naVerificar3.1037

Plataforma de firma digital para la Universidad de Málaga.

Juan Antonio Pérez Ariza
Escuela Técnica Superior Ingeniería Informática
Universidad de Málaga

Profesor:
Isaac Agudo Ruíz
Departamento Lenguajes y Ciencias de la Comunicación
Universidad de Málaga

5 de septiembre de 2012

Índice general

Capítulo 1

Introducción.

En este capítulo primero de la memoria vamos a explicar las motivaciones que nos llevaron a pensar en realizar dicho proyecto, los objetivos que nos marcamos al principio cuando lo diseñamos, los materiales usados y la organización de esta memoria.

1.1. ¿Por qué decidimos hacer el proyecto?

Al igual que muchos estudiantes de la Universidad de Málaga, yo suelo comer habitualmente en la cafetería de la facultad y hay mucha gente que se molesta cuando le piden que firme el papel con el que se lleva el recuento de los estudiantes que comen en las cafeterías para el descuento por ser estudiante. Además de dicho incoveniente hay un par de problemas más, que son lo molesto que es tener que firmar todos los días o habitualmente y las colas que se forman al tener que rellenar el nombre y la firma, por eso se decidió hacer una aplicación para terminales móviles con la que agilizar todo el proceso de firma y control de estudiantes, mediante la lectura de un código QR que tendría la información necesaria.

A medida que avanzaba el proyecto se vio que se podía ampliar no solo al comedor, si no también a alquiler de pistas o cualquier documento necesario en la Universidad de Málaga.

Además a mi me gusta la seguridad informática y vi en este proyecto una buena forma de aprender más sobre criptografía, particularmente la de clave pública, además vi una buena forma de aprender a programar para terminales android, debido al gran auge que tienen en este momento, y a crear aplicaciones web de las que no tenía ninguna idea. Al principio la aplicación web se penso en hacer directamente en java sin ninguna ayuda, pero se descartó ante la dificultad de encontrar un servicio de hosting gratuito, por lo que se decidió cambiar a una sugerencia que hizo director de proyecto de usar una plataforma que proporciona Google llamada Google App Engine, que es gratuito y se pueden crear aplicaciones web programadas con el lenguaje de programación Java y así tener la posibilidad de aprender otras apis, no solo Java2EE, para crear aplicaciones web en Google App Engine también hay que conocer aunque sea de forma básica Java2EE.

1.2. Objetivos que queríamos conseguir.

El principal objetivo que queríamos conseguir era que la forma de firmar fuera muy fácil y que no fuera un mecanismo muy engorroso. Para ellos decidimos realizar una aplicación para smartphone android y una aplicación web para el almacenamiento y posterior comprobación de las firmas.

Por lo que empezamos a diseñar un sistema con el cual se puediera firmar digitalmente inicialmente solo un recibo y finalmente cualquier documento de la UMA agilizando dicho proceso.

En la parte de la aplicación de android se decidió hacer una aplicación clara y que fuese fácil de usar. Para eso usamos la API nivel 14 que equivale a la versión 4.0 de android, llamada Ice Cream Sandwich. Se eligió porque proporciona una nueva forma de diseño de las interfaces, un nuevo tema llamado Holo y proporciona muchas nuevas herramientas como por ejemplo son los ActionBar, que es una barra que permanece siempre en la parte superior de la pantalla en la que va acomodando a las necesidades en cada parte de la aplicación cambiando los botones según las necesidades, por ejemplo si estamos en la pantalla principal pues tendremos siempre visible el botón de añadir un nuevos recibo que abrirá el lector de códigos QR, se puede observar en la primera barra que se ve en la figura 1.1, sin embargo si estamos visualizando un recibo solo tendremos el botón de volver atrás, como se puede ver en la segunda barra de la figura 1.1, en la que se puede ver que al lado

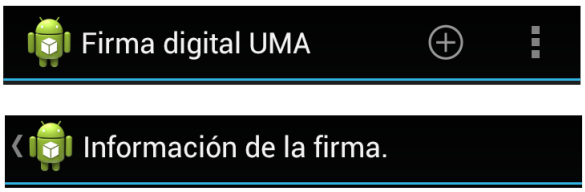


Figura 1.1: Action Bar.

del icono de la aplicación una flechita que indica que es el botón para volver atrás.

Al tomar la decición de programar para terminales con android 4.0 o mayor estuvimos sopesando los pros y los contras, y al final decidimos que la implantación de android 4.0 cada vez es mayor y que cada día hay más terminales con dicha versión, como se puede ver en este gráfico de la figura 1.2 y podemos observar que a finales de agosto de este año la cantidad de usuarios afectados sería de más del 15 % de terminales como podemos ver en la figura 1.3, aunque todavía sigue reinando la versión 2.3.3, aunque creemos que el cambio a la versión 4.0 o superior será rápida debido a todas las ventajas que aporta y mucho más ahora que hace unos meses Google sacó una nueva versíon, la 4.1, llamada Jelly Bean y casi todas las compañías querrán actualizar sus terminales a la última versión, por lo que a pesar de dejar a un gran número de usuarios sin poder usar la aplicación preferimos usabilidad y elegancia frente a gran cantidad de usuarios, ya que estos llegarán a medida que sus compañias actualicen sus terminales.

En la parte del servidor al elegir la plataforma de Google, hubo muchas cosas que resultaron más fáciles a costa de tener que aprender a usar el SDK que ellos proporcionan, que como era una de las cosas por la que elegimos dicha plataforma no nos importó. Una de las cosas que nos facilitaba es la gestión de usuarios, que los gestiona google directamente al tener que loguearte en la aplicación web con una cuenta de Google Account. Toda la seguridad, mantenimiento, copias de seguridad, balanceos de carga y un largo etcetera también lo hacen ellos por lo que no habría que preocuparse de ello.

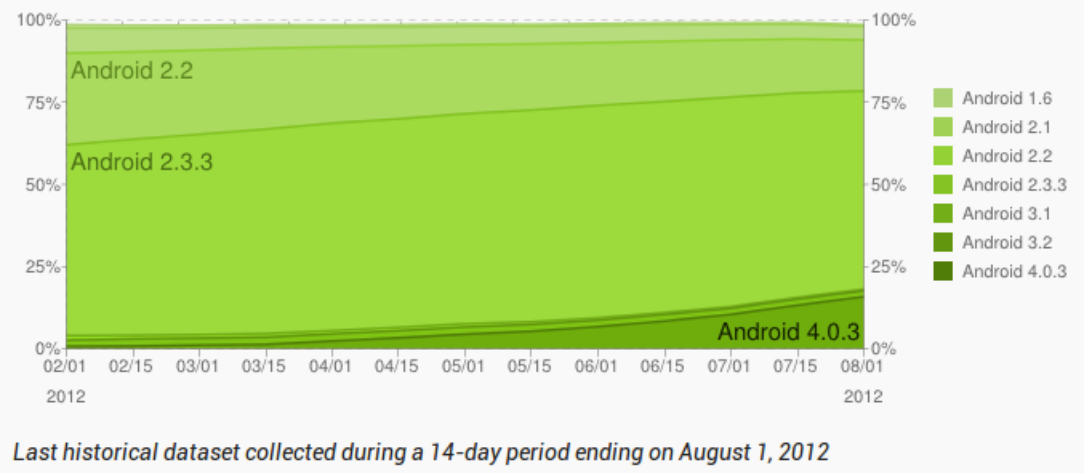
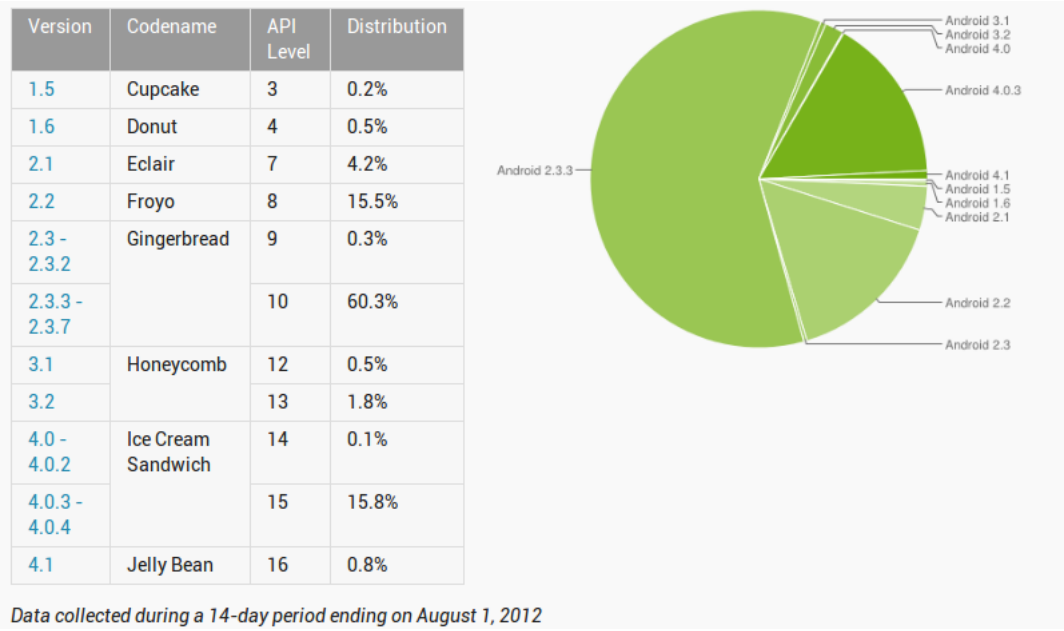


Figura 1.2: Gráfico de las versiones de android a finales de agosto del 2012.



Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.2%
2.2	Froyo	8	15.5%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	60.3%
3.1	Honeycomb	12	0.5%
3.2		13	1.8%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.1%
4.0.3 - 4.0.4		15	15.8%
4.1	Jelly Bean	16	0.8%

Figura 1.3: Gráfico del uso de las versiones de android a finales de agosto del 2012.

Figura 1.4: Samsung Galaxy Nexus.

1.3. Organización de la memoria.

1.4. Material usado.

Para la realización de este proyecto hemos usado un ordenador para todo lo que tiene que ver con la programación y un smartphone android para la depuración y prueba de la aplicación.

El ordenador es un ordenador portatil normal, con Ubuntu 12.04 como sistema operativo, un procesador Pentium Dual Core a 2.2Ghz, 4 Gb de memoria ram.

El móvil es un Samsung Galaxy Nexus que fue el primer terminal en tener android 4.0 y meses después el primero en recibir android 4.1. Sus característica son una pantalla de 4.65 pulgadas Super Amoled con resolución de 1280 x 720, procesador dual-core a 1.2Ghz, HSPA+, NFC, Wifi, GPS, etc.

Capítulo 2

Conocimientos previos sobre las tecnologías usadas.

En este segundo capítulo vamos a explicar cuales son y los conocimientos previos que teníamos sobre las tecnologías que hemos usado en el proyecto, así como una breve explicación de su funcionamiento, para que se usa y explicación breve sobre su historia y sus recientes versiones.

El elemento principal usado en el proyecto es el lenguaje de programación Java, en cual se programa tanto la aplicación web como la aplicación en el móvil, pero además de los diferentes SDK de Android y de Google App Engine, hace falta muchas otras tecnologías y lenguajes como pueden ser SQL, XML, UML, GIT. Además de los conocimientos básicos sobre criptografía de clave pública necesarios para realizar todo el proceso de firma digital.

2.1. El lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos que fue diseñado por Jame Gosling[1] para Sun Microsystems, que recientemente ha sido comprada por Oracle Corporation. Fue lanzado en 1995 y fue el centro de toda la plataforma Java de Sun Microsystems. Es un lenguaje con una sintaxis muy parecida a C o C++, pero con la gran ventaja de que el manejo de punteros

[1]Para más información sobre Jame Gosling: http://en.wikipedia.org/wiki/James_Gosling

Position Sep 2012	Position Sep 2011	Delta in Position	Programming Language	Ratings Sep 2012	Delta Sep 2011	Status
1	2	↑	C	19.295%	+1.29%	A
2	1	↓	Java	16.267%	-2.49%	A
3	6	↑↑↑	Objective-C	9.770%	+3.61%	A
4	3	↓	C++	9.147%	+0.30%	A
5	4	↓	C#	6.596%	-0.22%	A
6	5	↓	PHP	5.614%	-0.98%	A

Figura 2.1: Índice tiobe en septiembre del 2012. http://www.tiobe.com/

y objetos es automático, al igual que la recogida de basura.

Java es un lenguaje en el que hay que compilar los código fuentes para crear unos archivos intermedios llamados bytecodes, los archivos *.class, que luego serán interpretados por la máquina virtual de Java (JVM), que depende de la arquitectura en la que se quiera ejecutar la aplicación java. Gracias a esto se puede decir que java es un lenguaje multiplataforma, lo que significa que un mismo código java se puede ejecutar en un linux, en un windows, un mac o cualquier otro sistema para el cual exista una máquina virtual, lo que en inglés se llama "write once, run anywhere"(WORA). Además de esta gran ventaja Java es un lenguaje de propósito general, concurrente, basado en clases y orientado a objetos. Java es el segundo lenguaje de programación más popular de 2012, gracias a las aplicaciones web cliente-servidor que tienen tanto auge en estos momentos, como podemos ver en la figura 2.1.

La implementación original y las referencias del compilador de java, máquinas virtuales y las librerías de clases fue desarrollado por Sun en 1995, pero en el 2007 gracias a la lavor de la comunidad, Sun Microsystem cambio la licencia de todas las tecnologías Java a GNU General Public License, por lo que se abría la posibilidad a que se crearan versiones alternativas de compiladores bajo licencia GNU como GNU Compiler para Java o GNU Classpath.

En el proyecto la versión usada fue la versión Java SE 6.

2.1.1. Historia

Originalmente Java nació como un proyecto de James Gosling, Mike Sheridan, and Patrick Naughton en 1991, y estaba diseñado para una televisión interactiva pero era muy avanzado para lo que la industria de la televisión por clave de la época podía necesitar. En su origen fue llamado como Oak, por problemas con el nombre, ya que era una marca registrada cambiaron a llamarlo Green y posteriormente ya lo renombraron a Java como en la actualidad. Hay muchas teorías sobre el porque se llama Java, una de ellas es que había una cafetería llamada Java Coffe donde Jame, Mike and Patrick pasaron muchas horas consumiendo café.

La idea de James Gosling era crear una máquina virtual y un lenguaje de programación con la sintaxis y la estructura de C/C++ para que la curva de aprendizaje fuera muy suave para programadores que en la época sabían C/C++.

Sun Microsystem lanzó Java 1.0 en 1995, con la principal característica de que una vez escrito un código fuente no había que modificarlo para que funcionara en las diferentes máquinas, lo que anteriormente hemos llamado con el acrónimos en inglés WORA (Write Once, Run Anywhere). Rápidamente todos los navegadores de la época empezaron a soportar applet java en las páginas web, por lo que Java se volvió muy popular en la época. La nueva versión Java 2 fue lanzada en 1998-1999 y con ella llegarón las distinciones para las diferentes plataformas, como por ejemplo Java2EE para aplicaciones corporativas o una versión ligera llamada Java2ME que estaba diseñada para funcionar en los diferentes telefónos de la época, y todas las demás que se agrupan en la versión Java2SE, que es la versión estandar.

En 1997, Sun Microsystem intentó formalizar Java mediante una norma ISO/IEC pero se retiró del proceso y dio todo el control a la comunidad. Sun ofrecia implementaciones gratuitas y generaba dinero vendiendo algunas licencias de productos como Java Enterprise System. Una cosa importante es que Sun distingue entre el SDK (Kit de desarrollo) y el JRE (Entorno de ejecución) en el que van incluidos los compiladores, debugger, etc.

El 13 de noviembre del 2006, Sun lanzó Java gratis y software libre, bajo la licencia GNU General Public License (GPL). El proceso concluyó el 8 de mayo del 2007.

En 2009-2010 Oracle Corporation compró Sun Microsystem por lo que Java actualmente pertenece a Oracle.

2.1.2. Versiones

- **JDK 1.0** (23 de enero de 1996): Primer lanzamiento

- **JDK 1.1** (19 de febrero de 1997): Las primeras características añadidas fuero una reestructuración intensiva del modelo de eventos AWT (Abstract Windowing Toolkit), clases internas (inner classes), JavaBeans, JDBC (Java Database Connectivity), para la integración de bases de datos y RMI (Remote Method Invocation).

  (8 de diciembre de 1998): Recibió el nombre en clave Playground. Esta y las siguientes versiones fueron recogidas bajo la denominación Java 2 y el nombre "J2SE"(Java 2 Platform, Standard Edition), reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition). Se añadieron las siguientes mejoras, la palabra reservada strictfp, reflexión en la programación, la API gráfica (Swing) fue integrada en las clases básicas, la máquina virtual (JVM) de Sun fue equipada con un compilador JIT (Just in Time) por primera vez, Java Plug-in, Java IDL, una implementación de IDL (Lenguaje de Descripción de Interfaz) para la interoperabilidad con CORBA y Colecciones.

- **J2SE 1.3** (8 de mayo de 2000): Recibió el nombre en clave Kestrel. Los cambios más notables fueron: la inclusión de la máquina virtual de HotSpot JVM, RMI fue cambiado para que se basara en CORBA, JavaSound, se incluyó el Java Naming and Directory Interface (JNDI) en el paquete de bibliotecas principales (anteriormente disponible como una extensión), Java Platform Debugger Architecture (JPDA).

- **J2SE 1.4** (6 de febrero de 2002): Recibió el nombre en clave Merlin. Este fue el primer lanzamiento de la plataforma Java desarrollado bajo el Proceso de la Comunidad Java como JSR 59. Las principales características que se le añadieron fueron palabra reservada assert, expresiones regulares modeladas al estilo de las expresiones regulares Perl, encadenación de excepciones, non-blocking NIO (New Input/Output), logging API, API I/O para la lectura y escritura de imágenes en formatos como JPEG o PNG, parser XML integrado y procesador XSLT

```
void displayWidgets (Iterable<Widget> widgets) {
        for (Widget w : widgets) {
            w.display();
        }
}
```

Figura 2.2: Código ejemplo código for mejorado.

(JAXP), seguridad integrada y extensiones criptográficas (JCE, JSSE, JAAS), Java Web Start incluido.

- **J2SE 5.0** (30 de septiembre de 2004): Recibió el nombre en clave Tiger. Estos fueron los cambios mas importantes, plantillas (genéricos), metadatos, también llamados anotaciones, permite a estructuras del lenguaje como las clases o los métodos, ser etiquetados con datos adicionales, que puedan ser procesados posteriormente por utilidades de proceso de metadatos, autoboxing/unboxing, conversiones automáticas entre tipos primitivos (Como los int) y clases de envoltura primitivas (Como Integer), enumeraciones, varargs (número de argumentos variable), el último parámetro de un método puede ser declarado con el nombre del tipo seguido por tres puntos (por ejemplo *void drawtext(String... lines)*). En la llamada al método, puede usarse cualquier número de parámetros de ese tipo, que serán almacenados en un array para pasarlos al método, bucle for mejorado, La sintaxis para el bucle for se ha extendido con una sintaxis especial para iterar sobre cada miembro de un array o sobre cualquier clase que implemente Iterable, como la clase estándar Collection, de la siguiente forma:

- **Java SE 6** (11 de diciembre de 2006): Recibió el nombre en clave Mustang. En esta versión, Sun cambió el nombre "J2SE"por Java SE y eliminó el ".0"del número de versión. Los cambios más importantes introducidos en esta versión fueron un nuevo marco de trabajo y APIs que hacen posible la combinación de Java con lenguajes dinámicos como PHP, Python, Ruby y JavaScript, el motor Rhino, de Mozilla, una implementación de Javascript en Java, un cliente completo de Servicios Web y soporta las últimas especificaciones para Servicios Web, mejoras en la interfaz gráfica y en el rendimiento.

- **Java SE 7**: Su nombre en clave es Dolphin. Su lanzamiento fue en julio de 2011. Y las principales nuevas características fuero: soporte para XML dentro del propio lenguaje, un nuevo concepto de superpa-

quete, soporte para closures, introducción de anotaciones estándar para detectar fallos en el software.

2.2. El entorno de programación Eclipse.

Eclipse es un entorno integral de desarrollo que consta de un entorno de desarrollo integrado (IDE) y es extensible mediante plugins que está escrito en Java. Puede ser usado para una larga lista de lenguajes de programación como pueden ser, C, C++, Haskell, Perl, PHP, Python, Android y un largo etcetera. Fue originalmente desarrollado por IBM y fue lanzado con la licencia de software Eclipse Public License[2] la cual es una licencia de software libre. El SDK de Eclipse es libre y tiene licencia open source por lo que cualquier persona con los conocimientos puede programar el puglin que necesite para Eclipse. Fue el primer entorno de programación que funcionó bajo GNU Classpath y que funcionaba sin problemas con IcedTea. En la figura 2.3 se puede ver el aspecto que tiene.

2.2.1. Historia

Eclipse comenzó como un proyecto de IBM Canadá. En noviembre de 2001 se creó un grupo de empresas para promover el desarrollo de Eclipse como software libre, los miembros iniciales eran Borland, IBM, Merant, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and WebGain. Finalmente en enero de 2004 se creó la Eclipse Foundation.

Todas las versiones de eclipse empezaron llamandose como las lunas del planeta Júpiter.

[2]Para más información visite: http://en.wikipedia.org/wiki/Eclipse_Public_License

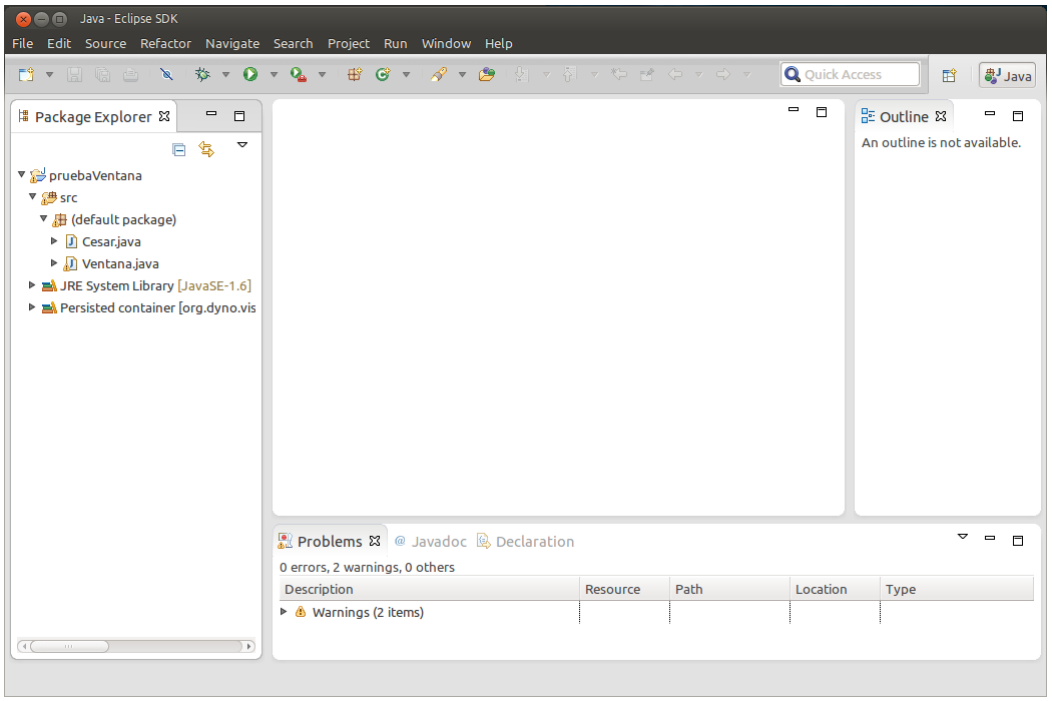


Figura 2.3: Eclipse 4.2 Juno.

Capítulo 3

Google App Engine.

En este apartado de la memoria voy a explicar lo que es, la configuración y el como usar la plataforma Google App Engine.

3.1. Introducción.

Google App Engine es una conjunto de apis que proporciona Google para construir tus propias aplicaciones web, que pueden ser alojadas y usadas en su servicio Google App y vendidas en Google Apps Marketplace. Además de alojamiento gratuito Google, ofrecen un dominio, que es: http://nombre_de_la_aplicacion.appspot.com y una base de datos propietaria de Google que se accede transparentemente a través de la api, gestión de usuarios mediante autentificación con cuentas Google del tipo: *usuario@gmail.com*, autentificación por federación o *openID*.

Además de todas esas características Google proporciona apis para Java, Python y Go, este último un lenguaje experimental del propio Google. Para usar dicha API, Google también da un plugin para Eclipse, en caso de que el lenguaje elegido sea Java, que ayuda al despliegue de la aplicación web, autocompletado y gestión de de las aplicaciones creadas.

En el proyecto solo he usado la API de Google App Engine de Java, por lo que todo lo que puedo comentar es de dicha API, la parte de Python y Go no se han estudiado.

En general el uso de Google App Engine para crear aplicaciones web es idéntico a crear una aplicación web con Java 2 Enterprise Edition (Java2EE), se pueden crear servlet que recogen valores **GET** o **POST** y además clases java para hacer operaciones con dichos valores. A su vez para mostrar la infomación se pueden generar archivos **.jsp*, que son archivos html con bloques o líneas de código java que se introducen con estas etiquetas: <%= línea de código Java %> o <% Bloque de código Java %>. A parte de archivos **.java* y **.jsp*, debemos tener una carpeta llamada *war* en la que tiene que ir toda la información de la aplicación web que queremos deplegar. En dicha carpeta hay varias subcarpetas como pueden ser *css* en la que tiene que ir el estilo de la web o *WEB-INF* en la que están todos los archivos de configuración, como pueden ser los permisos que tenemos que tener para poder acceder al uso de un servlet, si la web tiene conexión https, la configuración de la base de datos, etc.

Para este proyecto se han tenido que desarrollar dos aplicaciones web, una que es un servidor de timestamp y otra que es una aplicación para gestión de las firmas digitales que realice cada usuario. A continuación vamos a explicar en profundida la tecnología usada y ambas aplicaciones web.

3.2. Explicación de una aplicación web genérica en Google App Engine.

En esta parte voy a explicar en profundidad que es un servlet, los archivos de configuración, los archivos **.jsp* y el resto de archivos necesarios para poder desplegar una aplicación en Google Apps.

3.2.1. ¿Qué es un servlet?.

Un servlet es la evolución de los antiguos applet, su uso más comun es generar páginas web dinámente con los parámetros que recibe mediante una petición realizada por el navegador web y datos que están almacenados en el servidor web.

Un servlet es un objeto java que tiene que ser ejecutado en un servidor web o contenedor J2EE, que recibe unos parámetros, realiza una o varias

acciones y devuelve un resultado que puede ser desde un código html, un JSP que genera dinámicamente un código html, un JSON o una simple cadena de texto.

Los servlets, junto con JSP, son la solución de Oracle a la generación de contenido dinámico equivalente al lenguaje PHP, ASP de Microsoft, Ruby, etc.

Los servlet forman parte de Java Enterprise Edition (JEE) que a su vez es una amplicación de Java Standard Edition (JSE), para usarlos necesita un servidor web que pueda interpretar código java, el más famoso es Apache Tomcat que está desarrollado y mantenido por Apache Foundation, que son los encagardos también de mantener y desarrollar el famoso servidor web Apache, aunque existen otro como JBoss, Jetty o GlassFish, pero como veremos en este proyecto no son los únicos, ya que el propio Google Apps también funciona internamente a base de servlets y JSP.

Para crear un servlet hay que generar una clase java que implemente la interfaz *javax.servlet.Servlet* o que extienda cualquier clase que herede de una clase o que implemente la interfaz anterior, como puede ser *javax.servlet.http.HttpServlet* que es específico para conexiones HTTP. Una vez generada la clase hay que implementar el método **doGet** para peticiones tipo **GET** o el método **doPost** para peticiones de tipo **POST**. En el siguiente trozo se código se puede ver la implementación mas básica de los métodos **doGet** y **doPost**, con las llamadas sus respectivas llamadas a *super*.

```
@Override
protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException, IOException {
        // TODO Auto-generated method stub
        super.doGet(req, resp);
}

@Override
protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
        // TODO Auto-generated method stub
        super.doPost(req, resp);
}
```

Una vez implementados los métodos que se necesiten se pueden usar el parámetro *HttpServletRequest req* para recibir los valores que queramos enviar a la aplicación web y podemos usar *HttpServletResponse resp* para enviar lo que queramos desde una redirección a JSP o una página web a una JSON o cadena de texto. Un ejemplo de como se reciben los parámetros sería:

```
String num_sec = req.getParameter("sec");
```

Y si queremos enviar algo por *HttpServletResponse resp* podríamos usar:

```
PrintWriter out = resp.getWriter();
out.print(jsonArray);
out.flush();
```

Como podemos ver el objeto *resp* nos da la posibilidad de conseguir un objeto *java.io.PrintWriter* por el que podemos enviar lo que necesitemos.

La forma de acceder a un servlet mandandole peticiones **GET** sería la siquiente: https://servertimestamp.appspot.com/search?id=63texto=Prueba, como se puede ver la dirección base seria: https://servertimestamp.appspot.com/, el servlets estaría mapeado internamente en el servidor web, como ya veremos en próximas sección, en la dirección /search y el primer parámetro va precedido de ?id_parámetro y el resto de &id_parámetro. En nuestro ejemplo tendría dos parámetros que son *id* y *texto*, con sus respectivos valores después del =.

El método **POST** es el utilizado para pasar parámetros por medio de formularios.

3.2.2. ¿Qué es JSP?.

JSP es el acrónimo de JavaServer Pages y es una tecnología que ayuda a crear dinámicamente páginas web badas en HTML o XML y es la solución equivalente a PHP de Oracle. En la figura 3.1 se puede observar el proceso que sigue desde que se hace la petición en el navegador hasta que se muestra.

Un fichero **.jsp* es la unión de código HTML con código java, el cual es interpretado en el momento de visualización de la página web. Un ejemplo es el siguiente:

```
<!DOCTYPE html>
<html>
<body>
<table>
<tr>
```

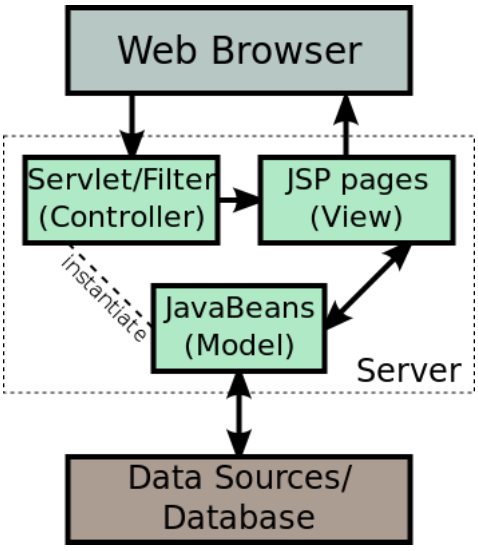


Figura 3.1: Modo de interpretación de un archivo JSP

```
            <th>ID</th>
            <th>Num sec</th>
            <th>Token de tiempo</th>
            <th>Mensaje</th>
            <th>URL para ver la firma</th>
            <th>Fecha</th>
            <th>Usuario</th>
            <th id="filadestino">Destino</th>
            <th>Verificado?</th>
</tr>

<% for (RowRepositorioGeneral row : rows) {%>
<tr>
            <td><%=row.getId()%></td>
            <td><%= row.getNum_sec()%></td>
            <td><%=row.getToken_tiempo()%></td>
            <td><%=row.getTexto_claro()%></td>
            <td><a href=<%=row.getUrl_firma()%>>URL para ver el token
                        de tiempo</a></td>
            <td><%=row.getFecha()%></td>
            <td><%=row.getUsuario()%></td>
            <td id="filadestino"><%=row.getDestino()%></td>
            <td>
```

```
                    <%
                    Boolean confirmado = row.getConfirmado();
                    if (!(confirmado == null) && confirmado) {
                    %>
                    <center>
                            <img src="ok.png" />
                    </center> <%} else {%>
                    <center>
                            <img src="cancel.png" />
                    </center> <%} %>
            </td>
</tr>
<%} %>
</table>
</body>
</html>
```

Como se puede ver en este trozo de código este jsp genera una tabla que se rellena dinámicamente con los valores que devuelve un objeto java, se puede observar que se entrelazan trozos de código Java con etiquetas HTML. Si mostramos esta web y acto seguido introducimos otro objeto RowRepositorioGeneral en la estructura, cuando recarguemos la tabla tendrá una fila nueva.

3.2.3. La carpeta WAR.

La carpeta WAR es la carpeta principal para el despliegue de una aplicación web, ya que en ella es donde tienen que ir todos los archivos que necesitemos, desde archivos HTML, CSS, JSP, imágenes, etc. En la figura 3.2 se puede ver un ejemplo de la carpeta WAR de mi aplicación web.

Se puede ver las diferentes carpetas y ficheros que la forman. Se ve la carpeta css que contiene los archivos de estilo que la página web usará, también se pueden ver los archivos *web.xml* y *app.yalm* que son archivos de configuración del servidor que se verán en el proximo apartado 3.2.4 y además los archivos *jsp* que se usan en la aplicación junto con los archivos *html* y *javascript* que se necesiten.

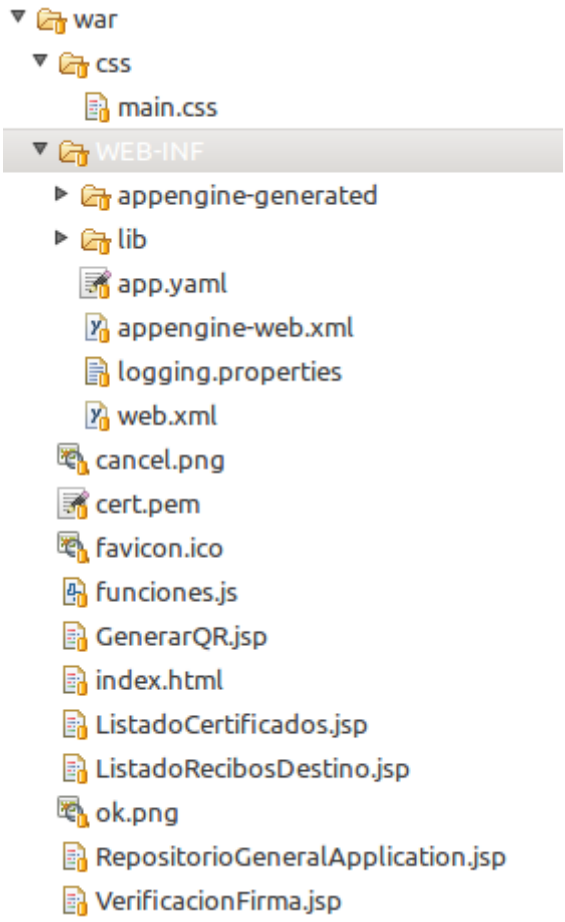


Figura 3.2: Carpeta WAR

3.2.4. Archivos de configuración.

Los principales archivos de configuración son *web.xml* y *app.yalm*, este segundo es solo una forma de escribir de forma más legible el xml, para que nos sea más sencillo escribirlo y leerlo a los humanos.

Un ejemplo de un archivo *web.xml* es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">

        <servlet>
                <servlet-name>AddRow</servlet-name>
                <servlet-class>pfc.ServletCreateRow</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>AddRow</servlet-name>
                <url-pattern>/add</url-pattern>
        </servlet-mapping>

        <welcome-file-list>
                <welcome-file>ServerTimestampApplication.jsp</welcon
        </welcome-file-list>
</web-app>
```

Como se puede observar en el código se ha definido un servlet que se llamará **AddRow** que usará la clase **ServletCreateRow** y que estará mapeado en la dirección web **/add**, también podemos observar que el fichero que nos mostrará el servidor será **ServerTimestampApplication.jsp** si entramos a la url principal.

A continuación veremos como es un archivo *app.yalm*:

```
application: repositoriorecibos
version: 1
runtime: java
```

```
handlers:
  - url: /add
    servlet: pfc.ServletCreateRow
    secure: always
welcome_files:
  - RepositorioGeneralApplication.jsp
```

Como podemos observar es mucho más fácil de entender y de escribir, el único problema que tienen los archivos YALM es que son sensibles a los espacios en blanco y tabuladores, por lo que hay que tener cuidado al redactarlos. En este archivo se crea un servlet en la ruta **/add**, que es la clase java **ServletCreateRow** del paquete **pfc** y que siempre hay que estar registrado en la aplicación para poder acceder a él. También podemos observar el fichero de bienvenida para cuando accedemos a la aplicación web. Al tener el archivo *app.yalm* en la carpeta WEB-INF el parseador de YALM interpreta dicho archivo y genera un archivo *web.xml* que es el que usará el servidor web para su configuración automáticamente.

Para ver todas las opciones de configuración que se pueden modificar en *app.yalm* o en *web.xml* se puede consultar estos enlaces https://developers.google.com/appengine/docs https://developers.google.com/appengine/docs/java/configyaml/. En el primero podemos ver todas las opciones configurables de *web.xml* y el la segunda las de *app.yalm*.

3.3. Servidor de timestamp.

En este apartado voy a explicar en profundidad todo lo relacionado con la aplicación de timestamp que he tenido que desarrollar, desde el diseño que se ha seguido hasta los problemas que me han surgido.

En principio me gustaría explicar para que se usa un servidor de timestamp en general. Un servidor de timestamp es un registro donde toda persona puede subir un documento y el servidor guarda ese documento añadiendole la fecha en la que se realizó la subida, dicha aplicación luego ofrece el servicio de consultar a que hora fue subido dicho documento. Un ejemplo podría ser https://seguro.ips.es/servidortimestamp/index.asp que se puede ver una captura de pantalla en la figura 3.3. En dicha captura podemos ver que tiene las opciones básicas de un servidor de timestamping como puede ser generar

un sello, consultar su validez, etc.

Figura 3.3: Servidor Timestamp https://seguro.ips.es

La veracidad de que el sellado de dicho documento fue en el instante que dice ser, depende de la confianza que se tenga en ese servicio. Es similar a cuando se necesita que te sellen un documento físico, que dependiendo de para quien lo necesite, necesitas que lo firme un notario o un empleado público si es para una entidad pública. Normalmente suelen existir servidores de timestamping en los que se tiene confianza y los documentos sellados se consideran verdaderos.

Existen tres modelos principales de servidor de timestamping que son los siguientes:

- **Solución Arbitrada básica:** En esta solución el usuario que quiere sellar algo mandaría una copia del documento que quiere sellar a la entidad de sellado, que a su vez pondría el sello de tiempo y a su vez guardaría una copia de dicho documento, este es el modelo mas parecido a la vida real. Esta solución tiene un par de problemas grandes como puede ser que la privacidad del documento se pierde, tenemos que tener en cuenta que el servidor de timestamping puede estar en España, EEUU o en cualquier otro país y a su vez la base de datos para almacenar todos los documentos tiene que ser enorme, por lo que almacenar todos los documentos nos puede acarrear muchos problemas.

- **Solución Arbitrada avanzada:** Esta solución es una evolución de la anterior, en ella el cambio que se hace es que el usuario que quiere que le sellen el documento manda el hash de dicho documento, un hash es

el resultado de una función unidireccional que recibe un documento y devuelve un valor único, teniendo dicho valor no se puede saber el documento original, pero dicho documento siempre creará ese valor único, y la entidad solo tendría que almacenar dicho hash junto con el sello de tiempo que se ha generado. Esta solución no tiene los inconvenientes de la anterior, ya que el tamaño de los documentos se reduciría a unos pocos bytes, y la privacidad del documento no se ve comprometida. El problema que si persiste es que el usuario conozca a la entidad de certificación y puedan generar timestamp falsos, pero este problema ya va dentro de la confianza que queramos darle a ese servicio. Suponemos que si es un servicio oficial y serio este problema no va a suceder, de todas formas existen otras soluciones que arreglan este problema.

- **Solución Arbitrada avanzada y distribuida:** Esta forma consigue arreglar el problema de la anterior que se produzca un uso fraudulento del timestamping es usando varias entidades de timestamping, por lo que el usuario mandaría el hash a varias entidades de sellado y el usuario guarde los reguardos que están firmados digitalmente de todas las entidades. Así si en una hay un problema tiene varias copias que certifican que se selló dicho instante.

- **Solución mediante enlaces:** Esta solución es la mas compleja y a su vez la que soluciona todos los problemas anteriores, además tiene la ventaja de que no tiene que usar multitud de entidades de certificación. Consiste en que cuando un usuario quiere sellar algo, manda el hash del documento, la entidad añade el número de serie del documento anterior, el timestamp y lo firma digitalmente, entonces el problema de que se introduzcan valores fraudulentos por mitad se anula, ya que cada recibo está enlazado con el anterior.

En nuestro caso hemos desarrollado un servidor de timestamping en su versión solución arbitrada avanzada.

A continuación voy a explicar la aplicación web, los servlets que la componen y sus funciones.

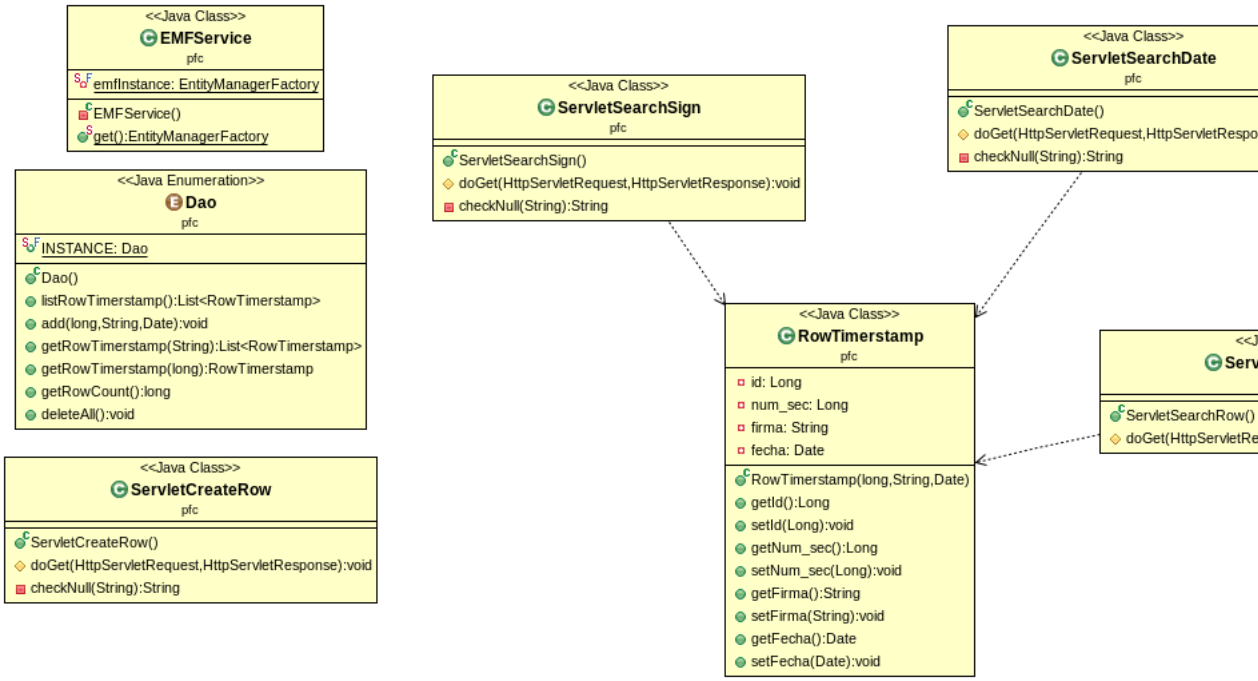


Figura 3.4: Detalles del paquete pfc

3.3.1. Explicación de la aplicación web.

En este capítulo voy a explicar todas las partes que componen la aplicación web que he desarrollado para la implementación del servidor timestamp.

En la figura 3.4 se puede ver las clases que forman el paquete *pfc*.

A continuación voy a explicar una a una las clases desarrolladas.

Dao.java: Esta clase es la encargada de todos los accesos a la base de datos, desde insercción, borrado y listado de las filas, hasta consultas que se necesiten hacer. Se puede observar que las consultas que son de listado de columnas se ejecutan con una sentencia SQL, un ejemplo es la siguiente:

```
EntityManager em = EMFService.get().createEntityManager();
Query q = em.createQuery("select t from RowTimerstamp t where t
q.setParameter("num_sec", id);
RowTimerstamp RowTimerstamps = (RowTimerstamp)q.getSingleResult
```

Pero las consultas que implican borrado o inclusión de filas no se realizan con sentencias SQL convencionales, se añaden con métodos que proporciona la API, un ejemplo es el siguiente:

```
EntityManager em = EMFService.get().createEntityManager();
RowTimerstamp RowTimerstamp = new RowTimerstamp(num_sec, firma, fecha
em.persist(RowTimerstamp);
em.close();
```

RowTimerstamp.java: En esta clase se diseña el formato de las filas de la base de datos, que como se ha explicado anteriormente no se crea con sentencias SQL, se usa un modelo de programación llamado JPA. Para dicho modelo hay que crear una clase que contenga como variables de clase las columnas de la tabla de la base de datos. Como podemos ver mediante anotaciones Java se le indica que campo es la clave primaría, también se puede indicar que ese campo es autoincrementado y otras opciones que habría que indicar en la creación de la tabla.

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
private Long id;
private Long num_sec;
private String firma;
private Date fecha;
```

Se puede ver que el campo id será la clave primaría que se indica con *@id* y que es autoincremental, a su vez también podemos ver el resto de datos que se van a guardar, el campo *num_sec* es el número de secuencia, ya que el campo *@id* lo usa la base de datos para organizarse ella, el campo *firma* es hash firmado por el usuario y que se quiere tener constancia de que se subío en la fecha que indica el campo *fecha*. El resto de métodos que tiene esta clase es un contructor, getter para consultar los campos y setter para insertar valores.

ServletCreateRow.java: Esta clase es un servet que se encagar de recibir todos los parámetros necesarios y añadirlos a la base de datos. Al recibir los parámetros mediante **GET** tiene que implementar el método *doGet*, casi todos los servlets implementados en el proyecto mandan los parámetros mediante **GET**. La forma de recibir parámetros es la siguiente:

```
String firma = req.getParameter("firma");
```

El resto de parámetros que se necesitan se generan en el servidor para que no puedan ser falseados, como es el número de secuencia y la fecha.

Si la inserción se produce correctamente se devuelve una cadena que tiene el siguiente formato: "ok;;num_sec;;fecha"que será interpretado en la aplicación android y que parseará dicha cadena para conseguir los valores que necesitemos.

ServletDeleteAll.java: Es un servlet "secreto"que se usa para borrar todas las filas del servidor, cosa que no se debería poder para no poder falsear los datos introducidos en el servidor de timestamp. Hay que llamarlo con un parámetro que es **borrar** con valor **5**.

ServletSearchDate.java: Es un servlet que devuelve una cadena con la fecha de una fila que tiene el número de secuencia que se le pasa en el parámetro **token**.

ServletSearchRow.java: Es un servlet que devuelve una página web donde se puede observar en una única columna toda la infomación almacenada que corresponde con el número de secuencia que se le pasa en el parámetro **id**. La forma de hacerlo es la siguiente:

```
PrintWriter pw = resp.getWriter();
pw.print("<!DOCTYPE html>");
pw.print("<html><head><title>Lista Time Stamp</title><link rel=
                    "href=\"css/main.css\"/> <meta charset=\"utf-8\
pw.print("<body><table><tr><th>ID</th><th>Num sec</th><th>Firma
                    "<td>"+ row.getId() +"</td><td>"+ row.getNum_se
                    row.getFecha() + "</td></tr> </table></body>");
pw.flush();
```

Como se puede ver se crea una tabla en una web que su fila se generan dinámicamente dependiendo del número de secuencia que se le pase.

ServletSearchSign.java: Es un servlet que devuelve una cadena con la firma que corresponde al número de secuencia que se pasa por el parámetro **token**.

3.4. Servidor de registro de firmas.

El servidor de registro de firmas que hemos desarrollado es una aplicación web en la que se pueden consultar las firmas que tu has realizado, las firmas

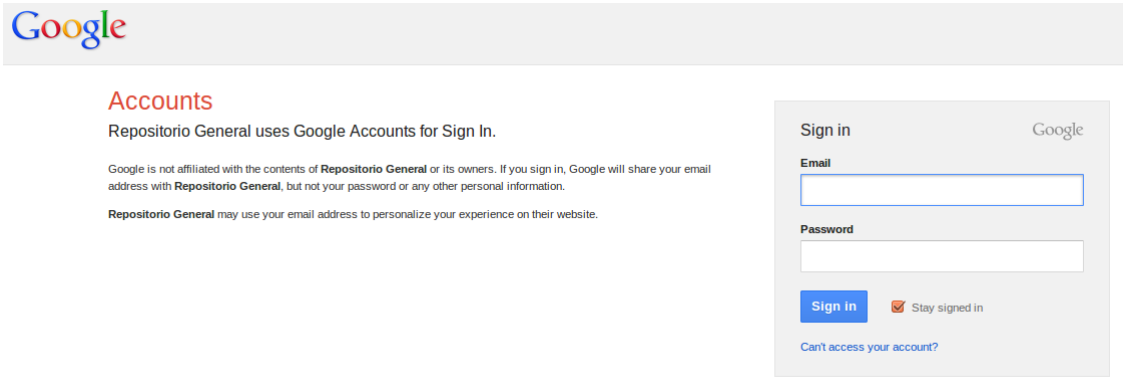


Figura 3.5: Login en Repositorio General

en las que el destinatario eres tú, gestión del certificado de clave pública, verificar una firma, exportar una cadena con la que cualquier persona puede mirar si la firma que has realizado es válida para comprobaciones en caso de algún problema, y generar códigos QR para que que alguien que lo necesite pueda firmarlo.

El sistema de gestión de usuarios la proporciona google, y para entrar en la aplicación web hay que tener una cuenta de google, si no se produce una redirección a la página de logueo que se puede ver en la figura 3.5. La parte de la seguridad de los usuarios, logueo y mantenimiento de las base de datos ya las proporciona el mismo Google.

La aplicación web se puede ver en la figura 3.6.

La aplicación tiene varias pestañas, que se explicarán posteriormente cuando expliquemos cada archivo **.jsp*, pero principalmente cada una de ellas se encarga de hacer una de las funciones que hemos comentado anteriormente.

3.4.1. Explicación de la aplicación web.

En la figura 3.7 se puede ver las clases que forman el paquete *pfc* de la aplicación web repositorio general.

A continuación vamos a explicar una a una las clases desarrolladas.

Dao.java: Al igual en el servidor de timestamp esta clase es la encargada de hacer todas operaciones contra la base de datos. Para mas información mirar el apartado 3.3.1.

DaoUserCert.java: En este aplicación hemos utilizado dos bases de datos, una para guardar las firmas y otra para guardar los certificados de clave pública que se necesitan para verificar si una firma es correcta o no. Esta clase es la encargada de todos los accesos, tanto inserciones como consultas, a dicha tabla.

RowRepositorioGeneral.java: Esta es la clase con la que se crea la tabla en la que se almacenan las firmas de los usuario, tiene los siguientes campos:

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE) //
GenerationType.IDENTITY
private Long id;
private Long num_sec;
private String url_firma;
private String texto_claro;
private Long token_tiempo;
private String usuario;
private Boolean confirmado;
private String destino;
private BlobKey blobKey;
private Date fecha;
```

Tiene una clave primaría que es *id* que es usada por Google internamente para el almacenado de la información, *num_sec* es el número de secuencia dentro la tabla que va incrementandose automáticamente, *url_firma* es la dirección en la cual se puede consultar la firma del texto en claro que está en el campo *texto_claro*. También se guarda el *token_tiempo* que es el *num_sec* de en la aplicación web del servidor de timestamp. La columna *usuario* almacena el usuario que ha subido la firma, y en La columna *destino* se guarda a quien va dirigido la firma, ya que todas firmas tienen un destinatario. En la columna *blobkey* se guarda la referencia al certificado de clave pública que estaba en activo cuando fue subido a la aplicación web, la columna *confirmado* puede valer true or false e indica si al subir la firma se pudo verificar y el texto en claro coincidía con el texto cifrado, en *fecha* está la fecha en la que se almacenó.

RowUserCert.java: Esta clase es la encagargada de crear la tabla que usamos para guardar los archivos con la clave pública. Los campos que usaremos para almacenarlos serán los que se pueden ver en

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
private Long id;
private String usuario;
private Date fecha;
private BlobKey certificado;
```

Como podemos observar el campo *id* será la clave pública y como hemos explicado será usado por la base de datos de Google para autogestión de las filas, el campo *usuario* guardará una cadena con el email de la persona que ha subido ese archivo, el campo *fecha* es la fecha en la que se subió el archivo, dicho campo se usará para comprobar que no se puedan falsear firmas, con varias comprobaciones y para anular certificados en caso de perdidas o que se necesite reemplazarlo, *certificado* es un campo del tipo **BlobKey** que es como la ruta al archivo de certificado.

A continuación vamos a explicar los diferentes servlets que hemos desarrollado para la aplicación web.

ServletCreateCertificate.java: Este servlet es el encargado de añadir a la base de datos el certificado de clave pública. Es usado en la pestaña de certificados de la aplicación web y es llamado cuando se pulsa subir certificado. Se puede observar el botón subir certificado en la figura 3.8.

ServletCreateRowRepositorio.java: Este servlet está mapeado en la dirección: https://repositoriorecibos.appspot.com/add y recibe los siguientes parámetros: *texto*, *url_firma*, *token*, *destino* y *fecha*. Es el encargado de añadir una fila por cada llamada a dicha dirección, a dicho dirección no hay forma de acceder desde la aplicación web, solo se pueden comprobar las firmas ya introducidas. A su vez antes de introducir la fila comprueba que la firma se puede validar y marca como verdadero o falso la columna verificado que posteriormente en el archivo *RepositorioGeneralApplication.jsp* se cambiará por una imagen para hacer la verificación más visual. Si se hemos podido insertar la fila, el servlet devuelve la cadena **OK**, si no se devuelven varias cadenas con los fallos que se han producido.

ServletDeleteAll.java: Servlet "secreto"que borra todas las filas de firmas almacenadas, hay que llamarlo con un parámetro que es *borrar* con valor *7*

ServletExport.java: Este servlet es el encargado de exportar una fila de nuestras filas para que otra persona pueda comprobar si es válida. Este servlet es llamado cuando se pulsa el botón exportar de la pestaña principal de la aplicación web. Se puede observar en la figura 3.9

El servlet recibe los siguiente parámetros:

```
String mensaje = checkNull(req.getParameter("mensaje"));
String url_firma = checkNull(req.getParameter("token"));
String id_blob = checkNull(req.getParameter("id_blob"));
String user = checkNull(req.getParameter("usuario"));
```

Una vez se tienen esos parámetros creamos una cadena de texto en la que unimos los siguiente campos y cada parámetro va separado por el separador: *;/:*.

```
String cadACodificar = mensaje + ";/:" + url_firma + ";/:" + id
```

Acto seguido codificamos la cadena con **Base64**, que es una forma simple de codificar los caracteres para que no viajen en texto claro.

```
String cadCodificada = Base64.encode(cadACodificar.getBytes("UT
```

También añadimos unos limitadores para que cuando tengamos que decodificar ese mensaje podamos saber donde empiezan y donde termina la exportación.

```
pw.println("BEGIN_EXPORT");
pw.println("--------------------------------");
pw.println(cadCodificada);
pw.println("--------------------------------");
pw.println("END_EXPORT");
```

ServletListRow.java: Este servlet es el encargado en devolver todas las filas de la tabla que pertenecen a un usuario. La forma de hacerlo es la siguiente, primero se identifica el usuario con el que se ha logueado de esta forma:

```
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
```

Una vez se consigue el usuario se llama a la función *public List¡RowRepositorioGeneral¿getRowRe userId, Long num_sec)* de la clase *Dao.java*, esta última función nos devuelve una lista con todas las filas. Al llamar al servlet le pasaremos el último número de secuencia que tenemos guardado en el telefono móvil, para así agilizar las transferencias de datos, de esta forma solo nos devolverá las filas nuevas. La forma de devolvernos las filas será en un JSONArray, que es un objeto que dentro contiene varios objetos JSON[1].

La creación de los objetos JSON la realizamos de la siguiente forma:

```
JSONObject jsonObject = new JSONObject();

jsonObject.put("num_sec", rowRepositorioGeneral.getNum_sec().toString
jsonObject.put("texto", rowRepositorioGeneral.getTexto_claro());
jsonObject.put("url_firma", rowRepositorioGeneral.getUrl_firma());
jsonObject.put("token_tiempo", rowRepositorioGeneral.getToken_tiempo(
jsonObject.put("usuario",rowRepositorioGeneral.getUsuario());
jsonObject.put("fecha", rowRepositorioGeneral.getFecha().toString());
jsonObject.put("verificado", rowRepositorioGeneral.getConfirmado().to
jsonObject.put("destino", rowRepositorioGeneral.getDestino());
```

Ese objeto JSON se añade un objeto JSONArray que a su vez es el que devolveremos como respuesta al final de la ejecución de nuestro servlet y que espera la aplicación android que lo ha pedido.

ServletVerify.java: Este servlet es el utilizado en la pestaña verificar de nuestra aplicación web, como se puede observar en la figura ñaVerificarfig:pestañaVerificar

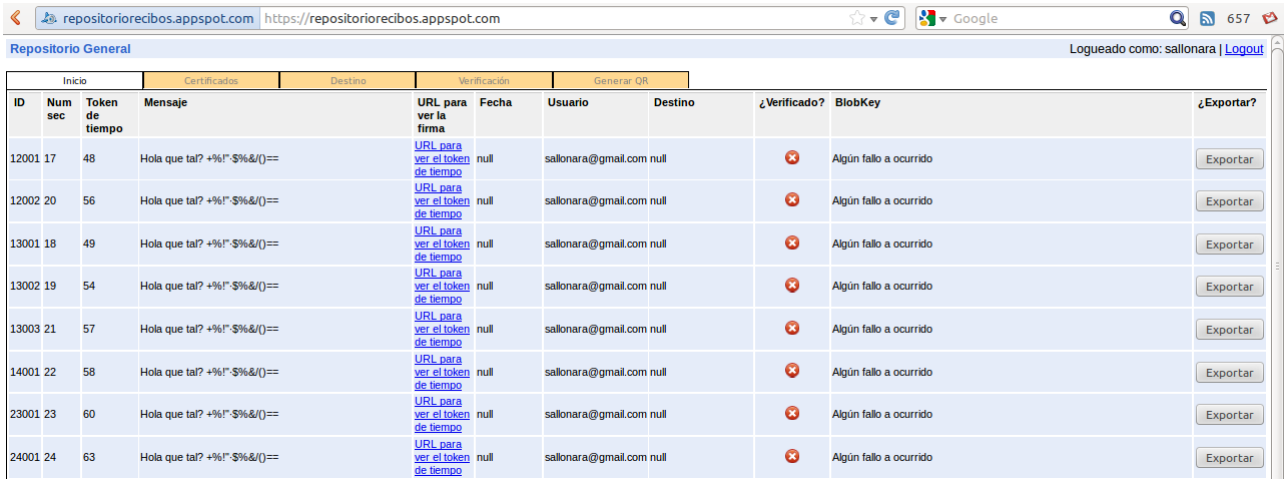
Como podemos observar hay un cuadro de texto para introducir la cadena que devolvería al pulsar el botón exportar. Cualquier usuario puede verificar si una firma es correcta o no. En este servlet se hace el proceso contrario que hicimos en exportar, quitamos los indicadores de inicio y final de exportación, desencriptamos la cadena en Base64 y hacemos varias comprobaciones. Comprobamos que en la fecha en la que se firmó el certificado era válido y que no habiamos revocado ese certificado, también se comprueba que no fuera reemplazado por otro certificado antes de su expiración, ya que entonces la firma no sería válida. También comprobamos la integridad del mensaje, que la

[1]Para saber que es un objeto JSON pueden consultar los siguientes enlaces: http://www.json.org/ o http://en.wikipedia.org/wiki/JSON

cadena no esté mal formada y que siga el formato que hemos obligado anteriormente.

Los archivos **.jsp* que hemos creado en su mayoría solo rellenan tablas dinámicamente haciendo llamadas a funciones de la clase *Dao.java*. Solo habría uno que no realiza esas funciones que es el siguiente:

GenerarQR.jsp: Este archivo JSP es el que se muestra en la pestaña *Generar QR*, se puede ver en la figura 3.11. Su función es generar un código QR para que pueda ser leido por la aplicación del móvil. Hay que rellenar los campos de destino y el texto que queremos que firme dicha persona. Al darle a *Generar código QR* se hace una llamada a la API Google Chart y se genera un código QR que contiene dichas cadenas y se muestra en la parte de la derecha, como se puede ver en la figura 3.12.



ID	Num sec	Token de tiempo	Mensaje	URL para ver la firma	Fecha	Usuario	Destino	¿Verificado?	BlobKey	¿Exportar?
12001	17	48	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar
12002	20	56	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar
13001	18	49	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar
13002	19	54	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar
13003	21	57	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar
14001	22	58	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar
23001	23	60	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar
24001	24	63	Hola que tal? +%!":$%&/()==	URL para ver el token de tiempo	null	sallonara@gmail.com	null		Algún fallo a ocurrido	Exportar

Figura 3.6: Repositorio General

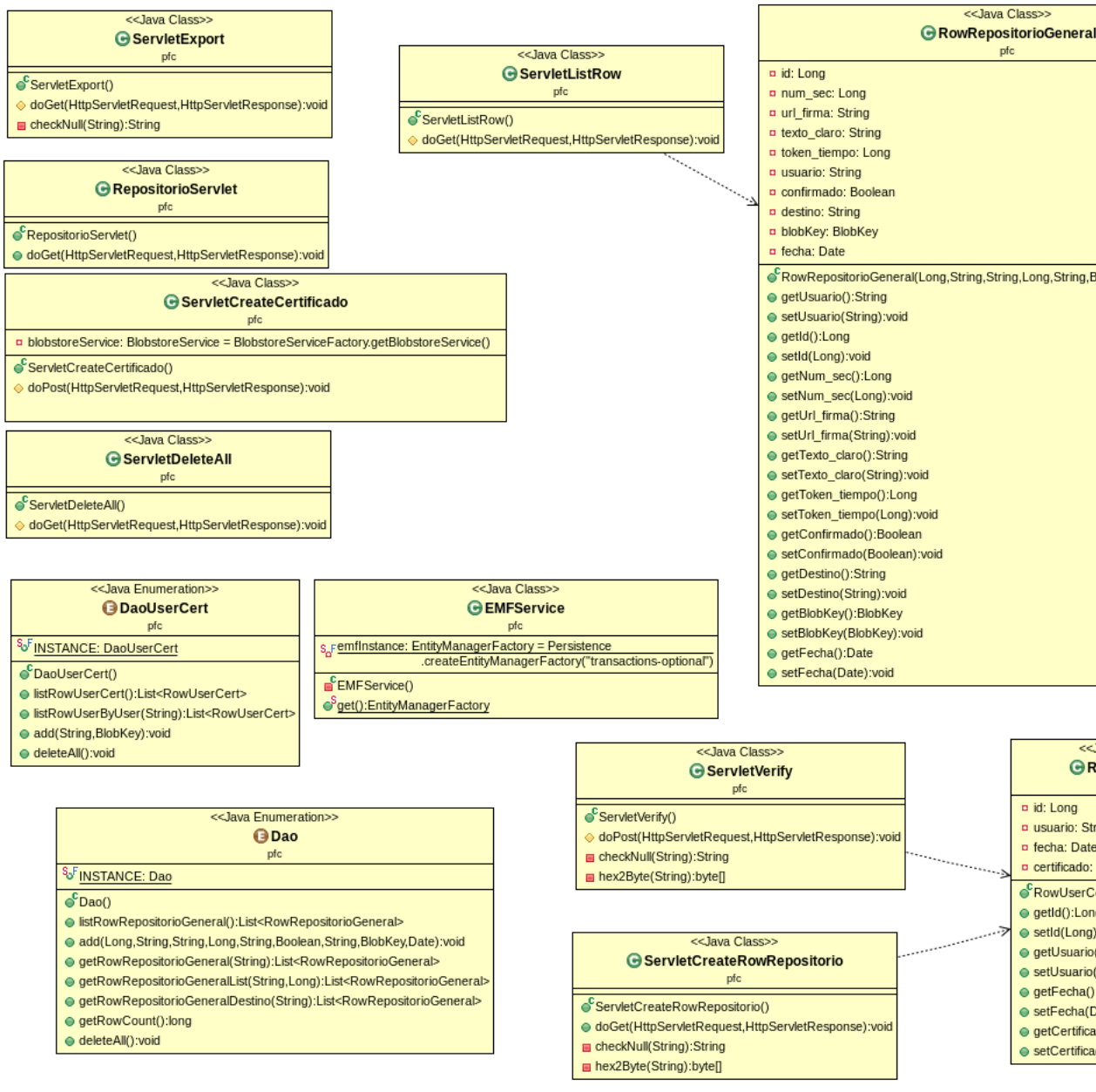


Figura 3.7: Detalles de las clases Repositorio General.

Figura 3.8: Pantallazo de la pestaña certificados.

Figura 3.9: Detalle del botón exportar.

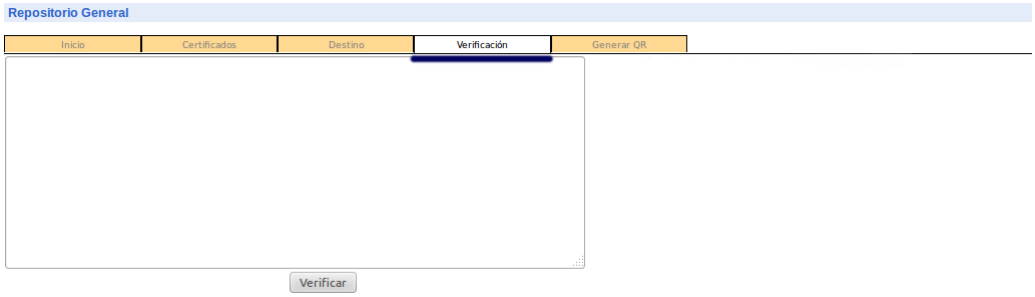


Figura 3.10: Detalle de la pestaña verificación.

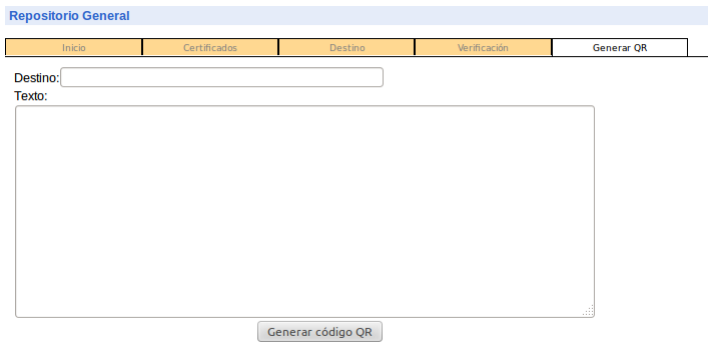


Figura 3.11: Pestaña para generar el código QR.

Figura 3.12: Pestaña con el código QR generado.

Índice de figuras