

DemoCritics

Aplicación Android de participación política con edición colaborativa en tiempo real

**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 Menéndez



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/>

Agradecimientos

Dejando a un lado su obligatoriedad, no suele ser tarea sencilla sacar adelante un proyecto de estas características, cuya finalidad entendemos que es combinar gran parte de los conocimientos que hemos aprendido durante nuestro paso por las aulas y lanzarnos por primera vez al desarrollo de un trabajo formal. Exige, entre otras cualidades, curiosidad, trabajo duro, investigación, empeño, dedicación...

Queremos expresar nuestro agradecimiento a todas aquellas personas que han colaborado para que este trabajo, pese a algunas complicaciones surgidas por el camino, haya llegado a buen destino.

Lo primero, a los directores Pablo y Samer, sin cuyos consejos, orientaciones y ayuda no habríamos sido capaces de finalizar este trabajo.

Lo segundo, a los chicos de Labodemo y a Javier de la Cueva, por su inestimable colaboración al dedicarnos un poco de su preciado tiempo para compartir ideas, aportar nuevos puntos de vista al proyecto y de paso ponernos un poco los pies sobre la tierra.

Lo tercero, a amigos y familiares, por apoyarnos y hacernos más llevadero el arduo proceso de terminar un proyecto como este.

Y por último, pero no por ello menos importante, a ti, lector, por invertir un poco de tu valioso tiempo en prestar atención a lo que se expone en estas páginas. Esperemos que encuentres en su lectura algo que haga volar tu imaginación, reflexionar y quién sabe si aprender tanto como nos lo ha hecho hacer a nosotros.

”Cuanto más siniestros son los deseos de un político, más pomposa, en general, se vuelve la nobleza de su lenguaje.”

Aldous Huxley. *Un Mundo Feliz.*

Índice general

	Page
Autorización de Difusión y Uso	III
Agradecimientos	V
Abstract	IX
Resumen	XI
1. Introducción	1
1.1. Marco teórico de la idea: Política en el mundo de la Informática	1
1.2. Adentrándonos en la idea	2
1.3. Objetivos del Proyecto	3
1.4. Estructura del Documento	4
2. Estado del Arte	5
2.1. Edición Colaborativa en Tiempo Real	5
2.1.1. APIs Centralizadas	5
2.1.1.1. Google Realtime API	5
2.1.1.2. Microsoft RTC Client API	6
2.1.1.3. WebRTC	6
2.1.1.4. Mozilla TogetherJS	6
2.1.1.5. ShareJS	7
2.1.1.6. Goodow	7
2.1.2. Plataformas Web y Android	7
2.1.2.1. Wave: Wave In A Box (WIAB)	7
2.1.2.2. Google Docs	8

2.1.2.3.	Etherpad	9
2.1.2.4.	Colorillo	10
2.1.2.5.	ShareLaTeX	11
2.1.2.6.	Samepage	12
2.1.2.7.	Quip	13
2.2.	Aplicaciones de Participación Política y Ciudadana	14
2.2.1.	Programas Políticos	14
2.2.1.1.	UPyD Parla	15
2.2.1.2.	¶RecuperaCórdoba	16
2.2.1.3.	PSOE Andalucía	17
2.2.1.4.	PP Canarias	18
2.2.2.	Participación Ciudadana	19
2.2.2.1.	Reddit	19
2.2.2.2.	Change.org	20
2.2.2.3.	Programas Colaborativos de Ahora Madrid y Zaragoza en Común	22
2.2.2.4.	Appgree	24
3.	Metodología del Proyecto	27
3.1.	Uso de Software Libre	27
3.2.	Metodología de Migración de Wave a Android	28
3.2.1.	Objetivo	28
3.2.2.	Plataforma: Entorno de Desarrollo, Construcción y Depuración	28
3.2.2.1.	Eclipse	28
3.2.2.2.	Proceso de Construcción por Consola	31
3.2.2.3.	Proceso de Depuración	33
3.2.3.	Migración: Identificación y Solución de Problemas	34
3.2.3.1.	Conexión HTTP	35
3.2.3.2.	Conexión WebSocket	37

3.2.3.3.	Conexión final y Logging	39
3.2.3.4.	Organización del código: Servicio Android . .	40
3.2.4.	Dependencias	44
3.2.5.	Resultado de la Migración	44
3.3.	Implementación de DemoCritics	46
3.3.1.	Metodología	46
3.3.1.1.	Control de versiones	46
3.3.1.2.	GitHub	47
3.3.1.3.	Reparto de tareas	47
3.3.1.4.	Revisiones de código	47
3.3.1.5.	Evaluaciones de usabilidad	47
4.	Tecnologías del Proyecto	49
4.1.	Tecnologías de Wave	51
4.1.1.	Google Wave	51
4.1.2.	Apache Wave	51
4.1.3.	Características de Wave	51
4.1.3.1.	Federación	51
4.1.3.2.	Consistencia en tiempo real	52
4.1.3.3.	Escalabilidad	52
4.1.4.	Servidores Wave: SwellRT	52
4.1.5.	Eclipse	53
4.1.6.	Java	53
4.1.7.	GWT	54
4.1.8.	JavaScript	54
4.1.9.	HTTP	55
4.1.10.	WebSocket	55
4.1.11.	Git	55
4.1.12.	GitHub	56
4.2.	Tecnologías de la Aplicación Android	56

4.2.1.	Android Studio	56
4.2.2.	Android	57
4.2.3.	XML	57
4.2.4.	JSON	58
4.2.5.	SQL	58
4.2.6.	MySQL	58
4.2.7.	PhpMyAdmin	59
4.2.8.	PHP	59
4.2.9.	Laravel 5	59
4.2.10.	OpenShift	60
4.2.11.	Sublime Text	60
4.2.12.	POP: Prototyping On Paper	61
5.	Migración de SwellRT a Android	63
6.	Diseño de la Aplicación	65
6.1.	Introducción	65
6.2.	Brainstorming de ideas	65
6.3.	Diseño Guiado Por Objetivos (DGO)	67
6.3.1.	Investigación	68
6.3.1.1.	Intención Inicial: Prototipo básico	68
6.3.1.2.	Hipótesis de personas	70
6.3.1.3.	Entrevista con Labodemo	71
6.3.1.4.	Conclusión	72
6.3.1.5.	Entrevista con Javier de la Cueva	73
6.3.1.6.	Conclusión	74
6.3.2.	Modelado de Personas	75
6.3.3.	Definición de personas	76
6.3.4.	Definición de Escenarios y Requisitos	76
6.3.5.	Framework de diseño	80

6.3.5.1.	Framework de interacción	80
6.3.6.	Principios de Diseño	89
7. Arquitectura del Proyecto		93
7.1.	Arquitectura de Wave	93
7.1.1.	Modelo Conversacional Wave	94
7.1.2.	Modelo de Contenidos SwellRT	95
7.2.	Arquitectura de la aplicación	96
7.2.1.	Base de datos	97
7.2.1.1.	<i>political_party</i>	99
7.2.1.2.	<i>section</i>	99
7.2.1.3.	<i>proposal</i>	99
7.2.1.4.	<i>comment</i>	100
7.2.1.5.	<i>opinion</i>	100
7.2.2.	Service REST	100
7.2.2.1.	Arquitectura	101
7.2.2.2.	Rutas	104
7.2.2.3.	Controladores	105
7.2.3.	Cliente Android	107
7.2.3.1.	Interfaz Gráfica	107
7.2.3.2.	Estructuración de Programas Políticos	112
7.2.3.3.	Conexión y Peticiones a Service REST y Base de Datos	115
7.2.3.4.	Gestión de Usuarios: ID de Android	117
7.2.3.5.	Servicio Android, Conexión con Wave/SwellRT y Gestión de Propuestas Colaborativas	117
8. Evaluación con Usuarios		121
8.1.	Propósitos y objetivos de la evaluación	121
8.2.	Preguntas de investigación	121
8.3.	Requisitos para los participantes de la evaluación	122

8.4.	Diseño experimental	123
8.5.	Lista de tareas a realizar	123
8.6.	Entorno y herramientas que vamos a emplear	124
8.7.	Tareas del moderador	125
8.8.	Resultados de la evaluación con los usuarios	125
8.8.1.	Comentarios sobre la interfaz	126
8.8.2.	Resultados de las tareas	127
8.8.3.	Informe de hallazgos y recomendaciones	128
9.	Resultados y Trabajo Futuro	131
9.1.	Discusión de Resultados	131
9.1.1.	El porqué de DemoCritics	132
9.1.2.	Usando Wave/SwellRT en DemoCritics	133
9.1.3.	Incentivando el uso social de DemoCritics	134
9.1.4.	Aspectos técnicos de DemoCritics	135
9.2.	Reparto de Trabajo entre los componentes del grupo	136
9.3.	Trabajo Futuro	138
9.3.1.	Mejoras a la versión actual	138
9.3.2.	Nuevas características	139
9.3.3.	SwellRT Android	140
A.	Listados de Código Fuente	141
A.1.	Migración de Wave a Android: SwellRT	141
A.1.1.	Esquema de conexión HTTP	141
A.1.2.	Esquema de conexión con AsyncTask	142
A.1.3.	Esquema de uso de WAsync	142
A.2.	Service REST (Laravel)	144
A.2.1.	Esquema de Rutas Simple	144
A.2.2.	Esquema de Rutas Agrupadas	144
A.2.3.	Esquema de Rutas de Propuestas Agrupadas	144

A.2.4. Esquema de Controlador de Propuestas	145
A.3. Aplicación Android	145
A.3.1. Función de construcción del árbol de Programa Político	145
A.3.2. Registrar Usuario en SwellRT	146
A.3.3. Crear nueva Wave (Model) en SwellRT	148
A.3.4. Abrir Pad colaborativo en SwellRT	150
Bibliografía	150

Índice de figuras

2.1.	Cliente Wave In A Box	8
2.2.	Capturas de Google Docs	9
2.3.	Captura de TitanPad: Ejemplo de uso de EtherPad	10
2.4.	Captura de Colorillo	11
2.5.	Captura de ShareLaTeX	12
2.6.	Capturas de Samepage	13
2.7.	Capturas de Quip	14
2.8.	Capturas de UPyD Parla	15
2.9.	Capturas de #IURecuperaCórdoba	16
2.10.	Capturas de PSOE Andalucía	17
2.11.	Capturas de PP Canarias	18
2.12.	Plaza Podemos utilizando la plataforma Reddit	20
2.13.	Change.org · La mayor plataforma de peticiones del mundo . .	21
2.14.	Creación colaborativa del programa de Ahora Madrid.	23
2.15.	Creación colaborativa del programa de Zaragoza en Común. .	23
2.16.	Capturas de Appgree	24
3.1.	Distribución Actual de Versiones Android (Fuente: Google) . .	30
3.2.	Emulador Android API 19	32
3.3.	Ejemplo de Traza de Error en Logcat	36
3.4.	Pantalla de Login de WaveAndroid	40
3.5.	Proceso de conexión WebSocket con Servicio	43
3.6.	Proceso de conexión Http con Servicio	44
3.7.	Esquema de Clases de SwellRT-Android con Servicio	45
4.1.	Nube de Tecnologías utilizadas en el Proyecto	49

4.2. Frameworks PHP más populares. Fuente: SitePoint	60
6.1. Capturas del brainstorming	66
6.2. Estudiante universitario	77
6.3. Activista político	77
6.4. Modelo de desarrollo de los prototipos.	83
6.5. Primeros prototipos sobre programas electorales.	84
6.6. Prototipos para el desarrollo de propuestas.	85
6.7. Pantalla principal de la aplicación.	88
6.8. Vista de categorías.	89
6.9. Principios de diseño en la aplicación.	92
7.1. Arquitectura de Wave	94
7.2. Modelo Conversacional de Wave	95
7.3. Modelo de Contenidos de SwellRT	96
7.4. Arquitectura general de la aplicación.	97
7.5. Modelo entidad-relación de la base de datos.	98
7.6. Arquitectura de Laravel	103
7.7. Camino desde la ruta al método del controlador.	106
7.8. Esquema de Implementación de TopItem (Con vista de Section y Proposal)	110
7.9. Esquema de Implementación de Menú Izquierdo (Con vista del Menú)	111
7.10. Ejemplos de Codificación de Secciones y Subsecciones	113
7.11. Esquema de estructura en árbol de Programa Político	114
7.12. Esquema de Clases de Conexión HTTP mediante AsyncTask .	116
7.13. Esquema de Uso de Wave/SwellRT	119

Índice de tablas

3.1. Software libre utilizado durante el desarrollo.	27
3.2. Dependencias de SwellRT-Android	46
4.1. Tecnologías usadas en el Proyecto	50
7.1. Funciones CRUD	101
7.2. Códigos de estado de la respuesta del servidor	102
7.3. Adapters, Views y Clases utilizadas en el proyecto	112
8.1. Dificultad y tiempos de las tareas para el usuario del tipo <i>ciudadano</i>	127
8.2. Dificultad y tiempos de las tareas para el usuario del tipo <i>activista social</i>	128

Abstract

Nowadays we live in a society of democratic change in which people begin to take a decisive role in the political area. There are increasingly social movements that proliferate due to the interest generated by taking part in politics. This participation has been channeled by the emergence of new technologies that allows to establish new ways for people to organize and express their opinion. The vast majority of these new tools are developed in a scope near to Web-based interfaces and interconnected network platforms. Being as we are in a time subject to several electoral events that favours the increased interest of people involved in politics, the development of such type of tools is becoming increasingly necessary.

Keeping in mind this context, we contemplate the development of an application that provides access to new ways of participation from mobile devices. This application will have two clear objectives: to bring to the forefront political programmes, which usually few people read, encouraging its reading and discussion; and open a common space in which citizens express their alternative proposals to those put forward by the political parties that represent us.

We will base on the use of already existing Open-Source technologies for collaborative editing in real-time, adjusting them to mobile devices. Specifically we will explore the use of Web-based platform Apache Wave, carrying out a migration process that allows to take advantage of its potential in Android-based devices. This migration supposes an additional work that involves an effort to research into the development of an innovative platform, because there is no equivalent free software currently on Android. By using this technology, users will have the opportunity to compose real-time collaborative texts.

As a result for this work we will develop a first version of the application which, as a proof of concept, will make use of the features described above. Thus, the ultimate aim we set out for this project is to put into practice its usefulness during future electoral events.

Keywords:

Collaborative Writing, Politics, Democracy, Citizen Participa-

tion, Real-Time, Android, Apache Wave, Elections, Election Programme, Citizen Proposals.

Resumen

Actualmente vivimos en una sociedad de cambio democrático en la que las personas comienzan a adquirir un papel determinante en el terreno político. Son cada vez más los movimientos sociales que proliferan a raíz del interés generado por participar en política. Esta participación se ha visto canalizada por la aparición de nuevas tecnologías que permiten establecer nuevas formas para que las personas se organicen y expresen su opinión. La gran mayoría de estas nuevas herramientas se desarrollan en un ámbito cercano a plataformas interconectadas en red y basadas en una interfaz Web. Encontrándonos en un momento sujeto a varias citas electorales que propicia el aumento del interés de las personas por involucrarse en política, el desarrollo de este tipo de herramientas se hace cada vez más necesario.

Teniendo en cuenta este contexto, se plantea el desarrollo de una aplicación que proporcione acceso a nuevas formas de participación desde dispositivos móviles. Dicha aplicación tendrá dos objetivos claros: poner en un primer plano los programas electorales (que normalmente pocas personas leen) incentivando su lectura y debate; y abrir un espacio común en el que la ciudadanía exprese sus propuestas alternativas a las soluciones expuestas por los partidos políticos que les representan.

Nos basaremos en el uso de tecnologías de Software libre ya existentes de edición colaborativa en tiempo real, adaptándolas a dispositivos móviles. Concretamente se explorará el uso de la plataforma Web Apache Wave, llevando a cabo un proceso de migración que permita explotar su potencial en dispositivos basados en Android. Esta migración supone un trabajo adicional que conlleva un esfuerzo de investigación y desarrollo de una plataforma innovadora, ya que no existe una tecnología de software libre equivalente en Android actualmente. Haciendo uso de esta tecnología los usuarios tendrán la oportunidad de redactar textos colaborativos en tiempo real.

Como resultado de este trabajo se desarrollará una primera versión de la aplicación que, a modo de prueba de concepto, haga uso de las funcionalidades antes descritas. De esta manera el objetivo final que nos planteamos es poner en práctica su uso durante futuras citas electorales.

Palabras Clave:

Escritura Colaborativa, Política, Democracia, Participación Ciudadana, Real-

Time, Android, Apache Wave, Elecciones, Programas Electorales, Propuestas Ciudadanas.

Capítulo 1

Introducción

A continuación se exponen de forma resumida los objetivos que se persiguen al realizar este Trabajo de Fin de Grado y un esquema de la estructura de esta Memoria.

ANTECEDENTES Y ALARGAR

1.1. Marco teórico de la idea: Política en el mundo de la Informática

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

A primera vista se puede pensar que la política no parece entusiasmar a las personas dedicadas al mundo de la informática. Se puede ubicar la política como una parte de las ciencias sociales o la actividad política, situando la informática en ciencias exactas o formales. Pero pensando en factores como la gestión de los privilegios de una aplicación entre los que se define de alguna forma una jerarquía, indirectamente se estará haciendo política. También se pueden encontrar características políticas en el diseño relacional de una base de datos. Definiendo los campos de una base de datos se pueden ver algunos valores como el sexo, la nacionalidad, la edad o incluso las relaciones o restricciones que existen entre las tablas. Se estarán definiendo unas reglas básicas de funcionamiento de la base de datos establecidas por unos principios políticos.

Adentrándose en el mundo de las Licencias (copia, modificación, distribución, etcétera) en el desarrollo de *Software* se encuentra más contenido político. 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 se establecen en

CAPÍTULO 1. INTRODUCCIÓN

la metodología del desarrollo orientado a objetos, estableciendo las relaciones de herencia, restricción de métodos, variables, etcétera.

Regresando a la actualidad y basándose en no muy lejanos acontecimientos pasados, se ha oído cómo algunos gobiernos recopilan datos de la actividad de los usuarios en las redes sociales [?], analizando todo el contenido que generan. Incluso cómo algunas aplicaciones móviles piden aprobar permisos con los que operar libremente en los dispositivos.

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

1.2. 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 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

CAPÍTULO 1. INTRODUCCIÓN

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 2.2.2).

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.

1.3. Objetivos del Proyecto

El objetivo principal de este proyecto es desarrollar una aplicación Android de utilidad social y que haga uso de una tecnología poco usada en estas plataformas móviles como es la colaboración en tiempo real, que desde hace unos años sí que viene estando más presente en plataformas Web. Para ello primero se evaluará la tecnología existente en el proyecto SwellRT (basado en Apache Wave) y se adaptará dicha tecnología para poder hacer uso de ella de forma nativa desde Android. Después se discutirán posibles ideas de aplicación que puedan hacer uso de esta tecnología y se diseñará y desarrollará una primera versión estable de esa idea de aplicación Android que pueda ser evaluable por usuarios. También se estudiará la viabilidad de seguir desarrollando a futuro la aplicación con vistas a conseguir un producto que podamos poner a disposición del público en el *marketplace* de Android.

- 1^a parte: Migración de Wave (SwellRT) a Android
 - Proporcionar un entorno de colaboración en tiempo real federado para Android basado en SwellRT.
- 2^a parte: Creación de la aplicación Android.
 - Desarrollar una aplicación que recopile y permita interactuar con los programas electorales de los partidos políticos.
 - Desarrollar una aplicación que permita la participación colectiva de ciudadanos mediante propuestas colaborativas editables en tiempo real.

1.4. Estructura del Documento

En esta memoria se han querido reflejar los aspectos más destacados del proceso de desarrollo de este Trabajo de Fin de Grado. Aunque a lo largo del documento se detallen aspectos técnicos queremos dejar patente que no se trata de un tutorial de cómo funciona Wave o Android. Aunque a veces se entre en mayor detalle por cuestiones de claridad a la hora de entender el funcionamiento de la tecnología, se anima al lector si quiere profundizar más en el tema a que haga uso de las múltiples referencias bibliográficas que se incluyen en este documento.

A lo largo de este documento el lector se encontrará con una estructura basada en capítulos en los que se irán detallando distintas facetas del desarrollo del proyecto:

- **Capítulo 2 - Construyendo la idea de la aplicación DemoCritics:** se expone cómo y por qué se llegó a la idea de esta aplicación tras haber realizado la migración de SwellRT a Android.
- **Capítulo 3 - Estado del Arte:** se hace un pequeño análisis de las características de actuales soluciones software que hacen uso de tecnologías de colaboración en tiempo real y de aplicaciones dedicadas a la participación política.
- **Capítulo 4 - Tecnologías del Proyecto:** se repasan las tecnologías y herramientas utilizadas durante todo el desarrollo del proyecto.
- **Capítulo 5 - Metodologías del Proyecto:** se detallan el proceso de migración de SwellRT y metodologías utilizadas el diseño, implementación y evaluación de DemoCritics.
- **Capítulo 6 - Arquitectura del Proyecto:** se indaga en aspectos técnicos destacados de la organización de DemoCritics.
- **Capítulo 7 - Resultados, Conclusiones y Trabajo Futuro:** se discuten los resultados obtenidos con el objeto de sacar algunas conclusiones. Se discuten los siguientes pasos a realizar con el objetivo de mejorar o cambiar DemoCritics.

Capítulo 2

Estado del Arte

2.1. Edición Colaborativa en Tiempo Real

En esta sección veremos algunas de las soluciones disponibles actualmente que, al igual que Wave, permiten editar contenido de forma colaborativa y en tiempo real. Este contenido puede ser muy variado, aunque lo más usual suele ser proporcionar edición de texto en tiempo real mediante el uso de Transformaciones Operacionales (OT) [?]. Sin embargo, todas estas plataformas utilizan una arquitectura de servidor centralizado para ofrecer estas funcionalidades, mientras que Wave utiliza una arquitectura federada en la que no existe un servidor central (Ver Sección 4.1.3.1).

De esta manera, primero veremos las principales herramientas de desarrollo (APIs) más utilizadas actualmente, para después pasar a ver algunas de las plataformas que hacen uso de esta tecnología. En su mayoría son para web, aunque existen unas pocas para Android.

2.1.1. APIs Centralizadas

La mayor parte del desarrollo de tecnologías de Colaboración en Tiempo Real (RTC) se realiza mediante APIs que son propiedad de determinadas compañías y que no utilizan una arquitectura federada, sino que toda la información pasa por un servidor central que controla el RTC. A continuación veremos algunas de las más utilizadas hoy en día.

2.1.1.1. Google Realtime API

Google Realtime API [?] permite construir aplicaciones de colaboración en tiempo real utilizando la tecnología de Transformaciones Operacionales (OT) presente en Google Docs. Utiliza JavaScript para poder construir construir en nuestro cliente web un modelo de datos (*Realtime Data Model*) que se guarda en sus servidores y gestiona automáticamente los cambios realizados

por cualquiera de los usuarios que colaboran en tiempo real. Estos cambios en el modelo son notificados al servidor y al resto de usuarios mediante eventos que se lanzan al modificar los datos sobre los que se está colaborando. Para utilizarlo es necesario tener cuenta en la Consola de Desarrolladores de Google y activar el uso de esta API con nuestras credenciales de usuario. Existe asimismo una versión (privativa) de esta API compatible con Android.

2.1.1.2. Microsoft RTC Client API

Microsoft RTC Client API [?] permite construir aplicaciones que permitan realizar llamadas de audio/video o sesiones de mensajería instantánea (IM) de texto por Internet y en tiempo real. Las aplicaciones se deben escribir en C++ o Visual Basic y pueden ser utilizadas tanto en PC como en dispositivos móviles siempre que ejecuten un sistema operativo Windows.

2.1.1.3. WebRTC

WebRTC [?] (Web Real-Time Communication) es un API open-source (bajo licencia BSD) actualmente en desarrollo por Google y la World Wide Web Consortium (W3C) y que pretende dotar a los navegadores web de capacidades de comunicación en tiempo real entre sí sin necesidad de plugins externos. En la actualidad se encuentra en su versión 1.0 y soporta los navegadores Firefox y Chrome.

2.1.1.4. Mozilla TogetherJS

Mozilla TogetherJS [?] es una librería gratuita y de código libre (bajo licencia pública Mozilla v2) que permite añadir capacidades de colaboración en tiempo real a una página web. Utiliza WebRTC y webSockets para establecer comunicaciones peer-to-peer (P2P) entre dos o más navegadores Web. No proporciona almacenamiento persistente de los datos y es necesario tener un Servidor que establezca la conexión. Con esta herramienta se puede editar texto en tiempo real (usando Transformaciones Operacionales), establecer chats de audio/video y sincronizar el contenido de los navegadores. Está disponible en *GitHub* [?] para contribuir a su desarrollo.

2.1.1.5. ShareJS

ShareJS [?] es una plataforma open-source (bajo licencia MIT) que dispone de un pequeño servidor basado en Node.js y una librería de cliente JavaScript que permiten la edición colaborativa de contenido mediante Transformaciones Operacionales. Permite actuar sobre objetos JSON o sobre texto plano. Está disponible en *GitHub* para contribuir a su desarrollo.

2.1.1.6. Goodow

Goodow [?] es un framework open-source de reciente desarrollo que proporciona un API muy similar a *Google Real-Time API* (Ver Sección 2.1.1.1) para colaboración en tiempo real mediante el uso de Transformaciones Operacionales. Dispone asimismo de dos clientes básicos para Android e iOS, utilizando una implementación del Servidor propia.

2.1.2. Plataformas Web y Android

En las siguientes secciones veremos algunas de las plataformas Web y Android que hacen uso de tecnologías de Colaboración en Tiempo Real.

2.1.2.1. Wave: Wave In A Box (WIAB)

Wave In a Box (WIAB) [?] es el código fuente liberado por Google y completado por la comunidad de Apache tras pasar el proyecto a sus manos en el año 2012. Al igual que el resto del código de la tecnología que heredó de Google, está implementado en Java usando OpenJDK [?]. La instalación trae consigo un cliente web desarrollado en Java usando el framework Google Web Toolkit (GWT) [?], lo que genera código JavaScript ejecutable en el navegador. Este cliente web sirve como prueba de concepto de las funcionalidades básicas del Modelo Conversacional de Wave, pudiendo gestionar waves, usuarios y extensiones. Actualmente cualquiera puede descargar y desplegar WIAB en su ordenador siguiendo los pasos que nos proporcionan en su wiki [?]. La aplicación se distribuye en forma de código fuente, accesible entre otras formas desde su repositorio de GitHub [?]. Existen asimismo servidores de prueba ya desplegados en Internet sobre los que se puede observar el funcionamiento de WIAB [?].

CAPÍTULO 2. ESTADO DEL ARTE

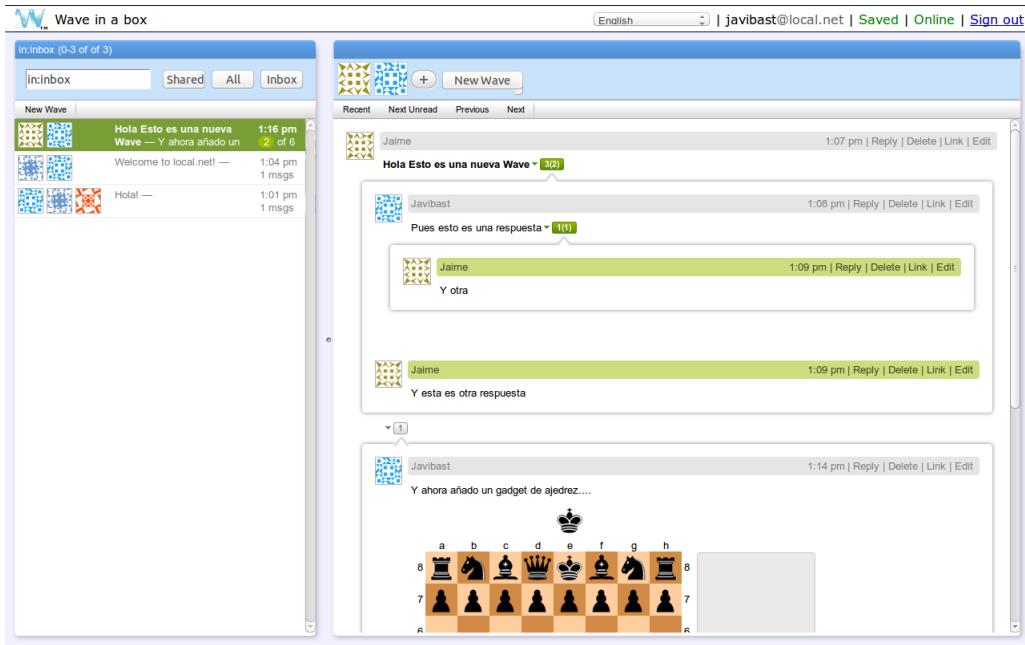
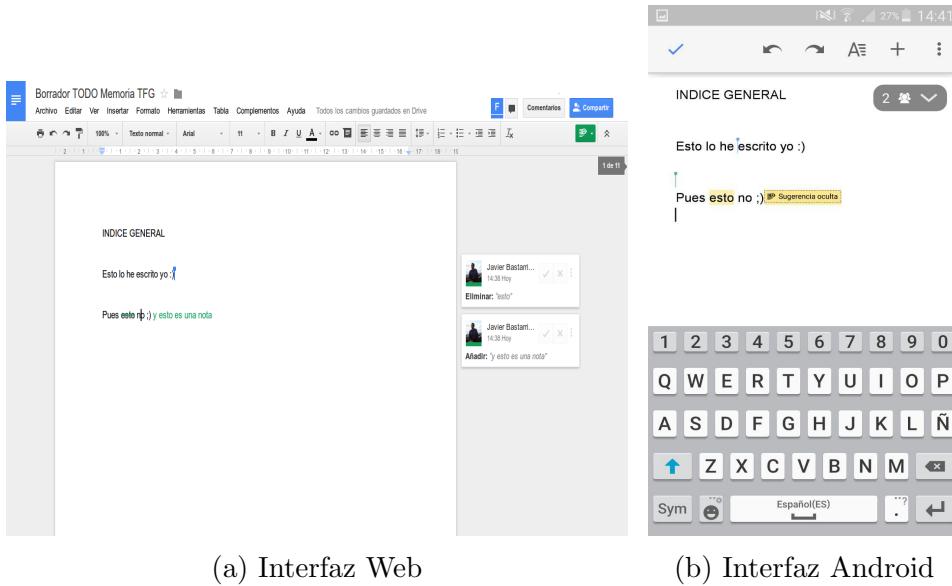


Figura 2.1: Cliente Wave In A Box

2.1.2.2. Google Docs

Google Docs [?] es la plataforma de Google para edición de documentos de forma colaborativa y en Tiempo Real usando su API Realtime (Ver Sección 2.1.1.1). Permite que varios usuarios con cuenta de Google creen y editen colaborativamente un documento a la vez. Puedes ver los cursores de cada usuario e interactuar con ellos mediante un chat. También permite escribir sugerencias a modo de notas en el margen sobre lo ya escrito. Dispone de versión web y móvil, pudiendo interactuar entre ellas sin problemas.

CAPÍTULO 2. ESTADO DEL ARTE



(a) Interfaz Web

(b) Interfaz Android

Figura 2.2: Capturas de Google Docs

2.1.2.3. Etherpad

Etherpad [?] es un editor colaborativo en tiempo real open-source (bajo licencia Apache 2.0). Permite a varios autores editar a la vez un mismo documento de texto, resaltando en distintos colores lo editado por cada persona y con la opción de un chat para comunicarse entre sí. Existen múltiples servicios que hacen uso de este editor, siendo uno de las más conocidas TitanPad [?].

CAPÍTULO 2. ESTADO DEL ARTE

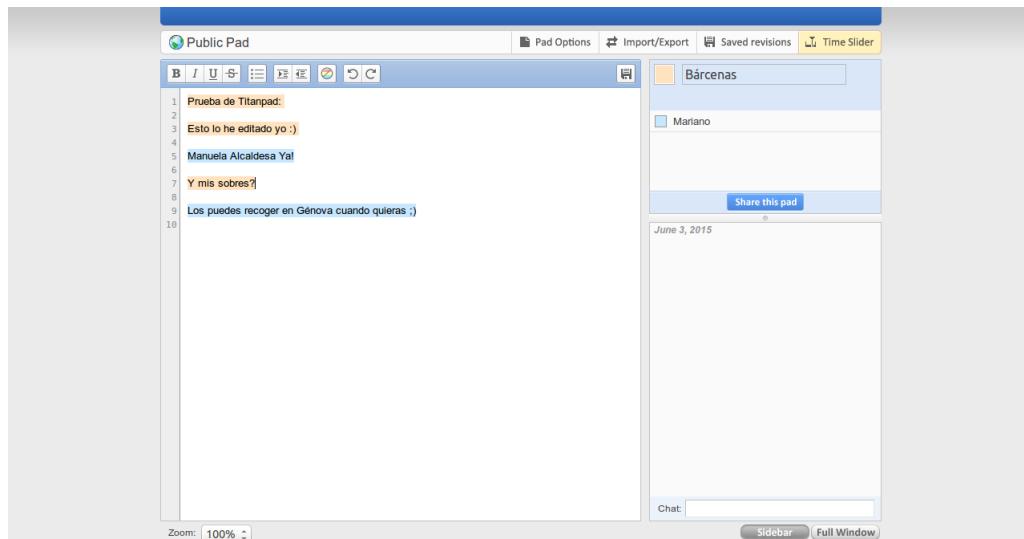


Figura 2.3: Captura de TitanPad: Ejemplo de uso de EtherPad

2.1.2.4. Colorillo

Colorillo [?] es una aplicación web básica de dibujo colaborativo en tiempo real. Cualquier usuario puede empezar a dibujar sobre un nuevo lienzo en blanco con diversos colores y compartir este lienzo con otros usuarios para dibujar entre todos. De cada usuario podemos ver su procedencia aproximada sobre un mapa y el color que actualmente está utilizando. Existe también opción para chatear con otros usuarios. Los dibujos se pueden descargar y tienen todos licencia Creative Commons BY 3.0.

CAPÍTULO 2. ESTADO DEL ARTE

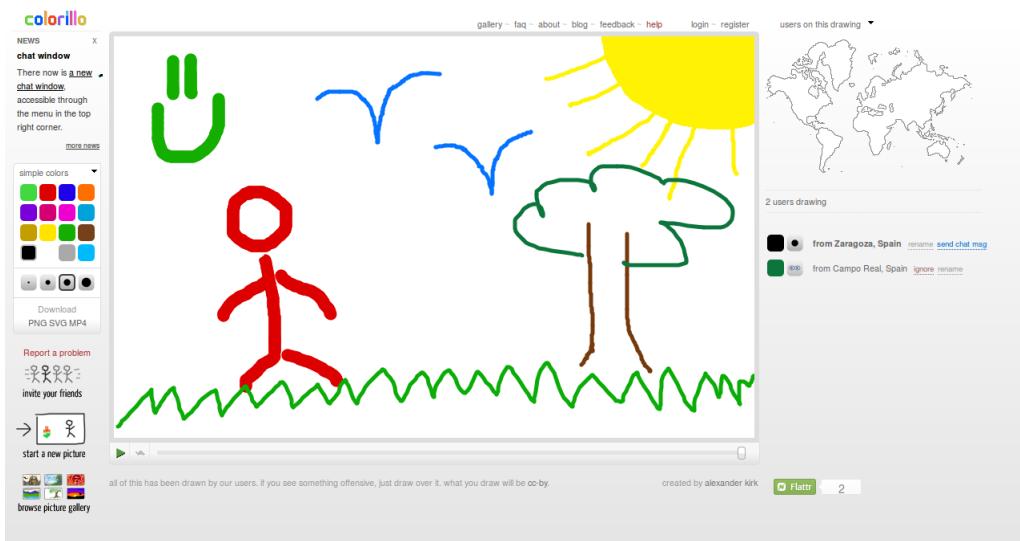


Figura 2.4: Captura de Colorillo

2.1.2.5. ShareLaTeX

ShareLaTeX [?] es un editor web colaborativo de documentos escritos en LaTeX en tiempo real. Permite elaborar documentos LaTeX entre varias personas, ofreciendo una interfaz que incluye una previsualización del resultado en PDF. Desde 2014 es open-source (bajo licencia AGPL v3) y cualquiera puede descargarselo de *GitHub* e instalar su propio servidor (escrito en Node.js) de ShareLaTeX. En su web ofrecen también opciones de pago con extras como un control de versiones o sincronización con Dropbox.

CAPÍTULO 2. ESTADO DEL ARTE

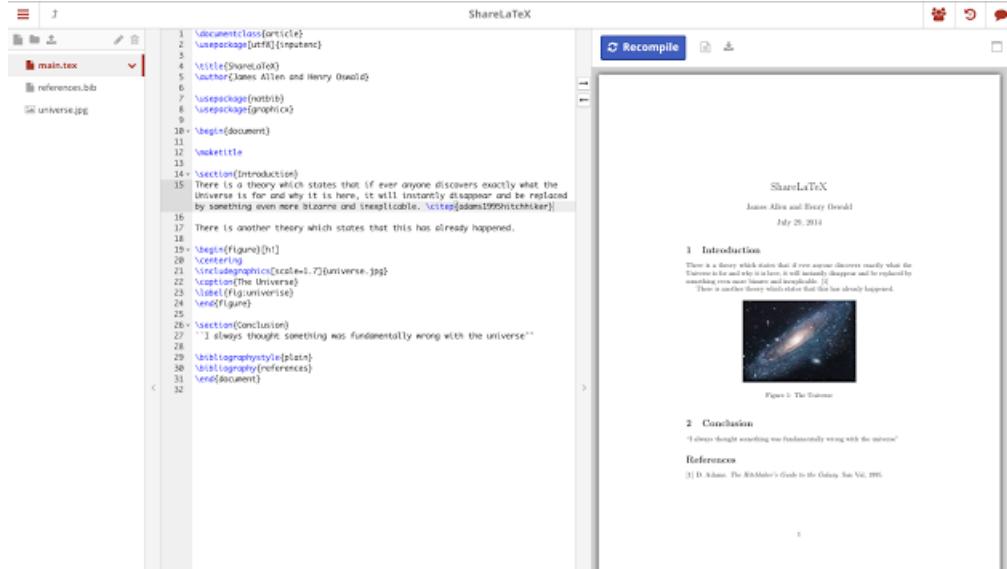
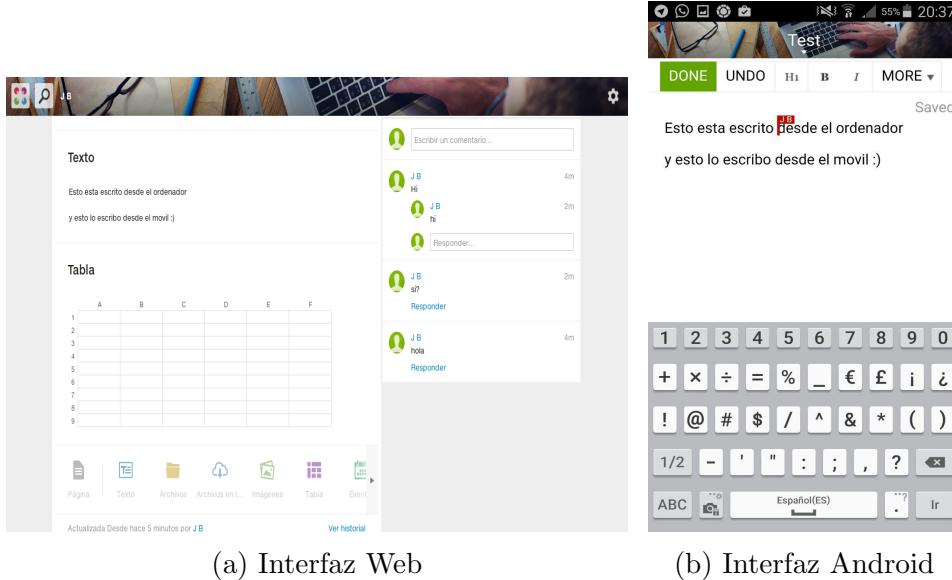


Figura 2.5: Captura de ShareLaTeX

2.1.2.6. Samepage

Samepage [?] es una aplicación, tanto para web como para plataformas móviles (Android e iOS), que permite crear documentos que pueden ser editados de forma colaborativa y en tiempo real por múltiples usuarios. Para ello es necesario solo tener una cuenta de samepage. Tanto el cliente web como el móvil permiten interactuar entre ellos para crear nuevas páginas, editar texto y hacer comentarios, aunque la versión web permite también añadir tablas, imágenes y archivos.

CAPÍTULO 2. ESTADO DEL ARTE



(a) Interfaz Web

(b) Interfaz Android

Figura 2.6: Capturas de Samepage

2.1.2.7. Quip

Quip [?] es una aplicación para dispositivos móviles (Android e iOS) desarrollada con el objetivo de aumentar la productividad en los trabajos en grupo. Permite elaborar documentos, hojas de cálculo y listas de tareas compartidas y editables de forma colaborativa en tiempo real, pudiendo realizar también comentarios sobre ellas. Dispone también de una versión web, pero es necesario tener cuenta para usarla.

CAPÍTULO 2. ESTADO DEL ARTE

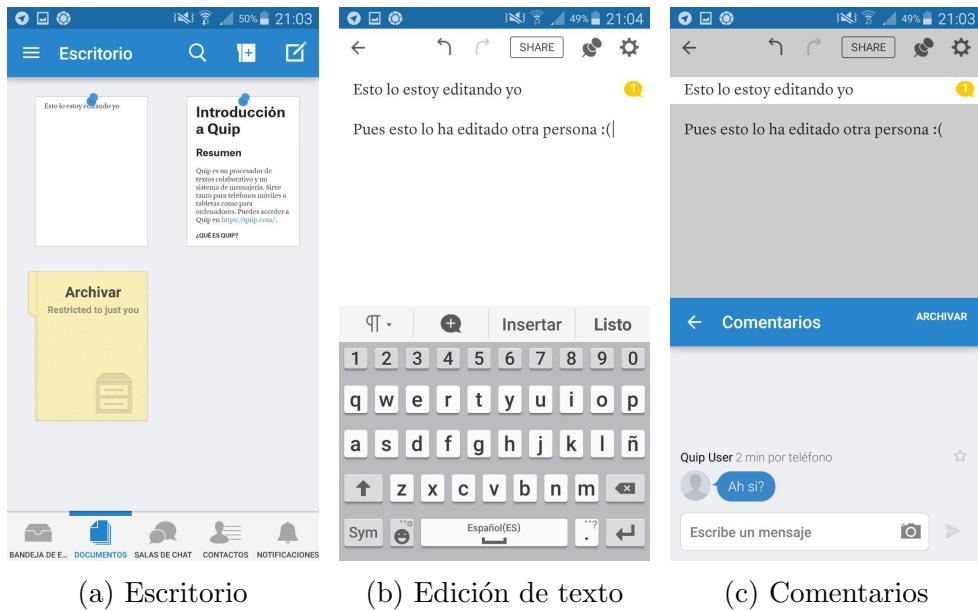


Figura 2.7: Capturas de Quip

2.2. Aplicaciones de Participación Política y Ciudadana

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.

2.2.1. 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, la presentación de la 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

CAPÍTULO 2. ESTADO DEL ARTE

y negativos.

2.2.1.1. UPyD Parla

La aplicación presenta al candidato de UpyD Carlos Alt Bustelo para la alcaldía de Parla. Se trata de una alicación divulgativa donde podemos conocer todo lo esencial de la candidatura de UpyD para las elecciones del municipio de Parla en Mayo de 2015: los candidatos, el programa, vídeos, etc.



Figura 2.8: Capturas de UPyD Parla

- Aspectos positivos:

- 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 2. ESTADO DEL ARTE

2.2.1.2. #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.

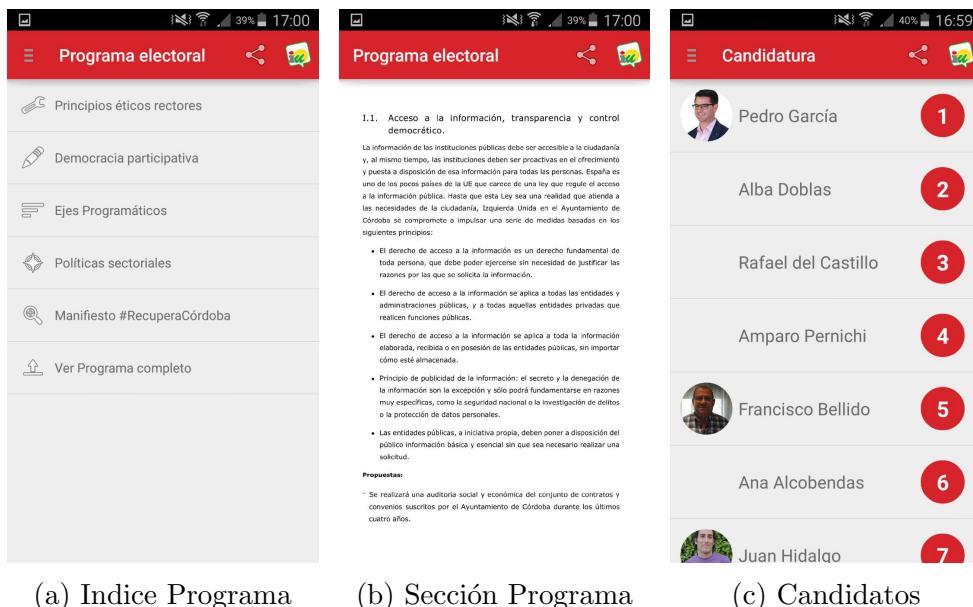


Figura 2.9: Capturas de #IURecuperaCórdoba

- Aspectos positivos:

- Interfaz limpia y sencilla de diseño plano.
- Consistencia en la aplicación: la información se presenta siempre en formato de lista ofreciendo los mínimos datos necesarios sin sobrecargar de información al usuario.
- Menú lateral disponible en cualquier pantalla con las principales acciones de la aplicación: Candidatos, Programa, Videos y Noticias.
- El programa electoral está estructurado en un índice primer nivel y a veces con segundo nivel.
- Aporta la opción de ver el programa completo.

CAPÍTULO 2. ESTADO DEL ARTE

- Aspectos negativos:

- Utiliza a veces iconos cuyo propósito no se entiende: ¿Un avión de papel para noticias? ¿Un "a>z" para la lista de candidatos?
- Las distintas secciones se abren dentro de la aplicación, pero da acceso a una navegación lateral por las páginas del PDF del programa en cuestión.
- Si haces zoom en una sección no permite pasar de página.

2.2.1.3. 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 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.

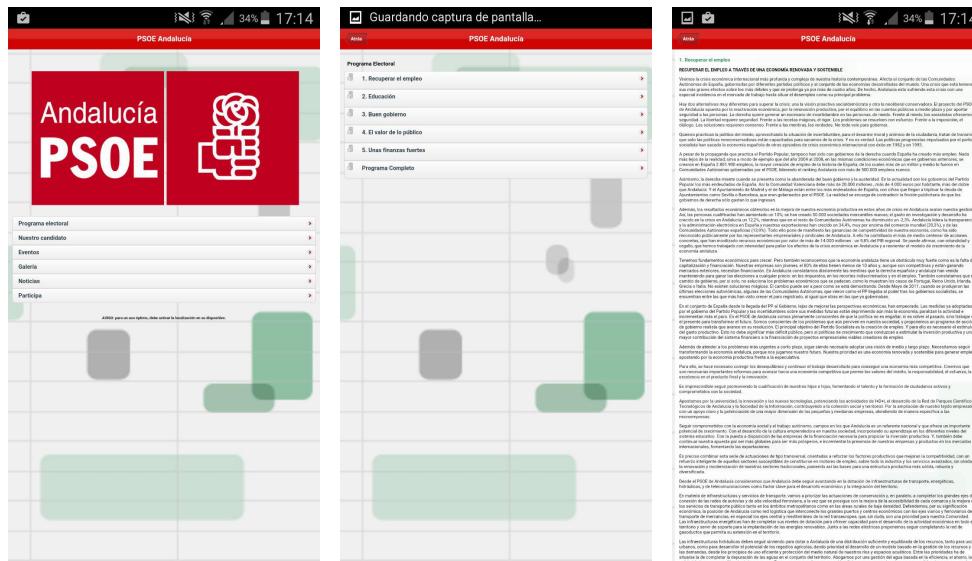


Figura 2.10: Capturas de PSOE Andalucía

- Aspectos positivos:

CAPÍTULO 2. ESTADO DEL ARTE

- Posee un indice de primer nivel para estructurar el programa.

- Aspectos negativos:

- La interfaz y los botones no se adaptan al tamaño de pantalla y permanecen de un tamaño pequeño que dificulta la interacción.
- El programa electoral se visiona en forma de una página que parece descargada directamente de la versión PDF y que permanece en un tamaño pequeño e ilegible, no permitiendo tampoco hacer zoom. En general, la interfaz parece hecha para una web más que para un móvil.

2.2.1.4. 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.

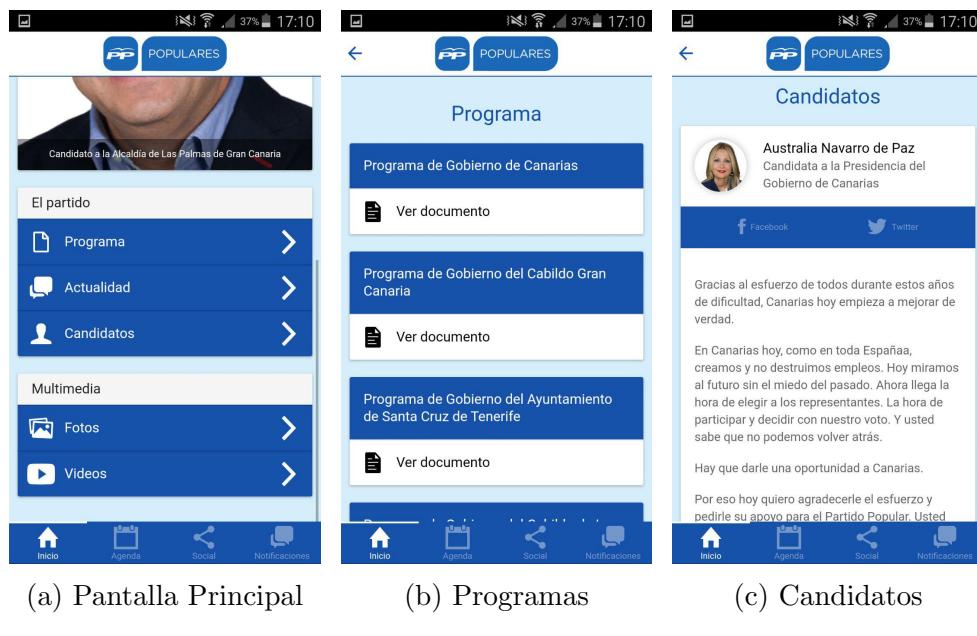


Figura 2.11: Capturas de PP Canarias

- Aspectos positivos:

CAPÍTULO 2. ESTADO DEL ARTE

- Interfaz limpia y atractiva con una organización clara en secciones que aparecen en una barra inferior al estilo de iOS.

- Aspectos negativos:

- Aunque dispone de una sección para programas electorales, no dispone de ellos en local, si no que te obliga a abrir un navegador para ver la versión entera en PDF.
- El problema de la barra inferior de menús es que quita espacio al contenido principal. Se podría haber puesto el menú en alun sitio menos intrusivo.

2.2.2. 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.

2.2.2.1. Reddit

Reddit [?] es una plataforma web de código libre 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 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. A esto lo denominan "filtrado colaborativo" pues 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 [?]: 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 en el gobierno o cuáles son los temas que más les preocupan.

CAPÍTULO 2. ESTADO DEL ARTE



Figura 2.12: Plaza Podemos utilizando la plataforma Reddit

- Aspectos positivos:

- Opción de filtrado por tipos de contenido (propuestas, noticias...) y ordenación de distintas formas (nuevos, populares, activos...)
- Sistema de votación sencillo desde la propia previsualización del contenido mediante flechas (arriba y abajo), pudiendo ver entre ellas el número actual de votos.

- Aspectos negativos:

- Para un usuario que lo usa por primera vez puede resultar confuso que haya contenido redactado con usuarios mezclado con enlaces a noticias externas.

2.2.2.2. Change.org

Change.org [?] es un portal web que permite lanzar múltiples peticiones de cambio en Internet. Podríamos definirlo como la evolución de la recogida

CAPÍTULO 2. ESTADO DEL ARTE



Figura 2.13: Change.org · La mayor plataforma de peticiones del mundo

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.

- Aspectos positivos:

- Página principal con peticiones cuyo objetivo de firmas se ha conseguido (“victoria”) y las más destacadas aun por conseguir.
- De cada petición se muestra un ”preview” con los aspectos más destaca-

CAPÍTULO 2. ESTADO DEL ARTE

cados: foto, título, autor, número de firmas y fecha de creación.

- Aspectos negativos:

- Aunque existen filtros para peticiones (Destacadas, Populares y Recientes) ¿de qué depende esta clasificación? No queda claro cómo se elige la opción en dicha lista de peticiones,

2.2.2.3. 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.

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

- Aspectos positivos:

- Las propuestas están organizadas por distintos temas: "Ejes temáticos" en el caso de Zaragoza en Común y "Áreas y Objetivos" en el caso de Ahora Madrid.
- En ambos casos se puede filtrar la lista de propuestas de distintas formas ("Más valoradas", "Más consenso" y "Más debatidas")
- En el "preview" de la propuesta se muestra el número de comentarios y el porcentaje de votos positivos en relación al número total de votos recibidos.

- Aspectos negativos:

- En cada propuesta se muestra un número a la izquierda que no explica lo que significa (entendemos que es el número de votos a favor).

CAPÍTULO 2. ESTADO DEL ARTE

#AhoraPrograma



Figura 2.14: Creación colaborativa del programa de Ahora Madrid.

The screenshot shows the homepage of the Zaragoza en Común website, featuring a red header with the Ganemos Zaragoza logo and links for INICIO and ENTRAR. The main title is "Programa colaborativo de Zaragoza en Común". A message at the top states: "Tienes que estar identificado para votar o comentar las propuestas." Below this, there are two columns: "Un programa elaborado en Común" and "Ejes temáticos".

Un programa elaborado en Común

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. 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. 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. 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.

primera fase: Un programa de consensos ciudadanos

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
- Derechos sociales
- Economía, trabajo y desigualdad

Figura 2.15: Creación colaborativa del programa de Zaragoza en Común.

CAPÍTULO 2. ESTADO DEL ARTE

2.2.2.4. Appgree

Appgree[?] 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[?].

En la aplicación podemos acceder a una lista de canales: que aglutan 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 de listas de preguntas más candentes y recientes. Además se pueden compartir preguntas por redes sociales para aumentar su difusión.

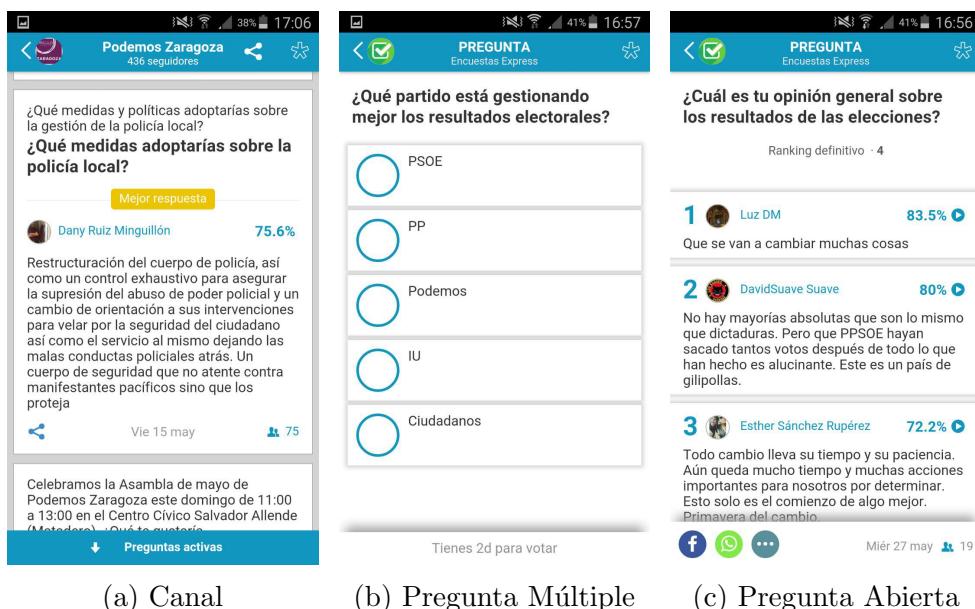


Figura 2.16: Capturas de Appgree

- Aspectos positivos:

- Interfaz limpia de colores planos (blanco y azul) con pantalla principal de ultimas preguntas destacadas.
- Organización por canales temáticos de las preguntas.

CAPÍTULO 2. ESTADO DEL ARTE

- Opción de marcar canales como favoritos para poder seguirlos y ser notificados de las últimas preguntas añadidas.

- Aspectos negativos:

- Las preguntas se organizan por antigüedad (activas y no activas) pero se navega de abajo hacia arriba, cuando uno esperaría hacerlo al revés.
- Los tipos de preguntas (abiertas, de respuesta múltiple, de sí o no, encuestas...) aparecen todas mezcladas.

Capítulo 3

Metodología del Proyecto

A continuación se exponen las diferentes metodologías que hemos utilizado durante el desarrollo de este *Trabajo de Fin de Grado*.

3.1. Uso de Software Libre

Para el desarrollo de todo el software que compone el proyecto siempre hemos utilizado una aproximación de software libre. Esto permite contribuir al común y que cualquier persona pueda hacer uso de nuestro código si lo necesita. Hemos usado software libre tanto con las herramientas necesarias para el desarrollo de código y documentación como para el resultado final de nuestra aplicación. Con esto damos la posibilidad a que otros usuarios puedan visualizar el código desarrollado o utilizarlo libremente. Para ello realizamos aportaciones a toda la comunidad subiendo el código del proyecto a **GitHub** bajo una licencia **GNU GPLv3** [?].

A continuación se expone un breve resumen del software libre utilizado y las licencias que poseen:

Software	Licencia
Eclipse	Eclipse Public License
Android Studio	Apache License 2.0
Laravel	MIT License
Apache Wave	Apache License
phpMyAdmin	GNU GPLv2
MySQL	GNU GPL
PHP	PHP License
Android	Apache License 2.0 y GNU GPLv2
Java	GNU GPL
OpenShift	Apache License 2.0

Tabla 3.1: Software libre utilizado durante el desarrollo.

3.2. Metodología de Migración de Wave a Android

3.2.1. Objetivo

El framework de SwellRT utiliza un servidor WIAB y el protocolo Wave, ambos desarrollados en Java/GWT. El **SDK de Android** [?] 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.

3.2.2. Plataforma: Entorno de Desarrollo, Construcción y Depuración

Existen dos entornos de desarrollo (IDE) recomendados por Google para desarrollar en Android: Eclipse [?] y Android Studio.[?] 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 [?] 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.

3.2.2.1. Eclipse

El IDE de Eclipse [?] soporta el desarrollo con Android a través del plugin **ADT (Android Development Tools)** [?], que integra en un solo paquete

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

te todas las herramientas necesarias para desarrollar, construir y depurar el código de la aplicación fácilmente.

Android SDK [?]: 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 [?]**
- **Android Virtual Device Manager (AVDM) [?]** herramienta para gestionar la creación, modificación, ejecución y eliminación de emuladores en Android. Un **emulador [?]** 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 [?]** 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) [?]** herramienta que provee las características de entorno de depuración para las aplicaciones en desarrollo.

Teniendo en cuenta la distribución actual de versiones instaladas en dispositivos Android [?] (Ver figura 3.1) se ha decidido realizar la migración de SwellRT con el API 19 de Android (Versión 4.4 "KitKat"). El emulador desplegado para las pruebas de desarrollo utilizará por tanto Android 4.4 .

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

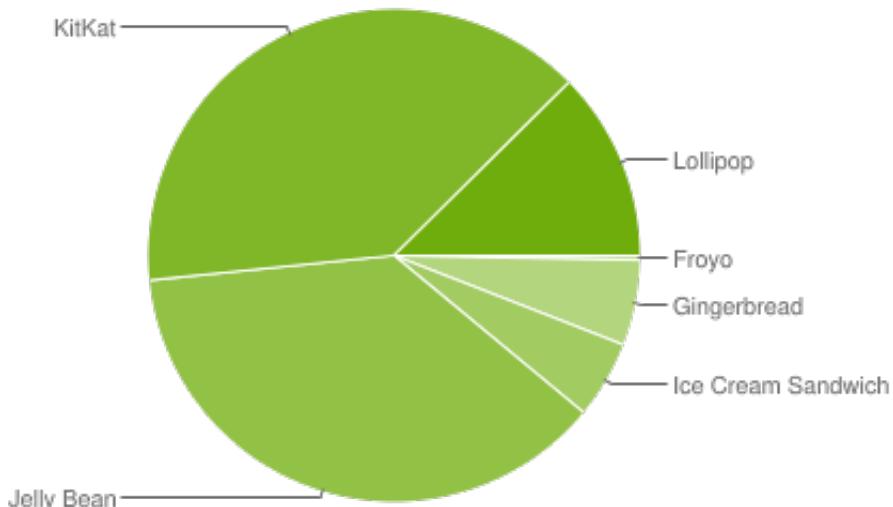


Figura 3.1: Distribución Actual de Versiones Android (Fuente: Google)

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 [?]** por cuestiones de eficiencia. Para evitar esta limitación, durante el proceso de construcción el SDK de Android utiliza, entre otras, una herramienta llamada **ProGuard [?]**. 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.

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.

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

3.2.2.2. Proceso de Construcción por Consola

Para construir la aplicación por consola de comandos, Android utiliza la herramienta Apache Ant [?] para automatizar el proceso de construcción [?]. 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 esta 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 [?]:

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

```
$ ant clean debug
```

Esto nos generará una aplicacion 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 3.2.2.1) 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, etcétera.) siendo lo más importante elegir un API (versión de Android) 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

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

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) [?] 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.



Figura 3.2: 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.

3.2.2.3. 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

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

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.

3.2.3. Migración: Identificación y Solución de Problemas

Resumen:

- Migrando Conexión HTTP a Android 3.2.3.1.
- Migrando Conexión WebSocket a Android 3.2.3.2.
- Migrando Logging a Android 3.2.3.3.
- Creando el Servicio Android 3.2.3.4.

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 WIAB que también 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 [?]. 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 estamos 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** [?], que representan las

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

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 [?] y WebSockets [?].

3.2.3.1. 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** [?] para controlar los privilegios de cada aplicación. Estos permisos se declaran en el **Manifiesto** de la aplicación [?], 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 fácil solución pues el **emulador de Android define unas direcciones IP de red especiales** [?] 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: `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 3.2.2.3) estudiando el LogCat [?] (Ver Figura 3.3) 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.

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

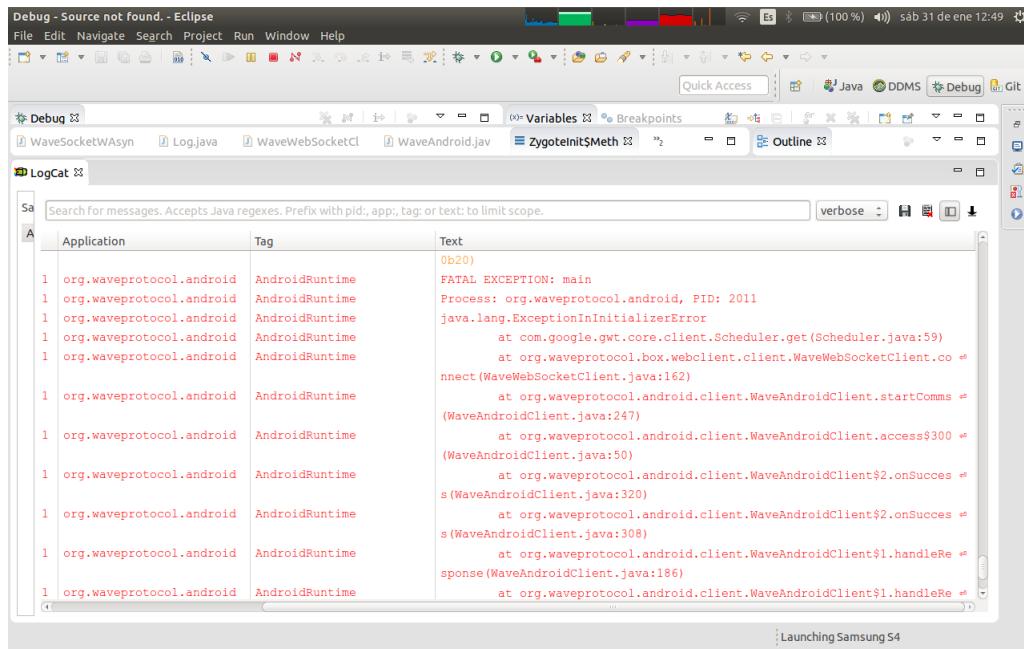


Figura 3.3: Ejemplo de Traza de Error en Logcat

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** [?], incluida en el SDK de Android desde sus primeras versiones. Sin embargo, Google recomienda [?] a partir del API 10 (Android 2.3 "Gingerbread") utilizar la **librería HttpURLConnection**[?], también incluida en el API. Por tanto esta última es la que elegimos para la migración.

Se puede ver un esquema simplificado de la nueva estructura del login HTTP en la Sección A.1.1 del Apéndice.

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**) [?]:

- 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

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

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 [?]. 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.

Se puede ver un esquema del AsyncTask encargado del Login en la Sección A.1.2 del Apéndice.

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.**

3.2.3.2. Conexión WebSocket

Para realizar una conexión con el servidor Wave es necesaria una conexión mediante WebSockets[?], 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 asíncronía es necesaria ya que para mantener la consistencia en tiempo real 4.1.3.2 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.

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

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 [?], 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 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. Así mismo nuestro Websccket 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 WebSoccket 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 WebSoccket en Android sobre Atmosphere y que porporcione las mismas funciones básicas descritas en el párrafo anterior.

La solución encontrada fue utilizar wAsync[?], 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.

Se puede ver un ejemplo sencillo de utilización de wAsync en la Sección A.1.3 del Apéndice.

Como la arquitectura Wave utilizaba el patrón factoria, fue necesario sustituir la clase de WebSocket GWT por una de nueva creación llamada **WaveSocketWAsync.java[?]** que implementa los métodos de creación y configuración de un Websocket y de callback antes descritos. Asimismo se modificó la clase

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

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 Atmosphere, tal y como recomienda hacer wAsync en su wiki [?]. Para ello basta con añadir al proyecto las librerías oportunas (Ver Tabla de dependencias 3.2) y configurar el cliente segun lo descrito en dicha wiki:

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

3.2.3.3. 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 3. METODOLOGÍA DEL PROYECTO

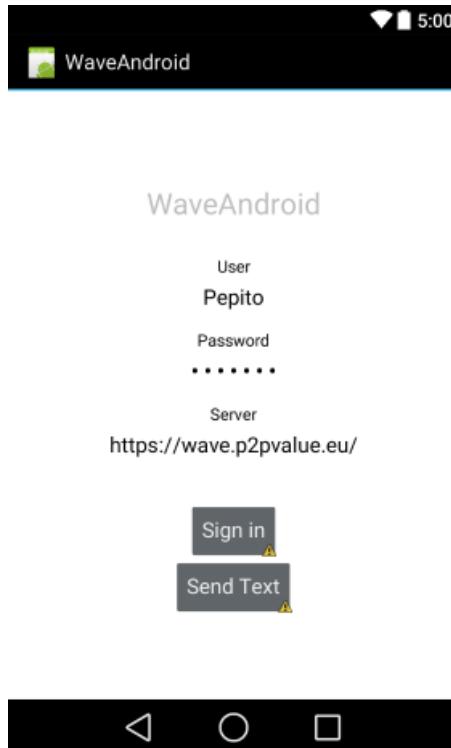


Figura 3.4: Pantalla de Login de WaveAndroid

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[?].

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.**

3.2.3.4. 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 3. 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[?] 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 [?] 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 3.2.3.1 y 3.2.3.2) 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[?].

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 3. 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 3.2.3.1) 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.

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 WebSocket responder a eventos del servidor (recordemos el funcionamiento de WAsync descrito en la Sección 3.2.3.2) 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:

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

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

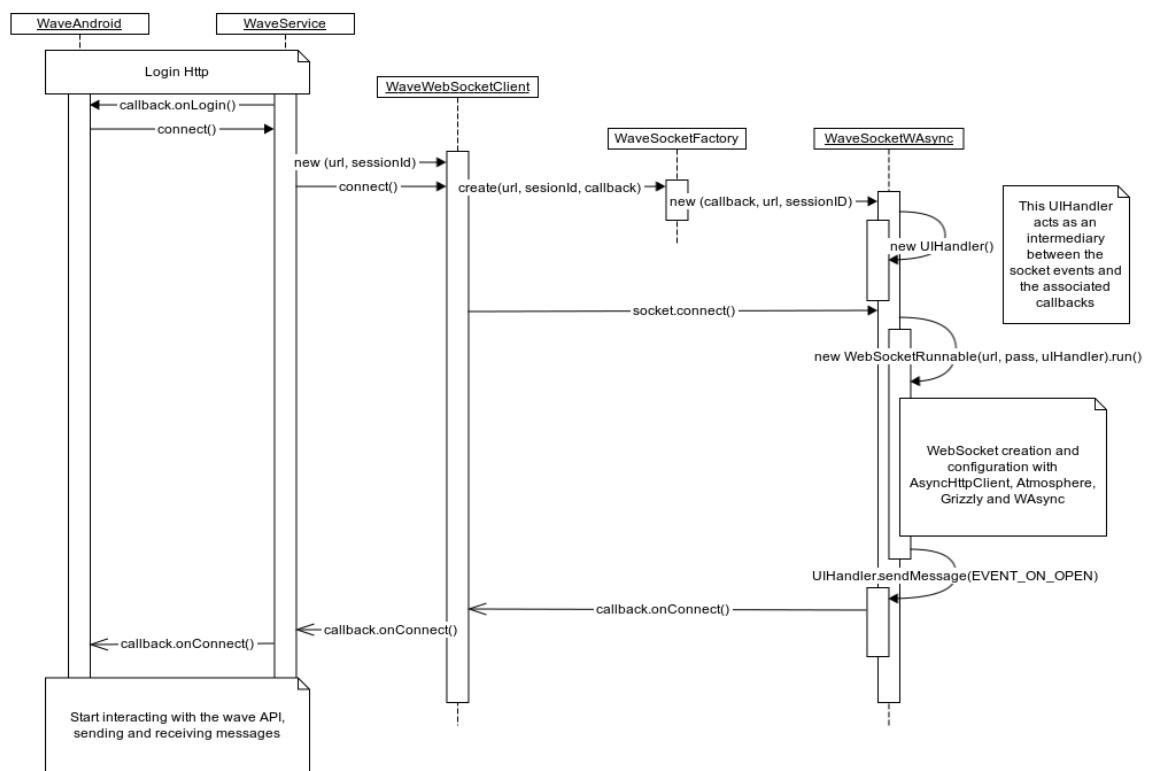


Figura 3.5: Proceso de conexión WebSocket con Servicio

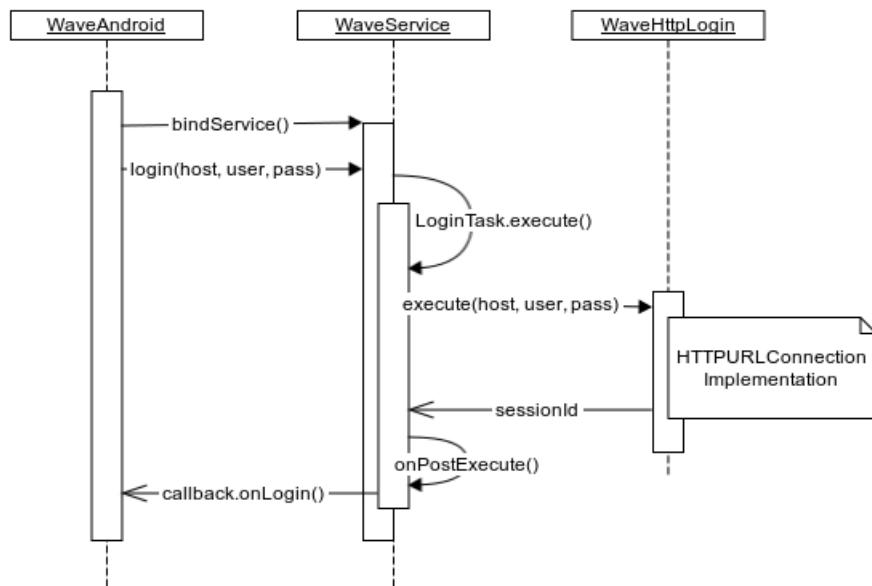


Figura 3.6: Proceso de conexión Http con Servicio

3.2.4. 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:

3.2.5. 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[?].**

Solo restaba poner el API de SwellRT en una capa superior para poder acceder y trabajar a nivel de wave con el modelo de datos de SwellRT. De esto se encargó el director de proyecto Pablo, desarrollador del proyecto inicial Web de SwellRT. Con este API trabajaremos en la siguiente parte del proyecto para crear una aplicación Android que haga uso de las funcionalidades de SwellRT. El API se encuentra en el GitHub de SwellRT.

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

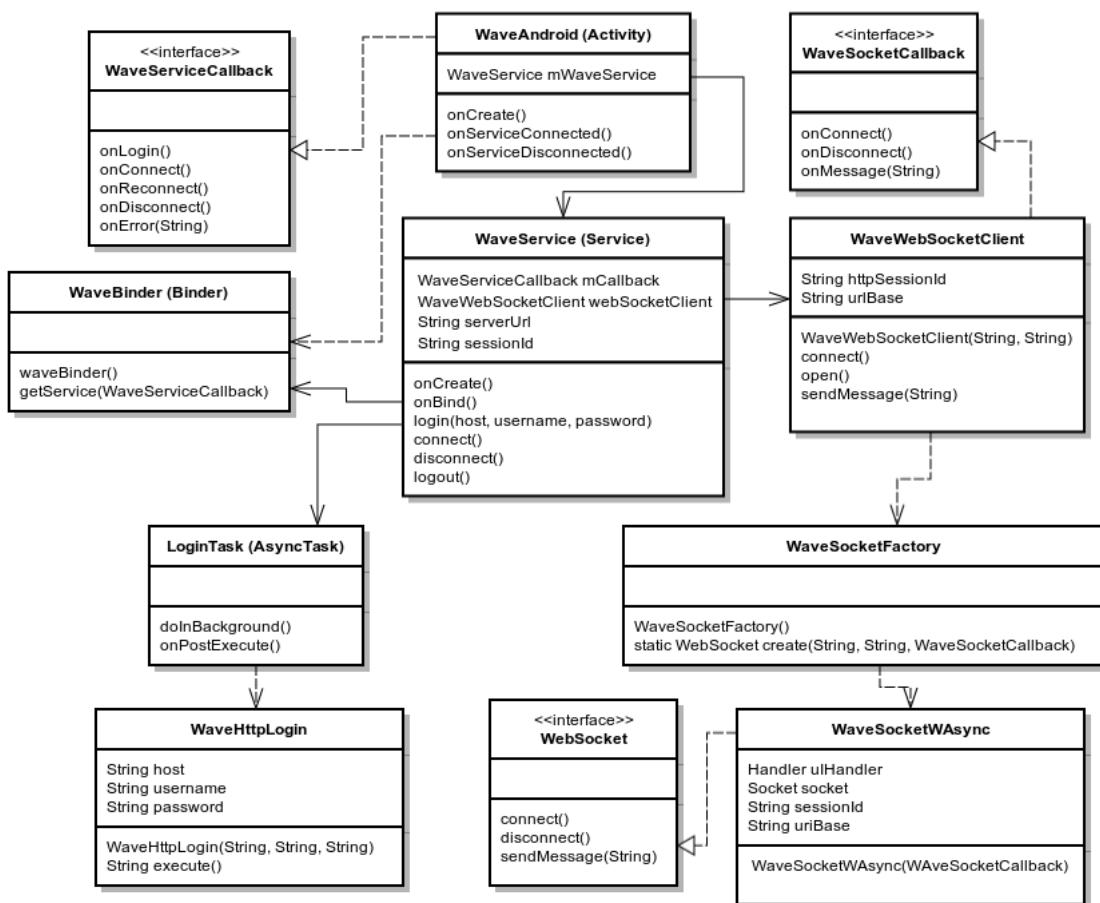


Figura 3.7: Esquema de Clases de SwellRT-Android con Servicio

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

Nombre	Versión	Descripción
WAsync [?]	1.4.3	Librería que contiene el cliente WebSockets/HTTP para comunicación asíncrona.
AsyncHttpClient [?]	1.8.14	Librería que permite a aplicaciones Java ejecutar de forma sencilla peticiones HTTP y procesar de forma asíncrona la respuesta HTTP recibida.
Grizzly-Framework [?]	2.3.18	Framework del núcleo para aplicaciones Grizzly que proporciona conexiones TCP/UDP, gestión de memoria, servicios/buffers, eventos NIO de bucle/cadenas de filtro/filtros
Grizzly-Http [?]		Framework HTTP que contiene la lógica para el tratamiento de mensajes HTTP en el servidor y en el cliente.
Grizzly-WebSockets [?]		Websocket API que permite crear aplicaciones con WebSockets en el servidor y en el cliente.
slf4j-Android [?]	1.6.1	”Simple Logging Facade for Java”: Framework de <i>logging</i> compatible con Android.

Tabla 3.2: Dependencias de SwellRT-Android

3.3. Implementación de DemoCritics

En esta fase llevamos a cabo la implementación de un prototipo final de la aplicación basado en el estudio de los resultados de la fase de diseño previa. Además se diseñó e implementó también el servidor que alojaría la base de datos y se comunicaría con la aplicación.

3.3.1. Metodología

Durante el desarrollo del código del sistema hemos utilizado las metodologías que se describen a continuación.

3.3.1.1. Control de versiones

Para mantener una copia de las versiones locales, hemos utilizado GIT como herramienta de control de versiones. Realizando *commits* y subiendo los cambios de cada versión en un repositorio público alojado en GitHub.

CAPÍTULO 3. METODOLOGÍA DEL PROYECTO

3.3.1.2. GitHub

Para compartir el código del sistema y visualizar los cambios de cada commit, utilizamos una organización pública donde se almacenan todos los repositorios de sus distintas partes. La organización está disponible en el siguiente enlace: <https://github.com/Zorbel>.

3.3.1.3. Reparto de tareas

Durante la implementación del proyecto dividimos las tareas en dos grupos principales. Por un lado teníamos la parte de *back-end*, formada por la API Rest desarrollada en Laravel y la base de datos. Mientras que por otro lado se encontraba la parte de *front-end* formada por la aplicación en Android. De cada una de estas partes y debido a la experiencia previa con las tecnologías involucradas se encargó uno de nosotros. De esta manera Jaime trabajó más en el desarrollo del back-end y Javier en el del front-end. No obstante ambos revisábamos el código del otro para verificar su correcto funcionamiento. El desarrollo inicial en Wave/SwellRT y su implementación final en la aplicación se produjo en paralelo por ambos componentes del grupo.

3.3.1.4. Revisiones de código

Cada vez que completábamos una funcionalidad la subíamos a *GitHub* y se realizaba una pequeña revisión de código para verificar cómo se había implementado el objetivo a desarrollar. De esta forma evitábamos malinterpretar algunos aspectos que durante la implementación pueden diferir de su planteamiento inicial. También recibimos revisiones de código externas por parte de los directores del Trabajo de Fin de Grado, que aportaron una visión más eficiente y organizada del código desarrollado.

3.3.1.5. Evaluaciones de usabilidad

Recibimos una evaluación de usabilidad externa por parte de un director del Trabajo de Fin de Grado, que valoraba aspectos a mejorar de la Intefaz de Usuario (UI). Esta evaluación nos sirvió para modificar algunos aspectos de representación gráfica que mejoraríaían la interpretación de los elementos por parte del usuario.

Capítulo 4

Tecnologías del Proyecto

En las siguientes secciones se explicarán brevemente las tecnologías y herramientas que se han utilizado para llevar a cabo este proyecto. Concretamente se hablará de las tecnologías utilizadas durante la Migración del cliente web de SwellRT (Wave) a Android (Ver Sección 3.2) y durante el desarrollo de la aplicación Android que hará uso del resultado de dicha Migración.

En el caso de Wave, al tratarse de una tecnología bastante reciente, se explicará más detalladamente en qué consiste esta tecnología, pero para el resto de casos se hará una pequeña reseña sobre en qué consiste la tecnología y sus usos. En todos los casos se intentará también explicar de qué manera y para qué se ha utilizado dentro de este proyecto.

Para hacerse una idea general de las tecnologías utilizadas se recomienda ver el resumen que se muestra en la Tabla 4.1.



Figura 4.1: Nube de Tecnologías utilizadas en el Proyecto

CAPÍTULO 4. TECNOLOGÍAS DEL PROYECTO

Tipo	Nombre	Uso	Sección
Framework	SwellRT	Framework que permite desarrollar aplicaciones con la tecnología y la infraestructura federada de Wave a través de un API JavaScript y Android.	4.1.4
	Android	Plataforma para la Migración y el Desarrollo de la app.	4.2.2
	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
Servidor	Wave In A Box (WIAB)	Servidor que aloja las Waves. Migración y Desarrollo de la App.	??
	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
Formato de Datos	JSON	Formato para intercambio de información con Servidor Wave y REST.	4.2.4
	XML	Formato para definición de datos estructurados (y de interfaces en Android).	4.2.3
Protocolo	Wave	Protocolo de intercambio de Waves basado en XMPP y 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.	4.2.1
	Sublime Text	Editor de texto y de código fuente, utilizado para desarrollar el Service REST.	4.2.11

Tabla 4.1: Tecnologías usadas en el Proyecto

4.1. Tecnologías de Wave

4.1.1. Google Wave

Ideado y presentado en 2009 por ingenieros de Google [?], Wave es a la vez un protocolo de comunicaciones basado en XMPP [?] y una plataforma web de código libre, que permiten a sus usuarios comunicarse y colaborar entre sí en tiempo real (Ver sección 4.1.3.2) y de forma federada (Ver sección 4.1.3.1) 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 [?] 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 [?] y a su red social Google + [?]. 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 [?] como software de código libre bajo licencia Apache [?]. 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:

4.1.3.1. Federación

El Protocolo Wave [?] fue desarrollado para utilizar un modelo federado [?] [?] de comunicación basado en la tecnología XMPP [?] [?]. Se trata por tanto

de un modelo descentralizado en el que cualquiera de los participantes en la edición del documento (o "conversación") es libre de actuar tanto como servidor como cliente sin que ello afecte a su participación en el documento. 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.

4.1.3.2. Consistencia en tiempo real

El Protocolo Wave [?] utiliza la tecnología de Transformaciones Operacionales (OT) [?] 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. Esto se hace sin pérdida de información y garantizando que, independientemente del orden en el que se produjeron los cambios, el estado final del documento será el mismo para todos y por ende verán lo mismo. [?].

4.1.3.3. 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: SwellRT

Como parte del proyecto europeo de investigación P2Pvalue [?] se ha desarrollado SwellRT (Swell Real Time), un fork de Wave In a Box (Ver sección 2.1.2.1) 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

CAPÍTULO 4. TECNOLOGÍAS DEL PROYECTO

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 [?], 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 [?]. Se pretende con esto que SwellRT haga uso de las funcionalidades nativas de Android.

4.1.5. Eclipse

Eclipse [?] es un Entorno de Desarrollo Integrado (IDE) open-source (bajo licencia Eclipse Public License) desarrollado por la Eclipse Foundation. Lanzado en 2003 alcanza actualmente su versión 4.4, también denominada Eclipse "Luna". Se trata de uno de los IDEs más utilizados que proporciona herramientas que permiten el desarrollo de código para múltiples lenguajes y tecnologías. Posee además la capacidad de extender sus capacidades gracias al uso de plug-ins que se pueden descargar desde su marketplace. Es el caso por ejemplo del plugin ADT (Android Developer Tools), que permite utilizar el SDK de Android para desarrollar para esta plataforma utilizando Eclipse como IDE. Hasta finales del año 2014 Eclipse + ADT era el IDE recomendado por Google para el desarrollo en su plataforma móvil, aunque recientemente ha pasado a utilizar su propio IDE llamado Android Studio (Ver Sección 4.2.1)

En este proyecto usaremos Eclipse con el plugin ADT como IDE para estudiar el código del cliente web de SwellRT y llevar a cabo una migración a Android que permita hacer uso de las características nativas de esta plataforma.

4.1.6. Java

Java [?] es un lenguaje de programación de propósito general y orientado objetos. Fue desarrollado en 1995 por Sun Microsystems, actualmente parte de Oracle, y se distribuye en forma de JDK (Java Development Kit), cuya versión oficial privativa alcanza hoy en día la 8. Existe no obstante una versión de código libre (bajo licencia GPLv2) llamada Open JDK [?] disponible actualmente en su versión 7. Java tiene como ventaja su independencia del

CAPÍTULO 4. TECNOLOGÍAS DEL PROYECTO

hardware, ya que cualquier código compilado en java se traduce a un formato bytecode que se ejecuta en una Máquina Virtual Java (JVM) que es independiente del hardware que haya por debajo.

En este proyecto usaremos Java para la Migración del cliente SwellRT a Android, pues dicho cliente (y el que desarrollo Google inicialmente) esta escrito en su mayor parte en este lenguaje de programación. Lo utilizaremos también para el desarrollo de la app, ya que el SDK de Android se basa en Java para desarrollar código para dicha plataforma.

4.1.7. GWT

GWT [?] (Google Web Toolkit) es un framework de código libre (bajo licencia Apache 2.0) que facilita el desarrollo de aplicaciones web basadas en AJAX y JavaScript. Concretamente el API de GWT permite desarrollar código en Java que posteriormente será traducido a JavaScript. Fue creado por Google en 2006 y actualmente es un proyecto independiente open-source que alcanza ya su versión 2.7

En este proyecto utilizaremos GWT durante la Migración de Wave a Android. ya que una parte significativa del cliente web de SwellRT está desarrollada con esta tecnología y es necesario estudiar el código para sustituirla por código nativo de Android, ya que la plataforma móvil de Google no es compatible con GWT.

4.1.8. JavaScript

JavaScript [?] (JS) es un lenguaje de programación dinámico y débilmente tipado utilizado a menudo para la programación de aplicaciones web del lado del cliente (navegador web), aunque también existen implementaciones para el lado del servidor (como Node.js). Originalmente desarrollado en 1995 por la extinta Netscape Communications, su versión actual es la 1.8.5.

En este proyecto se utilizará JavaScript para estudiar la implementación del cliente web de SwellRT hecha con GWT y JavaScript que deberemos migrar a código nativo de Android, pues la plataforma móvil de Google no es compatible de forma nativa con JavaScript.

4.1.9. HTTP

HTTP [?] (HyperText Transfer Protocol) es el protocolo de capa de aplicación más utilizado para establecer comunicaciones en la World Wide Web. Fue desarrollado en 1996 conjuntamente por la World Wide Web Consortium (W3C) y el Internet Engineering Taskforce (IETF), aunque la definición de su versión más actual (1.1) se encuentra especificada en la serie de RFCs (Request For Comments) 7230 [?]. Se trata de un protocolo orientado a transacciones que siguen un esquema de petición-respuesta entre cliente y servidor.

En este proyecto utilizaremos el protocolo HTTP para establecer las conexiones con ambos servidores: el servidor SwellRT que aloja las Waves y el servidor Openshift que aloja el Service REST y la Base de Datos de la aplicación. Concretamente con este protocolo realizaremos el proceso de login en Wave y las peticiones al Service REST en la aplicación Android.

4.1.10. WebSocket

WebSocket [?] es un protocolo que permite establecer comunicaciones bidireccionales (de cliente a servidor y viceversa) y full-dúplex (de forma simultánea en ambos sentidos) en la red. Permite utilizar la tecnología de los sockets TCP en la capa de aplicación. Se trata de un protocolo estandarizado por la IETF en 2011 en el RFC 6455 [?].

En este proyecto utilizaremos esta tecnología para establecer un canal de comunicación basado en sockets con el servidor Wave de SwellRT, pues la característica de bidireccionalidad full-dúplex es necesaria para aprovechar la potencialidad de consistencia en tiempo real de Wave. Concretamente en la Migración deberemos sustituir la implementación actual de WebSocket en GWT por otra que sea compatible de forma nativa con Android.

4.1.11. Git

Git [?] es un Sistema de Control de Versiones (VCS) distribuido y open-source (bajo licencia pública GNU) diseñado para gestionar las distintas versiones de una aplicación independientemente del tamaño de la aplicación y del número de personas que trabajan en ella. Fue creado por Linus Torvalds en 2005 para trabajar en el desarrollo del kernel de Linux, aunque su versión actual (2.4.2) es una de las herramientas de control de versiones mas utilizada para

gestionar proyectos software. Su interfaz es por consola de comandos.

En este proyecto usaremos Git por consola de comandos como Sistema de Control de Versiones de todo el software generado, tanto de la Migración de Wave/SwellRT a Android como del desarrollo de la app y el Service REST del que hace uso. Incluso las distintas versiones del código látex de esta Memoria estará gestionado con Git.

4.1.12. GitHub

GitHub [?] es una plataforma web lanzada en 2008 que permite alojar de forma gratuita y en sitios llamados repositorios, proyectos software que utilicen Git como Sistema de Control de Versiones. Los repositorios gratuitos son públicos y accesibles por todo el mundo, de manera que cualquiera puede participar en la elaboración de código, aunque existe la opción de pagar por tener acceso a repositorios privados. GitHub proporciona una interfaz web que además de gestionar los repositorios permite alojar wikis, páginas web, gestión de tareas pendientes (issues) y control de acceso entre otras funcionalidades.

Para este proyecto utilizaremos GitHub como plataforma para alojar todo el código open-source desarrollado durante el proyecto: la Migración de SwellRT a Android (cuyo cliente web antiguo también está disponible en GitHub [?]), el desarrollo de la aplicación Android, el Service REST y la Memoria. Todo esto esta disponible en forma de repositorios bajo la organización llamada "Zorbel" en la siguiente URL:

[HTTPs://github.com/Zorbel](https://github.com/Zorbel)

4.2. Tecnologías de la Aplicación Android

4.2.1. Android Studio

Android Studio [?] es el Entorno de Desarrollo Integrado (IDE), basado en IntelliJ IDEA [?], oficial que Google proporciona para desarrollar aplicaciones para Android. Fue anunciado en 2013 y en diciembre de 2014 dejó su fase de beta y pasó a ser el IDE de referencia para el desarrollo en Android, dejando atrás el antiguo IDE de Eclipse junto al plug-in ADT. Android Studio integra en un solo lugar todas las herramientas necesarias para desarrollar en esta plataforma como un gestor de versiones de Android (SDK Manager),

CAPÍTULO 4. TECNOLOGÍAS DEL PROYECTO

un gestor de emuladores virtuales (AVD Manager) o una herramienta de Debug (DDMS) entre otras. Permite asimismo trabajar tanto con código de aplicación como con la interfaz gráfica (XML), la cual podremos previsualizar en el propio IDE.

En este proyecto, y ya que desde principio de año es el IDE de referencia, utilizaremos Android Studio para el desarrollo de la aplicación Android que hará uso de las funcionalidades de Wave.

4.2.2. Android

Android [?] es el Sistema Operativo de código libre (bajo licencia Apache 2.0) para dispositivos móviles de Google basado en el kernel de Linux. Lanzado en 2007, actualmente su API alcanza ya la versión 21 [?], también denominada Android 5.0 "Lollipop". Para desarrollar en esta plataforma basta con descargarse su SDK [?], accesible entonces desde consola de comandos, aunque siempre resulta más cómodo utilizar un Entorno de Desarrollo como Eclipse o Android Studio (Ver Secciones 4.1.5 y 4.2.1).

En este proyecto utilizaremos Android como plataforma móvil sobre la que llevar a cabo el desarrollo de la migración del cliente de Wave/SwellRT y para el desarrollo de una aplicación que haga uso de las funcionalidades y características de Wave. Concretamente utilizaremos el API 21 (Android 5.0 Lollipop) para tener acceso a la interfaz gráfica de diseño plano "Material Design" que incluye.

4.2.3. XML

XML [?] (eXtensible Markup Language) es un formato de definición, almacenamiento e intercambio de datos de forma estructurada. Fue definido en 1996 por el W3C y actualmente existe una versión 1.1 (2008). Se trata de un lenguaje de marcado que define dicho formato estructurado mediante marcas o "etiquetas" que aportan información acerca del texto que rodean. En el caso particular de Android, esta plataforma define la interfaz gráfica de sus pantallas mediante documentos XML llamados Layouts [?].

En este proyecto se utilizará XML para definir la interfaz gráfica de las pantallas que conforman la aplicación Android.

4.2.4. JSON

JSON [?] (JavaScript Object Notation) es un estándar para intercambio de datos de forma simple y ligera mediante pares clave-valor. Originalmente derivaba de un subconjunto de datos de JavaScript pero actualmente se encuentra definido en el RFC 7159 y el ECMA-404 y es independiente de lenguaje que se utilice para interpretar ("parsear") los datos que contiene. En JSON se puede estructurar estos datos de dos formas: en objetos (que contienen pares clave-valor) y en arrays (que contienen objetos).

En este proyecto utilizaremos JSON para intercambiar datos con el servidor de Wave/SwellRT y con el Service REST de la aplicación. En el caso de SwellRT no trabajaremos directamente a nivel de JSON ya que su API abstracta el formato de intercambio de datos. Sin embargo, en la aplicación sí que trabajaremos directamente con esta tecnología ya que el Service REST responde a las peticiones de la aplicación en forma de mensajes JSON que la propia aplicación debe tambien "parsear" para tratarlos.

4.2.5. SQL

SQL [?] (Structured Query Language) es un lenguaje diseñado específicamente para interactuar con Bases de Datos Relacionales pudiendo definir la estructura de los datos y manipularlos. Desarrollado en 1986, actualmente está estandarizado por la International Standards Organization (ISO) en el estándar ISO/IEC 9075 SQL [?]. Este lenguaje permite mediante la construcción de "consultas" crear tablas, acceder a sus datos y modificarlos entre otras cosas.

En este proyecto utilizaremos SQL en el Service REST para construir las consultas a Base de Datos necesarias para devolver al cliente de la aplicación Android los datos que solicite o pida cambiar.

4.2.6. MySQL

MySQL [?] es un Sistema Gestor de Base de Datos (SGBD) Relacionales que permite el almacenamiento, creación y modificación de dicho tipo de Bases de Datos. Desarrollado por Oracle, su versión actual (5.7.4) se distribuye tanto en forma open-source (bajo licencia GPL) o de uso comercial.

En este proyecto usaremos MySQL como SGBD para almacenar la Base de Datos de nuestra aplicación Android.

4.2.7. PhpMyAdmin

PhpMyAdmin [?] es una herramienta de código libre (bajo licencia pública GNU) escrita en PHP y que proporciona un sistema de administración de SGBD MySQL a través de una interfaz web. Fue desarrollado en 1998 y actualmente la versión 4.4.8 es desarrollada y mantenida por The PhpMyAdmin Project. Posee una interfaz sencilla que permite administrar la Base de Datos mediante las operaciones básicas de creación, modificación, eliminación de tablas, creación de consultas SQL y gestión de permisos y usuarios entre otros.

En este proyecto usaremos PhpMyAdmin como sistema de administración de la Base de Datos MySQL que contiene los datos de la aplicación Android.

4.2.8. PHP

PHP [?] es un lenguaje de propósito general del lado del servidor diseñado originalmente para crear aplicaciones web que generaran contenido dinámico. Diseñado en 1996 por Rasmus Lerdorf, su versión actual 5.6.7 es de código libre y se distribuye bajo licencia PHP. Tiene la ventaja de poder ser fácilmente incorporado dentro de los documentos HTML.

Para este proyecto utilizaremos el lenguaje PHP para programar el comportamiento del Service REST que hace de intermediario entre las peticiones de la aplicación Android y la Base de Datos MySQL.

4.2.9. Laravel 5

Laravel [?] es un framework que permite desarrollar aplicaciones y servicios web con PHP 5. Fue desarrollado por Taylor Otwell en 2011 y su versión actual 5.0 se distribuye en forma de código abierto (bajo licencia MIT) en su propio repositorio público en GitHub [?]. Su funcionalidad es extensible mediante módulos.

En este proyecto utilizaremos Laravel 5 para construir en PHP el Service REST que hará de intermediario entre las peticiones HTTP del cliente Android y la Base de Datos de la aplicación.

La elección de Laravel como framework PHP se debió a su flexibilidad, la facilidad para programar la aplicación y el gran soporte que tiene de la comunidad. Además, es uno de los frameworks más populares actualmente en uso.

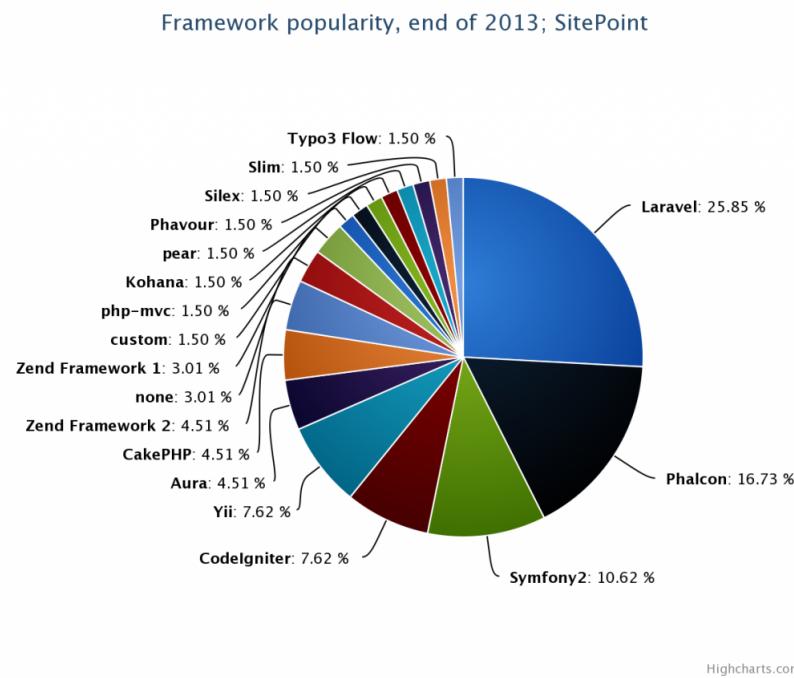


Figura 4.2: Frameworks PHP más populares. Fuente: SitePoint

4.2.10. OpenShift

OpenShift [?] es una plataforma de computación en la nube que ofrece alojar servicios de forma gratuita en sus servidores mediante un modelo de arquitectura Software as a Service (SaaS). Disponible desde 2011, se distribuye bajo licencia Apache 2.0. Dispone también de planes de pago que aumentan las prestaciones del servidor.

En este proyecto usaremos OpenShift como servidor web para alojar los servicios de los que hace uso nuestra aplicación Android: el Service REST y la Base de Datos. Concretamente nuestro servidor esta accesible desde la siguiente URL:

<HTTPs://apptfg-servicerest.rhcloud.com/>

4.2.11. Sublime Text

Sublime Text [?] es un editor de texto gratuito multiplataforma muy versátil que proporciona características de atajos de teclado y resaltado de código para diferentes lenguajes de programación muy útiles cuando se desarrolla

CAPÍTULO 4. TECNOLOGÍAS DEL PROYECTO

sin un IDE. Fue desarrollado en 2008 por Jon Skinner y actualmente se encuentra en su versión 2.0.2.

En este proyecto utilizaremos Sublime como editor de texto sobre todo para configurar y programar el Service REST en PHP, pues no vemos necesario utilizar un IDE específico para ello.

4.2.12. POP: Prototyping On Paper

POP [?] (Prototyping On Paper) es una aplicación con versiones web y para dispositivos móviles que permite elaborar en pocos pasos prototipos (mockups) interactivos basados en fotos de prototipos realizados en papel. De esta forma se puede elaborar un prototipo de interfaz gráfica de bajo coste con la que el usuario pueda interactuar.

En este proyecto se utilizará POP durante la fase de diseño de la aplicación Android para elaborar mockups sencillos e interactivos que podamos enseñar a la gente y que nos ayuden a refinar el diseño de la interfaz gráfica de la aplicación. Tiene además la ventaja de que se puede enseñar en la aplicación móvil de POP para simular el aspecto final que tendría la aplicación y que el usuario se haga una mejor idea de lo que pretendemos diseñar.

Capítulo 5

Migración de SwellRT a Android

Capítulo 6

Diseño de la Aplicación

6.1. Introducción

Tras haber adaptado (Ver Sección 3.2) la tecnología de Wave/SwellRT a Android que soportaría el núcleo de nuestra aplicación, era hora de decidir qué aplicación le íbamos a dar de cara a desarrollar una aplicación Android que hiciera uso de ello. Era importante tener en cuenta las características que nos ofrecía Wave:

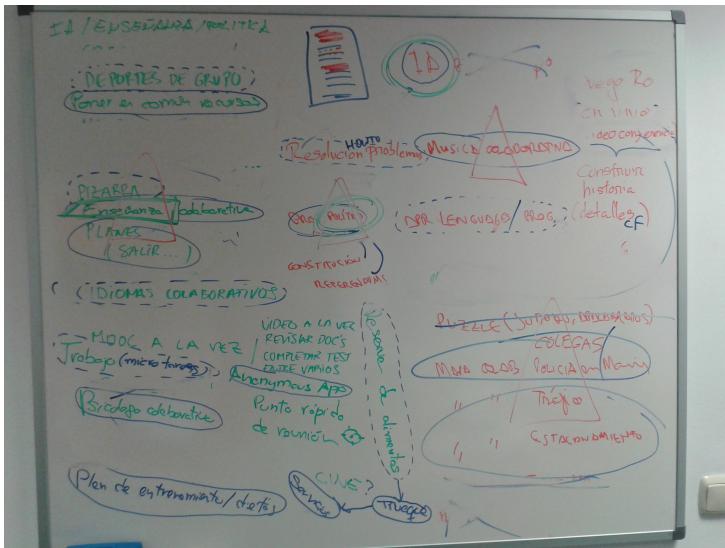
- Edición colaborativa.
- Consistencia en Tiempo Real.

6.2. Brainstorming de ideas

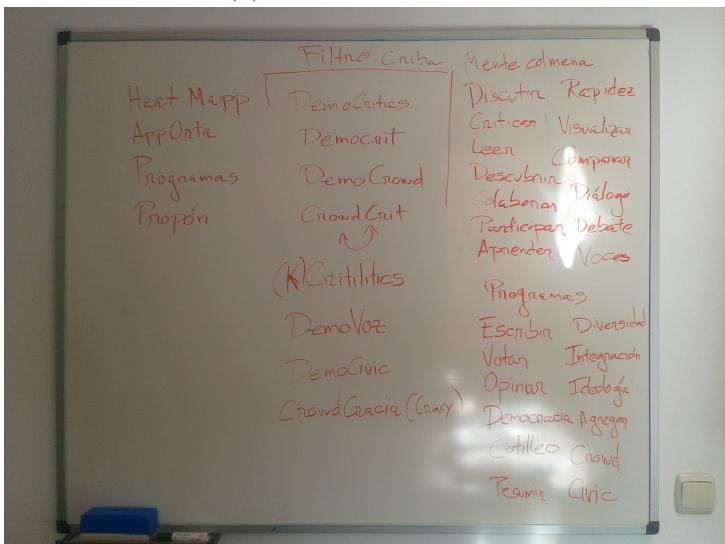
Un *brainstorming* [?] (o lluvia de ideas en español) es una técnica de trabajo en grupo que persigue que todos sus integrantes se junten para generar ideas originales en un ambiente distendido. El objetivo es primar la cantidad por encima de la calidad de éstas para después de la sesión realizar un proceso de selección de las más interesantes. De esta manera pueden salir ideas que a priori podrían resultar absurdas pero que otros integrantes del grupo pueden aprovechar para que surjan otras nuevas.

Después de considerar posibles ideas de implementaciones que podrían hacer uso de estas características, decidimos realizar una sesión de *brainstorming* junto a nuestros directores de proyecto 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 6. DISEÑO DE LA APLICACIÓN



(a) Idea de Aplicación



(b) Posibles Nombres

Figura 6.1: Capturas del brainstorming

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

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

ras 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 actual para desarrollar una aplicación 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.

Por otro lado, queríamos elegir un nombre para la aplicación que fuera a la vez atractivo y significativo de la participación ciudadana que representa. Para esto hicimos también una sesión de *brainstorming* con varias personas, de la cual sacamos varias posibilidades por el nombre. Tras someter a votación estos nombres nos quedamos con el que más gustó a todos: **DemoCritics**.

6.3. Diseño Guiado Por Objetivos (DGO)

Para elaborar el diseño de la aplicación, nos hemos basado en la metodología del Diseño Guiado por Objetivos (**DGO** o *Goal-Directed Design*), que implementa el proceso de la Ingeniería de la Usabilidad propuesto por Alan Cooper [?]. Este proceso constará de las siguientes fases:

1. Investigación

Esta fase consistirá en la realización de estudios para obtener datos cualitativos sobre los usuarios y/o reales de la aplicación y cuáles son sus necesidades. Se realizarán tareas para comprender a los usuarios, saber sus inquietudes y lograr empatía. A lo largo de esta fase se irán identificando patrones de comportamiento que sugerirán los objetivos y motivaciones del usuario. Por último se realizarán estudios de mercado, revisiones y auditorías que ayudarán al diseñador a comprender

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

el dominio, el modelo y las restricciones técnicas que el sistema debe cumplir.

2. Modelado

A lo largo de esta fase se utilizarán los datos provenientes de la fase previa para crear los modelos del dominio y los usuarios. En esta parte se crearán las *personas*, aquetipos de usuarios que contienen información sobre objetivos, motivaciones y comportamientos de los usuarios con el sistema. El resultado final de esta fase serán los tipos de *persona* que representarán a los usuarios del sistema, y que más adelante serán utilizados fases para proporcionar algún tipo de *feedback*.

3. Definición de Requisitos

Durante esta fase se utilizarán las personas y los datos de la fase anterior, para crear los *escenarios* de contexto e identificar los requisitos o necesidades del usuario. Estos requisitos serán definidos en tres componentes: objeto, acciones y contexto. También se definirán requisitos relacionados con el negocio, la aplicación, requisitos técnicos, etcétera.

4. Definición del framework de Diseño

En esta fase se creará el concepto general del sistema, definiendo los comportamientos y diseño visual. Identificaremos el **framework de interacción**, como un concepto de diseño estable que define la estructura del sistema a partir de patrones y principios de diseño. Por último, se definirá el *framework visual*, como el aspecto visual de la aplicación (diseño, tipografía, colores, iconos, etcétera).

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.

6.3.1. Investigación

6.3.1.1. Intención Inicial: Prototipo básico

Vivimos en una época donde la política parece estar de moda. Esto puede ser debido al cabreo general que muestra la ciudadanía frente a gobiernos conservadores, situaciones de austeridad provocada por la crisis económica,

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

el nacimiento de nuevas fuerzas políticas, . . . , pero sobre todo las numerosas citas electorales a las que seremos citados en 2015. Por tanto, el proyecto podría ponerse a prueba en un escenario real durante la campaña de las elecciones generales previsiblemente convocadas durante el último trimestre del 2015.

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 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.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

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 6.3.5) que mostraba programas políticos estructurados y permitía navegar a nivel de Sección por ellos para leer y emitir opiniones (likes, dislikes, comentarios, etcétera.).

6.3.1.2. Hipótesis de personas

Para identificar a los usuarios objetivo, tenemos que encontrar a miembros representativos de esos usuarios y animarlos a que participen en nuestra investigación. Para ello reuniremos una serie de características básicas que el usuario deberá cumplir para poder aportarnos los objetivos funcionales de la aplicación. Estas son algunas de las caracaerísticas deseadas:

- **Edad:** Entre 16 y 65 años.
- **Sexo:** Indiferente.
- **Profesión:** Indiferente.
- **Aptitudes deseables:** Activismo social, conciencia política, trabajo colaborativo,
- **Habilidades técnicas:** Usuario con experiencia media en uso de aplicaciones móviles.

Simplificando el tipo de persona que queremos encontrar, lo dividiremos en dos tipos de persona. Por un lado buscaremos al activista social, activo en movimientos sociales, participación en portales con carácter social, etcétera. Mientras que por otro lado buscaremos una persona que muestre cierto

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

interés en el mundo de la política, pero que no participe en movimientos sociales. Finalmente pudimos contactar con dos miembros de Labodemo [?], una organización que se dedica al desarrollo de nuevas formas de participación ciudadana. Este perfil cubría en caso del activista social, mientras que para el tipo de persona entendido de la política pero no tan involucrado, escogimos a Javier de la Cueva [?].

El siguiente paso sería estudiar la viabilidad de esta aplicación entrevistando a las personas seleccionadas 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:

6.3.1.3. Entrevista con Labodemo

Tuvimos la oportunidad de mantener una conversación con dos miembros de Labodemo [?], 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 proba-

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

blemente acabariamos excluyendo temas. Así, serían los colectivos los que se encargaran de "tematizar.^{el} 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.

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.

6.3.1.4. Conclusión

Puntos positivos:

- Nueva forma de participación ciudadana de cara a las elecciones.
- Métodos alternativos para discutir propuestas, partidos, programas, etcétera.
- Ligar propuestas a programas concretos.

Puntos negativos:

- Visualizar un programa electoral en el móvil puede no resultar demasiado interés para los usuarios.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

- Desarrollar propuestas colaborativas en tiempo real no maniene una estabilidad en la aplicación.
- La aplicación debería desarrollarse en otras plataformas móviles y de escritorio.

Puntos a tener en cuenta:

- Dejar libertad a usuarios y colectivos creando sus propias categorías o hilos como "Espacios" que puedan elaborar según sus intereses y generar contenido para la aplicación.
- Desarrollar comparativas por secciones, temas o partidos. De tal forma que un usuario pueda visualizar las diferencias de aquellos temas que le preocupan.
- Contactar con colectivos, asociaciones y/o periodistas que anteriormente hayan elaborado comparativas entre programas políticos en puntos concretos

6.3.1.5. 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 [?]. 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,

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

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

6.3.1.6. Conclusión

Puntos positivos:

- Llevar los programas políticos de una forma más atractiva a la ciudadanía es algo esencial en la actualidad.
- Categorizar las secciones de los programas políticos para que se puedan explorar por temas.
- Incluir los recursos necesarios que hacen falta para llevar una propuesta a cabo.

Puntos negativos:

- Tratar de convertir la aplicación en un *foro* inconscientemente.
- Dejar libertad a la hora de crear categorías específicas o hilos puede generar confusión entre los usuarios.

Puntos a tener en cuenta:

- Desarrollar categorías semanales en función de las novedades o actualidad política.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

- Añadir la posibilidad de solicitar la ayuda de expertos sobre un tema para elaborar una propuesta.
- Crear una emeroteca de programas políticos para realizar análisis sobre el cumplimiento de los programas en legislaturas pasadas.
- Informar sobre la legislación actual cuando visualicemos una propuesta o sección de un programa que quiera mejorar o cambiar la legislación actual.

6.3.2. Modelado de Personas

Las *personas* son una herramienta de diseño y ayuda al diseño de la aplicación. El objetivo es centrar el diseño en este tipo de persona, para lograr los objetivos a la hora de utilizar un sistema. La *persona* será nuestro modelo, una descripción detallada de un individuo imaginario que representa a un grupo de usuarios a los que va destinada la aplicación. Es una representación ficticia pero desarrollada con gran detalle, que será fruto de los datos recogidos en la investigación previa de la fase anterior.

En esta fase definiremos el tipo de persona que interactuará con nuestra aplicación. Para ello hemos identificado dos tipos de personas primarias; un **activista social** y un **ciudadano** de a pie que forme parte del electorado.

- 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:

- 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.

6.3.3. Definición de personas

En las figuras 6.2 y 6.3 se exponen una serie de personas ficticias que podrían representar en la vida real los tipos de *persona* a la que va dirigida la aplicación.

6.3.4. Definición de Escenarios y Requisitos

En esta sección se definirán los posibles escenarios que puedan surgir en la aplicación. La idea es situarnos en un escenario real que pudiera ocurrir en cualquier momento a lo largo del día, para detallar la solución al problema. En cada uno se especificarán los requisitos necesarios para solventar el problema o los pasos a seguir para lograr el objetivo. Diferenciaremos los términos de **acción**, como la actividad inmediata que requiere la solución. El **objeto**, como el sujeto principal del escenario. Y por último definimos **contexto** reflejando el objetivo final del requisito.

Escenario I

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

Nombre: Lucía
Edad: 21 años.

Ocupación y actividades:

- Estudiante de *Administración y Dirección de Empresas* en la Universidad Rey Juan Carlos.
- Le preocupa la situación actual del estado del bienestar y la gestión de los recursos públicos.

Cita: "Los programas políticos deberían de tener mayor peso en la sociedad".



Actividad política:

Lucía es un estudiante universitario que será citado a votar en las elecciones generales que serán celebradas en dos semanas. Para estar al día de la actualidad política sigue las redes sociales, debates televisivos y otros medios.

Hay diversas opciones políticas para Lucía, pero ella aún no ha decidido su voto. Quiere explorar los proyectos de futuro que tienen las formaciones políticas para el país.

Figura 6.2: Estudiante universitario

Nombre: Carlos
Edad: 34 años.

Ocupación y actividades:

- Trabaja como profesor de educación primaria en un colegio público de Barcelona.
- Miembro de la *Plataforma de Afectados por la Hipoteca*.
- Participa en diversos movimientos sociales.

Cita: "Debemos de incentivar la participación de la gente común en la política".



Actividad política:

Carlos es un activista social que nunca se pierde la oportunidad en participar en todos los movimientos sociales que lo rodea. A menudo portales de participación ciudadana para organizarse con los miembros de su colectivo.

Para cambiar la sociedad Carlos piensa que las personas deberían involucrarse más en la política que desarrollen nuevas ideas y propuestas para el estado. Algunas personas le preguntan de qué forma pueden ayudar a generar nuevas propuestas e ideas.

Figura 6.3: Activista político

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

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:

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 qué 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. Visualizar (acción) el listado de propuestas (objeto), clasificados por la categoría seleccionada por el usuario (contexto).
2. Mostrar (acción) la propuesta (objeto), seleccionada por el usuario para que pueda puntuarla y/o comentarla (contexto).

Escenario III

Lara es una profesora de un colegio de la Comunidad de Madrid. Se acercan las vacaciones de verano y muchos niños se quedarán sin acceso a comedor. Lara está planteándose cómo elaborar una propuesta ciudadana para abrir los colegios en horario no lectivo y que todos los niños tengan derecho a comedor.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

en los meses de verano. Lara no es una experta en gestión pública ni sabe cómo llevar a cabo la propuesta.

Para ello, Laura comienza a desarrollar una propuesta ciudadana en la aplicación, explicando la base de la propuesta. Al no saber cómo financiarlo, deja la propuesta abierta para que la comunidad la pueda ayudar a desarrollarla.

Requisitos:

1. Agregar (acción) una nueva propuesta (objeto) llenando los principales campos del formulario (contexto).
2. Publicar (acción) la propuesta (objeto) como una propuesta colaborativa para editar (contexto)
3. Mostrar (acción) la propuesta (contexto) en la lista de propuestas colaborativas para que puedan colaborar otros usuarios (contexto).

Escenario IV

Fran es un periodista de un periódico digital. Se acercan las elecciones y está redactando un pequeño artículo acerca de los programas de algunos partidos políticos. Necesita acceder a los programas completos de los partidos y visualizar las secciones que le resulten de interés para comentarlas en su artículo.

Accediendo a la aplicación, Fran puede seleccionar los programas de todos los partidos que se presentan a las elecciones. Listando el índice del programa y accediendo a sus secciones.

Requisitos:

1. Visualizar (acción) todos los partidos (objeto) que se presentan a las elecciones (contexto).
2. Seleccionar (acción) un partido político (objeto) para visualizar el programa electoral (contexto).
3. Listar (acción) el índice (objeto) de secciones del programa seleccionado (contexto).
4. Visualizar (acción) la sección (objeto) del programa seleccionado por el usuario (contexto).

6.3.5. Framework de diseño

En esta sección detallaremos todos los aspectos relacionados con el desarrollo del aspecto visual y la interacción con la aplicación. Se utilizarán los escenarios y requisitos definidos en la fase anterior para crear los bocetos y prototipos interactivos. Detallaremos los prototipos desarrollados en papel, algunos prototipos intermedios y el prototipo final de la aplicación presentado.

6.3.5.1. Framework de interacción

Para definir el Framework de interacción realizaremos un proceso de seis etapas:

Factor de forma, postura y métodos de entrada

La aplicación será visualizada en un smartphone bajo el sistema operativo Android (con API 15 como mínimo), con tamaños de pantalla entre 3,5 y 6 pulgadas. Que pueda visualizarse en interiores y exteriores.

La **postura** será **temporal**. El usuario puede utilizarlo en periodos de tiempo muy breves como puede ser la consulta de alguna sección, propuesta o dar su valoración. Elaborando una propuesta o participando en el desarrollo de una, el usuario necesitará algo más de tiempo, pero seguirá utilizando funciones básicas. Propias de una postura temporal.

Elementos y datos funcionales

Elementos de datos:

- Programa político
 - Atributos: partido, programa, secciones, índice, temas.
 - Relaciones: partido-programa, índice-sección.
- Temas
 - Atributos: categoría, programa, sección, partido, propuesta.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

- Relaciones: categoría-programa, categoría-propuesta, partido-sección.
- Propuestas Ciudadanas
 - Atributos: propuesta, categoría, usuario.
 - Relaciones: propuesta-usuario, propuesta-categoría.
- Propuestas Colaborativas
 - Atributos: propuesta, categoría, usuario, edición colaborativa.
 - Relaciones: usuario-propuesta, propuesta-categoría.

Elementos funcionales:

- Visualizar los partidos políticos que se presentan a las elecciones.
- Mostrar el índice de cada programa político.
- Mostrar, valorar y debatir las secciones de los programas políticos.
- Visualizar categorías por temas, distinguiendo entre propuestas y secciones de programas.
- Mostrar, valorar y debatir las propuestas de los usuarios.
- Crear una nueva propuesta en una categoría.
- Mostrar las propuestas colaborativas y participar en su desarrollo.
- Crear una nueva propuesta incompleta para desarrollarla de forma colaborativa.

Grupos funcionales y jerarquías

Pantalla principal de la aplicación. Grupos de elementos funcionales que contiene:

- Lectura y valoración de Programas políticos.
 - Visualizar programas políticos.
 - Ver secciones de programas.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

- Visualización, valoración y debate de secciones de programas políticos.
- Clasificación de secciones de programas y propuestas ciudadanas por categorías.
 - Ver secciones de programas políticos por tema.
 - Ver propuestas ciudadanas por tema.
- Lectura, valoración y creación de Propuestas Ciudadanas.
 - Ver propuestas ciudadanas.
 - Valorar y debatir propuestas ciudadanas.
 - Creación de propuesta ciudadana.
- Lectura, colaboración y creación de Propuestas Colaborativas.
 - Visualizar las propuestas ciudadanas en desarrollo.
 - Crear una propuesta colaborativa.
 - Colaborar en el desarrollo de una propuesta.

Boceto del framework de interacción

El proceso de desarrollo de bocetos se ha realizado en espiral, de tal forma que una vez que identificábamos una determinada funcionalidad, la representábamos en un mockup a papel para debatirlo y discutirlo en las entrevistas. A continuación trasladábamos el mockup a un prototipo de alto nivel desarrollado en Android. Para las siguientes implementaciones que añadían nueva funcionalidad el proceso era el mismo:

1. Desarrollo de un boceto a papel identificando los elementos funcionales.
2. Definir la interacción a través de la herramienta POP [?].
3. Implementar un prototipo de mayor fidelidad en Android.

De esta forma tuvimos que dar un total de tres vueltas a nuestro modelo en espiral para completar el desarrollo.

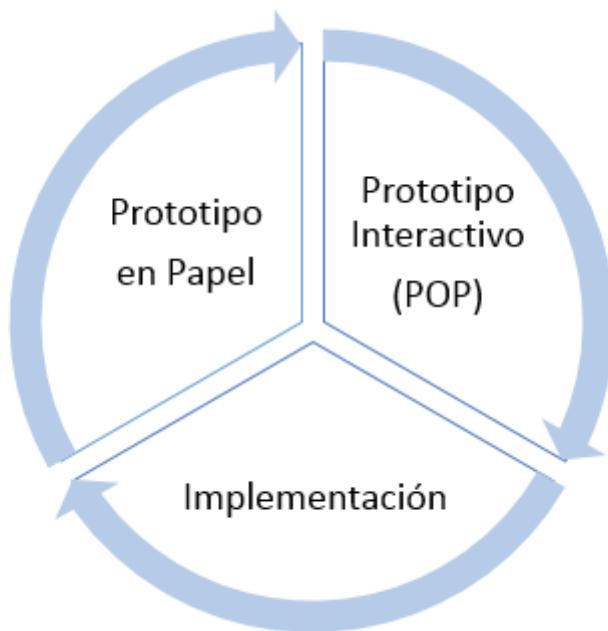


Figura 6.4: Modelo de desarrollo de los prototipos.

1. Programas Políticos

Estos fueron los primeros bocetos que definían la interacción de la visualización de los programas políticos:

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

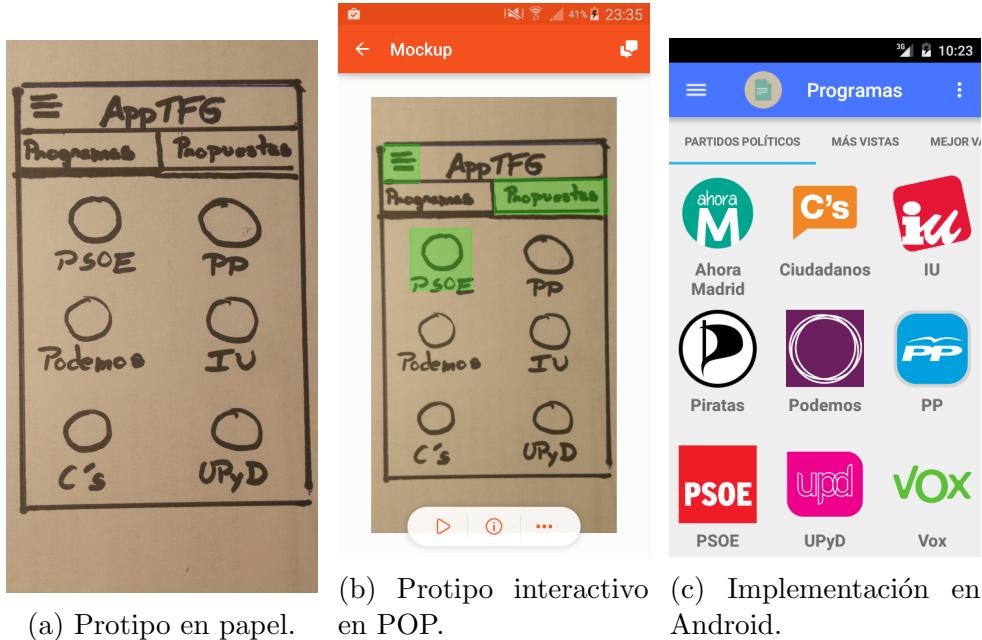


Figura 6.5: Primeros prototipos sobre programas electorales.

Elementos del prototipo en papel:

- a) Partidos políticos
- b) Programas políticos
- c) Índice de programas políticos
- d) Secciones de prorammas políticos

Funcionalidades del prototipo interactivo:

- a) Visualizar la lista de partidos políticos.
- b) Seleccionar un programa político.
- c) Navegar por el índice de un programa polítcico.
- d) Visualizar secciones de un programa, valorarla y comentarla.

Implementación en Android:

- a) Descargar la lista de partidos políticos del servidor.
- b) Descargar el programa seleccionado.
- c) Navegar por el índice del programa.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

- d) Descargar el contenido de las secciones.
- e) Añadir comentarios al servidor.
- f) Modificar parámetros sociales en el servidor (like, dislike, views, etc).

2. Bocetos de Propuestas Ciudadanas

A partir de lo desarrollado hasta el momento, se procedió a diseñar las propuestas en la aplicación. Manteniendo la coherencia con el diseño anterior, contuamos desarrollando los prototipos que añadían la funcionalidad de las propuestas ciudadanas.

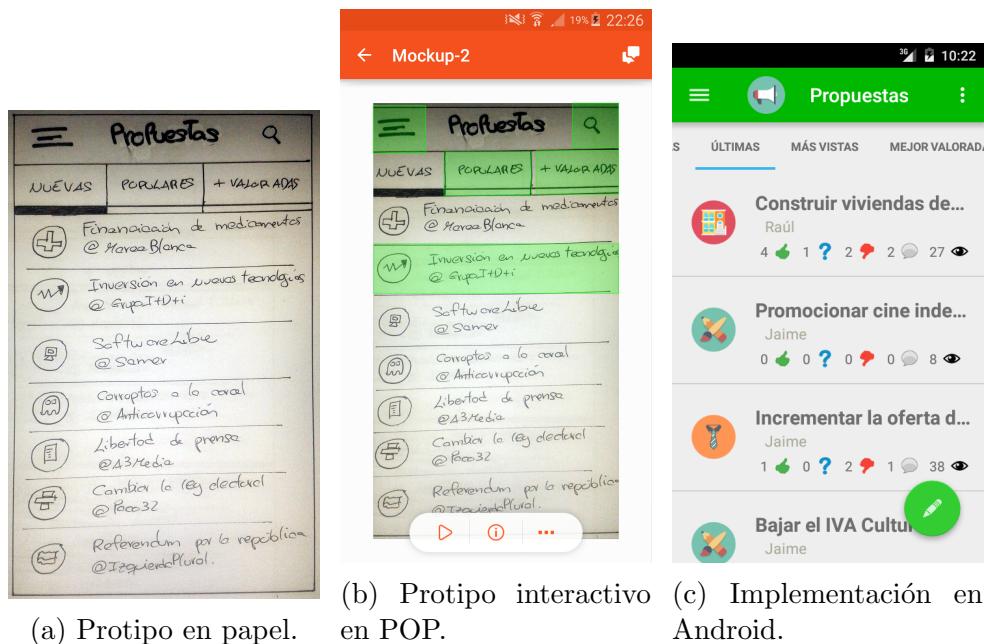


Figura 6.6: Prototipos para el desarrollo de propuestas.

Elementos del protipo en papel:

- a) Propuestas ciudadanas.
- b) Listado de propuestas por tops.
- c) Listado de secciones de programas por tops.
- d) Desarrollar una propuesta.
- e) Pantalla principal rediseñada.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

Funcionalidades del prototipo interactivo con POP:

- a) Visualizar propuesta.
- b) Listado de propuestas por tops.
- c) Listado de secciones de programas por tops.
- d) Desarrollar una propuesta.
- e) Pantalla principal rediseñada.

Implementación en Android:

- a) Visualizar propuesta almacenada en el servidor.
- b) Listado de propuestas por tops sincronizada con el servidor.
- c) Agregar una nueva propuesta al sistema.
- d) Pantalla principal rediseñada.

3. Bocetos de Propuestas Colaborativas

Escenarios *key path*

En esta sección describiremos los escenarios *key path*, que describen cómo interactúa una persona con la interfaz diseñada en el framework de interacción. Son la evolución de los escenarios de contexto, pero describiendo cómo interactúa la persona con los elementos de datos funcionales en la interfaz.

I Escenario de cómo visualizar un programa político de un partido que se presenta a las elecciones.

Faltan quince días para las elecciones generales y Carlos aún no ha decidido su voto. Un amigo le recomienda que instale *DemoCritics* para poder leer algún programa político y ver las propuestas de las diferentes opciones políticas. Dentro de la aplicación, Carlos visualiza la lista de todos los partidos que se presentan a las próximas elecciones generales, y dentro de cada uno, puede visualizar el programa. Obteniendo un *top* de las secciones más populares, Carlos visualiza las partes del programa que más llaman su atención.

II Escenario de cómo agregar una nueva propuesta en el sistema.

Laura trabaja en la administración de la EMT y lleva varios días discutiendo con sus compañeros de trabajo una forma de reestructurar las líneas de autobuses. De forma que no confluyan todas en el centro,

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

si no que alguna línea recorra los barrios exteriores del centro. Laura decide instalar *DemoCritics* para poner en conocimiento de toda la ciudadanía la propuesta que lleva semanas trabajando. Para ello Larua se sitúa en la parte de *propuestas* de la aplicación y agrega una nueva. Introduciendo la descripción de la propuesta, cómo piensa llevarla a cabo y el coste que supondría, Laura consigue publicar su propuesta en la aplicación para que los usuarios puedan valorarla y debatirla.

III Escenario sobre cómo solicitar ayuda para elaborar una propuesta.

Manuel es un gran aficionado a la música, y en la localidad donde vive no hay escuelas de música municipales. Él quiere llevar a cabo una propuesta que se base en el desarrollo de nuevas escuelas de música municipales en puntos estratégicos y así fomentar la educación musical. Manuel toma su móvil en la pantalla principal de *DemoCritics* y solicita la colaboración de una propuesta que facilite la creación de escuelas musicales. Ya que a pesar de ser un gran aficionado musical, Manuel no sabe cómo llevar a cabo esa propuesta o qué presupuesto necesitaría. Así pidiendo ayuda a los usuarios, Manuel tendrá desarrollada su propuesta por aquellos interesados y expertos del tema en la comunidad.

IV Escenario para colaborar en el desarrollo de una propuesta.

Sofía lleva varios años trabajando en la administración pública del estado, y siempre ha querido ayudar utilizando la experiencia que le ha dado su trabajo estos años. Pero nunca ha sabido ni dónde ni cómo colaborar. Un conocido le recomienda instalar *DemoCritics* en su móvil personal, así podría ayudar a personas que necesitan ayuda de sus conocimientos. Con la aplicación instalada, Sofía se dirige a las propuestas colaborativas para explorar aquellas propuestas que están en desarrollo y pueden necesitar su ayuda. Seleccionada la propuesta, accede a su contenido y procede a editar el documento que explica cómo se llevaría a cabo la propuesta.

Escenarios de validación

En esta fase se pondrán a prueba los escenarios, validando a ciertas situaciones que nos ayudarán a encontrar fallos o posibles mejoras en el diseño. Utilizaremos preguntas para definir la respuesta del sistema para problemas o dudas que puedan surgir.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN



Figura 6.7: Pantalla principal de la aplicación.

- A** **Qué pasaría si un usuario accede a la aplicación por primera vez y quisiera consultar el programa político de un partido en concreto?**

El usuario accedería a la pantalla principal de la aplicación (ver figura 6.7), donde podría diferenciar cuatro grupos principales: *Programas Políticos*, *Categorías*, *Propuestas Ciudadanas* y *Propuestas Colaborativas*. Accediendo sobre el ícono de programas políticos, la aplicación mostraría al usuario el listado de partidos políticos para explorar los programas. Al usuario le bastaría con seleccionar el partido político del que quisiera consultar su programa electoral. Finalmente, la aplicación mostraría al usuario el índice del programa electoral del partido seleccionado.

- B** **¿Cómo podría acceder un usuario a las secciones de todos los programas donde tratan un tema en concreto?**

Desde el menú principal, el usuario accedería al menú de *categorías* para filtrar las secciones o propuestas por temas. Una vez allí, el usuario seleccionaría la categoría (ver figura 6.8) que deseara para obtener todas las secciones de los programas de los partidos políticos que estuvieran relacionadas con esa categoría en concreto.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

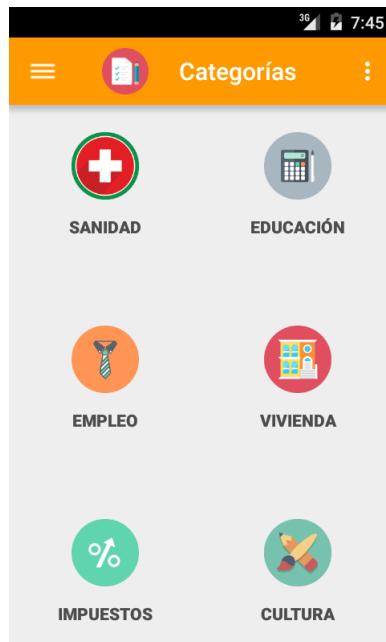


Figura 6.8: Vista de categorías.

C ¿Qué tendría que hacer un usuario que quiere publicar una nueva propuesta relacionada con la educación?

Debería ir al menú de *propuestas* de la aplicación y seleccionar el botón de *agregar* situado abajo a la derecha para acceder al formulario para publicar una nueva propuesta. Dentro del formulario, el usuario deberá de seleccionar la categoría de *sanidad*.

D ¿Cómo podría colaborar un experto en cultura en la aplicación?

Para ver si alguna propuesta requiere la ayuda de alguien en el mundo de la cultura, deberá acceder al menú *Propuestas Colaborativas* y seleccionar la categoría *Cultura* entre las listadas de la aplicación. Allí seleccionará aquellas propuestas que le resulten de interés para ayudar a su elaboración. Dentro de la propuesta podrá ayudar a redactar en un *pad* colaborativo cómo llevar a cabo la propuesta o cómo financiarla.

6.3.6. Principios de Diseño

Con el objetivo de verificar los principios de diseño que cumple la aplicación, utilizaremos los 10 principios de diseño propuestos por Jakob Nielsen [?] para

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

evaluar la calidad del sistema.

1. **Visibilidad del estado del sistema:** Siempre que el sistema ejecuta tareas o no puede continuar por algún motivo, se mantiene informado al usuario sobre lo que está pasando. Esto podemos encontrarlo cuando el usuario accede a los programas políticos y el sistema procede a descargar la información. Mientras se descarga la información, se muestra un pequeño diálogo indicando la descarga de los datos para mantener al usuario informado.
2. **Relación entre el sistema y el mundo real (Metáforas y lenguaje familiares):** El sistema mantiene un lenguaje próximo al usuario, evitando términos más formales del sistema. Cuando visualizamos una sección de un programa o una propuesta, el sistema emplea el lenguaje común de términos como *like* (mano con pulgar hacia arriba), *dislike* (mano con pulgar hacia abajo), etcétera, para valorar las secciones de un programa o una propuesta.
3. **Control y libertad del usuario:** La aplicación permite deshacer acciones fácilmente, como puede ser cambiar la opinión de un *like* o *dislike*, o cancelar la acción de publicar una nueva propuesta o comentario.
4. **Consistencia y estándares:** La aplicación sigue las pautas de diseño propuestas por las pautas de diseño *Material Design* [?]. Este estilo de diseño plano fue propuesto por Google en 2014, y en la actualidad la mayor parte de aplicaciones móviles Android lo utilizan a la hora de diseñar sus pantallas. Por esto, los usuarios no deberían tener problemas navegando y utilizando los controles de la aplicación, ya que son familiares al encontrarse en un amplio catálogo de aplicaciones. Agregar una propuesta desde un botón de lápiz inferior situado a la derecha, representar un listado de objetos mediante *tabs* deslizables, son algunas de las características destacables de *Material Design*.
5. **Prevención de errores:** La aplicación muestra mensajes de error cuando algo le impide funcionar con normalidad. De esta forma se avisa al usuario de que algo no está funcionando como debería. Cuando ejecutamos la aplicación y el sistema detecta que no hay conexión, la aplicación notifica al usuario en un sencillo diálogo la necesidad de tener conexión a internet para operar con la aplicación.
6. **Reconocimiento mejor que recuerdo:** Para agregar una nueva propuesta o comentario, tan sólo se necesita una transición entre dos pantallas. De esta forma facilitamos al usuario la generación de nuevos

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

contenidos sin saturarle la memoria u obligarle a recordar mucha información para llegar a su objetivo. Este principio se encuentra en la mayor parte de las aplicaciones móviles, pues el usuario suele interactuar con el dispositivo en pequeños intervalos de tiempo y además no suele tener una postura cómoda que le facilite la concentración. Por lo que se requieren pocas transiciones entre pantallas, tareas sencillas y elementos sencillos y distinguibles. Por ejemplo, también identificamos las distintas partes de la aplicación (Secciones, Categorías, Propuestas) con colores distintos para que el usuario sepa en todo momento dónde se encuentra.

7. **Flexibilidad y eficiencia:** Se utilizan atajos a funcionalidades disponible desde la mayor parte de las pantallas del sistema. Haciendo uso del botón "*hamburguesa*" en *material design* nos da la posibilidad de abrir un menú con accesos directos a las principales partes de la aplicación. De tal forma que no necesitemos volver hacia atrás para seleccionar otra opción. No obstante, ambos caminos pueden ser utilizados en función de la habilidad o conocimiento del usuario respecto a la aplicación.
8. **Estética y diseño minimalista:** Tanto los diálogos que muestra el sistema como la información visible en la mayoría de las pantallas del sistema es clara y concisa. Así evitamos dar información innecesaria al usuario o distraer su atención para evitar su confusión.
9. **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de los errores:** Cuando el usuario da con un error inesperado, el sistema informa de lo ocurrido mediante un sencillo diálogo de error. Simplificando el mensaje mostrado al usuario para que pueda resolverlo sin grandes problemas.
10. **Ayuda y documentación:** Cuando el usuario genera contenido, el sistema va indicando al usuario los pasos que tiene que seguir para completar su tarea de forma satisfactoria. Esto sucede por ejemplo en el caso en el que un usuario construye una nueva propuesta, donde el sistema le indica los campos que tiene que llenar y la información que debe ir en cada campo.

CAPÍTULO 6. DISEÑO DE LA APLICACIÓN

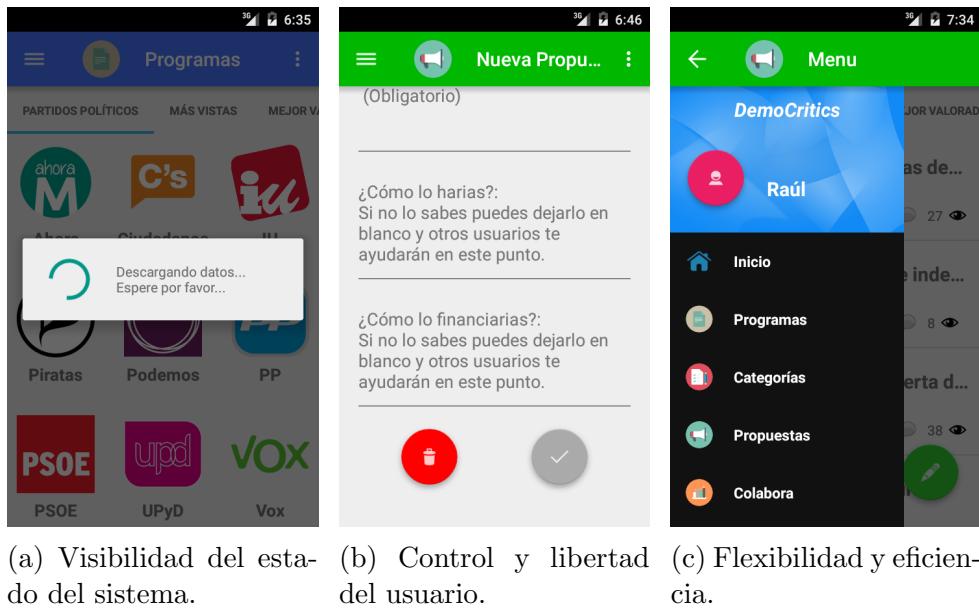


Figura 6.9: Principios de diseño en la aplicación.

Capítulo 7

Arquitectura del Proyecto

7.1. Arquitectura de Wave

La arquitectura que soporta Wave tal y como la diseñó originalmente Google está estructurada en forma de capas que interactúan entre sí. De forma general y de abajo hacia arriba disponemos de las siguientes capas:

- Capa de Conexión Cliente/Servidor: Se encarga de gestionar la conexión entre servidor y servidor mediante el protocolo XMPP y la conexión entre cliente y servidor mediante el protocolo Wave y WebSockets.
- Capa de Control de OT: Se encarga de gestionar la consistencia en tiempo real de los datos mediante la utilización de Transformaciones Operacionales (OT).
- Capa de Modelo de Datos Wave: Se encarga de gestionar los datos a nivel de las Waves que forman parte de una Conversación.
- Capa de Modelo Conversacional: Se encarga de gestionar los datos a nivel de las Conversaciones, junto a los documentos que las componen y los usuarios participantes (Ver Sección 7.1.1). En el caso de SwellRT se extendió este modelo a otro más generalizado llamado Modelo de Contenidos (Ver Sección 7.1.2).
- Capa de Interfaz de Cliente: API con operaciones para que el cliente interactue con el Modelo Conversacional subyacente mediante eventos. En el caso de SwellRT se extendió este API para utilizar el Modelo de Contenidos de esta tecnología.

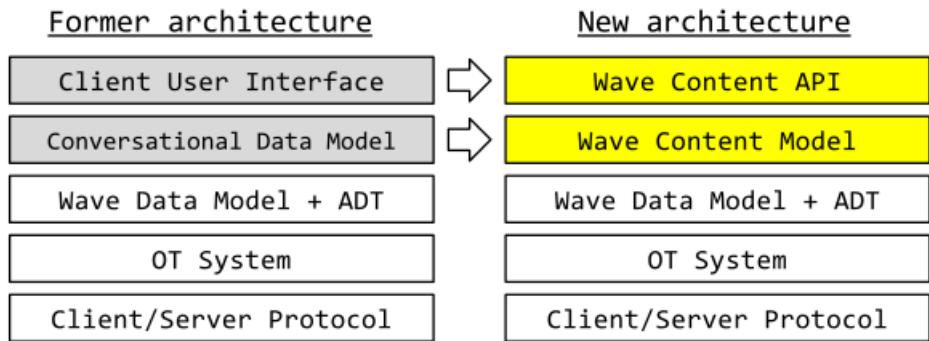


Figura 7.1: Arquitectura de Wave

7.1.1. Modelo Conversacional Wave

Además de definir el protocolo del que hace uso Wave, Google definió un Modelo de Datos Conversacional [?] que refleja la arquitectura de los datos que componen las conversaciones en Wave. Así, a grandes rasgos, se pueden 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 [?] [?]:

- **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.

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

- **Extensiones [?]**: 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.

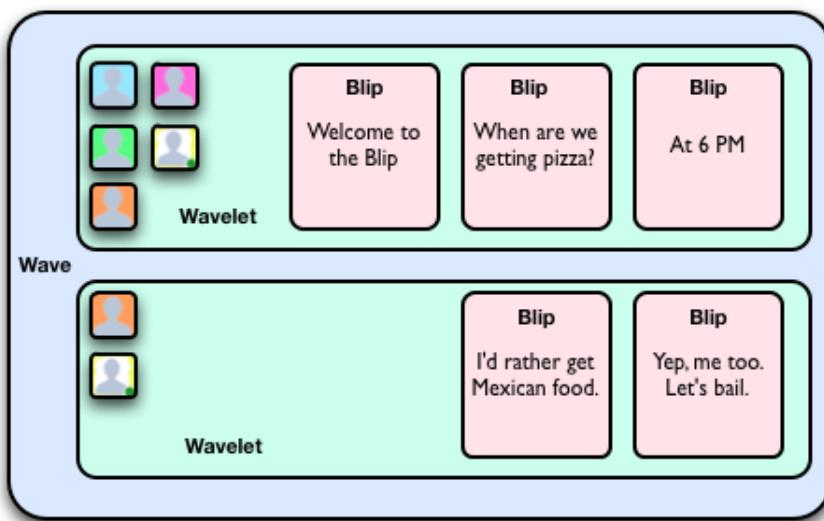


Figura 7.2: Modelo Conversacional de Wave

7.1.2. Modelo de Contenidos SwellRT

SwellRT [?] (Swell Real Time) es un framework de desarrollo para Wave que forma parte del proyecto europeo P2P Value [?]. Está basado en el servidor Wave In A Box [?] (WIAB) y extiende sus funcionalidades cambiando el Modelo Conversacional original creado por Google por uno nuevo de propósito más general llamado Modelo de Contenidos. En este modelo nuevo se pueden intercambiar estructuras de datos de forma colaborativa, federada y en tiempo real que combinan los siguientes 4 tipos: **Mapas, Listas, Strings y Documentos de Texto**.

De esta manera la estructura de datos intercambiada constará siempre de un Mapa inicial (llamado "root") del cual colgarán a modo de árbol el resto

de estructuras (de cualquiera de los cuatro tipos que maneja SwellRT) cuyos datos se gestionarán en tiempo real. Para ello existe también una API nueva que se encarga de la gestión de dicho arbol mediante la utilización de eventos que notifican de cambios en las estructuras de datos. Este modelo será el que se utilizará para la aplicación Android. El siguiente es un esquema de los cambios en el modelo original introducidos por SwellRT:

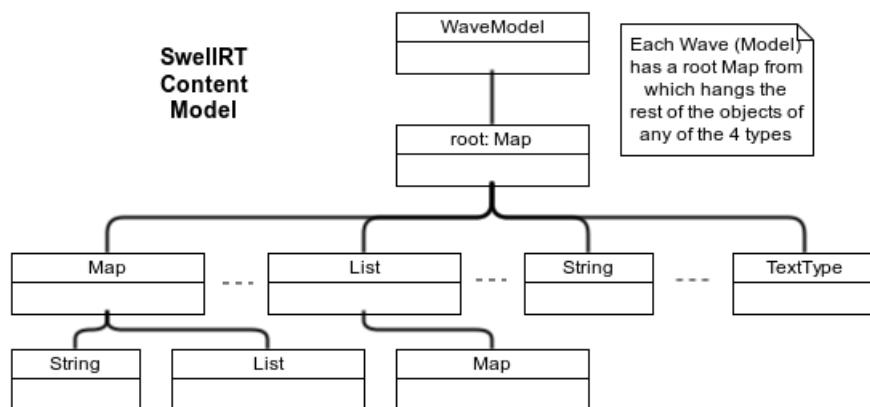


Figura 7.3: Modelo de Contenidos de SwellRT

7.2. Arquitectura de la aplicación

La arquitectura de DemoCritics está compuesta por cuatro módulos principales de los que se hablará en profundidad en las siguientes subsecciones. Como **Cliente móvil** se tendrá la aplicación desarrollada en Android, que realizará peticiones HTTP al Servicio Web RESTful (PHP+Laravel+MySQL). Dicho **Servicio Web RESTful** será quien gestione las peticiones de la aplicación móvil mediante el protocolo HTTP. La **Base de Datos MySQL** almacenará toda la información relacionada con la aplicación. El API del Servicio Web RESTful será quien actúe de intermediario entre las peticiones de la aplicación y las operaciones en Base de Datos. En este caso alojaremos la Base de Datos y el Servicio Web RESTful en caso en OpenShift [?], una plataforma que permite alojar servicios web de forma gratuita.

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

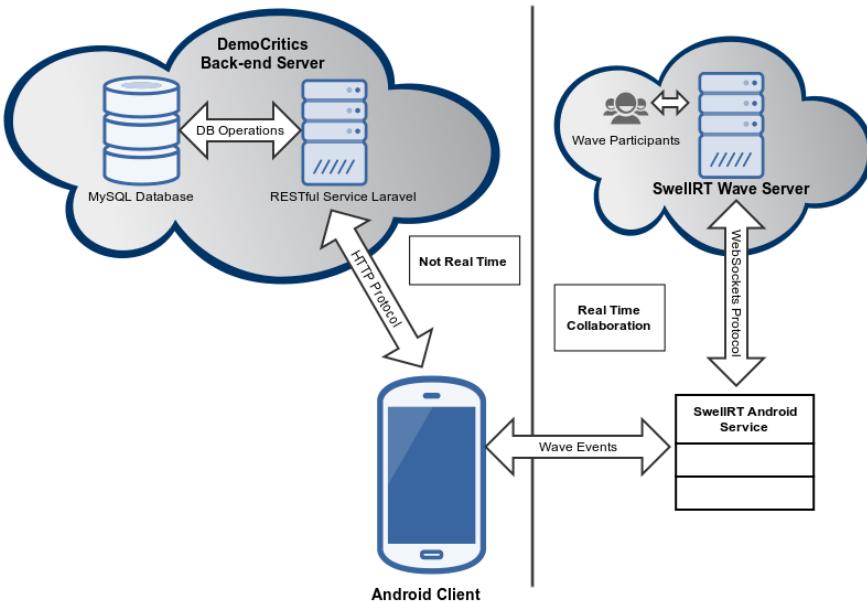


Figura 7.4: Arquitectura general de la aplicación.

Por otro lado, la aplicación hará uso del **Servicio Android** desarrollado en SwellRT-Android [?] para conectarse con el servidor de colaboración en tiempo real SwellRT (Java/Jetty+MongoDB) e intercambiar los datos que sea necesario mantener en tiempo real, es decir, la edición de una Propuesta de forma colaborativa entre varias personas.

7.2.1. 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 [?] como sistema de gestión de base de datos (SGBD) y phpMyAdmin [?] como herramienta de gestión gráfica de la base de datos.

La base de datos está formada por un total de nueve tablas donde se almacena toda la información relacionada con los programas de los partidos políticos, las propuestas ciudadanas, etc. 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 otros.

Las tablas *section* y *political_party* se utilizan 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*,

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

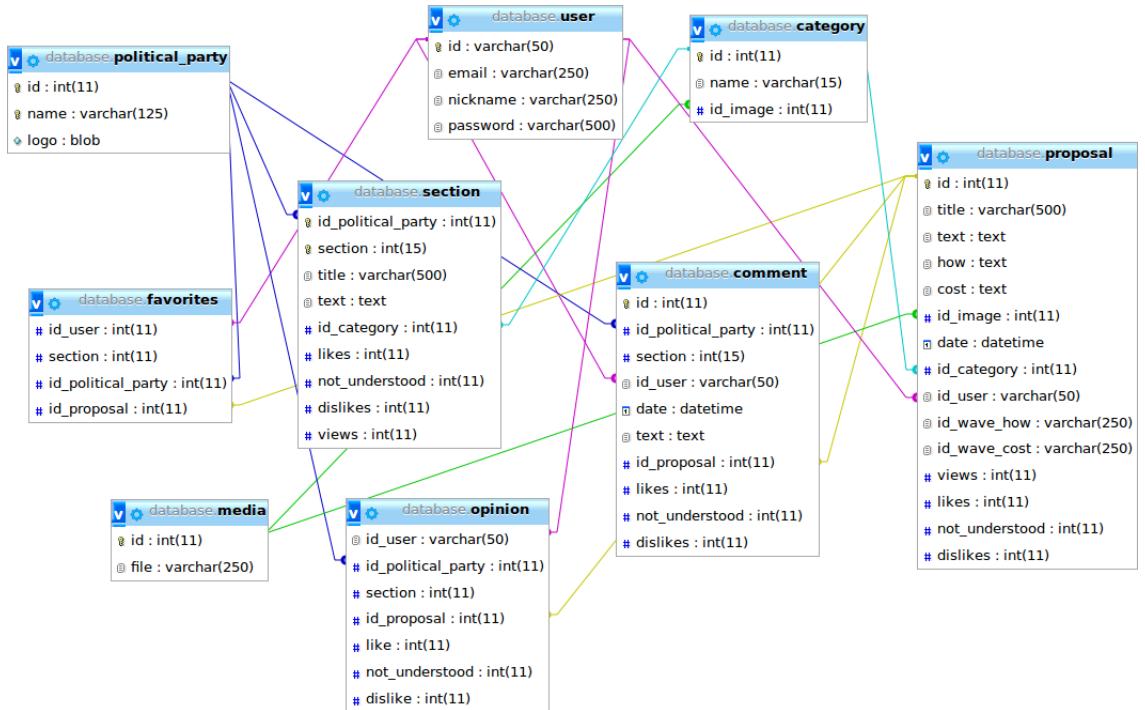


Figura 7.5: Modelo entidad-relación de la base de datos.

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 (salvo que la redacción del programa cambie).

Definimos como datos *estáticos* aquellos datos que permanecen intactos en la aplicación desde el inicio. Son datos que no requieren ser modificados y siempre permanecen tal y como están, a no ser que surgiera algún fallo o imprevisto en la aplicación. Estos datos principalmente estarán formados por la información de los partidos políticos y sus programas. Se entiende que de cara a unas elecciones los partidos políticos no suelen cambiar su nombre, logo o lo que es el programa electoral. Por lo tanto esos datos permanecerán intactos durante el uso de la aplicación. Sin embargo, la aplicación contará con datos *dinámicos* como aquellos datos que se irán generando y que con el tiempo pueden cambiar o eliminarse. Este tipo de dato estará relacionado con la mayor parte de la actividad del usuario con la aplicación. Por ejemplo, cuando un usuario agrega una nueva propuesta, esta propuesta será añadida como una fila más de la tabla *proposals*. Además esta nueva fila estará sujeta a cambios que definirán su valoración, el número de comentarios, el número de visualizaciones, etcétera.

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

A continuación se procederá a explicar las tablas más relevantes de la base de datos:

7.2.1.1. *political_party*

Esta tabla contiene la información relacionada con los partidos políticos que se presentarán a las elecciones. Por tanto nos interesará guardar el nombre del partido, un logo que los identifique en formato *blob* y un identificador único para relacionarlos con sus programas políticos.

7.2.1.2. *section*

Esta es una de las tablas más complejas de la aplicación, pues contiene toda la información de los programas políticos. Se encontrará una clave primaria compuesta definida como *id_political_party* y *section*, que harán referencia a una sección concreta de un partido político. De esta forma nunca se encontrará una fila que contenga la misma sección de un programa del mismo partido político dos veces, ya que a cada sección le correspondería un solo partido político (para más información sobre la estructura del campo *section* consultar el capítulo 7.2.3.2). Por otro lado está el contenido tanto de la propia sección como del programa político, es decir, el título de la sección, el contenido en texto, etc. como datos *estáticos* de la tabla. Y por último, los indicadores sociales de *likes*, *not_understood*, *dislikes* y *views* como datos *dinámicos* que se irán actualizando con la actividad de los usuarios.

7.2.1.3. *proposal*

La tabla encargada de almacenar las propuestas que se van agregando a la aplicación. Estará definida por campos como un identificador único para la propuesta, el identificador del usuario que ha publicado la propuesta y el contenido de dicha propuesta. Estos datos serán *estáticos* una vez que se inserten en la base de datos. Al igual que la tabla *section*, también tendrá columnas son datos de carácter social y estadístico que irán modificándose con la actividad de los usuarios en el sistema. Por último, para diferenciar una propuesta publicada de una propuesta colaborativa, siendo esta última la que hace uso de *SwellRT* para edición colaborativa, definiremos dos identificadores que harán referencia a los identificadores Wave de dos documentos colaborativos alojados en el servidor. De esta forma se puede saber cuando una propuesta se está desarrollando o está definitivamente publicada. Ya que

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

para las propuestas publicadas en la aplicación, estas dos columnas tendrán el valor *null*, ya que no harán referencia a ningún documento colaborativo.

7.2.1.4. *comment*

Para almacenar los comentarios generados por los usuarios en la aplicación, se utilizará la tabla *comment* que guardará el identificador del usuario y el texto del comentario. Esta tabla guardará los comentarios realizados en las secciones, como también los comentarios en las propuestas. De tal forma que un comentario para una sección de un programa político, irá acompañado de las columnas *id_political_party* y *section*, de la misma manera que se relacionan las secciones de programas con partidos políticos. De lo contrario, si un comentario pertenece a una propuesta, deberá contener el identificador único de la misma mientras que los anteriores campos estarán a *null*. El resto de las columnas estarán formadas por datos de valoración y carácter social.

7.2.1.5. *opinion*

Esta tabla será la referencia que almacenará las opiniones o valoraciones de un usuario en las propuestas y secciones de programas de la aplicación. De tal forma que se guardará el identificador del usuario que ha añadido esa opinión, acompañado del identificador de la propuesta si es el caso o el identificador de la sección del programa y el partido político en el caso de que se encuentre valorando una sección. Para indicar el valor de la opinión del usuario, es decir si ha valorado la sección o propuesta como *like*, *dislike*, etc, se utilizarán columnas que contendrán un entero que indicará el tipo de valoración que ha pulsado el usuario. De tal forma que el usuario siempre pueda cambiar su opinión o borrarla.

7.2.2. Service REST

Para establecer la conexión de la aplicación desarrollada en Android con la Base de Datos se ha utilizado **Laravel** como *framework* para desarrollo de servicios Web. Laravel [?] es un framework de código abierto para desarrollar aplicaciones web con PHP 5. Laravel permite además desarrollar una API REST (Representational State Transfer), un estilo de arquitectura software para sistemas hipermedia. Este término se originó en una tesis doctoral sobre la web escrita por Roy Fielding [?].

CAPÍTULO 7. 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.

Tabla 7.1: Funciones CRUD

Laravel 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 se quiera. Estas peticiones se realizan mediante el protocolo HTTP. En función de la operación que se desee hacer, se clasificarán estas funciones en el acrónimo **CRUD** [?] (del original en inglés: **C**reate, **R**ead, **U**pdate and **D**elete).

Dependiendo de las peticiones que se realicen al servicio mediante el protocolo HTTP, el servidor devolverá un código de estado para obtener *feedback* de lo sucedido. Por tanto se distinguirán diferentes códigos de estado cuando se quiera obtener un recurso, actualizar un recurso, crear uno nuevo o borrarlo. En la siguiente tabla se definen los códigos de estado que puede devolver el servidor en función de las peticiones.

Usar un servicio RESTful 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 se podrán hacer peticiones desde el cliente en Android, sino que más adelante se podría desarrollar una versión web o incluso un cliente para iOS sin tener que modificar el servicio, ya que las peticiones HTTP serán las mismas.

7.2.2.1. Arquitectura

Laravel utiliza e implementa un funcionamiento basado en el patrón **MVC** (Model–View–Controller), separando el modelo de la vista, y delegando la gestión al controlador. En este caso tendríamos un modelo almacenado en las tablas de la Base de 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 re-

CAPÍTULO 7. 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.

Tabla 7.2: Códigos de estado de la respuesta del servidor

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

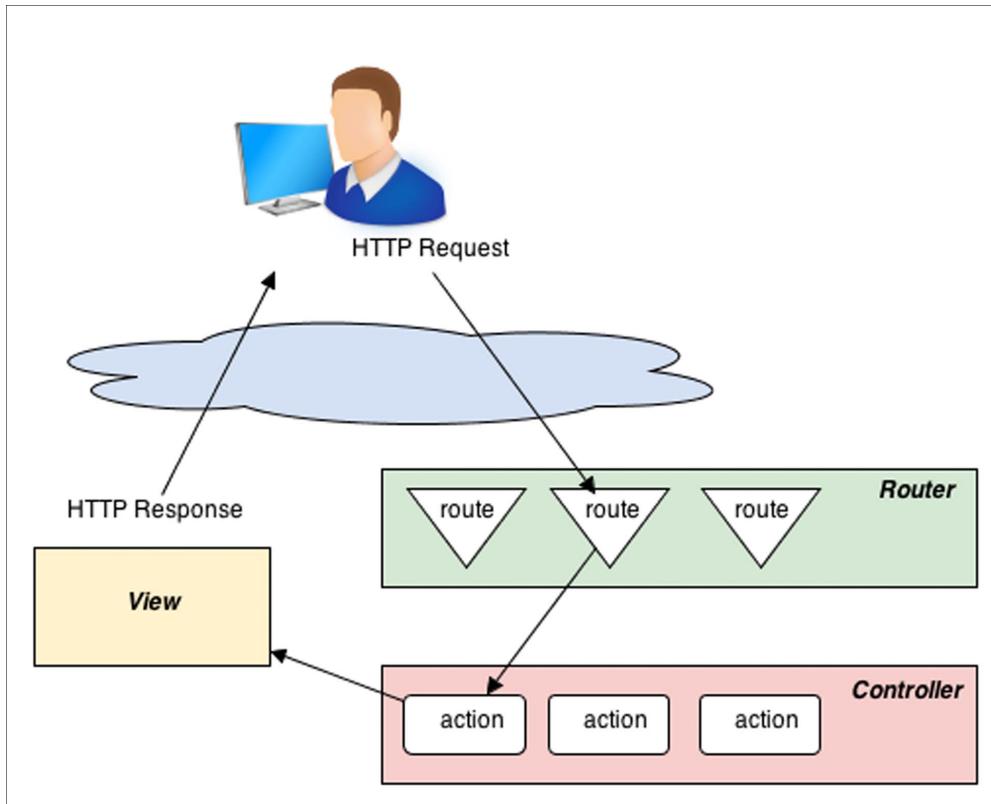


Figura 7.6: Arquitectura de Laravel

quiera. Por último, se tendría como vista el resultado devuelto por el servidor con los datos solicitados por el cliente, que a su vez los representaría 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 se consigue elaborar un código más claro y sencillo, así como también evitar conflictos entre el modelo y la vista.

Para comprender la arquitectura de Laravel se van a destacar tres componentes clave. En primer lugar está 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 desee realizar. Después, en función de la ruta a la 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. El último componente será la **vista**, que puede ser representada de muchas formas. En este caso, la vista estará compuesta por una serie de datos en formato JSON, que posteriormente el cliente móvil se encargará de representar en su propia

vista. Teniendo en cuenta todo esto, se procederá a detallar un poco más estos tres componentes fundamentales.

7.2.2.2. Rutas

Cómo se citaba 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 se definirán 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 7.1), seguido de la definición de la URL por la que será accesible, y por último se indicará el controlador responsable de procesar la petición. Aparte 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*, se han seguido una serie de buenas prácticas [?], habituales a la hora de desarrollar una RESTful API. Normalmente se utilizarán nombres en lugar de verbos para acceder a los recursos. Por ejemplo, si queremos visualizar el listado de propuestas de la aplicación, se utilizará 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 procesar 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 se quisiera acceder a la propuesta cuyo id es igual a 5, definiríamos la siguiente ruta:

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

Nótese que la ruta es la misma que la anterior, a la que se ha añadido un nuevo parámetro que deberá especificar el usuario. Así, la petición *GET /proposal/5*, devolvería la propuesta con id 5. Se puede ver que el controlador también es el mismo, pero esta vez el método encargado de procesar

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

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, se deberá realizar una petición POST. La ruta será exactamente igual que la primera, pero antes se deberá 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`. Se puede ver que la primera ruta dónde se obtenían las propuestas (`GET /proposal`, es exactamente igual que la que se acaba de definir. Sólo les diferencia el tipo de petición que está realizando el cliente. Para definir otros tipos de peticiones como PUT o DELETE, se usará la misma metodología. Un ejemplo del resultado final de las peticiones que podrían realizarse a una propuesta se puede ver en la Sección A.2.1 del Apéndice.

Para organizar el código y poder visualizarlo de forma clara se utilizarán grupos de rutas. Esto resultará especialmente útil cuando se tengan 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. Se puede observar en el ejemplo anterior 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 agrupará todas las rutas que comienzen de la misma forma. Así el ejemplo anterior quedaría más claro y ordenado (ver Sección A.2.2 del Apéndice).

De esta forma se pueden visualizar todas las posibilidades agrupadas dentro de la dirección `proposal`. Lo que permitirá utilizar grupos para ordenar funciones comunes dentro de un mismo prefijo, y además crear subgrupos dentro de los anteriores si se tiene algún prefijo repetido varias veces. Esto podría aplicarse a el ejemplo anterior, agrupando las peticiones que requieren como parámetro un `id` de la propuesta (ver Sección A.2.3 del Apéndice).

7.2.2.3. 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

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

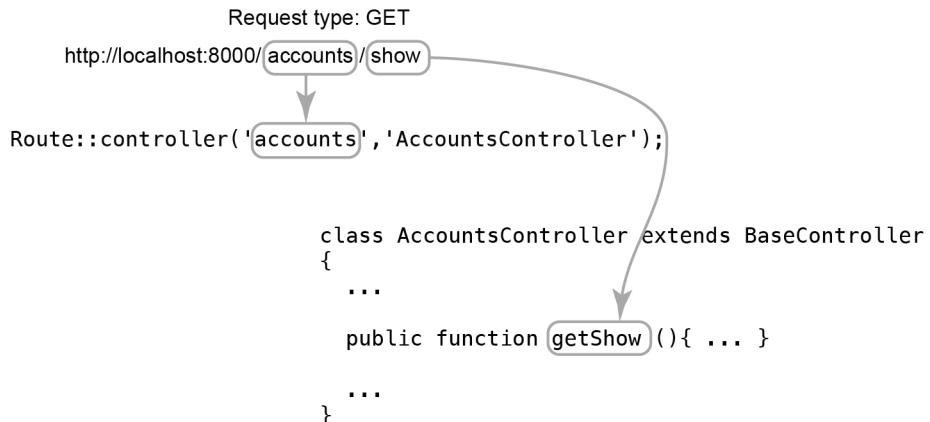


Figura 7.7: Camino desde la ruta al método del controlador.

en la aplicación. Cada controlador hereda de una clase abstracta llamada *Controller*, que a su vez hereda de la clase abstracta *BaseController*. La cual implementa la funcionalidad básica de los controladores.

Dentro de un controlador se definirán 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. Un ejemplo que muestra una implementación básica del método *index()* del *ProposalController* se puede ver en la Sección A.2.4 del Apéndice).

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 proporciona más independencia y versatilidad a la hora de programar. Como respuesta del método, Laravel utiliza por defecto un fichero JSON con los resultados de la variable a la que se asigne el resultado, o a la sentencia SQL de la consulta que se esté 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\u00edmites 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."}]
```

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

```
    "cost ":"Hacer\u20ac7\u20ac de \u20accalles\u20ac peatonales:\u20ac750.000\u20ac",
    "id_image":4,"date ":"2015-05-06\u20ac00:00:00","id_category"
    :4,"id_user ":"6d823fa54c6d1a12","views":4,"likes":3,
    "not_understood":0,"dislikes":0}
]
```

7.2.3. Cliente Android

La plataforma Android divide el desarrollo de una aplicación en dos partes: la implementación de la lógica detrás de la aplicación mediante el uso de Activities [?] o Services [?], y la configuración del aspecto de la interfaz mediante Layouts XML [?]. En las siguientes secciones se hablará de algunos de los aspectos más destacados de este proyecto en referencia a dichas características de Android.

7.2.3.1. Interfaz Gráfica

En el desarrollo de esta aplicación se quiso utilizar un diseño plano basado en las últimas guías de diseño de Android disponibles en su última versión 5.0. Google denominó a este diseño plano "Material Design". Por tanto se usa esta versión de Android (API 21) para desarrollar la aplicación. A pesar de que aún no mucha gente utiliza Android 5.0 "Lollipop" (Ver figura 3.1) muchas de sus características son retrocompatibles con versiones anteriores, por lo que aunque la gente no posea la última versión en sus dispositivos podrá hacer uso de las características de "Material Design".

En Android, cuando se crea una Actividad (en su método "onCreate()") se asigna una Vista ("View") a la actividad que representa la interfaz del usuario. Esta vista está definida en un archivo de Layout XML que se deberá modificar para crear la interfaz gráfica añadiendo componentes gráficos [?] ("widgets") y modificando sus atributos mediante etiquetas XML para adaptarlos a nuestras necesidades. Además se les puede asignar un identificador único para poder moverse por dichos widgets luego desde el código de la aplicación mediante una navegación en árbol utilizando el método "findViewById()" proporcionado por Android.

Estos componentes gráficos pueden ser los básicos proporcionados por Android (como cajas de texto, botones, listas, etc.) o se pueden crear *Vistas Personalizadas* en caso de que los widgets básicos no aporten la funcionalidad deseada.

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

En este último caso será necesario definir un Layout para la vista personalizada (que puede a su vez estar compuesto por widgets básicos o no) y una clase que posea referencias a los datos y a los componentes del layout que los albergarán.

Sin embargo a menudo se querrán usar varias de estas vistas, ya sean básicas o personalizadas, juntas dentro de una lista, por lo que necesitaremos un *Adaptador*[?] ("Adapter") que haga de intermediario entre el layout y la clase que lo define para "adaptar" los datos que posee la clase a los componentes del layout que los muestran.

Vamos a ver un ejemplo de esto con la vista (widget) personalizada llamada **PartyWidget**, creada para mostrar cada uno de los elementos de la lista de Partidos Políticos que se visualiza en la aplicación. El funcionamiento de este widget personalizado esta formado por los siguientes tres elementos:

- **party_widget.xml**: Layout XML que define la disposición de los dos componentes gráficos básicos que conforman esta vista: un "ImageView" para el logo del partido y un "TextView" para el nombre del partido.
- **PartyWidgetView.java**: Clase que define el objeto que contiene referencias a los dos componentes anteriores (mediante sus IDs) y los datos de la imagen (de tipo Bitmap) y el nombre (String) del partido.
- **PartyWidgetAdapter.java**: Clase que extiende de un "BaseAdapter" de Android y define el adaptador que se encargará de gestionar una lista de elementos gráficos de un mismo tipo (en este caso del tipo "PoliticalParty") y de crear las vistas que muestran los datos almacenados en dicha lista. El método más importante de un Adapter es el "getView", que crea la vista personalizada (en este caso de tipo "PartyWidgetView") y le asigna los datos que hay dentro de cada elemento de la lista de la forma que nosotros le definamos.

De esta manera, solo es necesario decirle al elemento que contiene la lista de Partidos Políticos (en este caso un "GridView" que permite visualizarlos en forma de una rejilla que se adapta al tamaño de pantalla) que use como adaptador el PartyWidgetAdapter con los datos de la lista de Partidos Políticos que previamente han sido descargados.

Navegación por Tabs

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

Tras la fase de investigación se decidió que sería interesante incluir en la aplicación distintos filtros para ver tanto las secciones como las propuestas más valoradas positivamente, más vistas, más comentadas, etc. Para implementar la vista de esta funcionalidad se decidió utilizar un diseño muy parecido al que utiliza Google en su tienda de aplicaciones "Google Play" para navegar de forma lateral mediante pestañas ("tabs") deslizantes. En este caso cada una de las tabs mostraría una lista (elemento "ListView" de Android) de elementos ordenados por distinto filtro.

No obstante, el método utilizado por Google en anteriores versiones de Android basado en añadir Tabs a la barra superior ("Action Bar")[?] de la aplicación está actualmente obsoleto en el API 21, por lo que se tuvo que recurrir a la actual implementación de Google, que no se encuentra todavía dentro del API 21 de desarrollo. Es por eso que hubo que utilizar un par de clases ("SlidingTabLayout" y "SlidingTabStrip") que definen las tabs utilizadas por Google en el diseño con "Material Design". Estas clases fueron extraídas del GitHub[?] de la app que Google mostró en su aplicación de prueba cuando presentó "Material Design" a finales de 2014. Este elemento trabaja definiendo el contenido de cada Tab mediante la utilización de Fragmentos[?] ("Fragment") de Android, que representan una porción de lo que se muestra dentro de una Actividad.

Sin embargo, la vista (Layout) de una Sección y de una Propuesta dentro de esta lista sería muy similar (con su foto, título e indicadores sociales de número de likes, vistas, etc.) por lo que ¿cómo hacer para utilizar la misma vista en dos objetos que son de tipos distintos? La solución que se encontró para utilizar el mismo elemento de visualización (definido en el layout llamado "top_ranking_item") en ambos casos fue utilizar una interfaz que implementarían tanto la clase "Section" como la clase "Proposal": la interfaz "TopItem".

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

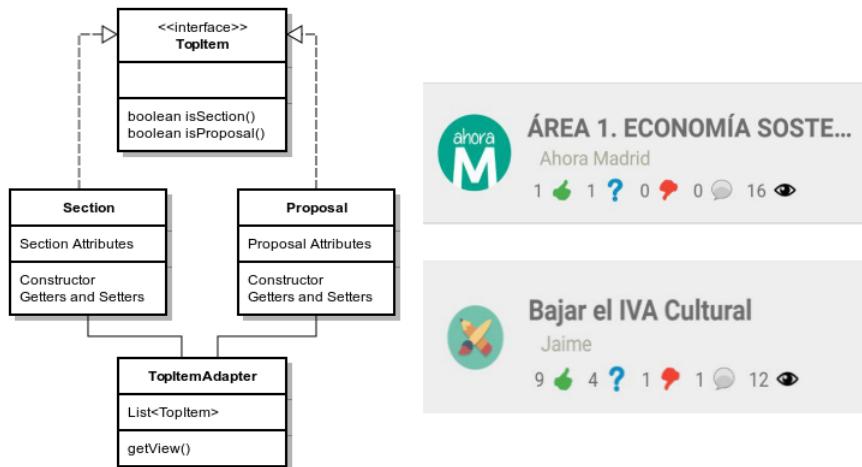


Figura 7.8: Esquema de Implementación de TopItem (Con vista de Section y Proposal)

De esta forma, las clases `Section` y `Proposal` implementan los métodos `"isSection()"` e `"isProposal()"` de la interfaz devolviendo true en cada caso respectivamente. Así, cuando dentro de el Adaptador (`"TopItemAdapter"`) se traten los elementos de su lista de `TopItems` en su método `"getView()"`, se podrá preguntar si se trata de un `Section` o un `Proposal` para llenar la vista del `ListView` con los datos correspondientes a cada caso. Solo hace falta por tanto crear listas de Secciones o Propuestas y pasárselas al adaptador personalizado después de definirlo como adaptador para el `ListView`.

Menús de Navegación

Con el objetivo de proporcionar al usuario una navegación por la aplicación que permita ir a cualquiera de sus secciones independientemente de en qué pantalla te encuentres actualmente, se decidió utilizar un menú lateral que se puede deslizar desde el lado izquierdo de la pantalla. Android permite añadir menús [?] a las Actividades mediante métodos específicos de éstas, aunque en este caso son menús fijos sencillos a los que se accede mediante el botón de opciones del móvil o de la app.

Se pretendía utilizar el menú lateral deslizante presente en la mayoría de aplicaciones del estilo `"Material Design"` introducido por Google en su última versión de Android 5.0 (API 21), por lo que se utilizó un componente gráfico llamado `"Navigation Drawer"` para ello. Este componente está presente en todos los layouts de la aplicación y contiene dentro un `"ListView"` con las

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

entradas del menú. Cada una de estas entradas es asimismo un componente personalizado, definido en el layout menu_left_item.xml (contiene un logo y el nombre de la entrada de menú). En consecuencia, y tal y como se ha hablado antes en esta sección, existirá también un Adapter "MenuLeftListAdapter" que se encargará de gestionar la vista de la lista de entradas del menú. Además, se le ha añadido una cabecera ("Header") al menú, que contiene la información de nombre de Usuario y un botón para cambiar dicho nombre.

Sin embargo, si todas las Actividades que componen la aplicación disponen de este menú, ¿Cómo hacer para evitar repetir el código de generación y configuración de dicho menú en todas las Actividades? La solución encontrada fue utilizar una Actividad padre llamada "MenuActivity" que se encargaría de generar y de albergar los métodos que configuran el menú. De esta manera, cualquier Actividad de la aplicación extendería de "MenuActivity" y llamaría a sus métodos de generación y configuración de menú con una referencia a su propia vista de Layout (véase que cada Layout contiene el elemento "Navigation Drawer" necesario para mostrar el menú).

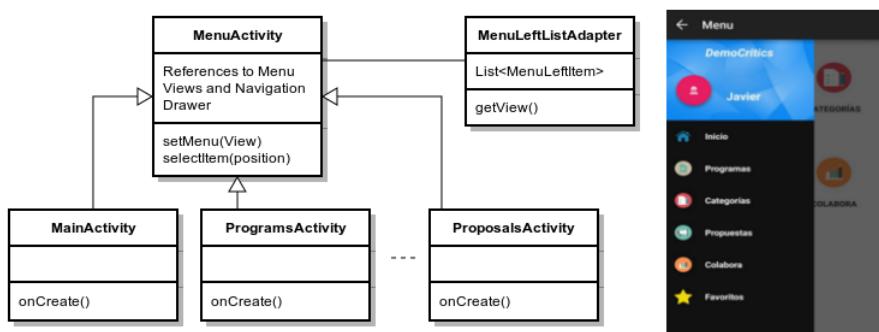


Figura 7.9: Esquema de Implementación de Menú Izquierdo (Con vista del Menú)

Existe también en la aplicación un Menú derecho deslizable que muestra un Índice del Programa de un Partido Político para poder navegar por él de manera más sencilla (similar al que utiliza la aplicación de la "Wikipedia" para navegar por un artículo). Este menú solo está presente en la actividad "SectionViewerActivity", ya que solo es necesario acceder al índice del programa cuando se está navegando por sus secciones. Utiliza también para ello un "Navigation Drawer" similar al del otro menú antes explicado. No obstante, en este caso se decidió utilizar un tipo distinto de lista ("ExpandableListView") que se diferencia de la lista normal en que posee dos niveles

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

de navegación que se pueden expandir. De esta forma es necesario definir los layouts de los distintos niveles ("groups") y los subniveles ("childrens") dentro de cada uno.

Adapter	Layout	Clase	Descripción
PartyWidgetAdapter	party_widget.xml	PartyWidgetView	Widget para mostrar un elemento de la lista de Partidos Políticos
MenuLeftListAdapter	menu_left_item.xml	MenuLeftItem	Widget para mostrar las entradas del Menú lateral izquierdo.
TopItemAdapter	top_ranking_item.xml	TopItem	Widget para mostrar un elemento de la lista con los Tops de Secciones y Propuestas.
ListIndexAdapter	(solo muestra texto)	No es necesario	TextView (Android) que muestra los nombres del índice de un Programa Político.
ExpandableListAdapter	list_child_item.xml list_group_item.xml	No es necesario	TextView que contienen los títulos de las distintas secciones y subsecciones (desplegables) del índice de un Programa Político.
CommentListAdapter	comment_item.xml	Comment	Widget para mostrar un comentario de un usuario dentro de la lista de comentarios de una Sección o una Propuesta.
SampleFragmentPagerAdapter	tab_page_*.xml	TabPageFragment	Widget para definir lo que contiene cada tab mediante Fragmentos cuya vista es generada de forma dinámica.

Tabla 7.3: Adapters, Views y Clases utilizadas en el proyecto

7.2.3.2. Estructuración de Programas Políticos

A la hora de almacenar los Programas Políticos en la Base de Datos hubo que estudiar la forma más adecuada de hacerlo para poder estructurar dichos programas en distintos niveles de secciones y subsecciones. En definitiva, el problema radicaba en cómo codificar la información de a qué nivel de anidamiento pertenecía cada Sección. Además, había dos opciones: o bien el service REST devolvía un programa ya estructurado o la propia aplicación

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

se encargaba de estructurarlos.

Codificación en Base de Datos

En principio se quería elegir una codificación que permitiera realizar una consulta a la base de Datos que devolviera las distintas secciones y subsecciones **en el orden en el que aparecen en el Programa**. Tras varios intentos con distintos campos de Strings y enteros como atributo de la tabla de Secciones en Base de Datos, se decidió que lo mejor sería utilizar un único número entero que codificara dicha información. Tras estudiar la composición de diversos programas se identificó que **no serían necesarios más de 4 niveles de anidamiento en subsecciones**. En consecuencia, el número estaría formado por 8 cifras, siendo cada dos cifras un nivel distinto de anidamiento. Los siguientes ejemplos muestran cómo funciona esta codificación:

01020302	<u>Ejemplos:</u>
Nivel: 1 2 3 4	
	01040000 -> 1.4
	10050200 -> 10.5.2
	02130607 -> 2.13.6.7
	01100101 -> 1.10.1.1

Figura 7.10: Ejemplos de Codificación de Secciones y Subsecciones

Utilizando esta codificación se consiguió que mediante una simple consulta SQL la Base de Datos devuelva todas las secciones del programa de un determinado partido de forma ordenada, siendo luego el Servicio Web REST (Ver Sección 7.2.2) el que pasará los resultados de esta consulta en JSON a la aplicación Android.

Estructuración y Almacenamiento de Secciones

Con dicha codificación se consiguió ordenar las secciones en el orden en el que aparecen en el programa, pero ahora se debían guardar en la aplicación en una estructura que permitiera navegar por el programa de forma estructurada, pudiendo en cada momento ver las subsecciones o volver a la sección anterior. Se decidió por tanto **guardar el programa en una estructura en árbol formada por listas de listas**. De esta forma cada elemento de la lista sería un objeto del tipo Section que contendría la información de dicha sección (título, texto, etc.) y otra lista con las subsecciones.

Pero, ¿cómo construir esta estructura a partir de la lista ordenada que de-

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

vuelve el Service REST? La solución encontrada fue utilizar una función que construyera dicho árbol de forma recursiva (Ver clase "GetProgramsData.java") sobre una sección "root" vacía que pertenecería a cada Partido Político. Dicha función recorrería la lista de secciones sin estructurar y para cada sección comprobaría su nivel de anidamiento son el fin de saber si colocarlo en el nivel actual, en el siguiente o en el anterior.

Se puede ver el código de esta función de construcción del árbol en la Sección A.3.1 del Apéndice.

Además, como no se pretendía tener que descargarse el programa continuamente, esta estructura en forma de arbol se guardaría en una clase llamada "PoliticalGroups" que alamacenaría el listado de Partidos Políticos y para cada uno de ellos almacenaría la estructura en árbol de su programa (su nodo "root"). Esta clase implementa el **patrón de diseño Singleton** para asegurar que solo existe una instancia de ella, accesible desde cualquier punto de la aplicación.

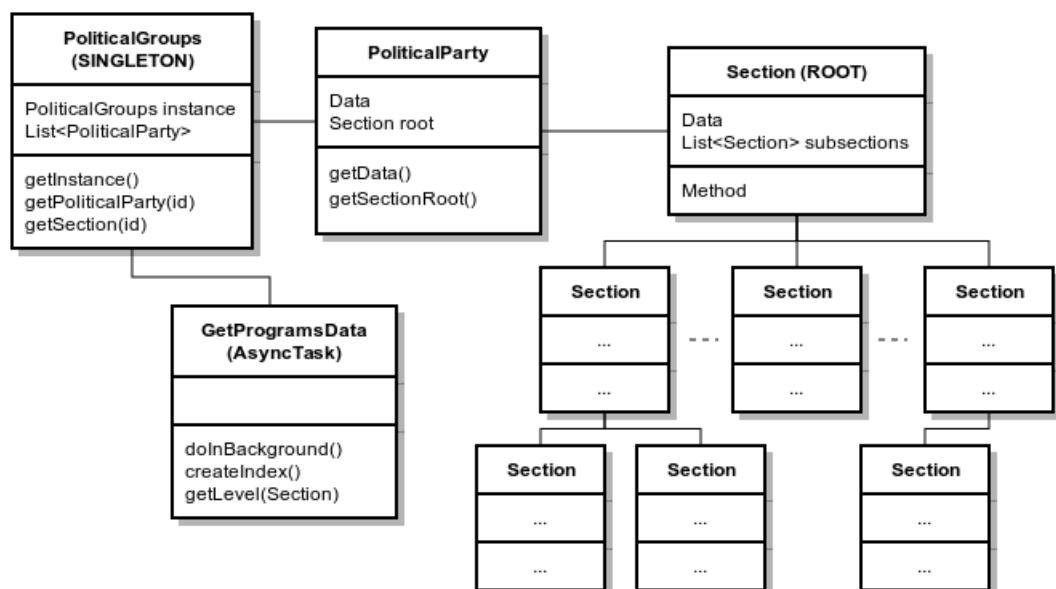


Figura 7.11: Esquema de estructura en árbol de Programa Político

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

7.2.3.3. Conexión y Peticiones a Service REST y Base de Datos

Las peticiones de información a Base de Datos (ya sea para crear datos, descargarlos o modificarlos) se realizan mediante el protocolo HTTP, enviando dichas peticiones al Servicio Web REST (Ver Sección 7.2.2). Para enviar dichas peticiones es necesario por tanto realizar una conexión HTTP al Servicio Web REST desde la aplicación Android. Aprovechando la experiencia previa adquirida durante la migración de Wave (Ver Sección 3.2.3.1), se decidió utilizar la librería "HttpURLConnection" de Android y un esquema basado en AsyncTask que ejecutan el proceso de conexión y descarga de datos en un thread separado del UI Thread tal y como recomienda Google hacer para trabajar con conexiones de red[?].

Así, en función de la operación que se quiera hacer en Base de Datos, para cada tipo de consulta existirá un AsyncTask encargado de conectarse al servidor con la URL apropiada y tratar los datos en JSON que éste le pueda devolver, ya sea para mostrarlos o guardarlos en algún objeto de la aplicación. De forma general se pueden identificar cuatro tipos de peticiones al servidor (Ver tabla 7.1).

Con el objetivo de no repetir código se decidió estructurar el esquema de peticiones HTTP mediante herencia de AsyncTasks. El siguiente diagrama esquematiza la estructura:

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

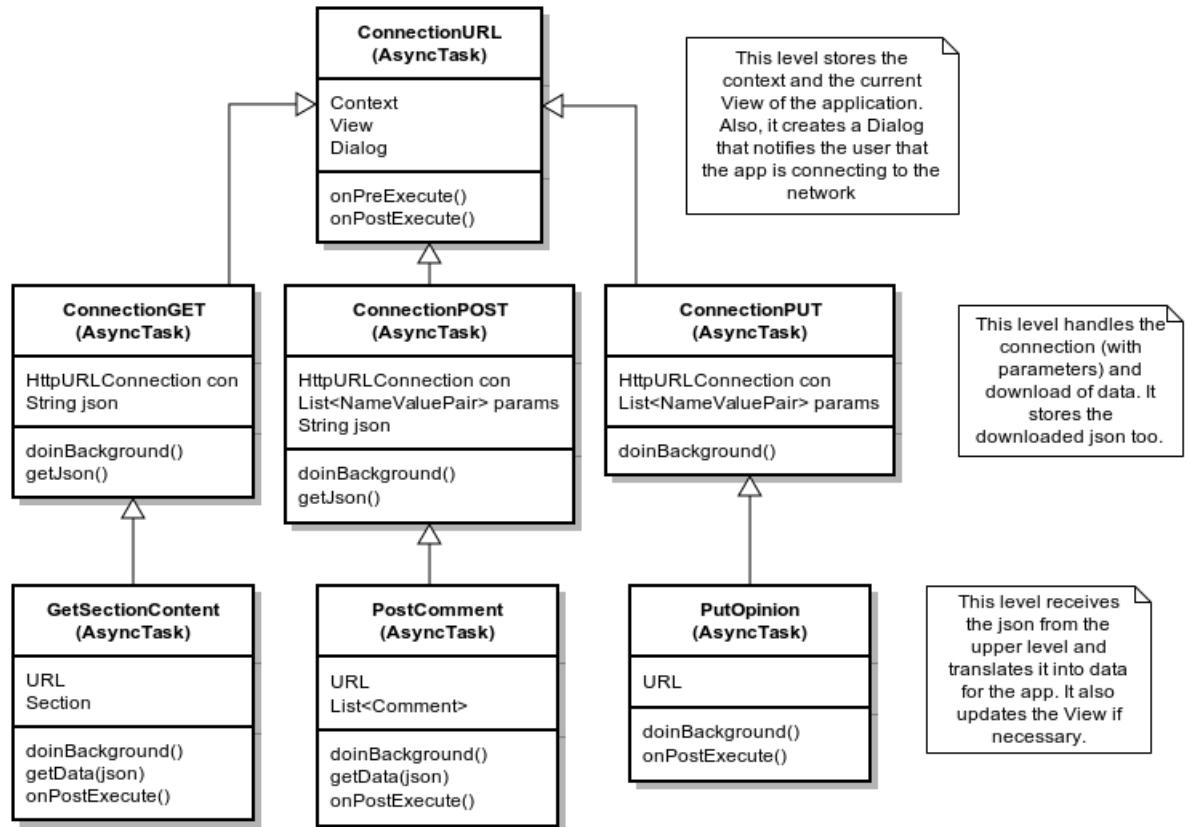


Figura 7.12: Esquema de Clases de Conexión HTTP mediante AsyncTask

Hace falta también actualizar la vista del usuario cuando se reciban los datos. Como un AsyncTask se ejecuta en un hilo aparte diferente del UI Thread (para no bloquear la interacción con el usuario), a priori no tiene acceso a la vista (Layout) que se le está mostrando al usuario. Por esta razón la solución que se encontró fue guardar una referencia a la vista del usuario dentro del AsyncTask (ver clase "URLConnection"). De esta manera en su método "onPostExecute()" que se ejecuta nada más terminar la operación de conexión del "doInbackground()", se obtiene dicha vista y se actualiza con los datos recién descargados. Este método de actualización de la vista es el que se utiliza en todos los casos.

Comprobación de Conexión a Internet

Por otro lado, para llevar a cabo un control de errores en la conexión al Servicio Web REST, antes de ejecutar la tarea en el AsyncTask se comprueba el

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

estado de la conexión del dispositivo móvil para asegurarse de que el usuario dispone de conexión a Internet. En caso contrario se le muestra un mensaje de aviso para que compruebe sus conexiones y no se lleva a cabo la tarea de conexión en el AsyncTask. Aprovechando que todas las actividades extienden de "MenuActivity" dicho método de comprobación se encuentra dentro de esa clase. Además, fue necesario definir el siguiente permiso en el "AndroidManifest.xml" de la aplicación para acceder a la información de estado de la conexión del dispositivo:

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

7.2.3.4. Gestión de Usuarios: ID de Android

Como esta versión de la aplicación no maneja información crítica y sensible del usuario se decidió utilizar un método de autenticación basado en el ID almacenado en el dispositivo móvil. En Android existen varios identificadores (IMEI, ID de dispositivo, ID de Android..) teniendo cada uno de ellos características distintas. En este caso **se decidió utilizar el ID de Android** [?] ya que es único para cada instalación de sistema operativo y acceder a su valor no requiere de permisos especiales en la aplicación. Se trata de un número de 64 bits en formato hexadecimal, al cual se accede directamente mediante la variable del sistema "Secure.ANDROID_ID".

El acceso a este identificador se produce nada más ejecutar la aplicación en su actividad "MainActivity" y se guarda de forma estática en la clase "User" para poder acceder a él desde cualquier punto de la aplicación. Con el identificador también se accede a la Base de Datos para descargar el nombre de Usuario que se corresponde con dicho identificador. Este nombre se muestra en el menú izquierdo de la aplicación y se puede cambiar desde ahí. De esta manera, todo el contenido generado por un usuario (Opiniones, comentarios, propuestas,...) se guarda también en base de datos asociado a este identificador único.

7.2.3.5. Servicio Android, Conexión con Wave/SwellRT y Gestión de Propuestas Colaborativas

Al servicio (ServiceSwellRT) desarrollado durante la migración de SwellRT a Android (Ver Sección 3.2.5) se le añadió encima el API del Modelo de Contenidos de SwellRT (Ver Sección 7.1.2). Este API trabaja con Waves (o "modelos") con identificadores únicos, y dentro de cada modelo habrá un

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

“objeto root” del tipo Mapa del que colgarán los distintos objetos con los que podremos jugar de cada uno de los cuatro tipos básicos de datos: Mapas, Listas, Strings y Documentos de Texto.

En este proyecto se utiliza SwellRT para soportar la edición de propuestas colaborativas. Este tipo de propuestas puede tener asociadas dos textos colaborativos que se corresponden con la manera de llevar a cabo la propuesta y cómo financiarla. Para implementar esto con wave será necesario por tanto crear una wave por cada propuesta colaborativa, que tendrá asociados a su mapa ‘root’ dos objetos del tipo Documento de Texto (TextType). Este tipo permite crear documentos colaborativos que pueden ser editados en tiempo real por todo aquel usuario de wave que esté registrado como participante.

El funcionamiento del Servicio Android de conexión con SwellRT exige primero enlazar (“bind”) la Actividad actual con el Service para poder hacer uso del API. Después de enlazar, se podrá registrarse en el servidor SwellRT y crear, abrir, cerrar o borrar waves, a las que también se podrán añadir participantes y nuevos objetos de los cuatro tipos antes mencionados.

En SwellRT se accede a estos objetos a partir del identificador que se les otorgue al crearlos. En este caso se creará dentro de cada wave dos TextTypes con identificadores “howPropz costProp” para cada uno de los textos colaborativos antes mencionados. Asimismo, cada propuesta tendrá asociada una wave con un identificador único generado por el servidor al crearse la wave. Este identificador será el que se guarde en la base de datos para que cualquier usuario pueda acceder a la edición colaborativa de la propuesta. El siguiente es un esquema del uso de SwellRT en el proyecto.

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

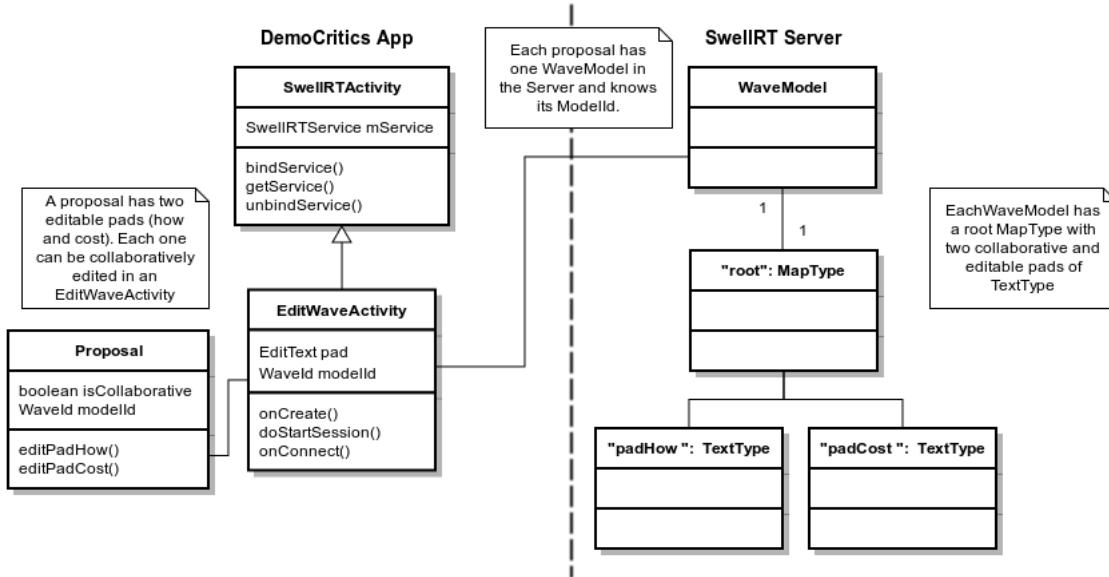


Figura 7.13: Esquema de Uso de Wave/SwellRT

En el caso de DemoCritics lo primero que se tendrá que hacer será registrar al usuario usando su ID (Ver Sección 7.2.3.4). Esto se hace en el ”MainActivity” al ejecutar la aplicación. Una vez registrado solo será necesario utilizar el ServiceSwellRT para llevar a cabo dos actividades: crear una nueva propuesta colaborativa y editar una propuesta ya creada.

Un ejemplo de cómo registrarse con SwellRT se puede encontrar en el Apéndice A.3.2.

Crear Propuesta Colaborativa

Esta es una opción que se le ofrece al usuario en el caso de que, al hacer una propuesta nueva, deje alguno de los campos de ”¿Cómo lo haría?” o de ”¿Cómo lo financiaría?” en blanco. En tal caso, lo primero que se hace es hacer un login en el servidor Wave llamando al método ”startSession” del Service. Una vez que se ha iniciado sesión, se abre un nuevo modelo (wave) en el servidor (ver Sección 7.1.2) para esta propuesta y se obtiene el Id (”WaveId”) de dicho modelo creado para guardarla en la base de datos junto al resto de datos de la propuesta. Además, se le asocian al elemento ”root” del modelo uno o dos documentos colaborativos del tipo TextType llamados ”padHow” y ”padCost”, en función de si ha dejado los dos campos vacíos o sólo uno. Por último se añade como participante del modelo creado

CAPÍTULO 7. ARQUITECTURA DEL PROYECTO

al usuario que lo ha creado (su ID_USER). Cabe destacar que la mayor parte de la programación con el serviceSwellRT se lleva a cabo mediante callbacks que responden a eventos como iniciar sesión, crear modelos, etc.

Un ejemplo simplificado del código encargado de crear el modelo se puede ver en el Apéndice A.3.3.

Editar Propuesta Colaborativa

Cuando un usuario quiere echar una mano a otro en la redacción de su propuesta, puede acceder a la propuesta colaborativa y editar los campos de ”¿Cómo lo haría?” o de ”¿Cómo lo financiaría?” en función de en qué necesite ayuda el usuario. Para cada uno de estos campos existe un ”pad” en el servidor wave asociado a una wave (ver Sección 7.1.2) cuyo identificador ”WaveId” está guardado en la base de datos. Cuando el usuario pulsa en uno de los dos campos, se abre una nueva actividad ”EditWaveActivity” que le ofrece un campo ”EditText” de Android en el que puede escribir y ver lo que otros escriben en tiempo real. Para esto hace falta abrir una sesión en Wave, abrir el modelo con su WaveId, obtener el modelo y el pad, y asociar (”bind”) el pad a el EditText del usuario para que el servicio sea capaz de obtener los eventos de escritura del usuario.

Un ejemplo simplificado del código encargado de abrir el modelo y asociar el pad colaborativo se puede ver en el Apéndice A.3.4.

Capítulo 8

Evaluación con Usuarios

En esta última fase del diseño, realizaremos evaluaciones de usuarios con el prototipo final desarrollado en Android. Con esto comprobaremos la experiencia de los usuarios evaluados con el resultado final del proyecto y podremos ver los objetivos que han sido logrados y posibles mejoras y modificaciones futuras.

8.1. Propósitos y objetivos de la evaluación

El objetivo principal de la evaluación pretende poner a prueba la funcionalidad de la aplicación con los usuarios. Queremos analizar cómo se desenvuelven los usuarios con la aplicación y cómo utilizan las funcionalidades que ofrece. Para ello determinamos si se cumplen los siguientes objetivos:

- El usuario siente el poder total sobre la aplicación.
- El usuario es capaz de comprender las actividades que realiza.
- Se cumple el objetivo inicial del proyecto.

8.2. Preguntas de investigación

A continuación se expone un listado de preguntas a las que queremos responder con los resultados de la evaluación.

- ¿Es fácil distinguir los elementos de la aplicación?
 - ¿Se distinguen bien los iconos y símbolos?
 - ¿Es fácil acordarse de la ubicación de cada elemento?
- ¿Se entienden las transiciones entre pantallas?

CAPÍTULO 8. EVALUACIÓN CON USUARIOS

- ¿Puede saberse lo que esta realizando la aplicación por detrás?
- ¿Se pueden hacer las acciones con pocas pulsaciones?
 - ¿Cuánto se tarde en realizar una acción?
 - ¿Hay alguna tarea que requiera muchas pulsaciones?
- ¿Se pueden visualizar todas las opciones disponibles de un solo vistazo?
 - ¿Es complicado localizar algún elemento o información a simple vista?

8.3. Requisitos para los participantes de la evaluación

La evaluación será realizada por dos tipos objetivo de participantes que se corresponden con los definidos en la fase de modelado:

■ Ciudadano

- Ciudadano de a a pie inscrito en el censo con capacidad para votar en las próximas elecciones.
- Edad: a partir de 18 años.
- Familiarizados con el uso de herramientas móviles, preferiblemente en dispositivos Android.

■ Activista social

- Activista social que participa activamente en movimientos sociales afines a una causa.
- Edad: a partir de 18 años.
- Familiarizados con el uso de herramientas de participación ciudadana.

8.4. Diseño experimental

Las evaluaciones tendrán una duración estimada entre 4 y 7 minutos. Realizando un total de 5 evaluaciones, formadas por ciudadanos y activistas sociales mayores de 18 años. En cada una se deberán cumplir las siguientes normas y pasos:

1. Se dará una breve introducción a los participantes, explicándoles la temática de la aplicación y el objetivo que se persigue con ella.
2. El usuario recibirá una lista con las tareas a realizar durante la evaluación. Para llevarlas a cabo, el usuario evaluado podrá tomarse todo el tiempo que estime oportuno.
3. Durante la evaluación se mantendrá una reunión en un entorno cerrado libre de distracciones con la persona evaluada para observar todas las reacciones a la hora de realizar las tareas en la aplicación. El moderador también apuntará los tiempos que tarda en finalizar las tareas.
4. Si el usuario evaluado no pudiera resolver una tarea en menos de 3-4 minutos, o quedase atascado en la misma por un tiempo superior a 45 segundos, el moderador le indicará que pase a la siguiente tarea. Se anotarán con especial atención las tareas que no pudieran ser completadas con éxito durante la evaluación.
5. Al final de la sesión el moderador apuntará todas las objeciones de la evaluación, así como las conclusiones del mismo, para posteriormente ponerlas en común y analizarlas.

8.5. Lista de tareas a realizar

Este es el listado de tareas que deberá realizar el usuario en la aplicación en función del tipo:

- Ciudadano
 - Visualizar los diferentes programas políticos que se presentan a las elecciones.
 - Acceder a las secciones más debatidas, valoradas, comentadas, etc.

- Visualizar una sección de la categoría *cultura* y realizar un comentario en ella.
- Cambiar su nombre de usuario de ”Anónimo” por otro.
- Añadir una sección a mis favoritos.
- Visualizar sus secciones favoritas.
- Valorar una sección después de haberla leído y realizar un comentario valorando la sección.

■ **Activista Social**

- Visualizar el listado de propuestas publicadas en el sistema.
- Mostrar las propuestas de la categoría *educación* publicadas hasta el momento.
- Leer el contenido de una propuesta, valorarlo y realizar un comentario.
- Añadir una propuesta a mis favoritos.
- Visualizar sus propuestas favoritas.
- Publicar una nueva propuesta en la categoría *vivienda*.
- Colaborar en el desarrollo de una propuesta colaborativa.
- Publicar una propuesta colaborativa para que pueda desarrollarla la comunidad.

8.6. Entorno y herramientas que vamos a emplear

Para realizar las evaluaciones nos reunimos con los usuarios a evaluar en un entorno cerrado y libre de distracciones que le permitiera realizar las tareas de evaluación explicando en voz alta sus impresiones y problemas. Toda esta información era anotada para después ser debatida y contrastada con el usuario.

En cuanto a la aplicación, se utilizó un archivo *.apk* en modo *debug* que se le proporcionó al usuario evaluado para que la pudiera instalar en su dispositivo. Como requisito fundamental, el dispositivo de la persona evaluada debía contar con una versión de android igual o superior al API 15 (*Ice Cream Sandwich*), y poseer una conexión a internet estable para interactuar con la aplicación sin problemas.

8.7. Tareas del moderador

El moderador será el responsable de guiar el proceso de la evaluación con el usuario. Al comienzo de la evaluación el moderador solicitará al usuario la instalación de *DemoCritics* en su dispositivo. Durante la evaluación el moderador deberá realizar las siguientes acciones:

1. Mostrar al usuario la lista de tareas a realizar con la aplicación.. Se informará al usuario de que deberá realizar las tareas sin ayuda, ni interrupciones hasta el final.
2. Indicar al usuario que debe expresar en voz alta sus impresiones y problemas que encuentre durante la interacción con la aplicación.
3. El moderador interrumpirá al usuario por un motivo de fuerza mayor, es decir, si sucede algún acontecimiento que impida continuar con la evaluación (fallo en la aplicación, servidor caído, etcétera).
4. Si el usuario evaluado estuviera más de 2.5 minutos para resolver una tarea, o quedara atascado más de 45 segundos en la misma, el moderador comunicará al usuario que prosesa con la siguiente tarea.
5. El moderador tomará nota de los tiempos que el usuario tarda en realizar cada tarea. Así como también el tiempo total que dura la evaluación con todas las tareas realizadas.
6. Tras completar todas las tareas asignadas, iniciará una breve sesión de *debriefing* con el usuario para contrastar sus anotaciones.

8.8. Resultados de la evaluación con los usuarios

Se realizaron un total de 5 evaluaciones: 3 con usuarios "ciudadanos" y 2 con usuarios con un perfil de "activista social". En la mayoría de los casos se completaron las tareas asignadas con mayor o menor soltura en función de su experiencia previa con herramientas similares.

La mayor parte de los usuarios respondieron de forma fluída al uso de la aplicación. Bien porque al estar acostumbrados al utilizar aplicaciones en Android que utilizaran interfaces basadas en *Material Design*, los elementos visuales de la aplicación les resultaron familiares. Sin embargo, algunos

conceptos internos en la aplicación resultaron algo confusos para algunos usuarios.

Comenzando por los usuarios que representaban el papel de *ciudadano*, estos mostraban cierta confusión respecto a la representación de los programas políticos. Una de las motivaciones principales del proyecto es fomentar la lectura de los programas políticos por los electores, pues actualmente pocos conocen cómo se estructura un programa político en líneas generales. Ha sido un factor determinante la incomprendición de los usuarios al ver secciones que incluían texto, otras que no y otras que incluían enlaces a otras subsecciones. No obstante es un factor que viene condicionado de la estructura que ha establecido el partido político para su programa electoral. Por lo que podemos encontrar programas que se adapten mejor o peor a esta forma de representarlos.

Para los usuarios que jugaban el rol de *activista social* la interacción con la aplicación fue más fluída que el perfil anterior, pues normalmente estaban acostumbrados a herramientas relacionadas con el mundo de la política y la participación ciudadana. Para las propuestas normales los usuarios reaccionaron de forma normal, visualizando, creando, valorando o comentando las propuestas publicadas. Sin embargo, una vez más el concepto de *propuesta colaborativa* no fue del todo comprendido en un primer momento. Pero una vez que el usuario vió cómo su propuesta podía ser editada en tiempo real por otros usuarios, el concepto quedó más claro y les pareció bastante útil y llamativo.

8.8.1. Comentarios sobre la interfaz

Respecto a la interfaz de la aplicación, su actual diseño no representó muchos problemas graves. La selección de los iconos quizás no fue la mejor para representar los elementos de la aplicación. Algunos usuarios tuvieron que leer la descripción de cada elemento para saber a qué acción les llevaría pulsar ese elemento. En cuanto a la navegación y representación de los menús dentro de la aplicación, los usuarios no tuvieron demasiados problemas por estar familiarizados con *Material Design* y el botón "hamburguesa" de menú. Por otro lado hubo usuarios que no entendieron a primera vista el uso de los botones de like, dislike, etc. pues no les quedó claro que pudieran pulsarlos para opinar. Además, el índice desplegable accesible en la parte superior derecha de las secciones solo fue utilizado por uno de los usuarios, pues los otros no se dieron cuenta de su existencia.

Por otro lado los usuarios entendieron bien la navegación por tabs, y todos

CAPÍTULO 8. EVALUACIÓN CON USUARIOS

Tabla 8.1: Dificultad y tiempos de las tareas para el usuario del tipo *ciudadano*.

Tarea	Dificultad	Tiempo
Visualizar los diferentes programas políticos que se presentan a las elecciones.	1/5	<30 s
Acceder a las secciones más debatidas, valoradas, comentadas, etc.	2/5	<30 s
Visualizar una sección de la categoría <i>cultura</i> de un partido político en concreto.	3/5	<45 s
Añadir una sección a mis favoritos.	2/5	<15 s
Visualizar tus secciones favoritas.	4/5	<45 s
Valorar una sección después de haberla leído y realizar un comentario valorando la sección.	1/5	<45 s

valoraron positivamente la opción de explorar el contenido por categorías y de guardar secciones o propuestas en favoritos. También destacaron la elección de distintos colores para distinguir las distintas partes de la aplicación.

Sin embargo, en el caso de la edición de propuestas colaborativas, el no poder diferenciar a los usuarios que estaban escribiendo en un mismo *pad* al mismo tiempo es un problema del que se quejaron varios usuarios, pues algunos estaban acostumbrados a utilizar herramientas como Google Docs que si que los distinguen. Diferenciar a los usuarios que están escribiendo al mismo tiempo por colores o marcadores ayuda a poder concentrarse en la escritura y a poder distinguir el contenido de todo el documento. No obstante, les gustó el uso de edición en tiempo real para colaborar unos con otros.

8.8.2. Resultados de las tareas

La mayor parte de las tareas fueron completadas con éxito. En las tablas 8.1 y 8.2, podemos ver el tiempo promedio que tardaron los usuarios en realizar las tareas propuestas.

En cada tabla se muestra el tiempo promedio en **segundos** que los usuarios consumieron en llevar a cabo la tarea, y la dificultad media sobre un total de **cinco puntos** (siendo 0 una tarea fácil de realizar y 5 una tarea compleja de realizar) que costó a los usuarios realizar las tareas.

CAPÍTULO 8. EVALUACIÓN CON USUARIOS

Tabla 8.2: Dificultad y tiempos de las tareas para el usuario del tipo *activista social*.

Tarea	Dificultad	Tiempo
Visualizar el listado de propuestas publicadas en el sistema.	1/5	<15 s
Mostrar las propuestas de la categoría {educación} publicadas hasta el momento.	3/5	<30 s
Leer el contenido de una propuesta, valorarlo y realizar un comentario.	2/5	<45 s
Añadir una propuesta a mis favoritos.	2/5	<15 s
Visualizar tus propuestas favoritas.	2/5	<15 s
Publicar una nueva propuesta en la categoría {vivienda} en el sistema.	3/5	<45 s
Colaborar en el desarrollo de una propuesta colaborativa.	4/5	<60 s
Publicar una propuesta colaborativa para que pueda desarrollarla la comunidad.	3/5	<30 s

8.8.3. Informe de hallazgos y recomendaciones

En líneas generales podemos decir que la aplicación ha sido recibida de forma positiva por la mayor parte de los usuarios, siendo la innovación en torno a la idea el factor potencial de la aplicación. Algunos de los usuarios estaban familiarizados con herramientas similares en otros entornos (web), y valoraron positivamente su traslado a una plataforma móvil.

Los usuarios valoraron la posibilidad de poder leer los programas en el móvil, como si de un programa de bolsillo se tratase, y poder valorarlos y opinar sobre ellos. No obstante, la apariencia o representación de los programas es algo que no ha atraído demasiado su atención. Por ello, deberemos trabajar más en otra posible representación gráfica que capte más la atención del usuario.

La interacción entre las tareas y la transición de las pantallas, no ha supuesto un problema para los usuarios. A excepción de las propuestas colaborativas, pues habría que rediseñar el concepto para hacer la edición en tiempo real más *amigable* en consonancia con otras herramientas similares.

Para la siguiente interacción del proceso de desarrollo, se proponen los siguientes puntos por orden de prioridad, teniendo en cuenta las recomendaciones y evaluaciones de los usuarios:

CAPÍTULO 8. EVALUACIÓN CON USUARIOS

1. Cambiar el aspecto de los iconos de la pantalla inicial por otros más representativos a la actividad que respresentan.
2. Dar más visibilidad a los botones de opinión en secciones y propuestas.
3. Rediseñar la visualización de las secciones para una lectura más cómoda y diferenciar bien los elementos con los que puede interactuar el usuario de los que no.
4. Estudiar la visibilidad del índice de programa expansible en cada sección.
5. Mejorar el concepto de propuestas colaborativas con diferenciación de usuarios por colores o etiquetas para mejorar su comprensión durante su edición.
6. Estudiar la creación de nuevas categorías que representen mejor las prouestas y secciones de la aplicación.
7. Ofrecer más funcionalidades de personalización al usuario.

Capítulo 9

Resultados y Trabajo Futuro

En este capítulo analizaremos y discutiremos los resultados en conjunto de todo el Trabajo de Fin de Grado, las conclusiones que se pueden sacar tras su desarrollo y algunas líneas de trabajo futuro para este proyecto.

Todo el código open-source desarrollado por nosotros está disponible en el GitHub de la organización del proyecto:

<https://github.com/Zorbel>

9.1. Discusión de Resultados

Inicialmente este proyecto nació para migrar la tecnología de colaboración en tiempo real de Apache Wave, presente en el proyecto SwellRT para plataformas web, a dispositivos móviles Android para posteriormente desarrollar una app que hiciera uso de esta tecnología. No habíamos trabajado nunca con tecnologías en tiempo real, así que tuvimos que realizar una pequeña investigación previa de los protocolos implicados (XMPP y WebSockets) y del funcionamiento de Wave para ponernos al día. Además trabajamos junto a uno de los desarrolladores de SwellRT, que nos orientó a la hora de "navegar" por el código de la versión web para identificar las incompatibilidades con Android y subsanarlas. En poco tiempo tuvimos lista una versión simple del cliente capaz de conectarse al servidor Wave desde Android. Se trata de un desarrollo en código libre, por lo que realizamos una aportación a un proyecto que cualquiera podría utilizar en su propio desarrollo Android. El resultado está disponible en el GitHub de SwellRT:

<https://github.com/P2Pvalue/swellrt/tree/master/android>

Tocaba entonces embarcarse en la el desarrollo de la idea de alguna aplicación que hiciera uso de la tecnología de SwellRT. Se nos dejó total libertad para empezar este proyecto de aplicación desde cero y decidimos que si hacíamos algo tenía que tener utilidad en el mundo real, más allá del alcance de este Trabajo de Fin de Grado. A continuación discutimos el por qué de

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

DemoCritics y la utilidad de los resultados obtenidos.

9.1.1. El porqué de DemoCritics

El desarrollo de herramientas que permitan explorar nuevas formas de colaborar, participar y expresar opiniones, o mejorar las herramientas ya existentes, siempre es un reto. Esto es así dada la gran cantidad de soluciones disponibles en la actualidad y ligadas sobre todo a la expansión que vivimos en los últimos años del uso para estos propósitos de Internet, que se ve reflejada en gran medida en las Redes Sociales. Cualquiera con conexión a la red tiene acceso a grandes cantidades de información que puede compartir y sobre la que puede opinar, ya no solo desde un ordenador sino también desde los "ordenadores de bolsillo" llamados smartphones que llevamos con nosotros prácticamente a todas partes.

Más específicamente, si nos centramos en el ámbito de la participación política y ciudadana, podemos detectar la reciente proliferación de herramientas que proporcionan métodos para organizarse, expresar opiniones, ponerse de acuerdo y construir nuevos proyectos políticos de forma más a menudo eficaz y eficiente que si se hiciera como tradicionalmente: "cara a cara". No hay más que ver cómo las nuevas formaciones y movimientos sociales, que tan a la orden del día están con las recientes elecciones, hacen cada vez más uso de las nuevas tecnologías (como las descritas en el Estado del Arte de este documento) para organizarse y darse a conocer.

Pero no solo los nuevos movimientos lo hacen. Los movimientos tradicionales (como los partidos políticos) son también conscientes de la importancia de estar presente en la red y adaptarse a las nuevas formas de comunicación. Así, encontramos que su presencia en Redes Sociales y la existencia de aplicaciones móviles para publicitar su mensaje son cada vez mayores. Sin embargo, ¿No se supone que su principal mensaje está contenido en su programa electoral? ¿Dónde puedo ver ese mensaje? La práctica más común hoy en día es la de colgar dicho programa electoral en su página web en forma de "megadocumentos" en PDF de gran extensión (los hay de hasta 200 páginas). Resultado: poca gente sabe de su existencia en las webs y los que lo saben rara vez se lo leen debido a su gran tamaño.

Tomando todo lo anterior en consideración hemos desarrollado **DemoCritics**, una plataforma para dispositivos móviles Android cuyo objetivo es **juntar en un único sitio el mensaje de la política tradicional (visto en forma de su Programa Político) con el mensaje de los ciudadanos (visto en forma de Propuestas)**.

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

Tratándose de una idea de proyecto desarrollado desde cero por nosotros, quisimos utilizar una metodología que nos permitiera investigar y diseñar una aplicación que realmente satisficiera necesidades reales y actuales. No obstante, a ambos simplemente nos interesaba la política. Ninguno de nosotros tenía el bagaje político necesario para poder considerarnos capaces de afirmar que nuestras ideas iniciales eran realmente viables a la hora de crear una aplicación que utilizara el público en general. Para ello fueron muy importantes las dos entrevistas que tuvimos oportunidad de hacer a personas involucradas en la participación política como fueron los chicos de Labodemo y Javier de la Cueva. De esta fase de investigación desecharmos algunas ideas iniciales y extrajimos otras nuevas más interesantes, plasmándose algunas de ellas en el estado actual de DemoCritics.

9.1.2. Usando Wave/SwellRT en DemoCritics

No nos podemos olvidar del "leitmotiv" inicial de este proyecto, utilizar las capacidades de colaboración en tiempo real de Wave/SwellRT en Android. Por ello lo primero que pensamos fue generar las propuestas ciudadanas de forma colaborativa en un texto editable por cualquier usuario en tiempo real. Sin embargo, tras la discusión en las entrevistas entendimos que esto podría ser algo caótico a la hora de generar una propuesta útil, ya que al tratarse de algo tan subjetivo y personal, alcanzar un consenso era complicado.

En su lugar, identificamos un problema de las propuestas que actualmente se realizan en la red: es muy fácil proponer algo pero a menudo la gente no tiene (ni tampoco es obligatorio que los tenga) los conocimientos necesarios para elaborar una propuesta viable y factible de ser llevada a cabo en un contexto actual. Al final, ¿qué diferencia una propuesta de realizar un comentario? ¿simplemente su longitud, que a priori es más larga en la propuesta? Tras debatir esta cuestión con Javier de la Cueva dimos con una solución a este problema. **Si queremos que las propuestas sean algo serio y razonablemente viable, ¿por qué no hacer que las propuestas vayan más allá de una declaración de intenciones y tengan asociadas campos que expliquen cómo se llevarían a cabo en la situación actual?** En este sentido identificamos dos elementos que nos parecieron clave para ello: que el usuario tras exponer su propuesta explicara "¿Cómo la llevaría a cabo?" (medidas a adoptar) y "¿Cómo la financiaría?" (necesidades de carácter monetario). Sin embargo un usuario no tiene por qué ser experto en estos temas ni tener muy claro cómo hacerlo, así que ¿cuál es la solución?.

Llegados a este punto retomamos la idea de usar Wave en las propuestas.

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

Pero ahora no para redactar propuestas en su totalidad por parte de muchos usuarios, si no específicamente para **fomentar la colaboración entre usuarios**. Es muy común escuchar hablar de gente que quiere involucrarse en alguna tarea pero no sabe cómo puede echar una mano. Bien, pues con DemoCritics daríamos solución a las propuestas que el usuario no supiera cómo llevar a cabo poniéndole en contacto con gente más versada en esos temas y que quisiera colaborar y echarle una mano.

Así, nacieron las **propuestas colaborativas** (o "abiertas"). Propuestas en las que un usuario al crearlas dejaba alguno de los dos campos ("¿Cómo la llevaría a cabo?" y "¿Cómo la financiaría?") en blanco para permitir a otros usuarios que le echaran una mano mediante la edición colaborativa y en tiempo real de esa parte de la propuesta. De esta manera utilizamos Wave/SwellIRT en un contexto más lógico que el de redactar la propuesta entera, pues al menos dichos campos están sujetos a menos subjetividad que la declaración de intenciones de una propuesta más al uso de las que se encuentran en las plataformas actuales. Además, únicamente el usuario inicial de la propuesta es el único capaz de convertir la propuesta en "definitiva", cerrando la posibilidad de su edición con Wave cuando estime que la comunidad de "expertos" ha podido ayudarle a proponer algo coherente y viable.

9.1.3. Incentivando el uso social de DemoCritics

Por otro lado, hemos querido aprovechar el auge de las redes sociales para "socializar" la plataforma. De esta forma los usuarios pueden generar contenido e interés en el uso de la aplicación. Pueden dar su opinión mediante el uso de "indicadores sociales" similares a los "Me gusta" o No me gusta" tan presentes en las redes sociales. También pueden realizar sus propios comentarios para generar debate y marcar contenido como favorito. Todo ello aplicado tanto a secciones de programas políticos como a propuestas hechas por otros usuarios. Esto, por otro lado, nos permite en cierta medida "cuantificar" el interés de las personas, lo cual siempre resulta útil para clasificar el contenido en forma de "tops" de lo más valorado, lo más comentado, etc.

Aunque no solo es de utilidad para elaborar dichos filtros de visualización que atraigan la atención de los usuarios, sino que también los propios partidos políticos podrían beneficiarse de ello para identificar sus propuestas políticas más controvertidas o más valoradas. Al final se trata de una especie de "red social" aplicada al ámbito político en el que cualquiera de los actores puede intervenir y beneficiarse del contenido que genera la comunidad.

También utilizamos un sistema de categorización tanto de secciones como

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

de propuestas para llevar a cabo esa unión entre los programas políticos y las propuestas que tan necesaria nos parecía para fomentar tanto la lectura de secciones como la elaboración de propuestas. De hecho, esto atraería a usuarios que, lejos de querer leerse un programa político, lo que buscan es conocer el contenido de una determinada categoría de su interés. Aunque actualmente existen 6 categorías (Sanidad, Educación, Empleo, Vivienda, Impuestos y Cultura) la intención es que existan más y que incluso los propios usuarios interesados puedan crear sus propias categorías, tal y como se discutirá después en el trabajo a futuro.

9.1.4. Aspectos técnicos de DemoCritics

Desde un punto de vista técnico lo primero que el desarrollo de DemoCritics nos ha exigido ha sido investigar un poco mejor el funcionamiento de la plataforma Android, tecnología para la que apenas habíamos desarrollado antes. Concretamente hemos tenido que indagar en aspectos tales como la conectividad con la red, el diseño y la interacción con la interfaz de usuario, la gestión de hilos y procesos, acceso a información del dispositivo móvil, etc. Afortunadamente no nos ha suscrito mucho problema, pues la documentación oficial disponible por parte de Google es bastante amplia y explicativa. El resultado de esta aplicación se encuentra disponible para la comunidad en GitHub:

<https://github.com/Zorbel/appTFG>

Por otro lado, el diseño de la arquitectura del sistema nos llevó a plantearnos cómo desarrollar una plataforma que no nos restringiera su uso exclusivamente a Android. Queríamos que la interfaz de los datos fuera independiente de los datos en sí. En consecuencia, y tras evaluar distintas opciones, desarrollamos un Service REST que accediera a la Base de Datos. Este Service REST hace de intermediario entre las peticiones HTTP recibidas y el acceso a Base de Datos. De esta manera cualquier tipo de cliente con posibilidad de conexión a Internet puede acceder a dichos datos mediante peticiones HTTP. Los datos son entonces devueltos en formato JSON y tratados por el cliente.

Ambos teníamos conocimientos previos de Bases de Datos como para desarrollar la parte del almacenamiento de los datos, pero ninguno de los dos conocía la tecnología detrás del Service REST. Sin embargo, si que teníamos algunos conocimientos de programación en PHP. Por tanto, dentro de la convicción de utilizar software libre, buscamos un framework que pudiera agilizar el desarrollo del Service REST en PHP y una plataforma online donde alojarlo. Como resultado utilizamos el framework open-source Laravel 5 para

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

desarrollar el Service REST que alojamos en el servidor gratuito de RedHat OpenShift.

El Service REST es accesible desde la siguiente URL:

<https://apptfg-servicerest.rhcloud.com/>

Asimismo, el código desarrollado para el Service REST está disponible en el GitHub del proyecto:

<https://github.com/Zorbel/ServiceRest>

Por otro lado, el uso de la tecnología de Wave/SwellRT no ha supuesto un gran hándicap pues disponíamos de la ayuda del desarrollador del proyecto original de SwellRT, Pablo Ojanguren. Fue necesario eso sí estudiar el funcionamiento del servicio Android de SwellRT para familiarizarnos con su estructura de callbacks y operaciones contra el servidor wave: creación de usuarios, gestión de participantes, creación de waves, modificación de waves, etc.

En el estado actual de la arquitectura del sistema detrás de DemoCritics, se ha comprobado que el cliente Android interactúa de forma correcta con el Service REST y la base de datos. Asimismo la conexión con el servidor Wave también se establece y la creación y edición de propuestas colaborativas es plenamente funcional. Se pronpondrán después en el trabajo futuro posibles mejoras a este sistema.

9.2. Reparto de Trabajo entre los componentes del grupo

DemoCritics es un proyecto desarrollado por dos personas. Para su desarrollo hemos elegido una metodología de control de versiones que nos permite repartir las tareas de forma eficaz (cualquiera de los dos puede ver en pocos pasos lo hecho por el otro) y eficiente (lleva poco tiempo subir o descargarse los cambios). Para ello creamos una organización en GitHub llamada "Zorbel" y gestionamos con git los cambios en el código de cualquiera de las partes que conforman DemoCritics. Además, esto nos permitió también facilitar la revisión del código por parte de los profesores del proyecto.

En general, todas las tareas se han llevado a cabo en forma paralela y con una carga de trabajo equitativa entre ambos miembros. A continuación se detallan un poco más estas tareas:

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

- **Migración de SwellRT a Android:** Tanto la investigación del funcionamiento de la tecnología como la metodología de migración se llevaron a cabo por ambos miembros. Estos cambios se contrastaron en reuniones con el director Pablo Ojanguren (desarrollador de SwellRT) a medida que se producían. De poner el API de trabajo con waves de SwellRT sobre el servicio migrado se encargó Pablo.
- **Desarrollo de la idea de la app:** los dos miembros del proyecto nos reunimos con los directores Samer y Pablo para hacer una sesión de brainstorming y dar con una idea de aplicación que a ambos nos gustaría realizar y que hiciera uso de SwellRT.
- **Diseño e Investigación:** ambos hicimos reuniones periódicas para discutir ideas iniciales, planificar la investigación y diseñar prototipos. Durante las entrevistas de investigación procuramos estar ambos presentes para exponer las ideas, discutirlas y enseñarles los prototipos básicos a los entrevistados. Posteriormente nos reunimos para sacar conclusiones y modificar los prototipos. En definitiva, toda la fase basada en el Diseño Guiado por Objetivos la realizamos en conjunto.
- **Implementación de la aplicación:** En este caso sí que hubo un reparto de tareas más marcado por la poca experiencia previa que teníamos con distintas tecnologías. El diseño general de la aplicación, el Service REST y la Base de Datos lo discutimos entre ambos. A la hora de implementarlo uno de nosotros (Javier) conocía mejor la arquitectura de Android y se dedicó a elaborar el funcionamiento del cliente mientras que el otro (Jaime) había trabajado previamente con Laravel y se dedicó a investigar y desarrollar el Service REST y la conexión con la Base de Datos. No obstante, ambos pusimos en común los cambios realizados, de manera que los dos tuvimos un conocimiento de cómo funcionaba la aplicación en su conjunto. Durante la última fase del proyecto de perfeccionamiento de detalles de la aplicación ambos trabajamos para mejorar el cliente Android.
- **Evaluación con Usuarios:** planificamos juntos la dinámica de las evaluaciones y realizamos dos evaluaciones con usuarios cada uno de un total de 4 evaluaciones, procurando poner luego en común los resultados para discutirlos y sacar conclusiones.
- **Elaboración de la memoria:** para la redacción de esta memoria procedimos a realizar varios posibles índices para su estructuración hasta que obtuvimos un primer esqueleto de su estructura. Posteriormente nos repartimos su redacción, procurando verificar con el compañero el contenido de todo lo escrito. Esto se vió facilitado por el uso también para la memoria de git GitHub como sistema de control de versiones (ver organización del proyecto)

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

en GitHub). En general, habremos escrito ambos aproximadamente la mitad del contenido de este documento cada uno (siendo siempre verificado por el otro).

Para el seguimiento del proyecto procuramos realizar mínimo una reunión al mes, ya fuera con Samer, con Pablo o con ambos. Cabe destacar que, si bien a veces no pudimos reunirnos en persona, el uso de herramientas de mensajería instantánea (con especial mención a Telegram) y de videollamada como Skype y Hangouts fue constante entre los involucrados en el proyecto.

9.3. Trabajo Futuro

Como ya se ha comentado con anterioridad la presente es una primera versión de DemoCritics que nos sirve para tener una primera prueba de concepto de algunas de sus funcionalidades y que haga uso de la migración de SwellRT a Android. En esta versión hemos podido evaluar el funcionamiento general de la idea y ponerla a prueba con algunos usuarios para mejorar su utilidad de cara a continuar su desarrollo. Nuestro objetivo en este momento es aprovechar la oportunidad que se nos presenta en este año electoral para poner realmente a prueba esta plataforma en las próximas elecciones generales. De esta forma también podríamos analizar su uso antes, durante y después de unas elecciones.

Para ello nos proponemos seguir poniéndonos en contacto con actores relacionados con el mundo de la política (activistas, políticos, periodistas, académicos, etc.) para realizar más entrevistas y someter la plataforma a nuevas iteraciones del proceso de diseño, implementación y evaluación con los que ir poco a poco mejorando sus características. De paso también iremos generando interés entre los entrevistados y publicitando su uso.

En cualquier caso, el proyecto está disponible en forma de software libre bajo licencia GNU GPLv3 que cualquiera puede estudiar, copiar y modificar para desarrollar sus propias ideas inspirado por las nuestras.

9.3.1. Mejoras a la versión actual

Teniendo en cuenta los resultados de la evaluación con usuarios creemos que las líneas de trabajo futuro deberían ir orientadas a mejorar la experiencia de usuario, modificando aspectos de la interfaz gráfica que pudieron resultar más incomprensibles a las personas evaluadas, como los iconos de la aplicación y

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

su significado. También habría que mejorar la visualización de las distintas secciones del programa, que resultó confusa para algunos de ellos.

Desde el punto de vista del desarrollo colaborativo en tiempo real de propuestas, se debería estudiar la posibilidad de poder distinguir a los usuarios participantes en la Wave y lo que escriben mediante el uso de indicadores de color o etiquetas al estilo de otras plataformas web como Google Docs.

9.3.2. Nuevas características

Durante el desarrollo del proyecto han ido surgiendo diversas ideas interesantes que se podrían aplicar en un futuro próximo dentro de la plataforma de DemoCritics. Algunas de las posibilidades son:

- Desarrollo de Encuestas: de intención de voto, de afinidad, etc. con posibilidad de obtener los resultados en tiempo real mediante SwellRT.
- Abrir la creación de categorías nuevas a los usuarios, de manera que demos más flexibilidad a la hora de crear y clasificar contenidos.
- Permitir que los usuarios hagan comparativas por temas entre las secciones de los distintos programas de partidos políticos. Buscar para ello la participación de colectivos sociales interesados.
- Creación de una hemeroteca de programas políticos que permita navegar por programas políticos de anteriores elecciones. Útil por ejemplo para comprobar la evolución de las propuestas de los partidos.
- Integración con redes sociales y mensajería instantánea para compartir contenidos de la aplicación.
- Organizar jerárquicamente los usuarios, pudiendo existir grupos de usuarios que comparten las mismas inquietudes e intereses.
- Uso de inteligencia artificial para integrar sistemas recomendadores de contenido por afinidad o de puesta en contacto entre usuarios por intereses similares.

Asimismo, y gracias a la flexibilidad que nos permite el uso de un Service REST, se estudiará la posibilidad de **desarrollar un cliente web** para la aplicación, especialmente pensado para usuarios que están más familiarizados con la elaboración de textos largos, como los de las propuestas, usando un teclado. De esta manera enfocaríamos la aplicación móvil a la lectura de contenidos y generación de opiniones y comentarios, mientras que las propuestas se podrían redactar en el ordenador. Entendemos que la postura de

CAPÍTULO 9. RESULTADOS Y TRABAJO FUTURO

uso de un móvil es para usarlo durante un espacio corto de tiempo, por lo que la existencia de un cliente web para ordenadores haría la generación de contenidos más serios y extensos, como las propuestas, mucho más cómoda.

Por otro lado, en el desarrollo de este proyecto hemos centrado la plataforma en los programas políticos, pero se podría utilizar para mostrar otros tipos de documentos estructurados susceptibles de ser sometidos a debate y opinión. Como por ejemplo normativas, leyes, manuales, etc.

9.3.3. SwellRT Android

El trabajo realizado para hacer compatible la tecnología de colaboración en tiempo real de Wave presente en SwellRT con dispositivos Android, aporta por primera vez la posibilidad de utilizar estas características de forma nativa en Android en un API de software libre. Hasta este momento las soluciones disponibles pasaban por utilizar librerías privativas como la de Google (Real Time API). Esta nueva aportación se encuentra ahora disponible para los desarrolladores de Android en el GitHub del proyecto de SwellRT: <https://github.com/P2Pvalue/swellrt/tree/master/android>

Es además plenamente compatible con el cliente web, por lo que las posibilidades de desarrollo de aplicaciones multi-plataforma son claras. Además, en este proyecto se ha optado por realizar una pequeña prueba de concepto con edición de texto en tiempo real, pero el desarrollo de plataformas colaborativas en tiempo real va más allá e incluye aplicaciones tan diversas como herramientas de dibujo, visualizado de contenidos (como vídeo y música) de forma sincronizada, mapas con información en tiempo real, generación de estadísticas en tiempo real, ...

En definitiva, **se ha puesto la colaboración en tiempo real a disposición de la comunidad de desarrolladores open-source Android para que hagan uso de estas funcionalidades en sus futuras aplicaciones.**

Capítulo 10

Conclusiones

Desarrollar un proyecto como DemoCritics no es una tarea baladí. Al tratarse de una aplicación que parte de cero desde nuestra propia idea e iniciativa, exige bastante dedicación en forma de tiempo y motivación. Más aún si hablamos de una temática como es la política, que desgraciadamente parece no atraer toda la atención e interés de las personas que relamente debería tener como actividad de la que formamos parte como sociedad. En este sentido el tiempo dedicado a las labores previas de planificación, investigación, diseño y evaluación de prototipos era necesario para establecer unas bases sólidas sobre las que asentar la futura implementación de características que resultaran a la vez útiles y atractivas para el usuario.

Este proceso fue bastante productivo y nos sirvió para depurar y pulir aspectos previos de nuestra idea inicial. Fue también enriquecedor para nosotros como experiencia, pues no hizo sino demostrarnos la importancia de que el desarrollador, acostumbrado normalmente a tener una visión bastante reducida a aspectos técnicos, sea consciente de la necesidad de implicar en el desarrollo otras voces. Dichas voces quizás no posean los conocimientos técnicos necesarios para "implementar" la idea, pero sí que aportan una visión de conjunto más amplia que ayuda a entender mejor las necesidades reales del usuario, que a fin de cuentas es el destinatario de la aplicación. Realizar por tanto entrevistas y posteriores evaluaciones para tener en cuenta la opinión del futurable "cliente" nos ha enfrentado a una realidad necesaria para cualquier desarrollador que quiera conseguir un producto serio. Más aun cuando nos planteamos DemoCritics como una aplicación que trascendiera el alcance del Trabajo de Fin de Grado y se convertiera en una herramienta útil en el futuro.

DemoCritics ha supuesto también investigar el uso de tecnologías que no conocíamos, con el consiguiente proceso de aprendizaje que ello conlleva. Hemos tenido que enfrentarnos al estudio del funcionamiento de una plataforma ya establecida como Wave/SwellRT para llevar a cabo una migración a Android. Hemos trabajado el uso de tecnologías del lado del cliente como las conexiones desde Android y del servidor como la implementación del Ser-

CAPÍTULO 10. CONCLUSIONES

vice REST. Pero sobre todo, hemos aprendido a llevar una metodología de trabajo basada en el control de versiones mediante Git y GitHub.

Los resultados obtenidos se plasman en el estado actual de DemoCritics: una aplicación capaz de juntar en un solo sitio la navegación por las secciones de programas políticos y crear propuestas, con la destacada opción de hacer esto último entre varias personas de forma colaborativa y en tiempo real utilizando la tecnología de SwellRT en Android. Explotamos también el potencial social de la aplicación mediante la posibilidad de opinar, realizar comentarios y marcar como favorito, en todo el contenido de la aplicación.

Se trata de un proyecto en el que aportamos una solución de software libre que no existía previamente, pues las opciones de colaboración en tiempo real en Android pasaban únicamente por soluciones privativas como el Real Time API de Google. Su código está disponible en GitHub y que cualquiera puede estudiar, contribuir, copiar y utilizar libremente. Tanto en el caso de la migración de SwellRT a Android como en el de la plataforma DemoCritics (la App, la base de datos y el Service REST).

Queda aún trabajo por hacer, pues la presente plataforma es una primera versión de un proyecto en el que pretendemos seguir investigando y desarrollando más funcionalidades, tal y como se discutirá a continuación en el Trabajo Futuro.

Apéndice A

Listados de Código Fuente

A.1. Migración de Wave a Android: SwellRT

A.1.1. Esquema de conexión 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
        .
    }
}
```

A.1.2. Esquema de conexión con AsyncTask

```

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
        }
    }
}

```

A.1.3. Esquema de uso de WAsync

```

//Create the atmosphere client
AtmosphereClient client = ClientFactory.getDefault().
    newClient(AtmosphereClient.class);

//Configure client with URL
AtmosphereRequestBuilder requestBuilder = client.
    newRequestBuilder()
    .method(Request.METHOD.GET).trackMessageLength(true).uri(
        WaveSocketWAsync.this.urlBase)
    .transport(Request.TRANSPORT.WEBSOCKET)

```

APÉNDICE A. LISTADOS DE CÓDIGO FUENTE

```
//Create and configure socket
WaveSocketWAsync.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>() { // 
    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()
// 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
}
});

try {
// connect to the server
socket.open(requestBuilder.build());
} catch (IOException e) {
    // catch possible exceptions
}

//send a given message to the server, equivalent to GWT send(
msg)
socket.fire(Data);
```

A.2. Service REST (Laravel)

A.2.1. Esquema de Rutas Simple

```
// 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}
```

A.2.2. Esquema de Rutas Agrupadas

```
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');
});
```

A.2.3. Esquema de Rutas de Propuestas Agrupadas

```
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');
    });
});
```

APÉNDICE A. LISTADOS DE CÓDIGO FUENTE

A.2.4. Esquema de Controlador de Propuestas

```
<?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`');
    }
}
```

A.3. Aplicación Android

A.3.1. Función de construcción del árbol de Programa Político

```
protected int createIndex(Section parent, List<Section>
JSONResult, int index) {

    // X = currentSection
    Section currentSection = JSONResult.get(index);

    if (parent.getSections() == null) {
        parent.setSections(new ArrayList<Section>());
    }

    parent.addSubSection(currentSection);

    // c = nextIndex
    int nextIndex = index + 1;

    while ((nextIndex < JSONResult.size()) && (getLevel(
        JSONResult.get(nextIndex)) >= getLevel(currentSection)
    )) {
```

APÉNDICE A. LISTADOS DE CÓDIGO FUENTE

```
    if (getLevel(JSONResult.get(nextIndex)) == getLevel(
        currentSection)) {
        currentSection = JSONResult.get(nextIndex);
        nextIndex++;
        parent.addSubSection(currentSection);

    } else if (getLevel(JSONResult.get(nextIndex)) >
        getLevel(currentSection))
        nextIndex = createIndex(currentSection, JSONResult,
            nextIndex);
    }

    return nextIndex;
}

protected int getLevel(Section sec) {

    int id_sec = sec.getmSection(), level;

    if (id_sec % 100 != 0) {
        level = 4;

    } else if (id_sec % 10000 != 0) {
        level = 3;

    } else if (id_sec % 1000000 != 0) {
        level = 2;

    } else
        level = 1;

    return level;
}
```

A.3.2. Registrar Usuario en SwellRT

```
public class MainActivity implements ServiceConnection,
    SwellRTServiceCallback {

    private SwellRTService mSwellRT;

    protected void bindSwellRTService() {

        if (mSwellRT == null) {
            final Intent mWaveServiceIntent = new Intent(this
                , SwellRTService.class);
```

APÉNDICE A. LISTADOS DE CÓDIGO FUENTE

```
        bindService(mWaveServiceIntent, this, Context.  
                    BIND_AUTO_CREATE);  
    }  
  
}  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    //Create view and get view references  
  
    bindSwellRTService(); //Bind to the service  
  
}  
  
//AsyncTask for registering the user.  
public void registerWaveUser() {  
  
    AsyncTask mRegisterTask = new AsyncTask<String, Void,  
    Boolean>() {  
  
        @Override  
        protected Boolean doInBackground(  
            String... params) {  
            return mSwellRT.registerUser(params  
                [0], params[1], params[2]);  
        }  
  
        @Override  
        protected void onPostExecute(Boolean  
            result) {  
            if (result)  
                Toast.makeText(MainActivity.this, "  
                    User created successfully", Toast.  
                    LENGTH_LONG).show();  
            else  
                Toast.makeText(MainActivity.this, "Error  
                    creating user", Toast.LENGTH_LONG).show();  
        }  
    };  
  
    //Execute task with the server, user and  
    //password  
    mRegisterTask.execute(WAVE_SERVER,  
        "" + User.ID_USER + "@local.net", "password")  
        ;  
}
```

APÉNDICE A. LISTADOS DE CÓDIGO FUENTE

```
//Callback called when the service is connected.  
@Override  
public void onServiceConnected(ComponentName name,  
IBinder service) {  
    mSwellRT = ((SwellRTService.SwellRTBinder) service).  
        getService(this);  
    Log.d(this.getClass().getSimpleName(), "SwellRT\u2014  
        Service\u2014Bound");  
  
    registerWaveUser(); //Do the register.  
}  
  
@Override  
public void onServiceDisconnected(ComponentName name) {  
    mSwellRT = null;  
    Log.d(this.getClass().getSimpleName(), "SwellRT\u2014  
        Service\u2014unBound");  
}  
}
```

A.3.3. Crear nueva Wave (Model) en SwellRT

```
public class NewProposalActivity extends  
    SwellRTActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState)  
    {  
        //Get the proposal info from the user view,  
        if there is no ''how'' or ''cost'', create  
        a new wave model.  
    }  
  
    public void doStartSession() {  
        // Open Session with ID_USER  
        try {  
            mSwellRT.startSession(MainActivity.WAVE_SERVER,  
                "" + User.ID_USER + "@local.net", "  
                password");  
        } catch (MalformedURLException e) {  
            e.printStackTrace();  
        } catch (InvalidParticipantAddress  
            invalidParticipantAddress) {  
            invalidParticipantAddress.printStackTrace();  
        }  
    }  
}
```

APÉNDICE A. LISTADOS DE CÓDIGO FUENTE

```
}

// SwellRT Service Callbacks

@Override
public void onServiceConnected(ComponentName name,
    IBinder service) {
    mSwellRT = ((SwellRTService.SwellRTBinder) service).
        getService(this);
    doStartSession();
    Log.d(this.getClass().getSimpleName(), "SwellRT\u2014
        Service\u2014Bound");
}

@Override
public void onStartSessionSuccess(String session) {
    getService().createModel();
}

@Override
public void onCreate(Model model) { // Callback called
    when a model is created.

    // Get the Id of the wave/model to store in the
    // database
    WaveId idW = model.getWaveId();

    String idWave = idW.toString();

    String id = "";
    id = idWave.substring(8, idWave.length() - 1);

    //Store the proposal (with the Wave Id) in the
    //database.
    postCollaborativeProposal(id);

    // Create a text document for each Pad
    TextType padHow = model.createText("Cmo\u2014\u2014lo\u2014
        har as?");
    TextType padCost = model.createText("Cmo\u2014\u2014lo\u2014
        financierias?");

    // Include the document as part of the wave/model in
    // the "root" map.
    if(createHowWave)
        model.getRoot().put("padHow", padHow);

    if(createCostWave)
```

```
        model.getRoot().put("padCost", padCost);

        // Allow current user to read/write the text of the
        // Pad
        model.addParticipant("") + User.ID_USER + "@local.net"
    );
}

}
```

A.3.4. Abrir Pad colaborativo en SwellRT