Trabajo de Fin de Grado

SchoolApp: Comunicación desde las aulas.

SchoolApp: Communication from the classroom. .

Gonzalo J. García Martín

La Laguna, 8 de septiembre de 2015

D. **Francisco de Sande González**, con DNI número 42067050–G profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

SchoolApp: Comunicación desde las aulas.

ha sido realizada bajo su dirección por D. **Gonzalo J. García Martín**, con N.I.F. 78.628.877-K.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre de 2015

Agradecimientos

A mis padres por todo el apoyo mostrado.
A mis amigos por soportar las divagaciones de mi mente con este proyecto y en general.
A mis compañeros de clase por todas las horas pasadas en la facultad, sin ellos se me habrían echo eternas.
A mi tutor de proyecto, Francisco de Sande González, por toda la dedicación, paciencia y tiempo empleados.

Licencia

Resumen

El objetivo general de este proyecto ha sido crear una aplicación para Android usando las nuevas tecnologías y los servicios de almacenamiento en internet, comúnmente conocidos como nube. Se pretendía establecer un sistema de comunicación escolar para que los padres, madres, tutores legales, profesores y alumnos estén comunicados entre sí.

Con el uso de estas tecnologías se ha incrementado la cantidad de problemas escolares. Ya no son solo los que se originan en el interior de las aulas y que los profesores pueden controlar, sino que además éstos se han extendido afectando a los estudiantes en otros ámbitos.

Partiendo de esta base, y con los conocimientos adquiridos a lo largo de los estudios del Grado en Ingeniería Informática de la Universidad de La Laguna, se ha diseñado una aplicación que pueda servir de ayuda para combatir estos problemas en el ámbito académico.

Para esto fue utilizado el entorno de desarrollo integrado, AndroidStudio, en conjunto con Firebase, un proveedor de servicios en la nube.

Palabras Clave: *Aplicaciones Android, dispositivos móviles, colegios, educación, herramientas comunicativas.*

Abstract

The overall objective of this project has been to create an Android application using new technologies and the Internet storage services, commonly known as cloud. It was intended to establish a system of school communication for parents, teachers and students to be interconnected.

With the use of these technologies the number of school problems has increased. It's not just those originating inside the classrooms and that teachers can control, but also they have spread to affect students in other areas.

On this basis, and with the knowledge acquired throughout studies in the Degree in Computer Engineering from the University of La Laguna, we have designed an application that can assist to solve this problems.

For this we used the integrated development environment, AndroidStudio, along with Firebase, a cloud services provider.

Keywords: *Android applications, mobile devices, schools, education, communication tools.*

Índice general

1. Introducción **1**
- 1.1. Objetivos 1
- 1.2. Antecedentes 3
  - 1.2.1. Aplicaciones similares 3

2. Especificación de Requisitos **6**
- 2.1. Funcionalidades 6
- 2.2. Sistemas de la Aplicación 8
- 2.3. Perfiles 10

3. Herramientas **12**
- 3.1. Otros Servicios en la Nube: Parse [1] 13
- 3.2. Instalación de las herramientas usadas 16
  - 3.2.1. AndroidStudio 17
  - 3.2.2. Firebase 18
- 3.3. Tutoriales 20

4. Descripción de la Aplicación **22**
- 4.1. Pantalla Principal (`MainActivity`) 24
  - 4.1.1. Bienvenida (`WelcomeFragment`) 24
  - 4.1.2. Ayuda (`HelpFragment`) 24
  - 4.1.3. Sobre mi (`AboutMeFragment`) 24
- 4.2. Cambio de idioma (`ChangeLanguajeActivity`) 24
- 4.3. Registro (`RegisterActivity`) 25
  - 4.3.1. Añadir alumno (`AddChildActivity`) 26
- 4.4. Acceso (`LoginActivity`) 27

4.5. Acceso de Usuarios 29
4.6. Notificaciones (`notificationsActivity`) 30
4.7. Chat (`ChatActivity`) 30
4.8. Mis Datos (`MyDataActivity`) 31
4.9. Cambio de Contraseña (`ChangePasswordactivity`) . . 31
4.10. Circulares (`CircularesActivity`) 31
4.11. Citas 32
4.12. Datos del contacto (`DataActivity`) 32

5. Desarrollo de la Aplicación 33
5.1. Clase `Student` 33
5.2. Clase `Father` 35
5.3. Clase `Teacher` 35
5.4. Clase `Message` 35
5.5. Clase `MessageSQLHelper` 36
5.6. `TabActivityes` 38
5.7. Problemas 41

6. Conclusiones y Trabajos futuros 46
6.1. Trabajos futuros 46
6.2. Conclusiones 47

7. Conclusions and Future works 48
7.1. Future works 48
7.2. Conclusions 49

8. Presupuesto 50

A. Diagrama de Clases 52

B. Diagrama de Actividades 53

Índice de figuras

1.1. Pantalla de bienvenida de la aplicación Remind 4
1.2. Pantalla de bienvenida de la Aplicación miColegioApp 5

3.1. Paquetes en SDK Manager 18

4.1. Navegación en las pantallas de `SchoolApp`. 23
4.2. Pantalla de Bienvenida de la Aplicación. 25
4.3. Pantalla de ayuda de la Aplicación. 26
4.4. Pantalla de registro para el perfil de usuario. 27
4.5. Registro de hijos. 28
4.6. Pantalla de acceso con el dispositivo en horizontal. . . . 28
4.7. Lista de contacto de alumnos para el perfil de profesor. 29
4.8. Notificaciones recibidas por los usuarios. 30
4.9. Datos del usuario. 32

8.1. Horas empleadas en el desarrollo de la aplicación . . . 50

A.1. Diagrama de Clases 52

B.1. Diagrama de Actividades 54

Índice de Tablas

2.1. Comunicaciones permitidas 9
2.2. Comunicaciones permitidas para el perfil de alumno . . 10
2.3. Comunicaciones permitidas para el perfil de profesor. . 11
2.4. Comunicaciones permitidas para el perfil de padre. . . 11

Índice de Códigos

3.1. Importación de la librería de *Parse* 14
3.2. Ejemplo de uso de *Parse* (parte del código) 16
3.3. Importación de la librería de *Firebase* 19
3.4. Ejemplo de uso de *Firebase* 20
5.1. Ejemplo de la clase Student. 34
5.2. Ejemplo de la clase MessageSQLHelper. 37
5.3. Ejemplo de la clase StudentTabActivity. 38
5.4. Ejemplo de la clase StudentActivity. 39
5.5. Ejemplo de la clase StudentRegisterActivity. 40
5.6. Solución a la duplicidad en `build.gradle`. 42
5.7. Solución al problema supportV13. 43
5.8. Solución en `build.gradle`. 44
5.9. Solución en `app/build.gradle`. 45

Capítulo 1

Introducción

`SchoolApp` persigue crear un sistema con el que la comunidad educativa se pueda comunicar de forma continua empleando las nuevas tecnologías. Esta temática surge a partir de los problemas escolares que se han ocasionado en los últimos años, como el acoso escolar o la falta de comunicación del personal docente con padres y alumnos.

La aplicación pretende ser una herramienta para que los profesores puedan transmitir continuamente toda la información que consideren relevante a los padres y a sus alumnos. Esto evitará sorpresas desagradables, malentendidos e incluso algunos problemas. Esta aplicación está limitada al uso que los usuarios hagan de ella.

Esta aplicación va dirigida a la comunidad educativa de primaria y secundaria porque son alumnos a los que se les debe prestar atención, tanto a su formación como a los problemas que puedan surgir. Sus estudios son una base fundamental de su vida y debe estar vigilada. Si a un problema sin importancia no se le presta atención, este puede pasar a ser algo grave.

1.1. Objetivos

Se pretende crear una aplicación para dispositivos móviles, que utilicen como sistema operativo *Android* [2], como trabajo de fin de

grado utilizando las nuevas tecnologías como los teléfonos inteligentes y los servicios de internet conocidos como Nube [3]. También se pretende adquirir las competencias relacionadas con la ingeniería del software y las ciencias de la computación.

A continuación se pasará a enumerar algunos de los objetivos específicos propuestos para este proyecto:

- Aplicar los conocimientos adquiridos en los estudios del Grado de Ingeniería Informática de la Universidad de La Laguna.
- Ampliar conocimientos sobre *Android*.
- Emprender el diseño y desarrollo de un proyecto.
- Adquirir conocimientos sobre el uso de servicios en la nube y su uso en aplicaciones *Android*.
- Gestión proyectos con *Github* [4]: Plataforma de desarrollo colaborativo utilizada para el diseño y producción de software. Utiliza el *control de versiones git* [5].
- Crear una memoria técnica sobre la aplicación desarrollada en el presente Trabajo de Fin de Grado.
- Adquirir conocimientos sobre *Latex* [6]: Sistema de composición de textos, orientado a la creación de documentos escritos que presenten alta calidad tipográfica. Se ha utilizado *Latex* para componer la presente memoria.

`SchoolApp` no está pensada para ofrecer funcionalidades de las que carecen otras aplicaciones, ni para competir formalmente con ellas. Más bien está pensada para adquirir nuevos conocimientos y poner en práctica los ya adquiridos.

1.2. Antecedentes

Con los nuevos avances tecnológicos se ha vuelto popular el uso de las comúnmente denominada *apps* en los dispositivos móviles. Hay aplicaciones para todos los gustos, desde lo más básico como aprender a cocinar o aplicaciones de comunicación, hasta de lo más importante como por ejemplo consultar la información de la cuenta del banco e incluso hacer operaciones con ella.

Se ha observado que en el ámbito docente hay una carencia de comunicación entre padres y profesores. También el aumento de problemas como el acoso, fracaso escolar o incluso problemas de ámbito familiar influyen en los alumnos. Por eso los hasta ahora recursos tradicionales no eran suficientes. Notas escritas y reuniones no son más que informes puntuales de un progreso continuo que puede decaer sin aviso previo.

Por eso se presenta una herramienta que intenta establecer un flujo de información continua sin comprometer los datos personales de los usuarios tales como el número de teléfono o el correo electrónico. Así éstos se sentirán cómodos a la hora de comunicarse de forma segura. Se entiende que es un esfuerzo extra para los equipos docentes, pero permitirá que haya una mejor comunicación para resolver problemas inesperados y actuar de forma casi inmediata.

1.2.1. Aplicaciones similares

A continuación se describirán aplicaciones similares a `SchoolApp`.

Remind

Remind [7] ofrece a los profesores una forma sencilla y segura para enviar SMS a estudiantes y padres. Profesores, monitores o administradores pueden enviar recordatorios, asignaciones, deberes, evaluaciones o mensajes motivadores directamente a los teléfonos de estudiantes y padres. El envío de mensajes es seguro porque

los números de teléfono se mantienen confidenciales. Los profesores pueden ahorrar tiempo mediante el envío unidireccional de anuncios, u optar por una comunicación por chat personalizada, con un solo estudiante o padre. Con Remind, es más fácil para estudiantes y padres mantenerse informados fuera de la clase. Profesores, estudiantes y padres pueden descargar la aplicación Remind para mantenerse conectados más fácilmente.

El profesor creará una o varias clases y obtendrá un código por cada clase que cree. Con este código los padres y alumnos podrán inscribirse a sus clases, recibiendo notificaciones o mensajes de los profesores, monitores y administradores.

En la figura 1.1 se puede apreciar la pantalla de bienvenida de la aplicación.

Figura 1.1: Pantalla de bienvenida de la aplicación Remind

miColegioApp

En *miColegioApp* [8] el usuario se registra como alumno, profesor, padre, madre, tutor legal o persona autorizada. Una vez registrado, se solicita el código al centro, que debe haber contratado los servicios de la empresa previamente, y se introduce en la aplicación. Cuando se

ha asociado al usuario con un centro, el dueño del dispositivo recibirá notificaciones y circulares del colegio. En la figura 1.2 se observa la pantalla de inicio.

CreaTáctil [9] es una empresa de base tecnológica que ofrece soluciones personalizadas en el campo de la tecnología y la informática, con especialización en el desarrollo de aplicaciones multiplataforma. Persiguen convertirse en una referencia en el desarrollo de aplicaciones móviles.

Figura 1.2: Pantalla de bienvenida de la Aplicación miColegioApp

Capítulo 2

Especificación de Requisitos

La especificación de requisitos es una parte fundamental de todo desarrollo ya que describe de forma completa el comportamiento funcional del software. Incluye casos de uso, requisitos funcionales, no funcionales, de usuarios y del sistema. Va dirigido tanto al cliente como al equipo de desarrollo y establece unos compromisos fundamentales entre ambos.

En este capítulo se describirán las funcionalidades, los sistemas integrados en la aplicación y los perfiles o roles que podrán tener los usuarios.

En el caso de `SchoolApp` son varias las funcionalidades integradas en la aplicación, desde el registro y el acceso de usuarios hasta el envío de mensajes y la recepción notificaciones.

2.1. Funcionalidades

A continuación se describen las funcionalidades de la aplicación:

- **Registro**: Tras rellenar el formulario y accionar el botón de registro, `SchoolApp` enviará los datos a un proveedor de servicios

en la nube. Éstos serán almacenados en la base de datos, en sus tablas correspondientes. También creará al usuario para que tenga acceso a la aplicación.

- **Acceso**: El usuario introducirá su dirección de correo electrónico y contraseña en los campos destinados a ello. El dispositivo móvil se autenticará contra el servidor y si todo es correcto permitirá el acceso.
- **Recuerdo de Contraseña**: La aplicación solicitará al servicio en la nube una contraseña temporal para que el usuario pueda acceder, tras lo cual tendrá que editar su contraseña en `SchoolApp`.
- **Contactos**: Se recuperarán todos los usuarios relacionados con el usuario que esté usando la aplicación. Esto permitirá que se puedan comunicar.
- **Chat**: Los usuarios se podrán enviar mensajes entre sí, éstos se almacenaran en el servidor para ser recuperados por la aplicación y luego borrados de la nube. La comunicación será del tipo alumno-profesor, padre-profesor y viceversa en ambos casos. En ningún ámbito de la aplicación se permitirá la comunicación entre padres y alumnos, pues no tiene sentido.
- **Notificaciones**: `SchoolApp` recuperará de la base de datos todos los mensajes que tenga el usuario, para después borrarlos de su almacenamiento en la nube. Éstas se diferenciarán por colores según el perfil del remitente.
- **Circulares**: Aviso que enviará un profesor a todos los integrantes de una clase, e incluso de varias. Estos mensajes se almacenarán en la nube y serán recuperados por la aplicación, que los eliminará de su almacenamiento en internet.
- **Citas**: Permitirá al usuario programar un evento en el calendario por defecto sin salir de la aplicación. Utiliza la funcionalidad para añadir eventos que implementa el sistema operativo del

dispositivo. Se introducirán los datos en los campos correspondientes y se almacenará la cita.

- **Visualización de Datos**: El usuario podrá visualizar la información de sus contactos al realizar una selección larga en cualquiera de ellos.
- **Modificación de datos**: El usuario podrá editar los datos en el dispositivo móvil, tras lo cual se actualizarán en la base de datos almacenada en la nube.
- **Darse de baja**: El usuario podrá eliminar su cuenta de la aplicación y su información en el servidor.

2.2. Sistemas de la Aplicación

- Sistema de Registro: Al usuario se le solicitarán los siguientes datos:
  - Nombre y Apellidos.
  - Número de teléfono.
  - D.N.I.
  - Nombre del centro con el que va a hacer el registro.
  - Clase y grupo.
  - Correo electrónico y contraseña.
  - En caso de que el usuario se registre como padre también tendrá que introducir:
    - Nombre y Apellidos del alumno.
    - D.N.I del alumno.
    - Clase en la que está el alumno.
    - Centro en el que está el alumno.
    - Curso en el que está el alumno.

- Teléfono del alumno, este dato es opcional.
- Correo electrónico del alumno, este dato es opcional.

- Sistema de "acceso": Se le solicitará al usuario que ingrese su identificación, es decir, su correo electrónico y contraseña.

- Sistema de comunicación entre usuarios: Este servicio enviará los mensajes a través de internet usando los servicios de un proveedor. Se explicará en la Tabla 2.1.

	Padre	Alumno	Profesor
Padre	Mensajes	-	Mensajes
Alumno	-	Mensajes	Mensajes
Profesor	Circulares y Mensajes	Circulares y Mensajes	Mensajes

Tabla 2.1: Comunicaciones permitidas

En la primera columna aparecen los emisores de información y en la primera fila los receptores. Esto quiere decir que si un padre pretende comunicarse con un profesor, solo puede hacerlo mediante mensajes directos tipo chat. Pero si es el profesor el que se comunica, puede hacerlo usando circulares o mensajes. Una circular es un mensaje que le llega a toda la clase.

No tiene sentido que un usuario con el perfil de padre se pueda comunicar con los compañeros de clase de su hijo.
Las comunicaciones bidireccionales profesor-padre y profesor-alumno son necesarias para que los alumnos y los padres puedan recibir mensajes y notificaciones.

- Sistema de citas: Permitirá al usuario programar un evento en el calendario del dispositivo sin salir de la aplicación, usando las funciones que implementa *Android*.

- Lista de contactos: Organizada por grupos desplegables, por ejemplo los profesores tendrán la lista organizada según las clases que impartan en el centro.

Esto permitirá establecer un sistema de comunicaciones entre los profesores y padres de alumnos.

2.3. Perfiles

- Alumnos: Podrán programar citas en el calendario y recibir circulares. Como se puede observar en la tabla 2.2, las comunicaciones entre alumnos y padres no están permitidas en ningún ámbito de la aplicación, ya que podría ocasionar problemas no deseados. Los estudiantes podrán conversar con los profesores para resolver las dudas y problemas que ellos consideren oportunos. También podrán comunicarse entre sí para solicitar apuntes y resolver dudas entre otras razones.

	Padre	Alumno	Profesor
Alumno	-	Mensajes	Mensajes

Tabla 2.2: Comunicaciones permitidas para el perfil de alumno

- Profesores: Podrán programar citas en el calendario y enviar circulares. Los padres y alumnos en la lista de contactos estarán organizados según las clases que impartan en el centro. Como se explica en la tabla 2.3, los profesores se pueden comunicar con todos los usuarios a través de mensajes directos tipo chat, mientras que con los padres y alumnos, además, pueden usar mensajes tipo circulares que le llegarán a todos los integrantes de la clase. Esto cumple con el objetivo de que tanto alumnos como padres estén informados de las incidencias ocurridas en clase.

	Padre	Alumno	Profesor
Profesor	Circulares y Mensajes	Circulares y Mensajes	Mensajes

Tabla 2.3: Comunicaciones permitidas para el perfil de profesor.

- Padre/Madre/Tutor Legal: Podrán recibir circulares y programar citas en el calendario. Los profesores y padres en la lista de contactos estarán organizados según las clases que impartan a los alumnos con los que el progenitor esté relacionado. En la tabla 2.4 se puede observar que los padres se pueden comunicar con los profesores para estar informados de todo lo que ocurre con sus hijos.

	Padre	Alumno	Profesor
Padre	Mensajes	-	Mensajes

Tabla 2.4: Comunicaciones permitidas para el perfil de padre.

Capítulo 3

Herramientas

El software usado en el desarrollo de `SchoolApp` ha sido el siguiente:

- **AndroidStudio** [10]: Entorno de desarrollo integrado oficial para el desarrollo de aplicaciones para *Android*, basado en *IntelliJ IDEA* [11]. Ha sido seleccionado por ser un entorno de programación moderno, de uso intuitivo y ligero en comparación con otros entornos de desarrollo ya existentes como *eclipse* [12] o *netbeans* [13] que son más pesados, ya que soportan más lenguajes de programación.
- **Android** [2]: Sistema operativo basado en el *núcleo de Linux* para dispositivos móviles, televisores, automóviles y relojes inteligentes.
- **Firebase** [14]: Proveedor de contenidos que proporciona servicios en la nube de forma fácil y segura, con una integración sencilla con las nuevas tecnologías. Ofrece servicios de recuperación y almacenamiento de datos, registro y acceso de usuarios, reglas de seguridad, simulador y análisis de datos entre otros. Los datos almacenados en este servicio no son datos *SQL*, sino que son datos *JSON* [15]. Ha sido elegido por ser uno de los servicios en la nube más sencillos de utilizar. Sistemas con los que *Firebase* está integrado:
  - **Android**: Sistema Operativo para dispositivos móviles propiedad de Google.

- **IOS** [16]: Sistema operativo para dispositivos móviles propiedad de Apple Inc.
- **Servicios Web**.
- **Servicios REST** [17]: Es un conjunto de principios de arquitectura, pero en la actualidad se usa en el sentido más amplio para describir cualquier interfaz entre sistemas que utilice directamente HTTP [18] para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP [19].

3.1. Otros Servicios en la Nube: Parse [1]

Proveedor de servicios en la nube alternativo a *Firebase* que ofrece eventos, autenticación de usuarios, almacenamientos de datos, análisis y *notificaciones push* entre otros.

Tras la implementación de una aplicación ejemplo para valorar *Parse*, se ha seleccionado *Firebase* por estar diseñado para la construcción de datos en tiempo real. Al acceder a través de una librería cliente los datos se sincronizan en tiempo real rápidamente. También ofrece una *interfaz de programación de aplicaciones* [20] de seguridad basada en la expresión altamente flexible que permite un control preciso sobre el acceso a los datos de los clientes.

Parse está integrado con las siguientes tecnologías:

- *IOS*.
- *Android* [2].
- *Javascript*.
- *OSX*.
- *Unity*: Software para la creación de videojuegos.
- *PHP*: Lenguaje de programación.

- *.Net + Xamarin*.
- *Arduino*: Hardware libre que consiste en una placa con un microcontrolador y un entorno de programación.
- *Embedded C*: Lenguaje de programción que extiende sus funcionalidades de C.
- *Servicios REST*.

Instalación

- **Descarga**: Descargar librería de *Parse*.
- **Librería**: Añadir la librería de *Parse* en el archivo `build.gradle` que esta dentro de la carpeta `app` en el directorio raíz del proyecto. Véase listado 3.1

```
apply plugin: 'com.android.application'

android {
    //...

    defaultConfig {
        //...
    }
    buildTypes {
        release {
            //...
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile files('libs/Parse-1.8.1.jar', 'libs/bolts-android-1.1.4.jar')
}
```

Listado 3.1: Importación de la librería de *Parse*

- **Uso**: En los archivos de clase `.java` [21] seguir los siguientes pasos:
  - **Librerías**: Importar las librerías necesarias:
    - **Cliente Firebase**: Importar la librería del cliente.
    - **Consultas**: Importar la librería de consultas (`ParseQuery`).
    - **Listeners**: Importar la librería de oyentes (`FindCallback`).
    - **Excepciones**: Importar la librería de excepciones (`ParseException`).
    - **Objetos**: Importar la librería de objetos (`ParseObject`).
    - **Usuarios**: Importar la librería de atenticación de usuarios (`ParseUser`).
  - **Claves**: Crear dos contantes de tipo cadena (*String*) en las introduciremos manualmente la identificación de la aplicación (*AppID*) y la clave del cliente (*CLIENT_KEY*).
  - **Contexto**: Añadir el contexto en que se va a usar en la función `onCreate()` con la identificación y la clave.
  - **Consultas**: Preparar la consulta que se va a hacer.
  - **Oyentes**:Añadir un oyente (*Listener*) a la consulta.

  En el listado 3.2 se presenta un ejemplo de uso de la librería.

```java
//Importar Librerias de Firebase: FindCallback, Parse
//ParseException, ParseObject, ParseQuery, ParseUser
public class MainActivity extends ListActivity {
    public static final String APPLICATION_ID = "APP_ID";
    public static final String CLIENT_KEY = "ClientKey";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...
        //Inicializando Parse
        Parse.initialize(this, APPLICATION_ID, CLIENT_KEY);
        //Para autenticacion de Usuarios
        ParseUser currentUser = ParseUser.getCurrentUser(); }
    private void refreshPostList() {
        Object[] authors = {null, ParseUser.getCurrentUser()};
        //Ejemplo de consulta a Parse
        ParseQuery<ParseObject> query = ParseQuery.getQuery("Post");
        query.whereEqualTo("author", null); //Notas sin autor
        query.whereContainedIn("author", Arrays.asList(authors));
        query.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> postList, ParseException e) {
                if (e == null) {
                    posts.clear();
                    for (ParseObject post : postList) {
                        Note note = new Note(post.getObjectId(),
                                post.getString("title"),
                                    post.getString("content"));
                        posts.add(note);}//for
                    ((ArrayAdapter<Note>)
                        getListAdapter()).notifyDataSetChanged(); } else {
                    Log.d(getClass().getSimpleName(), "Error: " +
                        e.getMessage());
                }}});//query.findInBackground
    }}//class
```

Listado 3.2: Ejemplo de uso de *Parse* (parte del código)

3.2. Instalación de las herramientas usadas

En esta sección se procederá a explicar la instalación de *AndroidStudio* y *Firebase*.

3.2.1. AndroidStudio

- **JDK**: Descargar e instalar *Java Development Kit* [22].
- **Descarga**: Descargar *AndrodiStudio*.
- **Instalación**: Ejecutar el archivo de instalación descargado y seguir los pasos indicados por el instalador.
- **SDK**: Añadir paquetes *SDK* con el gestor de paquetes (*SDK Manager*) seleccionándolo en la barra de herramientas.
- **Paquetes**: Seleccionar las versiones de *Android* [2] a instalar, se pueden elegir o quitar paquetes concretos de cada versión. Darle a *Install*. Es importante que ademas de instalar las versiones a utilizar, se instalen los siguientes paquetes:
  - *Android SDK Tools*.
  - *Android SDK Platform-tools*.
  - *Android SDK Build-tools* (La versión más actual).
  - Extras >*Android Support Repository*.
  - Extras >*Android Support Library*.
  - Extras >*Google USB Driver*.

En la figura 3.1 se pueden observar los paquetes que se pueden instalar.

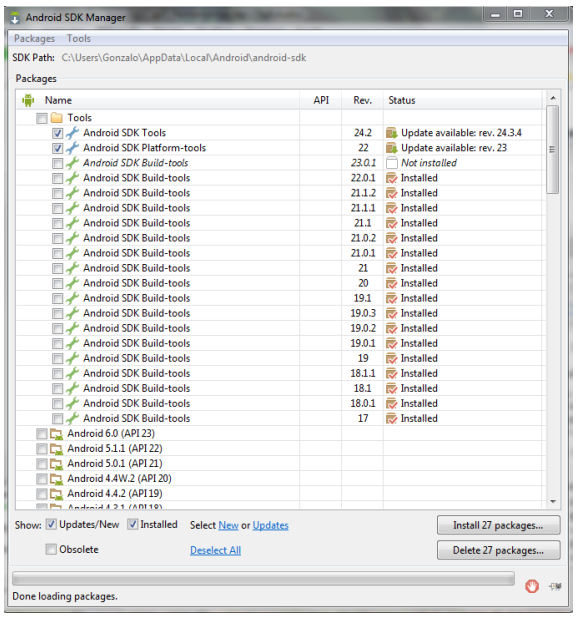

Figura 3.1: Paquetes en SDK Manager

3.2.2. Firebase

- **Librería**: Añadir la librería de *Firebase* en el archivo `build.gradle` que está dentro de la carpeta `app` en el directorio raíz del proyecto `SchoolApp`. En el listado 3.3 se puede observar cómo se importa la librería de *Firebase*.

- **Uso**: En los archivos de clase `.java` [21] seguir los siguientes pasos:

  - **Librerías**: Se han de importar la librerías necesarias:
    - **Cliente Firebase**: Importar la librería del cliente.
    - **Errores Firebase**: Importar la librería de errores en consultas.
    - **Datos Firebase**: Importar la librería que permite la devolución de datos desde *Firebase*.

```
apply plugin: 'com.android.application'

android {
    //...

    defaultConfig {
        //...
    }
    buildTypes {
        release {
            //...
        }
    }
    packagingOptions {
        //...
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile 'com.android.support:support-v4:21.0.3'
    compile 'com.android.support:support-v13:21.0.3'
    compile 'com.firebase:firebase-client-android:2.2.1'
}
```

Listado 3.3: Importación de la librería de *Firebase*

  - **Consultas Firebase**: Importar la librería para consultas.
  - **Librería Oyentes**: Importar la librería de los oyentes (*Listeners*).
- **Contexto**: Añadir el contexto en que se va a usar en la función `onCreate()`.
- **Referencias**: Añadir una referencia a la base de datos o tabla de la misma a la que se van a hacer las consultas.
- **Consultas**: Preparar la consulta que se va a hacer.
- **Oyentes**: Añadir un oyente (*Listener*) a la consulta.

En el listado 3.4 se presenta un ejemplo del uso de la librería *Firebase*.

```
//...
//Importar libreias de firebase.client ChildEventListener
//DataSnapshot, Firebase, FirebaseError, Query;
//...
public class FirebaseExample extends Activity {
  Firebase ref;
  //...
  public void onCreate (Bundle savedInstanceBundle) {
    super.onCreate(savedInstanceBundle);
    Firebase.setAndroidContext(this);
    ref = new Firebase ("URL de la tabla en la BBDD");
    //...
  }
  public void preparingData () {
    //Ejemplo de Consulta
    Query query = ref.orderByChild(colegio).equalTo(email);
    query.addChildEventListener( new ChildEventListener() {
      @Override
      public void onChildAdded(DataSnapshot dataSnapshot, String s) {}
      @Override
      public void onChildChanged(DataSnapshot dataSnapshot, String s) {}
      @Override
      public void onChildRemoved(DataSnapshot dataSnapshot) {}
      @Override
      public void onChildMoved(DataSnapshot dataSnapshot, String s) {}
      @Override
      public void onCancelled(FirebaseError firebaseError) {}
    });
  }}
```

Listado 3.4: Ejemplo de uso de *Firebase*

3.3. Tutoriales

Antes de comenzar con el desarrollo de la aplicación propiamente dicha, se han implementado diversas aplicaciones a modo de ejemplo para afianzar los conocimientos sobre *Android*.

1. *Build your first app* [23]
2. *Adding the action bar*
3. *Supporting Different Devices*
4. *Managing the Activity Lifecycle*
5. *Building a Dynamic UI with Fragments*

6. *Saving Data*
7. *Interacting with Other Apps*
8. *Sharing Simple Data*
9. *Sharing Files*
10. *Sharing Files with NFC*
11. *Managing Audio Playback*
12. *Capturing Photos*
13. *Printing Content*
14. *Displaying Bitmaps Efficiently*
15. *Displaying Graphics with OpenGL ES*
16. *Animating Views Using Scenes and Transitions*
17. *Adding Animations*
18. *Connecting Devices Wirelessly*
19. *Performing Network Operations*
20. *Syncing to the Cloud*
21. *Transferring Data Using Sync Adapters*
22. *Best Practices for Security & Privacy*
23. *Best Practices for Testing*
24. *Parse*
25. *Firebase Storage*
26. *Expandable ListView*
27. *TabHost Swipe* [24]
28. *ActionBar Tab Swipe*

Los tutoriales más relevantes para el desarrollo de `SchoolApp` fueron *Supporting Different Devices* (tutorial 3) que ayudó a implementar el soporte de idiomas y *Building a Dynamic UI with Fragments* (tutorial 5), *ActionBar Tab Swipe* (tutorial 28) y *TabHost Swipe* (tutorial 27) gracias a los cuales existe `WelcomeActivity`.

También han sido significativos *Adding Animations* (tutorial 17) para el *cuadro de diálogo* **cargando**, *Firebase Storage* (tutorial 25) que permitió la programación del uso de la base de datos y *Expandable ListView* (tutorial 26) cuyos conocimientos se utilizaron en el desarrollo de la lista de contactos desplegable.

Capítulo 4

Descripción de la Aplicación

En este capítulo se hará una descripción de la interfaz gráfica y de cada una de las funcionalidades de las pantallas de `SchoolApp`. Antes de comenzar con este apartado debemos explicar varios términos:

- *Activity*: Componente del modelo Vista-Controlador usado por Android y que provee al usuario de una interfaz gráfica. También llamado actividad. las *Activities* se corresponden con pantallas de la aplicación.
- *Fragment*: Representa un comportamiento o un elemento de interfaz de usuario en una actividad. Puede combinar varios fragmentos en una sola actividad para construir una interfaz de usuario con varios paneles y reutilizar un fragmento en múltiples actividades. También conocido como fragmento.
- *Dialog*: Es una pequeña ventana que solicita al usuario tomar una decisión o introducir información adicional. No ocupa toda la pantalla y se utiliza normalmente para eventos modales que requieren que los usuarios tomen una acción antes de que puedan proceder. También conocido como cuadro de diálogo.

La tabla 4.1 muestra un diagrama de flujo de las actividades de la aplicación.

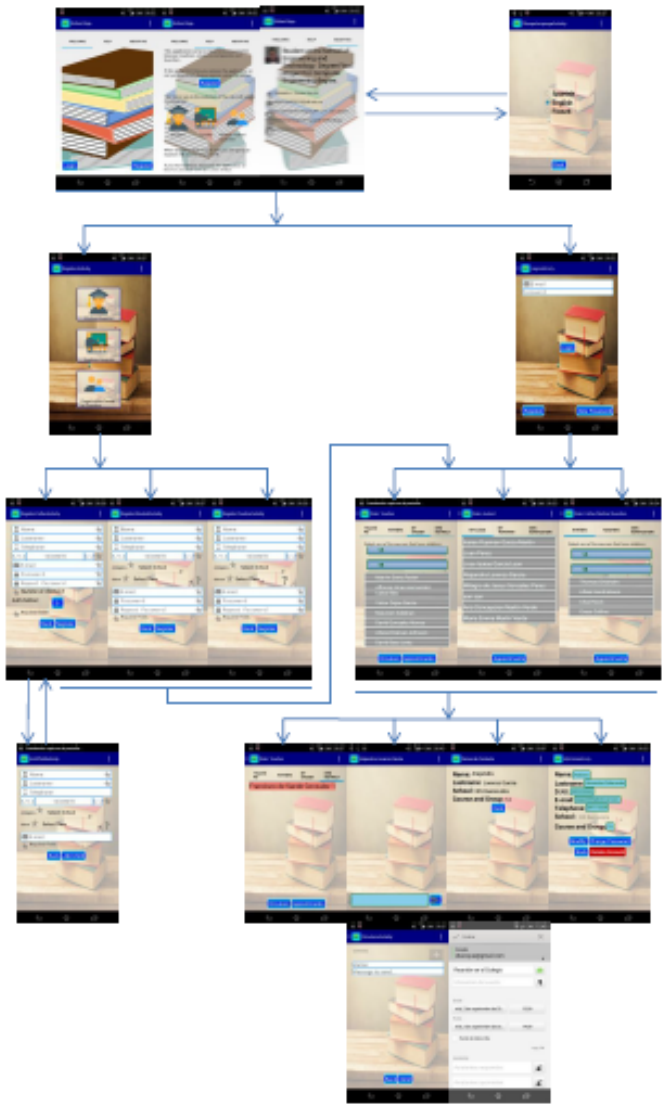

Figura 4.1: Navegación en las pantallas de `SchoolApp`.

4.1. Pantalla Principal (`MainActivity`)

Es la *activity* principal a la que accede el usuario cuando inicia la aplicación. Esta se compone de tres *fragments*.

4.1.1. Bienvenida (`WelcomeFragment`)

La pantalla que le da la bienvenida al usuario. Como se puede ver en la figura 4.2, contiene dos botones. Un botón que lleva a la pantalla de registro (ver sección 4.3) y otro que lleva a la pantalla de acceso a la aplicación (ver sección 4.4).

4.1.2. Ayuda (`HelpFragment`)

En la figura 4.3 se puede observar una ayuda orientada al uso de la aplicación. Los usuarios pueden consultar en esta pantalla la información básica sobre `SchoolApp`. El usuario puede deslizar la pantalla para ver toda la información.

4.1.3. Sobre mi (`AboutMeFragment`)

En `AboutMeFragment` se encontrará la información referente al autor de la aplicación.

4.2. Cambio de idioma (`ChangeLanguajeActivity`)

`ChangeLanguajeActivity` muestra los distintos idiomas disponibles de la aplicación. Al seleccionar uno de ellos y accionar el botón volver, quedará almacenado el idioma y las actividades lo cargarán al ser creadas. Los idiomas disponibles son español, inglés y francés, aunque con el mismo método se puede traducir la aplicación a cualquier otro idioma. Al seleccionar el idioma se cambia el lenguaje por defecto de la *máquina virtual de java* [25] y al crear las actividades obtiene el texto de la aplicación en el idioma seleccionado.

Figura 4.2: Pantalla de Bienvenida de la Aplicación.

4.3. Registro (`RegisterActivity`)

En la actividad de registro se pueden observar los perfiles con los que se puede registrar un usuario. Al seleccionar cualquiera de los tres botones, mostrará la pantalla de registro correspondiente. Los usuarios se pueden registrar como alumno (`StudentRegisterActivity`), padre (`FatherRegisterActivity`) o profesor (`TeacherRegisterActivity`).

Las pantallas de registro (figura 4.4) muestran un formulario que el usuario deberá rellenar. Presentan dos botones, uno que le devolverá

Figura 4.3: Pantalla de ayuda de la Aplicación.

a la pantalla de bienvenida (sección 4.1.1) y otro que completará el registro. La aplicación solo permitirá el registro si se han rellenado todos los campos obligatorios. Una vez hecho, al accionar el botón, la aplicación llevará a cabo el registro del usuario en la base de datos de *Firebase*. También se accederá de forma automática a la aplicación.

4.3.1. Añadir alumno (`AddChildActivity`)

Si el usuario se registra como padre, debe registrar al menos a un alumno como hijo (figura 4.5). Al introducir los datos del estudiante, la aplicación consultará en la base de datos si existe el alumno. Si no existe se añadirá la información y se asociará con el padre, mientras que si existe solo se le asociará. Tras el registro del alumno volverá a mostrar la pantalla de registro de padres (sección 4.3).

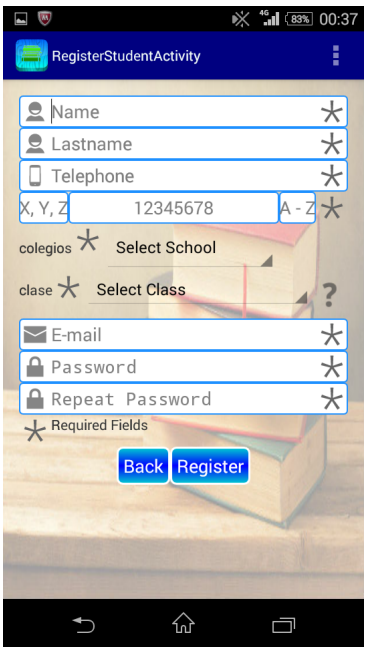

Figura 4.4: Pantalla de registro para el perfil de usuario.

4.4. Acceso (`LoginActivity`)

En `LoginActivity` (figura 4.6) se presentan dos campos en los que el usuario debe introducir su correo electrónico y contraseña para poder acceder a `SchoolApp` con el botón de acceso. Si selecciona el de registro, se mostrará la pantalla de selección de perfil (Véase 4.3). Para seleccionar *nueva contraseña* el usuario debe haber introducido previamente su correo electrónico. La aplicación se la solicitará al servidor de *Firebase* quien la enviará a la dirección con la que el usuario se ha registrado.

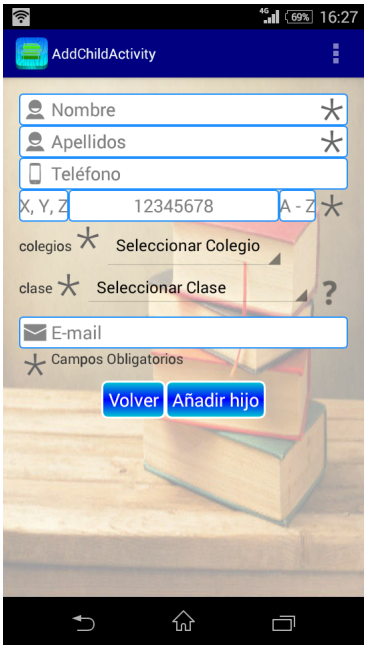

Figura 4.5: Registro de hijos.

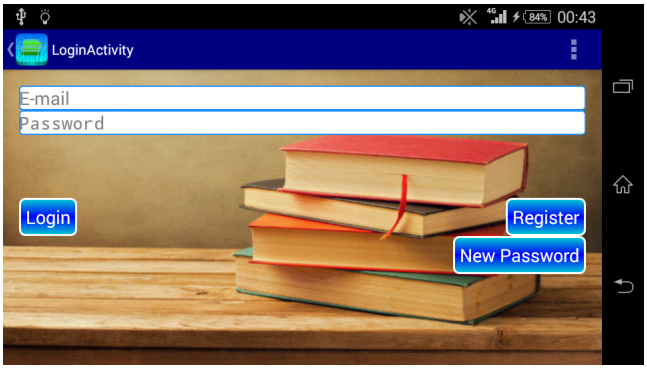

Figura 4.6: Pantalla de acceso con el dispositivo en horizontal.

4.5. Acceso de Usuarios

Al acceder a la aplicación, esta se encargará de consultar en la base de datos el rol del usuario. Si el acceso es correcto, se mostrará una pantalla con pestañas. Cada pestaña es una lista de contactos desplegable, excepto en el caso del alumno. También se mostrará una pestaña adicional para las notificaciones, es decir, mensajes que le lleguen al usuario (sección: 4.6). En las opciones puede seleccionar cambiar sus datos (sección 4.8) y salir de la aplicación. Al seleccionar el botón citas (sección 4.11) se mostrará la pantalla encargada de añadir un evento al calendario del dispositivo. Esta funcionalidad viene implementada por el *sistema operativo*.

La figura 4.7 presenta un ejemplo del acceso de un usuario con el rol de profesor.

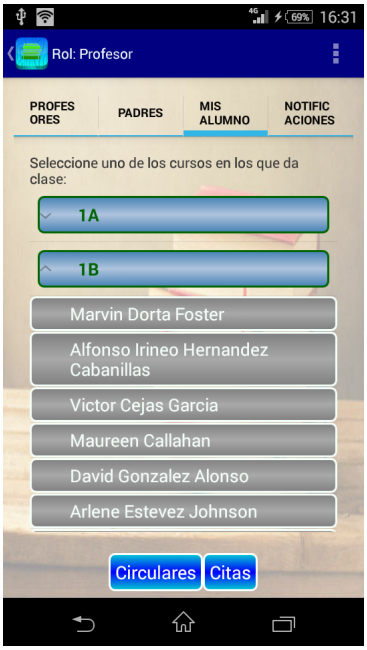

Figura 4.7: Lista de contacto de alumnos para el perfil de profesor.

4.6. Notificaciones (`notificationsActivity`)

En la figura 4.8 se muestran todas las notificaciones que reciben los usuarios. Éstas son recuperadas y eliminadas de la base de datos en *Friebase*. La notificación muestra el nombre del remitente y el número de mensajes que ha enviado y que el usuario tiene sin leer. Se clasifican por colores según el perfil del usuario que las envía. Al seleccionar cualquiera de ellas se podrá comunicar de forma directa con mensajes tipo chat (sección 4.7) con el remitente.

Figura 4.8: Notificaciones recibidas por los usuarios.

4.7. Chat (`ChatActivity`)

En esta pantalla obtenemos mensajería directa tipo chat con cualquiera de los contactos del usuario. Estos mensajes se almacenan en la base

de datos de *Firebase* de donde son recuperados y eliminados. También se guardan en una base de datos local, almacenada en el dispositivo. Si accedemos a las opciones podemos eliminar la conversación.

4.8. Mis Datos (`MyDataActivity`)

En la figura 4.9 se puede observar que el usuario puede ver los datos con los que se registró. Al seleccionar el botón modificar, podrá cambiar sus datos salvo el D.N.I. y el correo electrónico. Como un usuario conservará el mismo número en el documento nacional de identidad durante toda su vida, no se contempla que se pueda modificar. Con respecto al correo electrónico, es la clave con que el usuario se autentica contra el servidor, por lo cual no se podrá modificar para evitar inconsistencia de datos.

Si selecciona guardar se modificará la base de datos en el proveedor de servicios. Si selecciona dar de baja su cuenta, se borrará la información que hay en *Firebase*. También puede cambiar su contraseña con el botón destinado a ello (sección 4.9).

4.9. Cambio de Contraseña (`ChangePasswordactivi`

En la pantalla de cambio de contraseña el usuario podrá cambiar su contraseña introduciendo su correo electrónico y contraseña antigua. También deberá seleccionar una nueva contraseña y repetirla. Si desea hacer el cambio permanente solo deberá accionar el botón guardar.

4.10. Circulares (`CircularesActivity`)

En `CircularesActivity` se podrá enviar un mismo mensaje a una o varias clases, ya sean alumnos o padres. Al seleccionar el botón enviar, se enviará el mismo mensaje a cada uno de los alumnos que pertenezca a la clase seleccionada.

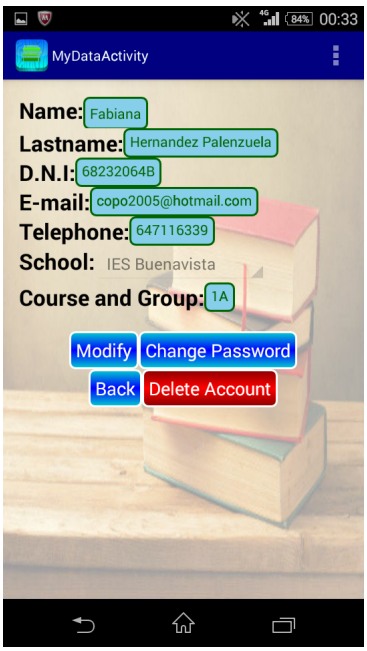

Figura 4.9: Datos del usuario.

4.11. Citas

Esta funcionalidad usa el calendario del dispositivo. Lanza la actividad que permite añadir eventos. Viene implementado en el *sistema operativo*. Se introducen los datos en los campos correspondientes y luego se selecciona guardar. Se puede elegir el calendario en el que almacenar el evento, en el dispositivo o en el de *Google*.

4.12. Datos del contacto (`DataActivity`)

En `DataActivity` se muestran los datos de cualquier contacto del usuario.

Capítulo 5

Desarrollo de la Aplicación

En este capítulo se expondrá la implementación de las clases más relevantes para la aplicación. Sería demasiado extenso explicar en profundidad cada uno de los elementos que componen la aplicación. La figura A.1 presenta un diagrama de clases de `SchoolApp` mientras que en el apéndice B muestra el diagrama de actividades(figura B.1).

Todo el código de `SchoolApp` [26] está disponible públicamente en el repositorio de github.

5.1. Clase `Student`

Esta es la clase encargada de almacenar los datos de los alumnos que se obtienen desde el *proveedor de servicios*. El constructor obtiene los datos desde un *HashMap* [27] que es el objeto que devuelven las consultas a la base de datos. También se ha implementado la interfaz *Parcelable* [28] que es la que permite compartir objetos de una clase entre *Activities*. En el listado 5.1 se puede observar la implementación de la clase.

Entre las líneas 19 y 25 se observa el método `writeToParcel()`, que implementa la forma de introducir datos de un objeto de la clase

```
1  //...
2  public class Student implements Parcelable {
3      private String name;
4      private String lastname;
5      private String school;
6      private String classroom;
7      private String mail;
8      private String telephone;
9      private String dni;
10     private String rol = "Alumno";
11     public Student(Map<String, Object> values) {
12         setName((String) values.get("nombre"));
13         setLastname((String) values.get("apellido"));
14         //...
15     }
16     //*****Parte de la interfaz Parcelable*****//
17     public Student(Parcel in) { readFromParcel(in); }
18     @Override
19     public int describeContents() { return 0; }
20     @Override
21     public void writeToParcel(Parcel dest, int flags) {
22         dest.writeString(name);
23         dest.writeString(lastname);
24         //...
25     }
26     public void readFromParcel(Parcel in) {
27         name = in.readString();
28         lastname = in.readString();
29         school = in.readString();
30         /*...En el mismo orden que writeToParcel...*/ }
31     public static final Parcelable.Creator CREATOR = new Parcelable.Creator () {
32         @Override
33         public Student createFromParcel (Parcel in) { return new Student(in); }
34         @Override
35         public Student[] newArray(int size) { return new Student[size]; }
36     };/*Parcelable.creator*/ }
```

Listado 5.1: Ejemplo de la clase Student.

Parcelable.

El método `readFromParcel()` (líneas de la 26 a la 30) muestra como se obtienen los datos de esta clase. En este método, los atributos deben estar en el mismo orden que el en `writeToParcel()`.

De la línea 31 a la 36, se se muestra como crear un objeto de la clase

Parcelable.

5.2. Clase `Father`

Esta es la clase encargada de almacenar los datos de los usuarios con el rol de padre que se obtienen desde la *nube*. Los atributos de la clase son: name, lastname, mail, telephone, dni, childrens (*ArrayList* [29] de la clase `Student` que son hijos del usuario) y rol. Funciona igual que la clase `Student` (sección 5.1).

Los métodos más importantes en esta clase son `getClassrooms()` que devuelve un ArrayList con las clases en las que están registrados los hijos y `getSchools()` que devuelve otro ArrayList con los colegios a los que asisten los hijos.

5.3. Clase `Teacher`

Esta es la clase encargada de almacenar los datos de los usuarios con el perfil de profesor que se obtienen desde la *nube*. Los atributos de la clase son name, lastname, mail, telephone, dni y rol. Funciona igual que la clase `Student` (sección 5.1).

El constructor obtiene los datos desde un *HashMap* [27] que es el objeto que devuelven las consultas a la base de datos. También se ha implementado la interfaz *Parcelable* [28] que es la que permite compartir objetos de una clase entre *Activities*.

5.4. Clase `Message`

Esta es la clase encargada de almacenar los datos de los mensajes que se obtienen desde la *nube*. A continuación se enumeran los atributos de la clase:

- *dniRemitter*: D.N.I del remitente del mensaje.

- *remitter*: Nombre del remitente del mensaje, se usa para mostrar el nombre en `NotificationsActivity` (sección 4.6).
- *message*: Mensaje que envían los usuarios.
- *rolRemmiter*: Perfil del remitente, se usa para colorear las notificaciones.
- *date*: fecha del mensaje.
- *destinatario*: Este campo solo está presente en la base de datos. Se usa para recuperar los mensajes.

Funciona igual que la clase `Student` (sección 5.1).

5.5. Clase `MessageSQLHelper`

Cada uno de los mensaje enviados entre los usuarios está almacenado en el dispositivo, en una base de datos local *SQLite*[30].

La tabla `messages` almacena los mensajes que se envían desde el dispositivo. Sus atributos son:

- *idConversation*: Identificación de la conversación.
- *dniRemitter*: D.N.I del remitente.
- *remitter*: Nombre del remitente.
- *message*: Mensaje.
- *day, month, year, hour, minute*: Datos del envío del mensaje.

La tabla `conversations` almacena el identificador de la conversación entre dos usuarios. Sus atributos son:

- *id*: Identificación de la conversación.
- *dniSender*: D.N.I del usuario que envía el mensaje.
- *dniRemitter*: D.N.I del remitente.

En el listado 5.2 se muestra la implementación de la clase.

```
1  package com.example.gonzalo.schoolapp.database;
2  //...
3  public class MessageSQLHelper extends SQLiteOpenHelper {
4  //...
5      @Override
6      public void onCreate(SQLiteDatabase db) {
7          String createMessagesTable = "CREATE TABLE messages ( " +
8                  "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
9                  "idConversation TEXT, " +
10                 "dniRemitter TEXT, " +
11                 "remitter TEXT, " +
12                 "day TEXT, " +
13                 "month TEXT, " +
14                 "year TEXT, " +
15                 "hour TEXT, " +
16                 "minutes TEXT, " +
17                 "message TEXT )";
18         String createConversationsTable = "CREATE TABLE conversations ( " +
19                 "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
20                 "dniSender TEXT, " +
21                 "dniRemitter TEXT )";
22         //Creando BBDD
23         db.execSQL(createMessagesTable);
24         db.execSQL(createConversationsTable);}
25     public void addMessage(Message message, String idConversation){/*...*/}
26     public void addConversation (String dniSender, String dniRemitter) {/*...*/}
27     public String getIdConversation (String dniRemitter) {/*...*/}
28     public List<Message> getAllMessagesConversation(String idConversation)
           {/*...*/}
29     public void deleteAllMessageConversation(String idConversation) {/*...*/}
30 }//class
```

Listado 5.2: Ejemplo de la clase MessageSQLHelper.

Desde la línea 6 hasta la 24, se muestra el método `onCreate()` donde se crean las las tablas descritas anteriormente. Para cada tabla se crea una variable de tipo cadena (*String*) con la definición cada uno de sus atributos. Después se ejecuta la función `execSQL()` y se le pasa como parámetro la variable creada.

5.6. TabActivityes

Cuando el usuario accede a `SchoolApp` se le muestra una pantalla con pestañas donde se hallan sus contactos. Cada pestaña es una actividad distinta a causa del problema 8. Al cambiar de pestaña se cambia de actividad y la consulta deja de ejecutarse. Esta solución ha sido la principal forma de construir la aplicación y solventar los problemas principales.

En el listado 5.3 se muestra como se crean las pestañas en las que se mostrarán las listas de contactos. En la línea 5 se obtiene el elemento donde se añadirán las pestañas, llamado `TabHost` [31]. Se crea el objeto que lanzará la actividad deseado (línea 10) y se añade al `TabHost`.

```
1  //...
2  public class StudentTabActivity extends TabActivity {
3          //...
4          //Annadiendo las Tabs
5          TabHost tabHost = getTabHost();
6          TabHost.TabSpec spec;
7          Intent intent;
8
9          //Student Tab
10         intent = new Intent().setClass(this, StudentActivity.class);
11         //...
12         spec = tabHost.newTabSpec(getString(R.string.tab_alumnos))
13                 .setIndicator(getString(R.string.tab_alumnos)).setContent(intent);
14         tabHost.addTab(spec);
15         //Otras tabs
16         tabHost.setCurrentTab(0);
17     }
18 //...
19 }
```

Listado 5.3: Ejemplo de la clase StudentTabActivity.

La clase `StudentActivity` (listado 5.4) muestra un ejemplo de la interacción con la base de datos. En la línea 8 se puede ver la consulta que obtiene los alumnos que asisten a la misma clase que el usuario. A esta consulta se le añade un oyente que conecta con la base de datos

y espera por si se modifican los datos. Se crea el objeto de la clase correspondiente en la línea 13 y se añade a un *ArrayList* con el que se rellenará la lista de contactos.

```
1  //...
2  public class StudentActivity extends ListActivity {
3      List<Student> alus;
4      //...
5      public void onCreate (Bundle savedInstanceBundle) {/*...*/}
6      public void preparingData(){
7          //Obtener los Student de la misma clase
8          Query getClassmates =
               aluRef.orderByChild(getString(R.string.bbdd_center)).equalTo(school);
9          getClassmates.addChildEventListener(new ChildEventListener() {
10             @Override
11             public void onChildAdded(DataSnapshot dataSnapshot, String s) {
12                 Map<String, Object> values = (Map<String, Object>)
                       dataSnapshot.getValue();
13                 Student alu = new Student(values);
14                 String dni = values.get(getString(R.string.bbdd_dni)).toString();
15                 if ((values.get(getString(R.string.bbdd_class)).equals(clase)) &&
                       (!alu.getMail().isEmpty()) &&
16                         (!dni.equals(myDNI))) {
17                     alus.add(alu); }//if
18                 //Seteamos el ArrayAdapter
19                 ArrayAdapter<Student> adapter = new
                       ArrayAdapter<Student>(StudentActivity.this,
20                         R.layout.list_item_layout, alus);
21                 setListAdapter(adapter);
22             }
23             @Override
24             public void onChildChanged(DataSnapshot dataSnapshot, String s) {}
25             @Override
26             public void onChildRemoved(DataSnapshot dataSnapshot) {}
27             @Override
28             public void onChildMoved(DataSnapshot dataSnapshot, String s) {}
29             @Override
30             public void onCancelled(FirebaseError firebaseError) {}
31         });}//function
32 /*...*/}//class
```

Listado 5.4: Ejemplo de la clase StudentActivity.

En listado 5.5 se muestra como se envía información a la base de datos. Al registrar al usuario se observa como existe una función que

comprueba que todos los datos del formulario sean correctos (línea 7, `haveEmptyFields()`). Para permitir el acceso a `SchoolApp` hay que crear al usuario como se muestra en la línea 8. Si se crea al usuario de forma satisfactoria, se procede a enviar la información a *Firebase*. Se crea un *HashMap* y se introducen los datos del usuario con el método `put()` (líneas 11 a 14). Se selecciona un identificador único (línea 15) y se envía a la base de datos. `studentRef` es la referencia a la tabla en la que se introducirán los datos, `child(uuid)` es la función que selecciona la fila correspondiente al usuario y `setValue(aluMap)` permite almacenar los datos que contiene el *HashMap* (línea 16).

```
1  //...
2  public class StudentRegisterActivity extends Activity {
3      //...
4      public boolean haveEmptyFields() {/*...*/}
5      public void submit (View view) {
6          //...
7          if (!haveEmptyFields()) {
8              rootRef.createUser(mail, password, new Firebase.ResultHandler() {
9                  @Override
10                 public void onSuccess() {
11                     Map<String, Object> aluMap = new HashMap<>();
12                     aluMap.put(getString(R.string.bbdd_name), name);
13                     aluMap.put(getString(R.string.bbdd_lastname), lastname);
14                     //...
15                     String uuid = UUID.randomUUID().toString();
16                       studentRef.child(uuid).setValue(aluMap);
17                     //Lanzando StudentTabActivity
18                     rootRef.authWithPassword(mail, password, new
                           Firebase.AuthResultHandler() {/*...*/}
19                 @Override
20                 public void onError(FirebaseError firebaseError) {/*..*/}
                       });//rootRef
21         }//if
22     }
23     //...
24 }
```

Listado 5.5: Ejemplo de la clase StudentRegisterActivity.

5.7. Problemas

Al diseñar una aplicación siempre surgen problemas inesperados en los que hay que agudizar el ingenio para resolverlos.

En este capítulo se expondrán los problemas ocurridos a la hora de programar, se irán enumerando y explicando la forma de resolverlos.

1. **Android Studio no render target**

   - **Solución**:
     a) Asegúrese de que hay un dispositivo real o virtual seleccionado.
     b) Tener una versión de *Android* seleccionada.
     c) Versiones necesarias para el desarrollo dela aplicación en *Android* instaladas.
     d) Crear un nuevo *Dispositivo Virtual* (*AVD*).
     e) Reiniciar Android Studio.

2. **Fallo al encontrar com.android.support:appcompat-v7:16.+**

   - **Solución**: Actualizar en las dependencias del archivo `build.gradle`, en el directorio `app` que está en la raíz del proyecto, la librería deberá ser la última versión disponible. Este cambio se realiza de forma manual.

3. **Fallo al encontrar Java**

   - **Solución**: Si java ya está instalado, averigüe el directorio. Una vez hecho esto necesita volver a establecer la variable de ámbito indicando la localización correcta. Seleccione `Iniciar` >`Equipo` >`Propiedades` >`Configuración Avanzada del Sistema`. Entonces abra la pestaña `Opciones Avanzadas` >`Variables de Entorno` y añada una nueva variable de sistema llamada `JAVA_HOME` que tenga como valor la dirección del directorio donde tenga instalado el *JDK*, por ejemplo, `C:\Program Files\Java\jdk1.7.0_21`.

4. **Duplicidad en las dependencias de los paquetes**

   - **Solución**: Si se tiene un error de compilación sobre archivos duplicados, se puede excluir esos ficheros añadiendo la directiva `packagingOptions` al archivo `build.gradle` que está en el directorio `app`. Como se muestra en el listado 5.6.

```
apply plugin: 'com.android.application'

android {
    //...

    defaultConfig {
        //...
    }
    buildTypes {
        release {
            //...
        }
    }
    packagingOptions {
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/LICENSE-FIREBASE.txt'
        exclude 'META-INF/NOTICE'
    }
}

dependencies {
    //...
}
```

Listado 5.6: Solución a la duplicidad en `build.gradle`.

5. **Fallo al encontrar android.support.v13**

   - **Solución**: Añadir en las dependencias del archivo `build.gradle`, en el directorio `app` que está en la raíz del proyecto, la orden `compile 'com.android.support:support-v13:21.+'` de forma manual. Como se puede analizar en el listado 5.7.

6. **Arregle Gradle (Fix Gradle)**

```
apply plugin: 'com.android.application'

android {
    //...

    defaultConfig {
        //...
    }
    buildTypes {
        release {
            //...
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:21.0.3'
    compile 'com.android.support:support-v13:21.0.3'
}
```

Listado 5.7: Solución al problema supportV13.

- **Solución**:
  - *a*) En el archivo `buil.gradle` que está en el directorio raíz del proyecto, añadir la dependencia de forma manual: `classpath 'com.android.tools.build:gradle:1.0.0'` . Listado 5.8.
  - *b*) En el archivo `buil.gradle`, que está en el directorio `app` en la raíz del proyecto, añadir dentro de `release` de forma manual, listado 5.9.

7. **SDK no encontrado**

- **Solución**: Si AndroidStudio no encuentra el *SDK* y está instalado seleccionar `Windows` >`Preferencias` >`Android` >`Localización SDK` y establecer el directorio donde se tiene instalado.

```
// Top-level build file where you can add configuration options common to all
    sub-projects/modules.

buildscript {
    repositories {
        //...
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.0.0'
    }
}

allprojects {
    repositories {
        //...
    }
}
```

Listado 5.8: Solución en `build.gradle`.

8. **Consultas a la espera de cambios en los datos**

   - **Solución**: Existen dos tipos de consultas en *Firebase*, las que devuelven un solo resultado (`addListenerForSingleValueEvent()`) y las que devuelven varios resultados (`addChildEventListener()`). Éstas últimas se quedan esperando por si se modifican datos de la base de datos. La solución sería remover el `listener` al final de su uso (`ref.removeEventListener(originalListener);`) o tener una sola consulta del segundo tipo por actividad.

```
apply plugin: 'com.android.application'

android {
    //...

    defaultConfig {
        //...
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    //...
}
```

Listado 5.9: Solución en `app/build.gradle`.

Capítulo 6

Conclusiones y Trabajos futuros

En este capítulo se revisarán los posibles trabajos por hacer en `SchoolApp`, ya que toda aplicación debe estar dispuesta a ser mejorada. También se hablará de las conclusiones y experiencias adquiridas a lo largo de la realización de este proyecto que ha sido especialmente enriquecedor.

6.1. Trabajos futuros

Se contemplarán casos de uso que no se han implementado a la hora de crear la aplicación. Algunos perfiles ordenados según su dificultad de desarrollo son:

- Padres con hijos en distintos centros: en algún momento, alumnos con el mismo padre, madre o tutor legal podrían asistir a centros escolares distintos.
- Usuarios con distintos perfiles en el mismo colegio, por ejemplo, un profesor que imparta clases en el colegio al que asiste su hijo.
- Profesores que trabajan en más de un centro escolar: En este caso se contemplará profesores que impartan clases en más de un

colegio.

También se contemplarán otras funcionalidades, estas están ordenadas en función de su dificultad de implementación:

- Búsqueda de contactos: El usuario podrá buscar un contacto concreto introduciendo su nombre en un cuadro de búsqueda.
- Envío de boletines de notas: Proporcionará a los profesores la funcionalidad de enviar las notas de sus alumnos a los padres de éstos.
- Envío de archivos a través de la aplicación: Permitirá a los usuarios compartir archivos.

6.2. Conclusiones

La realización de este proyecto nos ha permitido aplicar los conocimientos técnicos obtenidos durante los años de estudio, permitiendo la asimilación real de las competencias necesarias y la adquisición nuevos conocimientos. Esta experiencia da a conocer el grado de implicación que conlleva crear una aplicación totalmente funcional en un ámbito real. El análisis de otras aplicaciones similares permite concretar el estado del mercado con respecto al tipo de aplicación creada.

Programar una aplicación puede parecer sencillo, pero conlleva dedicación, tiempo, trabajo y esfuerzo.

Capítulo 7

Conclusions and Future works

This chapter reviews different ways to enhance `SchoolApp`, since every application must be willing to be improved. It will also discuss the conclusions and lessons learned along the realization of this project that has been especially rewarding.

7.1. Future works

Use cases that have not been implemented when creating the application behold. Some profiles are sorted according to their difficulty:

- Parents with children in different schools: At some point, students with the same parent or guardian attend different schools.
- Users with different profiles on the same school, for example, a school teacher in the school that your child attends.
- Teachers working in more than one school: In this case teachers to teach classes in more than one school will be contemplated.

Other features will also be envisaged, these are sorted according to their difficulty of implementation:

- Finding contacts.
- Sending report cards.
- Sending files through the application.

7.2. Conclusions

The realization of this project will apply the expertise gained during the years of study, allowing real assimilation of the necessary skills and acquire new knowledge. This experience discloses the level of involvement that involves creating a fully functional application in a real environment. The analysis of other similar applications can realize the state of the market for the type of application built.

An application program may seem simple, but it takes dedication, time and hard work.

Capítulo 8

Presupuesto

Este presupuesto está dirigido a empresas, principalmente a colegios que deseen adquirir `SchoolApp`. Se ha desglosado en el presupuesto la cantidad de horas de análisis y de implementación que se han empleado en cada funcionalidad de la aplicación.

Funcionalidad	Horas de Análisis	Horas de Implementación	Total Horas
Registro de Usuarios	10	30	40
Recuerdo de Contraseñas	5	10	15
Contactos	10	30	40
Chat	10	50	60
Notificaciones	15	30	45
Circulares	20	40	60
Citas	5	10	15
Modificación de Datos	6	10	16
Baja de Usuarios	10	15	25
		Total	316

Figura 8.1: Horas empleadas en el desarrollo de la aplicación

La cantidad de horas invertidas en la aplicación son **316 horas**. La tarifa de desarrollo establecida en este proyecto es de **3.5 euros la hora** lo cual conduce a un total de **1106 euros de honorarios**. A lo cual hay que agregarle la adquisición de dos dispositivos de gama media a **119 euros cada uno** para pruebas con la aplicación.

Dicho esto, la aplicación tiene un coste de **1334 euros**.

Apéndice A

Diagrama de Clases

La figura A.1 presenta el diagrama de clases correspondiente a `SchoolApp`.

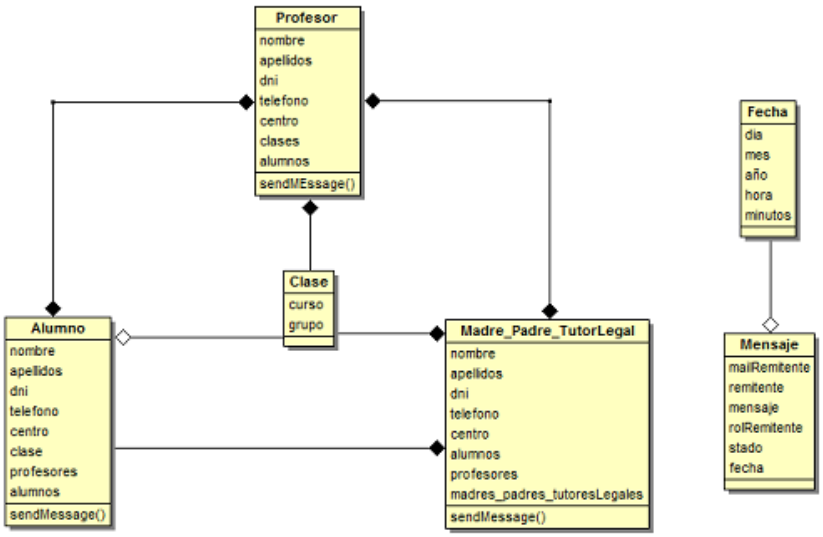

Figura A.1: Diagrama de Clases

Apéndice B

Diagrama de Actividades

La figura B.1 muestra el diagrama de actividades correspondiente a la aplicación.

Cada uno de los nodos es una actividad, representando las flechas el acceso a otras actividades. Las barras verticales representan que desde múltiples actividades se pueden acceder a una o varias.

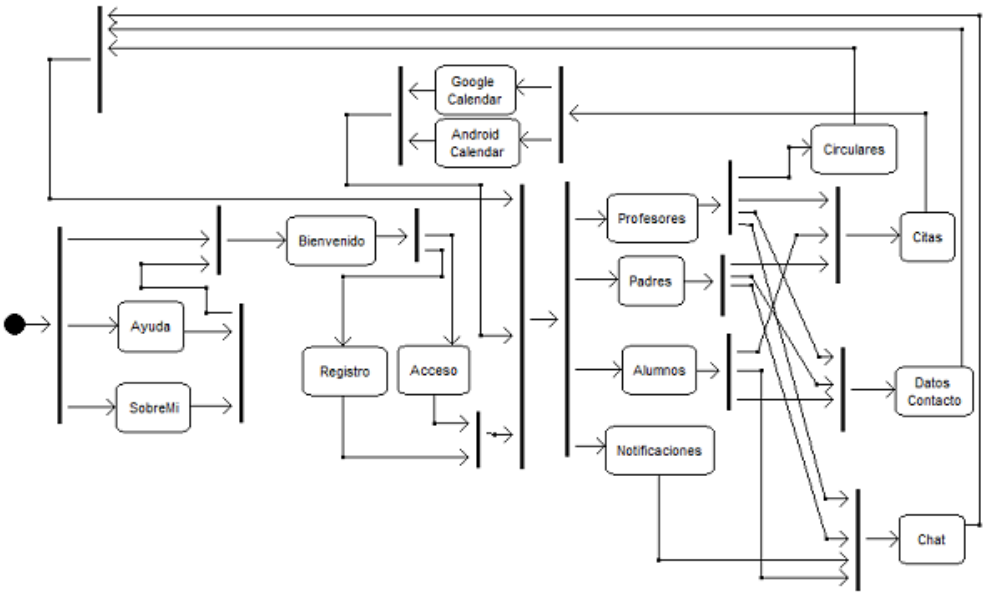

Figura B.1: Diagrama de Actividades

Bibliografía

[1] Parse, January. https://www.parse.com/ [Online; Último Acceso junio 2015].

[2] Google. Android, October 2007. https://en.wikipedia.org/wiki/Android_*operating_system* [Online; Último Acceso junio 2015].

[3] Computación en la nube, June. https://en.wikipedia.org/wiki/Cloud_computing [Online; Último Acceso junio 2015].

[4] github, September. https://github.com/ [Online; Último Acceso septiembre 2015].

[5] Git, April 2015. https://en.wikipedia.org/wiki/Git_(software) [Online; Último Acceso septiembre 2015].

[6] Latex, September. http://www.latex-project.org/ [Online; Último Acceso septiembre 2015].

[7] Remind101. Remind: Comunicación segura, September. https://play.google.com/store/apps/details?id=com.remind101 [Online; Último Acceso septiembre 2015].

[8] CreaTactil. micolegioapp, September. http://mc.creatactil.com/ [Online; Último Acceso septiembre 2015].

[9] Creatáctil, January. http://creatactil.com/ [Online; Último Acceso junio 2015].

[10] Android studio, May 2013. https://developer.android.com/sdk/index.html [Online; Último Acceso febrero 2015].

[11] JetBrains. Intellij idea, January 2015. https://www.jetbrains.com/idea/ [Online; Último Acceso septiembre 2015].

[12] Sun Microsystem. Web eclipse, November 2001. https://eclipse.org/ [Online; Último Acceso junio 2015].

[13] Netbeans, January. https://netbeans.org/ [Online; Último Acceso junio 2015].

[14] Firebase, December 2011. https://www.firebase.com/ [Online; Último Acceso junio 2015].

[15] JSON, December 2005. http://en.wikipedia.org/wiki/JSON [Online; Último Acceso junio 2015].

[16] Apple Inc. IOS, January. http://www.apple.com/es/ios/ [Online; Último Acceso junio 2015].

[17] Servicios rest, January. https://en.wikipedia.org/wiki/Representational [Online; Último Acceso junio 2015].

[18] HTTP, August 1996. https://en.wikipedia.org/wiki/Hypertext_Transfer [Online; Último Acceso agosto 2015].

[19] Soap, August. https://en.wikipedia.org/wiki/SOAP.

[20] Application programming interface, January. https://en.wikipedia.org/wiki/Application_programming_interface [Online; Último Acceso junio 2015].

[21] James Gosling. Java, January. https://en.wikipedia.org/wiki/Java_(programming_language) [Online; Último Acceso junio 2015].

[22] Java development kit, January. http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downl [Online; Último Acceso junio 2015].

[23] Developer Android. Training, January. https://developer.android.com/training/index.html [Online; Último Acceso enero 2015].

[24] Developer Android. Tabhost swipe, April. http://www.cs.dartmouth.edu/~campbell/cs65/lecture08/lecture08.htm [Online; Último Acceso abril 2015].

[25] Java virtual machine, August. https://en.wikipedia.org/wiki/Java_virtua [Online; Último Acceso agosto 2015].

[26] Gonzalo García Martín. Schoolaapp: repositorio en github, January. https://github.com/alu0100403619/TrabajoFinDeGrado [Online; Último Acceso enero 2015].

[27] Hashmap, September. http://docs.oracle.com/javase/7/docs/api/java/u [Online; Último Acceso septiembre 2015].

[28] Interface parcelable, January. http://developer.android.com/reference/an [Online; Último Acceso junio 2015].

[29] Arraylist, September. http://docs.oracle.com/javase/7/docs/api/java/ut [Online; Último Acceso septiembre 2015].

[30] SGoliver. Sqlite android, January 2015. http://www.sgoliver.net/blog/bases-de-datos-en-android-i-primeros- [Online; Último Acceso septiembre 2015].

[31] Tabhost, January. http://developer.android.com/reference/android/widg
[Online; Último Acceso junio 2015].