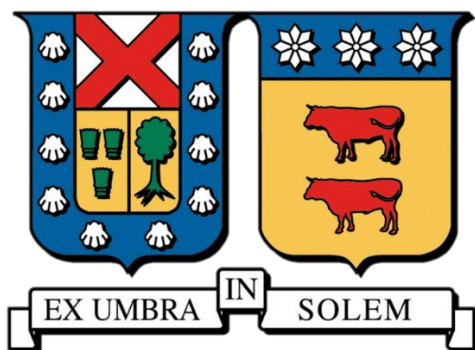


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAISO - CHILE



**“DESARROLLO DE UNA APLICACIÓN PARA DISPOSITIVOS CON SISTEMA
OPERATIVO ANDROID, ORIENTADA A DISMINUIR LA INCERTIDUMBRE EN LA
MOVILIDAD DEL CIUDADANO”**

DIEGO ORLANDO HINOJOSA CÓRDOVA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO MENCIÓN
COMPUTADORES

PROFESOR GUÍA: MARCOS ZUÑIGA BARRAZA

PROFESOR CO-REFERENTE: WALTER GROTE HAHN

ABRIL - 2017

RESUMEN

Esta memoria trata sobre el desarrollo de una aplicación para dispositivos móviles, como solución al desafío presentado por la organización Unidad de Ciudades Inteligentes de la Subsecretaría de Transportes, en el marco del nuevo Piloto de Memorias Multidisciplinarias, proyecto MECESUP “Fortalecimiento del desarrollo de competencias transversales en la formación profesional UTFSM”.

Como objetivo se tiene disminuir la incertidumbre del ciudadano en cuanto a su movilidad, entregando información relevante, integral y confiable en todo momento, complementando la plataforma web de la iniciativa transporTeinforma de la Subsecretaría de Transportes para aportar al desarrollo de ciudades más inteligentes, en este caso en la Región de Valparaíso.

Palabras claves: Aplicación móvil, Android, transporte, movilidad.

ABSTRACT

The present project is about the development of an app for mobile devices with Android operating system, as the solution to the challenge presented by the Smart City Unit of the Transport Ministry of Chile, within the experimental multidisciplinary thesis framework of the MECESUP project known as “Reinforcement of the Development of Transversal Skills in the professional formation at UTFSM”.

The app main objectives are to decrease the citizens’ uncertainty regarding their mobility, delivering relevant, integral and reliable information at all time; and complement the web service of the Transport Ministry’s initiative called “transporTeinforma” for the development of smarter cities, in this case only considering the Valparaíso region.

Keywords: Mobile application, Android, transport, mobility.

GLOSARIO

Activity: Actividad, representa una pantalla en una aplicación Android.

Android Manifest: Archivo en el que se definen los rasgos principales de una aplicación en Android, como lo son el nombre, icono, permisos, entre otros.

Fragmento: Porción de interfaz de usuario o un comportamiento en una actividad.

GPS: Sistema de Posicionamiento Global, nombre de un sistema de navegación y localización mediante satélites.

Intent: Un intent es un mensaje que podemos utilizar para solicitar una acción específica o pasar datos entre actividades.

Layout: Es un fichero XML que describe la composición de la interfaz de usuario.

SDK: Del inglés software development kit, se refiere al conjunto de herramientas para desarrollar las aplicaciones.

View: Objeto que se utiliza para la creación de elementos en la interfaz de la aplicación.

XML: del inglés extensible markup language, lenguaje de marcas extensible, es utilizado para definir los layouts y las vistas.

Índice general

RESUMEN	I
ABSTRACT	II
GLOSARIO	III
Índice de Figuras	VI
Índice de tablas	VII
Índice de códigos	VII
1. INTRODUCCION	1
1.1. MARCO GENERAL.....	1
1.1.1. Desafío o Problema a solucionar	1
1.2. MOTIVACION.....	2
1.3. OBJETIVOS GENERALES.....	2
1.4. OBJETIVOS ESPECÍFICOS	3
1.5. ESTRATEGIA.....	3
2. ESTADO DEL ARTE.....	4
2.1. Análisis.....	8
3. DESARROLLO DE APLICACIÓN	10
3.1. ANÁLISIS DE REQUERIMIENTOS.....	10
3.1.1. Requerimientos No Funcionales.....	10
3.1.2. Requerimientos Funcionales	10
3.1.3. Identificación de Actores del Sistema	11
3.1.4. Modelo de casos de Uso.....	12
3.1.5. Especificaciones de casos de Uso	13
3.2. DISEÑO DE PROTOTIPOS DE USABILIDAD.....	18
3.3. CLASES IMPORTANTES.....	21
3.4. MAPAS DE NAVEGACIÓN	22
3.4.1 Detalle del mapa de navegación.....	23
4. IMPLEMENTACIÓN	24
4.1. DESARROLLO DE LA INTERFAZ	24
4.2. USO DE LISTA EN LA APLICACIÓN	26
4.2.1. Implementación de Filtro en la Lista	31
4.3. USO DE MAPAS EN LA APLICACIÓN	33

4.3.1.	Implementación de la Activity que muestra el Mapa	35
4.4.	OBTENCIÓN DE DATOS DESDE PÁGINA WEB	36
4.5.	ALGORITMO DE FILTRADO DE INCIDENTES.....	43
5.	PRUEBAS Y RESULTADOS.....	44
5.1.	ANÁLISIS DE RESULTADOS OBTENIDOS.....	44
6.	CONCLUSIONES Y TRABAJOS FUTUROS	47
6.1.	CONCLUSIONES	47
6.2.	TRABAJOS FUTUROS	48
7.	REFERENCIAS	49
8.	ANEXO.....	50
8.1	Archivo XML proveniente de TransporTeinforma (Ejemplo breve).....	50
8.2	Códigos de clases devueltas por http://pojo.sodhanalibrary.com	52
8.2.1	Clase Result.java	52
8.2.2	Clase Xml.java	53
8.2.3	Clase ListaIncidencias.java	53
8.2.4	Clase Incidencia.java.....	54
8.2.5	Clase Coordenadas.java	56

Índice de Figuras

Figura 1: Captura de pantalla del sitio web de la UOCT región de Valparaíso.	4
Figura 2: Captura de pantalla del sitio web de transporTeinforma, selección de región.	5
Figura 3: Captura pantalla del sitio web transporTeinforma Valparaíso.	5
Figura 4: Logo de la aplicación waze.	6
Figura 5: Capturas de pantalla de la aplicación waze.	6
Figura 6: Logo de la aplicación Moovit.	7
Figura 7: Capturas de pantalla de la aplicación Moovit.	7
Figura 8: Diagrama de casos de uso.	13
Figura 9: Primeros sketching realizados a mano.	19
Figura 10: Idea de mapa de incidentes.	20
Figura 11: Idea de lista de incidentes.	20
Figura 12: Storyboard de pantallas principales.	21
Figura 13: Clases de mayor importancia.	22
Figura 14: Mapa de navegación.	23
Figura 15: Interfás simple.	25
Figura 16: Bosquejo de layout para cada item de la lista.	26
Figura 17: Interfaz de la Activity que contiene el ListView.	28
Figura 18: Diagrama de funcionamiento del Adaptador de incidentes.	28
Figura 19: Lista implementada con contactos.	31
Figura 20: Agragando una nueva Activity de tipo GoogleMaps.	33
Figura 21: Captura de pantalla de activación de Google Maps API.	34
Figura 22: Funcionamiento de marcadores de incidentes sobre mapa.	36
Figura 23: Esquema de funcionamiento de Retrofit. (Sacado de www.androidwarrior.com)	37
Figura 24: Lista de incidentes obtenidos de plataforma web.	42
Figura 25: Diagrama de filtro de incidentes.	43
Figura 26: Pantalla de inicio antes (Valpo Informa) y después (INCIT).	44
Figura 27: Pantallas de la aplicación antes y después.	45
Figura 28: Pantallas de la aplicación antes y después.	45
Figura 29: Página donde están alojados incidentes de prueba.	46

Índice de tablas

Tabla 1, Requerimientos no funcionales	10
Tabla 2, Tabla de requerimientos funcionales	11
Tabla 3, Actores del sistema.....	12
Tabla 4, caso de uso "Autenticar usuario".	14
Tabla 5, caso de uso "Ver incidentes cercanos".	14
Tabla 6, caso de uso "Cambiar radio de búsqueda".	15
Tabla 7, caso de uso "Filtrar por radio de búsqueda".	15
Tabla 8, caso de uso "Ver mapa con incidentes".....	15
Tabla 9, caso de uso "Ver detalles de incidente".....	16
Tabla 10, caso de uso "Compartir incidentes".....	17
Tabla 11, caso de uso "Abrir incidentes compartidos".	17
Tabla 12, caso de uso "Cerrar sesión".	18

Índice de códigos

Código 1: Interfaz simple en XML.....	24
Código 2: Layout XML para cada ítem de la lista.	27
Código 3: en Java del Adaptador personalizado.....	29
Código 4: Actividad donde se asocia el Adapter al ListView	30
Código 5: Implementación de objeto de tipo filtro.	32
Código 6: Permisos para geo localización.....	34
Código 7: Ubicación por defecto del marcador.	35
Código 8: Permiso para acceder a Internet	37
Código 9: Dependencias de la biblioteca retrofit	38
Código 10: Extracto de código XML devuelto por la plataforma web	39
Código 11: API encargada de realizar la petición de los incidentes.....	40
Código 12: Adaptador que realiza el parsing del archivo XML con los incidentes.	40
Código 13: Líneas que se agregan en la Activity principal para obtener los incidentes.....	41

1. INTRODUCCION

El aumento de la población y las crecientes migraciones del campo a las zonas urbanas, hace que la ciudad deba enfrentarse a nuevos desafíos [1], como el abastecimiento energético, planificaciones de transporte, mejoras en la seguridad, contaminación ambiental, entre otros.

Las ciudades inteligentes vienen a enfrentar estos desafíos aprovechando las oportunidades que ofrecen las nuevas tecnologías, pero no sin antes un cambio en los modelos de gobernanza, de gestión de servicios y recursos, mutando del modelo actual de gestión sectorial a otro de gestión integrada y territorial [2].

Chile es un país que sufre cada año diversos tipos de emergencias y debe estar preparado, sin importar si estos eventos son naturales o provocados por el hombre. Es por esta razón que es de gran importancia desarrollar soluciones orientadas a mejorar la movilidad de las personas, por medio de la entrega de información relevante, planificación de viajes, zonas de seguridad, entre otras, donde el ciudadano no sea un simple espectador, sino un protagonista y agente de cambio en su entorno y los procesos de mejora de su territorio.

Es por esto que en conjunto a la Subsecretaría de Transporte y su Unidad de Ciudades Inteligentes se trabajó levantando información para encontrar falencias en el sistema que se utiliza hoy en día para informar al ciudadano sobre eventos que afecten en su movilidad, y lograr dar con la solución idónea para el desafío propuesto, el cual se detalla a continuación.

1.1. MARCO GENERAL

1.1.1. Desafío o Problema a solucionar

El desafío planteado por la contraparte es el siguiente: *“Cómo podemos contar con la capacidad de obtener información en tiempo real y entregarla a la ciudadanía para aumentar la certidumbre de éstos frente a situaciones de emergencias que afecten la movilidad en un área, al mismo tiempo que se cuenta con la capacidad de reaccionar y realizar una gestión integrada del sistema de transportes y recursos asociados que permita sobrellevar el evento.”*

Para comprender la problemática y descubrir la oportunidad que en ella se presenta, es necesario analizar el desafío planteado por la contraparte a través de un desglose, para lograr con esto identificar y acotar el problema que viene implícito en el desafío.

1.2. MOTIVACION

Se busca como equipo crear una solución real y al alcance de todos los ciudadanos, al momento de enfrentar las problemáticas que presenta nuestro país cuando de emergencias se trata. Con esto unificar nuestro largo y angosto territorio, de tal forma de mejorar la información, comunicación y coordinación.

Quizás puede funcionar en complemento al resto de herramientas existentes hoy en día, aportando un granito de arena al mar de soluciones que hay y vienen en camino.

1.3. OBJETIVOS GENERALES

El piloto de memorias multidisciplinarias tiene como objetivo el desarrollo de las competencias transversales de los alumnos, el fomento de la innovación, el emprendimiento y la vinculación permanente con las industrias. Por esta razón se designa un equipo apto para enfrentar el desafío, compuesto por Francisco Olmedo, estudiante de ingeniería en diseño de productos, Gabriel Alfaro, estudiante de ingeniería electrónica con mención en telecomunicaciones y Diego Hinojosa, estudiante de ingeniería civil electrónica mención computadores.

La tarea específica de Francisco Olmedo es orientada al diseño de interfaz de la aplicación móvil. Esto incluye colores escogidos, logo de la aplicación, creación de íconos. Por otra parte, Gabriel Alfaro desarrolla la plataforma web que permite publicar los incidentes que luego consume la aplicación móvil. Con esto se logra crear una plataforma integral, en caso de que no se puedan obtener los incidentes desde la plataforma del gobierno.

Los objetivos que se plantean como equipo al momento de recibir el desafío son investigar la forma de obtener información en tiempo real y entregarla al ciudadano, de forma que permita aumentar la certidumbre en cuanto a la movilidad en un área. Sin embargo, la solución que se busca debe ser un medio efectivo de comunicación y atractivo para el ciudadano, de tal forma de tener la mayor cobertura posible.

1.4. OBJETIVOS ESPECÍFICOS

El objetivo principal es el diseño y desarrollo de una aplicación, que se alimente de eventos o incidentes publicados en las plataformas existentes del gobierno, que sea capaz de obtener la posición del usuario y con esto filtrar la información que afecte la movilidad en su entorno, de tal manera de asegurar que el ciudadano reciba sólo la información que le es útil.

Se busca que la aplicación sea atractiva, intuitiva y amigable para que el usuario tenga una buena experiencia y siga haciendo uso de esta, así asegurar que la solución se transforme en un canal efectivo de comunicación entre autoridades y ciudadanía.

1.5. ESTRATEGIA

- Primero se analiza el desafío con el equipo, a la vez que se tienen reuniones con la contraparte. En este caso los encargados de representar al Ministerio de Transporte y Unidad de Ciudades Inteligentes. Con esto lograr encontrar una dirección a seguir que sea del interés de ambas partes durante la investigación y desarrollo.
- Se buscan y analizan las soluciones existentes, ya sean dispuestas por el gobierno o por terceros, para ver sus puntos buenos y malos.
- Luego de tener una idea de solución, se analizan los requerimientos a nivel técnico, diseño y características que debe presentar.
- Una vez teniendo diseños de la solución, comienza el desarrollo e implementación del prototipo funcional.
- Finalizando, se realizan pruebas del prototipo, las que permitirán en el futuro seguir con el desarrollo iterativo incremental para mejorar la solución.

2. ESTADO DEL ARTE

Hoy en día existen numerosos dispositivos tecnológicos que permiten la recepción y entrega de datos en tiempo real, de igual forma, permiten una comunicación casi instantánea entre los usuarios, tales como teléfonos inteligentes, tabletas y computadores.

En la quinta región existen organismos como la UOCT (Unidad Operativa de Control de Transito) que hacen uso de estos dispositivos y que generan información útil para la ciudadanía. Algunas de las actividades de esta unidad son el control semafórico, manejo y supervisión de cámaras de vigilancia, cálculo de tiempo estimado de rutas importantes y flujo vehicular.

La forma en que la UOCT informa a los ciudadanos es a través de un *twitter*, pantallas informativas en algunas calles, y una página web, que se muestra a continuación.

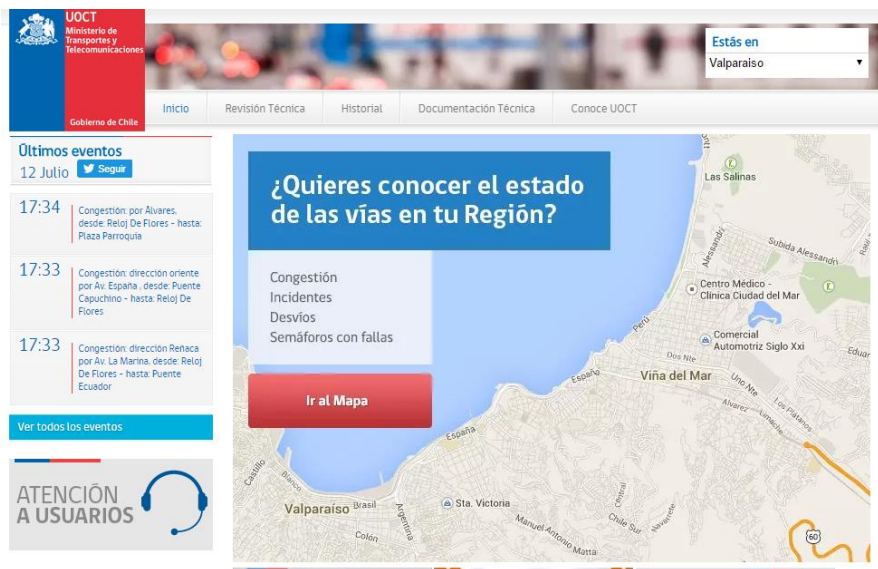


Figura 1: Captura de pantalla del sitio web de la UOCT región de Valparaíso (fuente, <http://www.transporteinforma.cl>).

Otro organismo en este contexto es TransporTeinforma, el cual se encuentra activo en Coquimbo, región Metropolitana, Valparaíso y la región de Los Lagos, entregando información por medio de

twitter, podcasts en la radio, videos en YouTube y un sitio web. Se concentra en información de transporte, ya sea accidentes o eventos programados que afecten la movilidad en las regiones.



Figura 2: Captura de pantalla del sitio web de transportTeinforma, selección de región (fuente, <http://www.transporteinforma.cl>).

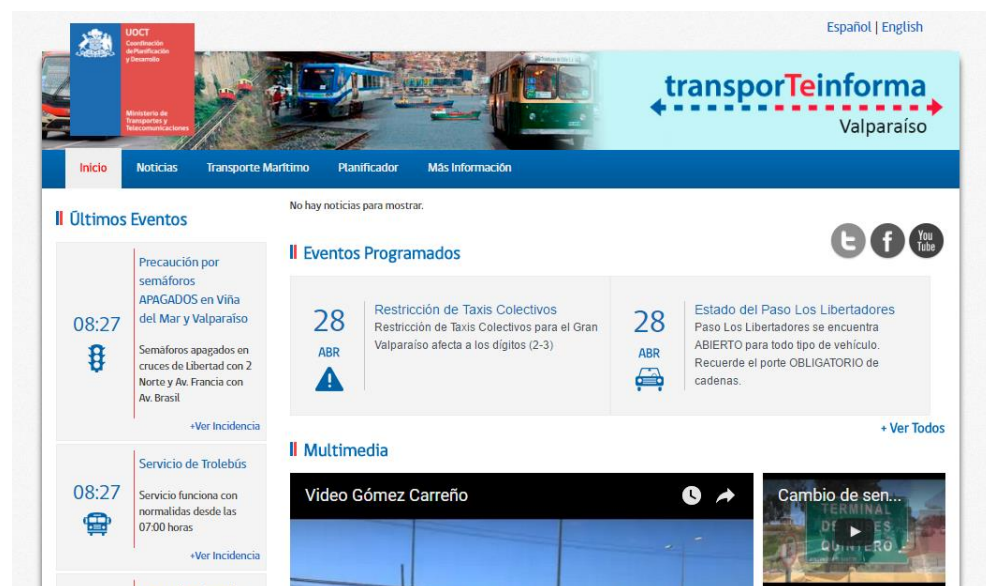


Figura 3: Captura pantalla del sitio web transportTeinforma Valparaíso (fuente, <http://www.transporteinforma.cl>).

También existen diversas aplicaciones para dispositivos móviles que tienen como finalidad informar al ciudadano sobre la movilidad en el transporte. Más del 60 por ciento de los usuarios de teléfonos móviles en Latinoamérica consideran **waze** como una de las aplicaciones que es importante en su día a día (comScore, 2015). Esta realiza cálculo de rutas eficientes en tiempo real, permite reportar accidentes, flujo vehicular lento, y hasta publicar el precio de los combustibles.



Figura 4: Logo de la aplicación waze (fuente, www.waze.com).



Figura 5: Capturas de pantalla de la aplicación waze (fuente, www.waze.com).

Otra aplicación móvil es **Moovit**, de origen español, que permite a los usuarios, al igual que **waze**, optimizar sus trayectos y planear las rutas, pero se enfoca mayormente en el transporte público, informando el tiempo estimado de los microbuses, paraderos, etc. También los usuarios actualizan la información y pueden reportar incidencias en tiempo real. Fue inaugurada en la Quinta región para la Copa América 2015, indicando el transporte necesario para que las personas llegaran a los estadios.



Figura 6: Logo de la aplicación Moovit (fuente, www.moovitapp.com).



Figura 7: Capturas de pantalla de la aplicación Moovit (fuente <https://okdiario.com>).

2.1. Análisis

Al analizar las alternativas presentadas por el gobierno, se puede notar que, si bien tienen diversas formas de entregar la información a los usuarios, no hacen uso de uno de los recursos más importantes de nuestra época que es el uso de dispositivos móviles del cual se puede obtener datos del usuario a la vez que se le entrega información.

Hoy en día nuestro país lidera el uso de smartphones en Latinoamérica con 7.9 millones de usuarios, y se espera que al 2018 más de la mitad de la población nacional va a poseer un teléfono inteligente [3].

Otro problema que presentan las plataformas del gobierno, es la saturación de información que se le entrega al ciudadano, obligándolo a tener que buscar entre toda ésta la que realmente le interesa, lo que no es muy práctico cuando suceden emergencias.

Respecto a las aplicaciones móviles existentes, **WAZE** y **MOOVIT** utilizan la información proveniente del usuario como pueden ser la posición, condiciones de tráfico actual, velocidad. Sin embargo, de forma activa los usuarios pueden informar a la comunidad de accidentes, peligros o eventos que afecten la movilidad.

No obstante, cada una apunta a un público diferente, *“Waze fue creada como una herramienta de navegación social para transporte privado”* [4], y *“Moovit es la aplicación líder mundial para el transporte público...”*[5]. Hoy en Chile según la encuesta origen destino de Valparaíso, nos revela que el porcentaje de viajes que se realiza en transporte público es de 39%, muy similar al 33% de viajes realizados en transporte privado. En tanto, los viajes no motorizados representan un 26,8% del total de viajes. En consecuencia, ambas no alcanzan a cubrir a toda la población cuando se trata de entregar información relevante a la ciudadanía.

Por estas razones que la solución propuesta, viene a ser un complemento a las soluciones existentes. Al igual que las grandes aplicaciones antes mencionadas, se quiere obtener información del usuario y su entorno, y desde las plataformas del gobierno obtener los incidentes o eventos que afecten la movilidad con el fin hacerlos llegar al usuario de forma precisa.

Incidente o evento en esta memoria se refiere a un acontecimiento que cambia un conjunto particular de circunstancias en cuanto a la movilidad. Por mencionar algunos: accidentes, desvíos, trabajos en la ruta.

3. DESARROLLO DE APLICACIÓN

En este capítulo se abordan los temas pertinentes al proceso de diseño y desarrollo de la aplicación. Es de gran importancia tener claros los aspectos que se presentan a continuación, pues de estos dependerá la implementación de la aplicación y su usabilidad.

3.1. ANÁLISIS DE REQUERIMIENTOS

Obtener los requisitos de un producto es la primera etapa antes de su creación. Estos nos dicen la funcionalidad y condiciones que debe cumplir el software al momento de estar implementado. Para ello es importante tener en cuenta la visión del usuario a la vez de las necesidades del cliente.

3.1.1. Requerimientos No Funcionales

Los requisitos no funcionales se refieren a propiedades emergentes del sistema como lo son la fiabilidad, rendimiento, disponibilidad, portabilidad.

En la tabla siguiente se muestran los requerimientos no funcionales del software a construir.

Atributo	Detalle
Tipo de Aplicación	Aplicación Móvil
Disponibilidad	Necesario GPS, Internet móvil e Incidentes desde un tercero.

Tabla 1, Requerimientos no funcionales

3.1.2. Requerimientos Funcionales

Los requisitos funcionales definen los servicios que debe proveer el sistema o el software, donde su ejecución depende del código, lo que permite separar en módulos según funcionalidad.

A continuación, se detallan los requerimientos funcionales en la tabla 2.

Ref.	Función	Módulo
RF1	Registro usuario	Usuario
RF2	Validación usuario	Usuario
RF3	Obtener posición del usuario	Posicionamiento
RF4	Obtener lista de incidentes	Obtención de incidentes
RF5	Desplegar lista con incidentes cercanos	Despliegue lista
RF6	Desplegar mapa con incidentes cercanos	Despliegue mapa
RF7	Desplegar información del clima en la ciudad	Obtención clima
RF8	Obtener distancia hacia cada incidente	Preparación incidentes
RF9	Asignar ícono a incidente en lista	Despliegue lista
RF10	Asignar ícono a incidente en mapa	Despliegue mapa
RF11	Compartir incidente en redes sociales	Despliegue lista
RF12	Filtrar incidentes por radio de búsqueda en lista	Despliegue lista
RF13	Filtrar incidentes por radio de búsqueda en mapa	Despliegue mapa
RF14	Modificar radio de búsqueda de incidentes	Despliegue mapa

Tabla 2, Tabla de requerimientos funcionales (fuente, elaboración propia).

3.1.3. Identificación de Actores del Sistema

En el sistema son partícipes dos actores; el usuario de la aplicación y por otro lado el sistema TransporTeinforma, los que serán explicados en detalle en la tabla siguiente.

Actor N°1		TransporTeinforma
	Descripción:	Grupo de profesionales, encargados de recolectar información de incidentes y subirlos a su plataforma.
	Características:	Este actor es el que provee la información que consumirá la aplicación móvil. De esta forma, los incidentes ya vendrán validados por un organismo confiable.
Actor N°2		Usuario de todo tipo de transporte que se mueve en la ciudad.
	Descripción:	Toda persona que se mueve en la ciudad, que desea informarse y a la vez informar a otros.
	Características:	Este actor es quien recibe la información validada, para así mejorar la movilidad en la ciudad. De la misma forma, es capaz de informar a sus pares, sin distorsionar la información.

Tabla 3, Actores del sistema (fuente, elaboración propia).

3.1.4. Modelo de casos de Uso

En la siguiente figura se representan los casos de uso identificados para el usuario que se mueve por la ciudad.



Figura 8:Diagrama de casos de uso (fuente, elaboración propia).

3.1.5. Especificaciones de casos de Uso

Se asume que al momento de usar la aplicación el usuario cuenta con las condiciones mínimas de funcionamiento, como lo son: Acceso a Internet y GPS.

Caso de Uso	Autenticar Usuario
ID	1
Descripción	El usuario ingresa sus datos de acceso a la aplicación para cargar información guardada.
Actores del Sistema	Usuario.
Actores Secundarios	No tiene.
Precondiciones	Ninguna.
Flujos principales	1. El usuario selecciona el tipo de ingreso 2. Ingresa los datos requeridos

	3. a. Si los datos están correctos, la aplicación sigue normalmente. 3. b. Si los datos son erróneos, muestra error y vuelve al punto 1.
Post condiciones	Ingreso a la aplicación.

Tabla 4, caso de uso "Autenticar usuario" (fuente, elaboración propia).

Caso de Uso	Ver incidentes cercanos
ID	2
Descripción	El usuario ve en una lista los incidentes cercanos.
Actores del Sistema	Usuario.
Actores Secundarios	TransporTeinforma
Precondiciones	- Que existan incidentes subidos por TransporTeinforma. - Que existan incidentes dentro del radio de búsqueda.
Flujos principales	1. El sistema muestra una lista con incidentes y la distancia hacia ellos. 2. En caso de no existir incidentes en la página web, muestra un error indicando la causa. 3. En caso de no existir incidentes dentro del radio de búsqueda, indica al usuario si prefiere aumentar el radio. 4. De existir incidentes despliega una lista con cada incidente dentro del radio de búsqueda y su distancia respectiva hacia el usuario.
Post condiciones	Información detallada de cada incidente.

Tabla 5, caso de uso "Ver incidentes cercanos" (fuente, elaboración propia).

Caso de Uso	Cambiar radio de búsqueda
ID	3
Descripción	El usuario se dirige al mapa donde puede cambiar el radio de búsqueda de incidentes.
Actores del Sistema	Usuario.
Actores Secundarios	TransporTeinforma
Precondiciones	Ninguna.
Flujos principales	1. El usuario presiona botón mapa 2. Mueve el indicador del <i>scrollbar</i>

	3. El sistema ajusta el radio, mostrando en el mapa un círculo alrededor del punto que indica la posición del usuario y el radio de búsqueda. A la vez muestra los incidentes dentro del radio.
Post condiciones	Filtrar por radio

Tabla 6, caso de uso "Cambiar radio de búsqueda" (fuente, elaboración propia).

Caso de Uso	Filtrar por radio de búsqueda
ID	4
Descripción	El usuario es capaz de filtrar la lista de incidentes por el radio escogido.
Actores del Sistema	Usuario.
Actores Secundarios	TransporTeinforma
Precondiciones	- Que existan incidentes subidos por TransporTeinforma.
Flujos principales	1- El usuario ajusta el radio 2- El sistema filtra la lista de incidentes, mostrando solo los que están dentro del radio de búsqueda.
Post condiciones	Ninguna.

Tabla 7, caso de uso "Filtrar por radio de búsqueda" (fuente, elaboración propia).

Caso de Uso	Ver mapa con incidentes
ID	5
Descripción	El usuario ve el mapa con los incidentes.
Actores del Sistema	Usuario.
Actores Secundarios	TransporTeinforma
Precondiciones	- Que existan incidentes subidos por TransporTeinforma.
Flujos principales	1- El usuario presiona el botón MAPA. 2- Se abre un mapa con los incidentes y el radio definido. 3- Se puede desplazar por el mapa y ver los incidentes.
Post condiciones	Ninguna

Tabla 8, caso de uso "Ver mapa con incidentes" (fuente, elaboración propia).

Caso de Uso	Ver detalles de incidentes.
ID	6
Descripción	El usuario puede ver los detalles de cada incidente desplegado en la lista.
Actores del Sistema	Usuario.
Actores Secundarios	TransporTeinforma
Precondiciones	- Que existan incidentes subidos por TransporTeinforma. - Que existan incidentes dentro del radio de búsqueda.
Flujos principales	1- El usuario presiona un incidente en la lista. 2- La aplicación despliega una ventana con los detalles de la aplicación.
Post condiciones	Compartir incidente.

Tabla 9, caso de uso "Ver detalles de incidente" (fuente, elaboración propia).

Caso de Uso	Compartir incidentes.
ID	7
Descripción	El usuario es capaz de compartir incidentes en las redes sociales.
Actores del Sistema	Usuario.
Actores Secundarios	TransporTeinforma
Precondiciones	- Que existan incidentes subidos por TransporTeinforma. - Que existan incidentes dentro del radio de búsqueda.
Flujos principales	1- El usuario presiona un incidente de la lista. 2- La aplicación despliega una ventana con detalles del incidente y un botón compartir. 3- El usuario presiona compartir. 4- El sistema muestra las opciones de redes sociales disponibles para compartir el incidente. 5- El usuario escoge la red social y el contacto.

	6- El sistema envía el incidente con una descripción y un link para abrir en un mapa.
Post condiciones	Ninguna.

Tabla 10, caso de uso " Compartir incidentes"(fuente, elaboración propia).

Caso de Uso	Abrir incidentes compartidos
ID	8
Descripción	El usuario puede abrir en la aplicación incidentes que le fueron compartidos por otros.
Actores del Sistema	Usuario.
Actores Secundarios	TransporTeinforma. Otro usuario de la aplicación.
Precondiciones	- Que existan incidentes subidos por TransporTeinforma. - Que existan incidentes dentro del radio de búsqueda. - Tener la aplicación instalada u otra aplicación que soporte mapas.
Flujos principales	1- Al usuario le llega un incidente compartido en las redes sociales. 2- Le hace un clic al link. 3- Si tiene la aplicación instalada le ofrece abrirlo y ver su ubicación y descripción. 4- Si no tiene instalada la aplicación, se puede abrir en cualquier aplicación que soporte mapas.
Post condiciones	Ninguna

Tabla 11, caso de uso "Abrir incidentes compartidos"(fuente, elaboración propia).

Caso de Uso	Cerrar sesión.
ID	9
Descripción	El usuario cierra la sesión de la aplicación.
Actores del Sistema	Usuario.
Actores Secundarios	No tiene.

Precondiciones	- Tener la sesión abierta.
Flujos principales	1- El usuario presiona el botón cerrar sesión. 2- La aplicación guarda los datos importantes.
Post condiciones	Ninguna.

Tabla 12, caso de uso "Cerrar sesión"(fuente, elaboración propia).

3.2 DISEÑO DE PROTOTIPOS DE USABILIDAD

Para el desarrollo de la interfaz de cualquier aplicación hay que tener en cuenta que debe ser intuitiva la navegación entre pantallas, conseguir una experiencia de usuario excelente para que el público no se canse de ella y sea fácil de utilizar.

Sabiendo que es una aplicación alojada en un dispositivo móvil, por consiguiente, hay cualidades de estos que pueden jugar en contra la usabilidad, como lo es el tamaño de la pantalla reducido, no se debe sobrecargar de información ni de procesos.

Se ha optado por un diseño basado en el Material Design, que presenta cualidades como: diseño llamativo, colorido, enfocado en la iconografía [6].

Algunos aspectos a tener en cuenta en una aplicación móvil:

- Que cada icono sea reconocible por el usuario.
- Colores adecuados que permitan una buena visibilidad.
- Letras del tamaño correcto para asegurar buena visibilidad.
- Uso de menús que faciliten el manejo de los usuarios.
- Uso de barras de progreso para que el usuario sepa que está esperando.
- Informar al usuario de errores u operaciones exitosas.

En esta etapa se realiza el *sketching*, que es una herramienta de trabajo que permite transmitir una idea de forma rápida y sencilla, de forma que cualquier persona pueda comprender. Lo importante en este proceso es definir de forma simple la interfaz de la aplicación, para luego analizar su funcionamiento y posibles falencias.

Estos dibujos ayudan al equipo a entender la futura aplicación y son de gran importancia para las próximas etapas del proyecto.

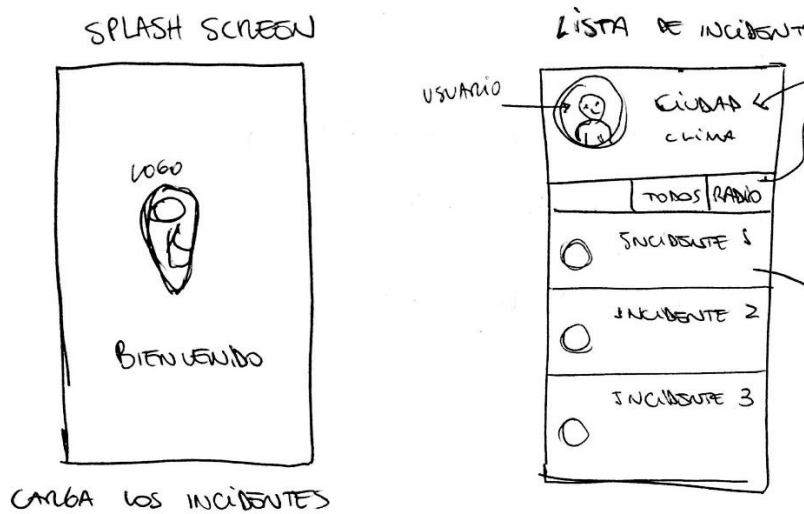


Figura 9: Primeros sketching realizados a mano (fuente: elaboración propia).

Según el sketch realizado anteriormente se puede investigar si existen aplicaciones que tengan un funcionamiento similar, ya sea botones, listas de datos, mapas, etc. Con esto controlar tiempos, diseños que sean más útiles, información técnica y de esta forma facilitar la tarea al momento de programar.

Como existen alrededor de 2.000.000 de aplicaciones para Android, se tiene mucho de donde sacar modelos, y más de una aplicación tendrá funcionalidades similares. Basta poner el buscador de imágenes “Android app template” y se mostrará una infinidad de modelos de aplicaciones que son de gran utilidad [7].

Un *wireframe* es el proceso de crear *mockup* o prototipo de la aplicación. Existen numerosos programas para realizar estos prototipos, lo que no impide su realización a mano. Las ventajas de los softwares es que permiten añadir la funcionalidad al realizar un clic en los elementos, como dirigirse a una vista al presionar un botón.

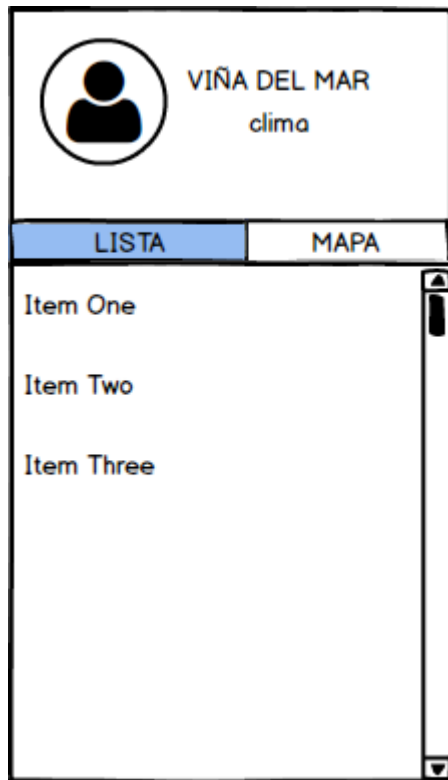


Figura 11: Idea de lista de incidentes (fuente: elaboración propia)..



Figura 10: Idea de mapa de incidentes (fuente: elaboración propia).

Un *storyboard* es un conjunto de imágenes que muestran el desarrollo temporal de la aplicación, como se conecta cada pantalla entre sí. Permite visualizar la forma en que el usuario navegará en la misma.

Esta etapa es una de las más importantes, ya que permite ver el funcionamiento antes de realizar la programación, evitar errores, y ponerse en el lugar del usuario.

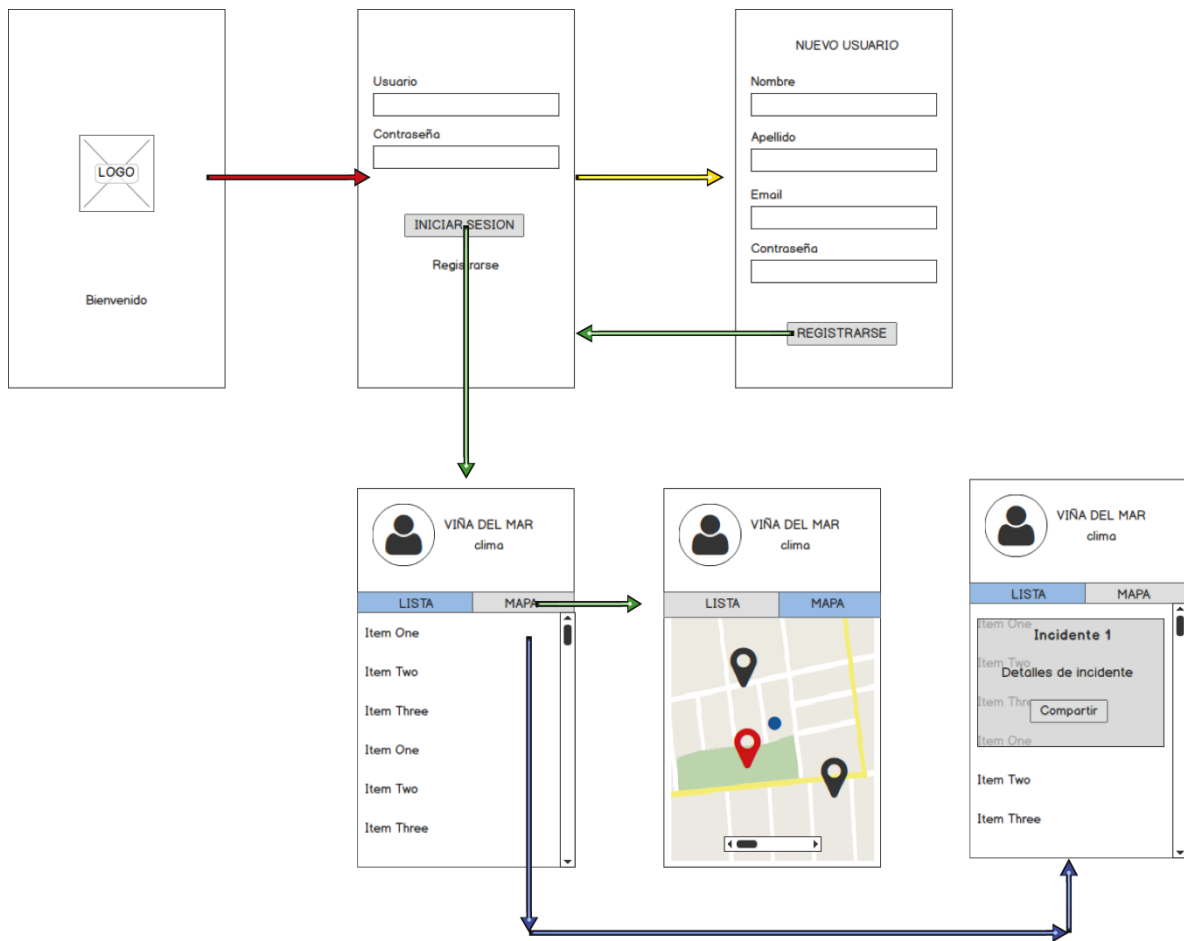


Figura 12: Storyboard de pantallas principales (fuente: elaboración propia).

3.3 CLASES IMPORTANTES

Esto ayuda a un mejor desarrollo antes de la programación, las clases más importantes son: Incidencia, Coordenada y Usuario.

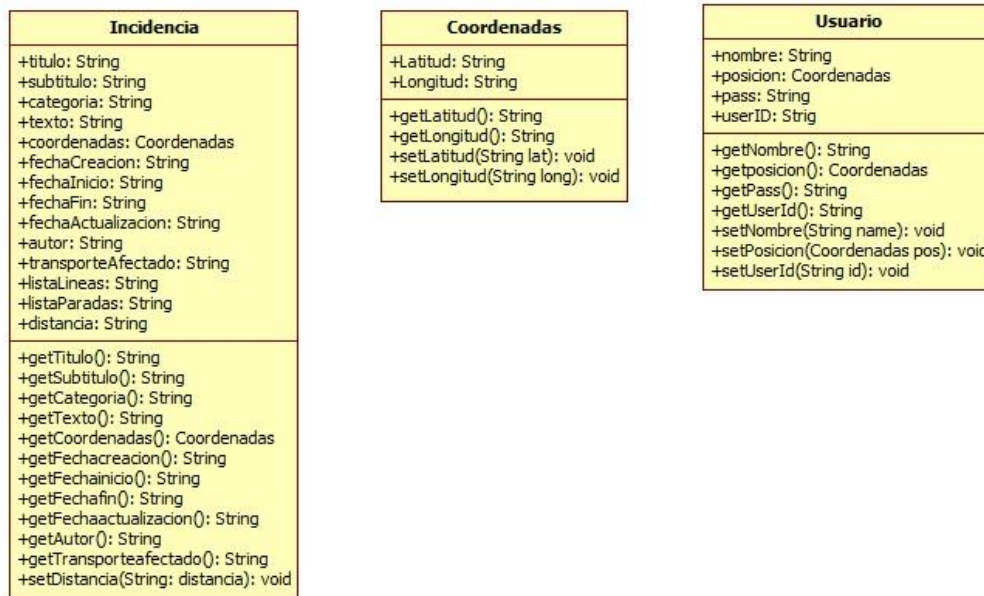


Figura 13: Clases de mayor importancia (fuente: elaboración propia).

Incidencia es la clase que define cada evento, indicando entre otras características lugar, titulo, fecha, tipo de incidente, distancia.

Coordenadas, es la clase creada para obtener la posición en latitud y longitud del usuario y de los eventos.

Usuario, es la clase que define a la persona y su relación con la aplicación, sus campos son nombre real, nombre de usuario, posición y contraseña.

3.4 MAPAS DE NAVEGACIÓN

Estos tipos de mapas permiten un manejo de las vistas de la aplicación, de manera que el usuario pueda entender mejor la interfaz, y qué acciones debe elegir para llegar a un fin específico.

En el siguiente diagrama se muestra el mapa de navegación de la aplicación.

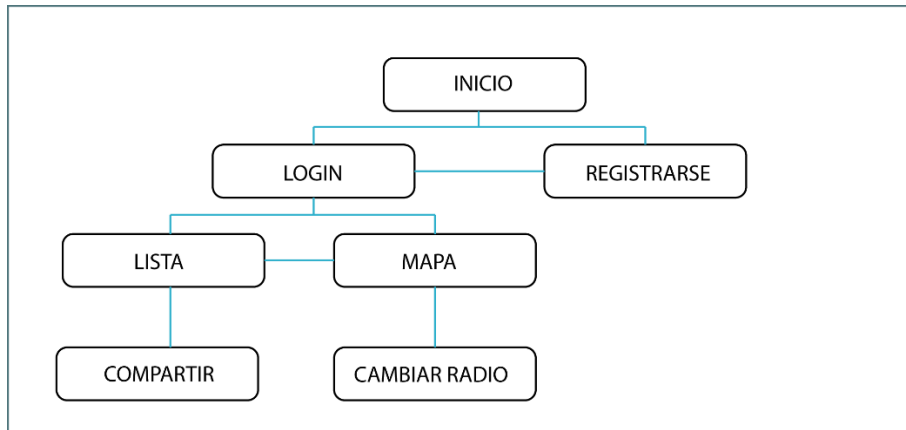


Figura 14: Mapa de navegación (fuente: elaboración propia).

3.4.1 Detalle del mapa de navegación

LOGIN: En esta parte el usuario introduce sus datos para la identificación, y obtener acceso a datos previamente almacenados para él.

LISTA: Permite visualizar en forma de lista los incidentes, con un ícono que permita un reconocimiento del incidente de forma fácil, y mostrando la distancia hacia este.

VER DETALLES INCIDENTES EN LISTA: Despliega una ventana con la información más detallada del incidente seleccionado.

COMPARTIR: Permite compartir un incidente seleccionado en las redes sociales.

MAPA: Se muestra un mapa, en donde se distingue el radio de búsqueda de incidentes, más un marcador personalizado para cada tipo de incidente. A la vez muestra el tamaño del radio de búsqueda en kilómetros.

AJUSTAR RADIO: En el mapa aparece un *seekbar*, que es una barra que permite el ajuste del radio de búsqueda, ya sea aumentar o disminuir.

VER DETALLES INCIDENTES EN MAPA: Al presionar un marcador de un incidente, se despliega la información detallada del mismo.

4. IMPLEMENTACIÓN

En este capítulo se verá más en profundidad algunos aspectos de la implementación de la aplicación en el IDE (Entorno de Desarrollo Integrado) Android Studio. Se utiliza lenguaje Java para la lógica y XML para la parte visual. [8] [9] [10]

4.1. DESARROLLO DE LA INTERFAZ

Como se mencionó anteriormente la parte de interfaz gráfica se trabaja con XML. Este IDE permite dos formas de crear la interfaz, ya sea con puro código o de forma visual.

A continuación, se muestra un ejemplo de código para implementar una interfaz simple.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_gravity="center_horizontal" />
</LinearLayout>
```

Código 1: Interfaz simple en XML

Existen distintos tipos de contenedores. En este caso se utiliza *LinearLayout* el que permite orientar los componentes dentro de forma horizontal o vertical. Se puede ver que la línea que ajusta esta orientación es *android:orientation="vertical"*, la que ordena todos los componentes internos de forma vertical, o sea uno debajo de otro.

En la imagen se aprecia una etiqueta de texto y un botón. Cada uno presenta características configurables como el ancho, largo, transparencia, tipo de texto, etc.

Se encuentran distintos tipos de medidas en Android Studio para fijar parámetros:

- px: pixeles.
- dp: densidad independiente de pixeles.
- sp: pixeles escalables.
- pt: puntos.
- dip: densidad de pixeles.
- mm: milímetros.
- In: pulgadas.

El código anterior genera la siguiente interfaz:

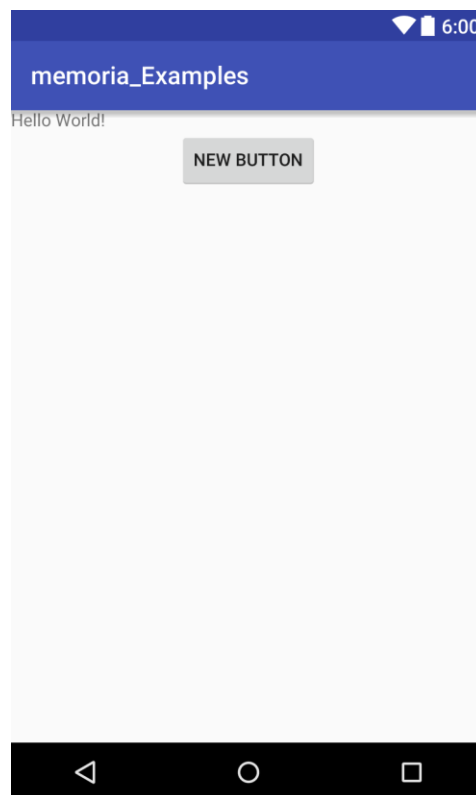


Figura 15: Interfaz simple (fuente: elaboración propia).

4.2. USO DE LISTA EN LA APLICACIÓN

Se utiliza un *ListView*, que es un grupo de ítems que permite desplazarse de forma vertical por la pantalla, donde cada elemento puede definirse como un *Layout* personalizado, ya sea con texto, imágenes u otras cosas.

Esta lista que se verá a continuación, es un ejemplo más básico del que se creó para mostrar la lista de incidencias en el proyecto.

Para iniciar la lista personalizada, primero se necesita:

- El diseño previo del *Layout* que contiene al *ListView*.
- Diseñar el *Layout* personalizado que se utilizará para cada ítem.
- Implementar una *Activity* que contenga el *ListView*
- Los datos que se utilizarán para rellenar cada *Layout* de la lista.

Para crear el *Layout* personalizado para cada ítem se genera un archivo XML. En nuestro caso tendremos una imagen en conjunto con un texto, como el boceto que se muestra a continuación.

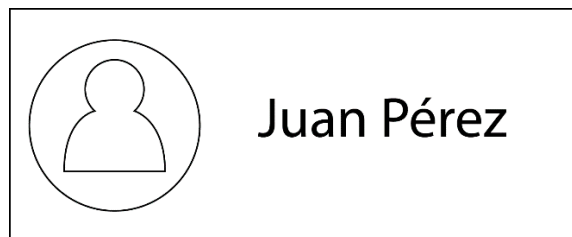


Figura 16: Bosquejo de layout para cada ítem de la lista (fuente: elaboración propia).

El código del archivo XML queda de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <ImageView
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:id="@+id/imageView"
        android:src="@drawable/ic_account_circle"
        android:layout_gravity="center"/>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Nombre Apellido"
        android:textSize="30sp"
        android:id="@+id/textView"
        android:paddingLeft="20dp"
        android:layout_gravity="center"/>
</LinearLayout>
```

Código 2: Layout XML para cada ítem de la lista.

Una vez listo el *Layout* que se utiliza en cada ítem, se procede a crear la Activity que contendrá un texto y nuestra ListView o lista personalizada. Al crear la Activity automáticamente genera un archivo XML para la interfaz y un archivo java para la lógica.

Se agregan algunas líneas de código lo que permita que la interfaz de la Activity quede de la siguiente forma:

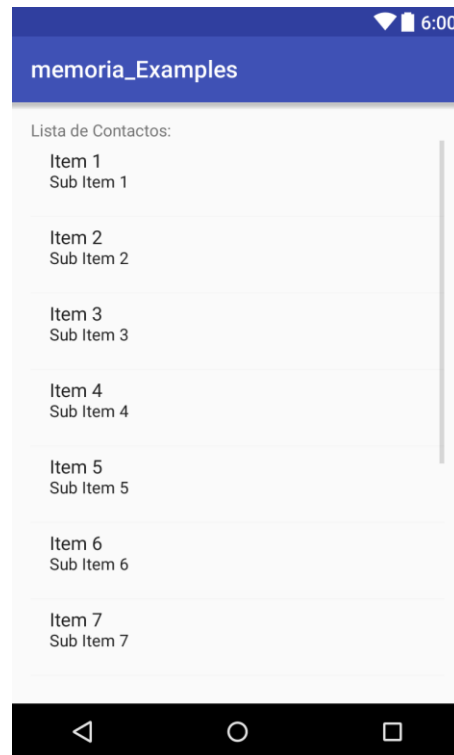


Figura 17: Interfaz de la Activity que contiene el ListView (fuente: elaboración propia).

Se observa que cada ítem de la lista tiene texto que aparece por defecto. Ahora se debe rellenar con los datos de las personas que se desea o bien en el caso del proyecto con incidentes, esto lo realiza un objeto llamado Adapter o Adaptador. Este objeto actúa como un puente entre los datos y cada elemento que contiene la lista.

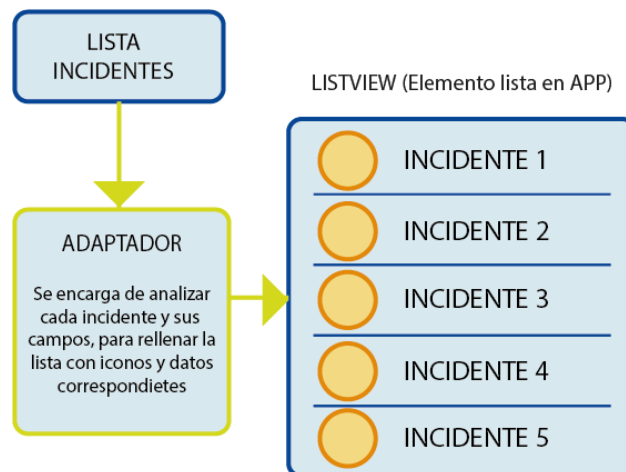


Figura 18: Diagrama de funcionamiento del Adaptador de incidentes (fuente, elaboración propia).

El código del adaptador en java es el siguiente:

```
public class ContactosAdapter extends ArrayAdapter<Contacto> {

    public ContactosAdapter(Context context, ArrayList<Contacto>
listaContactos){
        super(context, 0, listaContactos);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Get the data item for this position
        Contacto contacto = getItem(position);
        // Check if an existing view is being reused, otherwise inflate the
view
        if (convertView == null) {
            convertView =
LayoutInflater.from(getContext()).inflate(R.layout.item_layout, parent, false);
        }
        // Lookup view for data population
        TextView nameContact = (TextView)
convertView.findViewById(R.id.name_contact);
        ImageView imageContact= (ImageView)
convertView.findViewById(R.id.image_contact);
        // Populate the data into the template view using the data object
        nameContact.setText(contacto.name);
        if(contacto.getImage()==1){

        }

        // Return the completed view to render on screen
        return convertView;
    }
}
```

Código 3: en Java del Adaptador personalizado

Y este adaptador se asocia a la lista en la Actividad que contiene el ListView.

```
public class ListaItems extends AppCompatActivity {

    ListView listView;

    ArrayList<Contacto> listaContactos= new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_lista_items);

        listaContactos.add(new Contacto("Raul"));
        listaContactos.add(new Contacto("Felix"));
        listaContactos.add(new Contacto("Apolo"));
        listaContactos.add(new Contacto("Natalia"));
        listaContactos.add(new Contacto("Francisco"));
        listaContactos.add(new Contacto("Alejandra"));
        listaContactos.add(new Contacto("Hector"));
        listaContactos.add(new Contacto("Matias"));
        listaContactos.add(new Contacto("Alex"));
        listaContactos.add(new Contacto("Diego"));

        //Iniciando ListView con un Adapter personalizado
        ArrayList<Contacto> itemList = new ArrayList<>();
        ContactosAdapter contactosAdapter = new
ContactosAdapter(getApplicationContext(),listaContactos);

        listView = (ListView) findViewById(R.id.listView);
        listView.setAdapter(contactosAdapter);
    }
}
```

Código 4: Actividad donde se asocia el Adapter al ListView

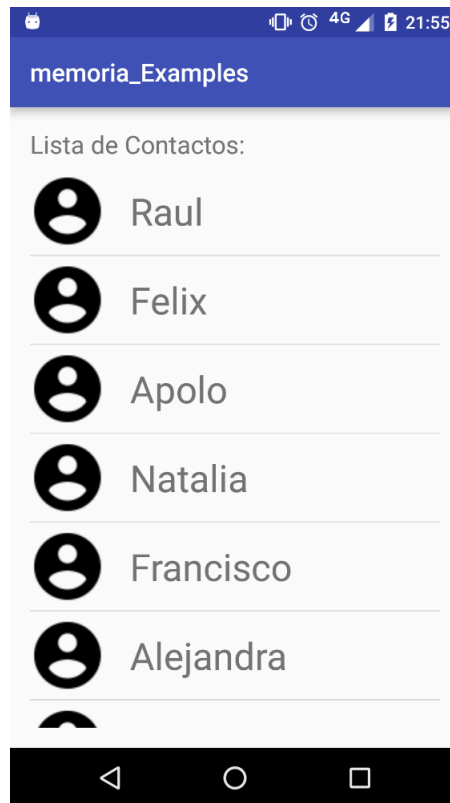


Figura 19: Lista implementada con contactos (fuente: elaboración propia).

4.2.1. Implementación de Filtro en la Lista

En este caso el ejemplo anterior se realizó con contactos, pero podría haber sido con cualquier otro objeto, incidentes, animales, países, números, etc. El Implementar un filtro que permita facilitar la búsqueda de la información que interesa se realiza directamente en el Adapter. En caso de tener una lista de incidentes, donde uno de sus atributos es la distancia, se puede implementar un filtro que se encargue de devolver los incidentes que estén a una distancia menor que la deseada. En este caso, almacenada en una variable llamada “radio”.

```
private class DistanceFilter extends Filter {
    @Override
    protected FilterResults performFiltering(CharSequence charSequence) {
        double radio = Double.parseDouble(String.valueOf(charSequence));
        FilterResults result = new FilterResults();

        if (charSequence.toString().length() > 0) {
            ArrayList<Incidente> filteredIncidents = new
ArrayList<Incidente>();
```

```

        for (int i = 0, l = listadoIncidentes.size(); i < l; i++) {
            Incidente incidente = listadoIncidentes.get(i);
            Log.d("FILTRO-Dist", i + " " + incidente.getDistance());
            if (radio >= Double.valueOf(incidente.getDistance())) {
                filteredIncidents.add(incidente);
                Log.d("FILTRO", i + "-" + incidente.getTitulo());
            }
            result.count = filteredIncidents.size();
            result.values = filteredIncidents;
        }
    } else {
        synchronized (this) {
            result.values = listadoIncidentes;
            result.count = listadoIncidentes.size();
        }
    }
    return result;
}
@Override
protected void publishResults(CharSequence charSequence, FilterResults
filterResults) {
    listadoIncidentesFiltrados = (ArrayList<Incidente>)
filterResults.values;
    notifyDataSetChanged();
    listadoIncidentes.clear();

    for (int i = 0, l = listadoIncidentesFiltrados.size(); i < l; i++)
        listadoIncidentes.add(listadoIncidentesFiltrados.get(i));

    notifyDataSetInvalidated();
}
}

```

Código 5: Implementación de objeto de tipo filtro.

En el código anterior, se crea una clase DistanceFilter que extiende de Filter, ya que es el filtro personalizado de distancia. Lo que realiza es tomar la propiedad distancia de cada incidente dentro del arreglo, y compararlo con la variable radio. Por consiguiente, si es menor o igual a radio lo agrega en un arreglo temporal. Cuando ya revisa todos los incidentes, muestra el nuevo arreglo con todos los que cumplen con la condición antes señalada.

4.3. USO DE MAPAS EN LA APLICACIÓN

La aplicación cuenta con un mapa proporcionado por Google Maps, que permite facilitar la tarea de ubicar incidentes y tener una mejor noción de las distancias hacia ellos, para planificar de mejor forma la movilidad en la ciudad.

Para agregar un mapa, se debe añadir una nueva Activity que nos ofrece el IDE llamada Google Maps Activity.

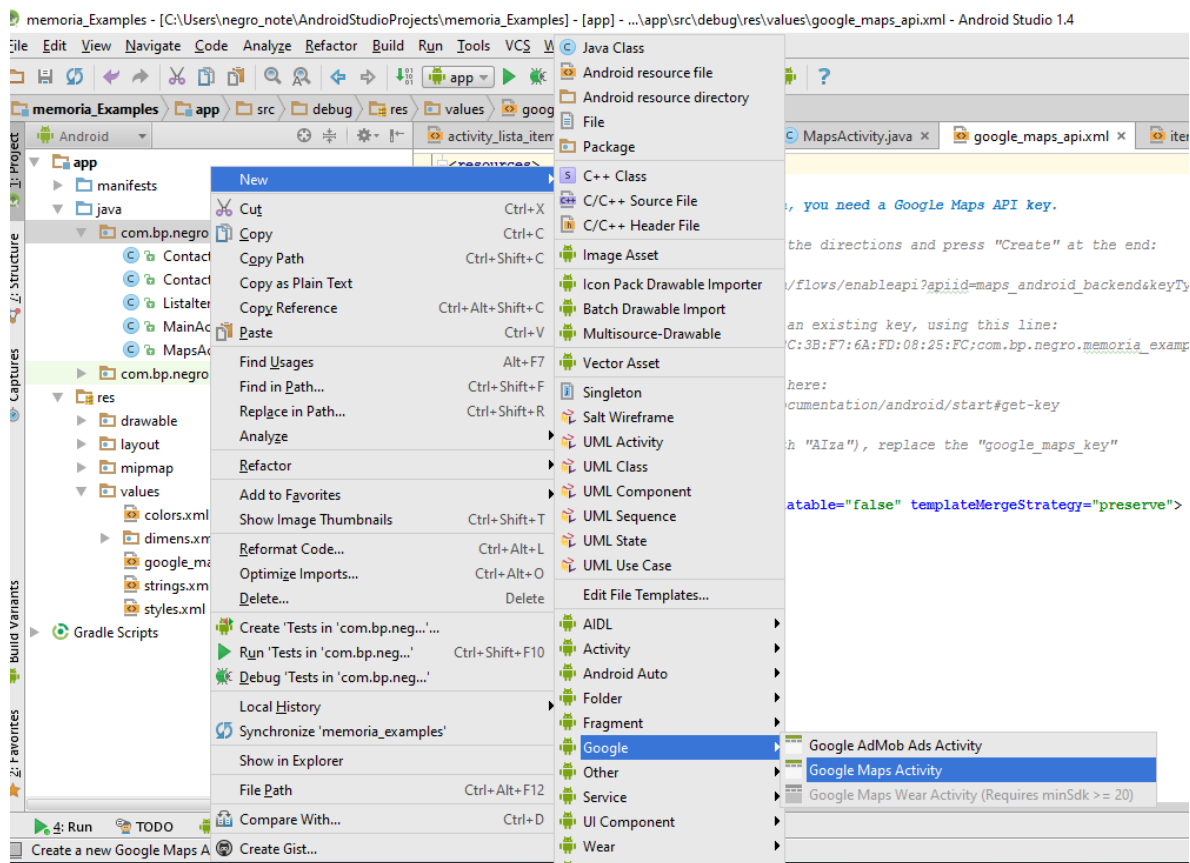


Figura 20: Agragando una nueva Activity de tipo GoogleMaps (fuente: elaboración propia)..

Al momento que se añade esta Activity crea tres archivos nuevos, el Java y dos de tipo XML, uno para la interfaz y otro que pide la “Google Maps API key”, que es la llave que proporciona Google para poder utilizar su API de mapas.



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!--
        TODO: Before you run your application, you need a Google Maps API key.

        To get one, follow this link, follow the directions and press "Create" at the end:

        https://developers.google.com/maps/documentation/android/start#get-key

        Alternatively, follow the directions here:
        https://developers.google.com/maps/documentation/android/start#get-key

        Once you have your key (it starts with "AIza"), replace the "google_maps_key"
        string in this file.
    -->
    <string name="google_maps_key" translatable="false" templateMergeStrategy="preserve">
        YOUR_KEY_HERE
    </string>
</resources>
```

Figura 21: Captura de pantalla de activación de Google Maps API (fuente: elaboración propia).

Además, añadirá una línea de código en el Android.Manifest.xml en donde se añaden los permisos, en este caso permiso de para usar la geo localización.

```
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
```

Código 6: Permisos para geo localización.

Para obtener la KEY, se debe dirigir a la dirección web de Google APIs que la proporcionará. Luego en la sección credenciales se crea una clave de API. Posteriormente se debe copiar y pegar en el código xml, y correr nuestra aplicación. De esta forma ya aparecerá el mapa.

4.3.1. Implementación de la Activity que muestra el Mapa

En el código java generado se manipula toda la lógica y acciones del mapa.

Por defecto viene este código:

```
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map
        is ready to be used.

        SupportMapFragment mapFragment = (SupportMapFragment)
        getSupportFragmentManager()
            .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in
        Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

Código 7: Ubicación por defecto del marcador.

Se puede observar en el código del método público *onMapReady* que primero crea un objeto del tipo **LatLng** llamado Sydney que se define con par de coordenadas latitud y longitud.

Luego añade un marcador en la posición agregándole como título “Marker in Sydney”, posteriormente mueve la cámara para que apunte al marcador.

Como cada incidente que se utiliza define su ubicación con latitud y longitud, solo basta tomar aquellos datos y añadir marcadores personalizados en cada incidente.

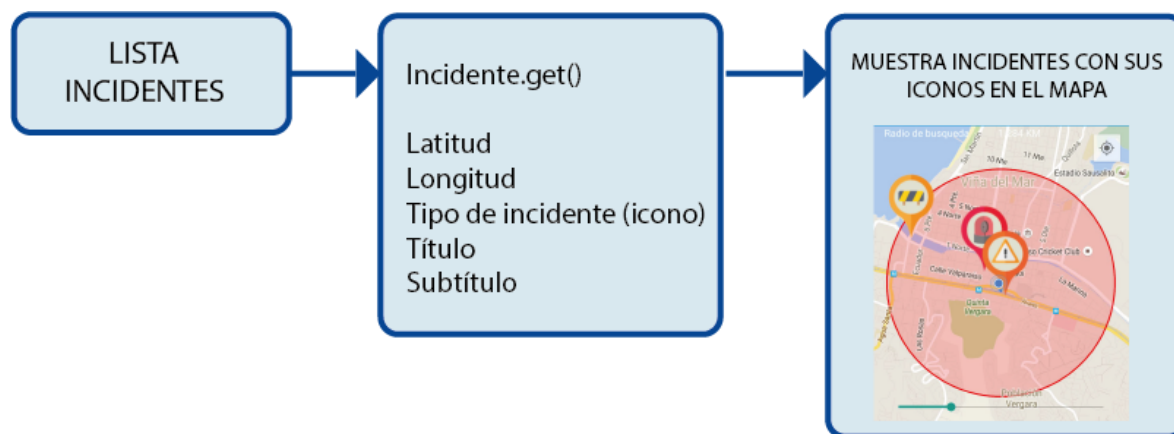


Figura 22: Funcionamiento de marcadores de incidentes sobre mapa (fuente: elaboración propia).

4.4. OBTENCIÓN DE DATOS DESDE PÁGINA WEB

En esta sección se explicará la forma en que se obtienen los distintos incidentes publicados desde la plataforma web de TransporTeinforma.

Se utiliza la biblioteca **Retrofit**, creada por Square Open Source [6], del tipo cliente REST (Representation State Transfer) para Android y Java. Con esta biblioteca es fácil hacer solicitudes del tipo HTTP a un servicio web, como lo son GET, PUT, POST.

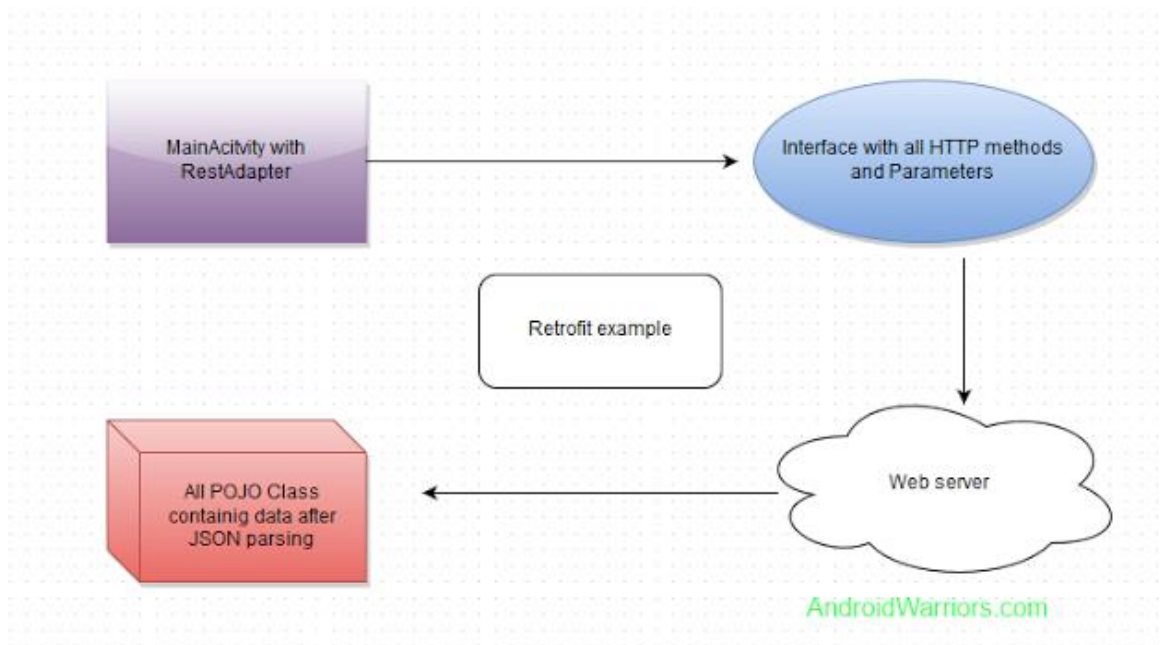


Figura 23: Esquema de funcionamiento de Retrofit. (Sacado de www.androidwarrior.com)

Primero que todo se debe dar permiso a la aplicación para utilizar internet, los permisos siempre se agregan en el Manifest.xml.

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
  
```

Código 8: Permiso para acceder a Internet

Luego se agregan las dependencias de la biblioteca para que el IDE Android Studio agregue todas las características de **Retrofit** al proyecto. Esto se realiza en el archivo del proyecto app/build.gradle.

```

dependencies {
    compile 'com.squareup.retrofit:retrofit:2.0.0-beta2'
    compile ('com.squareup.retrofit:converter-simplexml:2.0.0-beta2') {
        exclude module: 'stax'
        exclude module: 'stax-api'
        exclude module: 'xpp3'
    }
}
  
```

```

compile 'com.squareup.retrofit:adapter-rxjava:2.0.0-beta2'
compile 'io.reactivex:rxandroid:1.0.1'
}

```

Código 9: Dependencias de la biblioteca retrofit

Alguna de las dependencias que se muestran anteriormente se explicarán más adelante, cuando sea necesario.

Para seguir avanzando es necesario que se conozca el archivo de los eventos o incidentes que se solicita a la plataforma web.

```

<?xml version="1.0" encoding="UTF-8"?>
<xml>
  <ListaIncidentes>
    <Idioma>es</Idioma>
    <Incidencia>
      <Titulo>Angostamiento por trabajos en Ruta 5 Puerto Montt - Pargua</Titulo>
      <Subtitulo>Se recomienda a los automovilistas manejar con precaución y atentos a la señalética, debido a la existencia de tránsito alternado en los sectores indicados.</Subtitulo>
      <Categoria>Última Hora</Categoria>
      <Texto>texto html</Texto>
      <Coordenadas>
        <Latitud>-41.5714878661362</Latitud>
        <Longitud>-73.1899675447494</Longitud>
      </Coordenadas>
      <FechaCreacion/>
      <FechaInicio>2015-07-10 13:00:00</FechaInicio>
      <FechaFin>2015-07-10 23:59:00</FechaFin>
      <FechaActualizacion>2015-07-10 08:40:00</FechaActualizacion>
      <Autor>Transporteinforma</Autor>
      <TransporteAfectado>transporte_terrestre</TransporteAfectado>
      <ListaParadas/>
      <ListaLineas/>
    </Incidencia>
    <Incidencia>
      <Titulo>Angostamientos de calzada por trabajos en Ruta 5 Río Bueno - Puerto Montt</Titulo>
      <Subtitulo>Se recomienda a los automovilistas manejar con precaución y atentos a la señalética, respetando la velocidad máxima permitida por estos sectores de 70 km/h.</Subtitulo>
      <Categoria>Última Hora</Categoria>
      <Texto>texto html</Texto>
    </Incidencia>
  </ListaIncidentes>
</xml>

```

```

<Coordenadas>
  <Latitud>-41.1173832656353</Latitud>
  <Longitud>-73.0634882301092</Longitud>
</Coordenadas>
<FechaCreacion/>
<FechaInicio>2015-07-08 11:30:00</FechaInicio>
<FechaFin>2015-07-08 23:59:00</FechaFin>
<FechaActualizacion>2015-07-08 06:22:00</FechaActualizacion>
<Autor>Transporteinforma</Autor>
<TransporteAfectado>transporte_terrestre</TransporteAfectado>
<ListaParadas/>
<ListaLineas/>
</Incidencia>
</ListaIncidencias>
<ListaIncidencias>
  <Idioma>en</Idioma>
</ListaIncidencias>
</xml>

```

Código 10: Extracto de código XML devuelto por la plataforma web

Con este archivo descargado se utiliza para crear las clases respectivas que permiten convertir el texto XML a un objeto que luego se puede manejar dentro de la aplicación. A esta acción se le denomina “*parsing*”.

Existen páginas web que ayudan a convertir un archivo XML a un objeto en java, una de ellas es <http://pojo.sodhanalibrary.com/> , se introduce el código XML y esta indica las clases que se deben crear en el proyecto.

Para el proyecto basta con 5 clases. (Anexo 8.2)

Cuando se tienen las clases creadas, lo siguiente es crear una instancia de **Retrofit** y un Adapter que permita el parsing del archivo XML. Esto es relativamente fácil, ya que **Retrofit** cuenta con convertidores. En este caso se utiliza SimpleXmlConverter.

```

public interface IncitApi {

    @GET("/servicios/desarrolladores/logica/download_xml.php?cn=loslagos&tipoUrl=2&tipo=0")
    Call<ResultAll> getIncidents();

}

```

Código 11: API encargada de realizar la petición de los incidentes.

```

public class IncitRestAdapter implements Serializable {
    String TAG = getClass().getSimpleName();

    //reference to the library
    public Retrofit retrofit;
    //Reference to de IncitApi
    IncitApi api;

    public IncitRestAdapter(String base_url) {
        Log.d(TAG, "IncitRestAdapter");

        retrofit = new Retrofit.Builder()
            .baseUrl(base_url)
            .addConverterFactory(SimpleXmlConverterFactory.create())
            .build();

        //API lista para realizar peticiones.
        api = retrofit.create(IncitApi.class);
    }

    public IncitApi getApi() {return api;}
}

```

Código 12: Adaptador que realiza el parsing del archivo XML con los incidentes.

Creadas estas clases solo resta llamarlas en el método *onCreate* de la Activity principal.

```

IncitRestAdapter incit_api = new IncitRestAdapter(dir);

    final retrofit.Call<ResultAll> request =
incit_api.getApi().getIncidents();
    Log.i("PLEASE", request.toString());

    request.enqueue(new Callback<ResultAll>() {

        @Override
        public void onResponse(Response<ResultAll> response, Retrofit retrofit)
{
            Log.d("PLEASE", "onResponse");
            Log.d("PLEASE", response.toString());
            DataAvailable(response);
            request.cancel();
        }

        @Override
        public void onFailure(Throwable t) {
            Log.d("Error", "onFailure");
            Log.d("Error", t.toString());
        }

    });
}

//Accion para ver cuando estan listos los datos
void DataAvailable(retrofit.Response<ResultAll> result) {
    ResultAll result1 = result.body();

    List<Result> listaResultados = result1.getResulta2();
    List<Incident> incidentList = new ArrayList<>();
    for (Result listalistaIncidentes : listaResultados) {
        incidentList = listalistaIncidentes.getIncidencias();
        if(incidentList!=null) {
            for (Incident incident : incidentList) {
                Log.i("PLEASE", String.valueOf(incident.getTitulo()));
            }
            ListView lv = (ListView) findViewById(R.id.listView);
            MyAdapter myAdapter = new MyAdapter(getApplicationContext(),
(ArrayList) incidentList);
            lv.setAdapter(myAdapter);
        }
    }
}
}

```

Código 13: Líneas que se agregan en la Activity principal para obtener los incidentes.

El código anterior obtiene los incidentes y los guarda en un arreglo, que luego son utilizados para llenar la lista antes creada (ListView), quedando de la forma que se muestran a continuación.



Figura 24: Lista de incidentes obtenidos de plataforma web (fuente: elaboración propia).

Se observa que aparecen símbolos extraños en los eventos. Esto ocurre cuando el texto tiene tildes o letras ñ, ya que el conversor utilizado no está implementado para español.

4.5. ALGORITMO DE FILTRADO DE INCIDENTES

El objetivo principal de la aplicación es evitar la saturación de la información de incidentes o eventos que afecten la movilidad en un área al usuario. Por esto que se ha puesto énfasis en el filtrado de incidencias por radio de búsqueda. Esto quiere decir que se despliega una circunferencia alrededor de la posición del usuario, en la que se muestran solo los eventos que estén en su interior.

- Primero se obtiene la ubicación del dispositivo móvil cada cierto tiempo, o cuando se realice un desplazamiento de X distancia en metros.
- Se almacena la posición del punto anterior en una variable.
- Se obtienen los incidentes desde la plataforma web.
- Se calcula la distancia hacia cada incidencia, como la diferencia entre el punto del evento y la posición del usuario.
- Se almacena la distancia en cada incidente.
- Se define un radio por defecto si no se ha variado nunca, de lo contrario se utiliza el radio definido por el usuario.
- Se compara la distancia de cada evento, con el radio mencionado en el punto anterior. Si es menor o igual se deja visible, de lo contrario desaparece, ya sea de la lista o el mapa.



Figura 25: Diagrama de filtro de incidentes (fuente: elaboración propia).

5. PRUEBAS Y RESULTADOS

5.1. ANALISIS DE RESULTADOS OBTENIDOS

A nivel de interfaz, luego de usar la aplicación un tiempo prolongado notamos que se podía mejorar algunas cosas para adecuarse a las aplicaciones que se estaban usando hoy en día. De la misma forma se mejoraron problemas de usabilidad como tiempos de carga en mapas y esperas en la pantalla de inicio.

El logo de la aplicación, icono de lanzamiento y nombre fueron cambiados, por un tema de experiencia de usuario. Se escogieron colores más agradables como naranja y celeste, que transmitieran tranquilidad y confianza.

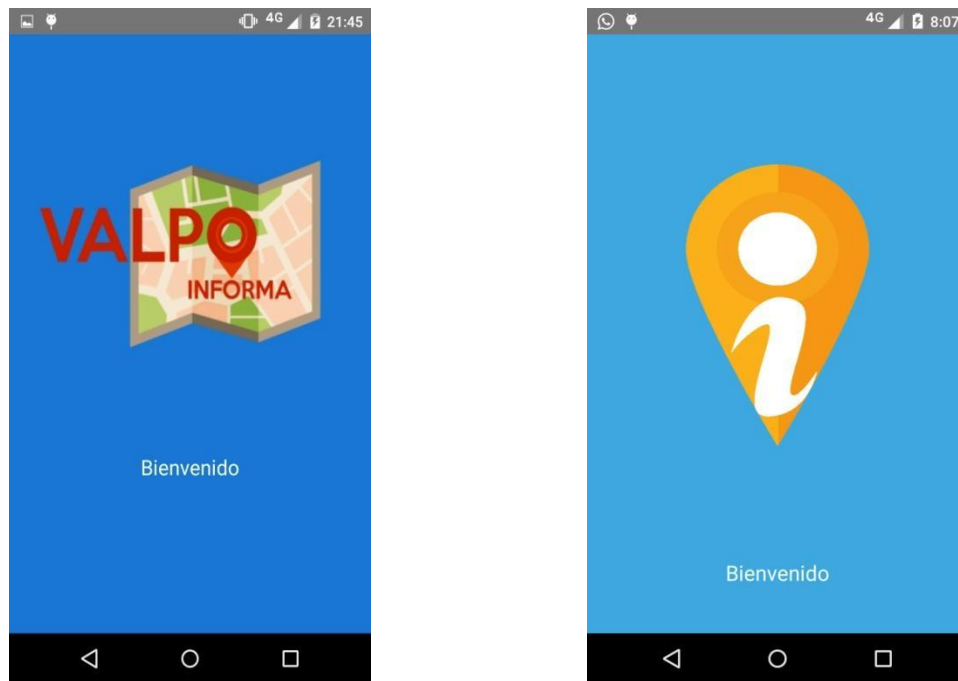


Figura 26: Pantalla de inicio antes (Valpo Informa) y después (INCIT) (fuente: elaboración propia).

Respecto a la lista se decidió añadir iconos que fueran fáciles de asociar con el tipo de incidente. De esta forma se mejora la experiencia de usuario, a la vez que se crea una interfaz más agradable.

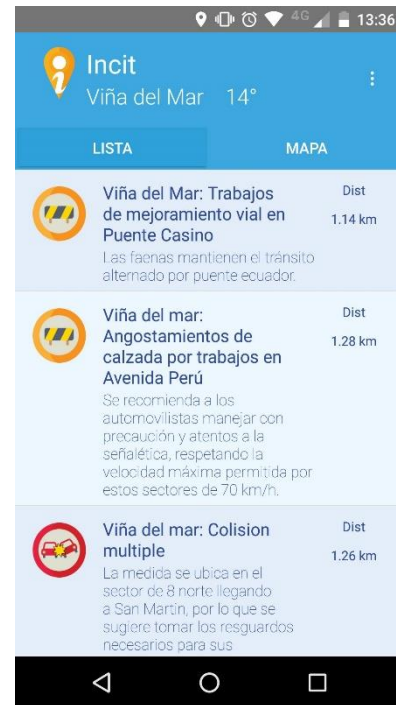


Figura 27: Pantallas de la aplicación antes y después (fuente: elaboración propia).

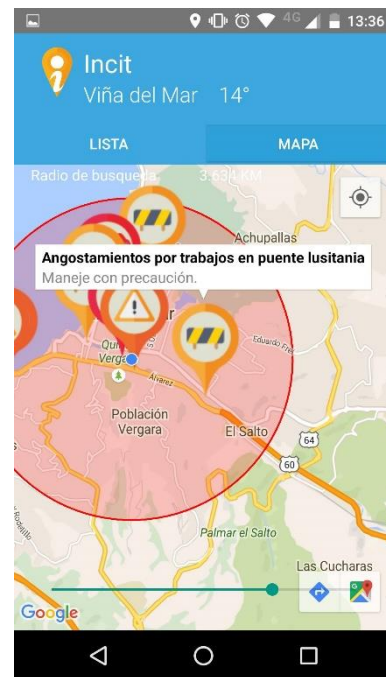
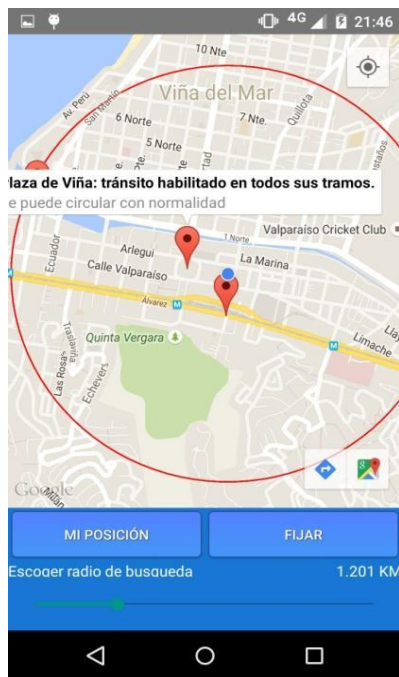


Figura 28: Pantallas de la aplicación antes y después (fuente: elaboración propia).

Para evaluar el correcto funcionamiento de la descarga de incidentes, se implementó una página web, con incidentes de prueba, de tal forma de simular un servicio web sin depender de otras páginas.

INCIDENTES TESTED INCIT®

Página de autoria de Diego O. Hinojosa Córdova

es utilizada para el testeo de aplicaciones móviles

[Stgo](#)

[Valpo](#)

[Valpo2](#)

Figura 29: Página donde están alojados incidentes de prueba (fuente: elaboración propia).

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1. CONCLUSIONES

Se presentan las conclusiones específicas del proyecto:

- El tiempo utilizado para el desarrollo (programación), 4 meses, fue suficiente para crear un prototipo funcional de aplicación. Por esto no se añadieron otras funcionalidades que se tenían pensadas de un inicio. Solo se dejaron los módulos que realmente aportaban funcionalidad para solucionar el desafío.
- El haber programado en Android, que es un carácter libre, ayudó en gran medida a encontrar mucho material en internet, como documentación presentada por el equipo de Android, que fue de gran utilidad, ya que previo a esta memoria solo se tenía conocimiento de programación orientada a objetos y Java.
- Un punto en contra de Android es la gran variedad de dispositivos móviles, lo que se pone a prueba al momento de crear la interfaz. La gran cantidad de tamaños de pantallas, características técnicas, y distintas versiones del sistema operativo, hacen que no se puedan aprovechar todos los recursos nuevos que nos entrega Android, ya que se estarían dejando de lado los dispositivos más antiguos.
- Es muy importante un buen equipo de trabajo, su comunicación y capacidad de resolver problemas, ya que la mayor parte del proyecto se trabajará en equipo.
- Se quitó la pantalla de Login y la opción de cerrar sesión, porque, al no tener un servicio web con base de datos, no se podían administrar los usuarios de forma correcta. La aplicación sigue funcionando de igual manera y almacena el radio de búsqueda de forma local en el dispositivo móvil.
- Como se tuvo que aprender a programar en Android desde cero, el desarrollo iterativo e incremental resultó ser muy útil, ya que al programar un módulo y dejarlo funcionando se añadía el siguiente, resultando una aplicación cada vez con mayores funcionalidades.

6.2. TRABAJOS FUTUROS

Uno de los trabajos que quedaron pendientes es el añadir las funcionalidades que se propusieron al inicio como:

- Login de usuario, ya sea asistido con Facebook, Google+, o algún otro administrador de usuarios.
- Guardar datos personales de cada usuario, para crear analítica, que permita seguir perfeccionando la aplicación.
- Crear interfaz que pueda acomodarse a las distintas pantallas, sin importar qué dispositivo móvil se está usando y de esta forma asegurar a cada usuario una mejor experiencia.
- Añadir el transporte afectado por el incidente de forma más específica como líneas de microbuses, y a la vez ofrecer líneas alternativas.
- Implementar la opción de añadir la ruta frecuente del usuario para advertir de incidentes de manera de ofrecer un mejor servicio al usuario.
- Un buen servicio web para que la aplicación genere una base de datos, así obtener estadísticas, ya sea de rutas frecuentes, congestiones, tiempos de viaje. Esto permite mejores planificaciones a nivel de transporte.
- Implementar un servicio web que permitiese enviar notificaciones Push a los usuarios.
- Otra tarea, no menos importante, es liberar una versión de prueba a un grupo pequeño de usuarios en Google Play, para obtener feedback, así mejorar y validar la aplicación.

7. REFERENCIAS

- [1] El aumento de la población y las crecientes migraciones del campo a las zonas urbanas. - <http://data.worldbank.org/indicator/SP.URB.TOTL.IN.ZS?locations=CL>
- [2] Estrategia de ciudad inteligente para el transporte Chile 2020, Subsecretaria de Transporte, Chile.
- [3] Uso de smartphones en Latinoamérica. (www.emarketer.com)
- [4] Foco de usuarios aplicación waze,
<https://support.google.com/waze/answer/6078702?hl=es>
- [5] Descripción aplicación Moovit,
<https://play.google.com/store/apps/details?id=com.tranzmate&hl=es>
- [6] Uso de colores para material design,
www.materialui.co
- [7] Plantillas de aplicaciones usando material design.
<http://materialdesignblog.com>
- [8] Tutoriales de programación en Android.
www.hermosaprogramacion.com
- [9] Tutoriales de programación en Android.
www.hermosaprogramacion.comcodeproject.com
- [10] Tutoriales de programación en Android.
www.sgoliver.net

8. ANEXO

8.1 Archivo XML proveniente de Transporteinforma (Ejemplo breve)

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
  <ListaIncidencias>
    <Idioma>es</Idioma>
    <Incidencia>
      <Titulo>Valparaíso: Angostamiento por trabajos en cauce Av
Argentina</Titulo>
      <Subtitulo>Se recomienda a los automovilistas manejar con precaución y
atentos a la señalética.</Subtitulo>
      <Categoría>Arreglos</Categoría>
      <Texto>Texto en html para página web</Texto>
      <Coordenadas>
        <Latitud>-33.045197</Latitud>
        <Longitud>-71.605076</Longitud>
      </Coordenadas>
      <FechaCreacion />
      <FechaInicio>2015-07-10 13:00:00</FechaInicio>
      <FechaFin>2015-08-30 23:59:00</FechaFin>
      <FechaActualizacion>2015-07-10 08:40:00</FechaActualizacion>
      <Autor>Transporteinforma</Autor>
      <TransporteAfectado>transporte_terrestre</TransporteAfectado>
      <ListaParadas />
      <ListaLineas />
    </Incidencia>
    <Incidencia>
      <Titulo>Viña del Mar: Trabajos de mejoramiento vial en Puente
Casino</Titulo>
      <Subtitulo>Las faenas mantienen el tránsito alternado por puente
ecuador.</Subtitulo>
      <Categoría>Arreglos</Categoría>
      <Texto>Texto para pagina web.</Texto>
      <Coordenadas>
        <Latitud>-33.021159</Latitud>
        <Longitud>-71.560734</Longitud>
      </Coordenadas>
      <FechaCreacion />
```

```

<FechaInicio>2015-08-15 08:00:00</FechaInicio>
<FechaFin>2015-08-31 23:59:00</FechaFin>
<FechaActualizacion>2015-08-10 06:37:00</FechaActualizacion>
<Autor>Transporteinforma</Autor>
<TransporteAfectado>transporte_terrestre</TransporteAfectado>
<ListaParadas />
<ListaLineas />
</Incidencia>
<Incidencia>
  <Titulo>Viña del mar: Angostamientos de calzada por trabajos en Avenida
Perú</Titulo>
  <Subtitulo>Se recomienda a los automovilistas manejar con precaución y
atentos a la señalética, respetando la velocidad máxima permitida por estos sectores de
70 km/h.</Subtitulo>
  <Categoría>Arreglos</Categoría>
  <Texto>Texto en html para página web</Texto>
  <Coordenadas>
    <Latitud>-33.016459</Latitud>
    <Longitud>-71.558910</Longitud>
  </Coordenadas>
  <FechaCreacion />
  <FechaInicio>2015-08-10 11:30:00</FechaInicio>
  <FechaFin>2015-08-12 23:59:00</FechaFin>
  <FechaActualizacion>2015-08-11 06:22:00</FechaActualizacion>
  <Autor>Transporteinforma</Autor>
  <TransporteAfectado>transporte_terrestre</TransporteAfectado>
  <ListaParadas />
  <ListaLineas />
</Incidencia>
<Incidencia>
  <Titulo>Viña del mar: Colision multiple</Titulo>
  <Subtitulo>La medida se ubica en el sector de 8 norte llegando a San
Martin, por lo que se sugiere tomar los resguardos necesarios para sus
desplazamientos.</Subtitulo>
  <Categoría>Colision</Categoría>
  <Texto>Texto para página web</Texto>
  <Coordenadas>
    <Latitud>-33.014574</Latitud>
    <Longitud>-71.555187</Longitud>
  </Coordenadas>
  <FechaCreacion />

```

```

        <FechaInicio>2015-08-14 08:00:00</FechaInicio>
        <FechaFin>2015-08-31 23:59:00</FechaFin>
        <FechaActualizacion>2015-08-17 06:35:00</FechaActualizacion>
        <Autor>Transporteinforma</Autor>
        <TransporteAfectado>transporte_terrestre</TransporteAfectado>
        <ListaParadas />
        <ListaLineas />
    </Incidencia>
</ListaIncidencias>
<ListaIncidencias>
    <Idioma>en</Idioma>
</ListaIncidencias>
</xml>

```

8.2 Códigos de clases devueltas por <http://pojo.sodhanalibrary.com>

8.2.1 Clase Result.java

```

public class Result
{
    private Xml xml;

    public Xml getXml ()
    {
        return xml;
    }

    public void setXml (Xml xml)
    {
        this.xml = xml;
    }

    @Override
    public String toString()
    {
        return "ClassPojo [xml = "+xml+"]";
    }
}

```

8.2.2 Clase Xml.java

```
public class Xml
{
    private ListaIncidencias[] ListaIncidencias;

    public ListaIncidencias[] getListaIncidencias ()
    {
        return ListaIncidencias;
    }

    public void setListaIncidencias (ListaIncidencias[] ListaIncidencias)
    {
        this.ListaIncidencias = ListaIncidencias;
    }

    @Override
    public String toString()
    {
        return "ClassPojo [ListaIncidencias = "+ListaIncidencias+"]";
    }
}
```

8.2.3 Clase ListaIncidencias.java

```
public class ListaIncidencias
{
    private String Idioma;

    private Incidencia[] Incidencia;

    public String getIdioma ()
    {
        return Idioma;
    }

    public void setIdioma (String Idioma)
    {
        this.Idioma = Idioma;
    }

    public Incidencia[] getIncidencia ()
    {
        return Incidencia;
    }

    public void setIncidencia (Incidencia[] Incidencia)
    {
        this.Incidencia = Incidencia;
    }

    @Override
    public String toString()
    {
        return "ClassPojo [Idioma = "+Idioma+", Incidencia = "+Incidencia+"]";
    }
}
```

8.2.4 Clase Incidencia.java

```
public class Incidencia
{
    private String Subtitulo;

    private String FechaInicio;

    private String Categoría;

    private String FechaCreacion;

    private String TransporteAfectado;

    private Coordenadas Coordenadas;

    private String Titulo;

    private String Texto;

    private String ListaParadas;

    private String Autor;

    private String ListaLineas;

    private String FechaFin;

    private String FechaActualizacion;

    public String getSubtitulo ()
    {
        return Subtitulo;
    }

    public void setSubtitulo (String Subtitulo)
    {
        this.Subtitulo = Subtitulo;
    }

    public String getFechaInicio ()
    {
        return FechaInicio;
    }

    public void setFechaInicio (String FechaInicio)
    {
        this.FechaInicio = FechaInicio;
    }

    public String getCategoría ()
    {
        return Categoría;
    }

    public void setCategoría (String Categoría)
    {
        this.Categoría = Categoría;
    }

    public String getFechaCreacion ()
```

```

    {
        return FechaCreacion;
    }

    public void setFechaCreacion (String FechaCreacion)
    {
        this.FechaCreacion = FechaCreacion;
    }

    public String getTransporteAfectado ()
    {
        return TransporteAfectado;
    }

    public void setTransporteAfectado (String TransporteAfectado)
    {
        this.TransporteAfectado = TransporteAfectado;
    }

    public Coordenadas getCoordenadas ()
    {
        return Coordenadas;
    }

    public void setCoordenadas (Coordenadas Coordenadas)
    {
        this.Coordenadas = Coordenadas;
    }

    public String getTitulo ()
    {
        return Titulo;
    }

    public void setTitulo (String Titulo)
    {
        this.Titulo = Titulo;
    }

    public String getTexto ()
    {
        return Texto;
    }

    public void setTexto (String Texto)
    {
        this.Texto = Texto;
    }

    public String getListaParadas ()
    {
        return ListaParadas;
    }

    public void setListaParadas (String ListaParadas)
    {
        this.ListaParadas = ListaParadas;
    }

    public String getAutor ()
    {
        return Autor;
    }

```

```

    public void setAutor (String Autor)
    {
        this.Autor = Autor;
    }

    public String getListasLineas ()
    {
        return ListaLineas;
    }

    public void setListasLineas (String ListaLineas)
    {
        this.ListaLineas = ListaLineas;
    }

    public String getFechaFin ()
    {
        return FechaFin;
    }

    public void setFechaFin (String FechaFin)
    {
        this.FechaFin = FechaFin;
    }

    public String getFechaActualizacion ()
    {
        return FechaActualizacion;
    }

    public void setFechaActualizacion (String FechaActualizacion)
    {
        this.FechaActualizacion = FechaActualizacion;
    }

    @Override
    public String toString()
    {
        return "ClassPojo [Subtitulo = "+Subtitulo+", FechaInicio = "+FechaInicio+", Categoría = "+Categoría+", FechaCreacion = "+FechaCreacion+", TransporteAfectado = "+TransporteAfectado+", Coordenadas = "+Coordenadas+", Titulo = "+Título+", Texto = "+Texto+", ListaParadas = "+ListaParadas+", Autor = "+Autor+", ListaLineas = "+ListaLineas+", FechaFin = "+FechaFin+", FechaActualizacion = "+FechaActualizacion+"]";
    }
}

```

8.2.5 Clase Coordenadas.java

```

public class Coordenadas
{
    private String Latitud;

    private String Longitud;

    public String getLatitud ()
    {
        return Latitud;
    }
}

```

```
public void setLatitud (String Latitud)
{
    this.Latitud = Latitud;
}

public String getLongitud ()
{
    return Longitud;
}

public void setLongitud (String Longitud)
{
    this.Longitud = Longitud;
}

@Override
public String toString()
{
    return "ClassPojo [Latitud = "+Latitud+", Longitud = "+Longitud+"]";
}
}
```