Trabajo Final

Marcos Pividori (marcos.pividori@gmail.com)

R-313 Análisis de Lenguajes de Programación

Índice

1. Introducción

El proyecto consiste en construir un sistema de comunicación entre una computadora y un conjunto de celulares, de manera de poder obtener datos del estado de los mismos. Para esto, se desarrollaron dos programas, por un lado un servidor, escrito en Haskell y por otro, una aplicación Android (Java) para los celulares.

2. Google Cloud Messaging

2.1. Descripción

Google Cloud Messaging [5] es un servicio que permite enviar datos desde el servidor a los usuarios de dispositivos Android. Este puede ser un mensaje de bajo peso, avisando a la aplicación que hay nueva información que recoger del servidor, o podría ser un mensaje que contiene hasta 4 KB de datos de carga útil.

El servicio GCM gestiona todos los aspectos de la cola de mensajes y la entrega a la aplicación de destino, corriendo en el dispositivo. GCM es completamente gratis, no importa cuán grande son las necesidades de mensajería.

2.2. Proceso

A continuación se muestran los pasos necesarios para que finalmente un mensaje pueda llegar a un dispositivo móvil Android:

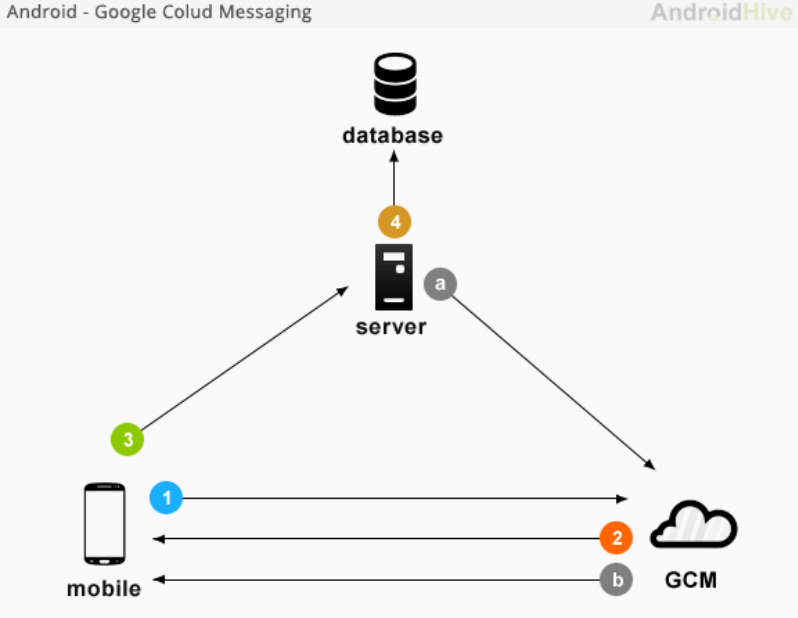
- **Sender ID**: Se obtiene desde la página de Google de APIs Console.
- **Application ID**: Identificador de la aplicación que se está registrando para recibir mensajes.
- **Registration ID**: Identificador enviado por los servidores GCM. Está asociado a una aplicación concreta existente en un dispositivo concreto.

1. La aplicación se registra en el dispositivo móvil para recibir mensajes. Se realiza enviando el **Sender ID** y **Application ID** al servidor **GCM**.
2. Si el registro ha terminado correctamente, el servidor **GCM** devolvería un **Registration ID**, que identificara al dispositivo.
3. La aplicación envía el **Registration ID** al servidor de terceros (nuestro servidor) para que sea guardado.
4. El Servidor lo guarda en la Base de Datos.

Nuestro servidor ya puede enviar un mensaje para que sea recibido por el dispositivo:

a Envía el mensaje a los servidores **GCM**, donde es puesto en cola.

b **Google** envía el mensaje al dispositivo cuando se encuentre accesible.



3. Panorama general

El proyecto se dividió en dos partes:

Por un lado, construir un servidor, que proporcione servicios mensajería JSON de manera de poder seguir el procedimiento GCM, antes descripto. Además, se desea contar con un servicio Web que permita a los usuarios acceder a los datos obtenidos desde cualquier lugar. Dicho servidor, se desarrolló en Haskell y usando el conjunto de librerías Yesod [1].

Por otro lado, desarrollar una aplicación para Android que pueda seguir el procedimiento GCM y proveer algunos datos útiles al usuario, ante los pedidos del servidor.

4. Servidor

La idea es tener un servidor, que permite ejecutar diferentes programas a la vez. Cada proceso, está definido por un **ProcessID** que lo identifica y está asociados a un celular en particular (aunque un celular puede tener muchos procesos). El servidor se inicializará con la función **runServer** que tomará un programa que será el programa por defecto ejecutado en cada celular. El servidor provee un conjunto de direcciones con distintos fines.

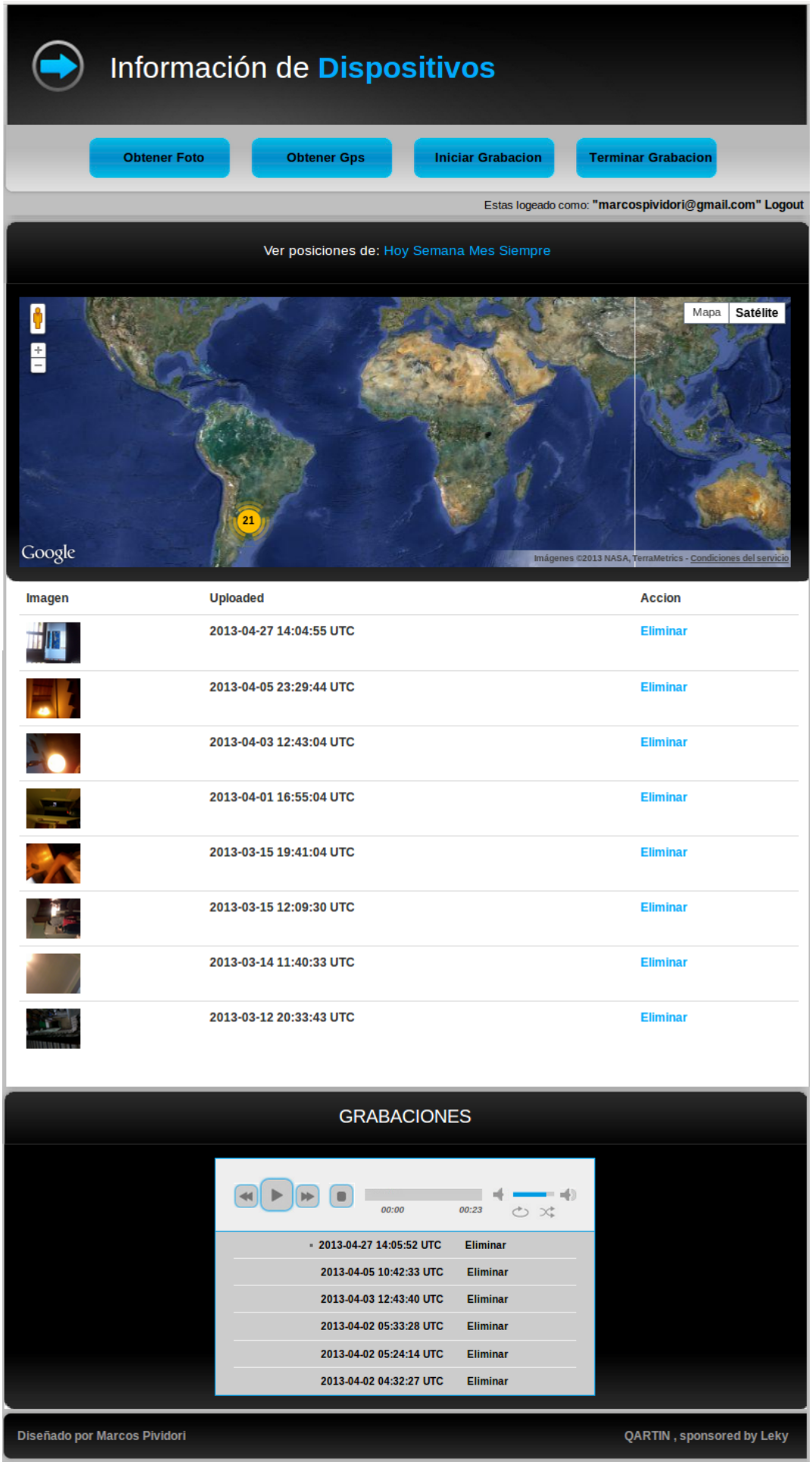
- /: Es la página Raíz, que redireccionará a la página de ingreso o a la del usuario según esté o no logueado. (GET)
- **/image/ImageId**: Provee acceso a las imágenes guardadas en la base de datos. (DELETE, GET)
- **/audio/AudioId** : Provee acceso a las grabaciones guardadas en la base de datos. (GET)
- **/audiodelete/AudioId** : Permite eliminar las grabaciones guardadas en la base de datos. (GET)
- **/userinfo/Periodo**: Página principal del usuario, donde puede acceder a la información disponible de su celular, como ser las distintas posiciones GPS, imágenes tomadas y grabaciones. También podrá pedir una nueva imagen , posición GPS o iniciar/terminar una grabación, con los botones disponibles. (GET)
- **/register**: Dirección donde son enviados los datos de los celulares al registrarse (Usuario,Contraseña,Registration ID). (POST)
- **/fromdevices**: Dirección donde son enviados los mensajes respuestas a instrucciones, desde los celulares. (POST)
- **/upload**: Dirección donde son enviadas las imágenes a guardar en la BD, desde los celulares. (POST)
- **/fromweb**: A esta dirección se envían los pedidos de nuevas posiciones o imágenes de la web. (POST)
- **/locations/Periodo**: A esta dirección se envían las consultas de las posiciones guardadas en la BD. (GET)
- **/static**: Dirección donde se guardan los datos estáticos.
- **/auth**: Subdominio de autentificación.

A grandes rasgos, el servidor interactuará por un lado con los celulares, permitiéndoles registrarse, subir imágenes y enviar datos. Por otro lado, proveerá un servicio web, que le permite al usuario ver estos datos y hacer ciertos pedidos.

4.1. Servicio Web:

Aquí se muestra un ejemplo de cómo se ve la página de un usuario logueado:

- Una barra principal, que proporciona la posibilidad de solicitar datos al servidor.
- Un mapa que muestra las posiciones obtenidas del celular. Se puede seleccionar un período de tiempo determinado y al posicionar el mouse sobre la posición se visualiza la fecha exacta (usando Apis de Google Maps[9]).
- Una lista de las imágenes tomadas, con la posibilidad de eliminarlas.
- Un reproductor de música donde se pueden escuchar/eliminar las grabaciones tomadas (usando Apis de jPlayer[10]).


Información de Dispositivos
Obtener Foto
Obtener Gps
Iniciar Grabacion
Terminar Grabacion
Estas logeado como: "marcospividori@gmail.com" Logout
Ver posiciones de: Hoy Semana Mes Siempre
Mapa
Satélite
21
Google
Imágenes ©2013 NASA, TerraMetrics - Condiciones del servicio
Imagen
Uploaded
Accion
2013-04-27 14:04:55 UTC
Eliminar
2013-04-05 23:29:44 UTC
Eliminar
2013-04-03 12:43:04 UTC
Eliminar
2013-04-01 16:55:04 UTC
Eliminar
2013-03-15 19:41:04 UTC
Eliminar
2013-03-15 12:09:30 UTC
Eliminar
2013-03-14 11:40:33 UTC
Eliminar
2013-03-12 20:33:43 UTC
Eliminar
GRABACIONES
00:00
00:23
2013-04-27 14:05:52 UTC
Eliminar
2013-04-05 10:42:33 UTC
Eliminar
2013-04-03 12:43:40 UTC
Eliminar
2013-04-02 05:33:28 UTC
Eliminar
2013-04-02 05:24:14 UTC
Eliminar
2013-04-02 04:32:27 UTC
Eliminar
Diseñado por Marcos Pividori
QARTIN , sponsored by Leky

4.2. EDSL

Para construir los programas que se ejecutarán en el servidor y interactuarán con los celulares, se desarrolló un Lenguaje Específico de Dominio Embebido (EDSL) sobre el tipo de datos **(Server a)** que se encuentra en el módulo **Server**. Para esto, se tuvo en cuenta qué casos debe diferenciar la función de ejecución **runProgram**. Por ejemplo, cuando espera un mensaje del celular, debe parar la ejecución y continuar recién cuando este llegue.

```
data Server a = Enviar [Celular] String (Server a)
                -- ^ Representa el envio de un mensaje a celulares
              | Leer (String -> Celular -> Server a)
                -- ^ Representa la pausa hasta la llegada de un mensaje.
              | Interpretar (String -> Celular -> Server a)
                -- ^ Representa la interpretación de un mensaje que ha llegado.
              | CelularLocal (Celular -> Server a)
                -- ^ Representa la solicitud de los datos del celular que ejecuta el
                   programa.
              | RegistroTotal ([Celular] -> Server a)
                -- ^ Representa la solicitud del registro total de celulares en el Server
              | Guardar Posicion (Server a)
                -- ^ Representa la operación de guardar una posición en la Base de Datos.
              | Error
                -- ^ Representa un error en algunas de las operaciones.
              | Ret a
                -- ^ Representa un valor de retorno a.
```

A partir de estos constructores, se crean algunas funciones útiles:

```
-- |Programa vacío.
emptyProgram :: Server ()

-- |'enviar' envía el mensaje a la lista de celulares.
enviar :: [Celular] -> MensajeCelular -> Server ()

-- |'leer' espera la llegada de un mensaje.
leer :: Server (String,Celular)

-- |'celularLocal' obtiene los datos del celular que ejecuta el programa.
celularLocal :: Server Celular

-- |'registroTotal' obtiene una lista de los celulares registrados en el Server.
registroTotal :: Server [Celular]

-- |'enviarConAviso' envía el mensaje a la lista de celulares 'con aviso', es decir, que
    el usuario del celular será notificado de la llegada del mismo.
enviarConAviso :: [Celular] -> MensajeCelular -> Server ()

-- |'getGps' obtiene la posición GPS del celular dado y devuelve Nothing (No hay señal) o
    Just (Latitud,Longitud,Fecha)
getGps :: Celular -> Server (Maybe (Posicion))

-- |'getFoto' pide una foto al celular, que será guardada en la Base de Datos.
getFoto :: Celular -> Server ()

-- |'grabarInicio' comienza la grabación de audio en el celular.
grabarInicio :: Celular -> Server ()

-- |'grabarFinal' termina la grabación de audio en el celular.
grabarFinal :: Celular -> Server ()

-- |'wait' pide al celular que espere un número de milisegundos.
wait :: Celular -> Int -> Server ()

-- |'guardar' toma una posición y la guarda en la Base de Datos.
guardar :: Posicion -> Server ()

-- |'recibirTextoDeCelular' establece que se desea recibir los mensajes de la consola del
    celular dado.
recibirTextoDeCelular :: Celular -> Server ()
```

4.3. Organización de los archivos

El desarrollo del servidor se dividió en módulos de acuerdo a la funcionalidad de cada uno. Está conformado por:

- **Programas_EDSL**: Muestra 4 ejemplos claros de programas escritos usando el EDSL.
- **Principal**: Este módulo une el resto de los módulos y proporciona la función principal: runServer.
- **Base**: Este módulo construye la estructura principal del server 'Messages'. Define los handlers para los diferentes pedidos, y maneja la ejecución de los programas escritos en el EDSL.
- **Almacenamiento**: Este Módulo presenta las herramientas para acceder a la representación en memoria de los programas en ejecución en el Server.
- **DataBase**: Este Módulo presenta la configuración de la Base de Datos utilizada, y los tipos de datos almacenados en ella.
- **SendGCM**: Este Módulo presenta las herramientas para comunicarse con el GCM Server.
- **Server**: Este Módulo presenta cómo se construyen los programas que se ejecutan en el Server. Es decir, el EDSL sobre el tipo de datos **Server a**.
- **Settings**: Provee algunas funciones útiles para estructurar el código Yesod.

Además se pueden visualizar 3 directorios:

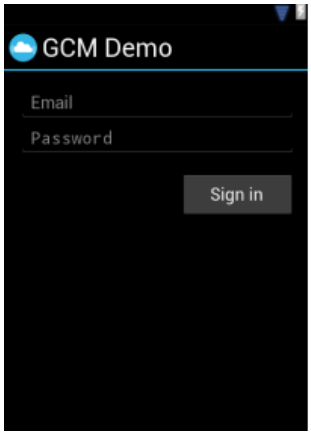
- **datos**: En este directorio se guardarán los datos particulares de cada usuario, como ser las imágenes y grabaciones tomadas.
- **static**: En este directorio se guardan los datos estáticos. Es decir, incluye los archivos necesarios para el funcionamiento del sitio web, como también el código de la aplicación para Android.
- **templates**: Incluye el conjunto de archivos que definen el funcionamiento del sitio web. Estos, fueron desarrollados en Shakespearean Languages, es decir se usan 3 lenguajes: Hamlet (para HTML), Julius (para JavaScript), and Cassius (para CSS).

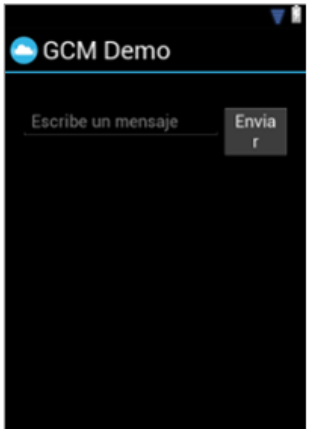
5. Aplicación del celular

La aplicación está desarrollada para dispositivos que usen Android y se podrá descargar de la dirección **/static/InformacionDeDispositivos.apk** . Esencialmente consiste en una actividad, que al ser iniciada por primera vez solicita un usuario (cuenta de Gmail) y contraseña (no necesariamente la misma de la cuenta Gmail), y se registra en el servidor GCM obteniendo su Registration Id. Luego envía todos estos datos a nuestro servidor y se registra también allí.

Una vez concretado el registro, se lanza un servicio que actuará de fondo todo el tiempo, esté o no abierta la aplicación. Este servicio responderá los pedidos del servidor en cualquier momento, sin que el usuario del celular lo note, y se reiniciará automáticamente al encenderse el celular.

Ya registrado, cada vez que se abra la aplicación, aparecerá una consola que permite ver algunos comandos, y ,de ser activada, permite enviar texto al servidor. De esta manera se pueden construir programas de intercambio de mensajes entre celulares.





5.1. Organización de los archivos

La aplicación se desarrolló en Java, usando las API de Android[8], y tomando como base el ejemplo provisto por el servicio de GCM[6]. Principalmente se dividió en las siguientes clases:

- **ActividadPrincipal**: Principal actividad para la aplicación. Mostrara una consola en la que se imprimen mensajes y se pueden enviar.
- **ServicioPrincipal**: Principal servicio que corre de fondo y continuamente maneja los pedidos del servidor.
- **CommonUtilities**: Esta clase provee métodos y constantes comunes a otras clases en la aplicación.
- **GCMIntentService**: Responsable de manejar los mensajes provenientes de los servidores que brindan el servicio GCM.
- **Registro**: Actividad que permite registrar al usuario.
- **ServerUtilities**: Clase usada para comunicarse con el servidor.
- **BootReceiver**: Con esta clase, nos aseguramos que el servicio principal se inicie al prender el celular.

6. Funcionamiento general

De esta manera entonces el funcionamiento general será:

- El server estará continuamente funcionando.
- Cuando un celular se registre, comenzará un nuevo proceso de ejecución del programa predeterminado, asociado a dicho celular.
- En cualquier momento, un usuario podrá entrar al servicio web y, logueandose con la cuenta de Gmail con que se ha registrado en el celular, ver u obtener nuevos datos del mismo.

7. Ejemplos de programas

Para iniciar el servidor, con un programa por defecto, utilizamos la función **runServer** del módulo **Principal**:

```
runServer :: Server () -> IO ()
```

7.1. Ejemplo 1

El siguiente programa, cada cierto tiempo obtendrá la posición del celular asociado y la del resto de los celulares registrados. Si están cerca suyo, le avisará al celular y al resto.

```
import Server

program1 :: Server ()
program1 =
  do
    let recursion = do
          lista <- registroTotal
          cel_local <- celularLocal
          let rec1 = do
                posicion_local <- getGps cel_local
                if posicion_local == Nothing
                then rec1
                else return posicion_local
          Just posicion <- rec1
          let quitarme [] = []
              quitarme (x:xs) = if x == cel_local
                                then xs
                                else (x : quitarme xs)
              lista'= quitarme lista
              rec [] = return ()
              rec (x:xs) = do
                pos <- getGps x
                case pos of
                  Nothing   -> rec xs
                  Just ubic -> do
                    distancia <- calculate_dist (lat posicion,lng posicion) (lat ubic,lng
                         ubic)
                    if (distancia  < 100)
                    then do
                           enviarConAviso [x] (user(cel_local)++" esta cerca, a "++show(
                               distancia)++"ms")
                           enviarConAviso [cel_local] (user(x)++" esta cerca, a "++show(
                               distancia)++"ms")
                           rec xs
                    else rec xs
          rec lista'
          wait cel_local 5000
          recursion
    recursion
    return ()

--Calcula la distancia en metros de dos posiciones
calculate_dist :: (Double,Double) -> (Double,Double) -> Server Double
calculate_dist (lat1,lon1) (lat2,lon2)=
  let
     r = 6371
     dLat = (lat2-lat1) * pi / 180
     dLon = (lon2-lon1) * pi / 180
     lat11 = lat1 * pi / 180
     lat22 = lat2 * pi / 180
     a = sin(dLat/2) * sin(dLat/2) + sin(dLon/2) * sin(dLon/2) * cos(lat11) * cos(lat22)
     c = 2 * (atan2  (sqrt a)  (sqrt(1-a)) )
     d = r * c
  in return d

main :: IO ()
main = runServer program1
```

7.2. Ejemplo 2

El siguiente programa, crea un servicio de mensajería. Escuchará los mensajes que lleguen del celular y los reenviará al resto de los celulares. Si se establece como programa por defecto en el servidor, todos los programas registrados tendrán acceso al servicio, escribiendo y recibiendo mensajes a través de la consola de la aplicación.

```
import Server

program2 :: Server ()
program2 =
 do
   cel_local <- celularLocal
   recibirTextoDeCelular cel_local
   let rec = do
               (msj,origen) <- leer
               if origen == cel_local
                  then do
                          lista <- registroTotal
                          let
                            quitar_cel_local [] = []
                            quitar_cel_local (x:xs) = if x == cel_local
                                                      then xs
                                                      else (x : quitar_cel_local xs)
                            lista'= quitar_cel_local lista
                          enviarConAviso lista' ((user cel_local)++" escribe: "++msj)
                  else rec
               rec
   rec

main :: IO ()
main = runServer program2
```

7.3. Ejemplo 3

El siguiente programa, cada cierto tiempo obtiene la posición GPS y la guarda en la base de datos.

```
import Server

program3 :: Server ()
program3 = do
             cel_local <- celularLocal
             let  rec = do
                           mi_posicion <- getGps cel_local
                           case mi_posicion of
                              Nothing  -> return ()
                              Just pos -> guardar pos
                           wait cel_local 20000
                           rec
             rec
             return ()

main :: IO ()
main = runServer program3
```

7.4. Ejemplo 4

El siguiente programa, cada cierto tiempo obtiene una foto que es automáticamente guardada en la Base de Datos.

```
program4 :: Server ()
program4 = do
             cel_local <- celularLocal
             let  rec = do
                           getFoto cel_local
                           wait cel_local 10000
                           rec
             rec
             return ()

main :: IO ()
main = runServer program4
```

Referencias

[1] Developing Web Applications with Haskell and Yesod. Safety-Driven Web Development. Michael Snoyman (2012). O'Reilly Media. ISBN-10: 1449316972

[2] Yesod Web Framework - Sitio Web.

[3] Ejemplo de desarrollo de una pagina web en Yesod.

[4] Android Developers - Sitio Web.

[5] Google Cloud Messaging - Sitio Web.

[6] GCM Demo App.

[7] Wikipedia.

[8] API Android.

[9] API Google Maps.

[10] API JPlayer.