



UNIVERSIDADE DA CORUÑA

Arquitecturas y Plataformas Móviles

Máster Universitario en Ingeniería Informática

Alejandro Fortes Lopes
Boris Caballero Lenza
Javier Rochela Calvo
Pablo Gómez Area

A Coruña, 19 de mayo de 2017

Índice

1. CEO	2
2. UX	3
3. Sensórica y geolocalización	4
3.1. Introducción	4
3.2. Metodos principales	4
4. APIs	7
4.1. Acceso a la aplicación	7
4.2. Obtención de información	8
4.3. Persistencia	8
5. Realidad aumentada	10
5.1. Marcadores	10
5.2. Funcionamiento	11

1. CEO

2. UX

3. Sensórica y geolocalización

En este apartado vamos a ver los elementos de sensórica utilizados en este proyecto, en este caso principalmente el uso de un sensor hardware que permite contar los pasos realizados.

3.1. Introducción

En versiones más antiguas de Android el conteo de pasos se realizaba leyendo directamente los datos del acelerómetro, que mide el cambio en la velocidad a lo largo de los tres ejes (X, Y, Z).

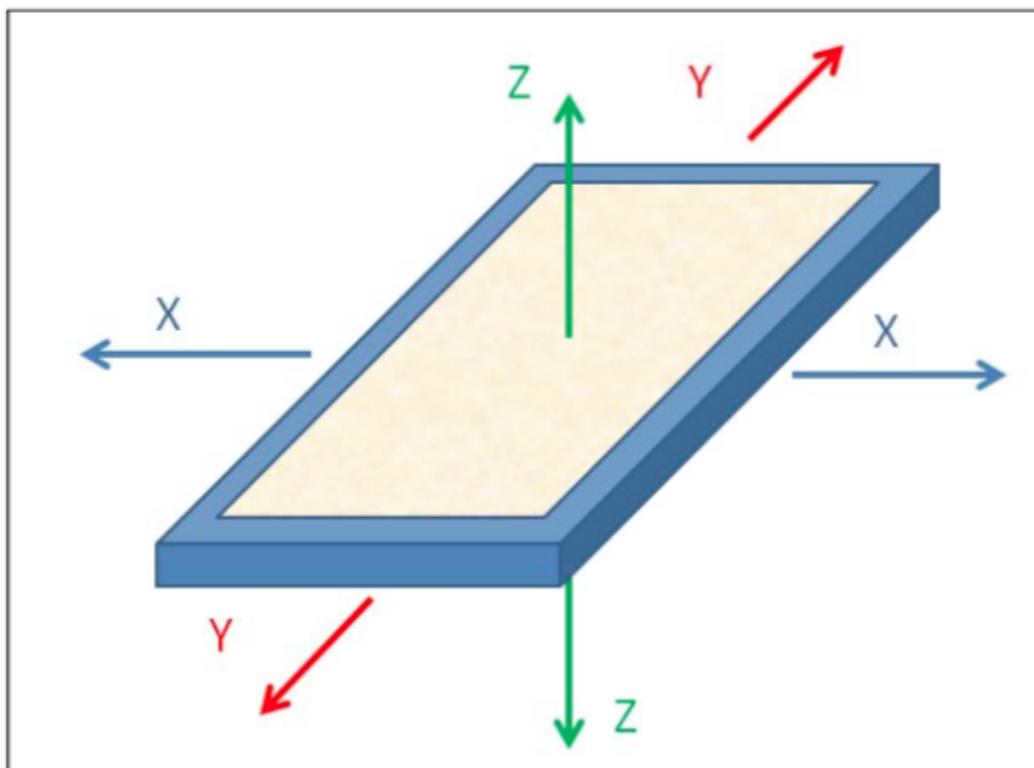


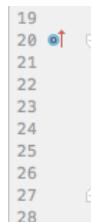
Figura 1: Ejes de coordenadas

Sin embargo, en este caso haremos uso de un sensor que nos proporcionan los dispositivos Android desde la versión 4.1, ya que consideramos que es una versión de Android relativamente antigua, y que abarcará a la inmensa mayoría de usuarios. Este es el sensor TYPE STEP COUNTER, ya que este activa un evento con cada paso detectado. También tendríamos la opción de usar TYPE STEP DETECTOR, pero los eventos TYPE STEP COUNTER se producen con una latencia mayor que aquellos que se generan desde TYPE STEP DETECTOR, eso se debe a que el algoritmo TYPE STEP COUNTER realiza un mayor procesamiento para eliminar los falsos positivos. Por lo tanto, TYPE STEP COUNTER puede ser más lento para generar los eventos, pero los resultados deben ser más precisos, con lo cual en este caso nos quedaremos con este último.

Una consideración muy importante a tener en cuenta, es que el TYPE STEP COUNTER solo se reinicia cuando se hace un reboot del dispositivo, así que deberemos tenerlo en cuenta para el correcto funcionamiento de la app, ya que debemos reiniciarlo "virtualmente" nosotros, para mostrar los pasos realizados desde la última apertura de la app.

3.2. Metodos principales

- **onCreate()**: En este método vamos a instanciar la clase SensorManager para poder obtener información y tener acceso a los sensores del sistema



```

19
20 ❶
21
22
23
24
25
26
27
28
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tv_steps = (TextView) findViewById(R.id.tv_steps);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    }

```

Figura 2: Método onCreate()

- **onResume()**: Si nuestra aplicación por alguna razón pasa por el método onPause, dejará de obtener los datos, por lo que si entraramos de nuevo no tendríamos respuesta, es por ello que hay que sustituir este método para volver a extraer la información.

Análogamente al metodo onCreate, volvemos a hacer uso de la clase SensorManager para recoger los sensores TYPE STEP COUNTER, y además en el caso de que se haya encontrado, registramos un listener el cuál acepta como parámetros el Context de nuestra Activity, el sensor que queremos registrar, si es que se ha encontrado, y una constante que recaudará los datos a una velocidad determinada. En este caso hemos escogido como constante de velocidad SENSOR DELAY IU.

En el caso de no encontrar el sensor que estamos utilizando, se lanzará un Toast de aviso.



```

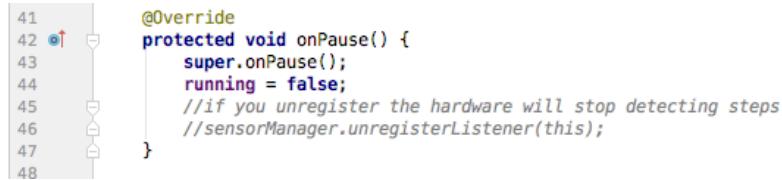
30 ❶
31
32
33
34
35
36
37
38
39
40
    protected void onResume() {
        super.onResume();
        running = true;
        Sensor countSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
        if(countSensor != null){
            sensorManager.registerListener(this, countSensor, SensorManagerSENSOR_DELAY_UI);
        } else {
            Toast.makeText(this, "Sensor not found", Toast.LENGTH_SHORT).show();
        }
    }

```

Figura 3: Método onResume()

- **onPause()**: En este método tendríamos dos opciones, o bien protegernos de un gasto desmesurado de la batería, que sería provocado al tener que salir de la aplicación. Para evitar esto dejaríamos de registrar el listener en el SensorManager cuando la actividad entrase en estado de pausa y evitar así que siguiera extrayendo datos del sensor.

Sin embargo hemos optado por no hacerlo, ya que consideramos que en el uso de nuestra aplicación no tendría mucho sentido dejar de registrar los pasos aún con la aplicación en estado de pausa. A pesar de esto, si quisieramos cambiar dicha configuración sería tan sencillo como descomentar la linea 46 de la siguiente imagen.



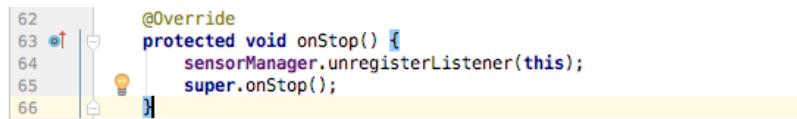
```

41
42     @Override
43     protected void onPause() {
44         super.onPause();
45         running = false;
46         //if you unregister the hardware will stop detecting steps
47         //sensorManager.unregisterListener(this);
48     }

```

Figura 4: Método onPause()

- **onStop()**: Al contrario de lo que hemos hecho en el método onPause(), en este método si quitamos el registro del listener en el SensorManager antes de cerrar la actividad, ya que una vez cerrada no tendría sentido seguir extrayendo datos del sensor, y nos crearía un serio problema de consumo de batería.



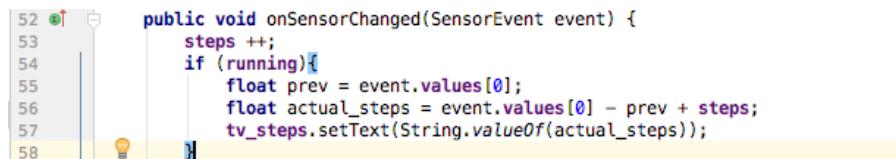
```

62
63     @Override
64     protected void onStop() {
65         sensorManager.unregisterListener(this);
66         super.onStop();

```

Figura 5: Método onStop()

- **onSensorChanged()**: Este método se ejecutará siempre y cuando cambie algún valor del sensor. En este caso, primero comprobamos si la app está ejecutándose, y si es el caso se muestra el valor en pantalla.



```

52     public void onSensorChanged(SensorEvent event) {
53         steps++;
54         if (running){
55             float prev = event.values[0];
56             float actual_steps = event.values[0] - prev + steps;
57             tv_steps.setText(String.valueOf(actual_steps));
58         }

```

Figura 6: Método onSensorChanged()

- **onAccuracyChanged:** este sería el método donde deberíamos implementar las acciones a realizar si la precisión del sensor cambia, en este caso no realizaremos ninguna ya que no lo consideramos necesario, pero es igualmente necesario declararlo.

3.3. Vistas dentro de la app

En la siguiente imagen vemos la vista del contador de pasos de la app.

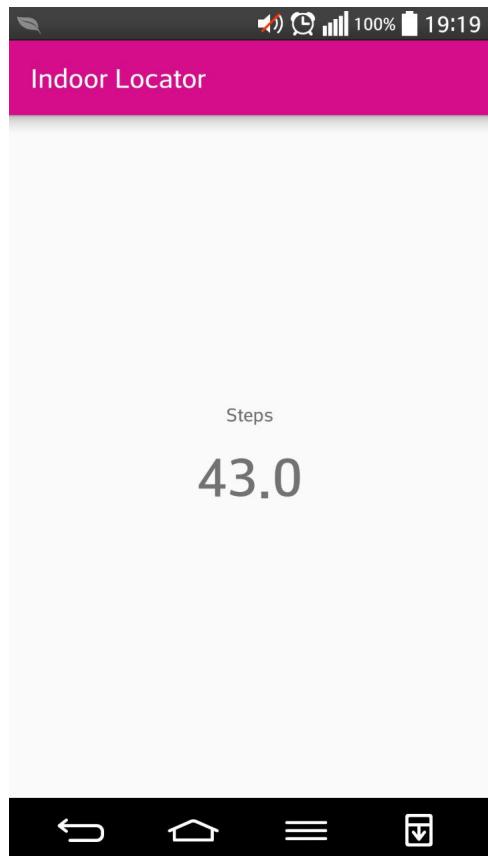


Figura 7: Vista de los pasos en la app

En el caso poco probable de que la versión de Android sea anterior a la 4.1, el sensor no será detectado.

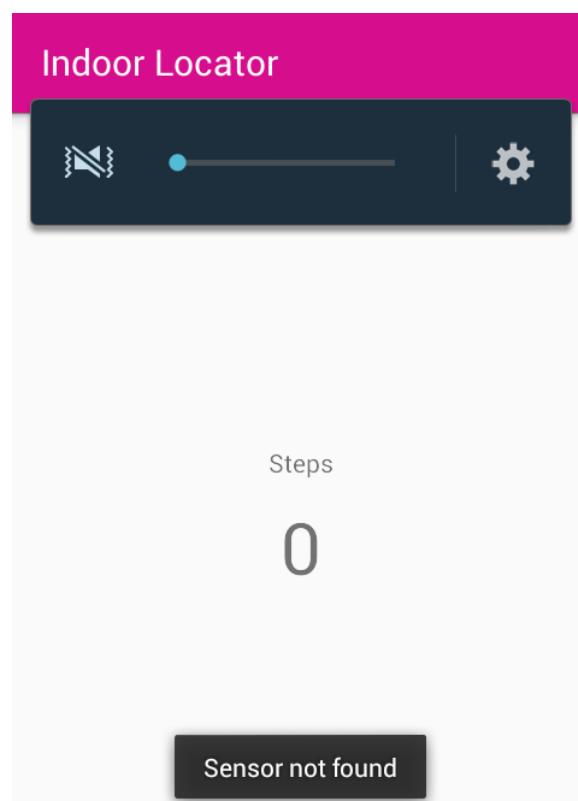


Figura 8: Sensor no encontrado

4. APIs

En este apartado vamos a ver el control de acceso de la aplicación, la obtención de información sobre los profesores y los despachos, y la persistencia de esta información.

4.1. Acceso a la aplicación

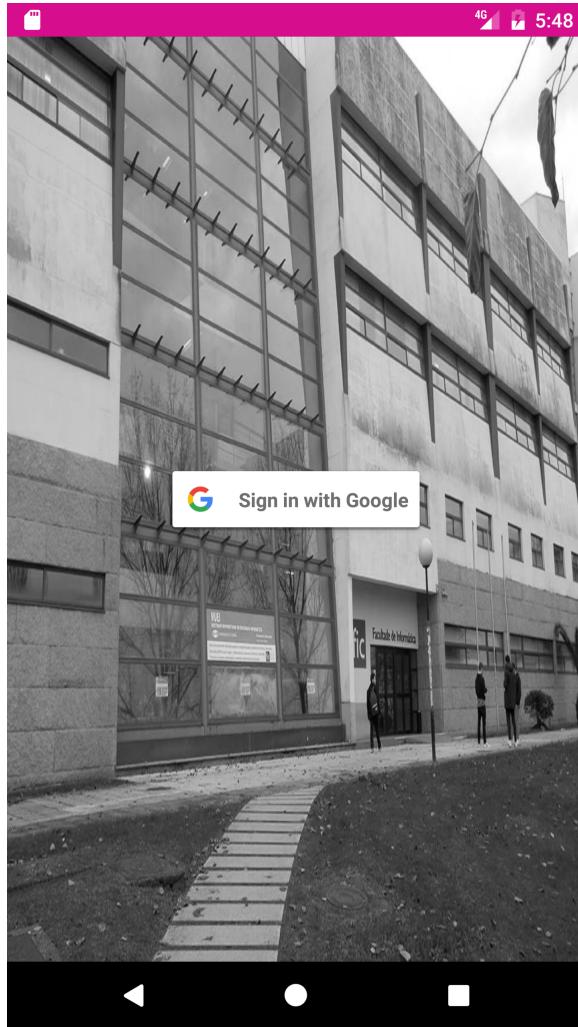


Figura 9: Login de la aplicación

Como se puede ver en la imagen superior hemos optado por utilizar el login de Google, para dotar a nuestra aplicación de autenticación. El flujo para la autenticación es el siguiente:

1. Primero se intenta acceder sin que el usuario tenga que realizar ninguna acción.
2. Si es posible, el usuario ya autenticado accede a la aplicación
3. En caso contrario, se le da la opción al usuario de acceder de forma manual, bien utilizando una cuenta de su dispositivo, o bien introduciendo una nueva dirección de correo y una clave. La api de Google también da la opción de crear una nueva cuenta.

4. Una vez autenticado el usuario accede a la aplicación.

4.2. Obtención de información

Para este punto hemos optado por crear nuestro propio servidor web REST dado que no existe un servicio web que nos permita disponer de esta información. Este servidor tiene implementados cinco métodos que permiten acceder a la información de diferentes profesores en formato JSON. Estos métodos son:

- GET «server»/api/rooms/: permite obtener una lista de las salas de la planta.
- GET «server»/api/rooms/{id}: permite obtener la información de una sala en concreto.
- GET «server»/api/teachers/: permite obtener una lista de todos los profesores.
- GET «server»/api/teachers/{name}: permite obtener la información de un profesor en concreto.
- GET «server»/api/all/: permite obtener toda la información disponible.

Estos métodos son invocados desde la aplicación mediante *AsyncTasks* con excepción de la última que se ejecuta thread ya que suponemos que en una aplicación un poco mas grande excedería la duración recomendada para una *AsyncTask*.

La clase *TeachersTask* es la encargada de hacer estas peticiones, y es utilizada desde las clases *TeacherFragment* y *OthersFragment* siempre y cuando el flag *teachersLoaded* (del que se hablará en el siguiente apartado) tenga valor negativo.

4.3. Persistencia

La aplicación permite sincronizar los datos obtenidos mediante el servicio web REST mencionado en el apartado anterior y almacenarlos en una base de datos para evitar futuras llamadas al servicio y reducir el consumo de datos de la aplicación. Esta opción esta disponible en el menú superior, bajo el nombre de Sincronizar. Esta opción solamente estará disponible si tenemos una conexión WIFI activa.

Al pulsar sobre esta opción se iniciará el proceso siguiente:

1. Se lanza en un nuevo thread, una petición al servicio web REST para obtener todos los datos.
2. Se procesa la respuesta de la petición y se convierte el JSON obtenido en objetos de tipo *Teacher*.
3. Dependiendo del valor de la variable *teachersLoaded* que indica si esta operación se ha llevado a cabo con anterioridad, se elimina el contenido actual de la base de datos.
4. A continuación se insertan en la base de datos la nueva información
5. Por último se modifica el valor de la variable *teachersLoaded* y se guarda en el archivo de preferencias compartidas de la aplicación.



Figura 10: Imagen de la opción de Sincronizar

A partir de este momento ya no se necesitará mas el servidor REST, dado que, en su lugar se harán las peticiones correspondientes a la base de datos.
En esta base de datos quedará almacenada una tabla *Teachers* que tendrá los siguientes atributos.

- **Id:** Identificador único de usuario. (generado automáticamente por la base de datos)
- **name:** Nombre.
- **department:** Departamento.
- **job:** Puesto de trabajo.
- **office:** Despacho del profesor.
- **extension:** Extensión telefónica.
- **email:** Dirección de correo electrónico.

5. Realidad aumentada

Para la realidad aumentada se ha utilizado Vuforia y Unity. Dentro de la aplicación, se muestran unas flechas con indicaciones para llegar a los despachos.

5.1. Marcadores

La aplicación dispone de 3 marcadores, uno al lado del ascensor en la planta 2 del edificio del área científica de la UDC, otro al lado del ascensor en la planta 3, y otro en la puerta del despacho 3.14 del área científica de la UDC.

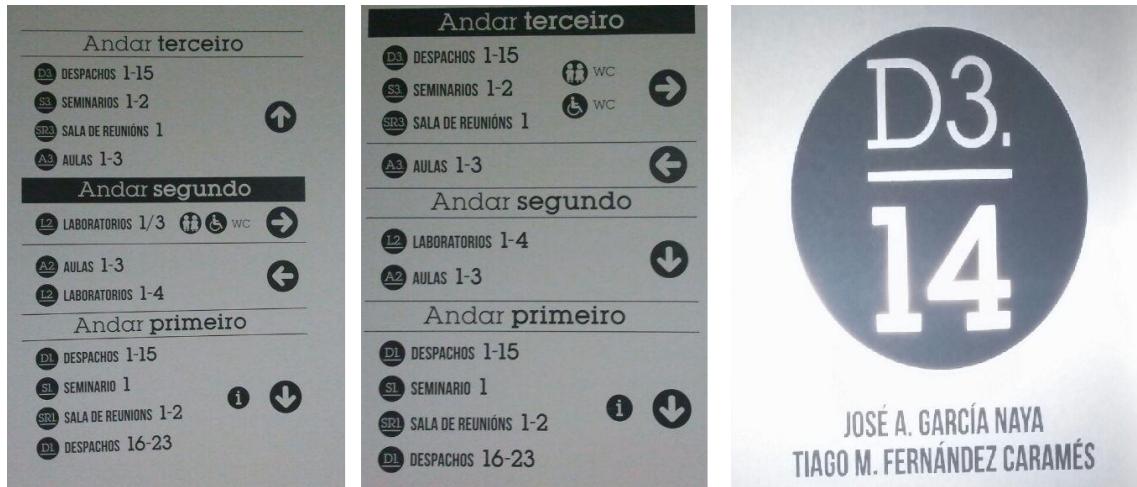


Figura 11: Marcadores utilizados

5.2. Funcionamiento

Aunque la idea es integrar la base de datos de despachos de la aplicación para dar indicaciones de cómo llegar a cada uno de ellos, por falta de tiempo solo se ha podido implementar las indicaciones para uno, el despacho 3.14 de la tercera planta.

Al pasar por la entrada de la segunda planta, se muestra una flecha que nos da la indicación de subir.



Figura 12: Indicaciones en la segunda planta

Ya en la tercera planta, al lado del ascensor, podemos ver el siguiente marcador, que da la indicación de seguir por el pasillo de la derecha.

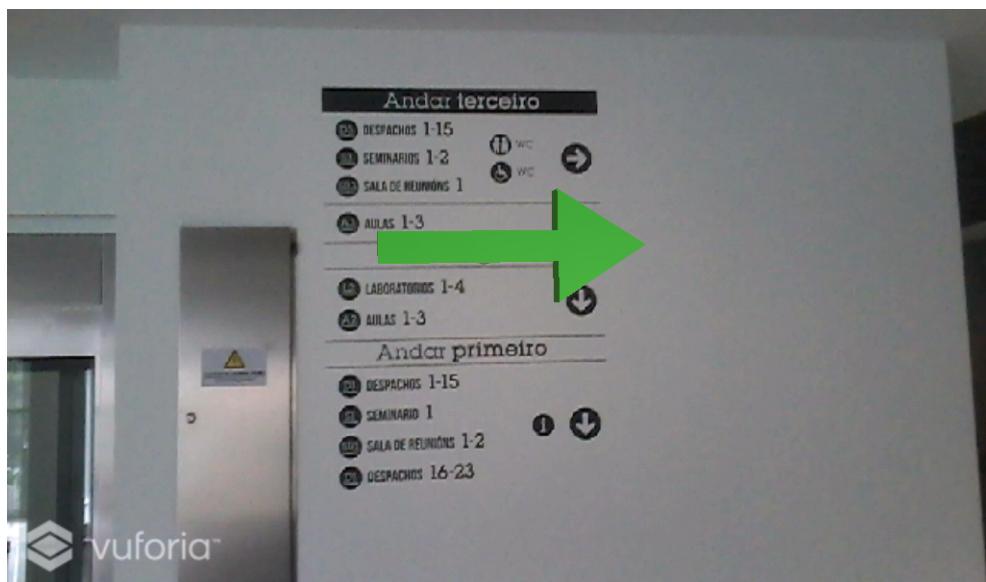


Figura 13: Indicaciones en la tercera planta

Siguiendo todo recto se llega al despacho, donde si se apunta a los nombres en la puerta del despacho, se despliega una fotografía del profesor con las horas de tutorías



Figura 14: Información al llegar al despacho