HERRAMIENTA DE GESTIÓN DE ALERTAS CATEGORIZADAS: MADALERT

TRABAJO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

AUTORES:
NEREA GÓMEZ DOMÍNGUEZ
SILVIA LENDÍNEZ FERNÁNDEZ
GONZALO MOLINA DÍAZ
ADRIÁN PANADERO GONZÁLEZ

PROFESOR TUTOR:
ANTONIO SARASA CABEZUELO

CURSO ACADÉMICO 2017/2018

AGRADECIMIENTOS

En primer lugar, queremos dar las gracias a nuestro tutor, Antonio Sarasa Cabezuelo, por darnos la oportunidad de realizar este proyecto y ayudarnos en el desarrollo del mismo aportando ideas, información y atendiendo a nuestras consultas cuando lo necesitábamos.

Agradecer también a nuestros amigos y compañeros que nos han estado animando cuando más difícil lo veíamos y aportando más ideas a nuestro proyecto.

Por último y más importante, el agradecimiento a nuestras familias, sin ellas no hubiéramos estudiado esta carrera y darles las gracias por estar en todo momento con nosotros.

PRÓLOGO

Todo comenzó durante el curso 2016-17 cuando se empezó a pensar cuál podría ser el tema del Trabajo de Fin de Grado, el coordinador e inclusive quiénes iban a ser los integrantes del grupo.

Inicialmente este proyecto iba a estar formado por más compañeros y aunque estaba claro que se quería realizar una herramienta de gestión de alertas, no se tenía del todo claro de qué tipo iban a ser. Tras darle varias vueltas al asunto, el grupo se formó por cuatro amigos que parecía que iban a congeniar e iban a trabajar bien como un equipo. El hecho de conocerse bien y haber trabajado anteriormente juntos hacía que fuera más fácil la decisión.

Durante esta etapa, en estos agotadores pero increíbles años de universidad, se va conociendo a gente, pero, como es lógico, solamente con algunos se llega a entablar una amistad. Qué mejor idea que juntarse con ellos para juntos poder realizar, siempre aprendiendo y luchando hasta el final, lo que podría considerarse nuestro último proyecto de todos estos años.

A principios de este curso, realizando lo que se conoce como un “brainstorming o lluvia de ideas” ya con nuestro coordinador, Antonio Sarasa, se decidió realizar un proyecto de gestión de alertas de sucesos en Madrid, queriendo dar visibilidad a incidencias que ocurren en nuestra ciudad, tanto en una página web como en una aplicación Android, para que los ciudadanos puedan estar informados de lo que ocurre de manera fácil y sencilla.

RESUMEN

El uso de aplicaciones en dispositivos móviles ha ido creciendo a lo largo de la última década a una velocidad vertiginosa, hasta llegar a convertirse hoy en día en una herramienta fundamental en la vida cotidiana de las personas.

Este proyecto presenta un sistema formado por una aplicación para Android y una aplicación web que permite la gestión de alertas sobre sucesos que ocurren en tiempo real utilizando la geolocalización que ofrecen los dispositivos móviles. Las alertas que pueden seguirse pueden ser de alguna de las categorías definidas en el sistema. La aplicación está realizada para ser utilizada por cualquier tipo de usuario que quiera mantenerse al tanto de las alertas que puedan ocurrir en la ciudad de Madrid (España) y sus 21 distritos en los que está dividida.

PALABRAS CLAVE:

Aplicación Android, aplicación web, geolocalización, alertas, Madrid, tiempo real, distritos, categorías, marcadores, mapa.

ABSTRACT

The use of applications in mobile devices has grown over the last decade at breakneck speed, having become a fundamental tool in people's daily lives.

This project presents an Android and a web application that allow the management of alerts about events that occur in real time using the geolocation offered by mobile devices. These alerts can be from the one of the categories defined in the system. The application is made to be used by any type of user who wants to keep abreast of the alerts that may occur in the city of Madrid (Spain) and its 21 districts in which it is divided.

KEYWORDS:

Android application, web application, geolocation, alerts, Madrid, real time, districts, categories, markers, map.

ÍNDICE GENERAL

AGRADECIMIENTOS *1*

PRÓLOGO *2*

RESUMEN *3*

ABSTRACT *4*

ÍNDICE GENERAL *5*

ÍNDICE DE FIGURAS *7*

1. INTRODUCCIÓN *15*

1.1 MOTIVACIÓN **15**

1.2 OBJETIVOS **16**

1.3 ESTADO DEL ARTE **16**

1.4 ESTRUCTURA DE LA MEMORIA **17**

1. INTRODUCTION *20*

1.1 MOTIVATION **20**

1.2 OBJECTIVES **21**

1.3 STATE OF THE ART **21**

1.4 MEMORY STRUCTURE **22**

2. ESPECIFICACIÓN DE LA APLICACIÓN *25*

2.1 FUNCIONALIDAD DE LA APLICACIÓN ANDROID **25**

2.1.1 DIAGRAMA DE CASOS DE USO 25

2.1.2 DIAGRAMAS DE ACTIVIDADES 26

2.2. FUNCIONALIDAD DE LA APLICACIÓN WEB **35**

2.2.1 DIAGRAMA DE CASOS DE USO 35

2.2.2 DIAGRAMAS DE ACTIVIDADES 36

2.3 FUNCIONALIDAD API Y RECOLECTOR DE DATOS SOCIALES **47**

2.3.1 FUNCIONALIDAD API 47

2.3.2 RECOLECTOR DE DATOS 48

3. TECNOLOGÍAS EMPLEADAS *49*

3.1 APLICACIÓN ANDROID **49**

3.2 APLICACIÓN WEB **50**

3.3 BASE DE DATOS **50**

4. ARQUITECTURA DE LA APLICACIÓN *53*

5. MODELO DE DATOS *55*

6. DISEÑO *61*

6.1 DISEÑO DE LA APP MAD ALERT **61**

6.2 DISEÑO DE LA PÁGINA WEB **69**

7. IMPLEMENTACIÓN *78*

7.1 IMPLEMENTACIÓN ANDROID **78**

7.2 IMPLEMENTACIÓN WEB **97**

7.3 IMPLEMENTACIÓN API Y RECOLECTOR DE DATOS **120**

7.3.1 IMPLEMENTACIÓN API REST 120

7.3.2 IMPLEMENTACIÓN RECOLECTOR DE DATOS 125

7.4 ALGORITMOS **147**

8. EVALUACIÓN *149*

9. CONCLUSIONES Y TRABAJO FUTURO *170*

9.1 CONCLUSIONES **170**

9.2 CONCLUSIONS **170**

9.3 TRABAJO FUTURO **171**

10. TRABAJO INDIVIDUAL *173*

10.1 NEREA GÓMEZ DOMÍNGUEZ **173**

10.2 SILVIA LENDÍNEZ FERNÁNDEZ **174**

10.3 GONZALO MOLINA DÍAZ **175**

10.4 ADRIÁN PANADERO GONZÁLEZ **176**

BIBLIOGRAFÍA *177*

APÉNDICES *187*

A- MANUAL DE INSTALACIÓN **187**

APLICACIÓN ANDROID 187

APLICACIÓN WEB 189

B- MANUAL DE USUARIO **196**

APLICACIÓN ANDROID 196

APLICACIÓN WEB 216

ÍNDICE DE FIGURAS

1- Figura 2.1.1.1 - Diagrama Casos de uso Android *26*

2- Figura 2.1.2.1 - A-Diagrama activ. - Cargar alertas según configuración *27*

3- Figura 2.1.2.2 - A-Diagrama activ. - Actualizar las pestañas *28*

4- Figura 2.1.2.3 - A-Diagrama activ. - Consultar alertas categoría y distrito *29*

5- Figura 2.1.2.4 - A-Diagrama activ. - Consultar alertas en el mapa *30*

7- Figura 2.1.2.6 - A-Diagrama activ. - Añadir alertas *32*

8- Figura 2.1.2.7 - A-Diagrama activ. - Contacto *33*

9- Figura 2.1.2.8 - A-Diagrama activ. - Soporte *34*

10 Figura 2.1.2.9 - A-Diagrama activ. - About us *34*

11- Figura 2.1.2.10 - A-Diagrama activ. - FAQ *35*

12- Figura 2.2.1.1 - Diagrama Casos de uso Web *35*

13- Figura 2.2.2.1 - W-Diagrama activ. - Ver alertas mapa *36*

14- Figura 2.2.2.2 - W-Diagrama activ. - Ver alertas marcador mapa *37*

15- Figura 2.2.2.3 - W-Diagrama activ. - Mostrar alertas por distrito y categoría *38*

16- Figura 2.2.2.4 - W-Diagrama activ. - Mostrar alertas por distrito *39*

17- Figura 2.2.2.5 - W-Diagrama activ. - Consulta estadísticas generales *40*

18- Figura 2.2.2.6 - W- Diagrama activ. - Consulta estadísticas por distritos *41*

19- Figura 2.2.2.7 - W-Diagrama activ. - Consulta estadísticas Policia Municipal Madrid *42*

20- Figura 2.2.2.8 - W-Diagrama activ. - Estadísticas accidentes / detenidos *43*

21- Figura 2.2.2.9 - W-Diagrama activ. - Añadir alertas *44*

22- Figura 2.2.2.10 - W-Diagrama activ. - Info de la web *45*

23- Figura 2.2.2.11 - W-Diagrama activ. - Actualizar mapa *45*

24- Figura 2.2.2.12 - W-Diagrama activ. - Actualizar por distritos y/o categoría *46*

25- Figura 2.3.1.1 - API *47*

26- Figura 2.3.2.1 - Recolector datos *48*

27- Figura 4.1 - Esquema arquitectura *53*

28- Figura 5.1 - Colecciones de la bd *55*

29- Figura 5.2 - Campos alertas Madridiario *56*

30- Figura 5.3 - Campos alertas Twitter *56*

31- Figura 5.4 - Campos estadísticas *57*

32- Figura 5.5 - Campos estadísticas detenidos *58*

33- Figura 5.6 - Campos estadísticas accidentes *59*

34- Figura 5.7 - Campos estadísticas seguridad *60*

35- Figura 5.8 - Alertas estadísticas seguridad *60*
36- Figura 6.1.1 - Tabs app *62*
37- Figura 6.1.2 - Icono menú "hamburguesa" *62*
38- Figura 6.1.3 - Menú lateral app *63*
39- Figura 6.1.4 - A-Prototipo 1 - Alertas *65*
40- Figura 6.1.5 - A-Prototipo 2 - Mapa *66*
41- Figura 6.1.6 - A-Prototipo 3 - Distrito *67*
42- Figura 6.1.7 - Vista previa paleta *68*
43- Figura 6.1.8 - Paleta detallada *68*
44- Figura 6.1.9 - Resultado aplicación paleta *69*
45- Figura 6.2.1 - W-Prototipo 1 - Alertas *73*
46- Figura 6.2.2 - W-Prototipo 2 - Estadísticas generales *74*
47- Figura 6.2.3 - W-Prototipo 3 - Estadísticas distritos *74*
48- Figura 6.2.4 - W-Prototipo 4 - Información - Soporte *75*
49- Figura 6.2.5 - W-Prototipo 5 - Información - About us *76*
50- Figura 7.1.1 - Permisos AndroidManifest *78*
51- Figura 7.1.2 - Paquetes Android Studio *79*
52- Figura 7.1.3 - **initSharedPreferences()** *80*
53- Figura 7.1.4 - Atributos privados **SharedPreferences** *80*
54- Figura 7.1.5 - Inicialización atributo editor *80*
55- Figura 7.1.A-1.1 - Comprobación primer inicio en la aplicación *81*
56- Figura 7.1.A-1.2 - Obtener distritos cuando el usuario tiene la ubicación activada *82*
57- Figura 7.1.A-1.3 - Obtener distritos cuando la ubicación no está activada. *82*
58- Figura 7.1.A-1.4 - Obtención de alertas mediante petición a la api *82*
59- Figura 7.1.A-1.5 - Rellenado de información de cada alerta de la lista *83*
60- Figura 7.1.A-2.1 - Actualización de alertas y distritos *84*
61- Figura 7.1.A-2.2 - Actualización de la pestaña mapa *84*
62- Figura 7.1.A-3.1 - Selección de distrito y categorías *85*
63- Figura 7.1.A-3.2 - Mostrar alertas según distrito y categorías *86*
64- Figura 7.1.A-3.3 - Petición de alertas si tiene categorías o es Todas *86*
65- Figura 7.1.A-3.4 - SeleccionarDistrito *86*
66- Figura 7.1.A-3.5 - ListaAlertas *86*
67- Figura 7.1.A-4.1 - Obtener distritos dentro del radio y ubicación *88*
68- Figura 7.1.A-4.2 - Crear marcadores en el mapa *89*
69- Figura 7.1.A-5.1 - Funciones clase ConfigActivity *89*
71- Figura 7.1.A-5.3 - Atributos en **SharedPreferences** *en Configuración* *90*

72- Figura 7.1.A-5.4 - Obtener distritos en función del radio *91*
73- Figura 7.1.A-6.1 - Validación de categoría con el texto *92*
74- Figura 7.1.A-6.2 - Añadir alertas *92*
75- Figura 7.1.A-7.1 - Validación campos rellenados *93*
76- Figura 7.1.A-8.1 - SoporteActivity *94*
77- Figura 7.1.A-8.2 - Soporte *94*
78- Figura 7.1.A-9.1 - AboutUsActivity *95*
79- Figura 7.1.A-9.2 - About us *95*
80- Figura 7.1.A-10.1 - FAQActivity *95*
81- Figura 7.1.A-10.2 - FAQ *96*
82- Figura 7.1.2 - Diagrama de clases Android *96*
83- Figura 7.2.W-1.1 - Código inicializar mapa *97*
84- Figura 7.2.W-1.2 - Código inicializar mapa - 2 *98*
85- Figura 7.2.W-1.3 - Mapa *98*
86- Figura 7.2.W-1.4 - Marcador Mapa *99*
87- Figura 7.2.W-1.5 - Alertas por distrito *99*
88- Figura 7.2.W-2.1 - Código búsqueda en index *100*
89- Figura 7.2.W-2.2 - Código mostrar distritos *100*
90- Figura 7.2.W-2.3 - Búsqueda en index *101*
91- Figura 7.2.W-2.4 - Mensaje error "al menos una categoría" *101*
92- Figura 7.2.W-2.5 - Código función validación categorías *101*
93- Figura 7.2.W-3.1 - Buscar alertas por distrito *102*
94- Figura 7.2.W-3.2 - Mostrar alertas por distrito *102*
95- Figura 7.2.W-3.3 - Mensaje no hay alertas *103*
96- Figura 7.2.W-4.1 - Buscador estadísticas generales *103*
97- Figura 7.2.W-4.2 - Código estadísticas generales *104*
98- Figura 7.2.W-4.3 - Código función obtener mes *104*
99- Figura 7.2.W-4.4 - Código obtener est. generales *105*
100- Figura 7.2.W-4.5 - Código obtener est. generales DAO *105*
101- Figura 7.2.W-4.6 - Estadísticas según una categoría concreta *105*
102- Figura 7.2.W-4.7 - Estadísticas generales mes de Abril *106*
103- Figura 7.2.W-4.8 - Estadísticas generales mes de Marzo *106*
104- Figura 7.2.W-4.9 - Estadísticas generales cursor encima *107*
105- Figura 7.2.W-4.10 - Función comprobación estad. generales *107*
106- Figura 7.2.W-4.11 - Código de la página de estad. generales *108*
107- Figura 7.2.W-5.1 - Código de la página de estad. por distritos *109*

108- Figura 7.2.W-5.2 - Buscador estadísticas por distritos *109*
109- Figura 7.2.W-5.3 - Estadísticas según un distrito concreto *109*
110- Figura 7.2.W-5.4 - Estadísticas por distrito mes de Abril *110*
111- Figura 7.2.W-5.5 - Estadísticas por distrito del mes de Marzo *110*
112- Figura 7.2.W-5.6 - Mensaje no hay estadísticas *110*
113- Figura 7.2.W-6.1 - Buscador estadísticas Policía *111*
114- Figura 7.2.W-6.2 - Estadísticas de detenidos por distrito *112*
115- Figura 7.2.W-6.3 - Estadísticas de accidentes por distrito *112*
116- Figura 7.2.W-6.4 - Estadísticas relacionadas con la seguridad *113*
117- Figura 7.2.W-7.1 - Código aniadirAlertas.php *114*
118- Figura 7.2.W-7.2 - Formulario añadir alertas *114*
121- Figura 7.2.W-7.5 - Procesar añadir alerta *115*
122- Figura 7.2.W-7.6 - Clasificador alertas *116*
123- Figura 7.2.W-8.1 - Soporte *117*
124- Figura 7.2.W-8.2 - Código About us *117*
125- Figura 7.2.W-8.3 - About us *118*
126- Figura 7.2.W-9.1 - Código actualizar mapa (index) *118*
127- Figura 7.2.W-9.2 - Botón actualizar mapa (index) *119*
128- Figura 7.2.W-10.1 - Botón actualizar alertas *119*
129- Figura 7.2.2 - Diagrama de clases de la aplicación web *120*
130- Figura 7.3.1.1 - Esquema de alertas *121*
131- Figura 7.3.1.2 - Definición de rutas del node.js *121*
132- Figura 7.3.1.3 - Tabla de clases y recursos de la API *122*
133- Figura 7.3.1.4 - Función de la API para obtener las alertas por distrito *123*
134- Figura 7.3.1.5 - Función que devuelve las alertas por distrito y categoría *123*
135- Figura 7.3.1.6 - Función que devuelve el nombre del distrito y el total de alertas por categoría *124*
136- Figura 7.3.1.7 - Función que añade una nueva alerta *124*
137- Figura 7.3.2.1 - Clase base de datos *125*
138- Figura 7.3.2.2 - Funciones clase base de datos *126*
139- Figura 7.3.2.3 - Función eliminar estadísticas accidentes *126*
140- Figura 7.3.2.4 - Código web scraping *127*
141- Figura 7.3.2.5 - Clase clasificador *128*
142- Figura 7.3.2.6 - Funciones clasificador *128*
143- Figura 7.3.2.7 - Código 5 min *129*
144- Figura 7.3.2.8 - Ruta .py *129*
145- Figura 7.3.2.9 - Nuevas carpetas en directorio .py *130*

146- Figura 7.3.2.10 - Nuevo archivo exe *130*
147- Figura 7.3.2.11 - Buscar programador de tareas *131*
148- Figura 7.3.2.12 - Lista acciones *132*
149- Figura 7.3.2.13 - Cron: Crear tarea *132*
150- Figura 7.3.2.14 - Cron: Nombre y descripción nueva tarea *133*
151- Figura 7.3.2.15 - Cron: Nueva acción *133*
152- Figura 7.3.2.16 - Cron: Desencadenador Programación *134*
153- Figura 7.3.2.17 - Twitter: MyListener.py *135*
154- Figura 7.3.2.18 - Twitter: MyListener.py - 2 *135*
155- Figura 7.3.2.19 - Twitter: ObtenerStream.py *136*
156- Figura 7.3.2.20 - ObtenerStream ejecutable *136*
157- Figura 7.3.2.21 - Obtener excel .py *137*
158- Figura 7.3.2.22 - insertarDatosPolicia.py *138*
159- Figura 7.3.2.23 - insertarDatosPolicia.py - 2 *138*
160- Figura 7.3.2.24 - InsertarDatosPolicia ejecutable *139*
161- Figura 7.3.2.25 - Borrado alertas *140*
162- Figura 7.3.2.26 - Borrado estadísticas *140*
163- Figura 7.3.2.27 - Borrado estadísticas Policía *141*
164- Figura 7.3.2.28 - Módulos: distritos *142*
165- Figura 7.3.2.29 - Root: Lista distritos *142*
166- Figura 7.3.2.30 - Clasificador: News *143*
167- Figura 7.3.2.31 - Clasificador: tweets *143*
168- Figura 7.3.2.32 - Categorías clasificador *144*
169- Figura 7.3.3.1 - Algoritmo cálculo de distancia entre dos coordenadas *147*
170- Figura 7.3.3.2 - Algoritmo para obtener los distritos que están dentro de una circunferencia *148*
171- Figura 8.a.1 - Pregunta 1 *149*
172- Figura 8.a.2 - Pregunta 2 *150*
173- Figura 8.a.3 - Pregunta 3 *150*
174- Figura 8.a.4 - Pregunta 4 *151*
175- Figura 8.a.5 - Pregunta 5 *151*
176- Figura 8.a.6 - Pregunta 6 *152*
177- Figura 8.a.7 - Pregunta 7 Noticias *152*
178- Figura 8.a.8 - Pregunta 7 Redes Sociales *153*
179- Figura 8.a.9 - Pregunta 8 *153*
180- Figura 8.a.10 - Pregunta 9 *154*
181- Figura 8.a.11 - Pregunta 10 *154*

182- Figura 8.a.12 - Pregunta 11 *155*
183- Figura 8.a.13 - Pregunta 12 *155*
184- Figura 8.a.14 - Pregunta 13 *156*
185- Figura 8.a.15 - Pregunta 14 *156*
186- Figura 8.a.16 - Pregunta 15 *157*
187- Figura 8.a.17 - Pregunta 16 *157*
188- Figura 8.a.18 - Pregunta 17 *158*
189- Figura 8.a.19 - Pregunta 18 *158*
190- Figura 8.a.20 - Pregunta 19 *159*
191- Figura 8.c.1 - Pregunta 1 *162*
192- Figura 8.c.2 - Pregunta 2 *163*
193- Figura 8.c.3 - Pregunta 3 *163*
194- Figura 8.c.4 - Pregunta 4 *164*
195- Figura 8.c.5 - Pregunta 5 *164*
196- Figura 8.c.6 - Pregunta 6 *165*
197- Figura 8.c.7 - Pregunta 7 *165*
198- Figura 8.c.8 - Pregunta 8 *166*
199- Figura 8.c.9 - Pregunta 9 *166*
200- Figura 8.c.10 - Pregunta 10 *167*
201- Figura 8.c.11 - Pregunta 11 *167*
202- Figura 8.c.12 - Pregunta 12 *168*
203- Figura 8.c.13 - Pregunta 13 *168*
204- Figura A.A-2.1 - Iniciar sistema nodemon *188*
205- Figura A.A-2.2 - Icono Android MadAlert *188*
206- Figura A.A-2.3 - Emulador *188*
207- Figura A.W-1.1 - Tabla paquetes python *190*
208- Figura A.W-1.2 - PHPInfo en XAMPP. *192*
209- Figura A.W-1.3 - mongodb en PHPInfo de XAMPP *192*
210- Figura A.W-2.1 - Run python *193*
211- Figura A.W-2.2 - Corriendo web_scraping.py *194*
212- Figura A.W-2.3 - Iniciando aplicación web *195*
213- Figura B.A.1 - Iniciando la app *196*
214- Figura B.A.2 - Mensaje para que configure la aplicación *197*
215- Figura B.A.3 - Menú *198*
216- Figura B.A.4 - Configuración *199*
217- Figura B.A.5 - Pestaña alertas con radio 0 kms *200*

218- Figura B.A.6 - Muestra de alertas con radio mayor que 0 kms *201*
219- Figura B.A.7 - Mensaje no hay alertas *201*
222- Figura B.A.10 - Mensaje en el campo que falta *203*
223- Figura B.A.11 - Añadir alerta con categoría incorrecta y añadida correctamente *203*
224- Figura B.A.12 - Mapa con radio 0 y ubicación activada *204*
225- Figura B.A-13 - Mapa con radio mayor de 0 kms y ubicación activada *205*
226- Figura B.A.14 - Mapa cuando se elige distrito y radio es 0 *206*
227- Figura B.A.15 - Mapa cuando se elige distrito y el radio es mayor que 0 *207*
230- Figura B.A.18 - Mensaje seleccionar al menos una categoría *209*
233- Figura B.A.21 - Visualización alertas de un distrito *210*
234- Figura B.A.22 - Selección varias categorías *211*
235- Figura B.A.23 - Alertas con varias categorías *211*
240- Figura B.A.28- Contacto *214*
241- Figura B.A.29- Soporte *215*
244- Figura B.W.1- Logo *216*
245- Figura B.W.2- Página de inicio *217*
246- Figura B.W.3- Mapa *217*
247- Figura B.W.4- Marcador mapa *218*
248- Figura B.W.5- Alertas distrito Retiro *219*
249- Figura B.W.6- Página noticia *220*
250- Figura B.W.7- Página noticia Parte inferior mapa *220*
251- Figura B.W.8- Selección categorías mapa *221*
252- Figura B.W.9 - Alertas distritos por categorías *221*
253- Figura B.W.10 - Menú lateral izquierdo *222*
254- Figura B.W.11 - Selección de distrito *223*
255- Figura B.W.12 - Mostrar distrito seleccionado *223*
256- Figura B.W.13 - Listado de alertas por distrito *224*
257- Figura B.W.14 - Formulario añadir nueva alerta *225*
258- Figura B.W.15 - Submenú Estadísticas *225*
259- Figura B.W.16 - Estad. Generales desplegable categorías *226*
260- Figura B.W.17 - Buscar est. generales por categoría seleccionada *226*
261- Figura B.W.18 - Estadísticas generales mes 1 *227*
262- Figura B.W.19 - Estadísticas generales mes 2 *227*
263- Figura B.W.20 - Desplegable Distrito *228*
264- Figura B.W.21 - Mostrar est. distrito seleccionado *228*
265- Figura B.W.22 - Estadísticas por distritos mes 1 *229*

266- Figura B.W.23 - Estadísticas por distritos mes 2 *229*
267- Figura B.W.24 - Desplegable Estadísticas Policía *229*
268- Figura B.W.25 - Mostrar Estadísticas Policía seleccionadas *230*
269- Figura B.W.26 - Gráfico estadísticas seguridad *230*
270- Figura B.W.27 - Submenú Información *231*
271- Figura B.W.28 - Información Soporte *231*
272- Figura B.W.29 - Información About us *232*
273- Figura B.W.30 - Formulario de contacto *232*
274- Figura B.W.31 - Error en formulario de contacto *233*

1. INTRODUCCIÓN

En un mundo globalizado y digitalizado como el actual resulta realmente indispensable conocer qué y dónde están ocurriendo hechos que pueden afectar a la vida cotidiana de las personas. La evolución de la tecnología ha traído como consecuencia que gran parte de la sociedad haga uso de dispositivos móviles para realizar sus tareas diarias e informarse acerca de lo que pasa en todo momento.

Esa necesidad existente de mantenerse constantemente informado es lo que ha motivado la realización de este proyecto. En este sentido, el objetivo principal de este trabajo es ayudar a los usuarios a gestionar alertas de noticias sobre sucesos que les interesan en tiempo real.

En este contexto se ha concebido este Trabajo de Fin de Grado correspondiente al Grado en Ingeniería Informática, en el cual se ha desarrollado una aplicación móvil orientada a cubrir las necesidades de información de los usuarios y una aplicación web donde además se recogerán estadísticas de interés general. Cabe remarcar que este proyecto se ha centrado únicamente en la ciudad de Madrid, aunque las técnicas utilizadas permiten generalizarlo a cualquier ciudad.

Se ha elegido la ciudad de Madrid debido a que todos los integrantes del grupo residen en ella, la conocen y creen que sería algo útil.

1.1 MOTIVACIÓN

En los últimos años, el crecimiento del uso del Smartphone ha traído como consecuencia que los usuarios quieran conocer de primera mano y en tiempo real los acontecimientos que ocurren a su alrededor. El problema reside en que para conocer estos hechos han de buscarlos ellos mismos.

Por ello, se decidió crear una aplicación Android que tratase de unificar todas estas noticias y otorgar al usuario la libertad de mostrarle lo que él desea. Es aquí donde se encuentra la motivación de este proyecto: facilitar al usuario el acceso a una información específica.

Por otro lado, también se consideró importante el desarrollo de una aplicación web como una forma alternativa de acceder a esta información (sin tener que descargar la aplicación en el teléfono). Hoy en día muchas personas prefieren utilizar sus ordenadores personales como medio habitual de acceso a diferentes noticias. Además, la visualización de las estadísticas con sus gráficos correspondientes se realiza mejor en una pantalla grande como es la de un ordenador que en la de un teléfono móvil.

1.2 OBJETIVOS

El objetivo principal del presente Trabajo de Fin de Grado es la creación de un sistema formado por una aplicación Android que permite a los ciudadanos de la ciudad de Madrid gestionar las alertas de las noticias en las que están interesados, y una aplicación web que facilite un acceso alternativo a la misma información y a estadísticas generadas a través de los datos obtenidos.

Los objetivos específicos que plantea el proyecto son los siguientes:

- Confeccionar una aplicación móvil y web que resulte atractiva e intuitiva a los usuarios, de manera que facilite su uso y compresión.

- Diseñar y elaborar una base de datos que recolecta datos procedentes de diferentes fuentes y mostrarlos de manera categorizada y en tiempo real a los usuarios.

- Mostrar desde la aplicación web una serie de estadísticas elaboradas a partir de los datos obtenidos y distinguirlas según tres filtros. Estos tres filtros que se plantean para ello son: por distritos, por categorías o por fuentes de procedencia.

- Conseguir extraer la geolocalización de un móvil, de forma que el usuario obtenga la información requerida respecto a su situación geográfica.

- Establecer un sistema que permita a los usuarios introducir alertas ellos mismos, guardándose las mismas en la base de datos y distinguirlas de aquellas alertas procedentes de sitios oficiales.

- Permitir al usuario la personalización de las alertas a recibir a través de un conjunto de filtros facilitados por la aplicación.

- Acceso libre, el acceso tanto a la aplicación web como a la aplicación móvil se realizará sin necesidad de autenticarse.

1.3 ESTADO DEL ARTE

Si bien es cierto que existen varias aplicaciones que utilizan la geolocalización y la notificación en tiempo real relacionado con el tiempo de espera de autobuses y metro de la ciudad de Madrid, tales como ‘Urban Step Madrid’[49] o ‘Madrid MBC’[46], no se ha

encontrado ninguna herramienta que unifique todas las alertas en una sola aplicación tal y como se pretende llevar a cabo en este proyecto.

Si se busca alguna aplicación similar respecto a la alerta de noticias, se puede mencionar a los diferentes lectores de noticias que cumplen una función similar. En Android los más destacados podrían ser 'Feedly'[45], 'Appy Geek'[43] y, en menor medida porque también actúa como red social, 'Reddit'[47]. Todas ellas tienen la característica en común de mantener informado al usuario a través de notificaciones.

Entre las aplicaciones existentes en el Play Store, las que más características en común tienen con este proyecto son las siguientes:

- **Citizen. Safety & Awareness** [44]

Se trata de una aplicación que alerta a sus usuarios de los delitos cometidos en la ciudad de Nueva York en tiempo real. La aplicación está en contacto directo con la policía neoyorkina, por lo que cuando uno de sus agentes recibe un aviso de la central, Citizen recoge los datos y los emite a sus usuarios.

- **SocialDrive** [48]

SocialDrive trata de crear una comunidad de conductores con el propósito de compartir información acerca del tráfico en tiempo real.

- **Urban Step** [49]

Detecta la geolocalización de una persona y ofrece un mapa interactivo con todas las paradas de buses que tienes alrededor. Al pulsar sobre la parada elegida, muestra el tiempo de espera de los buses.

En definitiva, existen varias aplicaciones que se sirven de la geolocalización del usuario para mostrar información en tiempo real, pero ninguna de las analizadas se corresponde con las características planteadas en este trabajo fin de grado.

1.4 ESTRUCTURA DE LA MEMORIA

La memoria está organizada en un total de diez capítulos, un resumen, dos índices: uno general sobre los contenidos de la memoria y otro sobre las figuras de la memoria, la bibliografía empleada y un apéndice. A continuación, se presentan brevemente los capítulos mencionados:

- **1. Introducción**

En este capítulo se exponen los objetivos y motivaciones del proyecto realizado, se muestra un breve estado del arte actual y se detalla la estructura que seguirá este documento.

- **2. Especificación de la aplicación**

En este capítulo se detallan las funcionalidades, tanto de la aplicación móvil como de la aplicación web. Se proporcionan los correspondientes diagramas de casos de uso y de actividades.

- **3. Tecnologías empleadas**

En este capítulo se exponen las tecnologías que se han utilizado para implementar la aplicación, comentando las características de cada una de ellas y el por qué se han escogido para el desarrollo de este trabajo.

- **4. Arquitectura de la aplicación**

En este capítulo se presentan los módulos que constituyen la aplicación, así como su organización.

- **5. Modelo de datos**

En este capítulo se explica la estructura y distribución de las bases de datos empleadas en la aplicación.

- **6. Diseño**

En este capítulo se presenta el diseño que se ha aplicado realizar tanto la aplicación web como la aplicación Android. Se incide en los patrones y principios de diseños empleados.

- **7. Implementación**

En este capítulo se profundiza sobre cómo se ha desarrollado este proyecto. Se describe los algoritmos usados, el recolector de datos empleado y diferentes detalles acerca de las librerías y plugins utilizados.

- **8. Evaluación**

En este capítulo, se realizan varias evaluaciones con usuarios tanto de la iniciativa como del resultado final de este proyecto. Gracias a ellas ha sido posible identificar errores o mejoras.

- **9. Conclusiones y trabajo futuro**

En este capítulo se detallan las principales conclusiones obtenidas en la realización del proyecto y se proponen distintas líneas de trabajo futuro.

- **10. Trabajo individual**

En este capítulo se expone de manera detallada el trabajo realizado y la experiencia personal adquirida por parte de cada uno de los miembros del equipo.

- **Bibliografía**

En este apartado se listan las distintas referencias usadas durante la elaboración del proyecto, incluyendo direcciones web de foros, tutoriales y páginas web oficiales empleadas.

- **Apéndices**

Se presentan dos apéndices. El primero detalla paso a paso la instalación del sistema tanto de la aplicación web como de la aplicación Android y el segundo constituye un manual de ayuda al usuario para el correcto empleo de las distintas funcionalidades ofrecidas.

1. INTRODUCTION

In a globalized and digitized world like the one we live in it is a must to know what events are happening in our daily lives, where they are occurring and the potential effects they may have. The evolution of technology has resulted in a large part of society making use of mobile devices to perform their daily tasks and inform themselves about what is happening at all times.

This existing need to stay constantly informed is what has pushed us to create this project. The purpose is to help users manage the alerts and news they want to receive, about matters that really interest them and in real time.

Taking all this into account, this final Degree Project; corresponding to the degree in Computer Engineering, is conceived with the idea of creating a mobile application that covers these needs of the users as well as a web application where statistics will be collected with regards to general interest. It should be noted that in order to carry out this project we have focused solely on the city of Madrid, although the techniques used allow it to be generalized to any city.

The city of Madrid has been chosen because all the members of the group reside in it, they know it and believe it would be useful.

1.1 MOTIVATION

In recent years, due to the increase in the use of smartphones, users want to know first-hand and in real time the events that occur around them. The information will need to be sought after by the user in order to remain informed.

For this reason it has been decided to create an Android application that tries to unify all this news and grant the user the freedom to know exactly what they want to know. This is where the motivation behind this project comes from: the fact of providing the user with access to specific information.

In addition, the existence of the web application could be explained as an alternative way of acceding to this information (without having to download the application on the phone). Nowadays many people prefer to use computers as a regular means of accessing different news. And not only this, the visualization of statistics with their corresponding graphics is best done on a large screen such as a computer instead of a mobile phone screen.

1.2 OBJECTIVES

The main objective of this Final Degree Project is the creation of a system formed by an Android application that will allow the citizens of the city of Madrid to manage the news alerts that they are interested in, and a web application that provides an alternative access to the same information and statistics generated through the data obtained.

The specific objectives proposed by the project are as follows:

- Create a mobile and web application that is attractive and intuitive for the user in order to facilitate the use and understanding of it.
- Design and develop a database that collects data from different kind of sources and display them to users by categories and in real time.
- Show various statistics from the web application based on the data obtained and distinguish it within three filters: by districts, by categories or by sources of origin.
- Get to extract the geolocation of a mobile, so that the user obtains the required information regarding their geographical location.
- Establish a system that allows users to input alerts themselves, keeping them in the database and distinguishing them from other alerts coming from official sites.
- Allow the user to customize the alerts through filters that will be created and differentiated by the application.
- Free access, access to both the web application and the mobile application will be done without the need to authenticate.

1.3 STATE OF THE ART

Although it is true that there are several existing applications that use geolocation and real time notification in relation to bus and metro waiting times within the city of Madrid, like 'Urban Step Madrid'[96] or 'Madrid MBC'[46], it is hard to found any tool that combines all alerts in a single application, as is tried to carry out with this project.

Looking for a similar application about the news alert, it is possible to mention some different news readers that fulfill an alike function. In the most out-standing Android they might be ‘Feedly’[45], ‘Appy Geek’[43] and, in minor measure because it also acts as a social network too, ‘Reddit’[47]. All of them have the common characteristic of keeping the user informed through notifications.

Among the existing applications in the Play Store, those that have more features in common with our project are the following:

- **Citizen. Safety & Awareness**[44]

An app that alerts its users in real time when crimes have been committed in New York City. The application is in direct contact with the New York Police Department so when one of its agents receives an alert from the central control office, Citizen collects the data and sends it to its users.

- **SocialDrive**[48]

It is made up of a community of drivers with the purpose of sharing information about traffic in real time.

- **Urban Step**[49]

It is able to detect the geolocation of a person and offers an interactive map of bus stops around you. When you click on the chosen stop it shows you the waiting time for the buses.

In conclusion, there are several applications that use the geolocation of the user to show information in real time, however none of the analyzed corresponds to the characteristics proposed in this final degree project.

1.4 MEMORY STRUCTURE

The memory is organized in a total of ten chapters, a summary and two indices: one general about the content of this memory and another on the figures of the memory, the bibliography used and an appendix. The following chapters are briefly presented below:

- **1. Introduction**

In this chapter the objectives and motivations of the project are exposed, the current brief state of the art is shown and the structure that this document will follow is detailed.

- **2. Specification**

This chapter details the functionalities of both the mobile application and the web application. The corresponding diagrams of use cases and activities are provided.

- **3. Technologies used**

In this chapter the technologies that the application has been used are exposed, commenting on the features of each of them and why they have been chosen for the development of this work.

- **4. App architecture**

In this chapter we present the modules that make up the application, as well as its organization.

- **5. Data model**

This section explains the structure and distribution that the databases used in the application.

- **6. Design**

This section presents the design that has been taken into consideration when making both the web application and the Android application. Incising in the patterns and principles of design used.

- **7. Implementation**

This section delves into how this project has developed. It describes the algorithms used, the data collector used and different details about the libraries and plugins used.

- **8. Evaluation**

In this chapter, several evaluations are carried out with users of both the initiative and the final result of this project. Thanks to them it has been possible to identify errors or improvements.

- **9. Conclusions and future work**

In this chapter, the main conclusions obtained in the realization of the project are detailed and are proposed different lines of future work.

- **10. Individual work**

In this chapter, the work carried out and the personal experience acquired by each of the team members is detailed.

- **Bibliography**

In this chapter, the different references used during the elaboration of the project are listed, including web addresses of forums, tutorials and official websites used.

- **Appendices**

Two appendices are presented: the first one details step by step the installation of the web application and the Android application and the second one constitutes a help manual to use correctly the different functionalities offered.

2. ESPECIFICACIÓN DE LA APLICACIÓN

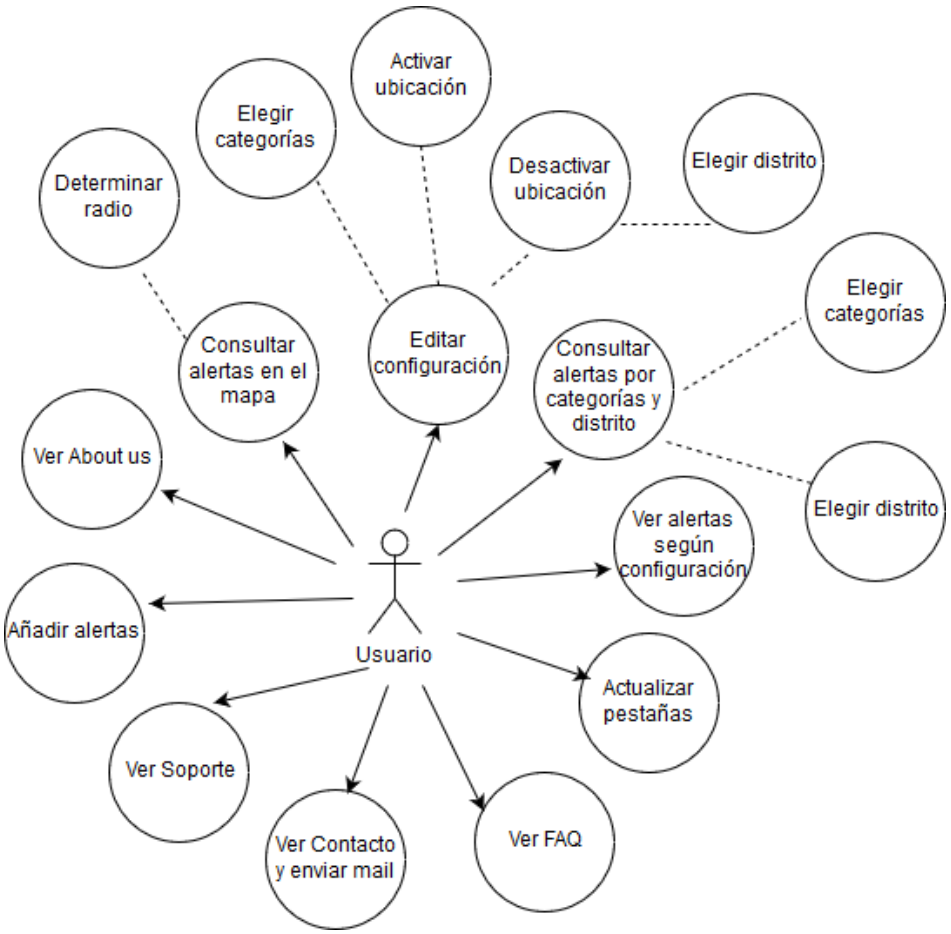
En este apartado se detalla la especificación tanto de aplicación web como de la aplicación Android. En ambos casos, primero se muestra un diagrama con todos los casos de uso para después describir cada uno de ellos con su correspondiente diagrama de actividades.

2.1 FUNCIONALIDAD DE LA APLICACIÓN ANDROID

La funcionalidad principal de la aplicación Android es informar al usuario en tiempo real de las alertas que suceden en los distritos Madrid, de acuerdo con las preferencias del usuario. Éste puede buscar alertas en distritos y categorías concretas, así como visualizar las alertas en el mapa de Madrid.

2.1.1 DIAGRAMA DE CASOS DE USO

En la Figura 2.1.1.1 se muestra un diagrama con los casos de uso posibles para el usuario de la aplicación Android.



1- Figura 2.1.1.1 - Diagrama Casos de uso Android

2.1.2 DIAGRAMAS DE ACTIVIDADES

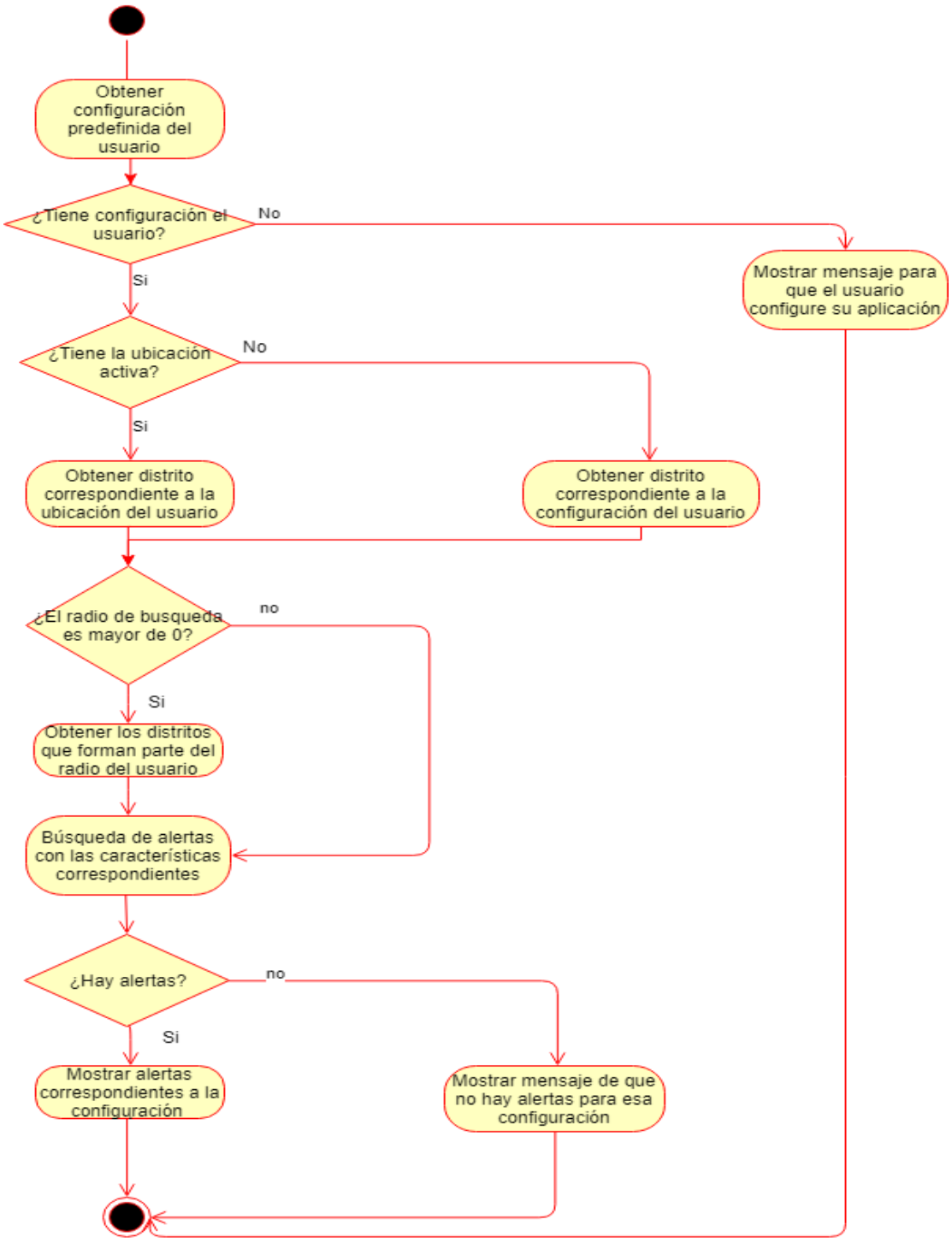
En este apartado se explicarán detalladamente todas las funcionalidades que abarca la aplicación Android junto con sus correspondientes diagramas de actividades.

A-1: Cargar alertas según la configuración del usuario

En la pestaña de 'Alertas' de la aplicación móvil aparecerán las alertas correspondientes a la configuración que tiene el usuario. Si el usuario no tiene configurada la aplicación aparecerá un mensaje informativo para que la establezca a su gusto.

Si la configuración que ha elegido es la de buscar alertas a través de la ubicación, se procederá a obtener el distrito correspondiente a sus coordenadas y se realizará una búsqueda de alertas sobre ese distrito y los distritos que engloba el radio seleccionado, aplicando el filtro de categorías elegido.

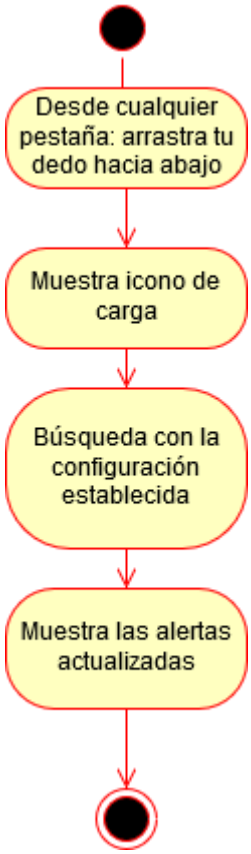
Si el usuario tiene la ubicación desactivada entonces se buscarán las alertas en el distrito elegido y en todos los distritos que engloba el radio, aplicando el filtro de categorías. En la Figura 2.1.2.1 se muestra el diagrama de actividades correspondiente a este caso de uso.



2- Figura 2.1.2.1 - A-Diagrama activ. - Cargar alertas según configuración

A-2: Actualizar las pestañas

La aplicación dispondrá de un conjunto de pestañas denominadas ‘ALERTAS’, ‘MAPA’ y ‘DISTRITOS’ que podrán refrescarse tantas veces como el usuario quiera. Si hay alguna alerta nueva respecto a la configuración en ese momento establecida o la búsqueda realizada, esta se mostrará. Este refresco se realizará como en todas las aplicaciones Android, arrastrando el dedo hacia abajo de la pantalla: en ese momento aparecerá durante breves instantes el icono de carga para después mostrar los datos obtenidos.
En la Figura 2.1.2.2 se muestra el diagrama de actividades correspondiente a este caso de uso.



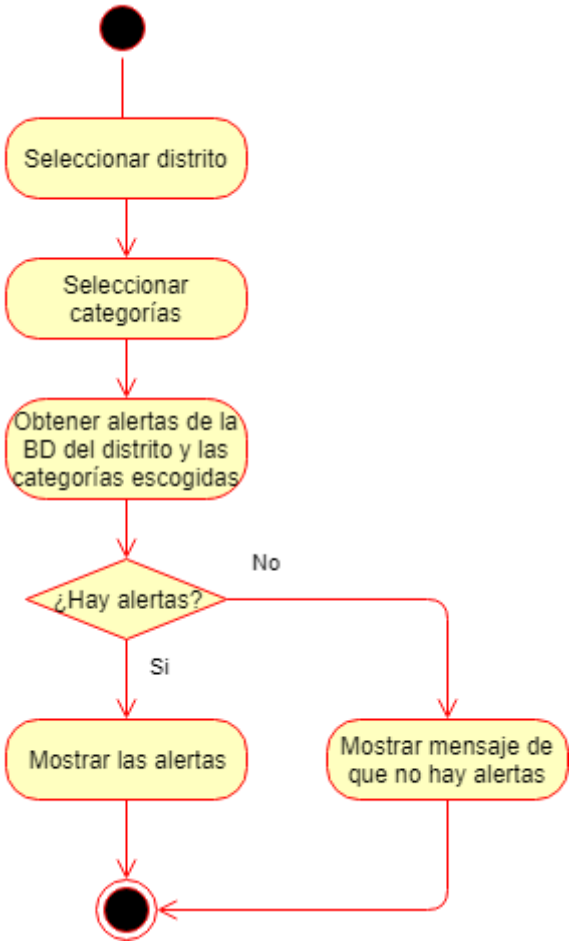
3- Figura 2.1.2.2 - A-Diagrama activ. - Actualizar las pestañas

A-3: Consultar alertas por categoría y distrito

Será posible consultar las alertas del distrito y las categorías deseadas a través de esta funcionalidad. Para ello será necesario seleccionar un distrito y al menos una categoría para que la búsqueda de alertas pueda realizarse. Al realizar la búsqueda se mostrarán las alertas del distrito y de las categorías seleccionadas, si ese distrito no dispone de alertas o las alertas que tiene no corresponden a las categorías seleccionadas se indicará con un mensaje.

Si se intenta realizar una búsqueda sin seleccionar una categoría se mostrará un mensaje de error y no se hará la búsqueda de alertas.

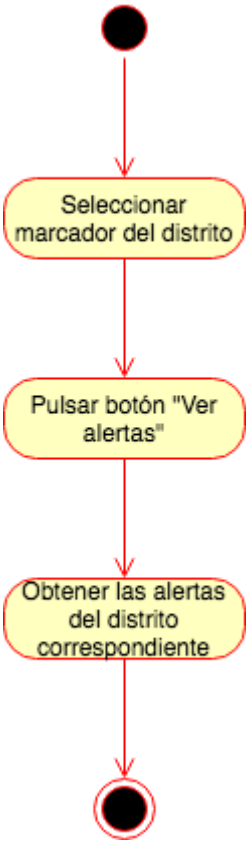
En la Figura 2.1.2.3 se muestra el diagrama de actividades correspondiente a este caso de uso.



4- Figura 2.1.2.3 - A-Diagrama activ. - Consultar alertas categoría y distrito

A-4: Consultar alertas en el mapa de Madrid

A través del mapa se mostrarán una serie de marcadores referenciando el distrito. Al pulsar en uno de los marcadores, aparecerá una ventana informando del nombre del distrito, el número de alertas y un botón. Si se pulsa dicho botón se verán las alertas de todas las categorías que tiene ese distrito.
En la Figura 2.1.2.4 se muestra el diagrama de actividades correspondiente a este caso de uso.



5- Figura 2.1.2.4 - A-Diagrama activ. - Consultar alertas en el mapa

A-5: Editar configuración del usuario

Para configurar la aplicación acorde a las preferencias de cada usuario, este deberá seleccionar una opción de 'Configuración' que estará disponible desde el menú desplegable. En este apartado el usuario podrá encontrar tres opciones configurables:

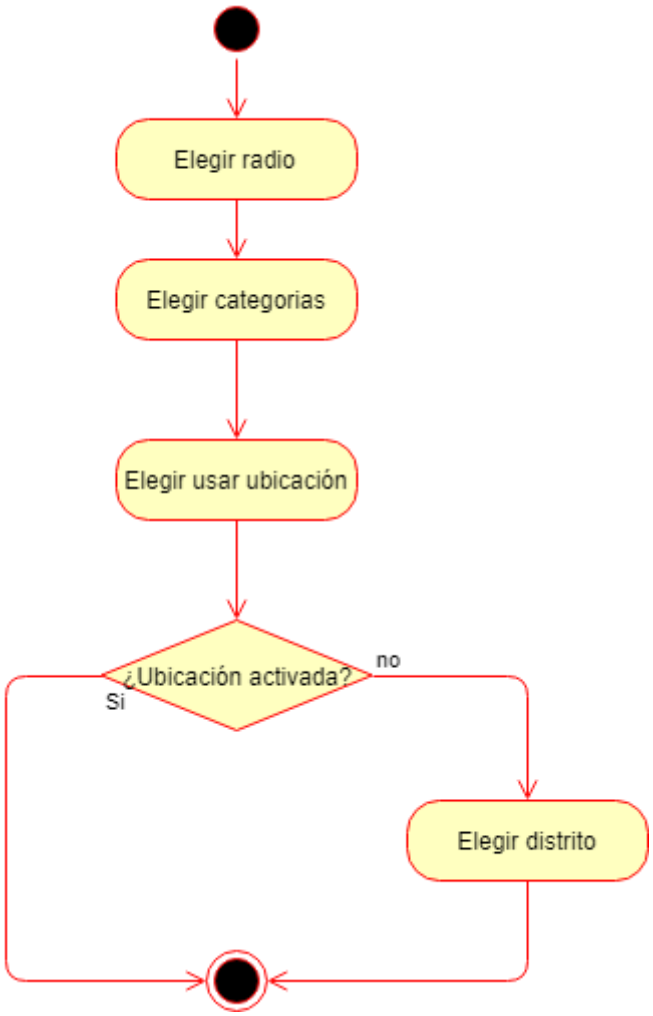
- El **radio** donde quiere que se obtengan alertas. Este radio puede variar entre 0 y 25, cuya unidad de medida es el kilómetro. Este rango de distancias es suficientemente grande como para abarcar los distritos que se quiera desde cualquier punto de la ciudad.

- Las **categorías** de las alertas que quiere ver el usuario. Existirá la posibilidad de seleccionar 'Todas' o tan sólo aquellas que el usuario prefiera. Nunca podrá estar marcada la opción 'Todas' con alguna otra categoría. Del mismo modo, si se seleccionan una por una todas las categorías existentes, automáticamente se rellenará 'Todas'.

- El **distrito** del que quiere ver las alertas. Si se activa la ubicación, el distrito será aquel que marquen las coordenadas. En caso de que la ubicación no estuviera activa, permitirá elegir entre los distritos existentes (sólo uno). Del mismo modo,

existirá la opción ‘Todos’, que abarca los 21 distritos que componen la ciudad de Madrid y ‘General’, que mostrará aquellas alertas que no tienen asignado un distrito en concreto.

En la Figura 2.1.2.5 se muestra el diagrama de actividades correspondiente a este caso de uso.



6- Figura 2.1.2.5 - A-Diagrama activ. - Editar configuración

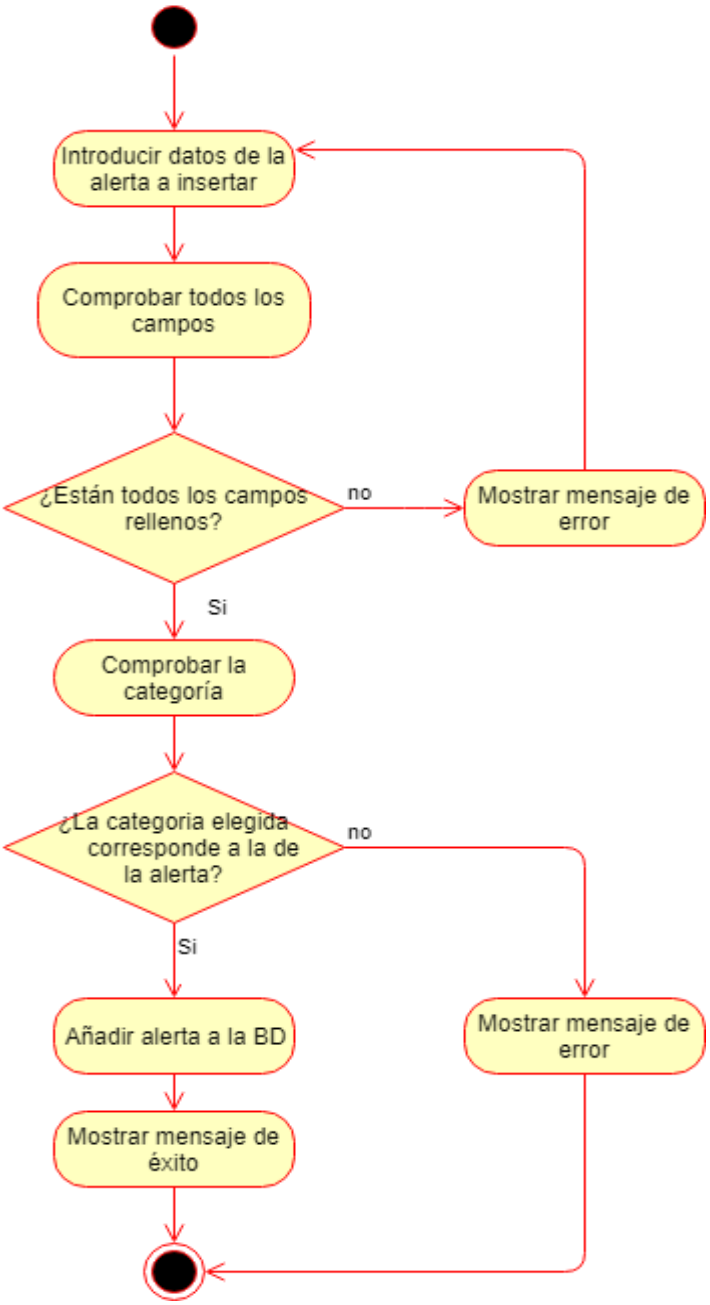
A-6: Añadir alertas

Con esta funcionalidad será posible añadir alertas en tiempo real, a través de un formulario en el que se deberá introducir nombre, email, distrito al que afecta la noticia, categoría correspondiente a la alerta y descripción de la alerta. La alerta se añadirá al pulsar el botón de ‘Añadir’ teniendo en cuenta las restricciones que se detallan a continuación:

- Si se añade la alerta con algún campo en blanco, esta no se añadirá y se indicará qué campo está vacío y se deberá rellenar.

- Se incluirá un filtro de error para evitar guardar en base de datos alertas falsas. Este consistirá en comprobar que la categoría indicada es la adecuada acorde a la descripción.

En la Figura 2.1.2.6 se muestra el diagrama de actividades correspondiente a este caso de uso.

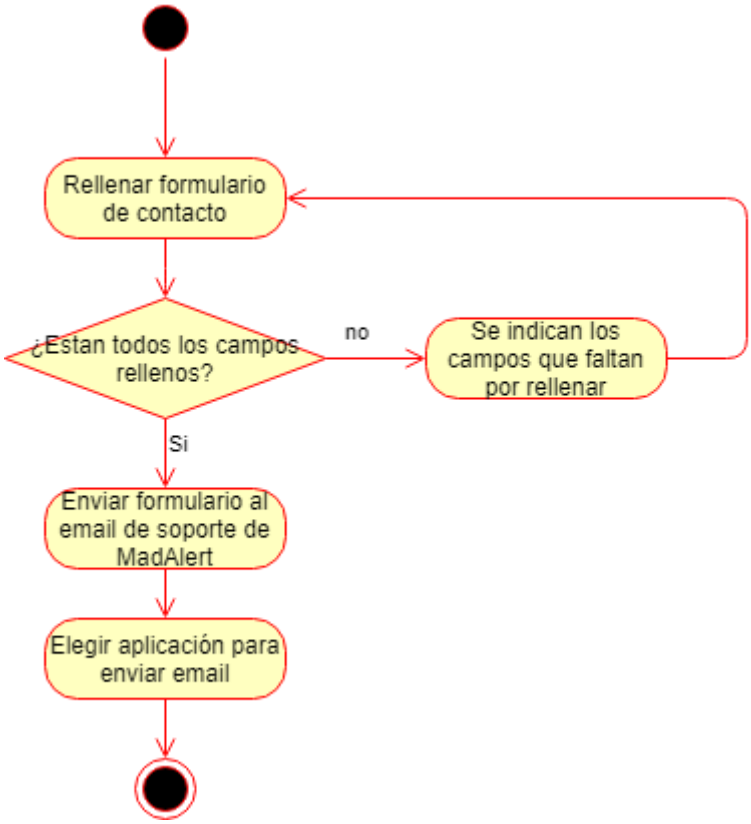


7- Figura 2.1.2.6 - A-Diagrama activ. - Añadir alertas

A-7: Contacto

Se podrá llegar a esta opción a través del menú desplegable lateral. Permitirá enviar un correo a los creadores de la aplicación. Para que pueda ejecutarse se deberán rellenar los siguientes campos de información: nombre, correo del emisor, asunto y mensaje. Tras pulsar a enviar, la aplicación permitirá elegir con qué cliente de correo se desea terminar de enviar el mensaje.

En la Figura 2.1.2.7 se muestra el diagrama de actividades correspondiente a este caso de uso.

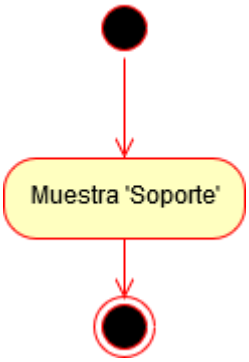


8- Figura 2.1.2.7 - A-Diagrama activ. - Contacto

A-8: Obtener información acerca de la aplicación. ‘Soporte’

Esta opción también estará presente en el menú desplegable lateral. Se mostrará la información legal de la aplicación y sus términos de uso.

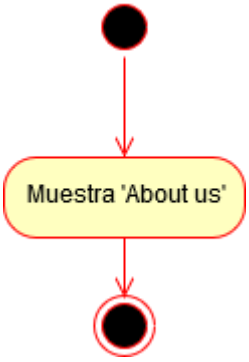
En la Figura 2.1.2.8 se muestra el diagrama de actividades correspondiente a este caso de uso.



9- Figura 2.1.2.8 - A-Diagrama activ. - Soporte

A-9: Obtener información sobre nosotros. ‘About us’

Al igual que las anteriores funcionalidades, a esta opción también se podrá llegar desde el menú desplegable lateral y mostrará la información acerca de los creadores de la aplicación de una manera breve y concisa. En la Figura 2.1.2.9 se muestra el diagrama de actividades correspondiente a este caso de uso.

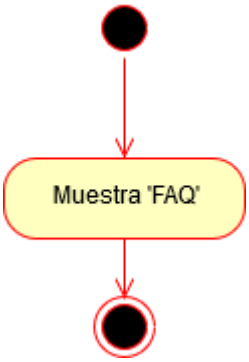


10- Figura 2.1.2.9 - A-Diagrama activ. - About us

A-10: Resolver dudas acerca de la aplicación. ‘FAQ’

Está opción estará accesible desde el menú lateral desplegable. En este apartado se recogerán las preguntas más frecuentes que podrán surgirle al usuario durante el uso de la aplicación.

En la Figura 2.1.2.10 se muestra el diagrama de actividades correspondiente a este caso de uso.



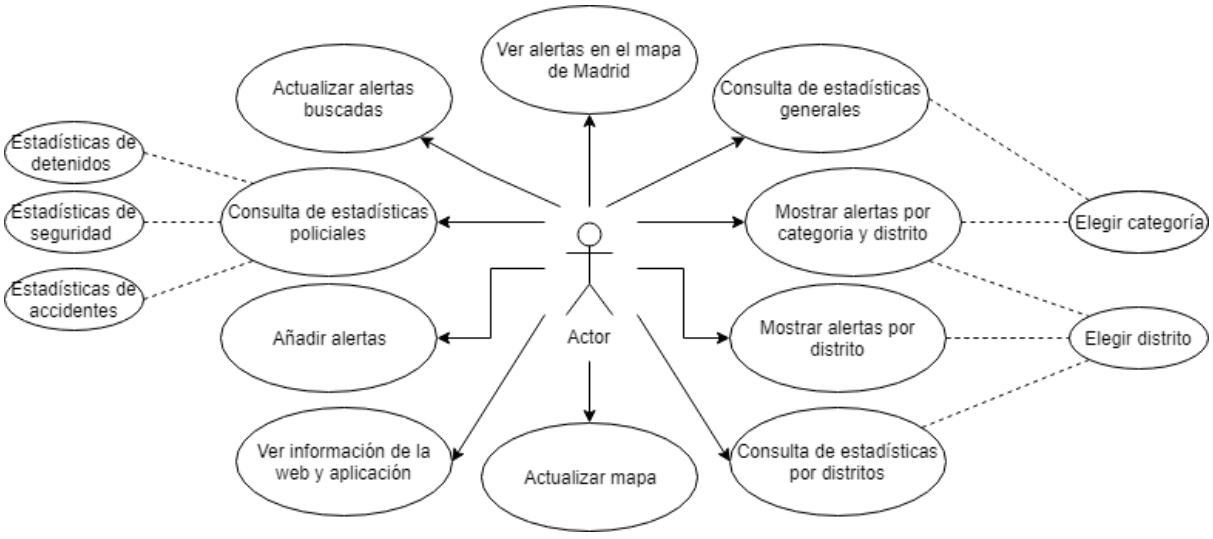
11- Figura 2.1.2.10 - A-Diagrama activ. – FAQ

2.2. FUNCIONALIDAD DE LA APLICACIÓN WEB

La funcionalidad principal de la aplicación web será mostrar en tiempo real las alertas de los distritos de Madrid. Se podrá interactuar con la web, de modo que se podrán visualizar las alertas de un distrito específico de las categorías deseadas, añadir alertas y consultar las estadísticas generales y por distritos de los últimos meses. Además, la web contará con unas estadísticas policiales obtenidas de la Policía Municipal de Madrid[23].

2.2.1 DIAGRAMA DE CASOS DE USO

En la Figura 2.2.1.1 se muestra el diagrama con los casos de uso posibles para el usuario de la aplicación web.



12- Figura 2.2.1.1 - Diagrama Casos de uso Web

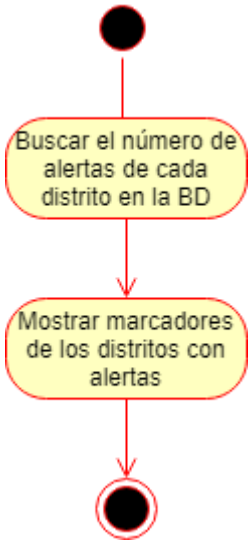
2.2.2 DIAGRAMAS DE ACTIVIDADES

En esta sección se describen con detalle las funcionalidades de la aplicación web junto con sus correspondientes diagramas de actividades.

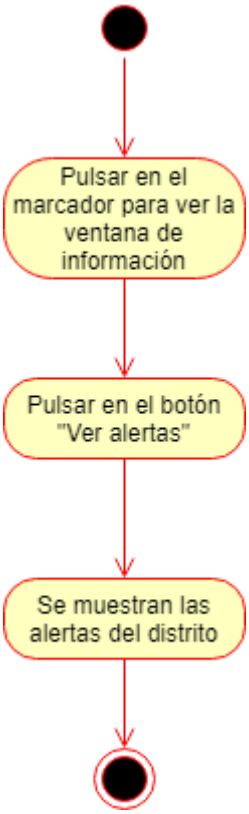
W-1: Ver alertas en el mapa en tiempo real en Madrid.

En la página principal se podrán visualizar los distritos de Madrid que tienen alertas en las últimas horas. Es una información en tiempo real y los distritos que tengan alertas se mostrarán en el mapa con su marcador. Cada marcador contendrá tanto el nombre del distrito como el número de alertas. Si se quiere ver en profundidad las alertas, cuenta con una opción de “ver alertas”, en la que se mostrarán las alertas de ese distrito ordenadas por fecha y de forma detallada.

En la Figura 2.2.2.1 y 2.2.2.2 se muestra el diagrama de actividades correspondiente a este caso de uso.



13- Figura 2.2.2.1 - W-Diagrama activ. - Ver alertas mapa



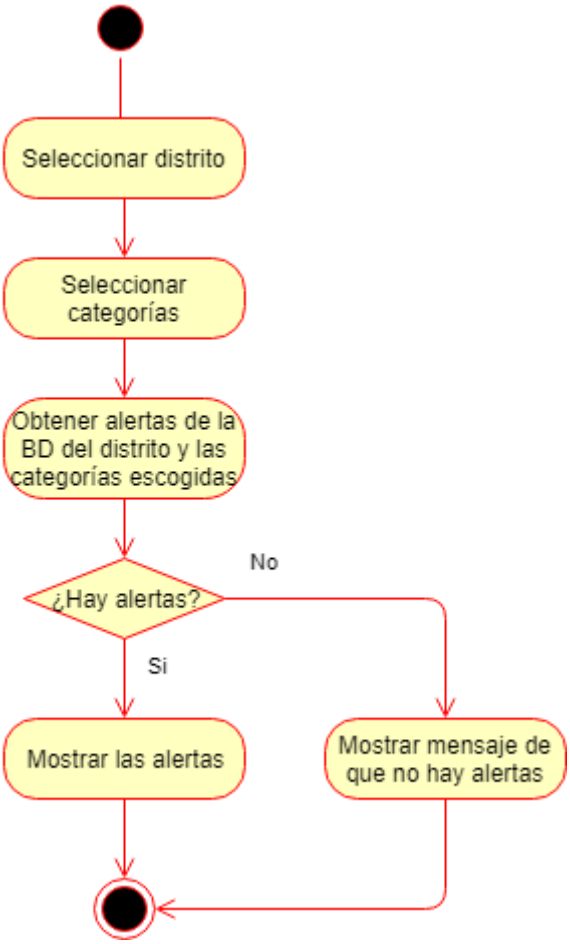
14- Figura 2.2.2.2 - W-Diagrama activ. - Ver alertas marcador mapa

W-2: Mostrar alertas por distrito y categoría

Para llevar a cabo esta funcionalidad correctamente se deberá seleccionar un distrito y al menos una categoría para que la búsqueda de alertas por distrito y categoría pueda realizarse. Al realizarse la búsqueda se mostrarán las alertas del distrito y de las categorías seleccionadas, si ese distrito no dispone de alertas o las alertas que tiene no corresponden a las categorías seleccionadas se indicará al usuario.

En caso de que se intentará hacer la búsqueda sin cumplir las condiciones anteriores, se mostrará un mensaje de error.

En la Figura 2.2.2.3 se muestra el diagrama de actividades correspondiente a este caso de uso.

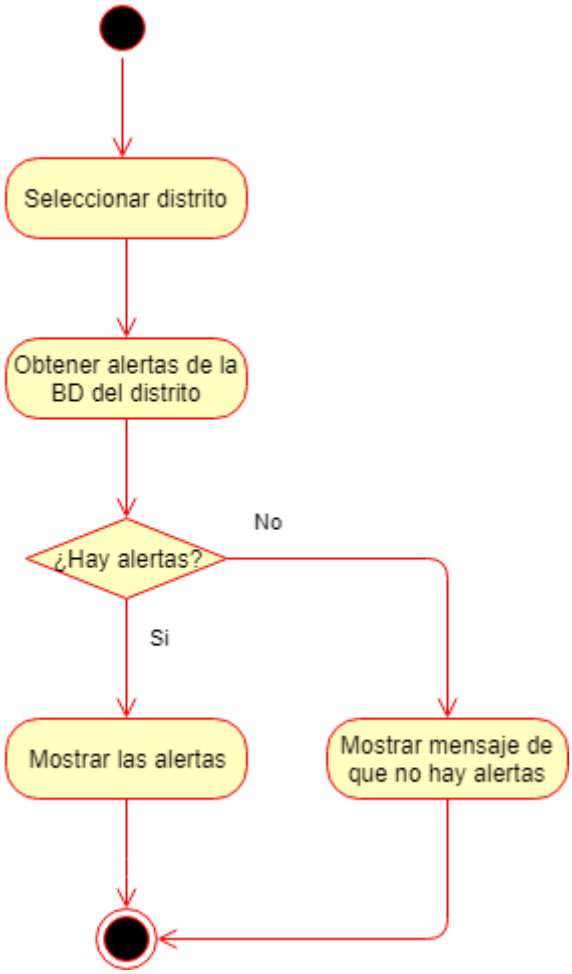


15- Figura 2.2.2.3 - W-Diagrama activ. - Mostrar alertas por distrito y categoría

W-3: Mostrar alertas por distrito

Similar a la funcionalidad anterior pero sólo se seleccionará el distrito, donde se mostrarán todas sus alertas independientemente de las categorías que tengan.
Si el distrito seleccionado no dispone de alertas se mostrará un mensaje indicándolo.

En la Figura 2.2.2.4 se muestra el diagrama de actividades correspondiente a este caso de uso.

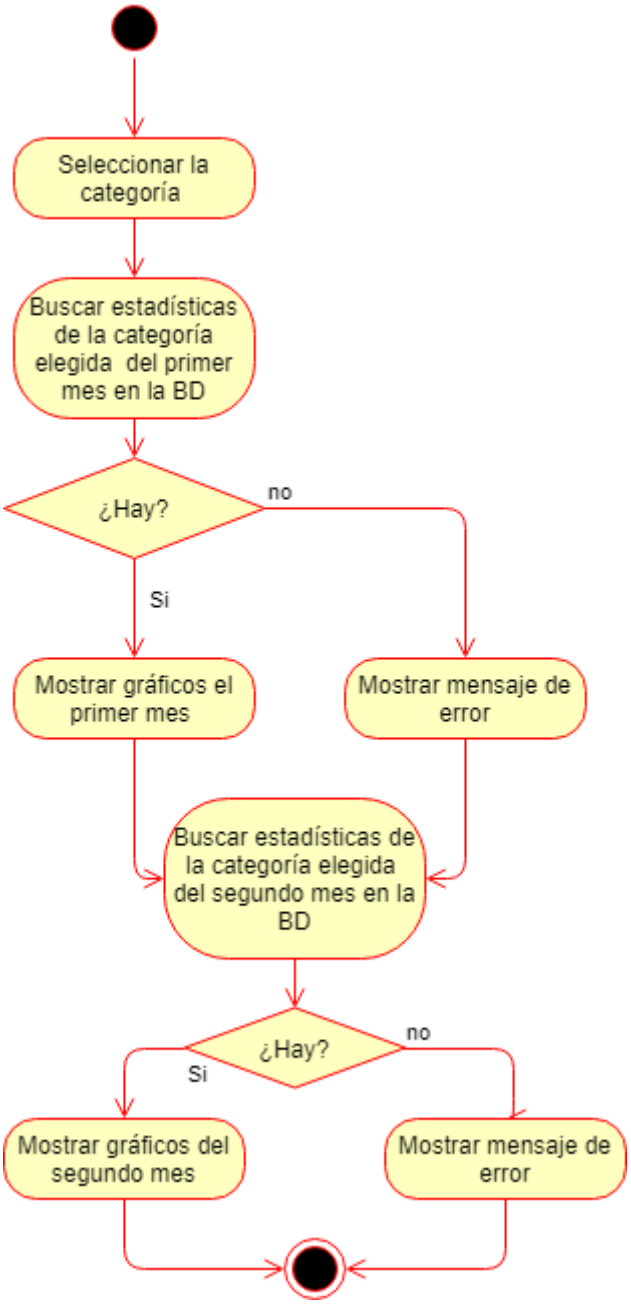


16- Figura 2.2.2.4 - W-Diagrama activ. - Mostrar alertas por distrito

W-4: Consulta de estadísticas generales

En esta sección se visualizarán las estadísticas generales según la categoría seleccionada por el usuario. Si la categoría que escoge el usuario no tiene datos registrados de esos meses se mostrará un mensaje informativo acerca de ello.

En la Figura 2.2.2.5 se muestra el diagrama de actividades correspondiente a este caso de uso.

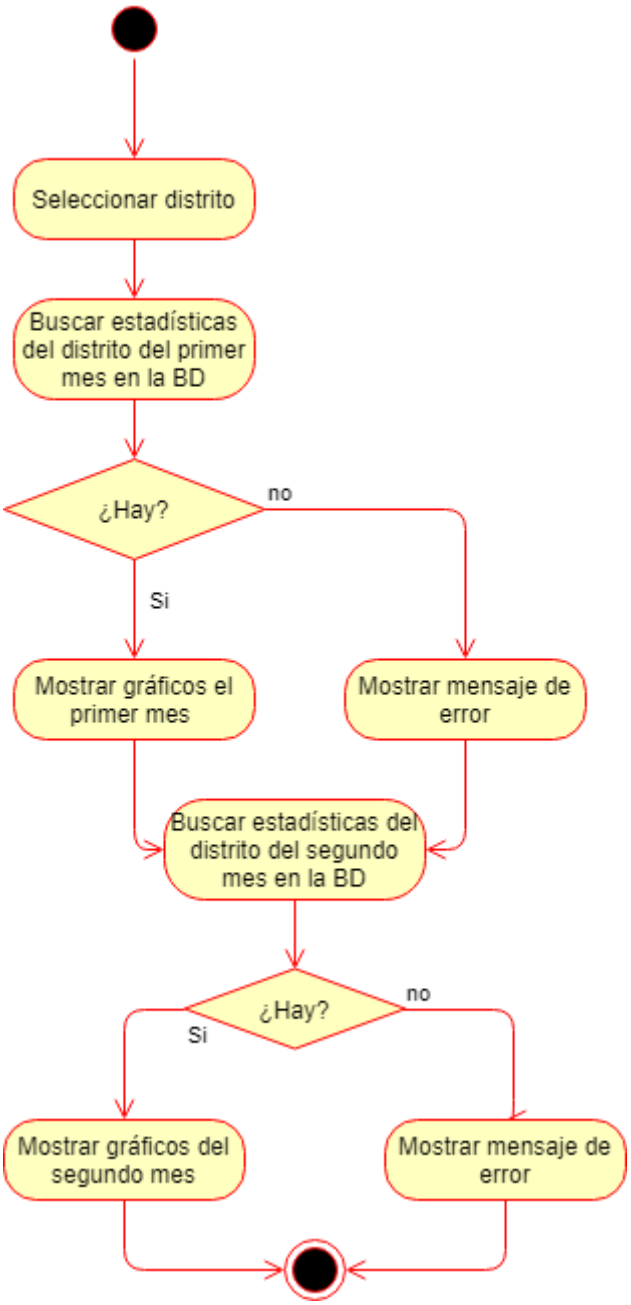


17- Figura 2.2.2.5 - W-Diagrama activ. - Consulta estadísticas generales

W-5: Consulta de estadísticas por distritos

En esta funcionalidad se realizará la consulta de estadísticas del distrito que se desee. Si el distrito seleccionado no tiene estadísticas se mostrará un mensaje informativo acerca de ello. Si la categoría escogida por el usuario no tiene datos registrados de esos meses se mostrará un mensaje informativo acerca de ello:

En la Figura 2.2.2.6 se muestra el diagrama de actividades correspondiente a este caso de uso.



18- Figura 2.2.2.6 - W- Diagrama activ. - Consulta estadísticas por distritos

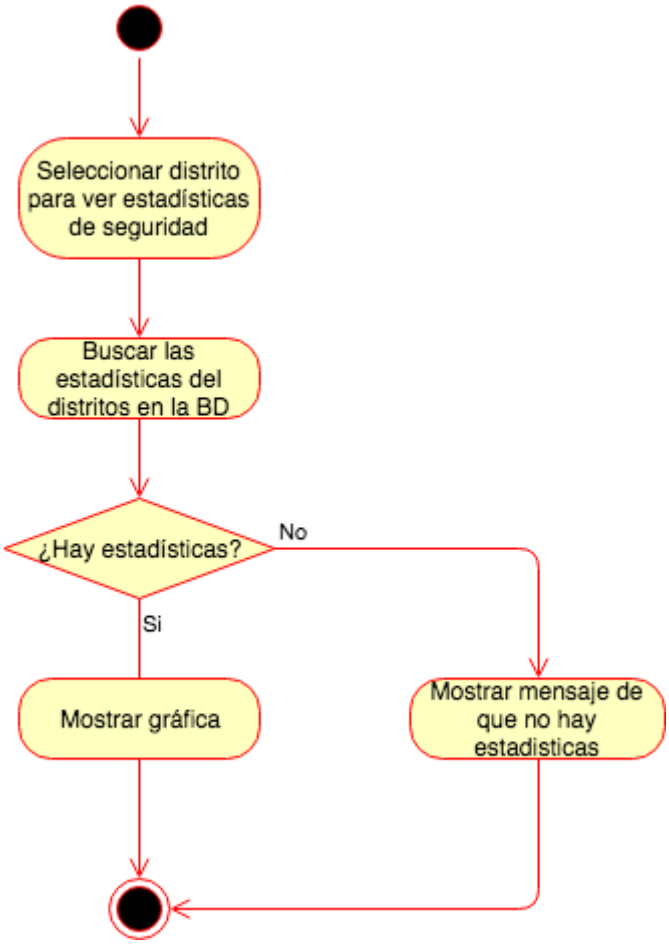
W-6: Consulta de estadísticas de la Policía Municipal.[23]

En esta funcionalidad se podrán consultar estadísticas relacionadas con la seguridad de los distritos [Figura 2.2.2.7], los accidentes que hay por distrito y los detenidos por distrito [Figura 2.2.2.8].

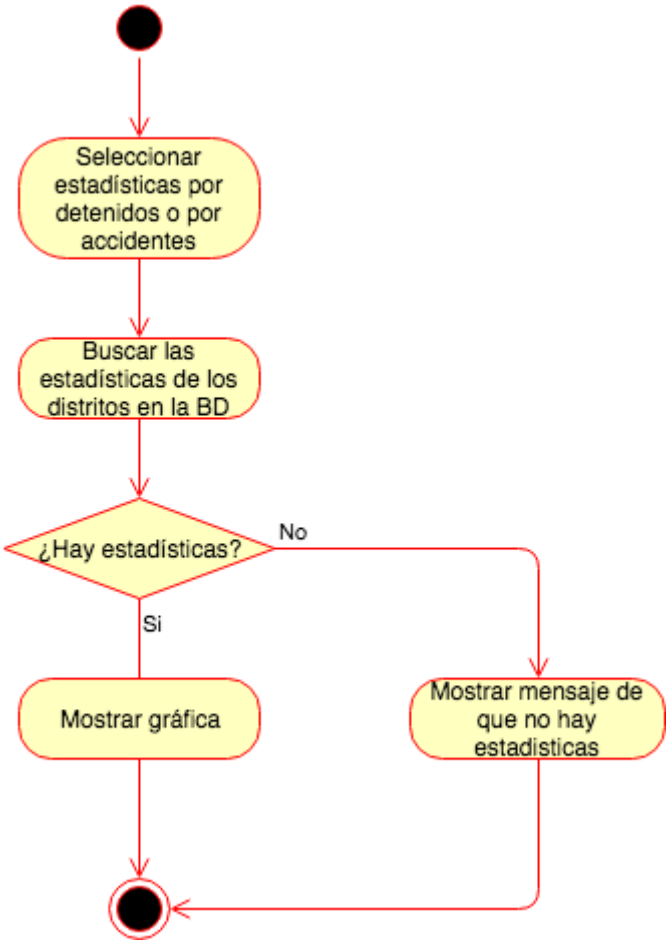
Al seleccionar cualquiera de las opciones se mostrarán las estadísticas correspondientes, pero en la opción de estadísticas relacionadas con la seguridad habrá que seleccionar además un distrito.

Si alguna opción no dispone de estadísticas aparecerá un mensaje informativo.

En la Figura 2.2.2.7 y 2.2.2.8 se muestra el diagrama de actividades correspondiente a este caso de uso.



19- Figura 2.2.2.7 - W-Diagrama activ. - Consulta estadísticas Policía Municipal Madrid



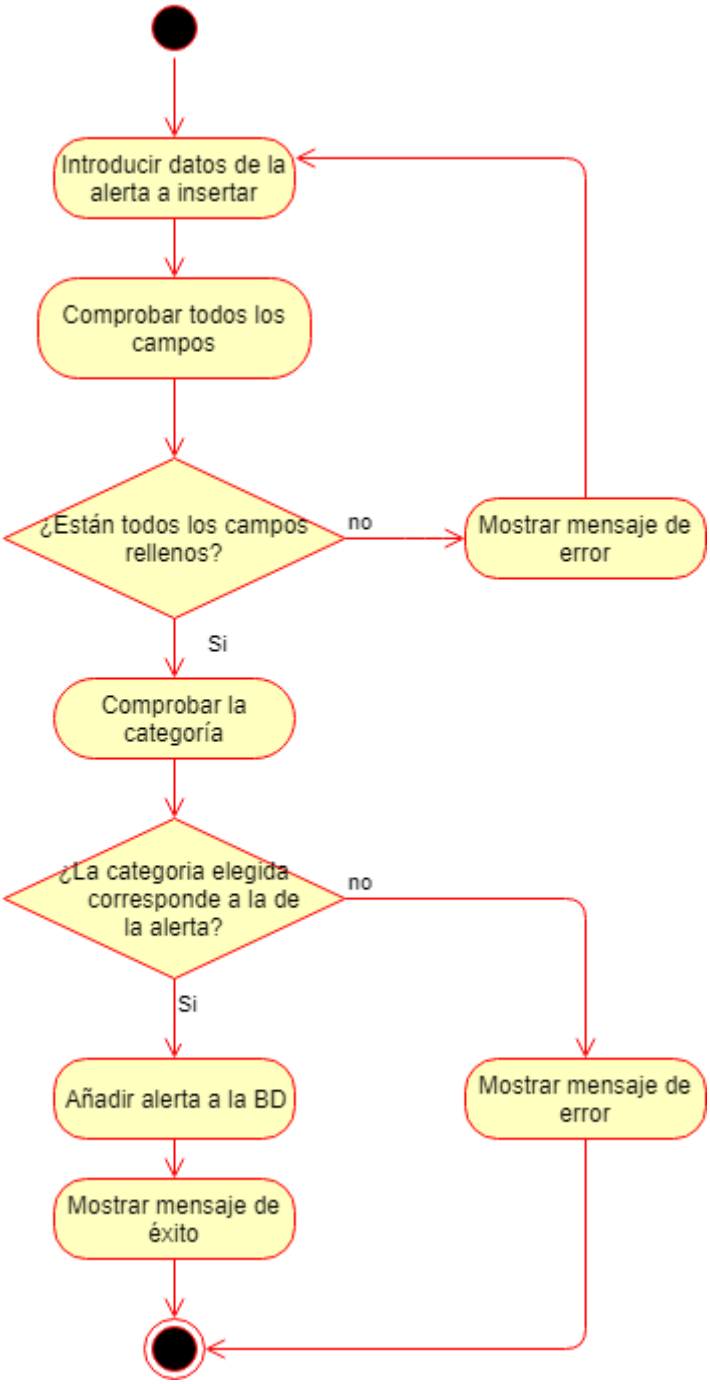
20- Figura 2.2.2.8 - W-Diagrama activ. - Estadísticas accidentes / detenidos

W-7: Añadir alertas

Otra de las funcionalidades de la Web es el alta de una alerta en tiempo real. Los usuarios tendrán que rellenar un formulario con sus datos de identificación, así como la alerta, la categoría y el distrito. Todos los campos tendrán que estar completos para que se pudiera añadir la alerta.

Para que esta funcionalidad tenga más credibilidad, se comprobará que la categoría indicada por el usuario es la correcta. En caso de que no sea correcta, se mostrará un mensaje de error y no se añadirá la alerta.

En la Figura 2.2.2.9 se muestra el diagrama de actividades correspondiente a este caso de uso.

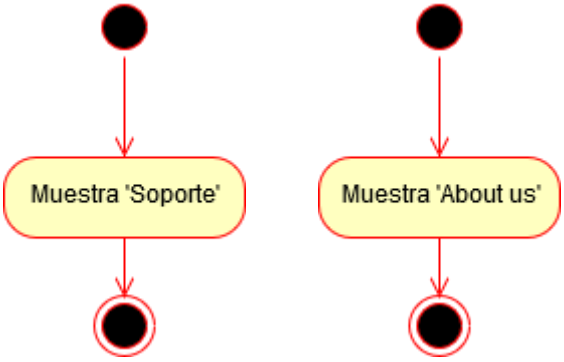


21- Figura 2.2.2.9 - W-Diagrama activ. - Añadir alertas

W-8: Información de la Web

En esta funcionalidad aparecerán las opciones de “Soporte” y “About us”. En la primera se podrá ver la información legal de la aplicación y sus términos de uso. En la otra pestaña se visualizará una breve información sobre los integrantes del grupo y cuenta con un formulario para enviar cualquier duda o sugerencia sobre la aplicación web.

En la Figura 2.2.2.10 se muestra el diagrama de actividades correspondiente a este caso de uso.

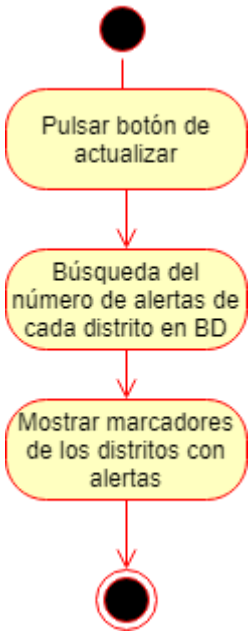


22- Figura 2.2.2.10 - W-Diagrama activ. - Info de la web

W-9: Actualizar mapa

La actualización del mapa consistirá en que aparezcan los marcadores antiguos y nuevos de los distritos que tengan alertas y el número de alertas actual de cada distrito.

En la Figura 2.2.2.11 se muestra el diagrama de actividades correspondiente a este caso de uso.

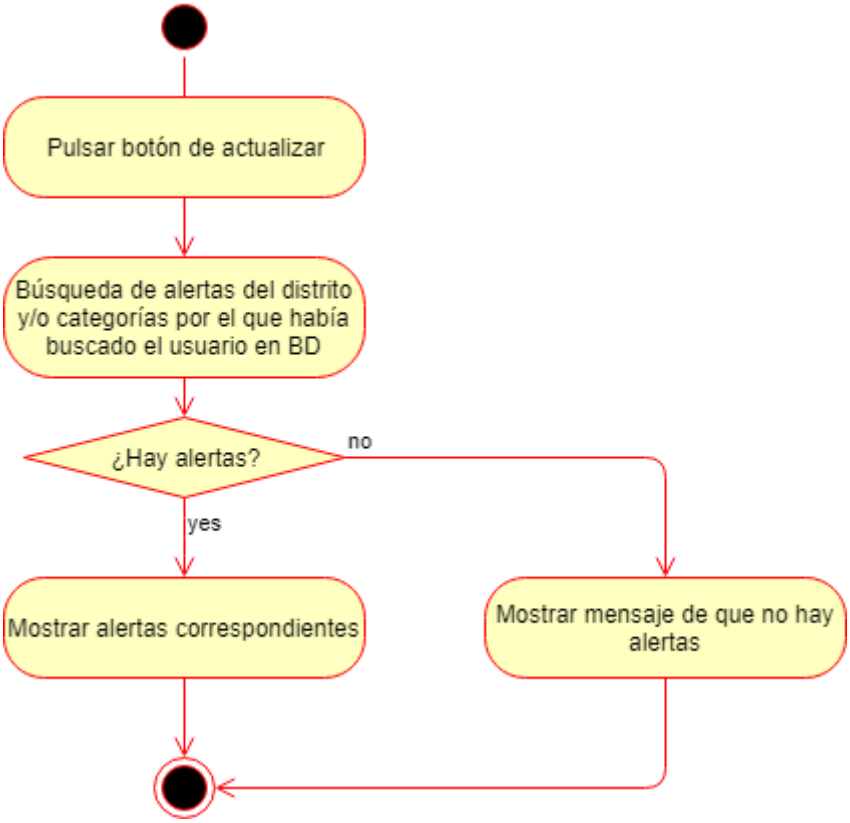


23- Figura 2.2.2.11 - W-Diagrama activ. - Actualizar mapa

W-10: Actualizar alertas por distritos y/o categoría

Esta funcionalidad actualizará la búsqueda de alertas por distrito y/o categoría elegido por el usuario. Al actualizar se mostrarán tanto las nuevas alertas como las anteriores con los filtros elegidos anteriormente.

En la Figura 2.2.2.12 se muestra el diagrama de actividades correspondiente a este caso de uso.



24- Figura 2.2.2.12 - W-Diagrama activ. - Actualizar por distritos y/o categoría

2.3 FUNCIONALIDAD API Y RECOLECTOR DE DATOS SOCIALES

2.3.1 FUNCIONALIDAD API

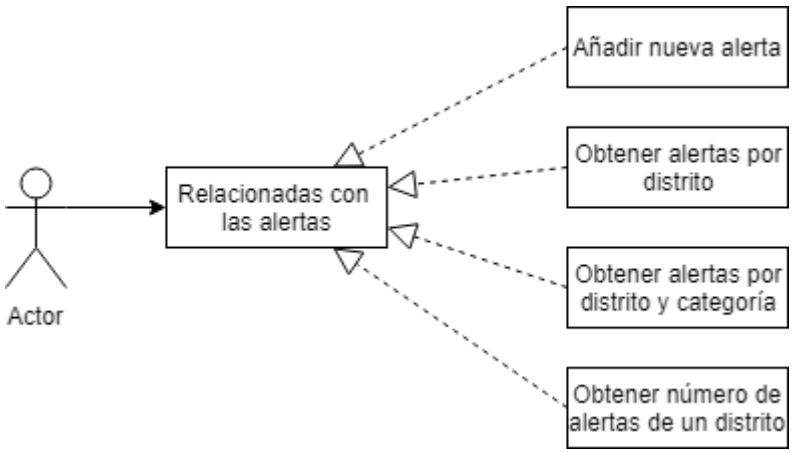
Una **API**[101] (siglas de 'Application Programming Interface') es un conjunto de reglas y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas: sirviendo de interfaz entre programas diferentes de la misma manera en que la interfaz de usuario facilita la interacción humano-software.

La API que se usará en este proyecto para la aplicación Android sirve para interactuar de forma abstracta con la base de datos. Esta API permitirá la obtención de alertas por distrito y/o categoría, así como la cantidad que hay por cada distrito y añadir nuevas alertas.

La funcionalidad que ofrecerá la API de Android es la siguiente:

1. **Obtener alertas por distritos**: se obtendrán las alertas del distrito indicado.
2. **Obtener alertas por distritos y categorías**: se obtendrán las alertas por los distritos y categorías que sean indicados.
3. **Obtener número de alertas de un distrito**: se obtendrán el número de alertas que haya actualmente en un distrito.
4. **Añadir una nueva alerta**: dado los datos de un usuario, la descripción de la alerta, la categoría y el distrito, se añadirá la alerta a la base de datos.

En la Figura 2.3.1.1 se muestra el diagrama de casos de usos correspondiente a la API.



25- Figura 2.3.1.1 - API

2.3.2 RECOLECTOR DE DATOS

Tanto nuestra aplicación móvil como web se va a alimentar de una base de datos que se va a ir rellenando a través de los scripts de Python[88].

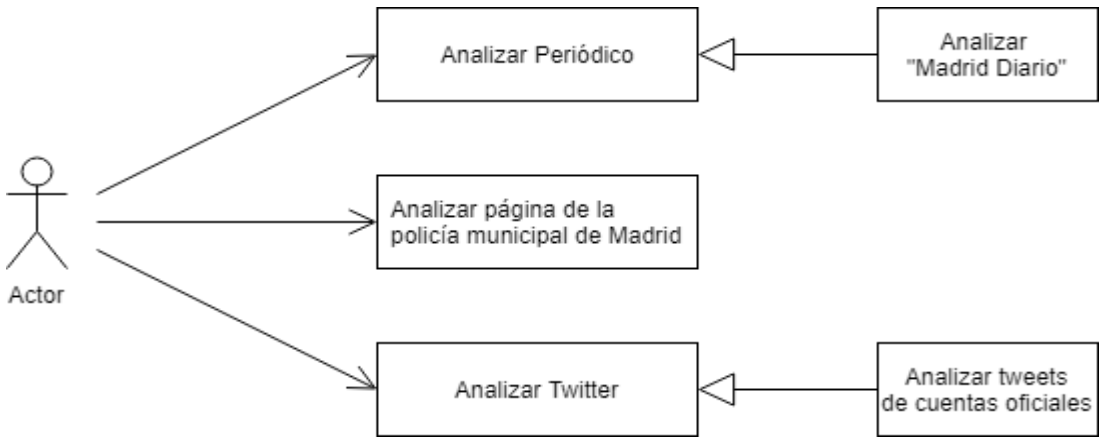
Para poder alimentar esta base de datos, se van a utilizar diversas fuentes. Se pensó que para estar totalmente actualizado y en tiempo real de todas las noticias emergentes, lo mejor es hacer uso de Twitter y de algún periódico de la ciudad de Madrid, en nuestro caso, Madridiario[58].

Para la obtención de los datos procedentes de Twitter se hará uso de la API de twitter, la cual es Tweepy[98][102]. Para obtener las alertas en tiempo real[22][25][79], se tendrá un programa en continuo funcionamiento y cuando se reciba un Tweet, se analizará con el propósito de añadirlo en nuestra base de datos si encaja con las características predefinidas.

Por su parte, para la extracción de datos del periódico Madridiario[58] se empleará una herramienta denominada web scraping[60], que consiste en una técnica que sirve para extraer información de páginas web de forma automatizada. Existe un programa que se ejecutará cada 5 minutos con el objetivo de obtener todas las noticias en tiempo real. Este programa incluirá una condición para evitar que las mismas alertas se almacenen de forma reiterada: si la hora publicación de la alerta es inferior a 5 minutos atrás, esta se descartará.

Del mismo modo, existirán varios programas que cumplirán la función de obtener los datos procedente de la página oficial de estadísticas de la Policía Municipal de Madrid[23]: uno para descargar el Excel[16] y otro para analizar ese documento e insertar las estadísticas recogidas en la base de datos.

En la Figura 2.3.2.1 se muestra el diagrama de casos de usos del recolector de datos.



26- Figura 2.3.2.1 - Recolector datos

3. TECNOLOGÍAS EMPLEADAS

En esta sección se van a detallar las tecnologías empleadas en cada parte del proyecto.

Para poder programar simultáneamente tanto la aplicación web como la móvil, se ha decidido utilizar **Git**[34] como sistema de control de versiones. Git[34] es un sistema de control de versiones distribuidas de código abierto y gratuito diseñado para gestionar proyectos de cualquier tamaño con velocidad y eficiencia.

Se ha utilizado la aplicación de escritorio "**GitHub Desktop**"[35] muy útil para el trabajo en equipo ya que se trata de una plataforma de desarrollo colaborativo de software donde se pueden alojar proyectos utilizando el sistema de control de versiones Git. Es por esto por lo que se ha decidido trabajar con ello.

Como editor de código se ha utilizado **"Sublime Text 3.0"**[100]**.** Se trata de un editor de texto multiplataforma sencillo de utilizar, muy visual, intuitivo y fácil de leer gracias a la variedad de colores con los que se resaltan las expresiones propias del lenguaje en el que se está programando.

Por otro lado para implementar la gestión de las alertas (bien de Twitter, bien de Madridiario[58] o bien de la Policía Municipal[23]) se ha utilizado **Python 3.6**[89] y como entorno de desarrollo "**IDLE**"[88] (Python's Integrated Development and Learning Environment), en concreto la versión 3.6.3. Python[89] es un lenguaje de programación que permite trabajar más rápido e integrar sus sistemas de manera más efectiva. Se ha decidido trabajar con él porque todos los integrantes del equipo teníamos conocimientos previos acerca de este lenguaje y su entorno. Además, Python[89] tiene una curva de aprendizaje corta y mucha documentación sobre lo que queríamos llevar a cabo.

3.1 APLICACIÓN ANDROID

Para el desarrollo de la aplicación móvil, se ha elegido utilizar el IDE **Android Studio**[9], debido a que se puede ejecutar en tiempo real la aplicación en un emulador y en nuestro propio móvil**.** Se trata de un potente entorno de desarrollo integrado oficial para la implementación de aplicaciones Android. Su lenguaje de programación incluye Java para el código fuente y XML[104] para los recursos tales como diseños, cadenas e imágenes, entre otros. Para su aprendizaje, se ha utilizado como principal fuente la documentación oficial que ofrece Android[5][6], y diversas páginas web señaladas en la bibliografía de este documento.

Para las consultas que realiza la aplicación móvil con MongoDB[68] ,se utilizan los servicios que ofrece **Node.js**[76]**.**Node.js[76] usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. La elección de Node.js[76] se debe a

la gran cantidad de información que se encontró para realizar la conexión a MongoDB desde Android[5][6] con este servicio frente a otras opciones. En este proyecto, se está utilizando como una API RESTFul[13], a partir de unos parámetros se realiza una consulta (request) y se crea una respuesta (response) en formato json.

Para depurar las funciones creadas en NodeJS[76] se ha utilizado la herramienta PostMan[85], que permite el envío de peticiones GET y POST. Esta herramienta ha sido de gran utilidad a la hora de corregir fallos y ver que las respuestas que se esperaban eran las correctas.

Del mismo modo, se ha hecho uso de **Retrofit**[93], un cliente REST[10] para Android y Java que actúa de nexo entre la aplicación Android y Node.js[76] a la hora de realizar consultas a la base datos, permite hacer peticiones get y post, entre otras, que sirven para obtener datos en formato json y añadir datos a la base de datos.

3.2 APLICACIÓN WEB

Para el desarrollo de la aplicación web, se han utilizado diversos lenguajes como **HTML5[103], CSS3[51], JavaScript[55] y PHP[82].**

Se ha utilizado JavaScript**[55]** para el control de errores, la generación de gráficas de estadísticas y el mapa.

Mediante PHP**[82]** se ha conectado la BD con la aplicación web y realizado diferentes formularios, entre otras cosas.

Además, se ha utilizado el Framework **Bootstrap[15]**, debido a su sencillez y su facilidad a la hora de realizar un diseño responsive (permitiendo así la visualización de la aplicación en distintos dispositivos).

3.3 BASE DE DATOS

Hoy en día cada vez se está extendiendo más y más el uso de las bases de datos no relacionales. Como se ve en la asignatura optativa “Bases de datos NoSQL”[18] , a la hora de tratar una gran cantidad de datos y si éstos pueden ser variables, se recomienda la utilización de este tipo de bases de datos. Toda la información que se va a mencionar a continuación no es más que un resumen de lo que el profesor Rafael Caballero cuenta en su página[18]:

El problema de las bases de datos relacionales llegó con la creación de un nuevo concepto conocido como Big Data. Se trata de un concepto que se suele asociar directamente con el de bases de datos NoSQL (Not Only SQL).

En los tiempos que corren, un teléfono móvil puede llegar a generar incluso más datos por minuto que el propio censo de un país (contando sus datos de localización, llamadas, mensajes, etc.). Estos datos interesa almacenarlos y tratarlos, ya que son una valiosa fuente de información.

Por esa razón, en el momento en el que algunas grandes compañías como es Google se encontraron con el problema de poder indexar grandes cantidades de datos y que además eran modificados continuamente. Es entonces cuando las bases de datos relacionales se les quedaron muy cortas. Inclusive, Twitter se encuentra por ejemplo con alrededor de unos 500 millones de tweets nuevos cada día (siendo además mensajes no homogéneos). Éste es el verdadero mundo de Big Data.

Se puede definir Big Data mediante las 3 V's, definidas por Doug Laney[56]:

1. **Volumen**: Grandes cantidades de datos.

2. **Variedad**: Son datos no necesariamente homogéneos.

3. **Velocidad**: La velocidad se refiere a la velocidad de llegada de estos datos.

Por consiguiente, en este proyecto al tener una gran cantidad de alertas y estadísticas no siempre homogéneas, se decidió utilizar este tipo de base de datos. En concreto, se ha utilizado la base de datos no relacional llamada **MongoDB** [68]**.** Ésta se encarga de guardar estructuras de datos, documentos, en vez de datos en registros como lo hace una base de datos relacional. Estos documentos son almacenados en formato BSON (Binary JSON), una representación binaria de JSON[105], haciendo que la integración de los datos sea fácil y rápida.

Para gestionar MongoDB, al principio del proyecto se utilizó el entorno de desarrollo gráfico denominado **“Robo 3T”**[96] (anteriormente conocida como “Robomongo”) que permite consultar, modificar o eliminar datos. Posteriormente fue necesario trasladar la base de datos en la nube.

Las principales razones por las que hemos decidido utilizar una base de datos no relacional son su optimización a la hora de realizar consultas con bases de datos que tienen grandes cantidades de datos, la posible flexibilidad de sus datos y a la naturaleza no estructurada de los datos que utilizamos

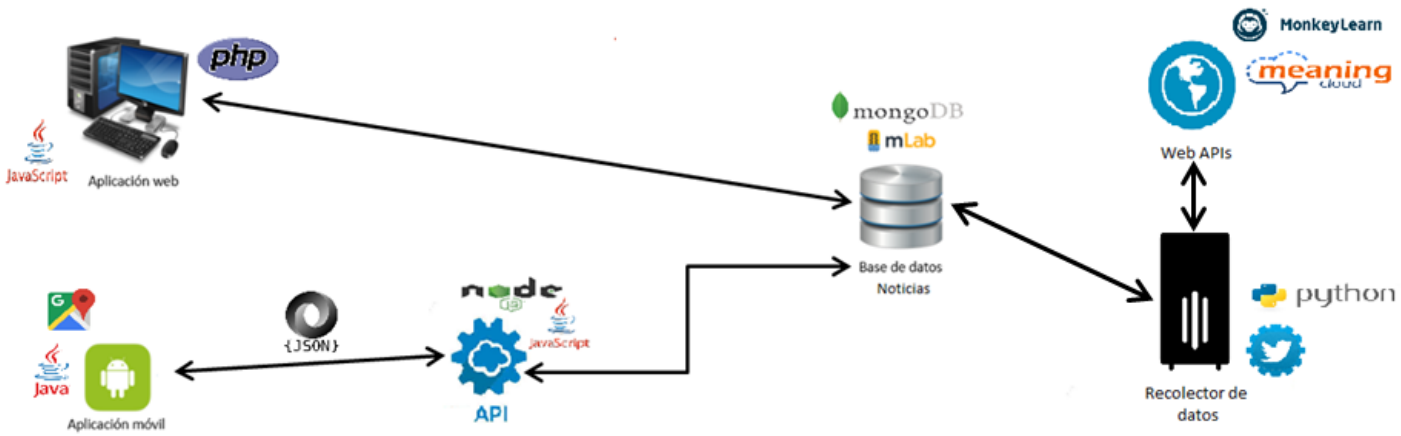
Comparando varias opciones finalmente se llegó a la conclusión de que era mejor utilizar un servicio llamado “**mLab”**[64] que permite alojar de una forma sencilla y sin cargos adicionales bases de datos MongoDB en la nube.

El único imprevisto que se encontró a la hora de utilizar este servicio para la base de datos fue el inconveniente de no poder utilizarlo en la Facultad de Informática. El wifi lo tiene censurado y por tanto es necesario utilizar otro método de acceso a internet.

4. ARQUITECTURA DE LA APLICACIÓN

El sistema se ha diseñado siguiendo una arquitectura cliente-servidor donde la parte **cliente** está formada por la aplicación web y la aplicación móvil Android. El servidor se encarga de dar servicio a los clientes y de gestionar una base de datos no relacional compartida por ambos clientes que almacena todos los datos que son recogidos a través del recolector de datos. En nuestro sistema el propio ordenador es el que actúa como servidor.

En el siguiente esquema de la Figura 4.1 se pueden observar los componentes de la arquitectura comentada. Una característica de esta arquitectura es el uso de la base de datos no relacional como elemento de comunicación entre el cliente y el servidor, de manera que cualquier actualización que se realiza en la base de datos, se muestra en ambos tipos de clientes.



27- Figura 4.1 - Esquema arquitectura

A continuación, se van a explicar brevemente los elementos de la arquitectura.

Aplicación para dispositivos **Android** a partir de la versión 4.1, por lo que es compatible con casi un 100% de los dispositivos Android existentes en el mercado [21]. De este modo, el objetivo la aplicación es convertirse en la fuente principal de las alertas de Madrid. Gracias al mapa integrado, se pretende facilitar el acceso a esta información a los usuarios.

Para poder comunicar la aplicación Android con la base de datos, se cuenta con una API[101] que es la encargada de enviar y recibir los datos correspondientes según las peticiones del usuario Android.

En la **aplicación web** será donde el usuario podrá consultar tanto las alertas de la comunidad de Madrid como las estadísticas obtenidas a partir de ellas. Además, a través de la web ofrecemos la posibilidad de añadir nuevas alertas (las cuales aparecerán como no verificadas), leer información sobre nosotros y sobre la página (política de privacidad, términos y condiciones y opción de ponerse en contacto con nosotros).

Toda la información visualizada en la web se obtiene de la **base de datos** de la que se hablará a continuación con más detalle en el siguiente apartado. Quien alimenta a la base de datos es el recolector de datos a través de varios scripts generados en Python.

Como se explica más adelante en detalle en el punto 5 de esta memoria, la base de datos que se ha usado se encuentra alojada en un servicio en la nube llamado "***mLab***"[65] y se va a encargar de almacenar toda la información relativa a las alertas de Madrid y a las estadísticas relacionadas con dichas alertas.

Tanto en la aplicación web como en la aplicación móvil, será necesario realizar diferentes consultas a esta base de datos para poder mostrar la información de las diversas alertas, así como las estadísticas que están disponibles.

El proyecto no cuenta con un servidor en la nube como tal, por lo que para las pruebas realizadas se ha usado un ordenador de cualquiera de los miembros del equipo que actúa como servidor web. Cabe remarcar que para el funcionamiento de la aplicación Android es necesario que el ordenador y el smartphone estén conectados a la misma red. No obstante, todo el proyecto está preparado para poder alojarse en cualquier servidor en la nube, algo que podría plantearse en un futuro.

5. MODELO DE DATOS

En esta sección se describe el modelo de datos utilizado. Como ya se ha comentado, se ha empleado como sistema de persistencia MongoDB[68], una base de datos NoSQL de tipo documental. Para implementar el proyecto ha sido necesario definir una base de datos denominado “Noticias” que incluye 5 colecciones diferentes (Figura 5.1).

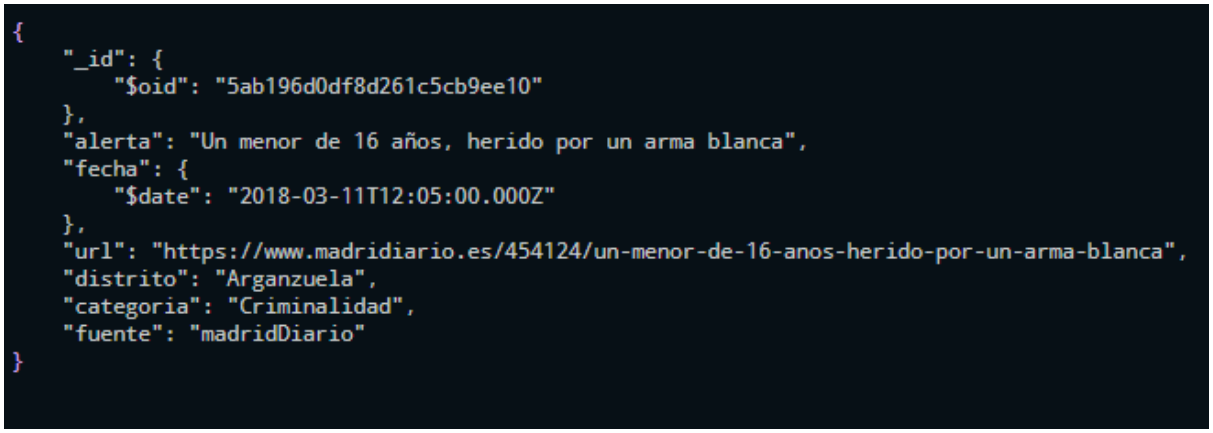
28- Figura 5.1 - Colecciones de la bd

En la colección ***alertas*** se almacenan todas las alertas recogidas de la red social Twitter y del periódico Madridiario[58], así como las alertas insertadas por lo usuarios desde la aplicación web y móvil.

En cada uno de los documentos (Figura 5.2, 5.3) de esta colección se tienen los siguientes campos:

- **_id:** id generado automáticamente por mongo al insertar un documento.
- **alerta:** descripción de la alerta obtenida de twitter, del periódico “Madrid diario” o de los usuarios de la aplicación.
- **fecha:** fecha en la que ha sucedido la alerta, viene indicado tanto el día, mes y año como la hora a la que se ha producido.
- **url:** dirección donde se encuentra la alerta referenciada con más información.
- **distrito:** distrito de Madrid donde ha sucedido la alerta correspondiente. Si es una alerta general de Madrid, este campo tomará el valor de general.
- **categoría:** categoría a la que corresponde la alerta. Puede tomar el valor de Criminalidad, Desastres y accidentes, Contaminación, Terrorismo, Eventos y Tráfico.
- **fuente:** corresponde a la cuenta de twitter, periódico o a la persona a la que ha aportado dicha información.

- **verificado:** aparece en los documentos que se insertan cuando un usuario añade una alerta. Este campo mostrar las alertas, ya que las que están verificadas se muestran de formas diferentes.



29- Figura 5.2 - Campos alertas Madridiario

30- Figura 5.3 - Campos alertas Twitter

En la colección ***estadísticas*** se tienen todas las alertas que van a ser utilizadas para las estadísticas generales y las estadísticas por distrito de la web. Se obtienen al igual que la colección alertas, tanto de la red social Twitter como del periódico Madridiario[58].

En cada documento de esta colección (Figura 5.4) se tienen los siguientes campos:

- **_id:** id generado automáticamente por mongo al insertar un documento.

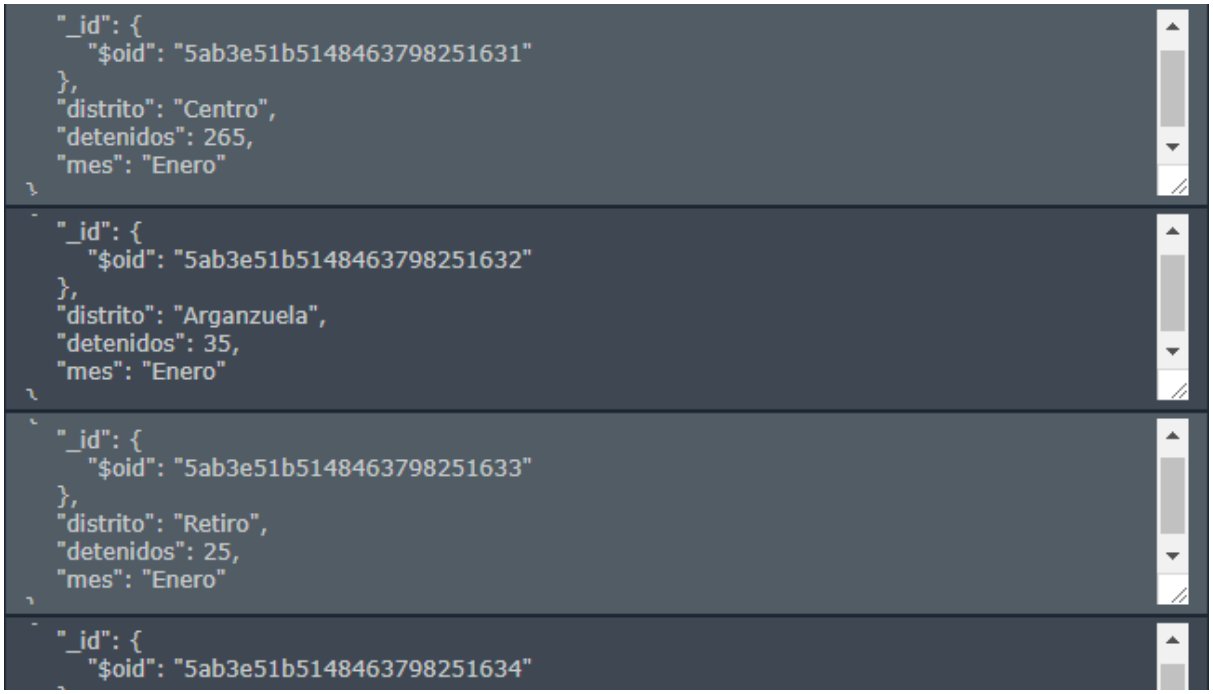
- **distrito:** distrito de Madrid donde ha sucedido la alerta correspondiente. Si es una alerta general de Madrid, este campo tomará el valor de general.
- **categoría:** categoría a la que corresponde la alerta. Puede tomar el valor de Criminalidad, Desastres y accidentes, Contaminación, Terrorismo, Eventos y Tráfico.
- **mes**: número del mes en el que ha sucedido la alerta.

31- Figura 5.4 - Campos estadísticas

A continuación, se detallan las tres colecciones utilizadas para las estadísticas relacionadas con los datos obtenidos de la Policía Municipal de Madrid[23]:

Dentro de la colección ***estDetenidos*** (Figura 5.5), se guardan las estadísticas relacionadas con las detenciones que ha habido en cada distrito de Madrid. En cada documento se tienen los siguientes campos:

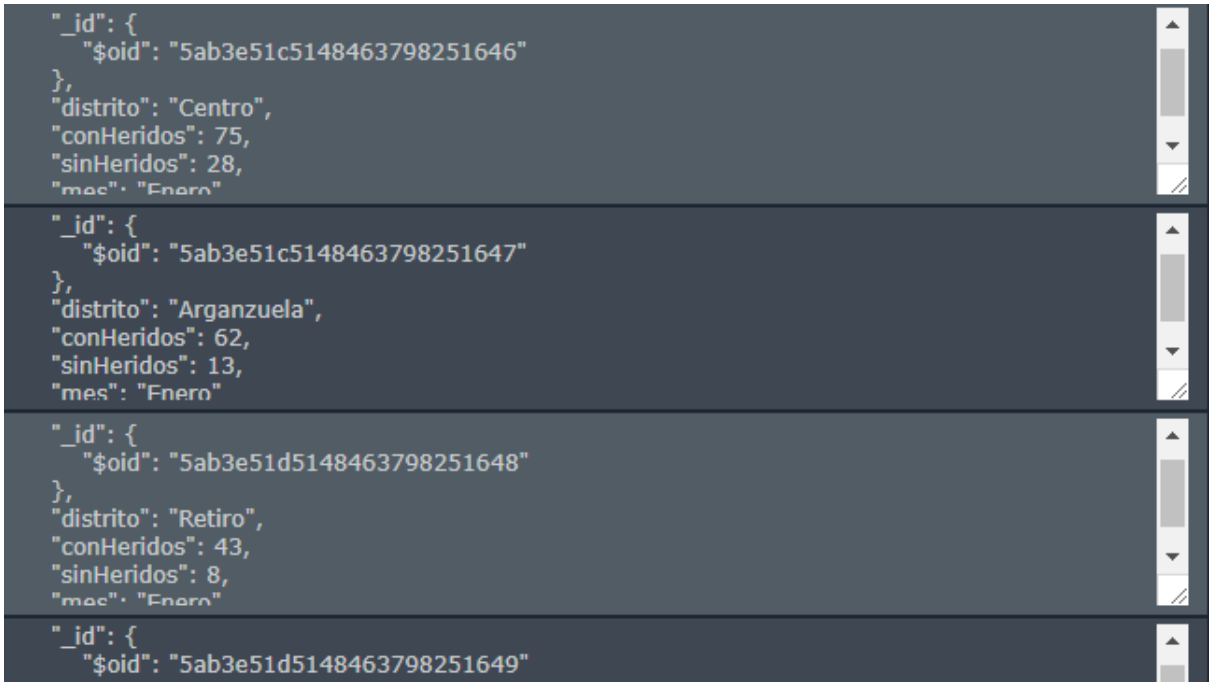
- **_id:** id generado automáticamente por mongo al insertar un documento.
- **distrito:** distrito de Madrid donde ha sucedido la alerta correspondiente. Si es una alerta general de Madrid, este campo tomará el valor de general.
- **detenidos:** número de estadísticas con detenidos en este distrito.
- **mes**: mes en el que ha sucedido la alerta.



32- Figura 5.5 - Campos estadísticas detenidos

Dentro de la colección ***estAccidentes*** (Figura 5.6), se guardan las estadísticas relacionadas con accidentes (con y sin heridos) en cada distrito de Madrid. En cada documento se tienen los siguientes campos:

- **_id:** id generado automáticamente por mongo al insertar un documento.
- **distrito:** distrito de Madrid donde ha sucedido la alerta correspondiente. Si es una alerta general de Madrid, este campo tomará el valor de general.
- **conHeridos:** número de estadísticas con heridos.
- **sinHeridos**: número de estadísticas sin heridos.
- **mes**: mes en el que ha sucedido la alerta.



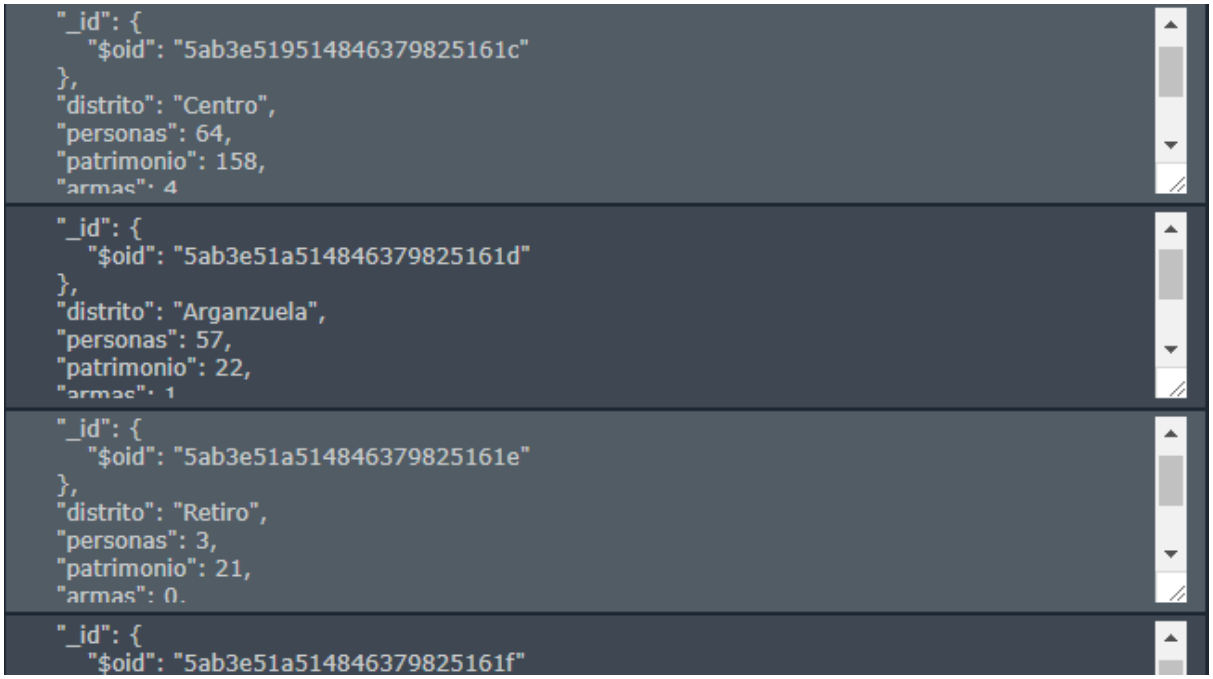
33- Figura 5.6 - Campos estadísticas accidentes

Dentro de la colección ***estSeguridad*** (Figura 5.7, 5.8) se guardan las estadísticas relacionadas con la seguridad ciudadana en cada distrito de Madrid. En cada documento se tienen los siguientes campos:

- **_id:** id generado automáticamente por mongo al insertar un documento.
- **distrito:** distrito de Madrid donde ha sucedido la alerta correspondiente. Si es una alerta general de Madrid, este campo tomará el valor de general.
- **personas:** número de estadísticas de la seguridad ciudadana relacionadas con personas.
- **patrimonio**: número de estadísticas relacionadas con el patrimonio.
- **armas**: número de estadísticas relacionadas con armas.
- **ten_drogas**: número de estadísticas relacionadas con la tenencia de drogas.
- **con_drogas**: número de estadísticas relacionadas con el consumo de drogas.
- **mes**: mes en el que ha sucedido la alerta.

```
{
    "_id": {
        "$oid": "5ab3e519514846379825161c"
    },
    "distrito": "Centro",
    "personas": 64,
    "patrimonio": 158,
    "armas": 4,
    "ten_drogas": 251,
    "con_drogas": 84,
    "mes": "Enero"
}
```


34- Figura 5.7 - Campos estadísticas seguridad



```
"_id": {
  "$oid": "5ab3e519514846379825161c"
},
"distrito": "Centro",
"personas": 64,
"patrimonio": 158,
"armas": 4

"_id": {
  "$oid": "5ab3e51a514846379825161d"
},
"distrito": "Arganzuela",
"personas": 57,
"patrimonio": 22,
"armas": 1

"_id": {
  "$oid": "5ab3e51a514846379825161e"
},
"distrito": "Retiro",
"personas": 3,
"patrimonio": 21,
"armas": 0

"_id": {
  "$oid": "5ab3e51a514846379825161f"
```


35- Figura 5.8 - Alertas estadísticas seguridad

6. DISEÑO

Uno de los aspectos fundamentales en cualquier aplicación es el diseño. Para desarrollar una buena aplicación o página web, es necesario disponer de una buena interfaz. Una interfaz poco intuitiva, pobre o mal diseñada podría provocar desinterés o falta de comprensión por parte del usuario y por tanto, un desastre o fracaso. Los usuarios no van a confiar en un sitio web con un aspecto anticuado o difícil de entender especialmente si se venden productos o servicios o si buscan información fiable.

En este sentido, hay varios aspectos que deben tenerse en cuenta: la usabilidad, la accesibilidad y el equilibrio entre la funcionalidad y la estética. Una buena combinación de ellos es algo fundamental para realizar un buen diseño.

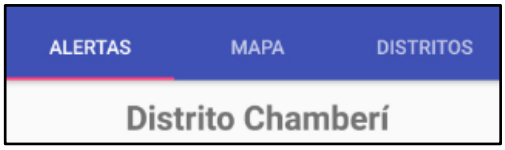
Para empezar el diseño, en primer lugar, hay que identificar a los distintos usuarios que lo van a utilizar. En este sentido, tanto la aplicación web como la app Android están pensadas para un usuario que disponga de conocimientos de informática a nivel de usuario. Por otro lado, el principal objetivo era crear interfaces que fueran sencillas, amistosas, consistentes, intuitivas y fáciles de usar.

A continuación, se explica los detalles del diseño realizado tanto en la aplicación web como en la app Android.

6.1 DISEÑO DE LA APP MAD ALERT

Dentro de las aplicaciones móviles la **navegación** es un aspecto esencial. La forma en la que se disponen los diversos contenidos y se navega por ellos en las diferentes pantallas es un factor clave en el diseño. Es por ello por lo que hay que conseguir evitar que el usuario tenga sensación de desorientación promoviendo una sensación de control con una navegación intuitiva y sencilla. Para ello, Android utiliza una serie de elementos que hacen posible todo esto como son las pestañas, los menús, las barras de navegación, las listas, los gestos, etc.

- **Pestañas**: conocidas como “tabs”, se pueden utilizar para agrupar elementos relacionados o para cambiar entre pantallas que tienen la misma jerarquía. Es muy importante ir indicando siempre dónde se está y hacia dónde se va. Por ello, un buen uso es destacar siempre la pestaña seleccionada. En la app de MadAlert la pestaña en donde se está se destaca subrayando justo esa pestaña con el color secundario de la app (Figura 6.1.1), color del que se hablará un poco más abajo.



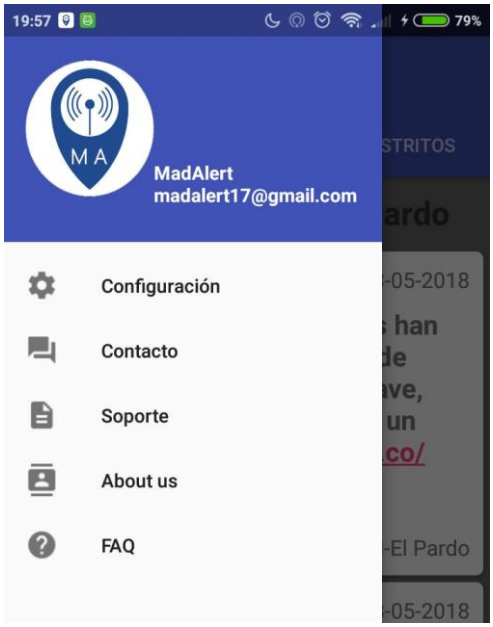
36- Figura 6.1.1 - Tabs app

- **Listas**: es una forma de mostrar ítems en la que se permite al usuario acceder a alguno de ellos tocando sobre él.

Una tarea recurrente en el desarrollo de software suele ser mostrar los datos de manera tabular. ASP.NET proporciona varias herramientas para mostrar los datos tabulares en una cuadrícula, entre las que se incluye el control GridView. Con este control se puede mostrar, editar y eliminar datos. En la pestaña distritos, al mostrar las posibles categorías a seleccionar, se dispone el contenido utilizando este control.

- **Menús**: la existencia de un menú es una de las cosas más imprescindibles en toda app. Existen varios tipos que predominan en la actualidad. Uno de ellos es el conocido como “icono de la hamburguesa” y es el que utiliza la app de MadAlert para el menú de configuración, contacto, soporte e información. Las ventajas del uso de este menú son un mejor aprovechamiento del espacio y una forma cómoda de navegar a lo largo del contenido. El único inconveniente podría ser la necesidad de desplegar el menú para poder visualizar todas las posibles opciones. El menú desplegable que utiliza esta aplicación móvil se puede visualizar en la Figura 6.1.2 (icono) y en la Figura 6.1.3:

37- Figura 6.1.2 - Icono menú “hamburguesa”



38- Figura 6.1.3 - Menú lateral app

- **Gestos**: conocer todas las posibilidades que proporciona Android a la hora de navegar por la app a través de diversos gestos con los dedos es muy importante para una buena experiencia con la aplicación. La incorporación de gestos facilita mucho la navegación y la fluidez. Algunos de los gestos son los siguientes:

- **Scroll**: Suele usarse para definir el gesto de desplazarse por la pantalla de arriba hacia abajo y viceversa. Pulsando en la pantalla consiste en mover el dedo arriba o abajo.

- **Swipe**: La traducción más adecuada para el contexto de las pantallas táctiles sería "deslizar", a diferencia de scroll, el desplazamiento sería de derecha a izquierda o viceversa.

- **Pinch**: Significa “pellizcar”, es la típica acción que hacemos juntando o separando dos dedos para, por ejemplo, hacer zoom en una imagen o mapa.

- **Swipe to Refresh**: También conocido como Pull to Refresh[1] es el mecanismo estándar para actualizar las aplicaciones móviles. Consiste en deslizar el listado o grid hacia abajo y así hacer que sea posible la actualización de su contenido.

Respecto a los gestos anteriormente mencionados, en la aplicación de MadAlert se utilizan de la siguiente manera:

- Cuando se abre el desplegable de seleccionar un distrito tanto en la opción de configuración como en la pestaña de buscar distritos es necesario utilizar el **Scroll**. Todo dependerá también del tamaño de la pantalla utilizada, es decir, cuanto más pequeña sea más necesario sería el Scroll.

- Para visualizar el menú lateral izquierdo y para moverse por las diferentes pestañas o tabs se debe utilizar el **Swipe**. Es decir, se desplaza el dedo a la izquierda o a la derecha en función del tab en el que se encuentre y a cuál desee moverse.

- En el mapa de Madrid donde se encuentran los marcadores, es posible utilizar el gesto llamado **Pinch** para modificar el zoom.

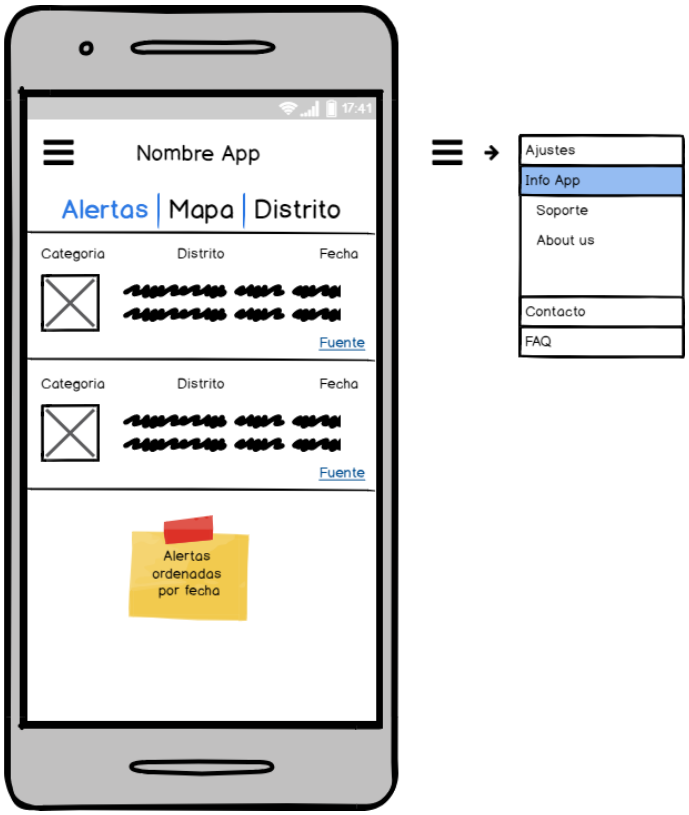
- A la hora de actualizar las pestañas de alertas y distritos de la aplicación es necesario utilizar el gesto de **Swipe to Refresh**. Deslizando hacia abajo se actualizan las alertas mostradas y se añaden las "nuevas" de la base de datos en el caso de que las haya.

En la página de desarrolladores de Android se explican unos principios de diseño muy interesantes. Entre ellos cabe destacar tanto el principio conocido como "Simplifica mi vida" como el de "Cautívame".

En resumen lo que se quiere remarcar con estos principios para desarrollar en Android es la importancia de la simplicidad y la facilidad de uso, la repetición en la utilización de botones o gestos típicos presentes en la mayoría de aplicaciones actuales, (en la medida de lo posible) siempre es mejor hacer uso de cosas gráficas antes que de texto, guardar las preferencias del usuario para que no deba buscar una y otra vez lo mismo, brevedad en las palabras, mostrar sólo lo realmente necesario y proporcionar al usuario sensación de control (el usuario debe saber dónde se encuentra).

Teniendo en cuenta lo máximo posible todos estos principios anteriormente mencionados y analizando las diversas funcionalidades que se iban a implementar, se diseñaron algunos prototipos. Para ello decidimos utilizar una herramienta muy fácil de utilizar y gratuita llamada "Balsamiq"[11]. El resultado de los prototipos fue el siguiente:

Prototipo 1



39- Figura 6.1.4 - A-Prototipo 1 - Alertas

En este primer prototipo, se puede visualizar el resultado de un boceto correspondiente a la pestaña de alertas. En esta pestaña se visualizan las alertas junto con el distrito al que pertenecen, su fecha, su categoría y su fuente.

Además, como se puede ver en la parte derecha del prototipo, la información de la app (contacto, about us, faq etc.) aparece en el menú desplegable (“icono de la hamburguesa”) de la parte superior izquierda de la pantalla.

Prototipo 2

40- Figura 6.1.5 - A-Prototipo 2 - Mapa

En este segundo prototipo, se puede visualizar el resultado de un boceto correspondiente a la pestaña del mapa. En dicho mapa aparecen los marcadores según la configuración establecida.

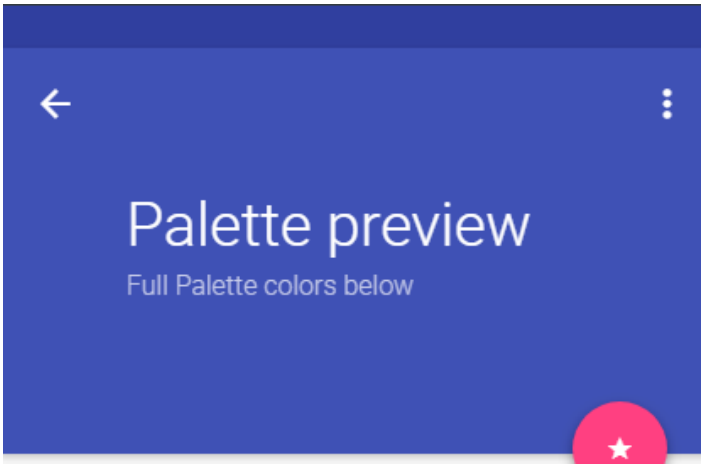
Prototipo 3

41- Figura 6.1.6 - A-Prototipo 3 - Distrito

En este último prototipo, se puede visualizar el resultado de un boceto correspondiente a la pestaña de distritos. Se debe elegir un distrito de entre los distritos existentes en Madrid y a continuación se mostrarán las alertas de dicho distrito.

A la hora de elegir los diferentes colores usados a lo largo de la app hemos utilizado los consejos de Material Design[61]. Ofrecen diferentes paletas de colores de las que se obtienen colores primarios y secundarios que deben de ser utilizados a lo largo de la app de forma adecuada. En esta aplicación la paleta que ha servido de base ha sido la siguiente:

.



42- Figura 6.1.7 - Vista previa paleta

Palette generated by Material Palette - materialpalette.com/indigo/pink

43- Figura 6.1.8 - Paleta detallada

A raíz de esta paleta base hemos obtenido el siguiente resultado:

44- Figura 6.1.9 - Resultado aplicación paleta

6.2 DISEÑO DE LA PÁGINA WEB

Con el fin de realizar un buen diseño para la aplicación web, se ha utilizado un framework de código abierto creado por Twitter llamada **Bootstrap[15]**. Se trata de un framework muy utilizado para el diseño de aplicaciones web que ofrece una gran cantidad de plantillas de diseño con diferentes tipografías, formularios, botones, menús de navegación y otros elementos de diseño basados en HTML5[103] y CSS3[51]. Además, contiene extensiones de JavaScript[55] opcionales adicionales.

Sus principales ventajas son que cuenta con un continuo mantenimiento y actualización (por parte de Twitter), sus plantillas son de sencilla adaptación responsive (adaptable a diferentes dispositivos), utiliza un sistema de cuadrilla o Grid system, se integra con librerías JavaScript[55] y se trata de una herramienta activa, sencilla, que se encuentra

en continuo crecimiento y que cada vez es más popular en el mundo de los desarrolladores web.

A pesar de todas las funcionalidades ofrecidas por Bootstrap, siempre es importante pensar en el diseño de la web teniendo en cuenta cuáles van a ser los diferentes usuarios de la misma. A su vez, es importante seguir también una serie de principios de diseño para conseguir que la aplicación web sea fácil de utilizar, intuitiva y esté bien diseñada. Para ello se han seguido las **8 reglas de oro del diseño de interfaces** sacadas de los apuntes de la asignatura "Interfaces de Usuario"[91].

1. **Esforzarse por conseguir consistencia**:

   Es decir, utilizar de forma consistente colores, composición, fuentes, secuencias de acciones, menús, pantallas de ayuda etc.

2. **Atender a la usabilidad universal**:

   Es necesario reconocer las necesidades de los distintos tipos de usuarios, facilitando la transformación del contenido. Se deben tener en cuenta las diferencias entre un principiante o un experto, rangos de edad, discapacidades etc. Para ello se añaden explicaciones (para principiantes) y atajos u opciones avanzadas (para expertos).

3. **Ofrecer retroalimentación informativa**:

   Por cada acción que realice el usuario debe haber algún tipo de retroalimentación por parte del sistema. Dependiendo de la frecuencia que tenga la acción la respuesta será más o menos grande o llamativa.

4. **Diseñar diálogos para conducir la finalización**:

   Las secuencias de acciones se deben organizar en varios grupos: comienzo, mitad y finalización. Esto es muy importante en todas aquellas aplicaciones que tengan un proceso de compra, no tanto precisamente en nuestro caso.

5. **Prevenir errores**:

   Siempre en la medida de lo posible debemos diseñar la interfaz de tal forma que se evite que los usuarios puedan cometer errores graves. Un ejemplo sería deshabilitar alguna opción del menú cuando no pueda ser usada, o al rellenar un formulario indicar claramente los errores, el por qué y cómo solucionarlos correctamente.

6. **Permitir deshacer acciones de forma fácil**:

   Las acciones deberían ser reversibles haciendo que los usuarios sepan que pueden recuperar cualquier error fomentando a su vez la exploración de opciones desconocidas.

7. **Maximizar la sensación de control**:

   Regla de los tres clicks: los usuarios deben de ser capaces de encontrar la información que están buscando en tres clicks de ratón.

8. **Reducir la carga de memoria a corto plazo**:

   Regla de siete más/menos dos elementos: según diversos estudios, los humanos solamente pueden retener a la vez entre 5 y 9 cosas en la memoria a corto plazo. Esta regla impone que las visualizaciones sean simples y se condense la información.

La aplicación que se ha realizado en el contexto de este proyecto ha aplicado los principios aprendidos en asignaturas como la anteriormente mencionada, Interfaces de Usuario[91]. Algunos de estos principios se basan en teorías psicológicas como los principios de Gestalt [17]:

- **Principio de proximidad**: este principio se basa en la siguiente idea: "*Cuando los elementos están próximos, el cerebro interpreta que están relacionados*". Hacer uso de este principio tiene como ventaja que al agrupar elementos similares entre sí se consigue que el usuario sepa que están relacionados, los relacione mentalmente entre sí y los recuerde como un grupo (chunk).

  Se puede explotar la proximidad de dos maneras, o bien agrupando los controles relacionados o bien agrupando los controles que representan secuencias de acciones.

  Además, toda secuencia debería tener un inicio, un desarrollo y un cierre.

  Una ley muy relacionada con este principio es la **Ley de Fitt**[24]**,** la cual explica que el coste en tiempo para activar un control es proporcional a la distancia a la que se encuentra e inversamente proporcional al tamaño. Con esto se puede sacar como conclusión que los movimientos deben ser cortos y precisos, que desplazar un elemento muy pequeño ubicado a mucha distancia es costoso, que los bordes y las esquinas son siempre más fáciles de apuntar.

- **Principio de consistencia**: los usuarios aprenden más fácilmente los conceptos que son consistentes con su conocimiento previo. Todos los controles con funcionalidad similar deberían tener una apariencia y exhibir un comportamiento similar. Por otra parte, el principio aconseja desarrollar aplicaciones consistentes con las expresiones habituales dentro de un sistema específico, como emplear los

mismos atajos de teclado, los mismos elementos visuales o los mismos procedimientos. Llevar a cabo y seguir este principio es importante para la memorabilidad y la facilidad de aprendizaje.

- **Principio de visibilidad y feedback**: el principio de visibilidad tiene que ver con cómo la interfaz hace uso de diferentes mecanismos para transmitir cuál es el estado actual y las diversas acciones posibles o recomendables en ese momento preciso. Existen también mecanismos para dirigir la atención como son la tipografía, la opacidad, el tono o el relieve. El sistema siempre debe reaccionar ante cualquier acción, especialmente si la respuesta puede tardar debido a un retardo en el sistema o el sistema de input de acciones no es fiable. Para ello es recomendable hacer uso de unos principios que siempre suelen ser ignorados: el proceso siempre debería ser cancelable, debería facilitarse al usuario feedback sobre la duración esperada aproximada y es suficiente una estimación del orden de magnitud. Para informar al usuario de la estimación del tiempo restante se pueden utilizar, por ejemplo, barras de progreso.

- **Principio de gestión del estado visible**: evitar las brechas entre el estado de la aplicación y el estado mental del usuario mostrando:

    - **Estado de navegación:** es el estado que informa al usuario de dónde se encuentra en un flujo de ejecución. Haciendo uso por ejemplo de: migas de pan en sitios web, jerarquía de opciones o pestañas resaltadas (para así recordar en qué pestaña se encuentra el usuario).

    - **Estado del modelo**: la interfaz debe sugerir el estado interno en el que se encuentra la aplicación. Se pueden mostrar: las propiedades del objeto seleccionado, el estado general del sistema (recursos disponibles, operaciones posibles…) o el estado de la última operación.

    - **Estado de la interfaz**: la propia interfaz debe ofrecer información sobre el estado actual indicando claramente los elementos activos, los seleccionados, las opciones posibles sobre el objeto seleccionado, cambios en el cursor...

- **Principio de libertad y control del usuario**: este principio dice que el usuario debe sentirse con el control del proceso. Debe sentir que puede explorar la interfaz cómodamente (sin miedo a romper o estropear nada) y que es él quien controla al programa y no a la inversa. Requiere encontrar un equilibrio entre llevar al usuario a donde se quiere, y permitirle ir a donde quiera.

Para conseguir todo esto es importante ofrecer al usuario opciones y no preguntas, ofrecer las herramientas que necesita, reducir el número de interacciones y cuadros de diálogo, evitar diálogos y ventanas modales (que bloqueen la ejecución del proceso padre y roban el control al usuario) y ofrecer manipulación directa e interacción gráfica.

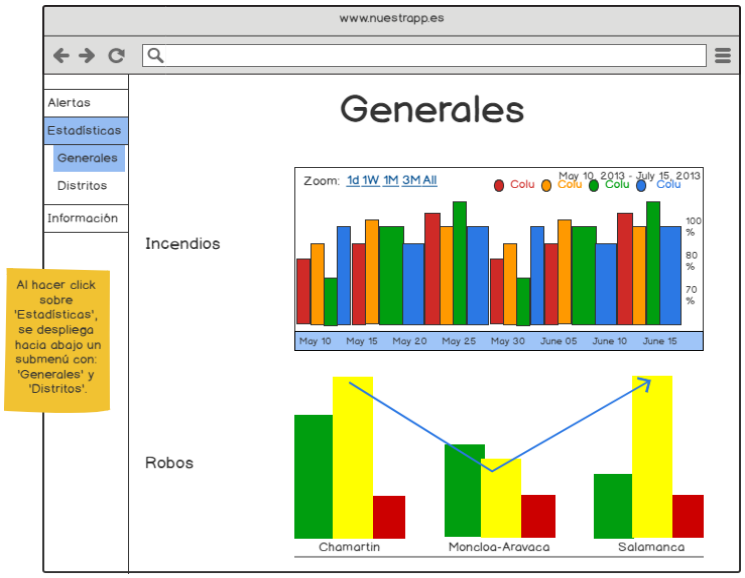
Teniendo en cuenta lo máximo posible todos estos principios mencionados y analizando las diversas interacciones que se iban a realizar, se diseñaron los siguientes prototipos:

Prototipo 1

45- Figura 6.2.1 - W-Prototipo 1 - Alertas

En este primer prototipo, se puede visualizar el resultado de un boceto correspondiente a una búsqueda de alertas por distrito. En primer lugar, se puede apreciar la existencia de un menú lateral izquierdo utilizado para navegar por la web de manera rápida y sencilla. En segundo lugar, se muestra cómo se podrían listar las alertas de un distrito seleccionado anteriormente. Por cada alerta, aparte del titular, se va mostrando también cuál es su categoría, su fecha y la fuente de la que proviene dicha información.

Prototipo 2



46- Figura 6.2.2 - W-Prototipo 2 - Estadísticas generales

En este segundo prototipo, se puede visualizar un posible resultado de cómo sería la página correspondiente a las estadísticas generales por categorías. Cabe mencionar que para acceder a esta página anteriormente se debe haber pulsado en la opción “Generales” (se encuentra en el submenú del menú lateral izquierdo). El objetivo principal es mostrar a los usuarios las estadísticas de la categoría seleccionada con sus gráficos correspondientes.

Prototipo 3

47- Figura 6.2.3 - W-Prototipo 3 - Estadísticas distritos

En este tercer prototipo, se puede visualizar un posible resultado de cómo sería la página correspondiente a las estadísticas por distritos. Cabe mencionar que para acceder a esta página anteriormente se debe haber pulsado en la opción “Distritos” (se encuentra en el submenú del menú lateral izquierdo) y haber seleccionado un distrito entre los que aparecen en el desplegable superior. El objetivo principal es mostrar a los usuarios las estadísticas por categorías del distrito seleccionado con sus gráficos correspondientes.

Prototipo 4

48- Figura 6.2.4 - W-Prototipo 4 - Información - Soporte

En este cuarto prototipo, se puede visualizar un posible resultado de cómo sería la página correspondiente al Soporte (un apartado dentro de la pestaña de información de la web). Es decir, en esta página se explican los términos y condiciones y la información legal de MadAlert.

Prototipo 5

49- Figura 6.2.5 - W-Prototipo 5 - Información - About us

En este último prototipo, se puede visualizar un posible resultado de cómo sería la página correspondiente al About us (un apartado dentro de la pestaña de información de la web). Es decir, en esta página se explica quiénes han sido los creadores.

En base al diseño presentado, se puede decir que se trata de una web consistente (se utiliza de forma consistente los colores, la composición, las fuentes etc), sencilla y fácil de utilizar para todo tipo de usuarios (desde los más expertos a los que no lo son), con retroalimentación en las acciones que realiza el usuario (por ejemplo, en el formulario de añadir una nueva alerta aparece un aviso indicando si ha sido posible o no añadirla), con prevención de errores (si el usuario no completa bien el formulario de añadir una nueva alerta aparece un aviso indicando cuál es o cuáles son los errores cometidos), en la que la información se encuentra con muy pocos clicks gracias al menú lateral. Asimismo, no existe un exceso de carga de memoria a corto plazo (no es necesario que el usuario retenga mucha información debido a las visualizaciones simples utilizadas).

Por último, cumpliendo con el principio de proximidad, dentro del menú lateral izquierdo, se ha decidido realizar dos agrupaciones de elementos. En primer lugar se ha agrupado los diferentes tipos de estadísticas en una sola pestaña llamada “Estadísticas” de tal forma que, cuando el usuario pulsa sobre ella, aparece un submenú inmediatamente debajo con tres opciones (generales, por distritos y de la Policía). En segundo lugar, ocurre exactamente lo mismo con la pestaña “Información” que, pulsando sobre ella, aparece un submenú con dos opciones (soporte y about us). Haciendo uso de este principio y agrupando elementos en nuestro menú, se consigue que el usuario sepa que los elementos están relacionados entre sí y así le sea más fácil recordarlos como grupo (chunk).

Como se trata de una web fundamentalmente informativa, es cierto que no existe la opción de deshacer ninguna acción que haya realizado el usuario. Pero también es cierto que lo único que se puede realizar en la aplicación web es o bien añadir una nueva alerta o bien ponerse en contacto con el equipo de desarrollo. Para estas dos acciones concretas, se ha realizado un control de errores con su feedback correspondiente. Así se consigue que el usuario sienta una sensación de control y además conozca qué está pasando en todo momento.

7. IMPLEMENTACIÓN

A continuación, se describe la implementación realizada tanto en la aplicación web como en la aplicación móvil. Del mismo modo, se detalla la implementación llevada a cabo para la obtención de datos a través de varios scripts de Python.

7.1 IMPLEMENTACIÓN ANDROID

En este apartado se explicará detalladamente cómo se han desarrollado las implementaciones de la aplicación Android.

Como todo proyecto desarrollado en Android Studio, existen tres directorios principales que abarcan todo el código de la aplicación. Estos paquetes son:

- ***Manifests***: contiene un solo archivo .xml, el cual toda aplicación Android debe tener para su correcto funcionamiento. El archivo de manifiesto proporciona información esencial sobre tu aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la app. En este archivo se declaran todas las actividades implementadas e incluye los permisos que la aplicación requiere. Para este proyecto han sido necesarios permisos de localización y acceso a internet:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

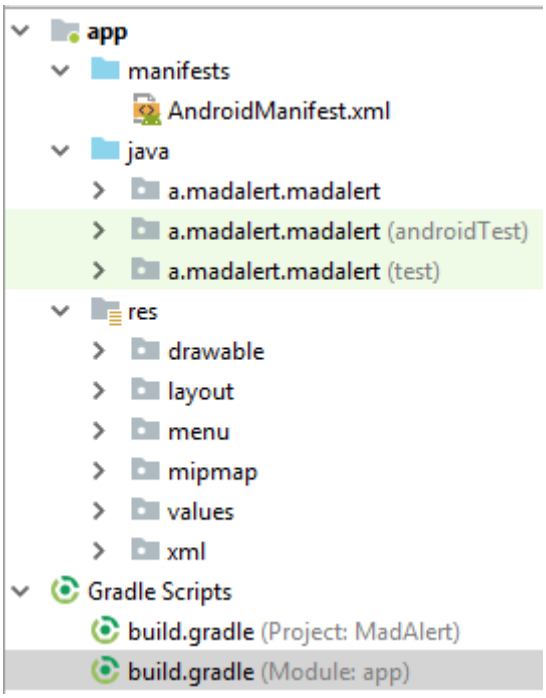
50- Figura 7.1.1 - Permisos AndroidManifest

- **Java**: aquí es donde se aloja todo el grueso del código, esto es, las clases que se han implementado para la creación de la aplicación. En el caso de este proyecto en particular, aparecen las distintas actividades que son las vistas de la app, junto con tres paquetes que recogen el código usado para la conexión a la base datos, el clasificador empleado, el paquete 'Localización' que contiene la clase 'Radio', implementada para el uso del algoritmo que explicaremos más adelante. Por último, la clase Adapter que sirve para las distintas vistas que se explicarán a lo largo de este punto.

- **Res:** recoge todos los recursos de la aplicación. Estos varían desde las imágenes (carpetas '*mipmap', 'drawable'*) a los distintos textos, colores y dimensiones empleadas (carpeta '*values'*). En este paquete destaca el directorio '*layout',* el cual

comprende todos los ficheros de las vistas que se asocian con sus homólogos en java.

Cabe mencionar que Android Studio crea por defecto también un apartado denominado '*Gradle Scripts*' donde se encuentra el archivo *'app'*, que se encarga de señalar todas las dependencias de la aplicación. Esto evita que se tenga que buscar, descargar y copiar manualmente paquetes ejecutables de las dependencias en el directorio del proyecto.



51- Figura 7.1.2 - Paquetes Android Studio

Para que la conexión con la base de datos fuera posible se ha hecho uso de varios archivos .jar. Estos simplemente son archivos java que se encuentran empaquetados para que Android Studio pueda interpretarlos. Estos tres archivos se encuentran en una carpeta denominada 'jar'[29] y simplemente añadiendo su ruta en el apartado *'dependencies'* se consigue que la aplicación se sirva de ellos. Estos son:

b-son-3.5.0.jar
mongodb-driver-3.6.3.jar
mongodb-driver-core-3.6.3.jar

Antes de pasar a presentar las funcionalidades de la aplicación y con el objetivo de no ser demasiado repetitivos en cada uno de los distintos puntos, es importante explicar que se ha usado la clase ***'SharedPreferences'***[99]. Las preferencias no son más que datos que una aplicación debe guardar para personalizar la experiencia del usuario, por ejemplo, información personal, opciones de presentación, etc. De este modo, cada vez que el usuario

ingresa en la aplicación, sus preferencias aparecen tal y como las dejó la última vez que las modificó. Por ello, en toda clase que requiere comprobar las preferencias del usuario, se ha creado una función *initSharedPreferences()* encargada de recoger todos esos valores que se usan para la vista. Esta función siempre comienza de este modo:

```
private void initSharedPreferences() {
    mSharedPreferences = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
```

52- Figura 7.1.3 - *initSharedPreferences()*

Para obtener y establecer el atributo de la preferencia, la clase *SharedPreferences*[99] posee funciones por defecto que se encargan de ello. Estas son getXXX y putXXX, respectivamente, siendo 'XXX' el tipo de la variable a guardar. Tras establecer la preferencia es necesario utilizar el método *apply()* para que esta se guarde correctamente.

Resulta imprescindible también inicializar el atributo privado (FIGURA 7.1.4) *Editor* para que las preferencias se vean afectadas en caso de ser modificadas. Esto se realiza en la función *onCreate()* de cada clase, tras inicializar las preferencias.

```
private SharedPreferences mSharedPreferences;
private SharedPreferences.Editor editor;
```

53- Figura 7.1.4 - Atributos privados *SharedPreferences*

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_config);
    CompositeDisposable mSubscriptions = new CompositeDisposable();
    initSharedPreferences();
    editor = mSharedPreferences.edit(); // para guardar las configuraciones
```

54- Figura 7.1.5 - Inicialización atributo editor

Esta es la estructura interna que presenta el proyecto. Una vez introducido esto, se procede a explicar detalladamente la implementación de las distintas funcionalidades ya mencionadas en el apartado 2.1 de este mismo documento.

A-1: Cargar alertas según la configuración del usuario

Al iniciar la aplicación de MadAlert, se inicia el fragmento de Alertas y se comprueba si es la primera vez que el usuario usa la aplicación o si la tiene configurada.

Si es la primera vez que el usuario entra en la aplicación o no tiene elegida la configuración se le muestra un mensaje para que configure la aplicación. Para saber si el

usuario tiene configuración establecida o no, se guarda en el SharedPreferences un booleano que indica si es la primera vez del usuario en la aplicación. De este modo, lo primero que se comprueba es si el booleano está a true y en caso de estarlo se muestra un mensaje que le indica al usuario que configure su aplicación.

```
if (mFirstTime) {
    // first time task
    firstTime.setText(R.string.firstTime);
}
else {
    try {
        loadAlerta();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (JSONException e) {
        e.printStackTrace();
    }

}
```

55- Figura 7.1.A-1.1 - Comprobación primer inicio en la aplicación

En caso de que el usuario tenga una configuración establecida, es decir, el booleano anterior este a false, se llama a la función *'loadAlerta'* que carga las alertas que coinciden con los requisitos de la configuración del usuario.

Esta función actúa de diferente forma, en función de si el usuario ha elegido buscar alertas por su ubicación o por un distrito en concreto.

- **Buscar alertas por ubicación del usuario:** se obtienen las coordenadas de la ubicación del usuario y se hace una llamada a la api de Google, para obtener los datos de las coordenadas del usuario. La api devuelve un Json, que es tratado para guardar solo el distrito al que corresponde. Una vez que se tiene el distrito correspondiente se hace uso del algoritmo *"Obtener los distritos que engloban una circunferencia de x kilómetro de radio*", con el que se obtienen los distritos que se van a buscar alertas y se guardan en un atributo privado.

  Cabe mencionar que las coordenadas de ubicación del usuario se van actualizando cuando se detecta un movimiento mayor a 50 metros. Se ha elegido esta distancia entre las coordenadas antiguas y las nuevas, debido a que se ha considerado significativa y podría suponer un cambio de distrito. El hecho de no actualizarlas en todo momento se debe a que por cada cambio hay que hacer una petición a la API de Google Maps para saber a qué distrito corresponden las nuevas coordenadas y el número de peticiones a dicha API está limitada por día.

```
private void loadAlerta() throws IOException, JSONException {
    auxDistrito="";
    distRadio = new ArrayList<>();
    if(mCheckedSw) {
        latitud = mSharedPreferences.getString( s: "latitud",  s1: "");
        longitud = mSharedPreferences.getString( s: "longitud",  s1: "");
        obtenerDistrito(latitud, longitud);
        for(int j = 0; j < distRadio.size(); j++) {
            mDistrito += "," + distRadio.get(j);
        }
    }
```

56- Figura 7.1.A-1.2 - Obtener distritos cuando el usuario tiene la ubicación activada

- **Buscar alertas por el distrito de configuración**: se comprueba que el distrito elegido es diferente de todos o el radio es mayor que cero, si se cumple está condición se llama al algoritmo *"Obtener los distritos que engloban una circunferencia de x kilómetro de radio"* y se obtienen los distritos por el que el usuario quiere buscar alertas. Si la condición anterior no se cumple, se busca por el distrito elegido.

```
else{
    mDistrito = mSharedPreferences.getString( s: "distritoConf",  s1: "");
    auxDistrito = mDistrito;
    if(!mDistrito.equals("Todos") || kms > 0) { //Cuando hay radio y distrito!=Todos
        recorrerRadio(mCheckedSw);
        for(int j = 0; j < distRadio.size(); j++) {
            auxDistrito += "," + distRadio.get(j);
        }
    }
```

57- Figura 7.1.A-1.3 - Obtener distritos cuando la ubicación no está activada.

A continuación, se obtiene del *SharedPreferences*[99] el booleano '*mTodas*', si es true indica que se han elegido todas las categorías y se realiza una petición a la api que devuelve la lista de alertas de los distritos obtenidos sin filtrar por categoría. En caso contrario se realiza una llamada a la api que devuelve la lista de alertas filtradas por categorías y distritos.

```
if(mTodas) {
    mSub.add(NetworkUtil.getRetrofit().getAlertasDistrito(auxDistrito)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(io.reactivex.schedulers.Schedulers.io())
            .subscribe(this::handleResponse, this::handleError));
}
else{
    mSub.add(NetworkUtil.getRetrofit().getAlertasDistritoCategoria(auxDistrito, mHayCategorias)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(io.reactivex.schedulers.Schedulers.io())
            .subscribe(this::handleResponse, this::handleError));
}
```

58- Figura 7.1.A-1.4 - Obtención de alertas mediante petición a la api

La lista de alertas obtenida se plasma en un *RecyclerView* que se rellena con la información de cada alerta que se encuentra en la clase *DataAdapter*.

El distrito se muestra en cada caja del *RecyclerView* si está dentro del radio o el distrito que está seleccionado es 'Todos', de esta manera se logra distinguir el distrito al que pertenece la alerta. Para mostrar si dicha alerta es verificada o no, se comprueba si es false, en este caso no aparece el icono de verificado y en caso contrario se le asigna la imagen. La categoría y la fecha se añaden con la información obtenida con *getCategoria* y *getFecha*, esa información viene de la alerta obtenida de base de datos. A cada alerta se le asigna una imagen u otra dependiendo de la categoría, por ello se mira cual es la categoría de la alerta y según la que sea se añade la imagen.

Algunas de las alertas contienen una *url* a la noticia de Madridiario[58], por eso se comprueba que la url sea distinta de *null* y su fuente sea Madridiario[58], de este modo se hace que cambie de color el texto y se le añade el enlace, de modo que cuando el usuario pulse sobre la alerta se le redirige a la url correspondiente. En caso de que no tengan url el texto no cambia de color y al pulsar no tiene efecto.

```
public void onBindViewHolder(ViewHolder holder, int position) {

    final Alertas alerta = mAndroidList.get(position);

    if((mRadio && mKm > 0) || mDistrito.equals("Todos")) {
        holder.mTvDistrito.setText(alerta.getDistrito());
    }
    holder.mTvCategoria.setText(alerta.getCategoria());
    holder.mTvFecha.setText(alerta.getFecha());

    if((alerta.getVerificado() != null) && !alerta.getVerificado()) {
        holder.mIvVerificado.setImageResource(R.drawable.blanco);
    } else {
        holder.mIvVerificado.setImageResource(R.drawable.verificado);
    }

    if(alerta.getUrl() != null && alerta.getFuente().equals("Madridiario")) { // para madridiario
        text = Html.fromHtml( source: "<a href=" + alerta.getUrl() + ">" + alerta.getAlertas() + "</a>");
        holder.mTvAlertaWeb.setMovementMethod(LinkMovementMethod.getInstance());
        holder.mTvAlertaOther.setText("");
        holder.mTvAlertaWeb.setText(text);
    }
    else { // para twitter e inserciones de usuario
        holder.mTvAlertaWeb.setText("");
        holder.mTvAlertaOther.setText(alerta.getAlertas());
    }

    if(alerta.getCategoria().equals("Desastres y accidentes")) {
        holder.mIvImagen.setImageResource(R.drawable.accident);
    } else if(alerta.getCategoria().equals("Transporte público")) {
        holder.mIvImagen.setImageResource(R.drawable.bus);
    } else if(alerta.getCategoria().equals("Eventos")) {
        holder.mIvImagen.setImageResource(R.drawable.events);
    } else if(alerta.getCategoria().equals("Criminalidad")) {
        holder.mIvImagen.setImageResource(R.drawable.criminal);
    } else if(alerta.getCategoria().equals("Terrorismo")) {
        holder.mIvImagen.setImageResource(R.drawable.terrorism);
    } else if(alerta.getCategoria().equals("Contaminación")) {
        holder.mIvImagen.setImageResource(R.drawable.contamination);
    } else if(alerta.getCategoria().equals("Tráfico")) {
        holder.mIvImagen.setImageResource(R.drawable.traffic);
    }

    holder.mTvFuente.setText("Fuente: " + mAndroidList.get(position).getFuente());
}
```

59- Figura 7.1.A-1.5 - Rellenado de información de cada alerta de la lista

A-2: Actualizar la pestaña de ‘ALERTAS’, ‘DISTRITO’, ‘MAPA’

La actualización de las pestañas alertas y distritos de la aplicación se realiza de la misma manera. Para que se actualice la información, el usuario hace el gesto Swipe Refresh sobre una de las pestañas. Cada una tiene implementado un listener del SwipeRefreshLayout, que cuando se realiza el gesto entra en el método y hace lo correspondiente a cada pestaña.

La actualización para la pestaña mapa es diferente. Para que se actualicen los marcadores existe un botón de *refresh* en la esquina superior derecha del mapa.

- **Pestaña alertas:** se vuelve a llamar al método ‘*loadAlerta’*, que consiste en cargar las alertas con la configuración que tenga el usuario. Aparecen tanto las nuevas alertas que se hayan añadido a la base de datos, como la anteriores.

- **Pestaña distritos:** se ejecuta de nuevo el método *‘loadAlerta’*, cargando las alertas, tanto las nuevas como las antiguas, que cumplen lo seleccionado en el fragmento *‘SeleccionDistritoFragmento.java’*.

  El código utilizado para la actualización tanto de las alertas como de los distritos es idéntico:

```
swipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
    @Override
    public void onRefresh() {
        // Esto se ejecuta cada vez que se realiza el gesto
        loadAlerta();
        swipeRefreshLayout.setRefreshing(false);
    }
});
```

60- Figura 7.1.A-2.1 - Actualización de alertas y distritos

- **Pestaña mapa:** se vuelve a crear el *mapView* cargando los marcadores con las nuevas alertas que se han introducido.

```
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Esto se ejecuta cada vez que se realiza el gesto
        mapView = view.findViewById(R.id.mapG);

        if (mapView != null) {
            mapView.onCreate( bundle: null);
            mapView.onResume();
            mapView.getMapAsync( onMapReadyCallback: MostrarMapa.this);
        }
    }
});
```

61- Figura 7.1.A-2.2 - Actualización de la pestaña mapa

Es preciso señalar que, a la hora de actualizar, si el usuario tiene la ubicación activada se tiene en cuenta las nuevas coordenadas mencionadas en la funcionalidad A-1, por lo tanto, si la aplicación detecta un movimiento mayor a 50 metros a la hora de actualizar tanto la pestaña alerta como mapa se hace con esa nueva ubicación.

A-3: Consultar alertas por categoría y distrito

Se selecciona un distrito del spinner y se selecciona una o varias categorías. Si el usuario selecciona alguna categoría y después 'Todas' se desmarcan las categorías seleccionadas y se marca solo la opción de 'Todas', en el caso de que primero se marque la categoría "Todas" y después cualquier otra se desmarca la opción 'Todas'.

Si no se selecciona ninguna categoría se muestra un mensaje en el SnackBar de que se debe seleccionar al menos una categoría. Si se selecciona Todas se añade un 0 al string *hayCategorías*, en cambio, si se seleccionan varias categorías se recorre *selectedStrings* añadiendo todas las categorías a un string separadas por "," para después ser añadido a *hayCategorias* al *mSharedPreferences*, para tenerlas disponibles en cualquier momento.

La información se envía al nuevo fragmento '*ListaAlertas'* donde se van a mostrar las alertas obtenidas según el distrito del spinner y de *hayCategorias* en el *RecyclerView*. En caso de que esté la categoría 'Todas' seleccionada, se llama a *getAlertasDistrito*, y si hay seleccionadas otras categorías se llama a la función *getAlertasDistritoCategoría*.

```
private void handleResponse(String alerta) {
    SharedPreferences.Editor editor = mSharedPreferences.edit();
    editor.putString("distrito", alerta);

    if(selectedStrings.size()==1 && selectedStrings.get(0)=="Todas"){
        editor.putString("hayCategorias", "0");
    }
    else if (selectedStrings.isEmpty()){
        showSnackBarMessage("¡Debes seleccionar al menos una categoría!");
    }
    else{
        String categorias = "";
        for(int i=0; i< selectedStrings.size();i++){
            categorias=categorias+selectedStrings.get(i);
            if(i < selectedStrings.size()-1){
                categorias=categorias+",";
            }
        }
        editor.putString("hayCategorias", categorias); //Hay que pasar el array de string por aqui

    }
    editor.apply();

    if(!selectedStrings.isEmpty()) {
        getFragmentManager().beginTransaction()
                .replace(R.id.distritos_frame, new ListaAlertas())
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN)
                .addToBackStack(null)
                .commit();
    }
}
```

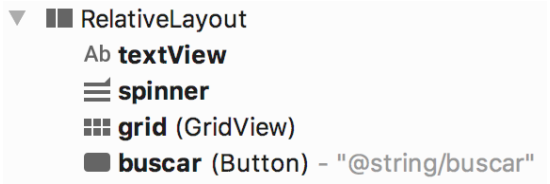
62- Figura 7.1.A-3.1 - Selección de distrito y categorías

```
private void handleResponse(List<Alertas> alertas) {
    String auxDistrito;
    if (!mMapa) {
        textView.setText("Distrito " + mDistrito);
        auxDistrito = mDistrito;
    } else {
        textView.setText("Distrito " + mDistritoMapa);
        auxDistrito = mDistritoMapa;
    }
    mAndroidArrayList = new ArrayList<>(alertas);
    mAdapter = new DataAdapter(mAndroidArrayList,  radio: false, auxDistrito);
    mRecyclerView.setAdapter(mAdapter);
    SharedPreferences.Editor editor = mSharedPreferences.edit();
    editor.putBoolean("vieneMapa", false);
    editor.apply();

    if(mAndroidArrayList.isEmpty())
        firstTime.setText("¡No hay nada que mostrar para esa combinación!");

}
```

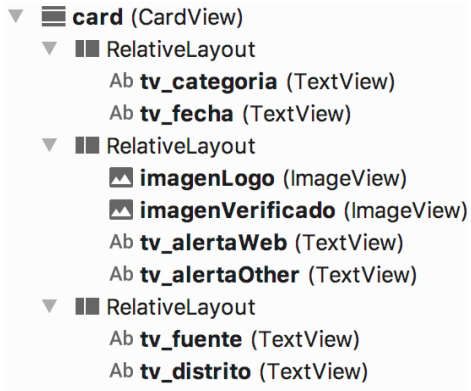
63- Figura 7.1.A-3.2 - Mostrar alertas según distrito y categorías

```
private void loadAlerta() {
    if (!mMapa) { //NO viene del mapa
        if (mHayCategorias.equals("0")) {
            mSub.add(NetworkUtil.getRetrofit().getAlertasDistrito(mDistrito)
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribeOn(io.reactivex.schedulers.Schedulers.io())
                    .subscribe(this::handleResponse, this::handleError));
        } else {
            mSub.add(NetworkUtil.getRetrofit().getAlertasDistritoCategoria(mDistrito, mHayCategorias)
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribeOn(io.reactivex.schedulers.Schedulers.io())
                    .subscribe(this::handleResponse, this::handleError));
        }
    } else { //SI viene del mapa
```

64- Figura 7.1.A-3.3 - Petición de alertas si tiene categorías o es Todas



65- Figura 7.1.A-3.4 - SeleccionarDistrito



66- Figura 7.1.A-3.5 - ListaAlertas

A-4: Consultar alertas en el mapa de Madrid

En el mapa de Google Maps se encuentran los marcadores [41] de los distritos de Madrid. Estos se muestran dependiendo de si tienen alertas que cumplan las condiciones de la actividad *'ConfigActivity.java'*, como pueden ser estar dentro del radio, categorías seleccionadas, etc.

La vista del mapa se crea a partir nuestra ubicación con *setMyLocationEnabled* o del distrito escogido en la configuración anteriormente. Para cada caso obtenemos la latitud y la longitud donde se sitúa el centro del mapa.

Para obtener los marcadores que se muestran en el mapa, se utiliza el radio que se encuentra en la configuración, y dependiendo del tamaño puesto, habrá mayor cantidad de marcadores.

Los distritos que cumplen las condiciones según el radio son añadidos a *distRadio*. Es posible que estos distritos tengan cero alertas porque no disponen de ninguna con esas categorías. Para no dibujar esos marcadores se realiza la consulta a la base de datos que devuelve los distritos y el contador del número de alertas que tiene con la función *getCountAlertasDistrito*.

A la hora de dibujar los marcadores, se obtiene la latitud y longitud, de cada uno de los distritos, del *HashMap* creado con lo devuelto por la función initCoord de la clase *Radio*, y se añaden los marcadores con *addMarker* con la latitud y longitud, y una ventana de información con el nombre del distrito y el número de alertas que tiene.

```
        if(isCheckedSw) {
            map.setMyLocationEnabled(true);

            parsLat = Double.parseDouble(latitud);
            parsLong = Double.parseDouble(longitud);
        }
        else {
            Iterator<Map.Entry<String, ArrayList<Pair<Double,Double>>>> iterator = distCoord.entrySet().iterator();
            boolean encontrado = false;
            while (iterator.hasNext() || !encontrado) {
                Map.Entry<String, ArrayList<Pair<Double,Double>>> it = iterator.next();
                if (distritoConf.equals(it.getKey()) && !encontrado) {
                    encontrado = true;
                    parsLat = it.getValue().get(0).first;
                    parsLong = it.getValue().get(0).second;
                }
            }
        }
        if(kms==0 && isCheckedSw){
            String distritoUbi = mSharedPreferences.getString( key: "distritoUbicacion", defValue: "");
            distRadio.add(distritoUbi);
            mSub.add(NetworkUtil.getRetrofit().getCountAlertasDistrito(distritoUbi,  count: true, cat)
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribeOn(io.reactivex.schedulers.Schedulers.io())
                    .subscribe(this::handleResponse, this::handleError));
        }
        else {
            if(!isCheckedSw && distritoConf.equals("Todos")){
                kms = 25;
                parsLat = distCoord.get("Todos").get(0).first;
                parsLong = distCoord.get("Todos").get(0).second;
            }
            mapaCords = Radio.obtenerDistritosRadioMapa(distCoord,kms,parsLat,parsLong);
            Iterator<Map.Entry<String, Pair<Double, Double>>> iterator = mapaCords.entrySet().iterator();
            while(iterator.hasNext()){
                Map.Entry<String,Pair<Double, Double>> it = iterator.next();
                mSub.add(NetworkUtil.getRetrofit().getCountAlertasDistrito(it.getKey(),  count: true, cat)
                        .observeOn(AndroidSchedulers.mainThread())
                        .subscribeOn(io.reactivex.schedulers.Schedulers.io())
                        .subscribe(this::handleResponse, this::handleError));
            }
        }


        circle = map.addCircle(new CircleOptions()
                .center(new LatLng(parsLat, parsLong))
                .radius(kms*1000)
                .strokeColor(Color.BLUE)
                .fillColor(0x220000FF)
                .strokeWidth(5));

        // Set a listener for marker click.
        map.setOnInfoWindowClickListener(this);
        CameraPosition camera = CameraPosition.builder().target(circle.getCenter()).zoom(12).bearing(0).build();
        map.moveCamera(CameraUpdateFactory.newCameraPosition(camera));
    }
```

67- Figura 7.1.A-4.1 - Obtener distritos dentro del radio y ubicación

```
private void handleResponse(JsonArray pair) {
    JsonObject objeto;
    String distrito="";
    int total=0;
    for(JsonElement obj: pair){
        objeto  = obj.getAsJsonObject();
        distrito = objeto.get("_id").getAsString();
        total = objeto.get("total").getAsInt();
    }
    if(!distrito.equals("")) {
       if( contador < mapaCords.size()){
           Pair<String, Integer> p = new Pair<>(distrito, total);
           markerDistrito.add(p);
           contador++;
       }
    }else{
            contador++;
    }
    if(contador == mapaCords.size()){
        for (int i = 0; i < markerDistrito.size(); i++) {
            //Añadir marcador al mapa
            Pair<String, Integer> disCount = markerDistrito.get(i);
            String dis = disCount.first;
            Pair<Double, Double> arrayPar = mapaCords.get(dis);
            map.addMarker(new MarkerOptions().position(new LatLng(arrayPar.first, arrayPar.second)).title(dis)
                    .snippet("Se han encontrado " + disCount.second + " alertas"));
        }
    }
}
```

68- Figura 7.1.A-4.2 - Crear marcadores en el mapa

A-5: Editar configuración del usuario

La actividad '*ConfigActivity.java*' es la encargada de mostrar al usuario las opciones configurables para las alertas que este desea ver. Está compuesta principalmente por un '*seekbar*' para elegir la cantidad kilómetros que se desea de radio, una lista de '*checkbox*' para mostrar las categorías que la app ofrece (uno por cada categoría más otro por si se desea seleccionar 'Todas') y por último, un *switch* que permite activar o desactivar la ubicación. Si esta se encuentra desactivada, aparece un *spinner* que despliega la lista de distritos. Con el objetivo de optimizar el código, se ha dividido cada elemento en funciones distintas, que a su vez, puede que tengan funciones auxiliares.

```
private void initSwitch() {...}

@Override
public void onCheckedChanged(CompoundButton compoundButton, boolean isChecked) {...}

private void initSeekBar() {...}

private void initListCheckBox() {...}

private void initSpinner() {...}
```

69- Figura 7.1.A-5.1 - Funciones clase ConfigActivity

En esta clase se ha definido también una función que permite conocer si el usuario tiene activa la localización [32], y si no la tiene, cuestionar si desea activarla.

```
private void AlertNoGps() {...}

private void localizacion() {

    Location location;
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {

        if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && che

            return;
        } else {
            location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        }
    } else {
        location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
    }


    locationListener = new LocationListener() {...};

    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  minTime: 0,  minDistance: 0, locationListener);

}
```

70- Figura 7.1.A-5.2- Funciones para la localización

Esta clase también consta de varios atributos privados y se sirve de *SharedPreferences* [99]. Cada vez que el usuario selecciona una opción en cualquiera de las opciones configurables, esta se guarda en una variable para que estos cambios se muestren siempre hasta que se decida volver a modificarlos. Estos atributos son:

```
private int km;
private boolean isCheckedSw;
private int pos; // posicion del distrito
private boolean todasBool;
private boolean dyaBool;
private boolean terrBool;
private boolean crimiBool;
private boolean trafBool;
private boolean eventosBool;
private boolean transpBool;
private boolean contBool;
private String listaCategoria;
```

71- Figura 7.1.A-5.3 - Atributos en *SharedPreferences* en Configuración

Los distritos que engloban el radio se obtienen a través de una función que obtiene los distritos que hay dentro de la circunferencia formada por el punto de origen que se toma en cada caso y el radio elegido por el usuario. Para ello, se ha tenido en cuenta que la Tierra es una esfera no tratándola como si fuera plana. El punto de origen es la ubicación del usuario si ésta está activada o el marcador central del distrito que haya escogido en caso de que la ubicación esté inactiva. Esta función [97] viene implementada en la clase *'Radio'* y es usada cuando se muestran las alertas.

```
public static ArrayList<String> obtenerDistritosRadio(HashMap<String, ArrayList<Pair<Double, Double>>> coordenadas, int kms, D
    coordenadas = initCoord();
    ArrayList distRadio = new ArrayList<String>();
    boolean marcadorEncontrado;
    Iterator<Map.Entry<String, ArrayList<Pair<Double, Double>>>> iterator = coordenadas.entrySet().iterator();

    while (iterator.hasNext()) {
        Map.Entry<String, ArrayList<Pair<Double, Double>>> it = iterator.next(); // iterador del HashMap
        if(!it.getKey().equals("Todos") && !it.getKey().equals("General")) {
            ArrayList<Pair<Double, Double>> listAux = it.getValue(); // obtengo el arrayList del distrito que estoy iterando
            Iterator<Pair<Double, Double>> itArrayList = listAux.iterator(); // iterador del arrayList
            marcadorEncontrado = false;
            while (itArrayList.hasNext() && !marcadorEncontrado) {
                Pair<Double, Double> parAux = itArrayList.next();
                Double lat = parAux.first;
                Double longi = parAux.second;
                Double var = distanciaCoord(parsLat, parsLong, lat, longi);
                if (var <= kms) {
                    distRadio.add(it.getKey());
                    marcadorEncontrado = true; // en el momento en el que encuentre una coordenada dentro, ya no es necesario
                }
            }
        }
    }
    return distRadio;
}
```

72- Figura 7.1.A-5.4 - Obtener distritos en función del radio

A-6: Añadir alertas

Para añadir alertas en tiempo real, el usuario pulsa el *FloatingActionButton* con el símbolo '+' que hay en la esquina inferior derecha de la pestaña alertas. Al pulsarlo se abre una actividad en la que hay que rellenar todos los datos sobre la alerta que se quiere añadir.

Esta actividad tiene un formulario donde se introduce los datos de la persona que añade la alerta, el distrito, la categoría y el texto de la alerta.

Al intentar insertar la alerta se comprueba que todos los datos estén completos y si alguno de ellos no lo está, se muestra un mensaje al lado del campo que falta por rellenar.

También se comprueba que la categoría indicada sea la correcta, para ello se hace uso del clasificador de alertas y se realiza una petición a la api de MeaningCloud pasando la alerta que se quiere añadir, esto devuelve un json [73] que se parsea y se obtiene la categoría correspondiente. Si la categoría obtenida por la API no es igual a la que ha insertado el usuario se muestra un '*snackBar'* indicando el error.

Si todos los campos están completos y la categoría es la correcta, entonces se hace una petición POST a la api de NodeJS [76] para insertar la alerta en la base de datos. Si la petición se ha realizado correctamente se muestra un mensaje de que la alerta se ha añadido.

```
public String categoriaValida() throws IOException, SAXException, ParserConfigurationException {
    String api = "http://api.meaningcloud.com/class-1.1";
    String key = "05ed9a7c754aeee5d5f99470a756a5f8";
    String txt = alerta.getText().toString();
    String model = "news";
    Post post = new Post (api);
    post.addParameter( name: "key", key);
    post.addParameter( name: "txt", txt);
    post.addParameter( name: "model", model);
    post.addParameter( name: "of",  value: "xml");
    String response = post.getResponse();

    // Prints the specific fields in the response (categories)
    DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
    Document doc = docBuilder.parse(new ByteArrayInputStream(response.getBytes( charsetName: "UTF-8")));
    doc.getDocumentElement().normalize();
    Element response_node = doc.getDocumentElement();
    try {
        NodeList status_list = response_node.getElementsByTagName("status");
        Node status = status_list.item( index: 0);
        NamedNodeMap attributes = status.getAttributes();
        Node code = attributes.item( index: 0);
        if(!code.getTextContent().equals("0")) {
            System.out.println("Not found");
        } else {
            NodeList category_list = response_node.getElementsByTagName("category_list");
            if(category_list.getLength()>0){
                Node categories = category_list.item( index: 0);
                NodeList category = categories.getChildNodes();
                String output = "";
                for(int i=0; i<category.getLength(); i++) {
                    Node info_category = category.item(i);
                    NodeList child_category = info_category.getChildNodes();
                    String label = "";
                    String code_cat = "";
                    String relevance = "";
                    for(int j=0; j<child_category.getLength(); j++){
                        Node n = child_category.item(j);
                        String name = n.getNodeName();
                        if(name.equals("code"))
                            code_cat = n.getTextContent();
                        else if(name.equals("label"))
                            label = n.getTextContent();
                        else if(name.equals("relevance"))
                            relevance = n.getTextContent();
                    }
                    output += code_cat;
                }
                if(output.isEmpty())
                    System.out.println("Not found");
                else
                    return output;
            }
        }
    } catch (Exception e) {
        System.out.println("Not found");
    }
    return "Nada";
}
```

73- Figura 7.1.A-6.1 - Validación de categoría con el texto

74- Figura 7.1.A-6.2 - Añadir alertas

A-7: Contacto

La implementación de la pantalla de 'Contacto' se basta simplemente de un formulario que envía un correo a una cuenta establecida por defecto. El archivo *'ContactActivity.java'* recoge el código de esta funcionalidad. Al pulsar en el botón de 'Enviar', se llama a la actividad de Android *ACTION_SEND.* Se ha creado una función que comprueba que todos los campos están rellenados (Figura 7.1.A-7.1):

```
public boolean validate() {
    boolean valido = true;
    if(etName.getText().toString().isEmpty()) {
        valido = false;
        etName.setError("Introduce la nombre");
    }
    if(etEmail.getText().toString().isEmpty()) {
        valido=false;
        etEmail.setError("Introduce tu email");
    }
    if(etSubject.getText().toString().isEmpty()) {
        valido=false;
        etSubject.setError("Introduce tu asunto");
    }
    if(etBody.getText().toString().isEmpty()) {
        valido=false;
        etBody.setError("Introduce tu mensaje");
    }
    return valido;
}
```

75- Figura 7.1.A-7.1 - Validación campos rellenados

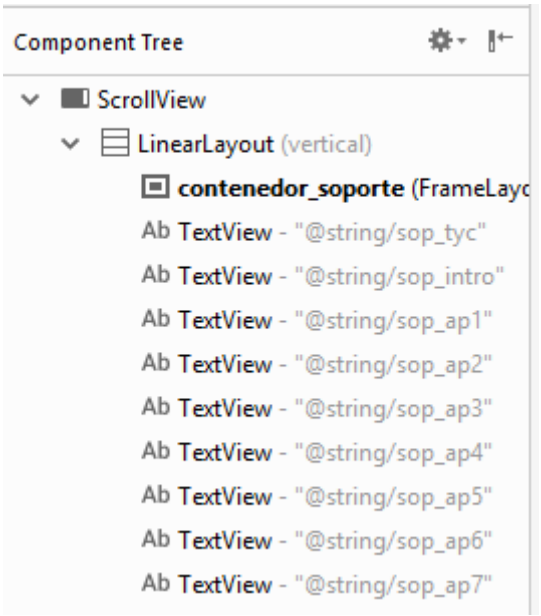
Las variables *etXXX* se tratan de *'EditText'* que referencian a los elementos creados en el archivo *'activity_contact.xml',* el cual es el encargado de proporcionar la interfaz a toda esta implementación.

A-8: Obtener información acerca de la aplicación. 'Soporte'

'SoporteActivity.java' lanza una actividad que muestra un conjunto de '*TextView'* que muestran información acerca de la aplicación al usuario. No realiza ninguna interacción con la base de datos y cumple una función meramente informativa.

```java
public class SoporteActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_soporte);

        FragmentTransaction ft = getFragmentManager().beginTransaction();
        ft.add(R.id.contenedor_soporte, new ActivitySoporte());
        ft.commit();
    }
```

76- Figura 7.1.A-8.1 - SoporteActivity



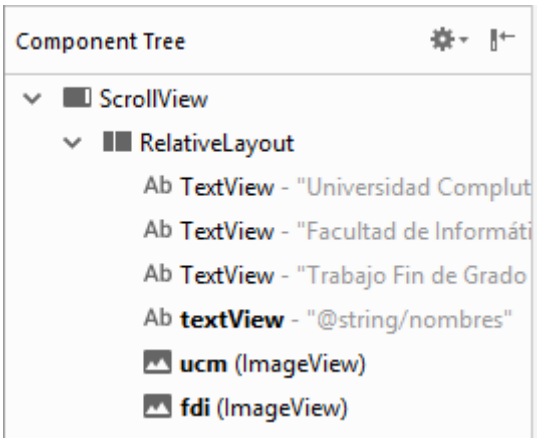
77- Figura 7.1.A-8.2 - Soporte

A-9: Obtener información sobre nosotros. ‘About us’

Del mismo modo que la funcionalidad anterior, *‘AboutUsActivity.java’* se limita a mostrar varios *‘TextView’* junto a dos elementos *‘ImageView’*.

```java
public class AboutUsActivity extends AppCompatActivity {

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

78- Figura 7.1.A-9.1 – AboutUsActivity



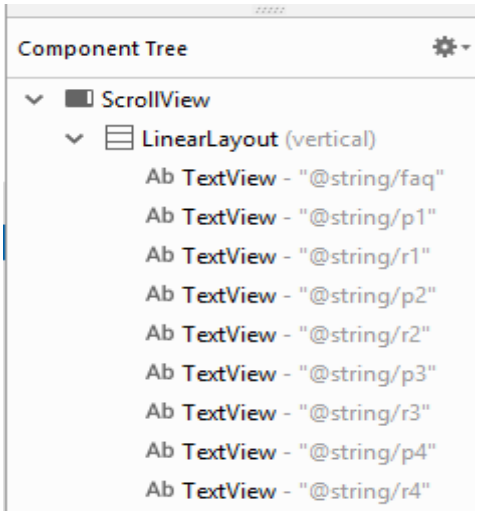
79- Figura 7.1.A-9.2 - About us

A-10: Resolver dudas acerca de la aplicación. 'FAQ'

De manera análoga a sus anteriores, *'FaqActivity.java'* contiene una serie de *'TextView'* responsables de mostrar el texto requerido.

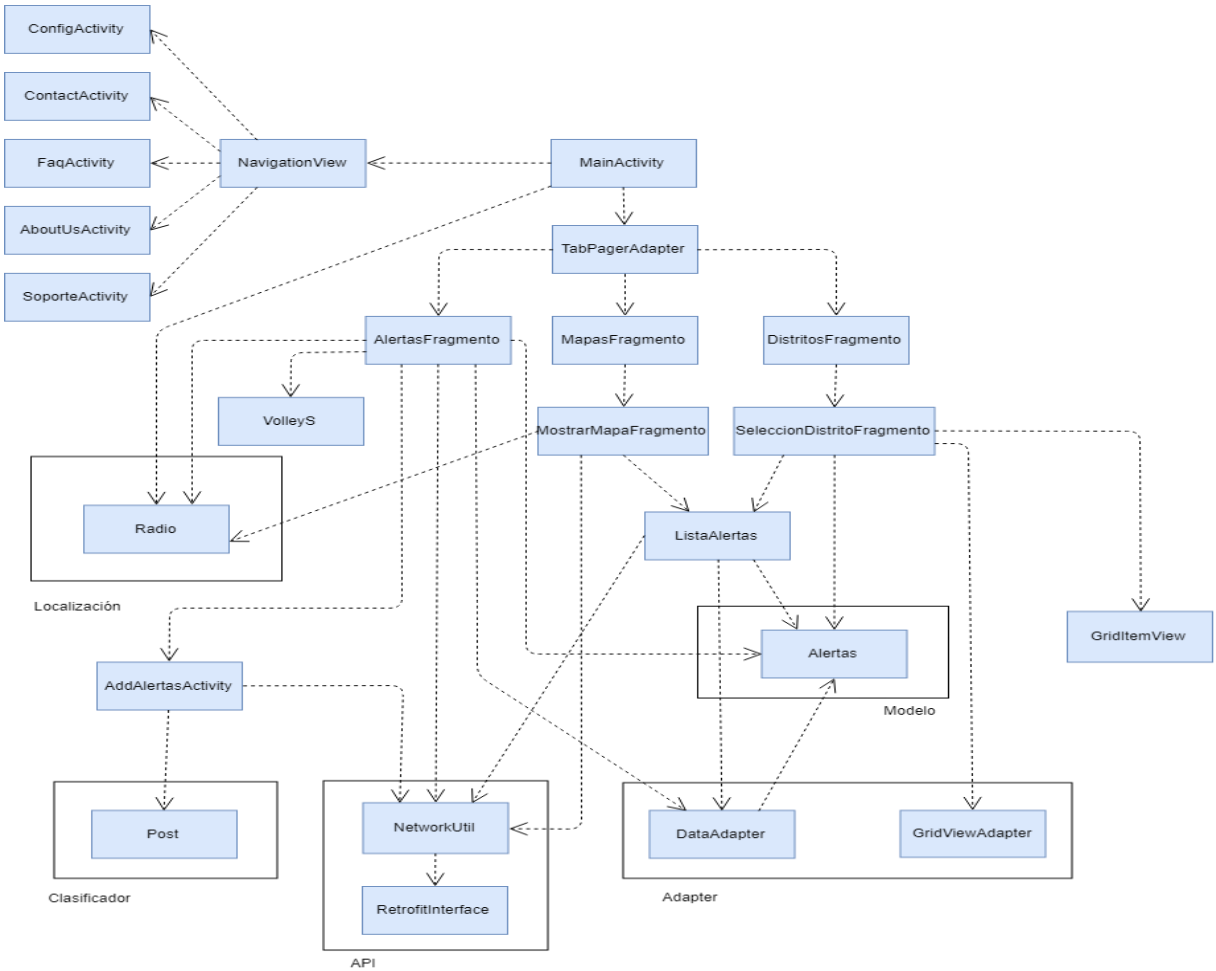
```java
public class FaqActivity extends AppCompatActivity {

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

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event){
        if(keyCode == KeyEvent.KEYCODE_BACK) {
            setResult(RESULT_OK);
            finish();
            startActivity(new Intent(getApplicationContext(), MainActivity.class));
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

80- Figura 7.1.A-10.1 - FAQActivity



81- Figura 7.1.A-10.2 - FAQ

En la siguiente figura se muestra el diagrama de clases completo de la aplicación Android.



82- Figura 7.1.2 - Diagrama de clases Android

7.2 IMPLEMENTACIÓN WEB

A continuación, se van a explicar la implementación de las funcionalidades de la web:

W-1: Ver alertas en tiempo real en Madrid.

En la página principal de la web se pueden visualizar en un mapa de Madrid los diferentes distritos que tienen alertas en las últimas horas. Se trata de información en tiempo real con la que se muestran marcadores en aquellos distritos que tienen alertas.

A continuación, se podrá ver cómo se inicializa el mapa (utilizando JavaScript[55]) (Figura 7.2.W-1.1 y 7.2.W-1.2) y cuál es el resultado final mostrado (Figura 7.2.W-1.3):

javaScriptMapa.js

```
function initMap() {
    var marcadores = [
        ['Arganzuela', 40.400861, -3.699350, document.getElementById("arganzuela").value], //Arganzuela
        ['Barajas', 40.4839402, -3.5701402, document.getElementById("barajas").value], //Barajas
        ['Carabanchel', 40.381607, -3.735203, document.getElementById("carabanchel").value], //Carabanchel
        ['Centro', 40.4169416, -3.7083759, document.getElementById("centro").value], //Centro
        ['Chamartín', 40.460367, -3.676567, document.getElementById("chamartin").value], //Chamartín
        ['Chamberí', 40.438656, -3.704180, document.getElementById("chamberi").value], //Chamberí
        ['Ciudad Lineal', 40.455531, -3.656119, document.getElementById("ciudadlineal").value], //Ciudad Lineal
        ['Fuencarral-El Pardo', 40.494289, -3.693477, document.getElementById("fuencarral").value], //Fuencarral-El Pardo
        ['Hortaleza', 40.485152, -3.634796, document.getElementById("hortaleza").value], //Hortaleza
        ['Latina', 40.387812, -3.773530, document.getElementById("latina").value], //Latina
        ['Moncloa-Aravaca', 40.443568,  -3.742829, document.getElementById("moncloa").value], //Moncloa-Aravaca
        ['Moratalaz', 40.407016, -3.644330, document.getElementById("moratalaz").value], //Moratalaz
        ['Puente de Vallecas', 40.386887, -3.658476, document.getElementById("puentevallecas").value], //Puente de Vallecas
        ['Retiro', 40.4101076, -3.6736514, document.getElementById("retiro").value], //Retiro
        ['Salamanca', 40.429807, -3.673778, document.getElementById("salamanca").value], //Salamanca
        ['San Blas-Canillejas', 40.436229, -3.599431, document.getElementById("sanblas").value], //San Blas-Canillejas
        ['Tetuán', 40.460158, -3.698835, document.getElementById("tetuan").value], //Tetuán
        ['Usera', 40.377026, -3.701982, document.getElementById("usera").value], //Usera
        ['Vicálvaro', 40.393974, -3.581134, document.getElementById("vicalvaro").value], //Vicálvaro
        ['Villa de Vallecas', 40.355089, -3.621192, document.getElementById("villavallecas").value], //Villa de Vallecas
        ['Villaverde', 40.345987, -3.693332, document.getElementById("villaverde").value] //Villaverde
    ];

    var map = new google.maps.Map(document.getElementById('mapa'),{
        zoom: 12,
        streetViewControl: false,
        scrollwheel: false,
        center: new google.maps.LatLng(40.422163, -3.689101),
    });

    var infowindow = new google.maps.InfoWindow(); // Abre ventana del marcador
```

83- Figura 7.2.W-1.1 - Código inicializar mapa

```
    var marker, i;
    var distritoM;
    for (i = 0; i < marcadores.length; i++) {
        if(marcadores[i][3] != 0) {  // Solo muestra los distritos que tienen alertas
            marker = new google.maps.Marker({
                position: new google.maps.LatLng(marcadores[i][1], marcadores[i][2]),
                map: map
            });
        }

        google.maps.event.addListener(marker, 'click', (function(marker, i) {
            return function() {
                if(marcadores[i][3] != 0) {
                    var contentString = '<form id="formulario_marcador" method="POST" action="alertas.php">' +
                    '<h2>' + marcadores[i][0] + '</h2>'+
                    '<input type="hidden" name="distritoM" value="'+ marcadores[i][0] +'" id="distritoM"/>' +
                    '<p> Se han encontrado <b>'+ marcadores[i][3] + ' alertas</b>. </p>' +
                    '<div class="form-center" style=" margin-left:25px; margin-top:20px">' +
                        '<button class="btn btn-info" type="submit">Ver alertas</button>' +
                    '</div>' +
                    '</form>';
                    infowindow.setContent(contentString);
                    infowindow.open(map, marker);
                }
            }
        })(marker, i));
    }

    google.maps.event.addDomListener(window, 'load', initMap);

}
```

84- Figura 7.2.W-1.2 - Código inicializar mapa – 2

85- Figura 7.2.W-1.3 - Mapa

Cada marcador contiene tanto el nombre del distrito como el número de alertas (Figura 7.2.W-1.4). Si se quiere ver en profundidad las alertas, cuenta con una opción de “Ver alertas” con la que se muestran las alertas de ese distrito ordenadas por fecha.

86- Figura 7.2.W-1.4 - Marcador Mapa

87- Figura 7.2.W-1.5 - Alertas por distrito

W-2: Mostrar alertas por distrito y categorías

En la misma página que la funcionalidad anterior, es decir, la página principal de la web, se encuentra la opción de buscar las alertas por el distrito y las categorías deseadas (Figura 7.2.W-2.1).

Con estos fragmentos de código se puede visualizar cómo se ha programado tanto la visualización de los distritos como la de las categorías:

```
<div class="form-group">
    <label class="col-sm-12">Selecciona un distrito</label>
    <div class="col-sm-12">
     <?php
         include ("claseAlertas.php");
         $alertas = new claseAlertas();
         $alertas->mostrarDistritos();
         $lista = $alertas->obtenerDatos();
         $alertas->crearCamposOcultosNumAlertas($lista);
     ?>
     </div>
 </div>
<div class="form-group">
        <label class="col-sm-12">Selecciona las categorías</label>
        <div class="items-collection">
            <div class="items col-xs-12 col-md-4 col-lg-3">
                <div class="info-block block-info clearfix">
                    <div data-toggle="buttons" class="btn-group
                    bizmoduleselect">
                        <label class="btn btn-default">
                            <div class="itemcontent">
                                <input type="checkbox" id="var_id[]" name="
                                var_id[]" autocomplete="off" value="
                                Desastres y accidentes">
                                 <h6>Desastres y accidentes</h6>
                             </div>
                        </label>
                    </div>
                </div>
            </div>

            <div class="items col-xs-12 col-md-4 col-lg-3">
                <div class="info-block block-info clearfix">
```

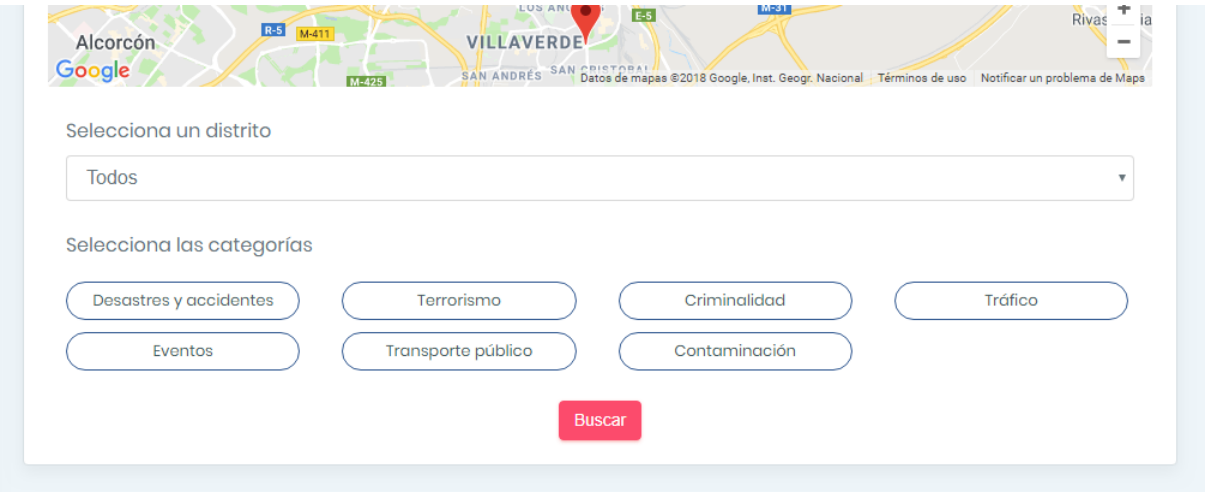
88- Figura 7.2.W-2.1 - Código búsqueda en index

claseAlertas.php

```
public function mostrarDistritos(){
  echo '<html>

  <select class="form-control form-control-line" name="distritos" id="distritos">
    <option>Todos</option>
    <option>Arganzuela</option>
    <option>Barajas</option>
    <option>Carabanchel</option>
    <option>Centro</option>
    <option>Chamartín</option>
    <option>Chamberí</option>
    <option>Ciudad Lineal</option>
    <option>Fuencarral-El Pardo</option>
    <option>Hortaleza</option>
    <option>Latina</option>
    <option>Moncloa-Aravaca</option>
    <option>Moratalaz</option>
    <option>Puente de Vallecas</option>
    <option>Retiro</option>
    <option>Salamanca</option>
    <option>San Blas-Canillejas</option>
    <option>Tetuán</option>
    <option>Usera</option>
    <option>Vicálvaro</option>
    <option>Villa de Vallecas</option>
    <option>Villaverde</option>
  </select>

   </html>';
}
```

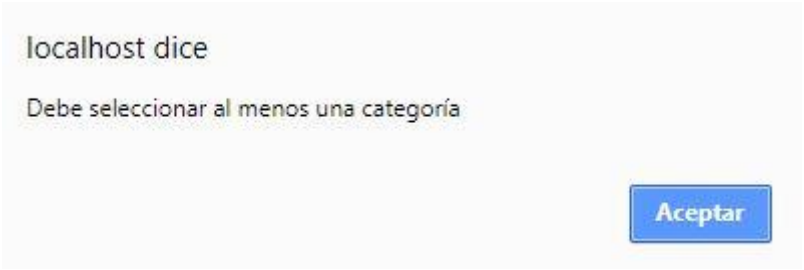
89- Figura 7.2.W-2.2 - Código mostrar distritos

Siendo el resultado:



90- Figura 7.2.W-2.3 - Búsqueda en index

Tras haber seleccionado el distrito y la/s categoría/s se debe pulsar el botón de “Buscar”. En caso de que se intente realizar la búsqueda sin haber seleccionado al menos una categoría se muestra el siguiente mensaje:



91- Figura 7.2.W-2.4 - Mensaje error “al menos una categoría”

La función de JavaScript que se encarga de realizar dicha comprobación es la siguiente:

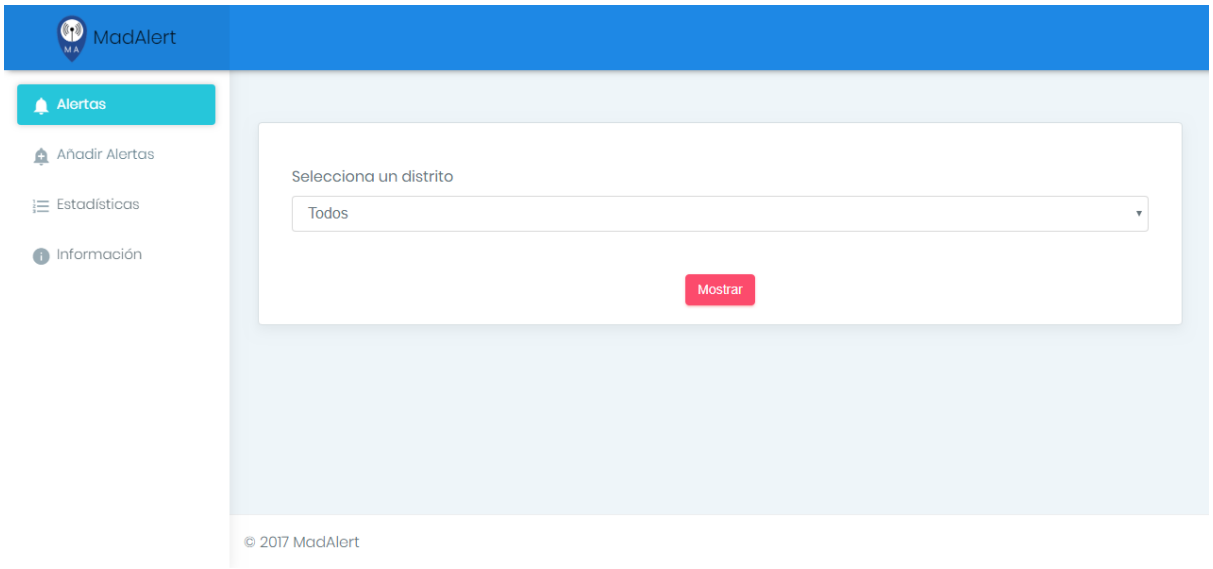
javaScript.js

```
//Para validar el registro
function valida() {
    var suma =0;
    var ok = true;
    var msg = "Debe seleccionar al menos una categoría\n";
    $("input:checkbox:checked").each(function() {
            suma++;
        });
        if(suma == 0){
            ok=false;
        }
    if(ok == false){
        alert(msg);
    }
    return ok;
}
```

92- Figura 7.2.W-2.5 - Código función validación categorías

W-3: Mostrar alertas por distrito

Similar a la funcionalidad anterior pero sólo se debe seleccionar el distrito deseado. Es decir, se van a mostrar todas las alertas de ese distrito sin tener en cuenta categorías.

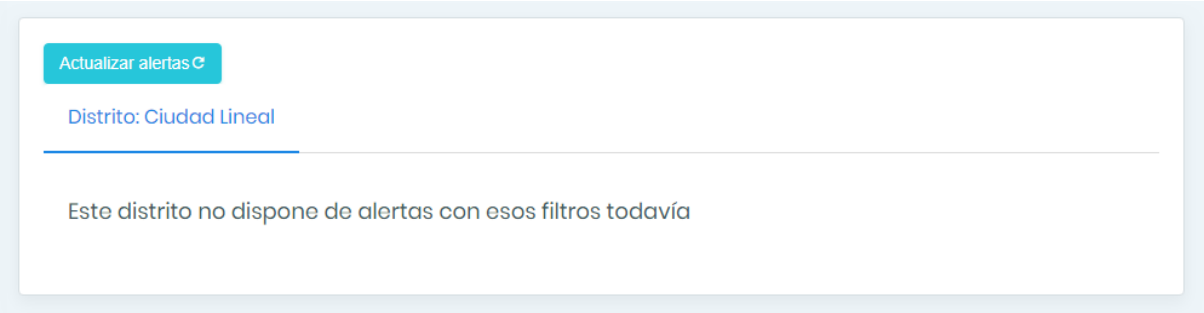


93- Figura 7.2.W-3.1 - Buscar alertas por distrito

Una vez seleccionado el distrito y habiendo pulsado el botón de “Mostrar” se muestran las alertas de dicho distrito ordenadas cronológicamente (Figura 7.2.W-3.2).

94- Figura 7.2.W-3.2 - Mostrar alertas por distrito

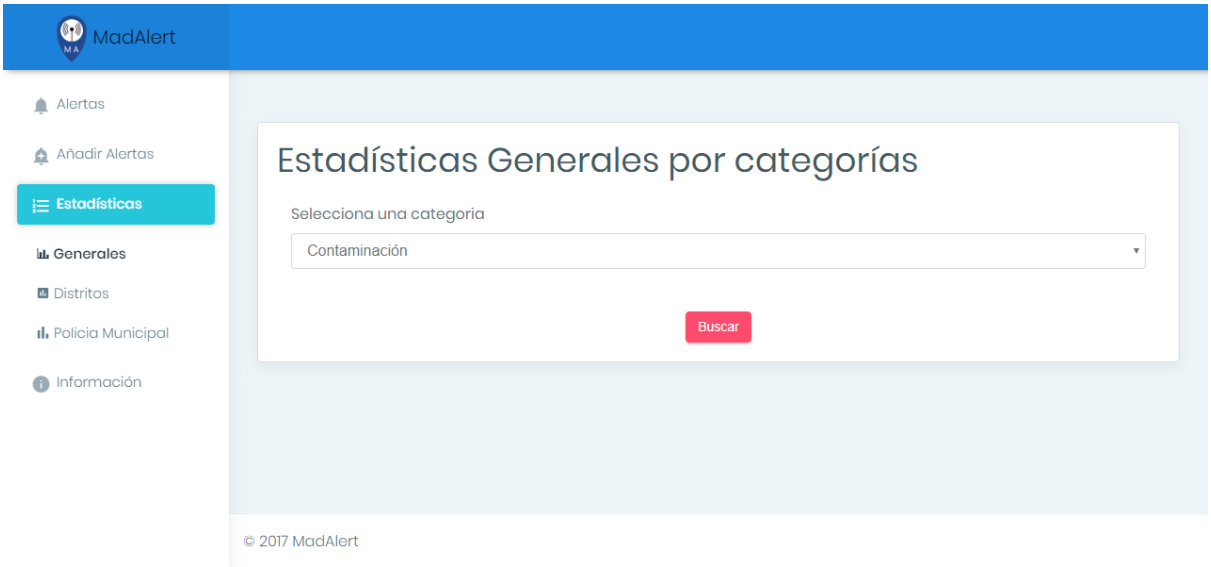
Si el distrito seleccionado no dispone de alertas se muestra un mensaje como el de la Figura 7.2.W-3.3:



95- Figura 7.2.W-3.3 - Mensaje no hay alertas

W-4: Consulta de estadísticas generales

En esta sección se visualizan las estadísticas generales, es decir, a partir de los datos obtenidos durante los dos meses anteriores se hace un análisis y unos gráficos de barras según la categoría seleccionada por el usuario.



96- Figura 7.2.W-4.1 - Buscador estadísticas generales

estadisticasGenerales.php

```php
<?php

    if(isset($_POST['categorias'])){
      $categoria = $_POST['categorias'];
      include ("claseEstadisticas.php");
      $estadisticas = new claseEstadisticas();

      //Obtengo la categoría
      echo ' <input type="hidden" name="categoria" value='.$categoria.' id="categoria"/>';
      echo ' <div class="row">
              <div class="col-12">
                  <div class="card">
                          <div class="card-block form-center">';
                              echo '<h2> Estadísticas de la categoría: '.$categoria .' </h2>';
                          echo ' </div>
                      </div>
              </div>
          </div>';
```

97- Figura 7.2.W-4.2 - Código estadísticas generales

Se obtienen las alertas según la categoría y el mes. Para obtener el mes se utiliza la siguiente función de JavaScript:

```php
    public function obtenerMes($i){
      $mesAnterior1 = date('m', strtotime('-1 month'));
      $mesAnterior2 = date('m', strtotime('-2 month'));
      if($i==1){
        $mes = $mesAnterior1;
      }
      else{
        $mes = $mesAnterior2;
      }
      return $mes;
    }
}
?>
```

98- Figura 7.2.W-4.3 - Código función obtener mes

Y para obtener las estadísticas de dichos meses se va a llamar a la siguiente función que se encuentra en la clase Estadísticas:

```php
public function obtenerEstGenerales($categoria, $mes){
  $totalArganzuela= $this->daoEst->obtenerEstadisticas("Arganzuela", $categoria, $mes);
  $totalBarajas= $this->daoEst->obtenerEstadisticas("Barajas", $categoria, $mes);
  $totalCarabanchel= $this->daoEst->obtenerEstadisticas("Carabanchel", $categoria, $mes);
  $totalCentro= $this->daoEst->obtenerEstadisticas("Centro", $categoria, $mes);
  $totalChamartin= $this->daoEst->obtenerEstadisticas("Chamartín", $categoria, $mes);
  $totalChamberi= $this->daoEst->obtenerEstadisticas("Chamberí", $categoria, $mes);
  $totalCiudadLineal= $this->daoEst->obtenerEstadisticas("CiudadLineal", $categoria, $mes);
  $totalFuencarral= $this->daoEst->obtenerEstadisticas("Fuencarral", $categoria, $mes);
  $totalGeneral= $this->daoEst->obtenerEstadisticas("General", $categoria, $mes);
  $totalHortaleza= $this->daoEst->obtenerEstadisticas("Hortaleza", $categoria, $mes);
  $totalLatina= $this->daoEst->obtenerEstadisticas("Latina", $categoria, $mes);
  $totalMoncloa= $this->daoEst->obtenerEstadisticas("Moncloa", $categoria, $mes);
  $totalMoratalaz= $this->daoEst->obtenerEstadisticas("Moratalaz", $categoria, $mes);
  $totalPuenteVallecas= $this->daoEst->obtenerEstadisticas("Puente Vallecas", $categoria, $mes);
  $totalRetiro= $this->daoEst->obtenerEstadisticas("Retiro", $categoria, $mes);
  $totalSalamanca= $this->daoEst->obtenerEstadisticas("Salamanca", $categoria, $mes);
  $totalSanBlas= $this->daoEst->obtenerEstadisticas("San Blas", $categoria, $mes);
  $totalTetuan= $this->daoEst->obtenerEstadisticas("Tetuan", $categoria, $mes);
  $totalUsera= $this->daoEst->obtenerEstadisticas("Usera", $categoria, $mes);
  $totalVicalvaro= $this->daoEst->obtenerEstadisticas("Vicálvaro", $categoria, $mes);
  $totalVillaVallecas= $this->daoEst->obtenerEstadisticas("Villa de Vallecas", $categoria, $mes);
  $totalVillaverde= $this->daoEst->obtenerEstadisticas("Villaverde", $categoria, $mes);
  $lista = [$totalArganzuela, $totalBarajas, $totalCarabanchel, $totalCentro, $totalChamartin, $totalChamberi,
    $totalCiudadLineal, $totalFuencarral, $totalGeneral, $totalHortaleza, $totalLatina, $totalMoncloa, $
    totalMoratalaz, $totalPuenteVallecas, $totalRetiro, $totalSalamanca, $totalSanBlas, $totalTetuan, $
    totalUsera, $totalVicalvaro, $totalVillaVallecas, $totalVillaverde];
  return $lista;
}
```

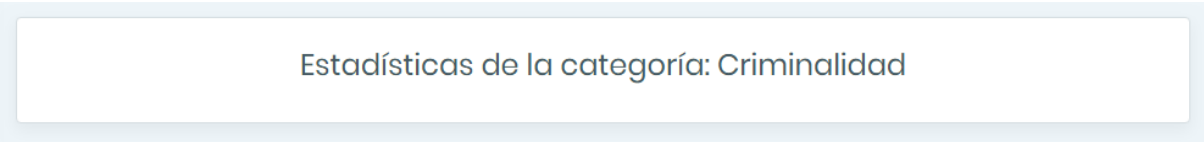
99- Figura 7.2.W-4.4 - Código obtener est. generales

Esta función se encarga de llamar a otra función que se encuentra dentro del daoEstadísticas:

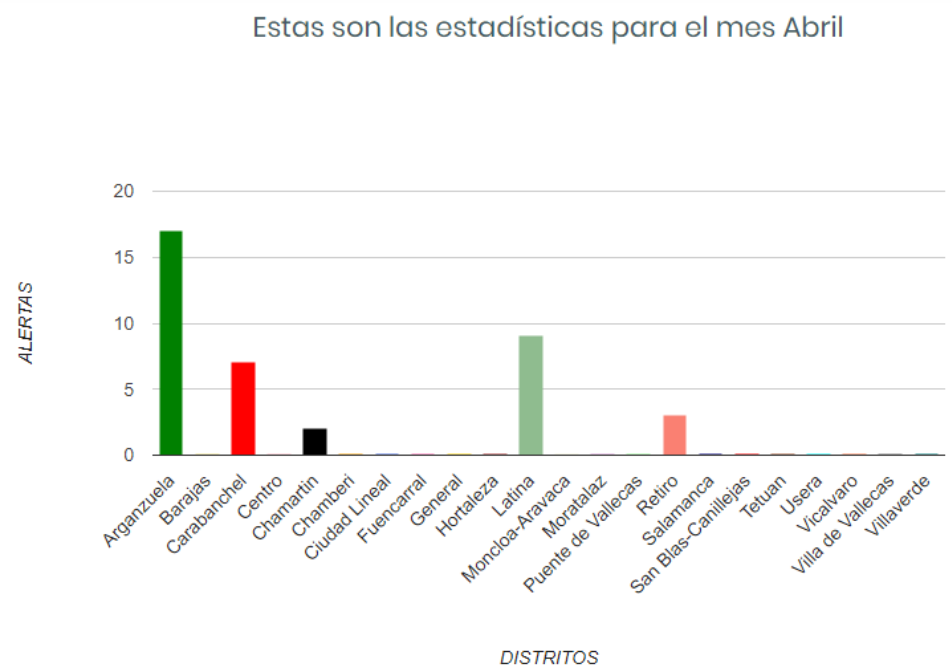
```php
//Devuelve el numero total de alertas en un distrito con una categoria dada
public function obtenerEstadisticas($distrito, $categorias, $mes) {
    $documento = $this->conEst->count(['distrito' => $distrito, 'categoria'=> $categorias , 'mes' => $mes]);
    return $documento;
  }
```

100- Figura 7.2.W-4.5 - Código obtener est. generales DAO

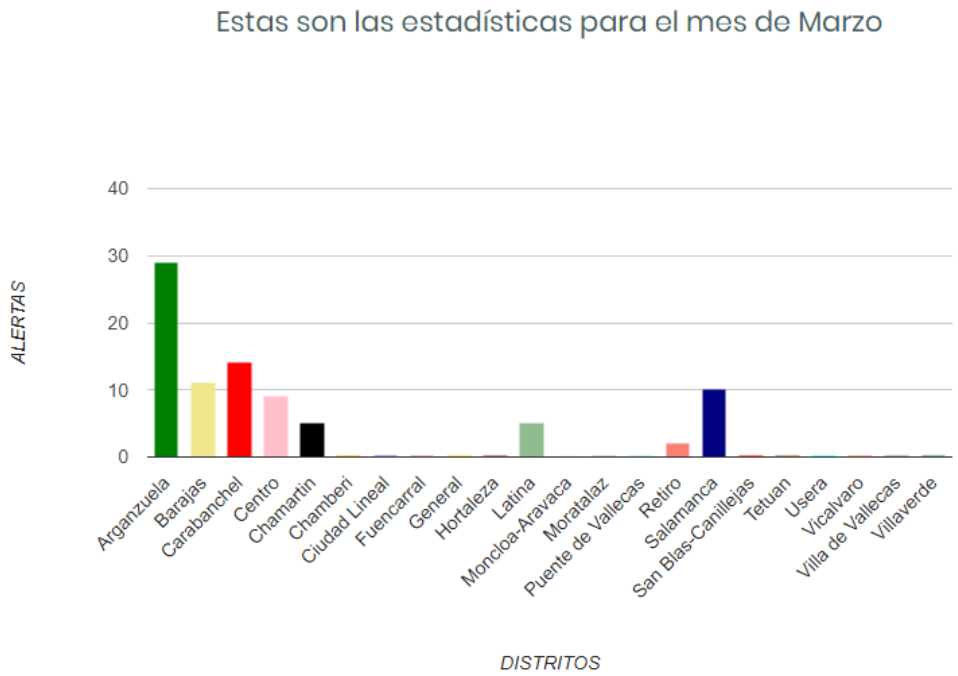
El gráfico de barras que se muestra a continuación contiene todos los distritos con las alertas de la categoría indicada.



101- Figura 7.2.W-4.6 - Estadísticas según una categoría concreta

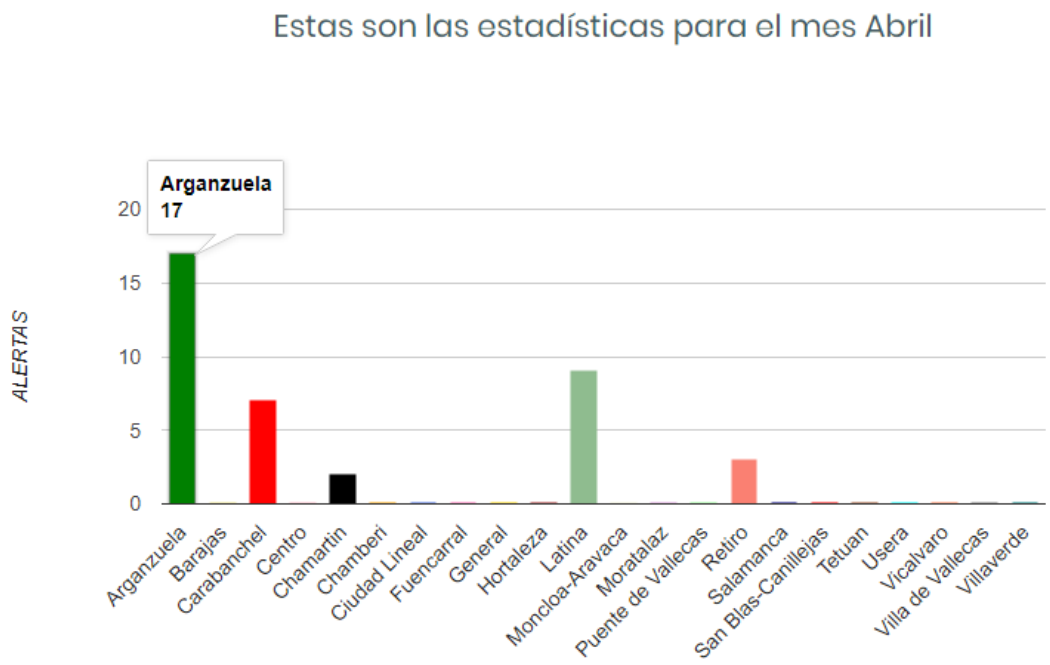


102- Figura 7.2.W-4.7 - Estadísticas generales mes de Abril



103- Figura 7.2.W-4.8 - Estadísticas generales mes de Marzo

Además, pasando el ratón por encima de las barras de la gráfica, se muestra el nombre del distrito y el número exacto de alertas que tiene de esta categoría (Figura 7.2.W-4.9):



104- Figura 7.2.W-4.9 - Estadísticas generales cursor encima

Si la categoría escogida por el usuario no tiene datos registrados de esos meses se muestra un mensaje informativo acerca de ello.

```
public function noHayEstadisticasGenerales($lista){
 if($lista==0){
   return true;
 }
 else{
   return false;
 }
}
```

105- Figura 7.2.W-4.10 - Función comprobación estad. generales

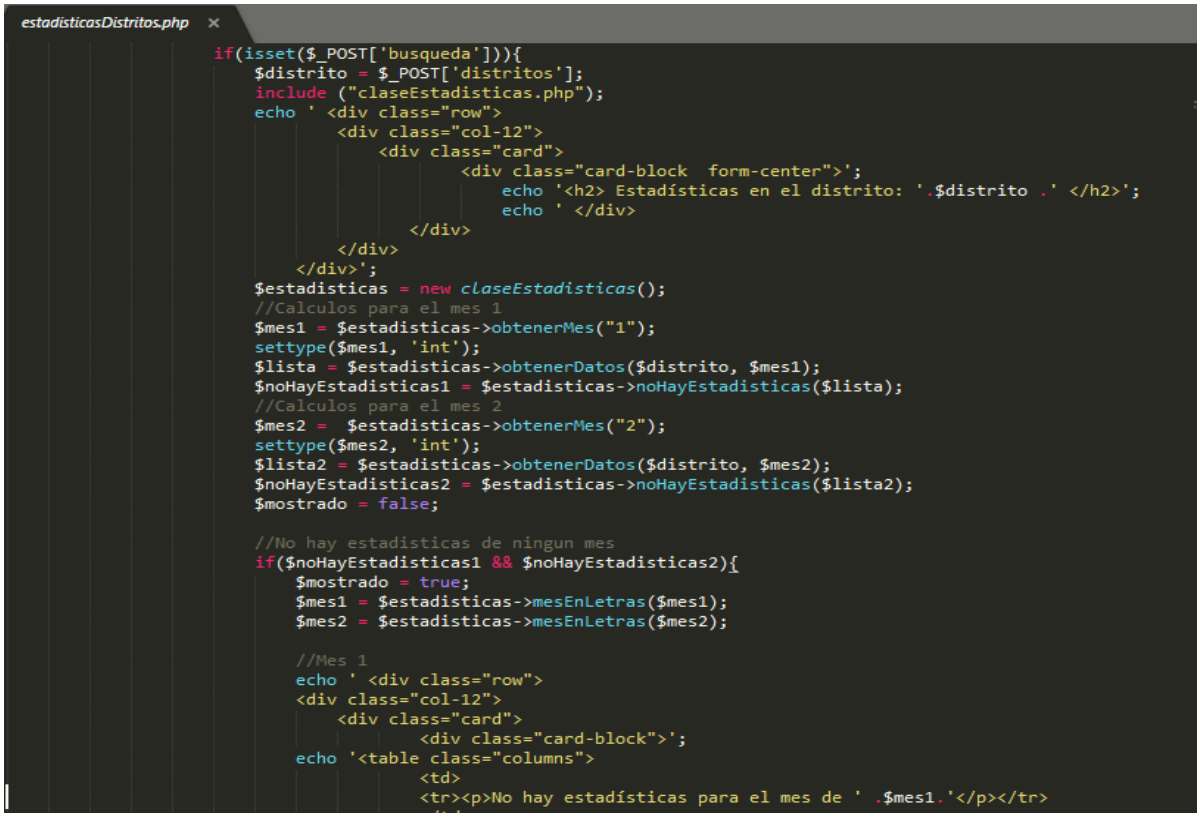
estadisticasGenerales.php

```php
// Si no hay estadísticas de ninguno de los dos meses
if ($noHayEstadisticasMes1 && $noHayEstadisticasMes2){
     $mes1 = $estadisticas->mesEnLetras($mes1);
     $mes2 = $estadisticas->mesEnLetras($mes2); // Pongo el mes con letras y no con números

    echo ' <div class="row">
    <div class="col-12">
        <div class="card">
                <div class="card-block">';
    echo '<table class="columns">
                <td>
                <tr><p>No hay estadísticas de '.$categoria.' para el mes de ' .$mes2.'</p><
                    /tr>
                </td>
            </table>';
     echo ' </div>
              </div>
        </div>
    </div>';


    echo ' <div class="row">
    <div class="col-12">
        <div class="card">
                <div class="card-block">';
    echo '<table class="columns">
                <td>
                <tr><p>No hay estadísticas de '.$categoria.' para el mes de ' .$mes1.'</p><
                    /tr>
                </td>
            </table>';
    echo ' </div>
              </div>
        </div>
    </div>';

}
```

106- Figura 7.2.W-4.11 - Código de la página de estad. generales

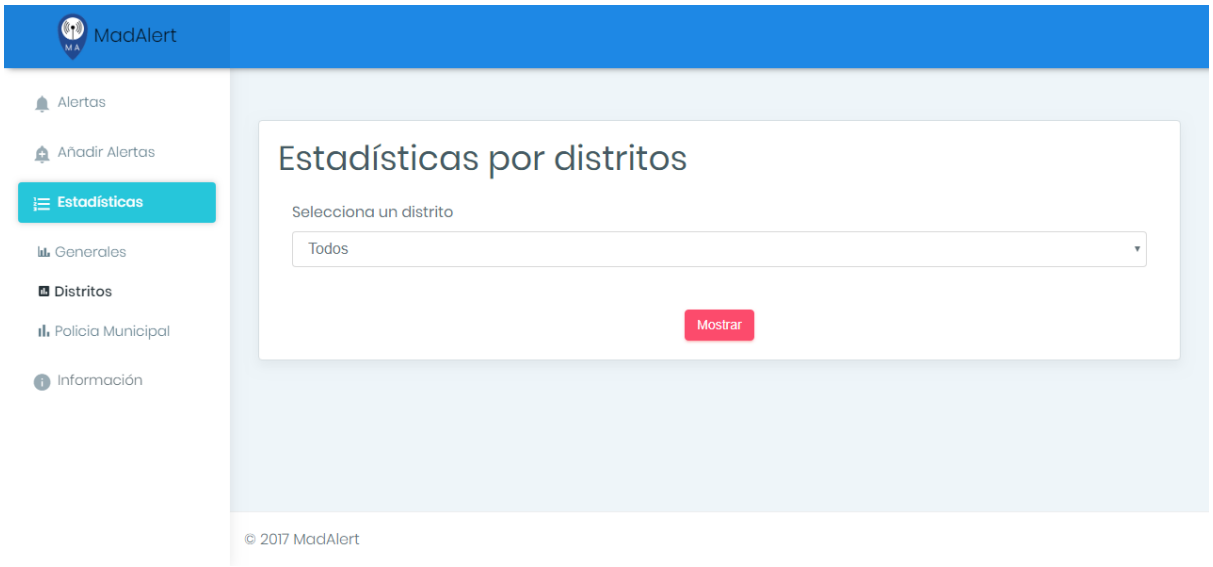
W-5: Consulta de estadísticas por distrito

Si se pulsa en la pestaña de estadísticas por distritos, aparece un buscador para seleccionar el distrito, del que se hace un resumen y análisis en forma de gráfico circular sobre los datos obtenidos de los últimos dos meses.



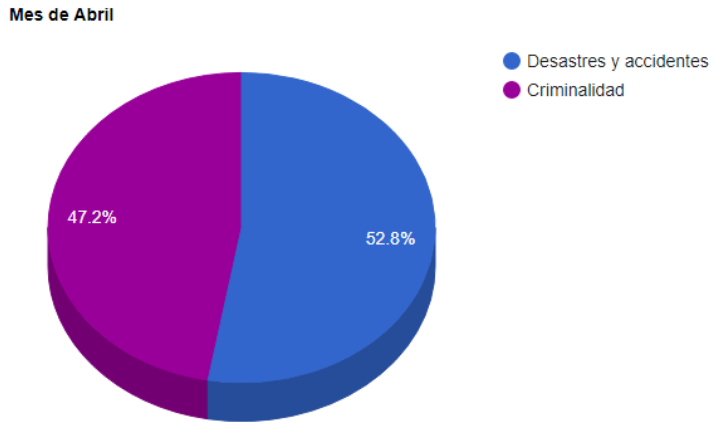
```php
if(isset($_POST['busqueda'])){
    $distrito = $_POST['distritos'];
    include ("claseEstadisticas.php");
    echo ' <div class="row">
            <div class="col-12">
                <div class="card">
                        <div class="card-block  form-center">';
                            echo '<h2> Estadísticas en el distrito: '.$distrito .' </h2>';
                            echo ' </div>
                    </div>
            </div>
        </div>';
    $estadisticas = new claseEstadisticas();
    //Calculos para el mes 1
    $mes1 = $estadisticas->obtenerMes("1");
    settype($mes1, 'int');
    $lista = $estadisticas->obtenerDatos($distrito, $mes1);
    $noHayEstadisticas1 = $estadisticas->noHayEstadisticas($lista);
    //Calculos para el mes 2
    $mes2 =  $estadisticas->obtenerMes("2");
    settype($mes2, 'int');
    $lista2 = $estadisticas->obtenerDatos($distrito, $mes2);
    $noHayEstadisticas2 = $estadisticas->noHayEstadisticas($lista2);
    $mostrado = false;

    //No hay estadisticas de ningun mes
    if($noHayEstadisticas1 && $noHayEstadisticas2){
        $mostrado = true;
        $mes1 = $estadisticas->mesEnLetras($mes1);
        $mes2 = $estadisticas->mesEnLetras($mes2);

        //Mes 1
        echo ' <div class="row">
        <div class="col-12">
            <div class="card">
                    <div class="card-block">';
        echo '<table class="columns">
                    <td>
                    <tr><p>No hay estadísticas para el mes de ' .$mes1.'</p></tr>
```

107- Figura 7.2.W-5.1 - Código de la página de estad. por distritos

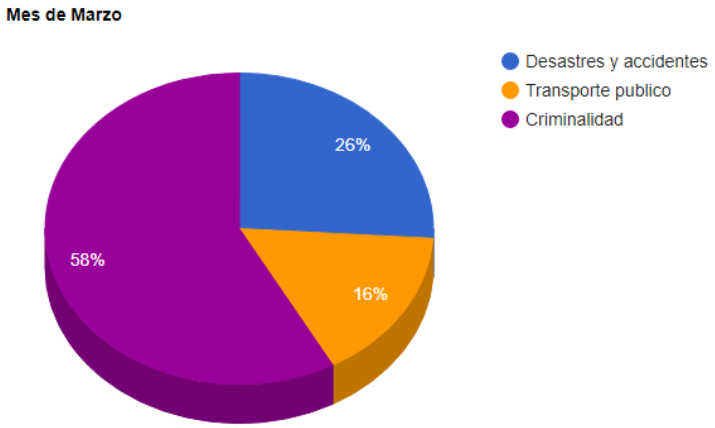


108- Figura 7.2.W-5.2 - Buscador estadísticas por distritos

109- Figura 7.2.W-5.3 - Estadísticas según un distrito concreto



110- Figura 7.2.W-5.4 - Estadísticas por distrito mes de Abril



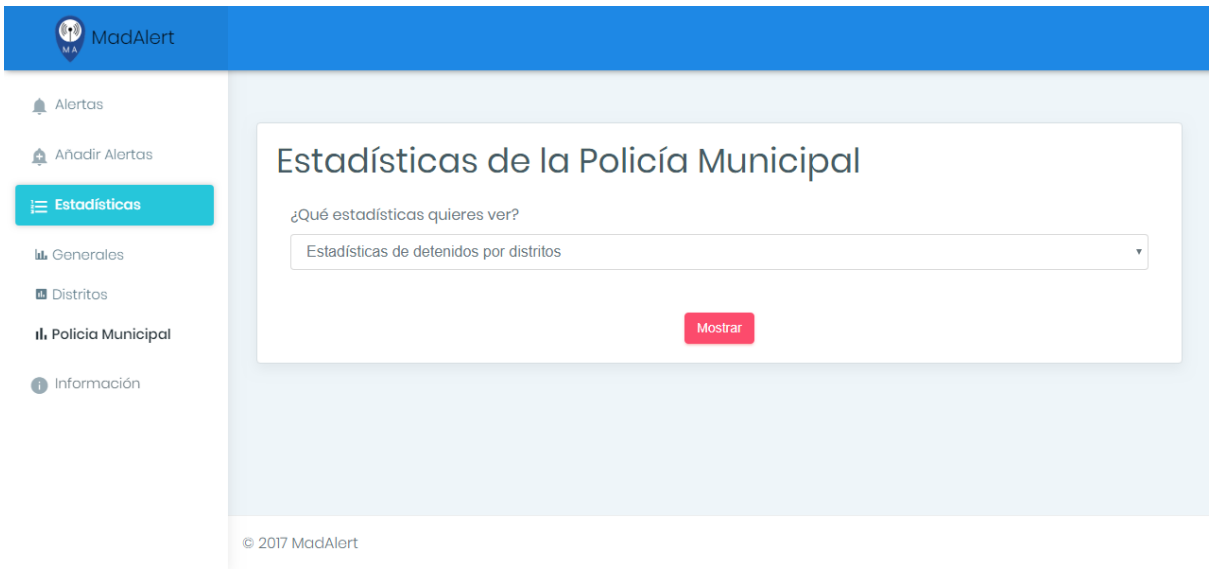
111- Figura 7.2.W-5.5 - Estadísticas por distrito del mes de Marzo

Si el distrito escogido por el usuario no tiene datos registrados de esos meses se muestra un mensaje informativo acerca de ello (Figura 7.2.W-5.6).

112- Figura 7.2.W-5.6 - Mensaje no hay estadísticas

W-6: Consulta de estadísticas de la Policía Municipal[23]

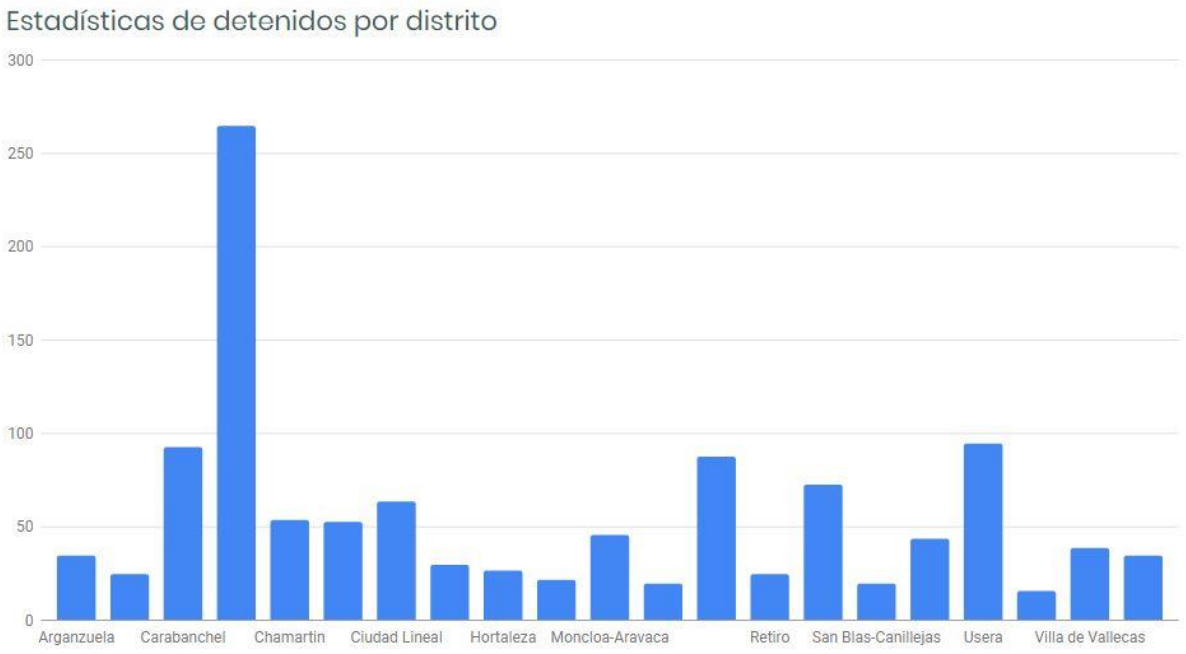
En la pestaña de estadísticas de la Policía Municipal de Madrid se pueden consultar gráficos relacionados con la seguridad de los distritos, los accidentes que hay por distrito y los detenidos por distrito. Estos gráficos se construyen mensualmente a través de los datos analizados de la página de datos del Ayuntamiento de Madrid[23], esta página sube mensualmente un Excel del mes anterior con datos ocurridos en Madrid. En la web se mostrarán las estadísticas del mes correspondiente a dos meses anteriores respecto a la fecha actual.



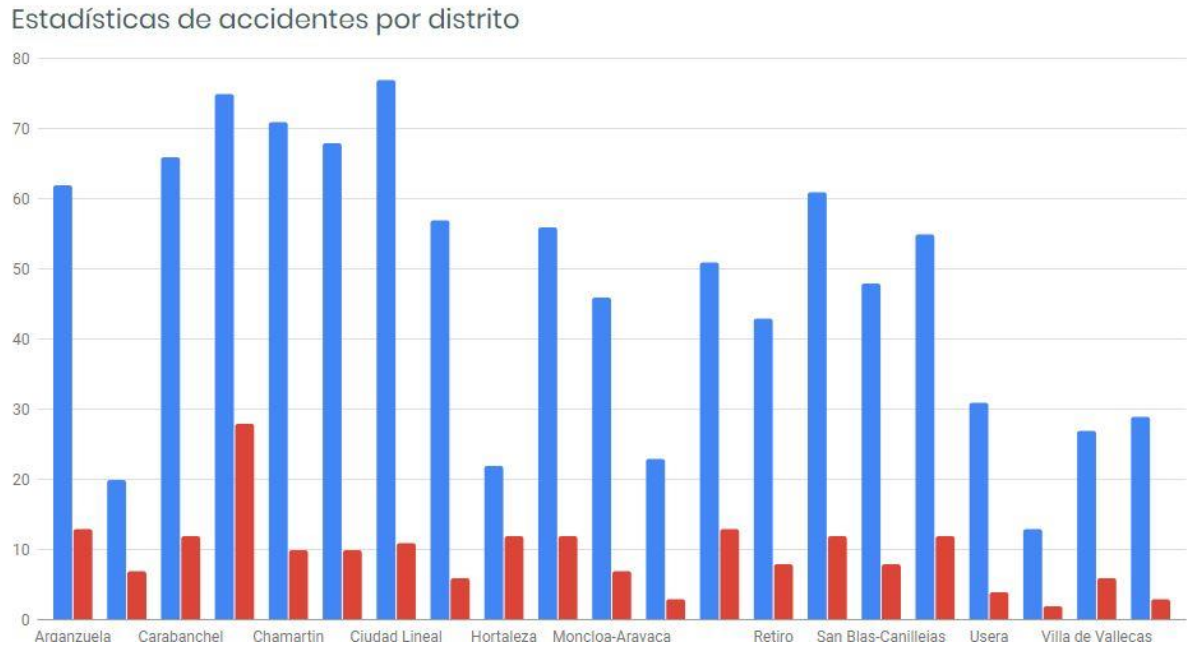
113- Figura 7.2.W-6.1 - Buscador estadísticas Policía

Consultar estadísticas policiales de accidentes o detenidos

Al seleccionar cualquiera de las opciones se muestra un gráfico de barras, pero en la opción de estadísticas relacionadas con la seguridad es necesario seleccionar un distrito y se muestra un gráfico circular con las estadísticas correspondientes.



114- Figura 7.2.W-6.2 - Estadísticas de detenidos por distrito



115- Figura 7.2.W-6.3 - Estadísticas de accidentes por distrito

116- Figura 7.2.W-6.4 - Estadísticas relacionadas con la seguridad

Si alguna opción no dispone de estadísticas aparece un mensaje informativo.

W-7: Añadir alertas

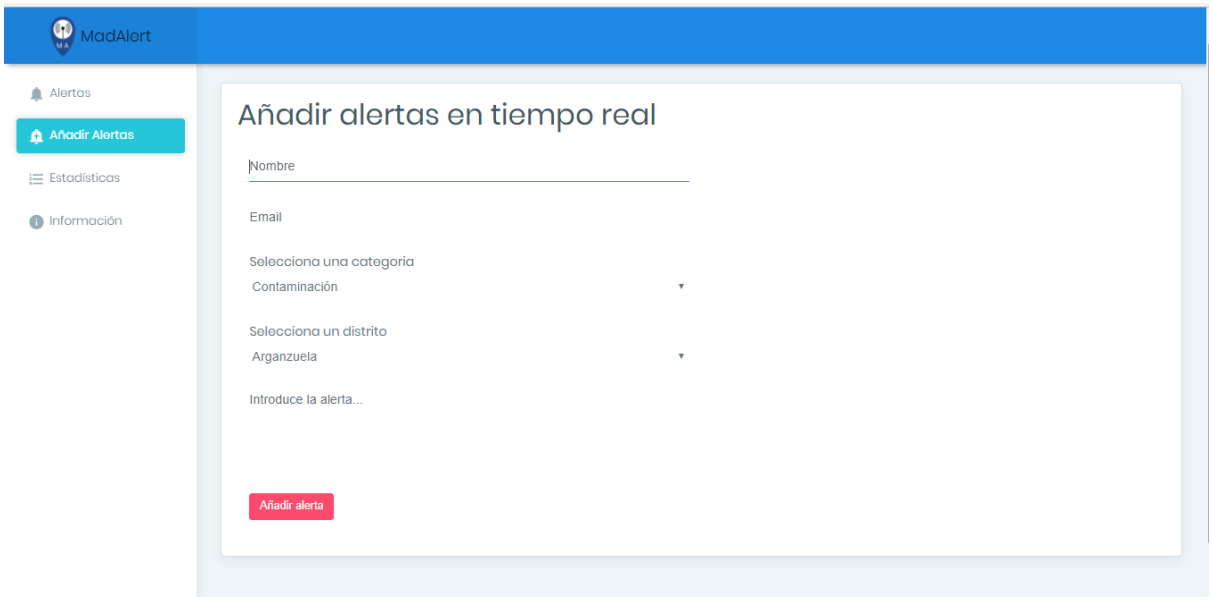
Otra de las funcionalidades de la Web es el alta de una alerta en tiempo real por parte de los usuarios. Cuenta con un formulario en el que el usuario tiene que introducir los datos de identificación, así como la alerta, la categoría y el distrito al que pertenece. Todos los campos han de estar rellenados para que se pueda realizar la inserción de alertas.

Las alertas que añaden los usuarios cuentan con un campo adicional en la base de datos, para que de cara a la visualización de las alertas se sepa que no está verificada.

aniadirAlertas.php

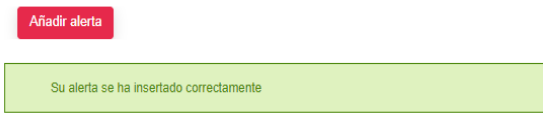
```
<div class= card-block >
    <h1 class="card-title"> Añadir alertas en tiempo real </h1>

    <form class="form-horizontal form-material" class="contacto" action="
    procesarAniadir.php" method="post">
        <div class="form-group">
            <div class="col-md-6">
                <input type="text" placeholder="Nombre" class="form-control
                form-control-line" name="nombre" id="nombre" required/>
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-6">
                <input type="email" placeholder="Email" class="form-control
                form-control-line" name="email" id="email" required/>
            </div>
        </div>
        <div class="form-group">
            <label class="col-md-6">Selecciona una categoria</label>
            <div class="col-md-6">
                <?php
                include ("claseAlertas.php");
                $alertas = new claseAlertas();
                $alertas->mostrarCategorias();
                ?>
            </div>
        </div>
        <div class="form-group">
            <label class="col-md-6">Selecciona un distrito</label>
            <div class="col-md-6">
                <?php
                $alertas->mostrarDistritos();
                ?>
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-6">
                <textarea rows="5" placeholder="Introduce la alerta..."
                class="form-control form-control-line" name="alerta" id="alerta"
                required></textarea>
```

117- Figura 7.2.W-7.1 - Código aniadirAlertas.php

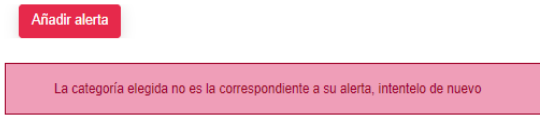


118- Figura 7.2.W-7.2 - Formulario añadir alertas

Para que esta funcionalidad tenga más credibilidad, se comprueba cuando el usuario inserta la alerta que esta corresponde a la categoría indicada por el usuario. Para ello se usará el algoritmo de clasificación de alertas (Figura 7.2.W-7.6) y si coinciden las categorías se añade a la base de datos (Figura 7.2.W-7.3). En caso contrario, se muestra un mensaje de error (Figura 7.2.W-7.4) y no se inserta.



119- Figura 7.2.W-7.3 - Mensaje alerta insertada



120- Figura 7.2.W-7.4 - Mensaje error al insertar alerta

procesarAniadir.php

```php
<?php
include ("claseAlertas.php");
include ("Clasificador.php");
$alertas = new claseAlertas();

$nombre = $_POST['nombre'];
$email = $_POST['email'];
$categoria = $_POST['categorias'];
$distrito = $_POST['distritos'];
$alerta = $_POST['alerta'];
?>

<?php

$clasificador = new Clasificador();
$sol = $clasificador->obtenerClasificacion($alerta);

if($sol!=false && $sol==$categoria){
    $alertas->insertarAlerta($nombre, $categoria, $distrito, $alerta);
    echo"1";
}
else{
    echo"0";?>
</form>
<?php
}
?>
```

121- Figura 7.2.W-7.5 - Procesar añadir alerta

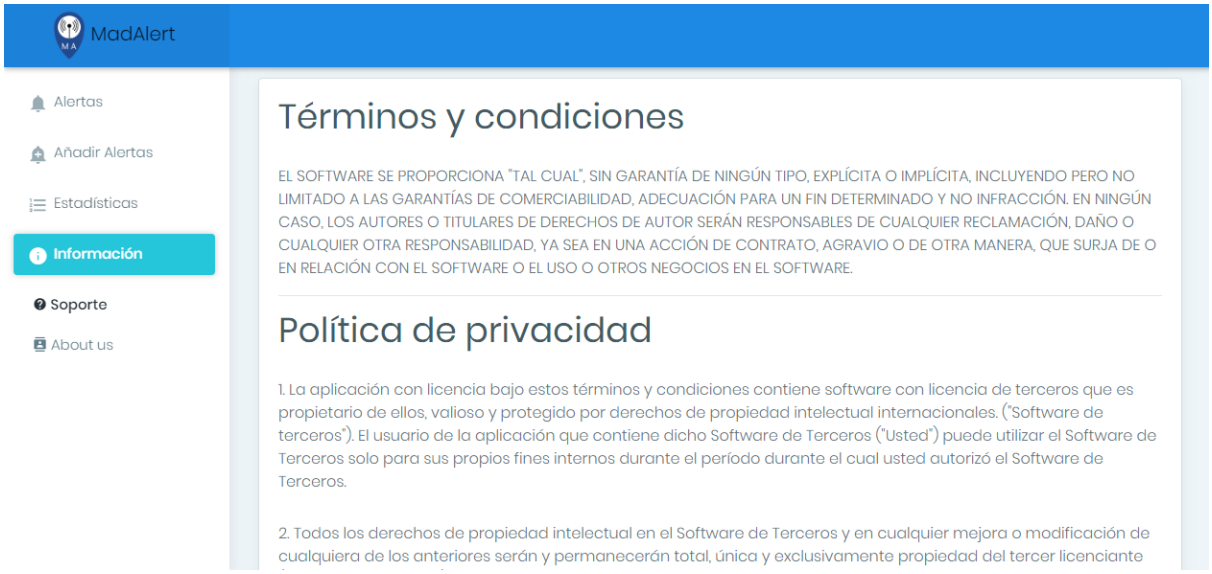
Clasificador.php

```php
<?php
require 'vendor/autoload.php';
class Clasificador {
    public $api =
    public $key =
    public $model = "news";

    private function obtenerResponse($api, $key, $model, $txt) {
      $data = http_build_query(array('key'=>$key,
                                     'model'=>$model,
                                     'txt'=>$txt,
                                     'src'=>'sdk-php-1.1')); // management internal parameter
      $context = stream_context_create(array('http'=>array(
            'method'=>'POST',
            'header'=>
              'Content-type: application/x-www-form-urlencoded'."\r\n".
              'Content-Length: '.strlen($data)."\r\n",
            'content'=>$data)));

      $fd = fopen($api, 'r', false, $context);
      $response = stream_get_contents($fd);
      fclose($fd);
      return $response;
    }

    public function obtenerClasificacion($alerta){
        $response = $this->obtenerResponse($this->api, $this->key, $this->model, $alerta);
        $json = json_decode($response, true);
        if(isset($json['category_list']) && count($json['category_list'])>0) {
          $i=0;
          foreach($json['category_list'] as $categorie) {
                $label = $categorie['label'];
                return $label;
          }
        }else{
          return false;
        }
    }
```

122- Figura 7.2.W-7.6 - Clasificador alertas

W-8: Información de la Web

En esta funcionalidad aparecen las opciones de “Soporte” y “About us”. En la primera se puede ver la información legal de la aplicación y sus términos de uso.



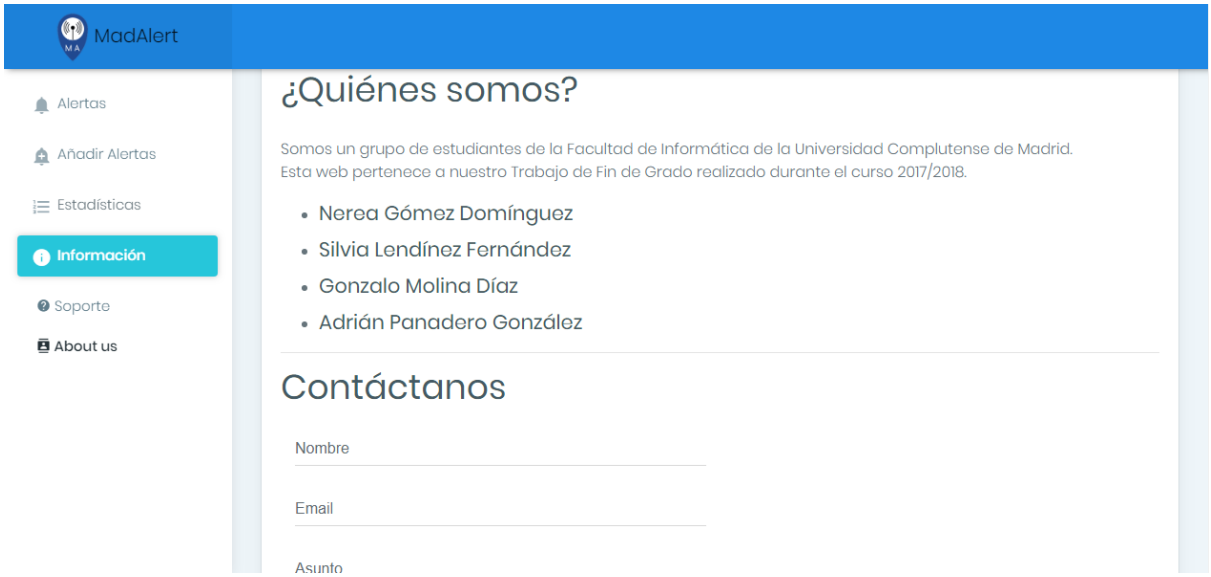
123- Figura 7.2.W-8.1 - Soporte

En esta otra pestaña se visualiza una breve información sobre los integrantes del grupo. Además, cuenta con un formulario para enviar cualquier duda o sugerencia sobre la aplicación web.

aboutus.php

```
<div class="card">
    <div class="card-block">
        <h1 class="card-title"> ¿Quiénes somos? </h1>
        <p class="m-t-30"> Somos un grupo de estudiantes de la Facultad de Informática de
        la Universidad Complutense de Madrid. </br> Esta web pertenece a nuestro Trabajo
        de Fin de Grado realizado durante el curso 2017/2018.
        </p>
            <ul>
                <li> <h3> Nerea Gómez Domínguez </h3> </li>
                <li> <h3> Silvia Lendínez Fernández </h3> </li>
                <li> <h3> Gonzalo Molina Díaz </h3> </li>
                <li> <h3> Adrián Panadero González </h3> </li>
            </ul>
        <hr>
        <h1 class="card-title"> Contáctanos </h1>
        <form class="form-horizontal form-material" class="contacto" action="
        mailto:madalert17@gmail.com" method="post">
            <div class="form-group">
                <div class="col-md-6">
                    <input type="text" placeholder="Nombre" class="form-control
                    form-control-line" name="nombre" id="nombre" required/>
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-6">
                    <input type="email" placeholder="Email" class="form-control
                    form-control-line" name="email" id="email" required/>
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-6">
                    <input type="text" placeholder="Asunto" class="form-control
                    form-control-line" name="asunto" id="asunto" required/>
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-6">
                    <textarea rows="5" placeholder="Introduce el mensaje..."
                    class="form-control form-control-line" name="Mensaje" required/></
```

124- Figura 7.2.W-8.2 - Código About us



125- Figura 7.2.W-8.3 - About us

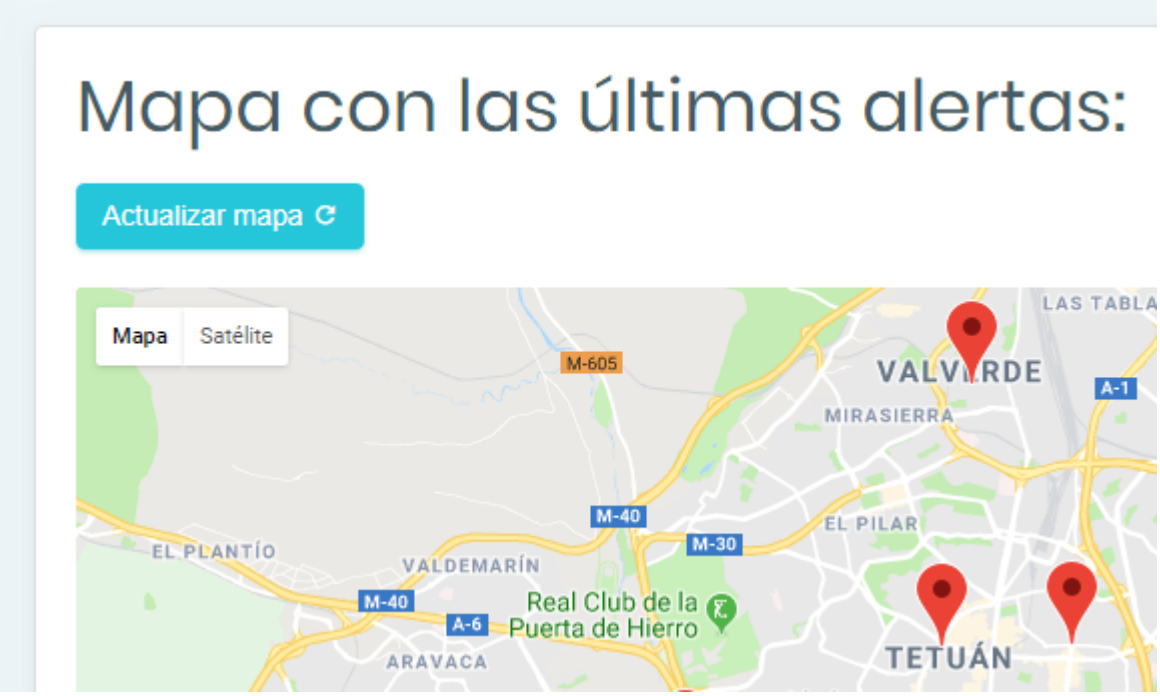
W-9: Actualizar mapa

La actualización del mapa consiste en que aparezcan los marcadores antiguos y nuevos de los distritos que tengan alertas y el número de alertas actual de cada distrito.

index.php

```
<div class="row">
    <div class="col-12">

        <script type="text/javascript">
            function reFresh()
            location.reload(true)
            }
            /* Establece el tiempo 1 minuto = 60000 milliseconds. */
            window.setInterval("reFresh()",100);
        </script>

            <div class="card">
                <div class="card-block">
                    <h1 class="card-title"> Mapa con las últimas alertas: </h1>
                    <div id="mapaMargin">

                        <button class="btn btn-success" onclick="javascript:window.location.reload(
                            );" > Actualizar mapa <i class="mdi mdi-refresh"></i></button>

                    </div>

                    <div id="mapa" class="gmaps"></div>
                    <form method="post" action="alertas.php" id="buscarAlertas" onsubmit="return
                        valida()">
```

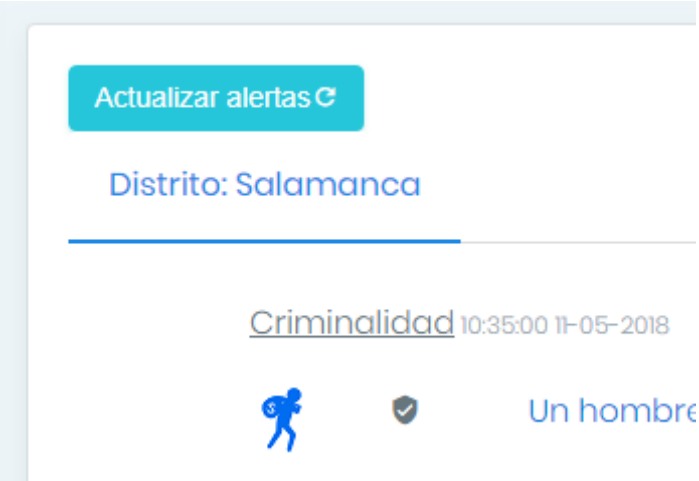
126- Figura 7.2.W-9.1 - Código actualizar mapa (index)



127- Figura 7.2.W-9.2 - Botón actualizar mapa (index)

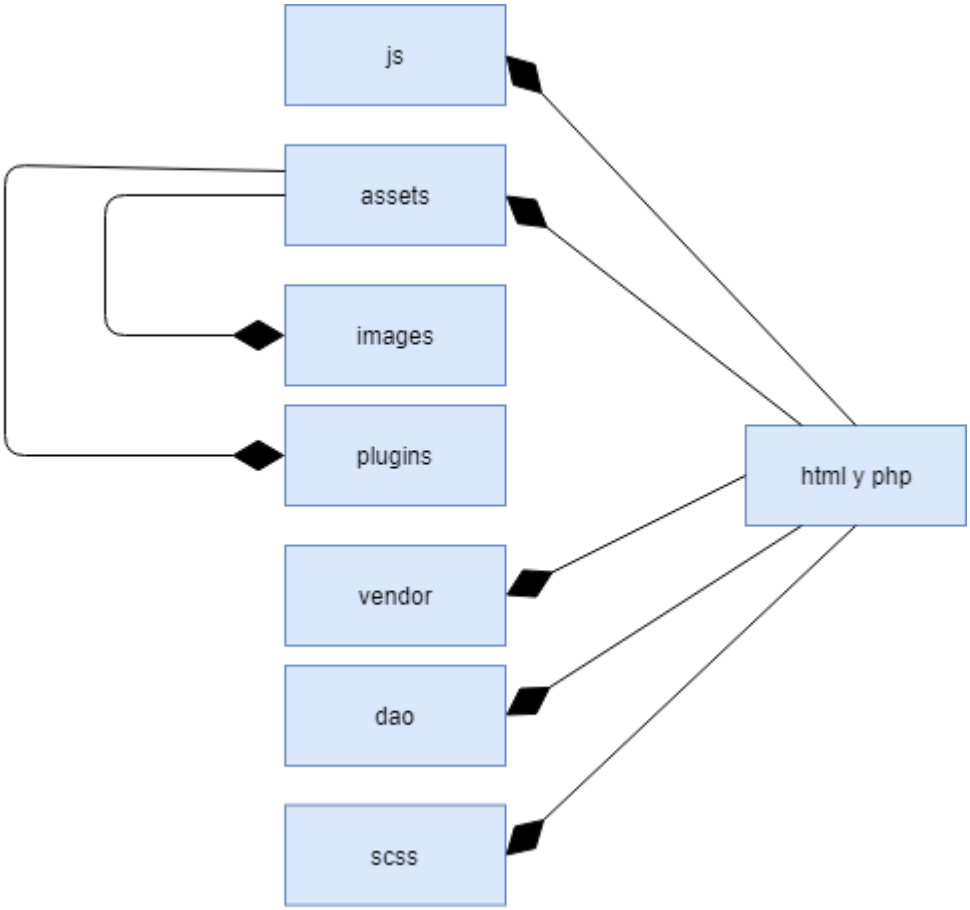
W-10: Actualizar alertas por distritos y/o categoría

Esta funcionalidad se encarga de actualizar la búsqueda de alertas por distrito y/o categoría seleccionado anteriormente por el usuario. Al actualizar se muestran tanto las nuevas alertas como las anteriores con los filtros elegidos antes.



128- Figura 7.2.W-10.1 - Botón actualizar alertas

En la siguiente figura, como si de un diagrama de clases se tratase, se muestra un esquema para explicar y simplificar la estructura de la aplicación web:



129- Figura 7.2.2 - Diagrama de clases de la aplicación web

7.3 IMPLEMENTACIÓN API Y RECOLECTOR DE DATOS

7.3.1 IMPLEMENTACIÓN API REST

Ya se explicó en apartados anteriores que la API REST[10] disponible en el sistema sirve para conectar la aplicación Android con la base de datos, tanto para realizar peticiones POST como GET.

La API ha sido implementada con JavaScript[55] mediante el entorno de ejecución de NodeJS[76]. NodeJS actúa como servidor entre la aplicación Android y la base de datos MongoDB[68]. Para que NodeJS[76] se pueda conectar, relacionar y mapear los modelos de datos de la BD de MongoDB[68] se ha usado el módulo mongoose.

Por lo tanto, se ha creado un esquema que corresponde a la colección de Alertas, en este esquema se define la forma del documento que se quiere obtener de MongoDB[68] y el tipo de datos con los que se trabaja.

```
'use strict';

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

var alertasSchema = new mongoose.Schema({
    alerta: String,
    fecha: Date,
    url: String,
    distrito: String,
    categoria: String,
    fuente: String,
    verificado: Boolean
});

mongoose.Promise = global.Promise;
mongoose.connect('mongodb://MadAlert:tfg20172018@ds235388.mlab.com:35388/noticias');

//return models
module.exports = mongoose.model('alertas', alertasSchema);
```

130- Figura 7.3.1.1 - Esquema de alertas

Se han usado el Framework de express[30] para poder crear el servidor y los métodos de enrutamiento de los que hace uso la aplicación, es decir las llamadas HTTP (POST y GET en este caso). El servidor creado escucha del puerto XXX del ordenador en el que se ejecuta la aplicación.

```
module.exports = router => {
    router.get('/', (req, res) => res.end('Bienvenido a MadAlert !'));

    //Para buscar alertas por distrito
    router.get('/alertas/:distrito', (req,res) => {
            alertasDistrito.getAlertasDistrito(req.params.distrito.split(','))
            .then(result => res.json(result))
            .catch(err => res.status(err.status).json({ message: err.message }));
    });

    //Para buscar alertas por distrito y categoria
    router.get('/alertas/:distrito/:arr', (req,res) => {
            alertasDistritoCategoria.getAlertasDistritoCategoria(req.params.distrito.split(','), req.params.arr.split(','))
            .then(result => res.json(result))
            .catch(err => res.status(err.status).json({ message: err.message }));
    });

    //Para buscar alertas por distrit
    router.get('/alertas/:distrito/:count/:categorias', (req,res) => {
            countAlertasDistrito.getCountAlertasDistrito(req.params.distrito.split(','),req.params.count, req.params.categorias.split(','))
            .then(result => res.json(result))
            .catch(err => res.status(err.status).json({ message: err.message }));
    });


    //Para añadir alertas
    router.post('/alertas/:titulo/:distrito/:fuente/:categoria', (req, res) => {
        const alerta = req.params.titulo;
        const distrito = req.params.distrito;
        const fuente = req.params.fuente;
        const categoria = req.params.categoria;
        addAlerta.addAlertaBD(req.params.titulo,req.params.distrito,req.params.fuente,req.params.categoria)
        .then(result => {
                res.status(result.status).json({ message: result.message })
        })
        .catch(err => res.status(err.status).json({ message: err.message }));
    });
}
```

131- Figura 7.3.1.2 - Definición de rutas del node.js

Las rutas definidas en la imagen superior recogen los parámetros que les llegan. En los casos en los que el parámetro recibido puede ser un array se hace un split para que sea

más fácil tratarlo en las funciones definidas. Los script que realizan las consultas y añadido en base datos están programados en JavaScript[55], todos ellos hacen peticiones a la base de datos de MongoDB[68] y devuelven el resultado esperado o un mensaje de error si se produce un fallo.

El lenguaje utilizado para intercambiar datos desde la API hacia la aplicación Android es JSON.[73]

La correspondencia entre clases, recursos implementados y URLS son los siguientes:

Clase/Recurso	URL	GET	POST	DELETE	PUT
Clase alertas	/api/v1/alertas	X	X		
Alertas por distrito	/api/v1/alertas/<distrito>	X			
Alertas por distrito y categoría	/api/v1/alertas/<distrito>/<cat>	X			
Número de alertas por distrito y/o categorias	/api/v1/alertas/<distrito>/<count>/<categorias>	X			
Añadir alerta	/api/v1/alertas/<alerta>/<distrito>/<fuente>/<cat>		X		

132- Figura 7.3.1.3 - Tabla de clases y recursos de la API

Si estas urls se quieren probar en el navegador se puede hacer accediendo a la siguiente url: http://ip:puerto/URL

La implementación de los recursos de la clase alertas se describe a continuación:

- **Alertas por distrito:** realiza una petición GET que tiene que recibir por parámetro el nombre del distrito del que se quieren buscar las alertas. Este también puede ser un string separado por comas, por lo que antes de llamar a la función **getAlertasDistrito(distrito)** se hace un split para tener en un array los distritos sobre los que se quieren obtener las alertas. En la función se comprueba si el distrito es igual a Todos, en este caso se hace una búsqueda en MongoDB[68] para obtener todas las alertas de Madrid, en caso contrario se buscan las alertas del distrito o los distritos indicados por el usuario.

```
'use strict';

const alerta = require('../alertas');

exports.getAlertasDistrito = distrito =>

    new Promise((resolve, reject) => {

        if( distrito == "Todos"){
            alerta.find().sort({fecha:-1})

            .then((alertas) => {resolve(alertas);})

            .catch(err => reject({status: 500, message: 'Internal Server Error! verAlertasDistrito.js'}))
        }
        else{
            alerta.find({distrito: {$in:distrito}}).sort({fecha:-1})

            .then((alertas) => {resolve(alertas);})

            .catch(err => reject({status: 500, message: 'Internal Server Error! verAlertasDistrito.js'}))
        }


    });
```

133- Figura 7.3.1.4 - Función de la API para obtener las alertas por distrito

- **Alertas por distrito y categoría:** se hace una petición GET que recibe por parámetro el distrito o distritos y las categorías de las alertas que se quieren recoger. Tanto como el parámetro distrito como categorías pueden ser un string separados por comas, por eso antes de llamar a la función **getAlertasDistritoCategoria(distrito, categoría)** se hace un split para tener ambas cosas en arrays y así la búsqueda en Mongodb resulte más simple. En la implementación de esta función en primer lugar se comprueba si el distrito es Todos y si es así se hace una búsqueda filtrando solo por categorías, si el distrito es diferente de “Todos” la búsqueda se hace con filtro por categorías y distritos.

```
'use strict';

const alerta = require('../alertas');

exports.getAlertasDistritoCategoria = (distrito, categorias) =>


        new Promise((resolve, reject) => {
            if( distrito == "Todos"){
                alerta.find({categoria:{$in:categorias}}).sort({fecha:-1})
                .then((alertas) => {resolve(alertas);})

                .catch(err => reject({status: 500, message: 'Internal Server Error! verAlertasDistritoCategoria.js'}))
            }
            else{
                alerta.find({distrito: {$in:distrito}, categoria:{$in:categorias}}).sort({fecha:-1})
                .then((alertas) => {resolve(alertas);})

                .catch(err => reject({status: 500, message: 'Internal Server Error! verAlertasDistritoCategoria.js'}))
            }
        });
```

134- Figura 7.3.1.5 - Función que devuelve las alertas por distrito y categoría

- **Número de alertas por distrito y/o categorías**: se ejecuta una petición GET para obtener el número de alertas correspondientes a las categorías indicadas que hay en un distrito. Por ello se recibe por parámetro el valor del distrito y de las categorías, siempre se hace un split al string que llega para convertirlo en un array y así realizar la cuenta de alertas de una forma más sencilla. En la implementación de la función **getCountAlertasDistritos(distritos,count, categorías)** se comprueba si

el distrito es igual a Todos, si es así se hace una búsqueda que cuenta el número de alertas que hay en total con las categorías indicadas. En caso contrario se cuenta el número de alertas del distrito o distritos recogidos y las categorías. En ambos casos se devuelve el distrito y el número de categorías correspondiente.

```
'use strict';

const alerta = require('../alertas');

exports.getCountAlertasDistrito = (distrito,count,categorias) =>


    new Promise((resolve, reject) => {
        if(categorias == "Todas"){
            console.debug("entra en este metodo y categorias es TODAS ");
            alerta.aggregate([
        {$match: {distrito: {$in:distrito}}},
        {$group:{_id:"$distrito",total:{$sum:1}}},
            {$project:{distrito:1,total:1}}])
            .then((alertas) => {resolve(alertas);})

            .catch(err => reject({status: 500, message: 'Internal Server Error! verAlertasDistrito.js'}))
        }
        else{
            alerta.aggregate([
        {$match: {distrito: {$in:distrito},categoria:{$in:categorias}}},
        {$group:{_id:"$distrito",total:{$sum:1}}},
            {$project:{distrito:1,total:1}}])
            //alerta.find({distrito: {$in:distrito},categoria:{$in:categorias}},{_id:0, distrito:1}).count()
            .then((alertas) => {resolve(alertas);})

            .catch(err => reject({status: 500, message: 'Internal Server Error! verAlertasDistrito.js'}))
        }
    });
```

135- Figura 7.3.1.6 - Función que devuelve el nombre del distrito y el total de alertas por categoría

- **Añadir alertas:** se efectúa una petición POST para añadir una nueva alerta a la base de datos, todos los parámetros que recibe están completos debido a que este control se hace desde Android. En la implementación de la función **addAlertaBD()** se crea una instancia al esquema de alertas y se hace un save, que añade la alerta a la base de datos de MongoDB[68].

```
'use strict';

const alertas = require('../alertas');

exports.addAlertaBD = (alerta, distrito , fuente , categoria) =>

    new Promise((resolve,reject) => {
        const newAlerta = new alertas({
            alerta: alerta,
            fecha: (new Date()).setHours((new Date()).getHours()+2),
            url: null,
            distrito: distrito,
            categoria: categoria,
            fuente: fuente,
            verificado: false
        });

        newAlerta.save()

        .then(() => resolve ({status: 201, message: "Se ha añadio ok"}))
        .catch(err => {
            reject({status: 500, message: "mierda error"});
        });
    });
```

136- Figura 7.3.1.7 - Función que añade una nueva alerta

7.3.2 IMPLEMENTACIÓN RECOLECTOR DE DATOS

Como ya se explicó en el apartado de funcionalidad, la recolección de datos se realiza a través de diversas técnicas. El lenguaje utilizado ha sido Python[88] y en este punto se explica la implementación de los scripts que cumplen esta función. Se han utilizado varias librerías que este propio lenguaje de programación ofrece para llevar a cabo las distintas tareas.

Para poder construir ambas aplicaciones, así como analizar los diferentes datos generados por las redes sociales, la página de datos de la Policía Municipal de Madrid[23] y Madridiario[58], hemos creado diferentes programas en los que se recoge y analiza toda la información procedente de estos medios. Toda esta información alimenta nuestras colecciones de la base de datos y otorgan los datos correspondientes a las aplicaciones y los usuarios.

Para la conexión a la base de datos alojada en el servicio '*mLab*'[65] existe una clase desarrollada en Python[88]. Esta clase[52][53] comprende varias funciones encargadas de establecer las conexiones a las distintas colecciones presentes en la base de datos. Del mismo modo, también definen funciones para la inserción y eliminación de estos datos (Figura 7.3.2.1, Figura 7.3.2.2 y Figura 7.3.2.3).

BaseDatos.py

```
class baseDatosClass():
    apikey = [redacted]

    def conexion(self):
        MONGODB_URI = [redacted]
        conexion = MongoClient(MONGODB_URI, connectTimeoutMS=30000)
        return conexion.get_default_database()

    def conexionAlertas(self, conexion):
        coleccion = conexion.alertas
        return coleccion

    def conexionEstadisticas(self, conexion):
        coleccion = conexion.estadisticas
        return coleccion

    def conexionEstSeguridad(self, conexion):
        coleccion = conexion.estSeguridad
        return coleccion

    def conexionEstDetenidos(self, conexion):
        coleccion = conexion.estDetenidos
        return coleccion

    def conexionEstAccidentes(self, conexion):
        coleccion = conexion.estAccidentes
        return coleccion

    def desconexion(self):
        mongoClient.close()

    def insertarAlerta(self, coleccion, alerta, fecha, url, distrito, categoria, fuente):
        diccionario = {"alerta": alerta, "fecha" : fecha, "url" : url, "distrito": distrito, "categoria": categoria, "fue
        coleccion.insert_one(diccionario)
```

137- Figura 7.3.2.1 - Clase base de datos

BaseDatos.py

```
def insertarEstadisticas(self, coleccion, distrito, categoria, mes):
    diccionario = {"distrito": distrito, "categoria": categoria, "mes": mes}
    coleccion.insert_one(diccionario)

def insertarEstSeguridad(self, coleccion, distrito, personas, patrimonio, armas, ten_drogas, con_drogas, mes):
    diccionario = {"distrito": distrito, "personas": personas, "patrimonio": patrimonio,
                   "armas": armas, "ten_drogas": ten_drogas, "con_drogas": con_drogas, "mes" : mes}
    coleccion.insert_one(diccionario)

def insertarEstDetenidos(self, coleccion, distrito, detenidos, mes):
    diccionario = {"distrito": distrito, "detenidos": detenidos, "mes": mes}
    coleccion.insert_one(diccionario)

def insertarEstAccidentes(self, coleccion, distrito, conHeridos, sinHeridos, mes):
    diccionario = {"distrito": distrito, "conHeridos": conHeridos, "sinHeridos": sinHeridos, "mes": mes}
    coleccion.insert_one(diccionario)

def eliminarAlerta(self, coleccion, fecha):
    coleccion.remove({"fecha":{'$lte':fecha}})

def eliminarEstadisticas(self, coleccion, mes, mesActual):
    coleccion.remove({"mes":{'$lte':mes}})
    coleccion.remove({"mes":{'$gt':mesActual}})

def eliminarEstSeguridad(self, coleccion,  mes, mesActual):
    coleccion.remove({"mes":{'$lte':mes}})
    coleccion.remove({"mes":{'$gt':mesActual}})

def eliminarEstDetenidos(self, coleccion,  mes, mesActual):
    coleccion.remove({"mes":{'$lte':mes}})
    coleccion.remove({"mes":{'$gt':mesActual}})
```

138- Figura 7.3.2.2 - Funciones clase base de datos

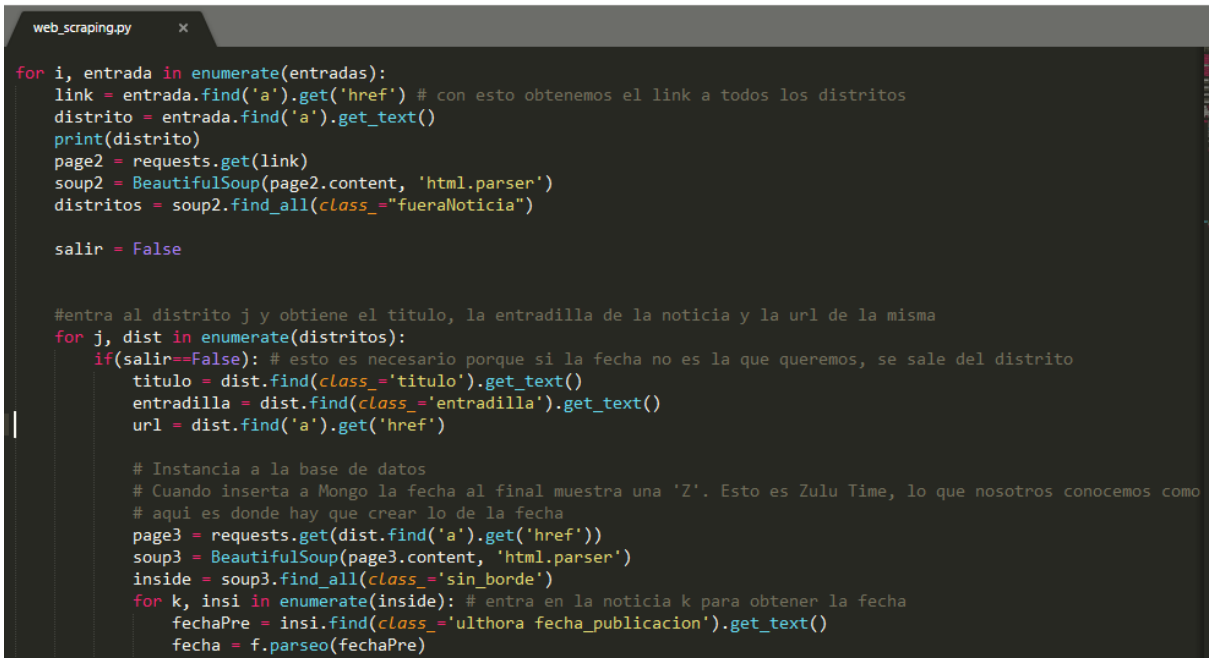
BaseDatos.py

```
def eliminarEstAccidentes(self, coleccion,  mes, mesActual):
    coleccion.remove({"mes":{'$lte':mes}})
    coleccion.remove({"mes":{'$gt':mesActual}})
```

139- Figura 7.3.2.3 - Función eliminar estadísticas accidentes

El periódico online elegido del que se recogen y analizan los datos es Madridiario[58], en concreto, se toman las noticias del apartado de sucesos.

Para la obtención de datos procedentes del periódico escogido, nos servimos de la técnica de web scraping[60], ya mencionada a lo largo del documento. Gracias a la biblioteca de Python llamada *BeautifulSoup*[95] el script es capaz de analizar un documento HTML[19] como es la página web del periódico. En este script, a parte de obtener las alertas, se ha aprovechado para insertarlas a la vez en otra colección de la base de datos para recabar información que alimenta a las estadísticas (Figura 7.3.2.4).



```
web_scraping.py

for i, entrada in enumerate(entradas):
    link = entrada.find('a').get('href') # con esto obtenemos el link a todos los distritos
    distrito = entrada.find('a').get_text()
    print(distrito)
    page2 = requests.get(link)
    soup2 = BeautifulSoup(page2.content, 'html.parser')
    distritos = soup2.find_all(class_="fueraNoticia")

    salir = False


    #entra al distrito j y obtiene el titulo, la entradilla de la noticia y la url de la misma
    for j, dist in enumerate(distritos):
        if(salir==False): # esto es necesario porque si la fecha no es la que queremos, se sale del distrito
            titulo = dist.find(class_='titulo').get_text()
            entradilla = dist.find(class_='entradilla').get_text()
            url = dist.find('a').get('href')

            # Instancia a la base de datos
            # Cuando inserta a Mongo la fecha al final muestra una 'Z'. Esto es Zulu Time, lo que nosotros conocemos como
            # aqui es donde hay que crear lo de la fecha
            page3 = requests.get(dist.find('a').get('href'))
            soup3 = BeautifulSoup(page3.content, 'html.parser')
            inside = soup3.find_all(class_='sin_borde')
            for k, insi in enumerate(inside): # entra en la noticia k para obtener la fecha
                fechaPre = insi.find(class_='ulthora fecha_publicacion').get_text()
                fecha = f.parseo(fechaPre)
```

140- Figura 7.3.2.4 - Código web scraping

Mediante la técnica del Web Scraping[60] somos capaces de recoger de cada suceso su título, su entradilla, su fecha, su url y el distrito al que pertenece.[90] Para saber a qué categoría pertenece cada uno, existe un clasificador (Figura 7.3.2.5) que se encarga de categorizarlos en función de ciertas palabras clave que puedan aparecer en su título. Así, conseguimos guardar en la base de datos su categoría (pudiendo ser: criminalidad, contaminación etc.). En el caso de que el clasificador le asigne la categoría “Nada”, ese suceso es descartado y por lo tanto no se guarda en nuestras colecciones de la base de datos. La clase Clasificador[62],[70] encargada de realizar lo mencionado se muestra a continuación. Cabe mencionar que se ha utilizado *requests*[92], una librería HTTP de Python simple, elegante y muy potente.

Clasificador.py

```
class ClasificadorClass:

    def parseoCategoria(self, codigo):
        c = ""
        if codigo == "1":
            c = "Desastres y accidentes"
        if codigo == "2":
            c =  "Contaminación"
        if codigo == "3":
            c = "Eventos"
        if codigo == "4":
            c = "Criminalidad"
        if codigo == "5":
            c =  "Nada"
        if codigo == "6":
            c = "Tráfico"
        if codigo == "7":
            c = "Transporte público"
        if codigo == "8":
            c = "Terrorismo"
        return c

    def clasificadorNoticias(self, noticia):
        parameters = {'key': keyNoticias, 'model': modelNews, 'txt': noticia}
        r = requests.request('POST', api, data=parameters)
        response = r.content
        response_json = json.loads(response.decode('utf-8'))
        if(response_json['category_list']):
            cat = self.parseoCategoria(response_json['category_list'][0]['code'])
        else:
            cat = "Nada"
        return cat
```

141- Figura 7.3.2.5 - Clase clasificador

Clasificador.py

```
    def clasificarTweets(self, tweet):
        parameters = {'key': keyTwitter, 'model': modelTwitter, 'txt': tweet}
        r = requests.request('POST', api, data=parameters)
        response = r.content
        response_json = json.loads(response.decode('utf-8'))
        if(response_json['category_list']):
            cat = self.parseoCategoria(response_json['category_list'][0]['code'])
        else:
            cat = "Nada"
        print(cat)
        print("\n")
        return cat


    def clasificadorZona(self, tweet):
        module_id = 'cl_iYYd3Hj2'
        res = mlZona.classifiers.classify(module_id, tweet, sandbox=True)
        resultado = res.result
        zona = resultado[0][0]["label"]
        print(zona)
        return zona
```

142- Figura 7.3.2.6 - Funciones clasificador

Este programa se ejecuta cada 5 minutos, comprobando que el suceso se encuentre dentro del intervalo de la hora actual y 5 minutos antes (Figura 7.3.2.7). De este modo, si y sólo si se encuentra dentro de este rango, se inserta en nuestra base de datos. Gracias a esto evitamos tener sucesos repetidos en nuestras colecciones.

```
dif = datetime.now() - timedelta(minutes=5)
fd = datetime.strptime(fecha, "%Y-%m-%d %H:%M:%S")

if(fd > dif): # si la fecha de la noticia es superior a la hora_actual - 5 min se tiene que guardar
    categoria = clasificador.clasificadorNoticias(titulo)
    print(categoria)
    if(categoria != "Nada"):
        bd.insertarAlerta(bdAlertas,titulo,fd,url,distrito,categoria,"Madridiario")
        mes = fd.month
        bd.insertarEstadisticas(bdEstadisticas,distrito,categoria,mes)
else:
    salir = True
```

143- Figura 7.3.2.7 - Código 5 min

Para ejecutar este programa se ha creado un Cron[14] que se encarga de ejecutar este proceso cada 5 minutos sin necesidad de tener que irlo ejecutando “a mano”. A continuación, se van a explicar cuáles son los pasos que hemos tenido que llevar a cabo:

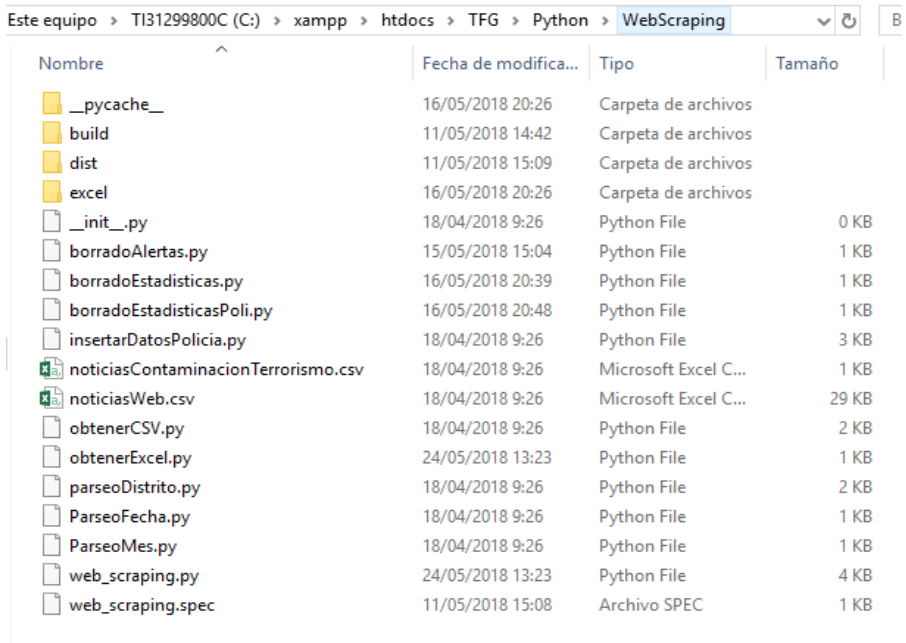
En primer lugar, se debe tener en cuenta la necesidad de crear el ejecutable del programa. Para ello, se ha utilizado un módulo de Python llamado “pyinstaller”[87] que permite en tan sólo unos sencillos pasos, crear un ejecutable a partir de un programa .py. Los pasos seguidos han sido los siguientes:

1) Instalación del módulo “pyinstaller”[87] en la consola de comandos utilizando: **pip install pyinstaller**.
2) Acceso a la ruta donde se encuentre el programa .py mediante el comando **cd** (change directory).

```
C:\Users\Silvia>cd C:\xampp\htdocs\TFG\Python\WebScraping

C:\xampp\htdocs\TFG\Python\WebScraping>
```

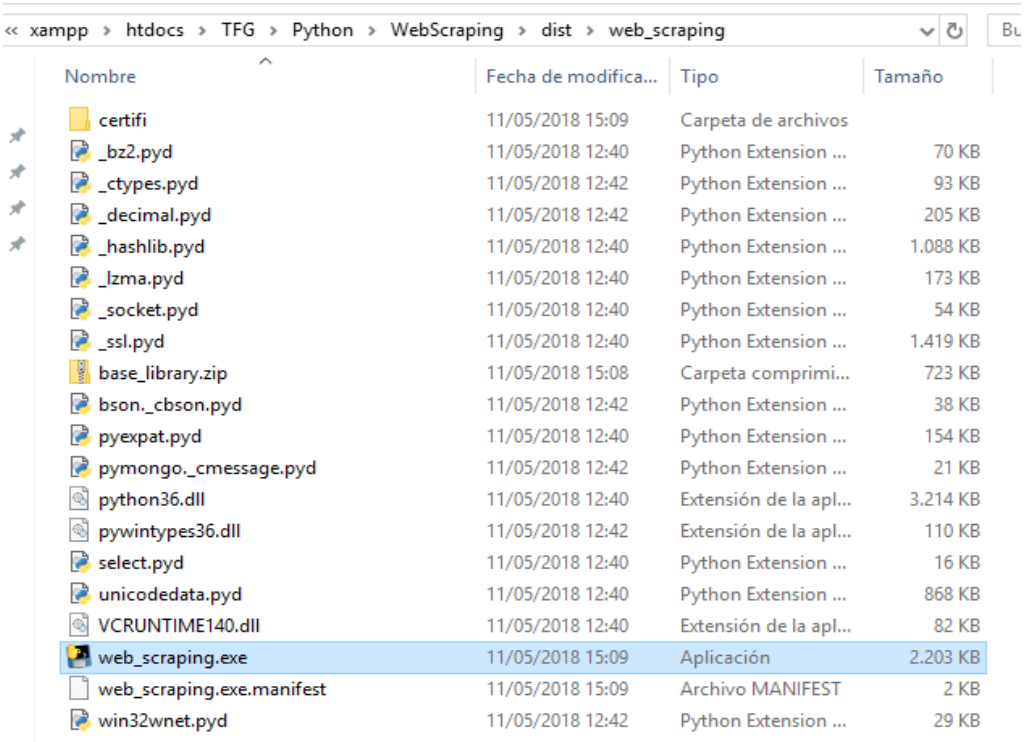
144- Figura 7.3.2.8 - Ruta .py

3) Utilización del siguiente comando: **pyinstaller nombre_del_programa.py**
4) Automáticamente se crean unas carpetas en ese directorio:



145- Figura 7.3.2.9 - Nuevas carpetas en directorio .py

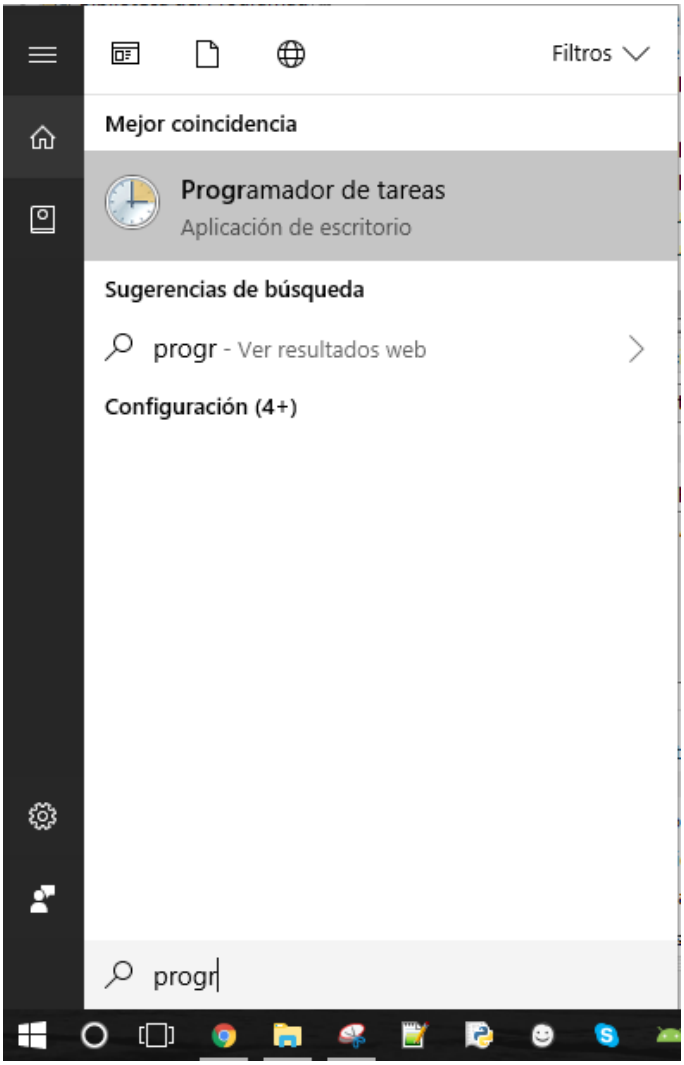
5) Abriendo la carpeta llamada “dist” y dentro de ella abriendo también una subcarpeta con el nombre del programa en cuestión, se puede encontrar el ejecutable que se buscaba crear:



146- Figura 7.3.2.10 - Nuevo archivo exe

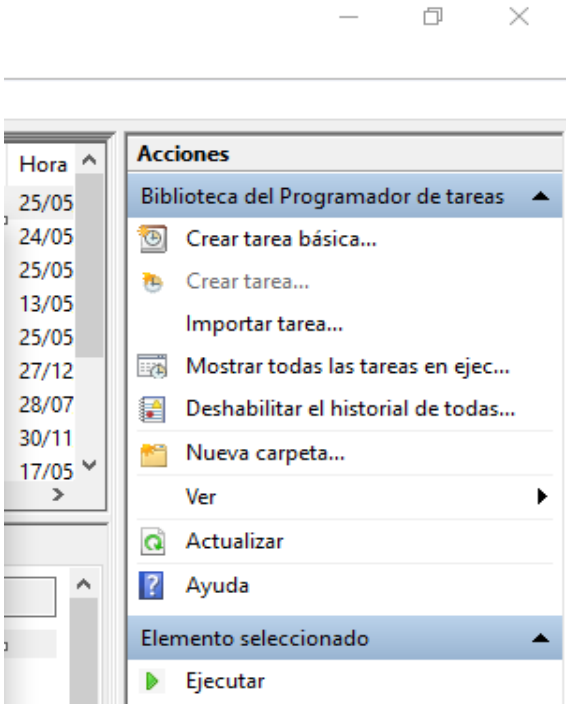
Una vez finalizada la creación del ejecutable se procede a la creación del Cron. Éste será el encargado de ejecutar el archivo ejecutable (creado anteriormente) cada 5 minutos. Para ello se han seguido los siguientes pasos.[14]

1) Búsqueda de la aplicación llamada “Programador de tareas”:

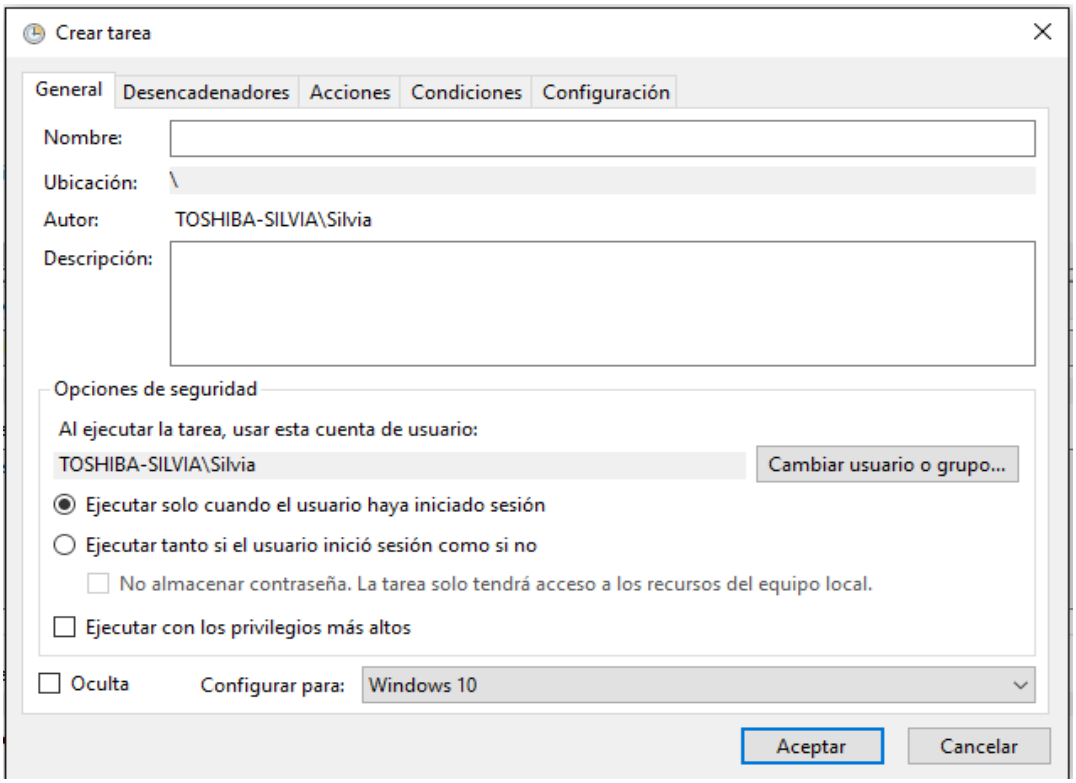


147- Figura 7.3.2.11 - Buscar programador de tareas

2) A continuación, es necesario pulsar en “Crear tarea” en el menú de acciones que aparece a la izquierda:

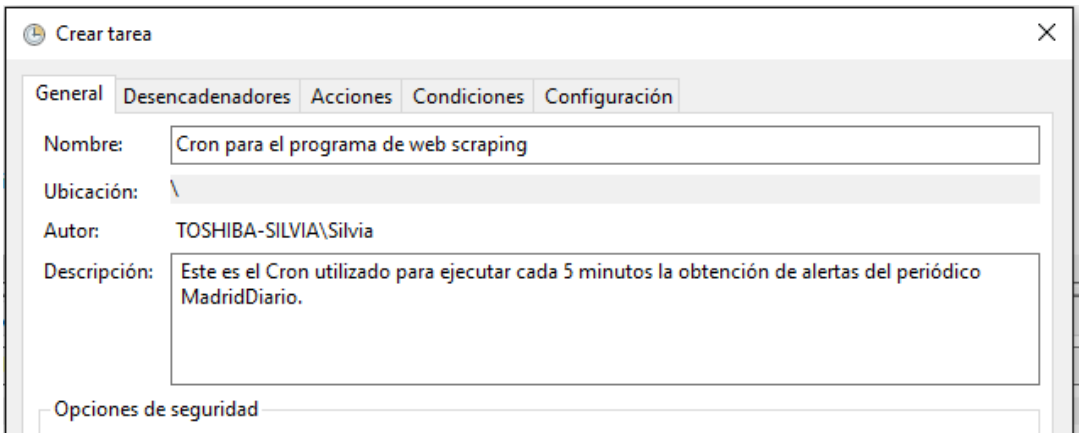


148- Figura 7.3.2.12 - Lista acciones



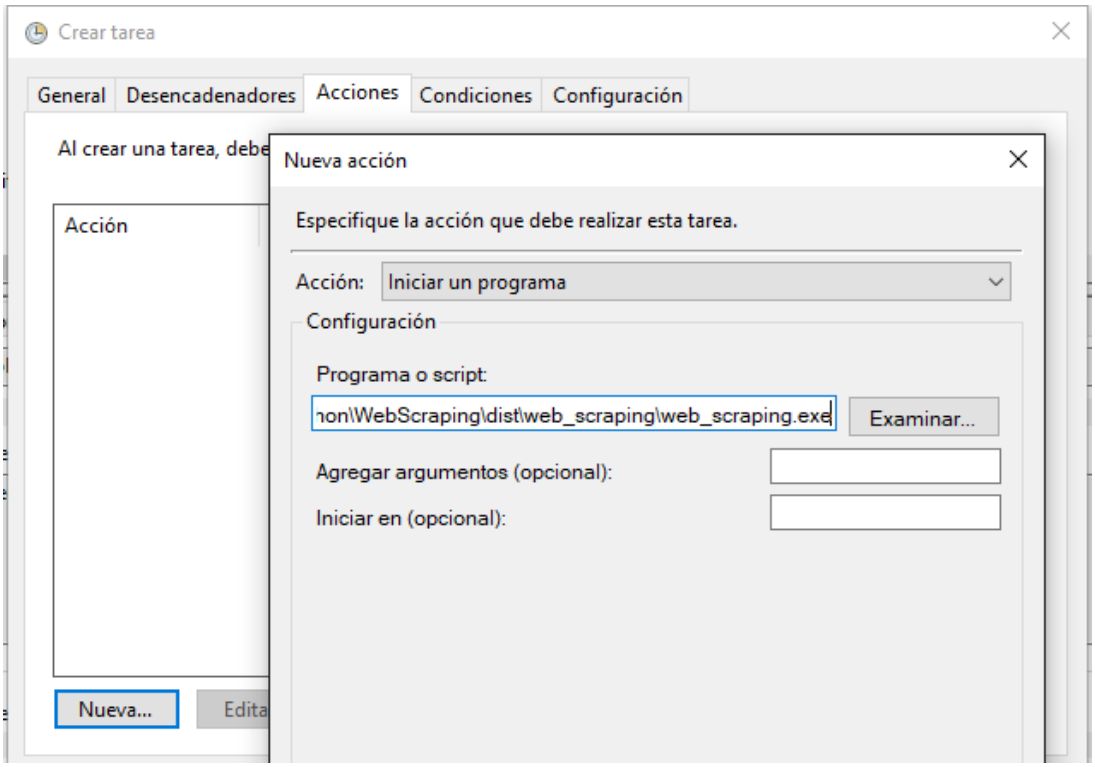
149- Figura 7.3.2.13 - Cron: Crear tarea

3) Se da nombre y descripción a la tarea:



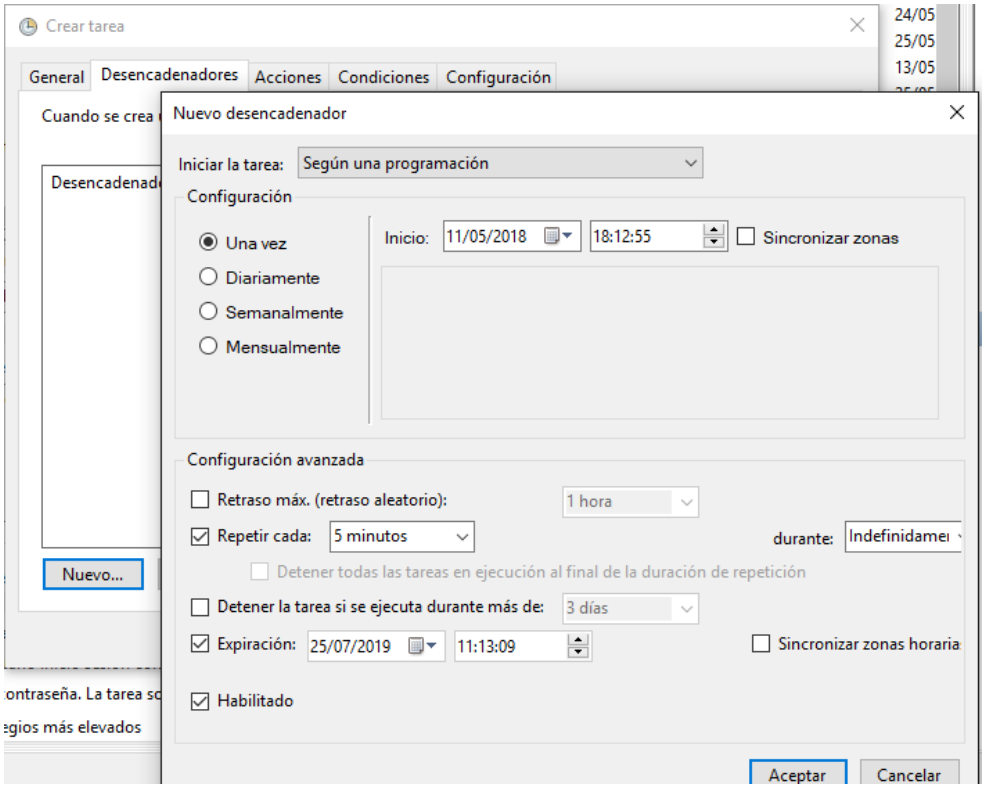
150- Figura 7.3.2.14 - Cron: Nombre y descripción nueva tarea

4) En la pestaña acciones, se elige la opción de “Iniciar un programa” y se selecciona la ruta del archivo exe que se desea ejecutar:



151- Figura 7.3.2.15 - Cron: Nueva acción

5) Por último, en la pestaña “Desencadenadores” se debe indicar cada cuánto tiempo es necesario que se realice esta tarea:



152- Figura 7.3.2.16 - Cron: Desencadenador Programación

Una vez finalizado el Cron, esta tarea programada va a ser la que se encargue de ejecutar el programa cada 5 minutos y por tanto, no es necesario preocuparse por tener que ejecutar nada más para la obtención de datos del periódico Madridiario[58].

Los datos generados por la red social Twitter se analizan y se recogen a partir de un programa que está en continuo funcionamiento. Se recogen sólo los tweets generados por las cuentas indicadas en el programa, en este caso, todas están relacionadas con sucesos y noticias de Madrid[59]. Cada vez que recibimos un tweet, el programa aplica un filtro para evitar tener contenido innecesario, que consiste en descartar los retweets, una respuesta a otro usuario o una mención por parte de otro usuario. Si el tweet ha pasado el filtro anterior, se le asigna una categoría a través del clasificador, si la categoría asignada es “Nada”, este también quedaría descartado. Por último se le asigna un distrito mediante un clasificador de zonas y se actualizan las colecciones correspondientes de la base de datos con la información del nuevo tweet, la categoría y distrito obtenidos, en caso de no asignarle ningún distrito al tweet entonces no se añadirá a las colecciones correspondientes (Figura 7.3.2.17, Figura 7.3.2.18).

MyListener.py

```
class listener(StreamListener):
    def on_data(self, data):
        try:
            data.rstrip('\n')
            carga = json.loads(data)
            mencionVacia = False
            mencionInteresa = False
            #existe mencion
            if "entities" in carga:
                #Obtenemos la mencion
                mencion =carga["entities"]["user_mentions"]
                #La lista de menciones es vacia, es porque no es una mencion
                if (not mencion):
                    mencionVacia= True
                #Es una mencion de un usuario de la lista a otro
                elif ((mencion[0]["id_str"]in usuarios) and (carga["user"]["id_str"] in usuarios)):
                    mencionInteresa= True
            if "created_at" in carga and not("RT" in carga["text"]) and (mencionVacia or mencionInteresa):
                self.insertarDatos(carga);
                print("Me sirve")
            else:
                print("Este tweet no me importa")
            return True
        except BaseException as e:
            print ("Error" + e)
        return True
```

153- Figura 7.3.2.17 - Twitter: MyListener.py

MyListener.py

```
    def insertarDatos(self, carga):
        bd = BaseDatos.baseDatosClass()
        con = bd.conexion()
        bdAlertas = bd.conexionAlertas(con)
        bdEstadisticas = bd.conexionEstadisticas(con)
        c = Clasificador.ClasificadorClass()
        tweet = carga["text"]
        lista = []
        lista.append(tweet)
        categoria = c.clasificarTweets(tweet)
        print ("La categoria es : " , categoria);
        if(categoria != "Nada"):
            fecha= time.strftime('%Y-%m-%d %H:%M:%S', time.strptime(carga['created_at'],'%a %b %d %H:%M:%S +0000 %Y'))
            datetime_object = datetime.strptime(fecha, '%Y-%m-%d %H:%M:%S');
            resultado = datetime_object + timedelta(hours=2)
            mes = time.strftime('%m', time.strptime(carga['created_at'],'%a %b %d %H:%M:%S +0000 %Y'))
            mes = int(mes)
            nombreUsuario = "@"+carga["user"]["screen_name"]
            zona = c.clasificadorZona(lista)
            print(zona)
            bd.insertarEstadisticas(bdEstadisticas,zona,categoria,mes)
            bd.insertarAlerta(bdAlertas,tweet,resultado,None,zona,categoria,nombreUsuario)
```

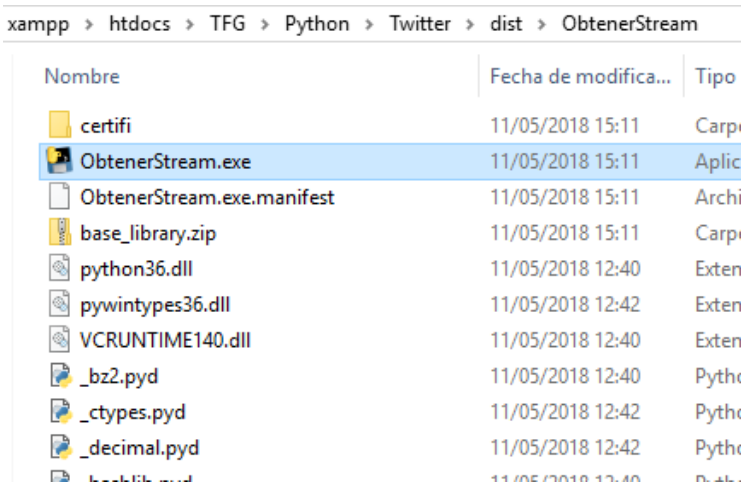
154- Figura 7.3.2.18 - Twitter: MyListener.py - 2

ObtenerStream.py

```python
#Obtener tweet en stream de un usuario x
#!/usr/bin/python
# coding: utf-8
import hidden
import MyListener
import tweepy
import json
import var
import time

from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
from MyListener import listener


#Funcion que llama al stream de los usuarios que deseemos
def streamUsuario(user):
    keys = hidden.oauth()
    auth = OAuthHandler(keys['consumer_key'], keys['consumer_secret'])
    auth.set_access_token(keys['access_token'], keys['access_secret'])

    listen = listener()
    twitter_stream = Stream(auth, listen)
    twitter_stream.filter(follow=user)


#Main
usuarios = var.var()
streamUsuario(usuarios)
```

155- Figura 7.3.2.19 - Twitter: ObtenerStream.py

Para que este programa se ejecute constantemente y obtenga alertas de Twitter en streaming, ha sido necesario crear otro Cron[14] siguiendo los pasos que se acaban de explicar en el punto anterior. En primer lugar, se crea el ejecutable del archivo “ObtenerStream.py” (Figura 7.3.2.20) y a continuación se crea un Cron[14] que hace que este nuevo archivo ejecutable se esté ejecutando en todo momento.



156- Figura 7.3.2.20 - ObtenerStream ejecutable

Del mismo modo, se ha creado un script con el objetivo de obtener datos para la elaboración de estadísticas procedentes de datos policiales. La fuente que se ha elegido para la extracción de estos es la propia página que ofrece el Ayuntamiento de Madrid acerca de datos abiertos de las actuaciones policiales[23]. En ella se proporciona de forma mensual un Excel con las intervenciones del mes anterior.

Con ello, en primer lugar, extraemos dicho Excel[16] una vez cada dos meses (debido a que la página oficial tarda ese periodo en subir sus estadísticas) para, posteriormente, guardar los datos que se han considerado más interesantes en diferentes colecciones de nuestra base de datos. Este script se ha servido de la librería wget[89], útil para la descarga de documentos a través de Python. Estos datos que se han considerado más sugerentes son estadísticas de detenidos por distrito, accidentes por distrito y relacionadas con la seguridad por distrito. Para poder realizar esto nos hemos servido, al igual que en ocasiones ya mencionadas más arriba, de la técnica de Web Scraping[60].

obtenerExcel.py

```
#-*- coding=utf-8 -*-

from bs4 import BeautifulSoup
import requests
import wget

init = 'http://datos.madrid.es'
url = 'https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a
page = requests.get(url)
print(page.text)

soup = BeautifulSoup(page.text, 'html.parser')

entradas = soup.find_all(class_='asociada-link ico-xlsx')
print(entradas)

link = entradas[0].get('href') # esto devuelve el link del ultimo mes

excel = init + link
filename = wget.download(excel, out='./excel/')
```

157- Figura 7.3.2.21 - Obtener Excel .py

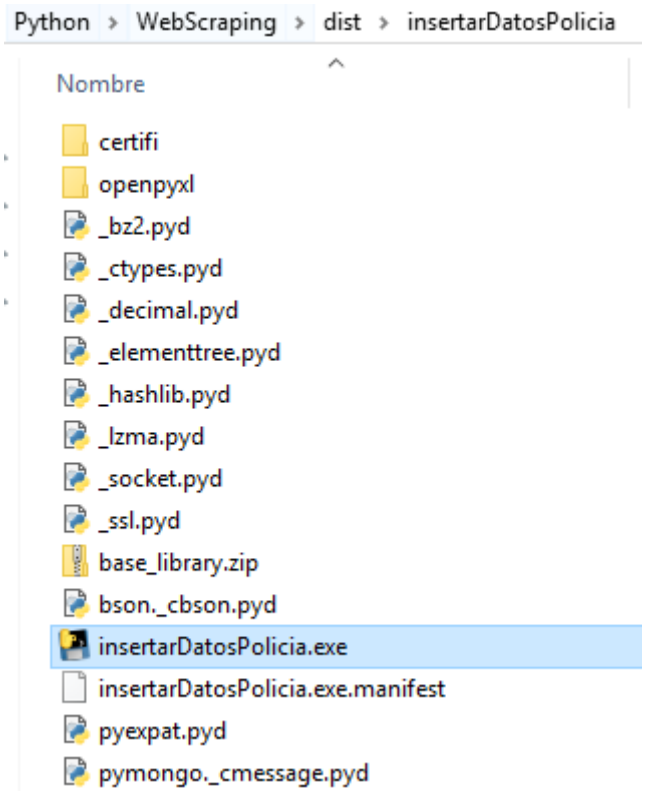
```
insertarDatosPolicia.py

seleccion = hoja['A4':'F24']
for filas in seleccion:
    for i in range(0,6):
        if i == 0:
            distrito = filas[i].value
            #print("Distrito: " + distrito)
        if i == 1:
            personas = filas[i].value
            #print("Personas: " + str(personas))
        if i == 2:
            patrimonio = filas[i].value
            #print(patrimonio)
        if i == 3:
            armas = filas[i].value
            #print(armas)
        if i == 4:
            ten_drogas = filas[i].value
            #print(ten_drogas)
        if i == 5:
            con_drogas = filas[i].value
            #print(con_drogas)
    d = parseoDistrito.ParseoDistritoClass()
    bd.insertarEstSeguridad(bdEstSeguridad, d.parseoDistrito(distrito), personas, patrimonio, armas, ten_drogas, con_drog

print("Datos insertados en la bd EstSeguridad")
```

158- Figura 7.3.2.22 - insertarDatosPolicia.py

```
insertarDatosPolicia.py

seleccion2 = hoja2['A4':'B24']
for filas in seleccion2:
    for i in range(0,2):
        if i == 0:
            distrito = filas[i].value
            #print("Distrito: " + distrito)
        if i == 1:
            detenidos = filas[i].value
            #print("Detenidos: " + str(detenidos))
    d = parseoDistrito.ParseoDistritoClass()
    bd.insertarEstDetenidos(bdEstDetenidos, d.parseoDistrito(distrito), detenidos, m.parseoMes(mes))

print("Datos insertados en la bd EstDetenidos")

seleccion3 = hoja3['A4:C24']
for filas in seleccion3:
    for i in range(0,3):
        if i == 0:
            distrito = filas[i].value
        if i == 1:
            conHeridos = filas[i].value
        if i == 2:
            sinHeridos = filas[i].value
    d = parseoDistrito.ParseoDistritoClass()
    bd.insertarEstAccidentes(bdEstAccidentes, d.parseoDistrito(distrito), conHeridos, sinHeridos, m.parseoMes(mes))

print("Datos insertados en la bd estAccidentes")
```

159- Figura 7.3.2.23 - insertarDatosPolicia.py - 2

Para recoger estos datos de la Policía Municipal de Madrid[23] también se ha creado otro Cron siguiendo los pasos ya explicados antes. En primer lugar, se crea el ejecutable del archivo “insertarDatosPolicia.py” (Figura 7.3.2.24) y a continuación se crea un Cron que hace que este nuevo archivo se realice el día 25 de cada mes.



160- Figura 7.3.2.24 - InsertarDatosPolicia ejecutable

Por otra parte, también se han implementado varios programas de borrado para ir limpiando la base de datos de alertas antiguas. Cada uno de ellos tiene su ejecutable correspondiente dentro de la carpeta “dist” para poder ser llamado y ejecutado cada cierto tiempo por un Cron. En concreto, existen 3 programas diferentes, uno para eliminar las alertas de hace más de 3 días (Figura 7.3.2.25), otro para eliminar las estadísticas que no sean de los últimos dos meses atrás (Figura 7.3.2.26) y por último, otro para eliminar las estadísticas de la Policía Municipal de Madrid[23] también de hace 3 meses o más (Figura 7.3.2.27).

borradoAlertas.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from bs4 import BeautifulSoup
import requests
import json
import sys
sys.path.append('../')

from BaseDatos import BaseDatos
import ParseoFecha
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta

bd = BaseDatos.baseDatosClass()
con = bd.conexion()
bdAlertas = bd.conexionAlertas(con)

#Calculo la fecha actual
fechaActual = datetime.now()

#Resto 3 dias
dias = relativedelta(days=3)
fecha_menos_dias = fechaActual - dias

#Elimino las alertas que tengan fecha inferior a la anteriormente calculada
bd.eliminarAlerta(bdAlertas, fecha_menos_dias )
```

161- Figura 7.3.2.25 - Borrado alertas

borradoEstadisticas.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from bs4 import BeautifulSoup
import requests
import json
import sys
sys.path.append('../')

from BaseDatos import BaseDatos
import ParseoFecha
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta

bd = BaseDatos.baseDatosClass()
con = bd.conexion()
bdEstadisticas = bd.conexionEstadisticas(con)


#Calculo la fecha actual
fechaActual = datetime.now()

#Resto 3 meses
mesActual = fechaActual.month
mes = mesActual - 3

#Elimino las alertas que tengan fecha inferior a la anteriormente calculada
bd.eliminarEstadisticas(bdEstadisticas, mes , mesActual)
```

162- Figura 7.3.2.26 - Borrado estadísticas

borradoEstadisticasPoli.py

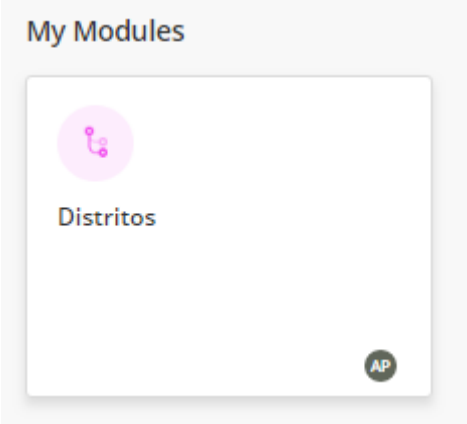
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from bs4 import BeautifulSoup
import requests
import json
import sys
sys.path.append('../')
from BaseDatos import BaseDatos
from datetime import datetime, timedelta

# Conexion a la bd (estadisticas policias)
bd = BaseDatos.baseDatosClass()
con = bd.conexion()
bdEstSeguridad = bd.conexionEstSeguridad(con)
bdEstDetenidos = bd.conexionEstDetenidos(con)
bdEstAccidentes = bd.conexionEstAccidentes(con)


#Calculo la fecha actual
fechaActual = datetime.now()

#Resto 3 meses
mesActual = fechaActual.month
mes = mesActual - 3

#Elimino las alertas que tengan fecha inferior a la anteriormente calculada
bd.eliminarEstSeguridad(bdEstSeguridad, mes , mesActual)
bd.eliminarEstDetenidos(bdEstDetenidos, mes , mesActual)
bd.eliminarEstAccidentes(bdEstAccidentes, mes , mesActual)
```

163- Figura 7.3.2.27 - Borrado estadísticas Policía

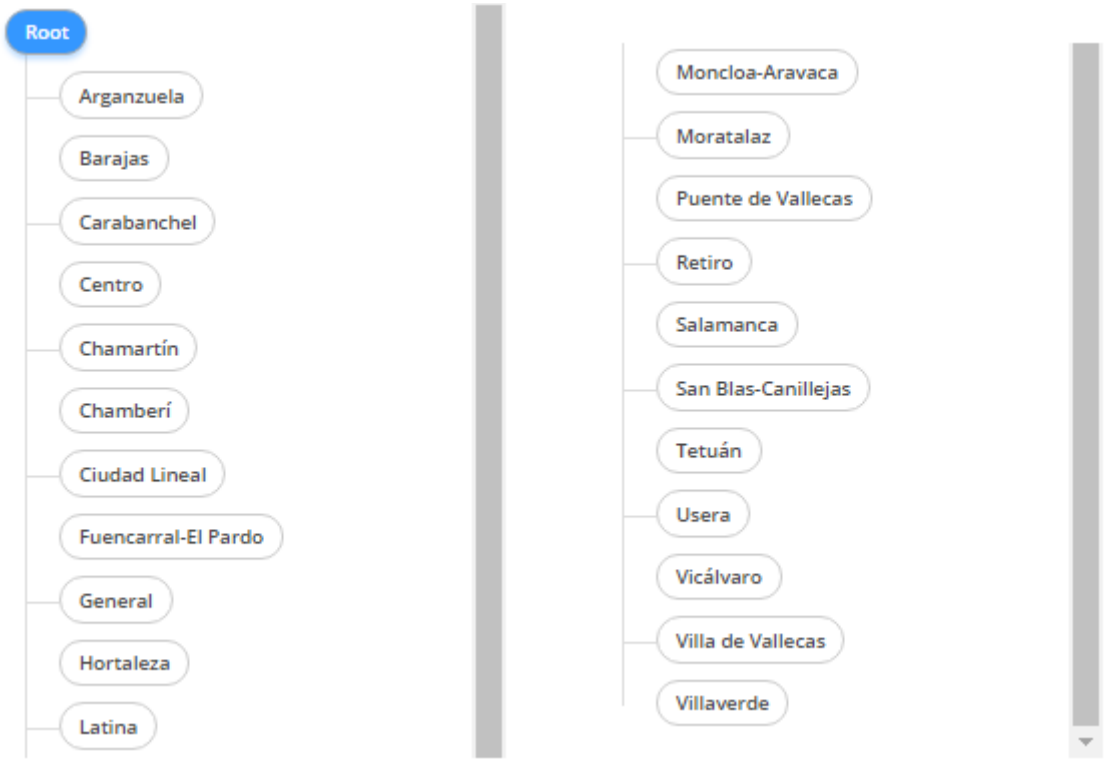
Además de estos programas que se encargan de recoger y de borrar los datos, se hace uso también de dos clasificadores.

Para saber el distrito al que pertenece cada alerta de Twitter según el texto, se ha utilizado un clasificador llamado "**MonkeyLearn**"[36]. Se trata de una plataforma de Machine Learning[80] para el análisis de texto que se encuentra en la nube.

En esta plataforma se tiene un módulo llamado Distritos que contiene los 21 distritos de Madrid más uno llamado General (para alertas que no son sólo de un distrito en concreto):



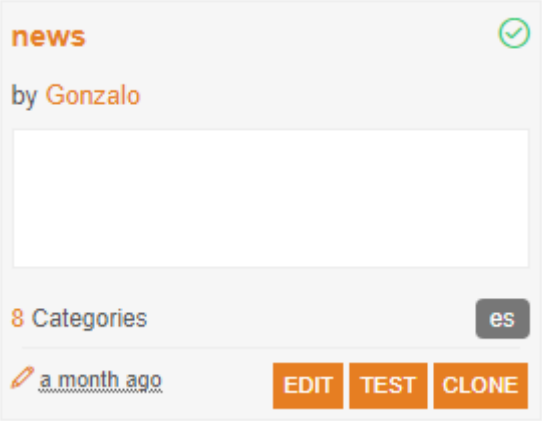
164- Figura 7.3.2.28 - Módulos: distritos



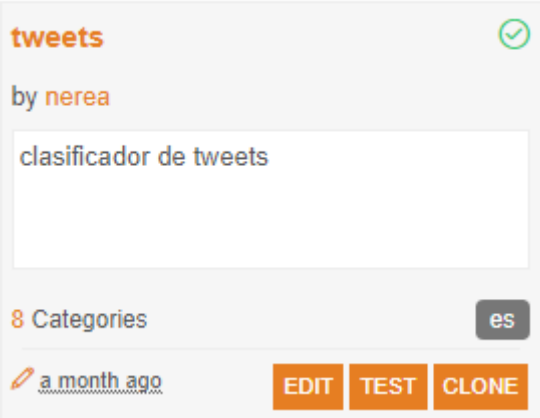
165- Figura 7.3.2.29 - Root: Lista distritos

Para insertar en la base de datos las alertas con sus categorías correspondientes se ha utilizado un clasificador llamado "**MeaningCloud"**[63]. Se trata de un servicio web de analítica y minería de textos gratuito y muy fácil de utilizar.

Dentro de este clasificador se ha creado un modelo de clasificación llamado "news" y otro llamado "tweets":



166- Figura 7.3.2.30 - Clasificador: News



167- Figura 7.3.2.31 - Clasificador: tweets

Ambos modelos utilizan las mismas categorías sin embargo el primero se encarga de categorizar las alertas que provienen del periódico Madridiario[58] mientras que el segundo se encarga de categorizar las procedentes de Twitter.

168- Figura 7.3.2.32 - Categorías clasificador

Como se puede observar en la Figura 7.3.2.32, dentro de ambos modelos existen 8 categorías. Dentro de cada una de ellas ha sido necesario ir escribiendo palabras clave que puedan ayudar a clasificar mejor cada alerta y una gran cantidad de ejemplos que van entrenando al clasificador.

A continuación, se va a explicar cada categoría de manera detallada con ayuda además de algún ejemplo:

1. **Contaminación**: las alertas que pertenecen a esta categoría son aquellas relacionadas con la contaminación de la ciudad de Madrid, es decir, avisos de activación o desactivación de escenarios o protocolos de contaminación, medidas preventivas, cortes o regulaciones de velocidad etc.

    - Ejemplos de algunos **términos relevantes**: anti contaminación, calentamiento global, humo, contaminación, plaga, polución, tóxico…
    - Ejemplos de **alertas**:
        - El Ayuntamiento implementa el protocolo 3, sólo circularán los coches con matrícula impar.
        - La capital sube un peldaño: activado el escenario 2.

2. **Criminalidad:** las alertas que pertenecen a esta categoría son aquellas relacionadas con la criminalidad presente en la ciudad de Madrid, es decir, relacionadas con acciones consideradas crímenes (actos ilegales graves). Por ejemplo, se consideran crímenes: homicidios, abusos, agresiones, tráfico de drogas...

- Ejemplos de algunos **términos relevantes**: abuso, acuchillar, agresión, amenaza, arma, arma blanca, acusado, cárcel, criminal, cocaína, desaparecido, desmantelado, detención, droga, homicidio, hurto...
- Ejemplos de **alertas**:
    - Un hombre, grave tras recibir dos puñaladas en Carabanchel.
    - Detenidos en Gran Vía con ocho kilos de cocaína.

3. **Desastres y accidentes:** las alertas que pertenecen a esta categoría son aquellas relacionadas con los desastres y accidentes presentes en la ciudad de Madrid, es decir, alertas relacionadas con sucesos imprevistos que causan daños personales o materiales. Por ejemplo, se consideran accidentes o desastres: incendios, atropellos, inundaciones, accidentes de tráfico...

    - Ejemplos de algunos **términos relevantes**: accidente, atrapado, desastre, choque, colisión, desprende, explosión, incendio, percance, terremoto...
    - Ejemplos de **alertas**:
        - Dos bomberos resultan heridos leves mientras extinguen un incendio en un colegio en el centro de Madrid.
        - Herido grave un joven atropellado por un autobús en Cuatro Caminos.

4. **Eventos:** las alertas que pertenecen a esta categoría son aquellas relacionadas con diferentes eventos que tienen lugar en la ciudad de Madrid como por ejemplo: manifestaciones, huelgas, conciertos, carreras...

    - Ejemplos de algunos **términos relevantes**: actividades, concentración, convención, fiesta, huelga, espectáculo, evento, feria, festival, marcha, manifestación, protesta...
    - Ejemplos de **alertas**:
        - Manifestación en la calle Atocha.
        - Carrera popular patrocinada por leche asturiana.

5. **Terrorismo:** las alertas que pertenecen a esta categoría son aquellas relacionadas con actos terroristas en la ciudad de Madrid. Por ejemplo, se consideran de esta categoría: medidas antiterroristas, amenazas, atentados...

    - Ejemplos de algunos **términos relevantes**: amenazas, antiterrorista, artefacto, atentado, bomba, estado islámico, explosivo, inmolación, terrorismo, yihadismo...
    - Ejemplos de **alertas**:
        - Estado Islámico amenaza con atentados en el centro de Madrid.
        - Desbaratada una célula que planeaba atentados en el barrio de Chamberí.

6. **Transporte público:** las alertas que pertenecen a esta categoría son aquellas relacionadas con el transporte público de la ciudad de Madrid. Por ejemplo, se consideran de esta categoría: avisos de interrupciones o reanudaciones de líneas de metro o buses, subidas o bajadas de las tarifas, nuevas paradas, obras en el metro...

    - Ejemplos de algunos **términos relevantes**: transporte público, metro, línea, cercanías, consorcio, parada bus, EMT, bus, taxi, movilidad, tarifas, tarjeta multi...
    - Ejemplos de **alertas**:
        - La línea 9 de Metro, interrumpida durante más de cuatro horas.
        - El Consorcio lanza a partir de este lunes el nuevo abono transporte de 10 euros para parados.

7. **Tráfico:** las alertas que pertenecen a esta categoría son aquellas relacionadas con el tráfico de la ciudad de Madrid. Por ejemplo, se consideran de esta categoría retenciones, atascos, cortes, carriles cortados...

    - Ejemplos de algunos **términos relevantes**: accesos, anchura, aparcamiento, atasco, calzada, carretera, carril bus, cortada, frecuencia, M-30, retención...
    - Ejemplos de **alertas**:
        - Cerrado el túnel de la avenida del Planetario hasta el sábado.
        - Carril cortado en el barrio de Usera.

8. **Nada:** las alertas que pertenecen a esta categoría son aquellas que no son relevantes para nuestra aplicación, es decir que no encajan con las categorías anteriores.

    - Ejemplos de **alertas** que no nos interesan:
        - 'Madrid Nuevo Norte': el Ayuntamiento ya estudia fechas para reunirse con la Comunidad.
        - Arrancan las negociaciones en Eulen Madrid.

Se han tenido que utilizar estos dos clasificadores diferentes porque utilizando únicamente “MonkeyLearn” se limitaban las consultas.

7.4 ALGORITMOS

Nuestro sistema cuenta principalmente con dos algoritmos:

- Cálculo de la distancia entre dos coordenadas: este algoritmo calcula la distancia que hay entre dos coordenadas (latitud y longitud), teniendo en cuenta la curvatura terrestre, ya que si no se obtendría la distancia entre dos puntos en un plano. que la tierra es redonda y por ello se hace uso del radio de la Tierra. Para desarrollar este algoritmo, se ha tenido en cuenta la Fórmula de *Haversine*[97] y por lo tanto el radio de la tierra. Al usar esta fórmula se asume que la Tierra es completamente redonda, lo que produce que la distancia calculada tenga una pequeña tasa de error.

```
public static double distanciaCoord(double lat1, double lng1, double lat2, double lng2) {
    //double radioTierra = 3958.75;//en millas
    double radioTierra = 6371;//en kilómetros
    double disLat = Math.toRadians(lat2 - lat1);
    double disLng = Math.toRadians(lng2 - lng1);
    double sindLat = Math.sin(disLat / 2);
    double sindLng = Math.sin(disLng / 2);
    double val = Math.pow(sindLat, 2) + Math.pow(sindLng, 2)
            * Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2));
    double valor2 = 2 * Math.atan2(Math.sqrt(val), Math.sqrt(1 - val));
    double distanciaCoor = radioTierra * valor2;
    return distanciaCoor;
}
```

169- Figura 7.3.3.1 - Algoritmo cálculo de distancia entre dos coordenadas

- Obtener los distritos que engloban una circunferencia de x kilómetros de radio.

A partir de este algoritmo se obtienen los distritos que están dentro de la circunferencia de x kilómetros de radio, siendo el centro las coordenadas de ubicación del usuario o las coordenadas del distrito elegido. Cada distrito tiene unas coordenadas, que suelen coincidir con el centro del distrito o algo característico de él.

Para poder obtener los distritos que hay dentro de la circunferencia formada por el radio elegido por el usuario, se recorre el HashMap de coordenadas que está formado por el distrito y un arraylist[71] de los puntos que forman el distrito. Los puntos que se han incluido por cada distrito son el punto central y puntos que limitan el distrito, de esta forma si un trozo de distrito está dentro de la circunferencia también se incluye y el margen de error es menor.

Cuando se recorre el *HashMap* de distritos-coordenadas en cada iteración se obtiene la lista de puntos del distrito correspondiente y a cada punto se le aplica el algoritmo anterior (distancia de coordenadas) respecto a la ubicación del usuario. Si la distancia de las coordenadas del usuario al punto del distrito es menor o igual que el radio quiere decir que dicho distrito se encuentra dentro de la circunferencia y por lo consiguiente se añade a la lista de distritos que están dentro de ella.

En el momento en el que un distrito se encuentre dentro de la circunferencia se pone el booleano de *marcadorEncontrado* a true y se deja de iterar la lista de puntos del distrito actual y se pasa a un nuevo distrito. De esta forma no habrá distritos repetidos en la lista que se devuelve.

```
public static ArrayList<String> obtenerDistritosRadio(HashMap<String, ArrayList<Pair<Double, Double>>> coordenadas, int kms, Double parsLat, Double parsLong){
    coordenadas = initCoord();
    ArrayList<String> distRadio = new ArrayList<String>();
    boolean marcadorEncontrado;
    Iterator<Map.Entry<String, ArrayList<Pair<Double, Double>>>> iterator = coordenadas.entrySet().iterator();
    while (iterator.hasNext()) {
        Map.Entry<String, ArrayList<Pair<Double, Double>>> it = iterator.next(); // iterador del HashMap
        if(!it.getKey().equals("Todos") && !it.getKey().equals("General")) {
            ArrayList<Pair<Double, Double>> listAux = it.getValue(); // obtengo el arrayList del distrito que estoy iterando
            Iterator<Pair<Double, Double>> itArrayList = listAux.iterator(); // iterador del arrayList
            marcadorEncontrado = false;
            while (itArrayList.hasNext() && !marcadorEncontrado) {
                Pair<Double, Double> parAux = itArrayList.next();
                Double lat = parAux.first;
                Double longi = parAux.second;
                Double var = distanciaCoord(parsLat, parsLong, lat, longi);
                if (var <= kms) {
                    distRadio.add(it.getKey());
                    marcadorEncontrado = true; // en el momento en el que encuentre una coordenada dentro, ya no es necesario buscar mas
                }
            }
        }
    }
    return distRadio;
}
```

170- Figura 7.3.3.2 - Algoritmo para obtener los distritos que están dentro de una circunferencia

8. EVALUACIÓN

La evaluación se ha realizado en varias fases:

a) En primer lugar, se hizo un muestreo para conocer qué grado de interés existía por un sistema como el que se pretendía desarrollar. Para ello se creó un Formulario de Google contestado por 160 personas diferentes.

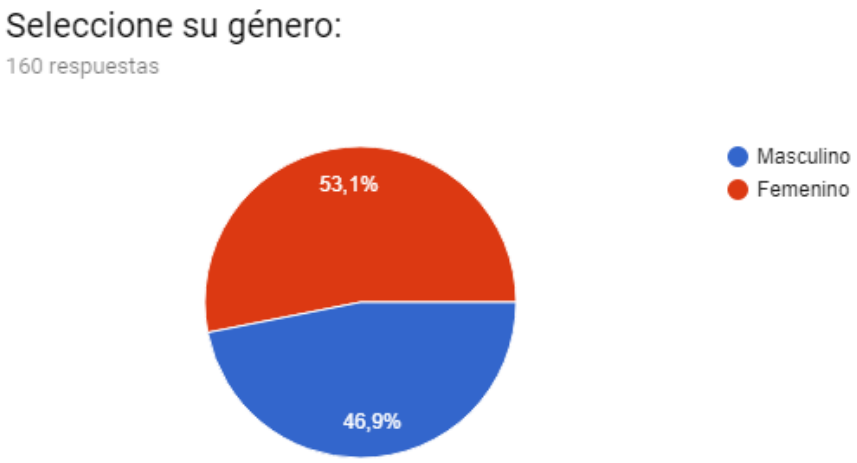
Las preguntas que formaban parte del cuestionario fueron las siguientes:

Pregunta 1

171- Figura 8.a.1 - Pregunta 1

En esta pregunta, el 77,5% de las personas que lo rellenaron se encontraban entre los 18 y los 30 años. Siguiendo con un 11,9% los de 50 años en adelante y continuando con un 10% los de entre 30 y 50 años.

Pregunta 2



172- Figura 8.a.2 - Pregunta 2

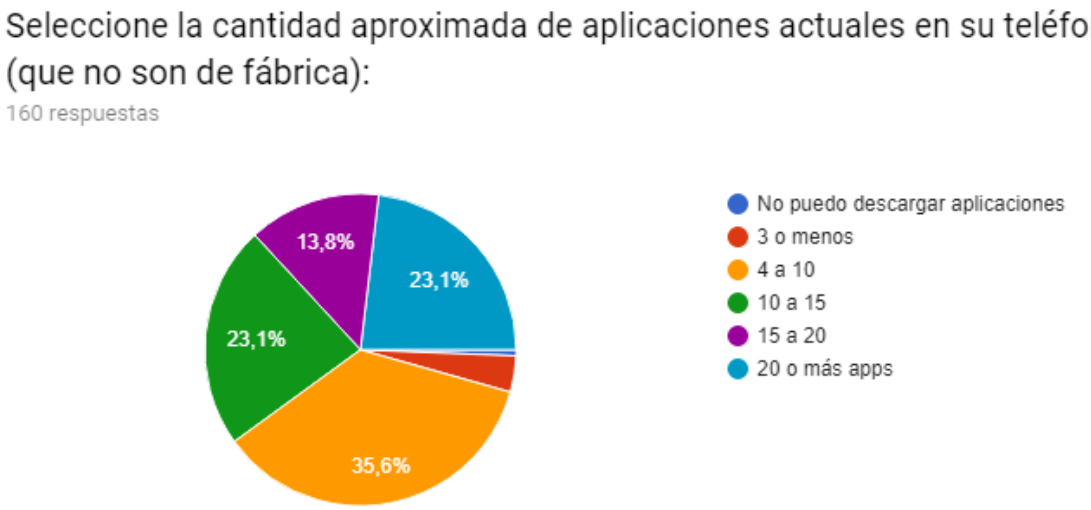
De las 160 respuestas se puede decir que prácticamente la mitad han sido hombres y la mitad mujeres.

Pregunta 3

173- Figura 8.a.3 - Pregunta 3

Todas y cada una de las personas que han rellenado este formulario disponen de un teléfono móvil con conexión a Internet.

Pregunta 4



174- Figura 8.a.4 - Pregunta 4

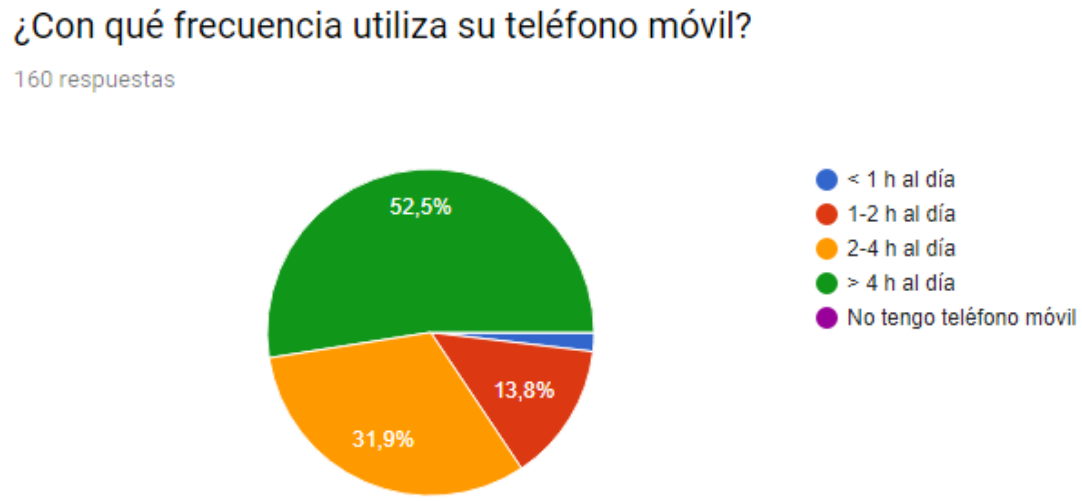
Como se puede observar en esta pregunta, la mayor parte de la gente tiene descargadas al menos 4 aplicaciones en su dispositivo móvil (sin contar con las aplicaciones que vienen ya instaladas).

Pregunta 5

175- Figura 8.a.5 - Pregunta 5

Sólo un 6,9 % (es decir, 11 personas) del total del muestreo diría que no sabe utilizar bien la tecnología.

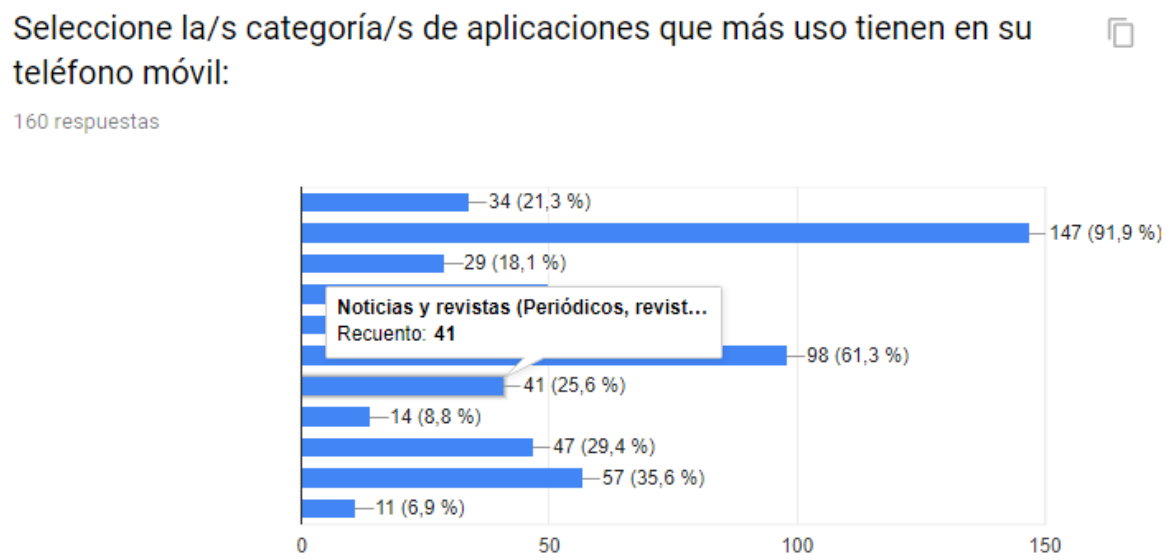
Pregunta 6



176- Figura 8.a.6 - Pregunta 6

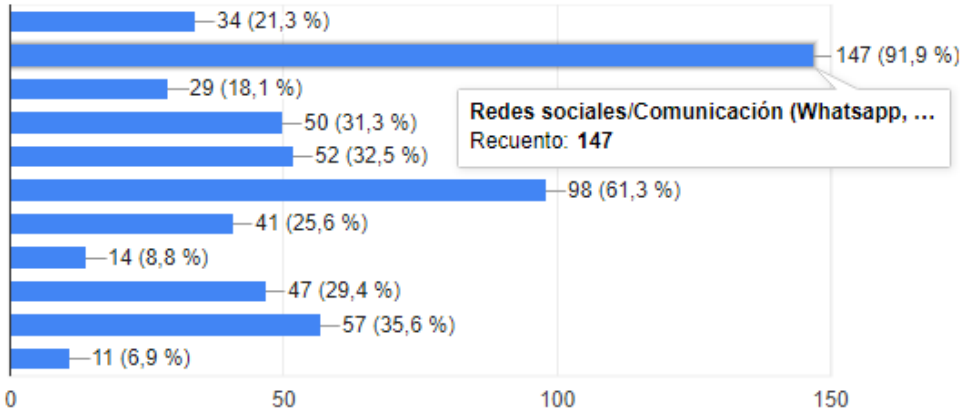
El 84,4 % de la gente que ha votado, utiliza el móvil más de 2 horas al día y concretamente, el 52,5% del total lo utiliza más de 4 horas.

Pregunta 7



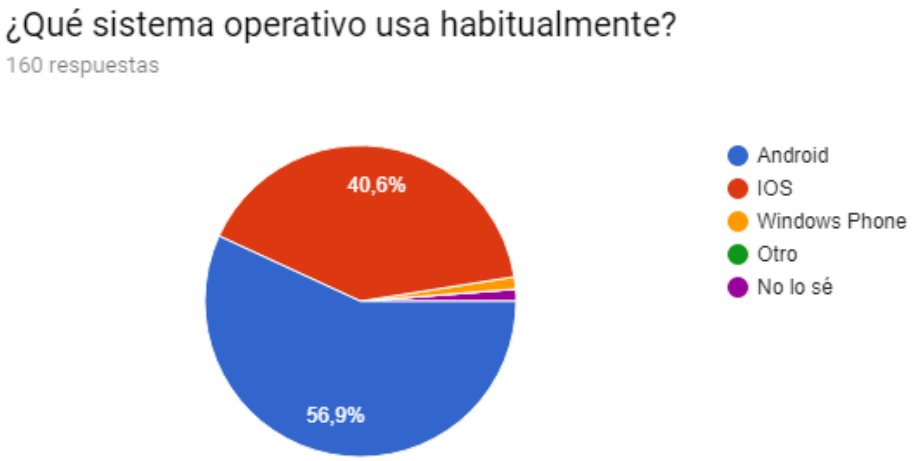
177- Figura 8.a.7 - Pregunta 7 Noticias

La categoría de “Noticias y revistas” ha sido votada por 41 personas como una de las aplicaciones que más uso dan en sus teléfonos móviles. Por otro lado, la categoría más utilizada, con 147 votos, es la de “Redes Sociales”:



178- Figura 8.a.8 - Pregunta 7 Redes Sociales

Pregunta 8



179- Figura 8.a.9 - Pregunta 8

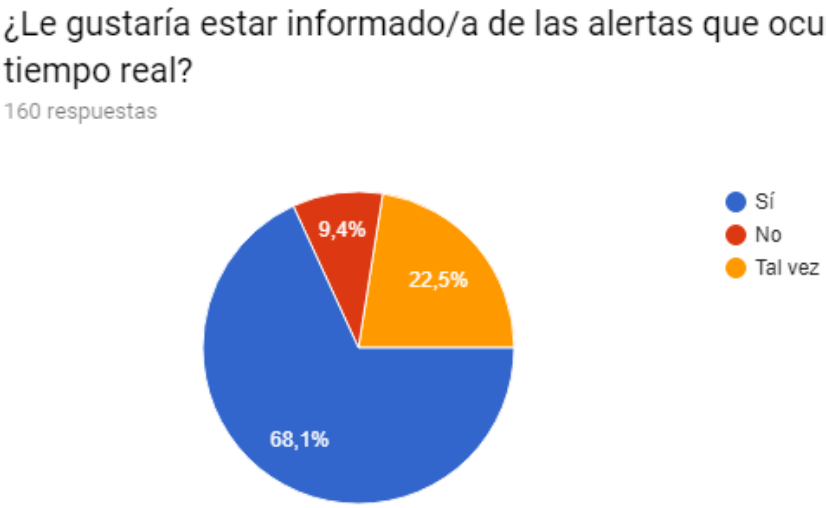
A pesar de estar bastante igualados, se ha obtenido que los usuarios utilizan más el sistema operativo Android que IOS u otros.

Pregunta 9

180- Figura 8.a.10 - Pregunta 9

El 82,5% de los usuarios no suele pagar por las aplicaciones que se descarga.

Pregunta 10



181- Figura 8.a.11 - Pregunta 10

Sólo el 9,4% del total de las respuestas revelan que a este pequeño porcentaje de gente no les gustaría estar informados acerca de las alertas de Madrid a tiempo real.

Pregunta 11

182- Figura 8.a.12 - Pregunta 11

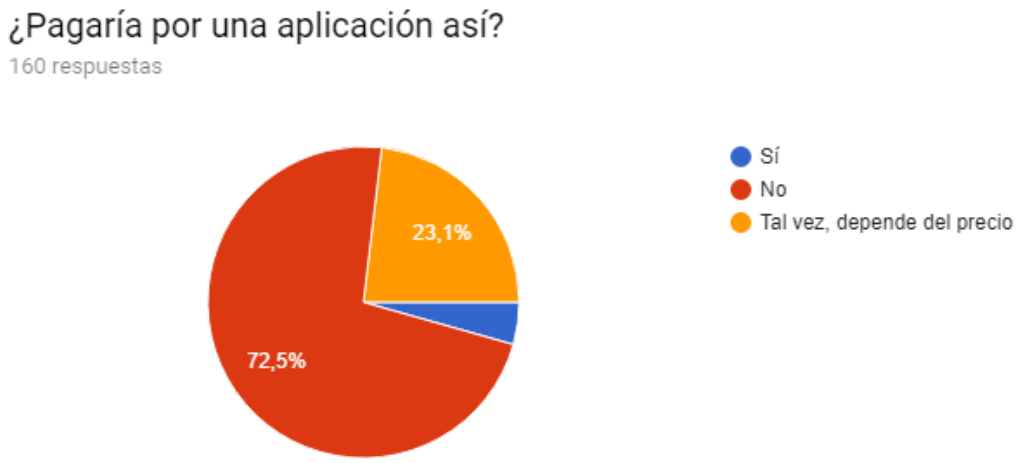
Con esta pregunta podemos afirmar que la mayor parte de la gente desconoce o duda conocer alguna aplicación similar.

Pregunta 12

183- Figura 8.a.13 - Pregunta 12

El 56,3% se descargaría una aplicación así y el 31,3% se lo pensaría. Es decir, sólo el 12,5% del total ni siquiera se plantearía descargarla.

Pregunta 13



184- Figura 8.a.14 - Pregunta 13

Como se puede observar, la mayor parte de la gente no está dispuesta a pagar por las aplicaciones que usan y, por tanto, tampoco lo haría por ésta.

Pregunta 14

185- Figura 8.a.15 - Pregunta 14

En base a las respuestas dadas se puede afirmar que a un 90% de las personas que han rellenado este formulario les parece interesante visualizar estas alertas por distritos, según una categoría etc. Además, al 28,1% de estos les parece muy interesante y no sólo interesante.

Pregunta 15

186- Figura 8.a.16 - Pregunta 15

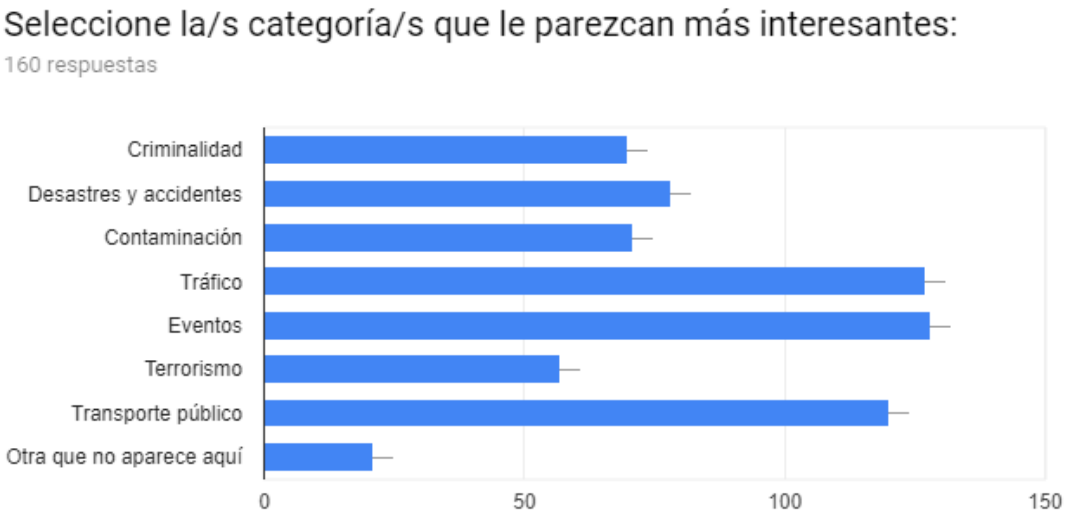
Con esta pregunta se puede afirmar que al 86,9% del total les parecería interesante poderlas visualizar colocadas sobre un mapa de Madrid. Además, al 30% de ellas les parece muy interesante y no sólo interesante.

Pregunta 16

187- Figura 8.a.17 - Pregunta 16

Sin embargo, a la hora de poder visualizar estadísticas de dichas alertas, el número de interesados se reduce al 56% de los votantes. Para un 25,6% del total, le sería irrelevante esta sección.

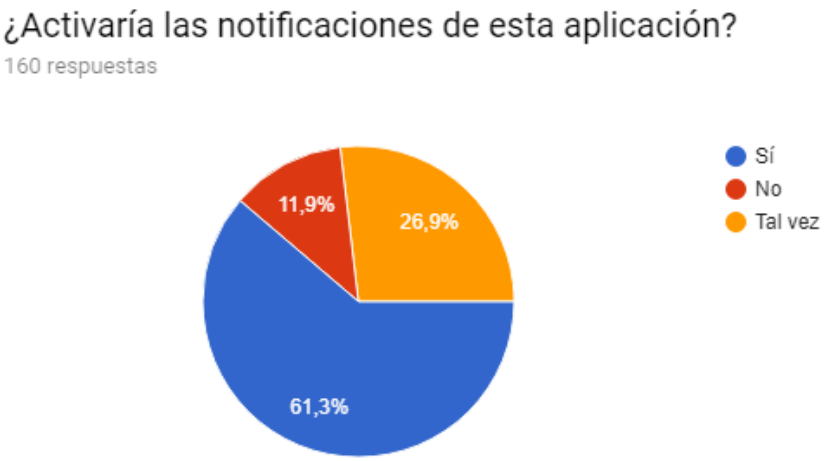
Pregunta 17



188- Figura 8.a.18 - Pregunta 17

Respecto a las categorías que han parecido más interesantes a los usuarios son: Tráfico, Eventos y Transporte Público.

Pregunta 18



189- Figura 8.a.19 - Pregunta 18

El 61,3% de las personas que han respondido a este formulario que sí activarían las notificaciones de esta aplicación y sólo el 11,9% estarían seguros de que no.

Pregunta 19

190- Figura 8.a.20 - Pregunta 19

Con las respuestas dadas, se puede afirmar que el 36,9% de las personas opinan que esta iniciativa es “muy buena” y el 49,4% que es “buena” Es decir, al 86,3% le parece que, en general, es una buena idea crear una aplicación de alertas de Madrid.

Asimismo, se decidió añadir al final del formulario una pregunta abierta preguntando qué es lo que más valoran en una aplicación. Algunas de las respuestas más repetidas fueron: utilidad, efectividad, sencillez, intuitiva, novedosa, fácil de utilizar, que funcione todo bien, accesibilidad, con información veraz y útil en el día a día.

Por consiguiente, como conclusión global y final de este formulario se puede decir que, aunque es cierto que ha habido una participación alta de perfiles de personas jóvenes y con un conocimiento tecnológico medio/alto, parece que es una buena iniciativa.

b) Finalizada la aplicación web se evaluó ésta con 5 usuarios diferentes con el fin de poder encontrar fallos y mejoras. A continuación, se va a mostrar la información recopilada:

Usuario 1

- **1. ¿Cómo te parece de fácil e intuitiva la web (del 1 al 5) ?:** 5
- **2. ¿Qué te parece su diseño (del 1 al 5) ?:** 3
- **3. Aspectos positivos:**
  - Las alertas se ven en pocos pasos.
  - Variedad de noticias.
- **4. Aspectos negativos:**

    - Sólo es posible seleccionar las categorías deseadas en la página del índex, por tanto, en su opinión añadiría en el menú desplegable de la izquierda “Alertas por categorías”.
    - Muy grandes los cuadros de los desplegables de Seleccionar distrito.
- **5. Puntuación global de la web (del 1 al 5):** 3
- **6. Sugerencias de posibles nuevas funcionalidades:**
    - Añadir un apartado relacionado con el clima de Madrid.

Usuario 2

- **1. ¿Cómo te parece de fácil e intuitiva la web (del 1 al 5) ?:** 5
- **2. ¿Qué te parece su diseño (del 1 al 5) ?:** 3
- **3. Aspectos positivos:**
    - Fácil de utilizar.
    - Útil.
- **4. Aspectos negativos:**
    - Al pasar el ratón sobre las categorías cambia de color a azul como si se tratase de un link cuando no lo es.
- **5. Puntuación global de la web (del 1 al 5):** 4

Usuario 3

- **1. ¿Cómo te parece de fácil e intuitiva la web (del 1 al 5) ?:** 4
- **2. ¿Qué te parece su diseño (del 1 al 5) ?:** 4
- **3. Aspectos positivos:**
    - Buen diseño (simple y claro).
    - Intuitiva.
- **4. Aspectos negativos:**
    - Al pinchar sobre un link de una noticia ésta se abre en la misma pestaña. Probando la web este problema le ocasionó perder la página actual y por tanto sugiere que se abra en una pestaña nueva.
- **5. Puntuación global de la web (del 1 al 5)**: 3
- **6. Sugerencias:**
    - Invertir la colocación en el índex del mapa y la selección de distrito y categorías con el fin de visualizarlo al principio sin tener que bajar el scroll.

Usuario 4

- **1. ¿Cómo te parece de fácil e intuitiva la web (del 1 al 5) ?**: 4
- **2. ¿Qué te parece su diseño (del 1 al 5) ?:** 5
- **3. Aspectos positivos**:
    - Útil para estar al corriente de las cosas que suceden en su ciudad.
- **4. Aspectos negativos**:

    - Pinchando sobre el link de una noticia ésta se abre en la misma pestaña (mismo problema que ha encontrado el usuario 3).
    - No existe la opción de visualizar las noticias de todos los distritos de Madrid.
- **5. Puntuación global de la web (del 1 al 5)**: 4

Usuario 5

- **1. ¿Cómo te parece de fácil e intuitiva la web (del 1 al 5) ?:** 5
- **2. ¿Qué te parece su diseño (del 1 al 5) ?**: 4
- **3. Aspectos positivos**:
    - Sencilla.
    - Útil.
- **4. Aspectos negativos**:
    - Ha querido ver las noticias de Tráfico de todo Madrid, pero no existe esta posibilidad en nuestra web. Por consiguiente, sugiere añadir la opción de poder ver noticias de todos los distritos (mismo problema que el usuario 4).
- **5. Puntuación global de la web (del 1 al 5)**: 4

Tras recibir estas evaluaciones y comprobar que existían varios errores que se repetían se decidió realizar una serie de cambios en la web. En primer lugar, se añadió una nueva opción en el desplegable de los distritos llamada “Todos” consiguiendo que fuera posible visualizar las alertas de todos los distritos de Madrid y no sólo de uno concreto. En segundo lugar, se decidió quitar el color azul de las categorías (al pasar el ratón sobre ellas) y así no dar pie a poder confundirlo con un link. El último cambio realizado ha sido que, al pulsar en el link de una noticia, ahora se abre en una pestaña nueva y no en la misma.

c) Evaluación final

Tras dar por finalizada y concluida la página web se decidió realizar una evaluación final a una serie de posibles usuarios. El objetivo principal de esta evaluación era encontrar los puntos fuertes de la web, las cosas que se deberían de cambiar, mejorar o inclusive añadir, si el diseño gusta o no, si es posible realizar todas las funcionalidades ofrecidas con cierta facilidad etc.

Con el fin de poder abarcar todas o la mayoría de las funcionalidades ofrecidas en la web, se decidió que cada persona encuestada debía leer y realizar los siguientes pasos:

“En primer lugar queremos darle las gracias por prestarse a participar en la evaluación final de nuestra página web. Esta evaluación es completamente anónima y nos sirve de gran ayuda para poder encontrar mejoras y fallos de la página.

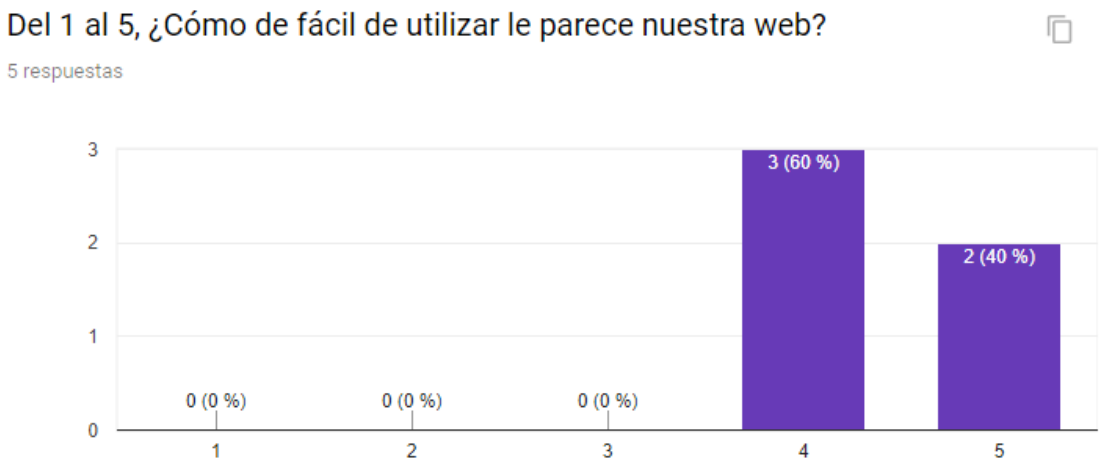
MadAlert se trata de una página de alertas de la Comunidad de Madrid creada para nuestro Trabajo de Fin de grado. Para una correcta evaluación hemos definido una serie de pautas o pasos a seguir con el fin de asegurar que, a lo largo de la navegación, se han utilizado las principales funcionalidades ofrecidas.

1. *Busca alertas utilizando los marcadores del mapa de Madrid.*
2. *Busca las alertas del distrito "Centro".*
3. *Busca las alertas del distrito "Arganzuela" sólo de las categorías "Criminalidad" y "Desastres y accidentes".*
4. *Visualiza las estadísticas de la categoría "Tráfico" de los últimos meses.*
5. *Visualiza las estadísticas del distrito "Salamanca" de los últimos meses.*
6. *Visualiza las estadísticas de la Policía Municipal de Madrid.*
7. *Lee información acerca de nuestra página.*

Muchas gracias. A continuación, se realizarán una serie de preguntas acerca de su experiencia en nuestra web. "

Después de que el usuario hubiera realizado todas estas pruebas y hubiera indagado un rato por la web de MadAlert, se realizó un formulario de Google (anónimo) que debían rellenar en sus teléfonos móviles. A continuación, se van a mostrar cuáles fueron las preguntas realizadas, así como las respuestas obtenidas en cada una de ellas:

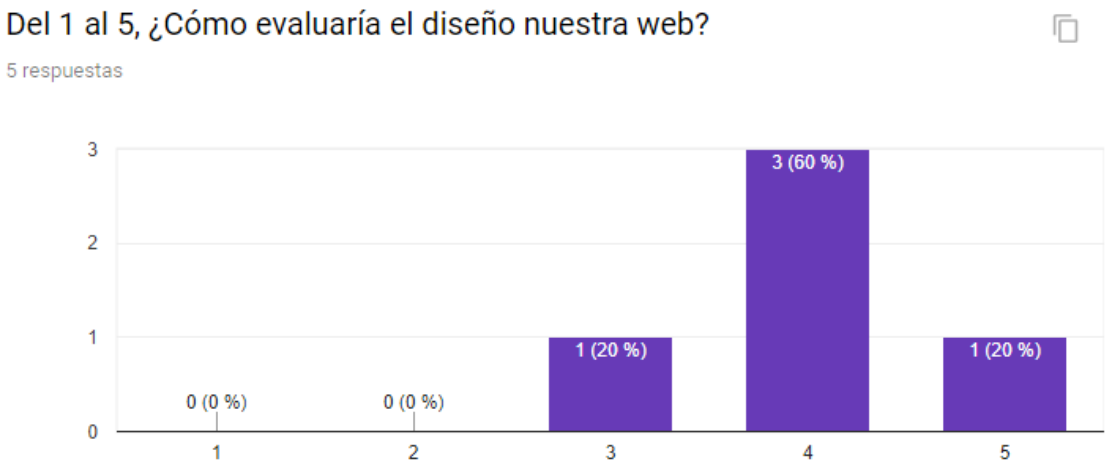
Pregunta 1



191- Figura 8.c.1 - Pregunta 1

Con esta pregunta se puede afirmar que la página de MadAlert se trata de una página sencilla y fácil de utilizar. El 60%, es decir, 3 de las respuestas puntúan la facilidad de uso con un 4 sobre 5 y las 2 respuestas restantes (40%) con un 5 sobre 5.

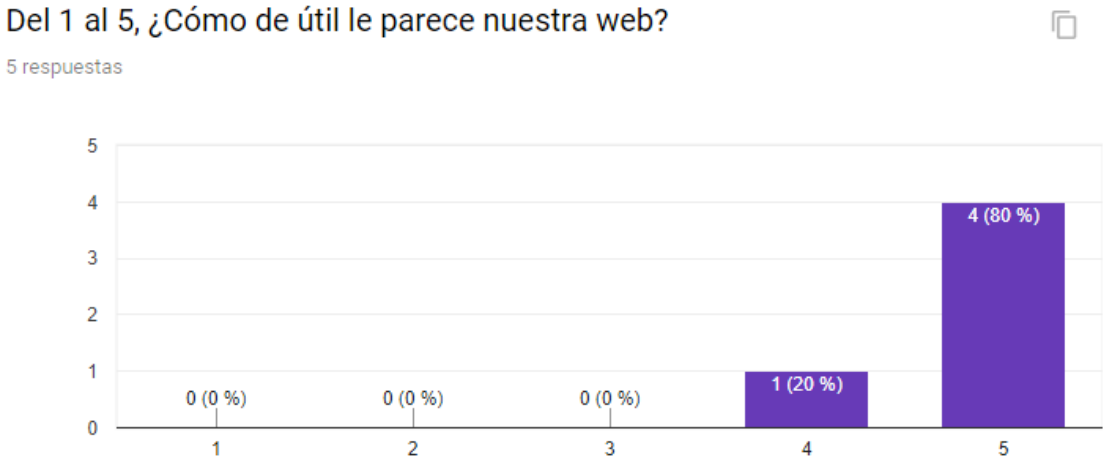
Pregunta 2



192- Figura 8.c.2 - Pregunta 2

Respecto al diseño de nuestra web, se puede decir que no parece que haya resultado una página con un mal diseño. Sólo uno de los encuestados ha contestado que el diseño no le resultó ni bueno ni malo (puntuación de 3 sobre 5) mientras que, de los 4 restantes, 3 opinan que es un buen diseño (puntuación de 4 sobre 5) y por último, otro que muy bueno (puntuación de 5 sobre 5).

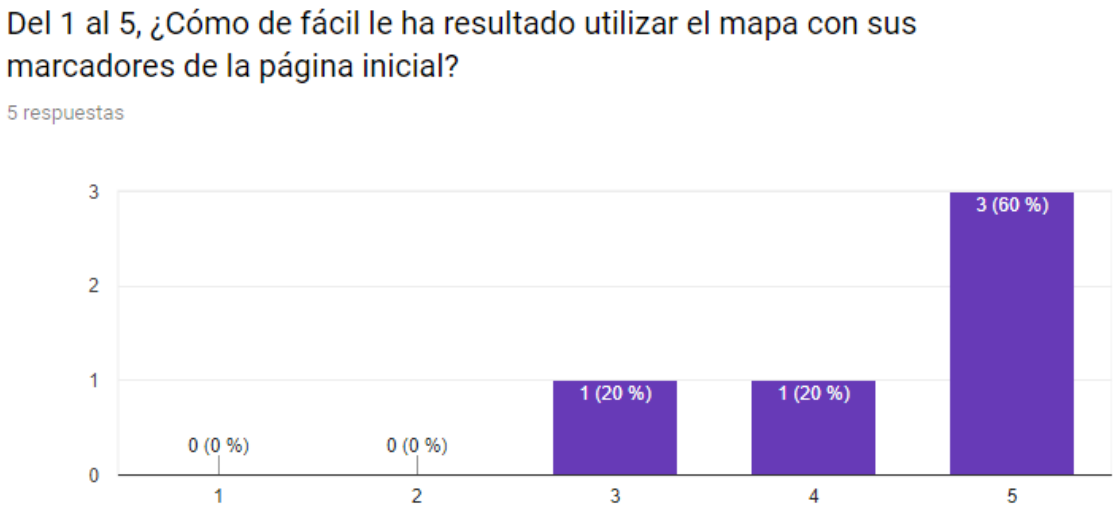
Pregunta 3



193- Figura 8.c.3 - Pregunta 3

Con esta pregunta se puede decir que nuestra página resulta útil a los usuarios. Tal y como se ve en las respuestas, le ha parecido útil a una de las personas encuestadas (puntuación de 4 sobre 5) y muy útil a las 4 restantes (puntuación de 5 sobre 5).

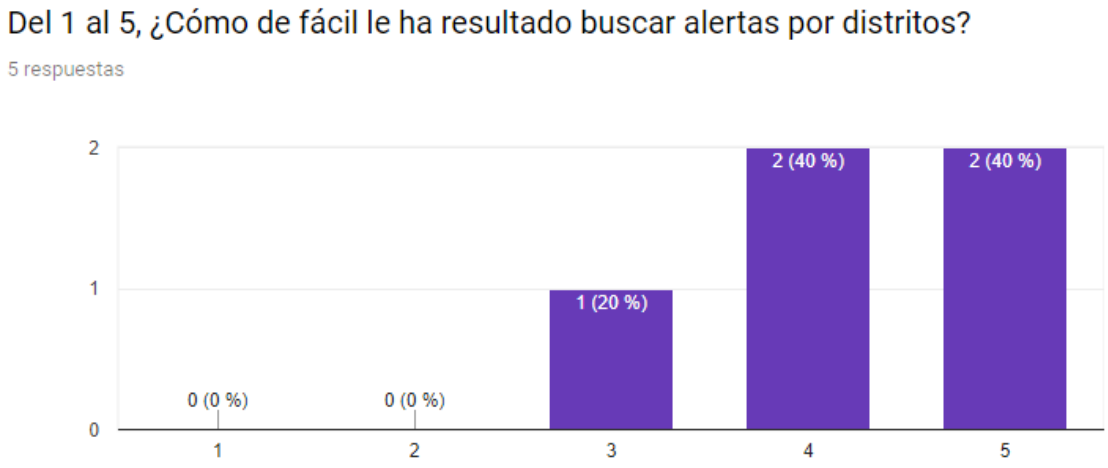
Pregunta 4



194- Figura 8.c.4 - Pregunta 4

Con esta pregunta se puede comprobar que no parece que haya resultado difícil utilizar el mapa de la página inicial. Se puede afirmar que sólo uno de los encuestados ha contestado que no le resultó ni fácil ni difícil (puntuación de 3 sobre 5) mientras que, de los 4 restantes, 1 opinan que fue fácil (puntuación de 4 sobre 5) y otros 3 que muy fácil (puntuación de 5 sobre 5).

Pregunta 5

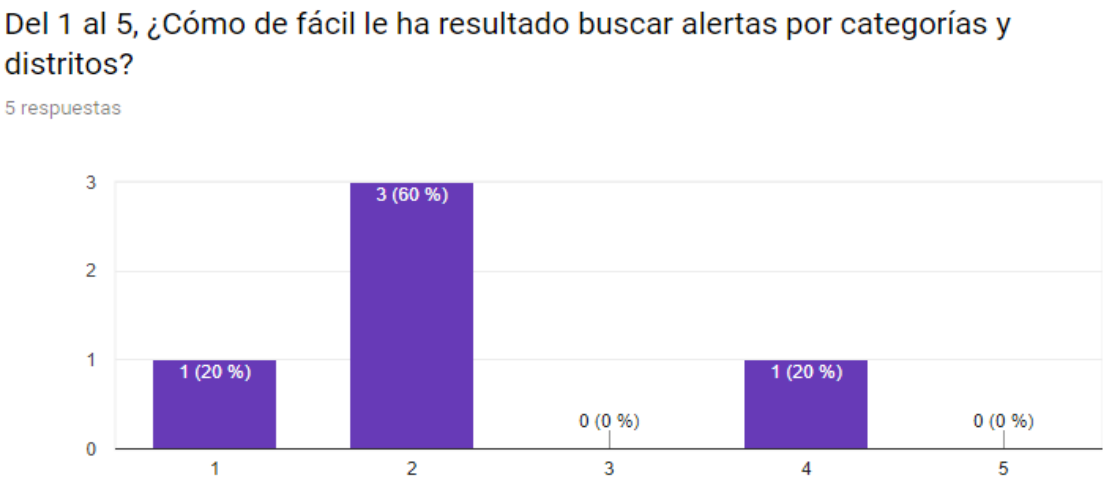


195- Figura 8.c.5 - Pregunta 5

Con esta pregunta se puede comprobar que no parece que haya resultado difícil realizar la búsqueda de alertas por distritos. Sólo uno de los encuestados ha contestado que no le resultó ni fácil ni difícil (puntuación de 3 sobre 5) mientras que, de los 4 restantes, 2

opinan que fue fácil (puntuación de 4 sobre 5) y otros 2 que muy fácil (puntuación de 5 sobre 5).

Pregunta 6

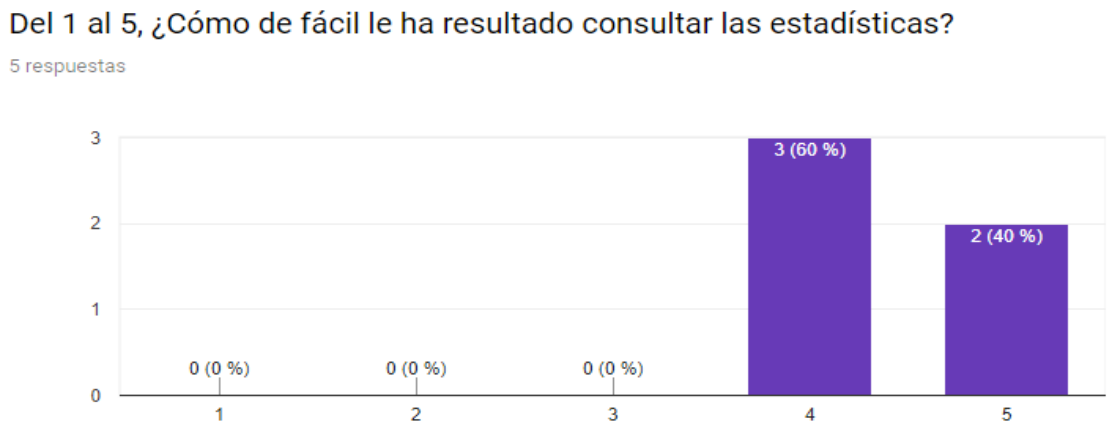


196- Figura 8.c.6 - Pregunta 6

Gracias a esta pregunta se ha detectado un problema a la hora de encontrar y realizar la búsqueda de alertas por distritos y por categorías. Sólo una de las personas que han realizado la evaluación ha sido capaz de realizarla sin apenas dificultad.

Esto claramente lleva a pensar que se trata de un fallo grave y que, por tanto, se debería intentar encontrar una solución e inclusive, si el tiempo lo permite, realizar los cambios pertinentes.

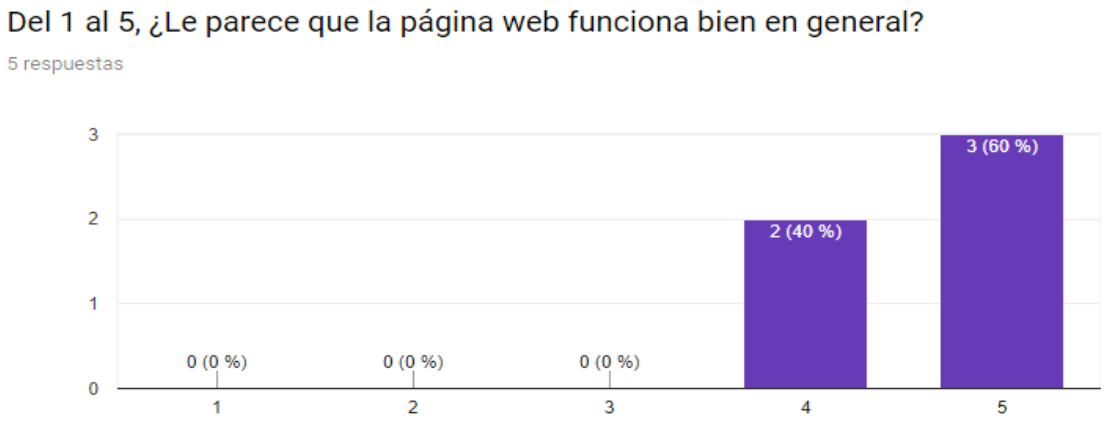
Pregunta 7



197- Figura 8.c.7 - Pregunta 7

Un total de 3 de las 5 respuestas puntúan con un 4 sobre 5 la facilidad a la hora de consultar las estadísticas mientras que el 40% restante, es decir, las otras dos respuestas, con un 5 sobre 5.

Pregunta 8



198- Figura 8.c.8 - Pregunta 8

En 3 de las 5 respuestas obtenidas se puntúa con un 5 sobre 5 el funcionamiento de la página de MadAlert y, en cambio, las 2 restantes con un 4 sobre 5.

Pregunta 9

199- Figura 8.c.9 - Pregunta 9

En el 100% de las respuestas obtenidas se afirma que tanto los enlaces como los botones se identifican con facilidad.

Pregunta 10

200- Figura 8.c.10 - Pregunta 10

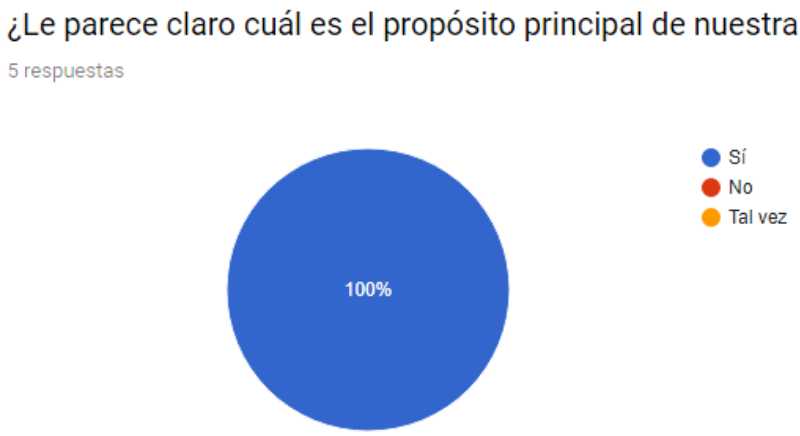
En un total de 4 de las 5 respuestas se afirma haber leído y visualizado todo correctamente y sólo el 1 restante dice que tal vez.

Pregunta 11

201- Figura 8.c.11 - Pregunta 11

Al 100% de las personas que han realizado esta evaluación les parece más interesante poder descargar y utilizar una app antes que utilizar la página web.

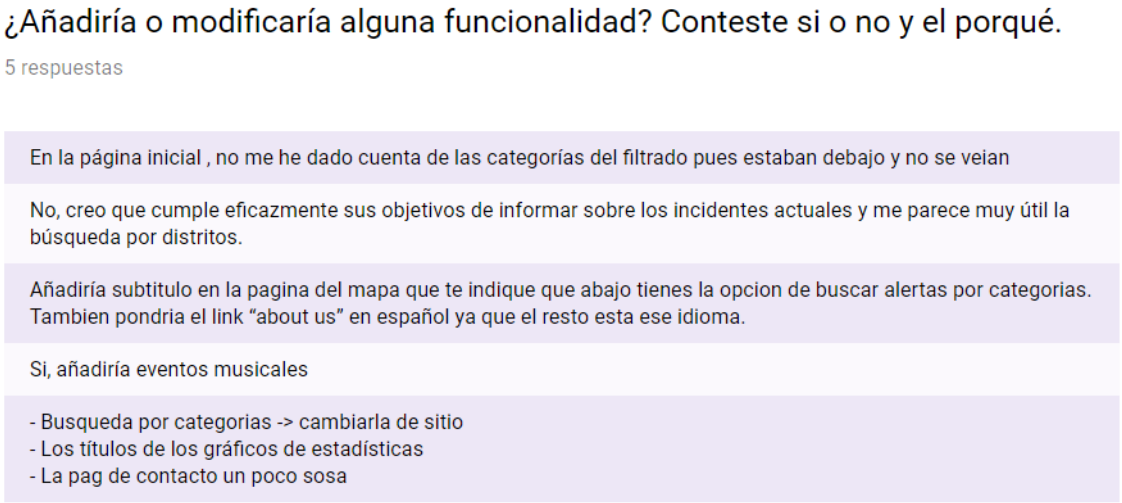
Pregunta 12



202- Figura 8.c.12 - Pregunta 12

Al 100% de las personas que han realizado esta evaluación les parece que queda claro cuál es el propósito principal.

Pregunta 13



203- Figura 8.c.13 - Pregunta 13

En esta última pregunta se ha dejado libertad a los usuarios para responder de manera breve cuáles son sus sugerencias.

Tras observar la repetición del problema al encontrar en qué parte de la página se encuentra la búsqueda de alertas por categorías, se ha decidido plantear una posible solución. Ésta consistiría en crear subpestañas en el apartado de “Alertas” (del menú lateral izquierdo) y así ser posible acceder desde ahí y no sólo desde la parte inferior del mapa (en la página de inicio). Esta solución se encuentra explicada en el apartado de “Trabajo Futuro” de esta memoria.

De cara a la presentación de este Trabajo de Fin de Grado, se realizará de igual forma una evaluación final de la app móvil. Será el día de la presentación cuando, se comentarán los resultados obtenidos y por consiguiente las conclusiones. Esta evaluación se encontrará en la carpeta llamada “Evaluación app MadAlert” dentro del Google Drive correspondiente a este proyecto.

9. CONCLUSIONES Y TRABAJO FUTURO

9.1 CONCLUSIONES

Una vez finalizado este trabajo y tras haber concluido la memoria, al fin somos conscientes de lo que supone crear desde cero un proyecto de esta magnitud. Se trata de una ardua tarea que requiere un sacrificio constante, una meticulosa organización y sobre todo, una buena comunicación entre los miembros del proyecto. Esto último es algo que se quiere remarcar, pues gracias a la amistad que hay entre nosotros y el afán de cada uno por colaborar ha sido algo fundamental.

Es cierto que hubo momentos en los que vivimos complicaciones, pero con el esfuerzo de todos sumado a nuestra experiencia, pudimos salir no solo del paso, sino que adquirimos nuevos conocimientos que nos ayudan a crecer.

El objetivo final de este proyecto era crear una herramienta fácil de usar y amigable para el usuario. Mantenerse constantemente informado acerca de los acontecimientos que nos rodean se ha convertido en algo imprescindible en la sociedad actual y esto obliga a las nuevas tecnologías a crear herramientas que ayuden a ello. Esta era la meta final, contribuir de algún modo a la sociedad.

En definitiva, un proyecto de estas dimensiones requiere siempre del aprendizaje de multitud de nuevas herramientas y lenguajes. Cabe mencionar también el constante avance que experimenta la tecnología y la influencia que posee en la vida cotidiana. Es de gran importancia, por tanto, estar dispuesto en todo momento a seguir aprendiendo y a evolucionar como desarrolladores. Esto ha sido una motivación constante que se todos los integrantes del equipo han tenido.

Teniendo en cuenta lo establecido, creemos que hemos cumplido plenamente con lo que nos propusimos al inicio de este proyecto y, aunque existen diversas mejoras, la base de ello ya ha sido cimentada y creemos que es una buena idea continuar con ello.

9.2 CONCLUSIONS

Once this work is finished and after having finished the memory, we are finally aware of what it means to create a project of this magnitude from scratch. It is an arduous task that requires a constant sacrifice, a meticulous organization and above all, a good communication between the members of the project. This is something that we want to point

out, because due to the friendship that exists between us and the spirit of collaboration has been something fundamental.

It is true that there were times when we experienced some complications but with the effort of all added to our experience, we were able to leave not only the step, but we acquired new knowledge that helps us grow.

The ultimate goal of this project was to create an easy-to-use and user-friendly tool. Keeping yourself constantly informed about the events that surround us has become essential in today's society and this forces new technologies to create tools to help it. This was the final objective, to contribute in some way to society.

Ultimately, a project of these dimensions, always requires learning a great quantity of new tools and languages. It is also worth mentioning the constant progress that technology is experiencing and the influence it has in everyday life. It is therefore of major importance to be willing at all times to continue learning and to evolve as developers. This has been a constant motivation that all the members of the team have had.

Taking into account the established, we believe that we have fully complied with what we proposed at the beginning of this project and, although there are several improvements, the basis of this has already been laid down and we believe that it is a good idea to go on with it.

9.3 TRABAJO FUTURO

A continuación, se van a explicar algunas posibles mejoras o nuevas funcionalidades que se podrían implementar de cara al futuro:

Para ambas aplicaciones:

- **Más fuentes**: Obtener información no sólo de diversas cuentas de Twitter y del periódico Madridiario sino también de más periódicos online y más cuentas oficiales de Twitter.

- **Multilenguaje**: Actualmente, tanto la web como la app sólo se encuentran disponibles en castellano. Una posible ampliación futura sería que ambas aplicaciones fueran multilenguaje.

- **Abarcar más ciudades**: En un principio se decidió implementarlo exclusivamente para la ciudad de Madrid, pero podría ser interesante ampliarlo a más ciudades de España e inclusive también de fuera de España.

- **Actualización automática:** Tal y como está implementado a día de hoy, para poder realizar las actualizaciones se debe pulsar un botón de “Actualizar”. Estaría bien que dichas actualizaciones se realizaran también cada cierto tiempo automáticamente.

  Para la aplicación web:

- **Servidor**: Este proyecto no cuenta con un servidor en la nube como tal pero si estaría preparado para poder estar alojado en uno de ellos.

- **Subpestañas**: Con el fin de poder encontrar más fácilmente las diversas opciones ofrecidas a la hora de realizar la búsqueda de alertas, se podría poner en el menú lateral izquierdo varias subpestañas dentro de la pestaña de “Alertas”. De esta forma se solventaría el problema encontrado al realizar la evaluación final con usuarios y sería mucho más intuitivo y sencillo. Al fin y al cabo, en esta página la sencillez es una de las prioridades a seguir.

- **Estadísticas anuales**: Añadir la posibilidad de visualizar estadísticas anuales y no sólo de los dos meses anteriores al actual.

  Para la aplicación móvil:

- **Notificaciones**: Para los usuarios que tuvieran activadas las notificaciones, se mandarían notificaciones push con las nuevas alertas que coincidan con sus preferencias.

10. TRABAJO INDIVIDUAL

En este último apartado se describe la aportación de cada miembro al proyecto. Para el desarrollo de este trabajo ha sido necesaria la colaboración de todos ellos para hacer frente a los obstáculos que han ido apareciendo a lo largo del transcurso del mismo.

Las diferentes tareas del proyecto se han ido repartiendo conforme se avanzaba, tratando de hacer un reparto equitativo en base a los conocimientos de cada integrante, sin por ello desatender la posibilidad de ayudar a un compañero ni ignorar las tareas que eran "ajenas" a cada uno.

10.1 NEREA GÓMEZ DOMÍNGUEZ

Desde el principio de este proyecto, me ha parecido muy interesante la idea que queríamos llevar a cabo, así como la oportunidad de aprender nuevas tecnologías y herramientas totalmente desconocidas para mi que han aportado gran valor a mi formación. Ha sido largo y en ocasiones duro al ver como no avanzaban las cosas en el tiempo deseado, pero entre todos hemos sido capaces de superarlo y sacarlo adelante.

Durante el transcurso del proyecto he intentado colaborar en todo momento, aportando mis conocimientos y dedicando el tiempo necesario, así como ayudando a mis compañeros cuando lo necesitaban y aprendiendo lo necesario para poder seguir adelante.

Respecto al desarrollo del recolector de datos hice con mi compañero Adrián la obtención de Tweets de los usuarios deseados en tiempo real usando la API de Twitter, asi como su filtrado utilizando las API de los clasificadores correspondientes, fue bastante interesante y observé la cantidad de datos que se podían obtener y analizar. En la aplicación web, me encargue de conectar MongoDB con PHP, así como hacer el traspaso de conexión de BD a mLab. También realicé parte de la pestaña de alertas, en concreto la búsqueda de alertas por distrito, así como las estadísticas por distritos usando Google Charts. Apliqué mis conocimientos de JavaScript para realizar el control de errores en el formulario de añadir alertas y la parte de procesado junto mi compañero Adrián. Por último, conecte la API de meaning Cloud con php para poder categorizar las alertas que añadían los usuarios.

En cuanto a Android, tuve que aprender desde 0, ya que no tenía conocimientos previos sobre este lenguaje. Gracias a las competencias adquiridas en la carrera y en mi trabajo actual sobre Java, supuso más fácil llevar a cabo este aprendizaje, aunque en ocasiones resultó complicado. Los que nos dedicamos a desarrollar esta parte hemos colaborado de igual forma y ayudándonos en todo momento, ya que quedábamos siempre

para que fuera mucho más rápido y fácil avanzar. En especial he colaborado en la creación de la API para conectar Android con MongoDB y en el algoritmo de obtener distritos.

En lo que respecta a esta memoria, he sido la encargada de realizar la mayoría de los diagramas de actividades y el diseño de la arquitectura de la aplicación. También he colaborado en la implementación de la API y de Android, así como el manual de usuario.

Estoy bastante satisfecha por el resultado final de este proyecto y por haberlo podido realizar con mis compañeros. Ha sido un gran reto que ha merecido la pena tanto a nivel profesional como personal.

10.2 SILVIA LENDÍNEZ FERNÁNDEZ

En primer lugar, me gustaría dar las gracias a mis 3 compañeros y sobre todo, amigos, por el trabajo y el esfuerzo realizado desde el mes de Octubre. Todo este proyecto no habría sido posible de no ser por ellos y el gran trabajo en equipo realizado.

Todos y cada uno de nosotros hemos puesto un gran empeño y hemos trabajado siempre teniendo como lema el compañerismo. Hemos ido quedando prácticamente todas las semanas para trabajar conjuntamente y cara a cara ya que nos parecía la mejor manera de avanzar y tener una buena comunicación entre nosotros (fundamental en proyectos de tal envergadura). Además, es un orgullo decir que siempre hemos estado ayudándonos y motivándonos los unos a los otros hasta en los peores días.

Hemos colaborado conjuntamente en la mayor parte del proyecto, pero es cierto que, como es lógico, ha sido necesario dividir y repartir tareas. A continuación, voy a intentar aclarar cuál ha sido mi aportación a este Trabajo de Fin de Grado.

A grandes rasgos, en lo que respecta a la aplicación web, es decir, a la primera parte de este proyecto, he sido la encargada de pensar nombres para la aplicación, diseñar el logo junto a mi compañero Adrián, obtener alertas del periódico Madridiario[58] con mi compañero Gonzalo (utilizando la técnica del web scraping), parte de la búsqueda de alertas, realizar lo correspondiente a las estadísticas generales (aprendiendo a utilizar Google Charts), hacer posible tanto la actualización del mapa como la actualización de las alertas y crear los diferentes ejecutables (para realizar un Cron y así hacer que la inserción de alertas en la base de datos fuera automática) y programas de borrado (para limpiar la base de datos de alertas).

Una vez pasamos a la siguiente parte de este proyecto, decidimos dividirnos. Mientras yo completaba todo lo correspondiente a la parte web de esta memoria, realizaba la evaluación de la misma con usuarios y corregía algunos fallos encontrados, mis

compañeros se encargaron de comenzar a programar la aplicación para Android. Así conseguimos avanzar mucho más rápido y evitamos dejar toda esta memoria para el final.

Ya, por último, como parte final del proyecto, hemos hecho entre todos un último esfuerzo y hemos acabado de completar esta memoria.

Como conclusión, he de decir que estoy muy satisfecha con el resultado de este proyecto y con haber tenido los compañeros que he tenido. Asimismo, puedo decir que todo esfuerzo tiene su recompensa y gracias a este proyecto me ha sido posible aprender a utilizar nuevas tecnologías y lenguajes que seguro me serán de gran utilidad en un futuro.

10.3 GONZALO MOLINA DÍAZ

Antes de nada, me gustaría agradecer a todos mis compañeros el esfuerzo realizado durantes estos meses. Ha sido un proyecto largo, pero del que estamos satisfechos y en el que todos hemos contribuido del mismo modo.

En la realización del proyecto he intentado aportar todos mis conocimientos y virtudes, intentando también aprender nuevas cosas para así contribuir en la medida de los posible.

En lo que respecta a la aplicación web, mi papel fundamental fue la realización de las estadísticas de la Policía, tanto su vista como su backoffice. Fue una ardua tarea aprender una nueva herramienta como era web scraping, pero fue un bonito reto. Tras extraer los datos procedentes del excel, fue turno de plasmarlos en la web, por lo que tuve que aplicar mis conocimientos de Javascript y conocí la aplicación de Google Charts.

En la segunda parte del proyecto tuve que aprender totalmente desde 0 a programar en Android. Fue complicado al principio, pero con los conocimientos adquiridos en la carrera acerca del lenguaje Java, en el momento en el que cogí práctica y me familiaricé con el entorno de Android Studio fue mucho más rodado. Ayudé prácticamente en todos los aspectos de la implementación Android, pero sobre todo en el diseño principal, el menú navegable izquierdo y las actividades que este contiene.

La memoria fue un trabajo conjunto en el que tratamos de plasmar cada uno en lo que más habíamos trabajado.

Personalmente, me encuentro muy satisfecho con el proyecto en sí y con mi aportación al mismo. Ha sido un bonito reto y he aprendido muchas cosas durante estos meses.

10.4 ADRIÁN PANADERO GONZÁLEZ

Para empezar, me gustaría dar las gracias y felicitar a mis tres compañeros y amigos por todo el trabajo realizado estos meses, juntos hemos conseguido realizar este proyecto que tanta ilusión nos hacía.

En lo que respecta al desarrollo de la aplicación me he encargado de diseñar el logo junto a mi compañera Silvia, y la recolección y filtrado de información de Twitter con mi compañera Nerea. En la parte de la aplicación web he realizado la página inicial donde se encuentra el mapa con los marcadores y la información de las alertas que tienen, además de la búsqueda de alertas por distrito y categorías. También he realizado el diseño de la web haciéndola “responsive” y modificándola para que fuera lo más cómoda y simple para los usuarios.

En cuanto la parte del desarrollo de la aplicación móvil, me encargué de buscar el sitio en la nube que al final fue mLab y de buscar las maneras de realizar la conexión de esa base de datos con Android junto a Nerea. Como ningún integrante del grupo sabía Android, por lo que esto era nuevo para nosotros, y tuvimos que dedicarle mucho tiempo a la búsqueda de información sobre la conexión. Al final, encontramos el uso de Retrofit y Node.js para llevar a cabo la conexión de la base de datos con mi compañera Nerea.

También me he encargado de hacer los fragmentos de distritos, de mapa colaborando con Gonzalo y Nerea, y el diseño de listar las alertas y añadir alertas con Nerea.

Por último, en la parte de la memoria he realizado los diagramas de actividades y casos de uso, la implementación Android y el manual de usuario Android colaborando con mis compañeros.

A nivel personal, estoy muy satisfecho con el trabajo realizado y todo lo que he podido aportar al proyecto y a mis compañeros. He aprendido nuevos lenguajes, nuevas tecnologías, que no sabía de su existencia, y me han aportado una nueva visión de la cantidad de trabajo que tiene desarrollar proyectos de este estilo. Pienso que ha sido todo un reto y que ha merecido la pena todo el tiempo invertido en realizarlo.

BIBLIOGRAFÍA

[1] Alvarez, D (2017): "Pull To Refresh en tu lista, RecyclerView en Android". *Medium*, 4 de Septiembre. Disponible en: https://medium.com/alvareztech/pull-to-refresh-en-tu-lista-recyclerview-en-android-9f2ce5657b30 [Consulta: Mayo 2018]

[2] Amal, R (2016): "Android User Registration and Login with Node.js and MongoDB – Client #2". *Learn2Crack*, 24 de Septiembre. Disponible en: https://www.learn2crack.com/2016/09/android-user-registration-login-node-client.html [Consulta: Marzo 2018]

[3] Amal, R (2016): "Android User Registration and Login with Node.js and MongoDB – Server #1". *Learn2Crack*, 24 de Septiembre. Disponible en: https://www.learn2crack.com/2016/09/android-user-registration-login-node-server.html [Consulta: Marzo 2018]

[4] Amal, R (2016): "Android working with RxJava 2 and Retrofit". *Learn2Crack*, 28 de Noviembre. Disponible en: https://www.learn2crack.com/2016/11/android-rxjava-2-and-retrofit.html [Consulta: Marzo 2018]

[5] Android Developers (2018): *Android Developers*. Disponible en: https://developer.android.com/studio/intro/?hl=es [Consulta: Marzo 2018]

[6] Android Developers (2018): *Principios de diseño para Android*. Disponible en: https://developer.android.com/design/get-started/principles?hl=Es [Consulta: Marzo 2018]

[7] Ardións, A (2015): "Cómo exportar un APK en Android Studio", *Android Studio FAQs*, 16 de Octubre. Disponible en: https://androidstudiofaqs.com/tutoriales/como-exportar-un-apk-en-android-studio [Consulta: Abril 2018]

[8] Ardións, A (2015): "Cómo instalar APK desde PC vía USB". *Android Studio FAQs*, 3 de Noviembre. Disponible en: https://androidstudiofaqs.com/root-rom/como-instalar-apk-en-android-desde-pc [Consulta: Abril 2018]

[9] Ardións, A (2017): "Código para “enviar email” en Android Studio". *Android Studio FAQs*, 9 de Enero. Disponible en: https://androidstudiofaqs.com/tutoriales/codigo-enviar-email-android-studio [Consulta: Mayo 2018]

[10] Azaustre, C (2013): "Cómo crear un API REST usando Node.js, Express y MongoDB". *Carlos Azaustre Formación JS*, 5 de Junio. Disponible en: https://carlosazaustre.es/como-crear-una-api-rest-usando-node-js/ [Consulta: Marzo 2018]

[11] Balsamiq (2018): *Download Mockups 3*. Disponible en: https://balsamiq.com/download/ [Consulta: Octubre 2017]

[12] baugarten, (2017): *A library for quickly providing a REST API with express or connect*. Disponible en: https://github.com/baugarten/node-restful [Consulta: Marzo 2018]

[13] BBVAOPEN4U (2016): "API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos". *bbvaopen4u*, 23 de Marzo. Disponible en: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos [Consulta: Marzo 2018]

[14] Betancur, J (2013): "Creación de un Cron Job en Windows 7", *7Sabores.com*, 17 de Junio. Disponible en: http://7sabores.com/blog/creacion-cron-job-windows-7 [Consulta: Mayo 2018]

[15] Bootstrap [En línea, v4]. Disponible en: https://getbootstrap.com/ [Consulta: Noviembre 2017]

[16] Bouhier, M, FJSevilla. (2017). Extracción datos Excel para crear lista Python. *Stack Overflow* [Foro de Internet]. Disponible en: https://es.stackoverflow.com/questions/113620/extraccion-datos-excel-para-crear-lista-python [Consulta: Diciembre 2017]

[17] Bradley, S. (2014): "Design Principles: Visual Perception And The Principles Of Gestalt". *Smashing Magazine*, 29 de Marzo. Disponible en: https://www.smashingmagazine.com/2014/03/design-principles-visual-perception-and-the-principles-of-gestalt [Consulta: Abril 2018]

[18] Caballero, R (2018) Bases de datos NoSQL - UCM, Rafael Caballero Roldán, Curso 2017-2018. Disponible en: http://gpd.sip.ucm.es/rafa/docencia/nosql/index.html [Consulta: Octubre 2017]

[19] Caraballo, D (2016): "Web Scraping con Python y BeautifulSoup", *Mi diario Python*, 8 de Octubre. Disponible en: http://www.pythondiario.com/2016/08/web-scraping-con-python-y-beautifulsoup.html [Consulta: Octubre 2017]

[20] Codevolution (2016): *MongoDB PHP Tutorial - 7 - Limit, Skip and Sort*. [Video online]. Disponible en: https://www.youtube.com/watch?v=mNcx8zKN4UA [Consulta: Noviembre 2017]

[21] Cosmos (2018): "Solo el 4,6% de los dispositivos Android están actualizados a la versión 8.0/8.1 Oreo", *xatakandroid.com*, 18 de Abril. Disponible en: https://www.xatakandroid.com/mercado/solo-el-4-6-de-los-dispositivos-android-estan-actualizados-a-la-version-8-0-8-1-oreo [Consulta: Abril 2018]

[22] Crespo, E (2013): "Mostrar tweets en tiempo real con twython y el API Stream de Twitter". *Blog de Ernesto Crespo*, 26 de Septiembre. Disponible en: https://ernestocrespo13.wordpress.com/2013/09/26/mostrar-tweets-en-tiempo-real-con-twython-y-el-api-stream-de-twitter/ [Consulta: Octubre 2017]

[23] Datos abiertos (2018): *Policía Municipal. Datos estadísticos actuaciones Policía Municipal*. Disponible en: https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=bffff1d2a9fdb410VgnVCM2000000c205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD [Consulta: Enero 2018]

[24] Del Valle E (2014): "Ley de Fitts: la piedra angular del diseño de interacción web", *socialmediaycontenidos.com*, 1 de Junio. Disponible en: http://www.socialmediaycontenidos.com/ley-de-fitts-la-piedra-angular-del-diseno-de-interaccion-web [Consulta: Octubre 2017]

[25] Developer Twitter (2018): *Filter realtime Tweets*. Disponible en: https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/basic-stream-parameters [Consulta: Noviembre 2017]

[26] Developeru (2016): *"Menú Lateral en Android Studio (Navigation Drawer)"*, [Vídeo online]. Disponible en: https://www.youtube.com/watch?v=bZW8GP9MTEQ [Consulta: Marzo 2018]

[27] dougwilson, (2018): *Node.js body parsing middleware*. Disponible en: https://github.com/expressjs/body-parser [Consulta: Marzo 2018]

[28] Draw.io: (2017): *github.com/jgraph/drawio*, 22 de Mayo. Disponible en: https://www.draw.io/ [Consulta: Octubre 2017]

[29] Enmanuel, Norman (2013): "AÑADIR LIBRERIAS .JAR EN ANDROIDSTUDIO". SINTAXIS PRAGMATICA, 18 de Julio. Disponible en: https://sintaxispragmatica.wordpress.com/2013/07/18/anadir-librerias-jar-en-androidstudio/ [Consulta: Marzo 2018]

[30] Express (n.d): *Express Infraestructura web, rápida, minimalista y flexible para Node.js*. Disponible en: http://expressjs.com/es/ [Consulta: Marzo 2018]

[31] Falticon (2018): "Free vector icons", *flaticon.com*. Disponible en: https://www.flaticon.com/ [Consulta: Diciembre 2017]

[32] Fran Garcia (2015): *"49. Activar GPS (Programación Android Studio tutorial español)"*, [Vídeo online]. Disponible en: https://www.youtube.com/watch?v=nenaJb66JL8 [Consulta: Mayo 2018]

[33] Gazoni y Clark. (2018). "A Python library to read/write Excel 2010 xlsx/xlsm files". *OpenPyXL*, 28 de Mayo. Disponible en: https://openpyxl.readthedocs.io/en/stable/ [Consulta: Diciembre 2017]

[34] Git (2018): *git-scm.com*, 2 de Abril. Disponible en: https://git-scm.com/ [Consulta: Octubre 2017]

[35] Github Desktop (2018): *desktop.github.com*. Disponible en: https://desktop.github.com/ [Consulta: Octubre 2017]

[36] gonz (2018). *MonkeyLearn API for Python* [GitHub]. Disponible en: https://github.com/monkeylearn/monkeylearn-python [Consulta: Noviembre 2017]

[37] Google Charts (2017): *Draw Multiple Charts* .Disponible en: https://developers.google.com/chart/interactive/docs/basic_multiple_charts [Consulta: Diciembre 2017]

[38] Google Charts (2017): *Visualization: Column Chart*. Disponible en: https://developers.google.com/chart/interactive/docs/gallery/columnchart [Consulta: Diciembre 2017]

[39] Google Charts (2017): *Visualization: Pie Chart*. Disponible en: https://developers.google.com/chart/interactive/docs/gallery/piechart [Consulta: Diciembre 2017]

[40] Google Maps API (2017): "Agregar un mapa de Google con un marcador a tu sitio web", *developers.google.com*. Disponible en: https://developers.google.com/maps/documentation/javascript/adding-a-google-map [Consulta: Enero 2018]

[41] Google Maps API (2017): *Marcadores*. Disponible en: https://developers.google.com/maps/documentation/android-api/marker?hl=es-419 [Consulta: Mayo 2018]

[42] Google Maps API (2017): *Objetos de mapas*. Disponible en: https://developers.google.com/maps/documentation/android-api/map?hl=es-419 [Consulta: Abril 2018]

[43] GooglePlay (2017): *Appygeek*. Disponible en: https://play.google.com/store/apps/details?id=com.mobilesrepublic.appygeek [Consulta: Noviembre 2017]

[44] GooglePlay (2018): *Citizen. Safety & Awareness*. Disponible en: https://play.google.com/store/apps/details?id=sp0n.citizen&hl=es [Consulta: Noviembre 2017]

[45] GooglePlay (2018): *Feedly*. Disponible en: https://play.google.com/store/apps/details?id=com.devhd.feedly [Consulta: Noviembre 2017]

[46] GooglePlay (2018): *Madrid Metro Bus Cercanías*. Disponible en: https://play.google.com/store/apps/details?id=com.greenlionsoft.free.madrid&hl=es [Consulta: Noviembre 2017]

[47] GooglePlay (2018): *Reddit*. Disponible en: https://play.google.com/store/apps/details?id=com.reddit.frontpage&hl=es [Consulta: Noviembre 2017]

[48] GooglePlay (2018): *SocialDrive*. Disponible en: https://play.google.com/store/apps/details?id=rge.tech.usuarios&hl=es [Consulta: Noviembre 2017]

[49] GooglePlay (2018): *Urban Step – Madrid*. Disponible en: https://play.google.com/store/apps/details?id=com.ascii164.urbanstep.android.mad&hl=es [Consulta: Noviembre 2017]

[50] Guzman, F (2016): "RESTful API: ¿Qué es y para que sirve?". *Weblantropia Magazine*, 24 de Mayo. Disponible en: http://www.weblantropia.com/2016/05/24/restful-api-que-es/ [Consulta: Marzo 2018]

[51] Hernánez, JR (n.d): "Lenguajes CSS", *lenguajecss.com*. Disponible en: https://lenguajecss.com/ [Consulta: Noviembre 2017]

[52] Indian Pythonista (2017): *mLab and Python | Part 1*. [Video Online]. Disponible en: https://www.youtube.com/watch?v=BX9BTQf9-Fc&t=4s [Consulta: Febrero 2018]

[53] Indian Pythonista (2017): *mLab and Python | Part 2*. [Video Online]. Disponible en: https://www.youtube.com/watch?v=8GRUwftKAps [Consulta: Febrero 2018]

[54] Jara, J, Ruben. (2017). Dudas gráficos de tipo ColumnChart de Google Charts. *Stack Overflow* [Foro de Internet]. Disponible en: https://es.stackoverflow.com/questions/81159/dudas-gr%C3%A1ficos-de-tipo-columnchart-de-google-charts [Consulta: Diciembre 2017]

[55] JavaScript (2016): *Tutorial JavaScript*. Disponible en: https://www.javascript.com/try [Consulta: Noviembre 2017]

[56] Laney, Doug(2001): "3D Data Management: Controlling Data Volume, Velocity, and Variety", *blogs.gartner.com*, 6 de Febrero. Disponible en: https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf [Consulta: Abril 2018]

[57] Learnedia (2017): "HOW TO INSTALL MONGODB AND CONFIGURE WITH PHP FOR XAMPP ON WINDOWS". *Learnedia*, 19 de Junio. Disponible en: https://learnedia.com/install-mongodb-configure-php-xampp-windows/ [Consulta: Octubre 2017]

[58] madridiario (2018): *Madridiario, primer periódico digital de la Comunidad de Madrid*. Disponible en: https://www.madridiario.es/ [Consulta: Octubre 2017]

[59] Madridiario (2018): *Noticias de distritos*. Disponible en: https://www.madridiario.es/distritos [Consulta: Octubre 2017]

[60] Martí, M (2016): "¿Qué es el Web scraping? Introducción y herramientas". *Blog de Sitelabs*, 8 de Abril. Disponible en: https://sitelabs.es/web-scraping-introduccion-y-herramientas/ [Consulta: Noviembre 2017]

[61] Material Palette (n.d): *Material Design Color Palette Generator*. Disponible en: https://www.materialpalette.com/ [Consulta: Febrero 2018]

[62] MeaningCloud (2018): *Text Classification Developer Tools*. Disponible en: https://www.meaningcloud.com/developer/text-classification/dev-tools/1.1 [Consulta: Diciembre 2017]

[63] MeaningCloud(2018): *Servicios web de analítica y minería de textos*. Disponible en: https://www.meaningcloud.com/es/ [Consulta: Diciembre 2017]

[64] mLab (2018): *mLab vs Atlas*. Disponible en: https://mlab.com/mlab-vs-atlas/ [Consulta: Febrero 2018]

[65] mLab (2018): *Mongodb Hosting: mLab*. Disponible en: https://mlab.com/ [Consulta: Febrero 2018]

[66] MongoDB (2008): *MongoDB Java Driver*. Disponible en: https://mongodb.github.io/mongo-java-driver/ [Consulta: Noviembre 2017]

[67] MongoDB (2008): *PyMongo 3.6.1 Documentation*. Disponible en: https://api.mongodb.com/python/current/ [Consulta: Noviembre 2017]

[68] MongoDB(2018): *What is MongoDB?*. Disponible en: https://www.mongodb.com/what-is-mongodb [Consulta: Noviembre 2017]

[69] Mongoose (2011): *Mongoose elegant mongodb object modeling for node.js* .Disponible en: http://mongoosejs.com/ [Consulta: Marzo 2018]

[70] MonkeyLearn (2018): *MonkeyLearn API Reference* .Disponible en: https://monkeylearn.com/api/v3/?python#classifier-api [Consulta: Noviembre 2017]

[71] Moya, R (2013): "Arraylist en Java con ejemplos", *Jarroba.com*, 28 de Marzo. Disponible en: https://jarroba.com/arraylist-en-java-ejemplos/ [Consulta: Abril 2018]

[72] Moya, R (2015): "Scraping en Python (BeautifulSoup), con ejemplos". *Jarroba.com*, 27 de Marzo. Disponible en: https://jarroba.com/scraping-python-beautifulsoup-ejemplos/ [Consulta: Octubre 2017]

[73] Moya, R(2013): "Gson (JSON) en Java, con ejemplos". *Jarroba.com*, 2 de Mayo. Disponible en: https://jarroba.com/gson-json-java-ejemplos/ [Consulta: Mayo 2018]

[74] Mughal, J (2015): "Add Hyperlink in android application through textview". *Android-examples.com*, 14 de Diciembre. Disponible en: https://www.android-examples.com/add-hyperlink-in-android-application-through-textview/ [Consulta: Mayo 2018]

[75] Nieto,A (2011): "¿Qué es Android?". *Blog Xatakandroid*, 8 de Febrero. Disponible en: https://www.xatakandroid.com/sistema-operativo/que-es-android [Consulta: Febrero 2018]

[76] NodeJS(n.d): *Node.js*. Disponible en: https://nodejs.org/es/ [Consulta: Marzo 2018]

[77] Nodemon(n.d): *nodemon*. Disponible en: https://nodemon.io/ [Consulta: Marzo 2018]

[78] Nurik Roman (2017): "Android Launch Generator". *romannurik.github.io*, 30 de Julio. Disponible en: https://romannurik.github.io/AndroidAssetStudio/index.html [Consulta: Abril 2018]

[79] Paruchuri, V (2016): "Working with streaming data: Using the Twitter API to capture tweets". *DataQuest*, 8 de Septiembre. Disponible en: https://www.dataquest.io/blog/streaming-data-python/ [Consulta: Octubre 2017]

[80] Pasillas, A (2017): "¿Qué es machine learning?", *blog.adext.com*. Disponible en: https://blog.adext.com/es/machine-learning-guia-completa

[81] Pherkad (2014): "Operaciones con fechas y horas. Calendarios". *Python 3 para impaciente*, 12 de Febrero. Disponible en: https://python-para-impacientes.blogspot.com.es/2014/02/operaciones-con-fechas-y-horas.html [Consulta: Noviembre 2017]

[82] Php (2018): *¿Qué es PHP? -Manual*. Disponible en: http://php.net/manual/es/intro-whatis.php [Consulta: Noviembre 2017]

[83] pineappslab (2014): "Android tutorial: cómo reemplazar un Fragment por otro dentro de un ViewPager". *pinesappslab*, 14 de Febrero. Disponible en: https://pineappslab.wordpress.com/2014/02/14/android-tutorial-como-reemplazar-un-fragment-por-otro-dentro-de-un-viewpager/ [Consulta: Marzo 2018]

[84] Pixlr (n.d): *Pixlr Editor*. Disponible en: https://pixlr.com/editor/ [Consulta: Marzo 2018]

[85] Postman(2018): *Postman Apps*. Disponible en: https://www.getpostman.com/apps [Consulta: Marzo 2018]

[86] Prudhvi Raj Kumar (2016): *Android - Simple Registration Form along with Validation*, [Vídeo online]. Disponible en: https://www.youtube.com/watch?v=bQGCgBnJixM [Consulta: Abril 2018]

[87] Pyinstaller (2018): *How to install Pyinstaller*. Disponible en: http://pyinstaller.readthedocs.io/en/stable/installation.html# [Consulta: Noviembre 2017]

[88] Python (2018): *25.5. IDLE - Python*. Disponible en: https://docs.python.org/3/library/idle.html [Consulta: Noviembre 2017]

[89] Python Software Fundation(2015): *wget 3.2*. Disponible en: https://pypi.org/project/wget/ [Consulta: Noviembre 2017]

[90] Ramirez, L (2017): "Tutorial de Web Scraping Python en Español; Scrapy". *LuisRamirez*, 11 de Marzo. Disponible en: https://luisramirez.me/tutorial-web-scraping-python/ [Consulta: Octubre 2017]

[91] Recio, J - Interfaces de Usuario(UCM), Juan Antonio Recio García, Curso 2017-2018. Disponible en: Campus virtual de la asignatura (UCM) [Consulta: Octubre 2017]

[92] Reitz, K (n.d): "Requests: HTTP for Humans", *Requests*. Disponible en: http://docs.python-requests.org/en/master/ [Consulta: Noviembre 2017]

[93] Retrofit(2013): *Retrofit A type-safe HTTP client for Android and Java*. Disponible en: http://square.github.io/retrofit/ [Consulta: Marzo 2018]

[94] Revelo James (2015): "TabLayout: ¿Cómo Añadir Pestañas En Android?", *hermosaprogramacion.com*, 27 de Junio. Disponible en: http://www.hermosaprogramacion.com/2015/06/tablayout-como-anadir-pestanas-en-android/ [Consulta: Marzo 2018]

[95] Richardson, L (2015): *Beautiful Soup Documentation*. Disponible en: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ [Consulta: Noviembre 2017]

[96] Robomongo (2017): *Robo 3T*. Disponible en: https://robomongo.org/ [Consulta: Noviembre 2017]

[97] Rock, D (2015): "Calculando la distancia entre dos coordenadas en Java", *DonnierocK*, 16 de Marzo. Disponible en: https://donnierock.com/2015/03/16/calculando-la-distancia-entre-doos-coordenadas-en-java/#comment-5156 [Consulta: Mayo 2018]

[98] Roesslein, J (2009): *Tweepy Documentation*. Disponible en: http://tweepy.readthedocs.io/en/v3.5.0 [Consulta: Noviembre 2017]

[99] sgoliver (2011): "Preferencias en Android I: Shared Preferences". *Sgoliver.net*, 14 de Marzo. Disponible en: http://www.sgoliver.net/blog/preferencias-en-android-i-shared-preferences [Consulta: Marzo 2018]

[100] Sublime Text (2018): *Download - Sublime Text*. Disponible en: https://www.sublimetext.com/3 [Consulta: Octubre 2017]

[101] Ticbeat (2014): *¿Qué es una API y para qué sirve?* Disponible en: http://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/ [Consulta: Abril 2018]

[102] Tweepy (2009): *Getting started tweepy 3.5.0*. Disponible en: http://tweepy.readthedocs.io/en/v3.5.0/getting_started.html [Consulta: Noviembre 2017]

[103] W3schools (2018): *HTML Introduction*. Disponible en: https://www.w3schools.com/html/html5_intro.asp [Consulta: Noviembre 2017]

[104] W3schools.com (2018): *Introduction to XML* .Disponible en: https://www.w3schools.com/xml/xml_whatis.asp [Consulta: Marzo 2018]

[105] W3schools.com (2018): *JSON - Introduction*. Disponible en: https://www.w3schools.com/js/js_json_intro.asp [Consulta: Noviembre 2017]

[106] WikiPedia (2018): *Type-safety*. Disponible en: https://en.wikipedia.org/wiki/Type_safety [Consulta: Noviembre 2017]

[107] Wrappixel (2018): *Material Pro Admin Template*. Disponible en: https://wrappixel.com/demos/admin-templates/material-pro/material/index.html [Consulta: Marzo 2018]

[108] ZEOKAT (2016): "Como instalar Node.js en Windows". *Vozidea*, 29 de Mayo. Disponible en: http://www.vozidea.com/instalar-node-js-windows [Consulta: Marzo 2018]

APÉNDICES

La memoria de este proyecto finaliza con el manual de instalación y el manual de usuario, respectivamente. En primer lugar, se exponen los manuales de instalación, guiados paso por paso. El manual de usuario tiene como objetivo guiar al usuario en las distintas funcionalidades que envuelve cada aplicación.

A- MANUAL DE INSTALACIÓN

APLICACIÓN ANDROID

1. Preparación del entorno

- **Android Studio**

Es necesario instalar la versión más reciente de Android Studio. Esta puede descargarse de manera gratuita desde la página oficial de Android:

https://developer.android.com/studio/?hl=es

- **Node.js:**

Se requiere instalar la versión correspondiente al ordenador que se vaya a usar desde la página oficial: https://nodejs.org/en/download/. Si se pretende automatizar el proceso de instalación en Windows es recomendable bajarse el window installer .msi.

Una vez instalado node.js, se necesitan instalar una serie de paquetes.

Este es el comando que se usa para la instalación de cada uno de ellos. Se ha de ejecutar desde la consola de comandos de Windows:

```
npm install -g <nombrePaquete>
```

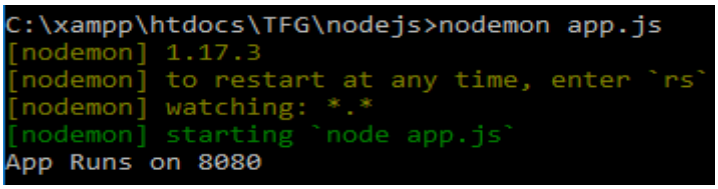
Estos son los paquetes que requiere la aplicación Android:

- body-parser
- express
- mongoose
- node-restful
- nodemon

2. Iniciando el sistema

En primer lugar, es necesario abrir la consola de comandos e ir hasta la ruta donde se encuentra el directorio 'nodejs' del proyecto. (C:\..\TFG\nodejs). Una vez ahí, ejecute el comando: **`nodemon app.js`**

Si todo ha ido correctamente, en su consola de comandos debe aparecer lo siguiente:



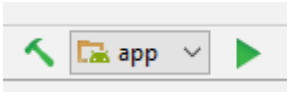
204- Figura A.A-2.1 - Iniciar sistema nodemon

En segundo lugar, es necesario arrancar Android Studio. Si es la primera vez que lo haces es posible que tarde unos minutos. Esto es algo normal, se debe a que Android necesita cargar todos los plugins y demás elementos requeridos.

Tras esto, se tiene que abrir el proyecto 'MadAlert'. 'File' > 'Open' y se busca la ruta donde está ubicado. (C:\..\TFG\MadAlert). Cuando se encuentre, verá el icono de Android Studio sobre el clicable. Haga click y espere a que se cargue completamente.

205- Figura A.A-2.2 - Icono Android MadAlert

Una vez cargado, el icono del martillo situado arriba a la derecha construirá el proyecto. Cuando aparezca el mensaje 'Gradle build finished' y si todo ha ido bien, basta con pulsar el play y elegir el emulador* donde queremos que nuestra app se muestre. Se recomienda *Nexus 5X API 27.* Si es la primera vez que inicias el emulador de Android es posible que tengas que instalarlo. Sigue los sencillos pasos que Android Studio te guía para ello. También puedes utilizar tu móvil como emulador.



206- Figura A.A-2.3 - Emulador

*Cabe remarcar que si se elige como emulador un Smartphone conectado vía USB o en el que previamente se ha instalado una APK[8] en él, es necesario para su funcionamiento que el ordenador y el Smartphone estén conectados a la misma red.

IMPORTANTE: El archivo **NetworkUtil.java** del proyecto Android debe tener la misma IP que el ordenador. Cambiar la siguiente línea de código:

return new Retrofit.builder().baseUrl("http://**La*IPdelOrdenador***:8080/api/v1/")

APLICACIÓN WEB

1. Preparación del entorno

- **Sistema operativo**

Se recomienda instalar esta herramienta sobre un sistema operativo Windows. Esta guía de instalación se ha realizado sobre la distribución de Windows 10 de 64 bits. En el siguiente enlace puede adquirirse dicha distribución:

https://www.microsoft.com/es-es/software-download/windows10#34de222e-6944-44aa-8f45-8e96988880a4

Una vez instalado el sistema operativo correspondiente, continuamos la preparación del entorno con la instalación de los componentes relacionados con las bases de datos, servidores web y herramientas para la recolección de datos procedentes de la web.

- **Python**

Una vez que tenemos nuestro sistema operativo adecuado, se procede a instalar Python. En esta ocasión, nos hemos servido de la versión de Python 3.6, la cual podemos descargar fácilmente en el siguiente enlace: https://www.python.org/downloads/

Tras su descarga, ejecutamos el archivo .exe obtenido y seguimos los pasos de instalación.

Para el correcto funcionamiento de todos scripts de Python, es necesario instalar los paquetes que se muestran a continuación:

Paquete	Nombre en la instalación
Beautiful Soup[95]	bs4
Json	json
request[92]	request
pymongo[67]	pymongo
tweepy[98]	tweepy
MonkeyLearn[36]	monkeylearn
wget 3.2[89]	wget
openpyxl[33]	openpyxl

207- Figura A.W-1.1 - Tabla paquetes Python

La instalación de estos paquetes se realiza de la manera aquí mostrada:

1. En primer lugar, abrimos la consola de comandos de Windows como administradores.

2. Introducimos el comando `pip install <Nombre en la instalación>` (ver tabla superior). Esperar al mensaje '*Successfully installed*'.

   NOTA: en caso de que el comando no se realice correctamente en la ruta que por defecto aparece al abrir la consola, diríjase a `C:\..\Python\Python36-32\Scripts` y ejecute ahí el comando señalado.
   NOTA 2: es posible que pip no se haya instalado por defecto. En caso de que esto ocurra, ejecute el siguiente comando para solucionar el problema:
   `python -m ensure pip --default-pip`

En este punto, todo lo relacionado con Python ya ha sido instalado y podemos continuar con la guía de instalación.

- **XAMPP y PHP[57]**

Como servidor web, usamos XAMPP, que por defecto en su última versión hasta la fecha viene con la instalación de php7 incluida. Puede descargarse aquí (se recomienda instalar la última versión 7 estable): https://www.apachefriends.org/es/download.html

Tras descargar la versión adecuada, ejecutamos el archivo .exe y seguimos los pasos que aparecen en el manual de instalación.

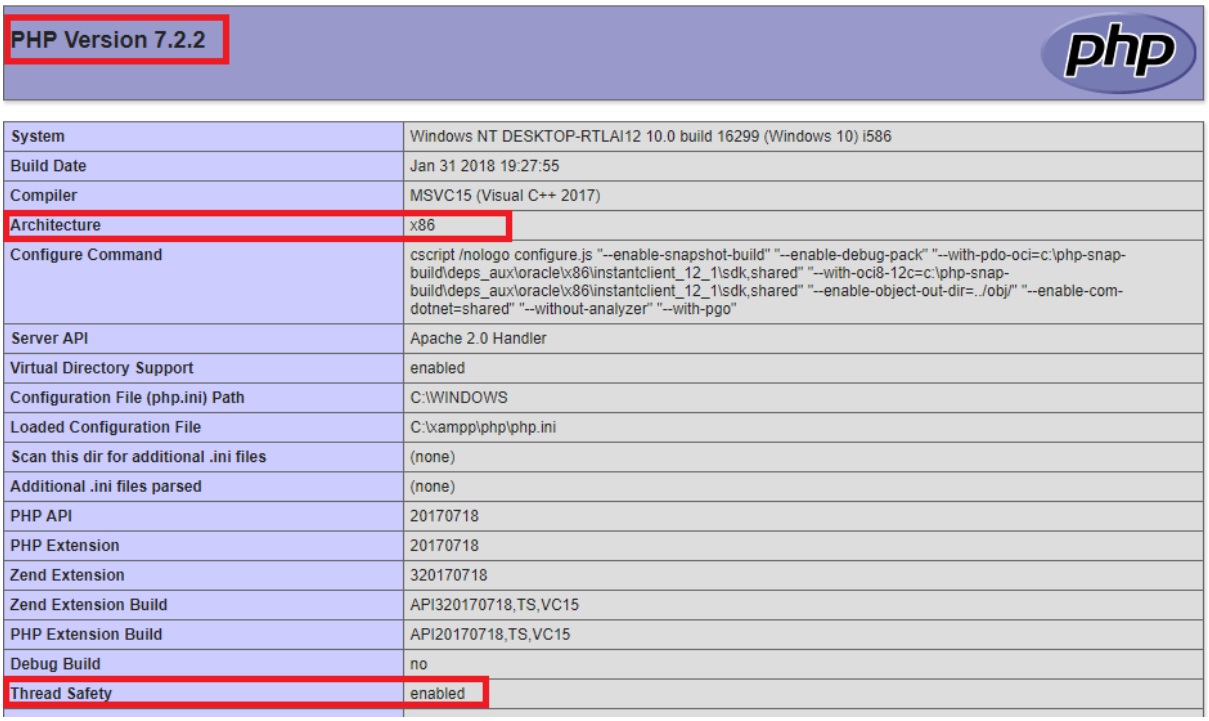
Adicionalmente a esto, es necesario incluir y configurar MongoDB [7] en nuestro servidor XAMPP. Para ello, seguimos las siguientes instrucciones:

1. Descarga el driver de MongoDB.
   https://pecl.php.net/package/mongodb
   Tenga en cuenta tres requisitos indispensables para saber cuál es el que necesita:
   a. La versión PHP que hay en su XAMPP.
   b. Su arquitectura (x86/x64).
   c. Thread Safety está activado o desactivado.
   Para conocer todos estos requisitos:
   1. Inicie XAMPP.
   2. Vaya a su navegador preferido para ir al dashboard de XAMPP (típicamente `http://localhost`).
   3. Haga click en PHPInfo que aparece en el menú superior. Vea la Figura A.W-1.2

2. Tras haber descargado el .zip adecuado, extraiga el archivo .dll y cópielo en el directorio `C:\..\xampp\php\ext`

3. Registrar el archivo copiado en el paso anterior en php.ini. Añada la siguiente línea a php.ini
   `extension=php_mongodb.dll`
   Asegúrese que el nombre del archivo copiado y el añadido en la línea anterior coinciden.

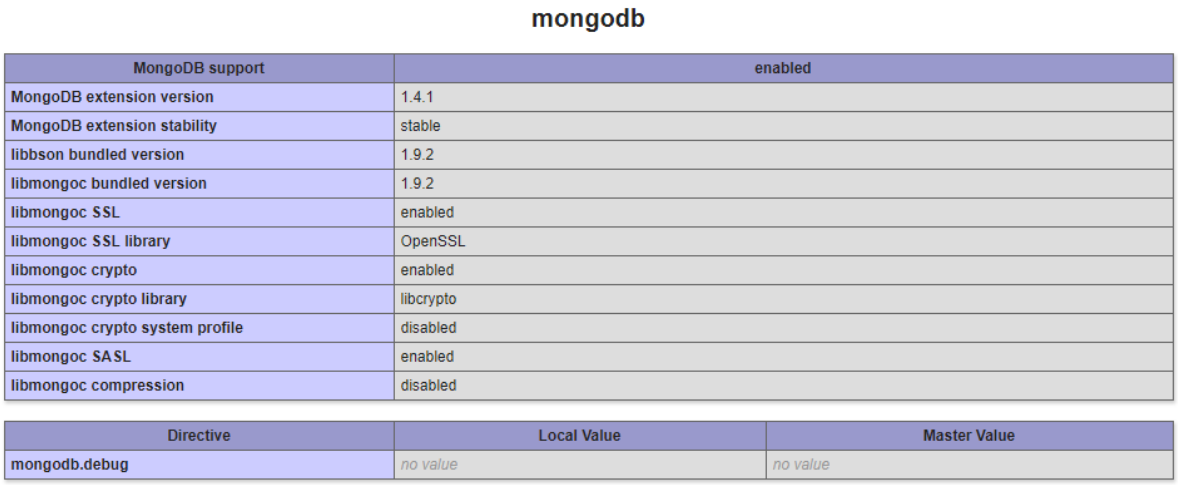
4. Reinicie XAMPP, vaya PHPInfo de nuevo y compruebe que mongo aparece (se aconseja buscar la palabra ‘mongo’ con Ctrl+F para mayor rapidez). Vea la Figura A.W-1.3.



PHP Version 7.2.2

System	Windows NT DESKTOP-RTLAI12 10.0 build 16299 (Windows 10) i586
Build Date	Jan 31 2018 19:27:55
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,TS,VC15
PHP Extension Build	API20170718,TS,VC15
Debug Build	no
Thread Safety	enabled

208- Figura A.W-1.2 - PHPInfo en XAMPP.



mongodb

MongoDB support	enabled
MongoDB extension version	1.4.1
MongoDB extension stability	stable
libbson bundled version	1.9.2
libmongoc bundled version	1.9.2
libmongoc SSL	enabled
libmongoc SSL library	OpenSSL
libmongoc crypto	enabled
libmongoc crypto library	libcrypto
libmongoc crypto system profile	disabled
libmongoc SASL	enabled
libmongoc compression	disabled

Directive	Local Value	Master Value
mongodb.debug	*no value*	*no value*

209- Figura A.W-1.3 - MongoDB en PHPInfo de XAMPP

De este modo, si todo ha ido bien, la instalación de XAMPP y sus componentes ha finalizado.

ATENCIÓN: este último punto dejó de ser necesario cuando la base de datos se subió a la nube ‘mLab’. Se mantiene por si se diese el caso de requerirse en un futuro.

- **MongoDB**

Es hora de instalar nuestro gestor de base de datos. Se ha utilizado la versión 3.4 de MongoDB, que puede descargarse en el apartado '*Community Server*' aquí: https://www.mongodb.com/download-center#community

Tras su descarga, ejecute el archivo .msi y siga los pasos hasta completar su instalación.

Es necesario crear una carpeta por defecto para las colecciones de datos que habrá: cree un directorio en el dispositivo C de tal modo **`C:\data\db.`**

2. Iniciando el sistema

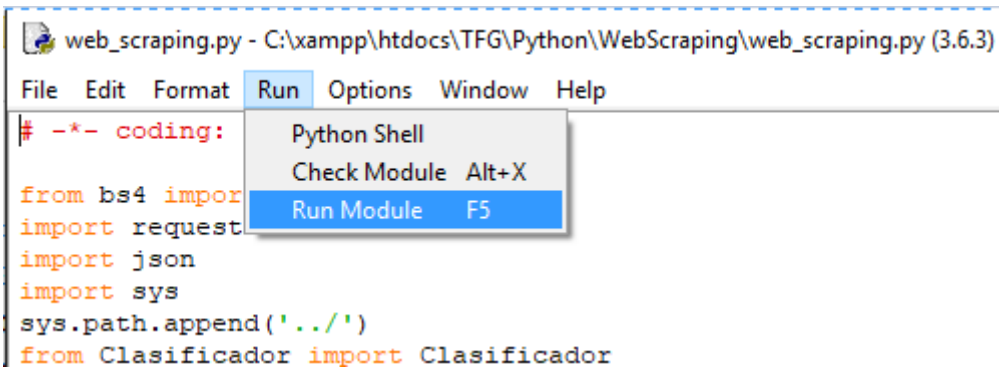
- **Aplicación web**

A continuación, se siguen los pasos para ejecutar el sistema en local (versión para prueba de Antonio):

1. Clone el proyecto en su servidor local (suele ser `C:\..\xampp\htdocs`)
2. Inicie XAMPP (`C:\..\xampp\xampp_start.exe`)
3. Lance MongoDB (`C:\..\MongoDB\Server\3.6\bin\mongod.exe`)
4. Para llenar las colecciones de datos, ejecute los siguientes archivos:

    a. `Python/WebScraping/web_scraping.py`

    `Para ejecutarlos es necesario abrir el IDLE y pulsar en` *`Run Module`*`:`



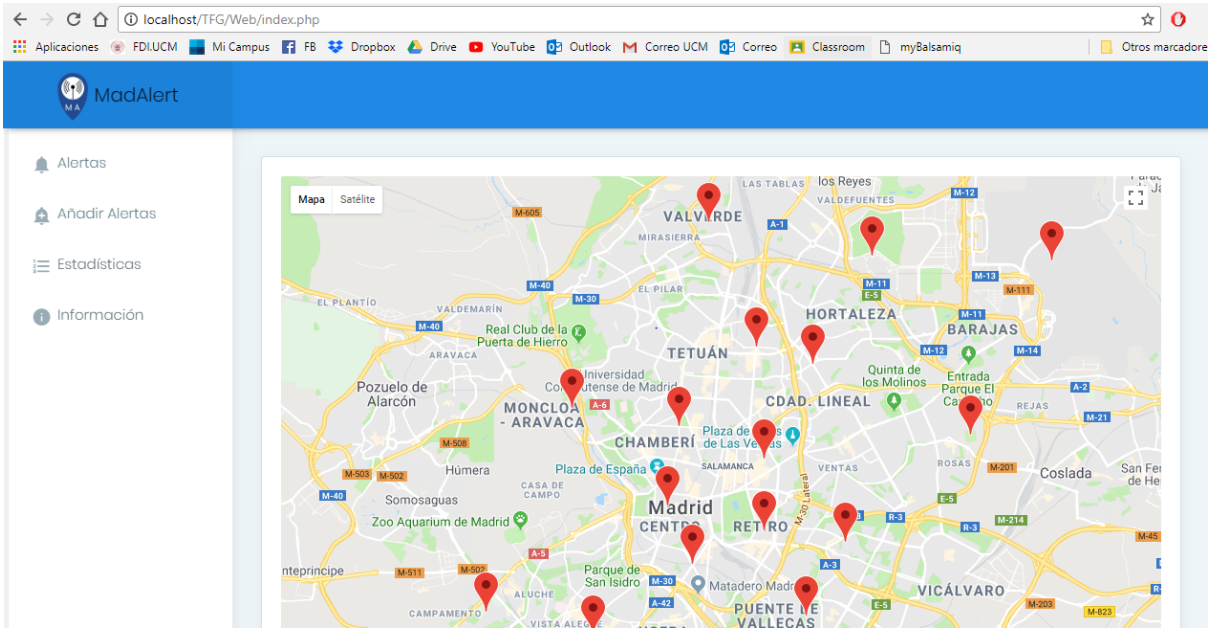
210- Figura A.W-2.1 - Run python

Inmediatamente se abrirá en una nueva ventana la shell de Python donde aparecerán los tweets. Es decir, se podrá algo como lo siguiente:

211- Figura A.W-2.2 - Corriendo web_scraping.py

b. Python/Twitter/ObtenerStream.py
c. Python/WebScraping/insertarDatosPolicia.py

5. Abra en su navegador preferido: `http://localhost/TFG/web/index.php`



212- Figura A.W-2.3 - Iniciando aplicación web

B- MANUAL DE USUARIO

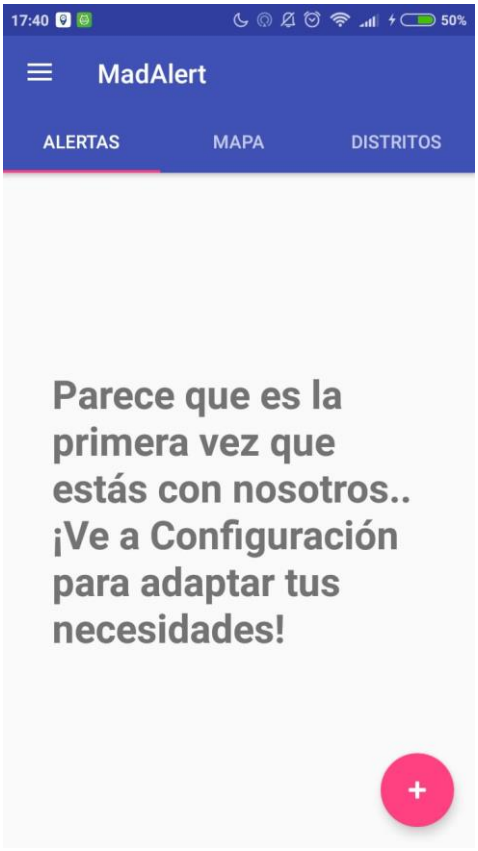
APLICACIÓN ANDROID

El objetivo de este apartado es mostrar detenidamente al usuario todas las vistas de la aplicación Android y explicarle lo que puede realizar en cada una de ellas.

Al iniciar la aplicación, aparece una ventana de carga que en Android se denomina *Splash* Consiste en un fondo totalmente en azul con el icono de MadAlert. Esta vista desaparece sola tras unos segundos (Figura B.A.1).

213- Figura B.A.1 - Iniciando la app

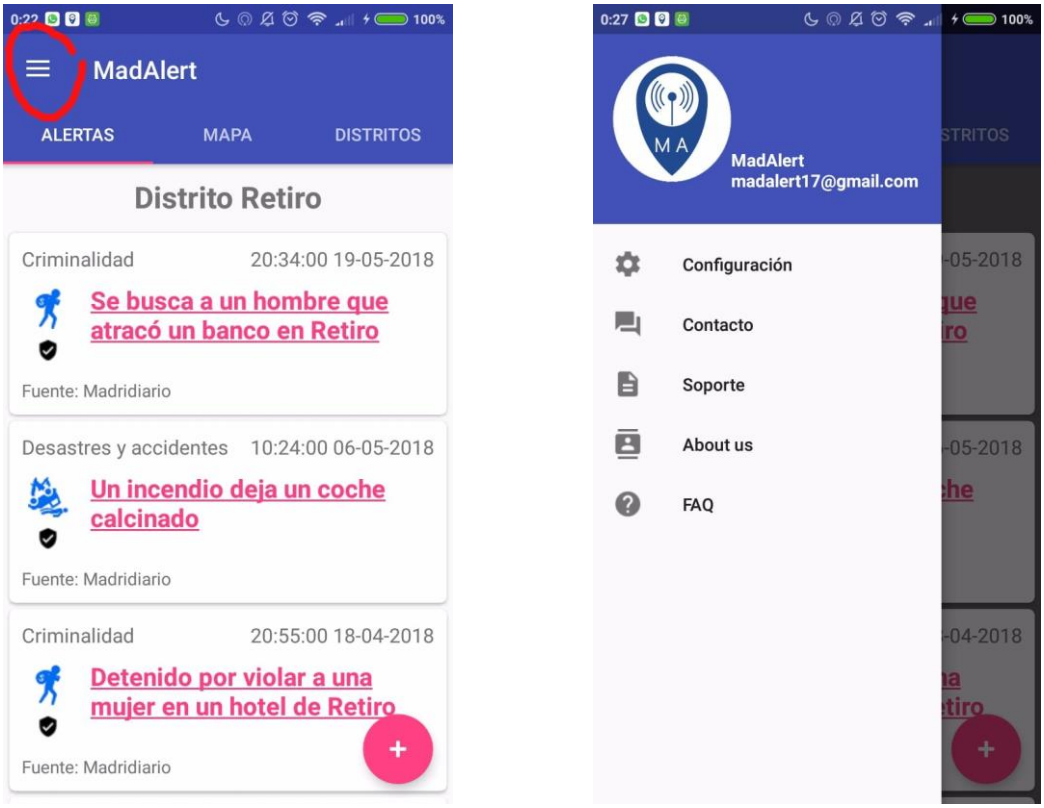
Una vez que se ha cargado, en el primer vistazo de la aplicación aparecen tres pestañas: Alertas, Mapa y Distritos. Por defecto la aplicación sitúa al usuario en la pestaña de alertas, donde se muestran las alertas acorde a la configuración que tiene el usuario. Si el usuario no dispone de configuración o es la primera vez que entra en la aplicación se muestra un mensaje para que configure la aplicación.



214- Figura B.A.2 - Mensaje para que configure la aplicación

Para configurar la aplicación hay que ir al menú, tan sólo es necesario hacer click en las tres rayas que se encuentran en la esquina izquierda superior o bien arrastrando el dedo desde la izquierda hacia el centro de la pantalla de nuestro dispositivo. Para que desaparezca, se ha de hacer el proceso inverso: empujar el menú hacia la izquierda de la pantalla.

En este desplegable podemos encontrar las distintas opciones que se exponen detalladamente más adelante: configuración del usuario, contacto, soporte, about us y faq (Figura B.A.3).



215- Figura B.A.3 - Menú

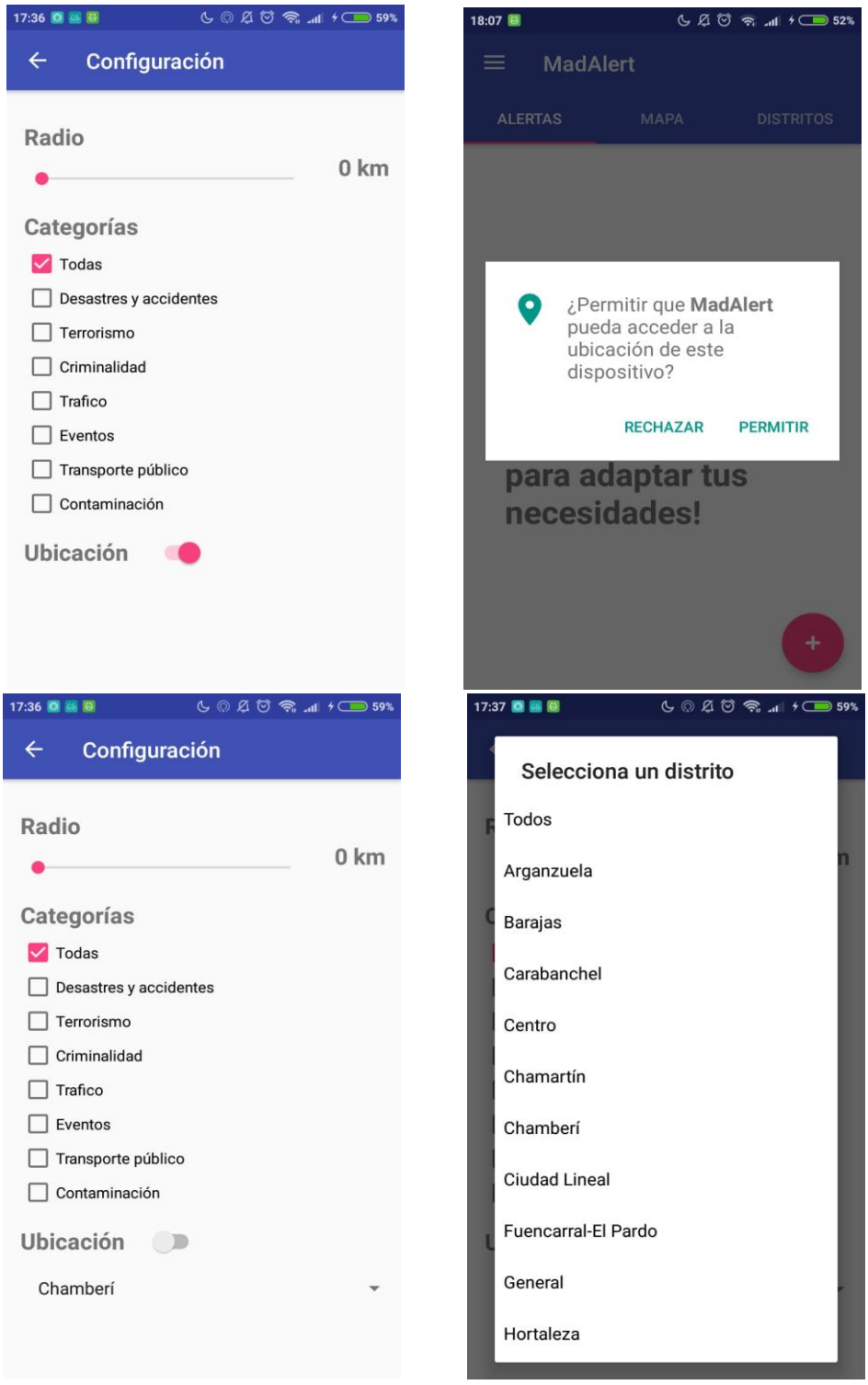
Para configurar nuestra aplicación elegimos la primera opción que aparece en el menú: configuración. Los campos que aparecen en esta ventana son los que determinan qué alertas se muestran en la pestaña ‘ALERTAS’ y en la de ‘MAPA’.

Desde esta pantalla, el usuario puede elegir los kilómetros que desea de radio. Ha de tener en cuenta que esta distancia se contará desde el punto de origen hasta el número que haya establecido **a la redonda.** La distancia mínima son 0 km y la máxima 25.

Después aparece una lista con las categorías que el usuario desea solicitar. Existe la opción de ‘Todas’, lo que significa que se requieren todas las categorías que hay. Pueden seleccionarse tantas categorías como el usuario estime.

En último lugar aparece un *switch* que establece si la ubicación está activa o no. Si la ubicación se activa y no se le ha dado los permisos de ubicación a esta aplicación los pedirá. Al tener la ubicación activada el punto de origen será la ubicación exacta del usuario. Si por el contrario esta se encontrará desactivada, el punto de origen será el marcador central del distrito que el usuario puede elegir en la lista desplegable que aparece. En este desplegable también aparece la opción ‘Todos’ por si el usuario desea información de los 21 distritos existentes. Además, la opción ‘General’ abarca todas aquellas alertas que no tienen un distrito definido.

Cuando estén todos los campos a gusto del usuario, con volver atrás estas configuraciones se guardan, no es necesario hacer click en ningún botón de guardar. (Figura B.A.4).



216- Figura B.A.4 - Configuración

Al tener ya la configuración establecida, si el usuario vuelve a la pestaña alertas aparecen las alertas correspondientes a la configuración. Como hay distintas variantes de configuración, se explican los diferentes casos a continuación:

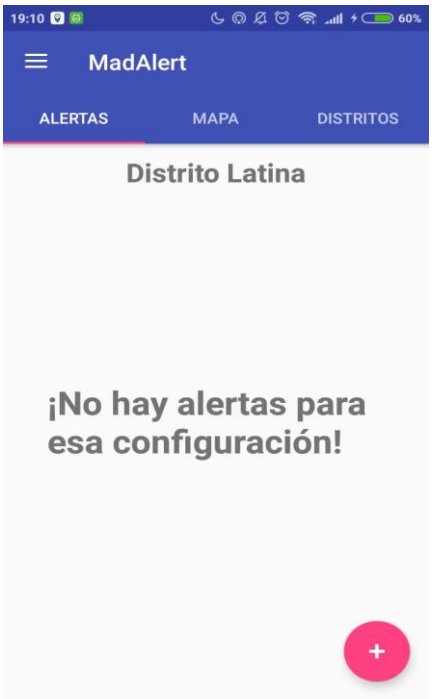
- **Radio 0 kms:** cuando el radio elegido por el usuario es de 0 kms solo se muestran las alertas del distrito correspondiente a la localización del usuario o al elegido por el usuario en la configuración. Aparece la categoría a la que pertenece la alerta, la fecha, la fuente y si está verificada o no.

217- Figura B.A.5 - Pestaña alertas con radio 0 kms

- **Radio mayor de 0 kms:** cuando el radio es mayor de 0, se buscan las alertas de los distritos que forman parte de la circunferencia del radio elegido por el usuario. Teniendo en cuenta que el centro es la ubicación actual o el distrito elegido. La visualización es igual que cuando el radio es mayor de 0 kms, solo que salen alertas de más distritos.

218- Figura B.A.6 - Muestra de alertas con radio mayor que 0 kms

En el caso de que en la pestaña alertas la configuración correspondiente no encuentre ninguna alerta aparece un mensaje indicándolo.

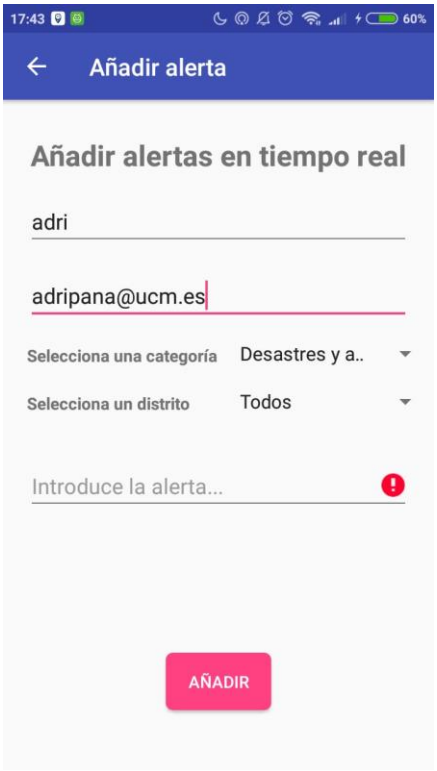


219- Figura B.A.7 - Mensaje no hay alertas

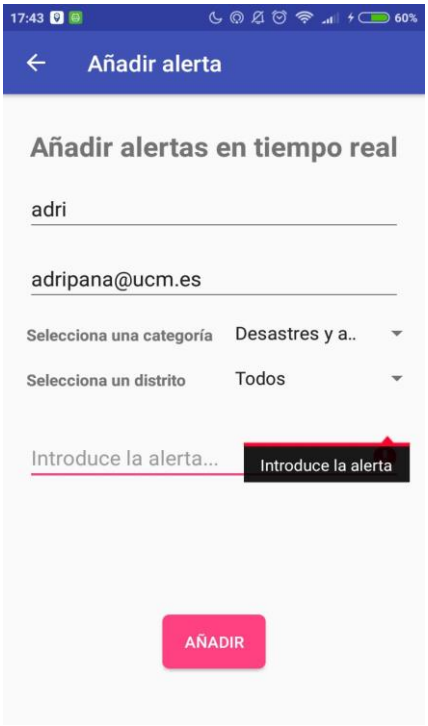
Para actualizar la pestaña de alertas se puede realizar el gesto de desplazar la ventana hacia abajo.

En esta pestaña también se puede añadir una alerta en tiempo real dándole al botón de '+'. Aparece un formulario en el que hay que completar los datos de la alerta y cuando estén rellenos darle a añadir, si algún dato no está completo se indica mediante un circulito y si se pulsa sobre el informa de lo que falta por añadir.

220- Figura B.A.8 - Formulario para añadir una alerta en tiempo real

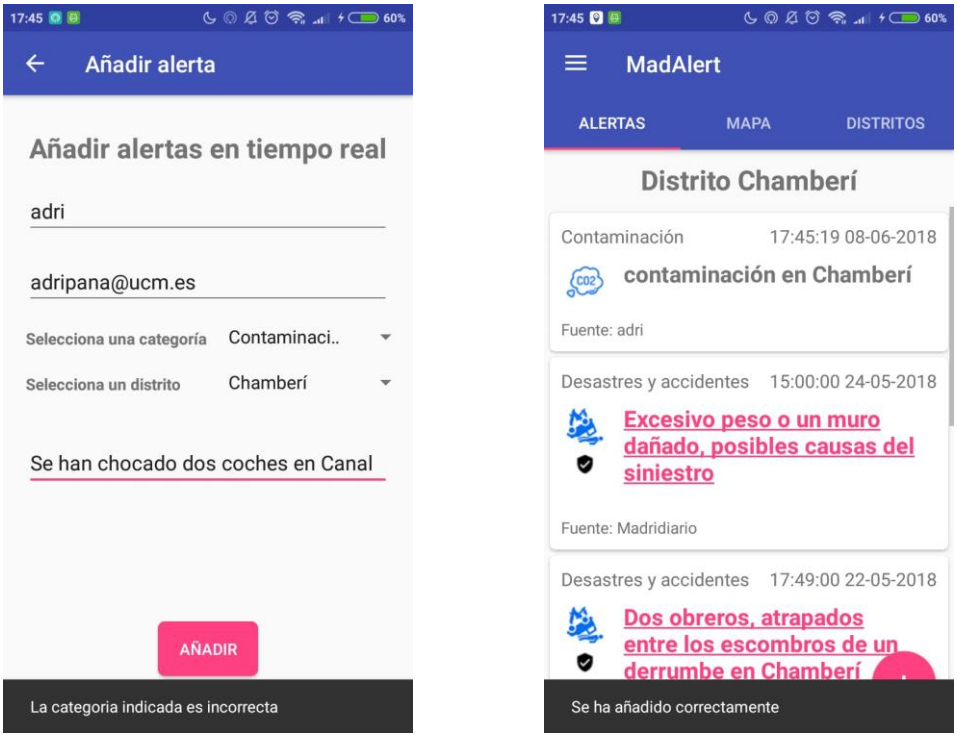


221- Figura B.A.9 - Añadir incompleto



222- Figura B.A.10 - Mensaje en el campo que falta

Para que se añada la alerta la categoría elegida tiene que ser la correspondiente a la alerta, sino aparece un mensaje de error y la alerta no se añade.

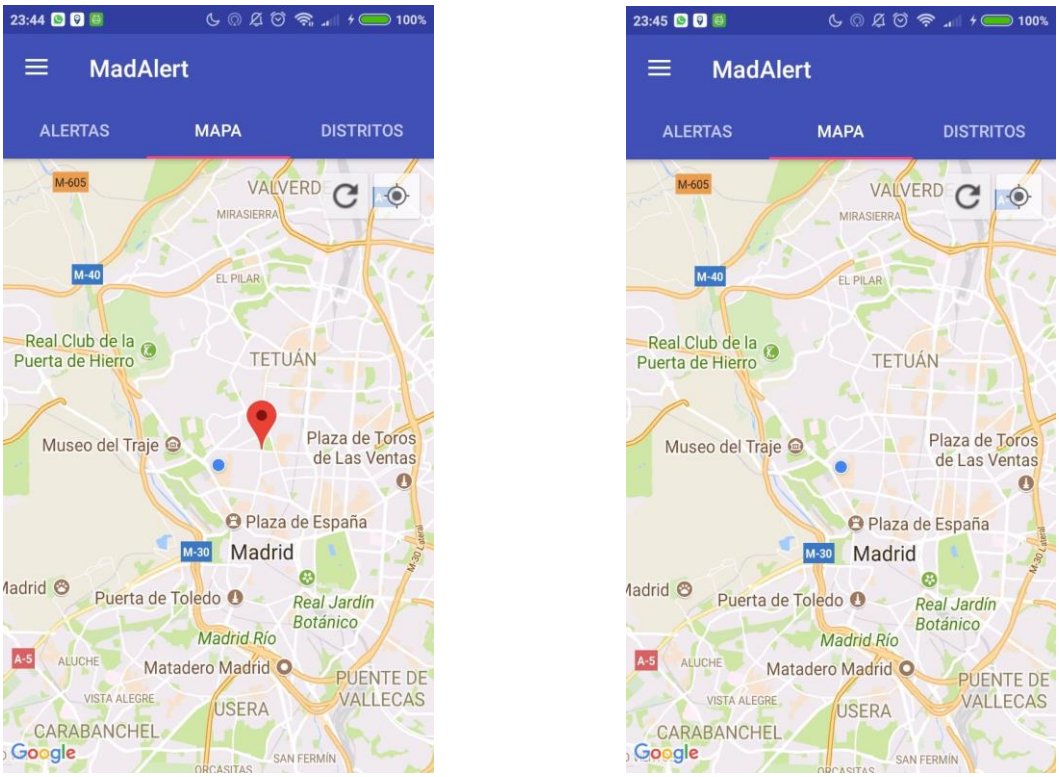


223- Figura B.A.11 - Añadir alerta con categoría incorrecta y añadida correctamente

En la pestaña "MAPA", aparece un mapa de la comunidad de Madrid y según la configuración del usuario se muestran los distintos marcadores correspondientes a los distritos de Madrid. El marcador aparece siempre y cuando haya alertas de ese distrito con las especificaciones indicadas en la configuración. Cuando se pulsa sobre un marcador aparece que distrito es y el número de alertas que hay con la configuración especificada.

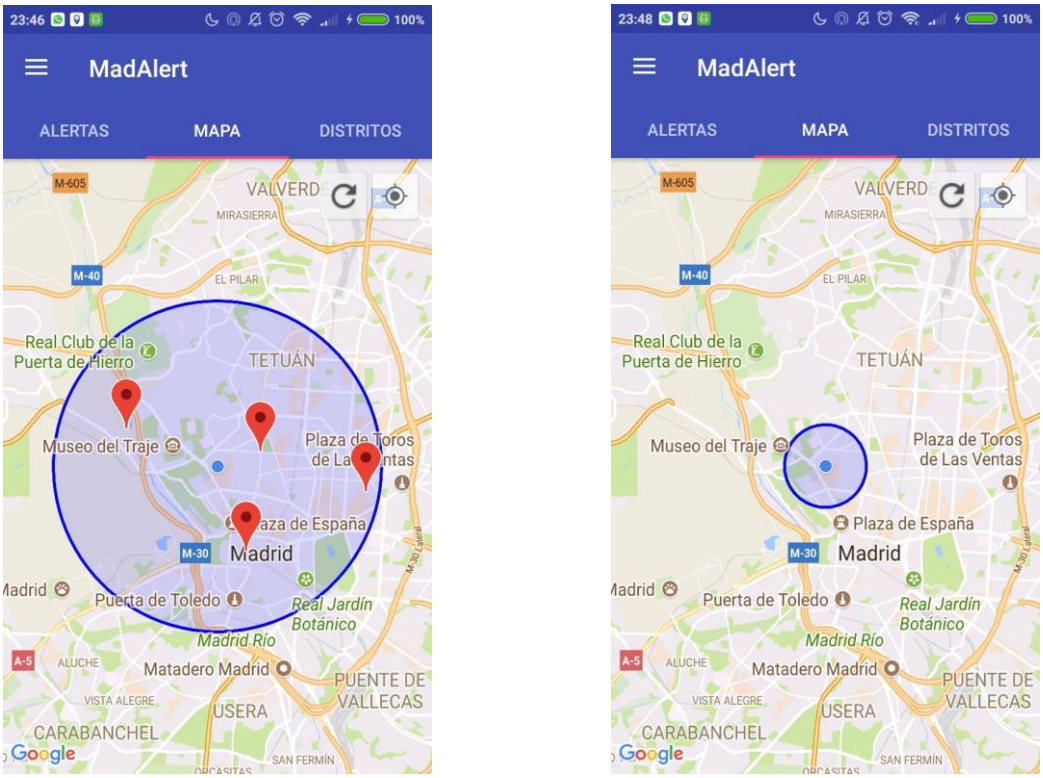
La vista del mapa será diferente dependiendo de lo que haya escogido el usuario en la configuración:

- **Radio 0 y ubicación activada:** aparece el punto azul que corresponde con la ubicación actual del usuario y el marcador que corresponde al distrito del usuario siempre que haya alertas en ese distrito con la configuración establecida.



224- Figura B.A.12 - Mapa con radio 0 y ubicación activada

- **Radio mayor que 0 kms y ubicación activada:** al tener un radio mayor de 0 kms en el mapa aparece una circunferencia con el radio elegido por el usuario y dentro de ella los marcadores de los distritos que tengan alertas y estén dentro de ella. El centro de esta circunferencia corresponde con el punto azul que es la ubicación del usuario.

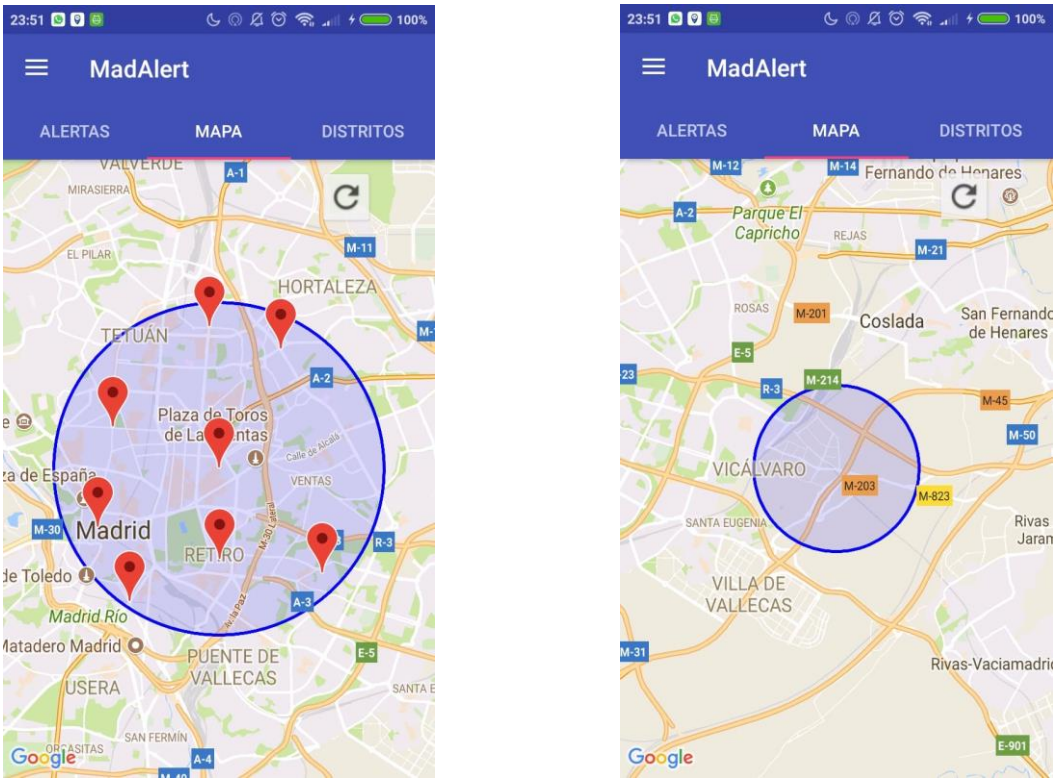


225- Figura B.A-13 - Mapa con radio mayor de 0 kms y ubicación activada

- **Radio de 0 kms y selección del distrito:** cuando el usuario tiene la ubicación desactivada selecciona el distrito del que quiere obtener las alertas. En este caso se buscan las alertas de ese distrito y aparece en el mapa solo el marcador correspondiente al distrito si tiene alertas. Si no hubiera alertas no aparece ningún marcador en el mapa.

226- Figura B.A.14 - Mapa cuando se elige distrito y radio es 0

- **Radio mayor de 0 kms y selección de distrito:** al tener un radio mayor de 0 kms en el mapa aparece una circunferencia con el radio elegido por el usuario y dentro de ella los marcadores de los distritos que tengan alertas y estén dentro de ella. El centro de esta circunferencia corresponde con el punto central del distrito elegido por el usuario.



227- Figura B.A.15 - Mapa cuando se elige distrito y el radio es mayor que 0

Si se quiere visualizar las alertas de un distrito a partir del mapa, basta con pulsar sobre el marcador correspondiente (el icono rojo) se abre la *infowindow* donde sale el nombre del distrito y el número de alertas que hay, si estas se quieren visualizar bastaría con pulsar sobre la *infowindow*.

228- Figura B.A.16 - Info marcador

229- Figura B.A.17 - Visualizar alertas

Para actualizar el mapa se puede pulsar en el botón con el icono de refrescar que se encuentra en la esquina superior derecha.

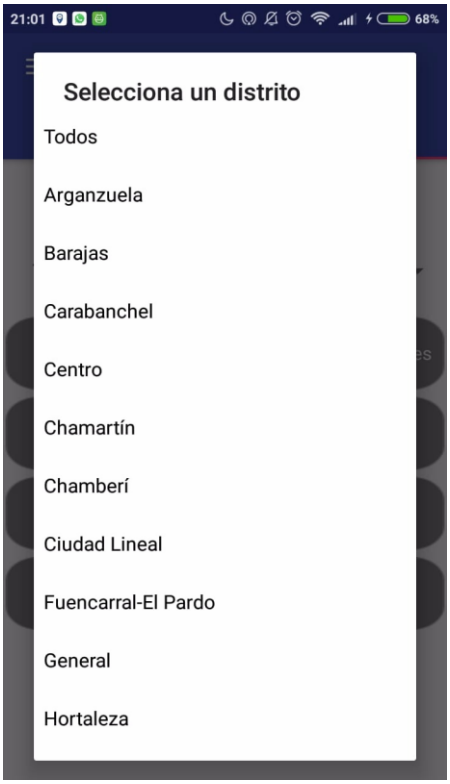
En la pestaña "DISTRITOS", aparece un texto con Selecciona un distrito, un desplegable para seleccionar el distrito y unos botones para marcar una o varias categorías, para realizar la búsqueda está el botón Mostrar.

La vista del distrito será diferente dependiendo de si existen alertas con esas especificaciones o no:

- **Si no seleccionas ninguna categoría:** se muestra abajo el mensaje de que debes seleccionar al menos una categoría.

230- Figura B.A.18 - Mensaje seleccionar al menos una categoría

- **Si seleccionas un distrito que no sea Todos y la categoría si es Todas:** se muestran todas alertas del distrito sin importar la categoría.



231- Figura B.A.19 - Desplegable distritos

232- Figura B.A.20 - Categoría “todas” seleccionada

233- Figura B.A.21 - Visualización alertas de un distrito

- **Si seleccionas un distrito que no sea Todos y varias categorías:** se muestran las alertas del distrito con las categorías seleccionadas. Si no tiene alertas de una de las categorías y si tiene de las otras, muestra las que tiene.

234- Figura B.A.22 - Selección varias categorías

235- Figura B.A.23 - Alertas con varias categorías

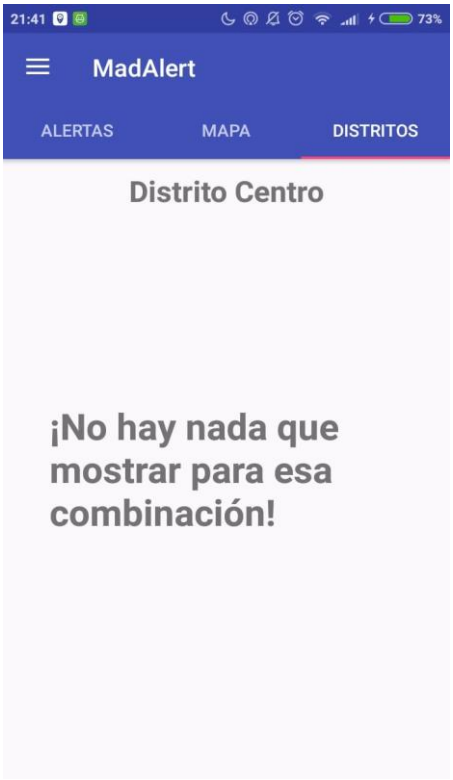
- **Si seleccionas el distrito Todos y cualquier categoría:** se muestran las alertas de todos los distritos que tengan alertas con las categorías seleccionadas. En cada alerta se muestra el distrito al que pertenecen.

236- Figura B.A.24 – Distrito ”Todos”

237- Figura B.A.25 - Alertas de todos los distritos

- **Si el distrito no tiene alertas con esa categoría:** se muestra un mensaje diciendo que no hay nada para mostrar con esa combinación.

238- Figura B.A.26 - Selección categoría y distrito

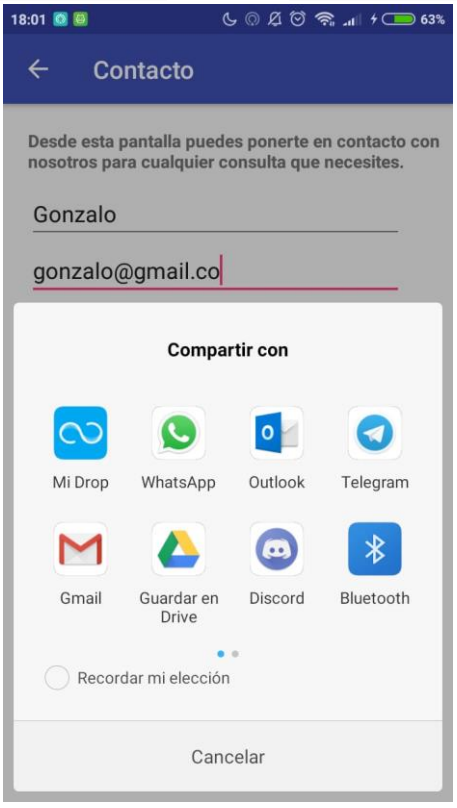


239- Figura B.A.27 - Mensaje no hay alertas esa combinación

Para actualizar la pestaña de distritos se puede realizar el gesto de desplazar la ventana hacia abajo cuando se muestran las alertas.

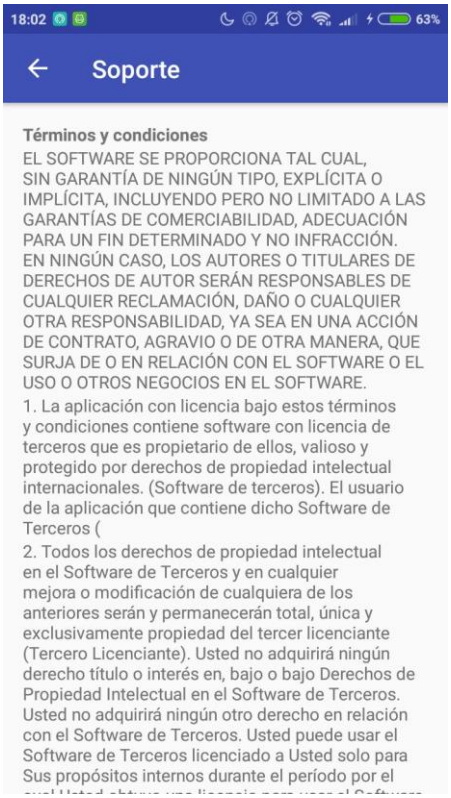
El menú lateral cuenta con más opciones a parte de configuración como se puede ver en la (Figura B.A.3). Estas opciones son las siguientes:

- **'Contacto'** ofrece al usuario la posibilidad de enviar un correo a los desarrolladores de la aplicación. Para ello, basta con rellenar todos los campos que aparecen: tu nombre, tu correo, asunto del mensaje y contenido del mismo. Cuando todo esté listo, al pulsar en enviar el usuario será redirigido al cliente mail que tenga instalado en su dispositivo para finalizar el envío del mensaje. Si se intenta enviar el mensaje sin rellenar algún campo, aparecerá un error en dicho campo (Figura B.A.28).



240- Figura B.A.28- Contacto

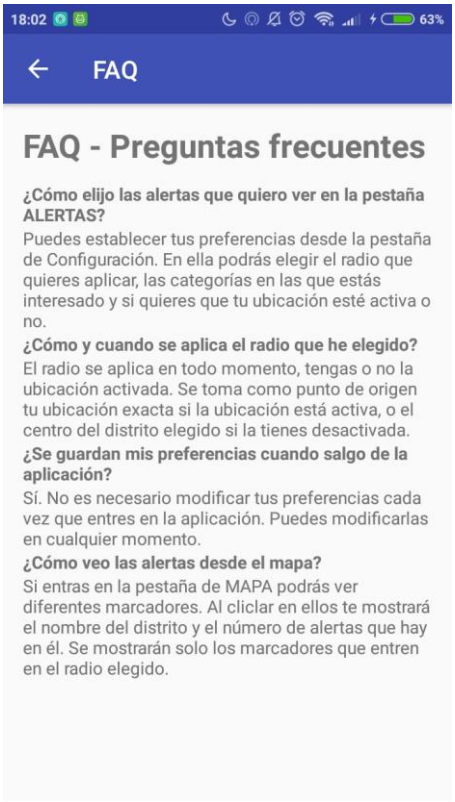
- **‘Soporte’** muestra al usuario los términos y condiciones de uso de la aplicación. Se trata de una pantalla meramente informativa (Figura B.A.29).



241- Figura B.A.29- Soporte

- ‘About us’ presenta al usuario información acerca de los creadores de la aplicación. (Figura B.A.30). Ocurre lo mismo con ‘FAQ’, donde se recogen las preguntas más frecuentes que pueden surgirle al usuario durante la navegación y que tienen por objetivo resolver en la medida de lo posible estas dudas (Figura B.A.31).

242- Figura B.A.30- About us



243- Figura B.A.31- FAQ

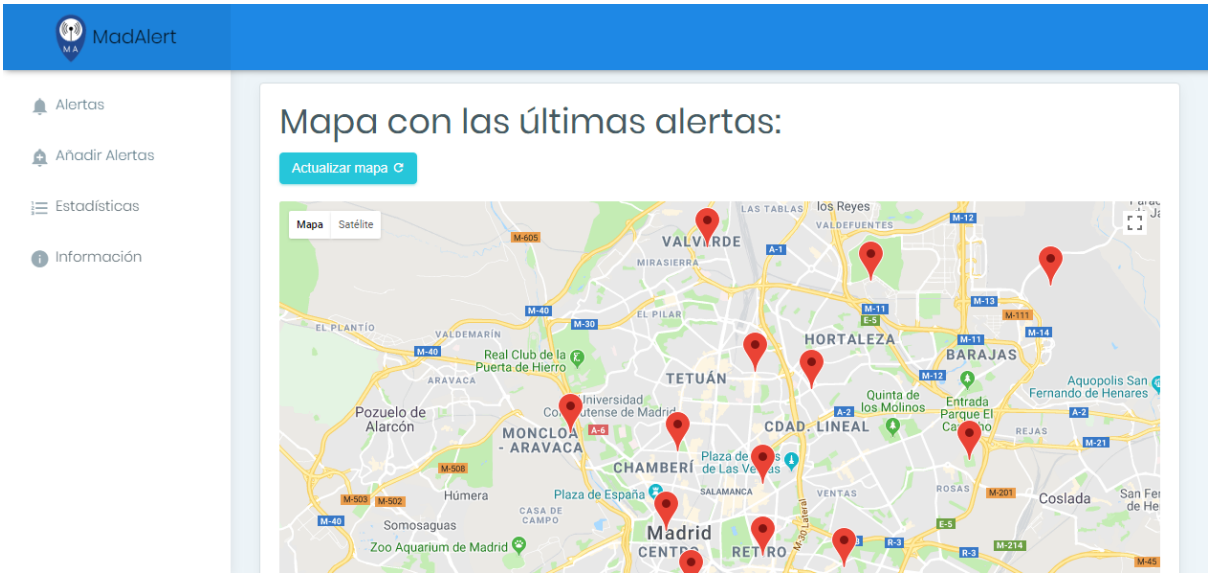
APLICACIÓN WEB

En este apartado se van a explicar detalladamente las instrucciones necesarias para poder realizar correctamente todas y cada una de las funcionalidades que nuestra web ofrece.

244- Figura B.W.1- Logo

En primer lugar, cabe mencionar que si se pulsa sobre el logo y nombre de la web (en la esquina superior izquierda), se accede a la página de inicio (desde cualquier parte de la web). Con esto se permite que, en tan sólo un click y de manera muy sencilla, el usuario tenga la posibilidad de acceder en todo momento a la página principal.

La pantalla de inicio es la primera pantalla que se muestra al usuario. En ella se puede interaccionar o bien con el mapa o bien con el menú lateral de la izquierda.



245- Figura B.W.2- Página de inicio

1. Mapa (Figura B.W.2):

En el mapa de la página principal se pueden visualizar aquellos distritos de Madrid que tienen alertas en las últimas horas (Figura B.W.3).

246- Figura B.W.3- Mapa

Para desplazarse por el mapa es necesario mover el ratón, pero al mismo tiempo mantenerlo pulsado.

Si se desea actualizar las alertas del mapa basta con pulsar sobre el botón azul de “Actualizar mapa” que aparece justo encima del mapa.

Si se pulsa sobre el marcador de un distrito en concreto aparece lo que se puede ver en la Figura B.W.4.

247- Figura B.W.4- Marcador mapa

Al pinchar sobre cada marcador es posible ver tanto el nombre del distrito como el número de alertas que tiene. Para ver cuáles son las alertas, existe un botón azul de “Ver alertas” (Figura B.W.4), con el que se muestran las alertas de ese distrito en orden descendente y de forma detallada (Figura B.W.5).

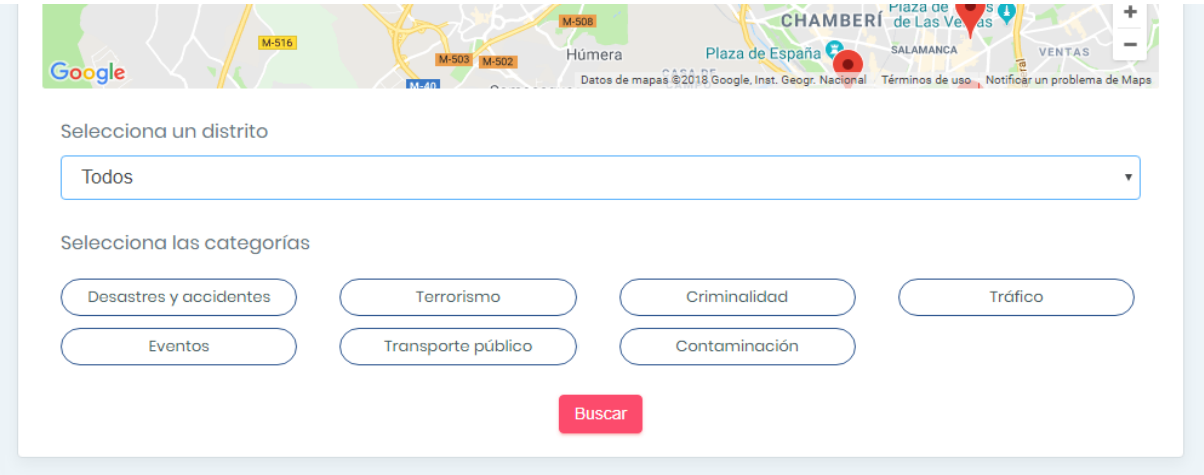
248- Figura B.W.5- Alertas distrito Retiro

Si se desea leer detalladamente la noticia, ver alguna foto etc. simplemente se puede pulsar sobre el título que aparece en azul (se trata de un link a la noticia) y automáticamente se abrirá una nueva pestaña con la página correspondiente a dicha noticia (Figura B.W.6).

249- Figura B.W.6- Página noticia

Visualización de alertas por categoría y distrito:

Esto es una de las posibles formas de ver las alertas de cada distrito, pero si además se quiere realizar una búsqueda por categorías, justo debajo del mapa (es necesario hacer scroll, es decir, bajar) se puede encontrar lo siguiente:



250- Figura B.W.7- Página noticia Parte inferior mapa

En primer lugar, aparece un desplegable con todos los distritos y a continuación todas las categorías.

Por ejemplo, si se quiere buscar las alertas que han ocurrido en cualquiera de los distritos relacionadas con Desastres y accidentes, Criminalidad y Transporte Público, se deben seleccionar las tres categorías (las categorías seleccionadas aparecen de color azul en vez de blanco) y pulsar en el botón rosa de “Buscar” (Figura B.W.8).

251- Figura B.W.8- Selección categorías mapa

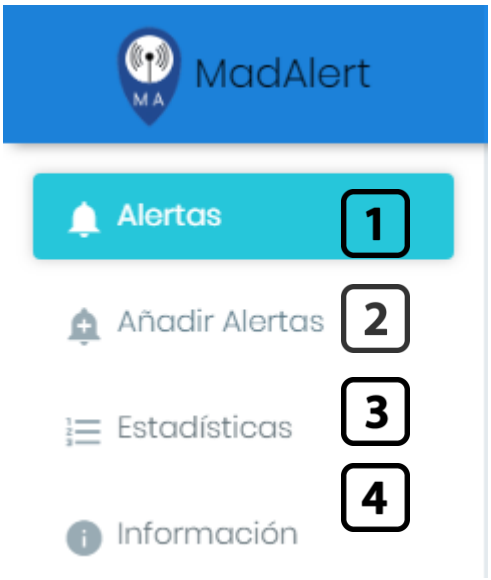
Una vez realizada la búsqueda aparecen las alertas según lo seleccionado anteriormente y ordenadas de forma descendente, es decir, la más actual aparecerá la primera (Figura B.W.9).

252- Figura B.W.9 - Alertas distritos por categorías

Si se desea actualizar las alertas basta con pulsar sobre el botón azul de “Actualizar alertas” que aparece justo antes del nombre del distrito.

Cabe mencionar que dicha actualización se realiza teniendo en cuenta lo anteriormente seleccionado en la búsqueda. Es decir, si se han buscado las alertas del distrito Arganzuela cuando se actualiza seguirán apareciendo las alertas de exclusivamente dicho distrito. A su vez también se tienen en cuenta las categorías seleccionadas en la búsqueda.

2. Menú lateral izquierdo:



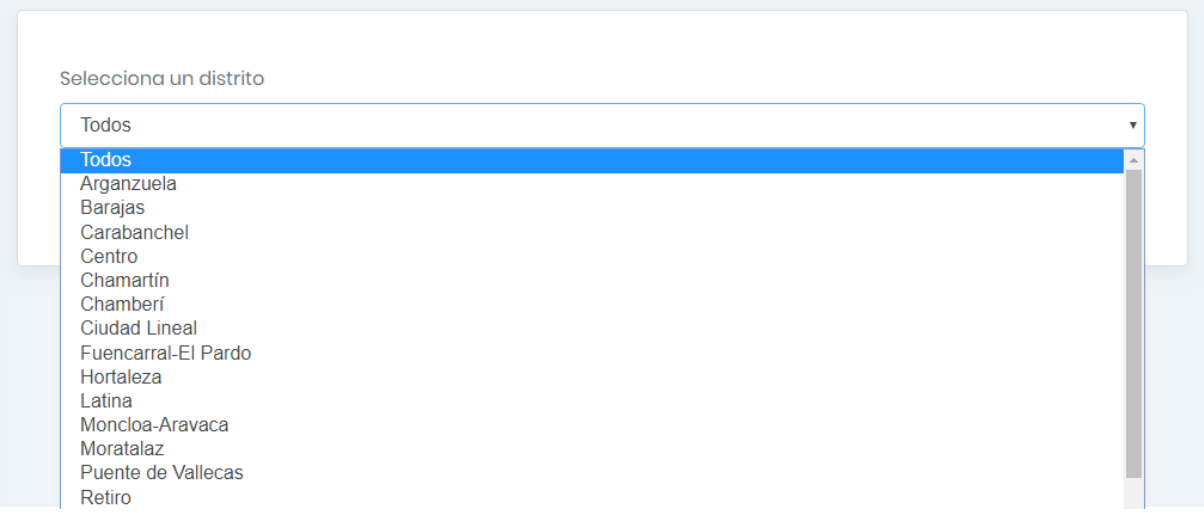
A través de este menú lateral (Figura B.W.10) se permite al usuario poder acceder de una manera rápida y sencilla a todas las páginas de nuestra web.

A continuación, se verá detalladamente cada uno de estos 4 puntos indicados en la figura de la izquierda.

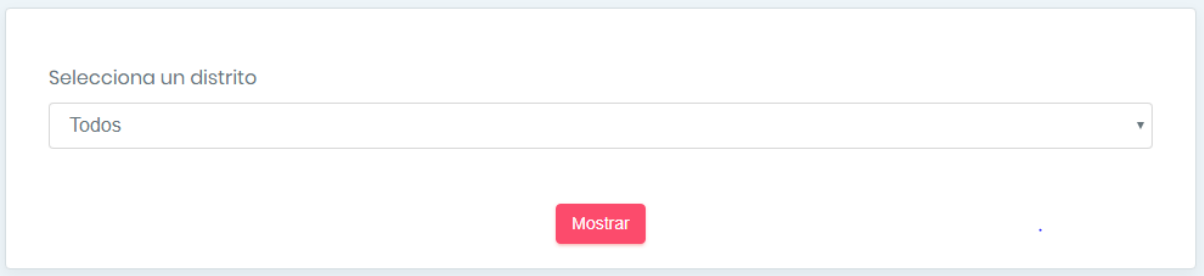
253- Figura B.W.10 - Menú lateral izquierdo

- Pulsando en **Alertas (1** de la Figura B.W.10**)**:

En primer lugar, aparece un menú desplegable (Figura B.W.11) en el que se debe seleccionar el distrito del que se desea leer las alertas.



254- Figura B.W.11 - Selección de distrito



255- Figura B.W.12 - Mostrar distrito seleccionado

Después de seleccionar el distrito y pulsar en el botón de “Mostrar” (Figura B.W.12), inmediatamente debajo aparecen las alertas del distrito seleccionado ordenadas de forma descendente (Figura B.W.13).

256- Figura B.W.13 - Listado de alertas por distrito

Como ya hemos dicho anteriormente, si se desea actualizar las alertas basta con pulsar sobre el botón azul de “Actualizar alertas” que aparece justo encima del nombre del distrito.

- Pulsando en **Añadir alertas (2** de la Figura B.W.10**)**:

El usuario tiene la posibilidad de añadir una nueva alerta. Para ello aparece un formulario (Figura B.W.14) con los siguientes campos a rellenar:

257- Figura B.W.14 - Formulario añadir nueva alerta

Para poder añadir correctamente una nueva alerta es necesario rellenar todos los campos “correctamente” y a continuación pulsar sobre el botón de “Añadir alerta”.

Cuando decimos “correctamente” nos referimos realmente a que la alerta debe pertenecer a la categoría seleccionada y el email debe ser válido.

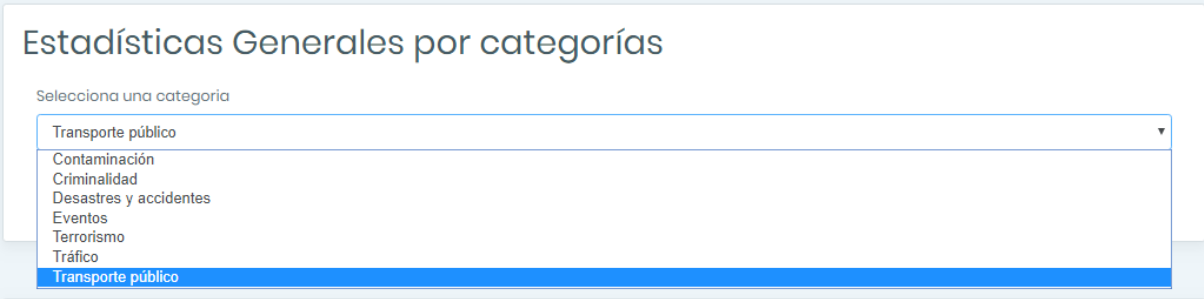
- Pulsando en **Estadísticas (3** de la Figura B.W.10**)**:

Al disponer de diferentes tipos de estadísticas, va a aparecer un submenú (Figura B.W.15) justo debajo de Estadísticas (menú lateral izquierdo) en el que se pueden observar los tres tipos de estadísticas disponibles:

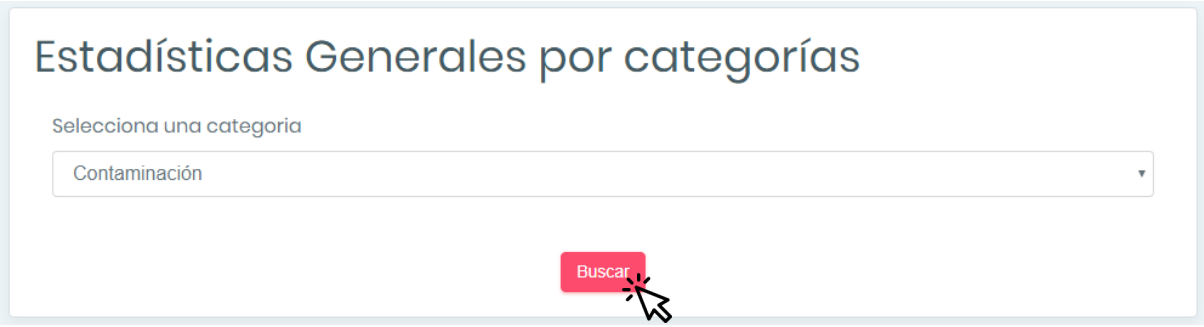
258- Figura B.W.15 - Submenú Estadísticas

1. **Estadísticas generales** por categorías (Figura B.W.15).

En primer lugar, aparece un menú desplegable (Figura B.W.16) en el que se debe seleccionar la categoría de la que desea ver las estadísticas.



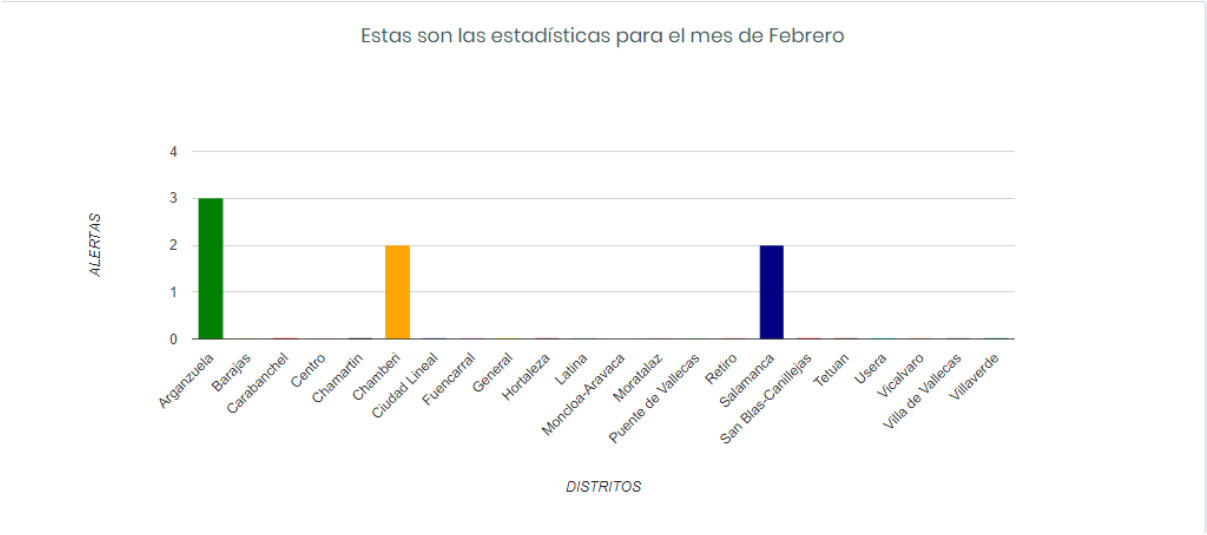
259- Figura B.W.16 - Estad. Generales desplegable categorías



260- Figura B.W.17 - Buscar est. generales por categoría seleccionada

Después de seleccionar la categoría deseada y pulsar en el botón de “Buscar” (Figura B.W.17), inmediatamente debajo aparecen las estadísticas de las alertas de la categoría seleccionada. En primer lugar, aparecen la del mes anterior al actual (es decir, el último mes del que se han podido realizar las estadísticas) (Figura B.W.18) y a continuación el anterior a éste (Figura B.W.19):

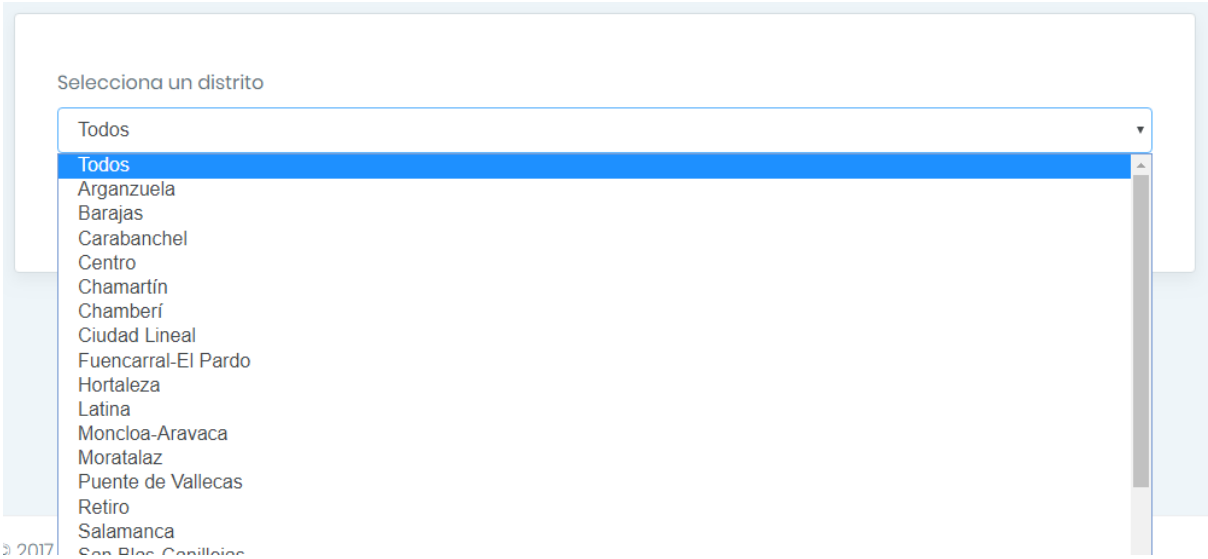
261- Figura B.W.18 - Estadísticas generales mes 1



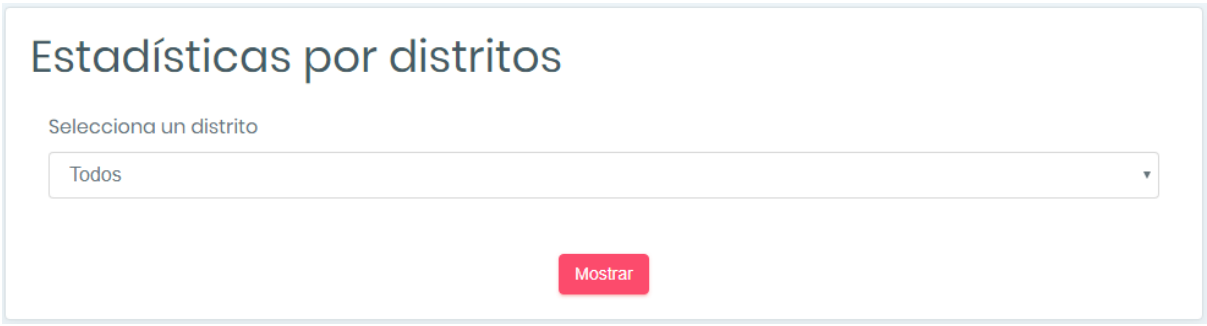
262- Figura B.W.19 - Estadísticas generales mes 2

2. **Estadísticas por distritos** (Figura B.W.15).

En primer lugar, aparece un menú desplegable en el que se debe seleccionar el distrito del que desea ver las estadísticas:



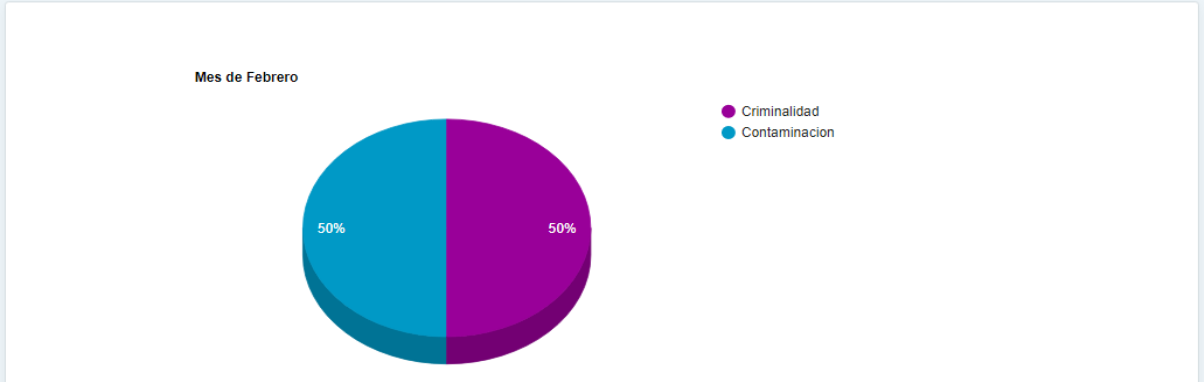
263- Figura B.W.20 - Desplegable Distrito



264- Figura B.W.21 - Mostrar est. distrito seleccionado

Después de seleccionar el distrito y pulsar en el botón de “Mostrar” (Figura B.W.21), inmediatamente debajo aparecen las estadísticas de las alertas del distrito seleccionado. En primer lugar, aparecen las del mes anterior al actual (es decir, el último mes del que se han podido realizar las estadísticas) (Figura B.W.22) y a continuación el anterior a éste (Figura B.W.23):

265- Figura B.W.22 - Estadísticas por distritos mes 1



266- Figura B.W.23 - Estadísticas por distritos mes 2

3. **Estadísticas** de los datos obtenidos de la **Policía Municipal** de Madrid (Figura B.W.15).

En primer lugar, aparece un menú desplegable en el que se debe seleccionar el tipo de estadísticas que se desea ver:

267- Figura B.W.24 - Desplegable Estadísticas Policía

268- Figura B.W.25 - Mostrar Estadísticas Policía seleccionadas

Después de seleccionar el tipo de estadísticas y pulsar en el botón de “Mostrar” (Figura B.W.25), inmediatamente debajo aparecen las estadísticas del tipo seleccionado.

Por ejemplo, si se decide ver las estadísticas relacionadas con la seguridad de Madrid se verá un gráfico como el siguiente:

269- Figura B.W.26 - Gráfico estadísticas seguridad

- Pulsando en **Información (4** de la Figura B.W.10**).**

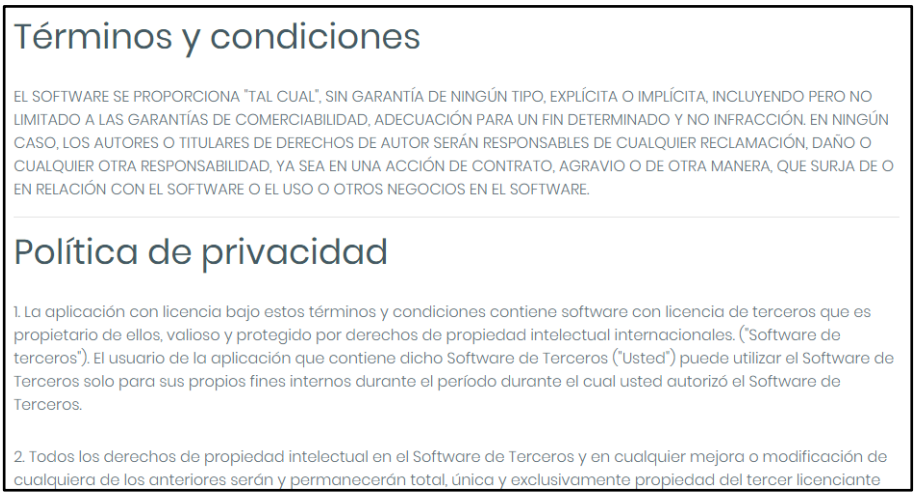
Al disponer de diferentes páginas de información, va a aparecer un submenú justo debajo de Información (menú lateral izquierdo) en el que se pueden observar (en la Figura B.W.27) los dos tipos de información disponibles:

1. **Soporte** (términos y condiciones, política de privacidad…)
2. **About us** (información sobre los miembros de este grupo que ha creado la página web)

270- Figura B.W.27 - Submenú Información

- Pulsando en **Soporte (1** de la Figura B.W.27**):**

En esta página se podrán leer los términos y condiciones del software y la política de privacidad:



Términos y condiciones

EL SOFTWARE SE PROPORCIONA "TAL CUAL", SIN GARANTÍA DE NINGÚN TIPO, EXPLÍCITA O IMPLÍCITA, INCLUYENDO PERO NO LIMITADO A LAS GARANTÍAS DE COMERCIABILIDAD, ADECUACIÓN PARA UN FIN DETERMINADO Y NO INFRACCIÓN. EN NINGÚN CASO, LOS AUTORES O TITULARES DE DERECHOS DE AUTOR SERÁN RESPONSABLES DE CUALQUIER RECLAMACIÓN, DAÑO O CUALQUIER OTRA RESPONSABILIDAD, YA SEA EN UNA ACCIÓN DE CONTRATO, AGRAVIO O DE OTRA MANERA, QUE SURJA DE O EN RELACIÓN CON EL SOFTWARE O EL USO O OTROS NEGOCIOS EN EL SOFTWARE.

Política de privacidad

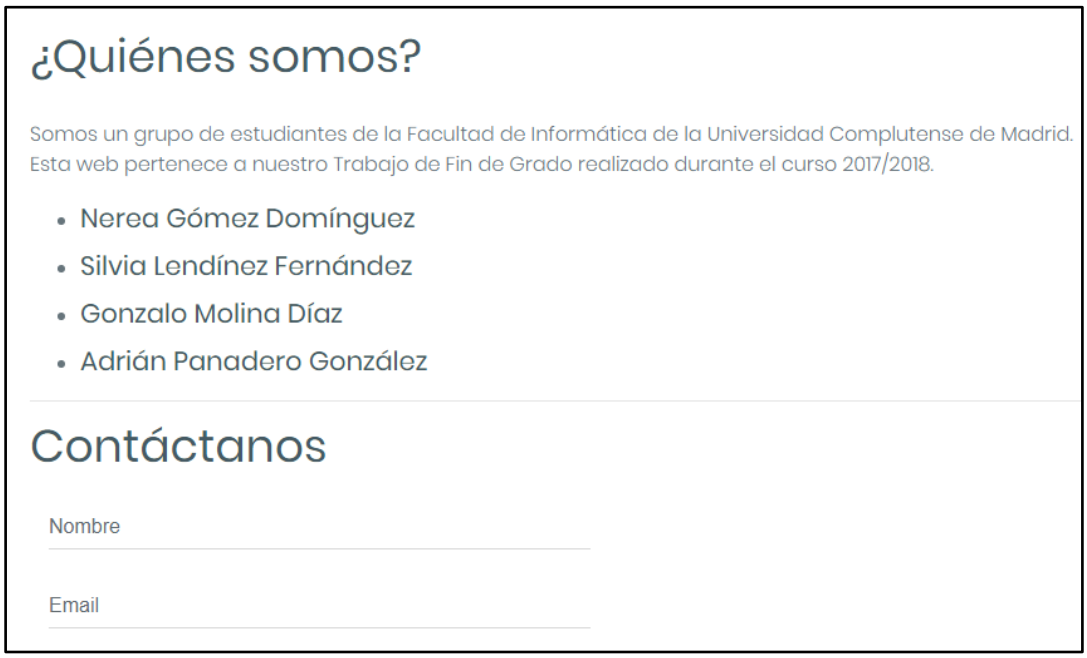
1. La aplicación con licencia bajo estos términos y condiciones contiene software con licencia de terceros que es propietario de ellos, valioso y protegido por derechos de propiedad intelectual internacionales. ("Software de terceros"). El usuario de la aplicación que contiene dicho Software de Terceros ("Usted") puede utilizar el Software de Terceros solo para sus propios fines internos durante el período durante el cual usted autorizó el Software de Terceros.

2. Todos los derechos de propiedad intelectual en el Software de Terceros y en cualquier mejora o modificación de cualquiera de los anteriores serán y permanecerán total, única y exclusivamente propiedad del tercer licenciante

271- Figura B.W.28 - Información Soporte

- Pulsando en **About us (2** de la Figura B.W.27**):**

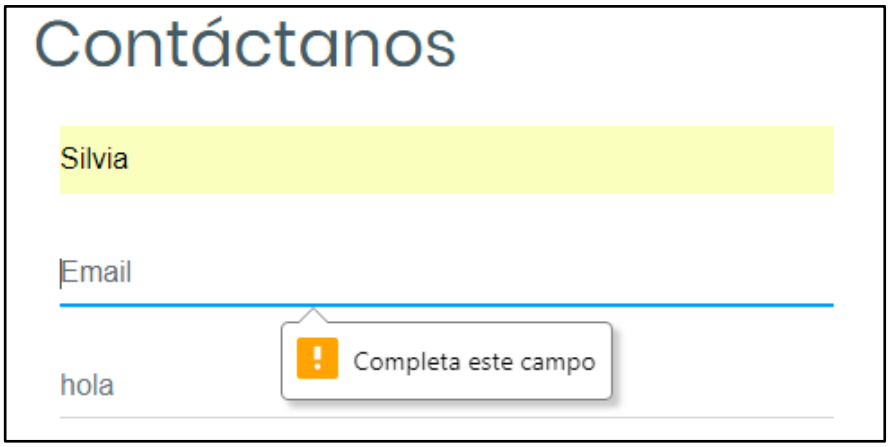
En esta página no sólo se podrá leer la información de los creadores (Figura B.W.29) de la web, sino que además se tendrá la posibilidad de rellenar un formulario de contacto (Figura B.2.30):



272- Figura B.W.29 - Información About us

273- Figura B.W.30 - Formulario de contacto

En el formulario de la figura anterior es necesario rellenar los campos correspondientes y a continuación pulsar en el botón de “Enviar”. Si no se rellenan todos los campos aparecerá un mensaje como el de la figura siguiente indicando qué campo está aún por completar:



274- Figura B.W.31 - Error en formulario de contacto