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
Sergio Sánchez Gil
Tutorizado por
José María Álvarez Palomo
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JULIO 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: Aplicación android para ayudar a los pacientes con EPOC (Enfermedad Pulmonar Obstructiva Crónica). Gracias a esta aplicación, el personal sanitario podrá personalizar un tratamiento para cada paciente, con la intención tanto de que este mejore su calidad de vida y su adherencia a dicho tratamiento, como de que sus cuidadores puedan hacer un seguimiento de la evolución de la enfermedad.

La aplicación presenta una sección reservada para el personal sanitario. En dicha sección, se podrán configurar tratamientos, recomendaciones, espirometrías, datos personales… También tiene una parte para el paciente, donde puede ver los tratamientos que tiene asignados, manuales de uso de los inhaladores, citas, medicación de rescate… y una de las cosas más llamativas, es que en la pantalla principal de la aplicación, el paciente puede ver si está siguiendo bien el tratamiento mediante un código de colores representado en un semáforo.

Hemos seguido un proceso iterativo con una clienta real, con la que nos hemos reunido varias veces presentándole prototipos de la aplicación, y pidiéndole nuevos requisitos para las siguientes iteraciones. Dicha clienta (enfermera en Nerja) nos ha proporcionado toda la información necesaria para la creación de la aplicación.

Palabras claves: APP., Android, EPOC, metodología iterativa, salud, Junta de Andalucía.

Resumen: Android application to help patients with EPOC (Chronic Obstructive Pulmonary Disease). Thanks to this application, medical personnel can customize treatment for each patient, with the intention aware that this improves their quality of life and adherence to such treatment, and that their caregivers can track the evolution of the disease.

The application has a section reserved for health workers. In this section, you can configure treatments, recommendations spirometry, personal data ... also has a part to the patient, where you can see the treatments are assigned, user manuals inhalers, quotes, rescue medication ... and one of the most striking things is that on the main screen of the application, the patient can see if you are following the treatment either by color coding represented at a stoplight.

We have followed an iterative process with a real client, with whom we have met several times presenting prototypes of the application, and asking new requirements for the following iterations. Such client (nurse in Nerja) has provided us with all the information necessary for creating the application.

Palabras claves: APP., Android, EPOC, iterative methodology, health, Junta de Andalucía.

Índice

1 Introducción

1.1 Motivación y objetivos:

El desarrollo de esta aplicación surgió a partir de un concurso propuesto por la Consejería de Salud y la Consejería de Empleo, Empresa y Comercio de la Junta de Andalucía, en colaboración con Vodafone, donde se pedían aplicaciones para móviles con el fin de ayudar tanto a los pacientes con EPOC como a sus allegados.

Había un grupo formado por una clase de Ingeniería de la salud, y una enfermera de Nerja, pero no tenían desarrolladores, por lo que contactaron con Diego Narbona Vílchez y conmigo. En la primera parte, los compañeros de salud han sido los que han hecho el papel de analistas y nosotros únicamente de programadores. Una vez presentada la aplicación al concurso, hemos seguido desarrollándola para presentarla como trabajo de fin de grado, pero en esta segunda parte, también hemos adoptado el papel de analista, a parte del de programador.

Hemos decidido hacerla en Android, ya que es el mercado con más demanda. No se ha usado ninguna fuente de financiación, promoción o patrocinio para desarrollar esta aplicación. No existen conflictos de intereses en la aplicación.

La aplicación ha sido desarrollada a partir de fuentes de información proporcionadas por una clienta real especializada en el ámbito sanitario, toda la información incorporada está basada en datos y pautas reales acerca de la EPOC.

La aplicación tiene como objetivo ayudar a los pacientes con EPOC, para ello, deberá ser el médico quien instale la aplicación en el móvil del paciente, y quien configure los distintos tratamientos, gestiones los datos personales del paciente, de sus cuidadores y de la enfermedad, cree espirometrías e informes de análisis, etc. Estas acciones solo serán posibles desde la parte reservada para el personal sanitario, a la que solo se puede acceder mediante una contraseña. Dicha contraseña se crea a la hora de instalar la aplicación.

1.2 Tecnologías usadas

Android: [1] Android es un sistema operativo orientado a dispositivos móviles, basado en una versión modificada del núcleo Linux. Inicialmente fue desarrollado por Android Inc., una pequeña empresa, que posteriormente fue comprada por Google; en la actualidad lo desarrollan los miembros de la Open Handset Alliance (liderada por Google).

El 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.
El código fuente de Android está disponible bajo diversas licencias de software libre y código abierto, Google liberó la mayoría del código de Android bajo la licencia Apache. Todo esto permite que un desarrollador no solo pueda modificar su código sino también mejorarlo. A través de esas mejoras puede publicar el nuevo código y con el ayudar a mejorar el sistema operativo para futuras versiones.

SQLite: [2] SQLite es un sistema de gestión de bases de datos relacional de software libre, que permite almacenar información en dispositivos empotrados de una forma sencilla, eficaz, potente, rápida y en equipos con pocas capacidades de hardware, como puede ser una PDA o un teléfono celular. SQLite implementa el estándar SQL92 y también agrega extensiones que facilitan su uso en cualquier ambiente de desarrollo. Esto permite que SQLite soporte desde las consultas más básicas hasta las más complejas del lenguaje SQL, y lo más importante es que se puede usar tanto en dispositivos móviles como en sistemas de escritorio, sin necesidad de realizar procesos complejos de importación y exportación de datos, ya que existe compatibilidad al 100% entre las diversas plataformas disponibles, haciendo que la portabilidad entre dispositivos y plataformas sea transparente.

Java: [3] 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, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

XML: [4] Extensible Markup Language (XML) es un formato de texto simple, muy flexible derivado de SGML (ISO 8879). Originalmente diseñado para cumplir con los retos de la publicación electrónica a gran escala, XML también

está desempeñando un papel cada vez más importante en el intercambio de una amplia variedad de datos en la Web y en otros lugares.

SQLCipher: [5] SQLCipher es una extensión de código abierto para SQLite que proporciona cifrado AES-256 bits a los archivos de base de datos. SQLCipher realiza el cifrado de forma transparente y sobre la marcha.

Git: [6] es un sistema distribuido de control de versiones libre y de código abierto diseñado para manejar todo, desde pequeñas a proyectos muy grandes con rapidez y eficacia.

SourceTree: [7] Es un cliente de Git que proporciona una interfaz para trabajar con repositorios locales y remotos de Git.

Bitbucket: [8] es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de versiones Mercurial y Git.

Android Studio: [9] es un entorno de desarrollo integrado para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. Ofrece:
- Un entorno de desarrollo claro y robusto.
- Facilidad para testear el funcionamiento en otros tipos de dispositivos.
- Asistentes y plantillas para los elementos comunes de programación en Android.
- Un completo editor con muchas herramientas extra para agilizar el desarrollo de nuestras aplicaciones.

Slack: [10] es una herramienta de comunicación en equipo. Slack no es un chat, tampoco un almacén de archivos, ni un calendario. Tampoco se puede definir como un sistema de alertas, pero suma todas esas opciones en un solo hilo de conversación.

1.3 Estructura de la memoria

El cuerpo principal de la memoria, estará dividida en tres partes, en la primera, se explicaran las funcionalidades implementadas desde el papel de programador. En esta parte se desarrolló: añadir, borrar y ver con detalle un tratamiento desde la parte reservada al personal sanitario. Insertar los datos personales del paciente, de los cuidadores y de la enfermedad cuando se instala la aplicación por primera vez y poder consultarlos y editarlos posteriormente desde la parte reservada al personal sanitario. Crear y borrar el tratamiento de rescate y las recomendaciones desde la parte reservada al personal sanitario. Consultar el tratamiento de rescate desde la interfaz principal. Botón para llamar al 112. Mejora de la interfaz gráfica de toda la aplicación cambiando los estilos de las letras y los botones. Base de datos.

En esta primera parte también se implementaron funcionalidades en conjunto entre Diego Narbona Vílchez y yo: Creación de las alarmas y acción de cambio de color del semáforo.

En la segunda, se explicaran las funcionalidades implementadas desde el papel de programador y analista. En esta parte se desarrolló: Editar las tomas configuradas desde la parte reservada al personal sanitario. Crear, consultar y editar espirometrías desde la parte reservada al personal sanitario. Guía para dejar de fumar.

En las dos primeras partes, de cada punto se explicará con detalle su funcionalidad dentro de la aplicación y las partes del código más relevantes. Ambas cosas ilustradas con fotos.

En la tercera parte, se hará mención del proceso iterativo seguido.

2 Primera parte

2.1 Añadir, borrar y ver con detalle un tratamiento desde la parte reservada al personal sanitario.

Una vez entramos a la parte reservada para el personal sanitario, podemos ver la opción de crear tratamiento:

Dentro de esta sección, el personal sanitario puede añadir un tratamiento, borrarlo, editarlo o consultarlo. Si ya teníamos algunas tomas guardadas, estas aparecerán en esta vista. Para ello, se carga la lista de tomas de la base de datos en el método onCreate, donde se crean dos lista de Tomas, esto servirá a la hora de actualizar la base de datos, más abajo explicado.

```
ArrayList<Toma> listaTomas = modelo.obtenerListaTomas();

for (Toma tm : listaTomas) {

    Toma t = new Toma(tm.getId(), tm.getMedicamento(), tm.getPrincipio(),
tm.getInhalador(),
            tm.getDosis(), tm.getHour(), tm.isDone(), tm.getInhalations(),
            tm.getPosMedicamento(), tm.getPosPrincipio(), tm.getPosInhalador(),
            tm.getPosDosis(), tm.getPosInhalations());

    listaTratamientosNuevos.add(t);
    listaTratamientosOriginal.add(t);
}
```

Una vez obtenida la lista de Tomas, se le pasa a un adaptador personalizado “AdapterListView”, y se asocia el adaptador y el objeto ListView que representa la lista de tomas.

```
adapterListView = new AdapterListView(noTomas, infoTomas, modelo, this,
listaTratamientosNuevos, listView);
listView.setAdapter(adapterListView);
Commodities.ajustarVistaListView(listView);
```

En dicho adaptador, por cada elemento de la lista de tomas que se le pasa, se crea una fila con la información de dicha toma, y un botón para poder borrarla.

La primera vez que se instale la aplicación, o en el caso de que no tenga ningún tratamiento asignado, aparecerá un mensaje informando de esto.
Para añadir un tratamiento, ha de pulsar sobre el botón añadir. Al presionar sobre dicho botón, se ejecuta el siguiente método:

```
//Accion del boton Añadir de crear tratamiento.
public void onClickBotonAdd(View view) {
    Intent intentAdd = new Intent(this, AddNewTake.class);
    intentAdd.putParcelableArrayListExtra("listaTomas", listaTratamientosNuevos);
    intentAdd.putExtra("position", "-1");
    startActivityForResult(intentAdd, 1);
}
```

Con el Intent, se lanza la actividad que se especifique en sus parámetros al inicializarlo, en este caso, se lanza la actividad AddNewTake. Una de las opciones del Intent es el paso de objetos (o lista de objetos), para ello, primero ha de implementar Parcelable dicho objeto que se desea enviar a la otra actividad. Gracias a esto, se puede enviar la lista de tomas ("listaTratamientosNuevos") que anteriormente se ha obtenido de la base de datos. También le paso un String con clave "position" y valor "-1". Gracias a esto, puedo saber si estoy añadiendo una toma nueva, en cuyo caso "position" es "-1", o editando, en cuyo caso "position" será igual a la posición del ítem seleccionado.

Al ejecutar la sentencia "startActivityForResult" (con el Intent anteriormente creado, y un código, en este caso el código es 1) al volver a esta actividad, se ejecutará el método "onActivityResult", donde se comprobará con que código vuelve. Si el código de vuelta es 1, significa que vuelve de la actividad "AddNewTake", que fue lanzada con ese código.

Cuando se lanza la nueva actividad (AddNewTake), se abrirá una vista lista para añadir una nueva toma.

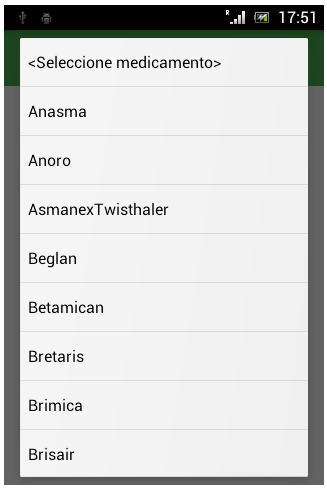
En esta vista, sólo se podrá seleccionar el medicamento, y una vez se seleccione el medicamento, se cargarán el resto de desplegables en relación al medicamento seleccionado. Gracias a esto, la aplicación es más eficiente, ya que no tiene que cargar todos los datos, y es más fácil para el usuario a la hora de crear una nueva toma ya que se filtran los datos según el medicamento seleccionado.

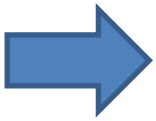
En la actividad “AddNewTake”, se recogen la lista de tomas, y la posición.

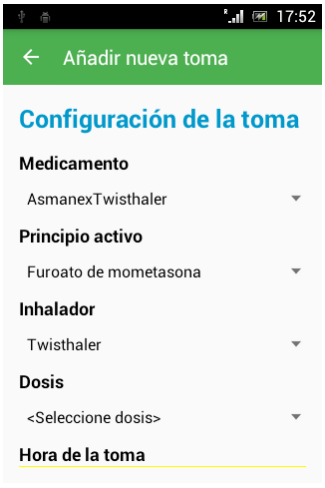
```
listaTomasNuevas = getIntent().getParcelableArrayListExtra("listaTomas");
pos = getIntent().getExtras().getString("position");
```

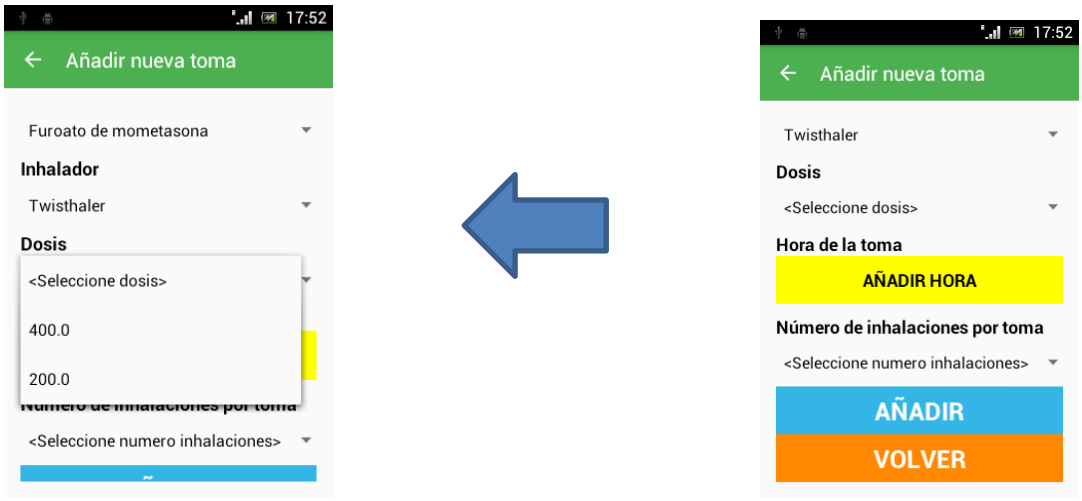
Gracias a que hemos recogido la lista de tomas que ya existe, se puede calcular si la configuración que estamos añadiendo ya existe o no, y de esta forma, la aplicación no permitirá que se añadan dos tomas con el mismo principio activo. En este caso aparecerá un mensaje de error.

Una vez en la vista de añadir una nueva toma, al seleccionar el medicamento, muchos de los otros campos se autocompletan al no tener más de una opción, pero habrá otros que el usuario tenga que introducir.

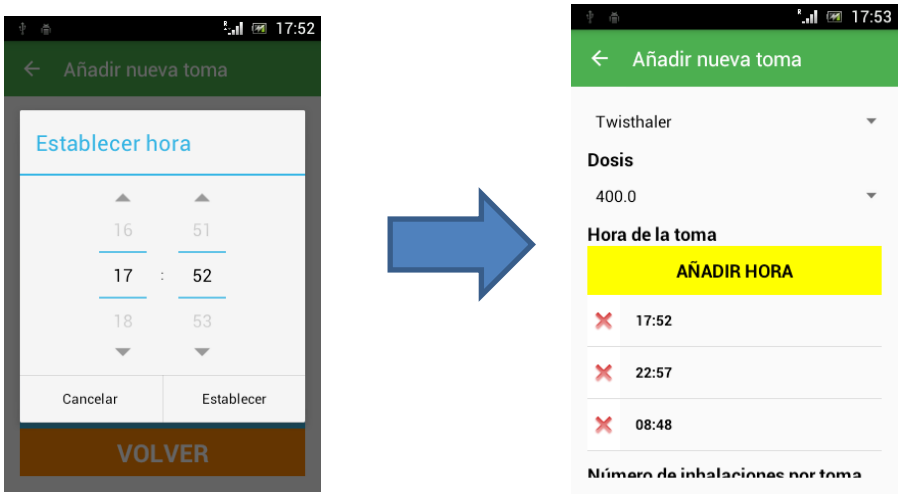




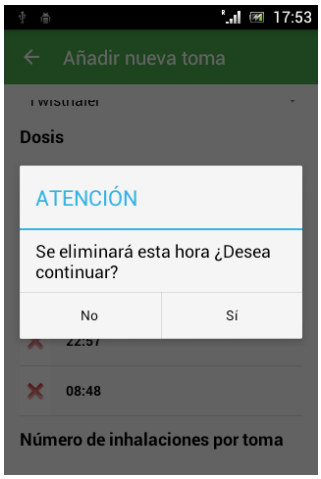




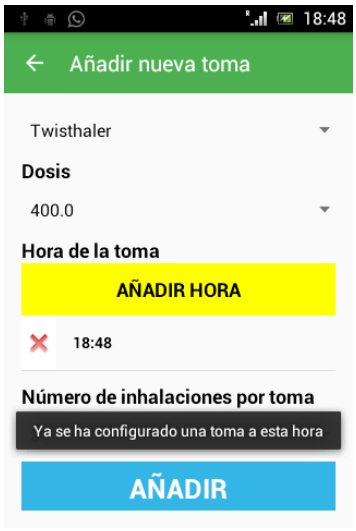
El usuario podrá añadir más de una hora para esa toma, para ello, al pulsar sobre el botón de “AÑADIR HORA”, se abre un reloj (o una lista, según la versión de android que tenga el dispositivo), donde podrá seleccionar la hora deseada. Para añadir otra hora, deberá repetir el proceso. Dichas horas se podrán visualizar en una lista debajo del botón.



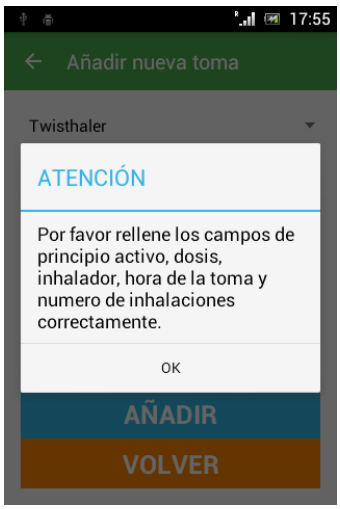
Al añadir una nueva hora, también podemos borrarla presionando sobre la X roja que aparece en la lista.



Si se introducen dos horas iguales, aparecerá un mensaje de error.



Si se intenta añadir una toma sin rellenar todos los campos, también aparecerá un mensaje de error.



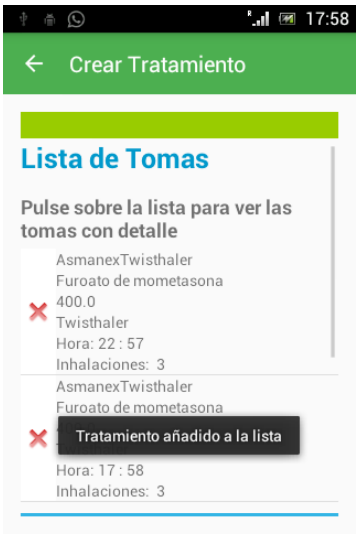
Una vez se hayan rellenado todos los campos correctamente, al presionar sobre el botón añadir, se crea una toma por cada hora que hayamos puesto, y se crea una lista de tomas.

```
for (String hora : listaHoras) {
    Hour h = new Hour(Integer.parseInt(hora.substring(0, 2)),
Integer.parseInt(hora.substring(hora.length() - 2,
            hora.length())));
    Toma t = new Toma(-1, m.getName(), p.getName(), in.getName(), d, h, false,
numInhalaciones,
            spinnerMedicamento.getSelectedItemPosition(),
spinnerPrincipio.getSelectedItemPosition(),
            spinnerInhaladores.getSelectedItemPosition(),
spinnerDosis.getSelectedItemPosition(),
            spinnerInhalaciones.getSelectedItemPosition());
    listaTomas.add(t);
}
```

Esta lista de tomas es la que se devuelve a la actividad padre (“CrearTratamiento”) con la clave “newTake”. Para ver que lista devolver, se comprueba el valor de “position” para ver si se está añadiendo una toma nueva, o editando una ya creada (en cuyo caso, “position” será igual a la posición de la toma en la lista de tomas recibida).

```
Intent newTakes = new Intent();
if (!pos.equals("-1")) {
    Toast pulsarAdd = Toast.makeText(this, R.string.toastAddTratamientoEditado,
Toast.LENGTH_SHORT);
    pulsarAdd.show();
    newTakes.putParcelableArrayListExtra("newTake", listaTomasNuevas);
    newTakes.putExtra("position", pos);
} else {
    Toast pulsarAdd = Toast.makeText(this, R.string.toastAddTratamiento,
Toast.LENGTH_SHORT);
    pulsarAdd.show();
    newTakes.putParcelableArrayListExtra("newTake", listaTomas);
    newTakes.putExtra("position", pos);
}

setResult(Activity.RESULT_OK, newTakes);

this.onBackPressed();
```

Gracias a que se crea una toma por cada hora, el personal sanitario podrá editar, consultar o borrar una toma a una hora específica. Al presionar sobre añadir, aparecerá un mensaje informando de que se ha añadido correctamente y se vuelve a la vista anterior, donde aparecerán las nuevas tomas creadas.

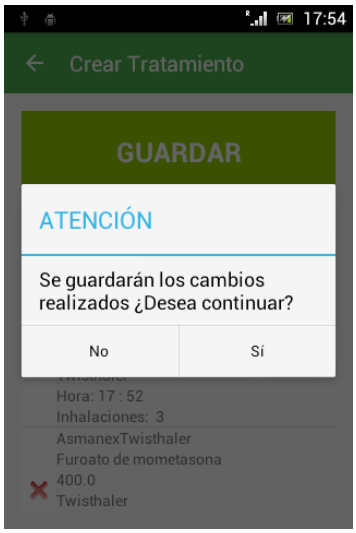


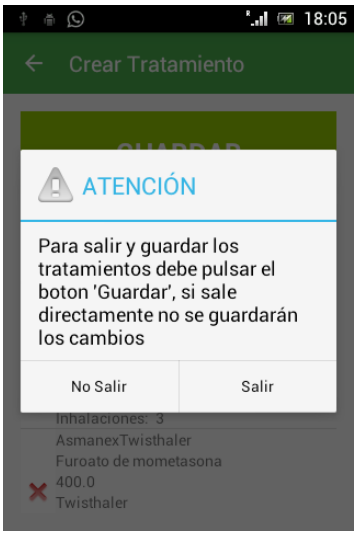
Cuando volvemos a “CrearTratamiento”, se ejecuta el método onActivityResult, donde se comprueba el requestCode. Al volver de la actividad AddNewTake, lanzada con el código 1, éste será el requestCode.

Tras esto, se recoge la lista de tomas nuevas, se añaden a la lista asociada al adaptador del ListView y se notifica el cambio para que se reflejen las nuevas tomas añadidas. Además se ajusta la vista para que aparezcan todos los ítems de la lista.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // Comprobamos si el resultado de la segunda actividad es "RESULT_CANCELED".
    if (resultCode == Activity.RESULT_OK && requestCode == 1) {
        //Toma t = data.getParcelableExtra("newTake");
        ArrayList<Toma> t = data.getParcelableArrayListExtra("newTake");

        for (Toma toma : t) {
            listaTratamientosNuevos.add(toma);
            adapterListView.notifyDataSetChanged();
            Commodities.ajustarVistaListView(listView);
        }

        if (listaTratamientosNuevos.isEmpty()) {
            noTomas.setVisibility(View.VISIBLE);
            infoTomas.setVisibility(View.GONE);
        } else {
            noTomas.setVisibility(View.GONE);
            infoTomas.setVisibility(View.VISIBLE);
        }

    }
```

Para poder guardar definitivamente el tratamiento del paciente, se deberá pulsar sobre el botón “GUARDAR”, en otro caso se avisará de que no se guardarán los cambios, y si se acepta, no se almacenarán las nuevas tomas en la base de datos.





Al presionar sobre el botón “GUARDAR”, se borran de la base de datos los tratamientos que había anteriormente, y que estaban guardados en “listaTratamientosOriginal”, y se inserta “listaTomasNuevas”, donde están las tomas que aparecen en el ListView.

Una vez creadas las tomas, también se pueden borrar, para ello, se deberá presionar sobre la X roja que aparece en la lista de tomas.

Esta X roja, se gestiona desde el adaptador personalizado adapterListView, donde para cada ítem que se pinta en la pantalla, se añade un botón (X roja) que borra de la lista de tomas, el ítem seleccionado.

```
ImageButton borrar = (ImageButton) item.findViewById(R.id.imageButtonBorrarVistaLista);
borrar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(context);
        alertDialogBuilder.setTitle(R.string.alerDialogTitulo);
        alertDialogBuilder
                .setMessage(R.string.alertDialogEliminarTexto)
                .setCancelable(false)
                .setPositiveButton(R.string.alertDialogBotonSi, new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        Toast pulsarAdd = Toast.makeText(context, R.string.toastBorrar,
Toast.LENGTH_SHORT);
                        pulsarAdd.show();
                        AdapterListView.this.remove(toma);
                        AdapterListView.this.notifyDataSetChanged();
                        Commodities.ajustarVistaListView(listView);

                        if(AdapterListView.this.isEmpty()){
                            noTomas.setVisibility(View.VISIBLE);
                            infoTomas.setVisibility(View.GONE);
                        }else{
                            noTomas.setVisibility(View.GONE);
                            infoTomas.setVisibility(View.VISIBLE);
                        }
                    }
                })
                .setNegativeButton(R.string.alertDialogBotonNo, new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                    }
                });

        AlertDialog alertDialog = alertDialogBuilder.create();

        alertDialog.show();

    }
});
```

Para guardar los cambios, a la hora de borrar tomas, en la base de datos, se deberá pulsar sobre el botón “GUARDAR”, al igual que para guardar las tomas nuevas.

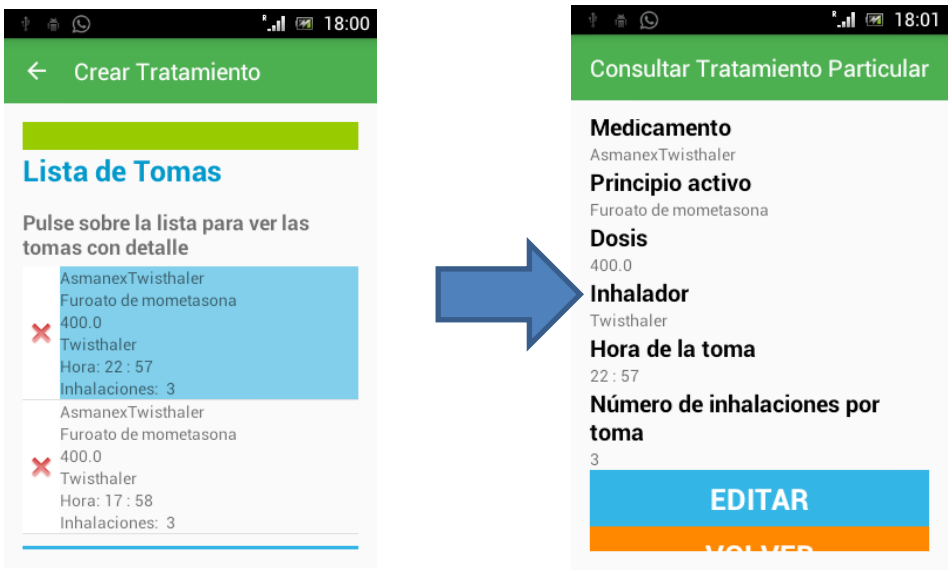
En el onCreate de la actividad CrearTratamiento, también se asocia un onItemClickListener para cada ítem de la lista de tomas, para que al pulsar sobre este, se lance una nueva actividad con la información de la toma detallada.

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

        Intent intent = new Intent(getApplicationContext(), ConsultarTomaMedico.class);

        intent.putParcelableArrayListExtra("listaTomas", listaTratamientosNuevos);

        intent.putExtra("position", Integer.toString(position));

        startActivityForResult(intent, 2);
```

Al igual que a la hora de añadir, también se envía la lista de tomas, y la posición, que esta vez, será la posición, dentro de la lista, de la toma que se desea ver con detalle.

Al ejecutar la sentencia “startActivityForResult”, en este caso el código es 2.

Para poder ver con detalle una toma, se deberá pulsar sobre esta en la lista, tras esto, se abrirá una nueva vista donde se puede ver con más detalle.



Dentro de esta vista, está la opción de editar, pero esta funcionalidad ha sido implementada en la segunda parte, por lo que será explicada más adelante.

2.2 Insertar los datos personales del paciente, de los cuidadores y de la enfermedad cuando se instala la aplicación. por primera vez y poder consultarlos y editarlos posteriormente desde la parte reservada al personal sanitario.

Cuando se instala la aplicación en el móvil, antes de mostrar la vista principal, se pide que el personal sanitario que la instale, rellene los datos personales del paciente. Si se sale de la aplicación sin rellenar estos datos (dándole al botón HOME para volver al menú principal del móvil), al volver a abrirla, se mostrará de nuevo esta pantalla hasta que se introduzcan los datos y se presione “GUARDAR”.

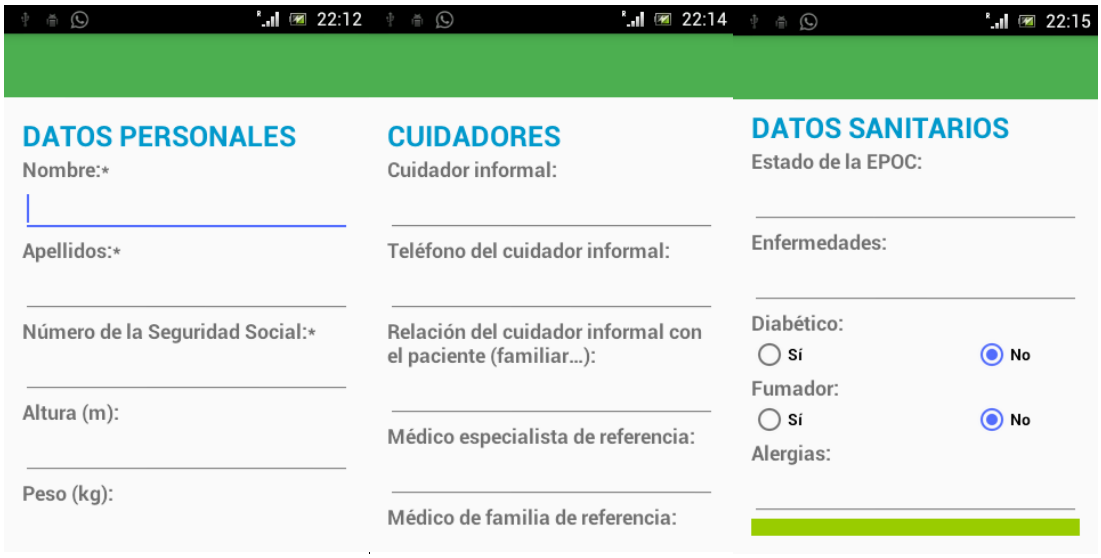
Para ello, cuando se instala la aplicación, si se presiona el botón de “GUARDAR” al introducir los datos personales, se ejecuta este código:

```
SharedPreferences prefs = getSharedPreferences("myPrefs", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = prefs.edit();
editor.putBoolean("datosIntroducidos", true);
editor.commit();
```

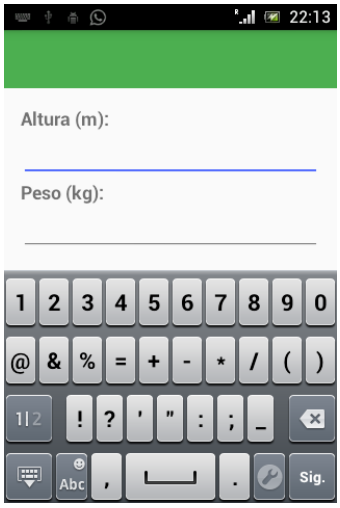
En el que se guarda en el objeto “prefs” el valor “true” para la clave “datosIntroducidos”. Cuando se lanza la aplicación, en el MainActivity (que es la actividad que se lanza cuando se abre la aplicación), se obtiene las preferencias compartidas (SharedPreferences) y se comprueba el valor de la variable datosIntroducidos. Si este está a “false”, quiere decir que aún no se han introducido los datos personales, por lo que vuelve a lanzar la actividad de introducir loa datos del paciente (“IntroducirDatosPaciente”).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    SharedPreferences preferences = getSharedPreferences("myPrefs",
Context.MODE_PRIVATE);
    boolean condicionesAceptadas = preferences.getBoolean("terminoAceptado", false);
    boolean datosIntroducidos = preferences.getBoolean("datosIntroducidos", false);
    if (!condicionesAceptadas) {
        Intent myIntent = new Intent(this, TermsAndConditions.class);
        startActivity(myIntent);
    }else if (!datosIntroducidos) {
        Intent myIntent = new Intent(this, IntroducirContrasena.class);
        startActivity(myIntent);
    }

    SharedPreferences.Editor editor = preferences.edit();
    editor.putString("databaseName", "retoEPOC_DB.db"); // value to store
    editor.commit();
```

En esta actividad se piden tanto datos personales del paciente, como de sus cuidadores o de su enfermedad.



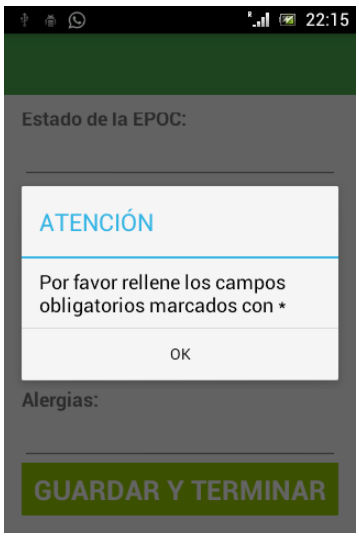
Al introducir datos numéricos como puede ser la altura o el peso, se inicia el teclado numérico directamente.



Esto es gracias a que en el xml del layout asociado a esta vista, se ha puesto que el inputType sea numérico:

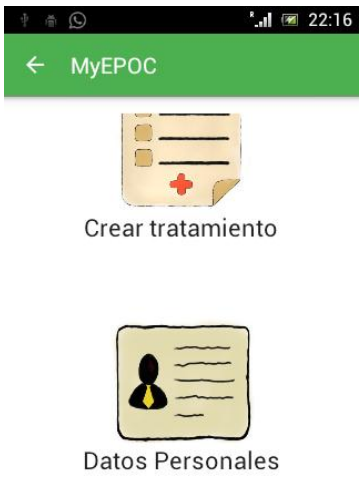
```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editTextAltura"
    android:inputType="numberDecimal"
    android:textStyle="bold" />
```

Hay tres datos que son obligatorios y que están marcados con un asterisco (*), si se intenta guardar los datos sin introducir estos, saltará un aviso.



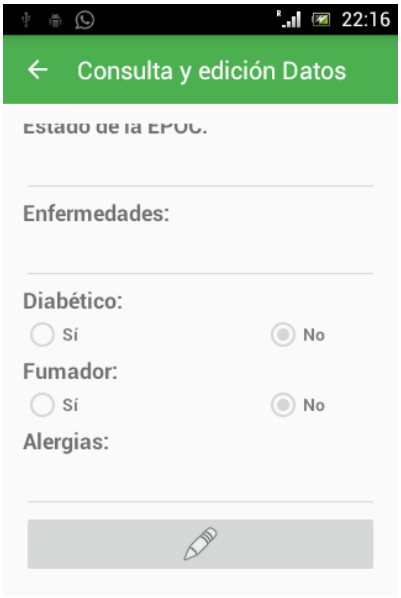
Una vez introducidos estos datos obligatorios, ya se puede guardar y disfrutar de la aplicación.

Desde la sección reservada para el personal sanitario, se pueden consultar los datos del paciente.



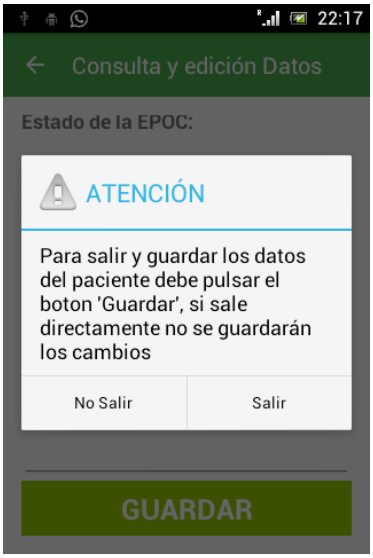
Para la consulta y edición de los datos, se ha utilizado el mismo layout que para insertarlos. Solo que se ha cambiado, de forma dinámica, el visibilidad y editabilidad de los campos.
Si se han introducido el peso (kg) y la altura (m) del paciente, cuando el personal sanitario vaya a consultar los datos, podrá ver el IMC que se calcula automáticamente.

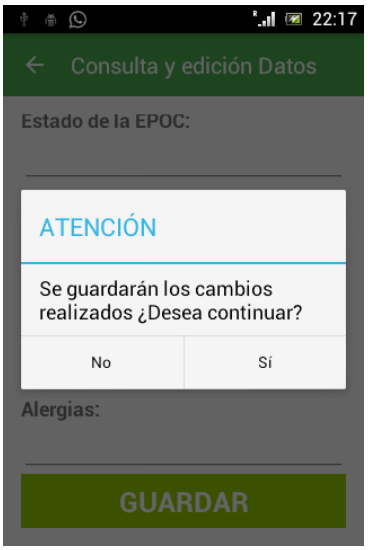
Dentro de la vista de consultar los datos, estos se podrán editar. Para ello es necesario pulsar el botón de editar al final del scroll.



Al pulsar este botón, como se ha mencionado anteriormente, se hacen editables los campos, pero se usa el mismo layout que al insertar o consultar.

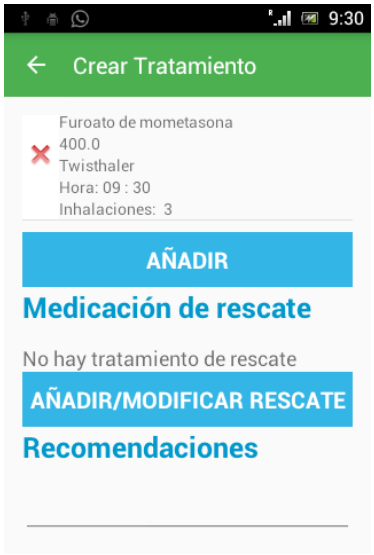
De esta forma se pueden editar los datos del paciente. Si se está en modo edición y se intenta salir sin presionar el botón guardar, se lanzará un aviso advirtiendo de que no se guardarán os cambios. Si en cambio se pulsa el botón de guardar, se lanzará un aviso advirtiendo de que se guardarán os cambios.





2.3 Crear y borrar el tratamiento de rescate y las recomendaciones desde la parte reservada al personal sanitario.

Desde la parte de crear tratamiento, en la sección reservada al personal sanitario, debajo de la lista de tomas, está la opción de añadir un tratamiento de rescate. Dicho tratamiento, será consultado desde la ventana principal de la aplicación por el paciente en caso de que sufra una crisis. Si hasta el momento no se ha configurado, aparecerá un mensaje informando de esto.



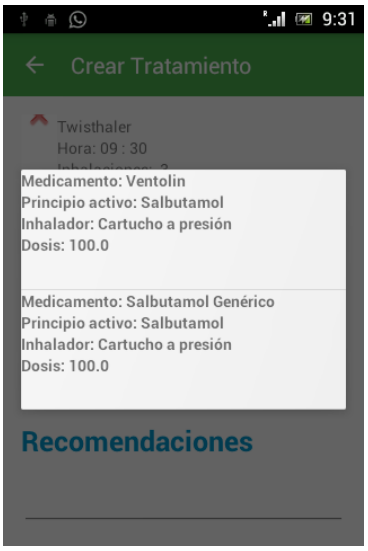
Para saber si hay tratamiento de rescate o no, se mira el id de este dentro de la tabla tratamiento. Cuando se inicializa la base de datos, se crean todas las tablas, y se inserta un nuevo tratamiento, con id = 1, un String vacío para las recomendaciones y un id = -1, este último corresponde al tratamiento de rescate.

```
public void onCreate(SQLiteDatabase db) {
    db.execSQL(SQL_CREATE_MEDICINE_TABLE);
    db.execSQL(SQL_CREATE_ACTIVEINGREDIENT_TABLE);
    db.execSQL(SQL_CREATE_PHYSIOTERAPY_TABLE);
    db.execSQL(SQL_CREATE_INHALER_TABLE);
    db.execSQL(SQL_CREATE_POSOLOGY_TABLE);
    db.execSQL(SQL_CREATE_MEDICINE_PRESENTATION_TABLE);
    db.execSQL(SQL_CREATE_USERMANUALSTEP_TABLE);
    db.execSQL(SQL_CREATE_USERMANUALPHYSIOMANUALSTEP_TABLE);
    db.execSQL(SQL_CREATE_RESCUE_TABLE);
    db.execSQL(SQL_CREATE_TREATMENT_TABLE);
    db.execSQL(SQL_CREATE_TAKE_TABLE);
    db.execSQL(SQL_CREATE_PATIENTPERSONALDATA_TABLE);
    db.execSQL(SQL_CREATE_APPOINTMENT_TABLE);
    db.execSQL(SQL_CREATE_ANALYSIS_TABLE);
    db.execSQL(SQL_CREATE_ESPIROMETRIA_TABLE);

    insertTreatmentRow(db, 1, "", -1);

    fillInitialData(db);
}
```

Gracias a esto, cuando se lanza la actividad de CrearTratamiento, donde se configura el rescate, se obtiene el tratamiento general insertado anteriormente, y si el id de rescate es -1, quiere decir que aún no hay ninguno configurado, en otro caso, el id tendrá el valor que le corresponde.
Según esto, se ponen visibles o escondidos los diferentes elementos de la interfaz.

```
spinnerEmergencia.setVisibility(View.GONE);

if (tratamiento.getRescate() == -1) {
    medicinaR.setVisibility(View.GONE);
    principioR.setVisibility(View.GONE);
    inhaladorR.setVisibility(View.GONE);
    dosisR.setVisibility(View.GONE);
    noRescate.setVisibility(View.VISIBLE);
    noRescate.setText(R.string.noSeleccRescate);
    botonRescate.setVisibility(View.VISIBLE);
    botonBorrarRescate.setVisibility(View.GONE);

} else {
    PresentacionMedicacionRescate rescate =
modelo.getRescueById(tratamiento.getRescate());
    botonRescate.setVisibility(View.GONE);
    botonBorrarRescate.setVisibility(View.VISIBLE);
    noRescate.setVisibility(View.GONE);
    medicinaR.setVisibility(View.VISIBLE);
    principioR.setVisibility(View.VISIBLE);
    inhaladorR.setVisibility(View.VISIBLE);
    dosisR.setVisibility(View.VISIBLE);
    medicinaR.append(": " + Commodities.retrieveResourceStringFromId(this,
rescate.getMedicine()));
    principioR.append(": " + Commodities.retrieveResourceStringFromId(this,
rescate.getActiveIngredient()));
    inhaladorR.append(": " + Commodities.retrieveResourceStringFromId(this,
rescate.getInhaler()));
    if (rescate.getDose_ai2() == 0) {
        dosisR.append(": " + rescate.getDose_ai1());
    } else {
        dosisR.append(": " + Float.toString(rescate.getDose_ai1()) + " / " +
                Float.toString(rescate.getDose_ai2()));
    }

}
```

El presionar el botón “AÑADIR/MODIFICAR RESCATE”, desaparece el texto informativo, y aparece una lista desplegable, donde se puede elegir que tratamiento se desea configurar como medicación de rescate.



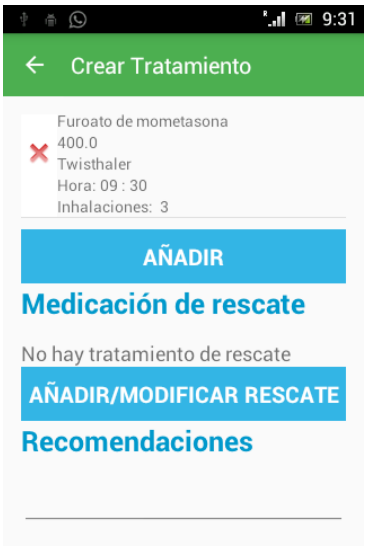
Una vez elegido uno, se debe pulsar el botón de “GUARDAR” para conservar los cambios y almacenarlos en la base de datos (igual que a la hora de gestionar las tomas).

Al presionar sobre “GUARDAR”, se comprueba la visibilidad del mensaje informativo de que no hay rescate configurado. Si está escondida, quiere decir que hay un tratamiento de rescate configurado, por lo que se añade su id a la base de datos, en otro caso, se guardará un -1.

```
Treatment t;
if (noRescate.getVisibility() == View.VISIBLE) {
    t = new Treatment(1, recomendaciones.getText().toString(), -1);
} else {
    t = new Treatment(1, recomendaciones.getText().toString(), rescateId);
}
```

El mensaje informativo, solo es visible cuando no hay ningún rescate configurado. De esta forma, según su visibilidad (VISIBLE o GONE) se puede saber si hay rescate o no, y almacenarlo en la base de datos. A la hora de guardarlo, se machaca el tratamiento anterior almacenado. Como solo teneos un tratamiento, siempre tiene id = 1.

En el caso de que ya exista un tratamiento de rescate, cuando accedemos a esta vista, se podrá observar dicho tratamiento. También se podrá borrar y añadir otro. Al pulsar sobre el botón “BORRAR RESCATE”, este se borra y aparece de nuevo la vista sin medicación de rescate, donde aparece el texto informativo y en la que se puede añadir uno. Igual que para cualquier cambio, hay que pulsar el botón guardar para conservar los cambios.



Los tratamientos de rescate no se pueden editar, ya que son configuraciones fijas puestas por médicos especializados en esto.

2.4 Consultar el tratamiento de rescate desde la interfaz principal.

Desde la vista principal de la aplicación, el paciente podrá acceder a consultar el tratamiento de rescate que el médico le haya configurado previamente. Si no tiene ningún tratamiento de rescate asociado, aparecerá un mensaje informando de esto.

Se sabe si no hay ningún rescate asociado obteniendo el tratamiento general, cuya id es siempre 1, y evaluando el id asociado al tratamiento de rescate. Si este es -1, significa que no hay ninguno configurado.

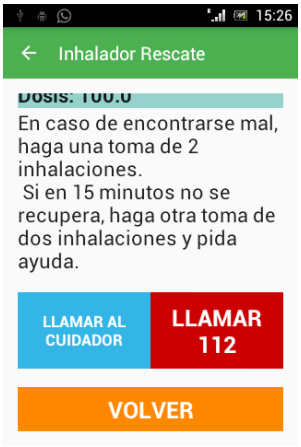
```
Treatment t = modelo.getTreatmentById(1);

if (t.getRescate() == -1) {
    nombre.setVisibility(View.GONE);
    imagen.setVisibility(View.GONE);
    medicina.setVisibility(View.GONE);
    dosis.setVisibility(View.GONE);
```

```
    texto.setVisibility(View.GONE);
    noRescate.setVisibility(View.VISIBLE);
```

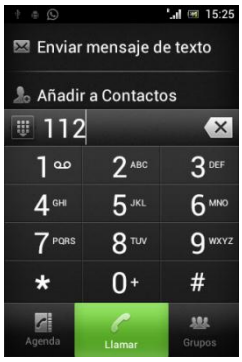
En el caso de que si tenga un tratamiento de rescate, se obtiene este de la base de datos,

```
PresentacionMedicacionRescate p = modelo.getRescueById(t.getRescate());
```

y se muestra el nombre del inhalador, una foto de este, el nombre del medicamento, la dosis que se deberá tomar y unas instrucciones que el paciente deberá seguir.



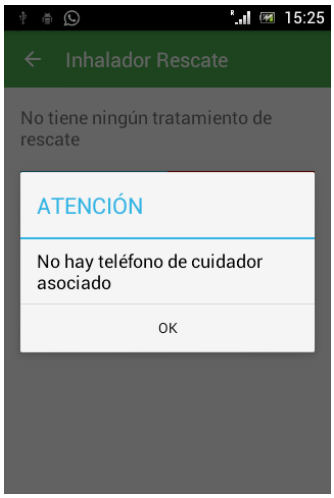
Además de todo esto, desde esta sección, el paciente podrá llamar a su cuidador o a emergencias. Al pulsar sobre cualquier botón, se abrirá la ventana para poder llamar con el número marcado directamente.



Para esto, se usa un Intent, al que se le pasa como parámetro el número al que se desea llamar. Se explicará con detalle más adelante.

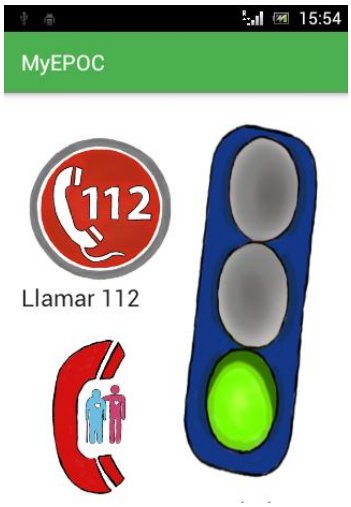
```
public static void callEmergency(Context context) {
    Intent intent = new Intent(Intent.ACTION_DIAL, Uri.fromParts("tel",
            context.getString(R.string.emergencyPhoneNumber), null));
    context.startActivity(intent);
}
```

En el caso de que no se haya introducido un número de algún cuidador en los datos del paciente, la aplicación avisará de esto.



2.5 Botón para llamar al 112.

Al igual que en la vista del tratamiento de rescate, también se facilita un botón para llamar a emergencias en la vista principal de la aplicación.



Al presionar este botón, se abre la vista lista para llamar con el número de emergencias marcado.
Como se ha mostrado anteriormente, se ejecuta este código:

```
public static void callEmergency(Context context) {
    Intent intent = new Intent(Intent.ACTION_DIAL, Uri.fromParts("tel",
            context.getString(R.string.emergencyPhoneNumber), null));
    context.startActivity(intent);
}
```

Para poder abrir la ventana de llamar, es necesario pasarle al Intent, la variable ACTION_DIAL, que al contrario de ACTION_CALL, esta no necesita permiso para poder llamar, ya que lo que hace es marcar el número, y no llamar directamente.

El según parámetro que se le pasa al Intent es un objeto de la clase Uri (Uniform Resource Identifier), al que le pasamos el String “tel”, para referenciar que le vamos a pasar un número de teléfono y dicho número de teléfono al que se desea llamar.

2.6 Mejora de la interfaz gráfica de toda la aplicación cambiando los estilos de las letras y los botones.

En las primeras versiones de la aplicación, había muchas diferencias entre una vista y otra, y los botones eran todos iguales, con el formato que trae android por defecto.

En las últimas versiones, cambié el estilo de casi todas las letras, para que tuviese coherencia toda la aplicación, y cambié la vista de los botones siguiendo un código de colores. Verde para guardar, azul para añadir, naranja para volver, rojo para borrar, amarillo para añadir horas…

2.7 Base de datos

2.7.1 Primera versión de la BD (clase Modelo.java).

[11]. En Android, la forma típica para gestionar una base de datos SQLite es mediante una clase auxiliar llamada SQLiteOpenHelper, o una clase propia que herede de ella.

```
public class DBHelper extends SQLiteOpenHelper {
```

Esta clase (DBHelper) será una subclase dentro de Modelo.java. La clase SQLiteOpenHelper tiene un constructor, que no necesitaremos sobrescribir,

```
public DBHelper(Context context, String database_name) {
    super(context, database_name, null, DATABASE_VERSION);
}
```

y dos métodos abstractos, onCreate() y onUpgrade(). Estos dos métodos debemos personalizarlos con el código necesario para crear o actualizar nuestra base de datos.

El método onCreate() será ejecutado automáticamente por nuestra clase DBHelper cuando sea necesaria la creación de la base de datos, es decir, cuando aún no exista. Las tareas típicas que deben hacerse en este método serán la creación de todas las tablas necesarias y la inserción de los datos iniciales si son necesarios. En la aplicación, al ejecutar este método, se crean todas las tablas necesarias y se inserta un tratamiento (el tratamiento general que contiene un String con las recomendaciones, y el id del tratamiento de rescate).

```
public void onCreate(SQLiteDatabase db) {
    db.execSQL(SQL_CREATE_MEDICINE_TABLE);
    db.execSQL(SQL_CREATE_ACTIVEINGREDIENT_TABLE);
    db.execSQL(SQL_CREATE_PHYSIOTERAPY_TABLE);
    db.execSQL(SQL_CREATE_INHALER_TABLE);
    db.execSQL(SQL_CREATE_POSOLOGY_TABLE);
    db.execSQL(SQL_CREATE_MEDICINE_PRESENTATION_TABLE);
    db.execSQL(SQL_CREATE_USERMANUALSTEP_TABLE);
    db.execSQL(SQL_CREATE_USERMANUALPHYSIOMANUALSTEP_TABLE);
    db.execSQL(SQL_CREATE_RESCUE_TABLE);
    db.execSQL(SQL_CREATE_TREATMENT_TABLE);
    db.execSQL(SQL_CREATE_TAKE_TABLE);
    db.execSQL(SQL_CREATE_PATIENTPERSONALDATA_TABLE);
    db.execSQL(SQL_CREATE_APPOINTMENT_TABLE);
    db.execSQL(SQL_CREATE_ANALYSIS_TABLE);
    db.execSQL(SQL_CREATE_ESPIROMETRIA_TABLE);

    insertTreatmentRow(db, 1, "", -1);

    fillInitialData(db);
}
```

Los Strings que se le pasan al método execSQL, son las sentencias SQL previamente definidas, con las que se crean las tablas.

```
private static final String SQL_CREATE_TREATMENT_TABLE =
        "CREATE TABLE " + ModeloContract.TreatmentTable.TABLE_NAME + " (" +
                ModeloContract.TreatmentTable._ID + " INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT, " + ModeloContract.TreatmentTable.RECOMMENDATION + " TEXT, " +
                ModeloContract.TreatmentTable.RESCUE_ID + " INTEGER NOT NULL, " +
                "FOREIGN KEY(" + ModeloContract.TreatmentTable.RESCUE_ID + ") REFERENCES
" + ModeloContract.InhalerTable.TABLE_NAME + "(" + ModeloContract.InhalerTable.NAME +
")" + " )";
```

El método onUpgrade() se lanzará automáticamente cuando sea necesaria una actualización de la base de datos. Para ello, como parámetros recibe la versión actual de la base de datos en el sistema, y la nueva versión a la que se quiere convertir. Al ejecutar este método, se borra la base de datos anterior, y se llama al método onCreate(SQLiteDataBase db) con la nueva base de datos.

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // This database is only a cache for online data, so its upgrade policy is
    // to simply to discard the data and start over
    db.execSQL(SQL_DELETE_ENTRIES);
    onCreate(db);
}
```

Una vez definida la clase DBHelper, la apertura de la base de datos desde nuestra aplicación se hace de la siguiente forma. Lo primero será crear un objeto de esta clase (DBHelper). En el método init(Context ctx) al que pasaremos el contexto de la aplicación, se comprueba que las variables app_context y database_name, inicializadas anteriormente, son null, y que el objeto Context pasado por parámetro es diferente de null.

Tras esto, se obtiene el nombre de la base de datos, y se inicializa el objeto de la clase DBHelper, al que se le pasa el contexto, y el nombre de la base de datos.

```
public class Modelo {

    private DBHelper dbHelper = null;
    private SQLiteDatabase db = null;
    private Context app_context = null;
    private String database_name = null;

    public void init(Context ctx) {
        if (app_context == null && database_name == null && ctx != null) {
            app_context = ctx.getApplicationContext();

            SharedPreferences preferences = app_context.getSharedPreferences("myPrefs",
Context.MODE_PRIVATE);
            String database_name = preferences.getString("databaseName", "");

            SQLiteDatabase.loadLibs(ctx);
            dbHelper = new DBHelper(app_context, database_name);
            if (dbHelper != null) {
                String password = preferences.getString("contrasena", "");
                db = dbHelper.getWritableDatabase(password);
            }
        }
    }
```

La simple creación de este objeto puede tener varios efectos:

- Si la base de datos ya existe y su versión actual coincide con la solicitada simplemente se realizará la conexión con ella.
- Si la base de datos existe pero su versión actual es anterior a la solicitada, se llamará automáticamente al método onUpgrade() para convertir la base de datos a la nueva versión y se conectará con la base de datos convertida.

- Si la base de datos no existe, se llamará automáticamente al método onCreate() para crearla y se conectará con la base de datos creada.

Una vez obtenida una referencia al objeto DBHelper, se llama a su método getWritableDatabase() para obtener una referencia a la base de datos, en la que podemos hacer modificaciones. Si solo deseáramos leer, se llamaría al método getReadableDatabase().

En la clase Modelo.java, a parte de la subclase DBHelper, hay otra subclase para poder implementar el patrón singleton con la base de datos, y un método getInstance(), que devuelve una instancia de la propia clase.

```
private static class SingletonHolder {
    public static final Modelo INSTANCE = new Modelo();
}

public static Modelo getInstance() {
    return SingletonHolder.INSTANCE;
}
```

De esta forma, la primera vez que se invoque a este método, creará una nueva instancia de la clase Modelo.java, pero posteriormente, devolverá dicha instancia en lugar de crear una nueva.

En las Activitys, esta es la forma de inicializar el objeto Modelo, mediante el getInstance(), y una vez se obtiene la instancia, se puede llamar a init(Context) para inicializar la conexión con la base de datos.

```
modelo = Modelo.getInstance();
modelo.init(this);
```

Todas las bases de datos SQLite creadas por aplicaciones Android utilizando este método se almacenan en la memoria del teléfono en un fichero con el mismo nombre de la base de datos situado en una ruta que sigue el siguiente patrón:

/data/data/paquete java de la aplicación/databases/nombre de la base datos

2.7.2 Algunos métodos para insertar, consultar, borrar y editar de la BD los tratamientos de rescate, datos personales del paciente, tratamientos del paciente (tomas) y espirometrías.

En la clase Modelo.java, además de todo lo explicado anteriormente, también están implementados los métodos para insertar, borrar, consultar o actualizar los datos de las tablas. Para poder gestionar los datos, he tenido que crear varios métodos, por ejemplo para obtener la lista de tomas:

```
//Cargar la lista de tomas
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()) {
        //DB Table Take: _id, treatment_id, MedicineTable.name,
ActiveIngredientTable.name,
        // InhalerTable.name, dose_id, posology_id
        //    public Toma(int id, int medicamento, int principio, int inhalador, int
dosis, int posologia)
        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;
}
```

De esta forma, en el resto de clases, se puede acceder a estos métodos una vez se sincronice el objeto Modelo con la base de datos.

```
ArrayList<Toma> listaTomas = modelo.obtenerListaTomas();
```

2.7.3 Creación de algunas entidades de persistencia.

Aparte de los métodos para gestionar los datos de la base de datos, también he tenido que crear las entidades de persistencia para poder gestionar los objetos obtenidos desde la base de datos a través del código. He creado, entre otras, las clases Dosis.java, Espirometria.java, Hour.java, Inhalador.java, Medicamento.java, Toma.java…

Si en alguna parte del código, es necesario pasar un objeto (o una lista de objetos) de alguna clase, ha sido necesario hacer que estas clases implementen Parcelable, con los métodos que esto conlleva.

```
public class Toma implements Parcelable{

protected Toma(Parcel in) {
    id = in.readInt();
    medicamento = in.readString();
    principio = in.readString();
    inhalador = in.readString();
    dosis = in.readParcelable(Dosis.class.getClassLoader());
    hour = in.readParcelable(Hour.class.getClassLoader());
    done = in.readByte() != 0;
    inhalations = in.readInt();
    posMedicamento = in.readInt();
    posPrincipio = in.readInt();
    posInhalador = in.readInt();
    posDosis = in.readInt();
    posInhalations = in.readInt();
}

public static final Creator<Toma> CREATOR = new Creator<Toma>() {
    @Override
    public Toma createFromParcel(Parcel in) {
        return new Toma(in);
    }

    @Override
    public Toma[] newArray(int size) {
        return new Toma[size];
    }
};

@Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeInt(id);
        dest.writeString(medicamento);
        dest.writeString(principio);
        dest.writeString(inhalador);
        dest.writeParcelable(dosis, flags);
        dest.writeParcelable(hour, flags);
        dest.writeByte((byte) (done ? 1 : 0));
        dest.writeInt(inhalations);
        dest.writeInt(posMedicamento);
        dest.writeInt(posPrincipio);
        dest.writeInt(posInhalador);
        dest.writeInt(posDosis);
        dest.writeInt(posInhalations); }
```

Cada entidad de persistencia, contiene un constructor, y todos los getters y setters necesarios para gestionarlas.En algunas entidades, ha sido necesario redefinir el método toString(), por ejemplo en la clase Hour.java,

```
@Override
public String toString(){

    String representation;
    if(hour < 10){
     representation   ="0" + Integer.toString(hour);

    }else{
        representation    = Integer.toString(hour);

    }
    if(min < 10){
        representation += " : 0" + Integer.toString(min);
    }else{
        representation += " : " + Integer.toString(min);
    }

    return representation;
}
```

o el compareTo(), por ejemplo en la clase Espirometria.java

```
@Override
public int compareTo(Object another) {
    Espirometria e = (Espirometria) another;
    return getCalendar().compareTo(e.getCalendar());
}
```

2.8 Funcionalidades implementadas junto a Diego Narbona Vílchez:

2.8.1 Creación de las alarmas.

A la hora de crear las tomas, también se le asocia una alarma. Dicha alarma sonará a la hora que el paciente tenga que tomarse la dosis.

La alarma se crea cuando se presiona sobre el botón de guardar en la activity CrearTratamiento. Se crean dos objetos Calendar para cada toma, uno se inicializa con la hora introducida en su creación, y el otro con la fecha del móvil.

```
for (Toma t : listaTratamientosNuevos) {
    modelo.insertarToma(t);

    Calendar now = Calendar.getInstance();
    Calendar alarm = Calendar.getInstance();
    alarm.set(Calendar.HOUR_OF_DAY, t.getHour().getHour());
    alarm.set(Calendar.MINUTE, t.getHour().getMin());
    alarm.set(Calendar.SECOND, 0);
```

En el caso que la alarma sea para el día siguiente, se añade un día la alarma de la toma.

```
    if (alarm.before(now)) alarm.add(Calendar.DAY_OF_MONTH, 1);
```

Antes de crear la alarma, se comprueba la versión del android que tiene instalada el móvil. Para versiones anteriores a KITKAT (API 19), se ejecuta el alarmManager.set(), y para las demas versions, el alarmManager.setExact().

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.KITKAT) {
    alarmManager.set(AlarmManager.RTC_WAKEUP, alarm.getTimeInMillis(), pendingIntent);
} else {
    alarmManager.setExact(AlarmManager.RTC_WAKEUP, alarm.getTimeInMillis(),
pendingIntent);
}
```

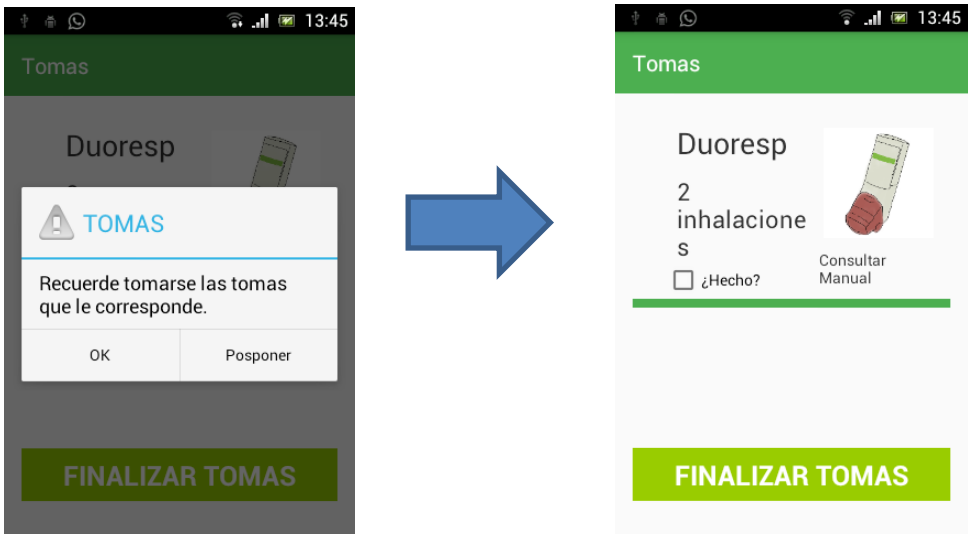
A partir de la API 19, la entrega de alarma es inexacta, el sistema operativo no comprueba a cada instante si hay alguna alarma programada con el fin de disminuir el uso de la batería. Para aplicaciones que necesitan estrictas garantías de entrega, existe el método setExact() .

Para las versiones anterior a la API 19, continua el comportamiento en el que todas las alarmas se entregan exactamente cuando se le solicite.

De esta forma se garantiza que la alarma sonará a la hora acordada, y el paciente podrá seguir un tratamiento de manera más eficiente. Al sonar la alarma, el móvil sonará y vibrará. Para esto, es necesario introducir en el AndroidManifest.xml, los distintos permisos.

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
```

Al sonar la alarma, el móvil mostrará una pantalla con información sobre la toma que se debe realizar, y un checkBox donde se puede indicar si la toma se ha hecho o no. Este checkBox servirá para determinar el color de semáforo (explicado más abajo).



2.8.2 Acción de cambio de color del semáforo.

En la ventana principal de la aplicación, el usuario podrá ver si está haciendo bien el tratamiento o no mediante un código de colores reflejado en un semáforo. El semáforo se pone rojo, si el usuario no se ha tomado ninguna toma, si se toma alguna, el semáforo se pondrá amarillo, y si se las toma todas, el semáforo se pondrá verde.

Siguiendo este código de colores, su cuidador también puede saber cómo lleva el tratamiento diariamente. Al principio del día, el semáforo estará rojo, puesto que aún no se ha tomado ninguna, y solo se pondrá verde cuando se tome la última toma (en el caso de que se haya tomado todas las anteriores).

Para esto, en el onResume() del MainActivity, se obtiene la lista de tomas, y según se hayan tomado ninguna, alguna o todas, se pone una imagen u otra.

```
ImageButton image = (ImageButton) findViewById(R.id.semaforo);
if (!password.equals("")) {

    Modelo modelo = Modelo.getInstance();
    modelo.init(this);

    ArrayList<Toma> listaTomas = modelo.obtenerListaTomas();
    int contador = 0;
    for (Toma t : listaTomas) {
        if (t.isDone()) {
            contador++;
        }
    }
    if (contador == listaTomas.size()) {
        image.setImageResource(R.drawable.lightgreen);
    } else if (contador > 0 && contador < listaTomas.size()) {
        image.setImageResource(R.drawable.lightyellow);
    } else if (contador == 0) {
        image.setImageResource(R.drawable.lightred);
    }
} else {
    image.setImageResource(R.drawable.lightgreen);
}
```

3 Segunda parte:

3.1 Editar las tomas configuradas desde la parte reservada al personal sanitario.

Desde la parte reservada al personal sanitario, se puede acceder a crear tratamientos. Una vez se hayan creado, aparecen listados en esta vista, donde al pulsarlos, se abre una nueva ventana con la información más detallada (punto 1.1).

Dentro de esta vista detallada de la toma, existe la opción de editarla. Para ello es preciso pulsar el botón “EDITAR”. Al presionar sobre este botón, se lanza la vista donde se añadían las tomas, pero esta vez, aparecerán cargados los datos de la toma que queremos editar.

Al presionar sobre el botón “EDITAR”, se ejecuta el siguiente código:

```
public void onClickEditar(View view) {
    Intent intentEditar = new Intent(this, AddNewTake.class);
    intentEditar.putExtra("position",position);
    intentEditar.putParcelableArrayListExtra("listaTomas",listaTomas);
    startActivityForResult(intentEditar, 1);
}
```

En el que se lanza la Activity AddNewTake, con código = 1, y a la que se le pasa la lista de todas las tomas, previamente obtenida (como se explica en el punto 1.1) y la posición de la toma que se va a editar dentro de esta lista.

En el onCreate de la Activity AddNewTake, se capturan estas dos variables,

```
listaTomasNuevas = getIntent().getParcelableArrayListExtra("listaTomas");
pos = getIntent().getExtras().getString("position");
```

y se cargan en los desplegables los datos de la toma a editar. Al ser la posición diferente de “-1”, (como se explica en el punto 1.1), se sabe que se está editando y no añadiendo una toma nueva.

```
if (!pos.equals("-1")) {
    spinnerMedicamento.setSelection(listaTomasNuevas.get(Integer.valueOf(pos)).
getPosMedicamento());

    listaHoras.add(listaTomasNuevas.get(Integer.valueOf(pos)).getHour().toString());
    adapterHoras.notifyDataSetChanged();
    Commodities.ajustarVistaListView(listViewHoras);
    spinnerInhalaciones.setSelection(listaTomasNuevas.get(Integer.valueOf(pos)).
getPosInhalations());

    refreshDependableSpinners();

    spinnerPrincipio.setSelection(listaTomasNuevas.get(Integer.valueOf(pos)).
getPosPrincipio());
    spinnerInhaladores.setSelection(listaTomasNuevas.get(Integer.valueOf(pos)).
getPosInhalador());
    spinnerDosis.setSelection(listaTomasNuevas.get(Integer.valueOf(pos)).getPosDosis());

    Button bAdd = (Button) findViewById(R.id.buttonAddNuevaToma);
    bAdd.setText(R.string.botonTerminarEditar);
    bAdd.setBackgroundColor(Color.rgb(153, 204, 0));
    Button bVolver = (Button) findViewById(R.id.buttonVolverTomas);
    bVolver.setText(R.string.botonCancelarEditar);
    Button bAddHora = (Button) findViewById(R.id.buttonAddHour);
    bAddHora.setText(R.string.botonEditarHora);
    listViewHoras.setVisibility(View.GONE);
    textViewHoras.setVisibility(View.VISIBLE);
    textViewHoras.setText(listaHoras.get(0));
}
```

Se cambia el texto del botón “AÑADIR HORA” por “CAMBIAR HORA”, ya que desde esta parte solo se puede cambiar la hora de la toma, no se podrá añadir más de una hora, porque esto causaría que se creen tomas nuevas. También

se cambia el texto al botón “AÑADIR” por “TERMINAR EDICIÓN” y el de “VOLVER” por “ CANCELAR EDICIÓN”, pero los métodos que los manejan siguen siendo los mismos.

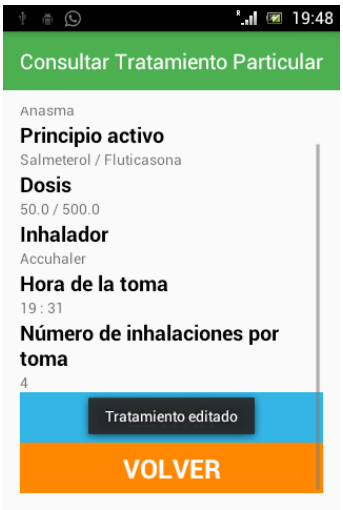
Al presionar sobre el botón “TERMINAR EDICIÓN”, si todos los campos han sido correctamente completados, se vuelve a comprobar si se está editando o no mediante el valos de la variable “pos”, y se actualiza la toma que está siendo editada con los nuevos valores.

```
if (!pos.equals("-1")) {
    Hour h = new Hour(Integer.parseInt(textViewHoras.getText().toString().substring(0,
2)),

Integer.parseInt(textViewHoras.getText().toString().substring(textViewHoras.getText().
                    toString().length() - 2,
textViewHoras.getText().toString().length())));
    int index = Integer.valueOf(pos);
    listaTomasNuevas.get(index).setMedicamento(m.getName());
    listaTomasNuevas.get(index).setPrincipio(p.getName());
    listaTomasNuevas.get(index).setInhalador(in.getName());
    listaTomasNuevas.get(index).setDosis(d);
    listaTomasNuevas.get(index).setHour(h);
    listaTomasNuevas.get(index).setInhalations(numInhalaciones);
    listaTomasNuevas.get(index).setPosMedicamento(spinnerMedicamento.
getSelectedItemPosition());
listaTomasNuevas.get(index).setPosPrincipio(spinnerPrincipio.getSelectedItemPosition());
    listaTomasNuevas.get(index).setPosInhalador(spinnerInhaladores.
getSelectedItemPosition());
    listaTomasNuevas.get(index).setPosDosis(spinnerDosis.getSelectedItemPosition());
    listaTomasNuevas.get(index).setPosInhalations(spinnerInhalaciones.
getSelectedItemPosition());

}
```

Tras esto, se vuelve a la vista detallada de la toma, a la que se le devuelve la lista de tomas y la posición de la lista que ha sido editada,

```
Intent newTakes = new Intent();
if (!pos.equals("-1")) {
    Toast pulsarAdd = Toast.makeText(this, R.string.toastAddTratamientoEditado,
Toast.LENGTH_SHORT);
    pulsarAdd.show();
    newTakes.putParcelableArrayListExtra("newTake", listaTomasNuevas);
    newTakes.putExtra("position", pos);
}

setResult(Activity.RESULT_OK, newTakes);

this.onBackPressed();
```

y la aplicación mostrará un mensaje indicando que se ha editado la toma.



Al volver a la vista detallada, se invoca el método onActivityResult(), donde se recoge la lista de tomas y la posición y se actualizan los valores de la toma editada.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // Comprobamos si el resultado de la segunda actividad es "RESULT_CANCELED".
    if (resultCode == Activity.RESULT_OK && requestCode == 1) {
        listaEditada = data.getParcelableArrayListExtra("newTake");
        position = data.getExtras().getString("position");

        medicamento.setText(Commodities.retrieveResourceStringFromId(this,listaEditada.
get(Integer.valueOf(position)).getMedicamento()));
        principio.setText(Commodities.retrieveResourceStringFromId(this,listaEditada.
get(Integer.valueOf(position)).getPrincipio()));
dosis.setText(listaEditada.get(Integer.valueOf(position)).getDosis().toString());
        inhalador.setText(Commodities.retrieveResourceStringFromId(this,listaEditada.
get(Integer.valueOf(position)).getInhalador()));
        hora.setText(listaEditada.get(Integer.valueOf(position)).getHour().toString());
inhalaciones.setText(Integer.toString(listaEditada.get(Integer.valueOf(position)).
getInhalations()));

        int index = Integer.valueOf(position);

listaTomas.get(index).setMedicamento(listaEditada.get(Integer.valueOf(position)).
getMedicamento());
        listaTomas.get(index).setPrincipio(listaEditada.get(Integer.valueOf(position)).
getPrincipio());
        listaTomas.get(index).setInhalador(listaEditada.get(Integer.valueOf(position)).
getInhalador());
        listaTomas.get(index).setDosis(listaEditada.get(Integer.valueOf(position)).
getDosis());
        listaTomas.get(index).setHour(listaEditada.get(Integer.valueOf(position)).
getHour());
listaTomas.get(index).setInhalations(listaEditada.get(Integer.valueOf(position)).
getInhalations());
listaTomas.get(index).setPosMedicamento(listaEditada.get(Integer.valueOf(position)).
getPosMedicamento());
listaTomas.get(index).setPosPrincipio(listaEditada.get(Integer.valueOf(position)).
getPosPrincipio());
listaTomas.get(index).setPosInhalador(listaEditada.get(Integer.valueOf(position)).
getPosInhalador());
        listaTomas.get(index).setPosDosis(listaEditada.get(Integer.valueOf(position)).
```

```
getPosDosis());
listaTomas.get(index).setPosInhalations(listaEditada.get(Integer.valueOf(position)).
getPosInhalations());


        ok = true;

    }
}
```

Tras actualizar los valores, se pone la variable global “ok” a true. Esto servirá para que a la hora de volver de la vista detallada a la Activity CrearTratamiento, se envíe la nueva lista con la toma editada, y la posición de dicha toma.

```
public void onClickBotonVolver(View view) {

    if(ok) {
        Intent newTakes = new Intent();
        newTakes.putParcelableArrayListExtra("listaEditada", listaEditada);
        newTakes.putExtra("position", position);
        setResult(Activity.RESULT_OK, newTakes);
    }
    this.onBackPressed();
}
```

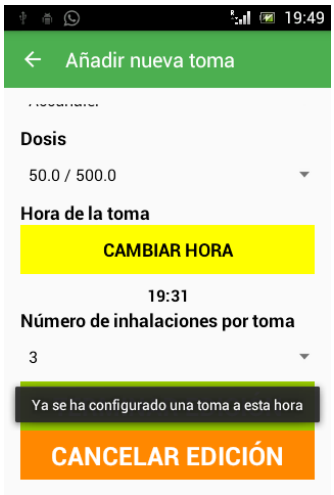
Al volver a CrearTratamiento, se ejecuta el método onActivityResult, pero esta vez, la actividad fue lanzada con el código 2, por lo que se ejecuta el siguiente código:

```
if (resultCode == Activity.RESULT_OK && requestCode == 2) {

    ArrayList<Toma> listaTomas = data.getParcelableArrayListExtra("listaEditada");

    listaTratamientosNuevos.clear();

    for (Toma toma : listaTomas) {
        listaTratamientosNuevos.add(toma);
    }

    adapterListView = new AdapterListView(noTomas, infoTomas, modelo, this,
listaTratamientosNuevos, listView);
    listView.setAdapter(adapterListView);

    Commodities.ajustarVistaListView(listView);

    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id)
{

            Intent intent = new Intent(getApplicationContext(),
ConsultarTomaMedico.class);
            intent.putParcelableArrayListExtra("listaTomas", listaTratamientosNuevos);
            intent.putExtra("position", Integer.toString(position));
            startActivityForResult(intent, 2);

        }
    });
}
```

Donde el requestCode es 2. Se obtiene la lista con la toma editada, y se actualiza la vista del listview con esta lista. Además se vuelve a redefinir el onItemClickListener().

Pero estos cambios no se guardarán definitivamente en la base de datos si, al igual que al añadir o borrar una toma, no se presiona sobre el botón "GUARDAR" de la vista de crear tratamiento.

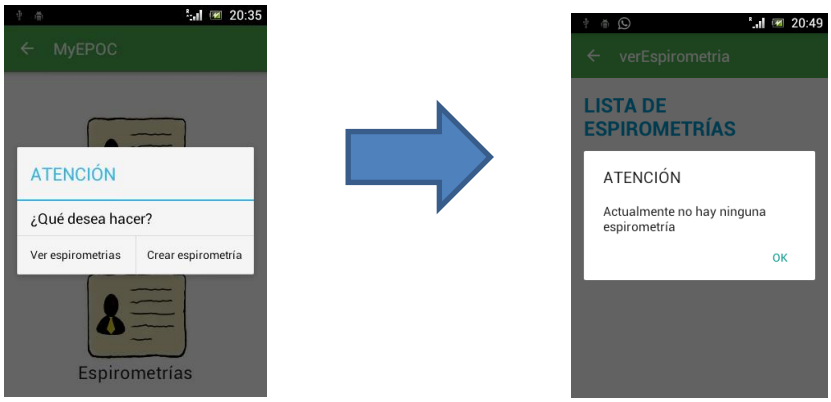
Si al editar, introducimos un principio activo a una hora en la que ya había una toma configurada con ese mismo principio activo, la aplicación avisará de que no es posible.



Esto es posible gracias a que en la Activity AddNewTake, se tiene la lista de todas las tomas, y cuando se va a añadir una nueva, o en este caso a editarla, se comprueba que no haya ninguna repetida.

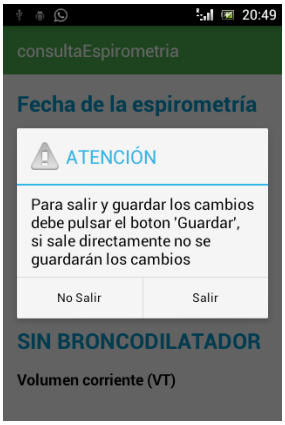
3.2 Crear, consultar y editar espirometrías desde la parte reservada al personal sanitario.

Desde la parte reservada al personal sanitario, se pueden crear, consultar o editar espirometrías. Para ello, al pulsar sobre el botón de Espirometrías, la aplicación pregunta si desea verlas, o añadir una nueva. En el caso que se deseen ver, pero aún no se haya creado ninguna, la aplicación avisará de esto.



En el caso que se presiones sobre “Crear espirometría”, se abrirá una nueva vista, donde el médico tiene que rellenar los datos. Los únicos campos obligatorios son los de la fecha. También se puede guardar una descripción, pero esto es opcional.

Si se intenta guardar la espirometría sin rellenar la fecha, se avisará con un mensaje de error. Una vez se hayan introducidos todos los datos, al presionar sobre el botón “Guardar”, se vuelve a la parte reservada al personal sanitario, y se avisa que la espirometría ha sido guardada correctamente. Si se sale sin presionar este botón, se alertará de que los cambios no se guardarán.



Una vez se hayan añadido alguna espirometría, se puede acceder a la sección de “Ver espirometrías”, donde aparecerá un listado en orden cronológico inverso.

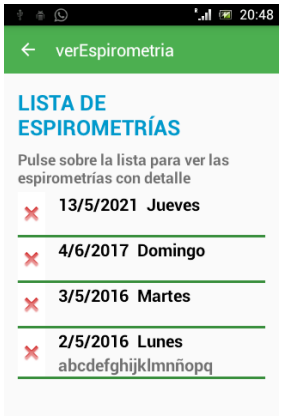
Este orden se consigue gracias a que la clase Espirometría.java implementa Comparable. En dicha clase, con el día, mes y año, se crea un objeto Calendar, y se implementa el método compareTo utilizando este objeto.

```
@Override
public int compareTo(Object another) {
    Espirometria e = (Espirometria) another;
    return getCalendar().compareTo(e.getCalendar());
}
```

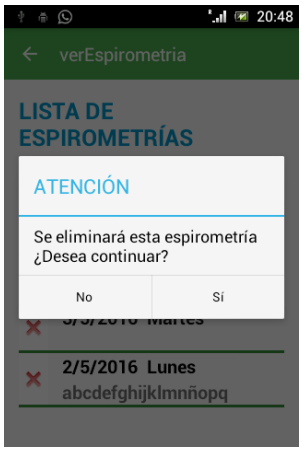
Gracias a esto, cuando se obtiene la lista de espirometrías, se puede ordenar según su fecha.

```
Collections.sort(listaEspirometrias, Collections.<Espirometria>reverseOrder());
```

En esta lista aparece la fecha, el día de la semana, que se obtiene directamente de la fecha con la que se crea el objeto Calendar, y la descripción en el caso que se haya introducido.



Si se desea borrar una espirometría, habrá que pulsar sobre la X roja, y se pedirá confirmación antes de borrar definitivamente la espirometría.



Para ver una espirometría, bastará con presionar sobre ella en la lista. Se abrirá una vista con los detalles de la espirometría, donde se da la opción de editar.

Al presionar sobre “EDITRAR”, se harán editables los campos cargados con los datos de dicha espirometría, y se sigue el mismo proceso que a la hora de crear una nueva. No se podrá guardar sin introducir una fecha, y si se sale sin guardar, no se almacenarán los cambios.

Para editar, consultar o añadir espirometrías, se ha usado el mismo layout. Para cada acción, se han cambiado las visibilidades y la editibilidad de los campos.

A la hora de crearla, se añade directamente a la base de datos al presionar sobre guardar,

```
modelo.insertEspirometria(espirometria);
```

pero para editar, cuando se presiona sobre guardar, se lanzan las actividades mediante startActivityForResult(), y cuando se edita la espirometría, a través del intent, se envía la nueva espirometría (editada) y la antigua (original).

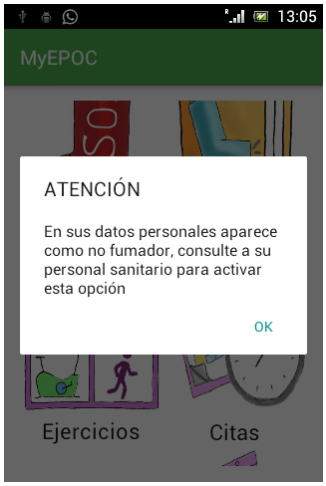
```
Intent i = new Intent();
i.putExtra("new", espirometria);
i.putExtra("old", e);
setResult(Activity.RESULT_OK, i);
```

En la clase VerEspirometrias.java, se ejecuta el método onActivityResult, donde se actualiza la base de datos con la nueva espirometría, y se notifica del cambio al adaptador de la lista para que refresque la vista.

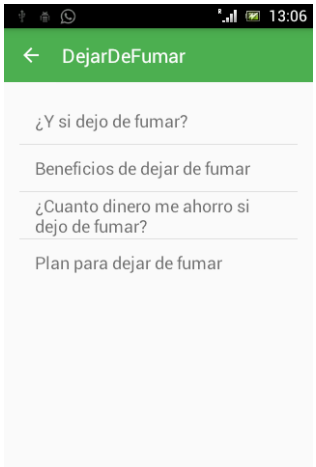
Caso de uso: El caso de uso de este requisito fue descrito por mis compañeros de salud en la primera parte del proyecto.

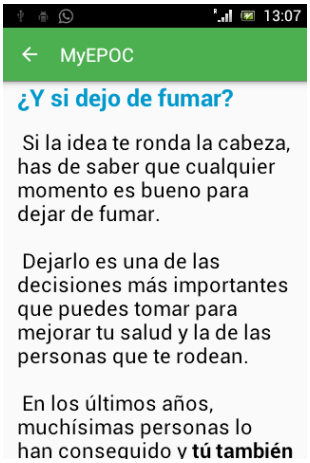
3.3 Guía para dejar de fumar.

Desde la pantalla principal, el paciente podrá acceder a una guía para ayudarle a dejar de fumar. En el caso de que en los datos personales, aparezca que el paciente no fuma, esta opción estará deshabilitada.



Dentro de esta sección hay diferente información al alcance del paciente para ayudarlo a dejar de fumar.

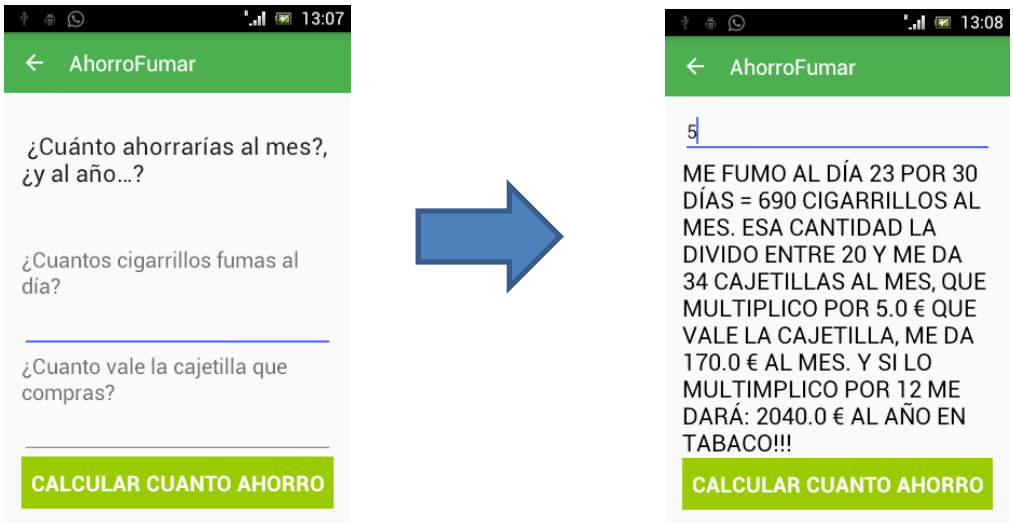




Las diferentes vistas de esta información, se muestra según el ítem de la lista que se presione. Según la posición de dicho ítem, se cargará un layout u otro.

```
private void startIntentWithLayout(int position) {
    int resourceId = R.layout.vista_y_si_dejo_de_fumar;
    if (position == 0) {
        resourceId = R.layout.vista_y_si_dejo_de_fumar;

    } else if (position == 1) {
        resourceId = R.layout.vista_ventajas_no_fumar;
    } else if (position == 2) {
        Intent intent = new Intent(getApplicationContext(), AhorroFumar.class);
        startActivity(intent);

    } else if (position == 3) {
        resourceId = R.layout.vista_plan_dejar_fumar;
    }

    if(position != 2){
        Bundle layoutBundle = new Bundle();
        layoutBundle.putInt("layout", resourceId);
        Intent intent = new Intent(getApplicationContext(), DesarrolloTutorial.class);
        intent.putExtras(layoutBundle);
        startActivity(intent);
    }

}
```

En el caso de que se quiera calcular cuánto dinero se puede ahorrar si se deja de fumar, se lanza una nueva actividad donde el paciente puede introducir el número de cigarros que fuma al día, y el precio de la cajetilla que compra, y la aplicación le mostrará cuanto puede ahorrarse.



4 Proceso iterativo

Para el desarrollo de la aplicación, se ha seguido un proceso iterativo, en el que no hemos reunido varias veces con la clienta, y a la que le hemos ido presentando la aplicación cada vez con más requisitos implementados. En estas reuniones, se decidían los cambios sobre lo ya implementado, y se recogían nuevos requisitos.

Se han realizado 5 iteraciones, de las cuales, las cuatro primeras han sido en las que he tomado el papel de programador, y en la última, de programador y analista.

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.

5 Conclusiones

El trabajo con una clienta real ha sido muy diferente a como trabaja en las prácticas de clase, ya que hemos tenido que adaptarnos a ella y ponernos en contacto con ella para que nos diera información y aclarase algunos requisitos, pero a pesar de esto, el formar parte de este proyecto ha sido una gran experiencia que seguro me ayudará para mi futuro profesional.

En cuando a la aplicación, ha superado mis expectativas, ya que cuando empezamos con el proyecto, no sabíamos casi nada sobre programación en Android, pero poco a poco hemos ido a prendiendo y gracias a este esfuerzo, se ha podido crear una aplicación muy completa con funcionalidades que cuando todo empezó, no sabía si íbamos a ser capaces de implementar.

Cuando comenzamos, nos pusimos el objetivo de entregar una aplicación competitiva al concurso, y creo que lo hemos conseguido. Tras esto nos propusimos seguir añadiendo funcionalidades y mejorando la aplicación, y creo que a día de hoy, es una de las mejores aplicaciones para la ayuda a enfermos de EPOC, es un lástima que no hayamos tenido tiempo de presentar esta última versión al concurso.

Creo que hemos hecho un buen trabajo, y que esta aplicación podrá ayudar mucho a los enfermos de EPOC. En un futuro, si tiene éxito, se podrá seguir completando y mejorando con nuevas funcionalidades, ya que la clienta sigue teniendo nuevas ideas para la aplicación como gráficas, autoevaluación… También tiene la posibilidad de adaptarlo a otro tipo de enfermedades ya que la aplicación está hecha de forma que se puede usar con cualquier medicamento, solo habría que cambiar los datos de la base de datos.

6 Bibliografía

[1]
http://conceptodefinicion.de/android/
[2]
http://sg.com.mx/revista/17/sqlite-la-base-datos-embebida#.V26lh_mLTIV
[3]
https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)
https://docs.oracle.com/javase/7/docs/api/
[4]
https://www.w3.org/XML/
[5]
https://www.zetetic.net/sqlcipher/about/
[6]
https://git-scm.com/
[7]
https://www.sourcetreeapp.com/
[8]
https://es.wikipedia.org/wiki/Bitbucket
https://bitbucket.org/
[9]
https://es.wikipedia.org/wiki/Android_Studio
http://android-studio.uptodown.com/windows
[10]
http://tecnologia.elpais.com/tecnologia/2015/12/14/actualidad/1450079229_273470.html
[11]
http://www.sgoliver.net/blog/bases-de-datos-en-android-i-primeros-pasos/

7 Anexos técnicos

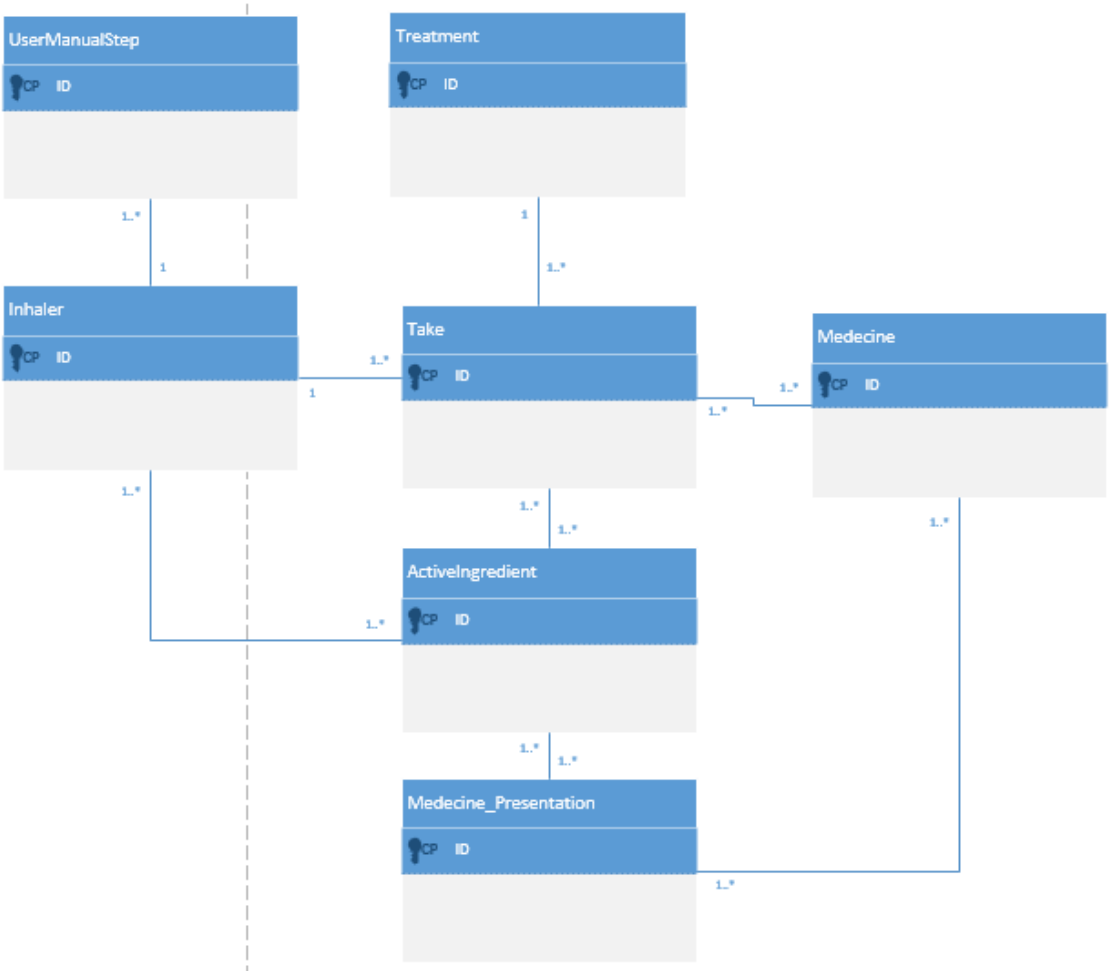
7.1 Casos de uso de la segunda parte del proyecto:

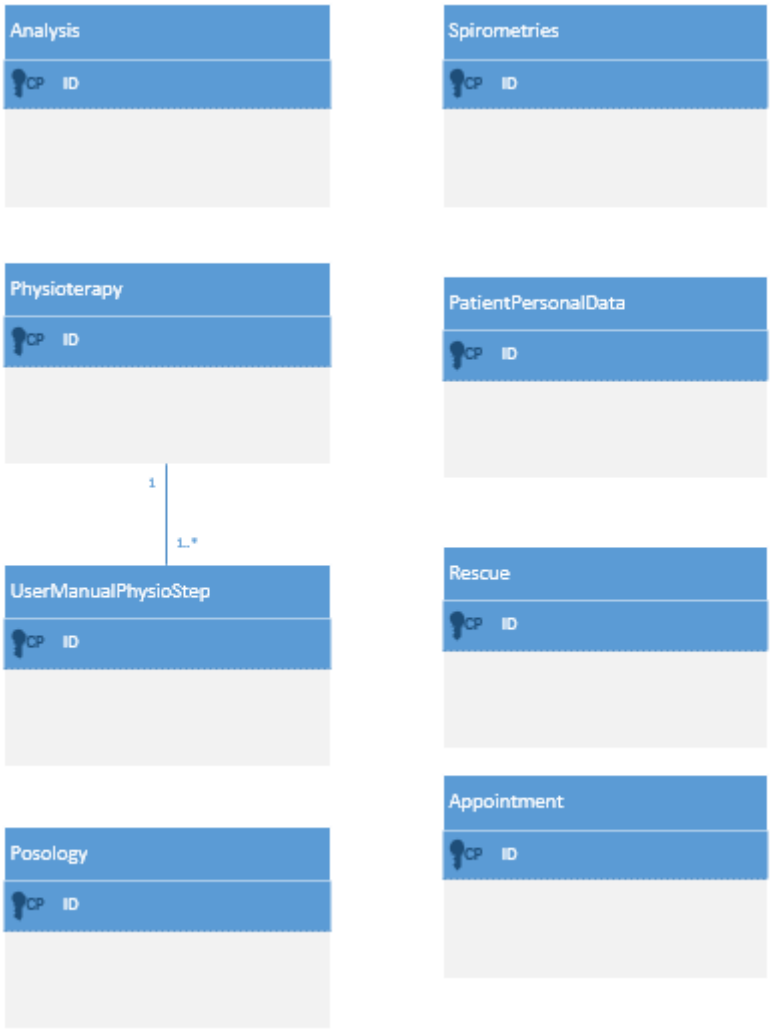
Título	Editar las tomas
Descripción	El personal sanitario podrá editar las tomas que se hayan configurado previamente.
Pre-condición	
Post-condición	
Prioridad	Alta
Autor(es)	Sergio Sánchez Gil
Control de cambios	
Escenario principal	
1. El usuario accede a la parte de creación de tomas en la sección reservada para el personal sanitario. 2. El sistema muestra una lista de las tomas que ya se hayan configurado previamente. 3. El usuario hace click sobre una toma de esta lista 4. El sistema muestra una vista con toda la información detallada de la toma seleccionada y un botón de editar 5. El usuario presiona sobre el botón editar 6. El sistema muestra la vista donde se crean las tomas pero con los datos de la toma que se va a editar cargados en los desplegables. 7. El usuario cambia los datos de la toma y presiona sobre el botón de finalizar toma 8. El sistema vuelve a la vista detallada donde se reflejan los cambios que se han hecho y muestra un mensaje informando sobre que la toma ha sido editada. 9. El usuario puede ver los cambios que ha hecho, volver a presionar sobre editar, o volver a la vista donde aparece la lista de tomas. 10. Al volver, para guardar los cambios, el usuario presiona sobre el botón guardar.	
Clases de análisis	
A. Clases de entidad	Toma
B. Clases de control	CrearTratamiento, ConsultarTomaMedico, AddNewTake

C. Clases de interfaz	activity_consultar_tratamiento_particular, activity_crear_tratamiento, activity_add_new_take

Título	Guía para dejar de fumar
Descripción	El usuario puede acceder a una guía para ayudarlo a dejar de fumar
Pre-condición	
Post-condición	
Prioridad	Media
Autor(es)	
Control de cambios	
Escenario principal	
11. El usuario accede a la guía desde la vista principal de la aplicación 12. El sistema muestra una lista con los diferentes tipos de información 13. El usuario selecciona algún elemento de la lista según lo que quiera consultar 14. El sistema muestra un texto con la información	
Clases de análisis	
A. Clases de entidad	
B. Clases de control	AhorroFumar, DejarDeFumar, DesarrolloGuiaNoFumar
C. Clases de interfaz	activity_ahorro_fumar, activity_dejar_de_fumar,

7.2 Diagrama de la base de datos:




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