>
     <br/><br/>
     <h:link value="Sign in with your new account" outcome="signin"
/>
</h:body>
</html>
```

Forgotpassword.xhtml:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough">
    <h:head>
        <title>Contraseña olvidada</title>
        <h:outputStylesheet library="css" name="bootstrap.min.css" />
```

```
        <h:outputScript name="bootstrap.min.js" library="js" />
    </h:head>
    <h:body>

        <div class="container" style="padding-top: 25px;">
            <h:form class="form-horizontal">

                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <h5>Introduzca la dirección de email que
utilizó para registrar su cuenta y se le enviará un correo con un
link para crear una nueva contraseña.</h5>
                    </div>
                </div>

                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
                            <label class="sr-only"
for="email">Email</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputText id="name"
value="#{forgotView.email}" required="true" requiredMessage="Por
favor, introduzca su email" class="form-control"
a:placeholder="Email" a:autofocus="true">
                                    <f:validator
validatorId="emailValidator" />
                                </h:inputText>
                            </div>
                        </div>
                    </div>
                </div>

                <div class="row" style="padding-top: 1rem">
```

```
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <h:commandButton class="btn btn-success"
action="#{forgotView.reset}" value="Enviar link"></h:commandButton>
                        <h:link class="btn btn-link" value="La he
recordado" outcome="signin" /><br></br>
                        <h:messages></h:messages>
                    </div>
                </div>
            </h:form>
        </div>
    </h:body>
</html>
```

Resetpassword.xhtml:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough">
    <h:head>
        <title>Nueva contraseña</title>
        <h:outputStylesheet library="css" name="bootstrap.min.css" />
    </h:head>
    <h:body>
        <div class="container" style="padding-top: 25px;">
            <h:form class="form-horizontal">
                <f:event listener="#{forgotView.validatePassword}"
type="postValidate" />
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
```

```
                            <label class="sr-only" for="email">Nueva
contraseña</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputSecret id="password"
value="#{forgotView.password}" required="true" requiredMessage="Por
favor, introduzca su contraseña" class="form-control"
a:placeholder="Contraseña" validatorMessage="La contraseña debe
contener como mínimo un caracter en minúscula, uno en mayúscula, una
cifra y un 6 caracteres al menos.">
                                    <f:validateRegex
pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,20})" />
                                </h:inputSecret>
                            </div>
                        </div>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <div class="form-group has-danger">
                            <label class="sr-only"
for="confirmpassword">Confirme la nueva contraseña</label>
                            <div class="input-group mb-2 mr-sm-2
mb-sm-0">
                                <h:inputSecret id="confirmpassword"
value="#{forgotView.confirmPassword}" required="true"
requiredMessage="Por favor, introduzca otra vez su contraseña"
class="form-control" a:placeholder="Confirme su contraseña"
validatorMessage="La contraseña debe contener como mínimo un caracter
en minúscula, uno en mayúscula, una cifra y un 6 caracteres al
menos.">
                                    <f:validateRegex
pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,20})" />
                                </h:inputSecret>
                            </div>
                        </div>
```

```
                    </div>
                </div>

                <div class="row" style="padding-top: 1rem">
                    <div class="col-md-3"></div>
                    <div class="col-md-6">
                        <h:commandButton class="btn btn-success"
action="#{forgotView.resetPassword(param['param'])}" value="Cambiar
contraseña" >
                            <f:param name="param"
value="#{param['param']}" />
                        </h:commandButton>
                        <h:messages ></h:messages>
                    </div>
                </div>
            </h:form>
        </div>
    </h:body>
</html>
```

Home.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:a="http://xmlns.jcp.org/jsf/passthrough">
    <h:head>
        <title>Home</title>
        <h:outputStylesheet library="css" name="bootstrap.min.css" />
        <h:outputScript name="bootstrap.min.js" library="js" />
    </h:head>
    <h:body>
        <nav class="navbar fixed-top navbar-light"
style="background-color: #F8F8F8; border-color: #E7E7E7;">
```

```
            <span class="float-right">Registrado como
#{loginView.authenticatedUser.email}</span>
            <h:form >
                <h:commandLink action="#{loginView.logout}"
value="Cerrar sesión" />
            </h:form>
        </nav>
        <h:form id="formlist" class="col-lg-6 col-md-8 col-sm-10
mx-auto" style="margin-top: 70px;">

            <div class="card" id="newnote" style="margin-top: 10px;
margin-bottom: 10px">
                <div class="card-title" style="margin-left: 20px;
margin-right: 20px; margin-top: 20px;">
                    <h:inputText value="#{notesBean.title}"
class="form-control" a:placeholder="Título"
id="newtitle"></h:inputText>
                </div>

                <div class="card-body">
                    <h:inputTextarea value="#{notesBean.content}"
class="form-control" rows="3" a:placeholder="Contenido"
id="newcontent"></h:inputTextarea>
                    <br></br>
                    <p:commandButton action="#{notesBean.saveNote()}"
resetValues="true" value="Añadir nueva" update=":formlist" class="btn
btn-primary btn-xs"></p:commandButton><br></br>
                    <br></br>
                    <h:messages ></h:messages>
                </div>
            </div>

            <h:panelGroup id="toUpdate">
                <ui:repeat value="#{notesBean.notesList}" var="note">
                    <div class="card" style="margin-top: 10px;
margin-bottom: 10px">
                        <div class="card-title" style="margin-left:
```

```
20px; margin-right: 20px; margin-top: 20px;">
                            <span class="float-left">
                                <p:inplace editor="true"
effect="slide" style="margin-top:10px; margin-bottom: 10px;"
class="form-group">
                                    <p:ajax event="save"
listener="#{notesBean.saveNote(note)}" />
                                    <f:facet name="output">
                                        <h4><h:outputText
value="#{note.title}"/></h4>
                                    </f:facet>
                                    <f:facet name="input">
                                        <p:inputText
value="#{note.title}" required="true"/>
                                    </f:facet>

                                </p:inplace>
                            </span>

                            <span class="float-right">
                                <p:commandButton
action="#{notesBean.deleteNote(note)}" class="btn btn-warning btn-xs
" icon="ui-icon-trash" update=":formlist:toUpdate"
></p:commandButton>
                            </span>
                        </div>

                        <div class="card-body">
                            <p:inplace editor="true"
style="margin-top:10px; margin-bottom: 10px;">
                                <p:ajax event="save"
listener="#{notesBean.saveNote(note)}" />
                                <f:facet name="output">
                                    <h:inputTextarea
value="#{note.content}" required="true" rows="5" class="form-control"
style=" margin-bottom: 10px; background-color: #FFFFFF !important;
border:none; padding-left: 0px; padding-right: 0px;"
```

```
readonly="true"/>
                                </f:facet>
                                <f:facet name="input">
                                    <p:inputTextarea
value="#{note.content}" required="true" rows="3" class="form-control"
style=" margin-bottom: 10px;"/>
                                </f:facet>
                            </p:inplace>
                        </div>
                    </div>
                </ui:repeat>
            </h:panelGroup>
        </h:form>
    </h:body>
</html>
```