

# 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 cómo utilizar bucles en TypeScript, Formularios y la reproducción de sonidos.



## Contenido

Capítulo I: Sonidos de Animales .....	3
1. Objetivos .....	3
2. Creando el proyecto .....	3
3. Estructura del proyecto.....	3
4. Modificando la vista .....	4
Añadiendo archivo de datos .....	5
5. Creando un bucle en la vista .....	8
6. Reproduciendo sonidos .....	9
7. Actualizando la vista.....	10
8. Mejorando la app.....	11
Incluyendo la Accesibilidad .....	13
9. Trabajando con estilos .....	14
10. Mejorando la accesibilidad .....	17
Capítulo II: Adivina el Animal.....	18
1. Objetivo .....	18
2. Creando una nueva página .....	18
3. Creación de formularios .....	18
4. Validando datos.....	19
5. Incluyendo un botón .....	21
6. Creando un enum.....	22
7. Creando la primera pista .....	24
8. Añadiendo un botón .....	24
9. Añadiendo una segunda pista .....	26
10. Ocultando las imágenes.....	28
11. Validando la respuesta.....	29
12. Añadiendo la lógica a Respuesta .....	30
13. Implementando la puntuación .....	31
14. Insertando un botón reiniciar .....	34
15. Mejorando la visibilidad .....	35
16. Pequeñas mejoras .....	36
Añadiendo un SideMenu .....	38
Creando una nueva página .....	38

## Capítulo I: Sonidos de Animales

### 1. Objetivos

En este manual se verá cómo utilizar sonidos en una aplicación iónica. Para ello se creará un nuevo proyecto del con la plantilla **blank** de Ionic y se generarán varias vistas donde se verá el uso de sonidos con Ionic. Los objetivos que se cubrirán son:

- Creación de un proyecto en blanco.
- Reproducir sonidos.
- Uso de componentes **ion-list** e **ion-item**.
- Uso de directivas Angular.
- Evento click/tap.
- Accesibilidad.

### 2. Creando el proyecto

Para crear el nuevo proyecto escribiremos en un terminal el siguiente comando:

```
$ionic start sonidos blank --type=angular
```

Con este comando estamos diciendo a Ionic que cree un nuevo proyecto llamado **sonidos** utilizando la plantilla **blank** y con el framework de **Angular**.

Tras crear el proyecto utilizaremos el comando de Ionic para ver el aspecto de la aplicación.

```
$ionic serve -l
```

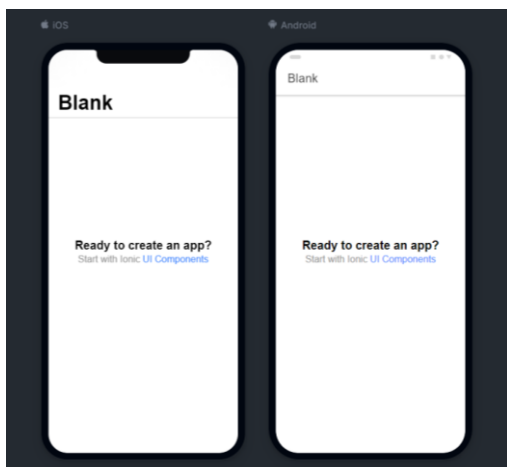


Figura 1. Vista inicial del proyecto.

### 3. Estructura del proyecto

Hemos utilizado la plantilla **blank** para la creación del proyecto; por lo que vamos a ver qué pinta tiene dicha plantilla.

La parte importante de la plantilla está contenida dentro de la carpeta **src/app/home**, que es donde está el archivo de la lógica de la aplicación, **home.page.ts**, y la parte visual, **home.page.html** (vista) y **home.page.css** (estilos). Trabajaremos principalmente en esta carpeta para la creación de la

aplicación; si bien, crearemos un sidemenu para lo que será necesario realizar algunas modificaciones en ficheros externos. Pero no adelantemos acontecimientos y veamos cómo desarrollar la primera vista de la aplicación.

#### 4. Modificando la vista

Lo primero que debemos hacer es modificar la vista actual de la aplicación. Para ello abriremos el fichero **home.page.html** y realizaremos algunas modificaciones en el mismo.

La primera modificación será la del título de la vista. Para ello dentro de la etiqueta **<ion-title>** escribiremos Sonidos. El código debería quedarnos de la siguiente manera:

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>
      Sonidos
    </ion-title>
  </ion-toolbar>
</ion-header>
```

El siguiente paso será asignar una clase a la etiqueta **<ion-toolbar>**; para ello escribiremos lo siguiente:

```
<ion-toolbar class="primary">
```

El siguiente paso será borrar todo el contenido que está entre las etiquetas **<ion-content>**. Tras realizar las modificaciones el aspecto de la aplicación debería de ser el siguiente.

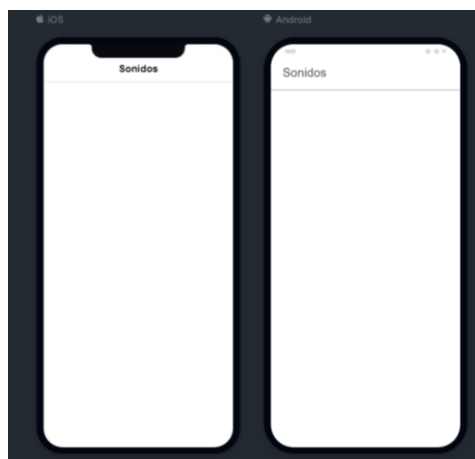


Figura 2. Aspecto de la app tras las modificaciones iniciales.

El siguiente paso será la creación de una lista que contendrá todos los botones para la reproducción de los diferentes sonidos. Para ello escribiremos lo siguiente dentro de la etiqueta **<ion-content>**:

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

```

<ion-list>
  <ion-item>
    <ion-avatar item-left>
      
    </ion-avatar>
    <h2>Caballo</h2>
  </ion-item>
</ion-list>
</ion-content>

```

Nuestra idea es la creación de un botón para cada uno de los sonidos, y se quiere utilizar un total de 8 sonidos diferentes, y para cada botón hemos de escribir 6 líneas de código. Por tanto, supondría que tenemos que escribir 48 líneas de código. Eso son muchas líneas de código, así que Ionic nos da la opción de crear un bucle para que las líneas se escriban de forma automática sin la necesidad de tener que escribir tanto código a mano. Pero antes de poder hacerlo debemos de realizar algunas configuraciones a nuestro proyecto.

### Añadiendo archivo de datos

Lo que vamos a hacer es generar un array que contendrá enum. Los enum (enumeraciones) sirven para presentar un grupo de constantes con un nombre. En este caso nuestras constantes serán los atributos de cada animal:

- Nombre.
- Imagen.
- Audio.
- Duración.
- Reproduciendo.

Al crear un enum que contenga los atributos de cada uno de los animales, podremos generar un bucle en la vista para evitar la creación de código y hacerlo más compacto, aunque algo menos sencillo de comprensión.

Así que vamos a crear un archivo con el nombre **data.animales.ts** en la ruta **src/app/data**. El archivo tendrá la siguiente forma:

```

export const ANIMALES = [
  {
    nombre: "Caballo",
    imagen: "assets/animales/caballo.png",
    audio: "assets/sonidos/caballo.mp3",
    duracion: 4,
    reproduciendo: false
  },
  {
    nombre: "Cabra",
    imagen: "assets/animales/cabra.png",

```

```
    audio: "assets/sonidos/cabra.wav",
    duracion: 4,
    reproduciendo: false
  },
  {
    nombre: "Cerdo",
    imagen: "assets/animales/cerdo.png",
    audio: "assets/sonidos/cerdo.wav",
    duracion: 2,
    reproduciendo: false
  },
  {
    nombre: "Gallo",
    imagen: "assets/animales/gallo.png",
    audio: "assets/sonidos/gallo.mp3",
    duracion: 4,
    reproduciendo: false
  },
  {
    nombre: "Mono",
    imagen: "assets/animales/mono.png",
    audio: "assets/sonidos/mono.mp3",
    duracion: 8,
    reproduciendo: false
  },
  {
    nombre: "Perro",
    imagen: "assets/animales/perro.png",
    audio: "assets/sonidos/perro.mp3",
    duracion: 5,
    reproduciendo: false
  },
  {
    nombre: "Serpiente",
    imagen: "assets/animales/serpiente.png",
    audio: "assets/sonidos/serpiente.mp3",
    duracion: 2,
    reproduciendo: false
  },
  {
    nombre: "Tigre",
    imagen: "assets/animales/tigre.png",
```

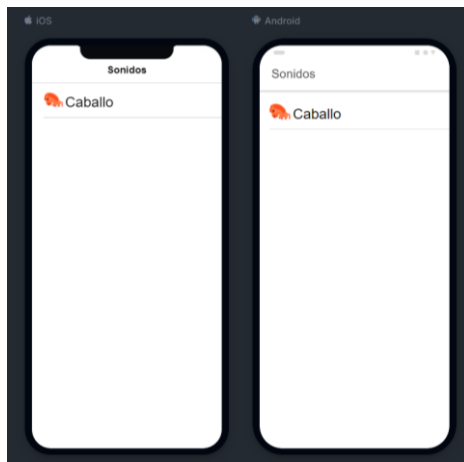
```

    audio: "assets/sonidos/tigre.mp3",
    duracion: 2,
    reproduciendo: false
  }
];

```

En este archivo hemos especificado las propiedades de los sonidos de cada uno de los animales que vamos a utilizar para la aplicación. Una vez creado este archivo ya podemos volver a la vista para generar un bucle que nos muestre todos los botones.

Ahora mismo, la apariencia de la aplicación debería ser la siguiente.



*Figura 3. Lista con un único elemento.*

Una vez creado el archivo de datos debemos importar el mismo a la página **home**. Para ello abriremos el archivo **home.page.ts** y añadiremos la siguiente línea debajo de la última importación.

```
import { ANIMALES } from 'src/data/data.animales';
```

Después de la importación hemos de definir una variable de entorno que la inicializaremos en blanco. Para ello añadiremos la siguiente línea debajo de **export class HomePage**.

```
animales:any[]=[];
```

Después cambiaremos la tipología de esta variable, una vez hayamos creado y definido la interfaz **Animal**.

El siguiente paso será la creación de una copia de ANIMALES en el constructor, para ello añadiremos al constructor las siguientes líneas.

```

constructor(public navCtrl:NavController) {
  this.animales=ANIMALES.slice(0);
}

```



Si nos hemos olvidado importa **NavController**, al insertarlo dentro del constructor se nos añadirá automáticamente la siguiente línea debajo del último **import**.

```
import { NavController } from '@ionic/angular';
```

Nuestro siguiente paso será la creación de una interfaz. Las interfaces nos permiten definir un tipo de dato y su estructura, de forma parecida a lo que haría una clase. Para crear la interfaz utilizaremos la línea de comandos y ejecutaremos el siguiente comando:

```
$ionic g interface ../interfaces/animal
```

Con este comando se nos creará la carpeta **interfaces** dentro de la carpeta **src** y contendrá el archivo **animal.ts** con el siguiente código:

```
export interface Animal {  
}
```

Ahora deberemos escribir lo siguiente entre las llaves:

```
nombre:string;  
imagen:string;  
audio:string;  
duracion:number;  
reproduciendo:boolean;
```

Tal y como puede observarse en el código, lo que estamos haciendo es decir qué tipo de datos es cada elemento del array que hemos definido en el archivo **data.animales.ts**.

Una vez creada la interface la importamos a **home.page.ts** añadiendo la siguiente línea debajo del último **import**.

```
import { Animal } from 'src/interfaces/animal';
```

Ahora ya podemos utilizar el tipo de dato **Animal** para definir la variable **animales**; por lo que cambiaremos su tipo de any a **Animal**.

```
animales:Animal[]=[];
```

En el archivo **home.page.ts** ya no hemos de realizar ningún cambio más por el momento. Lo siguiente que debemos hacer es implementar toda esta lógica en el archivo de la vista para poder realizar el bucle.

## 5. Creando un bucle en la vista

Para crear un bucle para que se nos creen los botones de todos los animales de forma automática sólo será necesario realizar algunos pequeños ajustes al listado que hemos mostrado anteriormente.

El listado con los bonotes tendrá la siguiente forma:

```

<ion-content [fullscreen]="true">
  <ion-list>
    <ion-item *ngFor="let animal of animales"
(click)="reproducir(animal)">
      <ion-avatar item-start>
        
      </ion-avatar>
      <h2>{{ animal.nombre }}</h2>
    </ion-item>
  </ion-list>
</ion-content>

```

El bucle lo creamos con el comando **\*ngFor** donde decimos que recorra el array de animales de elemento en elemento. Después para acceder a cada uno de los animales, para acceder a su propiedad hemos de poner el nombre del **elemento** (animal) un . (punto) y después la **propiedad** a la que se quiere acceder, tal y como puede verse en las etiquetas **<img>** y **<h2>**.

animal.imagen

Ahora mismo la aplicación debería tener la siguiente forma en su vista principal.

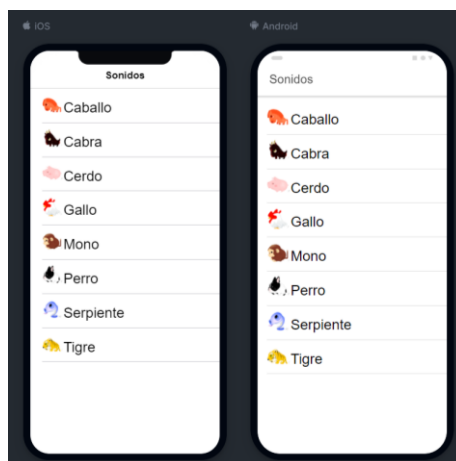


Figura 4. Lista con todos los animales.

## 6. Reproduciendo sonidos

El siguiente paso es la reproducción de sonidos para lo que crearemos la función **reproducir()**. Para empezar a implementar este método empezaremos realizando pruebas para detectar posibles errores durante la implementación.

Lo primero que haremos es que al pulsar el botón de reproducir nos aparezca el nombre del animal por consola. Para ello escribiremos debajo del constructor las siguientes líneas.

```

reproducir(animal:Animal){
  console.log(animal);
}

```

Si ahora vamos al navegador y consultamos la Consola veremos que cada vez que pulsemos sobre un elemento de la pantalla nos aparecerá escrito en consola las propiedades del elemento pulsado.

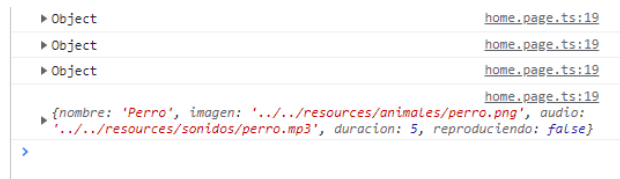


Figura 5. Vista en Consola del elemento pulsado.

Para la reproducción del sonido utilizaremos una de las herramientas que proporciona el lenguaje HTML5, así que añadiremos las siguientes líneas a la función **reproducir()**.

```
reproducir(animal:Animal){
  console.log(animal);

  let audio = new Audio();

  audio.src=animal.audio;
  animal.reproduciendo=true;
  audio.load();

  setTimeout(() => {
    animal.reproduciendo=false;
  },
  animal.duracion*100);
}
```

Lo que estamos haciendo en esta función ahora mismo es crear una variable local: **audio** y después accedemos a las propiedades de audio definidas en el fichero **data.animal.ts**. Lo que estamos haciendo con la llamada a la función **setTimeout()** es la acción que queremos que se realice al finalizar el ciclo de vida de la función.

## 7. Actualizando la vista

Ya hemos implementado la lógica para la reproducción de los sonidos, ahora modificaremos la vista para que aparezca un botón de play si el sonido no se está reproduciendo y uno de pausa si lo está haciendo. Para ello deberemos modificar el archivo **home.page.html** añadiendo las siguientes líneas dentro de la etiqueta **<ion-item>**.

```
<ion-item *ngFor="let animal of animales" (click)="reproducir(animal)">
  <ion-avatar item-start>
    
  </ion-avatar>
  <h2>{{ animal.nombre }}</h2>
  <ion-icon *ngIf="animal.reproduciendo" name="pause" item-end></ion-
  icon>
```

```
<ion-icon *ngIf="!animal.reproduciendo" name="play" item-end></ion-  
icon>  
</ion-item>
```

Ahora, si pulsamos sobre un elemento se reproduce el sonido del animal seleccionado y el botón de play que aparece al iniciar la aplicación se substituye por uno de pause mientras se reproduce el sonido.

#### 8. Mejorando la app

Si hemos pulsado más de un audio a la vez o el mismo audio varias veces seguidas habrá vista que se reproducen todos los sonidos de forma simultánea. Esto es un malfuncionamiento o un error de programación, ya que se debería detener el sonido en reproducción par a reproducirse el nuevo sonido.

El erro se debe a que estamos llamando varias veces a la función **reproducir()** y que ésta lanza su propio audio y controla su propio tiempo sin mirar si ya hay una función en ejecución.

La solución será hacer que el objeto audio y su tiempo de reproducción sean variables de instancia y no locales. Para ello debajo de la variable animales añadiremos las siguientes líneas.

```
audio = new Audio();  
tiempoAudio:any;
```

De esta forma se puede acceder a las variables desde cualquier función que se escriba en el fichero y no se crearán continuamente cuando se lance la función que las contiene.

Al haber elevado estas variables a variables de instancia (o globales), es necesario realizar una pequeña modificación en la función para indicar que la es una variable instancia. Esto se realiza añadiendo **this.** como prefijo en la variable, tal y como se muestra a continuación.

Ya que estamos modificando la función **reproduciendo()**, añadiremos un bucle **if** para saber cuándo se está reproduciendo un sonido y la llamada a dos funciones: **play()** y **pausarAudio()**. La función quedará de la siguiente manera.

```
reproducir(animal:Animal){  
  console.log(animal);  
  
  this.pausarAudio(animal);  
  
  if(animal.reproduciendo){  
    animal.reproduciendo=false;  
    return;  
  }  
  
  this.audio.src=animal.audio;  
  animal.reproduciendo=true;
```

```

this.audio.load();
this.audio.play();

setTimeout(() => {
  animal.reproduciendo=false;
},
animal.duracion*100);
}

```

El siguiente paso será implementar la función **pausarAudio()**, la cual tendrá la siguiente forma.

```

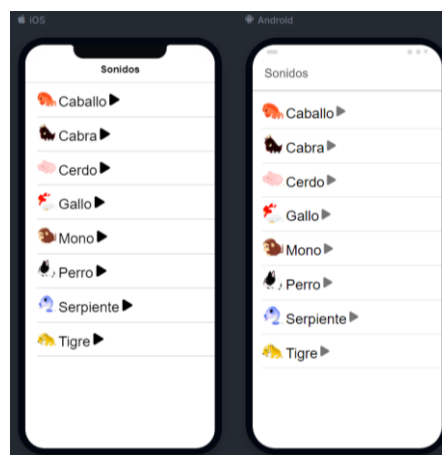
pausarAudio(animalseleccionado:Animal){
  clearTimeout(this.tiempoAudio);

  this.audio.pause();
  this.audio.currentTime=0;

  for(let animal of this.animales){
    if(animal.nombre != animalseleccionado.nombre){
      animal.reproduciendo=false;
    }
  }
}

```

Tras realizar esta actualización, el aspecto de nuestra aplicación debería ser el siguiente.



*Figura 6. Aspecto de la aplicación con los botones Play.*

Ahora mismo la aplicación sería podría utilizarse y podríamos pausar y reproducir sonido, y éstos no se solaparían unos con otros, ya que, al pulsar sobre un nuevo animal, pero existe un problema.

Aquellos usuarios de lectores de pantalla habrán observado que al deslizar el dedo en un dispositivo móvil nos vamos desplazando dentro de los elementos del botón. Es decir, el primer flick dice botón, el segundo dice imagen, el tercero el nombre del animal,... Esto no es accesible o si más no, es poco

accesible. Así que debemos arreglar ese problema para que el juego sea amigable para los usuarios de lectores de pantalla.

## Incluyendo la Accesibilidad

La accesibilidad ha de estar presente desde el inicio de la programación, es decir, es más fácil crear una aplicación accesible desde un inicio que hacer la aplicación y después integrar la accesibilidad, tal y como hemos hecho en este tutorial.

Aquí se ha hecho para mostrar los problemas que se generan a usuarios de lectores de pantalla si no se tiene en cuenta la accesibilidad desde antes de empezar a programar, es decir, desde el borrador de la aplicación, ya que la inversión de tiempo y la dificultad será mucho menor.

Afortunadamente, en nuestro caso sólo hemos de mejorar la accesibilidad de los botones. Vamos a ello.

Lo primero que vamos a hacer es modificar la vista, es decir, el archivo **home.page.html** y utilizar el elemento **<button>** para agrupar el contenido.

```
<ion-item *ngFor="let animal of animales">
  <ion-button (click)="reproducir(animal)">
    <span class="hidden">Reproducir</span>
    
    <h2>{{ animal.nombre }}</h2>
  </ion-button>
</ion-item>
```

Puede que llame la atención dos cosas del nuevo código. Así que vamos a explicar el porqué.

- **<span class="hidden">**. Esta etiqueta se utilizar para mostrar un texto que sólo será accesible para los lectores de pantalla.
- **alt=""**. El atributo alt se incluye para la descripción de la imagen. Así que deberíamos poner **{{ animal.nombre }}** ya que es el nombre del animal- El problema está en que si se pone ese texto el lector de pantalla nos enunciará dos veces el nombre del animal. Si se deja el atributo vacío estamos indicando al lector de pantalla que no lo lea.

Ahora mismo la aplicación debería mostrar el siguiente aspecto.

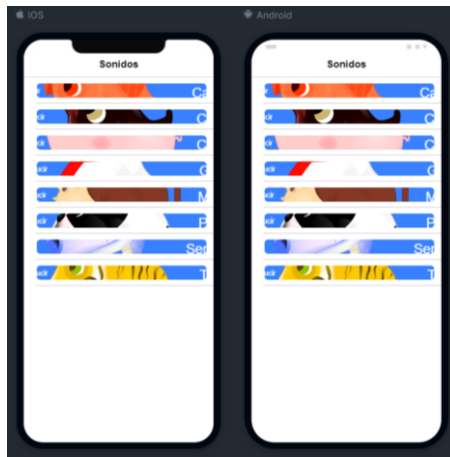


Figura 7. Vista de la aplicación.

La aplicación puede ser todo lo accesible que queramos, pero visualmente es un desastre. Nuestro siguiente paso será el trabajo con estilos para hacer que visualmente la aplicación sea más amigable.

### 9. Trabajando con estilos

Abramos el archivo **home.page.css**, que está en la ruta **src/app/**. Esta página contiene los estilos predefinidos para nuestra vista, sólo para la vista en la que estamos trabajando. Quiero incidir en él, porque si creamos un elemento idéntico en otra vista, este no se verá afectado por los diseños de esta vista, en caso de que los hayamos modificado. Una vez aclarado esto, borraremos el contenido del archivo ya que no nos es útil y lo reemplazaremos por uno más apropiado.

El mayor problema visual es que la altura de los botones es menor a la de las imágenes. Esto podemos arreglarlo creando el siguiente estilo.

```
ion-button{  
  height: auto;  
}
```

Al hacer esto veremos que un único botón nos ocupa toda la altura de la pantalla, así que debemos ajustar la altura de la imagen a una que sea adecuada para la pantalla. Así que hemos de crear un estilo para las imágenes; para ello escribiremos debajo del estilo anterior las siguientes líneas.

```
img{  
  height: 10vh;  
}
```

¿Qué quiere decir 10vh? Las siglas **vh** quieren decir Vertical Height e indican que el elemento ocupará el 10% de la altura del dispositivo en el que se muestre. Esta es la mejor forma de hacer que la parte visual de la aplicación sea responsive. Es preferible utilizar la nomenclatura **vh** a la **%** ya que el porcentaje (%) utiliza como tamaño absoluto el tamaño definido para el contenedor y no el del dispositivo.

Aclarado esto, si miramos el resultado en nuestra aplicación, será el siguiente.

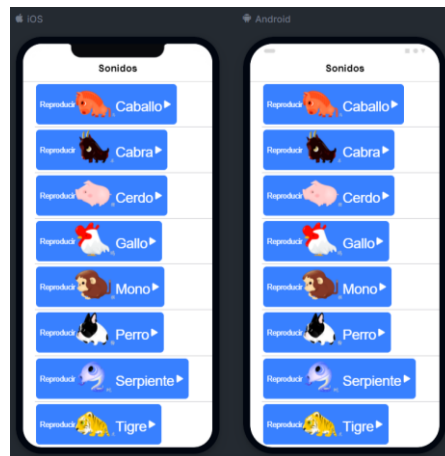


Figura 8. Vista accesible mejorada.

Ya tenemos bien definido la altura de las imágenes, pero aún existen problemas. El siguiente problema que vamos a resolver es la presencia del texto reproducir. Este texto debe ser accesible sólo para los lectores de pantalla y no debe de verse en la pantalla, así que vamos a crearle un estilo.

Para ello, añadiremos el siguiente estilo a nuestra hoja.

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

En este estilo lo que estamos haciendo es indicar a la parte visual de la aplicación que el texto debe aparecer en las coordenadas escritas. Estas coordenadas son muy inferiores al tamaño de cualquier pantalla que exista en la actualidad en el mercado, y, por lo tanto, nunca aparecerá en pantalla, pero sí que será visible para un lector de pantalla.

Nuestra vista va tomando forma. La verdad, es que ha mejorado mucho respecto a la apariencia inicial, pero aún tiene detalles a mejorar.

Nuestro siguiente paso va a ser homogeneizar la anchura de los botones, ya que ahora mismo no tienen todos la misma anchura. Para ello, como siempre, utilizaremos la hoja de estilos. Ya hemos creado un estilo para los botones, así que simplemente necesitamos retocarlo.

Una opción es hacer que el botón ocupe el 100% del ancho de la pantalla. Si queremos esta solución, la más fácil de aplicar, debemos modificar el estilo del botón por el siguiente.

```
ion-button{
  height: auto;
```



```
width: 100vw;
}
```

Otra opción, será establecer una anchura mínima del botón y centrarlo de horizontalmente en pantalla. En este caso, la anchura mínima la marcará el botón de la serpiente, al ser el más ancho de todos.

Si escogemos esta opción, el estilo para los botones será el siguiente.

```
ion-button{
  height: auto;
  width: 75vw;
  margin-left: 5vw;
  margin-right: 5vw;
}
```

Utilizando el último estilo para los botones, la aplicación debería de tener el siguiente aspecto.

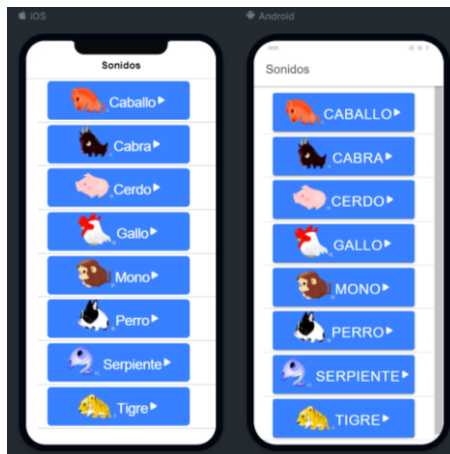


Figura 9. Aspecto de la aplicación con los nuevos estilos.

El último tema visual por arreglar es el botón del play, que no está muy bien alineado. A parte de esto, el nombre del icono es **play**, por lo que el lector de pantalla nos enunciará play al pasar sobre el botón. En este caso no es muy importante, ya que es una palabra de uso común. Si no lo fuera, deberíamos buscar un sistema similar al utilizado para insertar las imágenes.

En mi caso, he optado por eliminar el icono, ya que entiendo que el nombre de la aplicación es suficientemente claro como para indicar lo que produce un sonido.

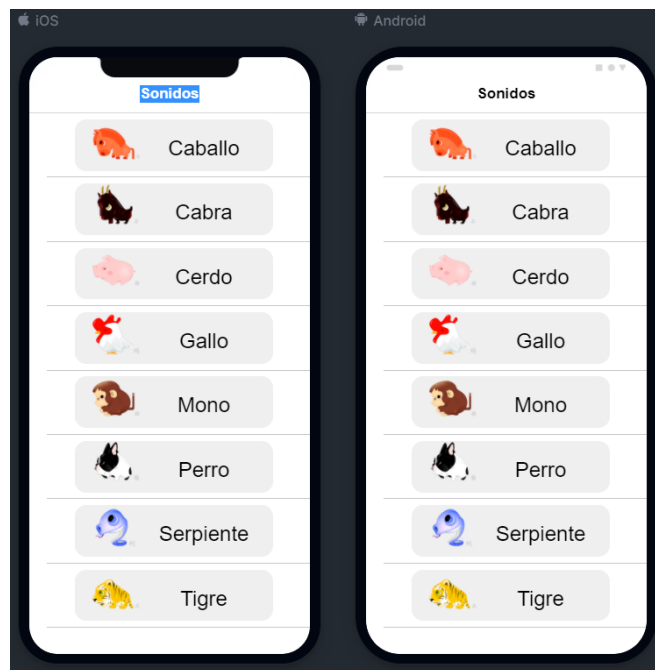
Ahora sí que visualmente es una aplicación amigable. Aún podemos realizar multitud de modificaciones, por ejemplo, cambiar el color del fondo del botón. Si queremos realizarlo podemos utilizar las propiedades de Ionic y dentro de la etiqueta **<button>** añadir el atributo color.

Las opciones para este atributo son:

- primary.

- secondary.
- tertiary.
- success.
- warning.
- danger.
- light.
- medium.
- dark.

Yo, después de jugar con la hoja de estilos y cambiando la etiqueta `<ion-button>` por `<button>` he dado el siguiente aspecto a la aplicación final.



*Figura 10. Aspecto final de la aplicación.*

## 10. Mejorando la accesibilidad

Aunque el lector de pantalla no nos enunciará nada para las imágenes, sí que las detecta. Por lo que la solución propuesta, pese a que es sencilla, no es la correcta desde el punto de vista de la accesibilidad.

Si queremos que las imágenes no sean detectadas por el lector de pantalla, lo que debemos hacer es insertar las mismas desde la hoja de estilos. El problema de insertar las imágenes de esta aplicación desde CSS es que no podemos utilizar el bucle for para generar el listado, es decir se han de introducir los elementos uno a uno con lo que se nos multiplican las líneas de código.

En caso de que alguien quiera ver el código de esta mejora de accesibilidad, pues descargarla desde el repositorio GitHub.

## Capítulo II: Adivina el Animal

### 1. Objetivo

En este capítulo vamos a modificar la ampliación anterior para que al pulsar el botón se reproduzca un sonido y el usuario tenga que adivinar el nombre del animal. Para ello utilizaremos el mismo material que en la aplicación anterior.

En este capítulo veremos lo siguiente:

- Creación de una nueva página.
- Creación de formularios.
- Creación de **enum** (enumeraciones).
- Validación de formularios.
- **Arrays**.
- Bucles **for**, **if** y **switch**.

### 2. Creando una nueva página

Lo primero que vamos a necesitar es la creación de una nueva página para nuestro proyecto, así que vamos a nuestra terminal o línea de comandos y escribimos el siguiente comando.

```
$ionic g page adivinaSonido
```

### 3. Creación de formularios

Lo primero que vamos a hacer es la creación de un formulario que replicaremos para la obtención de los diferentes animales.

Para crear el formulario el primer paso es la importación de **ReactiveFormsModule** en los módulos de la página. Estos módulos se encuentran en **adivina-sonido.module.ts**. Abrimos el fichero y añadimos la siguiente línea después de la importación de **CommonModule**.

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

A parte de esto deberemos incluir el módulo dentro del array de **imports**, el cual nos quedará de la siguiente manera.

```
imports: [  
  CommonModule,  
  FormsModule,  
  ReactiveFormsModule,  
  IonicModule,  
  AdivinaSonidoPageRoutingModule  
],
```

Ahora ya podemos abrir el fichero **adivina-sonido-page.ts** e importar la clase **FormControl**.

```
import { FormControl } from '@angular/forms';
```

El siguiente paso será crear una instancia para un campo del formulario. Para ello, en el mismo lugar donde creamos las variables, añadiremos la siguiente línea.

```
animal_desc = new FormControl('');
```

En este caso se para una cadena vacía; en caso de que queramos pasar un valor de inicio a **FormControl()** se hará en el **constructor**. El siguiente paso es la utilización del formulario en la vista, así que abriremos el fichero **adivina-sonid.page.html** e insertaremos las siguientes líneas.

```
<ion-item>
  <ion-label>
    Nombre del Animal
  <ion-input [formControl]="nombre"></ion-input>
</ion-label>
</ion-item>
```

Si vamos a la vista, veremos que se nos ha creado una entrada al formulario donde deberemos introducir el nombre del animal, pero aún no tenemos un botón para enviar los datos, Así que vamos a crear uno añadiendo el siguiente código justo debajo de **<ion-item>**.

```
<ion-button (click)="answer()">Respuesta</ion-button>
```

Ahora ya tenemos creado el botón con el que se recogerá la respuesta introducida y que lanzará la función **answer()**.

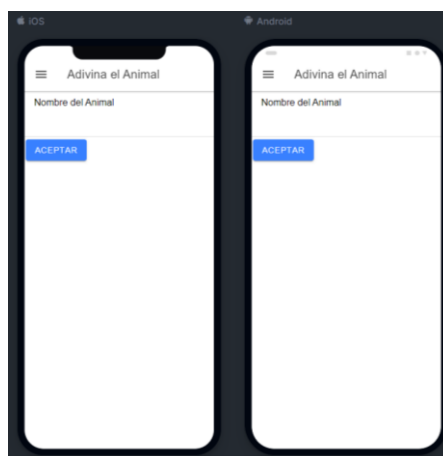


Figura 11. Aspecto inicial del formulario.

Antes de ponernos a crear esta función, veamos cómo mejorar nuestro formulario. Lo primero que podemos hacer es una validación de datos,

#### 4. Validando datos

Para validar los datos introducidos en el campo debemos importar antes el módulo **Validators**. Para ello volveremos a abrir el fichero **adivina-**

**sonido.page.ts** y en la línea donde hemos importado el control de formularios importamos **Validators**. La línea quedará de la siguiente manera.

```
import { FormControl, Validators } from '@angular/forms';
```

Con este módulo podemos validar los datos introducidos en el campo para comprobar que se ha insertado una palabra.

Tras importar el módulo, debemos modificar la llamada a **FormControl** para validar los datos. Por ello, se añadirá lo siguiente.

```
nombre = new FormControl('', [Validators.required,  
Validators.minLength(4)]);
```

En esta línea se está indicando que el campo nombre donde se debe insertar el nombre del animal es obligatorio (**required**) y que la longitud mínima es de 4 caracteres.

Con el validar activado podemos hacer que el botón para enviar la respuesta sólo esté activo cuando se ha rellenado el campo de forma correcta, cumpliendo las restricciones. Para ello hemos de modificar el archivo **adivina-sonido.page.html** y modificar el botón.

```
<ion-button type="submit" [disabled]="!nombre.valid">Respuesta</ion-  
button>
```

Con esta modificación hacemos que el botón se active cuando el campo sea válido. Si ahora ejecutamos la aplicación veremos que el botón permanece deshabilitado hasta que en el campo se introducen 4 caracteres.

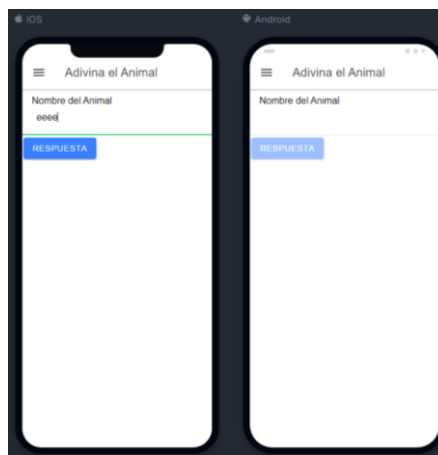


Figura 12. Vista con el botón activado y desactivado.

Ya que estamos con el botón de aceptar, podemos incluir a su lado un botón para borrar el texto introducido. La línea que debemos insertar es la siguiente.

```
<ion-button type="reset" color="danger"  
>Borrar</ion-button>
```

Como podemos ver, es muy parecido a la insertada para el botón aceptar.

El siguiente paso sería modificar la plantilla para que nos aparezca un mensaje de error cuando no se cumpla la validación de datos. Para ello se debe de modificar el formulario según.

```
<ion-label>
  Nombre del Animal
  <ion-input [formControl]="nombre"></ion-input>
</ion-label>
<ion-label color="danger" *ngIf="nombre.errors?.required &&
(nombre.touched || nombre.dirty)">*El campo es obligarotio</ion-label>
<ion-label color="danger" *ngIf="nombre.errors?.minlength &&
(nombre.touched || nombre.dirty)">*El campo debe tener al menos 4
caracteres</ion-label>
```

Al campo se le han añadido dos componentes **<ion-label>**. En el primero se muestra un mensaje de advertencia sobre que el campo es obligatorio, y en el segundo se indica que el campo debe tener al menos 4 caracteres de longitud.

El siguiente paso será la inclusión de un botón para reproducir el sonido de un animal al hacer.

## 5. Incluyendo un botón

Vamos a incluir en la parte superior un botón que al ser pulsado nos reproduzca el sonido de uno de los animales de la pantalla anterior. Para ello abriremos el archivo **adivina-sonido.page.html** y escribiremos la siguiente línea justo antes de la etiqueta **<form>**.

```
<button (click)="start()" id="play">Reproducir</button>
```

Desde este botón se está haciendo una llamada a una función **start()** que vamos a implementar a continuación. Para ello abriremos el fichero **adivina-sonido.page.ts** e insertaremos las siguientes líneas.

En la zona donde declaramos las variables añadiremos justo debajo de **nombre**.

```
randomAnimal = this.randomNum(1,8);
```

Esta variable llama a una función que nos generará un número aleatorio entre 1 y 8 para cargar el sonido del animal que ocupe esa posición.

La función tendrá la siguiente forma.

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

Ahora vamos a declarar la función **start()** a la que llamamos al pulsar sobre el botón **Reproducir**. Esta función tendrá la siguiente forma.

```
start(){
  let animalSound = this.getSound(this.randomAnimal);
  this.reproducir(animalSound);
}
```

Como puede verse, desde esta función se está llamando a una función **getSound()**, a la que le estamos pasando como variable el número aleatorio generado anteriormente. Esta función nos devuelve una cadena con la dirección del sonido que se reproducirá utilizando la función **reproducir()**. Así que vamos a ver cómo es la función reproducir.

```
reproducir(c){
  this.audio = new Audio(c);
  this.audio.load();
  this.audio.play();
}
```

Esta función es una pequeña modificación de la función vista para la pantalla donde se reproducen los sonidos de los animales al pulsar un botón, por lo que nos vamos a explicarla. Sólo hay que añadir que se ha de crear una nueva variable llamada **audio** donde hemos creado las otras dos variables y que debe ser del tipo **any**.

La función **getSound()** tiene el siguiente aspecto.

```
getSound(c){
  return Sounds[c];
}
```

Esta función toma la variable pasada, en este caso, un número y devuelve la cadena que ocupa esa posición en un **enum**, en este caso la dirección del sonido a reproducir.

Así que nuestro siguiente paso es la creación e importación de un **enum**.

## 6. Creando un enum

Para crear un enum abriremos nuestro Terminal o Línea de Comandos, nos situaremos dentro de la carpeta del proyecto y ejecutaremos el siguiente comando.

*\$ionic g enum enums/sounds*

Al ejecutarse este comando, Ionic nos generará una nueva carpeta denominada **enum** y dentro de la misma, nos generará un el archivo **sounds.ts**. Abramos este archivo para ver su contenido.

Como podemos apreciar, es prácticamente un archivo en blanco. A este archivo vamos a añadirle todas las direcciones de los ficheros de sonido; con lo que el archivo debe quedar de la siguiente manera.

```
export enum Sounds {  
  "assets/sonidos/caballo.mp3" = 1,  
  "assets/sonidos/cabra.wav",  
  "assets/sonidos/cerdo.wav",  
  "assets/sonidos/gallo.mp3",  
  "assets/sonidos/mono.mp3",  
  "assets/sonidos/perro.mp3",  
  "assets/sonidos/serpiente.mp3",  
  "assets/sonidos/tigre.mp3",  
}
```

Se ha añadido un 1 a la primera línea del **enum** para indicar que se empezará a contar por el número 1. Las siguientes líneas serán autonumeradas, a no ser que decidamos numerarlas de alguna forma especial.

Ya que estamos creando **enum** vamos a crear otro para la imagen del animal que denominaremos **images**.

En este caso, el código del fichero será el siguiente.

```
export enum Images {  
  "../../assets/animales/caballo.png" = 1,  
  "../../assets/animales/cabra.png",  
  "../../assets/animales/cerdo.png",  
  "../../assets/animales/gallo.png",  
  "../../assets/animales/mono.png",  
  "../../assets/animales/perro.png",  
  "../../assets/animales/serpiente.png",  
  "../../assets/animales/tigre.png",  
}
```

Con los **enum** ya creados, el siguiente paso será la importación a nuestra vista, para ello abriremos el fichero **adivina-sonido.page.ts** y escribiremos la siguiente línea en la zona de **import**.

```
import { Sounds } from '../enum/sounds';
```

Ya que estamos, también importaremos el de las imágenes, que necesitaremos más adelante.

```
import { Images } from '../enum/images';
```

Si ahora ejecutamos la ampliación y pulsamos sobre el botón reproducir se nos reproducirá de forma aleatoria el sonido de un animal que es el que deberíamos insertar en el campo de edición para acertarlo, pero antes de empezar a programar la función **answer()**, vamos a crear otro botón que permita la aparición de una pista sobre el animal.



## 7. Creando la primera pista

Esta primera pista que vamos a crear será una frase relacionada con el animal a descubrir por el usuario. Para ello crearemos un **enum** con el nombre de **sentence**.

*\$ionic g enum enums/sentence*

A la hora de escribir la frase que servirá de pista para descubrir el animal debemos de conservar el orden utilizado en el **enum sounds**, es decir, en **sounds** la primera línea era la dirección del sonido de un caballo, por lo que la pista que debemos para el caballo debe ocupar el primer lugar.

A mí se me han ocurrido las siguientes pistas, pero si tienes otras mejores puedes cambiarlas.

```
export enum Sentence {
  "Corro carreras y salto vallas" = 1,
  "Si haces tonterías dirán que estoy como yo, y dicen que siempre tiro al monte.",
  "En la pocilga habito y en barro me baño.",
  "Cada mañana al salir el sol te despierto.",
  "Plátanos como y amigo de Tarzán soy.",
  "Puedo ser tu mejor amigo.",
  "Ojo que pico y veneno puedo tener.",
  "De la India soy el rey y me has de temer.",
}
```

El siguiente paso será la importación del **enum** a nuestro vista, para eso añadiremos.

```
import { Sentence } from '../enum/sentence';
```

Al fichero **adivina-sonido.page.ts**.

## 8. Añadiendo un botón

Ahora, volvamos al fichero **adivina-sonido.page.html** y añadamos la parte visual. Aquí vamos a añadir un botón que llame a una función para que se nos muestre la frase que debe servirnos como pista. Para ello, añadiremos las siguientes líneas justo debajo del botón para reproducir el sonido.

```
<button (click)="askClue1()">Mostrar pista</button>
  <div *ngIf="clue1">
    <p>{{ animaPhrase }}</p>
  </div>
```

Tal y como puede observarse, se ha generado una nueva caja **<div>** que se muestra cuando **clue1** es cierto (al pulsarse sobre el botón **Mostrar pista**). Ahora, vamos al fichero **adivina-sonido.page.ts** y creamos la función **askClue1()**.

La función es un sencillo **if** donde **clue1** irá tomando valores de **false** o **true**. De esta forma podremos mostrar y ocultar la pista cada vez que se pulse el botón.

```
askClue1(){
    if(this.clue1){
        this.clue1 = false;
    }
    else{
        this.clue1=true;
    }
    this.count = 1;

    this.animaPhrase = this.getSentence(this.randomAnimal);
}
```

Como se puede ver, en la función estamos utilizando dos nuevas variables: **clue1**, **animalPhrase** y **count** las cuales debemos declarar en la zona de variables de la siguiente manera.

```
clue1=false;
animaPhrase:string;
count:number;
```

También tenemos una llamada a la función **getSentence()**, que es muy parecida a la función **getSound()** que ya hemos utilizado anteriormente.

```
getSentence(c){
    return Sentence[c];
}
```

El resultado de esta función se asigna a la variable **animalPhrase** que es la que se muestra en la vista.

Si ejecutamos nuestro proyecto veremos lo siguiente cuando pulsamos sobre el botón **Mostrar pista**.

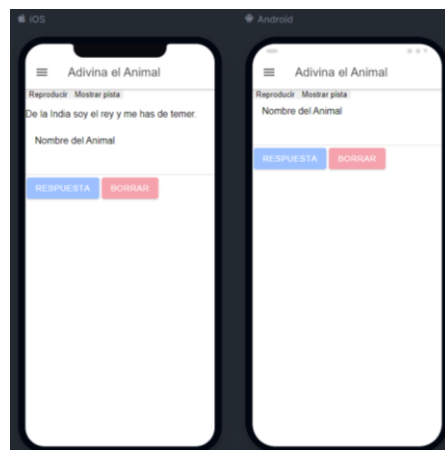


Figura 13. Vista con la primera pista desplegada.

Lo siguiente que vamos a hacer es añadir una segunda pista visual para que al pulsar sobre un segundo botón se nos muestre parcialmente una imagen del animal que nos ayuda a adivinar el animal escondido.

### 9. Añadiendo una segunda pista

Dentro de la caja de la pista número 1 vamos a añadir una nueva caja a la que podremos acceder tras consultar la pista 1 si se quieren más pistas para poder descubrir al animal oculto, para ello añadiremos las siguientes líneas al archivo **adivina-sonido.page.gtml** justo debajo de la etiqueta **<p>**.

```
<button (click)="askClue2()">Mostrar pista</button>
  <div *ngIf="clue2">
    
  </div>
```

El código es muy similar al de la pista anterior. Puede llamar la atención la presencia de **alt={{ animaAlt }}**. **Alt** nos permite dar una descripción de la imagen y es accesible para los lectores de pantalla. Al dar como descripción el contenido en **alt**, lo que vamos a hacer es pasar a este atributo el nombre del animal omitiendo algunas letras como podrían ser las vocales o cada cierto espacio.

Tras incluir este código, ahora, al pulsar sobre el botón de la pista 1 nos debería aparecer debajo de la pista un nuevo botón para poder activar la pista número 2.

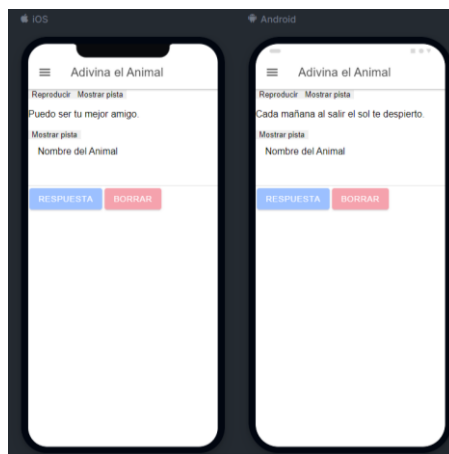


Figura 14. Vista con el botón para la segunda pista.

El siguiente paso es la creación de la función **askClue2()** a la que llama el botón recién creado. Así pues, abrimos el fichero **adivina-sonido.page.ts** y lo primero que hacemos es crear tres variables: **clue2**, **animaImg** y **animaAlt**.

Estas tres nuevas variables las insertaremos justo encima de la variable **count**.

```
clue2=false;
animaImg:any;
animaAlt:string;
```

Tras añadir las dos nuevas variables, vamos a añadir la nueva función **askClue2()** justo debajo de la **askClue1()**.

```
askClue2(){
  if(this.clue2){
    this.clue2 = false;
  }
  else{
    this.clue2=true;
  }
  this.count = 2;

  this.animaImg = this.getImg(this.randomAnimal);
  this.amimaAlt = this.getAlt(this.randomAnimal);
}
```

Si comparamos esta nueva función con **askClue1()** podemos ver que hay una mínima variación entre una y otra, ya que realmente lo único que varía es el nombre de la variable y de la función **get** a la que se llama, en este caso **getImg()** y a la función **getAlt()**.

La primera función nos devolverá una cadena con la dirección de la imagen y la segunda el texto alternativo que asociaremos a la imagen. Las funciones las insertaremos debajo de **getSentence()** y tendrá la siguiente forma.

```
getImg(c){
  return Images[c];
}

getAlt(c){
  return altTxt[c];
}
```

El siguiente paso que hemos de dar es crear el nuevo **enum altTxt** desde la Línea de Comandos o la Terminal y rellenar el mismo.

Lo que yo he hecho ha sido substituir las vocales de cada animal por \* (asterisco), pero si se tiene una idea mejor, no dudéis en ejecutarla. Mi **enum animaTxt** ha quedado de la siguiente manera.

```
export enum AnimaTxt {
  "c*b*ll*" = 1,
  "c*br*",
  "c*rd*",
  "g*ll*",
  "m*n*",
  "p*rr*",
  "s*rp**nt*",
  "t*gr*",
}
```

}

Ahora sólo nos queda importar este nuevo **enum** a **adivina-sonido.page.ts**.

Si ahora ejecutamos el proyecto veremos que se muestra completamente la imagen del animal, cuando nuestra intención es que se muestre una parte de este, pero no todo.

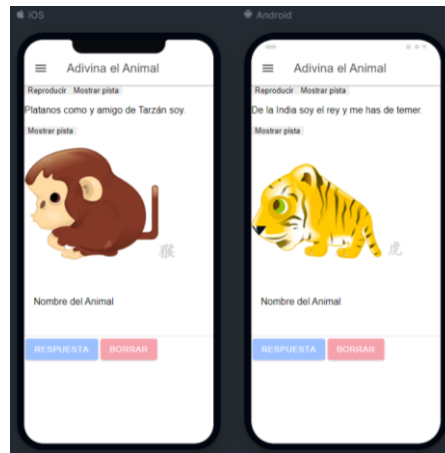


Figura 15. Vista de la segunda pista.

## 10. Ocultando las imágenes

Para semiocultar la imagen del animal vamos a crear una capa que contenga en su interior a la imagen y le daremos una altura máxima. El código del fichero **adivina-sonido.page.html** de la pista 2 debe quedar de la siguiente manera.

```
<button (click)="askClue2()">Mostrar pista</button>

<div *ngIf="clue2">
  <div id="img-content">
    
  </div>
</div>
```

Si ejecutamos el proyecto no notaremos ningún cambio, lo que debemos hacer es dar unas dimensiones a la caja contenedora para que sólo permita ver una parte de la imagen.

Esto debe hacerse desde los estilos, es decir, en el fichero **adivina-sonido.page.css**, que abramos dicho fichero y añadamos el siguiente código.

```
img{
  position: relative;
  top: -12vh;
  z-index: -999;
}

#img-content{
  height: 5vh;
```

```
overflow: hidden;  
}
```

Lo que estamos haciendo en estas líneas es dar una altura (**height**) a la caja contenedora de la imagen y diciendo que el contenido que sobresalga de la capa no se muestre (**overflow**). A la capa de la imagen lo que hacemos es desplazarla hacia arriba para que se muestre la parte central del animal, ya que si no se nos mostraría la parte superior del dibujo. Si ahora ejecutamos el proyecto y desplegamos la segunda pista obtendremos el siguiente resultado.

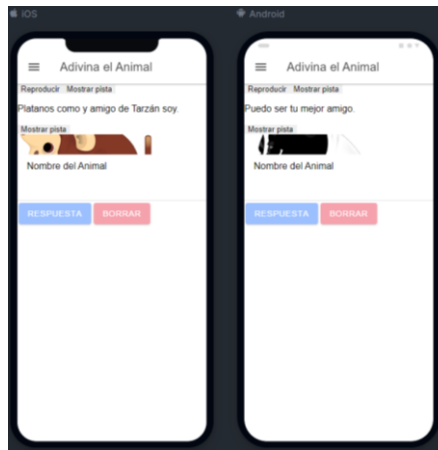


Figura 16. Vista parcial del animal.

Bueno, esto ya va tomando su forma definitiva, ahora sólo nos falta:

- Activar la validación de lo introducido en el campo de edición.
- Lógica del botón **Respuesta**.
- Mejorar visualmente la apariencia.

### 11. Validando la respuesta

Ahora mismo, si introducimos un número de cuatro dígitos en el campo de edición la aplicación nos dirá que el valor introducido es incorrecto y que debe introducirse una cadena alfabética y no numérica, así que vamos a ello.

Para crear ese validador de datos sólo debemos añadir un patrón de validación a la variable nombre, la cual deberá quedar de la siguiente manera.

```
nombre = new FormControl('', [Validators.required,  
Validators.minLength(4), Validators.pattern('a-zA-Z | *')]);
```

Una vez realizada esta modificación, debemos realizar una pequeña modificación en el elemento **<input>** del fichero **adivina-sonido.page.ts** para que sólo se active el botón de **Respuesta** si los valores introducidos son correctos, es decir que sólo se hayan insertado letras.

## 12. Añadiendo la lógica a Respuesta

Ahora ya sólo nos queda añadir la función que leerá la respuesta del usuario y mirará si es correcta o no. Para ello abrimos el fichero **adivina-sonido.page.ts** y creamos la función **answer()**.

Para la respuesta deberemos crear antes de todo un nuevo **enum** con la solución a la pregunta. No se va a explicar de nuevo como se ha creado el enum. En nuestro caso lo hemos llamado **Animals**.

Una vez se ha creado este nuevo archivo y rellenado con el nombre de los animales, en orden, lo importaremos a nuestro fichero **adivina-sonido.page.ts** para poder utilizarlo en la vista. A parte de este nuevo elemento necesitamos definir cinco nuevas variables.

```
msg:string;
msgSol:string;
score:number;
tries:boolean=false;
counter:number=0;
```

Con estas variables ya definidas vamos a implementar la función **answer()**.

```
answer(){
  ++this.counter;
  this.tries=true;

  if(this.counter<=5){
    if(this.animal===this.getAnimal(this.randomAnimal)){
      if(this.count==0){
        this.animalImg = this.getImg(this.randomAnimal);
        this.animalAlt = this.getAnimal(this.randomAnimal);
        this.clue3=true;
      }
      else if(this.count==1){
        this.animalImg = this.getImg(this.randomAnimal);
        this.animalAlt = this.getAnimal(this.randomAnimal);
        this.clue3=true;
      }
      else if(this.count==2){
        this.animalImg = this.getImg(this.randomAnimal);
        this.animalAlt = this.getAnimal(this.randomAnimal);
        this.clue3=true;
      }
    }
  }
  else{
    this.msg="Ya no tienes más intentos.";
    this.clue3=true;
    this.askClue3();
    this.score=0;
  }
}
```

```
}  
}
```

La función definida en las líneas superiores es el esqueleto de lo que va a ser nuestra función **answer()** definitiva, ya que vamos a implementar una puntuación al juego, para lo que ya hemos definido las variables.

Se ha decidido implementar un esquema de puntuación para hacer algo más divertido el juego.

### 13. Implementando la puntuación

Vamos a utilizar las variables **counter** y **count** para implementar un sistema de puntuación mediante bucles **switch**. Si hacemos las modificaciones, la función **answer()** nos debería quedar de la siguiente manera.

```
answer(){  
    ++this.counter;  
    this.tries=true;  
  
    if(this.counter<=5){  
        if(this.animal===this.getAnimal(this.randomAnimal)){  
            if(this.count==0){  
                this.animalImg = this.getImg(this.randomAnimal);  
                this.animalAlt = this.getAnimal(this.randomAnimal);  
                this.clue3=true;  
                switch(this.counter){  
                    case 1:  
                        this.score=15;  
                        this.msg="¡Puntuación Perfecta! No te ha hecho falta  
ninguna pista para adivinar que el animal escondido es ";  
                        break;  
                    case 2:  
                        this.score=14;  
                        this.msg="¡Puntuación casi Perfecta! No te ha hecho falta  
ninguna pista para adivinar el animal escondido, aunque sí más de un  
intento. El animal escondido es ";  
                        break;  
                    case 3:  
                        this.msg="¡A la tercera va la vencida! No te ha hecho falta  
ninguna pista para adivinar el animal escondido, aunque sí tres intentos  
para adivinar que el animal escondido es ";  
                        this.score=13;  
                        break;  
                    case 4:  
                        this.msg="¡A la cuarta! No te ha hecho falta ninguna pista  
para adivinar el animal escondido, aunque sí cuatro intentos para  
adivinar que el animal escondido es ";  
                        this.score=12;
```



```

        break;
    case 5:
        this.msg="¡Por poco! No te ha hecho falta ninguna pista
para adivinar el animal escondido, aunque deberías haberlas pedido. El
animal escondido es ";
        this.score=11;
        break;
    }
}
else if(this.count==1){
    this.animalImg = this.getImg(this.randomAnimal);
    this.animalAlt = this.getAnimal(this.randomAnimal);
    this.clue3=true;

    switch(this.counter){
        case 1:
            this.msg="¡No está mal! Te ha hecho falta una pequeña pista
para adivinar que el animal escondido es ";
            this.score=10;
            break;
        case 2:
            this.msg="¡A la segunda! Te ha hecho falta una pequeña
pista y dos intentos para adivinar que el animal escondido es ";
            this.score=9;
            break;
        case 3:
            this.msg="¡A la tercera va la vencida! Te ha hecho falta
una pequeña pista y tres intentos para adivinar que el animal escondido
es ";
            this.score=8;
            break;
        case 4:
            this.msg="¡Te has arriesgado! Te ha hecho falta una pequeña
y cuatro pistas pista para adivinar que el animal escondido es ";
            this.score=7;
            break;
        case 5:
            this.msg="¡Sobre el alero! Te ha hecho falta una pequeña
pista, ¿por qué no has pedido alguna? El animal escondido es ";
            this.score=6;
            break;
    }
}
else if(this.count==2){
    this.animalImg = this.getImg(this.randomAnimal);
    this.animalAlt = this.getAnimal(this.randomAnimal);
    this.clue3=true;

    switch(this.counter){

```

```

        case 1:
            this.msg="¡Hemos de afinar más el oído! Te han hecho falta
las dos pistas para adivinar que el animal escondido es ";
            this.score=5;
            break;
        case 2:
            this.msg="¡Los dos patitos! Te han hecho falta las dos
pistas y dos intentos para adivinar que el animal escondido es ";
            this.score=4;
            break;
        case 3:
            this.msg="¡Debes mejorar! Te han hecho falta las dos pistas
y tres intentos para adivinar que el animal escondido es ";
            this.score=3;
            break;
        case 4:
            this.msg="¡Aix...! Te han hecho falta las dos pistas para
adivinar y cuatro intentos que el animal escondido es ";
            this.score=2;
            break;
        case 5:
            this.msg="¡Te gusta el riesgo! Por poco, pero lo has
logrado ¡Así me gusta! El animal escondido es ";
            this.score=1;
            break;
    }
}
}
}
else{
    this.msg="Ya no tienes más intentos.";
    this.clue3=true;
    this.askClue3();
    this.score=0;
}
}
}

```

Con esto ya tendríamos la lógica del botón **Respuesta** finalizada, ahora debemos implementar las respuestas en la parte visual de la aplicación, así que vamos a ello.

Abrimos el fichero **adivina-sonido.page.html** e insertamos las siguientes líneas para mostrar el número de intentos, justo debajo de la etiqueta de cierre del formulario **</form>**.

```

<ion-card *ngIf="!clue3 && tries">
    <p>{{ msg }}</p>
    <p>Número de intentos: {{ counter }}</p>
</ion-card>

```

Debajo de estas líneas insertamos las respuestas de la aplicación para la finalización del juego que serán las siguientes.

```
<ion-card *ngIf="clue3">
  <h2>Solución</h2>
  <p>Tu puntuación ha sido: {{score }}</p>
  <p>{{ msg }}: <b>{{ animalAlt }}</b></p>
  
</ion-card>
```

Con esto tendríamos resuelto prácticamente todo el juego, a excepción de un botón para reiniciar.

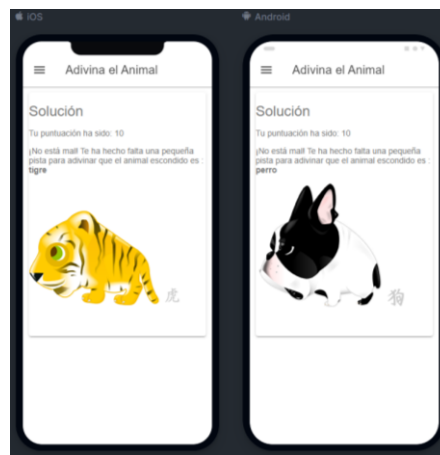


Figura 17. Vista de la finalización del juego.

#### 14. Insertando un botón reiniciar

Nuestro siguiente paso va a ser la inclusión de un botón que permita reiniciar el juego, en caso de que queramos, cuando éste finalice. Para ello vamos a insertar la siguiente línea en el fichero **adivina-sonido.page.html** justo antes del cierre de **</ion-card>**.

```
<ion-button (click)="restart()">Volver a jugar</ion-button>
```

Una vez insertado este código, vamos a crear la función **restart()** en el fichero **adivina-sonido.page.ts**. Como vamos a ver, esta función es muy sencilla y lo único que realiza es la reinicialización de las variables del juego para crear un nuevo animal y borrar todas las respuestas.

La función tiene la siguiente forma.

```
restart(){
  this.animal=null;
  this.randomAnimal = this.randomNum(1,8);
  this.clue1=false;
  this.clue2=false;
  this.count=0;
  this.clue3=false;
  this.tries=false;
```

```

this.counter=0;
this.msg=null;
this.score=0;
}

```

Con esto tendríamos el juego prácticamente finalizado, sólo nos queda por mejorar o retocar el aspecto visual de la pantalla para dar más visibilidad a los botones, centrar el contenido, modificar el tamaño de la letra, etc.

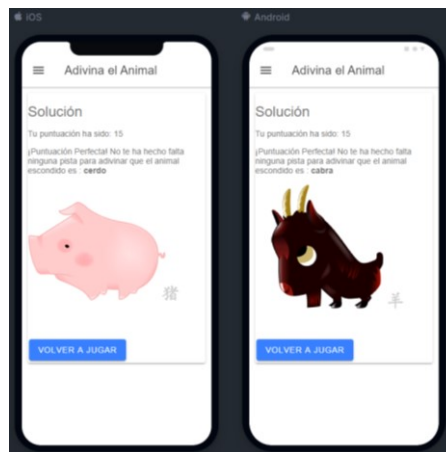


Figura 18. Aspecto final con el botón de reinicio.

## 15. Mejorando la visibilidad

Para mejorar el aspecto visual de la aplicación deberemos realizar los cambios en el fichero **adivina-sonido.page.scss**.

En mi caso, lo que he hecho ha sido aumentar un poco la separación entre los elementos para que respiren, centrar los elementos en la pantalla y dar formato a los botones.

Después de unas breves modificaciones de los estilos, el resultado ha sido el que se muestra a continuación.

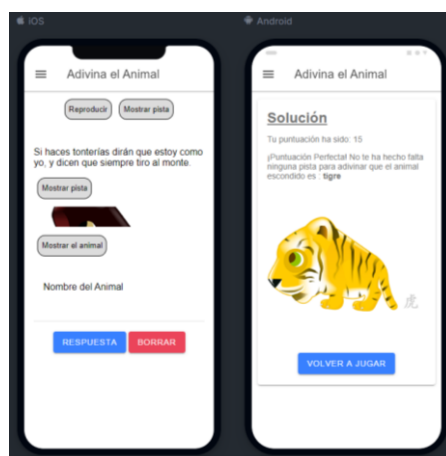


Figura 19. Aspecto final de la aplicación.

## 16. Pequeñas mejoras

Existen pequeñas mejoras a realizar en la ampliación como sería un apartado de configuración donde poder modificar la apariencia visual, o hacerla multilinguaje, pero la pequeña reforma que se va a mostrar es la resolución de un posible error que podría aparecer al ejecutar la aplicación.

El error es que, al pulsarse el botón de **Volver a jugar**, se vuelve a calcular un número aleatorio sin tener en cuenta el número anterior. Esto podría hacer que nos apareciera consecutivamente el mismo animal. Para evitar esto podemos crear un array donde almacenaríamos los animales que ya han salido, para evitar la repetición, y que volviera a cero una vez hayan salido todos los animales.

Para hacer esto necesitaremos crear una nueva variable, el array de almacenamiento, ya que el número de animales dentro del array podemos obtenerlo calculando la dimensión de este.

```
memoryArray:number[];
```

El siguiente paso será una mínima modificación de la función **randomNum()**, que quedará de la siguiente manera.

```
randomNum(a,b){
  let p=Math.round(Math.random() * (b-a) + parseInt(a,10));

  this.checkRandom(p);

  return p;
}
```

Lo que estamos haciendo es llamar a la función **checkRandom()** antes de devolver el valor.

Vamos a ver cómo debe ser esta función.

```
checkRandom(p){
  for(let i in this.memoryArray){
    if(p==this.memoryArray[i]){
      this.randomNum(1,8);
      break;
    }
    else{
      this.memoryArray.push(p);
    }
  }
}
```

Lo que realiza esta función es la comprobación de que el valor calculado no está en el array de valores anteriores. En caso de que este array contenga el

valor se vuelve a llamar a la función **randomNum()**, y si no lo contiene se da por válido y se guarda en el array de valores.

Ahora lo que falta por hacer es indicar a la función **restart()** que el array de valores debe reiniciarse cuando su tamaño (**length**) sea igual a 8, que es el número total de animales. Para ello añadiremos las siguientes líneas al final de la dicha función.

```
if(this.memoryArray.length==8){  
    this.memoryArray=[];  
}
```

Ahora sí que podemos dar por finalizada la programación del juego.

## Capítulo III: Añadiendo un SideMenu

### 1. Introducción

Para poder expandir la aplicación y crear más páginas a las que poder acceder tras el primer juego, vamos a crear un **SideMenu responsive** y dinámico.

El SideMenu es un menú hamburguesa cuando estamos ejecutando la aplicación en un móvil y se expande cuando se utiliza la aplicación en una Tablet en horizontal o en un ordenador. Es un menú muy intuitivo y fácil de utilizar.

### 2. Creando el SideMenu

La página que se va a cargar siempre al iniciar la aplicación se encuentra en **app.component.html**.

Dentro de la directiva **<ion-router-outlet>** se carga el contenido de la página que esté activa, es decir la que indique la ruta actual.

En este fichero vamos a añadir el siguiente código.

```
<ion-app>
  <ion-split-pane contentId="main-content">
    <ion-menu contentId="main-content">
      <ion-header>
        <ion-toolbar>
          <ion-title>Menú</ion-title>
        </ion-toolbar>
      </ion-header>

      <ion-content>
        <ion-list>
          <ion-menu-toggle auto-hide="false">
            <ion-item [routerDirection]="root" [routerLink]="'/home'">
              <ion-icon slot="start" [name]="home"></ion-icon>
              <ion-label>
                Sonidos de Animales
              </ion-label>
            </ion-item>
            <ion-item [routerDirection]="root" [routerLink]="'/adivina-sonido'">
              <ion-icon slot="start" [name]="person"></ion-icon>
              <ion-label>
                Adivina el Animal
              </ion-label>
            </ion-item>
          </ion-menu-toggle>
        </ion-list>
      </ion-content>
    </ion-menu>
  </ion-split-pane>
  <ion-router-outlet id="main-content"></ion-router-outlet>
```

```
</ion-split-pane>
</ion-app>
```

Lo primero que hacemos es envolverlos todo con una etiqueta **<ion-split-pane>** para adaptarnos a cualquier tipo de pantalla y mostrar el menú en el lateral del contenido. Se le asigna la propiedad **contentId="main-content"** que será el identificador que asignaremos a la etiqueta **<ion-router-outlet>**.

Después tenemos **<ion-menu>** que es el componente que contiene el menú y al que asignaremos el parámetro **contentId="main-content"**, al igual que a la etiqueta anterior, dentro del mismo añadimos su contenido como si fuera una página. Se ha añadido una etiqueta **<ion-header>** con un **toolbar** y dentro del título del menú.

Tras esto está **<ion-content>** que contiene un listado con dos elementos que son las opciones del menú.

Deben hacerse algunas apreciaciones al código expuesto:

- **routerDirection**. Indica la dirección de la animación de transición.
- **routerLink**. Define la ruta que debe cargarse la pulsarse. Debe de hacerse hincapié en que la ruta está escrita entre comillas dobles y simples para marcar que es un literal y no una variable.

También se podría haber definido una variable con un array de opciones con las rutas y hacer un bucle para mostrar las diferentes opciones. Como deberes se deja que se convierta el contenido dentro de la etiqueta **<ion-list>** en un bucle **\*ngFor**.

Si ejecutamos la aplicación en un navegador la aplicación debería de mostrarse de la siguiente manera.











Menú	Sonidos
 Sonidos de Animales	 Caballo
 Adivina el Animal	 Cabra
	 Cerdo
	 Gallo
	 Mono
	 Perro
	 Serpiente
	 Tigre

Figura 20. Vista del Menú lateral.



Los iconos de los botones no son los indicados para las opciones del menú, pero por el momento servirá, ya los modificaremos más adelante en la hoja de estilos.

Si en lugar de ejecutar la aplicación para un navegador la ejecutamos para que se visualice en un dispositivo móvil veremos que el menú lateral no aparece en la vista. Para poder mostrar el menú en este caso, debe crearse un botón que hará que se despliegue el menú cuando pulsemos encima.

Para esto es necesario modificar el archivo **home.page.ht.html** y añadir la directiva **<ion-menu-button>** que se encargará de mostrar/ocultar el menú al pulsarse.

El código que debemos añadir justo debajo de **<ion-toolbar>** es el siguiente.

```
<ion-buttons slot="start">
  <ion-menu-button></ion-menu-button>
</ion-buttons>
```

Debemos insertar el mismo código en la página **adivina-sonido.page.html**.

Ahora, si ejecutamos la aplicación en un dispositivo móvil veremos cómo aparece el menú en la esquina superior izquierda de la pantalla.

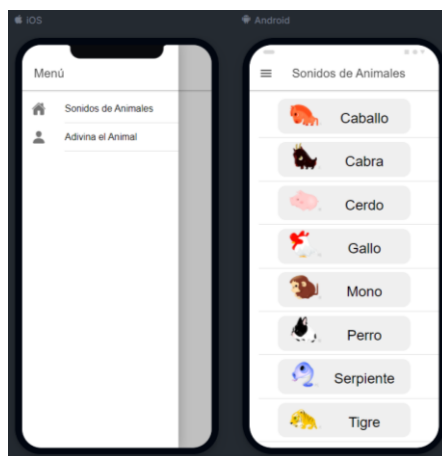


Figura 21. Vista del Menú Lateral en un dispositivo móvil.

Vamos a modificar los iconos para poner dos iconos que tengan algo que ver con la vista a la que se accede, para ellos iremos a una página de creación de iconos de forma gratuita<sup>1</sup> y crearemos dos iconos que nos gusten y sean adecuados para la aplicación.

Una vez creados los iconos los guardaremos dentro de la carpeta **img** que crearemos en **assets** y modificaremos el fichero **app.component.html** donde eliminaremos la directiva **<ion-icon>** y las substituiremos por

```

```

<sup>1</sup> En mi caso se ha utilizado <https://iconos8.es>

Y

```

```

El código completo del proyecto puede verse en el repositorio de GitLab.