ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

APLICACIÓN MÓVIL-CONTROL DIABETES

MOBILE APP-CONTROL DIABETES

Realizado por
Jesús Palomo Velasco
Tutorizado por
José Antonio Onieva González
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: En este proyecto se ha desarrollado una aplicación móvil desarrollada para la plataforma Android y pensada principalmente para la monitorización de las personas que padecen diabetes. En ella se informará de las actividades físicas realizadas mediante el uso de la API de Google Fit, además del tiempo de actividad, una estimación de las calorias consumidas, y una observación/consejo de tipo nutricional en función de dichas calorias. El usuario dispondrá de una sección de configuración donde podrá establecer un límite calórico, y datos de contacto que permitan realizer alertas automáticas (lamadas, sms, emails) en caso de existir peligro de hipoglucemia.
En la realización del proyecto se ha llevado a cabo una fase de extracción de requisitos de la aplicación así como un diseño de misma. En la implementación se ha seguido una metodología de desarrollo SCRUM y en el interfaz se ha seguido el principio de diseño de simplicidad y funcionalidad. Por ultimo, se han realizado las unidades de tests necesarias para comprobar, mínimamente, la funcionalidad de la app.

Palabras claves: Android, GoogleFit, diabetes

Abstract: In this project we have developed an Android with the aim of monitoring users who suffer diabetes. The app reports the completed sports activities using the Google Fit API, activity duration, nutritional suggestions and estimated calories consumed. The activities, durations and calories must register in the app. The user has a settings menu where he can save a limit of calories and contact details so as to produce automatic alerts when needed using GSM calls, SMS and emails. During the project, requisites have been extracted and the app designed. Implementation has followed a SCRUM iterative methodology and the user interface has been designed under the usability and simplicity principle. The different units have been tested in order to check app functionality.

Keywords: Android, GoogleFit, diabetes

Introducción

Hoy en día existe en un gran número de personas que padecen diabetes. La *diabetes miellitus* (nombre real) es el aumento de la concentración de glucosa en sangre debido a que el organismo pierde su capacidad de producir suficiente insulina o de utilizarla con eficacia.

La insulina es una hormona que se fabrica en el páncreas y que permite que la glucosa de los alimentos pase a las células del organismo, en donde se convierte en energía para que funcionen los músculos y los tejidos. Como resultado, una persona con diabetes no absorbe la glucosa adecuadamente, de modo que ésta queda circulando en la sangre (hiperglucemia) y dañando los tejidos con el paso del tiempo. Este deterioro causa complicaciones para la salud potencialmente letales.

Hay dos causas principales de diabetes: congénita (se nace con ella) o ambiental (debido a una mala alimentación). Teniendo en cuenta esto, podemos hablar de 3 tipos diferentes de diabetes: gestacional, tipo I y tipo II.

Gestacional: tiene lugar durante el embarazo surge debido a que el organismo no puede producir ni utilizar la suficiente insulina necesaria para la gestación. Suele desaparecer después del parto pero tanto la madre como el hijo corren mayor riesgo de desarrollar diabetes de tipo II en el futuro.

Tipo I: ésta requiere de la administración de insulina y su razón de aparición es una mala producción de insulina, es decir, ésta no es sintetizada por el páncreas o la insulina que se sintetiza es defectuosa, por esta razón hay que introducirla del exterior. Una persona con este tipo de diabetes puede llevar una vida normal y saludable mediante una combinación de dieta sana, ejercicio, monitorización e inyección diaria de insulina. Ésta es congénita.

Tipo II: no requiere la administración de insulina pero puede ser recetada para controlar la afección y la razón por la que aparece es que los receptores de insulina son defectuosos o también puede ser debido a un agotamiento del páncreas. Este tipo de diabetes es adquirida, es decir, surge como consecuencia del agotamiento del cuerpo (aunque se conocen causantes congénitos). Este tipo de diabetes puede evolucionar a tipo I.

En la actualidad, no existe una cura exacta de la diabetes para ninguno de los tipos explicados anteriormente pero existen diversas investigaciones con varios enfoques que pretenden llegar a una cura. Estos enfoques son trasplantes de páncreas, trasplantes de las células de los islotes del páncreas, elaboración de páncreas artificiales o mediante manipulación genética. [1] [2]

Motivación

Tal y como hemos visto, la diabetes es una enfermedad que está muy presente en la sociedad actual y que aún no tiene cura. Por ello nos planteamos un escenario en concreto que estará centrado en la actividad física, siendo esta la razón por la que nos planteamos la siguiente pregunta: *¿Cómo controlar la salud, en especial los niveles de azúcar, cuando realizo ciertas actividades físicas?*

Con el presente proyecto pretendemos dar una solución a dicha pregunta empleando los medios que tenemos en la actualidad, mediante el uso de los dispositivos móviles. Pretendemos desarrollar una aplicación que sirva como un diario para el usuario donde informaremos de su actividad física y otro tipo de detalles que veremos más adelante.

Esta aplicación se llevará a cabo para el sistema operativo Android por las numerosas ventajas que nos encontramos:

- Android es un sistema operativo muy sencillo de instalar y con un gran potencial, por lo que cada día está presente en más elementos tecnológicos, ya sean teléfonos móviles, portátiles, tabletas o navegadores GPS.
- Android es de código abierto, lo cual tiene la ventaja de poder acceder al código para el uso de sus características en nuestras aplicaciones. Otros sistemas operativos están restringidos a sus fabricantes, con lo que es más difícil depurar las aplicaciones.
- Al ser de código abierto, existe una inmensa comunidad de diseñadores y desarrolladores que lo utilizan nivel global, siendo de esta forma mucho más sencillo compartir soluciones e ideas o resolver dudas con las aplicaciones.
- Android es capaz de gestionar inteligentemente aplicaciones multitarea, dejando en suspensión las que no se utilizan o dando más recursos a las que los necesitan para una correcta gestión de los recursos del dispositivo.

Gracias a la evolución hardware y software, el rango de uso de las aplicaciones móviles se ha ampliado a varios campos, como el payment, el uso de linternas, brújulas, mirroring entre otros muchos más campos. La mezcla de estos usos, ha generado una demanda de aplicaciones que permitan realizar estas acciones de manera sencilla e intuitiva, consiguiendo que los móviles cubran las mismas necesidades que los ordenadores de sobremesa.

El otro aspecto motivacional del proyecto es el relacionado con las actividades físicas, para el cual emplearemos la reciente tecnología proporcionada por Google, en forma de app para usuarios y de API para desarrolladores, denominada Google Fit, con la cual disponemos de un amplio catálogo de funcionalidades relacionadas con todo el tema físico/deportivo desde el tipo de deporte (futbol, baloncesto, etc…) hasta el

número de calorías quemadas. Dicha tecnología es aun joven ya que vio la luz en octubre de 2014. [3]

Objetivos del TFG

El objetivo de este proyecto es el de desarrollar una aplicación para teléfonos móviles, que incorporen el S.O. Android y que permita, de forma sencilla, un autocontrol por parte del usuario de sus actividades físicas y algunos aspectos de su salud para el control de ésta y para facilitar el control de la alimentación.

Los principales objetivos de este proyecto son los siguientes:

- Conocer las principales características del sistema operativo Android para el desarrollo de aplicaciones.
- Estudiar el entorno de desarrollo de Android Studio en el cual se desarrollará la aplicación. Incluye numerosas ayudas, herramientas de programación, un completo emulador y las APIs de todas las clases y paquetes.
- Estudio y uso de la API Google Fit y sus librerías facilitadas por Google a los desarrolladores para el desarrollo de apps relacionadas con las actividades físicas.
- Desarrollo de una aplicación Android centrada en el seguimiento de las actividades físicas realizadas por los usuarios que padecen diabetes. La presente aplicación pretende establecer un control por parte de los usuarios de su actividad física realizada. Para ello, los mismos usuarios deberán agregar información asociada a la actividad realizada.
- Informar al usuario, a modo de recomendación, de una serie de alimentos para la ingesta de los mismos por su parte, en función de las calorías quemadas según la duración, distancia y tipo de ejercicio realizado.
- Informar al usuario, si así lo desea, de una recomendación de comidas para tomar en el día según cuando se realice la actividad o las actividades. Estas comidas se refieren al almuerzo o la cena.
- Proponer al usuario una sección de datos personales donde podrá establecerse el límite de calorías que se desee quemar, y si es así se le avisará, y unos datos de contacto.
- Si el usuario así lo desea, una vez superado el límite, se procederá a avisar al contacto asociado a sus datos mediante un correo electrónico o mediante la interfaz de llamada del dispositivo. El usuario será el que seleccione la manera con la que establecer contacto con la persona asociada.

Tecnologías utilizadas

Sistema Android

Fue desarrollado por Android inc., empresa comprada por Google en 2005. Inicialmente el sistema se desarrolló exclusivamente por Google, pero más tarde pasó a formar parte de la Open Handset Alliance, la cual está liderada por Google. El primer teléfono que salió al mercado con este sistema fue el HTC G1 o Dream, lanzado originalmente en EEUU a finales de 2008, venía con la versión Android 1.0 instalada, y rápidamente se convirtió en un éxito de ventas. Actualmente la versión 6.0 denominada Marshmallow poco tiene que ver ya con la versión inicial. [4]

Una de las ventajas de Android es que al pertenecer a Google, empresa que apoya actualmente diversos proyectos de software libre y licencias de código abierto, provoca que cualquier programador comience a trabajar cuando desee sin tener que pagar ninguna licencia para desarrollar aplicaciones o darlas a conocer en el Google Play. Google Play es un repositorio de aplicaciones al que se puede acceder desde cualquier Android, quedando a elección del desarrollador que su aplicación sea gratuita o de pago. [5]

La segunda ventaja, es la cantidad de productos pertenecientes a Google tales como Gmail, Youtube, Google Maps, Google Calendar, etc., que cuentan actualmente con un nivel de aceptación muy alta por parte de los usuarios. Todos los sistemas Android cuentan con un acceso rápido y eficiente a estos productos y, al mismo tiempo, Google proporciona API's para que el programador pueda incluir estos servicios en sus aplicaciones de forma muy sencilla.

Android constituye una pila de software, pensada especialmente para dispositivos móviles y que incluye tanto un sistema operativo, como middleware y diversas aplicaciones de usuario.

El núcleo de Android depende de un Linux versión 2.6 para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, stack de red, y modelo de drivers. El núcleo también actúa como una capa de abstracción entre el hardware y el resto del stack de software denominada como Deliverius.

La mayoría de las aplicaciones para Android se programan en lenguaje Java y son ejecutadas en una máquina virtual, especialmente diseñada para esta plataforma, que ha sido bautizada con el nombre de Dalvik. La máquina virtual Dalvik es una máquina virtual Java pero con una serie de características que la diferencian. Es distribuida como software libre bajo la licencia de Apache. Está basada en registros en vez de pila para aprovechar el rendimiento, optimizada para un escaso uso de la memoria, y diseñada para ejecutar varias instancias de la misma máquina

simultáneamente. La diferencia más destacada es el uso del bytecode el cual primero se transforma a Java y posteriormente al usado por Android (.dex) [6]

Los desarrolladores tenemos a nuestra disposición un entorno de desarrollo, llamado Android Studio, junto a su SDK y la opción de un plugin para el entorno de desarrollo Eclipse, que incluyen todas las API's necesarias para la creación de aplicaciones, así como un emulador integrado para su ejecución. Existe, además, una amplia documentación de respaldo para este SDK. [7]

Arquitectura

Android es un sistema diseñado por capas. Como se ha dicho anteriormente, utiliza el kernel de Linux 2.6 que le da acceso a la parte hardware de los dispositivos, a la par que le permite ser compatible con muchos de los drivers creados para Linux.

A continuación, daremos una visión global por capas de cuál es la arquitectura empleada en Android. Cada una de estas capas, utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos a las capas de nivel superior. Podremos visualizar esta arquitectura en la figura 1.

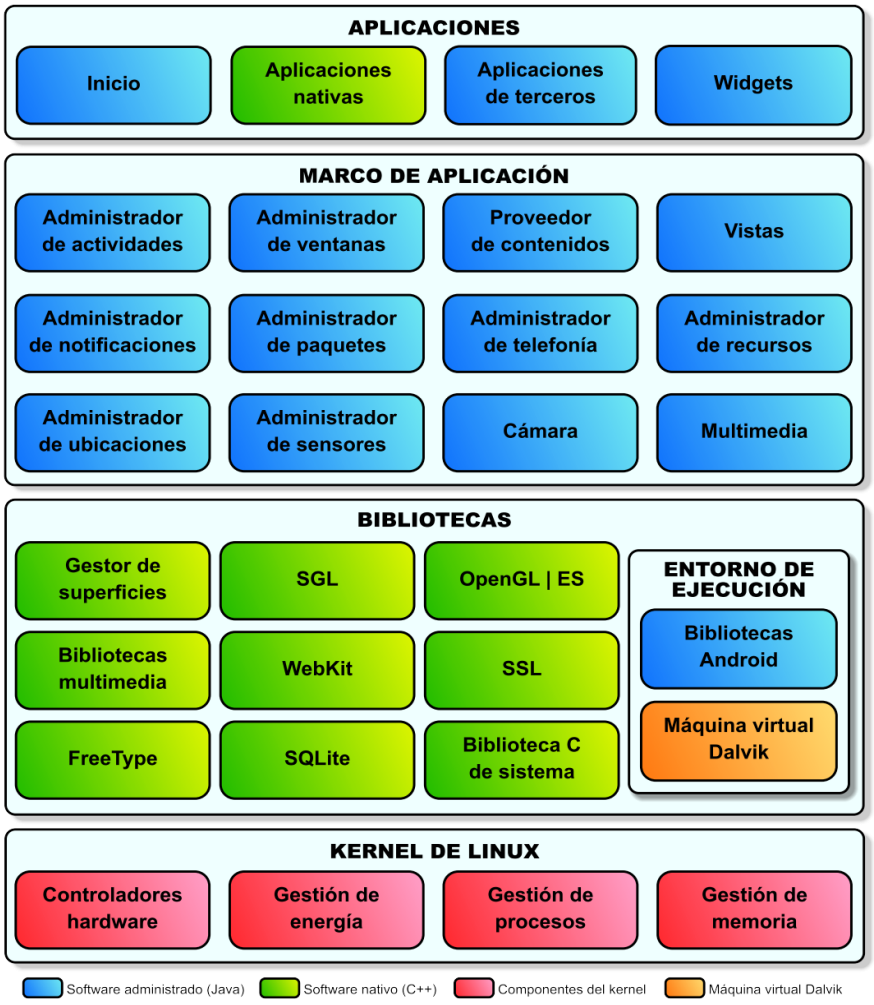


Figura 1.

- Aplicaciones: El nivel superior y en el que el usuario se desenvuelve. Contiene tantos las aplicaciones por defecto de Android como las que el usuario desarrolle o adquiera, gratuitamente o pagando, del repositorio Play Store. Todas estas aplicaciones requieren, para su completa funcionalidad, de los niveles inferiores de la arquitectura.
- Marco de aplicación (framework): Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación, al cual los desarrolladores tienen acceso para la elaboración de sus propias aplicaciones. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede hacer luego uso de esas capacidades (sujeto a reglas de seguridad del framework). Algunos de estos frameworks son: Activity Manager, Windows Manager, Telephone Manager entre otros.
- Bibliotecas: Escritas en C/C++ y usadas por varios componentes del sistema Android. Estas bibliotecas se exponen a los desarrolladores a través del framework de aplicaciones de Android. Algunas son: System C library (implementación librería C standard), librerías de medios, librerías de gráficos, 3d, SQLite, entre otras. Junto al núcleo, basado en Linux, estas librerías constituyen el corazón de Android.
- Entorno de ejecución: situado al mismo nivel que la biblioteca, es un conjunto de librerías base que proveen la mayor parte de las funcionalidades disponibles en el lenguaje de programación Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalkiv ha sido escrito de forma que un dispositivo puede correr en múltiples máquinas virtuales de forma eficiente.
- Kernel de Linux: Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android, es crear las librerias de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

Características de la versión empleada.

Para el desarrollo de nuestra aplicación emplearemos la versión Android 4.4.2, denominada como Kitkat. Esta versión salió a la luz en noviembre del año 2013 y ha sido elegida, para la realización del presente proyecto, debido a que uno de los dispositivos físicos empleados para el testeo es la que tiene instalada. Sus principales características respecto a su sucesora, Jelly Bean o versión 4.3, son las siguientes:

1. Barras de navegación y estado totalmente remodeladas.

2. Nuevo modo de pantalla completa y compatibilidad con caracteres a todo color.
3. Mejora de la compatibilidad con los subtítulos y del sistema de seguridad.
4. Uso mejorado de la batería.
5. Nuevas prestaciones para el desarrollo de aplicaciones.
6. Mejorada la aplicación del teléfono.
7. Añadido a la aplicación Galería nuevas opciones de edición de fotos.
8. Posibilidad de impresión de documentos a través de una red Wifi, Bluetooth y servicios alojados en Google.
9. Nueva versión de Google Hangouts. [8]

Componentes de una aplicación Android.

Todas las aplicaciones en Android, pueden descomponerse en cuatro tipos de bloques/componentes principales. Cada aplicación es una combinación de uno o más de estos componentes, que deberán ser declarados de forma explícita en un fichero con formato XML denominado "AndroidManifest.xml", junto a otros datos asociados como valores globales, clases que implementa, datos que puede manejar, permisos, etc... Este fichero es básico en cualquier aplicación en Android y permite al sistema desplegarla y ejecutarla correctamente. Estos bloques son:

- Activity: Es el encargado de construir la interfaz del usuario, presentar los elementos visuales y reaccionar ante los eventos de usuario. Para mostrar esta interfaz se hace el uso de vistas (*Views*), que se pueden comunicar entre sí mediante *Intents* donde se puede almacenar cualquier tipo de información. Cuando está activa una vista la predecesora permanece en un estado de pausa para volver a ella fácilmente mediante, por ejemplo, la tecla de "atrás" del dispositivo.
- Intents: Un Intent es un objeto mensaje que describe qué quiere hacer una aplicación. Las dos partes más importantes de un Intent son la acción que se quiere realizar y la información necesaria que se proporciona para poder realizarla (expresada en formato URI). Se encarga de notificar a las aplicaciones de varios eventos, como el cambio de estado de hardware o eventos de las aplicaciones entre otros. Así podemos crear actividades que respondan a los intents, también podemos crear intents que lancen actividades o usarlas para detonar eventos ante algunas situaciones específicas. Es importante mencionar, que el sistema es el que elige entre las actividades disponibles en el teléfono y dará respuesta al intent con la actividad más adecuada.
- Content Provider: Esta clase tiene unos métodos estándar que hace que las aplicaciones puedan consultar, guardar, o modificar la información general de las aplicaciones. Existe una serie de content providers ya definidos que se encuentran implementados y que permiten compartir todo tipo de datos,

información de los contactos, imágenes, video, mensajes de texto o incluso audio.

- Services: representa una aplicación ejecutada sin interfaz de usuario, y que, generalmente, tiene lugar en segundo plano mientras otras aplicaciones (éstas con interfaz) son las que están activas en la pantalla del dispositivo. Un ejemplo gráfico sería el caso del reproductor de música, el cual podemos tener activo escuchando música mientras ejecutamos otra aplicación distinta.

Ciclo de vida de una aplicación

Una aplicación Android corre dentro de su propio proceso Linux. Este proceso es creado con la aplicación, y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación. Como ya hemos comentado, una aplicación estará formada por actividades, y el sistema mantendrá una pila con las actividades previamente visualizadas, de forma que el usuario va a poder regresar a la actividad anterior pulsando la tecla "retorno".

Una característica importante, y poco usual, de Android es que la destrucción de un proceso no es controlado directamente por la aplicación. En lugar de esto, es el sistema quien determina cuándo destruir el proceso, basándose en el conocimiento que tiene el sistema de las partes de la aplicación que están corriendo (actividades y servicios), qué tan importante son para el usuario y cuánta memoria disponible hay en un determinado momento. Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esta aplicación aunque el programador podrá evitar este caso.

Android es sensible al ciclo de vida de una actividad, por lo que hay que entender los eventos relacionados con el ciclo de vida si se quiere crear aplicaciones estables:

- Activa (Running): La actividad está encima de la pila, es decir, que es visible y tiene el foco.
- Visible (Paused): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
- Parada (Stopped): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, etc.
- Destruida (Destroyed): Cuando la actividad termina al invocarse el método finish(), o es matada por el sistema.

Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación se muestra

un esquema (figura 2) que ilustra los métodos que capturan estos eventos y la descripción de dichos métodos.

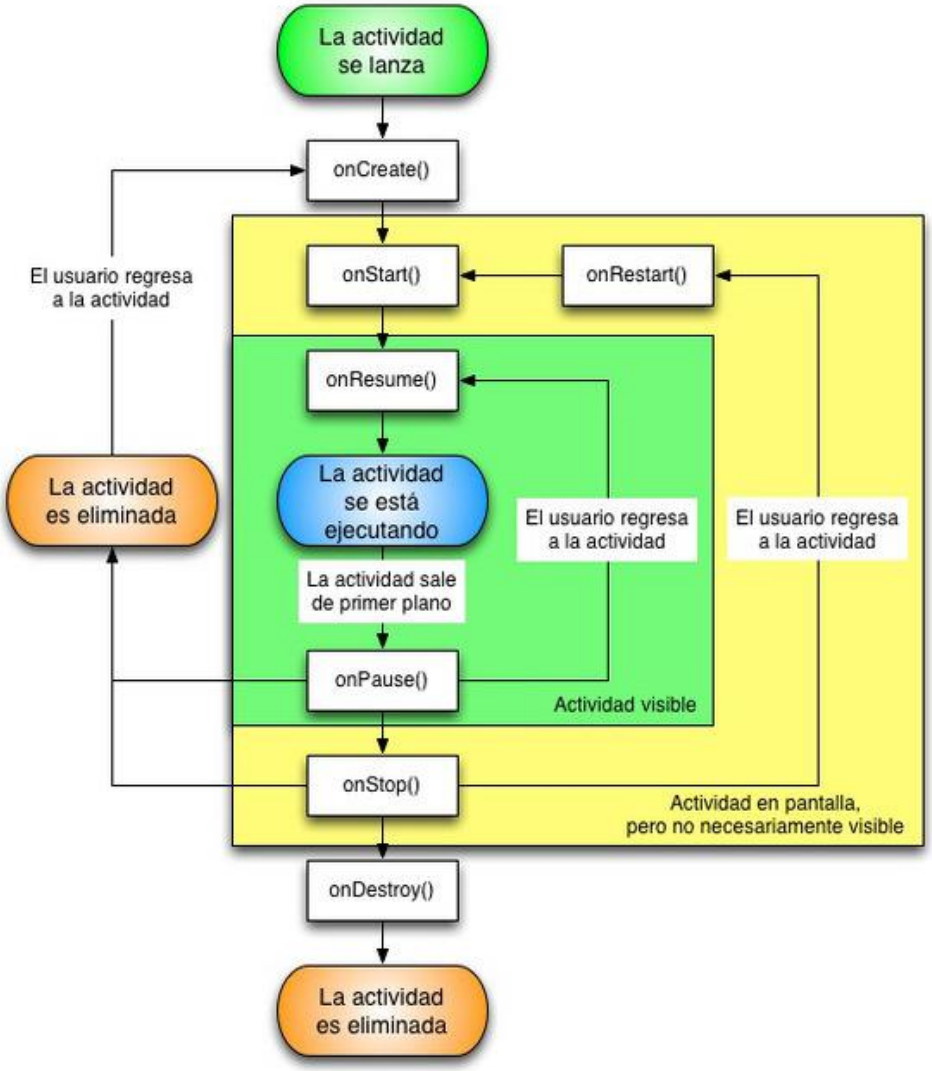


Figura 2.

- **onCreate(Bundle):** Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.
- **onStart():** Nos indica que la actividad está a punto de ser mostrada al usuario.
- **onResume():** Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

- **onPause():** Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.
- **onStop():** La actividad ya no va a ser visible para el usuario. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
- **onRestart():** Indica que la actividad va a volver a ser representada después de haber pasado por onStop().
- **onDestroy():** Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método finish(). Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Una vez visto el esquema y comentado los métodos podemos extraer las siguientes conclusiones:

- *onCreate(), onDestroy():* abarcan todo el ciclo de vida. Cada uno de estos métodos representan el principio y el fin de la actividad.
- *onStart(), onStop():* representan la parte visible del ciclo de vida. Desde onStart() hasta onStop() la actividad será visible para el usuario, aunque es posible que no tenga el foco de acción por existir otras actividades superpuestas con las que el usuario está interactuando. Pueden ser llamados múltiples veces.
- *onResume(),onPause():* delimitan la parte útil del ciclo de vida. Desde onResume() hasta onPause() la actividad no sólo es visible, sino que además tiene el foco de la acción y el usuario puede interactuar con ella.

Como hemos comentado al inicio de la sección, cada aplicación Android se ejecuta dentro de su propio proceso. Este proceso nace por la necesidad de ejecutar parte del código de la aplicación y finaliza cuando lo hace la aplicación o el sistema requiera de sus recursos para otra aplicación más prioritaria.

Los componentes descritos en la sección influyen mucho en el ciclo de vida de la aplicación y su proceso asociado. Es importante saber que el mal uso de estos componentes puede provocar comportamientos incorrectos por parte de la aplicación. Por este motivo el sistema Android construye una jerarquía donde evalúa los componentes que están ejecutándose y el estado de los mismos. Por importancia, los vemos a continuación.

1. *Procesos en primer plano:* como podemos deducir, son los necesarios para lo que el usuario esté realizando en esos instantes. Para categorizarlos como procesos de primer plano deben cumplir una de las siguientes condiciones:
   - Posee un componente Activity con el cual el usuario interactúa.

- Tiene un componente Broadcast Intent Recevier ejecutándose. Son los que detectan y reaccionan a eventos del sistema.
- Existe un componente Service ejecutándose en ese instante.

2. *Procesos visibles:* son los que contienen un proceso Activity en la pantalla pero no en el foco de la actividad en ese momento.
3. *Procesos de servicios:* son aquellos procesos que tienen un componente Service y se están ejecutando en background, que aunque no sean visibles directamente al usuario desempeñan tareas sí percibidas por éste.
4. *Procesos en segundo plano:* procesos con un componente Activity, que no son visibles al usuario. Estos procesos no tienen una importancia directa para el usuario en ese momento.
5. *Procesos vacíos:* son los que ya no ejecutan ninguna actividad, pero se mantienen en memoria para agilizar una posible nueva llamada por parte del usuario.

Además, Android también permite que el usuario manipule los procesos y los termine cuando él desee mediante distintas aplicaciones. Esto es una gran ventaja ya que, como se mencionó anteriormente, Android prioriza los procesos, y si decide que un proceso es importante, lo sigue manteniendo activo; entonces el usuario podría eliminarlo fácilmente para liberar más recursos. [9]

Estructura de una aplicación

Una aplicación Android está dividida en una serie de carpetas y archivos que podemos ver directamente sobre el entorno de Android Studio, aunque éste, por defecto, emplea una vista específica para el entorno de Android pero nosotros cambiaremos dicha vista por la de *"Project"* a la que estamos más habituados en entornos como NetBeans o Eclipse.

El directorio principal de un proyecto Android es el denominado app, del cual vamos a ver sus secciones más destacadas:

- src/main/java: Esta carpeta contendrá todo el código fuente de la aplicación, clases auxiliares, etc. Inicialmente, Android Studio creará por nosotros el código básico de la pantalla (actividad o activity) principal de la aplicación, que recordemos que en nuestro caso era MainActivity, y siempre bajo la estructura del paquete java definido durante la creación del proyecto.
- src/main/res: Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, layouts, cadenas de texto, etc. A continuación, vamos a ver algunos de los diferentes tipos de subdirectorios que nos encontramos:

    - drawable: Contiene las imágenes y otros elementos gráficos usados por la aplicación. Dependiendo del recurso, podremos encontrar otra serie de subdirectorios los cuales no vamos a tratar aquí.
    - mipmap: Contiene los iconos de lanzamiento de la aplicación las distintas densidades de pantalla existentes. Al igual que en el caso anterior, también disponemos de varios subdirectorios.
    - layout: Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica.
    - values: Contiene los valores de los strings que vamos a mostrar por pantalla, esta carpeta es especialmente importante para la realización de aplicaciones multilingües puesto que en vez de rehacer el código, creando varios archivos XML con los diferentes idiomas, podemos cargar la aplicación en un idioma o en otro.

- src/main/AndroidManifest.xml: Contiene la definición en XML de muchos de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, icono…), sus componentes (pantallas, servicios…), o los permisos necesarios para su ejecución.
- build.gradle: Contiene información necesaria para la compilación del proyecto, por ejemplo la versión del SDK de Android utilizada para compilar, la mínima versión de Android que soportará la aplicación, referencias a las librerías externas utilizadas, etc. En un proyecto pueden existir varios ficheros build.gradle, para definir determinados parámetros a distintos niveles.
- libs/: Puede contener las librerías de java externas (ficheros .jar) que utilice nuestra aplicación. Normalmente no incluiremos directamente aquí ninguna librería, sino que haremos referencia a ellas en el fichero build.gradle descrito en el punto anterior, de forma que entren en el proceso de compilación de nuestra aplicación.
- build/: Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que compilamos nuestro proyecto, la máquina de compilación de Android genera por nosotros una serie de ficheros fuente java dirigidos, entre otras muchas cosas, al control de los recursos de la aplicación.

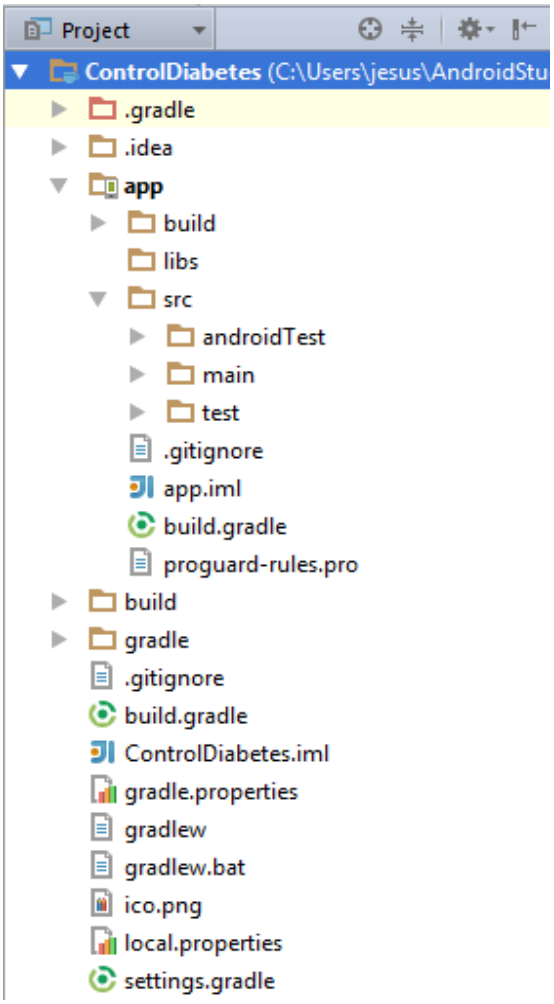


Figura 15. Estructura de ficheros de la aplicación

Seguridad en Android.

Parte de las medidas de seguridad empleadas por Android proceden de su núcleo, el cual está basado en el kernel de Linux 2.6. El mismo proceso en el que se ejecuta la aplicación ya de por sí nos ofrece un entorno seguro ya que aísla la aplicación del sistema y de otra aplicación.

Android utiliza el concepto de entorno de seguridad para aislar los permisos de comunicación entre aplicaciones para permitir o denegar el acceso de una aplicación a los recursos del dispositivo, como archivos y directorios, red, sensores y API en general. Esto obliga al programador a establecer explícitamente los permisos sobre el dispositivo, recursos, etc., en la aplicación a desarrollar.

Además, nos permite firmar las aplicaciones digitalmente mediante un certificado basado en claves públicas y privadas pero no es obligatorio para los desarrolladores, solo en el caso de que sea publicada en la Play Store de Google y para su posterior actualización. Nuestra aplicación contará con los siguientes privilegios:

- Internet: permitir el acceso y conexión a Internet.
- Cuentas: autorización para el uso de las cuentas, y sus propiedades, de Google.
- Localización.

Google Fit

App – Usuarios

Google Fit fue publicado por Google, a través de la Play Store, a finales de octubre del año 2014 para dispositivos Android 4.0 o superior. Esta aplicación está pensada para cuantificar nuestra actividad diaria mediante el uso de los sensores incorporados en nuestro dispositivo Android y un perfil de Google al que asociar los datos recogidos. Podremos visualizar los datos recopilados mediante las diversas gráficas que nos ofrece.

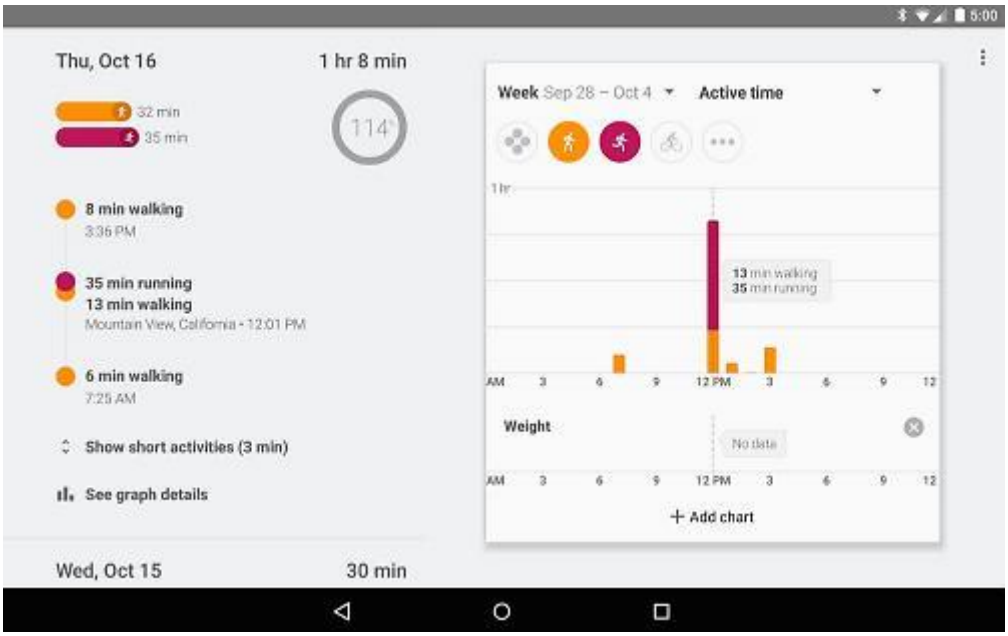


Figura 3. Visualización app

Este perfil, concretamente un perfil de fitness, nos permite hacer un completo seguimiento de todos nuestros ejercicios y datos recogidos por los sensores del dispositivo en tiempo real y ver nuestro progreso con el paso de los días, semanas o meses, ya dependiendo de nuestra configuración. Automáticamente, y según el ritmo, la app intentará categorizar el ejercicio en cuestión en ese mismo instante, hablamos de si estamos andando, corriendo o montando en bici. Nuestro perfil podemos configurarlo con nuestros datos básicos como puede ser la altura, peso o género, pudiendo incluso cambiar el sistema métrico en el caso de la altura y el peso.

Otra funcionalidad de la aplicación es que el usuario pueda añadir sus propias actividades detallándolas a modo de diario deportivo, indicando todos los datos que

deseemos, desde la actividad y tiempo hasta una estimación del número de calorías quemadas, pasando por la distancia recorrida o el número de pasos empleados. En este grupo de actividades nos referimos principalmente a actividades que el dispositivo no puede categorizar por el movimiento como, por ejemplo, artes marciales, fútbol, baloncesto, deportes de raqueta, etc. No obstante, cabe indicar que también se pueden añadir las "actividades automatizadas" de las que hablábamos antes, en casos como el que el dispositivo esté sin batería en el momento de la actividad o se nos haya olvidado.

Para usuarios que usen la aplicación únicamente para las actividades que reconoce automáticamente, a modo de podómetro por ejemplo, dispone de una opción para establecer metas u objetivos simples a diario como puede ser el tiempo de una actividad, realización de un cierto número de pasos, distancia recorrida o calorías quemadas, y nos avisará, en caso de no deshabilitar dicha opción, cuando hayamos conseguido cumplir el objetivo/meta establecido.

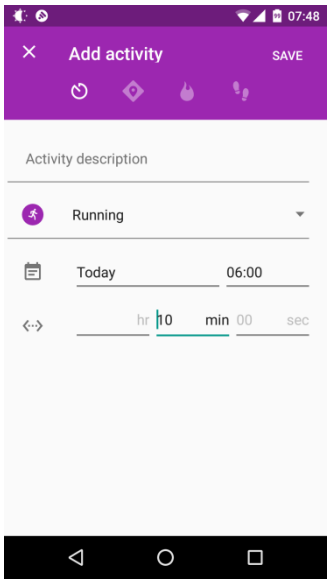


Figura 4. Agregar actividad y objetivo

Por último, y no menos interesante, podremos vincular la aplicación (en el menú de ajustes) a otras aplicaciones/dispositivos para sincronizar los datos como por ejemplo el empleo de un pulsómetro que permita compatibilidad con Google Fit (Android Wear, Nike+ Running, Strava, Runtastic…). [10]

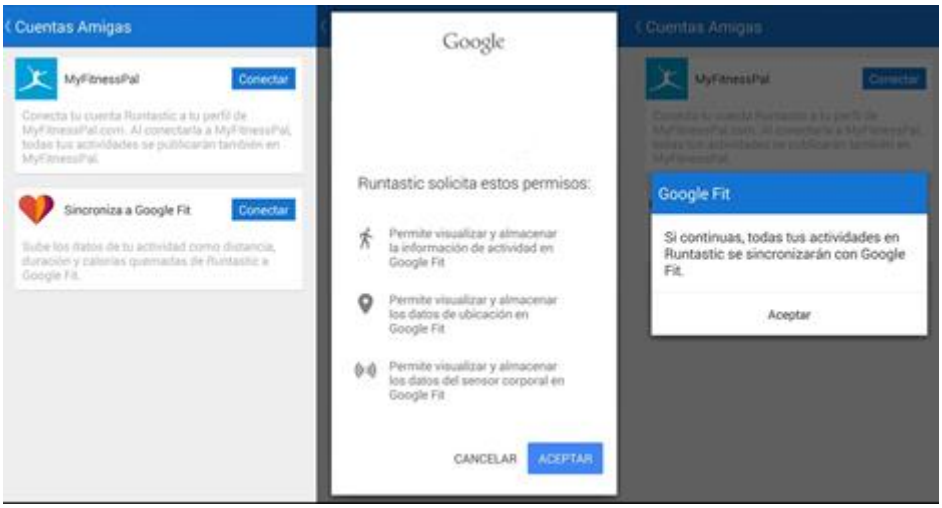


Figura 5. Conectar aplicaciones.

API

Paralelamente a la app para usuarios de google fit, Google publicó una interfaz de programación de aplicaciones (API) para los desarrolladores que permite la subida de datos, *fitness*, a un repositorio central, donde los usuarios pueden acceder a estos desde diferentes dispositivos y aplicaciones. Sus características más destacadas son:

- Las aplicaciones pueden almacenar datos desde cualquier sensor o dispositivo.
- Las aplicaciones pueden acceder a datos creados por cualquier aplicación.
- Los datos de usuarios permanecerán intactos cuando estos actualicen sus dispositivos o sensores.

Existe una responsabilidad por parte del programador a la hora de realizar este tipo de aplicaciones, ya que a menudo se acumula y administra información importante del usuario. Hay que tener en cuenta las siguientes claves:

- Explicar claramente al usuario qué datos se van a recopilar y por qué.
- Honrar la solicitud del usuario de borrar sus datos.
- Si lees datos desde Google Fit, debes escribir tus recopilaciones en Google Fit.
- No usar Google Fit para fines no relacionados con el ejercicio, como fines médicos, biométricos o comercialización.
- Cuidadosamente revisar los términos y condiciones de Google Fit. [11]

Requisitos para el desarrollo de Google Fit

Para el uso, y desarrollo de aplicaciones, de Google Fit se necesita un ID de cliente Oauth de para aplicaciones Android.

Todas las aplicaciones Android están firmadas con un certificado digital el cual contendrá la llave privada. Android Oauth está enlazado para especificar las parejas de certificados. Solo se necesita un ID por cada certificado, no importa cuántos usuarios hagan uso de la app. Para obtener el ID de cliente debemos seguir los siguientes pasos:

1- Localizar el certificado, y su información, de la aplicación.
2- Crear o modificar un proyecto en Google Developer Console.
3- Solicitar un ID de cliente Oauth 2.0. [12]

Componentes de Google Fit

A continuación, vamos a ver los componentes que forman Google Fit:

- Repositorio *fitness*: Repositorio central donde se almacenan los datos recopilados por los dispositivos y sensores. Este repositorio es un servicio en la nube totalmente transparente a los usuarios.
- Marco de aplicación de sensores: Conjunto de representaciones de alto nivel que nos facilita la interacción con el repositorio *fitness*. Podemos usar estas representaciones con la API de Google Fit.
- Permisos y control de usuario: conjunto de autorizaciones solicitadas al usuario para pedir permisos para trabajar con datos *fitness*. Google Fit requiere este consentimiento para acceder a estos datos.
- Google Fit APIs: librerías necesarias para el desarrollo de aplicaciones, tanto nativas en Android como REST (aplicaciones web), para el acceso al repositorio.

Dentro del framework de sensores, como ya hemos comentado, podemos trabajar con distintas representaciones de datos para datos de *fitness*, sensores, sesiones y *data points*. Estos nos facilitan la interacción con el repositorio *fitness*. Vamos a ver con un poco más de detalles estas representaciones:

- *Data Sources:* representan a los sensores y consisten en el nombre, tipo de dato almacenado y otros detalles de los sensores. Puede representar a un sensor hardware o software (definidos en una app).
- *Data Types:* representa los distintos tipos de datos, en el caso de *fitness* como el contador de pasos o el ritmo cardíaco por ejemplo. Consisten en el nombre y una lista ordenada de campos, por ejemplo para la localización (nombre) tenemos latitud, longitud y precisión (campos).
- *Data Points:* consisten en un array temporal de *data types*, leídos de *data source*. Se usan para grabar e insertar datos *fitness* en el repositorio *fitness* y la lectura de datos del *data source*.
- *Datasets:* representan un conjunto de *data points* del mismo tipo de un *data source* en particular en un intervalo de tiempo. Se usan para insertar datos en el repositorio *fitness*. Las consultas al repositorio *fitness* devuelven este tipo de datos.
- *Sessions:* representa un intervalo de tiempo en el cual el usuario realiza alguna actividad como correr, caminar o montar en bici. Nos ayudan a organizar los datos para una actividad concreta y su consulta/inserción en el repositorio *fitness.*[13]

Estructura de Google Fit

En el siguiente esquema vemos la estructura con la que interactúan los dispositivos y sensores con Google Fit mediante la app desarrollada por Android o REST, empezando desde el dispositivo/sensor hasta llegar al repositorio *fitness.* [13]

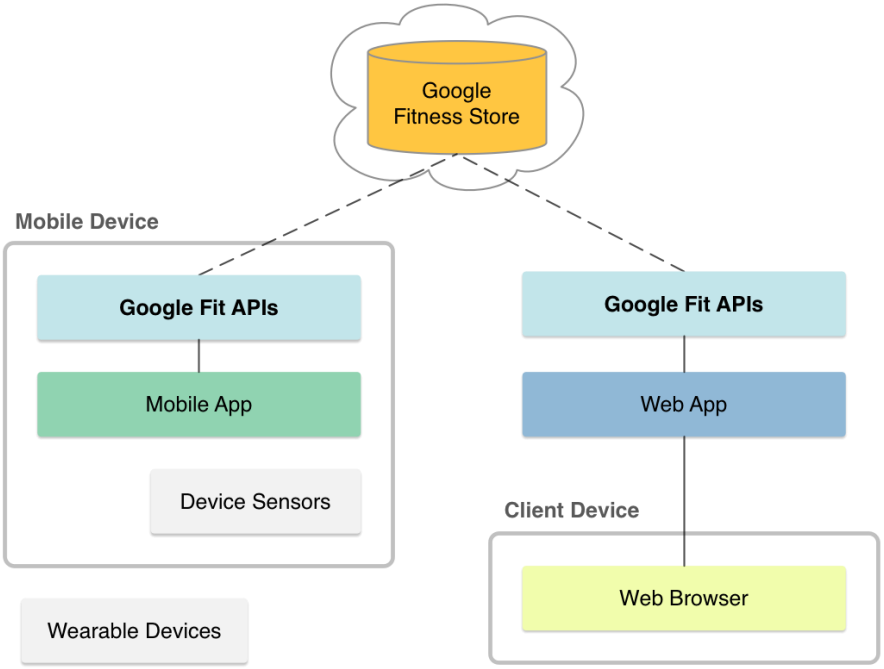


Figura 6. Esquema aplicaciones GoogleFit

A continuación en la siguiente sección nos centraremos en la API nativa para Android ya que es la que vamos a emplear para la realización del proyecto.

API – Android

La API de Google Fit para Android, la cual vamos a emplear en la aplicación, forma parte de los servicios de Google Play y está soportada desde Android 2.3 en adelante. Google Fit para Android está formado por las siguientes APIs:

- Sensors API: provee datos recogidos por los sensores disponibles en los dispositivos Android y por los dispositivos complementarios como por ejemplo las pulseras electrónicas.
- Recording API: provee almacenamiento automatizado de los datos *fitness* para ser usados a lo largo del ciclo de vida de la aplicación.
- History API: proporciona acceso al historial de datos almacenados por Google Fit y permite desempeñar operaciones como inserción, lectura y eliminación de datos *fitness*. Las apps también pueden importar lotes de datos de Google Fit.
- Sessions API: aporta funcionalidad al almacenamiento de datos con metadatos de sesiones. Estas sesiones representan un intervalo de tiempo en el cual los usuarios realizan una actividad.

- Bluetooth Low Energy API: permite el acceso a los sensores de Bluetooth por parte de Google Fit. Esta API activa en tu app la búsqueda de dispositivos Bluetooth disponibles para recoger sus datos y almacenarlos.
- Config API: nos permite la personalización de datos y configuraciones adicionales para Google Fit. [14]

Sensors API nos permite leer los datos recogidos por los sensores en tiempo real en nuestra aplicación. Esta API se usa para listar los Data Sources disponibles en el dispositivo y otros dispositivos asociados, registrar los receptores de los datos enviados por los sensores y parar y borrar estos receptores. Esta API no nos permite almacenar estos datos de una manera automática en el repositorio de Google Fit y los sensores registrados no persisten en caso de reinicio del sistema. Esta API se suele combinar junto con la de Recording API que es la encargada de grabar estos datos. [15]

Recording API permite a nuestra app el almacenamiento de los datos en el repositorio de Google Fit haciendo un uso eficiente de la batería mediante la creación de subscripciones. Una subscripción está asociada a la aplicación Android y consiste en un específico Data Type o Data Source. Podremos entonces crear múltiples subscripciones asociadas a diferentes Data Type o Data Sources. Google Fit almacena los datos *fitness* de nuestras suscripciones activas aun cuando la aplicación no está activa y se restauran estas suscripciones cuando se reinicia. Los datos almacenados están disponibles en el historial del usuario. Para almacenar datos *fitness* mediante sesiones deberemos emplear la API de Sessions o si no podremos almacenar los datos mediante la History API. [16]

Sessions API nos permite representar un intervalo de tiempo en el cual los usuarios realizan una actividad. También permite a la aplicación crear sesiones en el repositorio de Google usando los datos en tiempo real o habiéndolos recolectados previamente mediante Sensors API. En el transcurso de la actividad donde el usuario notifica que empieza y acaba la actividad mediante la aplicación, podemos crear la sesión en tiempo real. Además podemos insertar una sesión en el repositorio una vez la actividad haya finalizado o cuando se está importando datos y sesiones desde fuera de Google Fit. [17]

History API nos da otra manera de almacenar los datos, también permite el empleo de operaciones masivas sobre el repositorio *fitness*: inserción de datos, actualización de datos, eliminación de datos previamente almacenado en Google Fit, lectura de datos insertados o grabados por otra aplicación, importar datos en lote al repositorio. [18]

Google Fit nos ofrece también Bluetooth API la cual nos facilita la conexión de la aplicación con los sensores mediante Bluetooth, lo cual esto será necesario para el correcto uso de Sensors API y Recording API que ya hemos comentado previamente.

Nos permite la conexión y uso de los dispositivos Bluetooth cercanos de una manera eficiente. Dicha API requiere que los dispositivos Bluetooth sean de la versión 4.0, conocida como Bluetooth Low Energy, cuya principal característica es sin duda el poco consumo de energía de la batería del dispositivo, ya que esto ha sido durante años uno de los grandes defectos de la tecnología Bluetooth. [19]

Y por último, la última API que nos ofrece Google es la denominada como Config API. Dicha API nos ofrece las siguientes características: creación de un Data Type personalizado, lo cual emplearemos más adelante, lectura de nuestros Data Type personalizados y deshabilitar Google Fit de nuestra aplicación. Cabe mencionar que los Data Type personalizados que nos ofrece Google serán exclusivos de nuestra aplicación y no se podrán ver desde otra. [20]

Aplicación ControlDiabetes

Introducción ControlDiabetes

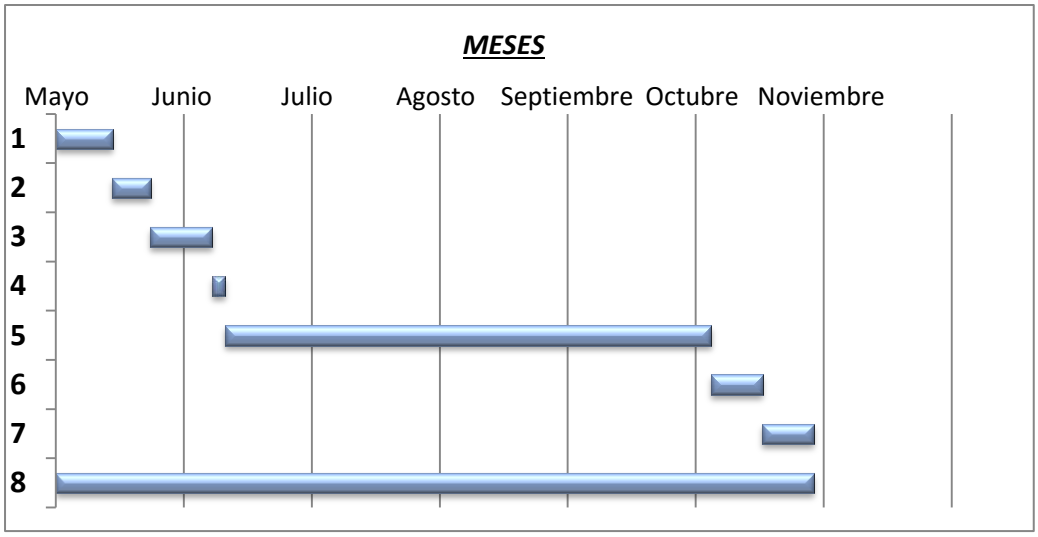
Una vez familiarizados con Android y teniendo configurado el entorno de trabajo, podemos comenzar con el desarrollo de la aplicación.

La aplicación estará basada en la funcionalidad de Google Fit, cuyo objetivo es dar al usuario, que padece diabetes, una aplicación donde controle sus hábitos deportivos y, además, reciba recomendaciones culinarias, tanto alimentos sueltos como comidas, según las actividades realizadas. Proporcionaremos también una sección donde el usuario podrá establecer unos valores de contactos y calóricos usando estos últimos como un umbral que, en caso de superarse, pueda establecer comunicación con alguno de los contactos por riesgos saludables (hablamos de una hiper/hipoglucemia a modo genérico).

Dicho proyecto estará divido en una serie de fases que procedemos a detallar a continuación:

1. Búsqueda y recopilación de información y documentación tanto del problema como de la tecnología a usar.
2. Estudio y análisis del problema a tratar.
3. Estudio y aprendizaje de dicha documentación.
4. Realización de un guion que marque los pasos hacia la solución, en nuestro caso las fases secuenciales de la metodología en cascada que veremos a continuación.
5. Desarrollo de la aplicación cumpliendo todos los requisitos.
6. Evaluación, corrección de errores y puesta en práctica.
7. Inclusión de posibles mejoras en la operatividad de la aplicación.
8. Redacción de la memoria / manual del proyecto.

Estas fases, y siguiendo las normas relacionadas con el TFG, las dividiremos en una serie de horas, a modo de estimación. Dicha división la vemos en el posterior diagrama de Gantt donde la barra vertical indicará el número de la fase de las anteriores especificadas, el contenido de la barra corresponderá a las horas empleadas y la evolución horizontal los días empleados.



Especificación de requisitos

En esta fase se procederá a definir qué tiene que hacer la aplicación y cómo tiene que hacerlo, así como la interacción que tendrá dicha aplicación con el usuario. Para ello se establecerán formalmente una serie de requisitos que deberá cumplir la aplicación, para así saber que estamos cumpliendo con lo acordado, y que la aplicación refleja lo que esperamos de ella.

Los requisitos que definiremos serán los siguientes:

- *Requisitos funcionales:* nos indican lo que hace la aplicación, lo que incluye y su comportamiento cara al usuario.
- *Requisitos no funcionales:* estos no tienen que ver directamente con la funcionalidad y el objetivo de la aplicación. En este caso especifican los criterios o las restricciones del sistema.

Comenzamos hablando de los **requisitos funcionales** que va a precisar nuestra aplicación ControlDiabetes para dispositivos móviles:

1. El usuario podrá ver información relativa a su límite de calorías, dirección de email de contacto y teléfono de contacto.
2. El usuario podrá modificar su límite de calorías, email de contacto y teléfono de contacto. El teléfono deberá ser de 9 dígitos, el correo deberá ser uno sintácticamente válido y el límite de calorías será un número con o sin decimales.
3. En la información relacionada con el usuario podrá establecer el envío automático de SMS o email si así lo desea.
4. El usuario podrá añadir información referente a las actividades físicas realizadas mediante una cuenta de Google a través de Google Fit. Esta información estará formada por el tipo de ejercicio realizado, el tiempo mediante un entero, la

distancia y una estimación de las calorías quemadas que será autocalculada (en función de la actividad) pero con la posibilidad de modificación por parte del usuario. Estos 2 últimos campos serán un número con o sin decimales. El tipo de actividad será elegido entre un listado de actividades, las unidades empleadas para la inserción de datos serán las siguientes:

   a. Distancia: usaremos el metro.
   b. Tiempo: usaremos los minutos
   c. Calorías: usaremos las kilocalorías.

5. La información de la actividad que se va a añadir, distancia, tiempo y calorías, no podrá ser nula (campo en blanco) y tendrá que ser un número mayor que 0.
6. El usuario podrá seleccionar, mediante un calendario en pantalla, el día que desee consultar los datos.
7. El usuario puede consultar la información deportiva, previamente insertada, asociada al mismo y al día de la realización e inserción de esta.
8. El usuario podrá obtener además un consejo nutricional asociado a la actividad realizada en base a las kilocalorías quemadas.
9. De la misma manera que en el caso anterior, el usuario podrá obtener un consejo sobre las comidas a seguir, almuerzo y cena, según cuando se realice la actividad.

Finalizaremos la sección comentando los **requisitos no funcionales**, que serán fundamentales para el correcto comportamiento de la aplicación:

1. Los usuarios que hagan uso de la aplicación deberán tener un dispositivo Android cuya versión sea, como mínimo, la 5.0.1 (Lollipop)
2. Deberán poseer una conexión a Internet en algún momento para poder sincronizar los datos de la aplicación con los servidores de Google. Será indiferente el tipo de conexión que realice el usuario, sea 3G, 4G o mediante redes Wi-Fi.
3. El dispositivo deberá poseer de 14 Megabytes libres para poder instalar la aplicación.
4. El dispositivo deberá disponer de unos 14/15 Megabytes de memoria RAM para su correcto funcionamiento.
5. Cada persona que padezca diabetes, y desee que se le realice el seguimiento de las actividades físicas, deberá tener asociada una cuenta de Google para el acceso y uso de Google Fit.

Diseño – Diagramas

Tras la fase de análisis, se realizan los correspondientes diagramas que permiten diseñar la aplicación. Con ello, se intenta mostrar cómo funcionará la aplicación o cuál es su flujo de datos.

Diagramas de casos de uso

En este apartado, se utilizan los casos de uso como forma de hacer ver al lector las funcionalidades que la aplicación ControlDiabetes va a desempeñar frente al usuario. Sin embargo, no se va a profundizar demasiado en las posibilidades que ofrece esta técnica aunque la definiremos brevemente a continuación, sino que se limitará solamente a informar sobre aquellos aspectos que son más ilustrativos para el lector y que le ayuden a comprender mejor qué es lo que realiza la aplicación.

Los diagramas de casos describen lo que hace un sistema desde el punto de vista de un observador externo sin importar cómo lo hace. Está formado por 4 componentes:

- Caso de uso: describe una interacción coherente y con un objetivo de un actor con un sistema.
- Actor: Es definido como un rol que interactúa con el sistema. Dicho rol lo puede asumir un individuo o un sistema externo. Siempre estará fuera del sistema.
- Límites del sistema: Es un rectángulo que engloba al sistema y los casos de uso y aísla al actor de éste.
- Asociación: establece la comunicación y relación entre los actores y los casos de uso.

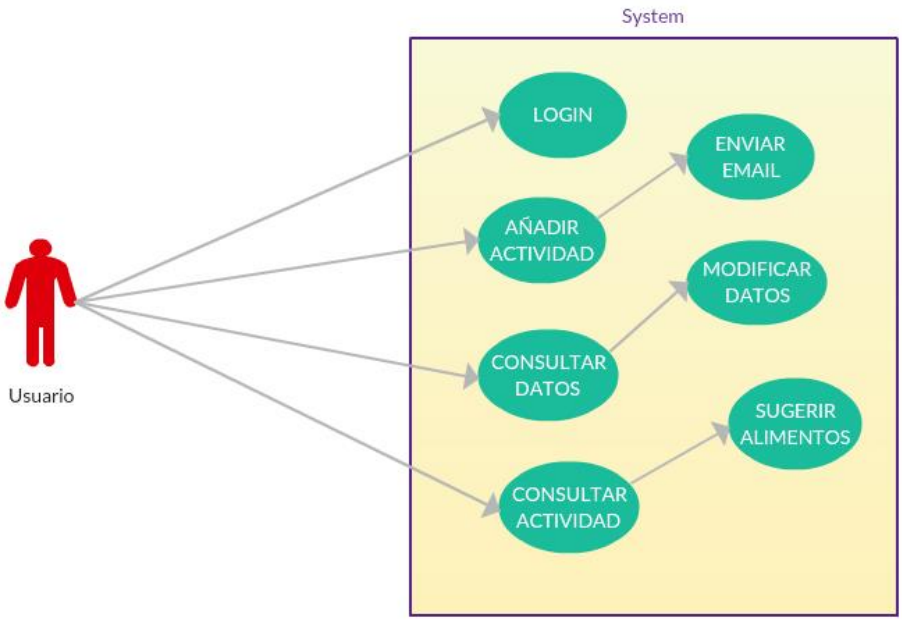


Figura 7. Diagrama Casos de Usos.

A continuación vamos a proceder a definir los casos de usos mostrados en la Figura 7 de una manera breve:

- Login: Autenticación de un usuario con cuenta Google.

- Escribir Actividad: Añadir registro a Google Fit.
- Consultar Actividad: Leer registro de Google Fit.
- Consejo Nutricional: Selección de alimentos en función de las calorías quemadas.
- Consultar Datos Diabéticos: Consulta los datos personales del usuario asociado a la diabetes y almacenado también como registro de Google Fit.
- Modificación Datos Diabéticos: Modificación de los datos personales del usuario asociados a la diabetes.
- Enviar Email: la denominaremos como Alerta Responsable y consiste en un aviso a un responsable/contacto, mediante un email o vía telefónica, en caso de síntomas de hiperglucemia o hipoglucemia según los valores calóricos.

Para finalizar, mostramos unas tablas que expresan, con más lujo de detalles, los casos de usos previamente definidos.

Nombre	Login
Descripción	Acceso a la aplicación
Actores	Usuario
Precondiciones	Existencia de cuenta Google.
Flujo Normal	- Seleccionar cuenta Google del dispositivo - Conectar
Excepción	No dispone de conexión a Internet
Postcondiciones	Ninguna

Caso Login.

Nombre	Añadir actividad
Descripción	Añadir registro a Google Fit
Actores	Usuario
Precondiciones	Usuario conectado con cuenta Google.
Flujo Normal	1- Seleccionar Deporte/Ejercicio a añadir 2- Añadir tiempo de duración en minutos 3- Añadir distancia recorrida en metros 4- Añadir/modificar las calorías quemadas. 5- Pulsar botón “Añadir” para escribir el registro.
Excepciones	Datos mal introducidos
Postcondiciones	Ninguna

Caso Añadir Actividad.

Nombre	Consultar actividad
Descripción	Consultar actividad de Google Fit
Actores	Usuario
Precondiciones	Usuario conectado con cuenta Google.
Flujo Normal	1- Seleccionar fecha 2- Mostrar información de actividad
Excepciones	Ninguna
Postcondiciones	Ninguna

Caso Consultar Actividad.

Nombre	Sugerir Alimentos
Descripción	Mostrar alimentos en función de las calorías
Actores	Ninguno
Precondiciones	Consultar actividad.
Flujo Normal	1- Mostrar listado alimentos. 2- Mostrar comidas asociadas.
Excepciones	Ninguna
Postcondiciones	Ninguna

Caso Sugerir Alimentos.

Nombre	Consulta Datos Diabéticos
Descripción	Mostrar los datos diabéticos asociados al usuario
Actores	Usuario
Precondiciones	Usuario conectado con cuenta Google.
Flujo Normal	1- Mostrar listado de datos personales relacionados con la diabetes, estos son: calorías, email y teléfono. Mostrar configuración de envíos automáticos.
Excepciones	Ninguna
Postcondiciones	Ninguna

Caso Consulta Datos Diabéticos.

Nombre	Modificación Datos Diabéticos
Descripción	Modificación de los datos diabéticos asociados al usuario cuando este lo estime oportuno.
Actores	Usuario
Precondiciones	Usuario conectado con cuenta Google.
Flujo Normal	1- Modificar límite de calorías. 2- Modificar teléfono de contacto. 3- Modificar email contacto. 4- Modificar configuración de envíos automáticos
Excepciones	Ninguna
Postcondiciones	Ninguna

Caso Modificación Datos Diabéticos.

Nombre	Alerta Responsable
Descripción	Alertar a un responsable mediante un email o contacto telefónico en caso de sobrepasar el umbral calórico.
Actores	Usuario
Precondiciones	Escribir Actividad y conexión a Internet.
Flujo Normal	1- Comprobar los valores introducidos. 2- En caso de pasar el umbral calórico, mostrar un aviso al usuario actual y mandar un email o avisar telefónicamente o sms al contacto asociado al usuario.
Excepciones	Datos mal introducidos
Postcondiciones	Ninguna

Caso Alerta Responsable

Diagrama de secuencia

A continuación mostraremos el diagrama de secuencias, para hacer ver al lector las interacciones, entre objetos, en el orden secuencial en el cual las interacciones ocurren. Dicho diagrama lo podemos ver en la Figura 8, no obstante vamos a comentar un poco más cómo funcionan estos diagramas.

Estos diagramas, detallan cómo se llevan a cabo las operaciones a través de mensajes, de los cuales hay varios tipos pero nosotros sólo emplearemos las llamadas a operaciones. Están organizados en función del tiempo, que es representado de manera descendente y los objetos se representan de izquierda a derecha en función de cuándo toman parte en la secuencia de mensajes.

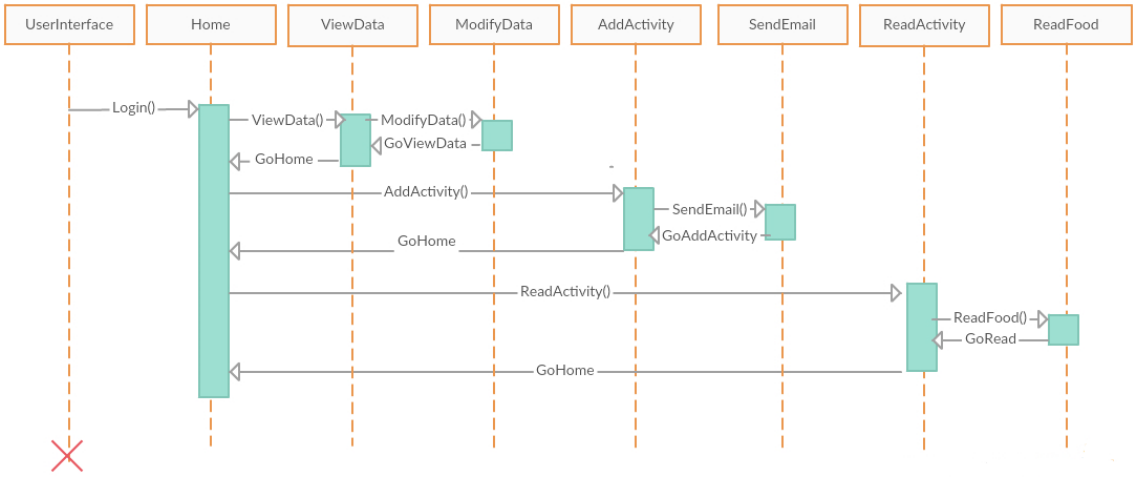


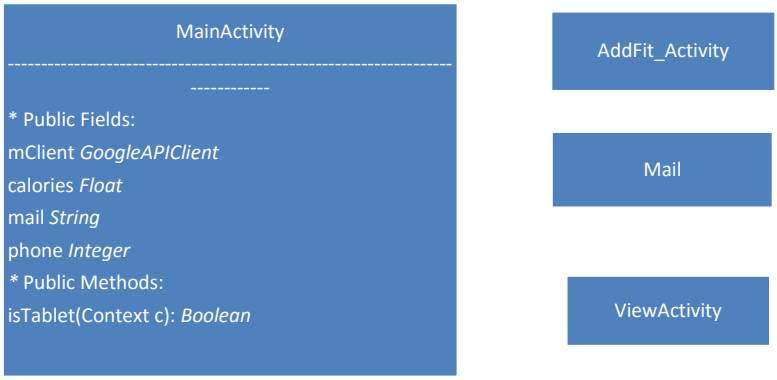
Figura 8. Diagrama de secuencias

Diagrama de clases

Otro de los diagramas a mostrar será el de clases, el cual nos detalla los atributos, métodos y visibilidad de las clases así como las relaciones existentes entre ellas y sus colaboraciones.

En nuestra aplicación, el modelo de clases empleado para el correcto funcionamiento de la aplicación es muy simple, formado únicamente por 4 clases y sin relaciones entre ellas a nivel de clase pero con dependencia de variables entre ellas. Vamos a describir brevemente el funcionamiento de cada clase, a destacar las dependencias que se dan entre ellas y por último mostraremos un pequeño diagrama.

- MainActivity: Actividad principal donde se establece la conexión con Google y se inicializan los valores de los datos del usuario. Cabe destacar que aquí haremos públicas 4 variables que corresponden al objeto de Google que almacena la conexión, y los 3 datos (calorías, teléfono, email) asociados al usuario y una función que comprueba el tamaño de la pantalla.
- AddFitActivity: Esta actividad depende de las 4 variables públicas comentadas en la Activity anterior para su correcto funcionamiento: añadir una actividad a Google Fit y avisar a un contacto en caso de requerirlo.
- ViewActivity: Aquí en esta actividad mostraremos los datos de las actividades asociadas a Google Fit y para ello únicamente necesitaremos el objeto público de la clase MainActivity donde usaremos esa conexión con Google para obtener esos datos.
- Mail: Esta actividad está enfocada al envió de un email de manera automática sin interacción por parte del usuario. Esta clase únicamente será creada desde la clase AddFitActivity en caso de que el usuario desee realizar dicha acción.



Metodología

La metodología con la que se va a llevar a cabo el desarrollo de la aplicación para móviles ControlDiabetes es la metodología ágil SCRUM. De igual manera, la organización de la memoria en los siguientes capítulos corresponde con dicha metodología.

SCRUM es un proceso pensado para equipos de trabajo, aunque en nuestro caso lo emplearemos de manera individual, que promueve la innovación, motivación y compromiso del equipo debido a la interacción que existe con el cliente final, basado en un trabajo constante, iterativo e incremental.

Una de las claves de esta metodología es la división del trabajo en distintos bloques que pueden llevarse a cabo en periodos cortos de tiempos, entre 1 y 4 semanas, que se denominan iteraciones (o sprints). Una vez finalizadas estas iteraciones, se deben presentar los resultados al cliente final consiguiendo así una mayor involucración por parte del cliente que permitan modificar ciertos requisitos que vea necesario en cada iteración y que además podrá contemplar el crecimiento de su producto. Las iteraciones se llevan a cabo siguiendo 3 procesos: planificación de la iteración, ejecución de la iteración, inspección y adaptación.

La planificación de la iteración consiste en realizar una reunión con el cliente, donde éste proporcionará al equipo una lista de requisitos priorizada del producto/proyecto, y el equipo resolverá sus dudas y seleccionará los requisitos más prioritarios para ser completados en la iteración, de manera que puedan ser entregados al cliente si éste los solicita. A continuación el equipo planificará la iteración elaborando una lista de tareas, que se repartirán entre ellos, necesarias para su cumplimiento.

Seguidamente, se procede a la ejecución de la iteración donde cada miembro del equipo realizará la/s tarea/s asignada/s y mediante una reunión de sincronización cada miembro presentará su trabajo y examinará el del resto del equipo (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. Estas reuniones son reuniones breves donde cada miembro deberá responder las 3 siguientes preguntas:

- *¿Qué he hecho desde la última reunión?*
- *¿Qué voy a hacer a partir de ahora?*
- *¿Qué impedimentos tengo o voy a tener?*

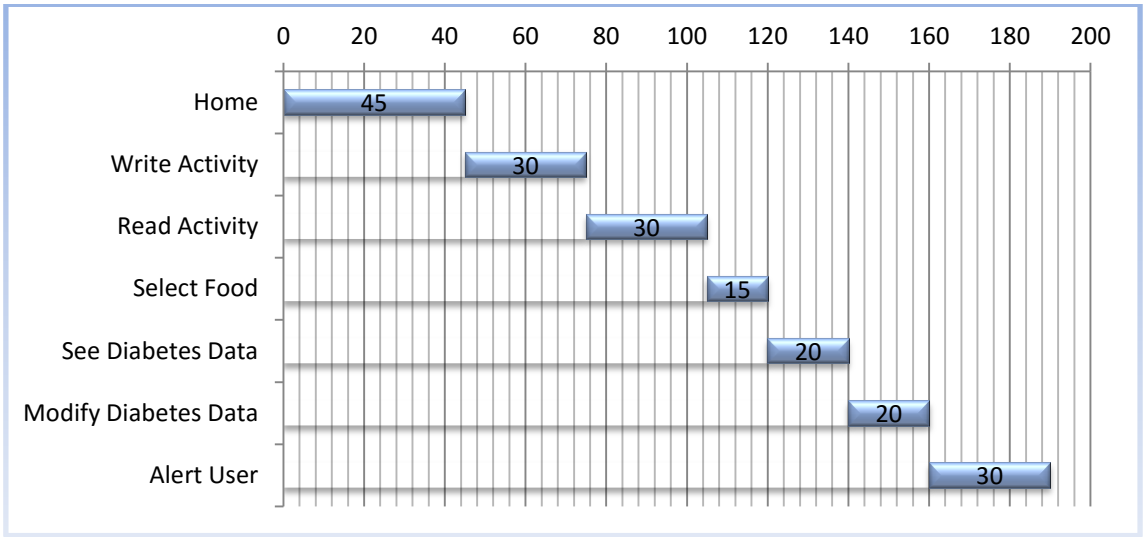
Y, por último, la inspección y adaptación, este paso se realiza el último día de la iteración, que consiste en una reunión de revisión de la misma. Formada por 2 partes:

- Demostración: el equipo muestra al cliente los requisitos completados y según los resultados y valoración del cliente se procede a realizar las adaptaciones necesarias replanificando el proyecto.
- Retrospectiva: se analiza cómo ha sido el trabajo del equipo y los posibles problemas que podrían surgir en el avance del proyecto. [21]

Una vez visto el funcionamiento de la metodología SCRUM, procedemos a definir las iteraciones llevadas a cabo en el proyecto ControlDiabetes para su resolución. Dichas iteraciones serán descritas en los siguientes capítulos de la memoria:

1. Home: conlleva todo el proceso de login y la carga de la pantalla principal de la aplicación.
2. Write Activity: proceso de inserción de datos en la nube de Google Fit asociados a la cuenta logueada.
3. Read Activity: proceso de lectura de datos de Google Fit asociados al usuario logueado.
4. Select Food: proceso de selección de alimentos en función de las kilocalorías. Proceso totalmente dependiente de Read Activity.
5. See Diabetes Data: lectura de datos de Google Fit asociados a los datos diabéticos del usuario logueado.
6. Modify Diabetes Data: modificación de datos diabéticos asociados al usuario logueado.
7. Alert User: aviso al responsable de la persona en caso de posibles síntomas de hiperglucemia o hipoglucemia al introducir los datos. Dicho aviso será mediante un email o empleando un contacto telefónico (sms).

Para finalizar lo referente a la metodología, mostraremos un diagrama de Gantt donde detallamos una estimación de las horas empleadas en cada iteración, sumando un total de 190, tal y como hemos visto previamente.



Home

Requisitos de implementación

En esta primera fase se llevará a cabo el diseño y funcionalidad de la página inicial, o home, de ControlDiabetes. Vamos a detallar los requisitos establecidos para la completa funcionalidad de dicha página y, por consiguiente, la finalización de esta primera fase.

El primer requisito que nos encontraremos, una vez creado el proyecto, es la configuración de la aplicación a través del fichero build.gradle del directorio app para la información de la app. Este fichero es creado automáticamente cuando creamos el proyecto, donde debemos indicar a la aplicación la versión mínima del sistema de Android necesaria para poder correr la aplicación, además de otros valores como el nombre del paquete por ejemplo. Dicho punto es importante ya que, al trabajar con la API de Google Fit, es necesario que, cómo mínimo, la versión sea la 9, Android 2.3, para su completa funcionalidad.

Una vez establecidas las bases del proyecto, comenzaremos con el diseño gráfico de la aplicación que constará de dos partes. La primera será la referente a la estructura base sobre la cual presentar el contenido. La segunda parte del diseño es la referida a los elementos a mostrar en la pantalla sobre la estructura ya definida, en este caso. Aquí es donde daremos usos de los Layouts que define el sistema Android para el entorno gráfico. Para el diseño gráfico combinaremos las opciones gráficas que nos proporciona Android junto a una librería de bootstrap especial para Android.

Por último, nos quedará el login de la aplicación a través de las cuentas Google y la configuración de éstas para el uso de la API Google Fit. Dicho paso, tal y como hemos visto en las especificaciones de Google Fit, requiere de una cuenta Oauth asociada al proyecto y agregar dicho proyecto a Google Developers Console para la conexión, de manera satisfactoria, con Google y sus servidores.

Desarrollo

Lo primero de todo, al ser la primera fase, deberemos crear el proyecto desde 0 en el entorno de Android Studio. El nombre del proyecto y el paquete serán controldiabetes y controldiabetes.jesus.controldiabetes respectivamente. El tipo de dispositivo que se va a emplear para el desarrollo de la aplicación será o podrá ser móvil/Smartphone o 32olíti y como SDK el indicado como primer requisito en el apartado anterior, API 9 (Android 2.3). Por último, seleccionaremos el tipo de Activity como principal, en nuestro caso Blank_Activity que viene por defecto, y lo

nombraremos como MainActivity, al igual que el Layout que se creará también por defecto.

Una vez establecida la estructura de ficheros del proyecto, procedemos a establecer los permisos básicos necesarios para el correcto funcionamiento del proyecto, mediante el fichero AndroidManifest.xml. En nuestro caso, los requisitos fundamentales serán la conexión a Internet, para el acceso a Google, y la autenticación y obtención de cuentas de Google para su uso en el proyecto. Dichos permisos los estableceremos en la parte superior del fichero quedando como muestra la Figura 9.

```
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figura 9. Permisos AndroidManifest.xml

Con esto ya cumpliríamos con el segundo requisito y el siguiente paso será la elaboración de la interfaz del usuario. Como ya hemos indicado en la fase de requisitos, la dividimos en 2 fases en donde la primera, la estructura base, será la definida por defecto por Android Studio la cual no vamos a modificar. Dicha parte consta de la definición de las variables de estilos, colores, dimensiones y cadenas de texto en el directorio *values*, respectivamente en los ficheros style.xml, colors.xml y strings.xml. Estos valores forman una interfaz básica formada por, de manera resumida, una cabecera con un tono azulado y el resto del cuerpo en un tono gris.

Tal y como hemos comentado, además de lo que nos ofrece Android, emplearemos una librería específica basada en bootstrap para Android. Dicha librería la obtendremos de un repositorio Git y la importaremos a nuestro proyecto como un módulo. A continuación, y para el correcto funcionamiento del módulo, en nuestro MainActivity.java deberemos añadir la siguiente llamada TypefaceProvider.registerDefaultIconSets(). Con esto ya tendremos operativa nuestra librería de bootstrap para su uso en la interfaz gráfica. [22]

Ahora procedemos a realizar el diseño de la interfaz de usuario correspondiente a la primera página a mostrar, la página inicial o home. Tal y como hemos comentado antes, al crear el proyecto se nos creó por defecto una Activity, que denominamos MainActivity, que elegimos de tipo Blank Activity. A nivel de diseño esta elección nos creó dos ficheros en el directorio *layout*, content_main.xml y activity_main.xml, donde este último contiene al primero mediante un include.

El content_main.xml está formado por un RelativeLayout, que es uno de los contenedores de interfaces que nos proporciona Android, y dentro del cual también usaremos otros contenedores. En primer lugar estableceremos un mensaje de texto, que inicialmente será “Conectando…” y una vez logueado cambiará a “Bienvenido *usuario*”, dicho mensaje estará contenido en un LinerLayout para que, independiente

del dispositivo y la orientación, aparezca centrado en la pantalla. A continuación, mostraremos los dos botones correspondientes a las funciones principales que realizarán la aplicación, “Añadir Actividad”, “Ver Actividad” y “DATOS”. Al igual que en el caso anterior aparecerán centrados en la página pero esta vez emplearemos otro RelativeLayout ya que nos permite el posicionamiento en relación con los otros elementos. Vemos en la Figura 10 una de las etiquetas, y sus atributos (los cuales serán los de la librería bootstrap y los nativos de Android), empleadas para la creación de uno de los botones, dicha estructura es semejante para el texto salvo en el nombre de la etiqueta.

```
<com.beardedhen.androidbootstrap.BootstrapButton
    android:id="@+id/add"
    android:layout_width="150dp"
    android:layout_height="60dp"
    android:layout_alignParentLeft="true"
    android:layout_marginTop="95dp"
    android:text="Añadir actividad"
    app:bootstrapSize="lg"
    app:bootstrapBrand="primary"
    app:buttonMode="regular"
    app:showOutline="false"
    app:roundedCorners="true"
    android:onClick="addActivity"/>
```

Figura 10. Etiqueta XML para botón

Y, por último, a modo de enriquecer la pantalla cara al usuario, mostraremos una imagen que simboliza a una persona haciendo deporte. A diferencia de los casos anteriores, la imagen, mediante una propiedad asociada a ésta, aparecerá centrada pero sin llevar un contenedor especial a la misma. El resultado visual de la página de inicio se muestra a continuación en la Figura 11.

Figura 11. Pantalla de inicio.

Además del include, activity_main.xml contendrá un elemento NavigationView que mostrará el menú a la izquierda clásico deslizante. Dicho menú contará con las mismas acciones que la pantalla principal, además de una cabecera con una imagen y el nombre del usuario logueado. El código XML será como se muestra a continuación:

```
<android.support.design.widget.NavigationView
    android:id="@+id/navview"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    app:headerLayout="@layout/header_navview"
    app:menu="@menu/menu_main" />
```

Tal y como vemos para completar el menú requerimos de los elementos header_navview y menu_main que serán unos ficheros XML donde el primero estará formado por la imagen y el campo texto donde se mostrará el usuario con un formato similar al visto anteriormente. Y el menu_main está formado por varios elementos, donde tenemos una raíz <menú> y después tenemos un primer elemento <group> donde tendremos los elementos, <ítem>, que deseemos como primer grupo. Y, por último, a continuación tendremos los elementos <ítem> que serán para indicar las distintas secciones con los submenús que deseemos. La estructura quedará como sigue:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/view_act"
            android:icon="@android:drawable/ic_menu_my_calendar"
            android:title="Ver Actividad"
            />
    </group>
    <item
        android:id="@+id/navigation_subheader"
        android:title="Ajustes">
        <menu>
            <item
                android:id="@+id/data"
                android:icon="@android:drawable/ic_menu_info_details"
                android:title="Datos"/>
        </menu>
    </item>
```

Para enlazar estos elementos debemos establecer estas líneas en el fichero java:

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
ActionBar actionBar = getSupportActionBar();
actionBar.setHomeAsUpIndicator(R.drawable.ic_menu_white);
actionBar.setDisplayHomeAsUpEnabled(true);
```

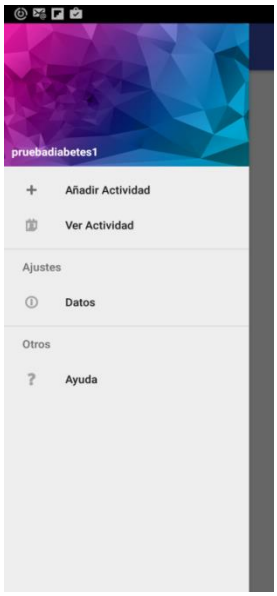
Y, en el mismo java, procedemos a programar el evento asociado a cada elemento del menú de la siguiente manera:

```
NavigationView n = (NavigationView) findViewById(R.id.navview);
n.setNavigationItemSelectedListener(new
```

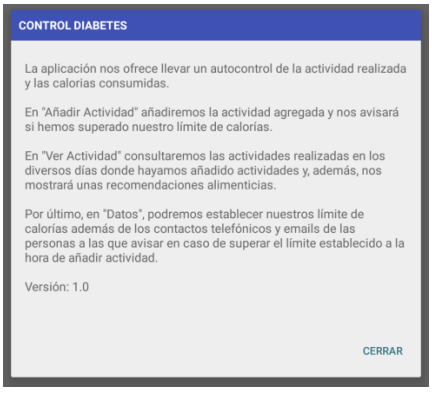
```
NavigationView.OnNavigationItemSelectedListener() {
    @Override
    public boolean onNavigationItemSelected(MenuItem menuItem) {

        switch (menuItem.getItemId()) {
            ////.......
            case R.id.add_act:
                addActivity(v);
                break;
            ..........
```

Una vez configurado el resultado visual del menú quedaría como se muestra a continuación:

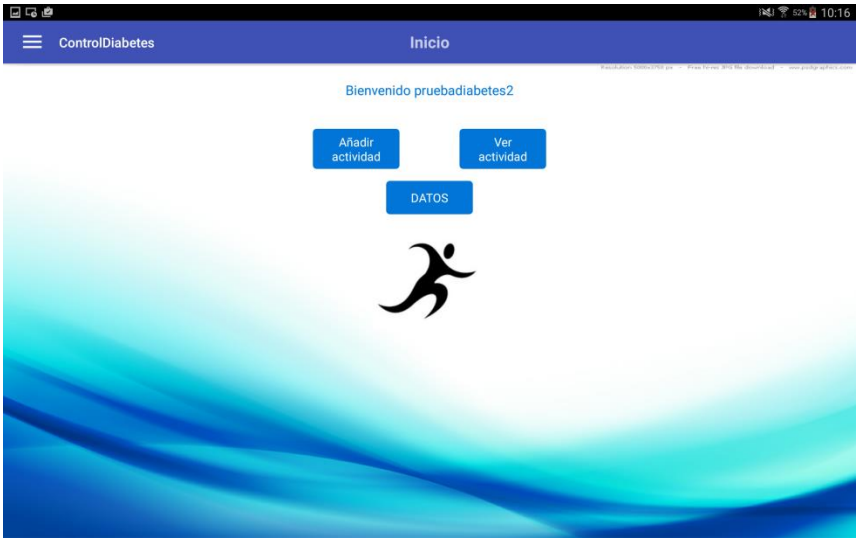


Como bien se aprecia en la imagen anterior hay una sección denominada Ayuda. Dicha sección nos desplegará una ventana a modo de pop-up formada por un LinearLayout que contendrá un FrameLayout formado a su vez por una imagen (que será a modo de background superior) y un título en un TextView, y, a continuación, un TextView el cual se empleará para mostrar la ayuda referente a la aplicación la cual estará alojada en un fichero. La lectura del mismo lo haremos mediante la clase BufferedReader, y que detallaremos más adelante. El resultado visual será el siguiente:



Como último detalle a nivel gráfico, vamos a implementar una función para que detecte el tipo de dispositivo, Smartphone o Tablet, en el que corre la aplicación en función del tamaño de la pantalla. Dicha función se empleara para establecer un fondo de pantalla en la página principal en caso de ser una Tablet para ganar enriquecimiento gráfico. La función y el resultado de la misma es la siguiente:

```
public static boolean isTablet(Context context) {
    return (context.getResources().getConfiguration().screenLayout &
Configuration.SCREENLAYOUT_SIZE_MASK) >=
Configuration.SCREENLAYOUT_SIZE_LARGE;
}
```



Ya, como último punto de esta primera fase, solo nos queda la conexión con Google para el uso de Google Fit, la cual está detallada en la documentación oficial de Google Fit. Para poder utilizar Google Fit en Android necesitamos:

- Habilitar la Fitness API en la consola Google Developers.
- Crear un ID cliente Oauth 2.0
- Conectar la Fitness API en la aplicación ControlDiabetes.

Primero debemos crear un nuevo proyecto a través de Google Developer Console. Una vez nos encontremos ahí, deberemos crear un proyecto haciendo clic en “Seleccionar Proyecto” y, a continuación, en “Crear Proyecto” situado en la esquina superior derecha. Nos pedirá el nombre del proyecto a crear, el cual solo nos permite letras, números, comillas, guiones, espacios y signos de exclamación, aceptaremos las condiciones y le daremos al botón “Crear”. Vemos dichas imágenes a continuación en la Figura 12.

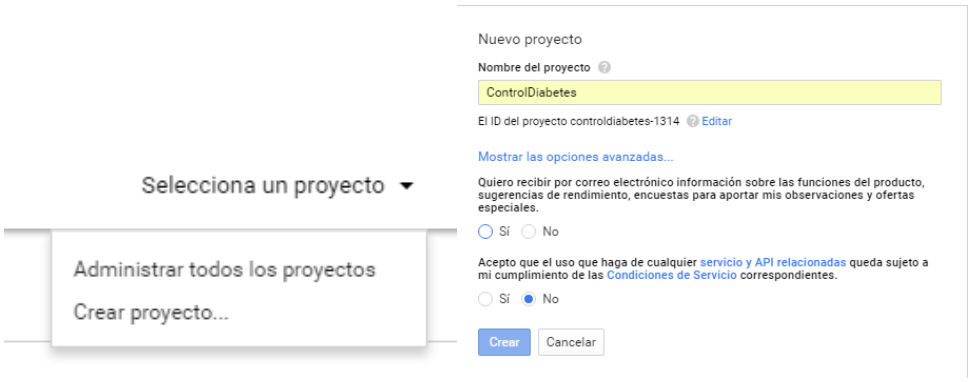


Figura 12. Creación proyecto Console Developers Google

Esto nos retornará a la página principal donde en el centro veremos un listado de las APIs de Google y un buscador, el cual vamos a emplear. Teclearemos la palabra "Fit" y le daremos a Intro, obteniendo la imagen de la izquierda de la Figura 14. Seleccionaremos Fitness API, nos cargará la información relativa a la API y veremos el botón de "Habilitar" en azul, el cual seleccionaremos, como vemos en la imagen de la derecha de la Figura 14.

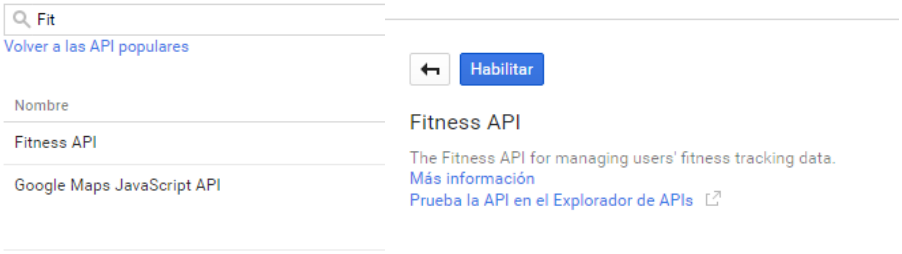


Figura 14. Habilitar Fitness API

Automáticamente Google nos notificará que debemos crear unas Credenciales y nos mostrará un enlace en forma de botón azul que nos llevará a la página adecuada. Una vez en esta página, crearemos una credencial seleccionando "Crear Credenciales" y nos pedirá el tipo de credencial, como vemos en la Figura 15. En nuestro caso la credencial será de Id de cliente Oauth.

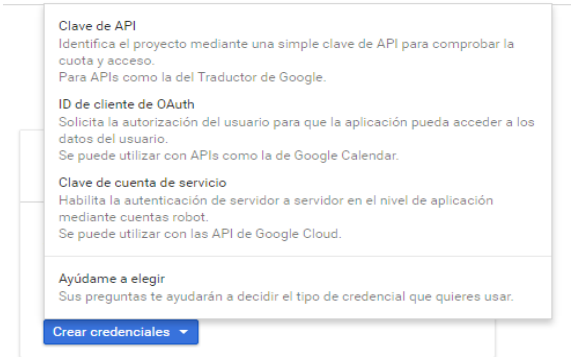


Figura 15. Crear credenciales I.

Seguidamente nos llevará a una página, Figura 16, que obligatoriamente nos hará seleccionar la opción de “Configurar pantalla de autorización”. En esta nueva pantalla, Figura 17, deberemos introducir los datos que se verán en la pantalla de autorización que se mostrará al usuario. Una vez rellenada volveremos a la pantalla anterior, Figura 16, para seleccionar el tipo de aplicación que en nuestro caso será Android.

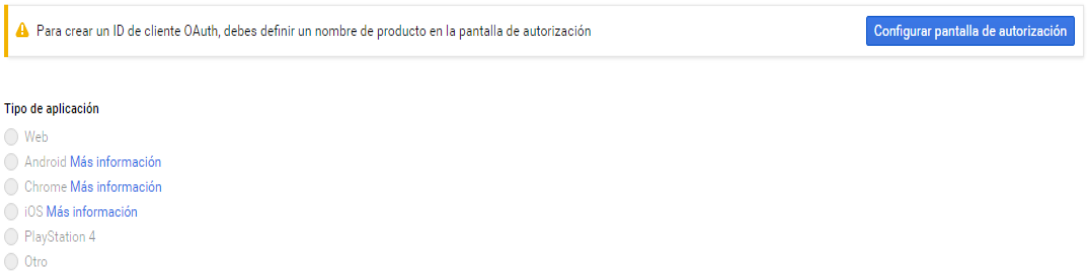


Figura 16. Crear Credenciales II.

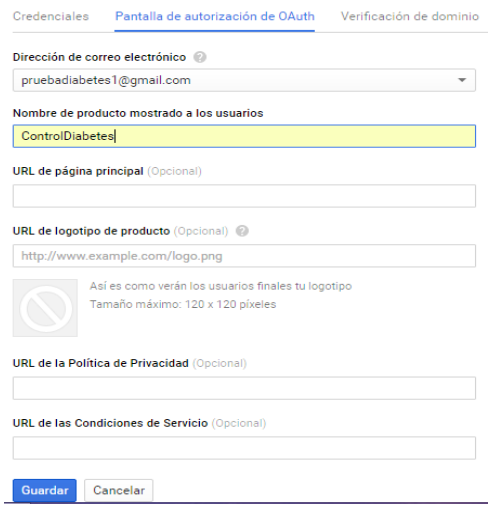


Figura 17. Crear Credenciales III.

Cuando seleccionemos Android, se nos desplegará un menú inferior con más información a facilitar, como vemos en la Figura 18. Aquí introduciremos el nombre de la credencial, el nombre del paquete Android definido en AndroidManifest.xml y la huella digital asociada a las aplicaciones del usuario, la cual obtendremos con las instrucciones mostradas en la Figura 18. Con estos datos introducidos, le daremos a “Crear” y nos aparecerá una ventana para “Aceptar” el ID creado.

Figura 18. Crear Credenciales IV.

Con este proceso finalizado ya tendremos creados el proyecto y la credencial de acceso y ahora deberemos configurar nuestro proyecto ControlDiabetes para su correcto funcionamiento con la Fitness API de Google. Buscamos el archivo build.gradle del proyecto y dentro de éste el nodo de *dependencies* para añadir la siguiente línea:

```
compile 'com.google.android.gms:play-services-fitness:8.4.0'
```

Para que esto dé resultado deberíamos tener agregado, mediante el SDK Manager, las librerías de Google Play al proyecto.

Ya disponemos de todo preparado para la parte más importante de esta fase, la conexión con Google. A continuación, y siguiendo la documentación oficial de Google, vamos a crear el objeto que usaremos para la conexión con Google Fit y la interacción con éste. Dicho objeto será creado e iniciado en la única clase que tenemos actualmente, MainActitivy.

Como primer paso deberemos crear el objeto asociado a Google que realizará la conexión y el resto de consultas a GoogleFit. Se denominará *mClient* y será un objeto de visibilidad pública para su uso en otras Activities:

```
public static GoogleApiClient mClient = null;
```

Posteriormente, en el procedimiento de creación de MainActivity, onCreate, definiremos una llamada al procedimiento *buildFitnessClient()* donde iniciaremos y conectaremos con Google nuestro objeto *mClient*.

```
Private void buildFitnessClient() {
    // Create the Google API Client
 mClient = new GoogleApiClient.Builder(this)
      .addApi(Plus.API)
      .addApi(Fitness.HISTORY_API)
      .addScope(new Scope(Scopes.FITNESS_ACTIVITY_READ_WRITE))
      .addScope(new Scope(Scopes.FITNESS_LOCATION_READ_WRITE))
      .addScope(Plus.SCOPE_PLUS_LOGIN)
      .addScope(Plus.SCOPE_PLUS_PROFILE)
      .addConnectionCallbacks(
       new GoogleApiClient.Conne ctionCallbacks() {
             @Override
          public void onConnected(Bundle bundle) {
            Log.i(TAG, "Connected!!!");

            TextView txtCambiado = (TextView)findViewById(R.id.saludo);
            txtCambiado.setText("Bienvenido " +
Plus.AccountApi.getAccountName(mClient).substring(0,Plus.AccountApi.ge
tAccountName(mClient).indexOf("@")));
                        }
              @Override
           public void onConnectionSuspended(int i) {
        // If your connection to the sensor gets lost at some point,
        // you'll be able to determine the reason and react to it here.
             if (i == ConnectionCallbacks.CAUSE_NETWORK_LOST) {
              Log.i(TAG, "Connection lost.  Cause: Network Lost.");
        }else if (i == ConnectionCallbacks.CAUSE_SERVICE_DISCONNECTED){
              Log.i(TAG, "Connection lost.  Reason: Service
Disconnected");
              }
            }
        }
             )
        .addOnConnectionFailedListener(
           new GoogleApiClient.OnConnectionFailedListener() {
               // Called whenever the API client fails to connect.
                 @Override
             public void onConnectionFailed(ConnectionResult result) {
                Log.i(TAG, "Connection failed. Cause: " +
result.toString());
               if (!result.hasResolution()) {
                // Show the localized error dialog

GooglePlayServicesUtil.getErrorDialog(result.getErrorCode(),
                                          MainActivity.this, 0).show();
                 return;
               }
         // The failure has a resolution. Resolve it.
         // Called typically when the app is not yet authorized, and an
         // authorization dialog is displayed to the user.
                if (!authInProgress) {
                  try {
                    Log.i(TAG, "Attempting to resolve failed
connection");
                    authInProgress = true;

result.startResolutionForResult(MainActivity.this,
                                                 REQUEST_OAUTH);
                } catch (IntentSender.SendIntentException e) {
            Log.e(TAG,"Exception while starting resolution activity",
e);
```

```
                    }
                }
            }
        }
    )
    .build();
}
```

Justo en el primer punto definimos las API de Google que va a llevar el objeto, en nuestro caso Google Fit y Google Plus con estás sentencias: *.addAPI(Plus.API)* y *.addAPI(Fitness.HISTORY_API)*. En el caso de Google Plus, el cual es la primera vez que mencionamos, únicamente lo usaremos para recoger el nombre del usuario conectado. En lo referente a Google Fit, como vimos hay varias API de las que disponer pero para nuestra aplicación usaremos HISTORY_API la cual nos permite la introducción y lectura de datos manuales. A continuación vemos *.addScope()* que nos va a definir el ámbito donde se moverá nuestra aplicación, en nuestro caso:

```
Scopes.FITNESS_ACTIVITY_READ_WRITE  // Leer/escribir actividad
Scopes.FITNESS_LOCATION_READ_WRITE  // Leer/escribir localización
Plus.SCOPE_PLUS_LOGIN               // Datos PLUS de Login
Plus.SCOPE_PLUS_PROFILE             // Datos PLUS del Perfil
```

Por último, tenemos las definiciones de las conexiones con Google, en caso de conexión exitosa y en caso de conexión fallida. A uno de estos procedimientos insertados, como nos especifica la documentación de Google, le hemos hecho una pequeña modificación. Dicho procedimiento es el de la conexión satisfactoria, en el cual podemos apreciar que cogemos el TextView que hemos definido a modo de saludo y le cambiamos el mensaje para que nos muestre el nombre de usuario conectado.

Con esto damos por finalizada la fase de conexión con Google, lo cual nos lleva al último punto de esta primera iteración: la definición de las funciones asociadas a los 2 botones creados. Dichas funciones son llamadas en el atributo *android:onClick* de la etiqueta XML definida en *content_main.xml*, ambas tienen una estructura similar y lo que realizan es lo siguiente: crear un objeto Intent para inicializar la llamada a la nueva Activity y lanzarlo, vemos dicho código a continuación para el ejemplo de añadir actividad:

```
public void addActivity(View view) {
    // Do something in response to button
    Intent i = new Intent(MainActivity.this, AddFit_Activity.class);
    MainActivity.this.startActivity(i);
}
```

La primera vez que conectemos la aplicación en nuestro dispositivo, nos aparecerá la selección de cuentas de Google para asociar a la aplicación. Cabe resaltar que en caso de que no se seleccione dicha cuenta la aplicación procederá a eliminar visiblemente los botones mostrados en el menú principal.

Write Activity

Requisitos de implementación

En esta fase se llevará a cabo el diseño y funcionalidad de la página que nos permitirá añadir datos, actividades en nuestro caso, a Google Fit de nuestra aplicación ControlDiabetes. Al igual que en la fase anterior, y en posteriores, vamos a detallar los requisitos establecidos para la completa funcionalidad de fase.

Primeramente es obligatorio la completa funcionalidad de la primera fase, especialmente la parte de Login, ya que será de vital importancia para ésta y las otras fases en las que operemos con Google Fit. Añadiremos una Activity nueva, que se llamará AddFit_Activity, que, al igual que en el caso anterior, será una Blank Activity, lo cual nos generará los 2 ficheros de interfaz gráfica (Layout), content_addfit y activity_addfit. A diferencia respecto a la fase anterior, cuando establecemos el nombre de la activity y del layout nos aparece la opción de establecer un padre de esta nueva activity. Estableceremos como padre la MainActivity que hemos creado y configurado en la fase anterior como vemos en la Figura 19, ya que de ésta usaremos el objeto de Google, mClient, para agregar las actividades.

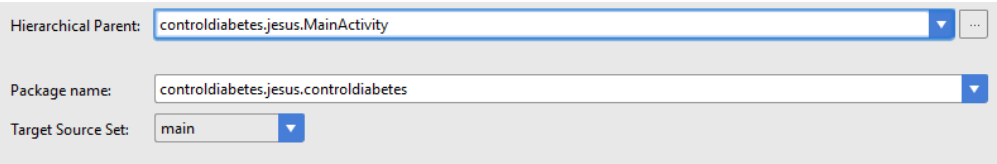


Figura 19. Herencia Activity

La interfaz del usuario será clara y simple para que el usuario no tenga dudas a la hora de rellenar los campos. Para la cual se empleará la base establecida en la fase previa que nos proporciona Android Studio.

Para la completa funcionalidad de esta fase, el usuario obligatoriamente deberá introducir todos los datos que se le solicitarán en pantalla para almacenarlos en Google Fit, en caso de que falte alguno no sé añadirá nada a Google Fit y se informará del parámetro que impide el almacenamiento de los datos. Los datos que el usuario deberá introducir son los siguientes:

- Actividad: seleccionar una actividad de las mostradas en el listado.
- Minutos: minutos empleados en la realización de la actividad.
- Metros: metros recorridos en el transcurso de la actividad.
- Calorías: calorías quemadas en dicha actividad.

Cabe mencionar que para algunas actividades específicas las calorías se calcularán automáticamente en función de esta actividad y del tiempo consumido. El

caso de los metros deberá ser introducido manualmente debido a que no se dispone de una relación entre tiempo y actividad que nos determine una cierta distancia recorrida.

Como última condición vamos a establecer que las actividades agregadas no podrán estar solapadas en el tiempo porque consideramos que el usuario agregará las actividades justo al final de éstas. Esta condición la llevaremos a cabo mediante una lectura previa de los datos de Google Fit insertados seleccionando únicamente los tiempos de dichas actividades, en caso de que existan, y comparando estos tiempos con los que se van a insertar para ver si están en el intervalo leído o no.

Desarrollo

En esta fase vamos a llevar a cabo 2 tareas principales: el diseño de la página para introducir los datos y la escritura de estos datos en los servidores de Google Fit. Comenzaremos con el diseño de la página.

Como hemos comentado en los requisitos, crearemos una Blank Activity nueva y dispondremos de 2 ficheros .xml para el diseño de la página, estos serán content_addfit y activity_addfit.xml. Al igual que en la fase anterior, el activity_addfit.xml contendrá la base del proyecto, fondo blanco y cabecera en azul, y dentro de esta incluiremos el contenido que se encuentra en el fichero content_addfit. Por lo que de este primer fichero no modificaremos nada.

El contenido de esta pantalla estará formado por los campos que vayamos a grabar en los servidores y un botón que será el encargado de llamar a la función correspondiente de realizar dicha tarea. Los campos a grabar, ya definidos en los requisitos, estarán formados por un elemento <TextView> que nos indica el valor a introducir, como los empleados en la home, y un campo <com.beardedhen.androidbootstrap.BootstrapEditText> como el que veremos en el siguiente fragmento de código. De los 4 campos establecidos, 3 serán introducidos de esta manera.

```
<com.beardedhen.androidbootstrap.BootstrapEditText
    android:layout_marginTop="10dp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text=""
    app:bootstrapSize="lg"
    app:bootstrapBrand="info"
    app:roundedCorners="true"
    android:inputType="phone"
    android:id="@+id/v1" />
```

El cuarto campo que nos falta por definir es el correspondiente a las actividades. Estas actividades las mostraremos en un listado desplegable mediante el elemento <Spinner> que vemos en la Figura 20.

```
<Spinner
android:layout_width="wrap_content"
android:layout_height="120dp"
android:id="@+id/select"
android:layout_gravity="center_horizontal"
/>
```

Figura 20. Elemento Spinner

A continuación vamos a hacer un pequeño paréntesis en lo que se refiere a los ficheros xml ya que veremos cómo agregar elementos al listado, en nuestro caso las actividades. Primero vamos a añadir en un fichero de texto plano el listado de actividades que deseemos, en este caso todas las actividades que nos proporciona Google Fit junto a su código identificador en Google. Dicho formato será el siguiente: *AEROBICS;9*.

Para esto usaremos AddFit_Activity.java donde usaremos un objeto lista de tipo List para almacenar los elementos del fichero y este será añadido al elemento <Spinner>. Usaremos la clase BufferedReader para la lectura del fichero y la inclusión en la lista donde solo añadiremos la parte correspondiente al nombre. Por último usaremos un ArrayAdapter para adaptar la lista al formato del <Spinner> y proceder con su carga:

```
Spinner spinner2 = (Spinner) findViewById(R.id.select);
List<String> list = new ArrayList<String>();

InputStream fraw = getResources().openRawResource(R.raw.activities);
BufferedReader reader = null;

try {
    reader = new BufferedReader(new InputStreamReader(fraw));
    String text = null;
    while ((text = reader.readLine()) ¡= null) {
        list.add(text.substring(0, text.indexOf(";")));
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (reader ¡= null) {
            reader.close();
        }
    } catch (IOException e) {
    }
}

ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item,list);
dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dr
```

```
opdown_item);
spinner2.setAdapter(dataAdapter);
```

Como hemos mencionado en los requisitos, uno de los campos introducidos, las calorías, será calculado automáticamente según la actividad, que será alguna específica, y la duración de la misma. No obstante, el usuario podrá estar en desacuerdo con el cálculo e introducir él mismo el valor que considere adecuado. Las actividades específicas sobre las que se auto calcularán son las siguientes:

- BASKETBALL
- FOOTBALL (SOCCER)
- BIKING
- WALKING
- TENNIS
- RUNNING
- SWIMMING

Este proceso de auto cálculo lo llevaremos a cabo en una hebra que se iniciará al iniciar la Activity y finalizará al finalizar ésta. Dicha hebra estará a la espera comprobando si los valores que se introducen en el tiempo son enteros. Una vez tengamos un valor entero en el campo de tiempo comprobaremos si la actividad seleccionada corresponde con alguna de las específicas y en dicho caso realizaremos el cálculo y lo mostraremos en el campo de texto. En caso de que la actividad no corresponda con una de las específicas no se hará ningún cálculo y el campo permanecerá como estaba. [23]

Por último, volviendo al tema gráfico centrado en los xml solo nos queda por comentar que a continuación de los campos de texto y entrada de datos habrá un botón “Añadir”, como los visto en el capítulo home, donde llamará a la función addFit() que será la encargada de todo el proceso de comprobación y grabación de datos en los servidores de Google. A continuación vemos el resultado de la interfaz gráfica en la Figura 21.

Figura 21. Pantalla añadir actividad.

De igual manera que en el capítulo previo, estableceremos un fondo de pantalla en el caso de que el dispositivo sea una Tablet.

Con esta función comenzamos la segunda parte de esta iteración que se corresponde con la inserción de los datos en los servidores de Google. Dicha función irá desarrollada en el fichero AddFit_Activity.java, el cual hemos mencionado antes para la carga de las actividades en el <Spinner>. Antes de empezar debemos señalar que el método para la inserción de datos, insertData, debe esperar la sincronización de los datos, lo que nos obligará a que la llamada a la inserción la hagamos en una tarea en segundo plano mediante una AsyncTask.

Primero recopilaremos los datos de los campos donde, en el caso de las actividades consultaremos el fichero para obtener el identificador de la actividad recibida. El resto de campos los obtendremos de los TextView y aplicaremos la siguiente conversión:

- Duración → Integer.
- Distancia → Float.
- Calorías → Float / 1000 (almacenamos las kilocalorías).

Al ser todos los campos numéricos manejaremos la excepción NumberFormatException para avisar al usuario en caso de que algún campo sea incorrecto. A continuación, usaremos la clase Calendar para establecer la duración y obtener el tiempo (momento de inicio y fin) en milisegundos:

```
Calendar cal = Calendar.getInstance();
Date now = new Date();
cal.setTime(now);
long endTime = cal.getTimeInMillis(); //cal.getTime().getTime();
cal.add(Calendar.MINUTE, -duracion);
long startTime = cal.getTimeInMillis();
```

El siguiente paso será establecer los DataSet, recordemos que son un conjunto de DataPoints los cuales representan los datos a insertar (manuales o recogidos de sensores). Primero crearemos un array de DataSet, denominado calDataSet, para añadir los distintos valores; después crearemos el DataSource que definirá el tipo de dato a almacenar y a partir de éste crearemos su DataSet correspondiente. Por último, crearemos el DataPoint que contendrá los datos indicándole además el inicio y fin de la actividad. Dicho proceso se muestra a continuación:

```
DataSource calory = new DataSource.Builder()
        .setAppPackageName(this)
        .setDataType(DataType.TYPE_CALORIES_EXPENDED)
        .setType(DataSource.TYPE_RAW).build();
calDataSet[0] = DataSet.create(calory);
DataPoint calDataPoint = DataPoint.create(calory).setTimeInterval(
        startTime, endTime, TimeUnit.MILLISECONDS);
```

```
calDataPoint.getValue(Field.FIELD_CALORIES).setFloat(kcal);
calDataSet[0].add(calDataPoint);
```

En este ejemplo ilustrado hemos empleado el tipo de datos de calorías quemadas, *type_calories_expended*, y el tipo de campo asociado a las calorías, *field_calories*. Para el resto de datos emplearemos respectivamente *type_distance_delta* y *field_distance* para la distancia, *type_activity_segment* y *field_activity* para las actividades. Por cada elemento, debemos aumentar la posición del array del DataSet.

Posteriormente grabamos los datos a Google Fit en la tarea asíncrona que hemos creado. Para ello recorreremos cada elemento del array y emplearemos el método comentado previamente para la inserción de los datos, insertData, como vemos aquí:

```
for (DataSet d: array) {
    Log.i(MainActivity.TAG, "Data :"+d);

    com.google.android.gms.common.api.Status insertStatus =
     Fitness.HistoryApi.insertData(MainActivity.mClient, d).await();

  if (¡insertStatus.isSuccess()) {
      Log.i(MainActivity.TAG, "There was a 48olítica inserting the
dataset. "+insertStatus.getStatusMessage());
  } else {
        Log.i(MainActivity.TAG, "Data insert was successful!");
  }
}
```

A continuación, volvemos brevemente al aspecto gráfico para indicar que mientras tenemos la tarea asíncrona en ejecución agregando los datos, visualizaremos una barra de progreso, *ProgressDialog*, que será lanzada por otra tarea, y una vez finalizado mostrará un mensaje al usuario indicando que se ha completado el proceso correctamente y vaciará los campos de texto. El código empleado es el siguiente:

```
dialogo = new ProgressDialog(AddFit_Activity.this);
dialogo.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
dialogo.setMessage("Agregando...");
dialogo.setMax(100);
……. ……

Toast.makeText(getApplicationContext(), "Actividad agregada",
Toast.LENGTH_LONG).show();
TextView txtCambiado = (TextView)findViewById(R.id.v1);
txtCambiado.setText("");
……. ……
```

Finalmente, y ya concluyendo esta fase, nos queda el último requisito, evitar el solapamiento de las actividades insertadas. Dicho proceso constará de 2 fases:

1- Lectura de tiempos de Google Fit.

2- Comparación de registros.

En la lectura de tiempos de Google Fit procederemos a leer los datos de Google Fit que hayamos insertados a lo largo del día pero únicamente cogeremos los tiempos de duración de dichas actividades. El proceso de lectura de datos de Google Fit lo explicaremos con más detalle en el siguiente capítulo. Los datos correspondientes a los tiempos los almacenaremos en un array unidimensional cuya longitud será de un número par y los tiempos serán almacenados uno a continuación del otro, siendo cada par de elementos el inicio y fin de la actividad.

Ahora comienza la segunda fase donde emplearemos estos tiempos para ver si el rango de duración de los que se van a insertar coinciden con alguno de estos pares almacenados. Dichos tiempos, tanto en la lectura como en la recopilación de los nuevos, serán expresados en milisegundos. La comparación será una comparación doble, de tal manera que compararemos que el inicio de la actividad a agregar no sea mayor que el inicio de una actividad almacenada y que no sea menor que el fin de la misma actividad almacenada. De la misma manera emplearemos esta lógica para el caso de fin de actividad. En caso de que por el inicio o por el fin de la actividad se descubra solapamiento, procederemos a actualizar el mensaje que se le mostrará al usuario al final del proceso y saldremos del proceso de agregación. La estructura empleada es la siguiente:

```
for (int i = 0; i < list_time.size(); i = i + 2) {
if ((startTime > list_time.get(i) && startTime < list_time.get(i + 1))
|| (endTime > list_time.get(i) && endTime < list_time.get(i + 1))) {
    mensj = "Actividad solapada, imposible agregar";
    return null;
  }
}
```

Una vez llegado al final del proceso donde se le informará al usuario del éxito o fracaso de la agregación de la actividad, procederemos nuevamente a lanzar la lectura de los datos, tiempos como hemos especificados, ya que en caso de que se hubiesen agregado los datos debemos hacer una nueva lectura para tener estos últimos datos más reciente para la comparación y con esto evitamos que el usuario añada 2 actividades de manera seguida evitando la solapación de actividades, que era nuestro principal problema en esta parte.

En el siguiente capítulo, veremos con más detalle el proceso de lectura de datos de Google Fit que hemos empleado en este capítulo.

Read Activity

Requisitos de implementación

Ahora llevaremos a cabo el proceso de lectura de los datos de Google Fit que hemos introducido mediante la fase anterior. Detallaremos el diseño de la página y el proceso de lectura de los datos.

El proceso de lectura de datos requerirá de interacción por parte del usuario, siendo necesaria la selección de una fecha para poder obtener los datos, si es que existen, para ese día concreto. Para ello, emplearemos un calendario que nos proporciona Android para una fácil elección del día por parte del usuario. Dicho calendario se denomina CalendarView. Al usar CalenderView de Android se nos presenta un requisito de compatibilidad y es que debemos actualizar la versión mínima del sistema Android a la 11 en el fichero build.gradle, ya que algunas de las funciones usadas están disponibles a partir de esta versión.

Este proceso se llevará a cabo en una Activity nueva que llamaremos ViewActivity y, al igual que los procesos anteriores, será de tipo Blank Activity. Los 2 ficheros de interfaz gráfica (Layout) se denominarán content_view_fit y activity_view_fit. Necesitaremos, obviamente, disponer de la variable que almacena la conexión con Google y la configuración de la API de Google Fit, por lo que la haremos hija de MainActivity.

La elección del uso del objeto CalendarView corresponde con la filosofía llevada a cabo en anteriores procesos, la de plantear al usuario una interfaz fácil y simple. En esta interfaz únicamente veremos el calendario y el usuario seleccionará el día que desee ver los datos. Estos datos se verán mediante una ventana emergente y será los establecidos en la fase de Write Activity, en caso de no haber datos la ventana aparecerá vacía.

Por lo tanto, y a modo de resumen, en esta fase el usuario podrá leer los datos introducidos manualmente mediante la fase anterior, Write Activity, mediante el uso de un calendario sin olvidar que es necesario, como siempre, la permanente conexión a Internet y el acceso al objeto de Google, mClient, que hemos creado y configurado en la primera fase Home.

Desarrollo

En esta fase podemos ver que el aspecto a grafico a tratar será muy simple por lo que lo verdaderamente importante será la lectura de los datos introducidos por parte del usuario y el tratamiento de estos datos de cara al resultado final, que es la representación de estos al usuario.

No obstante, y siguiendo como en los procesos anteriores, vamos a comentar los detalles de la interfaz gráfica y su realización. En este caso sólo precisaremos del fichero *content_view_fit.xml*, el cual contendrá únicamente el calendario del que hemos hablado previamente. La etiqueta XML correspondiente a dicho calendario es la siguiente:

```
<CalendarView
    android:id="@+id/calendar"
    android:layout_margin="10dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Con esto ya hemos concluido la fase perteneciente al diseño de la página de consultar los datos. El resultado visual lo podemos ver a continuación en la Figura 22.

Figura 22. Pantalla leer actividad

Ahora empezaremos con todo el proceso de lectura de datos. En primer lugar veremos cómo es la interacción con el calendario mostrado anteriormente. Inicializaremos el calendario con una serie de características, como, por ejemplo, no mostrar el número de semana, y estableceremos el evento para el cambio de fecha.

```
CalendarView calendar = (CalendarView) findViewById(R.id.calendar);
calendar.setShowWeekNumber(false);
calendar.setFirstDayOfWeek(2);

calendar.setOnDateChangeListener(new
CalendarView.OnDateChangeListener() {
    @Override
    public void onSelectedDayChange(CalendarView view, int year, int
month, int day) { … }
```

Como se aprecia, el contenido del evento lo hemos dejado en blanco para analizarlo más adelante ya que corresponde en gran medida con el que podría ser el último punto de esta fase, el tratamiento de los datos leídos para mostrarlos al usuario.

Vamos a ver ahora cómo es todo el proceso de lectura de datos del servidor de Google. Al igual que en el proceso de la inserción, el proceso de lectura, readData, también irá en una tarea asíncrona, AsyncTask, por el mismo motivo que el caso anterior.

Primero seleccionaremos el rango de horas a buscar, como esta función se activa cuando se selecciona un día automáticamente, el rango de horas lo estableceremos mediante un objeto Calendar entre las 00:00:00 y las 23:59:59 del día.

A posteriori construiremos el objeto DataReadRequest que nos servirá para la lectura de los datos en el cual indicamos el rango de la hora de inicio y de fin, y donde leeremos las actividades agregadas manualmente mediante el tipo *AGGREGATE_ACTIVITY_SUMMARY*, el cual también engloba los otros datos introducidos ya que van asociados a la actividad pero los tendremos que sacar más adelante. La construcción quedará así:

```
readRequest= new DataReadRequest.Builder()
       .aggregate(DataType.TYPE_ACTIVITY_SEGMENT,
DataType.AGGREGATE_ACTIVITY_SUMMARY)
        .bucketByTime(1, TimeUnit.DAYS)
        .setTimeRange(startTime, endTime, TimeUnit.MILLISECONDS)
        .build();
```

Una vez tengamos el dataReadRequest, leeremos los datos mediante el readData del DataReadResult y de este objeto obtendremos los dataSets asociados mediante la función printData. Para cada DataSet encontrado obtendremos los DataPoint, mediante dumpDataSet, almacenaremos sus valores en un listado de claves y valores, Map<String,String> *list_act*, y los tiempos asociados a cada dataPoint, *list_time*. En este caso, con lo llevado hasta ahora, solo sacaríamos los datos asociados a la actividad y su tiempo de realización por lo que ya tenemos 2 valores. Las funciones printData y dumpDataSet son las siguientes:

```
public void printData(DataReadResult dataReadResult) {
    Log.i(MainActivity.TAG, "Imprimiendo ");
    DateFormat dateFormat = getTimeInstance();

    if (dataReadResult.getBuckets().size() > 0) {
        Log.i(MainActivity.TAG, "Number of returned buckets of
DataSets is: " + dataReadResult.getBuckets().size());
        for (Bucket bucket : dataReadResult.getBuckets()) {
            List<DataSet> dataSets = bucket.getDataSets();
            Log.i(MainActivity.TAG, " Time bucket: " +
dateFormat.format(bucket.getStartTime(TimeUnit.SECONDS)));
            for (DataSet dataSet : dataSets) {
                dumpDataSet(dataSet);
            }
        }
    }
}
```

A continuación repetiremos este proceso para los 2 datos faltantes, calorías y distancia, con la particularidad que el dataReadRequest tendrá asociada la hora de inicio y fin de la actividad que ya hemos obtenido la cual estará almacenada en el objeto *list_time*. Esto estará en un bucle ya que se recorrerá tantas veces como actividades haya habido en el día seleccionado.

Ya para finalizar este punto nos queda coger los datos del *list_act* y mostrarlos al usuario. Para ello usaremos un objeto AlertBuilder donde estableceremos como título el día seleccionado. Para mostrar las actividades recorreremos el objeto en función de las actividades, es decir, el tamaño del objeto lo dividiremos entre 4 campos en total a mostrar para saber el total de actividades. En el objeto se almacena primero cada actividad y su duración y a continuación la duración y las calorías según cada actividad.

Recordemos que en Google almacenamos las calorías en kilocalorías pero para el usuario introducimos y mostramos calorías, por lo que realizamos la conversión a calorías. Para aclarar el tema de los índices lo vemos a continuación con un ejemplo de 2 actividades:

- ACTIVIDAD1
- DURACIÓN1
- ACTIVIDAD2
- DURACIÓN2
- CALORÍA1
- DISTANCIA1
- CALORÍA2
- DISTANCIA2

Estos datos están almacenados en un Array que, para cada actividad, debemos seleccionar sus datos correspondientes sin llegar a cruzarlos entre ellos. Por lo que en el ejemplo para la actividad1 (i), cogeríamos la duración1 (i+1), la caloría1 (i+1+((tamaño array dividido entre 4)*2-1)) y la duración1 (i+2+((tamaño array dividido entre 4)*2-1)).

Con esto conseguimos leer una actividad y formatear sus datos, por lo que posteriormente pasaremos a la siguiente fase.

Select Food

Requisitos de implementación

En esta fase procederemos a mostrar una recomendación de alimentos a tomar en función de las calorías quemadas. Dicha recomendación se mostrará por cada actividad realizada e indicará la cantidad de alimento a ingerir, concretamente serán 100 g y opcionalmente consultar, según las horas donde se ha realizado la actividad, una recomendación para las comidas, siendo éstas almuerzo y cena.

El primer requisito es la completa funcionalidad de la fase anterior ya que emplearemos los datos de esta fase para el cálculo de alimentos según las calorías y dicha recomendación se mostrará a continuación de los datos de cada actividad. Es decir, en el cuadro de texto que se le presenta al usuario con los datos de actividad, duración, distancia y calorías añadiremos una línea más correspondiente a la presente recomendación.

Al igual que en la fase previa, para mostrar esta recomendación será obligatorio que en el día seleccionado haya ocurrido al menos una actividad, en caso contrario, como antes, la ventana que se mostrará al usuario aparecerá vacía únicamente con la fecha del día seleccionado.

Para elaborar dicha recomendación, elaboraremos un fichero con un listado de alimentos categorizados por el número de calorías equivalentes a la consumición de 100 gramos de estos. Este fichero será pues un fichero de texto plano donde cada línea corresponderá a las centenas correspondientes de calorías de los alimentos citados a continuación. Por ejemplo:

- 000-leche, yogur, verduras, café.
- 100-castañas, queso fresco, huevos.

Aquí vemos en primera instancia los alimentos que tendrán 0 centenas de calorías y a continuación los alimentos que tendrán una centena de calorías. Todo esto con la condición de 100 gramos de alimentos. Por lo que entre cada línea se establece un intervalo de calorías: leche, yogur, verduras y café tienen entre 0 y 100 calorías por 100 gramos de alimentos.

Del mismo modo elaboraremos otro fichero para mostrar la segunda recomendación relacionada con las comidas. Dicha recomendación será solicitada mediante un enlace mostrado en la primera recomendación que nos abrirá un nuevo mensaje con las comidas.

Esta fase no requerirá, a diferencia de sus predecesoras, de una nueva activity y su desarrollo se incluirá como una función en la activity ViewActivity.java. Esto nos hace ver que tampoco tendremos que desarrollar nada nuevo gráficamente ya que usaremos los elementos desarrollados en la fase anterior.

Desarrollo

Tal y como hemos comentado, en esta fase no habrá ningún aspecto gráfico nuevo ya que emplearemos la interfaz presentada en la fase de Read Data. Este desarrollo se centra en un par de funciones implementadas en ViewActivity.

La primera función se denominará getFood y recibirá como parámetro un número de kilocalorías, a partir de los cuales devolverá una cadena con el listado de alimentos. Esta función será llamada cuando estemos formateando el mensaje a mostrar al usuario referente a los datos de las actividades. Por lo que en cada iteración del bucle que vimos para esta labor añadiremos a continuación de las calorías la llamada a la función, quedando como vemos a continuación:

```
mensaje += "Calorías:  " +
Float.parseFloat(list_act.get(i+1+((list_act.size()/4)*2-
1)).get("calories"))*1000 + "\n
"+getFood(list_act.get(i+1+((list_act.size()/4)*2-
1)).get("calories"));
```

Como hemos comentado los alimentos los obtendremos de un fichero de texto plano categorizado por el número de calorías. Primero convertiremos la cadena de calorías a un tipo decimal y según el valor de éste obtendremos la centena asociada que emplearemos para la búsqueda en el fichero. [23]

Posteriormente, y siguiendo el proceso empleado para el elemento Spinner en la fase Write Data, procederemos a abrir dicho fichero y buscar el candidato que corresponda con el valor obtenido anteriormente. Recordamos que la estructura de cada línea del fichero está formada por el valor numérico, formado por 3 cifras, un guión y el listado de alimentos. Por lo que iremos leyendo cada línea del fichero convirtiendo el valor numérico a decimal para compararlo con el obtenido. Una vez se cumpla dicha condición, devolveremos la cadena formada por una introducción, "Alimentos sugeridos (100gr)", y un alimento elegido de manera aleatoria perteneciente a la línea en cuestión. Dicha función, completa, es la siguiente:

```
public String getFood(String kcal){
    Float 55olítica = Float.parseFloat(kcal)*1000;
    float valor=0;

    if(100<=55olítica && 55olítica<200){
        valor=100;
    }else if(200<=55olítica && 55olítica<300){
        valor=200;
```

```
    }else if(300<=56olítica && 56olítica<400){
        valor=300;
    }else if(400<=56olítica && 56olítica<500){
        valor=400;
    }else if(500<=56olítica && 56olítica<600){
        valor=500;
    }else if(600<=56olítica && 56olítica<700){
        valor=600;
    }else if(700<=56olítica && 56olítica<800){
        valor=700;
    }else if(800<=56olítica && 56olítica<900){
        valor=800;
    }else if(900<=56olítica){
        valor=900;
    }

      Random r = new Random();
      Integer rand = 0;
      if(valor < 400){
          rand = r.nextInt(9);
      }else if (valor > 300 && valor < 700){
          rand = r.nextInt(5);
      }else if(valor == 700){
          rand = r.nextInt(3);
      }else {
          rand = 0;
      }

    InputStream fraw = getResources().openRawResource(R.raw.food);
    BufferedReader reader = null;

    try {
        reader = new BufferedReader(new InputStreamReader(fraw));
        String text = null;
        String result = "";

        while ((text = reader.readLine()) != null) {
            if(Float.parseFloat(text.substring(0, text.indexOf("-")))
== valor){
            StringTokenizer food = new StringTokenizer(text.substring(text.indexOf("-
")+1),",");

       for (int i=0;i<=rand-1;i++){
         food.nextToken();
       }
       result+=food.nextToken();

            }
        }
        return result;

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (reader != null) {
                reader.close();
            }
```

```
            } catch (IOException e) {
            }
        }
        return "0";
}
```

La segunda función en cuestión se llamara getFoods donde recibirá dos parámetros. Estos parámetros serán 2 valores numéricos que representarán el acumulado de las calorías quemadas antes de las 15:00 y del resto, suponiendo que el usuario realiza solo ejercicios por la mañana y por la tarde. Del primer parámetro obtendremos la comida correspondiente al almuerzo y del segundo la perteneciente a la cena. En caso de que no se realicen actividades por la mañana o por la tarde, no se mostrará la comida correspondiente a esa parte del día y al igual que con la primera función, si no existen actividades a lo largo del día no se mostrará nada. [24]

Bien, con esto ya damos por finalizada esta fase donde obtenemos nuestra recomendación alimenticia por actividad y su posterior visualización al usuario.

See Diabetes Data

Requisitos de implementación

En esta fase estableceremos un objeto personalizado con datos extras del usuario que también podremos ver y modificar aunque este último punto lo trataremos en la siguiente fase. Con esto pretendemos obtener más información del usuario referente a su diabetes para su uso en siguientes funciones.

Las funcionalidades que aquí se expondrán se desarrollarán en todo el marco de la página principal, la descrita en la fase Home, por lo que emplearemos principalmente el fichero MainActivity.java y no requeriremos de una Activity nueva.

A nivel gráfico, en esta fase agregaremos un nuevo botón en el fichero content_main.xml el cual desplegará una ventana emergente con la información. Dicha ventana emergente no irá en el fichero xml ya que la crearemos dinámicamente en MainActivity.java al llamar a la función asociada al botón.

Como hemos comentado, en esta fase crearemos un objeto personalizado, denominado *custom*, para añadir a Google Fit, puesto que éste nos deja crear nuestros propios tipos de DataTypes. Este nuevo tipo de DataType lo podemos formar con distintos campos, sean propios o pudiendo emplear los campos predefinidos que nos proporciona Google Fit. Este objeto, custom, estará compuesto por los siguientes campos, los cuales vamos a definir:

- Calorías: número decimal.
- Email: cadena de texto.
- Teléfono: número decimal.
- Email auto: booleano.
- SMS auto: booleano.

Las calorías será un tipo Float y emplearemos el campo Calories que nos proporciona Google. Nos indicará el límite de calorías que podría quemar el usuario. El email será una dirección de correo electrónico correctamente formateada que emplearemos en la fase Alert Email y para el teléfono emplearemos el tipo Integer y además comprobaremos que su número sea de 9 cifras.

Los campos Email auto y SMS auto serán números enteros cuyo valor será 0 o 1. Dicho valor se empleará para marcar si el checkbox de esa opción está seleccionado o no.

Desarrollo

Como vemos en lo expuesto anteriormente, en esta fase no habrá ningún aspecto gráfico nuevo ya que emplearemos la interfaz presentada en la fase de Home.

Cabe destacar que en dicha interfaz, mediante el fichero content_main.xml, añadiremos un botón más, como los ya empleados en esta página, que llamará a la función data. Está función está desarrollada en la clase MainActivity.java y como resultado mostrará una ventana al usuario con la información de los datos diabéticos personales.

Dicha ventana será creada dinámicamente en la clase MainActivity.java al llamar al botón Datos establecido en el content_main.xml. Será del tipo AlertBuilder, como las mostradas anteriormente, pero a diferencia de las otras empleadas usaremos una plantilla xml, data.xml, para darle formato a la ventana con los siguientes elementos:

- ImageView: estableceremos un fondo azul a modo de cabecera.
- Límite calorías: aquí empleamos 2 elementos, un TextView para mostrar el título del campo EditText que hay a posterior, referidos al límite de calorías que se va a establecer.
- Email: de la misma manera que con el caso anterior pero esta vez para el campo email de contacto.
- Teléfono contacto: de manera similar a lo mostrado previamente pero aquí tratamos el campo relacionado con usuario al que contactar.
- Email auto: será un checkbox que nos indicará si está activado o no.
- SMS auto: mismo funcionamiento que Email auto.

Para emplear dicha plantilla en el AlertBuilder haremos uso de un objeto View donde cargaremos el fichero data.xml. A continuación estableceremos en los campos EditText los valores de calorías, peso, y teléfono e indicaremos que serán campos editables. Por último procederemos a la carga de la View en el AlertBuilder y estableceremos 2 botones, el positivo para llamar modificar los datos y el negativo para cerrar el diálogo.

Este objeto será de uso exclusivo para nuestra aplicación y será inicializado una vez tengamos el objeto *mClient* conectado a Google mediante la función createCustom().

Esta función será una hebra debido a que, nuevamente, el proceso de inserción del objeto a crear deberá estar sincronizado con Google. A diferencia de las empleadas, esta hebra no la estableceremos como una AsyncTask sino como una hebra Runnable. Aquí deberemos indicar el nombre del objeto y los campos, y sus tipos, asociados a éste. Los tipos pueden ser los básicos que empleamos (Integer, String, etc.) o incluso podemos usar los tipos definidos por Google. El nombre del

objeto deberá estar formado obligatoriamente por el nombre del paquete más el nombre que se le vaya a dar al objeto. Ya tendremos el objeto creado y solo nos quedará almacenarlo en Google e incluso devolverlo una vez creado para su posterior uso, mediante un DataType. Dicho proceso se muestra a continuación:

```
DataTypeCreateRequest request = new DataTypeCreateRequest.Builder()
        .setName("controldiabetes.jesus.controldiabetes.custom")
        .addField(Field.FIELD_CALORIES)
        .addField("Email Contacto", Field.FORMAT_STRING)
        .addField("Telefono Contacto", Field.FORMAT_INT32)
        .addField("Email Auto",Field.FORMAT_INT32)
        .addField("SMS Auto", Field.FORMAT_INT32)
        .build();

PendingResult<DataTypeResult> pendingResult =
Fitness.ConfigApi.createCustomDataType(mClient, request);

pendingResult.setResultCallback(
        new ResultCallback<DataTypeResult>() {
            @Override
            public void onResult(DataTypeResult dataTypeResult) {
                // Retrieve the created data type
                custom = dataTypeResult.getDataType();
            }
        }
```

Antes hemos comentado que en los campos EditText estableceremos los valores correspondientes al límite de calorías, email y teléfono (variables c, e y t). Pues bien, estos valores serán leídos del objeto custom que hemos creado mediante una búsqueda en Google Fit de los últimos datos introducidos el último mes, donde siempre nos quedaremos con los últimos valores. Si en el último mes no hemos encontrado valores, por defecto los campos aparecerán vacíos.

La lectura de los valores del objeto custom seguirá exactamente el mismo procedimiento que el visto en la sección ViewData donde recopilamos los valores de la actividad. Esto quiere decir que nuevamente recurriremos a la AsyncTask debido al tema de la sincronización con Google. A diferencia del caso anterior, en esta ocasión leeremos el objeto custom, DataType, que hemos recogido anteriormente. Dicha lectura procede de la siguiente manera:

```
        final DataReadRequest readRequest = new
DataReadRequest.Builder()
        .read(custom)
        .setTimeRange(startTime, endTime, TimeUnit.MILLISECONDS)
        .build();
```

El resto de la lectura se hace de la misma manera que en el otro caso salvo que esta vez del DataSet asociado obtendremos los valores para las unidades de límite de calorías, email y teléfono.

Ya con esto mostramos los datos en el AlertBuilder y damos por finalizada esta fase. A continuación procedemos a ver la función con la que podremos actualizar estos datos mediante el botón “Modificar” que mostramos en el AlertBuilder desplegado.

Modify Diabetes Data

Requisitos de implementación

Esta fase está estrictamente relacionada con la fase anterior y sin ésta, tal y como está planteado, no podría ser posible.

El resultado final de la fase anterior, la ventana emergente, nos da la posibilidad de llevar a cabo esta fase, la modificación de los datos. Gráficamente, emplearemos esta misma ventana para modificar los datos que deseemos, recordamos que se muestran los últimos datos del último mes en caso de existir.

Cabe destacar que Google Fit, a partir de la versión 9 nos permite la acción de actualizar o modificar datos tal y como los conocemos. Las modificaciones solo las podremos realizar mediante una de las API mencionadas, la History API que es la que empleamos en los casos de lectura y escritura. Entonces la política a llevar a cabo en este proceso de actualización será la de insertar un nuevo objeto custom cuando no exista un objeto custom en los últimos 30 días y en caso de que exista procederemos a emplear la actualización que nos ofrece Google Fit. Esto quiere decir que podremos modificar los 3 campos vistos o sólo uno que si seleccionamos la opción de modificar, los valores serán insertados o actualizados en Google Fit.

Desarrollo

Tal y como comentamos en el capítulo anterior, aquí veremos cómo se desarrolla la función de “Modificar” que se muestra en el segundo botón mostrado en el cuadro de diálogo desplegado para ver los datos del usuario. Recordemos que estos son el límite de calorías, un email y teléfono de contacto.

Volviendo al último párrafo de los requisitos, establecimos que la actualización de los datos como tal será una nueva inserción en Google Fit en caso de que no exista un objeto en los últimos 30 días. Dicha inserción será de una manera similar a la vista en el capítulo Write Activity salvo que en este caso la diferencia es que agregaremos el objeto custom con los datos ya descritos.

Dicho procedimiento constará de una primera parte de recolección de los datos mostrados en los campos de texto. Una segunda parte tratará de una comprobación de datos introducidos, dicha comprobación será:

- Email: deberá estar debidamente formateado comprobando que tiene el símbolo @ en la cadena introducida y que no dispone de un ‘.’, a continuación de este símbolo (el dominio del correo)

- Teléfono: al no especificar y dar libertad al usuario de qué tipo de teléfono hablamos (móvil o fijo), se comprobará que la longitud de dicho número sea exactamente de los 9 dígitos que todos conocemos.

Una vez pasada esta comprobación, procederemos a insertar estos valores recogidos en nuestro objeto custom. Al igual que en el caso de Write Activity, deberemos crear, por este orden, un DataSource asociado a nuestro objeto custom, un DataSet a partir del DataSource y un DataPoint asociado al DataSet donde introduciremos los valores. Dicho procedimiento queda de la siguiente manera.

```
DataSource cnt = new DataSource.Builder()
        .setAppPackageName(this)
        .setDataType(custom)
        .setType(DataSource.TYPE_RAW).build();
calDataSet = DataSet.create(cnt);

DataPoint calDataPoint =
DataPoint.create(cnt).setTimeInterval(startTime, endTime,
TimeUnit.MILLISECONDS);


calDataPoint.getValue(custom.getFields().get(0)).setFloat(calories);
calDataPoint.getValue(custom.getFields().get(1)).setString(mail);
calDataPoint.getValue(custom.getFields().get(2)).setInt(tel);
calDataPoint.getValue(custom.getFields().get(3)).setString(mail);
calDataPoint.getValue(custom.getFields().get(4)).setInt(tel);


calDataSet.add(calDataPoint);
```

Una vez tengamos el DataSet, el proceso de inserción en Google Fit del objeto custom seguirá la misma estructura que en el caso de Write Activity. Esto es, en nuestra tarea asíncrona debido al procesamiento de datos en internet, llamar a la función insertData con nuestro conector de Google Fit, mClient, y el DataSet que acabamos de crear con el objeto custom.

En el caso de que exista un objeto, comprobado previamente en la lectura de los datos en los últimos 30 días, procederemos a modificar los datos que visualicemos por pantalla. Para controlar este hecho únicamente lo haremos mediante una variable booleana.

El proceso de actualización en sí, por parte de Google Fit, consiste en eliminar de los servidores cualquier DataPoint existente y reemplazarlo por el nuevo DataPoint. Por lo tanto, el proceso es exactamente igual que en el caso de la inserción, deberemos crear un DataSet con los datos y seguidamente crear el objeto para actualizar y llamar al método que nos ofrece Google de la siguiente manera:

```
DataUpdateRequest request = new DataUpdateRequest.Builder()
        .setDataSet(d)
.setTimeInterval(d.getDataPoints().get(0).getStartTime(TimeUnit.MILLIS
ECONDS), d.getDataPoints().get(0).getStartTime(TimeUnit.MILLISECONDS),
```

```
TimeUnit.MILLISECONDS)
        .build();

com.google.android.gms.common.api.Status updateStatus =
        Fitness.HistoryApi.updateData(mClient, request)
.await(1, TimeUnit.MINUTES);
```

Cuando es la primera vez que se inicia la aplicación y aún no exista el objeto custom porque el usuario no lo ha establecido, el procedimiento será similar salvo que llamaremos a insertData en vez de updateData.

Con esto finalizamos todo el proceso relacionado con un objeto personalizable para Google Fit, tanto la lectura en el capítulo anterior como la escritura y actualización en este capítulo. Proseguiremos viendo el motivo por el que vamos a usar este objeto en nuestra aplicación.

Alert User

Requisitos de implementación

En esta fase, y mediante el uso de los valores almacenados en el objeto custom que hemos visto en capítulos anteriores, pretendemos que la aplicación pueda avisar a un contacto de su confianza en caso de que supere un límite de calorías, el establecido en el objeto custom.

Dicho aviso podrá ser el envío automático de un email o de un sms, en caso de seleccionar alguna de estas opciones, donde se avisará de que el usuario que está interactuando con la aplicación, ha superado el límite establecido pudiendo producirse algún riesgo para su salud y se le indicará tanto el límite como el total de calorías en el email. La otra opción de aviso es mediante una llamada telefónica al contacto.

Este aviso, que se mostrará al usuario en caso de cumplirse la condición comentada, estará relacionado con la fase Write Activity ya que cogeremos las calorías establecidas en el objeto custom para la comparación con las calorías acumuladas a lo largo del día más las que se vayan a insertar. Por lo que se desarrollara en la clase Write Activity.

En este caso sí que necesitaremos conexión a Internet obligatoriamente que, a diferencia de los casos anteriores, solo será usada para el envío del email ya que los datos que necesitaremos los cogeremos de la misma aplicación, sin necesidad de conexión con los servidores de Google Fit y ni del objeto mClient. Para el envío del SMS únicamente necesitaremos añadir al AndroidManifest dicho permiso.

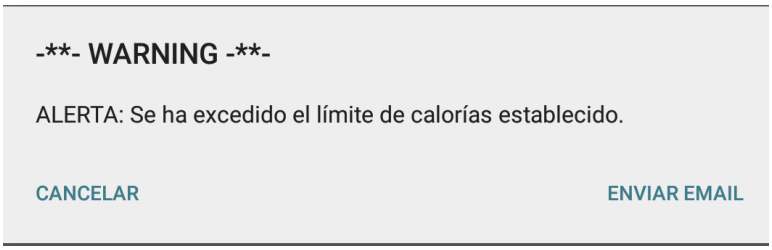
Para el caso del teléfono se hará uso del diálogo telefónico que nos proporciona el sistema de Android, únicamente se desplegará la pantalla con el número al que se va a llamar dejando al usuario la elección de realizar o no la llamada.

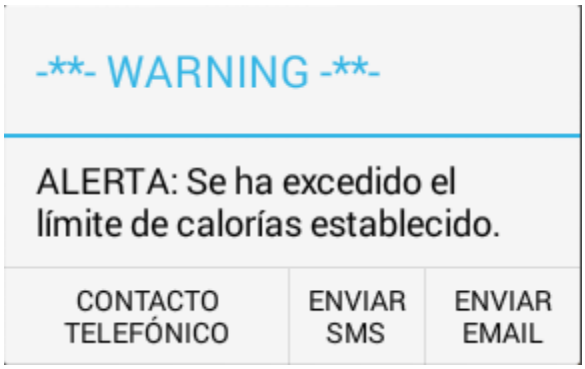
Desarrollo

Todo el desarrollo de esta parte se centrará en el archivo AddFit_Activity.java, correspondiente al capítulo Write Activity, dentro de la cual implementaremos una nueva clase, que será la encargada de mandar el correo electrónico en este caso, o una llamada al Activity interna de Android encargada de mostrar la pantalla telefónica con el número que le indiquemos.

Comenzaremos, tal y como hemos dicho en los requisitos, con una comparación entre las calorías que se van a insertar más las acumuladas a lo largo del día y el límite establecido en el objeto personalizado. Para obtener el acumulado, emplearemos, de igual manera que en el solapamiento de actividades, una lectura

previa de los datos calóricos de Google Fit. Cuando dicha suma es mayor al límite procederemos a mostrar una ventana con un aviso al usuario indicando que existe un riesgo y la opción que desea realizar. La ventana será un AlertBuilder como los vistos anteriormente solo que en este caso tendrá 3 botones o 2 en caso de que el dispositivo no posea red móvil (tarjeta SIM), uno para enviar email, otro para el teléfono y otro para sms, que corresponderán a los botones definidos como positivo, negativo y neutral. Para los dispositivos sin SIM suprimiremos el botón de sms y el de contacto telefónico. El resultado será el siguiente para los dispositivos sin y con tarjeta SIM respectivamente:



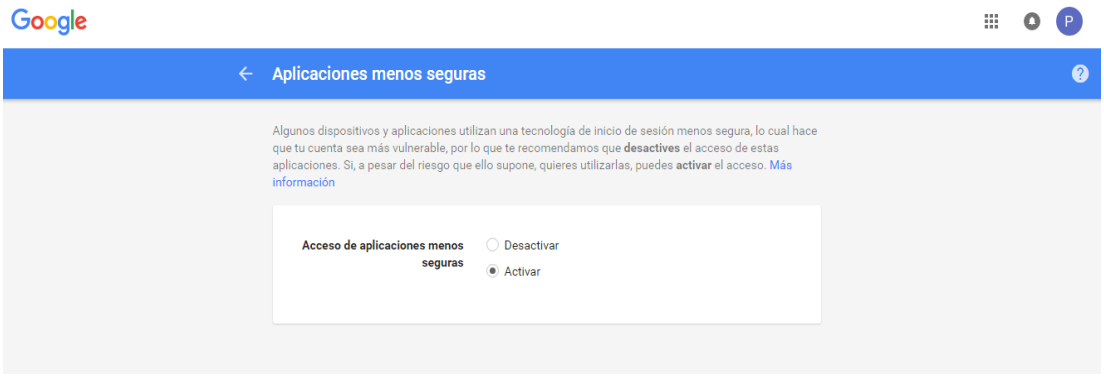


El botón cancelar, en dispositivos sin SIM, nos cerrará la ventana y omitirá cualquier acción. El siguiente botón, contacto telefónico será el encargado de desplegar la Activity correspondiente a la llamada telefónica y pasarle el número almacenado en el objeto custom. Para ello, emplearemos el Intent asociado a esta función, Action_Dial, asociaremos el teléfono a este objeto y lanzaremos la Activity asociada a él de la siguiente manera:

```
Intent intent = new Intent(Intent.ACTION_DIAL);
intent.setData(Uri.parse("tel:" + MainActivity.tel.toString()));
startActivity(intent);
```

Y, por último, nos queda el envío del correo electrónico automático. Primeramente, y antes de proceder con el desarrollo del código, tenemos que autorizar la cuenta de correo desde la cual se vaya a enviar el correo que permita dicha acción por parte de nuestra aplicación, considerada actualmente como aplicación no segura. Nosotros vamos a emplear la cuenta pruebadiabetes1@gmail.com para

realizar dichos envíos. Para aplicar dicha autorización lo haremos a través de la siguiente pantalla en la configuración de la cuenta:



Ahora ya nos centramos en el desarrollo de dicho envío. Para ello emplearemos una librería para Java denominada JavaMail donde existe una versión adaptada para los sistemas Android. El motivo de uso de esta librería es que el servicio de mail nativo de Android no nos da la posibilidad de enviar un email automáticamente. Esta librería a su vez necesita obligatoriamente de 2 librerías más que Google nos ofrece en forma de .jar que son activation.jar y additional.jar. Para agregar estos 3 módulos al proyecto, mediante Android Studio crearemos un nuevo módulo por cada uno que será directamente importado desde los jar. Una vez finalizado este paso, importaremos la clase JavaMail a la clase AddFit_Activity.java mediante la instrucción *import javax.mail.**. [25] [26]

A continuación vamos a crear una clase Mail que empleará estas librerías para lograr nuestra meta. Por lo que vamos a ver algunos de los detalles más relevantes de esta clase.

El primero de todos, y de manera obligatoria, será la configuración del servidor smtp para poder mandar los correos. Como ya hemos comentado el usuario encargado de esta labor será el usuario pruebadiabetes1@gmail.com, al que le hemos autorizado su uso por parte de la aplicación. Para esta configuración emplearemos los servidores de Google, Gmail en este caso, donde el puerto será el 587 y el servidor será smtp.gmail.com. Además también habilitaremos la opción de autenticación requerida y activaremos también la opción de TLS. Para estos datos usaremos el objeto Properties que nos facilita la librería, dicha configuración queda así.

```
Properties props = new Properties();
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", "smtp.gmail.com");
props.put("mail.smtp.port", "587");
```

Lo siguiente será ver cómo la clase procede a la elaboración del email y su posterior envió. En primer lugar, crearemos una sesión con las propiedades ya definidas y a partir de ésta crearemos un mensaje. A continuación estableceremos el emisor y el receptor del email y añadiremos el cuerpo al mensaje. Y por último, ya sólo nos queda mandar el mensaje que hemos preparado. Dicho proceso, al depender de Internet y al igual que los casos de GoogleFit, requerirá que se ejecute de manera paralela mediante una tarea asíncrona o una hebra. En nuestro caso, hemos optado por la segunda opción. Posteriormente podemos ver cómo se lleva a cabo esta parte:

```
javax.mail.Session ses = javax.mail.Session.getInstance(props, this);

final MimeMessage msg = new MimeMessage(ses);

msg.setFrom(new InternetAddress(_from));

InternetAddress[] addressTo = new InternetAddress[_to.length];
for (int i = 0; i < _to.length; i++) {
    addressTo[i] = new InternetAddress(_to[i]);
}
msg.setRecipients(MimeMessage.RecipientType.TO, addressTo);

msg.setSubject(_subject);
msg.setSentDate(new Date());

// setup message body
BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText(_body);
_multipart.addBodyPart(messageBodyPart);

// Put parts in message
msg.setContent(_multipart);

// send email
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            Transport.send(msg);
        } catch (MessagingException e1) {
            Log.i("MailSend", e1.toString() + " " + e1);
        }
    }
}).start();
```

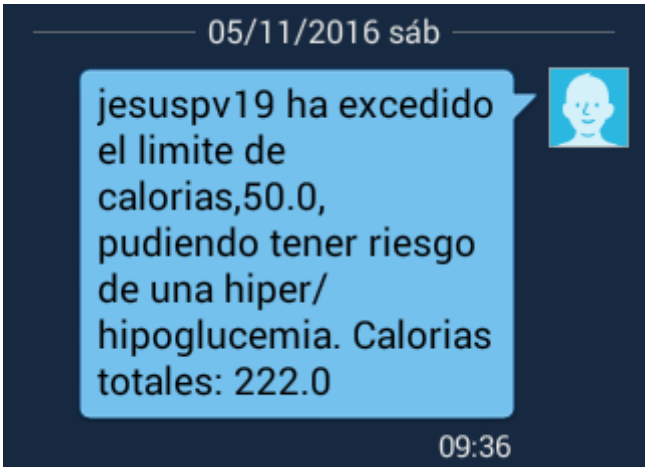
Por último nos queda la configuración de la acción al presionar dicha opción en el AlertBuilder. Aquí crearemos el objeto correspondiente a la clase que hemos descrito, denominada Mail, y asociaremos el contenido del mensaje y los destinatarios. El destinatario lo elegiremos del objeto custom de la clase MainActivity, y el emisor será el mismo usuario con el que lo hemos configurado, pruebadiabetes1. Estableceremos el asunto, donde usaremos al usuario logueado, y el cuerpo y llamaremos al método send() para que nos envié el mensaje.

```
Mail m = new Mail();

String[] toArr = {MainActivity.mail};
m.setTo(toArr);
m.setFrom("pruebadiabetes1@gmail.com");
m.setSubject("Alerta nivel de 69olítica usuario " +
MainActivity.user);
m.setBody("El usuario " + MainActivity.user + " ha excedido el límite
de calorías quemadas pudiendo tener riesgo de una
hiper/hipoglucemia");
m.send();
```

Como último caso de esta sección nos queda ver el botón de enviar SMS. Para ello únicamente emplearemos una clase SmsManager que nos proporciona Android para el envío de sms. Previamente comprobaremos si el dispositivo no dispone de tarjeta SIM para avisar al usuario que en este caso no podrá realizarlo. El contenido del sms será la misma cadena empleada para el email pero reducida para evitar que se envíen 2 sms por superar el límite de caracteres (como vemos en la imagen a continuación), y la función de envió quedaría de la siguiente manera:

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(phoneNo, null, message, null, null);
```



Estas 2 últimas funciones, si el usuario las ha activado como automática en el objeto custom, se enviarán sin necesidad de que aparezca el dialogo donde elegir la opción a realizar.

Y con esto acabamos la sección donde empleamos los datos del objeto personalizado, que hemos creado en el capítulo anterior, para proceder a avisar a un usuario de contacto si el usuario que está usando la aplicación así lo desea.

Bateria de pruebas

Elaboramos una serie de batería de pruebas para comprobar el correcto funcionamiento de la app. Dicha batería de pruebas estarán enfocadas en 2 vertientes: a nivel gráfico por un lado y a nivel global por otro lado. Para las pruebas a nivel gráfico emplearemos Robotium y para la otra vertiente más global emplearemos el conocido jUnit.

Robotium

Robotium es una librería que permite realizar pruebas funcionales o instrumentales de la interfaz visual del usuario sobre aplicaciones nativas e híbridas en Android. La dinámica de trabajo con este framework es la programación de acciones que simulen al usuario como por ejemplo realizar un clic o introducir un texto en una caja de entrada.

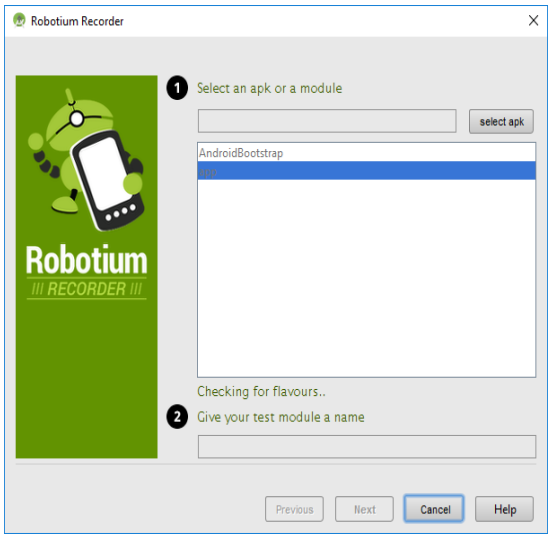
Este framework, por defecto, no viene incluido en Android Studio por lo que previamente procederemos a instalarla. Para ello, en Android Studio, nos iremos a File -> Settings -> Plugins y haremos clic en "Browse Repository". En la pantalla desplegada buscaremos "Robotium", seleccionaremos e instalaremos "Robotium Recorder" y una vez completado nos pedirá reiniciar el IDE.

Este plugin, además del framework, incluye un pulgin para generar nuestras propias pruebas sobre nuestra aplicación de forma automática con nuestro dispositivo conectado, en caso de no tener dispositivo físico empleará el emulador de Android.

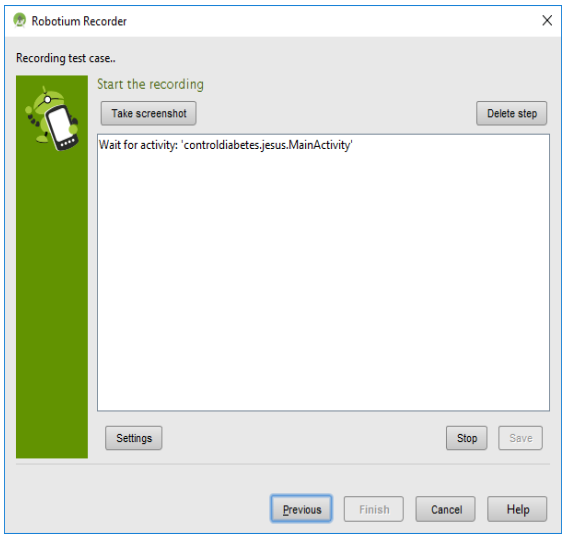
Para iniciarlo nos iremos a Tools -> Robotium Recorder y, a continuación, seleccionaremos nuestra app que aparecerá en el cuadro de texto o en caso de querer otra buscaremos su correspondiente apk. Junto a ello le daremos a Next y a New Robotium Test donde se lanzará la app en nuestro dispositivo e iremos interactuando con ella para guardar todas las interacciones como un test. Una vez deseemos finalizar la captura de interacciones le daremos a Stop y después a Save para que convierta nuestras interacciones en una clase java y realizar sobre ésta el test. [29]

Con esto ya tendremos generadas nuestra clase de pruebas sobre la UI de la aplicación en el directorio /java/controldiabetes.jesus.controldiabetes.test alojado en androidTest/. Ya solo nos queda realizar las pruebas, con el dispositivo conectado, seleccionando sobre la clase "Run *nombre de la clase*". Opcionalmente podemos hacer esto mismo sobre el directorio para lanzar todas las pruebas programadas.

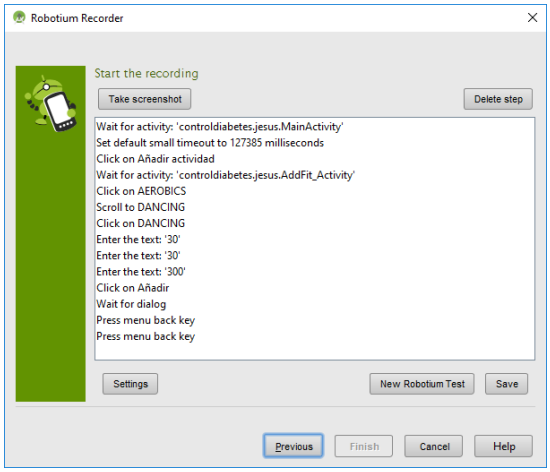
En las siguientes capturas mostraremos el proceso descrito con un caso de prueba de añadir una actividad física a Google.



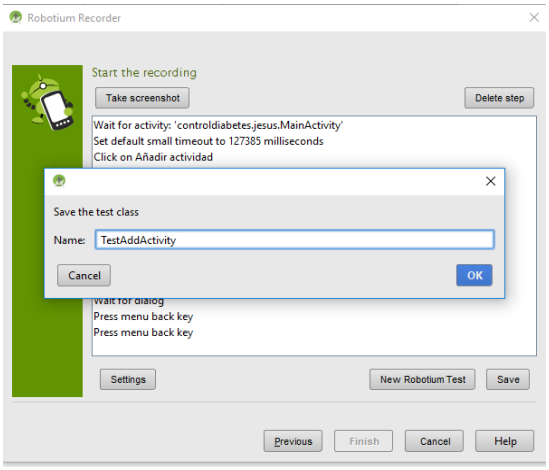
Seleccionar app.



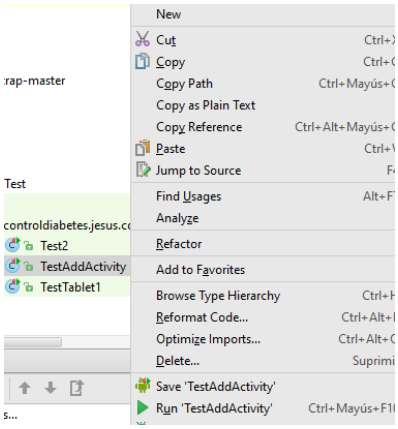
Iniciar captura.



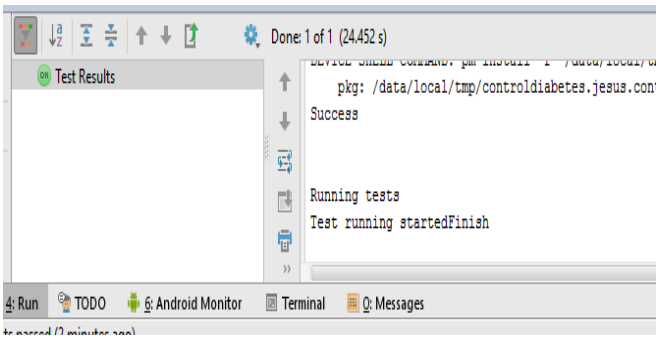
Parar captura.



Salvar captura.



Iniciar Test.



Resultado OK del test.

Esta prueba que hemos recogido con Robotium Recorder está enfocada a una prueba instrumental empleando las herramientas que nos ofrece Robotium. Concretamente, en la clase generada automáticamente, emplearemos una librería

denominada Solo que nos permitirá interactuar directamente con las Activity o Dialogs. Todos los movimientos a nivel gráfico que hemos realizado con la captura se generan a través de esta herramienta.

jUnit

jUnit es uno de los frameworks más populares para llevar a cabo tests de clases Java por lo que es igualmente válido para el entorno de Android. Vamos a ver algunos de los ejemplos empleados.

El primer ejemplo que vamos a ver es el de añadir actividad a Google Fit. Aquí, mediante asserts, validaremos el correcto formato de los campos que empleamos para añadir una actividad.

El primer elemento que nos encontramos es la lista desplegable con todas las actividades a elegir, por defecto tenemos la primera actividad seleccionada así que este elemento no es necesario validarlo ya que siempre tenemos una actividad válida. Los siguientes elementos que nos encontramos serán los campos a rellenar por parte del usuario que serán valores numéricos. El primero de estos, minutos, será un valor entero y el resto, metros y calorías, serán valores decimales por lo que nuestros valores a comprobar deberán ser positivos y no nulos. Para ello, emplearemos los siguientes asserts:

```
assertTrue("field empty", v1.getText().toString().length() >0);
assertTrue("num incorrect",
Integer.parseInt(v1.getText().toString())>0);
```

Donde v1 será el objeto BootstrapEditText correspondiente, en este caso, al valor introducido en el campo minutos. Repetiremos estas mismas instrucciones para los otros campos, v2 y v3, donde la diferencia residirá en que esta vez emplearemos tipo Float.

El siguiente ejemplo que vamos a ver es el correspondiente al capítulo de datos de configuración del usuario donde validaremos el correcto formato de los campos introducidos en el objeto custom que empleamos para modificar los datos o insertar unos nuevos en caso de primer uso.

Recordemos que esto es una ventana emergente que nos mostrará 3 campos de textos. Al igual que en anterior caso, primero comprobaremos si el campo es vacio. El primero de estos campos corresponde al límite de calorías que puede indicar el usuario el cual será de tipo Float y, al igual que en los casos anteriores, comprobaremos únicamente que sea mayor que 0.

El segundo campo a comprobar será el email introducido, para ello haremos comprobaremos que se introduce un correo válido. Dicha comprobación constará de dos partes: que incluye un '@' en su cadena y después de dicho carácter que exista un punto lo que nos indicará que es un dominio. En esta ocasión emplearemos un solo assert donde pondremos ambas condiciones unidas por el operador lógico and. El resultado es el siguiente:

```
assertTrue("field empty", email.getText().toString().length() >0);
   assertTrue("email incorrect",
email.getText().toString().contains("@")
   &&
email.getText().toString().substring(email.getText().toString().indexO
f("@")).contains("."));
```

El tercer y último caso es el del número de teléfono donde en este caso se comprobará la validez de un número de teléfono que únicamente se restringe a que el número sea de

```
assertTrue("field empty", telf.getText().toString().length() >0);
   assertTrue("num incorrect",
         Integer.parseInt(telf.getText().toString()) > 100000000 &&
Integer.parseInt(telf.getText().toString()) > 999999999);
```

Mejoras y conclusiones

Mejoras

Una vez dada por concluida la aplicación, vamos a redactar un listado de posibles mejoras definidas por iteración:

Select Food

Tal y como hemos visto, la selección de alimentos se realiza mediante lectura de ficheros de texto. La mejora de esta fase consiste en migrar estos datos a una base de datos SQLite [26] que iría integrada en la misma aplicación. Dicha base de datos, para este punto en concreto, constará de 2 tablas para las 2 funcionalidades que tenemos en esta iteración. Hablamos de la selección de alimentos por actividad y la selección de comidas por intervalos de horas. Las tablas podrán ser las siguientes y su posible diagrama E/R se muestra en la figura 23:

- Tabla alimentos: estará enfocada a la selección de alimentos donde almacenaremos los alimentos y sus calorías equivalentes. Cada registro de la tabla estará formado por el valor calórico (decimal) correspondiente y el alimento (varchar) asociado a éste de tal manera que las consultas a realizar por nuestra aplicación nos devolverían varios registros filtrados por este valor calórico. Luego elegiríamos uno de estos registros al azar.
- Tabla Comidas: en ésta almacenaremos 3 valores, el primero de estos, será el alimento (number) que se va a registrar (clave foránea de la tabla de alimentos), el siguiente será un valor booleano que nos indicará si ese alimento pertenece a un almuerzo o cena y el último indicará el tipo que representarán en el menú. Esto tipo de alimento sigue la lógica de entrante, plato fuerte o postre y, podrá definirse como un varchar o como un number (en caso de asociar el tipo de alimento mediante un ID que emplee el desarrollador).

Figura 23. Diagrama E/R base de datos

Write Activity

En este apartado comentaremos 2 posibles mejoras acerca de cómo poder introducir los datos de Google Fit además de la empleada en la elaboración del proyecto.

La primera de ellas es cambiar la API que empleamos, History API, para la introducción de datos. La otra variante para ello es usar Sessions API, y también Record API que la misma Sessions nos obliga a usar. Realmente no es una mejora en sí de la aplicación, únicamente proponemos un cambio en la manera de insertar los datos en Google.

Las sesiones en Google Fit representan un intervalo de tiempo durante el cual el usuario lleva a cabo una actividad como correr, ciclismo o un partido. Consisten en un tiempo de inicio y un tiempo de fin, un nombre de usuario, una descripción, un tipo de actividad y un identificador único. Las sesiones ayudan a organizar las actividades fácilmente agregando el intervalo de duración de los datos fitness al repositorio de Google. No es necesario implementar nuestros propios esquemas de almacenamiento de los datos y podremos mostrar las sesiones junto con la aplicación que cree cada sesión (pudiendo leer sesiones de otras aplicaciones). La Sessions API permite:

- Crear sesiones en tiempo real.
- Insertar sesiones en el repositorio fitness.
- Insertar segmentos de actividad que permiten pausas durante el ejercicio.
- Leer sesiones y datasets asociados a ésta.
- Iniciar otra aplicación para mostrar los datos de la sesión.
- Recibir un Intent cuando cualquier aplicación comienza una sesión.

Gracias a estas características y viendo el modelo actual de inclusión de datos nos da la ventaja de que automáticamente reconoce el tiempo de inicio y fin por lo que ya suprimimos un campo de introducción de datos cara al usuario.

Como hemos comentado Sessions API requiere obligatoriamente del empleo de Recording API ya que nos obliga a crear una subscripción. De cara al programador suponen más de horas de trabajo ya que necesitamos más líneas de código al requerir obligatoriamente el uso de otra.

Recordemos que una subscripción está asociada a una aplicación Android y a un específico Data Source o Data Type. Éste será nuestro primer paso a realizar donde crearemos una suscripción empleando el método RecordingApi.subscribe.

A continuación ya estamos en disposición de crear la sesión. Crearemos un objeto Session y estableceremos el nombre de la sesión, el identificador, la descripción, el tiempo de inicio. Una vez tengamos el objeto creado procederemos a iniciar la sesión mediante el método startSession y cuando se desee se parará la sesión

mediante stopSession. Llegados a este punto en la sesión dispondremos del tiempo de inicio y tiempo de fin.

Previo a la inserción, en este momento, deberemos eliminar la subscripción creada mediante Record API con el método RecordingApi.unsubscribe

Procederemos con la inserción de la sesión en el repositorio, donde lo primero será crear un objeto SessionInsertRequest al cual luego asociaremos la sesión que acabamos de crear y concluir, y, por último, a los DataSets que deseemos. En el caso de nuestra aplicación serán los referentes a actividad, distancia y calorías. Ya creado el objeto solo nos quedará insertarlo mediante el método insertSession.

Para aplicar este procedimiento deberíamos cambiar la interfaz gráfica de la sección de Write Activity. La modificación de esta parte ofrece un gran abanico de posibilidades según la imaginación/diseño del programado, una de las posibles modificaciones sería la siguiente:

- Eliminar el campo que recoge la duración en minutos de la actividad ya que este valor lo tenemos ya en la sesión.
- Eliminar los campos referentes a distancia y calorías ya que al principio no sabremos estos valores.
- Eliminar el botón “Añadir” y agregar 2 botones que sean “Inicio de actividad” y “Fin de actividad”, por ejemplo.
- El botón “Inicio de actividad” crearía la sesión y la iniciaría. Aquí ya cogeremos la actividad para asociarla a la sesión.
- El botón “Fin de actividad” finalizaría la sesión, mostraría al usuario mediante una ventana los 2 campos restantes, distancia y calorías, donde el segundo de ellos podría ser autocalculado como lo hemos visto según el deporte y recogiendo los valores de inicio de actividad y fin de actividad para calcular los minutos y así aplicar las fórmulas para dicho fin. Una vez el usuario haya introducido los datos se procede a la inserción de la sesión y los DataSet recolectados.

La siguiente mejora consiste en unir nuestra aplicación con dispositivos externos. Dicha mejora está encuadrada en esta iteración debido a que lo que se pretende es la recolección de los datos a agregar a Google por parte de algún dispositivo externo y conseguir así una mayor automatización y menor interacción del usuario.

Para conectar nuestra aplicación con dispositivos externos a ella debemos usar otra API de las que nos proporciona Google Fit que, en este caso, es Sensors API. Esta API nos proporciona listar los DataSources disponibles, registrar/eliminar eventos para la lectura de datos. Sin embargo, no permite el almacenamiento de los datos en él,

para esto se suele combinar esta API con la vista anteriormente, Recording API, si se desea.

Primeramente podremos hacer una búsqueda de los distintos DataSource que disponemos para obtener un listado de los posibles que hay y seleccionar el que deseemos para la lectura de los datos.

A continuación tendremos que crear el evento que se encargará de la lectura de los datos. Dicho evento, Listener, lo crearemos y una vez creado lo asociaremos al dataSource seleccionado e indicándole obligatoriamente el DataType.

Hemos comentado que podríamos usar Recording API para almacenar los datos. Esto es asociar los datos a una subscripción de las que nos ofrece Google. Las otras opciones de almacenamiento serán las vistas en la aplicación, History API, o la comentada anteriormente, Sessions API.

A través del listener ya leemos todos los DataPoint que puede recoger el sensor que asociemos y proceder a almacenarlos en el repositorio de Google. Para ello, a partir del DataPoint creamos un DataSet asociado a este DataPoint y procedemos a agregarlos al repositorio en el caso del uso de History API o Sessions API.Con Recording API deberemos crear tantas subscripciones como datos empleemos, estos datos son el tipo de actividad, la distancia y el número de calorías ya que el tiempo va integrado en los DataPoints.

Proyecto

Aquí no nos vamos a referir a una iteración en concreto sino al proyecto en su totalidad ya que lo que vamos a proponer es otro enfoque de realizar el mismo proyecto con la misma funcionalidad.

Dicha propuesta consiste en una migración del proyecto entero desarrollado de manera nativa en Android y empleando las librerías de Google Fit para Android. La migración consta de cambiar las librerías de Google Fit nativas para Android por el uso de la API de Google Fit específica para REST. Esto significa cambiar las librerías de Android de Google Fit, en Java, por la comunicación directa al servidor de Google Fit mediante el uso de peticiones HTTP. [27]

Por lo que aquí, en lo referente a Google Fit en nuestra aplicación, emplearíamos alguna de las clases de Java para el envío de peticiones y recepción de respuestas HTTP, como por ejemplo podría ser HttpServletRequest y HttpServletResponse. La “particularidad” de estas peticiones es que, por un lado, mediante la consola de proyectos de Google, y de manera similar a la empleada en nuestra aplicación, deberemos obtener un cliente Oauth para nuestra aplicación que

en este caso será un token de seguridad que nos proporcionará Google y que irá incluido en el campo “Authorization” del Header del request. Por otro lado, el body de la petición, y respuesta cuando proceda, estará formado por un JSON donde se indicará toda la información de los DataPoints tanto para la lectura como la escritura en los repositorios de Google. [28]

Conclusiones

Uno de los principales motivos para la realización de este proyecto, ha sido la posibilidad de trabajar con el más popular sistema operativo para móviles.

La realización de este proyecto, me ha aportado conocimientos sobre varias tecnologías pero, sobre todo, me ha permitido profundizar en el entorno de programación para móviles con Android Studio, del cual no tenía ningún conocimiento previo y que es una versión oficial por parte de Android para los desarrolladores. El trabajo con esta IDE llega a resultar bastante cómodo para aquellas personas que conozcan o hayan usado entornos como Eclipse o NetBeans.

Respecto a la plataforma Android, el hecho de ser código abierto y llevar varios años consagrada facilita mucho la labor del desarrollador. Existen herramientas disponibles que cubren la totalidad de las necesidades para desarrollar aplicaciones competitivas y con un alto grado de independencia.

El otro gran reto de este proyecto ha sido, además de Android, la integración de una de las funciones de Google con solo unos pocos años de vida de la cual, a día de hoy y comparados con otras funciones más longevas, existe poco documentación extra además de la oficial de Google. Hablamos de Google Fit cuya función está basada en todo lo relacionado al mundo del deporte de cara al usuario, permitiendo monitorizaciones de ejercicios, tiempos, metas, distancias además de otros datos que pueden ayudar al usuario a llevar una vida saludable en cuanto al deporte se refiere. Para llevar a cabo esta integración Google nos ofrece una serie de APIs con las que hemos podido trabajar con los servidores de Google Fit intentando explotarlas lo máximo posible y consiguiendo unos buenos resultados.

Se han cumplido los objetivos marcados en el anteproyecto respecto al desarrollo de la aplicación ControlDiabetes. Hablamos de la escritura y lectura de los datos de Google Fit, el uso de tipos de datos personalizados que nos ofrece Google para mostrar alertas al usuario y la opción de contactar con una persona mediante teléfono o email además de la lectura y escritura de estos datos personalizados en Google, la selección de alimentos y la agrupación de estos para mostrar ayudas al usuario referente a los hábitos culinarios. Además de esto, también se han integrado otras librerías que nos permiten el envío de emails, que ya hemos comentado, o el

mejorar la apariencia visual de cara al usuario mediante una librería basada en Bootstrap.

El gran objetivo personal planteado en este proyecto ha consistido en aprender y trabajar para la plataforma Android, en la cual durante los años de estudio de la carrera no se ha trabajado en profundidad. Por este motivo, gracias a los conocimientos que se han ido adquiriendo en Java durante estos años y a un costoso inicio de aprendizaje se han podido conseguir los resultados reflejados en este proyecto así como el combinarlo con el uso de unas las APIs propias de Google (Google Fit), la cual es aún muy reciente siendo, como hemos dicho anteriormente, uno de mis objetivos personales para poder adaptarme a una tecnología de la que sólo se dispone de la documentación oficial.

Referencias y Bibliografías

Referencias

[1] https://www.idf.org/diabetesatlas/5e/es/que-es-la-diabetes 31/12/2015

[2] http://www.cdc.gov/diabetes/spanish/basics/diabetes.html 31/12/2015

[3] https://play.google.com/store/apps/details?id=com.google.android.apps.fitness&hl=es 17/01/2016

[4] http://www.openhandsetalliance.com/index.html 17/01/2016

[5] https://play.google.com/store 17/01/2016

[6] http://www.monografias.com/trabajos101/sistema-operativo-android/sistema-operativo-android.shtml 17/01/2016

[7] http://developer.android.com/intl/es/tools/studio/index.html 17/01/2016

[8] http://www.samsung.com/latin/support/skp/faq/1070744 26/01/2016

[9] http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-actividad 02/02/2016

[10] http://www.cnet.com/es/como-se-hace/la-guia-completa-a-google-fit/ 11/02/2016

[11] https://developers.google.com/fit/ 26/02/2016

[12] https://developers.google.com/fit/android/get-api-key 26/02/2016

[13] https://developers.google.com/fit/overview 26/02/2016

[14] https://developers.google.com/fit/android/ 28/02/2016

[15] https://developers.google.com/fit/android/sensors 21/07/2016

[16] https://developers.google.com/fit/android/record 21/07/2016

[17] https://developers.google.com/fit/android/using-sessions 21/07/2016

[18] https://developers.google.com/fit/android/history 21/07/2016

[19] https://developers.google.com/fit/android/ble-sensors 21/07/2016

[20] https://developers.google.com/android/reference/com/google/android/gms/fitness/ConfigApi 21/07/2016

[21] https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Rational+Team+Concert+for+Scrum+Projects/page/SCRUM+como+metodolog%C3%ADa 22/03/2016

[22] http://www.awesomeprojects.xyz/2015/10/how-to-use-android-bootstrap-twitter.html 05/08/2016

[23] http://es.calcuworld.com/deporte-y-ejercicio/calculadora-de-calorias-quemadas/ 20/03/2016

[24] http://javapapers.com/android/android-email-app-with-gmail-smtp-using-javamail/ 25/07/2016

[25] https://java.net/projects/javamail/pages/Android 25/07/2016

[26] https://www.sqlite.org/ 07/08/2016

[27] https://developers.google.com/fit/rest/ 09/08/2016

[28] https://developers.google.com/fit/rest/v1/get-started 09/08/2016

[29] http://robotium.com/ 10/10/2016

Bibliografía

- Francisco José Tejido López, César Cruz Arrieta y Evelyn Galindo Hernández, Anatomofisiolgía y patologías básicas, editorial McGraw-Hill, 2013.

- Jesús Tomás Gironés, El Gran libro de Android: [actualizado a versión 6.0 marshvallow y android studio], editorial Marcombo, 5º edición, 2016.

- Jorge Santiago Nolasco Valenzuela, Desarrollo de aplicaciones móviles con Android, editorial Ra-Ma, 2015

- Joan Ribas Lequerica, Manual imprescindible de desarrollo de aplicaciones para Android, editorial Anaya, 2015

- Salvador Gómez Oliver, Manual Programacion Android SgoliverNet v3, 2013