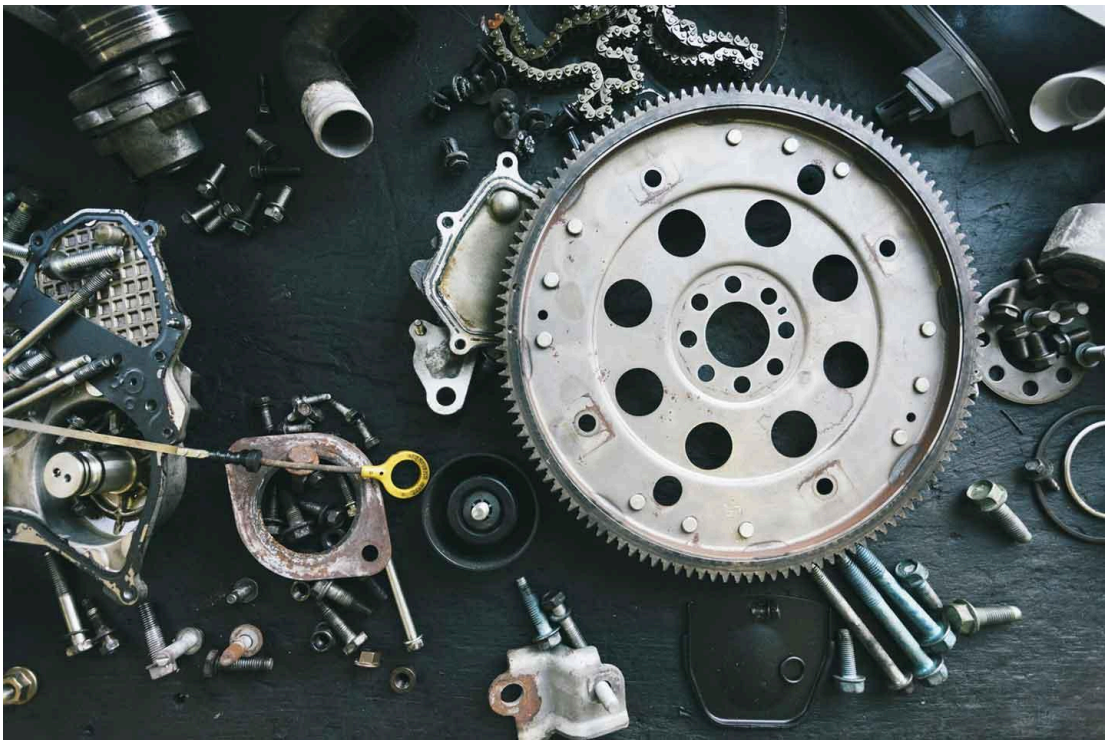
Ionic v8

Infinite Scroll



Progresa
Una formación de futuro

ÍNDICE

Introducción

El infinite scroll lanza un evento que se realizará cuando el usuario haga scroll a una distancia específica del principio o final de la página.

La expresión asignada al evento **ionInfinite** se llama cuando el usuario alcance esa distancia especificada. Cuando se terminen todas las tareas, hay que llamar a la función **complete()** de la instancia de infinite scroll.

La documentación del infinite scroll está en la página:

https://ionicframework.com/docs/api/infinite-scroll

Preparación

Componente

Para empezar a trabajar con el infinite scroll vamos a necesitar una página donde usarlo. Creamos la página en la **terminal**:

ionic g page pages/InfiniteScroll --spec=false

Icono

Añadimos en nuestro **app.component.ts** el icono que queremos usar para el menú en la lista de iconos(addIcons).

```
addIcons({grid, logoAppleAppstore, americanFootball, beaker, star, trash, shareSocial, caretForwardCircle, close, card, pin, car, logoFacebook,logoTwitter,logoTiktok,logoInstagram,logoYoutube,logoGithub,add, albums, ellipsisHorizontalOutline, list, heart, call, infinite});
```

JSON

Añadimos en nuestro **menu.json** el nuevo elemento que queremos que aparezca tanto en el menú como en la página de inicio.

```
,{
 "nombre": "Infinite Scroll",
 "ruta": "/infinite-scroll",
 "icono": "infinite"
}
```

EJEMPLO

LÓGICA

En primer lugar vamos a crear una lista de elementos para poder utilizar nuestro infinite scroll. Para ello en **infinite-scroll.page.ts** creamos un array de 20 elementos y una función que cargue la información con cada evento del infinite scroll:

```
export class InfiniteScrollPage implements OnInit {
 data: any[] = Array(20);
 constructor() { }
 ngOnInit() {
 }

 loadData(event: any) {
   console.log('Cargando siguientes ...');
   setTimeout(() =>{
     const nuevoArr = Array(20);
     this.data.push( ...nuevoArr);
     event.target.complete();
   }, 1000 );
 }
}
```

VISUALIZACIÓN

Y en nuestra página que visualiza el contenido, **infinite-scroll.page.html**, vamos a reproducir una lista de elementos:

```
<app-header titulo="Infinite Scroll"></app-header>
<ion-content>
 <ion-list>
   @for (item of data; track i; let i = $index){
     <ion-item >
       <ion-label> Item {{ i + 1 }} </ion-label>
     </ion-item>
   }
 </ion-list>
 <ion-infinite-scroll threshold="150px"
                      (ionInfinite)="loadData($event)">
   <ion-infinite-scroll-content
     loadingSpinner="crescent"
     loadingText="Cargando 20 más...">
   </ion-infinite-scroll-content>
 </ion-infinite-scroll>
</ion-content>
```

Explicación

```
 <ion-infinite-scroll threshold="150px"
      (ionInfinite)="loadData($event)">
```

Especificamos donde vamos a cargar el infinite scroll y sus propiedades:

- `threshold`: La distancia del scroll desde el borde superior o inferior .
- `(ionInfinite)`: Especificamos qué haremos cuando se lance el evento. En nuestro caso vamos a lanzar la función loadData.

```
<ion-infinite-scroll-content
     loadingSpinner="crescent"
     loadingText="Cargando 20 más...">
   </ion-infinite-scroll-content>
```

Es la parte visual del scroll. Definimos el Spinner(imagen) y texto si queremos.

Explicación

***setTimeout*(() =>{**: Vamos a simular una carga síncrona.

const nuevoArr = Array(20);: Creamos un nuevo array de 20 elementos.

this.data.push(...nuevoArr);: Concatenamos el nuevo array al array **data** que teníamos con la función **push**. Utilizamos el comando **spread**(**...**) de TypeScript para añadirlo al final.

event.target.complete();: Le indicamos al sistema que ya ha terminado de cargar para quitar el icono y el texto de carga del infinite scroll.

¿Pero qué pasa si nuestra colección de datos tiene 50 registros?

Para evitar el error que aparece al final de la colección, vamos a bloquear el infinite scroll al llegar a los 50. En **infinite-scroll.page.ts**:

```
export class InfiniteScrollPage implements OnInit {

      @ViewChild(IonInfiniteScroll,      {static:      false})
infiniteScroll: IonInfiniteScroll;

 data: any[] = Array(20);
 constructor() { }

 ngOnInit() {
 }

 loadData(event: any) {
   console.log('Cargando siguientes ...');
   setTimeout(() =>{
     if (this.data.length > 50) {
       event.target.complete();
       this.infiniteScroll.disabled = true;
       return;
     }
```

```
      const nuevoArr = Array(20);
      this.data.push(...nuevoArr);
      event.target.complete();
    }, 1000 );
  }
}
```

EJEMPLO 2

PREPARACIÓN

Componente

Vamos a ver otro ejemplo de uso del infinite scroll. Para empezar a trabajar con el infinite scroll vamos a necesitar una página donde usarlo. Creamos la página en la **terminal**:

ionic g page pages/Infinite2 --spec=false

Icono

Añadimos en nuestro **app.component.ts** el icono que queremos usar para el menú en la lista de iconos(addIcons).

```
addIcons({grid, logoAppleAppstore, americanFootball, beaker, star, trash, shareSocial, caretForwardCircle, close, card, pin, car, logoFacebook,logoTwitter,logoTiktok,logoInstagram,logoYoutube,logoGithub,add, albums, ellipsisHorizontalOutline, list, heart, call, infinite, infiniteOutline});
```

Json

Añadimos en nuestro **menu.json** el nuevo elemento que queremos que aparezca tanto en el menú como en la página de inicio.

```
,{
 "nombre": "Infinite Scroll 2",
 "ruta": "/infinite2",
 "icono": "infinite-outline"
}
```

VARIABLES DE ENTORNO

Lo primero que vamos a hacer en nuestro proyecto es definir unas variables de entorno. Estas son variables accesibles desde toda la aplicación con un valor fijo. En la carpeta environment accedemos al archivo **environment.ts**:

```
export const environment = {
 production: false,
 apiKey: '6a9954c562acf2cc4a9b2b0b9648e899',
 baseUrl: 'https://api.themoviedb.org/3',
 images: 'http://image.tmdb.org/t/p'
};
```

En la página https://developers.themoviedb.org/3/getting-started/introduction tenéis la información de la API que vamos a utilizar por si queréis crearos una cuenta para que os den una clave pública de uso. Es la base de datos de películas.

SERVICIO

Lo siguiente que vamos a hacer es cargar las películas en nuestra aplicación mediante una función en el servicio **data.service.ts**:

```
 loadMovies(page: number): Observable<ApiMoviesResult> {
   return this.http.get<ApiMoviesResult>(
`${environment.baseUrl}/movie/popular?api_key=${environment.ap
iKey}&page=${page}`);
 }
}
```

INTERFAZ

Creamos la interfaz para los datos de la API en **interfaces.ts**:

```
export interface ApiMoviesResult {
 page: number
 results: Movie[]
 total_pages: number
 total_results: number
}


export interface Movie {
 adult: boolean
 backdrop_path: string
 genre_ids: number[]
 id: number
 original_language: string
 original_title: string
 overview: string
 popularity: number
 poster_path: string
 release_date: string
 title: string
 video: boolean
 vote_average: number
 vote_count: number
}
```

GUARDAR DATOS

Ahora que hemos cargado las películas desde el servicio y tenemos las interfaces listas, vamos a guardarlas en un elemento en la página. Además configuraremos nuestra página con una función que cargue los elementos página a página. En **infinite2.page.ts**:

```
import {Component, inject, OnInit} from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import {IonContent, IonHeader, IonTitle, IonToolbar, LoadingController} from '@ionic/angular/standalone';
import {InfiniteScrollCustomEvent} from "@ionic/angular";
import {environment} from "../../../environments/environment";
import {DataService} from "../../services/data.service";
import {ApiMoviesResult, Movie} from "../../common/interfaces";

@Component({
 selector: 'app-infinite2',
 templateUrl: './infinite2.page.html',
 styleUrls: ['./infinite2.page.scss'],
 standalone: true,
  imports: [IonContent, IonHeader, IonTitle, IonToolbar, CommonModule, FormsModule]
})
export class Infinite2Page implements OnInit {
 private dataService: DataService = inject(DataService);
 movies: Movie[] = [];
 currentPage = 1;
 imageUrl = environment.images;

 constructor(private loadingCtrl: LoadingController) { }

 ngOnInit() {
   this.loadMovies();
 }
```

```
 async loadMovies(event?: InfiniteScrollCustomEvent) {
   const loading = await this.loadingCtrl.create({
     message: 'Loading..',
     spinner: 'bubbles'
   });
   await loading.present();

   this.dataService.loadMovies(this.currentPage).subscribe(
     {
       next:    (res: ApiMoviesResult) => {
         loading.dismiss();
         this.movies.push(...res.results);
         event?.target.complete();
         if(event){
           event.target.disabled =
                 res.total_pages === this.currentPage;
         }
       },
       error: err => { console.error(err); },
       complete: () => { console.log('Movies loaded!'); }
     }

   );
 }

 loadMore(event: InfiniteScrollCustomEvent) {
   this.currentPage++;
   this.loadMovies(event);
 }
}
```

Explicación

```
imageUrl = environment.images;
```

Cargo la url base de las imágenes en una variable para poder usarla desde el html.

```
constructor(private loadingCtrl: LoadingController) { }
```

En el constructor inyectamos el controlador del loading para poder utilizarlos.

```
async loadMovies(event?: InfiniteScrollCustomEvent) {
```

La función que cargará las películas es asíncrona porque tendremos un loading en marcha hasta que termine la tarea de cargar las películas.

```
      message: 'Loading..',
```

El mensaje que aparecerá en el loading.

```
      spinner: 'bubbles'
```

La imagen que tendrá el loading.

```
        loading.dismiss();
```

Una vez ya hemos cargado los datos, quitamos el loading con dismiss().

```
        this.movies.push(...res.results);
```

Cuando llamemos a la función, traeremos las películas del servicio y las añadimos a continuación del contenido de nuestro array de películas con la función push.

```
        event?.target.complete();
```

Paramos el Infinite Scroll.

```
if(event){event.target.disabled = res.total_pages === this.currentPage;}
```

Si viene del infinite scroll y la página actual es la última de la API, desactivamos el Infinite Scroll.

```
loadMore(event: InfiniteScrollCustomEvent) {
  this.currentPage++;
  this.loadMovies(event);
}
```

Cuando se active el infinite scroll desde el html, aumentamos la página actual y cargamos las siguientes películas llamando a loadMovies y pasando el evento.

Y ya por último definimos la parte visible, en **infinite2.page.html**:

```
<app-header titulo="Infinite Scroll v2"></app-header>
<ion-content>
 <ion-list>
   @for (movie of movies; track movie.id){
     <ion-item mode="ios" button >
       <ion-avatar slot="start">
                      <ion-img   [src]="imageUrl   +   '/w92/'   +
movie.poster_path" [alt]="movie.title"></ion-img>
       </ion-avatar>
       <ion-label>
         {{ movie.title }}
         <p>{{ movie.release_date | date:'y' }}</p>
       </ion-label>

                    <ion-badge   slot="end"   color="success">{{
movie.vote_average }}</ion-badge>
     </ion-item>
   }

 </ion-list>

 <ion-infinite-scroll (ionInfinite)="loadMore($event)">
   <ion-infinite-scroll-content
     loadingSpinner="bubbles"
     loadingText="Loading more data...">
   </ion-infinite-scroll-content>
 </ion-infinite-scroll>
</ion-content>
```