



UNIVERSIDADE DA CORUÑA

19/05/2017

# ARQUITECTURAS Y PLATAFORMAS MÓVILES

APLICACIÓN GRUPAL: CLASSROOMMANAGEMENT

DANIEL BARBEIRA HAYES  
ALEJANDRO FERNÁNDEZ GARCÍA  
DAVID SÁNCHEZ CANZOBRE  
MUEI. UDC.

## Índice de contenidos

Introducción .....	2
CEO (Arquitectura) .....	2
UX (Interfaz de usuario e interacción) .....	5
Geolocalización y sensórica.....	6
APIs de terceros .....	8
Realidad Aumentada .....	10
Errores conocidos y ampliaciones futuras .....	12
Problemas conocidos .....	12
Ampliaciones futuras.....	13

## Introducción

El proyecto de ClassroomManagement es una aplicación Android que facilita tanto a los profesores como a los alumnos asociados a una institución educativa (universidades, institutos, etc) el poder consultar información sobre aulas, facultades, etc. Se podrán consultar horarios, asignaturas impartidas y otra información relevante.

Para ello, se basa en el reconocimiento de marcadores para proporcionar dicha información. Un marcador esencialmente es un símbolo que se puede imprimir en papel y que contiene algún tipo de información cuando es reconocido por el software correspondiente. En el caso de esta aplicación se emplearán como marcadores códigos QR, que teóricamente se encontrarían en las puertas de las aulas y de las facultades.

La aplicación también hará uso de la geolocalización para mostrar la ubicación actual del usuario y la de los edificios de la institución académica sobre el mapa.

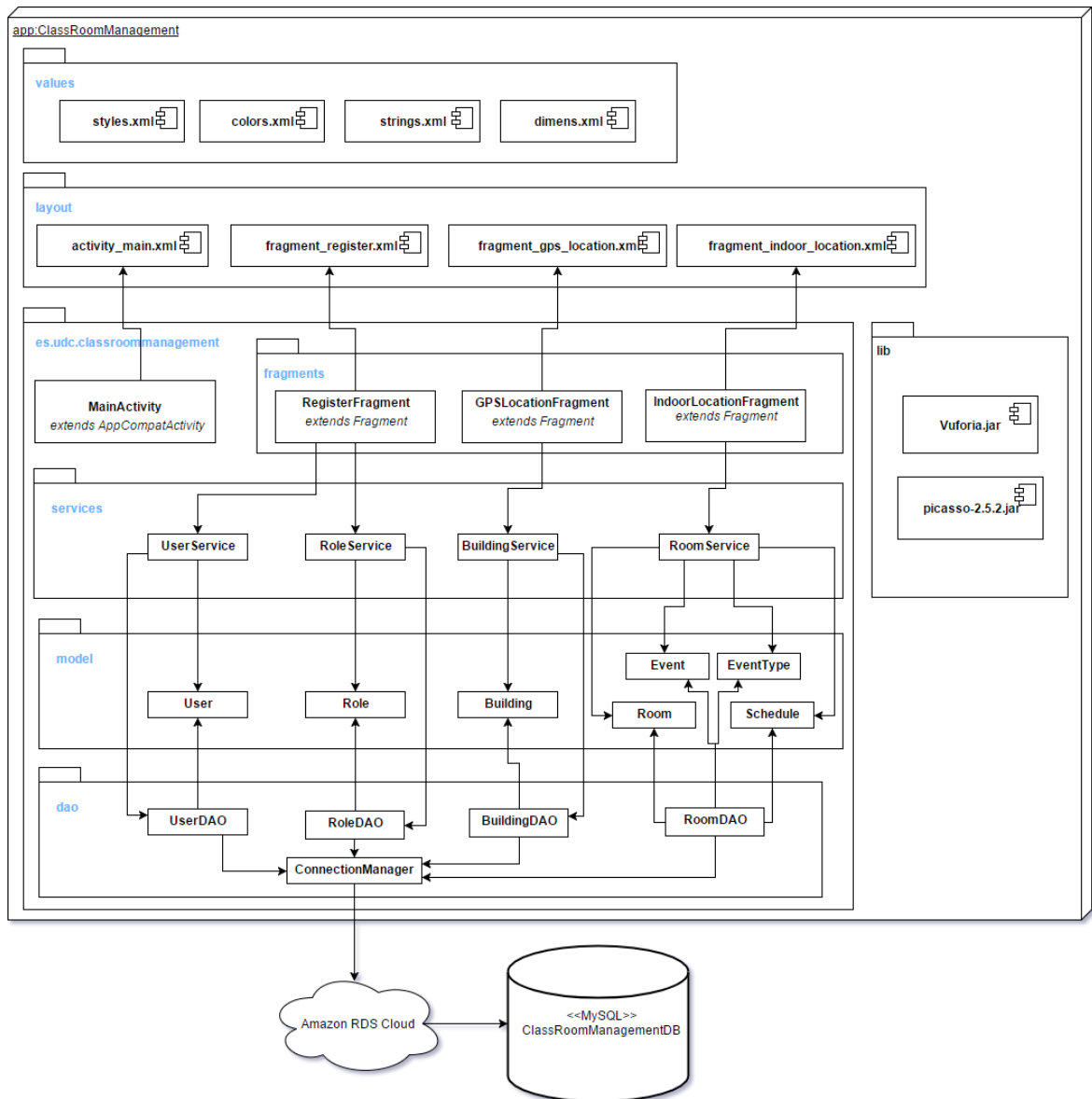
Además incluye el logueo con la cuenta de Google y una gestión de perfil de usuario, en el cual los usuarios podrán elegir el rol que ocupan dentro de la institución académica.

## CEO (Arquitectura)

A nivel de la arquitectura de la aplicación, se ha seguido una arquitectura multicapa estableciendo 4 capas claramente diferenciadas. La aplicación se dividirá en las siguientes capas:

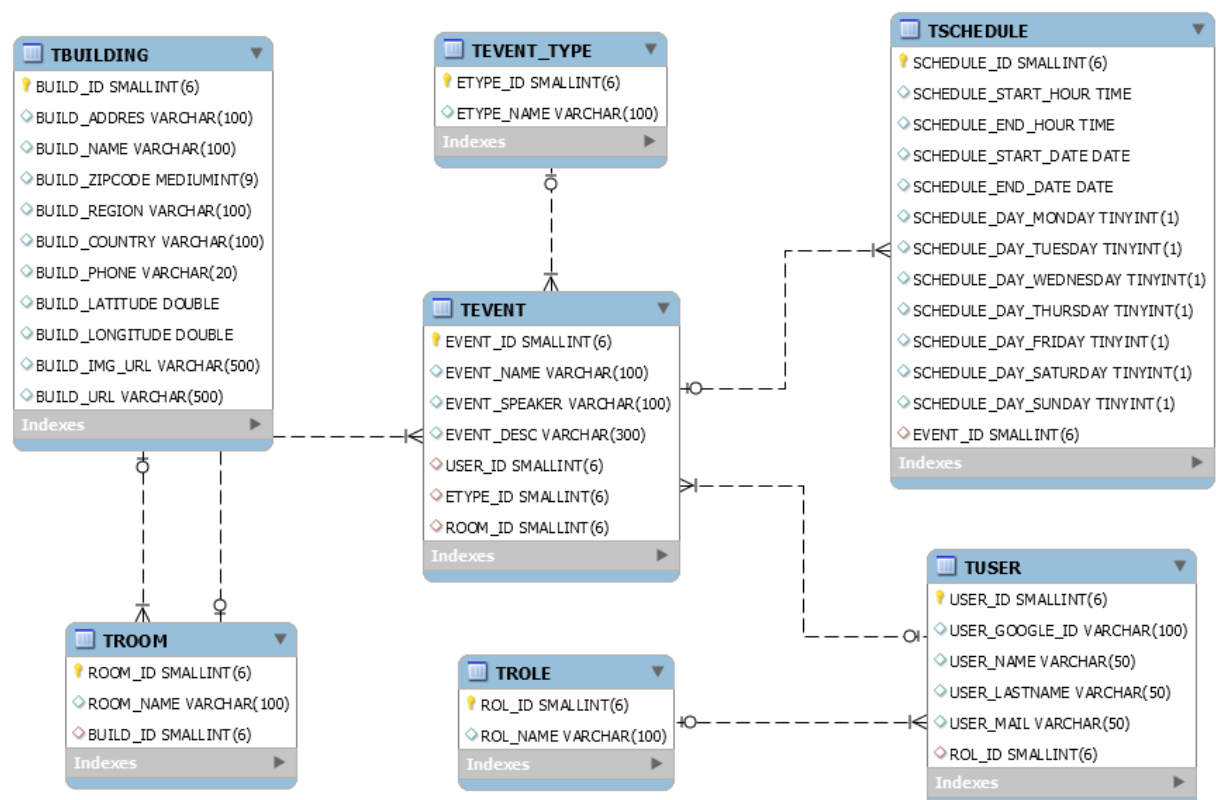
- Capa de modelo: Modela las entidades del dominio. Incluye las distintas entidades junto con sus atributos correspondientes.
- Capa de acceso a datos (DAO): Será la encargada de gestionar el acceso y modificación de los datos almacenados en la base de datos. De esta forma se consigue desacoplar el acceso a los datos persistentes del resto de la aplicación.
- Capa de servicios: Sirve como puente entre la capa de presentación y la capa de acceso a datos. Se procesan las peticiones desde la capa de presentación actuando en consecuencia para proporcionar la respuesta adecuada.
- Capa de presentación (fragments + activities): Modela la interfaz gráfica y presenta el sistema al usuario. Se encarga de gestionar las interacciones del usuario con la interfaz y de generar las respuestas a sus acciones.

De esta forma, tenemos el siguiente diagrama de arquitectura:



Del diagrama de arquitectura anterior cabe destacar el paquete Services. Para evitar que la pantalla se congele o que la aplicación se ralentice, se han implementado los servicios como `AsyncTasks`. Esto permite ejecutar las consultas de los DAOs en segundo plano de forma asíncrona, permitiendo a la aplicación seguir ejecutando su ciclo de vida.

Aparte del diseño y estructuración de la aplicación, se ha creado una base de datos relacional en MySQL que está desplegada en la nube en Amazon RDS (Relational Database Service). Este servicio permite acceder a través de una url, de la misma forma que si de una base de datos local se tratase. En esa base de datos, se ha creado un esquema de nombre homónimo a la aplicación acabado en DB, con las tablas que se exponen a continuación.



A continuación, se expone brevemente el significado de cada tabla y su uso:

- **TUSER:** Almacena los datos de los usuarios: ID de Google, nombre y apellidos y su cuenta de correo electrónico.
- **TROLE:** Contiene el ID de cada rol y su nombre.
- **TEVENT:** Modela los diferentes eventos que se celebran en una institución académica. Cada evento es de un tipo (clase, conferencia...), tiene un nombre de evento, un ponente, una descripción del evento y ocurre en un horario y sala determinadas.
- **TEVENT\_TYPE:** Almacena los tipos de evento definidos. Se incluyen clases, exámenes, conferencias y otros tipos de evento.
- **TSCHEDULE:** Modela los horarios de los eventos, las fechas y los días de la semana en los que se celebra dicho evento.
- **TROOM:** Tabla que contiene los nombres de las distintas salas en las que se pueden celebrar eventos. Pueden ser aulas, salones de actos, etc. Una sala pertenece a un determinado edificio.
- **TBUILDING:** Almacena toda la información relativa a los edificios de la institución académica. Se incluye la dirección, nombre, código postal, provincia, país, teléfono y la latitud y longitud que serán utilizados en la geolocalización. También se almacena una URL con la imagen del edificio en cuestión y una referencia a la web de la facultad/escuela correspondiente.

## UX (Interfaz de usuario e interacción)

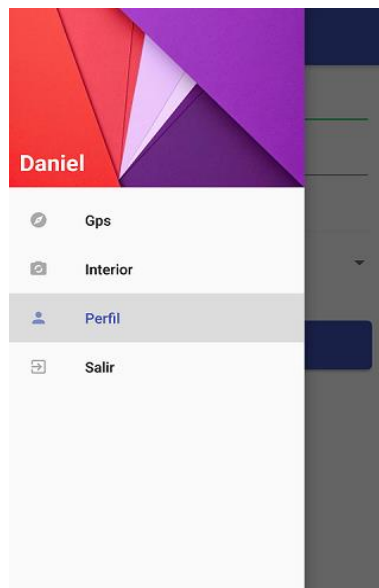
En cuanto a la experiencia de usuario de la aplicación, se ha trabajado en una línea de mantener la simplicidad de uso de la aplicación con textos simples y una navegabilidad fluida modelada mediante un menú lateral.

Se ha añadido contexto a la aplicación mediante algunos servicios de Google, como por ejemplo los servicios de geolocalización, que permiten interactuar con los mapas de Google, permitiendo además ver desde la aplicación la ubicación actual del usuario. También se ha incluido el logueo con la cuenta de Google, simplificando el proceso de registro de los usuarios, puesto que se utilizan los datos asociados a la cuenta de Google.

Otro punto importante es el almacenamiento de todos los datos de la aplicación, incluyendo el perfil de los usuarios en una base de datos externa en la nube, lo cual permite utilizar la aplicación con las mismas preferencias en múltiples dispositivos sin tener que repetir el proceso de registro múltiples veces.

A nivel de interfaz de usuario, se han aplicado estilos de Material Design manteniendo la compatibilidad con dispositivos con versiones de Android inferiores a la 5.0 (API 21). Para ello, se ha definido una paleta de colores personalizada utilizando la biblioteca de soporte v7 de Android, aplicando el tema `Theme.AppCompat.Light.DarkActionBar`. También se ha hecho uso de las elevaciones Z para incluir efectos de sombra en los dispositivos que sean compatibles. Los iconos del menú lateral también siguen estos estilos.

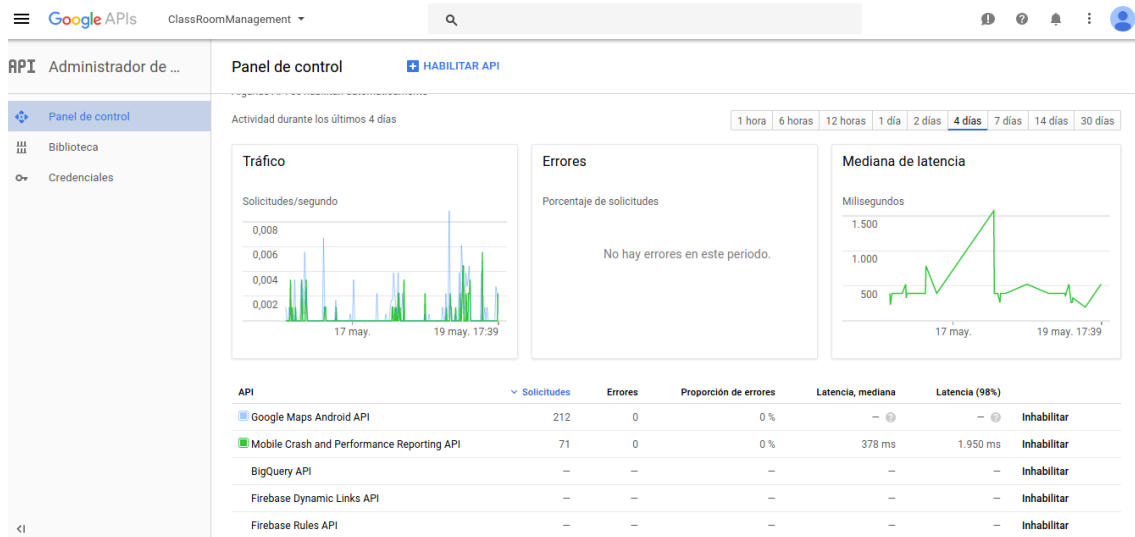
La navegación se ha definido y controlado mediante un menú lateral haciendo uso de la funcionalidad que ofrece una `NavigationView`. De esta forma se puede navegar de forma sencilla entre las distintas actividades con las distintas opciones definidas en el menú. También se muestra el nombre del usuario conectado:



## Geolocalización y sensórica

El Fragment encargado de la localización en exteriores, permite visualizar un mapa mostrando la ubicación actual y puntos de interés cercanos al usuario de los que se dispone de información.


Para poder utilizar el API de Google Maps, lo primero que hemos tenido que hacer es crear la clave de API en el Console Developer de Google para añadirlo en el manifest.xml de nuestra app. Una vez hecho esto tenemos que habilitar el API de Google Maps.



The screenshot shows the 'Clave de API' (API Key) page in the Google APIs console. It includes instructions on how to use the key, the creation date (22 feb. 2017 21:09:34), the creator (danibarbeira@gmail.com), and the generated key: AIzaSyDY0hnp1\_SU\_hv56F\_2g10FEMyWLeeXk8. The 'Nombre' (Name) field is set to 'Browser key (auto created by Google Service)'. There is a section for 'Restricción de clave' (Key restriction) with options: 'Ninguna' (selected), 'URLs de referencia HTTP (sitios web)', 'Direcciones IP (servidores web, tareas cron, etc.)', 'Aplicaciones para Android', and 'Aplicaciones para iOS'.

Una vez hecho esto último ya para poder utilizar el API de GoogleMaps, debemos editar el build.gradle de nuestra app y añadir en las dependencias esta línea: compile 'com.google.android.gms:play-services-maps:10.2.4'.

Este Fragment se encarga de las siguientes tareas:

- Mostrar el mapa. Sobre el mapa se colocan los botones de ajuste del zoom y el botón para centrar el mapa en nuestra última posición conocida. Inicialmente el mapa se muestra con un nivel zoom de 15 y está centrado en nuestra última posición conocida, en caso de no tener permisos de localización el mapa se iniciará centrado en las coordenadas de la FIC.
- Obtener de la base de datos las localizaciones de los edificios sobre los que tenemos información y dibujar un marcador sobre el mapa para cada localización.
- Si la aplicación tiene permisos de localización, muestra nuestra última localización (se habla de obtener nuestra última localización porque obtener la posición a través de un dispositivo de localización, como por ejemplo el GPS, no es una tarea inmediata sino que puede requerir de un cierto tiempo de espera y procesamiento, por lo que no tiene sentido hablar de localización actual) en el mapa, lo marca con el icono que se muestra en la imagen siguiente, y la actualiza a medida que nos desplazamos. 
- Si se pulsa sobre el marcador de un edificio se muestra un "popup" con la información relevante del edificio (dirección, imagen, teléfono, dirección web) y una pequeña fotografía que se obtiene de la URL almacenada en la BD que está asociada al edificio. Para mostrar la imagen obtenida de la red utilizamos la librería Picasso, esta librería nos resulta interesante utilizarla en nuestra app porque además de permitirnos mostrar imágenes a partir de una URL, las cachea de esta forma en las sucesivas veces que despleguemos el popup no se volverá a descargar la imagen ahorrando en el consumo de datos.



Referente a la sensórica si obviamos la cámara y el GPS el dominio de nuestra aplicación no da mucho juego para añadir más sensores que estos. Únicamente se ha añadido el uso del vibrador para notificar al usuario cuando se ha detectado un marcador QR.

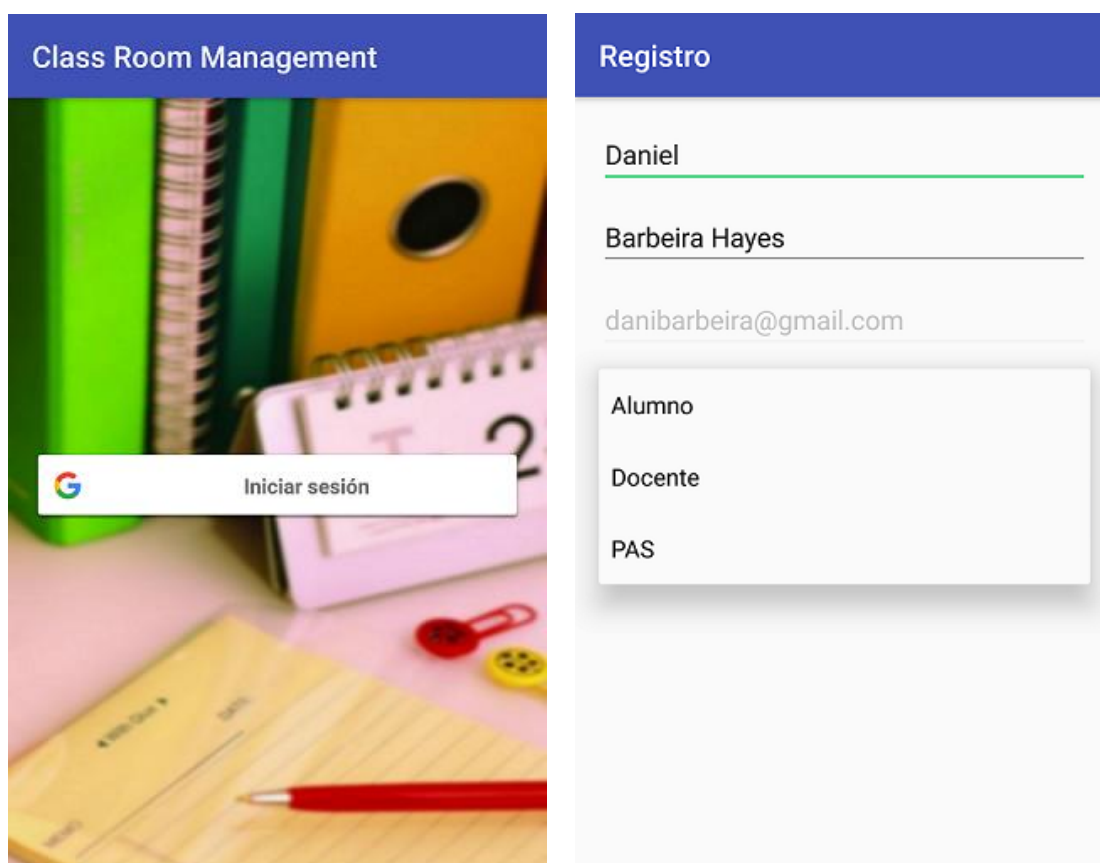


## APIs de terceros

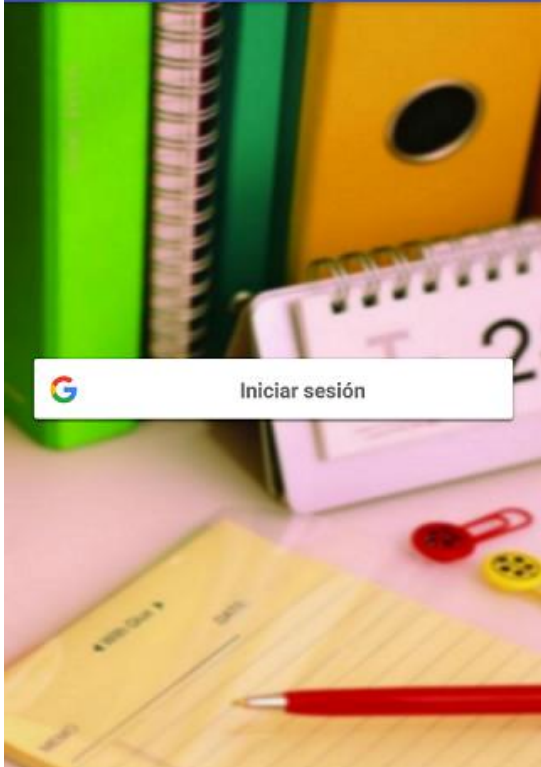
Las APIs con las que se ha trabajado desde la aplicación son las GoogleAPIs para el login y los mapas, y las APIs de [Vuforia](#) y [Picasso](#).


Para poder utilizar las APIs de Google, en primer lugar, hemos generado el fichero de configuración correspondiente, e incluido el fichero `google-services.json` correspondiente dentro del proyecto. Luego se han añadido las dependencias correspondientes para poder utilizar el plugin de Google Services (`com.google.android.gms:play-services-auth:9.8.0`) desde la aplicación y se han generado las claves correspondientes para poder utilizar sus APIs. Una vez incluida la clave de API en el manifest de la aplicación, esta queda habilitada para acceder a las APIs de Google. Para la parte de realidad aumentada, también es necesario el uso de API Keys para poder utilizar Vuforia, el proceso de obtención y asignación se explica en el siguiente apartado.

En la actividad implementada se incluye un botón para realizar el login, que permite seleccionar la cuenta de Google con la que se quiere autenticar. A continuación, se muestra un sencillo formulario de registro con los datos precargados que se obtienen de la cuenta de Google del usuario:



### Class Room Management



 Iniciar sesión

### Registro

Daniel

Barbeira Hayes

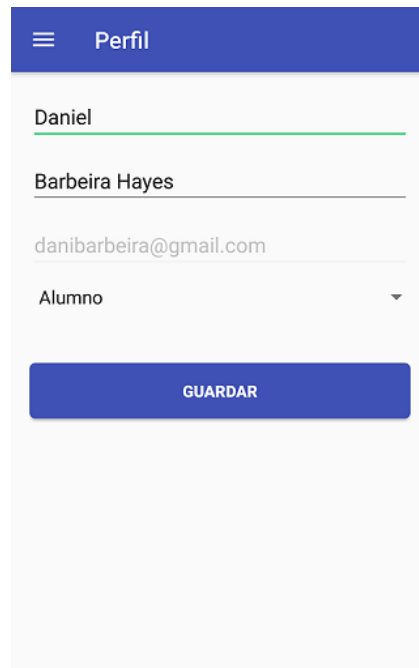
danibarbeira@gmail.com

Alumno

Docente

PAS

Además, la aplicación incluye una pantalla que permite visualizar o editar cierta información del perfil del usuario como: su nombre, sus apellidos y su rol dentro de la institución académica. Al pulsar en guardar se almacenan las modificaciones introducidas que se aplican de forma automática.



Con respecto a la API de Google Maps, utilizando la misma clave es necesario incluir la dependencia '*com.google.android.gms:play-services-maps:10.2.4*'. De esta API se ha utilizado la funcionalidad para integrar los mapas de Google dentro de la aplicación:



La API de Picasso se ha utilizado en la actividad de GPS para poder cargar las imágenes de los edificios de interés. Para poder incluirla en la aplicación, es necesario añadir la siguiente dependencia: '*com.squareup.picasso:picasso:2.5.2*'. También es necesario incluir la librería compilada en el directorio lib, añadiendo el archivo *picasso:2.5.2.jar*. Un ejemplo de visualización de la imagen de un edificio:



## Realidad Aumentada

Para guiado en interiores se ha utilizado Vuforia, con el que se podrán leer marcadores (en nuestro caso códigos QR) situados en puntos de interés (aulas, salas,...), que permitirán obtener información acerca de punto y el calendario de eventos de esa sala.

Lo primero que hay que realizar en esta parte, es seleccionar que tipos de marcadores se van a utilizar, nosotros hemos escogido códigos QR, básicamente por sencillez y popularidad. Los hemos generado utilizando la siguiente web [generador códigos QR](#).

Una vez disponemos de los marcadores, accedemos a la página oficial de [Vuforia](#) para subir los marcadores y generar una clave para nuestra aplicación. Una vez creado el proyecto y haciendo clic sobre él, podremos ver sus detalles, los cuales incluyen la clave que nos permitirá usar Vuforia en la aplicación.

License Manager
Target Manager

License Manager > ClassroomManageme...

## ClassRoomManagement

[Edit Name](#)
[Delete License Key](#)

License Key
Usage

Please copy the license key below into your app

```
AUE5oGH/////AAAAGWhq6TH+AkLgpAUBVOBChkNDB8QmC9daF/Ys
Sul01BHLapp0TgDMKbT/pZATLzs8FqtSP3K7dpMUnPgV/Djr9yvT
fRcblKbiW8euZPniSP5gRNtAXcSlEDe3mbWpaeDbKUwNQKbB6glM
dyVeI/ZawEF4XYtsi74ZKahIAy3TDC+8oVlnBUD8geeXkgq5mqAd
DlUEYFyE87aR2rf5yyePEJ6azwyg6VT2LDe8qng/TMcLyprmf1jE
X3iHMz3J3LnhOKRPoqaoCBU7UGOLuCzCFtxM2MgWWwhB13FWzcK5
TxTDL56IvxW+e8Pnp/4F6OYEW48A2Mcb1HnmZvgxAKeJ65B5c4U
XoL7K8/OA5LuT/in
```

**Type:** Develop  
**Status:** Active  
**Created:** May 07, 2017 19:53

En cuanto a los marcadores, una vez subidos y procesados, nos mostrará una lista en la que aparecerán todos los marcadores que hemos subidos con una puntuación. Esta puntuación indica la facilidad con la que Vuforia podrá detectar ese marcador, por lo que hay que tener una puntuación alta, para que los reconozca fácilmente. En nuestro caso, para todos los marcadores hemos obtenido la máxima puntuación.

License Manager
Target Manager

Target Manager > ApmMarks






## ApmMarks

[Edit Name](#)

**Type:** Device

Targets (5)

Add Target
Download Database (All)

<input type="checkbox"/>	Target Name	Type	Rating	Status	Date Modified
<input type="checkbox"/>	 ROOM_1	Single Image	★★★★★	Active	May 07, 2017 19:44
<input type="checkbox"/>	 ROOM_2	Single Image	★★★★★	Active	May 07, 2017 19:31
<input type="checkbox"/>	 ROOM_3	Single Image	★★★★★	Active	May 07, 2017 19:31
<input type="checkbox"/>	 ROOM_4	Single Image	★★★★★	Active	May 07, 2017 19:31
<input type="checkbox"/>	 ROOM_5	Single Image	★★★★★	Active	May 07, 2017 19:31

Desde esta última pantalla podremos descargar todos los marcadores post-procesados por Vuforia en un zip, para poder incluirlos en nuestro proyecto de Android Studio, de forma que luego podamos utilizarlos desde el propio framework para reconocer los marcadores y actuar de una u otra forma.

Partiendo de los ejemplos propuestos por Vuforia, se incluyen en nuestro proyecto los siguientes elementos:

- El sdk de Vuforia dentro de la carpeta "lib" en la raíz del proyecto.
- Los assets, es decir, nuestros marcadores post-procesados.
- Un par de clases que gestionan todo el comportamiento de Vuforia (iniciar, parar, pausar,...)

Una vez incluido todo esto, es necesario introducir nuestra clave de la aplicación dentro del motor de Vuforia, que en nuestro caso se haría en la clase `ApplicationSession` (utilizando el método `setInitParameters`), la cual expone métodos públicos para poder controlar el funcionamiento de Vuforia (iniciar, parar, pausar,...), además de almacenar el estado actual del mismo.

Acabada toda la configuración inicial, ya se puede comenzar a usar Vuforia. Haciendo uso de la clase mencionada anteriormente, se iniciaría Vuforia y todos sus componentes. Una vez iniciado Vuforia buscará en cada frame obtenido por la cámara, coincidencias con los marcadores que se le han indicado, si encuentra algo, el dispositivo vibrará y el State de Vuforia cambiará guardando una lista de los marcadores identificados en dicho frame. Comprobando este State, y viendo que marcador ha reconocido, se lanza una consulta a la base de datos para recuperar toda la información de la sala a la que pertenece el marcador. Una vez recuperada la información, se genera un popup que indica el edificio, nombre de la sala y su calendario de eventos que se muestra mientras el marcador está enfocado.

## Errores conocidos y ampliaciones futuras

En esta sección comentaremos los problemas que hemos detectado en nuestra app pero que están pendientes de ser solucionados además de las funcionalidades que aun faltan por implementar.

### Problemas conocidos

1. Las imágenes de los edificios que salen en los popups no se muestran la primera vez que se pulsa sobre el marcador, sin embargo, en las sucesivas pulsaciones si que se muestran.
2. En la pantalla de realidad aumentada no se muestra el menú lateral porque el layout de Vuforia queda por encima del layout del menú, por lo que no podemos ver el menú aunque en realidad si que se está desplegando. Esto imposibilita que desde esta pantalla podamos movernos a los otros Fragments y una vez dentro de este fragment, lo único que podemos hacer es salir de la aplicación pulsando el botón de volver atrás.

### Ampliaciones futuras

1. Crear un botón en Vuforia, que salga cuando se detecta un marcador, que nos permita abrir (en caso de tener el perfil de profesor) una activity en la que poder hacer reservas del aula en las horas que esté libre.
2. Integrar Google Calendar en nuestra app para gestionar los horarios.