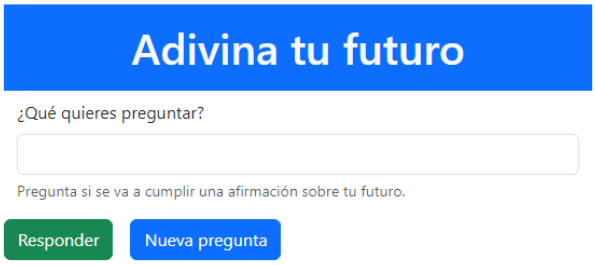
UD5 ACT1. Servicios Web con Ionic

En esta actividad vas a trabajar en la rama `feature/ud5_act1`. Fusiona dicha rama en `main` una vez que hayas finalizado la actividad.

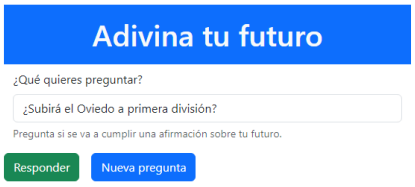
1. Vamos a usar la **API YesNo** (https://yesno.wtf/#api) para crear una App llamada **YesNoApp** que muestre una respuesta aleatoria (sí o no) al hacer clic en un botón.



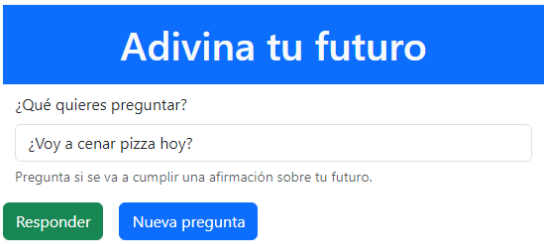
La API devuelve un JSON de este tipo:

```
{
  "answer":"no",
  "forced":false,
  "image":"https://yesno.wtf/assets/no/10d5ddf3f82134e781c1175614c0d2bab2.gif"
}
```

Al darle al botón de **Responder**, en la parte inferior del formulario se nos hace visible la respuesta:



El botón ***Nueva Pregunta*** simplemente restablecerá la aplicación al estado inicial y permitirá hacer una nueva pregunta:



Realiza validaciones para evitar que se pueda introducir una pregunta vacía.

Crea un servicio que te permita interaccionar con el servicio Web. Dicho servicio es intermediario entre tu aplicación y el propio servicio. Usa la librería HttpClient para realizar las peticiones a la API REST

Crea una interfaz que encapsule los datos del endpoint y defina el tipo de datos que vas a manejar en tu aplicación.

2. Vas a implementar una App llamada **`FutbolApp`** que nos permitirá visualizar resultados de las distintas ligas existentes, así como un buscador de jugadores.

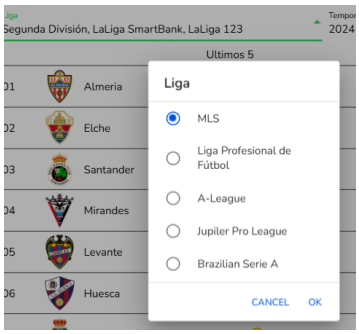
Cuando entramos a la App veremos una pantalla como la siguiente.

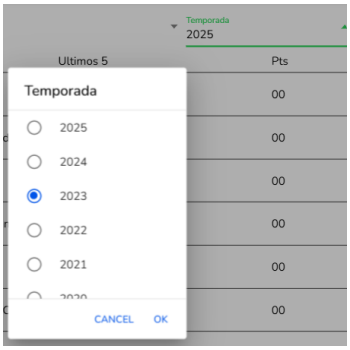
La aplicación está compuesta por dos pestañas:

- La pestaña de **Clasificaciones** nos permite consultar clasificaciones de distintas ligas en distintas temporadas (si escogemos la temporada actual se muestra la clasificación parcial, mientras que si escogemos temporadas pasadas se muestra la clasificación a final de temporada.

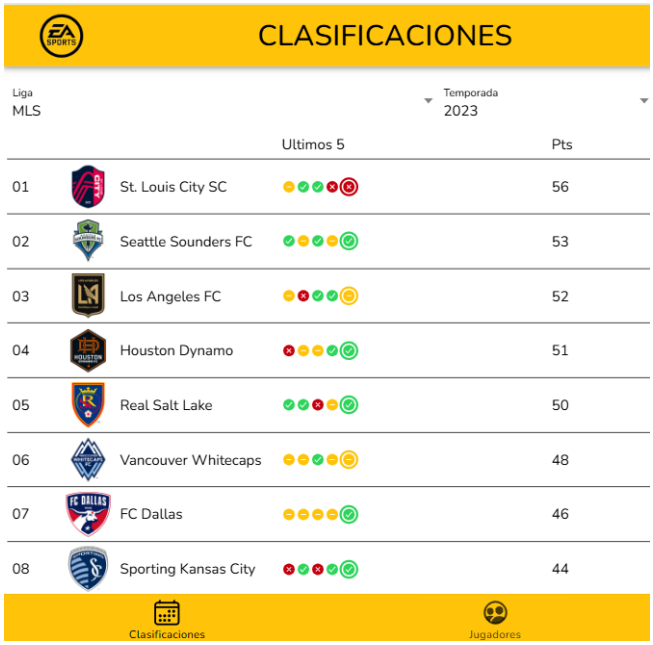
  Por defecto se muestra la clasificación de la temporada actual de la segunda división española.

  En cualquier momento podemos cambiar la liga y la temporada asociada:

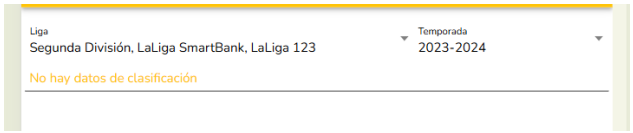




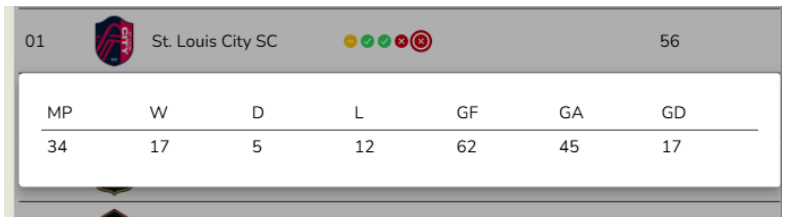
En ese caso se mostrará la clasificación de dicha liga/temporada:



Hay que tener en cuenta que la API no incluye datos de algunas ligas. En ese caso habrá que mostrar un mensaje indicativo.

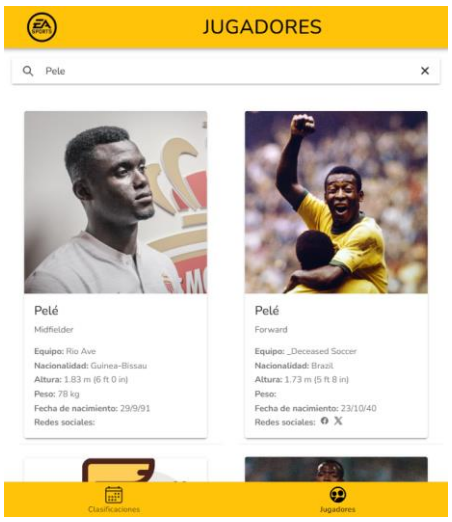


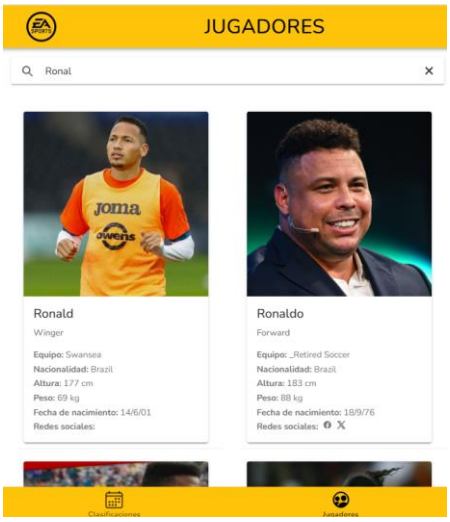
Cuando pulsamos sobre un equipo, se mostrará una ventana emergente (componente `ion-popover`) que mostrará datos adicionales de cada equipo de la tabla de clasificación.



- La pestaña de **Jugadores** nos permite consultar datos de jugadores. Por defecto se muestra una caja de búsqueda (`ion-searchbar`) que nos permite introducir el nombre de un jugador.

Cuando introducimos un nombre, se mostrarán todos los jugadores cuyo nombre incluya el término de búsqueda introducido. Nótese que si cambia el término de búsqueda habrá que cambiar el listado de jugadores mostrado.





Los datos se sacarán desde la capa gratuita de la API The Sports DB. Concretamente se usarán los siguientes endpoints.

La URL base para todas las peticiones es https://www.thesportsdb.com/api/v1/json/3/	
Endpoint	**Descripción**
`all_leagues.php`	Devuelve todas las ligas para las cuales se manejan datos
`search_all_seasons.php?id=idLiga`	Devuelve todas las temporadas de la liga con el id proporcionado.
`lookuptable.php?l=idLiga&s=temporada`	Devuelve la clasificación de una liga en una temporada concreta (ambos proporcionados como parámetro).
`/searchplayers.php?p=nombre`	Devuelve los jugadores (de todos los deportes) cuyo nombre incluya la cadena proporcionada.

Para tratar las llamadas deberías implementar los siguientes servicios

SERVICIOS OBLIGATORIOS	
Servicio	**Descripción**
`ClasificacionesService`	Encapsula las llamadas a la API relacionadas con las tablas de clasificación. Se usará la librería `HttpClient`. Debería tener los siguientes métodos: • `obtenerLigas()`. Devuelve los datos de todas las ligas. • `obtenerTemporadas(idLiga)`. Devuelve los datos de todas las ligas. • `obtenerClasificacion(idLiga,idTemporada).` Devuelve los datos de clasificación de una liga. Todos los métodos deberán devolver un `Observable` con los datos proporcionados por la API.
`JugadoresService`	Encapsula las llamadas a la API relacionadas con jugadores. Se usará la librería `CapacitorHttp` para realizar las llamadas a la API. Debería tener los siguientes métodos:

	• `obtenerJugadores(nombre)`. Devuelve los datos de todos los jugadores que incluyen la cadena proporcionada en su nombre. El método deberá devolver una `Promesa` con los datos proporcionados por la API.

Por otra parte, tendrás que implementar las siguientes interfaces:

INTERFACES OBLIGATORIAS	
Interfaz	Descripción
`LigasAPIResponse`	Encapsula la respuesta de la API al hacer una petición a los datos de todas las ligas
`Liga`	Representa los datos de una liga
`TemporadasAPIResponse`	Encapsula la respuesta de la API al hacer una petición de todas las temporadas de una liga.
`Temporada`	Representa los datos de una temporada.
`ClasificacionesAPIResponse`	Encapsula la respuesta de la API al hacer una petición de la clasificación de una temporada de la liga.
`Clasificación`	Representa la clasificación de un equipo en la liga en una temporada
`JugadoresAPIResponse`	Encapsula la respuesta de la API al hacer una petición de Jugadores
`Jugador`	Representa los datos de un jugador.

Además tendrás que implementar los siguientes componentes:

COMPONENTES		
Componente	**Tipo**	
`Cabecera`	Componente	Representa la cabecera de las páginas (`ion-header`). El componente recibe como @Input() el título de la página. Muestra el logo y el título. Se recomienda usar un `ion-grid` para maquetar el contenido de la cabecera. Este componente se usa en las páginas `Clasificacion` y `Jugadores`
`Futbol`	Página	Contiene el TabBar (componente `ion-tabs`) que permite la navegación a las otras páginas.
`Clasificacion`	Página	Contiene la página de clasificación. Contiene las siguientes propiedades: • Listado de ligas. • Listado de temporadas • Listado de resultados de clasificación. • Liga actual • Temporada actual
`Jugadores`	Página	Contiene la página de jugadores. Contiene las siguientes propiedades: • Listado de jugadores.

Consideraciones generales:

- Genera un logo a medida para la aplicación. Puedes utilizar un generador de logos como Looka. Tendrás que generar un logo en negro y con tono transparente.
- Personaliza la tipografía de la página (escoge una tipografía de Google Fonts o el proveedor de tipografías que prefieras).
- Deberías usar la nueva sintaxis de Angular para directivas en las plantillas (@if, @for, @switch)

Respecto a la página `Futbol`:

- Tanto ligas como temporadas deberían mostrarse en un componente `ion-select.` El componente debería almacenar en dos propiedades la liga actual y la temporada actual, y estos deberían ir ligadas a los `select` con 2-way data binding (`[(ngModel)]`).
    - Además, los ion-select deberían tener el campo `label` definido y la propiedad `label-placement="stacked"` para que esté superpuesta al select.
- Inicialmente, deberían cargarse todas las ligas de futbol (solo las de futbol) y éstas deberían estar ordenadas alfabéticamente de acuerdo al nombre en orden ascendente. Por otra parte, si se puede mostrar el nombre alternativo de la liga (`strLeagueAlternate`), aunque en caso de que este no esté definido habrá que mostrar el nombre oficial (`strLeague`). Inicialmente se mostrará la liga de segunda (id 4400).
- Inicialmente se mostrarán todas las temporadas de segunda división, ordenadas en orden inverso (la más reciente la primera).
- Deberías definir el evento de click `(ion-click)` sobre las ligas para que se carguen de nuevo las temporadas cada vez que cambia la liga. Y el evento de click sobre las temporadas para que cambie la clasificación cada vez que cambiamos la temporada.
- Todo el contenido de la página irá maquetado en un ion-grid, incluyendo los select de la parte superior. Todas las filas llevarán un borde inferior.
- Para mostrar la clasificación se usará:
    - Una fila de cabecera
    - Una fila para cada línea de clasificación:
        - Si el número de la clasificación es menor que 10, se añadirá un 0 delante, para estandarizar el modo que se muestra.
        - El texto de todas las columnas estará centrado vertical y horizontalmente.
        - Para mostrar los iconos de victorias/empates/derrotas, tendrás que recorrer el campo `strForm` (que contiene W en caso de victoria, D en caso de empate y L en caso de derrota.
            - Se mostrará un icono distinto en función de la letra que nos encontremos. Cada letra en un color distinto, dependiendo de si tenemos victoria, empate o derrota.
            - Para resaltar el último resultado, le pondremos un borde más grueso de 3 píxeles.
        - Ten en cuenta que el campo contiene las victorias/derrotas en orden inverso al producido, luego deberías dar la vuelta a los caracteres del campo antes de recorrerlo.
- Para mostrar las ventanas emergentes:
    - Tendrás que definir un `ion-popover` asociado a cada una de las filas.
    - El popover debería tener las siguientes propiedades establecidas, que de manera conjunta desencadenan que se muestre el popover al pulsar sobre su fila asociada:
        - `triggerAction: click`
        - `size: cover`

- `trigger:row-i`, siendo `i` el identificador que le daremos a la fila asociada. Es decir, la fila 1 debería tener el id `row-1` y su popover asociado debería tener el atributo `trigger:row-1`

Respecto a la página `Jugadores`:

- Define un `ion-searchbar` con los siguientes elementos definidos:
  - `Placeholder: "Buscar jugador…"`
  - `debounce:500.` Representa el retraso en milisegundos que hay que esperar para llamar al evento de cambio en caso de que no se pulse otra tecla. Esto evita sobresaturar con llamadas al servidor cada vez que pulsamos tecla.
  - `(ion-input): buscarJugador($event)`. Define el manejador de evento asociado a pulsación de tecla. Llamaremos a la función `buscarJugador()` pasándole información del evento. Desde el manejador de evento, accedemos al texto introducido con `event.target.value` y realizaremos una llamada a la API para obtener el listado de jugadores con el texto introducido. Deberíamos filtrar la respuesta obtenida para guardar solo los jugadores de futbol (`strSport` es "`soccer`").

- En la plantilla, si hay elementos en el array de jugadores, ese se recorrerá y se mostrará un panel para cada uno de los jugadores.
  - La fecha debería mostrarse en español. Para ello:
    - En `main.ts` deberías registrar la localización actual:

```
import localeEs from '@angular/common/locales/es';
import { registerLocaleData } from '@angular/common';

// Registrar la localización española
registerLocaleData(localeEs, 'es-ES');
```

    - En `appcomponent.ts` deberías definir los providers:

```
  providers: [{ provide: LOCALE_ID, useValue: 'es-ES' }]
```

    - En la plantilla, muestra la fecha mediante un pipe para formato corto:

```
<p><strong>Fecha de nacimiento:</strong> {{  (jugador.dateBorn) | date: 'shortDate' }}</p>
```

  - Los logos de redes sociales solo se mostrarán si los campos correspondientes están definidos.

3. Utiliza la herramienta https://retool.com/api-generator para generar una API REST con un endpoint llamado restaurantes. Para cada restaurante, define al menos 5 campos (uno de ellos debería ser de tipo URL de imagen). Genera la API con 200 restaurantes.

   A continuación, implementa una aplicación llamada `RestaurantesApp` que utilice la API `CapacitorHttp` para:

   - Mostar un listado de restaurantes (usando una petición GET paginada de 30 en 30). Para cada restaurante solo se mostrará su nombre.
   - La página incluirá un scroll infinito, de modo que al alcanzar el último restaurante que se ha mostrado en la página, se pedirán 30 nuevos restaurantes. Se mostrará un spinner mientras cargan los restaurantes.
   - Mientras carga el listado, deberías mostrar un elemento de carga (`ion-loading`).
   - Al pulsar sobre un restaurante, se abrirá una nueva página, con detalles sobre del mismo. Desde la página se realizará una nueva petición para obtener datos de únicamente ese restaurante.

Ten en cuenta las siguientes consideraciones:

- Crea un servicio que te permita interaccionar con el servicio Web. Dicho servicio es intermediario entre tu aplicación y el propio servicio.
- Crea una interfaz que encapsule los datos del endpoint y defina el tipo de datos que vas a manejar en tu aplicación.
- Crea las siguientes páginas (para mejor organización puedes generarlas dentro de una carpeta con el nombre del endpoint):
  - Nueva. Muestra un formulario que permite crear una nueva entidad del tipo escogido. Debería realizar las validaciones pertinentes. Si la entidad se ha creado correctamente, mostrarás un mensaje de confirmación.

- La aplicación debería mostrar el listado cuando entramos en la misma. Para mostrar la página de “Nueva”, la página de listado incluirá un “Floating Action Button” (consulta la documentación de Ionic para ver cómo se usa).

Añade a cada restaurante botones que permitan:

- Modificar un restaurante (peticiones PUT/PATCH)
- Eliminar un restaurante (petición DELETE).

Piensa como quieres integrar los datos anteriores en una interfaz de usuarios. Puedes usar botones (tab-buttons) o un menú.

Ten en cuenta las siguientes consideraciones:

- Integra la operación de borrado en un botón desde el listado o en el detalle (ten en cuenta que el borrado necesita el ID de entidad). Antes de realizar el mismo, deberías pedir confirmación. Si la respuesta del endpoint es correcta, se mostrará un mensaje de confirmación.

- Integra la operación de actualización de un botón desde el listado o el detalle (ten en cuenta que el borrado necesita el ID de entidad). Al pulsar el botón se irá a una nueva página con un formulario (puedes llamarla modificación). Dicho formulario debería estar pre-rellenado con los campos de la entidad que queremos modificar. Una vez que se han cambiado los campos, el envío de formulario realizará una petición UPDATE/PATCH. Si la respuesta es satisfactoria, mostramos un mensaje indicativo y volvemos al listado.

Modifica el servicio para mantener una caché en `localStorage`:

a. Si es la primera vez que entramos a la App (o no existe la clave en `localStorage`): Los datos se cargarán desde el servicio Web y se creará una clave en `localStorage` cuya clave es el nombre de la entidad y cuyo contenido es el listado en JSON.
b. Si al hacer la petición de lectura los datos no se cargan correctamente, el listado se cargará desde el `localStorage` en lugar de hacerlo desde el servicio Web.
c. Las operaciones Create/Update/Delete actualizarán el `localStorage` además de hacer las peticiones Web correspondientes.

RESULTADOS DE APRENDIZAJE Y CRITERIOS DE EVALUACIÓN

Esta actividad contribuye a la consecución del RA5. *Crea informes evaluando y utilizando herramientas gráficas.*

Los criterios de evaluación que permiten valorar la consecución del resultado de aprendizaje son los siguientes:

5a) Se ha establecido la estructura básica del informe.
4b) Se han generado informes básicos a partir de diferentes fuentes de datos mediante asistentes.