



Escola d'Enginyeria de Telecomunicació i
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Aplicaciones móviles para teleasistencia

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad en
Sistemas de Telecomunicación

AUTOR: Cristóbal Ramos Pérez

DIRECTOR: Juan López Rubio

FECHA: 18 Octubre de 2011

Título: Aplicaciones móviles para teleasistencia

Autor: Cristóbal Ramos Pérez

Director: Juan López Rubio

Fecha: 18 de octubre de 2011

Resumen

En este proyecto se ha desarrollado una aplicación de teleasistencia para el sistema operativo móvil Android. La aplicación consiste en un sistema de alarmas capaz de controlar la toma diaria de medicamentos por parte del paciente.

La programación de la aplicación se ha dividido en dos extremos bien diferenciados.

En un extremo encontramos la aplicación para el dispositivo móvil, desarrollado en el entorno de programación Eclipse, con los complementos Android adecuados. En este bloque, además, encontramos el sistema de alarmas, desarrollado como un servicio java independiente de la aplicación.

En el otro extremo encontramos la implementación de un servidor capaz de almacenar los datos de cada paciente mediante el marco de trabajo Play.

Los dos extremos estarán en constante actualización para que cualquier modificación, ya sea por eliminación o por cambios en los datos, sea detectada tanto en el servidor como por la aplicación.

Title: Teleassistance Mobile Applications

Author: Cristóbal Ramos Pérez

Director: Juan López Rubio

Date: June, 18th 2006

Overview

In this Project we have developed a teleassistance application for Android. This application consists in an alarm system able to control the daily medicine dose for the patient.

The programming of this application has been divided into two well-differenced blocks.

In the first block we can find the implementation of a web server developed with Play Framework. This server is able to store data of each patient we have.

In the second block, we can find the mobile application for Android, developed in Eclipse, with the appropriate Android complements. In this part, we can also find the alarm system, developed as a java independent service.

The whole system will be in continuous update. The user will have the application and the server updated in order to have all data modified in case the patients want to modify, remove or add some dose to their database.

Dedicado a mi familia.
Gracias a mis padres y a mi hermano, por apoyarme durante este largo viaje.
Por supuesto gracias a María, por animarme siempre.
A todos aquellos que alguna vez me preguntaron cómo me iba.
GRACIAS.

ÍNDICE

INTRODUCCIÓN.....	1
CAPÍTULO 1. VISIÓN GENERAL	2
1.1. Motivación.....	2
1.2. Objetivos	2
1.3. Teleasistencia	2
1.4. Terminales.....	3
1.5. Arquitectura	4
CAPÍTULO 2. DESARROLLO DE APLICACIONES PLAY FRAMEWORK	6
2.1 Introducción a Play Framework.....	6
2.2 Instalación y preparación del entorno para utilizar Play	6
2.3 Estructura de un proyecto Play Framework.....	8
2.3.1. Fichero routes	8
2.3.2 Clases Controller.....	9
2.3.3 Clases Model	9
2.3.4 Carpeta Views.....	10
2.4 Tutorial “Hola Mundo” de Play Framework.....	10
CAPÍTULO 3. DESARROLLO DE APLICACIONES ANDROID	13
3.1 Instalación del entorno de desarrollo: Eclipse + Android	13
3.2 Creación de un terminal Android virtual (AVD)	14
3.3 Estructura de un proyecto Android	15
3.3.1 Carpeta src.....	15
3.3.2 Carpeta gen	15
3.3.3 Carpeta res	16
3.3.4 Carpeta assets	16
3.3.5 Librerías	16
3.3.6 Fichero AndroidManifest.xml.....	16
3.4 Activities	17
3.5 Services	18
CAPÍTULO 4. DISEÑO DE LA APLICACIÓN	19
4.1 Diseño del servidor web.....	19
4.1.1 Aspectos generales.....	19
4.1.2 Funcionalidades	20

4.2	Diseño de la aplicación Android	20
4.2.1	Aspectos generales.....	20
4.2.2	Funcionalidades	21
4.3	Diseño del intercambio de datos. Parsers.	21
4.3.1	Parsing XML en Play. DOM Parser.....	22
4.3.2	Parsing XML en Android. SAX Parser	23
 CAPÍTULO 5. IMPLEMENTACIÓN DE LA APLICACIÓN		26
5.1	Consideraciones generales	26
5.2	Implementación del servidor Play!.....	27
5.2.1	Controller Application.java	28
5.2.2	Models Usuario.java y Pastilla.java.....	29
5.2.3	Fichero Routes.....	30
5.2.4	Carpeta Views.....	31
5.3	Implementación de la aplicación en Android.....	31
5.3.1	Objetos Usuario.java y Pastilla.java.....	32
5.3.2	Handlers HandlerPastillas.java y HandlerUsuarios.java	32
5.3.3	Servicio.java.....	35
5.3.4	Activities	36
5.3.5	Fichero AndroidManifest.xml.....	41
 CAPÍTULO 6. CONCLUSIONES		43
6.1	Conclusiones finales	43
6.2	Posibles ampliaciones	44
6.2.1	Interacción del usuario vía web.....	44
6.2.2	Modo sin conexión	44
6.2.3	Automatización del sistema mediante hardware externo.....	44
6.3	Impacto medioambiental.....	44
 CAPÍTULO 7. REFERENCIAS BIBLIOGRÁFICAS		45
 CAPÍTULO 8. ANEXOS		46
8.1	Anexo 1: Código completo de la aplicación Play	46
8.2	Anexo 2: Código completo de la aplicación Android.....	50

INTRODUCCIÓN

Hoy en día, los terminales móviles tienen la capacidad de ofrecernos multitud de servicios distintos al simple hecho de llamar o enviar mensajes cortos. Todo aquel que tiene un móvil en sus manos no sólo tiene un teléfono, sino que también posee una cámara de fotos, un GPS, una videoconsola...

Con la llegada de los nuevos terminales, podemos tener teléfonos móviles hechos a medida. Cada usuario puede personalizar su aparato para poder contar con multitud de aplicaciones que puedan satisfacer las necesidades de cada cual.

Aún así, los desarrolladores dejan muchas veces de lado a ciertos colectivos de la sociedad, que no se acaban de encontrar integrados en esta nueva era de las comunicaciones.

Así, nace la idea de realizar un proyecto que pueda hacer la vida de las personas mayores un poco más fácil. Un sistema de alarmas capaz de controlar las dosis a tomar de manera diaria.

En este proyecto, desarrollamos una aplicación para terminales Android que se sincronizará con un servidor web en el que encontraremos una base de datos con la información de pacientes y sus correspondientes dosis.

En el capítulo 1 realizamos una descripción del por qué de este proyecto, de las motivaciones que nos han impulsado a realizarlo y los objetivos que esperamos cumplir. Además, realizamos una introducción a los temas principales del proyecto: terminales móviles y teleasistencia.

En el segundo capítulo, se explica cómo desarrollar aplicaciones web con Play Framework, el marco de trabajo escogido para la implementación del servidor web.

En el capítulo 3, encontramos la explicación de cómo desarrollar aplicaciones para el sistema operativo Android mediante el entorno de programación Eclipse.

En el capítulo 4, encontramos el diseño de la aplicación tanto para el servidor web como para la aplicación Android, así como el diseño de la comunicación entre ellos.

En el capítulo 5, podemos encontrar la implementación de los dos bloques.

En el capítulo 6, encontramos las conclusiones del proyecto, así como posibles mejoras del mismo y el impacto medioambiental causado por el desarrollo de éste.

Por último, en el Anexo, podemos encontrar el código completo de la aplicación.

CAPÍTULO 1. VISIÓN GENERAL

1.1. Motivación

La tecnología móvil avanza a pasos agigantados y la aparición de aplicaciones móviles crece de manera exponencial. Todo aquel que disponga de un móvil con conexión a Internet y un Sistema Operativo adecuado tiene a su alcance una gran oferta de aplicaciones con todo tipo de funcionalidades y servicios.

La aparición de estas aplicaciones viene a cubrir la demanda de un sector muy extenso de la población, aunque suele dejar a algunos colectivos fuera del contacto con las nuevas tecnologías.

Uno de estos colectivos es el de las personas mayores. Éstas no suelen tener a su alcance ningún tipo de dispositivo móvil con conexión a Internet, debido seguramente a que no forman parte de la nueva generación tecnológica.

Así surge la idea de desarrollar una aplicación móvil de uso sencillo que les ofrezca un servicio esencial en su día a día. La aplicación, instalada en un teléfono Android, avisaría al usuario mediante alarmas sonoras de las dosis diarias que han de tomar.

1.2. Objetivos

Los objetivos de este proyecto son los siguientes:

- Facilidad de uso en el momento de introducir datos.
- Facilidad en el momento de apagar la alarma.
- Creación de una base de datos alojada en un servidor, ajena al usuario.
- Creación de un servicio disponible aún cuando no estamos dentro de la aplicación
- Intercambio de datos entre el teléfono y el servidor.
- Posibilidad de creación de usuarios y de inicio de sesión de estos, teniendo un fichero de usuarios en la base de datos.
- Posibilidad de modificar las dosis de los medicamentos, así como añadir o eliminar las dosis necesarias.

1.3. Teleasistencia

La teleasistencia se define como un sistema de ayuda dentro y fuera del hogar que cubre las necesidades de aquellas personas que pueden requerir de atención constante o puntual en casos de urgencia durante las 24 horas. Este servicio está principalmente enfocado a personas en situación de dependencia que:

- Viven solos o pasan gran parte del día sin compañía.
- Tienen un aislamiento geográfico o desarraigo social.
- Sufren los riesgos causados por la avanzada edad.
- Personas con discapacidad.
- Padecen enfermedades.
- Familiares y/o cuidadores informales.

También puede ser utilizado por otro tipo de personas tales como niños que permanecen varias horas al día solos, situaciones de post-operatorio, víctimas de la violencia de género, etc.

Mediante la incorporación de las tecnologías de la información y comunicación, en especial las comunicaciones de banda ancha, se permite el seguimiento remoto de los usuarios mejorando así su calidad de vida y la de sus familiares. Este servicio de teleasistencia evoluciona hacia un servicio de video teleasistencia, localización, alarmas técnicas, televigilancia... incluyendo servicios de integración de nuevas tecnologías extendiendo las prestaciones actualmente disponibles en la teleasistencia básica. Las nuevas funcionalidades que hoy en día pueden ser integradas en el servicio son:

- Comunicaciones multimedia.
- Movilidad: localización, seguimiento y presencia monitorizados.
- Facilidades de mensajería (SMS, MMS).
- Automatización y control

1.4. Terminales

El Smartphone es un término comercial para denominar a un teléfono móvil que ofrece más funciones que un teléfono común. Una característica importante de casi todos los teléfonos inteligentes es que permiten la instalación de programas que se adecuen a las necesidades de cada usuario. Estas aplicaciones pueden ser desarrolladas por el fabricante del dispositivo, por el operador o por un tercero.

Algunos ejemplos de teléfonos denominados inteligentes son: BlackBerry, iPhone, Palm y todos los que tienen el sistema operativo Android, S60 ó Windows Mobile.

Tanto Android como Palm están basados en Linux. BlackBerry está basado en un kernel propietario. iPhone se basa en OS X. S60 se basa en Symbian y Windows Mobile en Windows CE. El kernel es el núcleo del teléfono, el software responsable de facilitar a los programas acceso seguro al ordenador, el encargado de gestionar recursos.

Android es el sistema que mayor adaptabilidad presenta, ya que cada vez se está utilizando en más dispositivos, no sólo teléfonos móviles.

Otro punto importante en Smartphones es la interfaz, uno de los factores que más aprecia el usuario. iPhone tiene un estilo muy marcado con un sencillo acceso y con una pantalla que no requiere puntero. Android también cuenta con una interfaz sencilla e intuitiva con una gran precisión. Sin embargo, Windows Mobile siempre ha necesitado el uso de punteros.

Pero sin duda uno de los puntos fuertes de los teléfonos inteligentes es poder incluir nuevas funcionalidades y nuevas aplicaciones. Para ello es importante que la plataforma admita desarrollo de terceros.

Debido a la sencillez de interfaz de los teléfonos Android y al hecho de que podemos desarrollar sus aplicaciones de manera libre, hemos seleccionado este sistema operativo para desarrollar nuestro proyecto.

1.5. Arquitectura

El proyecto final tiene, como se ha avanzado con anterioridad, 2 bloques bien diferenciados.

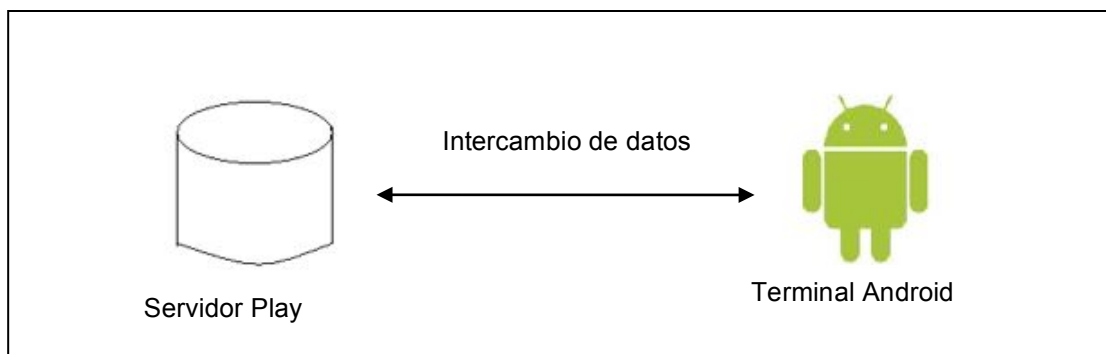


Fig. 1.1 Diferenciación por bloques del sistema

En un primer término tenemos la aplicación para Android. Este programa será el que esté visible para el usuario y con el que podrá interactuar: consultar sus dosis, añadir o eliminar dosis, detener las alarmas...

El segundo bloque del proyecto consiste en el servidor Web desarrollado con Play Framework. Esta parte será totalmente ajena y desconocida para el usuario.

La comunicación entre ambos extremos se realizará siguiendo la estructura presentada en la Fig.1.2.

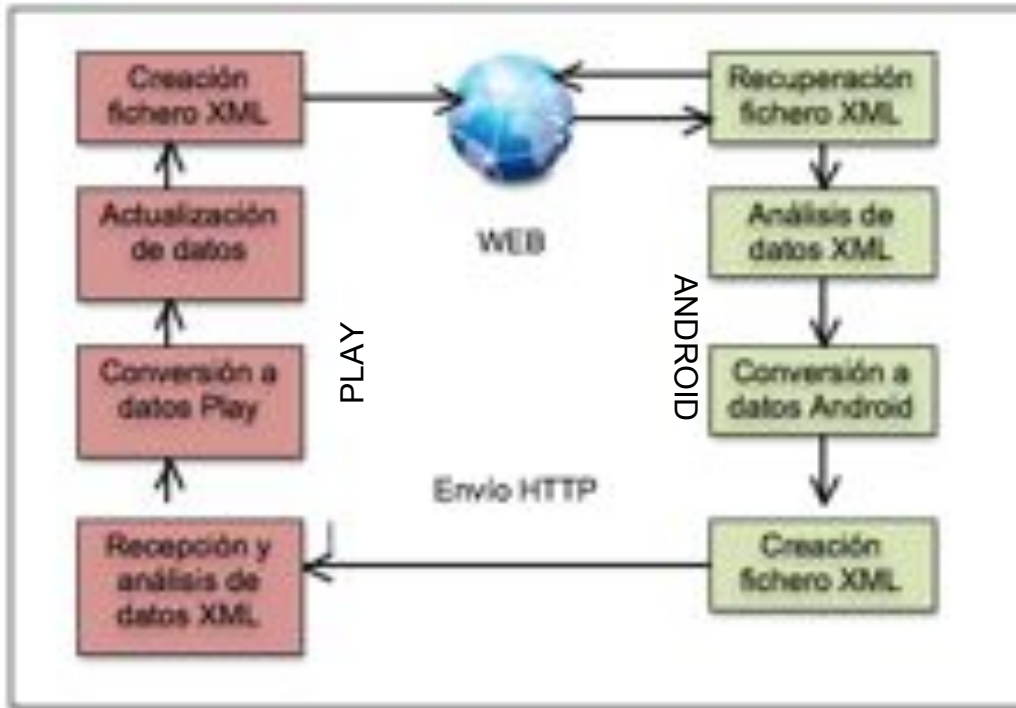


Fig. 1.2 Estructura de la comunicación Android-Play Framework

La aplicación Android descargará un fichero XML con los datos de las diferentes dosis del usuario. Cada vez que se realice un cambio en la aplicación, ésta enviará mediante el protocolo HTTP otro fichero XML hacia el servidor para tener actualizada en todo momento la base de datos.

Ahora ya lo tenemos instalado y listo para usar. Para crear una nueva aplicación simplemente hemos de realizar la siguiente instrucción en consola:

```
MacBook-Pro-de-Cristobal-Ramos-Perez:play Cristo$ ./play new myApp
~
~
~
~ play! 1.2.1, http://www.playframework.org
~
~ The new application will be created in /Users/Cristo/play/myApp
~ What is the application name? [myApp] myApp
~
~ OK, the application is created.
~ Start it with : play run myApp
~ Have fun!
```

Fig. 2.3 Creación de una nueva aplicación Play

Y para ejecutarla basta con:

```
MacBook-Pro-de-Cristobal-Ramos-Perez:play Cristo$ ./play run myApp
~
~
~
~ play! 1.2.1, http://www.playframework.org
~
~ Ctrl+C to stop
~
Listening for transport dt_socket at address: 8080
20:25:15,274 INFO ~ Starting /Users/Cristo/play/myApp
20:25:15,661 WARN ~ You're running Play! in DEV mode
20:25:15,798 INFO ~ Listening for HTTP on port 9000 (Waiting a first request to
start) ...
```

Fig. 2.4 Ejecución de la aplicación Play creada

Cabe destacar que nuestras aplicaciones se alojarán en nuestra propia *localhost*.

En nuestro caso, utilizaremos Play con el entorno de programación Eclipse. Para ello, necesitamos preparar la carpeta en la que hemos creado nuestra aplicación para utilizarla en él.

```
MacBook-Pro-de-Cristobal-Ramos-Perez:play Cristo$ ./play eclipsify myApp
~
~
~
~ play! 1.2.1, http://www.playframework.org
~
~ OK, the application is ready for eclipse
~ Use File/Import/General/Existing project to import /Users/Cristo/play/myApp in
to eclipse
~
~ Use eclipsify again when you want to update eclipse configuration files.
~ However, it's often better to delete and re-import the project into your works
pace since eclipse keeps dirty caches...
```

Fig. 2.5 Preparación de la nueva aplicación para su uso en Eclipse

Ahora ya tenemos la carpeta lista para importarla en Eclipse. Así, iniciamos el entorno de programación y clicamos en *File* → *Import* → *General* → *Existing Projects into Workspace*:



Fig. 2.6 Importación de la carpeta de la aplicación a Eclipse

Ahora ya tenemos nuestro proyecto importado en Eclipse.

2.3 Estructura de un proyecto Play Framework

2.3.1. Fichero routes

Este es el principal punto de entrada al proyecto. Este fichero define todas las URL's accesibles de la aplicación.

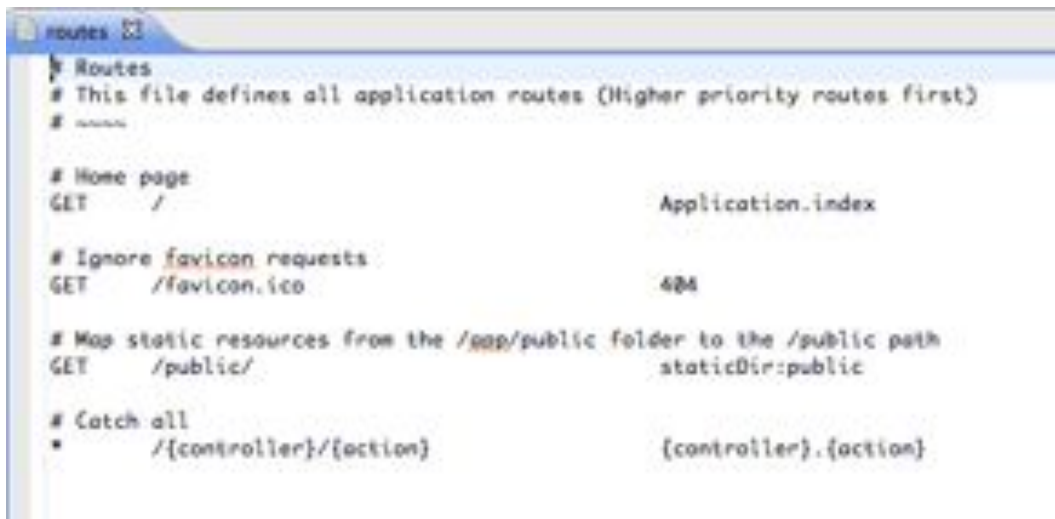
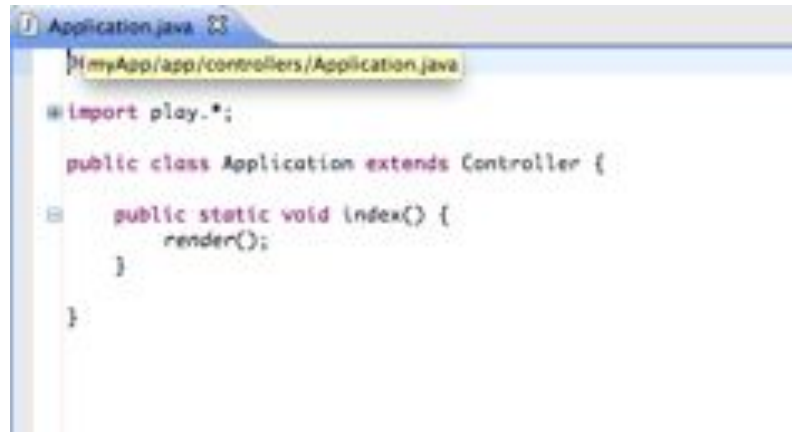


Fig. 2.7 Fichero routes

Este simple fichero comunica a Play que cuando el servidor reciba una petición GET para el camino designado, debe llamar al método Java `Application.index`. En este caso, `Application.index` es un acceso directo a `controllers.Application.index`.

2.3.2 Clases Controller

Una aplicación Play tiene multitud de puntos de entrada, uno para cada URL. Llamamos a estos métodos *ACTION*. Los métodos *ACTION* son definidos en unas clases especiales llamadas *controllers*. Veamos la estructura de una clase *controller*:



```
Application.java
| MyApp/app/controllers/Application.java
import play.*;

public class Application extends Controller {

    public static void index() {
        render();
    }
}
```

Fig. 2.8 Clase controller

Vemos que la clase *controller* extiende de la clase *play.mvc.Controller*. Esta clase entrega todos los métodos a utilizar en los *controllers*, como el método *render()* utilizado en el método *index()*.

El método *index()* está definido como *public static void*. Así es como se definen los métodos. Podemos ver que los métodos son estáticos, porque las clases *controller* nunca son instanciadas. Están marcados como públicos para autorizar al marco de trabajo a llamarlos en respuesta a una URL demandada. Siempre devuelven *void*.

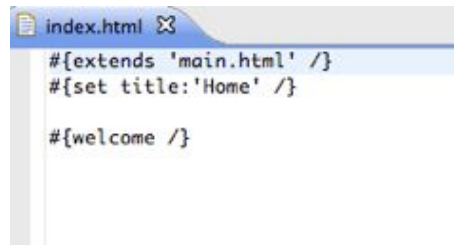
El Action *index* por defecto es simple: llama al método *render()*, el cual le dice a Play que devuelva una plantilla (*template*). Usar una plantilla es la manera más común (pero no la única) de generar una respuesta HTTP.

2.3.3 Clases Model

En esta carpeta encontraremos las clases u objetos a utilizar en la aplicación. Estos modelos se crean para poder almacenar los datos de manera que no se pierdan a la hora de cambiar entre aplicaciones.

2.3.4 Carpeta Views

Las plantillas son simples ficheros de texto almacenadas en la carpeta Views. Cuando no especificamos ninguna en especial, se selecciona la plantilla por defecto.



```
index.html
#{extends 'main.html' /}
#{set title:'Home' /}

#{welcome /}
```

Fig. 2.9 Plantilla por defecto index.html

El contenido de la plantilla parece ser muy sencillo. De hecho, todo lo que se ve son tags de Play:

- `#{welcome /}`: genera un mensaje de bienvenida para el navegador
- `#{extends /}`: avisa de que esta plantilla extiende de otra plantilla, main.html. La herencia de plantillas es un poderoso concepto que permite crear complejas páginas web reusando partes comunes.

La plantilla main.html tiene el siguiente aspecto:



```
main.html
<!DOCTYPE html>

<html>
<head>
  <title>#{get 'title' /}</title>
  <meta charset="utf-8">
  <link rel="stylesheet" media="screen" href="#{"/public/stylesheets/main.css"}">
  #{get 'moreStyles' /}
  <link rel="shortcut icon" type="image/png" href="#{"/public/images/favicon.png"}">
  <script src="#{"/public/javascripts/jquery-1.5.2.min.js"}" type="text/javascript" charset="utf-8"><
  #{get 'moreScripts' /}
</head>
<body>
  #{doLayout /}
</body>
</html>
```

Fig. 2.10 Plantilla main.html

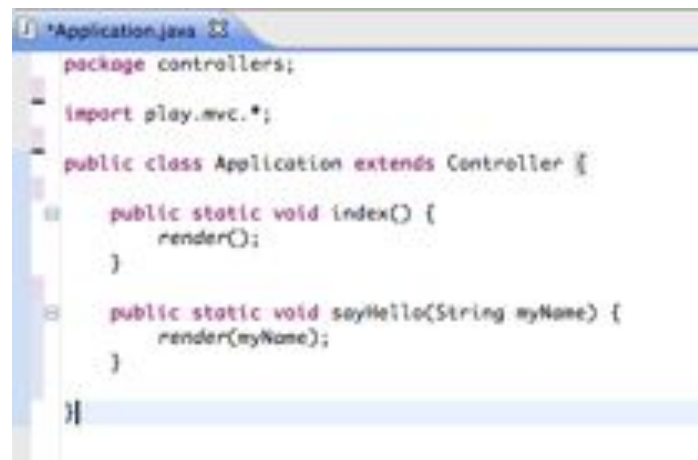
Por último, el tag `#{doLayout /}` indica el lugar en el que se insertará el contenido de index.html.

2.4 Tutorial “Hola Mundo” de Play Framework

Para familiarizarnos con el entorno de Play presentamos un sencillo tutorial en el que se presentan las principales características del marco de trabajo.

Para empezar nuestra aplicación diseñaremos un formulario en el que introducir nuestro nombre.

Como paso previo crearemos un nuevo *ACTION* en el *controller* Application, al que llamaremos `sayHello()`:



```
Application.java 33
package controllers;
import play.mvc.*;
public class Application extends Controller {
    public static void index() {
        render();
    }
    public static void sayHello(String myName) {
        render(myName);
    }
}
```

Fig. 2.11 Controller Application.java modificado

Hemos declarado el parámetro `myName` en el método *ACTION*, de manera que automáticamente se rellene con el valor del parámetro HTTP `myName`. Llamamos a `render()` para mostrar la plantilla y pasamos la variable a `render()`, para que esté disponible desde la plantilla.

Editamos la plantilla `index.html` para dejarla de la siguiente manera:



```
index.html 33
#{extends 'main.html' /}
#{set title:'Home' /}

<form action="@{Application.sayHello()}" method="GET">
  <input type="text" name="myName" />
  <input type="submit" value="Say hello!" />
</form>
```

Fig. 2.12 Plantilla index.html

Además, debemos crear la nueva plantilla, en la que mostraremos el mensaje de bienvenida. Crearemos `sayHello.html` en la carpeta `Views/Application`:



```
*sayHello.html
#{extends 'main.html' /}
#{set title:'Home' /}

<h1>Hello ${myName ?: 'guest'}!</h1>

<a href="@{Application.index()}">Back to form</a>
```

Fig. 2.13 Plantilla sayHello.html

Abriendo un navegador y entrando en el puerto adecuado de la red local <http://localhost:9000> encontraremos la plantilla creada:

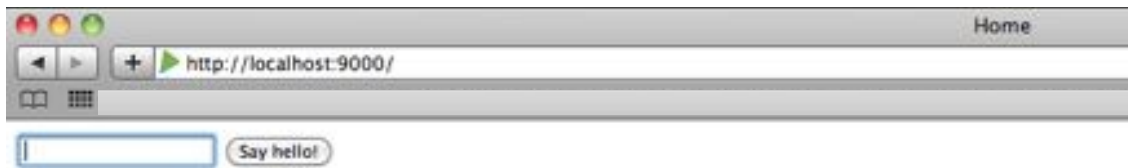


Fig. 2.14 Aplicación creada en *localhost*

Ya tenemos nuestro primer tutorial Play creado.

CAPÍTULO 3. DESARROLLO DE APLICACIONES ANDROID

3.1 Instalación del entorno de desarrollo: Eclipse + Android

Para el desarrollo de la aplicación Android utilizaremos el entorno de programación Eclipse, de libre distribución.

Debemos instalar el Kit de Desarrollo de Software (SDK) de Android en Eclipse, que es el conjunto de herramientas que nos permitirá crear aplicaciones para el sistema operativo deseado.

Para la configuración del SDK Android debemos descargar la última actualización de éste, asegurarnos que ya tenemos el Java Development Kit e instalar el plug-in para Eclipse (ADT).

Para descargar el ADT, deberemos ir a Help -> Install New Software. Haremos click en Add, y en la ventana de diálogo deberemos nombrar el plugin (ADT Plugin) y en Location, escoger la URL en la que se encuentra (<https://dl-ssl.google.com/android/eclipse>).

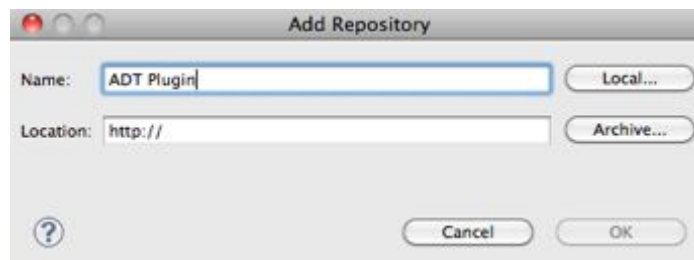


Fig. 3.1 Descarga e instalación de ADT Plugin

En la siguiente ventana, se hace clic en el checkbox y se pasa a la siguiente ventana, donde seleccionaremos las herramientas a descargar. Finalmente, aceptamos la licencia. Reiniciamos Eclipse.

Para configurar el ADT Plugin, vamos a *Window -> Preferences* para abrir el Panel de preferencias. Seleccionamos Android, y en SDK Location buscamos el directorio donde se ha guardado el SDK. Aplicamos y aceptamos.

Una vez tenemos instalado y configurado el ADT, descargamos los componentes. Para ello, haremos *Window -> Android SDK and AVD manager*, vamos a la pestaña Available packages, seleccionamos los paquetes disponibles e instalamos.

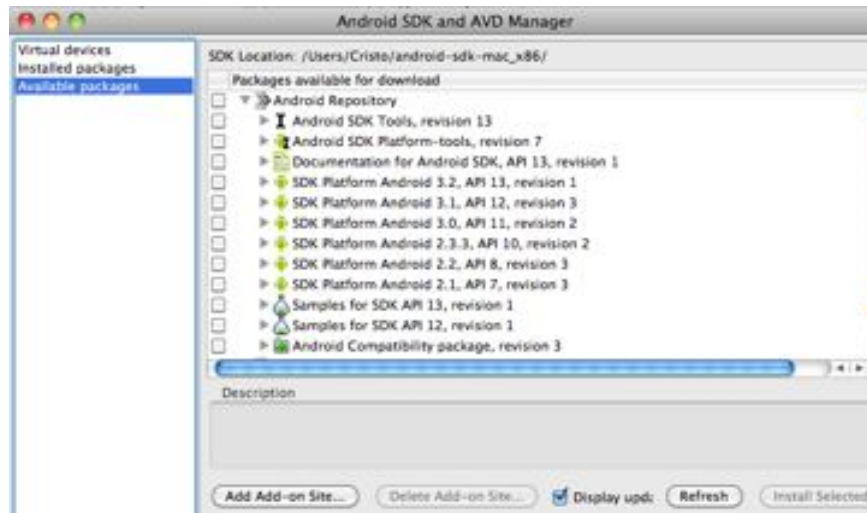


Fig. 3.2 Instalación de los paquetes Android

Una vez llegados a este punto ya podemos empezar a desarrollar en Android con Eclipse.

3.2 Creación de un terminal Android virtual (AVD)

La aplicación Android estará en manos del usuario, de manera que nos será muy útil ver en cada momento que es lo que verá quien utilice el programa. Así, tenemos la posibilidad de crear un terminal Android virtual (AVD) de manera que visualicemos los progresos de nuestro programa en un emulador de teléfono móvil. Para crear un nuevo AVD seleccionaremos *Window* → *Android SDK and AVD Manager*. Tras seleccionar *Virtual Devices*, clicaremos en *New*.

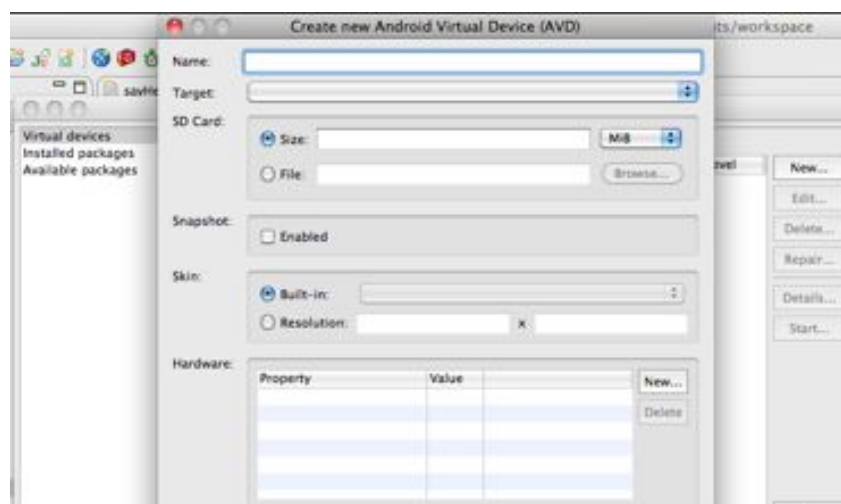


Fig. 3.3 Creación de un nuevo AVD

En la ventana de diálogo daremos un nombre al AVD y escogeremos la plataforma sobre la que queremos arrancar la aplicación (target). Finalizamos mediante *Create AVD*.

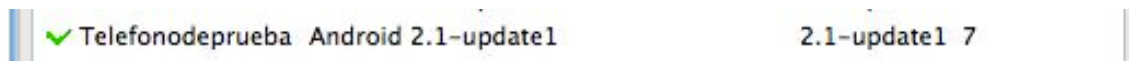


Fig. 3.4 AVD Versión 2.1 creado

3.3 Estructura de un proyecto Android

Para comprender como se construye una aplicación Android veremos cómo están organizados los proyectos.

Cuando se crea un nuevo proyecto Android en Eclipse, se generan automáticamente las carpetas necesarias para poder generar más tarde la aplicación. En la figura 3.6 podemos apreciar la estructura creada.

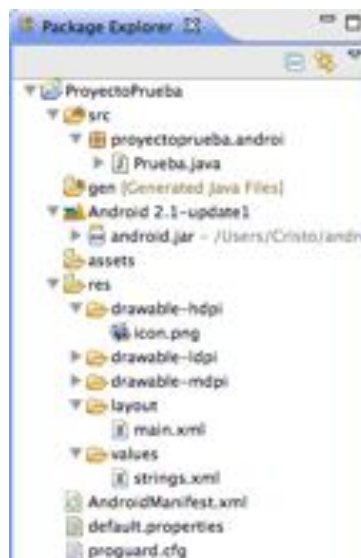


Fig. 3.5 Estructura de proyecto Android

3.3.1 Carpeta src

Esta carpeta contiene todo el código fuente de la aplicación: clases principales, auxiliares, actividades, servicios...

Estas clases se agruparán en paquetes, que harán más comprensible y visualmente atractiva la estructura.

3.3.2 Carpeta gen

La carpeta gen es generada automáticamente al compilar el proyecto y contiene la clase R.java, considerada nexa de unión entre los recursos y los XML.

3.3.3 Carpeta res

Esta carpeta contiene los recursos necesarios para el proyecto: imágenes, vídeos, sonidos, cadenas de texto, vistas, etc. Los diferentes recursos se distribuyen en las subcarpetas:

- Drawable: contiene las imágenes de la aplicación. También se encuentra subdividida en carpetas, según la resolución de los recursos.
- Layout: contiene los ficheros que definen las diferentes vistas/pantallas de la interfaz gráfica de la aplicación.
- Anim: contiene las animaciones a utilizar en la aplicación.
- Values: contiene recursos de la aplicación (cadenas de texto, estilos,...)
- Xml: contiene los ficheros XML utilizados en la aplicación.
- Raw: contiene recursos adicionales (formato diferente a XML) que no se incluyen en el resto de carpetas.

3.3.4 Carpeta assets

Contiene todos los demás ficheros auxiliares necesarios para la aplicación, como ficheros de configuración, de datos, etc.

La diferencia entre los recursos incluidos en la carpeta /res/raw/ y los incluidos en la carpeta /assets/ es que para los primeros se generará un ID en la clase R y se deberá acceder a ellos con los diferentes métodos de acceso a recursos. Sin embargo, para los segundos no se generarán ID y se podrá acceder a ellos por su ruta como a cualquier otro fichero del sistema. Se usará uno u otro según las necesidades de la aplicación.

3.3.5 Librerías

Las librerías nos dan toda la funcionalidad del sistema. Éstas dependen del SDK con el que estemos trabajando. Siempre y cuando lo necesitemos para nuestra aplicación, intentaremos estar actualizados en este tema.

3.3.6 Fichero AndroidManifest.xml

Contiene la definición en XML de los aspectos principales de la aplicación (nombre, versión, icono, pantallas, mensajes, permisos...).

- Package: situación de los ficheros a ejecutar.
- Uses-permission: definición de los permisos que necesitamos adquirir para nuestra aplicación.
- Uses-library: librerías a utilizar.

- Activity: permite declarar una actividad. Todas las actividades deben estar especificadas en este fichero.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="proyecto.prueba.android"
    android:versionCode="1"
    android:versionName="1.0">

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Prueba"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Fig. 3.6 Fichero AndroidManifest.xml

3.4 Activities

Las Activities son el motor fundamental de cualquier aplicación Android.

Una Activity es una pieza de código ejecutable que siendo iniciada por el usuario o por el sistema operativo, se ejecuta mientras es necesitada.

La característica principal de un Activity es que puede interactuar con el usuario, aunque también puede pedir datos o servicios de otras activities por medio de *queries* o *Intents*.

La mayor parte del código que se escriba para Android se ejecutará en el contexto de una Activity. En la mayoría de los casos cada Activity corresponde a una vista (*layout*). Esto quiere decir que por cada pantalla debe crearse un Activity, aunque no es requisito obligatorio vincular la parte visual a un Activity. Esto nos permite crear Activities que se ejecuten en segundo plano, sin necesidad de mostrar ninguna interfaz gráfica para el usuario.

Existen diferentes clases que extienden de la clase Activity, como pueden ser:

- MapActivity: encapsula toda la funcionalidad para mostrar un *MapView*.
- ListActivity: ayuda a mostrar una lista de elementos y soporta la selección de los elementos de la lista.
- TabActivity: permite mostrar múltiples Activities o Vistas dentro de una pantalla utilizando *Tabs* para cambiar entre ellas.

3.5 Services

Un Service es una clase que se ejecuta en segundo plano y puede actualizar contenidos, lanzar Intents (interactuar con otras Activities), notificaciones...

Los Services tienen dos características esenciales:

- Tienen la facilidad de avisar al sistema cuando algo quiere hacerse en segundo plano (aún cuando no se necesita interactuar con el usuario). Esto corresponde a llamar a *Context.startService()*, que pregunta al sistema para ser programado para dar el servicio hasta que alguien o algo lo pare de manera explícita.
- Expone algunas de sus funcionalidades a otras aplicaciones, permitiendo conexiones largas con el servicio para actuar con él.

Debido a que el Service por sí mismo es muy simple, se puede interactuar con él de maneras muy diferentes: desde tratarlo como un simple objeto de Java, hasta intentar que sea proveedor de una interfaz totalmente remota.

CAPÍTULO 4. DISEÑO DE LA APLICACIÓN

La aplicación a desarrollar se basa en un sistema de alarmas capaz de avisar al usuario que inicie sesión de las dosis de medicamentos a tomar en cada momento del día.

Como se ha comentado con anterioridad, la aplicación constará de dos bloques bien diferenciados.

El primer bloque es el servidor web en sí, desarrollado con tecnología Play Framework. El servidor será el encargado de almacenar los datos tanto de usuarios como de medicamentos y deberá estar actualizado a cada instante.

En el segundo bloque encontraremos la aplicación para dispositivos móviles Android, de sencilla interfaz y con una misión muy clara: avisar al paciente de forma que tome las dosis adecuadas sin saltarse ninguna.

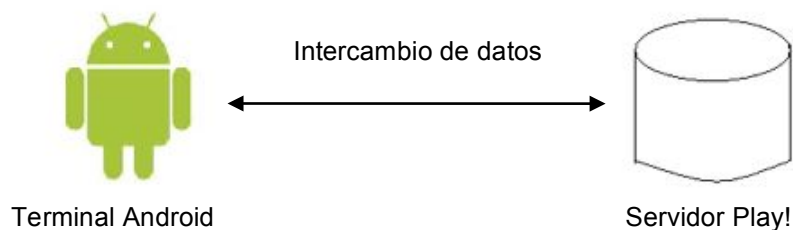


Fig. 4.1 Diferenciación por bloques del sistema

4.1 Diseño del servidor web

4.1.1 Aspectos generales

El servidor web estará alojado en el puerto 9000 de la red local del ordenador que utilizaremos como máquina de servicios.

El servidor se iniciará con un paciente y una dosis de ejemplo y esperará a una petición desde el terminal móvil para empezar a interactuar con el sistema.

Si la petición desde el terminal pide una lista de usuarios o medicamentos, se llamará a una función del servidor capaz de listar los elementos demandados.

El listado de los datos se realizará en formato XML y quedará guardado en una ruta a configurar en el programa.

Si la petición desde el terminal trata de almacenar nuevos datos, modificarlos o eliminarlos, se llamará a una función capaz de almacenar nueva información. La información recibida será en formato XML y el almacenamiento de los datos se realizará en un vector del tipo correspondiente a los usuarios o a las dosis.

Se podrán listar dos tipos de archivo XML, uno que contenga la información de todos los usuarios y otro con los datos correspondientes a todas las dosis del usuario que tenga iniciada la sesión. Así, trabajaremos con dos ficheros XML diferentes.

Para cada usuario, almacenaremos su nombre, su contraseña de acceso al sistema, su número de identificación único y si ha iniciado sesión.

En cambio, cada dosis tendrá el nombre del medicamento, la hora a la que se ha de avisar al usuario, los días de la semana en los que se ha de tomar la dosis, el número de identificación del usuario al que pertenece la dosis y el color con el que debe ser representada.

El usuario no podrá entrar en el servidor desde un navegador para realizar tareas de modificación o eliminación de datos en ningún momento. Así el usuario será totalmente ajeno a la base de datos, pudiendo interactuar con la base de datos solamente desde el terminal móvil.

4.1.2 Funcionalidades

Para el correcto funcionamiento del sistema necesitaremos que el servidor cumpla una serie de requisitos:

- Almacenamiento de usuarios y/o medicamentos.
- Modificación y/o eliminación de ciertos medicamentos.
- Interpretación de datos XML que provengan del terminal Android para almacenarlos y/o modificarlos (ver apartado *4.3 Diseño del intercambio de datos. Parsers*).
- Capacidad de listar en un fichero XML los datos de usuarios y/o medicamentos para que el usuario Android pueda leerlos.

4.2 Diseño de la aplicación Android

4.2.1 Aspectos generales

La aplicación para el terminal móvil deberá ser muy intuitiva y sencilla de manejar para el usuario.

En un primer término el usuario deberá registrarse en el sistema, indicando su nombre de usuario y su contraseña.

El usuario sólo deberá registrarse la primera vez que entre en la aplicación. Una vez registrado, iniciará sesión de manera automática y entrará en su pantalla de bienvenida, HOME.

HOME dispondrá de la información más útil para el usuario: las dosis previstas para el día en el que se encuentre. Además, el usuario podrá visualizar las dosis de una manera más intuitiva, utilizando el Calendario.

La primera vez que inicie sesión, el usuario deberá introducir las dosis que desee registrar en el sistema. Para esto, desde HOME, el usuario deberá moverse a la pantalla de introducción de datos en el sistema. Una vez introducida cada dosis, el usuario podrá ver una lista con las tomas almacenadas en la aplicación.

Pulsando sobre cada elemento de la lista, el usuario podrá modificar o eliminar dicha dosis, así como añadir una nueva.

De manera automática y sin necesidad de estar en la aplicación, el terminal avisará de las dosis a la hora prevista por medio de un Android Service, que se ejecutará en paralelo desde el momento en el que se inicia sesión en la aplicación. Los avisos se realizarán con notificaciones que lanzará el Service.

4.2.2 Funcionalidades

Para el correcto funcionamiento del sistema necesitaremos que la aplicación Android cumpla una serie de requisitos:

- Facilidad de manejo.
- Diseño muy visual.
- Capacidad de enviar peticiones HTTP al servidor para pedir tanto las listas de usuarios como las de medicamentos.
- Capacidad de analizar la información que se obtiene del servidor (*parsing*, apartado 4.3 *Diseño del intercambio de datos. Parsers*).
- Capacidad de enviar modificaciones o acciones a realizar en la base de datos mediante cliente HTTP.
- Capacidad de registro de diferentes usuarios.
- Indicación de las dosis a tomar mediante alarmas.
- Visualización de las dosis en modo calendario.

4.3 Diseño del intercambio de datos. Parsers.

El parser o procesador de XML es una herramienta indispensable en cualquier aplicación que trabaje con ficheros XML.

A través del parser no sólo podemos comprobar si nuestro documento está bien estructurado, sino que también podemos incorporarlos a nuestras

aplicaciones, de manera que estas puedan manipular documentos XML y trabajar con ellos. Existen multitud de parsers XML y, dependiendo de nuestra aplicación, elegiremos uno u otro.

De manera que podamos profundizar un poco más en el campo de los parsers, escogeremos dos parsers diferentes para nuestra aplicación.

Así, utilizaremos el parser DOM (Document Object Model) en el servidor web, y el parser SAX (Simple API for XML) en la aplicación Android.

4.3.1 Parsing XML en Play. DOM Parser.

4.3.1.1 *Introducción a DOM*

En el extremo del servidor utilizaremos el DOM Parser. Este es un conjunto de interfaces para describir una estructura abstracta para un documento XML.

Con DOM se puede modificar el contenido, la estructura y el estilo de los documentos. Todas estas funciones se realizan mediante llamadas a funciones o procedimientos que permitan acceder, cambiar, borrar o añadir nodos de información de los documentos XML.

Cuando nos referimos a interfaz al hablar de DOM (o más adelante, de SAX), no nos estamos refiriendo a interfaz gráfica, sino a interfaz de aplicaciones. Una interfaz es un dispositivo que permite comunicar dos sistemas que no hablan el mismo lenguaje.

DOM interpreta el fichero XML como una estructura de árbol, identificando cada *tag* del fichero como un nodo distinto, pudiendo diferenciar entre un nodo principal y sus 'hijos'.

4.3.1.2 *Cómo programar un DOM Parser*

El primer paso para crear un parser DOM es crear una instancia nueva de DocumentBuilderFactory e inicializar un Document:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
Document document = null;
```

A continuación, creamos el documento XML, declarando un contenedor en el que tendremos el texto del fichero:

```
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    document = builder.parse(request.body);
} catch (Exception e) {
    Logger.error(e.getMessage());
}
```

En el siguiente paso, comenzamos a identificar los nodos del documento:

```
Element NodoPrincipal = document.getDocumentElement();
Node Nodo1 = XPath.selectNode("nodo1",Nodo1);
String Nodo2 = XPath.selectText("nodo2",Nodo1);
String Nodo3 = XPath.selectText("nodo3",Nodo1);
```

Y así, extraeremos la información relativa a cada nodo 'hijo' del nodo principal.

Cómo último paso, sólo nos queda guardar la información extraída en nuestra aplicación (por ejemplo, en un Objeto del tipo que necesitemos):

```
Objeto.Nodo1 = Nodo1;
Objeto.save();
```

4.3.2 Parsing XML en Android. SAX Parser

4.3.2.1 Introducción a SAX

El Simple API for XML (SAX) es una interfaz simple para aplicaciones XML. Simple e intuitiva, es muy popular porque se utiliza en situaciones en las que los archivos XML están en una forma estructuralmente similar a la que se desea obtener.

El funcionamiento de un SAX es el siguiente: se comienza a leer el documento, se detectan los eventos de *parsing* (comienzo o finales de elemento), la aplicación procesa la parte leída y, por último, se reutiliza la memoria y se vuelve a leer hasta un nuevo evento.

SAX en ningún momento lee el documento completo, sino que cada vez que acaba con un nodo, guarda la información extraída y se olvida de él.

4.3.2.2 Cómo programar un SAX Parser

El SAX parser se ejecutará en el momento en el que la aplicación llame a un *handler* (manejador), que programaremos como una clase Java en la que tendremos los diferentes eventos que deberán ejecutarse al leer el documento XML.

Debemos tener en cuenta que en nuestra aplicación, el fichero XML se extraerá de una URL. Así comenzaremos por crear la URL de la que conseguiremos el fichero:

```
try {
    URL url = new URL("http://aqui_pondremos_nuestra_url");
```

A continuación, declararemos nuestro SAX Parser:

```
SAXParserFactory spf = SAXParserFactory.newInstance();
```

```
SAXParser sp = spf.newSAXParser();
```

Seguidamente, crearemos el lector de XML:

```
/* Get the XMLReader of the SAXParser we created. */  
XMLReader xr = sp.getXMLReader();
```

En este momento, debemos declarar el *handler* y llamarlo, aplicándolo al documento XML:

```
HandlerEjemplo miHandlerEjemplo = new HandlerEjemplo();  
xr.setContentHandler(miHandlerEjemplo);
```

Tras esto, analizaremos el documento que proviene de la URL definida:

```
xr.parse(new InputSource(url.openStream()));
```

Y finalmente, extraeremos la información que nos da el *handler*:

```
ParsedExampleDataSet parsedExampleDataSet = miHandlerEjemplo.getParsedData();  
}
```

Por último, capturamos y mostramos los errores que hayan podido surgir:

```
catch (Exception e) {  
Log.d("Error: " + e.getMessage());  
}
```

Ahora podemos guardar la información que proviene del *handler* en una lista del tipo Objeto que necesitamos,

```
List<Objeto> objetos = miHandlerEjemplo.getObjetos();
```

El *handler* al que llamamos para realizar el parseo se encuentra en una clase Java que contiene todos los eventos que han de ejecutarse.

En este *handler*, crearemos una máquina de estados para diferenciar en qué tag nos encontramos.

```
private enum Estado { E1, E2, E3, E4, NADA};  
private Estado estado = Estado.NADA;
```

Para iniciar la máquina de estados, la pondremos en NADA, que será el estado en el que nos encontremos cuando estemos entre tags, para no capturar espacios ni tabulaciones.

Tras esto, construiremos el vector sobre el que guardaremos la información de cada elemento.

```
public Vector<Objeto> getObjeto() {  
return objetos;  
}
```

Finalmente, solo nos quedará sobrescribir los métodos correspondientes al parser SAX. Estos son `startDocument()`, `endDocument()`, `startElement()`, `endElement()` y `characters()`.

Los métodos `startDocument` y `endDocument` hacen referencia a qué debe hacer el *parser* cuando empieza y acaba el documento.

Los métodos `startElement` y `endElement` hacen referencia a qué debe hacer el parser cuando empieza y acaba cada uno de los elementos del XML.

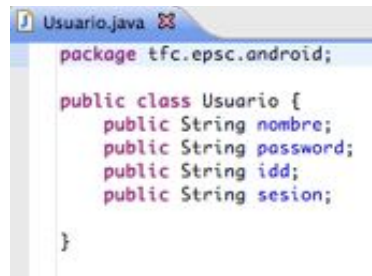
En el método `startElement` actualizaremos la máquina de estados. Por último, el método `characters` extrae la información en forma de `String` del tag en el que nos encontremos.

CAPÍTULO 5. IMPLEMENTACIÓN DE LA APLICACIÓN

5.1 Consideraciones generales

En nuestra aplicación utilizaremos dos Objetos diferentes: Usuario.java y Pastilla.java.

Usuario.java contendrá 4 campos definidos como String (nombre, contraseña, id y sesión).



```
Usuario.java
package tfc.epsc.android;

public class Usuario {
    public String nombre;
    public String password;
    public String idd;
    public String sesion;
}
```

Fig. 5.1 Clase Usuario.java

Pastilla.java contendrá otros 5 campos definidos como String (nombre, hora, día, userID y color).



```
Pastilla.java
package tfc.epsc.android;

public class Pastilla {
    public String nombre;
    public String hora;
    public String dia;
    public String userID;
    public String color;
}
```

Fig. 5.2 Clase Pastilla.java

La aplicación tendrá dos tipos de archivos XML, uno para usuarios y otro para pastillas. El archivo XML de usuarios tendrá la apariencia mostrada en la Fig. 5.3.



```
<usuarios>
  <usuario>
    <nombre>car los</nombre>
    <password>ooahfr876</password>
    <ident>1565</ident>
    <sesionno</sesionno>
  </usuario>
  <usuario>
    <nombre>luisantonio</nombre>
    <password>123jstio</password>
    <ident>3754</ident>
    <sesionno</sesionno>
  </usuario>
  <usuario>
    <nombre>nombre</nombre>
    <password>password</password>
    <ident>ident</ident>
    <sesionno</sesionno>
  </usuario>
</usuarios>
```

Fig. 5.3 Archivo XML de Usuario

Por su parte, el archivo XML de pastillas tendrá la apariencia representada en la Fig.5.4.



```
<pastillas>
  <pastilla>
    <nombre>Pastilla1</nombre>
    <hora>21:20</hora>
    <dia>TueFri</dia>
    <userid>3754</userid>
    <color>Rojo</color>
  </pastilla>
  <pastilla>
    <nombre>Pastilla2</nombre>
    <hora>00:16</hora>
    <dia>FriSat</dia>
    <userid>3754</userid>
    <color>Negro</color>
  </pastilla>
</pastillas>
```

Fig. 5.4 Archivo XML de Pastilla

De esta manera, cada dosis será relacionada para un usuario o para otro dependiendo de si su identificador coincide con el del usuario o no.

5.2 Implementación del servidor Play!

Para la implementación del servidor utilizaremos una clase Controller, Application.java, y las dos clases definidas en el apartado anterior (Usuario.java y Pastilla.java) como Models.

Además, definiremos el fichero routes, de manera que tengamos un punto de entrada a la base de datos para cada una de las funciones que definiremos en

la clase `Application.java`. Estas funciones serán `listXml`, `listXmlUser`, `saveXml` y `saveXmlUser`.

En la carpeta `Views`, definiremos las diferentes vistas que tendrán cada una de las funciones que consisten en listar objetos.

5.2.1 Controller `Application.java`

El Controller `Application` será el principal motor del servidor e implementará las diferentes funciones que necesitaremos para manejar la base de datos.

Estas funciones serán 4: `listXml`, `saveXml`, `listXmlUsers` y `saveUser`. Las dos primeras funciones harán referencia a listar y guardar pastillas, así como las dos segundas implementarán las funciones de listar y guardar usuarios.

La función `listXml` se encargará de mirar en la base de datos si existe alguna pastilla en la base de datos. Si no existiera ninguna, se crearía una de ejemplo. En el caso de que existieran pastillas, buscaría las que contuviesen el mismo identificador que el usuario que ha pedido la lista y crearía una lista de pastillas a recoger por el fichero `listXml.xml`.

La función `saveXml` implementará el DOM parser. Éste recogerá los datos relativos a nombre, hora, día, identificador y color de la pastilla.

```
Element pastillaNode = document.getDocumentElement();
Node nombreNode = XPath.selectNode("pastilla", pastillaNode);
String nombreName = XPath.selectText("nombre", nombreNode);
String horaName = XPath.selectText("hora", nombreNode);
String diaName = XPath.selectText("dia", nombreNode);
String userIDName = XPath.selectText("userid", nombreNode);
String colorName = XPath.selectText("color", nombreNode);
```

Dependiendo de si queremos añadir, eliminar o modificar una dosis realizaremos una acción u otra. El fichero XML que recibamos contendrá en el tag *nombre* la acción a realizar. Así, decodificaremos el tag, separando la acción del propio nombre.

Para añadir nuevas dosis crearemos un nuevo objeto `Pastilla` y lo guardaremos en la base de datos.

```
Pastilla pastilla = new Pastilla(nombreName, horaName, diaName, userIDName, colorName);
pastilla.save();
```

En caso de querer eliminar una dosis, deberemos buscar en la base de datos el medicamento que deseamos suprimir y deberemos borrar el objeto deseado.

```
List<Pastilla> eliminables = Pastilla.find("byNombre", nombre).fetch();
Pastilla eliminar = eliminables.get(0);
```

Por último, si lo que necesitamos es modificar una dosis ya existente, seguiremos los mismos pasos que para eliminarla. La buscaremos por nombre y modificaremos los campos necesarios.

```
List<Pastilla> modificables = Pastilla.find("byNombre", nombreantiguo).fetch();
Pastilla modificar = modificables.get(0);
modificar.delete();
modificar = new Pastilla(nombrenuevo,horaName,diaName,userIDName,colorName);
modificar.nombre=nombrenuevo;
modificar.dia=diaName;
modificar.hora=horaName;
modificar.userID=userIDName;
modificar.color = colorName;
modificar.save();
```

La función listXmlUser consultará en la base de datos si existen usuarios. Si es así, creará una lista con éstos y los pasará al fichero listXmlUsers.xml.

La función saveXmlUser también implementará el DOM parser. Éste recogerá los datos relativos a nombre, contraseña, identificador y sesión.

```
Element usuarioNode = document.getDocumentElement();
Node nombreNode = XPath.selectNode("usuario", usuarioNode);
String nombreName = XPath.selectText("nombre",nombreNode);
String passwordName = XPath.selectText("password",nombreNode);
String idName = XPath.selectText("ident", nombreNode);
String sesionName = XPath.selectText("sesion",nombreNode);
```

Dependiendo de si queremos añadir un nuevo usuario o iniciar o cerrar una sesión realizaremos una acción u otra. El fichero XML que recibamos contendrá en el tag *nombre* la acción a realizar. Así, decodificaremos el tag, separando la acción del propio nombre.

Si simplemente queremos añadir un usuario, crearemos un objeto Usuario y lo añadiremos a la base de datos.

```
Usuario user = new Usuario(nombreName,passwordName,idName,sesionName);
user.save();
```

Si queremos iniciar o cerrar una sesión, buscaremos en la base de datos el usuario a modificar y cambiaremos el estado de su sesión.

```
List<Usuario> modificables = Usuario.find("byNombre",nombre).fetch();
Usuario modificar = modificables.get(0);
modificar.delete();
modificar = new Usuario(nombre,passwordName,idName,sesionName);
modificar.save();
```

5.2.2 Models Usuario.java y Pastilla.java

Se utilizarán los modelos de Usuario y pastilla, los cuales tendrán los parámetros necesarios para la creación de objetos que sigan su estructura.

El modelo Usuario estará definido tal que

```
@Required
public String nombre;
@Required
public String password;
@Required
public String idd;
@Required
public String sesion;

public Usuario(String nombreU, String passwordU, String idU, String sesionU) {
this.nombre = nombreU;
this.password = passwordU;
this.idd = idU;
this.sesion = sesionU;
}
```

Y el modelo Pastilla se definirá por estos parámetros

```
@Required
public String nombre;
@Required
public String hora;
@Required
public String dia;
@Required
public String userID;
@Required
public String color;

public Pastilla(String nombreName, String horaName, String diaName,String userIDName,
String colorName) {
this.nombre = nombreName;
this.hora = horaName;
this.dia = diaName;
this.userID = userIDName;
this.color = colorName;
}
```

5.2.3 Fichero Routes

En este fichero registraremos los diferentes puntos de entrada a las funciones de nuestra aplicación Play.

GET	/xml/pastillas	Application.listXml(format:'xml')
GET	/xml/usuarios	Application.listXmlUsers(format:'xml')
POST	/guardar	Application.saveXml
POST	/guardaruser	Application.saveUser

Dependiendo de la función a la que se llame desde Android, se ejecutará una de las funciones u otra.

5.2.4 Carpeta Views

En esta carpeta tendremos las vistas de los dos ficheros XML que crearemos.

El fichero listXml.xml contendrá la estructura del archivo XML correspondiente a la lista de medicamentos.

```
<pastillas>
  #{list pastillas, as:'pastilla'}
  <pastilla>
    <nombre>${pastilla.nombre}</nombre>
    <hora>${pastilla.hora}</hora>
    <dia>${pastilla.dia}</dia>
    <userid>${pastilla.userID}</userid>
    <color>${pastilla.color}</color>
  </pastilla>
#{/list}
</pastillas>
```

El fichero listXmlUser.xml tendrá un aspecto parecido al anterior, pero con los tags correspondientes a Usuario.

```
<usuarios>
  #{list usuarios, as:'usuario'}
  <usuario>
    <nombre>${usuario.nombre}</nombre>
    <password>${usuario.password}</password>
    <ident>${usuario.idd}</ident>
    <sesion>${usuario.sesion}</sesion>
  </usuario>
#{/list}
</usuarios>
```

5.3 Implementación de la aplicación en Android

En nuestra aplicación para Android tendremos tantas Activities como pantallas diferentes. También tendremos las clases de Objeto Usuario.java, Pastilla.java y los *handlers* HandlerPastillas.java y HandlerUsuarios.java.

Además, programaremos el Android Service Servicio, necesario para que, de manera paralela a la aplicación, puedan aparecer las alarmas adecuadas en cada momento.

También incluiremos la clase Calendario.java, encargada de dibujar un calendario en el que aparezcan las diferentes tomas semanales.

Programaremos el fichero AndroidManifest.xml de manera que contenga todas las Activities y Services de la aplicación.

5.3.1 Objetos Usuario.java y Pastilla.java

Tal y como se ha avanzado en apartados anteriores, el objeto Usuario contendrá cuatro Strings y el objeto Pastilla, cinco.

La clase Usuario tendrá cuatro parámetros que serán nombre, password, idd y sesión, a los que se asignará el nombre del usuario, su contraseña, su identificador y un parámetro que indique si ha iniciado sesión.

```
public class Usuario {
    public String nombre;
    public String password;
    public String idd;
    public String sesion;
}
```

Por su parte, los parámetros de la clase Pastilla serán el nombre del medicamento, la hora y el día de la dosis, el identificador del usuario al que pertenecen y el color seleccionado para su representación gráfica.

```
public class Pastilla {
    public String nombre;
    public String hora;
    public String dia;
    public String userID;
    public String color;
}
```

5.3.2 Handlers HandlerPastillas.java y HandlerUsuarios.java

Nuestra aplicación constará de dos Handlers, uno para cada tipo de XML que recibamos del servidor.

El HandlerUsuarios será el encargado de analizar el contenido del XML referente a usuarios e identificar todos los campos con el adecuado tag.

Crearemos una máquina de estados para conocer en qué tag nos encontramos al inicio de cada elemento y sobrescribiremos los métodos de la clase Handler.

Para empezar, creamos la máquina de estados y ponemos el estado inicial en *NADA*.

```
private enum Estado { NOMBRE, PASSWORD, IDENT, SESION, NADA};
private Estado estado = Estado.NADA;
```

También creamos el constructor que devolverá los usuarios recogidos:

```
public Vector<Usuario> getUsuario() {
    return usuarios;}
}
```

Ahora, sobrescribimos los métodos `startDocument()`, `endDocument()`, `startElement()`, `endElement()` y `characters()`.

```
public void startDocument() throws SAXException {
    usuarios = new Vector<Usuario>();
}

public void endDocument() throws SAXException {
    // Nada que hacer
}

public void startElement(String namespaceURI, String localName,
    String qName, Attributes atts) throws SAXException {
    if (localName.equals("nombre")) {
        estado = Estado.NOMBRE;
    }else if (localName.equals("password")) {
        estado = Estado.PASSWORD;
    }else if (localName.equals("ident")) {
        estado = Estado.IDENT;
    }else if (localName.equals("sesion")) {
        estado = Estado.SESION;
    }else if (localName.equals("usuario")) {
        tmp = new Usuario(); //si llegamos al tag usuario, estamos en el comienzo, por tanto creamos
    }
}

public void endElement(String namespaceURI, String localName, String qName)
    throws SAXException {
    estado = Estado.NADA;
    if (localName.equals("usuario")) {
        usuarios.add(tmp);
    }
}

public void characters(char ch[], int start, int length) {
    String s = new String(ch,start,length);
    switch(estado) {
        case NOMBRE:
            tmp.nombre = s;
            break;
        case PASSWORD:
            tmp.password = s;
            break;
        case IDENT:
            tmp.idd = s;
            break;
        case SESION:
            tmp.sesion = s;
            break;
    }
}
```

El `HandlerPastillas` será el encargado de analizar el contenido del XML referente a pastillas e identificar todos los campos con el adecuado tag. Crearemos una máquina de estados para conocer en que tag nos encontramos al inicio de cada elemento y sobrescribiremos los métodos de la clase `Handler`.

```
private enum Estado { NOMBRE, HORA, DIA, ID, COLOR, NADA};
private Estado estado = Estado.NADA;
```

También creamos el constructor que devolverá las pastillas recogidas:

```
public Vector<Pastilla> getPastilla() {
return pastillas; }
```

Ahora, sobrescribimos los métodos `startDocument()`, `endDocument()`, `startElement()`, `endElement()` y `characters()`.

```
public void startDocument() throws SAXException {
pastillas = new Vector<Pastilla>();
}
```

```
public void endDocument() throws SAXException {
// Nada que hacer
}
```

```
public void startElement(String namespaceURI, String localName,
String qName, Attributes atts) throws SAXException {
if (localName.equals("nombre")) {
estado = Estado.NOMBRE;
}else if (localName.equals("hora")) {
estado = Estado.HORA;
}else if (localName.equals("dia")) {
estado = Estado.DIA;
}else if (localName.equals("userid")) {
estado = Estado.ID;
} else if (localName.equals("color")) {
estado = Estado.COLOR;
}else if (localName.equals("pastilla")) {
tmp = new Pastilla(); //si llegamos al tag pastilla, estamos en el comienzo, por tanto creamos
}
}
```

```
public void endElement(String namespaceURI, String localName, String qName)
throws SAXException {
estado = Estado.NADA;
if (localName.equals("pastilla")) {
pastillas.add(tmp);
}
}
```

```
public void characters(char ch[], int start, int length) {
String s = new String(ch,start,length);
switch(estado) {
case NOMBRE:
tmp.nombre = s;
break;
case HORA:
tmp.hora = s;
break;
case DIA:
tmp.dia = s;
break;
case ID:
tmp.userID = s;
break;
case COLOR:
tmp.color = s;
```

```
break;  
}  
}
```

Estos handlers serán llamados desde las Activities en cada momento que necesitemos recuperar las listas de Usuarios o Pastillas.

5.3.3 Servicio.java

De manera paralela a la Activity correspondiente, se ejecutará un Android Service, encargado de hacer saltar las alarmas adecuadas en cada instante del día correspondiente.

Servicio.java extenderá de la clase Service. Declararemos un *timer* (fijado a 60 segundos) que marcará cuando deberá ejecutarse el Service. Sobreescribiremos los métodos *onStart()* y *onDestroy()* para controlar qué sucederá cuando llamemos a estos métodos desde las Activities.

Para pedir que Servicio comience su rutina deberemos realizar un *Intent* al Servicio en el momento en el que un usuario inicie sesión.

```
Intent servicio = new Intent();  
servicio.setAction("tfc.epsc.android.Servicio");  
startService(servicio);
```

Esto llamará al método *onStart()* del Service que realizará una petición al servidor Web para que le entregue la lista de dosis de un determinado paciente. Tras esto, compararemos la hora y el día de cada dosis con la hora y el día actuales del sistema. En caso de que lleguen a coincidir se enviará una notificación a la barra de estados del terminal, que será visible desde cualquier pantalla en la que nos encontremos. La notificación que se podrá visualizar es la de la figura 5.5.

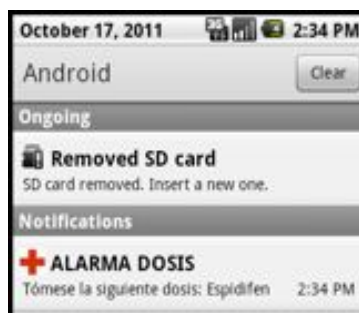


Fig.5.5 Notificación de la alarma

Cuando un usuario cierre sesión, llamaremos al método *onDestroy()* para detener el Service.

```
Intent servicio = new Intent();  
servicio.setAction("tfc.epsc.android.Servicio");  
stopService(servicio);
```

5.3.4 Activities

Las Activities son el motor de la parte visual del proyecto. Cada una de ellas tiene un layout asignado, que es lo que realmente verá el usuario.



Fig.5.6 Icono de la aplicación

5.3.4.1 *PaginaUno.java*

En esta pantalla, el usuario deberá escoger entre Crear un nuevo usuario o Iniciar sesión con un usuario ya creado haciendo clic en uno de los dos botones que se mostrarán. *PaginaUno* extiende del *layout* *paginauno.xml*.

En caso de seleccionar Crear un usuario nuevo, el sistema nos redirigirá a la Activity *CrearUser.java*. Si, por lo contrario, nuestro deseo es iniciar sesión con nuestro usuario, nos desplazaremos a *IniciarSesion*.



Fig.5.7 Pantalla *PaginaUno*

5.3.4.2 *CrearUser.java*

Esta Activity es la primera en comunicarse con el servidor. Debemos pedir a Play la lista de usuarios (y procesarla mediante el SAX parser) para que el usuario no pueda crear un nuevo usuario con un nombre ya almacenado en la base de datos. *CrearUser* extiende del *layout* *crear.xml*.

El usuario deberá introducir un nombre de usuario y una contraseña. En caso de que el nombre escogido no esté disponible, se mostrará un mensaje de error y se instará al usuario a escoger otro nombre. Si el nombre escogido es

nuevo para el sistema, se le asignará un ID al usuario y se guardará en la base de datos mediante una petición HTTP. El sistema nos redirigirá a la actividad InicioSesion.

En esta Activity también encontramos un botón de salida para volver a PaginaUno.



Fig.5.8 Pantalla CrearUser

5.3.4.3 InicioSesion.java

En la Activity InicioSesion, encontramos un formulario en el que introducir el nombre de usuario y la contraseña correspondiente.

InicioSesion extiende del *layout* iniciarsesion.xml.

La Activity pide al servidor la lista de usuarios para realizar la autenticación del usuario.

Pueden darse tras casos diferentes:

- El usuario introducido no existe en la base de datos: se informa al usuario y se vacían los campos.
- El usuario introducido existe en la base de datos pero la contraseña es incorrecta: se informa al usuario y se vacía el campo de la contraseña.
- El usuario y la contraseña que ha introducido el usuario son correctos: el sistema informa al usuario e inicia sesión.

Al iniciar sesión, el terminal envía al servidor una modificación del usuario, asignándole al parámetro *sesion* del usuario un "si". Esto permitirá al servidor conocer qué usuarios tienen la sesión iniciada en un determinado instante. Una vez se ha iniciado sesión, nos desplazamos a la Activity Home.

En esta Activity también encontramos un botón de salida para volver a PaginaUno.



Fig.5.9 Pantalla Iniciar sesion

5.3.4.4 *Home.java*

En Home encontramos la pantalla principal para el usuario. Lo primero que podrá ver serán las dosis que tiene programadas para el día actual y tras consultarlas, podrá decidir entre verlas en el formato de Calendario, visualizarlas en una lista o añadir dosis a su listado. Home contiene el *layout* home.xml.

También se ha añadido un botón para salir de Home, lo que implica cerrar la sesión del usuario actual.



Fig.5.10 Pantalla Home

5.3.4.5 *Anadir.java*

Esta Activity es la encargada de recoger los datos necesarios para añadir dosis a la lista de un determinado usuario. En esta pantalla encontramos un campo en el que escribir el nombre del medicamento, un TimePicker (widget de Android) para escoger la hora y una serie de CheckBox (widget de Android) para escoger los días en los que se ha de tomar el medicamento. También debemos escoger de una lista el color con el que queremos que se represente

dicho medicamento en la lista y en el calendario. Anadir extiende del *layout* anadir.xml.

Lo primero que hará la Activity será enviar al servidor los datos del usuario que va a añadir dosis a su lista. A continuación, se pedirá al servidor la lista de dicho usuario. Una vez tenga la lista en memoria, recorrerá dicha lista para buscar, en caso de que exista, un medicamento con el mismo nombre. Así, podemos encontrar tres casos diferentes:

- El medicamento ya se encuentra en la base de datos, pero con una hora diferente. En este caso, el sistema almacenará la dosis, pero con el mismo color que el medicamento que ya existe en la base de datos.
- El medicamento se encuentra en la base de datos con la misma hora. En este caso, el sistema avisará al usuario de que esta dosis ya existe.
- El medicamento no se encuentra en la base de datos. En este caso, se almacenará como un nuevo medicamento.

Tras guardar el medicamento, el sistema nos conduce a la Activity Lista. También tenemos un botón Home, con el que regresar a dicha actividad.



Fig.5.11 Pantalla Anadir

5.3.4.6 *Lista.java*

En la Activity *Lista.java* encontramos un listado actualizado de las dosis de medicamentos del usuario con sesión iniciada. Para cada dosis, podemos ver el nombre, la hora y los días de la semana en que está programada la toma. Además, la encontraremos representada con el color que el usuario haya escogida al almacenarla. *Lista* extiende de la clase *ListActivity* y contiene el *layout* *list.xml*.

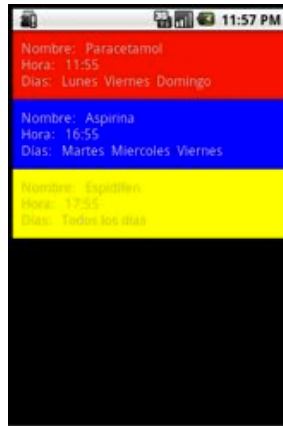


Fig.5.12 Pantalla Lista

5.3.4.7 Acciones.java

En Acciones, vemos las dos opciones que tenemos sobre una determinada dosis. Podemos eliminarla o modificarla. También tenemos la opción de crear una nueva dosis o incluso salir a Home. Acciones extiende del *layout* acciones.xml.

Para eliminar una dosis simplemente tendremos que clicar sobre el botón de suprimir. El terminal enviará los datos de la pastilla a eliminar al servidor con una cadena de caracteres insertada en un tag, el servidor reconocerá esta cadena y actuará en consecuencia.

Si lo que queremos es modificar una pastilla, el sistema nos dejará modificar su hora y sus días. Para cambiar el nombre del medicamento deberemos eliminar esta dosis y volver a crearla con el nombre deseado. Al clicar sobre el botón de edición, se habilitarán los campos Hora y Días, desaparecerán los botones de edición, eliminar y añadir para dar paso a un botón con la palabra OK. Al clicar sobre OK se enviarán los datos nuevos de la dosis a modificar al servidor y se incluirá una cadena de caracteres en uno de los tags para que el servidor sepa qué ha de hacer.

Si lo que queremos es añadir una nueva dosis, al clicar sobre añadir, la aplicación nos redirigirá a la Activity Anadir.



Fig.5.13 Pantalla Acciones

5.3.4.8 *Calendario.java*

Por último, la Activity *Calendario* es la encargada de dibujar un Calendario sobre un Canvas. Sobre la pantalla de Android podremos visualizar una cuadrícula con los días de la semana en el eje horizontal y las horas del día en el eje vertical.

Para dibujar sobre la pantalla utilizaremos el método *onDraw()* y el objeto *Paint*. Dibujaremos círculos del color seleccionado para cada medicamento en las coordenadas correctas.

La Activity pedirá la lista de medicamentos del usuario con sesión iniciada y procesará cada dosis de la siguiente manera:

- Consultará el color de la dosis y asignará dicho color al objeto *Paint*.
- Mirará la hora de la dosis y según cual sea le asignará una coordenada Y.
- Para cada día que contenga el parámetro *días* se asignará la posición X correspondiente y se pintará siguiendo las coordenadas (X,Y) y el color seleccionado.

Al clicar sobre la pantalla podremos observar una leyenda para conocer qué medicamento se relaciona con cada color.

Si queremos salir del calendario, deberemos clicar de manera prolongada sobre la pantalla.

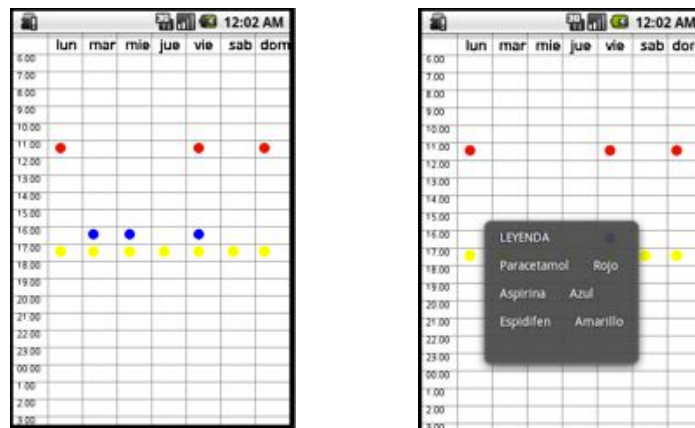


Fig.5.14 Pantalla Calendario

5.3.5 Fichero *AndroidManifest.xml*

Este fichero recoge todos los permisos que usamos en nuestra aplicación, así como detalles generales de ésta y todas las actividades y services que la conforman.

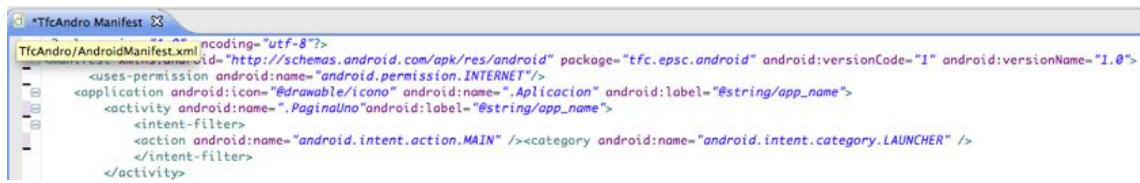


Fig.5.15 Cabecera del fichero AndroidManifest.xml

CAPÍTULO 6. CONCLUSIONES

6.1 Conclusiones finales

Este proyecto nace de la motivación de realizar una aplicación para Android, tras cursar la asignatura *Programació de Sistemes Empotrats i Mòbils* como Créditos Optativos de Ingeniería Técnica de Telecomunicación. Especialidad en Sistemas de Telecomunicación, con el profesor Juan López.

Tras consultar con Juan López sobre la posibilidad de realizar un Trabajo de Final de Carrera que comprendiera una aplicación Android y algún tema relacionado con la Teleasistencia, apareció la idea de realizar una aplicación capaz de controlar las dosis que debe tomar un determinado paciente.

El gran motor de este trabajo consistía en la investigación de cómo conseguir enlazar un dispositivo Android con una base de datos alojada en un servidor, para el que más tarde se escogería Play! Framework como la opción más adecuada.

Los capítulos 2 y 3 de este Trabajo se han desarrollado en colaboración con Ramón García García, autor del TFC *Sistema de Guiado para incendios forestales*. En ambos Trabajos se ha desarrollado la comunicación entre los sistemas Android y Play Framework mediante ficheros XML.

Tras ver los resultados del Trabajo, podemos concluir que se han cumplido los objetivos planteados desde el inicio.

Se ha desarrollado la forma de comunicación entre un terminal móvil Android y el marco de trabajo Play. De esta manera, hemos podido crear una base de datos alojada en un servidor que sea capaz de actualizarse a partir de los datos que llegan desde el terminal.

En cuanto al tipo de comunicación entre sistemas, se ha considerado válida la opción de utilizar archivos XML en los dos extremos. Así, se han implementado dos parsers (uno en cada extremo) capaces de analizar la información que provenía del extremo opuesto.

Se ha conseguido desarrollar un Servicio paralelo a la aplicación principal, capaz de actuar conjuntamente con la base de datos y aprovechar los datos de ésta para informar al usuario mediante notificaciones de la hora de toma de las dosis correspondientes.

Además se ha conseguido que el usuario pueda visualizar sus dosis de una manera muy gráfica, mediante la implementación de la clase Calendario. Dicha clase ejecuta un código de dibujo capaz de interpretar la pantalla del terminal como una cuadrícula en la que usar las coordenadas de ésta como puntos en los que pintar nuestros medicamentos. La clase Canvas nos permite dibujar multitud de formas sobre la pantalla Android de una manera sencilla e intuitiva.

Tras la realización de este proyecto concluimos que sería un sistema muy útil pensando a gran escala (a nivel de centros de salud). Los facultativos médicos serían quienes administrarían las listas de medicamentos de cada paciente y serían ellos quienes modificarían, añadirían o eliminarían dosis. Incluso podrían utilizarse *tablets* como el medio tecnológico en el que instalar el sistema. Estos *tablets* serían entregados a los pacientes, quienes no tendrían mayor complicación que la de parar las alarmas en el momento en el que saltaran.

6.2 Posibles ampliaciones

Durante el transcurso de este trabajo han ido surgiendo nuevas ideas y posibles ampliaciones, muchas de las cuales han sido incluidas en el prototipo final de la aplicación y otras tantas que se mencionan a continuación.

6.2.1 Interacción del usuario vía web

Una posible mejora sería la introducción de un módulo que capacitara al usuario para que entrara al sistema vía web y pudiera realizar las mismas acciones que realiza en el terminal, pero desde el navegador web.

6.2.2 Modo sin conexión

Otra posible ampliación sería la posibilidad de usar el sistema sin conexión de datos. De esta manera deberíamos implementar una base de datos local en el teléfono que almacenara todas las tomas que no han sido enviadas al servidor, para que en el momento en el que vuelva a haber conexión pueda volver a existir sincronización Android-Servidor.

6.2.3 Automatización del sistema mediante hardware externo

También podría estudiarse la posibilidad de añadir hardware externo para automatizar el sistema, minimizando la intervención del usuario. Esto podría conseguirse con la programación de sensores, leds y otros gadgets que mediante bluetooth intercambiasen información con el terminal móvil, actualizando así la base de datos del servidor.

6.3 Impacto medioambiental

Este proyecto consiste en una aplicación desarrollada para terminales móviles en un entorno de programación, de manera que no ha representado ningún impacto medioambiental a considerar.

CAPÍTULO 7. REFERENCIAS BIBLIOGRÁFICAS

- Páginas web

[1] Android Developers. <http://developer.android.com>

[2] Play Framework. <http://www.playframework.org/>

[3] Android Development Community. <http://www.anddev.org>

[4] Android Snippets. <http://www.androidsnippets.com/>

[5] Commonsware. <http://commonsware.com>

[6] GamingDroid. Programación de juegos para Android.
<http://www.gamingdroid.com/>

[7] Technology Blog. <http://www.ceveni.com/>

[8] Google Code. <http://code.google.com/intl/es-ES/>

CAPÍTULO 8. ANEXOS

8.1 Anexo 1: Código completo de la aplicación Play

Application.java

```

public class Application extends Controller {
    static String USUARIO_ACTUAL = "NO";
    //lista en Xml las pastillas de cierto usuario
    public static void listXml() {
        if (Pastilla.count()==0){
            Pastilla example = new Pastilla
("nombre","00.00","MonTueWedThuFriSatSun","1234","Blanco");
            example.save();
        }else{

            String idactual = USUARIO_ACTUAL;
            if(idactual=="NO"){
                return;
            }else{

                List<Pastilla> pastillas = Pastilla.find("byUserID",idactual).fetch();
                render(pastillas);
            }
        }
    }
    //lista en xml los usuarios
    public static void listXmlUsers(){
        if (Usuario.count()==0){
            Usuario user = new Usuario ("nombre","password","ident","sesion");
            user.save();
        }else{}
        List<Usuario> usuarios = Usuario.find("order by nombre, nombre").fetch();
        render(usuarios);
    }
    //guarda un usuario o sus datos
    public static void saveUser(){
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        Document document = null;
        try {
            //create xml document
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(request.body);
        } catch (Exception e) {
            Logger.error("****"+e.getMessage());
        }
        Element usuarioNode = document.getDocumentElement();
        Node nombreNode = XPath.selectNode("usuario", usuarioNode);
        String nombreName = XPath.selectText("nombre",nombreNode);
        String passwordName = XPath.selectText("password",nombreNode);
        String idName = XPath.selectText("ident", nombreNode);
        String sesionName = XPath.selectText("sesion",nombreNode);
        //si contiene CERRAR, deberemos cerrar sesión, recogiendo los nuevos datos
        if(nombreName.contains("CERRAR")){
            String nombre=nombreName.substring(6,nombreName.length());
            List<Usuario> modificables = Usuario.find("byNombre",nombre).fetch();
            Usuario cerrar = modificables.get(0);
            cerrar.delete();
            cerrar = new Usuario(nombre,passwordName,idName,sesionName);

```

```

        cerrar.save();
        USUARIO_ACTUAL="NO";
    }
    //si contiene INICIAR, deberemos iniciar sesión, colocando los nuevos datos en
    el usuario
    else if(nombreName.contains("INICIAR")){
        String nombre=nombreName.substring(7,nombreName.length());
        List<Usuario> modificables = Usuario.find("byNombre",nombre).fetch();
        Usuario modificar = modificables.get(0);
        modificar.delete();
        modificar = new Usuario(nombre,passwordName,idName,sesionName);
        modificar.save();
        //si nombreName contiene USUARIO, quiere decir que nos envian el ID del usuario
        actual
    }else if(nombreName.contains("USUARIO")){
        String user=nombreName.substring(7,nombreName.length());
        USUARIO_ACTUAL = user;
    }
    else{
        //crea nuevo usuario
        Usuario user = new Usuario(nombreName,passwordName,idName,sesionName);
        //guarda en db
        user.save();
    }
}
//función de guardar dosis
public static void saveXml(){
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    Document document = null;
    try {
        //create xml document
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.parse(request.body);
    } catch (Exception e) {
        Logger.error("****"+e.getMessage());
    }
    Element pastillaNode = document.getDocumentElement();
    Node nombreNode = XPath.selectNode("pastilla", pastillaNode);
    String nombreName = XPath.selectText("nombre",nombreNode);
    String horaName = XPath.selectText("hora",nombreNode);
    String[] time = horaName.split(":");
        String h = time[0];
        String m = time[1];

        if (h.length()==1){
            h="0"+h;
        }
        if (m.length()==1){
            m="0"+m;
        }
        horaName = (h+":"+m);

    String diaName = XPath.selectText("dia", nombreNode);
    String userIDName = XPath.selectText("userid", nombreNode);
    String colorName = XPath.selectText("color",nombreNode);
    //si contiene ELIMINAR, debemos suprimir la dosis adecuada
    if(nombreName.contains("ELIMINAR")){
        String nombre=nombreName.substring(8,nombreName.length());
        List<Pastilla> eliminables = Pastilla.find("byNombre",nombre).fetch();
        Pastilla eliminar = eliminables.get(0);

```

```

        eliminar.delete();
    }
    //si contiene MODIFICAR, cambiaremos los parámetros de la dosis indicada
    else if(nombreName.contains("MODIFICAR")){
        String nombre=nombreName.substring(9,nombreName.length());
        int inicio1 = nombre.indexOf(";");
        int inicio2 = (nombre.indexOf(";")+1);
        String nombreantiguo = nombre.substring(0, inicio1);
        String nombrenuevo=nombre.substring(inicio2);
        //modificar directamente???
        List<Pastilla> modificables = Pastilla.find("byNombre", nombreantiguo).fetch();
        Pastilla modificar = modificables.get(0);
        modificar.delete();
        modificar = new Pastilla
(nombrenuevo,horaName,diaName,userIDName,colorName);
        modificar.nombre=nombrenuevo;
        modificar.dia=diaName;
        modificar.hora=horaName;
        modificar.userID=userIDName;
        modificar.color = colorName;
        modificar.save();
    }
    //sino contiene ninguna palabra insertada, creamos la dosis como nueva
    else{
        //crea nueva pastilla
        Pastilla pastilla = new
        Pastilla(nombreName,horaName,diaName,userIDName,colorName);
        //guarda en db
        pastilla.save();
    }
}
}
public static void form(Long id) {
    if(id == null) {
        render();
    }
    Pastilla dosis = Pastilla.findById(id);
    render(dosis);
}
}
}

```

Model Pastilla.java

```

@Entity
public class Pastilla extends Model {
    @Required
    public String nombre;
    @Required
    public String hora;
    @Required
    public String dia;
    @Required
    public String userID;
    @Required
    public String color;

    public Pastilla(String nombreName, String horaName, String diaName,String userIDName,
String colorName) {
        this.nombre = nombreName;
        this.hora = horaName;
    }
}

```

```

        this.dia = diaName;
        this.userID = userIDName;
        this.color = colorName;
    }
}

```

Model Usuario.java

```

@Entity
public class Usuario extends Model{
    @Required
    public String nombre;
    @Required
    public String password;
    @Required
    public String idd;
    @Required
    public String sesion;
    public Usuario(String nombreU, String passwordU, String idU, String sesionU) {
        this.nombre = nombreU;
        this.password = passwordU;
        this.idd = idU;
        this.sesion = sesionU;
    }
}

```

listXml.xml

```

<pastillas>
    #{list pastillas, as:'pastilla'}
    <pastilla>
        <nombre>${pastilla.nombre}</nombre>
        <hora>${pastilla.hora}</hora>
        <dia>${pastilla.dia}</dia>
        <userid>${pastilla.userID}</userid>
        <color>${pastilla.color}</color>
    </pastilla>
    #{/list}
</pastillas>

```

listXmlUsers.xml

```

<usuarios>
    #{list usuarios, as:'usuario'}
    <usuario>
        <nombre>${usuario.nombre}</nombre>
        <password>${usuario.password}</password>
        <ident>${usuario.idd}</ident>
        <sesion>${usuario.sesion}</sesion>
    </usuario>
    #{/list}
</usuarios>

```

Fichero routes

```

# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

```

```

# Home page
GET /xml/pastillas Application.listXml(format:'xml')
GET /xml/usuarios Application.listXmlUsers(format:'xml')
POST /guardar Application.saveXml
POST /guardaruser Application.saveUser
# Map static resources from the /app/public folder to the /public path
GET / staticDir:public

```

8.2 Anexo 2: Código completo de la aplicación Android

Acciones.java

```

public class Acciones extends Activity{
    String color = Lista.colorselected;
    List<Pastilla> pastillas;
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.acciones);
    try {
        // Creamos la URL de donde obtendremos la información XML
        URL url = new URL("http://10.0.2.2:9000/xml/pastillas");
        // Conseguimos un parseador SAX
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser sp = spf.newSAXParser();
        // Declaramos el lector de XML
        XMLReader xr = sp.getXMLReader();
        // Creamos un objeto nueva de la clase EXAMPLEHANDLER
        HandlerPastillas myExampleHandler = new HandlerPastillas();
        xr.setContentHandler(myExampleHandler);
        // Parseamos los datos XML
        xr.parse(new InputStream(url.openStream()));
        //Fin del parseo
        pastillas = myExampleHandler.getPastilla();
    } catch (Exception e) {
        Log.d("****", "ERROR PARSEANDO");
    }
    //se declaran los editText y se rellenan con los datos de la dosis seleccionada
    final EditText editnombre = (EditText) findViewById(R.id.editnombre2);
    editnombre.setText(Lista.nombreselected);
    final String USUARIO_ACTUAL = Iniciosesion.USUARIO_ACTUAL;
    final TimePicker time2 = (TimePicker) findViewById(R.id.timePicker2);
    time2.setClickable(false);
    time2.setEnabled(false);
    String h = Lista.horaselected.substring(0,2);
    String m = Lista.horaselected.substring(3,5);
    int h1 = Integer.parseInt(h);
    int m1 = Integer.parseInt(m);
    time2.setCurrentHour(h1);
    time2.setCurrentMinute(m1);
    final CheckBox lunes = (CheckBox) findViewById(R.id.radioButton11);
    lunes.setClickable(false);
    lunes.setEnabled(false);
    //miramos que dias estan seleccionados y los señalamos
    if (Lista.diaselected.contains("Mon")){ // si LUNES está checkeado añadimos
las iniciales para enviarlas en XML
        lunes.setChecked(true);

```

```

    }else{
    final CheckBox martes = (CheckBox) findViewById(R.id.radioButton22);
    martes.setClickable(false);
    martes.setEnabled(false);
    if (Lista.diaSelected.contains("Tue")){// si MARTES está chequeado añadimos
las iniciales para enviarlas en XML
        martes.setChecked(true);
    }else{
    final CheckBox miercoles = (CheckBox) findViewById(R.id.radioButton33);
    miercoles.setClickable(false);
    miercoles.setEnabled(false);
    if (Lista.diaSelected.contains("Wed")){ // si MIERCOLES está chequeado
añadimos las iniciales para enviarlas en XML
        miercoles.setChecked(true);
    }else{
    final CheckBox jueves = (CheckBox) findViewById(R.id.radioButton44);
    jueves.setClickable(false);
    jueves.setEnabled(false);
    if (Lista.diaSelected.contains("Thu")){ // si JUEVES está chequeado añadimos
las iniciales para enviarlas en XML
        jueves.setChecked(true);
    }else{
    final CheckBox viernes = (CheckBox) findViewById(R.id.radioButton55);
    viernes.setClickable(false);
    viernes.setEnabled(false);
    if (Lista.diaSelected.contains("Fri")){ // si VIERNES está chequeado añadimos
las iniciales para enviarlas en XML
        viernes.setChecked(true);
    }else{
    final CheckBox sabado = (CheckBox) findViewById(R.id.radioButton66);
    sabado.setClickable(false);
    sabado.setEnabled(false);
    if (Lista.diaSelected.contains("Sat")){ // si SABADO está chequeado añadimos
las iniciales para enviarlas en XML
        sabado.setChecked(true);
    }else{
    final CheckBox domingo = (CheckBox) findViewById(R.id.radioButton77);
    domingo.setClickable(false);
    domingo.setEnabled(false);
    if (Lista.diaSelected.contains("Sun")){// si DOMINGO está chequeado añadimos
las iniciales para enviarlas en XML
        domingo.setChecked(true);
    }else{
    //creamos el ImageButtons y ponemos Enviar a INVISIBLE
    final ImageButton enviar = (ImageButton) findViewById(R.id.enviar);
    enviar.setVisibility(View.INVISIBLE);
    final ImageButton anadir = (ImageButton) findViewById(R.id.anadir);
    final ImageButton eliminar = (ImageButton) findViewById(R.id.eliminar);
    final ImageButton modificar = (ImageButton) findViewById(R.id.modificar);
    final ImageButton home = (ImageButton) findViewById(R.id.home2);
    //cuando clickamos en anadir...
    anadir.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            //cambiamos a Monitor
            Intent intent = new Intent(Acciones.this, Anadir.class);
            startActivity(intent);
        }
    });
    //cuando clickamos en eliminar
    eliminar.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {

```

```

//insertamos la palabra ELIMINAR para enviarla en el XML
String nueva="ELIMINAR";
//iniciamos el cliente HTTP
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost("http://10.0.2.2:9000/guardar");
try {
    httppost.setEntity(new StringEntity
("<pastillas><pastilla><nombre>"+nueva+Lista.nombreselected+"</nombre><hora>"+Lista.hora
selected+"</hora><dia>"+Lista.diaselected+"</dia><userid>"+USUARIO_ACTUAL+"</userid><
color></color></pastilla></pastillas>"));
    // Capturamos y mostramos la respuesta al HTTP Post
    HttpResponse response = httpclient.execute(httppost);
    Log.d("****", response.toString());
    //mostramos mensaje
    Toast.makeText(Acciones.this, "Dosis eliminada de la base de datos",
7).show();

    //enseñamos la lista guardada en servidor
    Intent intent = new Intent (Acciones.this, Lista.class);
    startActivity(intent);
} catch (ClientProtocolException e) {
    // TODO Auto-generated catch block
} catch (IOException e) {
    // TODO Auto-generated catch block
}
});
//cuando clickamos en Home
home.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
        //vamos a Home
        Intent intent = new Intent(Acciones.this, Home.class);
        startActivity(intent);
    }
});
//cuando clickamos en modificar
modificar.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        //activamos los editTexts referentes a la modificación de datos,el TimePicker,
los RadioButtons y el botón de Enviar
        //desactivamos los botones referentes a anadir, eliminar y modificar
        time2.setClickable(true);
        time2.setEnabled(true);
        lunes.setClickable(true);
        lunes.setEnabled(true);
        martes.setClickable(true);
        martes.setEnabled(true);
        miercoles.setClickable(true);
        miercoles.setEnabled(true);
        jueves.setClickable(true);
        jueves.setEnabled(true);
        viernes.setClickable(true);
        viernes.setEnabled(true);
        sabado.setClickable(true);
        sabado.setEnabled(true);
        domingo.setClickable(true);
        domingo.setEnabled(true);
        //editnombre.setEnabled(true);
        // editnombre.setFocusable(true);
        eliminar.setVisibility(View.INVISIBLE);
        anadir.setVisibility(View.INVISIBLE);
        modificar.setVisibility(View.INVISIBLE);

```

```

    enviar.setVisibility(View.VISIBLE);
  });
  //cuando clickamos en Enviar...
  enviar.setOnClickListener(new OnClickListener() {
  public void onClick(View v) {
    //pasamos a String los campos que se han rellenado para ser modificados
    final String nombremod = editnombre.getText().toString();
    String dia = new String();
    enviarla en XML
    if (lunes.isChecked()){ // si LUNES está checkeado añadimos la inicial para
        dia = "Mon";
    }else{}
    enviarla en XML
    if (martes.isChecked()){ // si MARTES está checkeado añadimos la inicial para
        dia = dia+"Tue";
    }else{}
    para enviarla en XML
    if (miercoles.isChecked()){ // si MIERCOLES está checkeado añadimos la inicial
        dia = dia+"Wed";
    }else{}
    enviarla en XML
    if (jueves.isChecked()){ // si JUEVES está checkeado añadimos la inicial para
        dia = dia+"Thu";
    }else{}
    enviarla en XML
    if (viernes.isChecked()){ // si VIERNES está checkeado añadimos la inicial para
        dia = dia+"Fri";
    }else{}
    enviarla en XML
    if (sabado.isChecked()){ // si SABADO está checkeado añadimos la inicial para
        dia = dia+"Sat";
    }else{}
    para enviarla en XML
    if (domingo.isChecked()){ // si DOMINGO está checkeado añadimos la inicial
        dia = dia+"Sun";
    }else{}

    String hf = time2.getCurrentHour().toString(); //pasamos h a string
    String mf = time2.getCurrentMinute().toString(); //pasamos m a
    string
    //si horas y minutos son de una cifra, insertamos 1 o 2 ceros
    para asegurar el formato XX:XX
    if (hf.length()==1){
      hf="0"+hf;
    }
    if (mf.length()==1){
      mf="0"+mf;
    }
    //formateamos la información de la hora y los días
    final String horamod = (hf+":"+mf);
    int j=0;
    int coincide=0;
    //recorremos el vector de pastillas para encontrar si ya existe
    while(j<pastillas.size())
    {
      //si la dosis ya está en la base de datos...
      if(pastillas.get(j).nombre.equals(nombremod)&&pastillas.get(j).hora.equals(horamod))
      {
        coincide=1;
      }
    }
  }
}

```



```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.anadir);
    final Button post = (Button) findViewById(R.id.post);
    final Spinner spinner = (Spinner) findViewById(R.id.spinner);
    ArrayAdapter<CharSequence> adapter =
    ArrayAdapter.createFromResource(Anadir.this, R.array.colores_array,
    android.R.layout.simple_spinner_item);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost("http://10.0.2.2:9000/guardaruser");
    try {
        //creamos el XML con la variable nueva, el nombre, el tiempo y los dias
        httpPost.setEntity(new StringEntity
       ("<usuarios><usuario><nombre>USUARIO"+USUARIO_ACTUAL+"</nombre><password></pa
        ssword><ident></ident><sesion></sesion></usuario></usuarios>"));
        // Ejecutamos y capturamos la respuesta HTTP
        HttpResponse response = httpClient.execute(httpPost);
        Log.d("****", response.toString());
    } catch (ClientProtocolException e) {
        // TODO Auto-generated catch block
    } catch (IOException e) {
        // TODO Auto-generated catch block
    }
    try {
        // Creamos la URL de donde obtendremos la información XML
        URL url = new URL("http://10.0.2.2:9000/xml/pastillas");
        // Conseguimos un parseador SAX
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser sp = spf.newSAXParser();
        // Declaramos el lector de XML
        XMLReader xr = sp.getXMLReader();
        // Creamos un objeto nueva de la clase EXAMPLEHANDLER
        HandlerPastillas myExampleHandler = new HandlerPastillas();
        xr.setContentHandler(myExampleHandler);
        // Parseamos los datos XML
        xr.parse(new InputSource(url.openStream()));
        //Fin del parseo
        pastillas = myExampleHandler.getPastilla();
        Log.d("****", "HAY "+pastillas.size()+" pastillas");
    } catch (Exception e) {
        Log.d("****", "ERROR PARSEANDO");
    }
    spinner.setOnItemSelectedListener(new OnItemSelectedListener(){
    public void onItemSelected(AdapterView<?> parent, View view, int pos, long
id){
        color = parent.getItemAtPosition(pos).toString();
        Toast.makeText(Anadir.this, color, 6).show();
    }
    public void onNothingSelected(AdapterView parent) {
        // Do nothing.
    }
    });
    final ImageButton home = (ImageButton) findViewById(R.id.home);
    home.setOnClickListener(new OnClickListener(){
    public void onClick(View v){

```

```

        Intent intent=new Intent(Anadir.this, Home.class);
        startActivity(intent);
    }
});
//cuando clickamos en post...
post.setOnClickListener(new OnClickListener() {
@Override
    public void onClick(View v) {
//declaramos EditText,RadioButtons y TimePicker...
final EditText editnombre = (EditText) findViewById(R.id.editnombre);
final String nombre = editnombre.getText().toString();
if(nombre.length()<1){
    Toast.makeText(Anadir.this, "Debe darle un nombre al medicamento",
6).show();
    return;
}
else{
//declaramos los RadioButtons y creamos el String de días...
final CheckBox lunes = (CheckBox)
findViewById(R.id.radioButton1);
String dia = new String();
if (lunes.isChecked()){ // si LUNES está checkeado añadimos la
inicial para enviarla en XML
        dia = "Mon";
    }else{}
    final CheckBox martes = (CheckBox)
findViewById(R.id.radioButton2);
if (martes.isChecked()){ // si MARTES está checkeado
añadimos la inicial para enviarla en XML
        dia = dia+"Tue";
    }else{}
    final CheckBox miercoles = (CheckBox)
findViewById(R.id.radioButton3);
if (miercoles.isChecked()){ // si MIERCOLES está checkeado
añadimos la inicial para enviarla en XML
        dia = dia+"Wed";
    }else{}
    final CheckBox jueves = (CheckBox)
findViewById(R.id.radioButton4);
if (jueves.isChecked()){ // si JUEVES está checkeado añadimos
la inicial para enviarla en XML
        dia = dia+"Thu";
    }else{}
    final CheckBox viernes = (CheckBox)
findViewById(R.id.radioButton5);
if (viernes.isChecked()){ // si VIERNES está checkeado
añadimos la inicial para enviarla en XML
        dia = dia+"Fri";
    }else{}
    final CheckBox sabado = (CheckBox)
findViewById(R.id.radioButton6);
if (sabado.isChecked()){ // si SABADO está checkeado
añadimos la inicial para enviarla en XML
        dia = dia+"Sat";
    }else{}
    final CheckBox domingo = (CheckBox)
findViewById(R.id.radioButton7);
if (domingo.isChecked()){ // si DOMINGO está checkeado
añadimos la inicial para enviarla en XML
        dia = dia+"Sun";

```

```

        }else{
            //cogemos la hora
            final TimePicker time = (TimePicker)
findViewById(R.id.timePicker1);
            String h = time.getCurrentHour().toString(); //pasamos h a string
            String m = time.getCurrentMinute().toString(); //pasamos m a
string
            //si horas y minutos son de una cifra, insertamos 1 o 2 ceros
para asegurar el formato XX:XX
            if (h.length()==1){
                h="0"+h;
            }
            if (m.length()==1){
                m="0"+m;
            }
            final String tiempo = (h+":"+m);
            final String diafinal = dia;
            //final int idnum = (int) (Math.random()*10000);
            final String id = USUARIO_ACTUAL;
            if((lunes.isChecked()==false)&&(martes.isChecked()==false)&&(miercoles.isChecked()
==false)&&(jueves.isChecked()==false)&&(viernes.isChecked()==false)&&(sabado.isChecked(
)==false)&&(domingo.isChecked()==false)){
                Toast.makeText(Anadir.this, "Introduzca los días a guardar", 6).show();
                return;
            }else{
                int j=0;
                int ya_existe = 0;
                //recorremos el vector de pastillas para encontrar si ya existe
                while(j<pastillas.size())
                {
                    //si el medicamento ya está en la base de datos...
                    if(pastillas.get(j).nombre.equals(nombre))
                    {
                        color = pastillas.get(j).color;
                        //mostramos mensaje de error
                        ya_existe=0;
                        if(pastillas.get(j).hora.equals(tiempo)){
                            //mostramos mensaje de error
                            Toast.makeText(Anadir.this, "Esta dosis ya se encuentra en su
base de datos. Cambie la hora.", 12).show();
                            //ponemos edittext en blanco
                            editnombre.setText("");
                            //ponemos a 1 la variable ya_existe
                            ya_existe=1;
                            return;}
                        else{
                            Toast.makeText(Anadir.this, "Este medicamento ya se encuentra en su base de datos.
Se le asignará el color "+color, 16).show();
                            j++;
                        }}
                        else{
                            j++;}
                    }
                }
                if(ya_existe==0){
                    // Creamos una petición HTTP
                    HttpClient httpclient = new DefaultHttpClient();
                    HttpPost httppost = new HttpPost("http://10.0.2.2:9000/guardar");
                    try {
                        //creamos el XML con la variable nueva, el nombre, el tiempo y los días

```



```

    }
    try {
        // Creamos la URL de donde obtendremos la información XML
        URL url = new URL("http://10.0.2.2:9000/xml/pastillas");
        // Conseguimos un parseador SAX
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser sp = spf.newSAXParser();
        // Declaramos el lector de XML
        XMLReader xr = sp.getXMLReader();
        // Creamos un objeto nueva de la clase EXAMPLEHANDLER
        HandlerPastillas myExampleHandler = new HandlerPastillas();
        xr.setContentHandler(myExampleHandler);
        // Parseamos los datos XML
        xr.parse(new InputStream(url.openStream()));
        //Fin del parseo
        pastillas = myExampleHandler.getPastilla();
    } catch (Exception e) {
        Log.d("****", "ERROR PARSEANDO");
    }
}
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    return super.onKeyDown(keyCode, event);
}
private class DemoView extends View {
    public DemoView(Context context) {
        super(context);
    }
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        Paint paint = new Paint();
        paint.setStyle(Paint.Style.FILL);
        //pintamos el canvas de blanco
        paint.setColor(Color.WHITE);
        canvas.drawPaint(paint);
        Display display = getWindowManager().getDefaultDisplay();
        //cogemos las dimensiones de la pantalla
        width = display.getWidth();//start
        height = display.getHeight();//end
        //dividimos ancho entre 8 casillas
        xpos = width / 8;
        //dividimos alto entre 23 casillas
        ypos = height/23;
        for (int i = 0; i < 8; i++) {
            //dibujamos las lineas verticales en gris
            paint.setColor(Color.GRAY);
            canvas.drawLine(xpos +(xpos*i), 0, xpos +(xpos*i), height, paint);
        }
        paint.setStyle(Paint.Style.STROKE);
        for (int i = 0; i < 23; i++) {
            //dibujamos las lineas horizontales en gris
            paint.setColor(Color.GRAY);
            canvas.drawLine(0, (ypos*pass)+ 5, width, (ypos*pass)+5, paint);
            pass++;
        }
        //creamos 2 nuevos objetos de pintar
        Paint paint2 = new Paint();
        paint2.setStyle(Paint.Style.FILL);
        Paint paint3 = new Paint();
    }
}

```

```

paint3.setStyle(Paint.Style.FILL);
paint2.setColor(Color.BLACK);
paint2.setFakeBoldText(true);
paint2.setTextSize(17);
paint3.setColor(Color.BLACK);
paint3.setTextSize(10);
//escribimos los días de la semana en la primera fila
canvas.drawText("lun", xpos+10, ypos, paint2);
canvas.drawText("mar", (2*xpos)+8, ypos, paint2);
canvas.drawText("mie", (3*xpos)+10, ypos, paint2);
canvas.drawText("jue", (4*xpos)+10, ypos, paint2);
canvas.drawText("vie", (5*xpos)+10, ypos, paint2);
canvas.drawText("sab", (6*xpos)+10, ypos, paint2);
canvas.drawText("dom", (7*xpos)+6, ypos, paint2);
//escribimos las horas del día en la primera columna
canvas.drawText("6.00", 5, (2*ypos)-6, paint3);
canvas.drawText("7.00", 5, (3*ypos)-6, paint3);
canvas.drawText("8.00", 5, (4*ypos)-6, paint3);
canvas.drawText("9.00", 5, (5*ypos)-6, paint3);
canvas.drawText("10.00", 5, (6*ypos)-6, paint3);
canvas.drawText("11.00", 5, (7*ypos)-6, paint3);
canvas.drawText("12.00", 5, (8*ypos)-6, paint3);
canvas.drawText("13.00", 5, (9*ypos)-6, paint3);
canvas.drawText("14.00", 5, (10*ypos)-6, paint3);
canvas.drawText("15.00", 5, (11*ypos)-6, paint3);
canvas.drawText("16.00", 5, (12*ypos)-6, paint3);
canvas.drawText("17.00", 5, (13*ypos)-6, paint3);
canvas.drawText("18.00", 5, (14*ypos)-6, paint3);
canvas.drawText("19.00", 5, (15*ypos)-6, paint3);
canvas.drawText("20.00", 5, (16*ypos)-6, paint3);
canvas.drawText("21.00", 5, (17*ypos)-6, paint3);
canvas.drawText("22.00", 5, (18*ypos)-6, paint3);
canvas.drawText("23.00", 5, (19*ypos)-6, paint3);
canvas.drawText("00.00", 5, (20*ypos)-6, paint3);
canvas.drawText("1.00", 5, (21*ypos)-6, paint3);
canvas.drawText("2.00", 5, (22*ypos)-6, paint3);
canvas.drawText("3.00", 5, (23*ypos)-6, paint3);
int i=0;
while(i<pastillas.size()){
leyenda = leyenda + pastillas.get(i).nombre+"      "+pastillas.get(i).color+"\n\n";
//creamos otro objeto de pintura para los puntos
Paint paint4 = new Paint();
paint4.setStyle(Paint.Style.FILL);
//cogemos el color de la dosis y utilizamos el color correspondiente
color=pastillas.get(i).color;
if(color.equals("Rojo")){
    paint4.setColor(Color.RED);
}else if(color.equals("Verde")){
    paint4.setColor(Color.GREEN);
}else if(color.equals("Negro")){
    paint4.setColor(Color.BLACK);
}else if(color.equals("Amarillo")){
    paint4.setColor(Color.YELLOW);
}else if(color.equals("Azul")){
    paint4.setColor(Color.BLUE);
}else if(color.equals("Naranja")){
    paint4.setColor(Color.parseColor("#FF8000"));
}else if(color.equals("Lila")){
    paint4.setColor(Color.parseColor("#4C0B5F"));
}else if(color.equals("Marron")){

```

```
        paint4.setColor(Color.parseColor("#3B240B"));
    }
    //cogemos la hora
    String hora = pastillas.get(i).hora.substring(0,2);
    //calculamos las coordenadas para cada par hora/dia
    if (hora.equals("06")){
        y=(2*ypos)-6;
    }else if(hora.equals("07")){
        y=(3*ypos)-6;
    }else if(hora.equals("08")){
        y=(4*ypos)-6;
    }else if(hora.equals("09")){
        y=(5*ypos)-6;
    }else if(hora.equals("10")){
        y=(6*ypos)-6;
    }else if(hora.equals("11")){
        y=(7*ypos)-6;
    }else if(hora.equals("12")){
        y=(8*ypos)-6;
    }else if(hora.equals("13")){
        y=(9*ypos)-6;
    }else if(hora.equals("14")){
        y=(10*ypos)-6;
    }else if(hora.equals("15")){
        y=(11*ypos)-6;
    }else if(hora.equals("16")){
        y=(12*ypos)-6;
    }else if(hora.equals("17")){
        y=(13*ypos)-6;
    }else if(hora.equals("18")){
        y=(14*ypos)-6;
    }else if(hora.equals("19")){
        y=(15*ypos)-6;
    }else if(hora.equals("20")){
        y=(16*ypos)-6;
    }else if(hora.equals("21")){
        y=(17*ypos)-6;
    }else if(hora.equals("22")){
        y=(18*ypos)-6;
    }else if(hora.equals("23")){
        y=(19*ypos)-6;
    }else if(hora.equals("00")){
        y=(20*ypos)-6;
    }else if(hora.equals("01")){
        y=(21*ypos)-6;
    }else if(hora.equals("02")){
        y=(22*ypos)-6;
    }else if(hora.equals("03")){
        y=(23*ypos)-6;
    }
    }
    //pintamos circulos a medida que encontramos los días, ya que una dosis puede
    tener asignados varios días
    if(pastillas.get(i).dia.contains("Mon")){
        x=(xpos)+15;
        canvas.drawCircle(x, y, 6, paint4);
    }else{
    if(pastillas.get(i).dia.contains("Tue")){
        x=(2*xpos)+13;
        canvas.drawCircle(x, y, 6, paint4);
    }else{
```



```

//Fin del parseo
usuarios = myExampleHandler.getUsuario();
} catch (Exception e) {
    Log.d("****", "ERROR PARSEANDO");
}

//se crean los botones y editText
final Button crear = (Button) findViewById(R.id.crearuser);
final ImageButton salida = (ImageButton) findViewById(R.id.salida1);
final EditText user = (EditText) findViewById(R.id.edituser);
final EditText password = (EditText) findViewById(R.id.editpassword);
//al clicar en salida, nos vamos a Main
salida.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent(CrearUser.this, PaginaUno.class);
        startActivity(intent);
    }
});
//al clicar en crear nuevo usuario...
crear.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        //cogemos el string de nombre de usuario
        final String username = user.getText().toString();
        //el usuario debe tener como mínimo 5 caracteres
        if(username.length()<5){
            Toast.makeText(CrearUser.this, "El usuario ha de tener como mínimo
5 caracteres", 7).show();
            return;
        }else{
            int i=0;
            int ya_existe = 0;
            //recorremos el vector de usuarios para encontrar si ya existe
            while(i<usuarios.size())
            {
                //si ya existe...
                if(usuarios.get(i).nombre.contains(username)){
                    //mostramos mensaje de error
                    Toast.makeText(CrearUser.this, "Nombre de usuario no
disponible", 6).show();

                    //ponemos editTexts en blanco
                    user.setText("");
                    password.setText("");
                    //ponemos a 1 la variable ya_existe
                    ya_existe=1;
                    return;
                }
                else{
                    i++;
                }
            }
            //si no existe...
            if(ya_existe==0){
                //cogemos contraseña, creamos un ID aleatorio y señalamos que no ha
                //iniciado sesión
                final String passwordname = password.getText().toString();
                if(passwordname.length()<6){
                    Toast.makeText(CrearUser.this, "La contraseña debe tener 6 o
más caracteres", 6).show();
                    password.setText("");
                    return;
                }else{
                    //creamos id

```



```

        estado = Estado.NOMBRE;
    }else if (localName.equals("hora")) {
        estado = Estado.HORA;
    }else if (localName.equals("dia")) {
        estado = Estado.DIA;
    }else if (localName.equals("userid")) {
        estado = Estado.ID;
    }else if (localName.equals("color")) {
        estado = Estado.COLOR;
    }else if (localName.equals("pastilla")) {
        tmp = new Pastilla(); //si llegamos al tag pastilla, estamos en el cierre, por tanto
guardamos
    }
}
//cuando acabamos un elemento, dejamos el estado en nada para no reconocer espacios ni
tabulaciones en el XML
@Override
public void endElement(String namespaceURI, String localName, String qName)
throws SAXException {
    estado = Estado.NADA;
    if (localName.equals("pastilla")) {
        pastillas.add(tmp);
    }
}
//que hacemos cuando estamos entre tags: miramos en que tag estamos y guardamos el String
como una u otra variable
@Override
public void characters(char ch[], int start, int length) {
    String s = new String(ch,start,length);
    switch(estado) {
        case NOMBRE:
            tmp.nombre = s;
            break;
        case HORA:
            tmp.hora = s;
            break;
        case DIA:
            tmp.dia = s;
            break;
        case ID:
            tmp.userID = s;
            break;
        case COLOR:
            tmp.color = s;
            break;
    }
}
}
}

```

HandlerUsuarios.java

```

public class HandlerUsuarios extends DefaultHandler{
    //Campos
    private enum Estado { NOMBRE, PASSWORD, IDENT, SESION, NADA};
    private Estado estado = Estado.NADA;
    private Vector<Usuario> usuarios;
    private Usuario tmp;

    //Constructor del vector
    public Vector<Usuario> getUsuario() {

```

```

        return usuarios;
    }
}
// Metodos de SAX
@Override
//cuando empieza el documento...
public void startDocument() throws SAXException {
    usuarios = new Vector<Usuario>();
}
@Override
//cuando acaba el documento...
public void endDocument() throws SAXException {
    // Nada que hacer
}
@Override
//cuando empezamos un elemento, miramos en que TAG nos encontramos
public void startElement(String namespaceURI, String localName,
    String qName, Attributes atts) throws SAXException {
    if (localName.equals("nombre")) {
        estado = Estado.NOMBRE;
    }else if (localName.equals("password")) {
        estado = Estado.PASSWORD;
    }else if (localName.equals("ident")) {
        estado = Estado.IDENT;
    }else if (localName.equals("sesion")) {
        estado = Estado.SESION;
    }else if (localName.equals("usuario")) {
        tmp = new Usuario(); //si llegamos al tag usuario, estamos en el cierre, por
        tanto guardamos
    }
}
//cuando acabamos un elemento, dejamos el estado en nada para no reconocer espacios ni
tabulaciones en el XML
@Override
public void endElement(String namespaceURI, String localName, String qName)
    throws SAXException {
    estado = Estado.NADA;
    if (localName.equals("usuario")) {
        usuarios.add(tmp);
    }
}
//que hacemos cuando estamos entre tags: miramos en que tag estamos y guardamos el String
como una u otra variable
@Override
public void characters(char ch[], int start, int length) {
    String s = new String(ch,start,length);
    switch(estado) {
    case NOMBRE:
        tmp.nombre = s;
        break;
    case PASSWORD:
        tmp.password = s;
        break;
    case IDENT:
        tmp.idd = s;
        break;
    case SESION:
        tmp.sesion = s;
        break;
    } } }

```

Home.java

```

public class Home extends Activity{
    List<Pastilla> pastillas;
    List<Usuario> usuarios;
    String USUARIO_ACTUAL = Iniciosesion.USUARIO_ACTUAL;
    String PROXIMO_NOMBRE="";
    String PROXIMO_HORA="";
    String PROXIMO_DIA="";
    String mensaje = "Dosis programadas para hoy:\n";
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.home);
        //lo primero que hacemos al entrar en la aplicación es una actualización de
nuestra Lista de dosis
        HttpClient httpClient = new DefaultHttpClient();
        HttpPost httpPost = new HttpPost("http://10.0.2.2:9000/guardaruser");
        try {
            //creamos el XML con la variable nueva, el nombre, el tiempo y los dias
            httpPost.setEntity(new StringEntity
("<usuarios><usuario><nombre>USUARIO"+USUARIO_ACTUAL+"</nombre><password></pa
ssword><ident></ident><sesion></sesion></usuario></usuarios>"));
            // Ejecutamos y capturamos la respuesta HTTP
            HttpResponse response = httpClient.execute(httpPost);
            Log.d("****", response.toString());
        } catch (ClientProtocolException e) {
            // TODO Auto-generated catch block
        } catch (IOException e) {
            // TODO Auto-generated catch block
        }
        try {
            // Creamos la URL de donde obtendremos la información XML
            URL url = new URL("http://10.0.2.2:9000/xml/pastillas");
            // Conseguimos un parseador SAX
            SAXParserFactory spf = SAXParserFactory.newInstance();
            SAXParser sp = spf.newSAXParser();
            // Declaramos el lector de XML
            XMLReader xr = sp.getXMLReader();
            // Creamos un objeto nueva de la clase EXAMPLEHANDLER
            HandlerPastillas myExampleHandler = new HandlerPastillas();
            xr.setContentHandler(myExampleHandler);
            // Parseamos los datos XML
            xr.parse(new InputSource(url.openStream()));
            //Fin del parseo
            pastillas = myExampleHandler.getPastilla();
        } catch (Exception e) {
            Log.d("****", "ERROR PARSEANDO");
        }
    }
    try {
        // Creamos la URL de donde obtendremos la información XML
        URL url = new URL("http://10.0.2.2:9000/xml/usuarios");
        // Conseguimos un parseador SAX
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser sp = spf.newSAXParser();
        // Declaramos el lector de XML
        XMLReader xr = sp.getXMLReader();
        // Creamos un objeto nueva de la clase EXAMPLEHANDLER
        HandlerUsuarios myExampleHandler = new HandlerUsuarios();
        xr.setContentHandler(myExampleHandler);
        // Parseamos los datos XML
        xr.parse(new InputSource(url.openStream()));
    }
}

```

```

//Fin del parseo
    usuarios = myExampleHandler.getUsuario();
} catch (Exception e) {
    Log.d("****", "ERROR PARSEANDO");
}

//declaramos los textviews
final TextView text=(TextView) findViewById(R.id.textViewHome);
text.setText("HOME");
final TextView text2=(TextView) findViewById(R.id.textViewHome2);
//cogemos hora actual
Date horaActual = new Date();
String fecha = horaActual.toString();
String hora= fecha.substring(11, 16);
int horaint=Integer.parseInt(hora.substring(0,2));
int minutoint=Integer.parseInt(hora.substring(3,5));
String dia = fecha.substring(0,3);
//si no hay dosis en nuestra lista
if (pastillas.size()==0){
    text2.setText("Usted aun no tiene dosis en su lista");
    //Toast.makeText(Home.this,"Usted aun no tiene dosis en su lista",
6).show();
}
} else{
int i=0;
while(i<pastillas.size()){
int h = Integer.parseInt(pastillas.get(i).hora.substring(0,2));
int m = Integer.parseInt(pastillas.get(i).hora.substring(3));
if(pastillas.get(i).dia.contains(dia)){
if(h<horaint){//si la dosis es anterior a la hora actual
PROXIMO_DIA="NO";
i++;
} else {//si la dosis es anterior a la hora actual
if(h==horaint&& m<minutoint){
PROXIMO_DIA="NO";
i++;
} else{//si la dosis es posterior a la hora actual
mensaje=mensaje+pastillas.get(i).nombre+"
"+pastillas.get(i).hora+"\n";
text2.setText(mensaje);
PROXIMO_DIA="SI";
i++;
}
}
}
}
} else{
i++;
}
}
}
if(PROXIMO_DIA=="NO"){//si no hay dosis para el dia de hoy
text2.setText("No tiene dosis programadas para hoy");
} else{text2.setText(mensaje);}
}
final ImageButton calendario = (ImageButton) findViewById (R.id.calendario);
//al clickar en calendario
calendario.setOnClickListener(new OnClickListener(){
public void onClick(View v){
//vamos al calendario
Intent intent = new Intent(Home.this, Calendario.class);
startActivity(intent);
}});

```

```

        //al clicar en AÑADIR..
        final ImageButton anadir = (ImageButton) findViewById(R.id.crear);
        anadir.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent(Home.this, Anadir.class);
                startActivity(intent);
            }
        });

        final ImageButton ver = (ImageButton) findViewById(R.id.ver);
        //deshabilitamos calendario y ver lista si no existe lista para el usuario iniciado
        if (pastillas.size()==0){
            ver.setClickable(false);
            ver.setEnabled(false);
            calendario.setClickable(false);
            calendario.setEnabled(false);
        }
        //clickando en Lista
        ver.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                //nos lleva a la lista
                Intent intent = new Intent(Home.this, Lista.class);
                startActivity(intent);
            }
        });

        //cuando cliamos en salida...
        final ImageButton salida = (ImageButton) findViewById(R.id.salida3);
        salida.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                int i = 0;
                //recorremos el vector de usuarios buscando el que tiene la sesión iniciada
                while(i<usuarios.size()){
                    if (usuarios.get(i).idd.equals(USUARIO_ACTUAL)){
                        //ponemos la variable de sesion iniciada a no
                        usuarios.get(i).sesion = "no";
                        HttpClient httpClient = new DefaultHttpClient();
                        HttpPost httpPost = new HttpPost("http://10.0.2.2:9000/guardaruser");
                        try {
                            //creamos el XML con la variable nueva, el nombre, el tiempo y los dias
                            httpPost.setEntity(new StringEntity
                                ("<usuarios><usuario><nombre>CERRAR"+usuarios.get(i).nombre+"</nombre><password>"+
                                usuarios.get(i).password+"</password><ident>"+usuarios.get(i).idd+"</ident><sesion>"+usuari
                                os.get(i).sesion+"</sesion></usuario></usuarios>"));
                            // Ejecutamos y capturamos la respuesta HTTP
                            HttpResponse response = httpClient.execute(httpPost);
                            Log.d("****", response.toString());

                            //cerramos el servicio
                            Toast.makeText(Home.this, "Sesion cerrada", 7).show();
                            Intent servicio = new Intent();
                            servicio.setAction("tfc.epsc.android.Servicio");
                            stopService(servicio);
                            Intent intent = new Intent (Home.this, PaginaUno.class);
                            startActivity(intent);
                            return;
                        } catch (ClientProtocolException e) {
                            // TODO Auto-generated catch block
                        } catch (IOException e) {
                            // TODO Auto-generated catch block
                        }
                    }
                }

            }else{

```

```

        i++;
    }
}
});}}

```

Iniciosesion.java

```

public class Iniciosesion extends Activity {
    /** Called when the activity is first created. */
    List<Usuario> usuarios;
    static String USUARIO_ACTUAL="user";
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.iniciarsesion);
        //creamos botones...
        final ImageButton salida = (ImageButton) findViewById(R.id.salida2);
        final Button iniciar = (Button) findViewById(R.id.iniciarsesion);
        //cuando clickamos en salida...
        salida.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                //nos vamos a Main
                Intent intent = new Intent(Iniciosesion.this, PaginaUno.class);
                startActivity(intent);
            }
        });
        //cuando clickamos en iniciar sesión...
        iniciar.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                try {
                    // Creamos la URL de donde obtendremos la información XML
                    URL url = new URL("http://10.0.2.2:9000/xml/usuarios");
                    // Conseguimos un parseador SAX
                    SAXParserFactory spf = SAXParserFactory.newInstance();
                    SAXParser sp = spf.newSAXParser();
                    // Declaramos el lector de XML
                    XMLReader xr = sp.getXMLReader();
                    // Creamos un objeto nueva de la clase EXAMPLEHANDLER
                    HandlerUsuarios myExampleHandler = new HandlerUsuarios();
                    xr.setContentHandler(myExampleHandler);
                    // Parseamos los datos XML
                    xr.parse(new InputSource(url.openStream()));
                    //Fin del parseo
                    usuarios = myExampleHandler.getUsuario();
                } catch (Exception e) {
                    Log.d("****", "ERROR PARSEANDO");
                }
                //declaramos edittexts y cogemos su contenido
                final EditText user = (EditText) findViewById (R.id.editusersesion);
                final String username = user.getText().toString();

                final EditText password = (EditText) findViewById
(R.id.editpasswordsesion);
                final String passwordname = password.getText().toString();
                int i=0;
                int encontrado=0;
                //recorremos el vector de usuarios en busca del usuario que quiere iniciar
                while(i<usuarios.size()&&encontrado==0){
                    //si lo encontramos...
                    if(username.equals(usuarios.get(i).nombre)){

```

```

        //...y la contraseña concuerda...
        if (passwordname.equals(usuarios.get(i).password)){
            //...iniciamos sesión
            Toast.makeText(Iniciosesion.this, "Datos correctos.
Iniciando sesión...", 5).show();

            encontrado=1;
            USUARIO_ACTUAL = usuarios.get(i).idd;
            usuarios.get(i).sesion="si";
            HttpClient httpclient = new DefaultHttpClient();
            HttpPost httppost = new HttpPost("http://10.0.2.2:9000/guardaruser");
            try {
                //creamos el XML con la variable nueva, el nombre, el tiempo y los dias
                httppost.setEntity(new StringEntity
               ("<usuarios><usuario><nombre>INICIAR"+usuarios.get(i).nombre+"</nombre><password>"+us
                uarios.get(i).password+"</password><ident>"+usuarios.get(i).idd+"</ident><sesion>"+usuarios.
                get(i).sesion+"</sesion></usuario></usuarios>"));
                // Ejecutamos y capturamos la respuesta HTTP
                HttpResponse response = httpclient.execute(httppost);
                Log.d("****", response.toString());
                Toast.makeText(Iniciosesion.this, "Sesion iniciada", 7).show();
                //realizamos petición para el inicio del Servicio
                Intent servicio = new Intent();
                servicio.setAction("tfc.epsc.android.Servicio");
                startService(servicio);
                //nos dirigimos a Home
                Intent intent = new Intent(Iniciosesion.this,
Home.class);

                startActivity(intent);

                } catch (ClientProtocolException e) {
                    // TODO Auto-generated catch block
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                }
            }
            //...si la contraseña no concuerda
            else{
                //Notificamos el error
                Toast.makeText(Iniciosesion.this, "Contraseña
incorrecta para este usuario", 5).show();
                password.setText("");
                return;
            }
        }
        }else{
            i++;
        }
    }
    //si el usuario no se encuentra en la base de datos...
    if (encontrado==0){
        Toast.makeText(Iniciosesion.this, "Usuario no encontrado. Regístrese",
5).show();
    }
}
});
}
}

```

Lista.java

```

public class Lista extends ListActivity {
    List<Pastilla> pastillas;
    static String nombreselected = new String();
    static String horaselected = new String();
    static String diaselected = new String();
    static String colorselected = new String();
    String USUARIO_ACTUAL = Iniciosesion.USUARIO_ACTUAL;
    String dias="";
    public void onCreate(Bundle savedInstanceState) {
        //al crear la vista, realizamos un parseo de un XML que nos llega del servidor
        para listar
        super.onCreate(savedInstanceState);
        HttpClient httpclient = new DefaultHttpClient();
        HttpPost httppost = new HttpPost("http://10.0.2.2:9000/guardaruser");
        try {
            //creamos el XML con la variable nueva, el nombre, el tiempo y los dias
            httppost.setEntity(new StringEntity
           ("<usuarios><usuario><nombre>USUARIO"+USUARIO_ACTUAL+"</nombre><password></pa
            ssword><ident></ident><sesion></sesion></usuario></usuarios>"));
            // Ejecutamos y capturamos la respuesta HTTP
            HttpResponse response = httpclient.execute(httppost);
            Log.d("****", response.toString());
        } catch (ClientProtocolException e) {
            // TODO Auto-generated catch block
        } catch (IOException e) {
            // TODO Auto-generated catch block
        }
        try {
            // Creamos la URL de donde obtendremos la información XML
            URL url = new URL("http://10.0.2.2:9000/xml/pastillas");
            // Conseguimos un parseador SAX
            SAXParserFactory spf = SAXParserFactory.newInstance();
            SAXParser sp = spf.newSAXParser();
            // Declaramos el lector de XML
            XMLReader xr = sp.getXMLReader();
            // Creamos un objeto nueva de la clase EXAMPLEHANDLER
            HandlerPastillas myExampleHandler = new HandlerPastillas();
            xr.setContentHandler(myExampleHandler);
            // Parseamos los datos XML
            xr.parse(new InputSource(url.openStream()));
            //Fin del parseo
            pastillas = myExampleHandler.getPastilla();
        } catch (Exception e) {
            Log.d("****", "ERROR PARSEANDO");
        }
        // creamos el adaptador para la vista de Lista
        setListAdapter(new ArrayAdapter <Pastilla>(this, R.layout.list, pastillas){
            public View getView(int position, View convertView, ViewGroup
parent) {
                View row = convertView;
                if (null == convertView) {
                    row = Lista.this.getLayoutInflater().inflate(R.layout.list,
null);
                } else {
                    row = convertView;
                }
                //enseñamos en la lista los valores de los objetos recuperados
                TextView tv = (TextView) row.findViewById(R.id.txt);
            }
        });
    }
}

```

```

//cada dosis sale del color seleccionado
if (pastillas.get(position).nombre!= null){
    if(pastillas.get(position).color.equals("Rojo")){
        tv.setBackgroundColor(Color.RED);
    }else if(pastillas.get(position).color.equals("Verde")){
        tv.setBackgroundColor(Color.GREEN);
    }else if(pastillas.get(position).color.equals("Azul")){
        tv.setBackgroundColor(Color.BLUE);
    }else if(pastillas.get(position).color.equals("Amarillo")){
        tv.setBackgroundColor(Color.YELLOW);
    }else if(pastillas.get(position).color.equals("Naranja")){
        tv.setBackgroundColor(Color.parseColor("#FF8000"));
    }else if(pastillas.get(position).color.equals("Negro")){
        tv.setBackgroundColor(Color.BLACK);
    }else if(pastillas.get(position).color.equals("Marron")){

tv.setBackgroundColor(Color.parseColor("#3B240B"));
    }else if(pastillas.get(position).color.equals("Lila")){

tv.setBackgroundColor(Color.parseColor("#4C0B5F"));
    }
}
//convertimos las iniciales de los dias a Español
if(pastillas.get(position).dia.contains("Mon")){
    dias=dias+"Lunes ";
}
}else{}
if(pastillas.get(position).dia.contains("Tue")){
    dias=dias+"Martes ";
}
}else{}
if(pastillas.get(position).dia.contains("Wed")){
    dias=dias+"Miercoles ";
}
}else{}
if(pastillas.get(position).dia.contains("Thu")){
    dias=dias+"Jueves ";
}
}else{}
if(pastillas.get(position).dia.contains("Fri")){
    dias=dias+"Viernes ";
}
}else{}
if(pastillas.get(position).dia.contains("Sat")){
    dias=dias+"Sábado ";
}
}else{}
if(pastillas.get(position).dia.contains("Sun")){
    dias=dias+"Domingo ";
}
}else{}
if(pastillas.get(position).dia.equals("MonTueWedThuFriSatSun")){
    dias= "Todos los días";
}
}
//mostramos el listado en textview
tv.setText("Nombre:
"+pastillas.get(position).nombre+"\n"+"Hora: "+pastillas.get(position).hora+"\n"+"Días:
"+dias);}

dias="";
return row;
}
});
ListView lv = getListView();
setTitle("Lista de Dosis");
lv.setTextFilterEnabled(true);
//en caso de clicar sobre algun elemento de la lista...
lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,int

```

```

position, long id) {
    try {
        //cogemos los parametros del objeto seleccionado para...
        (mostrarlos en la siguiente pantalla,
        //...utilizarlos para modificar o eliminar dicho objeto en el
        servidor...

        nombreselected = pastillas.get(position).nombre;
        horaselected = pastillas.get(position).hora;
        diaselected = pastillas.get(position).dia;
        colorselected=pastillas.get(position).color;
    } catch (Exception e) {
        Toast.makeText(Lista.this, "direccion", 5).show();
    }
    //nos vamos a Acciones...
    Intent intent = new Intent(Lista.this, Acciones.class);
    startActivity(intent);
}
});
}
}

```

PaginaUno.java

```

public class PaginaUno extends Activity{
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.paginauno);
        //cuando clicamos en login, nos vamos a Iniciosesion...
        final Button login = (Button) findViewById(R.id.login);
        login.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent (PaginaUno.this,Iniciosesion.class);
        startActivity(intent);
    }
});
        //cuando clicamos en registrar, nos vamos a Crear
        final Button registrar = (Button) findViewById(R.id.registrar);
        registrar.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent(PaginaUno.this, CrearUser.class);
        startActivity(intent);
    }
});
    }
}

```

Pastilla.java

```

public class Pastilla {
    public String nombre;
    public String hora;
    public String dia;
    public String userID;
    public String color;
}

```

Servicio.java

```

public class Servicio extends Service{
    List<Pastilla> pastillas;
    String USUARIO_ACTUAL = Iniciosesion.USUARIO_ACTUAL;
    private static final String TAG = "SERVICIO DE ALARMAS";
}

```

```

private Timer timer;
@Override
public IBinder onBind(Intent intent) {
    return null;
}
@Override
public void onCreate(){
    super.onCreate();
    Log.d(TAG, "Servicio de alarmas");
    timer = new Timer();
}
@Override
public void onStart(final Intent intent, final int startId){
    super.onStart(intent, startId);
    timer.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
            Log.d(TAG, "Servicio en marcha");
            HttpClient httpclient = new DefaultHttpClient();
            HttpPost httppost = new HttpPost("http://10.0.2.2:9000/guardaruser");
            try {
                //creamos el XML con la variable nueva, el nombre, el tiempo y los dias
                httppost.setEntity(new StringEntity
                ("<usuarios><usuario><nombre>USUARIO"+USUARIO_ACTUAL+"</nombre><password></pa
                ssword><ident></ident><sesion></sesion></usuario></usuarios>"));
                // Ejecutamos y capturamos la respuesta HTTP
                HttpResponse response = httpclient.execute(httppost);
                Log.d("****", response.toString());
                } catch (ClientProtocolException e) {
                // TODO Auto-generated catch block
                } catch (IOException e) {
                // TODO Auto-generated catch block
                }
            }
            try {
                // Creamos la URL de donde obtendremos la información XML
                URL url = new URL("http://10.0.2.2:9000/xml/pastillas");
                // Conseguimos un parseador SAX
                SAXParserFactory spf = SAXParserFactory.newInstance();
                SAXParser sp = spf.newSAXParser();
                // Declaramos el lector de XML
                XMLReader xr = sp.getXMLReader();
                // Creamos un objeto nueva de la clase EXAMPLEHANDLER
                HandlerPastillas myExampleHandler = new HandlerPastillas();
                xr.setContentHandler(myExampleHandler);
                // Parseamos los datos XML
                xr.parse(new InputSource(url.openStream()));
                //Fin del parseo
                pastillas = myExampleHandler.getPastilla();
            } catch (Exception e) {
                Log.d("****", "ERROR PARSEANDO");
            }
        }
        int i = 0;
        while(pastillas.size()>i){
            //recogemos hora y dia actual
            Date horaActual = new Date();
            String fecha = horaActual.toString();
            String hora= fecha.substring(11, 16);
            String dia = fecha.substring(0,3);

            //si hora y dia actual coincide con la hora de alguna dosis...

```

```

        if
        ((pastillas.get(i).hora.equals(hora))&&(pastillas.get(i).dia.contains(dia))){
            //creamos y mostramos la notificación
            String ns = Context.NOTIFICATION_SERVICE;
            NotificationManager mNotificationManager =
            (NotificationManager) getSystemService(ns);
            int icon = R.drawable.cruzroja;
            CharSequence tickerText = pastillas.get(i).nombre;
            long when = System.currentTimeMillis();
            Notification notification = new Notification(icon, tickerText,
            when);

            Context context = getApplicationContext();
            CharSequence contentTitle = "ALARMA DOSIS";
            CharSequence contentText = "Tómese la siguiente dosis:
            "+pastillas.get(i).nombre;

            //al clicar sobre la notificación, nos dirige a Home
            Intent notificationIntent = new Intent(context, Home.class);
            PendingIntent contentIntent = PendingIntent.getActivity(context,
            0, notificationIntent, 0);

            notification.flags |= Notification.FLAG_AUTO_CANCEL;
            notification.setLatestEventInfo(context, contentTitle,
            contentText, contentIntent);

            final int HELLO_ID = 1;
            mNotificationManager.notify(HELLO_ID, notification);

            i++;
        }
        else{
            i++;
        }
    }, 0, 60000);
}
@Override
public void onDestroy(){
    super.onDestroy();
    if (timer != null){
        timer.cancel();
    }
    Log.d(TAG, "Servicio parado");}
}

```

Usuario.java

```

package tfc.epsc.android;
public class Usuario {
    public String nombre;
    public String password;
    public String idd;
    public String sesion;
}

```