



E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA PLAN 2000

TEMA: Desarrollo Aplicaciones Móviles

TÍTULO: Propuesta de ejercicios prácticos de desarrollo de aplicaciones híbridas

AUTOR: César González Fernández

DEPARTAMENTO: DTE

Miembros del Tribunal Calificador:

PRESIDENTE: FRANCISCO AZNAR BALLESTA

VOCAL: ANTONIO DA SILVA FARIÑA

Vº Bº.

VOCAL SECRETARIO: CARLOS GONZÁLEZ MARTÍNEZ

Fecha de lectura:

Calificación:

El Secretario,

RESUMEN DEL PROYECTO:

Los Smartphones se han convertido en un compañero indispensable para la mayoría de personas. El auge de estos dispositivos ha ido acompañado con la aparición de software que aprovecha las capacidades de estos. Uno de los problemas que surgen a la hora de realizar este software es la variedad de dispositivos en los que va a ser ejecutado. Es por ello que han surgido nuevas tecnologías cuyo objetivo es unificar y facilitar el desarrollo haciendo frente a este problema.

En este PFC se introducirá al lector en el concepto de aplicación híbrida, viendo las herramientas disponibles para su desarrollo. Entre estas herramientas, nos vamos a centrar en Ionic2, un framework basado en Apache Cordova, Angular2 y Sass para la creación de aplicaciones híbridas utilizando tecnologías web proponiendo algunos desarrollos con los que comprobar sus características.



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA POLITÉCNICA TÉCNICA

**INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
TELEMÁTICA**

PROYECTO FINAL DE CARRERA

**PROPUESTA DE EJERCICIOS
PRÁCTICOS DE DESARROLLO DE
APLICACIONES HÍBRIDAS**

**AUTOR: CÈSAR GONZÁLEZ FERNÁNDEZ
TUTOR: ANTONIO DA SILVA FARIÑA**

24 de mayo de 2017

La utopía está en el horizonte. Camino dos pasos, ella se aleja dos pasos y el horizonte se corre diez pasos más allá. ¿Entonces para que sirve la utopía? Para eso, sirve para caminar.

Eduardo Galeano.

El software es como el sexo: mejor si es libre y gratis.

Linus Torvalds.

Dedicado a mi familia y amigos, que han tenido que aguantarme durante todos
estos años.

AGRADECIMIENTOS

Agradecer a mis padres el esfuerzo que hicieron para que pudiera estudiar y el apoyo que me dieron. También a mi hermana y al resto de mi familia, agradecerles que me han acompañado durante toda esta etapa.

A mis amigos, que gracias a ellos podía desconectarme de la universidad cuándo lo necesitaba y que han aguantado los múltiples desplantes.

A la gente que he conocido en estos años de universidad, que hicieron más soportable esos años encerrados en los laboratorios y en la biblioteca.

A mi tutor del proyecto, por volver a darme la oportunidad de hacer con él este PFC después de estar unos años desaparecido.

Y por último, a la educación pública. Que aún no estando en el mejor de sus momentos, y aunque muchos quieran acabar con ella, me ha permitido, al igual que ha muchos otros, llegar hasta aquí.

En los últimos tiempos los smartphones han experimentado un gran crecimiento tanto en el número de terminales en el mercado como en la variedad de estos. Junto con este crecimiento de hardware, se ha experimentado un aumento igual de significativo en el software que aprovecha las capacidades de estos dispositivos. A la hora de crear este software, los desarrolladores se encuentran con la falta de homogeneidad que encontramos en este tipo de dispositivos (diferentes fabricantes, diferentes plataformas, hardware con distintas capacidades, ...).

En este PFC trataremos desde un punto de vista teórico y práctico el uso de Ionic2 en la realización de aplicaciones híbridas para smartphones. Este tipo de aplicaciones intentan que el desarrollo de una aplicación orientada a smartphones pueda servir para los distintos dispositivos en el mercado sin importar la plataforma que ejecuten o el hardware que montan. En el caso de Ionic2, esto se consigue programando la aplicación utilizando tecnología web (HTML, JS y CSS) y ejecutándola sobre un contenedor nativo. Ionic2 se encarga de ofrecer las herramientas necesarias para facilitar este desarrollo, encargándose de compilar la aplicación, generar el contenedor, ofrecer una API unificada sin importar la plataforma, ...

A lo largo de esta memoria se hará una introducción a las tecnologías relacionadas con este tipo de aplicaciones, para más tarde, proponer una serie de prácticas con las que el lector obtendrá los conocimientos necesarios para la realización de una aplicación utilizando Ionic2.

In recent times the smartphones have experimented a great growth in the number of devices and in the diversity of these. In parallel way, the number of software that take advantage of these device's capabilities have experimented the same growth. When this software is created, the developers have a big problem with the lack of homogeneity in this kind of devices (different manufacturers, different platforms, hardwares with different capabilities, ...)

In this PFC we talk about the use of Ionic2 in the development of hybrid applications from a theoretical and practical point of view. These kind of applications try to join development of an application for several devices in an only one development. In order to get this purpose, Ionic2 use web tecnology (HTML, JS and CSS) for develop the application, and a native container run this application. Ionic2 offers necessary tools in order to ease this development, it compile the application's code, it generate the container, it offers an unified API for all platforms, ...

In this whole report we do an introduction to the technologies who are related with hybrid applications. Next, we propose some practices, with these practice the reader will can obtain the necesaries knowledge for create an application using Ionic2

| | |
|---|------------|
| Agradecimientos | I |
| Resumen | III |
| Abstract | V |
| Lista de figuras | IX |
| Glosario | XIV |
| 1. Introducción | 1 |
| 1.1. Plataformas móviles: Android, iOS y el resto. | 2 |
| 1.1.1. ¿Un nuevo contendiente?. Google Fuchsia. | 5 |
| 1.2. En la variedad está problema. Aplicaciones híbridas. | 5 |
| 1.2.0.1. Frameworks al rescate | 8 |
| 2. Marco Tecnológico | 9 |
| 2.1. Frameworks | 9 |
| 2.1.1. jQuery | 9 |
| 2.1.2. Framework CSS | 10 |
| 2.1.3. Framework Javascript | 13 |
| 2.1.4. Frameworks para aplicaciones híbridas | 15 |
| 2.2. Angular | 18 |
| 2.2.1. Angular versión 2 | 20 |
| 2.2.1.1. TypeScript | 21 |
| 2.2.1.2. Aspectos más importantes en Angular | 22 |
| 2.2.2. MEAN | 24 |
| 2.3. Apache Cordova | 26 |
| 2.4. Ionic | 28 |

| | |
|---|------------|
| 2.4.1. Ionic CLI | 29 |
| 2.4.2. Ionic Native | 31 |
| 3. Prácticas | 33 |
| 3.1. Hola Mundo!!! | 34 |
| 3.2. Construir y emular | 39 |
| 3.2.1. Añadir plataforma objetivo | 40 |
| 3.2.2. Emular aplicación | 41 |
| 3.2.3. Depurar una aplicación que está siendo emulada | 42 |
| 3.2.4. Ejecutar una aplicación en un terminal conectado | 46 |
| 3.2.5. Generar el instalador. El APK. | 47 |
| 3.3. Cronómetro | 49 |
| 3.3.1. Cronómetro con contador de vuelta | 57 |
| 3.4. Paisaje | 60 |
| 3.4.1. Primeros pasos. El cielo. | 62 |
| 3.4.2. Animaciones definidas sobre el componente. El sol y la luna. | 63 |
| 3.4.3. Animación mediante código JavaScript/TypeScript. El terreno. | 68 |
| 3.4.4. Animación CSS. El ave. | 73 |
| 3.4.5. Elementos extras. Sonidos y nubes. | 76 |
| 3.4.6. Apunte final y uso de animaciones en otros escenarios. | 77 |
| 3.5. Recordatorios asociados a localizaciones | 79 |
| 3.5.0.1. Análisis funcional | 82 |
| 3.5.0.2. Estructura de las páginas | 83 |
| 3.5.0.3. Instalando los plugins necesarios | 88 |
| 3.5.0.4. Implementación del modelo de datos y la persistencia | 89 |
| 3.5.0.5. ReminderProvider | 94 |
| 3.5.0.6. El mapa | 95 |
| 3.5.0.7. La lista | 103 |
| 3.5.0.8. Las notificaciones | 105 |
| Conclusiones | 107 |
| Trabajos futuros | 109 |
| Presupuesto | 111 |
| Bibliografía | 115 |
| ANEXOS | 119 |

| | |
|---|------------|
| A. Herramientas para trabajar con Android | 119 |
| A.1. Instalación de Android SDK | 120 |
| A.2. Android SDK Manager | 122 |
| Tools | 123 |
| Android SDK Platform | 123 |
| Extras | 123 |
| A.3. Android Virtual Device | 124 |
| A.4. Android Emulator | 127 |
| B. Instalar Node.JS y npm | 131 |
| C. Atom | 137 |
| C.1. Instalación | 138 |
| C.2. Personalización | 138 |
| C.2.1. Paquetes y temas | 138 |
| D. Ionic | 141 |
| E. Google APIs | 145 |
| E.1. Obtención de una clave para utilizar Google Maps | 146 |

LISTA DE FIGURAS

| | |
|---|----|
| 1.1. Crecimiento del número de aplicaciones disponibles en la tienda de Android, el sistema operativo para smartphones con más volumen de terminales en uso. [Dog16]. | 2 |
| 1.2. Cuota de mercado en el cuarto trimestre del año 2016 según la plataforma. [Gar17]. | 4 |
| 1.3. Distribución de los dispositivos conectados a internet teniendo en cuenta el sistema operativo que utilizan. [mar16]. | 4 |
| 2.1. Busquedas mundiales de los dos frameworks CSS más importantes. . | 12 |
| 2.2. Evolución de las búsquedas en Google de los diferentes frameworks. . | 15 |
| 2.3. La diferencia con Angular es tal, que impide ver correctamente los datos del resto de frameworks. | 15 |
| 2.4. Vemos como quitando Framework 7 y NativeScript, el resto de frameworks están a la par. | 18 |
| 2.5. El programador Ben Nadel describió (y de manera muy acertada) su experiencia con Angular en su blog de esta forma. [Nad13] | 19 |
| 2.6. Esquema de algunos de los elementos que encontramos en una aplicación realizada con Angular y como interactúan entre ellos. . . . | 22 |
| 2.7. Los cuatro elementos principales del stack en cada una de las fases. . | 25 |
| 2.8. Este diagrama muestra de forma simplificada como está compuesta una aplicación híbrida que utiliza Apache Cordova. | 27 |
| 2.9. Comparación de un selector de fechas en diferentes plataformas. . . . | 29 |
| 3.1. Estructura del proyecto <i>Hello World</i> al crearlo visto en Atom. | 35 |
| 3.2. Así es como se ve el template <i>blank</i> al ejecutarse. | 35 |
| 3.3. Página principal del template <i>blank</i> | 36 |

| | |
|---|-----|
| 3.4. Podemos ver nuestro Hello World en el navegador por primera vez. | 37 |
| 3.5. Nuestra aplicación ahora nos ofrece un saludo dedicado. | 38 |
| 3.6. Listado de plataformas disponibles para incluir en nuestro proyecto. | 40 |
| 3.7. Comando a ejecutar si queremos añadir la plataforma Android a la lista de plataformas objetivo de nuestra aplicación. | 40 |
| 3.8. Nuestra aplicación vista en el emulador. | 42 |
| 3.9. Opción para abrir la ventana de dispositivos remotos. | 43 |
| 3.10. Lista de dispositivos disponibles para depurar. Entre ellos se encuentra nuestro emulador con nuestra aplicación híbrida abierta. | 44 |
| 3.11. Diferencia entre como se ve el código sin compilar, emulado con la opción <i>-livereload</i> (arriba), y el código compilado (abajo). | 45 |
| 3.12. La aplicación muestra por consola el mensaje de inicio. | 46 |
| 3.13. El terminal Android conectado a nuestro equipo aparece listado entre los dispositivos disponibles en Chrome. | 47 |
| 3.14. Una pequeña parte de los iconos que ofrece Ionicons. Podemos ver la diferencia que existen entre ambas plataformas. | 50 |
| 3.15. Esto sí que es un cronómetro sencillo. | 51 |
| 3.16. En esta nueva versión hemos añadido dos nuevos botones además de una lista con los tiempos guardados. | 57 |
| 3.17. Nuestra aplicación nos mostrará un paisaje, en el que podremos cambiar entre noche y día. | 62 |
| 3.18. Así se vería el mapa, con los recordatorios marcados. | 79 |
| 3.19. Aquí vemos la lista de recordatorios, sobre los que podremos interactuar. | 80 |
| 3.20. La notificación aparece aunque no este abierta la aplicación. | 81 |
| 3.21. Diagrama de clases. | 82 |
| 3.22. Mockup de la aplicación. Podemos ver los enlaces entre páginas y los menús desde las que se acceden. | 84 |
| 3.23. Menú lateral que nos proporciona la plantilla sidemenu. | 85 |
| 3.24. Diagrama de actividad que muestra la creación de un nuevo recordatorio. | 98 |
| A.1. Java Development Kit está disponible para diferentes sistemas operativos y arquitecturas. | 120 |
| A.2. Durante la instalación podremos cambiar la ruta de instalación del Java Development Kit, aunque se recomienda no modificarla si no es estrictamente necesario. | 121 |

| | |
|---|-----|
| A.3. Podemos comprobar desde la consola la versión del JDK que tenemos instalada. | 121 |
| A.4. Aunque la opción está un poco “escondida”, podemos descargarnos el Software Developer Kit (SDK) de Android sin necesidad de instalar Android Studio. | 122 |
| A.5. Ficheros que se encuentran dentro del paquete comprimido. | 122 |
| A.6. Android SDK Manager nos permite instalar los diferentes componentes de Android SDK por separado. | 123 |
| A.7. Desde el propio manager de Android SDK podremos abrir el manager de avds. | 124 |
| A.8. AVDs Manager. | 125 |
| A.9. A la hora de crear un nuevo dispositivo, podemos configurar varias características de su hardware. | 125 |
| A.10. Vista con la que podemos crear nuestro Android Virtual Device (AVD). | 126 |
| A.11. Emulador de Android, con la pantalla a la izquierda y la botonera a la derecha. | 128 |
| A.12. Ventana de controles extendidos del emulador. | 129 |
| B.1. Paso de bienvenida del instalador. | 132 |
| B.2. Terminos de uso. | 132 |
| B.3. Configuración del directorio de instalación de node.JS. | 133 |
| B.4. Paso final. | 134 |
| B.5. La instalación se ha realizado correctamente. | 134 |
| C.1. Pantalla desde donde instalar paquetes y temas desde la propia aplicación de atom. | 139 |
| D.1. Estructura de archivos que sirve como base para empezar un nuevo proyecto con ionic. Han sido generados utilizando el comando <i>ionic start</i> | 142 |
| D.2. Vemos como el servidor de la aplicación ionic está en marcha. | 142 |
| D.3. La aplicación vista desde un navegador. | 143 |
| E.1. Consola para desarrolladores de Google. | 145 |
| E.2. Ventana para la creación de un proyecto dentro de nuestra cuenta. | 146 |
| E.3. Lista de proyectos asociados a nuestra cuenta. | 147 |
| E.4. Menú desde el que acceder a diferentes secciones relativas al proyecto. | 147 |
| E.5. APIs disponibles para poder asignar al proyecto. | 148 |
| E.6. Información relativa a la API de Google Maps Android. | 148 |

- E.7. Configuración de las credenciales para nuestra API. El servicio nos proporciona diferentes maneras para realizar la autorización. 149
- E.8. Opciones para restringir el acceso según el origen de la petición. Para una aplicación Android, necesitaríamos indicar el nombre del paquete de nuestra aplicación. 150
- E.9. Permisos sobre el proyecto para distintos usuarios de la consola de desarrolladores. Aquí podemos compartir el proyecto con el resto del equipo de desarrollo y asignarles diferentes permisos a cada uno. . . . 150
- E.10. Cuotas que se aplican a cada uno de los servicios asociados al proyecto. 151

ADB Android Debugger Bridge.

AOSP Versión básica del sistema operativo para smartphone de Google, Android..

API La interfaz de Programación de Aplicaciones es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción..

APKs Android Application Package.

AVD Android Virtual Device.

BBDD Base de datos.

CLI Command Line Interface.

CSS Cascading Style Sheets.

CSS3 Cascading Style Sheets v3.

DI Dependency Injector.

DOM El Modelo de Objetos para Representación de Documentos se trata de una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos..

GPS Global Positioning System.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

IAM Identify and Access Management.

IDE Integrated Development Environment.

IP Internet Protocol.

JDK Conjunto oficial de herramientas necesarias para el desarrollo de aplicaciones en Java..

JRE Entorno de ejecución para software desarrollado en Java..

JS JavaScript.

JSON JavaScript Object Notation.

LTS Long Term Support.

MEAN Conjunto de herramientas para la programación de aplicaciones distribuidas usando JavaScript como lenguaje en todas las fases. El nombre proviene de las cuatro herramientas principales usadas en este conjunto (MongoDB-ExpressJS-Angular-NodeJS).

mixin En la programación orientada a objetos, una clase mixin es una clase que contiene propiedades y métodos que pueden ser reutilizados por otras clases sin necesidad de tener una relación de parentesco con ellas. En el caso de SASS, permiten definir estilos que se pueden reutilizar y parametrizar..

MV* Model-View-Whatever.

MVC Model-View-Controller.

MVVP Model-View-ViewModel.

NFC Near Field Communication.

NoSQL NoSQL.

npm The Node Package Manager.

ORM Object Relational Mapping.

PC Personal Computer.

PFC Proyecto Fin de Carrera.

Sass Syntactically Awesome Stylesheets.

SDK Software Developer Kit.

SEMVER El sistema SEMVER es un conjunto de reglas para proporcionar un significado claro y definido a las versiones de proyecto de software. El sistema SEMVER se compone de 3 números, siguiendo la estructura X.Y.Z, donde X (major) indica un cambio rupturista, Y (minor) indica cambios compatibles con versiones anteriores y Z (path) indica resoluciones de bugs..

SPA Single Page Application.

SQL Structure Query Language.

URL Uniform Resource Locator.

USB Universal Serial Bus.

CAPÍTULO 1

INTRODUCCIÓN

En los últimos años hemos visto como los smartphones se han convertido en un dispositivo esencial para cualquier persona. La mayoría de gente, entre los que me incluyo, no pueden salir de casa sin él. ¿Quién me iba a decir que hasta mis padres los utilizarían?. Este crecimiento en el uso de los smartphones ha propiciado la aparición dispositivos más o menos relacionados como tablets, smartwatches y Smart TV.

El desarrollo de todo este nuevo hardware ha traído consigo el desarrollo nuevo software. Desde sistemas operativos que controlan y dan vida a los dispositivos, a las aplicaciones que sacan partido a todo su potencial y son las que ofrecen al usuario nuevas funcionalidades que al final es lo que busca al comprar este tipo de terminales.

Un software que por otro lado no para de crecer, y que se tiene que adaptar a unos dispositivos que evolucionan a un ritmo desorbitado. Unos dispositivos que cada vez son más potentes y que cada vez incorporan nuevas características a las que poder sacar provecho, como por ejemplo, los sensores de huella dactilar que han surgido como medio de autenticación del usuario sustituyendo a las habituales contraseñas. El software también debe hacer frente a la variedad de usuarios que hacen uso de estos dispositivos y lo que esperan encontrar en estos. Aplicaciones para la oficina, videojuegos, realidad virtual, aplicaciones para el aprendizaje de idiomas, mensajería, ...un sinfín de tipos de aplicaciones que podemos encontrarnos hoy en día.

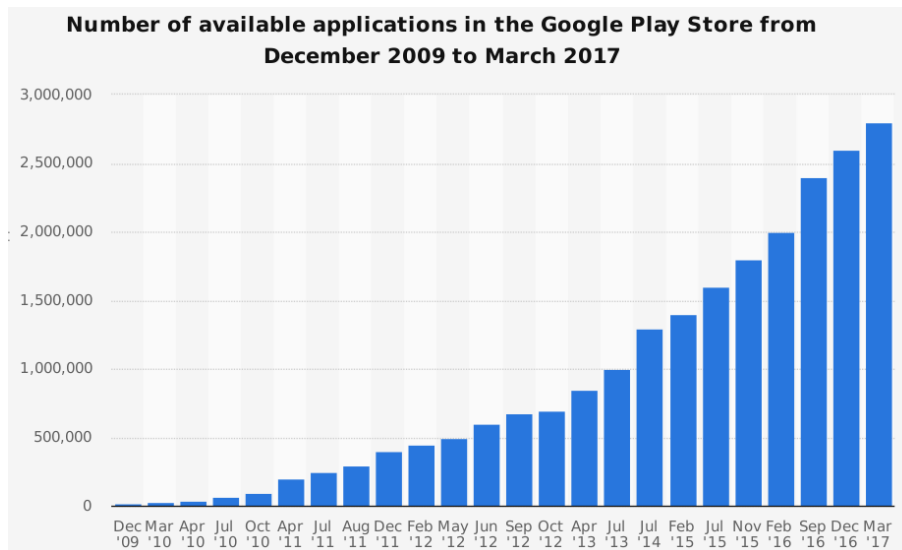


Figura 1.1: Crecimiento del número de aplicaciones disponibles en la tienda de Android, el sistema operativo para smartphones con más volumen de terminales en uso. [Dog16].

1.1. Plataformas móviles: Android, iOS y el resto.

Si dejamos a un lado la batalla entre los fabricantes de hardware (Samsung, LG, Sony, ...), nos encontramos con otra batalla en lo referente a la plataforma o sistema operativo utilizado.

En un principio, cada fabricante de teléfonos tenía su propio sistema operativo, el cual incluía en sus dispositivos. Además, eran los propios fabricantes los encargados de desarrollar las aplicaciones que se ejecutarían en sus terminales, lo que limitaba en gran medida la variedad de estas aplicaciones.

Más tarde, diferentes fabricantes empezaron a unirse para crear conjuntamente sistemas operativos que compartían. El más famoso sin duda alguna fue Symbian. Este sistema operativo, propiedad de Nokia, fue producto de la alianza de esta con otras compañías como Siemens, Samsung, LG, ...pero fue Nokia la que más partido le sacó y con la que se hizo famoso el sistema. Uno de los puntos fuertes de Symbian fue la gran comunidad de desarrolladores que realizaban aplicaciones para esta plataforma, lo que la hacía muy atractiva para un usuario final que, hasta entonces, apenas disponía de tonos y fondos de pantalla para personalizar su terminal.

Fue en 2007 cuando ocurrió un evento muy importante. La compañía Apple, ya famosa por sus ordenadores y su iPod, presentó iPhone OS, que más tarde pasaría a llamarse *iOS*, su nombre actual. Junto con el nuevo sistema aparecía el primer

iPhone, el smartphone que sin duda supuso el inicio del *boom* de este tipo de dispositivo. Este sistema es propietario de Apple, y solo lo podemos encontrar en sus terminales iPhone e iPad. Sus aplicaciones empezaron siendo escritas en Objective-C, lenguaje que años más tarde cambiaron para utilizar Swift. Este es un dato importante para lo que explicaremos a continuación.

Un año más tarde, allá por septiembre de 2008, Google presenta su sistema operativo, Android. Este sistema operativo estaba realizado por una empresa del mismo nombre, la cual Google había comprado unos años antes. La versión más básica es conocida como **Android Open Source Project**, que se trata de un proyecto de código abierto. A diferencia de iOS, podremos encontrar Android en multitud de dispositivos de multitud de fabricantes, desde smartphones a encontrarse instalado en sistemas multimedia para automóviles, desde Samsung a BQ. En el caso de Android las aplicaciones están escritas usando Java.

El carácter abierto de Android ha propiciado que aparezcan diversos firmwares basados en él. Quizás los más conocidos sean MIUI¹, propio de los terminales de Xiaomi, y el ya discontinuado CyanogenMOD. Pero por la web, podemos encontrar multitud de versiones y personalizaciones del sistema creada por la comunidad de desarrolladores. También es común que los fabricantes de smartphones que usen Android, creen su propia versión personalizada de este.

Junto a estos dos sistemas operativo han convivido otros como el ya mencionado Symbia, Windows y BlackBerry (que tuvo una época dorada allá por 2008). También intentos menos exitosos, como Fire OS de Amazon, Firefox OS (basado en HTML5) o Ubuntu Touch de Canonical. Pero si algo tienen en común todos ellos, es que se han convertido, o no han pasado de serlo, en sistemas operativos residuales en cuanto a cuota de mercado. En la siguiente tabla podemos ver esta cuota en el tercer cuatrimestre de 2016.

¹<http://en.miui.com/>

| Operating System | 4Q16 Units | 4Q16 Market Share (%) | 4Q15 Units | 4Q15 Market Share (%) |
|------------------|------------------|--------------------------|------------------|--------------------------|
| Android | 352,669.9 | 81.7 | 325,394.4 | 80.7 |
| iOS | 77,038.9 | 17.9 | 71,525.9 | 17.7 |
| Windows | 1,092.2 | 0.3 | 4,395.0 | 1.1 |
| BlackBerry | 207.9 | 0.0 | 906.9 | 0.2 |
| Other OS | 530.4 | 0.1 | 887.3 | 0.2 |
| Total | 431,539.3 | 100.0 | 403,109.4 | 100.0 |

Figura 1.2: Cuota de mercado en el cuarto trimestre del año 2016 según la plataforma. [Gar17].

Otro dato curioso para ver la importancia que han tomado estos sistemas operativos y este tipo de dispositivos, es comparar el número de dispositivos conectados a internet según su sistema operativo a lo largo del tiempo. En la siguiente gráfica se puede ver como peso de Windows para PC disminuye en tanto que el de los sistemas operativos para terminales móviles crecen.

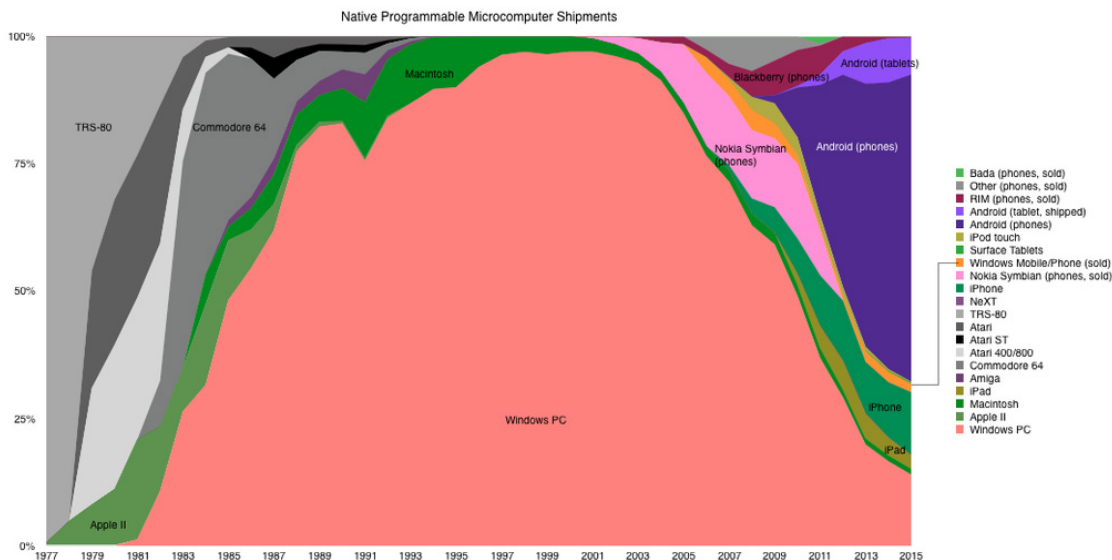


Figura 1.3: Distribución de los dispositivos conectados a internet teniendo en cuenta el sistema operativo que utilizan. [mar16].

1.1.1. ¿Un nuevo contendiente?. Google Fuchsia.

[Ama17] En el momento de escribir esta memoria (10 de mayo de 2017), se filtrado el que posiblemente sea un nuevo sistema operativo creado por Google. Este sistema vendría a unificar los dos sistemas operativos de la compañía, Android y Chrome OS².

Las mayores características de este nuevo sistema serían por un lado, el abandono del kernel de linux del que hacen uso tanto Android como Chrome OS. Este sería remplazado por un kernel desarrollado por Google y que responde al nombre de Magenta. La otra diferencia sería el uso de Flutter SDK³, un SDK también desarrollado por Google para la creación de aplicaciones multiplataforma que se centra en el rendimiento y que utiliza el lenguaje Dart⁴ (una alternativa a JavaScript lanzada por Google). Este SDK aun se encuentra en sus primeras fases de desarrollo, pero de ser usado en el nuevo sistema, significaría que las aplicaciones creadas para Fuchsia, podrían ser también usadas en Android. En la web de Flutter SDK podemos encontrar una aplicación no funcional válida para Android en la que podemos ver una pequeña muestra de la UI que ofrece, la cual es conocida con el nombre de Armadillo.

Aun es pronto para saber los planes que tiene Google con este sistema operativo, y que hará con Android. Pero podemos encontrarnos con que en un futuro próximo se una esta nueva plataforma a Android y a iOS, diversificando más el mercado. Y es que aunque Google apueste por el nuevo sistema, serán los fabricantes de smartphone los que tendrán que introducirlos en sus dispositivos, y como se comentó anteriormente, muchos de ellos cuentan con versiones personalizadas. Esto hará que el paso de Android a Fuchsia (en el caso de que se produzca) se alargue en el tiempo y ambas plataformas convivan.

1.2. En la variedad está problema. Aplicaciones híbridas.

A la hora de implementar una aplicación nos encontramos con el dilema de, ¿a qué dispositivo va dirigido?. Viendo las cuotas de mercado, se podría pensar que la opción mejor es lanzarse a crear nuestra aplicación para Android. Pero si investigamos un poco, veremos que las aplicaciones lanzadas en iOS producen hasta cuatro veces más beneficios que las de Android [Dil16]. Por otro lado, lo más probable

²<https://www.chromium.org/chromium-os>

³<https://flutter.io/>

⁴<https://www.dartlang.org/>

es que no nos interese desarrollarla solo para iOS ya que Android cuenta con 7 veces más usuarios. También tenemos que pensar en la función que va a tener nuestra aplicación. Si por ejemplo, nuestra aplicación se basa en recoger datos de cualquier tipo (por ejemplo, reportar accidentes de tráfico) y compartirlos entre usuarios de la aplicación, nos interesa que nuestra aplicación llegue al mayor número de usuarios posible para poder captar un volumen mayor de datos. O si nuestra aplicación forma parte de un producto, como puede ser manejar un drone, no podemos excluir a una parte de potenciales compradores del producto únicamente por no disponer de un terminal que le permita ejecutar nuestra aplicación. Y no nos olvidemos de Windows ...

Como vemos, parece más que recomendable abarcar al menos las dos plataformas mas importantes a la hora de crear una aplicación, pero no siempre se cuenta con los recursos o conocimientos necesarios para abordar dos proyectos diferentes y no solo eso, si no también mantenerlos.

Para solventar este problema surgieron las *aplicaciones híbridas*, ¿para qué programar dos aplicaciones desde cero si van a tener un mismo funcionamiento?. Mientras que las *aplicaciones nativas* se desarrollan en el lenguaje específico de cada plataforma y utilizando la **Application Programming Interface** propia de cada sistema, las aplicaciones híbridas utilizan tecnología web (HTML5, CSS y JavaScript) para crear la aplicación y a continuación, mediante el uso de diferentes herramientas, son compiladas lo que genera un ejecutable para cada una de las plataformas.

Las aplicaciones híbridas nos ofrecen ventajas sobre las nativas, pero también presentan sus desventajas. En internet se pueden encontrar multitud de comparativas entre ambos tipos de aplicaciones y según el autor, las diferencias entre ellas serán mas o menos importantes. aún así, a continuación expondré aquellos puntos que considero más importantes a la hora de decidir por que tipo de aplicación apostar.

Curva de aprendizaje En el caso de las aplicaciones nativas, debemos aprender como funcionan estas en cada una de las plataformas, el lenguaje que se utiliza para programar, las APIs que ofrece el sistema, ...en cambio, para las aplicaciones híbridas, solo será necesario saber HTML, CSS y JS junto con el framework o herramientas que utilicemos para ayudarnos a crear nuestras aplicación. Además, los frameworks que han aparecido para ayudar en la creación de aplicaciones híbridas facilitan mucho el trabajo con algunas de las **APIs** del sistema operativo, llegando a ser más fácil utilizarlas a través del framework, que de manera nativa. Este es el punto fuerte de

las aplicaciones híbridas.

Diseño de la aplicación Aunque cada vez son más y mejores los frameworks que nos ayudan a crear las aplicaciones híbridas, es cierto que en el desarrollo de este tipo de aplicaciones es más complicado el tener un diseño acorde a la plataforma. Puede parecer un problema menor, pero el aspecto de una aplicación es muy importante, y la experiencia de uso puede verse perjudicada al utilizar ciertos estilos a los que el usuario no está acostumbrado (widgets, botones, distribución de elementos, ...). El problema puede ser aún mayor si la aplicación está destinada para iOS, ya que Apple puede vetarla de su market si no cumple con ciertos estándares de diseño.

Puede parecer desproporcionado este énfasis en cuanto a la homogeneidad del diseño, pero si pensamos en los ordenadores de sobremesa, ocurre lo mismo. Por ejemplo, los usuarios de Windows estamos acostumbrados a que los botones para minimizar/cerrar una ventana se encuentren en la esquina superior derecha y sería complicado adaptarse a que eso no fuera así. Solo hay que ver las críticas que recibió Microsoft cuando lanzó Windows 8 con su menú, teniendo que volver a uno más tradicional en la versión 8.1.

Rendimiento de la aplicación Otro punto en contra de las aplicaciones híbridas es el rendimiento. Suelen mostrarse más lentas que las nativas. Este problema puede no ser crítico en función del tipo de aplicación y si tenemos en cuenta que cada vez los dispositivos móviles son más potentes. Aun así, es un punto a tener en cuenta.

Seguridad Por norma general, las aplicaciones nativas son más seguras que las híbridas. No nos olvidemos que estas últimas suelen hacer uso de frameworks que en su mayoría son libres y su código es abierto, lo que facilita la búsqueda de debilidades. Esto hace que su uso sea poco recomendable en aplicaciones sensibles, como por ejemplo aquellas que gestionen cuentas bancarias.

Disponibilidad de las capacidades del terminal Es posible que no tengamos acceso (o esté limitado) a ciertas capacidades del dispositivo, por ejemplo, algún sensor en concreto, desde una aplicación híbrida. Evidentemente, este problema no lo tendremos con una aplicación nativa. Es por tanto importante comprobar que la tecnología que elijamos cubra nuestras necesidades, y no encontrarnos con esta limitación a mitad del desarrollo lo que conllevaría un esfuerzo extra **MUY** importante.

Costes El factor que suele tener (lamentablemente) más peso a la hora de elegir. Es evidente que, una aplicación híbrida tiene un menor coste, ya que no requiere el dedicar un equipo de trabajo para cada plataforma. Incluso teniendo que implementar la aplicación para solo una, es muy posible que la aplicación híbrida sea una opción más económica (y deja la puerta abierta a ser lanzada en otras plataformas en el futuro).

En resumen, las aplicaciones híbridas son una buena alternativa para la mayoría de aplicaciones de hoy en día, y con la evolución de los framework, cada vez más. Pero si queremos una aplicación robusta, con una interfaz cuidada y un rendimiento óptimo, y siempre que el presupuesto lo permita, la opción nativa sigue siendo la más indicada.

Una buena estrategia es utilizar aplicaciones híbridas en la etapa inicial de nuestro proyecto y según cómo responda el mercado a nuestra idea, decidirse a dar el paso y hacer el cambio a aplicaciones nativas, eso sí, siempre que sea necesario.

1.2.0.1. Frameworks al rescate

Para ayudar al desarrollo de aplicaciones híbridas han surgido frameworks que facilitan la tarea de integración de las diferentes plataformas haciendo que converjan en una única interfaz.

Sin duda alguna, el proyecto que más ha impulsado la creación de aplicaciones híbridas ese a sido Adobe Cordova (como veremos más adelante, también conocido como PhoneGap), que proporcionan las herramientas necesarias para que nuestra aplicación basada en tecnología web pueda ser ejecutada en el dispositivo. Junto con Cordova, han aparecido multitud de frameworks, algunos de los cuales utilizan Cordova como base, los cuales facilitan el desarrollo ofreciendo conjuntos de componentes estéticos, añadiendo patrones de programación, potenciando las tecnologías web usadas ([HyperText Markup Language \(HTML\)](#), [Cascading Style Sheets \(CSS\)](#) y [JavaScript \(JS\)](#)), ...

CAPÍTULO 2

MARCO TECNOLÓGICO

En este capítulo vamos a ver el estado de las tecnologías relacionadas con las aplicaciones híbridas, haciendo mención de aquellas más importantes. A continuación, se verá con más profundidad aquellas que se usarán a lo largo de este **Proyecto Fin de Carrera (PFC)**, y que son Ionic, Angular y Cordova.

2.1. Frameworks

Al comienzo de la aparición de las aplicaciones web, los desarrolladores disponían únicamente de **HTML**, **JS** y **CSS** para su implementación, sin ningún tipo de librería ni framework que les facilitará esta tarea. Se conoce como *Vanilla JavaScript* al uso de JavaScript sin ningún tipo de librería ni framework. Con el aumento tanto en número como en complejidad de estas aplicaciones, se hizo necesario disponer de herramientas que facilitarán el desarrollo de estas.

Así comenzaron a surgir diferentes herramientas y frameworks que facilitaban las tareas de programación tanto en el lado de la lógica de negocio como en el apartado visual.

2.1.1. jQuery

jQuery¹ fue y sigue siendo una de las librerías más usada en el mundo del desarrollo web. Esta librería tiene como objetivo facilitar algunas de las deficiencias que nos encontramos a la de programar con JavaScript.

Con jQuery conseguimos simplificar el acceso a los nodos del árbol *DOM* que el navegador crea a partir de nuestro **HTML**, consiguiendo una sintaxis mas clara,

¹<https://jquery.com/>

fácil de recordar y además compatible con todos los navegadores. Por ejemplo:

```
// Vanilla JavaScript
var elem = document.getElementById("miElemento");

//jQuery
var elem = \$("#miElemento");
```

También proporciona herramientas útiles como métodos específicos para realizar llamadas *AJAX*, es decir, llamadas desde la aplicación web que se ejecuta en el navegador al servidor de manera asíncrona utilizando el protocolo **Hypertext Transfer Protocol (HTTP)**.

jQuery ha supuesto el punto de partida de muchos otros frameworks y complementos, lo que hace que tenga aún más valor. Ya que hablamos de la importancia de jQuery como base para otros proyectos, sería injusto no mencionar a *underscoreJS*², otra librería que ha venido a suplir algunas cadencias de JavaScript.

Mas tarde, se lanzo jQuery UI³, que consistía en un conjunto de herramientas y facilidades tanto CSS como JavaScript para la creación de interfaces de usuario. A su vez, de jQuery UI⁴ surgió jQuery Mobile, una versión adaptada a dispositivos móviles que por aquella época empezaban cobrar mayor importancia.

2.1.2. Framework CSS

[Jac16; Pri17] Con el paso de los años, el diseño de las páginas a ido haciéndose cada más y más complejo. Las diferentes versiones de **HTML** y de **CSS** han ido introduciendo nuevas opciones en este sentido, lo que aún siendo necesarias, a veces resultan insuficientes en el desarrollo de la página. Además de esto, la variedad de terminales desde los que se puede visualizar una página es también cada vez mayor, teniendo que pensar en el diseño según sea vista en un dispositivo móvil, una tablet, un PC, el navegador que se utiliza (ya que todos no reconocen los mismos parámetros), ...y no solo eso, también el tamaño de pantalla, la resolución, la orientación, Si conseguimos que nuestro diseño se adapte a todos estos casos, podremos decir que el nuestro se trata de un diseño *responsive*. Como nos podemos imaginar, el realizar un diseño *responsive* requiere una cantidad de tiempo y recursos importante, ya que no solo debemos de implementar el estilo para diferentes usos, también es necesario probarlo.

²<http://underscorejs.org/>

³<http://jqueryui.com/>

⁴<http://jquerymobile.com/>

Esta tarea de diseño ha sido facilitada con la aparición de lo que se conocen como *frameworks* **CSS**. Estos frameworks utilizan hojas de estilos creadas y testadas por equipos de desarrolladores y diseñadores con gran experiencias, listas para ser usadas como base en una maquetación web y no tener que partir desde cero. Una de la funcionalidades principales que suelen ofrecer, y que siempre a sido un dolor de cabeza para los diseñadores, razón por la que la destaco, son los *grid*. Gracias ellos podemos dividir nuestra página en una especie de cuadrícula y sobre la que poder distribuir los elementos de nuestra página.

Aunque las ventajas de usar uno de estos frameworks son evidentes, también tienen ciertos inconvenientes a tener en cuenta antes de decidirmos a usarlos. En primer lugar es la limitación a la hora de personalizar el diseño, ya que partimos de uno ya creado y pensado para dar soporte a una gran cantidad de dispositivos, por lo que su implementación es compleja y modificarlo suele ser complicado. Otro inconveniente es el código del mismo framework. Lo más normal es que no se utilice todo lo que te ofrece el framework, pero aun así, será descargado por completo por el cliente. Según el tamaño y la conexión del cliente, esto puede ser un problema el cual, con la mejora de las conexiones, cada vez es menos importante.



Este tipo de frameworks suelen ofrecer un único núcleo y módulos separados los cuales implementan diferentes características. Así, podemos importar solo aquellos módulos que nos interesan, por ejemplo, el widget para el calendario o el módulo que maneja las tablas. También cuentan con versiones reducidas en los que el código esta ofuscado, consiguiendo reducir aún más el tamaño del código.

También tenemos que tener en cuenta la dificultad de aprendizaje de estos frameworks, lo que puede hacerlos poco atractivos para pequeños proyectos si no tenemos conocimientos y experiencia previa con ellos.

El anteriormente mencionado jQuery UI se considera un framework **CSS**, pero han surgido durante los últimos años otros más potentes y más interesantes. Actualmente los dos más importantes se podría decir que son:

1. Bootstrap⁵: Desarrollado por Twitter, quien lo liberó como código abierto más tarde, se trata del framework **CSS** más conocido y utilizado, siendo uno de los proyectos con más apoyo en Github⁶. Se puede observar en la gráfica

⁵getbootstrap.com

⁶<https://github.com/>

de debajo de estas líneas la diferencia de interés que despierta en comparación con Foundation, el otro gran framework **CSS**, si tenemos en cuenta las búsquedas realizadas en Google. Sus puntos fuertes son su preocupación por la compatibilidad y la estabilidad (realizan test automáticos antes de publicar cualquier cambio en la rama principal del repositorio⁷), su sistema de *Grid* y la gran comunidad que hay detrás. En su última versión utiliza tecnologías recientes como **Syntactically Awesome Stylesheets (Sass)**, implementan *Flexbox* de **CSS**, usan **mixins**⁸ y hojas de estilos separadas para mejorar la escalabilidad ...



Figura 2.1: Búsquedas mundiales de los dos frameworks CSS más importantes.

- Foundation⁹: Nos encontramos ante un framework que, aunque menos usado y apoyado, cuenta con una potencia similar a lo visto en Bootstrap. Al igual que este, hace uso de **Sass**, *Flexbox*, **mixins**, ...además de un sistema de *Grid* considerado mejor que el de Bootstrap. Palidece, eso sí, si hablamos de compatibilidad y de estabilidad. Cuenta con una integración especial para **emails**, por lo que lo hace muy recomendable en caso de que nuestro diseño se tenga que adaptar al envío de correos. Todo esto hace que sea una alternativa igual de válida que Bootstrap.

Aunque estos son los más importantes, existen otros muchos framework los cuales merece la pena mencionar como es **Pure.CSS**¹⁰, cuya fuerza radica en su ligereza y sencillez, o **Materialize**¹¹, enfocado en el estilo *material design*.

⁷<https://github.com/twbs/bootstrap>

⁸<http://getbootstrap.com/css/#mixins>

⁹<http://foundation.zurb.com/>

¹⁰<https://purecss.io/>

¹¹<http://materializecss.com/>

2.1.3. Framework Javascript

[Sha15] Al igual que ha ocurrido con el diseño de las páginas, la lógica de negocio que se implementa en estas también ha aumentado en complejidad. Se ha producido además un incremento de popularidad de las aplicaciones **Single Page Application (SPA)**. En el desarrollo de este tipo de aplicaciones, trabajar solo con JavaScript se hace sumamente complicado, y jQuery por si mismo no llega a cubrir todas las necesidades que iban surgiendo. Se hacía patente por tanto la necesidad de disponer de herramientas que facilitaran el desarrollo de estas páginas.

Estos framework proporcionan multitud de facilidades, pero hay que ser cauto a la hora de elegir uno y usarlo para nuestro proyecto. Se debe tener en cuenta el tiempo de aprendizaje que requiere y que se debe sumar, además del propio tiempo de implementación. Si además, la aplicación va a tener que ser mantenida y va a sufrir actualizaciones, debemos de asegurarnos de que el framework elegido vaya a gozar de un buen futuro y que tendremos los conocimientos suficientes para abordar el proyecto con el framework escogido, de no ser así, correremos el riesgo de tener que enfrentarnos en el futuro con la deuda técnica que una mala decisión puede provocar.

Empezaron entonces a surgir multitud de frameworks muchas características en común: son de código abierto, implementan patrones de diseño tipo **Model-View-Whatever (MV*)** con los que intentan solventar los problemas de las SPA; todos ellos manejan los conceptos de vista, evento, modelo y ruta.

Sin mencionar Angular, del cual hablaremos más adelante con más profundidad, destacan:

1. Backbone.js¹²: Fue uno de los más populares y usados antes de la explosión de Angular. Este framework que implementa el patrón **Model-View-Controller (MVC)**, permite utilizar el sistema de plantillas que se quisiera, usar otras librerías, Además, y a pesar de tratarse de un framework bastante potente, destaca por su reducido tamaño, de apenas unos 7 Kb. Este framework es muy recomendable para proyectos de pequeño tamaño y poco complejos, ya que en aplicaciones grandes con cierta complejidad, la carecía de ciertas facilidades con las que otros frameworks cuentan se hace más evidente.
2. Knockout.js¹³: Como el anterior, se trata de un framework muy ligero, apenas 40 Kb, pero que ofrece soluciones muy interesantes y diferenciadoras como el uso del patrón **Model-View-ViewModel (MVVP)**, el poder hacer un binding

¹²<http://backbonejs.org/>

¹³<http://knockoutjs.com/>

entre los datos y los elementos del árbol **Document Object Model** o su sistema de plantillas. Al igual que ocurría con Backbone.JS, este framework no es muy recomendable para su uso en aplicaciones de cierto tamaño y complejidad.

3. Ember.js¹⁴: Otro framework relativamente joven y que implementa el patrón **MVC**. Está basado en jQuery y HandlebarsJS ¹⁵, por lo que ambas librerías deben ser importadas en nuestra página. Su filosofía es hacer que el código sea lo mas limpio y sencillo posible, lo que es conocido como *Ember way*. Se basa en el paradigma de programación de **Convención sobre configuración** en su búsqueda de la simplicidad. Este empeño por la simplicidad hace que sea menos flexible que otros framework. A parte de eso, implementa otras facilidades como *routing*, *binding*, funcionalidades para trabajar con APIs RESTful, ...
4. React [Aza15]: A decir verdad, no es del todo correcto incluir React en esta lista ya que no se trata de un framework como los anteriores, si no que se trata de una librería de JavaScript, pero he decidido incluirla aquí debido a la popularidad que está consiguiendo últimamente. Facebook se encuentra detrás del desarrollo y mantenimiento de esta librería que esta enfocada en el renderizado de la página, se puede decir que es la **V** del patrón **MVC**. Es por esto que puede ser usado junto otros frameworks que se encarguen de otras funciones del patrón, como por ejemplo, con Angular. Está basado en componentes, los cuales controlan su propio estado y cuya lógica se encuentra implementada en el JavaScript. Su característica más importante es el DOM virtual que genera con cada componente y el algoritmo *Diff* que le permite a la hora de producirse un cambio, calcular solo la parte del DOM a la que afecta este cambio, y solo repintar esta parte. Con esto se consigue que el tiempo de renderizado disminuya considerablemente.



La palabra **renderización** es una adaptación al castellano del vocablo inglés *rendering* que define un proceso de cálculo complejo desarrollado por un ordenador destinado a generar una imagen o secuencia de imágenes.

Podemos comparar el interés que despiertan estos frameworks usando la herramienta Google Trends:

¹⁴<https://emberjs.com/>

¹⁵<http://handlebarsjs.com/>



Figura 2.2: Evolución de las búsquedas en Google de los diferentes frameworks.

Vemos como hace unos años Backbone.js gozaba de gran popularidad, estando claramente por encima del resto, pero poco a poco esta popularidad ha ido bajando hasta ponerse al nivel de Knockout.js y EmberJS, los cuales mantienen más o menos el nivel pero también sufren. Por el contrario, React.js, aun siendo el último en llegar, consigue superar a los anteriores. Uno de los motivos de estas variaciones es la evolución que están teniendo las SPA y las aplicaciones híbridas. Vemos como Backbone.js y Knockout.js son los que mas han bajado y son a su vez, los menos eficaces para desarrollar páginas complejas.

Si os preguntéis por qué no he añadido Angular al gráfico anterior, debajo de esta línea podéis ver la razón.



Figura 2.3: La diferencia con Angular es tal, que impide ver correctamente los datos del resto de frameworks.

2.1.4. Frameworks para aplicaciones híbridas

[Aro16; Mar15] Junto a la aparición de las aplicaciones híbridas han ido aparecido nuevas necesidades. Como se ha visto en la [Introducción](#), el fuerte de

estas aplicaciones es la posibilidad de crear una aplicación que funcione en diferentes tipos de dispositivos a partir de un único desarrollo. La variedad de dispositivos, no solo si tenemos en cuenta la plataforma en la que se basa, si no también las características del hardware (en especial el tamaño y la resolución de la pantalla) puede suponer un problema a la hora de realizar el diseño de nuestra aplicación. Por otro lado, este tipo de aplicaciones suelen tener cierta complejidad, ya que suelen estar compuestas por diferentes vistas, utilizan servicios en la nube, necesitan comunicarse con la **API** del dispositivo,

Surgieron entonces diferentes frameworks que tratan de dar solución a estos problemas. Algunos de ellos pensados para crear aplicaciones web que se vieran bien en los navegadores móviles, pero que se han usado a la hora de crear aplicaciones híbridas, y otros cuyo único objetivo es servir para el desarrollo de estas últimas. Obviando tanto *Apache Cordova* como *Ionic* ya que hablaremos de ellos más adelante, los más destacables son:

1. **jQuery Mobile**: El abuelo de los framework para dispositivos móviles. Se trata de una versión mínima de jQuery UI y que como no, se apoya sobre jQuery. Se trata de un framework ligero cuyo objetivo principal es funcionar en el mayor número posible de terminales sin importar la plataforma, el tamaño o el navegador usado. Tiene su propio estilo, sin adaptarse al *look and feel* de la plataforma que lo ejecuta, lo que es una limitación importante si queremos decidirnos por él. Aunque no lo incorpora por defecto, es compatible con *Apache Cordova*.
2. **Appcelerator Titanium**: Este framework no sirve para crear aplicaciones híbridas tal como las hemos definido, pero es importante también nombrarlo debido a la popularidad con la que cuenta. A diferencia de *Apache Cordova*, que como veremos más adelante, utiliza una *WebView* sobre la cual corre la aplicación web, *Appcelerator* traduce el código JavaScript al código nativo de la plataforma. La interfaz gráfica se crea desde el propio **JS**. Solo se encuentran disponibles las plataformas Android y iOS, que aunque menor, puede ser un inconveniente en según que proyecto. Además, el código a utilizar en Android y en iOS presentan algunas diferencias, lo que nos puede llevar a tener que mantener dos códigos, aunque con variaciones mínimas entre ellos.
3. **React Native**: Este framework esta apoyado por Facebook, quien lo creo y libero hace unos años. React Native no se basa *Apache Cordova* como hace

Ionic, si no que ejecuta el código **JS** de la aplicación utilizando el motor Javascript de la plataforma (V8 en Android, Javascript Core en iOS) y a medida que se va ejecutando la aplicación, un interprete lee el código **HTML** y crea una vista nativa según este. Esto le proporciona la ventaja de que los elementos que se ven en realidad son nativos, haciendo que se vean mejor que los que generan otros frameworks que se basan en imitarlos usando estilos **CSS**.

4. NativeScript: Similar a React Native en cuanto a funcionamiento, pero permitiendo además el uso de TypeScript y Angular2 para programar nuestra aplicación y no tener que estar limitados al uso de ReactJS. Un inconveniente que tiene respecto a React Native es la empresa que está detrás. Mientras que ReactJS cuenta con Facebook a sus espaldas, NativeScript está soportado Telerik, que sin querer menospreciarles, no poseen el mismo empuje con el que cuenta el anterior.
5. Framework 7: Por último nos encontramos con Framework 7, el cual se centra únicamente en iOS y en Android. Permite la creación de aplicaciones híbridas usando PhoneGap, o de WebApps, es decir, aplicaciones que se ejecutan en el navegador, pero manteniendo el aspecto visual de la plataforma donde se ejecuta.



El término **look and feel** se refiere al aspecto y comportamiento que presenta una interfaz gráfica de cara al usuario. Por lo general, una aplicación o sistema cuentan con una serie de características visuales (paleta de colores, tipo y tamaño de letra, animaciones, iconos, ...) que el usuario es capaz de reconocer como propio de esta aplicación o sistema.

Como en los anteriores frameworks, podemos comprobar la popularidad de los comentados según sus búsquedas en Google.

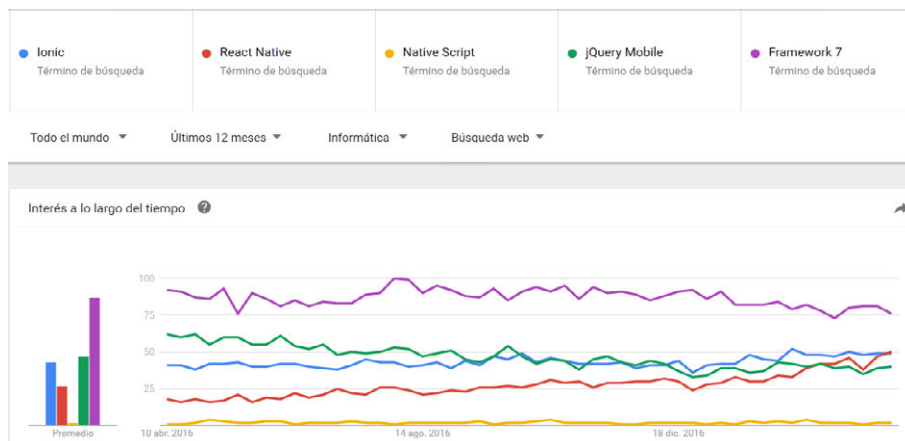


Figura 2.4: Vemos como quitando Framework 7 y NativeScript, el resto de frameworks están a la par.

Hay que decir que tanto Framework 7 como jQuery Mobile también son usados para la creación de WebApps y de paginas web con aspecto *responsive*, por lo que cuentan con cierta ventaja a la hora de hablar de popularidad. Si miramos por foros y en diversas comunidades de desarrolladores, parece que las opciones preferidas a la hora de crear aplicaciones híbridas son Ionic y React Native.

2.2. Angular

AngularJS¹⁶ es un framework para JavaScript utilizado en la creación de aplicaciones web funcionales, también conocidas como aplicaciones de una sola página o en inglés *Single Page Application*. Se trata de un framework de código abierto desarrollado y mantenido por Google. En los últimos años, AngularJS se ha convertido en un framework muy popular dentro de la comunidad de desarrolladores, siendo uno de los más utilizado.

AngularJS is what HTML would have been, had it been designed for building web-apps. Declarative templates with data-binding, MVW, MVVM, MVC, dependency injection and great testability story all implemented with pure client-side JavaScript!

Web del proyecto AngularJS.

AngularJS trata de traer el orden al código de nuestra aplicación. Para ello dispone de las facilidades necesarias a la hora de desarrollar para poder separar la lógica de negocio de la presentación. Se dice que el patrón de arquitectura que

¹⁶<https://angular.io/>

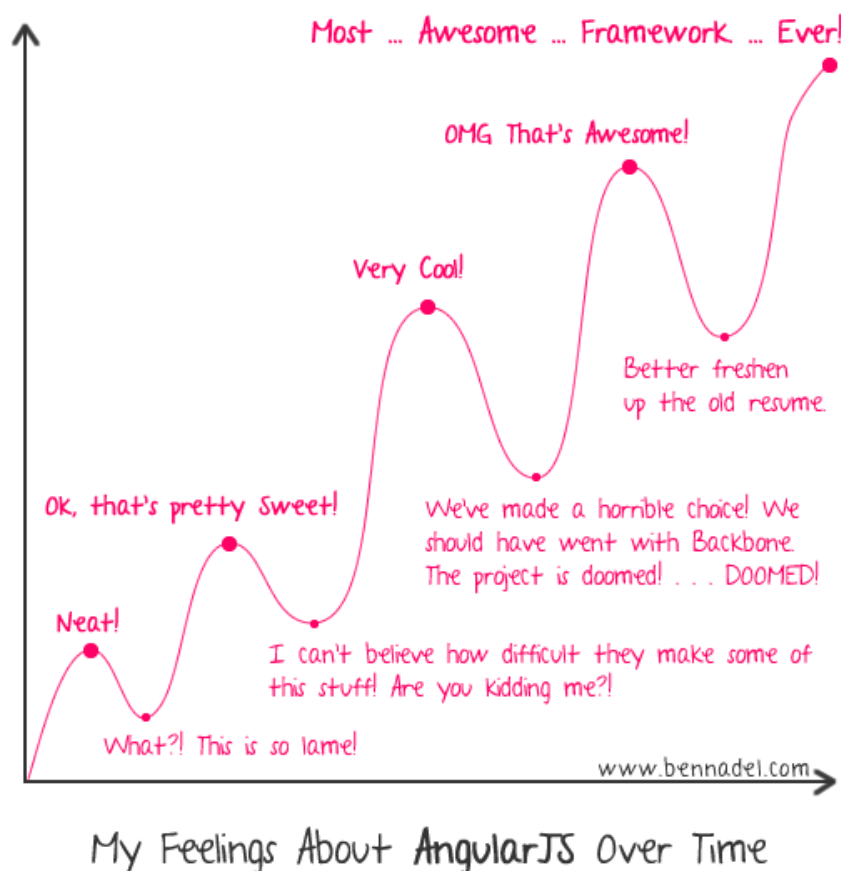


Figura 2.5: El programador Ben Nadel describió (y de manera muy acertada) su experiencia con Angular en su blog de esta forma. [Nad13]

sigue es MVW o MV* (Model-View-Whatever!). También se consigue una mejora del rendimiento a la hora de usar AngularJS, entre otras cosas, debido a que no es necesario recorrer el árbol DOM como si que ocurre con otras librerías como *jQuery*.

Como ya se vio en la sección **Framework Javascript**, Angular se ha hecho muy popular siendo hoy en día el framework que goza de más popularidad con diferencia. Esto es importante a la hora de decidir si usarlo o no en nuestro proyecto, ya que nos garantiza el soporte durante los próximos años. Pero hay que tener en cuenta que se trata de un framework complejo, con una curva de aprendizaje que es necesario que tengamos también en cuenta.

Forma parte del stack **MongoDB-ExpressJS-Angular-NodeJS**¹⁷ junto con MongoDB, Express y Node.JS. Hablaremos de él mas adelante en el apartado **MEAN**.

¹⁷<http://mean.io/>

2.2.1. Angular versión 2

[Sha16] Y llegó la conferencia europea de AngularJS de 2014 En esta conferencia se dio la noticia de que se estaba desarrollando Angular 2.0 y que esta nueva versión sería incompatible con la primera y no habría una hoja de migración desde 1.x a 2.0. En Angular 2.0 se acaba con pilares de la versión tan importantes como es el *\$scope* o los *controladores*, que son sustituidos por los *components*. Estos grandes cambios fueron muy criticados por la comunidad de desarrolladores, los cuales veían como todo lo aprendido iba a dejar de ser útil. Podemos decir por tanto que más que una segunda versión de AngularJS, se trata de un framework totalmente nuevo.



Angular, AngularJS, Angular2, ...¿Cómo debemos de llamar a este framework?. Pues bien, se ha establecido que:

1. Las versiones 1.x serán nombradas como **AngularJS**.
2. Las versiones 2.x en adelante tomarán el nombre de **Angular** a secas.
3. Se indicará el número de versión si se quiere mencionar a una característica concreta de esta. Por ejemplo, **Angular 2.4**
4. La versión **SEMVER** completa si queremos indicar algún bug.



En el momento de escribir esta memoria, acaba de salir la versión 4 de Angular. Esta nueva versión es compatible hacia atrás con la versión 2. También se anunció la futura versión 5. El motivo de saltar de numeración del 2 al 4 es para unificar la versión de los paquetes del core de Angular ya que estos paquetes se distribuyen por separado y utilizan **SEMVER**. Además, todos estos paquetes se encuentran en la misma versión que Angular excepto uno, *angular/route*, el cual se encuentra en la versión 3.x. Por ello, han decidido saltarse la versión 3 e ir directamente a la 4.

Aunque Angular es perfectamente compatible con Vanilla JavaScript, se recomienda el uso de TypeScript a la hora de programar nuestra aplicación.

Gracias a TypeScript el desarrollo será mucho mas sencillo, además, el propio framework está escrito utilizando TypeScript, por lo que la compatibilidad es completa.

2.2.1.1. TypeScript

TypeScript¹⁸ es un superconjunto de JavaScript, es decir, se trata de un lenguaje basado en JavaScript, al cuál le añade grandes beneficios. Es por esto que podemos usar código JavaScript en nuestro código TypeScript con la tranquilidad de que va funcionar correctamente. TypeScript implementa todas las funcionalidades que define ECMAScript 6¹⁹, añadiendo además algunas propias (algunas de las cuales se plantean introducir en ECMAScript 7).

Para que las aplicaciones escritas usando TypeScript puedan ser usadas en motores de ejecución de JavaScript, por ejemplo en los navegadores, la mayoría de los cuales reconocen ECMAScript 5, el código TypeScript debe ser compilado, lo que hace que sea traducido a JavaScript. Es este código traducido al JavaScript el que es realmente ejecutado por el navegador (o la WebView).

Algunas de las características (parte de las cuales son compartidas por los estándares mencionados antes) que TypeScript añade sobre JavaScript son:

1. El uso de clases, haciendo del lenguaje un lenguaje orientado a objetos, lo que es de gran ayuda en proyectos de cierta envergadura.
2. El tipado de variables y verificación en tiempo de compilación.
3. Interfaces y herencia.
4. Uso de anotaciones.
5. Funciones lambda²⁰.
6. Uso de clases *mixin*²¹.
7. *Tuplas* y las *Enumeraciones*.
8. ...

¹⁸<https://www.typescriptlang.org/>

¹⁹<http://es6-features.org/>

²⁰<https://basarat.gitbooks.io/typescript/docs/arrow-functions.html>

²¹<https://www.typescriptlang.org/docs/handbook/mixins.html>

2.2.1.2. Aspectos más importantes en Angular

[Ori16; COD16] Después del breve apunte sobre TypeScript, sigamos hablando de Angular. Sería necesario bastante más que un capítulo dentro del marco tecnológico de este PFC para poder explicar en detalle todas las características que ofrece Angular2. Por ello, vamos a centrarnos en comentar aquellas más relevantes y así estar algo más preparados a la hora de enfrentarnos a la realización de las prácticas más adelante. Como veremos, Angular se aprovecha de las *clases*, las *anotaciones*, la *herencia*, ...que define TypeScript para implementar las funcionalidades que ofrece, por ejemplo, los *componentes* o *directivas* que explicaremos a continuación se implementan como clases con unos metadatos definidos mediante una anotación.

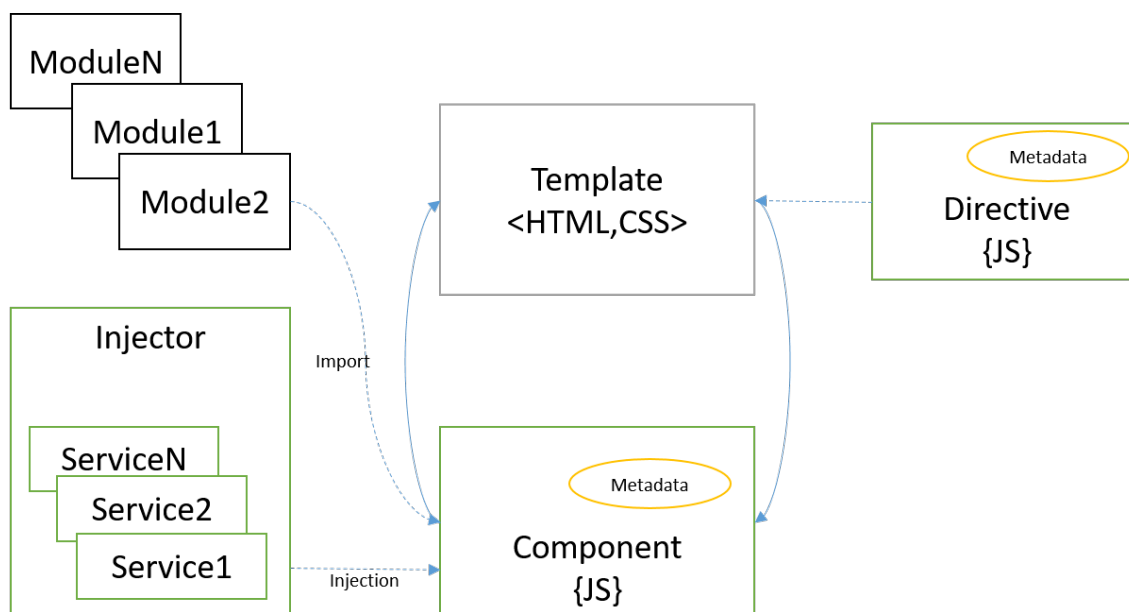


Figura 2.6: Esquema de algunos de los elementos que encontramos en una aplicación realizada con Angular y como interactúan entre ellos.

Módulos Angular es modular. Se organiza en módulos los cuales se caracterizan por tener un único objetivo. Estos módulos suelen exportar ciertos elementos que pueden ser usados por otros módulos. Un ejemplo de módulos serían las librerías que forman Angular y que debemos importar cada vez que queremos usar alguna de sus facilidades.



Dentro de las librerías de Angular, las más importantes son:

1. @angular/core
2. @angular/common
3. @angular/router
4. @angular/http

Veremos el uso de algunas de ellas a lo largo de **Prácticas**

Components Una aplicación hecha con *Angular* está formada por *componentes* los cuales cuelgan uno de otros formando una especie de árbol. Existe un *component* raíz del cual cuelgan el resto de *componentes*. Esta estructura es similar a la que nos podemos encontrar en un **HTML**, donde todos los elementos se encuentran anidados dentro de otro hasta llegar al elemento **<body>**.

Un componente está definido por una clase a la que se le añade la anotación *@component*, donde se definen los metadatos del componente, y un **template** que define la vista (engloba el **HTML** y el *CSS*). Tanto las propiedades como los métodos definidos en un componentes son accesibles desde la vista. Un elemento del **DOM** solo puede estar asociado a un componente.

Los componentes se utilizan para separar la aplicación en pedazos más pequeños.

Data Binding En mi opinión, una de las características más potentes de Angular. Gracias al *Data Binding* que ofrece Angular, podemos conectar valores entre la vista y la parte lógica de nuestro componente y convertir acciones que realice el usuario en llamadas a métodos sin tener que escribir toda la lógica que hay detrás de esto.

Directivas Una directiva nos permite añadir comportamiento a un elemento **DOM** ya existente. En realidad, los *componentes* son *directiva* con una vista asignada, mientras que una *directiva* puede ser reusada en diferentes elementos del **DOM**, además de poder usar varias sobre un mismo elemento.

Las directivas se definen utilizando la anotación *@directive*, en el que también se definen los metadatos de la directiva.

Servicios Los servicios son clases con una funcionalidad específica y que es compartido por los componentes que forman la aplicación. Estos servicios son

accesibles desde los diferentes componentes gracias al inyector de dependencias (*Dependency Injection*). Para definir un servicio se utiliza el decorador *@injectable*.

Un uso típico de los servicios es la recuperar datos desde una fuente externa para ser usados en nuestra aplicación.

Dependency Injection Este término hace referencia a un mecanismo para proporcionar nuevas instancias de una clase con todas aquellas dependencias que requiere, facilitando así la modularidad. Se utiliza sobretodo para inyectar los servicios a los componentes que los van a usar, y todo esto con solo especificar en el constructor del componente los servicios que espera.

A la hora de crear nuestra aplicación, debemos de indicar los servicios que se van a utilizar a través de un *provider* definido en el módulo lo contiene. Así, el inyector sabe como debe instanciar las dependencias que le "soliciten" los diferentes componentes (u otros servicios).

2.2.2. MEAN

[Ala15] Aunque esta sección está englobada dentro de **Angular**, no nos confundamos, es Angular la que en verdad forma parte del stack **MEAN**²². Aunque no veremos nada sobre este a lo largo de este **PFC**, y el único componente de los que esta formado y vamos a tratar sea Angular, me ha parecido interesante introducirlo mediante una breve introducción. Pero, ¿a qué nos referimos cuando hablamos del stack **MEAN**?

Se conoce como stack **MEAN** al conjunto de tecnologías que permiten la creación de aplicaciones distribuidas utilizando únicamente, y en todas sus fases, JavaScript. El nombre se debe a que las cuatro tecnologías principales que lo forman, son **M**ongoDB, **E**xpress, **A**ngular y **N**odeJS. Esto no quiere decir que no se puedan usar otros elementos como Browser, Selenium, Sass, ...para facilitarnos el trabajo.

²²<http://mean.io/>

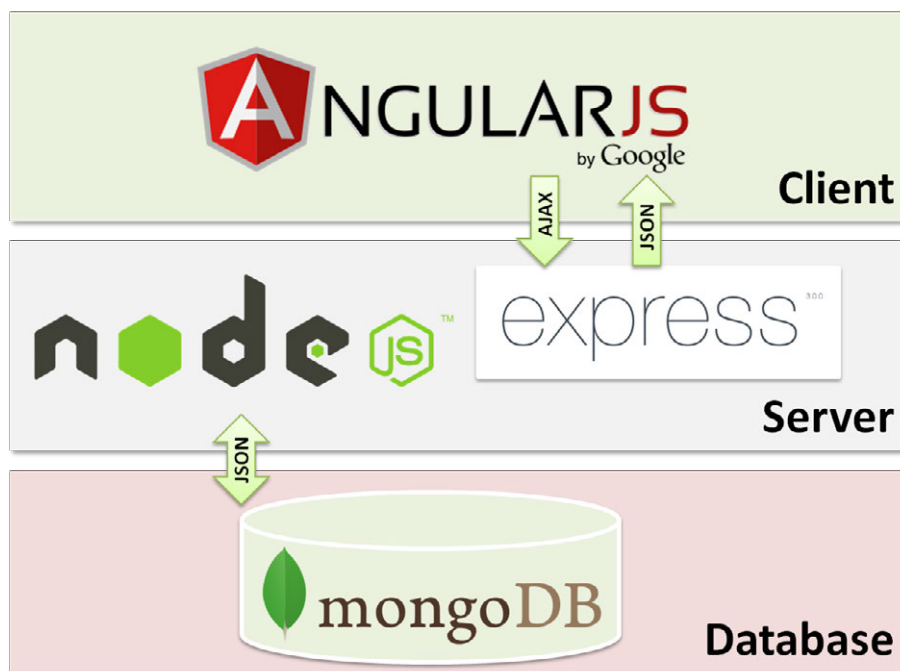


Figura 2.7: Los cuatro elementos principales del stack en cada una de las fases.

1. NodeJS: Un entorno de ejecución de JavaScript, basado a su vez en el motor de ejecución V8 desarrollado por Google. Este proyecto ha permitido que JavaScript de el salto del lado del cliente a poder ser usado también en la parte del servidor (aunque en sus inicios ya lo podíamos encontrar, pero no tuvo mucho recorrido). Provee de una arquitectura orientada a eventos (como la de los navegadores) así como una serie de APIs asíncronas que le proporcionan un rendimiento y una escalabilidad muy elevadas.
2. Express²³: Framework utilizado para implementar aplicaciones web sobre NodeJS. Aunque podemos crear nuestro servidor sin necesidad de utilizar ningún tipo de herramienta ajena a NodeJS, ExpressJS nos facilita esta tarea cubriendo las principales necesidades de cualquier servidor web (gestión de peticiones y respuestas, cabeceras, rutas, vistas, ...).
3. MongoDB²⁴: En la parte de almacenamiento se encuentra esta **Base de datos (BBDD) NoSQL (NoSQL)** de tipo documental, que almacena la información en un formato similar al **JavaScript Object Notation (JSON)**.
4. Angular: Utilizado como no podría ser de otra manera para implementar el cliente de la aplicación distribuida.

²³<http://expressjs.com/>

²⁴<https://www.mongodb.com/>

La razón por la que se ha hecho esta pequeña introducción a **MEAN** es por la importancia que está consiguiendo dentro de la comunidad de desarrolladores. JavaScript se esta posicionando como uno de los lenguajes principales, y herramientas como esta, hacen que su uso no se restrinja a solo el lado del cliente. Además, una aplicación móvil, y de esto si que trata el este **PFC**, suele apoyarse en servidores, por lo que es conveniente saber que alternativas se encuentran disponibles para su implementación, y si es usando un lenguaje (JavaScript) y una tecnología(NodeJS) ya conocida, mejor.



Por mi experiencia en el desarrollo de aplicaciones distribuidas, recomendaría a un desarrollador sin experiencia en backend empezar con un framework más estructurado y robusto, como puede ser Django (framework para Python), y una vez que se tiene un poco de experiencia, dar el salto (si se quiere) al uso tecnologías relacionadas con NodeJS.

2.3. Apache Cordova

Apache Cordova, o Cordova a secas, se trata de un entorno de desarrollo de aplicaciones móviles utilizando tecnologías web, **HTML** y **CSS** para la presentación, *JavaScript* para la lógica. Es posible que hayas escuchado hablar sobre algo llamado *PhoneGap*, pues bien, antes de continuar, convendría explicar que relación existe entre ambos. Si nos remontamos a 2008, una empresa llamada *Nitobi* presenta PhoneGap, un entorno para la creación de aplicaciones móviles usando tecnologías web (de momento no hay diferencias con Cordova). En 2011 deciden donar el código a la fundación *Apache*, lo que supone que se convierta en software libre, y debido al éxito que tiene, ese mismo año, *Adobe* compra *Nitobi*, quedándose con sus empleados, sus proyectos, sus marcas, ...y dentro de este paquete se encuentra PhoneGap. Decide entonces mantener PhoneGap como software Open Source, pero debido a que la marca PhoneGap pertenece a *Adobe*, optan por cambiarle el nombre, surgiendo así Apache Cordova²⁵. Así que como vemos, ambos proyectos se tratan del mismo. La única diferencia destacable es el servicio *Adobe PhoneGap Build*²⁶, que nos permite compilar nuestro código en la nube a cualquier plataforma sin necesidad de tener que tener instalado en nuestro

²⁵<https://www.campusmvp.es/recursos/post/PhoneGap-o-Apache-Cordova-que-diferencia-hay.aspx>

²⁶<https://build.phonegap.com/>

ordenador la SDK de cada plataforma. Entonces, ¿Cuál debemos usar?. Cualquiera de los dos, al final, se trata de lo mismo. Lo único a tener en cuenta es si vamos a usar alguno de los servicios que ofrece Adobe.

Tras esta breve explicación, necesaria ya que siempre surge la duda, veamos como funciona Cordova. El objetivo es conseguir que nuestro código realizado con tecnologías web sea ejecutado en cualquier plataforma como si de una aplicación nativa se tratase. Para conseguir esto, Cordova proporciona un entorno nativo para cada plataforma sobre el cual, se ejecutará nuestra aplicación. Esta envoltura está compuesta por un navegador (WebView) que se encarga de ejecutar nuestro código, y si son necesario, una serie de plugins que permiten la comunicación entre nuestro código y las APIs de cada plataforma a través de una API unificada. Estos plugins pueden ser ofrecidos por el propio Cordova, como ser realizados por terceros.

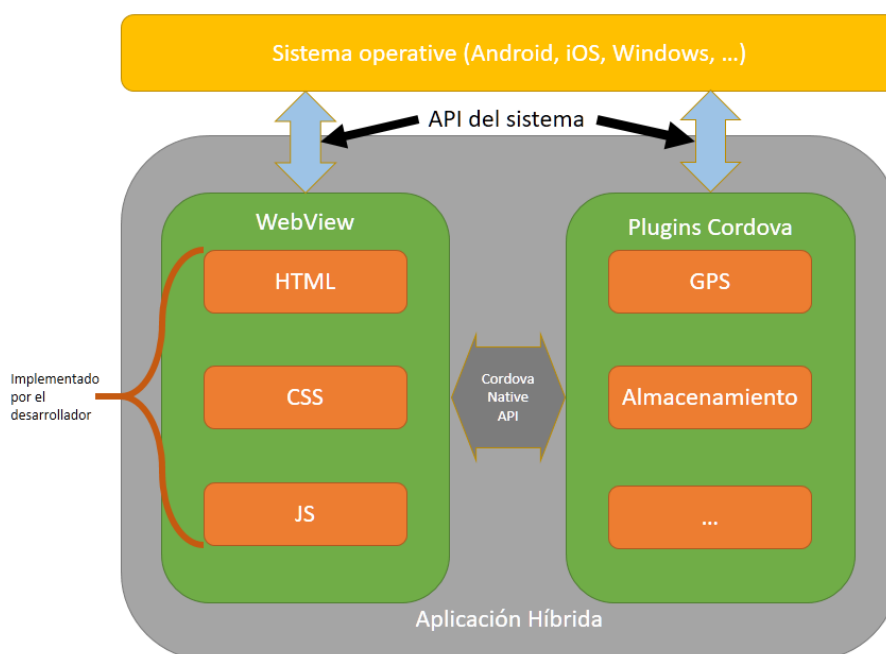


Figura 2.8: Este diagrama muestra de forma simplificada como está compuesta una aplicación híbrida que utiliza Apache Cordova.

Cordova cuenta con una interfaz de línea de comandos (**Command Line Interface (CLI)**) con la podremos crear proyectos, compilarlos, ejecutarlos, También nos servirá para añadir plugin a nuestro proyecto. El único requisito será el tener instalado en la máquina el SDK de las plataformas a las que vaya destinada nuestra aplicación.

Apache Cordova se encuentra disponible como paquete de **The Node Package Manager (npm)**, desde donde puede ser instalada. Ver el anexo sobre **Ionic**.

Algunos de los frameworks que se utilizan para crear aplicaciones híbridas se

colocan por “encima” de Cordova. Este es el caso de Ionic, framework en el que se centra este **PFC**. Muchos de los comandos disponibles en Ionic **CLI** no son más que envoltorios sobre el **CLI** de Cordova, así como también ocurre con Ionic Native, que se coloca sobre los plugin de Cordova.

2.4. Ionic

Ionic es un framework de código abierto utilizado para el desarrollo de aplicaciones híbridas para dispositivos móviles. En estos momentos se encuentra disponible la versión 2 de Ionic, que es la que vamos a usar para la realización de las prácticas que veremos en el capítulo siguiente.

Ionic funciona por encima de Apache Cordova, ayudando a conseguir que nuestras aplicaciones desarrolladas con tecnología web luzcan de cara al usuario lo más parecidas posible a las aplicaciones nativas.

Una de las características más importantes de Ionic2 es el uso que hace de Angular. Esto le permite crear aplicaciones robustas y fácilmente mantenible. El rendimiento también es otra característica fundamental. Ionic2 evita manipular el árbol DOM y hace cero uso de librerías tipo jQuery, esto unido al ya mencionado uso de Angular permite que las aplicaciones se vean realmente fluidas.

Otra facilidad que da al desarrollador, es lo que llaman Ionic Native²⁷. Esto es un conjunto de “envoltorios” sobre los plugins de Cordova, escritos en ES5/ES6/TypeScript y que facilitan su integración en nuestra aplicación.

Un aspecto fundamental de las aplicaciones es el apartado gráfico. Nuestra aplicación, además de funcionar bien, debe verse bien. Esto puede ser un problema ya que el mismo código deben servir para distintos tipos de terminales los cuales no comparten las mismas características de pantalla. Junto a esto, también nos enfrentamos al hecho de que cada plataforma aplica un estilo diferente a cada elemento que conforma una aplicación. Así por ejemplo, el mismo componente no se ve igual en un terminal que corra Android que uno que utilicen iOS.

Ionic ofrece un buen número de componentes²⁸ que ayudan al desarrollador a enfrentarse a ambos problemas. Por otro lado, al igual que hace con Angular, Ionic implementa por debajo **Sass**. **Sass** es un lenguaje cuyo propósito es suplir alguna de las carencias que tiene **CSS**. Cuando se compila la aplicación, los ficheros **Sass** son traducidos a **CSS** para que puedan ser interpretados por los navegadores. Esto es transparente para el programador, ya que es Ionic el que se encarga de realizar esta

²⁷<https://ionicframework.com/docs/v2/native/>

²⁸<https://ionicframework.com/docs/v2/components/>

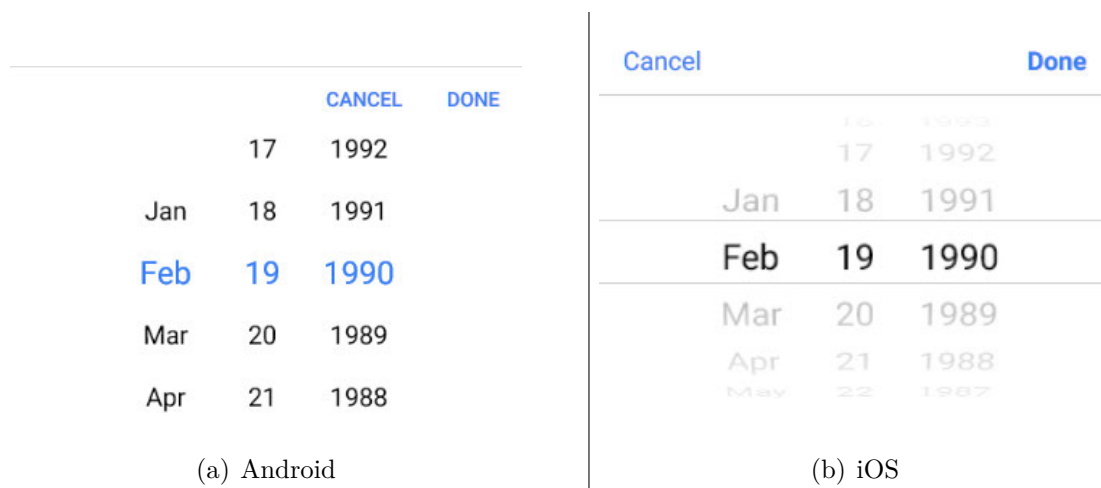


Figura 2.9: Comparación de un selector de fechas en diferentes plataformas.

traducción.

Como ya se ha comentado, Ionic es un framework de código abierto, pudiendo hacer uso de todas sus características de forma totalmente gratuita. A parte de esto, existen cuentas de pago (se pueden consultar en <https://ionicframework.com/pricing/index3.html>) para usuario o empresas que requieran un de un servicio más completo, como uso de Ionic Creator, soporte avanzado, uso de analíticas,

También existe un “market”²⁹ en el que poder descargar temas, plugins o plantillas creados por otros usuarios (no todos los recursos son gratuitos) o compartir los nuestros con el resto del mundo. Antes de empezar a desarrollar nuestra aplicación, es recomendable revisar este market ya que podemos encontrar recursos que nos resulten útiles y evitarnos el coste de desarrollarlos ad-hoc para nuestro proyecto.

2.4.1. Ionic CLI

Ionic CLI es una herramienta que ofrece Ionic para la realización de múltiples tareas, como veremos mas adelante. Tal como la definen en la página, es como una navaja suiza a la hora de desarrollar con este framework.

La herramienta Ionic CLI se utiliza a través de la consola de comandos, indicando en el comando la acción a realizar junto con las diferentes opciones. El comando para lanzar una de estas acciones es `ionic <accion>`. Cada acción tiene sus propias opciones. Muchas de estas acciones no son más que un envoltorio a herramientas propias de Cordova, como por ejemplo `build`. A continuación, se va a listar estas

²⁹<https://market.ionic.io/>

acciones junto a sus opciones más utilizadas:

1. `start`³⁰: Crea un nuevo proyecto en el directorio indicado a partir de una plantilla. Sus opciones son:
 - a) `PATH`: Directorio en el que crear el proyecto. Si no se indica, se utiliza la ruta en la que se ha ejecutado el comando.
 - b) `-template/-t`: Plantilla a utilizar para crear la app. Se pueden ver plantillas (a las que llaman *starter* en el *market* de Ionic. Para utilizar una plantilla vacía, utilizar la conocida como *blank*.
 - c) `-v2`: Indica que se quiere utilizar la versión 2 de Ionic.
2. `serve`³¹: Inicia un servidor local para servir la aplicación y poder ser probada desde un navegador web. Muy útil mientras se está desarrollando.
3. `generate`³²: Comando que genera la estructura necesaria para desarrollar una nueva página o un servicio en nuestra app.
4. `platform`³³: Este comando se utiliza para manejar las plataformas a las cuales esta destinada nuestra aplicación.
 - a) `list`: Lista las plataformas y versión de estas disponible.
 - b) `add`: Añade una nueva plataforma objetivo a nuestro proyecto.
 - c) `remove`: Elimina una plataforma objetivo de nuestro proyecto.
5. `run` o `emulate`³⁴: Inicia nuestra aplicación en un terminal conectado a nuestro PC o en un emulador. Es necesario que la plataforma que utiliza este terminal/emulador se haya incluido con anterioridad al proyecto (mediante el comando anterior).
 - a) `-livereload` (en fase beta en el momento de escribir este documento): Aplica los cambios que hagamos de manera automática funcionando de manera similar a como funciona *serve*. Muy útil también a la hora de depurar nuestra aplicación

³⁰[urlhttp://ionicframework.com/docs/v2/cli/start/](http://ionicframework.com/docs/v2/cli/start/)

³¹[urlhttp://ionicframework.com/docs/v2/cli/start/](http://ionicframework.com/docs/v2/cli/start/)

³²[urlhttp://ionicframework.com/docs/v2/cli/generate/](http://ionicframework.com/docs/v2/cli/generate/)

³³[urlhttp://ionicframework.com/docs/v2/cli/platform/](http://ionicframework.com/docs/v2/cli/platform/)

³⁴[urlhttp://ionicframework.com/docs/v2/cli/run/](http://ionicframework.com/docs/v2/cli/run/) o <http://ionicframework.com/docs/v2/cli/emulate/>

6. `build`³⁵: Genera una *app* para la plataforma que indiquemos (al igual que ocurría con el comando anterior, la plataforma debe ser añadida previamente al proyecto), la cual es almacenada en el directorio *PATH/platform*.

2.4.2. Ionic Native

Ionic Native es un conjunto de envoltorios sobre los plugins de Cordova unificando en una única interfaz **API** las **APIs** de las diferentes plataformas que soporta. Para facilitar su uso desde nuestra aplicación, implementa diferentes facilidades que ofrece Angular y TypeScript como *clases* y *promesas*.

Los diferentes módulos que conforman Ionic Native pueden ser instalados usando el **CLI** de Ionic de manera análoga a como se instalan los plugins de Apache Cordova.

Como ejemplo, el siguiente comando se utiliza para instalar el módulo que nos permite hacer uso de la base de datos⁵ del dispositivo:

```
# ionic plugin add cordova-sqlite-storage
```

³⁵[urlhttp://ionicframework.com/docs/v2/cli/build/](http://ionicframework.com/docs/v2/cli/build/)

CAPÍTULO 3

PRÁCTICAS

Una máxima en el mundo de la programación es: *"La mejor forma de aprender a programar es programando"*. En este capítulo se van a proponer una serie de prácticas con los que poder ver Ionic en acción. Para la realización de estas aplicaciones, daremos por sentado que se dispone de un ordenador con Ionic2 instalado (ver anexo [Ionic](#)) y un editor de código, en nuestro caso utilizaremos [Atom](#). Se recomienda crear un nuevo directorio que sirva como área de trabajo en el que guardaremos los proyectos.

Como se suele hacer en cualquier lenguaje de programación, comenzaremos programando un *Hello World!*, esto es, una aplicación sencilla que lo único que haga sea imprimir el mensaje "Hello World!" en el dispositivo de salida. Esta práctica servirá de toma de contacto con la estructura de las aplicaciones construidas con Ionic. A continuación veremos como depurar nuestro desarrollo utilizando un emulador o directamente sobre un dispositivo real y como construir nuestra aplicación. Con esta base, continuaremos viendo aspecto más específico de sobre la tecnología, como el *Data Binding*, los servicios, las animaciones, el uso de la [API](#) del sistema, ...a través de tres prácticas es las que programaremos: un **Cronómetro 3.3**, un **Paisaje 3.4** y una aplicación que gestione **Recordatorios asociados a una localización 3.5**.

Se dará por supuesto que el lector de este documento cuenta con unos conocimientos básicos de los lenguajes JavaScript, [HTML](#) y [CSS](#), por lo que no se profundizará en conceptos referentes a estas tecnologías si no es estrictamente necesario.

3.1. Hola Mundo!!!

El *Hola Mundo* (o *Hello World*) es la primera aplicación que típicamente se programa a la hora de aprender una nueva tecnología. Su funcionamiento es sencillo, imprimir por pantalla (o por el dispositivo de salida correspondiente) el saludo que da nombre a la aplicación. Con esta práctica se abordará la creación de un nuevo proyecto. También se creará una pequeña página para la que trabajaremos sobre el código **HTML** de la aplicación. Veremos como probar la aplicación usando el navegador (Chrome en nuestro caso) de la máquina de desarrollo que utilizamos para desarrollar. Por último, haremos una primera aproximación a la característica de *Data Binding* de Angular uniendo dos elementos del **HTML** que aparecerán por pantalla.

Para crear nuestro primer proyecto, abrimos el PowerShell de Windows (o la consola de comandos tradicional, según disponibilidad y gustos personales) y nos dirigimos a nuestro directorio de trabajo. Aquí se creará el directorio que contendrá nuestro proyecto. En este caso queremos un proyecto en blanco, usando la plantilla *blank*, para ello ejecutaremos el siguiente comando:

```
# ionic start HelloWorld blank --v2
```

Esto creará un directorio llamado HelloWorld dentro del cual veremos el proyecto sobre el cual vamos a trabajar.



Con la opción `-v2` indicamos que el queremos usar la versión de ionic para el nuevo proyecto.

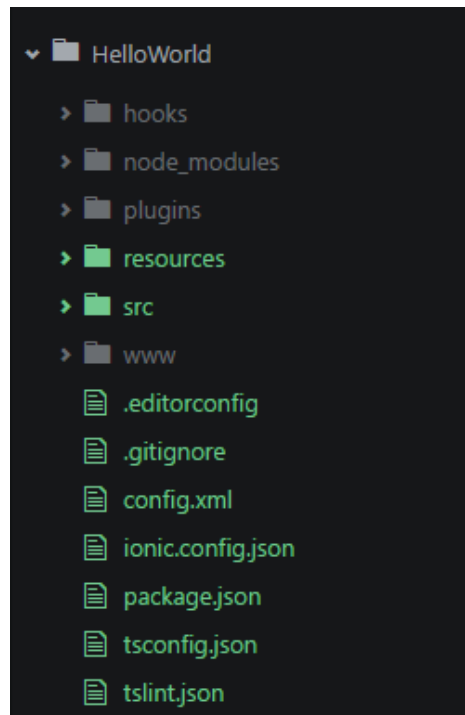


Figura 3.1: Estructura del proyecto Hello World al crearlo visto en Atom.

Podemos ejecutar la aplicación tal cual se encuentra en este momento y comprobar que funciona correctamente. Para esto, utilizando Ionic CLI, podremos ejecutar la aplicación sobre un servidor el cual servirá de servidor local para nuestra aplicación permitiéndonos acceder a ella a través de un navegador web:

```
# ionic serve
```

Tras unos segundos, nos aparecerá un mensaje indicando la **Uniform Resource Locator (URL)** en la cual se encuentra disponible la aplicación. Por lo general, cuando el servidor termina de arrancar, se abrirá el navegador configurado por defecto en nuestro sistema apuntando directamente a la dirección anterior. De no ser así, copiamos la **URL** y la abrimos manualmente. Vemos la página que por defecto viene con la plantilla utilizada.

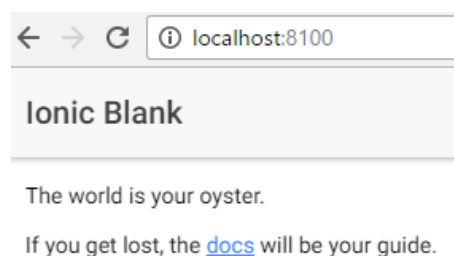


Figura 3.2: Así es como se ve el template blank al ejecutarse.

Vamos a editar la aplicación para que tenga el comportamiento antes indicado, es decir, que en nuestro caso es mostrar por pantalla el texto "Hello World!". Abrimos con el editor que vayamos a usar el fichero `./src/pages/home/home.html`. Dentro de este fichero se encuentra el código de la página que acabamos de ver en el navegador.

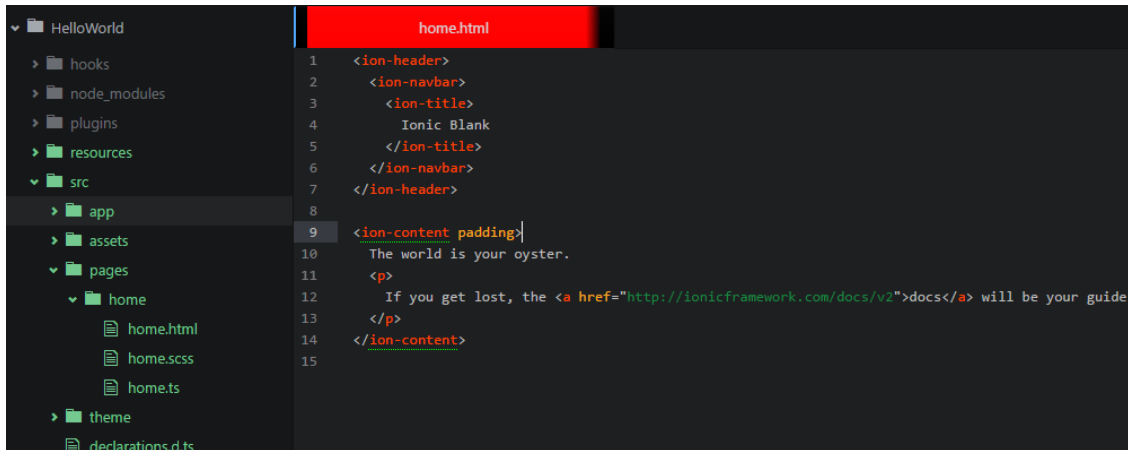


Figura 3.3: Página principal del template blank.



Si utilizamos un editor un poco avanzado, nos ofrecerá la posibilidad de abrir un directorio de trabajo entero, pudiendo ver la estructura de ficheros de la aplicación y navegar fácilmente entre ellos.

Podemos apreciar que la página dispone de una cabecera definida por el tag `<ion-header>` y el cuerpo de la página definido por el tag `<ion-content padding>`. Modificaremos el código de la siguiente manera:

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Hello App
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <p>HELLO WORLD!!!</p>
</ion-content>
```

Una vez guardado el cambio, veremos en la ventana del PowerShell en la que hemos arrancado el servidor de Ionic que la aplicación se ha compilado de nuevo, y la propia página en el navegador se ha refrescado mostrando los cambios.

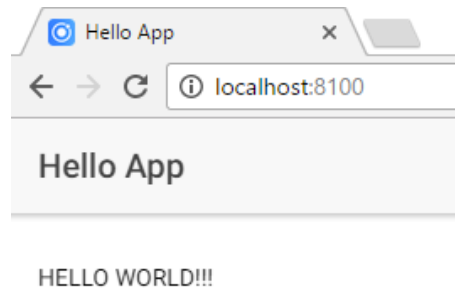


Figura 3.4: Podemos ver nuestro Hello World en el navegador por primera vez.

Vamos a ir un poco más allá con nuestra aplicación. Para ello vamos a utilizar una de las características que nos ofrece Angular, el *Data Binding*. Abrimos de nuevo el fichero *home.html* y modificamos el cuerpo de la siguiente manera:

```
<ion-content padding>
  <ion-list>
    <ion-item>
      <ion-label color="primary" >My name is </ion-label>
      <ion-input [(ngModel)]="name" placeholder="your name" ></
        ion-input>
    </ion-item>
    <ion-item>
      <h1>Hello {{ name }}, nice to meet you.</h1>
    </ion-item>
  </ion-list>
</ion-content>
```

Con esta modificación hemos introducido un cuadro de texto, `<ion-input>`, en el que podremos escribir nuestro nombre. También hemos cambiado el saludo, haciendo que este sea “personalizado” utilizando el valor introducido en `{{ name }}`. Esto se consigue gracias a la potencia del *Data Binding*. Si volvemos a ver nuestro input, vemos que uno de sus atributos es `[(ngModel)]`. Este atributo es una directiva de Angular que vincula un campo, el input en nuestro caso, con una propiedad del componente al que pertenece. Así, al declarar la directiva `[(ngModel)]="name"`, hemos hecho que se cree una propiedad dentro del componente (si no la hemos creado manualmente editando el componente) y que su valor esté vinculado al del input. Ya solo nos queda utilizar este valor donde nos interesa utilizando la expresión `{{ name }}`. Angular también se encarga de renderizar el valor cuando este cambia, por lo que a medida que vayamos escribiendo nuestro nombre, irá actualizándose la vista.

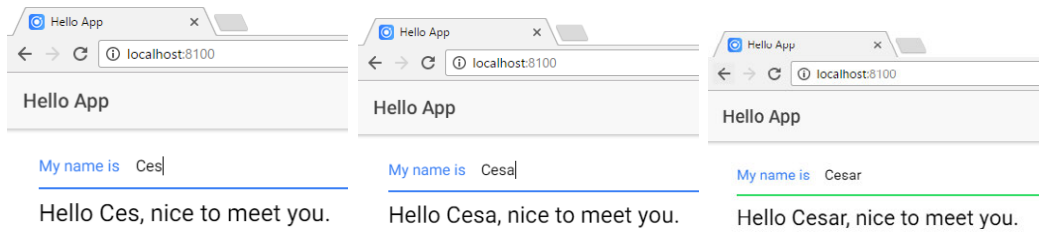


Figura 3.5: Nuestra aplicación ahora nos ofrece un saludo dedicado.

3.2. Construir y emular

Antes de continuar con las siguientes aplicaciones, y ya que tenemos lista nuestro primer desarrollo listo para ser usado, vamos ver como podemos ejecutarlo en un emulador y en un terminal Android real, además de poder depurar el código desde el navegador Chrome como si de una página web se tratase. De esta forma podremos ver probar la aplicación sobre un terminal real de manera fácil y rápida.

Este método no tiene porque sustituir al servidor de desarrollo local que ejecutamos con *ionic serve*. Este servidor se caracteriza por ser más rápido a la hora de testear pequeños cambios, en especial si estos afectan al diseño, y sobrecarga menos la máquina en la que lo ejecutemos que lo que lo hace un emulador. Aún así, resulta evidente la necesidad de probar las aplicaciones en los terminales objetivo. Una limitación muy importante que tiene ejecutar la aplicación sobre el servidor de desarrollo es la imposibilidad de emular las **APIs** de los dispositivos, por lo que si queremos hacer pruebas con los sensores, por ejemplo, el **Global Positioning System (GPS)**, necesitaremos utilizar un emulador o un dispositivo real. Por lo general, ambas formas son usadas a la hora de desarrollar la aplicación eligiendo una u otra según el cambio que queramos probar.

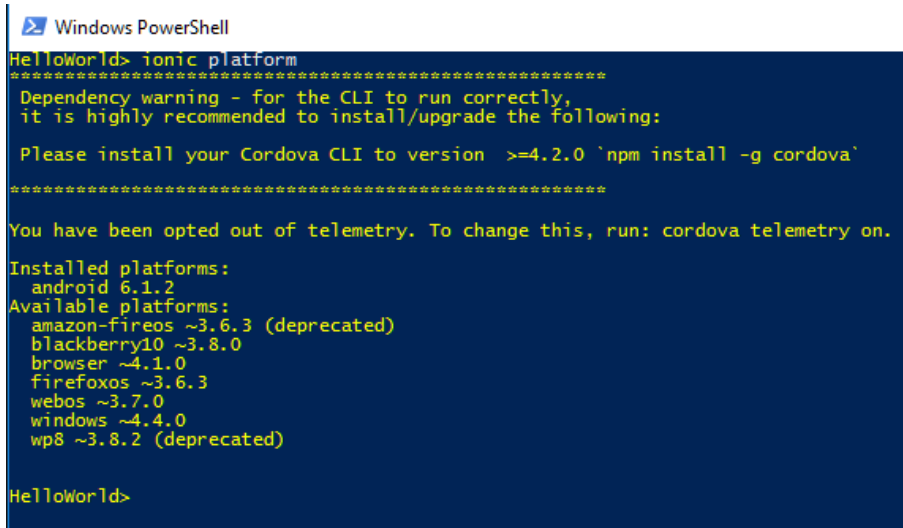
Por último tendremos que generar un ejecutable válido y reconocible por el sistema para poder distribuir nuestra aplicación. Explicaremos como hacer esto generando una **Android Application Package (APKs)** con la que poder instalar la aplicación en terminales Android.

Nos vamos a centrar en terminales con sistema operativo Android, debido a su popularidad, a que los recursos software necesarios son gratuitos y a que no requiere de un sistema operativo especial para poder desarrollar en él. Aún así, lo explicado en este capítulo es valido para otras plataformas siempre y cuando dispongamos de los recursos necesarios.

Antes de continuar, debemos asegurarnos de que tenemos instalado en nuestra máquina la suite de Android. Podemos ver una explicación de como instalar este software necesario en el anexo **Herramientas para trabajar con Android**. Aunque todas la acciones que veamos a continuación las hagamos a través de Ionic CLI, este necesita usar las herramientas que proporciona *Google* para generar una aplicación nativa, que es la que realmente se ejecuta en los terminales. Además, el emulador que utilizaremos forma parte de esta suite.

3.2.1. Añadir plataforma objetivo

Lo primero que debemos hacer es añadir una plataforma objetivo a nuestro proyecto. Con esto configuramos nuestro proyecto indicando para que plataformas queremos que se generen los diferentes instaladores. En primer lugar vamos a ver que plataformas tenemos disponible en nuestro ordenador. Esto lo hacemos utilizando Ionic CLI dentro del directorio de nuestro proyecto.



```
Windows PowerShell
HelloWorld> ionic platform
*****
Dependency warning - for the CLI to run correctly,
it is highly recommended to install/upgrade the following:

Please install your Cordova CLI to version >=4.2.0 'npm install -g cordova'
*****

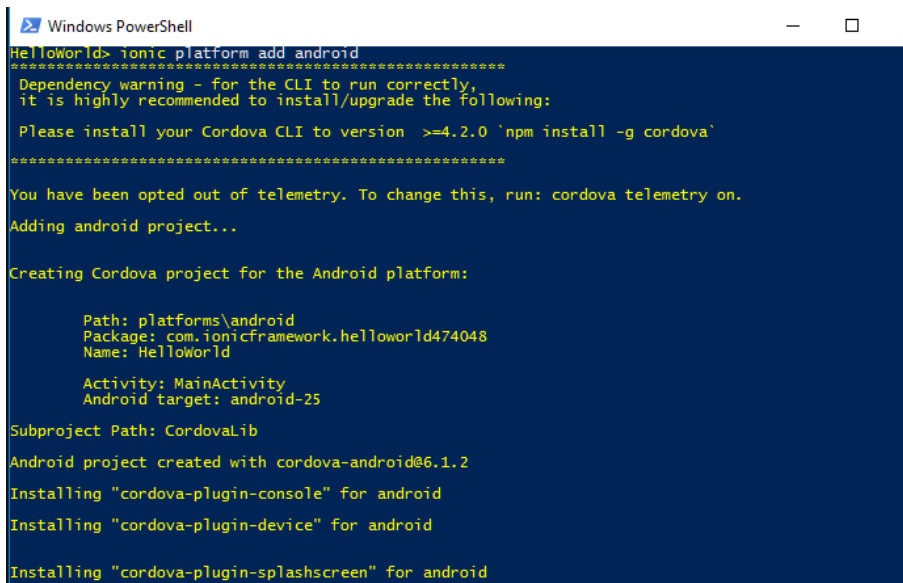
You have been opted out of telemetry. To change this, run: cordova telemetry on.

Installed platforms:
  android 6.1.2
Available platforms:
  amazon-fireos ~3.6.3 (deprecated)
  blackberry10 ~3.8.0
  browser ~4.1.0
  firefoxos ~3.6.3
  webos ~3.7.0
  windows ~4.4.0
  wp8 ~3.8.2 (deprecated)

HelloWorld>
```

Figura 3.6: Listado de plataformas disponibles para incluir en nuestro proyecto.

Como hemos dicho al inicio del capítulo, nos vamos a centrar en la plataforma Android, por lo que es la única que necesitamos añadir.



```
Windows PowerShell
HelloWorld> ionic platform add android
*****
Dependency warning - for the CLI to run correctly,
it is highly recommended to install/upgrade the following:

Please install your Cordova CLI to version >=4.2.0 'npm install -g cordova'
*****

You have been opted out of telemetry. To change this, run: cordova telemetry on.

Adding android project...

Creating Cordova project for the Android platform:

  Path: platforms\android
  Package: com.ionicframework.helloworld474048
  Name: HelloWorld
  Activity: MainActivity
  Android target: android-25

Subproject Path: CordovaLib
Android project created with cordova-android@6.1.2
Installing "cordova-plugin-console" for android
Installing "cordova-plugin-device" for android
Installing "cordova-plugin-splashscreen" for android
```

Figura 3.7: Comando a ejecutar si queremos añadir la plataforma Android a la lista de plataformas objetivo de nuestra aplicación.

Con esto nuestro proyecto estaría preparado para ser ejecutado en un terminal o emulador Android, y para más tarde, generar el **APKs**.

3.2.2. Emular aplicación

Como se ha comentado, Ionic CLI nos permite arrancar un emulador Android en nuestra máquina de desarrollo y ejecutar sobre él la aplicación. Para ello utiliza Android Emulator, un emulador que ofrece Google como parte del **SDK** de Android. Ionic se encarga, ejecutando un único comando, de compilar el proyecto, generar el instalable, arrancar el emulador e instalar en él nuestra aplicación. Un ejemplo del comando que hace todo esto es:

```
# ionic emulate android --target=Nexus5
```

Como vemos, en el comando debemos indicar sobre que plataforma queremos realizar la emulación. Como opción, podemos seleccionar que **AVD** queremos utilizar, en caso de no indicarlo, Ionic CLI utilizará el primero que encuentre. Podemos encontrar más información sobre los **AVD** en el anexo **Herramientas para trabajar con Android**, y más concretamente, en la sección **Android Emulator**.

La ejecución de este comando iniciará una serie de tareas que podremos ir viendo aparecer en la consola. Justo antes de acabar (tarda unos segundos), se iniciará el emulador (si no estaba ya en ejecución) y a continuación, se iniciará la aplicación automáticamente en este.

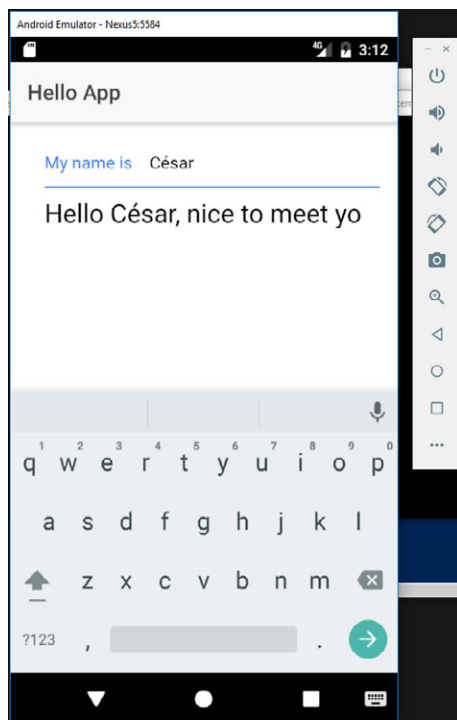


Figura 3.8: Nuestra aplicación vista en el emulador.

3.2.3. Depurar una aplicación que está siendo emulada

Ya hemos dicho que durante la fase de desarrollo, la mejor forma de probar y depurar nuestra aplicación es haciendo uso de la utilidad *ionic serve* y de nuestro navegador favorito. Aún así, en ocasiones es necesario poder depurar la aplicación mientras se ejecuta en el emulador, por ejemplo, para probar el uso de determinados sensores, gestos sobre la pantalla que no podemos realizar en el navegador o simplemente comprobar que la aplicación se ve como debe en un terminal similar al emulado.

Para esto vamos a utilizar el navegador Chrome, que nos permite acceder a las aplicaciones que se ejecuten en el emulador o terminal conectado que usen tecnología web. Aunque aquí nos centramos en aplicaciones híbridas, podemos usar también este mismo método para aplicaciones que se estén ejecutando en el navegador del dispositivo. Vamos a emular la aplicación del mismo modo que hicimos anteriormente:

```
# ionic emulate android --livereload --target=Nexus5
```

Añadiendo *--livereload* conseguimos algunas de las facilidades que tenemos al utilizar *ionic serve*:

1. Ver los cambios realizado en el código de manera inmediata sin necesidad de

tener que volver a lanzar el comando para que vuelva a compilar la aplicación y la vuelva a ejecutar.

2. Poder depurar el código sin que este esté compilado.



En el momento de escribir este documento, la opción *-livereload* se encuentra en fase *beta*, por lo que puede no funcionar correctamente. De ser así, nos veremos obligados a usar el método manual y recompilar la aplicación cada vez que efectuemos un cambio.

Una vez que la aplicación se esté ejecutando, iniciamos el *Chrome* y abrimos la consola para desarrolladores (desde las opciones de Chrome >Más herramientas >Herramientas de desarrolladores, o pulsando F12, o pulsando ctrl + shift + I). Dentro de la consola, en opciones >*More tools* seleccionamos *Remote Devices*.

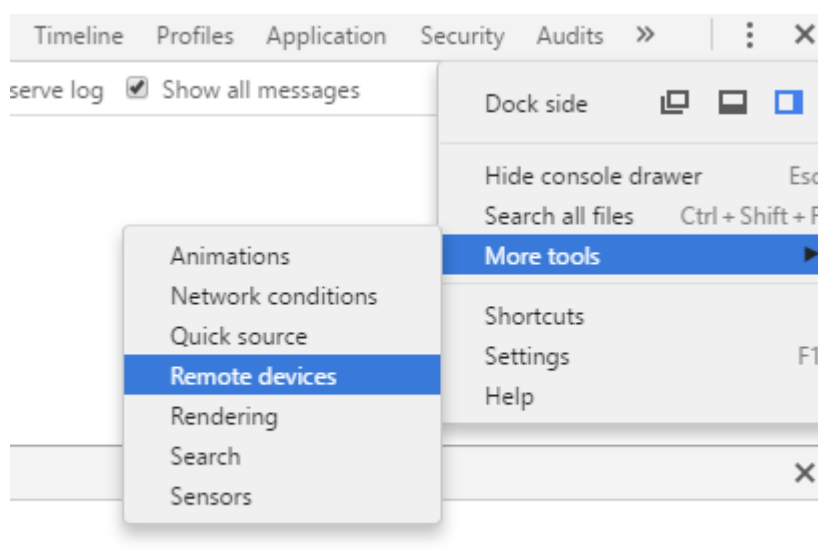


Figura 3.9: Opción para abrir la ventana de dispositivos remotos.

Esto nos abrirá una ventana dentro de la consola donde aparecerán todos los dispositivos que tengamos conectados (emuladores, terminales conectados al PC, incluso los Chromecast si disponemos de alguno en nuestra red). En esta lista de dispositivos encontraremos nuestro emulador, en él, aparecerá nuestra aplicación, la cual podremos *Inspeccionar*.

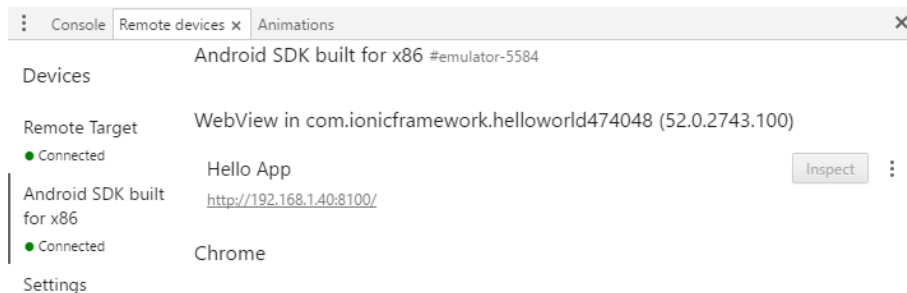


Figura 3.10: Lista de dispositivos disponibles para depurar. Entre ellos se encuentra nuestro emulador con nuestra aplicación híbrida abierta.

Si seleccionamos la opción inspeccionar, se nos abrirá una ventana del navegador en la que veremos lo mismo que se está viendo en esos momentos en el emulador. Podremos ver el detalle de los elementos del **HTML** de la misma manera que si fuera una página web corriente abierta directamente en el navegador. Además, podremos controlar la aplicación desde el navegador, así, cualquier acción que llevemos a cabo, como pulsar algún elemento, escribir, arrastar con el botón del ratón pulsado, ...se verá reflejado también en el navegador.

Podremos acceder a los ficheros `.ts` y poner puntos de ruptura donde nos interese detener la ejecución del código. Como se comentaba anteriormente, esta posibilidad solo se da con la opción `-livereload`. De no ser así, tendremos que buscar el trozo de código que nos interesa en el fichero ya compilado.

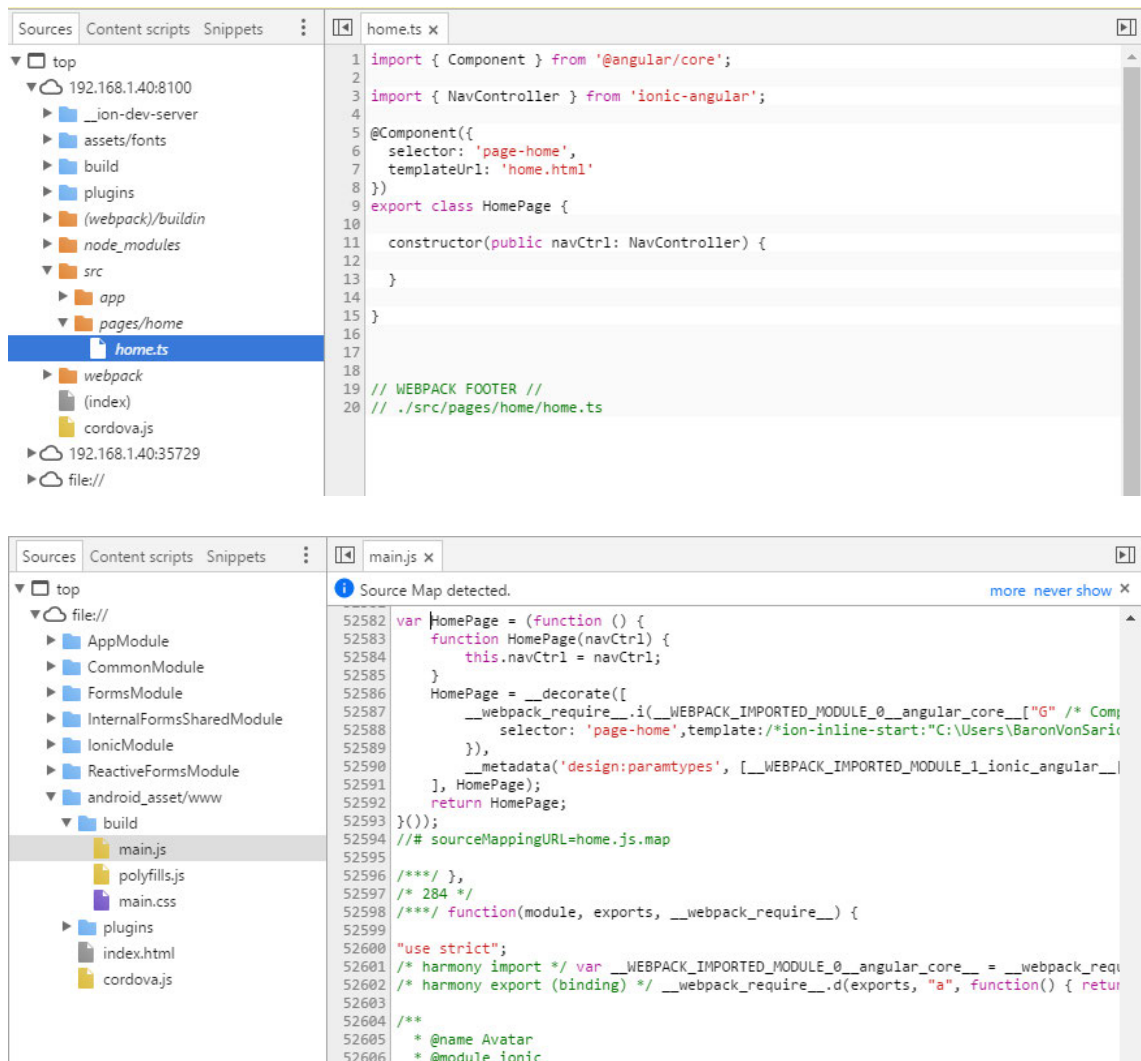


Figura 3.11: Diferencia entre como se ve el código sin compilar, emulado con la opción `-livereload` (arriba), y el código compilado (abajo).

Podemos probar a cambiar el código y ver como la aplicación se recarga nada más guardamos el cambio. Por ejemplo, vamos a hacer que se escriba una entrada en el log al iniciar la aplicación. Abrimos el fichero `home.ts` de nuestra página y editamos el constructor de la siguiente manera:

```

constructor(public navCtrl: NavController) {
  console.log("HelloWorld application is running.");
}

```

Una vez que la aplicación haya terminado de recargar, apenas tarda unos segundos, podremos comprobar que el cambio en efecto se ha efectuado y ver en la consola el mensaje anterior.

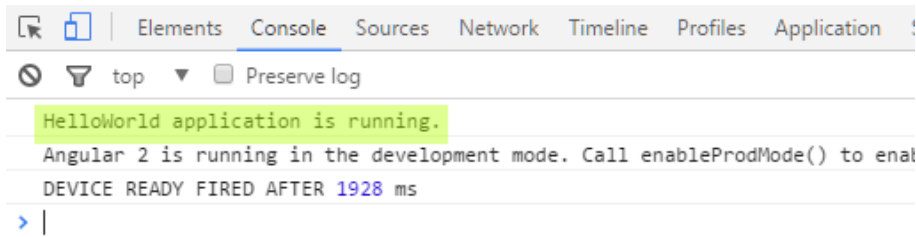


Figura 3.12: La aplicación muestra por consola el mensaje de inicio.

También podremos hacer el cambio sobre el script que se está ejecutando sobre el mismo navegador y guardar el cambio utilizando la combinación `ctrl + s`. Este cambio solo será temporal y hasta que se vuelva a recargar el fichero JavaScript, pero será válido siempre que el flujo de ejecución pase por él. Tampoco modifica el código sobre el que estamos trabajando, ya que el cambio se guarda en la versión del script que utiliza el navegador.

3.2.4. Ejecutar una aplicación en un terminal conectado

Si disponemos de uno o varios terminales en los que poder probar la aplicación, podemos conectarlos a nuestro ordenador y hacer que ejecute el código del mismo modo que ocurre en el emulador teniendo también la opción de depuración desde Chrome.

En primer lugar, debemos tener el terminal conectado al ordenador, y asegurarnos de tener todos los drivers instalados. Además, debemos de habilitar el modo *debug* vía **Universal Serial Bus (USB)** en el móvil y autenticar nuestro PC en el móvil. En la web de desarrolladores de Google nos explican como poder hacer esto <https://developers.google.com/web/tools/chrome-devtools/remote-debugging/>.

Cuando tengamos nuestro terminal listo, llegará el momento de ver nuestra aplicación en él. Como podremos imaginar, esto se consigue utilizando Ionic CLI:

```
# ionic run android --livereload
```

Y listo. Si todo va bien, al cabo de unos segundos debería aparecer nuestra aplicación en el terminal.

Al igual que ocurría con el emulador, podremos depurar nuestra aplicación utilizando el navegador Chrome, siguiendo los mismos pasos explicados anteriormente, pero seleccionando nuestro terminal en la lista de dispositivos.

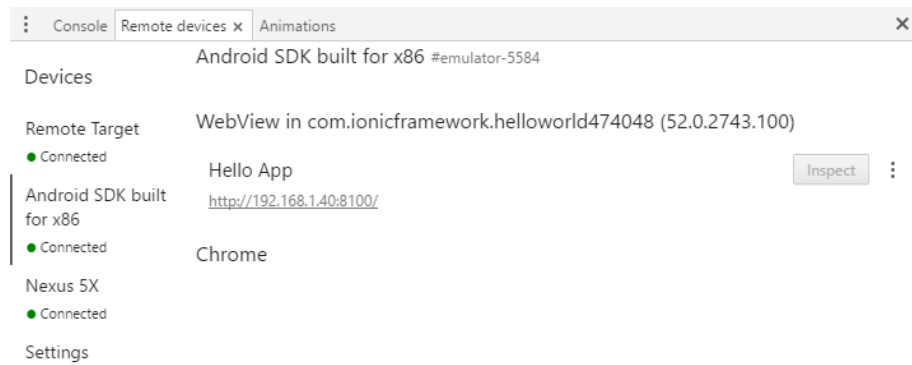


Figura 3.13: El terminal Android conectado a nuestro equipo aparece listado entre los dispositivos disponibles en Chrome.

Esta operación nos dejaría la aplicación instalada en nuestro terminal, por lo que al desconectar el dispositivo del ordenador podríamos seguir usandola como si de otra aplicación se tratase (algo útil si necesitamos hacer pruebas con el **GPS**).

3.2.5. Generar el instalador. El APK.

Cuando ya demos por buena nuestra aplicación, o simplemente queramos compartirla con alguien sin tener que compartir el código y haciendo que sea él quien lo compile, será el momento de generar el instalador para las diferentes plataformas. Como estamos trabajando para la plataforma Android, en nuestro caso generaremos un fichero *.apk*. Este fichero podremos compartirlo con cualquier usuario de Android y podrá instalarlo en su dispositivo. También podrá ser subido a la tienda de aplicaciones de Android, *Play Store*, para que pueda ser descargado por cualquier usuario. Para esto último, será necesario firmar el **APKs**.

Antes de esto, vamos a darle un último toque de personalización a nuestra aplicación. Si ya hemos ejecutado la aplicación en el terminal como se explicaba en el apartado anterior, nos habremos fijado en dos detalles. Por un lado, el icono que aparece en nuestro dispositivo representando a nuestra aplicación es el icono de Apache Cordova. Con el splash, la pantalla que aparece al iniciar la aplicación, ocurre lo mismo. Ambos son generados y configurados por defecto al añadir una nueva plataforma al proyecto.

Para poder personalizar tanto el icono como el splash, necesitamos de una imagen para cada uno de ellos en formato *.png*, *.psd* o *.ai*. Ambas tendrán que tener el elemento principal de la imagen debe estar centrado, ya que Ionic recortará la imagen para ajustarla a diferentes tamaños de pantalla. En cuanto al tamaño, el icono tendrá que tener un tamaño mínimo de 192x192 píxeles y no tendrá que tener los bordes

redondeados ya que este efecto lo aplica la propia plataforma. El splash, deberá medir un mínimo de 2208x2208 píxeles.

Una vez creadas las imágenes, tendremos que colocarlas dentro de nuestro proyecto en la capeta *resources* con los nombres *icon.png* y *splash.png* (o con la extensión del formato que hayamos elegido). Ejecutamos de nuevo un comando proporcionado por Ionic CLI el cual generará, a partir de nuestras imágenes, diferentes iconos y splash para los diferentes dispositivos y los dejará preparados para cuando se genere el instalador.

```
# ionic resources
```



El comando *ionic resources* dispone de dos opciones, *-icon* y *-splash*, por si solo queremos generar uno de los dos elementos.

Para más información, podemos consultar la documentación que proporciona Ionic en su primera versión, pero que también es aplicable para la versión 2 en <http://ionicframework.com/docs/cli/icon-splashscreen.html>.

Ya con todo listo, podemos generar nuestro APK. Ejecutamos el siguiente comando:

```
# ionic build android
```

Al finalizar, aparecerá la ruta donde se encuentra el APK. Lo renombramos con un nombre que nos resulte identificable y ya podemos probar a instalarlo en cualquier dispositivo Android.



En caso de tener ya la aplicación instalada por otro método, por ejemplo el explicado en el punto **Ejecutar una aplicación en un terminal conectado**, es posible que no podamos instalar la aplicación y recibamos un error a cambio. En este caso, es mejor desinstalar la aplicación y probar de nuevo. Lo mismo puede ocurrir si intentamos la misma operación pero en orden inverso.

Esto ocurre debido a que el nombre de la aplicación que usa Android internamente para identificarla es el mismo, pero la fuente de instalación son diferentes, por lo que pueden entrar en conflicto.

3.3. Cronómetro

Con el *Hello World!!!* hemos visto como crear nuestro proyecto y una de las características más potentes de Angular, el *Data Binding*. Ahora vamos a dar un pasito más y vamos a crear una aplicación que sirva como cronómetro. Para ello, además de usar el *Data Binding*, que por otro lado va a estar presente en la mayoría de desarrollos, veremos como modificar datos desde la lógica del componente y que estos cambios se vean reflejados en la vista. Además introduciremos algunas directivas de Angular, como es ***ngFor**, que utilizaremos para recorrer listas que formen parte del componente desde la propia plantilla

El primer paso, como no podría ser de otra manera, será crear un nuevo proyecto en blanco dentro de nuestro directorio de trabajo, de la misma forma que vimos en el capítulo anterior sobre el *Hola Mundo!!!*:

```
# ionic start Chronometer blank --v2
```

En primer lugar vamos a hacer un cronómetro que tenga como unidad de medida mínima el segundo. También añadiremos dos botones, uno que actuará como Start/Pause para iniciar y detener el cronómetro, y otro que actuará como Stop, y que pondrá a cero de nuevo el cronómetro.

En la página web oficial de Ionic podemos encontrar una sección dedicada a los componentes¹ que nos ofrece el framework. Aquí podremos encontrar el componente que se utiliza para poder crear un botón, y las opciones que permite, como por ejemplo las dimensiones. También podremos encontrar una sección, llamada *Ionicons*², en la que encontraremos iconos que ofrece el propio framework. Lo interesante de estos iconos es que se adaptan según la plataforma es la que se ejecuta la aplicación, haciendo que el *Look and Feel* de esta sea coherente con la plataforma en cuestión.

¹<https://ionicframework.com/docs/v2/components>

²<https://ionicframework.com/docs/v2/ionicons/>










| Name | iOS | iOS-Outline | Material Design |
|------------|---|---|---|
| add |  |  |  |
| add-circle |  |  |  |
| alarm |  |  |  |

Figura 3.14: Una pequeña parte de los iconos que ofrece Ionicons. Podemos ver la diferencia que existen entre ambas plataformas.

Empezaremos editando el fichero *home.html* y creando un cronómetro con los elementos que queramos que lo compongan. De momento usaremos valores fijos, sin añadir ningún tipo de lógica, pudiendo así ver como queda el diseño escogido. Aquí cada cual puede utilizar el diseño que quiera, en mi caso, he optado por uno sencillo:

```
<ion-content padding>
  <p style="text-align: center; font-size: -webkit-xxx-large;" >
    00:00:00
  </p>

  <ion-row>
    <ion-col>
      <button ion-button color="secondary" full large >
        <ion-icon name="play" ></ion-icon>
        Start
      </button>
    </ion-col>
  </ion-row>

  <ion-row>
    <ion-col>
      <button ion-button color="danger" full large >
        <ion-icon name="square" ></ion-icon>
        Stop
      </button>
    </ion-col>
  </ion-row>
</ion-content>
```

```
</ion-row>  
</ion-content>
```

Con este diseño, nuestra página se vera tal que así:

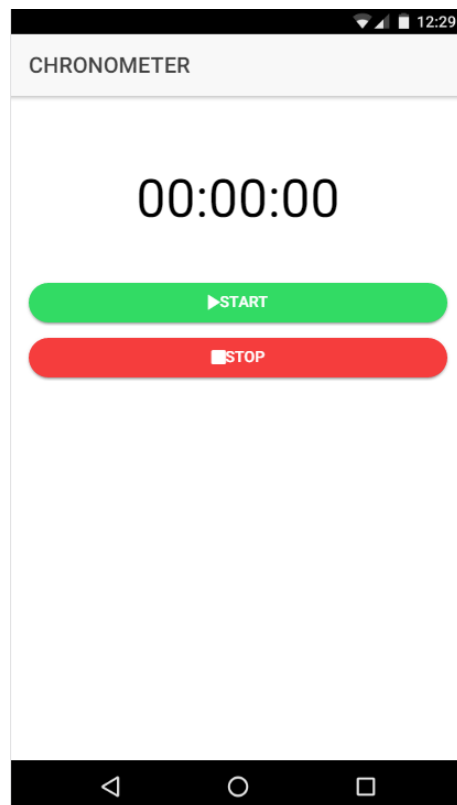



Figura 3.15: Esto sí que es un cronómetro sencillo.



El navegador Chrome nos da la opción de simular la vista de un dispositivo móvil y poder así comprobar como se ve nuestra aplicación limitada a diferentes tamaños de pantalla. Para ello, tendremos que abrir la herramienta para desarrolladores pulsando F12, y activamos *Device Toolbar* con el botón  o pulsando Ctrl+Shift+M. Esto nos abrirá una barra superior en la que podremos elegir entre diferentes modelos de dispositivos o definir el tamaño de la pantalla en píxeles manualmente. Aunque por lo general suele ser un comportamiento cercano a la realidad, en ocasiones el resultado visto aquí y el que después vemos en la realidad puede variar en lo referente a dimensiones y posiciones de algunos elementos.

Siguiente paso, añadir lógica a nuestra aplicación, para ello vamos a editar el

fichero *home.component.ts*. En este fichero *TypeScript* encontramos la clase *HomePage*, la cual esta precedida por el decorador *@Component* de Angular. Esto quiere decir que esta clase se trata de un componente de Angular. Es este decorador se definirán los metadatos del componente, en este caso, se han definido dos de ellos: el *template* y el *selector*, los cuales indican el código o fichero **HTML** que define el aspecto visual del componente en el caso del *template* y el nombre del tag **HTML** que representa este controlador en el caso del *selector*. Gracias al *Data Binding*, las propiedades y métodos que definamos dentro de este controlador, serán accesibles desde el código **HTML** si es necesario. En primer lugar, vamos a pensar que variables precisamos para la implementación del cronómetro:

- Un *flag* que nos indique si el cronómetro está o no en marcha.
- Un contador para que almacene los segundos desde la puesta en marcha.
- Para que el botón Start/Pause cambie según el estado, necesitaremos hacerlo usando dos variables:
 - Una cadena de texto que almacene la el texto que aparece en el botón: *Start* o *Pause*
 - Una cadena que indique que icono usar en el botón.

Todas estas variables serán inicializadas en el constructor. Para cumplir con esto requisitos utilizamos el siguiente código:

```
_is_running:boolean;  
seconds:number;  
start_pause_label: string;  
start_pause_icon: string;  
  
constructor() {  
  this.is_running = false;  
  this.seconds = 0;  
};
```



Es una práctica recomendable que los nombres de las propiedades, de las funciones, de las clases, ...sean autodescriptivos, ya estemos trabajando con TypeScript como con cualquier otro lenguaje.

No hemos inicializado ni *start_pause_label* ni *start_pause_icon* por la siguiente razón, estas variables están ligadas al estado en que se encuentra el cronómetro, es decir a *_is_running*. Para asegurarnos que estos valores cambian al cambiar

`_is_running`, vamos a personalizar el *getter* y el *setter*³ de esta propiedad, esta es la razón por la que se ha prefijado el nombre de la propiedad con una barra baja (`_`) evitando así el conflicto que se produciría con los nombres. Los métodos quedarían de la siguiente manera:

```
public get is_running() {
    return this._is_running;
};

public set is_running(new_state) {
    this._is_running = new_state;
    this.start_pause_label = this.is_running ? "Pause" : "Start";
    this.start_pause_icon = this.is_running ? "pause" : "play";
};
```

Cada vez que se haga una consulta a la propiedad `is_running`, se ejecutará la primera función, y cuando se asigne un valor (como ocurre en el constructor al ejecutar `this.is_running = false;`), se ejecutará la segunda función, la cual asigna el nuevo valor a `is_running`, y según sea este, a `start_pause_label` y `start_pause_icon`.

Llega el momento de implementar los métodos, para lo cual vamos a realizar un ejercicio anterior a la codificación similar al que hicimos con las propiedades. Los métodos que considero necesarios son para conseguir hacer funcionar nuestro cronómetro son:

- Un método que sirva para cambiar el estado del cronómetro y que funcione si fuera un interruptor.
- Otro que sirva para detener el cronómetro y ponerlo a 0 el contador.
- Un último método que aumente en 1 los segundos si cronómetro está en marcha.

Al implementar estos métodos, nos quedará el componente de la siguiente manera:

```
export class HomePage {
    private _is_running:boolean;
    seconds:number;
    start_pause_label: string;
    start_pause_icon: string;

    constructor() {
        this.is_running = false;
        this.seconds =0;

        setInterval(() => {
```

³<https://www.typescriptlang.org/docs/handbook/classes.html#accessors>

```

        this.tick();
    }, 1000);
};

public get is_running() {
    return this._is_running;
};

public set is_running(new_state) {
    this._is_running = new_state;
    this.start_pause_label = this.is_running ? "Pause" : "Start";
    this.start_pause_icon = this.is_running ? "pause" : "play";
};

tick(): void {
    if (this.is_running) {
        this.seconds++;
    }
};

toggle(): void {
    this.is_running = !this.is_running;
};

stop(): void {
    this.is_running = false;
    this.seconds = 0;
};
}

```

Para que la función *tick()* tenga sentido, es necesario que se ejecute cada segundo. Vamos a usar la función *setInterval()*. Esta función es nativa de JavaScript y por ello podemos hacer uso de ella desde TypeScript. La invocamos en el constructor indicando el método al que tiene que llamar y la espera entre una llamada y otra (el tiempo que se le pasa debe estar expresado en milisegundos):

```

constructor() {
    // RESTO DE FUNCIÓN
    setInterval(() => {
        this.tick();
    }, 1000);
};

```

Otra opción sería llamar a la función *setInterval()* en el *setter* de *is_running* y asignar el id devuelto a una variable del controlador, y así poder desactivarlo con la función *clearInterval()*, la cual acepta como parámetro el ID anterior. Algo como:

```

public set is_running(new_state) {
    this._is_running = new_state;
    if (this.is_running) {
        this.start_pause_label = "Pause"
        this.start_pause_icon = "pause"
        this.interval_id = setInterval(() => {

```



```

        this.tick();
    }, 1000);
} else {
    this.start_pause_label = "Start"
    this.start_pause_icon = "play"
    clearInterval(this.interval_id);
}
};

```

La ventaja de esta opción es que se evita la llamada a la función *tick()* cuando el cronómetro está detenido.

Con la lógica ya implementada, vamos a conectar las propiedades del componente con los elementos de la template, y para ello nos ayudaremos del famoso *Data Binding*. Como nuestra template esta asignada al componente definido por la clase **HomePage**, podremos hacer uso de sus propiedades y métodos. Para hacer funcionar el cronómetro, necesitaremos:

1. Por un lado modificar los valores mostrados en el **HTML** para que coincidan con los definidos en el componente. Esto se hace poniendo el nombre de la variable o del método entre llaves dobles (`{{ NOMBRE }}`) en el **HTML** allí donde queramos que el valor aparezca.
2. Vincular el evento de *click* sobre los botones a alguno de los métodos. Esto se consigue gracias a la directiva (**clic**) que se pone como atributo **HTML** en el botón.

El código se vería de la siguiente manera:

```

<p style="text-align: center; font-size: -webkit-xxx-large;" >
    {{ seconds }}
</p>

<ion-row>
    <ion-col>
        <button ion-button color="secondary" full large (click)="toggle
            ()" >
            <ion-icon name="{{ start_pause_icon }}" ></ion-icon>
            {{ start_pause_label }}
        </button>
    </ion-col>
</ion-row>

<ion-row>
    <ion-col>
        <button ion-button color="danger" (click)="stop()" style="width:
            100%;" >
            <ion-icon name="square" ></ion-icon>
            Stop
        </button>
    </ion-col>
</ion-row>

```

```
</ion-col>
</ion-row>
```

Vemos que hemos cambiado el display donde se visualizan los segundos, el icono y el texto del boton “Start”/“Pause”, se han añadido las directivas (*clic*) para que hagan llamadas a las funciones que implementamos en nuestra clase *HomePage* ...pero vemos que la forma de visualizar el tiempo transcurrido no es la que teníamos en mente, si no que vemos el número que utilizamos para contar los segundos desde el inicio, lo cuál no es intuitivo a la hora de leerlo.

Necesitamos por tanto, darle un formato más legible. Para esto vamos a usar otra herramienta que nos ofrece Angular, las llamadas *Pipes* o *tuberías*.

El concepto de *Pipe* es similar al que se utiliza por ejemplo en *bash* o *sh*, y nos permite transformar un dato, el cual se le pasa como entrada a la tubería, y obtenemos el mismo dato ya transformado. Para nuestra aplicación necesitamos que nuestros segundos, que no es más que un número que representa la cantidad de segundos desde el inicio a nivel lógico, se convierta en una cadena de texto con un formato más amigable para el usuario, *0h 00m 00s*. Angular dispone de algunos *Pipes* ya definidos⁴, pero en nuestro caso, definiremos uno propio.

Empezaremos creando un nuevo fichero *.ts* al que llamaremos *home.pipes.ts*. Con esto conseguimos que el código quede MAS ORDENADO. Dentro de este fichero crearemos nuestra clase, la cual deberá implementar la interfaz **PipeTransform** y ser decorada con *@Pipe*. Al implementar la interfaz **PipeTransform** tendremos que definir el método **transform()**. Este será el método que se ejecute cuando nuestra tubería sea usada, pasando la entrada de esta como parámetro. El decorador *@Pipe* acepta el argumento **name**, el cuál nos permite asignar un identificador a nuestra tubería.

```
@Pipe({name: 'time_format'})
export class MyTimePipe implements PipeTransform {
  transform(seconds: number): string {
    var s_num = seconds % 60;
    var m_num = Math.floor((seconds % 3600) / 60);
    var h_num = Math.floor(seconds / 3600);

    var s = ('0' + s_num).slice(-2);
    var m = ('0' + m_num).slice(-2);
    var h = '' + h_num;

    return `${h}h ${m}m ${s}s`;
  }
}
```

⁴<https://angular.io/docs/ts/latest/api/#!query=pipe>

Ya solo nos queda utilizar esta tubería en nuestro template. Cambiamos la llamada a la variable *seconds* para que pase por nuestra tubería:

```
<p style="text-align: center; font-size: -webkit-xxx-large;" >
  {{ seconds | time_format }}
</p>
```

Al utilizar esta tubería, conseguimos que el contador de segundos siga siendo un número, lo que facilita su uso en otras partes del código. También tenemos disponible un método sencillo y reusable para convertir segundos a un formato entendible para el usuario y que volveremos a usar en el siguiente apartado.

3.3.1. Cronómetro con contador de vuelta

Vamos a añadir una función extra a nuestro cronómetro, un contador de vueltas. Este contador nos permitirá guardar el registro en un momento dado sin detener el cronómetro. Estos registros que se vayan guardando aparecerán en una lista debajo de la botonera, la cuál también modificaremos para añadir dos nuevos botones.



Figura 3.16: En esta nueva versión hemos añadido dos nuevos botones además de una lista con los tiempos guardados.

Veamos a continuación las modificaciones hechas a nuestro **HTML** para introducir los nuevos botones y la lista.

```

<ion-content padding>
<p style="text-align: center; font-size: -webkit-xxx-large;" >
  {{ seconds | time_format }}
</p>

<ion-row>
  <ion-col>
    <button ion-button color="secondary" full large (click)="toggle
      ()" >
      <ion-icon name="{{ start_pause_icon }}" ></ion-icon>
      {{ start_pause_label }}
    </button>
  </ion-col>
</ion-row>

<ion-row>
  <ion-col>
    <button ion-button color="danger" (click)="stop()" style="width:
      100%;" >
      <ion-icon name="square" ></ion-icon>
      Stop
    </button>
  </ion-col>
  <ion-col>
    <button ion-button (click)="step()" style="width: 100%;" >
      <ion-icon name="flag" ></ion-icon>
      Step
    </button>
  </ion-col>
  <ion-col>
    <button ion-button (click)="clear()" style="width: 100%;" >
      <ion-icon name="trash" ></ion-icon>
      Clear
    </button>
  </ion-col>
</ion-row>
<ion-list>
  <ion-item *ngFor="let step of step_list; let i=index" >
    <h1>{{ i }} : {{ step | time_format }}</h1>
  </ion-item>
</ion-list>
</ion-content>

```

Los botones no tienen más misterio que los que se explicaron antes, cambiando la función a la que llaman, las cuales implementaremos más adelante.

Sí que nos encontramos dos componente nuevos de Ionic, **ion-list** y **ion-item**. Como bien se puede intuir por sus nombre, estos componentes sirven para crear listas y elementos dentro de ellas. Dentro del componente ion-item también vemos el atributo ***ngFor**. Este atributo se trata de una directiva de Angular que nos permite recorrer una lista, que definiremos en el componente, y por cada iteración renderizará el elemento que contiene la directiva, en este caso, el ion-item. En cada

iteración sobre la lista, el valor recuperado se almacena en una variable (let step of step_list) a la que se puede acceder del mismo modo que si se tratara de una propiedad del componente, pudiendo tratarla con nuestra tubería personalizada de igual manera que hacíamos con los segundos. Como opción, podemos recuperar el índice, que representa la posición que ocupa el elemento en la lista, y asignarlo a una variable (let i=index) que como no, también podemos usar como la anterior.

En lo que respecta a la lógica, necesitaremos implementar las dos funciones a las que se llamarán desde los nuevos botones, y una lista que almacene los registros que vayamos guardando. Desde una de las funciones haremos que se añada a la lista el tiempo actual, desde la otra limpiaremos la lista. Como ocurre con el resto de propiedades, cada vez que modifiquemos la lista, se actualizará la vista de la misma manera que pasaba con los segundos, sin importar que la variable se encuentre dentro de la directiva *ngFor.

```
export class HomePage {  
  // RESTO DE VARIABLES  
  step_list: number[];  
  
  // RESTO DE FUNCIONES  
  step(): void {  
    if (this.is_running) {  
      this.step_list.push(this.seconds);  
    }  
  }  
  
  clear(): void {  
    this.step_list = [];  
  }  
}
```

Estas nuevas funciones serán ejecutadas cada vez que pulsemos sobre los correspondientes botones ya que como vimos antes, han sido asignados utilizando la directiva (**click**).

3.4. Paisaje

En esta práctica vamos a centrarnos en el uso del **CSS** y de las animaciones. Antes de nada, haremos una pequeña introducción a qué es y como funciona **CSS** y como podemos trabajar con el en Ionic2.

CSS es un lenguaje utilizado para definir la presentación de un documento estructurado escrito en un lenguaje de marcado. Esto, cuando trabajamos con tecnología Web, se refiere al estilo a aplicar nuestra página, la cual está escrita en **HTML**, que actúa como lenguaje de marcado. Así, mientras que nuestro código **HTML** contiene la información, el código **CSS** define la presentación. Con esta separación entre información y presentación se consigue un mayor control y modularidad.

La última especificación de **CSS** es la número 3, la que se conoce como **Cascading Style Sheets v3 (CSS3)**.

Una de las características mas útiles y que más han evolucionado son las animaciones. Muy útiles a la hora de realizar páginas más vistosas. Con ellas, podemos definir transiciones entre estilos definidos en el **CSS**. Estas animaciones constan de dos elementos:

1. Uno que describe la animación que se produce en la transición.
2. Otro que define los frames del estado inicial y final de la animación. Incluso se puede llegar a definir estados intermedios.

Para facilitar el trabajo con **CSS**, Ionic2 incluye **Sass**. **Sass** es un lenguaje que amplía las capacidades de **CSS**. Los navegadores no entienden el lenguaje **Sass**, por lo que es necesario traducirlo a **CSS**, de forma similar a lo que ocurre con TypeScript y JavaScript.

Sass utiliza dos tipos de sintaxis, siendo validas cualquiera de las dos:

1. La sintaxis original, en la que se utiliza la indentación para separar los distintos bloques. La extensión de los ficheros escritos con esta sintaxis es **.Sass**.
2. Una sintaxis más parecida a la usada en **CSS**, con el uso de las llaves **()** para separar los bloques y del punto y coma **(;)** al final de cada línea. En este caso la extensión de los ficheros es **.scss**.



Si nos fijamos al crear un nuevo proyecto o una nueva página, las hojas de estilo que aparecen son nombradas como *.scss y no como *.css. Ionic, a la hora de compilar la aplicación, es lo suficientemente autónomo como para utilizar este fichero y los estilos en él definido en el ámbito de nuestro componente, sin necesidad de indicarlo manualmente.

Alguna de las capacidades que incluye **Sass** y que encontraremos de gran utilidad son:

1. La posibilidad de definir variables.
2. El uso de mixins⁵, gracias a los cuales nos evitaremos el tener que duplicar código.
3. Anidar los estilos en bloques.
4. Poder hacer uso de directivas de control y expresiones como *if*, *for*, *each*,
5. Herencias de estilos como ocurre en lenguajes orientados a objetos.
6. ...

Para más información sobre **Sass**, podemos visitar su página de documentación http://Sass-lang.com/documentation/file.Sass_REFERENCE.html.

Por último, y antes de entrar en faena, debemos mencionar el control de animaciones que nos ofrece Angular (que recordemos, es la base sobre la que funciona Ionic2). Las animaciones en Angular funcionan de la misma forma que las animaciones **CSS**, facilitándonos el uso de estas en determinados escenarios. Las animaciones de Angular se definen como parte de los metadatos del *@component* en los que se van a usar. En la definición de una animación se definen tres elementos principales que aunque veremos con más profundidad más adelante, durante el desarrollo, vamos a enumerar a continuación:

1. El *Animation trigger* o *trigger* a secas. Es el disparador que se vincula al elemento **HTML**. Dentro de él se define toda la animación. Un componente puede tener asignado cualquier cantidad de *trigger*.
2. Los *states*, o estados, son aquellos en los cuales puede estar el *trigger* y que definen el estilo que se tiene que aplicar al elemento cuando se encuentra en ese estado.

⁵<http://sass-lang.com/guide#topic-6>

3. Y las *transitions* o transiciones las cuales definen la animación que se produce cuando hay un cambio de estados.

Para poder ver en acción las animaciones, vamos a crear un paisaje con elementos animados. El resultado será algo como lo siguiente:



Figura 3.17: Nuestra aplicación nos mostrará un paisaje, en el que podremos cambiar entre noche y día.

3.4.1. Primeros pasos. El cielo.

Empezaremos como siempre, creando un nuevo proyecto dentro de nuestro directorio de trabajo utilizando el Ionic CLI. En mi caso, he decidido llamarle Landscape:

```
# ionic start Landscape blank --v2
```


Empezaremos editando los ficheros *home.html* y *home.scss* (podemos renombrar la página con un nombre más descriptivo si queremos, lo cual sería recomendable en caso de tener más páginas, pero en este caso, no será necesario). Eliminaremos todo el contenido de del fichero HTML y añadiremos un *div* que actuara como “lienzo” para nuestro paisaje. A este elemento le vamos a asignar las clases *canvas* y *sky*, haciéndolo más reconocible.

```
<div class="canvas sky" ></div>
```

En cuanto al fichero *.scss*, añadiremos los estilos para nuestro nuevo elemento. Estos estilos los anidaremos dentro del elemento *page-home*. El elemento *page-home* es el que representa a nuestro componente (la opción *selector* del metadata de un *@component* es la que indica el nombre de este selector).

```
page-home {  
  .canvas.sky {  
    height: 100%;  
    width: 100%;  
    background-color: #00BFFF;  
  }  
}
```

Aquí vemos una de las capacidades comentadas de *Sass*. Hemos anidado los estilos que aplican a las clases *.canvas.sky* dentro de *page-home*. Para ver la diferencia con *CSS*, podemos ver el código que esto genera al ser compilado:

```
page-home .canvas.sky {  
  height: 100%;  
  width: 100%;  
  background-color: #00BFFF;  
}
```

Resulta más intuitivo definir los estilos anidando las reglas de igual forma que está definidos los elementos dentro del **DOM** de la página, que la forma que se utiliza en **CSS**, en la que la relación padre e hijos se representa separando los identificadores con un espacio.

Si ejecutamos en estos momentos la aplicación, no veremos más que una pantalla azul, la cual hará la función de cielo en nuestro paisaje. Será sobre este cielo donde iremos añadiendo elementos.

3.4.2. Animaciones definidas sobre el componente. El sol y la luna.

Empezaremos por incluir un sol, y al igual que hicimos con el canvas, este será representado por un componente **HTML** al que le añadiremos como fondo la imagen

de un sol. Esta imagen tendremos que ponerla dentro de la carpeta `/src/assets` para que, una vez se genere la aplicación, esta sea incluida. Yo he creado una carpeta intermedia llamada *landscape* para tener las imágenes mas ordenadas. Así pues, añadimos el sol dentro de nuestro “lienzo” y lo estilamos dentro de nuestro fichero *scss*:

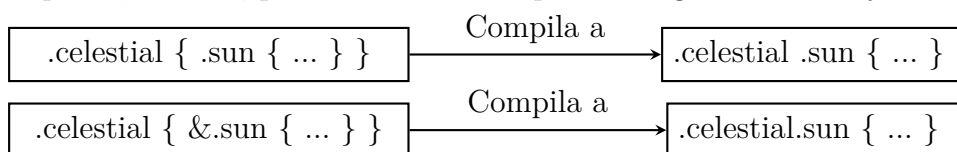
home.html

```
<div class="canvas sky" >
  <div class="celestial sun" >
    </div>
  </div>
```

home.scss

```
page-home {
  .canvas.sky{ ... }
  .celestial{
    position: fixed;
    height: 100px;
    width: 100px;
    left: 40%;
    top: 10%;
    background-size: contain;
    background-repeat:
      no-repeat;
    &.sun{
      background-image: url(.. /
        assets/landscape /
        sun.png);
    }
  }
}
```

Si nos fijamos, he asignado dos clases al elemento que contendrá mi sol, la clase *celestial* y la clase *sun*. A estas clases las he dado estilo anidando la clase *sun* dentro de la clase *celestial*, pero con un elemento diferente a la de otras anidaciones. En el caso de la clase *sun*, le he antepuesto el carácter `&`. ¿Y qué significa esto?. A la hora de generar el fichero *CSS*, el compilador sustituirá este carácter por el selector del estilo padre, es decir, por *.celestial*. Para que nos hagamos una mejor idea:



Con esto conseguimos tener nuestro sol en medio del cielo azul, pero uno de los comportamientos que buscamos es que nuestro paisaje tenga una versión nocturna además de una diurna. Para hacer esto, vamos a definir un estado dentro de nuestro *@component* que defina si es de día, o de noche, y haremos que este estado cambie al pulsar sobre nuestro sol. Para esto solo será necesario vincular el evento de *click* sobre el sol a una función que haga cambiar la variable de estado de nuestro componente:

```
<div class="canvas sky" >
  <div class="celestial sun" (
    click)="toggleState()" >
  </div>
</div>
```

```
export class HomePage {
  state: string = 'day';

  // otras funciones

  toggleState() {
    this.state = 'day' ==
      this.state ? 'night' : '
      day';
  }
}
```

Ya tenemos una variable que almacena el estado y que podemos cambiar pulsando nuestro sol, pero no vemos como esto se refleja en nuestro paisaje (a no ser que depuremos la aplicación o hagamos que imprima el valor por consola). Vamos por fin a hacer uso de las animaciones. La primera va a ser muy sencillita, cambiar el color del cielo a uno mas nocturno. En primer lugar vamos a crear una animación de Angular que haga esto y la añadiremos a nuestro componente. Para hacer esto modificamos el metadata de nuestro componente de la siguiente manera:

```
@Component({
  selector: 'page-home',
  templateUrl: 'home.html',
  animations: [
    trigger('skyState', [
      state('day', style({
        backgroundColor: '#00BFFF'
      })),
      state('night', style({
        backgroundColor: '#131862'
      })),
      transition('day <=> night', animate('1s')),
    ]),
  ],
})
```

No nos olvidemos de importar las nuevas clases que vamos a utilizar: <https://angular.io/docs/ts/latest/api/core/index/trigger-function.html>

```
import { Component, ElementRef, ViewChild, AfterViewInit, trigger,
  state, style, transition, animate } from '@angular/core';
```

Veamos que es lo hemos definido:

1. Hemos definido el metadata *animations* de nuestro componente como una lista en la que de momento solo definiremos un *trigger*⁶.
2. Este *trigger*, que es una llamada al constructor de la clase *trigger*, lo hemos nombrado *skyState* (primer parámetro que acepta el constructor).

⁶<https://angular.io/docs/ts/latest/api/core/index/trigger-function.html>

3. Como segundo parámetro le pasamos una lista de estados y transiciones que definen las animaciones.
4. Hemos definido dos estados mediante la clase *state*⁷:
 - a) Un estado llamado *day*, con un estilo que define el color de fondo de la misma manera que se definen las reglas en **CSS**.
 - b) Un estado llamado *night*, que también define también el color de fondo, pero cambiando el valor de este.
5. Junto a los estados hemos definido una transición. Como primer parámetro, le pasamos el nombre de los estados definidos previamente entre los que se hace el cambio, y la dirección entre ellos en las que se da (la flecha \leq indica bidireccionalidad, mientras que una flecha tipo \Rightarrow , indicaría solo un sentido). Como segundo parámetro, seleccionamos el tipo de animación que queremos mediante una clase *animate*⁸, en este caso, queremos que la animación dure un segundo en completarse ('1s').

Por último nos quedaría vincular nuestro *trigger* a nuestro cielo y al estado definido dentro del componente. Esto se hace directamente editando el elemento **HTML**.

```
<div class="canvas sky" [@skyState]="state" >
```

Que el sol aparezca en medio de un cielo nocturno no es muy común, necesitaremos hacer uso de otra animación para solucionar esto. En esta ocasión en vez de contar con un único elemento al que le cambiamos el estilo, añadiremos uno nuevo que representará la luna. Este elemento tendrá varios estilos en común con el sol definido anteriormente, por lo que compartirán clase *celestial* (ahora le vemos el sentido a la separación de estilos que hicimos anteriormente entre *celestial* y *sun*). El ver aparecer o desaparecer estos cuerpos celestiales en centro de la pantalla no es lo más bonito, así que para esta animación haremos que el cuerpo celeste aparezca por el borde superior, y desaparezca por el borde inferior según le toque. Nos vamos a ayudar de la regla *top* que define **CSS**, con la cuál podemos definir a que distancia respecto al borde superior se posiciona el elemento en cuestión (de igual manera, la regla *left* define la distancia respecto al borde izquierdo). Entre los valores que acepta, se puede indicar un tanto por ciento respecto a la altura del elemento padre y así, jugando con este valor, podremos

⁷<https://angular.io/docs/ts/latest/api/core/index/state-function.html>

⁸<https://angular.io/docs/ts/latest/api/core/index/animate-function.html>

crear la animación que queremos. Empezamos definiendo nuestro nuevo elemento y las posiciones de origen.

```
<div class="canvas sky" [
  @skyState]="state" >
  <div #sun class="celestial sun"
    (click)="toggleState()" >
    </div>
  <div #moon class="celestial
    moon" (click)="toggleState
    ()" ></div>
</div>
```

```
.celestial{
  position: fixed;
  height: 100px;
  width: 100px;
  left: 40%;
  top: 10%;
  background-size: contain;
  background-repeat: no-repeat;
  &.sun{
    background-image: url(../
      assets/landscape/sun.png)
    ;
  }
  &.moon{
    background-image: url(../
      assets/landscape/moon.png
    );
  }
}
```

Crearemos una animación que será la que usen ambos elementos, ambos cuerpos celestiales. En esta animación usaremos dos estados, pero no definiremos ninguno ¿Cómo es esto posible?. Angular ya define dos estados especiales, a saber:

1. El estado *void*, que se define un el estado de los elementos que desaparecen.
2. El estado ***, es un estado comodín para cuando no existe estado definido.

Con esto presente, crearemos una animación para los siguientes cambios de estado:

1. `void => *`: Esta transición definirá la entrada.
2. `* => void`: Por el contrario, esta definirá la salida.

```
animations: [
  trigger('skyState', [ ... ]),
  trigger('flyInOut', [
    transition('void => *', [
      style({top: '-100%'}),
      animate('1s')
    ]),
    transition('* => void', [
      animate('1s', style({top: '100%'}))
    ])
  ]),
]
```



El uso de animaciones de entrada y salida es frecuente, por ello en Angular se han asignado alias especiales a estos cambios de estado. En las transiciones, podemos cambiar la definición de cambio de estado `'void =>*'` por `'enter'` y `'* =>void'` por `'leave'`.

En la transición de entrada, al definir un estilo dentro de la transición, hacemos que antes de que se inicie esta, se le aplique el estilo definido. Así conseguimos que nuestro elemento se coloque por encima de la pantalla. Por el contrario, en la transición de salida, el estilo está definido dentro de la animación. Indicamos así el estilo al finalizar la animación. Este estilo no se guarda, si no que al terminar la animación, el estilo es borrado del elemento.

Solo nos queda asignar la animación a nuestros elementos, pero en esta ocasión no hay estados propiamente dichos, si no que hay que hacer que el elemento aparezca y desaparezca, ¿Cómo hacemos esto?. Usaremos la directriz de Angular **ngIf**. Esta directriz hace que aparezca o no el elemento asignado en escena según si el valor asignado es *true* o *false*. Usando esta directriz, y la propiedad *state* definida en el componente, nos será fácil conseguir nuestro objetivo:

```
<div class="canvas sky" [@skyState]="state" >
  <div class="celestial sun" [@flyInOut] *ngIf="state === 'day'" (
    click)="toggleState()" ></div>
  <div class="celestial moon" [@flyInOut] *ngIf="state === 'night'" (
    click)="toggleState()" ></div>
</div>
```

3.4.3. Animación mediante código JavaScript/TypeScript. El terreno.

Ya va cogiendo forma nuestro paisaje. Lo siguiente que vamos a añadir va a ser el terreno. Para que nuestro terreno sea un poco más interactivo, vamos a hacer que podamos moverlo con el dedo tanto a la izquierda, como a la derecha. Esto no podemos hacerlo usando las animaciones, ya que el desplazamiento del dedo no es siempre el mismo y las animaciones necesitan tener definido un inicio y un fin. Por tanto, debemos de tratar este movimiento desde el código JavaScript (o TypeScript como en nuestro caso). El gesto de mover un dedo sobre un elemento de nuestra página provoca que se genere un evento JavaScript, que como otro cualquiera, podemos capturar en nuestro código. Este evento nos da información sobre el recorrido que ha hecho nuestro dedo sobre la pantalla, y usando esta información, podremos variar en consecuencia la posición del elemento que

representa el terreno.

Vamos a empezar como es normal añadiendo el elemento a nuestro código **HTML** y dándole el estilo deseado.

```
<div class="canvas sky" [
  @skyState]="state" >
  <div class="celestial sun" [
    @flyInOut] *ngIf="state ===
    'day'" (click)="
    toggleState()" ></div>
  <div class="celestial moon" [
    @flyInOut] *ngIf="state ===
    'night'" (click)="
    toggleState()" ></div>
  <div #landscape class="
    landscape" ></div>
</div>
```

```
page-home { ... }
.celestial{ ... }
.landscape {
  position: fixed;
  height: 100vh;
  width: calc(100vh*1024/373);
  // El tamaño de la imagen
  // es 1024*373, así que
  // calculamos la relación
  // alto entre ancho
  background-image: url(../
  assets/landscape/
  landscape.png);
  background-size: auto 100%;
  background-repeat: no-repeat;
  pointer-events: none;
}
```

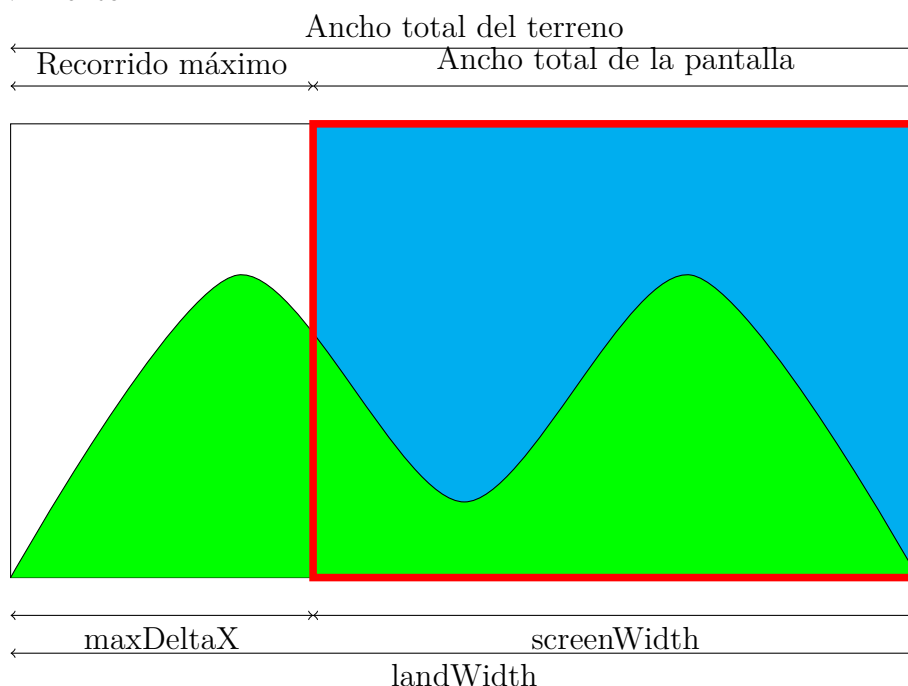
Ya que el *div* que contiene la imagen cubre nuestro *sol/luna*, aunque sea con zona transparente, interfiere en el evento del click, evitando que llegue a nuestro cuerpo celeste. Para solucionarlo, añadimos la regla *pointer-events: none*; y controlaremos el evento de pulsación con el dedo desde el *div* del elemento *canvas*.

Si nos fijamos, en nuestro nuevo *div* aparece un atributo que no habíamos usado anteriormente, *#landscape*. Al añadir un atributo prefijado con el caracter almohadilla (#), le estamos asignando un identificador gracias al cual podremos hacer uso de este elemento desde dentro del *component* y poder “jugar” con él. Ya que el tamaño de nuestro terreno se ajusta según la altura de la pantalla, necesitaremos calcular de manera programática el máximo desplazamiento horizontal para evitar que el terreno desaparezca por uno de los bordes laterales. Para hacer esto, es necesario que la vista este renderizada (y por tanto los tamaños ajustados en el momento de realizar los cálculos que veremos a continuación). Para esta tarea, Angular cuenta con un componente *AfterViewInit* el cual debemos implementar en nuestro componente, junto con el método *ngAfterViewInit* que se ejecuta al finalizar el renderizado de la página.



Angular controla un ciclo de vida para cada uno de los componentes que definimos, y nos provee a los desarrolladores de los mecanismos necesarios para poder realizar acciones en los diferentes momentos de este ciclo. En el caso que nos ocupa, nos interesa el momento una vez que la página ha sido dibujada, pero existen más. Para mas información sobre este ciclo, consultar la [API⁹](https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html) de Angular referente a ello.

El siguiente dibujo muestra nuestro terreno a modo de croquis donde podemos ver los parámetros que vamos a necesitar calcular para poder implementar el movimiento:



Estos valores pueden ser calculados y almacenados como propiedades del componente de la siguiente manera:

```
export class HomePage implements AfterViewInit {
  /* Resto de variables */
  @ViewChild('landscape') landscapeElement:ElementRef; //nuestro div
  al que identificamos con el atributo #landscape
  maxDeltaX:number;
  currentDeltaX: number;
  lastPanDeltaX: number = 0;

  ngAfterViewInit() {
    var landWidth = this.landscapeElement.nativeElement.clientWidth;
    var screenWidth =
      this.landscapeElement.nativeElement.parentElement.clientWidth;
    this.maxDeltaX = (landWidth - screenWidth) * -1;
  }
}
```

⁹<https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html>


```

        this.currentDeltaX = this.maxDeltaX / 2; // Aprovechamos para
            centrar el terreno al iniciar.
    }
    /* Resto de funciones */
}

```



Hay que tener en cuenta que el máximo desplazamiento se dará hacia la izquierda, por eso se trata de un número negativo.

Con las medidas ya calculadas, vamos a tratar el gesto de desplazar el dedo sobre la pantalla. Ionic2 permite vincular los siguientes gestos desde el *HTML*: **tap**, **press**, **pan**, **swipe**, **rotate** y **pinch**. Para nuestro propósito, el evento que necesitamos capturar es el de **pan** y lo haremos del mismo modo que capturamos el evento de click.

```

<div class="canvas sky" [@skyState]="state" (pan)="panEvent(\$event)"
>
  <div class="celestial sun" [@flyInOut] *ngIf="state === 'day'" (
    click)="toggleState()" ></div>
  <div class="celestial moon" [@flyInOut] *ngIf="state === 'night'" (
    click)="toggleState()" ></div>
  <div #landscape class="landscape" ></div>
</div>

```

Veamos primero el código del método *panEvent*, que es la que recoge y trata el gesto, y a continuación explicaré su funcionamiento.

```

panEvent(e) {
  if (e.eventType == 4) {
    this.lastPanDeltaX = 0;
    return true;
  };
  var dx = e.deltaX - this.lastPanDeltaX;
  this.lastPanDeltaX = e.deltaX;

  if ((this.currentDeltaX + dx) < this.maxDeltaX || (
    this.currentDeltaX + dx) > 0) {
    return true;
  }
  this.currentDeltaX += dx;

  return true;
}

```

Para entender este código hay que saber como funciona el evento que estamos capturando. Cada vez que se desplaza el dedo sobre la pantalla, por pocos píxeles que sean, se genera un evento que podemos capturar. Esto quiere decir que en caso de mover el dedo sin parar, el evento saltara cada pocos milisegundos. ¿Y que

información nos proporciona?. Entre los múltiples parámetros¹⁰ que tiene asociado el evento, el que nos indica el desplazamiento es *deltaX* (también nos encontramos con los parámetros *deltaY* y con *distance*, pero en nuestro caso, solo nos interesa el movimiento horizontal). Este parámetro se mide en píxeles y representa la distancia recorrida horizontalmente desde el inicio del gesto. Como nos interesa actualizar la posición del terreno cada vez que se genera el evento para así conseguir que el dibujo se mueva al mismo tiempo que el dedo, necesitamos saber la distancia recorrida no al punto de inicio del gesto, si no al punto en el que se produjo el evento anterior. Para eso calculamos la diferencia entre la *deltaX* del evento anterior, que guardamos en la variable *lastPanDeltaX*, y la *deltaX* del evento actual. Esta diferencia es la que se aplica al desplazamiento actual del terreno (*currentDeltaX*) siempre y cuando entre dentro del desplazamiento permitido (de *maxDeltaX* a 0). ¿Y cuándo termina el gesto?. Cuando el usuario levanta el dedo, se lanza un último evento en el que el parámetro *eventType*¹¹ toma el valor 4. En este momento, reseteamos *lastPanDeltaX*.

Ahora que tenemos el desplazamiento que tenemos que aplicar a nuestro terreno, nos falta justamente eso, aplicárselo. Angular nos permite modificar el estilo de un elemento según una variable desde el HTML. Para realizar un desplazamiento lateral vamos a aplicar una transformación¹² a nuestro elemento, mas concretamente, *translateX()*, al cual se le pasa el desplazamiento medido en píxeles.

Normalmente para cambiar un estilo según el valor de una variable con Angular se haría lo siguiente:

```
<div [style.CSS_PROPERTY_NAME]=" VARIABLE " ></div>
```

Pero en el caso del *transform*, esto viola algunas directivas de seguridad que aplica Angular, y es que sí, una de las cosas que Angular nos proporciona es protección contra las vulnerabilidades comunes que suelen tener las aplicaciones web ...aunque en esta ocasión interfiera con nuestros planes. Por suerte, existen mecanismos para saltarnos estas medidas de seguridad (bajo nuestra responsabilidad) para casos como el que nos ocupa. Estos mecanismos son conocidos como *bypass*¹³ y previenen de que el valor que les pasemos sea comprobado por los mecanismos de seguridad de Angular.

Para intentar dejar nuestro código lo más limpio posible, crearemos una tubería (ya vimos que son en la práctica del **Cronómetro**) que al pasarle el valor numérico, nos genere el valor a asignar a la propiedad *transform*.

¹⁰<https://hammerjs.github.io/api/#event-object>

¹¹<https://hammerjs.github.io/api/#input-events>

¹²https://www.w3schools.com/cssref/css3_pr_transform.asp

¹³<https://angular.io/docs/ts/latest/api/platform-browser/index/DomSanitizer-class.html>

```
import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizer } from '@angular/platform-browser';

@Pipe({name: 'transform_sanitizer'})
export class TransformSanitizer implements PipeTransform {

  constructor(private sanitizer:DomSanitizer){}

  transform(deltaX) {
    return this.sanitizer.bypassSecurityTrustStyle("translateX(" +
      deltaX + "px)");
  }
}
```

Ya solo nos quedaría añadirlo a nuestro *HTML*:

```
<div class="canvas sky" [@skyState]="state" (pan)="panEvent(\$event)"
  >
  <div class="celestial sun" [@flyInOut] *ngIf="state === 'day'" (
    click)="toggleState()" ></div>
  <div class="celestial moon" [@flyInOut] *ngIf="state === 'night'" (
    click)="toggleState()" ></div>
  <div #landscape [style.transform]="currentDeltaX |
    transform_sanitizer" class="landscape" ></div>
</div>
```

3.4.4. Animación CSS. El ave.

Nuestro siguiente paso será incluir un habitante en nuestro paisaje, yo me he decantado por un pájaro. Nuestro nuevo amigo estará de pie sobre el terreno, por ello, es necesario que se mueva cuando desplazemos todo el terreno. Para conseguirlo, el elemento que representa a este pájaro dentro del *div* del terreno. Veamos el código:

```

<div class="canvas sky" [
  @skyState]="state" (pan)="
  panEvent(\$event)" >
  <div class="celestial sun" [
    @flyInOut] *ngIf="state ===
    'day'" (click)="
    toggleState()" ></div>
  <div class="celestial moon" [
    @flyInOut] *ngIf="state ===
    'night'" (click)="
    toggleState()" ></div>
  <div #landscape [
    style.transform]="
    currentDeltaX |
    transform_sanitizer" class=
    "landscape" >
    <div class="bird" [@birdState
    ]="state" ></div>
  </div>
</div>

```

```

page-home {
  .canvas.sky{ ... }
  .celestial{ ... }
  .landscape {
    // Resto de propiedades.
    .bird {
      position: fixed;
      height: 100px;
      width: 100px;
      left: 30%;
      top: 60%;
      background-size: 100% 100%;
      background-repeat:
        no-repeat;
      pointer-events: none;
    }
  }
}

```

Ya que la fauna diurna no es la misma que la que nos encontramos en la noche, vamos a cambiar la especie de nuestro habitante utilizando una animación que modifique la imagen que se utiliza:

```

trigger('birdState', [
  state('day', style({
    'background-image': 'url(..../assets/landscape/bird.png)'
  })),
  state('night', style({
    'background-image': 'url(..../assets/landscape/owl.png)'
  })),
  transition('day <=> night', animate('0.5s 0.5s')), // La animación
    dura 0.5 segundos y empieza 0.5 segundos retrasada
]),

```

Para animar un poco a nuestro pájaro, y no parezca que se trata de una objeto sin vida, vamos a añadirle un poco de movimiento, y como no, vamos a usar una animación para esto. En este caso, al tratarse de una animación que se repite en el tiempo y de la que conocemos tanto su estado inicial como final e intermedios, definiremos la animación directamente en el **CSS**. Vamos a definir dos animaciones diferentes, durante el día, nuestro pájaro pivoteará de un lado a otro, y durante la noche, pegará saltos. **CSS** nos permite definir *keyframes*, en los que indicar el estilo según el porcentaje de animación que se haya completado.

```

@keyframes spin {
  0%, 45% {transform: rotateY(0deg);} // En el 45% empezará a
    cambiar
  50%, 95% {transform: rotateY(180deg);} // En el 50% ya se habrá
    realizado el cambio
}

```

```
@keyframes jump {
  0%, 45%, 55%   {top: 60%}
  50%   {top: 50%} // Al punto más alto se llega a mitad de la
                  animación
}
```

Completamos el estilo de nuestro elemento para añadir la animación. En este caso le pondremos por defecto la animación *spin* (y así ver en conjunto todas las opciones de la animación), aunque como veremos a continuación, esta animación cambiará junto al estado.

```
.bird {
  position: fixed;
  height: 100px;
  width: 100px;
  left: 30%;
  top: 60%;
  background-size: 100% 100%;
  background-repeat: no-repeat;
  pointer-events: none;
  animation-name: spin;
  animation-duration: 6s;
  animation-iteration-count: infinite;
  animation-timing-function: easeInQuint // https://www.w3schools.com/cssref/css3\_pr\_animation-timing-function.asp;
}
```

En nuestro componente cambiamos la definición del trigger *birdState*:

```
trigger('birdState', [
  state('day', style({
    'background-image': 'url(../assets/landscape/bird.png)',
    'animation-name': 'spin',
    'animation-duration': '6s'
  })),
  state('night', style({
    'background-image': 'url(../assets/landscape/owl.png)',
    'animation-name': 'jump',
    'animation-duration': '3s'
  })),
  transition('day <=> night', animate('0.5s 0.5s')),
])
```

Veamos como funciona la animación de giro con la configuración especificada. En primer lugar, la animación durará 6s. Esto, junto con la definición del *keyframe* tenemos que:

1. El elemento empieza rotado 0 grados.
2. En el segundo $6 \cdot 0.45 = 2.7$, empieza la animación de giro.

3. En el segundo $6 * 0.5 = 3$, el elemento está rotado 180 grados.
4. En el segundo $6 * 0.95 = 5.7$, el elemento empieza a rotar hacia la posición original.
5. En el segundo $6 * 1 = 6$, el elemento está en la posición original.
6. Vuelta a empezar.

El funcionamiento es análogo en la definición del salto.

3.4.5. Elementos extras. Sonidos y nubes.

Como broche final a nuestro paisaje, vamos a hacer que se reproduzca un sonido, en nuestro caso el canto del pájaro, cuando se realiza el cambio de estado. Las animaciones de Angular permiten asignar desde el *HTML* una acción tanto al comienzo de la animación, como al finalizarla.

```
<div #landscape [style.transform]="currentDeltaX | transform_sanitizer" class="landscape" >
  <div #bird class="bird" (@birdState.start)="playAudio()" [
    @birdState]="state" ></div>
</div>
```

```
export class HomePage implements AfterViewInit {
  //Resto de variables
  audio: any;

  //Resto de funciones
  playAudio() {
    if (this.audio) { //Nos aseguramos de que si existe algún sonido
      reproduciéndose, lo detenemos
      this.audio.pause();
    }
    this.audio = 'day' == this.state ? new Audio('../assets/sounds/
      bird.mp3') : new Audio('../assets/sounds/owl.mp3');
    this.audio.play();
  }
}
```

Con esto ya tendríamos completo nuestro paisaje. Podemos añadir más elementos si queremos poner en práctica lo aprendido o probar cosas nuevas, por ejemplo, yo he añadido unas nubes que también se mueven junto con el terreno, pero a una velocidad menor.

```

<div class="canvas sky" [
  @skyState]="state" (pan)="
  panEvent(\$event)" >
  <div class="celestial sun" [
    @flyInOut] *ngIf="state ===
    'day'" (click)="
    toggleState()" ></div>
  <div class="celestial moon" [
    @flyInOut] *ngIf="state ===
    'night'" (click)="
    toggleState()" ></div>
  <div [style.transform]="
    currentDeltaX / 3 |
    transform_sanitizer" class=
    "cloud left" ></div>
  <div [style.transform]="
    currentDeltaX / 3 |
    transform_sanitizer" class=
    "cloud right" ></div>
  <div #landscape [
    style.transform]="
    currentDeltaX |
    transform_sanitizer" class=
    "landscape" >
    <div class="bird" (
      @birdState.start)="
      playAudio()" [@birdState]
      ="state" ></div>
  </div>
</div>

```

```

page-home {
  .canvas.sky{ ...
  .celestial{ ... }
  .cloud {
    position: fixed;
    height: 60px;
    width: 150px;
    background-image: url(../
      assets/landscape/
      cloud.png);
    background-size: 100% 100%;
    background-repeat:
      no-repeat;
    pointer-events: none;
    &.left {
      top: 8%;
      left: 25%;
    }
    &.right {
      top: 12%;
      left: 65%;
    }
  }
  .landscape { ... }
}

```

3.4.6. Apunte final y uso de animaciones en otros escenarios.

Con esta práctica he querido abarcar diferentes formas de crear animaciones cuando se trata de una aplicación hecha con Ionic2. Se puede conseguir el mismo resultado aplicando métodos distintos. Por ejemplo, haciendo las animaciones solo con JS, utilizar jQuery u alguna otra librería externa, ...

Como último apunte, indicar que aunque en este caso hemos estado animando elementos que representaban imágenes para intentar hacer la explicación más visual y llamativa, no debemos pensar que las animaciones solo se utilizan para hacer "dibujitos". Las animaciones se pueden aplicar a cualquier elemento del árbol **DOM** y a cualquier regla que se pueda definir en **CSS**. Un ejemplo muy utilizado sería cambiar el tamaño de fuente en un texto cuándo nos posicionamos sobre él:

```

<head>
  <style media="screen" >
    .text {
      font-size: 14px;
    }
  </style>

```

```

        transition: font-size 1s ease-in; \\ Manera abreviada de definir
        una transición
    }
    .text:hover {
        font-size: 40px;
    }
</style>
</head>
<body>
    <div class="text" >
        Pasa sobre mí el ratón.
    </div>
</body>

```

O desplegar un menú lateral:

```

<head>
<style media="screen" >
    body {
        display: flex;
        height: 100%;
    }
    div {
        height: 100%;
    }
    div.menu {
        background-color: green;
        width: 20px;
        transition: width 1s, background-color 1s;
    }
    div.menu:hover {
        background-color: yellow;
        width: 200px;
    }
    div.main {
        background-color: cyan;
        flex-grow: 1;
    }
    button {
        min-width: 200px
    }
</style>
</head>
<body>
    <div class="menu" >
        <button>OPTION 1</button>
        <button>OPTION 2</button>
        <button>OPTION 3</button>
    </div>
    <div class="main" ></div>
</body>

```


3.5. Recordatorios asociados a localizaciones

En esta última práctica se propone crear una aplicación que permita al usuario gestionar una serie de recordatorios, pero en vez de estar asociados estos recordatorios a un instante de tiempo en concreto, se asocien a un área geográfica concreta. Vamos a definir esta área como una circunferencia de 100 metros de radio alrededor de un punto geográfico que será definido por el usuario cuando cree el recordatorio. A esta circunferencia la llamaremos “área de influencia”. Para ello, el usuario seleccionara un punto sobre el mapa y asignará un nombre y una descripción para este recordatorio.

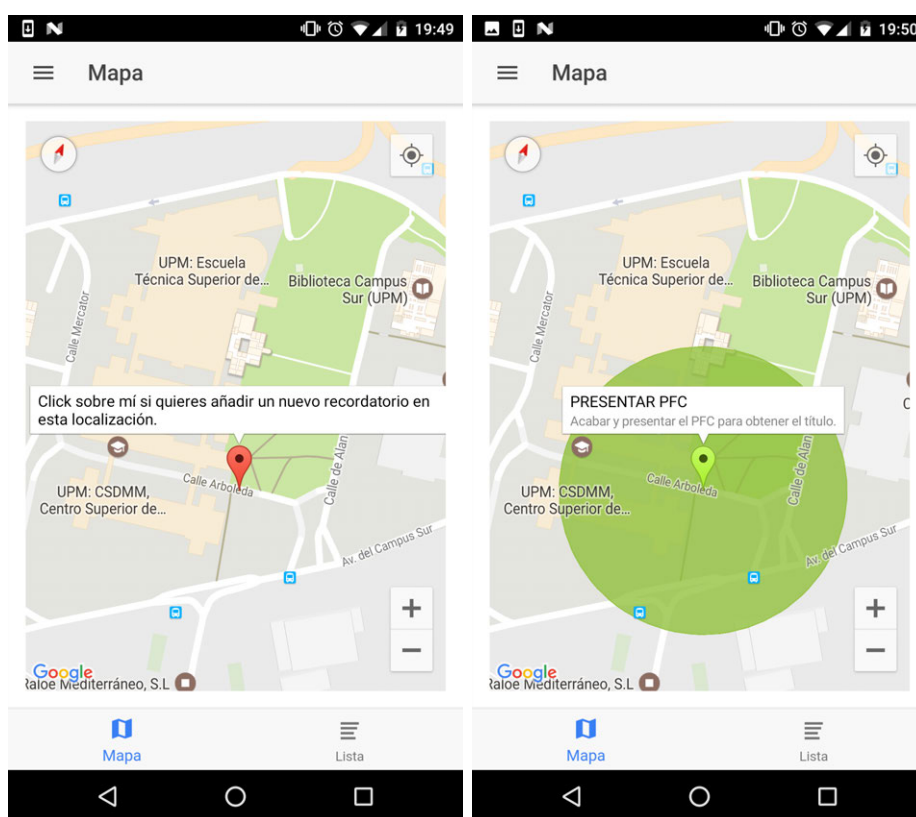


Figura 3.18: Así se vería el mapa, con los recordatorios marcados.

También podremos ver estos recordatorios listados, pudiendo interactuar con ellos:

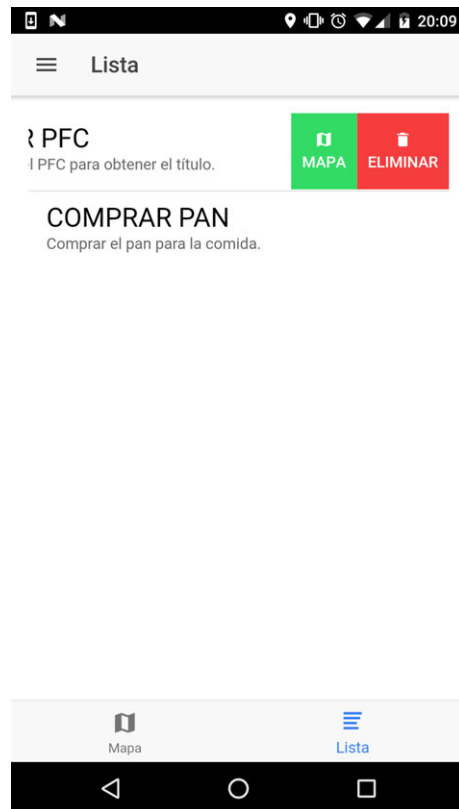


Figura 3.19: Aquí vemos la lista de recordatorios, sobre los que podremos interactuar.

Los recordatorios quedarán almacenados de manera persistente en la **BBDD** que implementa el sistema operativo, así, la información no se perderá al cerrar la aplicación ni al apagar el dispositivo.

Y como objetivo último de la aplicación, se notificará al usuario de que cerca de la zona en la que se encuentra, tiene registrado un recordatorio. Dicha notificación se activará al entrar dentro del área de influencia de este:

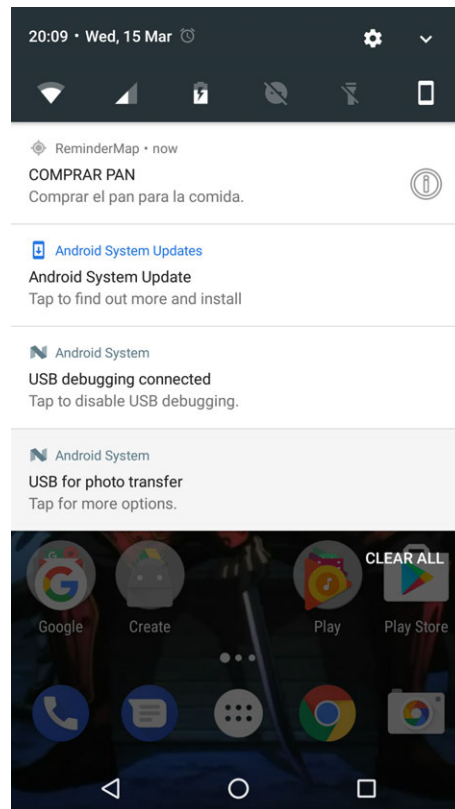


Figura 3.20: La notificación aparece aunque no este abierta la aplicación.

Para el desarrollo de esta aplicación se vamos a usar algunos módulos de Ionic Native. Como ya se vio en el apartado dedicado a **Ionic Native**, se trata de un conjunto de envoltorios sobre algunos de los plugins con los que cuenta Cordova facilitando así su uso desde nuestro código escrito en TypeScript.

Entre todos los plugins disponibles, vamos a usar los siguientes:

1. Google Maps¹⁴: Nos permitirá manejar de manera sencilla datos geográficos y visualizarlo sobre un mapa, crear y manipular estos mapas, acceder a los datos de geolocalización del dispositivo, ...para ello utiliza el SDK nativo de Google Maps.
2. GeoFence¹⁵: También relacionado con la localización, pero implementado en un plugin diferente al anterior, nos permite crear alertas para cuando el dispositivo entre dentro de una zona geográfica definida. Estas alertas se mantienen incluso con la aplicación apagada.
3. SQLite¹⁶: Nos permite acceder a la **BBDD** del dispositivo pudiendo hacer que

¹⁴<https://ionicframework.com/docs/native/google\discretionary{-}{-}{-}maps/>

¹⁵<https://ionicframework.com/docs/native/geofence/>

¹⁶<https://ionicframework.com/docs/native/sqlite/>

nuestros datos sean persistentes mediante consultas **Structure Query Language (SQL)**

Además, vamos a aprovechar esta práctica para ver como hacer uso de dos componentes de Ionic2 que son los *tabs o pestañas* y el *sidemenu o menú lateral*. Ambos componentes son muy utilizados en aplicaciones de todo tipo donde se quiere facilitar al usuario la navegación entre las diferentes páginas que la componen.

3.5.0.1. Análisis funcional

En esta aplicación se van a usar varios elementos como varias páginas, servicios, modelos de datos, plugins de Cordova, por lo que es interesante realizar un análisis funcional para tener una visión de los elementos que compondrán la aplicación y como interactúan entre ellos.

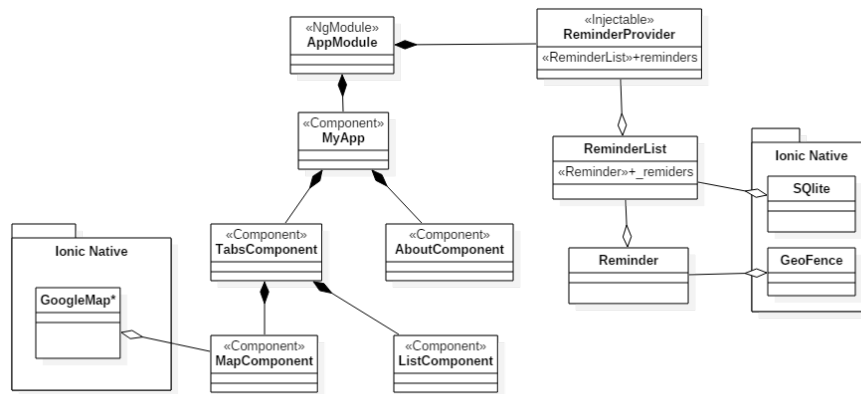


Figura 3.21: Diagrama de clases.

En el diagrama podemos ver como de nuestro módulo único parten las diferentes páginas de la aplicación. En primer lugar encontramos la página *MyApp*, de la cual cuelgan todas las demás. *MyApp* contendrá el menú lateral y desde el que se accederá a las páginas *TabsComponent* y *AboutComponent*. A su vez, *TabsComponent* actuará de contenedor para *MapComponent* y *ListComponent*.

Por otro lado tenemos la clase *Reminder*, que actuará como modelo y representará un recordatorio. Por encima de esta clase, se encuentra *ReminderList*, que actuara como *manager*, facilitando el manejo de los recordatorios y de la comunicación con la **BBDD**.

También existirá un *provider* que se usará para compartir inicializar un *ReminderList* y que pueda ser compartido por todos los componentes de nuestra

aplicación que lo necesiten, en nuestro caso, los componentes *MapComponent* y *ListComponent*. Esto se consigue utilizando el *Dependency Injector* de Angular (ver 2.2).

Por último podemos ver el uso de los plugins de Ionic Native, comentados en la introducción, por parte de algunas de las clases.

3.5.0.2. Estructura de las páginas

En primer lugar vamos a crear las paginas de la aplicación y la navegación entre ellas. El aspecto que queremos tener en nuestra aplicación, y la navegación entre páginas se puede ver de manera gráfica en el siguiente esquema.

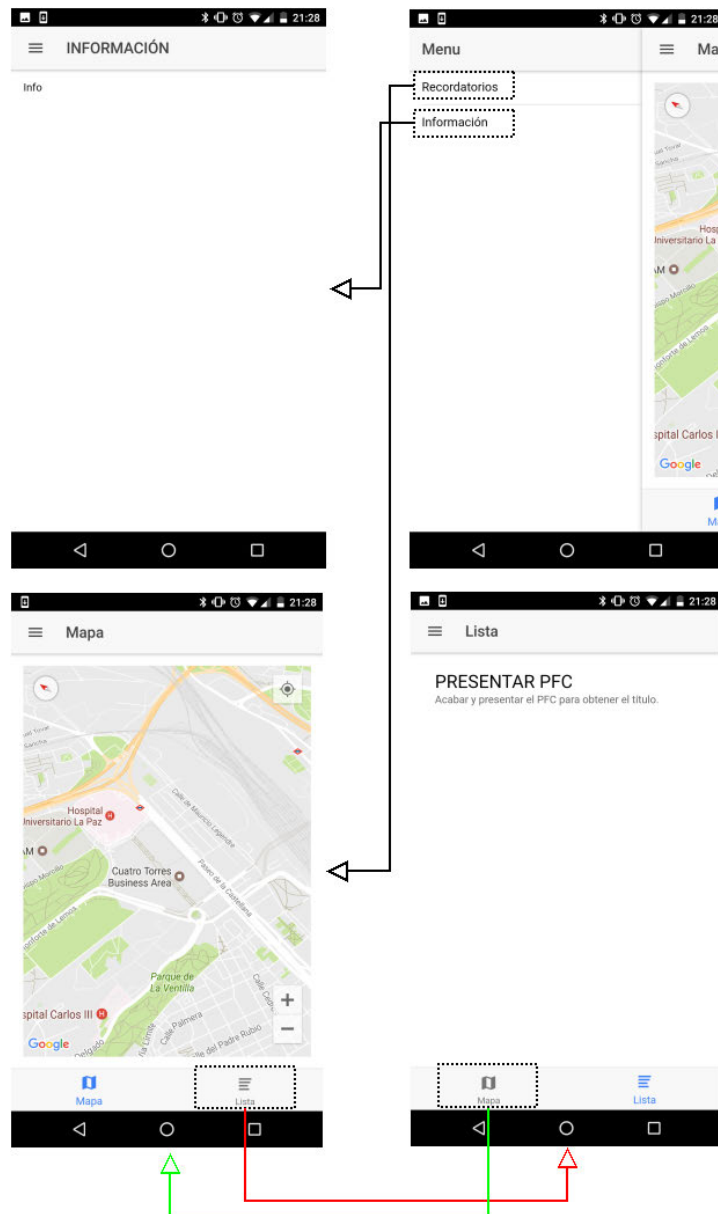


Figura 3.22: Mockup de la aplicación. Podemos ver los enlaces entre páginas y los menús desde las que se acceden.

Necesitaremos implementar cuatro páginas diferentes para nuestra aplicación:

1. En una de las páginas se colocarán la barra de *tabs* en la parte inferior. Además, deberá servir de contenedor para las siguientes dos páginas.
2. Una página mostrará el mapa y que irá contenida en la anterior.
3. Similar a la anterior, necesitaremos otra página para mostrar la información pero en formato de lista y también contenida en la primera.

4. Una última página, esta vez dependiente de la página de *tabs*, para mostrar la información sobre la aplicación.

A su vez, como ocurre en todos los proyectos, estas páginas estarán contenidas dentro del componente padre de la aplicación, *app*, y que será donde se implementará el menú lateral.

Vamos a empezar, pero en esta ocasión, en vez de usar un *starter* vacío como era *blank*, vamos a utilizar uno que ya implementa un menú lateral y al que iremos añadiendo nuestra páginas. Este *starter* lo podemos encontrar en el market de Ionic2 y nos va a proporcionar la base para empezar una aplicación que cuente con un menú lateral. He aquí la importancia de revisar este market antes de ponerse a desarrollar por si podemos aprovechar lo ya desarrollado por otra persona.

Iniciamos nuestro proyecto con el siguiente comando, en el que podemos ver que el nombre del *starter* a utilizar es **sidemenu**:

```
# ionic start ReminderMap sidemenu --v2
```

Si ejecutamos el proyecto tal cual, veremos que podemos abrir un menú lateral en el que aparecen dos páginas, a las que podemos acceder pulsando sobre ellas.



Figura 3.23: Menú lateral que nos proporciona la plantilla *sidemenu*.

Estas páginas que vemos se crean por defecto. Pueden ser editadas y añadir así nuestro contenido o, como vamos a hacer nosotros, podemos eliminarlas y crear las nuestras desde cero. Generaremos nuestras páginas utilizando Ionic CLI, consiguiendo que nuestras páginas tengan la estructura necesaria de ficheros usando un único comando. Las páginas a crear son las comentadas unos párrafos atrás:

```
# ionic generate page tabs
# ionic generate page map
# ionic generate page list
# ionic generate page about
```

Empezaremos modificando el menú lateral para que muestre la páginas *tabs* y *about*. A continuación, añadiremos el componente de Ionic **ion-tabs** a la página *tabs* y enlazaremos las páginas *maps* y *list* a este componente.

Editamos el archivo `/app/app.html`, que actúa como página padre de la aplicación. Aquí vemos la primera implementación que añade el *starter* **sidemenu**.

```
<ion-menu [content]="content" >
  <ion-header>
    <ion-toolbar>
      <ion-title>Menu</ion-title>
    </ion-toolbar>
  </ion-header>

  <ion-content>
    <ion-list>
      <button menuClose ion-item *ngFor="let p of pages" (click)="
        openPage(p)" >
        {{p.title}}
      </button>
    </ion-list>
  </ion-content>
</ion-menu>

<!-- Disable swipe-to-go-back because it's poor UX to combine STGB
with side menus -->
<ion-nav [root]="rootPage" #content swipeBackEnabled="false" ></
ion-nav>
```

Encontramos el componente **ion-menu**, el cual se inicia con los valores que encuentra dentro de la propiedad *pages*, que pertenece al componente *MyApp* que se encuentra dentro del fichero *app.component.ts*. Lo abrimos y vemos que se trata de un diccionario en el que se indica el título del botón y la página a la que hace referencia. También vemos la función **openPage**, que es llamada desde los botones del menú (función asignada al evento *click*) y que se encarga de abrir la página

correspondiente. Cambiamos pues este diccionario para introducir nuestras páginas de la siguiente forma:

```
this.pages = [
  { title: 'Recordatorios', component: TabsPage },
  { title: 'Información', component: AboutPage }
];
```

También cambiamos la variable **rootPage**, que define la página de inicio, por **TabsPage**. No nos debemos olvidar de importar nuestras páginas.

Como estamos usando páginas creadas por nosotros, tendremos que declararlas y añadirlas como *entryComponents* ¹⁷. Estos cambios se han de realizar en la definición del *ngModule*, en el fichero *app.module.ts*. Eliminamos las páginas que venían definidas en el *starter* y añadimos nuestras propias páginas.

```
declarations: [
  MyApp,
  AboutPage,
  TabsPage,
  MapPage,
  ListPage
],
...
entryComponents: [
  MyApp,
  AboutPage,
  TabsPage,
  MapPage,
  ListPage
]
```

Ya tenemos creado nuestro menú lateral con el que acceder a nuestra página principal y a la de información. Ahora vamos a añadir el componente **ion-tabs** a nuestra página *tabs.html*. La editamos con el siguiente código:

```
<ion-tabs selectedIndex="0" >
  <ion-tab [root]="mapTabRoot" tabTitle="Mapa" tabIcon="map" ></
    ion-tab>
  <ion-tab [root]="listTabRoot" tabTitle="Lista" tabIcon="list" ></
    ion-tab>
</ion-tabs>
```

Como se observa, se han añadido dos *tabs* a los que se les ha definido un texto (atributo **tabTitle**) y un icono (de los disponibles en la fuente Ionicons y definido por el atributo **tabIcon**). Con el atributo **[root]** se indica la página que se debe mostrar al seleccionar esa pestaña. El parámetro que se indica en este atributo debe estar definido como propiedad en el componente de la página, en *TabComponent*:

¹⁷<https://angular.io/docs/ts/latest/cookbook/ngmodule-faq.html>

```
@Component({
  templateUrl: 'tabs.html'
})
export class TabsPage {
  mapTabRoot: any = MapPage;
  listTabRoot: any = ListPage;

  constructor() {}
}
```

Por último, y para acabar con la estructura de páginas que tendrá la aplicación, añadiremos cabeceras que contengan a cada una de las páginas con un título que las identifique junto a un botón con el que abrir el menú lateral. La cabecera para todas ellas sería así:

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu" ></ion-icon>
    </button>
    <ion-title>INFORMACIÓN</ion-title>
  </ion-navbar>
</ion-header>
```

Esta cabecera deberá ir en todas las páginas menos en la página *tabs*, ya que está actúa de contenedor de páginas que poseen su propia cabecera.

3.5.0.3. Instalando los plugins necesarios

Antes de empezar a implementar la lógica, vamos a instalar los plugins que hemos enumerado y que necesitaremos para nuestra aplicación. Los plugins se pueden instalar utilizando Ionic CLI. Se recomienda añadir en primer lugar las plataformas objetivo al proyecto, y a continuación, instalar los plugins. Empecemos:

```
# ionic platform add android
```

Seguimos con un plugin usado para gestionar la política de acceso a páginas desde la aplicación. No lo usaremos directamente, pero es necesario para que nuestro mapa funcione.

```
# ionic plugin add cordova-plugin-whitelist
```

El caso del plugin de Google Maps tiene la particularidad de que hay que indicar que API KEY del servicio debe utilizar ya en el momento de la instalación del módulo. En el anexo [Google APIs](#) podemos ver como conseguir esta API KEY. En el momento de escribir este documento, el plugin se encuentra en la versión 1.4,

aunque la versión 2 ya está desarrollada y en fase beta. Vamos a usar la versión 1.4, la cual indicaremos en el comando para asegurarnos de que es la que se instala.

```
# ionic plugin add cordova-plugin-googlemaps@1.4 --variable API_KEY
```

El resto de plugin no tienen ninguna peculiaridad a destacar.

```
# ionic plugin add cordova-sqlite-storage
# ionic plugin add cordova-plugin-geofence
```

Cabe destacar la facilidad que ofrece no Ionic a la hora de añadir nuevos plugin a cualquier aplicación.

3.5.0.4. Implementación del modelo de datos y la persistencia

Como hemos visto en el análisis, nuestra aplicación va a contar con una clase que actúe de modelo y que contenga la información de un recordatorio. Ya que la aplicación podrá manejar varios recordatorios, y estos deberán ser guardados en la **BBDD** del dispositivo y poder ser recuperados de ella, se implementará una clase que abstraiga de estas tareas al resto de clases.



Como ocurre en muchos otros lenguajes, existen módulos para Ionic2 que implementan el modelo **ORM**¹⁸. Este modelo se basa en el mapeado de las tablas de una **BBDD** en entidades que simplifiquen las tareas básicas de acceso a esas tablas.

Vamos a empezar definiendo la clase *Reminder* que actuara de modelo para nuestros recordatorio.

```
export class Reminder {
  private _id: number;
  private _name: string;
  private _description: string;
  private _lat: number;
  private _lng: number;

  constructor(name: string, description: string,
    latLng: GoogleMapsLatLng | Array<number>, id: number=null) {
    this._id = id;
    this.name = name;
    this.description = description;
    if (latLng instanceof Array) {
      this._lat = latLng[0];
      this._lng = latLng[1];
    } else {
      this._lat = latLng['lat'];
    }
  }
}
```

¹⁸<https://github.com/BradyLiles/ionic-orm>

```

        this._lng = latLng['lng'];
    }
}

get id(): number { return this._id;
}

get name(): string { return this._name;
}

set name(name: string) { this._name = name.substr(0, 16);
}

get description(): string { return this._description;
}

set description(description: string) {
    this._description = description.substr(0, 512);
}

get latLng(): GoogleMapsLatLng {
    return new GoogleMapsLatLng(this._lat, this._lng);
}

set latLng(latLng: GoogleMapsLatLng ) {
    this._lat = latLng['lat'];
    this._lng = latLng['lng'];
}
}

```

Se puede observar que se han declarado las variables privadas y se ha decidido implementar sus métodos *getter* y *setter*. Con esto se consigue tener control sobre asignación de valores a estas variables, como por ejemplo, la longitud del nombre o la imposibilidad de modificar el id. Por otro lado, la localización se define como dos variables, una para la latitud y otra para la longitud, pero a la hora de querer modificar o recuperar esta localización, se usa un objeto de tipo *GoogleMapsLatLng*. Se ha decido hacer esto así ya que este tipo de objetos es el que usa el plugin de Google Map para representar una coordenada geográfica. Pero, ¿y por qué no definirlo también así en nuestro modelo?. La razón es que nuestro modelo va a ser almacenado dentro de una **BBDD**, y no podremos guardar directamente en ella el objeto *GoogleMapsLatLng* directamente, sino que necesitaremos serializarlo o descomponerlo en elementos de tipo básico. Además, no siempre se puede usar *GoogleMapsLatLng* para definir una coordenada, como ya veremos a la hora de usar *GeoFence*.

Implementamos también nuestro *ReminderList*, que tendrá como variable un *Array* de elementos de tipo *Reminder*, el cuál será privado y accedido a través de la función *all()*:

```
export class ReminderList {
  private _reminders: Array<Reminder>;

  constructor(public events: Events) {}

  all(): Array<Reminder> { return this._reminders;
}
}
```

Con estas dos clases ya definidas, vamos a ir añadiéndoles funcionalidades. Para empezar, añadiremos persistencia a los datos, es decir, crearemos los métodos necesarios para poder leer y escribir en la **BBDD** del dispositivo. Tendremos que importar la clase **SQLite** desde el paquete **ionic-native**.

Para utilizar una **BBDD**, debemos instanciar un objeto de esta clase con el cual podremos abrir una **BBDD** usando el método **openDatabase** y ejecutar queries sobre ella con el método **executeSql**. Ambos métodos nos devolverán promesas, en las cuales indicaremos la acción a realizar en función de si la operación haya ido bien o mal. Empecemos viendo como quedaría el constructor de nuestra clase *ReminderList*.

```
export class ReminderList {
  private _db: SQLite;
  // RESTO DE VARIABLES

  constructor(public events: Events) {
    this._db = new SQLite();
    this._db.openDatabase({
      name: 'reminder.db',
      location: 'default'
    }).then(() => {
      this._db.executeSql('CREATE TABLE IF NOT EXISTS reminders (id
        INTEGER PRIMARY KEY,' +
        'name VARCHAR(16), description VARCHAR(512), lat FLOAT, lng
        FLOAT)', []
      ).then((data) => {
        this.refresh();
      }, (err) => {
        console.error('Unable to execute sql: ', err);
      });
    }, (err) => {
      console.error('Unable to open database: ', err);
    });
  }
  // RESTO DE MÉTODOS
}
```

En primer lugar se instancia un objeto *SQLite* el cual asignamos a una propiedad privada de *ReminderList*. Abrimos la **BBDD** con el ya mencionado método **openDatabase**, al cual se le indica el nombre de la **BBDD** y la

localización (usaremos *default*). Si la **BBDD** se ha abierto sin problemas, podremos ejecutar nuestra primera query. Con esta primera query crearemos la tabla en la que se escribirá la información de los recordatorios, solo si no existe ya. Se han definido las columnas acorde al tipo de dato que van a almacenar, además, se ha definido la columna *id* como **PRIMARY KEY**. Si todo ha ido según se espera, podremos llamar al método *refresh()*, el cual poblará el **Array** *_reminder* con la información de la tabla. La definición de este método es la siguiente:

```
onUpdate: EventEmitter<Array<Reminder>> = new EventEmitter();

refresh() {
  this._reminders = [];
  this._db.executeSql('SELECT * FROM reminders', []).then(
    (data) => {
      for (let i = 0; i < data.rows.length; i++) {
        let item = data.rows.item(i);
        this._reminders.push(new Reminder(item.name,
          item.description,
          [item.lat, item.lng], item.id))
      }
      this.onUpdate.emit(this._reminders);
    }, (err) => {
      console.error('Unable to execute sql: ', err);
    }
  )
}
```

En este método se asigna un array vacío a la variable *_reminder*. Acto seguido, se ejecuta una query **SQL** para recuperar toda la información de la tabla. Si todo ha ido bien, las filas que devuelve la query son mapeados en un **JSON** que se pasa como parámetro a la primera función que se indica en la promesa que devuelve **executeSql**. Recorremos estas filas y vamos instanciando nuevos recordatorios que añadimos a nuestro **Array**. También vamos a añadir a la clase *ReminderList* un **EventEmitter**. A este emisor de eventos podrán subscribirse el resto de clases para ser notificadas, en este caso, de que se han actualizado la lista de recordatorios. Veremos como usarlo cuando implementemos el *MapsComponent*. Gracias a estos eventos, podemos comunicar diferentes clases dentro de nuestra aplicación de manera sencilla.

Ya tenemos como recuperar los recordatorios desde la **BBDD**, nos falta poder añadir nuevos y eliminar los ya creados. Esto lo haremos con dos nuevos métodos en *ReminderList*:

```
addReminder(name: string, description: string, latLng:
  GoogleMapsLatLng) {
  let rem = new Reminder(name, description, latLng); // Se llama al
    constructor sin pasarle un id como parametro
  rem.save(this._db);
```

```

        this.refresh();

        return rem;
    }

    removeReminder(rem: Reminder) {
        rem.remove(this._db);
        this.refresh()

        return rem;
    }

```

Curioso. En ninguno de los métodos se hace una consulta a la **BBDD**, si no que se invoca a unos métodos pertenecientes a la clase *Reminder* (veremos a continuación la implementación de estos métodos). Pero, ¿por qué?. Si pensamos que las acciones tanto de añadir como de eliminar solo afectan al recordatorio en cuestión, resulta comprensible que esta puedan ser llevadas a cabo por el recordatorio mismo. Además, si otra clase distinta a *ReminderList* quisiera guardar un recordatorio en la BBDD, no necesitaría implementar la lógica para ello en dicha clase, solo tendría que usar las facilidades que implementa el propio *Reminder*. Cabe destacar que a la hora de instanciar el nuevo objeto *Reminder* en el método *addReminder*, no se le pasa el valor del ID. Esto se debe a que esta columna fue definida como índice de la tabla y será generado por la **BBDD** como veremos a continuación.

Ahora solo nos queda implementar los métodos de la clase *Reminder* de los que hablábamos antes:

```

save(db) {
    db.executeSql('INSERT INTO reminders (name, description, lat, lng)
        VALUES (?, ?, ?, ?)',
        [this._name, this._description, this._lat, this._lng]).then(
            (data) => { this._id = data.insertId; // El ID devuelto por la
                \gls{BBDD}, se asigna al campo _id del recordatorio.
            }, (err) => { console.error('Unable to execute sql: ', err);
            }
        )
    return this;
}

remove(db) {
    if (!this._id) { return this;
    }
    db.executeSql('DELETE FROM reminders WHERE id = ?', [this._id]).
        then(
            (data) => {
                this._id = null;
            }, (err) => { console.error('Unable to execute sql: ', err);
            }
        )
    return this;
}

```

Como ya comentábamos, el ID se asigna cuando se añade el recordatorio a la **BBDD**, siendo este ID pasado como campo dentro del **JSON** de datos que envía la promesa en caso de que todo haya ido bien. Por su parte, el método de borrado comprueba antes de realizar la consulta *SQL* que el objeto tiene asignado un ID, ya que usará este ID para identificar el recordatorio dentro de la tabla de la **BBDD**.

Podemos ver que, aunque instanciamos un objeto de la clase *Reminder*, esto no quiere decir que ya lo hayamos almacenado en la **BBDD**. Será necesario llamar a su método *save* para ello.



Pensemos en el caso de que queramos acceder a la tabla de recordatorios no solo desde la clase *ReminderList*, si no también desde otras clases. Esto obligaría a duplicar el código que se utiliza para crear la tabla y para leerla. En ese caso sería interesante que toda esta lógica se encontrara también dentro de *Reminder* en forma de métodos estáticos. En nuestra aplicación no nos será necesario, por lo que se ha decidido dejar esa lógica dentro de *ReminderList*.



En este caso, al leer los recordatorios desde la base de datos, se hace una lectura de todos sin aplicar ningún filtro. Pero, ¿y si queremos filtrar por alguno de sus campos? ¿y si quisiéramos realizar búsquedas sobre el campo *nombre* del recordatorio?. En este caso podríamos o bien, leer y recuperar todos los recordatorios desde la **BBDD** y luego descartarlos utilizando JavaScript, o se puede directamente añadir condiciones mediante la cláusula *WHERE* a la query *SQL* para solo recuperar de la **BBDD** los registros deseados. Esta segunda opción es de lejos la más recomendable ya que se evita por un lado tener que recuperar información de la **BBDD** que luego va a ser descartada y además, uno de los puntos fuertes de las **BBDD** es precisamente aplicar filtros, por lo que será más rápido.

3.5.0.5. *ReminderProvider*

Una vez que tenemos implementadas las clases que nos permitirán manejar los recordatorios, debemos pensar como interactuarán los componentes de la aplicación con ellas.

Tenemos dos componentes que necesitarán acceder a la lista de recordatorios, que son *MapComponent* y *ListComponent*. Podríamos hacer que cada uno de ellos instanciara un objeto de la clase *ReminderList*, pero sería duplicar recursos y peticiones. Para solventar esto crearemos un *provider*, que se trata una clase *singleton* (solo se instancia una vez) y que estará disponible para ambos componentes gracias a **Dependency Injector (DI)**. Este *provider*, al igual que ocurría con las páginas, puede ser generado utilizando Ionic CLI:

```
# ionic generate provider Reminder
```

Sobrescribimos el contenido:

```
@Injectable()
export class Reminder {
  private _reminders: ReminderList;

  constructor() {
    this._reminders = new ReminderList();
  }

  get reminders(): Array<Reminder> {
    return this._reminders.reminders;
  }
}
```

Y no nos olvidemos de registrarlo en el *AppModule*:

```
@NgModule({
  // Resto de declaraciones
  providers: [{provide: ErrorHandler, useClass: IonicErrorHandler},
              {provide: ReminderProvider, useClass: ReminderProvider}
            ]
})
export class AppModule {}
```

Como se puede ver, se trata de una clase muy simple y que nos solventa el hecho de tener que compartir un mismo objeto entre diferentes clases.

3.5.0.6. El mapa

Una vez que contamos con los elementos necesarios para manejar los recordatorios dentro de nuestra aplicación, llega el momento de implementar la visualización de estos recordatorios. Empezaremos por la página del mapa. Para crear este mapa, necesitaremos un contenedor, dentro del cual, el plugin de Google Maps se encargará de “dibujar” el mapa. Este contenedor no será más que un **div** dentro del **HTML** de la página.

```
<ion-content padding>
  <div id="map" ></div>
</ion-content>
```

```
page-map {
  #map {
    width: 100%;
    height: 100%;
  }
}
```

Ahora abrimos el *component MapPage*, donde crearemos un método, *loadMap* en el que iniciaremos nuestro mapa.

```
loadMap() {
  let location = new GoogleMapsLatLng(40.4893538, -3.6827461);
  this.map = new GoogleMap('map', {
    'backgroundColor': 'white',
    'controls': {
      'compass': true,
      'myLocationButton': true,
      'zoom': true
    },
    'gestures': {
      'scroll': true,
      'tilt': true,
      'rotate': true,
      'zoom': true
    },
    'camera': {
      'latLng': location,
      'zoom': 15
    }
  });

  this.map.on(GoogleMapsEvent.MAP_READY).subscribe(() => {
    console.log('Map is ready!');
  });
}
```

Vemos como para crear un mapa, debemos instanciar la clase *GoogleMap*, contenida en **ionic-native**, cuyo constructor espera el id del **div** que usará para dibujar el mapa, y un **JSON** con la configuración del mapa (posición inicial, botones, gestos permitidos, ...) ¹⁹. Asignaremos este objeto a la variable *map* del componente para poder hacer uso de ella más adelante. También subscribiremos una función al evento *MAP_READY*, el cual se dispara cuando el mapa está listo. De momento solo haremos que se imprima un mensaje en consola cuando esto ocurra, más adelante, iremos poblando este método.

Llegados a este punto, y antes de añadir más código, se recomienda ejecutar nuestra aplicación y comprobar si el mapa funciona correctamente. No podremos usar para esto el servidor de Ionic (*ionic serve*) y acceder a él a través del

¹⁹<https://github.com/mapsplugin/cordova-plugin-googlemaps-doc/blob/master/v1.4.0/class/Map/README.md>

navegador, ya que el plugin Cordova Google Maps requiere que sea ejecutado sobre un dispositivo, ya sea emulado o real.



Nuestra aplicación requerirá de permisos para poder acceder al **GPS**. Debemos asegurarnos, una vez instalada en el terminal, de que cuenta con ellos.



En Android encontramos nos encontramos con la aplicación Google Play Services²⁰, a través de la cual, el resto de aplicaciones pueden acceder a diferentes servicios, en el caso de nuestra aplicación, a Google Maps. La razón de porque Android hace esto así es la de asegurarse de que todas las aplicaciones que ejecuta el dispositivo hacen uso de unas mismas librerías a la hora de utilizar sus servicios. Librería la cual puede mantener actualizada fácilmente ya que se trata de una aplicación como otra cualquiera.

Si después de añadir la plataforma Android a nuestro proyecto, abrimos el fichero `/platforms/android/project.properties`, y vamos añadiendo los plugin, veremos como se añaden líneas como: **cordova.system.library.1=com.google.android.gms:play-services-location:9.8.0**. Esta línea indica que librería, dentro de Google Play Services se va a usar, así como la versión. En el momento de desarrollar esta app y probar a ejecutarla, se producía un error al intentar construir el APK. Este error se debía a que el plugin Cordova Geofence añade la siguiente línea a `project.properties` **cordova.system.library.1=com.google.android.gms:play-services-location:+**, haciendo que se utilizara la versión 10.2 de la librería. Cambiando esta versión por la 9.8.0 se pudo evitar el error y generar la APK

Con el mapa ya funcionando, el siguiente paso será poder crear nuevos recordatorios desde él y poder visualizar los ya presentes en la **BBDD**. En el siguiente diagrama de actividad se describe como será el proceso para crear un nuevo recordatorio.

²⁰<https://developers.google.com/android/guides/overview>

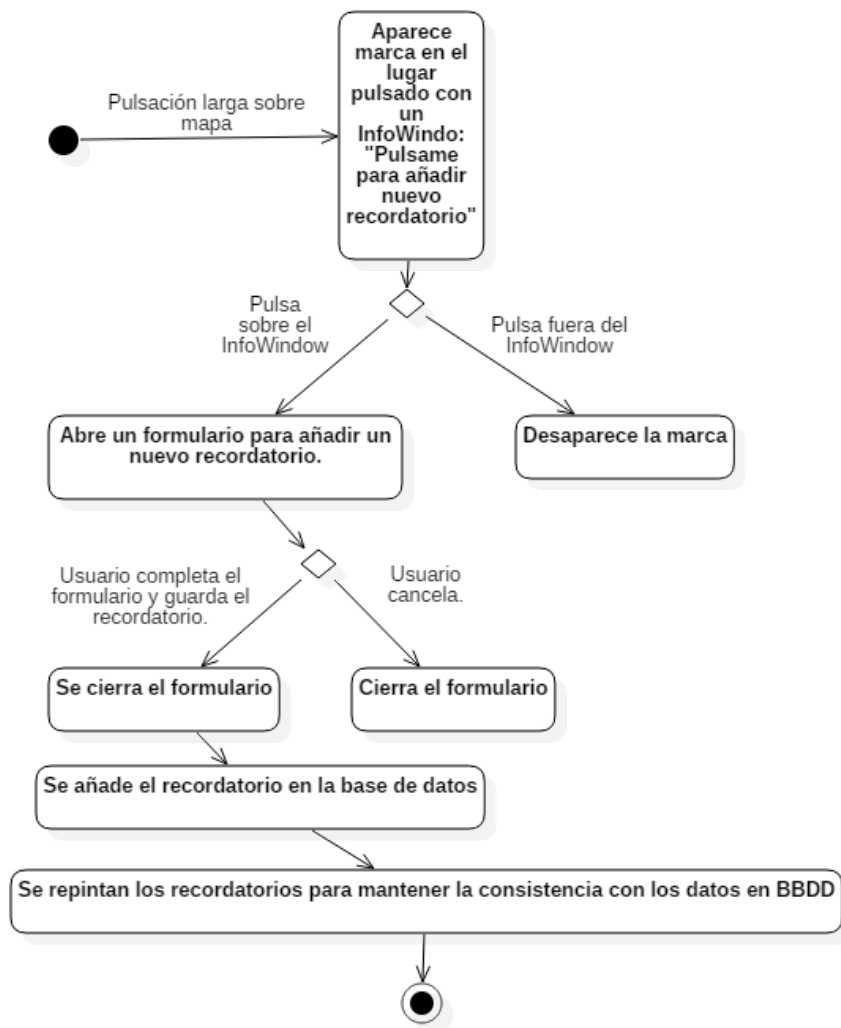


Figura 3.24: Diagrama de actividad que muestra la creación de un nuevo recordatorio.

Los elementos con los que el usuario interactuará durante el proceso (marca en el mapa, inforwindow, ...) son elementos que el propio framework pone a disposición del desarrollador, por lo que no tenemos que preocuparnos de implementarlos (diseñarlos, escribir el HTML, crear funciones que controlen la lógica, ...), lo cual supone un ahorro de tiempo considerable.

Modificaremos el código de nuestro *MapComponent* de la siguiente manera:

```

loadMap() {
  // RESTO DE LA FUNCIÓN

  this.map.on(GoogleMapsEvent.MAP_LONG_CLICK).subscribe((latLng) => {
    {
      this.showAddNewReminderInfoWindow(latLng);
    }
  });
}

```

```
showAddNewReminderInfoWindow(latLng: GoogleMapsLatLng) {
  let markerOptions: GoogleMapsMarkerOptions = {
    'position': latLng,
    // En las opciones de la marca, podemos configurar el infoWindow
    // asociado a esta.
    'title': 'NUEVO RECORDATORIO'
    'snippet': "Click sobre mí si quieres añadir un nuevo
    recordatorio en esta localización.",
    'infoClick': (marker) => {
      // Se elimina la marca del mapa y se abre el formulario para
      // añadir un nuevo recordatorio.
      marker.remove();
      this.showAddNewReminderPrompt(latLng);
    }
  };

  // Añadimos la marca al mapa, y cuando esté lista, hacemos que
  // muestre en el infoWindow.
  this.map.addMarker(markerOptions).then((marker: GoogleMapsMarker)
    => {
      marker.showInfoWindow();
    });
}

showAddNewReminderPrompt (latLng: GoogleMapsLatLng) {
  // Deshabilitamos la interacción con el mapa
  this.map.setClickable( false )
  let prompt = this.alertCtrl.create({
    // Elegimos El mensaje que queremos que aparezca en nuestro
    // formulario
    title: 'Nuevo recordatorio',
    message: "Se creará un nuevo recordatorio para la localización
    seleccionada.",
    // Configuramos los diferentes inputs que va a tener nuestro
    // formulario indicando el texto que se mostrará y el nombre
    // con el que será identificado este valor a la hora de
    // devolver el los datos introducidos.
    inputs: [
      {
        name: 'name',
        placeholder: 'Nombre'
      },
      {
        name: 'description',
        placeholder: 'Descripción',
        type: 'text'
      },
    ],
    // También configuramos los botones que queremos que aparezcan,
    // en este caso indicamos el texto del botón y la acción a
    // realizar
    buttons: [
      {
        text: 'Cancelar',
        handler: data => { console.log('Cancel clicked');

```

```

    }
  },
  {
    text: 'Guardar',
    handler: data => {
      this.remindersProvider.reminders.addReminder(data['name'],
        data['description'], latLng);
    }
  }
]
});
// Hacemos que la interacción con el mapa se vuelva a habilitar
// cuando el formulario se cierre
prompt.onDidDismiss(() => this.map.setClickable( true ));
// Mostramos el formulario
prompt.present();
}

```

Se puede ver que se ha añadido al método *loadMap()* una nueva instrucción para capturar el evento asociado a las pulsaciones largas sobre el mapa. Este evento lleva asociado las coordenadas sobre la que se ha pulsado. Cuando esto se produce, se ejecuta la función que hace que aparezca una marca sobre el mapa. Como comentaba antes, este elemento lo ofrece el propio framework, en este caso, a través del plugin de Google Maps. Para crearlo, se utiliza el método *addMarker()* de nuestro mapa (que fue instanciado en el constructor), al cuál se le pasa un objeto de tipo *GoogleMapsMarkerOptions* con la configuración de la marca y el *InfoWindow* asociado. Para ver más opciones, se puede consultar la documentación de este elemento en la página de Github ²¹. El método *addMarker()* devuelve una promesa, lo que nos permite ejecutar código en el momento en el que la marca esté lista. Vamos a aprovechar esta promesa para hacer que muestre el *infoWindow* asociado a la marca.

En caso de que el usuario pulse sobre el *InfoWindow*, se ejecutará el método *showAddNewReminderPrompt()* que hará que aparezca un formulario con el cual añadir un nuevo recordatorio. Al igual que con las marcas, este formulario será implementado utilizando componentes que ofrece el Ionic. En este caso nos aprovecharemos del servicio de alertas, el cual no es más que un *provider* que podemos inyectar desde el constructor y que permite la creación de ventanas flotantes en nuestra aplicación. Para ello, solamente tenemos que llamar al método *create()* del servicio de alertas pasándole la configuración que deseamos. En la documentación de Ionic podemos ver que parámetros podemos configurar²².

²¹<https://github.com/mapsplugin/cordova-discretionary-googlemaps/wiki/marker>

²²<http://ionicframework.com/docs/v2/components/#alert>



A la hora de mostrar el formulario, necesitamos deshabilitar la iteración con el mapa. ¿A que se debe esto?. Para comprenderlo, debemos saber que el mapa que se muestra no se renderiza con el vista web, si no que se trata de otra vista que se coloca por detrás. Es el propio plugin el que se encarga de hacer transparentes los elementos *HTML* que contiene el mapa. Por esta razón, la propia alerta no puede gestionar por si misma la iteración con el mapa, y siendo por tanto tarea del desarrollador realizar esta gestión. Esto se aplica no solo para el formulario, también para cualquier otro elemento *HTML* que se encuentre por encima del mapa y que se encuentren fuera del elemento que contiene el mapa²³

Ahora veamos los cambios a realizar para dibujar sobre el mapa los recordatorios que tenemos en la **BBDD**. Recorreremos la lista de recordatorios que nos proporciona el *ReminderProvider* e iremos creando marcas en el mapa para cada uno de ellos. En este caso, almacenaremos estas marcas en una colección en la que relacionaremos cada marca con el recordatorio correspondiente para poder usarlo más adelante, en una funcionalidad que implementaremos cuando veamos el componente *ListComponent*. También dibujaremos un círculo cuyo centro será la marca y que representará el área de influencia del recordatorio seremos notificados. Este círculo se dibujara utilizando el método *addCircle*²⁴ del mapa.

```
private markers : Map<Reminder, GoogleMapsMarker> = new Map<Reminder
    , GoogleMapsMarker>();

// Esta función genera un color aleatorio.
static getRandomColor () {
    return '#' + ('0000' + (Math.random()*(1<<24)|0).toString(16)).
        slice(-6);
}

loadMap() {
    // RESTO DE LA FUNCIÓN
    this.map.on(GoogleMapsEvent.MAP_READY).subscribe(() => {
        console.log('Map is ready!');
        this.drawReminderMarkers(this.remindersProvider.reminders.all
            ());
        // Nos subscribimos al evento implementado en ReminderList
        para poder actualizar el mapa cuando se produzca una
        modificación en los datos.
        this.remindersProvider.reminders.onUpdate.subscribe(reminders
            => this.drawReminderMarkers(reminders))
    });
}
```

²³<https://github.com/mapsplugin/cordova-plugin-googlemaps/wiki/Map>

²⁴<https://github.com/mapsplugin/cordova-plugin-googlemaps/wiki/Circle>

```

drawReminderMarkers(reminders: Array<Reminder>) {
    // Limpiamos el mapa de todo lo que tenga ya dibujado y reseteamos
    // el registro de marcas.
    this.map.clear();
    this.markers.clear();

    for (let rem of reminders) {
        // Generamos el color de manera aleatoria, consiguiendo una
        // mejor presentación.
        let color = MapPage.getRandomColor();

        let markerOptions: GoogleMapsMarkerOptions = {
            'icon': color,
            'position': rem.latLng,
            'title': rem.name,
            'snippet': rem.description,
            'markerClick': (marker: GoogleMapsMarker) => {
                marker.showInfoWindow();
            }
        };

        // Si la marca se ha añadido correctamente, la guardamos y
        // dibujamos el círculo a su alrededor.
        this.map.addMarker(markerOptions).then((marker: GoogleMapsMarker)
        => {
            this.markers.set(rem, marker);
            this.map.addCircle({
                'center': rem.latLng,
                'strokeColor': color,
                'strokeWidth': 1,
                'radius': 100,
                'fillColor': color
            });
        });
    }
}

```

Estás marcas tendrán un `infoWindow` asociado, que a diferencia de la marca que se usa para añadir recordatorios, aparecerá cuando se pulse sobre la marca. En este `infoWindow` aparecerá información sobre el recordatorio.

También hemos modificado las acciones a realizar cuando el mapa este listo para que dibuje los recordatorios y para subscribirse al evento de refresco de datos que implementamos en la clase *ReminderList*. Así, cada vez que se emita este evento, el mapa redibujara los recordatorios.

Con esto, podemos volver a probar la aplicación y comprobar como podemos añadir nuevos recordatorios y ver los ya registrados sobre el mapa, lo cual es recomendable antes de pasar a la siguiente funcionalidad.

3.5.0.7. La lista

Además de en el mapa, el usuario podrá ver los recordatorios en una lista, lo que le facilitará el ver los detalles de estos de un simple vistazo. Utilizaremos el componente de **ion-list** (que ya hemos utilizado para realizar el **Cronómetro**) para crear esta lista. Además, cada uno de los items de esta lista serán del tipo **ion-item-sliding**²⁵. Este tipo de componente permite añadir botones que se encuentran “debajo” de la opción y a los cuales podremos acceder deslizando el elemento en cuestión.

Veamos en primer lugar las modificaciones a realizar en el fichero **HTML** de la página para añadir esta lista:

```
<ion-content padding>
  <ion-list>
    <ion-item-sliding *ngFor="let reminder of
      remindersProvider.reminders.all()" >
      <ion-item>
        <h1> {{ reminder.name }} </h1>
        <p> {{ reminder.description }} </p>
      </ion-item>
      <ion-item-options side="right" >
        <button ion-button color="secondary" (click)="onClickMap(
          reminder)" >
          <ion-icon name="map" ></ion-icon>
          Mapa
        </button>
        <button ion-button color="danger" (click)="onClickRemove(
          reminder)" >
          <ion-icon name="trash" ></ion-icon>
          Eliminar
        </button>
      </ion-item-options>
    </ion-item-sliding>
  </ion-list>
</ion-content>
```

El funcionamiento es simple, se recorre la lista de recordatorios y por cada uno de ellos, se añade un item a la lista. Este item estará formado además de por el item propiamente dicho, en el que aparecen los datos del recordatorio, por dos botones que aparecerán cuando arrastremos el item. Uno de estos botones se usará para eliminar el recordatorio, el otro, para abrir el mapa centrado en el recordatorio en cuestión.

En *ListComponent*, tendremos que implementar los métodos a los que llaman los dos botones anteriormente nombrados.

```
constructor(public navCtrl: NavController, public remindersProvider:
  ReminderProvider,
```

²⁵<http://ionicframework.com/docs/v2/components/#sliding-list>

```

    public events: Events) {}

    onClickRemove(rem: any) {
        this.remindersProvider.reminders.removeReminder(rem);
    }

    onClickMap(rem: any) {
        this.events.publish('centerOnReminder:map', rem);
        this.navCtrl.parent.select(0);
    }

```

En método *onClickRemove* no tiene más misterio que el de llamar al método *removeReminder* de la clase *ReminderList* pasando como parámetro el recordatorio a eliminar.

En cuanto al método *onClickMap*, vemos como hacemos uso de dos *providers* del sistema. El primero de ellos, *Events*, nos permitirá emitir un evento al que, como veremos a continuación, nos deberemos subscribir desde *MapComponent*, seleccionando el nombre del evento (**'centerOnReminder:map'**) y pudiendo pasar cualquier objeto al subscriptor, en este caso el recordatorio en cuestión. El cuando a *NavController*²⁶, se trata del controlador de la navegación a través del cual podremos acceder al componente **tabs** que contiene la página (valor de *parent* dentro del *NavController*), y cambiar de pestaña programáticamente. Ya que el mapa es la primera pestaña, la identificamos con el valor 0.

Por último, volvemos a *MapController* para realizar la subscripción al evento **'centerOnReminder:map'** e implementar la función que centre el mapa sobre el marcador.

```

constructor(public navCtrl: NavController, public platform: Platform
, public remindersProvider: ReminderProvider, public alertController:
AlertController, public events: Events) {
    // RESTO DE LA FUNCIÓN
    events.subscribe('centerOnReminder:map', (rem) =>
        this.centerOnReminder(rem));
}
centerOnReminder(rem: Reminder) {
    // El guardar la el recordatorio y la marca en un Map, nos permite
    realizar búsquedas utilizando el recordatorio.
    let marker = this.markers.get(rem);
    marker.getPosition().then((latLng) => this.map.setCenter(latLng));
    marker.showInfoWindow();
}

```

²⁶<http://ionicframework.com/docs/v2/api/components/tabs/Tabs/>

3.5.0.8. Las notificaciones

Por último, debemos implementar la función principal para la que ha sido pensada esta aplicación, y es la de notificar al usuario cuando esté cerca de un recordatorio.

Para hacer esto, habrá que registrar un nuevo *GeoFence*²⁷ a la hora de crear un recordatorio. Este *GeoFence* se activará cuando el dispositivo entre dentro del área de influencia de un recordatorio y se encargará de lanzar una notificación que aparecerá en la barra de notificaciones como las de cualquier otra aplicación. Este proceso se ejecuta aunque nuestra aplicación esté apagada, y podremos configurar el *GeoFence* para que si el usuario pulsa sobre la notificación, se lance la aplicación. El código de creación y eliminación de estos *GeoFence* formará parte de la clase *Reminder*.

```
createAssociatedGeoFence() {
  let fence = {
    id: this._id.toString(), // Utilizaremos el id del recordatorio
                             // también como id de la notificación para después poder
                             // eliminarla.
    latitude: this._lat,
    longitude: this._lng,
    radius: 100, // Radio en metros de la circunferencia que define el
                 // área de influencia del recordatorio.
    transitionType: 1, // Enter transition
    // Configuración de la notificación. En ella aparecerá la
    // información sobre el recordatorio.
    notification: {
      id: 1,
      title: this._name,
      text: this._description,
      openAppOnClick: true
    }
  }

  Geofence.addOrUpdate(fence).then(
    () => console.log('Geofence added'),
    (err) => console.log(err)
  );
}

removeAssociatedGeoFence() {
  Geofence.remove(this._id.toString()).then(
    () => console.log('Geofence removed'),
    (err) => console.log('Geofence failed to remove')
  );
}
```

Debemos asegurarnos de llamar a estas funciones cada vez que se inserte o se

²⁷<https://ionicframework.com/docs/v2/native/geofence/>

elimine un recordatorio en la **BDD**. Estando seguros de que la operación se ha realizado con éxito para intentar mantener la coherencia entre los *geofences* registrados y los recordatorios guardados.

CONCLUSIONES

Las aplicaciones móviles se han convertido en uno de los sectores más importantes dentro del mundo tecnológico, muestra de ello es la gran comunidad de desarrolladores que existe alrededor de las tecnologías involucradas. Este crecimiento también se ha visto reflejado en las herramientas que disponemos para desarrollar estas aplicaciones. Resulta curioso ver como la gran mayoría de estas herramientas son libres, aún siendo creadas y mantenidas por grandes empresas. Esto es de agradecer, ya que nos permite tener un mayor número de opciones y elegir la que mejor se adapte a nuestras necesidades sin tener que preocuparnos en una posible inversión.

Por otro lado, también hay que destacar la evolución que han sufrido las tecnologías web en estos años, ya sea a través de nuevos estándares, frameworks, lenguajes, ...Por un lado ha permitido mejorar los diseños de las páginas, haciéndolos más atractivos para el usuario, y facilitando el trabajo del diseñador. En este sentido, **CSS3** y sus animaciones, diferentes frameworks como Bootstrap o la definición de nuevas propiedades como los Flex-box han sido los responsables. Por otro, y quizás donde más se ha avanzado, es la parte lógica de las aplicaciones. Se ha pasado de estar limitados por un lenguaje como JavaScript, el cuál no está pensado para aplicaciones complejas, a poder trabajar con características de lenguajes más avanzados (clases, herencias, tipado, ...) y poder utilizar patrones de diseños. Todo esto ha hecho que JavaScript se convierta en una alternativa muy valida no solo para su uso en front-end, si no también para utilizarlo en back-end. Destacar el papel de NodeJS en esta evolución.

En cuanto a las aplicaciones híbridas, el tema sobre el que gira este **PFC**, han encontrado su hueco al calor de la batalla que existe entre las dos grandes plataformas móviles, Android e iOS; y la variedad de dispositivos en un mercado donde a la lucha entre los fabricantes tradicionales se han unido nuevos originarios de China. Estas

aplicaciones se han aprovechado de los avances de las tecnologías web comentadas previamente y han ido evolucionando cada vez más convirtiéndose en una alternativa seria a las aplicaciones nativas. Gracias a este tipo de aplicaciones, y la cantidad de herramientas que han surgido, no hace falta tener unos conocimientos avanzados para crear tu propia aplicación.

Pero no todo son bondades en cuanto a las aplicaciones híbridas, y aún tienen ciertos aspectos que mejorar. Es por ello que cuando se trata de aplicaciones realmente complejas, que requieran de un diseño cuidado o que tengan ciertos requerimientos, por ejemplo, de seguridad, las aplicaciones nativas están un paso por delante siendo la mejor opción siempre que se tengan los recursos necesarios.

Por último añadir que la practicas propuestas en esta memoria podrá ayudar a los estudiantes a entrar en el mundo de la programación aplicaciones híbridas, ya que se empieza desde lo más básico y se va aumentando la dificultad de las aplicaciones gradualmente. Además, se intenta ver el mayor número de características de las tecnologías usadas, dejando en manos de los estudiantes el profundizar más ellas según sus necesidades.

En este **PFC** se ha visto los rasgos más importantes de la tecnología utilizada en aplicaciones híbridas y se ha profundizado en el framework Ionic2, y por tanto, también en Angular2. Lo visto en esta memoria puede servir como base para la realización de aplicaciones usando estas tecnologías.

Tanto Ionic2 como Angular2, son frameworks de gran tamaño con multitud de módulos y características que no se han tratado en esta memoria y que podrían ser usados para futuras prácticas. Una muy interesante podría ser la creación de una aplicación híbrida que actuase como cliente de una aplicación servidora, usando el stack **MEAN** en las diferentes partes del sistema. Este sistema podría consistir únicamente en una gestión de usuarios y perfiles muy simple pero que permitiera mostrar el esqueleto de un sistema de estas características y las herramientas que se ven involucradas, sirviendo de base para la creación de sistemas más complejos.

Si nos centramos solo en el lado del cliente, dentro de Ionic Native nos ofrece diferentes módulos con los que usar las distintas características del dispositivo, como por ejemplo la cámara, con las cuales poder añadir nuevas funciones a las prácticas ya mostradas (incluir los sensores de movimiento en la práctica del paisaje, o el sensor **Near Field Communication (NFC)** en la del mapa de recordatorios) o crear aplicaciones nuevas que aprovechen la potencia de estas.

Por último, hacer el mismo ejercicio que se ha hecho utilizando Ionic2, pero usando alguno de los otros frameworks comentados en la introducción (React Native, Native Script, ...) lo que permitirá comparar diferentes tecnologías permitiendo al lector elegir cual es la que más le conviene.

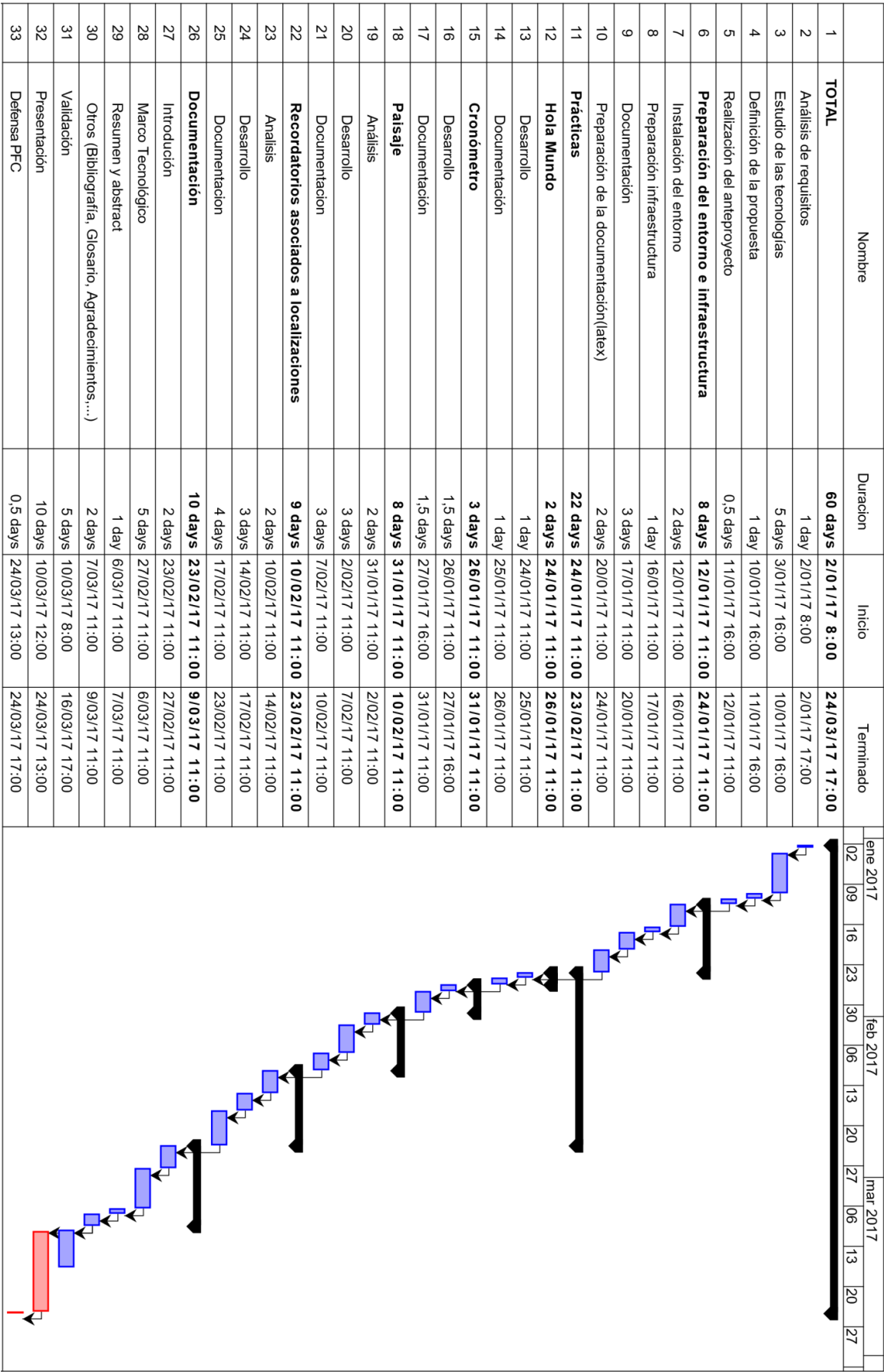
A continuación se hace una estimación de los costes que conlleva la realización de este PFC.

Para ello, en primer lugar se ha confeccionado un calendario con las tareas realizadas. Se ha asumido para la realización de este calendario que:

1. Los trabajos dan comienzo el día 2 de enero de 2017.
2. La dedicación de la persona que desarrolla el trabajo es del 100 % distribuida en una jornada laboral de 8 a 12 y de 13 a 17 horas.
3. La ausencia de días festivos. Solo se tienen en cuenta los fines de semana.
4. Se da una situación ideal sin interrupciones de ningún tipo de por medio, pudiendo realizar los trabajos uno detrás de otro.

Se considera también que la persona encargada de realizar estos trabajos tiene el nivel de ingeniero junior. La tarifa que se aplica a esta persona es de 35 euros la hora sin IVA.

En cuanto a la infraestructura, deberá estar disponible durante los 6 meses posteriores al inicio del proyecto. Este coste también será tenido en cuenta en el presupuesto.



| PAQUETES DE TRABAJO | | | | |
|---------------------|---|-----------------------|----------------|-------------------|
| | Nombre | Nombres del Recurso | Duración | Costo |
| 1 | TOTAL | | 60 days | 15960,00 € |
| 2 | Análisis de requisitos | Ingeniero Junior | 1 day | 280,00 € |
| 3 | Estudio de las tecnologías | Ingeniero Junior | 5 days | 1400,00 € |
| 4 | Definición de la propuesta | Ingeniero Junior | 1 day | 280,00 € |
| 5 | Realización del anteproyecto | Ingeniero Junior | 0,5 days | 140,00 € |
| 6 | Preparación del entorno e infraestructura | | 8 days | 2240,00 € |
| 7 | Instalación del entorno | Ingeniero Junior | 2 days | 560,00 € |
| 8 | Preparación infraestructura | Ingeniero Junior | 1 day | 280,00 € |
| 9 | Documentación | Ingeniero Junior | 3 days | 840,00 € |
| 10 | Preparación de la documentación(latex) | Ingeniero Junior | 2 days | 560,00 € |
| 11 | Prácticas | | 22 days | 6160,00 € |
| 12 | Hola Mundo | | 2 days | 560,00 € |
| 13 | Desarrollo | Ingeniero Junior | 1 day | 280,00 € |
| 14 | Documentación | Ingeniero Junior | 1 day | 280,00 € |
| 15 | Cronómetro | | 3 days | 840,00 € |
| 16 | Desarrollo | Ingeniero Junior | 1,5 days | 420,00 € |
| 17 | Documentación | Ingeniero Junior | 1,5 days | 420,00 € |
| 18 | Paisaje | | 8 days | 2240,00 € |
| 19 | Análisis | Ingeniero Junior | 2 days | 560,00 € |
| 20 | Desarrollo | Ingeniero Junior | 3 days | 840,00 € |
| 21 | Documentación | Ingeniero Junior | 3 days | 840,00 € |
| 22 | Recordatorios asociados a localizaciones | | 9 days | 2520,00 € |
| 23 | Análisis | Ingeniero Junior | 2 days | 560,00 € |
| 24 | Desarrollo | Ingeniero Junior | 3 days | 840,00 € |
| 25 | Documentación | Ingeniero Junior | 4 days | 1120,00 € |
| 26 | Documentación | | 10 days | 2800,00 € |
| 27 | Introducción | Ingeniero Junior | 2 days | 560,00 € |
| 28 | Marco Tecnológico | Ingeniero Junior | 5 days | 1400,00 € |
| 29 | Resumen y abstract | Ingeniero Junior | 1 day | 280,00 € |
| 30 | Otros (Bibliografía, Glosario, Agradecimientos,...) | Ingeniero Junior | 2 days | 560,00 € |
| 31 | Validación | Ingeniero Junior[20%] | 5 days | 280,00 € |
| 32 | Presentación | Ingeniero Junior[80%] | 10 days | 2240,00 € |
| 33 | Defensa PFC | Ingeniero Junior | 0,5 days | 140,00 € |

| INFRAESTRUCTURA | | | | |
|-----------------|--------------------|------------------|----------|-------|
| | Nombre | Coste por unidad | Duración | Costo |
| 1 | Repositorio GitHub | 7 € / mes | 6 meses | 42 € |

| CONCEPTO | TOTAL SIN IVA | TOTAL CON IVA |
|---------------------|----------------|-------------------|
| Paquetes de trabajo | 15960 € | 19311.6 € |
| Infraestructura | 42 € | 50.82 € |
| TOTAL | 16002 € | 19362.42 € |

- [Ala15] José Manuel Alarcón. *Qué es el stack MEAN y cómo escoger el mejor para ti*. 2015. URL: <https://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-para-ti.aspx> (visitado 09-04-2017).
- [Ama17] Ron Amadeo. *Google's "Fuchsia" smartphone OS dumps Linux, has a wild new UI*. 2017. URL: <https://arstechnica.com/gadgets/2017/05/googles-fuchsia-smartphone-os-dumps-linux-has-a-wild-new-ui/> (visitado 10-05-2017).
- [Aro16] Sunil Arora. *10 Best Hybrid Mobile App UI Frameworks: HTML5, CSS and JS*. 2016. URL: <http://noeticforce.com/best-hybrid-mobile-app-ui-frameworks-html5-js-css> (visitado 09-04-2017).
- [Aza15] Carlos Azaustre. *Empezando con ReactJS y ECMAScript 6*. 2015. URL: <https://carlosazaustre.es/blog/empezando-con-react-js-y-ecmascript-6/> (visitado 09-04-2017).
- [COD16] CODEANDYOU. *Difference between component and directive in Angular 2 ?* 2016. URL: <http://www.codeandyou.com/2016/01/difference-between-component-and-directive-in-Angular2.html> (visitado 09-04-2017).
- [Dil16] Daniel Eran Dilger. *Apple's iOS App Store now generating 4x revenues per app vs Android Google Play*. 2016. URL: <http://appleinsider.com/articles/16/07/19/apples-ios-app-store-now-generating-4x-revenues-per-app-vs-android-google-play> (visitado 09-04-2017).
- [Dog16] Artyom Dogtiev. *App Store Statistics Roundup*. 2016. URL: <http://www.businessofapps.com/app-store-statistics-roundup/> (visitado 22-05-2017).
- [Gar17] Gartner. *Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016*. 2017. URL: <http://www.gartner.com/newsroom/id/3609817> (visitado 22-05-2017).

- [Jac16] Brian Jackson. *Bootstrap vs Foundation – Top 2 CSS Frameworks*. 2016. URL: <https://www.keycdn.com/blog/bootstrap-vs-foundation/> (visitado 09-04-2017).
- [Mar15] Danny Markov. *Comparing The Top Frameworks For Building Hybrid Mobile Apps*. 2015. URL: <http://tutorialzine.com/2015/10/comparing-the-top-frameworks-for-building-hybrid-mobile-apps/> (visitado 09-04-2017).
- [mar16] marketingdirecto. *La eterna (y sangrienta) guerra entre sistemas operativos: Windows vs Android vs Apple*. 2016. URL: <https://www.marketingdirecto.com/digital-general/digital/eterna-sangrienta-guerra-sistemas-operativos-windows-vs-android-vs-apple> (visitado 22-05-2017).
- [Nad13] Ben Nadel. *My Experience With AngularJS - The Super-heroic JavaScript MVW Framework*. 2013. URL: <https://www.bennadel.com/blog/2439-my-experience-with-angularjs---the-super-heroic-javascript-mvw-framework.htm> (visitado 09-04-2017).
- [Ori16] Enrique Oriol. *Introducción a Angular 2 (parte I) – Modulo, Componente, Template y Metadatos*. 2016. URL: <http://blog.enriqueoriol.com/2016/06/introduccion-a-angular-2-parte-i-componente.html> (visitado 09-04-2017).
- [Pri17] Ricardo Prieto. *Mejores Frameworks CSS para diseño web responsive*. 2017. URL: <https://www.silocreativo.com/mejores-frameworks-css/> (visitado 09-04-2017).
- [Sha15] Uri Shaked. *AngularJS vs. Backbone.js vs. Ember.js*. 2015. URL: <https://www.airpair.com/js/javascript-framework-comparison> (visitado 09-04-2017).
- [Sha16] C S Sharif. *What is Angular 2? High level overview, feature and fundamentals*. 2016. URL: <https://www.technicaldiary.com/angular-2-angularjs-overview-feature-fundamentals/> (visitado 09-04-2017).

ANEXOS

ANEXOS A

HERRAMIENTAS PARA TRABAJAR CON ANDROID

Para desarrollar una aplicación, debemos de contar con las herramientas necesarias para hacerlo, y esto no es una excepción cuando trabajamos con Android. Las herramientas más habituales son:

1. Android **Software Developer Kit (SDK)**: Es un conjunto de herramientas para el desarrollo de aplicaciones en Android. Comprende un depurador de código, biblioteca, emulador de terminales, documentación, ejemplos y tutoriales.
2. **Android Debugger Bridge (ADB)**: Aplicación incluida dentro del **SDK** compuesta por tres componentes: un cliente y un servidor que se ejecutan en la máquina de desarrollo, y un daemon que se ejecuta en el termino. Esta aplicación nos permite la comunicación ya sea a través de línea de comandos (comando adb) o a través de una interfaz gráfica (consola de depuración de Google Chrome) con los dispositivos Android conectados vía **USB** o que están siendo emulados.
3. **Android Virtual Device (AVD)**: Herramienta para emular dispositivos Android basada en *qemu*¹
4. Android Studio: Se trata de un **Integrated Development Environment (IDE)** basado en IntelliJ y que vino a sustituir a Eclipse como **IDE** oficial a la hora de programar aplicaciones para Android.

Para instalar estas herramientas tenemos dos opciones. Instalar tanto Android Studio como Android SDK, o descargarnos las herramientas que conforman Android SDK en caso de querer trabajar con un IDE diferente. En nuestro caso, vamos a

¹<http://www.qemu-project.org/>

instalar solamente Android SDK, ya que para trabajar con tecnologías web existen alternativas mejores que Android Studio.

A.1. Instalación de Android SDK

Android SDK es un conjunto de herramientas que podemos descargar en forma de fichero comprimido, por lo que más que instalar, lo que haremos será descargarnos este fichero y descomprimirlo en nuestro ordenador. Antes de ver como poder descargárnoslo y explicar algunos de sus componentes y como usarlos, vamos a explicar como instalar el **Java Development Kit**, software necesario para poder utilizar las herramientas de Android.

1. Accedemos a la página de descarga de Java, propiedad de Oracle, <http://www.oracle.com/technetwork/java/javase/downloads/index.html>, donde veremos la opción de para descargarnos el **JDK**.
2. Deberemos aceptar los términos de uso y elegir la versión que necesitamos antes de empezar con la descarga.

| Java SE Development Kit 8u121 | | |
|---|-----------|---|
| You must accept the Oracle Binary Code License Agreement for Java SE to download this software. | | |
| <input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement | | |
| Product / File Description | File Size | Download |
| Linux ARM 32 Hard Float ABI | 77.86 MB | jdk-8u121-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 74.83 MB | jdk-8u121-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 162.41 MB | jdk-8u121-linux-i586.rpm |
| Linux x86 | 177.13 MB | jdk-8u121-linux-i586.tar.gz |
| Linux x64 | 159.96 MB | jdk-8u121-linux-x64.rpm |
| Linux x64 | 174.76 MB | jdk-8u121-linux-x64.tar.gz |
| Mac OS X | 223.21 MB | jdk-8u121-macosx-x64.dmg |
| Solaris SPARC 64-bit | 139.64 MB | jdk-8u121-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 99.07 MB | jdk-8u121-solaris-sparcv9.tar.gz |
| Solaris x64 | 140.42 MB | jdk-8u121-solaris-x64.tar.Z |
| Solaris x64 | 96.9 MB | jdk-8u121-solaris-x64.tar.gz |
| Windows x86 | 189.36 MB | jdk-8u121-windows-i586.exe |
| Windows x64 | 195.51 MB | jdk-8u121-windows-x64.exe |

Figura A.1: **Java Development Kit** está disponible para diferentes sistemas operativos y arquitecturas.

3. Es importante durante la instalación ver la localización donde se va a realizar, ya que puede ser necesario indicar esta ruta de instalación por si nos la requiere el SDK de Android.

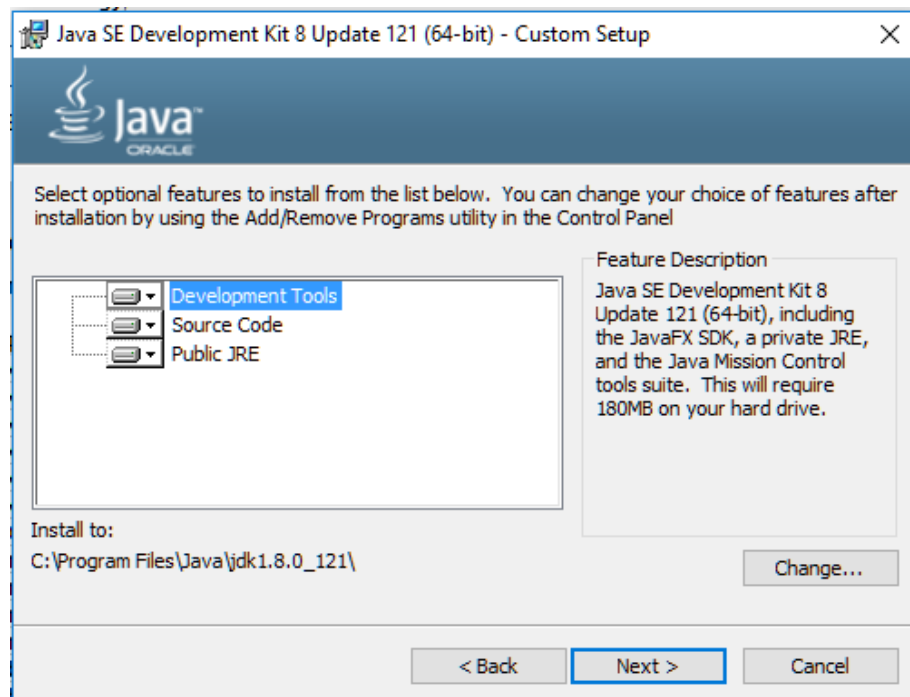


Figura A.2: Durante la instalación podremos cambiar la ruta de instalación del **JDK**, aunque se recomienda no modificarla si no es estrictamente necesario.

4. Lo mismo ocurre con la ruta de instalación de **Java Runtime Enviroment**.

Si todo ha ido bien, podremos ejecutar el comando de Java desde consola.

```
workspace> java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
workspace>
```

Figura A.3: Podemos comprobar desde la consola la versión del **JDK** que tenemos instalada.

Con el **JDK** ya disponible en nuestro equipo, podemos descargarnos el **SDK** de Android desde la página de desarrolladores que ofrece Google, <https://developer.android.com/studio/index.html#downloads>. En esta página podremos encontrar el instalador de Android Studio (opción mas destacada) y el **SDK** de manera individual. De ambos existen versiones para diferentes sistemas.

Obtener solo las herramientas de línea de comando

Si no necesitas Android Studio, puedes descargar las siguientes herramientas básicas de línea de comando de Android.

| Plataforma | Paquete de SDK Tools | Tamaño | Suma de comprobación de SHA-1 |
|------------|---|-------------------------------|--|
| Windows | tools_r25.2.3-windows.zip | 292 MB (306,745,639 bytes) | 23d5686ffe489e5a1af95253b153ce9d6f933e5dbabe14c494631234697a0e08 |
| Mac | tools_r25.2.3-macosx.zip | 191 MB (200,496,727 bytes) | 593544d4ca7ab162705d0032fb0c0c88e75bd0f42412d09a1e8daa3394681dc6 |
| Linux | tools_r25.2.3-linux.zip | 264 MB (277,861,433 bytes) | 1b35bcb94e9a686dff6460c8bca903aa0281c6696001067f34ec00093145b560 |

Consulta las [notas de la versión de SDK Tools](#).

*Figura A.4: Aunque la opción está un poco “escondida”, podemos descargarnos el **SDK** de Android sin necesidad de instalar Android Studio.*

Cuando se haya completado la descarga, podremos descomprimir el fichero en el directorio que queramos y poder ver los componentes que forman Android **SDK**.

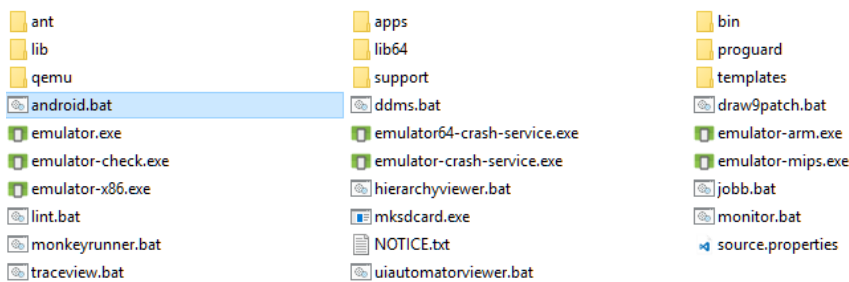


Figura A.5: Ficheros que se encuentran dentro del paquete comprimido.

A.2. Android SDK Manager

El paquete de Android SDK cuenta *Android SDK Manager*, que nos permite instalar las herramientas que ofrece el SDK, las plataformas y otros componentes de manera independiente. Una vez iniciado veremos una ventana como la siguiente:

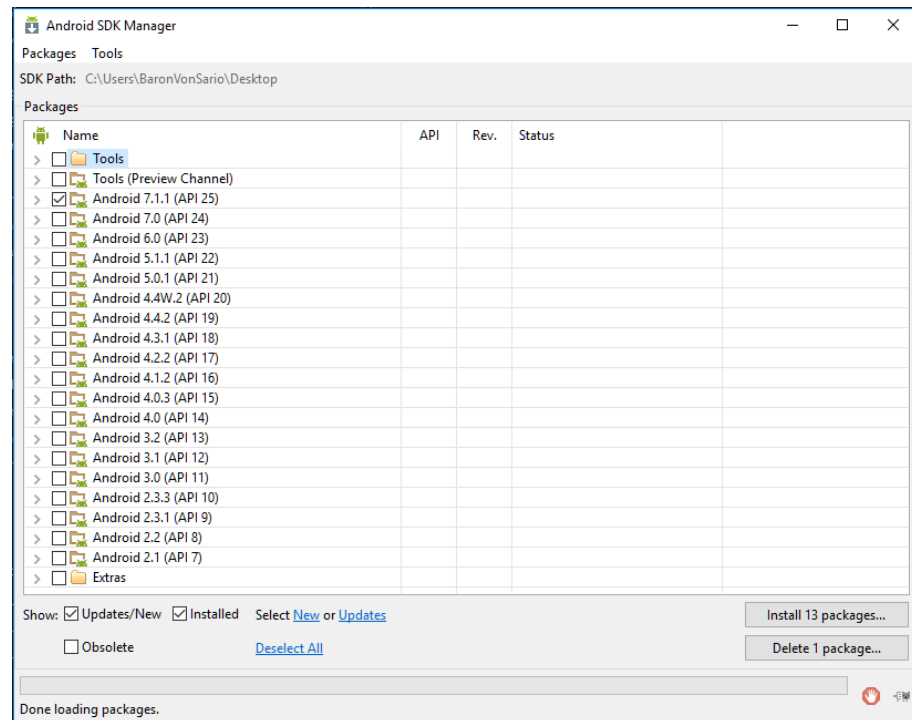


Figura A.6: Android SDK Manager nos permite instalar los diferentes componentes de Android SDK por separado.

Los componentes aparecen agrupados en diferentes grupos:

Tools Aquí se encuentran varias herramientas indispensables para poder crear una aplicación para Android. Es recomendable instalar la última versión de aquellas que aparezcan. Los paquetes herramientas que nos encontramos son **Android SDK tools**, que contiene elementos como el emulador, **Android SDK Platform-tools**, donde se incluye **Android Debugger Bridge (ADB)** y **Android SDK Build-tools**, con las herramientas que compilan y generan la aplicación.

Android SDK Platform Diferentes versiones de la plataforma Android. Al menos tendremos que tener una de ellas disponible, que bien puede ser la última, lo cuál es lo más recomendable, o bien una anterior, según los requisitos del desarrollo. Aquí también se encuentran las imágenes del sistema que podremos ejecutar en el emulador.

Extras Software que no se engloba en ninguno de los otros grupos pero que pueden ser de utilidad durante el desarrollo de nuestro proyecto. Aquí se podemos encontrarnos con los drivers **Google USB**, que dependiendo del dispositivo que tengamos, necesitaremos instalar para poder conectarlo a nuestro **Personal**

Computer (PC); el repositorio de Google, con diferentes bibliotecas; o Google Play services.

A.3. Android Virtual Device

Llamamos **Android Virtual Device (AVD)**² a la definición de características de un dispositivo Android, ya sea un teléfono, una tablet, un smartwatch, Entre otras propiedades, se encuentra el hardware del dispositivo, la imagen del sistema o el espacio de almacenamiento asociado. Los AVD pueden ser emulados usando **Android Emulator**.

Para gestionar nuestros AVDs disponemos de un manager al que podemos acceder desde el **Android SDK Manager**.

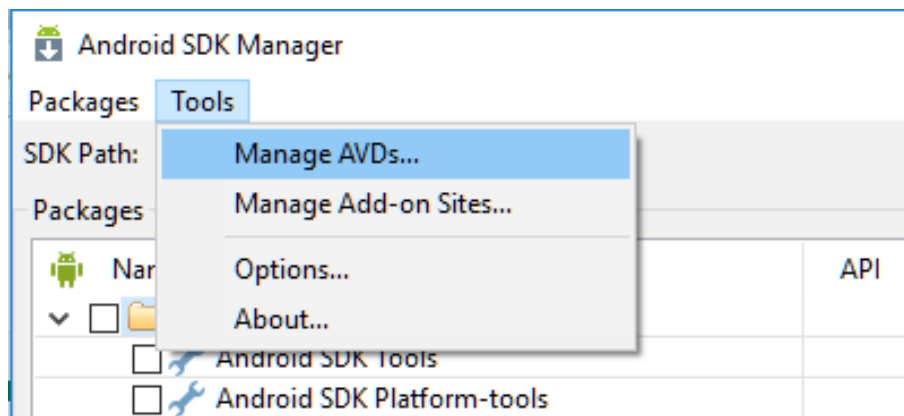
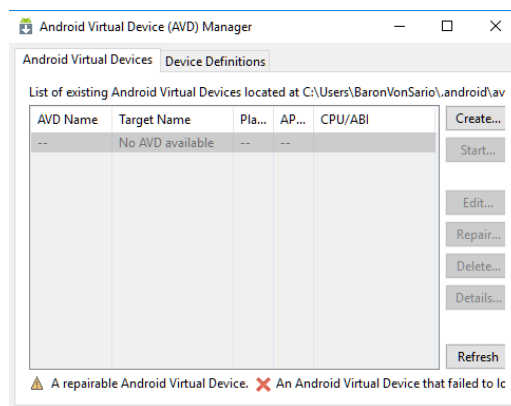


Figura A.7: Desde el propio manager de Android SDK podremos abrir el manager de avds.

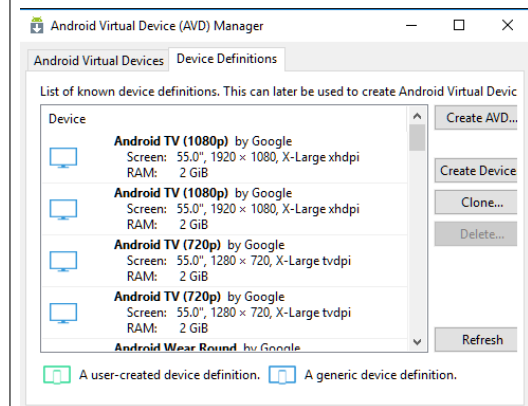
Esto nos abre una ventana como la siguiente en la que podemos ver dos pestañas. En una aparecerá una lista con los AVDs disponibles, en la otra, una lista de definiciones de dispositivos.

En el listado de dispositivos podemos ver algunos que vienen por defecto, entre los que encontramos terminales de la familia Nexus, Android TV, smartwatches y tablets. Con estos dispositivos debería ser más que suficiente, pero si queremos, podemos crear los nuestros propios desde cero, o clonando alguno de los ya existentes, y así poder personalizar al gusto las características del hardware.

²<https://developer.android.com/studio/run/managing-avds.html>



(a) Lista de AVDs



(b) Lista de definiciones de dispositivos

Figura A.8: AVDs Manager.

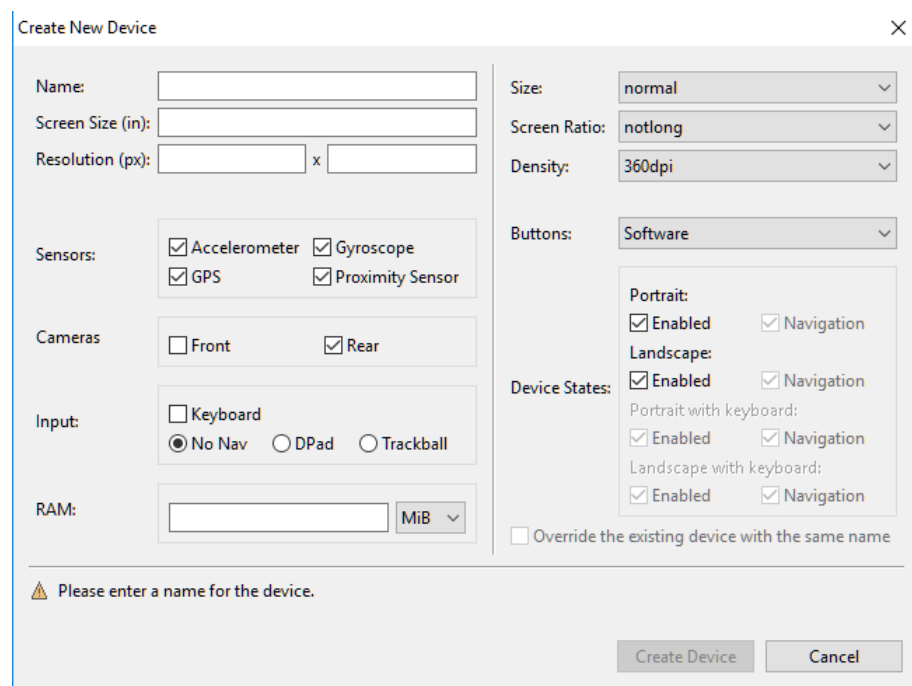


Figura A.9: A la hora de crear un nuevo dispositivo, podemos configurar varias características de su hardware.

Por otro lado, la lista de **AVDs** se encuentra vacía ya que no viene ninguna configuración establecida por defecto. Para crear la nuestra y poder emularla, pulsamos el botón **Create AVD**. Nos abrirá una ventana donde podremos configurar nuestro **AVD**.

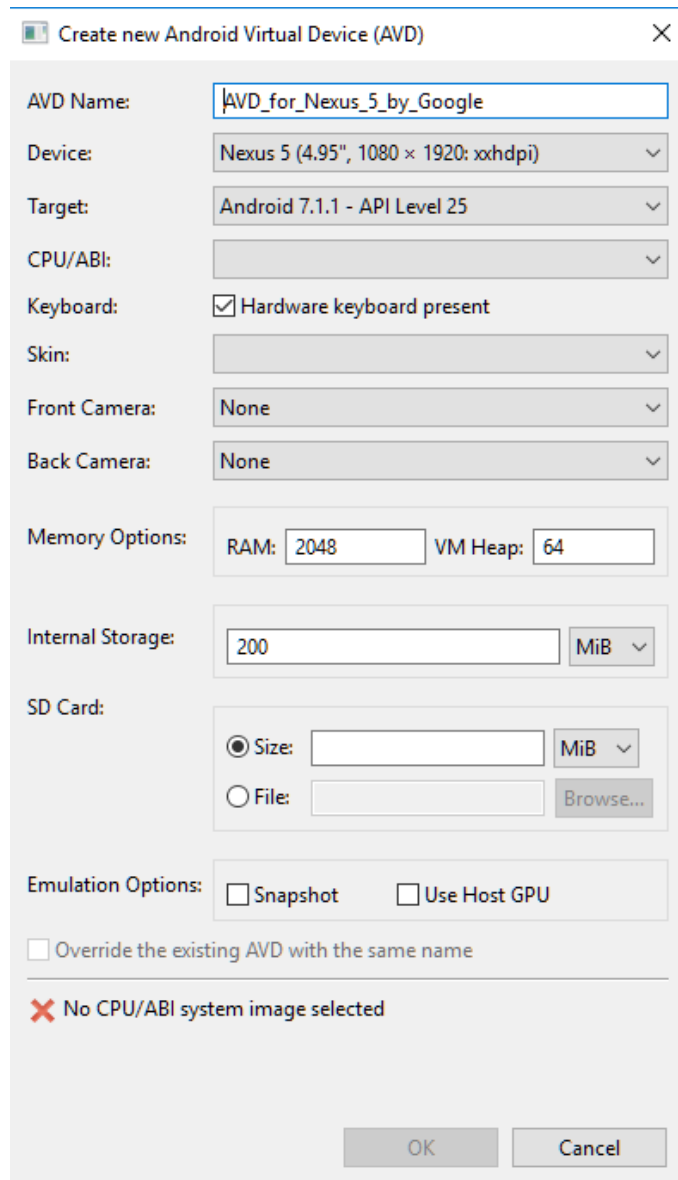


Figura A.10: Vista con la que podemos crear nuestro **AVD**.

Los campos que debemos rellenar en este paso son:

1. AVD Name: Nombre con el que identificar nuestro **AVD**, tanto dentro del manager, como fuera. Este nombre será el que se utilice para identificar el emulador en herramientas externas, como Ionic CLI cuando se usa la opción *emulate*.
2. Device: Definición del dispositivo que queremos que se use. En esta lista aparecerán los dispositivos por defecto y los que hayamos creado nosotros.
3. Target: Versión de la plataforma Android que usará. Solo aparecerán aquellas que se hayan descargado previamente desde el **SDK** manager.

4. CPU/ABI: Imagen del sistema. También tendrá que ser descargado con anterioridad junto con la versión de la plataforma desde el **SDK** manager.
5. Keyboard: Si habilitamos esta opción, el emulador entenderá que el dispositivo dispone de un teclado físico por lo que no se mostrará el teclado virtual.
6. Skin: Aspecto que debe utilizar el dispositivo y que definirá el tamaño con el que se verán los elementos de la aplicación, como los botones.
7. Front/Back Camera: Cámaras que tendrá el dispositivo. En caso de disponer, la imagen puede ser emulada, o utilizar una webcam conectada a nuestra máquina.
8. Memory Option: Memoria RAM que utilizará el emulador. Debemos de configurar esta cantidad teniendo en cuenta la memoria RAM disponible en la máquina en la cual vayamos a ejecutar el emulador.
9. Internal Storage y SD Card: Capacidad de almacenamiento que tendrá el dispositivo emulado y que podremos definir por separado según se trate del propio almacenamiento del sistema, o corresponde a una memoria extraíble.
10. Emulation Option: Estas opciones tienen que ver con el rendimiento del emulador. La opción *Snapshot* crea una “copia” del sistema emulado en la memoria RAM lo que acelera un posible reinicio de la aplicación. La opción *Use Host GPU* como se puede uno imaginar, hace que el emulador haga uso de la tarjeta gráfica de la máquina que lo ejecuta, acelerando las operaciones relacionadas con la visualización.

A.4. Android Emulator

El emulador de Android³ que ofrece Android **SDK**, basado en **qemu**⁴, nos permite emular nuestros **AVDs** y ejecutar dentro las aplicaciones Android que desarrollamos.

El emulador nos permite además, simular diferentes características de las que dispone el dispositivo real y con los que no contamos en la máquina de desarrollo. Un ejemplo son las cámaras con las que cuentan la mayoría de dispositivos, y que como vimos en la creación de **AVDs**, podemos simular.

³<https://developer.android.com/studio/run/emulator.html>

⁴<http://www.qemu-project.org/>

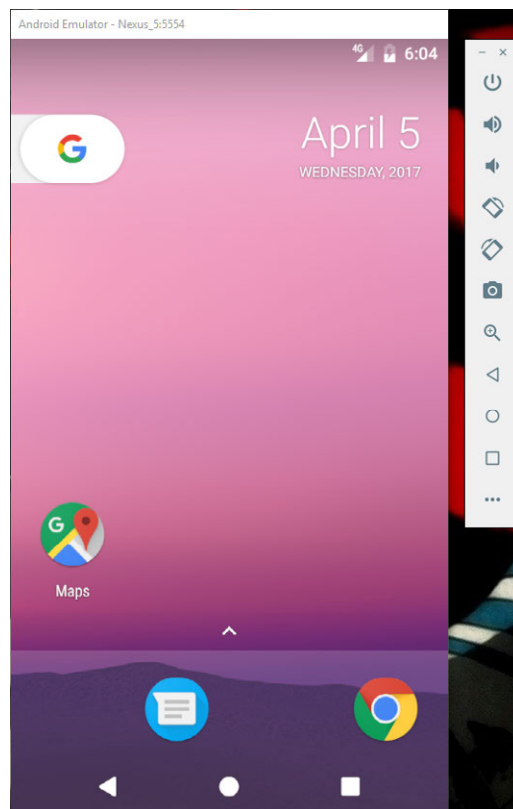


Figura A.11: Emulador de Android, con la pantalla a la izquierda y la botonera a la derecha.

Al ejecutar el emulador nos aparece por un lado una ventana que simula la pantalla del dispositivo, y por otro una barra vertical con botones. La pantalla actúa como la de un dispositivo real en la que el puntero del ratón hace de dedo. El gesto de pellizco, en el cual se usan dos dedos, se puede simular pulsando la tecla Ctrl o Command (). Además, si arrastramos nuestra **APKs** sobre esta pantalla, el emulador instalará la aplicación en el simulador; en caso de arrastrar otro tipo de archivos, este se almacenará como un fichero dentro del sistema emulado.

En la barra vertical contamos con diferentes funciones típicas de un terminal móvil y que se suelen reflejar mediante un botón físico o software en los terminales reales. Ejemplo de esto es el botón de apagado, los de volumen, la botonera inferior característica de Android (que según el modelo de dispositivo, son implementados vía software o mediante botones físicos),

Al final de la barra vertical se encuentra un botón con el cual podremos abrir los controles extendidos.

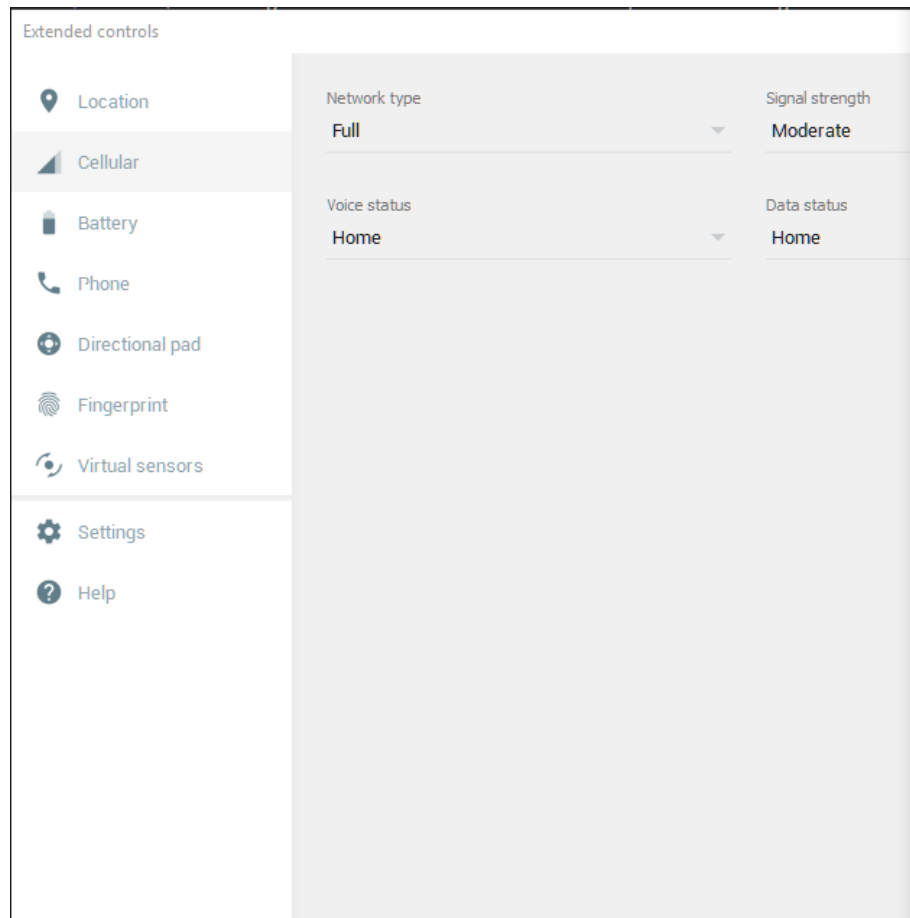


Figura A.12: Ventana de controles extendidos del emulador.

Desde esta ventana podremos simular diferentes sensores, llamadas, estado de la batería,

ANEXOS B

INSTALAR NODE.JS Y NPM

Desde la versión 0.6.3 de Node.JS, el gestor de paquetes **npm** se instala junto al mencionado entorno de ejecución. En esta sección, se explicarán los pasos a seguir para intalar Node.JS en una máquina con sistema operativo Windows.

1. Lo primero que debemos hacer es descargar el instalador de Node.JS. Esto lo podemos hacer desde la web oficial de Node.JS (<https://nodejs.org/es/download/>). Podremos escoger entre descargarnos la versión **Long Term Support (LTS)**, que en el momento de escribir este documento se corresponde a la versión 6.9.2, que incluye la versión 3.10.9 de **npm**; o la versión mas actualizada, Node.JS versión 7.2.0 junto con la 3.10.9 de **npm**. Para este manual, hemos escogido la versión mas actual (7.2.0) para Windows x64 en su formato binario (.exe).
2. Una vez descargado el binario, lo ejecutamos apareciéndonos la bienvenida al instalador de node.JS.

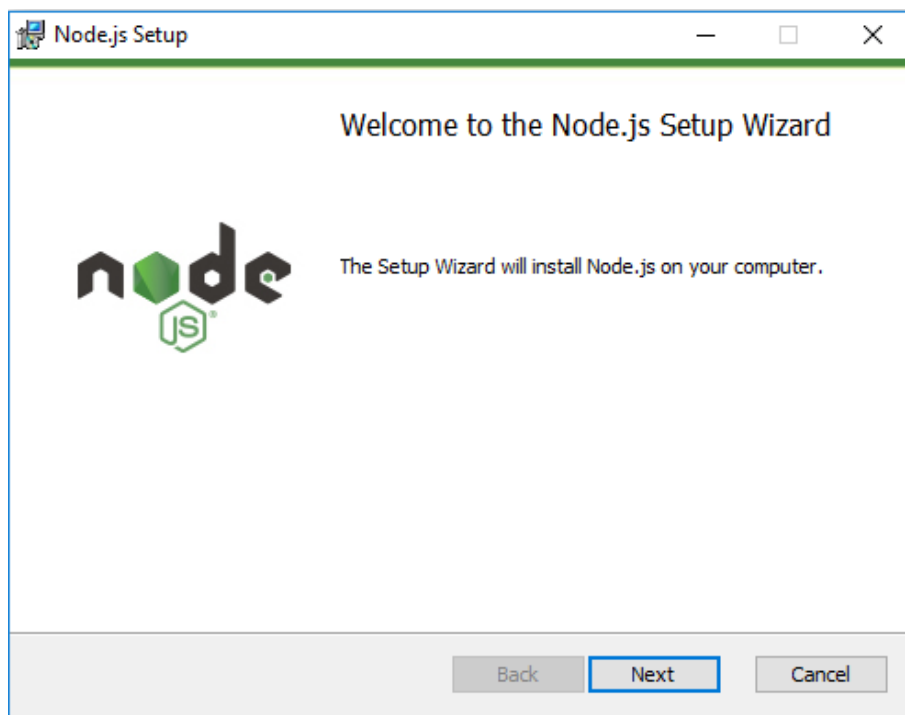


Figura B.1: Paso de bienvenida del instalador.

3. Pulsamos el botón siguiente, lo que nos llevará a una pantalla con los términos de uso.

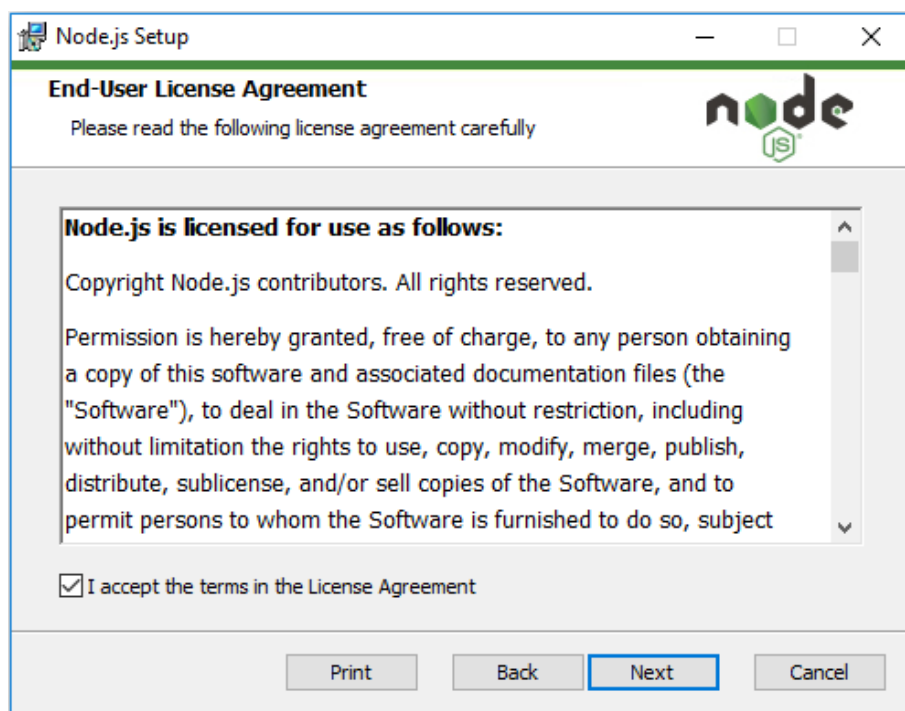


Figura B.2: Terminos de uso.

4. En el siguiente paso nos permitirá seleccionar la ruta de instalación. Bajo esta ruta se instalarán los scripts ejecutables, por lo que será necesaria recordarla en caso de tener que configurar manualmente las variables de entorno de Windows.

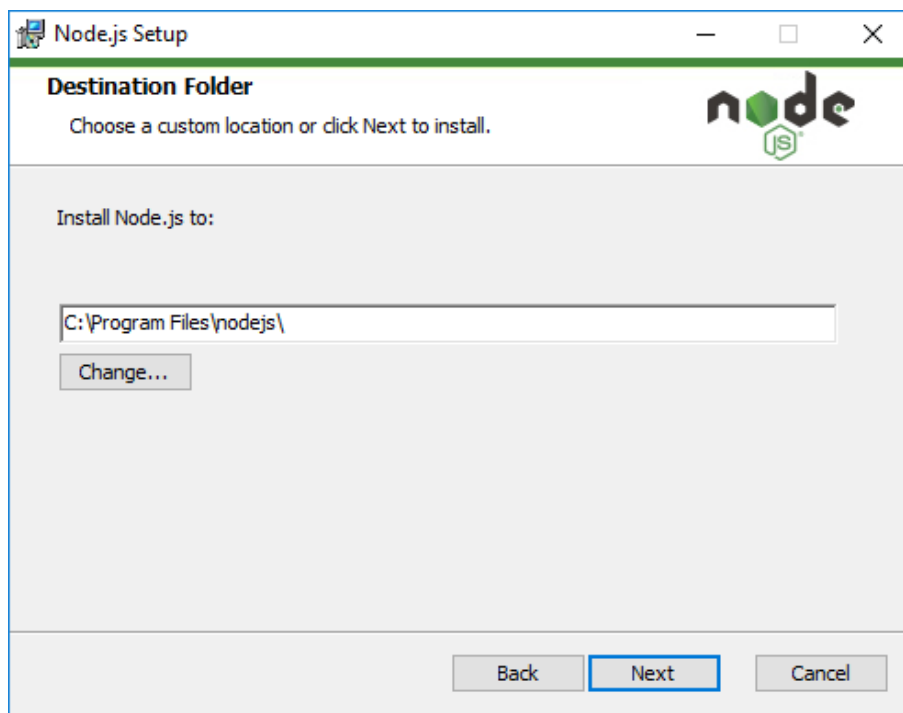


Figura B.3: Configuración del directorio de instalación de node.JS.

5. Por último nos aparecerá una ventana en la que ya iniciar el proceso de instalación. Será necesario otorgarle permisos de administrador para que pueda realizar la instalación.

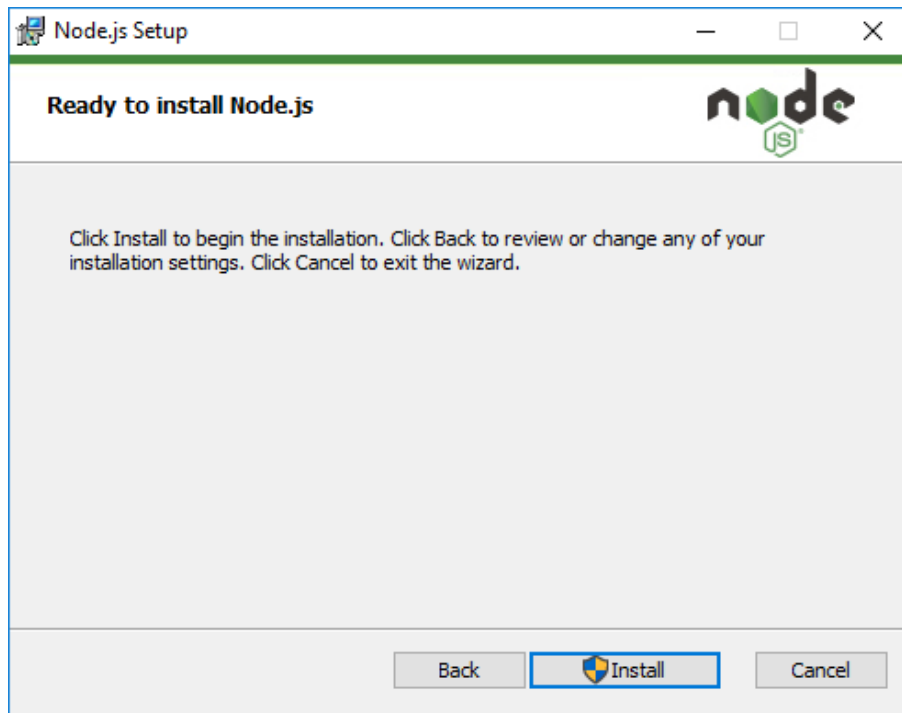


Figura B.4: Paso final.

Una vez completada la instalación, podremos hacer uso tanto de Node.JS como de npm a través de Windows PowerShell (o de la consola de comandos) o de programas de terceros que hagan uso de estas herramientas. Para comprobar que la instalación se ha realizado correctamente, desde Windows PowerShell comprobamos la versión de los programas instalados y ejecutamos un pequeño código JavaScript.

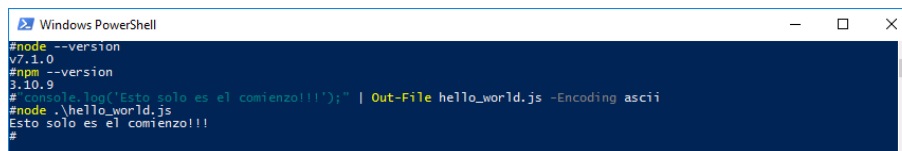


Figura B.5: La instalación se ha realizado correctamente.

Si el sistema no es capaz de reconocer los comando node o npm, tendremos que chequear que el directorio en el que se realizó la instalación se encuentra configurado en las variables de entorno de Windows. Desde Windows PowerShell se pueden ver estas variables con el siguiente comando:

```
# Get-ChildItem Env:
```

o para ver en más detalle la variable PATH:

```
# $Env:PATH
```


Si vemos que el directorio de instalación no aparece, tendremos que configurarlo manualmente:

```
# \ $env:Path += ";C:\ path \ to \ node "
```


A la hora de decidir que editor utilizar, influye los gustos de cada uno. Actualmente existe un catálogo enorme de editores, tanto gratuitos como de pago, los cuales ofrecen la mayoría de funcionalidades básicas que podemos esperar de un editor. La propia página de Ionic ¹ nos recomiendan los siguientes basándose en el soporte que ofrecen a la hora de trabajar con este framework:

- Visual Studio Code
- Atom
- WebStorm
- ALM
- Angular IDE by Webclipse



Es importante también a la hora de elegir un editor, como se adapta este al resto de lenguajes que utilicemos, sin tener que tener abierto un editor por cada lenguaje y poder manejar con solo uno todo el proyecto.

Para la realización de este PFC vamos he escogido como editor Atom. Vengo usando este editor desde hace ya un tiempo no solo para programación en HTML y JS, si no también para otros lenguajes como Python o SQL y escribir textos con L^AT_EX (como por ejemplo esta memoria). Atom es un editor de código abierto para Windows, Linux y MacOS desarrollado por GitHub (el código fuente se puede

¹https://ionicframework.com/docs/v2/resources/editors_and_ides/

descargar desde su repositorio ². Aún siendo una aplicación de escritorio, esta desarrollado utilizando tecnologías web, más concretamente Chromium y NodeJS. Esto da lugar a uno sus puntos fuertes, que es la personalización de la aplicación; como muy bien indican en su página, Atom es: A hackable text editor for the 21st Century.

C.1. Instalación

Como hemos comentado, Atom se encuentra disponible para los sistemas Windows, Linux y MacOS. Desde su propio repositorio en GitHub podemos descargarnos cualquiera de las versiones disponibles para cada uno de los sistemas antes mencionados (<https://github.com/atom/atom/tags>).

C.2. Personalización

Como ya se ha comentado, uno de los puntos fuertes de Atom es la personalización, permitiendo al propio usuario tener su propia configuración o utilizar recursos creados por terceros.

C.2.1. Paquetes y temas

Una característica habitual de las aplicaciones de código abierto es permitir a los usuarios crear y compartir modificaciones de la aplicación, y como no, Atom no podía ser menos. Estas modificaciones se agrupan en dos tipos:

- Los **paquetes (packages)**, los cuales añaden nuevas funcionalidades al editor,
- y los **temas (themes)**, que cambian el aspecto visual de la aplicación.

Para ambos existe un listado donde encontrarlos en la propia web de Atom (<https://atom.io/packages> y <https://atom.io/themes>), o más fácil aún, desde la propia configuración de la aplicación (File >Settings >Install).

²<https://github.com/atom/atom>

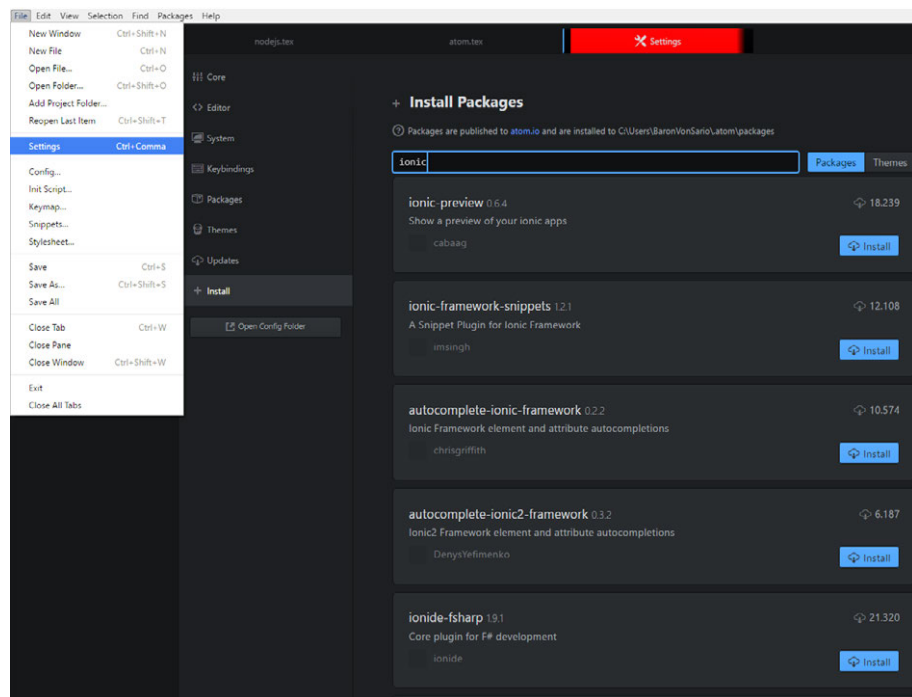


Figura C.1: Pantalla desde donde instalar paquetes y temas desde la propia aplicación de atom.

Al tratarse de un paquete nodeJS, Ionic puede ser instalado usando **npm** de una manera muy sencilla. A continuación detallaremos los pasos a seguir, asumiendo que en la máquina en la que estamos trabajando se encuentra instalado nodeJS y npm (ver anexo).

- En primer lugar instalaremos Cordova, usando también para ello **npm**. Para ello ejecutaremos el siguiente comando.

```
# npm install -g cordova
```

El proceso puede tardar unos minutos ya que tendrá que instalar varias dependencias.

- Una vez instalado Cordoba, procedemos a la instalación de ionic.

```
# npm install -g ionic
```



Existen dos maneras de instalar paquetes con **npm**, de manera global o de manera local. Si queremos utilizar el paquete como una herramienta vía línea de comandos, debemos instalarla de manera global; en cambio, si queremos que el paquete sea una dependencia del modulo que estamos programando, debemos instalarlo de manera local. La opción *-global/-g* indica a **npm** que el paquete debe ser instalado de manera global ya que por defecto lo instalará de manera local.

Para probar que se ha instalado de manera correcta, probaremos crearemos un nuevo proyecto y probaremos que funciona.

- Dentro de nuestro área de trabajo, crearemos un nuevo proyecto utilizando para ello el comando *ionic start* seguido del nombre de nuestro proyecto.

```
# ionic start --v2
```

- El anterior comando creará un nuevo directorio con el mismo nombre del proyecto. Dentro de este directorio descargará una template por defecto, por lo que dentro encontraremos diferentes ficheros y directorios:

```
#ls
Directorio: C:\Users\BaronVonSario\Google Drive\PFC\workspace\TestIonic

Mode                LastWriteTime         Length Name
----                -
d-r---           29/12/2016    20:28             hooks
d-r---           29/12/2016    20:29      node_modules
d-r---           29/12/2016    22:02       plugins
d-r---           29/12/2016    20:28         scss
d-r---           29/12/2016    20:29         www
-a----           29/12/2016    20:28          29 .bowerrc
-a----           29/12/2016    20:28       242 .editorconfig
-a----           29/12/2016    20:28       138 .gitignore
-a----           29/12/2016    20:28       118 bower.json
-a----           29/12/2016    20:29       962 config.xml
-a----           29/12/2016    20:28      1395 gulpfile.js
-a----           29/12/2016    20:29        42 ionic.config.json
-a----           29/12/2016    20:29       602 package.json
```

Figura D.1: Estructura de archivos que sirve como base para empezar un nuevo proyecto con ionic. Han sido generados utilizando el comando *ionic start*.

- Podemos hacer que ionic ponga en marcha un servidor de desarrollo para poder acceder a la aplicación a través de un browser. Solo tenemos que ejecutar el comando *ionic serve* dentro del directorio anterior. Aparecerá un mensaje indicando que el servidor está levantado además de cierta información como la url para acceder a la aplicación, los ficheros que está sirviendo o los comandos para realizar ciertas acciones.

```
#ionic serve
*****
Dependency warning - for the CLI to run correctly,
it is highly recommended to install/upgrade the following:

Please install your Cordova CLI to version >=4.2.0 'npm install -g cordova'
*****
Running live reload server: http://192.168.1.38:35729
Watching: www/**/*, !www/lib/**/*, !www/**/*.*map
✓ Running dev server: http://192.168.1.38:8100
Ionic server commands, enter:
  restart or r to restart the client app from the root
  goto or g and a url to have the app navigate to the given url
  consolelogs or c to enable/disable console log output
  serverlogs or s to enable/disable server log output
  quit or q to shutdown the server and exit
```

Figura D.2: Vemos como el servidor de la aplicación ionic está en marcha.

- Si abrimos nuestro navegador y accedemos a la url que indica el servidor (por defecto será la IP de la máquina y el puerto 8100), veremos la aplicación servida, que no es más que el template descargado anteriormente:

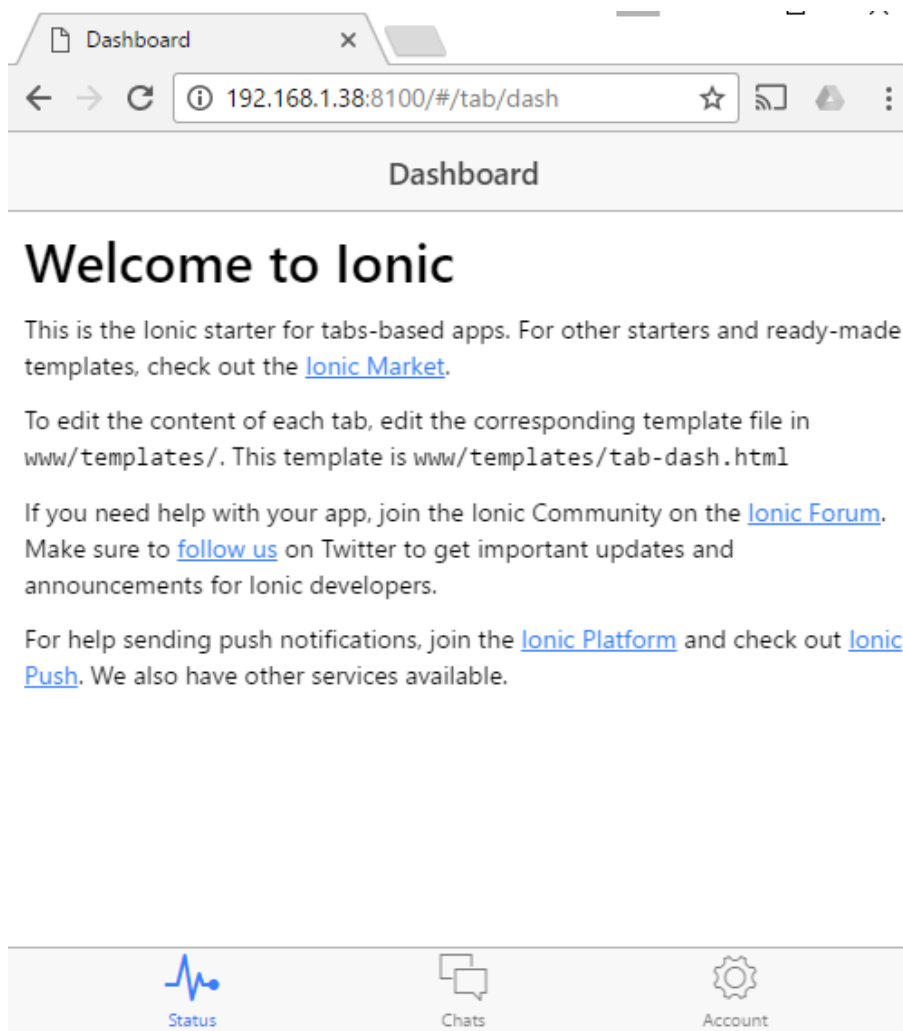


Figura D.3: La aplicación vista desde un navegador.

ANEXOS E

GOOGLE APIS

Se conoce por este nombre al conjunto de **APIs** desarrolladas por Google y que permiten a aplicaciones de terceros acceder a los servicios de Google, como pueden ser Search, Google Map, Google Traductor, Gmail, El uso de estas **APIs** suelen estar limitado a un número de peticiones diarias, acarreando un coste adicional todas aquellas que superen este límite.

Para poder hacer uso de estas **APIs** es necesario autorización y autenticación, es por ello necesario crear antes una cuenta en Google para poder obtener las credenciales con las que autenticarnos ante los servicios de Google. Si ya disponemos de una cuenta en Google, podremos acceder a la consola para desarrolladores¹. Desde esta consola podremos realizar diferentes gestiones sobre nuestra cuenta.

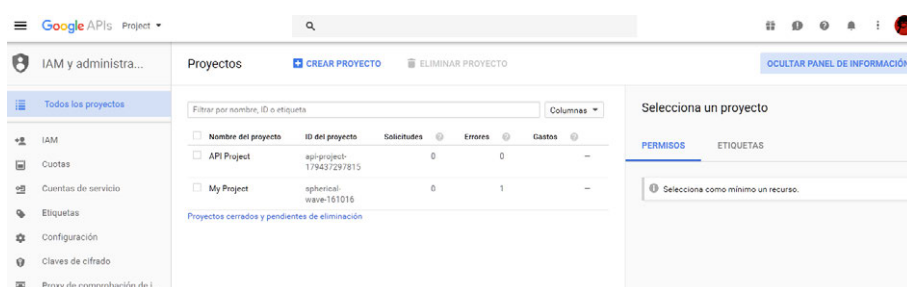


Figura E.1: Consola para desarrolladores de Google.

A nuestra cuenta le podemos asignar varios proyectos. Esto nos ayudará a gestionar por separado las credenciales que generemos y poder diferenciar el uso que se le ha dado a cada una de ellas (consumo, número de peticiones, acceso a servicios, ...). Esto es útil en caso de tener varias aplicaciones, o simplemente tener

¹<https://console.developers.google.com>

una sola aplicación, pero varios entornos (desarrollo, preproducción, producción, piloto, ...)

E.1. Obtención de una clave para utilizar Google Maps

Para la realización de una de las prácticas propuestas, va a ser necesario el uso de una clave, *API KEY*, que permita a nuestra aplicación hacer uso del servicio de Google Maps. Vamos a ver como podemos conseguir esta clave. Daremos por supuesto que disponemos de una cuenta de Google a la cual vincularemos nuestro proyecto.



Tened en cuenta que el uso de estas **APIs** pueden incurrir en costes que serán cargados a la cuenta a la que este asociado el proyecto. Debemos ser cautelosos con el uso que hacemos des de estas claves, y en caso de no necesitarlas, destruirlas.

En primer lugar, crearemos un proyecto dentro de nuestra cuenta. Le asignaremos un nombre para poder identificarlo, y si lo deseamos, podremos modificar el ID que asigna Google automáticamente.

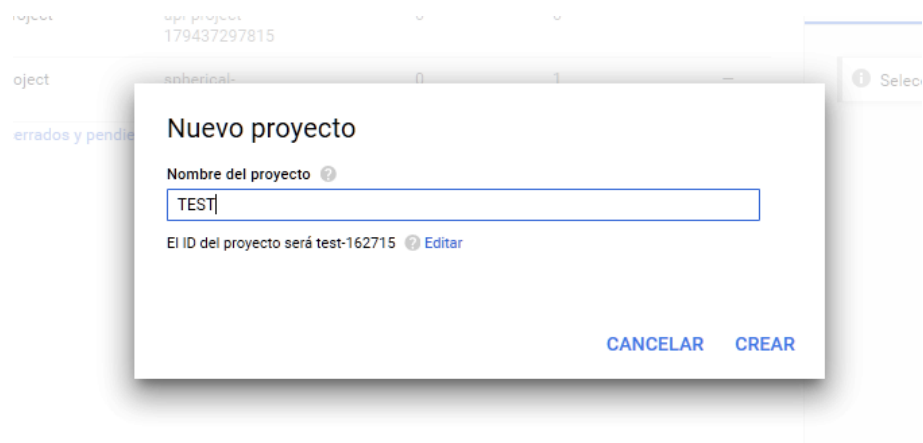


Figura E.2: Ventana para la creación de un proyecto dentro de nuestra cuenta.

Una vez creado, podremos verlo en la lista con el resto de proyectos y podremos seleccionarlo.

Proyectos [+ CREAR PROYECTO](#) [ELIMINAR PROYECTO](#)

Filtrar por nombre, ID o etiqueta Columnas ▾

| <input type="checkbox"/> | Nombre del proyecto | ID del proyecto | Solicitudes [?] | Errores [?] | Gastos [?] |
|--------------------------|---------------------|--------------------------|--------------------------|----------------------|---------------------|
| <input type="checkbox"/> | API Project | api-project-179437297815 | 0 | 0 | — |
| <input type="checkbox"/> | My Project | spherical-wave-161016 | 0 | 1 | — |
| <input type="checkbox"/> | TEST | test-162715 | 0 | 0 | — |

[Proyectos cerrados y pendientes de eliminación](#)

Selecciona un proy

PERMISOS ETIQUI

Selección como mini

Figura E.3: Lista de proyectos asociados a nuestra cuenta.

Si desplegamos el menú lateral, podremos ver diferentes secciones relativas a nuestro proyecto, dentro de estas secciones a su vez nos encontraremos diferentes apartados, de los cuales, veremos los más importantes.

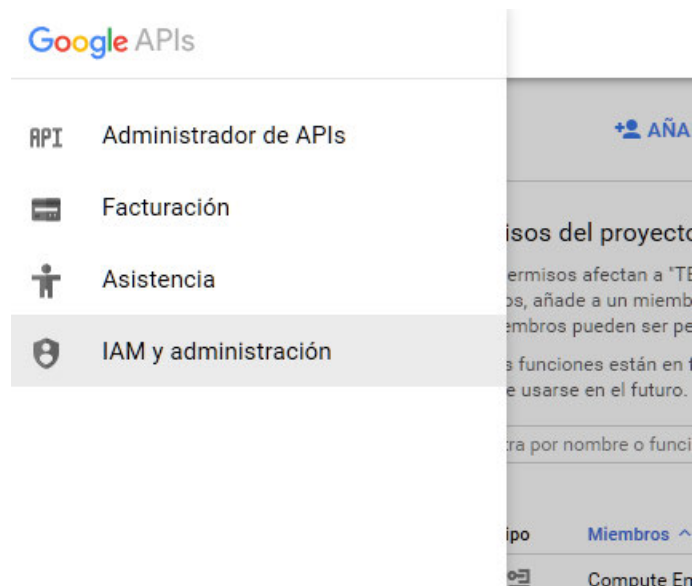


Figura E.4: Menú desde el que acceder a diferentes secciones relativas al proyecto.

Si accedemos a la sección **Administrador de APIs**, podremos ver el **Panel de control**. En este panel aparecerán las estadísticas de uso de los diferentes servicios asociados. Recién creado el proyecto, esta pantalla debería aparecer vacía. Si abrimos la **Biblioteca**, veremos una lista con los diferentes servicios de Google. Aquí seleccionaremos los servicios que queremos habilitar para nuestro proyecto.

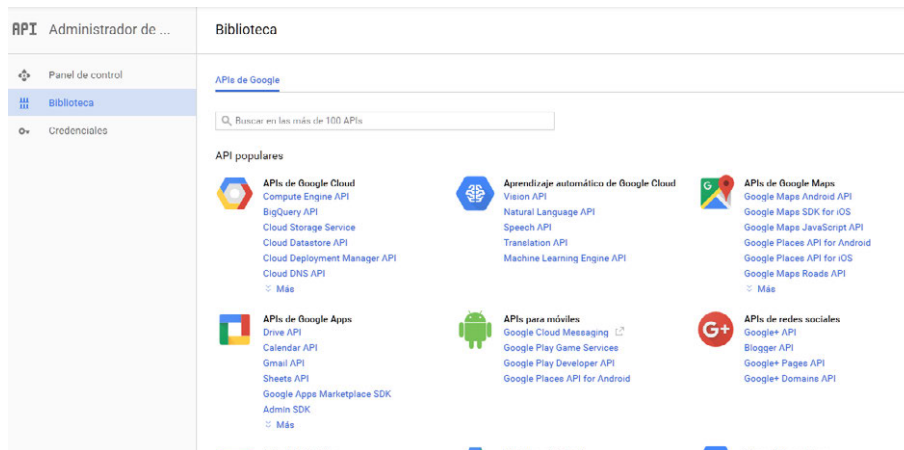


Figura E.5: APIs disponibles para poder asignar al proyecto.

A nosotros nos interesa el servicio **Google Maps Android API**. Lo seleccionamos y nos aparecerá una pantalla con una pequeña explicación del servicio y un botón con el que habilitarlo.

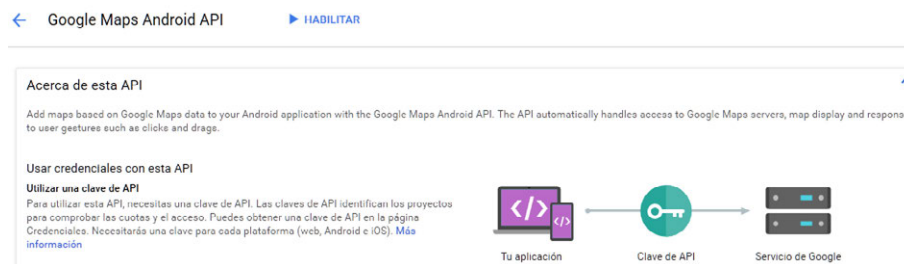


Figura E.6: Información relativa a la API de Google Maps Android.

Tras ello nos abrirá de nuevo el panel de control, esta vez, podremos ver las gráficas que nos ofrecen, que aun estando vacías, podremos hacernos una idea de la información que ofrecerán.

El siguiente paso será crear nuestra clave. Lo podremos apartado **Credenciales**, en la que veremos que al crear una, nos ofrecerá varias opciones, entre ellas, la de generar una clave que es la que nos interesa.

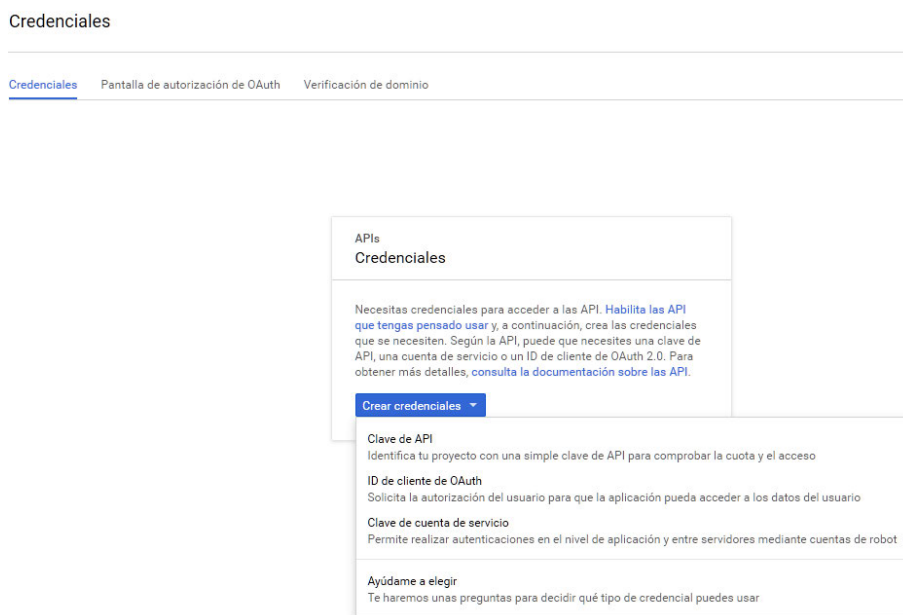


Figura E.7: Configuración de las credenciales para nuestra API. El servicio nos proporciona diferentes maneras para realizar la autorización.

Una vez creada, nos aparecerá para poder copiarla (aunque podremos consultarla cuando queramos desde la consola) y nos dará la opción de restringir el acceso. Esta opción es muy interesante, especialmente una vez puesta en producción, ya que nos permitirá limitar quien hace uso de ella ya sea mediante una dirección **Internet Protocol (IP)**, una URL, el nombre del paquete de una aplicación de Android, Si no ponemos esta restricción, alguien podría obtener la clave desde el código (por ejemplo depurando la aplicación) y utilizarla para su propia aplicación cargandonos a nosotros con los costes de uso.

Credenciales

[←](#)
[Volver a generar clave](#)
[Eliminar](#)

Clave de API

Esta clave de API se puede usar en este proyecto y con cualquier API compatible. Para usar esta clave en tu aplicación, transfírela con el parámetro `key=API_KEY`.

| | |
|-------------------|------------------------------|
| Fecha de creación | 26 mar. 2017 17:31:07 |
| Creada por | cesar.gon.fer@gmail.com (tú) |

Clave de API

AIZA5yAiCY-wkWwFKdAOoe72ac1a2e2x8QH7p40

Nombre

Clave de API 1

⚠ Restricción de clave

Esta clave no tiene restricciones. Para evitar un uso no autorizado y el robo de cuotas, restringela. Si restringes una clave, puedes especificar qué sitios web, direcciones IP o aplicaciones pueden usarla. [Más información](#)

☒ Ninguna
☐ URLs de referencia HTTP (sitios web)
☐ Direcciones IP (servidores web, tareas cron, etc.)
☐ Aplicaciones para Android
☐ Aplicaciones para iOS

Nota: Pueden pasar hasta 5 minutos antes de que se aplique la configuración

[Guardar](#)
[Cancelar](#)

Figura E.8: Opciones para restringir el acceso según el origen de la petición. Para una aplicación Android, necesitaríamos indicar el nombre del paquete de nuestra aplicación.

De vuelta al menú lateral, si nos vamos a la sección **IAM y administración**, podemos destacar el apartado **Identify and Access Management (IAM)**, donde podremos dar permisos sobre el proyecto a otros usuarios, por ejemplo, permisos de edición, de lectura, sobre la facturación, ...

IAM y administra...

Todos los proyectos

IAM

Cuotas

Cuentas de servicio

Etiquetas

Configuración

Claves de cifrado

Proxy de comprobación de i...

IAM

[+ AÑADIR](#)
[- QUITAR](#)

Permisos del proyecto "TEST"

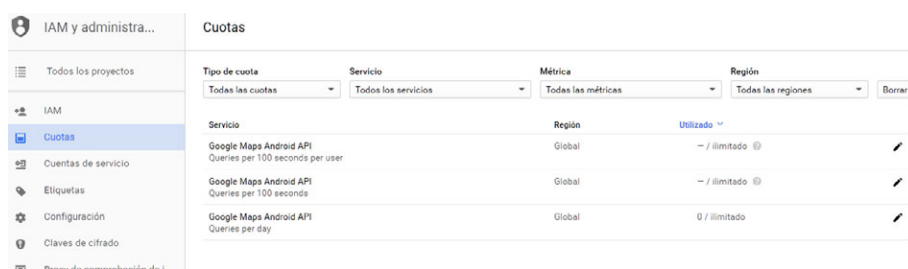
Estos permisos afectan a "TEST" proyecto y a todos sus recursos. Para otorgar permisos, añade a un miembro y selecciona la función que quieres asignarle. Los miembros pueden ser personas, dominios, grupos o cuentas de servicio. Algunas funciones están en fase de desarrollo beta y es posible que cambien o dejen de usarse en el futuro. [Obtén más información al respecto](#)

Ver por: Miembros

| <input type="checkbox"/> | Tipo | Miembros | Funciones |
|--------------------------|------|--|----------------------------|
| <input type="checkbox"/> | | Compute Engine default service account 429039849652-compute@developer.gserviceaccount.com | Editor ▼ |
| <input type="checkbox"/> | | Cesar Gonzalez Fernandez cesar.gon.fer@gmail.com | Propietario ▼ |

Figura E.9: Permisos sobre el proyecto para distintos usuarios de la consola de desarrolladores. Aquí podemos compartir el proyecto con el resto del equipo de desarrollo y asignarles diferentes permisos a cada uno.

Y el apartado de **Cuotas**, donde vemos las cuotas de los servicios activados y podremos limitarlas.



| Tipo de cuota | Servicio | Métrica | Región | |
|----------------------------------|---------------------|--------------------|--------------------|--------|
| Todas las cuotas | Todos los servicios | Todas las métricas | Todas las regiones | Borrar |
| Servicio | Región | Utilizado | | |
| Google Maps Android API | Global | — / ilimitado | | |
| Queries per 100 seconds per user | Global | — / ilimitado | | |
| Google Maps Android API | Global | — / ilimitado | | |
| Queries per 100 seconds | Global | 0 / ilimitado | | |
| Google Maps Android API | Global | 0 / ilimitado | | |
| Queries per day | | | | |

Figura E.10: Cuotas que se aplican a cada uno de los servicios asociados al proyecto.