UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Máster en Ingeniería de Computadores y Redes

Trabajo Fin de Máster

Diseño de una arquitectura servidora para la gestión de soluciones basadas en Crowdsensing

Autor: *Vera Burgos Elsa Patricia*

Director(es): *Dr.* Carlos Tavares Calafate

20 de diciembre de 2017

Resumen

La monitorización de la contaminación acústica se ha convertido en un requisito esencial para los países industrializados y desarrollados de todo el mundo. La manera tradicional para medir la contaminación acústica es mediante el uso de sonómetros profesionales, los cuales son caros y de gran tamaño. Actualmente, con la aparición de dispositivos móviles inteligentes ricos en sensores, ha surgido un nuevo paradigma, llamado crowdsensing, el cual aprovecha la capacidad de detección de multitudes de móviles, convirtiéndose en un campo popular de investigación y aplicaciones.

En este trabajo proponemos GRCSensing, un sistema de monitorización ambiental basado en crowdsensing que es capaz de medir la contaminación acústica. GRCSensing consta de un ciclo de vida de cuatro etapas, es decir, (i) creación de tareas, (ii) asignación de tareas, (iii) ejecución de tareas, y (iv) visualización de información de contaminación acústica de un área determinada en tiempo real. Para ello, el sistema ha sido desarrollado con los siguientes recursos: un servidor web que aloja la plataforma web, y que permite la creación de tareas y visualización de los niveles contaminación acústica, así como dispositivos móviles con SO Android para la recolección de los datos de ruido ambiental. En conjunto, con todos los dispositivos antes mencionados, se ha conseguido la transmisión y captura de datos de ruido ambiental. Esto da como resultado un sistema de información que ayuda al estudio de la contaminación acústica de una región específica de manera sencilla y barata.

Palabras clave: Crowdsensing, Monitorización Ambiental, Contaminación Acústica, Android

Abstract

The monitoring of noise pollution has become an essential requirement for industrialized and developed countries around the world. The traditional way to measure noise pollution is through the use of professional sound level meters, which are expensive and large-sized. Currently, with the emergence of smart mobile devices that are endowed with multiple sensors, a new paradigm has emerged, called crowdsensing, which takes advantage of the ability to detect mobile crowds, becoming a popular field of research and applications.

In this work we propose GRCSensing, an environmental monitoring system based on crowdsensing that is able to measure noise pollution. GRCSensing consists of a life cycle of four stages, that is, (i) task creation, (ii) task assignment, (iii) task execution, and (iv) visualization of acoustic contamination information of a specific area in real time. To this aim, the system has been developed with the following resources: a web server that hosts the web platform, and that allows the creation of tasks and visualization of noise pollution levels, as well as mobile devices with Android OS for the collection of environmental noise data. By combining all the aforementioned devices, the transmission and capture of environmental noise data becomes possible. This results in an information system that helps at studying noise pollution in a specific region in a simple and inexpensive way.

Keywords: Crowdsensing, Environmental Monitoring, Acoustic Pollution, Android.

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Motivación

En la actualidad la degradación del medio ambiente es uno de los mayores problemas que se plantea el hombre. El crecimiento económico, el desarrollo industrial, el aumento de vehículos en circulación, la expansión demográfica y las grandes concentraciones urbanas crean toda una serie de condicionantes que afectan, en menor o mayor grado, la calidad del medio ambiente [1].

Diferentes estudios [1], [2], [3], señalan que el desarrollo incontrolado de las actividades del hombre en las sociedades industrializadas han alterado más del 75% de la superficie de la tierra, lo cual ha originado contaminación del aire, del agua, la desaparición de zonas verdes, y el incremento incesante del ruido ambiental. En particular, este último elemento se puede convertir en una de las mayores fuentes de malestar de la vida moderna [4].

Es por esto que, las investigaciones realizadas en este campo [5], [6], han demostrado que las personas sometidas a ruido[1] de forma contínua presentan trastornos como: pérdida de la capacidad auditiva, trastornos gastrointestinales, alteración de la actividad cerebral, cardiaca y respiratoria. Además, en otros trabajos [7], [8], los autores afirman que el ruido también produce alteraciones conductuales tales como perturbación del sueño y el descanso, dificultades para la comunicación, irritabilidad, agresividad, así como problemas para desarrollar la atención y concentración mental.

La preocupación actual por la protección del medio ambiente se pone de manifiesto en la lucha contra el ruido. De hecho, diferentes estudios [2], [9], [10], han destacado la importancia del control del ruido en zonas con alta densidad de población. Se estima que, en los países de la Unión Europea cerca de 270 millones de ciudadanos están expuestos a niveles de contaminación acústica por encima del LEQ (nivel de presión acústica equivalente) de 65 decibelios, límite

[1] Sonido desagradable y molesto que, en niveles no necesariamente altos, son potencialmente nocivos para el aparato auditivo y el bienestar fisiológico y psicológico.

de tolerancia recomendado por la OMS[2], siendo España el país más ruidoso de Europa y el segundo de la OCDE[3] después de Japón [11] , [12]. Al mismo tiempo, los gobiernos de los países desarrollados y la Agencia Europea de Medio Ambiente ha promulgado normas que tratan de limitar la contaminación sonora en las ciudades [13], [14].

Respecto a lo anterior, existen soluciones tradicionales [11], [15] que miden la contaminación acústica mediante el uso de sonómetros profesionales que, aunque son de considerable coste y tamaño, ofrecen alta precisión y sensibilidad. Normalmente, estas sesiones de medición tienen lugar sólo en unos pocos puntos accesibles, y durante intervalos de tiempo cortos (por ejemplo, treinta minutos). Por eso, las soluciones actuales para medir la contaminación acústica han surgido de un nuevo paradigma llamado crowdsensing [16], [17]. Esta solución aprovecha los recursos y sensores integrados en teléfonos inteligentes, y promueve la monitorización colaborativa en áreas pobladas. En ambas soluciones los datos recopilados se almacenan en servidores para su análisis posterior.

La propuesta que se pretende llevar a cabo en este Trabajo Fin de Máster es el desarrollo de un sistema de monitorización ambiental basado en el paradigma de crowdsensing. Para ello se integra una arquitectura de servicios cliente-servidor a través de dispositivos móviles con el sistema operativo Android, de tal forma que se pueda obtener información de contaminación acústica en tiempo real de un punto determinado, para ser almacenada posteriormente en una base de datos y generar mapas de niveles contaminación, los cuales se pueden consultar a través de un portal web.

[2] Organización Mundial de la Salud
[3] Organización para la Cooperación y el Desarrollo Económico

1.2. Objetivos

El objetivo de este Trabajo de Fin de Master es el desarrollo de un sistema de monitorización de la contaminación acústica en una ciudad mediante crowdsensing. Concretamente, nos centraremos en el lado del servidor, el cual realizará las definiciones y el envío de las tareas a los dispositivos móviles, además de la recepción de información de contaminación acústica por parte de los clientes.

Para alcanzar el objetivo general se definen los siguientes objetivos específicos:

- Diseñar un sistema web que permita la administración y creación de tareas de monitorización acústica.
- Proponer una interfaz de comunicaciones que permita enviar y leer información de las tareas creadas y de los valores de contaminación obtenidos entre el servidor web y el cliente móvil de la manera más óptima posible.
- Crear una base de datos que permitirá el almacenamiento de las tareas definidas por el servidor y de los datos de contaminación acústica recolectados por el cliente móvil.
- Diseñar una aplicación en SO Android con la ayuda de APIs, permitiendo la recolección de datos de contaminación acústica.
- Tratar los datos con el fin de obtener mapas de contaminación de forma visual.

1.3. Estructura del documento

Capítulo II – Trabajos relacionados: En este capítulo se realiza una revisión bibliográfica de los estudios más relevantes realizados hasta la fecha.

Capítulo III - Tecnología empleada: En este capítulo se describen los componentes tecnológicos que forman parte de la solución planteada. El capítulo finaliza con la tecnología empleada para la comunicación entre las aplicaciones.

Capítulo IV – Arquitectura Propuesta: En este capítulo se describe en profundidad la solución implementada en el proyecto, detallando la arquitectura, y describiendo el sistema de administración y creación de tareas de crowdsensing.

Capítulo V – Diseño del Sistema: En este capítulo se describe la interfaz de los dos componentes del sistema propuesto.

Capítulo VI – Resultados experimentales: En este capítulo se muestran los resultados que se han obtenido mediante la realización de pruebas, evidenciando los resultados obtenidos en la aplicación móvil y en el sistema web.

Capítulo VII - Conclusiones: En este último capítulo se describen los principales logros alcanzados y los trabajos futuros.

2. Trabajos Relacionados

En esta sección vamos a realizar una revisión bibliográfica de los trabajos relacionados en las dos áreas que se combinan en este trabajo: Medición de la contaminación acústica mediante varios métodos, y medición de la contaminación acústica usando crowdsensing

2.1. Medición de la Contaminación Acústica

La contaminación acústica es cada vez más importante, especialmente en los países industrializados y desarrollados El ruido es un serio problema medioambiental, que causa molestias e interrupciones en las actividades diarias. Como consecuencia, podemos encontrar muchos trabajos relacionados con este tema en la literatura.

J. Granada et al. [18] y M. Marinov et al. [19] han realizado un estudio para identificar la correlación entre la contaminación acústica y su efecto sobre la salud humana. El objetivo principal fue medir los niveles de contaminación acústica en áreas de alto flujo de tráfico. Las mediciones de la contaminación acústica fueron obtenidas por los nodos sensores en una red de sensores inalámbricas (WSN). En detalle [19], los resultados recogidos de los diferentes escenarios de prueba demostraron una fuerte correlación positiva entre el nivel de ruido y el aumento del riesgo de enfermedades cardiovasculares como, ataques cardíacos y accidentes cerebrovasculares.

T. Yung-chung et al. [20] nos presentan un trabajo en el que describen la estructura de un sistema de recolección, almacenamiento, y visualización de datos de contaminación acústica a través de la creación de un prototipo que incluye un módulo de micrófono y un sistema empotrado Arduino.

D. Radu et al. [21] proponen un sistema de monitorización de ruido que utiliza una red VANET para la recogida de datos de detectores de ruido (teléfonos móviles equipados con GPS). Los nodos detectores transmiten periódicamente niveles de ruido a los automóviles vecinos, y los paquetes de datos se comparten y almacenan temporalmente mediante nodos VANET participantes y, en última instancia, se suben a un nodo colector conectado a Internet, proporcionando datos públicos en tiempo real.

2.2. Medición de la Contaminación Acústica usando Crowdsensing

Crowdsensing (CS) surge como una técnica de recolección de la información de variables físicas ambientales para contribuir a mejorar un objetivo social, o conocer la relevancia de un evento de interés para una sociedad concreta, todo ello gracias a la recolección de la información que ofrecen los individuos de la ciudad mediante el uso de dispositivos móviles [22].

Las recolecciones de información mediante CS poseen algunas características específicas. En primer lugar, los dispositivos móviles que participan en la recolección poseen más recursos de computación, comunicación y almacenamiento que un sensor convencional. En segundo lugar, el despliegue de la red ya está realizado una vez que los ciudadanos llevan consigo el dispositivo móvil, facilitando la recopilación de información de zonas sin necesidad de instalar dispositivos adicionales [23].

Por todas estas ventajas, crowdsensing se ha usado para realizar algunos trabajos relevantes.

En la literatura podemos encontrar varias soluciones donde los dispositivos móviles se utilizan para la detección del ruido ambiental. Por ejemplo, Laermometer [24], Ubisound [25], 2cloud [26] y OnoM@p [27] son aplicaciones que han sido diseñadas para recopilar, almacenar y visualizar, elaborando mapas de ruido que representen con precisión las fuentes de ruido urbano. Igualmente, trabajos como [27], [28], permiten comprobar los datos recopilados, y proponen soluciones personalizadas al usuario para reducir la exposición al ruido ambiental.

Muratoni et al. [29] y E. Kanjo [30] propusieron una plataforma para registrar, analizar y monitorizar los niveles de ruido ambiental con bajo impacto en el consumo de batería. En concreto, [30] y [31] ofrecen un paradigma de tiempo real y proporcionan un diseño escalable, gestionando las desconexiones inherentes (cambio de redes Wifi, 3G/4G, entre otras), lo cual significa que la aplicación almacena en caché todos los datos de ruido durante el tiempo que el dispositivo móvil se encuentra sin conexión a la red.

Sense2Health [31], NoizCrowd [32] y Ear-Phone [33] proponen un sistema para estudiar los niveles de ruido utilizando diferentes técnicas de interpolación espacial y temporal para resolver problemas de dispersión de datos. En particular, EarPhone nos describe un algoritmo que pretende optimizar la muestra de ruido detectando si el teléfono se coloca en el bolsillo de un pantalón o en una bolsa.

No sólo existen soluciones que estudian el nivel del ruido para plasmarlos en un mapa de contaminación urbano, sino que también existen propuestas como la de Pryss et al. [34], la cual nos ofrece una plataforma donde los micrófonos de los dispositivos móviles se utilizan como una ayuda para los aspectos médicos del tratamiento del *tinnitus*. De manera similar, Ren et al. [35] han capturado el comportamiento de la respiración durante el sueño utilizando teléfonos inteligentes.

Govindan et al. [36] han propuesto uno de los primeros marcos diseñados para aplicaciones de crowdsensing. Ellos especifican tareas de detección utilizando XML (para dirigir la expresividad), y un lenguaje de flujo de trabajo para especificar los pasos. Sin embargo, la naturaleza de la expresividad se centra en tareas que requieren intervención humana. De manera similar, USense [37] es un sistema que apunta a la creación de un cliente middleware oportunista y pasivo (es decir, que no requiere intervención humana). Está dirigido a aplicaciones de crowdsensing donde se hace una utilización pasiva de los sensores del teléfono inteligente.

En la **Tabla 1** se realiza una clasificación de los trabajos revisados.

Tabla 1. Clasificación de las diferentes investigaciones revisadas.

	Año	CrowdSensing	Gestión de conexiones (real time)	Envió de Tareas en tiempo real	Noise Map	Selección Área	Networking
Laermometer	2008	recolección, almacenamiento y visualización en mapas de calor	no	no	servidor	no	no detallado
NoiseTube	2009	recolección, almacenamiento y visualización en mapas de calor	no	no	móvil servidor	no	Restfull
NoisePSY	2010	recolección, almacenamiento y visualización en mapas de calor	Si	no	servidor	no	Restfull
NoiseMap	2011	recolección, almacenamiento y visualización en mapas de calor	no	no	servidor	no	no detallado
NoiseHound	2011	recolección, almacenamiento y visualización en mapas de calor	no	no	servidor	no	no detallado
USense	2012	Creación de tareas, recolección, almacenamiento y visualización	no detallado	si	no	no	SOAP Web Services (RPC)
WideNoise	2013	recolección, almacenamiento y visualización en mapas de calor	no	no	móvil servidor	no	Restfull
NoizCrowd	2013	recolección, almacenamiento y visualización en mapas de calor	no	no	servidor	no	no detallado
SoundOfTheCity	2013	recolección, almacenamiento y visualización en mapas de calor	no	no	servidor	no	SOAP
NoiseWatch	2014	recolección, almacenamiento y visualización en mapas de calor	no detallado	no detallado	no detallado	no detallado	no detallado
Sense2Health	2015	recolección, almacenamiento y visualización en mapas de calor	si	no	móvil servidor	no	RabbitMQ [38]
Ear-Phone 2015	2015	recolección, almacenamiento y visualización en mapas de calor	no	no	servidor	no	no detallado

3. Descripción de las Tecnologías Empleadas

En este apartado se describen las diferentes tecnologías de software empleadas para la realización de este proyecto.

3.1. Android

Android es un Sistema Operativo para dispositivos móviles. La estructura de este sistema se basa en el kernel de Linux, y se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos ejecutándose en la máquina virtual de Dalvik. Las aplicaciones se desarrollan en lenguaje Java con Android Software Development Kit que contiene las herramientas necesarias para el desarrollo de aplicaciones.

Para este trabajo se eligió Android debido a que, actualmente, es uno de los sistemas operativos más utilizados; el IDE de desarrollo que se ha utilizado es Android Studio en su versión 2.3.3 del 6 de junio de 2017 [39].

3.1.1. Arquitectura de Android

La arquitectura de Android está compuesta por cuatro partes agrupadas en capas, y siguiendo una estructura denominada pila (stack en inglés), como se muestra en la **Figura 1**, permitiendo que cada capa utilice los elementos de la capa inferior para realizar sus funciones. Una de las características más importantes de esta arquitectura es que todas las capas están basadas en software libre. A continuación se describe cada una de las capas.

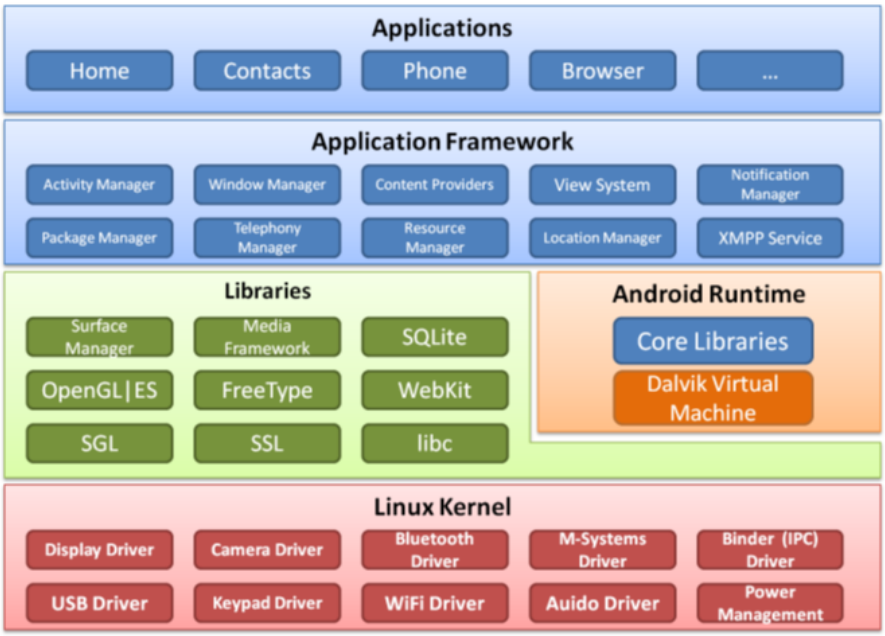

Figura 1. Arquitectura de Android.

- **Linux Kernel:** Este nivel proporciona servicios principales como la seguridad, el manejo de la memoria, los elementos de comunicación, el multiproceso, la pila de protocolos, y el soporte de drivers para dispositivos.
- **Librerías nativas:** Este nivel incluye un conjunto de librerías básicas escritas en C o C++, y compiladas para la arquitectura hardware del teléfono. Algunas de estas librerías son: SGL (Scalable Graphics Library), OpenGL ES, FreeType, SQLite, y SSL, entre otras.
- **Runtime de Android:** Este nivel no se considera como una capa en sí misma debido a que también está formada por un conjunto de librerías. El componente principal del entorno de ejecución es la máquina virtual Dalvik, la cual está diseñada para que facilite la optimización de recursos [1];
- **Marco de aplicaciones o Framework:** Este nivel proporciona una plataforma que incluye todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones (sensores, localización, servicios, barra de notificaciones, entre otros).
- **Aplicaciones:** En este nivel se incluyen las aplicaciones base tales como: cliente de correo, programa de mensajería, calendario, mapas, y contactos, entre otras, así como las instaladas por el usuario.

3.1.2. SQLite

SQLite es un motor de base de datos Open Source que está incrustado en Android. A diferencia del sistema de gestión de bases de datos cliente-servidor, SQLite no es un proceso independiente con el que la aplicación principal se comunica; en lugar de eso, la biblioteca SQLite se acopla en el programa pasando a incorporarlo, por lo que éste lee y escribe directamente a archivos de disco ordinarios. SQLite es la opción ideal para manipular datos dentro del Sistema Operativo Android.

3.2. Servidor Web

Un servidor web es un programa informático que atiende y responde a las diversas peticiones de los navegadores, proporcionándoles los recursos necesarios, sean estos una página web o información de acuerdo con lo que solicitan. La transmisión de estos datos se realiza mediante el protocolo HTTP o HTTPS.

El esquema de funcionamiento de un servidor web básico se basa en esperar peticiones en el puerto TCP asignado (el estándar para HTTP es el 80), recibir la petición del cliente, buscar el recurso en la cadena de peticiones y, para finalizar, entregar el recurso por la misma conexión por donde se ha recibido la petición. El cliente puede enviar peticiones al servidor web mediante los siguientes métodos:

GET: permite obtener información del servidor. Para esto tiene que solicitar los datos, ya sea un archivo o una base de datos. Este método es visible para el usuario debido a que el medio de envío es la URL.

POST: permite enviar información desde el cliente para que sea procesada, permitiendo que se actualice o agregue información en el servidor. Estos datos procesados dan como respuesta la carga o actualización de una página con información. Este método es invisible para el cliente debido a que los datos enviados aparecen en el cuerpo del mensaje HTTP.

A partir del esquema anterior se han construido los programas servidores HTTP como Apache, Nginx y Microsoft IIS, entre otros; variando solo el tipo de

petición; se pueden servir paginas estáticas, CGI, o executar Servlets, entre otros. Para este trabajo se ha utilizado el servidor web Nginx.

3.2.1. NGINX

NGINX es un servidor web HTTP y proxy inverso de código abierto de alto rendimiento que incluye servicios de correo electrónico con acceso al Internet Message Protocol (IMAP) y al servidor Post Office Protocol (POP). Además, Nginx en modo de proxy inverso se utiliza para equilibrar la carga entre los servidores back-end, o también para proporcionar almacenamiento en caché para un servidor back-end lento [40]. La característica fundamental de este servidor web es que se trata de un software asíncrono, a diferencia de Apache que está basado en procesos. La ventaja de ser asíncrono es su escalabilidad permitiendo gestionar las peticiones en muy pocos hilos, y reduciendo así las posibilidades de sobrecarga en el servidor.

3.2.2. Django Framework Web

Es un framework web de código abierto escrito en Python, inicialmente desarrollado para gestionar aplicaciones web de páginas orientadas a noticias de Lawrence Journal-World (Revista de publicación de noticias). Permite construir aplicaciones web más rápidas y con menos código usando los principios y los patrones de diseño que animan a la creación del código reusable. En particular, hace uso del principio de no repetirse (DRY) para que no posea duplicación innecesaria, reduciendo la cantidad de código.

3.2.2.1. Arquitectura de Django

La arquitectura de Django se basa en el patrón Modelo-Vista-Controlador (MVC). Sin embargo, Django define su estructura con una pequeña diferencia: lo que se llamaría "controlador" en un framework MVC, en Django se llama "vista", y lo que se llamaría "vista" se llama "plantilla" (*template*); entonces diríamos que éste es un framework MVT [41]. A continuación, describiremos que hace cada uno de ellos con un poco más de detalle.

- **M significa "Model" (Modelo),** es la capa de acceso a la base de datos, y contiene toda la información sobre los datos, es decir; acceso, validación

y las relaciones entre los datos. En la **Figura 2** se muestra un ejemplo de código que se aplica en esta capa.

```
@python_2_unicode_compatible
class Tarea(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    nombre = models.CharField(max_length=80, default='')
    descripcion = models.CharField(max_length=400)
    fecha_creacion = models.DateField('Fecha de creacion',
    default=timezone.now)
    estado = models.CharField(max_length=2, default='c')
    def __str__(self):
        return self.descripcion
    class Meta:
        verbose_name_plural = "Tareas"
```

Figura 2. Ejemplo de código de la capa Modelo.

- **T significa "Template" (Plantilla),** es la capa de presentación, contiene las decisiones relacionadas con la presentación, es decir, permite crear contenido HTML, XML, CSS, y JavaScript, entre otros, que se muestran sobre un navegador web. En la **Figura 3** se muestra un ejemplo de código que se aplica en esta capa.

```
<!DOCTYPE html>
{% load static %}
<html lang="es">
<head>
{% block head %}{% endblock %}
<title>{% block title %}{% endblock %}</title>
</head>
<body>
{% block content %}{% endblock %}
</div>
{% block extrajs %}{% endblock %}
</body>
</html>
```

Figura 3. Ejemplo de código de la capa Plantilla.

- **V significa "View" (Vista),** y es la capa de la lógica de negocios. Contiene la lógica que accede al modelo, y la delega a la plantilla apropiada; se puede pensar en la vista como un puente entre el modelo y las plantillas (template). En la **Figura 4** se muestra un ejemplo de código que se aplica en esta capa.

```
# Administracion de tareas
def admin_tareas(request):
    template = loader.get_template('sensing/admin_tareas.html')
    context = {
            'tareas': Tarea.objects.all(),
            'tarea_id': random.randrange(50000, 51000),
            }
    return HttpResponse(template.render(context, request))
```

Figura 4. Ejemplo de código de la capa Vista.

3.2.3. JavaScript

JavaScript es un lenguaje de programación interpretado que surgió por la necesidad de ampliar las posibilidades del HTML, y se utiliza principalmente del lado del cliente permitiendo crear efectos atractivos y dinámicos en las páginas web. Además, al ser un lenguaje de programación interpretado, no es necesario compilar los programas para ejecutarlos, si no que se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

Para nuestro trabajo, JavaScript se utiliza para agregar algunas funcionalidades al sistema web como:

1. Insertar mapa de Google
2. Seleccionar o eliminar, (según sea el caso) un área determinada en el mapa de Google, y
3. Dibujar el mapa de calor de los valores de la contaminación acústica (dB) en el mapa de Google

3.2.4. MySQL

MySQL es un sistema de gestión de base de datos relacionales, de código abierto, multihilo y multiusuario más usado en el desarrollo de aplicaciones web ligeras. Su popularidad se debe a que se suele usar juntamente con PHP, y a veces se utiliza Perl o Python, en lugar de PHP.

3.2.5. Pyrebase

Es una API Rest escrita en Python que permite la comunicación entre nuestro servidor y Firebase [42]. Su configuración se muestra en la **Figura 5**.

```
import pyrebase
    config = {
            "apiKey": "AIzaSyD5AMhu0HFHHaaxKSctxUBpwQ4axbPL-mM",
            "authDomain": "grcsensing-396c0.firebaseapp.com",
            "databaseURL": "https://grcsensing-396c0.firebaseio.com",
            "storageBucket": "grcsensing-396c0.appspot.com",
            "serviceAccount": settings.BASE_DIR +
            "/sensing/serviceAccountCredentials.json",
            }
    firebase = pyrebase.initialize_app(config)
```

Figura 5. Ejemplo de código de configuración del API Pyrebase.

3.2.6. Json

JSON (JavaScript Object Notation) es un formato basado en texto, usado para el intercambio de datos entre aplicaciones cliente-servidor o aplicaciones que se ejecuten en diferentes escenarios [43]. Para el caso de nuestro proyecto se hizo uso de JSON para el intercambio de datos entre aplicaciones, y también para el almacenamiento de datos en Firebase.

3.2.7. jQuery y Ajax

Son librerías de JavaScript de código abierto que permiten realizar peticiones HTTP a un sitio web que necesita respuesta del servidor sin recargar el sitio, consiguiendo crear sitios web dinámicos.

3.2.8. Google Maps JavaScript API.

Son librerías de Google Maps que permiten incorporar mapas en los sitios web. El API está formado por archivos HTML, CSS y JavaScript que contienen

las clases, métodos y propiedades que se usan para controlar el comportamiento de los mapas. Los mapas son solo imágenes que se cargan a través de peticiones ejecutadas con Ajax y se insertan en una página HTML [44]. Mientras se navega en el mapa, el API envía información acerca de las nuevas coordenadas y los niveles de "zoom" del mapa a través de Ajax, y esto provoca la actualización de las imágenes.

En nuestro proyecto se utiliza esta API para dibujar un polígono, circulo o cuadrado en el mapa (ver **Figura 6**), o para agregar un mapa de calor (ver **Figura 7**).

Figura 6. Tipos de Área a seleccionar.

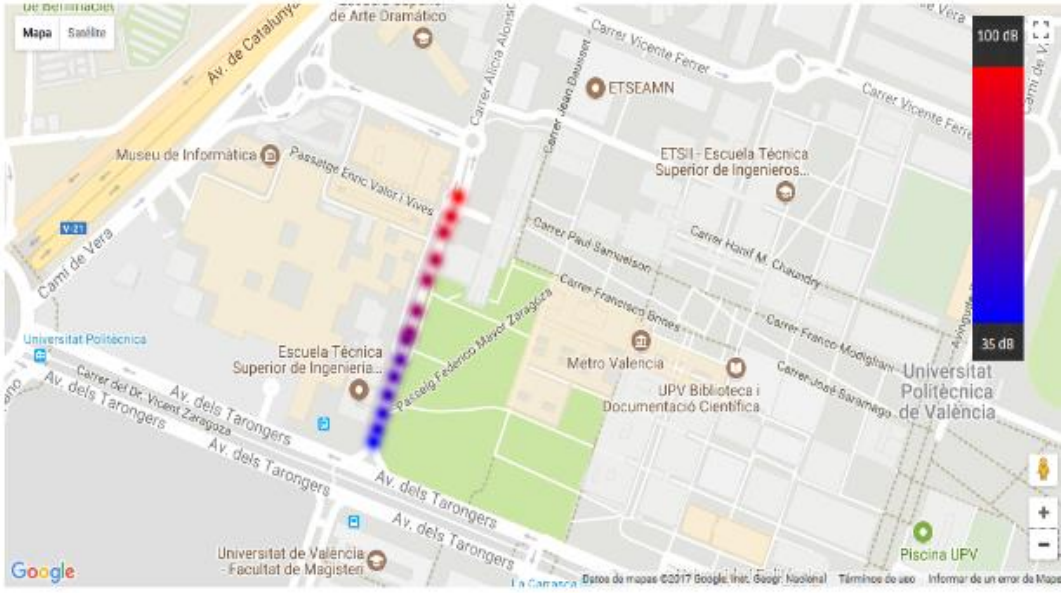

Figura 7. Mapa de Calor.

3.3. Tecnologías para la comunicación entre aplicaciones

Como alternativas a la comunicación entre las aplicaciones, las cuales permiten la distribución de los mensajes hacia los usuarios, se han analizado a nivel teórico dos opciones, Firebase y RabbitMQ, las cuales se describen a continuación.

3.3.1. Firebase

Firebase era un startup que ofrecía el servicio de base de datos en tiempo real, permitiendo a los desarrolladores almacenar y sincronizar los datos de múltiples clientes a través una API [45]. Fue adquirida por Google en octubre de 2014, el cual modificó esta aplicación añadiendo un conjunto de 15 herramientas divididas en 3 áreas (Develop, Grow, Earn), como se muestra en la **Figura 8**.

Para este trabajo se usó Realtime Database que se describirá a continuación con mayor detalle.

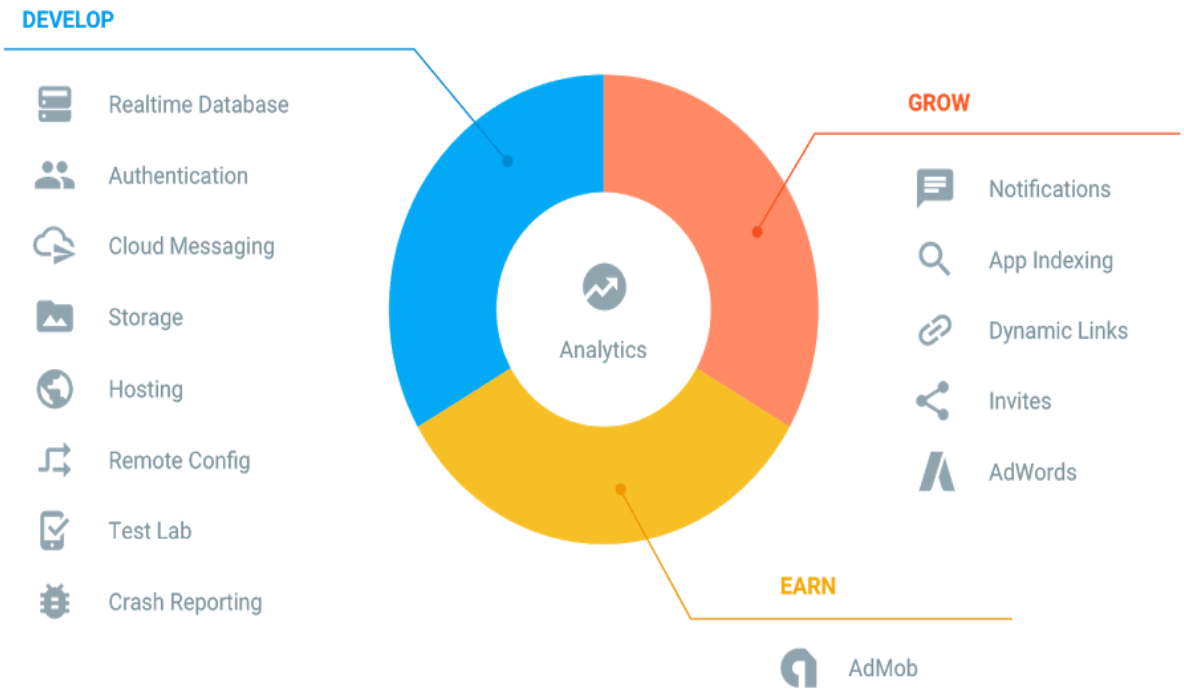

Figura 8. Conjunto de Herramientas de Firebase.

3.3.1.1. Arquitectura de Firebase

En Firebase existen tres posibles arquitecturas de aplicación, las cuales se describen a continuación.

- **100% Firebase**

Esta arquitectura se compone de aplicaciones desarrolladas con contenidos estáticos, y todo su contenido dinámico y datos de usuario se almacenan y recuperan desde Firebase [46], como se muestra en la **Figura 9**. Esta arquitectura es ideal cuando:

- La aplicación necesita una integración mínima con sistemas heredados o servicios de terceros
- La aplicación no realiza procesamiento de datos ni necesita requisitos complejos de autenticación de usuario

Figura 9. Arquitectura 1: Firebase 100%.

- **Firebase con código de servidor**

En esta arquitectura como se muestra en la **Figura 10**, Firebase se encuentra entre el servidor y los clientes. Los servidores pueden conectarse a Firebase e interactuar con los datos como cualquier otro cliente, es decir, nuestro servidor se comunica con los dispositivos móviles mediante la manipulación de datos en Firebase. Además, puede escuchar los cambios realizados en los datos y responder adecuadamente [46]. En esta arquitectura, aunque todavía se está ejecutando un servidor, Firebase está manejando todo el levantamiento a gran escala y realiza actualizaciones en tiempo real.

En nuestro proyecto se utiliza esta configuración, debido a que se hace uso de un servidor para el procesamiento de funciones complejas, y se utiliza Firebase para la comunicación con los dispositivos móviles, mediante la API REST de Firebase, la cual se describe con detalle en el apartado 3.3.1.3.

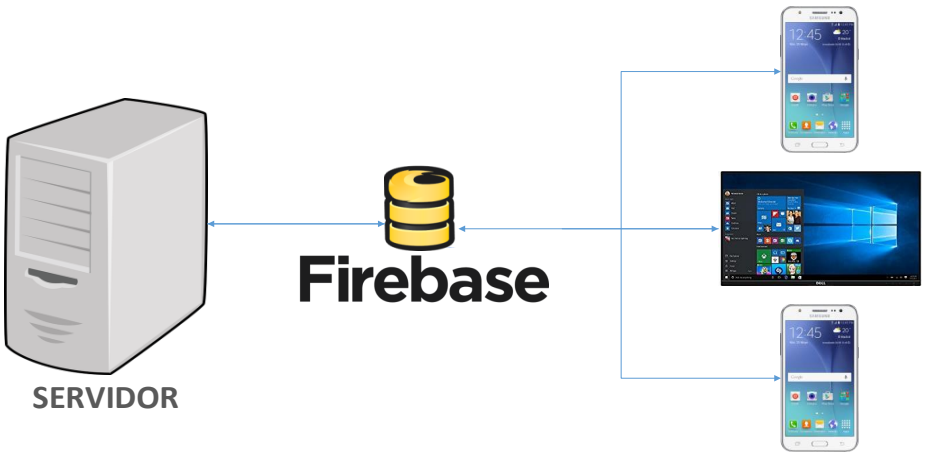

Figura 10. Arquitectura 2: Firebase con código del lado del Servidor.

- **Aplicaciones existentes con funciones Firebase**

En esta arquitectura, Firebase se encuentra junto a un servidor ya existente como se muestra en la **Figura 11**. Los clientes se conectarán tanto a un servidor como a Firebase y utilizarán Firebase para alimentar a las aplicaciones con características de tiempo real, sin interferir con el resto de la aplicación [46]. Por ejemplo, se puede agregar un sistema de notificación en tiempo real o incrustar un sistema de chat, entre otros.

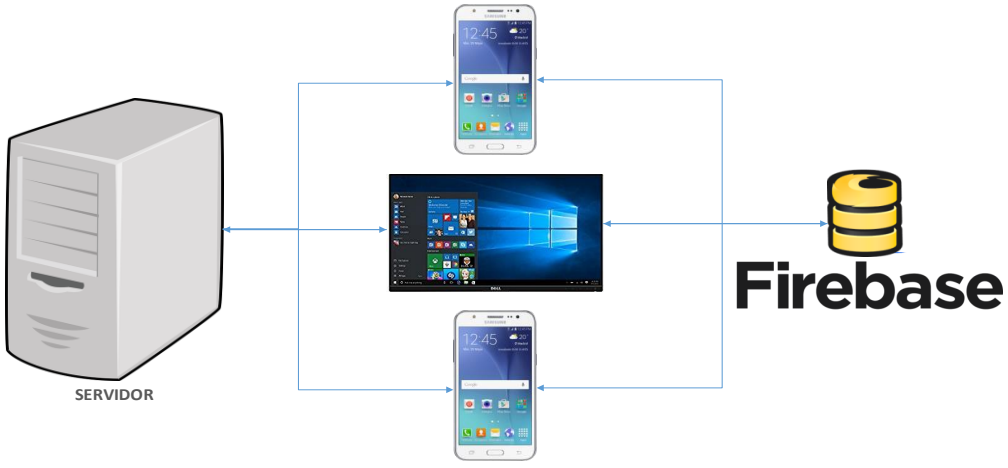

Figura 11. Arquitectura 3: Aplicaciones existentes con funciones Firebase.

3.3.1.2. Realtime Database

Firebase Realtime Database es una base de datos NoSQL alojada en la nube. Por lo tanto, el diseño del sistema gira en torno a objetos JSON. Para nuestro proyecto, estos objetos incluyen información adicional sobre el nombre de la tarea, día de inicio, día de fin, hora de inicio, hora de fin, el área geográfica, y los valores de ruido.

Realtime Database [47], es una tecnología que permite la sincronización de los datos con todos los clientes en tiempo real, y la persistencia de los datos de forma local, incluso cuando no hay conexión, es decir, los eventos en tiempo real se siguen activando y, cuando el dispositivo vuelve a conectarse, Realtime Database sincroniza los cambios de los datos locales con las actualizaciones remotas que ocurrieron mientras el dispositivo estuvo sin conexión.

En Firebase Realtime Database las conexiones se realizan mediante sockets. Cuando un cliente abre una página donde se usa Firebase se mantiene abierto un canal de comunicaciones bidireccional con el servidor, lo cual permite la comunicación desde el cliente al servidor, y desde el servidor a todos los clientes. También cuenta con una serie de librerías mediante las cuales podemos conectarnos y mantenernos suscritos a los cambios de los datos, compatibles con los sistemas más comunes como son iOS, Android y Web, pero también a varios lenguajes de programación del lado del servidor como podrían ser Python o PHP, por medio de una interfaz "REST”.

3.3.1.3. API Rest Firebase

Firebase define a partir de http cuatro métodos para escribir datos, y un método para leer datos de la base de datos. A continuación se describen cada uno de estos métodos [48].

- **PUT:** Es la operación de escritura básica de la API Rest de Firebase, la cual permite escribir o reemplazar datos de una ruta determinada.
- **PATCH:** Esta operación permite actualizar elementos secundarios específicos en una ubicación del árbol JSON, sin sobrescribir los datos existentes.

- **POST:** Esta operación permite agregar una lista de datos en la base de datos de Firebase. Cada vez que se envía una solicitud de POST se genera una clave única.
- **DELETE:** Esta operación permite eliminar una lista de datos de la base de datos de Firebase.
- **GET:** Esta operación permite leer datos de la base de datos de Firebase.

3.3.2. RabbitMQ

Es un software de mensaje de colas denominado intermediario de mensajes o gestor de colas [49]. Es un software (o intermediario) de alto rendimiento que implementa el protocolo AMQP, permitiendo que las colas puedan ser definidas y las aplicaciones puedan conectarse y transferir un mensaje en la cola.

El Protocolo Avanzado de Mensaje de Cola (Advanced Message Quening Protocol), es un protocolo de estándar abierto de mensajería en la capa de aplicaciones de un sistema de comunicación [38]. Las características que definen a este protocolo son el encolamiento, la orientación a mensajes, enrutamiento (sea punto a punto o publicación/subscripción), exactitud y seguridad [50].

Para finalizar RabbitMQ puede ser administrado por servidores en la nube mediante un servicio llamado CloudAMQP, permitiendo organizar colas de mensajes [51] .

3.3.2.1. Flujo de mensajes en RabbitMQ

El flujo básico de una cola de mensajes es simple, hay una aplicación denominada productor que crea el mensaje y los entrega al intercambiador (la cola de mensajes). Otra aplicación denominada consumidor que se conecta a la cola y se subscribe a los mensajes para procesarlos. Una aplicación puede ser ambas, cosas productor y consumidor de mensajes. Los mensajes que se envían a la cola se almacenan hasta que los recupere un consumidor.

3.3.2.2. Componentes Principales

Aquí se describen algunos componentes importantes para el trabajo del protocolo AMQP.

- **Producer:** Aplicación que envía el mensaje al exchange.
- **Consumer:** Aplicación que recibe el mensaje.
- **Message:** Información que se envía del productor al consumidor a través de RabbitMQ.
- **Queues:** Es un buffer que almacena de forma temporal el mensaje que publica el producer.
- **Bindings:** Es el responsable de establecer un enlace entre la cola y el exchange (intercambiador). En este caso, tenemos un solo enlace a una cola del exchange.
- **Exchange:** Recibe el mensaje y es el responsable de enrutar los mensajes a las diferentes colas.

    - **Tipos de Exchange**

      Existen cuatro diferentes tipos de exchanges, como se muestra en la **Figura 12**, que permiten enrutar de modo diferente el mensaje, usando parámetros y configuraciones de enlaces diferentes.

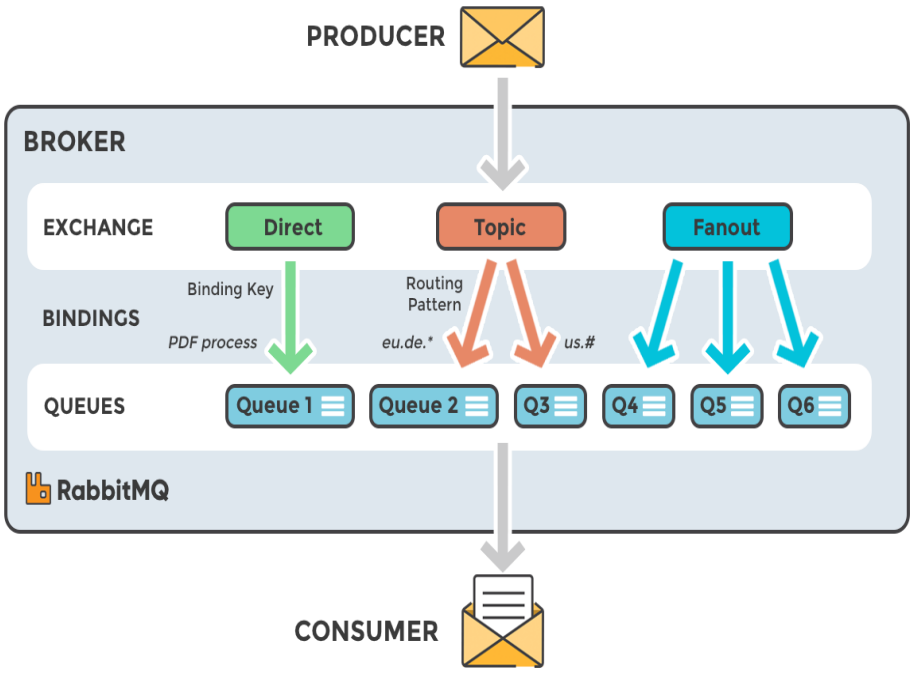

Figura 12. Componentes principales de RabbitMQ.

- **Direct Exchange:** Permite enrutar mensajes desde cero a N colas, siempre que coincida la clave de enrutamiento del mensaje con la clave del enlace de la cola de intercambio.
- **Fanout Exchange:** Permite transferir los mensajes a todas las colas dentro del intercambiador (exchange) independientemente de la clave de encaminamiento que tengan.
- **Topic Exchange:** Permite el enrutamiento de mensajes en función de la coincidencia de patrones entre la clave de enrutamiento de un mensaje y la clave del enlace de la cola de intercambio.
- **Header Exchange:** Permite el enrutamiento basado en expresiones complejas de la cabecera de un mensaje. También es considerado con un intercambiador directo, con la diferencia que el enrutamiento se lleva a cabo en base a la cabecera en lugar de la clave de enrutamiento.

3.3.3. Selección de tecnología

Una vez realizado el análisis de las dos tecnologías en los apartados anteriores, hemos decidido usar Firebase Realtime Database por las siguientes conclusiones:

1. Firebase Realtime Database, al ser una base de datos en tiempo real, proporciona un lenguaje de reglas basadas en expresiones, permitiendo definir en qué momento se pueden leer o escribir los datos. Además, integrar Firebase Authentication permite que los programadores definan quién tiene acceso a qué datos, y cómo acceden a ellos. RabbitMQ no cuenta con esta característica de protección de datos.
2. Tanto Firebase Realtime Database como RabbitMQ ofrecen persistencia de la información. En detalle, Firebase Realtime Database ofrece este servicio de forma automática sin configuraciones, y en cambio en RabbitMQ este servicio no está habilitado de forma predeterminada.
3. En Firebase Realtime Database la sincronización se realiza cada vez que los datos cambian, y así todos los dispositivos conectados reciben

esa actualización, proporcionando una experiencia colaborativa. RabbitMQ no cuenta con esta característica.

Para finalizar, en la **Tabla 2** se muestras un resumen las características que ofrecen cada una de las tecnologías estudiadas.

Tabla 2. Características de las Tecnologías de comunicación analizadas.

	Persistencia en disco	Cloud Server	Multiplataforma	Sincronización en tiempo real	Protección de datos
Firebase	■	■	■	■	■
RabbitMQ	■	■	■	□	□

4. Arquitectura Propuesta

4.1. Descripción del entorno

Para la elaboración de nuestro sistema de monitorización mediante crowdsensing, es necesario contemplar el escenario donde se llevarán a cabo las comunicaciones. Como se puede ver en la **Figura 13**, tenemos el servidor web, el cual envía información de la tarea definida por el administrador y recibe los datos de contaminación acústica; por otro lado tenemos a los clientes que están recibiendo la información de tarea, y a su vez enviando al servidor los datos del ruido ambiental recolectados.

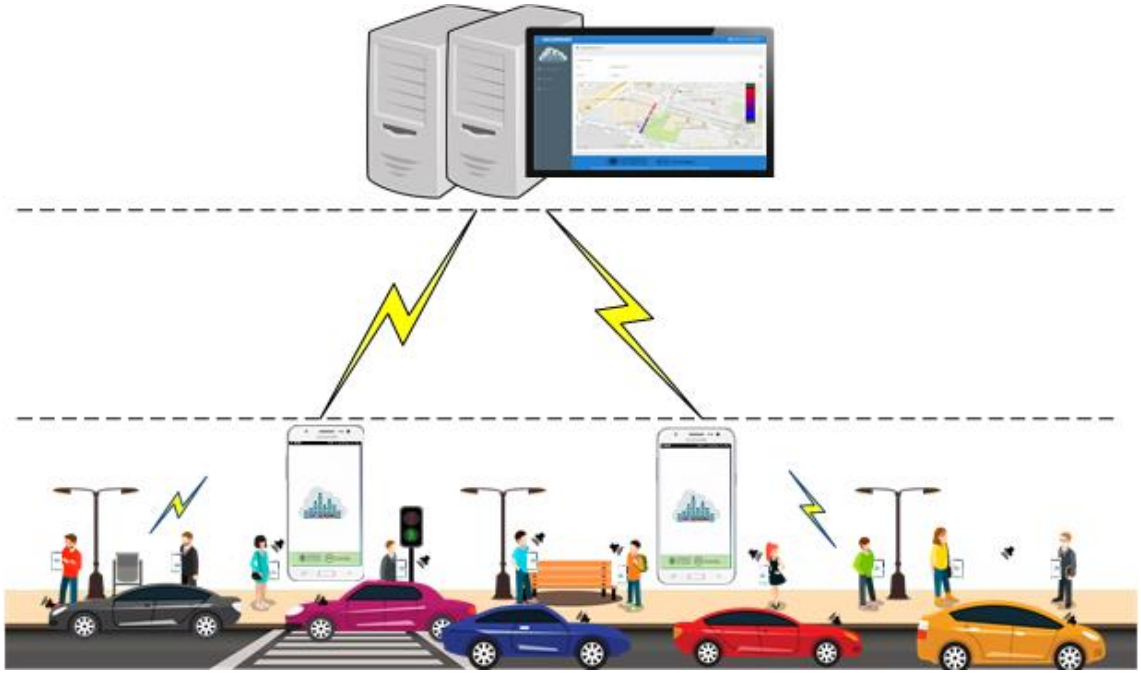

Figura 13. Escenario General.

Para poder realizar la comunicación entre el servidor web y el dispositivo móvil se ha utilizado la base de datos en tiempo real de Firebase.

4.2. Descripción del Sistema

En la **Figura 14** se muestra la arquitectura propuesta del sistema de monitorización de ruido ambiental denominado "GRCSensing", y que permite a los usuarios explorar un área de la ciudad mientras que colectivamente recopila los niveles de ruido urbano en tiempo real.

Figura 14. Esquema general de tecnologías utilizadas para la elaboración del sistema de monitorización.

La arquitectura propuesta está compuesta por tres componentes principales: el cliente o dispositivo móvil, el servidor web, y el intermediario. A continuación se describe con más detalle cada uno de esos componentes.

4.2.1. Cliente Móvil

El objetivo del cliente móvil es recolectar lo niveles de ruido ambiental a través de una aplicación desarrollada en Android Studio. La aplicación primero realizará una petición a Firebase para recibir el fichero JSON con la información de la tarea. Luego la aplicación procede a deserializar el fichero JSON, y almacena la información en su base de datos local SQLite. Llegados a este punto, la aplicación procede a realizar las recolecciones del nivel de ruido ambiental, y enviará a Firebase los datos obtenidos para que éste, a su vez, los reenvíe al servidor web.

4.2.2. Servidor Web

Como se ha podido conocer en el capítulo 3, a través de tecnologías como Django, JavaScript, CSS, Bootstrap y MySQL, entre otras, es posible elaborar y enriquecer nuestro sistema de monitorización con contenido agradable a la vista.

En el esquema cliente-servidor observado en la **Figura 14** se aprecia que el servidor web se encarga de la definición o creación de tareas con diferentes requisitos:

- **Nombre de Tarea:** nombre asignado para cada tarea.
- **Fechas de Inicio/Fin:** día en que se desea que empiece y finalice la toma de muestras. Por ejemplo, se quiere que las muestras se tomen del 01/06/2017 al 02/06/2017.
- **Horas de Inicio/Fin:** hora en la que se desea que empiece y finalice la toma de muestras. Por ejemplo, se quiere que las muestras se tomen desde las 18:00 hasta las 20:00.
- **Tiempo mínimo entre muestras:** tiempo que necesita el dispositivo móvil para recoger los datos de crowdsensing entre muestras.
- **Tiempo mínimo entre intentos:** tiempo que necesita el dispositivo móvil para volver a realizar el intento de tomar una muestra. Por ejemplo, cuando el dispositivo móvil no logró establecer todos los parámetros de validación para tomar la muestra, el proceso se detiene, y se espera 30s para iniciar de nuevo el proceso.
- **Área Geográfica:** área en el que se va a recoger los datos de crowdsensing. El sistema permite seleccionar tres tipos de área diferentes: circunferencia, cuadrado y polígono.

Una vez que las tareas hayan sido definidas, éstas se almacenan en la base de datos local y son serializadas en formato JSON. Como se explicó en el apartado 3.2.5, se utiliza el API Pyrebase para la comunicación entre el servidor web y el intermediario Firebase. Esta API abre un canal de comunicación para la publicación o envío de las tareas a la base de datos en tiempo real de Firebase, además de mantener una escucha activa para recibir los datos de ruido ambiental obtenidos por el dispositivo móvil. Esta información se almacena en la base de datos local MySQL.

Además, el servidor también es el encargado de la visualización de los datos en forma de mapas de calor (para ello utiliza el API de Google Maps), es decir, que permite interpretar los datos de contaminación acústica obtenidos por el dispositivo móvil, para así conocer de forma gráfica los niveles de ruido

ambiental. Para ello, el usuario realizará peticiones a Django que, a su vez, realizará peticiones a MySQL para obtener la información solicitada.

4.2.2.1. Estructura de la Base de Datos

El modelo que se ha seguido para el diseño de nuestro modelo entidad-relación se ha basado en las siguientes consideraciones:

- Llevar registro de las tareas creadas.
- Llevar registro de los valores de nivel de ruido recolectados por el cliente.
- Disponer de un registro de cuentas de usuario que permita el acceso al sistema de monitorización GRCSensing.

A través del diagrama entidad-relación de la **Figura 15** es posible contemplar cómo se gestiona el almacenamiento de información en nuestro sistema de monitorización GRCSensing.

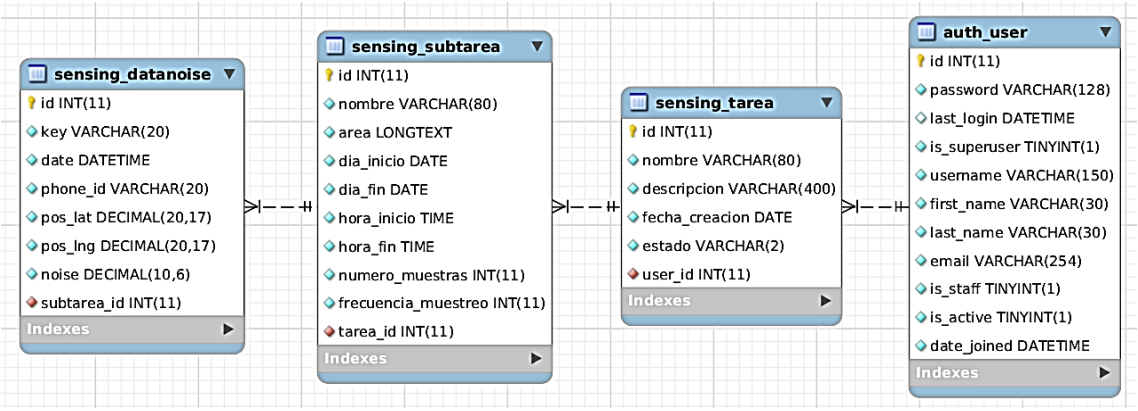

Figura 15. Diagrama Entidad-Relación del sistema de monitorización del ruido ambiental.

4.2.3. Intermediario

El objetivo que cumple el intermediario o la base de datos en tiempo real de Firebase es llevar a cabo las comunicaciones para el envío/recepción de datos provenientes de los dispositivos móviles y el servidor web. Además, almacena temporalmente la información hasta que se asigne a los clientes móviles. Los tamaños del mensaje en función de los diferentes tipos de área se muestran en la **Tabla 3**. Al ser una base de datos NoSQL alojada en la nube, los datos se almacenan en formato JSON, como se muestra en la **Figura 16**.

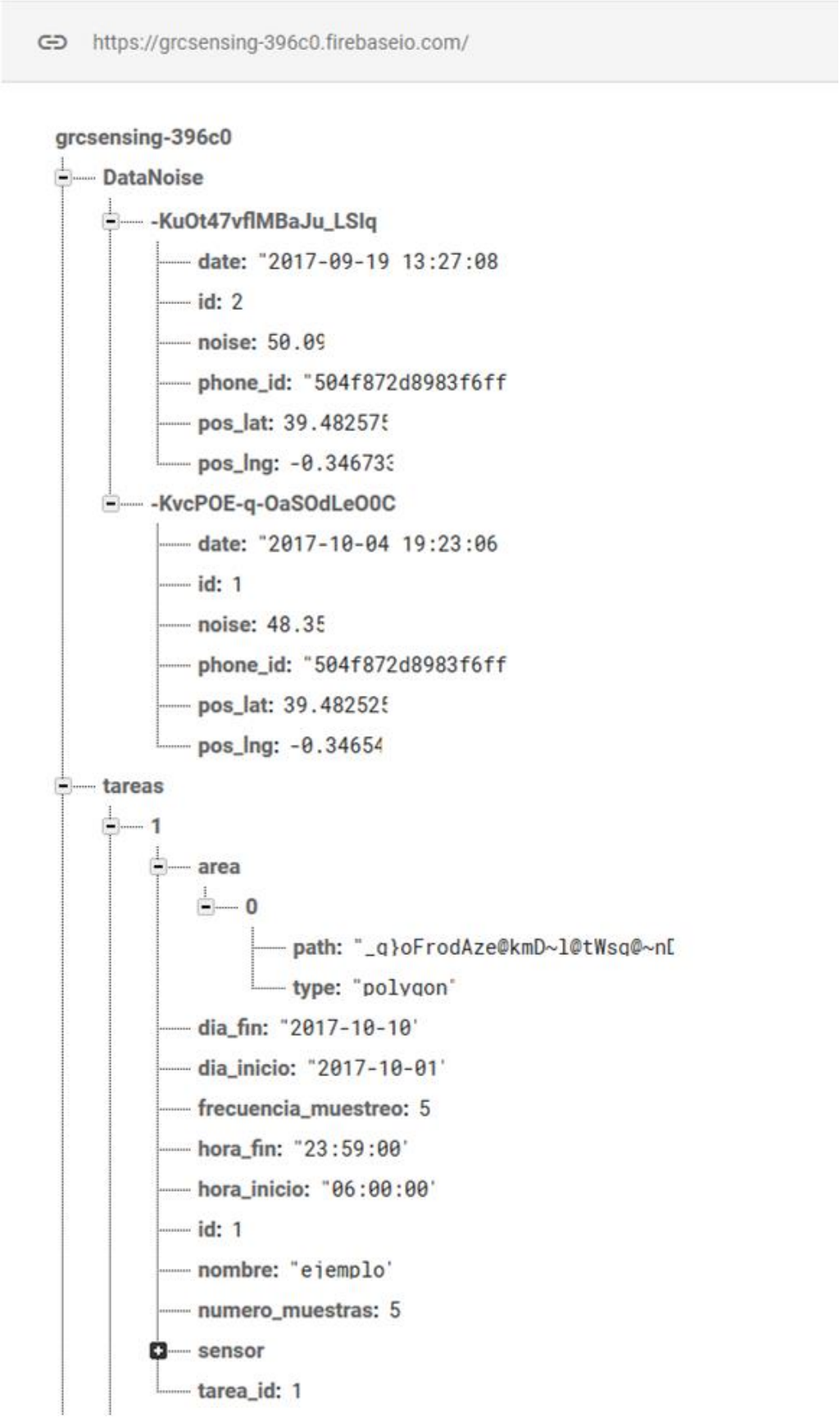

Figura 16. Almacenamiento de Tareas en Firebase en formato JSON.

Tabla 3. Tamaño del mensaje en bytes.

Información	Tamaño (bytes)
Datos de nivel de ruido	236
Tarea con tipo de área: polígono de 5 puntos	363
Tarea con tipo de área: polígono de 10 puntos	404
Tarea con tipo de área: polígono de 20 puntos	411
Tarea con tipo de área: circulo	461
Tarea con tipo de área: rectángulo	509

El esquema de enrutamiento de Firebase permite crear un sistema de crowdsensing en tiempo real altamente escalable y eficiente. Esto quiere decir que Firebase Realtime Database permite la sincronización de los datos con todos los clientes en tiempo real, incluso cuando no hay conexión, es decir, que Firebase gestiona las desconexiones de las comunicaciones móviles (por ejemplo, cambio en las redes WiFi y 3G / 4G), manteniendo los eventos en tiempo real activos. Cuando el dispositivo vuelve a conectarse, se sincronizan los cambios de los datos locales con las actualizaciones remotas que ocurrieron mientras el dispositivo estuvo sin conexión.

5. Diseño del Sistema

En este capítulo se describe la interfaz de los dos componentes del sistema, aplicación móvil y aplicación web, permitiendo distinguir las funcionalidades de cada uno.

5.1. Interfaz del Sistema Web

El sitio web muestra una interfaz simple e intuitiva (ver **Figura 17**). En su página principal se muestra una descripción de la solución planteada. Entre otras opciones del menú se encuentra el enlace de inicio de sesión. Esta página permite acceder a un nuevo menú de opciones destinadas a la administración y creación de las tareas, además de la visualización de los gráficos de contaminación respectivos. Las opciones que se muestran después de iniciar sesión son: Administrador de Tareas, Dashboard, e Información sobre el usuario administrador.

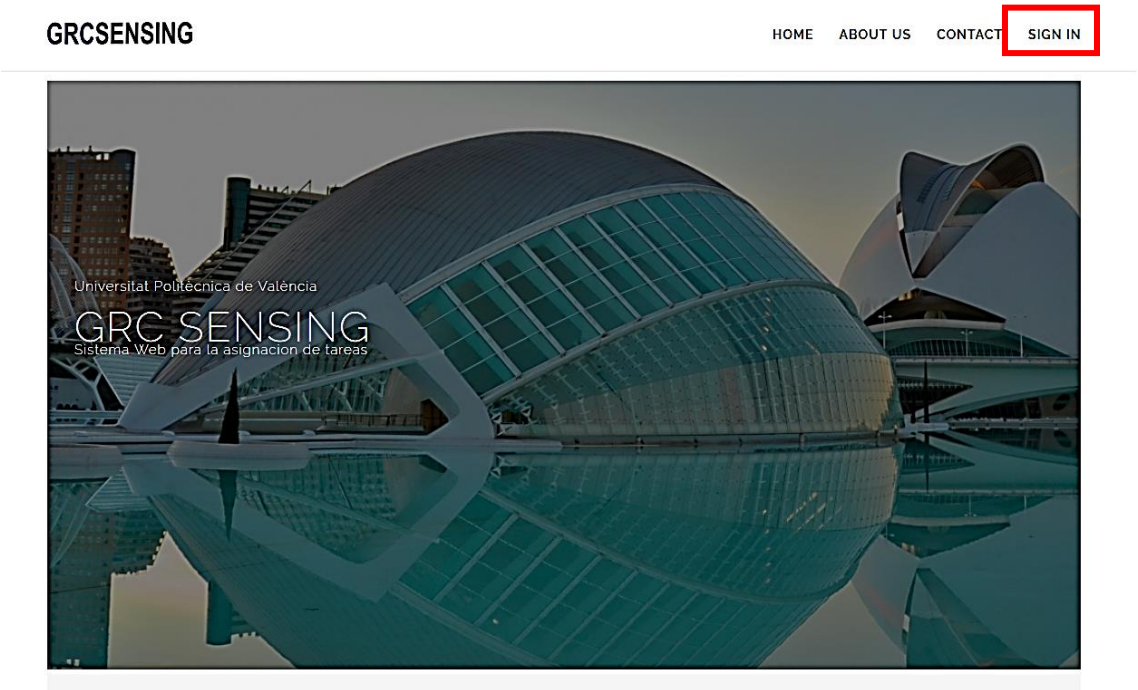

Figura 17. Pantalla principal del sistema web.

5.1.1. Administrador de Tareas

La opción de Administrador de tareas (ver **Figura 18**) permite enlistar todas las tareas, de las cuales se puede visualizar el nombre, descripción y fecha de creación de cada una de las tareas creadas por el administrador. Además,

esta primera vista permite (1) crear nuevas tareas, (2) modificar las tareas creadas, (3) enviar las tareas a los dispositivos móviles y (4) eliminar tareas. Es importante señalar que, una vez enviadas las tareas a los dispositivos móviles, éstas no se podrán eliminar.

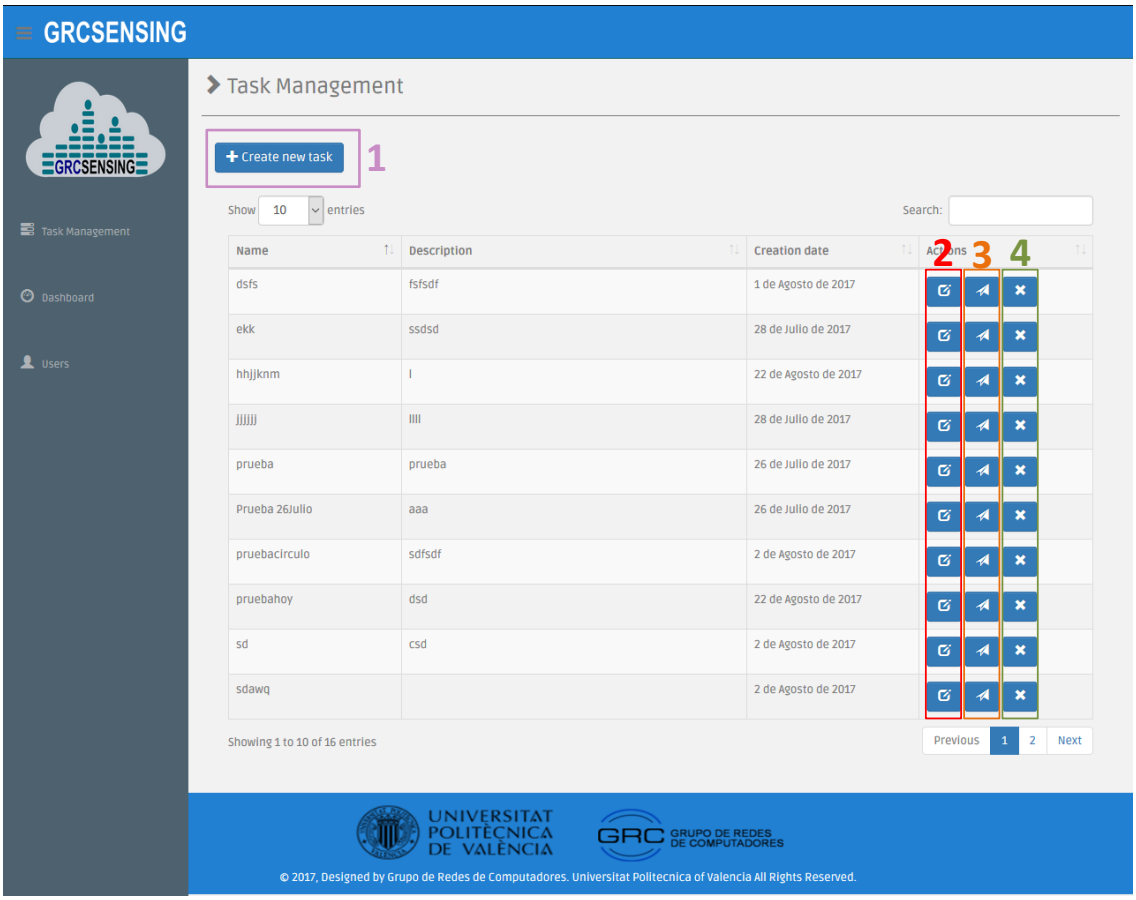

Figura 18. Administrador de tareas del sistema de monitoreo ambiental.

Creación de Nueva Tarea

Al presionar el botón “Create New Task”, se visualiza la opción que permite al usuario administrador la creación de nuevas tareas (ver **Figura 19**). Esta vista está dividida en dos partes.

En primer lugar (1) encontramos el formulario para rellenar la información (nombre, descripción y fecha de creación) relacionada con la tarea, y también encontramos dos botones (1.1) “Save Task”, para guardar la tarea con sus subtareas, y “Return”, para regresar a la vista anterior; es necesario recalcar que al hacer clic en este botón la información rellenada en los formularios se perderá.

En segundo lugar (2) encontramos el formulario en el que se listan las diferentes subtareas de la tarea a crear. Además, también encontramos el botón (2.1) “Create New Subtask” que permite la creación de nuevas subtareas.

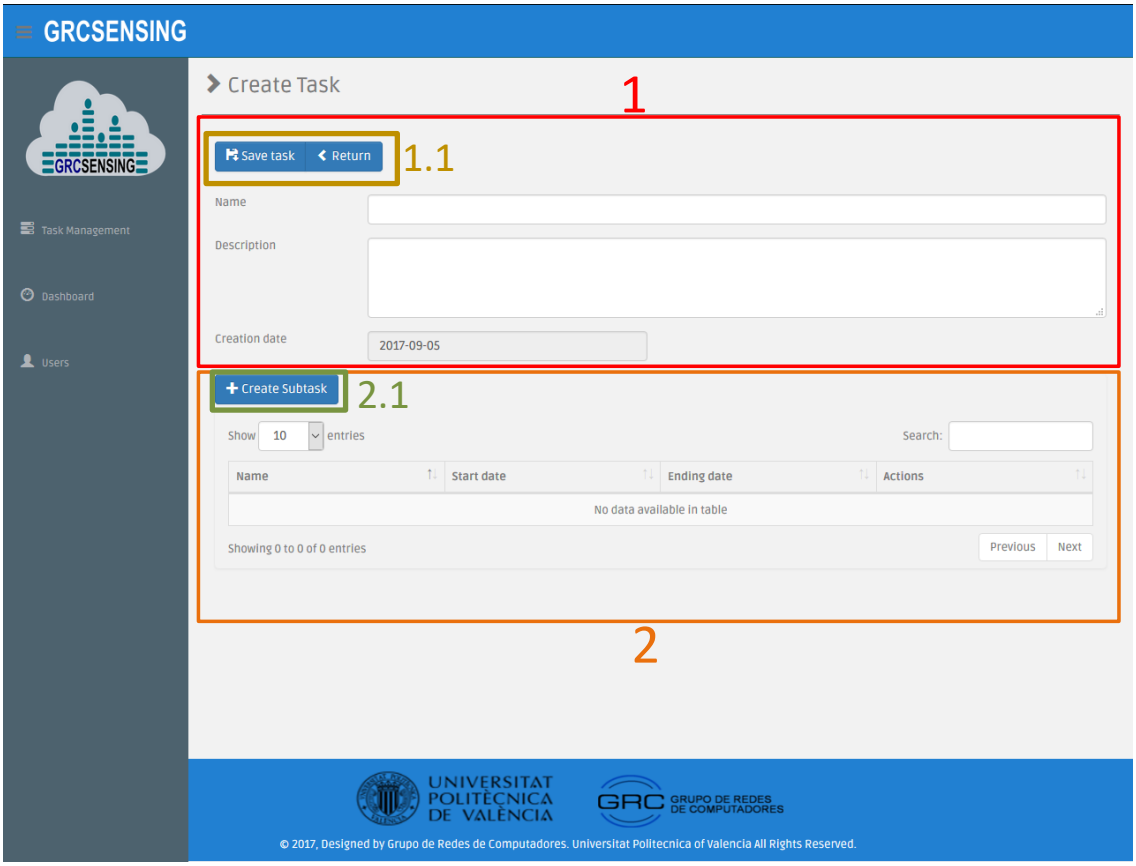

Figura 19. Creación de nueva tarea.

Así mismo, al presionar el botón “Create New Subask” (2.1) se despliega una ventana (ver **Figura 20**) en la que se muestra el formulario a rellenar para la creación de las subtareas. Este formulario está dividido en tres partes:

En primer lugar, visualizamos la información a rellenar de la subtarea que contiene: nombre de la subtarea, fecha de inicio de la subtarea, fecha de finalización de la subtarea, hora de inicio de la subtarea, hora de finalización de la subtarea, el número de muestras, y la frecuencia de muestreo.

En segundo lugar, visualizamos un mapa en el cual se va a seleccionar la ubicación geográfica en la que se va a realizar la recolección de los niveles de ruido. Cabe señalar que existen tres formas (2.1) de seleccionar la ubicación geográfica, como son cuadrado, circulo y polígono.

Para finalizar tenemos dos botones, el primero, “Clear selected area”, permite eliminar el circulo, polígono o cuadrado en caso de que la ubicación geográfica seleccionada no sea la correcta, y el segundo, “Save Subtask”, permite guardar la subtarea; al presionar este botón se regresa a la vista anterior. Cabe destacar que, al regresar a la vista de creación de tareas, ya podemos ver enlistadas cada una de las subtareas que se van creado. Además, también aparecen los botones para eliminar y editar estas subtareas en caso de que se deban realizar modificaciones.

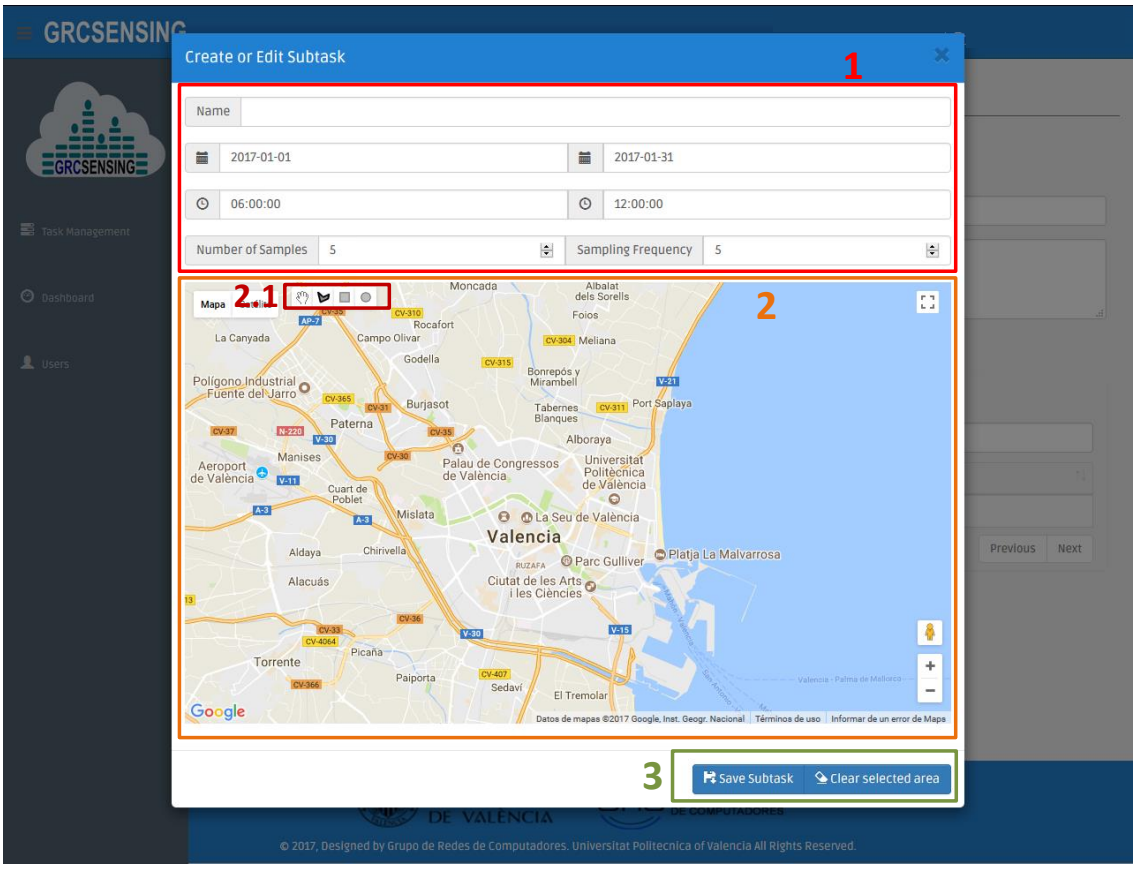

Figura 20. Creación de Subtareas.

Modificación de Tareas

Para realizar la modificación de las tareas partimos de la vista “Task Management” (ver **Figura 18**). Al presionar el botón “Edit Task” se nos presenta una ventana parecida a la de creación de tareas (ver **Figura 21**), en la que podemos realizar modificadores de los campos del formulario mencionado. Además, se pueden realizar modificaciones a las diferentes subtareas que contiene la tarea. Al presionar el botón “Edit Subtask” se abrirá una ventana

parecida a la de crear subtarea (ver **Figura 22**), se procederá a la modificación de los campos relacionados con las subtareas.

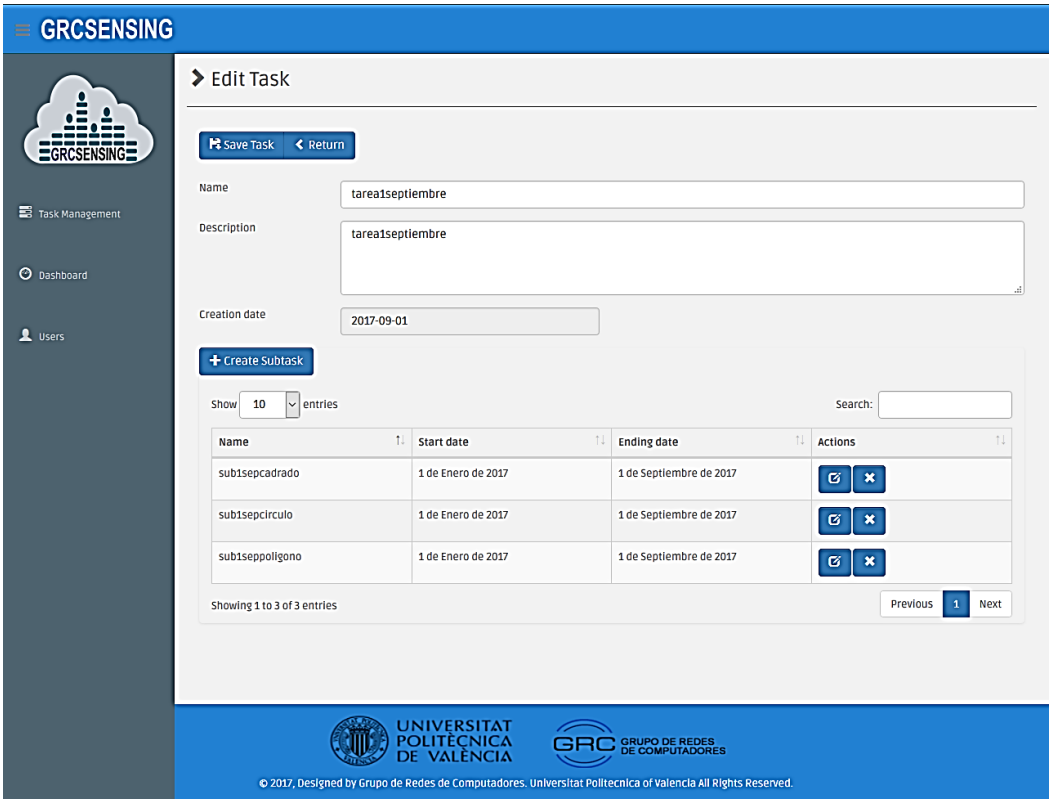

Figura 21. Editar Tareas.

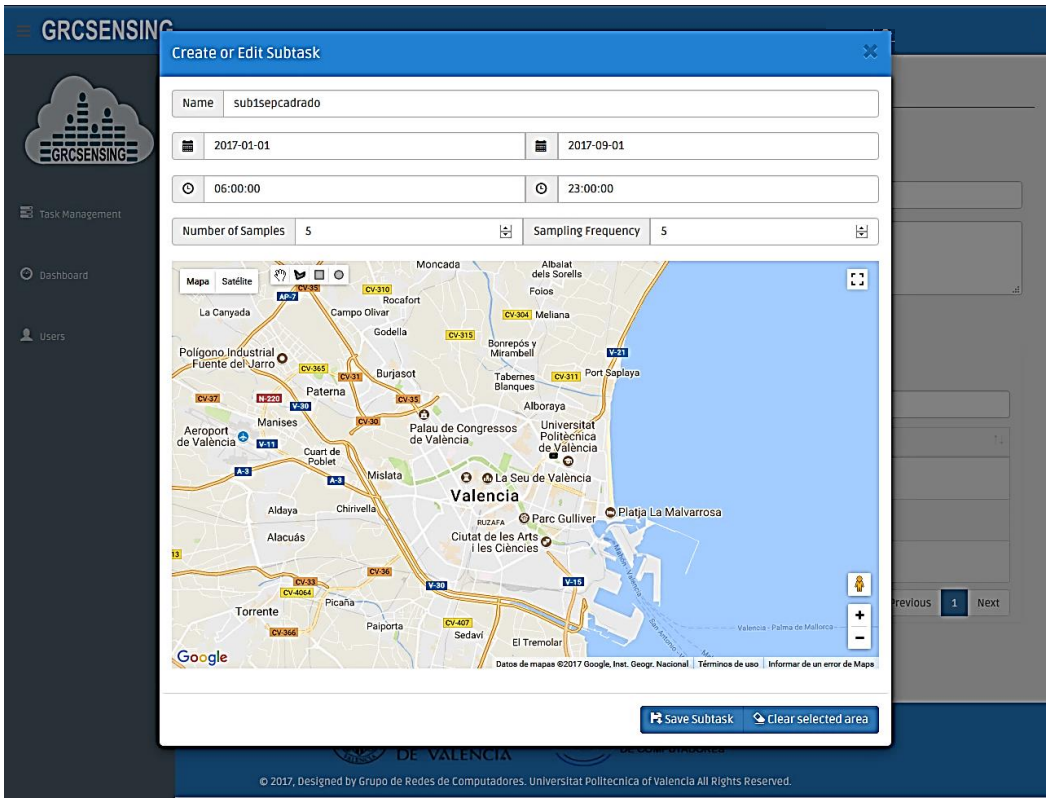

Figura 22. Editar Subtarea.

Envío de tareas a Firebase

Para el envío de tareas a los dispositivos móviles partimos desde la vista “Task Management”. Aquí, como se ha explicado anteriormente, encontramos el botón “Send Task”. Al hacer clic sobre él nos muestra un mensaje como se ve en la **Figura 23**, en el que nos pregunta si deseamos enviar la tarea al dispositivo móvil, y hacemos clic en el botón aceptar para que la tarea se envíe.

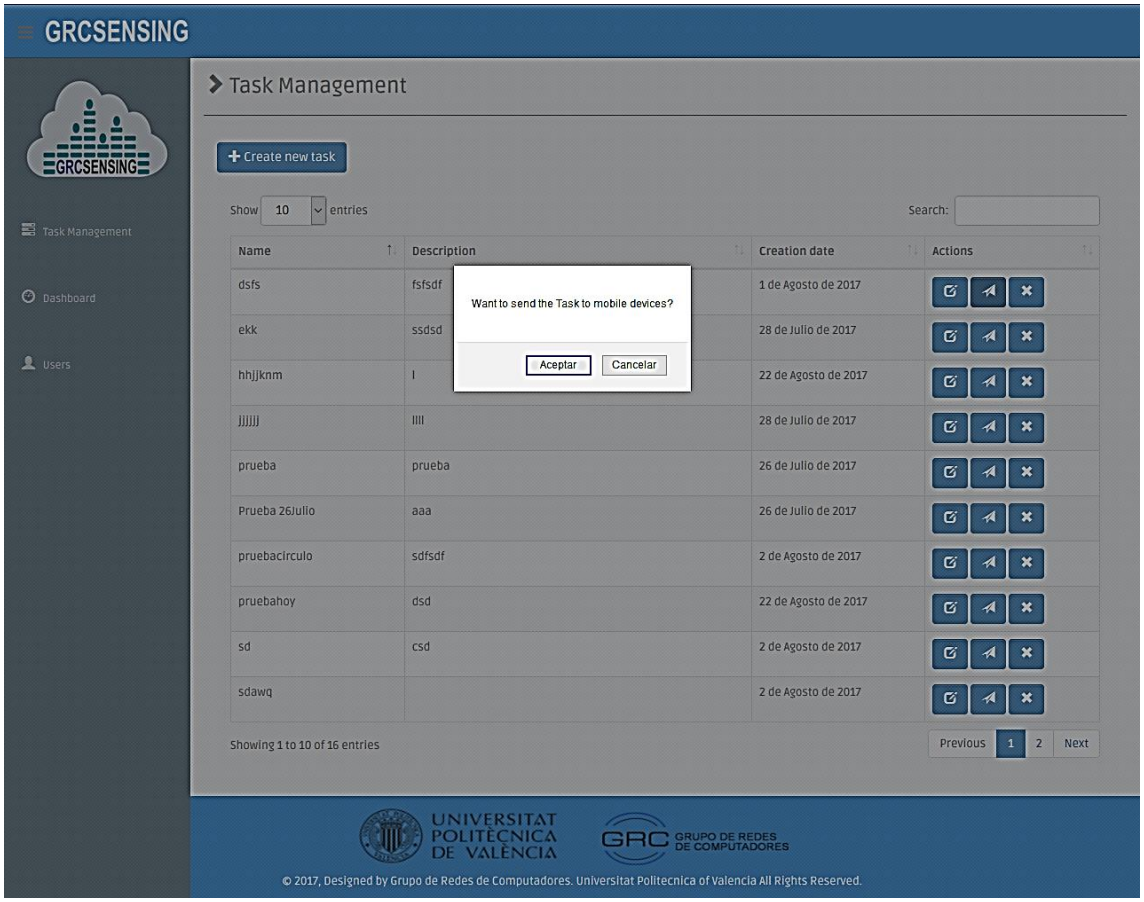

Figura 23. Envío de Tarea al Dispositivo Móvil.

Para verificar que la tarea se ha creado se puede acceder a la consola de administración de nuestro intermediario Firebase y, como se muestra en la **Figura 24**, la información de la tarea se encuentra almacenada en formato JSON. Como se ha dicho en el Capítulo 4, apartado 4.2.3, Firebase almacenará esta información temporalmente y la sincronizará en tiempo real con todos los clientes conectados.

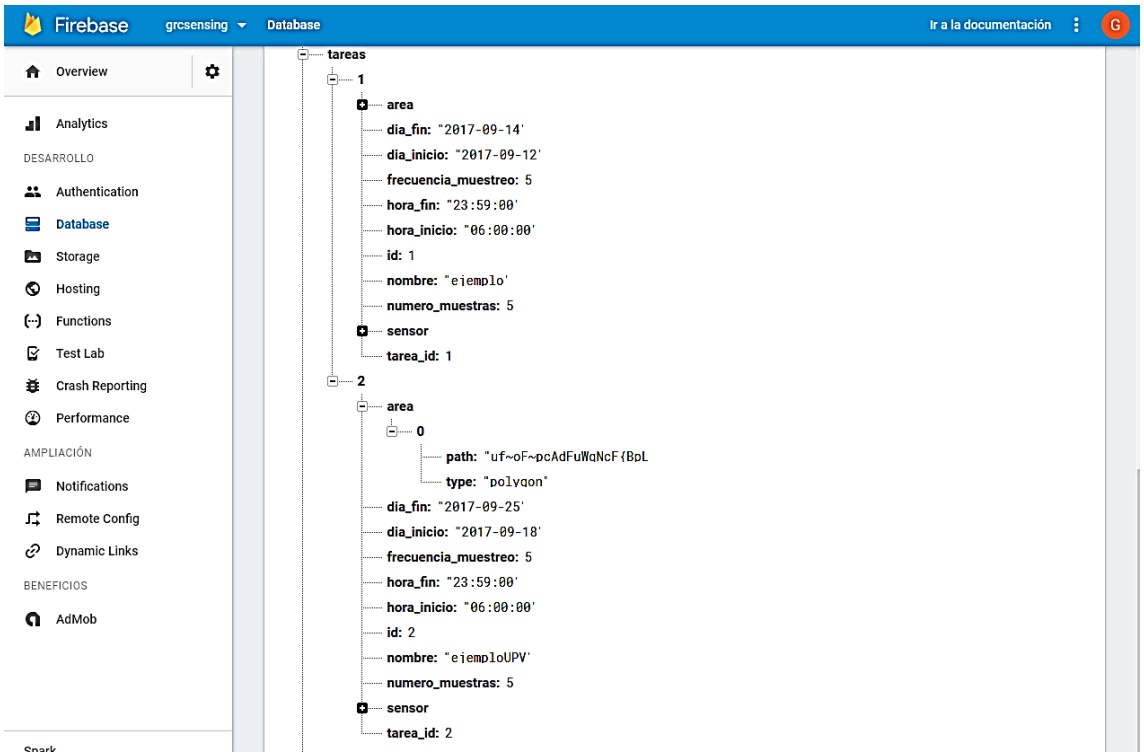

Figura 24. Consola de Administración Firebase verificación de creación de Tareas.

5.1.2. Dashboard

La opción de Dashboard permite la generación de gráficos relativos a una tarea específica. En la **Figura 25** se puede visualizar el proceso a seguir para la creación de esta gráfica.

En primer lugar, permite filtrar por nombre de tarea haciendo clic en el cuadro desplegable (1), enlistando las diferentes tareas que se han creado. Segundo, una vez seleccionada la tarea, se activará el segundo cuadro desplegable (2) que corresponde a los nombres de las subtareas que pertenecen a la tarea antes seleccionada y, para finalizar, se muestra el mapa de calor que se ha generado asociado a la tarea y subtarea seleccionada (3).

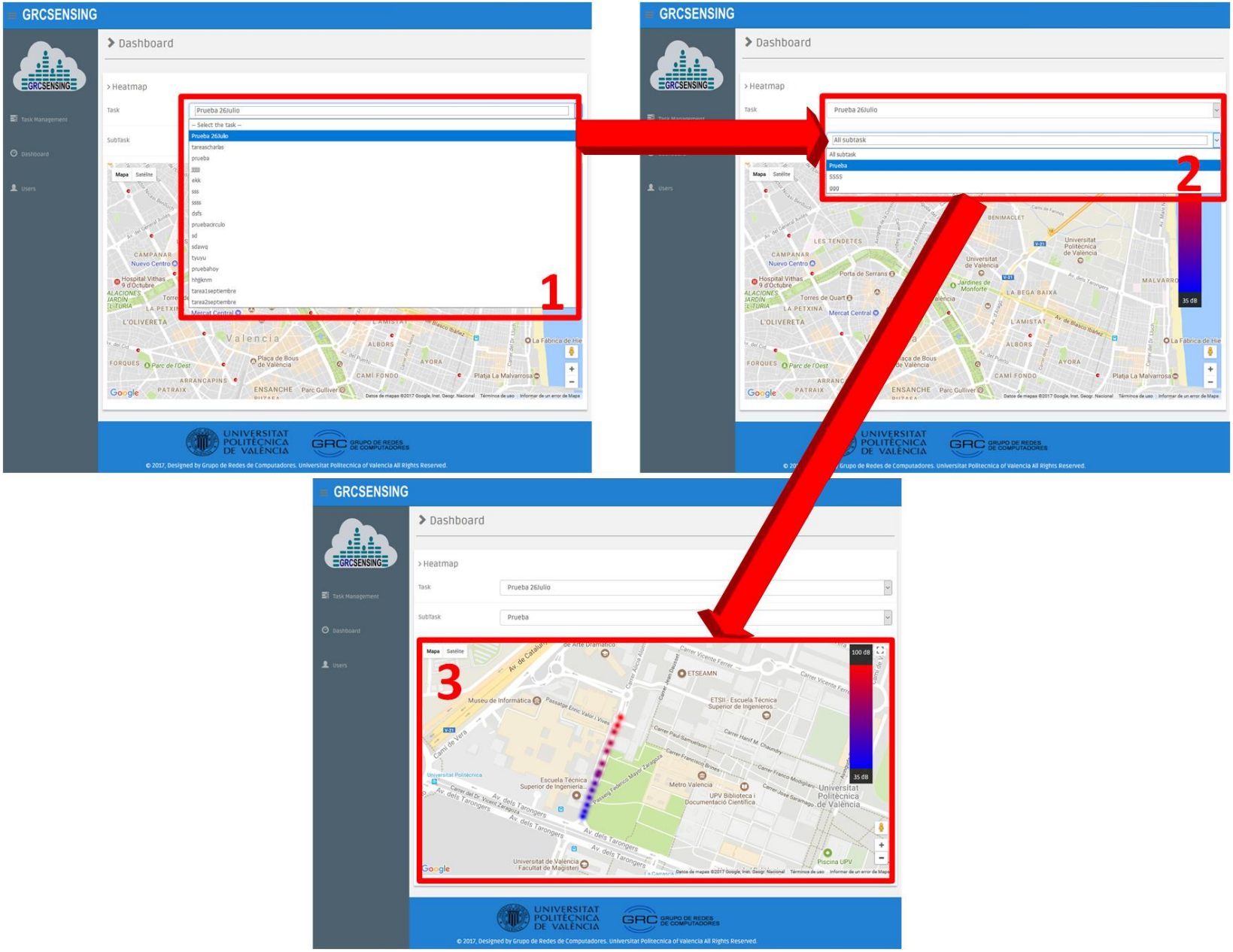

Figura 25. Dashboard: visualización de mapas de calor.

5.1.3. Información sobre el usuario

Esta vista muestra un formulario con información relacionada con el usuario administrador que ha iniciado sesión en el sistema de monitorización ambiental.

5.2. Interfaz de la Aplicación Móvil

La solución permite leer el ruido ambiental con poca intervención del usuario. La interfaz principal de las aplicaciones se muestra en la **Figura 26**, y permite configurar los diferentes permisos asociados con el sistema operativo, como almacenamiento, audio y ubicación. Después de un momento, la pantalla de bienvenida se cierra y el servicio principal se activa.

Figura 26. Pantalla principal cliente móvil.

El proceso de recolección del nivel de ruido ambiental se muestra en la **Figura 27**, y consiste presionar el botón (1) “Start Service”, momento en el cual la aplicación recibe una nueva tarea que es suministrada por el servidor, la cual

se almacena en la base de datos local SQLite. Para evitar redundancia de información, las tareas son previamente consultadas y, si no existen, se almacena.

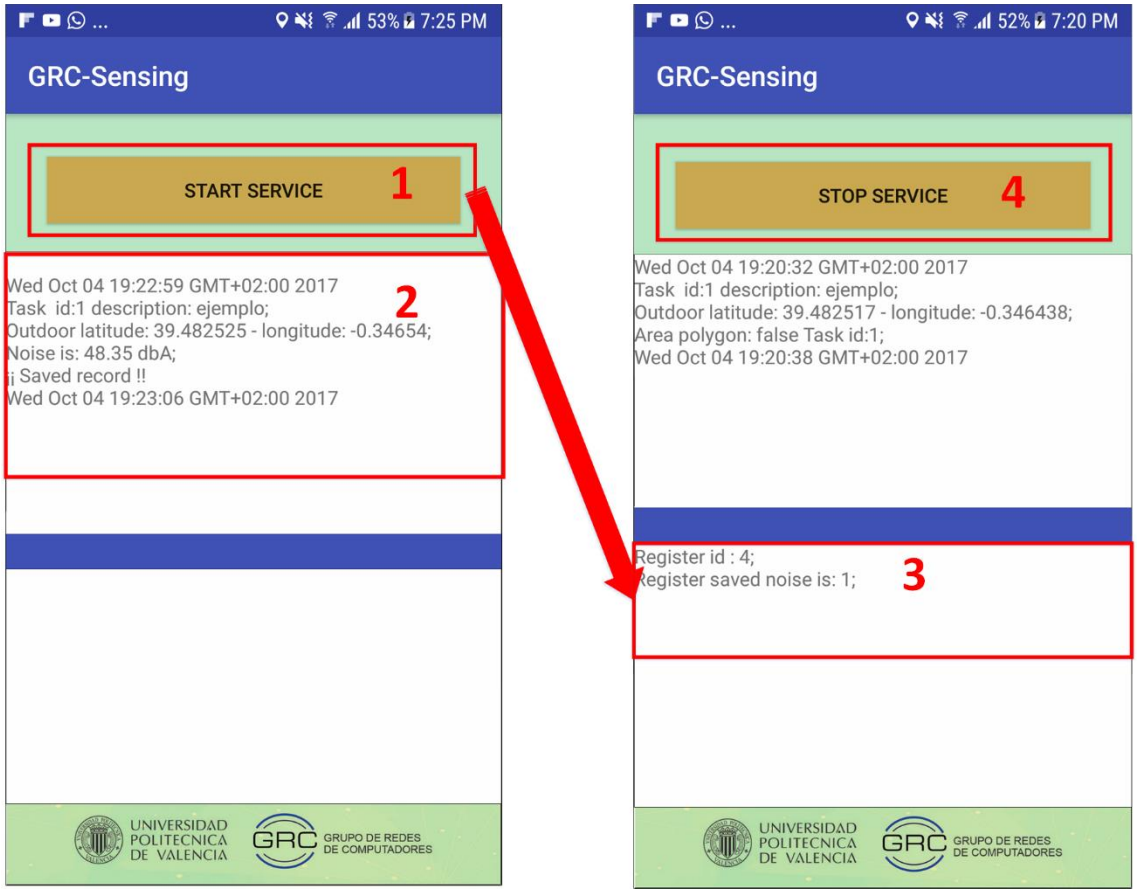

Figura 27. Móvil GRCSensing- Recolección de datos.

Con la información de la tarea recibida se procede a la lectura del ruido ambiental. Los datos de la tarea y los valores del nivel de ruido ambiental que se están leyendo se puede visualizar en (2). Además, en (3) se muestra la cantidad de registros que han sido enviados al servidor. Como se ha explicado en el Capítulo 4, apartado 4.2.3, el envío de los datos de ruido se realiza a través de Firebase, el cual ofrece la opción de persistencia en disco, lo que quiere decir que, si el dispositivo móvil se queda sin conexión de red, Firebase almacena en cache los registros de nivel de ruido y, cuando éste se recupera, la conexión sincroniza los datos almacenados en cache con el servidor. Para finalizar el servicio de recolección de datos se presiona el botón (4) “Stop Service”.

Para verificar que los registros han sido enviados al servidor se puede acceder a la consola de administración de nuestro intermediario Firebase y,

como se muestra en la **Figura 28**, la información del nivel de ruido asociado a la tarea se encuentra almacenada en formato JSON.

Figura 28. Consola de administración Firebase: verificación de registro de nivel de ruido.

6. Validación de la Arquitectura Propuesta

En este capítulo se detallan las pruebas realizadas en diferentes puntos de la ciudad de Valencia, y se presentan los resultados obtenidos. Se eligieron 3 escenarios distintos que corresponden a: (i) el centro comercial “Bonaire”, (ii) el campus de la Universidad Politécnica de Valencia (UPV), y (iii) el Centro de la Ciudad de Valencia. La idea de este primer escenario es realizar una validación para demostrar que nuestra solución se puede utilizar en entornos que se encuentran alejados de la ciudad y para los cuales, en general, no existen datos de nivel de ruido ambiental. La idea de realizar el segundo escenario es demostrar que el valor del ruido ambiental obtenido dentro del campus de la Universidad Politécnica de Valencia puede variar con respecto a los datos de ruido ambiental que son entregados por el ayuntamiento de la ciudad de Valencia. La idea de realizar el tercer escenario es estudiar el nivel de ruido ambiental en un habiente con actividad normal. Para realizar este experimento se escogió el Centro de la Ciudad de Valencia, y se realizará la comparación de los resultados obtenidos por nuestra solución con respecto a los datos de ruido ambiental que son entregados por el sitio web del Ayuntamiento de la Ciudad de Valencia.

6.1. Escenario 1: Centro Comercial “Bonaire”

El primer experimento evalúa nuestra propuesta en el centro comercial Bonaire, el cual, tiene libre circulación peatonal y pasillos descubiertos. Para realizar este experimento se definió una tarea en el sistema web, con las siguientes configuraciones:

- **Dia de Inicio:** sábado 14 de octubre de 2017
- **Dia de Fin:** sábado 14 de octubre de 2017
- **Hora Inicio:** 12:30
- **Hora Fin:** 13:00
- **Tipo de área:** polígono

En la **Figura 29** se muestra el área de cobertura del primer experimento.

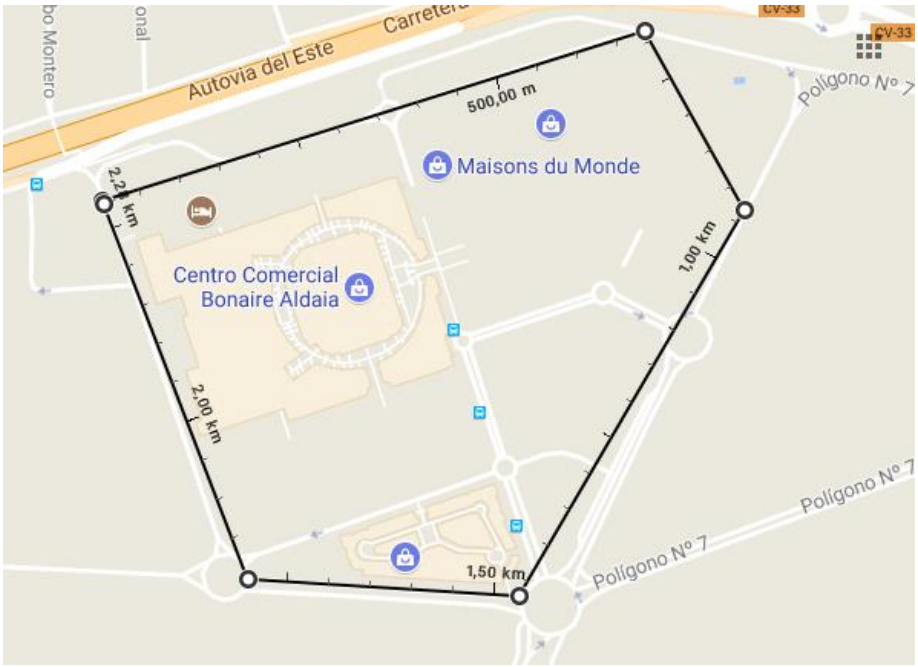

Figura 29. Definición de área para la Tarea 1 usando un polígono.

Con respecto a los clientes móviles, en este primer experimento se utilizaron 4 diferentes dispositivos móviles, donde cada uno de ellos tenía instalada y activada la aplicación GRCSensing. Los recorridos realizados por los dispositivos móviles se hicieron de forma aleatoria y de manera simultánea por las diferentes instalaciones interiores (pasillos) y exteriores del centro comercial, como se muestra en la **Figura 30**.

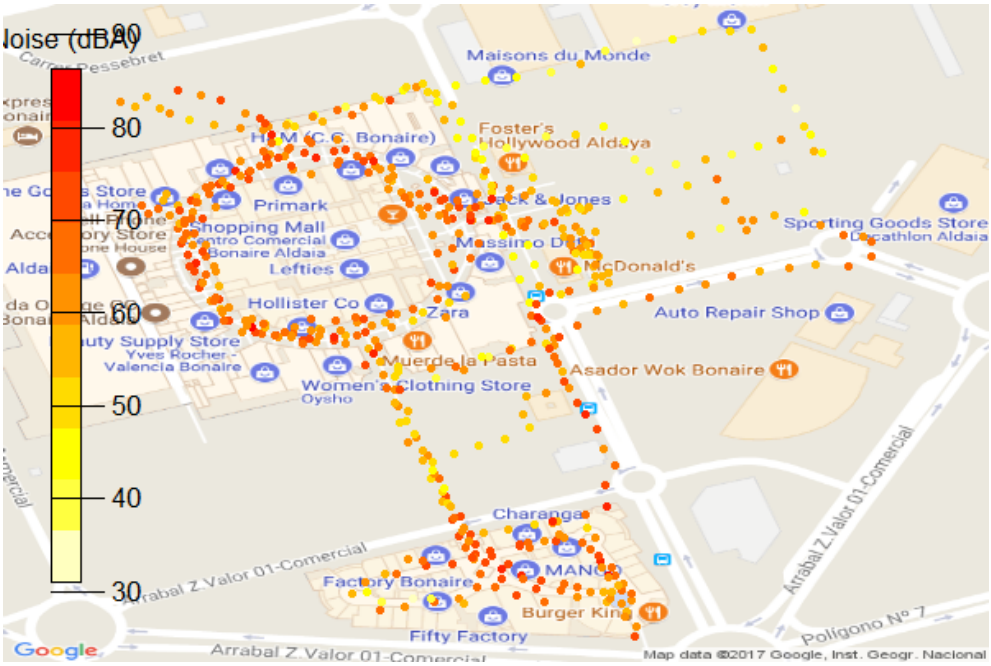

Figura 30. Puntos de recorrido aleatorios de los clientes móviles en el Centro Comercial Bonaire.

En la **Figura 31** se muestran los resultados obtenidos mediante mapas de calor, permitiendo observar que los lugares donde existe un nivel alto de ruido ambiental se encuentran en zonas de aparcamiento de vehículos. Por el contrario, se observa que el interior del centro comercial los niveles de ruido ambiental son bajos.

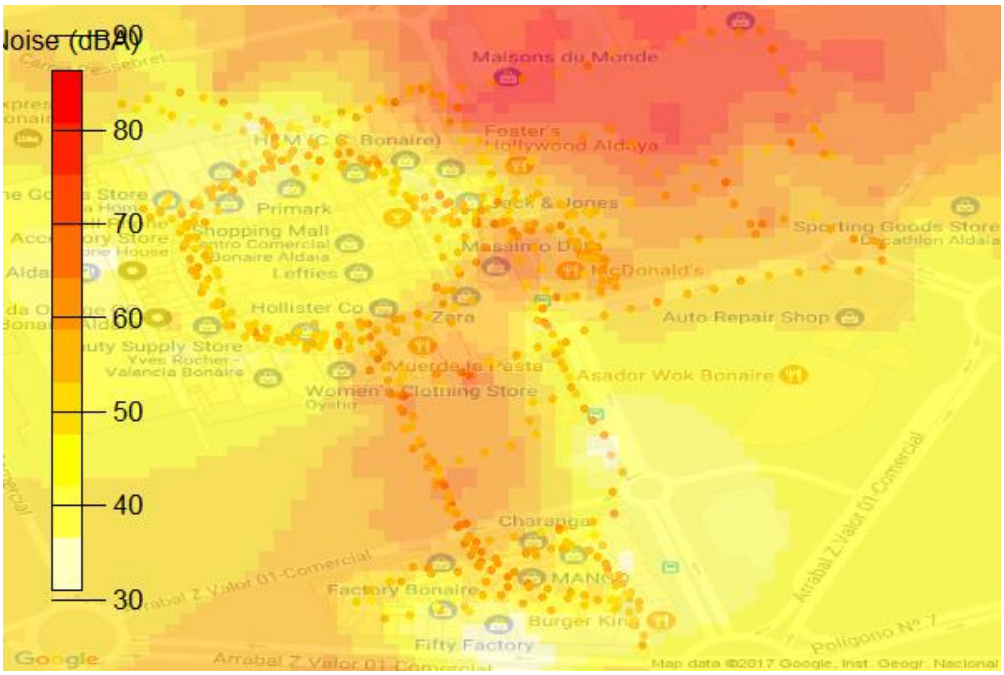

Figura 31. Mapa de calor del Centro Comercial Bonaire.

6.2. Escenario 2: Campus de la UPV

El segundo experimento evalúa nuestra propuesta en el campus de la Universidad Politécnica de Valencia (UPV). El objetivo de realizar esta prueba es comparar los valores de ruido ambiental de nuestra propuesta con el diagnóstico del ruido ambiental que muestra el sitio web del Ayuntamiento de Valencia.

En la **Figura 32** se muestra un mapa de ruido ambiental cercano a la UPV obtenido del sitio web del ayuntamiento de Valencia, en el cual se puede observar que este sitio web muestra en color gris el campus de la UPV, indicando que no existes valores registrados para esa zona.

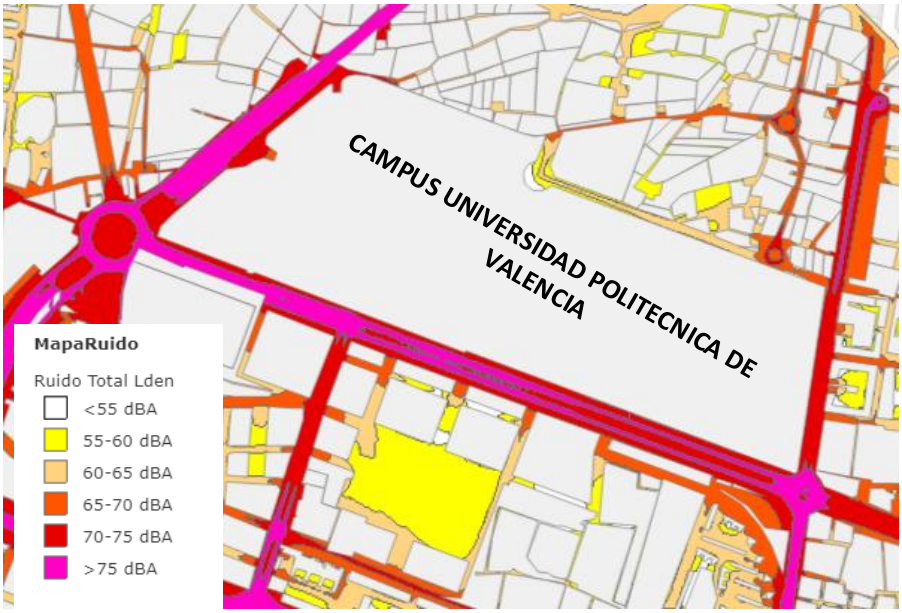

Figura 32. Mapa de Calor del Sitio Web del Ayuntamiento de Valencia.

Así, para la realización de este segundo experimento, se han considerado las horas en la cuales se movilizan muchos estudiantes por el campus de la UPV. De esta manera, la tarea se definió con las siguientes configuraciones:

- **Dia de Inicio:** martes 17 de octubre de 2017
- **Dia de Fin:** martes 17 de octubre de 2017
- **Hora Inicio:** 17:15
- **Hora Fin:** 17:45
- **Tipo de área:** polígono

En la **Figura 33** se muestra el área de cobertura del segundo experimento.

Figura 33. Definición de área para la Tarea 2 usando un polígono.

Con respecto a los clientes móviles, en este segundo experimento se utilizaron 2 dispositivos móviles, donde cada uno de ellos tenían instalada y activada la aplicación GRCSensing. Los recorridos realizados por los dispositivos móviles se seleccionaron de forma aleatoria, y se realizaron en simultaneo por el campus de la UPV, como se muestra en la **Figura 34**.

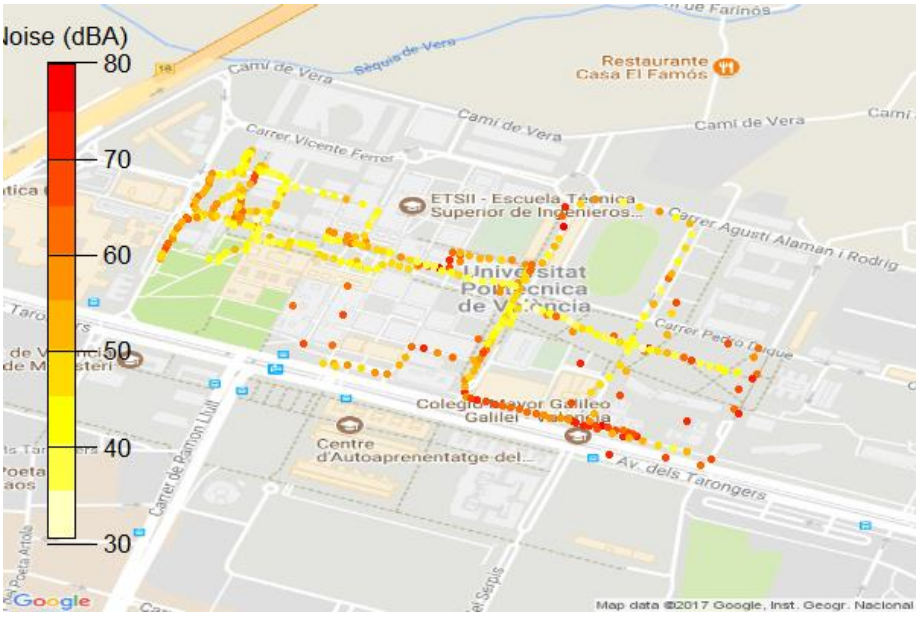

Figura 34. Puntos de recorrido por los clientes móviles en el campus de la UPV.

En la **Figura 35** se muestra los resultados obtenidos mediante mapas de calor, permitiendo observar que los lugares donde existe un nivel alto de ruido ambiental se encuentran en zonas cercanas a las cafeterías y restaurantes. En general, si comparamos los resultados obtenidos con los valores del sitio web que se muestra en la **Figura 32**, se percibe una diferencia significativa con respecto a los valores obtenidos por nuestra propuesta.

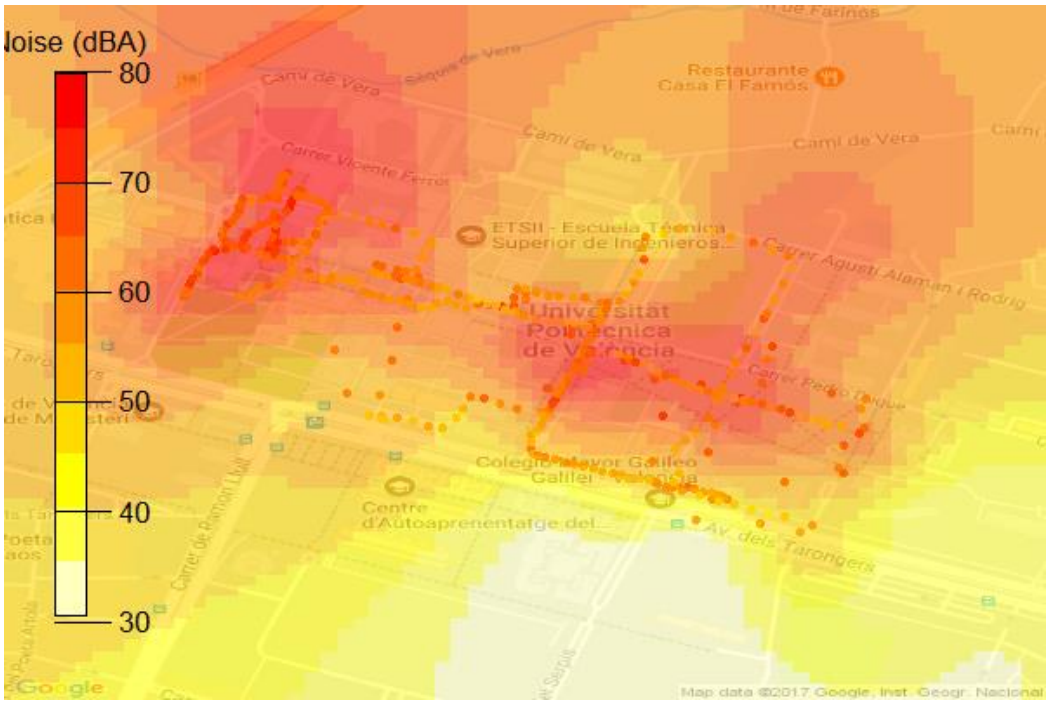

Figura 35. Mapa de calor del Campus de la Universidad Politécnica de Valencia.

6.3. Escenario 3: Centro de la Ciudad de Valencia

En el tercer experimento se trata de evaluar nuestra propuesta en el centro de la Ciudad de Valencia. Además, se comparan los valores de ruido ambiental de nuestra propuesta con el diagnóstico del ruido ambiental que muestra el sitio web del ayuntamiento de Valencia. En la **Figura 36** se presenta un mapa de ruido ambiental del centro de la Ciudad de Valencia, en el cual se puede observar que el nivel ruido en este punto de la Ciudad, según el sitio web, es superior a los 70 dBA.

Figura 36. Mapa de Calor del Centro de Valencia obtenido del Sitio Web del Ayuntamiento de Valencia.

Así, para la realización de este tercer experimento, se definió la tarea con las siguientes configuraciones:

- **Dia de Inicio:** martes 24 de octubre de 2017
- **Dia de Fin:** martes 24 de octubre de 2017
- **Hora Inicio:** 18:15
- **Hora Fin:** 18:45
- **Tipo de área:** polígono

En la **Figura 37** se muestra el área de cobertura del segundo experimento.

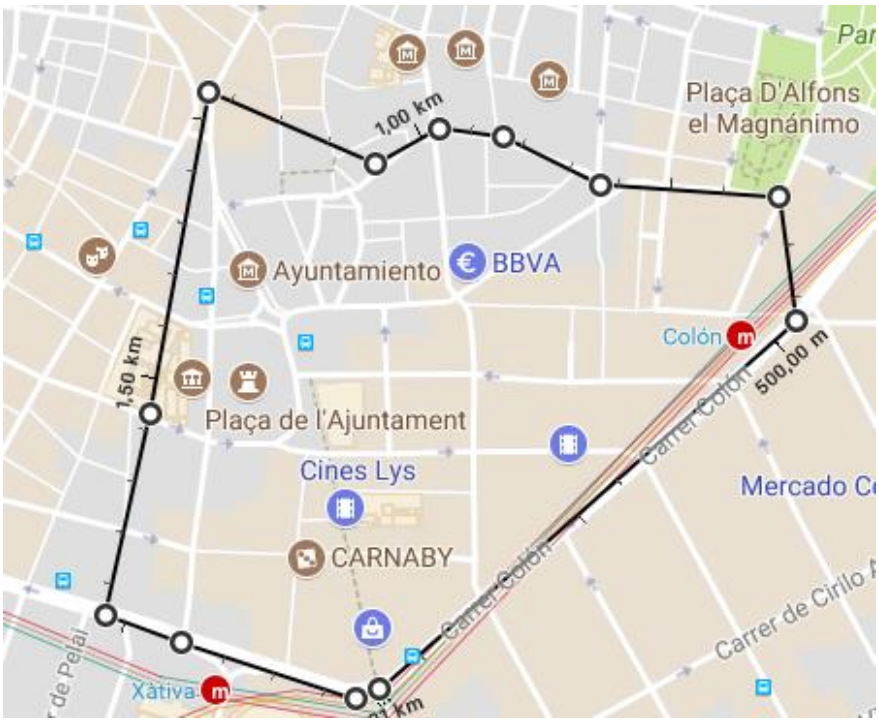

Figura 37. Definición de área para la Tarea 2 usando un polígono.

Con respecto a los clientes móviles, en este tercer experimento se utilizaron 4 dispositivos móviles diferentes, los cuales tenían instalada y activada la aplicación GRCSensing. Los recorridos realizados con los dispositivos móviles se hicieron de forma aleatoria y en simultáneo por las calles y avenidas del centro de la Ciudad, como se muestra en la **Figura 38**.

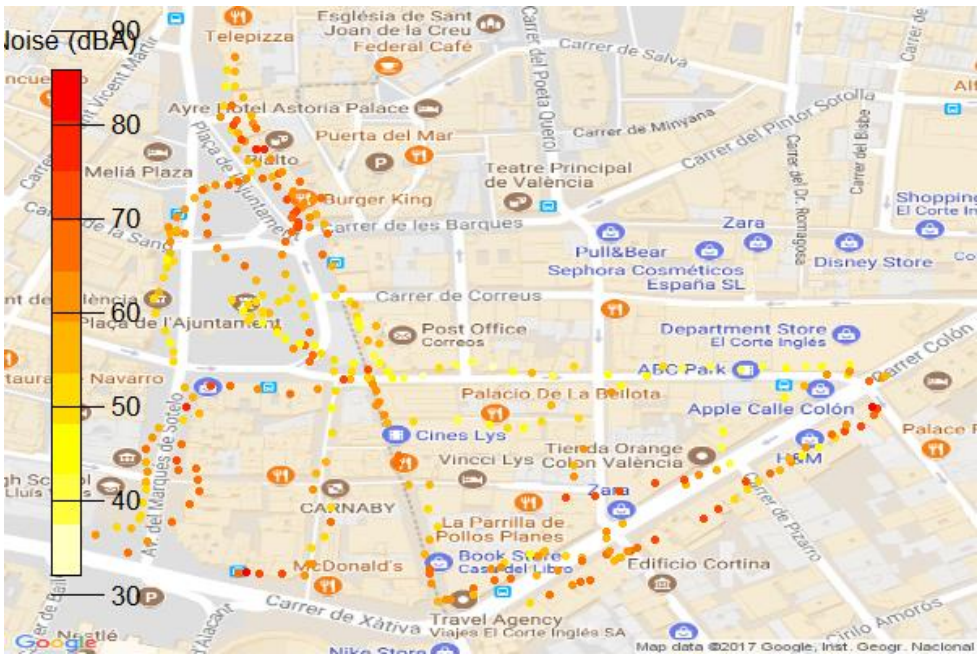

Figura 38. Puntos de recorrido por los clientes móviles en el Centro de la Ciudad de Valencia.

En la **Figura 39** se muestra los resultados obtenidos mediante mapas de calor, permitiendo observar que los lugares donde existe un nivel alto de ruido ambiental se encuentran en zonas donde existe una mayor circulación, tanto vehicular como de personas. En general, al comparar los resultados obtenidos por nuestra propuesta con los valores del sitio web que se muestran en la **Figura 38**, se evidencia que los niveles de ruido son semejantes, y que superan los 70 dBA.

Figura 39. Mapa de Calor del Centro de la Ciudad de Valencia.

7. Conclusiones y Trabajo Futuro

Las soluciones de crowdsensing que se benefician de los teléfonos inteligentes se están incrementando debido a las múltiples ventajas que ofrecen, y se espera que permitan promover aún más el estudio del comportamiento ambiental en base a los datos de un área geográfica determinada.

En este trabajo se propone una plataforma de crowdsensing que permite la monitorización del ruido ambiental. En concreto nos centraremos en la parte del servidor y la integración con el cliente. Para ello se han unificado diferentes tecnologías con el fin de desarrollar un sistema que permita la creación de tareas, además, de la visualización de los niveles de contaminación acústica correspondientes a un área específica.

Por otro lado, gracias al uso de la Firebase Realtime Database, se ha conseguido una estructura de comunicación y almacenamiento eficiente y escalable, permitiendo la sincronización de los datos con todos los clientes en tiempo real.

Globalmente consideramos que, se ha conseguido cumplir con los objetivos definidos para este Trabajo Fin de Master, pues se ha desarrollado completamente el sistema monitorización de la contaminación acústica. Además, se verificó su correcto funcionamiento a través de la realización de diferentes pruebas, capturando datos desde diferentes puntos de la Ciudad de Valencia.

En lo que respecta a trabajo futuro destacar lo siguiente:

- Planeamos introducir en la aplicación móvil funcionalidades adicionales de recolección de datos, realizando el procesamiento de éstos en segundo plano mientras el usuario usa la aplicación para otros fines. Además, se pretende incluir mecanismos para aumentar la calidad del ruido ambiental recolectado por los dispositivos móviles, mediante el uso de sonómetros profesionales y de nuevos algoritmos.
- En el lado del servidor, se puede añadir un nuevo tipo de planificación de tareas, el cual permitirá indicarle a usuario administrador, por medio de la generación de tareas, de cual tarea está cerca de vencerse.

- Esperamos poder utilizar esta plataforma para otros tipos de aplicaciones, como la medición o recolección de datos de otros sensores, incluyendo sensores de temperatura y de luminosidad.

8. Bibliografía

[1] A. García, La contamincación acústica, Valencia: Universidad de Valencia, 2014.

[2] A. M. Ferreira, P. H. Zannin y B. Szeremetta, «Evaluation of Noise Pollution in Urban Parks,» *Environmental Monitoring and Assessment,* vol. 118, nº 1, pp. 423--433, 01 Julio 2006.

[3] F. Sanchis, G. J. Segura , C. Navarro y R. A. García, «Estudio de ruido ambiental y sus efectos en una pequeña ciudad: Banyeres de Mariola,» *Revista de Acústica,* vol. 31, nº 1, pp. 26-31, 2000.

[4] P. M. Usbeth, R. C. Iñiguez , J. Cevo y F. Ayala , «Medición de los niveles de ruido ambiental en la ciudad de Santiago de Chile,» *Revista de otorrinolaringología y cirugía de cabeza y cuello,* vol. 67, nº 1, pp. 122 - 128, 2007.

[5] P. Vinita, B. D. Tripathi y k. Virendra, «Evaluation of traffic noise pollution and attitudes of exposed individuals in working place,» *Atmospheric Environment,* vol. 42, nº 16, pp. 3892 - 3898, 2008.

[6] S. Morrell, R. Taylor y D. Lyle, «A review of health effects of aircraft noise,» *Australian and New Zealand Journal of Public Health,* vol. 21, nº 2, pp. 221-236, 1997.

[7] X. García , I. García y J. García, «Los efectos de la contaminación acústica en la salud: conceptualizaciones del alumnado de Enseñanza Secundaria Obligatoria de Valencia,» *DIDÁCTICA DE LAS CIENCIAS EXPERIMENTALES Y SOCIALES,* nº 24, pp. 123-137, 2010.

[8] B. García Sanz y F. Garrido, La contaminación acústica en nuestras ciudades, Barcelona: Fundación "La Caixa", 2003.

[9] I. Álvarez, J. Martínez, L. Pèrez, F. Figueroa, J. Armas Metre y M. Rivero, «Contaminación ambiental por ruido,» *Revista Médica Electrónica,* vol. 39, nº 3, pp. 640-649, 2017.

[10] E. Kanjo, «NoiseSPY: A Real-Time Mobile Phone Platform for Urban Noise Monitoring and Mapping,» *Mobile Networks and Application,* vol. 15, nº 4, pp. 562-574, 2010.

[11] D. Radu, C. Avram, A. Aştilean , B. Parrein y J. Yi, «Acoustic noise pollution monitoring in an urban environment using a VANET network,» *Proceedings of 2012 IEEE International Conference on Automation, Quality and Testing, Robotics,* nº 5, pp. 244-248, 2012.

[12] A. d. E. Alonso, «Contaminación acústica y salud Noise pollution and health,» *Observatorio medioambiental,* nº 6, pp. 73-95, 2003.

[13] E. Parliament, «Legislation - directive 2002/49/ec of the european parliament and of the council of 25 june 2002 relating to the assessment and management of environmental noise,» *Official Journal of the European Communities,* vol. 45, nº 189, pp. 12-25, 2002.

[14] E. Parliament y Council of the European Union, «"Directives- commission directive (eu) 2015/996 of 19 may 2015 establishing common noise assessment methods according to directive 2002/49/ec of the european parliament and of the council,» *Official Journal of the European Union,* vol. 58, nº 168, p. 1, 2015.

[15] S. Santini, B. Ostermaier y A. Vitaletti, «First Experiences Using Wireless Sensor Networks for Noise Pollution Monitoring,» *Proceedings of the Workshop on Real-world Wireless Sensor Networks,* nº 5, pp. 61-65, 2008.

[16] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, A. Campbell y D. College, «A survey of mobile phone sensing,» *IEEE Communications Magazine,* vol. 48, nº 9, pp. 140-150, 2010.

[17] Z. Willian, C. T. Calafate, J. C. Cano y P. Manzoni, «A Survey on Smartphone-Based Crowdsensing Solutions,» *Mobile Information Systems,* vol. 2016, pp. 1-26, 2016.

[18] J. Catuña, S. Solorzano y J.-M. Clairand, «Noise Pollution Measurement System using Wireless Sensor Network and BAN sensors,» *2017 Fourth International Conference on eDemocracy eGovernment (ICEDEG),* pp. 125-131, 2017.

[19] M. Marinov, D. Nikolov y B. Gaven, «Environmental noise monitoring and mapping,» *2017 40th International Spring Seminar on Electronics Technology (ISSE),* 2017.

[20] T. Yung-chung, S. Bo-rui, L. Chin-tan y W. Chia-chun, «An Implementation of a Distributed Sound Sensing System to Visualize the Noise Pollution,» *Proceedings of the 2017 IEEE International Conference on Applied System Innovation IEEE-ICASI 2017 - Meen, Prior & Lam (Eds),* pp. 625-628, 2017.

[21] D. Radu, C. Avram y A. Astilean, «Acoustic noise pollution monitoring in an urban environment using a VANET network,» *Proceedings of 2012 IEEE International Conference on Automation, Quality and Testing, Robotics,* pp. 244-248, 2012.

[22] C. Tanas y J. Herrera-Joancomart, «Crowdsensing Simulation Using ns-3,» *Citizen in Sensor Networks: Second International Workshop, CitiSens 2013,*

Barcelona, Spain, September 19, 2013, Revised Selected Papers, vol. 8313, pp. 51-62, 2014.

[23] D. Zhao, H. Ma y L. Liu, «Frugal Online Incentive Mechanisms for Mobile Crowd Sensing,» *IEEE Transactions on Vehicular Technology,* vol. 66, nº 4, pp. 3319-3330, 2017.

[24] K. Tollmar y B. Jonsson, «A Mobile Noise Mapping Application,» *NordiCHI '08: Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges,* pp. 415-418, 2008.

[25] S. Soleimani, K. Ehsanali y M. Reza, «Ubisound: Design a User Generated Model inUbiquitous Geospatial Information Environment for SoundMapping,» *ISPRS – International Archives of the Photogrammetry, Remote Sens- ing and Spatial Information Sciences,* vol. XL, pp. 243-247, 2014.

[26] S. Leao, K.-L. Ong y A. Krezel, «2loud?: Com- munitymapping ofexposure to trafficnoisewithmobile phones,» *Environmental Monitoring and Assessment,* vol. 186, nº 10, pp. 6193-6206, 2014.

[27] R. Zuvala, E. Fišerová y L. Marek, «Mathematical aspects of the kriging applied on landslide in Halenkovice (Czech Republic),» *Open Geosciences,* vol. 8, nº 1, pp. 140-156, 2016.

[28] I. Schweizer, F. Probst, R. Bärtl, M. Mühlhäuser y A. Schulz, «NoiseMap – Real-time partic- ipatory noise maps,» *Real-time partic- ipatory noise maps. In Proceedings of the 2nd International Workshop on Sensing Applications on Mobile Phones,* 2011.

[29] A. Muratoni y G. Pau, «Feeling the pack: Strategies for an optimal participatory system to sense and recognize noise pollution,» *2011 IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin),* pp. 17-21, 2011.

[30] E. Kanjo, «Phone Platform for Urban Noise Monitoring and Mapping,» *Mobile Networks and Applications,* vol. 15, nº 4, pp. 562-574, 2010.

[31] S. Hachem, V. Mallet, R. Ventura, A. Pathak y V. Issarny, «Monitoring Noise Pollution Using the Urban Civics Middleware,» *2015 IEEE First International Conference on Big Data Computing Service and Applications,* pp. 52-61, 2015.

[32] M. Wisniewski, G. Demartini, A. Malatras y P. Cudré-Mauroux, «NoizCrowd: A Crowd-Based Data Gathering and Management System for Noise Level Data,» *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* pp. 172-186, 2013.

[33] K. Rana, T. C. Chou, S. Kanhere, N. Bulusu y W. Hu, «Ear-Phone: An End-to-End Participatory Urban Noise Mapping System,» *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks - IPSN '10,* p. 105, 2010.

[34] R. Pryss, B. Langguth, M. Reichert y W. Schlee, «Mobile Crowd Sensing Services for Tinnitus Assessment, Therapy, and Research,» *2015 IEEE International Conference on Mobile Services,* vol. 32, nº 2, pp. 352-359, 2015.

[35] Y. Ren, C. Wang, J. Yang y Y. Chen, «Fine-grained sleep monitoring: Hearing your breathing with smartphones,» *Proceedings - IEEE INFOCOM,* vol. 26, pp. 1194-1202, 2015.

[36] M.-R. Ra, B. Liu, T. La Porta y R. Govindan, «Medusa: A Programming Framework for Crowd-Sensing,» *MobiSys '12 Proceedings of the 10th international conference on Mobile systems, applications, and services,* pp. 337-350, 2012.

[37] V. Agarwal, N. Banerjee, D. Chakraborty y S. Mittal, «USense -- A Smartphone Middleware for Community Sensing,» *2013 IEEE 14th International Conference on Mobile Data Management,* vol. 1, pp. 56-65, 2013.

[38] RabbitMQ, «RabbitMQ,» RabbitMQ, [En línea]. Available: http://www.rabbitmq.com/tutorials/amqp-concepts.html. [Último acceso: 29 Mayo 29].

[39] «Android Studio,» [En línea]. Available: https://developer.android.com/studio/index.html. [Último acceso: 29 09 2017].

[40] R. Soni, «Introduction to Nginx Web Serve,» de *Nginx: From Beginner to Pro*, Berkeley, CA, Apress, 2016, pp. 1--15.

[41] J. Forcier , P. Bissex y W. J Chun, «Python Web Development with Django,» de *Developrs Library*, Boston, Addison-Wesley, 2009, pp. 50-60.

[42] J. Childs-Maidment, «Github,» 7 enero 2017. [En línea]. Available: https://github.com/thisbejim/Pyrebase. [Último acceso: 2017 septiembre 12].

[43] T. Itagaki, J. Cosmas y M. Haque, «An interactive digital television system designed for synchronised and scalable multi-media content over DVB and IP networks,» *2004 IEEE International Conference on Multimedia and Expo (ICME),* vol. III, nº 3, pp. 2155-2158, 2004.

[44] A. Dincer y B. Uraz, Google Maps JavaScript API, Reino Unido: Packt Publishing Ltd, 2013.

[45] «Firebase,» 23 junio 2017. [En línea]. Available: https://firebase.google.com/?hl=es-419. [Último acceso: 12 septiembre 2017].

[46] Anant Narayanan, «Firebase,» 25 Marzo 2013. [En línea]. Available: https://firebase.googleblog.com/2013/03/where-does-firebase-fit-in-your-app.html. [Último acceso: 11 Septiembre 2017].

[47] Firebase, «Firebase,» 7 Agosto 2017. [En línea]. Available: https://firebase.google.com/docs/database/. [Último acceso: 11 Septiembre 2017].

[48] Firebase, «Firebase,» 22 agosto 2017. [En línea]. Available: https://firebase.google.com/docs/database/rest/start. [Último acceso: 12 septiembre 2017].

[49] L. Johansson, Getting Started with RabbitMQ, 2016.

[50] S. S, «API Design Matters,» *ACM Queue Architecting tomorrows's computing,* vol. 5, nº 4, pp. 48-55, 2007.

[51] Cloudamqp, «Cloudamqp,» Cloudamqp, 2 Mayo 2016. [En línea]. Available: https://www.cloudamqp.com/. [Último acceso: 29 Mayo 2017].