ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

"App móvil para Android de información sobre FARMACIAS DE PAMPLONA"

Iker Pimoulier Ugarte

Alfredo Pina Calafi

Pamplona, 29-06-2015

ÍNDICE

1 – Introducción:

La tecnología ha cambiado el desarrollo de la vida cotidiana a nivel mundial, esto es algo que ha venido cogiendo más fuerza con el paso de los años hasta el punto de llegar a aparecer términos que mezclan un poco ambas, la tecnología y la vida. Un ejemplo de estos nuevos términos que han cogido fuerza en estos últimos años es el de *Smart City* o Ciudad Inteligente.

Y, ¿qué es una Ciudad Inteligente? Según se puede leer en *Wikipedia*, por ejemplo,

*La «**ciudad inteligente**» a veces también llamada «**ciudad eficiente**» o «**ciudad súper-eficiente**», se refiere a un tipo de desarrollo urbano basado en la sostenibilidad que es capaz de responder adecuadamente a las necesidades básicas de instituciones, empresas, y de los propios habitantes, tanto en el plano económico, como en los aspectos operativos, sociales y ambientales.*

Es decir, un desarrollo eficiente de la ciudad sin descuidar ninguna de sus áreas y pensado sobre todo en el posible beneficio para la persona, el ciudadano que en ella habita. Existen proyectos de todo tipo pero a mí me gustaría centrarme en la parte que ocupa la tecnología y más concretamente las TIC (Tecnologías de la Información y la Comunicación).

A día de hoy es casi impensable que una persona normal y corriente pueda vivir sin móvil, acceso a internet de alguna manera o incluso, aunque en menor medida, estar relacionado a alguna red social. Es por esto que cada vez se está mirando más por aprovechar todo esto y realizar proyectos que ayuden a los ciudadanos a estar en contacto con la ciudad y lo que en ella ocurre. Actualmente se está haciendo más y más frecuente el hecho de ver en puntos estratégicos de ciudades códigos QR que al leerlos te muestran información de ese lugar, edificio o simplemente qué lugares de interés puedes tener a tu alrededor visualizando un mapa. Sin ir más lejos, en Pamplona tenemos un buen ejemplo en las marquesinas de las villavesas, donde puedes consultar el tiempo de llegada del siguiente transporte con tan sólo leer un código.

Otro término que ha cobrado fuerza y que es muy importante para el desarrollo de proyectos, artículos o simplemente para ofrecer información al ciudadano curioso, es el de *Open Data*. Estos "Datos Abiertos" lo que persiguen es que cualquiera pueda tener acceso a ellos de manera libre, gratuita, sin restricciones de patentes o cualquier otro tipo de control, que puedan ser reutilizados según al usuario le convenga. En este campo, podemos encontrar universidades, gobiernos, revistas, que ofrecen sus datos de manera libre pensando que es algo a lo que cualquiera debería tener acceso.

2- Mi proyecto:

Teniendo en mente estas dos ideas principales de “Smart City” y “Open Data” y también la curiosidad y ganas de meterme con la programación Android, algo que no había tocado hasta empezar con este proyecto, me decidí a realizar una aplicación para Móviles.

Ya tenía el primer paso, sabía que quería hacer una App que además fuese de utilidad para la gente, algo que pudiese ayudar a los pamploneses. Tras días y días de desechar malas ideas, la propia vivencia me dio una posible solución. Tras tener que ir a Urgencias del Hospital un domingo, debía comprar una medicación y resulta que en ese momento no sabía que farmacias tenía a mi disposición, al final mirando el periódico resolví el problema. En un primer momento no caí en la idea y tuvo que ser una amiga la que me sugiriese el tema que finalmente sería el bueno para la aplicación, las farmacias de Pamplona.

De esta forma empezó a tomar cuerpo la idea de realizar una aplicación sencilla que informase de las farmacias de Pamplona y de aquellas que estuviesen de guardia. Tras realizar una primera búsqueda de aplicaciones sobre farmacias de Pamplona o Navarra existentes, encontramos sólo una llamada ***Farmanav,*** luego veríamos que si que existía otra, que estaba hecha en 2011 pero que al intentar probarla no nos funcionaba y se detenía al iniciarla. Visto esto, decidí proponerle desarrollar una aplicación de farmacias de Navarra a mi tutor, a quien al parecerle una buena idea me dio luz verde para que empezase. Pero lo primero era familiarizarse con el lenguaje *Android*. Para poder tener unos conocimientos básicos, me hice con el libro “*El gran libro de Android”* y me adentré en el mundo de la programación de *Apps* a lo largo de unos meses. Una vez tenía ya una pequeña base de este lenguaje y de *Eclipse* y el *ADT Bundle (Android Development Tools)* que es el entorno de programación preparado para este lenguaje donde iba a desarrollar la aplicación, era momento de empezar con el proyecto propio y el primer paso era probar otras aplicaciones existentes del mismo tipo para ver tipos de estructuras, posibles ideas a tener en cuenta y ver un poco que opciones tomábamos para nuestro caso.

3- Otras aplicaciones:

Consultando la "*Play Store*" de Android desde donde se pueden descargar aplicaciones de todo tipo, podemos encontrar varias aplicaciones sobre farmacias que pueden ayudarnos a tener más claros los conceptos que queremos para nuestro proyecto y también aquellos aspectos que creamos que nuestra aplicación no debe tener.

La primera de ellas es ***Farmacias Valladolid*****,** una aplicación sobre las farmacias de guardia de Valladolid. Una vez accedemos a ella aparece la opción de solicitar una fecha de consulta así como si la guardia es diurna o nocturna. Una vez seleccionado todo nos aparece en un listado las correspondientes farmacias con el nombre, dirección y teléfono. También nos da la posibilidad de llamar directamente mediante un botón o de ver la localización de la farmacia en un mapa. Este último botón lo que hace es abrir la localización bien en la app de google maps, bien en una aplicación de GPS que tengamos en nuestro móvil. Los datos que pasa al mapa son las coordenadas del lugar. (https://play.google.com/store/apps/details?id=com.gilsimon.farmacias)

Se trata de una interfaz sencilla y fácil de utilizar, aunque al realizar consulta de las farmacias de guardia nocturnas en fechas posteriores algunas con varias semanas de separación, el resultado es siempre el mismo, las mismas farmacias. No sé si esto puede ser lo correcto o si por el contrario existe algún tipo de error.

Otra de las aplicaciones que he probado recoge las farmacias de guardia de las Islas Canarias al completo, lo que supone una mayor aplicación y cantidad de datos. Su nombre, ***Farmacias de Guardia Canarias***. Al entrar en ella la pantalla principal es un tanto confusa, con 4 botones que no se sabe bien para qué es cada uno. Si le damos al botón de buscar farmacias, lo primero que tenemos que hacer es seleccionar la provincia y una vez ahí la isla y la zona. Una vez seleccionado todo, nos salen los resultados correspondientes también en forma de listado. Si entramos en una, se nos muestra en pantalla los diferentes datos de esa farmacia y además, se nos da la posibilidad de ver su localización en un mapa que se despliega dentro de la aplicación. También tenemos la opción del GPS, así como la de llamar directamente o compartir dicha farmacia por las diferentes opciones que hay hoy en día (WhatsApp, Twitter, mensaje, email…). (https://play.google.com/store/apps/details?id=tk.jairamrb.farmaciasdeguardia)

A mi parecer, la visualización del mapa es algo peor que en la app anterior y además esta aplicación es algo más difícil de manejar para un usuario no muy puesto en las tecnologías.

A continuación he probado una app sobre las farmacias de Barcelona, ***Farmaguia***. En la pantalla principal tenemos varias opciones a las que poder acceder, la farmacia abierta más cercana, las que se encuentran a mí alrededor o un buscador avanzado. Para poder utilizar las dos primeras es necesario tener activado el GPS para que el móvil pueda saber nuestra posición. En la búsqueda nos aparecen diferentes opciones de filtrado como la fecha y hora y un filtro según el tipo de farmacia que estemos buscando. Es interesante la opción que se da en la pantalla principal de ajustes, donde podemos cambiar el idioma, ver la información legal correspondiente así como un buzón de sugerencias. (https://play.google.com/store/apps/details?id=net.cofb.android.farmaguia)

Creo que es una buena aplicación en la que fijarse, aunque se requiere el GPS en todo momento y no hay forma de consultar las farmacias sin él.

También he investigado el funcionamiento de ***Farmacias de Guardia A Coruña***, se trata de una aplicación en la que la pantalla principal tiene un selector de municipio y también un selector de fecha de consulta. Además la selección del municipio se puede hacer también por voz o escribiéndolo. Al realizar la búsqueda nos aparecen listadas las farmacias de guardia y al seleccionar una se carga una pantalla emergente con diferentes opciones, ver en mapa, cómo llegar andando, cómo llegar en coche y la opción de si la farmacia está mal posicionada ponerse en contacto vía email. Si le damos a ver en mapa, se muestra la farmacia posicionada en un mapa y al pulsar sobre la cruz verde que aparece, salta otra ventana tipo *popup* con los datos de la farmacia. Las dos opciones de cómo llegar no he podido probarlas ya que al pulsar en cualquiera de las dos la aplicación me deja de funcionar. (https://play.google.com/store/apps/details?id=es.android.farmacias.coruna&hl=es)

A mi parecer la interfaz es sencilla pero casi demasiado con el fondo blanco y los diferentes elementos en gris. La opción de mostrar al pulsar sobre las farmacias encontradas una ventana emergente con las diferentes opciones, es algo diferente a lo visto hasta ahora pero tampoco me acaba de convencer, más aún cuando no me funcionan dos de las opciones.

Por último he probado una aplicación que nos muestra las farmacias de Zaragoza. La pantalla inicial nos muestra un mapa con todas las farmacias señalizadas en él mediante un icono al cual se puede acceder. Si accedemos a una de estas farmacias, nos aparece en una pantalla los datos así como la opción de cómo llegar o la de llamar directamente. Para poder utilizar las opciones de cómo llegar, nos pide nuestra localización por lo que se ha de activar el GPS. También en la parte superior de la pantalla principal podemos elegir entre la visión del mapa antes comentado ó un listado donde aparecen todas las farmacias de Zaragoza con diferente color en el signo para indicar aquellas que están de guardia o las que tienen el horario habitual. Además, si accedemos al menú, tenemos las opciones de elegir fecha y hora, que se

muestren también las farmacias que están cerradas, la información de la empresa que ha hecho la app, refrescar el mapa, así como una lista con teléfonos de interés. (https://play.google.com/store/apps/details?id=com.geoslab.farmaciasahorazgz)

Me parece original el hecho de que la pantalla principal sea directamente el mapa o el listado completo, hace que sea una aplicación sencilla ya que nada más entrar te van a aparecer las farmacias que tienes disponibles. En cambio echo en falta un buscador para encontrar una farmacia específica.

En el momento inicial de buscar aplicaciones sobre farmacias de distintas zonas de España así como las de Pamplona, no vimos que existía otra aplicación sobre las farmacias de guardia de navarra que sí que funcionaba en el *play store*. ***Farmacias de Guardia – Navarra*** que es como se llama esta app, es una aplicación desarrollada en 2011 por la empresa GeoActio y la cual recibió el premio Open Data Navarra en ese mismo año, con lo que deja ver que es una gran aplicación. Al descubrir esto, nos entran las dudas de si realmente podemos seguir adelante o no con nuestra aplicación ya que hay otra existente y al parecer muy buena. Al probarla vemos que la pantalla principal nos da la opción de ver las farmacias cercanas y también ver las farmacias de guardia. En ambas dos opciones lo primero que nos pide es nuestra posición por lo que se ha de activar el GPS, si le dices que no quieres activarlo te da la opción de meter una dirección manualmente o bien que elija un punto antiguo recordado. Una vez completado esto, nos aparecen las farmacias más cercanas en el primer caso y las de guardia en el segundo, con la opción de cómo llegar y verla con el *street view* cada una del listado. Esta segunda opción nos detiene la aplicación, mientras que la primera nos abre el navegador web con la explicación de cómo llegar al lugar dada por google maps. También tenemos la opción de verlas en un mapa y otra opción muy novedosa que es la realidad aumentada, la pena es que para ejecutarla necesitas instalar otro programa en tu dispositivo, *Wikitude*. (https://play.google.com/store/apps/details?id=farmacias.com)

Con todo esto vemos que es una aplicación muy encarada a la posición del usuario y más enfocada en la muestra en mapa y localización más realista. La pena es que si no tienes acceso a internet poco o nada puedes hacer en ella.

4 – Diseño e implementación de la App:

Con los datos obtenidos de probar e investigar las aplicaciones mostradas en el apartado anterior, decidimos que características debe tener nuestra aplicación. En primer lugar, queremos que pueda llegar al mayor número de usuarios posible por lo que tiene que ser sencilla y lógica en el manejo, es decir, deberemos desarrollar una interfaz intuitiva y fácil de usar para cualquier persona tenga la edad que tenga. La idea de mostrarlas en un mapa es algo necesario y que además todas las probadas incluyen al ser de gran utilidad.

Por otro lado, tampoco queremos que sea una aplicación esclava de internet y que una persona que no disponga de conexión en el móvil en todo momento ya no pueda acceder a ningún dato o no pueda navegar por ella. Este es uno de los motivos por el cual desechamos la idea que teníamos en un principio de colocar la base de datos con todas las farmacias en un servidor al cual se pudiese acceder desde la aplicación. Para solventar el posible problema de internet, decidimos que la base de datos iría integrada dentro de la aplicación.

Otro punto importante para nosotros es el buscador, nuestra aplicación deberá tener un buscador bastante completo en el cual el usuario pueda limitar su búsqueda bastante.

También queríamos aportarle algún toque distintivo que se saliese un poco de lo común por lo que decidimos incluir en el proyecto los códigos QR, de esta manera integrábamos un poco también la idea de *smart city* haciendo que el usuario pueda interactuar de alguna manera con la ciudad.

Una vez tenemos claro que es lo que queremos incluir y hacer, es hora de empezar a implementar la aplicación y empezaremos con la base de datos.

4.1 – Base de datos:

Para realizar la base de datos se ha utilizado SQLite ya que te permite crear y trabajar con una base de datos y luego integrarla en android de manera sencilla. Nuestra base de datos se compone de 2 tablas:

- actualizacion:
  Esta tabla tiene como campos el identificador que hace de clave primaria y el campo “fechaActualizacion” de tipo texto y con la condición que no puede ser nulo. Este campo se utiliza para ver la última fecha en la que se han actualizado las farmacias de guardia de manera si ya se han actualizado un día, la aplicación no entre a actualizarlas otra vez si vuelves a entrar en la app. Solamente se actualizarán si la fecha de uso es distinta a la que se encuentra guardada en la base de datos.

- farmacias:

  Esta es la tabla importante ya que es en ella donde se guardan todos los datos de las farmacias de Pamplona y alrededores. Los 10 campos que tiene son:

  · _id: Campo de identificación que será la clave primaria.
  · farmacéutico: Apellidos y nombre del farmacéutico.
  · dirección: Dirección de la farmacia.
  · zona: Zona en la que se encuentra la farmacia.
  · teléfono: Teléfono de contacto.
  · abre_24h: Campo SI/NO para indicar si es farmacia de guardia.
  · latitud: Coordenada de latitud del lugar donde está la farmacia.
  · longitud: Coordenada de longitud del lugar donde está la farmacia.
  · código: Código distintivo de cada farmacia.
  · horario: Rango horario de apertura de la farmacia.

Para rellenar esta tabla con cada una de las farmacias, se han utilizado dos vías de datos, la página oficial del colegio de farmacéuticos de navarra (www.cof-navarra.com), así como los datos de libre acceso proporcionados por el Gobierno de Navarra (www.gobiernoabierto.navarra.es/es/open-data/datos/farmacias-abiertas-al-publico).

Se han utilizado ambas fuentes de información para contrastar y completar los datos, ya que había farmacias en la página del colegio de farmacéuticos en las que no aparecía teléfono de contacto y a su vez, había farmacias que aparecían en ambos lados pero indicando un propietario diferente en cada fuente.

En un principio, pensamos en centrar nuestro trabajo sólo en las farmacias de Pamplona, pero más adelante vimos que incluir a pueblos colindantes tampoco suponía demasiado trabajo y de esa manera abríamos la aplicación a mucha más gente. De esta manera las distintas zonas incluidas en la aplicación son:

- Pamplona:
  · Azpilagaña
  · Casco Viejo
  · Chantrea
  · Ermitagaña
  · I Ensanche
  · II Ensanche
  · Iturrama
  · Milagrosa
  · Rochapea
  · San Jorge
  · San Juan

- Ansoain
- Barañain
- Berriozar
- Burlada
- Echavacoiz
- Huarte
- Mendillorri
- Mutilva Alta
- Mutilva Baja
- Nuevo Artica
- Ripagaina
- Sarriguren
- Villava
- Zizur Mayor
- Zizur Menor

Haciendo un total de 284 farmacias repartidas entre las distintas zonas.

4.2 – Pantallas de interfaz:

Una vez hecha la base de datos, lo siguiente ha sido desarrollar la aplicación empezando por la interfaz. Todas las pantallas están hechas en ficheros con extensión xml y se guardan dentro de la carpeta *res/layout*. Todas las pantallas se visualizarán con una barra de título con el icono y el nombre de la aplicación en la parte superior, esto lo configuramos en el fichero *AndroidManifest.xml* con el campo *application* definiendo el icono con el parámetro *android:icon="@drawable/ic_launcher"* y el nombre de la aplicación con *android:label="@string/app_name"*.

Empezamos con la pantalla principal ***nafarma.xml*** (figura 1) que muestra el logo de la aplicación y las distintas posibilidades a las que puede acceder el usuario, como las farmacias de guardia del día, un buscador de farmacias, la lista completa de farmacias y la opción de lanzar la lectura de código QR. Además de un botón de ayuda que explicará cada apartado, un botón para cerrar la aplicación y el logo del Gobierno de Navarra ya que al utilizar el Open Data que ofrece se debe nombrar.

Figura 1

Para realizarla utilizamos un *LinearLayout* de orientación vertical que configuramos con un fondo en degradado que creamos mediante un *gradient* y que guardamos en la carpeta *drawable*. Dentro del *layout* principal, creamos otros 5 *layouts* en este caso con orientación horizontal y que contendrán a la imagen del logo de la aplicación y a los diferentes botones. Por último utilizamos un *RelativeLayout* para posicionar el logo del Gobierno de Navarra, el botón de salir y el de ayuda en el mismo plano horizontal tomando como referencia para la alineación izquierda, centro o derecha al *layout* padre.

Todos los botones tienen un estilo creado y llamado *st_main_button* donde damos borde y sombras a los botones, que además dependiendo cuál sea, tomará un fondo diferente también creado en fichero *xml*. Así, los botones se diferenciarán en el color no en el efecto, siendo los fondos en degradado de rojo para el botón que accede a las farmacias, degradado de verde para los botones del buscador, la lista y el lector de QRs y gris para el botón de salir. Además agregamos el efecto que al ser pulsado el degradado se invierta verticalmente consiguiendo un efecto de pulsado mucho más realista.

Para el botón de ayuda hemos optado por un logo de un interrogante en azul que desentona un poco de los principales colores que aparecen en la aplicación como son el verde y el rojo. Se han elegido estos dos colores en relación al color símbolo de las farmacias y al rojo de Navarra, pudiéndose apreciar la sintonía tanto en el logo de

arranque (figura 3) como en el logo principal de la aplicación (figura 2). Ambos logos son de diseño propio y se han creado utilizando el programa *Photoshop* siguiendo con la misma idea del nombre de la aplicación que no es otra que la de englobar las farmacias y Navarra. En el caso del icono de la *app*, se ha utilizado la web http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html para obtener los distintos tamaños que puede tener dependiendo del tipo de pantalla que tenga el dispositivo en el que se ejecute. Estos iconos se alojarán en las carpetas *drawable-dpi*.

Figura 2

Figura 3

Este botón de ayuda nos mostrará la pantalla ***ayuda.xml*** de forma emergente sobre la principal donde se explican las diferentes funcionalidades de los botones (figura 4). Para conseguir que esta pantalla aparezca de ese modo *Popup* declaramos en el *AndroidManifest* de nuestra aplicación que el tema para la actividad que lanza la pantalla sea tipo diálogo. Todo el tema de actividades o *Activities* será explicado más adelante cuando indaguemos en el código de programación de la aplicación.

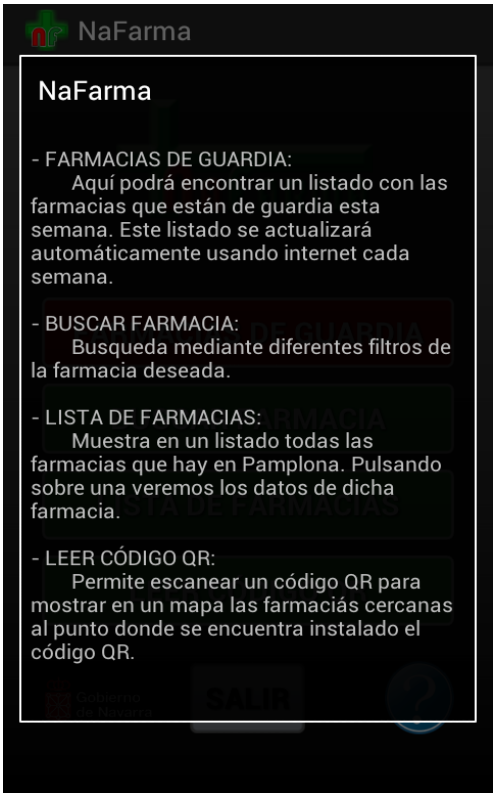

Figura 4

Siguiendo con las diferentes pantallas que nos podemos encontrar en la aplicación, vamos a describir ahora la del buscador ya que tanto el resultado de la búsqueda, como el listado de farmacias, como las farmacias de guardia, comparten estructura y se explicará conjuntamente más adelante.

La pantalla que contiene el buscador de farmacias, ***buscador.xml*** (figura 5), sigue la tónica general de la aplicación construyéndose sobre un *LinearLayout* de orientación vertical, de fondo el degradado oscuro y los colores verdes en botones y detalles. Dentro del *LinearLayout* vertical incluiremos otros de orientación horizontal para meter los diferentes elementos a modo de filas.

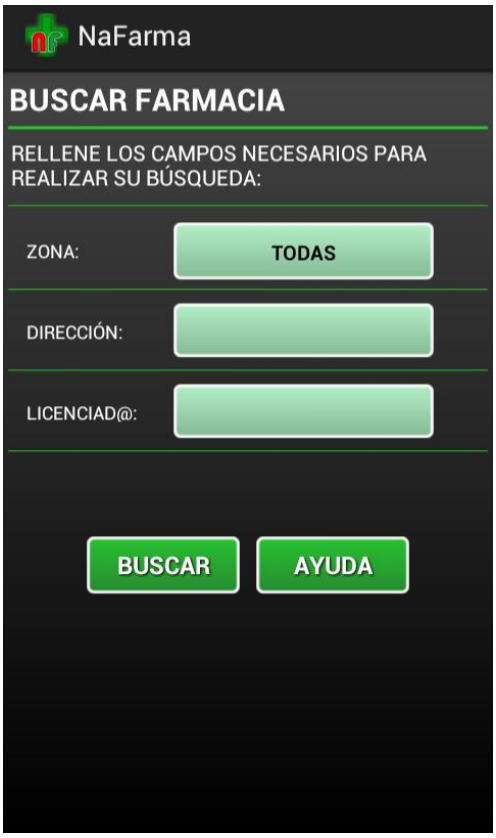

Figura 5

Esta pantalla tiene la parte superior con el título del apartado en el que nos encontramos y debajo una pequeña explicación para el usuario de que rellene los campos que crea conveniente para la búsqueda. Tras esto tendríamos otros 3 *LinearLayout* conteniendo los diferentes filtros que podemos utilizar para realizar nuestra búsqueda. El primero de ellos es un desplegable o *spinner* que nos muestra para que elijamos la zona donde buscar, por defecto viene elegida la opción de todas las zonas. El desplegable también se ha creado en el fichero ***mispinner.xml*** que a su vez utiliza el estilo *MySpinnerLook* que se encuentra dentro del fichero *styles.xml*, para conseguir que la lista aparezca con un tono verde no tan intenso como en los botones

y un tamaño de letra adecuado. También seguimos dando el realismo de pulsación al botón usando el intercambio de ángulo del degradado.

Los dos siguientes apartados de la búsqueda son dos campos de texto donde el usuario puede introducir una dirección o el nombre o apellidos de un licenciado. Por último tenemos los botones de buscar y ayuda, con las mismas características que todos los botones de la aplicación explicadas con anterioridad. El botón de ayuda nos mostrará, al igual que el de la pantalla principal, una ventana emergente que en este caso explica al usuario como ha de hacer la búsqueda (figura 6). Su fichero de *layout* es ***ayudabuscador.xml***.

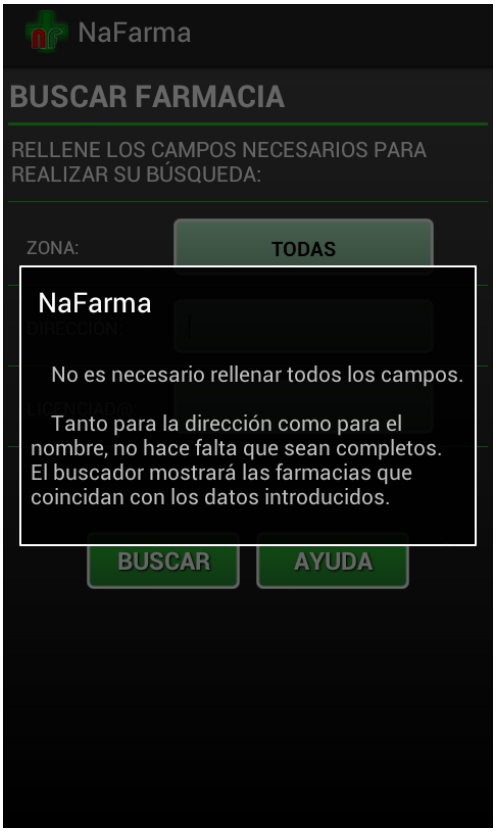

Figura 6

El botón buscar nos mostrará otra pantalla nueva, ***verfarmacias.xml*** donde se podrá ver el número de farmacias encontradas con esos parámetros de búsqueda así como una lista de dichas farmacias (figura 7). En la imagen se ha realizado una búsqueda de ejemplo para la zona de la Chantrea y podemos ver que aparece un listado con las 16 farmacias que se encuentran en dicha zona.

Figura 7

Para realizar tanto este listado como el que aparece en la pantalla de las farmacias de guardia y la pantalla que lista todas las farmacias, hemos utilizado el elemento *ListView* que toma el mismo fondo verde en degradado que teníamos en el desplegable de las zonas de búsqueda. Se puede ver que para cada farmacia encontrada aparece el nombre del farmacéutico como principal y la dirección como subtitulo. A la izquierda aparece una cruz verde como elemento decorativo y para ayudar en alguna manera a diferenciar unas de otras. Esta especificación para cada elemento de la lista, la definimos en el fichero ***listado.xml***. Al igual que los botones, al pulsar en alguno de los elementos de la lista, el degradado del fondo cambia el ángulo para crear el efecto visual de selección. En este *Layout* también existe un campo de error al principio que sólo será visible en caso de no encontrarse ninguna farmacia en la búsqueda.

Este mismo estilo de *ListView*, así como el color verde, lo utiliza también la pantalla en la que aparece el listado completo de las farmacias, ***listafarma.xml*** (figura 8). En cambio la pantalla que muestra las farmacias de guardia, ***farmaguardia.xml***, sí que utiliza el mismo estilo, pero en esta ocasión el degradado del fondo es de color rojo acorde al botón a través del cual se accede y en vez de mostrar en el subtítulo la dirección, muestra la zona, para ayudar al usuario a situarla más fácilmente (figura 9).

Figura 8

Figura 9

Decir que todos los listados muestran los datos ordenados alfabéticamente por los apellidos de los farmacéuticos.

En cualquiera de las pantallas anteriores con listados de farmacias, tenemos la posibilidad de consultar una farmacia en concreto si pulsamos sobre ella ya que el programa nos llevara a la pantalla donde podremos ver los detalles de cada farmacia, ***detallesfarmacia.xml*** (figura 10). En dicha pantalla podemos ver el nombre del farmacéutico en la parte superior a modo de título acompañado de la misma cruz verde del listado y separado por una línea, aparecen los campos dirección, zona y teléfono, en el caso de estar consultando la farmacia de forma normal, y también aparecerá el horario de apertura en el caso de que sea una farmacia de guardia. Por último en la zona inferior se encuentran dos botones con los indicativos de un teléfono y el de una posición en mapa.

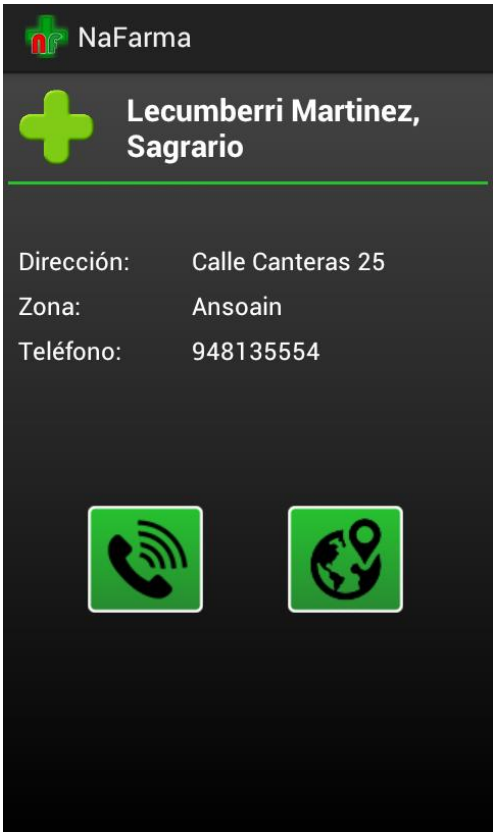

Figura 10

La pantalla se estructura a partir de un *LinearLayout* de base con orientación vertical y con el fondo degradado oscuro, algo común al resto de pantallas. La primera parte con la cruz que insertamos mediante un *ImageView* y el nombre del farmacéutico introducido sobre un *TextView*, se encuentra dentro de un *LinearLayout* con orientación horizontal. Tras esto metemos una línea divisoria de color verde, aparece también en otras pantallas, que la conseguimos mediante otro *LinearLayout* de orientación horizontal y al cual le damos a diferencia del resto de elementos del mismo tipo, una altura de tan sólo 2dp. Además le decimos que su fondo sea de color verde y le damos unos márgenes a ambos lados, con lo que conseguimos esa línea divisoria.

La parte de los campos está compuesta de otro *LinearLayout* vertical y dentro del cual se despliegan tantos *LinearLayout* horizontales como campos, es decir 4, dirección, zona, teléfono y horario; y otro más para los botones. Estos botones al igual que otros que aparecen con algún tipo de logo, los creamos a partir del elemento *Button* y tras darle el estilo propio que tienen todos los botones en nuestra aplicación con el color verde y el reborde blanco, le añadimos la imagen que se encuentra en la carpeta *drawable*. En este caso concreto, al ya utilizar la propiedad *background* para darle los formatos de color y bordes, le pasamos los logos mediante el parámetro *drawableLeft* que lo que hace es posicionar la imagen a la izquierda de nuestro botón. Con esto y jugando un poco con el *padding* para centrarlos, conseguimos los botones

con los que tras pulsar en ellos accederemos, bien a la cola de llamadas del teléfono donde ya estará el de la farmacia marcado y listo para simplemente darle a llamar (figura 11), o bien a una pantalla donde se nos mostrará un mapa, ***vermapa.xml*** y veremos su localización mediante una marca verde que si pulsamos nos mostrará la información de la farmacia (figura 12).

Figura 11

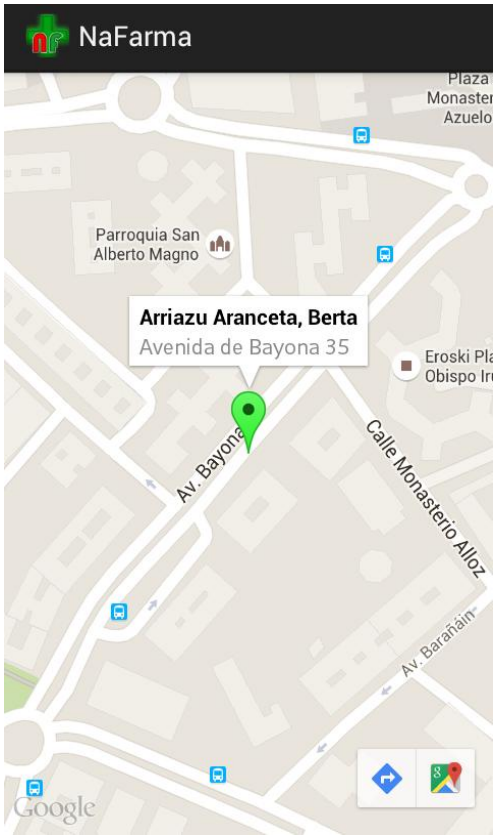

Figura 12

Para poder obtener en la pantalla el mapa, hemos utilizado el elemento *Fragment,* que se explica en las APIs de Google para android dentro de *Google Developers*, ya que es una manera sencilla de posicionar un mapa sobre una aplicación dentro de un simple xml y que nos permite luego acceder mediante el código y manejarlo a nuestra disposición. Para poder utilizarlo es necesario añadir una serie de comandos dentro de nuestro fichero *AndroidManifest* que explicaremos más adelante, así como tener acceso a la librería *google-play-services-lib*.

Hemos creado dos ficheros ***vermapa.xml*** y ***vermapaqr.xml*** para de esta forma trabajar sobre dos mapas diferentes a la hora de mostrar sólo una farmacia, o para cuando mostramos más farmacias además de una posición de usuario. Con este duplicado conseguimos que no aparezcan más farmacias posicionadas en el mapa de las que deberían.

4.3 – Clases java:

A continuación pasamos a describir cada una de las clases java que hemos creado para el funcionamiento de la aplicación. Empezaremos por la clase ***Farmacia.java*** que es un pilar fundamental en la aplicación.

Esta clase tiene como parámetros aquellos que tienen todas nuestras farmacias en la base de datos (farmacéutico, dirección, zona, teléfono, abre 24h, latitud, longitud, código, horario) y la usaremos cuando consultemos de la base de datos alguna farmacia ya que de esa manera tendremos acceso a todos los datos de dicha variable en tan solo una variable. Para ello primero describimos el constructor de clase al cual se le pasarán todos los parámetros antes citados. A continuación tenemos dos funciones por cada parámetro, una de *set* y otra de *get* que nos permitirán dar un valor a cada uno de los parámetros o bien consultarlo para realizar con él una comparación o cualquier otra operación. Las funciones *set* deben recibir un valor de entrada del mismo tipo que el parámetro que se va a sobrescribir y las funciones *get* lo que hacen es devolver el valor del parámetro de la clase *Farmacia*.

Tras esta clase, seguimos con unas de tipo más auxiliar en las que incluimos varias funciones útiles a las que luego llamaremos en nuestros programas.

La primera de ellas es la clase ***UtilesInternet.java*** que contiene las siguientes funciones:

- boolean verificaConexion():
  Esta función se encarga de comprobar si el teléfono móvil tiene acceso a alguna red de datos. Para ello creamos una variable de tipo *ConnectivityManager* en la que obtendremos las distintas redes de conexión.
  La información de esas redes la almacenaremos en un array de tipo *NetworkInfo* para luego mirar si su estado es el de conectado, en caso de serlo pondremos la variable booleana de salida a verdadera indicando de esta manera que si existe conexión de datos.

- String obtenerFecha():
  Con esta función obtenemos la fecha actual que nos servirá para controlar la entrada a la actualización de las farmacias de guardia comparándola con la fecha de la última actualización que guardamos en nuestra base de datos. La fecha la obtenemos con la función *getTime()* aplicada sobre una variable de la clase *GregorianCalendar()* que será de tipo *Calendar*. Esa fecha la almacenamos en una variable de tipo *Date* a la cual damos formato a nuestro gusto con un *SimpleDateFormat* y el resultado es un *String* que la función devolverá.

- int obtenerHora():
  En esta ocasión lo que obtenemos con la función es la hora actual, para lo que usamos la función *getCurrentTimezone()* de la clase *Time*. Una vez obtenida la hora en la variable de tipo *Time*, la actualizamos al momento actual con la función *setToNow()* y después elegimos el parámetro hora de nuestra variable para pasarle a la variable de salida de tipo entero.

Otra de las clases que sirve de apoyo a las otras es ***UtilesBD.java***, que contiene todo lo relacionado con la base de datos de nuestra aplicación. De esta forma en esta clase se encuentran todas las funciones que trabajan contra la base de datos. Debe ser una extensión de *SQLiteOpenHelper* para poder llamar a funciones que trabajan con *SQLite*.

Lo primero que hacemos es crear unas variables estáticas de tipo *String* que van a contener la dirección donde se va a encontrar nuestra base de datos alojada. A continuación tenemos el constructor de la clase y tras este, empiezan las funciones importantes:

- createDataBase():
  Esta función creará nuestra base de datos en el sistema. Primero consultará si ya existe la base de datos con la función *checkDataBase()*, en caso de no existir creará una base de datos vacía utilizando *getReadableDatabase()*. Una vez creada limpia, copiará nuestra base de datos desde la dirección de las variables antes descritas a la que se encuentra vacía por media de la función *copyDataBase()*. Esto último dentro de un *try…catch* de seguridad por si ocurriese algún problema durante la copia.

- boolean checkDataBase():
  Esta función comprueba si ya existe la base de datos intentando abrirla sólo en modo de lectura en la dirección establecida en las variables *DB_PATH* y *DB_NAME*. Si la variable que recibe teóricamente la base de datos no está vacía, significa que ya hay una base de datos creada y la cerrará. En caso contrario devolverá falso.

- copyDataBase():
  La función *createDataBase()* la llama cuando la función anterior da falso y entonces se encarga de copiar nuestra base de datos a una vacía. Para conseguirlo usamos un *InputStream* para recibir nuestra base de datos que se encuentra en la carpeta *assets* del programa y que abrimos con *getAssets().open(DB_NAME)*. De forma ponemos en un OutputStream la que va a ser nuestra base de datos vacía con dirección, la conjunta de las variables *DB_PATH* y *DB_NAME*. Una vez tenemos ambos archivos, leemos

del input y escribimos en el output mientras haya datos, terminamos cerrando ambos ficheros y haciendo un *flush()* al output por si quedasen elementos sin escribir.

- open():
  Función sencilla que abre en modo lectura y escritura nuestra base de datos y que deberemos ejecutar siempre antes de realizar cualquier acción contra la base de datos.

- close():
  Función opuesta a la anterior que se encarga de cerrar la base de datos en caso de que esté abierta.

Tras estas funciones generales de bases de datos, pasamos a las funciones ya propias para nuestra base de datos de farmacias:

- boolean updateFGuardia(String nombre, String horario):
  Esta función se encarga de actualizar en la tabla *farmacias* de nuestra base de datos el campo *abre_24h* a *SI* y el campo *horario* por el nuevo que se le pasa como entrada. Estos cambios los hará en aquella farmacia que coincida el nombre del farmacéutico con el que se le pasa como valor a la función.
  Para realizarlo escribimos nuestra consulta SQL *update* en una variable *String* y la ejecutamos sobre nuestra base de datos con la función *rawQuery*, devolviéndonos la respuesta en una variable de tipo *Cursor* con la que comprobaremos si realmente se ha ejecutado la consulta correctamente. Dicho cursor tendrá las filas afectadas por la ejecución de la consulta, por lo que sí el total de filas es distinto de 0, es que se ha ejecutado de forma satisfactoria.

- boolean updateFGuardiaNO():
  Del mismo modo que antes ejecutábamos una sentencia para actualizar a SI las farmacias de guardia, en esta función lo que hacemos es ponerles a todas las farmacias un NO en el campo *abre_24h*.

- Farmacia[] getFarmaciasBusqueda(String zona, String lcdo, String dir):
  Se trata de la función que controla la búsqueda de farmacias que se realiza en la pantalla del buscador y por lo tanto una función de gran importancia.
  Recibe tres parámetros de entrada que son los tres campos que se pueden rellenar en el buscador, la zona, el farmacéutico y la dirección. Como variable de salida, esta función devuelve un *String* de la clase *Farmacia* que hemos

creado antes, donde se guardarán aquellas farmacias que se hayan encontrado a partir de los parámetros que ha introducido el usuario.
La búsqueda la dividimos en dos partes para llegar a comprobar todos los casos, si el parámetro de entrada *zona* tiene como valor *TODAS*, o si se ha definido una zona específica. Dentro de cada una de estas dos partes de la sentencia condicional, tendremos 4 casos también condicionales con los que cubriremos todas las posibilidades. La diferencia será que a la hora de la sentencia que hace la consulta a la base de datos, en el primer caso no se restringirá la zona, mientras que en el segundo sí que se hará una restricción a la hora de hacer el *select* a la base de datos.
Los 4 casos que tiene cada parte son:

· Si no se ha introducido ni nombre del farmacéutico ni tampoco dirección. En este caso no habrá más restricciones salvo la de la zona en la segunda parte de la condicional principal.

· Si no se ha introducido ninguna dirección. Ahora se añadirá la condición a la sentencia de consulta de que el campo *farmacéutico* de la tabla *farmacias* de nuestra base de datos tenga al menos en algún momento la cadena que se haya introducido. Para ello utilizamos la sentencia *farmaceutico like '%"+lcdo+"%'* haciéndole saber que la cadena introducida en el buscador puede encontrarse con caracteres delante, detrás o ambos lados.

· Si no se ha introducido ningún nombre. Igual que el caso anterior pero en esta ocasión cambiamos el campo *farmaceutico* por el de *direccion* a la hora de comparar la cadena introducida.

· Si se han introducido tanto el nombre como una dirección. En esta ocasión debemos comparar en la sentencia ambos campos, el de *farmaceutico* y el de *direccion*. En el caso de tener también una zona, la búsqueda queda limitada por los 3 campos.

A la hora de hacer la consulta, le decimos que nos coja todos los datos que hay de cada farmacia y utilizaremos también un cursor con todas las filas coincidentes con la *query* para rellenar el *String* de tipo farmacia que hemos dicho que nos devuelve la función. Para introducir todas, ejecutamos un bucle *while* con la condición de que el cursor tenga aún una fila a la que acceder con la función *moveToNext()*. Para cada iteración del bucle guardamos en la posición del *array* que le corresponde una variable del tipo *Farmacia* nueva cuyos parámetros rellenamos con las funciones *set* correspondientes pasándoles como valores los que obtenemos de cada posición de la fila del cursor con la función *.getString(1) o getFloat(6),* por ejemplo para el nombre y la latitud respectivamente.

- int getNFarmaciasBusqueda(String zona, String lcdo, String dir):
  Esta función repite la búsqueda que realiza la función anterior con la diferencia que esta devuelve el número de farmacias que coinciden con la búsqueda realizada. Para ello cambiamos el parámetro de salida a un entero, cuando realizamos la consulta hacemos un *select count(*)…* para que el resultado de la búsqueda no sean todos los parámetros si no que sea la cuenta de las líneas que se han obtenido. Por lo tanto la variable que devuelve la función tomará el valor que obtiene el cursor en el lugar 0, que es la primera respuesta que obtiene de la consulta y por lo tanto el número de farmacias.

- String [] getFarmaceuticosZona(String zona):
  Función que consulta los nombres de los farmacéuticos en la base de datos dándole como parámetro restrictivo una zona que es el parámetro pasado como entrada. Seguimos utilizando una variable de tipo cursor para introducir el resultado de la búsqueda dentro del *array* de tipo *String* que devolverá la función con los nombres de los farmacéuticos que haya en esa zona elegida. Destacar que en la consulta le imponemos con el comando *order by farmaceutico,* que la respuesta se dé con los nombres ordenados alfabéticamente.

- String[] getZonas():
  En esta sencilla función lo que hacemos es devolver en un *array* de tipo *String* todas las zonas existentes en la base de datos sin que se repitan y ordenadas alfabéticamente. Para evitar que se nos repitan zonas introducimos delante del campo a buscar el comando *distinct*, de la siguiente manera, *select distinct zona…*. Como esta función la utilizamos para rellenar el desplegable de zonas que encontramos en la pantalla de búsqueda, para que en primer lugar aparezca por defecto *TODAS*, lo que hacemos es a la hora de crear el *array* de las zonas, darle un tamaño igual al número de filas obtenidas en el cursor más una. Y antes de empezar a rellenar el *array* con las distintas zonas, introducimos en la posición 0 el valor *"TODAS"*.

- Farmacia[] getTodasFarmacias():
  Esta función realiza una consulta sin restricciones pidiendo todos los datos y diciéndole que el resultado salga ordenado por el nombre del farmacéuticos con la orden antes vista *order by farmaceutico*. Al igual que en las anteriores funciones, meteremos todos los datos en el *array* de salida utilizando un cursor. También usaremos las funciones *set* y *get* de la clase *Farmacia* para rellenar todos los parámetros de cada farmacia que introduzcamos en la variable de salida.

- Farmacia[] getFarmaciasGuardia():
  Se trata de la misma función anterior salvo que en esta ocasión al hacer la consulta le restringimos a que el campo *abre_24h* tenga el valor *SI*. Con esto obtendremos un *array* con todas las farmacias que estén de guardia ordenadas alfabéticamente.

- Farmacia[] getFarmacias24():
  Función idéntica en cuanto a variable de salida que las anteriores pero en este caso le estamos diciendo en la consulta que farmacias queremos por el valor del campo_*id* de la base de datos. Las farmacias que le pedimos son las 3 que abren las 24 horas del día y que tienen como código identificador el 123 (farmacia de la calle Yanguas y Miranda), el 230 (farmacia de Nuevo Artica) y el 273 (farmacia de Ripagaina).

- boolean updateFechaActualización(String fechaNueva):
  Con esta función lo que hacemos es actualizar el valor del campo *fechaActualizacion* de la tabla *actualizacion* por la nueva fecha que pasamos como entrada a la función. Para controlar que se haya realizado correctamente, miramos el número de filas afectadas en el cursor y si es distinto de 0, ponemos la variable de salida a verdadera.

- String leeFechaActualizacion():
  Función simple que consulta la fecha de la última actualización de las farmacias de guardia que se encuentra en nuestra base de datos. Para ello hacemos la consulta de todos los datos a la tabla *actualizacion* y luego metemos en la variable de tipo *String* que devuelve la función el valor de la posición 1 de la fila 1 del cursor.

Con esto ya tendríamos explicadas las clases de apoyo de nuestra aplicación y podemos pasar a explicar las clases que manejan el funcionamiento de la aplicación.

Empezaremos con la clase de la pantalla principal, ***NaFarma.java***, que es la más importante y la que gestiona gran parte del funcionamiento de la aplicación. Para que sea la pantalla principal de inicio, en el fichero *AndroidManifest.xml* declaramos dentro de la *activity* correspondiente a la dicha pantalla, que sea la principal. Esto lo hacemos añadiendo dentro de un *<intent-filter>* las siguientes instrucciones:

<action android:name="android.intent.action.MAIN" />

<category android:name=*"android.intent.category.LAUNCHER"* />

Esta clase extiende de la clase *Activity* y al igual que todas las demás clases, una vez son llamadas, entrarán en el *OnCreate()* que es la función que ejecuta las diferentes acciones de cada clase. En el caso de la clase *NaFarma.java*, en el *OnCreate()* se encuentran las operaciones necesarias para la actualización de las farmacias de guardia. Para que se cargue la pantalla *nafarma.xml* al entrar en esta función, utilizamos la instrucción *setContentView(R.layout.nafarma)*.

Tras declarar las variables correspondientes a los diferentes elementos que existen en el *layout* buscándolos en el fichero nafarma.xml por su nombre mediante la función *findViewById()*, lo primero que hacemos es actualizar las farmacias que están de guardia en el día. Para esto es necesaria una conexión de datos y es por esto que comprobaremos el estado de la conexión llamando a la función *verificaConexion()* de la clase *UtilesInternet*. En un bucle condicional comprobaremos si existe conexión y si además la variable *actualizado* tiene un valor 1 o 0. Esta variable estará a 1 en el caso de que ya se haya realizado la actualización de las farmacias de guardia en ese día, en caso contrario estará a 0.

Si *verificaConexion()* nos devuelve falso y además *actualizado* tiene un valor distinto de 1, no se podrá realizar la actualización y lo que hacemos es mostrar un mensaje durante unos segundos y acceder a la base de datos y poner primero el campo *abre_24h* de todas las farmacias a *NO* con la función *updateFGuardiaNO()* para después actualizar tan sólo las 3 farmacias que sabemos abren 24 horas utilizando la función *updateFGuardia()*.

En caso contrario, obtendremos la fecha de la última actualización de la base de datos con *leeFechaActualizacion()* y también la fecha actual con *obtenerFecha()* de las clases *UtilesBD* y *UtilesInternet* respectivamente. A continuación comparamos ambas fechas y si no son iguales llamamos a una tarea asíncrona en forma de *ProgressDialog* que se encargará de actualizar las farmacias de guardia. Definimos que sea un *ProgressDialog* con estilo *spinner* que muestre el mensaje *Actualizando farmacias de guardia…* y además le damos la opción de que se pueda cancelar. De esta manera logramos que en caso de mala conexión o fallo, la aplicación no se quede bloqueada como si pasaría en caso de ser una tarea principal.

Por último declaramos la variable *tarea* de la clase *FarmaciasGuardiaJSON.java* llamando al constructor de dicha clase y después la ejecutamos con la función *execute()*.

Hacemos un inciso en la clase *NaFarma.java* para ir a explicar la clase a la que acabamos de llamar con la tarea.

La clase ***FarmaciasGuardiaJSON.java*** es la encargada de ejecutar la tarea asíncrona que actualiza las farmacias de guardia, por lo que esta clase extenderá de la clase *AsyncTask*. Este tipo de clases que realizan una tarea de forma asíncrona mediante un cuadro de progreso, están estructuradas en 5 funciones principales:

- onPreExecute():
  Realizará las operaciones pertinentes antes de empezar la tarea y mostrar el diálogo de progreso.
- doInBackground(void …parametros):
  Todo lo que incluimos en esta función se ejecuta en segundo plano y llamará a la función *onProgressUpdate()* con la instrucción *publishProgress(0).*
- onProgressUpdate(Integer value):
  Se encarga de actualizar el *ProgressDialog* que mostramos por pantalla con el valor que le pasamos, en nuestro caso 0 para que el *spinner* gire todo el rato.
- onPostExecute(boolean result):
  Recibe como entrada el resultado de la función *doInBackground()* y en caso de ser positivo, es decir que se ha terminado de ejecutar, se realizarán las tareas correspondientes después de la tarea.
- onCancelled():
  En caso de que se cancele la operación, se entrará en esta función.

Ahora entraremos en profundidad en las operaciones que se realizan en cada función en nuestro caso.

En la función *onPreExecute()* nos encargamos de configurar que se pueda cancelar el diálogo, así como abrir la base de datos para poder trabajar sobre ella, inicializar el diálogo de progreso y mostrarlo.

El grueso de la clase se encuentra en la función *doInBackground()* que es una de las partes fuertes de la aplicación ya que es la encargada de actualizar la base de datos con las farmacias de guardia de cada día. Para conseguirlo tomamos los datos del fichero *open data* que nos ofrece el Gobierno de Navarra en la web http://www.gobiernoabierto.navarra.es/es/open-data/datos/farmacias-guardia

Aquí nos hemos encontrado con el problema de que los ficheros con diferentes formatos que hay en la página, al clicarlos no se abren en modo web, dificultando el acceso a los datos, sino que se descarga directamente. Esto nos supone un gran problema ya que si en el programa ponemos que acceda al fichero con tratamiento web se nos descargará en el móvil, con lo que tendríamos que tratar los ficheros descargados para que no se almacenasen demasiados y al final crear un problema de memoria en el móvil del usuario. Tras experimentar con diferentes vías de acceso a los

datos, finalmente hemos conseguido acceder al fichero de forma satisfactoria y sin necesidad de que se descargue. A continuación explicamos los diferentes pasos e instrucciones empleados.

Empezamos actualizando la base de datos poniendo que no hay ninguna farmacia de guardia, para a continuación crear una conexión con la dirección http://www.navarra.es/appsext/DescargarFichero/default.aspx?codigoAcceso=OpenData&fichero=GuardiasFarmacias/Guardias.json

mediante una variable de la clase *HttpGet* que ejecutamos con a través de un *HttpClient* previamente creado. La respuesta que se obtiene de esta ejecución, la guardamos en una variable *HttpResponse* que usaremos para pasarle la entidad de la respuesta, si la hubiese, a otra variable *HttpEntity*. En esta variable ya tendríamos el mensaje obtenido de la conexión con la dirección de arriba y para acceder a su contenido, creamos una variable de la clase *InputStream* y mediante la función *getContent()* de la variable *entity*, pasamos el contenido. A continuación, leeremos toda esa variable *InputStream* con otra de la clase *BufferedReader* y almacenaremos todo el contenido línea a línea en un *StringBuilder*, utilizando para ello un bucle cuya condición sea que la línea que lea el *reader* no esté vacía. En cada iteración de este bucle llamamos a la función del dialogo de progreso *publishProgress(0)* para que siga apareciendo el círculo girando. Una vez acabe el bucle, cerramos el *InputStream* y pasamos a una variable *String* el contenido del constructor. Con esto ya tendríamos todo el fichero con las farmacias de guardia de Navarra de varios días en una sola variable de tipo *String*, ahora ya sólo queda acceder a los datos que queremos.

Para conseguir los datos necesarios y correctos para nuestra aplicación, primero obtenemos la fecha actual con la función *obtenerFecha()* de la clase *UtilesInternet* ya que luego compararemos con los valores de fecha que aparecen en el fichero. El fichero que elegimos para obtener los datos era el que tenía extensión *.json* ya que ahora podemos crear una variable *JSONArray* pasándole el *String* con todos los datos y esto nos permite tratar cada farmacia que aparece con los datos como un objeto. Entonces creaos un bucle que recorra todos los objetos del *array* y en cada posición creamos una variable de tipo *JSONObject* que nos permitirá acceder directamente a los datos poniendo como referencia el nombre del campo identificativo. Limitaremos la obtención de las farmacias a aquellas cuyo campo *FECHA* coincida con la actual que hemos obtenido antes, de esta forma evitamos leer todos los datos de farmacias que están de guardia otros días.

Si coinciden las fechas, guardamos el código de la farmacia, la hora desde la que abre y la hora hasta la que permanece abierta situados estos datos en los campos *Cod_FARMACIA*, *DESDE* y *HASTA* del fichero *json (Ver Anexo 5)*. Para coger por ejemplo el código de una farmacia usamos la función *getString("Cod_FARMACIA")* de la variable de tipo *JSONObject.* Una vez tenemos los datos de esa farmacia de guardia,

comparamos en un bucle el código obtenido con el de todas las farmacias de nuestra base de datos y si coincide actualizamos esa farmacia como farmacia de guardia con la función de nuestra clase *UtilesBD*.

Así ya tendríamos todas las farmacias de guardia del día de las que tenemos metidas en la base de datos.

Una vez hecha la actualización, el programa salta a la función *onPostExecute()* que recibe como parámetro de entrada el resultado de la función anterior. En caso de que ese resultado sea verdadero, es decir se ha ejecutado correctamente el *doInBackground()*, lo que hacemos es actualizar en la base de datos la fecha de actualización con la fecha de ese instante. También pondremos la variable actualizado a 1 para que la clase *NaFarma.java* no vuelva a entrar a actualizar si no es un día distinto. Para acabar cerramos el diálogo de progreso con el comando *dismiss* de la variable de tipo *ProgressDialog* asociada a dicho diálogo.

Por último tenemos la función *onCancelled()* a la que entrará el programa en caso de que el usuario decida cancelar el proceso de actualización o surja algún error. Lo que hacemos en esta función es actualizar en la base de datos que al menos las 3 farmacias de guardia que abren las 24 horas todos los días estén como farmacias de guardia.

Una vez que el programa ha terminado en esta clase, retornará a la clase principal donde se quedará a la espera de que se pulse alguno de los botones. Cada variable asociada a cada elemento del tipo *Button* del *Layout*, llama a la función *setOnClickListener()* pasándole como valor un nuevo elemento de escucha *OnClickListener()* y la función *OnClick()* que es la encargada de realizar las operaciones que correspondan cuando se pulsa dicho botón. Dentro de esta función, se hace los mismo para todos los botones excepto para el de lectura de *QRs* que tiene alguna instrucción más que las otras y que la veremos más adelante.

En la función *OnClick()* de cada botón lo que hacemos es crear un nuevo *Intent* al que pasaremos la clase que queremos ejecutar y arrancaremos una nueva *Activity* con el comando *startActivity(intent)* que nos llevará a que se ejecute la clase elegida en una nueva actividad. Para poder hacer esto hay que declarar cada actividad en el fichero *AndroidManifest.xml* declarando la actividad y cómo queremos que se ejecute la pantalla que en ella se carga.

A continuación veremos una a una las clases que carga cada botón y dejaremos para el final la última parte de la clase principal, que es la correspondiente a la lectura de los códigos QR.

Por seguir el orden en que nos aparecen en pantalla, empezamos con el botón de las farmacias de guardia que lanza la *Activity* encargada de cargar ***FarmaGuardia.java*.**

Al entrar en la función *onCreate()* lo primero que hacemos, al igual que en todas las clases que controlan *layouts*, es cargar la pantalla que se va a visualizar que en este caso es *farmaguardia.xml*. A continuación consultaremos en la base de datos las farmacias que están de guardia y las guardaremos en un *array* del tipo *Farmacia*. Si la longitud de dicha cadena es tan sólo de 3 farmacias, nos indica que no se han podido actualizar las farmacias y tan sólo están las que abren 24 horas que ponemos siempre como de guardia en la base de datos, en este caso mostraremos un mensaje al usuario mediante un *Toast* informándole.

Tras esto creamos una variable de la clase ***ElementoListaAdapterGuardia.java*** pasándole la cadena de farmacias que hemos obtenido antes, para después configurarlo como adaptador para la *ListView* que tenemos en el *layout*. Esta clase lo que hace es controlar los elementos de la lista y en la función *getView()* lo que hacemos es dar valor a los campos de cada elemento llamando al constructor de la clase ***ElementoListaGuardia.java***. En esta clase que extiende de la clase *LinearLayout* lo que hace el constructor es llamar a otra función de la clase llamada *inicializar()* que lo que hace es instanciar el *layout listado.xml* como vista de salida para cada elemento mediante un *LayoutInflater*. Además le damos valor tanto al farmacéutico como a la zona donde se encuentra la farmacia que le hemos pasado a esta función. De esta manera cada elemento de la lista tendrá la apariencia definida en *listado.xml.*

Una vez de vuelta a la clase en la que estábamos, *FarmaGuardia.java*, ya tendríamos la lista cargada con nuestras farmacias de guardia y lo que hacemos ahora es declarar un nuevo *OnItemClickListener()* con su función *onItemClick()* que se encarga de lanzar una nueva *Activity* que carga la clase *DetallesFarmacia.java* a la que pasamos mediante una variable del tipo *Bundle* todos los datos que aprovecharemos en ella para mostrar la información de la farmacia que se ha elegido. Para agregar datos a esta variable, se utiliza el comando *putString* pasándole en primer término el identificador para luego poder acceder a él y en segundo término la variable o aquello que le queremos pasar y que será en este caso del tipo *String*. Por ejemplo si le queremos pasar el nombre del farmacéutico al *Bundle*, la instrucción sería la siguiente:

b.putString("FARMACEUTICO",farmaciasGuardia[position].getFarmaceutico());

Una vez metidos todos los datos en la variable, se la pasamos al *Intent* creado para lanzar la nueva actividad con la función *putExtras()*.

Explicaremos ahora la clase *DetallesFarmacia.java* a la cual se accederá cada vez que haya un listado en pantalla y se pulse sobre una de las farmacias como nos ocurrirá tras realizar una búsqueda de farmacias o al pulsar sobre el botón que lista todas las farmacias. Explicaremos ambas más adelante cuando llegue su turno.

En la clase ***DetallesFarmacia.java*** una vez entrados en la función *onCreate()* lo primero que hacemos es adjuntar los campos del *layout* a variables para luego poder acceder a ellos y modificar sus valores. Una vez tenemos los campos administrados, sacamos todos los datos del *Bundle* que habíamos pasado con la función *getString()* pasándole el identificador que antes habíamos definido para cada datos, y le pasamos esos valores a las variables que controlan los campos de manera que aparezcan en pantalla.

Para los dos botones que hay en la pantalla creamos un *OnClickListener()* para saber cuándo se pulsan y poder hacer para cada uno la acción correspondiente. De este modo, en la función *onClick* del botón del teléfono, lo que hacemos es lanzar una nueva actividad que carga la pantalla de llamada del móvil dejando ya marcado el teléfono de la farmacia pero sin realizar la llamada directamente como medida de seguridad por si se pulsase el botón de manera involuntaria. Esto lo conseguimos definiendo en el *Intent* que su acción sea *Intent.ACTION_DIAL*.

En caso de pulsar el botón del mapa, lo que hacemos es lanzar una activity que llama a la clase *VerMapa.java* pasándole además por medio de un *Bundle* el nombre del farmacéutico y la dirección de la farmacia, así como los puntos longitud y latitud donde se encuentra.

La clase ***VerMapa.java*** carga la pantalla *vermapa.xml* y tras sacar los datos del *Bundle* y crear una variable del tipo *LatLng* que será un punto con ambas coordenadas, asociamos el *MapFragment* que se encuentra en el *layout* a una variable para después configurarlo y manejarlo. Definimos que sea un mapa normal con *setMapType(1)* y después le añadimos un marcador en el punto antes definido y decidiendo que el símbolo sea el que aparece por defecto pero en este caso en color verde. También le definimos que al ser pulsado aparezca el nombre y la dirección de la farmacia, y también hacemos que al cargarse el mapa se cree una animación que acerque el zoom al punto hasta dejarlo en 17.

Visto ya el recorrido posible del programa tras pulsar sobre el botón de farmacias de guardia, vamos a entrar ahora con el del buscador cuya clase es ***Buscador.java***.

Esta clase al igual que las anteriores carga al entrar en su función *onCreate()* la pantalla que se va a visualizar y que en este caso es *buscador.xml*. Tras asociar variables a los elementos del *layout*, consultamos las zonas haciendo la consulta a la base de datos y estas zonas se las asignaremos al *spinner* creado para que el usuario pueda elegir. Para definir el formato en que van a aparecer las diferentes zonas en el desplegable, le pasamos al *ArrayAdapter* encargado de manejar el *spinner*, el fichero *mispinner.xml*. Después definimos el comportamiento al desplegarlo que sea el normal por defecto y asociamos este *adapter* a la variable asociada al *spinner*.

Tras esto nos quedan los botones, que como hemos venido haciendo hasta aquí los manejamos con *OnClickListener()* y la función *onClick()*. De esta forma, si se pulsa el botón de búsqueda, cogemos los datos introducidos en los capos de filtrado y lanzamos una nueva actividad a la que pasamos dichos datos. Esta nueva *Activity* cargará la clase encargada de ver las farmacias encontradas que es la veremos después de explicar el funcionamiento del botón de ayuda.

En el *onClick()* del botón de ayuda también lanzamos una nueva actividad que carga la clase ***AyudaBuscador.java*** cuya única instrucción será la de cargar la pantalla *ayudabuscador.xml* que se cargará como *popup* ya que así la hemos declarado en el fichero *AndroidManifest.xml*.

Ahora sí, veamos qué es lo que hace nuestra clase ***VerFarmacias.java***. Esta clase cargará la pantalla *verfarmacias.xml* y tras sacar los datos del *bundle*, miraremos cuantas farmacias hay que coincidan con los datos de zona, nombre y dirección. En caso de que el número sea 0, mostraos un mensaje de error en pantalla, en caso contrario, consultamos a la base de datos para obtener esas farmacias y luego las mostramos en pantalla en modo de lista utilizando para ello los mismos métodos descritos en la parte respectiva al listado de las farmacias de guardia. La única diferencia es que en este caso en vez de llamar a la clase *ElementoListaAdapterGuardia.java,* llamamos a la clase ***ElementoListaAdapter.java,*** que utilizara para dar formato a cada elemento de la lista la clase ***ElementoLista.java*** en vez de la usada antes *ElementoListaGuardia.java*. Entre estas dos, la única diferencia radica en que la de las farmacias de guardia configuraba para visualizarse el nombre y la zona, y en esta ocasión visualizaremos el nombre del farmacéutico y la dirección de la farmacia. De igual modo que antes si pulsamos en una de las farmacias de la lista, saltaremos a la clase *VerDetalles.java* mediante un *Intent*, y veremos ahí toda la información de la farmacia descrita anteriormente.

Tras la búsqueda, vamos ahora al botón con el que accedemos a la lista completa de todas las farmacias. Este botón llama a la clase ***ListaFarma.java*** que es la encargada de mostrar el *layout* de igual nombre y tras obtener de la base de datos todas las farmacias, las saca a pantalla a través de otro listado de características iguales al comentado justo antes en *VerFarmacias.java*. Al ser el mismo tipo de listado recibe también el mismo tipo de tratamiento, por lo que si pulsamos en una de las farmacias también iremos a ver toda su información y podremos llamar o verla en el mapa.

De vuelta a nuestra pantalla principal, nos quedan por ver el funcionamiento de 3 botones, dos de ellos, el de ayuda y el de salir, de carácter más sencillo que el encargado de la lectura de los QR. Por ello explicaremos antes el funcionamiento de los primeros y dejaremos para el final el complejo.

El botón de ayuda realiza una función idéntica al botón de ayuda que nos encontramos en la pantalla del buscador y que anteriormente se ha explicado. Al pulsarlo, se lanzará una actividad que llamará a la clase ***Ayuda.java*** que lo que hará será cargar la pantalla *ayuda.xml* también en modo emergente.

Por otro lado, el botón de salir lo que hace es terminar con la ejecución de la aplicación mediante la función *finish()*.

Ahora sí, entramos a explicar el funcionamiento del botón encargado de la lectura de los códigos QR. Antes de nada explicar que para poder leer los códigos QR, necesitamos cargar un programa especializado en esa tarea y luego trabajar con aquello que lea. Para poder conseguir realizarlo y tras investigar por internet maneras de hacerlo, dimos con una librería llamada *zxing* que nos permitía conseguir el funcionamiento que queríamos. Tras ver la cantidad de opciones y clases que tenía dicha librería, vimos que con agregar tan sólo dos de esas clases a nuestro proyecto, éramos capaces de obtener el resultado deseado. Estas clases son ***IntentIntegrator.java*** e ***IntentResult.java,*** donde la primera de ellas se encarga de buscar en el móvil aquellas aplicaciones que sean capaces de leer códigos de los tipos que se definen también en esta clase. Nos da la opción de configurarle una serie de variables de manera que si no encuentra ninguna aplicación de ese estilo, nos muestra un mensaje. En nuestro caso, hemos configurado ese mensaje para que informe al usuario de que se requiere una aplicación lectora de códigos QR y le damos la opción de si quiere instalar una. Elegimos la aplicación *BarcodeScanner* ya que es una aplicación de lectura de códigos sencilla y que ocupa poco espacio, por eso si el usuario acepta la opción de instalarla, se le abre el *play store* directamente en esa aplicación. La segunda de las clases es la encargada de empaquetar el resultado de la lectura que se hace tras invocar la clase anterior.

Una vez explicado esto, en la función *onClick()* de nuestro botón, creamos un *Intent* pero esta vez le pasamos como acción el comando *com.google.zxing.client.android.SCAN* e iniciamos la actividad de forma que esperamos un resultado con la función *startActivityForResult()*. Dicho resultado lo controlaremos con la función *onActivityResult()* donde si el código resultado de la lectura es distinto de 0 significa que se ha conseguido ejecutar la aplicación. Después extraemos los datos leídos a una variable de tipo *String* y si contiene algún elemento, llamamos a la función *descifrarCodigoQR()* pasándole todo lo que se haya leído. En caso de que lo leído esté vacio mostramos un mensaje informando al usuario.

La función *descifrarCodigoQR()* analizará lo que se ha leído, que en nuestro caso los códigos QR son cadenas de texto que contienen en la primera parte el texto *Geoloc-NafarmaCod* que utilizamos para saber que el código leído es nuestro. Tras este texto y separado por el signo dólar, tenemos la coordenada latitud del punto donde se encuentra el QR posicionado, y separado por otro signo de dólar, tenemos la coordenada de longitud. Cogemos ambos datos y los pasamos con un *Bundle* a la nueva actividad que lanzaos y que llamará a la clase *VerMapaQR.java* que explicamos a continuación.

La clase ***VerMapaQR.java*** carga la pantalla *vermapaqr.xml* y lo primero que hace es crear un diálogo de proceso igual al que describimos cuando se actualizaban las farmacias, y ejecutar una tarea asíncrona sobre la clase ***RellenaMapa()***. Esta clase tiene la misma estructura que la asíncrona descrita páginas más arriba, con las diferentes funciones que se ejecutan en orden.

En la función *onPreExecute()* lo que hacemos es dependiendo la hora que sea, meter en un array de tipo *Farmacia* todas las farmacias, las de guardia o sólo las que abren 24 horas, dependiendo si es la hora esta dentro del horario laboral, en horario de guardia normal o en un horario en que solo están abiertas las farmacias que no cierran en ningún momento. De esta forma al entrar en la función *doInBackground()*, lo que hacemos es recorrer el *array* de farmacias y para cada una ir metiendo en un *array* de *LatLng* los puntos creados a partir de sus coordenadas de latitud y longitud. Una vez acabada esta función, vamos a *onPostExecute()* donde añadimos por cada punto del *array* que hemos llenado en el paso anterior, un marcador en el mapa que controlamos en la clase *VerMapaQR.java*. Estos marcadores serán igual que los que posicionábamos en el otro mapa, de color verde y con la opción de que al pulsarlos nos muestren el nombre y la dirección. Tras esto cerramos el diálogo y volvemos a la clase en la que estábamos.

Una vez que ya hemos llamado asíncronamente a la clase anterior, cogemos del *Bundle* la latitud y longitud leída del QR y lo que hacemos es añadir ese punto al mismo mapa que hemos rellenado, pero indicándole que este sea de color rojo para diferenciarlo de los marcadores de farmacia. Además configuraremos el zoom del mapa dependiendo de la hora, de manera que si es horario laboral se acerque más para ver mejor nuestra posición, mientras que si es horario de guardia esté más alejado para poder ver mejor las farmacias abiertas que pueden quedar más distantes del punto donde se encuentra el QR.

Con esto acabaríamos la explicación de todas las clases que son las encargadas del funcionamiento de la aplicación y sólo nos queda explicar el fichero *AndroidManifest.xml.*

4.4 Fichero AndroidManifest.xml:

En este fichero están definidos todos aquellos permisos que necesita la aplicación para poder ejecutar todas sus funciones. Para poder consultar si tenemos una conexión de datos, necesitamos definir el permiso *ACCESS_NETWORK_STATE* así como el permiso *INTERNET.* Para poder realizar la llamada necesitamos añadir el permiso *CALL_PHONE* mientras que los demás permisos son aquellos necesarios para poder actuar con los mapas de *google*.

Tras los permisos, se define la aplicación y en ella debemos definir unos metadatos de *google* así como una *API_KEY* necesaria para poder trabajar con los mapas en nuestra aplicación. A continuación viene la definición de cada una de las *Activity* que lanzamos en nuestras clases y en las que definimos que aparezca el nombre de la aplicación en la barra superior y cómo ha de ejecutarse cada actividad, por lo general en modo *portrait*, es decir pantalla vertical, salvo las pantallas de ayuda que como ya hemos dicho en su momento, se ejecutan como *popup* con el estilo *Theme.Dialog.*

De este modo ya tendríamos explicada toda nuestra aplicación.

5 – Pruebas de usabilidad:

Para llevar a cabo las pruebas de uso, nos hemos puesto en todas las situaciones que hemos creído posibles, empezando por mi móvil con el que he ido probando cada paso que hacía en la programación conectándolo con el programa *eclipse* habilitando las opciones de desarrollador. De esta manera he podido ir probando y corrigiendo errores corriéndolo en modo *debug*.

Una vez probada la *app* en mi dispositivo, lo siguiente ha sido probarla en otros móviles con diferentes tamaños de pantalla para verificar que no se desencuadraba ninguna parte de las pantallas. Probando en cada uno de estos teléfonos todas las opciones que nos ofrece la aplicación y con las diferentes variantes que podemos tener como por ejemplo el hecho de tener o no conectados datos en el móvil.

Otra de las pruebas importantes en los dispositivos ha sido la de comprobar que toda la parte encargada de la lectura de QRs se realiza correctamente. Tanto la parte de cargar los programas que capturan el código, como la parte de descifrado y posterior muestreo del mapa. Hemos visto que en efecto cuando un móvil no tiene ninguna aplicación capaz de leer estos códigos, se muestra la ventana que da opción a descargarse la que hemos elegido *Barcode Scanner*. Sin embargo, nos ha pasado que algunos móviles abren la cámara pero no son capaces de leer el código ya que realmente no tienen una *app* para ello. Sería un punto a estudiar porqué pasa y cómo se podría solucionar.

Para ver que realmente nuestra aplicación cumple con las premisas que nos marcamos en un principio, interfaz sencilla e intuitiva que permita a cualquier usuario manejarla y navegar por ella sin dificultad, hemos realizado varias pruebas entre amigos y familiares de distintos rangos de edad para ver si encontraban algún problema o si les parecía difícil desenvolverse en ella. En ningún caso no hemos intervenido para explicarles el funcionamiento o qué debían hacer ya que esto nos anularía completamente el experimento. El resultado ha sido muy bueno e incluso ha habido quién proponía posibles mejoras para la aplicación, de las cuales algunas vienen explicadas en el siguiente apartado.

6 – Líneas futuras:

Una vez lista la aplicación y probada, nos surgen algunas posibles implementaciones de mejora o de usabilidad que en un futuro se podrían llegar a realizar. La primera y quizás más importante sería la de proponérselo al gobierno de navarra o al colegio de farmacéuticos de navarra para ver si les interesaría publicarla de manera oficial. También se podría proponer extender el rango de alcance hasta tener todas las farmacias de Navarra controladas en la aplicación.

Ver la viabilidad de la aplicación y la fabricación de los códigos QR así como su instalación. El hecho de tener que generar un código QR distinto para cada lugar puede ser costoso, de manera que una posible solución sería poner los códigos con la localización integrada sólo en puntos importantes y en el resto de lugares como por ejemplo las farmacias, poner un mismo código que al leerse, el programa coja la localización del móvil mediante GPS.

Otras posibles mejoras podrían llegar en línea al funcionamiento de las villavesas de Pamplona, es decir, desarrollar en dos partes el funcionamiento de la aplicación. De esta manera tendríamos por una parte la aplicación para móviles y por otro lado los códigos QR que nos redireccionarían a una aplicación web donde se mostrase el mapa con las diferentes localizaciones. También sería interesante otorgar la posibilidad de poder elegir idioma por ejemplo entre castellano, euskera e inglés.

Ya por último estaría la posibilidad, casi necesidad, de desarrollar la misma app para iOS y Windows Phone ya que de esta manera se conseguiría mayor espectro de usuarios al no limitarlos sólo a móviles con sistema Android.

7 – Conclusión y agradecimientos:

Para finalizar y a modo de conclusión, decir que decidí realizar este tipo de proyecto porque me entusiasmaba la idea de meterme en el mundo android ya que es una rama de programación que ofrece muchas posibilidades y muy variadas con la aparición en el mercado tecnológico de nuevos aparatos controlados por *Android* como los *smart watch*, *Android TV*, etc.

También creo que como informático y de cara al mundo laboral, es importante tener conocimiento de todo tipo de lenguajes de programación y más de aquellos que están creciendo tan rápido. De hecho, en la empresa en la que entré a trabajar al poco de empezar con este proyecto, me he encontrado varias aplicaciones para empresas en las que se les ha creado una aplicación para el móvil que muestre posibles alarmas o avisos que surjan en la producción.

Aunque ha sido costoso y por momentos duro compaginar trabajo y proyecto, estoy satisfecho del trabajo realizado y de todo el aprendizaje constante que me ha llevado a poder realizar el proyecto, a partir de aquí seguiré investigando el mundo *Android* y estoy seguro que no será mi última aplicación.

Agradecer a mi familia y amigos por su apoyo y sus ánimos, a mi compañero de trabajo Julián y a mi tutor Alfredo Pina por su paciencia, constancia y supervisión.

8 – Bibliografía y software:

- El Gran libro de Android –Tomás Girones, Jesus
- Página web de Google Developers
  http://developer.android.com/develop/index.html
- http://www.androidcurso.com
- Eclipse jungo + ADT Bundle
- SQLiteBrowser
- Adobe Photoshop
- Web crea iconos
  http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html
- Web crea QR
  http://www.codigos-qr.com/generador-de-codigos-qr/

Anexo 1: Codigos QR

QR – Hospital de Navarra

QR – Virgen del Camino

QR – Nuevas Urgencias

QR – Ambulatorio San Martín

QR – Oficina de turismo

Anexo 2: Fichero farmacias de guardia JSON (fragmento)


```
[
    {
        "FECHA": "02/03/2015",
        "DESDE": 9,
        "HASTA": 22,
        "LOCALIDAD": "ESTELLA",
        "GRUPO": "Estella",
        "DIRECCIÃ‘N": "PÂ° de la Inmaculada, 70",
        "FARMACIA": "Echeverria Garisoain, Maria Rosa",
        "Cod_FARMACIA": "F00298",
        "TELÃ‰FONO": "948 546534"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 10,
        "HASTA": 21,
        "LOCALIDAD": "HUARTE",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ‘N": "C/ Intxaurdia, 5. Centro Comercial
Itaroa",
        "FARMACIA": "Vallejos Huesa, Maria Josefa",
        "Cod_FARMACIA": "F00611",
        "TELÃ‰FONO": "948 357364"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 10,
        "HASTA": 21,
        "LOCALIDAD": "PAMPLONA",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ‘N": "Avda Bayona, 37",
        "FARMACIA": "Arriazu Aranceta, Berta",
        "Cod_FARMACIA": "F00618",
        "TELÃ‰FONO": "948 252401"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 10,
        "HASTA": 21,
        "LOCALIDAD": "PAMPLONA",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ‘N": "Avda Sancho el Fuerte, 31",
        "FARMACIA": "Gimeno Oses, Maria Pilar",
        "Cod_FARMACIA": "F00502",
        "TELÃ‰FONO": "948 198003"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 10,
        "HASTA": 21,
        "LOCALIDAD": "PAMPLONA",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ‘N": "Pl Iturriotzeaga, 4 trasera. Rochapea",
```

```
        "FARMACIA": "GoÃ±i Latasa, Mercedes",
        "Cod_FARMACIA": "F00147",
        "TELÃ‰FONO": "948 136706"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 10,
        "HASTA": 21,
        "LOCALIDAD": "PAMPLONA",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ“N": "C/ Yanguas y Miranda, 17",
        "FARMACIA": "Marfil Garcia, Alberto",
        "Cod_FARMACIA": "F00205",
        "TELÃ‰FONO": "948 245030"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 10,
        "HASTA": 21,
        "LOCALIDAD": "PAMPLONA",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ“N": "C/ Puente Miluce,  2. C. Comercial
Carrefour",
        "FARMACIA": "Navarro Zapatero, Maria Eugenia",
        "Cod_FARMACIA": "F00480",
        "TELÃ‰FONO": "948 199896"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 10,
        "HASTA": 21,
        "LOCALIDAD": "SARRIGUREN",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ“N": "Avda ErripagaÃ±a, 24. Ripagaina",
        "FARMACIA": "Ballesteros Morales, Francisco Javier",
        "Cod_FARMACIA": "F00731",
        "TELÃ‰FONO": "948787058"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 21,
        "HASTA": 10,
        "LOCALIDAD": "ARTICA",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ“N": "C/ Madres de la Plaza de Mayo, 32. Nuevo
Artica",
        "FARMACIA": "Aguado Menendez, Carmen",
        "Cod_FARMACIA": "F00739",
        "TELÃ‰FONO": "948 307609"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 22,
        "HASTA": 9,
        "LOCALIDAD": "ESTELLA",
        "GRUPO": "Estella",
```

```
        "DIRECCIÃ‘N": "PÂ° de la Inmaculada, 70",
        "FARMACIA": "Echeverria Garisoain, Maria Rosa",
        "Cod_FARMACIA": "F00298",
        "TELÃ‰FONO": "948 546534"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 21,
        "HASTA": 10,
        "LOCALIDAD": "PAMPLONA",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ‘N": "C/ Yanguas y Miranda, 17",
        "FARMACIA": "Marfil Garcia, Alberto",
        "Cod_FARMACIA": "F00205",
        "TELÃ‰FONO": "948 245030"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 21,
        "HASTA": 10,
        "LOCALIDAD": "SARRIGUREN",
        "GRUPO": "Pamplona y comarca",
        "DIRECCIÃ‘N": "Avda ErripagaÃ±a, 24. Ripagaina",
        "FARMACIA": "Ballesteros Morales, Francisco Javier",
        "Cod_FARMACIA": "F00731",
        "TELÃ‰FONO": "948787058"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 9,
        "HASTA": 9,
        "LOCALIDAD": "ABARZUZA",
        "GRUPO": "Comarca de Estella",
        "DIRECCIÃ‘N": "Ctra. de Arizala, 1",
        "FARMACIA": "Pascual Echavarri, Miguel Angel",
        "Cod_FARMACIA": "F00242",
        "TELÃ‰FONO": "948 520082"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 9,
        "HASTA": 9,
        "LOCALIDAD": "ABLITAS",
        "GRUPO": "BuÃ±uel - Cascante",
        "DIRECCIÃ‘N": "Pl Fueros, 10",
        "FARMACIA": "Escalada Abraham, Montserrat",
        "Cod_FARMACIA": "F00422",
        "TELÃ‰FONO": "948 813811"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 9,
        "HASTA": 9,
        "LOCALIDAD": "ALTSASU/ALSASUA",
        "GRUPO": "Altsasu/Alsasua",
        "DIRECCIÃ‘N": "C/ La Paz, 47",
```

```
        "FARMACIA": "Perea/Uriarte Garcia/TotoricagÃ¼ena, Juan
A./Eva",
        "Cod_FARMACIA": "F00449",
        "TELÃ‰FONO": "948 564210"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 9,
        "HASTA": 9,
        "LOCALIDAD": "ARTAJONA",
        "GRUPO": "Artajona",
        "DIRECCIÃ“N": "C/ La Cruz, 51",
        "FARMACIA": "Buesa de Caceres, Francisco Javier",
        "Cod_FARMACIA": "F00581",
        "TELÃ‰FONO": "948 364080"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 9,
        "HASTA": 9,
        "LOCALIDAD": "AZAGRA",
        "GRUPO": "San AdriÃ¡n",
        "DIRECCIÃ“N": "C/ RamÃ³n y Cajal, 34",
        "FARMACIA": "Cirarda Sasia, Pablo",
        "Cod_FARMACIA": "F00136",
        "TELÃ‰FONO": "948 692105"
    },
    {
        "FECHA": "02/03/2015",
        "DESDE": 9,
        "HASTA": 9,
        "LOCALIDAD": "BERA",
        "GRUPO": "Lesaka",
        "DIRECCIÃ“N": "Pl de los Fueros, 6",
        "FARMACIA": "Fernandez Diez, Arantzazu",
        "Cod_FARMACIA": "F00353",
        "TELÃ‰FONO": "948 630631"
    },
```

Anexo 3: Código Android Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="es.iker.farmacias"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="11"
      android:targetSdkVersion="19" />

   <!-- Google Maps related permissions -->
   <permission
      android:name="com.ecs.google.maps.v2.actionbarsherlock.permission.MAPS_RECEIVE"
      android:protectionLevel="signature" />

   <!-- Network connectivity permissions -->
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
   <uses-permission android:name="android.permission.INTERNET" />

   <!-- Access Google based webservices -->
   <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />

   <!-- Maps API needs OpenGL ES 2.0. -->
   <uses-feature
      android:glEsVersion="0x00020000"
      android:required="true" />

   <!-- Calling permission -->
   <uses-permission android:name="android.permission.CALL_PHONE" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <uses-library android:name="com.google.android.maps" />

      <meta-data
         android:name="com.google.android.gms.version"
         android:value="@integer/google_play_services_version" />
      <meta-data
         android:name="com.google.android.maps.v2.API_KEY"
         android:value="AIzaSyBGzg4WNGL0QCqeUzAKCUTfR-Y5O7cHqIw" />

      <activity
         android:name="es.iker.farmacias.NaFarma"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity
         android:name=".Ayuda"
```

```
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Dialog" >
        </activity>
        <activity
            android:name=".AyudaBuscador"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Dialog" >
        </activity>
        <activity
            android:name=".Buscador"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".FarmaGuardia"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".ListaFarma"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".VerFarmacias"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".DetallesFarmacia"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".VerMapa"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".VerMapaQR"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
        </activity>
    </application>

</manifest>
```