UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Desarrollo de una aplicación de gestión de tareas para Android

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática, Rama de Ingeniería del Software

Curso: 2013/2014 – Convocatoria Julio

Autor: Sammy Patenotte

Tutor: Vicente Pelechano Ferragud

Índice

1 Tabla de Ilustraciones

2 Introducción

2.1 Motivación

La principal motivación de este Trabajo Final de Grado (TFG) es el aprendizaje de la programación de aplicaciones en la plataforma *Android*.

He elegido *Android* porque me parece interesante y actualmente es la plataforma más extendida en el ámbito de los dispositivos móviles, con aproximadamente un 78.6% de cuota de mercado (Llamas R., Reith R. y Shirer M., 2014).

Esta presencia la hace una oportunidad muy grande para alcanzar un mercado muy amplio e internacional, aprovechando las oportunidades económicas y emprendedoras que permite la aparición de una nueva plataforma cómo lo puede ser *Android.*

También ha permitido el auge de una nueva categoría de empleos, la de desarrollador de aplicaciones para teléfonos móviles, una competencia que se pide cada vez más, y abre un nuevo camino para el futuro de los ingenieros informáticos.

Además, al ser basado en Java, *Android* es el candidato perfecto para desarrollar este TFG, ya que es el principal lenguaje de programación enseñado durante el Grado de Ingeniería Informática.

2.2 Objetivo del Trabajo Final de Grado

El objetivo de este TFG es desarrollar una aplicación de gestión de tareas para la plataforma *Android*.

En el mercado de aplicaciones de Android, el *Google Play Store*, ya existen muchas aplicaciones de este tipo. Estas aplicaciones, a lo largo de sus sucesivas versiones han visto su funcionalidad extenderse cada vez más, llegando a ser tan completos que su utilización puede haberse hecho complicada para los usuarios inexpertos de la plataforma.

En este TFG se ha querido tener en cuenta esto, y hacer una aplicación que quizás tenga menos funcionalidad, pero que sea fácil de usar por cualquier usuario de un *Smartphone* con Android.

En esta aplicación una tarea se considera cómo cualquier trabajo o actividad que el usuario tenga que realizar en un futuro más o menos lejano.

La principal funcionalidad de la aplicación será poder crear y definir tareas, con su correspondiente alarma. La alarma sonará en el día y la hora que el usuario haya elegido, pudiendo además definir recordatorios, que notificarán al mismo que tiene que trabajar en la tarea asociada con el recordatorio.

El usuario podrá definir la fecha y hora en la que empezarán los recordatorios, además de su frecuencia.

Cuando suene la alarma o algún recordatorio, el usuario podrá consultar el título y la descripción de la tarea, y borrarla si ya la ha completado.

También tendrá una vista de calendario, para poder identificar rápidamente las fechas próximas de tareas pendientes, que serán resaltadas, y se podrán consultar de una pulsación en la tarea correspondiente.

2.3 Aplicaciones existentes

Como se ha comentado antes, existe una gran cantidad de aplicaciones de gestión de tareas para *Android*.

En este apartado, veremos dos de las más conocidas y utilizadas por los usuarios de esta plataforma.

La más conocida de ellas es *Any.DO.* Tiene una interfaz original, distinta de lo que recomiendan las guías de diseño de *Google*, por lo que puede desconcertar a algunos usuarios habituados a las aplicaciones que siguen esas guías de diseño.

En el apartado de la funcionalidad, tiene mucha funcionalidad, todo lo que se podría pedir de un gestor de tareas, además de tener reconocimiento de voz, sincronización con *Google* y soporte de los movimientos del móvil (por ejemplo agitar el teléfono para suprimir las tareas completadas).

Es una aplicación muy completa, pero que, al diferenciarse mucho de la apariencia convencional de las aplicaciones *Android*, puede ser difícil de manejar por un usuario inexperto.

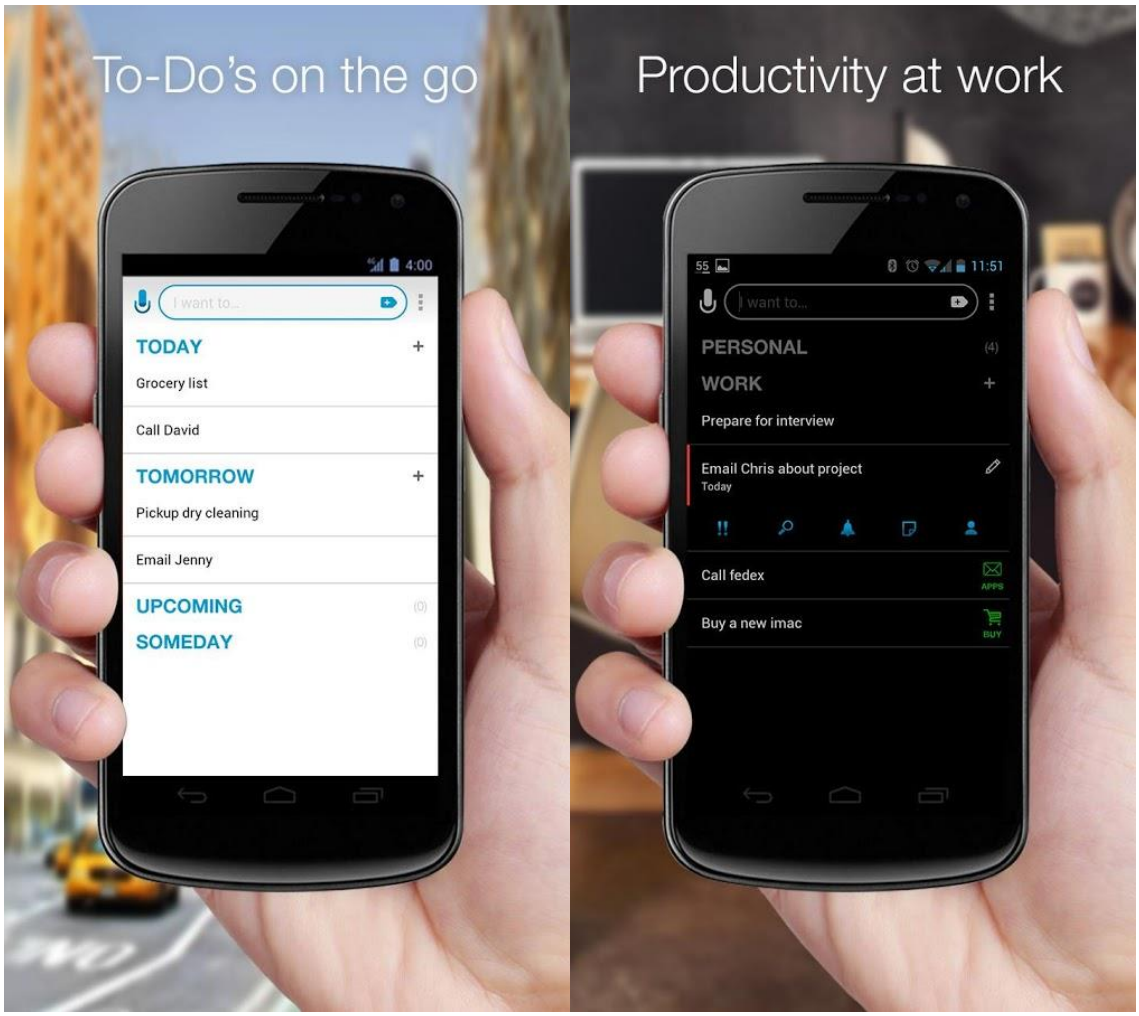

Ilustración 1 - Capturas de Any.Do en el Google Play Store

Otra de las aplicaciones de gestión de tareas disponibles para *Android* es *Todoist.* Esta aplicación sí que sigue las guías de diseño de *Android,* y tiene una interfaz fácil de entender.

Esta aplicación proporciona las funciones habituales de gestión de tareas, además de poder sincronizar las tareas con Dropbox y Google Drive, añadir imágenes y videos a las tareas.

Su mayor defecto podría ser que algunas funciones, tales como la definición de recordatorios o la vista de calendario necesitan la compra de la versión Premium, por lo que no todos los usuarios podrían disfrutar de estas funciones.

Ilustración 2 - Captura de Todoist en el Google Play Store

Podemos ver que estas aplicaciones son de muy buena calidad, pero que, dependiendo del tipo de usuario, no pueden convencer, o por su interfaz, o por sus funciones necesitando un pago. Por estas razones, la aplicación de este TFG puede atraer a nuevos usuarios que no se sienten cómodos con estas dos aplicaciones.

2.4 Herramientas utilizadas

2.4.1 Eclipse

Es la herramienta principal utilizada para este TFG. Es un entorno de desarrollo completo, desarrollado por la *Eclipse Foundation*. Permite diseñar, implementar y depurar proyectos en muchos lenguajes de programación., los principales siendo C, C++ y Java.

Además de soportar una gran cantidad de lenguajes de programación, *Eclipse* proporciona facilidades para programar usando estos lenguajes.

Por ejemplo, proporciona la coloración del código, para identificar fácilmente las distintas partes y funciones de este código. También posee un analizador sintáctico que notifica el usuario de los errores presentes en su proyecto y, en el caso de que sea posible, propone soluciones rápidas a estos errores.

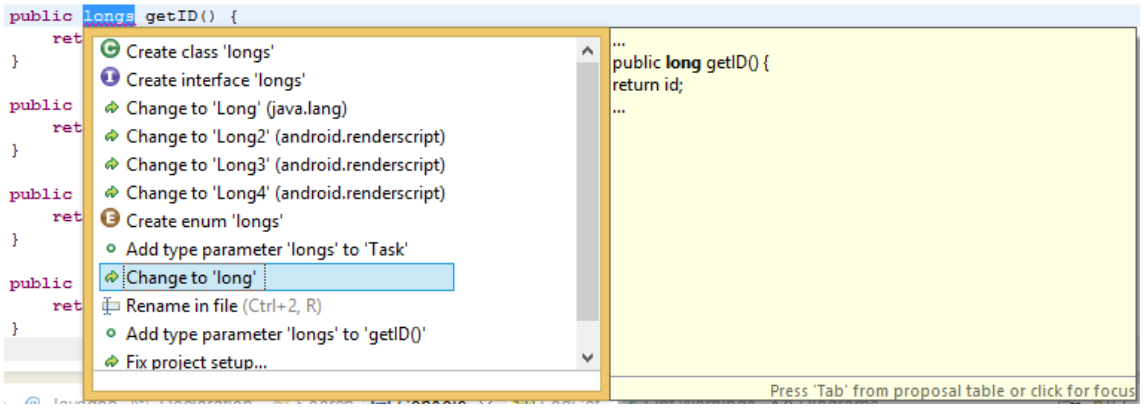

Ilustración 3 - Ejemplo de solución proporcionada por el analizador sintáctico

Junto a Eclipse se han utilizado otras herramientas que se describen a continuación, que gracias a su sistema de *plug-ins* permiten extender la funcionalidad de este entorno de desarrollo para poder soportar proyectos destinados a la plataforma Android.

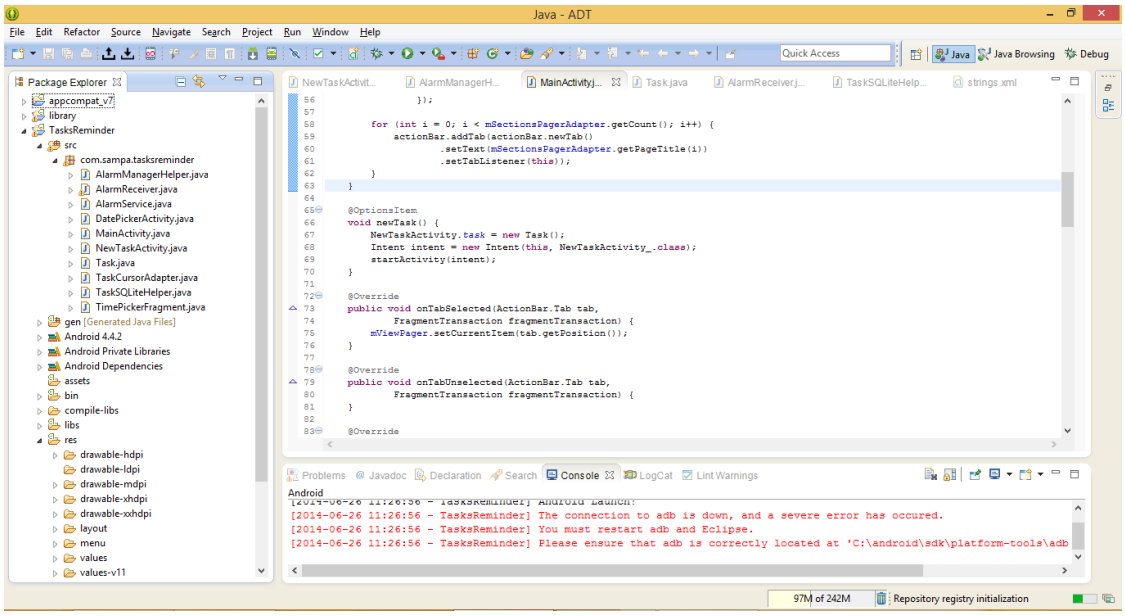

Ilustración 4 - Entorno de desarrollo Eclipse

2.4.2 ADT Plugin for Eclipse

Este *plug-in* es el que permite extender la funcionalidad de Eclipse para soportar la plataforma Android. Permite tener las mismas facilidades que propone Eclipse (coloración y verificación sintáctica de código, autocompletado, etc...) pero en proyectos destinados a dispositivos con Android.

Se puede obtener desde la página *web* oficial para los desarrolladores *Android*[1], y para instarlo se tienen que seguir las instrucciones proporcionadas por la misma página de descarga.

[1] http://developer.android.com/sdk/installing/installing-adt.html

2.4.3 Android SDK Tools

El componente *Android SDK Tools* nos proporciona todas las herramientas necesarias para compilar y ejecutar las aplicaciones Android implementadas con Eclipse. Sus principales componentes son un compilador, un emulador y un depurador.

Desafortunadamente, aunque se ha mejorado notablemente con las últimas versiones de Android, el emulador proporcionado por los *Android SDK Tools* no es la mejor herramienta para depurar aplicaciones Android, ya que es algo lento.

Por lo tanto, para depurar la aplicación objeto de este TFG se ha utilizado un teléfono móvil con Android conectado por USB al ordenador donde se ha desarrollado dicha aplicación.

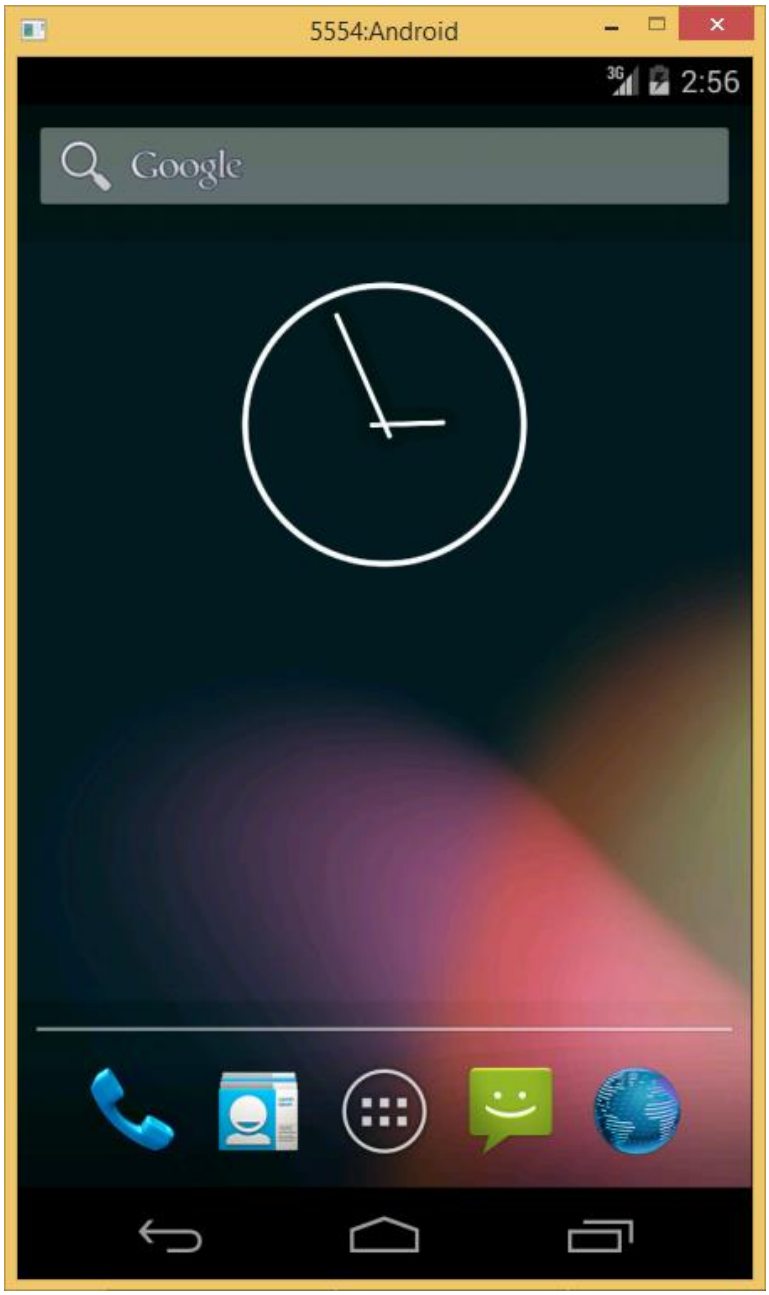

Ilustración 5 - Emulador proporcionado por los Android SDK Tools

2.4.4 WireFrameSketcher

WireFrameSketcher es una herramienta de diseño de bocetos basada en *Eclipse*. Ha sido desarrollada por Petru Severin en 2008, primero como *plug-in* para *Eclipse,* y a continuación se convirtió en una aplicación independiente.

Esta aplicación proporciona muchas plantillas para diseñar bocetos en muchos sistemas distintos. En este caso, se van a usar las plantillas para Android.

Su interfaz permite arrastrar los elementos gráficos deseados en la ventana de trabajo para formar la interfaz necesaria. Permite cambiar el texto contenido en estos mismos elementos y añadir elementos externos.

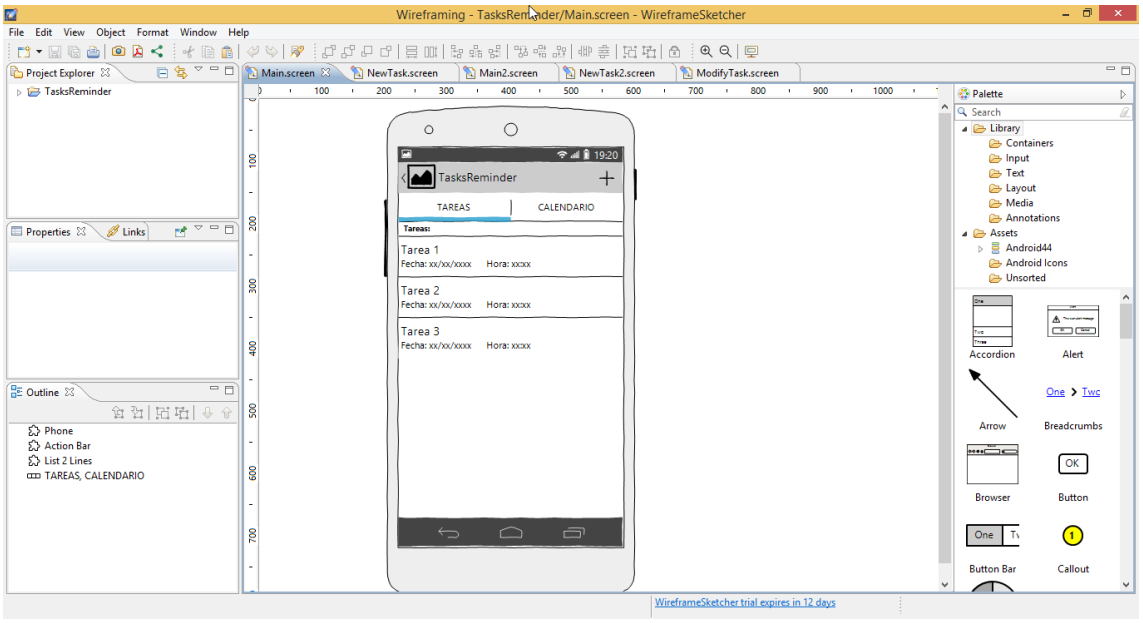

Ilustración 6 - Ventana principal de WireFrameSketcher

2.5 Estructura del documento

Este documento se compone de dos grandes partes:

- La primera tratará de las técnicas *software* y métodos de desarrollo utilizados durante la evolución de este TFG, mostrando la relación de este con la rama de *Ingeniería del Software.*
- La segunda parte tratará del proceso de desarrollo en sí. Tendrá cuatro grandes apartados
    - El primer apartado constará del análisis de los requisitos funcionales de la aplicación que se implementa. Es decir, la especificación de la funcionalidad que tendrá esta aplicación.
    - El segundo apartado consistirá en la definición de los casos de uso para definir los distintos pasos que se tienen que llevar a cabo para completar cualquier proceso posible en la aplicación.
    - A continuación, en el tercer apartado, se mostrarán los bocetos iniciales de la aplicación, para poder tener una idea de la apariencia final de la aplicación y su navegabilidad.
    - Para acabar, en el cuarto y último apartado, contaremos el proceso de implementación en sí.

3 Técnicas software y métodos de desarrollo utilizados

En los dos primeros apartados veremos las técnicas *software* que se han utilizado para diseñar e implementar este TFG. En los apartados siguientes veremos otros dos métodos de desarrollo que también se han utilizado en este proyecto. Estas técnicas y métodos se relacionan con la rama de *Ingeniería del Software*

3.1 Desarrollo iterativo e incremental

La principal técnica de *software* utilizada en este proyecto es la del desarrollo iterativo e incremental.

Este proceso es una mejora del proceso en cascada. Este último consiste en 6 etapas bien definidas. Las etapas son:

- Análisis de requisitos
- Diseño del sistema
- Implementación
- Pruebas
- Implantación
- Mantenimiento

Estas etapas se pueden solapar las unas a las otras.

Desafortunadamente, esta metodología, presentada por primera vez por Winston Royce en 1970 (Cataldi Z. et al., 2003), ha resultado inadaptada para el desarrollo real de *software* por dos principales razones: (McCracken D. y Jackson A., 1982)

- Los proyectos rara vez siguen tal linealidad, cuando se detectan errores se tiene que volver a etapas anteriores para solucionarlos, por lo que este modelo no es válido.
- Ya que el sistema no estará en funcionamiento hasta finalizar el proyecto, no se podrán arreglar posibles errores debidos a una incomprensión entre el cliente y el desarrollador del producto, dado que ya se habrán consumido casi la totalidad de los recursos.

La metodología de desarrollo iterativo e incremental permite solucionar algunos de los problemas del modelo en cascada.

Las etapas de esta metodología son muy parecidas respecto a las del primer modelo, pero con algunos cambios. Las cuatro primeras etapas son las mismas (análisis, diseño, implementación y pruebas) pero en el modelo iterativo, después de efectuar las pruebas, se vuelve a la etapa de análisis.

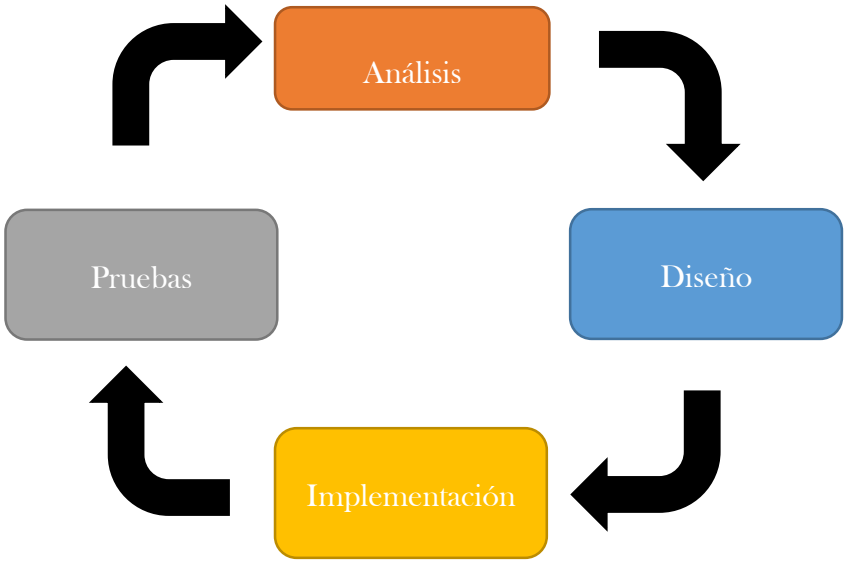

Ilustración 7 - Ciclo de desarrollo iterativo e incremental

El principal objetivo de esta metodología es empezar con un producto simple en las primeras iteraciones, e ir completándolo en las siguientes, añadiendo más funcionalidad para obtener un producto final de muy buena calidad.

Esto es posible gracias a las diferentes iteraciones, ya que en la etapa de análisis siguiendo la etapa de pruebas de la fase anterior, se pueden detectar errores de diseño, y arreglarlos antes de que sea más difícil hacerlo, debido al tamaño del sistema.

Por estas razones, la metodología de desarrollo iterativo e incremental me parece más adecuada, ya que permite construir un producto correcto y de calidad, evitando los errores de diseño gracias a las diferentes iteraciones. Además permite comprobar que la funcionalidad desarrollada en cada iteración funciona correctamente antes de pasar a la siguiente e añadir más funcionalidad.

3.2 Metodología Ágil (Feature-driven development)

El segundo concepto que se ha utilizado para llevar a cabo este TFG ha sido el de metodología ágil, más concretamente el *Feature-driven Development.*

El concepto de desarrollo ágil apareció por primera vez en el *Manifiesto para el desarrollo ágil de software* (Beck K. et al., 2001).

Este manifiesto apareció para definir nuevos métodos de desarrollo ligeros para competir con los métodos tradicionales, que se consideraban demasiado estrictos, y poco abiertos al cambio.

La metodología ágil se basa en el desarrollo iterativo e incremental. Suele usarse en proyectos pequeños, y a corto plazo. Cada iteración suele durar entre una y cuatro semanas, y en cada una de estas iteraciones se implementa una o varias funcionalidades, teniendo que tener siempre un producto funcional con el menor número de *bugs* posible.

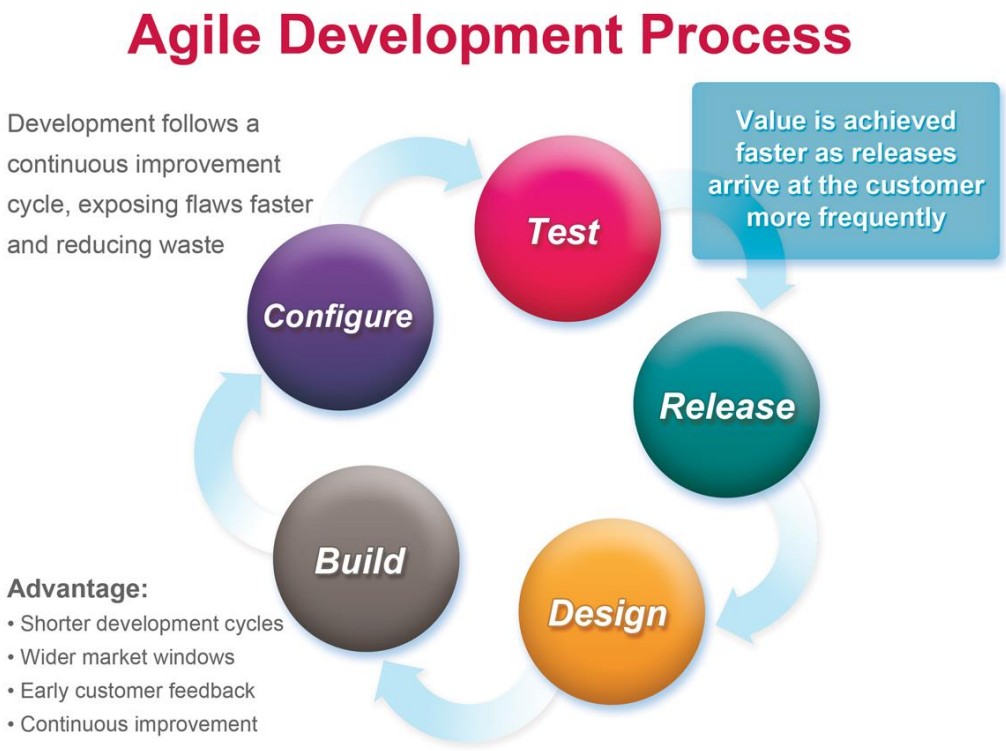

Ilustración 8 - Proceso de desarrollo ágil

En la ilustración 4, podemos ver una representación gráfica de un proceso de desarrollo ágil.

Podemos comprobar que efectivamente, la metodología ágil se basa en el modelo de desarrollo iterativo e incremental, ya que está compuesto por las distintas fases de este modelo (diseño, implementación, pruebas y entrega).

Podemos notar que también es cíclico, lo que es una prueba más de su afiliación con metodología de desarrollo iterativo e incremental.

Esta ilustración también nos explica las ventajas de utilizar el paradigma de desarrollo ágil frente a otros paradigmas más tradicionales.

Sus ventajas son las siguientes:

- Los ciclos de desarrollo son más cortos.
- Se obtiene el *feedback* del cliente más tempranamente, pudiendo comprobar que lo que se desarrolla en cada iteración es conforme a lo que el cliente desea.
- Se mejora el producto en cada iteración, ya que en cada una de ellas, se vuelve a analizar el producto, y cualquier problema de diseño o implementación se puede detectar y corregir lo más pronto posible.

En este proyecto, la metodología ágil que se ha usado ha sido la del *Feature-driven Development.*

Esta metodología tiene todas las características del desarrollo ágil, pero su punto de enfoque es la funcionalidad del producto. En cada iteración se tienen que implementar una o varias funcionalidades más en el producto, y comprobar que son correctas y libres de errores al final de estas iteraciones.

El *Feature-driven Development* permitirá incrementar la funcionalidad de la aplicación en cada iteración, teniendo una base funcional cada vez mayor, siempre de calidad y libre de errores.

Por lo tanto, todos estos argumentos nos hacen pensar que esta metodología ágil puede ser la más adecuada para desarrollar el objeto de este TFG.

Permitirá identificar errores de diseño muy tempranamente, y comprobar que cada vez que se implementa una nueva parte de la aplicación, esta funciona correctamente antes de pasar a la siguiente.

3.3 Diagrama de casos de uso

Los diagramas de casos de uso son una herramienta muy potente y muy utilizada en el desarrollo de *software*.

Permiten identificar claramente y de forma gráfica los actores (personajes o entidades) que formarán parte de la aplicación desarrollada, y los pasos que se deben de realizar para llevar a cabo cualquier operación posible en esa aplicación.

Cuando se diseña un diagrama de casos de uso, se puede identificar muy fácilmente la comunicación entre los distintos componentes del producto, y corregir potenciales errores incluso antes de empezar la implementación, lo que permite ahorrar mucho tiempo y dinero, dos recursos muy importantes para las empresas de desarrollo de *software*.

Esta herramienta podría ser muy útil para este TFG ya que permitirá identificar la secuencia de acciones necesarias para llevar a cabo cualquier operación posible en la aplicación desarrollada, y poder implementar esas operaciones sabiendo claramente cuál es el camino a seguir para hacerlo correctamente.

3.4 Prototipado de interfaz de usuario

El prototipado de interfaz de usuario es la técnica que consiste en identificar las interfaces de usuario que formarán parte del producto *software* que se desarrolla.

Se efectúa en la etapa de diseño, después de hacer el análisis de requisitos, pero antes de implementar el producto.

Es una herramienta muy potente para identificar de forma muy clara la navegabilidad entre los distintos componentes de una aplicación, y crear un diseño efectivo y coherente.

Permite tener una idea de la funcionalidad y apariencia del producto final sin tener que diseñar el sistema entero, y por lo tanto identificar incoherencias de diseño, incluso antes de empezar la implementación

Por estas razones, el prototipado de interfaz de usuario ha sido una herramienta más para desarrollar este TFG, dado sus ventajas.

4 Proceso de desarrollo

En este apartado, se va a explicar el proceso de desarrollo que se ha llevado a cabo para desarrollar la aplicación objeto de este TFG.

Lo primero que se ha hecho ha sido especificar los requisitos funcionales de esta aplicación

Después de tener claramente definidos estos requisitos, se ha pasado a definir los casos de uso basándose en la funcionalidad definida por los requisitos funcionales.

Estos primeros pasos forman parte del análisis de requisitos.

El siguiente paso ha sido diseñar unos *mockups* preliminares (que corresponden al prototipado de interfaz de usuario), en la etapa de diseño de la aplicación, para tener una idea gráfica de lo que se va a implementar.

Lo último fue la parte de implementación de la aplicación, donde se hace uso de todo lo planificado anteriormente para desarrollar un producto de calidad y con pocos fallos.

4.1 Requisitos funcionales

La primera parte del análisis de requisitos ha sido la especificación de los requisitos funcionales.

Los requisitos funcionales son unas herramientas que se suelen utilizar durante el desarrollo de un producto software, para determinar lo que hará el sistema implementado y las restricciones que puede tener.

Cada requisito consiste en una capacidad que permita al usuario conseguir un objetivo en la aplicación.

A continuación describiremos los requisitos funcionales de este TFG:

1. El usuario debe poder crear tareas y asignarles una alarma.
2. El usuario debe poder identificar estas tareas mediante un título o una descripción o fecha de fin.
3. El usuario debe poder modificar cualquier campo de una tarea.
4. El usuario debe poder especificar recordatorios. Estos recordatorios tienen que ser anteriores a la fecha de fin de la tarea.
5. El usuario debe poder suprimir una tarea.
6. El usuario debe poder ver las tareas en forma de lista o calendario.
7. El sistema debe avisar al usuario cuando se dispare algún recordatorio o la alarma de alguna tarea.
8. Al recibir una notificación de tarea, el usuario debe poder suprimir la tarea o consultar los datos de esta.

Habiendo especificado los requisitos funcionales de la aplicación, el siguiente paso es definir los casos de uso, usando estos requisitos.

4.2 Diseño de casos de uso

En esta aplicación, ya que el sistema no está compuesto de muchos actores, los diagramas de casos de uso resultan ser relativamente simples.

Podemos considerar los siguientes actores:

- El usuario de la aplicación
- El sistema de alarmas de Android
- El sistema de base de datos de Android

Los casos de uso posibles en nuestra aplicación son los siguientes:

- Crear Tarea
- Modificar Tarea
- Suprimir Tarea

A continuación describiremos cada uno de estos casos de uso, usando una plantilla creada por la Universidad Politécnica de Valencia[2]. Esta plantilla se ha elegido porque permite especificar precisamente en que consiste cada caso de uso, sus distintos pasos y el resultado final, y también los problemas que pueden ocurrir.

[2] http://users.dsic.upv.es/asignaturas/facultad/lsi/trabajos/032000(2).doc

4.2.1 Crear Nueva Tarea

CU- 01	Crear Nueva Tarea	
Descripción	El caso de uso correspondiente a la acción de crear una tarea, iniciada por el usuario de la aplicación	
Precondición	No hay precondición	
Secuencia Normal	Paso	Acción
	1	El usuario pulsa el botón de crear tarea nueva
	2	El sistema solicita los siguientes datos para crear la tarea nueva: título, descripción, fecha y hora. Si se habilitan los recordatorios también se solicitan fecha y hora de inicio de los recordatorios, y su frecuencia.
	3	Al pulsar el botón de aceptar, se guarda la tarea en la base de datos del sistema y se crea la alarma correspondiente
Resultado	El usuario ya tiene la tarea creada, y será notificado por la aplicación del próximo recordatorio o en la fecha final de la tarea	
Excepciones	Paso	Acción
	2	Si alguno de los campos obligatorios (título, fecha y hora) no se ha rellenado correctamente, se le notificará al usuario resaltando el primer campo que se ha rellenado incorrectamente
	2	Si se ha pulsado el botón de habilitar recordatorios y no se han rellenado los campos correspondientes, se le notificará al usuario resaltando el primer campo que se ha rellenado incorrectamente
	3	Si el usuario solicita cancelar la operación, el sistema cancela la operación, y se vuelve a la pantalla principal de la aplicación

4.2.2 Modificar Tarea

CU- 02	Modificar Tarea	
Descripción	El caso de uso correspondiente a la acción de modificar una tarea, iniciada por el usuario de la aplicación	
Precondición	Tener alguna tarea creada en la base de datos	
Secuencia Normal	Paso	Acción
	1	El usuario pulsa sobre alguna tarea ya creada
	2	El sistema busca la tarea en la base de datos y rellena los campos del formulario con los datos de la tarea.
	3	El usuario modifica los campos que desea
	4	Al pulsar el botón de aceptar, se sobreescribe la tarea en la base de datos del sistema, se cancela la alarma anterior y se crea una nueva alarma
Resultado	La tarea está modificada, y el usuario será notificado por la aplicación del próximo recordatorio o en la fecha final de la tarea	
Excepciones	Paso	Acción
	3	Si alguno de los campos obligatorios (título, fecha y hora) no se ha rellenado correctamente, se le notificará al usuario resaltando el primer campo que se ha rellenado incorrectamente
	3	Si se ha pulsado el botón de habilitar recordatorios y no se han rellenado los campos correspondientes, se le notificará al usuario resaltando el primer campo que se ha rellenado incorrectamente
	4	Si el usuario solicita cancelar la operación, el sistema cancela la operación, y se vuelve a la pantalla principal de la aplicación, dejando la tarea en su estado inicial.

4.2.3 Suprimir Tarea

CU- 03	Suprimir Tarea	
Descripción	El caso de uso correspondiente a la acción de suprimir una tarea, iniciada por el usuario de la aplicación	
Precondición	Tener alguna tarea creada en la base de datos	
Secuencia Normal	Paso	Acción
	1	El usuario pulsa el botón de suprimir de alguna tarea
	2	El sistema busca la tarea en la base de datos y la borra.
	3	El sistema cancela la alarma asociada a esta tarea.
Resultado	La tarea está suprimida.	
Excepciones	Paso	Acción
	--	--

El diagrama de casos de uso resultante es el siguiente:

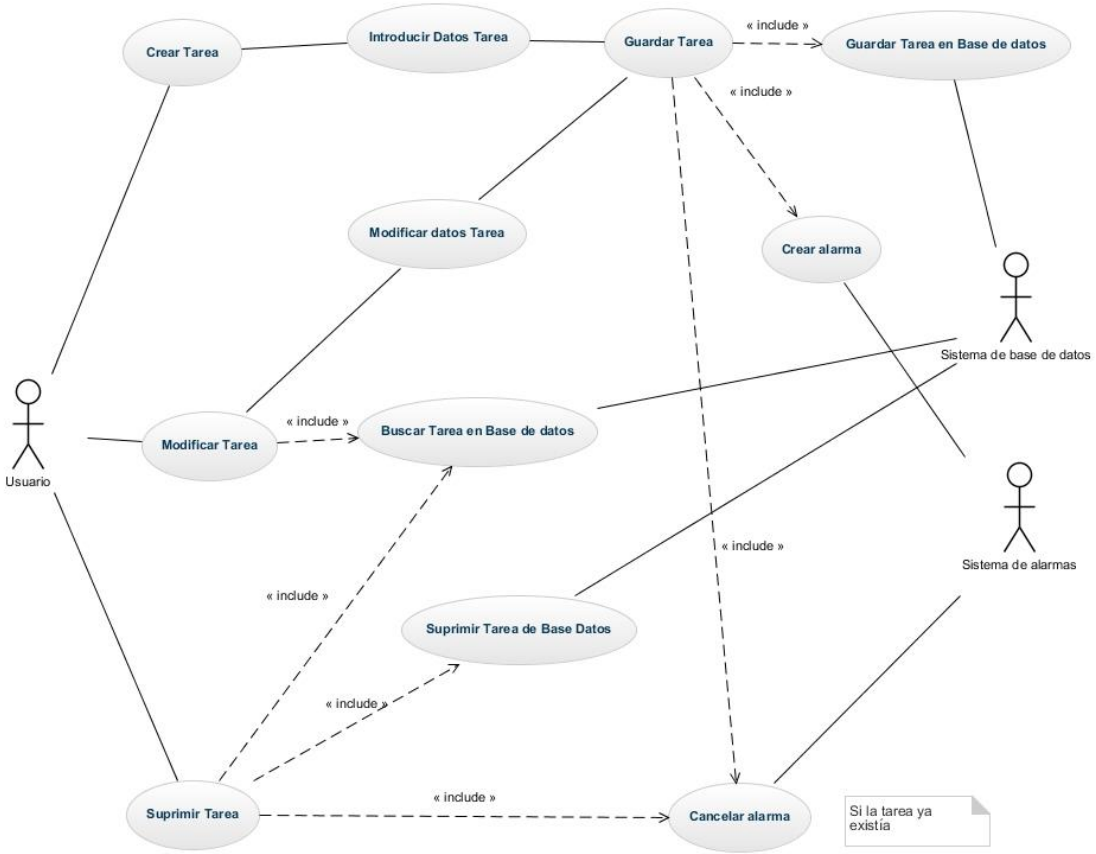

Ilustración 9 - Diagrama de casos de uso de la aplicación

4.3 Prototipado de la interfaz de usuario

Antes de empezar a implementar la aplicación, hay que efectuar la etapa de diseño de la misma.

En este apartado nos haremos una idea preliminar de la interfaz gráfica de la aplicación, gracias a unos *mockups*, es decir unos bocetos o borradores de las distintas pantallas presentes en las distintas partes de esta aplicación.

Para diseñar los bocetos, usamos la herramienta *WireframeSketcher*, mencionada anteriormente, que permite diseñar una interfaz gráfica muy fácilmente, arrastrando los elementos gráficos deseados en la pantalla.

A continuación vemos esos bocetos:

4.3.1 Pantalla principal

La pantalla principal se compondrá de dos pestañas, permitiendo ver las tareas pendientes de dos formas. La primera será una simple lista donde se podrá ver el título, fecha y hora de cada tarea. Pulsando sobre cualquiera de las tareas se accede a una nueva ventana que permitirá editar esa tarea o suprimirla si se desea La segunda pestaña será una vista de calendario que, al tener los días conteniendo tareas pendientes resaltados, permitirá ver muy rápidamente los días donde el usuario tiene trabajo. Pulsando sobre cualquier día se podrán ver las tareas de ese mismo día en forma de lista debajo del calendario. Pulsando en el botón en la esquina superior derecha se accede a la vista de crear nueva tarea.

Ilustración 10 - Bocetos pantalla principal

4.3.2 Crear Nueva Tarea

La vista de Crear Nueva Tarea será compuesta de un formulario, dónde estarán todos los campos necesarios para crear una nueva tarea. Los campos que tendrá son los siguientes:

- Título
- Descripción
- Fecha y hora
- Habilitar Recordatorios
- Fecha y hora de inicio de los recordatorios
- Frecuencia de los recordatorios

Ilustración 11 - Bocetos Nueva Tarea

4.3.3 Modificar tarea

La vista de modificar tarea será la misma que la de nueva tarea, aprovechando los recursos ya existentes. El único cambio será que los campos no serán vacíos, los que ya tengan valor serán automáticamente rellenados desde la base de datos. También se cambiará el título de la vista, en la esquina superior izquierda, será el de "Modificar Tarea".

Ilustración 12 - Boceto Modificar Tarea

4.4 Implementación

La implementación, conformemente a la metodología de desarrollo ágil *Feature-Driven development*, se hizo en varias iteraciones, añadiendo siempre nuevas funciones y comprobando siempre que el producto era funcional y libre de errores al acabar una iteración, antes de empezar la siguiente.

A continuación veremos un resumen de las diferentes iteraciones.

4.4.1 Primera iteración

En esta primera iteración, se implantó la base de la aplicación, es decir la primera pestaña de la pantalla principal, la vista de nueva tarea, la estructura de datos correspondiendo a una tarea y las clases para poder guardar las tareas en base de datos.

Se partió de un proyecto por defecto generado por *Eclipse*, que genera una aplicación *Android* ejecutable, con una pantalla vacía conteniendo una *ActionBar,* que consiste en una barra de tareas con el nombre de la aplicación, y tres pestañas, con el contenido por defecto.

Después se añadió el botón en la *ActionBar*, para poder añadir tareas nuevas.

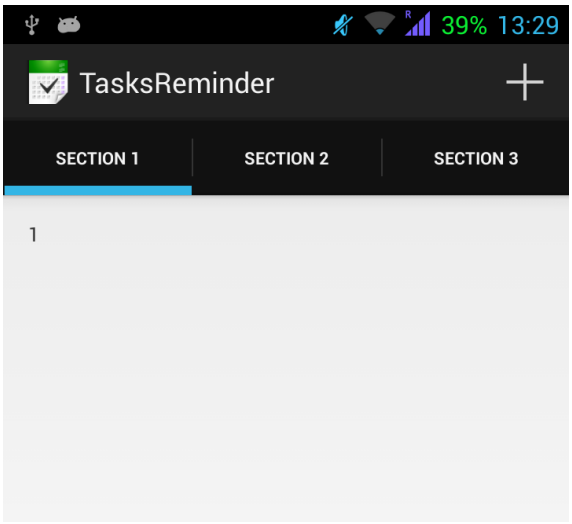

Ilustración 13 - Apariencia inicial de la aplicación

A continuación se creó la clase Tarea, que será la que soportará el objeto de tipo Tarea.

Durante la evolución del proyecto sufrió varios cambios, pero en su estado final se compone de los siguientes campos:

- **id**, una variable de tipo *long*, almacenando la posición de la tarea en la base de datos
- **title**, una variable de tipo *String* representando el título de la tarea
- **description**, una variable de tipo *String* representando la descripción de la tarea
- **date** y **dateReminder**, unas variables de tipo *LocalDate* representando respectivamente la fecha de la tarea y la de los recordatorios
- **time** y **timeReminder**, unas variables de tipo *LocalTime* representando respectivamente la hora de la tarea y la de los recordatorios
- **useReminder**, una variable de tipo *boolean* representando el estado de los recordatorios (activados o no)
- **frequency**, una variable de tipo *int* y **frequencyUnit**, una variable de tipo byte, representando la frecuencia de los recordatorios

Los tipos de datos *LocalDate* y *LocalTime* provienen de una librería *Java* llamada *JodaTime*. Esta librería proporciona unos métodos mucha más potentes que las librerías estándar de Java para gestionar las fechas, la razón por la cuál se han utilizado en este proyecto.

Gracias a *Eclipse*, se han podido generar automáticamente los *getters* y *setters* para todas estas variables.

Después de diseñar el formulario para crear una nueva tarea, se ha procedido a implementar los métodos de verificación de campos y de guardado en la base de datos en la clase *NewTask.*

Para verificar los campos, se comprueba cada uno de ellos individualmente, y si no es correcto, se notifica el error al usuario y no se guarda la tarea en la base de datos.

```
@OptionsItem
void accept() {
    boolean error = false;

    if (isEmpty(titleInput)) {
        titleInput.setError(getString(R.string.no_title));
        error = true;
        return;
    } else if (!task.isDateSet()) {
        dateButton.setError(getString(R.string.no_date));
        error = true;
        return;
    } else if (!task.isTimeSet()) {
        timeButton.setError(getString(R.string.no_time));
        error = true;
        return;
    }
```

Ilustración 14 - Código de verificación de formulario

Si todo es correcto se procede a guardar la tarea en la base de datos, llamando al método *saveDB().*

Este método se encarga de crear una instancia de la clase *TaskSQliteHelper*, una clase creada para acceder fácilmente a la base de datos conteniendo las tareas.

La clase *TaskSQLiteHelper* extiende la clase *SQLiteOpenHelper*, y sólo se tienen que sobreescribir los métodos *onCreate()* y *onUpgrade().*

Estos métodos son los que se encargan, respectivamente, de crear la base de datos si no existe, o de actualizarla si hace falta (por ejemplo durante una actualización de la aplicación).

Después de instanciar la clase *TaskSQLiteHelper,* se llama al método *getWritableDatabase()*. Este método nos devuelve un objeto del tipo *SQliteDatabase* que nos permite escribir en la base de datos.

Teniendo esta referencia ya se puede guardar la tarea en la base de datos, utilizando la clase *ContentValues* para transmitir los valores a guardar.

```
TaskSQLiteHelper taskHelper = new TaskSQLiteHelper(this, "DBTasks",
        null, 1);
SQLiteDatabase db = taskHelper.getWritableDatabase();

ContentValues newTask = new ContentValues();
newTask.put("title", titleInput.getText().toString().trim());
if (!isEmpty(descriptionInput))
    newTask.put("description", descriptionInput.getText().toString()
            .trim());
else
    newTask.put("description", "");
newTask.put("date", task.getCompleteDate().toString());
if (toggleReminderCheckBox.isChecked()) {
    newTask.put("useReminder", true);
    newTask.put("reminderDate", task.getCompleteReminderDate()
            .toString());
    newTask.put("freqReminder",
            Integer.parseInt((reminderInput.getText().toString())));
    newTask.put("freqReminderUnit",
            reminderSpinner.getSelectedItemPosition());
```

Ilustración 15 - Código de guardado de la tarea en la base de datos

Después de esto, pasamos a la implementación de la primera pestaña de la pantalla principal, la que contiene la lista de tareas pendientes.

Para esto se utilizó la vista *ListView* que, como su nombre lo indica, permite crear una vista de lista, creando cada elemento a partir de una fuente, en este caso la base de datos.

Antes de poder añadir las tareas a la lista, se tiene que crear la clase *TaskCursorAdapter*, que es la que se encarga de asociar los datos de cada tarea a un elemento de la lista.

Esta clase tiene dos métodos: *newView()* y *bindView().*

El método *newView()* se encarga de inicializar el patrón de los elementos de la vista (el *layout*), para luego poder asociar cada tarea a este *layout*, componiendo un elemento de la lista final.

```
<TextView
    android:id="@+id/title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="25sp"/>

<TextView
    android:id="@+id/date"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/title"
    android:layout_alignLeft="@+id/title"
    android:textSize="20sp" />

<TextView
    android:id="@+id/time"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/title"
    android:layout_toRightOf="@+id/date"
    android:paddingLeft="30dip"
    android:textSize="20sp" />
```

Ilustración 16 - Layout para la lista de tareas de la pantalla principal

En el método *bindView()* se asocian los datos de cada tarea al *layout*, leyendo de un cursor que apunta a la base de datos, para formar un elemento más de la lista.

```
@Override
public void bindView(View view, Context context, Cursor cursor) {

    TextView titleView = (TextView) view.findViewById(R.id.title);
    titleView.setText(cursor.getString(cursor.getColumnIndex("title")));

    String dateString = cursor.getString(cursor.getColumnIndex("date"));

    LocalDate date = LocalDate.parse(dateString.substring(0, 10));
    LocalTime time = LocalTime.parse(dateString.substring(11,
            dateString.length() - 1));

    DateTimeFormatter fmt = DateTimeFormat.forPattern("dd/MM/yyyy");
    TextView dateView = (TextView) view.findViewById(R.id.date);
    dateView.setText(date.toString(fmt));

    fmt = DateTimeFormat.forPattern("HH:mm");
    TextView timeView = (TextView) view.findViewById(R.id.time);
    timeView.setText(time.toString(fmt));
}
```

Ilustración 17 - Método bindView() para asociar una tarea a un layout

Cuando se ha asociado cada tarea a un elemento de la lista, llamamos al método *setAdapter()* en *MainActivity*, pasándole cómo parámetro la instancia de *TaskCursorAdapter* que contiene los elementos que se quieren mostrar en la pantalla.

El método *setAdapter()* se encarga de mostrar cada vista de la lista en la vista sobre la que se aplica.

```
new Handler().post(new Runnable() {
    @Override
    public void run() {
        customAdapter = new TaskCursorAdapter(getActivity(),
                taskHelper.getAllData());
        listView.setAdapter(customAdapter);
    }
});
```

Ilustración 18 - Código de inicialización de la vista de Tarea

Después de implementar todas estas funciones, comprobamos que estas funcionan correctamente y sin errores. Una vez hecho esto podemos pasar a la siguiente iteración.

Ilustración 19 - Vista de Tarea en un dispositivo real

4.4.2 Segunda iteración

Después de haber implementado con éxito las vistas de Tarea, Crear Nueva Tarea y el guardado en base de datos, en esta segunda iteración adaptaremos la vista de Crear Nueva Tarea para que pueda ser utilizada también en el caso de Modificar Tarea.

También añadiremos un botón que permitirá suprimir la tarea si se desea.

Después de esto implementaremos la vista de calendario de la pantalla principal.

Para empezar, se añade un *OnItemClickListener()* a la vista *listView* de *MainActivity* que se encarga de mostrar la lista de tareas en la pantalla principal.

Este *OnItemClickListener()* se encarga de detectar las pulsaciones en pantalla sobre esta vista, y cuando detecta una, se ejecuta el método asociado al *listener*.

En nuestro caso, al detectarse una pulsación sobre una tarea, creamos un nuevo *Intent* (el objeto que permite llamar a otra Actividad de *Android*), pasándole el identificador de la tarea en la base de datos y un dato de tipo *boolean* para notificar a la nueva actividad que se desea modificar una tarea.

```
listView.setOnItemClickListener(new OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, View view,
            int position, long id) {
        newTask(position);
        Intent intent = new Intent(getActivity(),
                NewTaskActivity_.class);
        intent.putExtra("taskReminder_DBid", id);
        intent.putExtra("taskReminder_modifyTask", true);
        startActivity(intent);
    }
});
```

Ilustración 20 - Código para llamar a Modificar Tarea

Al ejecutarse este trozo de código, se llama a la actividad de Crear Nueva Tarea, pero notificándole que se tiene que modificar una tarea, no crear una nueva.

Para soportar este cambio se ha tenido que modificar la clase *NewTaskActivity*.

Lo primero que se ha hecho es modificar el método *onCreate()* de esta clase. El método *onCreate()* es el primero que se ejecuta cuando se muestra la vista, por lo tanto es el candidato perfecto para efectuar las modificaciones.

Para poder detectar que se desea modificar la tarea y no crear una nueva, llamamos al método *getBooleanExtra()* sobre el *Intent* que ha originado la ejecución de la vista actual.

Si el *boolean* devuelto tiene el valor *true*, se llama al método *setButtons()*, que se encarga de rellenar los distintos valores del formulario con los valores leídos en la base de datos.

A continuación se modifica el título de la vista para que el usuario sepa que se está modificando una tarea existente.

```
boolean modifyTask = getIntent().getBooleanExtra(
        "taskReminder_modifyTask", false);
if (modifyTask) {
    setButtons();
    this.setTitle(R.string.modify_task);
}
```

Ilustración 21 - Código para detectar que se quiere modificar tarea

También se ha tenido que modificar el método *saveDB()* para que, cuando se modifique una tarea, no se introduzca una nueva en la base de datos.

Para eso, llamamos al método *getLongExtra()* sobre el mismo *Intent* que anteriormente, para obtener el identificador de la tarea en la base de datos. Si el método devuelve -1, es que es una nueva tarea y por lo tanto se debe de insertar en una nueva fila. Si devuelve cualquier otro valor, significa que se modifica una tarea, por lo que se actualizar la fila cuyo identificador corresponde con este valor, con los datos actualizados del formulario.

```
if (id != -1) {
    newTask.put("_id", id);
    db.update("DBTasks", newTask, "_id=" + id, null);
} else {
    id = db.insert("DBTasks", null, newTask);
```

Ilustración 22 - Código de guardado de la tarea en base de datos

Esto hecho, pasamos a añadir el botón para añadir la posibilidad de suprimir la tarea.

Para conseguirlo, añadimos el botón correspondiente al *layout* de la vista de Crear Nueva Tarea, pero en modo invisible por defecto, ya que si se crea una nueva tarea, no se debe de poder borrar.

Cuando se desee modificar la tarea, al llamar el método *setButtons()*, cambiaremos el estado del botón a visible para que el usuario la pueda suprimir.

```
deleteButton.setVisibility(View.VISIBLE);
```

Ilustración 23 - Código para cambiar el estado del botón suprimir tarea a visible

A continuación creamos el método *deleteTask()* que se encarga de borrar la tarea seleccionada de la base de datos.

Para esto se obtiene el identificador de la tarea, y se ejecuta una simple sentencia SQL *DELETE*, especificando el identificador.

```
db.execSQL("DELETE FROM DBTasks WHERE _id=" + task.getID());
```

Ilustración 24 - Sentencia SQL para suprimir tarea

Hecho esto, podemos pasar a la implementación de la vista de calendario.

Para mostrar el calendario en la pantalla principal, se ha creado un nuevo *layout,* y a ese layout se le han añadido los elementos necesarios: una vista de calendario y una *ListView* para ver las tareas.

Para la vista de calendario se ha utilizado una librería externa llamada *TimesSquare*, que nos proporciona un calendario simple y elegante.

Al pulsar alguno de los días del calendario, la *ListView* se actualiza para mostrar las tareas que tienen como fecha de fin ese día.

Después de esto, comprobamos que todo funciona correctamente y sin errores. Y efectivamente, se actualizan correctamente las tareas, y se pueden suprimir también, por lo que se puede pasar a la siguiente iteración.

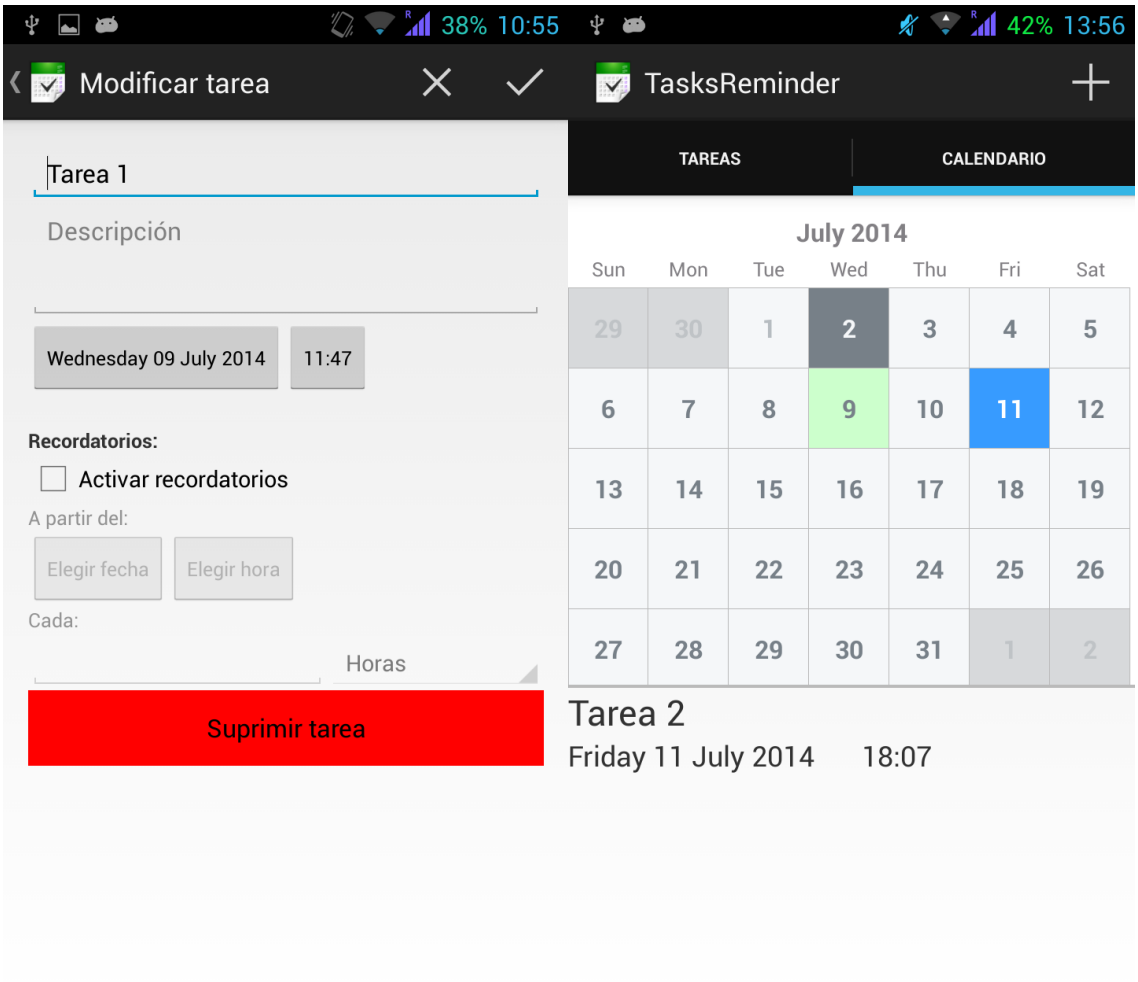

Ilustración 25 - Vista para modificar tarea con el botón Suprimir, y vista de calendario

4.4.3 Tercera iteración

En esta tercera y última iteración se han implementado las alarmas y las notificaciones asociadas a estas alarmas.

Para eso se han creado tres clases:

- *AlarmManagerHelper*
- *AlarmReceiver*
- *AlarmService*

La clase *AlarmManagerHelper* es la que se encarga de definir y cancelar las alarmas, gracias a los métodos *cancelAlarm()* y *setAlarm().*

El método *cancelAlarm()* es muy simple, recibe el *Intent* utilizado para crear la alarma, y utilizando una instancia de *AlarmManager*, el servicio de alarmas de *Android*, cancela esta alarma.

```
public static void cancelAlarm(PendingIntent intent, Context context) {
    AlarmManager am = (AlarmManager) context
            .getSystemService(Context.ALARM_SERVICE);
    am.cancel(intent);
}
```

Ilustración 26 - Código para cancelar una alarma

El método *setAlarm()* tiene más complejidad, ya que además de definir las alarmas tiene que tener en cuenta los recordatorios.

Para no definir demasiadas alarmas en el sistema, se ha decidido sólo crear la próxima alarma para cada tarea, y cuando se dispare la alarma, definir la siguiente.

Para esto, el método recibe el objeto correspondiente a la tarea cuya alarma se quiere definir. Si esta tarea no tiene recordatorios, se crea una alarma para el día de fin de la tarea, sin ningún problema.

Si tiene recordatorios, se comprueba que la fecha del próxima recordatorio sea anterior a la fecha de fin de la tarea.

Si lo es, se crea la alarma con fecha y hora del próximo recordatorio, y se cambia a continuación la fecha del recordatorio a la próxima fecha.

Si la fecha del próximo recordatorio es posterior a la de fin de tarea, se desactivan los recordatorios en la tarea (llamando a *setUseReminder()*, con el valor *false*) y se crea la alarma con la fecha y hora de fin de la tarea.

En los dos casos, antes de crear la alarma, se anula la alarma anterior, contemplando el caso en el que se modifica una tarea, para que no suene una alarma correspondiente a un estado anterior de la tarea.

```
public static void setAlarm(long id, Task task, PendingIntent intent,
        Context context) {
    AlarmManager am = (AlarmManager) context
            .getSystemService(Context.ALARM_SERVICE);
    am.cancel(intent);
    if (task.isDateReminderSet()) {
        am.set(AlarmManager.RTC_WAKEUP, task.getCompleteReminderDate()
                .toDate().getTime(), intent);
        TaskSQLiteHelper taskHelper = new TaskSQLiteHelper(context,
                "DBTasks", null, 1);
        LocalDateTime nextAlarm = task.setNextAlarm();
        SQLiteDatabase db = taskHelper.getWritableDatabase();
        if (nextAlarm != null) {
            db.execSQL("UPDATE DBTasks " + "SET reminderDate='"
                    + task.getCompleteReminderDate().toString() + "'"
                    + "WHERE _id='" + task.getID() + "'");
        } else {
            db.execSQL("UPDATE DBTasks " + "SET useReminder='false'"
                    + "WHERE _id='" + task.getID() + "'");
        }
    } else {
        am.set(AlarmManager.RTC_WAKEUP, task.getCompleteDate().toDate()
                .getTime(), intent);
    }
}
```

Ilustración 27 - Código del método setAlarm()

La clase *AlarmReceiver* es la que recibirá la alarma cuando se dispare. Esta clase extiende la clase *BroadcastReceiver* para poder recibir el aviso de la alarma, gracias al método sobreescrito *onReceive().*

Este método, al recibir la alarma, obtendrá el identificador de la tarea correspondiente, gracias a la información contenida en el *Intent* que ha originado esta alarma, y llamará al método *setNextAlarm(),* que se encargará de llamar al método *setAlarm()* de la clase *AlarmManagerHelper*, para que se cree la próxima alarma.

Después de obtener el identificador de la tarea, se crea una notificación, con el título y la descripción de la tarea, y se muestra en la barra de notificaciones.

También se añade un diálogo que se abrirá al pulsar en esta notificación. Este diálogo mostrará el título, descripción, fecha y hora final de la tarea. También tendrá un botón que permitirá suprimir la tarea si el usuario lo desea.

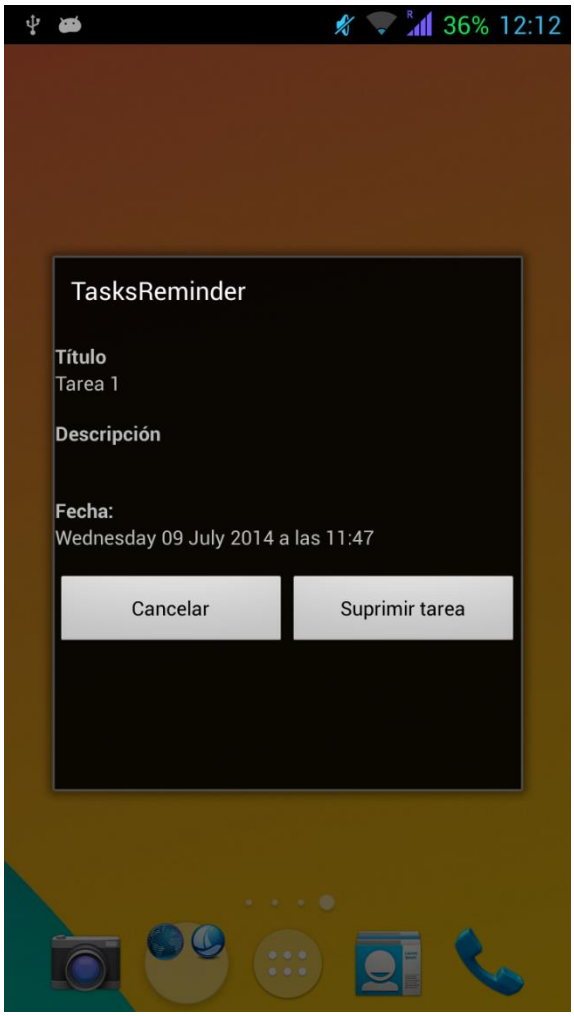

Ilustración 28 - Diálogo asociado a una notificación de alarma

Desafortunadamente estas alarmas no son guardadas cuando se apaga el teléfono. Por lo tanto se ha creado la clase *AlarmService*. Esta clase, como su nombre lo indica, es un servicio cuya única funcionalidad es crear las alarmas que hagan falta cuando el teléfono se enciende.

El método *onStartCommand()* se ejecuta cuando el teléfono se acaba de encender y llama al método *setAlarms()* de la clase *AlarmManagerHelper*.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    AlarmManagerHelper.setAlarms(this);
    return super.onStartCommand(intent, flags, startId);
}
```

Ilustración 29 - Método para crear las alarmas al encender el teléfono

Después de hacer pruebas y comprobar que las alarmas y los recordatorios funcionan correctamente, incluso sin haber ejecutado la aplicación previamente, podemos dar esta tercer y última iteración como acabada.

5 Manual de instalación y de usuario

La aplicación está disponible en la siguiente dirección:

https://play.google.com/store/apps/details?id=com.sampa.tasksreminder

Después de instalarla e abrirla, el usuario verá la siguiente pantalla:

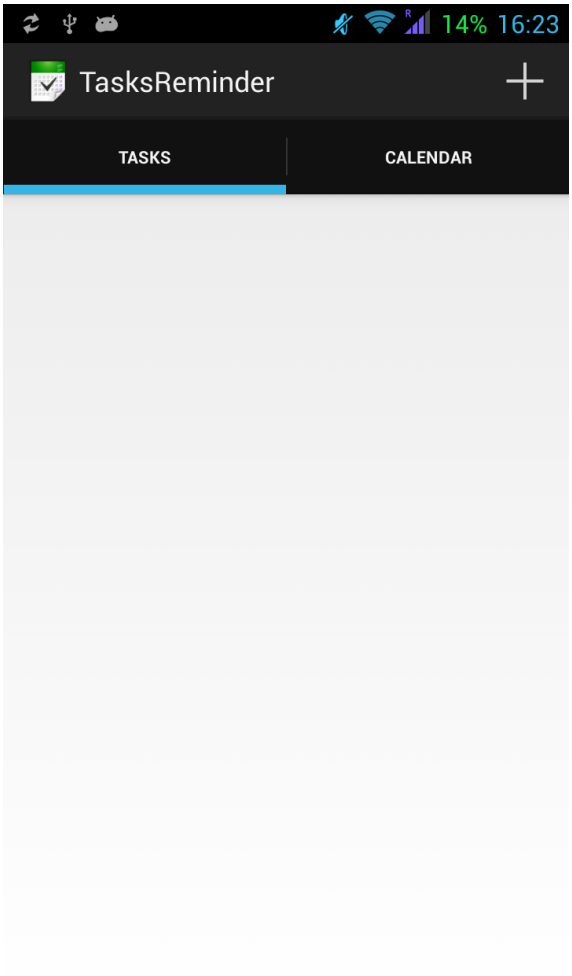

Ilustración 30 - Primer lanzamiento de la aplicación

Si el usuario desliza el dedo de derecha a izquierda sobre esta pantalla, podrá ver la vista de calendario. Para volver a la vista original tendrá que deslizar el dedo en el otro sentido.

Al pulsar el botón en la esquina superior derecha, podrá acceder a la vista de Crear Nueva Tarea.

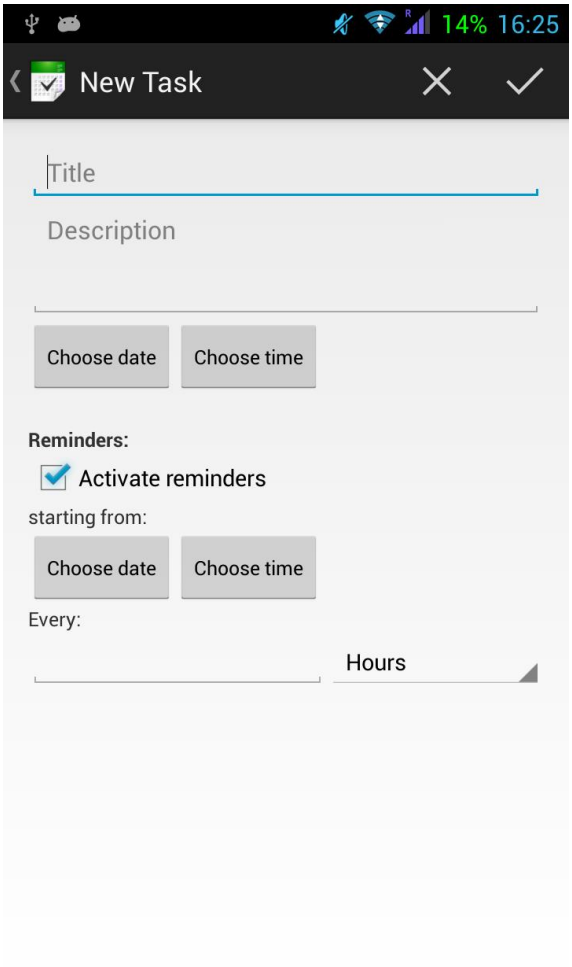

Ilustración 31 - Vista de Crear Nueva Tarea

En esta vista tendrá que rellenar los distintos campos, el título y la fecha y hora de la tarea siendo obligatorios. Para rellenar la fecha u hora, tendrá que pulsar los botones correspondientes, respectivamente "Elegir fecha" y "Elegir hora".

Al pulsar "Elegir fecha", se abrirá una vista de calendario. Para elegir la fecha deseada, el usuario tendrá que pulsar esta fecha y aceptar su elección pulsando el botón de aceptar en la esquina superior derecha. Esto hecho, volverá a la vista de Crear Nueva Tarea.

Al pulsar “Elegir hora”, se abrirá un diálogo donde el usuario podrá elegir la hora deslizando las ruedas de horas y minutos hacia arriba o hacia abajo según necesite. Al pulsar el botón de “Aceptar”, volverá a la vista de Crear Nueva Tarea.

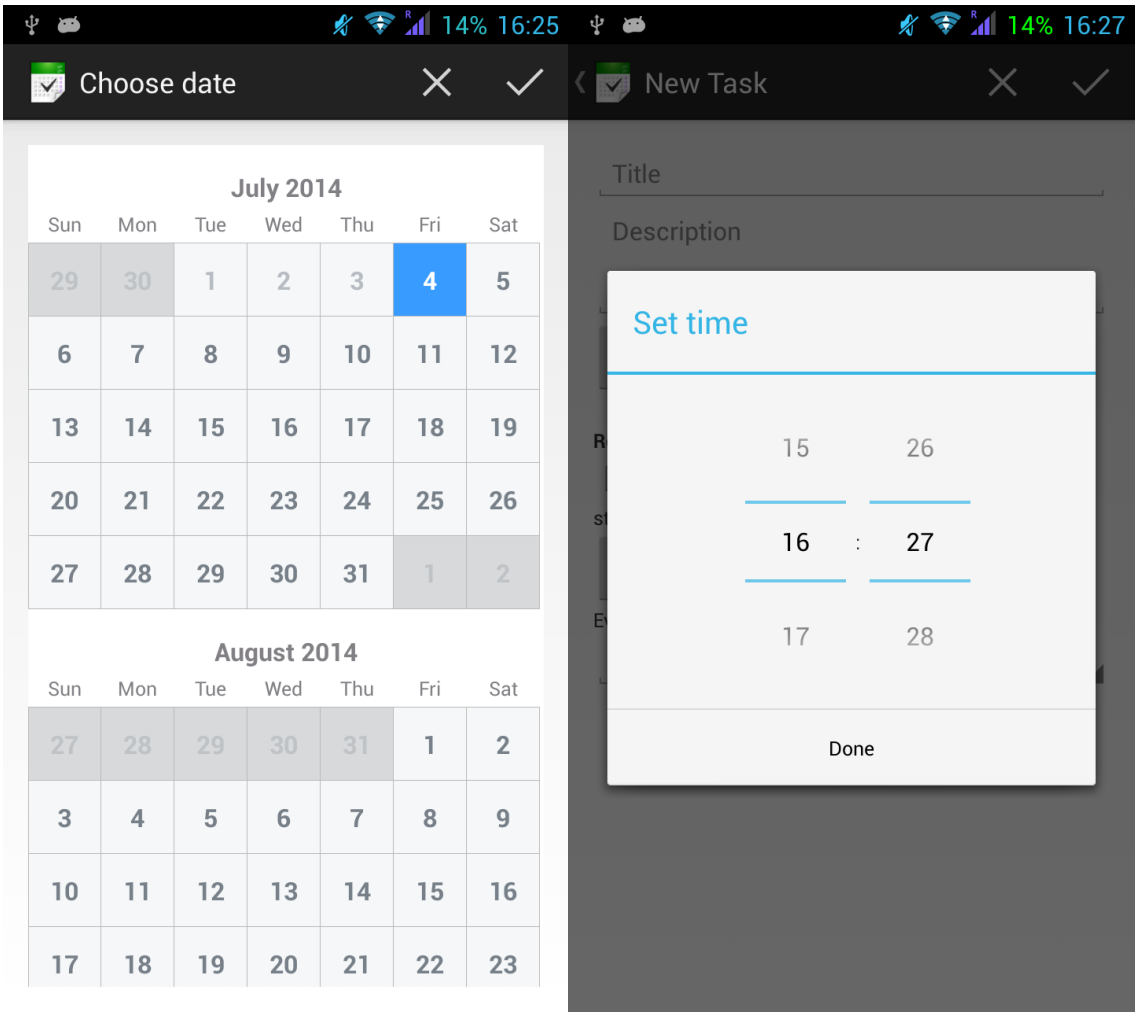

Ilustración 32 - Vistas de Elegir Fecha y Elegir hora

Si el usuario desea especificar recordatorios, tendrá que activar la casilla “Activar recordatorios”. Al hacerlo se activarán los campos correspondientes, que podrá rellenar. Para elegir la fecha u hora, el usuario deberá seguir el mismo procedimiento que anteriormente. La fecha y hora elegidas corresponderán al momento en el que el primer recordatorio notifique al usuario. A continuación el usuario debe de elegir la frecuencia

de estos recordatorios, introduciendo un número en el campo correspondiente y eligiendo la unidad de frecuencia en la lista despegable junto a este campo.

Cuando el usuario haya acabado de rellenar los campos, podrá pulsar en el botón de aceptar, en la esquina superior derecha. Si los campos están correctamente rellenados, podrá volver a la pantalla principal con su tarea creada. Si no lo están, se le notificarán al usuario los campos por completar.

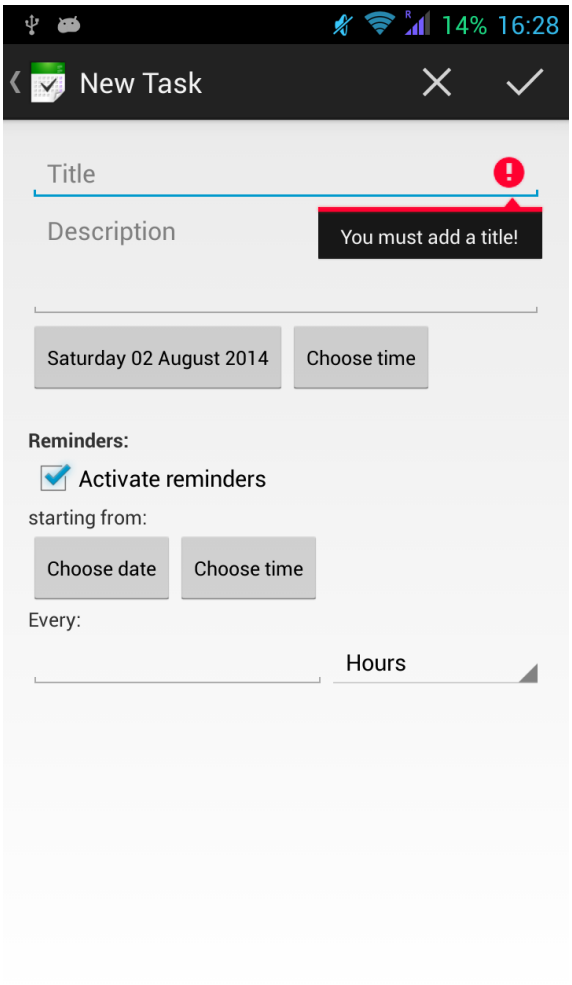

Ilustración 33 - Mensaje de error en vista Crear Nueva Tarea

Si el usuario pulsa en una de las tareas de la pantalla principal, podrá modificarla o suprimirla. Se abrirá una nueva vista, idéntica a la de Crear Nueva Tarea, pero con los campos existentes en la tarea rellenados, un botón de suprimir y el título de la vista cambiado adecuadamente. El funcionamiento de esta vista es el mismo que la anterior,

pudiendo guardar los cambios al pulsar el botón de aceptar en la esquina superior derecha.

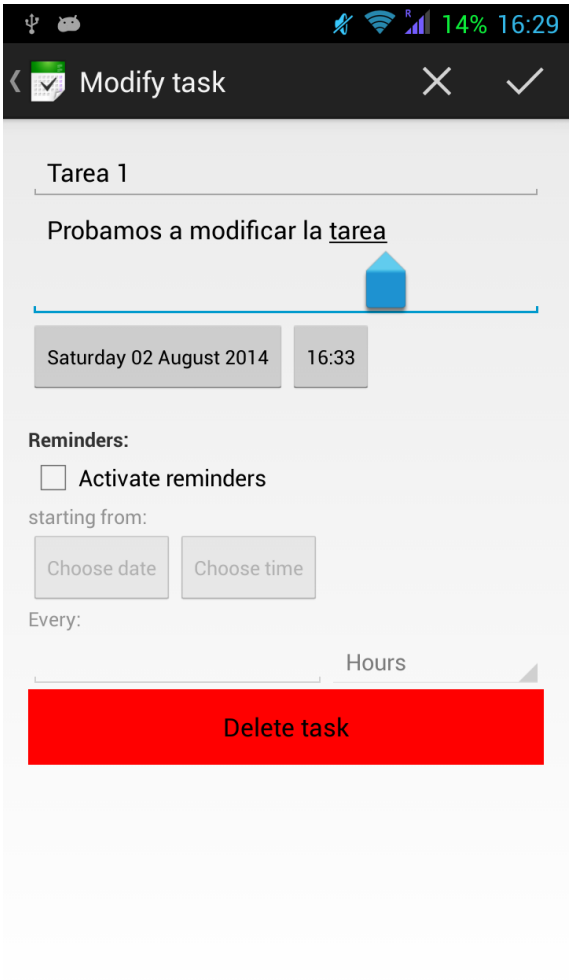

Ilustración 34 - Vista de Modificar Tarea

En la vista de calendario, el usuario podrá ver las fechas en las que tiene tareas definidas, resaltadas en verde. Cuando pulse en esas fechas, podrá consultar esas tareas en forma de lista.

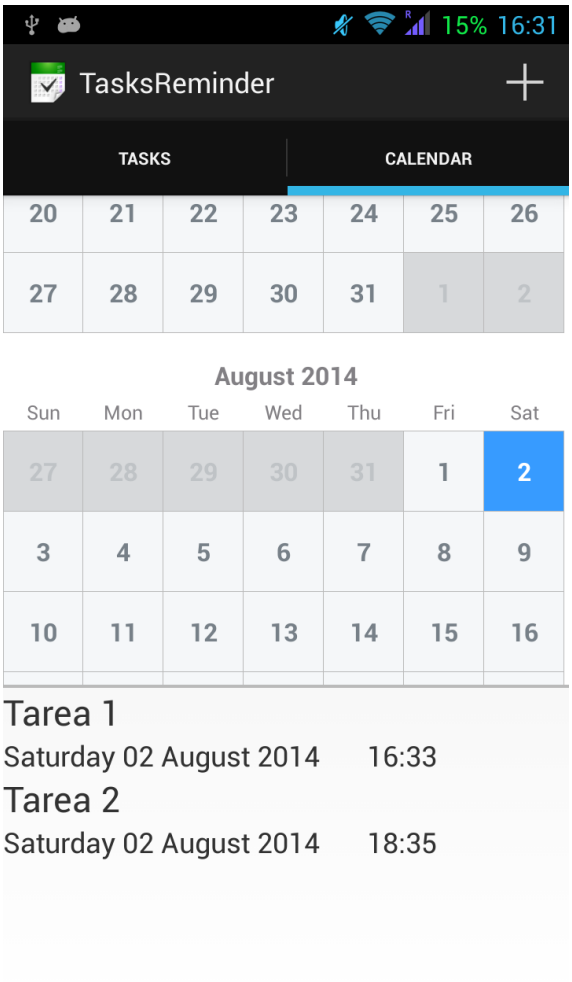

Ilustración 35 - Vista de calendario

Cuando se le notifique al usuario un recordatorio o una alarma, este podrá pulsar en la notificación pendiente, consultar la tarea y suprimirla si lo desea.

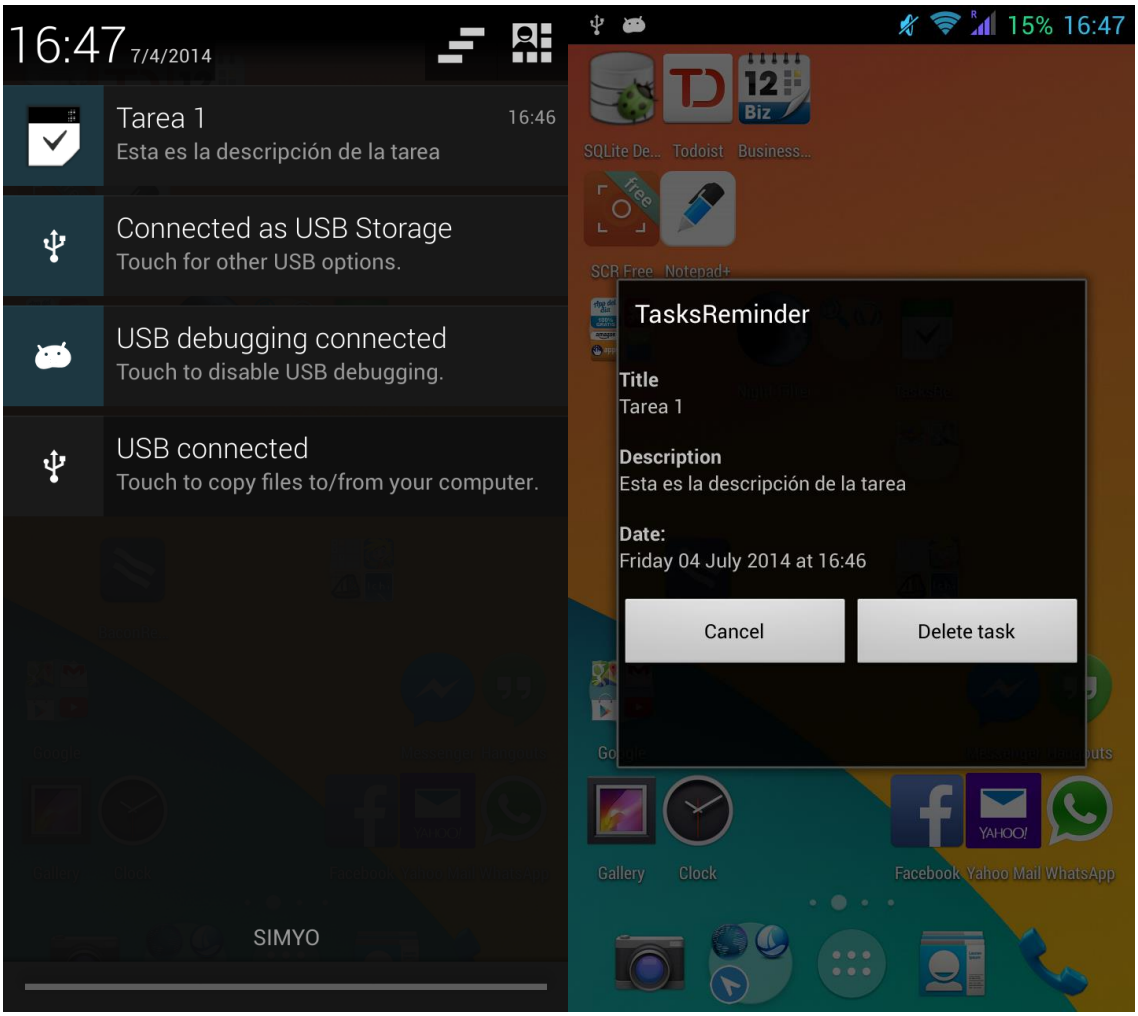

Ilustración 36 - Notificación y diálogo de notificación

6 Conclusiones

Este proyecto me ha permitido aprender mucho sobre la programación en *Android*. Al empezarlo, tenía pocos conocimientos sobre la plataforma, adquiridos durante un curso *online* sobre programación en *Android.*

El principio fue largo y duro, ya que esa falta de experiencia me hizo cometer errores de programación debidos al desconocimiento de la plataforma.

Pero poco a poco fui arreglando esos errores, aprendiendo trucos y formas de programar correctamente para Android.

El resultado es una aplicación relativamente simple, pero que me ha permitido aprender mucho sobre la plataforma y sobre mí mismo, y me ha dado ganas de crear más aplicaciones para dispositivos móviles.

Si tuviera que volver a hacer la aplicación, cambiaría algunas cosas de su implementación, ya que ahora tengo más experiencia y me he dado cuenta de que algunas cosas podrían haberse hecho mejor.

En conclusión, este TFG ha sido una experiencia positiva, y me ha permitido descubrir y apreciar más la plataforma *Android* del lado de un desarrollador de aplicaciones.

7 Bibliografía

Beck K. et al. (2001). *Manifiesto por el Desarrollo Ágil de Software*. Obtenido de Agile Manifesto: http://www.agilemanifesto.org/iso/es/manifesto.html

Cataldi Z. et al. (20 de Junio de 2003). *Ingenieria De Software Educativo.* Obtenido de Instituto de investigación y desarrollo en Informática avanzada: http://www.iidia.com.ar/rgm/comunicaciones/c-icie99-ingenieriasoftwareeducativo.pdf

Llamas R., Reith R. y Shirer M. (12 de Febrero de 2014). Obtenido de International Data Corporation: http://www.idc.com/getdoc.jsp?containerId=prUS24676414

McCracken D. y Jackson A. (1982). Lifecycle concepts considered harmful. En *Sigsoft Software Engineering Notes vol 7 num 2* (págs. 29-32). New York, USA: ACM.