

naVerificar5.10106Detalle de la pestaña verificaciónfigure.5.10

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

2 de octubre de 2012

Índice general

1. Introducción.	1
1.1. ¿Por qué decidimos hacer el proyecto?	1
1.2. Objetivos que queríamos conseguir.	2
1.3. Organización de la memoria.	5
1.4. Material usado.	5
2. Conocimientos previos	7
2.1. El lenguaje de programación Java	7
2.1.1. Historia	9
2.2. El entorno de programación Eclipse.	12
2.2.1. Historia	13
2.2.2. Versiones	13
2.3. Criptografía.	13
2.3.1. Historia	14
2.3.2. Criptografía clásica.	14
2.3.3. Criptografía durante la época de los ordenadores.	16
2.4. Android.	20
2.4.1. Historia.	21
2.4.2. Versiones.	22
2.5. Google App Engine.	24
2.6. SQLite.	26
2.7. XML.	27
2.8. UML.	28
2.9. GIT.	29
3. Criptografía usada en el proyecto.	31
3.1. RSA.	31
3.2. Public-Key Infrastructure.	34
3.3. Criptografía en Java.	37

4. Android	43
4.1. Introducción.	43
4.2. Arquitectura de la plataforma Android.	44
4.2.1. Desarrollando en Android.	53
4.2.2. Proyecto básico de Android en Eclipse.	58
4.3. Proyecto firma digital UMA.	62
4.3.1. Uso de la aplicación.	62
4.3.2. Explicación del proyecto de Android.	64
5. Google App Engine.	83
5.1. Introducción.	83
5.2. Explicación de una aplicación web genérica en Google App Engine.	84
5.2.1. ¿Qué es un servlet?.	84
5.2.2. ¿Qué es JSP?.	86
5.2.3. La carpeta WAR.	88
5.2.4. Archivos de configuración.	90
5.3. Servidor de timestamp.	91
5.3.1. Explicación de la aplicación web.	94
5.4. Servidor de registro de firmas.	97
5.4.1. Explicación de la aplicación web.	98

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 inconveniente 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 pudiera 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

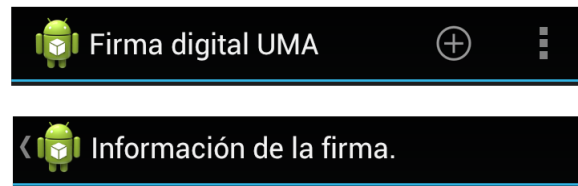


Figura 1.1: Action Bar.

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

Al tomar la decisión 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 versión, 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 compañías 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.

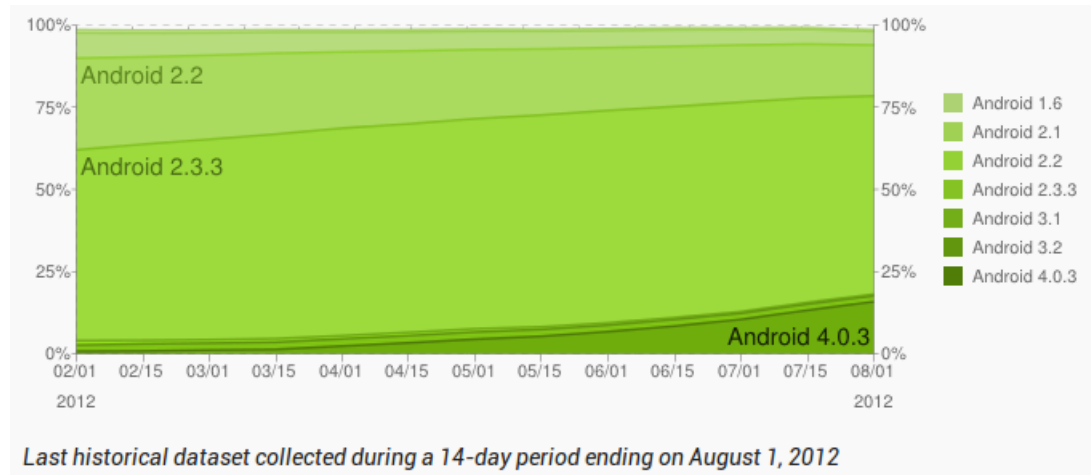


Figura 1.2: Gráfico de las versiones de android a finales de agosto del 2012.

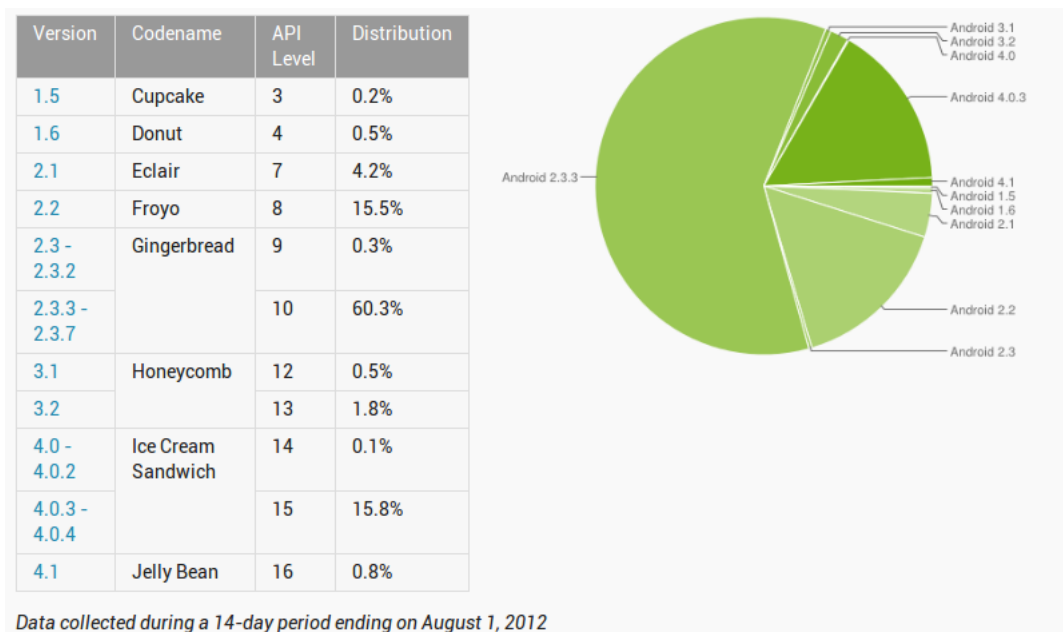


Figura 1.3: Gráfico del uso de las versiones de android a finales de agosto del 2012.



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 Gosling¹ 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

¹Para 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/>

parecida a C o C++, pero con la gran ventaja de que el manejo de punteros y objetos es automático, al igual que la recogida de basura.

Java es un lenguaje en el que hay que compilar los códigos 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 labor 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 llegaron 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 teléfonos 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 ofrecía 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.

Versiones

- **JDK 1.0** (23 de enero de 1996): Primer lanzamiento
- **JDK 1.1** (19 de febrero de 1997): Las primeras características añadidas fueron 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 (JAXP), seguridad integrada y extensiones criptográficas (JCE, JSSE, JAAS), Java Web Start incluido.


```
1 void displayWidgets (Iterable<Widget> widgets) {  
2     for (Widget w : widgets) {  
3         w.display();  
4     }  
5 }
```

Figura 2.2: Código ejemplo código for mejorado.

- **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 draw-text(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 fueron: soporte para XML dentro del propio lenguaje, un nuevo concepto de superpaquete, soporte para closures, introducción de anotaciones estándar para detectar fallos en el software.

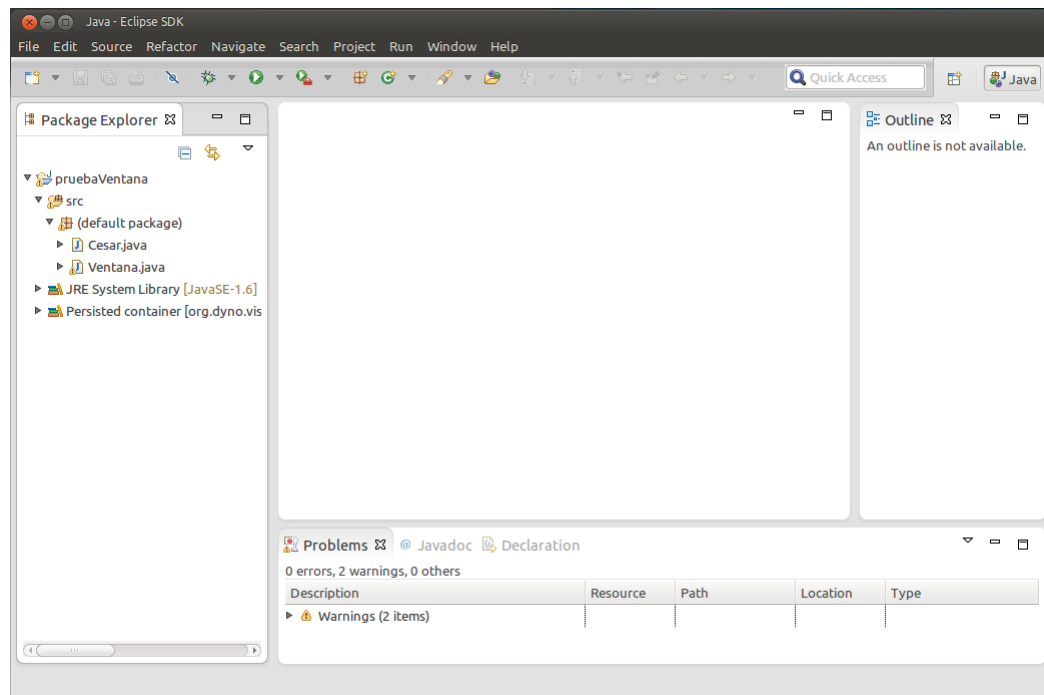


Figura 2.3: Eclipse 4.2 Juno.

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 License² 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 plugin 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.

En el proyecto hemos usado la versión Indigo, que equivale a la versión 3.7 de eclipse.

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

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.

2.2.2. Versiones

- **Versión 3.0** (21 de junio de 2004)
- **Versión 3.1** (28 de junio de 2005)
- **Versión 3.2** (30 de junio de 2006): recibió el nombre de Callisto.
- **Versión 3.3** (29 de junio de 2007): recibió el nombre de Europa.
- **Versión 3.4** (25 de junio de 2008): recibió el nombre de Ganymede.
- **Versión 3.5** (24 de junio de 2009): recibió el nombre de Galileo.
- **Versión 3.6** (23 de junio de 2010): recibió el nombre de Helios.
- **Versión 3.7** (22 de junio de 2011): recibió el nombre de Indigo.
- **Versión 4.2** (27 de junio de 2012): recibió el nombre de Juno.
- **Versión 4.3** (26 de junio de 2014): esta será la próxima versión, que saldrá el próximo año y recibirá el nombre de Kepler.

2.3. Criptografía.

La criptografía es la ciencia que se encarga del estudio de técnicas para proteger una comunicación, para que solamente la gente autorizada pueda verla, leerla y entenderla. En la actualidad la criptografía es un término que se usa de forma similar a encriptación, que es el proceso para transformar información mediante diferentes algoritmos en un mensaje que no pueda entender un atacante que intercepte la comunicación.

En el proyecto hemos usado una criptografía llamada Criptografía de Clave Pública, que como veremos a continuación en la historia y en el capítulo específico en el que se explica la criptografía usada en profundidad, consta de dos claves que están enlazadas matemáticamente y que si sabemos una no podemos averiguar la otra, una es pública que se le pasaría a la persona que quiera descryptar el mensaje, que es la que da nombre a este algoritmo y otra privada que solo conoce la persona que quiere encriptar el mensaje.

La criptografía ha evolucionado y ya no solo se usan para proteger mensajes, si no que también se usa para proteger la integridad de dichos, que es uno de los usos más común de este tipo de criptografía.

2.3.1. Historia

Podemos hacer dos grandes grupos dentro de la historia de la criptografía, la criptografía clásica y la criptografía durante la época de los ordenadores.

2.3.2. Criptografía clásica.

Durante la época de la criptografía clásica solo se quería proteger el mensaje que se enviaba de la mirada de curiosos y enemigos por lo que solo existían algoritmos de encriptación, la integridad del mensaje no importaba en esa época.

En dicha época todos los algoritmos de cifrados que existían eran por transposición o sustitución de caracteres. A continuación vamos a poner unos ejemplos de los algoritmos mas famosos.

- **Cifrado Cesar:** dicho cifrado es famoso porque los usaban las centurias romanas para comunicarse entre ellas y que si un mensaje era interceptado no pudiera ser leído. Consiste en sustituir cada caracter del mensaje por el que hay tres lugares a la derecha. Por ejemplo si tenemos el mensaje “Hola” si lo ciframos con este cifrado conseguimos “Krod”, en la figura 2.4 podemos ver como es el cifrado. Para descryptar solo habría que intercambiar por la tercera letra anterior.
- **Cifrado Homofónico:** Es una evolución del siguiente, pero en vez de

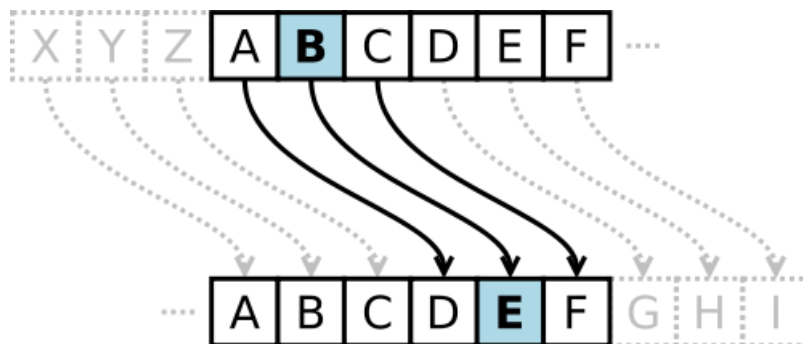


Figura 2.4: Ejemplo Cifrado Cesar

Letra	Símbolos asignados
A	10, 11, 23, 45, 76, 79, 87, 98
L	02, 15, 21, 25, 56, 60
N	44, 63, 71
O	04, 16, 28, 29, 37, 52, 69, 90
P	30, 88
T	24, 77

Figura 2.5: Tabla para cifrado homofónico

sustituir siempre por el mismo carácter lo que se hace es tener la posibilidad de poder realizar varios cambios posibles, por lo que un mismo mensaje podría generar varios textos cifrados, complicando así su descryptación. En la figura 2.5 podemos ver una tabla sencilla de sustitución para realizar el cifrado. Por ejemplo si ciframos la palabra “PLATON” nos daría de resultado “882110772963”, pero podríamos sustituir la P no solo por 88 si no por cualquier valor de la tabla dando lugar a que pudieramos crear varios mensajes cifrados.

- **Cifrado por Transposición:** consiste en realizar una permutación de las posiciones que ocupan las letras escritas, un ejemplo podría ser escribir todo el texto con una cierta longitud preestablecida y luego leerlo por columnas en vez de por filas. En la figura 2.6 podemos ver un ejemplo del mecanismo de cifrado.
- **Cifrado Producto:** Es un cifrado que combina sustitución y transposición y se puede considerar como un encadenamiento de varios cifrados. Esto da lugar a cifrados complejos, seguros y difíciles de atacar, ya que

“ANDALUCIA, CON EL MULHACEN Y EL VELETA,
VAYA PAR DE MONTAÑAS”



ANDALUCIA CON EL MULHACEN Y EL
LVELETA VAYA PAR DE MONTAÑAS



Mensaje cifrado:

ALNVDEALLEUTCAIVAACYOANPEALRMDUELMHOANCTEANNAS

Figura 2.6: Ejemplo de cifrado por sustitución

tendríamos que averiguar no solo el método utilizado igual que la clave también tendríamos que saber el orden en el que se ha ejecutado.

- **Cifrado Vernam:** es un tipo de cifrado que se denomina cifrado de flujo. El texto en claro se combina con una cadena del mismo tamaño que el texto en claro de número aleatorios o pseudoaleatorio por medio de la función XOR. Lo inventó Gilbert Vernam que era un ingeniero de AT&T en 1917. Es también conocido como RC4 en internet.

2.3.3. Criptografía durante la época de los ordenadores.

La criptografía dio un gran salto en cuanto a calidad en el momento en el que se pudieron empezar a usar ordenadores para encriptar y desencriptar textos, debido a que los ordenadores son máquinas que las tareas repetitivas las hacen muy bien y muy rápidos.

Se empezaron a idear nuevos algoritmos de cifrado mucho más complejos, los cuales se pueden dividir en dos grandes grupos, la criptografía de clave simétrica y la criptografía de clave pública.

A continuación vamos a explicar brevemente los algoritmos más famosos de estos dos grupos.

Critografía de Clave simétrica

Esta técnicas de criptografía se principal característica es que usan la misma clave para encriptar y desencriptar.

- **Data Encryption Standard (DES):** Fue presentado por IBM en 1974, para generar un estandar para cifrado de transmisión de datos y para cifrado de almacenamiento de datos, para el gobierno, empresas privadas o cualquier tipo de usuario. IBM comenzó el desarrollo basandose en un dispositivo de cifrado llamado Lucifer el cual tenia una clave de 128 bits. Es un criptosistema de clave secreta que cifra en bloques de 64 bits del texto en claro y genera otros bloques de 64 bits del texto cifrado. La clave utilizada también es de 64 bits, pero el bit final de cada octeto de los 64 bits de la clave se usa como bit de paridad para control de errores. El cifrado se realiza en 16 iteraciones en las que se usan varias operaciones como son operaciones XOR, permutaciones y sustituciones. El esquema para cifrar se puede ver en la figura 2.7.
- **AES (Rijndael):** Fue presentado al concurso AES el 2 de enero de 1997 y anunciado ganador en 2001. Fue diseñado por dos criptólogos llamados Joan Daemen y Vincent Rijmen, ambos estudiantes de la Katholieke Universiteit Leuven de Bélgica. Al contrario que DES, AES es una red de sustituciones y permutaciones no una red de Feistel, se transformó en estándar efectivo el 26 de mayo de 2002 y en la actualidad es uno de los algoritmos de encriptación más famosos. Opera con bloques de 128 bits y tiene claves de 128, 192 y 256 bits.

El mayor problema que tiene este tipo de criptografía es que para que el destinatario pueda leer el mensaje necesita saber la clave y el intercambio de clave puede ser un problema muy grande, si los dos usuarios no se pueden comunicar directamente, ya que usando cualquier otro método podría ser interceptado y todo el proceso de encriptación no serviría de nada.

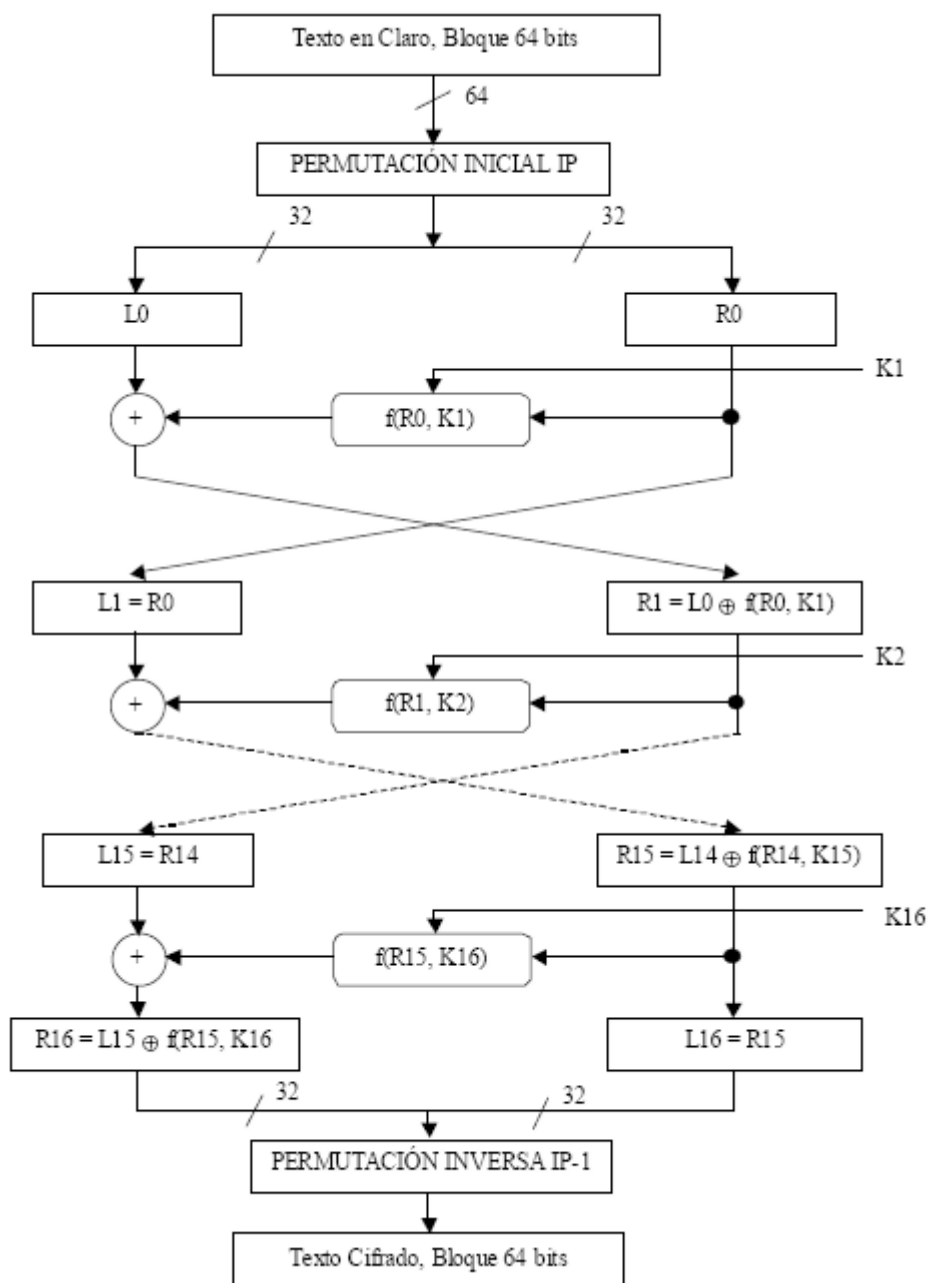


Figura 2.7: Iteraciones en el cifrado DES

Criptografía de Clave Pública

La criptografía de clave pública fue inventada por Diffie y Hellman y paralelamente por Merkle y ambos grupos aportaron a la criptografía el concepto de la utilización de pares de claves.

La característica principal es que cada usuario posee dos claves una privada que solo conoce el dueño de la clave y será usada para descifrar todo lo que otros usuarios cifren con su clave pública, así de esta forma si una persona quiere comunicarse con otra de forma secreta solo tiene que conocer su clave pública, cifrar con ella y el destinatario podrá descifrar el mensaje con su clave privada. También posee una clave pública que será conocida por el resto de usuarios y que estos usarán para encriptar el mensaje que queremos que sea secreto.

Otras características son que las claves son imposibles de deducir una a partir de la otra, cualquier usuario puede encriptar un mensaje con la clave pública pero no puede descifrarla, ya que solo se puede descifrar con la clave privada. Ambas claves son de una gran longitud y son generadas por exponenciación y/o productos de números primos grandes.

En los primeros años de existencia de la criptografía de clave pública se inventaron tres sistemas, Algoritmo de la mochila de Merkle-Hellman que fue roto, el esquema de McEliece que está considerado imposible de llevar a la práctica y un tercero que es el que explicaremos a continuación llamado RSA que es el más usado actualmente.

- **RSA:** Su nombre proviene de sus creadores que son Rivest, Shamir y Adleman y se basaron en la idea de “es muy fácil multiplicar dos números enteros primos grandes, pero extremadamente difícil hallar la factorización del producto cuando inventaron el RSA en 1977. Se trata de un algoritmo exponencial. Tanto el mensaje como el texto cifrado tienen que ser un código decimal, por lo que se usaría por ejemplo el valor ASCII de la letra. Un ejemplo de uso sería el siguiente, lo primero que se debe de hacer antes de enviar el mensaje es acordar el algoritmo que se va a usar, lo siguiente el emisor cifra el mensaje usando la clave pública del receptor y lo envía por el método que el prefiera, acto seguido el receptor descifra el mensaje que ha enviado el receptor usando su propia clave privada, la gran ventaja de todo este método que soluciona el gran problema que dijimos que tenían los algoritmos

de cifrado simétrico, que antes de nada había que intercambiar la clave con el problema que eso implicaba.

El algoritmo RSA será explicado con más profundidad en el capítulo de criptografía correspondiente.

Los algoritmos de clave pública tienen un gran problema y es que son muy lentos cifrando y descifrando por lo que en la vida real se suelen usar para intercambio de claves de algoritmos de clave simétrica que son mucho más rápido y también seguros.

2.4. Android.

Android es un sistema operativo basado en Linux especialmente diseñado para smartphone, tablet, smart TV y una infinidad de dispositivos, desarrollado por Google con Open Handset Alliance. Android empezó siendo desarrollado por la compañía llamada Android que inicialmente fue financiada y después comprada por Google en 2005. En 2007 cuando se presentó por primera vez Android también se anunció la fundación de Open Handset Alliance que es un conjunto de 86 empresas entre las que hay compañías de hardware, software y de telecomunicaciones, interesadas en el mundo de los dispositivos móviles. Android es código abierto y está distribuido bajo licencia Apache³. La tarea del mantenimiento y desarrollo de Android es de Android Open Source Project (AOSP).

Android tiene una gran comunidad de desarrolladores que pueden extender las funcionalidades de los teléfonos o dispositivos donde se pueda ejecutar android, android se puede desarrollar en Java usando el SDK de Android o en C++ usando el NDK de Android. Android posee una tienda online llamada Google Play (anteriormente Android Market) desde la que cualquier desarrollador por una pequeña cantidad de dinero (Alrededor de 25 por una cuenta vitalicia de desarrollador) puede subir todas las aplicaciones que desee, de forma gratuita o pagada. En Junio de 2012 había alrededor de 600.000 aplicaciones en Google Play.

³Para saber más sobre la licencia visite: http://en.wikipedia.org/wiki/Apache_License

En el primer cuarto de 2012, Android tenía el 59 % del mercado de smartphones en el mundo, de ahí la importancia de esta plataforma para los desarrolladores, ya que proporciona un mercado muy amplio y una forma muy fácil y barata de conseguir un gran número de usuarios.

Los detalles técnicos de android se explicarán con más profundidad en el tema de android

2.4.1. Historia.

Como hemos dicho anteriormente Android fue diseñado y creado originalmente por una compañía llamada Android que fue fundada en Palo Alto, California en 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White. Originalmente solo estaba diseñado para funcionar con smartphones ya que ellos pensaban que un smartphone era algo más que un dispositivo que sirviera para usar el GPS y tener preferencias.

Google compró Android el 17 de agosto de 2005 con la intención de entrar en el mercado de los teléfonos móviles. Después de varios años de rumores el 5 de noviembre de 2007, Google presentó la Open Handset Alliance un grupo de empresas que incluían a Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile and Texas Instruments entre otras muchas empresas que estaban interesadas en generar estándares para dispositivos móviles. Ese mismo día también se lanzó el primer producto android basado en el kernell de Linux 2.6.

Android ha sido muy criticado por la gran fragmentación que tiene debido al gran número de versiones que tiene, que son compatibles hacia abajo pero no hacia arriba, por lo que necesita que los fabricantes actualicen el software de sus teléfono lo que es un gran problema debido a que muchos no lo hacen, hubo un pequeño intento de solucionar este problema haciendo que los fabricantes sacaran actualizaciones en al menos en los 18 meses posteriores a la salida al mercado del terminal, pero al final no hubo ningún acuerdo.

2.4.2. Versiones.

Como curiosidad todas las versiones de Android se denominan con un nombre en clave que es un postre.

- **1.0 (Apple Pie):** primera versión lanzada el 23 de septimbme del 2008.
- **1.1 (Banana Bread):** lanzada el 9 de febrero del 2009.
- **1.5 (Cupcake):** fue presentada el 30 de abril del 2009, esta fue la primera versión con la que android empezó a despuntar y entrar en el mundo de los teléfonos móviles, anteriormente apenas si era conocido. Tenía características nuevas muy interesantes como poder grabar y reproducir video, podía subir videos a Youtube e imágenes a Picasa directamente desde el teléfono, un nuevo teclado predictivo, nuevos widget y carpetas para colocar en la pantalla de inicio y transiciones animadas.
- **1.6 (Donut):** fue presentada el 15 de septiembre de 2009. Se le añadieron las siguientes características nuevas como una interfaz ingrada para la cámara, la grabadora de video y la galería, se actualizó la búsqueda por voz añadiendo soporte a más aplicaciones nativas y la posibilidad de llamar a contactos, se añadió un buscador general en la pantalla de inicio donde se podía buscar contactos, historiales y páginas web, se añadió un nuevo framework de gestos y las herramientas de desarrollo llamado GestureBuilder.
- **2.0 / 2.1 (Eclair):** la versión 2.0 fue presentada el 26 de octubre de 2009 y la 2.1 fue liberada el 3 de diciembre del 2009. Se añadieron un gran número de mejoras, se optimizó la velocidad de hardware, se soportaron más tamaño de pantallas y resoluciones, se rediseñó la interfaz de usuario, el navegador también fue renovado y se le añadió soporte para HTML5, nueva lista de contactos, se añadió soporte para el flash de la cámara, zoom digital, soporte para bluetooth 2.1, se mejoraron la captura de eventos multi-touch con MotionEvent y fondos de pantalla animados.
- **2.2 (Froyo):** fue lanzada el 20 de mayo de 2010. Se optimizó el sistema Android, la memoria y el rendimiento, se mejoró la velocidad de las aplicaciones gracias a la implementación de JIT, se implementó el motor JavaScript V8 de Google Chrome en el navegador del móvil,

nueva funcionalidad de WiFi hotspot y tethering por USB, se actualizó el android market para que tuviera actualizaciones automáticas, marcación por voz y compartir contactos por Bluetooth, soporte para contraseñas numéricas y alfanuméricas, soporte para Adobe Flash 10 y soporte para pantallas de Hdpi, como pueden ser pantallas de 4z resolución de 720p.

- **2.3 (Gingerbread):** fue presentado el 6 de diciembre del 2010. Cambiaron el diseño de la interfaz de usuario, añadieron soporte para pantallas extra grandes y resoluciones WXGA, soporte nativo para VoIP SIP, reproducción nativa de videos WebM/VP8 un formato de video patrocinado por Google que es la alternativa al H264 en la reproducción de video en HTML5 y decodificación de audios en AAC, se añadió soporte a NFC (Near Field Communication), nuevo teclado multitactic, soporte mejorado para programar en código nativo, soporte nativo de más sensores como pueden ser acelerómetros o barómetros, soporte para múltiples cámaras y cambio del sistema de archivos YAFFS a ext4. La versión 2.3.3 sigue siendo la versión de Android más usada actualmente.
- **3.0 / 3.1 / 3.2 (Honeycomb):** Esta versión fue diseñada exclusivamente para tablet, por lo que no hubo smartphones que actualizaran a esta versión. Las características principales fueron un escritorio en 3D con widget rediseñado, sistema multitarea mejorado, mejoras en el navegador de internet, videochat mediante Google Talk, mejoras en el soporte de redes WiFi, añadidos soporte para gran cantidad de periféricos y conexión USB.
- **4.0 (Ice Cream Sandwich):** Fue una de las actualizaciones más importantes que ha recibido Android y fue lanzada el 19 de octubre de 2011, en ella se unificaron todas las versiones y se tenía una sola versión para smartphne, televisores, tablets, netbooks, etc. Se añadió una nueva versión de interfaz mucho más limpia y usable llamada Holo, una nueva fuente llamada Roboto, se da la opción de utilizar botones virtuales en la interfaz de usuario en vez de botones físicos, soporte para aceleración gráfica por hardware, por lo que la interfaz es manejada y dibujada por la GPU, aumentando notablemente el rendimiento, multitarea mejorada, se ha añadido un nuevo corrector ortográfico, en la lista de notificaciones se pueden eliminar las que no sean interesantes, capturas de pantalla pulsando el botón de encendido y el de bajar volumen, mejorada la aplicación encargada de hacer fotografías, añadida una nueva opción para crear fotos panorámicas, Android Beam, una nueva

característica que nos permite compartir contenidos entre teléfonos mediante NFC, reconocimiento de voz del usuario, reconocimiento facial, para bloqueo y desbloqueo del teléfono, añadidas nuevas carpetas que se crean solo con arrastrar y soltar, un único y nuevo framework para crear aplicaciones y soporte para contenedor MKV.

- **4.1 (Jelly Bean):** Esta es la última versión de Android que hay en el mercado, fue lanzada el 27 de junio de 2012 durante la última Google I/O. Se mejoró la fluidez y la estabilidad gracias al proyecto “Project Butter”, ajuste automático de widget cuando se añaden al escritorio, se añadió soporte para lenguas no occidentales, mejora de Android Beam para poder enviar video por NFC, dictado de voz mejorada y sin tener que tener conexión a internet para usarlo, nuevas notificaciones en las que se puede añadir botones para controlar o tener acceso a opciones más comunes, como puede ser responder a un email, pulsar pause o pasar de canción, nueva función Google Now que intenta ser el SIRI de Android, cifrado de aplicaciones y nuevas actualizaciones incrementales, en las que no es necesario volver a bajar toda la aplicación para actualizarla, solo se baja las partes nuevas, Google Chrome se convierte en el navegador por defecto de Android y se pone fin al soporte de Adobe Flash Player, se añade una nueva función llamada Sound Search que permite identificar la canción que está sonando, se ha añadido una nueva función llamada Gestual Mode para personas discapacitadas visualmente.

En el proyecto se ha usado la versión 4.0 para el desarrollo.

En la figura 2.8 se puede como se ve visualmente android en la actualidad con la versión 4.1.

2.5. Google App Engine.

Google App Engine es una plataforma de cloud computing para desarrollar y almacenar aplicaciones web que ofrece Google. Las aplicaciones web pueden ser escalables y si necesitan más recursos automáticamente le son asignados para poder seguir ofreciendo servicio. Google App Engine es gratis para un cierto número de peticiones y almacenamiento, la primera versión fue lanzada en abril del 2008.

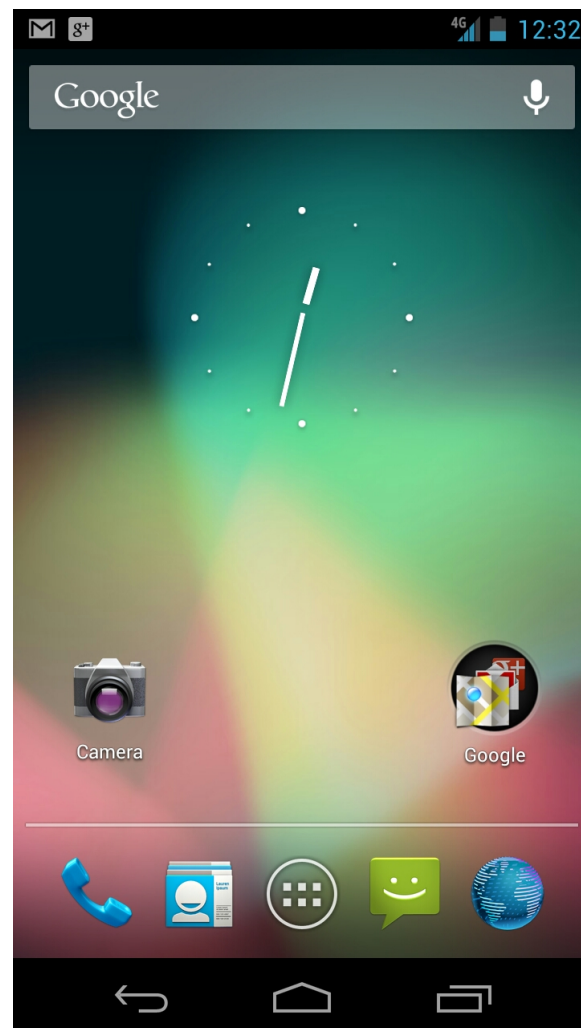


Figura 2.8: Versión de Android 4.1

Actualmente se puede desarrollar en tres lenguajes, que son Java, Python y Go, este último un lenguaje propiedad de Google. En el proyecto hemos usado el lenguaje Java para desarrollar la aplicación web.

Google App Engine para Java soporta muchos estándares y framework, y el Core está hecho con la tecnología servlet 2.5 usando el servidor de software libre llamado Jetty Web Server acompañado con otras tecnologías como JSP. El almacén de datos puede ser muy intuitivo pero se puede acceder fácilmente con JPA y los métodos JDO para escritura y lectura de datos. También se pueden usar tecnologías como Spring Framework.

Google garantiza que aplicación estará disponible el 99.95 % del tiempo, al estar altamente replicada.

La base de datos a la que dan acceso no es una base de datos que tiene una sintaxis muy parecida a SQL pero es GQL, una de las principales características es que no admite sentencias join, debido a la ineficiencia de la misma. La versión de java soporta consultas asincronas no bloqueantes, ofreciendo así una forma de procesamiento paralelo de datos.

Para más información se puede visitar la web diseñada para desarrolladores que proporciona Google, <https://developers.google.com/appengine/>.

En el capítulo 5 veremos más ampliamente todo lo relacionado con Google App Engine que se ha usado en el proyecto.

2.6. SQLite.

SQLite es un sistema de gestión de bases de datos relacionadas compatibles con ACID, ACID es un acrónimo de Atomicity, Consistency, Isolation and Durability que son todas las características que deben de tener una bases de datos para que se consideren bases de datos relacional. La característica principal es que ocupa muy poco espacio alrededor de 275 Kb y fue escrita en C por Richard Hipp. Está distribuida bajo licencia de dominio público.

A diferencia de los sistemas de gestión de bases de datos clientes-servidor el motor de SQLite pasa a ser parte del programa que quiere usarlo ya que se integra con él. Esto hace que tenga mayor rendimiento debido a la co-

municación por medio de funciones es mucho más eficiente que mediante comunicación de procesos. La totalidad de la base de datos, tablas, índices y los datos, se guardan en un solo fichero estándar en la máquina host. La versión 3 de SQLite permite base de datos de hasta 2 Terabytes y permite campos del tipo BLOB.

SQLite está muy extendido y se puede programar en infinidad de lenguajes de programación como pueden ser C, C++, Perl, Python, PHP, Java, etc.

Es utilizado en infinidad de programas y sistemas, que van desde editores de imagen como puede ser Adobe Photoshop Elements, reproductores de sonido como Clementine o navegadores como Firefox, Chrome u Opera.

Esta es una de las tres formas que proporciona Android para guardar datos, al estar embebida en cada aplicación para mejorar el rendimiento, cada aplicación debe de tener una.

2.7. XML.

XML son las siglas en inglés de eXtensible Markup Language que es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C), deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos para estructurar documentos grandes.

Es una de las formas de intercambio estructurado de información mas extendidas en internet, ya que se puede usar en base de datos, hojas de cálculo o en casi cualquier información que se quiera usar.

XML es un lenguaje que puede ser analizado sintácticamente para averiguar si está bien formado o no, por lo que para que cualquier parser (analizador sintáctico) confirme que es correcto debe de mantener una estructura que viene definida en el estándar. Todos los documentos tiene que tener las siguientes partes para que sean bien formados, prólogo, cuerpo, elementos y atributos.

Un ejemplo de la figura 2.9 podemos ver un trozo de código XML para almacenar un libro en una librería.

```
1 <?xml version="1.0"?>
2 <libro>
3   <titulo> Cien años de soledad </titulo>
4   <disponible tiempo="24" unidad="horas"/>
5   <autor> Gabriel García Márquez </autor>
6   <formato> Rústica </formato>
7   <publicacion>1967 </publicacion>
8   <precio cantidad="9.99" moneda="euro"/>
9   <descuento cantidad="5"/>
10  <enlace libro href="/exec/ISBN/84-473-0619-4"/>
11 </libro>
```

Figura 2.9: Código ejemplo XML.

En el proyecto hemos usado XML para los archivos de configuración de android o para el diseño de las interfaces en android. En una de las aplicaciones web realizadas para los archivos de configuración también hemos usado XML, en la otra hemos usado un lenguaje parecido llamado YALM, que es equivalente a XML.

2.8. UML.

UML es un lenguaje de modelado de propósito general más usado en la actualidad para el diseño de software. UML son las siglas de Unified Modeling Language. UML tiene la ventaja de que se puede observar visualmente el diseño del software. Se puede desde especificar, construir o documentar un sistema o un software.

Hemos usado UML para diseñar las clases usadas a lo largo de todo el proyecto, además de para mostrar en la memoria del proyecto la relación de las clases, los casos de uso...

2.9. GIT.

Git es un software de control de versiones a la vez que mantenedor al coherencia y cohesión del código fuente orientado a la velocidad. Fue desarrollado para el manejo del código fuente de linux y al principio fue diseñado por Linus Torvalds. GIT es un repositorio con un completo historial y una capacidad de identificación de cada cambio realizado sin depender del acceso a la red o a un servidor central. Está liberado con licencia GNU versión 2.

En el proyecto hemos estado usando GIT como control de versiones por si en algún cambio ocurría algún problema y debíamos volver atrás y para tener una copia de seguridad del proyecto alojado en todo momento en un servidor externo por si había algún fallo.

A lo largo del proyecto hemos usado dos servicios gratuitos como el conocido <https://github.com/> y <https://bitbucket.org/>. El primero es muy famoso actualmente y hay mucho software libre en dicha plataforma, tiene un problema y es que el código tiene que ser libre y en la versión gratuita no se pueden tener repositorios privados, cosa que en el segundo servicio si te los ofrece. Para todo el código fuente usando, tanto en la aplicación de android como en las dos aplicaciones hemos usado bitbucket y para la memoria de proyecto hemos usado github. La dirección del repositorio de la memoria es la siguiente: <https://github.com/t321/memoriaPFC>, el repositorio de las aplicaciones web <https://bitbucket.org/t321/pfcaeg> y el de la aplicación de Android <https://bitbucket.org/t321/pfcandroid>.

La configuración básica para el uso de GIT tanto con Eclipse como con la terminal se puede ver en el apendice !!!?????¿¿

Capítulo 3

Criptografía usada en el proyecto.

En la sección 2.3 del capítulo 2 hemos hecho una breve introducción a la criptografía de clave pública, a continuación vamos a explicarla con más extensión y los usos que le hemos dado en el proyecto.

Toda la criptografía que hemos usado es criptografía de clave pública, y en concreto el criptosistema llamado RSA. Se pensó en usar DSA pero finalmente usamos RSA. En el proyecto se ha usado la criptografía para firmar digitalmente, no para cifrar los mensajes.

3.1. RSA.

RSA se basa en la idea de que multiplicar dos números enteros primos, pero muy difícil sabiendo el resultado de la multiplicación averiguar dichos dos números.

El proceso para generar las claves es el siguiente:

- Se eligen dos números primos grandes que llamaremos **p** y **q**, dichos números se multiplican y obtenemos **n**, $n = p \cdot q$, dicho valor de n es el que se usa como módulo de la clave privada y pública.

- Calculamos $\varphi(n)$ de la siguiente forma $\varphi(n) = (p-1) \cdot (q-1)$, donde $\varphi(n)$ es la función φ de Euler.
- Se elige un entero positivo e menor que $\varphi(n)$ y que sea coprimo con $\varphi(n)$. El valor e es el exponente de la clave pública.
- Se busca un valor d que satisfaga la congruencia $d = e^{-1} \bmod \varphi(n)$. Este valor se suele calcular con el algoritmo de Euclides extendido y el valor d es el exponente para la clave privada.

La clave pública será (n, e) y la clave privada será (n, d) .

El proceso de cifrado y descifrado será el siguiente.

- **Cifrado:** el receptor (Alice) envía su clave pública (n, e) al emisor (Bob) y guarda la clave privada, (n, d) , en secreto, a partir de este momento Alice usando la clave pública de Bob puede comunicarse seguramente. El mensaje que Bob quiere enviar a Alice será M . Bob primero convierte M en un número menor que n , y que sea de una forma reversible de forma que sabiendo n podamos volver a conseguir M . Acto seguido calcula c de esta forma, $c \equiv m^e \pmod{n}$. Bob mandaría c a Alice y la comunicación finalizaría.
- **Descifrado:** Alice empezará el proceso para recuperar m a partir de c y d . $m \equiv c^d \pmod{n}$, una vez ha calculado m puede conseguir M .

El proceso de descifrado funciona porque $c^d = (m^e)^d \equiv m^{ed} \pmod{n}$ y hemos elegido d y e de forma que $e \cdot d = 1 + k \cdot \varphi(n)$, se cumple que $m^{ed} \cdot m^{1+k \cdot \varphi(n)} = m(m^{\varphi(n)})^k = m \pmod{n}$, esta última congruencia se obtiene directamente del teorema de Euler cuando m y n son coprimos.

Un ejemplo de cifrado y descifrado mediante el algoritmo RSA es el siguiente:

- Elegimos $p = 61$, $q = 53$. Podemos observar que ambos son primos. Se ha elegido para el ejemplo unos valores pequeños para que podamos hacer los cálculos con facilidad.
- Calculamos $n = p \cdot q = 3233$.

- Elegimos $e = 17$ y $d = 2753$, como exponente público el e y como exponente privado el d . La clave pública quedaría como $(17, 3233)$ y la clave privada sería $(2753, 3233)$.
- La función de cifrado quedaría como la siguiente: $\text{encrypt}(m) = m^e \bmod(n) = m^{17} \bmod(3233)$, donde m es el texto sin cifrar. Imaginemos que $m = 123$, luego los cálculos quedarían, $\text{encrypt}(123) = 123^{17} \bmod(3233) = 855$.
- La función de descifrado sería: $\text{decrypt}(c) = c^d \bmod(n) = c^{2753} \bmod(3233)$, donde c es el mensaje cifrado. Usando la función para descifrar $c = 855$ obtenemos lo siguiente: $\text{decrypt}(855) = 855^{2753} \bmod(3233) = 123$.

Podemos observar que hemos conseguido cifrar y descifrar de forma segura usando dos claves diferentes y una de ellas siendo pública.

Una de las características del algoritmo RSA es que permite la verificación de la veracidad de documentos mediante un proceso que se llama firma digital. Con ello no garantizamos que un atacante no pueda leer el mensaje, ya que no daría igual, pero si podemos garantizar que dicho mensaje no ha sido modificado.

La forma de realizar la firma es la siguiente, en vez de firmar todo el documento, que sería un gran problema ya que estos algoritmos son muy lentos, Alice genera un hash del documento con una longitud fija (128 o 256 bits) y se firma dicho hash con su firma privada. Una función hash es una función matemática que para una entrada devuelve un valor único, por lo que podemos garantizar que un documento solo va a devolver un único valor numérico y si modificamos algo del documento ese valor se alterará. Acto seguido se envía el valor de la función hash firmado junto con el documento original. Cuando Bob recibe el mensaje lo único que tiene que hacer es volver a generar el hash del documento de la misma forma que lo hizo Alice y usando la clave pública de Alice descifrar el hash firmado. Si ambos valores son iguales podemos garantizar que el mensaje es el original que Alice quería enviar.

En el proyecto hemos usado librerías que proporciona Java en su API *java.security*.*. En dicha librería, como veremos en la sección 3.3 con más profundidad, proporciona todas las utilidades para manejo y creación de claves públicas y privadas, keystores, certificados de clave pública, funciones hash, algoritmos como RSA o DSA y algoritmos de cifrados simétricos, etc.

Generalmente cuando se intercambian las claves no se pasan los números en un par como hemos hecho anteriormente ya que los números primos no son de dos dígitos son números primos muy grandes, se usan contenedores de certificados de claves en los que van incluidos ambas claves o solo la pública, además de más datos como entidades que verifican que el certificado es real. Uno de los grandes problemas que tiene la criptografía de clave pública es que cualquiera puede generar un certificado y decir que es otra persona, por lo que se necesita de entidades ajenas y que no tengan ningún compromiso con ninguna de las partes. Esto es lo que se llama una PKI (Public-Key Infrastructure).

3.2. Public-Key Infrastructure.

Una PKI es una combinación de hardware, software, políticas y procedimientos de seguridad que garantizan las operaciones de cifrado, firma digital y el no repudio de las transacciones electrónicas, y que se encargan de crear, almacenar, distribuir, usar y revocar los certificados digitales usados por empresas y particulares para que garantizar las comunicaciones electrónicas. Esto se puede garantizar gracias a autoridades de certificación, que son empresas o entidades públicas que garantizan todo lo anterior.

Hay varias entidades dentro del sistema como puede ser una CA (Certificate Authority) que es la entidad que firma todos los certificados, solo se usa para autofirmarse el certificado de ella y para firmar a las entidades certificadoras subordinadas, si este nodo del sistema es comprometido, toda la estructura caerá. Las autoridades subordinadas son las encargadas de certificar digitalmente a un usuario y garantizar que el usuario es real y de verdad es quien dice ser. Las VA (Validation Authority) es la autoridad que se encarga de comprobar el estado del certificado emitido, si es válido o está revocado. Para asegurar que cada clave pública está única y pertenece a un usuario real se usa una RA (Registration Authority), la cual se encarga de verificarlo. Hay otra entidad importante que son los repositorios para almacen de certificados y los repositorios de listas de revocación de certificados. La lista de revocación es un componente muy importante de la PKI, ya que en dicha lista están los certificados revocados antes del tiempo indicado dentro del certificado. También suelen incluir TSA (TimeStamp Authority) que es la encargada de certificar que la operación fue realizada a un tiempo en concreto.

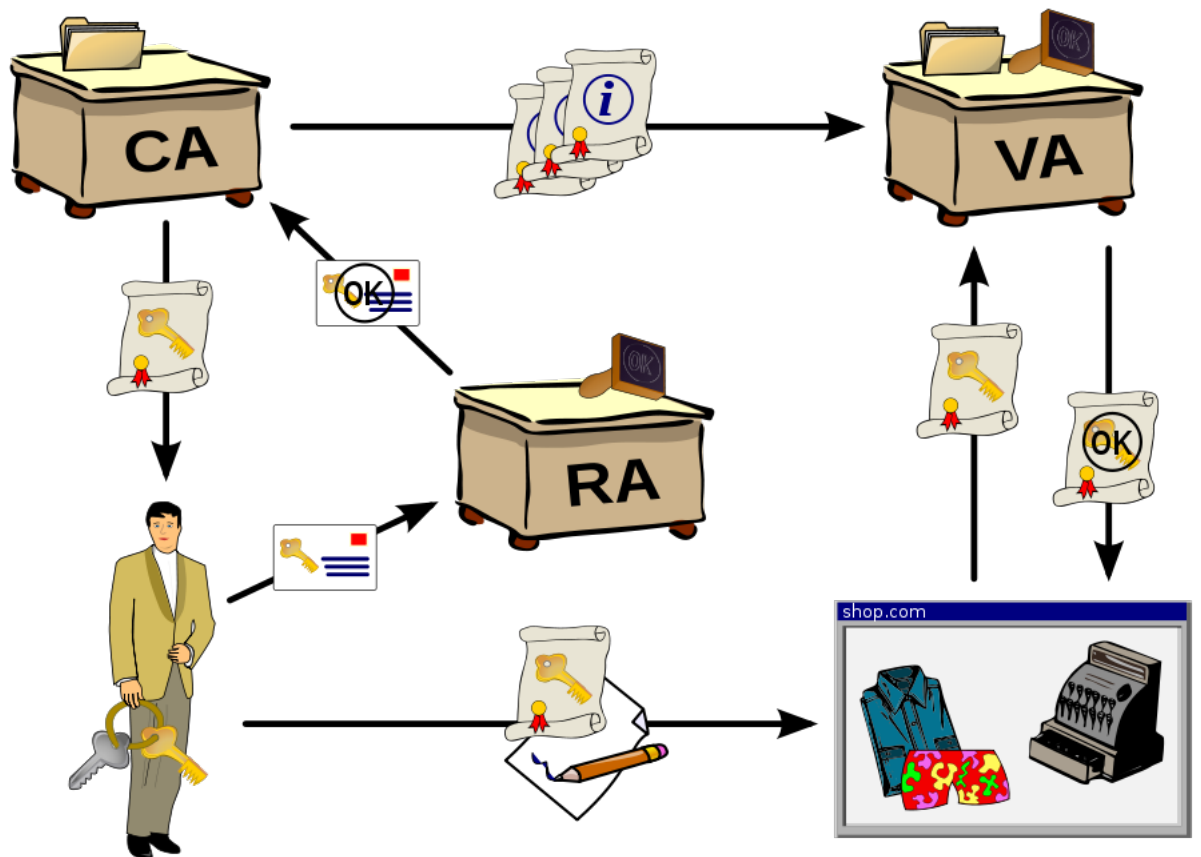


Figura 3.1: Estructura genérica de una PKI.

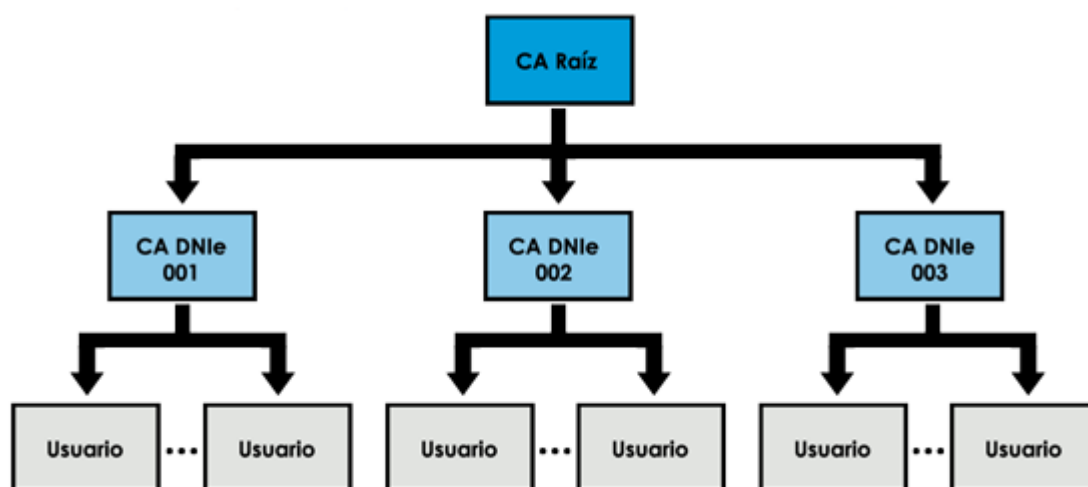


Figura 3.2: Jerarquía de la PKI del DNIe.

En la imagen 3.1 podemos ver todas las autoridades que hemos explicado anteriormente.

En España tenemos dos PKI, la FNMI y la creada por la Policía Nacional para dotar al DNIe de todo los certificados necesarios para que se pueda verificar la veracidad de dicho usuario de forma telemática. Tiene una estructura en la que hay una CA principal que está en la raíz y de ella cuelgan tres autoridades certificadoras subordinadas las emiten los certificados que se usan en el DNIe. La VA serían las comisarías donde se consigue el DNIe, donde hay que certificar que eres dicha persona para conseguir el DNIe. En la figura 3.2 se puede ver la estructura.

Una PKI puede garantizar la veracidad de la autenticación frente a otros usuarios y proporciona métodos para conseguir certificados de identidad de otros usuarios para firmar digitalmente, cifrar y descifrar. En toda operación hay tres pasos principales, el primero es el inicio de un usuario de la operación, el segundo un sistema de servidores que dan fe de que la operación ha ocurrido y garantizan la veracidad, como puede ser servidores de timestamp, la autoridad de certificación y la autoridad de registro, el tercer paso es el receptor recibe los datos firmados o cifrados y puede garantizar la veracidad de los documentos.

La tecnología PKI se puede usar para los siguientes casos, autenticación de usuarios y sistemas, identificación del interlocutor, cifrado de datos digi-

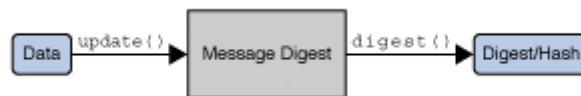


Figura 3.3: Uso de la clase MessageDigest.

tales, firmado digital de datos, tales como documentos, software, etc, asegurar comunicaciones y garantía de no repudio.

Existen varios tipos de certificados entre los que podemos encontrar certificados de tipo personal, que acredita la identidad del titular, un certificado de pertenencia a una empresa, que además de garantizar la identidad garantiza la empresa para la que trabaja, certificado de representante, que acredita los poderes que tiene dicho usuario sobre la empresa, certificado de servidor seguro, que utilizan los servidores para asegurar las conexiones con sus clientes, certificados de firma de código para garantizar que los cambios realizados en el código son legítimos y muchos otros tipos.

En el ámbito de servidores y sistemas informáticos hay varias PKI importantes como Verisign: <http://www.verisign.com/>, o Comodo: <http://www.comodo.com>

En el proyecto hemos usado certificados generados por una CA imaginaria, en el anexo podemos ver como generar certificados, para poder hacer las pruebas pero si el proyecto se llevase a cabo podría ser fácilmente extensible con una PKI propia de la Universidad de Málaga o con cualquier que dote de validez legal en España.

3.3. Critografía en Java.

Como ya hemos dicho Java dota de una API llamada *java.security* donde proporciona todo tipo de algoritmos de hash, de firma digital, cifrado, generación de números aleatorios, etc. En <http://docs.oracle.com/javase/6/docs/technotes/guides/security/> está toda la documentación necesaria sobre todos los métodos de cifrado o firmado que proporciona la API.

El paquete *java.security.** proporciona diferentes clases que vamos a explicar a continuación:

```
1  MessageDigest md = MessageDigest.getInstance("SHA");  
2  md.update(toChapter1);  
3  byte[] toChapter1Digest = tc1.digest();
```

Figura 3.4: Ejemplo de un trozo de código fuente con el uso de MessageDigest.

MessageDigest: es la clase que proporciona todos los mecanismos para realizar funciones hash. La forma de generar el hash se puede observar en la figura 3.3.

La forma de uso es la siguiente, se genera un objeto del tipo MessageDigest, el objeto se inicializa con la función getInstance() de la fábrica de métodos estáticos, acto seguido se usa el método update() para añadir los byte del mensaje al que queremos realizar la función hash y seguidamente usamos la función digest() que devuelve el hash del mensaje que hemos introducido con el método update(). Un ejemplo de esto se puede ver en la figura 3.4.

Signature: es la clase que proporciona todos los métodos de firmado mediante algoritmos DSA o RSA. En la figura 3.5 podemos ver el modo de uso. Para usarlo primero hay que poseer una clave (Key), puede ser una clave pública o privada y dependiendo de cual poseamos podemos firmar o verificar una firma. En el proyecto hemos usado la instancia *SHA1withRSA*.

El uso es muy parecido a la clase de **MessageDigest**, lo primero es crear un objeto usando una de las clases fábrica que queramos, acto seguido usamos el método initSign(Key) con el cual añadimos la clave al proceso. Acto seguido usamos el método update() y añadimos los datos con los que queremos operar, tanto para firmar o verificar. Si lo que queremos es firmar usamos la función sign() y si queremos verificar la función verify(). La función sign() devuelve un array de bytes que sería la firma de los datos y la función verify() devuelve un booleano con valor true si la firma es válida o false si no se puede verificar.

En la figura 3.6 se puede observar un trozo de código donde se firma un mensaje. En la figura 3.7 podemos ver un ejemplo de una verificación de un texto cifrado.

Cipher: es una de las clases que más posibilidades ofrece, ya que implementa un gran número de algoritmos de cifrado simétrico. En la figura 3.8 se puede observar la forma de uso.

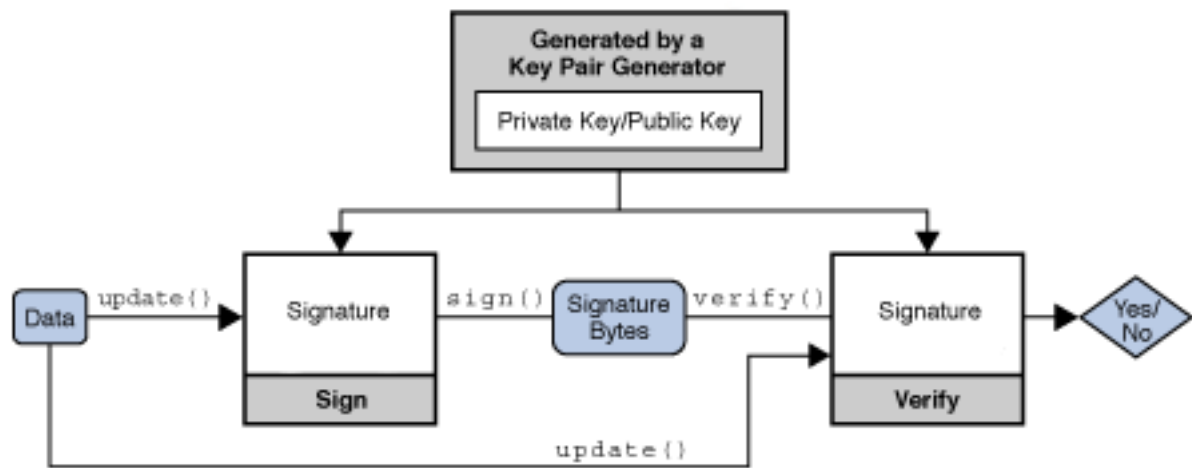


Figura 3.5: Modo de uso de la clase Signature.

```

1  Signature instance = Signature.getInstance("
    SHA1withRSA");
2  instance.initSign(key);
3  instance.update((plainText).getBytes());
4  byte[] signature = instance.sign();

```

Figura 3.6: Ejemplo de un trozo de código fuente con el uso de la función sign().

```

1  Signature instance = Signature.getInstance("
    SHA1withRSA");
2  instance.initSign(key);
3  instance.update((cipherText).getBytes());
4  boolean verify = instance.verify();

```

Figura 3.7: Ejemplo de un trozo de código fuente con el uso de la función verify().

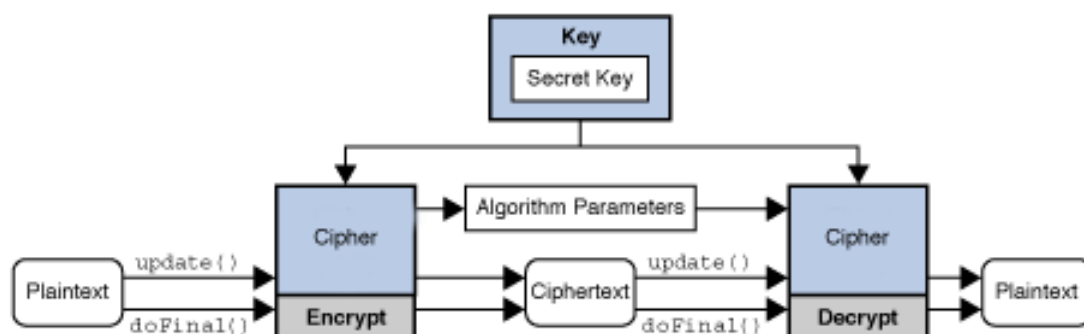


Figura 3.8: Modo de uso de la clase Cipher.

En el proyecto solo hemos usado criptografía de clave pública, pero la forma de uso es sencilla, se genera un objeto llamando a la fábrica de objetos que tiene la clase con `getInstance()` mandándole una cadena con el algoritmo que queremos usar, DES, AES, etc y además el padding usado, si es necesario, como por ejemplo “DES/CFB8/NoPadding.” o “DES/OFB32/PKCS5Padding”, acto seguido debemos de usar el procedimiento `init()` donde debemos de indicar si el objeto se va a usar para encriptar o desencriptar y la clave usada, para finalizar podemos usar la función `doFinal()` a la que le pasaríamos el texto cifrado o en claro dependiendo del tipo de proceso que queramos usar.

Existen varias clases que se basan en la clase Cipher para cifrado de flujos o archivos enteros sin tener que ir byte a byte. Por ejemplo **CipherInputStream** y **CipherOutputStream**.

Key: para el manejo de las claves dentro del paquete hay una interfaz con la que se manejan todos los tipos de claves que podemos tener, desde claves públicas y privadas hasta claves de cifrado simétrico. Esta interfaz la implementan un gran número de clases como pueden ser `PrivateKey`, `PublicKey`, `SecretKey`, `RSAPrivateKey`, `RSAPublicKey`, `DSAPrivateKey`, `DSAPublicKey`, etc. Para generar las claves hay que usar clases factoría como pueden ser `KeyGenerator`, `KeyPairGenerator` o `KeyFactory` dependiendo de si queremos generar una clave simétrica o un par de claves o usar `KeyStore` si lo que queremos es cargarla llave de un llavero (archivo de claves) que tengamos.

En el proyecto las no hemos tenido que generar claves, ya que queríamos que se pudiera cargar un llavero con la clave privada y pública, en el trozo

```
1  KeyStore ks = KeyStore.getInstance("PKCS12");  
2  ks.load(new FileInputStream(path), password.  
    toCharArray());  
3  PrivateKey key = (PrivateKey) ks.getKey(ks.aliases().  
    nextElement(), password.toCharArray());
```

Figura 3.9: Ejemplo de un trozo de código donde se carga un KeyStore.

de código de la figura 3.9 podemos ver como se usa la clase KeyStore en el proyecto.

Como podemos ver lo primero que tenemos que hacer es generar un objeto KeyStore, usando la función `getInstance()` a la que tenemos que pasarle el tipo de llavero que vamos a usar, nosotros hemos usado PKCS12¹ que es un archivo creado especialmente para almacenar claves. Acto seguido cargamos el fichero, con el método `load()`, al cual tenemos que pasarle la ruta hasta el archivo y el password del archivo PKCS12. Para conseguir la clave privada usamos la función `getKey()` a la que hay que pasarle el alias con el que está guardada la clave en el llavero y el password para poder sacarla. En el código se puede ver un casting ya que la función `getKey()` devuelve un elemento que implementa la interfaz `Key`, pero nosotros sabemos que es un objeto de la clase `PrivateKey`. Acto seguido ya podemos usar la clave para firmar lo que necesitamos.

¹Para más información visite <http://en.wikipedia.org/wiki/PKCS12>.

Capítulo 4

Android

4.1. Introducción.

Android es un sistema operativo basado en Linux, libre y multiplataforma. Inicialmente empezó como un sistema operativo solo para móviles pero con el tiempo ya podemos encontrar sistemas android tanto en móviles, tablets, pc, neveras, relojes, cámaras de fotos y una gran cantidad de aparatos.

En el proyecto lo usaremos para diseñar y desarrollar una aplicación móvil, con la que poder firmar digitalmente un texto leído previamente mediante un lector de códigos QR.

Android es propiedad de Google actualmente y es el encargado de dar soporte y ayudar a los desarrolladores. Esta función la hace muy bien, dando una muy buena API y una gran documentación. Podemos encontrar toda la información que necesitamos para crear una aplicación en esta web, <http://developer.android.com/index.html>. En la web anterior podemos encontrar tanto la documentación de la API, como consejos de diseño para que la aplicación tenga un aspecto bonito y usable, todas las novedades incluidas en las versiones nuevas del sistema operativo, etc.

Android también proporciona una plataforma en la que cualquier desarrollador puede publicar las aplicaciones que hace, sin restricciones.

4.2. Arquitectura de la plataforma Android.

Como hemos dicho anteriormente Android es una plataforma que engloba desde el sistema operativo, al software intermedio que comunica el sistema operativo y las aplicaciones, llamado en inglés middleware y las posibilidad de hacer funcionar las aplicaciones en la plataforma elegida, ya sea un telefono, una tables o cualquier aparato con Android.

Para los desarrolladores Android proporciona dos kit de desarrollo, uno que usa la tecnología Java (SDK), el que hemos usado durante el proyecto y otro que da la posibilidad de programar a más bajo nivel (NDK), este último desarrollado en C++.

El SDK de Android proporciona ayuda para las siguientes características, un navegador basado en WebKit, graficos optimizados en 2D, gráficos en 3D basados en OpenGL ES 1.0 con aceleración gráfica, una base de datos para almacenar datos que necesitemos, llamada SQLite, soporte para ficheros gráficos (JPG, PNG, GIF, etc), de vídeo (MPEG4, H.264) y audio (MP3, AAC), telefonía GSM, tecnologías inalámbricas como son Bluetooth, 3G, Wifi, uso de cámaras, GPS, brújulas, etc. Además de todo esto proporciona ayuda en la reutilización y remplazo de componentes, una máquina virtual optimizada para dispositivos móviles llamada Dalvik y un emulador donde poder probar antes del lanzamiento de la aplicación sin tener que poseer un móvil Android.

La arquitectura se puede ver en la figura 4.1. En ella se pueden ver las cinco capas principales en las que se divide Android.

- **Applications:** en esta capa están todas las aplicaciones que nos proporciona el sistema operativo de base, como pueden ser la lista de contactos, un gestor de SMS, el navegador, el lanzador de aplicaciones, todas las aplicaciones de servicios de Google, como pueden ser Gmail, Google Maps, Google Calendar, Google Reader.
- **Application framework:** en esta capa está todo lo relacionado con los manejadores de activitys, llamadas de telefono, controladores de vistas. Es una capa que hay intermedia para manejo de hardware, con la que se pueden controlar tanto las notificaciones, como servicios que se ejecuten en segundo plano, etc. Los desarrolladores tienen el mismo acceso mediante esta capa de la API a todos los servicios que las

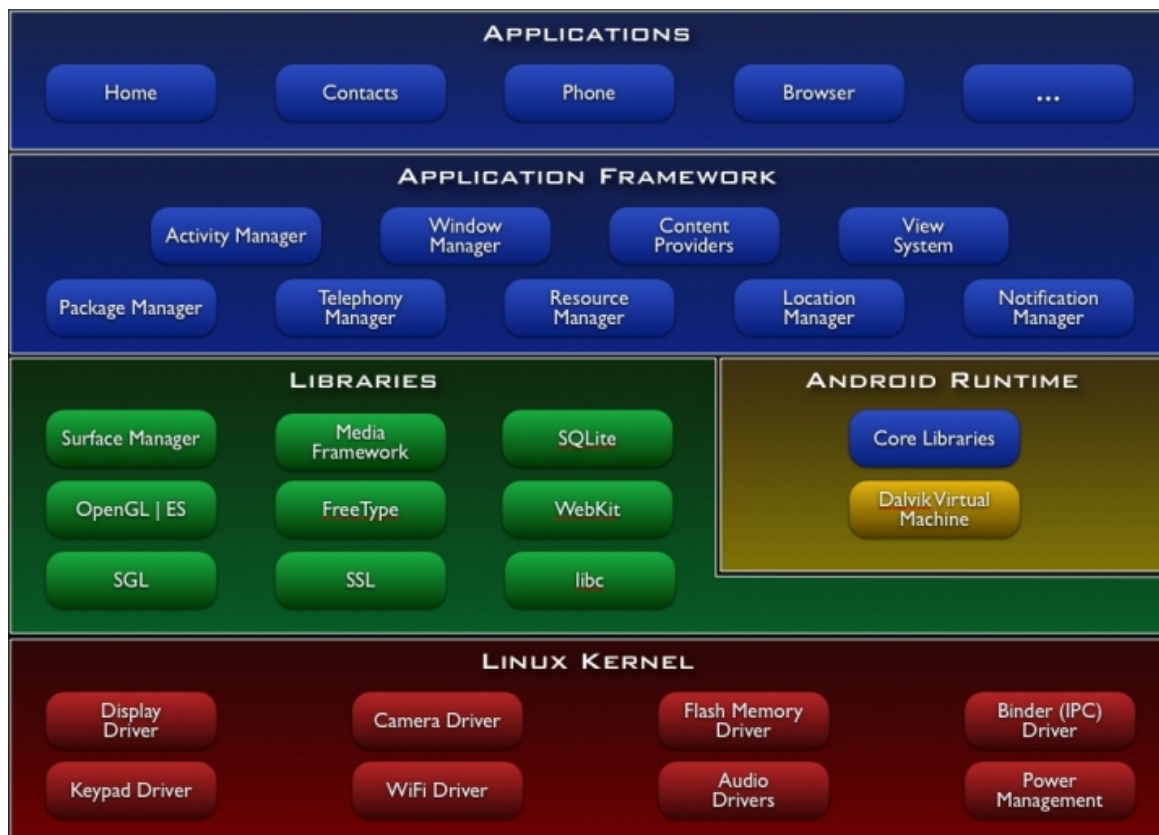


Figura 4.1: Arquitectura de Android.

aplicaciones nativas que proporciona el sistema operativo.

- **Libraries:** como su nombre indica en esta capa están todas las librerías que el sistema operativo necesita, para manejo de archivos multimedia, manejo de gráficos 3D, renderizado web, etc.
 - **System C library:** una implementación del estándar C, optimizada para funcionar en sistemas móviles para las funciones del kernel de linux.
 - **Media Libraries:** librerías basadas en PacketVideo's OpenCORE para grabación y reproducción de formatos de audio y video mas populares del momento como MP3, H.264, JPG o PNG.
 - **Surface Manager:** librería para manejo de graficos 2D y 3D para varias aplicaciones.
 - **LibWebCore:** motor de renderizado para navegadores embebidos de páginas web.
 - **SGL:** motor para renderizado 2D.
 - **3D libraries:** librería que implementa la API de OpenGL ES 1.0, que proporciona aceleración 3D por hardware si es posible o una alta optimización para renderizado por software en sistemas que no posean aceleración por hardware.
 - **FreeType:** librería para manejo de fuentes, tanto bitmap como vectoriales.
 - **SQLite:** librería para el manejo de la base de datos que proporciona Android.
- **Android Runtime:** es una capa que está al mismo nivel que la capa de librerías. En esta capa se añaden muchas librerías para dotar de la mayoría de las funcionalidades que proporciona Java, también está en esta capa la máquina virtual (Dalvik) encargada de ejecutar el código Smali de los archivos DEX.
- **Linux Kernel:** Android está basado en el kernel de Linux 2.6 y todo lo relativo a seguridad, manejo de memoria, control de procesos, pila de protocolos de red y modelo de drivers es el mismo. El kernel es el que proporciona una capa de abstracción entre el hardware y el software que usará Android.

Como ya hemos dicho antes Android corre con cada aplicación en una máquina virtual, esta máquina virtual recibe el nombre de Dalvik. Dicho

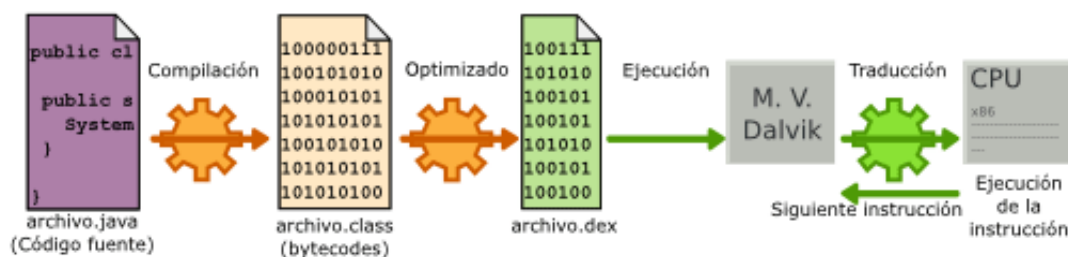


Figura 4.2: Proceso de ejecución en Android.

nombre viene de un pueblo de Islandia donde viven los familiares del creador de esta, Dan Bornstein. La máquina virtual ejecuta un byte code especial llamado DEX (Dalvik Executable), que está especialmente diseñado y optimizado para funcionar en sistemas móviles, tablets, etc. En la figura 4.2 podemos ver todo el proceso desde la creación del archivo JAVA a la ejecución.

Los ficheros DEX meten las cadenas duplicadas y las constantes en un mismo fichero para ahorrar espacio, normalmente los archivos DEX suelen ser más pequeños que los archivos JAR de la máquina virtual de Java. Una vez instalados los archivos DEX pueden ser modificados en el terminal para añadir optimizaciones, reordenado de byte en ciertos datos, quitado de clases vacías, etc. En la versión 2.2 de Android se añadió una nueva característica llamada JIT (just-in-time) que es compilación en tiempo real de los archivos DEX, por lo que se pueden añadir nuevas optimizaciones dependiendo de la plataforma.

Todas las aplicaciones de Android se distribuyen en unos archivos con extensión APK. Estos archivos no son más que archivos ZIP con la extensión cambiada. Todos deben de tener una estructura idéntica, que se explica a continuación. Contiene diferentes carpetas en las que se incluyen ficheros de configuración o necesarios para la aplicación y para comprobar la integridad de los archivos.

- **META-INF:** en un directorio que contiene tres archivos, *MANIFEST.MF* que es el archivo de manifest, *CERT.RSA* que es el certificado con el que está firmada la aplicación, *CERT.SF* que contiene el hash en SHA-1 de todos los componentes de la aplicación, un ejemplo del archivo *CERT.SF* es el siguiente:

```

Signature-Version: 1.0
Created-By: 1.0 (Android)
SHA1-Digest-Manifest: wxqnEAI0UA5n05QJ8CGMwjkgGWE=
...
Name: res/layout/exchange_component_back_bottom.xml
SHA1-Digest: eACjMjESj7Zkf0cBFTZ0nqWrt7w=
...
Name: res/drawable-hdpi/icon.png
SHA1-Digest: DGEqylP8W0n0iV/ZzBx3MW0WGCA=

```

- **lib:** esta carpeta puede contener otras dependiendo de la plataforma para la que esté diseñada la aplicación. Si por ejemplo tiene código específicamente diseñado para x86 tendrá una carpeta llamada x86, si tiene código para MIPS una llamada mips donde estaría el código especialmente diseñado para esta plataforma. Puede que dicha carpeta no exista.
- **res:** este directorio contiene los recursos que no tienen que ser compilados, como pueden ser las imágenes, los sonidos, etc.

Además de estas carpetas todos incluyen estos tres archivos.

- **AndroidManifest.xml:** es un archivo que sirve para indicar la versión, los permisos que contiene la aplicación, las referencias a librerías, las activitys que contiene la aplicación. En el siguiente listado podemos observar un extracto de los permisos que usamos en la aplicación del proyecto.

```

1  <uses-permission android:name="android.permission.
    INTERNET" />
2  <uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE" />
3  <uses-permission android:name="android.permission.
    WRITE_SETTINGS" />
4  <uses-permission android:name="android.permission.
    GET_ACCOUNTS" />
5  <uses-permission android:name="android.permission.
    USE_CREDENTIALS" />
6  <uses-permission android:name="android.permission.
    MANAGE_ACCOUNTS" />

```

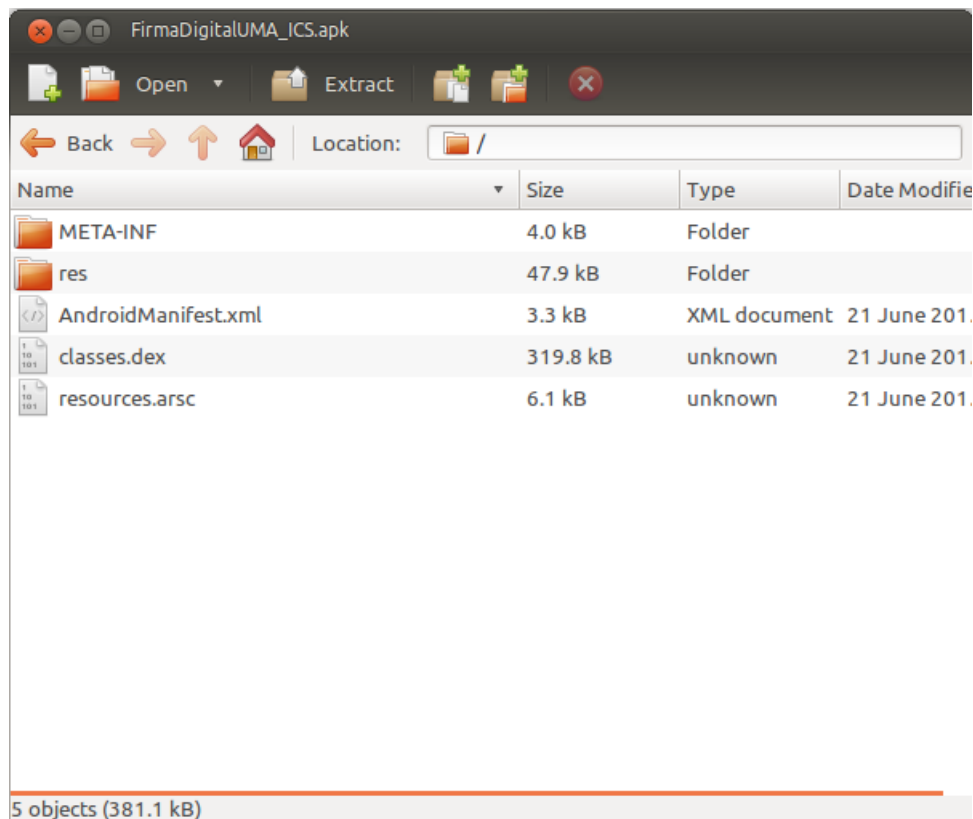


Figura 4.3: Estructura del fichero APK del proyecto.

Podemos ver que con esto garantizamos que la aplicación pueda conectarse a internet, usar la tarjeta SD y manejar las cuentas almacenadas en el móvil, como puede ser la dirección de correo usada para dar de alta el móvil en Google Play.

- **classes.dex:** el archivo DEX donde están todas las clases precompiladas en byte code para la máquina virtual Dalvik.
- **resources.arsc:** es un fichero que contiene los recursos precompilados como pueden ser los XML de las interfaces de la aplicación, etc.

En la imagen 4.3 se pueden observar los ficheros y las carpetas comentadas anteriormente del proyecto.

Toda aplicación en Android se contruye con unos componentes básicos como pueden ser activities, intents, views, services, contents providers, widgets y un largo etcétera. A continuación vamos a explicar los más importantes.

- **Activity:** una activity es la entidad más básica de una interfaz de usuario en la que se puede mostrar información en android, podemos pensar que es como una ventana de cualquier aplicación de escritorio. Cada interfaz que vemos en una aplicación de Android es un activity. Android tiene un ciclo de manejo de activitys bastante complejo, se explicará en la sección *!!?!?!?* donde pondremos más incapié en el desarrollo de la aplicación móvil del proyecto.
- **Intent:** un intent es un componente prácticamente imprescindible en cualquier aplicación de Android, es una forma de comunicación entre cualquier componente. Se pueden definir como mensajes o peticiones, ya que también puede comunicar aplicaciones entre sí. En el proyecto usamos intent para comunicar las activities y poder pasarnos datos, también la usamos para abrir el lector de códigos QR que necesitamos, dentro de nuestra aplicación llamamos al intent que nos proporciona la aplicación lectora y cuando termine ella nos devuelve el valor que había en el código QR para que podamos tratarlo.
- **View:** son los componentes básicos con los que podemos contruir las interfaces gráficas, como pueden ser botones, barras de texto, campos de texto, spinner, etc. Android pone a nuestra disposición una gran cantidad de estos elementos, y además brinda la posibilidad de crear nuevos, según los vayamos necesitando. Estos objetos normalmente se añaden a una vista y se pueden añadir, modificar o borrar en ella. Estos objetos tienen un ID único en la aplicación por el que podemos controlarlo y añadirle por ejemplo el texto si es un campo de texto o un listener si es un botón para que cuando se pulse realizar la acción que necesitemos.

En el proyecto no hemos tenido que generar controles nuevos, nos hemos bastado con los que vienen por defecto.
- **Services:** un servicio es un componente de Android que no tiene interfaz gráfica asociada y se ejecuta en segundo plano. Es similar a los servicios que ofrece cualquier sistema operativo. Pueden realizar cualquier acción, tanto recoger o actualizar datos, lanzar notificaciones cada cierto tiempo o mostrar activity para que el usuario introduzca algún valor que necesite.
- **Content Provider:** es un mecanismo que posee android para intercambiar información entre aplicaciones. Por ejemplo cuando usamos la opción de compartir en el teléfono, dentro de una aplicación, nos salen varias aplicaciones con las que podemos compartir directamente, estas

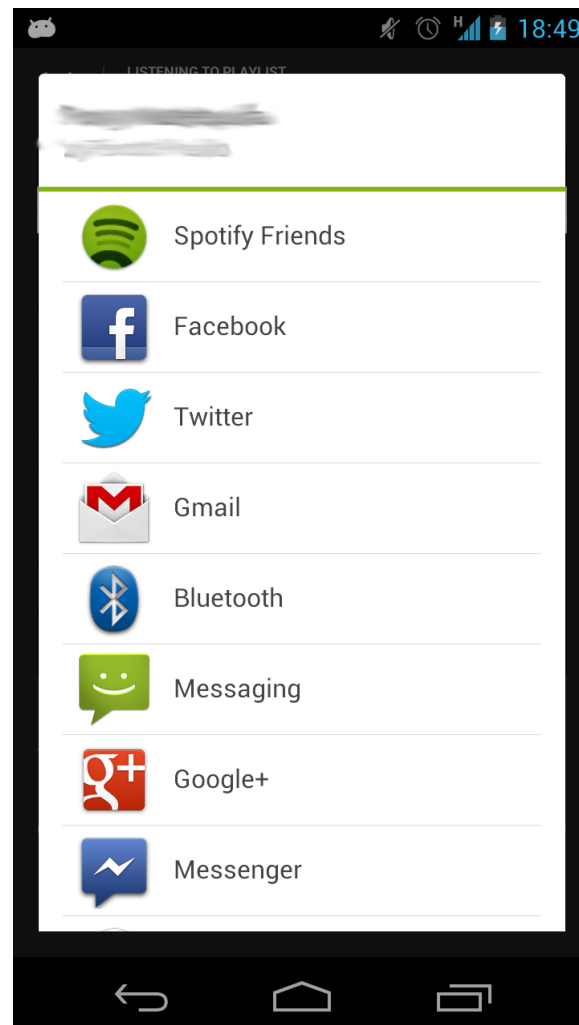


Figura 4.4: Opción compartir de una aplicación.

son todas las aplicaciones que han implementado el content provider que necesita esta aplicación para compartir la información. En la figura 4.4 podemos ver la opción de compartir de una aplicación y podemos observar como aparece por ejemplo GMail para mandar un email directamente desde aquí sin tener conocimiento de como se produce el intercambio de datos.

- **Broadcast Receiver:** es un componente de Android diseñado para actuar cuando ocurre un evento general del sistema, como puede ser la recepción de un SMS, la batería se está agotando, una llamada entrante, etc. También una aplicación puede generar eventos de este tipo para

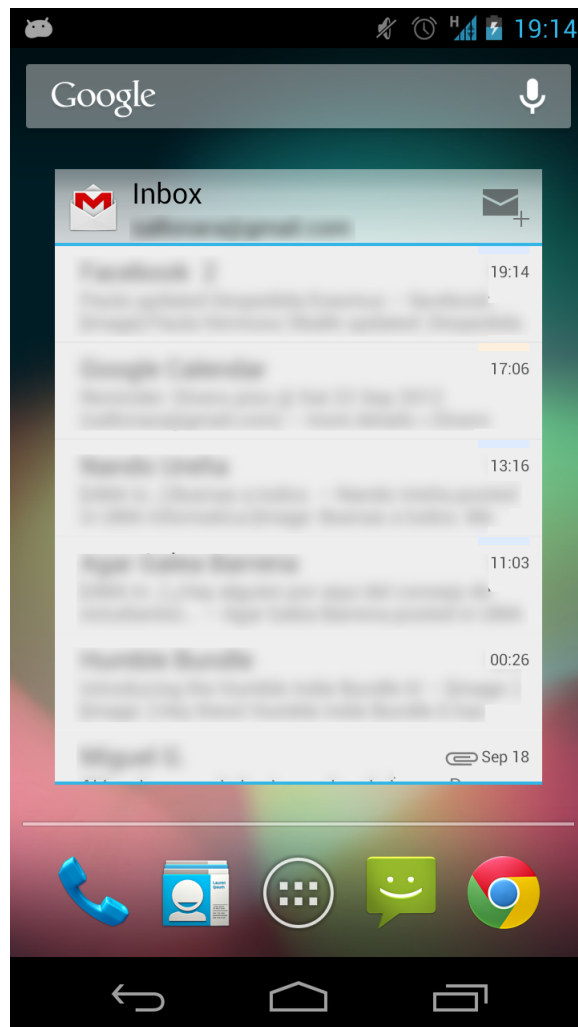


Figura 4.5: Widget de GMail.

que cualquiera que implemente un Broadcast Receiver pueda recibirlo.

- **Widget:** son elementos visuales y generalmente para que el usuario realice alguna acción, tal como poner en pausa la música, pasar de canción, revisar los feed RSS, mirar los correos pendientes, etc. Suele estar en alguna de las pantallas principales de Android. En la figura 4.5 podemos ver un widget de GMail.

4.2.1. Desarrollando en Android.

Como ya hemos dicho anteriormente para el desarrollo de la aplicación móvil hemos usado el IDE de programación Eclipse. En el anexo !!!?¿; podemos ver toda la configuración de Eclipse para la programación de aplicaciones Android. En la sección 4.2.2 se explicará con más detenimiento la estructura de un proyecto básico en Android.

En este apartado vamos a explicar las entidades más importantes que existen en Android, como pueden ser las activity, fragment, y view usados en el proyecto.

En el proyecto la entidad más importante que hemos usado es la **activity**. Normalmente las activity están enfocadas a interactuar con el usuario de alguna forma, ya puede ser necesitando de alguna acción del usuario o mostrando información y normalmente ocupando toda la pantalla del teléfono, pero puede flotar dentro de otra activity o agruparse con otras usando la clase *ActivityGroup*. Todas las subclases que hereden de Activity tienen que implementar dos métodos imprescindibles. Estos son *onCreate(Bundle)* y *onPause()*. El primero es donde se inicializa la activity y en el segundo es lo que ocurre cuando el usuario abandona la activity.

Habitualmente toda activity tiene una interfáz gráfica que se diseña en un fichero XML, como veremos posteriormente, este archivo se le tiene que indicar a cada activity con el método `setContentView(int)`; al cual hay que pasarle una constante que genera automáticamente la clase R de Android. Una vez se le ha indicado el archivo XML se puede obtener cada uno de los componentes (botones, cuadros de texto, etc) que componen la intefaz gráfica con el método `findViewById(int)`. Todo este proceso se tiene que realizar dentro del método `onCreate` y se puede observar en el siguiente trozo de código.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.initialconfigurescreen);
5
6      TextView tAccountOk = (TextView) findViewById(R.id
7          .tAccountOk);
8      EditText tCertificate = (EditText) findViewById(R.
9          id.tCertificate);
10     ImageView image = (ImageView) findViewById(R.id.
11         imageView1);
```

```
9
10     Button bSelectAccount = (Button) findViewById(R.id
11         .bConfigureScreenAccount);
    }
```

A continuación vamos a explicar las características y su ciclo de vida de las activity, ya que es un tema muy importante cuando desarrollamos una aplicación Android. En la figura 4.6 podemos ver todos los estados por los que pasa una activity desde que se crea hasta que finaliza.

Todos estos estados se pueden controlar mediante la implementación de los siguientes métodos.

```
1 public class Activity extends ApplicationContext {
2     protected void onCreate(Bundle savedInstanceState)
3         ;
4     protected void onStart();
5
6     protected void onRestart();
7
8     protected void onResume();
9
10    protected void onPause();
11
12    protected void onStop();
13
14    protected void onDestroy();
15 }
```

Como podemos ver en la figura 4.6 podemos ver que el ciclo completo de una activity es desde el método `onCreate(Bundle)`; hasta que se realiza la llamada al método `onDestroy()`; . Como hemos visto antes en `onCreate(Bundle)`; se genera todo lo necesario para que la activity funcione, como puede ser la inicialización de la interfaz, la creación de un hilo para que realice una operación en background o cualquier otra acción que necesite de una inicialización. El método `onDestroy()`; se pararía el hilo y se liberaría la memoria usada por la activity. Entre los procesos `onStart()`; y `onStop()`; es donde se mantienen los recursos para que la activity pueda mostrar los datos al usuario. Por ejemplo si tenemos un `BroadcastReceiver` que nos puede cambiar la interfaz de usuario pues lo registramos en el método `onStart()`; y lo paramos

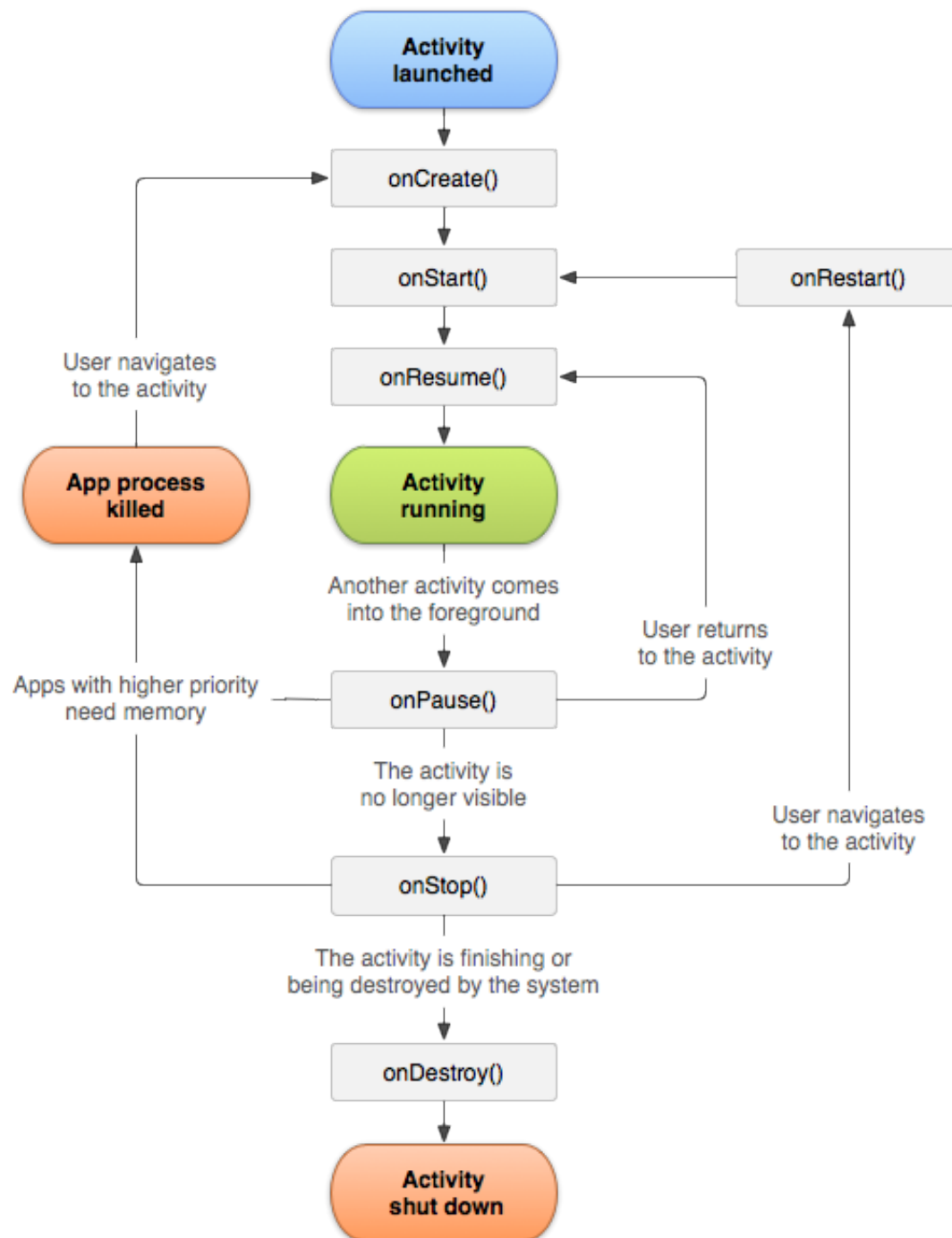


Figura 4.6: Ciclo de vida de una actividad.

en `onStop()`; . Estos dos métodos se llaman mucho a lo largo de la ejecución de la activity cada vez que el usuario oculta la activity y vuelve a ejecutarla. Los métodos `onResume()`; y `onPause()`; se usan para intercambio de activity, cuando apagamos la pantalla del móvil y volvemos a encenderla, cuando giramos la pantalla, etc. En estos métodos se suelen usar Bundles para intercambiar información entre los estados y así conseguir por ejemplo restaurar el texto de un cuadros de texto cuando vuelve a generarse.

En esta tabla podemos observar en cada estado del ciclo de vida de una activity puede ser matada y cual sería el próximo estado.

Method	¿Terminable?	Proximo estado
<code>onCreate()</code>	No	<code>onStart()</code>
<code>onRestart()</code>	No	<code>onStart()</code>
<code>onStart()</code>	No	<code>onResume()</code> o <code>onStop()</code>
<code>onResume()</code>	No	<code>onPause()</code>
<code>onPause()</code>	No	<code>onResume()</code> o <code>onStop()</code>
<code>onStop()</code>	Sí	<code>onRestart()</code> o <code>onDestroy()</code>
<code>onDestroy()</code>	Sí	Ninguna

Antes de la versión 3.0 de Android las activity tenían que ocupar toda la ventana y para cambiar o mostrar otra pantalla había que generar una nueva activity de la siguiente forma:

```

1  Intent intent = new Intent(activity,
    SplashScreenActivity.class);
2  startActivity(intent);

```

Ese trozo de código se ejecuta en una activity y podemos ver que se crea un objetos Intent al cual se le dice la activity en la que está y la activity que tiene que iniciar, en este caso la variable activity es la actual, y SplashScreenActivity es una activity que tiene la función de iniciar todas las variables y realizar las conexiones básicas en el proyecto, acto seguido se usa el procedimiento `startActivity(intent)`; al que se le pasa el objeto Intent creado anteriormente y con esto tendríamos la nueva activity corriendo.

Desde la versión 3.0 y posteriores las activity siguen ocupando toda la pantalla pero se dio la posibilidad al programador de que usara solo trozos de ella con una clase llamada Fragment de esta forma no tendría que iniciar una nueva activity cada vez que quiera modificar la interfáz, de este modo se pudieron empezar a usar gestos de scroll laterar para

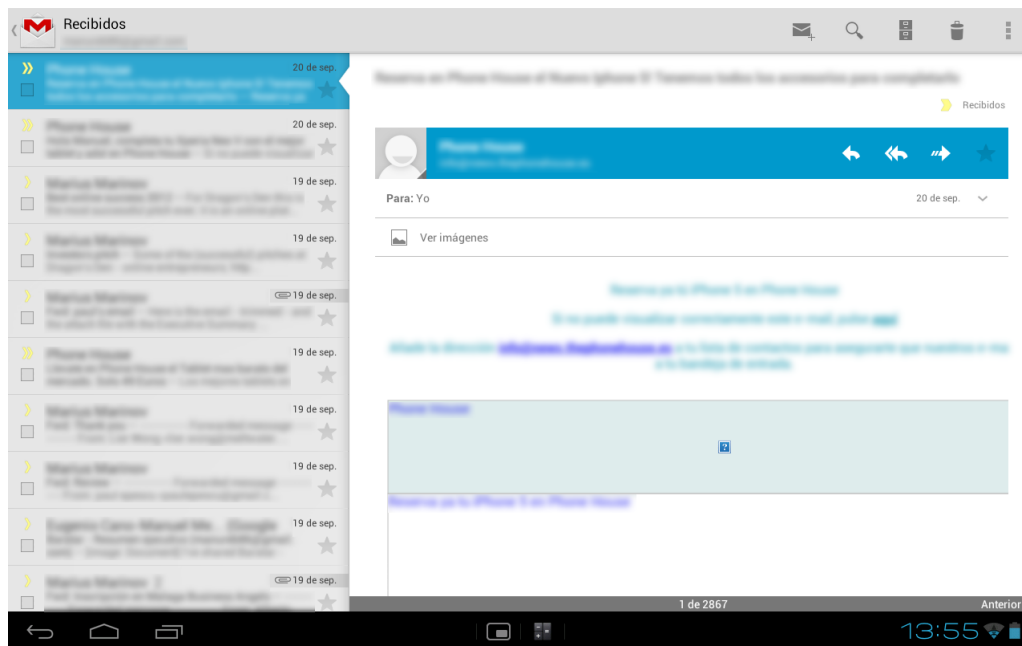


Figura 4.7: Aplicación de Gmail para tablet.

mostrar varias interfaces o en pantallas grandes como una tablet poder modificarla sin tener que generar una nueva actividad.

En la imagen 4.7 podemos ver la aplicación de Gmail diseñada mediante fragment y en ella si pulsamos algún correo en la parte izquierda de la aplicación nos mostraría el correo en la derecha sin tener que recargar la aplicación. En la imagen 4.8 podemos ver como en la versión movil no se usa esta forma por falta de espacio en la pantalla. Nosotros hemos realizado un diseño para intercambio de fragment mediante un gesto llamado swype o scroll lateral y como se puede ver en la imagen 4.9 podemos ver que no hay que volver a cargar otra activity ni nada, por lo que dotamos a la aplicación de una mayor fluidés.

Para que una aplicación pueda usar una determinada actividad, el programador primeramente tiene que definir el uso y su función en el archivo *AndroidManifest.xml*. Podemos ver un extracto de dicho archivo donde definimos una de las actividades del proyecto.

```

1 <activity android:name=".FirmaDigitalUMA_ICSAActivity
  " />
2 <activity android:name=".InitialConfiguration"
  android:noHistory="true" />

```

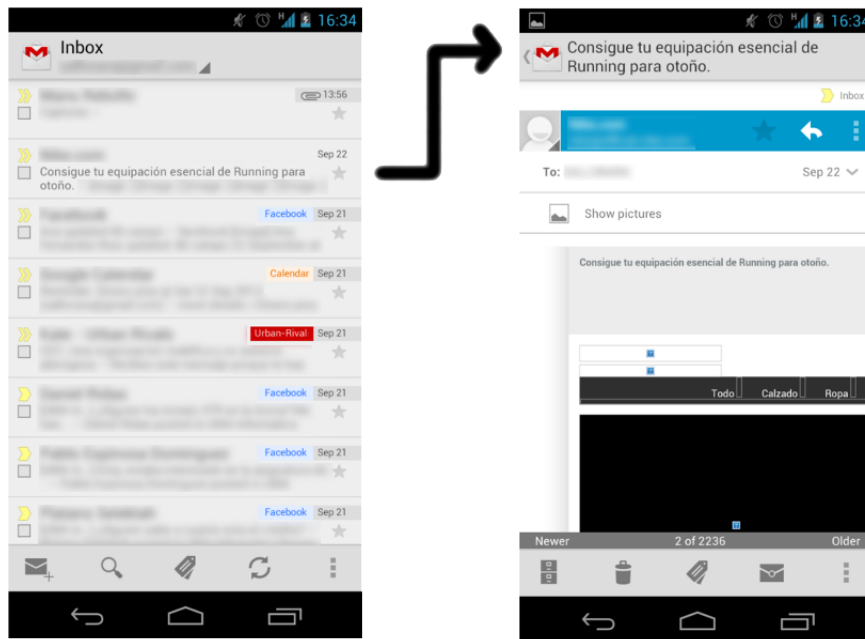


Figura 4.8: Aplicación de Gmail para móvil.

Se puede observar que hemos declarado dos actividades una sin ninguna opción y otra en la que no se guardará en la pila de llamadas de actividad, por lo que si pulsamos el botón atrás no se abrirá de nuevo. Si no realizamos este proceso nos dará un error en tiempo de ejecución la aplicación diciendo que hemos intentado ejecutar una actividad que no está declarada.

4.2.2. Proyecto básico de Android en Eclipse.

A continuación vamos explicar con más detenimiento la estructura que tiene un proyecto básico de Android en Eclipse.

En la figura 4.10 podemos ver una captura de un proyecto acabado de crear. Podemos observar que se genera una carpeta principal de la que posteriormente colgarán el resto de carpetas necesarias. Estas carpetas son *src*, *gen*, *assets*, *bin*, *res* y varios archivos sueltos como con *AndroidManifest.xml*, *proguard-project.txt*, *project.properties*, vamos a explicar brevemente que contiene y cual es la función de dichas carpetas y documentos.

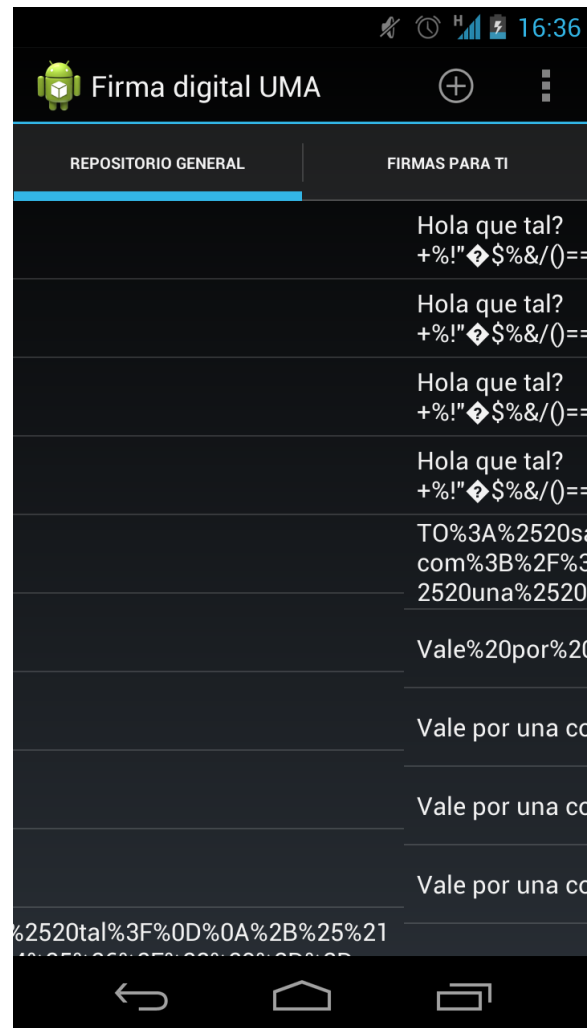


Figura 4.9: Gesto Swype en la aplicación móvil.

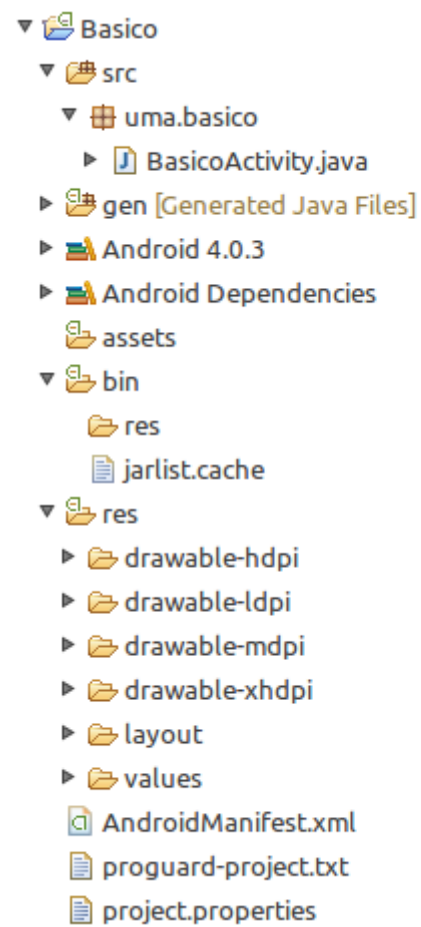


Figura 4.10: Estructura básica de un proyecto Android.

- **src:** en esta carpeta está todo el paquete que contiene los archivos de código fuente que se necesitan en el proyecto.
- **gen:** esta carpeta es donde se almacena todo lo que el proyecto de android necesita para funcionar, casi todos los ficheros que se encuentran en el interior se generan cada vez que se construye el proyecto y si los modificamos nosotros cuando volvamos a construir el proyecto borrará todos los cambios. Dentro está la clase R donde se declaran la mayoría de las constantes con direcciones de memoria que luego en tiempo de ejecución se usarán para realizar la conversión en bytecode del archivo java.
- **bin:** es una carpeta donde se almacenan todos los archivos binarios, como puede ser el archivo APK, los archivos DEX, etc.
- **res:** esta carpeta la encargada de contener todos los recursos necesarios para nuestra aplicación. Esta carpeta se divide en varias, como por ejemplo *drawable-hdpi*, *drawable-ldpi*, *drawable-mdpi*, *drawable-xhdpi* es donde se añaden todas imágenes usadas, sonidos, videos, etc. Pero no todos los recursos son contenido multimedia, hay otras carpetas como por ejemplo la carpeta *layout* donde se almacenan las diferentes interfaces usadas en el en formato XML o la carpeta *values* donde se guardan todas las cadenas constantes en un archivo XML y se pueden usar posteriormente en una interfaz gráfica por ejemplo.
- **AndroidManifest.xml:** como ya hemos explicado anteriormente es el archivo donde se declara todos los permisos e información de interes de la aplicación, como pueden ser las activity, los intent, el SDK mínimo que tiene que poseer el móvil para ejecutar nuestra aplicación, etc.
- **proguard-project.txt:** ProGuard es una herramienta que ofrece Google dentro del SDK de Android para ofuscación de código, ya que hay muchas herramientas de ingeniería inversa que mediante la decompilación de los archivos DEX se puede llegar casi a conseguir el código realizado sin permiso. En este archivo se puede configurar los diferentes valores para el uso de esta herramienta, tales como son que tipo de ofuscación queremos utilizar si solo sintáctica o semántica, si queremos que se pueda tracear la salida del archivo, etc. Para ampliar conocimientos sobre dicha herramienta podemos visitar esta web, <http://developer.android.com/tools/help/proguard.html> donde está toda la información necesaria.

- **project.properties:** es un archivo donde se pueden configurar diferentes parámetros del proyecto, como puede ser la API sobre la que se va a ejecutar el proyecto, o si queremos usar una herramienta que ofrece Google dentro del SDK para ofuscar el código llamada ProGuard.

4.3. Proyecto firma digital UMA.

La aplicación Firma digital UMA es la aplicación móvil que hemos realizado para facilitar la visualización y firma de nuevos documentos. En ella podemos comprobar los documentos anteriormente firmados o firmar nuevos de una forma fácil, el resto de gestión se tiene que realizar desde la aplicación web, donde se puede verificar, generar nuevos documentos, etc. En la figura 4.11 podemos ver el aspecto de la aplicación.

4.3.1. Uso de la aplicación.

Lo primero que pensamos cuando nos pusimos a diseñar la aplicación era que el método de firma tiene que ser rápido y fácil, por lo que se decidió usar los códigos QR para agilizar la lectura de información. Solo hay que pulsar el botón de añadir nuevo recibo (figura 4.12) y se abrirá el lector de códigos QR, una vez leídos el código este se firmará y se subirá automáticamente al servidor sin necesitar que el usuario realice otra acción.

La aplicación se puede dividir principalmente en dos partes claramente diferenciadas, una en la que se muestra las firmas realizadas y otra en la que se muestran que tienen como destino el usuario que está ejecutando la aplicación. Las dos partes son prácticamente idénticas y tienen el mismo uso. Si pulsamos encima de cualquier recibo de la lista nos mostrará toda la información de dicha firma, podemos ver un ejemplo en la figura 4.13. Podemos observar todos los datos necesarios, como pueden ser destino, quien la ha mandado, la fecha, el texto, etc y también podemos ver si está verificada con el tick de color verde o una señal de error roja si no está verificada, que podemos ver en la esquina superior derecha. En el caso de los recibos de los cuales se es recipiente se pone de una forma más clara quien es el que usuario que envía el mensaje pero el resto de la información es la misma. En la información que nos proporcionan podemos ver la dirección del servidor de tiempo

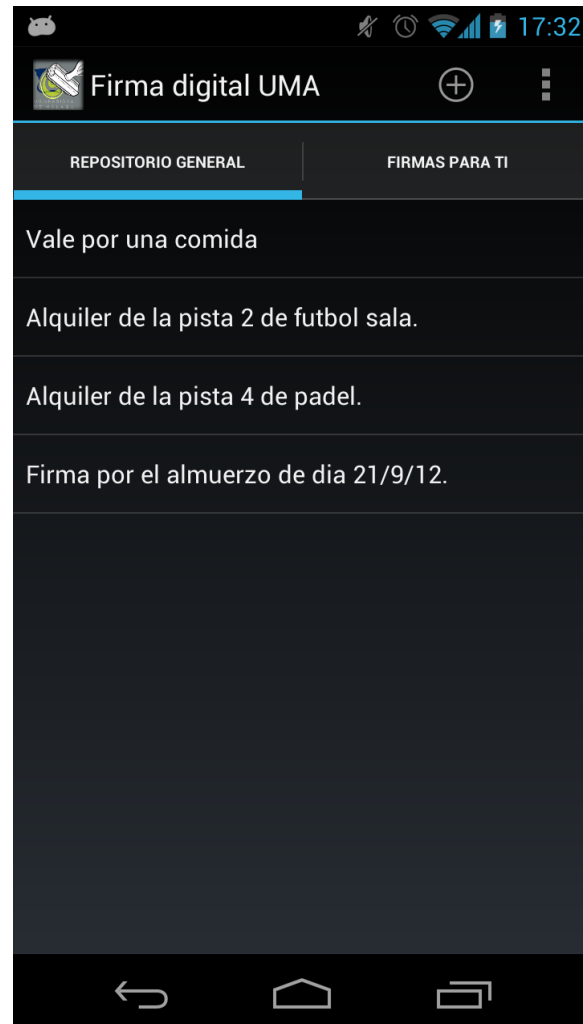


Figura 4.11: Pantalla inicial de la aplicación.



Figura 4.12: Detalle del botón añadir.

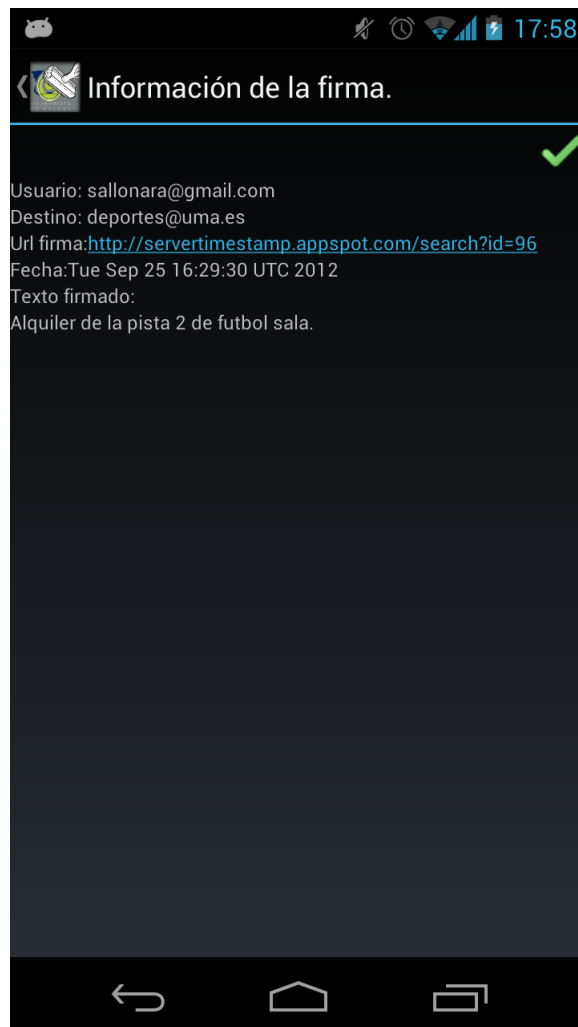


Figura 4.13: Información de una firma realizada.

usado y si pulsamos en ella se abrirá el navegador con una dirección donde podremos confirmar el hash y la fecha en la que se realizó la firma.

4.3.2. Explicación del proyecto de Android.

En la figura 4.14 podemos ver el proyecto de eclipse con todas las clases que hemos tenido que desarrollar, desde la conexión a la base de datos, el inicio de la aplicación, el cifrado, etc. La estructura del proyecto es la misma que ya hemos explicado anteriormente.

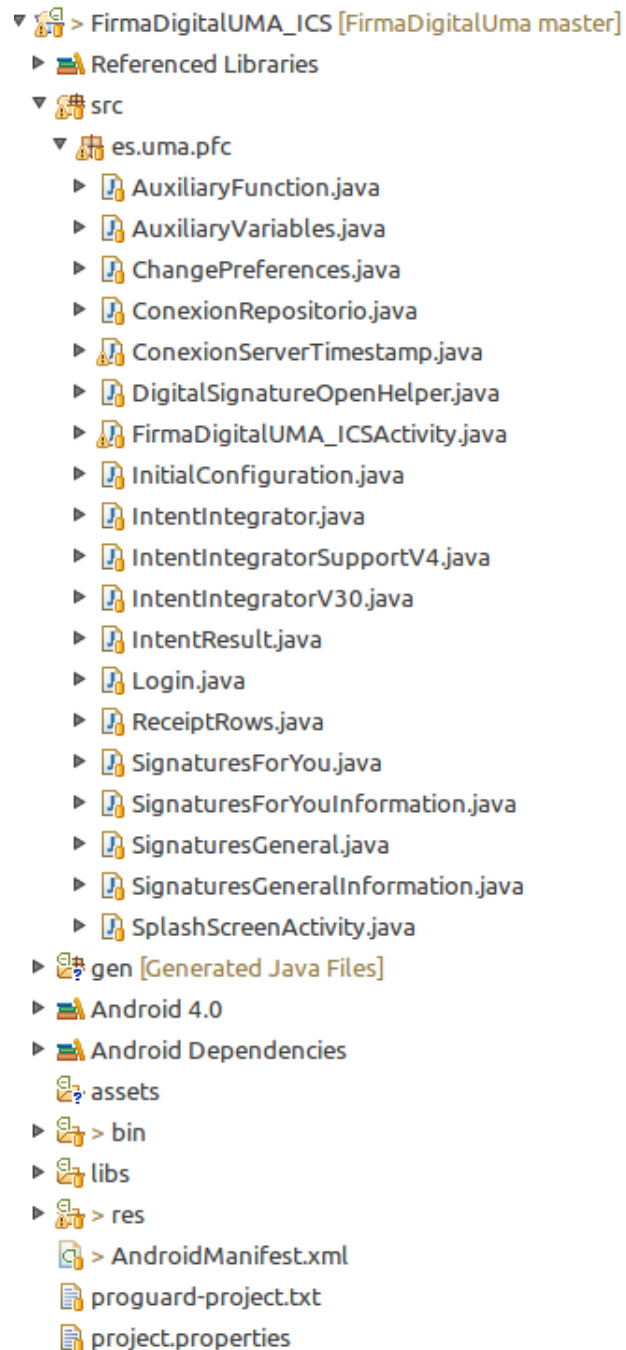


Figura 4.14: Proyecto de android en Eclipse.

A continuación vamos a explicar cada una de las clases y la función que tiene dentro del proyecto.

- **AuxiliaryFunction.java:** en esta clase se han implementado todas las funciones que son usadas muchas veces en el proyecto, como puede ser la función que verifica si un certificado existe en un path determinado y si se puede abrir, también está la función que pasa de un array de bytes a una cadena de texto.
- **AuxiliaryVariables.java:** aquí se almacenan todas las variables comunes que vamos a necesitar para que la aplicación funcione correctamente. En el siguiente trozo de código podemos ver que son variables como el login con el cual se está logueado, la cookie para autenticación en el servicio, el key store con las claves para realizar las firmas, si la aplicación está sin internet para mostrar la base de datos que tengamos almacenados y no intentar bajar recibos nuevos y otras.

```
1 private static Login LOGIN_GLOBAL;  
2 private static Cookie COOKIE_GLOBAL;  
3 private static KeyStore KEYSTORE_GLOBAL;  
4 private static boolean WITHOUT_INTERNET;  
5 private static List<ReceiptRows> LIST_RECEIPT;  
6 private static String THEME;
```

Además de las variables tenemos todos los getter and setter de las variables.

- **ChangePreferences.java:** esta clase es la encargada de realizar todo el manejo del cambio de preferencias si se produce. En la aplicación hemos usado una clase que proporciona android para el almacenado de configuración llamada SharedPreferences, al usar este método android proporciona listener para que cuando cambien se activen, de esta forma podemos controlar todos los cambios que realicemos actualizando en el acto la interfaz y guardando los datos en el acto. Un trozo de la clase en el que se puede observar como se crea y se implementa un listener para cuando se produzca un cambio en la configuración podamos actualizar tanto las shared preferences como la interfaz.

```
1 EditTextPreference textPath = (  
    EditTextPreference) findPreference(  
        "textPath");
```



```

    getActivity().getResources().getString(R.
    string.key_cert_path));
2
3 // Para que cuando cambie el texto lo cambie
  tambi n en el t tulo...
4 textPath.setOnPreferenceChangeListener(new
  OnPreferenceChangeListener() {
5   public boolean onPreferenceChange(Preference
    preference, Object newValue) {
6     EditTextPreference textPath = (
      EditTextPreference) preference;
7     String s = (String) newValue;
8
9     String pass = settings.getString("
      certificate_password", "NotValue");
10    if (!pass.equals("NotValue")) {
11      boolean check = AuxiliaryFunction.
        checkCert(s, pass);
12
13      if (check) {
14        textPath.setSummary(s);
15        textPath.setText(s);
16        Editor editor = settings.edit();
17        editor.putString("path_certificate", s);
18        editor.commit();
19        return true;
20      } else {
21        textPath.setSummary("No se ha podido
          cargar el certificado,\nla ruta no es
          correcta");
22        return true;
23      }
24    } else {
25      textPath.setSummary("Password no guardado"
        );
26      return true;
27    }
28  }
29 }
30
31 });
```

Creamos la variable EditTextPreference donde mostraremos el

password del certificado que estamos usando para firmar el texto. A continuación implementamos el listener `OnPreferenceChangeListener`, el cual se activará cuando digamos de cambiar el password del certificado. Podemos observar que cuando introducimos un nuevo password hacemos una comprobación para ver si después del cambio podemos seguir usando dicho certificado, si no es posible informamos al usuario diciendole que no se ha guardado el password.

- **ConexionRepositorio.java:** en esta clase se ha implementado todo lo referente a interactuar con la aplicación web repositorio general. En dicha clase se ha implementado el método para añadir un recibo nuevo, listar todos los recibos que hay guardados, etc. Todos los métodos son estáticos para no tener que crear instancias de esta clase, además se vio que teníamos que crear una nueva conexión cada vez que queríamos añadir un nuevo nuevo recibo, por lo que no tendría sentido crea instancias de la clase. A continuación podemos ver el método que añade un nuevo recibo.

```

1  public static String addRow(String plainText,
2      String tokenTime, Account account, String to)
3      {
4      /*
5       * Primero se le codifican los espacios porque
6       * si no la funci n
7       * URLEncoder.encode los cambia por '+' y si
8       * el documento tiene '+'
9       * luego los pone como si fueran espacios
10      */
11      String textoSinEspacios = plainText.replace("
12          ", "%20");
13      textoSinEspacios = URLEncoder.encode(
14          textoSinEspacios);
15      String token = tokenTime.split(";");[1];
16      String fecha = tokenTime.split(";");[2];
17      String url = cadServer + cadAdd +
18          cadServerTimestampSearch + token + "&texto="
19          + textoSinEspacios + "&token=" + token +
20          "&usuario=" + account.name
21          + "&destino=" + to + "&fecha=" + fecha;
22      String response = "";
23      try {
24          URL u1 = new URL(url);

```

```
16
17     // Si queremos usar el proxy inicializarlo
    de esta forma, donde
18     // sa es: SocketAddress sa = new
19     // InetSocketAddress("proxy.alu.uma.es",
    3128);
20     // HttpURLConnection c = (HttpURLConnection)
21     // u.openConnection(new Proxy(Proxy.Type.
    HTTP, sa));
22
23     Cookie cookie = AuxiliaryVariables.
    getCOOKIE_GLOBAL();
24
25     HttpURLConnection con1 = (HttpURLConnection)
    u1.openConnection();
26
27     con1.addRequestProperty("Cookie", cookie.
    getName() + "=" + cookie.getValue());
28     con1.setRequestMethod("GET");
29     con1.connect();
30
31     DataInputStream is1 = new DataInputStream(
    con1.getInputStream());
32     response = is1.readLine();
33
34     con1.disconnect();
35
36     } catch (MalformedURLException e) {
37         response = "MalformedURLException";
38         e.printStackTrace();
39     } catch (ProtocolException e) {
40         response = "ProtocolException";
41         e.printStackTrace();
42     } catch (IOException e) {
43         response = "IOException";
44         e.printStackTrace();
45     }
46     return response;
47 }
```

La función necesita una serie de parámetros para poder ser invocada, estos parámetros son los datos que necesitamos almacenar, como pueden ser el texto en claro, el token de tiempo que ha de-

vuelto la aplicación web de timestamp, la cuenta con la que se está usando la aplicación y el destino, el resto de valores o los añade la aplicación web o la aplicación del móvil. A continuación se realiza la conexión con el servidor y se recibe una cadena donde se indica si se ha realizado bien la operación.

- **ConexionServerTimestamp.java:** al igual que la clase anterior en esta se ha realizado todo lo relacionado con la aplicación web del servidor timestamp. Solo tiene una función que es la de añadir al servidor que tiene un aspecto similar a la mostrada en el apartado anterior.
- **DigitalSignatureOpenHelper.java:** esta clase es la encargada de la creación de la base de datos SQLite, cuando vamos a necesitar tener acceso a ella hay que generar un objeto de dicha clase y llamar al constructor, que devuelve un objeto con el que podremos insertar, buscar o borrar elementos de la base de datos. Al pensar en el diseño de la aplicación se creyó necesario el uso de la base de datos para almacenar todos los recibos que se han realizado hasta la fecha, la aplicación sabe cual es el último recibo que tiene almacenado y solo pide a la aplicación web que le de los recibos nuevos, de esta forma de ahorra en tiempo de inicio de la aplicación y además en la cantidad de datos móviles que usamos. Esta clase tiene que heredar de SQLiteOpenHelper que es la clase genérica que proporciona Android para manejo de la base de datos SQLite que tiene. Hay que implementar el constructor y dos métodos más obligatoriamente `public void onCreate(SQLiteDatabase db);` y `public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion);`, el primero es usado para crear la base de datos en la primera ejecución y el segundo para actualizar la base de datos cuando sea necesario.

La base de datos se crea que con la siguiente sentencia SQL, podemos verla en el siguiente listado.

```
1 static final String KEY_SEQ_NUM = "NUM_SEC";
2 static final String KEY_SIGN_URL = "URL_FIRMA";
3 static final String KEY_PLAIN_TEXT = "
  TEXTO_CLARO";
4 static final String KEY_TIME_TOKEN = "
  TOKEN_TIEMPO";
5 static final String KEY_USER = "USUARIO";
6 static final String KEY_VERIFY = "VERIFICADO";
```

```

7  static final String KEY_DESTINY = "DESTINO";
8  static final String KEY_DATE = "FECHA";
9
10 private static final String
    DIGITALSIGNATURE_TABLE_CREATE = "CREATE TABLE
    " + DIGITALSIGNATURE_TABLE_NAME + " (" +
11 BaseColumns._ID + " INTEGER PRIMARY KEY
    AUTOINCREMENT," +
12 KEY_SEQ_NUM + " TEXT, " +
13 KEY_SIGN_URL + " TEXT, " +
14 KEY_PLAIN_TEXT + " TEXT, " +
15 KEY_TIME_TOKEN + " TEXT, " +
16 KEY_USER + " TEXT, " +
17 KEY_VERIFY + " TEXT, " +
18 KEY_DESTINY + " TEXT, " +
19 KEY_DATE + " TEXT );";

```

- **FirmaDigitalUMA_ICSActivity.java:** esta es la clase principal del proyecto, en ella se crea la activity principal de la aplicación. Se puede ver en el código que no extiende a la clase `Activity`, si no que lo hace de `FragmentActivity`, esto es así porque como ya hemos explicado anteriormente se han usado fragment para el diseño de la aplicación y no solo activitys. Además de esto se ha añadido funcionalidad mediante el uso de `ViewPager` y `TabsAdapter`, el primero para el uso del scroll lateral y el segundo para tener las dos pestañas de los recibos que el usuario a firmado y de los que es recipiente. En el siguiente trozo de código podemos ver la creación de los dos elementos que hemos dicho antes, junto con la action bar.

```

1  mViewPager = new ViewPager(this);
2  mViewPager.setId(R.id.pager);
3  setContentView(mViewPager);
4
5  final ActionBar bar = getActionBar();
6  bar.setTitle("Firma digital UMA");
7  bar.setNavigationMode(ActionBar.
    NAVIGATION_MODE_TABS);
8
9  mTabsAdapter = new TabsAdapter(this, mViewPager)
    ;
10 mTabsAdapter.addTab(bar.newTab().setText("

```

```

        Repositorio general"), SignaturesGeneral.
        class, null));
11 mTabsAdapter.addTab(bar.newTab().setText("Firmas
        para ti"), SignaturesForYou.class, null);

```

Además de la creación de los elementos anteriores esta clase también es la encargada de crear los menús, que en la versión 4.0 de android tienden a desaparecer, debido a la desaparición del botón menú de los nuevos terminales, pero que en las nuevas versión se añade a la action bar. A continuación podemos ver como se crean y añaden los botones de añadir un nuevo recibo, el de configuración o el de salir de la aplicación.

```

1  @Override
2  public boolean onCreateOptionsMenu(Menu menu) {
3      MenuInflater inflater = getMenuInflater();
4      inflater.inflate(R.menu.main, menu);
5      return true;
6  }
7  @Override
8  public boolean onOptionsItemSelected(MenuItem
9      item) {
10     switch (item.getItemId()) {
11     case R.id.menuitem_add:
12         Log.d(AuxiliaryVariables.TAG_DEBUG, "Add");
13         IntentIntegrator intentQR = new
14             IntentIntegrator(this);
15         intentQR.initiateScan();
16         return true;
17     case R.id.menuitem_quit:
18         Log.d(AuxiliaryVariables.TAG_DEBUG, "Quit");
19         finish();
20         return true;
21     case R.id.menuitem_about:
22         Log.d(AuxiliaryVariables.TAG_DEBUG, "About")
23             ;
24         Toast.makeText(context, "about", Toast.
25             LENGTH_SHORT).show();
26         return true;
27     case R.id.menuitem_settings:
28         Log.d(AuxiliaryVariables.TAG_DEBUG, "

```

```

        Settings");
27     Toast.makeText(context, "ajustes", Toast.
        LENGTH_SHORT).show();
28     Intent prefsIntent = new Intent(
        getApplicationContext(),
29         ChangePreferences.class);
30     startActivity(prefsIntent);
31     return true;
32 }
33 return false;
34 }

```

Además de la creación de la activity y de los menús es la encargada de llamar a la función de subir un recibo a la aplicación web después de recibir por medio de un intent el resultado de la lectura del código QR, se puede observar en este trozo de código.

```

1  public void onActivityResult(int requestCode,
    int resultCode, Intent intent) {
2      IntentResult scanResult = IntentIntegrator.
        parseActivityResult(requestCode, resultCode
        , intent);
3      String plaintext = "";
4      if ((plaintext = scanResult.getContents()) !=
        null) {
5          progressDialog = new ProgressDialog(activity
        );
6          progressDialog.setMessage("Subiendo el
        recibo...");
7          progressDialog.show();
8
9          ConexionRepositorio.addRow(handler,
        plaintext);
10
11     }
12 }

```

- **InitialConfiguration.java:** esta clase es la activity que se encarga de realizar la configuración de la aplicación cuando es la primera ejecución en un teléfono, en ella se tiene que configurar el certificado y su password y la cuenta. Es la encargada de crear las shared preferences para que la aplicación funcione correctamente.

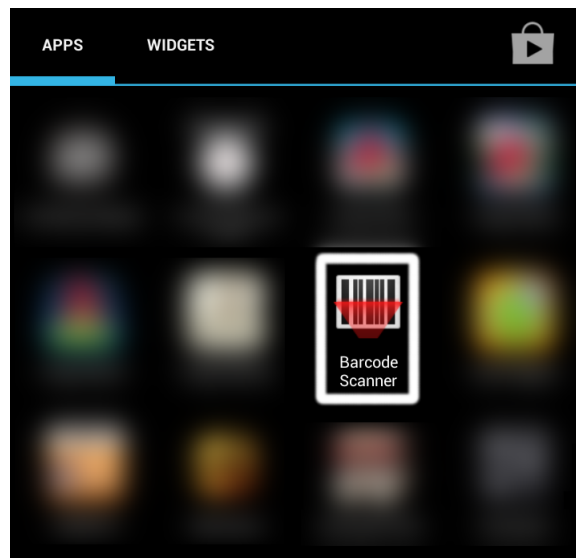


Figura 4.15: Aplicación Barcode Scanner.

- **InitialConfiguration.java, IntentIntegratorSupportV4.java, IntentIntegratorV30.java, IntentResult.java:** estas clases son las encargadas de la lectura de los códigos QR. Están hechas por ZXing y se distribuyen bajo licencia Apache License, Version 2.0. Proporcionan mediante un intent la posibilidad de lectura de códigos QR con su aplicación Barcode Scanner (figura 4.15) gratuita en Google Play, si no se tiene la aplicación instalada proporciona los métodos para hacerlo.
- **Login.java:** esta clase es la encargada de realizar el login en la aplicación web Repositorio General. Google proporciona un método de login si se tienen las credenciales, cosa que tenemos gracias a que en Android se necesita tener una cuenta de Google para poder usar Google Play y los diferentes servicios que ofrece. La dirección a la que tenemos que dirigirnos es a: https://repositoriorecibos.appspot.com/_ah/login. Esta clase tiene una variable privada que es el token de autenticación que conseguimos mediante la realización del login, para conseguirlo mostramos todas las cuentas de Google que hay configuradas en el terminal Android, cuando el usuario selecciona una de ellas iniciamos la conexión con el servidor y recibimos el token. Esto se hace todo el segundo plano para que la aplicación no se quede parada mientras se va a la página web y se consigue el token. En el código que sigue podemos ver el

proceso de obtención del token.

```

1 Thread t = new Thread() {
2     public void run() {
3         try {
4             AccountManagerFuture<Bundle> future =
3             manager.getAuthToken(account, "ah",
4             null, activity, null, null);
5             Bundle bundle;
6             bundle = future.getResult();
7             token = bundle.getString(AccountManager.
8             KEY_AUTHTOKEN);
9         } catch (OperationCanceledException e) {
10            e1.printStackTrace();
11        } catch (AuthenticatorException e) {
12            e1.printStackTrace();
13        } catch (IOException e) {
14            e1.printStackTrace();
15        }
16    }
17 }.start();

```

La parte mas interesante del código es en la llamada `manager.getAuthToken(account, "ah", null, activity, null, null);` con esta función conseguimos la cookie donde está el token de acceso para posteriormente con la función `bundle.getString(AccountManager.KEY_AUTHTOKEN);` conseguirlo.

La función `public Cookie getAuthCookie(String authToken);` se puede ver en el siguiente trozo de código.

```

1 public Cookie getAuthCookie(String authToken)
2     throws ClientProtocolException, IOException {
3     DefaultHttpClient httpClient = new
4     DefaultHttpClient();
5     Cookie retObj = null;
6     String cookieUrl = gaeAppLoginUrl + "?continue
7     =" + URLEncoder.encode(gaeAppBaseUrl, "UTF
8     -8") + "&auth=" + URLEncoder.encode(
9     authToken, "UTF-8");
10
11     HttpGet httpget = new HttpGet(cookieUrl);
12     HttpResponse response = httpClient.execute(
13     httpget);

```

```

8      if (response.getStatusLine().getStatusCode()
9          == HttpURLConnection.HTTP_OK
10         || response.getStatusLine().getStatusCode()
11            == HttpURLConnection.HTTP_NO_CONTENT) {
12
13          if (httpClient.getCookieStore().getCookies()
14              .size() > 0) {
15              retObj = httpClient.getCookieStore().
16                  getCookies().get(0);
17          }
18      }
19
20      return retObj;
21  }

```

En ella podemos ver que hacemos una conexión a la dirección `http://repositoriorecibos.appspot.com/_ah/login?continue=http://repositoriorecibos.appspot.com&auth=token`, en la que se le indica el token de acceso y la url a la que tenemos que seguir cuando se realice el login. Una vez realizada la identificación devolvemos la cookie donde irá el token de acceso.

- **ReceiptRows.java:** esta clase es la que representa un recibo, con ella se pueden crear objetos para almacenar todos los valores que necesitamos para tener un recibo identificado, como pueden ser el texto, la url de la firma, la fecha, el destino, etc. Este es el tipo de objeto que se usa para recoger la información de la base de datos y para almacenarla. Durante toda la ejecución de la aplicación tendremos una lista de objetos de esta clase para tener acceso a los recibos. Es una clase simple con un constructor con todos los valores que se guardan como parámetros y todos los getter y setter de las diferentes variables. También tiene definidas todas las constantes para el nombre de la tabla de la base de datos.
- **SignaturesGeneral.java:** esta clase extiende a `ListFragment`, es la lista de los recibos que el usuario a firmado. Es una de las dos pantallas principales de la aplicación, se puede ver en la figura 4.16. Cuando se genera la interfaz se hace una consulta a la base de datos y se recibe un objeto de la clase `Cursor`, en el cual se puede iterar para conseguir todos los recibos que has firmado. En el siguiente trozo de código podemos ver como se hace.

```

1  DigitalSignatureOpenHelper
    digitalSignatureOpenHelper = new
    DigitalSignatureOpenHelper(activity);
2  SQLiteDatabase sqLiteDatabase =
    digitalSignatureOpenHelper.
    getReadableDatabase();
3  Cursor cursor = sqLiteDatabase.query(
    DigitalSignatureOpenHelper.
    DIGITALSIGNATURE_TABLE_NAME,
4      new String[] { DigitalSignatureOpenHelper.
        KEY_PLAIN_TEXT }, null, null, null,
        null, null);

```

- **SignaturesGeneralInformation.java:** esta clase es la encargada de mostrar la información cuando pulsamos en algún recibo que hemos firmado. Es una clase que extiende de la clase `Activity` y es la encargada de cargar la interfaz modelada en el XML `signaturesgeneralinfo.xml` y mostrar toda la información del recibo que ha sido pulsado de la lista de recibos. Cuando se pulsa un recibo en la pantalla anterior que es la generada por `SignaturesGeneral.java`, antes de cargar esta se le manda la posición que ha sido pulsada para después poder localizarla y acto seguido se muestra la información, esto se hace por medio de un `Intent` al cual se le añade un valor entero con la posición. En el siguiente trozo de código podemos ver se recupera el valor, se realiza la consulta en la base de datos y se rellenan todos los campos donde mostraremos la información del recibo seleccionado.

```

1  super.onCreate(savedInstanceState);
2  setContentView(R.layout.signaturesgeneralinfo);
3
4  Intent intent = this.getIntent();
5  int pos = intent.getExtras().getInt("position");
6
7  DigitalSignatureOpenHelper
    digitalSignatureOpenHelper = new
    DigitalSignatureOpenHelper(this);
8  SQLiteDatabase sqLiteDatabase =
    digitalSignatureOpenHelper.
    getReadableDatabase();

```

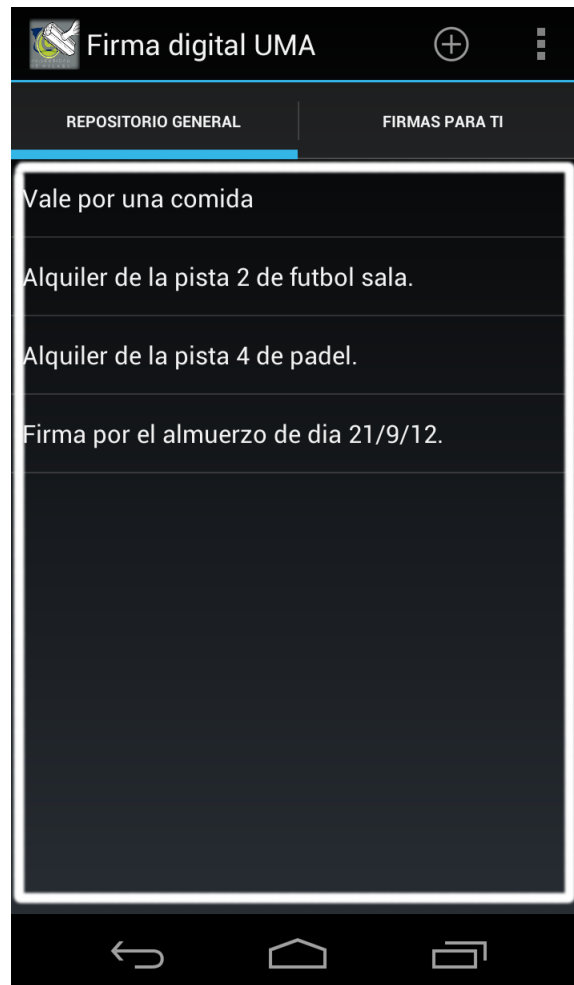


Figura 4.16: Detalle fragment generado por la clase SignaturesGeneral.java.

```
9  Cursor cursor = sqLiteDatabase.query(  
    DigitalSignatureOpenHelper.  
    DIGITALSIGNATURE_TABLE_NAME, new String[] {  
    DigitalSignatureOpenHelper.KEY_USER,  
10     DigitalSignatureOpenHelper.KEY_DESTINY,  
        DigitalSignatureOpenHelper.KEY_SIGN_URL,  
        DigitalSignatureOpenHelper.KEY_PLAIN_TEXT  
    },  
11     DigitalSignatureOpenHelper.KEY_VERIFY,  
        DigitalSignatureOpenHelper.KEY_DATE },  
        null, null, null, null, null);  
12  
13  if (cursor.moveToPosition(pos)) {  
14  
15     final ActionBar bar = getActionBar();  
16     bar.setTitle("Informaci n de la firma.");  
17     bar.setDisplayHomeAsUpEnabled(true);  
18  
19     TextView textUser = (TextView) findViewById(R.  
        id.text_sign_general_user);  
20     TextView textDestiny = (TextView) findViewById  
        (R.id.text_sign_general_to);  
21     TextView textUrl = (TextView) findViewById(R.  
        id.text_sign_general_url);  
22     TextView textPlainText = (TextView)  
        findViewById(R.id.  
            text_sign_general_plain_text);  
23     ImageView imageVerify = (ImageView)  
        findViewById(R.id.image_sign_general_verify  
            );  
24     TextView textDate = (TextView) findViewById(R.  
        id.text_sign_general_date);  
25  
26     textUser.setText(textUser.getText() + " " +  
        cursor.getString(0));  
27     textDestiny.setText(textDestiny.getText() + "  
        " + cursor.getString(1));  
28     textUrl.setText(cursor.getString(2));  
29     textPlainText.setText(cursor.getString(3));  
30  
31     String verify = cursor.getString(4);  
32     if (verify.equals("true")) {  
33         imageVerify.setImageResource(R.drawable.ok);
```

```

34     } else {
35         imageVerify.setImageResource(R.drawable.
            cancel);
36     }
37     textDate.setText(cursor.getString(5));
38
39     SQLiteDatabase.close();
40 }

```

Podemos ver como se genera el objeto `DigitalSignatureOpenHelper` y se realiza la consulta con el método `public Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)`; a continuación generamos todos los `TextView` necesarios para mostrar la información y los rellenamos con los datos.

- **SignaturesForYou.java:** igual que su clase equivalente `SignaturesGeneral.java` es la encargada de mostrar una lista con todos los recibos de los que se es destinatario, la estructura es practicamente idéntica, con solo una diferencia y es que se realiza un filtrado para que el campo destino sea el mismo que la cuenta de usuario usada en la aplicación. Podemos ver el resultado en la figura 4.17.
- **SignaturesForYouInformation.java:** al igual que la clase `SignaturesGeneralInformation.java` es la encargada de mostrar la información de los recibos que van dirigidos a tí. El proceso es igual que en `SignaturesGeneralInformation.java`, se envia la posición que ha sido marcada y se busca dicho recibo y se muestra toda la información.
- **SplashScreenActivity.java:** esta clase es la encargada de generar la primera activity que se muestra en la aplicación cuando se inicia, además de cargar todas las variables que serán usadas durante la ejecución. Esta clase crea una interfaz en la que solo muestra el logo de la aplicación y una barra progreso que muestra el proceso de carga, pero en background comprueba que no sea la primera ejecución, si lo es llama a la clase `InitialConfiguration.java` para generar la configuración, si no es la primera ejecución abre el archivo de preferencias compartidas y genera todos los objetos necesarios para la ejecución como pueden ser, la cuenta que utilizaremos posteriormente, el tema que estamos usando, hacemos el login en la aplicación web, el keystore que usaremos para firmar en la aplicación y a parte realizamos la conexión con

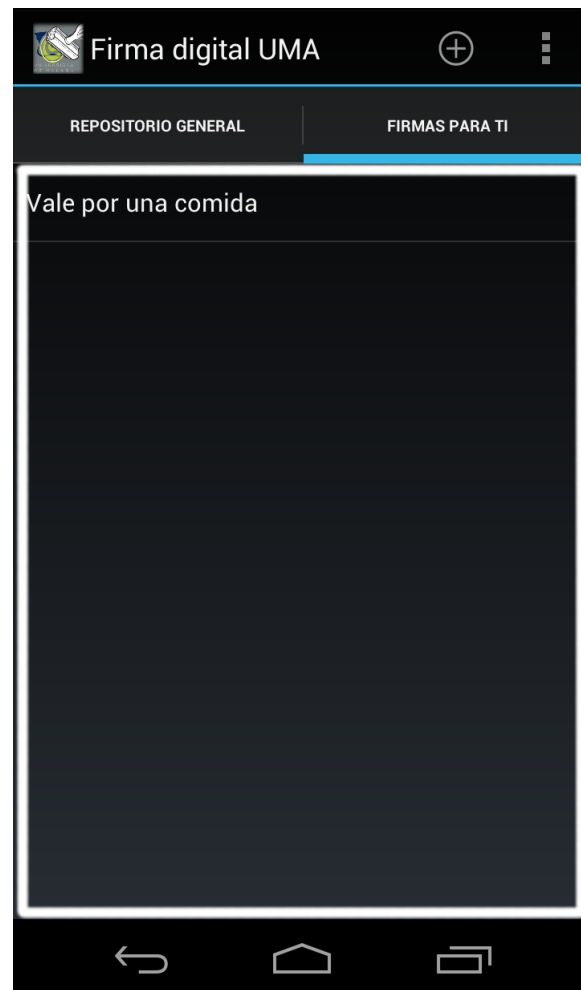


Figura 4.17: Detalle fragment generado por la clase SignaturesForYou.java.

la aplicación web Repositor General y conseguimos las filas que se hayan insertado nuevas añadiéndolas a la base de datos de la aplicación para que posteriormente podamos usarlas sin tener que pedir las al servidor, para ello se utiliza se tiene que parsear un objeto de tipo JSONArray, una vez realizadas todas esas acciones se procede a la inicialización de una nueva activity que genera la clase `FirmaDigitalUMA_ICSAActivity.java` y que dará lugar a la creación de la interfaz principal de la aplicación.

Capítulo 5

Google App Engine.

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

5.1. 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 autenticación con cuentas Google del tipo: *usuario@gmail.com*, autenticación 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 información 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 desplegar. 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 profundidad la tecnología usada y ambas aplicaciones web.

5.2. 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.

5.2.1. ¿Qué es un servlet?.

Un servlet es la evolución de los antiguos applet, su uso más común es generar páginas web dinámicamente 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.

5.2. EXPLICACIÓN DE UNA APLICACIÓN WEB GENÉRICA EN GOOGLE APP ENGINE.85

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 ampliación 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 encargados 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 de código se puede ver la implementación más básica de los métodos **doGet** y **doPost**, con las llamadas a sus respectivas llamadas a *super*.

```
1  @Override
2  protected void doGet(HttpServletRequest req,
3      HttpServletResponse resp) throws ServletException,
4      IOException {
5      // TODO Auto-generated method stub
6      super.doGet(req, resp);
7  }
8
9  @Override
10 protected void doPost(HttpServletRequest req,
11     HttpServletResponse resp) throws ServletException,
12     IOException {
13     // TODO Auto-generated method stub
14     super.doPost(req, resp);
15 }
```

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:

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

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

```
1 PrintWriter out = resp.getWriter();
2 out.print(jsonArray);
3 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 mandándole peticiones **GET** sería la siguiente: <https://servertimestamp.appspot.com/search?id=63&texto=Prueba>, como se puede ver la dirección base sería: <https://servertimestamp.appspot.com/>, el servlets estaría mapeado internamente en el servidor web, como ya veremos en próxima sección, en la dirección `/search` y el primer parámetro va precedido de `?id_parametro` y el resto de `&id_parametro`. 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.

5.2.2. ¿Qué es JSP?.

JSP es el acrónimo de JavaServer Pages y es una tecnología que ayuda a crear dinámicamente páginas web basadas en HTML o XML y es la solución equivalente a PHP de Oracle. En la figura 5.1 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:

```
1 <!DOCTYPE html>
```

5.2. EXPLICACIÓN DE UNA APLICACIÓN WEB GENÉRICA EN GOOGLE APP ENGINE.87

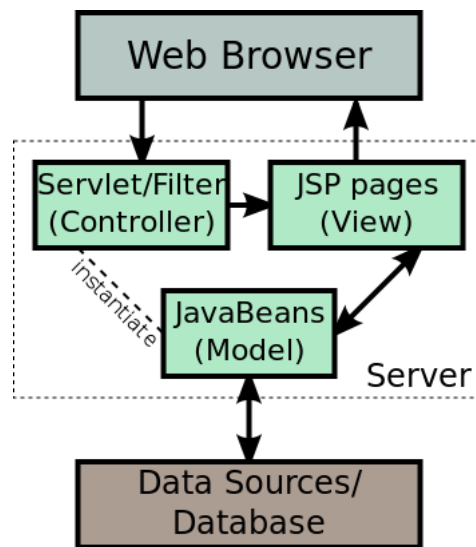


Figura 5.1: Modo de interpretación de un archivo JSP

```

2  <html>
3  <body>
4  <table>
5  <tr>
6    <th>ID</th>
7    <th>Num sec</th>
8    <th>Token de tiempo</th>
9    <th>Mensaje</th>
10   <th>URL para ver la firma</th>
11   <th>Fecha</th>
12   <th>Usuario</th>
13   <th id="filadestino">Destino</th>
14   <th>Verificado?</th>
15 </tr>
16
17 <% for (RowRepositorioGeneral row : rows) {%>
18 <tr>
19   <td><%=row.getId()%></td>
20   <td><%= row.getNum_sec()%></td>
21   <td><%=row.getToken_tiempo()%></td>
22   <td><%=row.getTexto_claro()%></td>
23   <td><a href=<%=row.getUrl_firma()%>>URL para ver
24     el token
25     de tiempo</a></td>
26   <td><%=row.getFecha()%></td>
  
```

```

26     <td><%=row.getUsuario()%></td>
27     <td id="filadestino"><%=row.getDestino()%></td>
28     <td>
29         <%
30         Boolean confirmado = row.getConfirmado();
31         if (!(confirmado == null) && confirmado) {
32             <%
33             <center>
34                 
35             </center> <%} else {<%
36             <center>
37                 
38             </center> <%} %>
39         </td>
40     </tr>
41     <%}%>
42 </table>
43 </body>
44 </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.

5.2.3. 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 5.2 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.yaml` que son archivos de configuración del servidor que se verán en el proximo apartado 5.2.4 y además los archivos `jsp` que se usan en la aplicación junto con los archivos `html` y `javascript` que se necesiten.

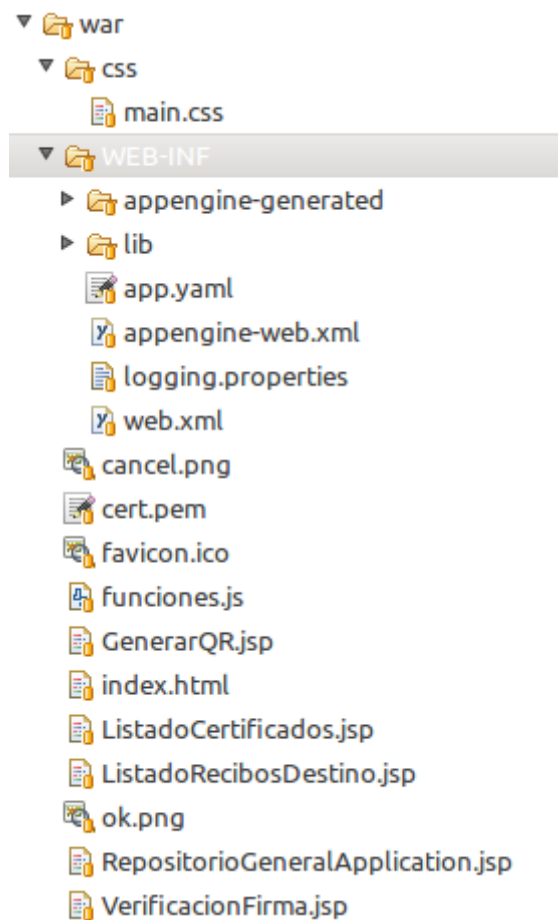


Figura 5.2: Carpeta WAR

5.2.4. Archivos de configuración.

Los principales archivos de configuración son *web.xml* y *app.yaml*, 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:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema
   -instance"
3  xmlns="http://java.sun.com/xml/ns/javaee"
4  xmlns:web="http://java.sun.com/xml/ns/javaee/web-
   app_2_5.xsd"
5  xsi:schemaLocation="http://java.sun.com/xml/ns/
   javaee
6  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   version="2.5">
7
8      <servlet>
9          <servlet-name>AddRow</servlet-name>
10         <servlet-class>pfc.ServletCreateRow</servlet-
            class>
11     </servlet>
12     <servlet-mapping>
13         <servlet-name>AddRow</servlet-name>
14         <url-pattern>/add</url-pattern>
15     </servlet-mapping>
16
17     <welcome-file-list>
18         <welcome-file>ServerTimestampApplication.jsp</
            welcome-file>
19     </welcome-file-list>
20 </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.yaml*:

```

1  application: repositoriorecibos

```



```
2 version: 1
3 runtime: java
4
5 handlers:
6   - url: /add
7     servlet: pfc.ServletCreateRow
8     secure: always
9 welcome_files:
10  - 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/java/config/>, <https://developers.google.com/appengine/docs/java/configyaml/>. En el primero podemos ver todas las opciones configurables de *web.xml* y en la segunda las de *app.yalm*.

5.3. 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 añadiéndole 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 5.3. 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 5.3: 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.

5.3.1. 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

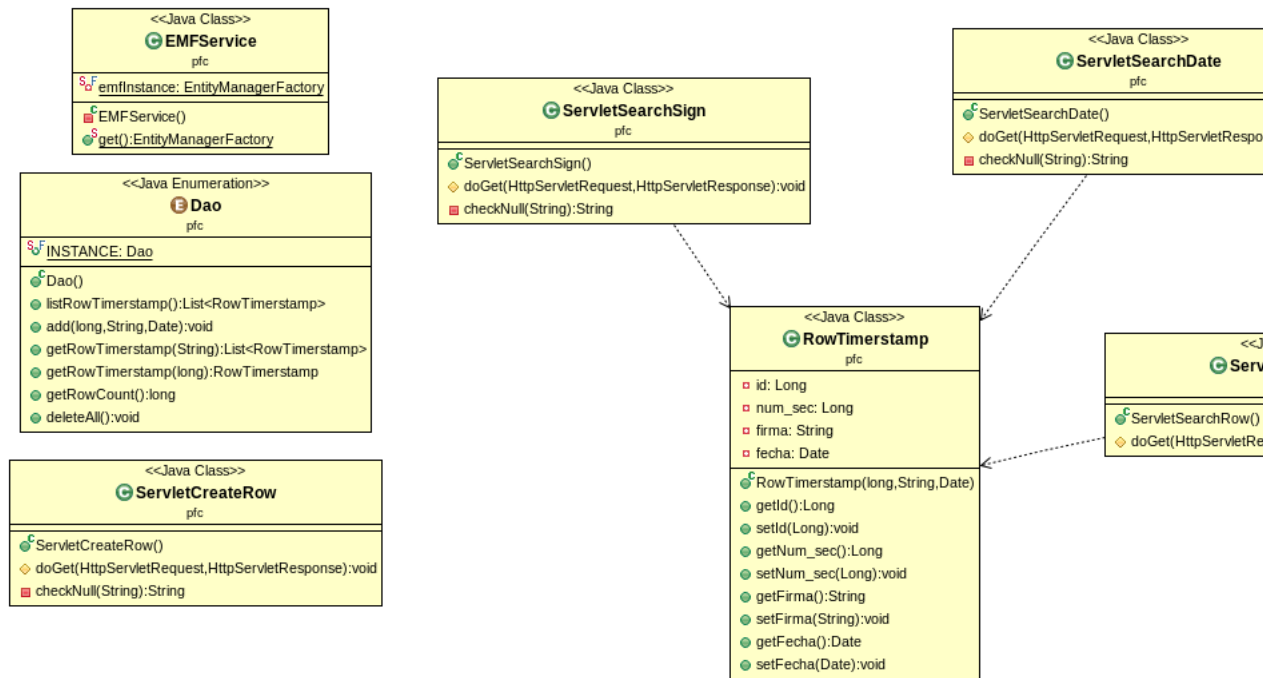


Figura 5.4: Detalles del paquete pfc

timestamp.

En la figura 5.4 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:

```

1 EntityManager em = EMFService.get().
  createEntityManager();
2 Query q = em.createQuery("select t from
  RowTimestamp t where t.num_sec = :num_sec");
3 q.setParameter("num_sec", id);
4 RowTimestamp RowTimestamps = (RowTimestamp)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:

```
1 EntityManager em = EMFService.get().
    createEntityManager();
2 RowTimestamp RowTimestamp = new RowTimestamp(
    num_sec, firma, fecha);
3 em.persist(RowTimestamp);
4 em.close();
```

RowTimestamp.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 primaria, también se puede indicar que ese campo es autoincrementado y otras opciones que habría que indicar en la creación de la tabla.

```
1 @Id
2 @GeneratedValue(strategy = GenerationType.
    SEQUENCE)
3 private Long id;
4 private Long num_sec;
5 private String firma;
6 private Date fecha;
```

Se puede ver que el campo *id* será la clave primaria 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 subió en la fecha que indica el campo *fecha*. El resto de métodos que tiene esta clase es un constructor, getter para consultar los campos y setter para insertar valores.

ServletCreateRow.java: Esta clase es un servlet que se encarga 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:

```
1 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 insercción 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 necesitamos.

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 información almacenada que corresponde con el número de secuencia que se le pasa en el parámetro **id**. La forma de hacerlo es la siguiente:

```
1 PrintWriter pw = resp.getWriter();
2 pw.print("<!DOCTYPE html>");
3 pw.print("<html><head><title>Lista Time Stamp</title><link rel=\"stylesheet\" type=\"text/css\" " +
4     "href=\"css/main.css\"/> <meta charset=\"utf-8\"> </head>");
5 pw.print("<body><table><tr><th>ID</th><th>Num sec</th><th>Firma</th><th>Date</th></tr><tr>" +
6     "<td>"+ row.getId() +"</td><td>"+ row.getNum_sec() +"</td><td>"+ row.getFirma()
7     +"</td><td>"+ row.getFecha() + "</td></tr> </table></body>"
8     ");
9 pw.flush();
```

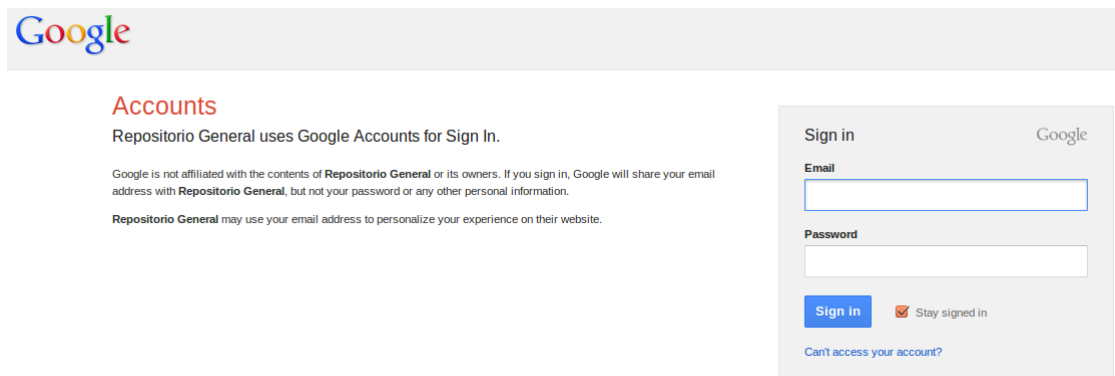


Figura 5.5: Login en Repositorio General

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**.

5.4. 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 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 5.5. 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 5.6.

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.

5.4.1. Explicación de la aplicación web.

En la figura 5.7 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 5.3.1.

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:

```

1  @Id
2  @GeneratedValue(strategy = GenerationType.
      SEQUENCE) //      GenerationType.IDENTITY
3  private Long id;
4  private Long num_sec;
5  private String url_firma;
6  private String texto_claro;
7  private Long token_tiempo;
8  private String usuario;
9  private Boolean confirmado;
10 private String destino;
11 private BlobKey blobKey;
12 private Date fecha;
```

Tiene una clave primaria 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 encargada 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

```
1  @Id
2  @GeneratedValue(strategy = GenerationType.
    SEQUENCE)
3  private Long id;
4  private String usuario;
5  private Date fecha;
6  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 5.8.

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 5.9

El servlet recibe los siguiente parámetros:

```
1 String mensaje = checkNull(req.getParameter("
    mensaje"));
2 String url_firma = checkNull(req.getParameter("
    token"));
3 String id_blob = checkNull(req.getParameter("
    id_blob"));
4 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: `;/:`.

```
1 String cadACodificar = mensaje + ";/:" +
    url_firma + ";/:" + id_blob + ";/:" + user;
```

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

```

1 String cadCodificada = Base64.encode(
    cadACodificar.getBytes("UTF-8"));

```

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

```

1 pw.println("BEGIN EXPORT");
2 pw.println("-----");
3 pw.println(cadCodificada);
4 pw.println("-----");
5 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:

```

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

```

Una vez se consigue el usuario se llama a la función *public List<RowRepositoryGeneral> getRowRepositoryGeneral(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 JSON¹.

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

```

1 JSONObject jsonObject = new JSONObject();
2
3 jsonObject.put("num_sec", rowRepositoryGeneral.
    getNum_sec().toString());
4 jsonObject.put("texto", rowRepositoryGeneral.
    getTexto_claro());

```

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

```

5  jsonObject.put("url_firma",
    rowRepositorioGeneral.getUrl_firma());
6  jsonObject.put("token_tiempo",
    rowRepositorioGeneral.getToken_tiempo().
    toString());
7  jsonObject.put("usuario", rowRepositorioGeneral.
    getUsuario());
8  jsonObject.put("fecha", rowRepositorioGeneral.
    getFecha().toString());
9  jsonObject.put("verificado",
    rowRepositorioGeneral.getConfirmado().
    toString());
10 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.

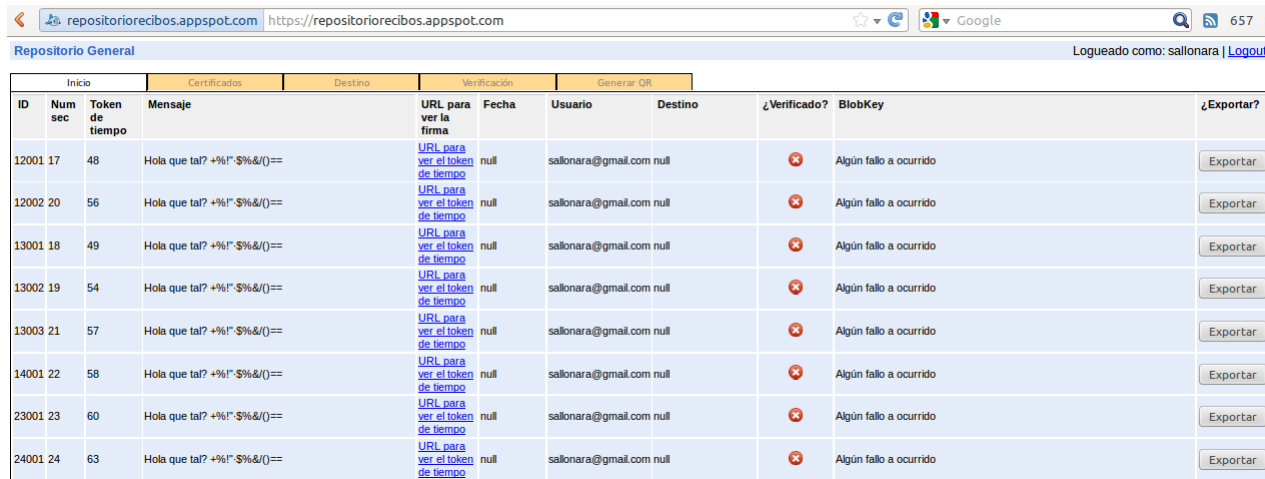
Servlet Verify.java: Este servlet es el utilizado en la pestaña verificar de nuestra aplicación web, como se puede observar en la figura

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, descriptamos 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 habíamos 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 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 5.11. Su función es generar un código QR para que pueda ser leído 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 5.12.



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	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar
12002	20	56	Hola que tal? +%!*-\$%&()/=	URL para ver el token de tiempo	null	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar
13001	18	49	Hola que tal? +%!*-\$%&()/=	URL para ver el token de tiempo	null	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar
13002	19	54	Hola que tal? +%!*-\$%&()/=	URL para ver el token de tiempo	null	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar
13003	21	57	Hola que tal? +%!*-\$%&()/=	URL para ver el token de tiempo	null	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar
14001	22	58	Hola que tal? +%!*-\$%&()/=	URL para ver el token de tiempo	null	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar
23001	23	60	Hola que tal? +%!*-\$%&()/=	URL para ver el token de tiempo	null	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar
24001	24	63	Hola que tal? +%!*-\$%&()/=	URL para ver el token de tiempo	null	salonara@gmail.com	null	✗	Algun fallo a ocurrido	Exportar

Figura 5.6: Repositorio General

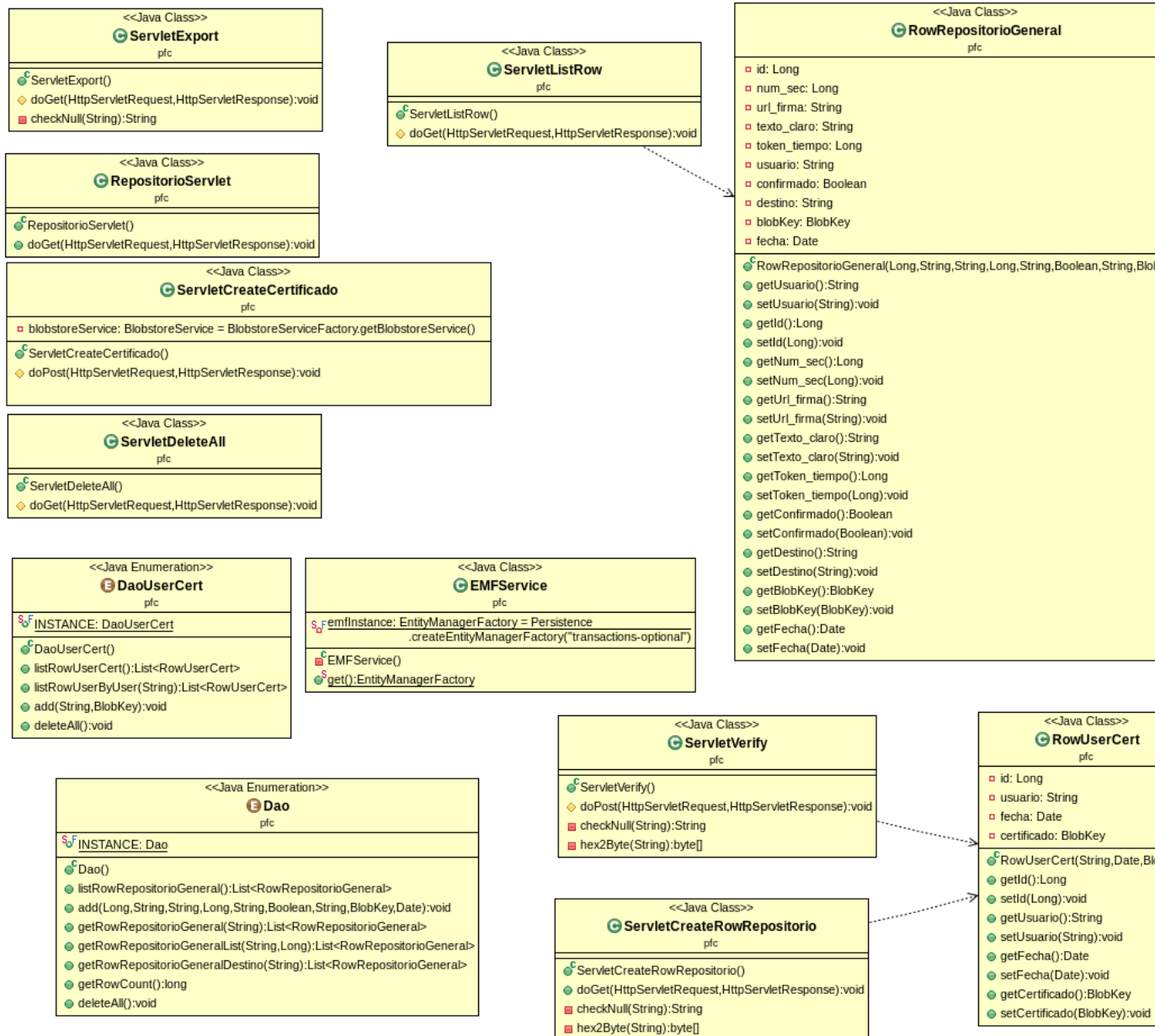


Figura 5.7: Detalles de las clases Repositorio General.

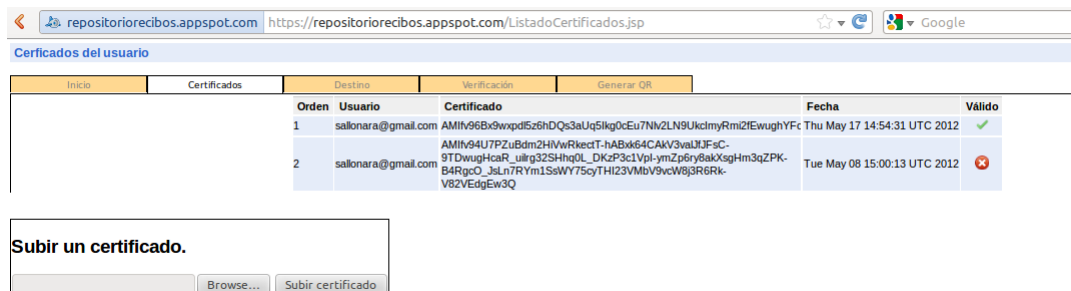


Figura 5.8: Pantallazo de la pestaña certificados.



Figura 5.9: Detalle del botón exportar.

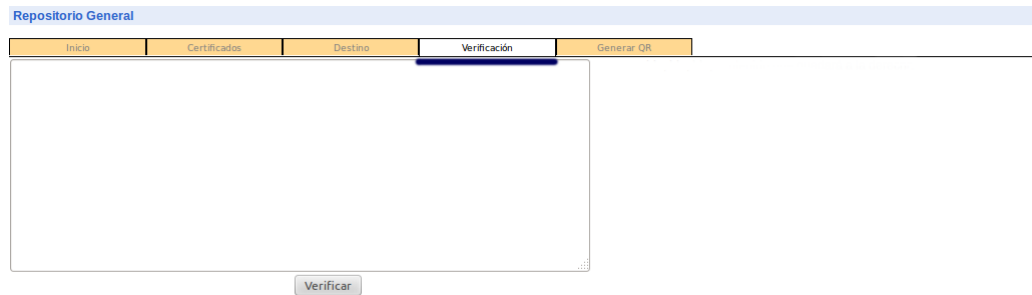


Figura 5.10: Detalle de la pestaña verificación.

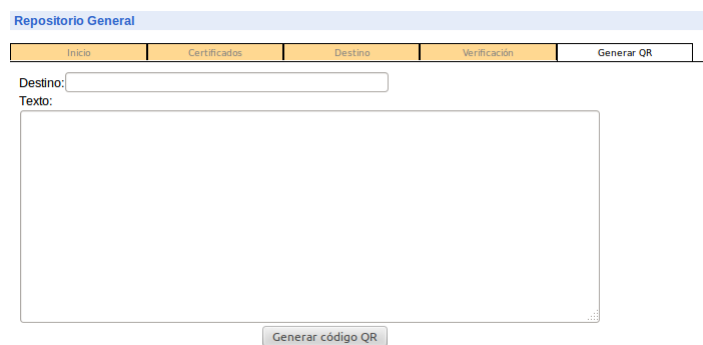


Figura 5.11: Pestaña para generar el código QR.

Repositorio General Logueado como: sallonara | [Logout](#)

Inicio	Certificados	Destino	Verificación	Generar QR
--------	--------------	---------	--------------	------------

Destino:

Texto:

Fírmame esto por favor.



Figura 5.12: Pestaña con el código QR generado.

Índice de figuras

1.1. Action Bar.	3
1.2. Gráfico de las versiones de android a finales de agosto del 2012.	4
1.3. Gráfico del uso de las versiones de android a finales de agosto del 2012.	4
1.4. Samsung Galaxy Nexus.	5
2.1. Índice tiobe en septiembre del 2012. http://www.tiobe.com/	8
2.2. Código ejemplo código for mejorado.	11
2.3. Eclipse 4.2 Juno.	12
2.4. Ejemplo Cifrado Cesar	15
2.5. Tabla para cifrado homofónico	15
2.6. Ejemplo de cifrado por sustitución	16
2.7. Iteraciones en el cifrado DES	18
2.8. Versión de Android 4.1	25
2.9. Código ejemplo XML.	28
3.1. Estructura genérica de una PKI.	35
3.2. Jerarquía de la PKI del DNIE.	36
3.3. Uso de la clase MessageDigest.	37
3.4. Ejemplo de un trozo de código fuente con el uso de MessageDigest.	38
3.5. Modo de uso de la clase Signature.	39

3.6. Ejemplo de un trozo de código fuente con el uso de la función <code>sign()</code>	39
3.7. Ejemplo de un trozo de código fuente con el uso de la función <code>verify()</code>	39
3.8. Modo de uso de la clase <code>Cipher</code>	40
3.9. Ejemplo de un trozo de código donde se carga un <code>KeyStore</code> . 41	
4.1. Arquitectura de Android.	45
4.2. Proceso de ejecución en Android.	47
4.3. Estructura del fichero APK del proyecto.	49
4.4. Opción compartir de una aplicación.	51
4.5. Widget de Gmail.	52
4.6. Ciclo de vida de una <code>activity</code>	55
4.7. Aplicación de Gmail para tablet.	57
4.8. Aplicación de Gmail para móvil.	58
4.9. Gesto Swype en la aplicación móvil.	59
4.10. Estructura básica de un proyecto Android.	60
4.11. Pantalla inicial de la aplicación.	63
4.12. Detalle del botón añadir.	63
4.13. Información de una firma realizada.	64
4.14. Proyecto de android en Eclipse.	65
4.15. Aplicación Barcode Scanner.	74
4.16. Detalle fragment generado por la clase <code>SignaturesGeneral.java</code>	78
4.17. Detalle fragment generado por la clase <code>SignaturesForYou.java</code>	81
5.1. Modo de interpretación de un archivo JSP	87
5.2. Carpeta WAR	89
5.3. Servidor Timestamp https://seguro.ips.es	92
5.4. Detalles del paquete pfc	94
5.5. Login en Repositorio General	97
5.6. Repositorio General	104

<i>ÍNDICE DE FIGURAS</i>	111
--------------------------	-----

5.7. Detalles de las clases Repositorio General.	105
5.8. Pantallazo de la pestaña certificados.	106
5.9. Detalle del botón exportar.	106
5.10. Detalle de la pestaña verificación.	106
5.11. Pestaña para generar el código QR.	106
5.12. Pestaña con el código QR generado.	107