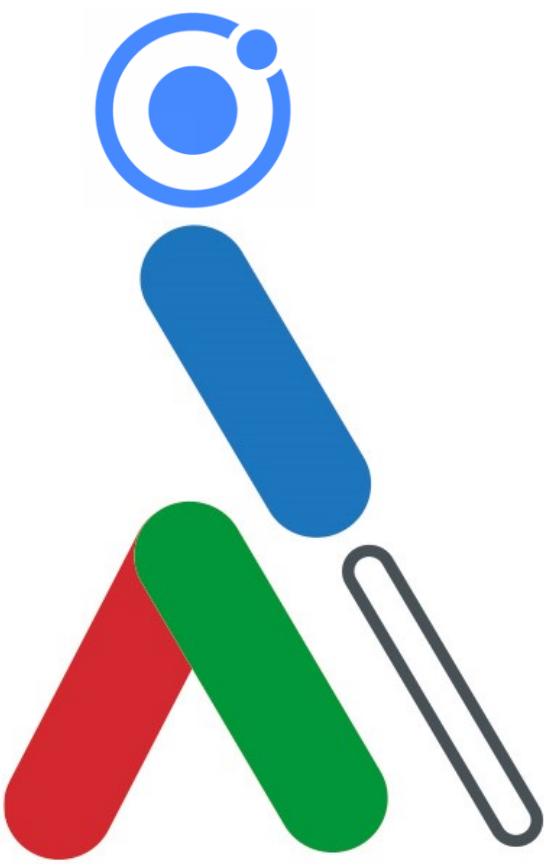

IONIC

Creación de una App

DESCRIPCIÓN BREVE

Manual sobre como crear una aplicación utilizando el Framework Ionic desde cero. En el manual se explica paso a paso desde la instalación de Ionic en el equipo hasta la creación de la aplicación para iOS y/o Android.

Tabla de contenido

Capítulo I: Configurando el entorno para Ionic

1. Introducción

El framework Ionic es un framework multiplataforma para crear aplicaciones para dispositivos móviles, ya sea en Android o iOS. Ionic permite la creación de estas aplicaciones mediante

tecnología web; por lo que si sabes crear una página web ya sabes prácticamente utilizar Ionic, sólo tienes que saber dónde va cada cosa. Pues en Ionic cada componente tiene su archivo específico.

Hoy vamos a ver cómo podemos configurar nuestro equipo para utilizar Ionic y que aplicaciones necesitaremos tener instaladas previamente. Lo bueno de Ionices que las aplicaciones se crean fundamentalmente a través de la línea de comandos, lo que nos permite avanzar y escalar con rapidez a la hora de crear una aplicación; pues en 5 minutos podemos crear un esqueleto sobre el que ir construyendo y edificando nuestra aplicación a medida que vamos añadiendo piezas.

Básicamente vamos a utilizar la Línea de Comando de Ionic, CLI, por sus siglas en inglés, para crear el esqueleto de nuestra aplicación y sus módulos y componentes. Por tanto, debemos instalar las últimas versiones de CLI y Cordova en nuestro equipo. Para poder realizar esto hemos de instalar antes la última versión de Node.js. Una vez tenemos Node.js y npm instaladas el resto de las instalaciones utilizando el terminal del equipo, ya sea Windows, Mac o Linux.

Con estas aplicaciones ya instaladas sólo nos queda abrir un terminal y teclear el siguiente comando para iniciar la instalación de Ionic y Cordova:

$ npm install –g ionic cordova[1]

Si estamos utilizando un terminal en Linux o iOS es posible que sea necesario escribir sudo antes de npm y después la contraseña del usuario administrador.

2.. Creando nuestra primera aplicación

Ahora que ya tenemos el entorno de Ionic instalado en el equipo estamos en disposición de crear nuestra primera aplicación. Para ello iremos a la línea de comandos y nos situaremos en la carpeta donde queremos crear nuestra aplicación. Para tener todas las aplicaciones Ionic en un mismo lugar se aconseja crear una carpeta, por ejemplo, Ionic mediante el comando medir:

$ mkdir Ionic

Tras la creación de la carpeta nos situaremos dentro de la misma utilizando el comando cd:

$ cd Ionic

Ahora dentro de esta carpeta crearemos nuestra primera aplicación utilizando el siguiente comando:

$ ionic start primeraApp

Al pulsar intro se nos permitirá elegir entre utilizar Angular, React y Vue. En nuestro caso utilizaremos Angular, que es la opción preseleccionada, por lo que pulsaremos intro. Esta parte es inaccesible con JAWS ja que sólo nos lee la última línea: vue.org.

Si queremos evitar que aparezca esta lista de opciones, al crear la aplicación podemos indicar a Ionic que framework queremos utilizar. Para ello, si queremos utilizar Angular escribiremos:

$ ionic start --type=angular

A continuación, se nos permitirá elegir la plantilla de la aplicación, Para hacer esta selección hemos de tener muy claro qué queremos crear y cómo queremos que sea nuestra aplicación, ya que la elección de una buena plantilla de inicio nos facilitará mucho el trabajo de desarrollo. Las opciones disponibles son:

[1] $ es el símbolo del prompt y –g significa que se escriba la variable global en el sistema.

- **Tabs**. Crea una interfaz sencilla con pestañas.
- **Sidemenu**. Crea una interfaz con un menú lateral con navegación en el área central.
- **Blank**. Crea una aplicación vacía.
- **My-first-app**. Crea una aplicación de ejemplo que consiste en una cámara con una galería.
- **Conference**. Crea una aplicación multiusos donde se muestra el potencial de lo que se puede crear utilizando Ionic.

Esta selección también es inaccesible por JAWS. JAWS lee todas las opciones cuando se carga la pantalla, pero luedo no dice que opción estamos seleccionando, a no ser que se seleccione la última opción. Por lo tanto, deberemos saber el orden de la opciones para ejecutar la que queremos.

Existe una forma de evitar que aparezca esta selección i es diciendo por línea de comando que plantilla quiere utilizarse al crear la aplicación. Por ejemplo, si queremos utilizar la plantilla de pestaña escribiremos el siguiente comando para la creación del proyecto.

$ ionic start –type=angular tabs

Con este comando estamos diciendo que queremos crear un proyecto utilizando el framework Angular y tabs como plantilla.

A continuación, se muestran las diferentes plantillas en un iPhone y un Android.

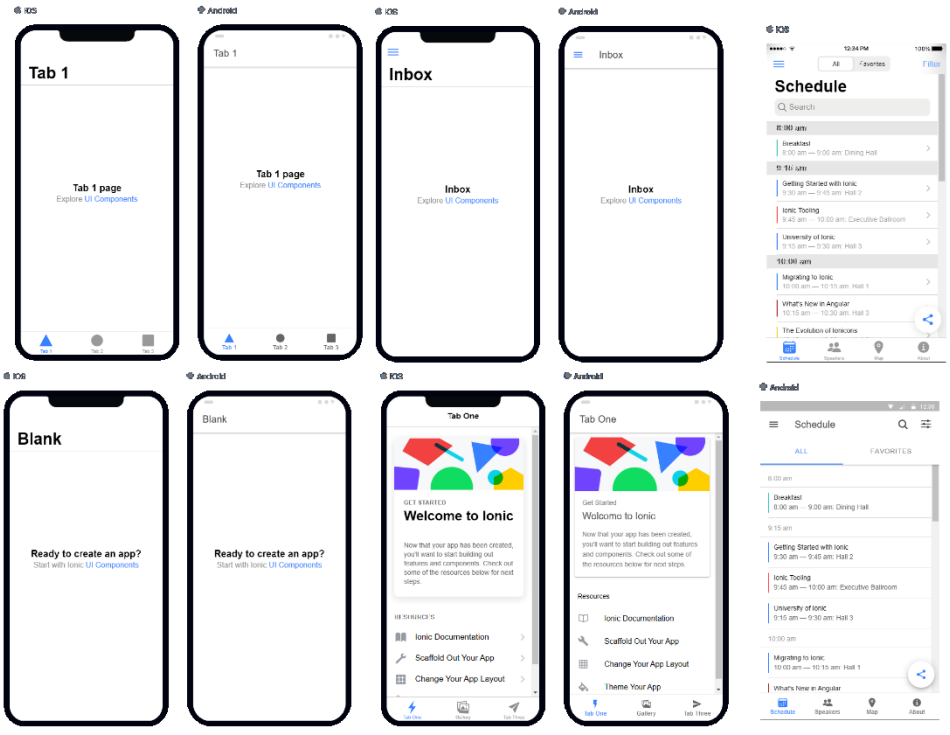

Figura 1. Plantillas de Ionic para iOS y Android.

Al finalizar la creación de la aplicación Ionic nos preguntará si queremos crear una cuenta gratuita de Ionic. Aquí responderemos y o n dependiendo de si la queremos crear o no. Se debe indicar que, si se quiere probar la aplicación en un dispositivo móvil sin crear el instalador, deberemos crearnos una cuenta en Ionic.

Tras pulsar intro se empezará a generar un nuevo proyecto para la aplicación, esto suele tardar unos minutos, tras los cuales se nos habrá creado una carpeta con el nombre primeraApp.

Lo primero que deberemos hacer es entrar en la carpeta mediante el comando:

$ cd primeraApp

Y después lanzaremos la aplicación con el comando:

$ ionic serve

Al pulsar intro se creará una aplicación y se abrirá nuestro navegador de internet por defecto para mostrar la aplicación en tiempo de ejecución. Mientras tengamos ese terminal abierto podremos ver la aplicación en tiempo de ejecución; por lo que para poder seguir creando la aplicación será necesario abrir otro terminal donde poder ir introduciendo los comandos para crear los objetos nuevos que queremos en la aplicación. Si queremos finalizar y cerrar la vista de la aplicación pulsaremos la combinación de teclas Ctrl + C o cerraremos el servidor del terminal.

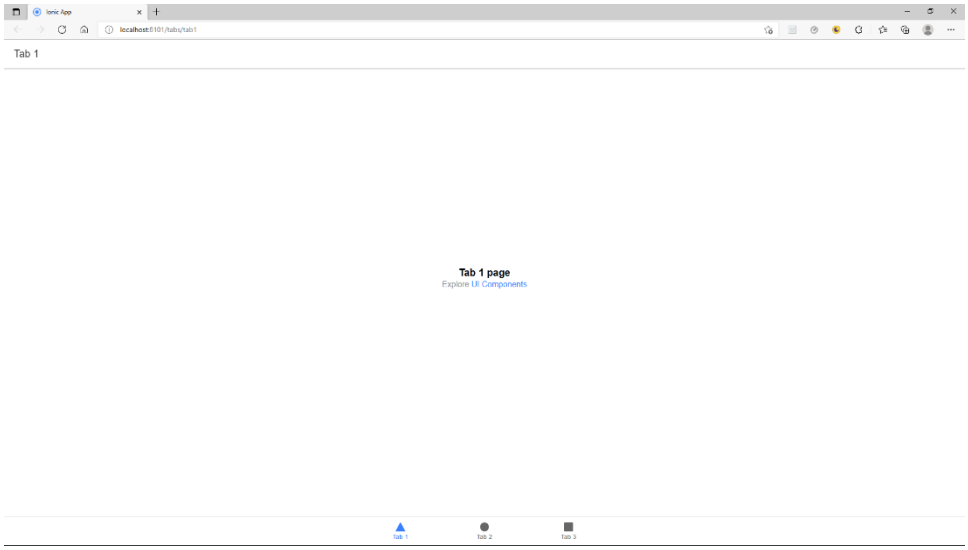

Figura 2. Vista del proyecto en una página web.

En caso de que no se nos abra el navegador de internet de forma automática, podemos abrir el navegador que más nos guste y en la barra de direcciones (Alt + D) escribir localhost:8100, que es donde está apuntando el servidor.

Si queremos ver como quedaría visualmente la aplicación en un terminal iOS y otro Android, deberemos utilizar el siguiente comando en lugar del anterior:

$ ionic serve –l

Si es la primera vez que utilizamos este comando en el proyecto, se nos pedirá que instalemos el pluging lab de Ionic. Cuando se nos muestre la opción deberemos pulsar y después intro para que se inicie la instalación. Una vez finalizada la instalación se abrirá el navegador con un dispositivo iOS y otro Android.

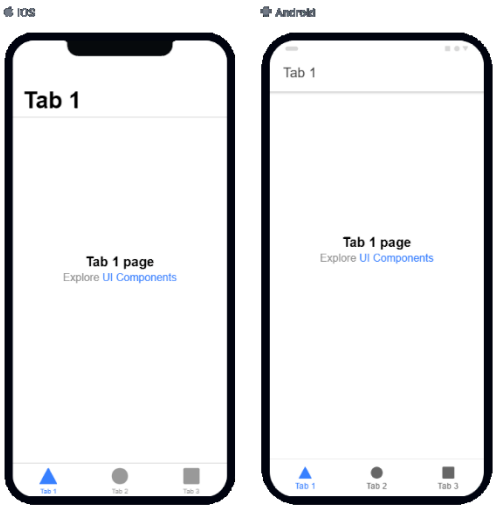

Figura 3. Vista del proyecto en iOS y Android.

Capítulo II: Creando un proyecto

1. Creando la aplicación

Para crear la aplicación, una vez realizada la instalación de Ionic y su entorno, abriremos la línea de comandos, o terminal, y ejecutaremos el siguiente comando:

$ ionic start numeros –type=angular tabs

Tras la creación del proyecto accederemos a la carpeta del proyecto: primeraApp mediante el comando:

$ cd numeros

Una vez dentro de la carpeta se comprobará si el proyecto se ha creado de forma correcta utilizando uno de los siguientes comandos:

$ ionic serve

$ ionic serve -l

Comprobamos que el proyecto se ha creado de forma correcta e iniciamos las modificaciones para crear la aplicación que deseamos, que en esta ocasión es crear en la primera pestaña un juego donde se debe adivinar un número creado de forma aleatoria.

2. Cambiando el Favicon

Lo primero que vamos a hacer es modificar el Favicon. El Favicon es el icono de la pestaña del navegador; por lo que vamos a crear uno para nuestra aplicación. Este icono debe de tener un logo o un tema que haga que el usuario sepa sobre qué consiste la aplicación que se ejecutará.

En caso de que no sepamos que icono utilizar o no tengamos ninguno en nuestro equipo, podemos utilizar un generador de favicon utilizando favicon generator, por ejemplo, o utilizar un buscador para encontrar otros. En caso de que no tengamos ninguna idea, podemos descargarlos desde internet utilizando, por ejemplo, icons8, o utilizar un buscador para encontrar otras webs para descargar alguno de forma gratuita.

Una vez tenemos el icono que queremos, vamos a modificarlo. Para ello tendremos que nombrar a nuestro icono como favicon.png y remplazar el de Ionic por el nuestro. El favicon de Ionic puede encontrarse en la siguiente ruta: /primeraApp/src/assets/icon. Simplemente tenemos que sobreescribir el archivo fanicon.png existente.

Si ahora vamos al navegador donde hemos ejecutado nuestro proyecto, veremos como el icono se modifica automáticamente. Para poder ver la modificación del favicon hemos de cargar el proyecto como si fuera una web (*ionic serve*).

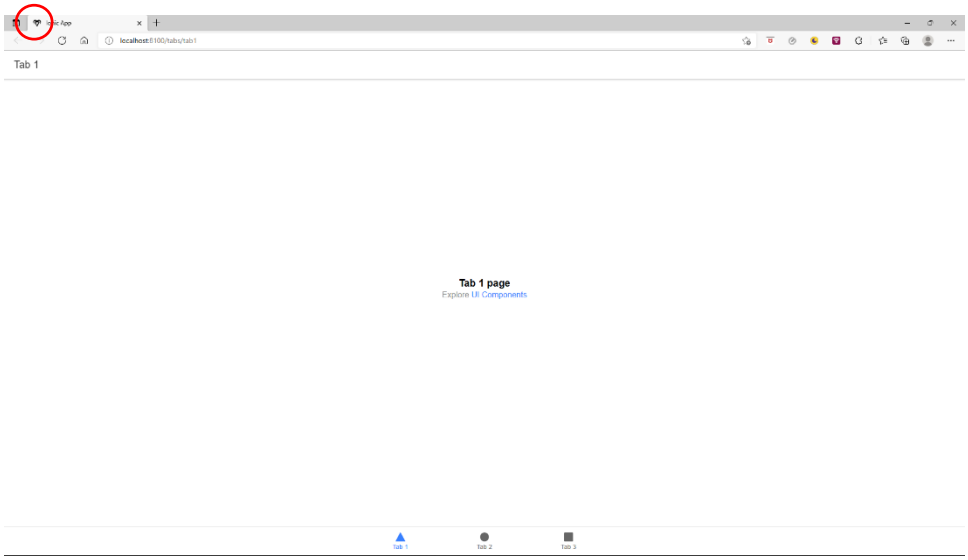

Figura 4. Modificación del Favicon.

¡Genial! Ya hemos modificado el icono de nuestra futura aplicación. Lo siguiente que vamos a hacer es la modificación del título de la página, donde pondremos un nombre apropiado; en este caso el nombre del juego: Adivina.

3. Cambiado el icono de la aplicación

Para cambiar el icono de la aplicación, es decir, el que aparecerá en el dispositivo móvil después de la instalación de la aplicación debemos crear un icono de 1024x1024px con el nombre de icono y otro con de 2732x2732px con el nombre de splash. En ambos casos se utilizará la terminación .png.

Una vez creado estos iconos se pegarán en la siguiente ruta: \primeraApp\resources sustituyendo a los dos que ha generado Ionic por defecto. Tras hacer esto escribiremos el siguiente comando en el Terminal:

ionic cordova resources

Tras ejecutar el comando, se nos crearán tres nuevas carpetas dentro de resources: Android, iOS y XML que contendrán los iconos necesarios para la aplicación. Ahora sólo nos queda asignar un nombre apropiado a la aplicación, ya que si empaquetamos la aplicación como está ahora mismo se creará con el nombre myApp.

¡Vamos a ello!

Es posible que antes tengamos que instalar *cordo-res*. Para ello ejecutaremos primero el comando:

npm install –g cordova-res

4. Modificando el nombre de la aplicación

Para modificar el nombre que queremos que aparezca cuando se instale la aplicación en un dispositivo, se debe modificar el archivo config.xml que encontraremos dentro de la carpeta raíz del proyecto: primeraApp. Dentro de esta carpeta debemos modificar el texto entre las etiquetas <name>, donde escribiremos el nombre que queremos dar a la aplicación.

En este archivo podemos realizar algunas otras modificaciones interesantes, aunque si no sabemos que estamos modificando, mejor no hacerlo. Por el momento, podemos modificar el texto de las siguientes etiquetas:

- **<description>**. Escribiremos una breve descripción de la aplicación.
- **<author>**. Escribiremos una dirección de correo de contacto, una página web de referencia y el nombre del autor.

Así pues, en la etiqueta descripción escribiremos un breve texto con la descripción de nuestra aplicación y nuestro nombre. El siguiente paso será la modificación del Título de la aplicación.

5. Modificando el Título

Para modificar el título que aparece en la primera pestaña de la página deberemos realizar una pequeña modificación en el archivo *tab1.page.html* que encontraremos en la siguiente ruta: primeraApp/src/app/tab1 y modificaremos la palabra Tab1 escrita dentro de la etiqueta **<ion-title>** y pondremos el siguiente texto: Adivina el número Secreto.

Si vamos al navegador veremos que ahora donde antes aparecía la palabra Tab1 como encabezado de la vista, ahora aparecerá Adivina el número.

Si queremos que el título en el iPhone sea diferente, modificaremos el contenido de la <ion-title> dentro de **<ion-toolbar>**. En este caso el texto que pondremos es Adivina.

Figura 5. Dispositivos móviles con el título modificado.

Aún tenemos cosas que modificar, ya que en las pestañas inferiores aún tenemos los nombres de la plantilla, por lo que debemos, al menos, modificar el nombre de la primera pestaña. ¡Vamos a ello!

6. Modificando el nombre de la pestaña inferior

El nombre de la pestaña sigue siendo Tab1. Este nombre no nos ofrece ninguna información sobre el contenido de la pestaña, con lo que lo modificaremos escribiendo Adivina en su lugar o directamente lo borraremos. Para ello hemos de modificar el archivo tabs.page.html que encontraremos en **primeraApp/src/aplicación/tabs**. En la etiqueta **<ion-label>** modificaremos Tab 1 por Adivina. Al salvar el archivo veremos que ahora el nombre de la pestaña es Adivina, tal y como queríamos. el texto en la etiqueta **<ion-title>** dentro de la etiqueta **<ion-toolbar>**

Figura 6. Dispositivos móviles con la primera pestaña modificada.

Esto ya va tomando forma, pero aún se puede ver el texto Ionic App en la pestaña superior del navegador, vamos a cambiarlo.

7. Cambiando el Nombre de la Pestaña Superior

Para modificar el nombre de la pestaña deberemos modificar el archivo index, este archivo se encuentra en la siguiente ruta: /primeraApp/src. Abriremos el archivo y modificaremos el texto Ionic App que encontraremos dentro de la etiqueta **<title>** por Primera App.

Si guardamos y vamos al navegador veremos que ahora se nos muestra Primera App en lugar de Ionic App.

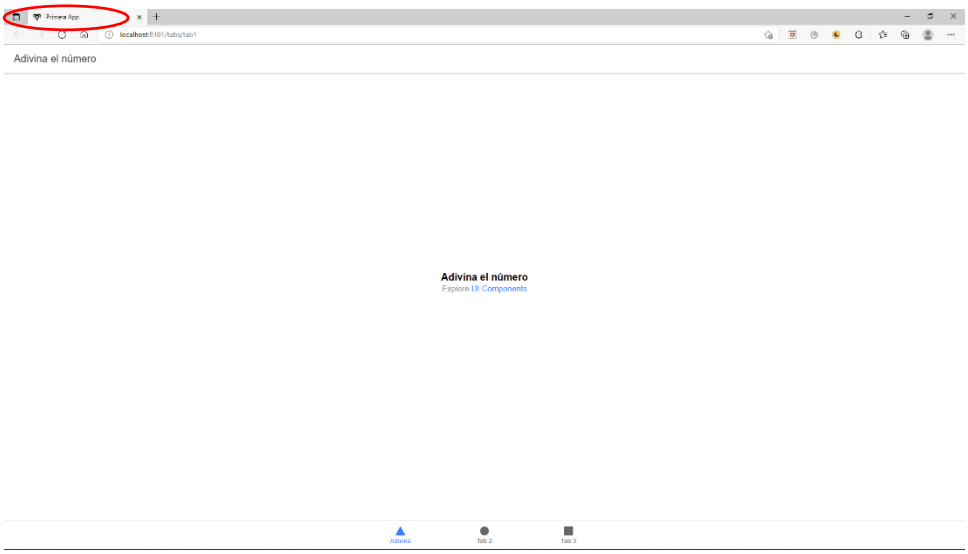

Figura 7. Modificación del Título de la Pestaña.

Ya que estamos aquí, vamos a hacer otra modificación a la página, en este caso va a ser el idioma.

8. Modificando el Idioma

Los usuarios de lector de pantalla habrán visto que al leer la página éste nos la lee en inglés, pero estamos haciendo, o queremos hacer una aplicación en castellano, por lo que modificaremos el texto en insertado en la etiqueta HTML por **es-es**. Y guardaremos.

Esta acción no tendrá ningún efecto visual, pero acabamos de decir que la aplicación está en castellano, por lo que los lectores de pantalla leerán de forma correcta la aplicación y podremos utilizar caracteres propios del castellano como los acentos y la ñ y el lector de pantalla no nos cambiará de idioma si tenemos activada la detección de idiomas automática.

Hasta el momento lo único que hemos hecho ha sido modificar el código existente en el proyecto. Ahora empezaremos a implementar elementos.

Lo que vamos a hacer es crear un juego para la primera pestaña de nuestra aplicación. Esto lo vamos a ver en el siguiente capítulo

Capítulo III: Juego Adivina el Número Secreto

1. Introducción

En esta primera pestaña vamos a crear un juego en el que tendremos que adivinar el número secreto que nos propone el móvil. Crearemos el juego de forma sencilla paso a paso y luego jugaremos un rato. ¡Vamos a ello!

2. Modificando Adivina (Tab 1)

Como el juego se presentará en la primera pestaña, deberemos modificar el código existente en la primera pestaña, así que lo primero que debemos hacer es abrir el fichero tab1.page.html para modificar su código allí donde sea necesario.

Todo lo que está entre las etiquetas **<ion-content>** es el contenido de la pestaña seleccionada, así que podemos borrar todo lo que hay entre esas etiquetas. El código del archivo debería ser el siguiente:

```
<ion-header [translucent]="true">
 <ion-toolbar>
   <ion-title>
    Adivina el número
   </ion-title>
 </ion-toolbar>
</ion-header>

<ion-content [fullscreen]="true">

</ion-content>
```

Si guardamos y vamos al navegador para ver qué pinta tiene nuestra aplicación, veremos que se ha borrado todo el contenido de la pestaña a excepción del título (header). Ahora vamos a rellenar con contenido la pestaña. Lo primero que haremos será crear una etiqueta ** ** Éstas etiquetas son como cajas de contenido y agrupan en su interior otras cajas; por lo que podríamos decir que son un gran baúl donde poner cajas más pequeñas. A esta etiqueta le asociaremos una clase (class).

Las clases permiten que las cosas se muestren de formas diferentes. Ya sea color, formato, sangrado, …

Dentro de las etiquetas ** ** escribiremos una etiqueta input. Estas etiquetas permiten la entrada de texto, en este caso un número, del usuario a la aplicación. Tras la etiqueta input escribiremos una etiqueta de párrafo <p> donde se nos mostrará si el número introducido es superior o inferior al número escogido por el juego. Por último, escribiremos una etiqueta botón (bottom) para decirle al juego que número hemos introducido y que nos dé una respuesta. Así pues, el código resultante sería:

```
<ion-header [translucent]="true">
 <ion-toolbar>
   <ion-title>
    Adivina el número
```

```
  </ion-title>
 </ion-toolbar>
</ion-header>


<ion-content [fullscreen]="true">
 <div class="ion-padding">
  <ion-input type="number" min="1" max="100" [(ngModel)] = "num"
placeholder="Inserta un número del 1 al 100"></ion-input>
  <p>El número secreto es {{mayorMenor}} {{num}}</p>
  <ion-button expand="block" (click) = "compruebaNumero()">Adivina</ion-
button>
 </div>
</ion-content>
```

Si guardamos y vamos al navegador veremos el siguiente resultado en pantalla.

Figura 8. Parte Visual de la app.

Si intentamos ejecutar un lector veremos que no nos lee ninguna etiqueta en donde hemos de introducir el número y que tras introducir el número el texto que hemos escrito desaparece y por lo tanto perdemos la información que necesitamos. Eso es un error de accesibilidad.

Si alguno a intentado poner un número y ha pulsado sobre el botón habrá comprobado que no sucede nada, eso se debe a que hemos programado la parte visual de la aplicación, conocido en inglés como front-end, pero no hemos programado la lógica de la aplicación, conocida en inglés como back-end. Esto también vamos a arreglarlo.

3. Haciéndola más Accesible

Lo primero que vamos a corregir es la accesibilidad. Si prestamos atención a la accesibilidad desde el inicio nos costará muy poco tiempo y recurso hacer que la aplicación sea accesible. Si, por el contrario, dejamos la accesibilidad para el final después nos llevará mucho tiempo y recursos hacer que la aplicación sea accesible. Por lo que es muy aconsejable invertir algo de tiempo a hacer la aplicación accesible desde el principio.

Para hacer que la aplicación sea accesible debemos introducir una etiqueta label antes de input i vincularlas entre ella. Para la vinculación deberemos utilizar el atributo for en la etiqueta label y los atributos name e id en la etiqueta input. Tras la modificación, las etiquetas label e input quedarán de la siguiente manera:

```
<ion-label for="Adivina">Inserta un número del 1 al 100</ion-label>
  <ion-input type="number" min="1" max="100" [(ngModel)] = "num"
name="adivina" id="adivina" placeholder="Inserta un número del 1 al
100"></ion-input>
```

Si ahora vamos al navegador, veremos que sobre el cuadro input aparece el texto que le hemos asignado a la etiqueta input. La verdad es que visualmente no queda muy bien, así que más tarde veremos cómo modificar el estilo u ocultarlo, como prefiramos.

Figura 9. Vista más accesible.

4. Programando la Lógica

La lógica de esta pestaña la escribiremos en TypeScript, un lenguaje muy parecido a JavaScript, utilizado para la realización de páginas web. EL código TypeScript de esta pestaña está en el archivo tab1.page.html, que encontraremos en el mismo lugar que **tab1.page.html**[2]

Lo primero que debemos hacer es incluir en nuestra clase las dos variables que hemos creado en la vista: num y mayorMenor. Para ello introduciremos las siguientes líneas después de **export Class Tab1Page {** y por encima de la línea del **constructor**.

```
num:number;
 mayorMenor='...';
```

A parte de estas dos variables que utilizamos en la parte visual de la aplicación, también necesitamos una variable para almacenar el número que creará el juego. Esta variable llamará a una función que

[2] Recordad que al inicio dijimos que cada elemento tenía su archivo de código, HTML y css.

es la que generará de forma aleatoria el número que hemos de adivinar. Así pues, debajo de mayorMenor escribiremos:

```
numSecret:number=this.numAleatorio(0,100);
```

Si, en efecto, el siguiente paso será la creación de la función para que genere el número de forma aleatoria. El nombre de la función ya lo hemos establecido en la variable numSecret y es numAleatorio(). La función las escribiremos debajo del construcor y tendrá la siguiente forma:

```
numAleatorio(a,b) {
   return Math.round(Math.random() * (b-a) + parseInt(a, 100));
 }
```

La función lo que hará será devolver un número aleatorio entre 1 y 100; por eso se ha insertado la palabra clave return al final de la función.

Ahora sólo nos falta crear la función que comprobará si el número que hemos introducido es el número correcto (igual) o si es mayor o menor al número que ha generado la función **numAleatorio()**.

El nombre de la función lo hemos definido en la parte visual de la aplicación, dentro de la etiqueta **button**, donde hemos creado una llamada a la función **compruebaNumero()**. La función de comprobación consiste en una concatenación de bucles if para seleccionar el texto que debe aparecer la pantalla y tiene la siguiente forma:

```
compruebaNumero() {
   if(this.num){
    if(this.numSecret < this.num){
     this.mayorMenor = 'menor que';
    }
    else if(this.numSecret > this.num){
     this.mayorMenor = 'mayor que';
    }
    else {
     this.mayorMenor = '';
    }
   }
 }
```

Si guardamos la función que acabamos de crear y abrimos el navegador, veremos que ahora sí que podemos ejecutar la aplicación, que cuando insertamos un número y pulsamos sobre el botón Adivina, se nos devuelve como resultado so nuestro número es mayor, menor o igual al seleccionado por la aplicación.

Figura 10. Ejemplo de funcionamiento de la app.

Ya tenemos una aplicación funcional, es decir, que hace lo que queremos, pero ahora vamos a lavarle un poco la cara para que sea algo más amigable. Ahora mismo, cuando adivinamos el número nos aparece prácticamente el mismo mensaje que cuando es mayor o menor.

5. Añadiendo una Tarjeta

Vamos a añadir una etiqueta **card** justo encima del botón para que aparezca cuando acertamos el número. Para ello, volveremos a abrir el archivo tab1.page.html y añadiremos las siguientes líneas:

```
. <ion-card *ngIf = "mayorMenor==''">
    <ion-card-header>
      ¡¡¡ Enhorabuen !!!
    </ion-card-header>
    <ion-card-content>
      En número secreto era el {{num}}.
    </ion-card-content>
  </ion-card>
```

Guardemos la modificación y juguemos para ver el resultado.

Figura 11. Mostrando la Card.

¡Genial!, al ganar nos aparece el resultado del vencedor, pero... hay un pequeño problema, sigue apareciendo el botón con el texto Adivina cuando ya hemos ganado y por tanto terminado el juego. Debería de aparecer un botón con el texto de Volver a jugar en su lugar. ¿Alguien sabe cómo se podría hacer? Efectivamente, utilizando el componente *ngIf en el botón.

6. Modificando el Botón

Así pues, lo que tenemos que hacer es una pequeña modificación en la etiqueta button y añadir un nuevo botón que será visible cuando acertemos el número secreto. Después de la modificación la etiqueta button debe quedar de la siguiente manera:

```
<ion-button *ngIf = "mayorMenor != '"" expand="block" (click) =
"compruebaNumero()">Adivina</ion-button>
```

Y el nuevo botón[3] debe tener la siguiente forma:

```
<ion-button *ngIf = "mayorMenor==''"" expand="block" (click) =
"reinicia()">Volver a la jugar</ion-button>
```

Guardemos y vayamos al navegador.

[3] Es una buena idea guardar y mirar cómo queda el juego en el navegador antes de insertar el código del nuevo botón.

Figura 12. Modificación del botón.

¡Funciona! En cuanto adivinamos el número desaparee el botón inicial y aparece uno con el teto de Volver a jugar. Pero... Un momento... El botón Volver a jugar no funciona. Cuando pulsamos este botón no sucede nada y sigue la misma pantalla, por lo que no se puede volver a jugar. ¿por qué?

El problema, está en que no hemos implementado la nueva función en el archivo tab1.page.ts.

7. Implementando Reiniciar

Vamos a implementar la nueva función reiniciar() para que al pulsar el botón se vuelva a calcular un nuevo número y de esta forma podamos volver a jugar. Para ello abriremos, como ya hemos dicho el archivo tab1.page.html e insertaremos la nueva función justo debajo del final de la última función del archivo.

La función **reinicia()** tendrá la siguiente forma:

```
reinicia(){
   // Reiniciamosls variables
   this.num = null;
   this.mayorMenor = null;
   this.numSecret = this.numAleatorio(0, 100);
 }
```

Como se puede ver, la función reinicia() lo único que hace es reiniciar las variables; para que de esta forma vuelva a mostrarse el botón de Adivina y se vuelve a calcular un nuevo número. ¡El juego funciona!

Ya tenemos un juego funcional y accesible, ahora vamos a implementar diferentes estilos para que también sea algo más vistoso y visualmente bonito. Para ello tendremos que trabajar con la hoja de estilos CSS.

8. Modificando los Estilos

Los estilos de esta pestaña los encontraremos en el archivo tab1.page.css. Si abrimos el archivo comprobaremos que está en blanco; ya que hasta ahora hemos estado utilizando los estilos propios de Ionic.

Una de las mejoras visuales podría ser que la palabra ¡¡¡ Enhorabuena !!! Estuviera en negrita y fuera más grande. Esto podría realizarse de dos formas diferentes o mediante el uso de identificadores de etiquetas o clases, añadiendo dentro de la etiqueta que contiene la palabra las palabras clave id o class o mediante la ruta de etiquetas. Por ahora, escogeremos este segundo método.

Si observamos el archivo tab1.page.html vemos que la palabra ¡¡¡ Enhorabuena !! Está entre etiquetas <ion-card-header>; así pues, en la hoja de estilos escribiremos los siguiente:

```
ion-card-header{
   font-size: 20px;
   font-weight: bolder;
}
```

Con esto, haremos que la palabra ¡¡¡ Enhorabuena !!! Aparezca en negrita y aumente su tamaño para que el usuario se fije más

9. Mejorando la Aplicación

Nuestro juego tiene muy buena pinta, hace lo que se le pide, es decir, hemos de adivinar el número, pero... puede mejorarse. ¿Cómo? La primera mejora sería que no nos apareciera la etiqueta del campo input una vez hemos adivinado el número, si no podemos volver a utilizar el botón de Adivina, ¿para qué nos sirve dicho campo?

Así pues, esa será nuestra primera mejora.

10. Ocultando el Campo Input

¿Cómo podemos hacer que se oculte este campo? Sí, tienes toda la razón, de la misma forma que hacemos desaparecer el botón Adivina, con *ngIf.

Para que la etiqueta del campo input desaparezca debemos modificar la etiqueta de la siguiente manera:

```
<ion-label for="Adivina" *ngIf = "mayorMenor != ''">Inserta un número del 1 al 100</ion-label>
```

Después de realizar la modificación guardamos y vamos al navegador. Ahora cuando acertamos el número ya no nos realiza la pregunta, con lo que la vista queda más limpia. Ahora tengo dos ejercicios para ti. ¿Cómo harías desaparecer el campo input (donde se estribe el número) y la respuesta que nos da el juego debajo?

Esto te los dejo como un ejercicio para que pienses como hacerlo. Aunque aquí debajo te dejo la solución

11. Solución Ejercicio

La solución es volver a utilizar *ngIf en las etiquetas que queremos que desaparezcan cuando hemos solucionado el número. Las cuáles deberían de quedar como sigue:

```
<ion-input type="number" min="1" max="100" [(ngModel)] = "num"
name="adivina" *ngIf = "mayorMenor != '"" id="adivina" placeholder="Inserta un
número del 1 al 100"></ion-input>
   <p *ngIf = "mayorMenor != '"">El número secreto es {{mayorMenor}}
{{num}}</p>
```

12. Aspecto Final

¡Genial! ¡Sabía que lo conseguirías! El aspecto final de la aplicación es bueno, por lo general. Después de eliminar de la pantalla todo aquello que no es necesario en dicho momento hace que el juego quede más limpio y no nos dé información que en ese momento no necesitamos y puede liarnos. En la siguiente imagen se muestra como es la aplicación al iniciarse y cómo debe de quedar al finalizar el juego.

Figura 13. Pantalla inicial de la App.

Figura 14. Pantalla durante el desarrollo de la App.

13. Complicando un poco el Juego

Podemos complicar un poco el juego insertando un contador de intentos. De esta forma sabremos si vamos mejorando nuestra puntuación y afinando más el método de búsqueda del número oculto.

14. Creando un Contador

Para crear el contador en la parte visual nos fijaremos en la línea donde mostramos el número que hemos introducido como intento para resolver el acertijo. Así pues, la línea con los intentos tendría la siguiente forma:

```
<p>Número de intentos: {{ count }}</p>
```

En nuestra aplicación el resultado será el siguiente.

Figura 15. Mostrando los intentos.

Si ejecutamos el juego veremos que el contador no muestra el número de intentos. Eso se debe a que nos falta por implementar la lógica de la aplicación.

15. Implementando la Lógica del Contador

Bien, vamos a implementar la lógica para la etiqueta que nos muestra el contador. Así que abriremos el archivo tab1.page.ts y añadiremos la nueva variable count debajo de numSecret. Lo que debemos añadir es:

```
count:number=0;
```

En este caso hemos de inicializar la variable count para poder empezar a sumar los intentos que hemos realizado.

Para contar los intentos lo más sencillo es utilizar la función compruebaNumero() y dentro de cada bucle realizar una suma por intento realizado. Para ello, la función deberá quedar de la siguiente forma:

```
compruebaNumero() {
  if(this.num){
    if(this.numSecret < this.num){
      this.mayorMenor = 'menor que';
```

```
      this.count++;
    }
    else if(this.numSecret > this.num){
      this.mayorMenor = 'mayor que';
      this.count++;
    }
    else {
      this.mayorMenor = '';
    }
  }
}
```

Si ahora ejecutamos el código veremos que tenemos un error. Al reiniciar el juego nuestro contador no vuelve a 0, es decir, los intentos anteriores siguen sumando para los nuevos juegos y esto es un error. El error se debe a que debemos reiniciar el contador cada vez que pulsamos el botón para iniciar un nuevo juego. Para ello añadiremos la siguiente línea al final de la función reiniciar():

```
this.count = 0;
```

Ahora sí, al reiniciar el juego nuestro contador vuelve a cero reiniciándose y por tanto los intentos que estamos contando son los utilizados para adivinar cada número y no mantenemos los intentos de juegos anterior.

16. Ejercicio

Como puede observarse, el juego es mejorable. Como ejercicio se deja para que se modifique el código para obtener el resultado que muestran la siguiente imagen.

Figura 16. Pantalla final de juego ganado.

17. Más Allá

El juego puede mejorarse limitando el número de intentos a uno determinado, y en caso de sobrepasarse mostrar una vista donde se diga que se han superado en número máximo de intentos y

se nos da el juego por perdido. Se deja a vuestra elección cómo realizar las modificaciones necesarias para obtener dicho comportamiento del juego.

Capítulo III: Sumando números

1. Introducción

En la segunda pestaña de nuestra aplicación vamos a hacer un juego para calcular la suma, se pondrá como ejercicio que la multiplicación, de dos números aleatorios dados por el teléfono, de esta forma practicaremos nuestras matemáticas y afianzaremos los conocimientos que hemos adquirido en el capítulo anterior.

Sin más preámbulos vamos a ello.

2. Cambiando el Nombre a la Pestaña

Lo primero que haremos es cambiar el nombre de la pestaña 2 a la que asignaremos el nombre de Operaciones. Supongo que recordaréis que los nombres de las pestañas están en el archivo tabs.page.html, así que sólo tendremos que ir a dicha página y donde pone **Tab 2** poner **Operaciones**.

Veo que ya lo habéis hecho, ¡Así me gusta!

3. Cambiando el Título

El siguiente paso, como siempre, será modificar el título de la página de Operaciones. ¿Alguien se acuerda donde se realiza ese cambio? Sí, en el archivo tab2.page.html. Así que abramos el archivo y veamos su contenido.

Lo primero que haremos será substituir Tab 2 por Operación matemática. Una vez hecho esto vamos a vaciar de contenido la página, así que todo lo que esté entre las etiquetas <ion-content> lo vamos a borrar. La segunda pestaña de la aplicación debería tener la siguiente forma:

Figura 17. Título de la App.

4. Llenando de Contenidos

Tal y como hicimos en el ejemplo anterior. Lo primero que haremos será montar la estructura visual de la pestaña. Para ello modificaremos el código presente en el archivo **tab2.page.html** para obtener una apariencia como la que se muestra a continuación.

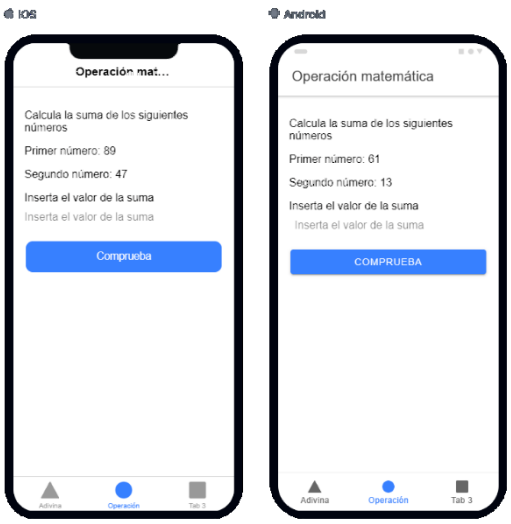

Figura 18. Pantalla inicial de Operaciones.

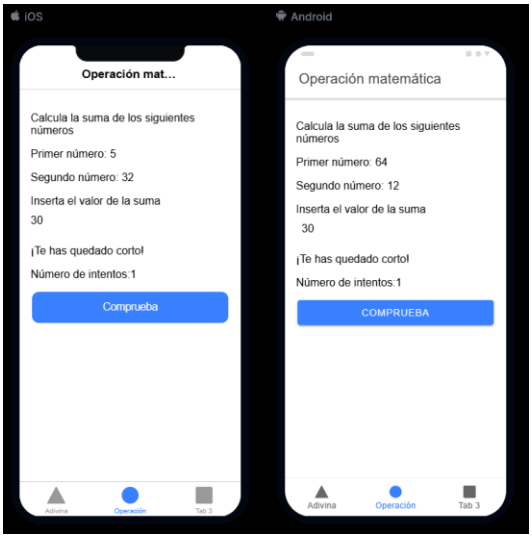

Figura 19. Pantalla de suma errónea (resiltado mayor).

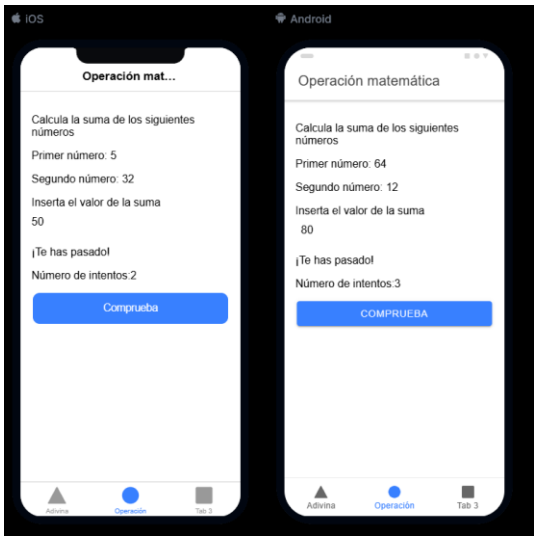

Figura 20. Pantalla de suma errónea (resultado menor).

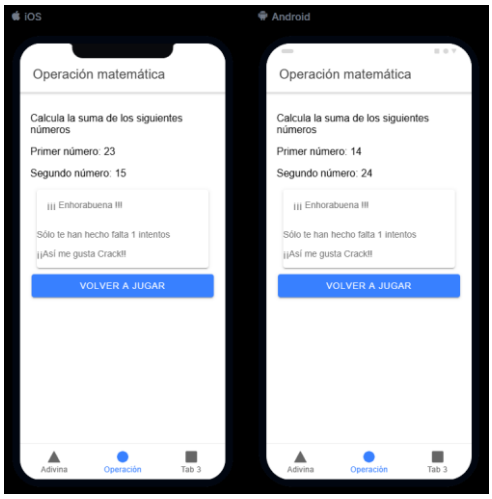

Figura 21. Pantalla de suma correcta.

En esta ocasión se dejará que creéis la parte visual de la aplicación basándoos en la apariencia antes del juego y la apariencia durante el juego y con la pantalla de juego superado o juego perdido.

En las vistas se muestran todos los elementos necesarios para crear la aplicación, y estudiando las vistas podéis empezar a pensar que elementos y condiciones serán necesarias para que aparezcan unos elementos u otros durante el juego.

Una vez tengamos la parte visual resuelta, será la hora de ir a la lógica de la aplicación para que se nos muestren los elementos.

5. Lógica de Calcula

La lógica de la pestaña hemos de implementarla en el archivo **tab2.page.ts**. Mirando las variables y funciones de la parte lógica podréis completar o intuir por dónde va la parte visual de la aplicación, así que un buen estudio del código puede ser una buena idea. Lo primero que debemos hacer es crear las variables. En este caso necesitaremos un total de 8 variables:

- Una para la respuesta introducida.
- Dos para los números aleatorios generados.
- Una para el resultado de la suma.
- Una para devolver el mensaje del resultado.
- Una para mostrar el nivel del participante.
- Una para devolver false o true.
- Una para mostrar el mensaje de victoria o derrota.

A continuación, se muestran las variables:

```
num:number;
  num1:number = this.numAleatorio(0,100);
  num2:number = this.numAleatorio(0,100);
  sum:number = this.calculateSum();
  result = "";
  level = "";
  check:boolean = false;
```

```
count:number = 0;
```

También necesitaremos algunas funciones para realizar los cálculos necesarios para dar una respuesta al usuario dependiendo del número introducido, así como reiniciar las variables una vez finalizado el juego.

6. Función numAleatorio()

La función **numAleatorio()** será una copia de la utilizada en la aplicación anterior, pero esta vez será llamada desde num1 y num2, pues necesitamos 2 números aleatorios. La función tiene la siguiente forma:

```
numAleatorio(a,b) {
  return Math.round(Math.random() * (b - a) + parseInt(a, 10));
 }
```

7. Función calculateSum()

Esta función calculará la suma a partir de los dos números aleatorios proporcionados por la función anterior. El cálculo de la suma es necesario para después realizar la comparación con el valor introducido por el usuario.

La función tiene la siguiente forma:

```
calculateSum() {

  return this.num1 + this.num2;

 }
```

8. Función checkAnswer()

Esta función comprueba si el resultado introducido por el usuario y el calculado es el mismo. En caso de serlo devolverá una victoria y llamará a la función playerLevel(), si no es el mismo resultado informará si nos hemos pasado o quedado cortos con el número introducido. Por último, en caso de realizar un 4 intento fallido devuelve un mensaje de derrota y finaliza el juego.

La función tiene la siguiente forma:

```
checkAnswer(){
  if(this.sum == this.num){
    this.result = "Respuesta correcta";
    this.check = true;
    this.count++;
    this.playerLevel();
  }
  else if(this.sum < this.num){
    this.result = "¡Te has pasado!";
    this.count++;
  }
```

```
  else if(this.sum > this.num){
    this.result = "¡Te has quedado corto!";
    this.count++;
  }


  if(this.count == 4){
    this.playerLevel();
  }
}
```

9. Función playerLevel()

Esta función es llamada por la función anterior y nos envía un mensaje en función de los intentos que hayamos necesitado para resolver la suma propuesta.

En esta función se ha optado por el uso de un bucle switch en lugar de un bucle if para mostrar el uso de este bucle. Los buces switch se suelen utilizar en lugar de los if cuando se deben realizar muchas concatenaciones (bucles else if, ya que es algo más rápido en su ejecución.

La función tiene la siguiente estructura:

```
playerLevel(){
  switch(this.count){
    case 1:
      this.level = "Crack!!";
      break;
    case 2:
      this.level ="Bueno!!";
      break;
      case 3:
        this.level ="Normalillo!!";
        break;
        case 4:
          this.level ="...!!";
          break;
  }
}
```

10. Función reiniciar()

Esta función realiza lo mismo que la función reiniciar de la aplicación anterior, es decir, resetea todas las variables a sus valores originales para poder retomar el jugo.

Tiene la siguiente forma:

```
reinicia(){

  this.num = null;
  this.num1 = this.numAleatorio(0, 100);
```

```
    this.num2 = this.numAleatorio(0, 100);
    this.result = "";
    this.level = "";
    this.count = 0;
    this.check = false;
  }
```

11. Ejercicios

Se proponen diferentes ejercicios para ver si se ha comprendido el uso del lenguaje y de la aplicación. Como podréis ver, los ejerccios propuestos son sencillos y fácil de resolver si se ha comprendido el código.

1. Modificar la aplicación que en lugar de una suma se realice lo mismo con una multiplicación.
2. Modificar los mensajes de victoria y derrota dependiendo de los intentos realizados.
3. Hacer que los intentos permitidos sean 5.
4. Incluir un botón que dé la posibilidad de reiniciar el juego a partir del 3 intento.

Capítulo IV: Mayor que … Menor que

1. Introducción

EN la última pestaña vamos a generar un juego para ordenar 6 números aleatorios de mayor a menos. Esta vez la lógica va a ser más complicada, debido a que también se comprobará que se está introduciendo. Hasta ahora, se podía introducir cualquier valor en el campo de edición o simplemente no introducir ninguno. En esta aplicación se verá si se ha introducido un número y que el número introducido es uno de los números aleatorios dados. Por tanto, la parte visual y la de la lógica serán algo más complicadas que la de los capítulos anteriores.

Pero no nos atemoricemos antes de empezar, como verás, se volverán a reutilizar aplicaciones y conceptos ya vistos hasta ahora, así que no será tan duro como parece.

2. Parte visual de la pantalla

Vamos a dar por sentado que ya se sabe modificar el título, en nombre de la pestaña y otras partes visuales que ya se han visto en los capítulos anteriores. La pantalla inicial tendrá la siguiente apariencia.

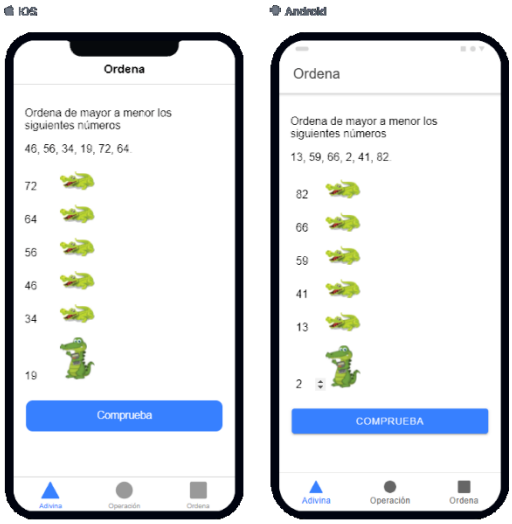

Figura 22. Pantalla inicial de Mayor que...

Como se ya se han visto la forma de implementar los campos de edición (**input**), generar números aleatorios, párrafos y demás, sólo se va a explicar la forma de insertar los dibujos de los cocodrilos con los que se realiza el signo de mayor qué.

Se ha optado por la utilización de estos dibujos ya que es una aplicación para niños y se intenta hacer algo más divertida y visualmente motivadora, divertida y bonita.

El código para insertar las imágenes[4] es el siguiente:

```
<img src="../../assets/cocodrilo.png">
```

Al insertar este código se nos mostrará la imagen que esté en la ruta especificada dentro de **src**.

[4] Las imágenes pueden descargarse de la página web:

¿Cómo? Que los cocodrilos no te salen al lado del campo de edición. Hmmm, ya veo. El problema está en los estilos.

Para que la pantalla tengo el aspecto visual que se muestra en la figura anterior, se deberá retocar los estilos de la vista. Aunque antes deberemos de escribir el código HTML formando bloques. A continuación, se muestra el código para el primero de los 6 bloques que conforman el campo de edición y el dibujo del cocodrilo:

```
<div>
 <ion-input type="number" min="1" max="200" [(ngModel)]="numIni1"
name="num1"></ion-type>
  <img src="../../assets/img/cocodrilo.png">
</div>
```

Ahora veamos el archivo de estilos **tab3.page.css**.

3. Modificando los estilos.

Para conseguir la apariencia que se muestra en la pantalla inicial, se debe abrir la hoja de estilos para la pestaña 3 de nuestro proyecto: **tab3.page.css** e insertar las siguientes líneas.

```
div ion-input, div img {
  padding-top: 1vh;
  display: inline-block:
}
Img {
  width: 50px;
}
```

Con estas líneas estamos diciendo al programa cómo queremos que nos disponga los elementos en pantalla. Para ello primero debemos poner la ruta hasta el elemento a configurar. Por ejemplo, **div ion-input** significa que queremos que lo que está entre claves afecte a todos los elementos **ion-input** que estén contenidos dentro de un elemento **div**. En caso de que queramos afectar a más de un elemento de la misma forma, separaremos los elementos mediante comas. Tal y como se realiza en la primera línea del código.

Con estas líneas dentro de la hoja de estilos ya tendríamos en pantalla el resultado que buscamos. Ahora vamos a meternos con la lógica de la aplicación.

5. Las variables

Para poder realizar esta aplicación vamos a necesitar un total de 14 variables:

- Una para cada campo **input** que denominaremos: numIni1, numIni2…, numIni6.
- Una para cada número aleatorio que denominaremos: num1, num2…, num6.
- Una para realizar la comprobación.
- Una para devolver el resultado.

Por el momento, no nos harán falta más variables, aunque a medida que vayamos desarrollando la aplicación iremos añadiendo hasta un total de 22 variables.

6. Las funciones

Nos harán falta un total de 3 funciones. Dos de ellas ya las hemos visto con anterioridad: **numAleatorio()** y **reinicia()** son las mismas que hemos visto con anterioridad. A la función **numAleatorio()** no le debemos realizar ninguna modificación, y a la función reinicia sólo se le debe indicar que también reinicie las variables nuevas.

Así pues, la única función nueva será la de **checkAnswer()**, que es la función a la que llama el botón comprobar. Esta función es básicamente la concatenación de bucles **if** para comprobar cada dos campos si el número introducido es mayor, Es decir, comprobaremos si el número introducido en el campo **numIni1** es superior al del **numIni2**, en caso de serlo se comprobará si el introducido en **numIni2** es superior al introducido en **numIni3**, y así sucesivamente.

Mientras vaya siendo cierto se irá realizando una nueva comprobación hasta llegar al último número, en caso de que una comprobación no ser cierta se finaliza el bucle y se devuelve el mensaje de error.

```
checkAnswer(){
  if(this.numIni1 > this.numIni2) {
    if(this.numIni2 > this.numIni3) {
      if(this.numIni3 > this.numIni4) {
        if(this.numIni4 > this.numIni5) {
          if(this.numIni5 > this.numIni6) {
            this.check=true;
          }
          else {
            this.result = “El número “ + {{ this.numIni5 + “ no es mayor que “ + numIni6;
            return;

          }
        }
        else {
            this.result = “El número “ + {{ this.numIni4 + “ no es mayor que “ + numIni5;
            return;

        }
      }
      else {
          this.result = “El número “ + {{ this.numIni3 + “ no es mayor que “ + numIni4;
          return;

        }
      }
      else {
          this.result = “El número “ + {{ this.numIni2 + “ no es mayor que “ + numIni3;
          return;
```

```
 }
}
else {
    this.result = “El número “ + {{ this.numIni1 + “ no es mayor que “ + numIni2;
    return;
 }
}
```

Una vez implementada la función, si vamos al navegador y ejecutamos el juego veremos que ya funciona.

- ¡Oye!

- ¿Qué?

- El lector de pantalla no me dice nada. No habla, ¡no sé lo que estoy haciendo!

- ¡Ups! Me olvidé la accesibilidad.

7. Haciéndolo accesible

Cierto, cuando ejecutamos un lector de pantalla, éste sólo nos va de un campo de edición a otro si decir que hemos de introducir en ese campo de edición, ese es un error en la accesibilidad, pero también hay otros errores. Por ejemplo, no podemos navegar entre botones. Así que vamos a solucionarlo.

a. Detectando botones

Lo primero que vamos a hacer es que todos los bloques que contienen los campos de edición y los dos botones se incluyan dentro de una etiqueta de formulario **<form>**.

b. Diciendo que hemos de introducir en los campos

Para que los lectores de pantalla nos den información sobre lo que hemos de introducir en los campos de edición necesitamos añadir las etiquetas del campo de edición y enlazarla con dicho campo.

Esto se hace de la siguiente manera:

```
<div>
  <ion-label for=”num1” >Inserta el primer número</ion-label>
  <ion-input type=”number” min=”1” max=”200” [(ngModel)]=”numIni1”
id=”num1” name=”num1”></ion-type>
  <img src=”../../assets/img/cocodrilo.png”>
</div>
```

Tras realizar estas modificaciones nos quedará hacer que los lectores de pantalla detecten los dibujos y diga “mayor que” cuando pasamos de un capo de edición a otro.

c. Detectando los dibujos

Para que los lectores de pantalla detecten los dibujos y envíen el mensaje sonoro que deseamos, en lugar del nombre del archivo, debemos utilizar la propiedad de texto alternativo utilizando la propiedad **alt**. Para ello realizaremos la siguiente corrección en el código mostrado con anterioridad.

```
<div>
  < label for="num1" class="hidden" >Inserta el primer número</ion-label>
  <input type="number" min="1" max="200" [(ngModel)]="numIni1" id="num1"
name="num1"></ion-type>
  <img src="../../assets/img/cocodrilo.png" alt="mayor que">
</div>
```

Como se puede ver, se ha modificado la etiqueta <ion-label> por la <label>. Esto se ha hecho porque se ha visto que la etiqueta <ion-label> no se enlaza de forma correcta con la etiqueta <input> y por tab, no es tan accesible como la otra. Lo mismo se ha hecho con la etiqueta <ion-input>

Con estas modificaciones, en principio, deberíamos tener solucionado el problema de la accesibilidad de la pantalla, pero tenemos un problema. Si vamos a ver la ejecución de la aplicación veremos que delante de cada cuadro de edición nos aparece la etiqueta, y esto visualmente no queda bonito, destroza la vista.

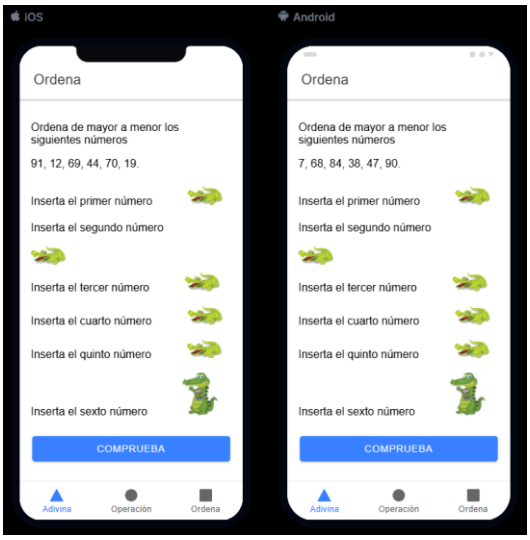

Figura 23. Etiquetas label visibles.

d. Ocultando las etiquetas

Hemos de ocultar las etiquetas de los campos de edición, ¿a alguno se le ocurre cómo podríamos hacerlo? Sí, efectivamente, hemos de modificar los estilos. Así que vayamos al fichero de estilos y añadamos las siguientes líneas:

```
.hidden{
  position: absolute;
  overflow: hidden;
  top: -9999px;
```

```
}
```

Lo que estamos realizando en estas líneas es decir que los elementos de la clase hidden se posicionen en la a -9999 píxeles de la parte superior de la pantalla. De esta forma nos llevamos la etiqueta fuera de la pantalla y no aparece en la misma, pero al estar vinculado con el elemento del campo de edición, los lectores de pantalla nos indicarán qué debemos escribir en el campo de edición.

8. Mejorando la aplicación

La aplicación puede mejorarse añadiendo los siguientes comportamientos:

- Detección de campos en blanco.
- Marcar donde está el error.
- Animación de finalización de juego.

Se os deja pensando cómo resolver estas mejoras del juego.

9. Ejercicio

Como ejercicio se propone que se modifique la aplicación vista para que, en lugar de ordenar los números de mayor a menor, se orden de menos a mayor.

Capítulo V: Generando la aplicación

1. Introducción

Existen diferentes formas para desplegar una aplicación realizada con Ionic. En este capítulo se mostrarán dos formas diferentes, mediante la página web de Ionic (de pago) y mediante línea de comando (gratuito).

Mediante línea de comando se mostrará como configurar el equipo para hacerlo y los diferentes comandos disponibles que lo permiten.

2. Utilizando Ionic Framework.

La página web de Ionic Framework permite el empaquetado de aplicaciones para Android y/o iOS de una forma sencilla y en pocos clicks del ratón. Para poder utilizarla nos deberemos inscribir previamente, la inscripción es gratuita.

Una vez inscritos y dentro de la sección de Dashboard desplegaremos el botón New y seleccionaremos Import existing app. Tras hacerlo se nos mostrarán diferentes repositorios donde podemos almacenar código. Debemos tener una cuenta en uno de estos repositorios, donde habremos subido previamente el código de la aplicación. Los repositorios son:

- Github.
- Gitlab.
- Bitbucket.

Tras seleccionar el repositorio, se deberá seleccionar la carpeta en la que está la aplicación que queremos crear. Una vez seleccionada y pulsado Choose se nos mostrará en la página del dashboard la aplicación que queremos empaquetar. El siguiente paso será pulsar sobre la aplicación para acceder a la misma.

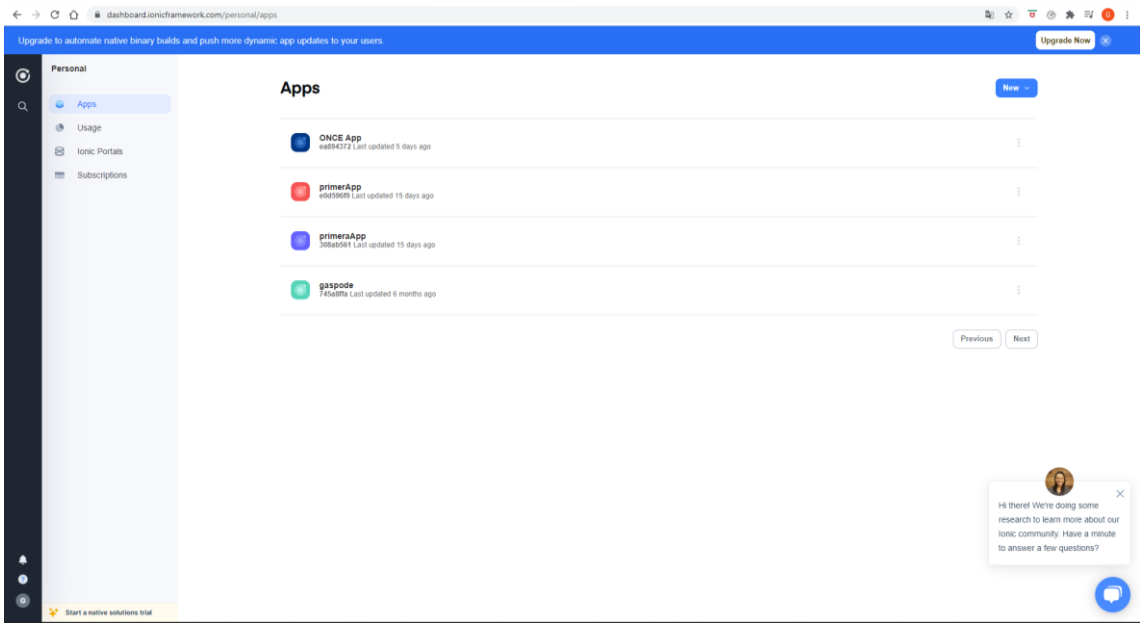

Figura 24. Aplicacioones cargadas en el Dashboard.

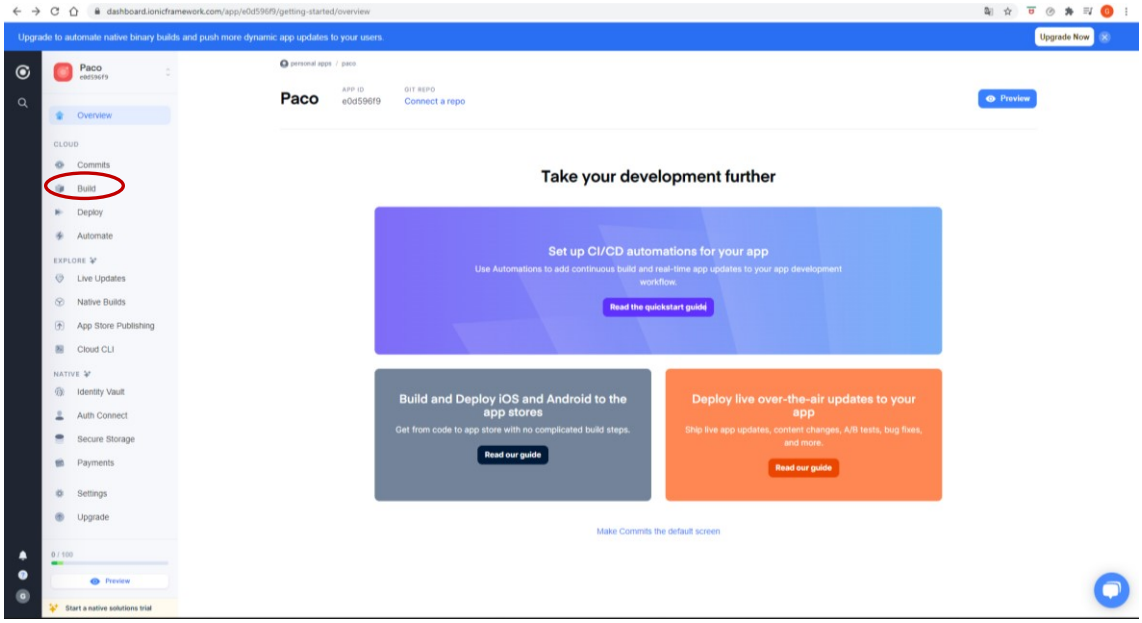

Figura 25. Dentro de la aplicación del Dashboard.

Tras acceder a la aplicación pulsaremos sobre la opción build que aparece en el menú vertical de la izquierda de la pantalla y seleccionaremos el sistema en el que queremos empaquetar la aplicación. Los sistemas disponibles son:

- Web (JavaScript).
- Android.
- iOS.

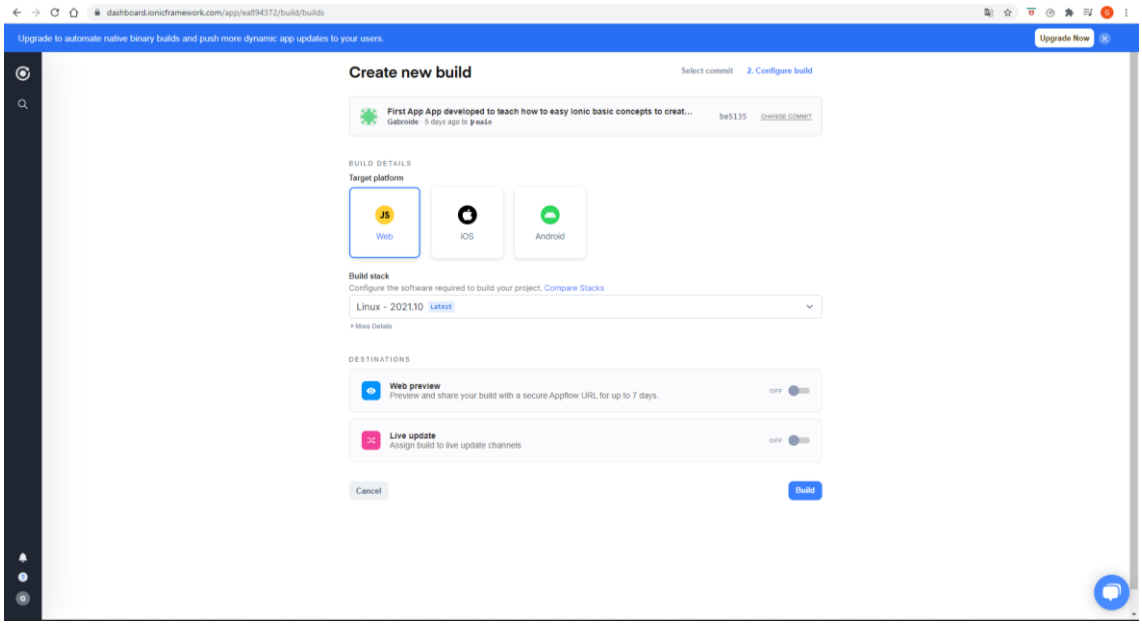

Figura 26. Pantalla para seleccionar el Sistema Operativo.

Para empaquetar la aplicación para los sistemas Android y iOS debemos ser usuarios de pago de la plataforma, no es así si queremos empaquetar la aplicación para utilizarse en un navegador web. Como ejemplo empaquetaremos para un navegador web, para ver los pasos a dar.

Tras selecciona el sistema, activaremos las opciones de Web preview, que nos permitirá mostrar la aplicación en un web de la plataforma durante un tiempo y la opción Live update para permitir que se realicen actualizaciones. Una vez realizados estos pasos nos quedará seleccionar el canal, que sólo podrá ser de Producción. Tras hace lo pulsaremos sobre el botón Build para iniciar el empaquetado.

El proceso es automático y cuando finalice tendremos ya la aplicación empaquetada y lista para utilizar en la plataforma que hayamos seleccionado con anterioridad.

3. Utilizando la línea de comandos de Ionic

Otra forma de empaquetar la aplicación es utilizando la línea de comando de Ionic. Para hacerlo deberemos introducir comandos mediante el teclado en un orden determinado. Así que vamos a abrir el terminal[5] y situarnos en la ubicación del proyecto de Ionic.

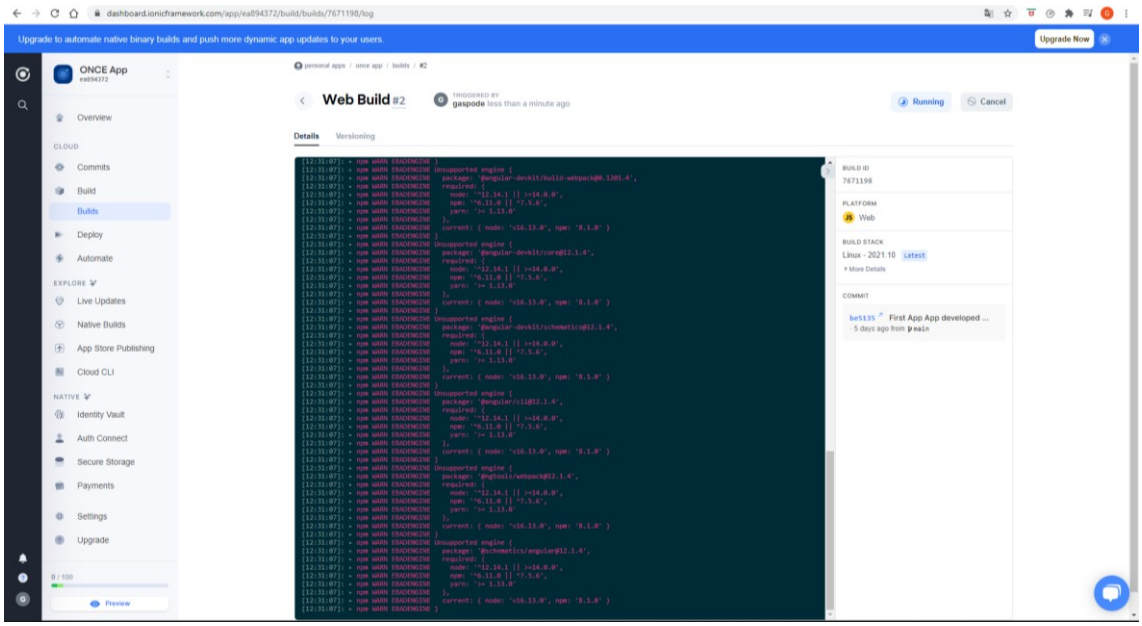

Figura 27. Panralla de creación de la aplicación.

Una vez dentro de la carpeta introduciremos el comando para añadir la plataforma o plataformas para las que queremos empaquetar la aplicación:

ionic build

A continuación, seleccionaremos la plataforma o las plataformas para nuestro proyecto:

ionic cap add ios

ionic cap add android

Al hacerlo se crearán dos carpetas en el directorio raíz del proyecto con código nativo totalmente independiente en cada carpeta, una para iOS y otra para Android. Cada vez que se ejecute el comando *ionic build* para actualizar la aplicación, se deberán copiar las actualizaciones con el comando:

ionic cap copy

[5] En caso de que estemos ejecutando el comando serve deberemos cancelarlo mediante la combinación Control + C.

Después de la actualización deberemos realizar la sincronización mediante el comando:

ionic cap sync

a. Desarrollo para iOS

Para poder desarrollar la aplicación para iOS deberemos ejecutar los comandos en un Mac, ya que estos dispositivos contienen los paquetes nativos necesarios para poder empaquetar la aplicación.

Las aplicaciones iOS de Capacitor están configuradas a través de XCode. Antes de ejecutar esta aplicación se deben realizar algunas pequeñas configuraciones.

1. Ejecutar el comando para abrir el proyecto nativo de iOS en XCode:

   ionic cap open ios

2. Para algunas aplicaciones serán necesarios algunos permisos para nuestra aplicación. Para dar estos permisos se debe abrir el archivo *Info.plist* y añadir los permisos necesarios dentro de *Propiedades objetivo-personalizadas de iOS*.
3. En el navegador de proyectos pulsar sobre *App* y luego dentro de *Signing & Capabilities* seleccionar *Equipo de Desarrollo*.
4. Conectamos un dispositivo iOS y seleccionamos *App > iPhone de …* y pulsamos el botón *Compilar* para compilar y ejecutar la aplicación en el dispositivo.

b. Desarrollo para Android

La aplicación *Capacitor* está configurada y gestionada mediante la aplicación Android Studio; por lo que será necesaria su descarga e instalación. Como en el caso anterior, será necesaria una pequeña configuración del equipo antes de poder empaquetar la aplicación.

1. Ejecutar el comando para abrir el proyecto nativo en Android Studio:

   ionic cap open android

2. Modificar los permisos necesarios para la aplicación en el archivo AndroidManifest.xml que está en la ruta: ./android/app/src/main/.
3. Conectar un dispositivo Android y pulsar sobre el botón Ejecutar.
4. Seleccionar el dispositivo conectado dentro de las opciones.

5. Utilizando comando de Cordova

Otra opción es la utilización de Cordova. Utilizando esta opción no será necesario el uso de XCode o AndroidStudio, ya que el proyecto es empaquetado por Cordova. Para ello primero añadiremos la plataforma:

cordova platform add ios

cordova platform add android

Como en el caso anterior, deberemos configurar el equipo para las siguientes plataformas.

a. Empaquetando para iOS

Para poder empaquetar la aplicación para iOS, es necesario ejecutar el comando en un Mac y tener instalado la aplicación XCode. El comando para ejecutar es:

cordova build ios

Tras la ejecución del comando se realizará la conversión del proyecto a código nativo de iOS y se empaquetará el proyecto para ser utilizado en dispositivos iOS.

b. Empaquetando para Android

Para poder empaquetar el proyecto para dispositivos Android, será necesario la descarga de ciertas aplicaciones y la creación de variables de entorno del sistema.

La primer aplicación que necesitaremos será JDK 1.8.x de Java. Tras la descarga y la actualización será necesario crear una nueva variable del sistema con el nombre JAVA_HOME y donde se escribirá la dirección completa de la instalación de la aplicación.

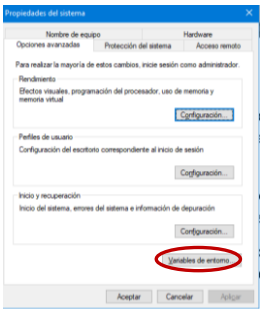

Figura 28. Pantalla de Propiedades del Sistema.

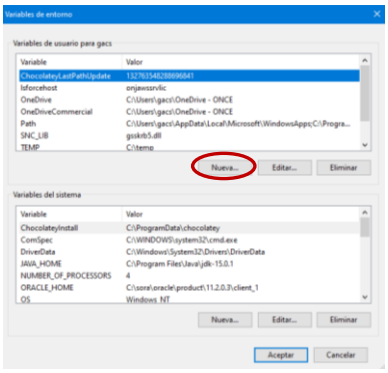

Figura 29. Pantalla de Variables de Entorno.

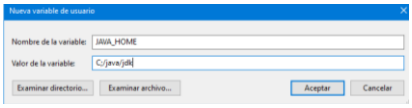

Figura 30. Creación de una nueva Variable de Entorno.

También será necesaria la descarga e instalación de la aplicación AndroidStudio, ya que es necesario el SDK de Android y éste está contenido dentro de la aplicación. Tras descargar la aplicación será necesaria la creación de variables de entorno: ANDROID_SDK_RROT y ANDROID_HOME. En estas variables se pondrán la ruta de la carpeta tools y platform-tools.

Con esta última modificación ya sería suficiente la configuración del sistema, pero dependiendo de cómo se haya realizado la instalación de la aplicación AndroidStudio, se deberá modificar la variable PATH añadiéndose la ruta de la carpeta gradle. En caso de que no se sepa la ruta de dicha carpeta, se puede descargar la aplicación desde su página web.

Tras realizar esta configuración ya estamos en disposición de empaquetar el proyecto para el sistema Android. Para ello ejecutaremos el comando:

cordova build andrid

De forma automática se empezará el empaquetamiento de la aplicación y se convertirá el código en código nativo para Android.