Trabajo de Fin de Grado

Diseño, desarrollo e implementación de una aplicación m-government para el Parlamento de Canarias

Design, development and implementation of an m-government application for the Parliament of Canary Islands

Víctor Hernández Pérez

La Laguna, 4 de septiembre de 2017

Dª. **Elena Sánchez Nielsen**, con N.I.F. 42.848.599-J profesora titular adscrita al Departamento de Ingeniería Informática de Sistemas de la Universidad de La Laguna, como tutora

C E R T I F I C A

Que la presente memoria titulada:

"*Diseño, desarrollo e implementación de una aplicación m-government para el Parlamento de Canarias*"

ha sido realizada bajo su dirección por D. **Víctor Hernández Pérez**, con N.I.F. 78.643.409-S.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de septiembre de 2017

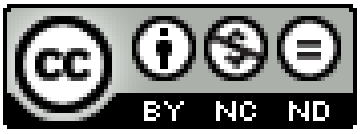
Agradecimientos

Gracias a mi tutora, Elena Sánchez Nielsen, por la orientación y ayuda prestada durante el desarrollo del presente trabajo.

Mencionar también a todos aquellos compañeros de clase con los que he tenido el placer de enriquecer mutuamente nuestros conocimientos y debatir acerca de diferentes ámbitos dentro de lo que a la ingeniería informática respecta.

Finalmente darle las gracias a mi familia y amigos por ayudarme en esta bonita etapa de formación.

Licencia



Resumen

El objetivo de este trabajo ha sido diseñar e implementar una aplicación móvil que permita seguir la actividad en instituciones parlamentarias, en concreto, del Parlamento de Canarias. De esta manera se le permitirá al usuario consultar información acerca de diputados y los distintos órganos, además de poder recibir notificaciones en el dispositivo móvil acerca de información de interés sobre la actividad parlamentaria. Además, con el fin de manejar la generación y envío de notificaciones ha sido necesario el desarrollo de un servidor que permita realizar ese tipo de gestiones.

Las herramientas que se han utilizado para el desarrollo del trabajo han sido: Node.js, Firebase y PostgreSQL (para la parte del servidor) y Phonegap, HTML5, CSS, JavaScript, jQuery y jQuery Mobile (para la parte de la aplicación móvil). Además para el despliegue del servidor se ha hecho uso de la plataforma Heroku y también se ha utilizado Android Studio con respecto al despliegue de la aplicación en Android.

Este documento está compuesto por los siguientes apartados, que serán descritos en profundidad en los diferentes capítulos: definición de aplicación móvil, evaluación de las plataformas de desarrollo de aplicaciones móviles, herramientas y tecnologías empleadas en el desarrollo, generación de notificaciones push, despliegue, conclusiones, líneas futuras del trabajo y posibles mejoras, además de un capítulo con el presupuesto necesario para desarrollar la aplicación.

Palabras clave: Phonegap, m-government, actividad parlamentaria, Android, IOS

Abstract

The aim of this study has been designing and implement a mobile app that allows to follow the activity in parliamentary institutions, in particular, the Parliament of the Canary Islands. So the user could consult information about the members and parliamentary bodies, besides being able to receive notifications about the parliamentary activity. Also, with the purpose of handle the generation and sending of notifications, it has been necessary to develop a server that allows to make this type of managements.

The tools that have been used for the development of the work have been: Node.js, Firebase and PostgreSQL (for the server side) and Phonegap, HTML5, CSS, JavaScript, jQuery and jQuery Mobile (for the mobile application). Also for the server has been used the Heroku platform and Android Studio for the deployment of the application on Android.

This document consist of the following sections, which it will be explained in depth in the different chapters: mobile application definition, evaluation of mobile application development platforms, tools and technologies used in the development, push notifications generation, deployment, conclusions, future works and improvements, plus a chapter with the necessary budget to develop the application.

Keywords: *Phonegap, m-government, parliamentary activity, Android, IOS*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Una aplicación móvil, está diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Éstas se encuentran disponibles a través de plataformas de distribución de los distintos sistemas operativos, por ejemplo App Store en el caso de iOS y Google Play en el caso de Android.

El presente Trabajo Fin de Grado (TFG) se enmarca en el ámbito de las aplicaciones móviles, se pretende diseñar e implementar un *Mobile goverment* (mGoverment) [1] para el Parlamento de Canarias con el fin de que los usuarios puedan consultar información de interés acerca de las actividades parlamentarias.

Cuando hablamos del término *mGoverment* nos referimos a un conjunto de servicios y aplicaciones gubernamentales destinadas al uso en dispositivos móviles, portátiles, *personal digital assistants* (PDAs), etc.

El primer *mGoverment* del mundo nace de la mano del profesor Ibrahim Kushchu en el *Mobile Goverment Consortium Int* (mGCI) en Japón. Los defensores de los servicios *mGoverment* opinan que éstos pueden ayudar a hacer que los servicios gubernamentales e información pública sean accesibles en cualquier momento y lugar.

Capítulo 2

Estado del arte

En este capítulo se describirán las herramientas y tecnologías software utilizadas para el desarrollo del presente trabajo de fin de grado.

2.1. Actividades y tareas

Como primer paso para la elaboración del trabajo se han definido una serie de actividades y tareas a desarrollar para conseguir como resultado final la aplicación plenamente operativa.

	Objetivo	Descripción
ACT1	Elección de la plataforma de desarrollo	Evaluación de las diferentes plataformas para el desarrollo de aplicaciones móviles con el fin de determinar cuál se ajusta mejor al propósito de la aplicación.
ACT2	Diseño de la aplicación	Determinar qué elementos son los que se quieren dar acceso al usuario y mostrarlos de manera sencilla y que resulte cómodo a la hora de interactuar
ACT3	Implementación de la aplicación	Implementación de la aplicación móvil que permita: - Visualizar información parlamentaria. - Base de datos local en el móvil. - Generación de alertas con información de interés.
ACT3	Memoria del trabajo	Elaboración de la memoria del trabajo donde se describan las tecnologías empleadas, la funcionalidad y la implementación de la aplicación y la bibliografía utilizada.

Tabla 2.1: Actividades del TFG

2.2. Plataformas de desarrollo

Como tarea previa al diseño de la aplicación móvil se ha realizado un estudio de las características de las diferentes plataformas de desarrollo en el ámbito de las aplicaciones móviles. En este apartado se desarrollarán algunos conceptos de cada una de ellas y se determinarán cuales son las ventajas e inconvenientes de cada vía. Finalmente se determinará cuál será el camino a seguir y se argumentarán los motivos por los que se ha regido la decisión final.

2.2.1. Aplicaciones web

Figura 2.1: Navegadores

Las aplicaciones web se desarrollan en lenguajes como: HTML, JavaScript, etc. Son ejecutadas en el navegador del dispositivo móvil y por este motivo pueden ser utilizadas en cualquier plataforma, el requisito que se ha de tener en cuenta es que sea compatible con el navegador del teléfono. Por la misma razón, tampoco es necesaria su instalación en el dispositivo.

Un aspecto interesante a tener en cuenta es que no es necesario que el terminal disponga de grandes prestaciones para poder ejecutar dichas aplicaciones.

Sin embargo, su funcionamiento basado en recursos web hace necesario que haya buena conectividad y que los servidores soporten que multitud de usuarios accedan a los datos de manera simultánea, sin estos dos factores este tipo de aplicaciones no rinden de manera óptima.

Además algunas de estas aplicaciones no funcionan sin conexión, lo que puede resultar incómodo para el usuario.

2.2.2. Aplicaciones nativas

Figura 2.2: Sistemas Operativos Nativos

Las aplicaciones nativas son aquellas que se desarrollan en un lenguaje específico para cada sistema operativo. Por ejemplo:

- Java y XML en Android
- Objective C y Swift en IOS
- Visual Basic en Windows

Esta característica nos permite poder acceder a las APIs disponibles para el sistema operativo en cuestión y otras características del hardware.

Gracias a los SDK que nos proporcionan podemos seguir las guías de diseño de la plataforma, como por ejemplo Material Design en Android, y que resulte una experiencia más cómoda para el usuario.

Además, estas aplicaciones, al ejecutarse a un nivel más bajo que, por ejemplo, las aplicaciones web, pueden lograr un mayor rendimiento. De esta manera, la aplicación puede ofrecer una mayor fluidez y por consiguiente una mejor experiencia del usuario.

Por otro lado hay que tener en cuenta que el código desarrollado para una plataforma no es reutilizable para otras plataformas por el hecho de que cada sistema operativo tiene su propio lenguaje de programación, con lo cual, habría que desarrollar una aplicación para cada una de ellas. Este factor influye en el coste de desarrollo de la aplicación, tanto en medidas económicas como de tiempo que supone el diseño y desarrollo de la misma.

2.2.3. Aplicaciones híbridas

Una aplicación híbrida es una combinación de los dos tipos desarrollados en los puntos anteriores. Este tipo de aplicaciones reúne tanto las ventajas de las aplicaciones nativas y como las aplicaciones web.

Por un lado, las aplicaciones híbridas son desarrolladas con lenguajes web, de esta manera y a diferencia de las nativas, pueden ser utilizadas en distintas plataformas. Además, permiten acceder a las características hardware del dispositivo móvil, al contrario que las aplicaciones web. Finalmente, este tipo de aplicaciones se pueden distribuir en la app store.

2.2.4. Comparativa y elección final

Dados los requisitos, que ha de cumplir la aplicación, siguientes:

- La aplicación ha de poder ser utilizada en distintos sistemas operativos.
- No basta con que sea una aplicación web, dado que el parlamento ya tiene una operativa.

Y teniendo en cuenta que, desarrollar nativamente una aplicación para cada uno de los sistemas operativos supondría un esfuerzo, de tiempo, mucho mayor de lo que el trabajo requiere, finalmente se ha optado por seguir un desarrollo híbrido.

De esta manera el factor 'costo de desarrollo' se reduce y los requisitos establecidos que la aplicación debe cumplir se verían cumplidos.

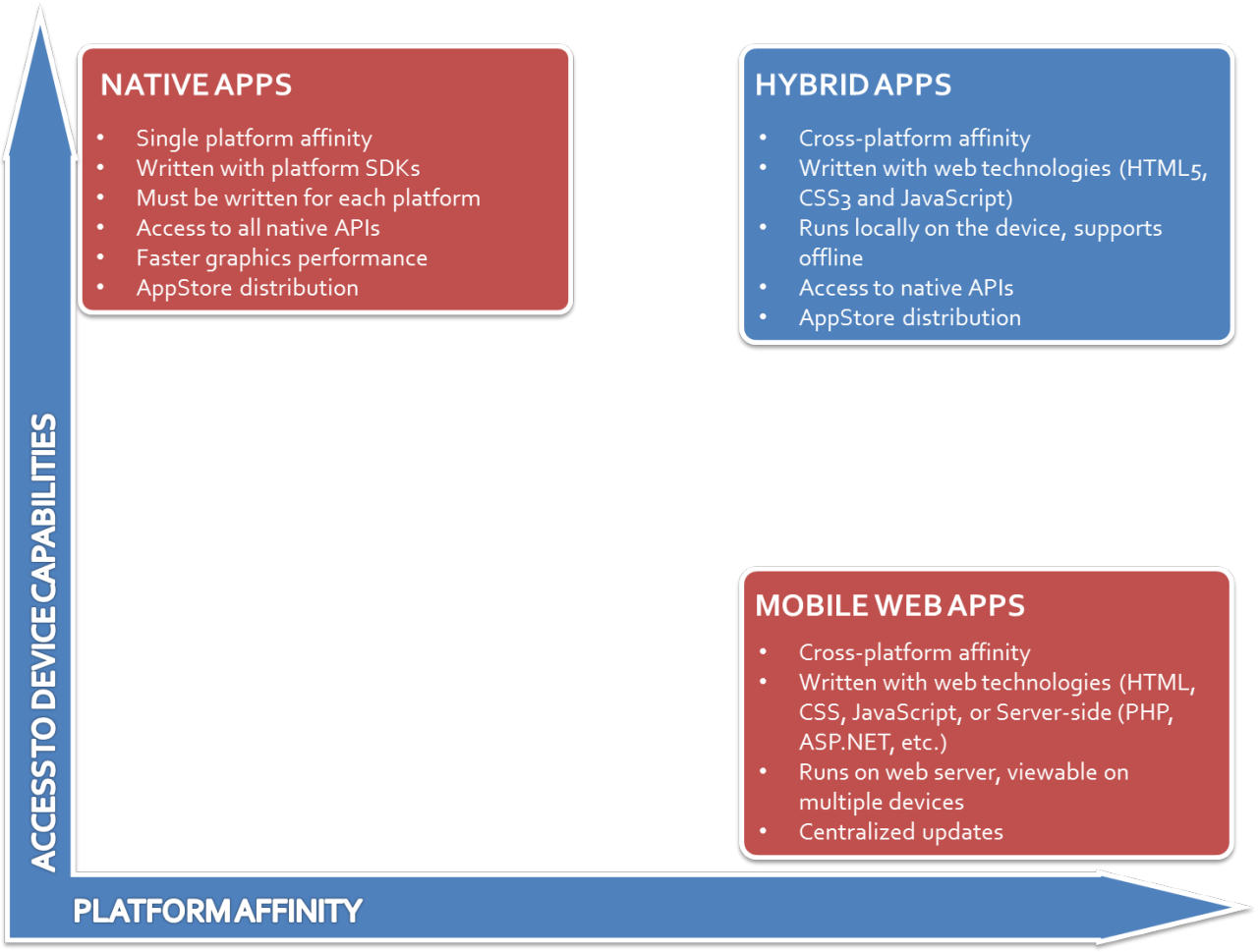


Figura 2.3: Comparativa plataformas de desarrollo [2]

2.3. Herramientas y tecnologías software

2.3.1. PhoneGapp

PhoneGap [3] es un framework para el desarrollo de aplicaciones móviles producido por Nitobi, y comprado posteriormente por Adobe Systems. Phonegap permite desarrollar aplicaciones móviles utilizando herramientas web tales como JavaScript, HTML5 y CSS3.

Las aplicaciones desarrolladas son híbridas, es decir, no son realmente aplicaciones nativas al dispositivo, ya que el renderizado se realiza mediante vistas web y no con interfaces gráficas específicas de cada sistema, pero no se trata tampoco de aplicaciones web, dado que se despliegan en el dispositivo y hacen uso de la API del sistema operativo nativo.

2.3.2. Cordova

Apache Cordova [4] es la versión de código abierto de Phonegap originalmente creado por Nitobi. Adobe compró Nitobi en 2011, le cambió el nombre a PhoneGap. Apache Cordova permite encapsular CSS, HTML, y código de Javascript dependiendo de la plataforma del dispositivo. Mejora las características de HTML y Javascript para trabajar con dispositivos móviles. Las aplicaciones que se crean con este framework son híbridas.

2.3.3. HTML5

HTML5 [5] (HyperText Markup Language, versión 5) es un lenguaje de marcas de hipertexto usado para estructurar y presentar el contenido para la web. Define una estructura básica y un código para la definición de contenido de una página web, así como: texto, imágenes, vídeos, etc.

Es la quinta revisión del estándar creado en 1990 y hace algunos años la W3C [6] la recomendó para transformarse en el estándar a ser usado en sustitución de HTML4.

```
1 <html>
2 <head>
3     <meta charset="UTF-8"/>
4     <title>Titulo de la pagina</title>
5 </head>
6 <body>
7     Contenido de la pagina
8 </body>
9 </html>
```

Listing 2.1: Ejemplo de código HTML5

2.3.4. Javascript

JavaScript [7] es un lenguaje de programación orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en el lado del cliente de una web, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Con Javascript podemos crear efectos en las páginas y hacer que el usuario pueda interactuar con ellas.

```
1 function holaMundo() {
2   var mensaje = "Hola Mundo!";
3   console.log(mensaje);
4 }
```

Listing 2.2: Ejemplo de código JavaScript

2.3.5. CSS3

Cascading Style Sheets (CSS, versión 3) [8], en español, hojas de estilo en cascada, es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado como por ejemplo HTML5 [5].

CSS está diseñado principalmente para separar el contenido del documento y la manera en la que se presenta este. Con esto se busca:

- Mejorar la accesibilidad del documento.
- Permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos.
- Reducir la complejidad y la repetición de código en la estructura del documento.

```
1  body {
2      background-color: lightblue;
3  }
4
5  h1 {
6      color: white;
7      text-align: center;
8  }
9
10 p {
11     font-family: verdana;
12     font-size: 20px;
13 }
```

Listing 2.3: Ejemplo de código CSS

2.3.6. JQuery

jQuery [9] es una biblioteca multiplataforma de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML.

Entre las características más importantes de jQuery podemos ver:

- Selección de elementos DOM.
- Eventos.
- Manipulación de la hoja de estilos CSS.

- Desarrollo de efectos y animaciones.
- Interacción con AJAX.

```
1 $(document).ready(function(){
2     $("p").click(function(){
3         $(this).hide();
4     });
5 });
```

Listing 2.4: Ejemplo de función jQuery

2.3.7. JQuery Mobile

JQuery Mobile [10] es un Framework optimizado para dispositivos táctiles. Está centrado en ser compatible con la gran variedad de smartphones y tablets que existen actualmente. jQuery Mobile es compatible con otros frameworks móviles y plataformas como PhoneGap [3], usada en este proyecto.

Para que la funcionalidad sea correcta al usarlo en un proyecto, es importante incluir también la librería JavaScript de jQuery, así como sus hojas de estilo. Tanto las librerías como las hojas de estilo están disponibles para ser descargadas o ser enlazadas desde el Content Delivery Network (CDN) de jQuery.

2.3.8. Firebase

Firebase [11] es una plataforma de desarrollo de aplicaciones web y móviles que ofrece distintos servicios. La compañía fue fundada en 2011 por Andrew Lee y James Tamplin hasta que en octubre de 2014 fue comprada por Google. Entre los servicios que ofrece podemos encontrar:

- Firebase Cloud Messaging: envío de mensajes y notificaciones multiplataforma.
- Firebase Auth: autenticación de usuarios.
- Firebase Storage: almacenamiento de imágenes, audio, video y otros contenidos.
- Firebase Hosting: soporte para alejar archivos estáticos así como CSS, HTML, JavaScript, etc.
- Firebase Crash Reporting: informes detallados de errores en la aplicación.

2.3.9. Node.js

Node.js [12] es un entorno de ejecución para JavaScript que usa un modelo de operaciones de entrada/salida sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. Node.js es un entorno orientado para la capa del servidor, pero no se limita solo a ello.

Node Package Manager, npm [13], es el ecosistema de paquetes de Node.js y es a su vez, el ecosistema más grande de librerías de código abierto en el mundo

2.3.10. PostgreSQL

PostgreSQL [14] es un sistema de código abierto para la gestión de bases de datos relacionales. Es compatible con distintos sistemas operativos así como Linux, UNIX y Windows. Incluye la mayoría de tipos de datos de SQL 2008 y permite el almacenamiento de imágenes, audio o video. En cuanto a las características que ofrece podemos destacar:

- Multi-Version Concurrency Control (MVCC).
- Replicación asíncrona.
- Transacciones anidadas.
- Recuperación de la base de datos.

Capítulo 3

Desarrollo

En este capítulo se expondrá como se ha ido desarrollando tanto la aplicación móvil como la parte del servidor. En un primer lugar, cómo tener configurado correctamente *Phonegap*, luego cómo se ha ido desarrollando el *HTML, CSS y JavaScript*, después cómo configurar *Firebase* para poderlo usar en nuestro servidor y por último las partes más relevantes del servidor, como por ejemplo: la manera en la que se generan las notificaciones push.

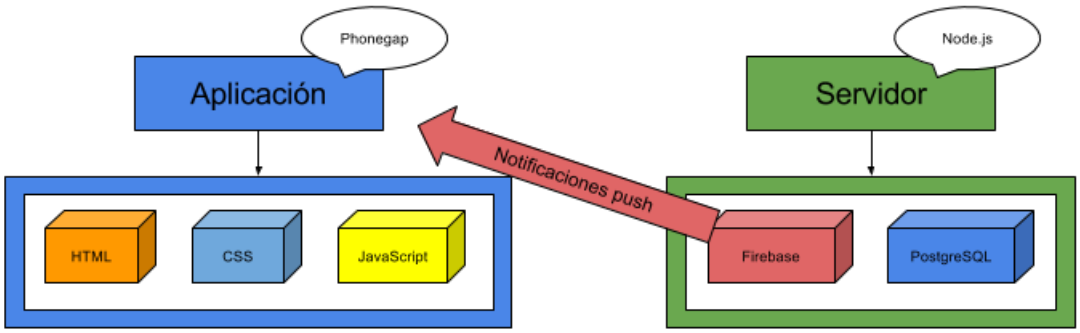


Figura 3.1: Vista general del desarrollo

3.1. Configuración de Phonegap

Para el desarrollo de la aplicación móvil se ha elegido *Phonegap*. En una primera instancia se explicarán los pasos que se han de seguir para tener la plataforma lista para empezar a desarrollar. Luego se irán mostrando y explicando en detalle las partes más relevantes del código.

Como paso inicial para empezar a desarrollar con Phonegap hay que instalar la herramienta. Existen dos vías:

Por un lado podemos instalar Phonegap CLI, interfaz de línea de comandos para crear aplicaciones Phonegap. Como requisitos para poder instalarla debemos tener disponible node.js y git. Una vez tenemos instaladas estas dos otras herramientas, solo debemos instalar Phonegap CLI a través de ***npm*** (*node package manager*) con el siguiente comando:

npm install -g phonegap

Por otra parte podemos optar por hacer uso de la aplicación *Phonegap Desktop*, simplemente hay que descargar el [15] instalador para el sistema operativo deseado y seguir los pasos del mismo.

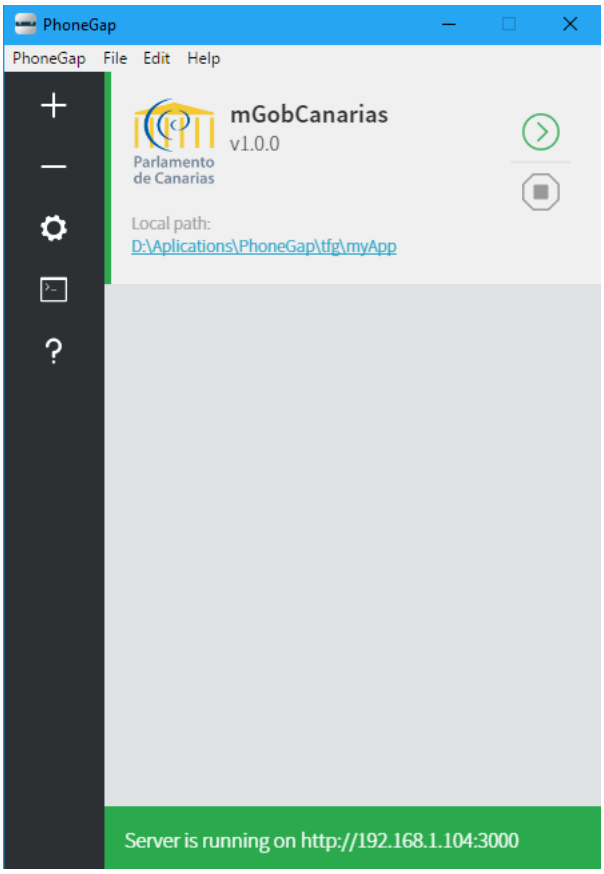


Figura 3.2: Phonegap Desktop application

Una vez tenemos instalada la herramienta, el siguiente paso a seguir es descargar en nuestro dispositivo móvil *Phonegap Developer*.

Ésta es una aplicación móvil que permite previsualizar y probar nuestra aplicación creada con Phonegap sin importar cual sea el sistema operativo y sin necesidad de configurar ningún SDK adicional.

Además nos permite tener acceso a las características nativas del dispositivo sin tener que compilar nuestra aplicación localmente.

Figura 3.3: Phonegap Developer

Una vez creada la aplicación podemos visualizarla tanto en nuestro navegador como en nuestro dispositivo móvil.

Phonegap Desktop inicia un pequeño servidor web para albergar nuestro proyecto devolviendo la dirección del servidor para que podamos visitarla en nuestro navegador o usarla en Phonegap Developer.

3.2. Desarrollo con Phonegap

El desarrollo de la aplicación consiste en crear y trabajar con los ficheros ***html, css y javascript*** dentro de la carpeta ***www*** del proyecto. En el fichero ***index.html*** haremos referencia por un lado, a nuestros ficheros de estilo (*css*) para darle aspecto a nuestra aplicación y por otro, a nuestros ficheros *javascript* para darle funcionalidad a la misma.

3.2.1. Desarrollando el HTML

En el proyecto, el desarrollo de *HTML* ha sido únicamente a través de la modificación del archivo *index.html*

Primeramente se han añadido las librerías *jQuery, jQuery Mobile* y *jQuery UI* (en esta última solo haciendo uso del *widget* **datepicker**) para darle un aspecto y funcionalidad inicial a la aplicación: barras de navegación, paginación, botones, etc.

Por otro lado se ha añadido tanto la librería *async.js* para trabajar con peticiones asíncronas, como ficheros con funcionalidad específica para cada parte de la aplicación.

Además se ha añadido algún estilo personalizado (*style.css*) y se ha modificado el cuerpo del código *HTML* para crear un esqueleto de la aplicación en el cual existan distintas secciones: una para la agenda, otra para los diputados, otra para los órganos y una sección más de ajustes.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      ...
5      <link rel="stylesheet" href="jquery-ui-1.12.1/jquery-ui.
       css">
6      <link rel="stylesheet" type="text/css" href="css/index.css
         " />
7      <link rel="stylesheet" type="text/css" href="css/style.css
         " />
8      <link rel="stylesheet" type="text/css" href="css/jquery.
         mobile-1.4.5.css" />
9      <title>mGoverment</title>
10 </head>
11 <body>
12     <div data-role="page" id="agenda">
13       ...
14     </div>
15     <div data-role="page" id="diputados">
16       ...
17     </div>
18     <div data-role="page" id="organos">
19       ...
20     </div>
21     <div data-role="page" id="ajustes">
22       ...
23     </div>
24     ...
25     <script src="js/jquery-1.11.3.min.js"></script>
26     <script src="js/jquery.mobile-1.4.5.js"></script>
27     <script src="jquery-ui-1.12.1/jquery-ui.js"></script>
28     <script src="js/async.js"></script>
29     <script src="js/mensajes.js"></script>
30     <script src="js/ajustes.js"></script>
31     <script src="js/agenda.js"></script>
32     <script src="js/diputados.js"></script>
33     <script src="js/organos.js"></script>
34 </body>
35 </html>
```

Listing 3.1: Fichero index.html

La mayor parte de la aplicación se genera dinámicamente. La sección de ajustes es la única que es íntegra en *HTML* siendo los elementos más relevantes: los *popup* (para notificar al usuario si se ha realizado la operación correctamente o no) y un formulario compuesto de una pareja de *slider* y *select* por cada sección (agenda, diputados y órganos) que permiten al usuario personalizar las alertas que se generarán en el dispositivo y guardar dicha configuración.

El *slider* sirve para activar o desactivar el tipo de alerta en cuestión mientras que el *select* sirve para seleccionar la frecuencia con la que se generan alertas.

En los siguientes fragmentos de código se muestra una parte de la sección de ajustes:

```
<div data-role="main" class="ui-content" id="content_ajustes">
  ...
  <form id="guardar_ajustes" class="ui-body ui-body-a
   ui-corner-all" data-ajax="false">
    <label for="ajuste_agenda">Agenda:</label>
    <div class="ui-grid-a" id="ajuste_agenda">
      ...
    </div>
    <label for="ajuste_diputados">Diputados:</label>
    <div class="ui-grid-a" id="ajuste_diputados">
      ...
    </div>
    <label for="ajuste_organos">Organos:</label>
    <div class="ui-grid-a" id="ajuste_organos">
      <div class="ui-block-a">
        <select name="organos_alerta" id="organos_alerta"
           data-role="slider">
          <option value="false">Off</option>
          <option value="true">On</option>
        </select>
      </div>
      <div class="ui-block-b">
        <select name="organos_tiempo" id="organos_tiempo"
           data-native-menu="false">
          <option value="1">Hora</option>
          <option value="2">Dia</option>
          <option value="3">Semana</option>
        </select>
      </div>
    </div>
    ...
```

Listing 3.2: Campos del formulario

```
1     ...
2     <div class="ui-field-contain">
3         <input type="button" name="submit" id="submit" value="
              Guardar" class="ui-shadow ui-btn ui-corner-all"
              data-position-to="window">
4         <div data-role="popup" id="popupSuccess" data-theme="b
              ">
5           <p>Cambios guardados correctamente.</p>
6         </div>
7         <div data-role="popup" id="popupError" data-theme="b">
8           <p>Los cambios no se han podido guardar.</p>
9         </div>
10        <div data-role="popup" id="popupNotnew" data-theme="b"
              >
11          <p>No hay cambios que guardar.</p>
12        </div>
13    </div>
14  </form>
15  ...
16 </div>
```

Listing 3.3: Mensajes *popup*

3.2.2. Desarrollando el CSS

El desarrollo del estilo de la aplicación ha sido escaso dado que se ha hecho uso de las librerías *jQuery Mobile y Jquery UI*. Sin embargo, éstas librerías no tienen a penas disponibles elementos gráficos que sirvan para alertar al usuario en los cambios en la información que se presenta en la aplicación. En este hecho y en algunos pequeños ajustes visuales se ha centrado el desarrollo de la parte de *CSS*.

El fichero *style.css* contiene este estilo personalizado. El fragmento de código que se muestra a continuación contiene el código *CSS* comentado por partes para que refleje lo que hace cada una de ellas:

```
/*Para cambiar como se muestran las imagenes de los lugares de
    las reuniones*/
img.centrada {
    position: absolute;
    top: 0;
    bottom: 0;
    margin: auto;
    margin-left: 5px;
}
/*Para personalizar la presentacion de algunos textos*/
.wrap {
    white-space: normal !important;
}
td.dato {
  font-weight: bold;
}
/*Para personalizar las alertas*/
.alerta-collap {
  color: red;
}
.collap-yel >h1>a{
  background-color: #EEDE2A !important;
}
.collap-blue >h1>a{
  background-color: #3388cc !important;
  border-color: #3388cc !important;
  color: white !important;
}
/*Para cambiar el color de algunas cabeceras*/
.listViewHeader {
  background-color: #3388cc !important;
  color: white !important;
}
/*Para modificar como se muestran algunos iconos*/
.iconLeft {
  position: relative;
  vertical-align: middle;
  margin-right: 15px;
}
.inlineIconNoDisk {
    display: inline-block;
    position: relative;
    vertical-align: middle;
    margin-right: 6px;
}
.inlineIconNoDisk :after {
    background-color: transparent !important;
}
```

Listing 3.4: Mensajes *popup*

3.2.3. Desarrollando el JavaScript

La mayor parte del desarrollo de la aplicación móvil se centra en la implementación del *JavaScript*. Esto se debe a que la mayor parte del contenido que ofrece la aplicación es generado dinámicamente.

Para cada una de las secciones está asociado un fichero *JavaScript* que implementa toda su funcionalidad. En los casos de la agenda, diputados y órganos hay cierta funcionalidad similar:

- Se piden los datos a la *API* por medio de *AJAX* haciendo uso de *jQuery*, la *API* nos devuelve un *JSON* que hay que parsear.
- Se eliminan de la base de datos aquellos objetos (reuniones, diputados, órganos) que la *API* no nos devuelva.
- Se recorren todos los objetos que nos da la *API* y se comprueba si es nuevo o ya existe en la base de datos.
- En caso de que sea nuevo se inserta en la base de datos y se construye su código *HTML* con un estilo de alerta para que el usuario se percate de este hecho.
- De la misma manera, en caso de que haya sufrido alguna modificación en su información, se actualiza la base de datos y se construye su *HTML* con un estilo de alerta.
- En caso de que no haya sufrido modificaciones se construye su *HTML* con el estilo básico.

En los siguientes fragmentos de código del *JavaScript* de la sección de la agenda podemos ver cómo se ha desarrollado estos conceptos.

En el primero podemos ver como se llama a la función **getJSON** de *jQuery* al cual se le proporciona la **url** de la *API* correspondiente a la agenda para obtener los datos de la misma.

Si nos devuelve reuniones se comprueban, de entre las que están en la base de datos, cuáles no se corresponden a las de la *API* para, en este caso, borrarlas.

```
1  $.getJSON("http://www.parcan.es/api/agenda/entrefechas.py",
       function(result){
2    var data = result.result;
3    if(data.lenght > 0){ //Si la API nos devuelve reuniones
4      db.transaction(function (tx) {
5        tx.executeSql("SELECT * FROM AGENDA", [], function (tx,
             result) {
6          var len = result.rows.length;
7          var found = false;
8          //Recorrer reuniones y comparar con la base de datos
9          for(var r = 0; r < len; r++){
10          for(var i = 0; i < data.length; i++){
11           if(result.rows.item(r).id_jornada == data[i].
                id_jornada){
12            found = true;
13            break;
14           }
15          }// Si no esta en la base de datos se elimina
16          if(found == false){
17           tx.executeSql("DELETE FROM AGENDA WHERE id_jornada =
                 ?",
18             [result.rows.item(r).id_jornada], deleteSuccess,
                  deleteError);
19          } else {
20           found = false;
21          }
22          ... //Comprobar actualizaciones
23         }
24       }, function (error) {
25         console.log(error);
26       });
27     }, selectError, selectSuccess);
28   }else{ //No hay reuniones en la API
29     ...
30   }
31 });
```

Listing 3.5: Borrar reuniones de la base de datos

En este otro fragmento se comprueba primero si la reunión está en la base de datos. En caso afirmativo se revisa si la información ha cambiado, si esto ocurre se construye el HTML con un estilo de alerta para que el usuario detecte qué información ha sido modificada. Si algún dato se ha modificado, se procede a actualizar la base de datos.

Si resulta que la reunión no se encuentra en la base de datos, se inserta en ésta y también se construye el HTML con un estilo de alerta para que el usuario sepa que es una reunión nueva.

```
for(var i = 0; i < data.length; i++){
 var id = data[i].id_jornada;
 var nombre_sala = data[i].nombre_sala;
 ...
 for(var r = 0; r < len; r++){
  if(result.rows.item(r).id_jornada == data[i].id_jornada){
   found = true;
   var update = false;
   var content = '<li id="'+ data[i].id_jornada +'" onclick="
      removeAlerta('+ data[i].id_jornada +')"><a href="#">';
   ...
   if(result.rows.item(r).nombre_sala != data[i].nombre_sala){
    update = true;
    content += '<div class="alerta">Nueva!</div><h2>'+ data[i
       ].nombre_sala +'</h2>';
   }else{
    content += '<h2>'+ data[i].nombre_sala +'</h2>';
   }
   ...
   if(update == true){
    tx.executeSql("UPDATE AGENDA SET ruta_imagen = ?,
       nombre_sala = ?, fecha = ?, hora = ?, descripcion = ?
       WHERE id_jornada = ?",
         [ruta_imagen, nombre_sala, fecha, hora, descripcion,
            id], updateSuccess, updateError);
   }
   ...
  }
 }
 if(found == false){
  tx.executeSql("INSERT INTO AGENDA (id_jornada, ruta_imagen,
     nombre_sala, fecha, hora, descripcion) VALUES
     (?,?,?,?,?,?)",
     [id, ruta_imagen, nombre_sala, fecha, hora, descripcion],
        insertSuccess, insertError);
  var content = '<li id="'+ data[i].id_jornada +'" onclick="
     removeAlerta('+ data[i].id_jornada +')"><a href="#">';
  ...
  content += '<div class="alerta">Nueva!</div><h2>'+ data[i].
     nombre_sala +'</h2>';
  ...
 }
 ...
}
```

Listing 3.6: Comprobar actualizaciones

En el caso de las secciones de diputados y órganos se sigue la misma dinámica. Lo único que cambian son las URLs de la peticiones, los datos que hay que parsear y el estilo que se le aplica al HTML que se forma a partir de esos datos.

Ahora bien, hay partes del código que son un poco más específicas de cada

sección.

En el caso de la agenda, se le permite al usuario que pueda realizar una consulta en un rango de fechas y para un órgano dado (de manera opcional).

En el siguiente fragmento de código podemos ver que se ha implementado una función que se ejecutará cuando se haga click en el botón con identificador *submit_agenda* la cual: obtiene los valores de los campos de las fechas y del campo del listado de órganos, si éstos están correctos lanza una petición a la URL correspondiente y construye el código HTML con los datos obtenidos.

Además se le notifica al usuario si todo ha ido correctamente, si no hay reuniones, si no se ha seleccionado alguna fecha para la consulta o si ha habido algún otro error y la consulta no se ha podido realizar.

```
1  $("#submit_agenda").bind( "click", function(event, ui) {
2      var desde = $("#desde").val();
3      var hasta = $("#hasta").val();
4      var organo = $("#listado_organos").val();
5      if(desde != '' && hasta != ''){
6        $.getJSON('http://www.parcan.es/api/agenda/entrefechas.
           py?desde='+desde+'&hasta='+hasta+'&organo='+organo,
           function(result){
7          var data = result.result;
8          if(data.length > 0){
9            ... //Se construye el HTML
10           $("#consultarAgendaSuccess").popup("open");
11         }else{
12           $("#noReuniones").popup("open");
13         }
14       }).fail(function(){
15         $("#consultarAgendaError").popup("open");
16       });
17     }else{
18       $("#faltanDatos").popup("open");
19     }
20 });
```

Listing 3.7: Consulta de la agenda

En lo que a la sección de diputados se refiere, se ha implementado una función que se ejecutará cuando el usuario haga click en el botón **Seguir/Dejar de seguir** disponible para cada diputado y que permite que se generen notificaciones concretas del mismo.

Para ello, dado el identificador del diputado, se hace una consulta a la base de datos para saber en que estado se encuentra éste (seguido o no). Según un caso u otro, se realiza un *post* con el *token* del dispositivo y el identificador del diputado a una *url* de un servidor propio, el cual hará que se generen notificaciones

específicas para ese diputado. Una vez la consulta es satisfactoria, se actualiza la base de datos en función del estado (seguido o no) que corresponda del diputado en cuestión.

En el siguiente fragmento de código podemos encontrar la implementación de la función descrita:

```
function seguirDiputado(id_miembro) {
  ...
  db.transaction(function (tx) {
    tx.executeSql("SELECT * FROM DIPUTADOS WHERE id_miembro =
       ?", [id_miembro], function (tx, result) {
      if(result.rows.item(0).seguido == false+''){//seguir
         diputado
        $.post("http://192.168.1.104:8080/follow_diputado",
        {
          id_miembro: id_miembro,
          token: localStorage.getItem('token')
        },
        function(data, status){
          $("#s"+id_miembro).text('Dejar de seguir');
          db.transaction(function (tx) {
            tx.executeSql('UPDATE DIPUTADOS SET seguido = ?
               WHERE id_miembro = ?',
                    [true, id_miembro], updateSuccess,
                       updateError);
          });
          $( "#popupSuccess" ).popup( "open" );
        }).fail(function(){
          $("#popupError").popup("open");
        });
      }else if(result.rows.item(0).seguido == true+''){//dejar
          de seguir
        ... //el post se hace a http://192.168.1.104:8080/
           unfollow_diputado
      }
    }, function (error) {
        console.log(error);
    });
  }, selectError, selectSuccess);
}
```

Listing 3.8: Consulta de la agenda

Por último, la sección de ajustes contiene dos métodos: una para fijar los valores de *slider* y *select* y otra para guardar los ajustes.

En la primera función se hace uso de la librería *jQuery Mobile* que nos permite el manejo de eventos del dispositivo móvil, en este caso ejecutaremos el código en el estado *pagebeforeshow* (ántes de que la página se muestre) y lo que haremos será, por medio de *local storage*, obtener los valores actuales de los *slider* y *select*, que corresponden a las notificaciones de agenda, diputados y órganos, para mostrar correctamente dichos elementos.

En el siguiente fragmento se muestra lo que se ha hecho:

```
$(document).on("pagebeforeshow","#ajustes",function(){
  if(localStorage.getItem('agenda_alerta') == 'true'){
    $("#agenda_alerta").val('true').slider('refresh');
  }else{
    $("#agenda_alerta").val('false').slider('refresh');
  }
  switch(localStorage.getItem('agenda_tiempo')) {
  case "1":
    $("#agenda_tiempo").val('1').selectmenu('refresh');
    break;
  case "2":
    $("#agenda_tiempo").val('2').selectmenu('refresh');
    break;
  case "3":
    $("#agenda_tiempo").val('3').selectmenu('refresh');
    break;
  default:
    console.log('error');
  }
  ... //Se repite para diputados y organos
});
```

Listing 3.9: Consulta de la agenda

La segunda función es muy parecida a la función para consultar fechas descrita en la sección de agenda. Al hacer clic en el botón de guardar: se obtienen los valores de los campos *slider* y *select* de cada sección, se comprueba si hay cambios con respecto a lo que hay en *local storage* y en caso afirmativo se realiza una petición *post* con el *token* del dispositivo y dichos datos a la url de **ajustes** del servidor para que los use en la generación de notificaciones.

Si los ajustes que el usuario quiere enviar no difieren de los ya guardados, no se realiza ninguna petición y se le lanza al usuario un mensaje comunicando este hecho. Por otro lado, si la operación se realiza correctamente, se actualizan los valores que hay en *local storage* y se le notifica al usuario de que los cambios han sido guardados correctamente. Por último, en caso de que no se hayan podido guardar los ajustes también se le notifica al usuario.

La función que realiza lo descrito es la siguiente:

```
$("#submit").bind( "click", function(event, ui) {
    var aa = $("#agenda_alerta").val();
    var at = $("#agenda_tiempo").val();
    var da = $("#diputados_alerta").val();
    var dt = $("#diputados_tiempo").val();
    var oa = $("#organos_alerta").val();
    var ot = $("#organos_tiempo").val();

    if(aa != localStorage.getItem('agenda_alerta') || at != localStorage.getItem('agenda_tiempo') ||
       da != localStorage.getItem('diputados_alerta') || dt != localStorage.getItem('diputados_tiempo' ||
       oa != localStorage.getItem('organos_alerta') || ot != localStorage.getItem('organos_tiempo'))){
      $.post("http://192.168.1.104:8080/ajustes",
      {
        token: localStorage.getItem('token'), agenda_alerta: aa, agenda_tiempo: at,
        diputados_alerta: da, diputados_tiempo: dt, organos_alerta: oa, organos_tiempo: ot
      },
      function(data, status){
        localStorage.setItem('agenda_alerta', aa);
        localStorage.setItem('agenda_tiempo', at);
        localStorage.setItem('diputados_alerta', da);
        localStorage.setItem('diputados_tiempo', dt);
        localStorage.setItem('organos_alerta', oa);
        localStorage.setItem('organos_tiempo', ot);
        $( "#popupSuccess" ).popup( "open" );
      }).fail(function(){
        $("#popupError").popup("open");
      });
    }else{
      $("#popupNotnew").popup("open");
    }
});
```

Listing 3.10: Consulta de la agenda

3.3. Generación de alertas: notificaciones Push

Las notificaciones push son mensajes enviados desde un servidor o aplicación a una aplicación de escritorio o aplicación móvil. Las notificaciones push ayudan a que el usuario tenga acceso rápido y fácil a determinada información. Un claro ejemplo es *WhatsApp*, los avisos que nos aparecen de los mensajes que nos mandan nuestros contactos son notificaciones push.

3.3.1. Configuración de Firebase

Utilizaremos el servicio de *Firebase Cloud Mesaging* [16], que nos permitirá el envío de notificaciones a nuestra aplicación móvil. Además, es necesario montar un servidor de aplicaciones que se encargue de gestionar el envío de dichas notificaciones a nuestra aplicación cliente. Los conceptos utilizados para el desarrollo y configuración del servidor serán expuestos en el siguiente apartado.

Para empezar a usar los servicios que nos ofrece *Firebase* es necesario crear un proyecto en la **Consola de Firebase** [17], en la cual se nos presenta la opción de añadir un proyecto nuevo asignándole un nombre y un país.

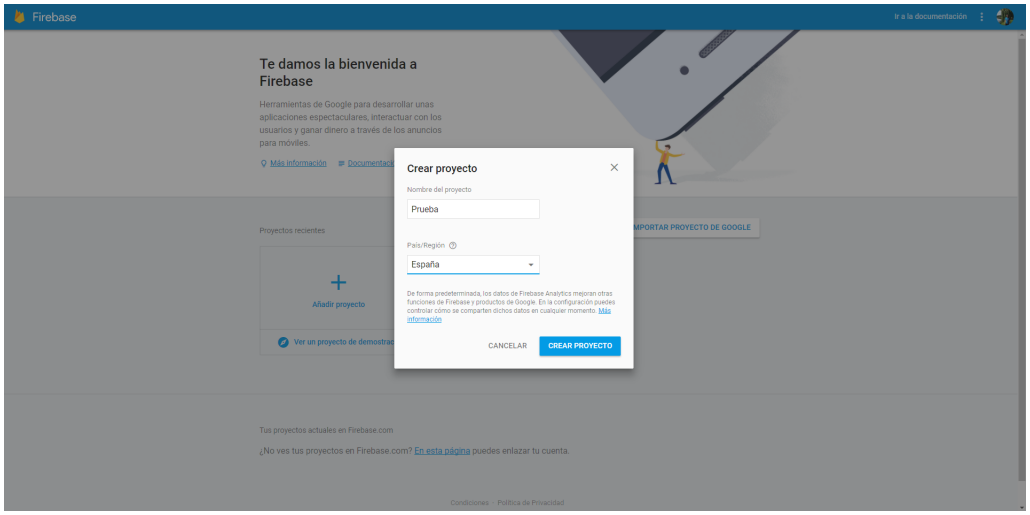


Figura 3.4: Crear un proyecto en Firebase

Una vez creado el proyecto, el siguiente paso será agregar *Firebase* a nuestra aplicación de *Phonegap*. Para ello tenemos que descargar un archivo de configuración para *Android* y otro para *IOS*.

Para esto hacemos click sobre la opción: Añade *Firebase* a tu aplicación de *Android/IOS* y seguimos los pasos de configuración. Se nos pedirá el nombre del paquete de Android y el ID del paquete de IOS.

Una vez realizado este paso se nos descargará un archivo google-services.json en el caso de Android y un archivo GoogleService-Info.plist en el caso de IOS.

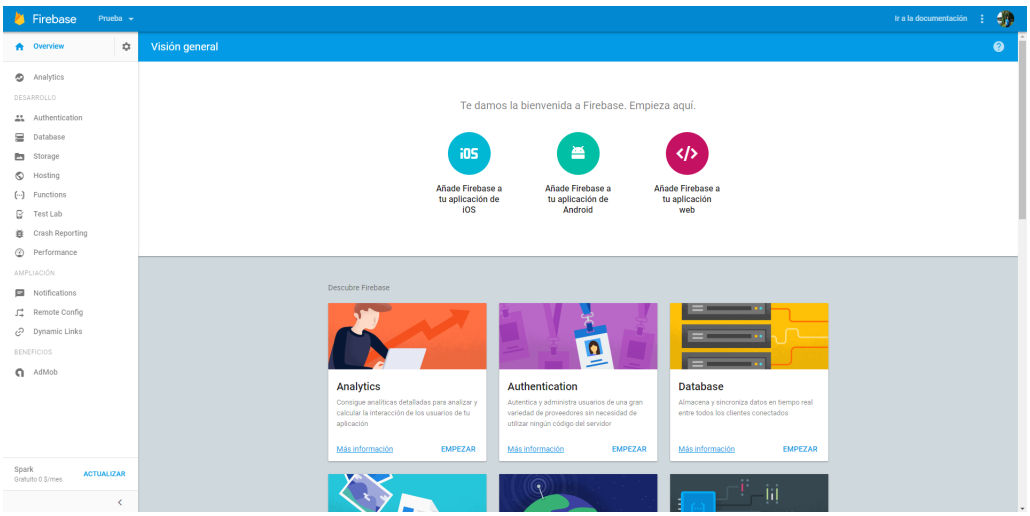


Figura 3.5: Agregar Firebase a la aplicación

Cuando obtengamos estos archivos de configuración, tendremos que añadirlos al directorio raíz de nuestra aplicación Phonegap.

Ahora toca agregar Firebase a nuestro servidor. Al lado del apartado ***Overview*** hay una pestaña de *settings*, hacemos click en ella y seleccionamos la opción permisos, que nos llevará a una página de administración. Nos dirigimos al apartado cuentas de servicio y creamos una nueva cuenta de servicio. Le damos un nombre y seleccionamos la opción: suministrar una nueva clave privada, dejando el tipo de clave como *JSON*. El resto lo dejamos como está y hacemos clic en Crear.

Una vez realizado este paso, se nos descargará un archivo *JSON* con las credenciales de la cuenta de servicio que hemos creado. Lo necesitaremos a la hora de configurar el servidor en el siguiente paso.

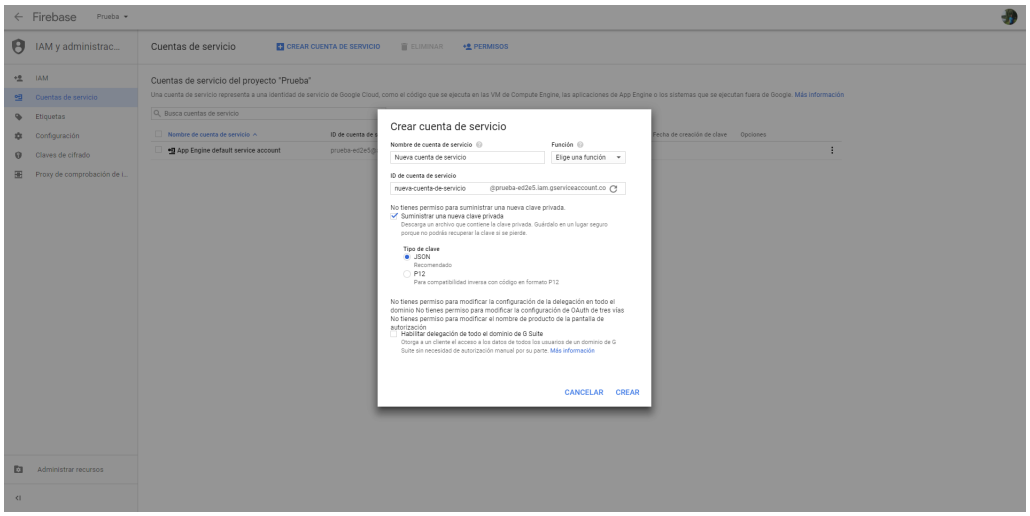


Figura 3.6: Agregar Firebase al servidor

3.3.2. Lado del servidor

Como primer obstáculo nos encontramos en cómo montar nuestro servidor de aplicaciones, tras investigar y recopilar información de la documentación de *Firebase* nos encontramos con que se puede usar *Node.js* [12] como servidor. Éste se encargará de gestionar las notificaciones push de nuestra aplicación.

Como primer paso para poder usar *Node.js* es necesario descargar e instalar la herramienta así como el gestor de paquetes de *Node.js* (npm). Una vez completado este paso, podemos empezar a usar la herramienta para iniciar el desarrollo nuestro servidor.

Mediante el gestor de paquetes (npm) instalamos *Express.js* [18], el cual nos proporciona una delgada capa de características de aplicación web básicas.

El siguiente paso será añadir al servidor: *Firebase Admin SDK* [19]. Ésto nos permite interactuar con *Firebase* de un modo privilegiado para poder realizar acciones tales como:

- Leer y escribir datos de la base de datos en tiempo real con privilegios de administrador.
- Enviar notificaciones push.
- Generar y verificar tokens de Firebase.

Para ello hay que instalar el paquete de *Firebase Admin* a través de node (npm install firebase-admin), requerir dicho paquete en nuestra aplicación *Node.js* e inicializarlo tal y como podemos apreciar en el siguiente fragmento de código:

```
1  var admin = require("firebase-admin");
2  var serviceAccount = require("D:/Aplications/Phonegap/tfg/
       nodeserver/parlamento-canarias-firebase-adminsdk-axlli
       -20589f3293.json");
3
4  var express = require('express');
5
6  //firebase
7  admin.initializeApp({
8    credential: admin.credential.cert(serviceAccount),
9    databaseURL: "https://parlamento-canarias.firebaseio.com/"
10 });
11
12 var app = express();
13
14 app.listen(8080, function () {
15   console.log('App listening on port 8080!')
16 })
```

Listing 3.11: Configuración inicial del servidor

El archivo al que se hace referencia en la variable *serviceAccount* es el archivo de configuración que nos descargamos desde la consola de *Firebase* cuando configuramos el proyecto. Además, cuando realizamos la inicialización tenemos que referenciar la dirección de la base de datos de nuestra aplicación que podemos encontrar también en la consola de *Firebase*, en el apartado *database* del panel de control.

Una vez realizados estos sencillos pasos, ya tendríamos configurado nuestro servidor para que pueda enviar notificaciones push. Ahora los siguientes pasos a seguir serían:

- Añadir una base de datos para guardar toda la información necesaria para la generación de notificaciones.
- Establecer una vía para que el dispositivo se pueda comunicar con el servidor y enviar los datos que hagan falta.
- Desarrollar las funciones que se encargarán de gestionar las notificaciones.
- Crear tareas programadas para que dichas funciones se ejecuten cada cierto tiempo.

Para la base de datos se ha elegido ***pg-promise*** [20], el cual es una interfaz para *PostgreSQL* que nos ofrece la funcionalidad necesaria para el uso que le queremos dar a la base de datos en nuestro servidor. Para incluirla en el proyecto solo debemos instalar el paquete vía *npm* con el comando:

npm install pg-promise

Una vez instalado el paquete, solo hace falta cargarlo e inicializarlo en nuestro servidor. Para ello, al fragmento anterior se le debe añadir el siguiente código:

```
...
//base de datos
var pgp = require("pg-promise")(/*options*/);
var cn = {
    host: 'localhost',
    port: 5432,
    database: 'postgres',
    user: 'postgres',
    password: 'password'
};
var db = pgp(cn);
...
app.listen(8080, function () {
  console.log('App listening on port 8080!')
})
```

Listing 3.12: Base de datos PostgreSQL

Además usaremos *pgAdmin* para crear las tablas necesarias así como:

- Una tabla para guardar la información de los dispositivos móviles: *token*, información referente a ajustes en las notificaciones.
- Una tabla para la información de la agenda, otra para diputados y otra para órganos.
- Una tabla para guardar la información referente a los miembros de un órgano.
- Y una tabla para guardar los diputados a los que sigue un usuario.

Una vez la base de datos está configurada es necesario establecer una vía por la cual la aplicación envíe, por un lado, su token único identificativo del dispositivo (para poder establecer el envío de notificaciones al mismo), y por otro lado, las preferencias acerca de la generación de notificaciones.

Para conseguir ésto se han creado varias *URLs*. La primera de ellas tiene la funcionalidad de recibir y guardar los *tokens* enviados por los dispositivos. Podemos ver cómo se inserta el mismo en la tabla *devices* de la base de datos de nuestro servidor, la cual contiene la información asociada a los dispositivos que usan nuestra aplicación.

```
1  ...
2  var db = pgp(cn);
3  ...
4  app.post('/token', function (req, res) {
5    db.none('INSERT INTO devices(token) VALUES($1)', [req.body.t
       ])
6      .then(() => {
7          console.log("Token insertado correctamente");
8      })
9      .catch(error => {
10         console.log("Error insertando token: ", error);
11     });
12   res.send('Token recibido');
13 })
14 ...
```

Listing 3.13: Almacenamiento de tokens

Por otro lado tenemos esta otra *URL* similar a la anterior, a la cual se le envían las preferencias acerca de las notificaciones(activada o no y la frecuencia en la que se enviará la misma). Dado el *token* del dispositivo, actualizamos también la tabla *devices* de la base de datos con dichos ajustes.

```
...
var db = pgp(cn);
...
app.post('/ajustes', function (req, res) {
  db.none('UPDATE devices SET agenda_alerta = $1,
     agenda_tiempo = $2, diputados_alerta = $3,
     diputados_tiempo = $4, organos_alerta = $5,
     organos_tiempo = $6 WHERE token = $7',
  [req.body.agenda_alerta, req.body.agenda_tiempo, req.body.
     diputados_alerta, req.body.diputados_tiempo, req.body.
     organos_alerta, req.body.organos_tiempo, req.body.token])
  .then(() => {
      res.send('Ajustes guardados');
  })
  .catch(error => {
      res.send('Error guardando ajustes');
  });
})
...
```

Listing 3.14: Actualización de ajustes de las notificaciones

Además necesitamos guardar en otra tabla las preferencias para los diputados en concreto, es decir, los casos en los que el usuario quiere alertas de un diputado específico.

La solución a ésto se puede abordar de diferentes maneras. En este caso se ha optado por crear dos rutas: una para seguir al diputado y otra para dejar de seguirlo. La tabla a la cual se realizan las consultas en ambos casos es **seguimiento_diputados** donde se asocia ***token*** y **diputado** como clave primaria.

A continuación podemos ver cómo están implementadas las *URLs* descritas:

```
...
app.post('/seguir_diputado', function (req, res) {
  db.none('INSERT INTO seguimiento_diputados(id_miembro, token
     ) VALUES($1, $2)', [req.body.id_miembro, req.body.token])
    .then(() => {
        res.send('Diputado seguido');
    })
    .catch(error => {
        res.send('Error siguiendo diputado');
    });
})
...
```

Listing 3.15: Ruta para seguir diputado

```
...
app.post('/dejar_de_seguir_diputado', function (req, res) {
  db.none('DELETE FROM seguimiento_diputados WHERE id_miembro
     = $1 AND token = $2',
  [req.body.id_miembro, req.body.token])
  .then(() => {
      res.send('Diputado dejado de seguir');
  })
  .catch(error => {
      res.send('Error al dejar de seguir diputado');
  });
})
...
```

Listing 3.16: Ruta para dejar de seguir a un diputado

Una vez todas las rutas destinadas al envío de ajustes del dispositivo están configuradas, debemos dar paso a la implementación de aquellas funciones que se encargarán de generar los mensajes que se enviarán a los dispositivos.

Dichos métodos tienen un carácter similar al descrito en la sección 3.2.3, donde se explicaba cómo se implementan las alertas visuales en la aplicación. Las diferencias son que ahora:

- La manera de realizar las peticiones a la *API* cambia: en el cliente se utiliza *jQuery* y ahora en el servidor se usa el paquete ***request***.
- Al ser distintas bases de datos, el código que construye las consultas es ligeramente distinto.
- No hace falta todo el código destinado a construir el *HTML* resultante dado que en el servidor no hace falta.

En el fragmento 3.17 podemos ver cómo se construye la petición con la librería *request* dentro del método destinado a generar las notificaciones de la agenda, se le debe pasar una *URL*, el tipo de datos que esperamos en la respuesta (en este caso *json*) y una *callback* donde trataremos los datos obtenidos.

Además podemos ver que a ésta se le pasa una variable **tiempo**, la cual representa la frecuencia de las notificaciones que más adelante explicaremos cómo la utilizamos.

```
1  ...
2  function actualizarAgenda(tiempo){
3    var url = "http://www.parcan.es/api/agenda/entrefechas.py";
4    request({
5      url: url,
6      json: true
7    }, function (error, response, body) {
8      ...
9    })
10 }
11 ...
```

Listing 3.17: Librería *request* para peticiones **http**

Como podemos apreciar en 3.18 y como ha aparecido en varias ocaciones, las consultas a la base de datos se crean a partir de la variable **db** (en rojo en el código) que hace referencia a la base de datos definida anteriormente.

Las opciones ***any*** y ***none*** configuran la consulta de manera que, de ésta, se esperen cualquier número de filas o que no se espere ningún dato devuelto respectivamente.

Para actualizar correctamente la base de datos debemos:

- Borrar las reuniones que estén en la base de datos y que no se encuentren en la *API*.
- Comprobar qué reuniones no se encuentran en la base de datos para insertarlas.
- Y en caso de que la reunión esté en la base de datos, comprobar si la información asociada a la misma ha cambiado para actualizar los datos correspondientes.

Además, a medida que se realizan las comprobaciones mencionadas, haremos uso de las variables ***updateMensaje*** y ***insertMensaje*** para en caso de que haya algún cambio en alguna reunión o hayan reuniones nuevas, actualicemos dichas variables para luego podamos notificar al usuario como se muestra en 3.19.

```
...
var data = body.result;
if (!error && response.statusCode === 200) {
 db.any('SELECT * FROM agenda')
  .then(function(result) {
   var len = result.length;
   var found = false;
   //borrar las reuniones que no estan en la api
   ...
   var updateMensaje = false;
   var insertMensaje = false;
   //insertar reuniones
   for(var i = 0; i < data.length; i++){
    for(var r = 0; r < len; r++){
     if(result[r].id_jornada == data[i].id_jornada){
      found = true;
      var update = false;
      if(result[r].ruta_imagen_sala != data[i].
         ruta_imagen_sala || result[r].nombre_sala != data[i].
         nombre_sala
       || result[r].descripcion != data[i].descripcion ||
          result[r].fecha != data[i].fecha || result[r].hora
          != data[i].hora){
        update = true;
      }
      if(update == true){
       updateMensaje = true;
       db.none('UPDATE agenda SET nombre_sala = $1, fecha = $2
          , ruta_imagen_sala = $3, descripcion = $4, hora = $5
           WHERE id_jornada = $6',
              [nombre_sala, fecha, ruta_imagen_sala,
                 descripcion, hora, id])
       ...
      }
      break;
     }
    }
    if(found == false){
     insertMensaje = true;
     //insertar en la base de datos
     ...
    }else{
     found = false;
    }
   }
   ...
 })
}
...
```

Listing 3.18: Actualizando agenda

Una vez realizadas las comprobaciones procedemos a enviar los mensajes a los dispositivos correspondientes. Hacemos una petición a la base de datos para obtener todos los *tokens* de la tabla *devices* que tienen las alertas de la agenda activadas (**agenda_alerta = true**) y que la frecuencia preferida corresponde al parámetro **tiempo**.

Si la variable *insertMensaje* o *updateMensaje* están a ***true*** se enviarán los mensajes correspondientes a que hay nuevas reuniones o que los datos de alguna han sido modificados, de otra manera no se enviará ninguna notificación a los dispositivos. En el siguiente fragmento podemos ver el código que acabamos de explicar:

```
1  ...
2  db.any('SELECT token FROM devices WHERE agenda_alerta = true
       AND agenda_tiempo = $1', tiempo)
3    .then(function(data) {
4      // success;
5      for(var i = 0; i < data.length; i++){
6        if(insertMensaje == true){
7          mensaje(data[i].token, 'Agenda', 'Hay nuevas reuniones
               en la agenda');
8        }
9        if(updateMensaje == true){
10         mensaje(data[i].token, 'Agenda', 'Los datos de alguna
              reunion han sido modificados');
11       }
12     }
13   })
14   .catch(function(error) {
15     // error;
16   });
17 ...
```

Listing 3.19: Envío de notificaciones

Para el caso de los diputados y de los órganos se sigue el mismo procedimiento que se ha expuesto para la agenda. Lo único que cambia son las *URLs* de la *API* a las cuales se les pide los datos, las tablas de la base de datos a las que se realizan las consultas y los textos de los mensajes que se envían a los dispositivos. Por este hecho y para no ser redundantes no se expondrá más código con respecto a cómo se generan las notificaciones.

Una vez tenemos la generación de alertas implementada, necesitamos que éstas se ejecuten cada cierto tiempo, es decir, que se ejecuten cada hora, cada día y cada semana, dado que son las opciones que le hemos permitido al usuario elegir.

Este problema lo hemos abordado haciendo uso de la librería de *Node.js*: **node-cron** [21], que nos permitirá programar tareas par que se ejecuten en intervalos regulares las funciones que hemos implementado.

Como podemos ver en 3.20 se hace uso de la función *schedule* la cual acepta como parámetros distintas configuraciones. Como podemos ver en la siguiente imagen:

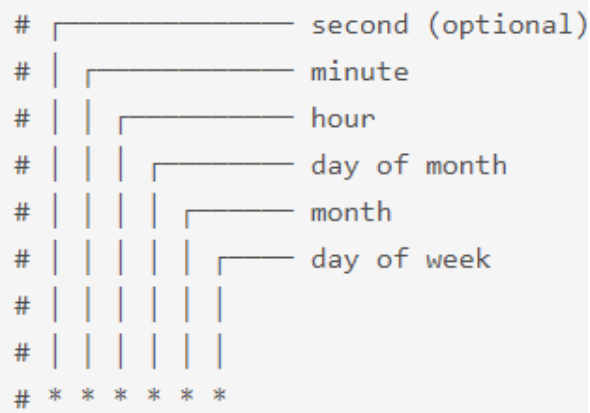


Figura 3.7: CSS3

Por lo tanto, usamos el segundo, tercer y cuarto campo para que las tareas se ejecuten cada hora, cada día y cada siete días respectivamente. Dentro de las tareas programadas llamamos a las funciones que se encargan de generar las alertas con su respectiva frecuencia de tiempo.

```
1  var cron = require('node-cron');
2  ...
3  //Tarea para cada hora
4  cron.schedule('* */1 * * *', function(){
5    actualizarAgenda(1);
6    actualizarDiputados(1);
7    actualizarOrganos(1);
8  });
9  //Tarea para cada dia
10 cron.schedule('* * */1 * *', function(){
11   actualizarAgenda(2);
12   actualizarDiputados(2);
13   actualizarOrganos(2);
14 });
15 //Tarea para cada semana
16 cron.schedule('* * * */7 *', function(){
17   actualizarAgenda(3);
18   actualizarDiputados(3);
19   actualizarOrganos(3);
20 });
```

Listing 3.20: *Cron* para la ejecución de tareas programadas

Para finalizar con la parte del servidor mostraremos la función encargada de configurar el mensaje que se envía al dispositivo 3.21.

En ella, se le pasa como parámetro un *token*, un título para el mensaje y un cuerpo, el cual será una breve explicación de en lo que la notificación consiste. Además se hace uso de *Firebase Admin*, expuesto con anterioridad en 3.11 que es lo que realmente nos permite enviar el mensaje al dispositivo.

```
1  var admin = require("firebase-admin");
2  var serviceAccount = require("D:/Aplications/Phonegap/tfg/
       nodeserver/parlamento-canarias-firebase-adminsdk-axlli
       -20589f3293.json");
3  ...
4  //firebase
5  admin.initializeApp({
6    credential: admin.credential.cert(serviceAccount),
7    databaseURL: "https://parlamento-canarias.firebaseio.com/"
8  });
9  ...
10 function mensaje (token, titulo, cuerpo){
11   var payload = {
12     "notification": {
13       "title": titulo,
14       "body": cuerpo
15     }
16   };
17
18   admin.messaging().sendToDevice(token, payload)
19     .then(function(response) {
20       console.log("Successfully sent message:", response.
           results);
21     })
22     .catch(function(error) {
23       console.log("Error sending message:", error);
24     });
25 }
```

Listing 3.21: Función que configura las notificaciones

Capítulo 4

Descripción de la aplicación

En este capítulo se muestra y describe la funcionalidad al completo de la aplicación móvil. Las pantallas que se mostrarán son de un dispositivo android aunque el estilo que tiene la aplicación en un dispositivo iOS no varía dado que el desarrollo es híbrido y esta es una de las características de este tipo de desarrollo.

4.1. Pantalla de carga de la aplicación

Como elemento inicial al abrir la aplicación se nos muestra una pantalla de carga o *splash screen* con un fondo blanco y el logo del Parlamento de Canarias. Esta característica gráfica nos es útil para hacer que el usuario se percate de que la aplicación está cargando. Una vez la misma haya cargado, el *splash screen* desaparecerá y se mostrará la página principal de la aplicación.

Figura 4.1: SplashScreen de la aplicación

4.2. Sección de la agenda

La sección de la agenda se toma como página principal de la aplicación.

En ésta se le muestra al usuario un apartado con las próximas reuniones que se celebrarán en un plazo de 3 meses. La información que aparece asociada a cada reunión consta de: la fecha, la hora, el lugar de la reunión (con una imágen asociada de la misma) y una breve descripción del tema que se tratará.

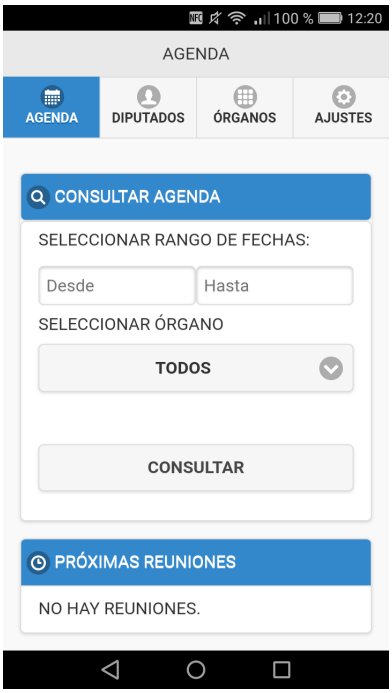


Figura 4.2: Sección de la agenda

Además se le permite al usuario que pueda consultar la agenda entre el rango de fechas que elija y, de manera opcional, de un órgano determinado.

Para elegir las fechas, se dispone al usuario de un calendario (o datepicker) desplegable al tocar los cuadros de texto del apartado **Seleccionar rango de fechas**. El usuario está obligado a elegir tanto una fecha inicial como una final para poder realizar la consulta de manera satisfactoria, de lo contrario se le notificará al usuario que debe proporcionar un rango de fechas para poder realizar la acción.

De manera opcional en la consulta, se le proporciona al usuario la posibilidad de seleccionar un órgano de manera que solo se muestren las reuniones asociadas al mismo. Al seleccionar el menú desplegable aparece una lista de los órganos existentes y además se da la posibilidad de buscar el órgano por medio de un filtro añadido.

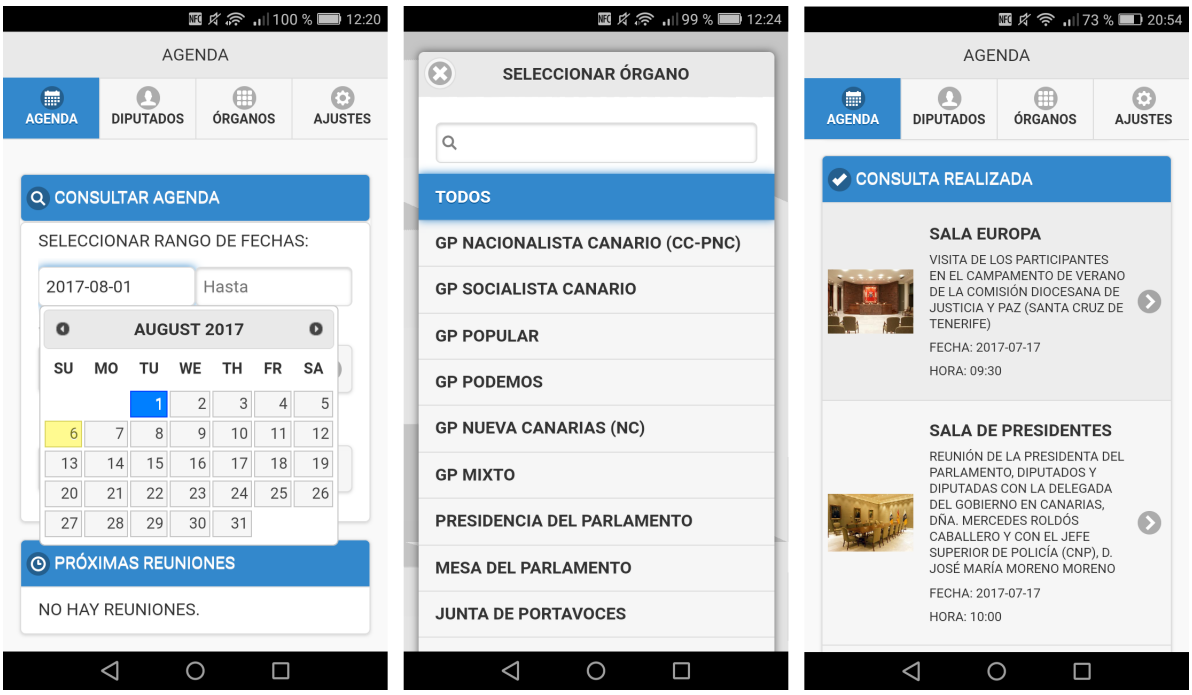


Figura 4.3: Consulta de la agenda

4.3. Sección de los diputados

En esta sección se muestra al usuario una lista de elementos desplegables con la información de los diputados del parlamento. Además la lista tiene un filtro añadido para que resulte más cómoda la búsqueda de un diputado en concreto.

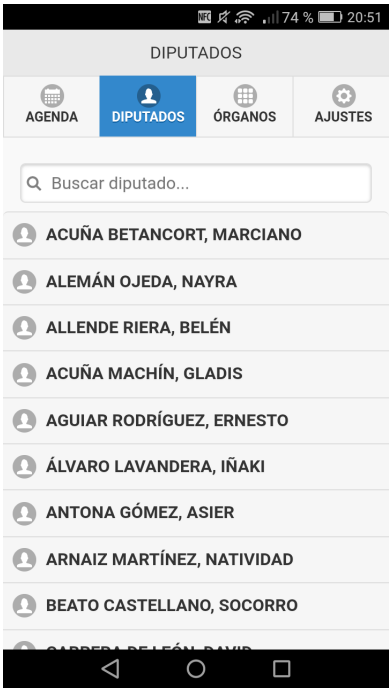


Figura 4.4: Sección de los diputados

Al seleccionar un diputado se expande el elemento plegado para mostrar información referente al mismo, así como: sexo, la fecha de alta en el cargo, la fecha de baja (si la hubiese), el grupo al que pertenece, si presenta declaración de bienes, etc.

También se encuentra otro desplegable con un resumen de la actividad profesional y la trayectoria política del diputado.

Por otro lado, antes de toda la información mencionada, hay situado un botón que sirve para seguir o dejar de seguir al diputado, en caso de que se quieran recibir o no alertas específicas referente al diputado en cuestión.

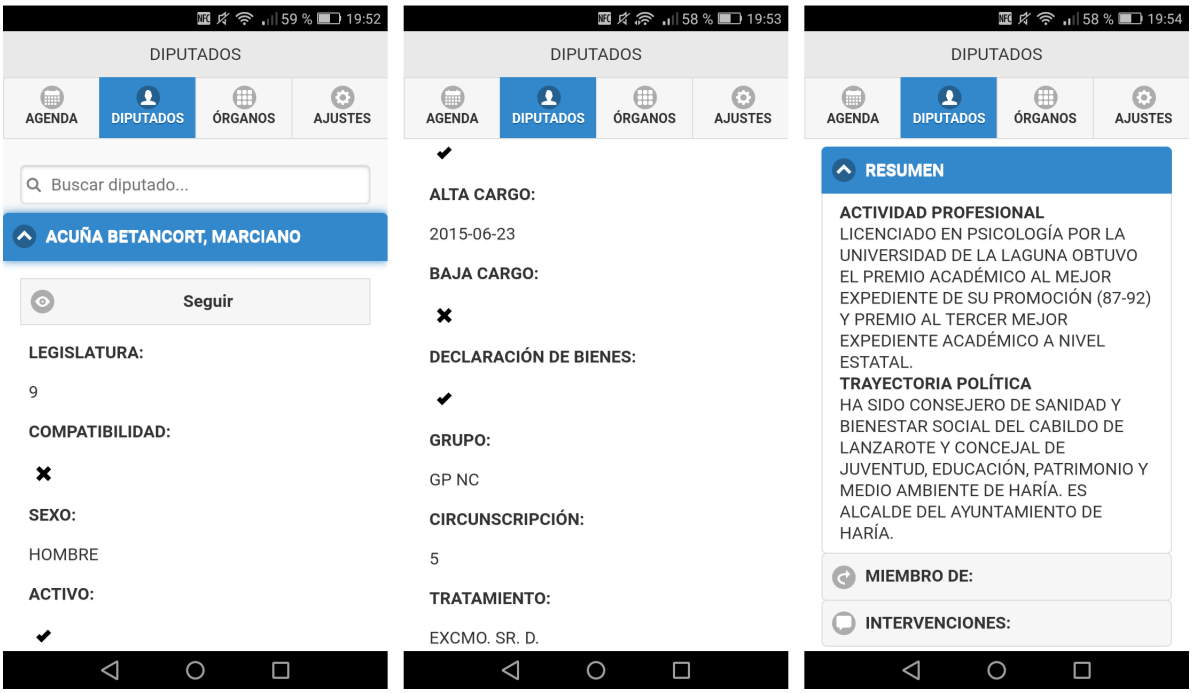


Figura 4.5: Detalle del diputado

Además, después de la sección de resumen, se encuentran otros dos desplegables, uno con la lista de órganos a los que pertenece y otra con la lista de intervenciones que ha realizado.

Al seleccionar un órgano se muestra, entre otras cosas, el cargo que desempeña y cuándo se le dió de alta en el mismo. Por otro lado, las intervenciones están ordenadas por año y la información que contiene es: la fecha, hora, el número de la intervención al que corresponde de entre todas las intervenciones realizadas por el diputado y una breve descripción acerca de a lo que la intervención se refirió.

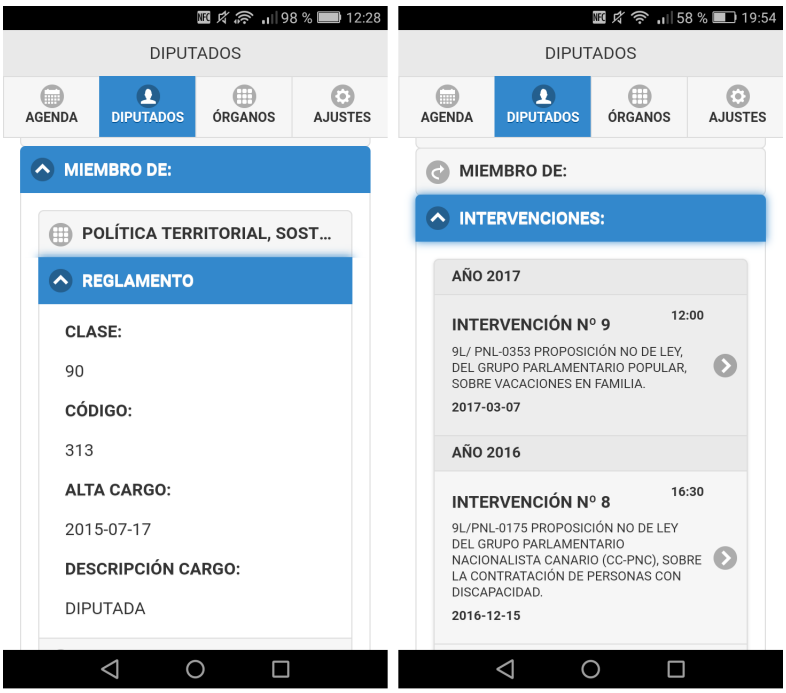


Figura 4.6: Membresía e intervenciones

4.4. Sección de los órganos

En esta sección, de manera similar a la sección de los diputados, se muestra al usuario una lista de desplegables con la información de los órganos del parlamento. Además la lista tiene un filtro añadido para que resulte más cómoda la búsqueda de un órgano en concreto. En la imagen de abajo se muestra como, al hacer uso del filtro, aparecen solo en la lista los ayuntamientos.

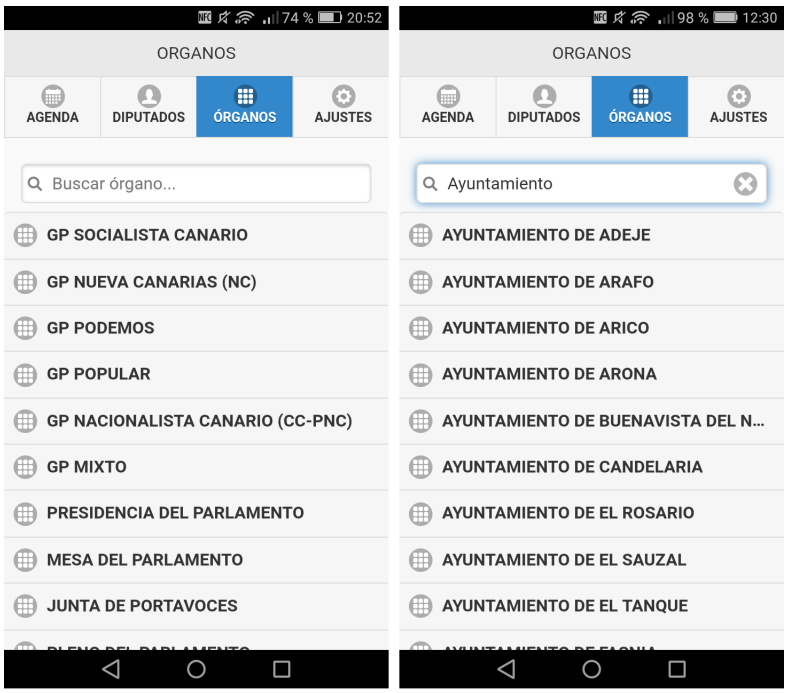


Figura 4.7: Sección de los órganos

Al expandir un órgano se muestra la información referente al mismo, entre otras cosas: el alias, la legislatura a la que pertenece, el tipo de comisión que tienen, etc. También, a modo de de desplegable, hay situada una sección de descripción en la cual se enumeran las múltiples funciones que desempeña el órgano en cuestión.

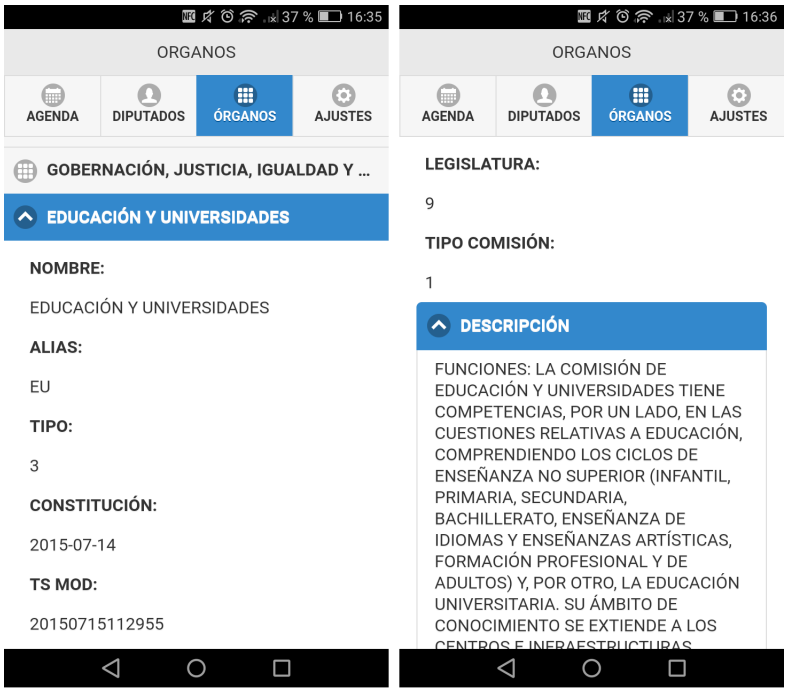


Figura 4.8: Información de los órganos

Además, muchos de los órganos están compuestos por una serie de diputados, en los casos que así sea y debajo de la sección de resumen, aparece otro desplegable que muestra la lista de dichos miembros con información acerca del mismo, como por ejemplo: el cargo que desempeña, la fecha en la que se dio de alta y el grupo al que pertenece.

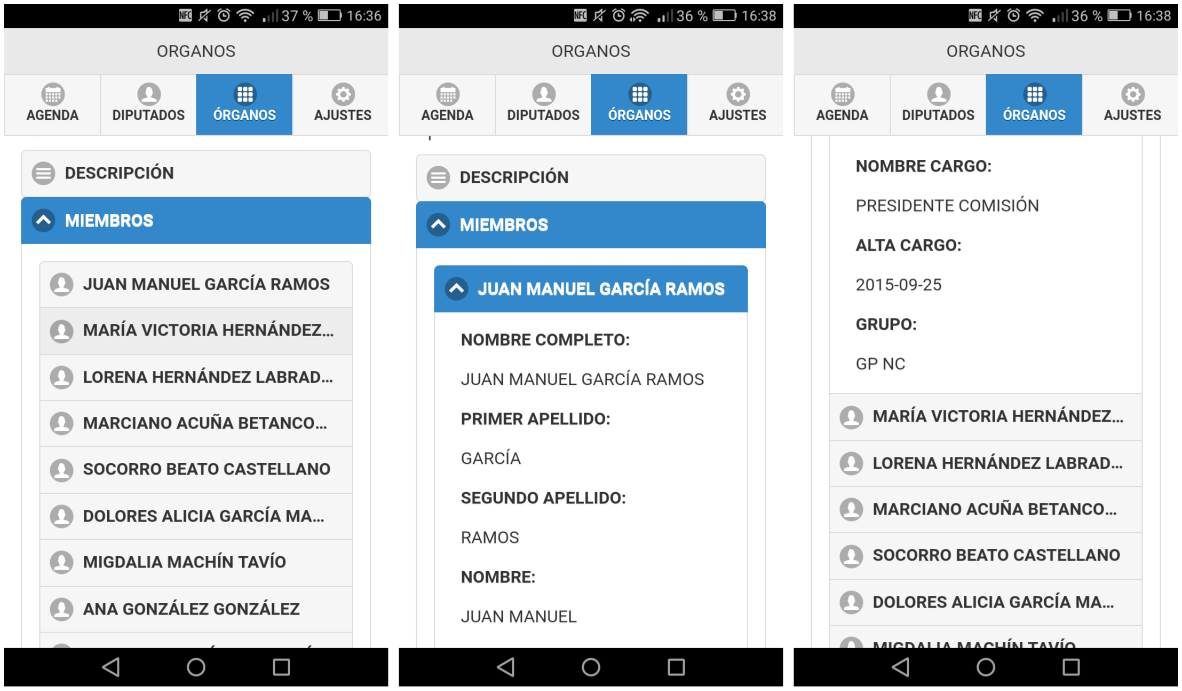


Figura 4.9: Miembros de un órgano

4.5. Ajustes

En esta pestaña, de momento, se encuentra un apartado para que el usuario tenga un cierto control sobre las alertas que se generan en su dispositivo. Éste puede elegir sobre qué información se generarán las alertas y cada cuanto tiempo se le notificarán dichas alertas en caso de que haya nueva información disponible.

El rango de tiempo sobre el que el usuario puede elegir para las notificaciones varía entre: cada hora, cada día y cada semana. Y la información sobre la que se generan las alertas son: las reuniones futuras de la agenda, la información referente a los diputados así como de las intervenciones que hagan éstos y la información referente a los órganos.

Un detalle a tener en cuenta es que, la parte de las notificaciones de diputados en esta sección es para todo el conjunto de los diputados, es decir, se generarán notificaciones sobre cualquier información nueva de cualquier diputado. Mientras que, por otro lado y como ya se ha mencionado anteriormente, el usuario puede seguir individualmente a un diputado en concreto haciendo uso del botón

'seguir' de éste. Con lo cual, si en la sección de alertas la parte de diputados está activada, se generarán notificaciones sobre todos los diputados mientras que, si está desactivada esta parte, se generarán notificaciones sobre aquellos diputados seguidos individualmente, si los hubiese.

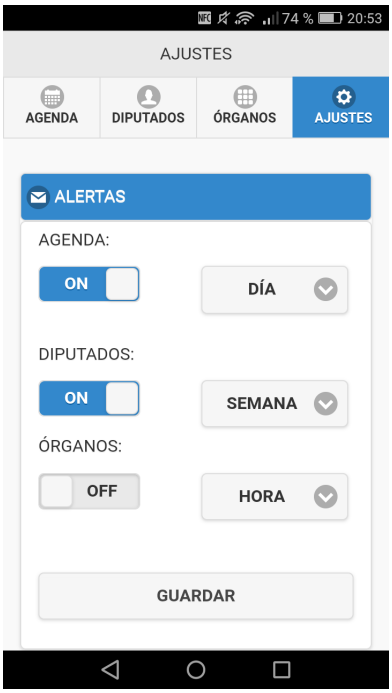


Figura 4.10: Sección de ajustes

4.6. Alertas

En esta parte se mostrarán tanto los tipos de notificaciones push que se envían al dispositivo según los cambios que se hayan producido en la información como la manera en la que se alerta al usuario de qué información ha sido modificada.

4.6.1. Notificaciones push

Según qué información nueva esté disponible para alertar al usuario, el tipo de notificación varía. Como bien podemos apreciar en la siguiente imagen:

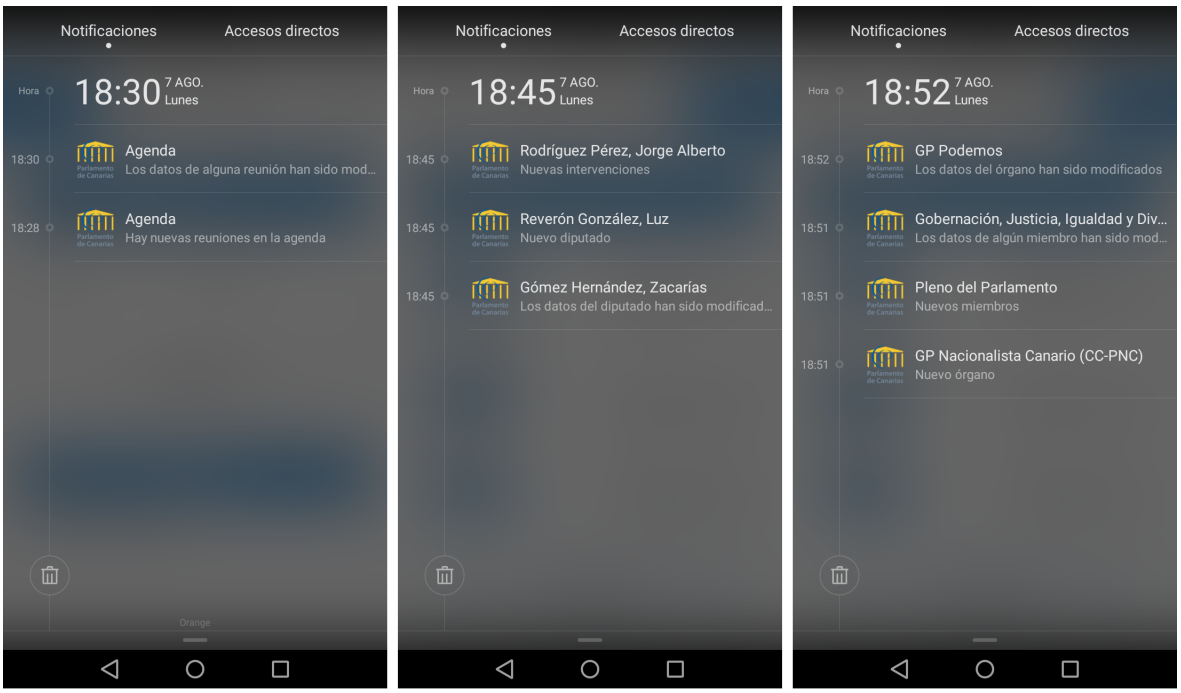


Figura 4.11: Notificaciones push

A la izquierda las notificaciones relacionadas con la agenda, en el centro las notificaciones de los diputados y a la derecha la de los órganos.

4.6.2. Alertas en las actualizaciones de la información

Para conseguir que el usuario se percate de los cambios que se producen en la información que está disponible en las distintas secciones de la aplicación, se producen básicamente dos tipos de alertas visuales. Por un lado las alertas visuales en los desplegables y por otro las alertas visuales de algún dato concreto.

Por un lado, en la sección de los diputados, cada uno de ellos está representado en su totalidad por un elemento desplegable. Entonces, todos aquellos que han sufrido alguna modificación en la información que proporcionan (datos del diputado, intervenciones...) se presentan de un color amarillo, de manera que el usuario entienda que la información de dicho diputado ha sido actualizada. En la imagen siguiente podemos apreciar la idea:

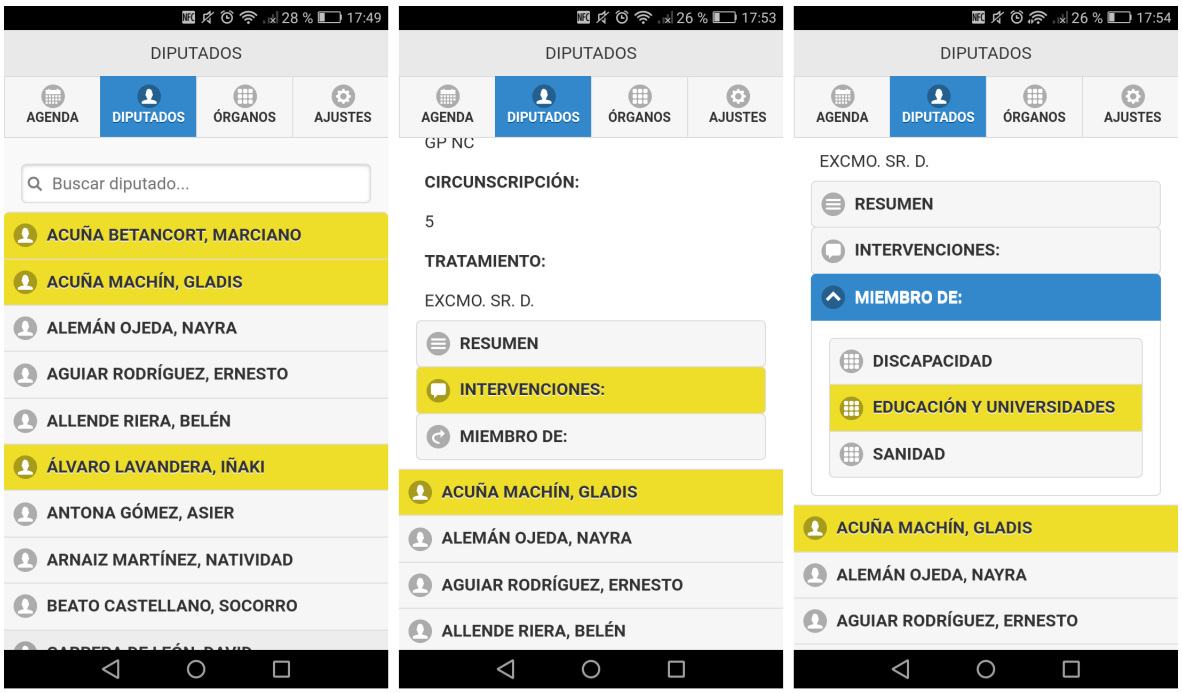


Figura 4.12: Alertas visuales en elementos desplegables

Por otra parte, dichos datos que han sido modificados, se muestran con un texto tanto representativo (con respecto a la modificación que sufre) como llamativo para que el usuario lo identifique de manera inmediata. En la imagen de abajo podemos ver que el diputado ha pasado a ser activo y que ha realizado dos nuevas intervenciones.

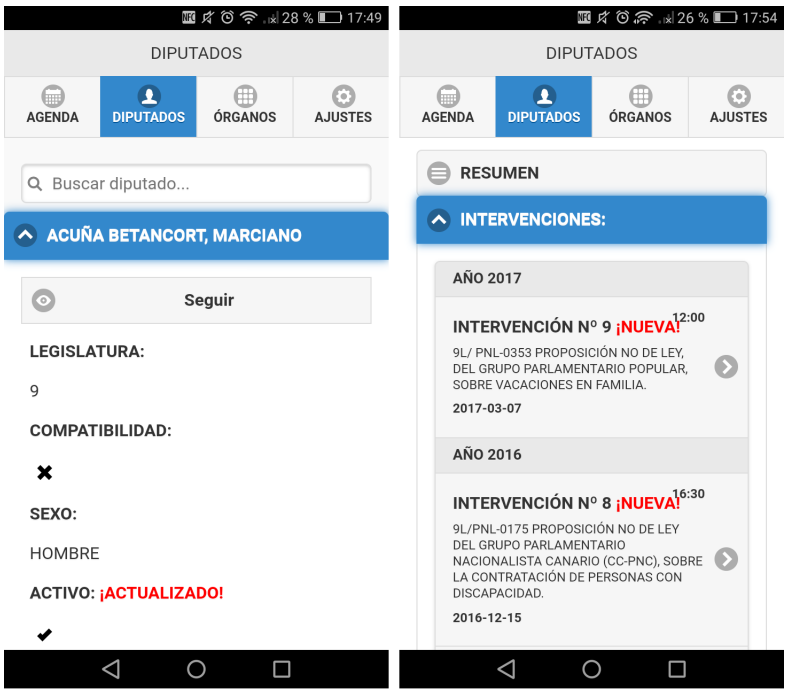


Figura 4.13: Alertas visuales en datos concretos

Capítulo 5

Despliegue

En este capítulo se describirán los pasos a seguir para desplegar la aplicación en un dispositivo Android. Cabe destacar que el desarrollo del proyecto se ha llevado a cabo en un sistema operativo Windows. Asimismo hablaremos también del despliegue de la aplicación servidor.

5.1. Despliegue en Android

Para poder desplegar nuestra aplicación en un dispositivo *Android* es necesario seguir una serie de pasos que nos permitirán construir un archivo con extensión **.apk** el cual nos hará falta para poder instalar la aplicación en nuestro móvil.

Como paso previo necesitamos tener instalado en nuestro sistema el *Java Development Kit (JDK) 7* (o superior) y, al desarrollar en *Windows*, haber definido la variable de entorno **JAVA_HOME** con la ruta de instalación del JDK.

Una vez tenemos ésto es necesario instalar el **SDK de Android**. Para ello se ha optado por hacer uso de *Android Studio* [22], a través de ésta herramienta podemos descargar el SDK, tenemos acceso al emulador y podemos compilar la aplicación de manera sencilla.

Cuando tengamos completados los pasos anteriores, desde el directorio raíz donde tengamos nuestra aplicación de *Phonegap*, a través de la línea de comandos, añadimos la plataforma *Android* haciendo uso de *cordova* como se muestra a continuación:

```
1 $ cordova platform add android
```

A continuación ejecutamos este otro comando para construir la aplicación para *Android*:

```
1 $ cordova build android
```

Estos comandos nos crean una carpeta dentro del directorio raíz del proyecto con el nombre: **/platforms/android** la cual usaremos en *Android Studio* para desplegar la aplicación en un dispositivo *Android*. Para conseguir ésto, ejecutamos *Android Studio*, elegimos la opción de importar proyecto y seleccionamos la carpeta mencionada. El proyecto tardará unos segundos en sincronizarse y estará listo para ser ejecutado.

5.2. Despliegue del servidor

Una vez terminado el desarrollo de la parte del servidor se ha optado por desplegar el mismo en *Heroku* [23]. *Heroku* es una plataforma como servicio de computación en la nube la cual soporta distintos lenguajes de programación, entre ellos, da soporte a *Node.js*.

Como paso previo para poder usar esta herramienta es necesario tener instalado *Node.js* y *npm*, además de tener una cuenta de *Heroku* creada. Por otro lado debemos descargar interfaz de líneas de comando de *Heroku*, ***Heroku CLI***, que podemos descargar desde la página oficial de la herramienta.

Una vez hemos descargado *Heroku CLI* debemos ejecutar desde la consola el comando:

```
1 $ heroku login
```

Para iniciar sesión en la plataforma haciendo uso de la dirección de correo y la contraseña que usamos para crear la cuenta de *Heroku*.

Para poder desplegar el servidor en la plataforma es necesario definir un fichero *Procfile* en el directorio raíz de nuestra aplicación servidor. Este fichero se encarga de definir qué comando se usará para arrancar la aplicación, en nuestro caso, el código del servidor lo tenemos en el fichero **index.js** así que en el *Procfile* debemos añadir que éste se ejecutará con el comando ***node*** y que es de tipo *web*. Entonces quedaría de la siguiente manera:

```
1 web: node index.js
```

Listing 5.1: Archivo Procfile

Además del fichero *Procfile* debemos tener otro llamado *package.json* el cual determina tanto la versión de *Node.js* que será usada para correr el servidor en *Heroku* como todas aquellas dependencias que la aplicación usa y que deben ser instaladas. Cuando la aplicación es desplegada en la plataforma, *Heroku* lee este fichero e instala la versión de node y las dependencias especificadas usando el comando ***npm install***. Este comando también lo podemos ejecutar localmente para instalar todas las dependencias y poder correr el servidor de manera local.

Una vez tenemos configurados los ficheros necesarios para que *Heroku* pueda trabajar correctamente es necesario crear una *app* en *Heroku* para que la herramienta esté preparada para recibir el código de nuestra aplicación. Para ello, desde la línea de comandos ejecutamos:

```
$ heroku create mGovServer
```

El cual creará una aplicación en la plataforma llamada *mGovServer* y añadirá a nuestro proyecto una rama remota *git* (llamada **heroku**) que nos servirá para enviar nuestros cambios en la aplicación. De esta manera ejecutamos:

```
$ git push heroku master
```

Una vez realizado este paso ya tenemos nuestra aplicación desplegada en *Heroku*.

Por último hay que mencionar que se ha tenido que cambiar las dependencias de la base de datos dado que la plataforma utiliza una específica en la versión gratis que estamos usando para el despliegue. Para añadir la base de datos a nuestro proyecto de *Heroku* debemos ejecutar el siguiente comando:

```
$ heroku addons:create heroku-postgresql:hobby-dev
```

Una vez añadida la base de datos, debemos editar nuestro *package.json* y añadir el módulo correspondiente a la sección de dependencias. Nuestro archivo quedaría de la siguiente manera:

```
{
  "name": "mGoverment",
  "version": "0.2.6",
  "description": "mGoverment para el Parlamento de Canarias",
  "engines": {
    "node": "6.10.2"
  },
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "body-parser": "^1.17.2",
    "ejs": "2.5.6",
    "express": "4.15.2",
    "firebase-admin": "^5.0.0",
    "node-cron": "^1.2.0",
    "pg": "6.x"
  },
  "keywords": [
    "node",
    "heroku",
    "express"
  ]
}
```

Figura 5.1: Archivo package.json

Capítulo 6

Conclusiones

En los capítulos anteriores se han presentado las herramientas usadas en el desarrollo del Trabajo de Fin de Grado, así como la manera en la que se ha ido trabajando e implementando, tanto la aplicación como el servidor.

En este capítulo se presentan las conclusiones a las que se han llegado a medida que se ha ido desarrollando el proyecto y una vez finalizado el mismo.

Además se expondrán una serie de líneas futuras a la par que posibles mejoras que se pueden realizar en la aplicación.

6.1. Conclusiones

Aunque muchas de las herramientas usadas en el presente trabajo eran conocidas previamente tras haber sido usadas en diversas asignaturas de la carrera, se ha logrado profundizar ampliamente en los detalles y prestaciones que éstas ofrecen, con lo que los conceptos que se tenían de las mismas ido creciendo a medida que el desarrollo del trabajo ha ido avanzando.

Las competencias acerca del trabajo con bases de datos han sido mejoradas dado que, en lo que a casos reales se refiere, hasta ahora, no había desarrollado prácticamente nada. Solo en la asignatura *Bases de datos* se han visto conceptos sobre las mismas, en gran parte teoría y algunos supuestos prácticos pero nada en una aplicación real.

Por otro lado, el desarrollo de aplicaciones servidor es una de las facetas que menos he tenido la oportunidad de desarrollar en la carrera, así que en este caso se ha tenido que investigar bastante. Este ha hecho que los conceptos adquiridos en este ámbito sean extensos y personalmente muy satisfactorios.

También el diseño, implementación y gestión de notificaciones era una tema completamente nuevo para mi y ha sido, entre otras cosas, uno de los aspectos que más me ha gustado desarrollar en el trabajo, si no el que más. Ahora una

vez visto y trabajado, me parece una de las características más importantes que puede tener una aplicación móvil.

Además, para el desarrollo de la presente memoria se ha hecho usando *LaTeX*, herramienta la cual previamente no se ha usado y que por lo tanto se ha dedicado parte del tiempo del desarrollo al estudio de sus características y que ahora considero una herramienta muy potente que posiblemente podré usar en futuros proyectos.

Para acabar he de decir que estoy muy contento tanto con los conceptos adquiridos como con los resultados obtenidos en el desarrollo del Trabajo de Fin de Grado y que estoy seguro que todo lo aprendido en el mismo me hará mejor Ingeniero Informático.

6.2. Líneas futuras

Como posibles mejoras o cambios en la aplicación que se ha implementado podemos centrarnos en dos partes:

6.2.1. Cambios en el diseño de la aplicación

Por un lado y siempre y cuando queramos seguir con el carácter híbrido de la aplicación, se puede contemplar el cambio del ***framework*** *jQuery Mobile* por el de algún otro más potente y vistoso, dado que este presenta un diseño bastante básico. Por ejemplo el *frameworkr* que ofrece *Ionic* [24] incluye muchas prestaciones y el diseño que presenta es más vistoso y moderno que el de *jQuery Mobile*.

Por otro lado, se podría optar por desarrollar una aplicación nativa para las distintas plataformas (Android, iOS, etc), aunque esto supondría más incremento en el coste, tanto a nivel de rendimiento como visualmente la aplicación resultaría ser mucho más elegante y funcional.

6.2.2. Cambios en las prestaciones que ésta ofrece

Con respecto al segundo punto, en un futuro la aplicación podría incluir acceso a más información del parlamento así como alguna otra funcionalidad adicional, como por ejemplo, acceso a las iniciativas presentadas.

En un principio se quería que las iniciativas presentadas se pudieran consultar desde la aplicación. Al no tener *API* disponible de esta parte, se intentó utilizar alguna técnica de ***Web scraping*** para conseguir los datos necesarios a través de la página web del parlamento pero finalmente no se consiguió. Por este hecho, una sección de iniciativas se podría implementar en un futuro.

Capítulo 7

Conclusions

In the previous chapters we have introduced the tools used in the development of this project, as well as the way in which has been working and implementing both the application and the server.

This chapter presents the conclusions that have been reached as the project has been developed and completed.

In addition, some guidelines of future work will be presented along with possible improvements that can be made in the application.

7.1. Conclusions

Many of the tools used in this project were previously known because have been used in different subjects of the career but I've improved those concepts as the project has been advancing.

The knowledge about working with databases have been improved because this is a more real case than the projects made in the other subjects like *Databases*.

In the other hand, the server application development is one of the things which I've learned less in the career so, in this case I've had to research pretty much. Because of this I've improved a lot in this subject and it's satisfying.

Also the design, implementation and notification management was completely new for me and it has been one of the things which I most liked to develop. Now to me, it's one of the most important features than an app can have.

In addition, this document has been written using *LaTeX*, tool which hasn't been used previously and for that reason I've need time to study it. Now I consider it a powerful tool I will be able to use in future protects.

To finish I must say I'm very glad with the acquired knowledge and the results obtained in the development of this project and I'm sure that all I've learned will make me a better Computer Engineer.

7.2. Future guidelines

As possible improvements or changes in the application we can point out two groups:

7.2.1. Changes in the application design

In one side, if we want to preserve the hybrid character of the application, we can swap the framework jQuery Mobile for one more powerful and colorful because the old one is very basic. For example, the Ionic framework has a lot of features and a better design than jQuery Mobile.

In the other hand, we could develop a native application for the different platforms, Android or iOs for example. In this case the application performance will be better and the design will be more elegant but the cost of development would increase.

7.2.2. Changes in the application features

With regard to the second point, in the future the application could include access to more information as well as some additional functionality, for example, access to the presented initiatives.

Initially we wanted that the presented initiatives could be consulted from the application but there wasn't API available from this part, so I attempted to use some web scraping technique to get the necessary data through the parliament website but I couldn't get it. For this reason, a section of initiatives could be implemented in the future.

Capítulo 8

Presupuesto

Este capítulo detalla el presupuesto para la elaboración del presente Trabajo Fin de Grado. Por un lado el presupuesto referente al coste de desarrollo de la aplicación y por otro, el coste referente a la contratación de un servicio para alojar la parte del servidor.

8.1. Presupuesto del trabajo

Para la elaboración de este apartado del presupuesto se ha tenido en cuenta el tiempo dedicado a todas las tareas realizadas. Se ha contemplado la contratación de un único desarrollador que cobraría un supuesto de **20€/hora** durante un periodo de seis semanas, trabajando treinta horas semanales hasta llegar a un total de ciento ochenta horas.

En la siguiente tabla se detallan las distintas actividades realizadas y el coste asociado al tiempo dedicado en cada una de ellas:

Actividad	**Duración (horas)**	**Coste (€)**
Evaluación de las plataformas de desarrollo	10	200
Diseño de la aplicación	30	600
Implementación	100	2000
Elaboración de la memoria	40	600
TOTAL	**180**	**3600**

Tabla 8.1: Tiempo dedicado y presupuesto

8.2. Presupuesto destinado al alojamiento del servidor

Para la fase de desarrollo de la aplicación se ha hecho uso de la versión gratis de la plataforma **Heroku** la cual nos permite alojar nuestro servidor pero tiene el pequeño inconveniente de que, tras treinta minutos de inactividad, éste se duerme.

Como tenemos tareas programadas que se deben ejecutar cada cierto tiempo, sería necesario contratar, al menos, la versión más barata de la plataforma para que este hecho ya no fuera un problema. El coste del servicio es de 7 dólares al mes.

En la siguiente imagen la comparativa de la versión gratis con la versión que contratamos:

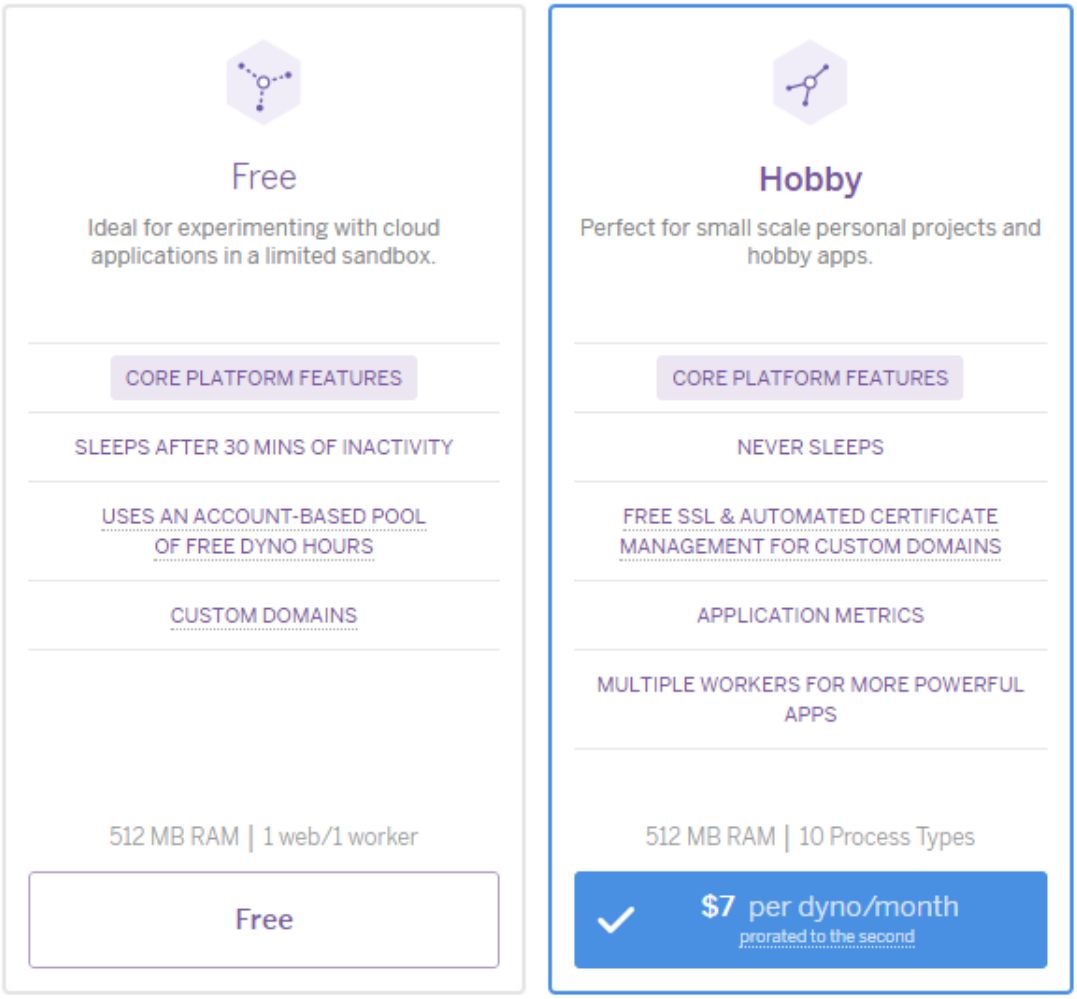


Figura 8.1: Comparativa de prestaciones de la plataforma Heroku [25]

Bibliografía

[1] **Artículo de Wikipedia sobre mGovernment** - artículo web. Enlace [Disponible electrónicamente; Último acceso septiembre de 2017]. 1

[2] **Comparativa plataformas de desarrollo** - artículo web. Enlace [Disponible electrónicamente; Último acceso septiembre de 2017]. 6

[3] **PhoneGap** - página web oficial. Enlace [Disponible electrónicamente; Último acceso abril de 2017]. 6, 9

[4] **Cordova** - página web oficial. Enlace [Disponible electrónicamente; Último acceso agosto de 2017]. 7

[5] **HTML5** - página web oficial. Enlace [Disponible electrónicamente; Último acceso abril de 2017]. 7, 8

[6] **W3C** - consorcio world wide web, españa. Enlace [Disponible electrónicamente; Último acceso abril de 2017]. 7

[7] **JavaScript** - página web oficial. Enlace [Disponible electrónicamente; Último acceso abril de 2017]. 7

[8] **CSS3** - página web oficial. Enlace [Disponible electrónicamente; Último acceso abril de 2017]. 8

[9] **JQuery** - página web oficial. Enlace [Disponible electrónicamente; Último acceso abril de 2017]. 8

[10] **JQuery Mobile** - página web oficial. Enlace [Disponible electrónicamente; Último acceso abril de 2017]. 9

[11] **Firebase** - página web oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 9

[12] **NodeJS** - página web oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 10, 27

[13] **Node Package Manager (npm)** - página web oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 10

[14] **PostgreSQL** - página web oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 10

[15] **Install Phonegap** - página web oficial. Enlace [Disponible electrónicamente; Último acceso agosto de 2017]. 12

[16] **Firebase Cloud Messaging** - documentación oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 25

[17] **Firebase Console** - página web oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 25

[18] **ExpressJS** - página web oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 27

[19] **Firebase Admin Setup** - documentación oficial. Enlace [Disponible electrónicamente; Último acceso mayo de 2017]. 27

[20] **pg-promise** - repositorio github. Enlace [Disponible electrónicamente; Último acceso agosto de 2017]. 28

[21] **Node-cron** - repositorio github. Enlace [Disponible electrónicamente; Último acceso agosto de 2017]. 35

[22] **Android Studio** - documentación oficial. Enlace [Disponible electrónicamente; Último acceso agosto de 2017]. 47

[23] **Heroku** - página web oficial. Enlace [Disponible electrónicamente; Último acceso agosto de 2017]. 48

[24] **Ionic** - página web oficial. Enlace [Disponible electrónicamente; Último acceso agosto de 2017]. 51

[25] **Planes de contratación de la plataforma Heroku** - página web oficial. Enlace [Disponible electrónicamente; Último acceso septiembre de 2017]. 55