

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE  
TELECOMUNICACIÓN UNIVERSIDAD DE MÁLAGA



**PROYECTO FIN DE CARRERA**

SISTEMA DE MONITORIZACIÓN  
WEB DEL SISTEMA  
CARDIOVASCULAR MEDIANTE EL  
USO DEL PROTOCOLO  
BLUETOOTH 4.0 DE BAJA ENERGÍA

INGENIERÍA DE TELECOMUNICACIÓN

Málaga, 2015

Raúl Gómez Acuña



ESCUELA TÉCNICA SUPERIOR DE  
INGENIERÍA DE TELECOMUNICACIÓN  
UNIVERSIDAD DE MÁLAGA

**Titulación:** Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D./D<sup>a</sup>. \_\_\_\_\_

D./D<sup>a</sup>. \_\_\_\_\_

D./D<sup>a</sup>. \_\_\_\_\_

para juzgar el Proyecto Fin de Carrera titulado:

**SISTEMA DE MONITORIZACIÓN WEB DEL SISTEMA  
CARDIOVASCULAR MEDIANTE EL USO DEL  
PROTOCOLO BLUETOOTH 4.0 DE BAJA ENERGÍA**

del alumno *D./D<sup>a</sup>. Raúl Gómez Acuña*  
dirigido por *D./D<sup>a</sup>. Álvaro Durán Martínez*

ACORDÓ POR \_\_\_\_\_ OTORGAR LA

CALIFICACIÓN DE \_\_\_\_\_

y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia.

Málaga, a \_\_\_\_ de \_\_\_\_ de \_\_\_\_

El Presidente:

El Vocal:

El Secretario:

Fdo.:\_\_\_\_\_ Fdo.:\_\_\_\_\_ Fdo.:\_\_\_\_\_



Universidad de Málaga  
Escuela Técnica Superior de Ingeniería de  
Telecomunicación

SISTEMA DE MONITORIZACIÓN  
WEB DEL SISTEMA  
CARDIOVASCULAR MEDIANTE EL  
USO DEL PROTOCOLO  
BLUETOOTH 4.0 DE BAJA ENERGÍA

**REALIZADO POR**  
Raúl Gómez Acuña

**DIRIGIDO POR**  
Álvaro Durán Martínez

**Dpto. de:** Ingeniería de Comunicaciones (IC)

**Palabras clave:** Bluetooth, Android, aplicación, localización, web, corazón

**Titulación:** Ingeniería de Telecomunicación

**Resumen:** El presente proyecto pretende realizar una prueba de concepto sobre el diseño e implementación de un sistema de monitorización del ritmo cardíaco, a través del conjunto de dos aplicaciones, una aplicación Android que se conectará y recibirá datos del sensor mediante el protocolo Bluetooth 4.0 de baja energía, transmitiendo dichos datos a un servidor web y una aplicación web, que usará una conexión web socket para mostrar los datos en tiempo real

Málaga, 10 de septiembre de 2015



Este Proyecto Fin de Carrera está dedicado a mis padres, por su apoyo incondicional durante esta larga etapa, y que sin duda han sido pieza clave para llegar hasta donde estoy

*El autor*



# Acrónimos

<b>AFH</b>	Adaptive Frequency Hoping, o espectro ensanchado adaptable con salto de frecuencia
<b>AJAX</b>	Asynchronous Javascript And XML, o JavaScript asíncrono y XML
<b>AMD</b>	Asynchronous Module Definition, o definición asíncrona de módulos
<b>ATT</b>	Attribute Protocol, o protocolo de atributo
<b>API</b>	Application Programming Interface, o interfaz de programación de aplicaciones
<b>BLE</b>	Bluetooth Low Energy, o Bluetooth de baja energía
<b>CMOS</b>	Complementary Metal Oxide Semiconductor, o semiconductor complementario de óxido metálico
<b>CSS</b>	Cascading Style Sheets, o hoja de estilo en cascada
<b>DOM</b>	Document Object Model, o modelo de objetos del documento
<b>DSL</b>	Domain Specific Language, o lenguaje específico del dominio

<b>EDR</b>	Enhanced Data Rate, o velocidad de transmisión de datos mejorada
<b>ETSIT</b>	Escuela Técnica Superior de Ingeniería de Telecomunicación
<b>GATT</b>	Generic Attribute Profile, o Perfil de Atributo Genérico
<b>GMS</b>	Google Mobile Services, o servicios móviles de Google
<b>GPS</b>	Global Positioning System, o sistema de posicionamiento global
<b>HAL</b>	Hardware Abstraction Layer, o capa de abstracción del hardware
<b>HTML</b>	HyperText Markup Language, o lenguaje de marcas de hipertexto
<b>IDE</b>	Integrated Development Environment, o entorno de desarrollo integrado
<b>ISM</b>	Industrial, Scientific and Medical, o industrial, científico y médico
<b>JNI</b>	Java Native Interface, o interfaz nativa Java
<b>JSON</b>	JavaScript Object Notation, o notación de objetos de JavaScript
<b>MAC</b>	Media Access Control, o control de acceso al medio
<b>MVC</b>	Model-View-Controller, o modelo-vista-controlador
<b>PCB</b>	Printed Circuit Board, o placa de circuito impreso

<b>PFC</b>	Proyecto Fin de Carrera
<b>POO</b>	Programación Orientada a Objetos
<b>RAM</b>	Random Access Memory, o memoria de acceso aleatorio
<b>RSSI</b>	Received Signal Strength Indicator, o indicador de fuerza de la señal recibida
<b>SIG</b>	Special Interest Group, o grupo de especial interés
<b>SVG</b>	Scalable Vector Graphics, o gráficos vectoriales escalables
<b>TDD</b>	Time-Division Duplexing, o duplexación por división de tiempo
<b>UMA</b>	Universidad de Málaga
<b>URL</b>	Uniform Resource Locator, o localizador de recursos uniforme
<b>UUID</b>	Universally Unique Identifier, o identificador universal único
<b>XML</b>	eXtensible Markup Language, o lenguaje de marcado extensible



# Índice

<b>I</b>	<b>Introducción</b>	<b>1</b>
Presentación del Proyecto . . . . .	3	
Motivación . . . . .	4	
Estado del arte . . . . .	6	
Herramientas y metodologías utilizadas . . . . .	7	
Estructura del documento . . . . .	8	
<b>II</b>	<b>Desarrollo del proyecto</b>	<b>9</b>
<b>1</b>	<b>Introducción teórica</b>	<b>11</b>
1.1	Bluetooth . . . . .	11
1.1.1	Historia . . . . .	11
1.1.2	Usos y aplicaciones . . . . .	12
1.1.3	Especificaciones . . . . .	14
1.1.4	Arquitectura . . . . .	18
1.1.5	Bluetooth de baja energía . . . . .	19
1.2	Android . . . . .	24
1.2.1	Interfaz . . . . .	26
1.2.2	Componentes de una aplicación . . . . .	27
1.2.3	Vistas . . . . .	32

1.2.4	Procesos e hilos de ejecución . . . . .	33
1.2.5	Ubicación . . . . .	35
1.2.6	BLE en Android . . . . .	36
1.3	Desarrollo de aplicaciones web modernas . . . . .	39
1.3.1	Definiciones básicas . . . . .	39
1.3.2	MEAN . . . . .	42
1.3.3	MongoDB . . . . .	43
1.3.4	Express . . . . .	44
1.3.5	AngularJS . . . . .	45
1.3.6	Node.js . . . . .	47
<b>2</b>	<b>Implementación</b>	<b>51</b>
2.1	Análisis de requisitos . . . . .	51
2.1.1	Requisitos funcionales . . . . .	51
2.1.2	Requisitos no funcionales . . . . .	54
2.1.3	Diseño de la interfaz gráfica . . . . .	54
2.1.4	Dispositivo Android . . . . .	58
2.2	Aplicación Android . . . . .	59
2.2.1	Entorno de desarrollo integrado . . . . .	59
2.2.2	Estructura del código . . . . .	62
2.2.3	Interfaz gráfica . . . . .	63
2.2.4	Diagrama de flujo . . . . .	67
2.2.5	Comunicación con el sensor mediante BLE . . . . .	69
2.2.6	Ubicación . . . . .	76
2.2.7	Comunicación con el servidor web . . . . .	78
2.3	Aplicación web . . . . .	80
2.3.1	Entorno de desarrollo integrado . . . . .	80
2.3.2	Estructura del código . . . . .	81

2.3.3	Diagrama de flujo . . . . .	83
2.3.4	Servidor . . . . .	85
2.3.5	Cliente . . . . .	90
<b>3</b>	<b>Plan de pruebas y verificación</b>	<b>101</b>
3.1	Funcionamiento de la aplicación . . . . .	101
3.2	Pruebas realizadas . . . . .	108
3.2.1	Comparación con electrocardiograma . . . . .	108
3.2.2	Consumo de energía . . . . .	111
3.2.3	Retardo del sistema . . . . .	111
3.3	Realización de los objetivos . . . . .	113
3.3.1	Aplicación Android . . . . .	113
3.3.2	Aplicación web . . . . .	115
<b>III</b>	<b>Conclusiones y líneas futuras</b>	<b>119</b>
Conclusiones . . . . .	121	
Líneas de trabajo futuras . . . . .	122	
<b>IV</b>	<b>Apéndices</b>	<b>125</b>
<b>A</b>	<b>Proceso de integración de una aplicación node.js en Heroku</b>	<b>127</b>
<b>Referencias</b>		<b>135</b>



# Índice de figuras

1	Sistema de monitorización del ritmo cardíaco (visión general) . . . . .	5
1.1	Origen del logotipo comercial de Bluetooth [SIG00] . . . . .	12
1.2	Arquitectura de Hardware del protocolo Bluetooth [dV12] . . . . .	18
1.3	Arquitectura de Software del protocolo Bluetooth [dV12] . . . . .	20
1.4	Capa Física BLE [Arg14] . . . . .	22
1.5	Diagrama de la estructura del sistema operativo Android[And08] . .	25
1.6	Notificaciones minimizadas en el área de notificaciones[And08] . . .	26
1.7	Diagrama del ciclo de vida de una actividad en Android [And08]. . .	29
1.8	Diagrama del ciclo de vida de un servicio [And08] . . . . .	32
1.9	Esquema Servidor y Cliente GATT [And08] . . . . .	39
1.10	Ejemplo de directivas AngularJS incrustadas en HTML [Pen14] . . .	47
1.11	Sistema de eventos en NodeJS [Dah09] . . . . .	49
2.1	Bosquejo Android estado Reposo . . . . .	55
2.2	Bosquejo Android estado Búsqueda . . . . .	55
2.3	Bosquejo Android estado Desconectado . . . . .	56
2.4	Bosquejo Android estado Conectado . . . . .	56
2.5	Bosquejo web con sensor desconectado . . . . .	58
2.6	Bosquejo web con sensor conectado . . . . .	59
2.7	Entorno de desarrollo integrado Android Studio . . . . .	60

2.8	Entorno de desarrollo integrado Eclipse . . . . .	61
2.9	Organización del código fuente . . . . .	63
2.10	Editor de archivos de recurso XML con vista previa . . . . .	64
2.11	Edición completamente visual de la interfaz gráfica de usuario de la aplicación . . . . .	65
2.12	Barra de Acción en la actividad principal <code>MainActivity</code> . . . . .	67
2.13	Diagrama de flujo de la aplicación Android . . . . .	68
2.14	Arquitectura de Bluetooth en Android [And07] . . . . .	70
2.15	Diagrama de interacción de la aplicación y los servicios . . . . .	77
2.16	Entorno de desarrollo Atom . . . . .	81
2.17	Estructura global de la aplicación web . . . . .	82
2.18	Diagrama de flujo del servidor web . . . . .	84
2.19	Árbol de dependencias en la aplicación Angular . . . . .	96
3.1	Pantalla inicial . . . . .	102
3.2	Dispositivos encontrados . . . . .	102
3.3	Conexión GATT sin establecer . . . . .	103
3.4	Conexión GATT establecida . . . . .	103
3.5	Cliente web cuando el pulsómetro está desconectado . . . . .	104
3.6	Cliente web cuando el pulsómetro está conectado . . . . .	105
3.7	Notificación por valor fuera de rango . . . . .	106
3.8	Notificación de estabilización . . . . .	106
3.9	Cliente web en un dispositivo móvil . . . . .	107
3.10	Ubicación del usuario en el servicio de mapas de Google . . . . .	107
3.11	Servicio de almacenamiento MongoLab en la nube . . . . .	108
3.12	Sigma R1 Blue Comfortex+ . . . . .	109
3.13	Captura del sistema en funcionamiento . . . . .	112
3.14	Resumen de las pruebas realizadas . . . . .	114

A.1	Autentificación en el terminal en Heroku [Her07]	128
A.2	Preparando a Heroku para crear la aplicación [Her07]	129
A.4	Consulta de logs de nuestra aplicación [Her07]	129
A.3	Enviando el código fuente a Heroku [Her07]	130
A.5	Apariencia del dashboard en Heroku [Her07]	131



# Índice de fragmentos de código

2.1	Descripción de la interfaz gráfica de la actividad <code>MainActivity</code> . . . . .	66
2.2	Detección de soporte BLE en un dispositivo Android . . . . .	71
2.3	Configuración BLE en Android . . . . .	72
2.4	Proceso de escaneo de dispositivos BLE en Android . . . . .	73
2.5	Implementación de la interfaz <code>BluetoothAdapter.LeScanCallback</code> . . . . .	74
2.6	Parsing para el perfil de ritmo cardíaco . . . . .	76
2.7	Sección de dependencias dentro del archivo <code>build.gradle</code> . . . . .	77
2.8	Solicitud de conexión a los GMS. . . . .	78
2.9	Método para el envío de valores de frecuencia cardíaca al servidor web	79
2.10	Método para el envío de valores de localización al servidor web . . . . .	80
2.11	Configuración del servidor Express . . . . .	85
2.12	Controlador POST para valores de frecuencia cardíaca . . . . .	86
2.13	Modelo del objeto usado en la base de datos . . . . .	87
2.14	Funcion expuesta en <code>heartRate-notifications.js</code> . . . . .	89
2.15	Ejemplo de controlador Angular en Javascript . . . . .	93
2.16	Referencia del controlador Angular en la vista . . . . .	93
2.17	Vista principal <code>index.html</code> . . . . .	98



# Índice de Tablas

1	Usuarios de teléfonos móviles inteligentes [eMa14] . . . . .	6
2	Cuota de mercado de los distintos sistemas operativos móviles [RL14]	7
1.1	Clases de dispositivos Bluetooth [Wik04] . . . . .	13
1.2	Comparación de BLE con otros protocolos [Arg14] . . . . .	21
1.3	Comparación básica de LAMP y MEAN [Pen14] . . . . .	42
1.4	Comparación entre lenguajes utilizados en LAMP y MEAN [Pen14] .	42
3.1	Tabla de comparacion de mediciones . . . . .	110



# Parte I

## Introducción



# Introducción y visión general

## Presentación del Proyecto

Este proyecto es fundamentalmente una prueba de concepto. Para los desconocidos del término podemos resumir en breves palabras que se trata del diseño e implementación de un prototipo, con el propósito de verificar que el concepto es susceptible de ser explotado de una manera útil.

Nos hemos atrevido con una idea interesante, y a su vez desafiante, como resulta ser un sistema de monitorización del sistema cardiovascular en tiempo real, y usando unos recursos muy simples y al alcance de cualquiera, como son el uso de un pulsómetro comercial con Bluetooth integrado de última generación y un teléfono inteligente Android. Se ha desarrollado una aplicación Android, la cual recibirá datos transmitidos por el pulsómetro mediante el protocolo Bluetooth 4.0, en su modo de baja energía, y enviará dichos datos a un servidor web a través del método POST del protocolo HTTP.

En este prototipo el término “tiempo real” es el aspecto clave, puesto que el objetivo es que cualquier persona (en un principio) sea capaz de ver inmediatamente reflejado en la web el último valor de frecuencia cardíaca detectado por el pulsómetro. En otras palabras, usted se encuentra realizando ejercicio en el exterior con el pulsómetro atado al pecho y con su teléfono móvil Android en el bolsillo, y cualquier persona interesada, por ejemplo un doctor, podría consultar en directo su ritmo car-

diovascular y ver dicha evolución en una gráfica desde cualquier dispositivo móvil con acceso a Internet.

Sin embargo, para tal propósito nos falta la otra pieza del puzzle para que todo encaje perfectamente: conexiones de baja latencia cliente-servidor y servidor-cliente, siendo el servidor un servidor web el cual implementaremos desde cero y tendremos total control sobre él, y cliente cualquier navegador web. ¿Se preguntan cómo conseguir dicha baja latencia? Los Web Sockets nos abren el camino.

La figura 1 provee una visión general del sistema que se ha implementado.

## Motivación

Entre los tecnólogos, la salud móvil está en auge. Según estudios del Instituto Tecnológico de Massachusetts, desde principios de 2013 se han invertido más de 550 millones de euros en capital riesgo en empresas para crear productos de salud móvil e invirtiendo en *start-ups*.

La idea es simple: cada vez hay más teléfonos inteligentes, lo que significa que el uso generalizado de sensores pequeños y baratos, Bluetooth de baja energía y software de análisis permitirán que pacientes y médicos registren todo tipo de datos para mejorar la atención médica.

Con el lanzamiento de la versión 4.3 a mediados de 2013, Android introdujo la compatibilidad con Bluetooth 4.0 y su modo de baja energía. Esto hecho abrió la puerta a poder conectar sensores a aplicaciones Android, como por ejemplo medidores de frecuencia cardíaca, los cuales pueden adquirirse por un precio que ronda los 30-40 euros y proporcionan una alta precisión en la medida.

La combinación de leer y almacenar datos de sensores en nuestro dispositivo Android, junto con un acceso completo a Internet desde este, nos garantiza un mundo lleno de posibilidades en el campo de la monitorización, poniendo esos datos a

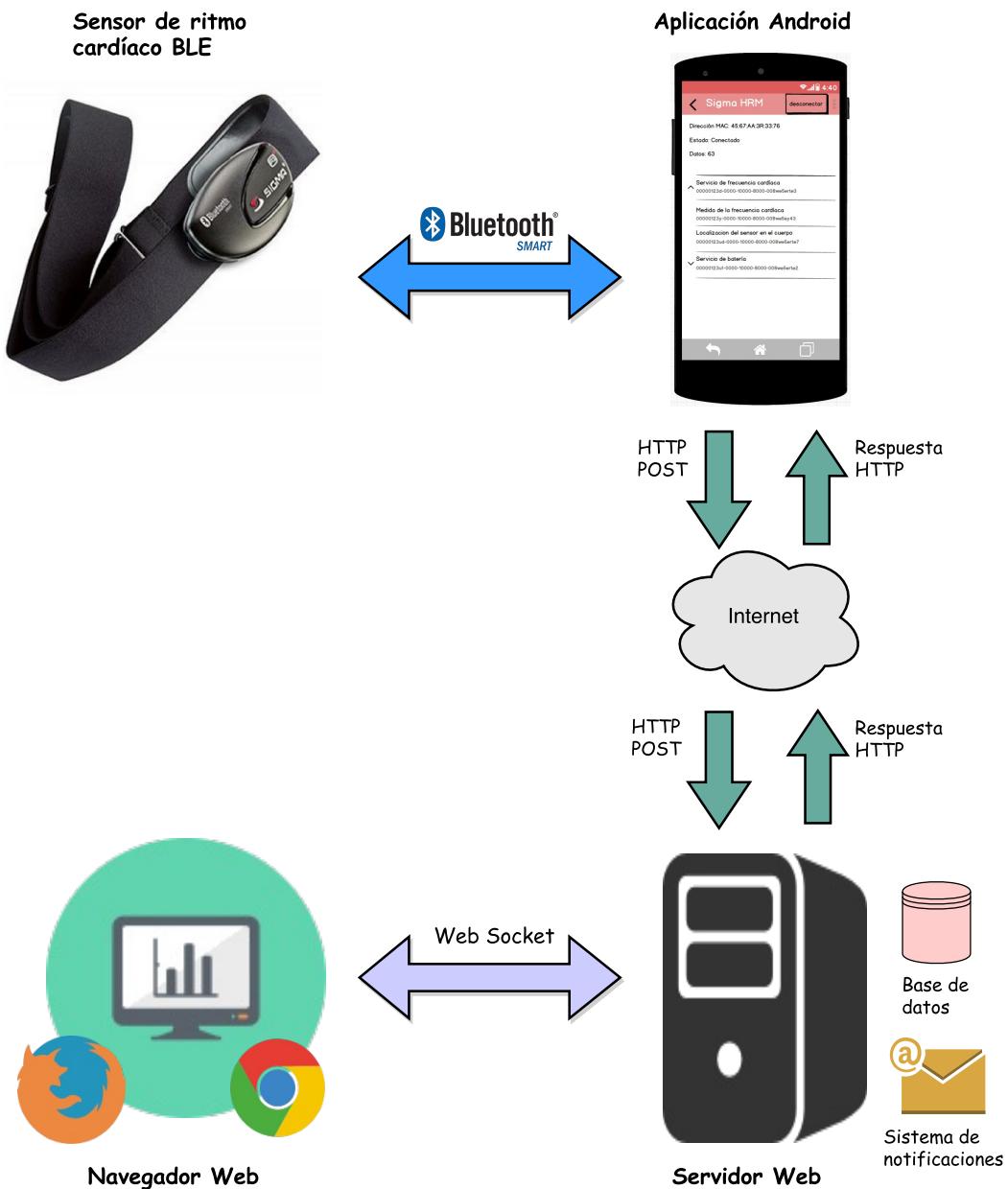


Figura 1: Sistema de monitorización del ritmo cardíaco (visión general)

disponibilidad de un amplio público, todo ello en tiempo real gracias a la experiencia que la web nos brinda.

## Estado del arte

La explosión que el desarrollo web ha sufrido en los últimos años, proporcionando notables avances tecnológicos como NodeJS (JavaScript del lado del servidor), pilas o *stacks* de desarrollo web completo como MEAN (usando Javascript como único lenguaje en todo la pila) e implementaciones de Web Sockets usando JavaScript y NodeJS, ha conseguido que podamos desarrollar ricas aplicaciones web en menor cantidad de tiempo y con una menor complejidad técnica.

Por otra parte, la penetración de mercado de los teléfonos inteligentes crece día a día, superando 1.75 mil millones de usuarios para final de 2014[eMa14]. Como se observa en la tabla 1, cerca de dos quintos de la base total de usuarios de teléfonos móviles, utiliza teléfonos móviles inteligentes. Esto supone cerca de un cuarto de la población mundial.

	<b>2012</b>	<b>2013</b>	<b>2014</b>	<b>2015</b>	<b>2016</b>	<b>2017</b>
<b>Usuarios totales (miles de millones)</b>	1.13	1.43	1.75	2.03	2.28	2.50
<b>% de incremento</b>	68.4 %	27.1 %	22.5 %	15.9 %	12.3 %	9.7 %
<b>% de usuarios móviles</b>	27.6 %	33.0 %	38.5 %	42.6 %	46.1 %	48.8 %
<b>% de población mundial</b>	16.0 %	20.2 %	24.4 %	28.0 %	31.2 %	33.8 %

Tabla 1: Usuarios de teléfonos móviles inteligentes [eMa14]

Dentro del segmento de usuarios de teléfonos móviles inteligentes, existen varios sistemas operativos móviles, cada cual provee su propia plataforma de desarrollo,

entorno de desarrollo y ecosistema de aplicaciones. A raíz de la información de la tabla 2, se observa que Android es el líder del mercado, con sobrada ventaja.

<b>Período</b>	<b>Android</b>	<b>iOS</b>	<b>Windows Phone</b>	<b>BlackBerry OS</b>	<b>Otros</b>
<b>T3 2014</b>	84.4 %	11.7 %	2.9 %	0.5 %	0.6 %
<b>T3 2013</b>	81.2 %	12.8 %	3.6 %	1.7 %	0.6 %
<b>T3 2012</b>	74.9 %	14.4 %	2.0 %	4.1 %	4.5 %
<b>T3 2011</b>	57.4 %	13.8 %	1.2 %	9.6 %	18.0 %

Tabla 2: Cuota de mercado de los distintos sistemas operativos móviles [RL14]

## Herramientas y metodologías utilizadas

Para el desarrollo de la aplicación Android, la principal herramienta utilizada ha sido el entorno de desarrollo integrado Android Studio. Es un entorno de relativa novedad, ya que aunque su primera versión estable fue liberada en Diciembre de 2014, su desarrollo se remonta a Mayo de 2013. Android Studio está disponible de manera gratuita para las plataformas mayoritarias Windows, Mac OS X y Linux. Al ser un entorno de desarrollo enfocado completamente al ecosistema Android, provee múltiples ayudas y mejoras sobre un entorno de desarrollo integrado más genérico como puede ser Eclipse. Características que han hecho decantarse por Android Studio y no Eclipse se analizan en el apartado 2.2.1.

Para el desarrollo de la aplicación web, se ha recurrido al editor de texto gratuito y de código fuente abierto Atom. Es un entorno altamente personalizable que explota lo mejor de los editores más populares del mercado.

Ha sido necesario también un teléfono inteligente con Android, tanto durante la fase de desarrollo como durante las posteriores de utilización y realización de pruebas, así como un ordenador portátil Macbook Pro, actuando como servidor web

durante la fase de desarrollo.

## Estructura del documento

### Introducción teórica

En el capítulo 1 se hará una introducción a los conceptos teóricos requeridos para la comprensión del trabajo realizado. Se explicará el protocolo Bluetooth y su versión de baja energía. A su vez, también se explicarán conceptos relativos a la programación en Android, tratados a lo largo del proyecto. Por último se realizará una breve introducción al desarrollo de aplicaciones web modernas, haciendo hincapié en el *stack* MEAN, ya que ha sido el utilizado en el presente proyecto.

### Implementación

En el capítulo 2 se explicará cómo se ha llevado a cabo toda la implementación de la aplicación, empezando por un análisis de los requisitos necesarios para la realización de esta, y siguiendo con los aspectos relevantes a la programación de la aplicación tanto Android como Web: métodos, interfaz gráfica, diagramas de flujo, algoritmos, etc.

### Plan de pruebas y verificación

En el capítulo 3 se explicarán los procedimientos llevados a cabo para cerciorar el correcto funcionamiento de la aplicación.

### Conclusiones y trabajo futuro

En este apartado se pretende representar las conclusiones obtenidas durante la realización de este proyecto, así como plasmar posibles ideas que se consideran interesantes de cara a una futura continuidad en el desarrollo de la aplicación.

## Parte II

# Desarrollo del proyecto



# Capítulo 1

## Introducción teórica

### 1.1. Bluetooth

#### 1.1.1. Historia

En 1994 Ericsson inició un estudio para investigar la viabilidad de un sistema de comunicaciones vía radio, de bajo coste y bajo consumo, para la interconexión entre teléfonos móviles y otros accesorios con la intención de eliminar cables entre aparatos. El estudio partía de un largo proyecto que investigaba sobre unos dispositivos transceptores conectados a una red celular, hasta llegar a un enlace de radio de corto alcance, llamado *MC link*.

Conforme este proyecto avanzaba, se empezó a visualizar que este tipo de enlace podía ser utilizado ampliamente en un gran número de aplicaciones, ya que tenía como principal virtud el basarse en un chip de radio relativamente económico.

A comienzos de 1997, según avanzaba el proyecto *MC link*, Ericsson fue despertando el interés de otros fabricantes de equipos portátiles. De seguida se vio claramente que para que el sistema tuviera éxito un gran número de equipos deberían estar equipados con esta tecnología. Esto fue lo que llevó, a principios de 1998,

a crear un Special Interest Group, o grupo de especial interés (SIG), formado por 5 promotores que eran: Ericsson, Nokia, IBM, Toshiba e Intel. La idea era lograr un conjunto adecuado de áreas de negocio, dos líderes del mercado de las telecomunicaciones, dos líderes del mercado de los ordenadores portátiles y un líder de la fabricación de chips. A día de hoy hay más de 20.000 empresas que lo conforman.

Obviamente Bluetooth no se llamó así desde un comienzo. El origen surge en 1994, cuando uno de los desarrolladores, Jim Kardach, propuso el nombre de uno de los reyes vikingos, específicamente el de Harald Blåtand, cuya traducción al inglés es Harald Bluetooth.

Este rey fue conocido por unificar las tribus noruegas, suecas-danesas y por convertirlas al cristianismo, que en paralelo es la fusión de la comunicación de los sistemas digitales, que buscaba reemplazar el cable para los dispositivos móviles.

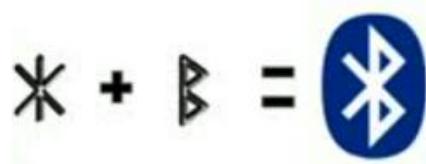


Figura 1.1: Origen del logotipo comercial de Bluetooth [SIG00]

Aún hay más misterios escandinavos relacionados con la tecnología como cuál es el origen del logotipo. El logotipo es una combinación de dos letras del alfabeto rúnico, precisamente la H y la B. Dicha fusión se aprecia en la figura 1.1.

### 1.1.2. Usos y aplicaciones

Se denomina Bluetooth a la norma que define un estándar global de comunicación inalámbrica, que posibilita la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia. Los principales objetivos que se pretende conseguir con esta norma son: facilitar las comunicaciones entre equipos móviles

y fijos, eliminar cables y conectores entre estos, ofreciendo la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre nuestros equipos personales.

Los dispositivos que incorporan este protocolo pueden comunicarse entre sí cuando se encuentran dentro de su radio de alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión es suficiente. Estos dispositivos se clasifican como Clase 1, Clase 2 o Clase 3 en referencia a su potencia de transmisión, tal y como muestra la tabla 1.1

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Alcance (aproximado)
Clase 1	100 mW	20 dBm	100 metros
Clase 2	2.5 mW	4 dBm	5-10 metros
Clase 3	1 mW	0 dBm	1 metro

Tabla 1.1: Clases de dispositivos Bluetooth [Wik04]

Para utilizar Bluetooth, un dispositivo debe implementar alguno de los perfiles Bluetooth. Los perfiles son descripciones de comportamientos generales que los dispositivos pueden utilizar para comunicarse, formalizados para favorecer un uso unificado. La forma de utilizar las capacidades de Bluetooth se basa, por tanto, en los perfiles que soporta cada dispositivo. Los perfiles permiten la manufactura de dispositivos que se adapten a sus necesidades. Ejemplos de estos son el perfil *File Transfer Profile* (FTP) para la transferencia de ficheros a través de Bluetooth, o el perfil *Advanced Audio Distribution Profile* (A2DP), el cual define cómo se puede propagar un flujo de audio (mono o estéreo) entre dispositivos a través de una conexión Bluetooth.

## Aplicaciones

- Conexión sin cables para el intercambio de datos.
- Transferencia de fichas de contactos, citas y recordatorios entre dispositivos.
- Reemplazo de la tradicional comunicación por cable entre equipos GPS y equipamiento médico.
- Controles remotos (tradicionalmente dominado por el infrarrojo).
- Enviar pequeñas publicidades desde anunciantes a dispositivos con Bluetooth.  
Un negocio podría enviar publicidad a teléfonos móviles cuyo Bluetooth (los que lo posean) estuviera activado al pasar cerca
- Enlace inalámbrico entre sistemas de audio y los altavoces correspondientes.
- Controladores remotos inalámbricos para videojuegos.

### 1.1.3. Especificaciones

#### La tecnología

La especificación de Bluetooth define un canal de comunicación de máximo 720 kbps (kilobits por segundo) con un rango óptimo de 10 metros (opcionalmente 100 m con repetidores).

Para poder operar en todo el mundo es necesaria una banda de frecuencia abierta a cualquier sistema de radio independientemente del lugar del planeta donde nos encontremos. Sólo la banda Industrial, Scientific and Medical, o industrial, científico y médico (ISM) de 2,45 GHz cumple con este requisito, con rangos que van de los 2.400 MHz a los 2.480 MHz, con algunas restricciones en países como Francia, España y Japón.

Para evitar interferencias, ya que la banda ISM está abierta a cualquiera, se utiliza el sistema de salto de frecuencia. Este sistema divide la banda de frecuencia en varios canales de salto donde los transceptores, durante la conexión, van cambiando de uno a otro de manera pseudo-aleatoria, consiguiendo que el ancho de banda instantáneo sea muy pequeño y también una propagación sobre el total de la banda.

El canal pues, queda dividido en intervalos de  $625 \mu\text{s}$ , llamados ranuras o *slots*, donde cada salto de frecuencia es ocupado por una ranura. Esto da lugar a una frecuencia de salto de 1600 veces por segundo. Un paquete de datos puede ocupar una ranura para la emisión y otro para la recepción y pueden ser usados alternativamente, dando lugar a un esquema de tipo Time-Division Duplexing, o duplexación por división de tiempo (TDD). Los saltos de frecuencia se dan entre un total de 79 frecuencias con intervalos de 1MHz, proporcionando seguridad y robustez.

La potencia de salida para transmitir a una distancia máxima de 10 metros es de 0 dBm (1 mW), mientras que la versión de largo alcance transmite entre 20 y 30 dBm (entre 100 mW y 1 W).

Para lograr alcanzar el objetivo de bajo consumo y bajo coste, se ideó una solución que se puede implementar en un solo chip utilizando circuitos Complementary Metal Oxide Semiconductor, o semiconductor complementario de óxido metálico (CMOS). De esta manera, se logró crear una solución de 9x9 mm y que consume aproximadamente 97% menos energía que un teléfono celular común.

El protocolo de banda base (canales simples por línea) combina conmutación de circuitos y paquetes. Para asegurar que los paquetes no lleguen fuera de orden, las ranuras pueden ser reservadas por paquetes síncronos. A su vez, un salto diferente de señal es usado para cada paquete. Por otro lado, la conmutación de circuitos puede ser asíncrona o síncrona.

Tres canales de datos síncronos (voz), o un canal de datos síncrono y uno asíncrono, pueden ser soportados en un solo canal. Cada canal de voz puede soportar

una tasa de transferencia de 64 kbps en cada sentido, la cual es suficientemente adecuada para la transmisión de voz. Un canal asíncrono puede transmitir como mucho 721 kbps en una dirección y 56 kbps en la dirección opuesta, sin embargo, para una conexión asíncrona es posible soportar 432,6 kbps en ambas direcciones si el enlace es simétrico.

## Versiones

- Bluetooth v1.0 y v1.0b

Las versiones 1.0 y 1.0b han tenido muchos problemas, y los fabricantes temían dificultades para hacer sus productos interoperables. Las versiones 1.0 y 1.0b incluyen en hardware de forma obligatoria la dirección del dispositivo Bluetooth en la transmisión (el anonimato se hace imposible a nivel de protocolo), lo que fue un gran revés para algunos servicios previstos para su uso en entornos Bluetooth.

- Bluetooth v1.1 (2002)

Ratificado como estándar IEEE 802.15.1-2002, se corrigieron muchos errores en las especificaciones 1.0b. Se añadió soporte para canales no cifrados así como Received Signal Strength Indicator, o indicador de fuerza de la señal recibida (RSSI)

- Bluetooth v1.2 (2003)

Incorpora la técnica Adaptive Frequency Hoping, o espectro ensanchado adaptable con salto de frecuencia (AFH), que ejecuta una transmisión más eficiente y un cifrado más seguro.

- Bluetooth v2.0 + EDR (2004)

Esta versión de la especificación principal de Bluetooth fue lanzada en 2004 y es compatible con la versión anterior 1.2. La principal diferencia está en

la introducción de Enhanced Data Rate, o velocidad de transmisión de datos mejorada (EDR) para acelerar la transferencia de datos. La tasa nominal de EDR es de 3 Mbps, aunque la tasa de transferencia de datos práctica sea de 2,1 Mbps. EDR puede proporcionar un menor consumo de energía a través de un ciclo de trabajo reducido.

- **Bluetooth v2.1 + EDR (2007)**

La versión 2.1 de la especificación Bluetooth Core + EDR es totalmente compatible con 1.2, y fue adoptada por el Bluetooth SIG el 26 de julio de 2007. Se mejora la experiencia de emparejamiento de dispositivos Bluetooth, mientras que aumenta el uso y la fuerza de seguridad.

- **Bluetooth v3.0 + HS (2009)**

La versión 3.0 + HS de la especificación principal de Bluetooth fue aprobada por el Bluetooth SIG el 21 de abril de 2009. El bluetooth 3.0 +HS soporta velocidades teóricas de transferencia de datos de hasta 24 Mbps entre sí, aunque no a través del enlace Bluetooth propiamente dicho. La conexión Bluetooth nativa se utiliza para la negociación y el establecimiento mientras que el tráfico de datos de alta velocidad se realiza mediante un enlace 802,11 (Wi-Fi).

- **Bluetooth v4.0 (2010)**

El SIG de Bluetooth ha completado la especificación del Núcleo de Bluetooth en su versión 4.0, que incluye al Bluetooth clásico, el Bluetooth de alta velocidad y los protocolos Bluetooth de bajo consumo. El Bluetooth de alta velocidad se basa en Wi-Fi, y el Bluetooth clásico consta de protocolos Bluetooth preexistentes. Esta versión ha sido adoptada el 30 de junio de 2010.

Bluetooth Low Energy, o Bluetooth de baja energía (BLE) es un subconjunto de Bluetooth v4.0 con una pila de protocolo completamente nueva para desarrollar rápidamente enlaces sencillos. Como alternativa a los protocolos

estándar de Bluetooth que se introdujeron en Bluetooth v1.0 a v4.0 está dirigido a aplicaciones de muy baja potencia alimentados con una pila botón.

### 1.1.4. Arquitectura

#### Arquitectura de Hardware

El hardware que compone el dispositivo Bluetooth esta compuesto por dos partes, un dispositivo de radio, encargado de modular y transmitir la señal y un controlador digital. La figura 1.2 muestra dicha separación

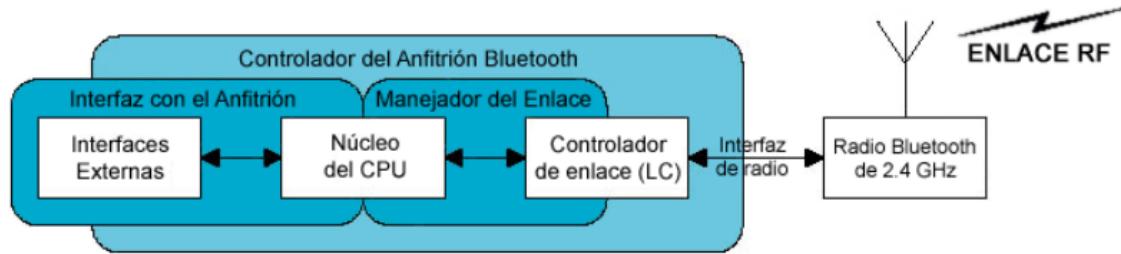


Figura 1.2: Arquitectura de Hardware del protocolo Bluetooth [dV12]

El controlador digital esta compuesto por un CPU, por un procesador de señales digitales llamado Controlador de Enlace y por los interfaces con el dispositivo anfitrión. El controlador de enlace se encarga de hacer el procesamiento de la banda base y del manejo de los protocolos ARQ y FEC de la capa física para el manejo y corrección de errores. Además, se encarga de las funciones de transferencia (tanto asíncrona como síncrona), codificación de audio y encripción de datos. El CPU del dispositivo se encarga de atender las instrucciones relacionadas con Bluetooth del dispositivo anfitrión, para así simplificar su operación. Para ello, sobre el CPU corre un software denominado Manejador del Enlace, que tiene la función de comunicarse

con otros dispositivos por medio del protocolo LMP.

### Arquitectura de Software

Buscando ampliar la compatibilidad de los dispositivos Bluetooth, los dispositivos que se apegan al estándar utilizan como interfaz entre el dispositivo anfitrión (ordenador portátil, teléfono celular, etcétera) y el dispositivo Bluetooth como tal (chip Bluetooth) una interfaz denominada HCI (Host Controller Interface).

Los protocolos de alto nivel como el SDP (Protocolo utilizado para encontrar otros dispositivos Bluetooth dentro del rango de comunicación, encargado, también, de detectar la función de los dispositivos en rango), RFCOMM (Protocolo utilizado para emular conexiones de puerto serial) y TCS (Protocolo de control de telefonía) interactúan con el controlador de banda base a través del Protocolo L2CAP (Logical Link Control and Adaptation Protocol). El protocolo L2CAP se encarga de la segmentación y re-ensamblaje de los paquetes para poder enviar paquetes de mayor tamaño a través de la conexión Bluetooth.

La arquitectura de software completa puede contemplarse en la figura 1.3.

#### 1.1.5. Bluetooth de baja energía

Bluetooth es uno de los protocolos inalámbricos más populares, y ha estado disponible en los teléfonos inteligentes, computadoras y otros dispositivos durante más de una década. El crecimiento explosivo de los dispositivos Bluetooth llevó a Bluetooth SIG y a distintas empresas a la realización de que Bluetooth consumía demasiada energía y empleaba demasiado tiempo para conectarse en ciertas aplicaciones.

Bluetooth v4.0 introdujo BLE , oficialmente conocido como *Bluetooth Smart* o Bluetooth Inteligente. Esta especificación introduce un estándar radio completamente diferente, utilizando menos energía y con un coste menor, para así satisfacer las necesidades de estas nuevas aplicaciones. El amplio soporte de BLE en teléfonos

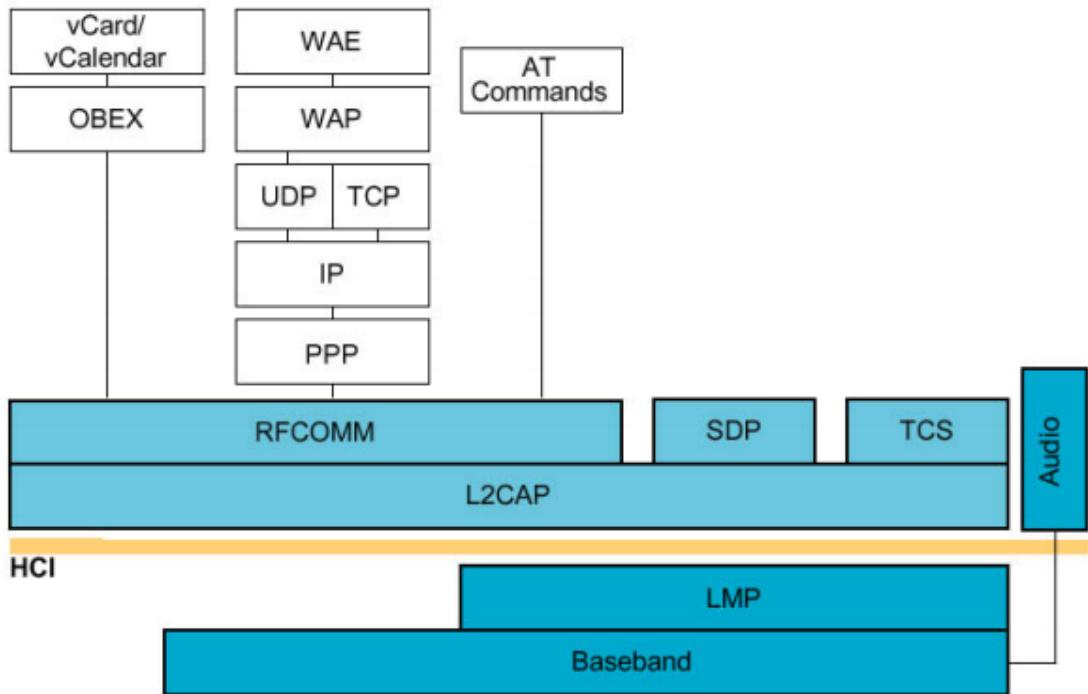


Figura 1.3: Arquitectura de Software del protocolo Bluetooth [dV12]

inteligentes y tabletas hace que las aplicaciones BLE sean fáciles de implementar y fáciles de usar para los consumidores.

El iPhone 4s de Apple introdujo soporte para BLE y abrió el camino a un número masivo de dispositivos pequeños que son capaces de funcionar con pequeñas baterías. Antes de BLE, el Bluetooth Clásico necesitaba baterías más grandes por el gran uso de energía, y también requería de un chip de autenticación que era costoso. BLE resuelve estos problemas para conectarse con iPhones y iPads. Muchos teléfonos basados en Android también presentan soporte para BLE, concretamente desde la versión de Android 4.3. El SIG de Bluetooth predice que para el 2018 más del 90 % de los teléfonos móviles soportarán BLE

Uno de los aspectos más poderosos de BLE es su flexibilidad y baja energía, que

permite intercambiar información en forma genérica, a diferencia de la estructura rígida del Bluetooth Clásico.

### Comparación con otros protocolos

Bluetooth y BLE son protocolos que pueden simplificar la conectividad de productos, pero es importante entender como se comparan a otras tecnologías inalámbricas. WiFi, Zigbee y otros protocolos son mejores en ciertas aplicaciones y BLE no es siempre la mejor solución. La tabla 1.2 muestra algunas de las características que diferencian a BLE de ellos:

	<b>BLE</b>	<b>Wi-Fi</b>	<b>Zigbee</b>
<b>Banda de Frecuencia</b>	2.4 GHz	2.4 GHz / 5 GHz	2.4 GHz
<b>Modulación</b>	GFSK	OFDM, DSSS	DSSS
<b>Rango</b>	<100m	<300m	<100m Punto a Punto
<b>Topología de Red</b>	Scatternet	Star	Mesh
<b>Velocidad</b>	1 Mbps	11 Mbps, 54 Mbps 150 Mbps+	250 kbps
<b>Corriente Pico</b>	<15 mA	60 mA Rx 200 mA Tx	19mA Rx 35mA Tx
<b>Corriente en Espera</b>	<2 $\mu$ A	<100 $\mu$ A	5 $\mu$ A

Tabla 1.2: Comparación de BLE con otros protocolos [Arg14]

Zigbee, Wi-Fi y BLE utilizan la banda ISM de 2,4 GHz, pero son muy diferentes en sus capacidades. El transmisor BLE es claramente un dispositivo de menor alcance que consume menos energía que Zigbee y especialmente Wi-Fi. El consumo de corriente pico más bajo es fundamental en el momento de elegir una batería. Wi-Fi, con un máximo de 200 mA nunca podría operar con una pila de botón. La corriente

pico de BLE, sin embargo, es mucho más baja.

A diferencia de Zigbee, BLE no puede actualmente formar una red de malla, aunque si hay empresas que proveen de implementaciones específicas. Comunicación Punto a Punto implica que los dispositivos tienen que estar en el rango del teléfono inteligente para controlarlo. También hay soluciones utilizando una pasarela inalámbrica que conecta dispositivos BLE a un enrutador, pero estos aumentan el coste final del producto.

### Capa física BLE

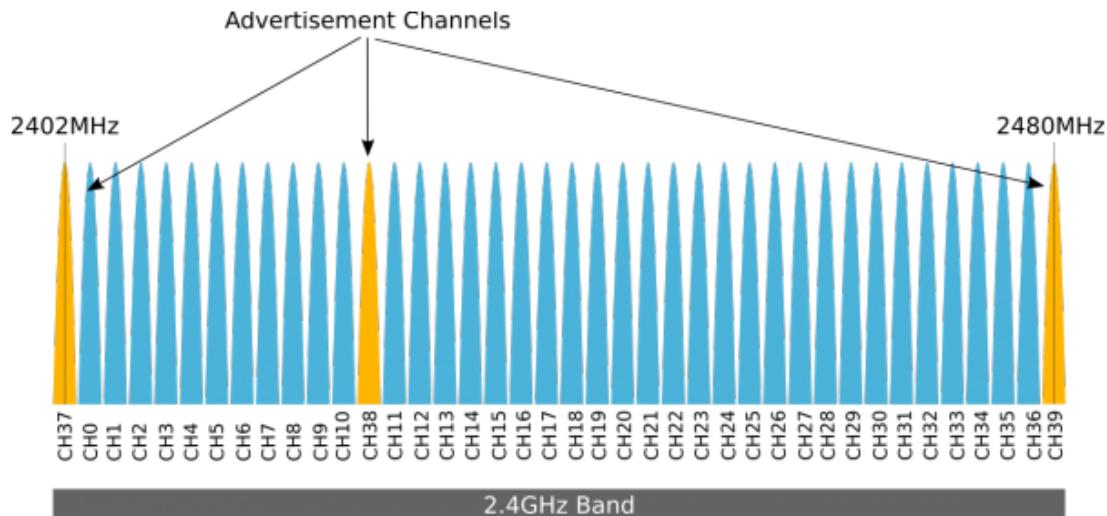


Figura 1.4: Capa Física BLE [Arg14]

La capa física se relaciona directamente con la manera en la que los dispositivos BLE transmiten y reciben datos.

BLE utiliza la misma banda ISM de 2,4 GHz que Bluetooth Clásico, abarcando desde 2400 MHz hasta 2483.5 MHz. Bluetooth v4.0 y especificaciones posteriores dividen la banda en 40 canales. Tal y como muestra la figura 1.4, 3 de estos canales

son llamados “publicidad” y son utilizados por los dispositivos para enviar paquetes de publicidad con información sobre ellos para que otros dispositivos BLE puedan conectarse. Estos canales se seleccionaron en la parte baja, media, y alta de la banda para evitar la interferencia de Wi-Fi. Por ejemplo, si el Canal 38 y canales cercanos están siendo interferidos por Wi-Fi, entonces todavía hay otros 2 canales de publicidad 37 y 39, que no se verán afectados y que BLE puede usar.

### **Capa de Enlace de Datos**

El verdadero funcionamiento de BLE sucede en lo que se llama la capa de Enlace. Esta es la capa que tiene como funciones principales gestionar las conexiones y controlar los paquetes enviados y recibidos. Un dispositivo BLE presenta varios estados posibles, aunque solo uno de ellos es permitido a la vez:

- Espera

El dispositivo no está transmitiendo o recibiendo, si no que está “dormido” para así ahorrar energía.

- Publicidad

Un dispositivo que tiene un papel periférico entrará en el estado de publicidad, donde se enviarán paquetes en los canales de publicidad. En este estado también escuchará las respuestas a los paquetes desde un dispositivo central. Este modo es uno de los más críticos de entender desde una perspectiva de energía debido a que un dispositivo periférico pasará gran parte de su tiempo en el modo de Publicidad (dependiendo de la aplicación). El intervalo de Publicidad pues afecta directamente al consumo y duración de la batería.

- Escaneo

Escaneo se refiere a escuchar los paquetes de publicidad que se envían a través de esos canales. Este modo se utiliza para buscar los dispositivos.

- Iniciador

Este estado es el estado en el que un dispositivo central generalmente entra antes de establecer una conexión. El dispositivo central va a escuchar anuncios en los periféricos, pero una vez que el anuncio de la periférica deseada se recibe en el dispositivo, la central puede conectarse mediante el envío de los datos correctos.

Para el dispositivo esclavo, el estado Publicidad es también el estado inicial antes del estado de conexión. El estado de conexión es el estado final en el que el esclavo (periférico) y maestro (central) pueden intercambiar datos. En BLE, los datos se intercambian periódicamente sobre eventos de conexión.

Los datos se transmiten en los canales de datos, que son los 37 canales no utilizados para la publicidad. Ambos dispositivos se pondrán de acuerdo sobre los canales a usar y alternarán el envío de los datos, siguiendo las normas establecidas por la especificación.

Una aplicación típica BLE implica un chip conectado a diversos sensores. Un usuario con un teléfono inteligente entra en el rango del dispositivo BLE. El sensor con BLE transmite paquetes de publicidad y una vez que el teléfono recibe el anuncio del paquete, comenzará el proceso de conexión para obtener los datos de los sensores.

## 1.2. Android

Android es un sistema operativo que en sus inicios fue concebido para dispositivos móviles con pantalla táctil. Sin embargo, ha evolucionado en una plataforma que actualmente también engloba relojes inteligentes, televisores, automóviles e incluso electrodomésticos.

Está basado en el núcleo de Linux. Sobre este, corre el entorno de ejecución

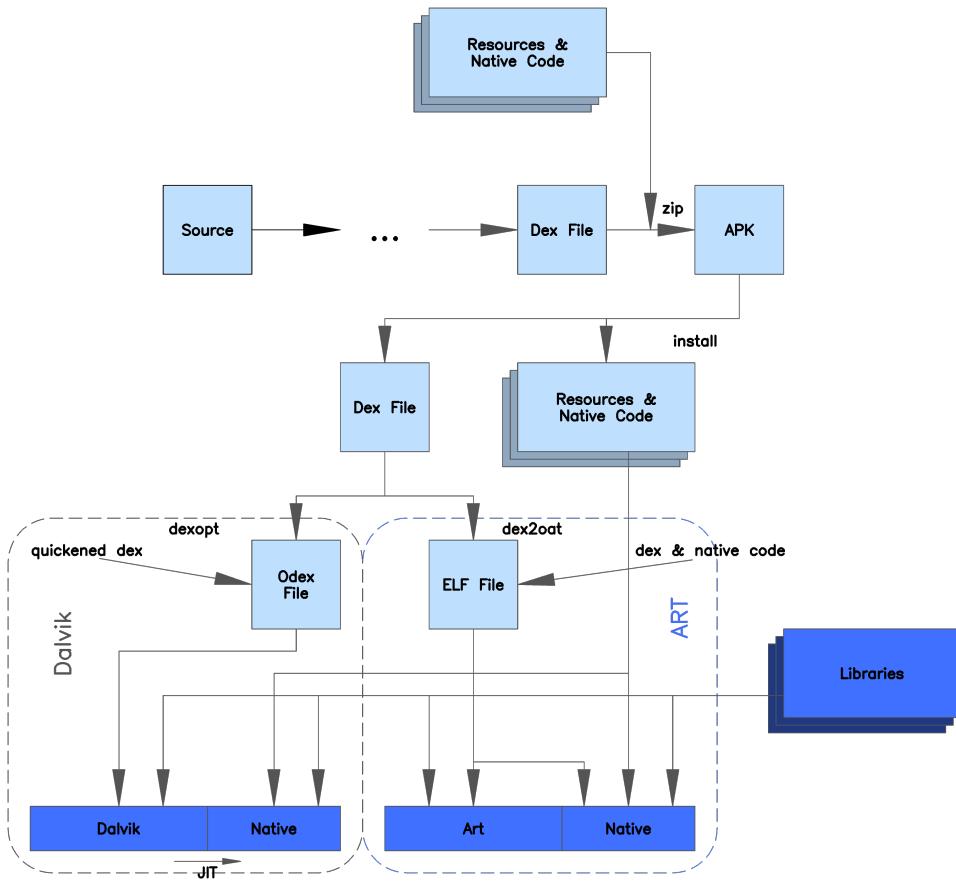


Figura 1.5: Diagrama de la estructura del sistema operativo Android[And08]

propio de Android, que provee una máquina virtual (ya sea Dalvik o ART). Este a su vez ejecuta el código de las aplicaciones, escritas mayoritariamente en Java, aunque se permiten extensiones en C/C++. Las aplicaciones son precompiladas a un código intermedio denominado Dex y este es empaquetado y comprimido en el archivo estándar de aplicación Android, el APK.

Cada aplicación instalada en el dispositivo, es optimizada según los recursos y tipo de máquina virtual disponibles en el dispositivo en concreto. Al ser iniciada, es ejecutada dentro de un “cajón de arena”, técnica que aísla los procesos de cada aplicación. Esto resulta en una mayor robustez y seguridad del sistema, ya que el

fallo de una aplicación no afecta a los recursos de las demás. Tampoco podría una aplicación maligna, si se diera el caso, modificar o apropiarse de los recursos de otras. En la figura 1.5 se observa la relación entre todos los elementos anteriormente descritos.

### 1.2.1. Interfaz

La interfaz de usuario por defecto de Android está basada en una manipulación directa, usando entrada táctil con gestos que vagamente corresponden a acciones físicas reales, tales como deslizar, golpear, pellizcar, etcétera. Para manipular objetos en pantalla. Además, la mayoría de dispositivos dispone de un teclado virtual, manipulado de la misma manera.

La respuesta del sistema a la entrada del usuario está diseñada para ser inmediata y dar una sensación de fluidez, utilizando las capacidades de vibración presentes en la mayoría de los dispositivos para proveer una respuesta háptica (no visual, no auditiva). Adicionalmente, algunas aplicaciones utilizan la información proveída por sensores tales como acelerómetros, giroscopios y sensores de proximidad para responder a interacciones adicionales, como por ejemplo ajustar la orientación de la pantalla cuando el dispositivo se encuentra apaisado o controlar alguna parte de la aplicación basándose en el azimut relativo del dispositivo.

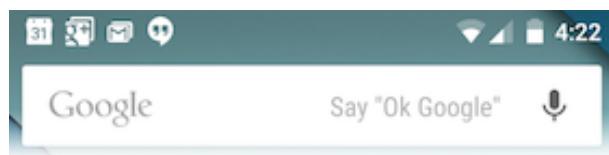


Figura 1.6: Notificaciones minimizadas en el área de notificaciones[And08]

Una parte importante de la interfaz general de Android son las notificaciones, presentes en la barra de estado. Una notificación es un mensaje que una aplicación muestra al usuario fuera de los límites habituales de la interfaz de usuario de la

aplicación. Las aplicaciones solicitan al sistema que emita una notificación por ellas, proveyendo el contenido, y el sistema operativo es quien maneja el resto del ciclo de vida de la notificación. En la figura 1.6 se pueden observar cuatro notificaciones minimizadas en el área de notificaciones

### 1.2.2. Componentes de una aplicación

#### Contexto

Uno de los conceptos más importantes cuando se utiliza la plataforma Android es el contexto, **Context**. La clase **Context** en si misma no es más que una interfaz a información global acerca del entorno de una aplicación, y como tal es abstracta. Sin embargo, es importante ser consciente de qué elementos representan un contexto válido y qué elementos no. Un contexto permite acceso a recursos y clases específicos de la aplicación, y llamadas al sistema para operaciones a nivel de aplicación. Por ejemplo lanzar actividades, emitir mensajes de difusión o recibirllos.

#### Actividades

En Android, una actividad (**Activity**) representa una única cosa concreta que el usuario puede realizar en la aplicación. La mayoría de las actividades interaccionan con el usuario, por tanto la clase **Activity** se encarga de crear una ventana donde se puedan insertar los componentes de la interfaz de usuario. Aunque las actividades suelen ser vistas por el usuario como ventanas a pantalla completa, también pueden ser usadas en otras múltiples maneras, ya sea como ventanas flotantes, o incrustadas dentro de otra actividad (mediante un **ActivityGroup**)

## Ciclo de vida de una actividad

Las distintas actividades de las distintas aplicaciones instaladas en el dispositivo Android son gestionadas en forma de una “pila de actividades”.

Cuando el sistema Android se inicia, se presenta al usuario una pantalla principal, desde donde puede lanzar varias acciones y aplicaciones. A partir de ahí, cuando una nueva actividad es empezada, se emplaza arriba de la pila y se convierte en la actividad en ejecución. Las actividades previas, si las hubiera, siempre permanecen por debajo en la pila, y no se traerán al frente hasta que la nueva actividad finalice.

Una actividad tiene cuatro estados básicos:

### Activa

Una actividad está *activa* cuando está presente en primer plano en la pantalla, es decir, arriba de la pila. También se puede decir que la actividad está *En ejecución*.

### Pausada

Si una actividad ha perdido el foco (ha dejado de estar en primer plano) pero todavía es visible, es decir, si una nueva actividad que no ocupa la totalidad de la pantalla o es transparente obtiene el primer plano, se encuentra *pausada*. Una actividad pausada se conserva completamente íntegra (mantiene todos los estados y se mantiene suscrita al gestor de ventanas) pero puede ser matada por el sistema en condiciones extremas de baja memoria disponible.

### Parada

Si una actividad se encuentra oculta por completo, el sistema la deja *parada*. Mantiene estados e información de los miembros, pero sin embargo al no ser visible por el usuario es más probable que el sistema se deshaga de ella para liberar recursos cuando hagan falta.

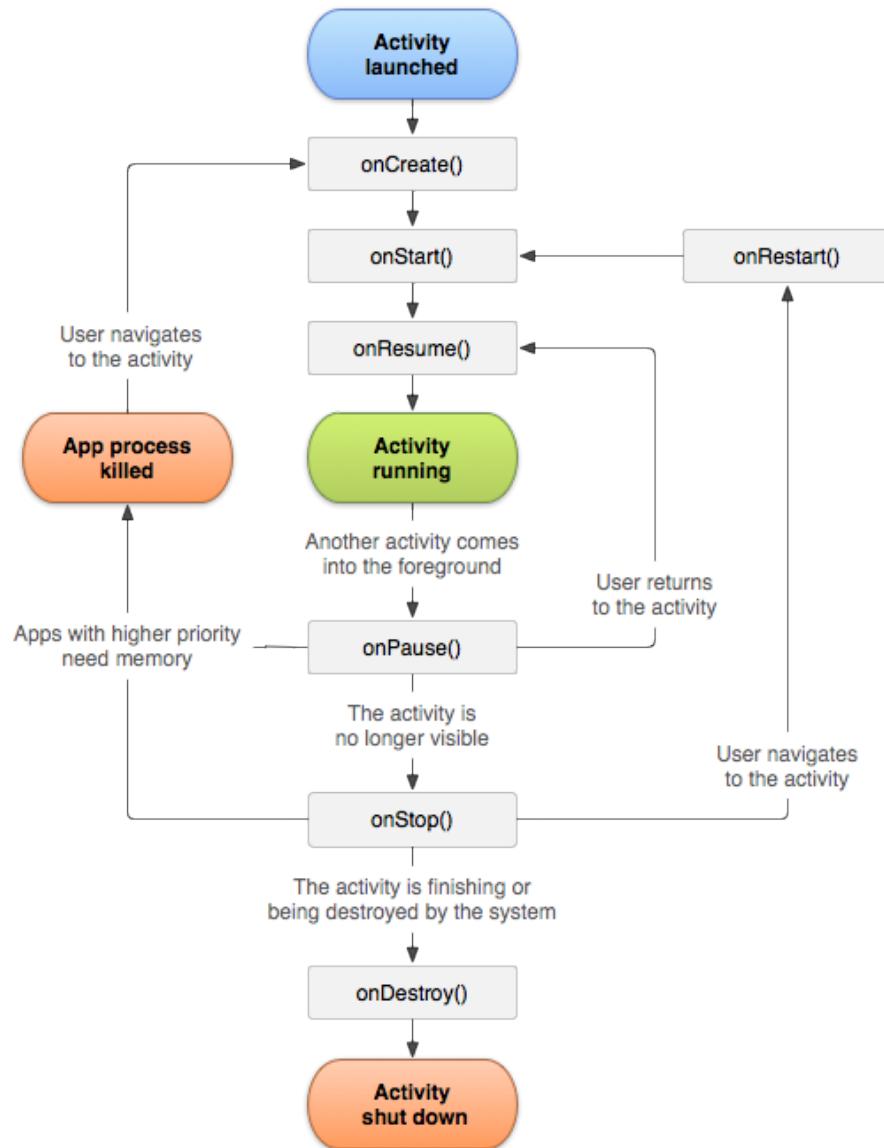


Figura 1.7: Diagrama del ciclo de vida de una actividad en Android [And08].

## Muerta

Cuando el sistema decide mover la actividad fuera de memoria, puede o bien finalizarla o matar el proceso. Cuando sea mostrada de nuevo al usuario, debe ser completamente reiniciada y restaurada a su estado previo.

Los cuatro estados y las acciones que llevan a una aplicación de un estado a otro, junto con los métodos de la actividad que son invocados en cada etapa, están representados en la figura 1.7.

## Servicios

Un Servicio (**Service**) es un componente de la aplicación que puede realizar tareas de larga duración en segundo plano y no provee ninguna interfaz de usuario. Otro componente de la aplicación puede iniciar un servicio y este continuará en marcha en segundo plano, incluso si el usuario cambia a otra aplicación distinta. Además, un componente puede adherirse (**bind**) a un servicio para interactuar con él, e incluso realizar comunicación inter-procesos (IPC por sus siglas en inglés, Inter-Process Communication). Por ejemplo, un servicio puede manejar llamadas de red, reproducir música, realizar operaciones en el sistema de archivos, todo en segundo plano.

Un servicio puede tomar dos estados:

## Started

Un servicio está en estado **Started** (empezado) cuando un componente de la aplicación, por ejemplo una actividad, lo empieza llamando al método `startService()`. Una vez empezado de esta manera, un servicio puede continuar en segundo plano de manera indefinida, incluso si el componente que lo empezó ha sido destruido. Normalmente, suelen ser servicios que realizan una única operación y no devuelven ningún resultado. Por ejemplo, puede descar-

gar o subir un archivo en la red. Cuando la operación ha sido completada, el servicio debe pararse a si mismo.

### Bound

Un servicio está en estado **Bound** (adherido) cuando un componente de la aplicación se adhiere a él llamando al método `bindService()`. Un servicio adherido ofrece una interfaz servidor-cliente que permite a los componentes interaccionar con el servicio, mandar peticiones, obtener resultados, e incluso hacerlo entre distintos procesos mediante comunicación inter-proceso (IPC). Un servicio adherido solamente es activo durante el tiempo que otro componente esté adherido a él. Varios componentes pueden estar adheridos en un momento dado al servicio, pero cuando todos se desadhieren del servicio, el servicio es destruido.

Es necesario tener cuidado, dado que un servicio corre en el hilo principal de ejecución del proceso que lo llama, a no ser que se especifique lo contrario. Esto implica que, si el servicio va a realizar alguna tarea intensiva en CPU, o alguna operación bloqueadora, se debe de crear un nuevo hilo de ejecución dentro del servicio para ese propósito. De no hacerlo, se corre el riesgo de que la aplicación deje de responder y sea matada por el sistema operativo.

Los dos tipos de ciclo de vida de un servicio, sus etapas y los métodos invocados en el transcurso de ellas están representados en la figura 1.8.

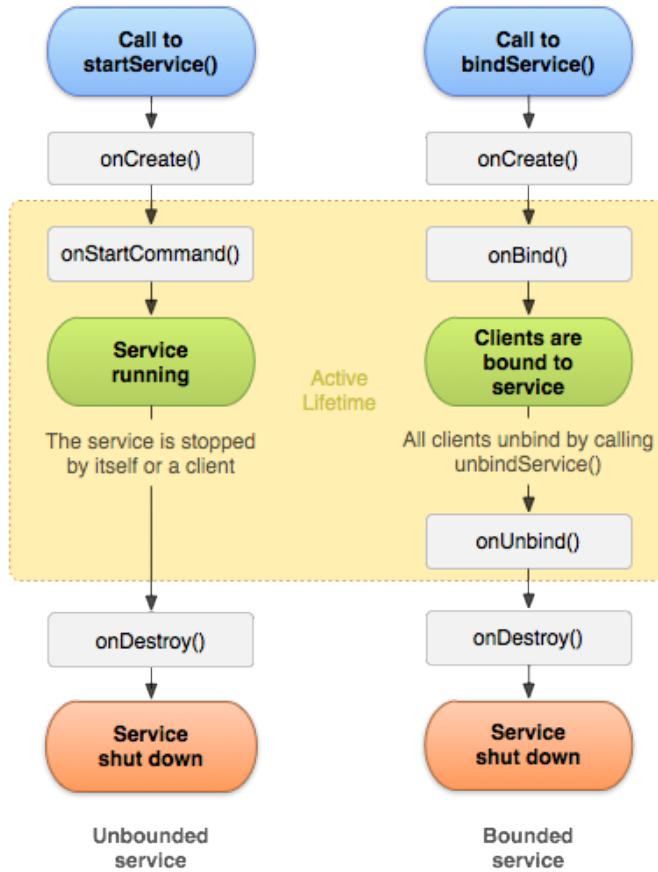


Figura 1.8: Diagrama del ciclo de vida de un servicio [And08]

### 1.2.3. Vistas

La interfaz gráfica de usuario en una aplicación Android está construida usando una jerarquía de vistas, objetos de la clase `View`, y grupos de vistas, objetos de la clase `ViewGroup`. Los objetos `View` suelen ser artílugos (widgets) de la interfaz de usuario, tales como botones o campos de texto, y los objetos `ViewGroup` son contenedores invisibles que definen cómo se posicionan las vistas que dependen de ellos, por ejemplo dispuestas en forma de rejilla o lista vertical.

Android provee un vocabulario XML que se corresponde con las subclases de

`View` y `ViewGroup`, y permite definir la interfaz de usuario en XML usando una jerarquía de elementos de interfaz de usuario.

#### 1.2.4. Procesos e hilos de ejecución

En sistemas operativos, son básicos los conceptos de proceso e hilo de ejecución. En Android, el sistema operativo comienza un nuevo proceso Linux por cada primer componente de cada aplicación, con un único hilo de ejecución. Por defecto, todos los componentes de la misma aplicación corren en los mismos proceso e hilo de ejecución, el hilo principal de ejecución (*main thread* en inglés).

En caso de que un componente de una aplicación sea inicializado, y ya exista un proceso para dicha aplicación en el sistema que otro componente de la misma aplicación ha inicializado, el nuevo componente se inicializa dentro del proceso original y usa el mismo hilo de ejecución. Sin embargo, se puede configurar una aplicación de manera que diferentes componentes corran en procesos separados, y siempre se pueden crear hilos de ejecución adicionales para cualquier proceso.

#### Procesos

Como ya ha sido expuesto anteriormente, por defecto todos los componentes de la misma aplicación corren en el mismo proceso, y la mayoría de las aplicaciones no deberían de cambiarlo. Sin embargo, es posible controlar qué proceso pertenece a qué componente de ser necesario.

El sistema operativo puede decidir apagar un proceso, cuando la cantidad de memoria disponible sea baja y haya requerimiento de ella por otro proceso que sirva de manera más inmediata al usuario. En este caso, los componentes dentro de dicho proceso que es apagado, son destruidos. Cuando estos componentes sean necesarios de nuevo, un nuevo proceso será comenzado por el sistema operativo para ello.

## Hilos de ejecución

Previamente se ha mencionado que todos los componentes de la misma aplicación corren en el mismo hilo de ejecución, el hilo principal de ejecución o *main thread*. Este hilo es de suma importancia, dado que carga con la responsabilidad de despachar los eventos al widget de la interfaz de usuario que sea pertinente, incluyendo los eventos de dibujado en pantalla. Es también el hilo de ejecución en el que la aplicación interactúa con los componentes básicos de interfaz de usuario de Android, también conocidos como *Android UI toolkit* (ubicados dentro de los paquetes java `android.widget` y `android.view`). Por todo esto, no es extraño encontrar denominado este hilo de ejecución como el hilo de la interfaz de usuario, o *UI Thread*.

Dado que todos los componentes que corren en el mismo proceso son instanciados en el hilo de ejecución principal, las llamadas del sistema operativo a cada componente son despachadas desde dicho hilo. En consecuencia, todos los métodos que responden a retrolllamadas del sistema (*system callbacks*), como por ejemplo para indicar que una tecla ha sido pulsada, siempre corren en el hilo principal de ejecución del proceso.

Cuando el usuario toca un botón en la pantalla, el hilo principal de la aplicación despacha el evento de toque al widget pertinente, que reacciona cambiando su estado a presionado y manda una petición de invalidación a la cola de eventos. El hilo de ejecución principal entonces desencola la petición y notifica al widget que debe de re-dibujarse.

Cuando una aplicación realiza trabajo intensivo en respuesta a una interacción del usuario, el modelo de hilo de ejecución único puede resultar en una falta de rendimiento. Concretamente, de suceder todo el procesamiento en el hilo principal de ejecución e iniciar tareas de larga ejecución tales como acceso a red o consultas a bases de datos, resultará en un bloqueo completo de la interfaz de usuario y

su correspondiente hilo principal. Cuando el hilo de ejecución está bloqueado, no se pueden despachar eventos, eventos de dibujado en pantalla incluidos. Desde el punto de vista del usuario, esto se traduce en una aplicación que parece colgarse. En peores casos, en los que el hilo principal de ejecución está bloqueado por más de unos cuantos segundos (cinco segundos en la actualidad) el sistema operativo entrará en acción y mostrará una pantalla explicando que la aplicación ha dejado de responder, y matará la aplicación bloqueada.

Para evitar dicha penalización en el rendimiento, deben usarse hilos de ejecución alternativos para toda tarea que bloquee la ejecución o sea de alta carga procedural.

### 1.2.5. Ubicación

#### GPS

El Global Positioning System, o sistema de posicionamiento global (GPS) es un sistema de navegación por satélite que provee información sobre la posición y el reloj del dispositivo de recepción. El sistema fue desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos. Está constituido por 24 satélites en órbita geosíncrona y se basa en un sistema de triangulación para determinar posiciones con precisión de metros.

El funcionamiento del GPS necesita la recepción de la señal de al menos cuatro de los 24 satélites en órbita. En base a dichas señales, que incluyen la identificación del satélite emisor y la hora de reloj de emisión, el aparato receptor sincroniza su reloj interno y calcula el tiempo que tardan en llegar las señales al equipo. Con dicha información, el dispositivo es capaz de conocer su distancia a cada uno de los satélites cuya señal es recibida, y por ende la posición absoluta del punto de medición.

### Servicios Móviles de Google

Conocer la localización GPS del dispositivo móvil es crítico en el funcionamiento de la aplicación. Es posible en un dispositivo Android utilizar el receptor GPS a bajo nivel y recibir actualizaciones de posición en formato NMEA. Sin embargo, no se aprovecharían cantidad sustancial de optimizaciones que Google ofrece como parte de Google Mobile Services, o servicios móviles de Google (GMS) tales como tiempo de respuesta mejorado, localización mixta y consumo de batería mejorado.

GMS es un servicio en segundo plano propietario de Google presente en todos los dispositivos Android que satisfagan las condiciones de entrada de la empresa. En este proyecto, la aplicación se conectará a dicho servicio para requerir actualizaciones de posición.

#### 1.2.6. BLE en Android

Android 4.3 (Nivel API 18) introduce una función de soporte de la plataforma para Bluetooth de baja energía y proporciona una API que las aplicaciones pueden utilizar para detectar dispositivos BLE, consultar servicios y realizar operaciones de lectura o escritura. En contraste con el Bluetooth clásico, BLE está diseñado para proporcionar un consumo de energía significativamente menor. Esto permite que aplicaciones Android puedan comunicarse con dispositivos BLE que tienen requisitos de baja potencia, tales como sensores de proximidad, pulsómetros, sensores publicitarios y otros muchos más.

#### Conceptos y Terminología clave

A continuación presentamos un resumen de los conceptos y terminología que son de vital importancia cuando hablamos del protocolo BLE:

- Generic Attribute Profile, o Perfil de Atributo Genérico (GATT)

El GATT es una especificación general para el envío y recepción de pequeñas cantidades de datos conocidos como “atributos” a través de un enlace BLE. Todos los perfiles actuales de aplicaciones de baja energía se basan en el GATT. El SIG de Bluetooth define muchos perfiles para dispositivos de baja energía. Un perfil es una especificación para el funcionamiento de un dispositivo en una aplicación particular, teniendo en cuenta que un dispositivo puede aplicar más de un perfil. Por ejemplo, un dispositivo podría contener un monitor de ritmo cardíaco y un detector de nivel de batería.

- Attribute Protocol, o protocolo de atributo (ATT)

GATT se define encima del protocolo ATT. El conjunto también se conoce como el protocolo GATT/ATT. ATT está optimizado para funcionar en dispositivos BLE. Con este fin, se utiliza el menor número de bytes posible. Cada atributo está identificado por un identificador único universal (UUID), que es una cadena de caracteres representando un identificador con un formato de 128 bits normalizado, para así identificar de forma única la información. Los atributos transportados por ATT reciben el nombre de características y servicios.

- Característica

Una característica contiene un único valor y 0-n descriptores que describen el valor de dicha característica. Una característica puede ser interpretada como un tipo, análogo a una clase en Programación Orientada a Objetos (POO)

- Descriptor

Los descriptores son atributos definidos que describen el valor de una característica. Por ejemplo, un descriptor podría especificar una descripción legible por el ser humano, un rango aceptado para el valor de una característica, o una unidad de medida que es específica a un valor de una característica deter-

minada.

- Servicio

Un servicio es una colección de características. Por ejemplo, un usuario podría disponer de un servicio llamado *Monitor de Frecuencia Cardíaca*, el cual incluye características tales como *Medida del ritmo cardíaco* o *Posición del dispositivo medidor en el cuerpo humano*, entre otras.

## Roles y Responsabilidades

Los roles y responsabilidades aplicables cuando un dispositivo Android interactúa con un dispositivo BLE son los siguientes:

- Central vs. periférica. Esto se aplica a la propia conexión BLE. El dispositivo con el rol o papel central escanea, en busca de publicidad, y el dispositivo con el rol periférico realiza el anuncio.
- Servidor GATT vs. cliente GATT. Esto determina cómo se realiza la comunicación entre los dos dispositivos una vez que la conexión ha sido establecida.

Para entender la distinción, podemos imaginar que tenemos por un lado un teléfono Android y un rastreador de actividad como dispositivo BLE por otro. El teléfono es compatible con el papel central mientras que el rastreador de actividad se apoya en el papel periférico (para establecer una conexión BLE se necesitan dos dispositivos que presenten roles opuestos, ya que dos dispositivos que solo soporten modo periférico no podrían comunicarse entre sí, al igual que dos dispositivos que solo soportasen modo central).

Una vez que el teléfono y el rastreador de actividad han establecido una conexión, comienzan a transferir meta-datos GATT entre sí. Dependiendo del tipo de datos que se transfieren, el uno o el otro puede actuar como servidor. Por ejemplo, si el rastreador de actividad quiere reportar datos del sensor al teléfono, tendría sentido que

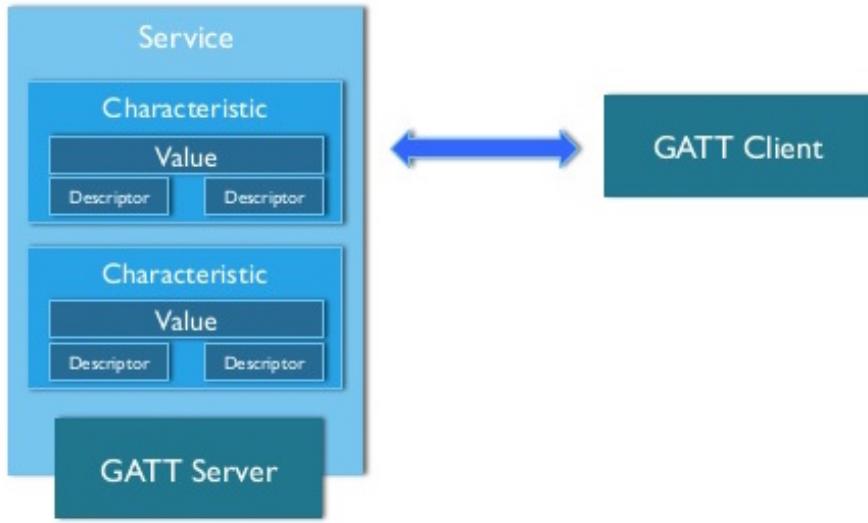


Figura 1.9: Esquema Servidor y Cliente GATT [And08]

el rastreador de actividad actuase como servidor. Si el rastreador de actividad quiere recibir actualizaciones desde el teléfono, entonces sería más conveniente designar al teléfono como el servidor, actuando el rastreador como cliente.

En nuestro caso particular, la aplicación Android (que se ejecuta en un dispositivo Android) es el cliente GATT. La aplicación recibe datos desde el servidor GATT, que es un pulsómetro comercial, el cual soporta el perfil BLE de ritmo cardíaco. La figura 1.9 muestra dicho esquema cliente-servidor.

## 1.3. Desarrollo de aplicaciones web modernas

### 1.3.1. Definiciones básicas

#### Frontend

El Frontend es un término muy utilizado en el desarrollo web actual, y podríamos resumirlo a que se refiere a la parte que el usuario común ve, es decir, la interfaz

gráfica, aunque no se queda solo en eso, sino que dentro del Frontend podríamos decir que está el Javascript que se ejecuta del lado del cliente. Los lenguajes y tecnologías que componen el frontend son:

- HyperText Markup Language, o lenguaje de marcas de hipertexto (HTML)  
Define el contenido de una página web, como texto, imágenes y videos entre otros. Es un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación.
- Cascading Style Sheets, o hoja de estilo en cascada (CSS)  
Es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.
- Javascript  
Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. En este caso nos referimos a Javascript del lado del cliente, implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

## Backend

Es lo que se ejecuta del lado del servidor. Los elementos del backend son los siguientes:

- Lógica de negocio.
- Lenguaje de servidor (PHP, Ruby, Python, Javascript).
- Bases de Datos

## Stack

Un Stack, podríamos decir que es un conjunto de paquetes configurado de cierta manera que se complementan entre sí. El ejemplo de un stack bastante conocido es LAMP, que es el conjunto de Linux, Apache, MySQL y PHP ejecutándose juntos como una plataforma para desarrollo web, o MEAN, el cual es bastante reciente y es usado en este proyecto, agrupando las tecnologías MongoDB, Express, AngularJS y Node.

## DOM

Document Object Model, o modelo de objetos del documento (DOM) es la estructura de objetos interna que genera el navegador cuando se carga un documento HTML y la cual se puede alterar mediante Javascript para cambiar de forma dinámica los contenidos y aspecto de la página. Es una estructura jerárquica donde existen varios objetos y unos dependen de otros.

Los objetos del DOM modelan tanto la ventana del navegador como el historial, el documento o página web, y todos los elementos que pueda tener dentro la propia página, como párrafos, divisiones, tablas, formularios y sus campos, etcétera. A través del DOM se puede acceder, por medio de Javascript, a cualquiera de estos elementos, es decir, a sus correspondientes objetos para alterar sus propiedades o invocar a sus métodos. Con todo, a través del DOM, queda disponible para los programadores de Javascript cualquier elemento de la página, para modificarlos, suprimirlos, crear nuevos elementos y colocarlos en la página, etcétera.

## Web Socket

Especificación web que permite establecer una conexión persistente entre el cliente y el servidor, autorizando a ambas partes a enviar datos en cualquier momento.

### 1.3.2. MEAN

MEAN es un *stack* para el desarrollo de aplicaciones web, el cual usa JavaScript como lenguaje de programación tanto para el frontend como para el backend, gracias a sus herramientas MongoDB, Express, AngularJS y NodeJS. Este es el motivo por el cual se denomina un *full stack*. Podríamos considerarlo una alternativa a LAMP, aunque de cierta manera un poco lejana, ya que en LAMP utilizamos tecnologías distintas en cada cosa que queremos hacer.

	LAMP/XAMPP	MEAN
<b>Sistema Operativo</b>	Linux, Windows, Mac OS X, etc...	Linux, Windows, Mac OS X, etc...
<b>Servidor HTTP</b>	Apache	Node.js
<b>Base de Datos</b>	MySQL	MongoDB
<b>Web Framework</b>	PHP	Express
<b>Frontend Framework</b>	-	AngularJS

Tabla 1.3: Comparación básica de LAMP y MEAN [Pen14]

	LAMP	MEAN
<b>Base</b>	Bash	Javascript
<b>Servidor HTTP</b>	Apache Config	Javascript
<b>Base de Datos</b>	SQL	Javascript
<b>Lenguaje de Programación</b>	PHP	Javascript

Tabla 1.4: Comparación entre lenguajes utilizados en LAMP y MEAN [Pen14]

Como se puede observar en las tablas 1.3 y 1.4 , lo único que comparten ambas pilas de desarrollo es la finalidad para la que se utilizan: desarrollo web. Otro de

los puntos a revisar es el hecho de que LAMP no cuenta en sí con una librería JavaScript para el frontend, aunque podría usarse cualquier librería sin problema, como por ejemplo AngularJS o jQuery.

Por tanto, podemos decir que MEAN es un *full stack* de desarrollo web con el que se pretende trabajar todo el tiempo con JavaScript. Este hecho se apoya en el gran progreso de optimización del lenguaje, llegando incluso a ser más rápido que otros lenguajes utilizados del lado del servidor, como PHP, Ruby y Python.

A continuación vamos a analizar en detalle cada una de las herramientas y tecnologías que conforman el stack MEAN.

### 1.3.3. MongoDB

Sin duda alguna, MySQL es la base de datos más popular del mercado, aunque recientemente ha experimentado un bajón en su uso en favor de otros sistemas de gestión de bases de datos más abiertos, como MariaDB o PostgreSQL. Lo que tienen en común estos sistemas, es que son considerados “Sistemas de Bases de Datos Relacionales” y cumplen muy bien con su cometido, que es guardar información y recuperarla en cualquier momento con un alta disponibilidad. También tienen en común que para comunicarse con ellas necesitaremos dominar el lenguaje SQL.

El problema que presentan estas bases de datos son la velocidad de lectura cuando se tiene una gran demanda de datos y un crecimiento exponencial de la carga. Un ejemplo sencillo es el reflejado por las redes sociales, las cuales reciben miles de consultas por segundo que deben ser servidas de forma eficiente y siempre cuidando que, si hay algún error en uno de sus servidores, dicha información se encuentre disponible.

Esta es una de las principales razones del nacimiento de las bases de datos NoSQL, que tal y como su nombre indica, no son bases de datos que se manejen con SQL y proveen una alta disponibilidad y eficiencia. Algunos ejemplos de

estas bases de datos son MongoDB, HyperTable, Cassandra, CouchDB, etcétera.

Sin embargo, a pesar de que existen muchas alternativas de bases de datos NoSQL, podemos decir que pocas tienen la madurez, seguridad y velocidad necesarias para entrar en entornos de producción de alta exigencia. MongoDB es una de las pocas que puede hacer gala de ello, ya que presenta funciones bastante interesantes que la llevan a ser la base de datos favorita para muchas personas. A continuación destacamos los puntos clave:

- Desarrollada en C++

C++ es un lenguaje de bajo nivel, lo que promete que alcanzará un gran rendimiento a la hora de exigirle. Los documentos son almacenados en estilo JavaScript Object Notation, o notación de objetos de JavaScript (JSON), por lo que no existen tablas, ni filas, ni columnas. Permite el uso de JavaScript para su manipulación.

- Replicación

La base de datos incluye por defecto un sistema de replicación. Si introducimos un nuevo documento en nuestra base de datos, esta automáticamente lo replicará en otra base de datos en un servidor distinto. De esta manera nos aseguramos que nuestra información permanece segura y siempre disponible.

- GridFS

Nos permite guardar documentos de cualquier tamaño sin afectar al rendimiento de la base de datos.

#### 1.3.4. Express

Express es un framework de desarrollo de aplicaciones web minimalista y flexible para Node. Si bien la base sobre la que estaremos trabajando será Node, el sistema que nos ofrece por defecto para servir páginas es un poco complejo. Express nos

abstactae de estas capas de bajo nivel, permitiéndonos crear un servidor web en muy pocos pasos.

Compete con rivales bastante fuertes en el mercado, como Meteor o Restify. Sin embargo, ninguno de ellos tiene la comunidad que Express tiene detrás brindando soporte. Prácticamente cualquier duda técnica que surja, podemos encontrar la solución en foros dedicados al soporte informático como StackOverflow o Quora. En cambio, otros frameworks, a pesar de ser muy buenos, no cuentan con esta ventaja.

Algunas de las características que Express nos ofrece son las siguientes:

- Sencillez
- Comunidad bastante activa
- Documentación de alta calidad
- Compatibilidad con otros módulos
- Alto Rendimiento, focalizándose en el enrutamiento

### 1.3.5. AngularJS

AngularJS es un framework de JavaScript de código abierto, mantenido por Google, que ayuda con la gestión de lo que se conoce como aplicaciones de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

Algunas características que vienen acompañadas con AngularJS son:

- Comunidad

Al igual que Express, la comunidad es uno de sus puntos fuertes.

- Herramientas

AngularJS proporciona numerosas herramientas que nos brindan facilidades con llamadas Asynchronous Javascript And XML, o JavaScript asíncrono y XML (AJAX), bucles, sistemas para clonado de objetos o arrays, sistema de monitorización de variables, etcétera.

- Data-Binding

Se refiere a que, si desde el backend cambia de alguna manera el valor de una variable (y lo actualizamos en nuestro controlador), cambiará también en el frontend. Si en el frontend hay algún cambio, el valor de la variable automáticamente se actualizará sin tener que hacer llamadas extras para detectar dicho cambio.

- MVC

La forma en la que está estructurado AngularJS prácticamente nos obliga a tener nuestro código ordenado, utilizando controladores para mandar datos a la vista, además de modelos, directivas, servicios y filtros, entre otros.

- Manejador de rutas

AngularJS incluye por defecto un manejador de rutas para implementar aplicaciones de una sola página, *single page web applications*, es decir, compactar en una sola página todo el HTML y manejar toda la información mediante rutas y controladores, únicamente cargando lo que nos interesa y no toda la página. Con esto utilizamos menos recursos del servidor y mejoramos el rendimiento desde el punto de vista del cliente con menos tiempo de carga y mejores respuestas.

Aunque profundizaremos a nivel técnico más adelante en el apartado 2.3.5, se ha incluido en la figura 1.10 un ejemplo sencillo de la sintaxis utilizada en AngularJS.

Esta puede parecer un poco extraña al principio, pero que una vez que uno se acostumbra, es muy intuitiva y fácil de manejar.

```
<html ng-app="angularApp">
...
<body ng-controller="homeCtrl">
    <h1>Hola mundo</h1>
    <table>
        <tr ng-repeat="resultado in resultados">
            <td>{{resultado.valor}}</td>
        </tr>
    </table>
</body>
</html>
```

Figura 1.10: Ejemplo de directivas AngularJS incrustadas en HTML [Pen14]

### 1.3.6. Node.js

Node es un intérprete Javascript del lado del servidor que cambia la noción de cómo debería trabajar un servidor. Su meta es permitir a un programador construir aplicaciones altamente escalables y escribir código que maneje decenas de miles de conexiones simultáneas en una sola máquina física.

#### ¿Qué problema resuelve Node?

La meta número uno declarada de Node es proporcionar una manera fácil para construir programas de red escalables. En lenguajes como Java y PHP, cada conexión genera un nuevo hilo que potencialmente viene acompañado de 2 MB de memoria. En un sistema de 8 GB de Random Access Memory, o memoria de acceso aleatorio (RAM), el número teórico de conexiones concurrentes soportadas daría servicio como

máximo a 4.000 usuarios al mismo tiempo. A medida que crece la base de clientes, si deseamos que nuestra aplicación soporte más usuarios, necesitaremos agregar más y más servidores.

Esto dispara el coste de servidor del negocio y el coste de tráfico, entre otros. Además de estos, están los costes causados por problemas técnicos potenciales. Por ejemplo, un usuario podría estar usando diferentes servidores para cada solicitud, así que cualquier recurso compartido debería almacenarse en todos los servidores. Por todas estas razones, el cuello de botella en toda la arquitectura de aplicación web (incluyendo el rendimiento del tráfico, la velocidad del procesador y la velocidad de la memoria) era el número máximo de conexiones concurrentes que podía manejar un servidor.

Node resuelve este problema cambiando la forma en que se realiza una conexión con el servidor. En lugar de generar un nuevo hilo en el sistema operativo para cada conexión (y de asignarle la memoria correspondiente), cada conexión dispara una ejecución de un evento dentro del proceso central del motor de Node. Node también asegura que nunca se quedará en punto muerto, puesto que no se permiten bloqueos y porque por naturaleza no se bloquea directamente para llamadas de entrada y salida. Con esto nos garantiza que un servidor que lo ejecute podrá soportar decenas de miles de conexiones concurrentes. La figura 1.11 resume el sistema de eventos de Node.

## Funcionamiento

Node ejecuta V8 JavaScript, es decir JavaScript del lado del servidor. El motor V8 JavaScript es el motor JavaScript subyacente que Google usa con su navegador Chrome. Un motor JavaScript interpreta el código y lo ejecuta. Con el V8, Google creó un intérprete ultra-rápido escrito en C++ con otro aspecto único: cualquiera puede descargar el motor e incorporarlo a cualquier aplicación que desee. Gracias

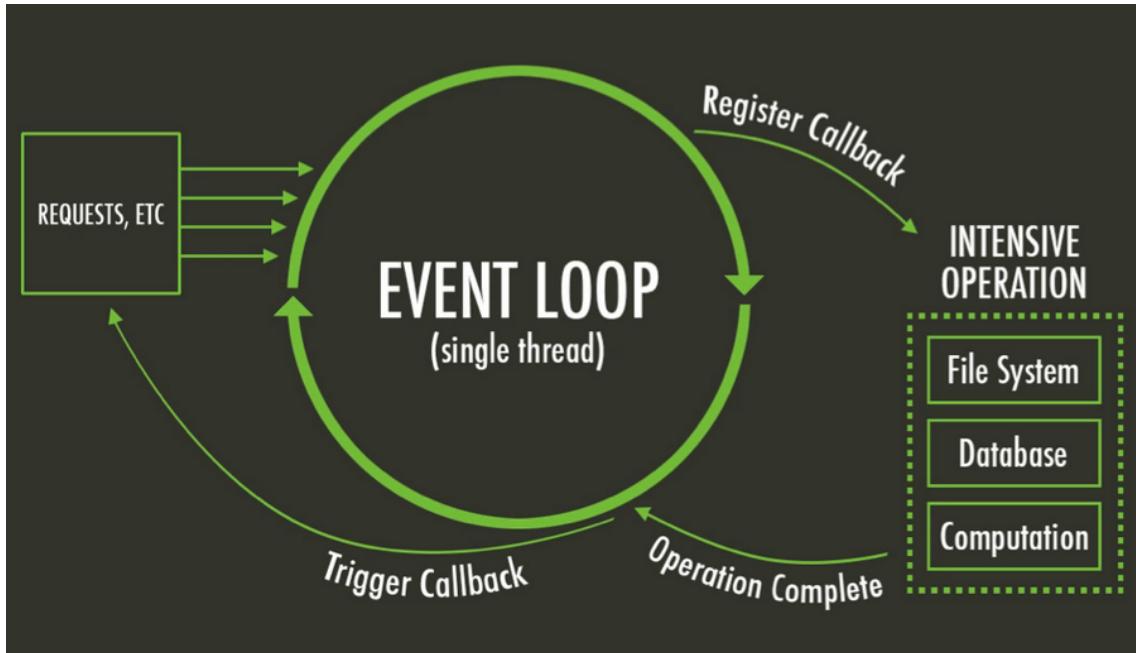


Figura 1.11: Sistema de eventos en NodeJS [Dah09]

a que este no está restringido a ejecutarse en un navegador, Node aprovechó esta faceta y le dio otro propósito para así usarlo en el servidor.

### Programación orientada a eventos

Node utiliza lo que se conoce como modelo de programación orientado por eventos, a diferencia de la programación orientada a objetos que utilizan lenguajes comúnmente usados como Java. El lado del servidor realmente no es tan diferente del lado del cliente. Es cierto que el usuario no está interactuando con una página web en concreto, ya sea presionando botones o ingresando texto en campos, pero a un nivel superior, están sucediendo eventos. Ejemplos de estos pueden ser una conexión establecida, la recepción de datos a través de dicha conexión, o el hecho de dejar de recibir datos por esa conexión. ¿Por qué este tipo de configuración es ideal para Node? JavaScript es un gran lenguaje para la programación orientada a eventos, ya

que permite funciones y cierres anónimos, y más importante, la sintaxis es similar para casi cualquier persona que haya programado en Java o C++. Las funciones de retrollamada o *callbacks* que se llaman y ejecutan cuando ocurre un evento pueden escribirse en el mismo punto en el que el evento es capturado, con lo cual facilita la programación y el mantenimiento de estas. No existen arquitecturas complicadas orientadas a objetos ni tampoco interfaces, simplemente esperar por un evento y escribir una retrollamada qué contendrá las tareas a realizar cuando dicho evento ocurra.

## Módulos Node

En NodeJS el código se organiza por medio de módulos, que son como los paquetes o librerías de otros lenguajes como Java. Por su parte, NPM es el nombre del gestor de paquetes que usamos en NodeJS.

NPM es bastante similar a los gestores de paquetes de Linux (apt-get en Ubuntu) y se puede entender como una forma de administrar módulos que deseamos tener instalados, distribuir los paquetes y agregar dependencias a nuestros programas.

El gestor de paquetes NPM, no obstante, difiere ligeramente de otros gestores de paquetes conocidos, ya que los instala localmente en los proyectos. Es decir, al descargarse un módulo, este se agrega a un proyecto local, que es el que lo tendrá disponible para incluir.

Cabe decir que también existe la posibilidad de instalar los paquetes de manera global en nuestro sistema, los cuales facilitan tareas relacionadas con el sistema operativo. Estos paquetes, una vez instalados, se convierten en comandos disponibles en el terminal, con los que se pueden realizar multitud de tareas. Ejemplos de este tipo de módulos son Bower, Grunt o Gulp.

# Capítulo 2

## Implementación

### 2.1. Análisis de requisitos

Para el desarrollo de este proyecto se han tenido en cuenta una serie de requisitos previos mínimos, necesarios para una correcta implementación del mismo. Los requisitos funcionales del sistema pueden ser divididos en 2 categorías, dependiendo si nos referimos a la aplicación Android o Web, y fueron acordados en los siguientes:

#### 2.1.1. Requisitos funcionales

##### Aplicación Android

- Capacidad de detectar sensores BLE próximos al dispositivo

La aplicación Android debe ser capaz de escanear y detectar cualquier sensor BLE que se encuentre dentro del radio de alcance del dispositivo Android, mostrando su nombre y dirección Media Access Control, o control de acceso al medio (MAC) en pantalla.

- Capacidad de conectarse a cualquier sensor BLE registrado.

La aplicación Android debe ser capaz de establecer una conexión GATT cliente-servidor, mostrando en pantalla los servicios y características ofrecidas por el sensor.

- Capacidad de leer características GATT ofrecidas por el sensor.

La aplicación Android debe ser capaz de interpretar la información proporcionada por un sensor de ritmo cardíaco o pulsómetro comercial, pudiendo determinar entre otros, la frecuencia cardíaca del usuario o el porcentaje de batería restante.

- Capacidad de determinar la posición del dispositivo

La aplicación Android debe de ser capaz de determinar la posición geográfica del dispositivo, en términos de latitud y longitud.

- Capacidad de enviar los parámetros relevantes a través del protocolo HTTP a nuestro servidor web.

La aplicación Android debe ser capaz de enviar valores de frecuencia cardíaca y localización al servidor web implementado, a través de solicitudes tipo POST del protocolo HTTP.

## **Aplicación Web**

- Capacidad de determinar si el pulsómetro está conectado al sistema

La aplicación Web deberá reflejar en todo momento si el pulsómetro se encuentra conectado al sistema y emitiendo datos, o si este se encuentra desconectado.

- Capacidad de mostrar la frecuencia cardíaca actual en tiempo real

El servidor web permanecerá a la escucha de peticiones POST efectuadas por la aplicación puente Android. Estas peticiones contendrán el último valor leído

de frecuencia cardíaca y serán inmediatamente transmitidas al cliente web (cualquier navegador) a través de una conexión web socket. El navegador por tanto mostrará en pantalla el último valor recibido mediante una etiqueta numérica.

- Capacidad de mostrar la evolución de la frecuencia cardíaca mediante el uso de una gráfica dinámica

La aplicación web será capaz de retener un cierto número de valores pasados (dependiendo del tamaño de la pantalla en la cual realizamos la visualización), junto con el valor actual y mostrará dichos valores en una gráfica lineal, la cual dará la sensación de avanzar en el tiempo conforme nuevos valores llegan al cliente web a través de la conexión web socket.

- Capacidad de guardar los datos de frecuencia cardíaca en una base de datos

El servidor web implementado almacenará cada valor recibido en una base de datos no relacional, reflejando la fecha exacta en la cual tuvo lugar dicha muestra. Esto facilita la futura consulta de valores en ciertos rangos de tiempo para así permitir la posible realización de gráficas y estadísticas a tal efecto.

- Capacidad de determinar la posición del usuario usando el servicio de mapas de Google

La aplicación web debe ser capaz de mostrar en todo momento la localización del usuario haciendo uso del pulsómetro.

- Capacidad de notificar por e-mail a un grupo de usuarios interesados en realizar tareas de monitorización

Cuando el valor de ritmo cardíaco sobrepase un cierto rango, se enviará una notificación por e-mail alertando de condiciones anormales, junto con la localización actual del usuario usando el pulsómetro y se mantendrá al interesado

o grupo de interesados informado con actualizaciones frecuentes, hasta la que la situación vuelva a la normalidad.

- Capacidad de adaptarse fácilmente a pantallas más pequeñas, tales como teléfonos móviles y tabletas

La aplicación debe ser *web responsive* y amigable para el usuario en cualquier tipo de pantalla.

### 2.1.2. Requisitos no funcionales

Adicionalmente, los siguientes requisitos no funcionales fueron considerados, refiriéndose a la Aplicación como el conjunto Android + Web.

- Extensibilidad futura. La aplicación debe de ser fácilmente extensible en un futuro. Para ello se tendrá en mente la modularidad y el código autoexplicativo durante el desarrollo de la aplicación
- Tolerancia a fallos. La aplicación debe de tolerar pequeños fallos sin que estos entorpezcan en medida alguna la posible medición en curso.
- Rendimiento. La aplicación no debe de ser una carga importante para el rendimiento del sistema, ya que esto podría resultar en mediciones imprecisas o pérdida de muestras por incapacidad del sistema de copiar con la carga.
- Usabilidad. La aplicación no debe presentar un dificultoso manejo de cara al usuario.

### 2.1.3. Diseño de la interfaz gráfica

Como paso previo a la implementación en código, se han diseñado unos borradores de las distintas pantallas de las que constará la aplicación, tanto web como Android.

## Android

La aplicación se ha dividido en cuatro pantallas básicas: reposo, búsqueda, desconectado y conectado, las cuales se pueden observar en las figuras 2.1, 2.2, 2.3 y 2.4 respectivamente:

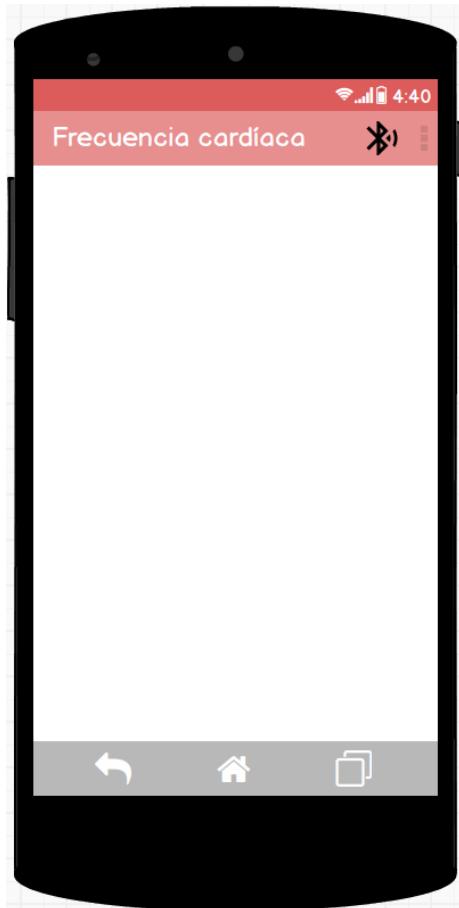


Figura 2.1: Bosquejo Android estado  
Reposo

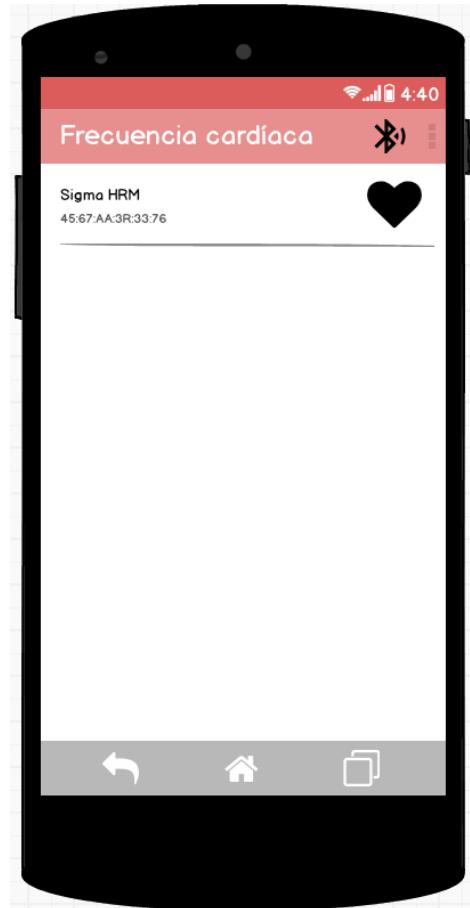


Figura 2.2: Bosquejo Android estado  
Búsqueda

### Reposo

Representa la primera actividad que será cargada en primer plano cuando lancemos la aplicación. Para empezar a escanear dispositivos dentro del alcance

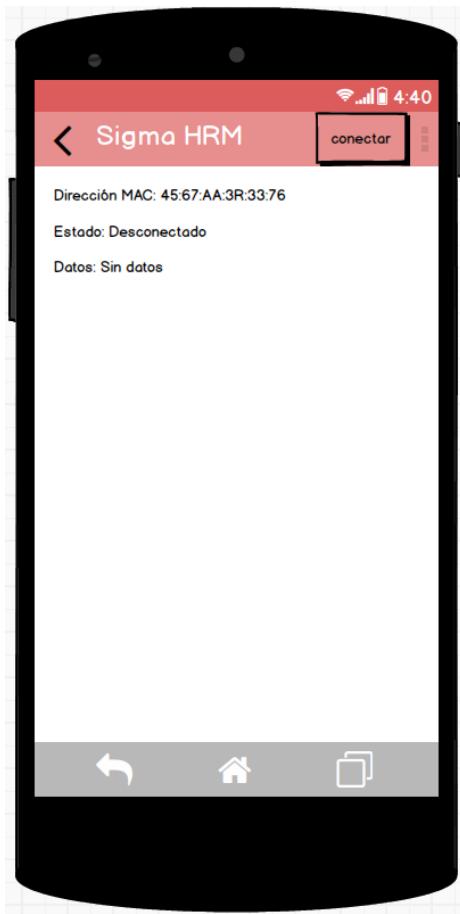


Figura 2.3: Bosquejo Android estado Desconectado

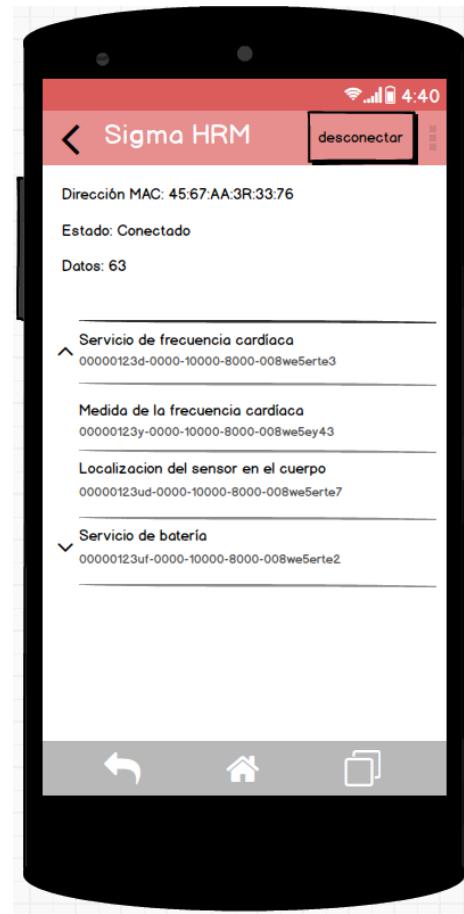


Figura 2.4: Bosquejo Android estado Conectado

de nuestro teléfono debemos pulsar el botón situado en la barra de acción superior, cuyo ícono es el logotipo de Bluetooth.

### Búsqueda

Muestra una lista con los dispositivos sensores BLE encontrados, en particular el nombre y dirección MAC del dispositivo. Para obtener información de cualquiera de los sensores, debemos pulsar el elemento correspondiente en la lista. Esto nos llevará a la actividad de perfil del dispositivo, en principio en su estado *desconectado*.

### Desconectado

Muestra una interfaz gráfica básica para la monitorización de la conexión, junto con la dirección MAC del sensor. Para conectarnos al dispositivo debemos pulsar en el botón conectar situado en la barra de acción superior. Una vez que la conexión Bluetooth ha sido establecida, la interfaz de usuario cambiará a su estado *conectado*.

### Conectado

Muestra una lista expandible de los servicios ofrecidos por el sensor. Para consultar las características que un servicio en particular contiene, debemos pulsar en dicho elemento de la lista y esto hará que se expandan las características disponibles. Para empezar a recibir datos del sensor debemos pulsar en cualquier característica, lo que actualizará el campo “datos” de la actividad mostrando el valor recibido. Mediante el pulsado de la característica “Medida de la Frecuencia Cardíaca” activaremos e instanciaremos el cliente HTTP para así comenzar a enviar valores de ritmo cardíaco en tiempo real y la localización geográfica del usuario a nuestro servidor web mediante solicitudes de tipo POST y la aplicación podrá seguir funcionando en segundo plano, hasta que decidamos desconectar.

## Web

La aplicación web presenta dos estados básicos en cuanto a la interfaz gráfica se refiere, que son desconectado y conectado, las cuales pueden contemplarse en las figuras 2.5 y 2.6 respectivamente. Si la aplicación Android se encuentra en estado *conectado*, la aplicación web automáticamente se encontrará en estado *conectado* mostrando una gráfica con los últimos valores recibidos junto con el valor actual. El resto de posibles estados de la aplicación Android se corresponderán con el estado *desconectado* de la aplicación web.

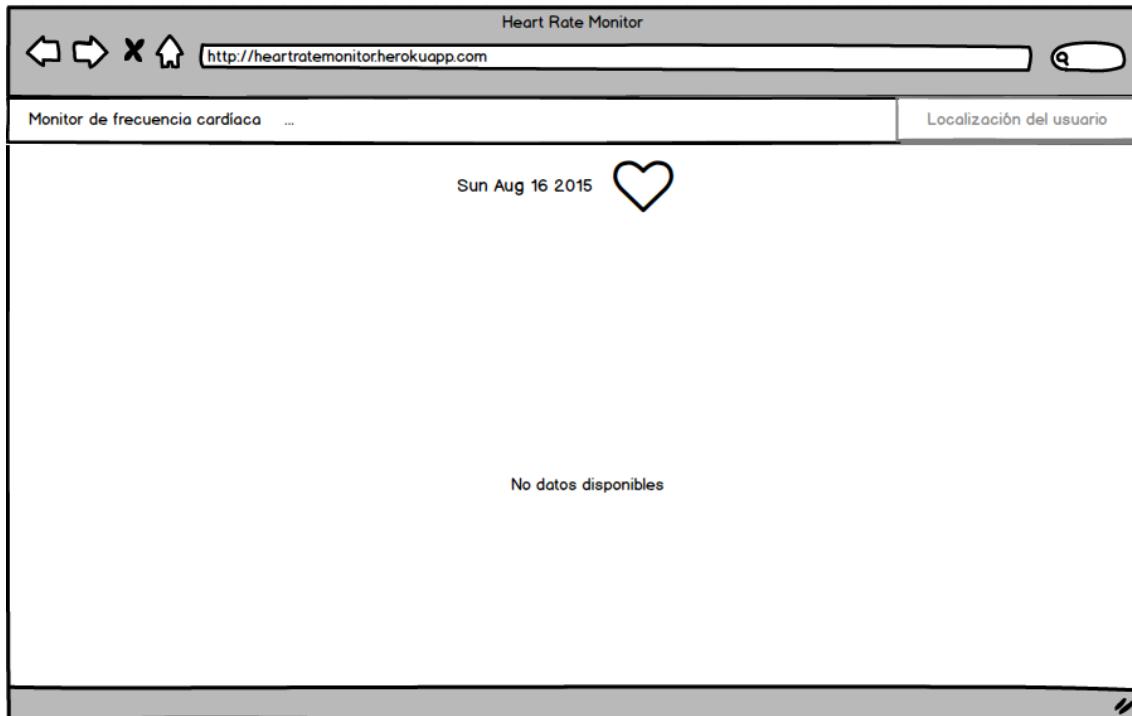


Figura 2.5: Bosquejo web con sensor desconectado

El botón con etiqueta “Localización del usuario” nos redirigirá al servicio de mapas de Google mostrando la localización del poseedor del sensor en ese preciso instante y estará habilitado únicamente si nos encontramos en el estado *conectado*.

#### 2.1.4. Dispositivo Android

El desarrollo de este proyecto ha sido realizado y probado en un Google Nexus 5, bajo la versión Android 5.0, lo cual garantiza que la aplicación conserva toda su funcionalidad en los modelos más modernos, tanto en software como en hardware.

La versión mínima de Android requerida para el correcto funcionamiento de esta aplicación, es el nivel de Application Programming Interface, o interfaz de programación de aplicaciones (API) 18, correspondiente a Android 4.3, ya que es la primera versión que incorpora soporte para BLE.



Figura 2.6: Bosquejo web con sensor conectado

## 2.2. Aplicación Android

### 2.2.1. Entorno de desarrollo integrado

Es posible elegir entre varios entornos integrados de desarrollo a la hora de desarrollar una aplicación para la plataforma Android. Los dos principales son Eclipse y Android Studio.

Android Studio es un Integrated Development Environment, o entorno de desarrollo integrado (IDE) específico para desarrollar en la plataforma Android. Está basado en el popular IntelliJ IDEA de la compañía JetBrains, pero fuertemente modificado, tanto en aspecto como en funcionalidad. Android Studio es de reciente desarrollo, llegando a su versión 1.0 inicial en Diciembre de 2014. La versión 1.0.2

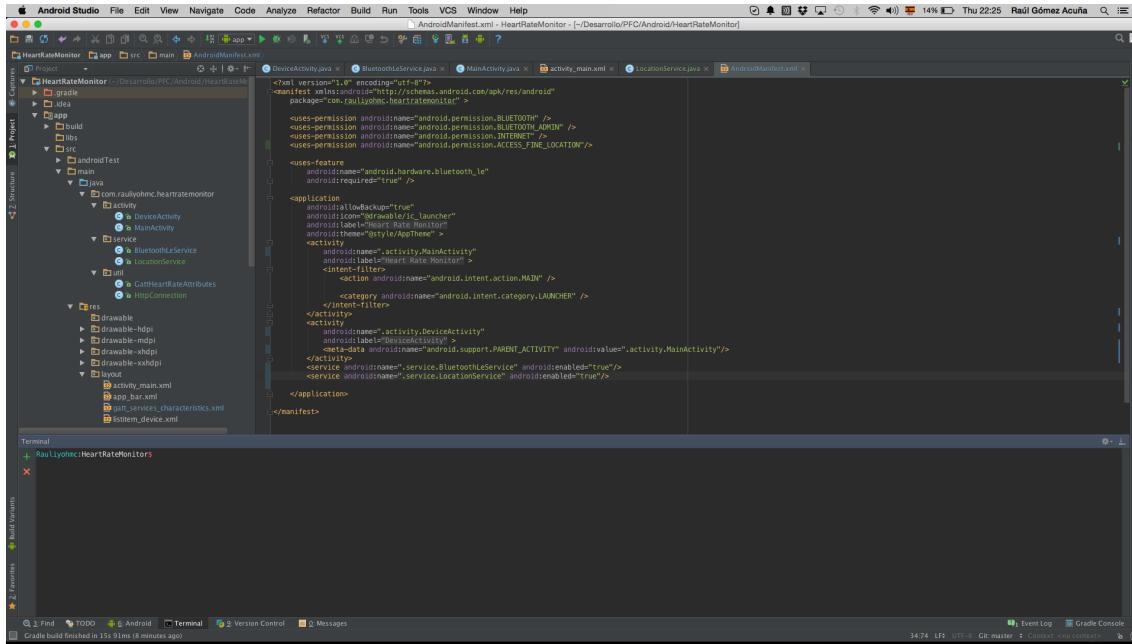


Figura 2.7: Entorno de desarrollo integrado Android Studio

se puede observar en la figura 2.7.

Eclipse es un IDE de propósito general, que provee un espacio de trabajo y un sistema de complementos muy extensible y flexible. Está escrito en Java, e inicialmente se enfocó al desarrollo en dicho lenguaje.

Gracias a su sistema de complementos, Eclipse puede ser utilizado para desarrollar aplicaciones en múltiples lenguajes: Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Lua, Natural, Perl, PHP, Prolog, Python, R, Ruby (incluyendo el framework Ruby on Rails), Scala, Clojure, Groovy, Scheme y Erlang. Eclipse fue también el primer IDE con soporte para desarrollo en la plataforma Android. La figura 2.8 muestra la versión 4.4 Luna.

A la hora de realizar el proyecto, se consideraron ambas soluciones, y se optó por utilizar el entorno de desarrollo integrado Android Studio, por varias razones.

- Android Studio utiliza Gradle como su sistema de compilación, el cual resulta

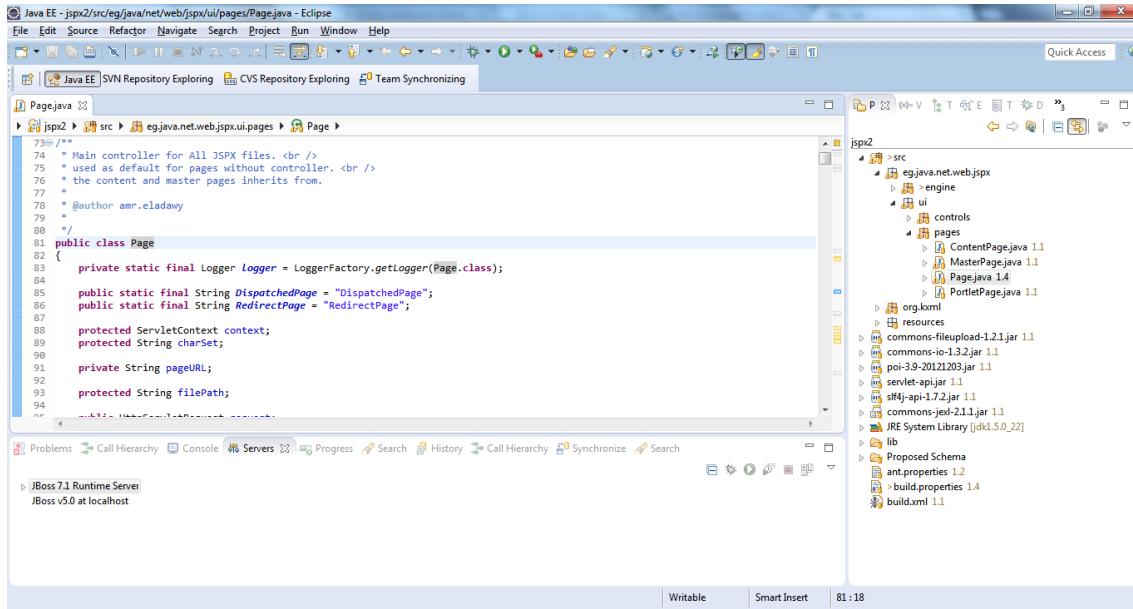


Figura 2.8: Entorno de desarrollo integrado Eclipse

mucho más claro y menos tedioso de lidiar que ANT, el utilizado por Eclipse. Con Gradle, es mucho más sencillo declarar dependencias de librerías externas.

- Android Studio posee una superior cantidad y calidad de análisis en tiempo real del código del proyecto. Autocompletado de código, refactorización asistida y sugerencias de optimización son algunas de las características en las que supera a Eclipse.
- El editor gráfico de interfaces gráficas de usuario, aunque presente en Eclipse de manera básica, es mucho más completo, rápido y preciso en Android Studio.
- El único aspecto en el que Eclipse supera a Android Studio es en el soporte para extensiones nativas de Android, escritas en C++, que todavía no ha sido implementado en Android Studio. Sin embargo, en este proyecto no se hace uso de dichas extensiones, por lo que no supone un problema.

### 2.2.2. Estructura del código

La estructura de la aplicación sigue la estructura estándar de proyecto Android que se sigue en el entorno de desarrollo Android Studio. Se distinguen cuatro grupos principales:

**Manifiestos** son archivos que presentan la aplicación al sistema operativo y proveen información básica sobre la misma. Esta información incluye el paquete Java de la aplicación, los componentes de la misma (actividades, servicios, eventos de emisión, proveedores de contenido...), los permisos requeridos por la aplicación, el nivel mínimo de API Android requerido y los permisos proporcionados por la aplicación.

**Código fuente** escrito en el lenguaje de programación Java, el cual será compilado posteriormente y convertido a código intermedio. Toda la lógica de la aplicación está implementada de esta manera. El código fuente a su vez se organiza dentro de paquetes, que son agrupaciones lógicas y funcionales de distintos archivos de código fuente.

**Recursos** que incluyen todo elemento gráfico, diseño de interfaces gráficas de usuario o componentes de las mismas, composición de menús contextuales y valores de variables. Es posible proveer distintas versiones de cada uno de los recursos enfocadas a características en concreto de los dispositivos. Por ejemplo, es posible especificar un recurso para un idioma del teléfono, tamaño de pantalla, orientación del dispositivo, versión de la API Android e incluso si el dispositivo se encuentra en modo nocturno o no.

**Instrucciones de compilación** escritos en el Domain Specific Language, o lenguaje específico del dominio (DSL) de Gradle. Esto es una herramienta de automatización de proyectos que permite manejar de manera cómoda y eficiente

tareas como gestión de dependencias, compilación, empaquetado y publicado de artefactos.

A su vez, el código fuente está dividido en subpaquetes. Se ha creado un paquete para las actividades, otro paquete para los servicios y un último paquete para las clases de utilidad y misceláneas. Dicha estructura puede observarse en la figura 2.9.

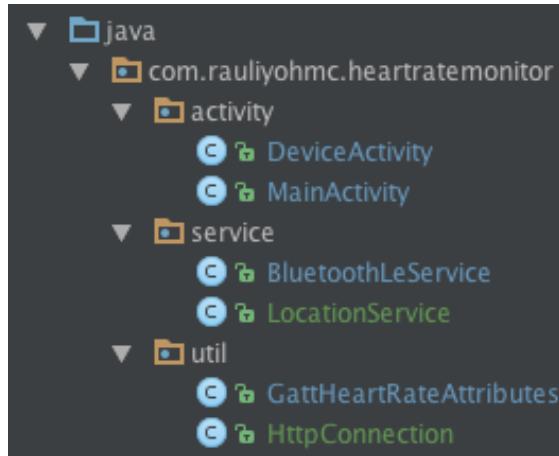


Figura 2.9: Organización del código fuente

### 2.2.3. Interfaz gráfica

La interfaz gráfica de esta aplicación se ha desarrollado siguiendo las nuevas guías de diseño que incorpora el sistema Lollipop. La versión 5.0 del sistema operativo de Android recomienda una nueva tendencia de diseño, llamada *Material Design* (diseño material), la cual tiene como objetivo crear un lenguaje visual que sintetiza los principios clásicos de buen diseño mediante el uso de la innovación, tal y como su principio rige.

En concreto se han seguido las recomendaciones para el diseño de los elementos de la lista de dispositivos, la paleta de colores, y la barra de acción o *Action bar* como es comúnmente conocida en el mundo Android.

En Android hay dos maneras de definir una interfaz gráfica: programáticamente o por archivos de recurso XML. Se ha optado por la segunda opción. Esto disminuye el acoplamiento en el código, aumenta la reusabilidad y la claridad del mismo. Por otra parte, el entorno de desarrollo permite previsualizar el resultado de dichos archivos XML con bastante fidelidad.

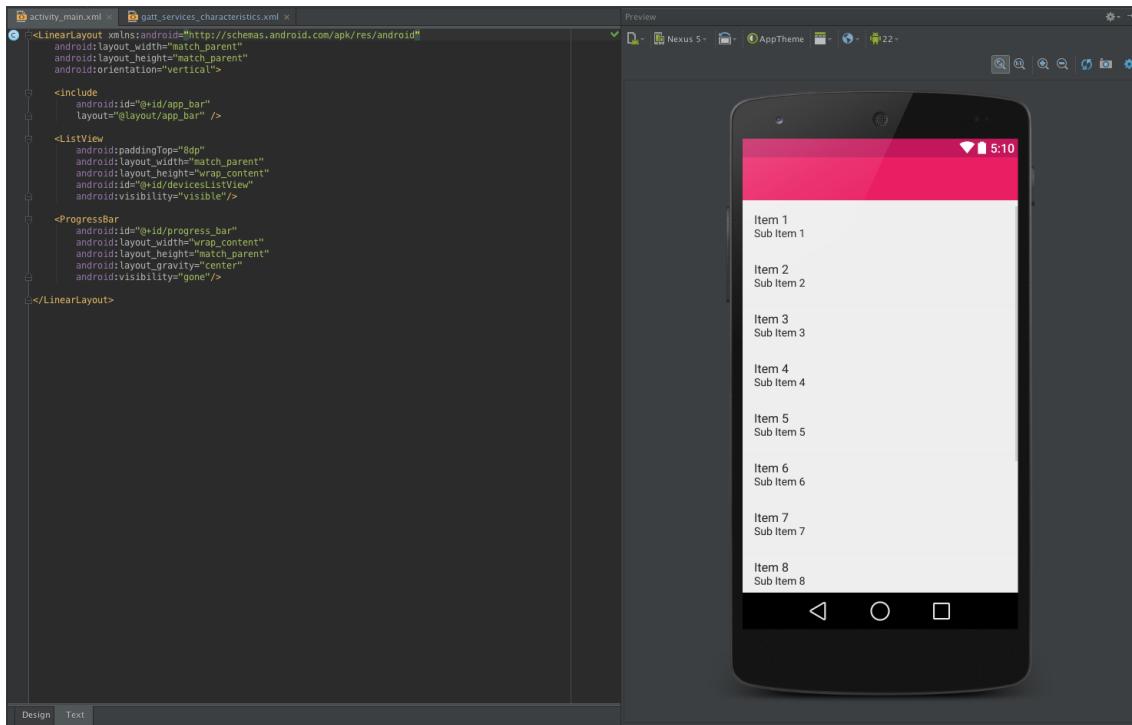


Figura 2.10: Editor de archivos de recurso XML con vista previa

Para el diseño de los archivos XML se ha hecho uso de la herramienta incorporada para tal propósito en el entorno de desarrollo Android Studio; la cual nos brinda la posibilidad de tener una vista previa del resultado de la descripción en XML que estamos realizando de la interfaz gráfica de usuario. Como ejemplo, en la figura 2.10 se puede observar la edición de la interfaz gráfica de usuario de la actividad encargada de mostrar los servicios y características de un sensor en concreto.

Si así se desea, también cambie la posibilidad de diseñar las interfaces gráficas

de usuario de la aplicación mediante la misma herramienta, pero de una manera completamente visual, tal y como se observa en la figura 2.11. De esta manera, se sacrifica precisión y control por comodidad y claridad.

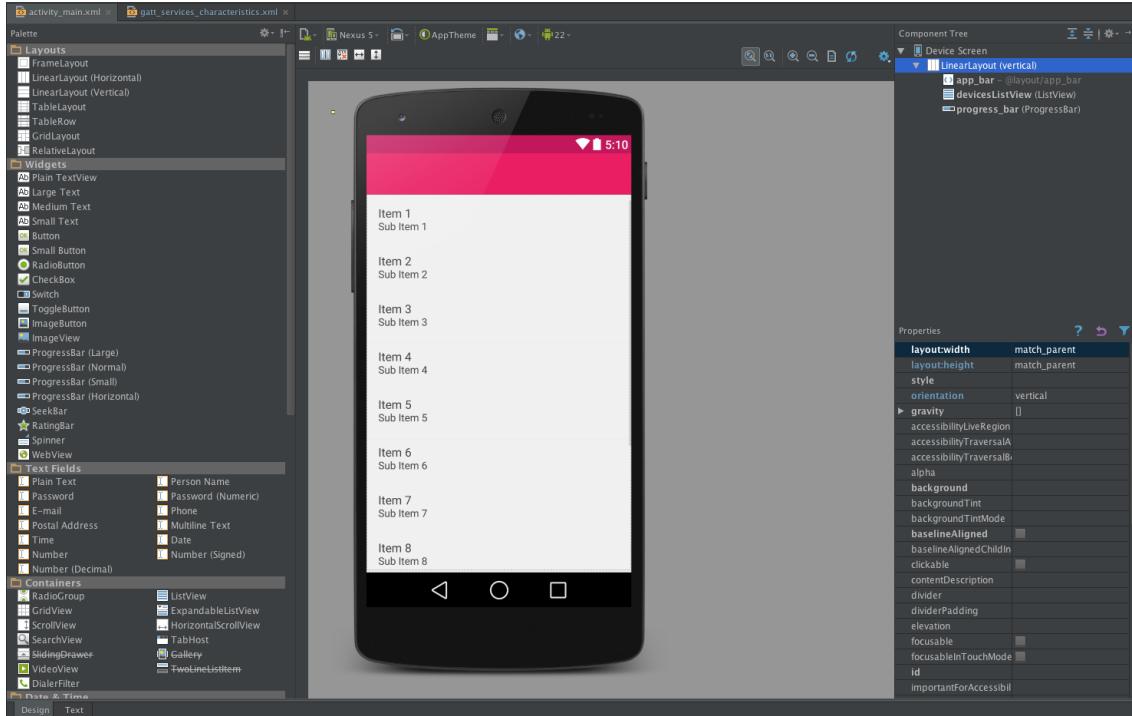


Figura 2.11: Edición completamente visual de la interfaz gráfica de usuario de la aplicación

Dentro de cada pantalla, la estructuración de los elementos que se observan se realiza a través de layouts. Los layouts pueden definirse como contenedores de una o más vistas, que ayudan al posicionamiento de cada una de ellas dentro de la aplicación así como a controlar el comportamiento de las mismas. Este concepto encaja a la perfección con el anidado de etiquetas de XML. En el ejemplo de la figura 2.10, el código XML es el presente en el extracto 2.1. En dicho extracto se puede observar cómo el elemento `LinearLayout` contiene a 3 elementos. La directiva `include` nos permite importar fragmentos de xml contenidos en otros archivos,

facilitando la modularización y reutilización de código.

```
1 <LinearLayout
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:orientation="vertical">
5     <include
6         android:id="@+id/app_bar"
7         layout="@layout/app_bar" />
8     <ListView
9         android:paddingTop="8dp"
10        android:layout_width="match_parent"
11        android:layout_height="wrap_content"
12        android:id="@+id/devicesListView"
13        android:visibility="visible"/>
14     <ProgressBar
15         android:id="@+id/progress_bar"
16         android:layout_width="wrap_content"
17         android:layout_height="match_parent"
18         android:layout_gravity="center"
19         android:visibility="gone"/>
20 </LinearLayout>
```

---

Fragmento de código 2.1: Descripción de la interfaz gráfica de la actividad **MainActivity**

En este caso incluimos la barra de acción, la cual es común en varios de nuestros layouts. **ListView** hace referencia a una lista con elementos, que representarán los dispositivos BLE encontrados, siendo esta visible cuando la aplicación se inicializa. Por último tenemos un elemento **ProgressBar**, o barra de progreso, que permanecerá activa cuando la aplicación se encuentre escaneando dispositivos. Como podemos observar cada elemento define sus propios atributos o propiedades, como pueden ser el margen, anchura, altura o alineamiento del widget.

El atributo **id** es un identificador único asociado a cada elemento xml, el cual será usado posteriormente en nuestro código Java para referenciar dichos elementos programáticamente y poder operar con ellos.

## Menús y barra de acción

La barra de acción, más comúnmente conocida por su versión inglesa `ActionBar`, es uno de los patrones de diseño más comunes en una aplicación Android. Su función es albergar las funciones importantes y proveer al usuario un fácil, rápido e intuitivo acceso a ellas. La forma de definir los elementos que componen un `ActionBar` en un determinado instante, así como sus prioridades y jerarquía, es mediante xml, concretamente usando el elemento `menu` como parente. La figura 2.12 muestra las acciones de escaneo y detención, las cuales están representadas por los dos iconos de color blanco.

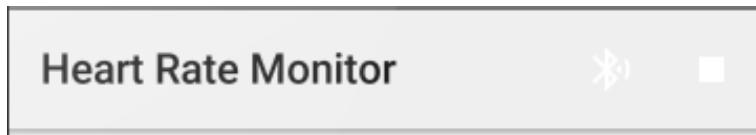


Figura 2.12: Barra de Acción en la actividad principal `MainActivity`

### 2.2.4. Diagrama de flujo

Antes de entrar en detalles técnicos de implementación relacionados con las interacciones que nuestra aplicación Android realiza con el sensor mediante la API de Bluetooth de baja energía, el acceso a servicios de localización y la comunicación con el servidor web, vamos a mostrar un diagrama de flujo completo de la aplicación, para así dotar al lector de una visión general de cómo la aplicación se comporta desde que esta se inicia hasta que esta se finaliza. Este puede ser visualizado en la figura 2.13.

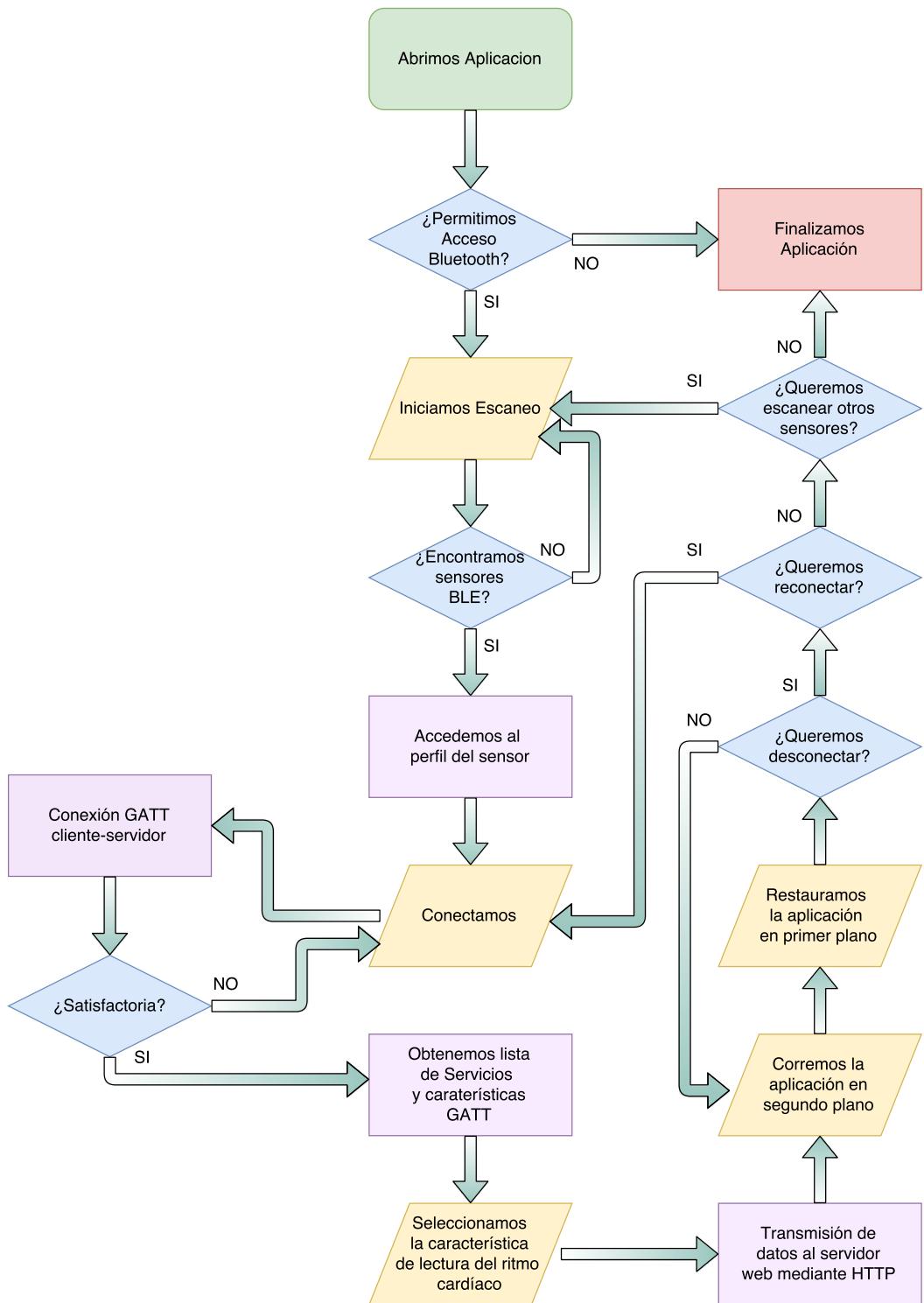


Figura 2.13: Diagrama de flujo de la aplicación Android

### 2.2.5. Comunicación con el sensor mediante BLE

La figura 2.14 muestra los distintos niveles de la arquitectura de Bluetooth en el sistema operativo Android y su interconexión.

En el nivel de marco de aplicación, el cual es el nivel en el que nuestras aplicaciones corren, estas usan las clases del paquete `android.bluetooth.*` para interactuar con el hardware Bluetooth. Internamente, estas clases llaman al proceso Bluetooth a través de un juntador, que es el llamado Binder y lo que permite dicha interconexión.

El servicio del sistema Bluetooth está empaquetado como una Aplicación Android de por sí, e implementa el servicio Bluetooth y los distintos perfiles soportados en Android. Esta aplicación se conecta con la capa Hardware Abstraction Layer, o capa de abstracción del hardware (HAL) a través de un Java Native Interface, o interfaz nativa Java (JNI), el cual es un framework de programación que permite que un programa escrito en Java ejecutado en la máquina virtual java (JVM) pueda interactuar con programas escritos en otros lenguajes como C, C++ y ensamblador.

El sistema operativo Android provee una HAL que conecta todas las API de alto nivel con los controladores y hardware subyacentes en cada dispositivo, así como con las posibles extensiones añadidas a la pila de Bluetooth proporcionadas por terceras partes.

Lo primero que debemos hacer es declarar en el manifiesto permisos de acceso de la aplicación a Bluetooth, para que así la aplicación pueda utilizar las respectivas APIs y acceder al proceso Bluetooth.

Si el dispositivo Android no soporta BLE, se lo indicaremos al usuario mediante un mensaje informativo, justo al inicializar la aplicación, finalizando esta tan pronto como el mensaje desaparezca.

En el fragmento 2.2, vemos el código encargado de dicha función;

Previo a que nuestra aplicación pueda comunicarse a través de BLE, y una vez

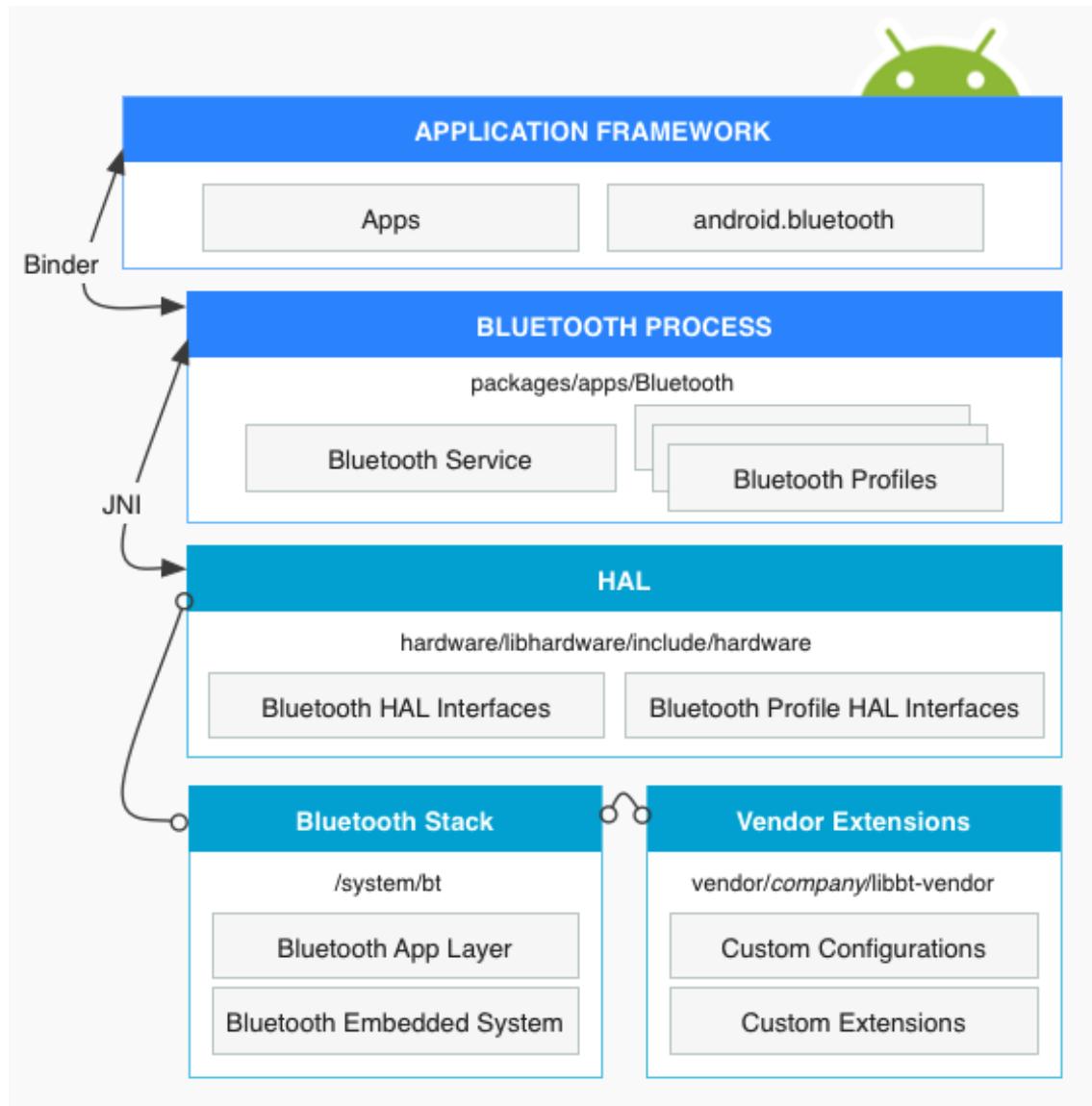


Figura 2.14: Arquitectura de Bluetooth en Android [And07]

```
1 if (!getPackageManager()  
2     .hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {  
3     Toast.makeText(this,  
4         R.string.ble_not_supported,  
5         Toast.LENGTH_SHORT)  
6     .show();  
7 }  
8 finish();
```

---

Fragmento de código 2.2: Detección de soporte BLE en un dispositivo Android

que hemos comprobado que BLE es soportado en nuestro dispositivo, debemos configurarlo. Primero necesitamos obtener una instancia de la clase `BluetoothAdapter`, la cual es requerida para cualquier actividad relacionada con Bluetooth. Esta clase representa el adaptador Bluetooth del dispositivo (la radiofrecuencia Bluetooth). Existe únicamente una instancia compartida a través de toda la aplicación, conocido como *singleton*. Para obtener el adaptador debemos usar `getSystemService()` para devolver una instancia de `BluetoothManager`, la cual es finalmente usada para obtener el adaptador. Como siguiente paso debemos habilitar Bluetooth en nuestro dispositivo. Para ello comprobamos si está actualmente habilitado y si no es así lo habilitamos usando uno de los `Intent` implícitos que Android provee. Un `Intent` no es más que una descripción abstracta de una operación que debe ser realizada. Permite de esta forma interactuar con otras actividades, en este caso con la Actividad preinstalada en Android que maneja el controlador Bluetooth. El fragmento 2.3 muestra este proceso:

Para encontrar dispositivos BLE dentro del alcance de nuestro teléfono Android, debemos usar el método `startLeScan`. Este método presenta una versión que nos permite especificar los servicios GATT en los cuales estamos interesados como primer parámetro, mediante un arreglo o array de objetos `UUID`. Universally Unique Identifier, o identificador universal único (`UUID`) representa un có-

---

```

1 // Inicializa el adaptador Bluetooth
2 final BluetoothManager bluetoothManager =
3     (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
4 mBluetoothAdapter = bluetoothManager.getAdapter();
5 ...
6 // Asegura que Bluetooth esta habilitado en el dispositivo. Si no,
7 // muestra un dialogo requiriendo al usuario su activacion
8 if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
9     Intent enableBtIntent =
10        new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
11     startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
12 }
```

---

Fragmento de código 2.3: Configuración BLE en Android

digo identificador estándar que se utiliza en el proceso de construcción de software. En el caso del servicio GATT de frecuencia cardíaca, su UUID asociado es el 0000180d-0000-1000-8000-00805f9b34fb. El segundo parámetro de la función es un objeto `BluetoothAdapter.LeScanCallback`. Debemos implementar esta retrolllamada, porque es la forma en la que los resultados de escaneo serán devueltos. Debido a que escanear dispositivos consume bastante batería, debemos ser cuidadosos y seguir los siguientes criterios:

- Tan pronto como encontremos el dispositivo deseado, paramos de escanear.
- No debemos escanear en bucle, fijando un límite de tiempo para el escaneo.

La función encargada de empezar y parar el escaneo es la mostrada en el fragmento 2.4.

El fragmento 2.5 muestra un ejemplo de implementación de la interfaz `BluetoothAdapter.LeScanCallback`, usada para entregar resultados de escaneo BLE.

Para garantizar que las operaciones GATT servidor-cliente no se ven interferidas por otras funciones de la aplicación, se ha optado por crear un servicio que corra

```
1 private static final long SCAN_PERIOD = 10000;
2 private static final UUID[] heartRateGattServices = new UUID[1];
3 heartRateGattServices[0] =
4     UUID.fromString("0000180d-0000-1000-8000-00805f9b34fb");
5 ...
6 private void scanLeDevice(final boolean enable) {
7     if (enable) {
8         // Detiene el escaneo despues del tiempo predefinido, 10s aqui
9         handler.postDelayed(new Runnable() {
10             @Override
11             public void run() {
12                 scanning = false;
13                 bluetoothAdapter.stopLeScan(leScanCallback);
14             }
15         }, SCAN_PERIOD);
16         scanning = true;
17         bluetoothAdapter
18             .startLeScan(heartRateGattServices, leScanCallback);
19     } else {
20         scanning = false;
21         bluetoothAdapter
22             .stopLeScan(heartRateGattServices, leScanCallback);
23     }
24     ...
25 }
```

---

Fragmento de código 2.4: Proceso de escaneo de dispositivos BLE en Android

en segundo plano. Este responde a los comandos de la aplicación en primer plano, tal y como se explicó en el apartado 1.2.2. Este servicio, `BluetoothLeService`, será del tipo `Bound` y estará adherido a la actividad `DeviceActivity`, la cual se encarga de mostrar los datos relativos al sensor que nos queremos conectar, tal y como podíamos ver en los bosquejos 2.3 y 2.4 en sus estados desconectado y conectado respectivamente. Recordemos que un servicio del tipo `Bound` está activo durante el tiempo en que la actividad `DeviceActivity` esté adherida a él y que es destruido cuando dicha actividad se desadhiere.

```

1  private LeDeviceListAdapter deviceListAdapter;
2  ...
3  //Retrollamada de escaneo BLE
4  private BluetoothAdapter.LeScanCallback leScanCallback =
5      new BluetoothAdapter.LeScanCallback() {
6      @Override
7      public void onLeScan(final BluetoothDevice device, int rssi,
8          byte[] scanRecord) {
9          runOnUiThread(new Runnable() {
10         @Override
11         public void run() {
12             deviceListAdapter.addDevice(device);
13             deviceListAdapter.notifyDataSetChanged();
14         }
15     });
16   }
17 };

```

---

Fragmento de código 2.5: Implementación de la interfaz BluetoothAdapter.LeScanCallback

Una vez pulsemos en “Conectar” en dicha actividad, se llamará en código al método `connectGatt()` dentro del servicio. Este método recibe 3 parámetros, un objeto `Context`, en este caso el contexto del servicio, un booleano indicando si debemos conectarnos automáticamente al dispositivo BLE tan pronto como este se encuentre disponible, y una retrollamada del tipo `BluetoothGattCallback`.

```
bluetoothGatt = device.connectGatt(this, false, bluetoothGattCallback);
```

Esta expresión devuelve una instancia de la clase `BluetoothGatt`, la cual podemos usar para llevar a cabo operaciones GATT del lado del cliente, o sea la aplicación Android. El tercer parámetro `bluetoothGattCallback`, instancia de la clase abstracta `BluetoothGattCallback`, es una retrollamada utilizada para entregar resultados al cliente, tales como el estado de la conexión o cualquier futura operación GATT del lado del cliente. La forma de definirla es mediante la implementación de su clase Abstracta, en particular aquellos métodos en los cuales estamos interesados.

Estos métodos serán los listados a continuación:

- **onConnectionStateChange**

Retrollamada que será ejecutada cuando el estado de la conexión GATT cambie. En nuestro caso, en cuanto la conexión con el sensor se haya establecido, procederemos a descubrir los servicios GATT ofrecidos por este. Una vez que nos hayamos desconectado enviaremos una petición HTTP al servidor web para informar al usuario apropiadamente.

- **onServicesDiscovered**

Retrollamada que será ejecutada justo cuando el proceso de descubrir los servicios GATT ofrecidos por el sensor haya finalizado. Aquí procederemos a actualizar la interfaz de usuario en la actividad `DeviceActivity`, mostrando una lista expandible con los servicios GATT ofrecidos.

- **onCharacteristicRead**

Una vez que seleccionemos la característica lectura del ritmo cardíaco, dentro del servicio del ritmo cardíaco, procederemos a recibir periódicamente y con una frecuencia en torno al segundo datos provenientes del sensor BLE. Cada vez que recibamos un nuevo valor de frecuencia cardíaca, esta retrollamada será ejecutada, y llevaremos a cabo el envío de dicho valor al servidor web.

El proceso de análisis sintáctico y gramatical de los datos que nos llegan del sensor (parsing) se lleva a cabo siguiendo las especificaciones del perfil Bluetooth de la medida de ritmo cardíaco<sup>1</sup>. Estos datos serán recibidos por la aplicación Android como una agrupación de bytes. Nuestro objetivo es obtener un número entero legible es por lo que debemos transformar los datos. El código encargado de la transformación es el mostrado en el fragmento 2.6.

---

<sup>1</sup>Para consultar las especificaciones completas, se invita al lector a visitar <https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx>

---

```

1 if (UUID_HEART_RATE_MEASUREMENT.equals(characteristic.getUuid())) {
2     int flag = characteristic.getProperties();
3     int format = -1;
4     if ((flag & 0x01) != 0) {
5         format = BluetoothGattCharacteristic.FORMAT_UINT16;
6     } else {
7         format = BluetoothGattCharacteristic.FORMAT_UINT8;
8     }
9     final int heartRate = characteristic.getIntValue(format, 1);
10 }
```

---

Fragmento de código 2.6: Parsing para el perfil de ritmo cardíaco

En la actividad `DeviceActivity` usaremos un objeto de la clase `BroadcastReceiver` para poder recibir eventos desde el Servicio y así actualizar la interfaz de usuario acordemente.

Por último, una vez que la aplicación ha terminado la interacción con el sensor BLE, debemos liberar conexiones y recursos apropiadamente. En nuestro caso, el proceso de cerrar la conexión ocurrirá cuando pulsemos el botón “Desconectar” en la barra de acción. De esta forma la actividad `Device Activity` se desconectará del servicio y la conexión GATT cliente-servidor será cerrada.

### 2.2.6. Ubicación

Para obtener la ubicación absoluta del dispositivo móvil, la aplicación se comunica con los GMS. Estos no solo actúan como una capa de abstracción del hardware GPS del dispositivo, sino que además proveen servicios de valor añadido. Algunos de los beneficios incluyen un menor tiempo de precalentamiento, localización por redes Wi-Fi y optimización del consumo energético. No obstante, antes de hacer uso de los GMS, es necesario configurar el proyecto para que obtenga como dependencias las librerías pertinentes. Gracias al sistema de compilación de Gradle, este paso es sumamente sencillo.

---

```

1 dependencies {
2     compile fileTree(dir: 'libs', include: ['*.jar'])
3     compile 'com.android.support:appcompat-v7:21.0.3'
4     compile 'com.android.support:recyclerview-v7:+'
5     compile 'com.google.android.gms:play-services:7.3.0'
6 }
```

---

Fragmento de código 2.7: Sección de dependencias dentro del archivo `build.gradle`

Los GMS en su versión más reciente se han fragmentado, así que solo es necesario declarar como dependencias las partes que sean necesarias en el proyecto, para evitar sobrecargar la aplicación. Las dependencias se añaden editando el archivo `build.gradle`. Para utilizar la parte de ubicación de los GMS, se añade la línea 4 del fragmento 2.7, dado que ubicación se encuentra en la parte general de las librerías. Las dependencias son añadidas con el descriptor Maven<sup>2</sup> del paquete, siguiendo el esquema `paquete-padre:librería:versión`, y son descargadas del repositorio central de Maven.

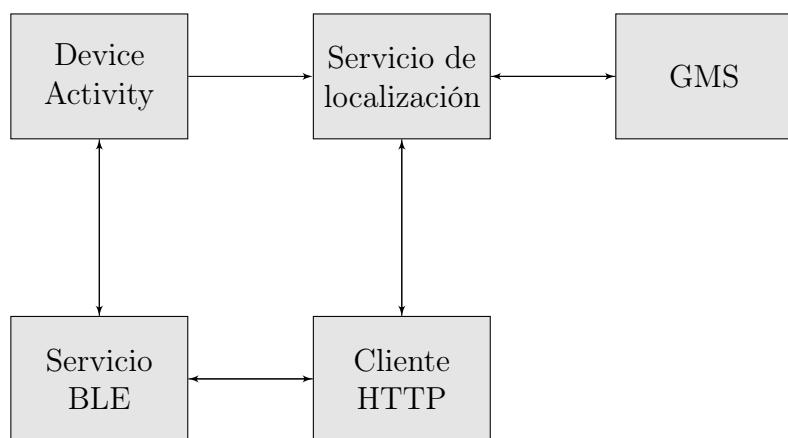


Figura 2.15: Diagrama de interacción de la aplicación y los servicios

Para comunicarse con los GMS, es utilizado un servicio en segundo plano por los

<sup>2</sup>El funcionamiento de la herramienta Maven y el sistema de gestión de proyectos de software asociado quedan fuera del ámbito de este proyecto. No obstante, se invita al lector a visitar [maven.apache.org](http://maven.apache.org) para conocer más acerca de los mismos.

mismos motivos que en el apartado anterior de comunicación con el sensor BLE. De hecho, es un servicio intermedio, ya que la obtención en sí de la posición la realiza el servicio GMS. La interacción de la aplicación con los distintos servicios se puede observar en el diagrama 2.15. El servicio de localización creado en la aplicación se conecta a los GMS, solicita actualizaciones y las pone a disponibilidad del resto de la aplicación de la manera mostrada en 2.8.

---

```

1 googleApiClient = new GoogleApiClient.Builder(this)
2     .addConnectionCallbacks(this)
3     .addOnConnectionFailedListener(this)
4     .addApi(LocationServices.API)
5     .build();
6 createLocationRequest();
7 googleApiClient.connect();

```

---

Fragmento de código 2.8: Solicitud de conexión a los GMS.

### 2.2.7. Comunicación con el servidor web

Para enviar al servidor web el valor de frecuencia cardíaca y la ubicación del usuario, decidimos hacer uso de las clases `httpClient` y `httpPost` en Android, las cuales implementan el protocolo HTTP y el método HTTP POST respectivamente, proporcionándonos una API flexible y fácil de usar. Con tal propósito se creo una clase en java encargada de todas las operaciones HTTP, `HttpConnection`, la cual contiene dos métodos, uno para enviar el valor de frecuencia cardíaca y otro para enviar la localización en forma de valores de longitud y latitud. El fragmento 2.9 muestra el primero de los métodos:

En primer lugar creamos un objeto JSON, que no es mas que una tabla hash o diccionario con claves y valores. Adoptamos el convenio de enviar el valor “-1” como frecuencia cardíaca al servidor, para comunicarle que nos hemos desconectado del pulsómetro. Utilizamos la clase `HttpPost` para efectuar peticiones de tipo POST al

```
1 public void sendHeartRateToWebServer(int heartRate){  
2     ...  
3     JSONObject jsonObject = new JSONObject();  
4     // Le decimos al servidor de desconectar enviando el valor -1  
5     if(heartRate == -1){  
6         jsonObject.put("emitting", "false");  
7     } else {  
8         jsonObject.put("emitting", "true");  
9     }  
10    jsonObject.put("heartRate", heartRate);  
11    StringEntity se = new StringEntity(jsonObject.toString());  
12    httpPost.setEntity(se);  
13    httpPost.setHeader("Accept", "application/json");  
14    httpPost.setHeader("Content-type", "application/json");  
15    HttpResponse httpResponse = httpClient.execute(httpPost);  
16    ...  
17 }
```

---

Fragmento de código 2.9: Método para el envío de valores de frecuencia cardíaca al servidor web

servidor. De esta forma podemos enviar datos al servidor en el cuerpo del mensaje de la solicitud HTTP. La ejecución de la petición se produce justo al llamar al método `connect` de la clase `HttpClient`.

El fragmento 2.10 muestra el segundo de estos métodos.

Cabe destacar que el envío de valores de frecuencia cardíaca se realiza aproximadamente cada segundo mientras que comprobamos si existe una actualización en la ubicación del usuario cada 2 minutos, ya que la localización no es tan propensa a cambiar tan rápidamente y así realizamos un ahorro considerable de consumo.

---

```

1 public void sendLocationToWebServer(String latitude, String longitude) {
2     ...
3     JSONObject jsonObject = new JSONObject();
4     jsonObject.put("latitude", latitude);
5     jsonObject.put("longitude", longitude);
6     StringEntity se = new StringEntity(jsonObject.toString());
7     httpPost.setEntity(se);
8     httpPost.setHeader("Accept", "application/json");
9     httpPost.setHeader("Content-type", "application/json");
10    HttpResponse httpResponse = httpClient.execute(httpPost);
11    ...
12 }
```

---

Fragmento de código 2.10: Método para el envío de valores de localización al servidor web

## 2.3. Aplicación web

### 2.3.1. Entorno de desarrollo integrado

En el mundo de desarrollo web moderno, es muy común recurrir a simples editores de texto para crear y estructurar los distintos archivos que compondrán nuestra aplicación web, usar extensiones (plugins) para realizar tareas como análisis sintáctico del código en un lenguaje específico o proporcionar plantillas o *snippets* de código y hacer uso del terminal para el proceso de interpretación y compilación del código. Entre los múltiples editores de texto que se pueden adquirir mediante descarga gratuita, hemos decidido usar Atom, un proyecto de código abierto creado por el equipo de GitHub. Atom es un editor multiplataforma, es decir, lo tenemos disponible en Windows, MAC OS X y en GNU/Linux y es muy fácil de instalar. A nivel informativo Atom ha sido escrito en C++, Node.js, CoffeeScript, JS, CSS y HTML. De hecho está basado en Chromium, que es un proyecto de navegador web de código abierto, a partir del cual se basa el código fuente de Google Chrome y

utiliza la licencia libre MIT License. Permite la instalación de módulos, extensiones y temas para así aumentar sus posibilidades y es fácilmente configurable.

La figura 2.16 muestra la versión más actual de Atom en el momento de la escritura de dicha memoria.

File: heartRateMonitor-controller.js

```
 1 define(function(){
 2   'use strict';
 3
 4   var heartRateMonitorController = function ($scope, $rootScope, $window, socket){
 5     var dateObj = new Date();
 6     var month = dateObj.getUTCMonth() + 1, //months from 1-12
 7     day = dateObj.getUTCDate(),
 8     year = dateObj.getUTCFullYear(),
 9     currentDay = year + "/" + month + "/" + day,
10     w = angular.element($window);
11
12   //initializes chart dimensions in function of viewport size
13   var initialChartProperties = { //default for desktop
14     width: $window.innerWidth - 8,
15     height: $window.innerHeight - 200,
16     margin: {
17       top: 40,
18       right: 100,
19       bottom: 100,
20       left: 100
21     }
22   };
23   if ($window.innerHeight <= 480){
24     initialChartProperties.height = $window.innerHeight - 78;
25     initialChartProperties.margin.top = 15;
26   } else if ($window.innerHeight >= 640){
27     initialChartProperties.height = $window.innerHeight - 148;
28     initialChartProperties.margin.top = 15;
29   }
30   if ($window.innerWidth < 640){
31     initialChartProperties.margin.left = 50;
32     initialChartProperties.margin.right = 50;
33   }
34
35   $scope.numSamples = 10;
36   $scope.disconnected = true;
37   var realTime;
38   $scope.value = '';
39   $scope.geographicCoordinates = '';
40   $scope.currentDay = dateObj.toDateString();
41
42   $scope.checkCurrentLocation = function() {
43     $window.open('http://maps.google.com/?q=' + $scope.geographicCoordinates);
44   };
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
```

Figura 2.16: Entorno de desarrollo Atom

### 2.3.2. Estructura del código

La figura 2.17 muestra la estructura a nivel genérico de nuestra aplicación web. A continuación pasamos a explicar cada una de las partes relevantes:

**bower\_components** Esta carpeta albergará todos los módulos y librerías externas usadas para el desarrollo de la parte cliente, interfaz visual, o *frontend* de la aplicación, tales como la librería **AngularJS** para el JavaScript del lado del cliente, **d3** para la creación de las gráficas o **RequireJS** para la modularización del código.

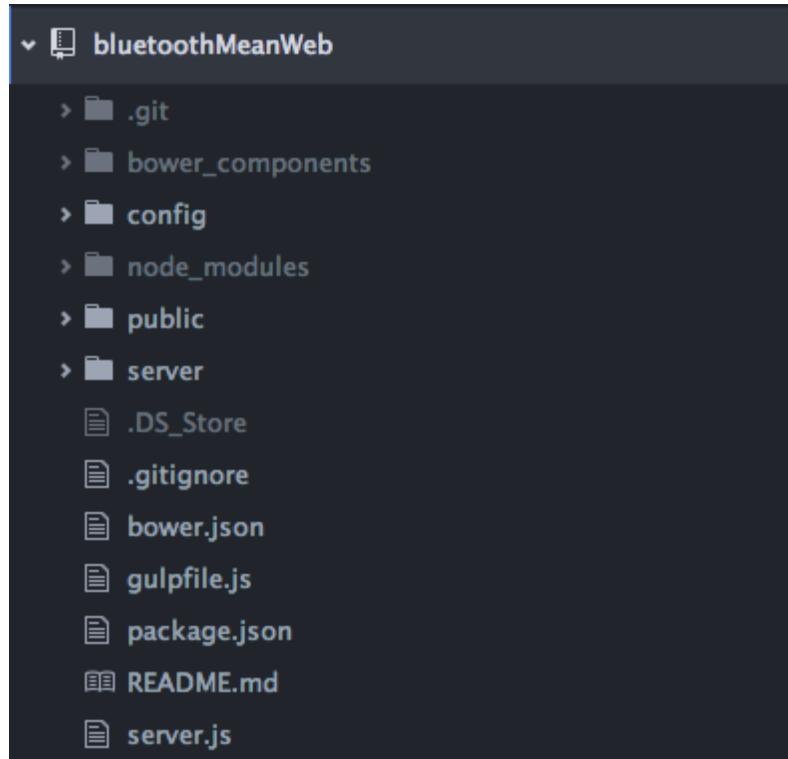


Figura 2.17: Estructura global de la aplicación web

**node\_modules** Es el equivalente a bower\_components, pero para la parte dedicada al servidor. Incluye módulos como Express para la creación del servidor web y las distintas rutas, Mongoose para el manejo de bases de datos o Socket.io para el uso de web sockets.

**package.json y bower.json** Estos archivos especifican los distintos módulos requeridos por la aplicación para que esta funcione correctamente, es decir, las dependencias en términos de módulos, así como sus correspondientes versiones. Pueden incluir también instrucciones de código (scripts), para ser ejecutados antes o después de ciertos procesos. Cualquier usuario que quiera correr la aplicación localmente deberá en primer lugar descargarse el código fuente y en segundo lugar instalar todas las dependencias mediante el uso de los comandos `npm install` o `bower install`. Nótese la flexibilidad y rapidez para correr

una aplicación usando NodeJS como plataforma de desarrollo web.

**config** Contiene archivos con credenciales para el acceso a la base de datos y al cliente de correo electrónico para el envío de notificaciones. Su visibilidad es privada.

**server.js** Es el punto de entrada de la aplicación web. Mediante el comando `node server`, se correrá la aplicación en la plataforma Node.

**server** Contiene toda la lógica del servidor, como la interacción de este con la base de datos, emisión de mensajes a través de una conexión socket y notificaciones a través de un cliente de e-mail entre otros. También proporciona una RESTful API, para que la aplicación Android pueda interactuar y enviar información al servidor.

**public** Se refiere a todos los archivos que serán entregados al cliente web, es decir cualquier navegador, una vez que accedamos a la página principal. En este caso todos los archivos html, css y Javascript del lado del cliente.

### 2.3.3. Diagrama de flujo

Tal y como hicimos con la aplicación Android en el apartado 2.2.4, en esta sección mostraremos el diagrama de flujo funcional del servidor web. Este se ilustra en la figura 2.18, el cual proporciona una visión general de cómo reacciona el servidor ante las distintas solicitudes POST recibidas y qué acciones lleva a cabo dependiendo del tipo de esta, incluyendo respuestas HTTP con código informativo al cliente, interacciones con la base de datos y lógica a ejecutar relacionada con el sistema de notificaciones.

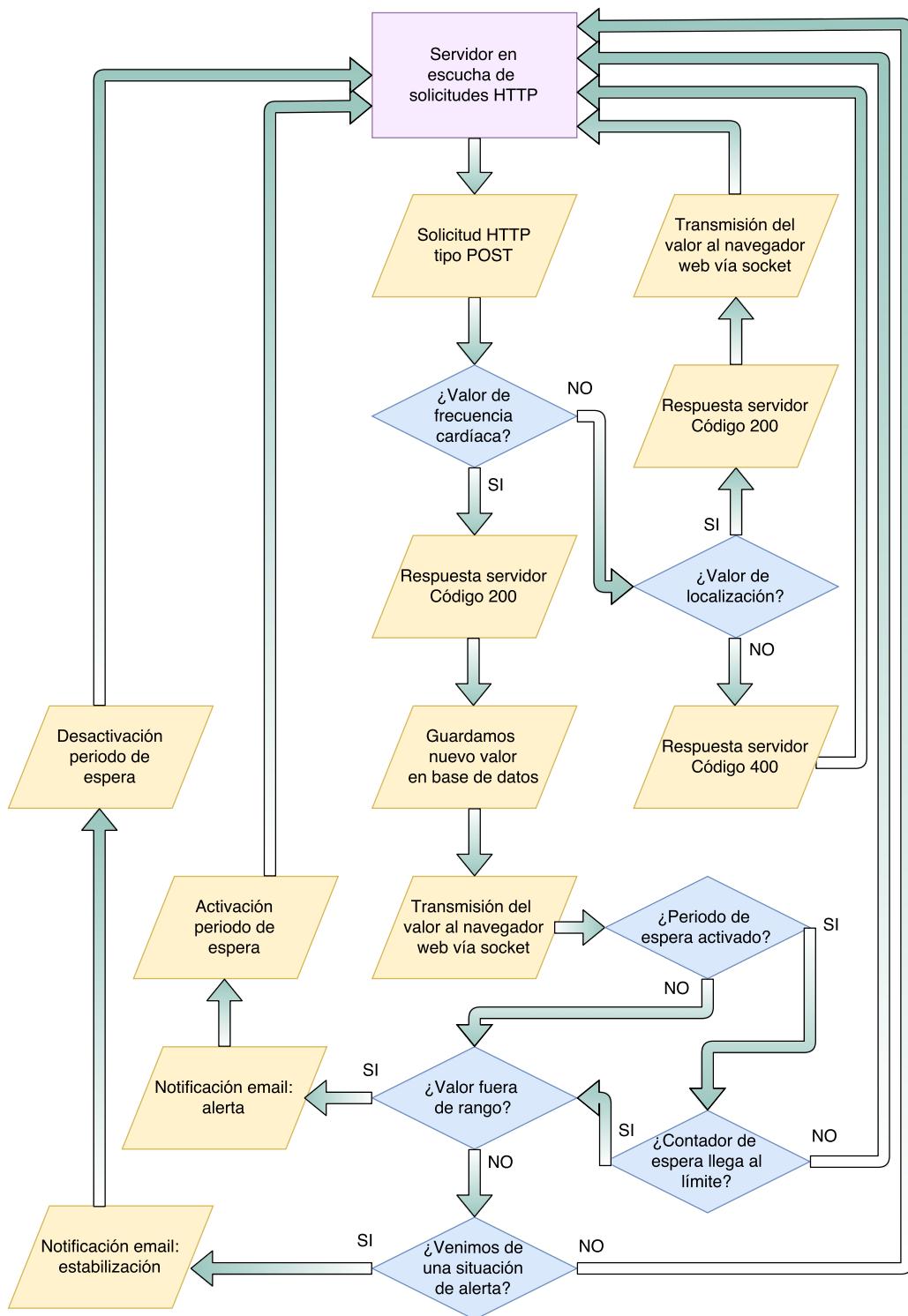


Figura 2.18: Diagrama de flujo del servidor web

### 2.3.4. Servidor

Tal y como mencionamos en la sección anterior, el punto de entrada a la aplicación es el archivo `server.js`. Vamos a desglosar el código involucrado en la configuración básica del servidor y las rutas usando el módulo Express. Éste es el mostrado en el fragmento 2.11.

---

```

1 var app = express();
2 // Sirve contenido estatico contenido en la carpeta public
3 app.use(express.static(__dirname + '/public'));
4 app.use('/bower_components',
5         express.static(__dirname + '/bower_components'));
6 app.use('/node_modules', express.static(__dirname + '/node_modules'));
7 // Modulo usado para manejar solicitudes de tipo POST
8 app.use(bodyParser());
9 // Configuracion del socket
10 var server = http.createServer(app);
11 var io = socketio.listen(server);
12 app.set('socketio', io);
13 app.set('server', server);
14 // Rutas de la aplicacion
15 app.get('/', function (req, res) {
16     res.sendFile(__dirname + '/public/index.html');
17 });
18 //API expuesta por el servidor
19 app.post('/api/heartRate', heartRateController.postHeartRateValue);
20 app.post('/api/location', heartRateController.postLocation);
21 app.get('server').listen(port, function(){
22     console.log('Server ready, listening in port 3000');
23 });

```

---

Fragmento de código 2.11: Configuración del servidor Express

En primer lugar le indicamos al servidor la localización de las distintas carpetas involucradas, para que así el navegador web pueda localizar y cargar correctamente todos los archivos estáticos. `__dirname` representa el nombre del directorio en el cual reside el programa ejecutado. `bodyParser` es un módulo de utilidad para peticiones

de tipo POST. A continuación inicializamos y abrimos el socket. El navegador web, una vez cargue la página principal se conectará a este socket permaneciendo en todo momento en escucha de eventos. Cualquier evento que ocurra en el servidor, como por ejemplo la llegada de un nuevo valor de frecuencia cardíaca, será emitido al navegador y de esta forma mantenemos una fiel representación de una conexión en tiempo real. Esta opción es mucho más rentable que el uso de la técnica conocida como *server polling*, que consiste en enviar peticiones de tipo GET al servidor desde el cliente web cada cierto tiempo para comprobar si tenemos nuevos datos disponibles. En nuestro caso involucraría enviar peticiones cada segundo, implicando una mayor sobrecarga y retardo del servidor. La API expuesta por el servidor permite a la aplicación Android enviar tanto valores de frecuencia cardíaca como de localización mediante el uso de solicitudes de tipo POST. Por último corremos el servidor Express mediante el método `listen`, de esta forma permanecerá indefinidamente a la escucha de peticiones HTTP.

La función `heartRateController.postHeartRateValue` es ejecutada cada vez que la aplicación Android envía un nuevo valor de ritmo cardíaco usando el método POST mediante el protocolo HTTP. Podemos ver el código implicado en el fragmento 2.12:

---

```

1 postHeartRateValue: function (req, res){
2     var socketio = req.app.get('socketio');
3     socketio.sockets.emit('heartRateValue', req.body);
4     saveToDatabase.fixedBuffer(req, res); // metodo para la BD
5     notifications(parseInt(req.body.heartRate), location);
6 }
```

---

Fragmento de código 2.12: Controlador POST para valores de frecuencia cardíaca

En primer lugar obtenemos la instancia de socket y enviamos el valor de frecuencia recibido a todos los clientes conectados, en este caso al navegador web. A continuación se ejecuta la función encargada de guardar dicho valor en la base de

datos. Por último entra en juego el sistema de notificaciones, el cual evaluará el valor recibido y en función de este decidirá si notificar al usuario. El sistema de notificaciones será explicado con detalle más adelante.

## Base de datos

Cada valor nuevo de frecuencia cardíaca que nos llega al servidor será almacenado en una base de datos NoSQL MongoDB, mediante el uso del módulo `mongoose`, el cual nos permite interactuar con ella mediante el uso de una API fácil e intuitiva. El modelo de objeto empleado para guardar valores de frecuencia cardíaca puede observarse en el fragmento 2.13.

---

```
1 var mongoose = require('mongoose'),
2     Schema = mongoose.Schema;
3 var databaseSchema = new Schema({
4     day: String,
5     time: String,
6     values: [
7         emitting: {type: String},
8         value: {type: String}
9     ]
10 });
11 module.exports = mongoose.model('storageModel', databaseSchema);
```

---

Fragmento de código 2.13: Modelo del objeto usado en la base de datos

Los campos día y tiempo indicarán el día y la hora exacta en la que dicho valor de frecuencia cardíaca fue recibido, mientras que el campo `emitting` nos indicará si el pulsómetro se encuentra emitiendo (conectado a la Aplicación Android) o no. La última linea de código `module.exports` representa la manera de exportar públicamente módulos en Node para que otros archivos dentro del proyecto puedan requerirlos y usarlos, facilitando la compartimentación del código y realizando una clara separación de las distintas partes en función del rol que desempeñan.

Cada documento en la base de datos almacenará 60 muestras de frecuencia cardíaca, correspondiéndose aproximadamente con una muestra cada segundo. De esta forma, cualquier usuario interesado en consultar valores dados en el pasado podrá acceder a ellos usando como clave de acceso el día y hora exacta, obteniendo los valores ocurridos en dicho minuto de tiempo.

## Notificaciones

El sistema posee un sistema de notificaciones que permite alertar por correo electrónico al usuario encargado de la monitorización, en caso de que ocurran valores anormales o peligrosos de frecuencia cardíaca. Para ello, decidimos usar el módulo `nodemailer`, el cual proporciona una API de alto nivel para el uso de clientes de correo electrónico. Los valores límite son totalmente configurables y serán representados en la gráfica mostrada en el cliente web como dos trazos horizontales.

Si el valor de frecuencia cardíaca supera el límite configurado, se enviará una notificación por correo electrónico al usuario interesado, indicándole el valor registrado así como la ubicación actual del usuario mediante un enlace al sistema de mapas de Google con la longitud y latitud exactas.

Posteriormente, se establece un periodo de espera de un minuto (también configurable). Si al cabo de un minuto el valor registrado sigue siendo peligroso se reiterará mediante el envío de otra notificación por correo electrónico, con los valores pertinentes de ritmo cardíaco y ubicación. Si por el contrario el valor registrado vuelve a caer dentro del rango de normalidad, se enviará una notificación al usuario indicando que se ha estabilizado y estamos fuera de peligro. Una vez que se han restablecido valores normales se detendrán las notificaciones a no ser que vuelvan a sucederse valores peligrosos fuera de rango, con lo cual la lógica implicada en el sistema de notificaciones actuaría de nuevo.

El fragmento 2.14 muestra la lógica implicada en caso de entrar en zona de

peligro:

```
1 module.exports = function(heartRateValue, location){  
2     if (backToNormality) counter++;  
3     if (counter == timerBuffer) counter = 0;  
4     if ((heartRateValue >= limitValue) && (counter == 0)) {  
5         backToNormality = true;  
6         smtpTransport.sendMail({  
7             from: credentials.userMail, // cliente web de notificaciones  
8             to: 'userInterested@gmail.com',  
9             subject: 'Notificacion servicio de monitorizacion del corazon',  
10            text: 'Pulsacion mayor que ' + limitValue +'!  
11                Actual ubicacion del usuario: http://maps.google.com/?q=  
12                ' + location.latitude + ',' + location.longitude  
13            }, function(error, response){  
14                ...  
15            });  
16            counter++;  
17        }  
18        if (backToNormality && heartRateValue < limitValue && counter == 0){  
19            // Envio correo indicando vuelta a la normalidad  
20            ...  
21            backToNormality = false;  
22            counter = 0;  
23        }  
24    }  
}
```

---

Fragmento de código 2.14: Funcion expuesta en heartRate-notifications.js

La variable `counter` actúa como contador de muestras. Una vez recogidas otras nuevas 60 muestras (lo que equivale aproximadamente a un minuto) volvemos a comprobar el valor registrado y ejecutamos la lógica necesaria, la cual es regida a su vez por la variable booleana `backToNormality`.

### 2.3.5. Cliente

Todos los archivos HTML, CSS y Javascript que el cliente web necesita para obtener una representación gráfica de la aplicación web se encuentran dentro de la carpetas `public` (nuestro código) y `bower_components` (las librerías externas requeridas) y serán enviadas al cliente, en este caso cualquier navegador web que acceda a nuestra página mediante la Uniform Resource Locator, o localizador de recursos uniforme (URL) adecuada, en nuestro caso <https://hearratemonitor.herokuapp.com>.

En el apartado 1.3.5 expusimos una breve presentación teórica al framework AngularJS. Aquí profundizaremos en los aspectos clave que definen este lenguaje para entender cómo este se integra e interactúa con las plantillas HTML.

#### Directivas en Angular

La cantidad de etiquetas HTML que en navegador sabe interpretar es finita, como por ejemplo las listas sin ordenar `<ul>`. Angular soluciona este problema mediante el uso de directivas, las cuales nos dan el poder de crear nuestros propios elementos HTML. De esta forma podremos crear elementos DOM (o atributos) personalizados y especificar su comportamiento, mediante el uso de etiquetas, atributos o clases CSS. Angular proporciona por defecto un conjunto de directivas predefinidas mediante el prefijo `ng-`, las cuales pueden ser por ejemplo `ng-click` para indicarle a un elemento que debe ejecutar una función cuando se haga click sobre él o `ng-show` y `ng-hide` para mostrar o ocultar elementos en función de una condición booleana. Las directivas proveen de un gran poder a Angular y son una parte de lo que hace que Angular sea declarativo.

## ¿Cómo arranca Angular?

Cuando incluimos el Javascript de Angular dentro de nuestras páginas HTML, este enlazará un método al evento `DOMContentLoaded` (esto significa que Angular se ejecutará cuando toda la página se haya terminado de cargar). Cuando la página se ha cargado, Angular visitará cada uno de los elementos del documento HTML (es decir, recorrerá cada elemento del árbol DOM). Si Angular visita un elemento que contiene una directiva, le adjuntará un comportamiento determinado.

La primera directiva que Angular necesita encontrar es la directiva `ngApp`. Angular asociará nuestra aplicación con el elemento donde encuentre la directiva `ngApp`. Pasándole un nombre como argumento a esta directiva, podemos declarar el módulo el cual será referenciado en nuestro código Javascript. ¿Por qué es importante sobre qué elemento esté nuestra aplicación? Angular funciona adjuntándose a sí mismo sobre elementos específicos y sólo dará funcionalidad a sus elementos hijos. Esto significa que podemos usar otros frameworks aparte de Angular. Esta característica hacer que sea fácil introducir Angular en subcomponentes de nuestras páginas, aunque en el caso de nuestra aplicación se usa en su totalidad.

## Módulos

Para decirle a Angular dónde se encuentra nuestra aplicación escrita en Javascript debemos usar el API `angular.module('appName', [])`. El segundo argumento representa un array de módulos como dependencias.

## Ámbitos

AngularJS utiliza ámbitos para comunicarse entre componentes, particularmente entre nuestro Javascript y nuestro HTML. Los ámbitos son el pegamento que une nuestro código y lo que muestra el navegador. Esto permite al HTML de nuestra aplicación acceder a variables definidas en un cierto ámbito en Javascript desde la

vista y así enlazarlas en nuestra presentación. Cuando decimos enlazar en Angular, nos referimos a cualquier valor que se esté mostrando en la vista. Esto ocurre principalmente con datos que se muestran en la vista usando la sintaxis `{} {}`. Un detalle importante a tener en cuenta es que este enlace funciona en ambos sentidos y nuestra vista también puede cambiar el valor de una variable.

El ámbito `$rootScope` es el ámbito de mayor nivel en el contexto de nuestra aplicación. Esto significa que desde cualquier lugar de la vista (es decir, en todos los elementos hijos que hay debajo del elemento DOM con la directiva `ngApp`) se pueden referenciar variables que se encuentren en el objeto `$rootScope`.

Sin embargo, si abusamos del objeto `$rootScope`, podemos pecar de dejar demasiada información en un único ámbito. Dejar todas las variables en un único ámbito hace que actúen como variables globales. En una aplicación grande y compleja acabaremos teniendo problemas para gestionar los datos y el código se hará confuso. Para evitar esto, Angular tiene la capacidad de organizar ámbitos mediante relaciones padres/hijos.

Al igual que los elementos DOM se incluyen unos dentro de otros, los ámbitos también pueden incluirse unos dentro de otros. Del mismo modo que las etiquetas HTML pueden tener una etiqueta padre, los ámbitos también pueden tener padre. Cuando Angular busca el valor de una variable mirará tanto en el ámbito actual como en los ámbitos ascendentes.

Angular crea ámbitos en muchas situaciones. Por ejemplo, se crea un ámbito hijo para cada controlador de nuestra aplicación, de los cuales hablaremos en la próxima sección.

Un ámbito es un objeto plano Javascript. Aunque un ámbito no tenga una funcionalidad que lo haga realmente útil, no sale de la nada. Los ámbitos son objetos Javascript como el resto de cosas de nuestro programa.

Por último, cabe destacar que AngularJS utiliza el signo del dólar \$ como prefijo

de muchas funciones y objetos que vienen con Angular. El \$ se utiliza para identificar fácilmente palabras reservadas de Angular, por lo que no conviene usarlo para nombrar nuestras variables o servicios, ya que podríamos entrar en conflicto con la propia librería.

## Controladores

Mientras que las directivas se usan generalmente en un único elemento del árbol DOM, Angular utiliza el concepto de controladores como un elemento con el que organizar grupos de elementos del árbol DOM. Un controlador es un trozo de código que define la funcionalidad de una parte de la página. Usando la API de Angular podemos crear un nuevo controlador de la forma vista en el fragmento 2.15.

---

```
1 angular.module('myApp')
2   .controller('HomeController', function($scope) {
3     // Tenemos acceso a este nuevo
4     // objeto scope, donde podemos colocar
5     // datos y funciones para poder interactuar con el
6   });
```

---

Fragmento de código 2.15: Ejemplo de controlador Angular en Javascript

Una vez definido el controlador podemos ubicar este en la vista usando la directiva `ng-controller` de la forma mostrada en el fragmento 2.16.

---

```
1 <div ng-controller='HomeController'>
2   <!--
3   Aqui tenemos acceso al objeto
4   scope definido en el HomeController
5   -->
6 </div>
```

---

Fragmento de código 2.16: Referencia del controlador Angular en la vista

En lugar de colocar toda nuestra funcionalidad en el objeto `$rootScope`, podemos colocarla en el objeto `$scope` de `HomeController` y mantener nuestro `$rootScope` limpio.

El beneficio de crear un ámbito nuevo es que nos permite ser capaces de mantener las variables y los datos ubicados en una parte específica de la página. De este modo podemos definir una variable nueva sin contaminar o entrar en conflicto con otras partes de la página.

Técnicamente, cada vez que creamos un nuevo controlador, Angular crea un nuevo objeto `$scope` por debajo del ámbito que se encuentre encima suyo (en este caso `$rootScope`).

Utilizamos el término “por debajo” porque los ámbitos están jerarquizados. Cuando Angular trata de buscar el valor de una variable, si no la encuentra en un ámbito la buscará en el ámbito superior en la jerarquía (hasta que alcance `$rootScope`)

## Angular y RequireJS

Las aplicaciones de Javascript cada vez han aumentado mas de tamaño, lejos quedan esos pequeños scripts que validaban formularios o añadían unos pequeños efectos a nuestras páginas web. Hoy en día podemos hablar de bases de código escritas en su mayor parte en Javascript, lo cual genera problemas a la hora de organizar nuestro código, ya que Javascript por defecto no trae ninguna manera de declarar explícitamente módulos en una aplicación.

La solución hasta ahora había sido añadir muchas etiquetas `<script>` a lo largo de las vistas HTML, intentando así cargar en orden los distintos archivos que conforman el código. Esta solución rápidamente quedó obsoleta. Sin saber explícitamente que dependencias tiene cada archivo, dependemos de nuestra memoria a la hora de declarar el orden de carga de estos archivos.

Otra solución que se planteó fue la de desarrollar con el código repartido en dis-

tintos archivos y usar alguna herramienta para concatenar el código de los distintos archivos en uno, de manera que solo tengamos que cargar un archivo y tengamos las ventajas de ambas formas de desarrollo. El problema con este acercamiento es que, en el entorno de desarrollo, añadir un paso intermedio a la hora de poder probar la aplicación no siempre es beneficioso, además de que puede crear problemas de colisión de nombres, ya que Javascript no provee de ninguna manera de declarar explícitamente nombres de espacio o *namespaces*.

Por ello, y hasta que se hagan oficiales las nuevas especificaciones de Javascript (las cuales intentarán solucionar estos problemas con una implementación nativa de módulos), se creó la especificación Asynchronous Module Definition, o definición asíncrona de módulos (AMD), la cual intenta solucionar ambos problemas, es decir, definir módulos y declarar explícitamente las dependencias necesarias para cada módulo.

RequireJS es una librería JavaScript basada en AMD que nos permite aislar mediante módulos los componentes de nuestra aplicación cliente y resolver las dependencias de estos mismos. Funciona muy bien en conjunto con AngularJS, dado su carácter modular.

Una vez que arrancemos la aplicación Angular, RequireJS construirá un árbol de dependencias entre los distintos módulos y se encargará de injectar las clases o módulos requeridos en los distintos archivos Javascript, sin tener nosotros que preocuparnos de ello, mejorando el rendimiento y la calidad de nuestro código.

El árbol de dependencias de módulos en el caso de nuestra aplicación es el mostrado en la figura 2.19. `heartRateMonitor` es el submódulo principal que engloba toda la lógica de la aplicación y el cual presenta tres dependencias, las cuales listamos a continuación:

`btford.socket-io` representa el módulo que integra socket.io (a su vez, otra librería JavaScript que implementa Web Sockets) dentro del mundo Angular, permitiendo

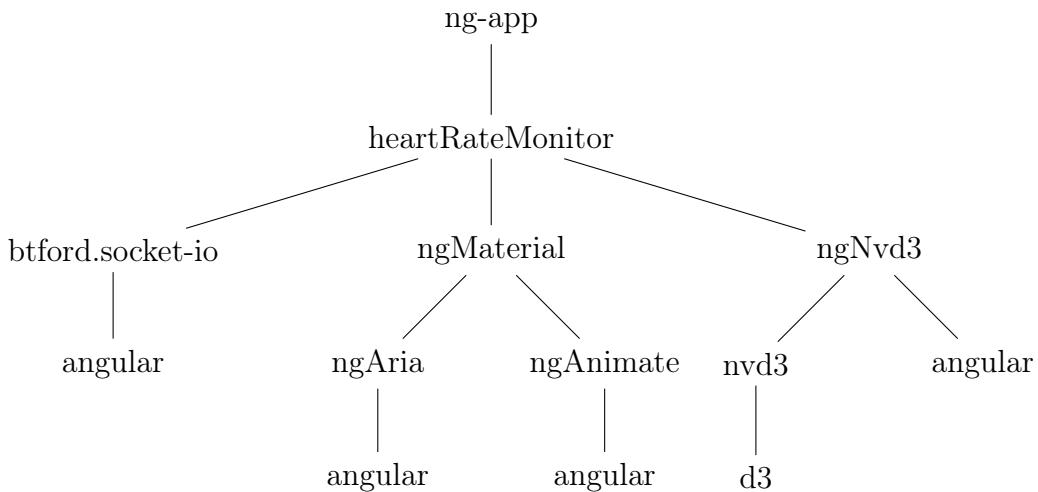


Figura 2.19: Árbol de dependencias en la aplicación Angular

tiéndonos establecer comunicaciones en tiempo real con el servidor.

**ngMaterial** librería que implementa el concepto visual de Diseño Material o Material Design, siguiendo las nuevas recomendaciones de diseño proporcionadas por Google. Depende a su vez de **ngAria**, la cual proporciona soporte para accesibilidad mediante la habilitación de atributos ARIA y el uso de tecnologías asistidas usadas por personas con algún tipo de discapacidad y **ngAnimate**, la cual proporciona soporte en Angular para el uso de animaciones.

**ngNvd3** es un módulo que permite integrar en Angular gráficas reutilizables basadas en la librería nvd3, gracias al uso de las directivas que implementa. Esta depende de d3, que es una librería para manipular documentos basados en datos, como elementos Scalable Vector Graphics, o gráficos vectoriales escalables (SVG). **ngNvd3** será el módulo usado para la representación de una gráfica en tiempo real con valores de frecuencia cardíaca.

## Vista

Una vez llevada a cabo una explicación técnica más detallada de Angular, podemos pasar a desglosar los distintos componentes del paradigma MVC, en términos de nuestra aplicación. El fragmento de código 2.17 representa la vista HTML de nuestra aplicación web. Cabe decir que es la única vista que contiene.

El controlador Angular de nuestra aplicación se encuentra enlazado al elemento DOM `<body>`, que no es más que el cuerpo del documento HTML. Por tanto, tenemos un único controlador, `heartRateMonitor.controller`, asociado al elemento padre del contenido, que nos permitirá aplicar directivas Angular a todos sus elementos hijos, y el cual crea un nuevo ámbito o entorno de variables aisladas del entorno global, las cuales podrán ser accedidas también por los elementos hijos de las vistas usando la notación `{{ }}`.

Los elementos `<md-toolbar>` y `<md-content>` son elementos DOM personalizados proporcionados por la librería `ngMaterial`, los cuales permiten configurar fácilmente la predisposición de los elementos en la barra de acción y contenido mediante atributos HTML, los cuales serán compilados a código CSS. Las variables enlazadas `currentDay` y `value` representan respectivamente la fecha actual y el valor de frecuencia cardíaca recibido en cada momento. El elemento `nvd3` es otro elemento DOM personalizado agregado por la librería `ngNvd3`, el cual representa la gráfica de tipo *line-chart* o gráfica de línea, que usaremos para mostrar los últimos valores recibidos de frecuencia cardíaca, así como el actual. Por último la etiqueta `<script>` es el único script necesario en nuestro HTML, actuando como el punto de entrada que la librería RequireJS necesita, para efectuar así la construcción del árbol de dependencias previo al arranque de AngularJS.

```
1 <html>
2     <head>
3         <!-- Titulo y carga de los archivos externos css -->
4     </head>
5     <body ng-controller="heartRateMonitor.controller">
6         <div layout="column" layout-fill>
7             <md-toolbar>
8                 <!-- Contenido de la barra de herramientas superior -->
9             </md-toolbar>
10            <md-content layout-padding layout-fill flex
11                layout="column" style="padding-top: 24px">
12                <div class="heartRate-value">
13                    <span style="padding-right: 10px">{{currentDay}}</span>
14                    <i class="fa fa-heart-o fa-2x" style="color: pink;">
15                        ng-show="disconnected"></i>
16                    <i class="fa fa-heartbeat fa-2x" style="color: #E91E63;">
17                        ng-hide="disconnected"></i>
18                    <span style="padding: 0px 10px 0px 10px;
19                        font-size: 24px;">{{value}}</span>
20                </div>
21                <nvd3 options="options" data="data"
22                    config="config" api="api"></nvd3>
23            </md-content>
24        </div>
25        <!--Punto de entrada al codigo javascript'-->
26        <script type="text/javascript"
27            src="../bower_components/requirejs/require.js"
28            data-main="js/main.js"></script>
29    </body>
30 </html>
```

---

Fragmento de código 2.17: Vista principal index.html

## **Controlador y modelo**

El controlador y subsecuente modelo de objetos se encuentran definidos en el archivo `heartRateMonitor-controller.js`. En él se alberga toda la lógica JavaScript del lado del cliente, como las opciones, configuración y datos de la gráfica, los cuales se definen mediante objetos JSON.

También se encuentra el manejo de la conexión socket del lado del cliente, de forma que el controlador se encuentra suscrito a dos eventos. El primero de ellos es la llegada de nuevos valores de frecuencia cardíaca provenientes del servidor. Cuando esto ocurre se añade el nuevo valor al final de la gráfica y se elimina el valor más antiguo, simulando un comportamiento de tiempo real, ya que la gráfica transmite la sensación de ir desplazándose con el tiempo. El segundo de ellos es la llegada de un nuevo valor de localización. Esto permitirá al usuario acceder a la ubicación actualizada en un mapa mediante el pulsado del botón “Localización” pertinente.

Por último decir que el controlador también se encarga de manejar el dinamismo de la página en cuanto a cambios de tamaño de pantalla se refiere. De esta forma, si estamos en un ordenador portátil típico con monitor de 15 pulgadas y tenemos el navegador en pantalla completa, mostraremos 10 muestras en la gráfica, mientras que si nos encontramos en un dispositivo móvil, como puede ser un teléfono inteligente de 4 pulgadas, mostraremos 2 o 4 muestras, en función de la orientación de este. Esto se verá reflejado en el siguiente capítulo.



# Capítulo 3

## Plan de pruebas y verificación

### 3.1. Funcionamiento de la aplicación

Una vez ejecutada la aplicación Android, se nos solicitará la activación de Bluetooth en caso de que este no estuviera previamente activado. Una vez que Bluetooth se encuentra en funcionamiento podemos empezar a escanear dispositivos BLE en el radio de alcance de nuestro teléfono móvil mediante el pulsado del botón en la barra de acción, el cual presenta una imagen con el logotipo de Bluetooth emitiendo señal, tal y como se muestra en la figuras 3.1 y 3.2.

Al pulsar en el dispositivo encontrado justo después del escaneo, en este caso nuestro pulsómetro comercial adquirido, una nueva actividad aparecerá en pantalla mostrándonos la información relativa al sensor, así como el estado de la conexión, que en un principio será desconectado, tal y como se puede apreciar en la figura 3.3. Para conectarnos a él, debemos pulsar en el botón “Conectar”. Una vez que la conexión GATT cliente-servidor se ha establecido, la interfaz de usuario se actualizará, mostrando los servicios ofrecidos por el sensor, así como las características pertenecientes a cada uno de estos servicios.

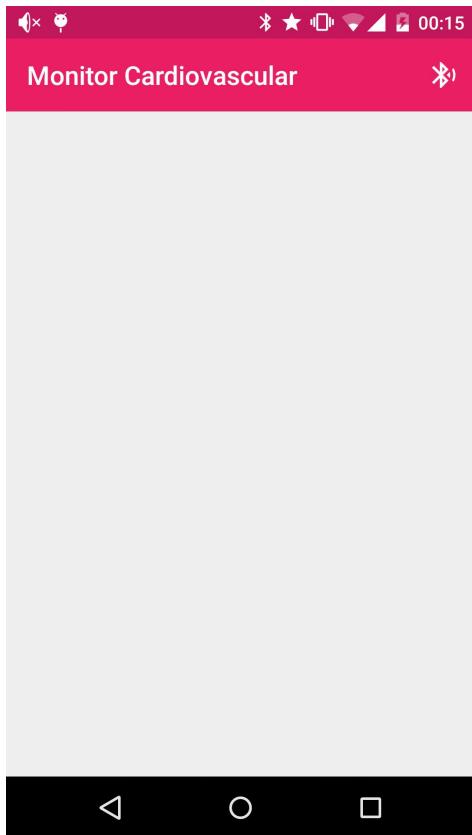


Figura 3.1: Pantalla inicial

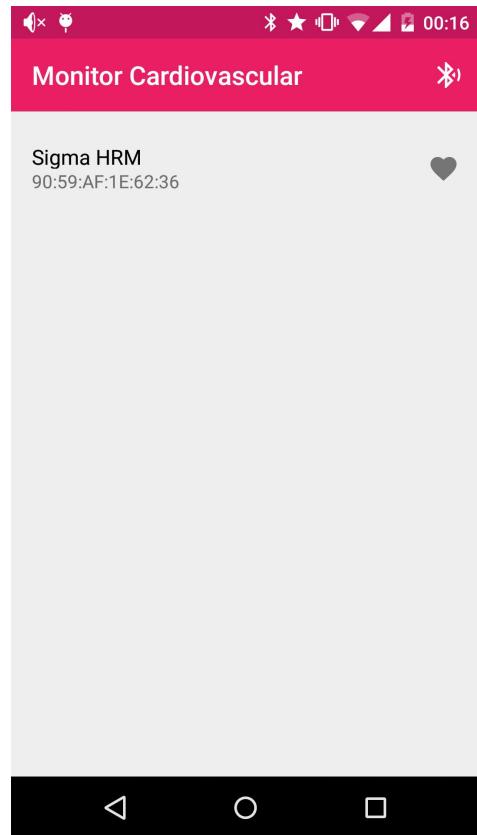


Figura 3.2: Dispositivos encontrados

En este caso nos encontramos con dos servicios: el servicio “Servicio Cardiovascular”, que presenta las características “Medida del ritmo cardíaco”, “Ubicación del sensor” y “Punto de control” y el servicio “Servicio de batería”, que presenta una característica que nos permite comprobar el porcentaje de batería restante del pulsómetro. Para empezar a recibir datos de frecuencia cardíaca debemos hacer click en la característica “Medida del ritmo cardíaco”. Esto activará el cliente HTTP, de forma que enviaremos a nuestro servidor web cada valor que nos llega del pulsómetro. La interfaz de usuario también reflejará cada valor recibido en el campo “Datos”. La figura 3.4 muestra dicho proceso.

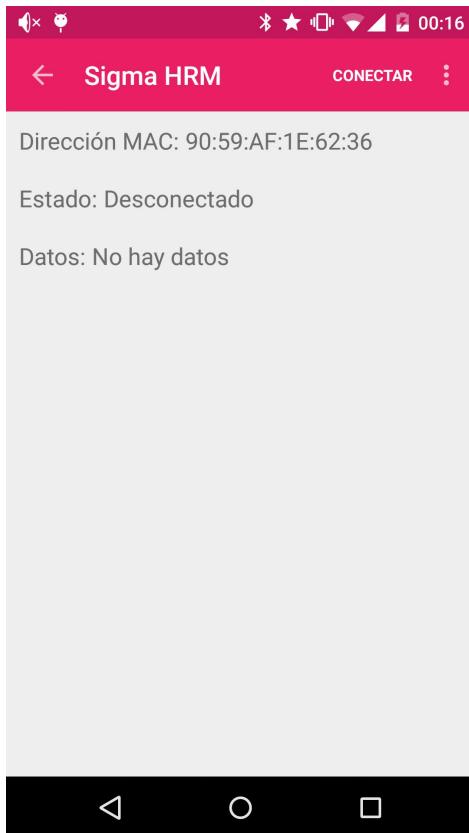


Figura 3.3: Conexión GATT sin establecer

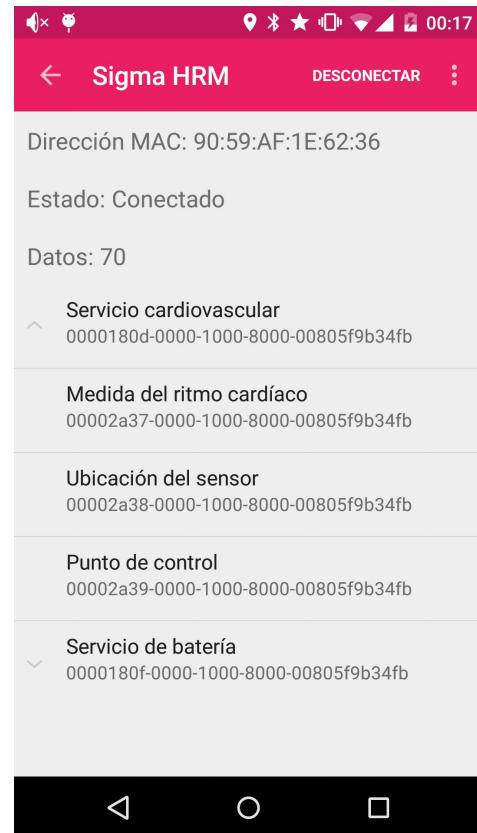


Figura 3.4: Conexión GATT establecida

Es importante mencionar que puesto la conexión GATT cliente-servidor es manejada mediante un servicio en segundo plano, podemos salir de la aplicación y realizar otras actividades, tales como abrir el navegador web o lanzar otras aplicaciones y esto no afectará al comportamiento de la aplicación. Para desconectarnos del pulsómetro, solo debemos abrir la aplicación de nuevo y pulsar en el botón “Desconectar”. Esto finalizará el servicio en segundo plano, la conexión GATT y hará que se liberen los recursos usados.

Mientras que estemos desconectados del pulsómetro, nuestro cliente web de monitorización mostrará el estado *Desconectado* en todo momento, tal y como se observa

en la figura 3.5

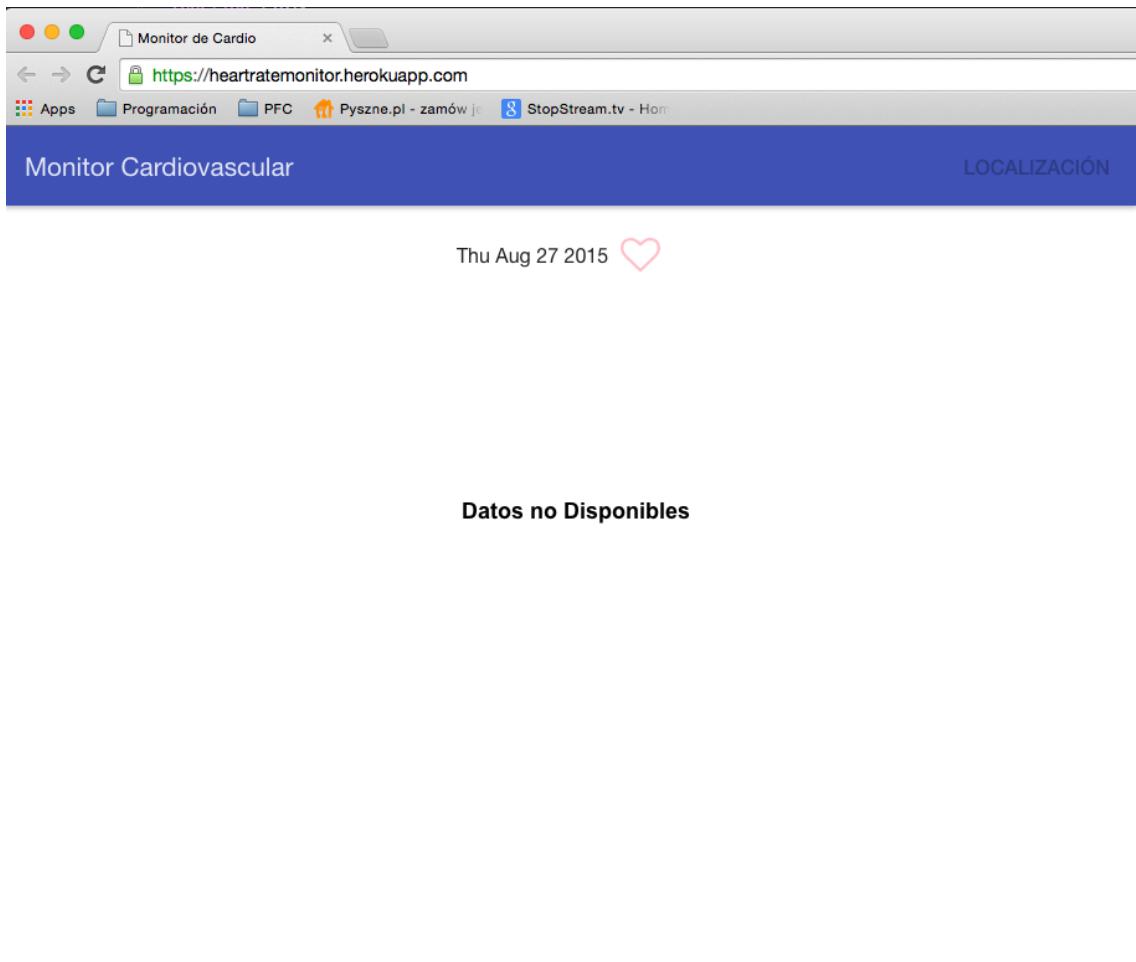


Figura 3.5: Cliente web cuando el pulsómetro está desconectado

En cuanto conectamos con el pulsómetro y seleccionamos la característica de “Medida del ritmo cardíaco” en nuestra aplicación Android, los valores empiezan a llegar al servidor web, el cual establece una conexión permanente Web Socket con el cliente web, permitiendo una transmisión en tiempo real de dichos valores. La figura 3.6 representa el cliente web en dicho estado.

En la parte superior, justo debajo de la barra de navegación azul, se muestra la fecha actual, un ícono representando la palpitación del corazón y el valor actual

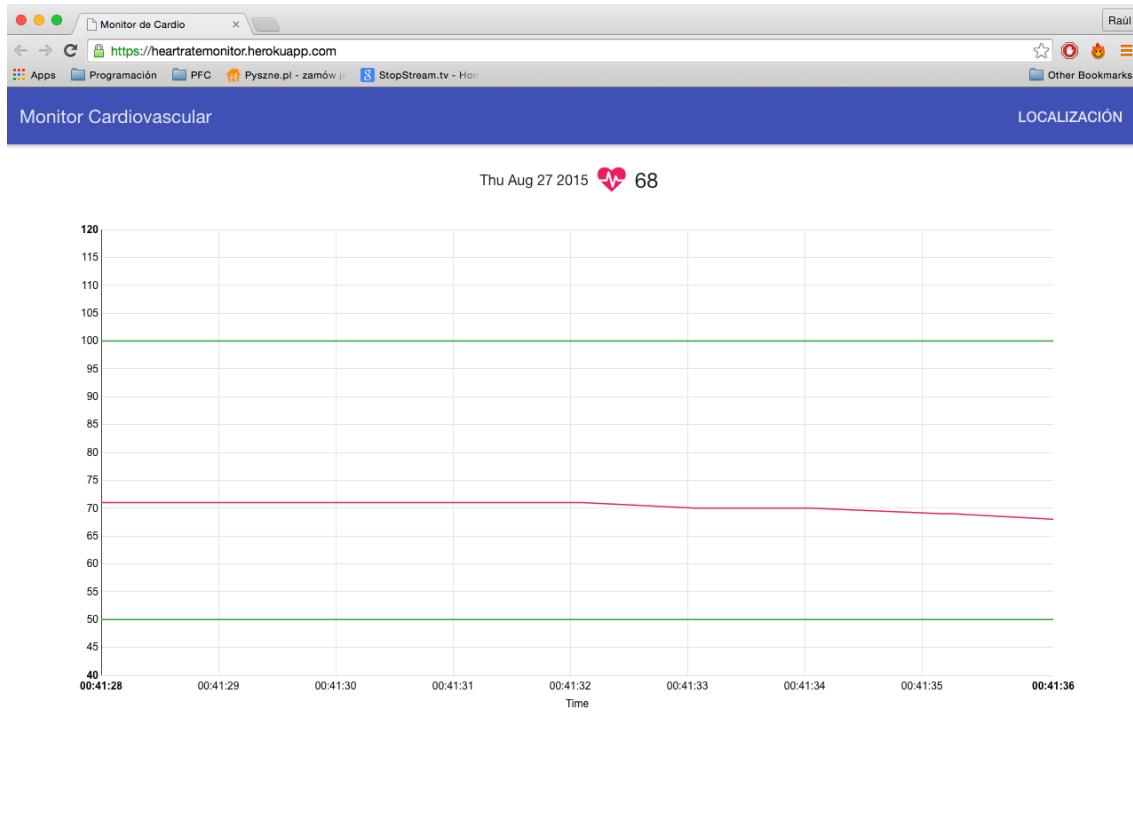


Figura 3.6: Cliente web cuando el pulsómetro está conectado

de frecuencia cardíaca recibido. Justo debajo aparece una gráfica en tiempo real mostrando el valor actual y los últimos valores recibidos con un trazo de color rosa. Las 2 líneas horizontales verdes, las cuales son fijas, delimitan un rango de valores estable (totalmente configurable en función del usuario a monitorizar). Esto está intrínsecamente relacionado con nuestro sistema de notificaciones, ya que se notificará por correo electrónico al usuario encargado de la monitorización siempre que nos topemos con valores fuera de rango, entrando en funcionamiento la lógica explicada en la sección 2.3.4, concretamente en la parte de notificaciones.

Esto se puede apreciar en las figuras 3.7 y 3.8:

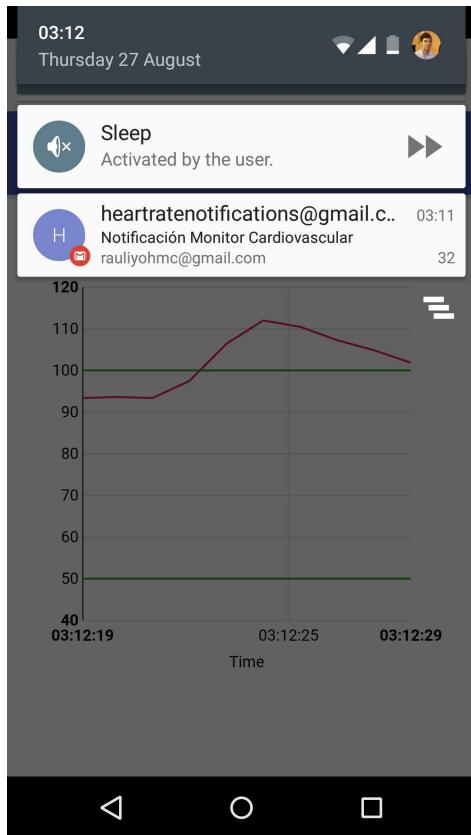


Figura 3.7: Notificación por valor fuera de rango

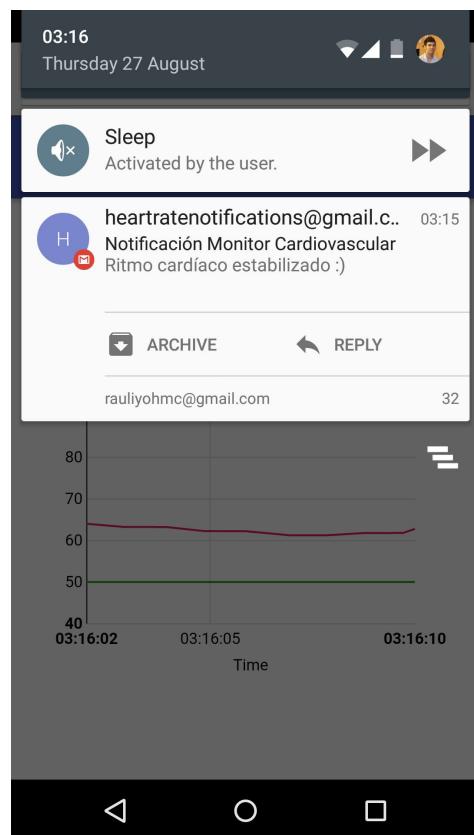


Figura 3.8: Notificación de estabilización

Para consultar la localización exacta del usuario en un determinado instante, debemos de pulsar el botón “Localización”. Esto nos abrirá automáticamente la aplicación de Mapas de Google, determinando la localización en el mapa, así como la presentación de las coordenadas geográficas, como se puede ver en las figuras 3.9 y 3.10:

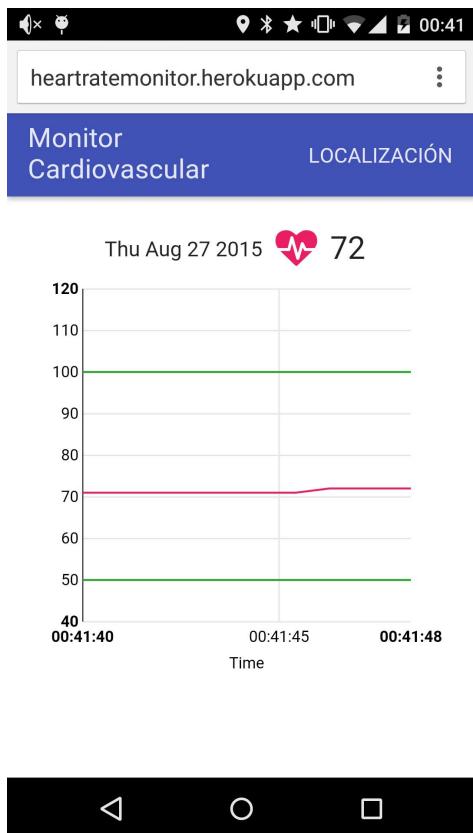


Figura 3.9: Cliente web en un dispositivo móvil

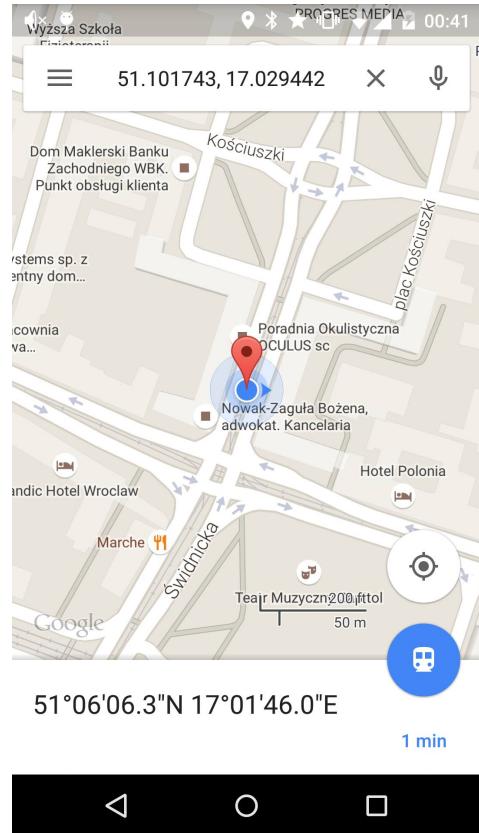
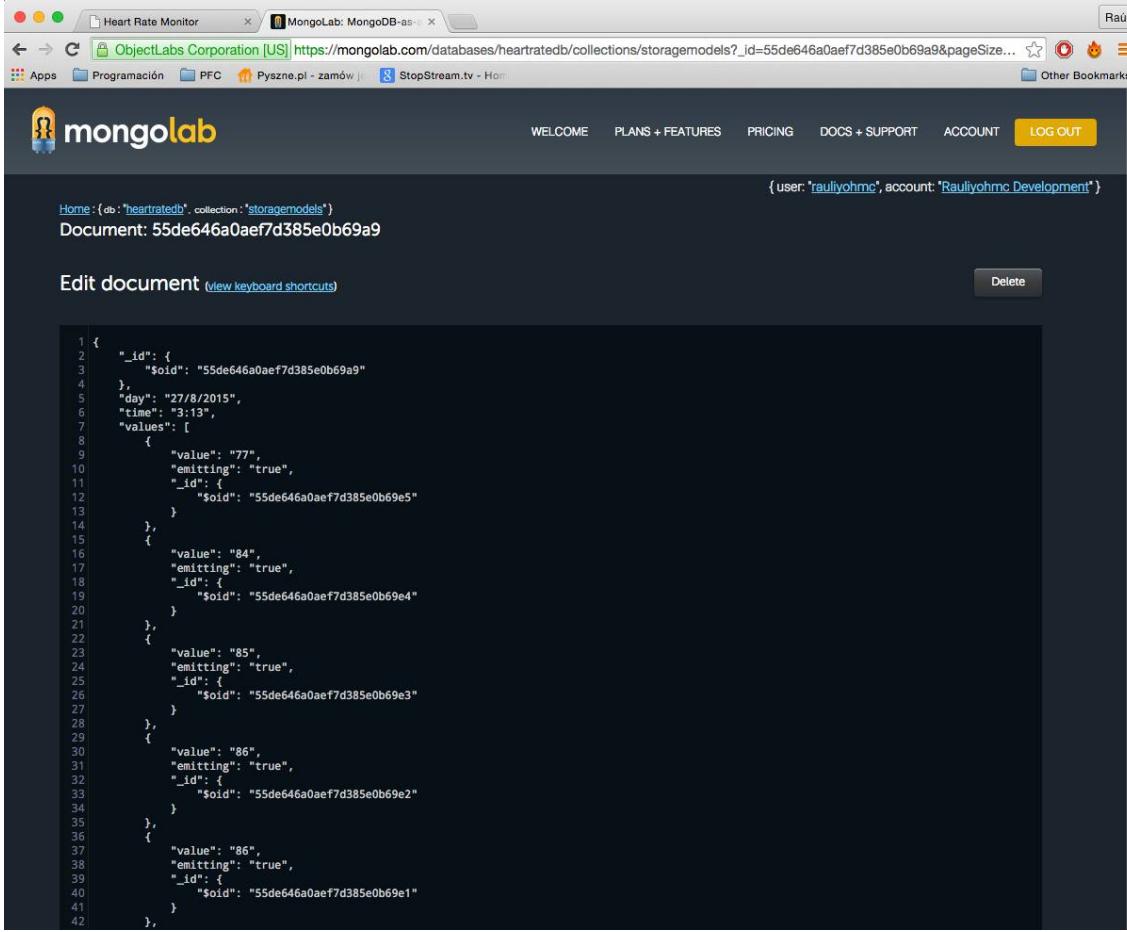


Figura 3.10: Ubicación del usuario en el servicio de mapas de Google

Por último y no menos importante, usamos una base de datos para registrar y guardar cada uno de los valores de frecuencia cardíaca que el servidor web recibe. Para ello hemos recurrido a Mongolab, un servicio de almacenamiento de datos en bases del tipo MongoDB en la nube, el cual nos proporciona de forma gratuita 512 MB de espacio. La figura 3.11 muestra un extracto de uno de los documentos almacenados, conteniendo muestras pertenecientes al momento Jueves 27 de Agosto, 3:13 a.m.



The screenshot shows a browser window with the title 'MongoLab: MongoDB-as-a-Service'. The URL is 'ObjectLabs Corporation [US] https://mongolab.com/databases/heartratedb/collections/storage...'. The page displays a JSON document with line numbers from 1 to 42. The document structure is as follows:

```

1 {
2   "_id": {
3     "$oid": "55de646a0aef7d385e0b69a9"
4   },
5   "day": "22/8/2015",
6   "time": "3:13",
7   "values": [
8     {
9       "value": "77",
10      "emitting": "true",
11      "_id": {
12        "$oid": "55de646a0aef7d385e0b69e5"
13      }
14    },
15    {
16      "value": "84",
17      "emitting": "true",
18      "_id": {
19        "$oid": "55de646a0aef7d385e0b69e4"
20      }
21    },
22    {
23      "value": "85",
24      "emitting": "true",
25      "_id": {
26        "$oid": "55de646a0aef7d385e0b69e3"
27      }
28    },
29    {
30      "value": "86",
31      "emitting": "true",
32      "_id": {
33        "$oid": "55de646a0aef7d385e0b69e2"
34      }
35    },
36    {
37      "value": "86",
38      "emitting": "true",
39      "_id": {
40        "$oid": "55de646a0aef7d385e0b69e1"
41      }
42  },

```

Figura 3.11: Servicio de almacenamiento MongoLab en la nube

## 3.2. Pruebas realizadas

### 3.2.1. Comparación con electrocardiograma

Para cerciorarse de que los valores de frecuencia cardíaca mostrados en la aplicación se ajustan a la realidad, es necesario comparar los valores obtenidos por nuestro pulsómetro pectoral con un aparato de medición calibrado. Para ello, recurrimos a la ayuda de una doctora ejerciendo en la ciudad de Wroclaw (Polonia), la cual amablemente se ofreció voluntaria para realizar medidas de nivel de frecuencia cardíaca



Figura 3.12: Sigma R1 Blue Comfortex+

emitidas por un electrocardiograma y así comparar los resultados con las medidas emitidas por el pulsómetro.

El pulsómetro utilizado ha sido el Sigma R1 Blue Comfortex+, el cual se muestra en la figura 3.12. En el envoltorio podemos observar que afirma tener precisión de ECG (electrocardiograma).

Realizamos el estudio de 3 distintas situaciones, con todos los electrodos del electrocardiógrafo conectados a través del cuerpo y el pulsómetro Sigma comercial atado al pecho mediante su respectiva correa, con los dos electrodos situados en el pecho. Los resultados obtenidos pueden verse reflejados en la tabla 3.1:

Actividad	Rango	Error (%)
<b>Reposo</b>	50-70 bpm	5.8 %
<b>Actividad Ligera</b>	80-100 bpm	1.5 %
<b>Actividad intensa</b>	150-170 bpm	0 %

Tabla 3.1: Tabla de comparacion de mediciones

En principio empezamos midiendo el latido del corazón en situación de reposo, mediante una posición estática estando de pie. Se pudo comprobar que existe una pequeña variación en la medida del pulsómetro relativa al electrocardiógrafo profesional, en torno a los  $+3/4$  bpm (latidos por minuto). Posteriormente empezamos una actividad cardiovascular ligera, alcanzando valores en el rango de las 80-100 pulsaciones por minuto y obtuvimos una mayor precisión, en torno a los  $+1/2$  bpm. Por último realizamos un trote intenso sobre el sitio hasta alcanzar valores cercanos a las 160 pulsaciones por minuto. De manera sorprendente, las medidas en este caso coincidían plenamente con las proporcionadas por el electrocardiograma.

Concluimos pues con unas medidas bastante exactas en todos los casos, llegando a igualar la medición del electrocardiograma en situaciones de ejercicio intenso. Esto se debe a que, en esencia, el fundamento del pulsómetro comercial de pecho es el mismo que el del electrocardiograma, registrando la actividad eléctrica del corazón que se produce en cada latido cardíaco.

Como contrapartida, existen sensores ópticos del ritmo cardíaco, los cuales funcionan mediante la emisión de luz LED a los capilares, y midiendo la frecuencia de bombeo de la sangre. Estos proveen una precisión alta en situaciones de reposo e inmovilidad, pero no son capaces de medir la frecuencia cardíaca en movimiento. Y en situaciones en las que paramos a medir el latido del corazón después de un intenso ejercicio proveen una precisión bastante baja, con errores en la medida de hasta el 40 %.

### 3.2.2. Consumo de energía

Hemos realizado una estimación del consumo de batería medio que nuestra aplicación Android exige. Para ello, hemos usado el dispositivo conectado y emitiendo al servidor durante una hora en reposo, y durante una hora haciendo ejercicio en el exterior. El teléfono móvil utilizado ha sido un Nexus 5. Durante el periodo en reposo, utilizando conexión inalámbrica a Internet, el teléfono disminuyó su batería un 5 % en una hora, lo que nos da una estimación de duración de unas 20 horas, siempre que no usemos otras aplicaciones, ya que aquí solo consideramos el consumo del servicio funcionando en segundo plano. Durante el periodo de ejercicio, y usando la red LTE de un plan de pago, la batería bajó en un 12 % en una hora, lo que nos da una duración aproximada de 8.5 horas.

Los resultados obtenidos son sorprendentes y nos brindan un sistema capaz de aguantar medio día sin tener que recargar la batería, todo ello considerando todos los recursos utilizados mientras el servicio funciona en segundo plano, que son ni más ni menos el servicio de localización de Google, la conexión GATT cliente-servidor y los envíos de datos al servidor web a través del protocolo HTTP. Este bajo consumo se ve justificado por el hecho de que el protocolo BLE consume muy poca energía. La poca carga de datos que las peticiones HTTP acarrean y el alto intervalo de escaneo del servicio de localización (comprobando cada 2 minutos si existe una nueva posición del usuario) también contribuyen a dicho propósito.

### 3.2.3. Retardo del sistema

Cuando hablamos de retardo del sistema, nos referimos al tiempo que ha pasado desde que recibimos una nueva muestra de frecuencia cardíaca en nuestra aplicación puente Android, hasta que dicho valor aparece reflejado en la aplicación web. Obviamente esto está influenciado en parte por la velocidad de conexión a Internet en la

cual nos encontremos sumergidos. En cualquier caso hoy en día las conexiones que manejamos a diario, ya sea en casa usando una conexión ADSL de alta velocidad, o nos encontremos en la calle usando una conexión HDSPA o LTE proporcionada por el proveedor de servicios contratado nos garantizan un rápido acceso a Internet.

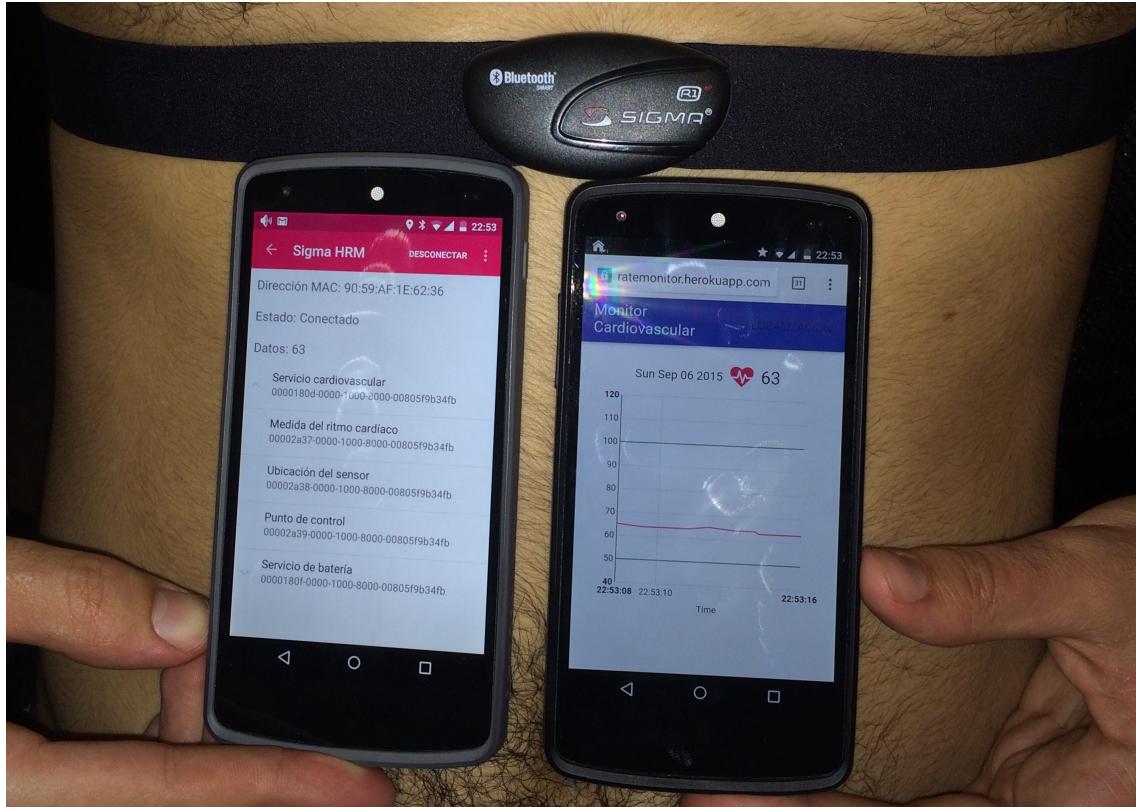


Figura 3.13: Captura del sistema en funcionamiento

Hemos puesto nuestro sistema a prueba en el caso extremo para estudiar su viabilidad. Para ello hemos recurrido a Heroku, un servicio gratuito de almacenamiento de aplicaciones web en la nube. El servidor que se nos ha asignado se encuentra en Virginia, Estados Unidos, con lo cual vamos a experimentar una conexión Wroclaw-Virginia-Wroclaw. El primer paso se trata de enviar los datos desde nuestra aplicación Android a nuestro servidor web localizado en Estados Unidos y

posteriormente el servidor deberá entregar dichos datos a nuestro navegador web mediante la emisión de eventos utilizando una conexión de tipo socket.

La figura 3.13 muestra una captura tomada durante la realización de la prueba. Como se observa, el móvil de la izquierda muestra la aplicación Android, que recibe la información del pulsómetro y la envía al servidor, mientras que el móvil de la derecha es el que muestra la información proporcionada por el servidor en el navegador web, pudiéndose ver como los valores de pulsación coinciden.

Los resultados son asombrosos, es prácticamente imposible apreciar un retraso desde que la aplicación Android muestra el valor recibido en el campo de datos, hasta que este se ve reflejado en la web. Podríamos estimar un retardo en torno a los 100 milisegundos siendo algo conservadores, con lo cual se cumple con creces el objetivo de simulación de tiempo real del sistema.

Para finalizar, hemos incluido un resumen de las pruebas realizadas en la figura 3.14.

### 3.3. Realización de los objetivos

#### 3.3.1. Aplicación Android

##### Capacidad de detectar sensores BLE próximos al dispositivo

La aplicación cumple con el objetivo de ser capaz de escanear sensores con pila de reloj que usan el protocolo BLE para emitir información, ya sean monitores de frecuencia cardíaca u otra orientación médica, emisores de publicidad, etcétera.

##### Capacidad de conectarse a cualquier sensor BLE registrado

La aplicación permite conectarnos a cualquiera de ellos para así consultar los servicios GATT ofrecidos, junto con cada una de sus características GATT, tal y

Objetivo	Verificación
Precisión de ECG de los valores medidos por el pulsómetro, para valores de pulsación medios y altos	
Bajo consumo de batería de la aplicación Android, cuando esta se encuentra transmitiendo datos de frecuencia cardíaca y localización al servidor web a través de HTTP	
Retardo del sistema menor que 1s, desde que el teléfono Android recibe los valores de pulsación enviados por el sensor, hasta que estos son visualizados por el cliente o navegador web	

Figura 3.14: Resumen de las pruebas realizadas

como se observa en la figura 3.4.

#### Capacidad de leer características GATT ofrecidas por el sensor

La aplicación es capaz de leer el valor contenido en una cierta característica GATT, mediante la interpretación interna de sus descriptores, proporcionando un valor legible para el ser humano, como puede ser el valor de frecuencia cardíaca (mostrado en el campo “datos” de la figura 3.4), la posición del sensor en el cuerpo, o el porcentaje de batería restante del sensor.

**Capacidad de enviar los parámetros relevantes a través del protocolo HTTP a nuestro servidor web**

La aplicación cumple con el objetivo de enviar al servidor web los valores de ritmo cardíaco y localización obtenidos a través de los servicios que corren en segundo plano.

**3.3.2. Aplicación web****Capacidad de determinar si el sensor está conectado al sistema**

La aplicación cumple con el objetivo de reflejar en todo momento el estado de la conexión del sensor. El estado desconectado se manifiesta mediante la iconografía de un corazón hueco y mediante la ausencia de gráfica en el centro de la pantalla, tal y como se aprecia en la figura 3.5. El estado conectado se representa mediante la iconografía de un corazón palpitando.

**Capacidad de mostrar la frecuencia cardíaca actual en tiempo real**

La aplicación es capaz de mostrar el valor de frecuencia cardíaca en tiempo real, mediante la indicación numérica justo a la derecha del ícono de corazón latiendo, tal y como se aprecia en la figura 3.6.

**Capacidad de mostrar la evolución de la frecuencia cardíaca mediante el uso de una gráfica dinámica**

La aplicación cumple con el objetivo de dibujar los valores que van llegando en una gráfica dinámica, la cual mostrará el valor actual junto con los más recientes (cuyo número dependerá del tamaño de la pantalla). Esta gráfica simula un comportamiento en tiempo real, desplazando valores a la izquierda conforme nuevos valores

llegan. También presenta dos trazos horizontales configurables representando unos posibles límites dañinos o peligrosos.

### **Capacidad de guardar los datos de frecuencia cardíaca en una base de datos**

La aplicación es capaz de guardar en una base de datos todo registro de frecuencia cardíaca recibido, ordenándolos en documentos de 60 muestras cada uno, representando aproximadamente un minuto de actividad. Esto permitirá un fácil acceso o consulta mediante la indicación del minuto exacto como clave de acceso, así como el día en cuestión.

### **Capacidad de determinar la posición del usuario usando el servicio de mapas de Google**

La aplicación cumple con el objetivo de ser capaz de determinar la posición absoluta del dispositivo móvil, gracias al servicio de mapas de Google, tal y como se puede observar en la figura 3.10. El punto azul representa dicha posición.

### **Capacidad de notificar por e-mail a un grupo de usuarios interesados en realizar tareas de monitorización**

La aplicación cumple con las expectativas de notificar al usuario en caso de valores fuera de rango, mediante el envío instantáneo de correos electrónicos, los cuales informarán en todo momento de la evolución de la situación hasta que se detecte una estabilización de los valores, lo que conllevará el fin del periodo de notificaciones a través de una notificación final informativa. En dichos correos también se adjuntará la localización del usuario. Esto se ve reflejado en las figuras 3.7 y 3.8.

## **Capacidad de adaptarse fácilmente a pantallas más pequeñas, tales como teléfonos móviles y tabletas**

La aplicación cumple con el objetivo de adaptarse también a dispositivos móviles, los cuales presentan pantallas más pequeñas, siendo capaz de mostrar información con la misma calidad que si nos encontrásemos en un ordenador personal, sin forzar al usuario a enfocar la pantalla mediante un aumento. Esto se aprecia perfectamente en la figura 3.8.



# Parte III

## Conclusiones y líneas futuras



# Conclusiones y líneas de trabajo futuras

## Conclusiones

En este proyecto se ha investigado, planificado y desarrollado un sistema, presentado como una combinación de dos aplicaciones Android y web funcionando como un todo, cuyo objetivo es la monitorización de parámetros del sistema cardiovascular, en particular el ritmo cardíaco, todo ello buscando una conexión en tiempo real que presente el menor retardo posible.

Durante el proyecto se han tenido en cuenta los conocimientos adquiridos durante el estudio de la titulación de Ingeniero de Telecomunicación, en particular aquellos relacionados con arquitectura de computadores y programación. Debido a que Comunicaciones fue la especialidad elegida en el último curso, se ha optado por usar e integrar uno de los protocolos de comunicaciones de última generación como es el protocolo Bluetooth 4.0, en concreto su modo de baja energía, una de las novedades que ha permitido la comercialización de millones de sensores de bajo coste y consumo.

Uno de los mayores retos del proyecto ha sido las implementaciones de ambas aplicaciones Android y web. Durante la titulación no se han cursado asignaturas relativas a programación móvil, Java o programación web. Habiendo desarrollado

algo de código Java por cuenta propia, el factor de mayor novedad y en el que más se ha aprendido durante la realización del proyecto es el componente web del mismo, gracias a la implementación del servidor, base de datos, y cliente web, es decir, todo el stack completo usando Javascript. Este valioso conjunto de conocimientos sienta las bases para, en un futuro, poder construir aplicaciones web de mayor complejidad técnica.

La aplicación generada como resultado de este Proyecto Fin de Carrera, responde de manera satisfactoria a los objetivos autoimpuestos al principio del mismo, tal y como se ha demostrado en el capítulo de pruebas. Por tanto, al haber obtenido un producto funcional, se considera el proyecto como exitoso.

## Líneas de trabajo futuras

Durante el desarrollo del proyecto, y especialmente durante las pruebas finales del mismo, han surgido ideas y mejoras para el mismo, que son posibles de implementar en un futuro.

### **Aplicación Android cliente de monitorización**

Las aplicaciones como Facebook o Twitter presentan tanto versiones web como nativas para el sistema operativo Android. Los usuarios de teléfonos móviles descargan decenas de aplicaciones nativas y odian tener que abrir el navegador para acceder a una cierta aplicación, lo que les fuerza recordar la URL de la misma, accesos más lentos en caso de sobrecarga del servidor, etcétera. Mediante una aplicación Android cliente, mostraríamos la misma interfaz gráfica que presenta nuestra aplicación web, con la única diferencia que tenemos la aplicación nativa y fácilmente accesible, haciendo al cliente más feliz.

### **Guardar dispositivos preferidos**

Consecuentemente ahorrando batería en nuestro teléfono Android. De esta

forma antes de volver a escanear, intentamos en primer lugar conectar directamente al dispositivo.

### **Extension de los servicios GATT ofrecidos al servidor web**

Dado que solo mandamos al servidor web datos de frecuencia cardíaca y localización, podríamos extender su funcionalidad mediante el envío de otras características GATT disponibles en el sensor, tales como la posición del sensor en el cuerpo, o mostrar el porcentaje de batería restante del sensor, para así prever con antelación un posible remplazo de la misma.

### **Reconocimiento de actividad**

Android incorpora una API, conectada a los servicios de localización de Google, que permite estimar la actividad física realizada por el usuario con una probabilidad bastante alta. De esta forma podríamos mandar dicha información al servidor web para así ser capaces de monitorizar también la actividad actual del usuario y ver si este se encuentra corriendo, pedaleando una bicicleta, o sentado, entre otras muchas posibilidades.

### **Extensión a relojes con sistema operativo Android**

Con la reciente aparición de Android wear, podríamos desarrollar la aplicación compatible con este tipo de dispositivos, con lo cual la aplicación sería mucho más portable, al no depender de nuestro teléfono móvil. El hecho de que la aplicación funciona la mayoría del tiempo en segundo plano justifica la viabilidad de dicha extensión.

### **Inclusión de gráficas estadísticas**

Mediante la consulta de valores de ritmo cardíaco almacenados en la base de datos, podríamos construir en nuestra aplicación web gráficas tales como históricos, evoluciones diarias, o en cualquier otro caso obtener valores mediante

el procesado de dichos datos, tales como la frecuencia cardíaca media, intervalos RR, máximos y mínimos, etcétera, los cuales pueden ser de gran utilidad a analistas especializados.

### **Sistema de autentificación**

Se podrían crear perfiles personalizados para cada usuario individual, configurando parámetros como rango de valores peligrosos, o correo electrónico a notificar. Para ello dotaríamos a nuestra página web de un sistema de registro y autentificación, haciendo uso de la base de datos. De esta forma solo el personal autorizado tendría acceso a los datos y estadísticos oportunos proporcionando privacidad a los clientes, mediante el uso de las credenciales adecuadas (usuario y contraseña).

### **Exportación de archivos**

Posibilidad de exportar gráficas y estadísticos mediante el pulsado de un botón, facilitando el compartir los datos con otros analistas que puedan presentar interés en la información.

Raúl Gómez Acuña

10 de septiembre de 2015

## Parte IV

### Apéndices



# Apéndice A

## Proceso de integración de una aplicación node.js en Heroku

El proceso para integrar nuestra aplicación node.js en cualquiera de los servidores dedicados ofrecidos por el servicio Heroku es bastante sencillo. Asumiendo que nos hemos registrado satisfactoriamente con nuestro correo electrónico, en primer lugar debemos instalar el cinturón de herramientas Heroku, el cual puede ser accedido a través del enlace

<https://devcenter.heroku.com/articles/getting-started-with-nodejs>.

Este nos brinda acceso a la interfaz de comandos Heroku mediante el terminal y nos proporciona un comando local ejecutable llamado `heroku`, el cual nos ayudará a la gestión de las aplicaciones de forma local. La figura A.1 muestra el proceso de autentificación mediante el uso del comando `heroku login`

```
$ heroku login  
Enter your Heroku credentials.  
Email: zeke@example.com  
Password:  
Authentication successful.
```

Figura A.1: Autentificación en el terminal en Heroku [Her07]

Debemos tener en cuenta que si usamos un cortafuegos o firewall que requiere el uso de un proxy para conectarse con los servicios externos HTTP/HTTPS, tendremos que configurar las variables de entorno `HTTP_PROXY` or `HTTPS_PROXY` en nuestro entorno de desarrollo local previo a correr el comando.

El siguiente paso consiste en preparar la aplicación que queremos integrar en el servidor. Para ello nos situamos, localmente y desde el terminal, en el directorio el cual contiene nuestro archivo `package.json` con todas las dependencias. Debemos tener el código fuente enlazado y asociado a un repositorio git, como el proporcionado por el servicio Github por ejemplo. Ahora debemos ejecutar el comando `heroku create` para crear una aplicación en Heroku. Esto preparará a Heroku para que reciba nuestro código fuente.

Cuando se crea la aplicación en Heroku, también se crea un repositorio remoto git llamado `heroku`, el cual estará asociado con nuestro repositorio local.

Heroku genera un nombre aleatorio para nuestra aplicación. Para asignar un nombre específico, lo pasamos como parámetro al comando. Esto se ve reflejado en la figura A.2

```
$ heroku create
Creating sharp-rain-871... done, stack is cedar-14
http://sharp-rain-871.herokuapp.com/ | https://git.heroku.com/sharp-rain-871.git
Git remote heroku added
```

Figura A.2: Preparando a Heroku para crear la aplicación [Her07]

Ahora debemos enviar nuestro código fuente mediante el comando `git push heroku master`. El resultado se puede apreciar en la figura A.3.

La aplicación ya se encuentra desplegada en el servidor y podemos asegurarnos de que al menos una instancia de la aplicación está funcionando mediante el comando `heroku ps:scale web=1`.

Ahora ya podemos visitar la aplicación a través de la URL generada, en base al nombre pasado como parámetro (o el obtenido aleatoriamente). Independientemente, existe un comando útil para abrirla desde el terminal, `heroku open`.

Una vez que la aplicación se encuentra funcionando podremos consultar siempre que queramos los logs generados por Heroku y los nuestros personalizados en el código fuente a través del comando mostrado en la figura A.4

```
$ heroku logs --tail
2011-03-10T10:22:30-08:00 heroku[web.1]: State changed from created to starting
2011-03-10T10:22:32-08:00 heroku[web.1]: Running process with command: `node index.js`
2011-03-10T10:22:33-08:00 heroku[web.1]: Listening on 18320
2011-03-10T10:22:34-08:00 heroku[web.1]: State changed from starting to up
```

Figura A.4: Consulta de logs de nuestra aplicación [Her07]

Por último, mencionar que Heroku proporciona un Tablero de Instrumentos o Dashboard en su página web, desde el cual podremos monitorizar el estado de nuestra aplicación, controlar los permisos de acceso, configurar distintas opciones, consultar

```
$ git push heroku master
Counting objects: 343, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (224/224), done.
Writing objects: 100% (250/250), 238.01 KiB, done.
Total 250 (delta 63), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:           NPM_CONFIG_LOGLEVEL=error
remote:           NPM_CONFIG_PRODUCTION=true
remote:           NODE_MODULES_CACHE=true
remote:
remote: -----> Installing binaries
remote:           engines.node (package.json): 0.12.2
remote:           engines.npm (package.json): unspecified (use default)
remote:
remote:           Downloading and installing node 0.12.2...
remote:           Using default npm version: 2.7.4
remote:
remote: ....
remote: -----> Build succeeded!
remote:           |--- ej@2.3.1
remote:           |--- express@4.9.8
remote:
remote: -----> Discovering process types
remote:           Procfile declares types -> web
remote:
remote: -----> Compressing... done, 9.4MB
remote: -----> Launching... done, v8
remote:           http://sharp-rain-871.herokuapp.com deployed to Heroku
To https://git.heroku.com/nameless-savannah-4829.git
 * [new branch]      master -> master
```

Figura A.3: Enviando el código fuente a Heroku [Her07]

métricas específicas, etcétera. Algunas de las opciones son gratuitas y otras de pago.

La interfaz presentada es la mostrada en la figura A.5

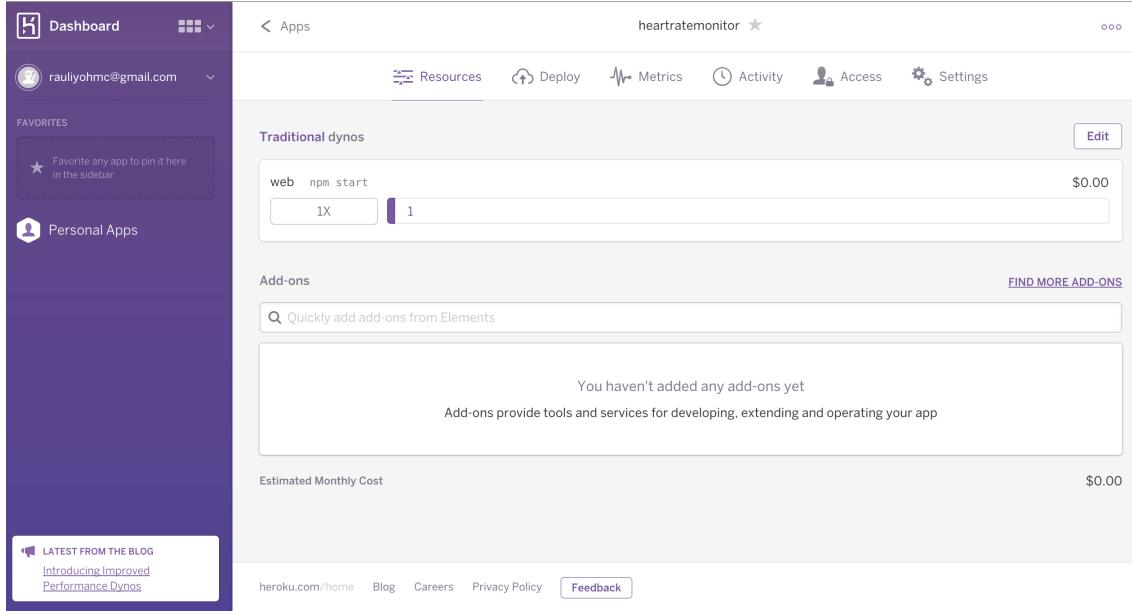


Figura A.5: Apariencia del dashboard en Heroku [Her07]



# Bibliografía

- [And07] Android. “Android open source Bluetooth”, Junio 2015. Documento en formato HTML accesible por internet en la dirección: <https://source.android.com/devices/bluetooth.html>, 2007.
- [And08] Android. “Guía para desarrolladores en Android”, Agosto 2015. Documento en formato HTML accesible por internet en la dirección: <http://developer.android.com/>, 2008.
- [Arg14] Argenox. “Introducción al protocolo Bluetooth de bajo consumo”, Mayo 2015. Documento en formato HTML accesible por internet en la dirección: <http://www.argenox.com/es/bluetooth-baja-energia-ble-desarrollo/biblioteca/introduction-bluetooth-bajo-consumo/>, 2014.
- [Dah09] Ryan Dahl. “Página oficial de NodeJS en inglés”, Enero 2015. Documento en formato HTML accesible por internet en la dirección: <https://nodejs.org/>, 2009.
- [Day14] B. Dayley. *Node.js, MongoDB, and AngularJS Web Development*. Addison-Wesley Professional, 1st edition, 2014.

- [dV12] Universidad de Valencia. “Teoría de codificación”, Julio 2015. Documento en formato pdf accesible por internet en la dirección: <http://www.uv.es/~hertz/hertz/Docencia/teoria/codificacion.pdf>, 2012.
- [eMa14] eMarketer. Smartphone users worldwide will total 1.75 billion in 2014. *eMarketer*, 2014.
- [Esp01] Real Academia Española. “Diccionario de la lengua española”, Vigésima segunda edición (versión electrónica consultable). [www.rae.es](http://www.rae.es), 2001.
- [Exc08] Stack Exchange. “Página para preguntas y respuestas de programadores”, Agosto 2015. Documento en formato HTML accesible por internet en la dirección: <http://stackoverflow.com/>, 2008.
- [Exp11] Express. “Página oficial de ExpressJS en inglés”, Julio 2015. Documento en formato HTML accesible por internet en la dirección: <http://expressjs.com/>, 2011.
- [Gir13] J. T. Gironés. *El gran libro de Android*. marcombo, 3<sup>a</sup> edition, 2013.
- [Git08] Github. “Página oficial de Github”, Agosto 2015. Documento en formato HTML accesible por internet en la dirección: <https://github.com/>, 2008.
- [Hav14] A. Q. Haviv. *MEAN Web Development*. Packt Publishing, 1st edition, 2014.
- [Her07] Heroku. “Página oficial de Heroku”, Septiembre 2015. Documento en formato HTML accesible por internet en la dirección: <https://www.heroku.com/>, 2007.
- [Mon09] MongoDB. “Página oficial de MongoDB en inglés”, Mayo 2015. Documento en formato HTML accesible por internet en la dirección: <https://www.mongodb.org/>, 2009.

- [Pen14] Penflip. “Introducción a MEAN”, Junio 2015. Documento en formato HTML accesible por internet en la dirección: <https://www.penflip.com/ocruz/introduccion-a-mean/blob/master/mean.txt>, 2014.
- [RL14] Kathy Nagamine Ramon Llamas, Ryan Reith. Smartphone market share q3 2014. Technical report, IDC Corporate USA, 2014.
- [SIG00] Bluetooth SIG. “Página Oficial del SIG de Bluetooth”, Septiembre 2015. Documento en formato HTML accesible por internet en la dirección: <https://www.bluetooth.org/>, 2000.
- [Sim14a] K. Simpson. *You Don't Know JS: Scope & Closures*. O'Reilly Media, Inc., 1st edition, 2014.
- [Sim14b] K. Simpson. *You Don't Know JS: this & Object Prototypes*. O'Reilly Media, Inc., 1st edition, 2014.
- [Sim15a] K. Simpson. *You Don't Know JS: Types & Grammar*. O'Reilly Media, Inc., 1st edition, 2015.
- [Sim15b] K. Simpson. *You Don't Know JS: Up & Going*. O'Reilly Media, Inc., 1st edition, 2015.
- [Wik04] Wikipedia. “Página de Bluetooth en Wikipedia”, Septiembre 2015. Documento en formato HTML accesible por internet en la dirección: <https://es.wikipedia.org/wiki/Bluetooth>, 2004.

