Proyecto Fin de Carrera

UNIVERSITAT OBERTA DE CATALUNYA

MONITORIZACIÓN DEL CONSUMO ELÉCTRICO

CONTROL DEL CONSUMO ELÉCTRICO CON OPENDOMO DESDE ANDROID

PRESENTADO POR:
Iosu González Albín

DIRIGIDO POR:
Antoni Morell Pérez

Valencia, Junio de 2014

CONTENIDOS

INTRODUCCIÓN

Esta memoria forma parte de mi Proyecto de fin de carrera, realizado entre Marzo y Junio de 2014 en la Universidad Oberta de Catalunya (UOC).

TÍTULO

MONITORIZACIÓN DEL CONSUMO ELECTRICO: Control del consumo con ODEnergy desde la plataforma Android.

OBJETIVOS DEL PROYECTO DE FIN DE CARRERA

El objetivo es realizar un programa que permita sencillamente desde cualquier lugar a través de un móvil o tableta controlar el consumo de una red eléctrica en un domicilio particular, como será el caso de las pruebas realizadas, como de una empresa.

La idea básica del programa es saber en todo momento si la red está activa o ha caído, saber el consumo en tiempo real para poder controlar el consumo del sistema y detectar cualquier anomalía. Adicionalmente se establecerán más alertas como una destinada a avisar de que estamos alcanzando el límite máximo de potencia de la red o se puede implementar avisos por gasto máximo en €.

Esta idea surge al conocer esta empresa: www.opendomo.es , y su sistema ODEnergy que permite la monitorización y realiza unas estadísticas sobre el consumo. Este sistema dispone de varios mecanismos de acceso a la información que capta, mediante su configuración como cliente (envía los datos a una página web de consulta) o como servidor, el caso que emplearemos, con el cual se puede acceder a los datos en tiempo real mediante un sistema de lectura de un fichero XML almacenado en la memoria del dispositivo que se conecta en el panel eléctrico de casa.

Como ejemplos concretos de utilización podemos ver los siguientes:

- Saber en todo momento el consumo de la red y detectar anomalías o aparatos que se han quedado encendidos (calefacción por la noche, luces por el día, etc..) o consumen en exceso.
- Detectar caídas de la red para poder acudir al domicilio lo antes posible para rearmar el diferencial o el ICP o avisar a la compañía eléctrica si es un problema ajeno y evitar problemas asociados: arrancar el servidor en una empresa, encender el congelador en un hogar, etc..
- Detectar excesos de consumo que vayan a generar una caída del sistema por sobrepasar el límite del ICP instalado y poder desconectar elementos a tiempo antes de que caiga el sistema

También quería tener una aplicación que me informara continuamente de la potencia que estoy consumiendo para saber la necesidad real de potencia ya que la parte fija de la factura eléctrica cada vez tiene más peso y el termino de potencia (la potencia máxima contratada) ha subido un 18% solo en una de las subidas de este año[1].

[1] https://www.facua.org/es/noticia.php?Id=8235

VIABILIDAD

Este proyecto es muy barato de implementar ya que solo requiere de un sistema Opendomo Ethernet o Wifi instalado en el panel eléctrico del hogar o la empresa y de cualquier teléfono móvil o tableta con Android. El coste actual del dispositivo empleado es de 179.90€[2].

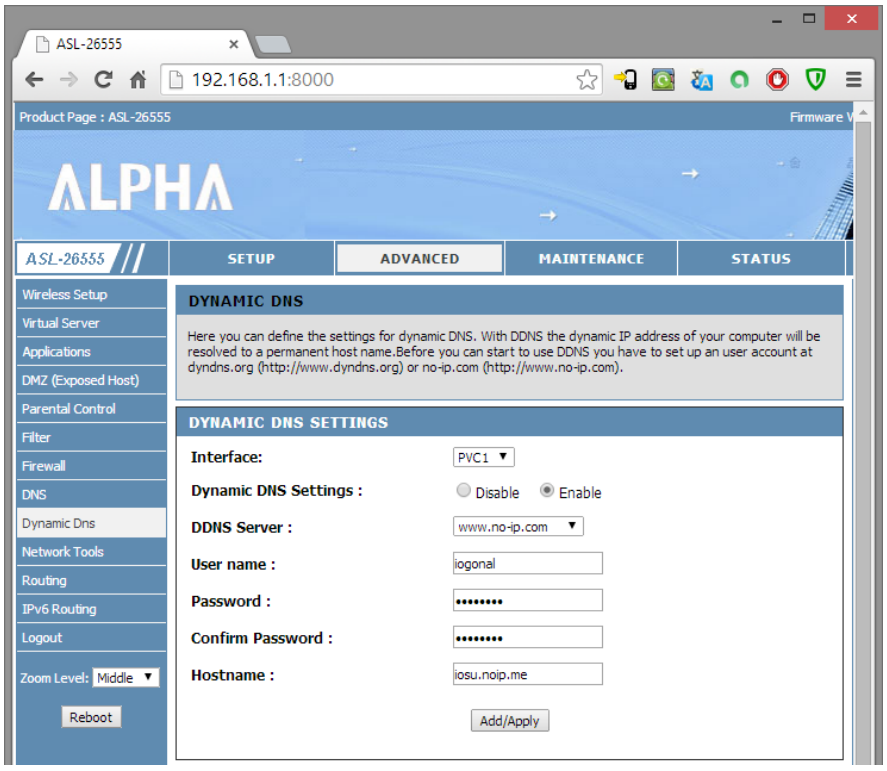
http://www.opendomo.es/products/ODEnergyM63EW

Este dispositivo dispone de conexiones Ethernet y Wifi y es apto para instalaciones monofásicas como las de cualquier hogar. Se instala en cualquier regleta estándar junto al ICP (Interruptor de Control de Potencia[3]) y resto de conexiones eléctricas. Para entornos corporativos o diferentes necesidades se disponen de otros aparatos similares, para instalaciones trifásicas o grandes consumidores:

http://www.opendomo.es/products/ODEnergyT63EW;http://www.opendomo.es/products/ODEnergyT200EW;http://www.opendomo.es/products/ODEnergyTN5EW

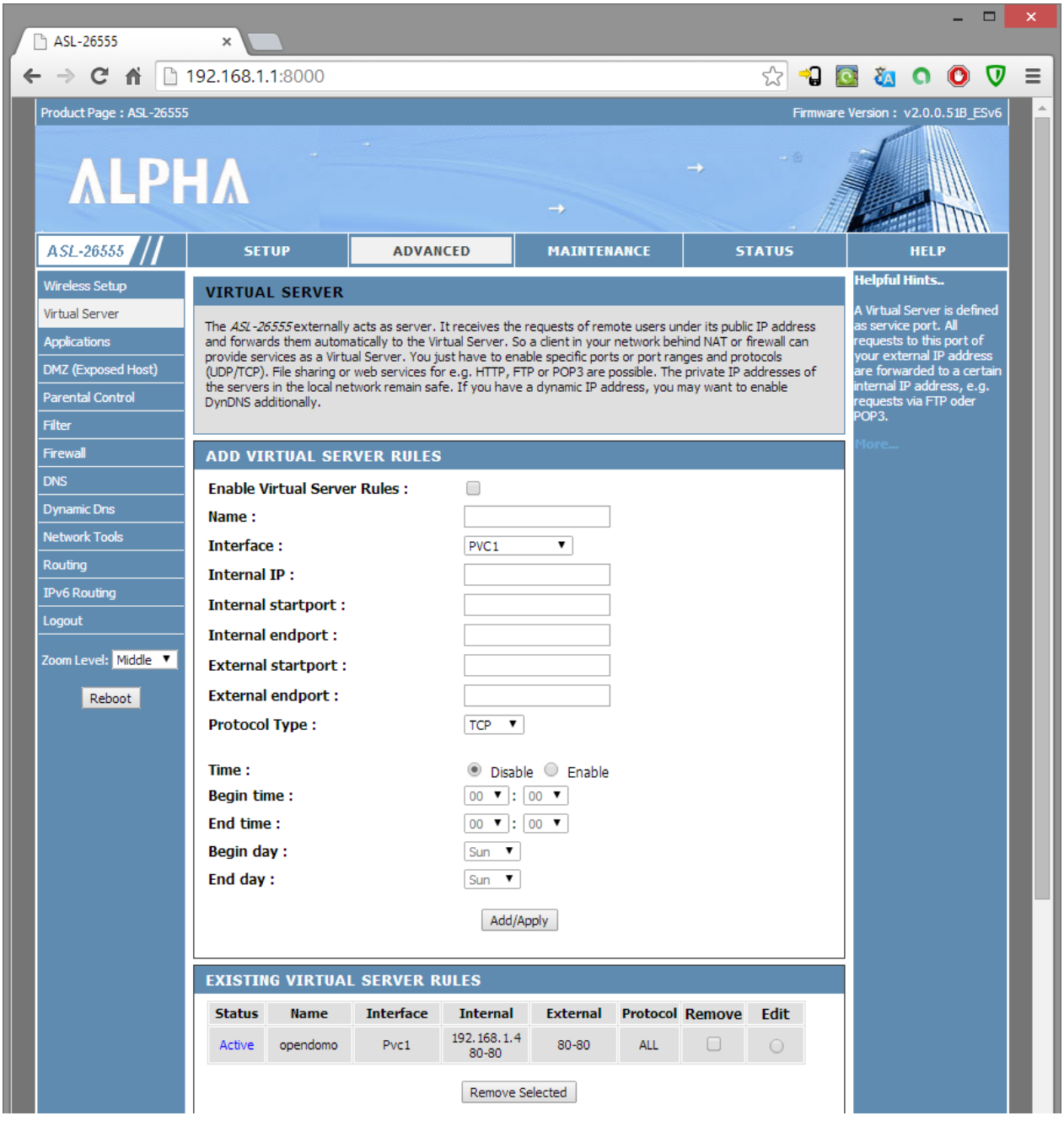
ACCESO VIA INTERNET

Además de la instalación del ODEnergy y de una conexión a internet, para el acceso desde fuera de cada necesitaremos configurar el router de casa con un sistema de DNS gratuita si no tenemos una ip fija. En mi caso he utilizado el servicio noip porque además de ser gratuito, mi router tiene una función para modificar el servicio DNS cada vez que hay un cambio de IP.



[2] http://www.efimarket.com/opendomo-odenergy-m63ew-monofasico-wifi
[3] http://es.wikipedia.org/wiki/Interruptor_de_control_de_potencia

Una vez configurado este servicio, ya tendremos enlazado nuestro Router mediante su dirección IP pública a la dirección DNS virtual iosu.noip.me, pero necesitaremos redireccionar todo el tráfico que llegue a nuestro ODEnergy, el cual en mi red local tiene dirección IP 192.168.1.4.



1 . EL SISTEMA ODENERGY

INSTALACIÓN

Para la realización de este proyecto se ha empleado el sistema de control de consumo eléctrico Opendomo, en concreto el modelo ODEnergy M63EW(Figura 1), el cual permite funcionamiento mediante lectura de datos a través de un cable Ethernet o inalámbricamente a través del protocolo Wifi.

Figura 1- ODEnergy M63W

Figura 2- Sonda amperimétrica

La lectura de la corriente se realiza mediante una sonda(Figura 2), la cual es un transformador de intensidad de conexión directa (Z80-63A) por la que se pasa el cable de fase por su agujero central y después conectaremos a nuestro ODEnergy(Figura 3).

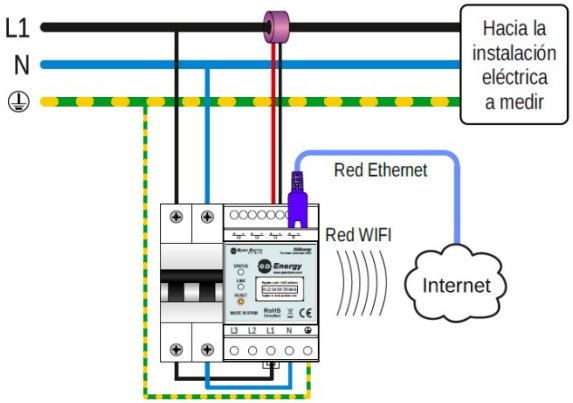


Figura 3 - Conexión a la red eléctrica

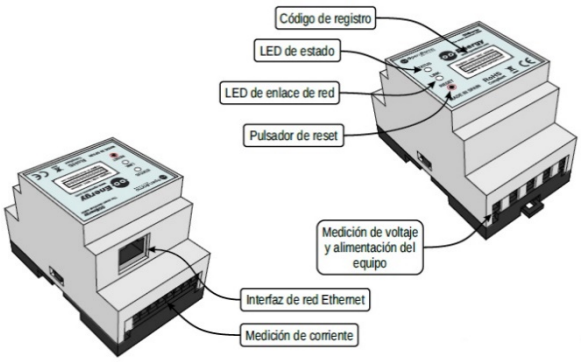


Figura 4- Elementos del ODEnergy

El ODEnergy M63W se conecta en la caja eléctrica estándar de casa a los cables de fase, neutro y tierra(Figura 3). Esta conexión se realiza antes de los diferenciales magnetotérmicos y tras el ICP (Interruptor de Control de Potencia) en el carril DIN, ocupando tres posiciones

CONFIGURACIÓN ODENERGY

Una vez instalado en nuestro cuadro eléctrico el aparato viene por defecto configurado en la dirección 169.254.0.40[4], y podemos conectarnos mediante un cable Ethernet (Figura 4) para configurarlo con nuestra propia red de cable(Figura 6) o configurar la red Wifi(Figura 5).

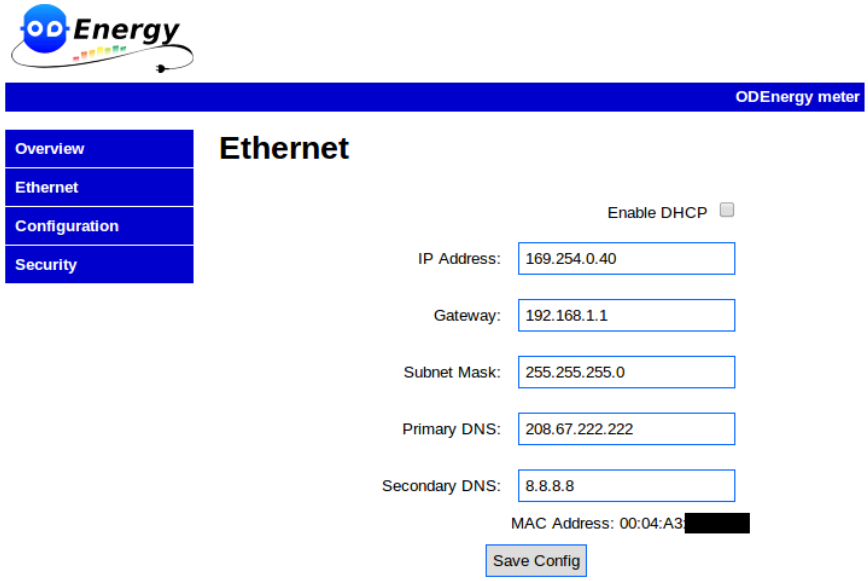


Figura 6- Configuración de red ODEnergy

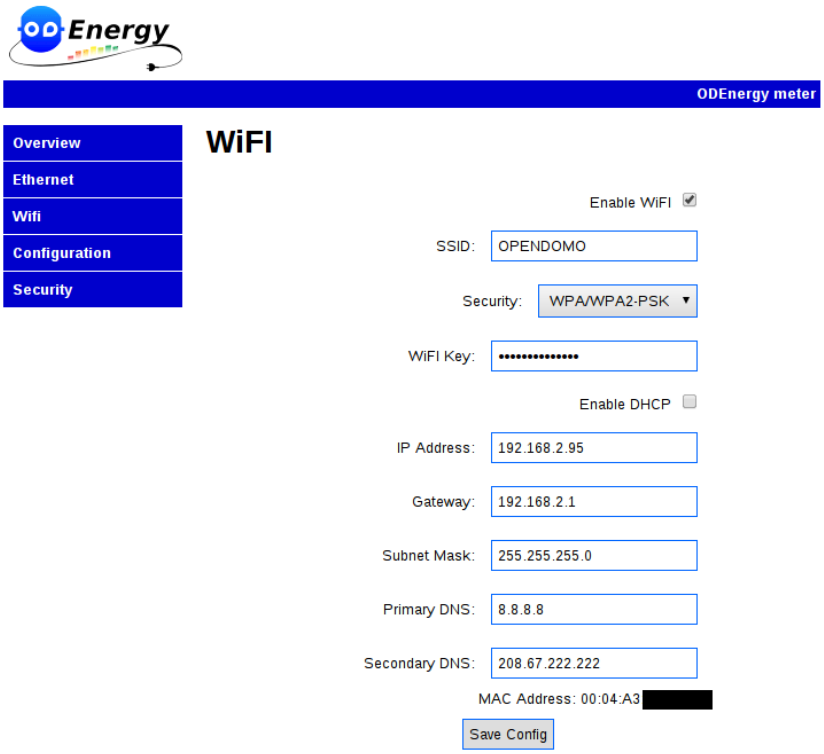


Figura 5 - Configuración WIFI ODEnergy

[4] http://www.opendomo.com/wiki/index.php/ODEnergy_EW

Una vez configurado con nuestra red ya podemos acceder inalámbricamente al ODEnergy (Figura 8) y consultar los datos de lectura que realice ya que el sistema incorpora un servidor web, como ya hemos visto, donde acceder a la configuración pero además podremos acceder a un archivo XML(Figura 10) que el aparato actualiza cada 5 segundos todos los datos de nuestro consumo. Para esto deberemos configurarlo en modo servidor(Figura 8)

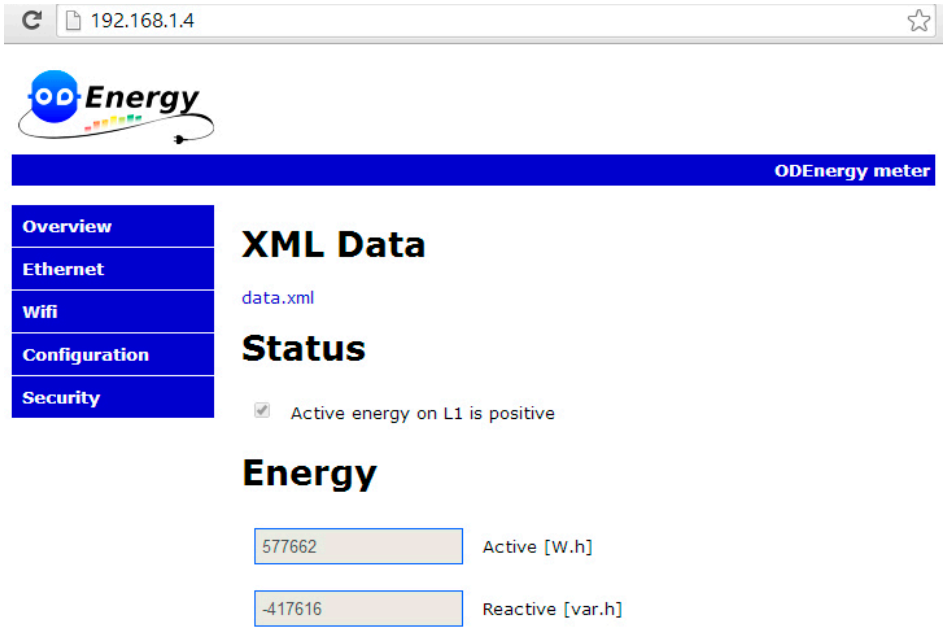


Figura 8 – Página Web generada

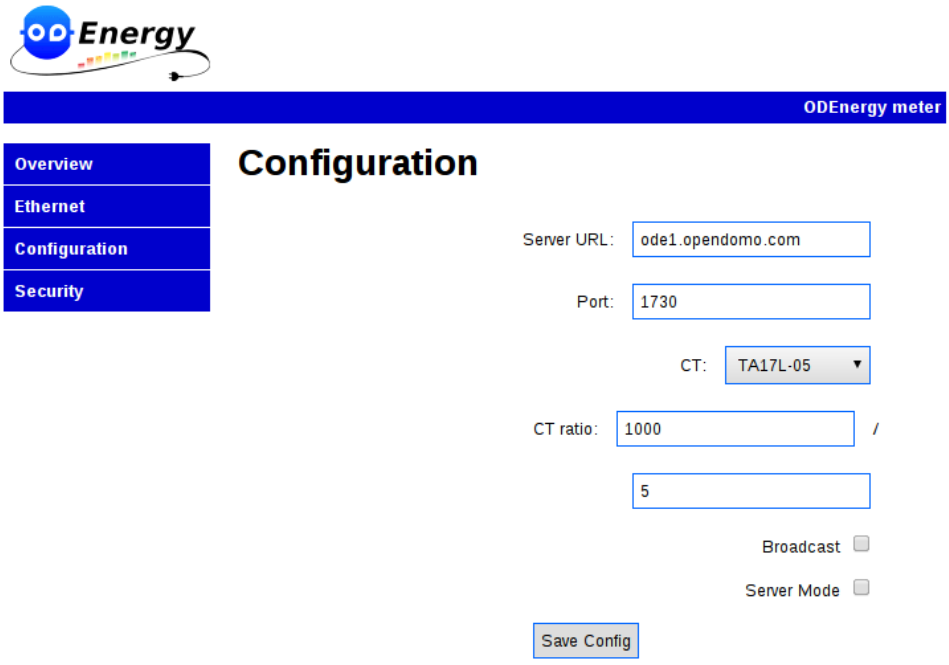


Figura 7 - Configuración ODEnergy como Servidor

192.168.1.4/data.xml

192.168.1.4/data.xml

This XML file does not appear to have any style information associated with it.

```
▼<device uid="0004A381265E" version="1.0.0_WM">
  ▼<accumulated>
     <boottime>1402890404</boottime>
     <timestamp>1403370312</timestamp>
     <active_L3>0</active_L3>
     <active_L2>0</active_L2>
     <active_L1>577775</active_L1>
     <reactive_L3>0</reactive_L3>
     <reactive_L2>0</reactive_L2>
     <reactive_L1>-417666</reactive_L1>
   </accumulated>
  ▼<instant>
     <voltage_L3>0</voltage_L3>
     <voltage_L2>0</voltage_L2>
     <voltage_L1>231</voltage_L1>
     <current_L3>0</current_L3>
     <current_L2>0</current_L2>
     <current_L1>2480</current_L1>
     <current_N>0</current_N>
     <active_L3>0</active_L3>
     <active_L2>0</active_L2>
     <active_L1>496</active_L1>
     <reactive_L3>0</reactive_L3>
     <reactive_L2>0</reactive_L2>
     <reactive_L1>-297</reactive_L1>
   </instant>
 </device>
```

Figura 10 - fichero XML generado

Tras configurarlo y registrar su MAC (código de registro en Figura 4) en la página web del fabricante podremos acceder a una web de control del gasto energético(Figura 9):

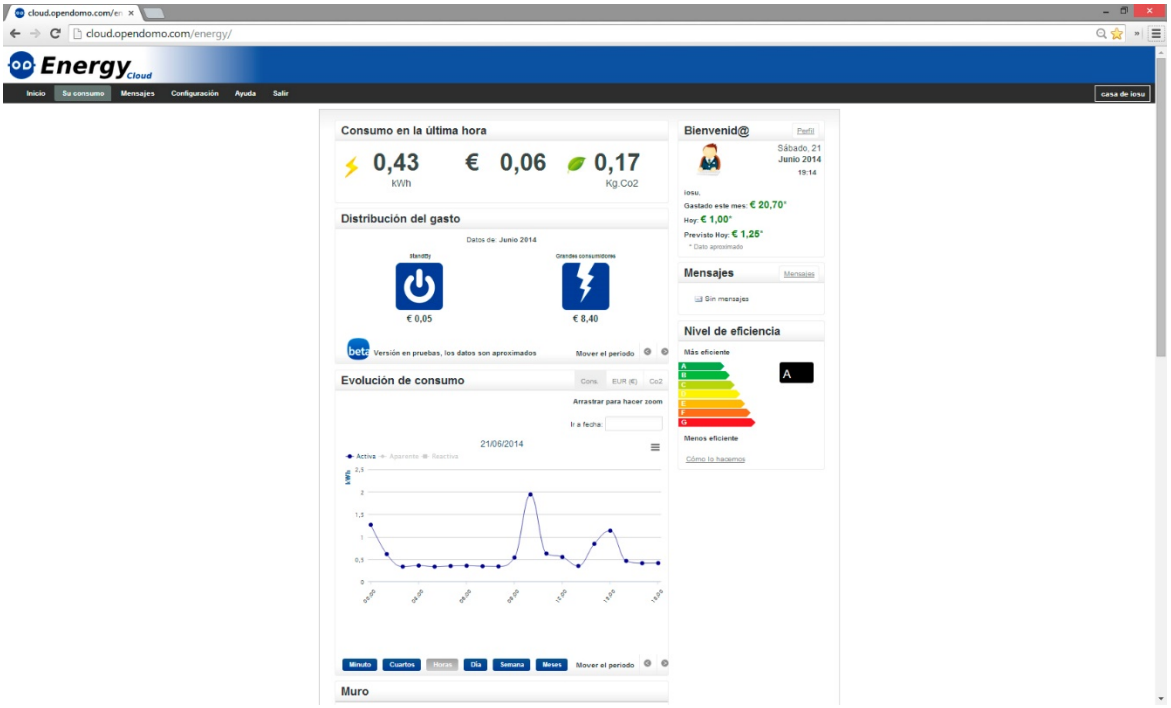


Figura 9 - Consumo Mensual

2. SOFTWARE NECESARIO

Para implementar la aplicación necesitaremos un terminal (teléfono móvil o tableta) con Sistema Operativo Android donde ejecutar después la app creada, un entorno de programación donde escribir el código y el software Java JDK (Java Development Kit) y el JRE5 (Java Runtime Environment).

ENTORNO DE PROGRAMACIÓN

El entorno elegido para el desarrollo de la aplicación ha sido el programa "Android Studio" desarrollado por Google, en detrimento del más conocido ECLIPSE.

Android Studio está basado en la plataforma INTELLIJ IDEA de desarrollo de Java y el sistema de herramientas de construcción Gradle. Este programa tiene poco tiempo de vida (salió a la luz el año pasado) y se encuentra en una fase "preview" todavía, pero resulta muy cómodo de utilizar y viene todo integrado en un mismo programa, como las herramientas SDK TOOLS y AVD Manager que luego explicaré.

El entorno Eclipse también lo podríamos haber usado, ya que es muy similar, mediante la descarga del propio programa y las herramientas que lo hacen compatible con Android, el Eclipse Plugin[6] de Android y las SDK Tools.

Android Studio es un entorno (Figura 11) muy completo y amigable, tipo WYSIWYG (What You See Is What You Get) similar a Eclipse y otros conocidos como Visual Studio, aunque también podemos editar, los layout por ejemplo, en modo texto. En el mismo entorno podemos ver el árbol del proyecto, mensajes de la aplicación, acceso a un terminal, así como editar nuestros diseños de Layouts (Figura 12) e incluso editar imágenes PNG, etc...

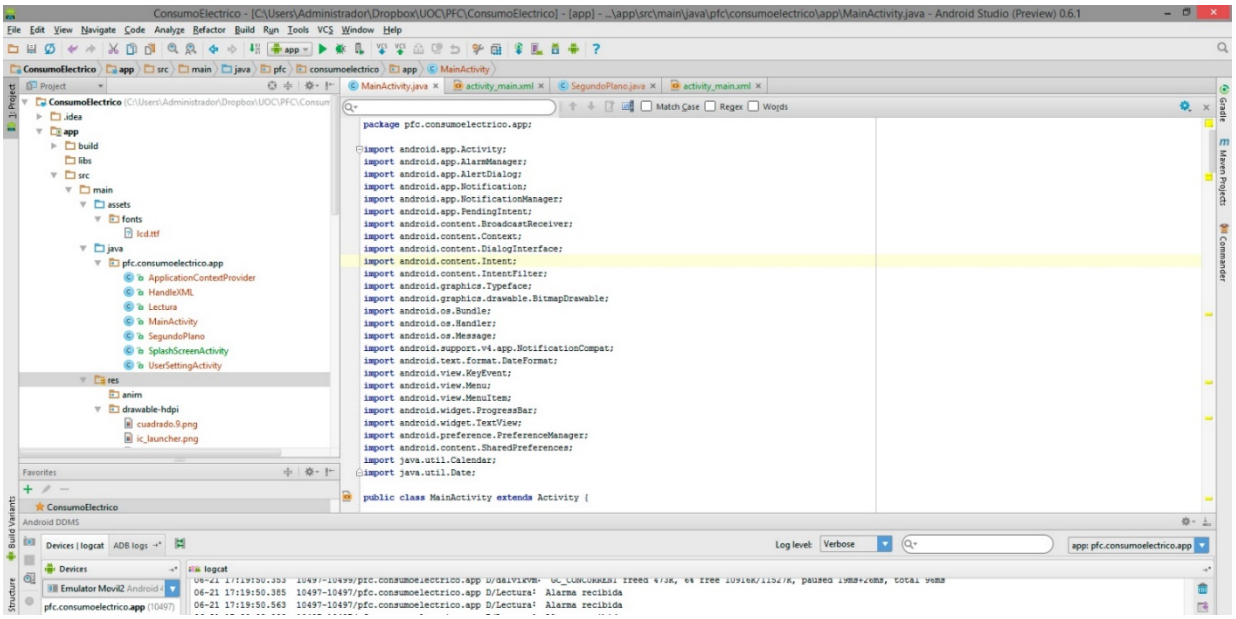


Figura 11- Entorno Android Studio

[5] http://www.java.com/es/download/index.jsp
[6] http://developer.android.com/sdk/installing/installing-adt.html

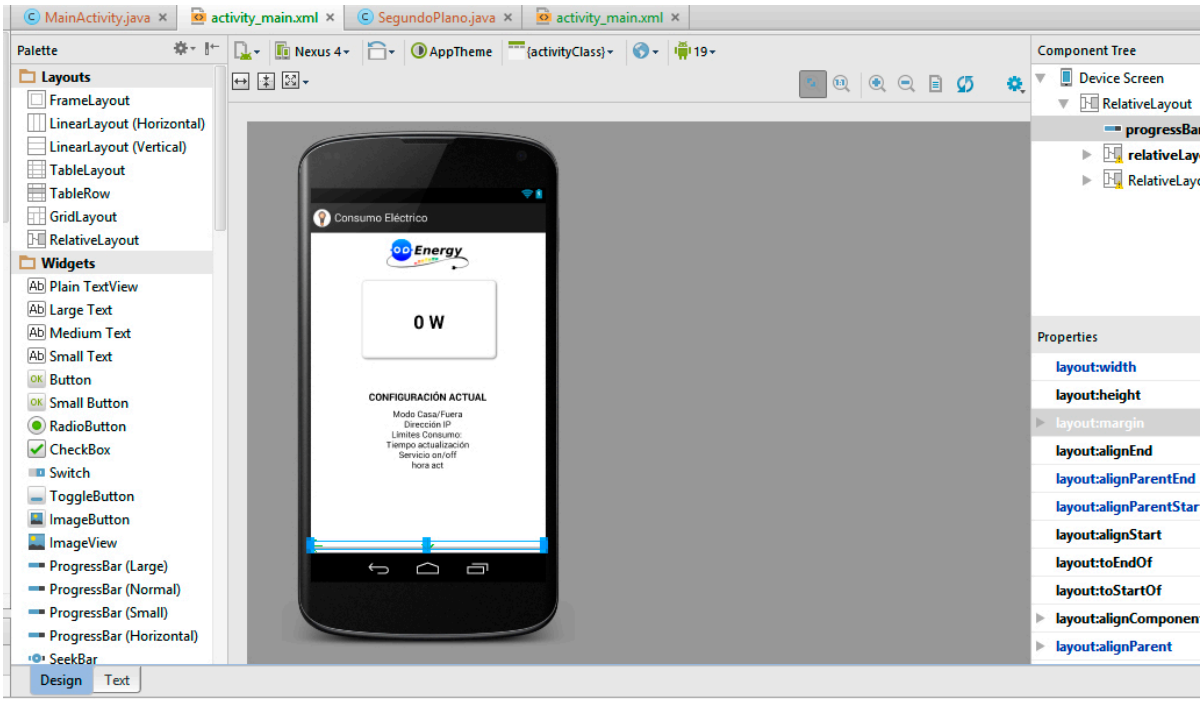


Figura 12 - Diseño WYSIWYG

Con la herramienta AVD (Android Virtual Device Manager) disponemos de terminales emulados donde ejecutar nuestra aplicación y que podemos generar con diferentes configuraciones de Hardware y de S.O instalado (Android 4.0, 4.1.2, etc..).

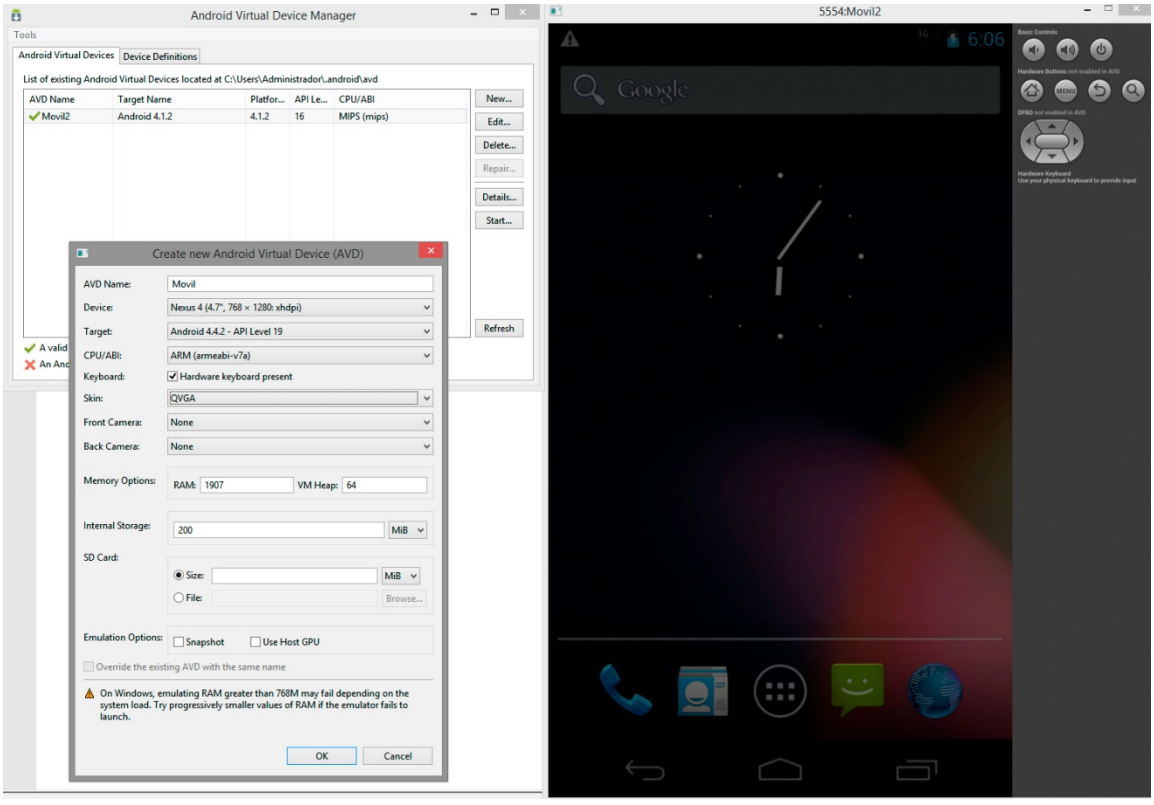


Figura 13 - AVD Manager

Como comentaba desde Android Studio podemos manejar el repositorio del SDK de Android, ya que viene integrado en el programa el Android SDK Manager (Figura 14).

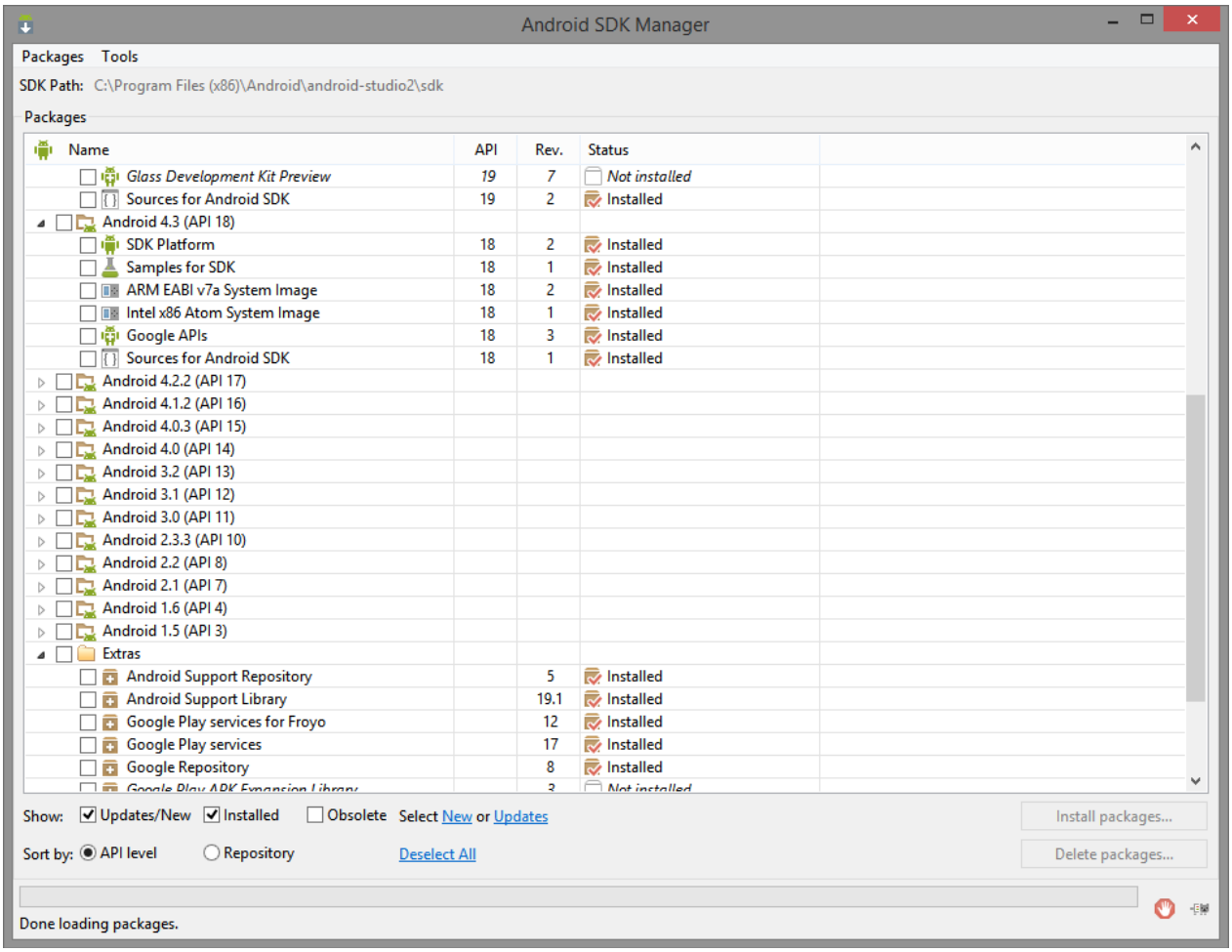


Figura 14 - Android SDK Manager

También tenemos otras funcionalidades muy interesantes como manejar un sistema de control de versiones (VCS), por ejemplo mediante Mercurial.

JAVA

Las aplicaciones para el sistema operativo Android se programan en el lenguaje de programación Java. Java es un lenguaje orientado a objetos con una sintaxis derivada de C y C++, creado en el año 1995 por la empresa Sun Microsystems y actualmente es propiedad de la empresa Oracle.

Necesitaremos tener instalado en nuestro equipo el JRE7 (Java Runtime Environment), necesario para ejecutar las aplicaciones desarrolladas en lenguaje Java, independiente del JDK, para el desarrollo de aplicaciones pues incorpora un compilador.

[7] http://www.java.com/es/download/index.jsp

3. PROYECTO EN ANDROID

Una vez instalado el IDE Android Studio y el software adicional mencionado, Java JRE y JDK y Android SDK, procedemos a la creación de nuestro proyecto. Antes de comenzar con el proyecto conviene explicar las razones de la elección de dicha plataforma.

¿POR QUÉ ANDROID?

He elegido la plataforma Android por ser usuario de la misma y conocerla más a fondo y principalmente porque actualmente es la plataforma más usada en todo el mundo y especialmente en España, con más del 90% (Figura 15- Market Share por S.O. - Junio 2014 (f. GFK)) de cuota.

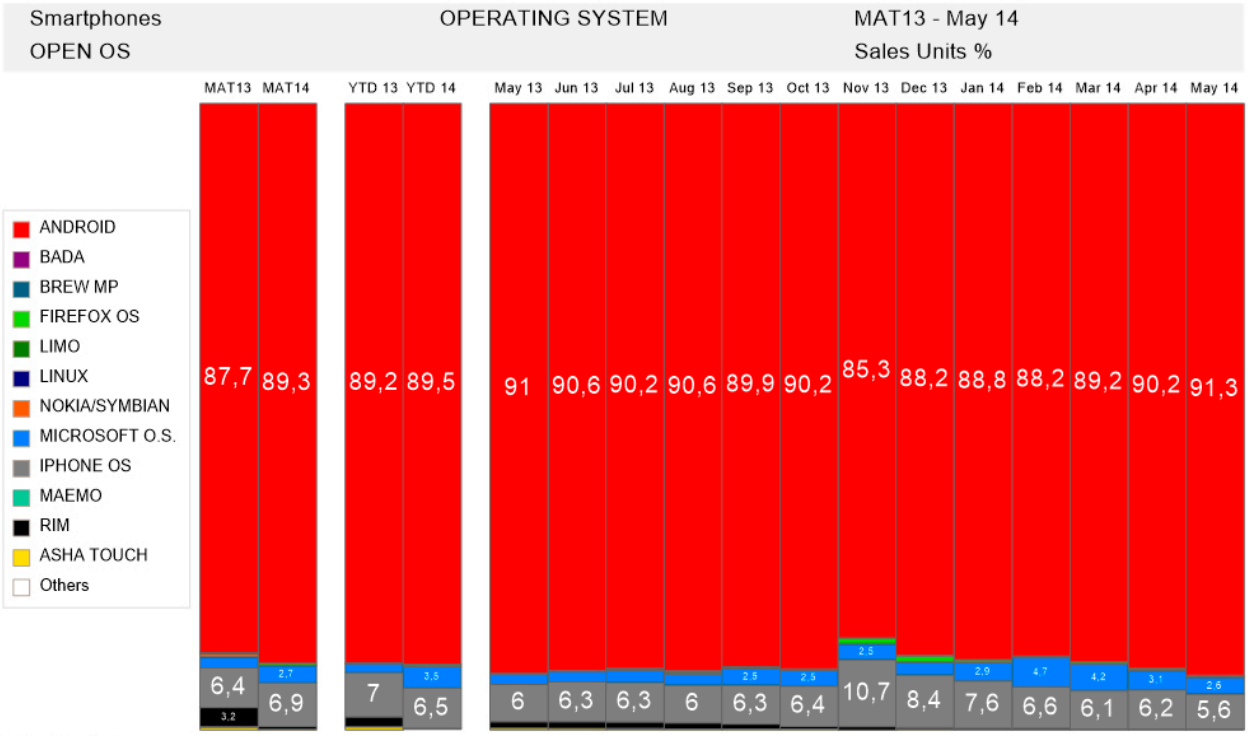


Figura 15- Market Share por S.O. - Junio 2014 (f. GFK)

VERSIONES DE ANDROID

Las versiones de Android (ANEXO 1) reciben, en inglés, el nombre de diferentes postres. En cada versión el postre elegido empieza por una letra distinta, conforme a un orden alfabético (Figura 16). Para el desarrollo de la aplicación nos hemos centrado a partir de la API 14 (Ice Cream Sandwich), puesto que esta versión junto a las versiones siguientes representan más del 85% de los terminales Android (Figura 17) y la evolución de las versiones anteriores es decreciente (Figura 18).

Figura 16 - Versiones de Android

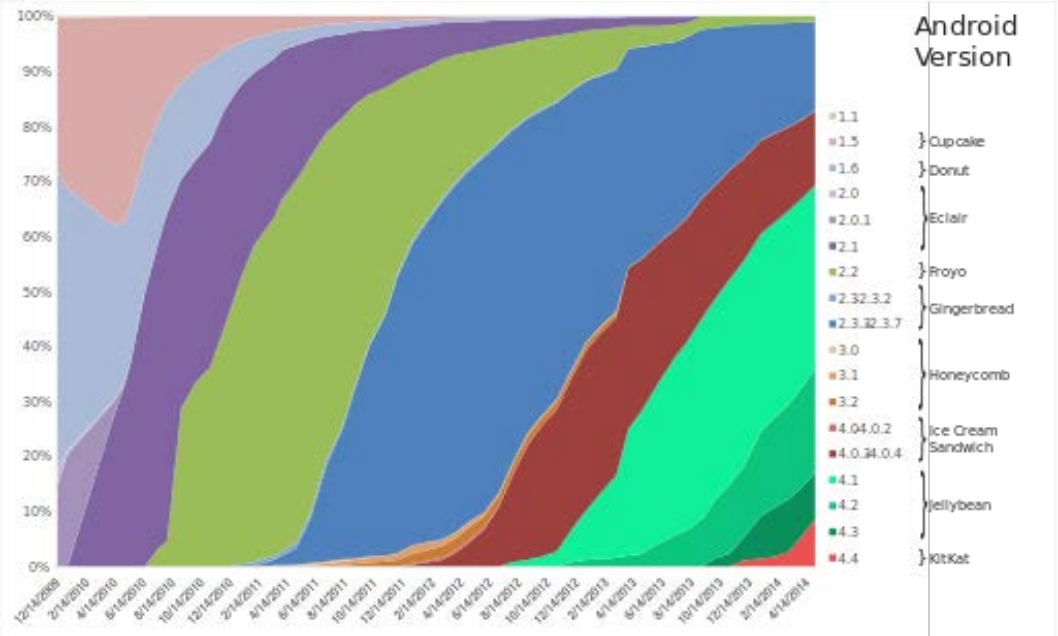


Figura 17- Evolución fragmentación Android (Wikipedia)

Version	Codename	API	Distribution
2.2	Froyo	8	0.8%
2.3.3 - 2.3.7	Gingerbread	10	14.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	12.3%
4.1.x	Jelly Bean	16	29.0%
4.2.x		17	19.1%
4.3		18	10.3%
4.4	KitKat	19	13.6%

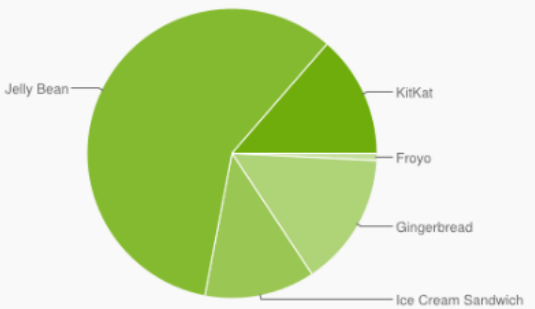


Figura 18 - Fragmentación Android a Junio de 2014 – Android Developer

CREACIÓN DE UN PROYECTO

El primer paso es abrir Android Studio y crear un nuevo Proyecto, al que damos nombre y definimos el campo de compañía, y automáticamente nos genera un nombre para el paquete de aplicaciones que se van a generar (Figura 20 - Creación de un nuevo proyecto

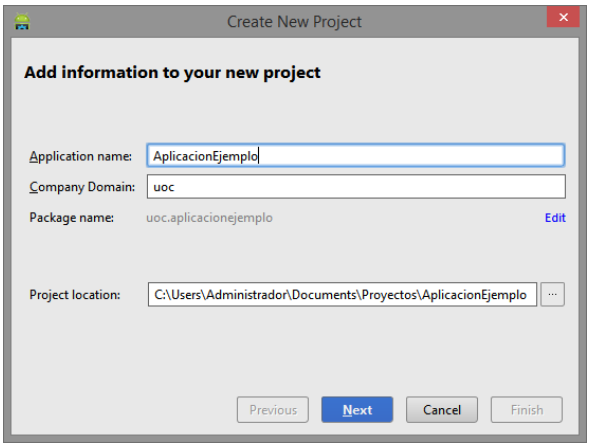


Figura 20 - Creación de un nuevo proyecto

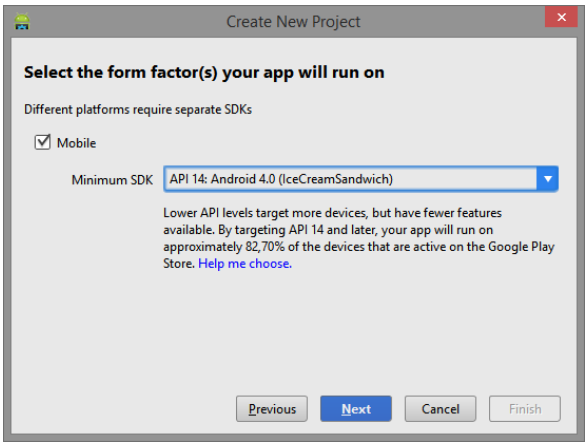


Figura 19 - Selección API mínima

). Definimos también a qué plataformas estará dirigido el programa (Figura 20), especificando a partir de qué nivel de aplicación (API[8]) mínimo será compatible

El programa nos ofrece varias opciones de aplicaciones tipo para seleccionar (Figura 21)

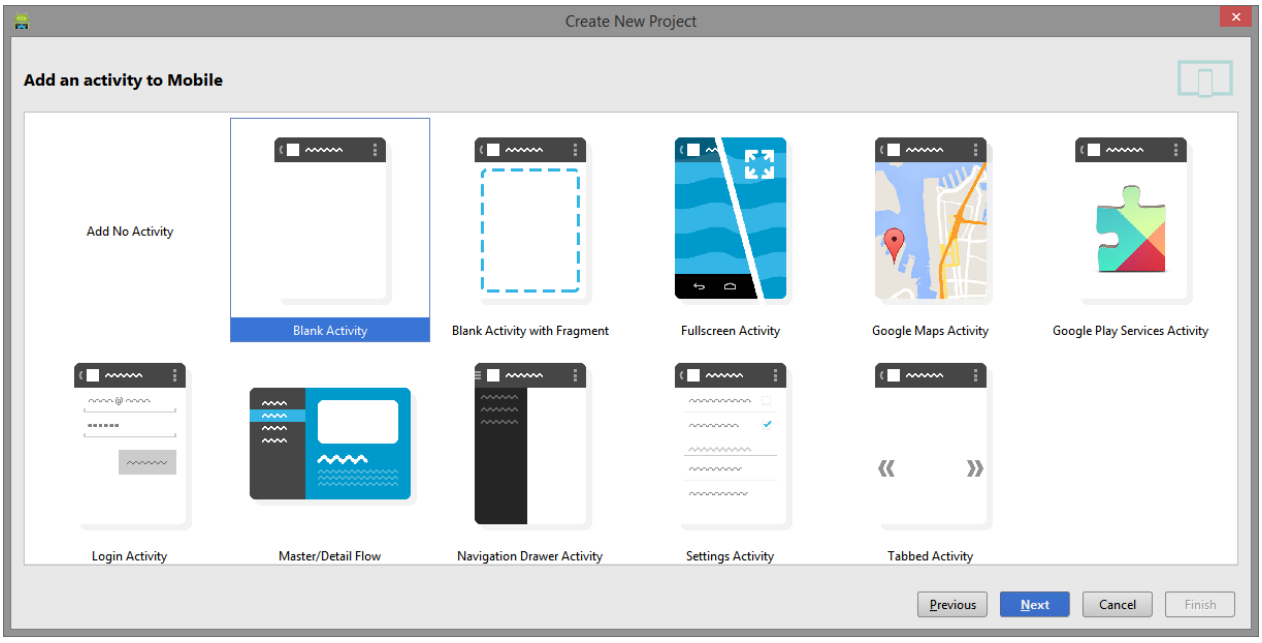


Figura 21 - Selección de tipo de actividad

[8] http://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android

Elegimos por ejemplo la actividad en blanco y ya estaríamos listos para comenzar a configurar nuestra aplicación.

ESTRUCTURA DE UN PROYECTO

Cualquier aplicación en Android sigue una misma estructura básica (Figura 22), que se compone del código fuente en sí, archivos de recursos y vistas, librerías de código y el Android Manifest que veremos más adelante y otros directorios como librerías, siendo estos los más importantes ya que debemos manejarlos:

DIRECTORIO SRC

Aquí tendremos la llamada lógica de aplicación, es decir, todas las clases programadas en JAVA.

ASSETS

Directorio importante para manejar archivos auxiliares como fuentes. En eclipse basta con incorpor las fuentes al directorio, pero en Android Studio es necesario configurar también Gradle (ver nota).

DIRECTORIOS RES

Se encuentran todos los archivos con los recursos que usa la aplicación: Las imágenes, archivos de idiomas, estilos, etc..:

- **Drawable**: donde tendremos todas las imágenes que utiliza la app, divididas por carpetas que contienen las imágenes en distintas resoluciones y tamaños que se usarán dependiendo el dispositivo usado.
- **Layout**:Aquí se encuentran las distintas "pantallas" de la aplicación, es decir, los archivos xml con las interfaces asociadas a las activities.
  En el directorio layout-land pondremos la vista de la actividad en apaisado por si fuera necesario.
- **Values**: Carpeta con los xml de contenido de la app, como constantes, textos multidioma y otra seria de configuraciones (estilos, colores, etc..)

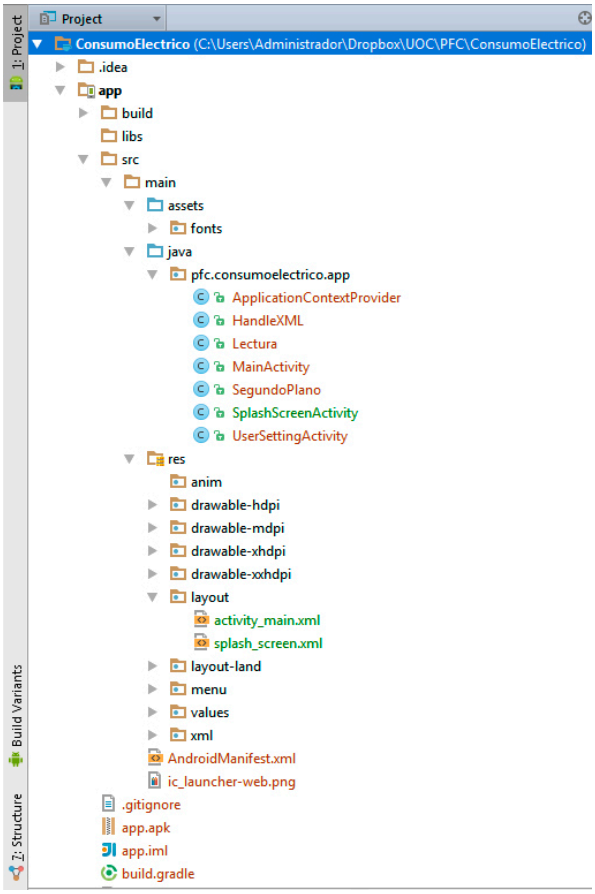


Figura 22 -Estructura de un proyecto

Debo destacar que para compilar nuestra aplicación hay que emplear una herramienta llamada Gradle, o el script Gradle Wrapper[9] en el caso de Android Studio, que es la encargada de automatizar la construcción de nuestros proyectos, por lo que tenemos otros archivos muy importantes que hay tener en cuenta:

- app.iml: configuración de directorios de Gradle, por ejemplo los "assets"
- build.gradle : configuración de opciones de Gradle, por ejemplo API usada y directorio de "assets"

Por último el archivo app.apk es el instalador que generamos necesario para instalar desde cualquier dispositivo nuestra aplicación.

[9] http://www.gradle.org/docs/current/userguide/gradle_wrapper.html

ANDROID MANIFEST

En la raíz de nuestro proyecto tendremos un archivo muy importante llamado **AndroidManifest.xml**, y es un archivo de configuración donde podemos aplicar las configuraciones básicas de nuestra app tales como nombre de la aplicación, versión o API utilizada.

En este archivo definiremos los permisos que la aplicación usará, por ejemplo en el caso de una aplicación que hiciera uso de las llamadas de teléfono que se le permitiera, o en nuestro caso acceso a internet, aviso que obtendremos al instalar la aplicación (Figura 23).

```
package="pfc.consumoelectrico.app" >
  <uses-permission
    android:name="android.permission.INTERNET"/>
```

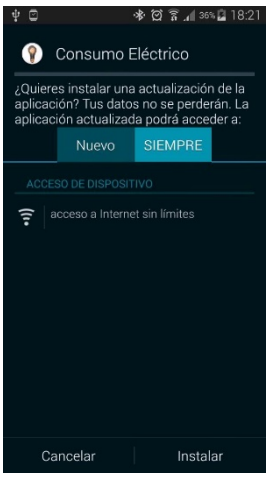


Figura 23 – Instalación de la app

También definiremos en este archivo los "intent", acciones, a los que responderá la aplicación como por ejemplo MAIN y SEND, por defecto las que se utilizan para arrancar la aplicación.

```
<activity
        android:name="pfc.consumoelectrico.app.SplashScreenActivity"
        android:label="@string/app_name">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
        </intent-filter>
      </activity>
```

Como se puede ver cada actividad (a continuación veremos para que se utilizan) tiene que estar indicada en el manifest, con el nombre de la clase

```
<activity
        android:name=".UserSettingActivity"
        android:label="Ajustes"/>
```

En el caso de los métodos que se usan como receptores de anuncios broadcast, desde el Android manifest debemos declarar a que tipo de anuncios debe de contestar cada clase:

```
<service android:name=".SegundoPlano" />
    <receiver android:name=".Lectura" />
```

ACTIVITIES

El componente más básico de Android es el llamado Activity. Una *activity* representa una pantalla o interface que puede recibir datos del usuario. Al realizar un cambio de una pantalla a otra se cierra o pausa la *activity* actual y se arranca la *activity* nueva. Durante este cambio es posible pasar datos de la primera a la segunda *activity* o salvar los datos para retomarlos al volver a esta pantalla, como por ejemplo al entrar en la pantalla de configuración del programa.

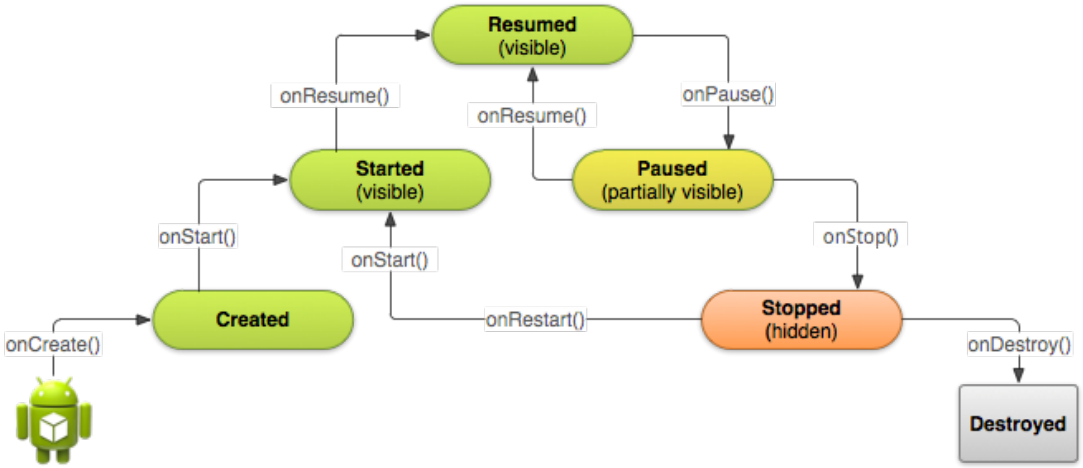


Figura 24 - Ciclo de vida de una Activity

En la Figura 24 podemos ver el ciclo de vida de una actividad, comienza cuando se crea, pasando a un estado de pausa cuando deja de visualizarse y pudiendo volver a la misma mediante la llamada al método onResume() . También podemos pararla y volverla a arrancar.

El sistema Android almacena los datos relativos a las aplicaciones que permanecen abiertas mientras que tenga memoria, con un método de reciclaje automático de las actividades.

Las aplicaciones que permanecen "dormidas" aparecen en la pila de aplicaciones en uso o recientes y su acceso a las mismas será más rápido que si se arrancan de cero y además incorporan los datos que ya estuviera mostrando.

En algunos terminales se accede incluso mediante pulsación de botones, por ejemplo en la mayoría de terminales de la marca Samsung se accede a esta lista pulsando prolongadamente el botón "home", o en el caso del nuevo Samsung Galaxy S5 se dispone de un botón específico para acceder a este listado.

Como se puede observar en la Figura 25 , además en la vista previa que se muestra podemos ver datos de la aplicación en su último estado, como la última pantalla visualizada en la aplicación consumo eléctrico o el último contacto visualizado.

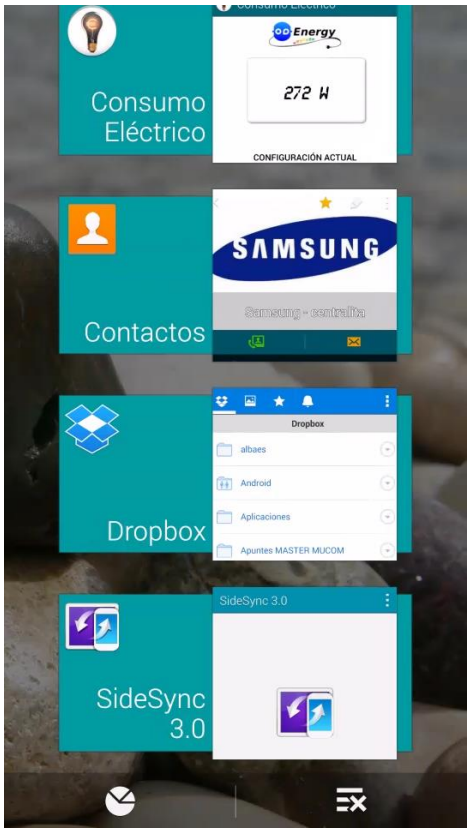


Figura 25 - aplicaciones recientes

Este concepto es básico para entender que a diferencia de las aplicaciones que se ejecutan en un PC, éstas no se cierran si no se especifica la llamada al método onDestroy() como veremos en la aplicación y que podemos salvar la última vista si queremos recuperarla al reanudar la aplicación

Es importante también resaltar que cuando generemos una actividad en un layout vertical si no salvamos los datos de estado cuando giremos la pantalla la aplicación realmente generará una nueva actividad sobre el layout-land con todos los datos a 0. Esta circunstancia la debemos tener en cuenta también durante el desarrollo de una aplicación.

Para arrancar una actividad se puede hacer de varias maneras, por ejemplo:

- Mediante el método startActivity()
- Mediante un intent desde otra actividad
- "Despertando" a la actividad, por ejemplo mediante un intent (acción) de arranque completado si se quiere que la aplicación se ejecute automáticamente al arrancar el terminal.

La forma en que las actividades se muestran en pantalla se configura mediante un archivo XML, ya sea en modo texto o en modo gráfico, como ya contábamos antes (WISIWYG).

4. DISEÑO DE LA APP

LA APLICACIÓN CONSUMO ELÉCTRICO

La aplicación diseñada será lo más simple posible para que sea funcional y sirva para el propósito para el que se ha creado, monitorizar el consumo eléctrico y generar avisos de consumo.

Los elementos fundamentales serán:

- Cuadro Monitorización Consumo
- Botón Activación servicio automático
- Botón Comprobación manual
- Ajustes de la aplicación

A continuación explicaré el modelo empleado para el funcionamiento de la clase y os distintos elementos que componen el proyecto de la aplicación.

MODELO DE LA APLICACIÓN

El modelo UML usado de forma esquemática será el de la Figura 26 y consta de una actividad (luego veremos más adelante este concepto) principal MainActivity que es la encargada de mostrar toda la información y llamar al resto de clases, servicios y actividades.

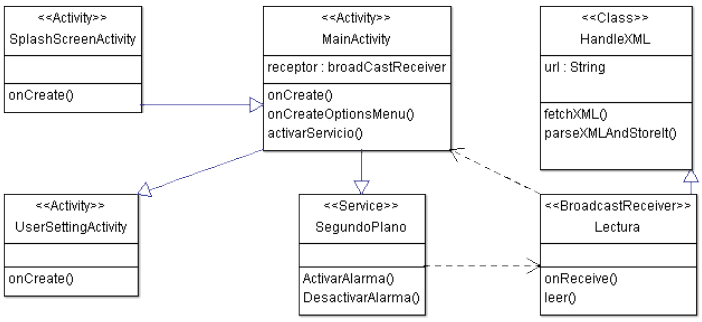


Figura 26 - Modelo UML

La activity SplashScreenActivity simplemente es una pantalla de arranque que tiene dos funciones: una de presentación de la aplicación al arrancarse y mientras tanto se van cargando los elementos de la actividad principal.

Tendremos otra actividad para configurar los ajustes de la aplicación, UserSettingActivity, donde guardamos todos los ajustes.

La Clase Lectura es de tipo BroadcastReceiver, es decir que se activará cuando reciba un aviso de que debe de realizar la operación de lectura, la cual efecturá a través de la clase HandleXML que por comodidad y por limpieza de código se ha creado en una clase independiente aunque podría haberse implementado su funcionalidad dentro de esta misma Clase.

COMPONENTES DE LA APLICACIÓN

A continuación voy a exponer los elementos clave de la aplicación para que funcione correctamente, como es el Android Manifest, los layouts, los ficheros gradle, etc..

ANDROID MANIFEST

Lo principal que tenemos que declarar en este fichero, además de las cosas anteriormente comentadas, como el nombre del paquete y el número de la versión del programa, son los intents ante los que reaccionará nuestra aplicación, como expuse las acciones SEND y MAIN. Al poner estas acciones dentro de la activity SplashScreenActivity, esto indica que es esta clase la que reaccionará ante estos intents

```
<activity
        android:name="pfc.consumoelectrico.app.SplashScreenActivity"
....
<action android:name="android.intent.action.MAIN" />
...
<action android:name="android.intent.action.SEND" />
```

También indicaremos que el servicio SegundoPlano tiene un receptor de mensajes:

```
<service android:name=".SegundoPlano" />

    <receiver android:name=".Lectura" />
```

Nuestra actividad Principal tiene una barra de acción especial (Split action bar), tendremos que declarar la actividad y poner las condiciones necesarias para que se muestre correctamente:

```
android:uiOptions="splitActionBarWhenNarrow"
        android:name="pfc.consumoelectrico.app.MainActivity"
        android:theme="@style/MyCustomTheme"
        android:label="@string/app_name" />
    <meta-data android:name="android.support.UI_OPTIONS"
        android:value="splitActionBarWhenNarrow" />
```

Por último como cada actividad, deberemos declarar la que ejecuta el control de los ajustes:

```
<activity
        android:name=".UserSettingActivity"
        android:label="Ajustes"/>
```

ARCHIVOS GRADLE

¡Error! No se encuentra el origen de la referencia.

Para que el sistema funcione correctamente necesitaremos configurar el archivo *build.gradle* con la ruta donde guardamos los archivos "assets", es decir, los recursos externos empleados como la fuente LCD empleada

```
    sourceSets {
        main {
            assets.srcDirs =
['src/main/assets']
        }
    }
```

También deberemos asegurarnos que tenemos bien configurado el SDK a emplear para que no de error al compilar: compileSdkVersion 19; buildToolsVersion "19.1.0" y la versión de gradle empleada:

```
task wrapper(type: Wrapper) {
    gradleVersion = '1.12'
```

Normalmente se aconseja no tocar estos ficheros, pero en Android Studio al ser una versión preliminar no me ha actualizado las configuraciones correctamente en el caso de la versión SDK y no contemplaba el uso de assets externos en el directorio usado (donde se utiliza normalmente en Eclipse)

APP.IML

Para terminar de configurar los assets debemos poner el path en el archive app.iml dentro de

"<facet type="android" name="Android"> <configuration>" :
<option name="ASSETS_FOLDER_RELATIVE_PATH" value="/src/main/assets" />

Clase SplashScreenActivity

Esta clase la utilizo para generar una pantalla de presentación mientras cargan los datos de la pantalla principal

```
setContentView(R.layout.splash_screen);

    TimerTask task = new TimerTask() {
      @Override
      public void run() {
        // intent necesario para llamar a la siguiente aplicación a ejecutarse
        Intent mainIntent = new Intent().setClass(SplashScreenActivity.this,
MainActivity.class);
        startActivity(mainIntent);
        // destruímos la actividad, no se puede volver
        finish();
      }
    };
    // Simula un proceso que tarda al arrancar la aplicación
    Timer timer = new Timer();
    timer.schedule(task, SPLASH_SCREEN_DELAY);
```

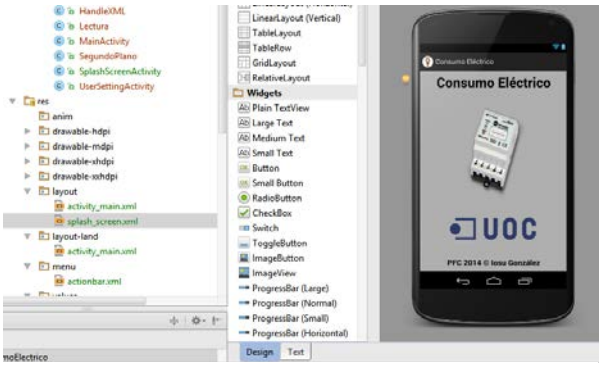


Figura 27 - Layout SplashScreen

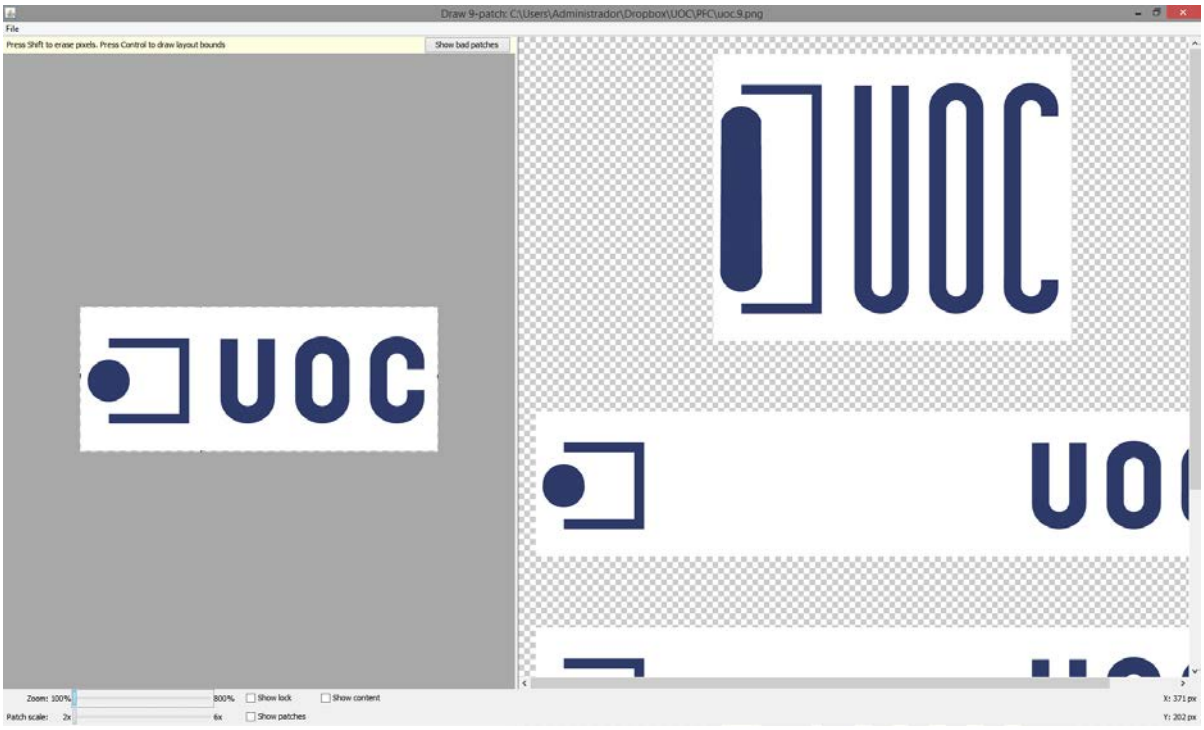
Necesitaremos un Layout específico (Figura 27 - Layout SplashScreen) que es el que cargará la actividad, el cual estará almacenado con todos los archivos de configuración de pantalla en la carpeta layout:

DRAW9PATCH

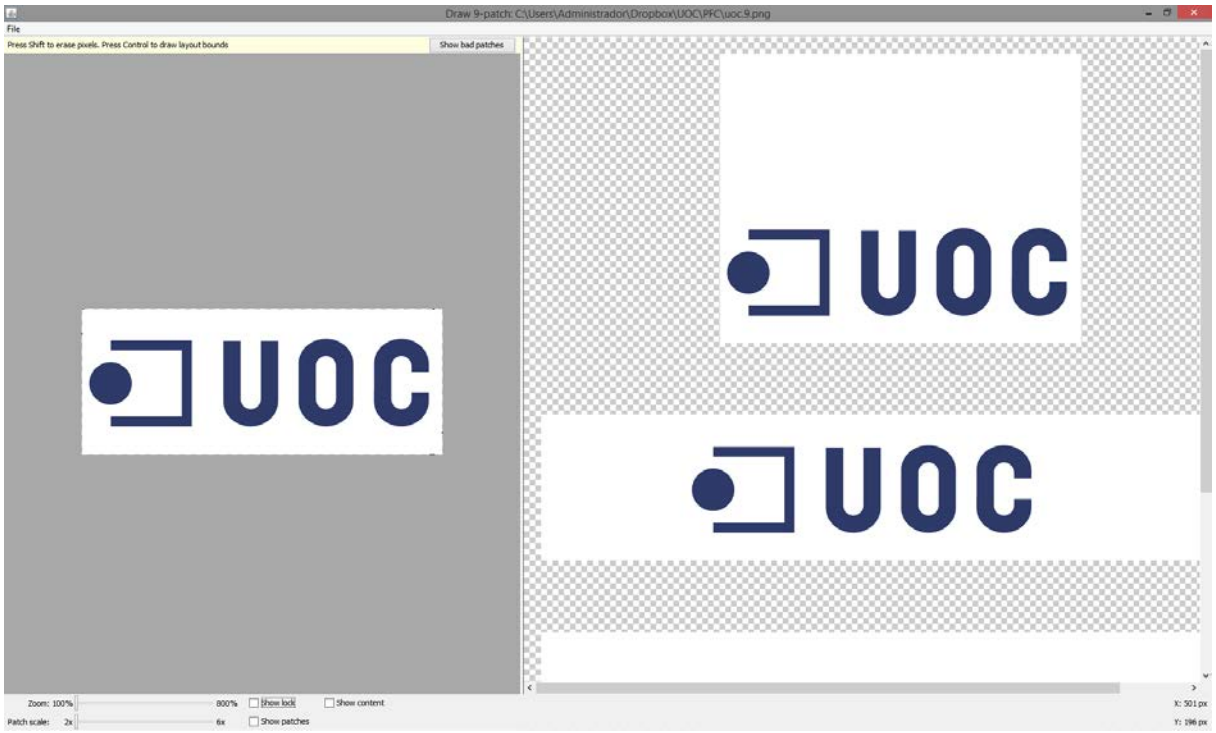
Dentro del SDK disponemos de una herramienta muy útil para configurar nuestros archivos de imagen png (portable network grafics10) de forma que no se deformen al cambiar de resolución. Esta utilidad se llama “Draw9Patch” y consiste en un editor de imágenes en el cual debemos de seleccionar qué partes son deformables para que a la hora de mostrarla en la pantalla quede correctamente.

El mismo editor nos da una vista previa de cómo va a quedar si se deforma y podemos grabar el archivo con la extensión 9.png

[10] http://es.wikipedia.org/wiki/Portable_Network_Graphics



Para poder generar un archivo que se expanda correctamente deberemos marcar las áreas “expandibles”, por ejemplo si fuera un botón con los laterales redondeados podríamos marcar los bordes para que solo el cuerpo central se expandiera. En nuestro caso por ejemplo podríamos marcar las áreas en blanco para que sean las únicas que se deformen:



Clase MainActivity

LAYOUTS

Esta es la clase principal, la que ejecuta la mayor parte del proceso. El funcionamiento es el siguiente: la activity muestra la pantalla principal cargando el archivo Layout (Figura 28 - Layout) (o layout-land si el terminal está apaisado) y crea el menu (inflate) de ajustes (UserSettingActivity). También muestra la barra de accesos directos (botones de la parte inferior), la cual tiene su propio archivo xml de configuración (Actionbar.xml)

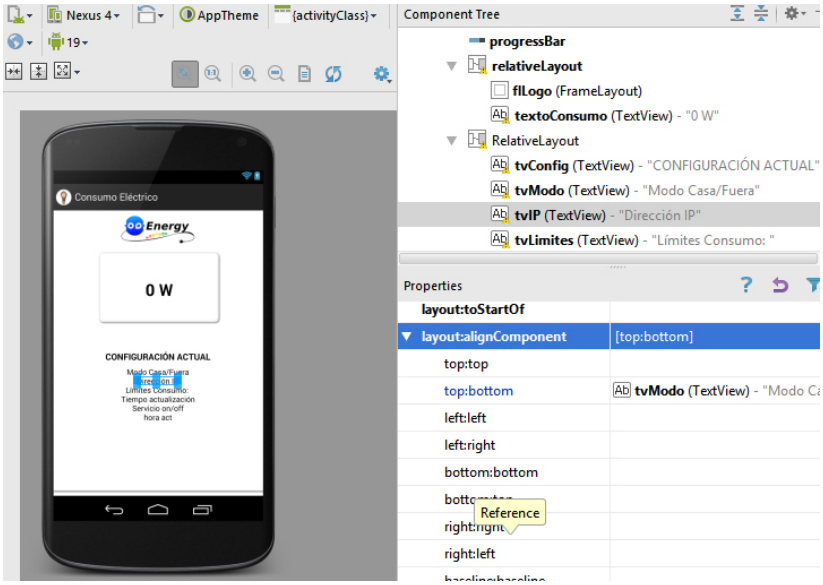


Figura 28 - Layout

Destacar que los elementos se han dispuesto mediante Layouts Relativos, de forma que la disposición de los elementos depende de la posición entre ellos y no de una absoluta que podría no quedar bien si empleamos una pantalla diferente a la del nexus 4, la que utilizamos en Android Studio. Por ejemplo, El Textview que muestra la dirección IP estará siempre debajo del tipo de modo.

En el caso del archivo empleado en Layout-Land, que recordamos es el directorio donde se guardan los ficheros de configuración con la disposición apaisada, hemos cambiado los elementos de sitio como se puede ver en la (Figura 29 - Layout-land).

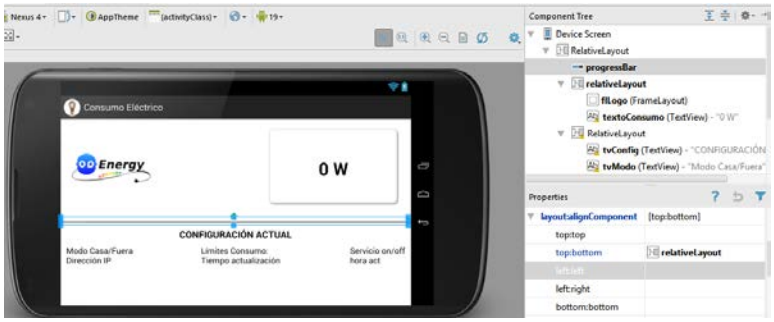


Figura 29 - Layout-land

En la clase principal cada vez que debamos referirnos a un elemento del layout deberemos indicarles de cual se trata, puesto que las variables de la clase principal pueden ser diferentes, aunque en caso de que se llamaran igual tampoco sería capaz de relacionarlas. Así por ejemplo para escribir en el campo de texto tenemos un textview en la aplicación que debemos relacionarlo con el campo en el layout:

```
tvLectura = (TextView) findViewById(R.id.textoConsumo);

tvTime = (TextView) findViewById(R.id.tvTime);
```

MÉTODOS

Esta clase dispone de varios métodos que sobreescribimos con respecto a los heredados de la clase activity, por ejemplo el que indica lo que se tiene que hacer al crearse la actividad que es dibujar la vista conforme está diseñado en el layout que vimos anteriormente (activity.main):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
}
```

También tendremos un método necesario para responder al pulsar sobre los botones de la action bar, por ejemplo al pulsar el botón de ajustes se ejecutará la actividad UserSettingActivity mediante un intent:

```
@Override //menu action bar
   public boolean onOptionsItemSelected(MenuItem item) {
      SharedPreferences.Editor editor = sharedPrefs.edit();
      switch (item.getItemId()) {
         case R.id.ajustes:
            Intent i = new Intent(this, UserSettingActivity.class);
            startActivityForResult(i, RESULT_SETTINGS);
            break;
         case R.id.actualizar:
            actualizar();
            break;
etc……
```

Otro método implementado es onActivityResult, el cual se ejecuta siempre que se vuelve de otra actividad, retornando a la vista en la que estábamos antes de entrar en los ajustes, como hemos comentado en el ciclo de actividades, los datos se quedan guardados al pausar la actividad principal.

Las lecturas en segundo plano es el método que reacciona ante una llamada desde otra clase, el método receptor, de tipo BroadCastReceiver el cual se ejecuta al recibir una acción definida en el android manifest, y al recibir la acción LEER se ejecurá el método Receptor():

```
IntentFilter filter = new IntentFilter();
    filter.addAction(Lectura.LEER);
    Receptor receptor = new Receptor();
    registerReceiver(receptor, filter);
```

Tenemos más métodos que empleamos para diversas acciones, como actualizar los datos de ajustes en pantalla (showUserSettings()) o procesarLectura, que analiza el dato recibido de W para ver si hay que generar un aviso. También he implementado un método iniciar el servicio (segundoplano) cuando se activa la alarma, la cual realizará lecturas según el tiempo definido en ajustes, etc..

ACTIONBAR

En la pantalla principal dispondremos de una barra de acciones con botones (Figura 30), dependiendo de si se muestra la pantalla en vertical o apaisado la apariencia será distinta. Si la pantalla está en vertical veremos la barra abajo, si está en apaisado, sobre la misma barra de acción normal

Figura 30 - Barra de acciones

Las acciones que realizaremos por orden de izquierda a derecha serán estas:

- Actualizar manualmente
- Activar el modo casa: Cambia la dirección IP y el periodo de actualización
- Activar la alarma: gestiona las actualizaciones del valor de forma automática
- Modo Vigilancia: incrementa el ratio de refresco de la lectura
- Ajustes de la aplicación

ProcesarLectura()

Este método lo que realiza es una comparativa entre los W medidos y los límites establecidos en los ajustes, de forma que si la lectura está por encima del primer límite generará un aviso en la barra de notificaciones, encenderá el led de avisos, sonará el sonido predeterminado de mensaje del terminal y veremos el rótulo ("Límite de consumo", "Superado consumo de XXX W").

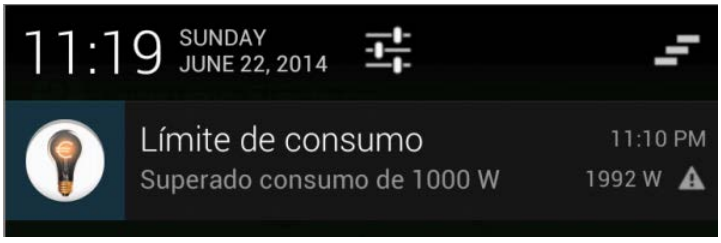


Figura 31 - detalle notificación

El aviso contendrá un ticker con detalle de la notificación y la lectura leída

Si se supera el límite máximo recibiremos el aviso ("Consumo máximo alcanzado", "Superado límite de consumo"), y el móvil activará la vibración

Figura 32 - Aviso de notificación

Pulsando sobre la notificación volveremos a nuestra aplicación principal mediante el intent main

Clase ApplicationContextProvider

Esta clase la utilizo para poder acceder al contexto de la aplicación desde cualquier clase o actividad. Para poder utilizar sus métodos y variables hay que es especificarlo en el Android Manifest. Es un "truco" para poder tener acceso a variables que no se pueden instanciar de la actividad principal

```
public class ApplicationContextProvider extends
Application {

    /**
     * Keeps a reference of the application context
     */
    private static Context sContext;


    @Override
```

SEGUNDO PLANO

Esta clase se iniciará cada vez que encendamos la alarma o se ponga el modo vigilancia, el cual la activa automáticamente, y su función principal es gestionar una alarma de sistema que ejecutará un intent que recibirá la clase Lectura cuya función es leer los datos mediante una clase auxiliar que sirve para procesar archivos XML.

LECTURA

Esta clase es de tipo BroadCastReceiver y se activará al finalizar la alarma iniciada por la clase de servicio SegundoPlano. Una vez recibida la alarma, esta leerá del fichero XML los datos por mediación de la clase auxiliar HandleXML

HandleXML

La clase HandleXML tiene como finalidad conectarse a la url destino, recuperar el archivo XML, leerlo y "parsearlo", es decir, recorrer los nodos que componen el archivo y recuperar los datos asociados, en este caso el valor de L1, el consumo en W.

OTROS ARCHIVOS

arrays.xml : almaceno los datos de configuración de los parámetros de la aplicación

```
<string-array name="syncFrecCasa">
    <item name="5">Cada 5 minutos</item>
    <item name="15">Cada 15 minutos</item>
    <item name="30">Cada Media Hora</item>
    <item name="60">Cada Hora</item>
  </string-array>
  <string-array name="syncCasaValores">
    <item name="5">5</item>
    <item name="15">15</item>
    <item name="30">30</item>
    <item name="60">60</item>
  </string-array>
```

styles.xml : almaceno los datos de configuración del tema empleado en la main activity

```
<resources>
  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar"/>
  <style name="MyCustomTheme" >
    <item name="android:backgroundSplit">@color/Black</item>
  </style>
  <style name="MyActionBar" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="android:backgroundSplit">@color/Black</item>
    <item name="android:textColor">@color/Black</item>
    <item name="android:background">@color/Black</item>
  </style>

</resources>
```

Colors.xml: para acceder fácilmente a los colores sin conocer los códigos rgb (@color/White)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="White">#FFFFFF</color>
  <color name="Ivory">#FFFFF0</color>
  <color name="LightYellow">#FFFFE0</color>
  <color name="Yellow">#FFFF00</color>
```

strings.xml: para acceder fácilmente nombres comunes sin tener que nombrarlos cada vez

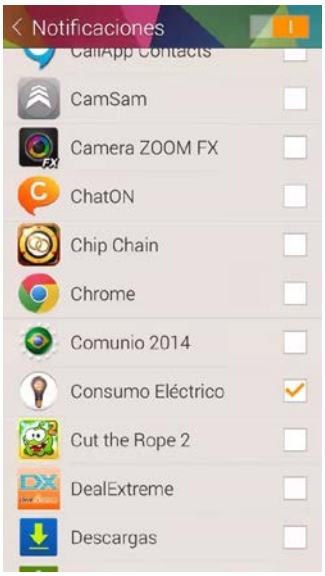
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Consumo Eléctrico</string>
  <string name="menu_settings">Ajustes</string>
 </resources>
```

USOS ADICIONALES

Este sistema puede aprovecharse para ver las lecturas por ejemplo en un smartwatch como el Samsung Galaxy Gear 2, para lo cual necesitaríamos configurar mediante la plataforma Tizen un widget que recibiera los datos desde la actividad principal mediante el uso de broadcast.

Pero existe una alternativa mucho más sencilla, ya que el programa Gear Manager que enlaza nuestro dispositivo móvil con el "wearable" Gear 2 es capaz de recoger las notificaciones del sistema y notificarlas sobre el reloj (Figura 34), para lo cual deberemos configurar el gestor de notificaciones (Figura 33).

Figura 34 - Notificaciones en Gear2



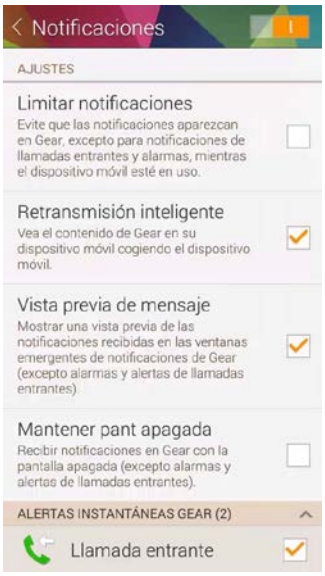


Figura 33 - Configuración Notificaciones Gear2

Otra utilidad encontrada es instalar un programa tipo Floatifications[11] que recoge las notificaciones y las muestra en la pantalla principal o en la de bloqueo, así que sin acceder a la aplicación podemos ver los avisos que nuestra app manda a la barra de notificaciones.

[11] https://play.google.com/store/apps/details?id=robj.floating.notifications&hl=es

BIBLIOGRAFÍA

LIBROS Y TUTORIALES

El gran libro de Android – Jesús Tomás Gironés

El gran libro de Android avanzado – Varios autores

Curso Programación Android – Salvador Oliver – Curso online

Android Development – Vogella.com

ALGUNOS RECURSOS ONLINE CONSULTADOS

Algunas de estas páginas web me han resultado muy útiles para resolver pequeños problemas y dudas durante el desarrollo de la aplicación

http://www.powenko.com/en/?p=2550

https://github.com/thecodepath/android_guides/wiki/Extended-ActionBar-Guide

http://www.silverbaytech.com/2013/05/27/themes-for-the-android-actionbar-actionbaritems/

http://stackoverflow.com/questions/5502427/resume-application-and-stack-from-notification/5502950#5502950

http://elbauldelprogramador.com/programacion-android-interfaz-grafica_11/

http://www.aprendeandroid.com/l7/notificacion.htm

http://amatellanes.wordpress.com/2013/08/27/android-crear-un-splash-screen-en-android/

http://www.dafont.com/es/search.php?fpp=50&af=on&q=lcd (Fuente LCD empleada)

http://timnew.github.io/blog/2013/12/13/asset-path-in-android-stuidio/

WEBS DE REFERENCIA

https://developer.android.com/develop/index.html

http://www.opendomo.es/products/ODEnergyMEW

SOFTWARE ESPECÍFICO EMPLEADO

Android Studio

Hypercam (grabar pantalla PC)

Samsung Sydesync (captura pantalla terminales Samsung con Android)

ANEXO 1

VERSIONES DE ANDROID[12]

Platform Version	API Level	VERSION_CODE	Notes
Android 4.4	19	KITKAT	Platform Highlights
Android 4.3	18	JELLY_BEAN_MR2	Platform Highlights
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	Platform Highlights
Android 4.1, 4.1.1	16	JELLY_BEAN	Platform Highlights
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1	Platform Highlights
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH	
Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	Platform Highlights
Android 3.0.x	11	HONEYCOMB	Platform Highlights
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1	Platform Highlights
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD	
Android 2.2.x	8	FROYO	Platform Highlights
Android 2.1.x	7	ECLAIR_MR1	Platform Highlights
Android 2.0.1	6	ECLAIR_0_1	
Android 2.0	5	ECLAIR	
Android 1.6	4	DONUT	Platform Highlights
Android 1.5	3	CUPCAKE	Platform Highlights
Android 1.1	2	BASE_1_1	
Android 1.0	1	BASE	

[12] Fuente: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html

ANEXO 2 – CÓDIGO FUENTE

ANDROID MANIFEST

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="pfc.consumoelectrico.app" >
   <uses-permission
      android:name="android.permission.INTERNET"/>
   <uses-sdk
      android:minSdkVersion="14"
      android:targetSdkVersion="16" />
   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      android:name=".ApplicationContextProvider"
      android:label="Contexto">
      <activity
         android:name="pfc.consumoelectrico.app.SplashScreenActivity"
         android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
         <intent-filter>
         <action android:name="android.intent.action.SEND" />
         <category android:name="android.intent.category.DEFAULT" />
         <data android:mimeType="text/plain" />
         </intent-filter>
      </activity>
      <activity
         android:uiOptions="splitActionBarWhenNarrow"
         android:name="pfc.consumoelectrico.app.MainActivity"
         android:theme="@style/MyCustomTheme"
         android:label="@string/app_name" />
      <meta-data android:name="android.support.UI_OPTIONS"
         android:value="splitActionBarWhenNarrow" />
      <activity
         android:name=".UserSettingActivity"
         android:label="Ajustes"/>
      <service android:name=".SegundoPlano" />
      <receiver android:name=".Lectura" />
   </application>
</manifest>
```

BUILD.GRADLE

```
apply plugin: 'android'

android {
   compileSdkVersion 19
   buildToolsVersion "19.1.0"

   defaultConfig {
      minSdkVersion 10
      targetSdkVersion 19
      versionCode 1
      versionName "1.0"
   }
   buildTypes {
      release {
         runProguard false
         proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.txt'
      }
   }
   sourceSets {
      main {
         assets.srcDirs = ['src/main/assets']
      }
   }

   dependencies {
      compile 'com.android.support:appcompat-v7:19.1.0'
      compile 'com.android.support:gridlayout-v7:19.1.0'
      compile 'com.android.support:support-v4:19.1.0'
      compile fileTree(dir: 'libs', include: ['*.jar'])
   }
   task wrapper(type: Wrapper) {
      gradleVersion = '1.12'
   }
}
```

Actionbar.xml

Archivo de configuración de la barra de acción de la Main Activity:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <!-- barra de botones de acciones -->

    <!-- actualización manual-->
    <item android:id="@+id/actualizar"
        android:icon="@android:drawable/stat_notify_sync_noanim"
        android:title="Actualizar"
        android:showAsAction="ifRoom" />

    <!-- modo casa o remoto-->
    <item android:id="@+id/modocasa"
        android:icon="@android:drawable/ic_menu_myplaces"
        android:title="Modo Casa"
        android:showAsAction="ifRoom" />

    <!-- servicio automático de actualización-->
    <item android:id="@+id/servicio"
        android:icon="@android:drawable/ic_lock_idle_alarm"
        android:title="Auto"
        android:showAsAction="ifRoom" />

    <!-- horapunta - aumento refresco automático - vigilar!!-->
    <item android:id="@+id/horapunta"
        android:icon="@android:drawable/ic_menu_view"
        android:title="Vigilar"
        android:showAsAction="ifRoom" />

    <item
        android:id="@+id/ajustes"
        android:orderInCategory="100"
        android:showAsAction="ifRoom"
        android:title="@string/menu_settings"
        android:icon="@android:drawable/ic_menu_preferences"/>

</menu>
```

SEGUNDO PLANO

```
package pfc.consumoelectrico.app;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.IBinder;
import android.preference.PreferenceManager;
import android.util.Log;
import android.widget.Toast;
import java.util.Calendar;

public class SegundoPlano extends Service {

   public static final String NOTIFICATION = "pfc.consumoelectrico.app";
   public static final String FIN = "Fin Servicio";
   //public Context contexto = ApplicationContextProvider.getContext();
   public SharedPreferences sharedPrefs;
   public boolean modoCasa=false;
   public int tiempoAct=5;
   private static PendingIntent pendingIntent; //lo usamos para la alarma no static!

   @Override
   public void onCreate() {
      super.onCreate();
      Log.d("Service", "onCreate");
      sharedPrefs = PreferenceManager.getDefaultSharedPreferences(this);
   }

   @Override
   public int onStartCommand(Intent intent, int flags, int idArranque) {
      super.onStartCommand(intent, flags,idArranque);
      Log.d("servicio","servicio arrancado");

      ActivarAlarma();
      /*Intent i = new Intent();
      i.setAction(NOTIFICATION);
      i.putExtra("Lectura","123");
      this.sendBroadcast(i);
      //this.stopSelf();*/
      return START_STICKY;
   }

   @Override
   public void onDestroy() {
     /*Intent bcIntent = new Intent();
      bcIntent.setAction(FIN);
      sendBroadcast(bcIntent);*/
      DesactivarAlarma();
      this.stopSelf();
      super.onDestroy();
      Log.d("servicio","servicio destruido");
   }

   @Override
   public IBinder onBind(Intent arg0) {
      return null;
```

```
    }

    /**
     * Desactiva o activa la alarma, estableciéndola al estado contrario al actual
     */
    /*private void CambiarEstadoAlarma()
    {
        if (pendingIntent == null)
        {
            //La alarma está desactivada, la activamos
            ActivarAlarma();
        }else
        {
            //La alarma está activada, la desactivamos
            DesactivarAlarma();
        }
    }

    /**
     * Desactivar la alarma
     */
    private void DesactivarAlarma()
    {
        AlarmManager alarmManager = (AlarmManager)getSystemService(ALARM_SERVICE);
        alarmManager.cancel(pendingIntent);
        pendingIntent = null;
        Toast.makeText(this, "Alarma detenida", Toast.LENGTH_LONG).show();
    }

    /**
     * Activa la alarma
     */
    private void ActivarAlarma()
    {
       /* boolean modoCasa=sharedPrefs.getBoolean("modoCasa", true);
        if (modoCasa) {
            tiempoAct=Integer.parseInt(sharedPrefs.getString("prefActCasa","0"))*60;//tiempo en prefs en
minutos
            }
        else{
            tiempoAct = Integer.parseInt(sharedPrefs.getString("prefActFuera", "0"))*60;
        }*/
        tiempoAct=sharedPrefs.getInt("tiempoAct",0);

        Intent myIntent = new Intent(getApplicationContext(), Lectura.class);
        //pendingIntent = PendingIntent.getService(getApplicationContext(), 0, myIntent, 0);
        pendingIntent = PendingIntent.getBroadcast(getApplicationContext(), 0, myIntent, 0);

        AlarmManager alarmManager = (AlarmManager)getSystemService(ALARM_SERVICE);
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(System.currentTimeMillis());
        calendar.add(Calendar.SECOND, 10);
        alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), tiempoAct *
1000, pendingIntent);

        Toast.makeText(this, "Alarma en "+tiempoAct+" segundos", Toast.LENGTH_LONG).show();
    }

}
```

LECTURA

```
public class Lectura extends BroadcastReceiver {
   public static final String LEER = "lectura";
   public int lectura=111;
   public SharedPreferences sharedPrefs;
   public boolean modoCasa=false;
   private HandleXML obj;
   private String url;
//    public Context contexto = ApplicationContextProvider.getContext();

   public int Lectura(){
      int lectura=0;
      return lectura;
   }

   @Override
   public void onReceive(Context context, Intent intent) {
      //recibimos aviso de alarma y leemos
      Log.d("Lectura", "Alarma recibida"); //mensaje para el log y ver la recepcion de alarma
      lectura=leer(context);
      Intent i = new Intent(LEER).putExtra(LEER,lectura);//Mandamos a la aplicación principal la lectura
      context.sendBroadcast(i);
   }

   public int leer(Context contexto){
      sharedPrefs = PreferenceManager.getDefaultSharedPreferences(contexto);
      modoCasa=sharedPrefs.getBoolean("modoCasa", true);
      if (modoCasa) { //Recupera de las preferencias la IP
         url="http://"+sharedPrefs.getString("ipCasa","192.168.1.4")+"/data.xml";
         }
      else{
         url="http://"+sharedPrefs.getString("ipRemoto","iosu.noip.me")+"/data.xml"; //IP
ALTERNATIVA EN SERVIDOR LOCAL}
      }
      obj = new HandleXML(url);
      obj.fetchXML();
      while (obj.parsingComplete) ;
      return Integer.parseInt(obj.getActive_L1());
   }
}
```

HandleXML

```
public class HandleXML {

   private String active_L1 = "0";
   private String urlString = null;
   private XmlPullParserFactory xmlFactoryObject;
   public volatile boolean parsingComplete = true;
   public HandleXML(String url){
      this.urlString = url;    }
   public String getActive_L1(){
      return active_L1;    }

   public void parseXMLAndStoreIt(XmlPullParser myParser) {
      int event;
      String text=null;
      try {
         event = myParser.getEventType();
         while (event != XmlPullParser.END_DOCUMENT) {
            String name=myParser.getName();
            switch (event){
               case XmlPullParser.START_TAG:
                  break;
               case XmlPullParser.TEXT:
                  text = myParser.getText();
                  break;
               case XmlPullParser.END_TAG:
                  if(name.equals("active_L1")){
                     active_L1 = text;
                  }
                  break;
            }
            event = myParser.next();
         }
         parsingComplete = false;
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
   public void fetchXML() {
      Thread thread = new Thread(new Runnable() {
         @Override
         public void run() {
            try {
               URL url = new URL(urlString);
               HttpURLConnection conn = (HttpURLConnection)url.openConnection();
               try {
                  conn.setReadTimeout(10000 /* milliseconds */);
                  conn.setConnectTimeout(15000 /* milliseconds */);
                  conn.setRequestMethod("GET");
                  conn.setDoInput(true);
                  conn.connect();
                  InputStream stream = conn.getInputStream();
                  xmlFactoryObject = XmlPullParserFactory.newInstance();
                  XmlPullParser myparser = xmlFactoryObject.newPullParser();
                  myparser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES , false);
                  myparser.setInput(stream, null);
                  parseXMLAndStoreIt(myparser);
                  stream.close();
                  conn.disconnect();
               } catch (Exception e) {
                 e.printStackTrace(); //muestra excepción si falla la lectura
               }
            } catch (Exception e) {
               e.printStackTrace(); //muestra excepción si falla la conexión
            }
         }
      });
      thread.start();
   }   }
```

MainActivity

```
public class MainActivity extends Activity {

   private TextView tvLectura, tvModo, tvIP, tvLimites, tvTime, tvAct, tvServicio;
   private boolean modoCasa, activo = true, horaPunta = false;
   private String url, active_L1 = "0";
   private static final int RESULT_SETTINGS = 1, CURRENT_ACTIVITY = 666;
   public SharedPreferences sharedPrefs;
   private int alerta1 = 0, lectura = 666, alerta2 = 0, tiempoAct = 15;
   private static final int NOTIF_ALERTA_ID = 1;//identificador alerta notificación alarmas
   private ProgressBar progressBar;
   int progressValue = 0;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
      // Defino una nueva fuente tipo LCD cargandola desde el fichero .ttf
      Typeface miPropiaTypeFace = Typeface.createFromAsset(getAssets(), "fonts/lcd.ttf");
      miPropiaTypeFace = Typeface.create(miPropiaTypeFace, Typeface.BOLD);
      tvLectura = (TextView) findViewById(R.id.textoConsumo);
      tvLectura.setTypeface(miPropiaTypeFace); //establecemos la fuente LCD para este textview

      sharedPrefs = PreferenceManager.getDefaultSharedPreferences(this);

      progressBar = (ProgressBar) findViewById(R.id.progressBar);
      showUserSettings();
      //actualizar(); //actualiza los valores al arrancar

      //necesario registrar los tipos de mensajes a recibir:
      IntentFilter filter = new IntentFilter();
      filter.addAction(Lectura.LEER);
      Receptor receptor = new Receptor();
      registerReceiver(receptor, filter);
   }

   @Override //"inflado" menu settings
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.actionbar, menu);
      return true;
   }

   @Override //menu action bar
   public boolean onOptionsItemSelected(MenuItem item) {
      SharedPreferences.Editor editor = sharedPrefs.edit();
      switch (item.getItemId()) {
         case R.id.ajustes:
            Intent i = new Intent(this, UserSettingActivity.class);
            startActivityForResult(i, RESULT_SETTINGS);
            break;
         case R.id.actualizar:
            actualizar();
            break;
         case R.id.servicio:
            activo = sharedPrefs.getBoolean("servicio", false);
            if (!activo) { //si no está activo el servicio lo activamos
               activo = true;
               editor.putBoolean("servicio", true);
               editor.commit();
            } else {
               activo = false;
               editor.putBoolean("servicio", false);
               editor.commit();
```

```
            }
            ActivarServicio();
            break;
        case R.id.horapunta:
            //chequeamos si se ha activado la hora punta o desactivado

            horaPunta = sharedPrefs.getBoolean("horaPunta", true);
            if (!horaPunta) { //si estaba desactivado lo activamos
                editor.putBoolean("horaPunta", true);
                editor.commit();
                if (!activo) {
                    activo = true;
                    editor.putBoolean("servicio", true);
                    editor.commit();//sino está activo lo activamos
                    tiempoAct = Integer.parseInt(sharedPrefs.getString("prefActPunta", "0"));

                    editor.putInt("tiempoAct", tiempoAct);
                    editor.commit();
                    switch (tiempoAct) {
                        case 60:
                            tvAct.setText("T. de Actualización: 1 minuto");
                            break;
                        default:
                            tvAct.setText("T. de Actualización: " + tiempoAct + " segundos");
                    }
                    ActivarServicio();
                }
                ;
            } else {
                editor.putBoolean("horaPunta", false);
                editor.commit();
                activo = false;
                editor.putBoolean("servicio", false);
                editor.commit();
            }

            break;
        case R.id.modocasa:
            modoCasa = sharedPrefs.getBoolean("modoCasa", true);
            if (modoCasa) {
                editor.putBoolean("modoCasa", false);
                editor.commit();
            } else {
                editor.putBoolean("modoCasa", true);
                editor.commit();
            }
            showUserSettings();
            actualizar();
            break;
    }
    return true;
}

public void actualizar() {
    Intent intent = new Intent(this, Lectura.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(this.getApplicationContext(), 234324243, intent, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
    alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis(), pendingIntent);
}

//ACTUALIZACION DATOS AL VOLVER DE CONFIGURACION:
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode) {
        case RESULT_SETTINGS:
            showUserSettings();
```

```
            actualizar(); //esto no es necesario si hay segundo plano  actualizar(this.getCurrentFocus());
            break;
        }
    }

    private void showUserSettings() { //muestra configuración en pantalla
        modoCasa = sharedPrefs.getBoolean("modoCasa", true);
        tvModo = (TextView) findViewById(R.id.tvModo);
        tvIP = (TextView) findViewById(R.id.tvIP);
        tvAct = (TextView) findViewById(R.id.tvAct);
        tvServicio = (TextView) findViewById(R.id.tvServicio);

        if (modoCasa) {
            url = "http://" + sharedPrefs.getString("ipCasa", "192.168.1.4") + "/data.xml";
            tvModo.setText("Modo Casa");
            tvIP.setText("IP: " + sharedPrefs.getString("ipCasa", "0.0.0.0"));
            tiempoAct = Integer.parseInt(sharedPrefs.getString("prefActCasa", "0"));
            switch (tiempoAct) {
                case 60:
                    tvAct.setText("T. de Actualización: 1 Hora");
                    break;
                default:
                    tvAct.setText("T. de Actualización: " + tiempoAct + " minutos");
            }
        } else {
            url = "http://" + sharedPrefs.getString("ipRemoto", "iosu.noip.me") + "/data.xml"; //IP ALTERNATIVA EN
SERVIDOR LOCAL
            tvModo.setText("Modo Remoto");
            tvIP.setText("IP: " + sharedPrefs.getString("ipRemoto", "IP REMOTA"));
            tiempoAct = Integer.parseInt(sharedPrefs.getString("prefActFuera", "0"));
            switch (tiempoAct) {
                case 30:
                    tvAct.setText("T. de Actualización: 30 minutos");
                    break;
                case 60:
                    tvAct.setText("T. de Actualización: 1 hora");
                    break;
                default:
                    tvAct.setText("T. de Actualización: " + tiempoAct + " horas");
            }
        }
        tiempoAct = tiempoAct * 60; //pasamos a segundos para tiempo actualización
        SharedPreferences.Editor editor = sharedPrefs.edit();
        if (horaPunta) {
            tiempoAct = Integer.parseInt(sharedPrefs.getString("prefActPunta", "0"));

            switch (tiempoAct) {
                case 60:
                    tvAct.setText("T. de Actualización: 1 minuto");
                    break;
                default:
                    tvAct.setText("T. de Actualización: " + tiempoAct + " segundos");
            }
        }
        alerta1 = Integer.parseInt(sharedPrefs.getString("alerta1", "1111"));
        alerta2 = Integer.parseInt(sharedPrefs.getString("alerta2", "2222"));
        tvLimites = (TextView) findViewById(R.id.tvLimites);
        tvLimites.setText("Límites Alertas: " + alerta1 + " W - " + alerta2 + " W");

        editor.putInt("tiempoAct", tiempoAct);
        editor.commit();
        ActivarServicio();
    }

    //cuadro de diálogo si queremos salir de la aplicación
    @Override
```

```
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            new AlertDialog.Builder(this)
                    .setIcon(android.R.drawable.ic_dialog_alert)
                    .setTitle("Salir")
                    .setMessage("¿Estás seguro?")
                    .setNegativeButton(android.R.string.cancel, null)//sin listener
                    .setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {//un listener que al pulsar,
cierre la aplicacion
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            stopService(new Intent(getBaseContext(), SegundoPlano.class));

                            MainActivity.this.finish();
                        }
                    })
                    .show();
            // Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer nada mas
            return true;
        }
        //para las demas cosas, se reenvia el evento al listener habitual
        return super.onKeyDown(keyCode, event);
    }

    public void ActivarServicio() {
        // start progress bar

        tvServicio = (TextView) findViewById(R.id.tvServicio);

        // servicio activo on/off
        SharedPreferences.Editor editor = sharedPrefs.edit();

        if (activo) {
            Intent msgIntent = new Intent(MainActivity.this, SegundoPlano.class);
            new Thread(myThread).start(); //arranca la barra
            startService(msgIntent);
            tvServicio.setText("Servicio Activado");
        } else {
            stopService(new Intent(getApplicationContext(), SegundoPlano.class));
            progressBar.setProgress(0);
            progressBar.setMax(0);
            tvServicio.setText("Servicio Desactivado");
        }
    }

    public class Receptor extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            //mensaje recibido mediante broadcast
            if (intent.getAction().equals(Lectura.LEER)) {
                lectura = intent.getIntExtra(Lectura.LEER, 123);
                ProcesarLectura(lectura);
            }
        }
    }

        private void ProcesarLectura(int lectura) {
            //esta clase procesa el dato de lectura recibido y lo muestra, gestionando también las alertas
            Date fecha = Calendar.getInstance().getTime();

            tvTime = (TextView) findViewById(R.id.tvTime);
            tvTime.setText(DateFormat.getTimeFormat(getApplicationContext()).format(fecha.getTime()));
//fecha.toLocaleString() df.format(fecha)
            MainActivity.this.tvLectura.setText(lectura + " W");
            if (lectura > alerta2) {
```

```
            // Realizo una notificacion por medio de un metodo hecho por mi
            notificacion(true, "Consumo máximo alcanzado", "Superado límite de consumo (" +
sharedPrefs.getString("alerta2", "0") + " W)", 2);
        } else if (lectura > alerta1) {
            notificacion(false, "Límite de consumo", "Superado consumo de " + sharedPrefs.getString("alerta1", "0") + "
W", 1);
        }
    }
    public void notificacion(boolean vibracion, String titulo, String texto, int led) {
        long[] pattern = {1000, 1000}; //patrón de vibrado
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        intent.setAction(Intent.ACTION_MAIN);
        intent.addCategory(Intent.CATEGORY_LAUNCHER);
        intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_SINGLE_TOP);
        PendingIntent pIntent = PendingIntent.getActivity(getApplicationContext(), NOTIF_ALERTA_ID, intent, 0);
        //construímos la notificación
        NotificationCompat.Builder mBuilder =
            new NotificationCompat.Builder(getApplicationContext())
                .setSmallIcon(android.R.drawable.stat_sys_warning)
                .setLargeIcon((((BitmapDrawable) getResources()
                    .getDrawable(R.drawable.ic_launcher)).getBitmap()))
                .setContentTitle(titulo)
                .setContentText(texto)
                .setContentInfo(lectura + " W")
                .setTicker("Consumo: " + lectura + " W")
                .setDefaults(Notification.DEFAULT_SOUND)
                .setOnlyAlertOnce(true)
                .setAutoCancel(true); //se autoelimina al seleccionar
        if (vibracion) {
            mBuilder.setVibrate(pattern); // desactivar en emulador - fuerza cierre aplicación!!
        }
        switch (led) {
            case 1:
                mBuilder.setLights(R.color.White, 1000, 5000); //notificacion en blanco, un segundo on, 5 off
                break;
            case 2:
                mBuilder.setLights(R.color.Red, 1000, 5000); //notificacion en rojo, un segundo on, 5 off
                break;
        }
        mBuilder.setContentIntent(pIntent);
        NotificationManager mNotificationManager =
            (NotificationManager) getApplicationContext().getSystemService(Context.NOTIFICATION_SERVICE);

        mNotificationManager.notify(NOTIF_ALERTA_ID, mBuilder.build());
    }
    private Runnable myThread = new Runnable() { //usado para barra de progreso

        @Override
        public void run() {
            progressBar.setActivated(false);
            progressBar.setMax(tiempoAct);
            while (progressValue < tiempoAct) {
                try {
                    myHandle.sendMessage(myHandle.obtainMessage());
                    Thread.sleep(1000);
                } catch (Throwable t) {
                }
            }
        }

        Handler myHandle = new Handler() {

            @Override
            public void handleMessage(Message msg) {
                progressValue++;
                progressBar.setProgress(progressValue);            }        };      };}
```