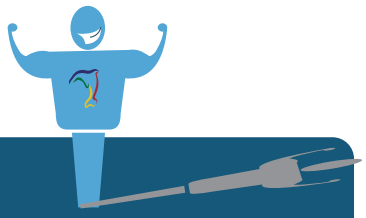
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

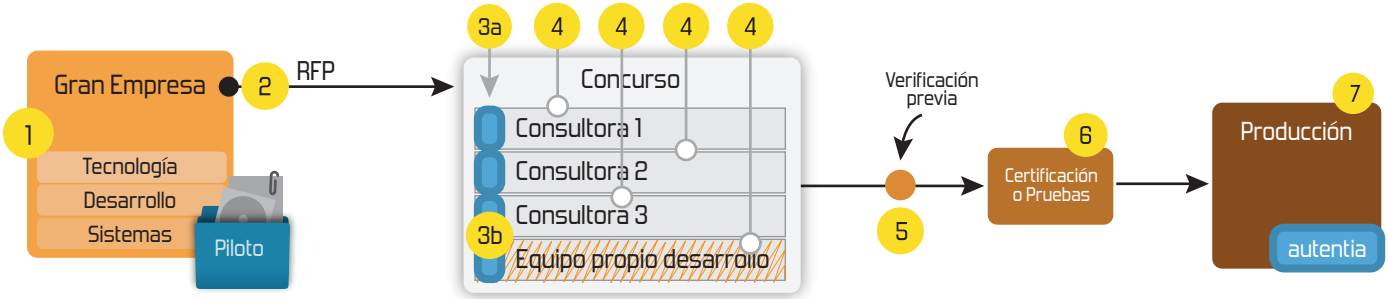
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.

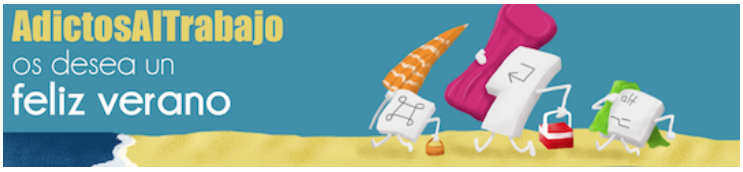


4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com



autentia
Soporte a desarrollo informático

Entra en Adictos a través de
E-mail
Contraseña
Entrar
Registrarme
Olvidé mi contraseña

Inicio Quiénes somos Formación Comparador de salarios Nuestros libros Más

» Estás en: Inicio Tutoriales Notificaciones push con Android, Google Cloud Message y JEE

Alberto Pla Martín

Desarrollador en JEE y Android

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2014-07-30 Tutorial visitado 137 veces Descargar en PDF

Notificaciones push con Android, Google Cloud Message y JEE

0. Índice de contenidos.

1. Entorno

Hardware: portátil Macbook Air (Intel Core I7, 8GB) y un Nexus 5 (este tutorial solo se ha probado con un dispositivo físico, no hay garantías que funcione con un dispositivo virtual)

Sistema operativo: OS X 10.9.3 y Android 4.4.4
JDK 1.7.0.51
Apache Tomcat 7.0.52
Eclipse Luna (lado del servidor)
Eclipse ADT Juno (lado del cliente Android)
Apache Maven 3.2.1

2. Introducción

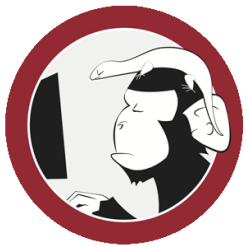
Una notificación push es un tipo de comunicación entre un dispositivo cliente y un servidor en el que es este último es el que inicia la petición, es decir, el servidor notifica al dispositivo cliente sobre algún evento sin que el usuario final tenga que realizar acción alguna. Esto se entiende mejor con el ejemplo de la aplicación de gmail en Android que nos alerta mediante una notificación visual y/o sonora de la llegada de nuevos emails incluso aunque la aplicación en si no la hayamos abierto o el terminal este en “stand by” y bloqueado.

Las notificaciones push tienen como ventaja frente a la técnica del polling (peticiones periódicas al servidor para averiguar si hay nuevos eventos pendientes de notificar) que consume menos recursos y las notificaciones llegan al instante y no al cabo de un periodo de tiempo.

En este tutorial veremos como podemos implementar una notificación push enviada desde una aplicación jee a una aplicación Android usando el servicio gratuito GCM (Google Cloud Messaging) que es proporcionado por Google. Para ilustrar todo el proceso se va hacer una pequeña aplicación jee a través de la cual se puede mandar una notificación a una determinada aplicación cliente de un terminal Android en concreto, con un mensaje introducido en un formulario de dicha aplicación.

En la imagen de abajo se ve un simple formulario web en el que se introduce el mensaje de la notificación y un botón para enviarlo

Catálogo de servicios Autentia



Síguenos a través de:

Últimas Noticias

» Comentando el libro Start-up Nation, La historia del milagro económico de Israel, de Dan Senor & Salu Singer

» Screencasts de programación narrados en Español

» Sorteo de entradas para APIdays Mediterranea

» Concurso del Día de la Madre:

» Aprende gratis ReactiveCocoa

Histórico de noticias

Últimos Tutoriales

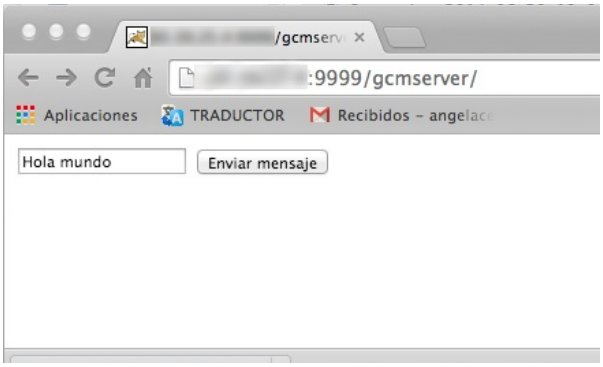
» Phonegap/Cordova y las Notificaciones Push

» Metodología ágiles. Catalizando el cambio en sector informático

» ¿Qué es Go?

» Grabación y edición multicámara en Final Cut Pro X

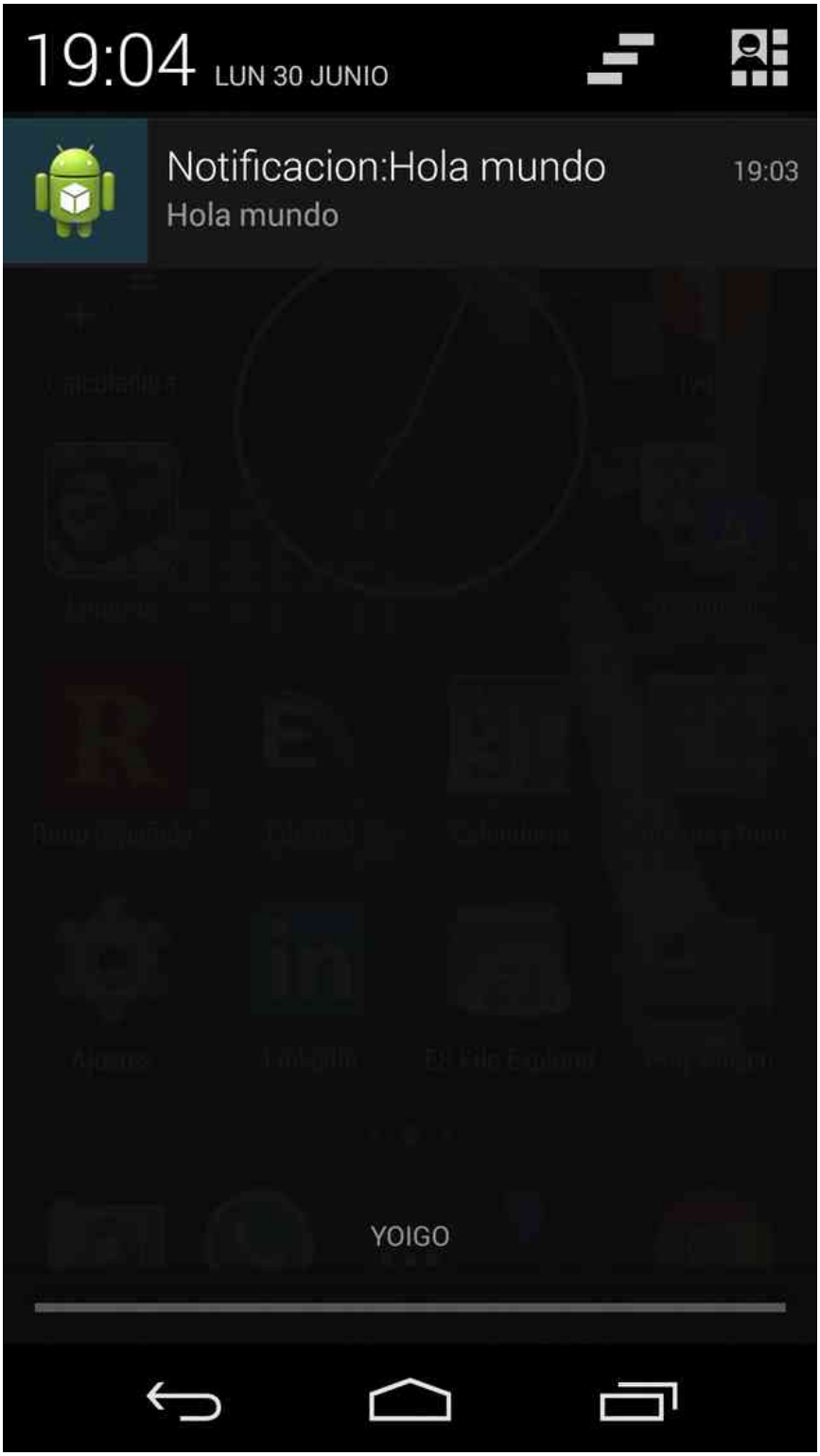
» GitLab: Crear y gestionar nuestro servidor propio de Git



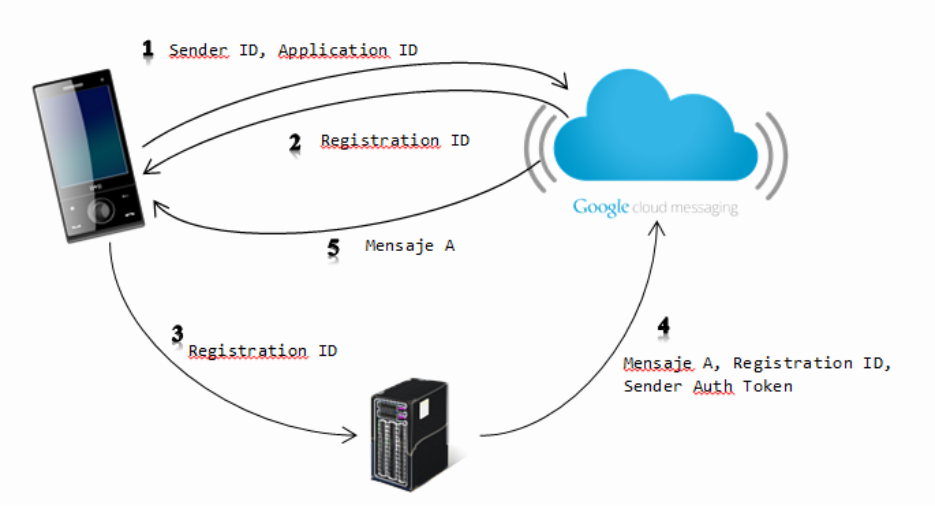
Al pulsar el botón del formulario se envía el mensaje a los servidores de GCM y se muestra un simple página de confirmación

El mensaje "Hola mundo" ha sido enviado

Una vez el mensaje ha llegado a los servidores GCM este lo reenvía al dispositivo Android de prueba que mostrará la notificación en la barra de notificaciones con el mensaje que hemos introducido en el formulario web, incluso si la aplicación cliente no esta abierta.



Para el uso de GCM intervienen tres actores, el dispositivo Android que recibirá la notificación, el servidor en el que se ejecuta el Tomcat, y los servidores de GCM proporcionados por Google. En el esquema que se presenta a continuación se puede ver que el proceso de notificación consta de 5 pasos enumerados por orden y que se describe con detalle.

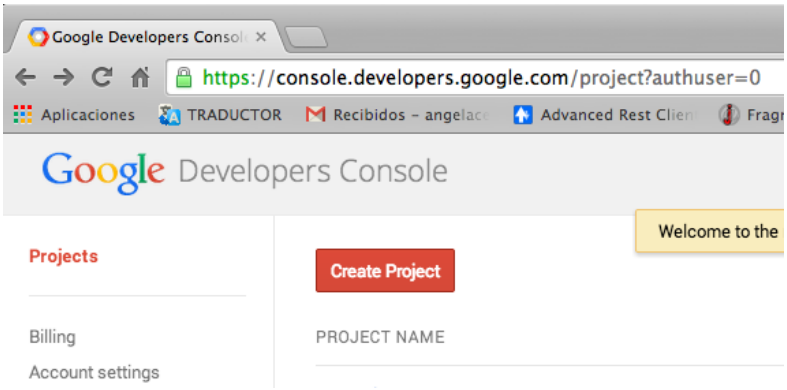


- Paso 1: La aplicación instalada en nuestro terminal Android se registra enviando a los servidores GCM el “Sender ID” y el “Application ID”. El “Sender ID” es el identificador de la instancia del paquete de servicios de “Google Play Services” entre los cuales se encuentra GCM y que obtendremos en la pagina de “Google Apis Console”, y finalmente la “Application ID” es el identificador de la aplicación formado a partir del nombre del paquete del mismo.
- Paso 2: Si el proceso de registro se ha realizado correctamente los servidores de GCM devolverán un “Registration ID” a la aplicación móvil. El Registration ID es un identificador que identifica una aplicación concreta en un dispositivo concreto.
- Paso 3: Reenviamos el “Registration ID” desde la aplicación Android hacia nuestro servidor en el cual se ejecuta el Tomcat y que guardará dicho identificador para el siguiente paso.
- Paso 4: Se manda el mensaje desde nuestro servidor a los servidores de GCM junto a el “Registration ID” que indica el destinatario del mensaje y que hemos guardado previamente en el paso 3, y finalmente el “Sender Autch Token” que permite autentificarnos contra una determinada instancia del paquete de servicios de “Google Play Services”.
- Paso 5: Los servidores de GCM manda el mensaje a los dispositivos destinatarios y se visualiza en la barra de notificaciones.

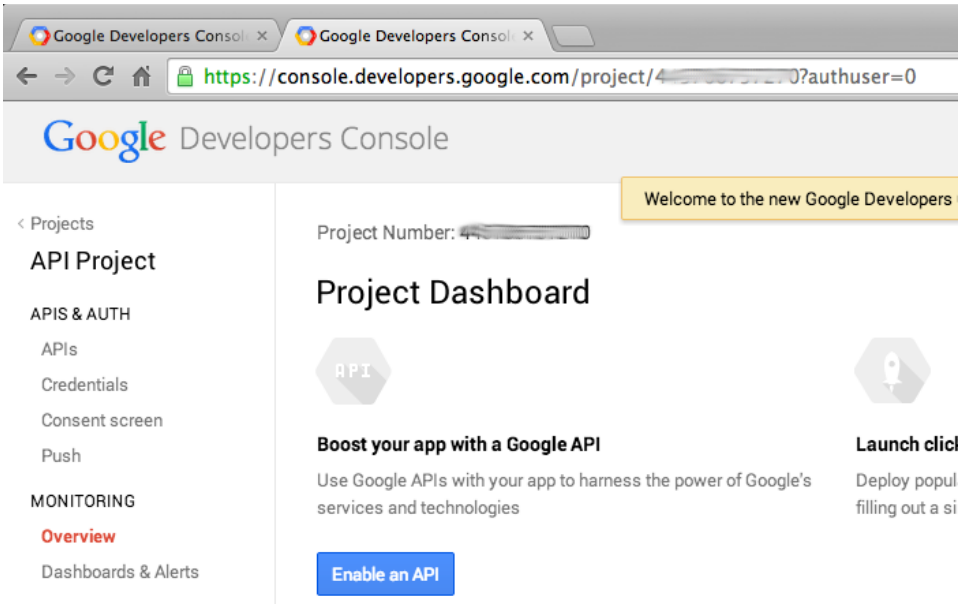
3.- Registro

La aplicación cliente del terminal Android se debe de registrar en el servicio de GCM, para ello se debe primero activar y configurar una instancia de dicho servicio desde la consola de Apis de Google a través de la siguiente URL: https://code.google.com/apis/console

Luego se crea un proyecto nuevo, a no ser que ya tengamos uno creado. En este contexto , un proyecto es una instancia del paquete de servicios que Google ofrece bajo el nombre de “Google Play Services”.

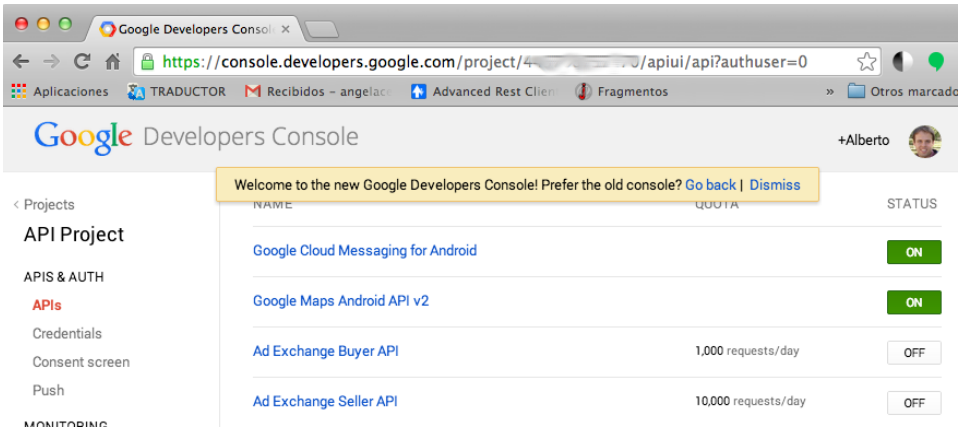


Una vez creado el proyecto, se va a “Mi proyecto”/MONITORING/Overview y se visualiza el “Project Number” (en la captura de abajo se muestra borroso) que se copiara en alguna parte ya que es necesario mas adelante.

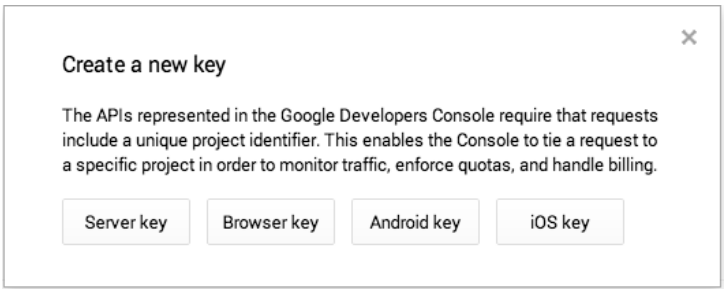


Una vez el Project Number esta guardado en alguna parte, se ira a “TU proyecto”/APIS&AUTH/APIs donde se visualiza un listado de todos los servicios disponibles en la Google Play Services, se activa el servicio “Google Cloud Messaging for Android”

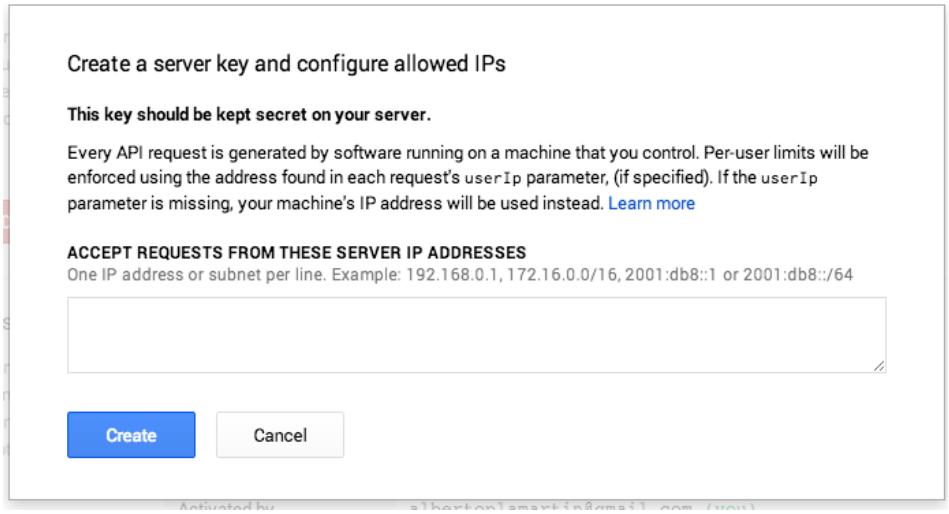
Ya solo falta generar el server Key, se va a “Mi proyecto”/APIS&AUTH/Credentials y se hace click en “Create new Key”. En la siguiente pantalla se ve que ya existe un “Key for server application” creado previamente pero en realidad no debe existir todavía en este paso.



Y luego se hace click en el botón Server Key



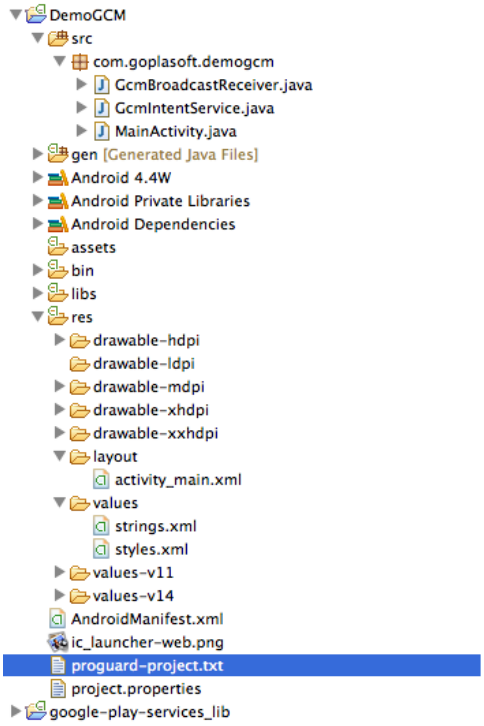
En la siguiente pantalla esta la opción de restringir la IP desde la cual el servidor con el Tomcat puede acceder al servicio GCM, si no se pone nada será posible acceder a dicho servicio desde cualquier IP. En este ejemplo se va a dejar vacío.



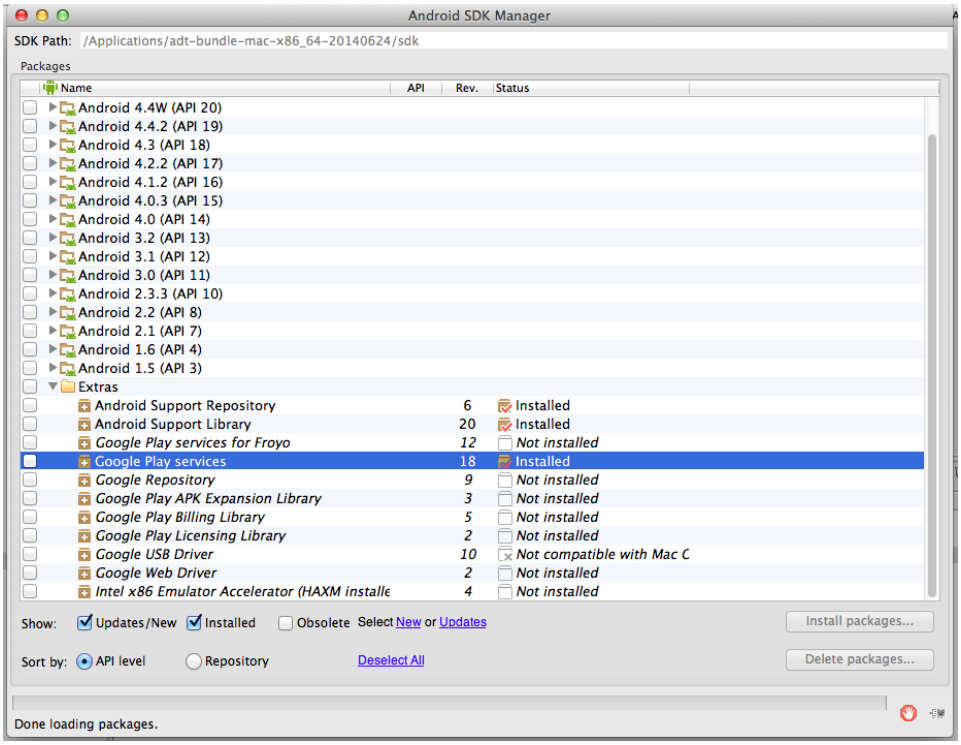
Después de que se ha creado el “Server key” copiamos al “Api key” en alguna parte ya que se necesitara mas adelante en el paso 5.

Ahora que ya se ha creado, activado y configurado el servicio GCM, el siguiente paso es crear un nuevo proyecto Android que hace de cliente y configurar la “Google Play Services” para que podamos invocar al servicio de registro.

En la siguiente pantalla se puede ver la estructura de la aplicación cliente ya finalizada.



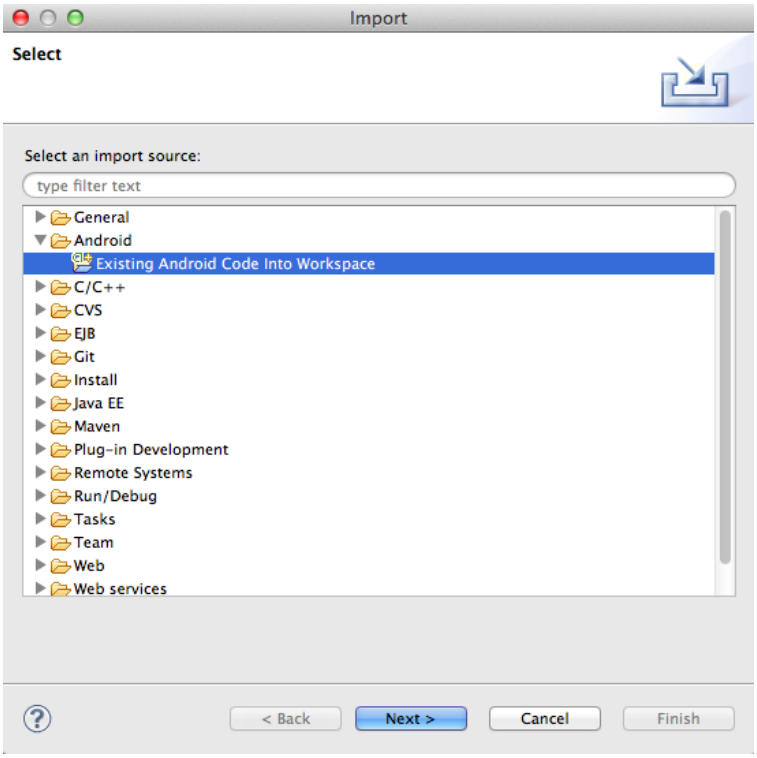
Para configurar la “Google Play Services” en nuestro proyecto debemos primero descargarnos la librería pertinente desde Android SDK Manager, concretamente en Extras/Google Play Services



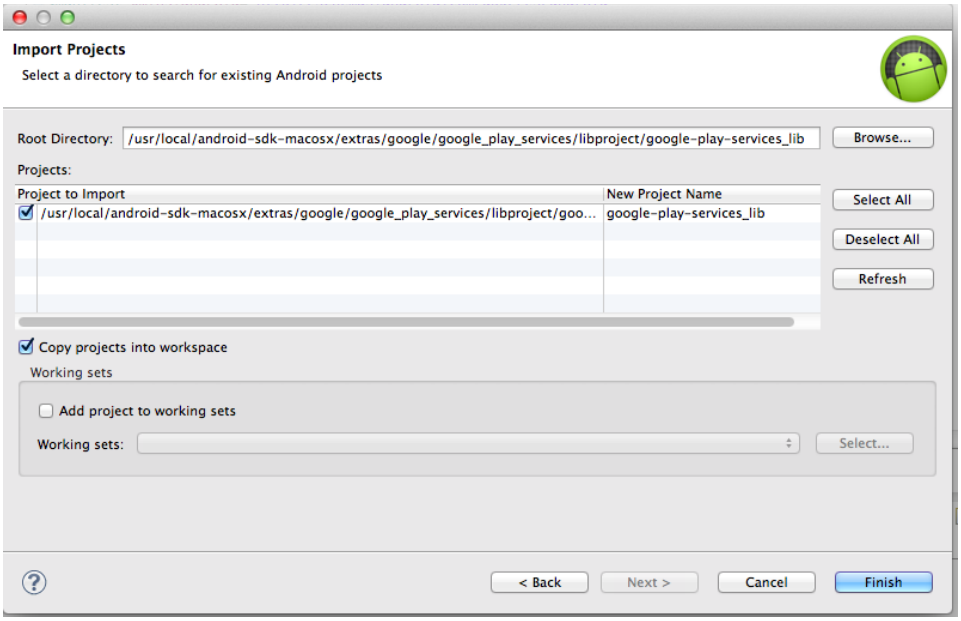
Una vez ya se ha descargado, se puede encontrar la librería en /extras/google/google_play_services/libproject/google-play-services_lib la cual se importa al workspace.

Para los que no estén familiarizado con el entorno de Android he decir que la forma en que se añaden librerías a un proyecto Android difiere con una aplicación jee, así que tomad buena nota de los siguientes pasos.

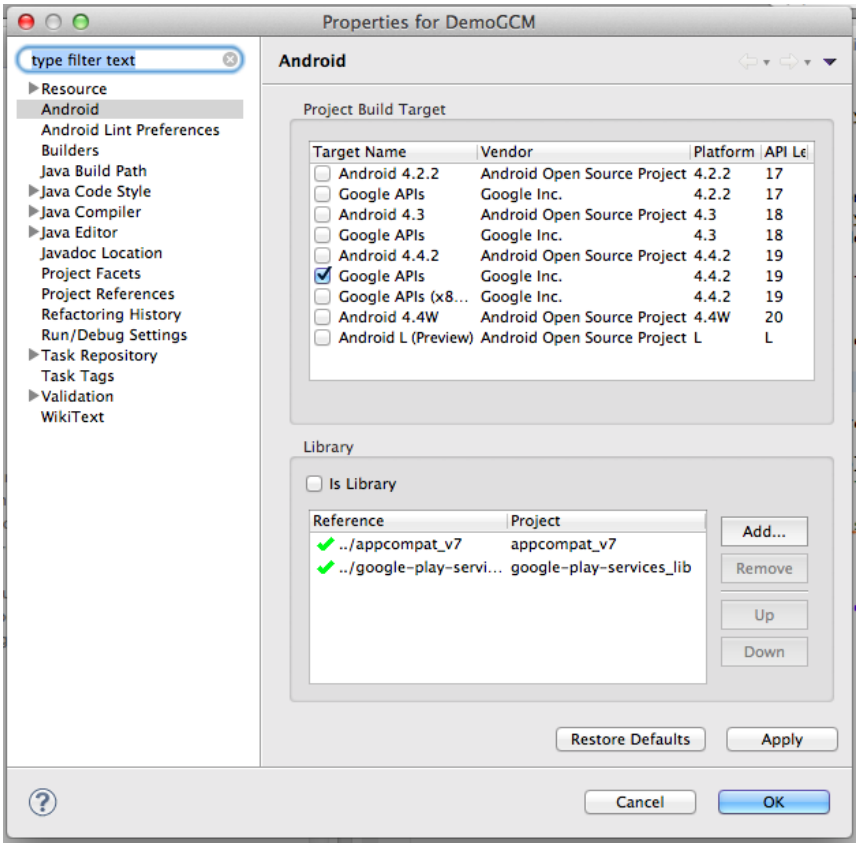
Para importar la librería se va a File -> Import -> Android -> Existing Android Code into Workspace y se busca la carpeta mencionada justo arriba.



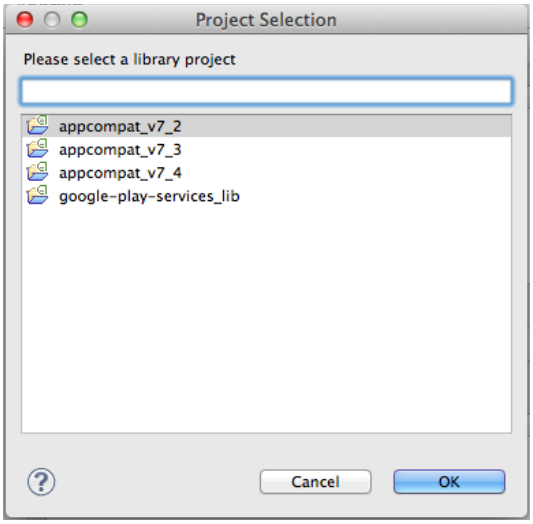
Una vez indicada la ubicación de la librería, marcamos el check “Copy proyects into workspace” y se hace click en el botón Finish.



Ahora se asocia la librería importada al proyecto Android que hace de cliente, para ello se hace click en el botón derecho sobre la raíz del proyecto, y en Properties->Android aparece una pantalla como la de abajo,primero en “Proyect Build Target” seleccionamos Google APIs y en segundo lugar, en el apartado “Library” se hace click en el botón Add para asociar una nueva librería



luego se selecciona google-play-services_lib y se hace click en OK



Y ya tendríamos la librería añadida a nuestro proyecto, ahora es preciso añadir algunas configuraciones para poder invocar los servicios de GCM. En el AndroidManifest.xml añadimos dentro de la etiqueta <application>

```
<meta-data android:name="com.google.android.gms.version" android:value="@integer/google
```

Y dentro de la tag <manifest> añadimos los siguientes permisos

```
  <permission android:name="com.goplasoft.democfg.permission.C2D_MESSAGE" android:prote
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<uses-permission android:name="com.google.android.c2dm.permission.REGISTER" />
<uses-permission android:name="com.goplasoft.democfg.permission.C2D_MESSAGE" />
```

Y en el fichero proguard-project.txt ubicado en la raíz del proyecto Android añadimos la siguiente configuración después de la última linea del mismo:

```
  -keep class * extends java.util.ListResourceBundle {
    protected Object[][] getContents();
}

-keep public class com.google.android.gms.common.internal.safeparcel.SafeParcelable {
    public static final *** NULL;
}

-keepnames @com.google.android.gms.common.annotation.KeepName class *
-keepclassmembernames class * {
    @com.google.android.gms.common.annotation.KeepName *;
}

-keepnames class * implements android.os.Parcelable {
    public static final ** CREATOR;
}
```

Ahora que esta instalado en el proyecto la librería de Google Play Service, se procede a explicar el código de la aplicación. Para empezar la interfaz de la actividad principal, en este ejemplo MainActivity no tiene nada, salvo un mensaje al iniciarse la misma. Esto es así porque para recibir una notificación lo único necesario es que la aplicación se ejecute una primera vez para se efectúe el registro, luego aunque la aplicación no este abierta se recibirá la notificación de todas formas.

El layout /DemoGCM/res/layout/activity_main.xml queda asi:

```
  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.goplasoft.demogcm.MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/arranque" />
</RelativeLayout>
```

Y el fichero de literales /DemoGCM/res/values/strings.xml queda así:

```
  <?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Ejemplo GCM</string>
    <string name="arranque">Al arrancar la aplicación se produce el registro en los servi
    <string name="action_settings">Settings</string>
</resources>
```

En el metodo onCreate() de la actividad principal, en este caso "/DemoGCM/src/com/goplasoft/demogcm/MainActivity.java", verificamos que Google Play Services esta operativo y si es así se procede con el registro.

Ahora se explica el código paso a paso, lo primero se declara las siguientes variables a nivel de clase del Activity:

```
// Url del servicio REST que se invoca para el envio del identificador de
// registro a la aplicación jee
public static final String URL_REGISTRO_ID = "http://XX.XX.XX.XXX:XXXX/gcmserver/webapi/
// Seña númerica que se utiliza cuando se verifica la disponibilidad de los
// google play services
```

```
6   private static final int PLAY_SERVICES_RESOLUTION_REQUEST = 9000;
7   // Una simple Tag utilizada en los logs
8   private static final String TAG = "Demo GCM";
9
10  public static final String EXTRA_MESSAGE = "message";
11  // Clave que permite recuperar de las preferencias compartidas de la
12  // aplicación el dentificador de registro en GCM
13  private static final String PROPERTY_REG_ID = "registration_id";
14  // Clave que permite recuperar de las preferencias compartidas de la
15  // aplicación el dentificador de la versión de la aplicación
16  private static final String PROPERTY_APP_VERSION = "appVersion";
17  // Identificador de la instancia del servicio de GCM al cual accedemos
18  private static final String SENDER_ID = "XXXXXXXXXXXX";
19  // Clase que da acceso a la api de GCM
20  private GoogleCloudMessaging gcm;
21  // Identificador de registro
22  private String regid;
23  // Contexto de la aplicación
24  private Context contexto;
```

La constante SENDER_ID debe de inicializarse con el “Project Number” que aparece en la Api console y que se dijo que se copiara en alguna parte previamente en este tutorial. En cuanto a la URL_REGISTRO_ID se tiene que reemplazar las X por la ip y puerto del servidor Tomcat.

Advertencia : Si se pone “localhost” y se prueba con un dispositivo Android físico no funciona ya que el Tomcat no esta instalado en el propio terminal Android sino en una servidor externo.

En el metodo onCreate() pondremos lo siguiente:

```
1   /**
2    * Nada mas arrancar la aplicación se procede al registro de la misma en la
3    * Google Play Services que engloban a los servicios GCM Se comprueba que
4    * Play Services APK esta disponibles, Si lo esta se proocede con el
5    * registro, de lo contrario se mostrara un dialogo para que el usuario se
6    * descargue la Google Play
7    */
8   @Override
9   protected void onCreate(Bundle savedInstanceState) {
10    super.onCreate(savedInstanceState);
11    setContentView(R.layout.activity_main);
12
13    contexto = this;
14    // Se comprueba que Play Services APK estan disponibles, Si lo esta se
15    // proocede con el registro en GCM
16    if (chequearPlayServices()) {
17      gcm = GoogleCloudMessaging.getInstance(contexto);
18      // Se recupera el "registration Id" almacenado en caso que la
19      // aplicación ya se hubiera registrado previamente
20      regid = obtenerIdentificadorDeRegistroAlmacenado();
21      // Si no se ha podido recuperar el id del registro procedemos a
22      // obtenerlo mediante el proceso de registro
23      if (regid.isEmpty()) {
24        // Se inicia el proceso de registro
25        registroEnSegundoPlano();
26      }
27    } else {
28      Log.i(TAG, "No valid Google Play Services APK found.");
29    }
30  }
```

En el primer if hay un método que se encarga de chequear que el paquete de Servicios de Google Play Services esta disponible, su código es el siguiente:

```
1   /**
2    * Este metodo comprueba si Google Play Services esta disponible, ya que
3    * este requiere que el terminal este asociado a una cuenta de google.Esta
4    * verificación es necesaria porque no todos los dispositivos Android estan
5    * asociados a una cuenta de Google ni usan sus servicios, por ejemplo, el
6    * Kindle fire de Amazon, que es una tablet Android pero no requiere de una
7    * cuenta de Google.
8    *
9    * @return Indica si Google Play Services esta disponible.
10   */
11   private boolean chequearPlayServices() {
12    int resultCode = GooglePlayServicesUtil
13        .isGooglePlayServicesAvailable(contexto);
14    if (resultCode != ConnectionResult.SUCCESS) {
15      if (GooglePlayServicesUtil.isUserRecoverableError(resultCode)) {
16        GooglePlayServicesUtil.getErrorDialog(resultCode, this,
17            PLAY_SERVICES_RESOLUTION_REQUEST).show();
18      } else {
19        Log.i(TAG, "Dispositivo no soportado.");
20        finish();
21      }
22      return false;
23    }
24    return true;
25  }
```

Posteriormente se intenta obtener el identificador de registro almacenado en el dispositivo en caso que dicho identificador haya sido cacheado, el código que hace esto es el siguiente:

```
1     /**
2      * Metodo que recupera el registration ID que fue almacenado la ultima vez
3      * que la aplicación se registro, En caso que la aplicación este
4      * desactualizada o no se haya registrado previamente no se recuperara
5      * ningón registration ID
6      *
7      * @return identificador del registro, o vacio("") si no existe o esta
8      *         desactualizado dicho registro
9      */
10    private String obtenerIdentificadorDeRegistroAlmacenado() {
11      final SharedPreferences prefs = getPreferenciasCompartidas();
12      String registrationId = prefs.getString(PROPERTY_REG_ID, "");
13      if (registrationId.isEmpty()) {
14        Log.i(TAG, "Registration not found.");
15        return "";
```

```
    }
    // Comprueba si la aplicación esta actualizada
    int registeredVersion = prefs.getInt(PROPERTY_APP_VERSION,
        Integer.MIN_VALUE);
    int currentVersion = getVersionDeLaAplicacion();
    if (registeredVersion != currentVersion) {
      Log.i(TAG, "App version changed.");
      return "";
    }
    return registrationId;
  }

/**
   * Metodo que sirve para recupera las preferencias compartidas en modo privado
   *
   * @return Application's {@code SharedPreferences}.
   */
  private SharedPreferences getPreferenciasCompartidas() {
    return getSharedPreferences(MainActivity.class.getSimpleName(),
        Context.MODE_PRIVATE);
  }

  /**
   * Recupera la versión aplicación que identifica a cada una de las
   * actualizaciones de la misma.
   *
   * @return La versión del codigo de la aplicación
   */
  private int getVersionDeLaAplicacion() {
    try {
      PackageInfo packageInfo = contexto.getPackageManager()
          .getPackageInfo(contexto.getPackageName(), 0);
      return packageInfo.versionCode;
    } catch (NameNotFoundException e) {
      // should never happen
      throw new RuntimeException("Could not get package name: " + e);
    }
  }
```

Si el identificador de registro no estuviese previamente almacenado entonces es necesario realizar el proceso de registro para obtener dicho identificador que se almacena en el propio dispositivo Android a modo de caché y también se envía al servidor Tomcat a través de un servicio REST. Mas adelante se ve como se implementa dicho servicio REST. El código que hace todo lo descrito es:

```
/**
 * En este método se procede al registro de la aplicación obteniendo el
 * identificador de registro que se almacena en la tarjeta de memoria para
 * no tener que repetir el mismo proceso la próxima vez. Adicionalmente se
 * envía el identificador de registro al a la aplicación jee , invocando un
 * servicio REST.
 */
private void registroEnSegundoPlano() {
  new AsyncTask<Object, Object, Object>() {
    @Override
    protected void onPostExecute(final Object result) {
      Log.i(TAG, result.toString());
    }

    @Override
    protected String doInBackground(final Object... params) {
      String msg = "";
      try {
        if (gcm == null) {
          gcm = GoogleCloudMessaging.getInstance(contexto);
        }
        // En este metodo se invoca al servicio de registro de los
        // servicios GCM
        regid = gcm.register(SENDER_ID);
        msg = "Dispositivo3 registrado, registration ID=" + regid;
        Log.i(TAG, msg);
        // Una vez se tiene el identificador de registro se manda a
        // la aplicacion jee
        // ya que para que esta envie el mensaje de la notificación
        // a los servidores
        // de GCM es necesario dicho identificador
        enviarIdentificadorRegistroALaAplicacionJ2ee();
        // Se persiste el identificador de registro para que no sea
        // necesario repetir el proceso de
        // registro la proxima vez
        almacenarElIdentificadorDeRegistro(regid);
      } catch (Exception e) {
        msg = "Error :" + e.getMessage();
        e.printStackTrace();
      }
      return msg;
    }

  }.execute(this, null, null);
}

/**
 * Se almacena el identificador de registro de "Google Cloud Message" y la
 * versión de la aplicación
 *
 * @param regId identificador de registro en GCM
 */
private void almacenarElIdentificadorDeRegistro(String regId) {
  final SharedPreferences prefs = getPreferenciasCompartidas();
  int appVersion = getVersionDeLaAplicacion();
  Log.i(TAG, "Saving regId on app version " + appVersion);
  SharedPreferences.Editor editor = prefs.edit();
  editor.putString(PROPERTY_REG_ID, regId);
  editor.putInt(PROPERTY_APP_VERSION, appVersion);
  editor.commit();
}
```

Advertencia: Es posible que cuando se ejecute el método de registro:

```
identificadorRegistro = gcm.register(SENDER_ID);
```

Se devuelva un error “service not available”, este es un error muy genérico que puede tener múltiples causas, para solucionarlo prueba con verificar que el reloj del terminal Android esta en hora, revisar los permisos y quitar los puntos de ruptura ubicados previamente o en esa misma linea de código si se esta ejecutando en modo debug.

4.- Reenvío del “Registation ID” al servidor Tomcat

En este paso se envía el identificador de registro que esta en posesión del cliente Android a la aplicación jee para que esta última pueda indicar al servicio de GCM el destinatario del mensaje de la notificación. El envío de dicho identificador se hace invocando un servicio REST por parte de la aplicación Android y cuya implementación esta hecha en la aplicación jee que en este ejemplo he llamado “gcmserver”.

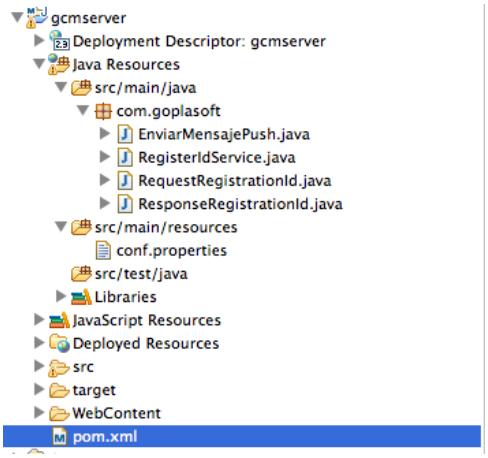
En el método registroEnSegundoPlano descrito en el paso 3 se invoca a otro método enviarIdentificadorRegistroALaAplicacionJ2ee donde esta el cliente del servicio REST. Dicho servicio se hace por método post y recibe los parámetros envueltos por un objeto JSON y este su vez responde con un mensaje que también esta envuelto en otro objeto JSON. A continuación se muestra el código del cliente del servicio:

```
/**
 * Se envía el identificador de registro de GCM mediante la invocación de un
 * servicio REST por el método POST, pasándole por parámetro un objeto json
 * que envuelve dicho identificador
 *
 * @param url
 *            URL del servicio REST al cual invocar
 * @param json
 *            Objeto json que contiene el identificador de registro a enviar
 * @return Devuelve un objeto json que contiene un mensaje de confirmación
 *         del envio del identificador del registro
 * @throws Exception
 */

private void enviarIdentificadorRegistroALaAplicacionJ2ee()
    throws Exception {
  JSONObject requestRegistrationId = new JSONObject();
  requestRegistrationId.put("registrationId", regid);
  BufferedReader in = null;
  try {
    HttpClient client = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost();
    httpPost.setURI(new URI(URL_REGISTRO_ID));
    httpPost.setEntity(new StringEntity(requestRegistrationId
        .toString(), "UTF-8"));
    httpPost.setHeader("Accept", "application/json");
    httpPost.setHeader("content-type", "application/json");

    HttpResponse response = client.execute(httpPost);
    InputStreamReader lectura = new InputStreamReader(response
        .getEntity().getContent());
    in = new BufferedReader(lectura);
    StringBuffer sb = new StringBuffer("");
    String line = "";
    while ((line = in.readLine()) != null) {
      sb.append(line);
    }
    in.close();
    Log.i("INFO", sb.toString());
  } catch (Exception e) {
    Log.e("ERROR", e.getMessage(), e);
  } finally {
    if (in != null) {
      try {
        in.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Ahora se procede a explicar la implementación del servicio REST en la parte del servidor. Para ello se ha utilizado JAX-RS y en concreto la implementación Jersey proporcionada por Oracle. La aplicación jee sigue la estructura estándar de Maven tal como se muestra en el siguiente captura:



Lo primero de todo es necesario añadir las dependencias de jersey en nuestro fichero /gcmserver/pom.xml, que son:

```
<dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet-core</artifactId>
```

```
      <version>${jersey.version}</version>
    </dependency>

    <dependency>
      <groupId>org.glassfish.jersey.media</groupId>
      <artifactId>jersey-media-moxy</artifactId>
      <version>${jersey.version}</version>
    </dependency>
```

Y las propiedades a añadir son:

```
<properties>
    <jersey.version>2.9</jersey.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
```

También tendremos que añadir el repositorio:

```
<repository>
    <id>snapshot-repository.java.net</id>
    <name>Java.net Snapshot Repository for Maven</name>
    <url>https://maven.java.net/content/repositories/snapshots/</url>
    <layout>default</layout>
  </repository>
```

Para poder empezar a usar Jersey se tiene que añadir las siguientes lineas en el fichero /gcmserver/src/main/webapp/WEB-INF/web.xml

```
   <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.goplasoft</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/webapi/*</url-pattern>
 </servlet-mapping>
```

El servlet recibe un parámetro cuya clave es “jersey.config.server.provider.packages” y su valor será el paquete donde dicho servlet empezará a escanear las clases en búsqueda de alguna que esté anotada con las anotaciones de JAX-RS. Se deberá por tanto adaptar la siguiente linea código:

```
<param-value>com.goplasoft</param-value>
```

A continuación se ve el código del servicio que lo único que hace es guardar el identificador de registro pasado por parámetro en un fichero de texto plano. Normalmente cada uno de los identificadores de registro de los múltiples clientes Android se almacenaría y gestionaría en una BBDD pero por motivos didácticos y para hacer el ejemplo mas simple no se utiliza ninguna.

La URL para invocar el servicio REST es:

```
http://XX.XX.XXX.XX:XXXX/webapi/registration/id/add
```

Donde webapi esta definido en el web.xml, “registration/id” esta definido en la anotación @Path("registration/id") a nivel de clase y finalmente /add en la anotación @Path("/add") a nivel de método. A continuación se muestra el código:

```
   @Path("registration/id")
public class RegisterIdService {
  // Ubicación del fichero donde se persistira el identificador de registro
  public static final String PATH = "/Users/albertopla/Documents/programacion/data-appli
  /**
   * Implementación del servicio REST que almacenara en el servidor el
   * identificador de registro pasado por parametro en un fichero de texto plano
   * @param requestRegisterId Objeto Json que envuelve el identificador de registro pasa
   * @return Devuelve un objeto JSON que envuelve el mensaje de respuesta con la confirm
   */
  @POST
  @Path("/add")
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON + ";charset=UTF-8")
  public ResponseRegistrationId addRegistationId(
      RequestRegistrationId requestRegisterId) {
    ResponseRegistrationId responseRegistrationId = new ResponseRegistrationId();
    try {
      BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(
          new File(PATH)));
      bufferedWriter.write(requestRegisterId.getRegistrationId());
      bufferedWriter.flush();
      bufferedWriter.close();
      responseRegistrationId.setCodeResponse(ResponseRegistrationId.OK);
      responseRegistrationId
          .setMessageResponse("Registro efectuado satisfactoriamente");
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      responseRegistrationId.setCodeResponse(ResponseRegistrationId.KAO);
      responseRegistrationId.setMessageResponse(e.getMessage());
    }
    return responseRegistrationId;
  }
}

/**
 * Objeto Json que envuelve el identificador de registro
 * @author albertopla
 */
public class RequestRegistrationId {
  //Identificador del registro
  private String registrationId;

  public String getRegistrationId() {
    return registrationId;
```

```
    }

    public void setRegistrationId(String registrationId) {
      this.registrationId = registrationId;
    }
  }

  /**
   * Objeto Json que encapsula la respuesta de confirmación o no del servicio REST
   * @author albertopla
   */
  public class ResponseRegistrationId {
    public static final int KAO=0;
    public static final int OK=1;
    //Codigo de la respuesta de confirmación
    private int codeResponse;
    //Mensaje de la respuesta
    private String messageResponse;

    public int getCodeResponse() {
      return codeResponse;
    }
    public void setCodeResponse(int codeResponse) {
      this.codeResponse = codeResponse;
    }
    public String getMessageResponse() {
      return messageResponse;
    }
    public void setMessageResponse(String messageResponse) {
      this.messageResponse = messageResponse;
    }
  }
```

5.- Envío del mensaje a los servidores GCM

En este paso se crea un formulario web donde se podrá introducir el mensaje de la notificación y enviar dicho mensaje al servicio de GCM que su vez lo reenviará al dispositivo Android correspondiente. Para ello se utiliza un par de jsp con JSTL y un servlet de los de toda la vida.

Lo primero es añadir las dependencias de Gson y JSTL en el pom.xml

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.2.4</version>
  </dependency>
  <dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
```

El /gcmserver/pom.xml al completo debe quedar así:

```
  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/X
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.goplasoft</groupId>
  <artifactId>gcmserver</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>gcmserver Maven Webapp</name>
  <url>http://maven.apache.org</url>

  <repositories>
    <repository>
      <id>snapshot-repository.java.net</id>
      <name>Java.net Snapshot Repository for Maven</name>
      <url>https://maven.java.net/content/repositories/snapshots/</url>
      <layout>default</layout>
    </repository>
  </repositories>

  <dependencies>
      <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.2.4</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.tomcat</groupId>
      <artifactId>servlet-api</artifactId>
      <version>6.0.41</version>
    </dependency>
    <dependency>
      <groupId>org.apache.tomcat</groupId>
      <artifactId>jsp-api</artifactId>
      <version>6.0.41</version>
    </dependency>
    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>

    <dependency>
      <groupId>org.glassfish.jersey.containers</groupId>
      <artifactId>jersey-container-servlet-core</artifactId>
```

```
      <version>${jersey.version}</version>
    </dependency>

    <dependency>
      <groupId>org.glassfish.jersey.media</groupId>
      <artifactId>jersey-media-moxy</artifactId>
      <version>${jersey.version}</version>
    </dependency>

  </dependencies>
  <build>
    <finalName>gcmserver</finalName>
    <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <inherited>true</inherited>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
        </plugins>
  </build>

  <properties>
    <jersey.version>2.9</jersey.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

</project>
```

A continuación se crea la primera JSP “/gcmserver/src/main/webapp/index.jsp” que se visualiza inicialmente y que contiene el formulario web donde se introduce el mensaje de la notificación:

```
  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  </head>
  <body>
    <form method="get" action="EnviarMensajePush">
      <input type="text" name="mensaje" size="20px">
      <input type="submit" value="Enviar mensaje">
    </form>
  </body>
</html>
```

Luego se crea el servlet EnviarMensajePush, dicho servlet se encarga de recoger del formulario web el mensaje de la notificación y reenviarlo al servicio REST de GCM para que este a su vez se lo mande al dispositivo Android pertinente.

Para invocar al servicio REST es necesario realizar la petición http por método post y además pasar una serie de parámetros tanto en la cabecera de la petición como en su cuerpo de la misma:

En la cabecera de la petición debemos indicar los siguientes parámetros:

```
  Authorization: key=XXXXXXXXXX
Content-Type: application/json
Accept-Encoding:application/json
```

En “Authorization” debemos reemplazar las X por la “ Api key” que se obtuvo de la “api console” en el paso 3 de este tutorial, y en el cual se dijo que se guardara justo después de haber creado el “Server key”.

En el cuerpo de la petición http pasaremos un objeto json que tendrá al menos los atributos “registration_ids” y “data” con una estructura similar a esta:

```
  {
  "registration_ids": ["4", "8", "15", "16", "23", "42"],
  "data" : {
    "mensaje":"Hola mundo",
    "otro posible atributo": "XXXX"

  },
}
```

“registration_ids” es un array de identificadores de registro, uno por cada dispositivo Android destinatario de la notificación, en este tutorial, como solo hay un único dispositivo destinatario de prueba el array contendrá un único identificador.

"data" es un objeto json que contiene el conjunto de datos de la notificación y cuya estructura no esta definida de antemano.

Ademas de los atributos obligatorios hay otros que son opcionales que en este tutorial no se van a utilizar pero no esta de más echar un vistazo:

Atributo	Descripción
notification_key	Una cadena que se asigna a un solo usuario con múltiples ID de registro asociados con dicho usuario. Esto permite que un servidor de 3 ª parte pueda enviar un mismo mensaje a varias instancias de la aplicación (por lo general en varios dispositivos), propiedad de un único usuario. Un servidor tercero-partido puede utilizar notification_key como destino de un mensaje en lugar de un ID de registro individual (o matriz de identificadores de registro). El número máximo de miembros permitidos para un notification_key es 10.
collapse_key	Una cadena arbitraria (como "Actualizaciones disponibles") que se utiliza para contraer un grupo de mensajes como cuando el dispositivo está en línea, por lo que sólo el último mensaje se envía al cliente. Con ello se pretende evitar enviar demasiados mensajes en el teléfono cuando este vuelve de nuevo a estar en linea. Tenga en cuenta que, dado que no hay ninguna garantía del orden en que los mensajes son enviados, el mensaje "último" en realidad no puede ser el último mensaje enviado por el servidor de aplicaciones.

delay_while_idle	Si se incluye, indica que el mensaje no debe ser enviado de inmediato si el dispositivo está inactivo. El servidor esperará a que el dispositivo se active, y luego será enviado sólo el último mensaje para cada valor collapse_key. El valor predeterminado es falso, y debe ser un booleano JSON.
time_to_live	Valor numérico que indica el tiempo en segundos por el cual debe mantenerse el mensaje en el almacenamiento de GCM si el dispositivo esta fuera de linea, por defecto esta puesto a 4 semanas
restricted_package_name	Una cadena que contiene el nombre del paquete de la aplicación. Cuando se establece, sólo se enviarán los mensajes de ID de registro que coinciden con el nombre de paquete.
dry_run	Si se incluye, permite a los desarrolladores probar su solicitud sin tener que enviar un mensaje. El valor predeterminado es falso, y debe ser un booleano en el objeto JSON.

A continuación se muestra el código del servlet EnviarMensajePush

```
1   public class EnviarMensajePush extends HttpServlet {
2   private Logger log = Logger.getLogger(EnviarMensajePush.class.getName());
3   //Url que necesitaremos para invocar el servicio de envio de notificaciones a los ser
4   public static String URL_GOOGLE_CLOUD_MESSAGE="https://android.googleapis.com/gcm/ser
5
6   //La API_KEY se inicializa con el valor obtenido desde la api console
7   public static String API_KEY="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
8   private static final long serialVersionUID = 1L;
9
10    /**
11     * @see HttpServlet#HttpServlet()
12     */
13    public EnviarMensajePush() {
14        super();
15    }
16
17  /**
18   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
19   */
20  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
21    //Recuperamos el mensaje de la notificación introducido y enviado a traves del form
22    String mensaje = request.getParameter("mensaje");
23    //Se lee el identificador de registro guardado previamente a traves del servicio RE
24    String idRegistro=recuperarIdRegistro();
25    //A partir de aqui se crea un objeto JSON que envuelve todos los parametros que le
26    JsonObject jsonObject = new JsonObject();
27    JsonObject data = new JsonObject();
28    data.addProperty("mensaje",mensaje);
29    JsonArray registration_ids = new JsonArray();
30    registration_ids.add(new JsonPrimitive(idRegistro));
31    /*
32     * Por convención el objeto Json tendrá como mínimo los siguientes atributos "data"
33     * aunque hay muchos otros atributos que son opcionales. En este ejemplo solo se pa
34     * de registro pero como pueden ser mas de uno estos se encapsulan en un array de i
35     * lo que es posible mandar una misma notificación a multiples dispositivos Androic
36     */
37    jsonObject.add("data",data);
38    jsonObject.add("registration_ids",registration_ids);
39    //Justo en la siguiente linea de codigo se invoca el servicio GCM de envio de notif
40    //y este nos devuelve una respuesta de confirmación
41    String respuesta = invocarServicioGCM(jsonObject.toString(),new URL(URL_GOOGLE_CLOU
42    log.info(respuesta);
43    //Se almacena el mensaje de la notificación en el contexto de request para luego po
44    request.setAttribute("mensaje", mensaje);
45    //Por ultimo redirigimos hacia la jsp que visualiza la confirmación del envio de la
46    getServletContext().getRequestDispatcher("/confirmacion.jsp").forward(request, resp
47  }
48
49  /**
50   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
51   */
52  protected void doPost(HttpServletRequest request, HttpServletResponse response) throw
53    doGet(request,response);
54  }
55  /**
56   * Metodo que permite recuperar el identificador de registro que asido previamente gu
57   * servicio REST implementado por la clase RegisterIdService.
58   * @return Devuelve el identificador de registro
59   * @throws IOException
60   */
61  private static final String recuperarIdRegistro() throws IOException{
62    BufferedReader bufferedReader = new BufferedReader(new FileReader(new File(Registe
63    String registroId = bufferedReader.readLine();
64    bufferedReader.close();
65    return registroId;
66  }
67  /**
68   * Metodo que implementa un cliente del servicio REST de GCM que se encarga del envio
69   * @param json Objeto Json que envuelve un array con los destinatarios (array con   i
70   * y  los datos de la notificación (el mensaje en este ejemplo)
71   * @param url Es la url del servicio REST de envio de notificaciones de GCM
72   * @param apikey Es la clace del servidor con la que podemos acceder a una instancia
73   * @return Devuelve la respuesta de confirmación del servicio REST
74   */
75  public static final String invocarServicioGCM(final String json, final URL url,final
76    try {
77      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
78      conn.setRequestMethod("POST");
79      conn.setRequestProperty("Content-Type", "application/json");
80      conn.setRequestProperty("Accept-Encoding", "application/json");
81      //Se pasa el Api key como parametro de la cabecera de la petición http
82      conn.setRequestProperty("Authorization","key=" +apikey);
83      if(json!=null){
84        conn.setDoOutput(true);
85        OutputStream os = conn.getOutputStream();
86        os.write(json.getBytes("UTF-8"));
87        os.flush();
88      }
89
90      if (conn.getResponseCode() != 200) {
91        throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode
92      }
```

```
        BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream
        String outputLine;
        StringBuffer totalSalida = new StringBuffer();
        System.out.println("Output from Server .... \n");
        while ((outputLine = br.readLine()) != null) {
          totalSalida.append(outputLine/*new String(outputLine.getBytes("ISO-8859-1"), "U
        }
        conn.disconnect();
        return totalSalida.toString();
      } catch (MalformedURLException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
      return null;
    }

}
```

Luego se registra en el web.xml el servlet EnviarMensajePush que gestiona la petición del formulario web descrito previamente de la siguiente forma y dentro de la etiqueta :

```
  <servlet>
    <servlet-name>EnviarMensajePush</servlet-name>
    <servlet-class>com.goplasoft.EnviarMensajePush</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>EnviarMensajePush</servlet-name>
    <url-pattern>/EnviarMensajePush</url-pattern>
</servlet-mapping>
```

El /gcmserver/src/main/webapp/WEB-INF/web.xml al completo debería quedar de la siguiente forma:

```
<web-app id="WebApp_ID">
  <display-name>gcmserver</display-name>
    <servlet>
    <servlet-name>EnviarMensajePush</servlet-name>
    <servlet-class>com.goplasoft.EnviarMensajePush</servlet-class>
  </servlet>

  <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.goplasoft</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

  <servlet-mapping>
    <servlet-name>EnviarMensajePush</servlet-name>
    <url-pattern>/EnviarMensajePush</url-pattern>
  </servlet-mapping>

    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/webapi/*</url-pattern>
    </servlet-mapping>

    <welcome-file-list>

      <welcome-file>index.jsp</welcome-file>

    </welcome-file-list>
</web-app>
```

Y ahora se crea la segunda JSP "/gcmserver/src/main/webapp/confirmacion.jsp" que muestra una confirmación de envío del mensaje una vez el servlet EnviarMensajePush ha enviado el mensaje de la notificación al servicio de GCM. El código de la jsp es el siguiente:

```
  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>El mensaje "<c:out value="${requestScope.mensaje}"/>" ha sido enviado</h1>
</body>
</html>
```

6.- Recepción de la notificación por parte del dispositivo.

En este momento, desde la aplicación web ya se puede enviar notificaciones a los servidores de GCM, ahora solo queda preparar la aplicación cliente para recibir la notificación que GCM reenvía al dispositivo Android. Para ello es necesario crear un BroadcastReceiver que hereda de WakefulBroadcastReceiver y que se encarga de capturar el intent com.google.android.c2dm.intent.RECEIVE que se produce cuando hay una notificación a la espera de ser tratada. El código del BroadcastReceiver es el siguente:

```
 /**
 * Lo único que hace este BroadcastReceiver es iniciar el servicio
 * GcmIntentService al capturar el intent, el servicio a su vez visualizara la
 * notificación en la barra de notificaciones ya que si se manipula directamente
 * la interfaz de usuario desde el propio BroadcastReceiver se corre el riesgo
 * que si el proceso dura mas de 5 segundos, el sistema operativo lance una
 * excepción, mientras que si se manipula la interfaz desde un servicio no
 * existe tal inconveniente
 *
```

```
 * @author albertopla
 *
 */
public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    // Especificar explicitamente que GcmIntentService debe manejar el
    // intent
    ComponentName comp = new ComponentName(context.getPackageName(),
        GcmIntentService.class.getName());
    // Se inicia el servicio, manteniento el dispositovo despierto mientras
    // se esta lanzando
    startWakefulService(context, (intent.setComponent(comp)));
    //
    setResultCode(MainActivity.RESULT_OK);
  }

}
```

También es necesario declarar explícitamente el BroadcastReceiver en el AndroidManifest.xml para así poder capturar el intent aunque la aplicación no este activa. A continuación se muestra las lineas de código que hay que añadir dentro de la tag <application>, después de hacer las adaptaciones pertinentes con el nombre de los paquete .

```
<receiver android:name="com.goplasoft.demogcm.GcmBroadcastReceiver"       android:permi
            <intent-filter>
                <action android:name="com.google.android.c2dm.intent.RECEIVE" />

                <category android:name="com.goplasoft.demogcm" />
            </intent-filter>
        </receiver>
```

Ahora se crea el servicio que se encarga de recuperar los datos de la notificación del intent que recibe por parámetro desde el BroadcastReceiver para luego mostrarlos en la barra de notificaciones. El codigo del servicio es:

```
  public class GcmIntentService extends IntentService {

  public static final int NOTIFICATION_ID = 1;
  private NotificationManager mNotificationManager;
  NotificationCompat.Builder builder;

  public GcmIntentService() {
    super("GcmIntentService");
  }
  /**
   * Metodo que recupera el mensaje de la notificación contenida en el intent
   * para luego mostrar dicho mensaje en la barra de notificaciones del dispositivo
   */
  @Override
  protected void onHandleIntent(Intent intent) {
    GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);

    String messageType = gcm.getMessageType(intent);
    Bundle extras = intent.getExtras();

    if (!extras.isEmpty()) {
      if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE.equals(messageType)) {
        //Se visualiza el mendaje en la barra de notificaciones
        sendNotification(extras.getString("mensaje"));
      }
    }

    GcmBroadcastReceiver.completeWakefulIntent(intent);
  }

  /**
   * Este metodo lo que hace es visualizar una notificación en la barra de
   * notificaciones con el mensaje pasado por parametro
   *
   * @param msg mensaje que se muestra en la notificación
   */
  private void sendNotification(String msg) {
    mNotificationManager = (NotificationManager) this
        .getSystemService(Context.NOTIFICATION_SERVICE);

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, MainActivity.class), 0);

    NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(
        this).setSmallIcon(R.drawable.ic_launcher)
        .setContentTitle("Notificacion:" + msg)
        .setStyle(new NotificationCompat.BigTextStyle().bigText(msg))
        .setContentText(msg);

    mBuilder.setContentIntent(contentIntent);
    mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
  }

}
```

Solamente falta declarar el servicio en el AndroidManifest.xml con la siguiente linea de código dentro de la etiqueta <application> después de hacer las adaptaciones pertinentes con los nombre de los paquete.

```
<service android:name="com.goplasoft.demogcm.GcmIntentService" />
```

Al final el AndroidManifest.xml debe quedar de manera similar a este:

```
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.goplasoft.demogcm"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="21" />

    <meta-data
```

```
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />

    <permission android:name="com.goplasoft.demogcm.permission.C2D_MESSAGE" android:prot
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="com.google.android.c2dm.permission.REGISTER" />
    <uses-permission android:name="com.goplasoft.democfg.permission.C2D_MESSAGE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name="com.goplasoft.demogcm.GcmBroadcastReceiver"  android:per
                <intent-filter>
                    <action android:name="com.google.android.c2dm.intent.RECEIVE" />

                    <category android:name="com.goplasoft.demogcm" />
                </intent-filter>
          </receiver>
          <service android:name="com.goplasoft.demogcm.GcmIntentService" />
    </application>
</manifest>
```

Y esto esto todo, ya se puede probar si llega la notificación al terminal Android, hay que tener en cuenta que a veces no es algo inmediato y que puede durar unos cuantos segundos.

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

IMPULSA Impulsores Comunidad ¿Ayuda?

---- sin clicks

0 personas han traído clicks a esta página

powered by karmacracy

Copyright 2003-2014 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto