Proyecto Fin de Carrera Ingeniería de Telecomunicación

Autor: Ignacio González de la Peña Puerta Tutor: Juan Manuel Vozmediano Torres

> Departamento de Ingeniería Telemática Escuela Técnica Superior de Ingeniería Universidad de Sevilla

> > Sevilla, 2015

## GeoNudge: Aplicación de recordatorio de notas mediante GPS.

Tutor: Juan Manuel Vozmediano Torres Autor: Ignacio González de la Peña Puerta

Sevilla, Julio 2015

#### Índice

1	Resumen	5
	Abstract	6
2	Introducción	7
	2.1. Objetivo	7
	2.2. Motivación	7
	2.3. Antecedentes	8
	2.4. Arquitectura y funcionamiento	8
3	Entorno de trabajo	10
	3.1 Entorno de desarrollo	10
	3.2 Instalación del entorno de desarrollo	10
	3.3 Elementos de un proyecto Android	12
	3.4 Componentes de una aplicación Android	16
	3.5 Otras herramientas	18
	3.5.1. Generador y editor de iconos	18
	3.5.2. Guía de desarrollo de Android	19
	3.5.3. Sitios webs de consulta	20
	3.5.4. Google Developers Console	20
	3.5.5. KeyTool	21
4	Planificación de la aplicación.	23
	4.1. División en fases	23
	4.2. Fase I: Desarrollo de un bloc de notas y gestión de su base de datos.	23
	4.3. Fase II: Utilización y aplicación de la API de Google Maps en la situación de la nota almacenada.	25
	4.4. Fase III: Uso y gestión de las alertas por proximidad.	26
	4.5. Diagrama total.	28
	4.6. Planificación temporal	29
5	Estructura y desarrollo de la aplicación	31
	5.1. Estructura del programa.	31
	5.1.1 Carpeta /src/	31
	5.1.2 Carpeta /res/	32
	5.2. Descripción y funcionalidad de las clases.	33
	5.2.1. Pantalla principal	33
	5.2.2. Pantalla Mapa	35
	5.2.3. Pantalla de visualización de notas.	37
	5.2.4. Menú y pantallas de ajustes.	38

	5.2.4.1. Submenú ajustes.	41
	5.2.5. Clases secundarias.	45
	5.2.6. Clases auxiliares.	51
5	.3. Resumen estructural de la aplicación.	53
6	Problemas y soluciones	55
7	Plan de pruebas	59
7	1. Pruebas realizadas.	59
7	.2. Dispositivos empleados.	60
8	Conclusiones y líneas de continuación	61
8	.1. Conclusiones	61
8	2.2. Líneas de continuación.	62
9	Bibliografía	63
Ane	exo A: GeoNudge	64
Mar	nual de uso	64
A	Arrancando la aplicación	65
A	Agregando nota nueva.	66
A	Activando notificación.	67
D	Desactivando notificación.	67
E	Eliminando nota.	68
E	Eliminando todas las notas.	68
C	Configurando la aplicación: Submenú ajustes.	69
C	Configurando la aplicación: Localizaciones Preferidas.	69
H	laciendo uso de las localizaciones guardadas.	70
C	Configurando la aplicación: Radio de aviso.	70
C	Configurando la aplicación: Notificaciones.	71
R	Recibiendo una notificación.	71
R	Restricciones sobre su uso	72
Ane	exo B: Android Manifest	73

## **1** RESUMEN

En este documento se detalla el desarrollo y funcionamiento de la aplicación GeoNudge creada como proyecto fin de carrera.

En nuestros días, el uso del teléfono móvil Smartphone se ha convertido en una herramienta básica de comunicación y un instrumento clave para la resolución de las tareas cotidianas, es por ello por lo que es necesario que existan aplicaciones que nos faciliten todas estas tareas de forma sencilla.

El objetivo de este material es por un lado explicar la creación y evolución en el desarrollo de esta aplicación y por otro detallar el funcionamiento y manejabilidad de la misma.

This document details the development and operation of the GeoNudge mobile applicattion created as a degree final Project.

Today, smatphones are a basic tool of comunication and resolutions of daily tasks.For this reason, it's neccesary to create mobile applications to facilitate in a simple way all the task.

This material explain the creation process and evolution of the GeoNudge mobile applicattion development and explain all the instructions to be used.

n este apartado, se abordarán las motivaciones fundamentales para el desarrollo de este proyecto. Para ello primero comprenderemos el objetivo del mismo para después continuar realizando un análisis de la motivación que ha conducido al desarrollo de este proyecto finalizando con una breve explicación acerca de su arquitectura y funcionamiento.

#### 2.1. Objetivo

El objetivo de este proyecto es la realización de una aplicación para el sistema operativo de *smartphones* Android cuya utilidad es el almacenamiento y recordatorio de notas mediante posicionamiento GPS.

Para proporcionar dicho servicio la aplicación implementa los servicios de una agenda de notas combinada con un eficaz sistema de recordatorio. Los requisitos necesarios son un terminal móvil con sistema operativo Android y acceso a internet, aunque incluye cierta funcionalidad *offline* como se verá en apartados posteriores.

La aplicación por tanto, proporciona una doble funcionalidad por un lado servirá como bloc de notas pero a su vez ofrece la posibilidad de alertar al usuario por posicionamiento, esto es, el usuario introducirá una nota que desea almacenar y una ubicación donde quiere que se le recuerde dicha nota de forma que cuando el dispositivo detecte que se encuentra en una posición cercana a la ubicación fijada lanzará un aviso recordando al usuario el texto almacenado en la nota correspondiente.

#### 2.2. Motivación

El gran auge de la tecnología ha provocado que cada día que pasa estemos más acostumbrados a interactuar con máquinas y dispositivos que facilitan la realización de las distintas tareas a las que nos tenemos que enfrentar en nuestro día a día. El ritmo de vida actual es vertiginoso e implica una capacidad de adaptación para todas las circunstancias, situaciones y eventos a los que nos sometemos cada día.

Este ritmo de vida ha provocado que nuestra concentración en todas las tareas que desarrollamos sea mayor pero también ha hecho que tengamos que recordar multitud de cosas y eventos que en muchos casos solemos olvidar. Es este olvido o despiste de pequeñas cosas algo intrínseco en el ser humano y que propicia el uso de distintos elementos que nos recuerden todo aquello que no queremos olvidar.

Por ese motivo surgen aplicaciones para *smartphones* que nos permiten anotar las distintas cosas que queremos recordar pero suele ocurrir que tras apuntar algo para recordar, luego no consultamos nuestro *Smartphone* provocando nuevamente que no recordemos aquello que habíamos anotado.

Esta última premisa ha sido la motivación principal de la aplicación la cual se encarga, de manera sencilla, de recordar la nota introducida implementando una funcionalidad no

encontrada en las aplicaciones actuales como es la posibilidad de recordatorio mediante ubicación.

#### 2.3. Antecedentes

La evolución de las tecnologías de telecomunicación y concretamente el auge de la telefonía móvil en las últimas décadas ha propiciado que se proporcionen cada vez más servicios y herramientas que facilitan la vida diaria de los usuarios.

La telefonía móvil ha experimentado una evolución drástica desde la aparición de los primeros terminales que permitían exclusivamente realizar llamadas de voz. Esta evolución ha supuesto la introducción de más servicios además de la voz sin mermar en calidad, como ha sido la red de datos móvil que ha provocado la aparición y desarrollo de los teléfonos inteligentes o *smartphones*.

Actualmente el uso de estos dispositivos inteligentes, se ha convertido en una actividad cotidiana más en nuestra rutina diaria, incorporando cada vez más herramientas que facilitan nuestras tareas.

Las aplicaciones de bloc de notas, recordatorios y alarmas son utilidades básicas de gran popularidad en nuestros dispositivos existiendo un catálogo muy completo de aplicaciones que realizan estas tareas.

#### 2.4. Arquitectura y funcionamiento

La aplicación presenta una estructura basada en tres aspectos fundamentales: Almacenamiento en base de datos, gestión de la dirección y mapas y por último sistema de alertas.

El usuario introducirá una nota que se almacenará en una base de datos, si ha introducido una ubicación la aplicación conectará con el servidor *Google Maps* y localizará dicha ubicación. Por último, mediante posicionamiento GPS se analizará la posición actual del usuario y si procede se lanzará una alerta.

En el siguiente diagrama se observa la funcionalidad descrita.



Ilustración 2.1.Diagrama de funcionalidad de la aplicación haciendo uso de la utilidad de alertas.

## **3** ENTORNO DE TRABAJO

n este apartado se define el entorno de desarrollo empleado, así como su instalación y uso para posteriormente, realizar una breve introducción acerca de los componentes de una aplicación en relación con el entorno de desarrollo.

#### 3.1 Entorno de desarrollo

A continuación se describen las herramientas utilizada en lo respectivo al entorno de desarrollo de la aplicación.

#### 3.2 Instalación del entorno de desarrollo

La aplicación se ha desarrollado empleando el entorno de desarrollo Android Studio para el sistema operativo Windows.

Android Studio es un entorno de desarrollo integrado (IDE) para la plataforma Android, está disponible para desarrolladores para probarlo gratuitamente. Basado en IntelliJ IDEA de JetBrains, ha sido diseñado específicamente para desarrollar para Android. Las ventajas que ofrece Android Studio frente a otros entornos de desarrollo son renderización en tiempo real, consola de desarrollador(consejos de optimización, ayuda para la traducción, estadísticas de uso) y herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, y otros problemas, estas son las ventajas por las que se ha utilizado Android Studio, pese a estar en fase beta, y no otros entornos de desarrollo.

Para poder utilizar dicho entorno han de descargarse los siguientes paquetes:

• Java Development Kit (JDK):

En primer lugar habrá que descargar la última versión del JDK desde la web de Oracle, este software proporciona herramientas de desarrollo para la creación de programas en Java.

En este momento la versión disponible es la JDK 8, elegimos la versión y nuestro sistema operativo e iniciamos la descarga.

• Android Studio:

Para continuar con la instalación del entorno de desarrollo, se ha de descargar la última versión de Android Studio desde la página web de Android Developers.

Como se ha comentado anteriormente Android Studio contiene dentro del propio paquete varias funcionalidades como son Android Studio IDE, Android SDK tolos, Android 5.0 (Lollipop) Platform y Android 5.0 emulator system image with Google APIs. Además el entorno de desarrollo Android Studio cuenta con un sencillo asistente de configuración, permite importar ejemplos y plantillas. También permite ver, editar y pre visualizar los diseños de Android a través de múltiples tamaños de pantalla, idiomas e incluso versiones de API, así como analizar el rendimiento de las aplicaciones y muchas más herramientas que ayudarán a los desarrolladores a crear con mayor facilidad mejores aplicaciones para Android. • Paquetes SDK

Los paquetes SDK (Software Development Kit) no son más que un conjunto de herramientas de desarrollo de software que le permiten al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etc.

Es algo tan sencillo como una interfaz de programación de aplicaciones o API (del inglés *application programing interface*) creada para permitir el uso de un lenguaje de programación, o puede, también, incluir hardware sofisticado para comunicarse con un determinado sistema embebido. Las herramientas más comunes incluyen soporte para la detección de errores de programación como un entorno de desarrollo integrado o IDE (del *inglés Integrated Development Environment*) y otras utilidades. Los SDK frecuentemente incluyen, también, códigos de ejemplo y notas técnicas de soporte u otra documentación para ayudar a clarificar ciertos puntos del material de referencia primario.

Android Studio facilita esta tarea ya que incorpora en su barra de herramientas la herramienta SDK Manager, pinchando en ella nos aparecerá una ventana como la siguiente:

🐧 Android SDK Manager				
Packages Tools				
SDK Path: C:\Program Files (x86)\Android\android-sdk				
Packages				
👾 Name	API	Rev.	Status	*
▲ 🔽 🧰 Tools				
Android SDK Tools		16	🎒 Installed	
📝 🙀 Android SDK Platform-tools		10	Not installed	=
Android 4.0.3 (API 15)				
Documentation for Android SDK	15	1	Not installed	
👿 🌞 SDK Platform	15	2	🖊 Not installed	
📝 📥 Samples for SDK	15	1	🖊 Not installed	
👿 👾 ARM EABI v7a System Image	15	1	🖊 Not installed	
📝 🖏 Google APIs by Google Inc.	15	1	🖊 Not installed	
📝 🔝 Sources for Android SDK	15	1	🖊 Not installed	
[] 🔄 Android 4.0 (API 14)     []				
Image: Marce Marce Marce Marce Marcel (API 13)				
[]      [a] Android 3.1 (API 12)     [a]     [b]     [				
Martin Contraction (ADI 11)				-
Show: 🗹 Updates/New 📝 Installed 📃 Obsolete Select	New or Up	dates	Install 8 pac	kages
Sort by:  API level  Repository Desele	ect All		Delete 1 pag	:kage
Done loading packages.				

Ilustración 3.2. Pantalla de Android SDK Manager.

Como se puede observar, mediante selección podemos descargar los paquetes que necesitemos para desarrollar nuestra aplicación, en nuestro caso será necesario por ejemplo descargar la API de *Google play services* (entre otros paquetes) para poder utilizar los mapas que nos proporciona *Google Maps*.

Una vez descargado lo anterior, el entorno de desarrollo está listo para su uso. Al ejecutarlo observaremos esta pantalla principal:



Ilustración 3.3. Pantalla principal de Android Studio

#### 3.3 Elementos de un proyecto Android

Una vez disponemos en nuestro ordenador del entorno de desarrollo, hemos de conocer la estructura básica para iniciar un proyecto en Android.

Cuando generamos un nuevo proyecto Android en Android Studio, automáticamente se genera una estructura de carpetas que van a organizar nuestro proyecto. Dicha estructura es común para todos los proyectos independientemente del tipo o la complejidad del mismo.



Ilustración 3.4.Estructura básica de carpetas en un proyecto de Android Studio.

La carpeta principal dentro del proyecto será la que posee el nombre que se le ha asignado al proyecto, y dentro de ella, la carpeta src es la que contiene todos los archivos del código fuente de la aplicación, así como sus recursos. El resto de archivos y carpetas corresponden a archivos de configuración del proyecto, archivos de la herramienta Gradle (que realiza la compilación automatizada del proyecto), y archivos correspondientes al sistema de control de versiones *git*.

La carpeta src contiene 3 subcarpetas principales:

- java: Esta carpeta contiene los archivos de código fuente (.java) que se crean para dirigir el funcionamiento de la aplicación. Se organiza en paquetes de igual manera que las aplicaciones Java, e inicialmente se encuentra en ella el archivo correspondiente al código fuente de la *Activity* que se ha creado al generar el proyecto.
- res: Es la carpeta de recursos de la aplicación. En ella se mantendrán una serie de archivos en formato XML con los datos referentes a los recursos usados por la aplicación. Por ejemplo, cadenas de texto (*Strings*), dimensiones, colores, menús, elementos contenidos en las ventanas (*layouts*), etc. Además almacenará las imágenes que usará la aplicación. Esta carpeta se organiza en subcarpetas en función del contenido que almacena cada una, y a su vez se crearán dentro de ellas otras subcarpetas según el idioma, tamaño de la pantalla, y otras características. Esta organización permite utilizar distintos valores y elementos en función de las características del dispositivo en el que se esté ejecutando la aplicación.
- AndroidManifest.xml: Es un archivo de control que contiene información sobre las características generales de la aplicación y sus componentes.Describe algunas características sobre las *activities, services, intent receivers, y content providers* que va a utilizar la aplicación, los permisos que requiere la aplicación, las librerías

externas que va a necesitar, las características requeridas para los dispositivos, los niveles de la API que se soportan o son requeridos etc.

Tras comprender la estructura de carpetas del entorno de desarrollo, es necesario conocer también una herramienta de simulación incluida en Android Studio llamada AVD Manager. AVD Manager proporciona una interfaz gráfica en el que podremos configurar y ejecutar los distintos dispositivos Android (Android Virtual Devices) para posteriormente ejecutar sobre los mismos nuestro proyecto y así poder comprobar su funcionamiento y testear errores.

En la siguiente imagen se muestra la pantalla inicial que aparece al arrancar esta herramienta, se observa en ella los distintos dispositivos configurados con un breve resumen de sus características.

				AVD Manager			
	Your Virtual De	evices					
Туре	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Galaxy Nexus API 21	720 × 1280: xhdpi	21	Android 5.0	x86_64	650 MB	► / ×
	Nexus One API 21	720 × 1280: xhdpi	21	Android 5.0	x86_64	650 MB	► # <del>*</del>
+ Cr	eate Virtual Device						9

Ilustración 3.5. Pantalla principal de AVD Manager.

En la ilustración 3.5 se muestra la pantalla de configuración del dispositivo que aparece al crear un nuevo dispositivo virtual, observamos que aparecen los siguientes campos:

- AVD Name: Nombre del Dispositivo Virtual.
- Device: Es el campo que aparece en la segunda línea referente al dispositivo comercial que se quiere emular.
- Target: Es el campo que aparece en la tercera línea referente a la versión de Android que se quiere utilizar para la emulación.
- Startup size and orientation: Elegimos la escala para la resolución gráfica de la aplicación en el dispositivo, así como la orientación de la pantalla del mismo.
- Emulated performance: Es recomendable marcar la opción *Snapshot*, para que en sucesivas emulaciones la velocidad de apertura del emulador sea más rápida.

	Virtual Device Confi	iguration	· · · · ·	×
Configure AVD				
AVD Name	Galaxy Nexus API 21			
Galaxy Nexus	4,65" 720x1280 xhdpi	Change		
Scollipop	Android 5.0 x86_64	Change		
Startup size and Scale: 10dp of orientation Orientation: Po	ortrait		Nothing Selected	
Emulated Performance	<ul> <li>□ Use Host GPU</li> <li>✓ Store a snapshot for faster startup</li> <li>You can either use Host GPU or Snapshots</li> </ul>			
Show Advanced Settings			Previous Next Cancel	<u>F</u> inish

Ilustración 3.6.Pantalla de configuración de un nuevo dispositivo en AVD Manager.

Una vez configurado el dispositivo virtual en el que ejecutar la aplicación sólo tenemos que hacer uso del botón "run" le que una vez terminado el código, lo ejecutará y cargará sobre nuestra máquina virtual.



Ilustración 3.7. Vista del emulador ejecutado desde Android Studio.

#### 3.4 Componentes de una aplicación Android

Para desarrollar una aplicación, necesitamos antes conocer una serie de conceptos básicos. Una aplicación está fundamentalmente formada por una serie de componentes principales que actúan sobre las ventanas, control, eventos o servicios.



Grafica 3.1. Componentes principales de una aplicación Android.

A continuación se definen los siete elementos principales que conforman la mayoría de acciones en una aplicación.

• Activity

Las actividades (*activities*) representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana en cualquier otro lenguaje visual.

• View

Los objetos *view* son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, análoga por ejemplo a los controles de Java o .NET. De inicio, Android pone a nuestra disposición una gran cantidad de controles básicos, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear nuestros propios controles personalizados.

• Service

Los servicios son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son exactamente iguales a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (p.ej. *activities*) si se necesita en algún momento la interacción con del usuario.

#### • Content Provider

Un *content provider* es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los *content provider* que se hayan definido.

• Broadcast Receiver

Un *broadcast receiver* es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: "Batería baja", "SMS recibido", "Tarjeta SD insertada",...) o por otras aplicaciones (cualquier aplicación puede generar mensajes (*intents*, en terminología Android) *broadcast*, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo.

• Widget

Los *widgets* son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (*home screen*) del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal.

• Intent

Un *intent* es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un *intent* se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje *broadcast*, iniciar otra aplicación, etc.

#### 3.5 Otras herramientas

Además de lo anteriormente comentado es necesario conocer también otras herramientas que sirven de apoyo a la hora de realizar un proyecto en Android Studio, como son:

- Generador y editor de iconos
- Guía de desarrollo de Android
- Sitios webs de consulta
- Google Developers Console
- KeyTool

#### 3.5.1. Generador y editor de iconos

Como se ha visto antes, la carpeta "res" contenía imágenes e iconos con diferentes tamaños. Estos iconos se almacenan en las carpetas "Drawable", dentro del directorio "res", existiendo un total de cuatro carpetas creadas de forma automática al generar nuestro proyecto en blanco, estas son: drawable-hdpi, drawable-mdpi, drawable-xhdpi y drawable-xxhdpi. Las carpetas están organizadas por tamaños y se utilizan para almacenar en ellas el mismo icono con distintas dimensiones con el fin de que cuando el dispositivo tenga una resolución mayor automáticamente se tome el tamaño acorde a la resolución y tamaño de la pantalla del dispositivo.

Normalmente, la utilización de una herramienta gráfica que genere automáticamente el icono en varios tamaños facilita mucho el trabajo. Por este motivo en internet encontramos multitud de herramientas para este fin. Una de ellas, que ha sido proyecto. empleada este Android Asset Studio en es (http://romannurik.github.io/AndroidAssetStudio/index.html) que mediante un sencillo menú nos permite adaptar los distintos tipos de iconos, que necesita nuestra aplicación, a los tamaños adecuados.

Foreground	IMAGE	CLIPART	TEXT		
	TRIM	IM DON'T	TRIM		
	PADDING	•	c	196	
Color For transparent foregrounds	•		0%		
Scaling	CROP	CENTER			
Shape					
NONE BEVEL	SQUARE C	IRCLE VE	RTICAL RECTANGLE	HORIZONT	AL RECTANGLE
Background					
Effect	NONE	LONG SHADOV	V SCORE D	OG-EAR	
DOWNLOAD ZIP	GENERATE WE	3 ICON			
		xxhdpi	xhdpi	hdpi	mdpi web, hi-re
xxxhdpi					
xxxthdpi					
xxxhdpi					

Ilustración 3.8.Pantalla de generación de iconos de Asset Studio.

#### 3.5.2. Guía de desarrollo de Android

Como en todo proyecto, para empezar a desarrollar una aplicación para Android debemos tener una guía que nos sirva de referencia para la consulta de prototipos y clases.

En este caso la página web oficial de Android Developers nos permite acceder a todo el contenido oficial acerca de clases, recursos y API útiles para el desarrollo de cualquier aplicación. Está organizado todo por grupos según su uso y en cada uno de ellos se puede consultar una descripción acerca del contenido y funcionalidad de una clase java o de la implementación y configuración de una API.

📫 🕂 Develop > API	uides > Introduction to Android			9
App Fundamentals	Introductio	n to Android		
Device Compatibility				
System Permissions	Android provides a rich ap innovative apps and game	plication framework that allo s for mobile devices in a Jav	ws you to build a language	To learn how apps work, start with App Fundamentals.
App Components	<ul> <li>environment. The docume details about how to build</li> </ul>	nts listed in the left navigatio apps using Android's various	n provide APIs.	To begin coding right away, read Building Your First App.
App Resources	V If you're new to Android de	evelopment, it's important that	it vou	
App Manifest	understand the following t	fundamental concepts about	the Android app fra	mework:
User Interface	Apps provide multip	le entry points	Apps adapt to	o different devices
Animation and Graphics	Android apps are built as a	a combination of distinct	Android provides	an adaptive app framework that
Computation	<ul> <li>components that can be in instance, an individual act</li> </ul>	nvoked individually. For ivity provides a single	allows you to pro device configura	ovide unique resources for different tions. For example, you can create
Media and Camera	<ul> <li>screen for a user interface independently performs w</li> </ul>	, and a <i>service</i> ork in the background.	different XML lay and the system o	yout files for different screen sizes determines which layout to apply
Location and Sensors	From one component you	can start another	based on the cur	rent device's screen size.
Connectivity	<ul> <li>component using an inten component in a different a</li> </ul>	t. You can even start a app, such an activity in a	You can query th runtime if any ap	e availability of device features at p features require specific hardware
Text and Input	<ul> <li>maps app to show an add multiple entry points for a</li> </ul>	ress. This model provides single app and allows any	such as a camera features your app	a. If necessary, you can also declare p requires so app markets such as
Data Storage	<ul> <li>app to behave as a user's other apps may invoke</li> </ul>	"default" for an action that	Google Play Stor	e do not allow installation on device ort that feature
Administration	Learn more:		Learn more:	or that reature.
Web Apps	App Fundamentals		Device Compatib	ility
	'ntents and Intent Filters		Resources Overv	riew

Ilustración 3.9. Pantalla principal de la web Android Developer.

#### 3.5.3. Sitios webs de consulta

Además de la guía oficial para desarrolladores Android, existen multitud de sitios de consultas concretas acerca de la funcionalidad de un código implementado o de fallos en la ejecución de una determinada aplicación...Entre los más destacado se sitúa Stack Overflow (stackoverflow.com), sitio web donde consultar como solucionar problemas a la hora de desarrollar aplicaciones ofreciendo una fiabilidad considerable en sus respuestas.

#### 3.5.4. Google Developers Console

Además de nuestro entorno de desarrollo y la implementación del código de nuestra aplicación, en muchos casos necesitaremos usar herramientas de Google para distintos cometidos como pueden ser Google Maps, Google Cloud, Google Drive... Para ello necesitaremos registrarnos como desarrolladores en la web Google Developers para así obtener una clave que nos permita hacer uso de sus servicios.

Una vez autorizados, podemos activar las APIs necesarias mediante el uso de la consola central de la web.



Por último destacar que este registro también sirve para posteriormente, de manera opcional, poder introducir nuestra aplicación en la tienda Google Play Store y así hacer accesible nuestra aplicación a todos los dispositivos Android.

#### 3.5.5. KeyTool

Una vez creada la aplicación se genera un paquete de extensión .apk que tendrá que ir firmado con un certificado propio. Mediante la herramienta KeyTool incluida en el paquete Java, se puede generar un certificado propio con el que firmar la aplicación generada.

Para ello solo hay que introducir la siguiente orden en la consola de comandos de Windows:

### keytool -genkey -v -keystore mi\_certificado.keystore -alias MI\_ALIAS -keyalg RSA -keysize 2048 -validity 10000

Y se accede a la creación del certificado para el cual se ha de responder una serie de preguntas que van apareciendo en la consola de comandos como se puede observar en la ilustración 3.11.

:\Users\ignac_000 lerramienta de Gest	>keytool tión de Certificados y Claves
Comandos =	
-certreq -changealias -delete -exportcert -genkeypair -genseckey -gencert	Genera una solicitud de certificado Cambia un alias de entrada Suprime una entrada Exporta el certificado Genera un par de claves Genera un clave secreta Genera un clave secreta Genera un certificado a partir de una solicitud de certific
ido −importcert −importpass −importkeystore	Importa un certificado o una cadena de certificados Importa una contraseña Importa una o todas las entradas desde otro almacén de clav
-keypassud -list -printcert -printcertreq -printcrl -storepassud	Cambia la contraseña de clave de una entrada Enumera las entradas de un almacén de claves Imprime el contenido de un certificado Imprime el contenido de una solicitud de certificado Imprime el contenido de un archivo CRL Cambia la contraseña de almacén de un almacén de claves
Itilice "keytool — C:\Users\ignac_000 RUEBA_MEMORIA —key Introduzca la cont ¿Cuáles son su nom [Unknown]: Prue ¿Cuál es el nombre [Unknown]: Memo: [Unknown]: Memo: ¿Cuál es el nombre	command_name -help" para la sintaxis de nombre_comando >keytool -genkey -v -keystore mi_certificado.keystore -alias P alg RSA -keysize 2048 -validity 10000 raseña del almacén de claves: bre y su apellido? ba de su unidad de organización? ria PFC de su organización?
[Unknown]: US ¿Cuál es el nombre [Unknown]: Sevi ¿Cuál es el nombre [Unknown]: Sevi ¿Cuál es el código [Unknown]: ES	de su ciudad o localidad? lla de su estado o provincia? lla de país de dos letras de la unidad?

Ilustración 3.11. Herramienta KeyTool en la consola de comandos.

# **4** PLANIFICACIÓN DE LA APLICACIÓN.

n este apartado se aborda la planificación previa a la creación de clases y actividades que conformarán el funcionamiento de la aplicación.

#### 4.1. División en fases

Como comienzo de la planificación y estructurado de la aplicación, se dividirá su desarrollo en una serie de fases las cuales requerirán posteriormente una serie de actividades y clases asociadas a cada una de ellas.

Las fases en las que se va a dividir la aplicación serán las siguientes:

- Fase I: Desarrollo de un bloc de notas y gestión de su base de datos.
- Fase II: Utilización y aplicación de la API de Google Maps en la situación de la nota almacenada.
- Fase III: Uso y gestión de las alertas por proximidad.

A continuación, se desarrollará cada una de estas fases de manera pormenorizada.

#### 4.2. Fase I: Desarrollo de un bloc de notas y gestión de su base de datos.

En esta fase se comienza a fijar el punto de partida de la aplicación.

La aplicación, en cuanto a funcionamiento básico, se va a asemejar a un bloc o libreta de notas con la particularidad de añadir un campo de localización, por tanto se debe partir de un planteamiento básico: la creación de un bloc de notas que cumpla los requisitos necesarios de la aplicación.

Este bloc de notas incluirá dos campos:

- Campo Ubicación: En él se reflejará la ubicación asociada a la nota.
- Campo Nota: En él se reflejará el contenido de la nota que queremos almacenar.

Estos dos campos serán almacenados en una base de datos SQL que tendrá esas dos columnas además de un indicador o ID para cada dupla de datos. Dicha base de datos nos debe permitir el borrado de cada entrada, su modificación y la eliminación completa del contenido de la base de datos.

A continuación, se muestra un diagrama donde se explica el planteamiento respectivo al funcionamiento de la fase I.



Ilustración 4.1.Diagrama de funcionamiento de la fase I.

En el diagrama se puede observar que para añadir una nueva nota hay que rellenar forzosamente el campo nota. El campo ubicación no será necesario rellenarlo ya que, aunque la aplicación añade la funcionalidad de alerta en proximidad de una ubicación dada, si no se da dicha ubicación funcionaría como un bloc de notas.

Por último por cada entrada se añadirá un ID autoincrementado que nos servirá para modificar y borrar la entrada creada mediante los métodos que se diseñen para tal cometido.

## 4.3. Fase II: Utilización y aplicación de la API de Google Maps en la situación de la nota almacenada.

Esta fase utilizará la API de Google Maps para mostrarnos sobre un mapa dicha ubicación y para modificar, si fuera necesario, la ubicación fijada en la Fase I.

En la fase anterior se ha almacenado una nueva nota con los campos título, ubicación y nota, partiendo de la premisa de que el campo ubicación no está vacío, se plantea en esta fase mostrar dicha ubicación sobre el mapa y permitir modificarla moviéndonos sobre el mismo. Si se decide modificar la nota habrá que hacer uso de los métodos necesarios de gestión de la base de datos para modificar la entrada correspondiente almacenada en la base de datos.

El planteamiento con respecto al desarrollo de esta fase II se muestra en el siguiente diagrama.



Ilustración 4.2.Diagrama de funcionamiento de la fase II.

La planificación de funcionamiento de la fase II reflejada en el diagrama, es la siguiente: De la base de datos se obtiene la ubicación y se realiza una consulta para su situación en el mapa mediante la API de Google Maps, si esta consulta devuelve un resultado nulo la dirección no se ha reconocido y vuelve de nuevo a la fase I eliminando la entrada de la base de datos. Si por el contrario la consulta devuelve un resultado diferente, la dirección se ha reconocido y se muestra en el mapa. Una vez situada en el mapa, si se modifica moviendo dicho mapa, se modificará la entrada en la base de datos cambiando dicha ubicación, o de lo contrario finalizará la fase II.

#### 4.4. Fase III: Uso y gestión de las alertas por proximidad.

La última fase del planteamiento inicial de la aplicación, sería la fase de gestión de alertas. En ella se va a analizar si la posición gps del dispositivo está cerca, entendiéndose cerca como en un radio de metros fijado por el usuario, y se van a tomar en función de esta cercanía o proximidad las acciones de alerta necesarias.

Para ello será necesario emplear funciones que consigan triangular la posición del dispositivo en todo momento y a partir de sus resultados lanzar o no una alerta.

El planteamiento con respecto al desarrollo de esta fase III se muestra en el siguiente diagrama.



Ilustración 4.3.Diagrama de funcionamiento de la fase III.

#### 4.5. Diagrama total.



Ilustración 4.4. Diagrama total de planificación del funcionamiento de la aplicación.

#### 4.6. Planificación temporal

Para culminar el capítulo de planificación, se detalla a continuación los hitos temporales cumplidos en el desarrollo de la aplicación.

La planificación temporal aproximada de la realización de la aplicación se divide en siete fases que detallo a continuación:

- PFC: Duración del desarrollo total de la aplicación y del proyecto en sí.
- Recopilación de información: Fase donde se toman datos e información acerca de programación Android relacionada con el objetivo del proyecto.
- Instalación y configuración de las herramientas de desarrollo: También se incluye el boceto a grandes rasgos de cómo desarrollar la aplicación.
- Fase I: Primer prototipo con la funcionalidad de bloc de notas únicamente.
- Fase II: Se incluyen y elaboran las funciones de localización y de posicionamiento.
- Fase III: Se añaden por último las funciones de alerta y notificación.
- Configuración global y ajustes: La fase más larga ya que trata de unir todo lo anterior sin ningún error y adaptarlo a distintos dispositivos. En ella se realiza también un importante ajuste de la interfaz gráfica.

Fase	Fecha inicio prevista	Días trabajados	Fecha final prevista	Situación	Días para el final
PFC	1-oct14	243	1-jun15	Terminado	0
Recopilación de información	2-oct14	13	15-oct14	Terminado	0
Instalación y configuración de					
herramientas de desarrollo	15-oct14	2	17-oct14	Terminado	0
Fase I: Prototipo bloc de notas	18-oct14	23	10-nov14	Terminado	0
Fase II: Funciones de localización y					
posicionamiento	11-oct14	60	10-dic14	Terminado	0
Fase III: Sistema de avisos y					
recordatorios	11-dic14	65	14-feb15	Terminado	0
Configuración global y ajustes	16-feb15	105	1-jun15	Terminado	0

Tabla 4.1.Tabla con los hitos de la planificación temporal y su duración.



Ilustración 4.5.Diagrama de Gantt asociado a la tabla 4.1

## **5** ESTRUCTURA Y DESARROLLO DE LA APLICACIÓN

aciendo uso de todo lo visto en el capítulo anterior, en este capítulo se procede a explicar de manera detallada la estructura de ficheros y clases de la aplicación así como recorrer la interfaz gráfica explicando cada pantalla y su funcionalidad.

#### 5.1. Estructura del programa.

La estructura básica de organización de los ficheros que componen nuestra aplicación es la descrita en el apartado anterior. Partiendo de esa estructura de carpetas nos centraremos en los archivos que las componen, su descripción y su funcionalidad.

#### 5.1.1 Carpeta /src/

En esta carpeta se concentran todos los archivos .java que implementan las acciones del programa.



Ilustración 5.1. Estructura de archivos de la carpeta /src/

En la ilustración 5.1 podemos observar los archivos .java que componen nuestra aplicación, por comodidad para su descripción procederé a dividirlos en varios bloques:

- Primer bloque: Archivos principales asociados a pantallas gráficas de nuestra aplicación. Dentro de este primer bloque se encontrarían los siguientes archivos:
  - MyActivity.java

- ShowNotasActivity.java
- MapsActivity.java
- Milocalizaciones.java
- Notificacionesmenu.java
- Radiomenu.java

• Segundo bloque: Archivos secundarios que aportan funcionalidad a los archivos principales. Dentro de este segundo bloque se encontrarían los siguientes archivos:

- Lista\_entrada.java
- Notificar\_GPS.java
- ReceptorProximidad.java

• Tercer bloque: Archivos auxiliares necesarios para el manejo de ciertas estructuras funcionales. Dentro de este tercer bloque se encontrarían los siguientes archivos:

- GlobalClass.java
- Handler\_NotasB.java
- Lista\_adaptador.java
- PlacesAutoCompleteAdapter.java

#### 5.1.2 Carpeta /res/

Como se explicó en el capítulo anterior, esta carpeta almacena en gran parte todo el aspecto gráfico de la aplicación.

Por comodidad para un posterior desarrollo vamos a agrupar las subcarpetas pertenecientes a /res/ en cuatro grandes grupos:

• Iconos y gráficos: Carpetas destinadas al almacenamiento de los archivos que corresponden a iconos y demás. [Ver Ilustración 5.2 color azul]

• Menú: Carpeta destinada al almacenamiento de los archivos de menú contextual. [Ver Ilustración 5.2 color rojo]

• Layout: Carpeta destinada al almacenamiento de los distintos archivos .xml que conforman la interfaz gráfica de la aplicación. [Ver Ilustración 5.2 color verde]

• Values: Carpeta destinada al almacenamiento de los archivos que almacenan *Strings* y valores en los diferentes idiomas soportados por la aplicación. [Ver Ilustración 5.2 color amarillo]



Ilustración 5.2. Estructura de subcarpetas de la carpeta /res/

Estas son fundamentalmente los dos directorios donde vamos a concentrar la mayor parte del desarrollo de nuestra aplicación.

A continuación se ofrece un desarrollo más amplio de cada una de las clases y ficheros definidos en nuestra aplicación.

#### 5.2. Descripción y funcionalidad de las clases.

Este apartado se centra en realizar un análisis de las diferentes pantallas que componen la aplicación y exponer la funcionalidad de las clases que constituyen cada pantalla.

#### 5.2.1. Pantalla principal

Esta pantalla abre la aplicación y su cometido es distribuir las diferentes acciones que podemos realizar en la aplicación.

Al ejecutar la aplicación lo primero que haría es comprobar si el usuario tiene activado la localización mediante satélites GPS, si no está activa esta funcionalidad automáticamente nos redirige a la pantalla de ajustes de seguridad y ubicación donde podemos activarlo.



Ilustración 5.3.Pantalla de ajustes de seguridad y ubicación.

Una vez activado el GPS observamos la pantalla principal de la aplicación que nos permite realizar dos acciones: agregar nueva nota y consultar la lista de notas guardadas.

Para agregar una nueva nota se ha de rellenar el formulario con los siguientes campos: el campo nota, donde introduciremos el contenido de nuestra nota, y el campo ubicación que especifica la ubicación a partir de la cual queremos que nos recuerde la nota introducida.

Cabe destacar que la aplicación, como se comentó en la introducción, incorpora una doble funcionalidad por un lado la de bloc de notas y por otro la de recordatorio. De esta

forma podemos también introducir una nueva nota sin necesidad de rellenar el campo ubicación implementando exclusivamente la funcionalidad de bloc de notas.

El campo nota ha de ser rellenado obligatoriamente, de no ser así se nos mostrará un mensaje avisándonos de que debemos rellenar este campo para continuar con la inclusión de una nueva nota.

En el campo ubicación se hace uso de la función de autocompletado incluida en la API Google Places<sup>1</sup> para facilitar la tarea de introducción de direcciones, así mismo también se nos permite mediante un botón seleccionar unas ubicaciones previamente almacenadas como se describirá en el apartado 5.2.4.1.

Para consultar las notas almacenadas solo hay que pulsar sobre el botón destinado a dicha acción y podremos consultar una lista de notas que se describirá en el apartado 5.2.3.

Ilustración 5.4.Pantalla principal de la aplicación.

En situaciones donde no exista conexión a internet, la

aplicación puede funcionar empleando las localizaciones predeterminadas, para ello es necesario que previamente se hayan almacenado las mismas disponiendo en tal momento de conexión a la red. En el modo *offline*, la aplicación no mostrará la pantalla mapa, redirigiendonos directamente a la pantalla de visualización de notas.

El código de la parte visual de esta pantalla principal se almacena como vimos en el apartado anterior en la carpeta /res/layout/y fundamentalmente el archivo .xml está compuesto por los siguientes elementos:

•TextView: Un total de tres *TextView* utilizados para los títulos de cada campo a introducir así como del título principal "Agregar nueva nota".

•EditText: Se utiliza un *EditText* para el campo nota.

•AutoCompleteTextView: Se utiliza un *AutoCompleteTextView* para el campo ubicación ya que este campo implementa la función autocompletado.

•ImageButton: Se utiliza un ImageButton para el botón .

<sup>&</sup>lt;sup>1</sup> Para ello será necesario activar la función en la consola de Google Developers (Capitulo 3, apartado 3.5) e implementar la clase java "PlacesAutoCompleteAdapter.java" descrita en el apartado 5.2.6.

•Button: Se utilizan dos *Button* para cada una de las acciones posibles en esta pantalla "agregar nota nueva" y "Ver todas las notas".

El código que implementa la funcionalidad de estos elementos se almacena en la carpeta /src/main/java/.

#### 5.2.2. Pantalla Mapa

Esta pantalla<sup>2</sup> se muestra tras introducir todos los campos comentados en el apartado anterior (incluyendo el campo ubicación), y en ella se observa un mapa con la posición que hemos introducido textualmente señalizada mediante un *marker* o marcador rojo acompañado de un título "Situación". Además con un punto azul se nos muestra nuestra posición actual.



Ilustración 5.5. Vista principal de la pantalla mapa.

La pantalla mapa incluye la funcionalidad de poder cambiar la dirección introducida textualmente por otra seleccionada en el mapa, para ello solo tenemos que mover el mapa deslizando el dedo por la pantalla y utilizando los controles de zoom y una vez conseguida la vista de la calle deseada dando un *click* o golpe de dedo sobre la misma se nos conducirá a la pantalla de vista de todas las notas, donde comprobaremos que se muestra la nueva nota con la posición modificada.

Se muestra en esta pantalla también un botón, el botón "siguiente" que hemos de pulsar en el caso que estemos conformes con la dirección introducida textualmente y mostrada en el mapa. Este botón nos lleva a la pantalla de vista de todas las notas.

Para poder mostrar la pantalla con el mapa, hay que efectuar el registro en la web de Google Developers. Tras ello, mediante la consola de Windows y la herramienta Java

<sup>&</sup>lt;sup>2</sup> La pantalla Mapa sólo se mostrará si el dispositivo tiene activada la conexión a internet.

"Keytool", debemos de obtener un certificado para poder firmar nuestra aplicación una vez esté lista. En dicho certificado se incluye una clave SHA1 con la que tenemos que generar la clave que incluiremos en nuestro archivo Android Manifest para poder así usar la API de Google Maps.

as Administrator: C:\Windows\system32\cmd.exe	
<pre>c:\Program Files\Java\jre6\bin&gt;keytool -genkey -keyalg RSA -alias selfsigned -keystore -storepass password -validity 360 -keysize 2048 What is your first and last name? IUnknown1: www.google.com What is the name of your organizational unit? IUnknown1: Web Development What is the name of your organization? IUnknown1: Google, Inc. What is the name of your City or Locality? IUnknown1: Mountain Uiew What is the name of your State or Province? IUnknown1: California What is the name of your State or this unit? IUnknown1: US Is CN=www.google.com, OU=Web Development, 0="Google, Inc.", L=Mountain View, ST=Califor rect? Ino]: yes</pre>	keystore.jks rnia, C=US cor
Enter key password for <selfsigned> (RETURN if same as keystore password): Re-enter new password:</selfsigned>	
c:\Program Files\Java\jre6\bin>	

Ilustración 5.6. Pantalla de muestra de la herramienta keytool de generación de certificados.

Como se muestra en la ilustración 5.7, hemos de activar la API de Google Maps en el panel de control de la web Google Developers.

< Proyectos	APIs habilitadas		
NotasB Descripción general	Algunas Art se activan automaticamente. Puedes desactivarias si no usas sus servicios.	CUOTA	ESTADO
Permisos	BigQuery API	0 %	A
APIs v autenticación	Google Cloud SQL		A
APIs	Google Cloud Storage		A
Credenciales Pantalla de autorización	Google Cloud Storage JSON API		A
Push	Google Maps Android API v2		A
Supervisión			
Código fuente	Google Maps Geolocation API	100 %	A
Cálculo	Google+ API	0 %	A
Redes			
Almacenamiento	Navegar por las API		
Big Data	Filtrar nor el nombre o la descritoción de la API		
Asistencia			
~	NOMBRE A	CUOTA	ESTADO

Ilustración 5.7. Pantalla de habilitación de las APIs de Google (web Google Developers)

En la siguiente ilustración se muestran las líneas de código donde se configura el marcador así como el punto azul que indica la posición actual.
```
mMap.addMarker(new MarkerOptions().position(punto).title("Situación").draggable(true)).showInfoWindow();
//Añado un marker rojo en el punto indicado como ubicación.
mcamara = CameraUpdateFactory.nevLatLngZoom(punto, 14);
//Centro el mapa en torno al punto indicado como ubicación.
mMap.animateCamera(mcamara);
//Activamos la capa o layer MyLocation
```

mMap.setMyLocationEnabled(true);

Ilustración 5.8. Fragmentos de código java donde se añade el marcador y

el puntero de localización actual.

## 5.2.3. Pantalla de visualización de notas.

Esta pantalla se muestra tras haber pulsado el botón "siguiente" de la pantalla mapas o haber introducido una nota con el campo ubicación vacío o por último haber presionado el botón "Ver todas las notas". En ella se nos muestra una lista de todas las notas almacenadas en la base de datos.

prueba	OFF
Camino de los Descubrimientos, Seville	e, Spain
prueba	OFF X
Camino de los Descubrimientos, Seville	e, Spain
prueba3	OFF X

Ilustración 5.9. Pantalla de visualización de notas.

En la vista de cada nota podemos ver en la parte superior el texto correspondiente al campo nota y en la parte inferior el texto correspondiente al campo ubicación.

En el recuadro destinado a cada nota se incluyen también dos botones:

- El botón eliminar 🔀 cuyo cometido es borrar de la base de datos la nota correspondiente y actualizar la pantalla de visualización de notas.
- El botón ON/OFF cuyo cometido es activar o desactivar la alerta de

proximidad y su notificación.

La información mostrada está estructurada, como ya se ha visto, en forma de listado debido a la facilidad para visualizar toda la información almacenada al agregar una nueva nota. Esta estructura visual permite desplazando nuestro dedo hacia arriba o hacia abajo movernos en el listado de notas almacenadas.

El código de la parte visual de esta pantalla queda almacenado en la carpeta /res/layout/ y fundamentalmente el archivo .xml está compuesto por los siguientes elementos:

•ListView: Es el elemento que hace posible la vista listado y está incluido en un .xml aparte de los demás elementos.

•TextView: Un total de dos *TextView* utilizados para cada nota que permiten la visualización de los dos campos principales que componen la misma.

•ToggleButton: Un botón ON/OFF por cada nota, activará o desactivará el aviso de proximidad y sus notificaciones.

•ImageButton: Un botón por cada nota cuya utilidad es la eliminación de dicha nota.

El código que implementa la funcionalidad de estos elementos se almacena en la carpeta /src/main/java/.

## 5.2.4. Menú y pantallas de ajustes.

En este apartado se realiza un análisis acerca del menú contextual, su definición y sus pantallas de opciones. El menú de opciones es la colección primaria de *ítems* que aparecen en una actividad cuando el usuario oprime el botón menú del dispositivo.

En nuestra aplicación el menú aparecerá tras pulsar la tecla menú del dispositivo, y presenta las siguientes opciones:

• Eliminar todo: Opción que sirve para eliminar todas las notas almacenadas en la base de datos y actualiza la pantalla de visualización de notas. Sólo se aplica a la pantalla de visualización de notas, en la pantalla principal no está activo su funcionamiento.

• Acerca de: Opción que sirve para mostrar un mensaje con la información de versión de la aplicación y de su desarrollador.

• Ajustes: Opción que sirve para redirigirnos al submenú de ajustes. Ver apartado 5.2.4.1.

• Nueva nota: Opción que tras ser pulsada nos redirige a la pantalla principal para agregar una nota nueva.

Agregar nota nueva	Ver todas las notas	
Introduzca la ubica	ación: 🕒	
Nota:		
8	¢	
🙁 Eliminar todo	Ç Ajustes	
Eliminar todo	C Ajustes	

Ilustración 5.10. Menú sobre la pantalla principal de la aplicación.

Cabe destacar que el menú aparece en la pantalla principal y en la pantalla de visualización de notas, en la pantalla de mapas no aparece este menú para no obstaculizar la visión del mapa.

El menú con sus opciones queda definido en un archivo .xml dentro de la carpeta /res/menú/ y en nuestro caso se llama "my.xml".

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MyActivity" >
   <item android:id="@+id/action settings"
       android:title="Settings"
       android:icon="@drawable/stting"
        android:orderInCategory="2"
       app:showAsAction="never">
        <menu>
            <item android:id="@+id/SubMnuOpc2"
                android:title="Preferred locations"
                android:icon="@drawable/localizaciones"
                />
            <item android:id="@+id/SubMnuOpc3"
                android:title="notification radius">
            </item>
            <item android:id="@+id/SubMnuOpc4"
                android:title="Notifications"
                android:icon="@drawable/notificacion"/>
       </menu>
        </item>
   <item android:id="@+id/action_delete"</pre>
       android:title="Delete all"
       android:orderInCategory="1"
       android:icon="@drawable/icon delete"
        app:showAsAction="never" />
   <item android:id="0+id/action acercade"
        android title-"Nhout the ann
```

Ilustración 5.11. Muestra del código de my.xml.

El comportamiento de cada opción del menú se define en los archivos .java correspondientes. En este caso el menú de opciones se define en los archivos "MyActivity.java" y "ShowNotasActivity.java", que son los correspondientes a la pantalla principal y a la pantalla de visualización de notas.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.my, menu);
    return true:
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
    if (id == R.id.action acercade) {
        Toast.makeText(
                this,
                "P.F.C Ignacio Glez. de la Peña Puerta" +
                        " (c)All right reserved",
                Toast.LENGTH LONG)
                .show();
if (id==R.id.SubMnuOpc3) {
    Intent am=new Intent(this, radiomenu.class);
        startActivity(am);
        }
    if(id==R.id.SubMnuOpc4){
```

Ilustración 5.12. Muestra del código de menú implementado en MyActivity. java.

# 5.2.4.1. Submenú ajustes.

Este submenú se lanza tras pulsar en el botón "Ajustes" del menú de opciones de la aplicación. Dicho submenú se divide en tres opciones correspondientes a tres aspectos configurables del entorno de funcionamiento de la aplicación, estos son: localizaciones preferidas, radio de aviso y notificaciones.

A continuación se describe cada uno de estos aspectos con más detenimiento.



Ilustración 5.13. Submenú ajustes.

• Localizaciones preferidas.

Esta opción del submenú ajustes nos permite, mediante una sencilla interfaz gráfica, guardar tres ubicaciones frecuentes o preferidas en las categorías casa, trabajo y P.O.I (*Point Of Interest*). En cada campo introduciremos la dirección concreta para cada caso y así cuando agreguemos una nueva nota, podemos evitar escribir de nuevo una dirección que es usada con frecuencia.

En la pantalla se observan, además de los distintos recuadros para introducir direcciones (*AutoCompleteTextView*), un botón que nos permitirá guardar las direcciones introducidas en los campos descritos (Botón Ok).

GeoNudge	* 🗇 🤿	44% 🗎 2	20:56
Casa			
Trabajo			
• P.O.I			
	Ok		

Ilustración 5.14.Pantalla de la opción del submenú ajustes "Localizaciones preferidas".

El código java de esta opción se encuentra en "milocalizaciones.java" y hace uso de la clase de Android "SharedPreferences" que tiene la funcionalidad inicial de recordar un valor y compartirlo con otros métodos o actividades. Lo realmente interesante de esta función es que este valor continuará grabado y recordado aunque cerremos nuestra aplicación o reiniciemos el móvil.

Por otro lado como se ha comentado antes, esta opción del submenú ajustes emplea "AutoCompleteTextView" ya que como ocurría en la pantalla principal, se hace uso de la función autocompletado incluida en la API de Google Places<sup>3</sup>.

<sup>&</sup>lt;sup>3</sup> Para ello será necesario activar la función en la consola de Google Developers (Capitulo 3, apartado 3.5) e implementar la clase java "PlacesAutoCompleteAdapter.java" descrita en el apartado 5.2.6.

```
public void aceptarloc (View v) {//Método para cuando se pulsa el botón "Ok".
   EditText casa, trabajo, poi;
    casa= (EditText) findViewById(R.id.editTextCASA);
   trabajo=(EditText) findViewById(R.id.editTextTRABAJO);
   poi=(EditText) findViewById(R.id.editTextPOI);
   final Geocoder geoCoder = new Geocoder(getBaseContext(), Locale.getDefault());
   final GlobalClass globalVariable = (GlobalClass) getApplicationContext();
    //Obtenemos las ubicaciones introducidas en los distintos EditText.
    SharedPreferences prefs = getSharedPreferences("preferenciasMiApp", this.MODE PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
   //Las almacenamos en nuestro archivo de preferencias.
   if ((prefs.getString("casa", null) !=null) & (!casa.getText().toString().equals(""))) {
      editor.putString("casa", casa.getText().toString());
          }//laprimera vez
    if (prefs.getString("casa", null) ==null) {
        editor.putString("casa", casa.getText().toString());
    }
    //laprimera vez
   if (prefs.getString("trabajo", null) ==null) {
        editor.putString("trabajo", trabajo.getText().toString());
   }
    if ((prefs.getString("trabajo", null) !=null) && (!trabajo.getText().toString().equals("")))
        editor.putString("trabajo", trabajo.getText().toString());
    }
```

Ilustración 5.15.Fragmento de código donde se emplea la clase SharedPreferences.

Una vez introducidas estas direcciones, la siguiente vez que se acceda a esta sección del menú se podrán visualizar estas direcciones ya que aparecerán en los cuadros destinados a introducir direcciones, como texto por defecto.

Hay que destacar que por cada localización introducida se realiza una consulta de sus coordenadas y se almacena en variables del tipo *SharedPreferences* a modo de copia de seguridad para situaciones en las que no exista conexión a internet.

En cuanto a los archivos .xml, utilizados para la interfaz gráfica, constan de los siguientes elementos:

• <u>AutoCompleteTextView</u>: Uno por cada ubicación a introducir, en total tres *AutoCompleteTextView* que recogen los datos introducidos por el usuario.

• <u>Button</u>: Como hemos comentado anteriormente, en la pantalla de introducción de direcciones se emplea uno.

• <u>TextView</u>: En la pantalla de introducción de ubicaciones se emplean tres elementos *TextView* para indicarnos lo que hemos de introducir en cada recuadro.

• Radio de aviso.

Mediante esta opción del submenú ajustes, podemos configurar el radio medido desde la ubicación que fijemos para que la aplicación nos recuerde la nota almacenada. Para ello, mediante un *EditText* se introduce el radio deseado en metros; por defecto sino se introduce nada, la aplicación tomará un radio de 500 metros.

El código asociado a esta pantalla se encuentra en "radiomenu.java" y es muy simple: Obtenemos el radio introducido en el *EditText* y mediante la clase *SharedPreferences* almacenamos este dato en nuestro archivo de preferencias, dejándolo disponible para las funciones que necesiten emplear este dato.

Hay que reseñar, que la configuración de este parámetro ha de hacerse antes de activar ninguna alerta para el aviso de notas, de lo contrario, la alerta ya activada tomará el radio anteriormente introducido o sino el radio por defecto.

En cuanto al archivo .xml, este consta únicamente de dos elementos principales:

- EditText: Recuadro donde introducimos el radio deseado.
- Button: Tras pulsarlo nuestra configuración quedará almacenada.

GeoNudge Configure avisado	la distancia a la que quiere ser
	ОК

Ilustración 5.16. Pantalla del submenú radio de aviso.

• Notificaciones.

Esta opción del submenú de ajustes permite configurar la notificación, es decir el mecanismo que tendrá la aplicación de avisar cuando el usuario esté dentro del radio de proximidad fijado.

En la pantalla creada para tal fin, se puede elegir entre tres formas de aviso:

- Aviso mediante vibración.
- Aviso mediante sonido.
- Aviso mediante luz.

Estas tres formas se pueden activar conjuntamente sin ningún problema. En el caso de activar el aviso luminoso, se nos informa en la propia pantalla de que dicho aviso está limitado a dispositivos que dispongan de luz para notificaciones.

GeoNudge	
Configure sus notificacione	S
Notificación por vibración	OFF
Notificación sonora	OFF
Notificación luminosa	OFF
(*)Sólo en algunos dispositivos	
Guardar	

Ilustración 5.17.Pantalla de configuración de notificaciones.

El código asociado a esta pantalla se encuentra en "notificacionesmenu.java", en dicho código se hace uso nuevamente de variables del tipo *SharedPreferences* que almacenará el estado de los *ToggleButton* que activan o desactivan cada opción de notificación, para así poder limitar mediante banderas la activación de un tipo u otro de alerta al crear la notificación.

El archivo .xml que define la interfaz gráfica de esta pantalla, consta de los dos elementos siguientes:

- ToggleButton: Que permite la activación o desactivación de cada una de las opciones de alerta; un total de tres *ToggleButton* se emplean en esta pantalla.
- Button: Permite almacenar la configuración seleccionada.

## 5.2.5. Clases secundarias.

Tras haber analizado con detenimiento las clases principales asociadas a las distintas pantallas de la aplicación, es necesario realizar una descripción de las clases que componen el segundo bloque comentado en el apartado 5.1.

Estas clases, aunque no tienen asociación directa con la interfaz gráfica de la aplicación, están muy ligadas a las clases principales y sin estas el funcionamiento de la interfaz gráfica y de la aplicación no sería posible.

Estas clases son:

- Lista\_entrada.java.
- Notificar\_GPS.java.

- ReceptorProximidad.java.
- Lista\_entrada.java

En este archivo se auto contiene la clase del mismo nombre destinada a almacenar la estructura básica de cada elemento de la lista que muestra las notas guardadas.

Cada vez que se introduce una nota, se crea una nueva entrada en la lista que después mostrará la totalidad de notas almacenadas; cada entrada creada responde a un esquema o estructura fijo que está descrito en esta clase y que se compone de:

- String TextoC: Almacenará el contenido del campo nota.
- String TextoP: Almacenará la dirección introducida en el campo ubicación.

• Int pos: Almacenará la posición que ocupa la entrada dentro de la lista de notas.

```
package es.us.dit.ignaciogpp;
```

```
//Clase que contiene los metodos necesarios para inicializar una entrada en la lista,
```

```
public class Lista entrada {
```

```
private String textoC;
private String textoP;
private int pos;
```

I

1

}

public Lista\_entrada ( String textoP, String textoC, int posicion) {

```
this.textoC = textoC;
this.textoP= textoP;
this.pos=posicion;
}
public String get_textoC() { return textoC; }
public String get_textoP() {return textoP;}
```

public int get\_pos() { return pos; }

Ilustración 5.18. Código de la clase Lista\_entrada.

Esta clase es usada en la clase "ShowNotasActivity" para rellenar el elemento ListView que configura la vista en forma de lista de las notas almacenadas. Para ello se crea un array de elementos Lista\_entrada tal que así: ArrayList<Lista\_entrada> datos = new ArrayList<Lista\_entrada>(); Este array servirá para contener todas las notas almacenadas en la base de datos para después poder mostrarlas. El llenado de este array se hace utilizando el método "add" con los datos obtenidos de la base de datos como se observa a continuación: En apartados posteriores se verá como ese *array* sirve, mediante un método y un "adaptador de lista", para llenar la estructura de lista que nos ofrece *ListView*.

```
do {
    //Obtengo cada campo de cada entrada de la base de datos y lo almaceno en variables
    contenido = c.getString(c.getColumnIndex("content"));
    posicion = c.getString(c.getColumnIndex("position"));
    datos.add(new Lista_entrada(posicion, contenido, pos_lista));
    /*Añado lo obtenido a un array de datos que contendrá
    toda la información a transferir a la lista.*/
    pos_lista++;
} while (c.moveToNext());
```

Ilustración 5.19. Fragmento de código de la clase ShowNotasActivity.

• Notificar\_Gps.java

Esta clase constituye el centro de toda la aplicación ya que en ella es donde se crea y se gestiona todo lo relativo a los avisos por posicionamiento.

Esta clase extiende de "Service", es decir, una aplicación que se ejecuta en *background* o segundo plano sin que el usuario interactúe. Su uso es necesario ya que esta clase es empleada justo cuando se activa el *ToggleButton* de cada nota para activar su alerta y en ese punto el usuario no tiene por qué tener la aplicación abierta.

```
SharedPreferences prefs = getSharedPreferences("preferenciasMiApp", this.MODE PRIVATE);
Handler NotasB handler = new Handler NotasB(this);
int indice=intenc.getIntExtra("pos",0);/*Obtengo el indice de la nota en
la que se ha pulsado el boton de notificación*/
Cursor cursor = handler.getNotas();
Intent intent = new Intent (ACTION PROXIMITY ALERT);
this.radius= prefs.getInt("radio", 500);
cursor.moveToPosition(indice);
myLoc = (LocationManager) getSystemService(Context.LOCATION SERVICE);
    /*Pending intent es lanzado a OnReceive cdo se entra en el radio indicado.
    OnReceive recibe el intent que le pasamos a PendingIntent.*/
    /*Le asociamos a este intent el índice de posición de la nota para poder en
   OnReceive obtener su campo contenido.*/
    intent.putExtra("posic", indice);
   contador=globalVariable.getBandera registro rec();
   lanzar notificacion = PendingIntent.getBroadcast(this,globalVariable.
   getBandera registro rec(), intent, PendingIntent.FLAG CANCEL CURRENT);
    contador=contador+1;
    globalVariable.setBandera_registro_rec(contador);
    String situacion = cursor.getString(cursor.getColumnIndex("position"));
    cursor.close();
   handler.cerrar();
    this.estado conex=prefs.getBoolean("conexion", true);
    this.flag casa=prefs.getInt("Flag casa",0);
    this.flag trabajo=prefs.getInt("Flag trabajo",0);
    this.flag poi=prefs.getInt("Flag poi",0);
    try {
       if(this.estado conex==false){
```

Ilustración 5.20.Código de la clase Notificar\_GPS.

En la ilustración 5.20 se puede observar el código completo de esta clase cuya funcionalidad se resume a continuación: Al comienzo de la clase se definen todas las variables a usar, entre ellas observamos una variable de tipo entero llamada índice que se inicializa obteniendo información "extra" almacenada en el *intent* que recibe esta clase. Esta variable almacena la posición en la lista de la nota que ha activado la alerta.

Tras ello, se obtiene el radio de aviso que el usuario ha fijado así como la ubicación introducida para obtener las coordenadas necesarias para lanzar la alerta.

Una vez fijados los parámetros, se lanza la alerta mediante el método *addProximityAlert* que comprueba la posición actual y si está dentro del radio fijado lanza un *PendingIntent*, con el cometido de activarse en *broadcast* o difusión (necesitará de un receptor de este tipo de elementos). En su creación para cada llamada a la clase Notificar\_GPS, se varía el campo contador permitiendo así que las

alertas no se sobrescriban cuando se activa más de una al mismo tiempo.

• ReceptorProximidad.java

Esta clase constituye la otra parte de la clase anterior, en ella se reciben las alertas lanzadas desde Notificar\_GPS. Esta clase extiende de *BroadcastReceiver* ya que es el tipo de *PendingIntent* que vamos a recibir.

```
public class ReceptorProximidad extends BroadcastReceiver {
    private static final int NOTIFICATION ID = 1000;
   int luz;
    int vibra;
   int sound:
    @Override
    public void onReceive(Context context, Intent intent) {
        /*El intent que recibe agui es un pendingIntent generado en
         Notificar_GPS que lo lanza addproximityaert.*/
        final GlobalClass globalVariable =(GlobalClass) context.getApplicationContext();
        String key = LocationManager.KEY PROXIMITY ENTERING;
        Boolean entra = intent.getBooleanExtra(key, false);
        // Comprobamos si estamos entrando dentro del radio fijado.
        if (entra) {
            /*Obtiene la posicion a partir del intent almacenado en globalVariable.getIntent con
            el indice almacenado en el intent pasado por parametro..Ya tenemos el intent que
            activó el service y en su campo Flags tenemos la posicion de la nota.*/
            Bundle recoger=intent.getExtras();
            int indice=recoger.getInt("posic");
            if(globalVariable.getactivado(indice)==1) {
            //Si no hemos desactivado la notificación lanzada.
                Toast.makeText(context, "Notification in progress", Toast.LENGTH SHORT).show();
                Intent intent_llamada = globalVariable.getIntent(indice);
                NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION
                Intent notificationIntent = new Intent(context, Notificar GPS.class);
```

Ilustración 5.21. Fragmento de código de la clase ReceptorProximidad.

En la ilustración 5.21 se puede observar el primer fragmento de código de esta clase. En él se observa que comenzamos comprobando si se ha entrado en la proximidad fijada.

Una vez comprobamos que estamos dentro del área definida mediante el radio, obtenemos la posición de la nota que ha activado la alerta. Dicha posición la habíamos asociado al *intent* lanzado en la clase Notificar\_GPS; a continuación, comprobamos que siga activado el botón de activar notificaciones, ya que puede ocurrir que estemos llegando a la ubicación fijada y el usuario cancele la alerta fijada antes de que sea lanzada.

```
if (globalVariable.getactivado(indice) == ON) {
//Si no hemos desactivado la notificación lanzada.
   Toast.makeText(context, "Notification in progress", Toast.LENGTH_SHORT).show();
   Intent intent_llamada = globalVariable.getIntent(indice);
   NotificationManager notificationManager = (NotificationManager) context.getSystemService (Context.NOTIFICATION SERVICE);
   Intent notificationIntent = new Intent(context, Notificar_GPS.class);
   PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, notificationIntent, 0);
   Notification notification = createNotification(context);
   Handler_NotasB handler = new Handler_NotasB(context);
   final String nota;
   Cursor cursor = handler.getNotas();
   cursor.moveToPosition(indice):
   nota = cursor.getString(cursor.getColumnIndex("content"));
   notification.setLatestEventInfo(context, "Recuerda", "" + nota, pendingIntent);
    //Lanzo notificación.
   notificationManager.notify(NOTIFICATION ID, notification);
   globalVariable.setactivado(OFF, indice);//desactivo el TogleButton correspondiente.
   Activity cont = globalVariable.getContexto();//Obtengo el contexto de la llamada,
   cursor.close();
   handler.cerrar():
    cont.finish();
   globalVariable.setPosicion(indice);
   context.stopService(intent_llamada);
      }
```

Ilustración 5.22. Fragmento de código de la clase ReceptorProximidad.

En la ilustración 5.22 se puede observar la segunda parte del código de la clase "ReceptorProximidad", una vez realizadas las comprobaciones anteriores se hace una llamada al método "createNotification", incluido en este archivo, que se encarga de crear la notificación con la configuración fijada por el usuario en el submenú de ajustes destinado para ello.

Una vez creada, lanzamos la notificación al usuario y tras ello desactivamos el botón destinado a activar la alerta y cerramos los distintos manejadores así como detenemos el servicio iniciado en "Notificar\_GPS".

Hay que destacar que para que el receptor realice correctamente su labor ha de estar registrado como receptor. En este caso, por simplicidad, el receptor se ha registrado añadiendo unas líneas en el fichero *manifest* como se aprecia en la ilustración 5.23 (Ver Anexo B), aunque podía haber sido declarado en el propio código.

```
android:value="AlzaSyCueXWneAeMtlsh64GUkwVKsC_4W5mRGts" />
<service android:name=".Notificar_GPS" />
<receiver android:name="es.us.dit.ignaciogpp.ReceptorProximidad">
<intent-filter>
</action android:name="es.us.dit.ignaciogpp" />
</intent-filter>
</receiver>
```

Ilustración 5.23. Registro del receptor en Android Manifest.

## 5.2.6. Clases auxiliares.

En apartados anteriores se ha analizado las clases asociadas a las distintas pantallas de la aplicación así como las clases que realizan las operaciones necesarias para implementar la funcionalidad de las distintas pantallas.

En este apartado se describen las clases que en el apartado 5.1.1 clasificamos en el bloque tercero es decir, clases que dan soporte a algunas de las gestiones que necesitan el resto de clases. Estas clases son:

- GlobalClass.java
- Handler\_NotasB.java
- Lista\_adaptador.java
- PlacesAutoCompleteAdapter.java
- GlobalClass.java

Esta clase contiene todas las variables globales necesarias en las clases restantes para el desarrollo y funcionamiento de la aplicación.

Además de la declaración e inicialización de dichas variables, se incluyen los métodos *getter* y *setter* para poder modificar y obtener el valor de cada una de ellas.

#### • Handler\_NotasB.java

}

Esta clase se encarga de la creación y gestión de la base de datos asociada a la aplicación.

```
public void onCreate(SQLiteDatabase db){
    String q ="CREATE TABLE notas(ID INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT, position TEXT, content TEXT)";
    db.execSQL(q);
```

```
Ilustración 5.24. Sentencia de creación de la base de datos.
```

Mediante la sentencia que se muestra en la ilustración 5.24 se crea la base de datos con los campos descritos, destacando el campo ID el cual tiene la orden de incrementarse automáticamente por cada entrada almacenada en la base de datos.

Entre los métodos incluidos se encuentran:

- AddNotas: Método que inserta una nueva entrada en la base de datos.
- getNotas: Método que devuelve un cursor para seleccionar la posición de la entrada deseada en la base de datos.
- SetValues: Método que permite modificar una entrada determinada de la base de datos.
- RemoverNota: Elimina de la base de datos la nota seleccionada pasándole su campo ID.

#### • Lista\_adaptador.java

Esta clase se crea para facilitar la tarea de gestión de la lista, en ella se ofrecen métodos que nos permiten listas en base a un *array* de elementos.

Empleamos esta clase sobre escribiendo un método para mapear los datos del *array* a cada entrada de nuestra lista como se puede ver en la ilustración 5.25.

```
lista.setAdapter(new Lista adaptador(this, R.layout.activity shownotas, datos) {
                     @Override
                     public void onEntrada(Object entrada, View view) {
                         if (entrada != null) {
                             TextView texto_inferior_entrada = (TextView) view.findViewById(R.id.verContenido);
                             if (texto_inferior_entrada != null)
                                texto_inferior_entrada.setText(((Lista_entrada) entrada).get_textoC());
                             TextView texto_centro = (TextView) view.findViewById(R.id.verPosicion);
                             if (texto_centro != null)
                                 texto_centro.setText(((Lista_entrada) entrada).get_textoP());
                             ToggleButton boton= (ToggleButton) view.findViewById(R.id.boton_act);
                             boton.setTag(((Lista_entrada)entrada).get_pos());
                             ImageButton boton1=(ImageButton)view.findViewById(R.id.deLete);
                             boton1.setTag(((Lista_entrada)entrada).get_pos());
                             /*Para cada botón le asigno su posición en la lista
                             La primera vez que se muestra la lista de notas
                             la var. global Bandera está a 0.1a segunda vez
                             que entra se pone a 1 e indica que ha podido haber
                             cambio en los toggleButton y que hay que actualizar la vista de estos*/
                             if(globalVariable.getBandera() == ON) {
                              if (globalVariable.getactivado(((Lista_entrada) entrada).get_pos()) == ON) {
                                boton.setChecked(true):
                                  //si el botón se ha pulsado anteriormente se actualiza su vista.
                                1
                            if (globalVariable.getactivado(((Lista_entrada) entrada).get_pos()) == OFF) {
                                boton.setChecked(false);
```

Ilustración 5.25.Fragmento de código donde se sobre escribe el método OnEntrada de Lista\_Adaptador.java

#### • PlacesAutoCompleteAdapter.java

Esta clase se crea para gestionar la función de autocompletado incluida en la pantalla principal y en la pantalla "localizaciones" del submenú ajustes.

En esta clase, se realiza una consulta a una dirección de Google Places de forma simultánea a la introducción de caracteres en el campo correspondiente (que debe estar declarado en el archivo .xml correspondiente como *AutoCompleteTextView*) y proporciona los resultados más precisos en formato de lista desplegada encima o debajo del recuadro de introducción.

En la ilustración 5.26, se muestra su funcionamiento.



Ilustración 5.26. Funcionamiento de la herramienta autocompletado.

Para que esta funcionalidad sea posible, será necesario activar la API de Google Places en la consola de GoogleDevelopers así como obtener mediante dicha consola una clave de servidor que introduciremos en la constante estática "API\_KEY" de esta clase para poder realizar las consultas necesarias.

# 5.3. Resumen estructural de la aplicación.

En este apartado se presenta una correspondencia en formato gráfico entre las fases definidas en la ilustración 4.4 (apartado 4.5) y las distintas pantallas y clases de la aplicación definidas en el apartado anterior.

#### Fase I

- •Pantalla Principal (5.2.1)
- •Pantalla de visualización de notas (5.2.3)
- •Clases secundarias (5.2.5): Lista\_entrada.java
- •Clases auxiliares(5.2.6): GlobalClass.java Handler\_NotasB.java Lista\_adaptador.java PlacesAutoCompleteAdapter.java

#### Fase II

- •Pantalla Mapa (5.2.2)
- •Pantalla Principal (5.2.1) •Clases auxiliares (5.2.6): *GlobalClass.java*
- GlobalClass.java Handler\_NotasB.java

#### Fase III

- •Clases secundarias (5.2.5): Notificar\_Gps.java ReceptorProximidad.java
- •Clases auxiliares(5.2.6): GlobalClass.java Handler\_NotasB.java



Menú y pantallas de ajustes (5.2.4)

Ilustración 5.27. Diagrama de relaciones: fases proyecto / clases y pantallas empleadas.

# 6 PROBLEMAS Y SOLUCIONES



continuación se exponen las dificultades más relevantes encontradas durante el desarrollo de la aplicación GeoNudge. Tras cada una de estas dificultades se resume la solución adoptada.

Las dificultades encontradas son las siguientes:

- Soporte para distintos dispositivos.
- Soporte para distintos idiomas.
- Soporte para orientación horizontal.
- Ajustes de elementos gráficos.
- Problemas con versiones de Android.

## 6.1. Soporte para distintos dispositivos

Este problema surge a raíz de la realización de pruebas de funcionamiento en múltiples dispositivos Android, en dichas pruebas se comprobó que el ajuste de las diferentes pantallas de la aplicación a la densidad y resolución de la pantalla de cada dispositivo no era óptimo.

Los problemas encontrados fueron los siguientes:

- Desalineación de los elementos.
- Falta de visibilidad de algunos elementos como títulos y botones.
- Paleta de colores que imposibilitaba la visión de otros componentes.

Para dar solución a dichos problemas, se procedió como sigue:

- 1) Situados en el directorio /res/layout se crean dos carpetas para los distintos tamaños de pantalla estandarizados para Android:
  - /res/layout-large.
  - /res/layout-normal.



Ilustración 6.1.Estructura de directorios de la carpeta /Res/.

2) En cada carpeta se copian los distintos archivos .xml que componen las pantallas de la aplicación.



Ilustración 6.2. Archivos .xml en los diferentes directorios layout.

- 3) Para cada archivo .xml de cada carpeta se realizan los cambios siguientes:
  - Alineamiento de los distintos elementos.
  - Cambio del color y tamaño de la fuente para títulos.
  - Cambio de color de fondo para permitir la correcta visualización de todos los componentes.

Para realizar estos cambios utilizamos la vista *Design* de la herramienta de edición de *layouts* de Android Studio, que permite observar el resultado en las pantallas de los distintos dispositivos.



Ilustración 6.3. Herramienta de edición de layouts de Android Studio.

# 6.2. Soporte para distintos idiomas

La aplicación GeoNudge, incorpora un sistema de traductor de la aplicación en un total de cinco idiomas (español, inglés, francés, italiano y alemán).

El usuario debe tener configurado el terminal en el idioma preferido por el usuario y la aplicación lo detectará adaptando sus textos y menús al idioma correspondiente o en su defecto al inglés.

Para llevar a cabo esta solución se incorporan cinco carpetas nuevas en el directorio de trabajo de la aplicación, como se observa en la siguiente figura:



Ilustración 6.4. Arbol de carpetas del directorio "values"

Cada carpeta representa un idioma, siendo la carpeta "values" aquella que contiene los archivos en el idioma por defecto. Cada una de las restantes carpetas contiene idénticos archivos pero en la lengua marcada por el código del país incluido en el título de la misma, recogiéndose en el fichero "strings.xml" de cada una de ellas una traducción de todos los textos en el idioma correspondiente.

## 6.3. Soporte para orientación horizontal

Este problema guarda relación con el descrito en el apartado 6.1, al incluir la posibilidad de utilizarla aplicación en distintos dispositivos, surge también la problemática del giro o rotación de la pantalla en estos dispositivos.

Para ello, se ha diseñado unos nuevos archivos *layout* que siguiendo las líneas de diseño de los *layouts* por defecto (los que tienen orientación vertical), distribuyen los elementos para una configuración horizontal de la pantalla.

Esto ha sido posible agregando estos archivos a unos directorios nombrados con la terminación "-land" (del inglés *landscape* o apaisado) que dependen a su vez de los directorios comentados en el apartado 6.1. En la siguiente ilustración podemos observar el árbol de directorios correspondientes a la orientación horizontal de la pantalla.



Ilustración 6.5. Arbol de directorio de carpetas "-land"

## 6.4. Ajuste de elementos gráficos

Aunque ya se comentó algo relacionado en el apartado 6.1, al realizar pruebas de compatibilidad en los diferentes dispositivos nos encontramos con la problemática de que los elementos fijados en los *layouts* podían desplazarse provocando el mal funcionamiento de la aplicación.

Esto sucedía debido a que en primera instancia se realizó un diseño estático para un determinado *Smartphone* fijando todas las dimensiones con respecto a este terminal. Este primer diseño al ejecutarlo sobre una Tablet era completamente inservible ya que no se ajustaban los elementos a las dimensiones de la pantalla.

La solución adoptada finalmente ha sido emplear *RelativeLayouts*, una variable del contenedor de elementos gráficos básico (o *layout*) que permite situar los elementos y sus dimensiones en función del resto de elementos y de la pantalla.

## 6.5. Problemas con versiones de Android

Otro problema que se encontró durante el desarrollo de la aplicación fue la compatibilidad de las clases y métodos usados en las diferentes versiones de Android.

Se ha probado con éxito en distintos terminales con distinta versión del sistema, pero en concreto se han encontrado problemas en la versión 4.1.2 en la que reportaba fallos al emplear un método de Google que convierte de dirección a coordenadas.

Según se ha comprobado, existen usuarios a los que tras una actualización de las herramientas de Google de su terminal y un reseteo, la aplicación ha funcionado sin error. Por el contrario, también hay casos en los que no se ha podido solucionar. Este fallo fue reportado a Google hace unos años sin ninguna novedad.

Es por esto, por lo que se recomienda actualizar las herramientas de Google del terminal, no garantizándose el funcionamiento correcto en esa versión de Android.

# **7** PLAN DE PRUEBAS

### n este apartado se describen las pruebas realizadas en los distintos dispositivos. Hay que reseñar que al ser una aplicación con un gran componente dependiente del usuario, no ha sido posible la creación de una herramienta automática de prueba de la misma.

## 7.1. Pruebas realizadas.

Se ha realizado un plan de pruebas compuesto por:

• Prueba 1: Añadir una nota nueva sin ubicación.

Se comprueba la funcionalidad como bloc de notas.

• Prueba 2: Añadir una nota nueva con ubicación definida manualmente.

Se comprueba el uso de la función autocompletado de Google, la localización en el mapa y el sistema de recordatorios con radio por defecto.

#### • Prueba 3: Modificar dirección añadida manualmente mediante el mapa.

Se comprueba la posibilidad de modificarla ubicación y el correcto funcionamiento del sistema de recordatorios tras esta modificación.

#### • Prueba 4: Borrado de nota individual.

Comprobación de la herramienta borrado de una nota del listado.

#### • Prueba 5: Borrado de lista completa.

Comprobación de la herramienta borrado del listado completo.

#### • Prueba 6: Pruebas sobre gráficos y orientaciones de pantalla.

Comprobación de ajuste correcto de los elementos gráficos en los distintos tamaños y orientaciones.

#### • Prueba 7: Uso de ajustes.

Comprobación del correcto funcionamiento de los ajustes.

#### • Prueba 8: Uso de localizaciones pre guardadas.

Comprobación del correcto funcionamiento y almacenaje de las ubicaciones pre guardadas.

#### • Prueba 9: Uso de localizaciones pre guardadas en modo offline.

Comprobación del correcto funcionamiento del modo *offline* disponible con el uso de ubicaciones pre guardadas.

#### • Prueba 10: Comportamientos no habituales.

Pruebas varias acerca de casos tales como: dirección no encontrada, campo nota vacío, uso de la comprobación de conexión disponible en *Tablets*...

# 7.2. Dispositivos empleados.

A continuación se muestra el listado de dispositivos empleados en las pruebas realizadas.

- Samsung Galaxy Mini 2: Android v2.3 Gingerbread, pantalla de 3.27".
- Bq Aquaris E 4.5: Android v4.4, pantalla de 4.5".
- Sony ericcsson LT26i: Android 4.1.2, pantalla de 4,3".
- Hisense u939: Android v4.2, pantalla de 4.5".
- Sony Tablet S: Android v3.2 Honeycomb, pantalla de 9.4".

# 8 CONCLUSIONES Y LÍNEAS DE CONTINUACIÓN

*ara finalizar*, se desarrollan en este apartado unas breves conclusiones acerca de la aplicación y su desarrollo, así como unas posibles líneas de mejora para su continuación.

# 8.1. Conclusiones

Una vez expuesto en el presente documento toda la información acerca de la aplicación GeoNudge, su desarrollo y su uso, es momento de realizar una valoración destacando las conclusiones relevantes de su creación.

GeoNudge es una herramienta práctica y útil que actualmente ofrece una funcionalidad inédita en el resto de aplicaciones, la alerta por posicionamiento. En este sentido la aplicación ha alcanzado plenamente el objetivo de diseño fijado permitiendo la edición de la posición de manera intuitiva así como una configuración sencilla de sus parámetros que hacen que sea manejable por todos los usuarios sin necesidad de conocimientos previos.

Además de ofrecer esta funcionalidad de recordatorios, el hecho de implementar un bloc de notas y un modo *offline* complementan considerablemente las posibilidades de funcionamiento de la aplicación.

Por otro lado para el desarrollo de la misma hemos encontrado dificultades añadidas a las definidas en el apartado anterior como son:

- Dificultades relativas a la planificación temporal.
- Dificultades relativas a la herramienta de desarrollo.
- Dificultades relativas a funcionalidad.
- Dificultades relativas a la planificación temporal:

En estas englobamos todo el desfase temporal acontecido durante el desarrollo del proyecto.

Según avanzaba el proyecto la planificación temporal ha sido compleja debido a las distintas necesidades no planteadas previamente que la aplicación debía garantizar (fallos de programación, casuísticas posibles en cuanto a la funcionalidad...).

En este aspecto el desarrollo de la aplicación ha ido sufriendo retrasos variables que previamente no se habían contemplado.

• Dificultades relativas a la herramienta de desarrollo:

Aunque la aplicación se ha desarrollado sobre Android Studio, existían algunas alternativas a este pero sin embargo la decisión recayó sobre Android Studio debido a las funcionalidades que incorpora como pueden ser por ejemplo los emuladores.

Estos emuladores, requieren de un equipo con gran capacidad de procesamiento con lo que la decisión de Android Studio conllevó a trabajar en un equipo con mejor rendimiento.

A pesar de ello, los distintos emuladores han sido de gran ayuda para la configuración gráfica de la aplicación.

• Dificultades relativas a la funcionalidad:

Como se comentaba en el apartado anterior, no en todas las versiones Android está garantizado el correcto funcionamiento de la aplicación, y esto es debido a que se emplean servicios y componentes de Google, muchos de ellos en continuo proceso de mejora.

Esto ha provocado que existan algunas dificultades por el momento irresolubles como por ejemplo el uso de "GeoCoder" para la conversión a coordenadas que provoca error en la versión 4.1.2.

## 8.2. Líneas de continuación.

En cuanto a las líneas de continuación, podríamos dividir las mismas en dos grupos:

- Características vinculadas con la funcionalidad actual.
- Características no vinculadas con la funcionalidad actual.
- Características vinculadas con la funcionalidad actual.

Si nos centramos más en las características actuales, GeoNudge dispone de un menú de ajustes de opciones sencillas que podría ser mejorado dando más protagonismo al usuario de la aplicación haciendo que este pueda elegir un tono de notificación personalizado o definir el color de la notificación luminosa.

También existen líneas de mejora en lo relativo a la notificación que actualmente es mediante posicionamiento pero podría incluirse la posibilidad de notificación por conexión *bluetooth* con otros dispositivos, notificación por conexión a red WiFi o por último notificación por proximidad a otro terminal que disponga de la aplicación.

• Características no vinculadas con la funcionalidad actual.

Por otro lado alejándonos de las características actuales, GeoNudge, gracias a su diseño práctico, puede ser incluida como herramienta complementaria de otras aplicaciones de los distintos ámbitos de la comunicación y entretenimiento.

Cabe reseñar por último que GeoNudge es un bloc de notas y no añade características de edición, funcionalidad que complementaría potencialmente la aplicación, por tanto esta podría ser otra línea de mejora posible.

- 1. Curso de programación Android. Consulta web: http://www.sgoliver.net/blog/curso-de-programacion-android/
- 2. Estructura básica de un proyecto Android. Consulta web: <u>http://javiergarbedo.es/80-android/primeros-pasos/338-estructura-de-un-proyecto-</u> <u>en-android-studio</u>
- 3. Creación de menús en Android. Consulta web: http://androideity.com/2011/09/05/creando-menus-en-android/
- 4. Documentación Android. Consulta web: http://developer.android.com/index.html
- 5. Android Studio. Consulta web: https://developer.android.com/sdk/index.html
- 6. Foro de consultas. Consulta web: <u>http://stackoverflow.com/</u>
- Mapas en Android. Consulta web: <u>http://www.sgoliver.net/blog/mapas-en-android-google-maps-android-api-v2-i/</u>
- 8. Hilo acerca de problema de GeoCoder en algunas versiones. Consulta web: <u>https://code.google.com/p/android/issues/detail?id=38009</u>
- 9. Ajustes gráficos y otras consultas. Consulta web: <u>http://cursoandroidstudio.blogspot.com.es/2014/07/rotate-rotacion.html</u>
- 10. Foro y portal web con documentación de Android en Español. Consulta web: <u>http://www.androidsis.com/</u>

# Anexo A: GeoNudge

Manual de uso



# Arrancando la aplicación

Para arrancar la aplicación, buscaremos en nuestro dispositivo el icono 🕝 y pulsamos sobre él.



Una vez abierta la aplicación, aparecerá un mensaje advirtiéndonos de que hemos de activar la localización GPS en nuestro dispositivo. Automáticamente se nos conduce a la pantalla que se muestra en la imagen superior donde tendremos que marcar la casilla que dice "Utilizar satélites GPS".



Tras esto, pulsamos el botón atrás is de nuestro dispositivo y aparecerá la pantalla principal mostrada en la imagen izquierda. En ella podemos elegir agregar nota nueva o ver todas las notas, mediante los botones destinados a cada opción.

# Agregando nota nueva.



Para agregar una nota nueva, rellenamos los campos nota y ubicación (si deseamos alertas de proximidad) y pulsamos sobre el botón "Agregar nota nueva".

GeoNudge			19.30
orueba			04
Camino de l	los Descubrimi	ientos, Sevi	X Te, Spain
orueba			OFF
			×
Camino de l	los Descubrimi	entos, Sevi	le, Spain
			OFF
prueba3			×
orueba3			×



Nos aparecerá la vista del mapa con un *marker* indicándonos la situación introducida, si estamos conformes pulsamos siguiente. Si queremos modificar la ubicación no tenemos más que pulsar sobre la calle que deseemos en el mapa.<sup>2</sup>



Comprobamos que la nota se ha añadido correctamente.

El campo nota debe ser rellenado para poder agregar una nota.

<sup>&</sup>lt;sup>2</sup>En el caso que la dirección introducida no se encontrase, aparecerá un mensaje informando de ello y se nos reconducirá a la pantalla principal.

Si cuando se realiza la descarga del mapa y situación del *marker* la conexión de datos fallase, se nos reconducirá a la pantalla principal.

# Activando notificación.



# Desactivando notificación.



Nos situamos en la pantalla de visualización de notas y pulsamos sobre el botón señalado en la imagen.



Observamos como el botón se cambia a OFF, apareciendo también un mensaje informándonos de que hemos desactivado las notificaciones para esa nota.



# Eliminando nota.



Nos situamos en la pantalla de visualización de notas y pulsamos sobre el botón señalado en la imagen.



Observamos como la nota se elimina\*, apareciendo también un mensaje informándonos de que se está eliminando dicha nota.

\*En caso de que eliminemos la única nota que quedase almacenada, se nos reconducirá a la pantalla principal.



# Eliminando todas las notas.



Nos situamos en la pantalla de visualización de notas y pulsamos sobre el botón "menú" del dispositivo.

Se nos desplegará un menú contextual y pulsando sobre el botón "Eliminar todo" podremos borrar todas las notas almacenadas.

Aparecerá un mensaje en pantalla informando de que el borrado ha sido exitoso y se nos reconducirá la pantalla principal.

# Configurando la aplicación: Submenú ajustes.



# Configurando la aplicación: Localizaciones Preferidas.



Pulsando sobre el botón "localizaciones" del menú accederemos a la pantalla, donde podremos ver las direcciones guardadas e introducirlas de nuevo.

# Haciendo uso de las localizaciones guardadas.



Para hacer uso de las localizaciones almacenadas previamente, tan solo habrá que pulsar sobre el botón señalado y se nos desplegará el cuadro de selección de la imagen siguiente.



Seleccionamos, pulsando sobre cada uno, la ubicación favorita que queremos asociar a la nota.



Configurando la aplicación: Radio de aviso.



Pulsando sobre la opción "Radio de aviso" accederemos a la pantalla mostrada en la imagen izquierda.

Insertamos el valor deseado para el radio de aviso (en **metros**) y pulsando sobre el botón "OK" guardaremos esta configuración apareciendo un mensaje informándonos de ello.

# Configurando la aplicación: Notificaciones.



# Recibiendo una notificación.

	• =			
09/02/2015	۵ ۲	,∏;≑	atl 📋	😭 12:37
<b>?</b>	*	$\diamond$	*	5
Wi-Fi Bl	uetooth	GPS	Datos	Rotación auto
Orange				Borrar
Notificacio	ones		_	
A Recue	erda			12:36
		=		
	C		)	±

Pulsando sobre la opción "Notificaciones" accederemos a la pantalla mostrada en la imagen izquierda.

Pulsando sobre cada botón activamos cada uno de las posibles opciones de notificación, iluminándose cada botón y cambiando a ON.

Está configuración así como todas las anteriores, será guardada mientras la aplicación siga instalada en el dispositivo.

Cuando nos acerquemos a la ubicación fijada y entremos dentro del radio de aviso, se activará la notificación mostrando un mensaje similar al de la imagen además de notificarnos mediante las distintas opciones configuradas a través del submenú "notificaciones".

## Restricciones sobre su uso

- La aplicación requiere conexión a internet y posicionamiento GPS para desempeñar la totalidad de sus funcionalidades.
- El funcionamiento en modo offline o sin internet sólo es posible utilizando previamente las ubicaciones predefinidas.
- Se requiere actualizar Google Maps así como todos sus servicios para el correcto funcionamiento de la misma.
- En la versión de Android 4.1.2 no se garantiza el correcto funcionamiento de la aplicación.
- En pantallas inferiores a 3.4" no se garantiza el correcto ajuste gráfico de la aplicación.
## Anexo B: Android Manifest

```
<?xml version="1.1" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
         package="es.us.dit.ignaciogpp" >
          <application
                     android:allowBackup="true"
                     android:icon="@drawable/ic launcher"
                     android:label="@string/app name"
                     android:theme="@style/Italic"
                     android:name="es.us.dit.ignaciogpp.GlobalClass">
                     <activity
                               android:name=".MyActivity"
                               android:label="@string/app_name" >
                               <intent-filter>
                                          <action android:name="android.intent.action.MAIN" />
                                          <category
android:name="android.intent.category.LAUNCHER" />
                               </intent-filter>
                     </activity>
                     <activity
                               android:name=".milocalizaciones"
                               android:label="@string/app name" >
                     </activity>
                     <activity
                               android:name=".ShowNotasActivity"
                               android:label="@string/app name" >
                     </activity>
                     <activity
                               android:name=".radiomenu"
                               android:label="@string/app name" >
                     </activity>
                     <activity
                               android:name=".notificacionesmenu"
                               android:label="@string/app name" ></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></activity></act
                     <activity
                               android:name=".MapsActivity"
                               android:label="@string/title activity Maps" >
                     </activity>
                     <meta-data
                               android:name="com.google.android.gms.version"
                               android:value="@integer/google_play_services_version" />
                     <meta-data
                               android:name="com.google.android.Maps.v2.API KEY"
                               android:value="AIzaSyCueXWneAeMfIsh64GQkwVKsC 4W5mRGfs" />
```

```
<service android:name=".Notificar GPS" />
        <receiver
android:name="es.us.dit.ignaciogpp.ReceptorProximidad">
        <intent-filter>
            <action android:name="es.us.dit.ignaciogpp" />
        </intent-filter>
        </receiver>
    </application>
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true"></uses-feature>
//Permisos necesarios
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
android:name="android.permission.WRITE EXTERNAL STORAGE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ GSERVIC
ES" />
    <!--
         The ACCESS COARSE/FINE LOCATION permissions are not required
to use
         Google Maps Android API v2, but are recommended.
    -->
    <uses-permission
android:name="android.permission.ACCESS COARSE LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS FINE LOCATION" />
    <uses-permission android:name="android.permission.VIBRATE" />
```

```
</manifest>
```

Desde estas líneas transmitir mi más sincero agradecimiento, en primer lugar, a D. Juan Manuel Vozmediano Torres por su trabajo, consejos y paciencia a lo largo de todo el desarrollo del proyecto fin de carrera.

A mi familia por su apoyo constante y su preocupación a lo largo de toda mi vida, especialmente en esta etapa universitaria.

Por último, agradecer la labor de todo el profesorado y personal de la Escuela Técnica Superior de Ingeniería de Sevilla ya que, en mayor o menor medida, han contribuido a mi formación académica y personal durante mi aprendizaje universitario.