



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**Trabajo fin de Grado**

**Grado en Ingeniería Informática  
Ingeniería de Computadores**

**Desarrollo de Aplicaciones Móviles Multi-Plataforma: Gamificación del  
proceso de aprendizaje del idioma**

**Realizado por  
Javier Macarro Medina**

**Dirigido por  
Francisco Jesús Martín Mateos**

**Departamento  
Ciencias de la Computación  
e Inteligencia Artificial**

**Sevilla, Septiembre de 2019**



---

## **Resumen**

---

Este Trabajo de Fin de Grado consiste en la creación de una aplicación híbrida multiplataforma para los sistemas operativos Android y iOS utilizando Ionic, que ofrezca tanto a niños como a adultos la posibilidad de aprender idiomas de una forma dinámica y divertida a través de los juegos. Sin limitaciones de dispositivo, conexión a internet o largas esperas para obtener actualizaciones de contenido. En esta documentación, realizaremos un recorrido que nos guiará, paso a paso, a través de las diferentes fases de desarrollo de una aplicación: desde la preparación del entorno de desarrollo hasta la ejecución de la aplicación en nuestros dispositivos.

Nuestra aplicación, una vez terminada, constará de dos juegos y varias tablas incluidas por defecto. Dado que las tablas pueden ser creadas por los propios usuarios, es relativamente fácil mantener la aplicación con contenido nuevo. Pero este no es nuestro único propósito; además, solo sería necesario lanzar actualizaciones para aumentar el número de juegos, ya que se ha diseñado para ser modular y escalable. Por ello, dedicaremos el último capítulo de esta memoria a detallar todos los pasos necesarios para ampliar nuestra aplicación con nuevos juegos.



---

## **Agradecimientos**

---

Esta es la ocasión definitiva, ya es hora de cerrar mi currículum académico. Después de haber postergado este complejo y minucioso trabajo, me gustaría dejar presente en este apartado a mi familia, que no ha dejado de apoyarme en todo momento, a pesar de que yo desistiera en más de una ocasión.

El TFG ha sido protagonista de muchas comidas y eventos familiares, pero bueno, más vale tarde que nunca. Antes, pensaba que llevar adelante mi trabajo, en el que llevo casi cuatro años, era una tarea complicada y que con eso tenía suficiente. Este proyecto me ha hecho darme cuenta de mis capacidades como futuro Ingeniero Informático, estirándolas hasta ser capaz de llevar dos tareas de tal magnitud a la vez y sobrevivir.

Todo llega a su debido momento, y este ha sido el mio, una vez que me he visto con fuerzas y preparado para dar la talla en un proyecto de este calibre y con el que he disfrutado en muchos sentidos. A veces, con suerte, llegan personas a tu vida que hacen que te des cuenta de que si algo es realmente importante para ti tienes que luchar por ello. Con suerte, con mucha mucha suerte, esas personas, te apoyarán pase lo que pase animándote y achuchándote en cada momento aunque decaigas, aunque estés cansado, aunque estés a punto de rendirte...

Yo tengo la enorme suerte de tener muchos de esos faros en mi vida, iluminándome y velando por mi. Sin más preámbulos solo puedo decir gracias Mamá, gracias Papá, gracias hermanita, gracias titos y titas sin los que no sería la persona que soy.

Y, sobre todo, muchas gracias Ana, por ser mi Sol, sin ti, realizar este gran proyecto no hubiese sido lo mismo, ni tan divertido.



---

## Índice general

---

Índice general	V
Índice de figuras	VII
Índice de código	IX
<b>1 Introducción</b>	<b>1</b>
1.1 Motivación . . . . .	1
1.2 Introducción a Ionic . . . . .	2
1.2.1 ¿Qué es una aplicación híbrida? . . . . .	2
1.2.2 Desarrollo Multi-plataforma vs Nativo . . . . .	3
1.2.3 Contexto histórico . . . . .	4
1.3 Conocimientos iniciales requeridos . . . . .	5
1.4 Composición . . . . .	5
1.4.1 Angular: Conceptos básicos . . . . .	7
1.4.2 Node.js . . . . .	8
1.4.3 NPM . . . . .	8
1.4.4 Ionic CLI . . . . .	9
1.4.5 Cordova CLI . . . . .	9
1.5 Preparación del entorno de desarrollo . . . . .	10
1.5.1 Instalar Node.js y NPM . . . . .	10
1.5.2 Control de versiones . . . . .	10
1.5.3 Visual Studio Code . . . . .	13
1.5.4 Instalar Ionic y Cordova . . . . .	14
1.5.5 Configurar entorno para Android desde Windows . . . . .	14
1.5.6 Configurar entorno para iOS desde macOS . . . . .	21
1.5.7 Tutoriales de ayuda . . . . .	22
1.5.8 Compilación y ejecución en un dispositivo Android . . . . .	22
1.5.9 Compilación y ejecución en un dispositivo iOS . . . . .	25
1.6 Estructura del proyecto . . . . .	27
1.7 Estructura de la documentación . . . . .	29
1.8 Hoja de ruta para el código . . . . .	30
<b>2 Página principal</b>	<b>31</b>
2.1 Mock Up . . . . .	31
2.2 Vista . . . . .	32

2.3 Código . . . . .	39
2.4 Resultado final . . . . .	40
<b>3 Jugar</b>	<b>41</b>
3.1 Elige temática . . . . .	41
3.1.1 <i>Mock Up</i> . . . . .	41
3.1.2 Vista . . . . .	42
3.1.3 Código . . . . .	43
3.1.4 Resultado final . . . . .	51
3.2 Página Elige juego . . . . .	52
3.2.1 Mock Up . . . . .	52
3.2.2 Vistas . . . . .	53
3.2.3 Código . . . . .	53
3.2.4 Resultado final . . . . .	55
3.3 Página Une palabras . . . . .	56
3.3.1 Mock Up . . . . .	56
3.3.2 Vistas . . . . .	57
3.3.3 Código . . . . .	58
3.3.4 Resultado final . . . . .	63
3.4 Página Di mi nombre . . . . .	64
3.4.1 Mock Up . . . . .	64
3.4.2 Vistas . . . . .	65
3.4.3 Código . . . . .	66
3.4.4 Resultado final . . . . .	70
<b>4 Gestión de tablas</b>	<b>71</b>
4.1 Página Crea tu tabla . . . . .	71
4.1.1 Mock Up . . . . .	71
4.1.2 Vista . . . . .	72
4.1.3 Código . . . . .	73
4.1.4 Resultado final . . . . .	75
4.2 Formularios de creación y modificación de tablas . . . . .	76
4.2.1 Mock Up . . . . .	76
4.2.2 Vista . . . . .	77
4.2.3 Código . . . . .	79
4.2.4 Resultado final . . . . .	81
<b>5 Resultados</b>	<b>83</b>
5.1 Mock Up . . . . .	83
5.2 Resultado final . . . . .	84
<b>6 Manual de uso</b>	<b>87</b>
<b>7 Conclusiones</b>	<b>101</b>
<b>Bibliografía</b>	<b>103</b>

---

## Índice de figuras

---

1.1	Creación de Repositorio GIT local . . . . .	11
1.2	Clone Repositorio GITHUB . . . . .	12
1.3	Sincronización con repositorio web GIT . . . . .	12
1.4	Configuración: Inicio Instalación Android Studio . . . . .	14
1.5	Configuración: Elección de componentes . . . . .	15
1.6	Configuración: Importar configuración de Android Studio . . . . .	15
1.7	Configuración: Setup de Android Studio . . . . .	16
1.8	Configuración: SDK Manager . . . . .	17
1.9	Android SDK Platforms . . . . .	17
1.10	Android SDK Tools . . . . .	18
1.11	Instalación Java JDK . . . . .	19
1.12	Configuración: Instalando distribución de Gradle . . . . .	19
1.13	Propiedades del sistema . . . . .	20
1.14	Variables de entorno . . . . .	20
1.15	Creación de equipo de desarrollo . . . . .	22
1.16	Ionic Cordova Prepare . . . . .	22
1.17	Ionic Cordova Build . . . . .	23
1.18	Ionic Cordova Run Error . . . . .	24
1.19	Ionic Cordova Run Solución . . . . .	24
1.20	Configurando Xcode . . . . .	25
1.21	Configurando Firma . . . . .	26
1.22	Desplegando iOS . . . . .	26
1.23	Estructura del proyecto . . . . .	27
2.1	<i>Mock up</i> Menú principal . . . . .	31
2.2	Estructura directorio src . . . . .	34
2.3	Menú principal . . . . .	40
3.1	<i>Mockup</i> Elige temática . . . . .	41
3.2	Elige temática . . . . .	51
3.3	Mockup Elige juego . . . . .	52
3.4	Elige juego . . . . .	55
3.5	Mockup Une palabras . . . . .	56
3.6	Ciclo de vida de una página de Ionic . . . . .	59
3.7	Une palabras . . . . .	63
3.8	<i>Mockup</i> Di mi nombre . . . . .	64

---

3.9 Comandos Ionic CLI . . . . .	65
3.10 Di mi nombre . . . . .	70
4.1 <i>Mockup</i> Crea tu tabla . . . . .	71
4.2 Crea tu tabla . . . . .	75
4.3 Mockup Modifica tu tabla . . . . .	76
4.4 Formulario: Crear Tabla . . . . .	81
4.5 Formulario: Modificar Tabla . . . . .	82
5.1 <i>Mock up</i> Resultados . . . . .	83
5.2 Resultados: Elegir Tabla . . . . .	84
5.3 Resultados: Con valores Disponibles . . . . .	85
5.4 Resultados: Sin Valores Disponibles . . . . .	86
6.1 Say My Name . . . . .	87
6.2 Splash Screen . . . . .	88
6.3 Menú de inicio . . . . .	89
6.4 Menú desplegable . . . . .	90
6.5 Información sobre la App . . . . .	91
6.6 Página de elección de la temática . . . . .	92
6.7 Página de elección del juego . . . . .	93
6.8 Juego Une Palabras . . . . .	94
6.9 Juego Di Mi Nombre . . . . .	95
6.10 Menú de observación de las tablas realizadas . . . . .	96
6.11 Creación de nueva tabla . . . . .	97
6.12 Crear o modificar tablas . . . . .	98
6.13 Página Resultados . . . . .	99
6.14 Visualización de resultados obtenidos . . . . .	100

---

## Índice de código

---

1.1 Fichero index.html para mostrar alert . . . . .	6
2.1 Fichero ejemHeader.html . . . . .	33
2.2 Fichero app.module.ts . . . . .	36
2.3 Fichero app.component.ts . . . . .	37
2.4 Fichero index.html . . . . .	37
2.5 Función initializeApp . . . . .	38
2.6 Código app-routing.module.ts . . . . .	38
2.7 Función ionViewWillEnter . . . . .	39
3.1 Fichero select-table.page.scss . . . . .	42
3.2 Fichero select-table.page.html . . . . .	42
3.3 Fichero database.service.ts, Constructor . . . . .	43
3.4 Fichero database.service.ts, seedDatabase . . . . .	44
3.5 Fichero database.service.ts, @injectable y Observables . . . . .	44
3.6 Fichero select-table.page.ts . . . . .	45
3.7 Ejemplo de programación tradicional . . . . .	47
3.8 Ejemplo de programación reactiva . . . . .	47
3.9 Fichero database.service.ts, declaración BehaviourSubjects . . . . .	48
3.10 Fichero database.service.ts, loadTables . . . . .	48
3.11 Fichero database.service.ts, getTables . . . . .	49
3.12 Fichero notifications.component.html, Uso de routerLink . . . . .	53
3.13 Ejemplo Angular Router desde .ts . . . . .	53
3.14 Fichero une-palabras.page.html . . . . .	57
3.15 Fichero select-game.page.ts, paso de parámetros . . . . .	58
3.16 Fichero une-palabras.page.ts, recepción de parámetros . . . . .	58
3.17 Fichero une-palabras.page.ts, solución problema observables . . . . .	61
3.18 Fichero database.service.ts, solución problema observables . . . . .	62
3.19 Fichero une-palabras.page.ts, función para timer asincrono . . . . .	62
3.20 Fichero une-palabras.page.ts, uso de funcion timer . . . . .	62
3.21 Fichero pipes.module.ts . . . . .	66
3.22 Fichero spell.pipe.ts . . . . .	67
3.23 Fichero di-mi-nombre.page.ts, lockSwipes . . . . .	67
3.24 Fichero di-mi-nombre.page.ts, presentToast . . . . .	68
3.25 Fichero notifications.component.html . . . . .	69
3.26 Fichero notifications.component.html . . . . .	69
3.27 Fichero di-mi-nombre.page.ts, notifications . . . . .	69
4.1 Fichero manage-tables.page.ts, presentAlertConfirmDeletion . . . . .	73
4.2 Fichero manage-tables.page.ts, presentFormCreationModal . . . . .	74

---

4.3 Fichero <code>form-creation.page.html</code> , header . . . . .	77
4.4 Fichero <code>form-creation.page.html</code> , content . . . . .	78
4.5 Fichero <code>form-creation.page.ts</code> , mensajes de validación . . . . .	79
4.6 Fichero <code>form-creation.page.ts</code> , ngOnInit . . . . .	79
4.7 Fichero <code>form-creation.page.ts</code> , añadir y borrar controles . . . . .	80

# CAPÍTULO 1

---

## Introducción

---

### 1.1– Motivación

Para dar comienzo a este trabajo de fin de carrera y como broche final de mi recorrido como estudiante, he decidido realizar un proyecto que pueda estar al alcance de todos los públicos. Para ello, he realizado un pequeño estudio de mercado en el que se perciben cuáles son las necesidades principales de nuestra actual y tecnológica sociedad.

En mi caso, he prestado una mayor atención al campo de los idiomas, y a los métodos de aprendizaje que usan algunas de las aplicaciones que he encontrado en las tiendas de *software* de los dos Sistemas Operativos Móviles más utilizados del momento: **Android** e **iOS**. Algunas de ellas tan sólo están disponibles en ciertas plataformas, dificultando así el poder usarlas dónde y cuándo queramos; otras, a mi parecer, disponen de una estructura y funcionamiento un tanto complejo y poco intuitivo, lo que causa una gran confusión al utilizarlas por primera vez. Si tenemos en cuenta además, que el usuario final de la gran mayoría de aplicaciones de este tipo son niños y padres que podrían no estar acostumbrados a gestionar bases de datos, o a utilizar un entramado sistema de configuración basado en tablas interrelacionadas, el resultado final obtenido es... que acaban dando por imposible el uso de la *App*, y esta cae en el olvido.

Tras este pequeño análisis, decidí que mi objetivo principal sería crear una aplicación disponible al menos en las dos plataformas móviles previamente mencionadas, fácil tanto de configurar como de usar, utilizando una metodología más cercana y amigable. Dicha metodología son los juegos, ya que es una forma más dinámica y divertida para el aprendizaje, sobre todo, en el caso de un niño. Actualmente, nuestra aplicación dispone de **dos juegos** que exponen esta idea. En ella, tendremos conjuntos de datos por defecto a modo de exemplificación, al mismo tiempo que podremos crear nuestros propios conjuntos de datos para no depender de una conexión a internet ni de actualizaciones desde la *Store*, pero de esto ya hablaremos más adelante.

Para abordar este objetivo, decidí utilizar como herramienta el *framework Ionic*, en lugar de desarrollar la misma *app* en dos plataformas tan distintas de forma nativa. A mi parecer, el desarrollo de aplicaciones multi-plataforma es una de esas novedosas incógnitas que a todo compañero del gremio le gustaría dominar, pero que, ya bien sea debido a cuestiones de tiempo o a la vertiginosa curva de aprendizaje inicial, han dejado apartado esperando el momento idóneo.

Si te has sentido identificado al leer esto último, ¡Enhorabuena, ese momento ha llegado!, puesto que mi otra gran motivación ha sido la de enfocar este proyecto, no tan solo como el desarrollo de una *App*, sino también como una guía paso a paso sobre como utilizar esta tecnología, visitando uno por uno los elementos más útiles y necesarios para el desarrollo de una app completa, desde la **preparación del entorno de desarrollo**, pasando por la ejecución de mecánicas complejas, hasta la ejecución de esta; permitiendo (una vez completado este *Tutorial*) a cualquier persona con mínimos conocimientos de programación, tener una plantilla bastante completa para crear una aplicación, ahorrando así muchísimas horas de aprendizaje y quebraderos de cabeza.

He de admitir que es una aplicación con un diseño minimalista, siguiendo las tendencias del momento, y que el nivel de dificultad inicial es demasiado fácil para un adulto. A lo largo de este trabajo desarrollaremos tan solo un enfoque principal hacia los niños o usuarios con bajo conocimiento del lenguaje que se quiere aprender, pero como proyecto de futuro, para obtener una aplicación completa, añadiríamos nuevos niveles de dificultad con mayor diversidad de juegos que permitirían el uso a un mayor número de usuarios con rangos superiores de conocimiento en el lenguaje, y también permitiendo a su vez a los usuarios más noveles, llevar una progresión ascendente conforme vayan superando los niveles inferiores.

Dicho proyecto de futuro es posible gracias a que hemos diseñado la *app* para ser modular y fácilmente escalable, permitiéndonos automatizar el proceso todo lo posible, como podremos ver en el último capítulo de esta memoria.

Se ha dado una breve (o quizás no tan breve) introducción acerca de la aplicación desarrollada en este trabajo, pero aún no se ha mencionado su nombre, os presento SAY MY NAME, y sin más preámbulos, adentrémonos en el mundo del *Cross-platform mobile app development* (o como a mí me gusta llamarlo *Desarrollo de Aplicaciones Móviles Multi-Plataforma*).

## 1.2– Introducción a Ionic

IONIC FRAMEWORK<sup>1</sup> es un kit de herramientas diseñado principalmente para la creación de aplicaciones móviles (tales como teléfonos inteligentes y tablets) de alta calidad, que tengan un gran rendimiento y que estén alineadas con los estándares de diseño más utilizados en los principales sistemas operativos de la actualidad. Y sí, decimos sistemas operativos porque Ionic, con la ayuda de otras herramientas como lo son **Cordova** (hablaremos de ella en el apartado 1.4.5) o Capacitor<sup>2</sup>, nos permite crear *apps* híbridas multi-plataforma.

### 1.2.1. ¿Qué es una aplicación híbrida?

Una aplicación **híbrida** es un subtipo de aplicación multi-plataforma móvil que se diferencia de otras metodologías facilitando a los desarrolladores la posibilidad de escribir el código de la *app* como si de una aplicación web se tratase, usando tecnologías web tan

---

<sup>1</sup>Conjunto de herramientas pensado para hacer más sencilla la programación de cualquier aplicación o herramienta actual mediante prácticas, conceptos y criterios que tienen como objetivo optimizar los costes de desarrollo de un proyecto.

<sup>2</sup>Lo nombro aquí para que conozcáis de su existencia, pero ya os adelanto que no lo hemos usado debido a que Cordova tiene mucho más recorrido y nos aporta mucha más documentación y experiencia a la hora de afrontar un proyecto desde cero.

conocidas (para cualquier programador web) como lo son **HTML5, JavaScript y CSS** entre otras, para luego empaquetar e implementar todo esto dentro de un contenedor nativo utilizando el correspondiente **componente** integrado del navegador de cada plataforma, lo que le permite ejecutarse como si fuese una aplicación nativa de un sistema operativo concreto. En definitiva, estas resolverían los problemas a los que se enfrentan muchas aplicaciones web, sobre todo, en los casos en los cuales no requieren conectividad a Internet, ya que ofrecen la ventaja de, una vez descargadas, sustentarse por sí mismas utilizando llamadas a su base de datos almacenada en los dispositivos. Las aplicaciones híbridas también pueden ser publicadas en las tiendas de aplicaciones, el cual es un mercado muy a la orden del día de cualquier usuario medio, donde estos pueden encontrarlas fácilmente, aumentando las posibilidades de descubrimiento y, en consecuencia, el número de usuarios.

Originalmente, la complejidad del desarrollo de aplicaciones móviles se vio agravada por la dificultad de construir un *back-end*<sup>3</sup> que funcionará en múltiples plataformas. Aunque consumía mucho tiempo y era costoso, a menudo era más fácil crear aplicaciones nativas para cada Sistema Operativo (SO) móvil. El problema era que el código creado para un sistema operativo no podía ser reutilizado para otro.

Punto y aparte, a pesar de que no entra dentro del alcance de esta memoria, también cabe mencionar que Ionic no solo nos permite desarrollar aplicaciones para dispositivos móviles. Con la ayuda de Electron<sup>4</sup>, la creación de una aplicación de escritorio (con SO WINDOWS 10 ANNIVERSARY UPDATE o MACOS) con Ionic es un hecho relativamente sencillo, permitiendo a los desarrolladores reutilizar el 100% de su código y ofrecer una aplicación de escritorio tradicional sin dejar de tener acceso a todas las funciones nativas del dispositivo móvil, como, por ejemplo, las notificaciones push.[2]

### 1.2.2. Desarrollo Multi-plataforma vs Nativo

Hoy en día, es mucho más fácil para los programadores desarrollar aplicaciones multi-plataforma, gracias a frameworks como Ionic, que suplen todos los inconvenientes mencionados anteriormente. Como ya llevamos bastante aprendido y soy consciente de que puede ser complejo de procesar, hagamos sumario de los conceptos fundamentales.[10]

- Ventajas:

1. Código reutilizable: Las herramientas de desarrollo multi-plataforma permiten, escribiendo el código una vez, exportar la aplicación a muchos sistemas operativos y plataformas sin tener que crear una aplicación nativa dedicada para cada una de ellas.
2. Conveniencia: Las herramientas de desarrollo multi-plataforma nos ahorran la molestia de tener que aprender múltiples lenguajes de programación y en su lugar nos ofrecen un sustituto común para todas estas diferentes y complejas tecnologías.
3. Código mantenible: Cada vez que modifiquemos o actualicemos nuestra *app*, tan solo tendremos que compilar y actualizar los binarios para cada plataforma a fin de lograr que los cambios se reflejen en ellas.

---

<sup>3</sup>El back-end es la parte del desarrollo web que se encarga de que toda la lógica, la conexión a las bases de datos y la comunicación con el servidor funcione.

<sup>4</sup>Es un framework que permite el desarrollo de aplicaciones gráficas de escritorio usando componentes del lado del cliente y del servidor originalmente desarrolladas para aplicaciones web: Node.js del lado del servidor y Chromium como interfaz.

4. Eficiencia de costes: El desarrollo multi-plataforma nos permite ahorrar el coste de tener varios equipos trabajando en diferentes versiones de la aplicación y sustituyéndolos por un solo equipo. La mayoría de las herramientas de desarrollo multi-plataforma también son de uso gratuito, y algunas ofrecen suscripciones de pago para funciones adicionales<sup>5</sup>.
5. Alcance del mercado: Al publicar la aplicación en múltiples plataformas, se está creando una red más amplia y se aumentan las posibilidades de tener una mayor base de usuarios y, en consecuencia, un mayor retorno de la inversión y mayores ingresos.

- Desventajas:

1. Rendimiento: Aunque algunas herramientas de desarrollo multi-plataforma ofrecen un rendimiento similar al de una aplicación nativa, nunca son tan buenas. Por eso no deberíamos utilizar este tipo de herramientas si el rendimiento de la aplicación es la máxima prioridad.
2. 3D y Gráficos: Las herramientas de desarrollo multi-plataforma no son conocidas por ofrecer los mejores gráficos y experiencias de usuario, y pueden carecer de acceso a las principales librerías<sup>6</sup> del SO, como las gráficas. El desarrollo multi-plataforma podría no ser la mejor opción si la aplicación depende en gran medida de los gráficos.
3. Características específicas del dispositivo: Alguna de las características hardware de los dispositivos podrían tener su acceso limitado a sus entornos nativos. En el caso de *apps* que requieran de ello, será necesario estudiar el alcance deseado antes de decidir utilizar un framework como Ionic.
4. Características específicas de la plataforma: Aunque las herramientas de desarrollo multi-plataforma ofrecen muchas de las funciones básicas compartidas entre diferentes plataformas, pueden carecer de algunas de las funciones específicas que ofrecen APPLE, GOOGLE y MICROSOFT en sus respectivos sistemas operativos.

Ninguno de estos supone un gran problema para el proyecto en el que nos encontramos sumergidos por lo que Ionic es idóneo en nuestro caso.

### 1.2.3. Contexto histórico

Quizás sean pocas las personas que conocen Drifty Co[9], y digo “pocas” en comparación con los grandes gigantes del mercado como son APPLE, GOOGLE o la archiconocida MICROSOFT. De hecho, antes de afrontar este proyecto, nunca había oído hablar de ellos. Con esta empresa, Max Lynch, Ben Sperry, y Adam Bradley[9] fundaron (allá por 2012) Ionic, cuando el uso de las tecnologías web como medio para construir aplicaciones nativas estaba todavía en su infancia, siendo lanzada su primera *release* estable, Ionic 1.0 en Mayo de 2015. Por aquellos entonces este iba acompañado de la mano de **AngularJS**. Mucho han evolucionado las cosas, hoy en día, Ionic es el pilar tecnológico de desarrollo

---

<sup>5</sup>Como Ionic que, entre otras cosas, ofrece un *IDE* o entorno de desarrollo propietario con muchas funciones que facilitan la programación de aplicaciones, sobre todo en cuanto a aspecto y visualización.

<sup>6</sup>Soy consciente del mal uso de esta palabra, pero por convención la utilizaremos durante toda la documentación para evitar confusión.

móvil multi-plataforma más popular del mundo, impulsando el rápido crecimiento de algunas de las empresas más importantes del sector. Año tras año han ido liberando versiones hasta que en Enero de 2019 presentaron Ionic 4 ahora acompañado de **Angular 7** [3]. esta, es la versión que utilizaremos para llevar a cabo nuestra hazaña, cuya potencia y ventajas con respecto a sus versiones anteriores comentaremos en el apartado 1.4.1.

### 1.3– Conocimientos iniciales requeridos

Antes de empezar a guiaros por la senda del desarrollo de *apps* multi-plataforma, me gustaría dejar claro que el primer día que me senté delante de mi ordenador para hacer frente a este proyecto, también fue la primera vez en mi vida que he tenido contacto con tecnologías web de este calibre. Obviamente, durante el transcurso de mi carrera he tenido contacto con las tecnologías web clásicas como lo son HTML, JavaScript, CSS e incluso PHP para comunicar páginas web con acceso a servidor. Pero todo ello a un nivel muy básico, debido a que mi especialidad dentro de Ingeniería Informática fue Computadores, y a que desde hace unos años trabajo como *Senior Firmware Developer* no tuve la oportunidad de adentrarme mucho más en este extraordinario y colosal universo. Por lo tanto, dado que el ecosistema que rodea a Ionic es tan complejo como potente, prácticamente podríamos decir que empecé desde cero, ya que este *framework* implica tener conocimientos en el patrón de diseño *Software MVC*<sup>7</sup>, a su vez, este implica conocer los distintos lenguajes, metodologías y paradigmas que lo conforman. Así que podéis estar tranquilos, puesto que este será el punto del cual partiremos y conforme se vayan viendo esas cuestiones, que requieran de una explicación más profunda, nos detendremos y afianzaremos dichos conocimientos.

Tan solo daré por hecho que la persona que está leyendo este documento, tiene conocimientos básicos de programación (haciendo especial hincapié en el lenguaje HTML) y de las estructuras básicas de código como lo son **for**, **if**, **while**... Recordemos que uno de los propósitos principales de este TFG es que la persona que lo lea aprenda a realizar un aplicación Ionic. Una vez dicho esto, ¡Empecemos!.

### 1.4– Composición

IONIC FRAMEWORK<sup>8</sup> se centra en la experiencia del usuario de *front-end*<sup>9</sup>, o en la interacción con la interfaz de usuario de una aplicación (controles, interacciones, gestos, animaciones). Es relativamente fácil aprender a utilizarlo, sobre todo si el desarrollo tan solo consta de elementos gráficos sin mucha intervención por parte de un *back-end*; además, la documentación oficial aportada desde la web de Ionic[1] es muy completa, clarificativa y está plagada de ejemplos acompañados de representaciones gráficas para cada uno de los componentes<sup>10</sup> propios de Ionic. También se integra perfectamente con librerías nativas (tales como SQLITE para la base de datos, control de orientación de la pantalla, etc...) o con *frameworks* como ANGULAR.

<sup>7</sup>Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Modelos y Controladores) separa la lógica y la parte gráfica(vista) en una aplicación. La mayoría de los frameworks modernos utilizan MVC (o alguna adaptación de este mismo) para su arquitectura, entre ellos podemos mencionar a Ruby on Rails, Django(Python), Angular y muchos otros más...

<sup>8</sup>De ahora en adelante, cada vez que hablemos de Ionic, nos referiremos a la versión 4 del framework, ya que es la que vamos a utilizar, a no ser que especifiquemos otro numero de versión.

<sup>9</sup>El front-end es la parte del desarrollo web que da formato a contenidos, se encarga del aspecto de la web y manipula resultados de datos obtenidos desde el back-end

<sup>10</sup>Bloques de construcción de alto nivel que nos permiten construir rápidamente la interfaz de usuario de nuestra aplicación.

Además, también cabe mencionar que Ionic puede ser usado de forma autónoma o *standalone*, sin necesidad de depender de un *framework* de *front-end* ya que según cuentan en la web oficial del producto, otro de los objetivos que tenían al desarrollarlo era el de eliminar cualquier necesidad de una estructura específica de proyecto a la hora de alojar los componentes. Esto significa que dichos componentes pueden funcionar de forma autónoma, tan sólo añadiendo una etiqueta `<script>` en nuestra página web. Un ejemplo de esto, para exponer de que estamos hablando de una forma menos técnica y más esclarecedora, sería añadir el siguiente código a un simple archivo `index.html`[7]. Dicho código, simplemente contendrá un botón, que al pulsarlo nos mostrará una alerta.<sup>11</sup>

```

1 <!doctype html>
2 <html lang="es">
3 <head>
4   <meta charset="utf-8">
5   <title>Usando Ionic sin ningún Framework</title>
6
7   <!-- Importamos el CSS de Ionic -->
8   <link href="https://unpkg.com/@ionic/core@latest/css/ionic.bundle.css" rel="stylesheet">
9   <!-- Importamos Ionic -->
10  <script src="https://unpkg.com/@ionic/core@latest/dist/ionic.js"></script>
11  <!-- Opcionalmente, importamos los iconos de Ionic -->
12  <script src="https://unpkg.com/ionicicons@latest/dist/ionicons.js"></script>
13 </head>
14 <body>
15
16  <!-- Creamos una simple función Javascript para llamar al alert -->
17  <script>
18    hello = async function () {
19      alert('Hello World!');
20    };
21  </script>
22
23  <!-- Declaramos la aplicación Ionic usando el elemento <ion-app> -->
24  <ion-app>
25    <!-- Como podemos observar, aquí usamos las propiedades CSS importadas arriba
26        -->
27    <ion-content text-center>
28      <h1>Basic usage</h1>
29      <!-- Añadimos un ion-button con un evento al pulsarlo -->
30      <ion-button onclick="hello()">Pulsar</ion-button>
31    </ion-content>
32  </ion-app>
33
34 </body>
</html>

```

Código 1.1: Fichero `index.html` para mostrar alert

---

<sup>11</sup> Componente de Ionic que suple las funciones de las alertas tan comúnmente usadas en las aplicaciones de hoy en día, tanto en Android como en iOS, mostrando un pequeño cuadro de texto por defecto en la parte inferior de la pantalla.

Una vez dejado claro que Ionic no tiene porqué necesitar de un framework para poder desarrollar una aplicación, os diré que aunque esta es una opción perfectamente válida para proyectos de poca envergadura, para el trabajo en el que estamos inmersos quizás nos llevaría a más de un quebradero de cabeza. Por este motivo, dado que nuestro proyecto consta de muchas partes, servicios y componentes, optaremos por usar un *framework* de *front-end* para facilitarnos el trabajo y aumentar nuestra productividad.

Actualmente, IONIC FRAMEWORK tiene integración oficial con el *framework* de *front-end* ANGULAR, pero el soporte para VUE y REACT está en desarrollo. En nuestro caso, dichos *frameworks* no suponen ninguna mejora con respecto a ANGULAR ya que, por el momento, Ionic no ha decidido integrar oficialmente estos frameworks debido a que no todos sus componentes están incluidos en ellos y tan solo nos supondrían más pasos para lograr el mismo objetivo. Por lo tanto, nosotros utilizaremos el **Angular** integrado oficialmente para desarrollar nuestra *app*. Si alguien estuviese interesado en informarse más acerca del asunto, encontrará en la bibliografía[11] un enlace con un artículo muy completo sobre cómo configurar Ionic para funcionar con esas dos alternativas.

Durante los siguientes subapartados, desmenuzaremos IONIC FRAMEWORK en cada una de las partes que componen su ecosistema.

#### 1.4.1. Angular: Conceptos básicos

Como ya hemos mencionado anteriormente, Angular es un *framework open-source*<sup>12</sup> de *front-end* para aplicaciones web. Su código fuente está desarrollado en TYPESCRIPT<sup>13</sup> y es mantenido por el equipo Angular de GOOGLE junto a una gran comunidad de desarrolladores y corporaciones y cumple con la especificación ECMASCIPT v6.<sup>14</sup> [9]

En el apartado 1.2.3 os conté que la primera versión de Ionic iba acompañada de **AngularJS**, y que la última versión en cambio usa **Angular 7**. Antes de entrar en más detalles sobre Angular, para evitar confusiones futuras en caso de que un lector de este trabajo buscase información o ejemplos sobre él, me gustaría dejar claro que pese a compartir el mismo nombre, **no tienen nada que ver** el uno con el otro. Angular es un *framework* desarrollado por el mismo equipo que construyó AngularJS, de forma completamente nueva, escribiéndolo desde cero y con conceptos, arquitectura y formas de trabajar completamente distintas. Angular utiliza un sistema de **inyección de dependencias jerárquico** que incrementa su rendimiento de forma demencial. También implementa la **carga dinámica de librerías** y la detección de cambios basados en **árboles unidireccionales**, lo que también incrementa el rendimiento. Según algunos datos oficiales, Angular puede llegar a ser 5 veces más rápido que AngularJS.[12]

Como dato extra, os expondré algunas de las ventajas de usar Angular (2 o superior) frente a AngularJS, para que podamos hacernos una idea de cuan incompatibles son entre ellos:

- Angular está orientado a **móviles** y tiene mejor rendimiento.

<sup>12</sup>Aplicación de código fuente abierto/disponible para todos los desarrolladores, permitiendo así la colaboración entre ellos a la hora de hacer modificaciones o uso de este.

<sup>13</sup>Es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es una evolución de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases.

<sup>14</sup>Abreviado como ES6 o ES2015. Es el estándar que sigue JavaScript desde Junio de 2015. Hasta ese momento la versión de JS que estábamos usando en nuestros navegadores y Node.js, era la v5. Esta versión trajo consigo cambios en el lenguaje, clases y herencia entre otros.[13]

- Nos ofrece la posibilidad de utilizar **TypeScript**, lo que nos permitirá afrontar proyectos más grandes con un menor esfuerzo.
- Angular se basa en componentes web, con las ventajas que ello supone al adoptar un estándar de futuro. Estos componentes nos ofrecen mucha eficiencia en cuanto a escalabilidad y mantenimiento de la aplicación, ya que al tratarse de una única entidad **reutilizable** en distintas páginas de esta, en caso de que algo falle o querer implementar una mejor, tan solo tendríamos que modificar el código una vez en lugar de replicar el cambio en todos los puntos del código donde este dicha funcionalidad.
- Ha cambiado la sintaxis de muchos mecanismos. Por ejemplo, en el caso de la inyección de dependencias, como en Angular existen las clases, se realiza mediante constructores. Todo esto, lo veremos en mayor profundidad cuando realicemos el **servicio de base de datos** de nuestra aplicación, aportando enormes beneficios en lo que respecta al rendimiento, ya que será el propio Angular quien se encargue de manejar la carga de entidades y servicios mediante el *bootstrapping*<sup>15</sup>

En resumen, al momento de crearse Angular se reescribió todo desde cero en **TypeScript** centrándose en los móviles, una mejor UI y enfocándose en el desarrollo de un código más estructurado, modular y reutilizable. Al mismo tiempo que se produjo esta evolución, Ionic también fue evolucionando, creciendo y perfeccionándose año tras año; desde el antiguo Ionic 1, dando un salto de gigante a Ionic 2 (debido a los cambios para incorporar el por aquellos entonces reciente Angular), hasta el actual Ionic 4 siendo este el más potente, óptimo y consistente de todos. Estos factores, sumados al soporte de cara al futuro, hicieron que me decidiese por esta versión como la mejor para llevar a cabo este proyecto.

#### 1.4.2. Node.js

NODE.JS es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Casi todas las herramientas para proyectos JavaScript modernos están basadas en NODE.JS. Ionic no es una excepción, en nuestro caso, es necesario para interactuar con el ecosistema de Ionic, y lo necesitamos para poder instalar y utilizar el Ionic CLI (Línea de comandos de Ionic). En resumidas cuentas, aparte de todo lo mencionado, Ionic utiliza NODE.JS para ejecutar Javascript del lado del servidor, permitiendo de esta forma que *front-end* y *back-end* queden unidos en un mismo entorno. Para más información acerca de NODE.JS, dada su gran extensión, he dejado un enlace en la bibliografía[14].

#### 1.4.3. NPM

NODE.JS viene incluido con NPM (*Node Package Manager*), un gestor de paquetes para JavaScript que nos permitirá tener cualquier librería disponible con solo una línea de código, NPM nos ayudará a administrar nuestros módulos, distribuir paquetes y agregar dependencias de una manera sencilla. Este, a su vez, como ya mencionamos en el apartado anterior, también nos servirá para instalar el Ionic CLI. Cuando instalamos nuevos paquetes lo que hace NPM es instalarlos de manera local en nuestro proyecto dentro de la carpeta **node\_modules**, pero nosotros podemos decirle que los instale de manera global de

---

<sup>15</sup>Proceso de arranque de la aplicación

ser necesario. Además, existen módulos que ya vienen instalados por defecto en NODE.JS por lo que no será necesario usar NPM para añadirlos, a estos módulos se les conoce como “módulos nativos” [15].

#### 1.4.4. Ionic CLI

El Ionic CLI, o **Línea de comandos de Ionic**, está construido con **TYPESCRIPT** y **NODE.JS**. Soporta **NODE.JS** en versiones superiores o iguales a la 6, pero siempre es recomendable usar el último **NODE LTS<sup>16</sup>** para así asegurarnos una mejor compatibilidad.

Es el método preferido de instalación, ya que ofrece una amplia gama de herramientas de desarrollo y opciones de ayuda a lo largo del proceso. También es la principal herramienta para crear proyectos, añadir componentes y páginas a estos, compilar y ejecutar la aplicación e incluso conectarla a otros servicios, como Ionic Appflow<sup>17</sup>.

#### 1.4.5. Cordova CLI

**APACHE CORDOVA** es una plataforma *Open Source* que permite construir aplicaciones híbridas multi-plataforma móviles utilizando tecnologías web estándar como lo son **HTML5**, **CSS** y **Javascript**. Consiste en un set de librerías Javascript que permite al desarrollador acceder a funciones nativas del dispositivo móvil, tales como la cámara, el acelerómetro, la agenda o el GPS.

Las aplicaciones que utilizan CORDOVA se basan en un archivo **config.xml** común para todas las plataformas, que proporciona información sobre la aplicación y especifica los parámetros que afectan a su funcionamiento, como, por ejemplo, si el dispositivo responde a los cambios de orientación.

Al instalar CORDOVA junto a nuestro Ionic, estaremos instalando también CORDOVA CLI (Línea de comandos de CORDOVA). Esta nos permitirá, entre otras muchas funciones, compilar la aplicación e instalarla en un dispositivo o emulador, añadir las plataformas para las que irá orientada nuestra *app*, instalar plugins nativos e incluso redimensionar de manera completamente automática recursos tales como el ícono de la aplicación o el splash-screen<sup>18</sup> para que se vean bien en todos los tamaños de pantalla.[16]

---

<sup>16</sup> El soporte a largo plazo (en inglés, Long Term Support, abreviadamente, LTS) es un término informático usado para nombrar versiones o ediciones especiales de software diseñadas para tener soportes durante un período más largo que el normal.[9]

<sup>17</sup> Appflow es una plataforma de integración (CI) y despliegue continuo (CD) para los equipos de desarrollo de Ionic. Esta herramienta ayuda a los equipos de desarrollo a crear y lanzar sus aplicaciones iOS, Android y Web de la forma más eficiente posible

<sup>18</sup> Pantalla de inicio usada comúnmente en el desarrollo de aplicaciones móviles, permitiendo mostrar algo “entretenido” al usuario mientras la CPU carga la app en cuestión.

## 1.5– Preparación del entorno de desarrollo

Una vez vistos los principales elementos que componen IONIC FRAMEWORK, tenemos un boceto del mundo que lo rodea. Ahora, podemos hacernos una idea de todas las herramientas de las que disponemos para llevar a cabo nuestro propósito de crear una app multi-plataforma híbrida para dispositivos móviles. Por lo tanto, ya es hora de entrar en tecnicismos y de ponernos manos a la obra con la preparación del entorno de desarrollo que utilizaremos para desarrollar nuestro proyecto.

Antes de ponernos a instalar este entorno, también quería dejar claro que pese a que la aplicación está creada para ser compilada tanto en Android como en iOS, no me ha sido posible compilarla para este último sistema operativo, ya que no poseo un ordenador de sobremesa de la marca APPLE<sup>19</sup> y tampoco me ha sido posible conseguir uno. Pese a ello, y aunque todas las capturas e instrucciones que os muestre serán tomadas desde un ordenador con Windows 10 y un móvil con sistema operativo Android, trataré de ilustrarlos todos los pasos necesarios para llevar a cabo la instalación desde macOS en un dispositivo con iOS, haciendo distinción entre las plataformas cuando sea necesario.

### 1.5.1. Instalar Node.js y NPM

#### Windows y macOS

El primer paso, será instalar **Node.js**, el cual como mencionamos anteriormente viene acompañado de **NPM**. Podremos encontrar un instalador del *software* para todas las plataformas dirigiéndonos a su sitio web oficial (<https://nodejs.org/en/download/>).

Tras descargar el instalador lo ejecutaremos. Al hacer esto se nos abrirá una ventana que nos dará la bienvenida al proceso de instalación. Tan solo tendremos que seguir todas las instrucciones, e **instalar en la ruta por defecto** que nos sugiera el instalador. Una vez completados todos los pasos y tras clicar en finalizar, tendremos instalado **Node.js** en nuestro equipo.

### 1.5.2. Control de versiones

En mi caso, opté por instalar GIT, no es obligatorio, pero es muy útil a la hora de desarrollar una aplicación de este calibre. GIT, para quien no lo sepa, es un *software* de **control de versiones** diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente[9]. Nos es de mucha utilidad para, por ejemplo, al acabar un arduo día de desarrollo hacer un *commit* (subida del código) a un repositorio GIT y de esta forma almacenar el estado actual del proyecto bajo una etiqueta, fecha y hora concretas.

Un ejemplo de las ventajas que esto puede aportarnos sería el siguiente: Imaginemos que hace una semana, subimos una versión funcional de nuestro proyecto a nuestro repositorio GIT. Ahora imaginemos que durante la semana actual modificamos la anterior versión y realizamos otra subida del código, dándonos cuenta posteriormente de que existe un error en nuestra *app*. Pues bien, gracias a GIT no tendríamos que preocuparnos de haber estropeado nuestro código. Tendríamos varias opciones para solucionarlo, podríamos volver a la anterior versión completamente funcional y testeada (esto solo es recomendable en un

---

<sup>19</sup>Los de Cupertino son conocidos por su restrictivas condiciones, entre ellas encontramos la obligación de poseer un PC con el sistema operativo macOS para compilar aplicaciones de iOS.

caso muy crítico), o bien ser más prácticos y usar las herramientas (como la comparadores de código entre versiones) que aporta la línea de comandos de la consola de GIT o los clientes de escritorio de GIT.

En mi caso, yo opté por usar el cliente TORTOISEGIT para el repositorio de GIT local, y GITHUB como repositorio web remoto, para evitar perdidas en el caso de sucederle algo a mi ordenador personal alojando allí todos mis cambios y versiones del código.

Otra utilidad de GIT es que algunos IDE ofrecen la posibilidad de integración con GIT, por lo que las etiquetas de las subidas y cambios nos aparecerían facilitando aun más nuestro trabajo al saber cuándo y por qué se hizo cada cambio.

Para instalar GIT tan solo tenemos que ir a su web oficial (<https://git-scm.com>), descargar el instalador para la plataforma desde la que estemos trabajando y seguir las instrucciones de instalación eligiendo las opciones predeterminadas (en caso de tener conocimientos sobre GIT las que se deseen).

## Windows

TORTOISEGIT solo esta disponible para Windows y podemos encontrarlo en la siguiente dirección <https://tortoisegit.org/download/>. Dado que la instalación para un uso tan básico es sencilla no entrará en más detalles acerca de ella. Tan solo seguiremos las instrucciones hasta finalizar la instalación. Una vez instalado, iremos a la carpeta sobre la que deseemos crear nuestro repositorio GIT local y al clicar sobre ella con el botón derecho de nuestro ratón aparecerá la siguiente ventana:

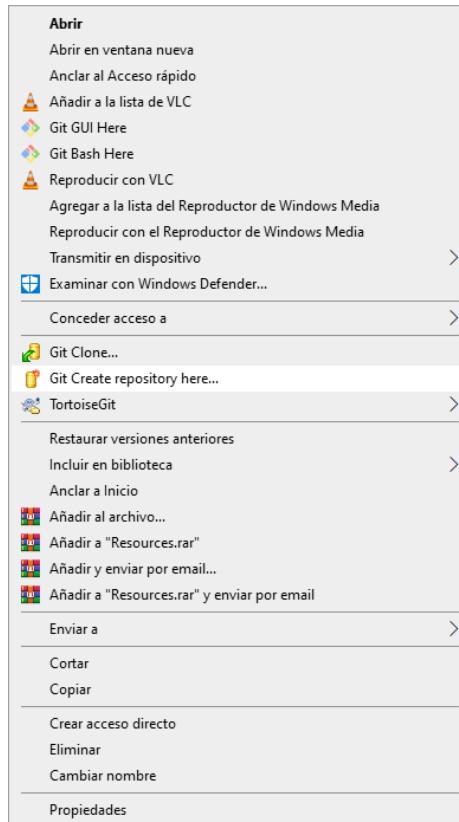


Figura 1.1: Creación de Repositorio GIT local

Elegiremos la opción (*Git create repository here...*). Una vez hecho esto, habremos creado un repositorio GIT en dicho directorio, y tras reiniciar el PC, aparecerán *checks* de color verde indicando que todo está actualizado. En caso de modificar algún archivo aparecería un símbolo de exclamación de color rojo indicando que algo no corresponde con la última versión en el repositorio.

Para finalizar, en caso de que deseemos subir nuestro proyecto a <https://github.com> y mantenerlo actualizado para una mayor seguridad de este, tan solo tendremos que crear un usuario (en caso de que no tenerlo), logearnos y añadir la carpeta de nuestro proyecto. Tras esto, tenemos dos opciones:

Hacer un *git clone or download* desde el proyecto recientemente añadido en la web de GITHUB.

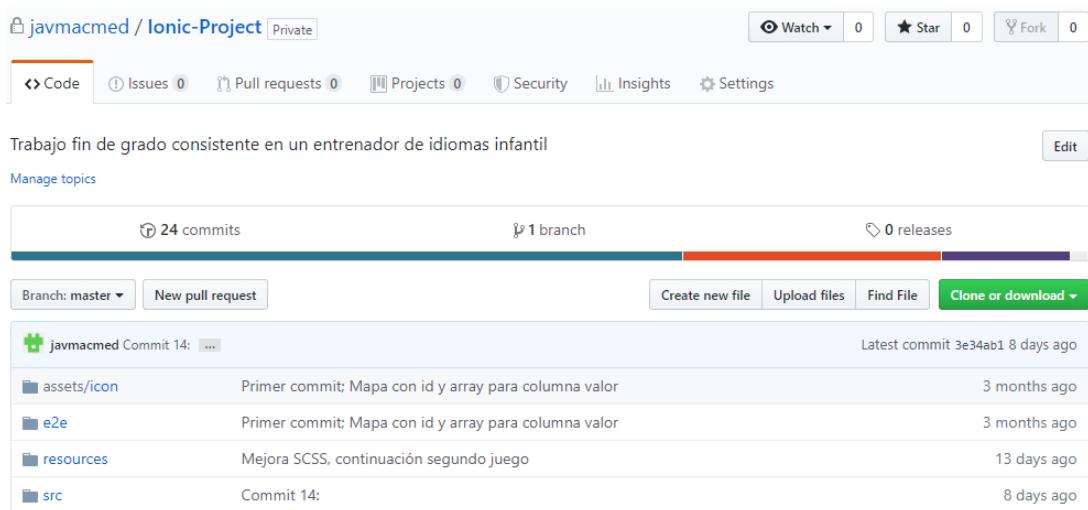


Figura 1.2: Clone Repositorio GITHUB

O usar la opción (*Git sync...*) del menú contextual de TORTOISEGIT que aparece al clicar con el botón derecho sobre la carpeta previamente indicada como repositorio GIT.

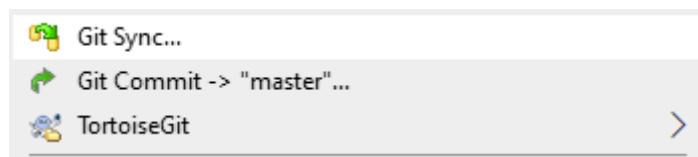


Figura 1.3: Sincronización con repositorio web GIT

En caso de alguna duda con el uso de GIT, os recomiendo seguir el enlace en la bibliografía [4].

## Apple

En el caso de realizar la instalación desde un Mac, tendréis que instalar alguno de los clientes de este enlace <https://www.fossmint.com/gui-git-clients-for-mac/>.

### 1.5.3. Visual Studio Code

La web oficial de Ionic nos recomienda en primera instancia el uso de su IDE oficial IONIC STUDIO. Dado que para poder usarlo es necesario pagar una cuota mensual, traté de ponerme en contacto con ellos desde mi correo universitario a través de una dirección de correo electrónico para consultas de precios especiales (por ejemplo, para estudiantes). Al escuchar mi propuesta de hacer este proyecto y de que eso les haría publicidad gratuita de cara a un futuro me respondieron que se volverían a poner en contacto conmigo, dado que les había encantado mi idea... aunque al final parece que no fue así dado que jamás volvieron a contactar conmigo (tal y como imaginé que ocurriría). Por lo que opté por la segunda recomendación dada en la web oficial, esta fue VISUAL STUDIO CODE.

VISUAL STUDIO CODE es un **entorno de desarrollo** integrado, en inglés *Integrated Development Environment* (IDE). Se trata de una aplicación informática que proporciona servicios integrales para facilitarnos el desarrollo de software. Normalmente, un IDE consiste en un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código (*IntelliSense*). Algunos IDE contienen un compilador, un intérprete, o ambos, tales como NETBEANS y ECLIPSE; otros no, tales como SHARPDEVELOP y LAZARUS [9].

Tras haberlo usado, solo puedo hablar maravillas de VISUAL STUDIO CODE:

- Es rápido, agradable e intuitivo.
- Tiene una **consola integrada configurable** (ctrl + ñ desde el IDE) en la que podremos establecer la *shell* que deseemos, ya bien sea cmd, powershell, Git Bash (la cual prefiero debido a la comodidad de trabajar con Bash). Desde ella podremos ejecutar todos los comandos (NPM, IONIC CLI y CORDOVA CLI) que deseemos.
- Nos ofrece la posibilidad de instalar plugins de todos los tipos y colores imaginables, estos nos permitirán literalmente evolucionar el IDE trasformándolo en un entorno de desarrollo totalmente personalizado y orientado hacia nuestras necesidades. A través de este enlace en la bibliografía [20] podréis encontrar una lista con la mayoría de los plugins recomendados para trabajar con Ionic y Angular 7 que yo mismo he usado (entre ellos, hay un plugin para el uso integrado de GIT).
- Tiene tanta documentación, comunidad y tutoriales para cada uno de sus plugins y funcionalidades como las que podría tener cualquier *framework*, lenguaje de programación, etc...
- Otra de sus virtudes, es la de tener un depurador configurable muy rico en funciones, el cual no llegué a usar, ya que siempre depuré sobre la aplicación ejecutada en el mismo dispositivo usando GOOGLE CHROME como veremos más adelante.

Para instalar VS CODE tan solo tendremos que navegar hacia su *website* oficial <https://code.visualstudio.com>. Tras descargar el instalador para la plataforma en la que nos encontramos, clicaremos en él y seguiremos las instrucciones de instalación hasta finalizarla.

Para añadir un proyecto y trabajar con él, tan solo tendríamos que abrir el IDE, y en la pestaña superior izquierda llamada Archivo, pulsar la opción Abrir Carpeta y seleccionar el raíz de nuestro proyecto.

### 1.5.4. Instalar Ionic y Cordova

Una vez instalados NODE.JS y NPM, tenemos puertas abiertas para instalar el resto de nuestro entorno. La instalación de IONIC FRAMEWORK y de CORDOVA CLI no podría ser más fácil, simplemente hay que ejecutar el comando `npm install -g ionic cordova` desde la *shell* (consola) que prefiramos, yo use la integrada en VS Code (recordemos que hay que pulsar **ctrl + ñ** para visualizar dicha consola). Pero aún no podremos empezar a desarrollar nuestra aplicación, primero es necesario configurar unas pocas de cosas más.

### 1.5.5. Configurar entorno para Android desde Windows

#### Instalar Android SDK

En primer lugar descargaremos el *software* desde la página oficial del mismo:

- <https://developer.android.com/studio>

Una vez descargado, procederemos con la instalación, clicando sobre él nos aparecerá la siguiente ventana:



Figura 1.4: Configuración: Inicio Instalación Android Studio

Pulsaremos **Next** y nos aparecerá esta otra ventana:

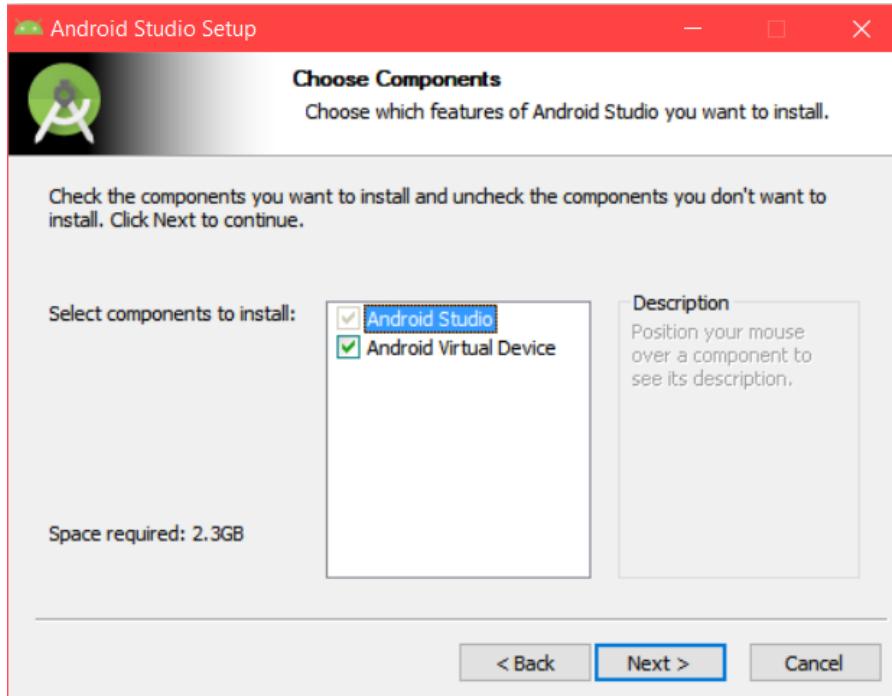


Figura 1.5: Configuración: Elección de componentes

Dejaremos marcada la opción **Android Virtual Device** para asegurarnos de que tenemos un dispositivo virtual disponible en caso de que quisieremos usarlo para probar la *app*.

Seguiremos los pasos del asistente hasta completar la instalación de ANDROID STUDIO; dejaremos marcada la opción **Start Android Studio** y clicaremos en **Finish**.

Tras esto, se nos abrirá el IDE y, al seguir los pasos del asistente nos mostrará la siguiente ventana:

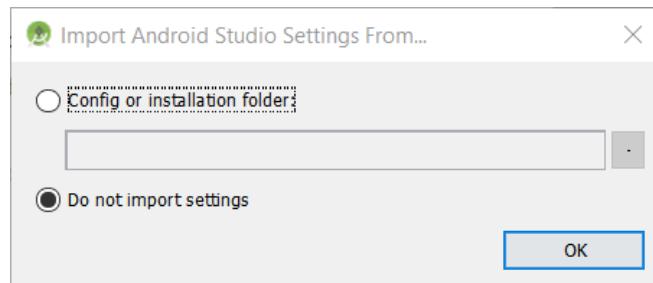


Figura 1.6: Configuración: Importar configuración de Android Studio

En caso de que entendamos como funciona el IDE y tengamos una configuración previa la usaremos, en caso contrario dejaremos marcada la opción de la captura.

Al pulsar **OK** visualizaremos una pantalla en la que nos dan la bienvenida al *setup* de ANDROID STUDIO:

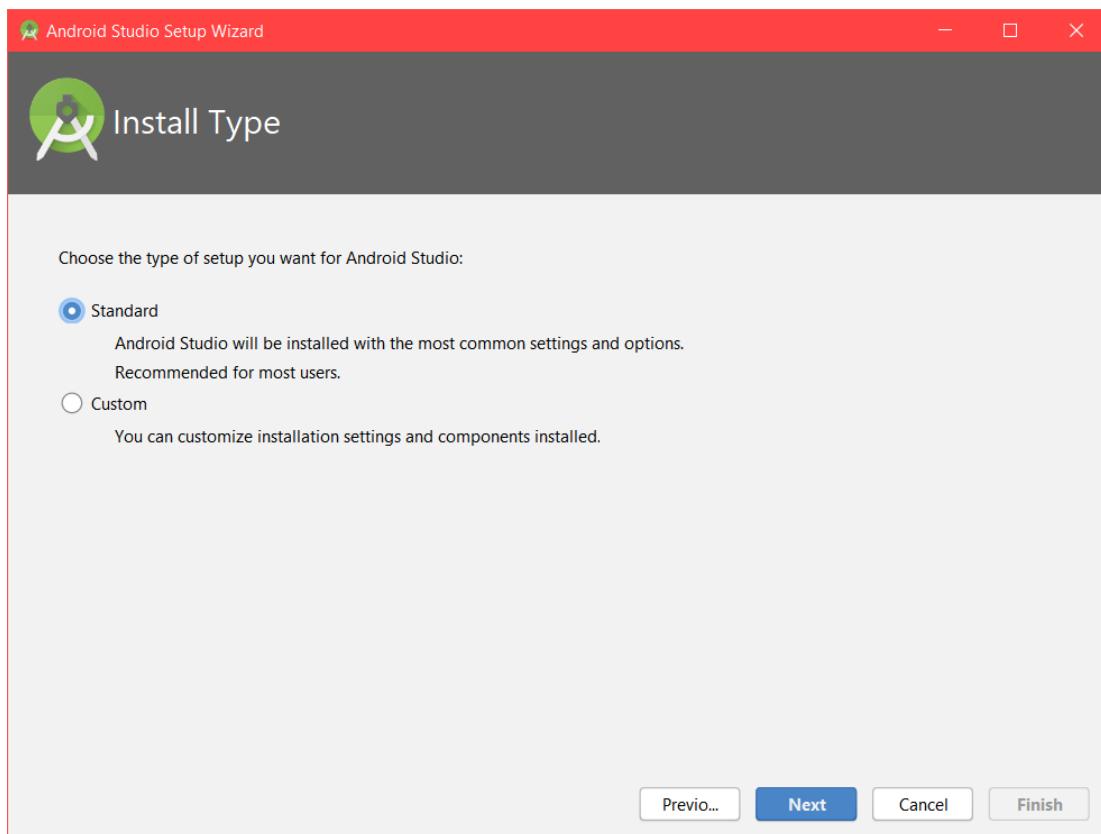


Figura 1.7: Configuración: Setup de Android Studio

Aquí dejaremos marcada la opción **Standard**, ya que nos instalará la última versión del SDK de Android y algunas herramientas que instalaríamos igualmente como por ejemplo el **Android Emulator** o **Android SDK Build-Tools**.

Antes de seguir, me gustaría puntualizar algo acerca de este proceso. La ruta por defecto del SDK que utiliza el instalador es C:\Users\myUser\AppData\Local\Android\Sdk. En caso de que nuestro usuario de Windows tenga espacios en el nombre, esto dará lugar a un error. La solución, por muy mala que sea, sería trasladar la ruta de instalación del SDK a otra parte que no requiera que el usuario de Windows quede incluido en dicha ruta.

Una vez hayamos instalado satisfactoriamente ANDROID STUDIO, junto al SDK de Android, el siguiente paso será proceder con la configuración del SDK requerida para nuestra aplicación Ionic. Por lo tanto, clicaremos en el ícono del IDE que se nos habrá creado en el escritorio.

Al hacer esto nos aparecerá una ventana dándonos la bienvenida a Android Studio. En la parte inferior derecha de esta ventana veremos una **rueda dentada** de configuración. Al clicarla nos aparecerá este menú contextual:

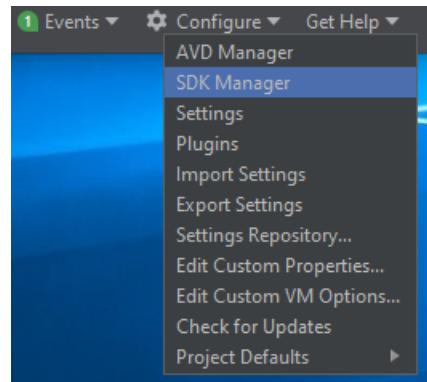


Figura 1.8: Configuración: SDK Manager

Tras pulsar la opción SDK Manager nos aparecerá otra ventana:

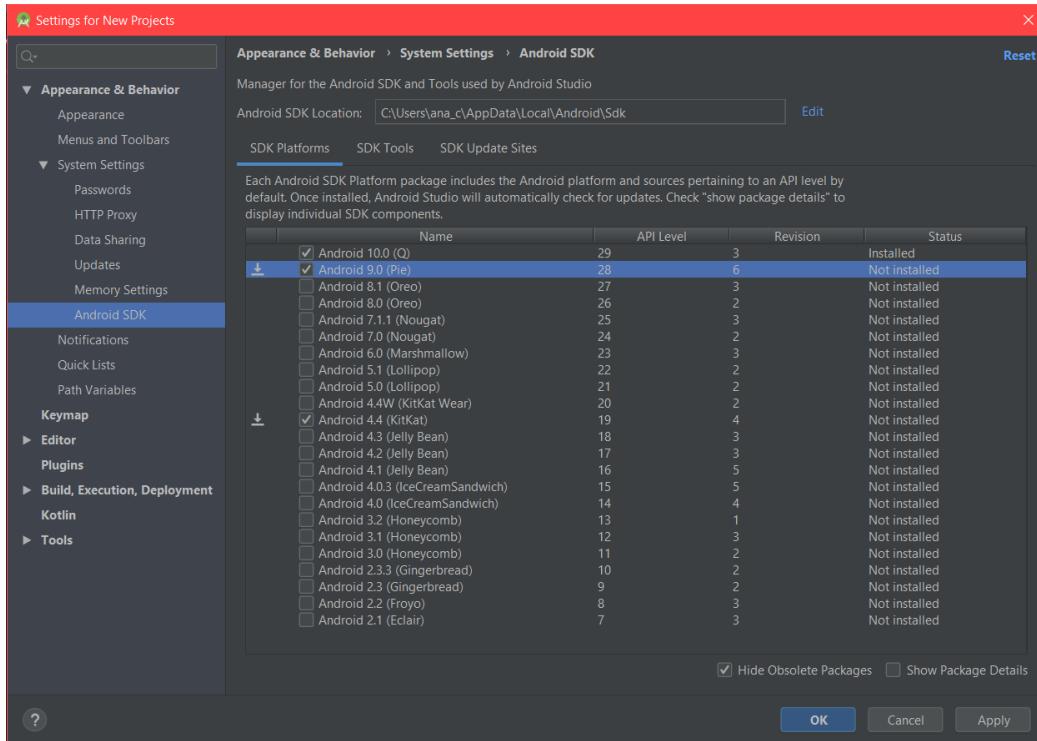


Figura 1.9: Android SDK Platforms

Ahora, nos aseguraremos de que tenemos instalados los siguientes paquetes: [17]

- **Android Platform SDK**, nuestras versiones objetivo de Android, es decir, las **API Level 19** y la **28**.
- **Android SDK build-tools** versión 19.1.0 o mayor (bajo la pestaña **SDK Tools**).
- **Android Support Repository** (bajo la pestaña **SDK Tools**, según el equipo de GOOGLE en las últimas versiones no es necesario instalarlo, yo lo tengo, pero en algunos equipos ya no aparece como opción).

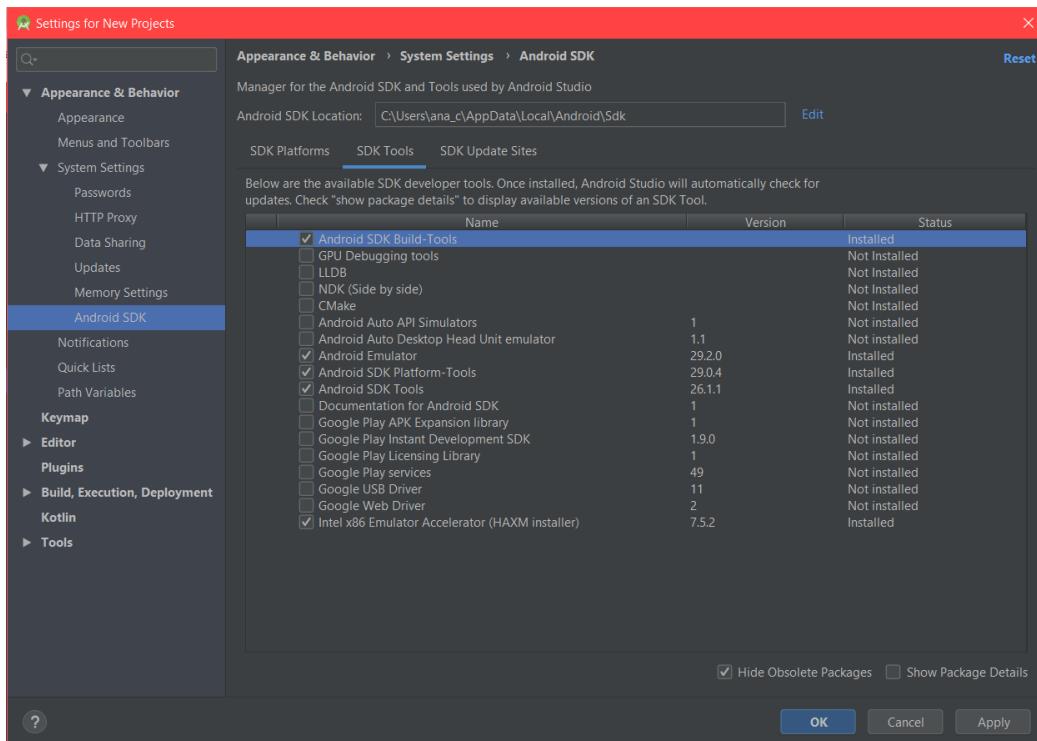


Figura 1.10: Android SDK Tools

## Instalar Java JDK

Para llevar a cabo la compilación en Android se debe instalar JDK 1.8 (Java 8u211 o similar, no Java 9, 10, 11 o superior dado que no son compatibles con Ionic) desde el link <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

En dicha página, descargaremos Java JDK 1.8 desde esta ventana:

Java SE Development Kit 8u221		
You must accept the <a href="#">Oracle Technology Network License Agreement for Oracle Java SE</a> to download this software.		
<input checked="" type="radio"/> Accept License Agreement	<input type="radio"/> Decline License Agreement	
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.9 MB	<a href="#">jdk-8u221-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	69.81 MB	<a href="#">jdk-8u221-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	174.18 MB	<a href="#">jdk-8u221-linux-i586.rpm</a>
Linux x86	189.03 MB	<a href="#">jdk-8u221-linux-i586.tar.gz</a>
Linux x64	171.19 MB	<a href="#">jdk-8u221-linux-x64.rpm</a>
Linux x64	186.06 MB	<a href="#">jdk-8u221-linux-x64.tar.gz</a>
Mac OS X x64	252.52 MB	<a href="#">jdk-8u221-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	132.99 MB	<a href="#">jdk-8u221-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	94.23 MB	<a href="#">jdk-8u221-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	133.66 MB	<a href="#">jdk-8u221-solaris-x64.tar.Z</a>
Solaris x64	91.95 MB	<a href="#">jdk-8u221-solaris-x64.tar.gz</a>
Windows x86	202.73 MB	<a href="#">jdk-8u221-windows-i586.exe</a>
Windows x64	215.35 MB	<a href="#">jdk-8u221-windows-x64.exe</a>

Figura 1.11: Instalación Java JDK

Será necesario crearnos un usuario de Oracle para descargar el *software*. Una vez descargado seguir los pasos por defecto del asistente de instalación.

### Instalar Gradle

Descargaremos la última distribución **completa** de GRADLE desde su web oficial <https://gradle.org/>. La podremos encontrar en la sección de instalación manual, tal y como se ve en la siguiente captura, y pulsaremos sobre el enlace que pone *complete*:

#### Installing manually

##### Step 1. Download the latest Gradle distribution

The current Gradle release is version 5.6.2, released on 05 Sep 2019. The distribution zip file comes in two flavors:

- [Binary-only](#)
- [Complete, with docs and sources](#)

If in doubt, choose the binary-only version and browse [docs and sources online](#).

Figura 1.12: Configuración: Instalando distribución de Gradle

Descomprimiremos el contenido del archivo .zip descargado desde la web de Gradle en C:\Gradle (crear directorio).

### Establecer las variables del entorno

Las **variables de entorno** son cadenas de texto que **contienen información** acerca del entorno para el sistema y el usuario que ha iniciado sesión en ese momento. Algunos programas de software usan esta información para determinar **dónde se colocan los archivos** (como los archivos temporales) [21].

A continuación, os mostraré algunas de las variables del sistema que tengo yo. Son fundamentales para que nuestro entorno de trabajo funcione correctamente, sobre todo

para poder ejecutar comandos de consola como `adb` y `javac`, que nos permiten interactuar con dispositivos nativos y compilar nuestro proyecto respectivamente.

Para modificar las variables de entorno del sistema en Windows 10, tan solo tendremos que ir a la barra de búsqueda, a la derecha del icono de Windows, en la parte inferior izquierda del escritorio, y buscar **Variables de entorno**; entonces nos aparecerá un resultado en el que pondrá **Editar las variables de entorno del sistema**. Al clicar en dicha opción, nos aparecerá una ventana llamada **Propiedades del sistema**:

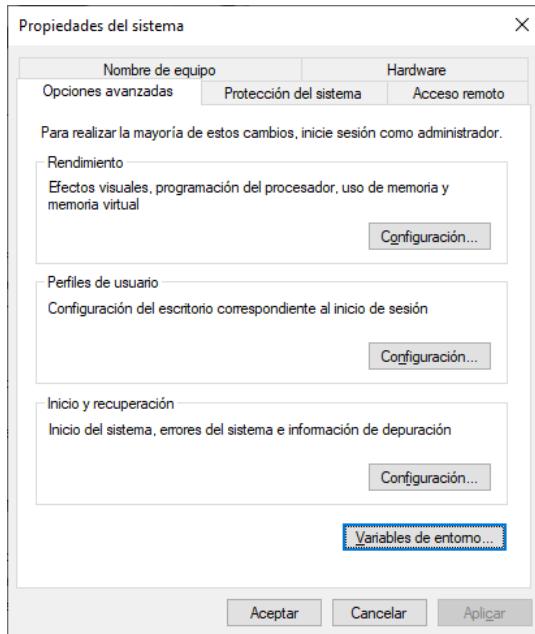


Figura 1.13: Propiedades del sistema

Ahora, en la pestaña **Opciones avanzadas** pulsaremos el botón **Variables de entorno...** y nos aparecerá la siguiente pantalla:

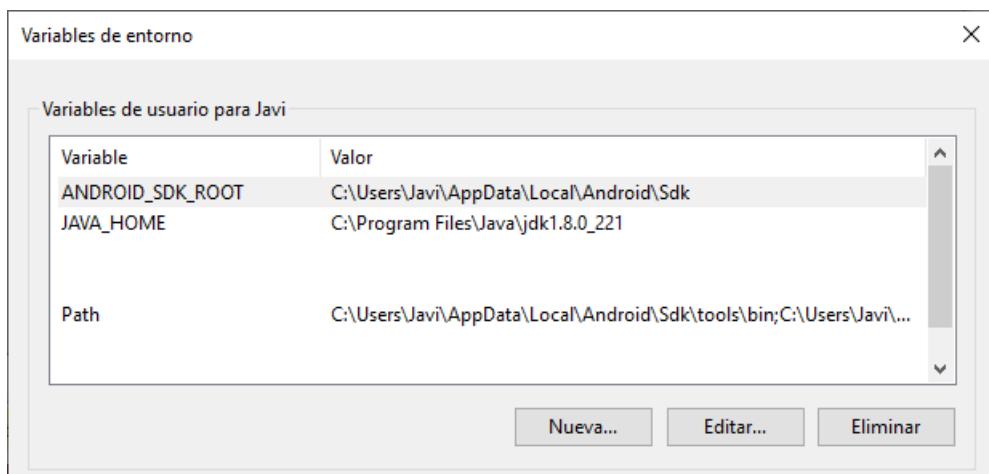


Figura 1.14: Variables de entorno

En esta ventana, podemos observar las variables de entorno para el usuario con el que nos hemos logueado en Windows. Es el momento de recordar/buscar las rutas que establecimos para la instalación del SDK de Android y del JDK de Java, e introducirlas en las **nuevas variables** llamadas **ANDROID\_SDK\_ROOT** y **JAVA\_HOME** que crearemos de forma similar a como se ve en la última captura.

Para finalizar, en la variable **Path** introduciremos lo siguiente (respetando los ;):

- %ANDROID\_SDK\_ROOT%\tools\bin;
- %ANDROID\_SDK\_ROOT%\platform-tools;
- %ANDROID\_SDK\_ROOT%\emulator;
- %JAVA\_HOME%\bin;
- Para GRADLE también es necesario la ruta de su directorio de binarios:  
C:\Gradle\carpetaDistribucionDescomprimida\bin;.

### 1.5.6. Configurar entorno para iOS desde macOS

Como ya hemos comentado anteriormente, APPLE es bastante restrictivo con respecto a que se desarrollen aplicaciones para alguno de sus SO desde otros sistemas operativos. Aun así, en este apartado, suponiendo que tenemos un PC con macOS, trataremos de indicar todos los pasos necesarios a seguir para configurar el entorno de desarrollo con el propósito de generar una aplicación capaz de ser ejecutada en un dispositivo con iOS.

El primer paso sería instalar XCODE<sup>20</sup>. Podremos descargarlo desde la APP STORE [18], tan solo siendo necesario un *Apple Account* o ID de APPLE, según prefiramos llamarlo. Una vez XCODE haya sido instalado, nos aseguraremos de que las herramientas de la línea de comandos están seleccionadas para su uso: \$ `xcode-select --install`.

Según la web oficial de Ionic[5], todas las aplicaciones iOS deben estar firmadas por código, incluso para su desarrollo. Afortunadamente, XCODE lo hace fácil con la firma automática de código. El único requisito es tener un ID de Apple.

Tan solo tendremos que abrir XCODE y navegar hasta **Preferencias** e ir a la pestaña **Cuentas**. Ahí, añadiremos un ID de Apple en caso de no haber ninguno en la lista. Una vez que hayamos iniciado sesión, aparecerá un equipo personal en la lista de equipos del ID de Apple.

Será necesario instalar dos utilidades más para poder desplegar *apps* en iOS, ya bien sea para un simulador **ios-sim** o bien directamente sobre un dispositivo con iOS **ios-deploy**. Se pueden instalar globalmente con npm:

- `npm install -g ios-sim`.
- `npm install -g ios-deploy`.

---

<sup>20</sup>IDE para crear aplicaciones iOS nativas. Incluye el SDK de iOS y las herramientas de línea de comandos de XCODE.

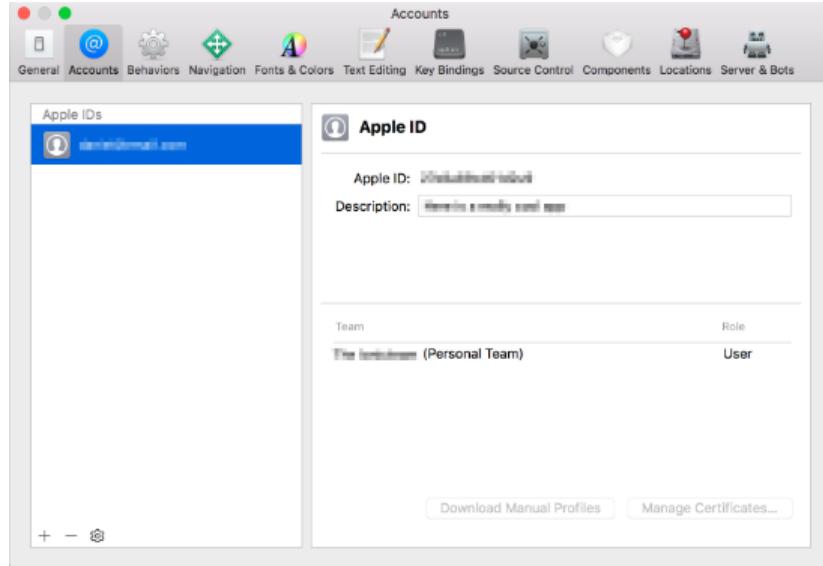


Figura 1.15: Creación de equipo de desarrollo

### 1.5.7. Tutoriales de ayuda

Ante cualquier tipo de duda o dificultad para dichas instalaciones, en la página oficial de **Ionic** podemos consultar las siguientes direcciones:

- <https://ionicframework.com/docs/installation/environment>
- <https://ionicframework.com/docs/installation/android>
- <https://ionicframework.com/docs/installation/ios>

### 1.5.8. Compilación y ejecución en un dispositivo Android

Antes de compilar nuestro proyecto, será necesario asegurarnos de que hemos informado a CORDOVA[17] de que queremos añadir la plataforma Android para poder compilarla. Para ello utilizaremos el comando: `ionic cordova prepare android` .

The terminal window shows the following output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\ana_c\Desktop\Macarro's Proyect\SayMyName> ionic cordova prepare android
  Creating ./www directory for you - done!
? Platform android is not installed! Would you like to install it? (Y/n) [Y]

```

Figura 1.16: Ionic Cordova Prepare

Al escribir Y se ejecutará el comando y el proyecto ya estará preparado para ser compilado para Android.

### Adaptación a múltiples tamaños de pantalla

Una vez hecho esto, pediremos a Cordova que **redimensione** los archivos dentro del directorio *resources* bajo el raíz de nuestro proyecto. Así lograremos que elementos como el ícono de la aplicación y el *Splash Screen* se visualicen correctamente en todos los tamaños de pantalla desde pequeños *smartphones* a grandes *tablets*. Es tan fácil como usar el comando **ionic cordova prepare android** (aunque en el proyecto que tengo subido a GITHUB y del que dispondréis ya están creados).

### Alternativas de compilación no disponibles

Ionic ofrece una utilidad capaz de desplegar aplicaciones instantáneamente llamada **DEVAPP**, esta ofrece una vista en tiempo real de los cambios a medida que se realizan, y posee una rica biblioteca de plugins nativos preinstalados para probar las características nativas de la aplicación. Es una pena que entre esos plugins no se encuentre uno de los plugins de **CORDOVA** que hemos utilizado, en concreto el plugin se llama **SQLLitePorter**. este nos permite cargar en la base de datos de la aplicación tablas a modo de semilla para iniciar la aplicación con contenido base a partir de un fichero con extensión **.sql**. Por lo que tendremos que prescindir del uso de esta utilidad.

Otra utilidad ofrecida por Ionic de la que no podremos disfrutar al completo será la ofrecida por el comando **ionic lab**, al igual que nos ocurre con **ionic serve** ya que usamos una base de datos SQLite, dado que estos comandos inician un servidor web de desarrollo local para el desarrollo/prueba de aplicaciones y este no es capaz de entender las secuencias de SQLite pensadas para ser ejecutadas de forma nativa por el dispositivo, y nos dará error.

Una solución para este problema sería usar *mocks* o traducciones de las llamadas a esta librería SQLite mediante llamadas a funciones JavaScript que realicen la misma función que las llamadas nativas, siendo entendidas perfectamente esta vez por el servidor web. Esto no ha sido incluido en nuestro proyecto, pero se detallará más en profundidad en el último capítulo de conclusiones y mejoras pendientes.

### Compilación

Para **compilar** nuestro proyecto para la plataforma Android, haremos uso conjunto del **IONIC CLI** y del **CORDOVA CLI**, utilizando el comando **ionic cordova build android** generaremos el apk en la siguiente ruta por defecto:

`sayMyName\platforms\android\app\build\outputs\apk\debug`.

```
BUILD SUCCESSFUL in 2m 38s
42 actionable tasks: 42 executed
Built the following apk(s):
C:\Users\ana_c\Desktop\Macarro's Project\SayMyName\platforms\android\app\build\outputs\apk\debug\app-debug.apk
PS C:\Users\ana_c\Desktop\Macarro's Project\SayMyName> []
```

Figura 1.17: Ionic Cordova Build

## Despliegue

En caso de que queramos **compilar y a su vez correr** la *app* en un dispositivo móvil Android, conectaremos el dispositivo al PC mediante un cable USB y, con el **modo desarrollador activado** en el móvil (puede variar en función de la marca del dispositivo) tal y como se explica en el enlace de la bibliografía [22], **activaremos la depuración USB** y la **instalación desde orígenes desconocidos**. Una vez hecho esto, nuestro dispositivo estará preparado para instalar una aplicación no descargada desde la PLAY STORE, el comando a utilizar sería **ionic cordova run android --device**, siempre encontrándonos en la raíz del proyecto sea cual sea la consola que utilicemos; daremos permisos cuando se nos pidan y... ¡¡Conseguido!! *App* ejecutándose en nuestro móvil/tablet.

O... también podría ocurrir que nos apareciese este error:

```
BUILD SUCCESSFUL in 2m 38s
42 actionable tasks: 42 executed
Built the following apk(s):
    C:\Users\ana_c\Desktop\Macarro's Proyect\SayMyName\platforms\android\app\build\outputs\apk\debug\app-debug.apk
PS C:\Users\ana_c\Desktop\Macarro's Proyect\SayMyName> ionic cordova run android --device
[ERROR] native-run was not found on your PATH. Please install it globally:
    npm i -g native-run
PS C:\Users\ana_c\Desktop\Macarro's Proyect\SayMyName>
```

Figura 1.18: Ionic Cordova Run Error

Pero debemos estar tranquilos, pues tiene fácil solución, tan solo ejecutar el comando NPM que nos aparece en el mensaje de error.

```
PS C:\Users\ana_c\Desktop\Macarro's Proyect\SayMyName> npm i -g native-run
C:\Users\ana_c\AppData\Roaming\npm\native-run -> C:\Users\ana_c\AppData\Roaming\npm\node_modules\native-run\bin\native-run
+ native-run@0.2.8
added 30 packages from 33 contributors in 2.927s
PS C:\Users\ana_c\Desktop\Macarro's Proyect\SayMyName>
```

Figura 1.19: Ionic Cordova Run Solución

Ahora, si que podremos **ejecutar ionic cordova run android --device** sin problemas junto a todos los comandos relacionados con la interacción con el dispositivo nativo.

Otra forma de probar nuestra aplicación sería utilizando un emulador. Para ello podemos crear un dispositivo virtual haciendo uso de la herramienta de creación de dispositivos virtuales dada por ANDROID STUDIO. A la hora de hacerlo, hemos de tener en cuenta que la versión del SDK de Android instalada en el dispositivo debería estar entre el mínimo y el máximo que hemos configurado en el fichero **config.xml** de nuestra aplicación.

Cuando hayamos creado el dispositivo, lo ejecutaremos y esperaremos a que nos muestre el escritorio de Android. Una vez realizado este procedimiento, probar nuestra *app* no podría ser más sencillo, tan solo tendremos que usar el comando:

**ionic cordova run --emulator.**

Quizás os estéis preguntando, bueno, pero si utilizamos el comando **ionic cordova build android** cómo hacemos para introducir el binario generado en el dispositivo. La respuesta, es que para eso también tenemos un comando:

**adb install -r ./platforms/android/app/build/outputs/apk/debug/app-debug.apk**

Por último, si deseásemos depurar la *app* previamente compilada e instalada en el dispositivo, utilizaremos la herramienta de **GOOGLE chrome://inspect**, como veremos a fondo más adelante.

### 1.5.9. Compilación y ejecución en un dispositivo iOS

Antes de que las aplicaciones puedan desplegarse en simuladores y dispositivos iOS, debe configurarse el proyecto nativo [6].

Al igual que hicimos para Android, antes de compilar nuestro proyecto, será necesario asegurarnos de que hemos informado a CORDOVA de que queremos **añadir la plataforma iOS** para poder compilarla. Para ello utilizaremos el comando: `ionic cordova prepare ios`. De esta forma, la plataforma iOS quedará instalada creando el directorio **ios** en **platforms**.

#### Configurar Xcode

En primer lugar, abriremos el programa XCODE. Luego iremos a **File** y clicaremos en **Open**. Ahora, seleccionaremos el directorio **platforms/ios** de nuestra aplicación [18].

En **Project Navigator**, seleccionaremos la raíz del proyecto para abrir el editor de proyectos. En la sección **Identity**, comprobaremos que el **Package ID** que se ha configurado en **config.xml** (Texto en atributo id) coincide con el **Bundle Identifier**.

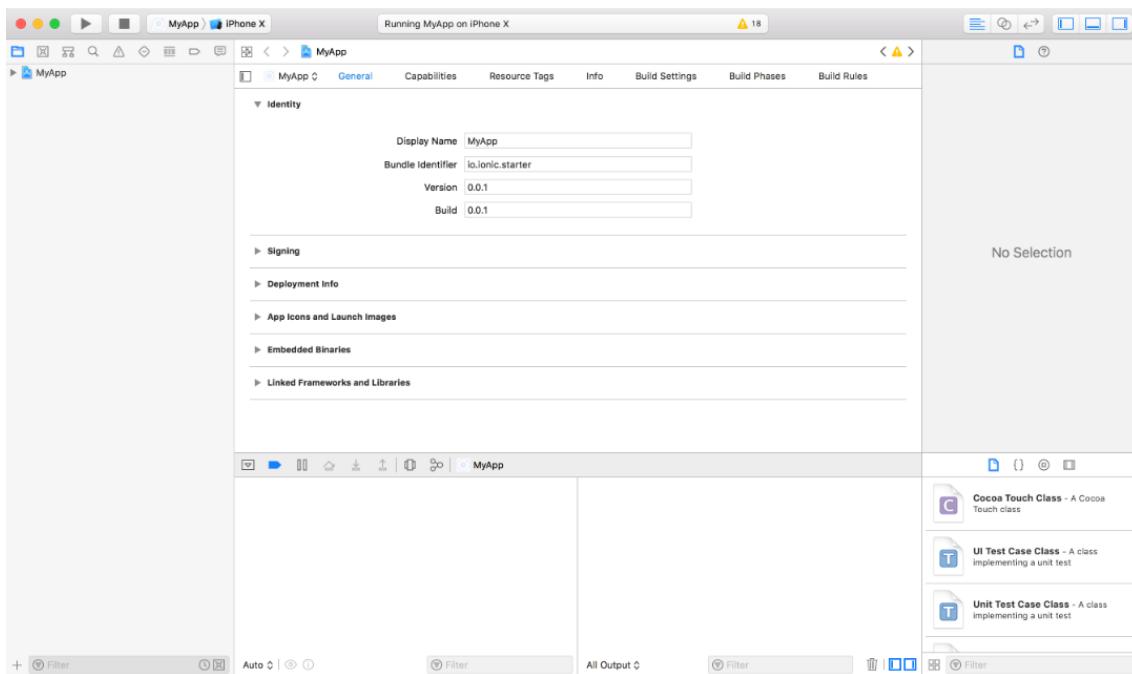


Figura 1.20: Configurando Xcode

En el mismo editor de proyecto, en la sección **Signing**, nos aseguraremos de que la función de gestión automática de firmas esté activada. A continuación, seleccionaremos un equipo de desarrollo. Una vez seleccionado, XCODE intentará preparar automáticamente el perfil de aprovisionamiento y la firma.

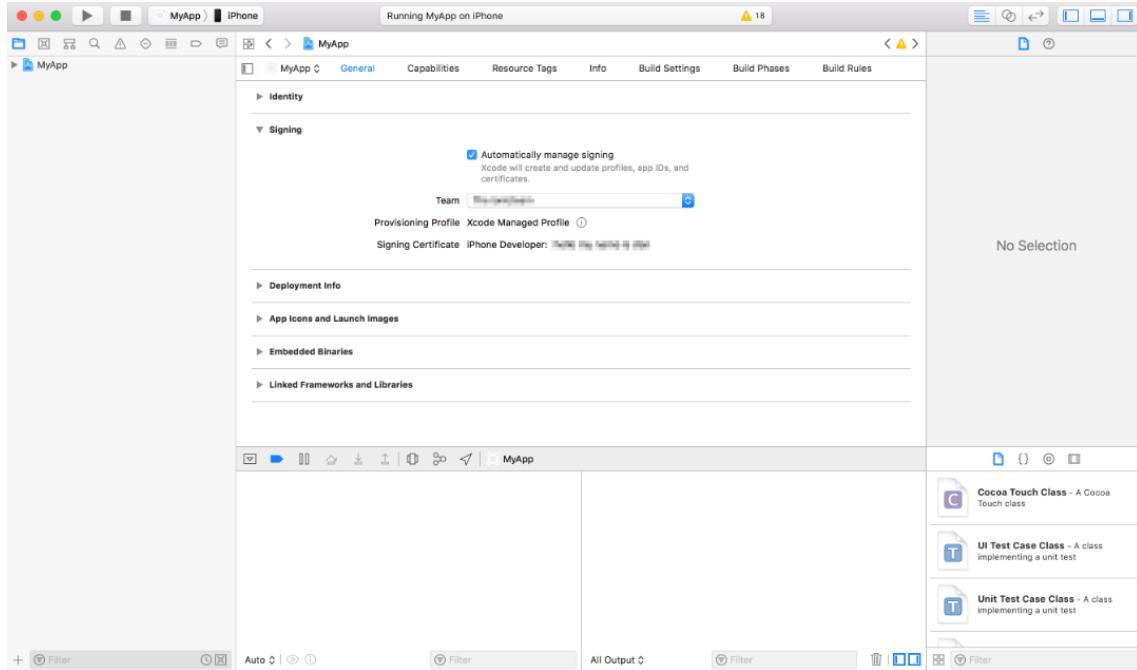


Figura 1.21: Configurando Firma

### Compilar y desplegar

En XCODE, seleccionaremos simulador destino o dispositivo destino conectado por USB a nuestro Mac, luego clicaremos el botón **Play**.

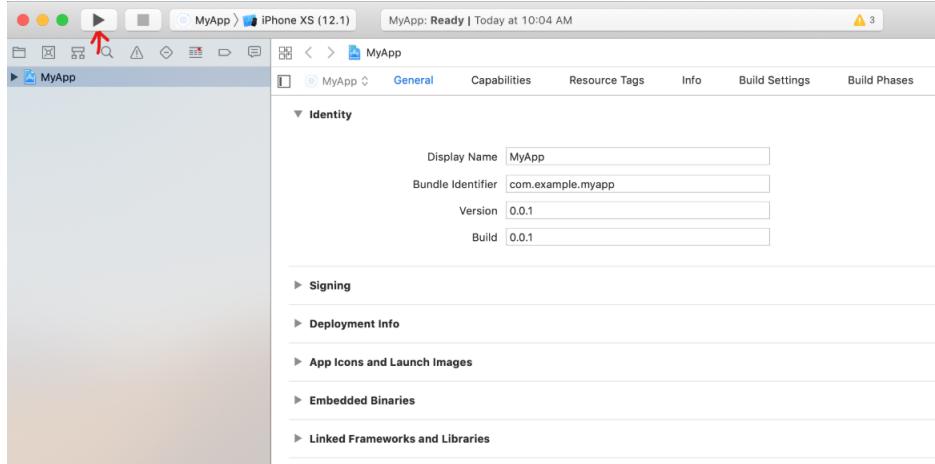


Figura 1.22: Desplegado iOS

Para crear un simulador desde Xcode iremos a **Window Devices and Simulators**. Aquí seleccionaremos el dispositivo (Iphone o Ipad) que deseemos.

## 1.6– Estructura del proyecto

Los proyectos realizados con tecnologías web modernas como Angular, prácticamente nunca se empiezan a desarrollar desde cero. En mi caso, dado que no tenía experiencia en este tema, opté por usar lo que llaman un *starter* o plantilla inicial con una pequeña (muy pequeña dado que las buenas plantillas son de pago) biblioteca de componentes, útiles para ser reusados o para tomarlos como referencia. Este lo encontré en el *market* de Ionic Framework. En la bibliografía dejo un enlace a dicha plantilla [23].

Para instalar la plantilla, tan solo tendremos que descargarla, abrirla desde VS CODE y usar el comando `npm install` estando en el raíz de la plantilla para instalar las librerías y dependencias que esta trae incluida en el fichero `config.xml`.

En caso de que no hubiesemos querido usar una plantilla, podríamos haber empezado un proyecto desde cero con `ionic start` el cual usará la última versión de Ionic instalada. Al usar este comando nos aparecerá un asistente de creación de proyectos por consola que nos preguntará el tipo de proyecto base que deseamos crear (en blanco, con barra lateral inicial, etc...), nombre de la aplicación y otras muchas cosas más.

Una vez explicado todo esto vayamos a lo que realmente importa, la estructura de nuestro proyecto Ionic 4 en el explorador de VS CODE:

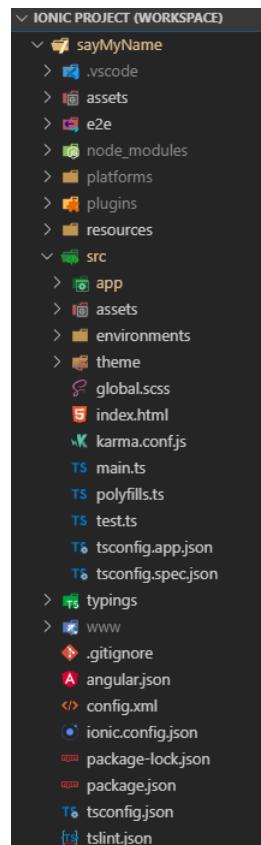


Figura 1.23: Estructura del proyecto

Este proyecto esta plagado de archivos y directorios, pero dado que muchos son transparentes para nosotros, tan solo hablaré de los más importantes. Es decir, los que hemos modificado/utilizado a grandes rasgos. En el capítulo 2, conforme vayamos entrando en materia, iremos explicando más a fondo las vicisitudes que estos presentan, sobre todo con respecto al directorio **app**.

- **node\_modules:** Como ya mencionamos anteriormente, en este directorio podremos encontrar los paquetes NPM propios de Angular, Ionic y los instalados por nosotros mismos. Estos, en resumidas cuentas, nos permitirán importar componentes y funciones para usar desde nuestro código TypeScript y plantillas HTML.
- **platforms:** Este directorio contendrá las plataformas instaladas o añadidas mediante `ionic cordova prepare`, es decir los proyectos nativos para las distintas plataformas, así como los ejecutables de nuestra aplicación una vez esta sea compilada.
- **plugins:** Su nombre lo dice todo. Contiene el grupo de plugins instalados en la aplicación.
- **resources:** Contiene los recursos usados por las distintas plataformas tales como ícono de la aplicación, *Splash Screen*, y como dato, aquí será donde se crearán los recursos redimensionados para los tamaños estándar de cada pantalla mediante el comando `ionic cordova prepare`.
- **src:** En este directorio, se encuentra la parte con mayor peso de todo la aplicación, dado que contiene todo el código fuente de nuestra aplicación, así como recursos y configuraciones usadas en dicho código
  - **app:** Literalmente contiene nuestra aplicación, su código fuente, componentes, páginas, servicios, tuberías, etc...
  - **assets:** Contiene los recursos usados desde nuestro código fuente, tales como imágenes, gifs, semilla SQL para el servicio de base de datos, etc...
  - **theme:** Contiene el fichero `variables.scss` con el tema principal de Ionic de nuestra aplicación. Este consiste en una serie de colores con nombres clave predefinidos que serán totalmente configurables. Como punto positivo al equipo de Ionic, he de decir que proporcionan desde la documentación oficial una herramienta de *theming* que nos generará absolutamente todo el código de `variables.scss` automáticamente en función de los colores que seleccionemos mediante un selector de colores o introduciendo su identificador en hexadecimal. Esta herramienta se llama Color Generator:  
<https://ionicframework.com/docs/theming/color-generator>
  - **global.scss:** Contiene una serie de estilos globales SASS que podrán ser utilizados desde toda la aplicación.
  - **index.html:** Inicio de la aplicación. Desde aquí, dentro de su etiqueta `<body>` se consume el componente raíz que cargará la aplicación, junto a sus módulos base y servicios.
- **www:** Aquí se encontrará, una vez hayamos compilado, nuestro proyecto web JavaScript. No nos es de mucha utilidad directamente, pero si indirectamente a la hora de depurar desde el navegador con `chrome://inspect`, dado que usará estos archivos para interpretar el código nativo que está siendo ejecutado en el dispositivo.

Archivos sueltos en el raíz:

- **config.xml**: Aquí se define toda la configuración que **Cordova** usará para las distintas plataformas, así como los nombres de los iconos y *Splash Screen* que usará la aplicación. En la parte inferior de este fichero también podremos observar los plugins instalados en la *app*, esto ayuda al desarrollador a no tener que subir a su repositorio de código la carpeta *plugins*, dado que al estar aquí definidos, al compilar el proyecto desde un PC nuevo, los plugins serán descargados automáticamente. Veremos su contenido más a fondo en el capítulo 2.
- **ionic.config.json**: Contiene información sobre el proyecto y la aplicación Ionic, tales como el nombre de la *app* y el framework usado para desarrollarla (En nuestro caso Angular).

## 1.7– Estructura de la documentación

Para finalizar este primer capítulo, detallaremos la estructura que seguirá la documentación a partir del siguiente capítulo, exceptuando los capítulos 6 y 7 que corresponden al Manual de Usuario y a las Conclusiones respectivamente.

En cada capítulo comentaremos en profundidad los contenidos de cada grupo de funcionalidades asociadas a la página principal de nuestra aplicación y a cada uno de los botones que esta contiene mediante apartados.

Dado que el *framework* con el que vamos a desarrollar la aplicación utiliza la arquitectura Modelo-Vista-Controlador, utilizaremos una estructura similar para cada uno de los apartados, estando compuesto cada uno de ellos por cuatro secciones:

1. **Mockup**: Es la idea inicial. En resumidas cuentas, detallaremos como debería verse la página una vez realizada. Para ello, mostraremos un boceto realizado con un *software* para crear diagramas, de esta forma quedará mucho más claro el resultado que esperamos conseguir.
2. **Vista**: En este apartado, profundizaremos nuestro conocimiento sobre la *template* o plantilla usada por la página en cuestión. Aquí, le daremos forma, color y aspecto, con el objetivo de aproximarnos lo máximo posible a nuestra idea inicial.
3. **Código**: Explicaremos los métodos y procedimientos más complejos que componen la lógica de la pagina.
4. **Resultado final**: Comprobaremos si el resultado de la página creada se ajusta al resultado que esperábamos cuando realizamos el boceto inicial. Para ello, mostraremos una imagen de la página en cuestión, y comentaremos cualquier posible cambio realizado y sus motivos

## 1.8– Hoja de ruta para el código

Como extra final, quería dejar constancia en esta memoria de una hoja de ruta para seguir el código en función de una serie de requisitos/*tickets* que me auto-impuse a la hora de realizarlo, para que de esta forma cualquier persona que quiera buscar una funcionalidad en el código pueda ver todos los puntos en los que se han realizado desarrollos relacionados con dicha funcionalidad:

- **DS001:** Instalación e implementación de proyecto Ionic 4 y adaptación inicial desde plantilla elegida.
- **DS002:** Importación SQLite y configuración de servicio.
- **DS003:** Desarrollo UNE PALABRAS.
  - **DS003.1:** Implementación de Timer asíncrono para devolver el estado por defecto a las duplas incorrectas pasado un tiempo determinado.
  - **DS003.2:** Implementar Timer para mostrar ventana que defina el final del juego.
  - **DS003.3:** Tubería para reordenar aleatoriamente un array.
    - \* **DS003.3.1:** Crear sentencia sql con límite de valores.
- **DS004:** Implementación de *splash screen* más animación.
- **DS005:** Bloquear en vertical la orientación de la pantalla.
- **DS006:** Implementación de `ion-popover` para mostrar el final del juego basado en el componente `notifications` de la plantilla.
- **DS007:** Preparar base de datos para las múltiples tablas creadas mediante CREA TU TABLA.
- **DS008:** Desarrollo página ELIGE-TABLA.
- **DS009:** Desarrollo gestión de formularios CREA TU TABLA
  - **DS009.1:** Implementación de formulario para modificación de tablas.
  - **DS009.2:** Arreglar *memory leak* provocado por suscripción.
  - **DS009.3:** Alert mostrado para confirmar borrado de tabla y elemento de tabla.
- **DS010:** *Fix view not push up when keyboard shown* y pantalla completa interrumpida.
- **DS011:** Página RESULTADOS con sus respectivas modificaciones en base de datos.
- **DS012:** Eliminar últimos elementos de la plantilla que no van a usarse en el proyecto final y preparar página ABOUT.
- **DS013:** Desarrollo DI MI NOMBRE.

# CAPÍTULO 2

---

## Página principal

---

### 2.1– Mock Up

Como idea inicial, realicé un boceto de todo lo que quería que mi aplicación tuviese, como por ejemplo la sección de juegos y la posibilidad de que el usuario tenga su propio papel protagonista en esta aplicación. Pero claro, siendo una app de aprendizaje de idiomas, también vi que requería de un apartado donde pudiese ver su evaluación y el progreso que haya tenido.

El nombre que da vida a este soporte virtual se debe a que, como ya estamos mencionando, es un juego para aprender (inicialmente) un idioma tan internacional y necesario como es el inglés. Para poder llevar a cabo dicho aprendizaje, tenemos que poner una palabra en español para que posteriormente **digamos su nombre** en inglés, aumentando así nuestro vocabulario.

Al finalizar este boceto, decidí plasmar mi idea en una plataforma online para realizar *Mock Ups*<sup>1</sup>, en concreto yo he decidido usar <https://creately.com> para visualizar cómo podría quedar en una pantalla, dándome como resultado el que se expone a continuación.

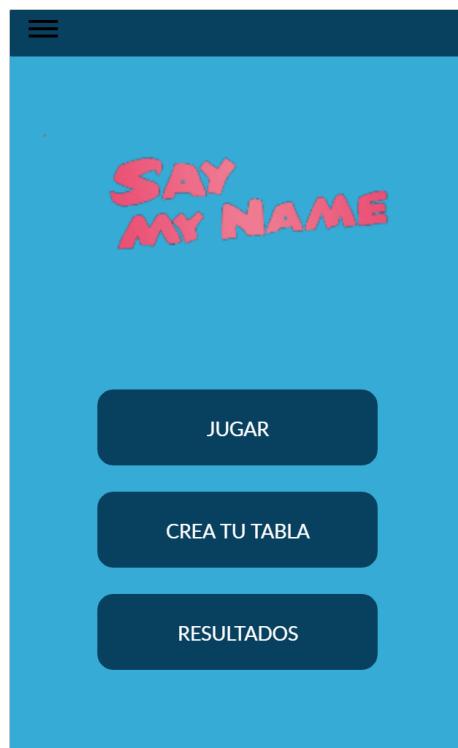


Figura 2.1: *Mock up* Menú principal

---

<sup>1</sup>Los *Mock Ups* son fotomontajes que permiten a los diseñadores gráficos y web mostrar al cliente cómo quedaran sus diseños.

## 2.2– Vista

En el capítulo anterior, explicamos cómo dejar preparado todo el entorno para poder empezar a trabajar, compilar y probar nuestra aplicación. Ahora, ya estáis listos para comenzar a aprender como desarrollar nuestra propia *app* en Ionic.

Como comentamos anteriormente, desarrollar una aplicación en Angular desde cero puede ser algo complejo, por ese motivo decidí usar una plantilla básica como línea de salida a este desarrollo. El problema, es que una plantilla no siempre viene al gusto de nuestras necesidades, por este motivo, lo primero que hice fue comentar todas esas funcionalidades y componentes extra que no nos son de ninguna utilidad para lograr el aspecto y comportamientos deseados. Siempre con la idea en mente de que quizás, en un futuro, pudiesen ser de utilidad.

Hablando de aspecto, aprovechando que este es nuestro primer contacto con la vista, *template* o plantilla de una página en esta memoria, aprovecharé para definir una serie de conceptos teóricos antes de ponernos técnicos y mostrar algo de código HTML. Estos conceptos, nos ayudarán a saber de qué herramientas disponemos para lograr que nuestra *template* en desarrollo cumpla nuestras expectativas visuales.

IONIC FRAMEWORK nos provee de distintos tipos de *layout*<sup>2</sup>; desde diseños de una sola página, hasta paneles divididos o emergentes. Estos *layout*, pueden ser usados para estructurar nuestra aplicación de la forma que deseemos. Usar el *layout* correcto desde un inicio nos ayudará mucho más de lo que cabría esperar, dado que nos evitará tener que realizar muchas correcciones mediante SASS<sup>3</sup>, o peor aún, tener que reestructurar todo desde cero.

Me gustaría dejar claro que, de ahora en adelante, para las secciones Vista de todos los capítulos y apartados de esta documentación, nuestra mejor aliada será la documentación oficial de Ionic [1]. Obviamente, durante el desarrollo de este proyecto, tuve que consultar muchas artículos y documentaciones externas a Ionic Docs. Pero dada la gran cantidad de componentes propios de Ionic usados en Say My Name, este sitio web ha sido el recurso más valioso, y por tanto, me ayudaré de él para que comprendáis con mayor claridad y veracidad el porqué de dichos componentes usados en las plantillas que iremos revisando juntos.

Volviendo al *layout*, el más usado en nuestra aplicación será el **Header Layout** o diseño de página con cabecera. Este, es el diseño más sencillo provisto por Ionic, consiste en un encabezado y un contenido. La mayoría de las páginas de una aplicación generalmente tienen ambas cosas, por el mero hecho de dar una arquitectura consistente a las páginas, pero no se necesita un encabezado para usar el contenido.

---

<sup>2</sup>El *layout* es en esencia el diseño de la página, es la parte del diseño gráfico que trata la disposición de los elementos visuales en esta.

<sup>3</sup>Sass es un lenguaje de hojas de estilo que compila a CSS y es usado por Ionic. SASS es como CSS, pero con características adicionales como variables, mixins y loops.

```

1 <ion-app>
2   <ion-header>
3     <ion-toolbar>
4       <ion-title>Header</ion-title>
5     </ion-toolbar>
6   </ion-header>
7
8   <ion-content>
9     <h1>Ejemplo header layout</h1>
10    </ion-content>
11 </ion-app>

```

Código 2.1: Fichero ejemHeader.html

Como podemos observar en el ejemplo, todos los componentes de Ionic van precedidos del prefijo **ion** en sus etiquetas HTML. Así podremos identificar qué etiquetas son componentes pertenecientes a la librería de componentes de Ionic y cuáles no. A su vez, en el ejemplo, podemos ver otra etiqueta **content**, que identifica el contenido de la página. Dentro del **<ion-content>** irán todos los componentes y elementos de la página que estén fuera del **header**.

La página principal de nuestra aplicación, en la que se versa este capítulo, es una excepción del uso de este *layout*. En su lugar, usaremos el ***Split Pane Layout*** o diseño con panel dividido (menú lateral). Este, tiene una estructura más compleja porque puede combinar muchos otros diseños. Permite la visualización de múltiples vistas cuando el ancho de la pantalla está por encima de un punto de interrupción especificado. Si el tamaño de la pantalla del dispositivo fuese inferior a un tamaño determinado, la vista de panel dividido se ocultaría. En caso de que estuviésemos visualizando esta *app* en el navegador de un ordenador a pantalla completa, siempre veríamos el panel dividido, y al redimensionar el navegador horizontalmente para que la ventana fuese más estrecha, la vista de panel dividido desaparecería. Al solo estar dirigida esta aplicación para dispositivos móviles, debido al ancho de la pantalla, esto nunca ocurriría, dado que el ***Split Pane*** siempre estaría oculto, pese a ello, creí conveniente contaros su funcionalidad.

Os estaréis preguntando entonces, el porqué decidí usar este tipo de *layout* en lugar de otro que simplemente tuviese el típico menú lateral tan usado en las apps de hoy día. La respuesta es simple, la plantilla que usé incorporaba ya este tipo de *layout* y decidí mantenerlo para ahorrar tiempo, ya que era transparente para mi objetivo.

Está claro que toda plantilla tiene sus inconvenientes, y el de esta, lo acabo de encontrar ahora, mientras trato de explicarlos en esta introducción teórica general (y digo general porque todo esto tan solo os lo explicaré aquí), ya que ahora viene la que quizás sea la parte más compleja y abstracta de toda la documentación.

Soy consciente, de que este apartado está centrado en la vista de la aplicación principal, y que quizás lo que os voy a explicar ahora fuese más propio del apartado de código, pero creo conveniente describiros este proceso ahora, antes de empezar a exponer el contenido que he desarrollado, a expensas de que esto signifique que otros apartados queden más pobres en cuanto a contenido. Está claro, que para repasar todas y cada una de las líneas de código que he realizado sería necesario adjuntar todos los ficheros del proyecto Say My Name en esta memoria, y considero que eso sería excederse en el propósito para el que fue pensada. Dicho esto, espero que haya quedado claro que los contenidos expuestos a partir de ahora, tan solo serán los fragmentos y partes de código más interesantes (bajo

mi criterio) de cada una de las páginas. Imagino que estaréis de acuerdo conmigo en esta decisión, ya que al ser el proyecto de código Ionic otro de los entregables de este Trabajo de Fin de Grado, sería absurdo y prácticamente de relleno el replicarlo aquí.

Comencemos con la parte más abstracta de este desarrollo, la explicación de cómo el *framework* ANGULAR hace para indicar cuál es el inicio/página principal de nuestra aplicación y de cómo interconecta todas las páginas/componentes entre sí permitiendo el uso compartido de sus funcionalidades.

Para ello, empezaré por mostraros la estructura del directorio **app** que antes disimuladamente os oculte en la estructura del proyecto:

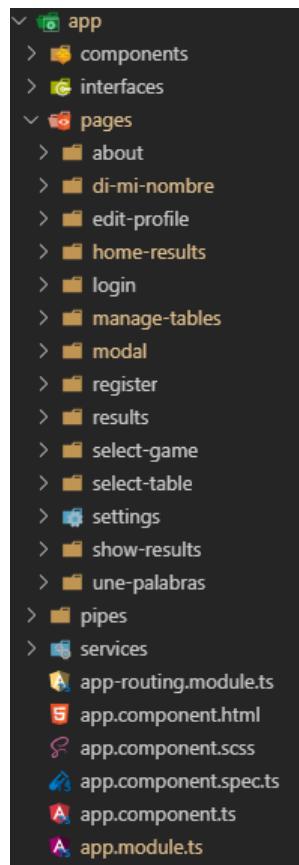


Figura 2.2: Estructura directorio src

- **components:** Contendrá los componentes de la aplicación, y cuando hablamos de componentes nos referimos a páginas de diverso tamaño encajonadas en un bloque reutilizable por las páginas “normales”.
- **interfaces:** Contiene las interfaces definidas de forma global para la aplicación, en nuestro caso solo tenemos una llamada **pages**.
- **pages:** Aquí, se almacenan las páginas de la aplicación, cuya estructura veremos de forma más detallada en el apartado código.
- **pipes:** Contiene las tuberías usadas en el desarrollo de la *app*. Estas son mecanismos

con una entrada, que es procesada de una cierta manera dando lugar a una salida con dicha transformación.

- **services:** Los servicios de la aplicación. En nuestro caso solo tenemos uno, el servicio de base de datos. Los servicios son clases TypeScript. Su propósito es contener lógica de negocio, clases para acceso a datos o utilidades de infraestructura. Estas clases son perfectamente instanciables desde cualquier otro fichero que las importe. Pero Angular nos sugiere y facilita que usemos su sistema de inyección de dependencias.
- **app-routing.module.ts:** Es un fichero global, contiene toda la definición de la navegación (Angular Router). Permite que ANGULAR sepa identificar cada página a la hora de navegar entre ellas. La clase que exporta será importada por el módulo raíz, cosa que explicaremos en breve.
- **app.component.html:** Es la vista del módulo raíz de la aplicación. Será cargada como atributo del decorador **@component<sup>4</sup>** en el componente **app.component.ts**.
- **app.component.scss:** Estilos usados por el módulo raíz.
- **app.component.spec.ts:** Se utiliza para *testing* y pruebas unitarias, por lo que queda fuera del alcance de esta memoria.
- **app.component.ts:** Aquí se encuentra la lógica del módulo raíz, dentro de ella se destaca la inicialización de la aplicación, es decir, comprueba que la plataforma de despliegue este lista, despliega la *statusBar*<sup>5</sup> y en nuestro caso oculta el *Splash Screen*. Por último, añadir que este componente es cargado mediante bootstrapping<sup>6</sup> por el modulo raíz **app.module.ts**.
- **app.module.ts:** Módulo raíz de la *app*. Declara componentes, importa módulos y registra proveedores que estarán disponibles desde todos los componentes páginas de la aplicación.

Se que es mucha información de golpe, pero es necesario para comprender de verdad cómo funciona la aplicación, cómo se inician y cómo se interrelacionan sus componentes. Una vez explicado todo esto, el resto de la memoria resultará mucho más fácil de comprender, solo teniendo que detenernos para explicar el uso y funcionamiento de algunos componentes.

Para allanar aún más el camino, os explicaré a grandes rasgos como funciona la aplicación desde que pulsamos sobre su ícono para abrirla desde nuestro dispositivo móvil hasta que se haya cargado completamente la página principal. Será entonces cuando expliquemos el código HTML de su vista, mucho más sencillo, vistoso y entretenido que toda esta parafernalia.

---

<sup>4</sup>En este caso nos referimos a un decorador de componente, nos permite marcar una clase como un componente angular y proporcionar metadatos adicionales que determinen cómo se debe procesar, instanciar y utilizar el componente en tiempo de ejecución.

<sup>5</sup>Barra de estado que podemos encontrar normalmente en la parte superior de nuestro dispositivo móvil. Suele contener el estado de la batería, notificaciones, etc...

<sup>6</sup>En pocas palabras, en el arranque de la aplicación.

Sin más dilación, comencemos... Pulsaremos el icono de Say My Name e inmediatamente la aplicación comenzará a abrirse por primera vez. Será ahora, cuando el módulo raíz despierte y comience a declarar componentes, importar módulos y registrar proveedores para asegurarse de que la aplicación, una vez lista, disponga de ellos cuando los necesite y por último, mediante *bootstrap*, arrancará/lanzará el componente raíz **app.component.ts** para cuando el **index.html** de la *app* lo solicite para su consumo.

```

1  @NgModule({
2    declarations: [AppComponent, NotificationsComponent], // SEJMM; Declaramos
3      componentes
4    imports: [
5      BrowserModule,
6      BrowserAnimationsModule,
7      IonicModule.forRoot(),
8      AppRoutingModule,
9      HttpClientModule,
10     FormCreationPageModule,
11     FormModPageModule, // DS009.1: Implementación de formulario para modificación
12       de tablas
13     SearchFilterPageModule
14   ],
15   entryComponents: [NotificationsComponent], // SEJMM; Necesario para poder
16     utilizarlo como un router component y cargarlo imperativamente
17   providers: [
18     StatusBar,
19     SplashScreen,
20     { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
21     /* SEJMMINI DS0002 */
22     SQLite,
23     SQLitePorter,
24     /* SEJMMFIN DS0002 */
25     Keyboard /*SEJMM DS010 */
26   ],
27   bootstrap: [AppComponent]
28 })
29
30 export class AppModule {}

```

Código 2.2: Fichero **app.module.ts**

Una vez **index.html** consuma el selector `<app-root> </app-root>`, este ejecutará su constructor, inicializando la app y cargando el **Split Pane Layout** definido en su template **app.component.html**:

```
1 @Component({
2   selector: 'app-root',
3   templateUrl: 'app.component.html',
4   styleUrls: ['./app.component.scss']
5 })
6 export class AppComponent {
7   public appPages: Array<Pages>;
8   showSplash = true; // SEJMM DS004 SplashScreen + animation
9
10  constructor(
11    private platform: Platform,
12    private splashScreen: SplashScreen,
13    private statusBar: StatusBar,
14    public navCtrl: NavController
15  ) {
16    this.appPages = [
17      {
18        title: 'Home',
19        url: '/home-results',
20        direct: 'root',
21        icon: 'home'
22      },
23      {
24        title: 'Sobre Say My Name',
25        url: '/about',
26        direct: 'forward',
27        icon: 'information-circle-outline'
28      }
29    ];
30    this.initializeApp();
31  }
}
```

Código 2.3: Fichero **app.component.ts**

```
1 <body>
2   <app-root></app-root>
3   <noscript>Please enable JavaScript to continue using this application.</
4     noscript>
</body>
```

Código 2.4: Fichero **index.html**

Como añadido, os comento por encima que esa variable `showSplash` es usada para determinar cuando finalizará la animación [25] que sigue al *Splash Screen* en la carga de la aplicación (`app.component.html`). Todo esto, es controlado en el interior de la función `initializeApp()`:

```

1  initializeApp() {
2    this.platform.ready().then(() => {
3      this.statusBar.styleDefault();
4      this.splashScreen.hide();
5
6      // SEJMM INI DS004; SplashScreen + Animation
7      // Timer para controlar la visibilidad de la animación
8      timer(3000).subscribe(() => this.showSplash = false);
9    }).catch(() => {});
10 }

```

Código 2.5: Función `initializeApp`

Ya sabemos como aparecen y desaparecen el *Splash Screen* y la animación en la carga de la aplicación, imaginamos la carga de la barra lateral de la página principal (aunque permanezca oculta) pero... ¿Cómo pasamos de este punto a la carga de la página principal con los tres botones que diseñamos en nuestro *mockup*? La respuesta es simple, y su nombre es **app-routing.module.ts**. Este fichero/clase, es importado por **app.module.ts** como podemos observar en el código 2.2. Su cometido es permitir que ANGULAR sepa identificar cada página a la hora de navegar entre ellas y redirigir la vista de la aplicación a la página principal mediante el path: ''.

```

1 const routes: Routes = [
2   // SEJMM DS001 Esta primera linea redirige a la pagina inicial de la app
3   { path: '', redirectTo: 'home-results', pathMatch: 'full' },
4   { path: 'login', loadChildren: './pages/login/login.module#LoginPageModule' },
5   { path: 'register', loadChildren: './pages/register/register.module#'
6     RegisterPageModule' },
7   { path: 'home-results', loadChildren: './pages/home-results/home-results.module'
8     #HomeResultsPageModule' },
9   { path: 'about', loadChildren: './pages/about/about.module#AboutPageModule' },
10  { path: 'settings', loadChildren: './pages/settings/settings.module#'
11    SettingsPageModule' },
12  { path: 'edit-profile', loadChildren: './pages/edit-profile/edit-profile.module'
13    #EditProfilePageModule' },
14  { path: 'une-palabras/:tableName', loadChildren: './pages/une-palabras/une-'
15    palabras.module#UnePalabrasPageModule' }, // SEJMM DS007; Preparación
16    multitable
17  { path: 'manage-tables', loadChildren: './pages/manage-tables/manage-tables.'
18    module#ManageTablesPageModule' },
19  { path: 'select-table', loadChildren: './pages/select-table/select-table.module'
20    #SelectTablePageModule' },

```

Código 2.6: Código `app-routing.module.ts`

En este último código tenemos la respuesta a nuestra pregunta. Como podremos observar, el *path* por defecto nos redirige a la página **home-results**, y esta no es sino otra que nuestra tan esperada página principal.

Poco más acerca de la Vista podemos aportar aquí, es una página sencilla, con una **<ion-toolbar>** en la cabecera, esta posee una fila de botones **<ion-buttons>** que contiene un solo botón **<ion-menu-button>** que se encargará de desplegar la vista del **Split Pane Layout** al pulsarlo. A esto me refería al inicio de esta sección cuándo decía que esta era la parte más compleja y abstracta de toda la memoria, ya que para explicarlos como se unen el panel lateral y la página principal, era necesario también explicarlos como Angular entiende y une cabos para llegar a esta situación. Espero que haya quedado lo suficientemente claro, si es así ¡Reto superado!

Volviendo a la Vista, lo único que me queda por añadir acerca de la página principal de nuestra aplicación, es que contiene una imagen (**<ion-img>**) con el logo de nuestra *app* y tres botones **<ion-button>** dentro del contenido (**<ion-content>**) con las siguientes características a destacar:

- **fill="outline"**: Botón con fondo transparente y borde visible. Existen otros valores posibles que veremos en otras páginas.
- **shape="round"**: Indica que deseamos que las esquinas del botón se tengan una forma redondeada.
- **expand="full"**: Para un botón que rellene el ancho de la pantalla sin bordes izquierdo y derecho.
- **color="secondary"**: Si echamos la vista atrás, en la estructura de nuestro proyecto teníamos un directorio llamado **theme** con un fichero **variables.scss** dentro que indicaba una serie de colores con nombres clave predefinidos, pues bien, el valor *secondary* es uno de ellos.

Para finalizar, la ultima propiedad del botón es **(click)="goToSelectTable()"** en el caso del botón superior (Jugar), esta es una especie de evento *onclick*, que llamará a la función pasada como valor al pulsar sobre él. Dicha función nos redirigirá a la página **Select Table**, el cómo y el porqué lo veremos en el siguiente capítulo, destinado a dicho botón **Jugar** en el apartado 3.2.3.

## 2.3– Código

Para no extenderme más en éste capítulo, tan solo mostraré el código de la función que se encarga de mostrar el **Split Pane** al pulsar el botón de la barra de herramientas. Para ello importará y declarará en su constructor la clase **MenuController**, y habilitará dicho menú mediante la función **enable**:

```
1 ionViewWillEnter() {  
2     this.menuCtrl.enable(true);  
3 }
```

Código 2.7: Función **ionViewWillEnter**

## 2.4– Resultado final

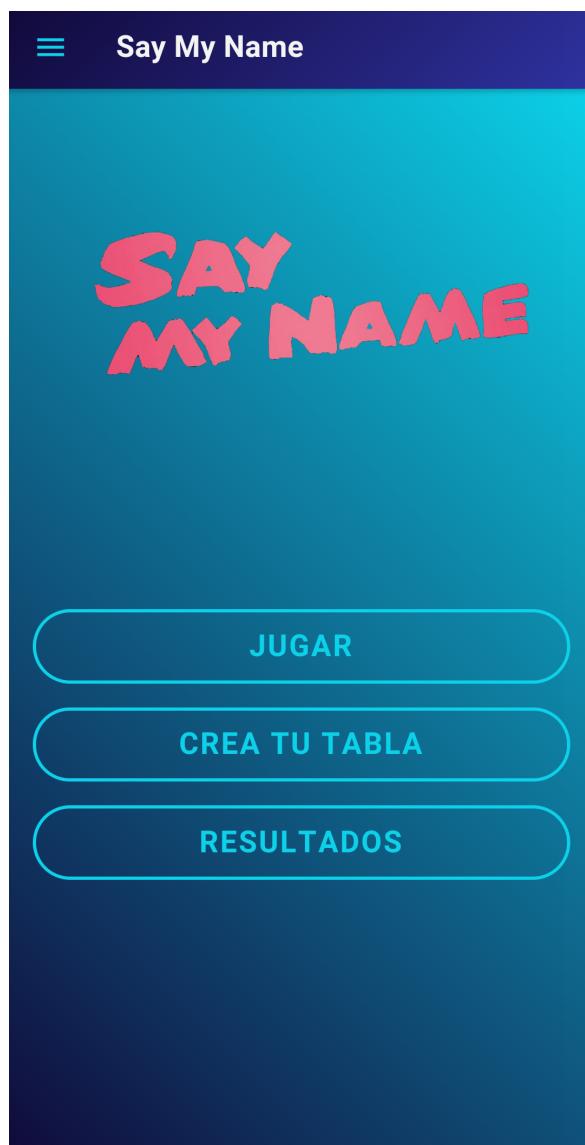


Figura 2.3: Menú principal

# CAPÍTULO 3

---

## Jugar

---

### 3.1– Elige temática

#### 3.1.1. *Mock Up*

Para llevar a cabo los juegos, decidí que era más cómodo escoger inicialmente la temática, y luego el juego con el que resulte más fácil comenzar el aprendizaje del que se hablará en el apartado siguiente 3.2.1. Ambas páginas, estarán acompañadas de una flecha que represente el poder volver a la página anterior en caso de error o 3 arrepentimiento.

La idea que expongo desde un principio es que quiero que sea una aplicación **apta para todos los públicos**, cuya dinámica sea **fácil de usar**, por lo que en este caso la idea es poner el título de la página en la que nos encontramos, y a continuación que se visualicen todos las temáticas que tengamos registradas. Actualmente solo ponemos de muestra las dos que tenemos a modo de ejemplo, **Animales** y **Colores**.



Figura 3.1: *Mockup* Elige temática

### 3.1.2. Vista

La vista de esta página es bastante sencilla, por lo que no nos extenderemos mucho comentándola. Se compone de una lista <ion-list> de botones (<ion-button>). La lista, posee una propiedad `inset` con valor a `true`, esto provocará que la lista tenga margen alrededor y esquinas redondeadas provocando así el efecto de que los botones son más estrechos sin necesidad de hacer modificaciones en la hoja de estilo. Aunque por defecto dicha lista tendría un fondo de color blanco por defecto que no se ajusta a nuestro diseño inicial, por lo que usaremos sobre ella una clase llamada `bg-list` definida en el `select-table.page.scss`. Este documento también contendrá el color del fondo de la página/contenido:

```

1 ion-content {
2   --background: linear-gradient(-135deg, var(--ion-color-secondary), var(--ion-
3     color-dark)); // SEJMM Color fondo pantalla UnePalabras (Todas las fases)
4 }
5 ion-button {
6   height: 45px;
7   font-size: 20px;
8 }
9 .bg-list{
10   background: transparent; // SEJMM Color fondo lista en UnePalabras (Todas las
11   fases)
12   padding: 0%;
13 }
```

Código 3.1: Fichero `select-table.page.scss`

En el siguiente fragmento de código correspondiente a la plantilla de la página, podemos observar cómo recorremos la tabla con una directiva llamada `*ngFor`, esta nos permitirá recorrer desde la misma `template` una lista definida en el controlador/`page.ts` como si de un **for extendido** se tratase, recorriendo uno por uno, todos los elementos del array y mostrándolos como contenido textual de cada botón con el formato `{{ x }}` siendo `x` el elemento de la iteración correspondiente del `*ngFor`:

```

1 <ion-content padding>
2   <ion-list class="bg-list" inset=true>
3     <ion-button margin shape="round" expand="full" *ngFor="let table of
4       tablesArrayName" (click)="goToSelectGame(table)">
5       {{ table }}
6     </ion-button>
7   </ion-list>
</ion-content>
```

Código 3.2: Fichero `select-table.page.html`

### 3.1.3. Código

A priori, esta página podría parecer sencilla y de poca complejidad, de hecho así es. La parte complicada recae sobre el uso que se hace en ella desde el controlador **select-table.page.ts** gracias a los datos aportados por el Servicio de Base de Datos que hemos desarrollado tomando como referencia el enlace referido en la bibliografía [26] y aumentando posteriormente sus funcionalidades y complejidad para adecuarlo a los requisitos que Say My Name requiere para cumplir el propósito esperado.

Lo primero que tenemos que hacer al crear un servicio de base de datos es preguntarnos que necesitamos que nos ofrezca. En mi caso, decidí que la aplicación ofrecería a los usuarios tablas creadas por defecto, y que además ellos podrían crear y gestionar sus propias tablas. Pues bien, **Cordova** soporta dos plugins que nos ofrecen dichas funcionalidades. Así que lo siguiente que tendremos que hacer será instalarlos.

Para ello, haremos uso de npm y del Ionic CLI y de los siguientes comandos para instalar los paquetes **sqlite** y **sqlite-porter**:

- `npm install @ionic-native/sqlite @ionic-native/sqlite-porter`
- `ionic cordova plugin add cordova-sqlite-storage`
- `ionic cordova plugin add uk.co.workingedge.cordova.plugin.sqliteporter`

**sqlite**, por su parte, nos brinda las clases **SQLite** que nos aporta la función `create` con la que crearemos la base de datos y **SQLiteObject** que nos ofrece la función `executeSql` con la que ejecutaremos todas las *queries* o peticiones a la base de datos para leer, modificar, borrar y escribir en ella. Esto último es posible debido a que la base de datos que devuelve `create` es de tipo **SQLiteObject**.

**sqlite-porter**, nos da la posibilidad de utilizar la función **importSqlToDb**, que nos permitirá importar las tablas creadas mediante un fichero con extensión `.sql` a modo de semilla (en nuestro proyecto, `seed.sql`), dicha función tiene como parámetros de entrada la base de datos en la que importaremos las tablas y el fichero `.sql`.

Para verlo con mayor claridad, os mostraré un fragmento de código con el constructor del servicio de base de datos, y la correspondiente función de carga de la semilla:

```

1  this.plt.ready().then(() => {
2    this.sqlite.create({
3      name: 'sayMyName.db',
4      location: 'default'
5    })
6    .then((db: SQLiteObject) => {
7      this.database = db;
8      this.seedDatabase();
9    })
10   .catch((e) => {
11     console.log(e);
12   });
13 });
14 }
```

Código 3.3: Fichero `database.service.ts`, Constructor

```

1  this.http.get('assets/seed.sql', { responseType: 'text' })
2    .subscribe(sql => {
3      this.sqlitePorter.importSqlToDb(this.database, sql)
4        .then(_ => {
5          this.dbReady.next(true);
6        })
7        .catch(e => console.error(e));
8    });
9  }
10 /**

```

Código 3.4: Fichero database.service.ts, seedDatabase

Para crear este servicio, me he valido de la ayuda del comando de Ionic CLI `ionic g service services/database`, este comando nos creará el directorio **services**, en él encontraremos dos ficheros, uno llamado **database.service.spec.ts** para pruebas unitarias y otro **database.service.ts** que contendrá toda la lógica de la base de datos.

A continuación, adjuntaré otro fragmento de código de nuestro servicio que nos servirá para entender otros dos conceptos importantísimos de Angular, el decorador `@Injectable` y los **observables**:

```

1 export interface ResElem { // SEJMM DS011
2   id: number;
3   aciertos: number;
4   errores: number;
5 }
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class DatabaseService {
11   private database: SQLiteObject;
12   private dbReady: BehaviorSubject<boolean> = new BehaviorSubject(false);
13   public tableReady: BehaviorSubject<boolean> = new BehaviorSubject(false);
14   public tableForResultsReady: BehaviorSubject<boolean> = new BehaviorSubject(
15     false);

```

Código 3.5: Fichero database.service.ts, `@Injectable` y Observables

Lo primero que vemos en la captura, es la declaración de una `interface`. Esto es un objeto con atributos, que podremos usar como tipo en las funciones de este servicio y en todos los componentes que importen esta `interface`. En cuanto a lo siguiente, en Angular, declarar y decorar la clase no es suficiente para poder reclamarla. Es necesario registrarla como un proveedor en algún módulo (dentro del decorador `@NgModule`) o en el raíz si es de carácter general para todos los componentes. Desde Angular 6 los servicios inyectables (identificados mediante el decorador `@Injectable`) se auto-proveen en el módulo raíz mediante la configuración `providedIn: 'root'` de su decorador. Esto es muy útil y cómodo en una gran cantidad de casos. El módulo raíz es visible para toda la aplicación, por lo que cualquier componente puede reclamar un servicio suyo sin problemas.

Al consumo de los servicios inyectables se le conoce como dependencia. Cada componente o servicio puede declarar en su constructor sus dependencias hacia servicios inyecta-

bles. Siempre especificando el tipo esperado. Este, es el procedimiento seguido por todos los componentes de Say My Name que necesitan de nuestro Servicio de Base de Datos. Por ejemplo, en la página que nos ocupa (**Elige Tabla**), el procedimiento sería el mostrado en la captura de código que os expondré a continuación:

```
1 import { Component, OnInit, OnDestroy } from '@angular/core';
2 import { DatabaseService } from '../../../../../services/database.service'; // Importamos clases DB
3 import { NavController} from '@ionic/angular'; // SEJMM DS007: Preparación multitable
4 import { Subject } from 'rxjs';
5 import { takeUntil } from 'rxjs/operators'; // SEJMM DS009.2; Fix memory leak provocado por suscripción y Fix de repetición de tablas provocado por suscripción
6 @Component({
7   selector: 'app-select-table',
8   templateUrl: './select-table.page.html',
9   styleUrls: ['./select-table.page.scss'],
10 })
11 export class SelectTablePage implements OnInit, OnDestroy {
12   private unsubscribe$: Subject<void> = new Subject();
13   tablesArrayName: string[] = [];
14   constructor(private db: DatabaseService, // DS002: Base de datos SQLite
15     public navCtrl: NavController // SEJMM DS007: Preparación multitable
16   ) {}
17   ngOnInit() {
18     this.db.getDatabaseState().pipe(takeUntil(this.unsubscribe$)).subscribe(rdy => {
19       if (rdy) {
20         this.db.loadTables();
21         this.db.getTables().pipe(takeUntil(this.unsubscribe$)).subscribe(tables => {
22           this.tablesArrayName = tables;
23           /* Ordenamos las tablas en orden alfabético */
24           this.tablesArrayName.sort(function (a, b) {
25             if (a < b) { return -1; }
26             if (a > b) { return 1; }
27             return 0;
28           });
29         });
30       }
31     });
32   }
33   ngOnDestroy() {
34     console.log('Select Table: ngOnDestroy');
35     this.unsubscribe$.next();
36     this.unsubscribe$.complete();
37   }
38   goToSelectGame(tableName: string) {
39     this.navCtrl.navigateForward(['/select-game', tableName]);
40   }
41 }
```

Código 3.6: Fichero `select-table.page.ts`

Como podemos observar, en la parte superior del código importamos el servicio que hemos creado, para después consumirlo en el constructor del componente indicando su tipo **DatabaseService**. Una vez hecho esto, estaremos listos para poder utilizar los datos inyectados desde **database.service.ts**.

Es ahora cuando entra en juego ese segundo concepto de Angular tan importante, los **observables**. Por norma general, las aplicaciones realizadas mediante el *framework* Angular son aplicaciones dinámicas, esto es equivalente a decir que si una página obtiene datos desde un proveedor y estos datos cambian, el uso que hace esta página de dichos datos tendrá en cuenta dichos cambios automáticamente, sin necesidad de recargar la página para obtener estos datos de nuevo. La forma más adecuada de asegurar este funcionamiento es el uso de observables, dada su utilidad y versatilidad.

Ahora, imagino que os estaréis preguntando... ¿Pero **qué son** los observables?, por lo que creo que es el momento idóneo para verlos detalladamente, dado que en esta página Elige Tabla son utilizados, y ya os adelanto que el resto de páginas que consuman el servicio de base de datos (casi todas) también lo harán. De paso, también nos servirá para explicar la última parte del código 3.5.

## Observables

Los observables nos ahorrarán tener que hacer consultas repetitivas de acceso a la fuente de información, aumentando el rendimiento de las aplicaciones. Estos proporcionan soporte para pasar mensajes entre servicios que publican datos y suscriptores (normalmente componentes) que los reciben. El patrón observable no es más que un modo de implementación de la **programación reactiva**<sup>1</sup>, que básicamente pone en funcionamiento diversos actores para producir los efectos deseados, que es reaccionar ante el flujo de los distintos eventos producidos. Mejor dicho, producir dichos eventos y consumirlos de diversos modos [27]. Los componentes principales que conforman este patrón son:

- **Observable**: Es aquello que queremos observar, que será implementado mediante una colección de eventos o valores futuros. Un observable puede ser creado a partir de eventos de usuario derivados del uso de un formulario, una llamada HTTP, un almacén de datos, etc. Mediante el observable nos podemos suscribir a eventos que nos permiten hacer cosas cuando cambia lo que se está observando.
- **Observer**: Es el actor que se dedica a observar. Básicamente se implementa mediante una colección de funciones callback que nos permiten escuchar los eventos o valores emitidos por un observable. Las callbacks permitirán especificar código a ejecutar frente a un dato en el flujo, un error o el final del flujo.
- **Subject**: es el emisor de eventos, que es capaz de crear el flujo de eventos cuando el observable sufre cambios. Esos eventos serán los que se consuman en los observers.

---

<sup>1</sup>La programación reactiva es un paradigma de programación declarativo relacionado con los flujos de datos finitos o infinitos de manera asíncrona y la propagación del cambio

Existen diversas librerías para implementar programación reactiva que hacen uso del patrón observable. Una de ellas es RxJS<sup>2</sup>, que es la que se usa en Angular y por supuesto en nuestro proyecto como se advierte en los *imports* de nuestro servicio de base de datos o incluso en el controlador con el **Subject** y la tubería **takeUntil**.

Todo esto es tan solo teoría y palabrería. En mi opinión, la mejor forma de comprender en qué consiste un concepto tan abstracto como lo es el patrón observable, es mostrar un ejemplo donde se vean expuestas las diferencias entre la programación tradicional que todos conocemos y la irracional programación reactiva:

```

1 let a = 2;
2 let b = 5;
3 let resultado = a + b; // resultado tendrá valor 7
4 // Más tarde en las instrucciones...
5 a = 3; // Asignamos otro valor a la variable a
6 // Aunque se cambie el valor de "a", resultado seguirá siendo 7

```

Código 3.7: Ejemplo de programación tradicional

```

1 //Ahora imaginemos que func es una función que devuelve un observable
2 //Y que su valor inicial fuese 2
3 func().subscribe(x => {
4     let a = x;
5     let b = 5;
6     let resultado = a + b; // resultado tendrá valor 7
7 });
8
9 // Más tarde alguien informa que el valor devuelto por el observable
10 // ha cambiado a 3
11 // En este caso, resultado cambiará a 8

```

Código 3.8: Ejemplo de programación reactiva

Básicamente, sería como decir que cada vez que el **subject** informe al **observer** de un nuevo evento de cambio en la variable observada, todo el código que se encuentre dentro de la suscripción (**subscribe()**) a dicho **observable** será ejecutado de nuevo. Con estos conceptos bien amarrados, ahora podremos comprender mucho mejor cómo se aplica todo esto a Say My Name.

En nuestro proyecto, el **servicio de base de datos** es el encargado de mantener toda la información almacenada y servir de nexo entre todos los componentes que hacen modificaciones sobre ella, manteniéndola actualizada dinámicamente siempre y cuando estos componentes tengan activo su **ciclo de vida** (el cual será detallado en el apartado 3.3.3).

Volviendo a los observables, el **servicio de base de datos** será el encargado de usar el **subject** para generar los eventos cuando se produzcan cambios en las variables observadas. Este servicio también será el encargado de **entregar el observable** a los componentes que necesiten escuchar los eventos emitidos por el subject.

---

<sup>2</sup>Reactive Extensions (Rx) es una librería hecha por Microsoft para implementar la programación reactiva, creando aplicaciones que son capaces de usar el patrón observable para gestionar operaciones asíncronas. Por su parte RxJS es la implementación en Javascript de ReactiveExtensions, una más de las adaptaciones existentes en muchos otros lenguajes de programación.

En nuestro caso, no hemos usado Subject, sino BehaviourSubject[28] que se diferencia de Subject en que requiere un valor inicial y lo mantiene hasta que este cambie, emitiendo dicho valor inicial o el valor actual a los nuevos suscriptores. En los servicios, dado que a menudo se inicializan antes que el componente que consume el servicio, normalmente es mucho mejor usar BehaviourSubject, de esta forma, nos aseguraremos de que dicho componente reciba los últimos datos actualizados incluso aunque no haya nuevas actualizaciones desde que se realizó la suscripción del componente a estos datos (cosa que Subject si necesitaría para emitir eventos).

Una vez dicho esto, para implementar todo esto mediante código, está claro que el servicio tiene que importar las clases BehaviorSubject y Observable. Usando BehaviourSubject para poder generar el stream de eventos y Observable para entregárselo a los observadores/componentes bajo demanda:

```
import BehaviorSubject, Observable from 'rxjs';
```

Lo siguiente sería declarar los BehaviourSubject tal y como se muestra aquí:

```
1 private dbReady: BehaviorSubject<boolean> = new BehaviorSubject(false);
2 public tableReady: BehaviorSubject<boolean> = new BehaviorSubject(false);
3 public tableForResultsReady: BehaviorSubject<boolean> = new BehaviorSubject(
4     false);
5
6 selectedTable = new BehaviorSubject<Elem>([]); // SEJMM DS007; Preparamos
7     para tabla creada mediante "Crea tu tabla"
8 selectedTableForGame = new BehaviorSubject<Elem>([]); // SEJMM DS009.2; Fix
9     memory leak provocado por suscripción y Fix de repetición de tablas
10    provocado por suscripción
11 selectedTableForResults = new BehaviorSubject<ResElem>([]); // SEJMM DS011;
12     Observable para mostrar en la tabla de resultados
13 tablesArrayName = new BehaviorSubject<string>([]); // SEJMM DS007; Preparamos
14     para multitabla.
15 tablesArrayNameForResults = new BehaviorSubject<string>([]); // SEJMM DS011;
16     Para obtener tablas de resultados
```

Código 3.9: Fichero database.service.ts, declaración BehaviourSubjects

En mi caso, creé BehaviourSubjects con dos propósitos distintos, debido a una problemática que me supuso el uso de estos que os explicaré en el apartado 3.3.3 referente a la página Une Palabras, ya que en la página Elige Tablas dicha problemática no se da. Ahora, es necesario que os detalle como estos BehaviourSubjects generan los eventos de aviso de cambio para los observadores.

```
1 loadTables() {
2     const query = 'SELECT tbl_name FROM sqlite_master WHERE type = \'table\' AND
3         tbl_name NOT LIKE \'sqlite_%\' AND tbl_name NOT LIKE \'res_%\'';
4     return this.database.execSQL(query, []).then(data => {
5         const tables: string[] = [];
6         if (data.rows.length > 0) {
7             for (let i = 0; i < data.rows.length; i++) {
8                 tables.push(data.rows.item(i).tbl_name);
9             }
10        }
11    }
12    this.tablesArrayName.next(tables);
13}
```

Código 3.10: Fichero database.service.ts, loadTables

Como podéis contemplar en el código anterior, el método `next` (utilizado sobre uno de los BehaviourSubject declarados) es el encargado de crear los eventos, como argumento le pasamos el `array` de tablas existentes en base de datos en ese momento, para que luego los observadores puedan saber cómo estaba el `array` al producirse este evento.

La última acción que vamos a necesitar para completar el trabajo dentro del servicio es la generación del observable, que se entregará a todos aquellos componentes que quieran observar cambios en el almacén de datos.

Es interesante este paso, puesto que el observable es un consumidor de los eventos del BehaviourSubject y es de sólo lectura. Es decir, puede estar atento a eventos, pero es incapaz de hacer nada más. Por tanto, lo único que el servicio entregará a externamente, a cualquier componente que lo necesite, es este observable. Mediante el observable, los componentes sabrán cuando el recurso observado en base de datos se ha modificado, pero ningún componente podrá generar nuevos eventos de cambio en la base de datos (tan solo pedirle que lo haga, llamando a funciones como `loadTables`), ya que es una responsabilidad del servicio y tarea principal del subject privado.

El observer se creará mediante un método del BehaviorSubject llamado `asObservable`. Para ello, usaremos un método *getter*<sup>3</sup> para devolver el observable, que tendrá el siguiente aspecto:

```
1  getTables(): Observable<string[]> {
2      return this.tablesArrayName.asObservable();
3  }
```

Código 3.11: Fichero `database.service.ts`, `getTables`

Una vez comentada la parte del servicio, le toca su turno a los componentes. Recapitulemos, aquellos componentes **escuchando eventos** son los observadores (observer). Estos observadores declaran y usan el observable y, **mediante una suscripción**, estarán enterados de cualquier cambio en aquello que se está observando. Ahora, os detallaré cómo se implementa esto en Elige tabla, siendo esto aplicable a cualquier otro componente que actué como observador en la aplicación.

Si echamos la vista atrás, en concreto hacia el Código 3.6, dentro de la función `ngOnInit()`, lo primero que se hace es obtener el observable del estado de la base de datos mediante el método `getDatabaseState()` y se suscribe a él, esperando a que el correspondiente BehaviourSubject emita el valor `True` esperado en el `if (rdy)` de la línea inferior. Este valor es emitido tras la carga de la semilla en la carga del servicio de base de datos, por lo que en este punto, siempre estará disponible (ya que la base de datos la carga el módulo raíz).

Posteriormente, vemos la llamada a `loadTables()` pidiendo al servicio que lance el evento con las tablas disponibles en base de datos y finalmente la suscripción a `getTables()` que recogerá el evento lanzado junto a sus datos y los procesará, ordenando las tablas alfabéticamente.

<sup>3</sup>Un getter es un método comúnmente usado en programación para poder pedir un recurso privado desde un sitio externo a ese recurso desde el que de manera natural no tendría acceso.

Tan solo nos queda un último detalle para finalizar esta incursión en el mundo de los observables, el darse de baja de ellos. Desuscribirse de un observable es fundamental, dado que de no hacerlo, estaríamos provocando un *leak* de memoria<sup>4</sup>, y al cabo del tiempo, tras hacer muchas suscripciones, nos quedaríamos sin memoria, lo que desembocaría en desastre, dado que este tipo de errores no son permisibles en el mundo de la informática.

¿Cómo podemos evitar esto?, la respuesta es sencilla, volviendo al mismo Código 3.6, veremos que tras la exportación de la tabla se declara una variable `unsubscribe$` (El `$` indica, por convenio y ayuda a los desarrolladores, que la variable cuyo nombre lo lleva es un observable, aunque en mi caso, apenas lo usé), esta variable se utiliza junto una *pipe/tubería* `takeUntil` antes de realizar las suscripciones, dado que la función de dicha tubería es la de asegurar la suscripción hasta que se cumpla la condición entre paréntesis, que en este caso sería que dicha variable `unsubscribe$` reciba algún valor mediante `next()`. Esto último, se llevará a cabo desde la función `ngOnDestroy`, finalizando con ello la suscripción.

---

<sup>4</sup>Un *leak* de memoria o fuga de memoria es un error de software que ocurre cuando un bloque de memoria reservada no es liberada en un programa de computación. Comúnmente ocurre porque se pierden todas las referencias a esa área de memoria antes de haberse liberado.

**3.1.4. Resultado final**

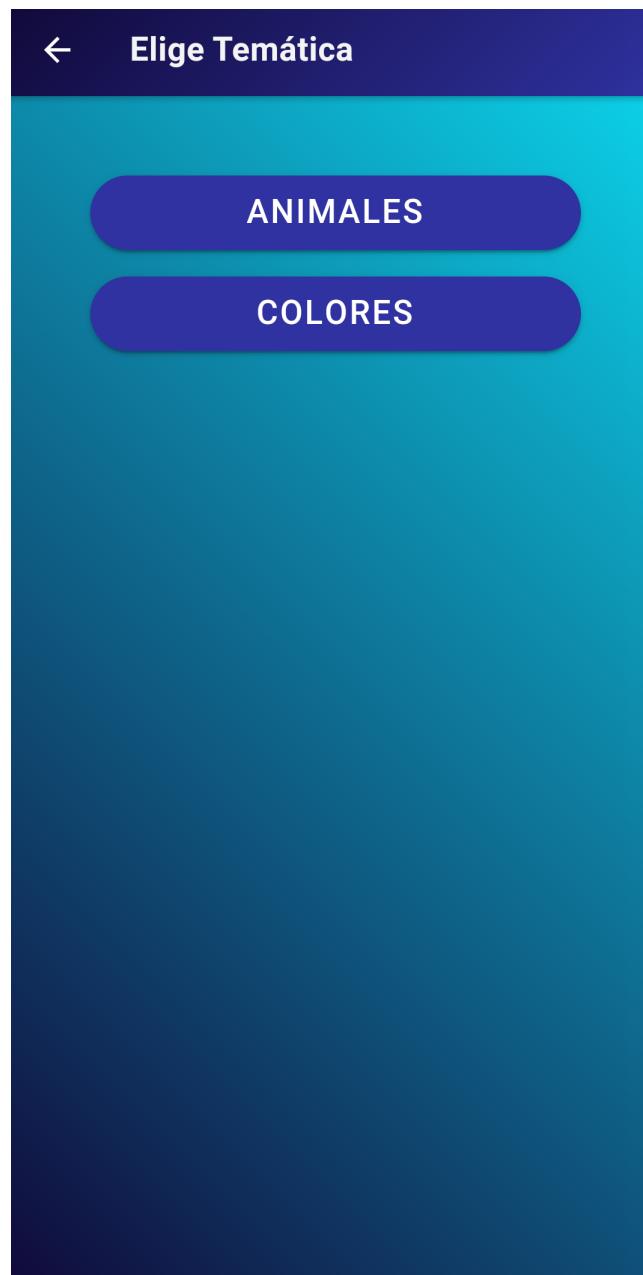


Figura 3.2: Elige temática

### 3.2– Página Elige juego

#### 3.2.1. Mock Up

A continuación mostramos otra de las páginas más sencillas, donde nuestro objetivo será seleccionar el juego que nos interese. Hasta ahora, disponemos de dos juegos **Une palabras** y **Di mi nombre**. Como nombramos en el apartado anterior, esta página dispone de una flecha que representa la vuelta a la página anterior, en este caso, volvería a **Elige temática**.



Figura 3.3: Mockup Elige juego

### 3.2.2. Vistas

Esta página, tan solo se compone de dos botones fijos, uno para cada juego de los dos que hemos creado. En caso de querer ampliar la aplicación en un futuro añadiendo más juegos, sería necesario añadir tantos botones como juegos nuevos se creen. Aparte de esto, poco más hay que comentar acerca de la vista de **Elige Juego**.

### 3.2.3. Código

Al ser ésta página tan sencilla, aprovecharemos para explicar un concepto de Angular utilizando en ella y en prácticamente casi todas las páginas de Say My Name, el *routing* o navegación.

#### Navegación

En Ionic 4, lo propio es usar el **router** de Angular, el Router Angular es una de las librerías más importantes en una aplicación Angular. Sin él, las aplicaciones serían aplicaciones de vista única o de contexto único o no podrían mantener su estado de navegación en las recargas del navegador[1].

Dicho esto, desde un archivo HTML de nuestra aplicación podríamos usar la directiva **routerLink** sobre un botón como hacemos en el siguiente fichero de nuestra *app*:

```

1 <ion-img [src]="CongratulationsCover"></ion-img>
2
3 <ion-button expand="block" [routerLink]="/home-results" routerDirection="root"
4   (click)="comeBack()" color="success">
5   Volver
</ion-button>
```

Código 3.12: Fichero notifications.component.html, Uso de routerLink

La parte importante aquí es el **<ion-button>** y la directiva **routerLink**. RouterLink funciona con una idea similar a la de los **hrefs** típicos, pero en lugar de construir la URL como una cadena, puede construirse como una matriz, lo que nos permitirá usar rutas más complicadas. Aquí se ve otra de las ventajas de usar Ionic 4, ya que desde que actualizaron a esta versión, routerLink ofrece otra posibilidad para viajar entre páginas sin necesidad de usar el evento **click** en un botón como se hacía en Ionic 3 (obligandonos esto a crear una función en el controlador que captase dicho evento para desplazarnos a otra vista).

También podremos navegar desde nuestro código **typescript** utilizando la API del router, usaremos un ejemplo para ver más claro como se llevaría a la práctica el Angular Router en este contexto:

```

1 import { Component } from '@angular/core';
2 import { Router } from '@angular/router';
3 @Component({...})
4 export class LoginComponent {
5   constructor(private router: Router){}
6   navigate(){
7     this.router.navigate(['/home-page'])
8   }
9 }
```

Código 3.13: Ejemplo Angular Router desde .ts

Pese a que todo esto suene muy bien, en Say My Name apenas he usado el Router de Angular (de la forma en que la documentación oficial de Ionic dice que lo hagamos), dado que pese a tener animaciones al cambiar de página, no nos permitía elegir la dirección de esta. En cambio **NavController** la alternativa usada en Ionic 3, si nos lo permitía, pero dado que nuestra aplicación implementa *Lazy Loading*,<sup>5</sup> (la única incompatibilidad con dicho mecanismo dado que su implementación depende de Angular Router) no pude usarla para llevar a cabo las navegaciones desde los controladores pese al mayor control que nos ofrecía.

Tras mucho investigar, descubrí que como Ionic/Angular se habían integrado tan estrechamente en el pasado, Ionic 4 trae incluido consigo un paquete @ionic/angular específico que ayuda a la integración con Angular. El NavController está incluido en este paquete y puede ser usado para interactuar con el Router de Angular, pero todavía está usando el Router de Angular por debajo. En conclusión, el NavController existente en Ionic 4 es diferente al NavController en Ionic 3, aunque haya conservado el mismo nombre[29], y por lo tanto es el que he usado en la mayoría de casos navegar en Say My Name.

Antes de continuar, indaguemos un poco más en este concepto del *Lazy Loading*, dado que lo considero un factor muy importante a tener en cuenta a la hora de crear una aplicación multiplataforma rápida, eficiente y con buenos tiempos de carga, ya que estas son características vitales para una *app* en el competitivo mercado actual. La forma de implementar el *Lazy Loading*, es la que podréis observar en el fichero de código 2.6. Es bastante similar a la navegación de Angular habitual, pero en lugar de proporcionar un componente para la ruta del *path*, proporcionamos una propiedad **loadChildren**. Como en este caso no estaríamos cargándolo todo por adelantado, esta propiedad está diciendo esencialmente "ve a este archivo para determinar qué componentes necesitan ser cargados". Finalmente, como valor de loadChildren, enlazaremos al archivo de módulo de la página mediante su ruta, y también necesitaremos enlazarlo mediante un # con el nombre de la clase del módulo. De todas formas, actualmente, la implementación del *Lazy Loading* es muy sencilla, ya que este es el modo por defecto en el que Ionic CLI crea un proyecto de Ionic y el modo en el que se crean las nuevas páginas.

Volviendo a NavController, para distinguir los distintos tipos de animaciones, este nos proporciona una serie de funciones:

- **navigateForward**: Animación de transición de página hacia la derecha.
- **navigateBack**: Animación de transición de página hacia la izquierda.
- **navigateRoot**: Animación de transición de página hacia arriba.

Una vez aprendidos e interiorizados bien todos estos conceptos, no tendréis problema ninguno a la hora de utilizar la navegación de Angular, y además, sabréis cómo crear una aplicación que optimice al máximo el consumo de sus recursos.

---

<sup>5</sup>La carga diferida, es un patrón de diseño comúnmente usado en la programación informática que consiste en retrasar la carga o inicialización de un objeto hasta el mismo momento de su utilización. Esto contribuye a la eficiencia de los programas, evitando la precarga de objetos que podrían no llegar a utilizarse, como, por ejemplo, la carga de imágenes. Aplicado a Angular, la idea básica es que descomponemos nuestra aplicación en trozos más pequeños y sólo carga lo que es necesario en ese momento. Cuando se carga la aplicación por primera vez, sólo es necesario cargar los componentes necesarios para visualizar la primera pantalla.

### **3.2.4. Resultado final**

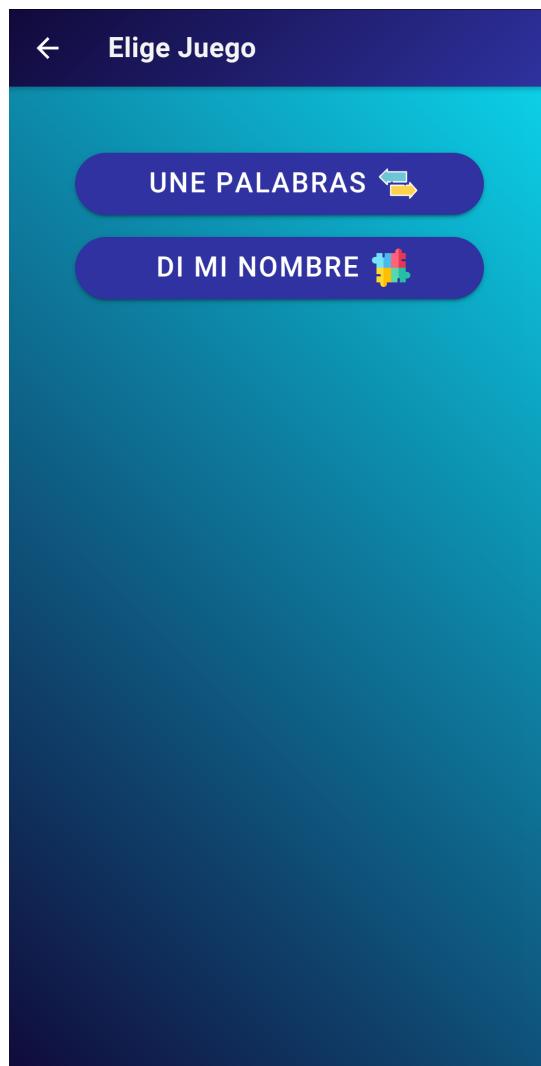


Figura 3.4: Elige juego

### 3.3– Página Une palabras

#### 3.3.1. Mock Up

Para este juego, llamado **Une Palabras** como dice el propio título, necesitamos como mínimo de dos palabras, y por lo tanto dos columnas. La primera columna estará compuesta por las palabras escritas en **español**, mientras que en la segunda encontraremos las escritas en **inglés**. Para que haya una clara distinción entre ambas, dejaremos en la parte superior de las columnas, los idiomas a los que equivale cada columna, con vistas a cambios futuros como la introducción de nuevos idiomas.



Figura 3.5: Mockup Une palabras

### 3.3.2. Vistas

En esta vista, básicamente usamos una fila `ion-row` y dos columnas `ion-col`. Al no especificar un número junto a dichas columnas, y ser solo dos, Ionic junto a Angular presupondrán que les estoy indicando que quiero que las columnas ocupen la misma cantidad de espacio horizontal. A su vez, cada columna, tiene una lista `ion-list`, en la que se iterarán un máximo de diez elementos obtenidos aleatoriamente desde la tabla pasada por la página ELIGE JUEGO.

En circunstancias normales, estos elementos (en una sola fila `ion-row`) se visualizarían en línea, y no obtendríamos el resultado diseñado en nuestro *mockup*, pero dado que estamos usando `ion-button` para realizar la iteración, y estos se encuentran dentro de una lista `ion-list`, es esta lista la que se encarga de colocar los botones en disposición vertical.

```

1   <ion-col>
2     <div class="ion-text-center">
3       <ion-text class="fw700" color="light" class="columnTitle">
4         <h3>Español</h3>
5       </ion-text>
6       <ion-icon size="large" src="../../assets/img/espana.svg"></ion-icon>
7     </div>
8     <ion-list class="bg-list">
9       <ion-button strong fill="clear" expand="block"
10      [ngClass]="{'sib': setElementColour(tableElement.id, 0) == 1, 'nob':
11        setElementColour(tableElement.id, 0) == 0, 'eligiendob':
12        setElementColour(tableElement.id, 0) == 2, 'defaultb':
13        setElementColour(tableElement.id, 0) == 3}"'
14        *ngFor="let tableElement of tableArrayElements | alterOrder" (click)="
15          selectWords(tableElement.id,0)"
16        [id]="tableElement.id">
17          {{ tableElement.spanishName }}
18        </ion-button>
19      </ion-list>
20    </ion-col>
```

Código 3.14: Fichero une-palabras.page.html

Como podréis observar en la captura, este es el procedimiento que usa cada columna para obtener los elementos que contendrá. Hace uso del comando `ngClass` para utilizar una clase CSS3 u otra en función del valor devuelto por una función, gracias a esto, los elementos se representarán de un color u otro en función de su estado.

Luego, también se hace uso de un evento (`click`) para mandar el ID del elemento en base de datos de la tabla jugada junto a un 0 o un 1 indicando que el botón pulsado se encuentra en la columna izquierda o derecha respectivamente.

La única otra cuestión importante a mencionar es el uso de una tubería personalizada utilizada en la iteración mediante `ngFor` que se corresponde con el uso del carácter `|` tras el nombre de dicha tubería. Todo este procedimiento lo veremos más detalladamente en el apartado 3.4.3.

### 3.3.3. Código

Antes de nada, os contaré que procedimiento usé para mandar el nombre de la tabla con la que trabajaremos en UNE PALABRAS desde la página ELIJE JUEGO y, por supuesto, cómo adquirir dicho parámetro.

#### Paso y recepción de parámetros

Como ya os mencioné anteriormente, he usado NavController para llevar a cabo prácticamente todo el control de la navegación de la aplicación. Pues bien, el paso de parámetros entre páginas también es su cometido. Para ello, tan solo es necesario añadir un parámetro más a la función utilizada para pasar a otra página, como por ejemplo:

```
1 goToUnePalabras() {
2     this.navCtrl.navigateForward(['/une-palabras', this.tableNameArg]);
3 }
```

Código 3.15: Fichero `select-game.page.ts`, paso de parámetros

Una vez hecho esto, tan solo tendremos que recibirla desde la página a la que nos hemos desplazado, para ello importaremos la clase `ActivatedRoute` desde `@angular/router` y después haremos lo siguiente:

```
1 constructor(private db: DatabaseService, // DS002: Base de datos SQLite
2             public popoverCtrl: PopoverController, // DS006: Implementación de ion-
3                         popover para mostrar el final del juego
4             private activeRoute: ActivatedRoute // DS007: Preparación multitabla
5             ) { }
6
7 ngOnInit() {
8     this.arguments = this.activeRoute.snapshot.paramMap.get('tableName'); // DS007: Preparación
9             multitabla
```

Código 3.16: Fichero `une-palabras.page.ts`, recepción de parámetros

Ahora, por si no quedo totalmente claro al explicar los observables, os contaré a grandes rasgos en que consiste el ciclo de vida de una página de Ionic. Así entenderéis con mucha más claridad el código del controlador de este juego.

### El ciclo de vida

El ciclo de vida de una página de Ionic es, en resumidas cuentas, el estado en el que se encuentran dichas páginas. Ionic, nos da la posibilidad de contemplar varios estados posibles:[1]

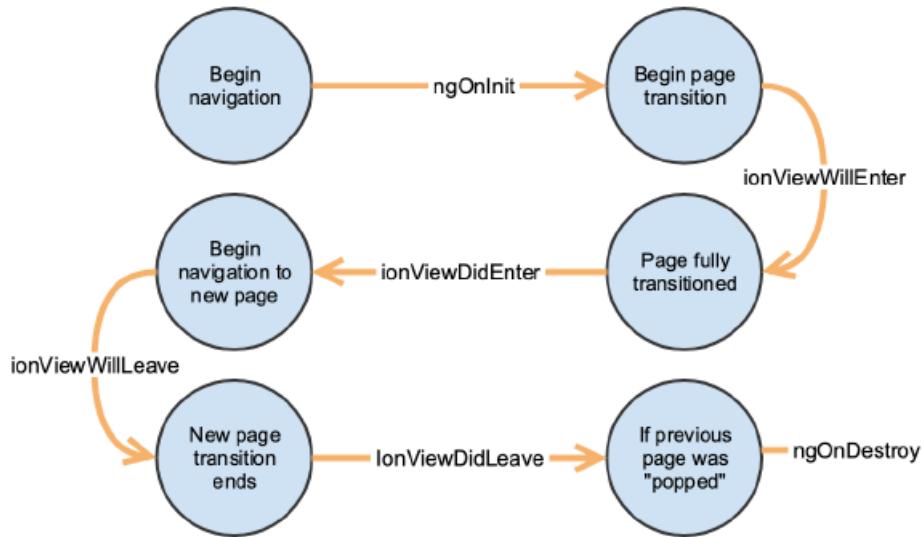


Figura 3.6: Ciclo de vida de una página de Ionic

De todos estos estados, yo solo he usado dos eventos. `ngOnInit` y `ngOnDestroy`.

- `ngOnInit`: Este evento comienza antes siquiera de que comience la animación de transición de la página, se lanza durante la inicialización del componente. En él, suelen inicializarse las variables locales implicadas en la página, se realizan las consumiciones/suscripciones de los servicios/observables, etc..
- `ngOnDestroy`: Disparado justo antes de que Angular destruya la vista. Útil para la limpieza de la página, como por ejemplo, darse de baja de los observables a los que nos hemos suscrito.

Para manejar el proceso del ciclo de vida, Ionic tiene su `RouterOutlet`<sup>6</sup>, llamada `<ion-router-outlet />`. Este, extiende el `<router-outlet />` de Angular con alguna funcionalidad adicional para permitir mejores experiencias para dispositivos móviles.

Cuando una aplicación está envuelta en `<ion-router-outlet />`, Ionic trata la navegación de una forma un poco diferente. Cuando navegas a una nueva página, Ionic mantendrá la página antigua en el DOM<sup>7</sup> existente, pero la ocultará de nuestra vista y cambiará a la nueva página.

<sup>6</sup>Actúa como un marcador de posición que Angular rellena dinámicamente basándose en el estado actual del router.

<sup>7</sup>*Document Object Model* o DOM (Modelo de Objetos del Documento) 'Modelo en Objetos para la Representación de Documentos) es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

La razón por la que hacemos esto es doble:

1. Por una parte, podremos mantener el estado de la página antigua (datos en la pantalla, posición de desplazamiento, etc...).
2. Por otra, podremos proporcionar una transición con un acabado más profesional y suave de vuelta a la página anterior, ya que no es necesario volver a recrearla.

Las páginas sólo se eliminan del DOM cuando, por ejemplo, se pulsa el botón Atrás en la interfaz de usuario o el botón Atrás del navegador.

Debido a este manejo especial, es posible que los métodos `ngOnInit` y `ngOnDestroy` no se disparen cuando pensamos, ya que `ngOnInit` sólo se lanzará cada vez que la página se cree de nuevo, pero no cuando se vuelva a navegar a la página pulsando el botón Atrás. Esto último no causaría problemas en nuestros juegos, ya que siempre son una página hoja o extremo del DOM de nuestra aplicación. `ngOnDestroy` sólo se disparará cuando una página sea “destruida”.

### Problemas derivados del uso de observables con el servicio de Base de Datos

Si echamos la vista atrás, en concreto a la explicación que dimos sobre los observables, recordaréis que os hablé de una problemática que se me dio derivada del uso de observables cuyos eventos de cambios se lanzaban desde el servicio de base de datos. Dicha anomalía, se produjo al suscribirme a los observables desde las páginas referidas a los juegos, es decir, al suscribirme al `BehaviourSubject` llamado `selectedTableForGame`, encargado de lanzar los eventos con los elementos actualizados obtenidos desde la tabla pasada como parámetro en la función `loadTableForGame`, definido en nuestro servicio de BD. El problema, observado desde el punto de vista de un usuario era que las columnas del juego no contenían nada, estaban vacías.

Tras un largo estudio, me percaté de que quizás este fallo se debía a la mayor complejidad de la `query` realizada a la base de datos, en la que teníamos que llenar un array de objetos del tipo `Elem` (interfaz definida también en el servicio) con las distintas columnas de la tabla solicitada desde los componentes/juegos/formularios. En comparación, la query que se hacía para obtener todas las tablas existentes en la base de datos, usada para listar dichas tablas en las páginas de nuestra `app` tan solo recorría una columna y la metía en un array, operación mucho menos costosa en cuanto a tiempo. Esta mayor complejidad, podría provocar que la suscripción se realizase antes de que nuestra función `loadTableForGame` enviase un nuevo evento con las tablas actualizadas por primera vez, lo que se provocaría que la suscripción cogiese como primer valor el inicializado en el `BehaviourSubject`.

Ahora, os estaréis preguntado ¿Cuál es la solución a este problema? Pues... en este caso no fue fácil para mi descubrirla, tras muchos intentos fallidos, di con el *kit* de la cuestión. Se me ocurrió, que quizás si usaba otro `BehaviourSubject` para informar del estado de ese procesamiento de la `query` solicitada a través de `loadTableForGame`, esto podría solucionar el problema, y de hecho así fue.

Para ello, utilicé el comando `next(True)` tras acabar el procesamiento y haber llenado la variable que posteriormente se pasaría como parámetro en el `next` del otro `BehaviourSubject` encargado de informar al observador de los cambios o el estado actual de los elementos de la tabla.

Desde el punto de vista del componente, modifiqué el código para que no solo se comprobase si el estado de la base datos era correcto, sino para que también comprobase el

estado de el procesamiento de la query solicitada mediante `loadTableForGame`. De esta forma, la suscripción se realizaría si y solo si el estado de dicho procesamiento indicase que este ha finalizado. Otra modificación que realicé, fue asegurarme de que en la función `ngOnDestroy` se reinicializaban a sus valores por defecto los dos BehaviourSubject utilizados, para así evitar que en una futura suscripción tomase este como un valor válido. De esta forma, todos los problemas relacionados con los observables quedaron solucionados por siempre.

Entiendo que todo esto, así leído, suene muy confuso y abstracto, por lo que os pondré un par de capturas de código (desde el lado del componente y del servicio) para que lo entendáis mucho mejor lo que he descrito:

```
1  ngOnInit() {
2      this.argumentos = this.activeRoute.snapshot.paramMap.get('tableName'); // Obtenemos la tabla enviada desde select-table. DS007: Preparación multitable
3      this.db.getDatabaseState().pipe(takeUntil(this.unsubscribe$)).subscribe(rdy => {
4          if (rdy) {
5              this.db.loadTableForGame(this.argumentos, 10);
6          }
7      });
8      this.db.getTableState().pipe(takeUntil(this.unsubscribe$)).subscribe(tableRdy => {
9          if (tableRdy) {
10             // A continuación nos suscribiremos al observable que almacena el resultado de SELECT * FROM TABLE desuscribiendonos inmediatamente despues con la pipe(take(1))
11             this.db.getSelectedTableForGame()
12                 .pipe(takeUntil(this.unsubscribe$)).subscribe(table => { // SEJMM DS009.2;
13                     Fix memory leak provocado por suscripción y Fix de repetición de tablas provocado por suscripción
14                     this.tableArrayElements = table;
15                     /* Inicializamos el mapa id-valorMap<idColumna, valor> para cada entrada en la tabla de animales con la que se va a jugar. De este modo definiremos como valor por defecto 3 (Color por defecto, no elegido).
16                     */
17                     for (const elem of this.tableArrayElements) {
18                         const auxMap = new Map<number, number>();
19                         auxMap.set(0, 3); // Columna izquierda; Color default
20                         auxMap.set(1, 3); // Columna derecha; Color default
21                         this.idResultsMap.set(elem.id, auxMap);
```

Código 3.17: Fichero `une-palabras.page.ts`, solución problema observables

```

1  loadTableForGame(tableName: string, limit: number) {
2    let query = 'SELECT * FROM [';
3    query = query.concat(tableName + '] LIMIT ' + limit); // DS003.3.1: UPDATE
4    REORDER CON LIMITE
5    return this.database.executeSql(query, []).then(data => {
6      const table: Elem[] = [];
7      if (data.rows.length > 0) {
8        for (let i = 0; i < data.rows.length; i++) {
9          table.push({
10            id: data.rows.item(i).id,
11            spanishName: data.rows.item(i).spanishName,
12            englishName: data.rows.item(i).englishName
13          });
14        }
15        this.selectedTableForGame.next(table);
16        this.tableReady.next(true);
17      });
}

```

Código 3.18: Fichero database.service.ts, solución problema observables

Para finalizar este capítulo, en lugar de explicaros paso a paso cada una de las líneas de código que llevan la lógica de este juego, cosa que veo poco provechosa e innecesaria puesto que prácticamente dichas funciones tienen un comentario explicativo por cada línea de código, prefiero hablaros del *timer* asíncrono que he implementado para utilizarlo dentro de la lógica de la aplicación.

La función de dicho *timer*, es esperar unos instantes tras equivocarse un jugador al hacer una selección de palabras errónea (botones rojos) para después devolver dichos botones al estado por defecto (blanco). Lo interesante de todo esto, es que dicha función no pararía la ejecución de la aplicación durante esos instantes, como si ocurriría con un *sleep* tradicional, sino que simplemente se pararía a sí misma esperando (750 ms) el momento adecuado (*await*), mientras todo sigue su transcurso habitual, y cuando llegue ese momento continuaría con su ejecución.

Esto, se realizaría de la siguiente manera:

```

1  delay(ms: number) {
2    return new Promise( resolve => setTimeout(resolve, ms) );
3 }

```

Código 3.19: Fichero une-palabras.page.ts, función para timer asíncrono

```

1  (async () => {
2    await this.delay(750);
3    auxMap.set(column, 3); // Por defecto
4  })();

```

Código 3.20: Fichero une-palabras.page.ts, uso de función timer

**3.3.4. Resultado final**

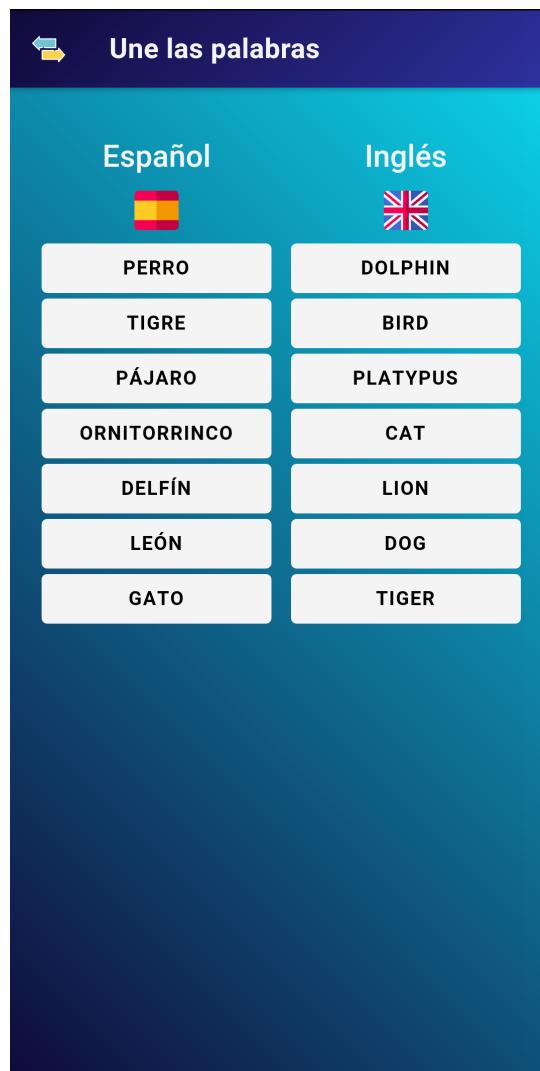


Figura 3.7: Une palabras

### 3.4– Página Di mi nombre

#### 3.4.1. Mock Up

Este juego tiene un poco más de complejidad en comparación con el juego anterior, encontraremos una palabra en español a continuación del encabezado. Justo por debajo de esta, colocaremos en una **fila de botones** las letras desordenadas de la palabra en inglés correspondiente a la traducción de la palabra en español colocada arriba de esta, de tal manera que la persona, al pulsar sobre alguno de estos botones, hará que aparezca la letra elegida sobre el guión destacado.

En caso de que la letra seleccionada no sea la correcta, he colocado un botón de retroceso, cuya función será la de **borrar** de una en una las letras introducidas. A su vez, al lado derecho de este botón, se encuentra otro botón con un tick, con el objetivo de **verificar** si la palabra es o no correcta, dicha verificación **solo será efectuada** en caso de que todas las letras **hayan sido colocadas** sobre todos los guiones.

A continuación, encontraremos **dos recuadros**, cuya función será la de exponer y contabilizar los resultados en cuanto a errores(**rojo**) y aciertos(**verde**). A medida que vayamos confirmando las palabras, nos desplazaremos hacia la derecha a otra página con el mismo formato hasta un total de **cinco páginas**, y se irán actualizando dichos resultados, teniéndolos sincronizados en todo momento.

Por último, y a modo de conocer en qué palabra nos encontramos, hemos colocado **cinco puntos** que representarán las cinco palabras escogidas de la base de datos, el punto más claro será el que nos indique el número de la palabra con la que estamos jugando.

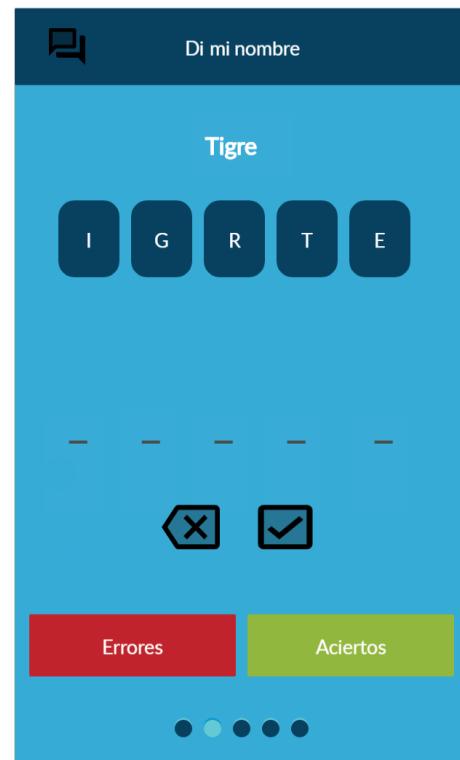


Figura 3.8: Mockup Di mi nombre

### 3.4.2. Vistas

Como dato inicial, quería comentaros algo que he omitido hasta este momento. Di MI NOMBRE es una página, y como tal ha de ser creada. Una página, en realidad no es más que un componente, a modo de directorio en nuestro proyecto, con una estructura específica que, para ser totalmente funcional, debe haber inicializado su módulo y `app-routing.module.ts` con los valores necesarios.

Para vuestra tranquilidad, os digo que yo no creé ni una sola página a mano, ese cometido forma parte del Ionic CLI, el cual nos aporta los siguientes comandos para la creación de páginas, componentes y más, encargándose de incluir todas las dependencias básicas que este pudiese tener con el resto de la aplicación.

```
$ ionic generate
$ ionic generate page
$ ionic generate page contact
$ ionic generate component contact/form
$ ionic generate component login-form --change-detection=OnPush
$ ionic generate directive ripple --skip-import
$ ionic generate service api/user
```

Figura 3.9: Comandos Ionic CLI

La novedad más destacable a comentar sobre la vista de este juego es el uso de `slides`. Este, es un contenedor de varias secciones/diapositivas. Cada una de ellas puede ser deslizada o arrastrada dejando a la vista otra diapositiva. Puede contener el número de `slide` o diapositivas que deseemos, e iterando con `ngFor` sobre ellas podemos dar un tratamiento similar a todas.

Para los `Slides` Ionic nos proporciona un gran número de propiedades y funciones que nos ayudarán a controlar todos los aspectos que deseemos de ellas [1]. En nuestra aplicación Di MI NOMBRE yo he usado un atributo `pager` y algunas funciones:

- `pager`: Atributo que, de valer True, nos permitirá mostrar pequeños puntos en la parte inferior del `slide` permitiéndonos así saber en qué página estamos.
- `getActiveIndex`: Devuelve el índice del `slide` actual.
- `isEnd`: Nos indica si nos encontramos o no en el último `slide`.
- `lockSwipes`: Bloquea o desbloquea el desplazamiento entre slides en ambas direcciones.
- `slideNext`: Nos permite hacer una transición hacia el próximo `slide`.

La propiedad `pager` será utilizada desde la Vista de la siguiente manera:

- `<ion-slides #slides pager=true>`

En dicha línea de código, el `#slides` sirve para permitir al controlador hacer uso de los `slides` y de sus funciones, pero de eso ya hablaremos en el apartado 3.4.3 junto al uso y procedencia de la tubería `shuffle`.

### 3.4.3. Código

Antes de entrar en detalles con los `slides` desde del punto de vista del controlador, hablaremos de un recurso muy utilizado en toda la aplicación, pero sobre todo en este juego Di mi nombre, las tuberías. En esta página usamos dos tipos de tuberías distintas, las implementadas a través de un paquete localizado en `node_modules` (como `takeUntil` de `rxjs` utilizada para desuscribirnos de los observables) y las implementadas manualmente por nosotros mismos. Vamos a hablar de ellas:

#### Tuberías

Como os he comentado en el apartado anterior, la vista de DI MI NOMBRE utiliza una tubería llamada `shuffle`, dicha tubería pertenece a la librería `ngx-pipes` obtenida desde el siguiente enlace: <https://www.npmjs.com/package/ngx-pipes#installation>.

Para instalar dicha librería tendremos que usar el siguiente comando `npm` desde nuestra consola:

```
npm install ngx-pipes --save
```

Una vez instalada, su implementación es sencilla, tan solo tendremos que importar el paquete completo generado en `node_modules` al instalar la librería o, para gastar la mínima cantidad de memoria posible, importar la clase/subgrupo de tuberías en función de nuestras necesidades en el módulo del componente en el que deseemos utilizar dichas tuberías.

En el caso de nuestro proyecto, solo importa la clase `NgArrayPipesModule`, dado que es la que contiene la tubería `shuffle` que necesitamos para desordenar el array de letras utilizados como botones en el juego. Esta librería nos es de mucha utilidad, ya que nos ayudará a conseguir la funcionalidad diseñada en nuestro *mock up*.

Pero qué ocurriría si ninguna de estas clases nos aportase una tubería con la funcionalidad que deseamos, ¿Nos quedaríamos sin nuestra preciada funcionalidad? Por supuesto que no, podemos crear nosotros mismos una tubería que implemente la funcionalidad que deseemos. Para ello, lo primero que haremos será crear en nuestro proyecto un directorio `pipes` al mismo nivel que `pages`. En él incluiremos un módulo `pipes.module.ts` (con decorador `@ngModule`) que nos permitirá importar, declarar y exportar nuestras tuberías permitiendo así que cualquier componente que lo importe desde su módulo pueda utilizarlas:

```
1 import { NgModule } from '@angular/core';
2 import { AlterOrderPipe } from './alter.order.pipe';
3 import { SpellPipe } from './spell.pipe';
4
5 @NgModule({
6   declarations: [AlterOrderPipe, SpellPipe],
7   imports: [],
8   exports: [AlterOrderPipe, SpellPipe],
9 })
10
11 export class PipesModule {}
```

Código 3.21: Fichero `pipes.module.ts`

Después, crearemos la tubería que necesitemos, teniendo en cuenta que esta tendrá una entrada que será procesada en función de la funcionalidad que deseemos aportarle a nuestra tubería y una salida con las modificaciones sobre la entrada. Un ejemplo, sería la ya citada tubería SpellPipe:

```

1 import { Pipe, PipeTransform } from '@angular/core';
2
3 @Pipe({
4   name: 'spell'
5 })
6 export class SpellPipe implements PipeTransform {
7   transform(word: string, args: any[]): string[] | boolean {
8     if (word === null) {
9       console.log('SpellPipe: palabra nula');
10      return false;
11    }
12    const charactersArray: string[] = [];
13    for (let i = 0; i < word.length; i++) {
14      // const words = value[i].split(' ');
15      charactersArray.push(word[i]);
16    }
17    return charactersArray;
18  }
19}

```

Código 3.22: Fichero spell.pipe.ts

Como ya he dicho antes, para poder usarla desde un componente tan solo tendremos que importarla desde su módulo y ya podrá ser usada desde la Vista, en caso de querer usarla desde el controlador, tendremos que importar la librería en concreto a utilizar y consumirla en el constructor de la clase de dicho controlador (como se hace con SpellPipe en di-mi-nombre.page.ts).

## Slides

Con respecto al uso de los **slides** desde el controlador no nos extenderemos en demasiado. Lo único que tenemos que tener claro es el procedimiento necesario para captar el componente de Ionic **slides** usado en la vista desde el controlador. Una vez conseguido, podremos llamar a cualquier función propia de dichos **slides** como llamaríamos a cualquier función corriente. Para hacer más sencilla esta explicación, tan solo nos centraremos en llamar a la función **lockSwipes**, de esta forma podremos extrapolar la misma fórmula para usar todas las funciones que necesitemos.

En primer lugar, para captar el componente usaremos el decorador **@ViewChild** exportar la clase del controlador: **@ViewChild('slides') slides;**

En segundo lugar, crearemos una función que devuelva el mismo tipo que la función que deseamos utilizar, en este caso sería un Boolean:

```

1 lockSwipes(lock: boolean) {
2   this.slides.lockSwipes(lock);
3 }

```

Código 3.23: Fichero di-mi-nombre.page.ts, lockSwipes

De esta forma, podremos usar cualquier función del componente `slides` desde el controlador, dándonos muchísimas posibilidades y control a la hora de lograr llevar a cabo una idea más ambiciosa como lo es Di Mi NOMBRE. Ahora, explicaremos como usar otro componente de Ionic también usado en este juego que puede causar confusión dado que solo tiene cabida desde el punto de vista del controlador, este es el **Toast**.

### Toast

Un **Toast** es una notificación sutil que se utiliza comúnmente en las aplicaciones móviles modernas. Puede utilizarse para proporcionar información sobre una operación o para mostrar un mensaje del sistema. Siempre aparece sobre el contenido de la aplicación, suele ser eliminada de la vista por la aplicación tras un tiempo determinado para reanudar la interacción del usuario con la aplicación. También puede contener botones y ser eliminado por el mismo usuario a través de ellos pero no es el uso más común.

En mi caso, lo implementé como una función asíncrona para que pudiese ser llamado desde cualquier punto del componente sin dañar la ejecución normal del código, de esta forma podemos notificar al usuario cuando intenta confirmar mediante el botón check de que no ha introducido los valores necesarios como para poder llevar a cabo el chequeo. Extrapolando, los **Toast** son un recurso de notificación muy útil en casi todas las situaciones imaginables, dado que no suponen ningún cambio en la navegación para el usuario y a cambio aportan mucha información.

```

1  async presentToast() {
2      const toast = await this.toastCtrl.create({
3          message: 'Debe llenar los huecos antes de confirmar',
4          duration: 2000
5      });
6      toast.present();
7  }

```

Código 3.24: Fichero `di-mi-nombre.page.ts`, `presentToast`

### popover

De forma similar al **Toast** tenemos el `ion-popover`, este se diferencia del anterior en que normalmente está ideado para usarse como ampliación de la botonera de una barra de herramientas, aunque en nuestra aplicación yo la he usado para notificar al usuario de que el juego ha terminado. Para ello, en lugar de implementarlo directamente como hicimos con **Toast** hice uso del **Componente notifications**. Resalto Componente porque en este caso no estamos hablando de él como si de una página se tratase. Sino de un componente aislado y reutilizable en múltiples páginas (este se utiliza tanto en Una Palabras como en Di Mi Nombre).

Implementar el `ion-popover` como un componente nos permite darle una mayor personalización asignándole la apariencia que deseemos a través de su vista.

```

1 <ion-img [src]="congratulationsCover"></ion-img>
2
3 <ion-button expand="block" [routerLink]=[ '/home-results' ]" routerDirection=""
4   root" (click)="comeBack()" color="success">
5   Volver
6 </ion-button>
```

Código 3.25: Fichero notifications.component.html

En la parte del controlador de dicho componente es necesario utilizar el controlador de `ion-popover` (`PopoverController`) de la siguiente forma:

```

1 import { PopoverController } from '@ionic/angular';
2 @Component({
3   selector: 'app-notifications',
4   templateUrl: './notifications.component.html',
5   styleUrls: ['./notifications.component.scss']
6 })
7 export class NotificationsComponent implements OnInit {
8   congratulationsCover = 'assets/img/minions.gif'; // SEJMM DS006: Imagen portada
9   modificada
10  constructor(public popoverCtrl: PopoverController) {} // DS006: Implementación
11    de ion-popover para mostrar el final del juego
12
13  ngOnInit() {
14    // this.popoverCtrl = this.hijo.popoverCtrl;
15  }
16  /**
17   * SEJMM DS006: Implementación de ion-popover para mostrar el final del juego
18   */
19  async comeBack () {
20    return await this.popoverCtrl.dismiss();
```

Código 3.26: Fichero notifications.component.html

Ya solo nos quedaría declarar el componente desde el módulo raíz (Código 2.2), en `declarations` para poder utilizarlo desde todos los componentes/páginas de la `app`.

Un ejemplo de este uso, sería el realizado en el componente Di Mi Nombre:

```

1 async notifications() {
2   const popover = await this.popoverCtrl.create({
3     component: NotificationsComponent,
4     event: null,
5     animated: true,
6     showBackdrop: true, // Muestra fondo translúcido de la page que invocó tras
7       el popover
8     backdropDismiss: false // Evita que el popover desaparezca al pulsar sobre
9       el backdrop
10   });
11   return await popover.present();
```

Código 3.27: Fichero di-mi-nombre.page.ts, notifications

### 3.4.4. Resultado final

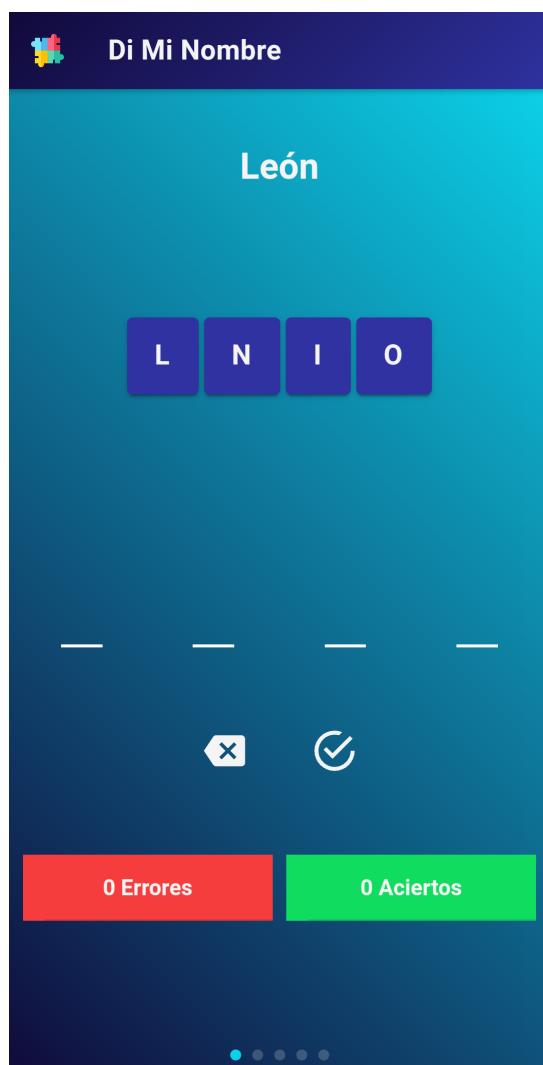


Figura 3.10: Di mi nombre

# CAPÍTULO 4

---

## Gestión de tablas

---

### 4.1– Página Crea tu tabla

#### 4.1.1. Mock Up

En este nuevo capítulo nos adentramos en la gestión de tablas, donde nuestro objetivo será **crear**, **modificar** o incluso **eliminar** las tablas.

A pesar de la sencillez de esta página, dispone de todas las funciones anteriores. Demos paso a su explicación correspondiente. Si observamos todos los elementos de esta propuesta de diseño, nos encontramos en la parte superior con la flecha nombrada en capítulos anteriores, para **volver** en este caso al **Menú Inicial**.

A continuación, observamos los botones de las temáticas registradas en nuestra base de datos, dándonos la opción de **pulsar** y adentrarnos en sus listas correspondientes, preparadas para ser modificadas o añadir nuevas parejas como comentaremos en el apartado siguiente 4.2.

A la derecha de los botones de cada temática nos encontraremos con otro botón de un tamaño menor con una papelera, esta nos dará la opción de **eliminar** la tabla completa de la temática situada a la izquierda de dicho botón.

Por último, el botón rojo con el más colocado en la parte inferior de la pantalla, da la opción de **añadir** una tabla nueva. Pulsando este botón, nos redirigirá al formulario de creación de la tabla.

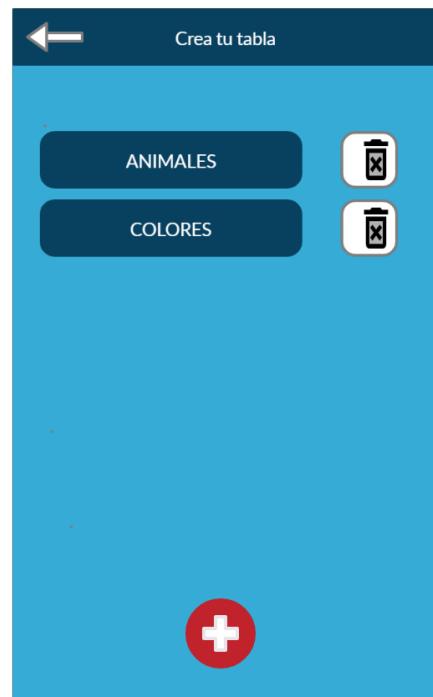


Figura 4.1: *Mockup Crea tu tabla*

### 4.1.2. Vista

La vista de la página `manage-tables` (Crea tu tabla) se basa casi totalmente en la vista de la página Elige Tabla, salvo por dos conceptos que comentaremos a continuación:

#### ion-grid

El *grid* o rejilla es un poderoso sistema de flexbox<sup>1</sup> móvil para la construcción de diseños personalizados.

Se compone de tres unidades un grid, filas y columnas. Las columnas se expandirán para llenar la fila y se redimensionarán para que quepan columnas adicionales. Se basa en una estructura de 12 columnas con diferentes puntos de interrupción según el tamaño de pantalla. El número de columnas se puede personalizar utilizando CSS.

Decidí usar este `ion-grid` en Crea tu tabla debido al control sobre la disposición de filas y columnas que nos aporta, permitiéndome así redistribuir el espacio dinámicamente mientras se mantiene la relación de aspecto que había diseñado en mi idea inicial. Por ejemplo, para conseguir que la columna concerniente a los botones con el icono de la papelera ocupase menos espacio que la de las tablas, añadí el atributo `size = '2'` a su etiqueta `ion-col` correspondiente. Aparte, `ion-grid` también nos ofrece muchas propiedades CSS personalizadas para usar directamente sobre las filas y las columnas que nos permitirán indicarles el posicionamiento que deseamos que tengan sus elementos tanto horizontal como verticalmente.

#### ion-fab

Los `ion-fab` son elementos tipo contenedor que contendrán uno o más botones fab. Deben colocarse en una posición fija que no se desplace con el contenido. En la página Crea tu tabla he añadido uno de estos botones fab, cuya simple función será la de lanzar una página sobrepuerta a esta que nos permitirá crear nuevas tablas. Aunque en nuestro caso no hayamos hecho uso de otras funcionalidades de los fab, también pueden contener listas de botones fab hijos que se mostrarán al hacer clic en el botón fab principal. La disposición de estos botones hijos es completamente configurable en las cuatro direcciones. En la documentación oficial de Ionic, en el apartado correspondiente a este tipo de componentes podréis encontrar ejemplos visuales que os aclararán mucho el funcionamiento de dichos botones.

---

<sup>1</sup>El Módulo de Caja Flexible, comúnmente llamado flexbox, fue diseñado como un modelo unidimensional de layout, y como un método que pueda ayudar a distribuir el espacio entre los ítems de una interfaz y mejorar las capacidades de alineación.

### 4.1.3. Código

Se podría afirmar completamente que Crea tu tabla tiene una función muy específica, y esta es controlar tres tipos de botones lanzadores de componentes. Uno (el de la papelera) se encargará de lanzar un `ion-alert` a modo de confirmación del borrado de las tablas. Otro (el `ion-fab`) se encargará de lanzar un `ion-modal` con la función de crear las tablas. Y por último, los botones correspondientes a cada una de las tablas existentes en la base de datos, lanzarán otro `ion-modal`, muy similar al anterior pero en este caso con el objetivo de modificar una tabla ya creada. Al ser estos dos últimos tan parecidos, tan solo comentaremos uno de ellos.

#### `ion-alert`

Una alerta es un cuadro de diálogo que presenta a los usuarios información o recopila información del usuario utilizando las entradas. Las alertas, aparecen sobre el contenido de la aplicación. Esta, debe ser eliminada manualmente por el usuario antes de que pueda reanudar la interacción con la aplicación (a diferencia de lo que ocurría con los `Toast`). También puede tener opcionalmente un título, un subtítulo y un mensaje.[1]

Para poder utilizarla, será necesario importar el controlador de alertas `AlertController` desde `@ionic/angular`, para después consumirlo en el constructor de la clase **ManageTablesPage**. Tras hacer esto, estaremos preparados para hacer uso de `ion-alert`. En lugar de explicaros como se usaría, voy a adjuntar una captura de la función que implementa dicho uso y que muestra la alerta en la Vista de la página incluyendo un botón para aceptar el borrado de las tablas y otro para cancelarlo:

```

1  async presentAlertConfirmDeletion(tableName: string) {
2    const tableNameUpperCase: string = tableName.toUpperCase();
3    const alert = await this.alertController.create({
4      header: 'Borrar Tabla ' + tableNameUpperCase,
5      message: 'Está seguro de que desea <strong>eliminar la tabla
6        definitivamente</strong>?',
7      buttons: [
8        {
9          text: 'Cancelar',
10         role: 'cancel',
11         cssClass: 'secondary',
12         handler: (blah) => {
13           console.log('Confirm Cancel: blah');
14         }
15       }, {
16         text: 'Aceptar',
17         handler: () => {
18           console.log('Borrado confirmado');
19           this.db.deleteTable(tableName);
20         }
21       ],
22       mode: 'ios'
23     });
24
25     await alert.present();

```

Código 4.1: Fichero `manage-tables.page.ts`, `presentAlertConfirmDeletion`

### ion-modal

Un Modal es un diálogo que aparece en **la parte superior del contenido** de la aplicación haciendo uso de una animación ascendente. Este, debe ser cerrado por la aplicación o el usuario antes de que la interacción pueda reanudarse. Es **útil** como componente de selección cuando **hay muchas opciones para elegir**, o cuando se filtran elementos en una lista, así como muchos otros casos de uso. En nuestro caso particular, se ha seleccionado para contener el formulario de creación y modificación de tablas para usar en los juegos, ya que el modal, se desliza como una persiana sobre la página Crea tu tabla, y esto nos permite que cuando un usuario cree una tabla al enviar el formulario y cerrarse el modal, dicho usuario vea como se actualiza la tabla que acaba de crear dinámicamente, aportando una sensación de profesionalidad y fluidez a nuestra. *app*.

```
1  async presentFormCreationModal() {  
2      const modal = await this.modalCtrl.create({  
3          component: FormCreationPage, // Usamos nuestro formulario como un routed  
4              entry component, definido en form-creation.module.ts  
5          mode: 'ios'  
6      });  
7      return await modal.present();  
}
```

Código 4.2: Fichero `manage-tables.page.ts`, `presentFormCreationModal`

En referencia al uso de los modals, solo nos quedaría añadir una cosa. Para poder usarlos, será necesario importar la clase del módulo de cada uno de ellos en el módulo raíz `app.module.ts`.

#### 4.1.4. Resultado final

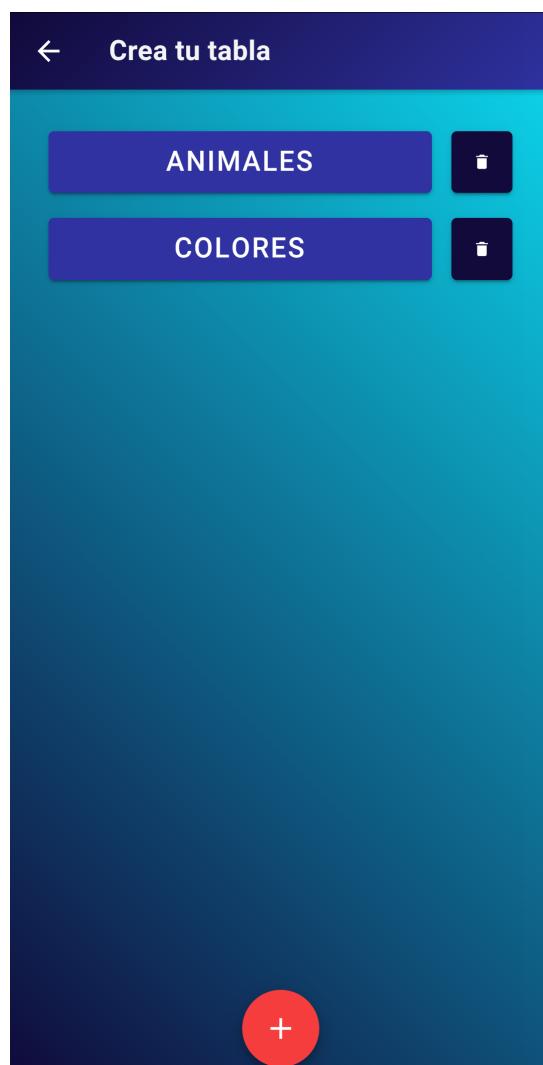


Figura 4.2: Crea tu tabla

## 4.2– Formularios de creación y modificación de tablas

### 4.2.1. Mock Up

Este apartado es de suma importancia para nuestro usuario, ya que toda pareja de palabras que inserte en este formulario, aparecerá en los juegos posteriormente.

En esta sección hay numerosos botones. En primer lugar encontramos el de **cerrar**, cuya función será la de cerrar esta página, sin guardar los cambios que se hayan realizado.

Por el contrario, y en la esquina superior derecha, tendremos a nuestra disposición **enviar** para registrar todos los cambios realizados o guardar la tabla nueva en nuestra base de datos.

A continuación, nos encontramos con **nombre de la tabla** donde colocaremos nuestro título, justo encima de la línea que nos aparecerá (donde tengamos que pulsar para activar el teclado y poder escribir).

En caso de querer **borrar** alguna pareja ya existente, disponemos de un botón circular con una cruz dentro, con el que podremos borrar dicha pareja con tan solo pulsar este botón. Sin embargo, si pulsamos el botón rojo inferior en el que pone **añadir pareja** aparecerán dos guiones nuevos situados respectivamente en cada columna (junto con el botón de la cruz), para poder incluir una pareja más a nuestro listado.

León	Lion	(X)
Tigre	Tiger	(X)
Perro	Dog	(X)
Gato	Cat	(X)
Ornitorrinco	Platypus	(X)
Pájaro	Bird	(X)
Delfín	Dolphin	(X)

Figura 4.3: Mockup Modifica tu tabla

### 4.2.2. Vista

Lo más importante, identificativo y novedoso de esta Vista es el uso de los formularios. La página Creación de Formularios se divide en dos partes, el **header** y el **content**:

#### Header

Aquí, encontraremos dos botones. A la izquierda, un botón para cerrar el formulario y cancelar toda progreso realizado con él. A la derecha un botón **submit** que enviará los datos del formulario a una función del controlador en formato JSON<sup>2</sup> para que posteriormente esta lo procese y haga las peticiones necesarias al servicio de base de datos para registrar los datos de la tabla añadida. Dicho botón **submit** tan solo estará habilitado en caso de que todos los controles del formulario hayan sido validados correctamente.

```

1 <ion-header>
2   <ion-toolbar color="primary">
3     <ion-buttons slot="start">
4       <ion-button size="small" shape="round" (click)="closeModal()" color=""
5         tertiary" strong="true">
6         <ion-icon slot="start" name="close"></ion-icon> Cerrar
7       </ion-button>
8     </ion-buttons>
9
10    <!-- Submit button del formulario, solo disponible en caso de no haber ningún
11        error -->
12    <ion-buttons slot="end">
13      <ion-button strong size="small" shape="round" color="tertiary" strong="true"
14        >
15        [disabled]="!validation_form.valid" (click)="onSubmit(validation_form.
16          value)">
17        Enviar <ion-icon slot="end" name="checkmark-circle"></ion-icon>
18      </ion-button>
19    </ion-buttons>
20  </ion-toolbar>
21</ion-header>
```

Código 4.3: Fichero `form-creation.page.html`, header

#### Content

En esta parte, lo primero que nos encontramos es la etiqueta `<form>`, seguida de `[FormGroup] = "validation_form"`. El valor anterior indica el nombre del grupo de controles de formulario declarado desde el controlador que se utilizará en este formulario. De esto hablaremos más a fondo en el apartado de código.

Después, si seguimos leyendo el código, encontraremos una etiqueta `ion-input`, esta se encargará de recoger y validar la entrada dada por el usuario para el nombre de la tabla que se va a crear. La validación se realiza dentro de una etiqueta `ng-container`, usando el estilo CSS dado por la clase `error-message`. Mediante un `*ngIf` comprobará si

<sup>2</sup>JSON (acrónimo de JavaScript Object Notation o notación de objeto de JavaScript) es un formato de texto sencillo para el intercambio de datos.

ya hemos escrito o pulsado, para no mostrar los errores de forma anticipada a la actuación del usuario.

Tras esto, lo primero que encontraremos será un bucle que recorra todos los controles de formulario encontrados en el grupo `validation_forms` previamente nombrado, usando una tubería `keyvalue` que nos permitirá usar el valor iterado para obtener las claves y valores de los controles. Como al entrar al formulario solo existirá uno, el nombre de la tabla, nos encargaremos de omitirlo mediante el `*ngIf (control.key != tableName)`.

Por último, solo me falta añadir el porqué del atributo `formGroupName` dentro de la etiqueta `<div>`. Como para las duplas de valores en español e inglés hemos usado un `FormGroup` anidado, es necesario usar dicho atributo para poder acceder a las claves y valores de dichos controles. En caso de no hacerlo, obtendríamos un error de acceso, al no saber el controlador de formularios que se trata de un grupo de control de formularios anidado.

```

1   <ion-row *ngFor="let control of validation_form.controls | keyvalue;">
2
3     <ion-col *ngIf="control.key != 'tableName'" size="5">
4       <ion-item>
5         <div formGroupName="{{control.key}}>
6           <ion-input type="text" formControlName="spanishName" placeHolder="
7             español"></ion-input>
8         </div>
9       </ion-item>
10      <div class="validation-errors">
11        <ng-container *ngFor="let validation of validation_messages.
12          duplaControlMessages">
13          <div class="error-message"
14            *ngIf="validation_form.get(control.key + '.spanishName').hasError(
15              validation.type) && (validation_form.get(control.key + '.
16              spanishName').dirty ||
17              validation_form.get(control.key + '.spanishName').touched)">
18            <ion-icon name="information-circle-outline"></ion-icon> {{{
19              validation.message }}}
20          </div>
21        </ng-container>
22      </div>
23    </ion-col>
24
25    <ion-col *ngIf="control.key != 'tableName'" size="5">
26      <ion-item>
27        <div formGroupName="{{control.key}}>
28          <ion-input type="text" formControlName="englishName" placeHolder="
29            inglés"></ion-input>
30        </div>
31      </ion-item>
32    </ion-col>
33  </ion-row>
34
```

Código 4.4: Fichero `form-creation.page.html`, content

### 4.2.3. Código

Lo primero que haremos para habilitar el control de los formularios será importar en el módulo del modal las siguiente clases:

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
```

Obviamente, también las importaremos dentro del decorador `@NgModule`. En concreto, `ReactiveFormsModule` nos permitirá habilitar el control de formulario desde la plantilla o *template*, esto es vital para que todo lo comentado en el apartado anterior funcione correctamente.

Ahora, iré comentando las distintas funciones implementadas en la clase `FormCreationPage`, para que quede mucho más claro cómo son creados los controles de formularios.

En primer lugar, encontraremos un array con los distintos mensajes a mostrar en función del tipo de error encontrado durante la validación:

```
1 validation_messages = { // Mensajes de validación para cada una de las
2   validaciones realizadas en cada control de formulario
3   'tableName': [
4     { type: 'required', message: 'Nombre de tabla requerido.' },
5     { type: 'minlength', message: 'Nombre de tabla debe tener al menos 3 cará-
6       cteres.' },
7     { type: 'maxlength', message: 'Nombre de tabla no puede tener más de 20 car-
8       ácteres.' },
9     { type: 'pattern', message: 'Nombre de tabla debe contener solo letras y
10       espacios' },
11     { type: 'validUsername', message: 'Your username has already been taken.' }
12   ],
13   'duplaControlMessages': [
14     { type: 'required', message: 'Nombre de tabla requerido.' },
15     { type: 'minlength', message: 'Nombre de tabla debe tener al menos 2 cará-
16       cteres.' },
17     { type: 'maxlength', message: 'Nombre de tabla no puede tener más de 15 car-
18       ácteres.' },
19     { type: 'pattern', message: 'Nombre de tabla debe contener solo letras y
20       espacios' },
21     { type: 'validUsername', message: 'Your username has already been taken.' }
```

Código 4.5: Fichero `form-creation.page.ts`, mensajes de validación

En segundo lugar, tenemos por supuesto al `ngOnInit` de la página, en el que se inicia ese primer control por defecto para el nombre de la tabla a crear junto a sus condiciones de error:

```
1 ngOnInit() {
2   // this.form-creation = this.sanitizer.bypassSecurityTrustStyle(this.value);
3   this.validation_form = this.formBuilder.group({
4     tableName: new FormControl('', Validators.compose([
5       Validators.maxLength(20),
6       Validators.minLength(3),
7       // Validators.pattern('^(?=.*[a-zA-Z])(?=.*[0-9])[a-zA-Z0-9]+$'),
8       Validators.pattern('[a-zA-ZñáéíóúÁÉÍÓÚüÜ ]*'), // Se admitirán nombres de
9       tabla que solo tengan letras y espacios
10      Validators.required
```

Código 4.6: Fichero `form-creation.page.ts`, `ngOnInit`

Posteriormente, veremos una función `closeModal` que llamará a un método `dismiss` del controlador de modals, esta se encargará de cerrar el modal cuando el usuario pulse el botón superior izquierdo situado en el header, o de cerrarlo automáticamente cuando se realice el `submit`.

Finalmente, podremos observar dos funciones encargadas de añadir y borrar (para la `x` que se encuentra a la derecha de cada par de controles en la vista del formulario) controles del grupo de control principal.

```

1  /**
2   * @description Incrementa el número de elementos de control y añade su dupla
3   * en español y en inglés.
4   */
5  addControl() {
6    this.validation_form.addControl('par' + (this.duplaCount + 1), new FormGroup
7    ({
8      spanishName: new FormControl('', Validators.compose([Validators.maxLength
9        (15), Validators.minLength(2), Validators.pattern('[a-zA-ZñáéíóúÁÉÍÓÚüÜ
10       ]*'), Validators.required])),
11      englishName: new FormControl('', Validators.compose([Validators.maxLength
12        (15), Validators.minLength(2), Validators.pattern('[a-zA-ZñáéíóúÁÉÍÓÚüÜ
13       ]*'), Validators.required]))
14    }));
15    console.log('Contador duplas: ' + this.duplaCount);
16
17    this.duplaCount++;
18    console.log('Contador duplas después: ' + this.duplaCount);
19
20  }
21 /**
22  * @description Elimina dupla de elementos de control español y en inglés.
23  * @param control Control a eliminar
24  */
25 removeControl(control) {
26   const controlKey: string = control.key;
27   console.log('Eliminar dupla de controles: ' + controlKey);
28   this.validation_form.removeControl(controlKey);
29 }
```

Código 4.7: Fichero `form-creation.page.ts`, añadir y borrar controles

Como guinda del pastel, encontraremos una última función llamada `onSubmit`, esta procesará los datos del `submit` del formulario, creando la tabla en caso de no existir, y añadiendo las duplas de elementos a dicha tabla.

Como nota informativa, el otro formulario, el de modificación de datos, es muy parecido a este en cuanto a funcionalidad, con la diferencia, de que se inyectaran tantos controles por defecto como filas tenga la tabla, aparte de ese primer control usado aquí para el nombre de la tabla. Otro cambio importante, es la forma en que `onSubmit` interaccionará con la base de datos, ya que en este caso no añadirá todas las duplas a la tabla, sino únicamente las que se hayan creado nuevas, las antiguas (identificadas en el código mediante una búsqueda en DB) serán modificadas.

#### 4.2.4. Resultado final

The screenshot shows a mobile application interface for creating a table. At the top, there is a dark blue header bar with a white 'X' icon labeled 'CERRAR' on the left and a 'ENVIAR' button with a checkmark icon on the right. Below the header is a light gray input field with the placeholder text 'Nombre de tabla'. At the bottom of the screen is a red rounded rectangular button with the text 'AÑADIR PAREJA' and a small white plus sign icon.

Figura 4.4: Formulario: Crear Tabla

X CERRAR ENVIAR ✓

Nombre de tabla  
Animales

León	Lion	X
Tigre	Tiger	X
Perro	Dog	X
Gato	Cat	X
Ornitorrinco	Platypus	X
Pájaro	Bird	X
Delfín	Dolphin	X

AÑADIR PAREJA +

Spanish Name	English Name	Action
León	Lion	X
Tigre	Tiger	X
Perro	Dog	X
Gato	Cat	X
Ornitorrinco	Platypus	X
Pájaro	Bird	X
Delfín	Dolphin	X

Figura 4.5: Formulario: Modificar Tabla

# CAPÍTULO 5

---

## Resultados

---

Antes de comenzar con este capítulo, me gustaría aclarar que no contemplaba esta sección en sus inicios, ya que implicaba hacerla en más tiempo. Como podréis observar, es una página sencilla que da los resultados de un juego en concreto **Di Mi Nombre**, ya que da opción a tener un registro de aciertos y errores, cosa que en **Une Palabras** resulta más complejo, debido a que los errores de este aparecen en rojo por décimas de segundo, dando la posibilidad de volver a escoger la palabra correcta y completar así el juego. Pero, como comenzábamos al principio de nuestro proyecto, todo aprendizaje requiere de una fase de evaluación, que permita ver el progreso que hemos tenido. Por tanto, tan solo expondremos las ideas principales de esta página junto con el resultado final de cómo nos quedaría en la aplicación, y dejaremos el desarrollo de la vista y el código de esta última parte como un aspecto de mejora de cara al futuro.

### 5.1– Mock Up

Este es uno de nuestros últimos recovecos, siendo esta página la que recoge todo el progreso realizado durante el juego, es decir los **resultados**. Como podemos ver en nuestro *mock up*, nos dará los resultados de una temática en concreto. Dicha temática será escogida en la página anterior a esta, para así evitar que se mezclen los contenidos y ver detalladamente cuántos errores y aciertos hemos obtenido de un grupo de palabras del mismo campo semántico. Para no equivocarnos de tabla escogida, en el encabezado, aparecerá **resultados más** el nombre de la tabla que hemos seleccionado.

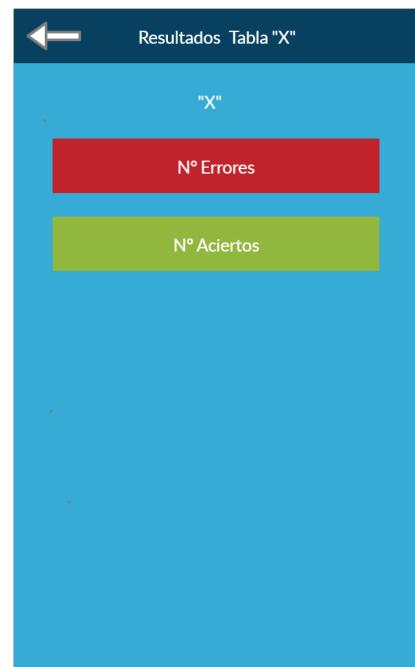


Figura 5.1: *Mock up* Resultados

## 5.2– Resultado final

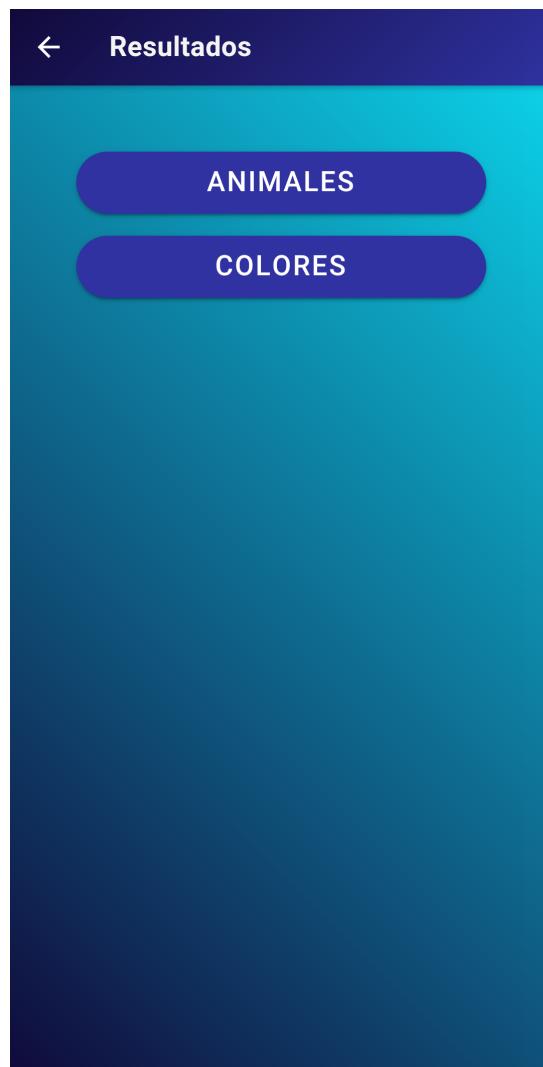


Figura 5.2: Resultados: Elegir Tabla

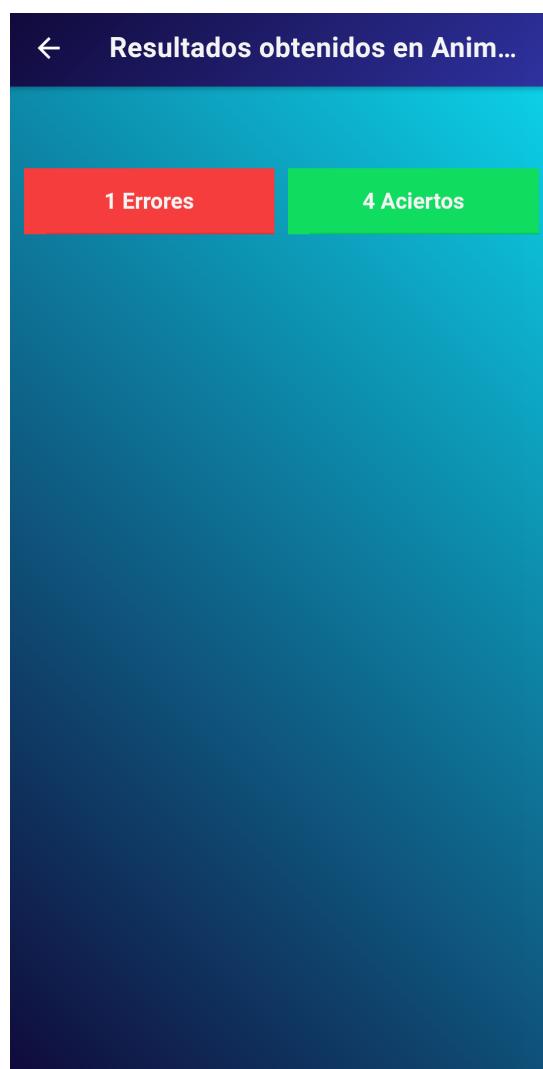


Figura 5.3: Resultados: Con valores Disponibles

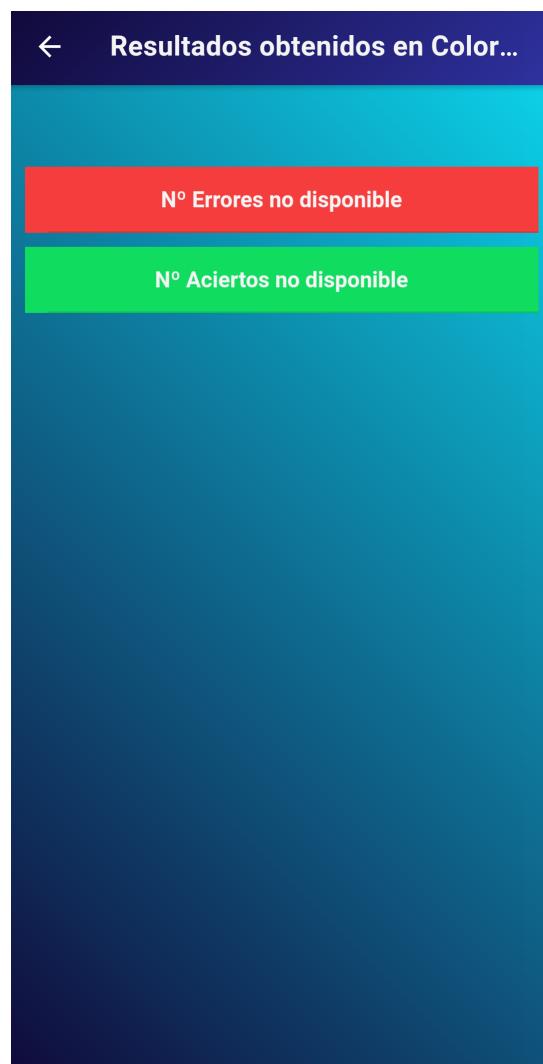


Figura 5.4: Resultados: Sin Valores Disponibles

# CAPÍTULO 6

---

## Manual de uso

---

SAY MY NAME



Figura 6.1: Say My Name

DESARROLLO DE APLICACIONES MÓVILES MULTI-PLATAFORMA: GAMIFICACIÓN DEL  
PROCESO DE APRENDIZAJE DEL IDIOMA

**Splash Screen**

Figura 6.2: Splash Screen

Página que da inicio a la aplicación. Aparecerá a modo de página de carga del contenido, previo al juego.

### Menú Principal

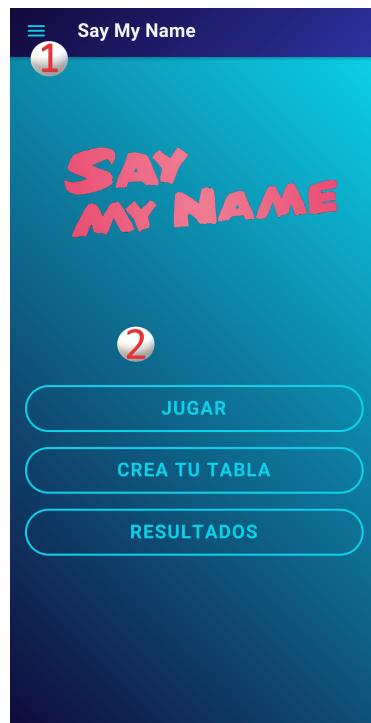


Figura 6.3: Menú de inicio

1. **Menú.** El símbolo celeste situado en la esquina superior izquierda, nos redirige a una ventana que se abre desde el lado izquierdo. Es un menú desplegable donde encontraremos los datos del usuario, en este caso los míos, junto con información sobre la aplicación.
2. **Botones de elección.** En esta parte inferior de la pantalla, nos encontramos con tres botones. El usuario podrá escoger el botón que deseé, las opciones son: jugar, crear su propia tabla o ver los resultados de la actividad realizada.

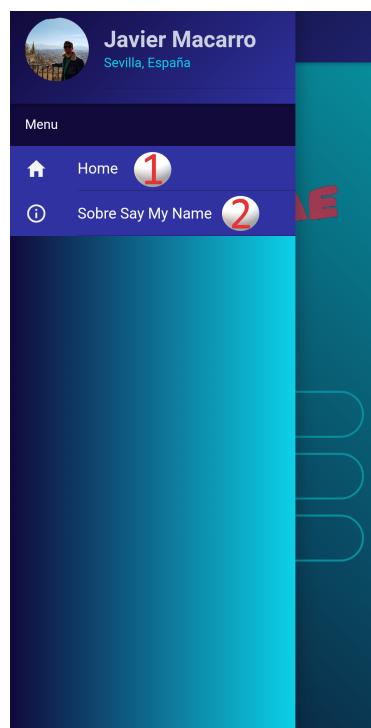
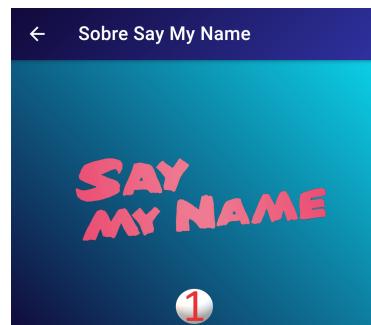


Figura 6.4: Menú desplegable

1. **Home.** Botón para volver al menú de inicio.
2. **Sobre Say My Name.** Pulsando en esta sección, aparecerá información acerca de nuestra aplicación.

**Descripción:**

Versión: 1.0 (Android)  
Aplicación híbrida multiplataforma desarrollada mediante el framework Ionic, a modo de juego sencillo e intuitivo, con el objetivo de agilizar el proceso de aprendizaje del idioma.  
Desarrollada por Javier Macarro Medina como TFG para la Universidad de Sevilla Dpto. Ciencias de la Computación e Inteligencia Artificial.  
Icons made by [Freepik](#) from [www.flaticon.com](#)



Figura 6.5: Información sobre la App

1. **Descripción.** Se exponen los principales datos sobre la aplicación. Además del nombre de la persona que lo ha realizado, junto a un link que indica el creador de los iconos usados en la *app* dado a que su licencia lo exige.

### Juego

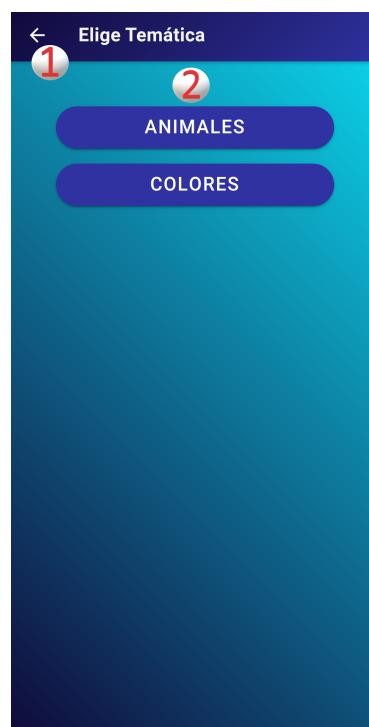


Figura 6.6: Página de elección de la temática

1. **Volver a la página anterior.** Con esta flecha, simbolizamos que puede regresar a la página previa a la actual.
2. **Temas.** Aparecerán en diferentes botones las dos temáticas por defecto (como en la imagen superior) junto al resto de temas que se creen posteriormente.

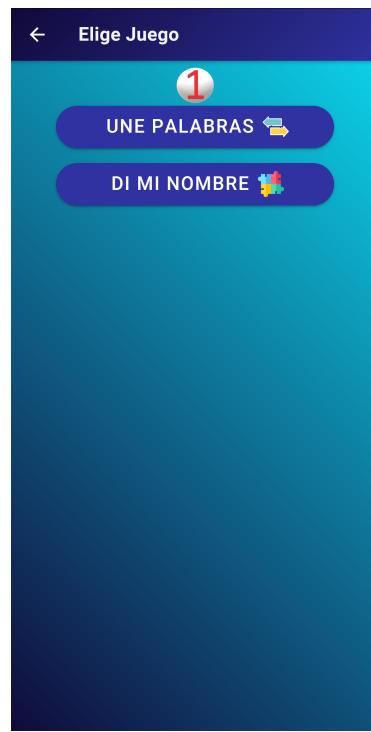


Figura 6.7: Página de elección del juego

1. **Juegos.** En esta página, el usuario escogerá el juego con el que llevará a cabo su aprendizaje. En este caso, disponemos de dos: **Une Palabras** y **Di Mi Nombre**.

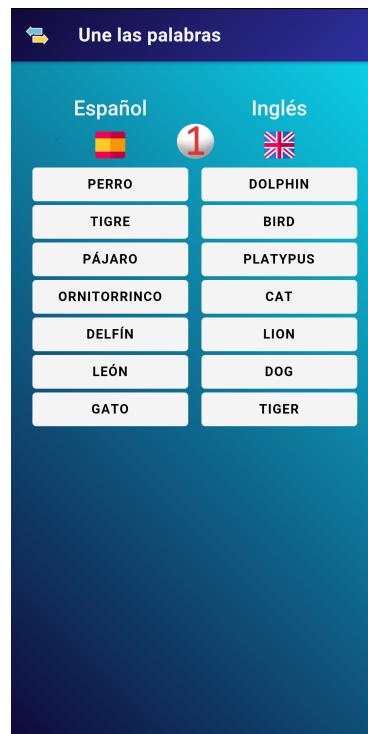


Figura 6.8: Juego Une Palabras

1. **Une Palabras.** En este juego, tendremos que seleccionar una de las palabras de la primera columna y relacionarla con otra de la segunda (o viceversa) pulsando sobre ella.



Figura 6.9: Juego Di Mi Nombre

1. **Letras desordenadas.** Conjunto de letras desordenadas que componen la palabra inglesa.
2. **Resolución.** Aparecerán tantos guiones como letras tenga la palabra inglesa.
3. **Borrar.** En caso de colocar mal una de las letras anteriores, podremos borrar las letras y recolocarlas.
4. **Verificar.** Una vez puesta la palabra completa, pulsaremos este botón para confirmar si es correcto el resultado, y pasar a la palabra siguiente.
5. **Registro de errores y aciertos.** Se recogerán todos los errores y aciertos de las palabras anteriores. Al finalizar, este será el registro que podamos observar en la página **resultados**.
6. **Guía del paginado.** En este juego tomaremos 5 palabras de la temática seleccionada. Por lo tanto, conforme vayamos avanzando, el puntito de color más claro, indicará cuántas páginas nos quedarán para llegar al final.

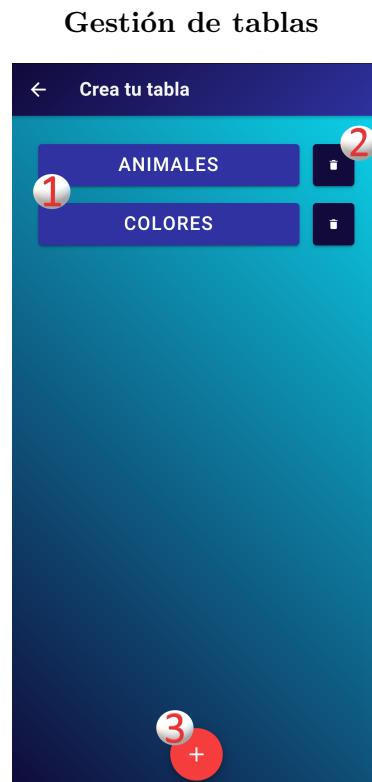


Figura 6.10: Menú de observación de las tablas realizadas

1. **Modificar.** Desde esta página, al elegir un botón de una temática cualquiera, te permite acceder a un formulario de modificación de los datos ya creados.
2. **Eliminar.** Pulsando este botón podremos borrar la tabla creada. Antes de eliminarla definitivamente, aparecerá una alerta de confirmación, que tendremos que aceptar o denegar.
3. **Añadir tabla.** Pulsando el botón **más** que aparece en la parte inferior, aparecerá un formulario que tendremos que llenar para obtener una tabla nueva.

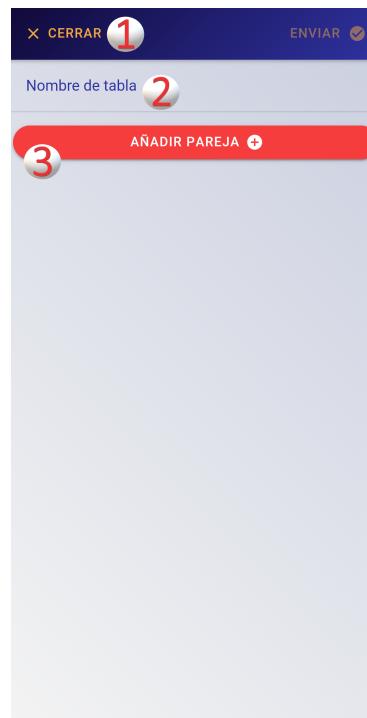


Figura 6.11: Creación de nueva tabla

1. **Cerrar.** En caso de no querer realizar una tabla nueva, pulsar este botón. Volverá a la página de **Crea tu tabla**.
2. **Nombre de la tabla.** Colocar el título de la tabla que queremos añadir, el juego permite colocar letras y espacios.
3. **Añadir pareja.** Con este botón podremos añadir tantas parejas como el usuario quiera registrar.

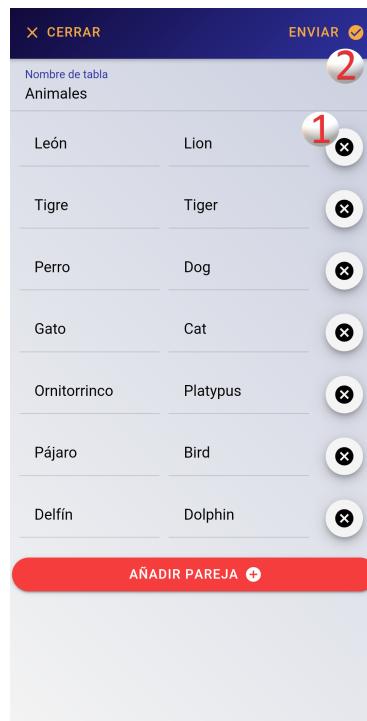


Figura 6.12: Crear o modificar tablas

1. **Borrar pareja.** Pulsando esta cruz, eliminaremos la pareja correlativa a esta, desapareciendo así estas dos palabras de nuestro listado.
2. **Enviar.** Una vez finalizada nuestra lista, para poder registrarla en nuestra base de datos y que quede guardada, tendremos que pulsar este botón.

## Resultados

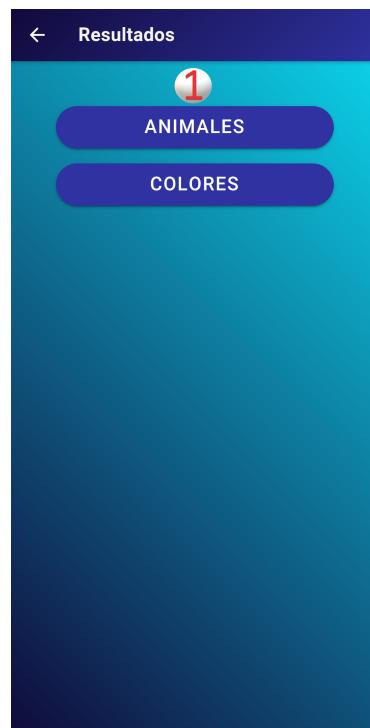


Figura 6.13: Página Resultados

1. **Elección de temas.** Una vez pulsado el botón **resultados**, tendremos acceso a esta página, donde tendremos que escoger la temática de los resultados que queremos ver.

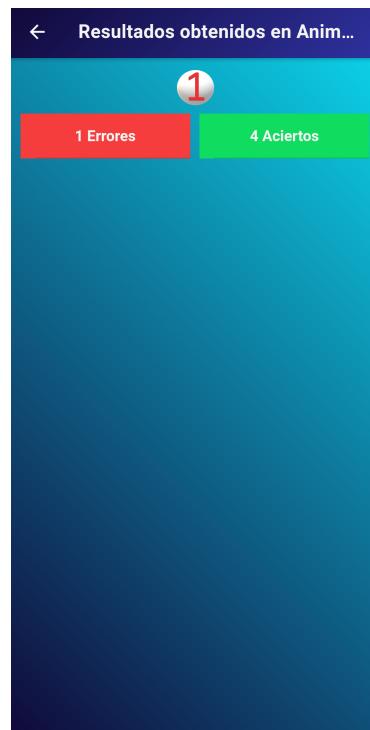


Figura 6.14: Visualización de resultados obtenidos

1. **Errores y Aciertos.** Una vez jugado al menos una vez, quedarán registrados los datos obtenidos. En caso de **no** haber jugado, aparecerán dos botones iguales a los que se muestran en la imagen superior pero con el texto **Nº de Aciertos/Errores no disponible**

# CAPÍTULO 7

---

## Conclusiones

---

Una vez terminada la aplicación y echando la vista atrás a todo este largo proceso, he podido comprobar la complejidad que conlleva la realización de una aplicación de tal envergadura (considerando que lo ha desarrollado una sola persona). Sobre todo, la preparación y documentación autodidáctica necesarias antes de comenzar siquiera a configurar el entorno de desarrollo.

Es necesario realizar una gran investigación para dar con los programas y utilidades adecuados, todo ello teniendo en cuenta las limitaciones de presupuesto que una empresa desarrolladora no tendría pudiendo utilizar *software* de pago que faciliten el trabajo. Como conclusión, puedo afirmar que el descubrir Ionic y todo el mundo que lo rodea (Angular, Cordova, Node.js, npm, etc...) ha sido un proceso muy gratificante para mi. Pienso que es una tecnología que, pese a ya ser una de las más punteras del mercado en cuanto a desarrollo de aplicaciones multiplataforma híbridas, lo será aún más, debido al gran soporte que tiene por parte de la comunidad y del equipo de Ionic: Una extensa y clara documentación, actualizaciones frecuentes del *framework* para adaptarse a las novedades y funcionalidades incorporadas por los SO nativos y una cuantiosa variedad de apoyo por parte de *software* de terceros, plantillas temas preestablecidos, etc...

Como propuestas de futuro para mejorar nuestra aplicación, se me ocurren varias ideas: En primer lugar, es obvio que para una puesta en mercado esta aplicación necesitaría incluir más contenido de base para resultar una apuesta atractiva en el competitivo mundo de las aplicaciones móviles, por lo que un cambio positivo para la aplicación de cara al usuario final sería la ampliación del número de juegos y tablas predefinidas. A su vez, como complemento a esta mejora, se me ocurre que las tablas usadas en los juegos podrían implementarse ordenadas por temáticas y que los juegos podrían aumentar su complejidad usando utilidades de terceros como el *framework* PHASER.IO[24] que nos permitirían darle mucha más atractivo a los juegos de cara a un público infantil gracias a sus librerías gráficas.

En definitiva, considero que la formación obtenida gracias a este proyecto, podría abrirme muchas puertas, sobre todo de cara a puestos de trabajo en los que se requiera dominio en las tecnologías de *front-end*, y que a nivel profesional me ha enriquecido, pudiendo ampliar así mi currículum y mis habilidades en este mundillo tecnológico.



---

## Bibliografía

---

- [1] Ionic Official WebSite Contributors. *Ionic Developer Manual*. Ionic WebSite 2019. [<https://ionicframework.com/docs>]
- [2] Ionic Official WebSite Contributors. *Electron Desktop App*. Ionic WebSite 2019. [<https://ionicframework.com/docs/publishing/desktop-app>]
- [3] Ionic Official WebSite Contributors. *About Ionic*. Ionic WebSite 2019. [<https://ionicframework.com/about>]
- [4] Ionic Official WebSite Contributors. *Environment Setup*. Ionic WebSite 2019. [<https://ionicframework.com/docs/installation/environment>]
- [5] Ionic Official WebSite Contributors. *iOS Setup*. Ionic WebSite 2019. [<https://ionicframework.com/docs/installation/ios>]
- [6] Ionic Official WebSite Contributors. *iOS Development*. Ionic WebSite 2019. [<https://ionicframework.com/docs/building/ios>]
- [7] David Dal Busco. *Using Ionic without any framework*. Medium, Oct 19 2018. [<https://medium.com/@david.dalbusco/using-ionic-without-any-frameworks-775dc757e5e8>]
- [8] StackOverflow. *StackOverflow troubleshooting*. [<https://stackoverflow.com>]
- [9] Wikipedia. *Wikipedia info*. [<https://en.wikipedia.org>]
- [10] Hady ElHady. *Guide to Cross-Platform Mobile App Development Tools*. [<https://instabug.com/blog/cross-platform-development/>]
- [11] Paul Halliday. *Ionic 4: Angular, Vue.js and React*. [<https://dev.to/paulhalliday/ionic-4-angular-vue-js-and-react-1o14>]
- [12] campusMVP. *Las 10 principales diferencias entre AngularJS y Angular*. CampusMVP, 10 de mayo de 2017. [<https://www.campusmvp.es/recursos/post/las-10-principales-diferencias-entre-angularjs-y-angular.aspx>]
- [13] Carlos Azaustre. *Aprende ECMAScript 6 (ES6 o ES2015), el nuevo estándar de JavaScript*. [<https://carlosazaustre.es/ecmascript6/>]
- [14] Michael Abernethy. *¿Simplemente qué es Node.js?* [<https://www.ibm.com/developerworks/ssaopensource/library/os-nodejs/index.html>]

- [15] Alexander Guevara Benites. *¿Qué es npm?* [<https://devcode.la/blog/que-es-npm/>]
- [16] Cordova Developers. *Cordova official Documentation*. Cordova Official Website, 2019. [<https://cordova.apache.org/docs/en/latest/>]
- [17] Cordova Developers. *Android Platform Guide*. Cordova Official Website, 2019. [<https://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html>]
- [18] Cordova Developers. *iOS Platform Guide*. Cordova Official Website, 2019. [<https://cordova.apache.org/docs/en/latest/>]
- [19] Google and Angular Developers. *Angular Official Docs*. Angular Official Website, 2019. [<https://angular.io/docs>]
- [20] 2PaeDev. *Configura VSCode para Angular 7 Ionic*. Medium, Dec 9 2018. [<https://medium.com/2paedev/configura-vscode-para-angular-7-ionic-abc3a68e83c5>]
- [21] Luis Rene Mas Mas. *Como agregar variables de entorno (S.O. Windows 10)*. Medium, Sep 3, 2016. [<https://medium.com/@01luisrene/como-agregar-variables-de-entorno-s-o-windows-10-e7f38851f11f>]
- [22] Android Developers. *Cómo configurar las opciones para desarrolladores en el dispositivo*. Developers Android, 2019. [<https://developer.android.com/studio/debug/dev-options>]
- [23] Ionic Theme Developers. *Ionic 4 Start Theme*. [<https://market.ionicframework.com/themes/ionic4-start-theme>]
- [24] Phaser Developers. *Phaser: Desktop and Mobile HTML5 Game Framework*. [<https://phaser.io>]
- [25] tobiasahlin. *Spinkit*. [<https://tobiasahlin.com/spinkit/>]
- [26] Simon Grim. *How to Build an Ionic 4 App with SQLite Database and Queries (And Debug It!)*. [<https://devdactic.com/ionic-4-sqlite-queries/>]
- [27] Miguel Angel Alvarez. *Introducción teórica a los observables en Angular*. [<https://desarrolloweb.com/articulos/introduccion-teorica-observables-angular.html>]
- [28] RXJS Developers. *BehaviorSubject*. [<https://www.learnrxjs.io/subjects/behaviorsubject.html>]
- [29] Josh Morony. *Using Angular Routing with Ionic 4*. [<https://www.joshmorony.com/using-angular-routing-with-ionic-4/>]