

DemoCritics

**Jaime Ramos Romero
Javier Bastarrica Lacalle**

**Grado en Ingeniería Informática
Facultad de Informática**

UNIVERSIDAD COMPLUTENSE DE MADRID



**Trabajo de Fin de Grado
Madrid, Junio 2015**

Directores:
Samer Hassan Collado
Pablo Ojanguren



Autorización de Difusión y Uso

Autorizo a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, tanto la propia memoria, como el código, la documentación y/o el software desarrollado.

Jaime Ramos Romero
Javier Bastarrica Lacalle

Madrid, Junio 2015

Copyleft by Jaime Ramos Romero and Javier Bastarrica Lacalle, released under the license Creative Commons Attribution Share-Alike International 4.0 available at:

<https://creativecommons.org/licenses/by-sa/4.0/>

Índice general

	Page
Autorización de Difusión y Uso	III
Abstract	IX
Resumen	XI
1. Introducción	1
1.1. Objetivos del Proyecto	1
1.2. Estructura del Documento	1
2. Construyendo la idea de la aplicación: DemoCritics	3
2.1. Introducción	3
Brainstorming de ideas	3
2.2. Marco teórico de la idea	5
2.2.1. Política en el mundo de la Informática	5
2.2.2. Democracia	6
Democracia representativa	6
Democracia participativa	6
Democracia directa	6
Democracia deliberativa	6
2.3. Adentrándonos en la idea	6
3. Estado del Arte	9
3.1. Servidor: Wave	9
3.2. Aplicación Android: DemoCritics	9

3.2.1.	Programas Políticos	9
UPyD Parla	9	
#RecuperaCórdoba	11	
PSOE Andalucía	11	
PP Canarias	12	
3.2.2.	Participación Ciudadana	13
Reddit	13	
Change.org	15	
Programas Colaborativos de Ahora Madrid y Zaragoza en Común	16	
Appgree	18	
4.	Tecnologías del Proyecto	21
4.1.	Tecnologías de Wave	23
4.1.1.	Google Wave	23
4.1.2.	Apache Wave	23
4.1.3.	Características de Wave	23
Federación	24	
Consistencia en tiempo real	24	
Escalabilidad	24	
4.1.4.	Servidores Wave	24
Wave in a Box	24	
SwellRT	25	
4.1.5.	Eclipse	26
4.1.6.	Java	26
4.1.7.	GWT	26
4.1.8.	JavaScript	26
4.1.9.	HTTP	26
4.1.10.	WebSocket	26
4.1.11.	Git	26

4.1.12. GitHub	26
4.2. Tecnologías de la Aplicación Android	26
4.2.1. Android	27
4.2.2. XML	27
4.2.3. Android Studio	27
4.2.4. JSON	27
4.2.5. SQL	27
4.2.6. MySQL	27
4.2.7. PhpMyAdmin	27
4.2.8. PHP	27
4.2.9. Laravel 5	27
4.2.10. OpenShift	27
4.2.11. Sublime Text	27
4.2.12. POP: Prototyping On Paper	27
5. Metodología del Proyecto	29
5.1. Uso de Software Libre	29
5.2. Metodología de Migración de Wave a Android	29
5.2.1. Objetivo	29
5.2.2. Plataforma: Entorno de Desarrollo, Construcción y Depuración	29
Eclipse	30
Proceso de Construcción por Consola	32
Proceso de Depuración	35
5.2.3. Migración: Identificación y Solución de Problemas	35
Conexión HTTP	36
Conexión WebSocket	40
Conexión final y Logging	43
Organización del código: Servicio Android	44
5.2.4. Dependencias	48

5.2.5.	Resultado de la Migración	48
5.3.	Diseño de la Aplicación: Diseño Guiado Por Objetivos	50
5.3.1.	Investigación	50
	Intención Inicial: Prototipo básico	50
	Entrevista con Labodemo	52
	Conclusión	53
	Entrevista con Javier de la Cueva	54
	Conclusión	55
5.3.2.	Modelado de Personas	55
5.3.3.	Definición de Escenarios y Requisitos	56
5.3.4.	Framework de diseño	58
	Metodología	58
	Prototipos en papel	58
	Prototipos intermedios	58
	Prototipo final	60
5.3.5.	Análisis de Usabilidad	60
5.4.	Implementación de DemoCritics	61
5.5.	Evaluación con Usuarios	61
6.	Arquitectura del Proyecto	63
6.1.	Arquitectura de Wave	63
6.1.1.	Modelo Wave	63
6.1.2.	Modelo SwellRT	64
6.2.	Arquitectura de la aplicación	64
6.2.1.	Base de datos	65
6.2.2.	Service REST	66
	Arquitectura	69
	Rutas	71
	Controladores	73
6.2.3.	Servicio Android de conexión con Wave	75

6.2.4. Cliente Android	75
7. Resultados y Conclusiones	77
7.1. Discusion de Resultados	77
7.1.1. Evaluación con usuarios	77
7.2. Conclusiones	77
8. Trabajo a Futuro	79
8.1. Mejoras	79
Bibliografía	79

Índice de figuras

2.1. Brainstorming sobre la idea a desarrollar	4
3.1. Capturas de UPyD Parla	10
3.2. Capturas de #IURecuperaCórdoba	11
3.3. Capturas de PSOE Andalucía	12
3.4. Capturas de PP Canarias	13
3.5. Plaza Podemos utilizando la plataforma Reddit	14
3.6. Change.org · La mayor plataforma de peticiones del mundo . .	15
3.7. Creación colaborativa del programa de Ahora Madrid.	17
3.8. Creación colaborativa del programa de Zaragoza en Común. .	17
3.9. Capturas de Appgree	18
4.1. Cliente Wave In A Box	25
5.1. Distribución Actual de Versiones Android (Fuente: Google) . .	31
5.2. Emulador Android API 19	34
5.3. Ejemplo de Traza de Error en Logcat	37
5.4. Pantalla de Login de WaveAndroid	44
5.5. Proceso de conexión Http con Servicio	46
5.6. Proceso de conexión WebSocket con Servicio	47
5.7. Esquema de Clases de SwellRT-Android con Servicio	49
5.8. Vista principal de secciones	59
5.9. Visualizando una sección	60
6.1. Modelo Conversacional de Wave	64
6.2. Arquitectura de la aplicación.	65

6.3.	Modelo entidad-relación de la base de datos.	66
6.4.	Pantalla principal de Laravel 5.	67
6.5.	Arquitectura de Laravel	70
6.6.	Camino desde la ruta al método del controlador.	74

Índice de cuadros

4.1.	Tecnologías usadas en el Proyecto	22
5.1.	Dependencias de SwellRT-Android	48
6.1.	Funciones CRUD	68
6.2.	Códigos de estado de la respuesta del servidor	69

Abstract

Abstract en inglés.

Keywords:

Apache Wave, Collaboration, Android, Apps, Real Time, Politics

Resumen

Abstract en español.

Palabras Clave:

Apache Wave, Colaboración, Android, Apps, Tiempo Real, Política

Capítulo 1

Introducción

1.1. Objetivos del Proyecto

Los objetivos son los siguientes:

- 1^a parte: Migración de Wave a Android
 - Estudiar la implementacion actual de Wave en Java y GWT.
 - Adaptar la implementación para hacer uso de las caracteristicas nativas de Android.
- 2^a parte: Creación de una aplicación Android
 - Evaluar posibles ideas de aplicación y estudiar su viabilidad.
 - Evaluar con usuarios su usabilidad.
 - Implementar la aplicación.
 - Testear y evaluar con usuarios el resultado.

1.2. Estructura del Documento

- Capítulo 2 - Migración de Wave a Android: Descripcion de la tecnologia Wave y de la Metodología utilizada para la migración a Android.
- Capítulo 3 - AppTFG:
- Capítulo 4 - Resultados y Conclusiones:
- Capítulo 5 - Trabajo Futuro:

Capítulo 2

Construyendo la idea de la aplicación: DemoCritics

2.1. Introducción

Una vez que habíamos migrado la tecnología de Wave que soportaría el núcleo de nuestra aplicación, era hora de decidir qué aplicación le íbamos a dar de cara a desarrollar una app Android que hiciera uso de ello. Era importante tener en cuenta las características que nos ofrecía Wave:

- Edición colaborativa.
- Consistencia en Tiempo Real.

Brainstorming de ideas

EXPLICAR UN POQUILLO BRAINSTORMING

Después de darle nosotros mismo unas cuantas vueltas a posibles ideas de implementaciones que podrían hacer uso de estas características, decidimos realizar una sesión de brainstorming junto a nuestros profesores para identificar ideas potenciales. En esta sesión aparecieron temas tan variados como wikis colaborativas, aplicaciones de inteligencia artificial, aportaciones colaborativas en política, edición de vídeos y música, cursos de formación colaborativos, visualización de mapas, etc.

CAPÍTULO 2. CONSTRUYENDO LA IDEA DE LA APLICACIÓN: DEMOCRITICS

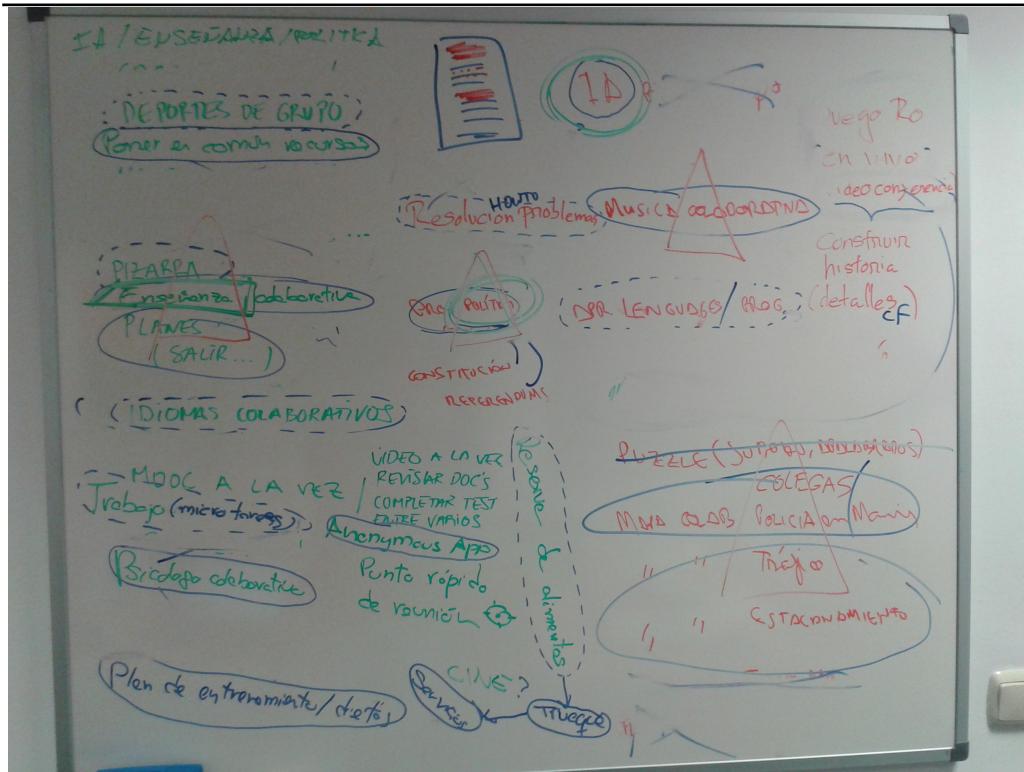


Figura 2.1: Brainstorming sobre la idea a desarrollar

Con un gran repertorio de ideas expuestas en la sesión, decidimos centrarnos primero en descartar aquellas que a nosotros no nos motivaba llevar a cabo. De esta manera nos quedamos con cuatro ideas fundamentales a desarrollar en nuestra aplicación: Política, Música, Inteligencia Artificial y Mapas. Centrándonos ahora solo en estos temas, surgieron varias ideas colaborativas como: desarrollar documentos políticos, programas electorales, comunicación entre colectivos en tiempo real, aprendizaje de música, edición de partituras y obras, aplicaciones colaborativas con inteligencia artificial, edición de mapas en tiempo real, lexicalización, etc.

Finalmente, ya que ambos teníamos interés por la política, decidimos realizar una aplicación colaborativa relacionada con dicho mundo. En esta aplicación podríamos recurrir a la edición de contenidos en tiempo real, ya fueran propuestas políticas, programas electorales u otro tipo de documentos. Más adelante también podríamos hacer uso incluso de alguna herramienta de Inteligencia Artificial para automatizar algunas tareas o realizar recomendaciones sociales.

Lo que sí que tuvimos claro desde el principio es la idoneidad del momento

CAPÍTULO 2. CONSTRUYENDO LA IDEA DE LA APLICACIÓN: DEMOCRITICS

actual para desarrollar una app de temática política, dado que nos encontramos en año electoral. Nos propusimos el objetivo de desarrollar algo que pudiera tener cierta repercusión y utilidad en las próximas citas electorales de este año 2015. Intentaríamos pensar en la aplicación no solo como un Trabajo de Fin de Grado sino como algo que pudiéramos llevar más allá y que resultara útil a la sociedad.

2.2. Marco teórico de la idea

En los siguientes puntos se discutirá acerca de una serie de consideraciones sobre el estado actual de la política y la democracia. Nos centraremos sobre todo en su relación con las nuevas tecnologías como herramientas capaces de cambiar la manera en la que se conciben ambas disciplinas.

2.2.1. Política en el mundo de la Informática

A primera vista podemos pensar que la informática no parece entusiasmar a los informáticos. Podemos ubicar la política como una parte de las ciencias sociales, situando la informática en ciencias formales. Pero si pensamos en factores como la gestión de los privilegios de una aplicación entre los que definiremos de alguna forma una jerarquía, estaremos en cierta manera haciendo política. También encontraremos características políticas en el diseño relacional de una base de datos. Definiendo los campos de una base de datos podemos encontrarnos con algunos valores como el sexo, la nacionalidad, la edad o incluso las relaciones o restricciones que existen entre las tablas. Estaremos definiendo unas reglas básicas de funcionamiento de la base de datos establecidas por unos principios políticos.

Además si nos sumergimos en el mundo de las Licencias de Uso en el desarrollo de software encontraremos más política aún. Licencias que determinan el uso de un tipo de software, ya sea para compartir, vender o distribuir copias. Multitud de reglas políticas” definidas en un documento de licencia de uso. Así como las restricciones que establecemos en la metodología orientada a objetos, estableciendo las relaciones de herencia, restricción de métodos, variables, etcétera.

Regresando a la actualidad y basándonos en no muy lejanos acontecimientos pasados, habremos oído cómo algunos gobiernos recopilan datos de la actividad de los usuarios en las redes sociales, analizando todo el contenido que generan. Incluso vemos cómo algunas aplicaciones móviles piden aprobar

CAPÍTULO 2. CONSTRUYENDO LA IDEA DE LA APLICACIÓN: DEMOCRITICS

permisos con los que operar libremente en tu dispositivo.

Observamos por tanto que la política está más integrada en la informática de lo que parece, sobre todo si dejamos a un lado la informática más científica y formal y pasamos a la informática social, la de los gobiernos, la de los negocios o la de las relaciones sociales.

2.2.2. Democracia

Democracia representativa

Democracia participativa

Democracia directa

Democracia deliberativa

2.3. Adentrándonos en la idea

La idea a desarrollar está generada en una época en la que la política parece haber despertado el interés de una parte considerable de la ciudadanía. Podría ser por tanto una herramienta útil para participar en temas políticos de forma sencilla y atractiva. Dejando así atrás los tópicos a menudo escuchados de *“yo no entiendo de política”*, *“la política es aburrida”*, *“no sé a quién votar”* o *“no he leído nunca un programa electoral”* entre otros.

La herramienta ofrecería una nueva forma de participar en la política y de llevar a los ciudadanos los programas electorales expuestos por las diferentes formaciones políticas. De forma que, para potenciar el uso social de la aplicación, los ciudadanos podrían leer aquellos puntos de los programas más vistos, debatidos, comentados, etc. Así, cualquier usuario tendría a su disposición todos los programas electorales en su bolsillo, por lo que no tendría que ir a la página web de cada formación política y descargarse un documento de 200 páginas. Pensamos que esta forma tradicional de presentar un programa político en un solo documento en un mundo donde las posibilidades de comunicarnos se han desarrollado exponencialmente mediante las nuevas tecnologías no es la mejor manera de generar interés por su lectura y la implicación en política de las personas.

Por otra parte, y teniendo en cuenta la tendencia actual de los nuevos movimientos ciudadanos de elaborar programas políticos en base a propuestas

CAPÍTULO 2. CONSTRUYENDO LA IDEA DE LA APLICACIÓN: DEMOCRITICS

de los ciudadanos, la aplicación también debía ofrecer alguna manera de realizar Propuestas y debatirlas entre todos. De esta forma tanto la ciudadanía como las formaciones políticas podrían saber en cualquier momento cuáles son las principales preocupaciones de los ciudadanos y qué medidas o soluciones proponen para resolverlas. Además pensamos que podríamos aprovechar las características de Wave para realizar estas Propuestas de forma colaborativa y en tiempo real, aportando un valor diferenciador respecto a las actuales soluciones desarrolladas para web (Ver sección 3.2.2).

Por tanto, desde un primer punto de vista subjetivo, la aplicación quedó dividida en dos partes. Por un lado tendríamos la presentación estructurada de los programas políticos que presentan las formaciones políticas. Y por otro todas las propuestas que elaboran de forma colaborativa los ciudadanos, ya sea individualmente o en colectivos sociales.

Capítulo 3

Estado del Arte

3.1. Servidor: Wave

3.2. Aplicación Android: DemoCritics

En esta sección exploraremos algunas de las principales aplicaciones informáticas que existen en la actualidad destinadas a la participación ciudadana en propuestas, lectura de programas electorales o divulgación de candidaturas.

3.2.1. Programas Políticos

En la actualidad no existe ningún tipo de aplicación móvil orientada a debatir los programas electorales de los partidos políticos en su conjunto. Concretamente no existe ningún tipo de plataforma que agrupe en un solo sitio los programas electorales de las diferentes candidaturas. Lo más parecido que hemos podido encontrar han sido aplicaciones elaboradas por un partido político, orientadas a dar a conocer su candidatura. En ellas podemos ver normalmente, entre otros, a presentación de candidatura, vídeos propagandísticos y el programa electoral.

Pasamos ahora a analizar algunas de las aplicaciones móviles encontradas, identificando en cada caso aspectos e ideas que nos han resultado positivos y negativos.

UPyD Parla

La aplicación presenta al candidato de UpyD Carlos Alt Bustelo para la alcaldía de Parla. Se trata de una aplicación divulgativa donde podemos conocer todo lo esencial de la candidatura de UpyD para las elecciones del municipio

CAPÍTULO 3. ESTADO DEL ARTE

de Parla en Mayo de 2015: los candidatos, el programa, vídeos, etc.



Figura 3.1: Capturas de UPyD Parla

- Aspectos positivos:

- Interfaz limpia y de colores planos (predominando el rosa que identifica al partido) con menú lateral que permite navegar por la aplicación estés donde estés.
- Presentación de Programa Electoral estructurado con Indice inicial.
- El Programa se lee dentro de la app, no nos lleva a leer el programa en PDF de la web.
- Presentación de una Sección del Programa Electoral de forma resumida, teniendo la opción de leer la sección entera al pulsar un botón.

- Aspectos negativos:

- Posee una sección llamada "Memes" cuyo nombre no se entiende ya que se limita a mostrar carteles propagandísticos de la candidatura.

CAPÍTULO 3. ESTADO DEL ARTE

#RecuperaCórdoba

Esta app presenta la candidatura de Pedro García de Izquierda Unida a la provincia de Córdoba, informando de su propuesta de gobierno de forma resumida. En la aplicación podremos encontrar la lista de los candidatos propuestos a la comunidad cordobesa, el programa electoral de la formación, las propuestas del partido, noticias de última hora y vídeos.

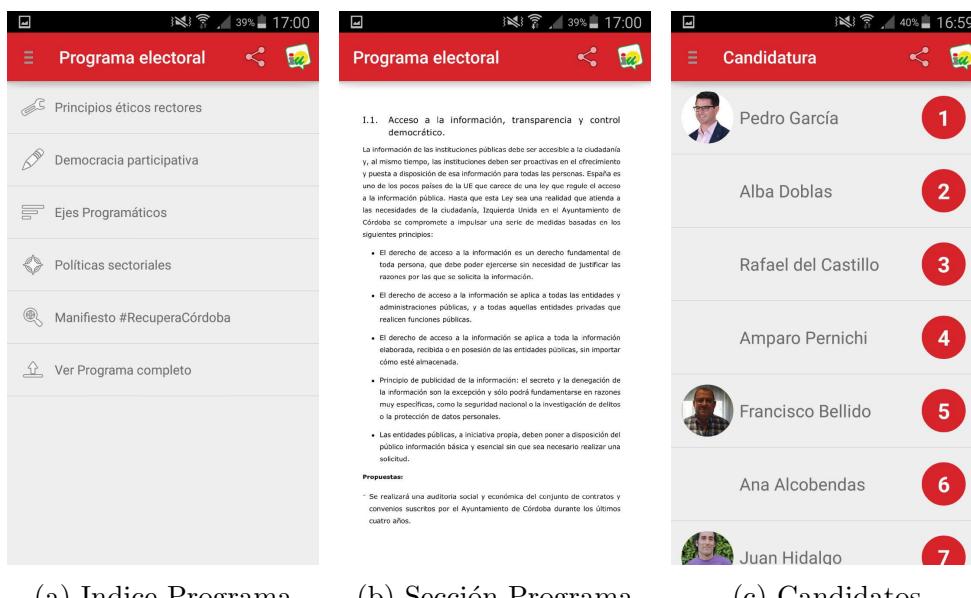


Figura 3.2: Capturas de #IURecuperaCórdoba

- Aspectos positivos:

- Aspectos negativos:

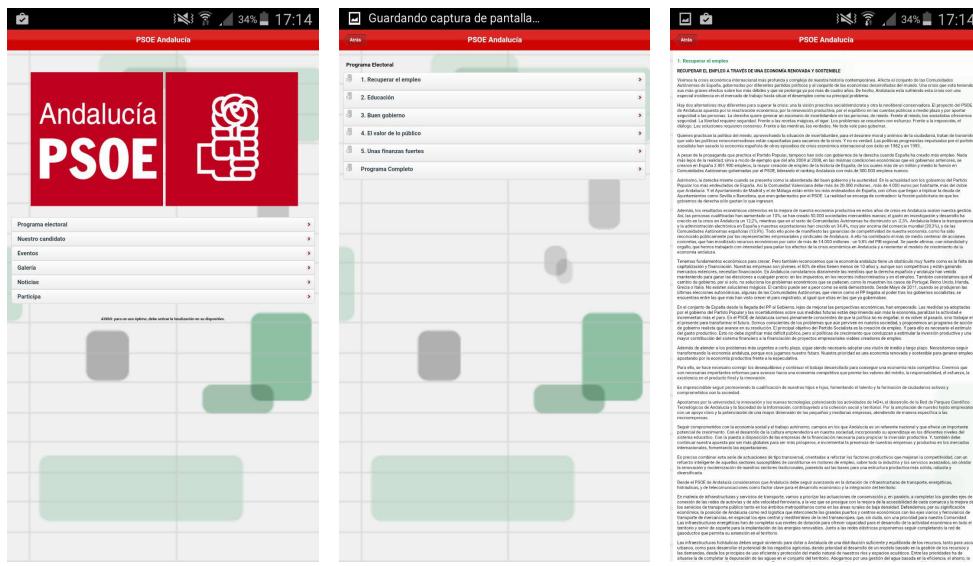
PSOE Andalucía

Esta app presenta la candidatura del PSOE a la junta de Andalucía para las elecciones del 22 de Marzo, promocionando básicamente su programa

CAPÍTULO 3. ESTADO DEL ARTE

electoral y a la candidata Susana Díaz. Permite también estar al día de noticias y eventos relacionados con dicha candidatura.

La navegación por el programa, aunque estructurada en un primer nivel, se realiza directamente visualizando páginas que parecen extraídas del programa en PDF.



(a) Pantalla Principal (b) Índice Programa (c) Sección Programa

Figura 3.3: Capturas de PSOE Andalucía

- Aspectos positivos:

■

- Aspectos negativos:

■

PP Canarias

La delegación del Partido Popular en Canarias presenta su aplicación móvil para promocionar a sus candidatos para las elecciones autonómicas y municipales de Mayo de 2015. La aplicación nos avisará de los eventos electorales, podremos consultar los candidatos, novedades, galería de imágenes y por supuesto ver el programa electoral.

CAPÍTULO 3. ESTADO DEL ARTE

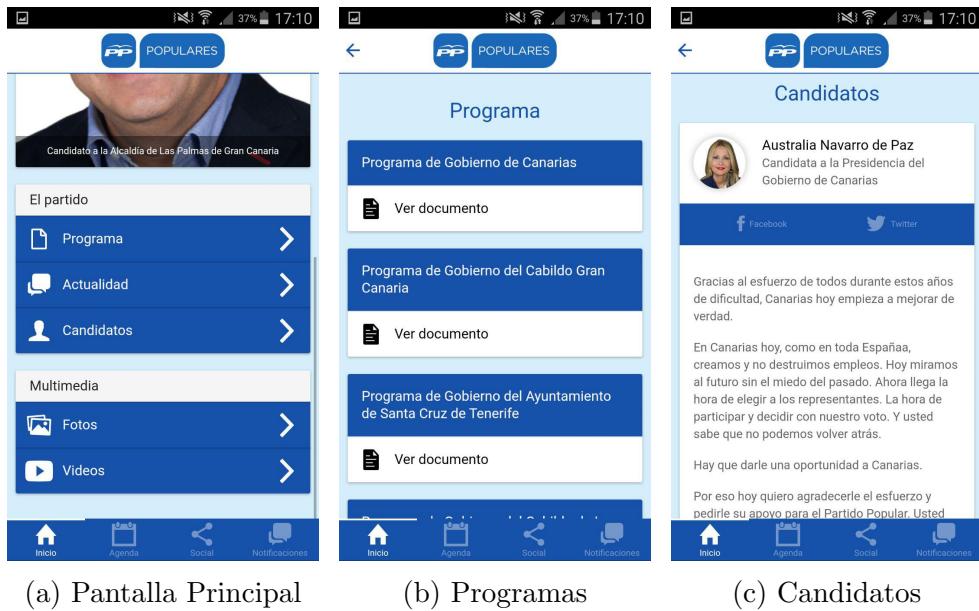


Figura 3.4: Capturas de PP Canarias

- Aspectos positivos:

■

- Aspectos negativos:

■

3.2.2. Participación Ciudadana

Centrándonos en la participación ciudadana ya sea mediante la generación de Propuestas, el desarrollo colaborativo de programas o la recogida de firmas, existen numerosos portales en internet y aplicaciones móviles destinadas a ello. Realizaremos un breve repaso a las aplicaciones más destacadas.

Reddit

Reddit [1] es una plataforma web donde los usuarios pueden crear temas, propuestas o compartir enlaces web a otros sitios. A primera vista puede parecer un foro, aunque la principal diferencia respecto a éste último radica

CAPÍTULO 3. ESTADO DEL ARTE

en que otros usuarios pueden votar a favor o en contra de los enlaces, haciendo que el sistema los haga aparecer como más o menos destacados. De esta forma los temas de conversación, enlaces, o propuestas aparecerán en el orden que haya escogido la comunidad según la puntuación positiva o negativa que le hayan dado. En principio el uso de reddit está destinado a todo tipo de temas, entre los que podemos encontrar algunos ejemplos fuertemente relacionados con la participación ciudadana. Es el caso de Plaza Podemos [2]: un espacio utilizado para que la ciudadanía pueda expresar sus propuestas, compartir noticias relacionadas con la actualidad política o debatir aquellos temas que más les preocupan. Así, aunque no seamos participantes de reddit, de un simple vistazo podemos saber qué es lo más debatido por la ciudadanía, las propuestas que quieren llevar a cabo en el gobierno o cuáles son los temas que más les preocupan.



Figura 3.5: Plaza Podemos utilizando la plataforma Reddit

- Aspectos positivos:

■

- Aspectos negativos:

CAPÍTULO 3. ESTADO DEL ARTE

Change.org

Change.org [3] es un portal web que permite lanzar múltiples peticiones de cambio en Internet. Podríamos definirlo como la evolución de la recogida de firmas en la calle: cualquiera puede realizar una petición para solicitar el apoyo de otros. Las personas que decidan apoyar la petición, dejarán sus datos personales y constarán entre el número de personas que han firmado a favor de la petición. Una vez que han alcanzado un número objetivo de apoyos se procede a entregar las firmas digitales al organismo, persona o entidad a la que va destinada la petición.

Por ejemplo: en mayo de 2011, en relación con las movilizaciones del Movimiento 15-M y Democracia Real Ya, y ante el desalojo por los Mossos de Esquadra se llevó a cabo la petición “Exige la dimisión fulminante del Coneller de Interior Felip Puig por la violencia utilizada en Pza. Catalunya”.

Desde su creación en 2007, Change.org ha logrado muchas de sus peticiones demandadas entre los que se incluyen la atención de pacientes con enfermedades complejas, protección sobre animales y medio ambiente, derechos públicos, leyes, etc.



Figura 3.6: Change.org · La mayor plataforma de peticiones del mundo

CAPÍTULO 3. ESTADO DEL ARTE

- Aspectos positivos:

■

- Aspectos negativos:

■

Programas Colaborativos de Ahora Madrid y Zaragoza en Común

Para las pasadas elecciones municipales del 24 de Mayo, la candidatura de unidad popular Ahora Madrid, desarrolló una plataforma en la web para elaborar su programa electoral de forma colaborativa. En esta plataforma, cualquier usuario tenía la oportunidad de explorar las propuestas por categoría o por distrito. De tal forma que podría debatirlas, puntuarlas o crear sus propias propuestas. Así las propuestas más valoradas por la comunidad, serían llevadas al programa final para las elecciones municipales del 24 de Mayo.

El resultado final fue determinar las cinco propuestas más votadas que fueron incluidas en el programa final como medidas urgentes para realizar en los 100 primeros días de gobierno.

CAPÍTULO 3. ESTADO DEL ARTE



Figura 3.7: Creación colaborativa del programa de Ahora Madrid.

También utilizaron una plataforma similar en la candidatura zaragozana de unidad popular Zaragoza en Común [4].

The screenshot shows the header with the 'GANEMOS Zaragoza' logo and navigation links for 'INICIO' and 'ENTRAR'. The main title is 'Programa colaborativo de Zaragoza en Común' in red.

A message in red text says: 'Tienes que estar identificado para votar o comentar las propuestas.'

Un programa elaborado en Común

Text: 'Desde el pasado septiembre Zaragoza en Común puso en marcha un proceso para elaborar de forma colectiva el programa electoral con las necesidades y demandas de los habitantes de la ciudad.'

Text: 'Hemos recogido propuestas a través de la web, en los foros que se están organizando en distintos barrios, en foros sectoriales y a través de la consulta a expertos y movimientos sociales.'

Text: 'Gracias al trabajo de cientos de personas en los grupos sectoriales se han canalizado todas estas propuestas y se han articulado en torno a ejes, objetivos generales, objetivos específicos y medidas concretas.'

Text: 'Ahora nos encontramos ante la necesidad de priorizar los objetivos que hemos recogido y elaborado desde las propuestas. En esta web se presenta un primer avance de esas propuestas para que, a través de una consulta ciudadana, continuemos elaborando entre todos un programa para Zaragoza en Común.'

Text: 'primera fase: Un programa de consensos ciudadanos'

Text: 'El 12 de marzo se presentó en Asamblea Ciudadana este primer borrador, y se continua a través de una asamblea ciudadana virtual en la cual hay un periodo de 4 días para debate y votación por parte de la ciudadanía.'

Ejes temáticos

- Modelo de ciudad** (Image: Aerial view of Zaragoza city)
- Derechos sociales** (Image: A group of people in a meeting)
- Economía, trabajo y desigualdad** (Image: People working at a table)

Figura 3.8: Creación colaborativa del programa de Zaragoza en Común.

CAPÍTULO 3. ESTADO DEL ARTE

- Aspectos positivos:

-

- Aspectos negativos:

-

Appgree

Appgree[5] es una plataforma desarrollada con el objetivo de poner a grandes grupos de personas de acuerdo en poco tiempo. Está disponible tanto para web como para móviles y permite que sus usuarios lancen propuestas y debatan sobre cualquier tema pudiendo votar y alcanzar un consenso, obteniendo los resultados de dicha votación casi en tiempo real gracias a un algoritmo estadístico desarrollado por ellos llamado DemoRank[6].

En la aplicación podemos acceder a una lista de canales: que aglutinan encuestas, propuestas y preguntas (ya sea de respuesta abierta o de votación entre dos o mas opciones), pudiendo votar y ver los resultados actuales de votación y respuestas. Para fomentar la participación potencian mucho el uso tops de preguntas más candentes y recientes. Además se pueden compartir preguntas por redes sociales para aumentar su difusión.

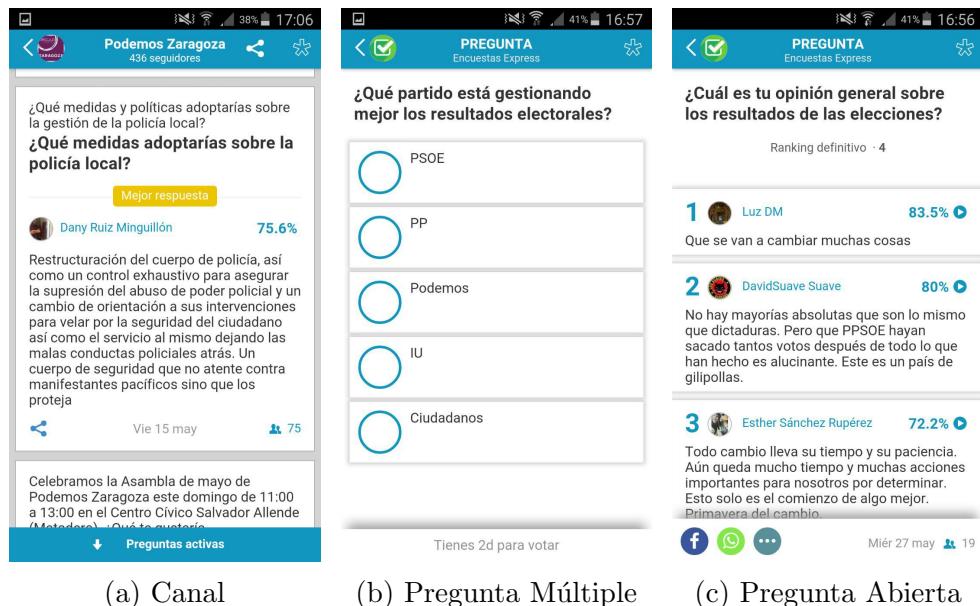


Figura 3.9: Capturas de Appgree

CAPÍTULO 3. ESTADO DEL ARTE

- Aspectos positivos:

■

- Aspectos negativos:

■

Capítulo 4

Tecnologías del Proyecto

Tipo	Nombre	Uso	Sección
API	SwellRT	Base de la que se parte para la Migración de Wave a Android	4.1.4
	Android	Plataforma para la Migración y el Desarrollo de la app.	4.2.1
Servidor	Wave In A Box (WIAB)	Servidor que aloja las Waves. Migración y Desarrollo de la App.	4.1.4
	OpenShift	Servidor que aloja el Service REST y la Base de Datos de la App.	4.2.10
Lenguaje	Java	Lenguaje base para trabajar con Android, tanto en la Migración como en la App.	4.1.6
	JavaScript	Lenguaje del antiguo cliente web de Wave presente en SwellRT.	4.1.8
	PHP	Lenguaje para definición del Service REST de la App.	4.2.8
	SQL	Lenguaje para definir e interactuar con la Base de Datos de la App.	4.2.5
	XML	Lenguaje para definición de interfaces en Android.	4.2.2
Formato de Datos	JSON	Formato para intercambio de información con Servidor Wave y REST.	4.2.4
Framework	GWT	Framework JavaScript utilizado por el antiguo Cliente web de Wave presente en SwellRT.	4.1.7
	Laravel 5	Framework de desarrollo del Service REST de la App en PHP.	4.2.9
Protocolo	Google Wave	Protocolo de intercambio de Waves usado para la migración y la App.	4.1.1
	HTTP	Protocolo de transferencia de datos por Internet usado por la Migración y por la app.	4.1.9
	WebSocket	Protocolo de transferencia de datos bidireccionales por Internet, usado por la Migración y por la App.	4.1.10
Base de Datos	MySQL	Sistema Gestor de la Base de Datos de la App	4.2.6
	PhpMyAdmin	Herramienta de Administración de la Base de Datos de la App.	4.2.7
Control de Versiones	Git	Software de control de versiones usado para la Migración y la App.	4.1.11
	GitHub	Plataforma web de control de versiones que aloja los desarrollos de la Migración y de la App.	4.1.12
Prototipado	POP	Aplicación para elaborar los Prototipos Interactivos basados en los Prototipos en Papel de la App.	4.2.12
IDE	Eclipse Luna	Entorno de Desarrollo utilizado para la Migración de Wave.	4.1.5
	Android Studio	Entorno de Desarrollo utilizado para el desarrollo de la App.	4.2.3
	Sublime Text	Editor de texto y de código fuente, utilizado para desarrollar el Service REST.	4.2.11

Cuadro 4.1: Tecnologías usadas en el Proyecto

4.1. Tecnologías de Wave

TABLA RESUMEN: Google Wave, Apache Wave, WIAB, SwellRT, GitHub, Java, GWT, JavaScript, HTTP

4.1.1. Google Wave

Ideado y presentado en 2009 por ingenieros de Google [7], Wave es a la vez un protocolo de comunicaciones [8] y una plataforma web de código libre, que permiten a sus usuarios comunicarse y colaborar entre sí en tiempo real (Ver sección 4.1.3) y de forma federada (Ver sección 4.1.3) a través de Internet. Inicialmente fue desarrollado con el objetivo de integrar en una sola plataforma servicios ampliamente utilizados como son el correo electrónico, las redes sociales y la mensajería instantánea. Pese al gran entusiasmo generado entre la comunidad de desarrolladores tras su anuncio, en el año 2010 Google anuncia el abandono del proyecto [9] debido a su poca acogida entre los desarrolladores y a que decide reorientar el uso de la tecnología hacia sus plataformas de edición de documentos Google Docs [10] y a su red social Google + [11]. Es en este momento cuando el desarrollo libre del proyecto pasa a manos de la Apache Software Foundation bajo el nombre de Apache Wave.

4.1.2. Apache Wave

Al cambiar de manos su desarrollo en 2010, la tecnología pasa a formar parte de la incubadora de la fundación Apache [12] como software de código libre bajo licencia Apache [13]. Así, se produce el desarrollo de Wave In a Box (WIAB) (Ver sección ??), plataforma que integra un cliente web sencillo y una implementación de un servidor Wave que cualquiera puede descargar y desplegar en su ordenador.

4.1.3. Características de Wave

Como plataforma de código libre desarrollada para ser utilizada en red, Wave hace uso de distintas tecnologías y protocolos bien conocidos. Entre sus características más destacadas están las siguientes:

Federación

El Protocolo Wave [8] fue desarrollado para utilizar un modelo federado [14] [15] de comunicación basado en la tecnología XMPP [16] [8]. Se trata por tanto de un modelo descentralizado en el que cualquiera de los participantes en la conversación es libre de actuar tanto como servidor como cliente sin que ello afecte a su participación en la conversación. Además, a diferencia de otras tecnologías (como el correo electrónico) en las que cada participante almacena su propia copia de la conversación y cada vez que hay cambios se debe transmitir la conversación entera a todos los participantes, Wave tiene la ventaja de que actúa de forma que es el servidor de la conversación el único que almacena la copia entera y se encarga de calcular los cambios que se han producido para transmitir solamente dichos cambios por la red a los participantes, con las consiguientes ventajas en términos de latencia que ello conlleva.

Consistencia en tiempo real

El Protocolo Wave [8] utiliza la tecnología de Transformaciones Operacionales (OT) [17] para garantizar la consistencia en la comunicación en tiempo real entre los participantes. Es decir, cualquier cambio producido por cualquiera de los participantes en la conversación se transmite automáticamente y en tiempo real al resto de los participantes sin pérdida de información y garantizando que los cambios se muestran en el estricto orden en el que se produjeron sin errores [18].

Escalabilidad

Wave fue desarrollado como un protocolo de alta escalabilidad que permite gestionar la existencia de una gran cantidad de conversaciones y participantes sin que por ello se resienta la productividad del sistema.

4.1.4. Servidores Wave

Wave in a Box

Wave In a Box (WIAB) [19] es el nombre de la implementación de un servidor Wave desarrollado por la Apache Software Foundation tras pasar el proyecto a sus manos en el año 2012. Al igual que el resto del código de la tecnología

CAPÍTULO 4. TECNOLOGÍAS DEL PROYECTO

que heredó de Google, está implementado en Java usando OpenJDK [20]. La instalación trae consigo un cliente web desarrollado en Javascript usando el framework Google Web Toolkit [21]. Este cliente web sirve como prueba de concepto de las funcionalidades básicas del Modelo Conversacional de Wave, pudiendo gestionar waves, usuarios y extesiones. Actualmente cualquiera puede descargar y desplegar WIAB en su ordenador siguiendo los pasos que nos proporcionan en su wiki [22]. La aplicación se distribuye en forma de código fuente, accesible entre otras formas desde su repositorio de GitHub [23]. Existen asimismo servidores de prueba ya desplegados en Internet sobre los que se puede observar el funcionamiento de WIAB [24].

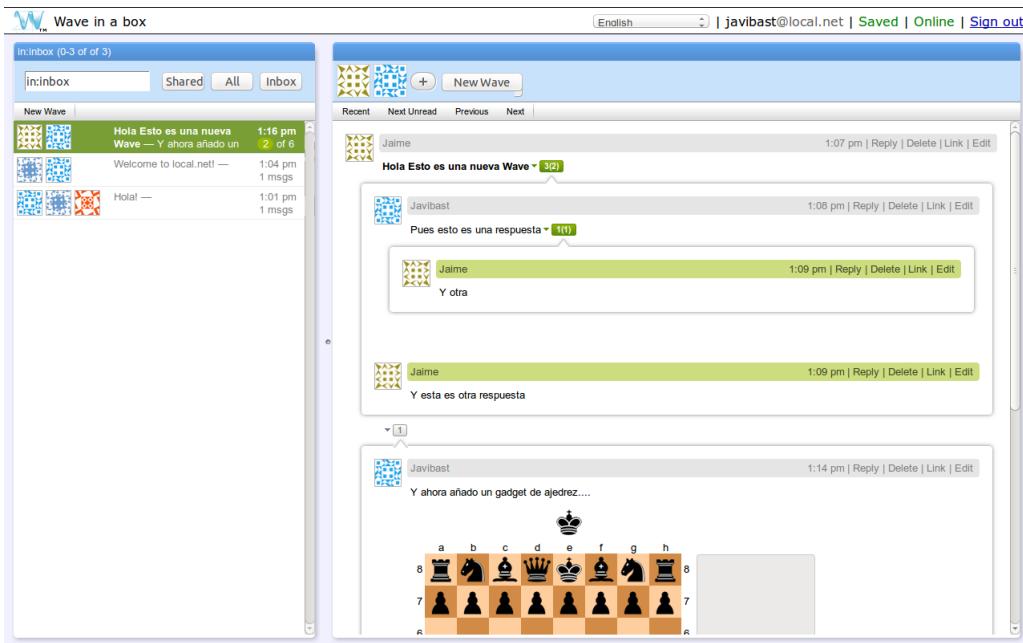


Figura 4.1: Cliente Wave In A Box

SwellRT

Como parte del proyecto europeo P2PValue [25] existe SwellRT, un fork de WIAB que amplía las características de éste último añadiendo un nuevo modelo de datos (Modelo de Datos Colaborativo) más allá del Modelo de Datos Conversacional de Wave original. Proporciona también un API escrito en Java que permite trabajar sobre los datos de ese nuevo modelo en forma de tres tipos básicos: mapas, listas y strings. Es por tanto un framework de colaboración en tiempo real que basa su funcionamiento en Apache Wave y cuyo principal popósito es permitir la integracion de la tecnología Wave en

otras aplicaciones, que podrán compartir objetos (de los tipos antes mencionados) de forma federada y en tiempo real. Su código fuente está disponible en GitHub [26], así como sus instrucciones de instalación (Ver el Readme en GitHub).

Para este proyecto se ha usado el framework SwellRT como base para la migración de la tecnología de Apache Wave a la plataforma Android [27]. Se pretende con esto que SwellRT haga uso de las funcionalidades nativas de Android.

4.1.5. Eclipse

4.1.6. Java

4.1.7. GWT

4.1.8. JavaScript

4.1.9. HTTP

4.1.10. WebSocket

4.1.11. Git

4.1.12. GitHub

4.2. Tecnologías de la Aplicación Android

TABLA RESUMEN: Android, JSON, Java, PHP, MySQL, SwellRT, Laravel, OpenShift, PhpMyAdmin, POP

CAPÍTULO 4. TECNOLOGÍAS DEL PROYECTO

4.2.1. Android

4.2.2. XML

4.2.3. Android Studio

4.2.4. JSON

4.2.5. SQL

4.2.6. MySQL

4.2.7. PhpMyAdmin

4.2.8. PHP

4.2.9. Laravel 5

4.2.10. OpenShift

4.2.11. Sublime Text

4.2.12. POP: Prototyping On Paper

Capítulo 5

Metodología del Proyecto

5.1. Uso de Software Libre

5.2. Metodología de Migración de Wave a Android

5.2.1. Objetivo

El framework de SwellRT utiliza un servidor WIAB y el protocolo Wave, ambos desarrollados en Java. El **SDK de Android** [33] es compatible con Java, así que a priori la implantación del servidor no supone problemas en los dispositivos móviles. Sin embargo, existe un problema con el API de SwellRT, ya que el lado del cliente fue desarrollado en Javascript usando el framework GWT. Android no soporta de forma nativa estas tecnologías, así que es necesario estudiar el código de SwellRT para sustituir todo el código que haga uso de Javascript/GWT por código compatible con Android. El objetivo de esta parte del proyecto es conseguir que un cliente desplegado en Android sea capaz de conectarse e interactuar con un servidor Wave sin problemas.

5.2.2. Plataforma: Entorno de Desarrollo, Construcción y Depuración

Existen dos entornos de desarrollo (IDE) recomendados por Google para desarrollar en Android: Eclipse [28] y Android Studio.[34] Eclipse es un entorno de desarrollo genérico que, mediante plugins, permite extender sus funcionalidades para desarrollar en diversas plataformas y lenguajes. Android Studio es un IDE basado en el entorno de desarrollo Java IntelliJ IDEA [35]

adaptado para trabajar con todas las funcionalidades de Android. En el momento de empezar con la migración Android Studio se encuentra en fase beta de desarrollo, pues Google pretende convertirla en el IDE de desarrollo oficial para Android. Mientras no se lanza la versión final de Android Studio, Google recomienda utilizar Eclipse para desarrollar en Android, y las guías para desarrolladores Android están escritas para Eclipse. En consecuencia tomamos la decisión de utilizar el entorno de desarrollo Eclipse para la migración de SwellRT a Android.

Eclipse

El IDE de Eclipse [28] soporta el desarrollo con Android a través del plugin **ADT (Android Development Tools)** [29], que integra en un solo paquete todas las herramientas necesarias para desarrollar, construir y depurar el código de la aplicación fácilmente.

Android SDK [33]: paquete que integra el conjunto de herramientas necesarias para desarrollar en Android. Entre estas herramientas destacan las siguientes:

- **Librerías con el API** de Android y **Documentación** asociada [36]
- **Android Virtual Device Manager (AVDM)** [37] herramienta para gestionar la creación, modificación, ejecución y eliminación de emuladores en Android. Un **emulador** [38] es una máquina virtual que ejecuta una determinada versión de Android. Permite desplegar un dispositivo móvil en el ordenador que imita las características software y hardware de uno real para poder hacer pruebas de desarrollo sin necesidad de poseer un dispositivo con Android.
- **Android SDK Manager** [39] herramienta para gestionar las versiones de SDK y herramientas asociadas instaladas. Android se encuentra actualmente en la versión 5.1 (API 22), pero un desarrollador puede elegir desarrollar para una versión anterior si lo estima necesario, por lo que puede descargarse por separado dicha versión y mantener varias API si lo necesita.
- **Dalvik Debug Monitor Server (DDMS)** [40] herramienta que provee las características de entorno de depuración para las aplicaciones en desarrollo.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

Teniendo en cuenta la distribución actual de versiones instaladas en dispositivos Android [41] (Ver figura 5.1) se ha decidido realizar la migración de SwellRT con el API 19 de Android (Version 4.4 "KitKat"). El emulador desplegado para las pruebas de desarrollo utilizará por tanto Android 4.4 .

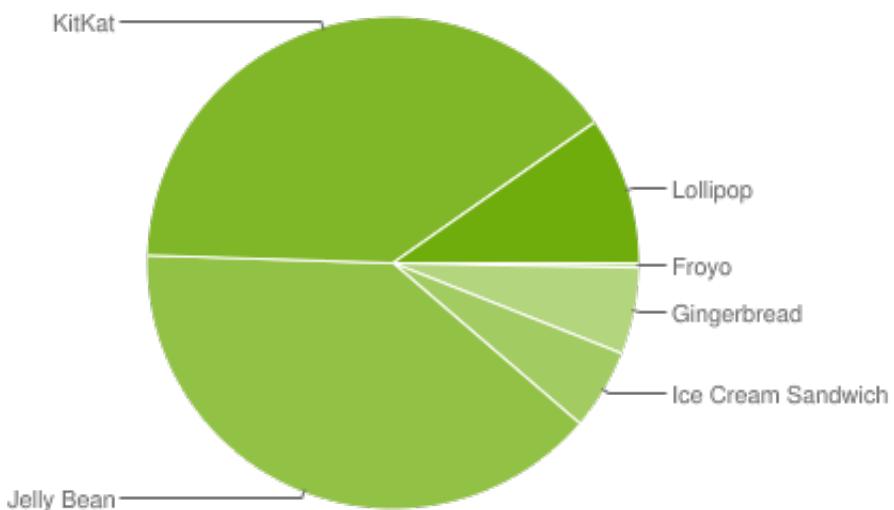


Figura 5.1: Distribución Actual de Versiones Android (Fuente: Google)

Sin embargo existe un problema con la construcción y depuración del código de SwellRT en Eclipse. Android **limita el número de métodos máximos de una aplicación a 65K** [42] por cuestiones de eficiencia. Para evitar esta limitación, durante el proceso de construcción el SDK de Android utiliza, entre otras, una herramienta llamada **ProGuard** [43]. Esta herramienta se encarga de optimizar el código de la aplicación buscando remover clases que no se utilizan y ofuscando el código para prevenir la ingeniería inversa. En el caso de SwellRT, el código posee un gran número de clases java necesarias para desplegar el servidor y el cliente de la herramienta, por lo que es necesaria dicha optimización de código realizada por ProGuard. El sistema de compilación de aplicaciones de Android tiene dos formas: compilación de la aplicación en modo debug (para hacer pruebas cuando todavía se encuentra en fase de desarrollo) y en modo release (la aplicación se encuentra en su versión final y se empaqueta y se firma digitalmente para lanzarla al público). En el caso de Eclipse, ProGuard solo se ejecuta cuando se construye en modo release, por lo que cuando se intenta compilar una aplicación con tantas clases como SwellRT mientras se desarrolla (modo debug) el sistema da error y no se puede compilar el código para probarlo en el emulador.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

La solución que encontramos fue desarrollar en Eclipse (por las facilidades que el entorno proporciona para escribir código) pero realizar el proceso de construcción del código por consola de comandos, ya que en este caso sí que se puede compilar la aplicación en modo debug utilizando ProGuard.

Proceso de Construcción por Consola

Para construir la aplicación por consola de comandos, Android utiliza la herramienta Apache Ant [44] para automatizar el proceso de construcción [45]. Es importante asimismo tener definida la variable de entorno JAVA_HOME con la ruta de acceso al JDK de java instalado en la máquina. Conviene también, por comodidad a la hora de trabajar con la consola, añadir al PATH del sistema las rutas a la carpeta donde está el SDK de android (/sdk) y dentro de esta ruta añadir asimismo rutas a las carpetas /tools y /platform-tools.

Existen dos formas de realizar la construcción en modo debug de una app:

1 - Sin tener previamente lanzado un emulador o conectado al ordenador un dispositivo android en modo debug [46]:

En este caso es necesario construir la aplicación y luego lanzar el emulador para después instalar la aplicación en él. Para construir la aplicación en modo debug nos vamos a la carpeta raíz de nuestro proyecto y ejecutamos el siguiente comando:

```
$ ant clean debug
```

Esto nos generará una aplicación instalable en el directorio /bin del proyecto bajo el formato que Android usa para sus aplicaciones (.apk). El siguiente paso es ejecutar un emulador o conectar un dispositivo android por USB. Para ejecutar un emulador, abrimos otra consola y utilizamos el siguiente comando:

```
$ android avd
```

Lo que nos despliega la herramienta Android Virtual Device Manager (Ver Sección 5.2.2) para que elijamos/creemos el emulador que queremos ejecutar. Podemos elegir multitud de parámetros [?] para el dispositivo que emula (resolución y tamaño de pantalla, de memoria Ram, elementos hardware emulados, etc.) siendo lo más importante elegir un API (versión de Android)

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

que se corresponda con el API que hemos elegido para nuestra aplicación (en nuestro caso API 19). Es recomendable también elegir una imagen del sistema que use un procesador con arquitectura Intel x86, ya que si elegimos la opción por defecto de ARM (los dispositivos móviles actuales usan procesadores ARM) la ejecución del emulador se ralentiza mucho al tener que emular una arquitectura de procesador distinta a la suya (los ordenadores actuales usan arquitectura Intel x86 en su mayoría). Esto únicamente afecta al rendimiento del emulador, la aplicación es independiente de la arquitectura que haya por debajo.

Una vez lanzado el emulador/dispositivo móvil, procedemos a instalar la aplicación en él ejecutando el siguiente comando en la primera consola (en la que construimos la aplicación):

```
$ adb install XXXX.apk
```

Siendo XXXX la ruta a donde se encuentra el .apk de la aplicación que previamente hemos construido (/bin). La herramienta ADB (Android Debug Bridge) [47] es la que permite la comunicación entre el proceso de la consola de comandos y el emulador/dispositivo móvil. Es importante destacar que si se tienen varios emuladores/dispositivos móviles en ejecución/conectados hay que especificar en cuál se quiere instalar la aplicación añadiendo al comando lo siguiente: **-s emulator -YYYY** siendo esto último el identificador del emulador que podemos encontrar en el título de la ventana del emulador.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO



Figura 5.2: Emulador Android API 19

De esta manera podemos probar la aplicación, que será lanzada en el emulador/dispositivo una vez termine su instalación.

2 - Teniendo un emulador previamente lanzado (ver sección anterior para ver cómo se lanza) o un dispositivo móvil ya conectado por USB:

En este caso es todavía más sencillo el proceso de construcción. Nos vamos a la carpeta raíz del proyecto y podemos compilar e instalar la aplicación con un solo comando:

```
$ ant debug install
```

Es importante destacar que este comando solo funciona si tenemos un único emulador o dispositivo conectado, de lo contrario habrá que utilizar el método anterior.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

Proceso de Depuración

Una vez instalada una aplicación, podemos depurar su código en ejecución usando la herramienta DDMS del ADT en conjunto con la vista de Debug de Eclipse. Pero antes hay que especificar qué aplicación queremos depurar de las que puedan estar instaladas en el dispositivo o emulador.

En el caso del emulador debemos lanzar la aplicación llamada “Dev Tools” y abrir el menú “Developer Options”. Dentro de este menú habilitaremos las opciones de “USB debugging” y de “Wait for debugger”. Además pulsaremos sobre “Select Debug app” y seleccionaremos la aplicación que queremos depurar.

En el caso de un dispositivo Android debemos ir a los Ajustes del dispositivo y seleccionar el menú de Opciones de Desarrollador. Aquí habilitamos las opciones de ”Depuración de USB”(si no esta habilitada ya) y de .^Esperar al depurador”. Además pulsamos donde pone ”Seleccione una aplicación para depurar” elegimos la aplicación que queremos depurar.

Una vez hecho esto, cada vez que ejecutemos la aplicación saldrá un mensaje de advertencia y se quedará esperando a que conectemos un depurador para continuar con su ejecución. Para esto, nos vamos a Eclipse y abrimos la vista de DDMS. Aquí nos aparecerá, entre otras cosas, un espacio con todos los procesos en ejecución en el dispositivo/emulador. Localizamos el proceso de nuestra aplicación y pulsamos sobre el bichillo verde para conectar el depurador a ella. Llegados a este punto la aplicación continua su ejecución en el emulador y aparece un escarabajo verde al lado del proceso de la app en la ventana de DDMS, que indica que se está depurando ese proceso. Es entonces cuando podemos abrir la vista de depuración de Eclipse y proceder a trabajar con breakpoints para depurar y estudiar el código con el fin de solucionar errores.

5.2.3. Migración: Identificación y Solución de Problemas

El objetivo de esta parte del proyecto es conseguir que el cliente de SwellRT se pueda desplegar en Android para así conseguir que se conecte al servidor

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

WIAB que tambien incluye. Para ello lo primero que haremos será desplegar el servidor en nuestro ordenador clonando el repositorio de GitHub de SwellRT y siguiendo los pasos descritos en el Readme del proyecto [26]. Para comprobar que el servidor se ha instalado correctamente, podemos ejecutarlo por consola (ver Readme) y abrir un navegador web con la dirección `http://localhost:9898`. Si nos aparece una ventana de Login de WIAB es que ya tenemos un servidor WIAB corriendo en nuestro ordenador. Creamos entonces un usuario y contraseña de prueba. Este paso es importante ya que la aplicación Android intentará conectarse contra este servidor mientras estemos haciendo pruebas de desarrollo.

A continuación crearemos un proyecto Android en Eclipse e incluiremos en él todas las clases de SwellRT. Uno de los componentes principales de Android a la hora de desarrollar son las **Actividades** [48], que representan las pantallas que se le muestran al usuario y que responden a su interacción programáticamente. Por tanto, crearemos una nueva actividad principal (`waveAndroid.java`) que se ejecutará al lanzar la aplicación y que por el momento intentará conectarse al servidor especificando por código el usuario y contraseña que hemos creado antes en el servidor. Wave realiza este login contra el servidor usando dos tecnologías: HTTP [49] y WebSockets [50].

Conexión HTTP

Wave fue desarrollado para utilizar el protocolo WebSocket para la conexión al servidor, pero esta tecnología necesita realizar una autenticación HTTP previa. Lo primero que haremos será otorgar **permisos de conexión a internet** a nuestra aplicación. Android utiliza un **sistema de permisos** [51] para controlar los privilegios de cada aplicación. Estos permisos se declaran en el **Manifiesto** de la aplicación [52], archivo que declara sus características. Para ello basta con añadir lo siguiente al manifest.xml de la aplicación:

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

También hay que tener en cuenta que cuando nos encontramos en el emulador no estamos en la misma red que el ordenador en el que trabajamos, por lo que la conexión a la URL `http://localhost:9898` no es válida. No obstante, esto tiene facil solución pues el **emulador de Android define unas direcciones IP de red especiales** [53] para este tipo de casos. Basta con sustituir localhost por la dirección 10.0.2.2 para conseguir acceder al servidor WIAB desplegado en el ordenador. La dirección URL sera por tanto:

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

<http://10.0.2.2:9898>.

Lo siguiente que haremos será ejecutar el código de Login del cliente SwellRT para intentar localizar dónde se lleva a cabo la conexión HTTP. Para ello llamamos desde la actividad principal (WaveAndroid.java) al método startSession() de la clase WaveClient.java pasándole el usuario y la contraseña antes creados.

Esto provoca un error de ejecución y la aplicación se cierra. Lo siguiente que hacemos es depurar la aplicación (Ver Sección 5.2.2) estudiando el LogCat [54] (Ver Figura 5.3) para ver dónde se produce el error. Descubrimos que el problema estaba localizado en el método login() de la misma clase, que intentaba realizar una **petición POST HTTP** al servidor utilizando un **RequestBuilder** de la librería **com.google.gwt.http.client**. He aquí el primer problema: la actual conexión utiliza métodos de GWT/Javascript para hacer la petición Post y Android no es compatible con esta tecnología.

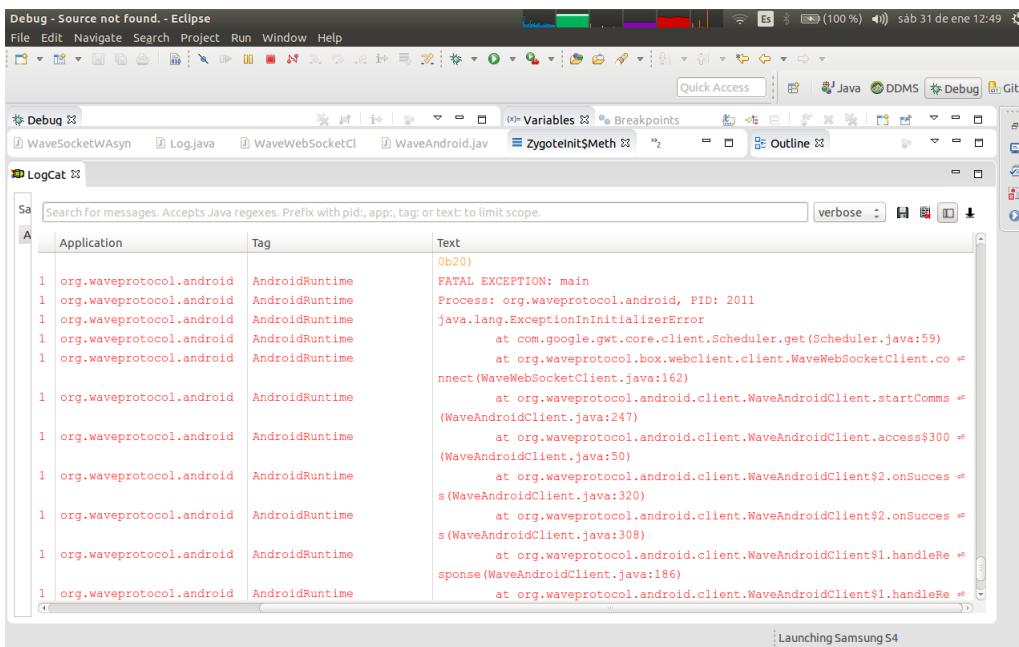


Figura 5.3: Ejemplo de Traza de Error en Logcat

Hay por tanto que encontrar una librería similar compatible con Android que construya una petición **HTTP POST** y la envíe al servidor. La primera opción que valoramos fue utilizar la **librería HTTP Apache** [55], incluida en el SDK de Android desde sus primeras versiones. Sin embargo, Google recomienda [56] a partir del API 10 (Android 2.3 "Gingerbread") utilizar la

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

librería **HttpURLConnection**[57], también incluida en el API. Por tanto esta última es la que elegimos para la migración.

De forma simplificada, éste sería un esquema de la nueva estructura del login HTTP:

```
import java.net.HttpURLConnection;

private void login(final String user, final String password,
    final Callback<String, String> callback) {

    //Construct the URL String urlStr with the
    //server, user and password parameters

    URL url = new URL(urlStr); //String
    HttpURLConnection connection = (HttpURLConnection)
        url.openConnection(); //Open the connection
        to the given URL

    connection.setDoOutput(true); // allow the POST
        connection
    connection.setRequestProperty("Accept-Charset",
        CHARSET);
    connection.setRequestProperty("Content-Type",
        "application/x-www-form-urlencoded; charset=" +
        CHARSET);

    OutputStream out = connection.getOutputStream();
    out.write(queryStr.getBytes(CHARSET)); //Set the
        POST parameters

    if (connection.getResponseCode() != 200) {
        //ERROR during the connection
        connection.disconnect(); //Disconnect
            from the server.
    } else {
        //Continue with the login process (
        //WebSocket)
        connection.disconnect(); //Disconnect
            from the server.
    }
}
```

Sin embargo, aquí no acaba el problema. Por cuestiones de usabilidad y de respuesta a la interacción del usuario, Android establece dos reglas para trabajar con el proceso de la actividad que se le está mostrando al usuario (llamado **UI Thread**) [58]:

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

- 1. No bloquear el UI Thread
- 2. No acceder al UI Thread directamente desde otro Thread

La conexión a un servidor es un proceso susceptible de durar un tiempo variable según las condiciones de la red, lo cual deja la aplicación en espera hasta que se realiza dicha conexión, bloqueando el UI Thread. Por tanto, decidimos usar un hilo (Thread) por separado en forma de **AsyncTask** [?] para llevar a cabo la tarea de Login, tal y como recomienda Google hacer para trabajar con conexiones a la red [59]. La ventaja por tanto de usar otro hilo para esto es que la actividad principal no se bloquea.

Es importante también destacar que la arquitectura de SwellRT y de Wave está planteada de manera que utiliza llamadas asíncronas (callbacks) para notificar al resto de la aplicación del resultado de los procesos de conexión al servidor, por lo que nuestro AsyncTask tendrá que usar el callback apropiado para notificar del éxito o fracaso de la conexión Http.

El siguiente es un esquema del AsyncTask encargado del Login:

```
private class LoginTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... params) { // method that executes on the new Thread without
        blocking the UI Thread
        login(params[0], params[1], params[2]); //Do the login
        return sessionId //String needed for the WebSocket
            connection and based on the cookie received from the
            server.
    }
    @Override
    protected void onPostExecute(String result) { //method
        that executes on the UI Thread once doInBackground()
        finishes its execution.
        if (result != null) {
            callback.onLogin(); //Notify the login success using
                the proper callback method
        } else { //The doInBackground method has had a problem
            and the result of its execution was null
            callback.onError("Wave>Login>Error"); //Notify the
                login error using the proper callback method
        }
    }
}
```

}

Este proceso de conexión Http nos deberá devolver una Cookie que trataremos con el objetivo de generar un SessionId que será necesario para seguir con la conexión al servidor. **Llegados a este punto, tenemos un proceso de login Http que hace uso de la librería HttpURLConnection y de un AsyncTask para realizar esa primera conexión al servidor.** Depuramos la aplicación y comprobamos que efectivamente el login Http se realiza correctamente (la respuesta del servidor tiene código 200). **Sin embargo la aplicación aún no funciona correctamente, pues se cierra al intentar ejecutar el código que se encarga del siguiente paso de la conexión: conectarse por WebSocket.**

Conexión WebSocket

Para realizar una conexión con el servidor Wave es necesaria una conexión mediante WebSockets[50], tecnología que permite conexiones bidireccionales y asíncronas entre el servidor y el cliente (recordemos que la conexión en el modelo cliente-servidor tradicional está definida como unidireccional de cliente a servidor), de manera que cualquiera de los dos puede iniciar una conexión con el otro en cualquier momento e intercambiar información con éste. En el caso del protocolo Wave este comportamiento es el deseable, ya que al tratarse de un protocolo de comunicaciones federado en el que cualquiera en la red puede ser cliente o servidor, es importante que la conexión sea bidireccional. Además la asincronía es necesaria ya que para mantener la consistencia en tiempo real 4.1.3 hace falta que el servidor que contiene las waves pueda iniciar una conexión con los clientes para notificar los cambios que se produzcan en dichas waves. Por tanto, nuestro cliente Android debe ahora establecer una conexión WebSocket con el servidor WIAB.

La metodología a utilizar será la misma que para la conexión HTTP, se ejecutará el código de SwellRT para identificar dónde falla y por tanto cómo está estructurada la creación y gestión de WebSockets en la versión GWT.

El cliente SwellRT original realiza esta conexión utilizando una librería llamada Atmosphere [60], que proporciona un framework para Java que permite gestionar conexiones WebSocket junto a la conexión HTTP que subyace por debajo. Sin embargo, esta librería se encarga solo de gestionar la conexión, no de crear el WebSocket propiamente dicho. En el caso de SwellRT este WebSocket se crea utilizando la implementación que proporciona GWT llamada también WebSocket (WebSocket.java). Esta clase nos define las funciones

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

básicas que debería tener nuestro WebSocket: **onOpen()** para establecer la conexión, **onMessage()** para recibir mensajes por el WebSocket, **send()** para enviar mensajes y **onClose()** para cerrar la conexión. Asimismo nuestro Websocket deberá también implementar una serie de callbacks (definidos en la interfaz `WebSocketCallback.java`) para notificar a la aplicación de la llegada de estos eventos del servidor. Los callbacks son: **onConnect()**, **onDisconnect()** y **onMessage(message)** respectivamente.

Este WebSocket se crea utilizando un **patrón de diseño Factory**, que abstrae la creación de un objeto de su implementación, de manera que el desarrollador tenga acceso al objeto sin tener que preocuparse de cómo este implementado el WebSocket por debajo (ver `WaveSocketFactory.java` en SwellRT). En este caso, como ya se ha dicho, con Atmosphere y WebSoc- ket GWT. No obstante, la aplicación no funciona tal y como está hecho en SwellRT ya que android no soporta GWT de forma nativa. Hay que sustituir este código buscando una librería open-source que implemente un WebSoc- ket en Android sobre Atmosphere y que porporcione las mismas funciones básicas descritas en el párrafo anterior.

La solución encontrada fue utilizar `wAsync`[61], librería proporcionada por Atmosphere para trabajar con Websockets en Node.js, Java y Android. `wAsync` trabaja creando un socket que responde a eventos diversos, estando entre ellos eventos similares a los utilizados por la versión GWT de SwellRT: **on(EVENT.name())**, siendo EVENT el nombre del evento al que debe responder. Un ejemplo sencillo de utilización de `wAsync` sería el siguiente:

```
//Create the atmosphere client
AtmosphereClient client = ClientFactory.getDefault().
    newClient(AtmosphereClient.class);
//Configure client with URL
SphereRequestBuilder requestBuilder = client.
    newRequestBuilder()
.method(Request.METHOD.GET).trackMessageLength(true).uri(
    WaveSocketWAsync.this.urlBase)
.transport(Request.TRANSPORT.WEBSOCKET)
//Create and configure socket
SocketWAsync.this.socket = client.create(client.
    newOptionsBuilder().runtime(ahc).build())
.on(Event.OPEN.name(), new Function<String>() { //Equivalent
    to GWT onOpen() method
@Override
public void on(String arg0) {
    // set the actions to do and call the proper
    // callback function (callback.onConnect())
}
}).on(Event.CLOSE.name(), new Function<String>() { //
```

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

```
Equivalent to GWT onClose() method
@Override
public void on(String arg0) {
    // set the actions to do and call the proper callback
    function (callback.onDisconnect())
}
}).on(Event.MESSAGE.name(), new Function<String>() {
@Override
public void on(String arg) { //Equivalent to GWT onMessage()
    () method
    // set the actions to do and call the proper callback
    function (callback.onMessage())
}
).on(new Function<Throwable>() {
@Override
public void on(Throwable t) {
    // catch possible exceptions
}
});
{
    // connect to the server
cket.open(requestBuilder.build());
tch (IOException e) {
    // catch possible exceptions

nd a given message to the server, equivalent to GWT send(msg)
    method
et.fire(Data);
```

Como la arquitectura Wave utilizaba el patrón factoria, fue necesario sustituir la clase de WebSocket GWT por una de nueva creación llamada **WaveSocketWAsync.java**[62] que implementa los métodos de creación y configuración de un Websocket y de callback antes descritos. Asimismo se modificó la clase `WaveSocketFactory.java` para hacer uso ahora de esta nueva implementación del WebSocket compatible con Android.

Sin embargo, ejecutamos esta nueva versión de código y nos encontramos con que la librería WAsync incluye dependencias a código que no está presente en la propia librería y que es necesario para crear el cliente `AsyncHTTPClient` que gestiona la conexión HTTP que subyace por debajo del WebSocket. Concretamente hace falta utilizar un `HTTPProvider` compatible con Atmósfera, tal y como recomienda hacer wAsync en su wiki???. Para ello basta con añadir al proyecto las librerías oportunas (Ver Tabla de dependencias 5.1) y configurar el cliente segun lo descrito en dicha wiki:

```
AsyncHttpClientConfig ahcConfig = new AsyncHttpClientConfig.
    Builder().build();
```

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

```
cHttpClient ahc = new AsyncHttpClient(new  
GrizzlyAsyncHttpProvider(ahcConfig));
```

Ahora, al ejecutar la aplicación comprobamos que la conexión al servidor se produce correctamente. Para completar esta parte del proyecto solo falta pedir al usuario su user y password, ya que hasta ahora las habíamos especificado a mano en el propio código para realizar pruebas.

Conexión final y Logging

Para probar que la aplicación migrada es funcional y es capaz de utilizar las características nativas de Android haremos una pequeña y sencilla pantalla de Login que pedirá al usuario la dirección del servidor Wave, un usuario y una contraseña.

En el diseño de aplicaciones Android el componente principal de una app es la Actividad, que se corresponde con la pantalla con la que interactúa el usuario. La interfaz gráfica se usuario (UI) de las pantallas se encuentra separada del código de la aplicación en ficheros xml de Layout[?]. A una Actividad se le especifica cuál es su archivo de layout en su método onCreate(), responsable de la creación de la actividad y sus recursos.

Creamos una Actividad llamada WaveAndroid.java que haga uso del layout Main.xml, en el cual incluimos tres cajas de texto (llamadas EditText en Android) para que el usuario introduzca los datos. Además guardamos una referencia a estas cajas de texto en la Actividad para poder acceder al texto introducido y pasárselo al método login de Wave que hemos migrado anteriormente.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

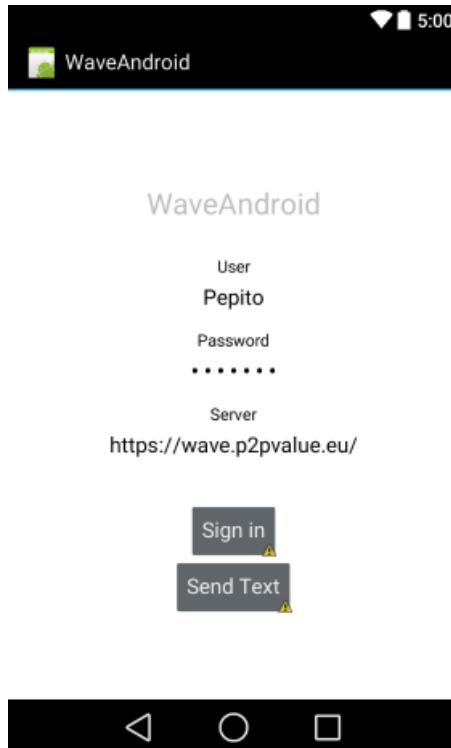


Figura 5.4: Pantalla de Login de WaveAndroid

Además decidimos mejorar el sistema de mensajes de Log de la aplicación sustituyendo el framework de la librería SLF4J para Java usada por SwellRT por una versión más reciente desarrollada para Android[63].

Por último instalamos la aplicación en el emulador o el dispositivo móvil y probamos que se nos muestra la pantalla de login anterior. Introducimos los datos del servidor WIAB (en este caso utilizaremos el servidor de P2PValue desplegado para pruebas en <https://wave.p2pvalue.eu/>), de usuario, contraseña y comprobamos que hemos conseguido el objetivo de esta parte del proyecto: **nuestro cliente Android realiza el login contra el servidor WIAB correctamente.**

Organización del código: Servicio Android

Una vez conseguida la conexión al servidor desde Android, decidimos revisar el código para intentar optimizarlo y organizarlo de manera que aprovechara mejor las características de Android y la arquitectura de Wave. Además, el API que permite gestionar el modelo de datos de SwellRT (Ver Sección 4.1.4)

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

está escrito en java, por lo que es plenamente funcional y compatible con el código de nuestra migración a Android. A continuación hablaremos de los motivos de dicha reorganización.

Como ya se ha comentado anteriormente, el proceso de login se debe hacer en un hilo de ejecución separado del hilo principal o UI Thread, ya que Android no recomienda[58] que tareas que tarden mucho tiempo en ejecutarse (como por ejemplo descarga de datos de la red) se ejecuten en el mismo hilo que la interfaz de usuario, pudiendo bloquear dicho hilo y obstaculizando por tanto la interacción del usuario con el dispositivo.

Hasta ahora habíamos utilizado para ello una Actividad que contenía el AsyncTask [64] encargado de ejecutar el código de conexión al servidor en un hilo separado del UI Thread. Sin embargo, tal y como está definida la arquitectura de Wave y de SwellRT, la utilización de callbacks (ver secciones 5.2.3 y 5.2.3) es necesaria para notificar al resto de la aplicación de los eventos relacionados con el intercambio de datos con el servidor. Un AsyncTask ejecuta de una sola vez y de forma asíncrona el código que se le asigne a su método doInBackground(), de manera que una vez que termina su ejecución puede notificar el resultado de la conexión, pero no queda a la espera de otros posibles eventos en el socket (como la recepción de mensajes o la desconexión). Es decir: **aunque se definan callbacks para esperar los eventos del servidor, el socket no podrá notificarlo porque no existe un hilo que quede a la espera de estos eventos.**

Por otro lado, cada vez que quisiéramos realizar algún tipo de interacción con el servidor habría que utilizar un AsyncTask específico para ello, lo cual no hace sino añadir más código a la aplicación. Otra desventaja de los AsyncTask en la implementación inicial es que si la Actividad que lo esta ejecutando pasa a segundo plano (por ejemplo si el usuario cambia de aplicación) se detiene el proceso de conexión.

Considerando todo esto decidimos utilizar otro componente de Android pensado para ejecutar tareas en background y con la opción de hacerlo de forma independiente de la aplicación: el Servicio[65].

Un Servicio Android se diferencia de una Actividad en que es un componente que se ejecuta en segundo plano y no proporciona una interfaz de usuario con la que éste pueda interactuar. Un Servicio ejecuta tareas de larga duración (como la descarga de datos de la red) a petición de otros componentes de la aplicación que estén *suscritos* (el término utilizado por android es *bind*) a dicho Servicio, a modo de cliente-servidor. De esta manera la Actividad (cliente) que se suscriba al Servicio (servidor) puede interactuar con éste último haciendo peticiones y recibiendo notificaciones cuando el Servicio ob-

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

tenga resultados. Por tanto, podremos acceder a la conexión al servidor desde cualquier punto de la aplicación, solo es necesario que la Actividad en ejecución se suscriba al Servicio para hacerlo.

Pero un Servicio se ejecuta dentro del mismo proceso que el UI Thread, por lo que aun así tendremos que utilizar métodos para ejecutar el código que nos interese en otros hilos de ejecución dentro del propio Servicio. De esta manera se dividió la reorganización en dos: la conexión Http y la conexión WebSocket.

Para la conexión HTTP, se decidió utilizar un AsyncTask llamado LoginTask muy similar al ya explicado anteriormente (ver Sección 5.2.3) pero esta vez definido dentro del propio Servicio y con un callback que notificaba a la aplicación si la conexión se realizaba correctamente. Además, se puso el código de la conexión en una clase aparte llamada WaveHttpLogin.java para tenerlo más organizado. El siguiente es un esquema en forma de Diagrama de Secuencia de la conexión Http:

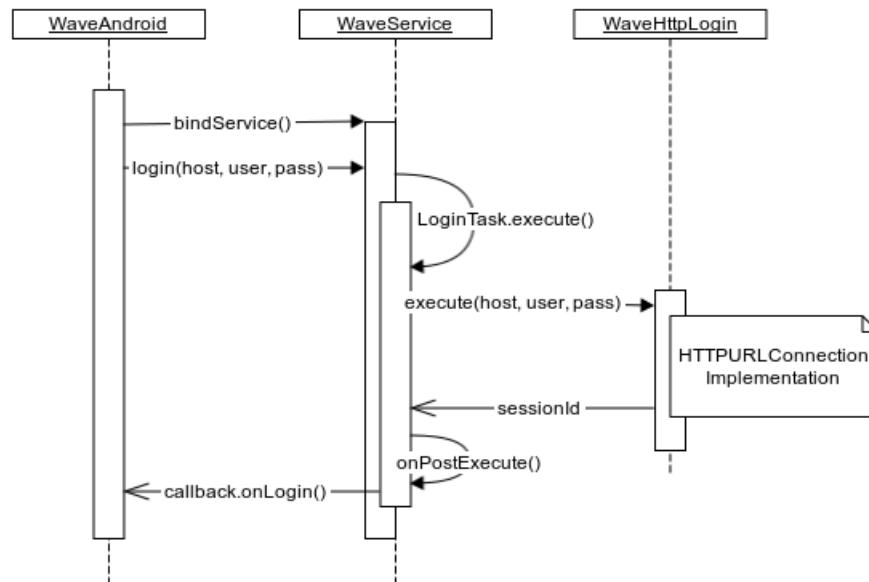


Figura 5.5: Proceso de conexión Http con Servicio

Como vemos, cuando se realiza esta conexión Http la Actividad inicial (WaveAndroid.java) es informada de ello mediante su callback onLogin() para poder notificar al usuario e iniciar la conexión por WebSocket.

Para dicha conexión se debía buscar una solución que permitiera al Web-

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

Socket responder a eventos del servidor (recordemos el funcionamiento de WAsync descrito en la Sección 5.2.3) para, mediante callbacks, informar a la aplicación de dichos eventos. **La solución final encontrada fue combinar el uso de un Thread y un Handler.** El Thread se encarga de crear el socket y configurar su respuesta a eventos en forma de mensajes, que enviará entonces al Handler, encargado de quedar a la espera de recibir estos mensajes de forma asíncrona y llamar al callback adecuado. Esta implementación se puede ver concretamente dentro de la clase WaveSocketWAsync.java, donde se define el Thread llamado WebSocketRunnable y el Handler UIHandler. El siguiente es un esquema en forma de Diagrama de Secuencia de la conexión inicial con WebSocket al servidor Wave:

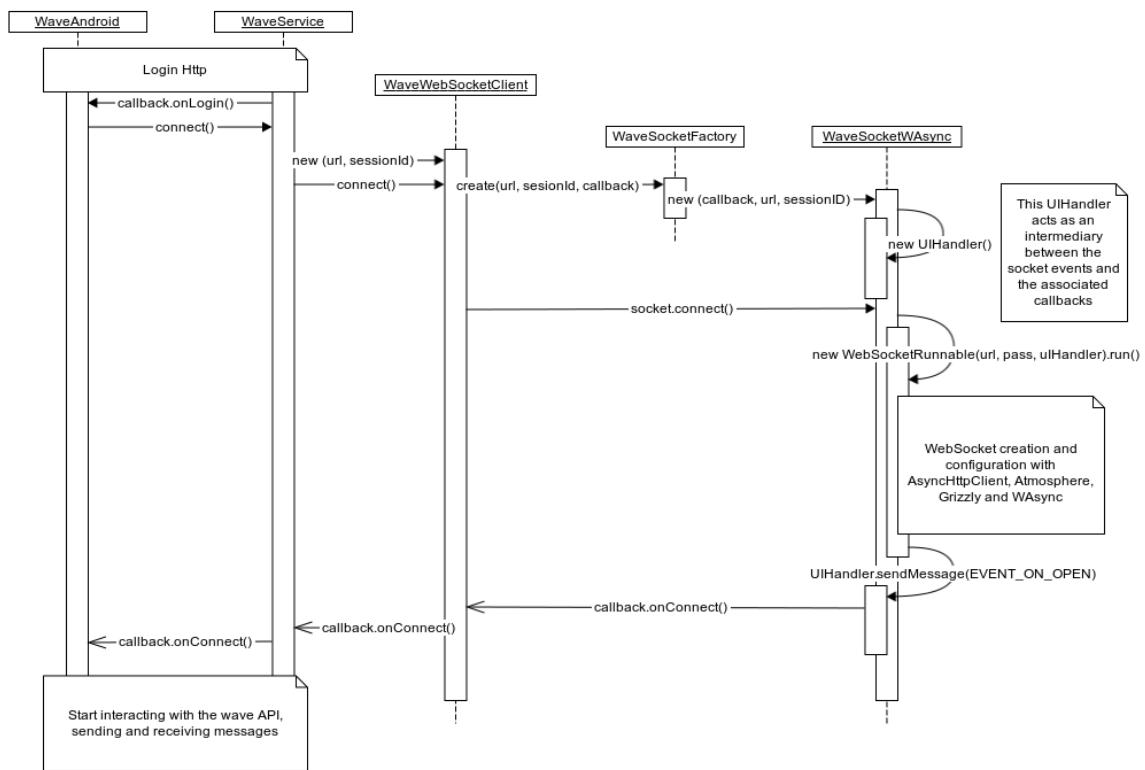


Figura 5.6: Proceso de conexión WebSocket con Servicio

Como vemos, cuando se realiza esta conexión al Servidor Wave la Actividad es notificada de ello mediante la propagación de callbacks onConnect() que informan del éxito de la conexión. A partir de este momento el socket informará de la misma forma (mediante el envío de mensajes al UIHandler) de otros eventos que se produzcan, ya sea la recepción de datos o la desconexión.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

por ejemplo. De la misma forma la aplicación podrá enviar datos al servidor, ya que el socket se mantiene en el Servicio.

5.2.4. Dependencias

El cliente Android de SwellRT migrado y desarrollado en esta parte del proyecto hace uso de las siguientes dependencias con librerías externas a Android:

Nombre	Versión	Descripción
WAsync [61]	1.4.3	WebSockets/HTTP Client Library for Asynchronous Communication
AsyncHttpClient [66]	1.8.14	Library that allows Java applications to easily execute HTTP requests and asynchronously process the HTTP responses.
Grizzly-Framework [67]	2.3.18	Core framework for Grizzly applications that provides TCP/UDP transports, memory management services/buffers, NIO event loop/filter chains/filters.
Grizzly-Http [67]		HTTP framework that contains the base logic for dealing with HTTP messages on both the server and client sides.
Grizzly- WebSockets [67]		WebSockets custom API for building Websocket applications on both the server and client sides.
slf4j-Android [63]	1.6.1	Simple Logging Facade for Java: logging framework for Android

Cuadro 5.1: Dependencias de SwellRT-Android

5.2.5. Resultado de la Migración

Después de todos estos pasos disponíamos de una versión funcional de SwellRT capaz de conectarse al servidor WIAB de forma nativa desde Android. **El resultado de esto se puede ver en el GitHub de esta parte del proyecto[62].** El siguiente es un esquema en forma de Diagrama de Clases del resultado final:

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

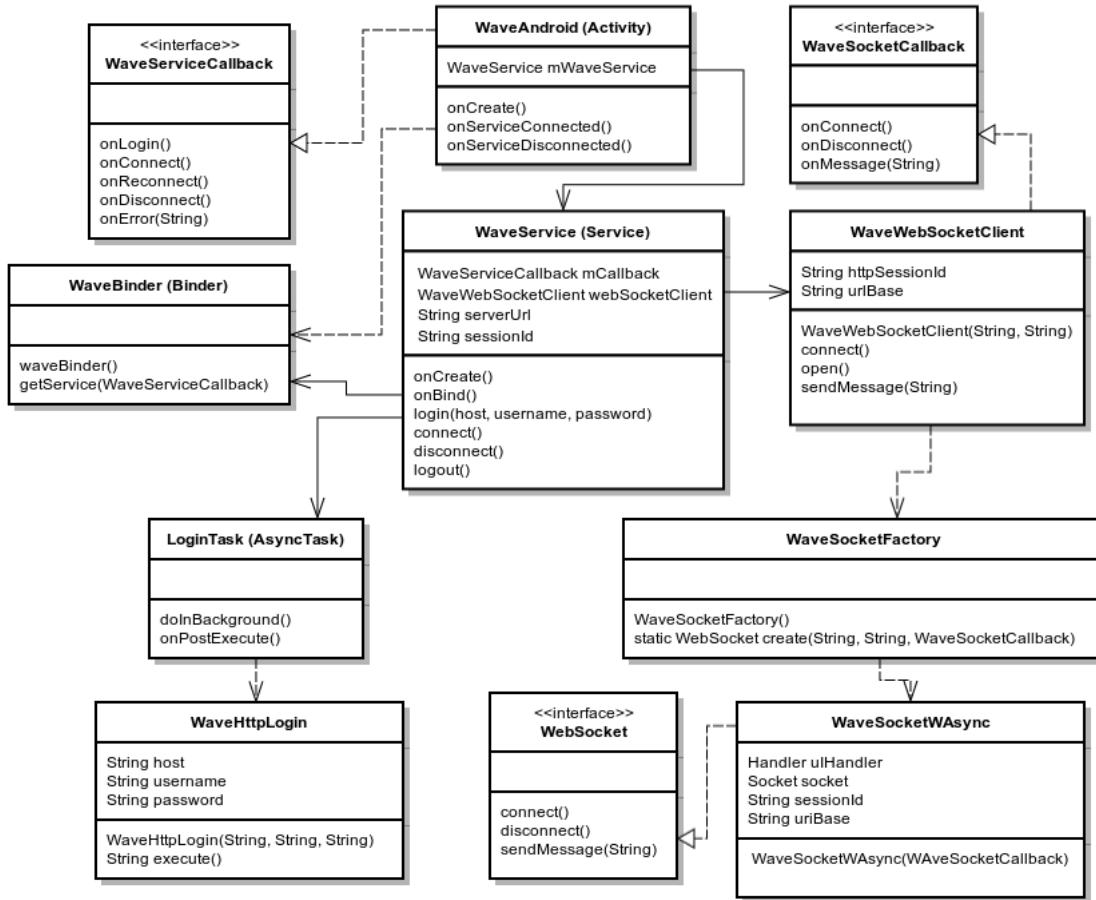


Figura 5.7: Esquema de Clases de SwellRT-Android con Servicio

Solo restaba poner el API de SwellRT encima de lo nuestro para poder acceder y trabajar a nivel de wave con el modelo de datos de SwellRT. De esto se encargó uno de los desarrolladores del proyecto inicial, Pablo Ojanguren, con el cual habíamos trabajado para realizar lo anteriormente descrito.

El API con el cliente de SwellRT adaptado a Android, con el cual trabajaremos en la siguiente parte del proyecto para crear una aplicación Android que haga uso de ello, se encuentra en el GitHub de SwellRT[68].

5.3. Diseño de la Aplicación: Diseño Guiado Por Objetivos

Para elaborar el diseño de la aplicación, nos hemos basado en la metodología del Diseño Guiado por Objetivos, que implementa el proceso de la Ingeniería de la Usabilidad propuesto por Alan Cooper [69]. Este proceso constará de las siguientes fases:

EXPLICARLOS: TEMA 3 PAGINA 21 DS1

- 1. Investigación
- 2. Modelado
- 3. Definición de Requisitos
- 4. Framework de Diseño

Queremos insistir eso sí en que únicamente nos hemos basado en esta metodología para seguir el proceso del diseño de la aplicación. No seguiremos todas las fases de esta metodología al pie de la letra ya que el alcance en tiempo de este proyecto escapa a una metodología tan formal y laboriosa (que implica gran cantidad de pruebas y procesos) como esta.

5.3.1. Investigación

Intención Inicial: Prototipo básico

¿HABLAR DE INTENCION FUTURA PARA ELECCIONES?

Programas electorales

La intención fundamental de la aplicación es llevar los programas electorales a los bolsillos de los ciudadanos y generar interés en participar más activamente en la política, ya sea emitiendo opiniones sobre dichos programas o elaborando nuevas propuestas. Vivimos en una sociedad digital, donde cada vez son más las personas que utilizan sus smartphones para realizar todo tipo de tareas en su vida cotidiana.

En los últimos años las diferentes formaciones políticas han subido sus programas electorales a un documento en formato PDF que suele estar disponible para su descarga en su página web. Este documento tiene generalmente una

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

gran extensión (los hay de 200 páginas), lo cual no hace sino dificultar que las personas se animen a leerlo. Por ello pensamos que una aplicación que pudiera visualizar las principales secciones de los programas políticos podría ser especialmente útil para acercar los programas a los electores.

Además también intentaríamos darle una estructura a estos programas, de manera que el usuario pudiera navegar por ellos a nivel de Sección, a diferencia del método actual de leer un "macro-documento.^{en} PDF. Así, la gente podría opinar sobre los programas políticos a nivel de sección mediante acciones familiares para ellos: "Me gusta", "No me gusta" realizar Comentarios". Añadimos también una acción de "No lo entiendo" que pensamos que sería útil para indicar cuándo la redacción de la sección era de significado difuso.

En definitiva, queríamos crear un espacio donde poder informarse sobre las distintas ofertas electorales y poder debatir sobre las propuestas que propone cada formación política, todo ello en forma una aplicación que podremos consultar en cualquier momento.

Propuestas y Wave

Por otro lado, y en línea con los últimos movimientos políticos ciudadanos, pretendíamos crear también un portal de propuestas ciudadanas en el móvil. Los usuarios podrían visualizar las propuestas de otros usuarios y tener la posibilidad de crear nuevas propuestas. Además, como queríamos aprovechar las características de la migración de Wave previamente desarrollada, pensamos en la posibilidad de elaborar estas propuestas de forma colaborativa y en tiempo real entre muchos usuarios.

Actualmente existen multitud de portales (en su gran mayoría web) donde la ciudadanía puede expresar su opinión, pero creemos que la integración de una aplicación donde puedan situarse las opiniones de los partidos políticos (en forma de sus programas) y la actividad ciudadana (en forma de propuestas), genera una nueva manera de tratar la política en los medios sociales.

También pensamos en la posibilidad de categorizar el contenido de la aplicación (Programas y Propuestas) por temas, para proporcionar filtros a la hora de navegar por dicho contenido. Sin embargo no teníamos muy clara la elección de temas, así como si debíamos darle al usuario la posibilidad de crear nuevos temas o dar nosotros unos temas preestablecidos.

Una vez pensada la intención y los principales objetivos de la aplicación, procedimos a realizar unos primeros prototipos en papel de nuestra idea y a implementar un prototipo básico en el móvil (Ver Sección 5.3.4) que mostraba programas políticos estructurados y permitía navegar a nivel de Sección por ellos para leer y emitir opiniones (likes, dislikes, comentarios, etc.).

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

HIPOTESIS DE PERSONAS pag 24

El siguiente paso sería estudiar la viabilidad de esta aplicación entrevistando gente para realizar una investigación sobre sus necesidades prácticas. Por otra parte también sería útil enseñarles los prototipos básicos que manejábamos para verificarlos. Entendimos que el tipo de persona con el que nos entrevistáramos debía de estar relacionado de alguna manera con el mundo de la política, pues nada mejor que hablar con gente ya interesada en dichos temas para orientarnos por el buen camino.

A continuación detallamos las entrevistas que realizamos en la investigación:

Entrevista con Labodemo

Tuvimos la oportunidad de mantener una conversación con dos miembros de Labodemo [70], en la que aprovechamos para mostrarles un prototipo de la aplicación que estábamos desarrollando. Ambos tenían experiencia en el desarrollo de plataformas de participación ciudadana en Internet: fueron los responsables del desarrollo de los portales de participación del partido político Podemos y la candidatura ciudadana de unidad popular Ahora Madrid.

En ese momento nuestro prototipo móvil se limitaba únicamente a mostrar las diferentes secciones de cada programa, lo cual les pareció útil, aunque no lo suficiente como para atraer a una cantidad considerable de usuarios. Conforme a su línea de trabajo habitual, eran más partidarios de dar a los usuarios la posibilidad de realizar Propuestas además de ver Programas políticos. Les comentamos entonces que antes de hablar con ellos ya habíamos planteado desarrollar Propuestas colaborativas en tiempo real aprovechando la tecnología de Wave. Pero ellos no eran partidarios de esta opción, ya que según ellos acabaría siendo caótico tener tanta gente editando la misma propuesta de cara a generar contenido útil.

Dándole una vuelta a la categorización del contenido, nos sugirieron que para atraer a usuarios, debíamos considerar la posibilidad de integrar en la aplicación a colectivos sociales que generaran y categorizaran dicho contenido. No eran partidarios de que dieramos nosotros ciertas temáticas preestablecidas, pues debido a la variedad de redacción en los distintos programas sería difícil identificar temáticas que incluyeran a todos los programas y probablemente acabaríamos excluyendo temas. Así, serían los colectivos los que se encargaran de "tematizar.^{el} contenido de la aplicación. Por ejemplo: un grupo de animalistas podría tener un espacio en la aplicación donde poder crear sus propias propuestas, e incluso hacer comparativas de lo que proponen los diferentes programas sobre los animales.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

De esta manera, y en relación a la aproximación a los foros tradicionales, se planteó la idea de crear "Hilos": elementos "temáticos" que agruparan en un solo sitio Secciones y Propuestas que hablaran sobre un determinado tema. Estos hilos serían también generados por dichos colectivos.

Esto sería útil también para usuarios que buscaran información sobre un determinado tema. Por ejemplo: un usuario poco activo, que resulta ser profesor, podría buscar un colectivo de profesores y ver las Propuestas que se llevan a cabo o visualizar una comparativa respecto las medidas de educación de los diferentes programas políticos.

Nos insistieron mucho en el tema de las comparativas. Sería de gran utilidad que la aplicación tuviera una parte de comparativas en la que los usuarios pudieran comparar los programas políticos en vez de leerlos sección por sección. Resultaría de gran interés a un autónomo visualizar las medidas que proponen los diferentes partidos políticos para los autónomos. Pero estas comparativas no podrían realizarlas cualquiera, por lo que deberían realizarlas periodistas o expertos que hubieran realizado algún tipo de comparativa similar anteriormente. Nos sugirieron contactar con periodistas o colectivos que hubieran publicado algún tipo de comparativa en cuanto a programas o medidas, para obtener algún tipo de ayuda o consejo a seguir.

Conclusión

¿QUE OPINAN Y COMO CAMBIAN LOS PROTOTIPOS?

La reunión con dos de los miembros de Labodemo resultó de gran interés. Se trataba de personas que tenían mucha experiencia en desarrollo de portales de participación ciudadana y que sobre todo estaban muy familiarizados con el uso común que le suele dar la gente a este tipo de aplicaciones. A priori, sabían mejor que nosotros cómo determinar las claves para que una aplicación tuviera una aceptación y uso considerable de los usuarios desde el principio.

Visualizar los programas políticos en la app no les entusiasmó demasiado. Expusieron que, a priori, un ciudadano de a pie no iba a molestarse en meterse en la aplicación para leer los programas políticos de los diversos partidos. Ellos argumentaban que los colectivos sociales serían los usuarios más activos en nuestra aplicación, por lo que debíamos enfocar más el desarrollo hacia la participación ciudadana y la generación de Propuestas o Comparativas de Secciones de Programas hechas por tales colectivos. Dichas Comparativas sí que generarían más interés por leer los programas políticos.

Tras la reunión tomamos nota de los consejos que obtuvimos de Labodemo.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

Incluimos la idea de Comparativasz empezamos a pensar la forma de colaborar de grupos y colectivos. También intentamos contactar con colectivos y personas interesadas en el desarrollo de la aplicación.

Finalmente, conseguimos concretar una entrevista con Javier de la Cueva [71].

Entrevista con Javier de la Cueva

La entrevista con Javier de la Cueva resultó bastante productiva. Ya le conocíamos de algunas conferencias que impartió en la facultad. Javier es abogado y doctorado en Filosofía, estando especializado en temas relacionados con tecnología, Internet y propiedad intelectual. Además, en sus últimas conferencias Javier habla sobre acciones micropolíticas [72]. Estas acciones definen la capacidad que tienen los ciudadanos para realizar aportaciones a la sociedad, el estado o el gobierno que favorezcan la participación ciudadana en una democracia participativa.

Representar los programas electorales en una aplicación móvil le pareció algo interesante y necesario para la sociedad actual. Si bien casi nadie hace el esfuerzo de visualizar un programa electoral en PDF, utilizar una herramienta que facilita el acceso al programa por secciones podría ser una nueva forma de incentivar su lectura y ayudar a fomentar la participación ciudadana en política.

Además nos sugirió la posibilidad de desarrollar una Hemeroteca de programas electorales. De esta forma cualquiera podría consultar los programas de los anteriores gobiernos y comprobar si se cumplieron los objetivos del programa, así como comparar programas de distintos años entre sí,

Pero si en algo nos insistió Javier, fue en la importancia de categorizar el contenido de la aplicación. Un usuario que no tenga conocimientos sobre diversos temas, se encontraría más cómodo si pudiera visualizar las diferentes partes de un programa o las propuestas ciudadanas por categorías o temas generales. Ya que dejar libertad a los usuarios para crear categorías personalizadas podría ser algo negativo para usuarios inexpertos o con pocos conocimientos sobre temas específicos.

Por último, centrándonos en las Propuestas ciudadanas, surgió la idea de elaborar propuestas que tuvieran una especificación concreta. Es decir, a parte de tener una idea de propuesta y redactarla, esta propuesta debería ir acompañada de los recursos que serían necesarios y sobre todo cómo se llevaría a cabo de una forma aproximada. También resultaría interesante

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

definir un pequeño presupuesto de lo que conllevaría realizar la propuesta o cómo se podría financiar. Así evitaríamos una elaboración de propuestas más real, evitando un listado de propuestas infinito sin planterase cómo se llevarían a cabo o cómo se financiarían.

Conclusión

¿QUE OPINAN Y COMO CAMBIAN LOS PROTOTIPOS?

Javier trató de abrirnos un poco más la mente, para plantearnos ideas que no están representadas en la sociedad. En la actualidad encontramos diversas plataformas de participación ciudadana que acaban convirtiéndose en un foro. Para evitar caer en esto, debíamos innovar algo más para volver a repetir esto, debíamos de intentar desarrollar nuevas formas de participación.

Clasificar nosotros las secciones de un programa así como también las propuestas ciudadanas en diferentes categorías y temas era algo que descartamos en un principio tras la reunión con Labodemo. Aunque para dar cierta libertad, podríamos incluir la posibilidad de crear categorías personalizadas o ir añadiendo categorías semanalmente.

5.3.2. Modelado de Personas

*¿QUE SON LAS PERSONAS? -*REPRESENTACION GRAFICA**

En esta fase definiremos el tipo de persona que interactuará con nuestra aplicación. Para ello hemos identificado dos tipos de personas primarias:

- Activista social, 16 años en adelante

Actividad:

- Estudia, trabaja o realiza otras actividades de voluntariado.
- Frecuenta asambleas, participa en diferentes movimientos sociales y está al día de la actualidad política.
- Utiliza redes sociales para comunicarse con otros colectivos, acudir a asambleas, promover ideas u otras actividades relacionadas con la política y el activismo social.

Otros:

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

- Desconoce las ideas que proponen algunos partidos
- Le gusta aportar nuevas soluciones a la sociedad.

- Ciudadano de a pie, 18 años en adelante

Actividad:

- Estudia, trabaja o realiza otras actividades de voluntariado.
- Es distante al mundo de la política, concibe ciertos temas pero no los conoce en profundidad.
- Visita diferentes medios de comunicación para enterarse de la actualidad.
- Utiliza redes sociales para compartir contenidos con sus amigos o establecer nuevas amistades.

Otros:

- Desconoce por completo los programas electorales.
- Tiene cierta indecisión a la hora de acudir a las urnas, no sabe qué propone cada partido.

5.3.3. Definición de Escenarios y Requisitos

En esta sección se definirán los posibles escenarios que puedan surgir en la aplicación:

Escenario I

Se acercan las elecciones municipales y Juan aún no ha decidido a qué partido va dar su voto. No conoce las propuestas que ofertan los partidos a la ciudadanía y tampoco se fía mucho de lo que dicen los medios de comunicación.

Juan coge su móvil y visualiza los diferentes programas electorales por categoría, seleccionando la categoría de educación que es la que más le afecta a él personalmente. La aplicación le muestra un listado de las secciones donde los diferentes partidos hablan de las medidas que van a tomar en torno a la educación.

Requisitos:

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

1. Visualizar (acción) las diferentes categorías (objeto), para ver las secciones de los programas de una determinada categoría (contexto).
2. Mostrar(acción) un listado de todas las secciones de los programas de los partidos políticos (objeto) en función de la categoría seleccionada por el usuario (contexto).

Escenario II

Pablo es un empleado sanitario de Hospital Clínico de Madrid preocupado por la gestión de los hospitales públicos. Parece que la situación no está muy controlada, por lo que quisiera saber que propuestas o alternativas propone la ciudadanía para mejorar la situación actual.

A través de su móvil puede explorar las diferentes propuestas por categorías. Eligiendo la categoría de sanidad, le aparece un listado de las últimas propuestas desarrolladas por la ciudadanía.

Requisitos:

1. Mostrar (acción) las diferentes categorías (objeto), para visualizar las propuestas ordenadas por la categoría seleccionada (contexto).
2. Visualizar (acción) el listado de propuestas (objeto), clasificados por la categoría seleccionada por el usuario (contexto).

Escenario III

Requisitos:

- 1.
- 2.
- 3.

Escenario IV

Requisitos:

- 1.
- 2.
- 3.

5.3.4. Framework de diseño

En esta sección detallaremos todos los aspectos relacionados con el desarrollo del aspecto visual y la interacción con la aplicación. Detallaremos los prototipos desarrollados en papel, algunos prototipos intermedios y el prototipo final de la aplicación presentado.

Metodología

En una primera fase realizamos una serie de prototipos rápidos a papel para visualizar una interacción con las principales características de la aplicación. A continuación llevamos estas ideas en papel a nuestra aplicación desarrollada en Android. Obteniendo un prototipo de media-alta fidelidad sobre la funcionalidad de la aplicación. Este fue el prototipo que enseñamos en las entrevistas durante la fase de investigación.

Cuando quisimos añadir nuevas características tras sacar conclusiones de las entrevistas, volvimos a desarrollar nuevos prototipos en papel de las pantallas a desarrollar. Por último, se desarrollaron las nuevas pantallas y se refinaron las anteriores para llegar a un prototipo de alta fidelidad o una versión alpha de la aplicación.

ELEMENTOS DE DATOS Y FUNCIONALES: DSI tema 3 pag 54

Prototipos en papel

Meter las fotos de los prototipos en papel y comentarlos.

Prototipos intermedios

La aplicación pretende llevar las principales partes de los programas electorales de los partidos que se presenten a las elecciones. Por tanto, cualquier usuario podrá visualizar el apartado que desee consultar de cualquier partido político. Siendo esta la forma menos amigable de leerlo, se utilizarán distintas formas para compartir o divulgar determinadas secciones más populares.

Al inicio de la aplicación, mostrará una lista de las secciones de los programas más valoradas, más debatidas, peor valoradas e incluso las más incomprendidas. Por tanto creemos que puede ser una forma de acercar aquellas secciones más populares de forma más eficaz, al contrario que tener que consultar una determinada página dentro de un extenso pdf.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO



Figura 5.8: Vista principal de secciones

Dentro de cada sección podemos visualizar el contenido de la sección a la que referencia el programa, y tendremos la opción de valorarla de forma positiva o negativa. También añadimos la posibilidad de indicar que no se ha entendido la sección. Pues a la hora de leer una propuesta de gobierno ubicada en una sección del programa, bien nos puede gustar, disgustar o simplemente no haber entendido la idea y por ello no votarla de forma positiva o negativa.

CAPÍTULO 5. METODOLOGÍA DEL PROYECTO

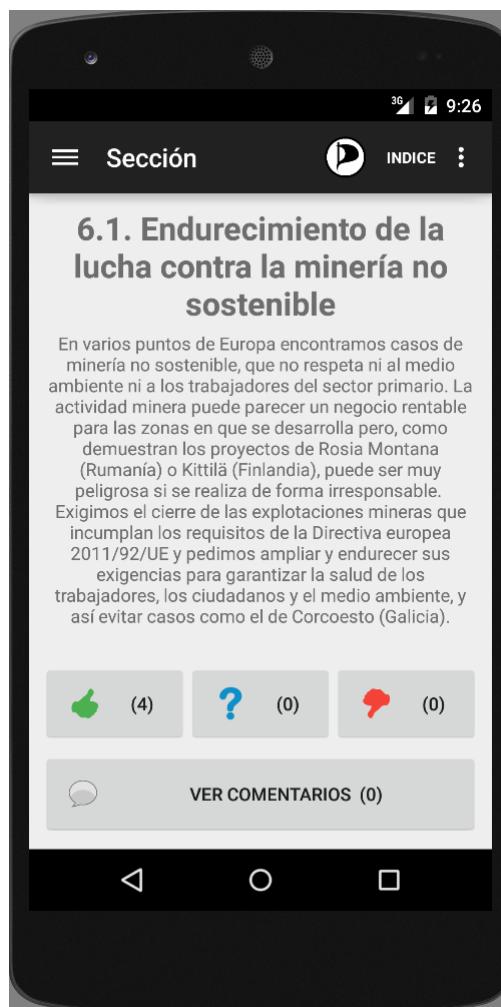


Figura 5.9: Visualizando una sección

Sin olvidarnos de la parte social, en cada sección podemos hacer comentarios para intentar debatir las ideas fundamentales que propone la sección. O incluso hacer referencia a una determinada frase o párrafo.

Prototipo final

5.3.5. Análisis de Usabilidad

Hablar de las reglas de oro, principios de diseño,...

HEURISTICA NIELSEN

5.4. Implementación de DemoCritics

METODOLOGIA GITHUB, CONTROL DE VERSIONES, GIT, REPAR-
TO DE TAREAS: ANDROID - BACKEND, DEBUGGER consola-a-dispositivo,
REVISIONES DE CODIGO, REVISION DE USABILIDAD.

5.5. Evaluación con Usuarios

Capítulo 6

Arquitectura del Proyecto

6.1. Arquitectura de Wave

6.1.1. Modelo Wave

Además de definir el protocolo del que hace uso Wave, Google definió un Modelo de Datos Conversacional [73] que refleja la arquitectura de los datos que componen las conversaciones en Wave. Así, a grandes rasgos, podemos ver dichas conversaciones como documentos XML sobre los que los usuarios participantes (cualquiera es libre de unirse a una conversación en cualquier momento) actúan creando nuevos elementos o modificando los ya existentes. Este modelo de datos define una nomenclatura propia para los elementos que componen esta tecnología [74] [15]:

- **Wave:** Conjunto de wavelets (conversaciones).
- **Wavelet:** conjunto de documentos de una conversación y sus participantes.
- **Blip:** documento con el contenido de un mensaje en la conversación. Un blip puede tener otros blips dentro de él y los blips pueden ser publicados o no en función de si su visibilidad se extiende o no al resto de participantes de la conversación respectivamente.
- **Manifiesto conversacional:** documento con metadatos que definen la estructura de una conversación.
- **Hilo conversacional:** conjunto de Blips consecutivos que forman parte de una conversación.
- **Extensiones** [75]: pequeñas aplicaciones que se ejecutan dentro de una Wave y aportan nuevas funcionalidades que no forman parte del modelo conversacional básico. Pueden ser de dos tipos:

- **Gadget:** aplicación que se ejecuta en el contexto de una Wave y en la que todos sus usuarios participan.
- **Robot:** aplicación que participa en una Wave a modo de usuario automatizado e interactúa con el contenido pudiendo modificarlo y responder a eventos por acciones de otros usuarios reales.

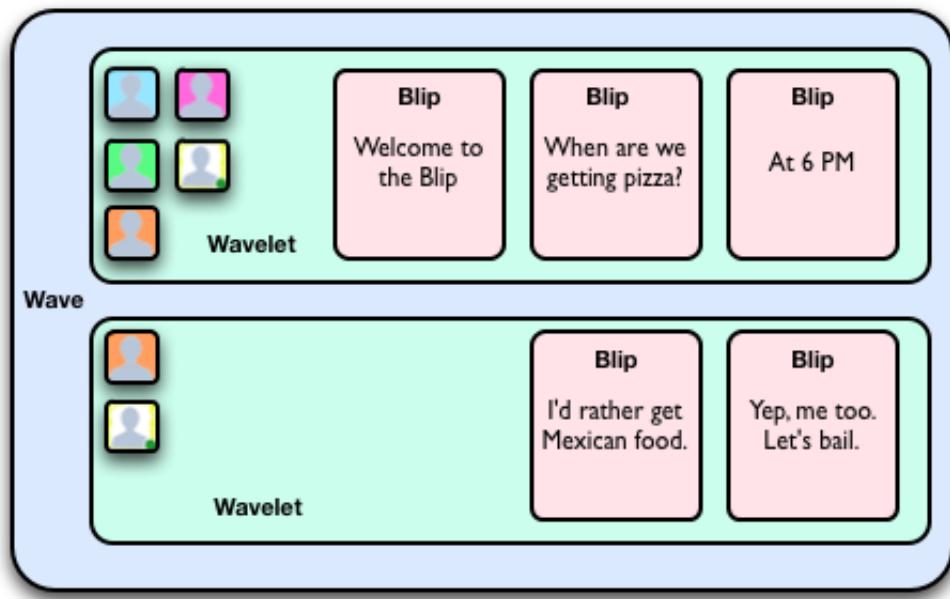


Figura 6.1: Modelo Conversacional de Wave

6.1.2. Modelo SwellRT

6.2. Arquitectura de la aplicación

La arquitectura está compuesta por cuatro módulos principales de los que hablaremos en profundidad en las siguientes subsecciones. Como **Cliente móvil** tendremos la aplicación desarrollada en Android, que realizará peticiones HTTP al servicio web alojado en OpenShift [30], una plataforma que permite alojar servicios web de forma gratuita. Dentro del servicio contaremos con una **API RESTful** que será quien gestione las peticiones de la aplicación móvil mediante el protocolo HTTP.

CAMBIAR FOTO PARA VER INTEGRACION CON WAVE

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

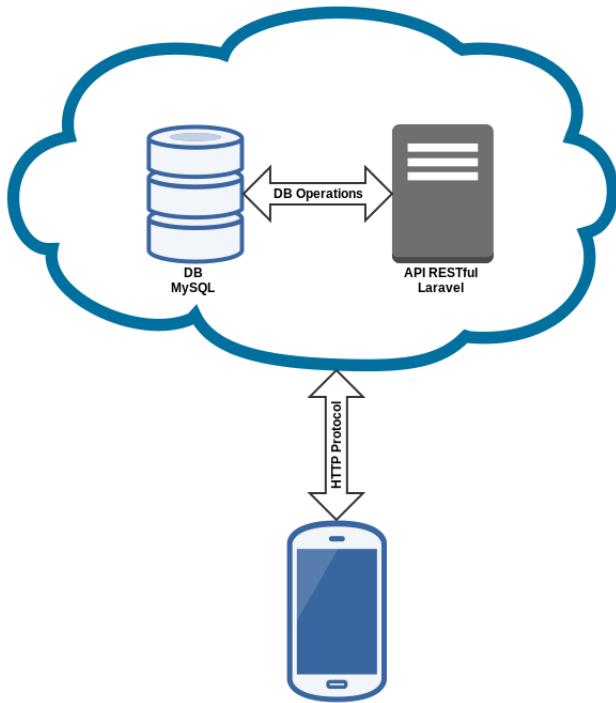


Figura 6.2: Arquitectura de la aplicación.

Por otro lado, la aplicación hará uso del **Servicio Android** desarrollado en SwellRT-Android [68] para conectarse con el servidor Wave alojado en <https://wave.p2pvalue.eu/> e intercambiar los datos que sea necesario mantener en tiempo real, es decir, la edición de una Propuesta de forma colaborativa entre varias personas.

Por último, la **Base de Datos MySQL** alojada en el servidor de OpenShift almacenará toda la información relacionada con la aplicación. La API RESTful será quien actúe de intermediario entre las peticiones de la aplicación y las operaciones en Base de Datos.

6.2.1. Base de datos

En la implementación de la Base de Datos se ha utilizado un Modelo Relacional para la definición de las tablas. Utilizando MySQL [76] como sistema de gestión de base de datos (SGBD) y phpMyAdmin [77] como herramienta de gestión gráfica de la base de datos.

La base de datos está formada por un total de 12 tablas donde se almacena

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

toda la información relacionada con los programas de los partidos políticos, las propuestas ciudadanas, encuestas, comparativas... y otros datos más técnicos como la gestión de los usuarios, la relación de los comentarios o la relación entre las secciones y comparativas entre otras.

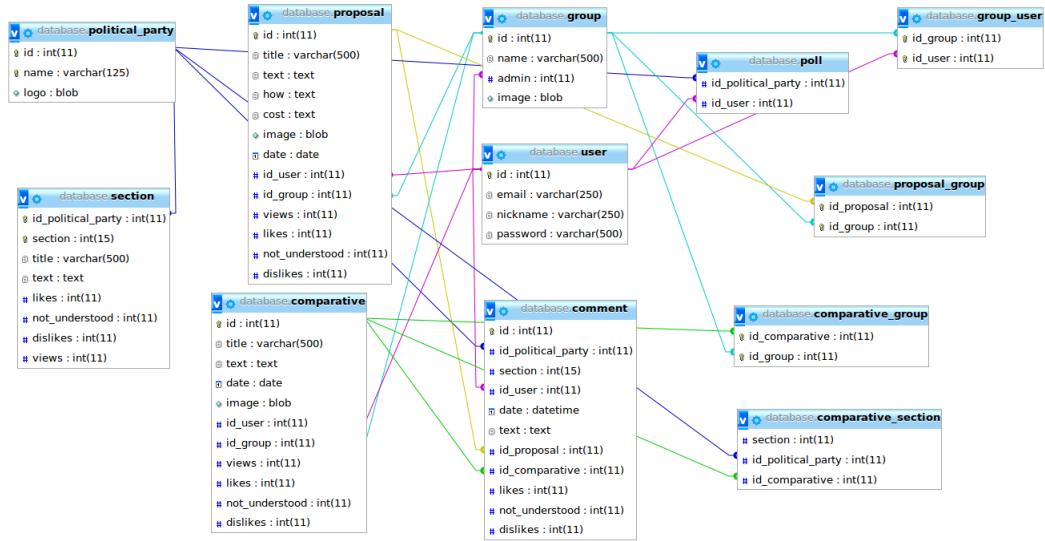


Figura 6.3: Modelo entidad-relación de la base de datos.

Las tablas *section* y *political-party* son utilizadas para guardar información estática en la aplicación. Es decir, en la tabla *political-party* se almacenan los partidos políticos que se presentan a unas elecciones, y en la tabla *section*, las diferentes secciones de un programa electoral. Tan sólo modificaremos las columnas de *likes*, *dislikes*, *not-understood* y *views* para obtener estadísticas de uso de cada sección. El resto de las columnas permanecerán intactas.

EXPLICAR RESTO DE TABLAS. DATOS ESTATICOS VS DINAMICOS

Las demás tablas serán utilizadas para guardar datos dinámicos de la aplicación. Datos que normalmente genera un usuario visitando una sección de un programa, creando una propuesta, haciendo un comentario, etc.

6.2.2. Service REST

Para establecer la conexión de la aplicación desarrollada en Android con la Base de Datos hemos utilizado **Laravel** como Servicio Web. Laravel [78]

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

es un framework de código abierto para desarrollar aplicaciones web con PHP 5. Laravel permite además montar una API REST (Representational State Transfer), un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. Este término se originó en una tesis doctoral sobre la web escrita por Roy Fielding [79]. La elección de Laravel como framework PHP fue debida a su flexibilidad, la facilidad para programar la aplicación y el gran soporte que tiene de la comunidad. Además Laravel cuenta con una licencia de software libre MIT, lo que nos permite continuar utilizando software en todo el proyecto, y fue el Framework más popular al finales de 2013.

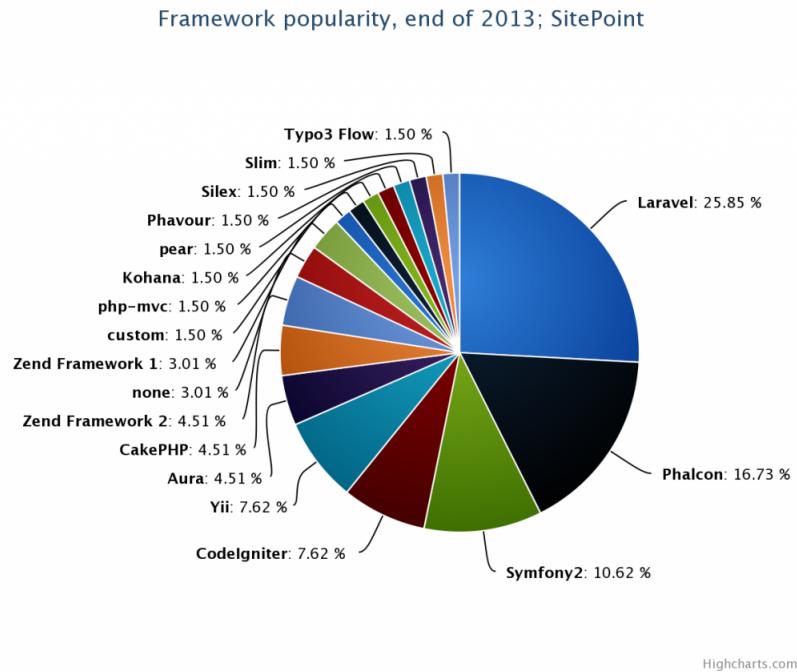


Figura 6.4: Pantalla principal de Laravel 5.

Laravel nos permite implementar un sistema RESTful para que el cliente móvil pueda hacer peticiones al servicio web y que dicho servicio responda a éstas de la forma que queramos. Estas peticiones se realizan mediante el protocolo HTTP. En función de la operación que deseemos hacer, clasificaremos estas funciones en el acrónimo **CRUD** [80] (del original en inglés: **C**reate, **R**ead, **U**pdate and **D**elete).

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

Petición	Operación	SQL	Utilidad
GET	Leer	SELECT	Obtener un recurso almacenado en el servidor.
POST	Crear	INSERT	Crear un nuevo recurso en el servidor.
PUT	Actualizar	UPDATE	Actualizar un recurso almacenado en el servidor.
DELETE	Borrar	DELETE	Eliminar un recurso almacenado en el servidor.

Cuadro 6.1: Funciones CRUD

En función de las peticiones que realicemos al servicio que serán realizadas mediante el protocolo HTTP, el servidor nos devolverá un código de estado para obtener el *feedback* de lo sucedido. Por tanto distinguiremos diferentes códigos de estado cuando queramos obtener un recurso, actualizar un recurso, crear uno nuevo o borrarlo. En la siguiente tabla se definen los códigos de estado que puede devolvernos el servidor en función de nuestras peticiones.

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

Petición	Status Code	Descripción
GET	200 (OK)	El recurso solicitado ha sido devuelto correctamente.
GET	404 (Not Found)	El recurso solicitado no ha sido encontrado.
POST	201 (Created)	El nuevo recurso ha sido creado correctamente.
POST	404 (Not Found)	No se ha especificado el nuevo recurso a crear.
POST	409 (Conflict)	No se ha podido crear el recurso porque ya existe.
PUT	200 (OK)	El recurso ha sido actualizado correctamente.
PUT	204 (No Content)	No se ha especificado el recurso que pretende ser actualizado.
PUT	404 (Not Found)	El recurso a actualizar no ha sido encontrado.
DELETE	200 (OK)	El recurso solicitado ha sido borrado.
DELETE	404 (Not Found)	El recurso solicitado para borrar, no ha sido encontrado.

Cuadro 6.2: Códigos de estado de la respuesta del servidor

Usar un servicio RESTful nos proporciona una gran flexibilidad a la hora de independizar la tecnología del servidor de la del cliente. Mediante la arquitectura basada en peticiones HTTP no solo podremos hacer peticiones desde el cliente en Android, sino que más adelante podríamos desarrollar una versión web o incluso un cliente para iOS sin tener que modificar el servicio, ya que las peticiones HTTP serán las mismas.

Arquitectura

Laravel utiliza implementa un funcionamiento basado en el patrón **MVC** (Model–view–controller), separando el modelo de la vista, y delegando la gestión al controlador. En nuestro caso tendríamos un modelo almacenado en las tablas de la base datos, donde se guardará toda la información dinámica y estática de la aplicación. El controlador sería encargado de gestionar las peticiones del cliente en Android en función del tipo de operación que requiera. Y por último, tendríamos como vista el resultado devuelto por el servidor con los datos solicitados por el cliente, que a su vez los representaría

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

en la aplicación móvil. Esta estructura de comportamiento hace independiente el modelo controlado por el servidor, de la vista que obtiene el usuario final en su cliente móvil. Con esto conseguimos elaborar un código más claro y sencillo, así como también evitar conflictos entre el modelo y la vista.

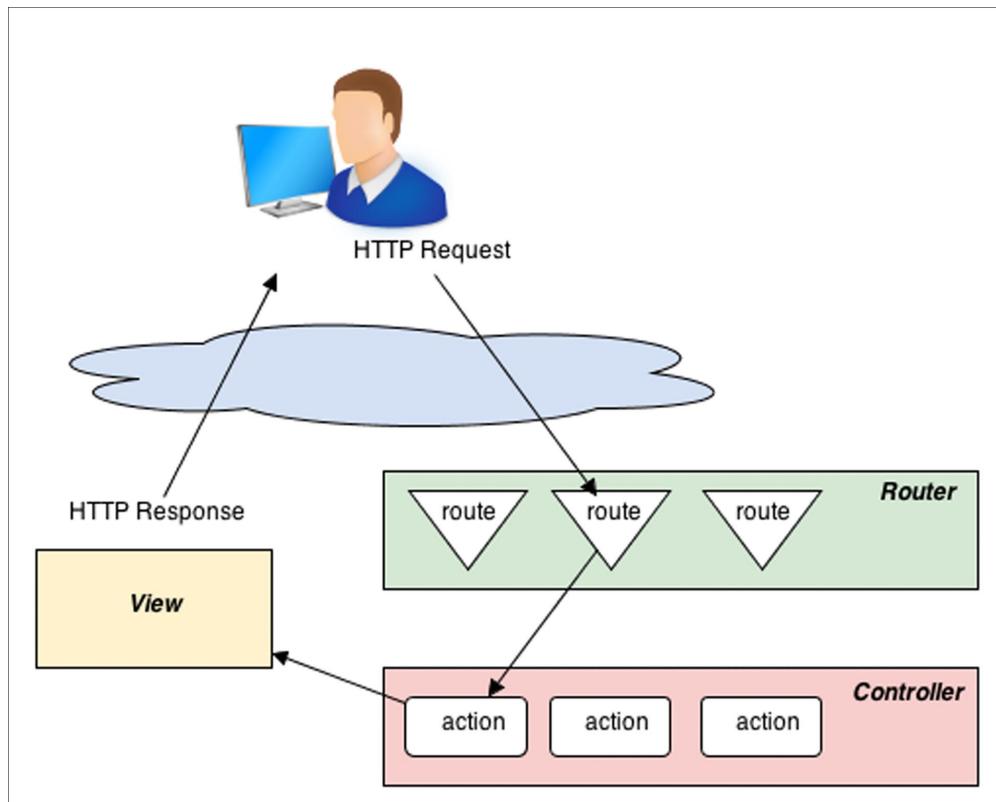


Figura 6.5: Arquitectura de Laravel

Para comprender la arquitectura de Laravel vamos a destacar tres componentes clave. En primer lugar tenemos una primera capa que gestiona las peticiones del cliente. Esta capa estará compuesta por una serie de **rutas** a las que el cliente realizará la petición en función de la operación que quiera realizar. Después, en función de la ruta que haya realizado la petición, se adentrará en una nueva capa compuesta por **controladores** que se encargarán de procesar la petición y devolver al cliente lo que ha solicitado. Éste último componente será la **vista**, que puede ser representada de muchas formas. En nuestro caso, la vista estará compuesta en una serie de datos en formato JSON, que posteriormente el cliente móvil se encargará de representar la vista. Visto esto, procederemos a detallar un poco más estos tres componentes fundamentales.

Rutas

Cómo citábamos en anteriormente, la primera capa de Laravel es un *mapa de rutas* que definirá todos los caminos posibles para llegar a la funcionalidad requerida de los controladores. Todas las rutas estarán definidas en el fichero **routes.php**, alojado en la carpeta *app/Http/routes.php* de nuestra instalación Laravel. El archivo *routes.php* es un fichero muy simple escrito en PHP donde definiremos todas las rutas posibles. La definición de una ruta irá acompañada de tres valores. El primero de ellos será el tipo de petición que realizará el cliente (ver tabla 6.1), seguido de la definición de la url por la que será accesible, y por último, indicaremos el controlador responsable de procesar la petición. A parte de indicar el controlador, también hay que indicar el método que procesará la petición dentro del controlador como veremos más adelante.

Para definir el conjunto de urls que forman el archivo *routes.php*, hemos seguido una serie de buenas prácticas [81] habituales a la hora de desarrollar una RESTful API. Normalmente utilizaremos nombres en lugar de verbos para acceder a los recursos. Por ejemplo, si queremos visualizar el listado de propuestas de la aplicación, utilizaremos la url */proposal*. Que será definida como:

```
Route::get('/proposal', 'ProposalController@index');
```

Así queda definida la ruta que accede a todo el listado de propuestas como una petición GET, que se encargará de procesarla el controlador *ProposalController* en su método *index()*.

Para acceder a una propuesta en concreto, se pasará un *id* para obtener el recurso adecuado. Este *id* irá a continuación de la url anteriormente definida. Por ejemplo, si quisiéramos acceder a la propuesta cuyo id es igual a 5, definiríamos la siguiente ruta:

```
Route::get('/proposal/{id}', 'ProposalController@show');
```

Nótese ruta es la misma que la anterior, a la que hemos añadido un nuevo parámetro que deberá especificar el usuario. Así la petición *GET /proposal/5*, nos devolvería la propuesta con id 5. Si nos fijamos, el controlador también es el mismo, pero esta vez el método encargado de procesar la petición será *show()*. Además este método recibirá el parámetro *id* que envíe el usuario en la petición.

Para crear una nueva propuesta en la aplicación, deberemos realizar una petición POST. Nuestra ruta será exactamente igual que la primera, pero

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

antes deberemos especificar que se trata de una ruta que atenderá a una petición POST:

```
Route::post('/proposal', 'ProposalController@create');
```

Donde una vez más en controlador *ProposalController* se encargará de procesar la petición en el método *create*. Si nos fijamos en la primera ruta dónde obteníamos las propuestas (*GET /proposal*, es exactamente igual que la que acabamos de definir. Sólo les diferencia el tipo de petición que está realizando el cliente. Para definir otros tipos de peticiones como PUT o DELETE, usaremos la misma metodología. El resultado final de las peticiones que podrían realizarse a una propuesta quedaría definida de la siguiente forma:

```
// Posibles operaciones con el objeto "propuesta"
Route::get('/proposal', 'ProposalController@index');
    // Obtiene el listado de todas las propuestas
Route::get('/proposal/{id}', 'ProposalController@show');
    // Obtiene la propuesta pasada por el campo {id}
Route::post('/proposal', 'ProposalController@create');
    // Crea una nueva propuesta en el servidor
Route::put('/proposal/{id}', 'ProposalController@update');
    // Actualiza la propuesta pasada por {id}
Route::delete('/proposal/{id}',
            'ProposalController@destroy');
    // Borra la propuesta pasada por {id}
```

Para organizar el código y poder visualizarlo de forma clara utilizaremos grupos de rutas. Esto nos resultará especialmente útil cuando tengamos varias operaciones que realizar dentro de una misma dirección, definiendo una nueva ruta cuya función será definida a continuación, incluyendo las rutas que vienen dentro del grupo. Si nos fijamos en el ejemplo anterior, podemos observar como todas las url comienzan con */proposal*. Por ello no es necesario definir en cada línea la ruta */proposal*, si no que bastará con definirla en un único grupo que adjuntará todas las rutas que comienzen de la misma forma. Así nuestro ejemplo anterior quedaría más claro y ordenado de la siguiente forma:

```
Route::group(['prefix' => 'proposal'], function()
{
    Route::get('/', 'ProposalController@index');
    Route::get('/{id}', 'ProposalController@show');
    Route::post('/', 'ProposalController@create');
    Route::put('/{id}', 'ProposalController@update');
    Route::delete('/{id}', 'ProposalController@destroy');
});
```

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

De esta forma podemos visualizar de un vistazo todas las posibilidades agrupadas dentro de la dirección *proposal*. Lo que nos permitirá utilizar grupos para ordenar funciones comunes dentro de un mismo prefijo, y además crear subgrupos dentro de los anteriores si tenemos algún prefijo repetido varias veces. Esto podría aplicarse a nuestro ejemplo anterior, agrupando las peticiones que requieren como parámetro un *id* de la propuesta:

```
Route::group(['prefix' => 'proposal'], function()
{
    Route::get('/', 'ProposalController@index');
    Route::post('/', 'ProposalController@create');
    Route::group(['prefix' => '/{id}'], function()
    {
        Route::get('/', 'ProposalController@show');
        Route::put('/', 'ProposalController@update');
        Route::delete('/', 'ProposalController@destroy');
    });
});
```

HACE FALTA ALGO MÁS...?

Controladores

Los controladores son los encargados de procesar todas las operaciones que intervienen en el modelo, es decir, en la información almacenada en la base de datos. Un controlador es un fichero escrito en PHP que será almacenado en la ruta */app/Http/Controllers* de la aplicación. Este fichero estará formado por una métodos y atributos que serán llamados desde las rutas definidas en la aplicación. Cada controlador hereda de una clase abstracta llamada *Controller*, que a su vez esta hereda de la clase abstracta *BaseController*. La cual implementa la funcionalidad básica de los controladores.

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

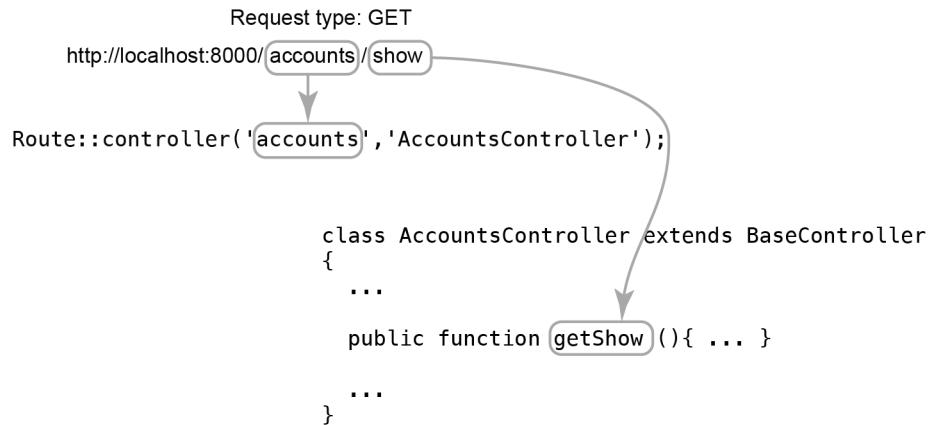


Figura 6.6: Camino desde la ruta al método del controlador.

Dentro de un controlador definiremos aquellos métodos que correspondan con las rutas de la aplicación. Estos métodos procesarán la información del cliente, operando con los parámetros que haya obtenido, accediendo a la base de datos, y devolviendo una respuesta en función del éxito de la operación. El siguiente ejemplo muestra una implementación básica del método *index()* del *ProposalController*:

```
<?php namespace App\Http\Controllers;  
  
use App\Http\Requests;  
use App\Http\Controllers\Controller;  
use DB;  
  
class ProposalController extends Controller {  
    /**  
     * Display a listing of the resource.  
     *  
     * @return Response  
     */  
    public function index()  
    {  
        return DB::select('SELECT * FROM `proposal`');  
    }  
}
```

La conexión a la base de datos se encuentra configurada en el archivo */config/database.php*, por lo que la gestión de la conexión y desconexión será controlada por Laravel. Lo que nos proporciona más independencia y versatilidad a la hora de programar. Como respuesta del método, Laravel utiliza por de-

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

fecto un fichero JSON con los resultados de la variable a la que asignemos el resultado, o a la sentencia SQL de la consulta que estemos devolviendo. El siguiente ejemplo muestra cuál sería el resultado de realizar una petición GET a la dirección */proposal*:

```
[{"id":1,"title":"Reducir la contaminaci\u00f3n en Madrid","text":"Actualmente la contaminaci\u00f3n en Madrid est\u00f3 llegando a unos 1\u00edl\u00f3metros por encima de la media de las principales ciudades de la Uni\u00f3n Europea. Por ello deber\u00e1mos reducir esa contaminaci\u00f3n ambiental acerc\u00e1ndonos a la media europea.", "how":"Cerrando el tr\u00f3fico en determinadas zonas de Madrid. Distrito: Zona Centro.", "cost":"Hacer 7 km de calles peatonales: 750.000 \u20ac", "id_image":4, "date":"2015-05-06 00:00:00", "id_category":4, "id_user":"6d823fa54c6d1a12", "views":4, "likes":3, "not_understood":0, "dislikes":0}, {"id":2,"title":"Restaurar el plan de estudios del '98", "text": "\u2022Lorem ipsum dolor sit amet, \u2022consectetur adipiscing elit. \u2022Praesent lobortis, sapien eget dignissim efficitur, augue eros molestie purus, et u gravida erat quam u lacus. \u2022Fusce vel eros cursus, u venenatis u dui u ac, aliquet u risus. \u2022Fusce euismod ex u at u varius u lacinia. \u2022Sed u at u urna u condimentum u orci u volutpat u eleifend u nec u id u erat. \u2022Sed u at u vestibulum u neque. \u2022Maecenas u vehicula u hendrerit u felis, quis u condimentum u arcu u commodou eu. \u2022Sed u egestas, u orci u eget u ultricies u mollis, u est u leo u porttitor u sem, u at u elementum u enim u arcu u ut u magna. \u2022", "how":"Suprimiendo el plan Bolonia de los grados de 4 a 10 licenciaturas y diplomaturas.", "cost":"Coste 0.", "id_image":2, "date":"2015-05-18 00:00:00", "id_category":2, "id_user":"d2d115f79ab3a7a4", "views":12, "likes":0, "not_understood":1, "dislikes":0}], ]
```

6.2.3. Servicio Android de conexión con Wave

REORDENAR LO DE WAVE

6.2.4. Cliente Android

INTERFAZ GRAFICA ->PRESENTACION AL USUARIO
ALGORITMO ESTRUCTURACION DE PROGRAMA

CAPÍTULO 6. ARQUITECTURA DEL PROYECTO

ARQUITECTURA DE PETICIONES AL REST

ARQUITECTURA DE PETICIONES A WAVE

Capítulo 7

Resultados y Conclusiones

7.1. Discusion de Resultados

7.1.1. Evaluación con usuarios

7.2. Conclusiones

Capítulo 8

Trabajo a Futuro

8.1. Mejoras

Bibliografía

- [1] Plaza Podemos: ¡Sí se puede!
[https://www.reddit.com/.](https://www.reddit.com/)
- [2] Reddit: the front page of the internet.
[https://www.reddit.com/r/podemos/.](https://www.reddit.com/r/podemos/)
- [3] Change.org · La mayor plataforma de peticiones del mundo.
[https://www.change.org/.](https://www.change.org/)
- [4] Programa colaborativo de Zaragoza en Común.
[http://programa.ganemoszaragoza.com/.](http://programa.ganemoszaragoza.com/)
- [5] S.A. APPGREE. Appgree.
[http://www.appgree.com/que-es-appgree/.](http://www.appgree.com/que-es-appgree/)
- [6] S.A. APPGREE. DemoRank: El Algoritmo.
[http://www.appgree.com/el-proceso-paso-a-paso/.](http://www.appgree.com/el-proceso-paso-a-paso/)
- [7] Inc. Google. Meet Google Wave.
[http://googlecode.blogspot.com.es/2009/05/hello-world-meet-google-wave.html.](http://googlecode.blogspot.com.es/2009/05/hello-world-meet-google-wave.html)
- [8] Inc. Google. Google Wave Federation Protocol Over XMPP.
[http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html.](http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html)
- [9] Inc. Google. End of Google Wave.
[https://support.google.com/answer/1083134?hl=en.](https://support.google.com/answer/1083134?hl=en)
- [10] Google Docs.
[https://drive.google.com/.](https://drive.google.com/)
- [11] Google+.
[https://plus.google.com/.](https://plus.google.com/)
- [12] Apache. Apache Wave (Incubating).
[http://incubator.apache.org/wave/about.html.](http://incubator.apache.org/wave/about.html)

- [13] Apache. Apache License 2.0.
<http://www.apache.org/licenses/LICENSE-2.0>.
- [14] Inc. Google. Wave Federation.
<http://www.waveprotocol.org/federation>.
- [15] Google. Google Wave Federation Architecture White Paper.
<http://wave-protocol.googlecode.com/hg/whitepapers/google-wave-architecture/google-wave-architecture.html>.
- [16] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. 2011.
RFC 6120 Available at <http://tools.ietf.org/html/rfc6120>.
- [17] Understanding Operational Transformation.
<http://www.codemirror.com/blog/java/understanding-and-applying-operational-transformation>.
- [18] Inc. Google. Google Wave Operational Transformation.
<http://www.waveprotocol.org/whitepapers/operational-transform>.
- [19] Apache. Wave In A Box.
<http://www.waveprotocol.org/wave-in-a-box/>.
- [20] Oracle. OpenJDK.
<http://openjdk.java.net/>.
- [21] Google Inc. Google Web Toolkit.
<http://www.gwtproject.org/>.
- [22] Apache. Install WIAB.
<https://cwiki.apache.org/confluence/display/WAVE/Install+WIAB>.
- [23] Apache. WIAB Repository.
<https://github.com/apache/incubator-wave>.
- [24] WIAB Server Example.
<http://waveinabox.net/>.
- [25] P2PValue. P2P Value European Project.
<http://www.p2pvalue.eu/>.

- [26] P2PValue. SwellRT, a real-time federated collaboration framework.
<https://github.com/P2Pvalue/swellrt>.
- [27] Google Inc. Android Developers Site.
<http://developer.android.com/index.html>.
- [28] Google Inc. Android SDK.
<https://developer.android.com/sdk/index.html>.
- [29] Eclipse Foundation. Eclipse IDE.
<https://eclipse.org/ide/>.
- [30] Google Inc. Android Studio IDE.
<https://developer.android.com/tools/studio/index.html>.
- [31] JetBrains. IntelliJ IDEA IDE.
<https://www.jetbrains.com/idea/>.
- [32] Google Inc. Eclipse Android Development Tools (ADT).
<https://developer.android.com/tools/help/adt.html>.
- [33] Google Inc. Android API Reference.
<http://developer.android.com/reference/packages.html>.
- [34] Google Inc. Android Virtual Device Manager.
<http://developer.android.com/tools/devices/managing-avds.html>.
- [35] Google Inc. Using the Android Emulator.
<http://developer.android.com/tools/devices/emulator.html>.
- [36] Google Inc. Android SDK Manager.
<https://developer.android.com/tools/help/sdk-manager.html>.
- [37] Google Inc. Android Dalvik Debug Monitor Server.
<http://developer.android.com/tools/debugging/ddms.html>.
- [38] Google Inc. Android Versions Distribution.
<https://developer.android.com/about/dashboards/index.html>.
- [39] Google Inc. Building Apps with Over 65K Methods.
<https://developer.android.com/tools/building/multidex.html>.
- [40] Google Inc. ProGuard.
<http://developer.android.com/tools/help/proguard.html>.

- [41] Apache Foundation. Apache Ant.
<http://ant.apache.org/>.
- [42] Google Inc. Building and Running from the Command Line.
<https://developer.android.com/tools/building/building-cmdline-ant.html>.
- [43] Google Inc. Setting up a Device for Development.
<https://developer.android.com/tools/device.html#setting-up>.
- [44] Google Inc. Android Debug Bridge.
<https://developer.android.com/tools/help/adb.html>.
- [45] Google Inc. Activities.
<https://developer.android.com/guide/components/activities.html>.
- [46] Roy Fielding. Hypertext transfer protocol (http/1.1): Authentication. 2014.
RFC 7235 Available at <https://tools.ietf.org/html/rfc7235>.
- [47] Ian Fette and Alexey Melnikov. Hypertext transfer protocol (http/1.1): Authentication. 2011.
RFC 6455 Available at <https://tools.ietf.org/html/rfc6455>.
- [48] Google Inc. System Permissions.
<https://developer.android.com/guide/topics/security/permissions.html>.
- [49] Google Inc. App Manifest.
<https://developer.android.com/guide/topics/manifest/manifest-intro.html>.
- [50] Google Inc. Emulator Networking: Network Address Space.
<https://developer.android.com/tools/devices/emulator.html#emulatornetworking>.
- [51] Google Inc. Android LogCat.
<https://developer.android.com/tools/help/logcat.html>.
- [52] Google Inc. Apache HTTP Library.
<https://developer.android.com/reference/org/apache/http/package-summary.html>.

- [53] Google Inc. Android's HTTP Clients.
<http://android-developers.blogspot.com.es/2011/09/androids-http-clients.html>.
- [54] Google Inc. HttpURLConnection Library.
<https://developer.android.com/reference/java/net/HttpURLConnection.html>.
- [55] Google Inc. Processes and Threads.
<https://developer.android.com/guide/components/processes-and-threads.html>.
- [56] Google Inc. Connecting to the Network.
<https://developer.android.com/training/basics/network-ops/connecting.html>.
- [57] Async-IO.org. Atmosphere: The Asynchronous WebSocket/Comet Framework.
<https://github.com/Atmosphere/atmosphere>.
- [58] Async-IO.org. wAsync: A WebSockets/HTTP Client Library for Asynchronous Communication.
<https://github.com/Atmosphere/wasync>.
- [59] Wave Client for Android.
<https://github.com/Zorbel/swell-android>.
- [60] QOS.ch. SLF4J for Android.
<http://www.slf4j.org/android/>.
- [61] Google Inc. Android AsyncTask.
<https://developer.android.com/reference/android/os/AsyncTask.html>.
- [62] Google Inc. Android Services.
<https://developer.android.com/guide/components/services.html>.
- [63] Async-IO.org. Async Http Client.
<https://github.com/AsyncHttpClient/async-http-client>.
- [64] Project Grizzly. Project Grizzly: NIO Event Development Simplified.
<https://grizzly.java.net/index.html>.

- [65] P2PValue. SwellRT Client for Android.
<https://github.com/P2Pvalue/swellrt/tree/master/android>.
- [66] Alan Copper. *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity*. Sams Publishing, 2004.
- [67] Labodemo.
<http://labodemo.net/es/>.
- [68] Javier de la Cueva. Javier de la Cueva: abogado, doctor en Filosofía y estudiioso de las relaciones entre el Derecho y la Tecnología.
<http://www.javierdelacueva.es/bio/>.
- [69] Javier de la Cueva. *Manual del ciberactivista. Teoría y práctica de las acciones micropolíticas*. Bandaàparte Editores, 2015.
- [70] Inc. Google. Google Wave Conversation Model.
<https://wave-protocol.googlecode.com/hg/spec/conversation/convspec.html>.
- [71] Inc. Google. Google Wave API Overview.
<http://www.waveprotocol.org/wave-apis>.
- [72] Inc. Google. Google Wave Extensions.
<http://www.waveprotocol.org/wave-apis/extensions>.
- [73] OpenShift by Red Hat.
<https://www.openshift.com/>.
- [74] Oracle Corporation. MySQL :: The world's most popular open source database.
<https://www.mysql.com/>.
- [75] PhpMyAdmin: Bringing MySQL to the web.
<http://www.phpmyadmin.net/>.
- [76] Laravel - The PHP Framework For Web Artisans.
<http://laravel.com/>.
- [77] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

- [78] Martin Heller. REST and CRUD: the Impedance Mismatch — InfoWorld.
[http://www.infoworld.com/article/2640739/
application-development/rest-and-crud--the-impedance-mismatch.html](http://www.infoworld.com/article/2640739/application-development/rest-and-crud--the-impedance-mismatch.html).
- [79] Stefan Jauker. 10 Best Practices for Better RESTful API — Thinking Mobile.
[http://blog.mwaysolutions.com/2014/06/05/
10-best-practices-for-better-restful-api/](http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/).