Diseño e implementación de un sistema de autenticación en un servidor web y un cliente Android

Trabajo Final de Máster presentado por Gerard Gutiérrez Vidal para obtener el título de Máster Interuniversitario de Seguridad de las Tecnologías de la información y Comunicación

Directora: Ángela María García Valdés

2015

MISTIC

Agradecimientos

Quiero agradecer al Instituto Nacional de Ciberseguridad (**Incibe**, anteriormente Inteco), y en particular a la directora de este trabajo final de master **Ángela María García Valdés** por darme la oportunidad de realizar este trabajo final de Máster. Además, agradecer a mi familia que me han apoyado moralmente durante la realización de este proyecto.

Resumen

En este trabajo final de máster se explica el diseño y la implementación, tanto de lado de cliente como de lado de servidor de un sistema de autenticación basado en token de autenticación, utilizado para una aplicación Android que consiste en un gestor de checklist en la nube. El usuario se registra en la aplicación y tendrá la opción de crear listas privadas, personalizarlas con imágenes y completarlas con ítems que se podrán marcar como completado o desmarcarlo como tal para llevar un control de dicha lista. La utilidad de esta aplicación reside en que es posible crear listas de tareas to-do fácilmente desde cualquier dispositivo Android y en cualquier lugar.

Durante el desarrollo de este documento se detallan algunos aspectos tanto en el lado de servidor, como son: administración de sistemas, seguridad en redes, bases de datos, o seguridad de la autenticación en general, mientras que en el lado de cliente se explican algunos aspectos de seguridad activa que he utilizado como por ejemplo la autenticación a dos pasos mediante correo electrónico.

Índice general

Índice de figuras

Capítulo 1

Introducción

En la actualidad, a causa de la utilización de las tecnologías de la información y las comunicaciones (a partir de ahora TIC), se utilizan, se almacenan y se transmiten grandes conjuntos de datos, por ejemplo, los datos que permiten la identificación de los usuarios de un servicio basado en las TIC (como pueden ser el prestador de servicios de Internet o el administrador de un foro online, etc.).

La ley de protección de datos de carácter personal (LOPD 15/1999) tiene como objetivo proteger, respecto al tratamiento de datos personales, las libertades públicas, los derechos fundamentales, intimidad personal y familiar y el honor de las personas físicas. Es por eso que cualquier tipo de informacioón que permita la identificacioón de un individuo y/o información sensible (económica, social, médica, religiosa) debe protegerse utilizando diferentes medidas, pasivas o activas.

A continuación se enumeran algunas de estas medidas tecnológicas que se suelen llevar a cabo en el mundo de las TIC:

- Pasivas: Éstas técnicas tienen como objetivo minimizar todo lo posible los daños causados por un ataque en un sistema informático.

    - Cifrado: Esta técnica consiste en almacenar en forma codificada los datos sensibles, o cualquier otro tipo de dato, de manera que sólo es posible identificarlos si se posee la clave de descodificación. En caso de un ataque satisfactorio, el autor, no podría obtener los datos sensibles a los que se refiere el código obtenido.

  - Copias de seguridad periódicas (backups): En caso de un ataque satisfactorio y en el caso que algunos datos se hayan perdido, como administradores, siempre podremos restaurar una copia de seguridad y recuperar la información tal y como se encontraba en el momento en que se ha hecho la copia de seguridad.
  - Monitorización de la transferencia de datos: Si para el funcionamiento del sistema es necesaria una transferencia de datos (como puede ser el terminal online de una entidad bancaria, o simplemente una página web), puede ser importante monitorizar constantemente el movimiento de datos que existe en nuestro sistema, de esta forma podremos identificar un posible problema de seguridad y actuar al instante para solventarlo e incluso hasta evitarlo.
  - Análisis forense: Si nuestro sistema ya ha sufrido un ataque, podemos identificar los resultados de ése ataque para obtener información sobre el mismo. Un ejemplo podría ser la inspección de logs que van generando nuestros servicios.

- Activas: Estos métodos tienen como objetivo evitar posibles a ataques al sistema informático que queremos proteger.

  - Mantener actualizados todas las aplicaciones: Si utilizamos las últimas versiones de todas las aplicaciones que necesitamos en nuestro sistema, conseguimos mitigar las posibles vulnerabilidades que tengan en sus versiones anteriores. Un atacante puede aprovechar estas vulnerabilidades para llevar a cabo un ataque contra el sistema (exploit, explotar una vulnerabilidad). El inconveniente de tener siempre la última versión de una aplicación és que existen posibilidades de que existan nuevas vulnerabilidades, pero que no sean conocidas.
  - Filtrado de datos/servicios: Esta medida trata de evitar cualquier tráfico de datos que no tenga nada que ver con el servicio que se quiera oferecer en nuestro sistema. En el campo de la Telemática, se conoce como el filtraje de puertos de una red y trata de evitar el tráfico de datos en puertos (servicios) en los cuales no trabaja nuestro servidor.

- Test de penetración (pentesting): Como administradores del sistema podemos jugar el papel de atacante y probar de acceder a nuestro propio sistema de forma ilegtima, siempre tomando precauciones, como pueden ser copias de seguridad, por ejemplo. De esta forma podemos identificar cómo responde nuestro sistema a intentos de acceso y dependiendo de esa información, ajustar parámetros del servicio para así evitarlo por parte de terceros.
- Autenticación: Para acceder a los datos es necesario autenticarse, y además tener los privilegios necesarios para obtener dicha información. En este trabajo final de máster se hablará con más detalle sobre ésta técnica.

Capítulo 2

Especificaciones

Dado que no es útil diseñar un sistema de autenticación que no tenga como objetivo proteger algún tipo de información, he decidido llevar a cabo una aplicación que requiera de un sistema de autenticación para un funcionamiento correcto y seguro para la información que se está tratando.

La aplicación se trata en un gestor de **checklist**, y como cliente del sistema podremos crear listas que pueden ser de cualquier tipo (de la compra, de viajes, TO-DO...) y poder marcar como completado (o no) cada uno de los elementos (a partir de ahora ítems) que las componen. Toda la información acerca de las listas e ítems que las componen se almacenará en la nube (nuestro servidor).

2.1. Historias de usuario

Las historias de usuario (que normalmente se utilizan en el método de desarrollo ágil) permiten determinar el objetivo de la aplicación desde el punto de vista de cualquier usuario que la utilizará y a partir de ahí como desarrolladores tendremos listados los objetivos a los que queremos llegar.

1. Como usuario, es necesario estar registrado para poder utilizar la aplicación.

2. Durante el registro, como usuario, será necesario validar la dirección de correo electrónico como confirmación mediante un enlace que el servidor enviará al

mismo, de esta forma evitamos que usuarios se registren mediante cuentas de correo electrónico de terceras personas.

3. Toda la información se almacenará en el servidor (listas, ítems, sesiones, usuarios...)

4. Como usuario será posible registrarse, iniciar y cerrar sesión desde la aplicación.

5. Los usuarios se deben identificar mediante su correo eléctronico, pero también pueden escoger un nombre y una imagen para identificar más fácilmente su perfil. Como usuario, podremos modificar el nombre y la imágen de perfil en cualquier momento desde la pantalla de perfil de la aplicación.

6. Como usuario, podremos crear o eliminar nuestras propias listas.

7. Las listas pueden tener asociada una imagen, como usuario, será posible modificar la imágen desde el mismo cliente.

8. Las listas contienen elementos (ítems) y el usuario puede crear y eliminar los que pertenezcan a sus listas.

9. Los ítems tienen dos estados (completado y no completado). Como usuario, será posible modificar el estado de los ítems que pertenezcan a sus listas en cualquier momento.

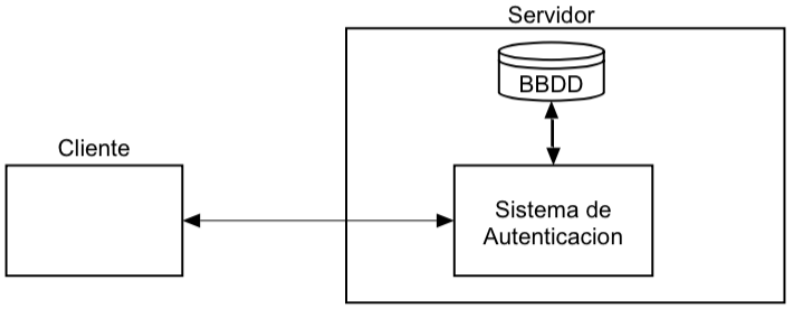


Figura 2.1: Diagrama del sistema.

Capítulo 3

Diseño

En el proceso de diseño se han tomado las decisiones sobre el sistema de autenticación que se utilizará, qué sistema utilizar para llevar a cabo las especificaciones enumeradas en el punto anterior, qué lenguajes de programación utilizar y confirmar la seguridad del sistema diseñado.

3.1. Sistema

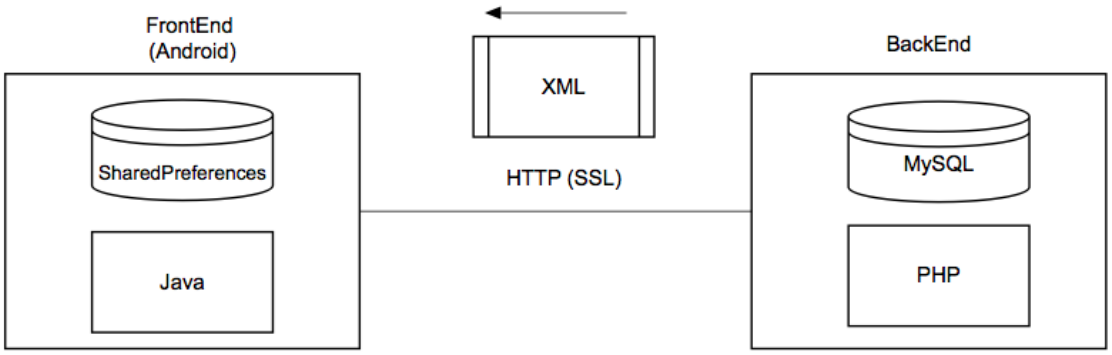


Figura 3.1: Diseño del sistema.

3.2. Lado de servidor

El lado de servidor se divide en dos grandes apartados, primero se explicarán las decisiones de diseño que conciernen al modelo de datos, es decir, se detallarán las diferentes clases que conforman la *base de datos*. Por otro lado, se explicarán las decisiones de diseño que se han tomado respecto al *sistema de autenticación* y toda su lógica.

3.2.1. Base de datos

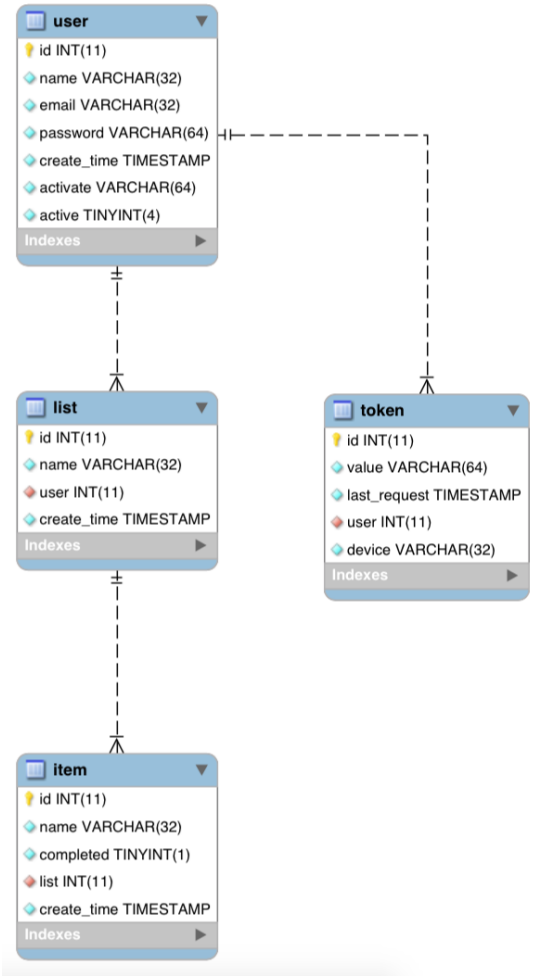


Figura 3.2: Diagrama de la base de datos.

User

En la tabla **user** se almacenarán los usuarios que se registren al sistema y cada uno de los registros de los usuarios podrán tener las siguientes propiedades:

1. **id**. Es un entero que identificará internamente al usuario de manera única.
2. **name**. Nombre del usuario, se puede modificar desde el cliente.
3. **email**. Correo electrónico del usuario, se utilizará como identificador del usuario.
4. **password**. Contraseña cifrada del usuario.
5. **create_time**. Timestamp con la fecha de creación del usuario. Se usará en contadas ocasiones como sal criptográfica para llevar a cabo algunos hash a elementos que debamos securizar.
6. **activate**. Código de activación del usuario. Se utilizará para verificar si el correo introducido durante el proceso de registro es de la persona que inicio dicho proceso y no de terceras personas, és unico por cada usuario.
7. **active**. Variable boleana que describe si el usuario se ha activado (mediante el correo de activación) o no.

Token

En la tabla **token** se almacenarán las sesiones activas de los usuarios que inicien sesión en el sistema, cada sesión tiene las siguientes propiedades:

1. **id**. Es un entero que identificará internamente la sesión de manera única.
2. **value**. Es el valor del token de sesión. El valor de este campo lo usará el usuario como método de identificación sólo durante la duración de su sesión.
3. **last_request**. Timestamp que almacena la última petición al servicio web. Se usará para determinar cuándo una sesión caduca (pasados 10 días de la última petición).

4. **user**. Es el identificador del usuario al cual pertenece la sesión. Es una clave foránea del identificador de la tabla usuario.
5. **id**. Es el identificador único de dispositivo. Se utiliza para que un mismo usuario pueda iniciar en varios dispositivos al mismo tiempo.

List

En la tabla **token** se almacenan todas las listas creadas por los usuarios, cada lista está formada por las siguientes propiedades:

1. **id**. Es un entero que identificará internamente una lista.
2. **name**. Es el nombre de la lista, introducido por el usuario durante su creación.
3. **user**. Es el identificador del usuario al cual pertenece la lista. Es una clave foránea del identificador de la tabla usuario.
4. **create_time**. Timestamp que representa la fecha de creación de la lista.

Item

En la tabla **item** se almacenan todas item creados por los usuarios, cada ítem está formado por las siguientes propiedades:

1. **id**. Es un entero que identificará internamente un ítem.
2. **name**. Es el nombre del ítem, introducido por el usuario durante su creación.
3. **completed**. Es una variable boleana que determina si ése ítem está marcado como completado o no.
4. **list**. Es el identificador de la lista a la cual pertenece el ítem. Es una clave foránea del identificador de la tabla list. Dado que es una relación 1:n, un ítem sólo puede pertencer a una lista y una lista puede contener más de un ítem. Si el usuario desea que un mísmo ítem pertenezca a dos o más listas diferentes deberá crear tantós ítems con el mismo nombre en tantas listas como desee y en la base de dátos se almacenarán con identificadores diferentes.
5. **create_time**. Timestamp que representa la fecha de creación del ítem.

3.2.2. Sistema de autenticación

Para llevar a cabo el sistema de autenticación que usará la aplicación tomaré como referencia el método de autenticación OAuth (Open Authentication).

La principal característica del método OAuth es que para identificar a un cliente de un servicio no es necesario conocer su identidad real, únicamente se utiliza un código que lo representará temporalmente, normalmente durante la duración de una sesión, ese código se conoce como token de sesión.

La principal ventaja de este método es que el cliente no facilitará sus datos personales durante una sesión si quiere utilizar los servicios del sistema, de esta forma, a un posible atacante le sería imposible conocer la información de el usuario, si éste escuchara el proceso de comunicación cliente-servidor. Por otra parte, el inconveniente viene dado a que si un posible atacante conoce el token de sesión de un usuario puede utilizar los servicios del sistema suplantando la identidad del usuario legítimo, es por eso que debemos proteger muy bien el token durante toda la sesión.

Para nuestra aplicación la sesión se considera el tiempo entre que el usuario se identifica para empezar a usar los servicio hasta que el sistema considera que no va a utilizarlos más, en mi caso, como decisión de diseño, el sistema considerará que no utilizará más el servicio pasados **10 días** desde el último uso del token de sesión. He considerado un timeout de sesión dado que si el token existiera indefinidamente, las probabilidades de conocer un token válido aleatoriamente, aumentarían conforme pasa el tiempo, dado que la cantidad de token disponibles iría aumentando, conforme se autentican más usuarios.

En el caso que el usuario decida cerrar la sesión, el cliente, simplemente eliminará el token de sesión de la base de datos local y pasado el tiempo de timeout el servidor eliminará la sesión, pero el servidor no sabe explícitamente cuándo un usuario ha decidido cerrar sesión, simplemente espera a que ésta caduque.

Inicio de sesión

Durante el proceso de inicio de sesión, el cliente debe enviar su dirección de correo electrónico, la contraseña y el id del dispositivo del usuario al servicio web y éste se encarga de generar una sesión, la cual se identifica mediante un token de sesión y tiene las siguientes características:

1. Es un valor pseudoaleatorio criptográficamente seguro.

2. Se genera utilizando como sal criptográfica, el id de dispositivo, fecha de creación del usuario y una constante:

$$\text{hash}(\text{pseudo} + \text{fecha} + \text{constante} + \text{iddispositivo})$$

3. Cuando el servidor genera el token, éste es enviado al cliente, el cual lo almacena y lo utilizará como prueba de autenticación para posteriores peticiones, únicamente durante la duración de su sesión.

Registro

De la misma manera que generamos y almacenamos de forma segura el token de sesión, es necesario almacenar la contraseña del usuario de manera cifrada, para evitar que un atacante la obtenga en texto claro y sea utilizable. Cuando un usuario se registra en la aplicación, introduce su nombre, su dirección de correo electrónico, y contraseña y se envía al servicio web. En vez de almacenar la contraseña en texto plano en la base de datos, únicamente se almacena un hash de ésta, añadiéndole como sal criptográfica la fecha de creacion del usuario (el timestamp, que se puede considerar como un valor pseudoaleatorio, dado que no hay forma de saber en que milisegundo exacto el servidor ha almacenado la información del usuario) y una constante. Lo que conseguimos aadiendo una sal criptográfica.

Los requisitos que debe cumplir una contraseña en esta aplicación son los siguientes:

1. Que tenga más de 7 caracteres.

2. Que tenga al menos un número.

3. Que tenga al menos una letra (ya sea mayúscula o minúscula).

Una vez creado el usuario es necesario activarlo mediante el correo electrónico proporcionado, esto es así para verificar esa dirección de correo electrónico y confirmar que realmente pertenece al usuario registrado y no a terceras personas. La verificación se hace mediante un enlace único (dado que contiene el código de activación de dicho usuario) que se envía mediante correo electrónico. Cuando el usuario

presiona el enlace, se ejecuta un script en el servidor que actualiza el campo de activo de dicho usuario.

Modificación de contraseña

Dado que se ha especificado que dentro de nuestra aplicación también sea posible modificar la contraseña, la manera de llevar a cabo la modificación mediante el cliente y utilizando una sesión existente, de esta forma, como identificación se usará (como en cualquier otro caso de comunicación cliente servidor) el token de sesión.

El cliente debe proporcionar al servidor la contraseña nueva, la antigua y el token de sesión actual, y en el propio servidor se comprueba que las propiedades de la nueva contraseña sean correctos.

Múltiples sesiones $\leftrightarrow$ mismo usuario

Despues de diferentes pruebas sólo almacenando id de usuario y token por cada sesión descubrí que si iniciaba sesión con un usuario en diferentes disposivos se cerraba la sesión del anterior, dado que en la base de datos, se almacenaba el id de usuario como variable única.

Para evitar ese inconveniente, era necesario hacer que la variable de id de usuario no sea única y ademas decidí introducir un tercer parámetro (a parte del identificador de usuario, y el identificador de la sesión) para poder iniciar sesión en diferentes disposivos al mismo tiempo. El parámetro utilizado es el identificador de dispositivo, único por cada dispositivo.

3.3. Lado de cliente

La aplicación se divide en cuatro grandes secciones, pantalla principal, pantalla de lista, pantalla de usuario y pantalla de inicio de sesión.

3.3.1. Pantalla principal

Es la pantalla de inicio de la aplicación y consta de 2 partes diferenciadas, la tabla de listas y el menú lateral. En el menú lateral se irán añadiendo los diferentes

apartados de la aplicación, de momento sólo hay un apartado que es el panel de usuario si se da el caso que existe una sesión abierta, en caso contrario llevará a la pantalla de inicio de sesión.

En la tabla de listas se muestran todas las listas creadas hasta el momento si existe una sesión abierta y el usuario tiene listas creadas, si no aparecerá un texto informativo. Cada elemento de la tabla nos dará la información de la lista, que es el nombre, la imagen y la fecha de creación.

3.3.2. Pantalla de lista

Esta pantalla es la que aparece cuando seleccionamos una de nuestras listas. En esta pantalla se muestra como encabezado la imagen que representa la lista junto a un listado de los ítems que conforman la lista seleccionada, cada ítem muestra la siguiente información:

1. Nombre del ítem.
2. Fecha de creación.
3. Checkbox modificable con el estado actual del ítem (completado o no).

3.3.3. Pantalla de usuario

Esta pantalla muestra la información de usuario y dará la opción de modificarla, la información que permite modificación es el nombre de usuario, la imagen de perfil y la contraseña. Para modificar la contraseña es necesario introducir la contraseña anterior y la nueva contraseña dos veces como verificación.

3.3.4. Pantalla de inicio de sesión

En pantalla de inicio de sesión hay un formulario que nos permitirá introducir nuestros datos de acceso (dirección de correo electrónico y contraseña) y de esta forma iniciar sesión. Además existe un botón que nos abre un nuevo formulario para registrarnos en el sistema, el cual nos pide, dirección de correo electrónico, nombre y contraseña que queramos usar, como en el caso anterior, la contraseña habrá que introducirla dos veces.

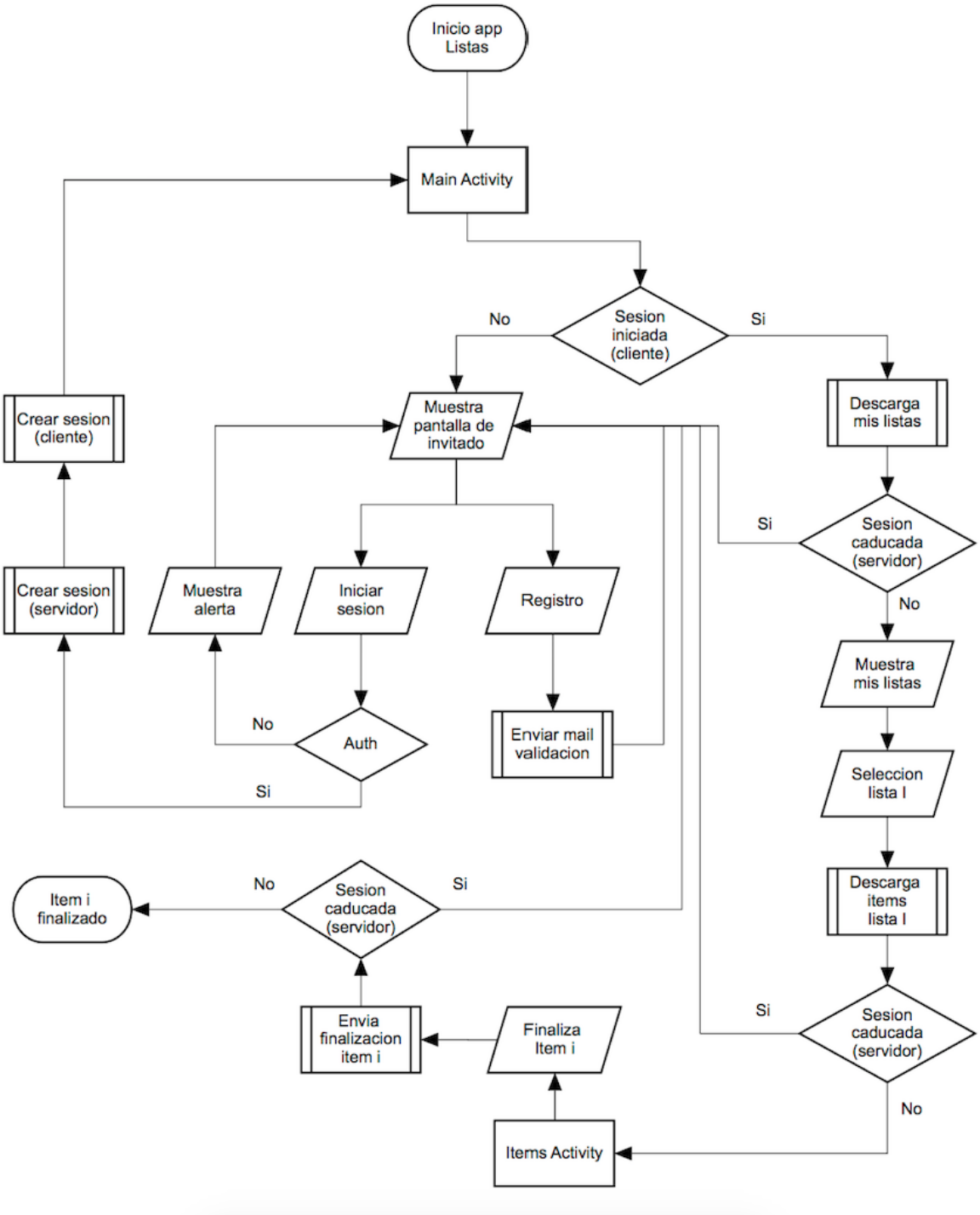


Figura 3.3: Diagrama de flujo de la aplicación.

Capítulo 4

Implementación

A continuación se detallará el desarrollo del sistema tanto del lado del servidor, como del lado del cliente.

4.1. Tecnologías Utilizadas

A continuación se enumeran las diferentes tecnologías utilizadas en las dos partes que conforman el sistema.

4.1.1. Servidor

- Para implementar la base de datos se utilizara MySQL.
- PHP, para desarrollar la lógica del servidor.
- HTTP como protocolo de comunicación.
- SSL como capa de conexión segura.
- XML como estructura de información durante la comunicación.

4.1.2. Cliente

- Programación nativa en Android (Java).

- Persistencia de datos en XML (llamado SharedPreferences en Android) para almacenar datos de usuario, como token de sesión, nombre o correo electrónico.

4.2. Lado del servidor

Como servidor de pruebas estoy utilizando el microcomputador Raspberry PI (Modelo B ARM 700MHz, 512 MB RAM y SD de 8GB) `http://www.raspberrypi.org` con el sistema operativo Raspbian `http://www.raspbian.org` basado en Debian con núcleo Linux.

Es un sistema muy limitado pero perfecto para hacer pruebas.

4.2.1. Administración del servidor de pruebas

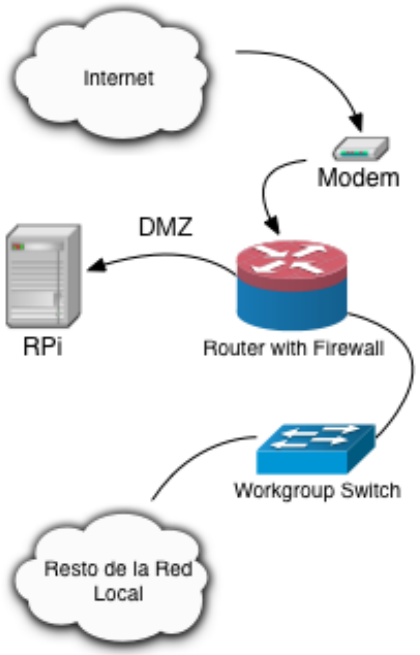


Figura 4.1: Topología de la red de pruebas.

Como puerta de enlace de la red, utilizo un modem-router comercial que facilitan las empresas proveedoras de servicios de Internet con firewall integrado que permite

bloquear ataques como IP Spoofing, Land Attack, Ping de la muerte, IP con logitud cero, Smurf Attack, UDP port loopback, Snork Attack, TCP null scan, y TCP SYN flooding. Además de estos servicios, estos dispositivos tienen la posibilidad de establecer una zona desmilitarizada o DMZ, la cual permite a las máquinas que la componen comunicarse libremente mediante conexiones a la red externa, mientras que la red interna se comunica con el exterior mediante el filtraje del cortafuegos.

Cortafuegos

Dado que el servidor se encuentra en una zona *insegura* de la red (DMZ) he establecido un cortafuegos interno mediante la aplicacion IPTables. Cada vez que se inicia el servidor se fijan las reglas que a continuación se listan, que establecen:

- Cualquier conexión de redirección no permitida (FORWARD), dado que este servidor no ejerce tareas de enrutamiento.
- Cualquier conexión saliente esta permitida por defecto.
- Cualquier conexión perteneciente al grupo fail2ban-ssh relacionada con el servicio ssh, se establecerá automáticamente por otra aplicación.
- Cualquier conexión de entrada local (localhost o lo) está permitida.
- Conexiones entrantes no permitidas por defecto, salvo las siguientes excepciones:
    - Que procedan de la red local y accedan a los servicios de los puertos 137, 138, 139 y 445 (que pertenecen a servicios de comparticion de ficheros, SMB y NetBIOS) dado que el servidor también como lo utilizo como NAS (Network Attached Storage) para compartir almacenamiento únicamente a la red local (que no tiene nada que ver con el trabajo final de máster realizado).
    - Cualquier conexión al puerto 80 (HTTP), aunque el propio servidor web redirige todas las conexiones al puerto 443 (HTTPS).
    - Cualquier conexión al puerto 443 (HTTPS) con el cual conecta el cliente.

- Cualquier conexión al puerto 22 (SSH) para poder conectarme remotamente a la consola del sistema, aunque por seguridad es recomendable mantenerlo cerrado.

- Conexiones de respuesta a una conexión iniciada por el servidor (RELATED,ESTABLISHED).

Normas IPTables

```
:INPUT DROP [16014:4486878]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [72828:44441417]
:fail2ban-ssh - [0:0]
-A INPUT -p tcp -m multiport --dports 22 -j fail2ban-ssh
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED
-j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A INPUT -s 192.168.1.0/24 -p udp -m udp --dport 1
37 -j ACCEPT
-A INPUT -s 192.168.1.0/24 -p udp -m udp --dport
138 -j ACCEPT
-A INPUT -s 192.168.1.0/24 -p tcp -m state --state
NEW -m tcp --dport 139 -j ACCEPT
-A INPUT -s 192.168.1.0/24 -p tcp -m state --state
NEW -m tcp --dport 445 -j ACCEPT
-A fail2ban-ssh -j RETURN
COMMIT
```

4.2.2. Base de datos

El servicio MySQL sólo permite conexiones de la propia máquina dónde escucha (localhost), ésa es una de las razones por que el cortafuegos local permite las conexiones localhost (lo).

4.2.3. Servidor web

Como servidor web estoy utilizando la aplicación Apache. Como he explicado en el punto anterior, el cortafuegos interno permite las conexiones entrantes mediante HTTP, pero el servidor Apache redirige (reescribe) automáticamente esas conexiones al puerto 443 (HTTPS) mediante la siguiente configuracion de Apache:

```
<VirtualHost *:80>
        RewriteEngine on
        ReWriteCond %{SERVER_PORT} !^443$
        RewriteRule ^/(.*) https://%{HTTP_HOST}/\$1
[NC,R,L]
</VirtualHost>
```

Además, para activar la capa de conexión segura en el puerto 443 es necesario indicarlo expresamente en la configuración de Apache:

```
<VirtualHost *:443>
    ServerName "SimpleLists Application"
    ServerAdmin appsimplelists@gmail.com
    SSLEngine on
    SSLCertificateFile /etc/apache2/certs/ca.crt
    SSLCertificateKeyFile /etc/apache2/certs/ca.key
    DocumentRoot /home/pi/simpleList
    <Directory />
        Options FollowSymLinks
```

```
        AllowOverride None
    </Directory>
</VirtualHost>
```

En esta configuración se establecen varios parámetros:

- Nombre del servidor y nombre de su administrador.
- Parámetro de activacion de SSL con el certificado y clave privada utilizados.
- Camino del directorio del servicio web y contra qué directorio corresponde remotamente (en este caso la url principal /").

Para las pruebas he generado un certificado autofirmado con algoritmo RSA y 1024 bits mediante la aplicacion OpenSSL con un año de validez:

```
openssl genrsa -out ca.key 1024
openssl req -new -key ca.key -out ca.csr
openssl x509 -req -days 365 -in ca.csr -signkey ca.key
-out ca.crt
```

- Línea 1: Generamos una clave privada de algoritmo RSA y 1024 bits.
- Línea 2: Generamos un CSR (Certificate Signing Request) que nos permite generar un certificado pendiente de firmar por una entidad certificadora de confianza, en este paso nos pedirán nuestros datos de identificación.
- Línea 3: Firmamos nosotros mismos el CSR, generando ya un certificado ya firmado y listo para usarse.

Dado que nuestra firma no tiene validez para ningún navegador, éste nos advertirá que el servidor puede estar suplantando la identidad del nombre que se especifica:

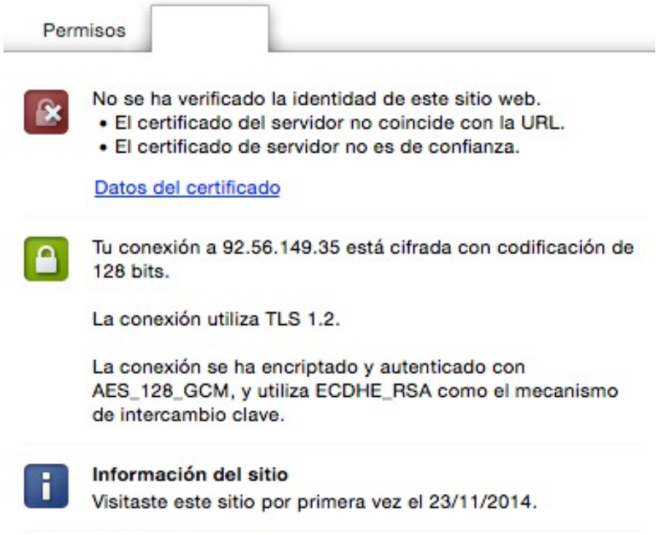


Figura 4.2: Navegador advirtiendo de un certificado firmado por una entidad en la que no confía.

Como podemos comprobar, nuestro servidor no está autenticado, dado que la firma del certificado no es de confianza y el nombre del emisor del mismo no coincide con la URL dado que no pertenece a ningún dominio. Pero a pesar de que no cumplimos con la premisa de seguridad de la información de la autenticación, si cumplimos con la confidencialidad e integridad, dado que la conexión va perfectamente cifrada utilizando el protocolo TLS 1.2 y encriptación AES de 128bits. Y como más adelante vamos a crear nuestro propio cliente podemos especificar que éste sí confíe en nuestra firma digital.

fail2ssh

Como he explicado en un punto anterior, he permitido el acceso mediante ssh utilizando el cortafuegos interno (no recomendable), pero de esta forma puedo acceder al servidor remotamente, por ejemplo desde mi dispositivo móvil. Dado que el puerto 22 se encuentra abierto desde el exterior me he encontrado con intentos de conexión fallidos de posibles atacantes, para solventar este problema he llevado a cabo las siguientes medidas de seguridad:

- Utilizar una contraseña del sistema robusta, con números, símbolos, mayúscu-

las y minúsculas, para dificultar un ataque por fuerza bruta.

- Utilizar la aplicacion fail2ban, que permite, limitar el numero máximo de conexiones fallidas a un servicio concreto durante un período de tiempo para una cierta IP utilizando el cortafuegos interno IPTables (por eso existe la cadena de IPTables fail2ban que he explicado en el apartado cortafuegos). En mi caso he limitado a 3 el número de conexiones fallidas del protocolo ssh, y un tiempo de baneo de 8h ignorando las IP de la red local:

```
[ssh]
enabled  = true
port     = ssh
filter   = sshd
logpath  = /var/log/auth.log
maxretry = 3
bantime  = 28800
ignoreip = 192.168.1.0/24
```

Otra curiosidad viene dada por un scan de puertos de un posible atacante contra el servidor web utilizando la siguiente aplicación https://github.com/robertdavidgraham/masscan, segun observo en el log de conexion de Apache:

Figura 4.3: Intento de escaneo de un posible atacante puertos mendiante HTTP.

Funciones criptográficas

Las funciones criptográficas utilizadas en los scripts PHP han sido:

- **openssl_random_pseudo_bytes**. Devuelve una cadena de bytes criptográficamente seguros. Qué quiere decir? Que la cadena no tiene ninguna relación

entre sí, es decir, no tiene porqué formar ningún carácter legible si la codificamos a utf-8, por ejemplo. Esta función la he utilizado para generar token aleatorios.

- **hash_sha256**. Crea un hash SHA256 que devuelve una cadena de 64 cifras hexadecimales. Esta función se utiliza para generar ya sean los token aleatorios o los hash de las contraseñas.

Imágenes de usuario/listas

La forma de obtener de forma segura las imágenes de usuario y de las listas por parte del cliente, ha sido mediante el token de sesión. En caso que el token de sesión no exista el servidor devuelve el error 403 (prohibido) como en cualquier otro caso.

Todas estas imágenes se almacenan en un directorio del servidor web, el cual no es accesible, dado que se ha utilizado el archivo **.htaccess** con el contenido **deny from all**, esto es así porque el usuario no accederá directamente a sus imágenes si no que accederá mediante un script que se encargará de obtener la imágen que está pidiendo en ése momento.

El script recibe el token de sesión, de forma que puede identificar al usuario y obtener la imágen que se especifique mediante otro parámetro *GET*. A continuación se muestra el script que permite renderizar una imagen dado un camino de directorio.

```
$img = imageCreateFromPng($path);
imageAlphaBlending($img, true);
imageSaveAlpha($img, true);
imagePng($img);
imageDestroy($img);
```

Caducidad de la sesión

Dado que el número de sesiones puede ir aumentando y no disminuye (cosa que puede hacer que se llene la memoria del sistema del servidor), y además hay más

probabilidades de obtener un token válido, dado que se van acumulando. He establecido un tiempo de sesión de 10 días si el usuario no utiliza la aplicación. En el servidor, existe una tarea de **cron** (una aplicación que permite ejecutar scripts periódicamente), que se ejecuta una vez al dia, va comprobando las sesiones que llevan más de 10 dias sin usarse y si es así las elimina:

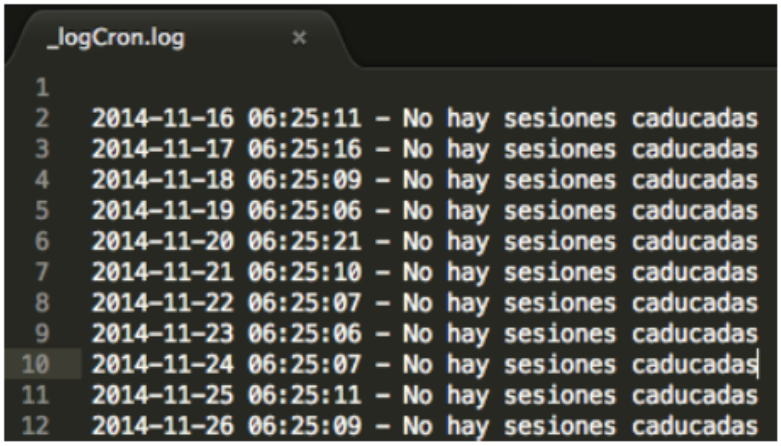


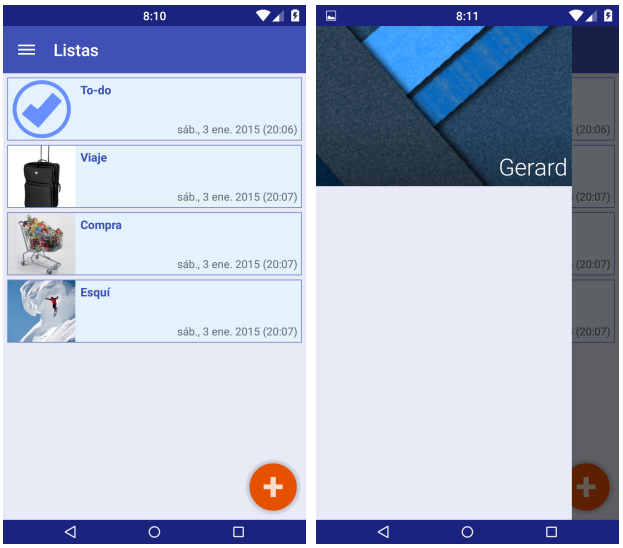
Figura 4.4: Log del script de sesiones caducadas.

4.3. Lado del cliente

Para llevar a cabo el proyecto Android se ha utilizado la aplicación de Android Studio 1.0.2 . En la compilación se ha utilizado el Android SDK 21 (Android Lollipop 5.0) y la aplicación es compatible con versiones de Android superiores a 4.0.

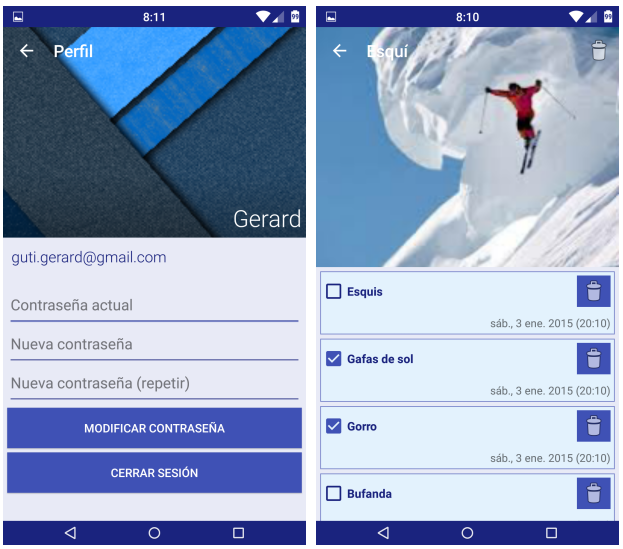
Figura 4.5: Android Studio.

4.3.1. Algunas capturas del cliente Android



(a) Pantalla de Listas (b) Menú Lateral

Figura 4.6: Capturas Android 1



(a) Pantalla de Usuario (b) Pantalla de Items

Figura 4.7: Capturas Android 2

Capítulo 5

Presupuesto

Concepto	Precio (€)	Cantidad	Total (€)
Raspberry PI Model B	39,5	1	39,5
SD 8GB	8	1	8
PC para programar	450	1	450
Router/Firewall comercial	70	1	70
Dominio + Hosting (mes)	4,95	12	59,4
Programador (hora)	20	20	400
TOTAL			1026,9

Figura 5.1: Presupuesto.

Capítulo 6

Futuras mejoras

6.1. Lado de cliente

1. Añadir la posibilidad de renombrar listas.
2. Añadir la posibilidad de renombrar ítems.
3. Añadir la posibilidad de modificar el correo electrónico del usuario.

6.2. Lado de servidor

1. Obtener un hosting con mayor capacidad.
2. Obtener un dominio, para identificar el servidor.
3. Obtener un certificado firmado por una entidad de certificación.
4. Mejorar el diseño del correo de activación de usuario, aplicándole una hoja de estilos.
5. Añadir un sistema de recordatorio de contraseña.
6. Añadir un sistema para que el usuario se pueda dar de baja del servicio.

Capítulo 7

Conclusión

El método de autenticación diseñado en este trabajo final de máster permite identificar a los usuarios de forma *anónima* durante la duración de la sesión, además también podemos decir que el proceso se ejecuta de manera rápida dado que, para el servidor, simplemente es necesario comprobar un campo de datos sin necesidad de procesarlo, para verificar la identidad del usuario (que es el token de sesión), pero el inconveniente de este sistema es que depende mucho de otras tecnologías para un funcionamento seguro, como es el caso de SSL, dado que si un atacante conoce el token de sesión de un usuario legítimo, éste podria utilizar el servicio suplantando la identidad del usuario.

Dado los casos de los últimos meses en materia de vulnerabilidades en SSL, es necesario encontrar un sistema de autenticación lo más autónomo posible, pero en muchas ocasiones es necesario recurrir a otras tecnologías para un buen funcionamiento del sistema en conjunto. Ya sea en criptografía, como es el caso de la utilización de diferentes algoritmos hash o criptográficos o diferentes tipos comunicación segura, como es el caso de SSL o TLS.

Bibliografía

[1] http://es.wikipedia.org/wiki/seguridad_informática.

[2] http://stackoverflow.com/questions/20318770/send-mail-from-linux-terminal-in-one-line.

[3] http://stackoverflow.com/questions/26440879/how-do-i-use-drawerlayout-to-display-over-the-actionbar-toolbar-and-under-the-st.

[4] http://stackoverflow.com/questions/3509333/how-to-upload-save-files-with-desired-name.

[5] http://stackoverflow.com/questions/5061675/emulate-a-403-error-page.

[6] http://www.fceia.unr.edu.ar/lcc/cdrom/instalaciones/latex/latex.html.

[7] http://www.php.net.

[8] http://www.w3.org.

[9] Reto Meier. *Professional Android Application Development.* Wrox, 2009.