ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

SOFTWARE PARA PACIENTE DE ENFERMEDAD PULMONAR OBSTRUCTIVA CRÓNICA

SOFTWARE FOR CHRONIC OBSTRUCTIVE PULMONARY DISEASE PATIENTS

Realizado por
Diego Narbona Vílchez
Tutorizado por
José María Álvarez Palomo
Departamento
de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, julio 2016

Fecha defensa:
El Secretario del Tribunal

Resumen:
Desarrollo de una aplicación para Android con la intención de mejorar la adherencia del paciente al tratamiento de EPOC (Enfermedad Pulmonar Obstructiva Crónica) que su médico le asigne, así como su seguimiento por parte de los allegados al paciente. La aplicación cuenta con dos áreas:

- El área del personal sanitario la cual permite la personalización del tratamiento del paciente, así como la introducción y consultas de los análisis y espirometrías de éste.

- El área del paciente en la cual puede consultar mediante un código de colores en un semáforo el tratamiento que le ha sido asignado, posibilidad de llamar al 112, llamar al cuidador, consultar la medicación de rescate, leer consejos sobre ejercicios a realizar, consejos sobre la dieta, consultar estado de citas y visualización de tutoriales de uso de los inhaladores que tenga asignados.

Durante todo el proceso de la creación de la aplicación se ha seguido una metodología iterativa y además se ha contado con un cliente real que ha expresado sus necesidades y de las cuales se ha extraído los documentos de requisitos necesarios para la implementación de funcionalidades que contiene la aplicación. La aplicación ha sido desarrollada a partir de fuentes de información proporcionadas por personal sanitario especializado.

Palabras claves:
APP, Android, EPOC, Enfermedad, Pulmonar, Obstructiva, Crónica, Iterativa, Especializado, Salud

Summary:
The development of an application for Android which aims to help patients follow the treatment for Chronic obstructive pulmonary disease (COPD) prescribed by their doctor as well as allowing their next of kin the possibility of monitoring their evolution. The application has two areas:

- The doctors' area which allows them to customize the patient's treatment, as well as introducing and consulting the analysis and spirometries.

- The patient's area in which they are able to consult, using a traffic light colour code, the treatment they have been assigned, as well as the possibility of calling the emergency services, calling their assistant,checking the emergency medication required, reading the advice about physical exercise, checking advice about diet, consulting future appointments and viewing inhalator tutorials.

During the creation of this application a recurrent method has been used and we have taken into account the needs expressed by a real client from whom we have obtained the documents with the necessary requirements to implement the functions contained in the application.We have created the application based on information obtained from specialized sanitary staff.

Key words:
APP, Android, COPD, Disease, Pulmonary, Obstructive, Chronic, Recurrent, Specialized, Health

Índice

Introducción:

Motivación:

La aplicación surgió como respuesta al reto presentado por la Consejería de Salud y la Consejería de Empleo, Empresa y Comercio de la Junta de Andalucía, en colaboración con Vodafone, como ayuda al tratamiento de EPOC a través de una aplicación móvil.

Objetivos:

La intención de esta app es mejorar la adherencia del paciente al tratamiento de EPOC que su médico le ha asignado, así como su seguimiento por parte de los allegados al paciente.

Características:

Durante todo el proceso de la creación de la aplicación se ha seguido una metodología iterativa y además se ha contado con un cliente real que ha expresado sus necesidades y de las cuales se ha extraído los documentos de requisitos necesarios para la implementación de funcionalidades que contiene la aplicación.

La App ha sido desarrollada a partir de fuentes de información proporcionadas por personal sanitario especializado: toda la información incorporada está basada en datos y pautas reales acerca de EPOC.

Esta aplicación preserva la privacidad de la información registrada: se aplica la seguridad lógica en torno a los datos que están sometidos a la ley de protección de datos vigentes.

Tecnologías usadas:

- **Android [1]**: Es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes.

  Android se trata de un sistema abierto, multitarea, que permite a los desarrolladores acceder a las funcionalidades principales del dispositivo mediante aplicaciones, cualquier aplicación puede ser reemplazada libremente, además desarrollarlas por terceros, a través de herramientas proporcionadas por Google, y mediante los lenguajes de programación Java y C.

- **SQLite [2]**: SQLite es un sistema de gestión de bases de datos relacional compatible con ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Una de sus principales características es que la base de datos completa se encuentra en un solo archivo.

- **Java [3]:** Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo.

- **XML [4]**: XML, siglas en inglés de e**X**tensible **M**arkup **L**anguage ("lenguaje de marcas Extensible"), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Es la base de las interfaces gráficas en el sistema operativo Android.

- **SQLCipher [5]**: SQLCipher es una extensión open source de SQLite que proporciona cifrado AES-256-bit para archivos de base de datos. SQLCipher posee un gran rendimiento así que es ideal para base de datos embebidas e idóneo para entornos móviles.

- **GIT [6]**: Es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

  Entre las características más relevantes se encuentran:

  - Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal. Una presunción fundamental en Git es que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente, conforme se pasa entre varios programadores que lo revisan.

  - Gestión distribuida. Al igual que Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.

- **BitBucket [7]:** Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de versiones Mercurial y Git.

- **SourceTree [8]**: Aplicación de escritorio que permite trabajar de una manera rápida y visual con repositorios locales y remotos de GIT. Agiliza las operaciones ya que no hay que memorizar los comandos de GIT.

- **Android Studio [9]:** Android Studio es un entorno de desarrollo integrado para la plataforma Android.
  Características:
  - Renderización en tiempo real
  - Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
  - Soporte para construcción basada en Gradle.
  - Refactorización específica de Android.
  - Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, y otros problemas.
  - Plantillas para crear diseños comunes de Android y otros componentes.
  - Soporte para programar aplicaciones para Android Wear.

- **Slack [10]:** Slack es una herramienta de comunicación en equipo creada por Stewart Butterfield, Eric Costello, Cal Henderson, y Serguei Mourachov. Slack ofrece salas de chat organizadas por temas, así como grupos privados y mensajes directos. Posee un cuadro de búsqueda que permite acceder a todo el contenido de la aplicación. Slack integra una gran cantidad de servicios a terceros y respalda las integraciones hechas por la comunidad. Las principales incorporaciones incluyen servicios tales como Google Drive, Dropbox, Heroku, Crashlytics, GitHub y Zendesk. Algunas fuentes piensan que Slack es tan innovador que reemplazará a IRC algún día.

Estructura memoria:

El cuerpo principal de esta memoria se dividirá en dos partes, la primera parte corresponde a la aplicación desarrollada que se entregó en el reto y la segunda parte corresponde con la continuación y ampliación de funcionalidades que se le han realizado a la aplicación para cumplimentar las horas estipuladas. En la primera parte los compañeros de ingeniería de la salud junto con los que participamos en el reto se encargaron de realizar la parte de recogida de requisitos y confección de los casos de uso mientras que nosotros nos encargamos de la parte de modelado e implementación. En la segunda parte hemos realizado tanto las tareas de obtención de requisitos como de modelado e implementación.

En la primera parte se desarrolló las características base de la aplicación, sistemas para ver los manuales de inhaladores y fisioterapia, listado del tratamiento, creación de alarmas, desarrollo de la base de datos, protección por contraseña cifrada del área del personal sanitario, interfaz de la aplicación, CRUD de citas, permisos de la aplicación, carga en memoria de manera eficiente de las imágenes de la aplicación, patrón ViewHolder y activación/desactivación de los SMSs.

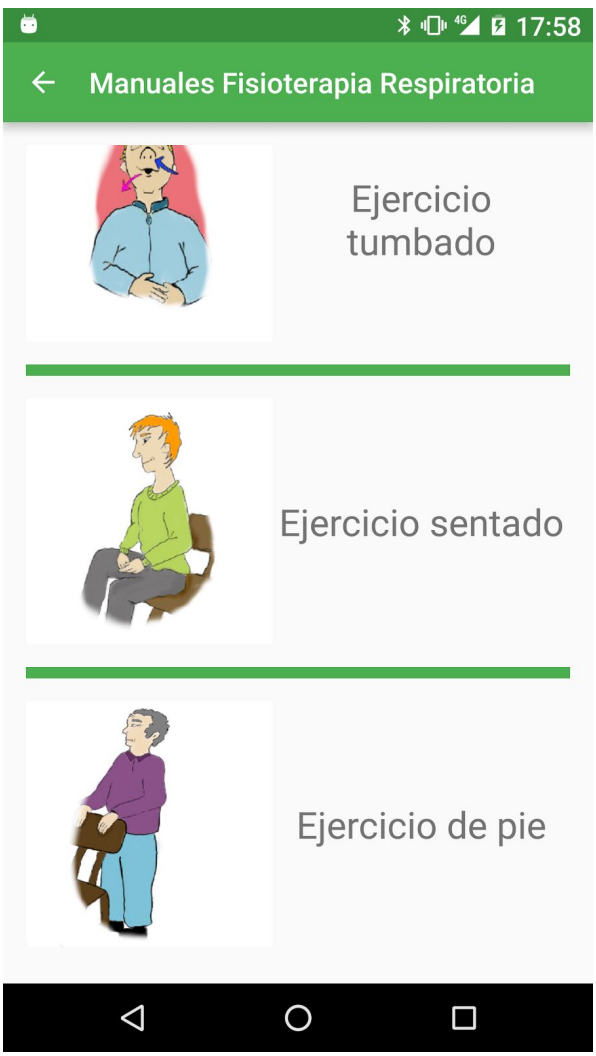
En la segunda parte se continuó el desarrollo de la aplicación, se realizaron fixes para la primera parte, se añadió un CRUD de análisis del paciente, se añadieron consejos para las dietas y se añadió el orden cronológico en los List Views.

1. 1ra Iteración

1.1 Sistema para ver los manuales de inhaladores y manuales de fisioterapia.

La aplicación posee un sistema dinámico que permite generar tutoriales de una manera rápida y sencilla, cada tutorial consistirá en una sucesión de pantallas por las cuales podremos navegar haciendo scroll horizontal o utilizando el menú de arriba a la derecha que nos indicará cuando estamos al principio del tutorial o al final.

Cada paso del tutorial contendrá, una imagen, un texto, un título y un botón que permitirá que el móvil reproduzca el texto de ese paso para aquellas personas que tienen dificultades al leer.



La clase encargada de realizar esta función es ScreenSlideActivity.java, para crear un tutorial de un determinado número de pasos simplemente deberemos llamar a la actividad de la siguiente manera:

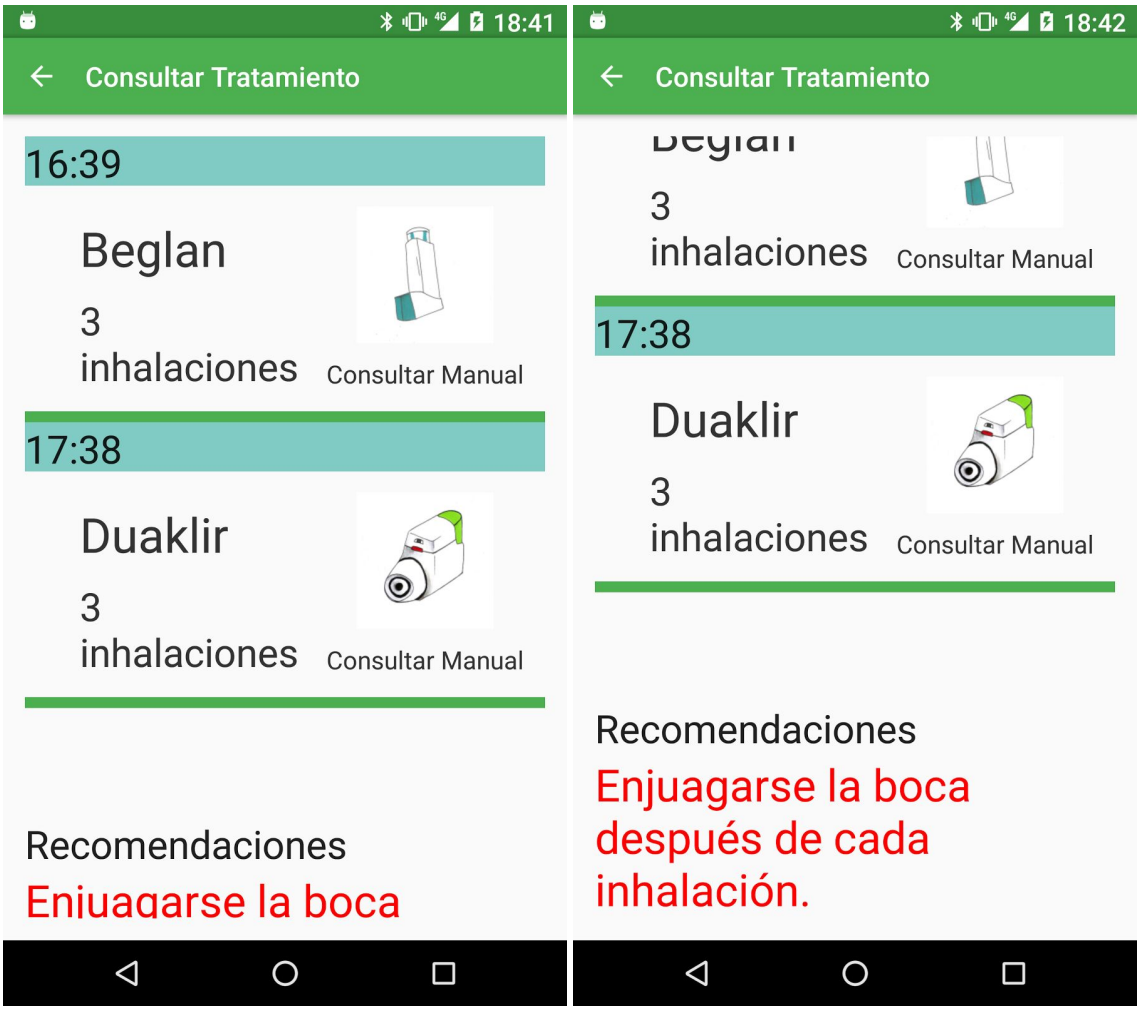
```
public void consultar(Inhalador inh) {
  ArrayList<String> titulos = new ArrayList<>();
  ArrayList<Integer> imagenes = new ArrayList<>();
  ArrayList<String> textos = new ArrayList<>();
  for(Paso p:inh.getListaPasosManual()){

    titulos.add(Commodities.retrieveResourceStringFromId(this,p.getTitulo()));
    imagenes.add(p.getImagen());
    textos.add(Commodities.retrieveResourceStringFromId(this,p.getTexto()));

  }

  Intent intent = new Intent(this, ScreenSlideActivity.class);
  Bundle b = new Bundle();

  b.putIntegerArrayList("imagenes", imagenes);
  b.putStringArrayList("texto", textos);
  b.putStringArrayList("titulos", titulos);

  intent.putExtras(b);
  startActivity(intent);

}
```

Al momento de crear la actividad hay que pasarle 3 ArrayList por un objeto Bundle los cuales contendrán los elementos que formarán cada paso del manual, un ArrayList contendrá los títulos de los pasos, otro ArrayList contendrá el identificador de las imágenes y otro ArrayList contendrá el texto explicativo de ese paso, el número de pasos que contendrá el manual quedará determinado por el tamaño de la lista que contiene los identificadores de las imágenes.

1.2 Listado del tratamiento

Una de las principales funciones de la aplicación es la posibilidad de consultar una lista con el tratamiento personalizado que el médico ha asignado al paciente en consulta. En esta lista aparece la medicación que debe tomar el paciente agrupado por horas, el número de inhalaciones que debe de realizar, recomendaciones que el médico le ha realizado y una imagen del inhalador que corresponde con la medicación que debe de tomar.

Al pulsar sobre cada elemento de la lista nos redireccionará al manual del inhalador que debemos de usar en esa toma.



```
public ArrayList<Toma> obtenerListaTomas() {
  String columnas[] = {ModeloContract.TakeTable._ID,
ModeloContract.TakeTable.TREATMENT_ID,
        ModeloContract.TakeTable.MEDICINE_NAME,
ModeloContract.TakeTable.ACTIVEINGREDIENT_NAME,
        ModeloContract.TakeTable.INHALER_NAME,
ModeloContract.TakeTable.DOSE_AI1,
        ModeloContract.TakeTable.DOSE_AI2,
        ModeloContract.TakeTable.HOUR,
        ModeloContract.TakeTable.MINUTE,
        ModeloContract.TakeTable.DONE,
        ModeloContract.TakeTable.INHALATIONS,
        ModeloContract.TakeTable.POSMED,
        ModeloContract.TakeTable.POSPRIN,
        ModeloContract.TakeTable.POSINH,
        ModeloContract.TakeTable.POSDOSIS,
        ModeloContract.TakeTable.POSINHALATIONS};
  ArrayList<Toma> lt = new ArrayList<>();
  Cursor crs = db.query(ModeloContract.TakeTable.TABLE_NAME, columnas,
null, null, null, null, null);
  while (crs.moveToNext()) {
     Toma t = new Toma(crs.getInt(0), crs.getString(2), crs.getString(3),
crs.getString(4),
          new Dosis(crs.getFloat(5), crs.getFloat(6)),
          new Hour(crs.getInt(7), crs.getInt(8)), crs.getInt(9) == 1, crs.getInt(10),
          crs.getInt(11), crs.getInt(12), crs.getInt(13), crs.getInt(14), crs.getInt(15));
     lt.add(t);
  }
  crs.close();
  return lt;
}
```

Cuando accedemos a la funcionalidad de consultar el tratamiento obtenemos de la base de datos lo datos de las tomas, que posteriormente tratamos para quedarnos solamente con los elementos relevantes para esta funcionalidad, nombre del medicamento, número de inhalaciones, hora de toma e imagen del inhalador.

```
for (Toma toma : ltoma){

  ArrayList<String> listaView = new ArrayList<>();

  Medicamento medForListView = modelo.getMedicineByName(toma.getMedicamento());
  listaView.add(medForListView.getNameText()); // MEDICINE.NAME
  Inhalador inhForListView = modelo.getInhalerByName(toma.getInhalador());
  listaView.add(String.valueOf(inhForListView.getId())); // INHALER._ID
  listaView.add(inhForListView.getNameText()); // INHALER.NAME_INHALER_XML
  listaView.add(inhForListView.getImagen()); // INHALER.IMAGE
  Hour h = toma.getHour();

listaView.add(String.valueOf(h.getHour())+":"+String.format(Locale.getDefault(),"%02d",
h.getMin())); // HOUR:MIN
  int numeroTomas = toma.getInhalations();
  listaView.add(String.valueOf(numeroTomas)); // NUMBER OF INHALATIONS

  lista_tomas.add(listaView);

}
```

Una vez tratado la lista de tomas y una vez obtenida la lista del tratamiento, el adaptador se encargará de mostrar la agrupación por horas del tratamiento.

```
if (position == 0) {
  showSeparator = true;
} else {
  if (!this.getItem(position - 1).get(4).equals(this.getItem(position).get(4))) {
     showSeparator = true;
  }else{
     showSeparator = false;
  }

if (showSeparator) {
  holder.divider.setText(getItem(position).get(4));
  holder.divider.setVisibility(View.VISIBLE);
} else {
  holder.divider.setVisibility(View.GONE);
}
```

Agrupación por horas:

El sistema para agrupar por horas que se ha utilizado es el siguiente:

1- Se ha ordenado por orden horario las tomas.
2- En el layout todas las filas de la lista contienen un separador que se muestra o se oculta según la siguiente lógica.
3- Si es la primera posición de la lista aparece el separador de la hora.
4- Si no es la primera posición se comprueba el elemento anterior de la lista, si es igual, no se añade el separador, si es distinto se añade el separador.

1.3 Creación de entidades de persistencia

Durante el desarrollo de la aplicación se ha mapeado las tablas de la base de datos en clases java para facilitar su utilización en el código.

En las imágenes que se encuentran en el anexo (Páginas 41,24) se puede observar la estructura general de la base de datos de la aplicación completa.

Entidades:

Treatment: El tratamiento que tendrá asignado el paciente, en el se encuentran las tomas que el paciente deberá tomarse. Las opciones de ampliación futuras son muy amplias ya que permitirá al paciente tener varios tratamientos según las enfermedades que tenga.

Take: La toma que el paciente debe suministrarse, tendrá asociado el inhalador, el medicamento y el principio activo asociado que el médico le ha indicado.

Inhaler: Representará los inhaladores que la aplicación tiene almacenados.

ActiveIngredient: Principios activos.

Medecine: Medicamentos.

Medecine_Presentation: Relación existente entre el medicamento y el principio activo, se utiliza para filtrar las listas desplegables en la selección del tratamiento.

UserManualStep: Representa un paso en un manual de usuario.

Analysis: Representará los datos de un análisis que el usuario se haya realizado.

Spirometries: Representará los datos de una espirometría que el usuario se haya realizado.

PatientPersonalData: Almacenará los datos personales del usuario.

Rescue: Medicación de rescate que el paciente deberá suministrarse en caso de crisis.

Appointment: Almacenará las citas médicas de las que disponga el usuario.

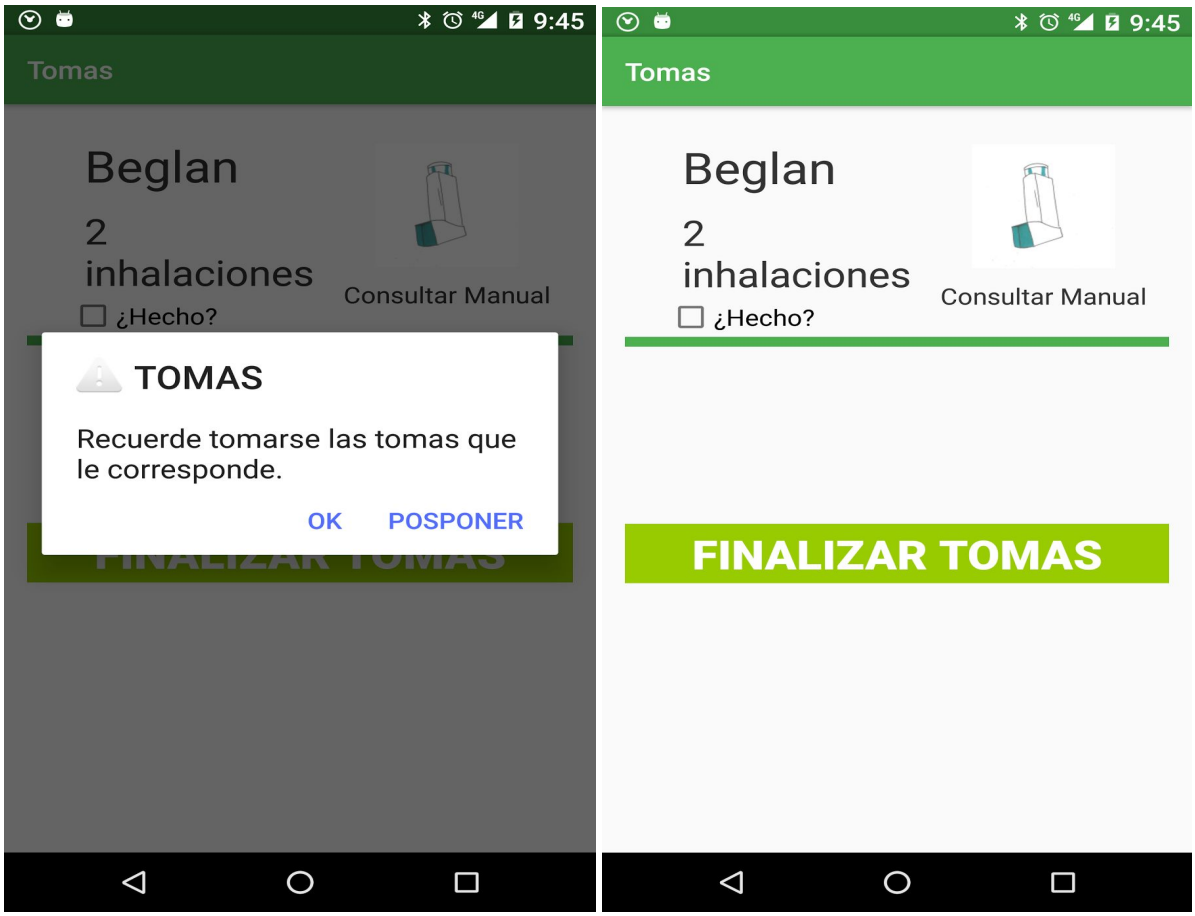
1.4 Creación alarmas:

La creación de alarmas en el momento de la creación de las tomas y el cambio de estado del semáforo han sido realizada en conjunto con mi compañero Sergio Sánchez Gil.

La aplicación dispone de un sistema de alarmas que permite al usuario recordarle cuándo debe tomarse su medicación, esta alarma se crea automáticamente en el momento que el médico crea las tomas.

Una vez la alarma suena, se lanza una actividad que en primer lugar pregunta si quiere realizar la toma o posponerla, si desea posponerla, pulsará en el botón “posponer” y el sistema moverá esa misma alarma 15 minutos más tarde(hasta un máximo de 3 veces), si desea realizarla simplemente se pulsa sobre el botón “ok”, si pulsa sobre el botón “ok” se creará la misma alarma al día siguiente.

Una vez marcado si queremos realizar la toma, aparece un listado con las tomas del tratamiento que el paciente se debe tomar a la hora que sonó la alarma, en esta lista aparece el medicamento que debe de tomar el paciente así como el número de inhalaciones que debe realizar,si el paciente pulsa sobre la imagen del inhalador será redirigido al manual correspondiente a este, una vez finalizada la toma simplemente el paciente debe marcar el checkbox correspondiente a las tomas que ha realizado y pulsar en finalizar.



La posibilidad de marcar mediante un checkbox si el paciente ha realizado o no la toma es de vital importancia para el seguimiento del tratamiento, ya que, alterará el estado del semáforo de seguimiento que se encuentra en el menú principal.

El semáforo dispondrá de 3 estados:

- Verde: Si el paciente ha realizado todas las tomas que le correspondía hoy.
- Ámbar: Si el paciente ha realizado algunas tomas que le correspondía hoy y además le faltan algunas por realizar.
- Rojo: Si el paciente no ha realizado ninguna toma que le correspondía hoy.

El estado del semáforo es reseteado todos los días a las 23:59, para ello, en el momento que se crean todas las alarmas también se crea la alarma de reseteo.

Otra de las características que han sido incluidas en la funcionalidad de las alarmas es la inclusión de una notificación sonora para alertar al usuario de que debe realizar la toma que le corresponde.

Implementación:

El elemento principal de las alarmas en Android son los receivers, un Broadcast Receiver es el componente que está destinado a recibir y responder ante eventos globales generados por el sistema, como un aviso de batería baja, un SMS recibido, un SMS enviado, una llamada, un

aviso de de la tajea SD, etc. y también a eventos producidos por otras aplicaciones como por ejemplo una alarma generado por una aplicación.

En esta aplicación se hace uso de 3 receivers.

1er Receiver: **AlarmReceiver**, este receiver será el encargado de recibir las alarmas de las que la aplicación genere.

2do Receiver: **ResetTakesReceiver**, este receiver será el encargado de recibir la actividad que resetea las tomas a las 23:59 de cada día.

3er Receiver: **AlarmOnBootReceiver**, este receiver será el encargado de recibir el evento del sistema que indica que el móvil ha terminado de iniciarse, se utiliza para refrescar el estado de todas las alarmas una vez se reinicia el móvil, hace falta añadir unas líneas en el archivo android manifest para poder realizar esta función.

```
<uses-permission android:name="android.alarm.permission.SET_ALARM" />
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

 <receiver
  android:name=".alarms.AlarmOnBootReceiver"
  android:enabled="true">
  <intent-filter>
     <action android:name="android.intent.action.BOOT_COMPLETED" />
  </intent-filter>
</receiver>
```

Android a grandes rasgos diferencia 2 tipos de alarmas, las alarmas exactas y las alarmas inexactas:

- Las alarmas exactas como su propio nombre indica sonarán en el momento exacto que le indiquemos, esto hace que el sistema operativo esté siempre esperando a dicha alarma y esto puede provocar problemas de batería si se abusa.

- Las alarmas inexactas, que pueden o no pueden sonar en el momento exacto que le indiquemos, esto es debido a que el sistema operativo intentará agrupar varias alarmas para que sean lanzadas a la vez y así evitar un consumo innecesario de batería.

En nuestra aplicación es necesario utilizar el sistema de alarmas exactas ya que no podemos depender del sistema para que el paciente pueda realizar el suministro de medicación que le corresponde.

```
Calendar alarm = Calendar.getInstance();
alarm.set(Calendar.HOUR_OF_DAY, hora);
alarm.set(Calendar.MINUTE, min);
alarm.set(Calendar.SECOND, 0);
alarm.add(Calendar.DAY_OF_MONTH, 1);

if (Build.VERSION.SDK_INT < Build.VERSION_CODES.KITKAT) {
  System.out.println("Alarm " + id + " is set at " + alarm.getTime());
  alarmManager.set(AlarmManager.RTC_WAKEUP, alarm.getTimeInMillis(),
pendingIntent);
} else {
  System.out.println("Alarm " + id + " is set at " + alarm.getTime());
  alarmManager.setExact(AlarmManager.RTC_WAKEUP, alarm.getTimeInMillis(),
pendingIntent);
}
```

En este ejemplo podemos observar la creación de una alarma, podemos diferenciar distintas partes:

1- Objeto calendar, utilizando este objeto podemos seleccionar la fecha y la hora que queremos que la alarma suene, una vez introducidos los datos simplemente deberemos llamar al método **alarm.getTimeInMillis()** que nos devolverá la hora señalada en el calendario en milisegundos (las alarmas funciona en milisegundos).

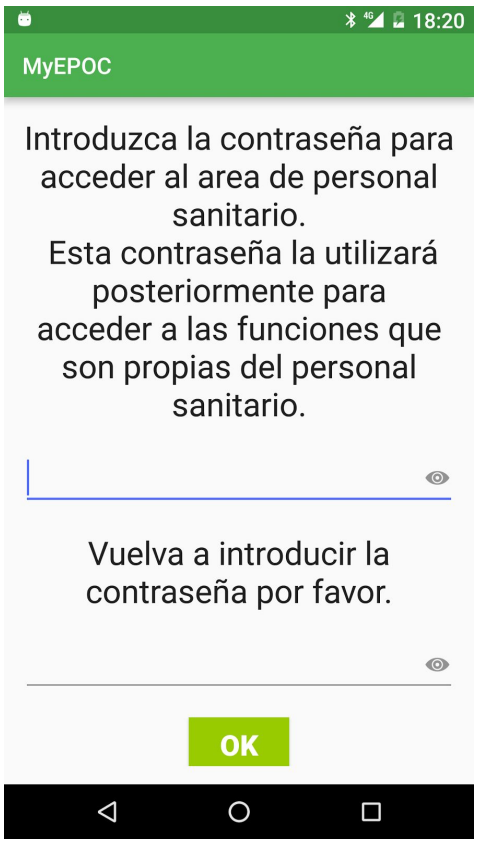
2- (Build.VERSION.*SDK_INT* < Build.VERSION_CODES.*KITKAT*) esta línea del código fue necesaria introducirla por el cambio que realizó google en el tratamiento de las alarmas dependiendo de la API en la que se ejecuta la aplicación, si el sistema usa una versión inferior a KITKAT, la creación de alarmas exactas se realizará con la función **alarmManager.set(...** esto creará una alarma exacta en el momento que le sea indicado, pero los dispositivos que utilizan una versión del sistema operativo superior o igual a KITKAT deben utilizar el método **alarmManager.setExact(...** ya que si utilizaran el método anterior se crearía una alarma inexacta.

3- pendingIntent, cuando la alarma suene se ejecutará la actividad que sea asignada a ese Intent.

1.5 Protección por contraseña cifrada del área del personal sanitario

Esta aplicación al contar tanto con el área del médico como con el área del paciente es necesario proteger la parte del médico para evitar modificaciones en el tratamiento de información delicada como análisis y espirometrías.

La primera vez que se instala la aplicación, ésta solicita al usuario que introduzca una contraseña para proteger el área médica, esta contraseña es almacenada en las shared preferences realizándole un HASH - SHA1 previamente.




```
SharedPreferences prefs =
getSharedPreferences("myPrefs",
Context.MODE_PRIVATE);
SharedPreferences.Editor editor = prefs.edit();
String pass = SHA1(ts);
editor.putString("contrasena", pass);
editor.putBoolean("sms", false);
editor.commit();
```

Como se puede observar se solicita que se introduzca la misma contraseña dos veces para evitar problemas al introducirla.

1.6 Interfaz menú principal con iconos

La interfaz de esta aplicación ha sido diseñado con el objetivo de que fuera amigable para el usuario y al mismo tiempo fuera válida para personas que puedan tener problemas en la vista, es por ello que se han utilizado iconos grande con un texto explicativo debajo.

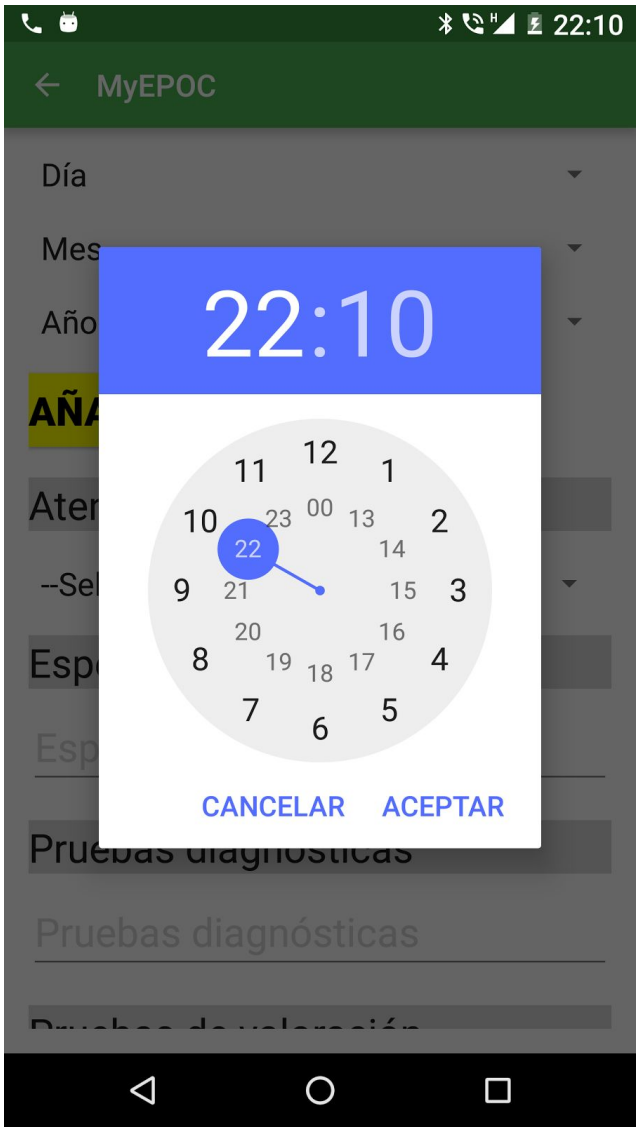
Todos los dibujos han sido hechos a mano para que la interfaz fuera lo más cercana al usuario.

1.7 Creación y listado de citas

Una de las funcionalidades de la aplicación es la posibilidad de almacenar citas que tenga el paciente en el futuro. Cuando se vaya a guardar una cita tenemos la posibilidad de seleccionar el día que se va a realizar la cita, la hora, por quien va a ser atendido el paciente, la especialidad a la que pertenece el facultativo que le va a tender, el tipo de pruebas que le van a realizar tanto diagnósticas como de valoración.

No es posible cumplimentar una cita si todos los campos no están rellenos así el paciente se asegura de que ha introducido correctamente la cita. Cuando la cita está guardada el sistema calcula mediante un objeto calendar el día de la semana que corresponde la fecha introducida, para facilitar al paciente la visualización de las citas.

Se ha introducido una funcionalidad visual que permitirá introducir la hora de la cita de una manera muy rápida e intuitiva.




```
public static class TimePickerFragment extends DialogFragment
      implements TimePickerDialog.OnTimeSetListener {
  @Override
  @NonNull
  public Dialog onCreateDialog(Bundle savedInstanceState) {
      // Use the current time as the default values for the picker
      final Calendar c = Calendar.getInstance();
      int hour = c.get(Calendar.HOUR_OF_DAY);
      int minute = c.get(Calendar.MINUTE);

      return new TimePickerDialog(getActivity(), this, hour, minute,
DateFormat.is24HourFormat(getActivity()));
  }
```

1.8 Permisos de la aplicación en tiempo de ejecución

Para proteger ciertos recursos y características especiales, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intención de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se generará una excepción de permiso y la aplicación será interrumpida inmediatamente.

Nuestra aplicación hace uso de los siguientes permisos.

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.alarm.permission.SET_ALARM" />
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"
/>
```

Permiso **vibrate**: Necesario para que las alarmas de las tomas cuando suenen también vibren.

Permiso **wake_lock**: Necesario para que cuando salten las alarmas de las tomas se superpongan a la aplicación que se encuentre ejecutando en ese momento.

Permiso **set_alarm**: Necesario para poder crear alarmas en el sistema.

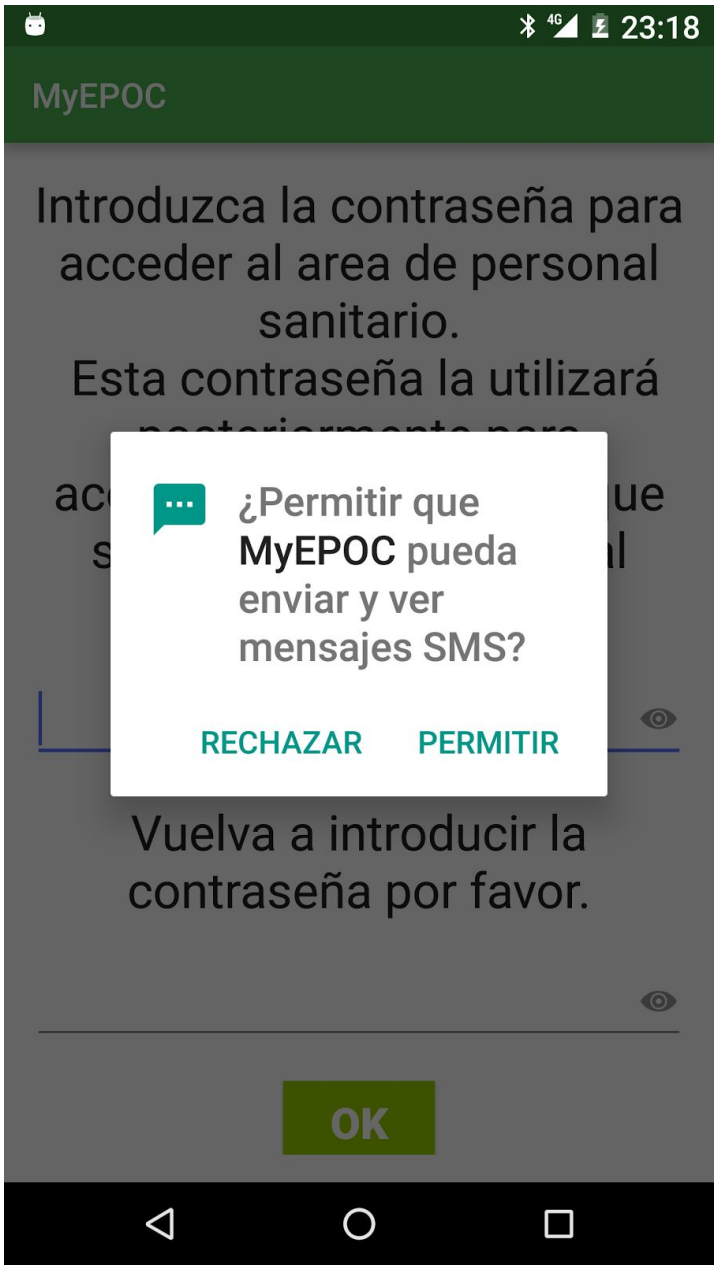
Permiso **receive_boot_completed**: Necesario para que el funcionamiento del receiver que resetea todas las alarmas una vez se haya reiniciado el dispositivo.

Permiso **send_sms:** Necesario para permitir a la aplicación enviar mensaje de texto.

Permiso **modfy_audio_settings**: Necesario para poder modificar los ajustes de sonido para las alarmas.

Android 6

Desde Android 6 (API 23) los permisos no se autorizan cuando se instala la APP, si no durante la ejecución ésta. Nuestra aplicación solicita los permisos durante la primera ejecución. Solicitamos permisos para todos los mencionados anteriormente, pero solamente aparece el de sms ya que es crítico y necesita confirmación expresa del usuario.



1.9 Carga en memoria de manera eficiente las imágenes para una correcta fluidez de los listView

Durante la ejecución de la aplicación se utilizan un gran número de imágenes, estas imágenes deben ser cargadas en memoria de la manera más eficientemente posible si no producirán problemas en el rendimiento y estabilidad de la aplicación, ya que no compensa cargar una imagen que tiene una resolución de 1024 x 768 en una botón que va a tener una dimensión de 100 x 100

Se ha seguido la documentación de developer de android que explica detalladamente los pasos a seguir [12].

El sistema carga en memoria un versión reducida de la imagen que se vaya a utilizar, para eso necesita saber las dimensiones del "contenedor" que tendrán esa imagen, una vez conocido este dato el método calcula el tamaño óptimo que debe cargarse en memoria.

Implementación

```
public static int calculateInSampleSize(
    BitmapFactory.Options options, int reqWidth, int reqHeight) {
  // Raw height and width of image
  final int height = options.outHeight;
  final int width = options.outWidth;
  int inSampleSize = 1;

  if (height > reqHeight || width > reqWidth) {

    final int halfHeight = height / 2;
    final int halfWidth = width / 2;

    // Calculate the largest inSampleSize value that is content_crear_tratamiento
power of 2 and keeps both
    // height and width larger than the requested height and width.
    while ((halfHeight / inSampleSize) > reqHeight
        && (halfWidth / inSampleSize) > reqWidth) {
      inSampleSize *= 2;
    }
  }
  return inSampleSize;
}

public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId,
```

```
                                        int reqWidth, int reqHeight) {

    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeResource(res, resId, options);

    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);

    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeResource(res, resId, options);
}

bfisio.setImageBitmap(
        decodeSampledBitmapFromResource(getResources(),
                this.getResources().getIdentifier("fisioterapiarespiratoria", "drawable",
getPackageName()),
                200, 200));
```

1.10 Patrón viewHolder en los listView

Esta aplicación utiliza una gran cantidad de List Views ya que es una manera rápida y consistente de mostrar elementos en pantalla, las filas de los List Views se van generando en tiempo real cuando el usuario va realizando scrolling para ver los elementos, si cada vez que se genera una fila del List View hay que buscar la referencia de los elementos que contiene podemos encontrarnos un problema de rendimiento.

Es por esto que se utiliza el patrón View Holder [11]:

Básicamente lo que hace este patrón es almacenar las referencias a los elementos como TextView, Buttons, etc en un holder que solamente se ejecutará la primera vez que se cree la fila, así cuando volvamos a ver esa fila del List View la referencia a ese elemento ya se encontrará almacenada y no habrá que volver a buscarla.

Implementación:

En primer lugar crearemos el holder que contendrá los elementos que aparecerán en cada fila de nuestro List View

```
static class ViewHolder {

  ImageButton botonBorrar;
  TextView fechaCita;
  TextView diaCita;
  TextView horaCita;
  TextView especialidad;
  TextView nombre_medico;
  TextView p_diag;
  TextView p_val;

}
```

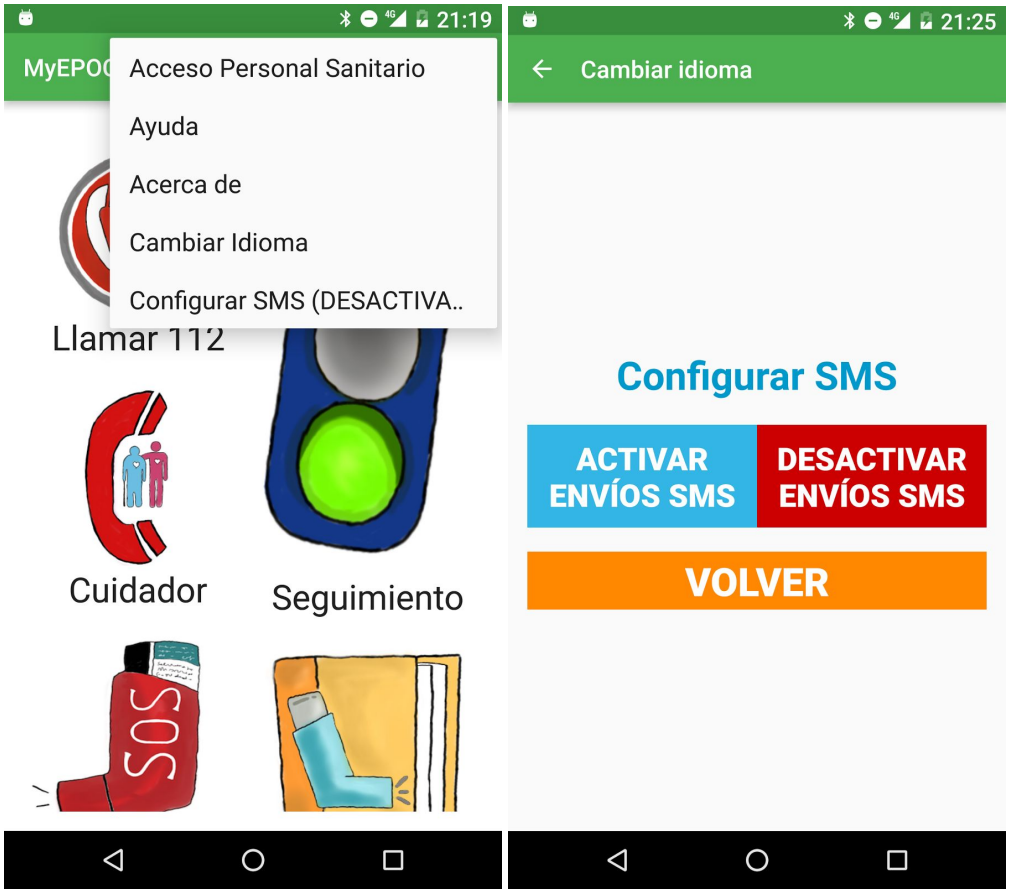
Ahora simplemente comprobamos si la fila ya ha sido generada, si no ha sido generado inicializamos los elementos del holder, si ya ha sido inicializada simplemente introduciremos los valores correspondientes en los elementos de la fila.

```
if(convertView==null){

   LayoutInflater inflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
   convertView = inflater.inflate(R.layout.listview_row_consultar_citas, parent, false);
   holder=new ViewHolder();
   holder.botonBorrar = (ImageButton) convertView.findViewById(R.id.imageButton);
   holder.fechaCita= (TextView) convertView.findViewById(R.id.textView11);
   holder.diaCita= (TextView) convertView.findViewById(R.id.textViewDia);
   holder.horaCita= (TextView) convertView.findViewById(R.id.textView12);
   holder.especialidad= (TextView) convertView.findViewById(R.id.textView13);
   holder.nombre_medico= (TextView) convertView.findViewById(R.id.textView14);
   holder.p_diag= (TextView) convertView.findViewById(R.id.textView15);
   holder.p_val= (TextView) convertView.findViewById(R.id.textView16);
   convertView.setTag(holder);
}else{

   holder = (ViewHolder) convertView.getTag();

}

holder.fechaCita.setText(context.getString(R.string.fechacita) + "
"+getItem(position).getDia()+"/"+getItem(position).getMes()+"/"+getItem(position).getAñ
o());

holder.diaCita.setText(getItem(position).getDay_week(getContext()));
holder.horaCita.setText(context.getString(R.string.horacita) + "
"+getItem(position).getHora()+":"+getItem(position).getMin());
holder.especialidad.setText(context.getString(R.string.especialidadcita) + "
"+getItem(position).getEsp());
holder.nombre_medico.setText(context.getString(R.string.atendidocita) + "
"+getItem(position).getN_doctor());
holder.p_diag.setText(context.getString(R.string.diagcita) + "
"+getItem(position).getP_diag());
holder.p_val.setText(context.getString(R.string.valcita) + " "+getItem(position).getP_val());
```

1.11 Activación desactivación de SMSs.

Una de las características más interesantes de la aplicación es la posibilidad de mantener actualizado al cuidador informal (entiéndase pareja, familiar) del progreso del tratamiento del paciente, utilizando el método de los mensajes de texto, en el momento que la alarma suena para realizar la toma que le corresponde al paciente la aplicación enviará un mensaje de texto indicando que el paciente debe realizar la toma.

Esta función está desactivada por defecto y puede ser activada y desactivada siempre que se desee, se permite esta posibilidad ya que los mensajes de texto pueden acarrear gastos económicos y el objetivo de esta aplicación es no producir ningún gasto.



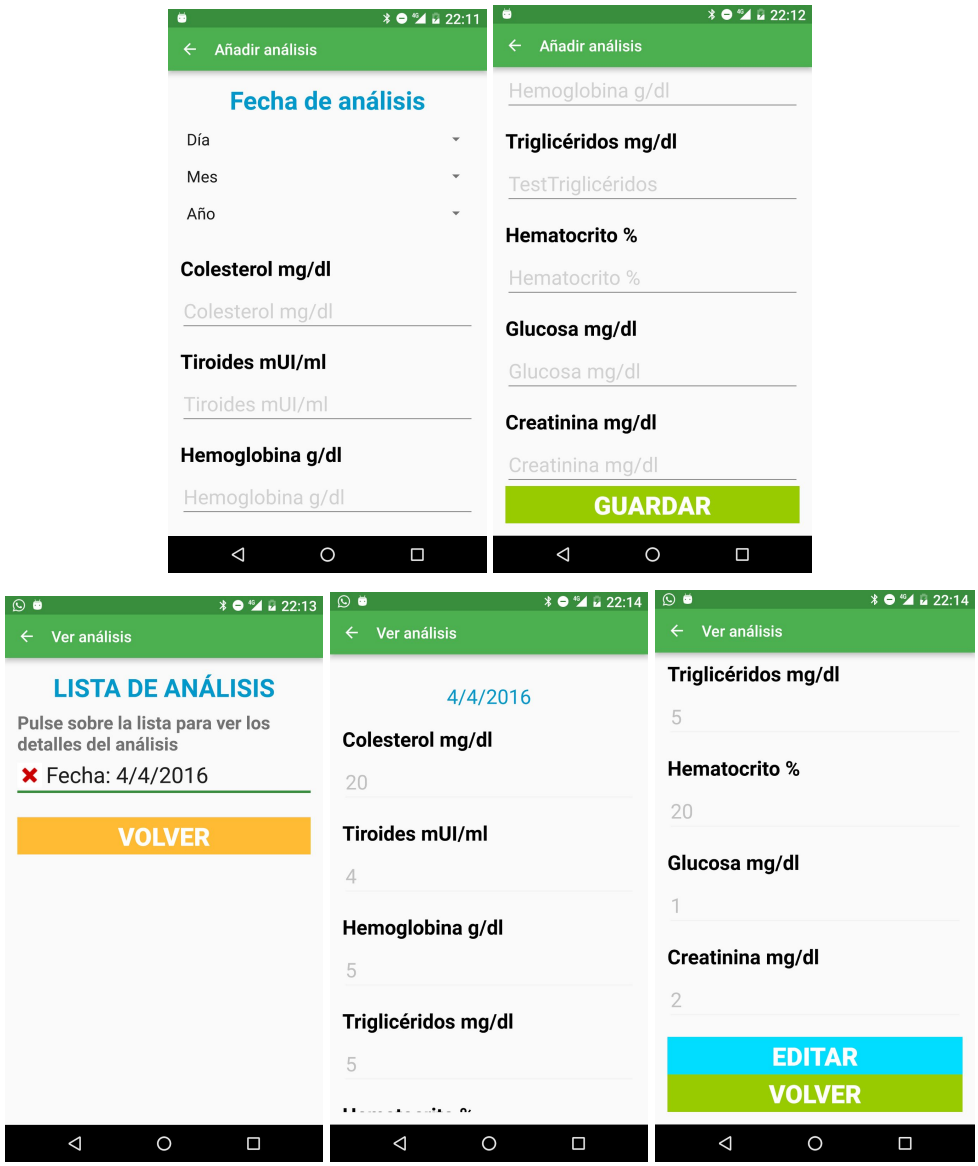
2da Iteración

2.1 Creación, consulta y modificación de análisis del paciente

Una de las funcionalidades que ofrece esta aplicación al personal sanitario es la posibilidad de llevar un registro cronológico de los análisis que se ha realizado el paciente para controlar el estado de su EPOC.

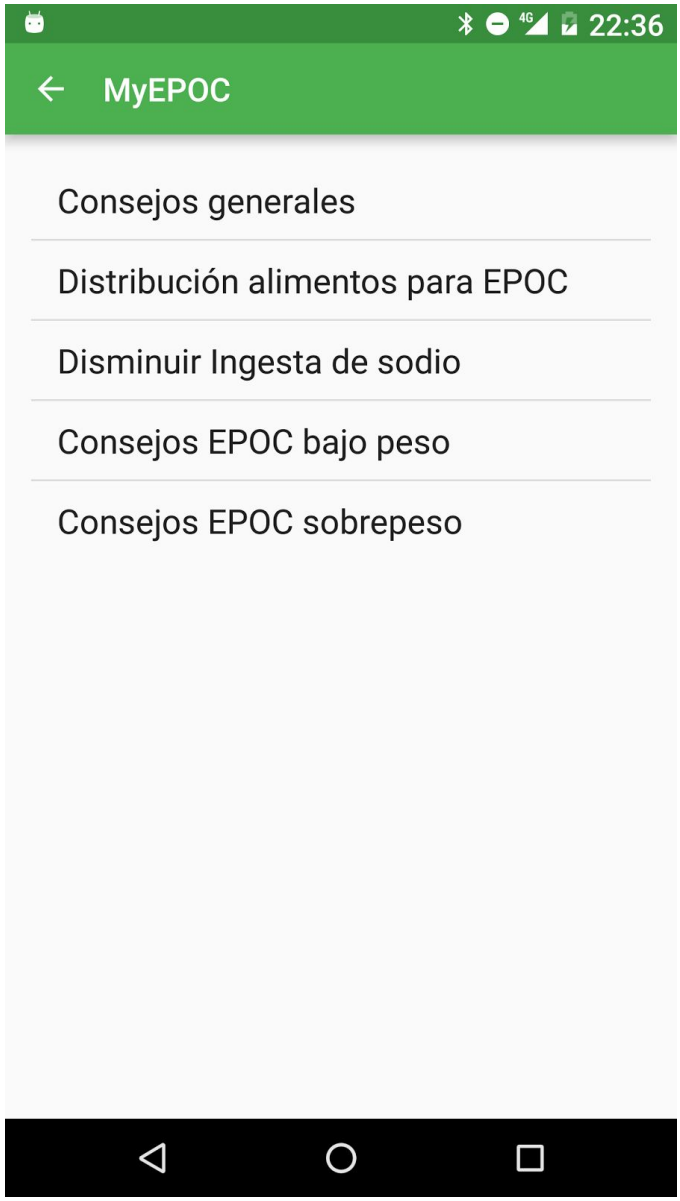
Una funcionalidad muy interesante ya que permitirá observar la evolución de los parámetros relacionados con la EPOC.

Los datos que se pueden almacenar son: colesterol, tiroides, hemoglobina, triglicéridos, hematocrito,glucosa y creatinina.



2.2 Consejos para la dieta del paciente

La aplicación contiene un gran número de consejos que permitirá a las personas con EPOC seguir una dieta adecuada a su enfermedad. Los consejos introducidos en la aplicación han sido suministrados por personal sanitario especializado.



2.3 Ordenación cronológica de los listView

Se ha introducido la ordenación de los List View que contengan fechas por orden cronológico ya que permitirá observar de manera mucho más clara culas citas pasadas y futuras.

Se ha realizado una implementación del método compareTo para así poder realizar un sort al ArrayList que contiene los objetos a ordenar.

```
public int compareTo(Analysis analysis) {
  if(Integer.parseInt(this.getAño())>Integer.parseInt(analysis.getAño())){
     return 1;
  }else if(Integer.parseInt(this.getAño())<Integer.parseInt(analysis.getAño())){
     return -1;
  }else{
     if(Integer.parseInt(this.getMes())>Integer.parseInt(analysis.getMes())){
        return 1;
     }else if(Integer.parseInt(this.getMes())<Integer.parseInt(analysis.getMes())){
        return -1;
     }else{
        if(Integer.parseInt(this.getDia())>Integer.parseInt(analysis.getDia())){
           return 1;
        }else if(Integer.parseInt(this.getDia())<Integer.parseInt(analysis.getDia())){
           return -1;
        }else{
           return 0;
        }
     }
  }
}
```

2.4 Desarrollo iterativo

Durante todo el desarrollo de la aplicación se ha realizado un desarrollo iterativo en el cual se hecho tanto las tareas de analista como de programador, se han mantenido reuniones con la clienta real en las cuales se han realizado la toma de requisitos, de los cuales se han creado los casos de uso y el boceto de las interfaces, estas interfaces han sido mostradas a la clienta para que diera su visto bueno. Durante todo este proceso se le han entregado versiones funcionales de la aplicación para que siguiera su desarrollo y pudiera detectar posibles errores que existieran.

Se han realizado 5 iteraciones, de las cuales las 4 primeras el equipo de Ing de la Salud tomó el papel de analista y el equipo de Ing de Software el papel de programador, y en la última el equipo de software ha tomado tanto el papel de programador como el de analista

En las distintas iteraciones se han desarrollado las siguientes funcionalidades:

En la primera iteración, se implementaron las funcionalidades de:

- Consultar manual de usuario de un inhalador.
- Consultar tratamiento (listado de tomas).
- Crear una toma.
- Consultar una toma.
- Borrar tomas.
- Creación Base de Datos.

En la segunda:

- Visualización del semáforo.
- Alarma de aviso de toma.
- Añadido acerca de de la aplicación.
- Fixes iteración anterior.

En la tercera:

- Fisioterapia Respiratoria.
- Consultar y crear tratamiento de Rescate y recomendaciones.
- Llamar al teléfono de emergencia.
- CRUD datos del paciente.
- Llamar al teléfono del cuidador.
- Fixes iteración anterior.

En la cuarta:

- Crear cita con personal sanitario.
- Consultar citas.
- Protección por contraseña área del personal sanitario.
- Activación y desactivación de envío de SMS.
- Actualización interfaz gráfica

En la quinta:

- CRUD análisis.
- Consejos sobre las dietas.
- Edición de las tomas.
- CRUD espirometría.
- Guía para dejar de fumar.

2.5 Conclusiones

El objetivo principal que pretendía cumplir esta aplicación era mejorar la adherencia del paciente al tratamiento de EPOC que su médico le ha asignado, así como su seguimiento por parte de los allegados al paciente.

Creemos que esta aplicación ha alcanzado satisfactoriamente este objetivo, ya que permitirá al paciente controlar su enfermedad de una manera mucho más efectiva gracias a la cantidad de funcionalidades que esta aplicación le ofrece, además de este objetivo hemos conseguido que la aplicación sea compatible con la gran mayoría de teléfonos móviles Android actuales.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.9%
4.1.x	Jelly Bean	16	6.8%
4.2.x		17	9.4%
4.3		18	2.7%
4.4	KitKat	19	31.6%
5.0	Lollipop	21	15.4%
5.1		22	20.0%
6.0	Marshmallow	23	10.1%

La versión mínima que puede utilizar esta aplicación es la API 14, así que, podrán utilizar la aplicación un **97,9%** de los dispositivos android que existen en el mercado actual.

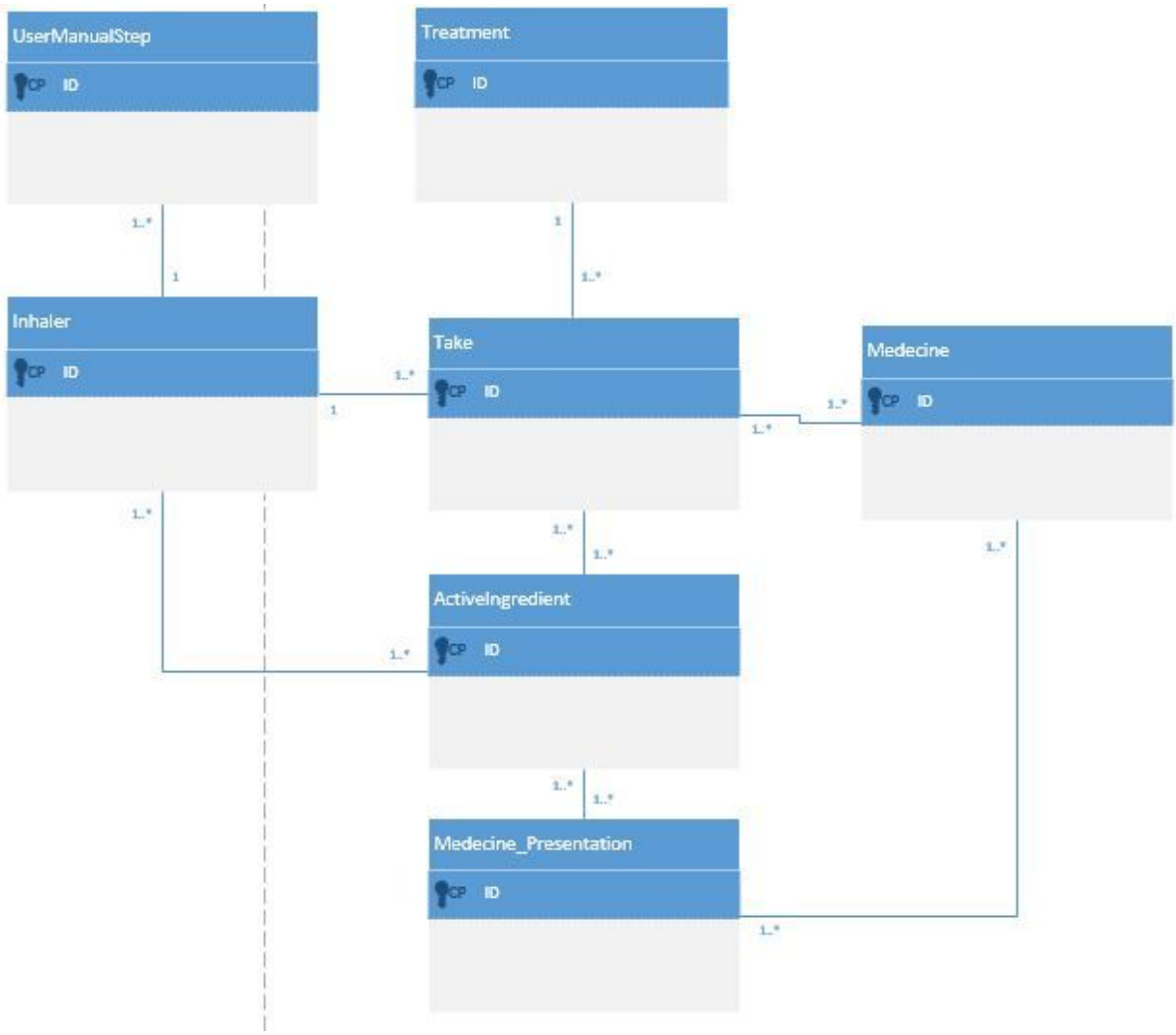
Otra de las características que ha alcanzado esta aplicación es la accesibilidad que proporciona a aquellas personas que tengan algún tipo de impedimento, ya sea visual o de otro tipo, esto es debido a que se ha buscado en todo momento la utilización de fuentes de un tamaño elevado, imágenes representativas y amigables, también se han añadido a los tutoriales la posibilidad de que el móvil reproduzca los textos, centrado para las personas que tengan problemas en la visión.

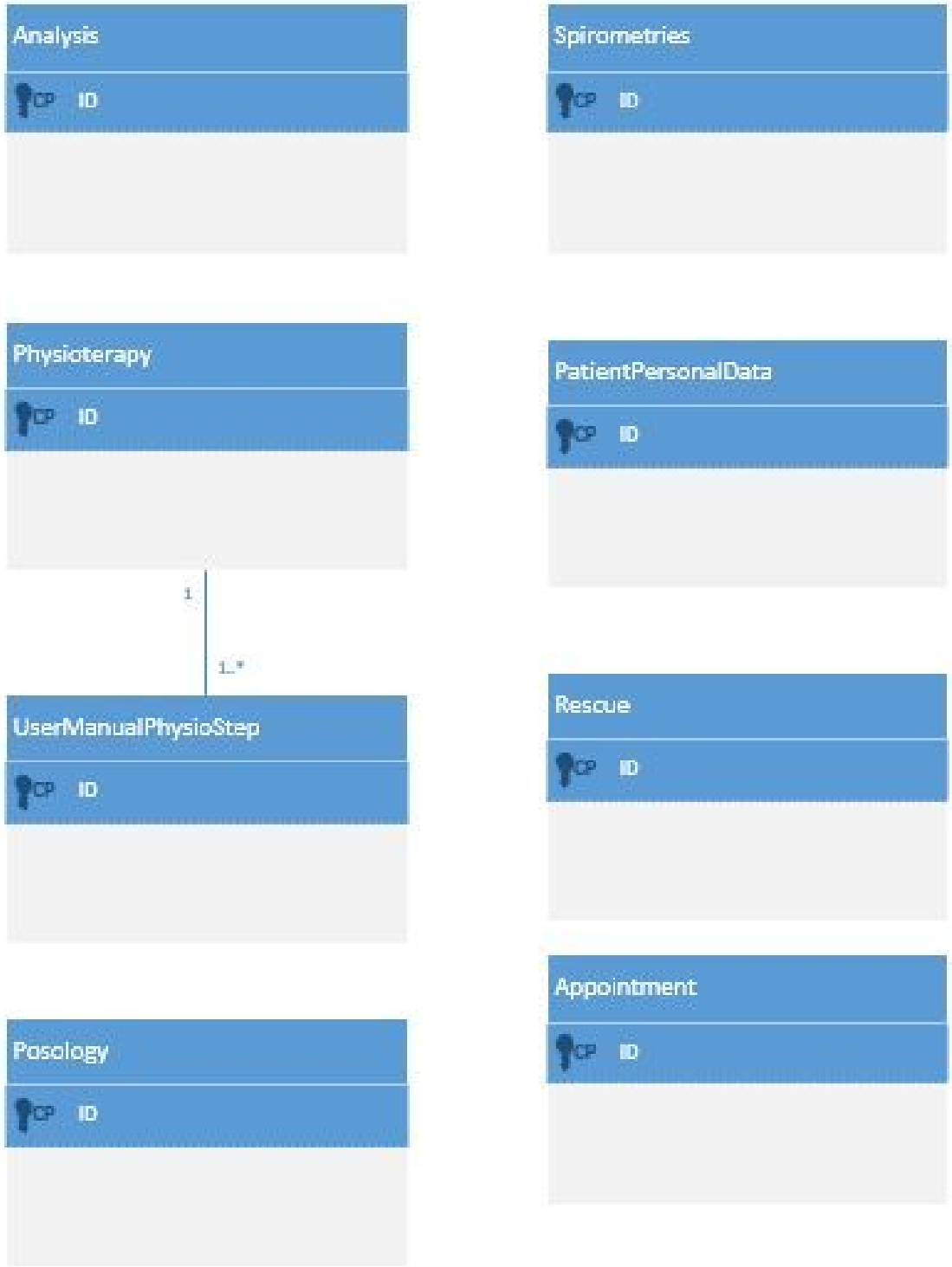
Las posibilidades de ampliación de esta aplicación son muy elevadas, ya que, podría ser utilizada para mejorar la adherencia a un tratamiento de un paciente de cualquier tipo de enfermedad, simplemente haría falta introducir los medicamentos que sean necesarios para esa enfermedad así como los tutoriales de uso de estos. Además la estructura de la base de datos permitirá el uso simultáneo de distintos tratamientos y perfiles de usuario, por lo que una familia podría utilizar un solo dispositivo (Ejemplo: una tablet) para llevar a cabo el control del tratamiento de todos los miembros de la familia.

Bibliografía

[1] https://www.android.com/intl/es_es/
[2] https://www.sqlite.org/docs.html
[3] https://docs.oracle.com/javase/7/docs/api/
[4] http://www.w3.org/XML/
[5] https://www.zetetic.net/sqlcipher/documentation/
[6] https://git-scm.com/documentation
[7]https://confluence.atlassian.com/bitbucket/bitbucket-cloud-documentation-home-221448814.html
[8] https://www.sourcetreeapp.com
[9] https://developer.android.com/studio/index.html
[10] https://slack.com
[11] https://developer.android.com/training/improving-layouts/smooth-scrolling.html
[12] https://developer.android.com/training/displaying-bitmaps/load-bitmap.html

Anexo con estructura de la base de datos.




Analysis
ID
Spirometries
ID
Physioterapy
ID
PatientPersonalData
ID
1
1..*
UserManualPhysioStep
ID
Rescue
ID
Posology
ID
Appointment
ID