

Aplicación de películas

1. Iniciamos un nuevo proyecto

ionic start películasAPP tabs

2. Navegamos dentro de la app creada

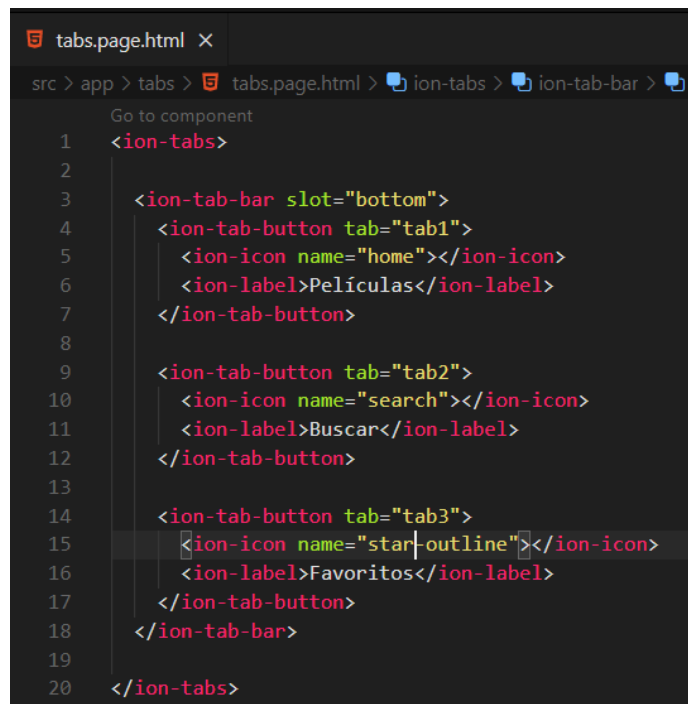
ionic start

3. Modifiquemos el Tabs. El html, coloquemos el siguiente código

En el tab 1 llamemosle name="home" y en titulo películas

En el tab 2, llamémosle name="search" y en titulo: Buscar

En el tab 3, Llamemosme name="start-outline" y en titulo: Favoritos



```
1 <ion-tabs>
2
3   <ion-tab-bar slot="bottom">
4     <ion-tab-button tab="tab1">
5       <ion-icon name="home"></ion-icon>
6       <ion-label>Películas</ion-label>
7     </ion-tab-button>
8
9     <ion-tab-button tab="tab2">
10      <ion-icon name="search"></ion-icon>
11      <ion-label>Buscar</ion-label>
12    </ion-tab-button>
13
14    <ion-tab-button tab="tab3">
15      <ion-icon name="star-outline"></ion-icon>
16      <ion-label>Favoritos</ion-label>
17    </ion-tab-button>
18  </ion-tab-bar>
19
20 </ion-tabs>
```

Ilustración 1: tabs.page.html modifica iconos al pie de pantalla

Clase 11- Proyecto práctico

4. Ahora esperamos que ya tengamos creada nuestra cuenta en themoviedb.org.
5. Crearemos nuestro primer servicio para traer películas de estreno

ionic generate service services/movies --skipTests=true

```
Use npx ng g --help to list available types of features.

C:\Users\David\Documents\IONIC projects\peliculasApp>ionic generate service services/movies --skipTests=true
> ng.cmd generate service services/movies --skip-tests=true --project=app
CREATE src/app/services/movies.service.ts (135 bytes)
[OK] Generated service!

C:\Users\David\Documents\IONIC projects\peliculasApp>
```

Ilustración 2: Instruccion para crear un servicio dentro de la carpeta servicios

Notar que se ha creado el fichero services

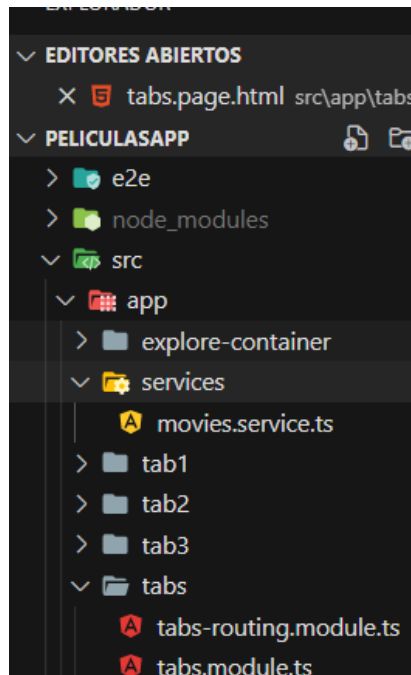


Ilustración 3: Se ha creado el fichero services

Clase 11- Proyecto práctico

6. Se ha creado el servicio y agregado en el root

```
movies.service.ts x
src > app > services > movies.service.ts > ...
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class MoviesService {
7
8    constructor() { }
9  }
10
```

Ilustración 4: Se ha creado el servicio

7. Ahora es de saber que para hacer peticiones http, necesitamos importar en el app.module.ts así

```
app.module.ts 1
src > app > app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { RouteReuseStrategy } from '@angular/router';
4
5  import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
6
7  import { AppRoutingModule } from './app-routing.module';
8  import { AppComponent } from './app.component';
9
10 import { HttpClientModule } from '@angular/common/http';
11
12 @NgModule({
13   declarations: [AppComponent],
14   entryComponents: [],
15   imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule],
16   providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
17   bootstrap: [AppComponent],
18 })
19 export class AppModule {}
20
```

Ilustración 5: Creando el modulo http

Clase 11- Proyecto práctico

Ahora recuerden que siempre que creamos un MODULO, tendremos que agregarlo en la parte de los imports-

Entonces agregaremos en los imports el httpClientModules

```
import {HttpClientModule} from '@angular/common/http'

@NgModule({
  declarations: [AppComponent],
  entryComponents: [],
  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule, HttpClientModule],
  providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Ilustración 6: Importando el modulo httpClientModule en los imports

8. Ahora vamos a inyectar el servicio httpClientModule, dentro del **movies.services.ts** así

```
movies.service.ts X
src > app > services > movies.service.ts > MoviesService > constructor
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class MoviesService {
8
9   constructor(private http: HttpClient) {}
10 }
11
```

Ilustración 7: Inyectando el servicio httpClient

Clase 11- Proyecto práctico

9. Ahora empecemos con nuestro primer servicio que se llame **getFetaures ()**

```
@Injectable({
  providedIn: 'root',
})
export class MoviesService {
  constructor(private http: HttpClient) {}

  getFetaures() {
    return this.http.get(`TODA LA URL OBTENIDA DE POSTMAN QUE PROBAMOS ANTES
    ENTRE BACK TIKS, COMILLAS INVERTIDAS`)
  }
}
```

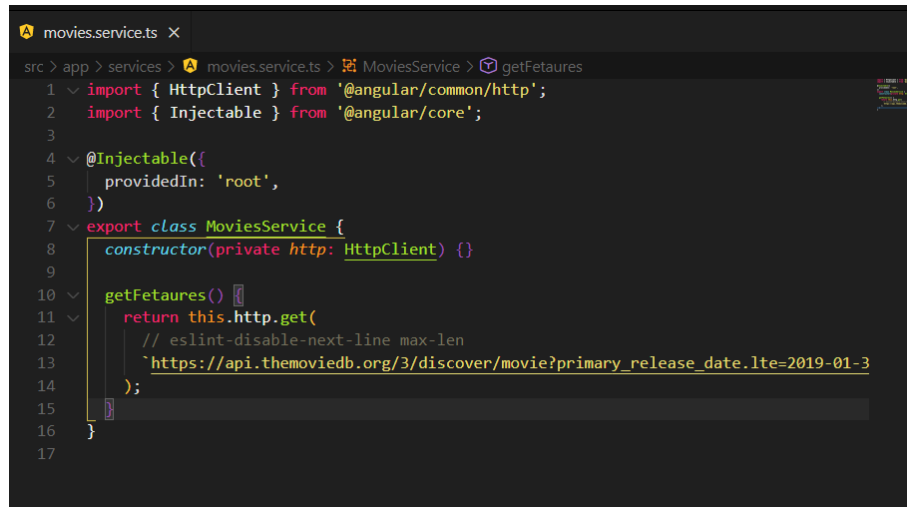
The screenshot shows the Postman interface with a GET request to the TMDB API. The URL is `https://api.themoviedb.org/3/discover/movie?primary_release_date.lt=2019-01-31&api_key=333d175cfa6dc69...`. The parameters are listed in a table below:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> primary_release_date.lt	2019-01-31	
<input checked="" type="checkbox"/> api_key	333d175cfa6dc69b4c15a2a00232692c	
<input checked="" type="checkbox"/> primary_release_date.gte	2019-01-01	
<input checked="" type="checkbox"/> language	es	
<input checked="" type="checkbox"/> include_image_language	es	

Ilustración 8: URL obtenido de la prueba realizada en postman

Si sale un error SLINT sugiere que acortemos esa URL, así que podemos comentar la línea que se sugiere

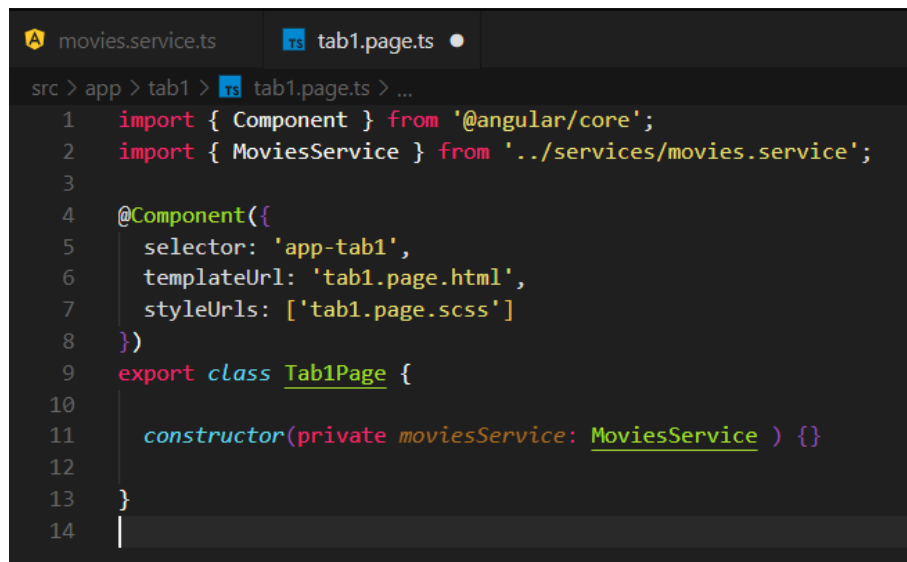
Clase 11- Proyecto práctico



```
src > app > services > movies.service.ts > MoviesService > getFetaures
1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3
4  @Injectable({
5    providedIn: 'root',
6  })
7  export class MoviesService {
8    constructor(private http: HttpClient) {}
9
10   getFetaures() {
11     return this.http.get(
12       // eslint-disable-next-line max-len
13       'https://api.themoviedb.org/3/discover/movie?primary_release_date.lte=2019-01-3
14     );
15   }
16 }
17
```

Ilustración 9: SLINT sugiere acortar la URL

Ahora vamos a probar el servicio desde la TAB 1. En el fichero: tab1.page.ts

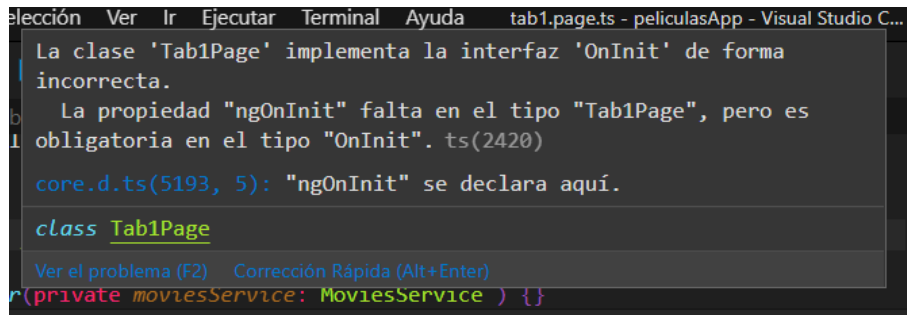


```
src > app > tab1 > tab1.page.ts > ...
1  import { Component } from '@angular/core';
2  import { MoviesService } from '../services/movies.service';
3
4  @Component({
5    selector: 'app-tab1',
6    templateUrl: 'tab1.page.html',
7    styleUrls: ['tab1.page.scss']
8  })
9  export class Tab1Page {
10
11    constructor(private moviesService: MoviesService ) {}
12
13  }
14
```

Ilustración 10: Inyectar el servicio e importarlo

Clase 11- Proyecto práctico

Ahora vamos a implementar el `OnInit`¹

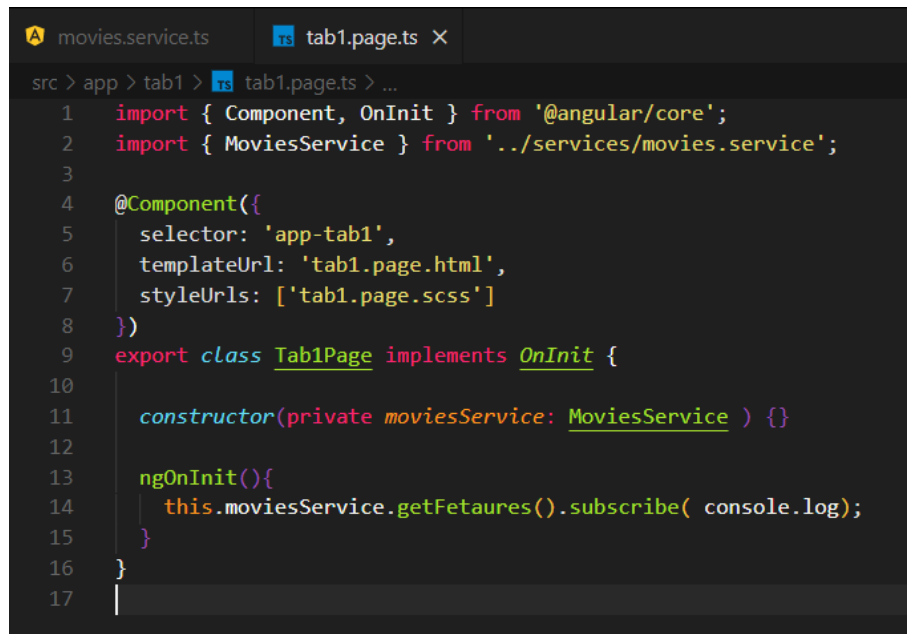


```
La clase 'Tab1Page' implementa la interfaz 'OnInit' de forma incorrecta.
La propiedad "ngOnInit" falta en el tipo "Tab1Page", pero es obligatoria en el tipo "OnInit". ts(2420)
core.d.ts(5193, 5): "ngOnInit" se declara aquí.

class Tab1Page
Ver el problema (F2)  Corrección Rápida (Alt+Enter)
r(private moviesService: MoviesService ) {}
```

Ilustración 11: Mensaje de aviso que el `OnInit` está mal implementado

Vamos a corregir ese error fuera del constructor en donde llamaremos ese servicio y por el momento lo podemos imprimir en la consola para ver como funciona o los resultados que obtendremos.



```
src > app > tab1 > ts tab1.page.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { MoviesService } from '../services/movies.service';
3
4 @Component({
5   selector: 'app-tab1',
6   templateUrl: 'tab1.page.html',
7   styleUrls: ['tab1.page.scss']
8 })
9 export class Tab1Page implements OnInit {
10
11   constructor(private moviesService: MoviesService ) {}
12
13   ngOnInit(){
14     this.moviesService.getFetaures().subscribe( console.log);
15   }
16 }
17
```

Con `subscribe()` se usa para lanzar la petición.

¹ `ngOnInit` pertenece al ciclo de vida propio de angular y es aquí donde le 'decimos' que el componente ya está listo para darle uso.

Comentado [D.1]: `ngOnInit` pertenece al ciclo de vida propio de angular y es aquí donde le 'decimos' que el componente ya está listo para darle uso.

Clase 11- Proyecto práctico

10. Si no tenemos errores, lanzamos **ionic serve** y veamos el resultado obtenido.

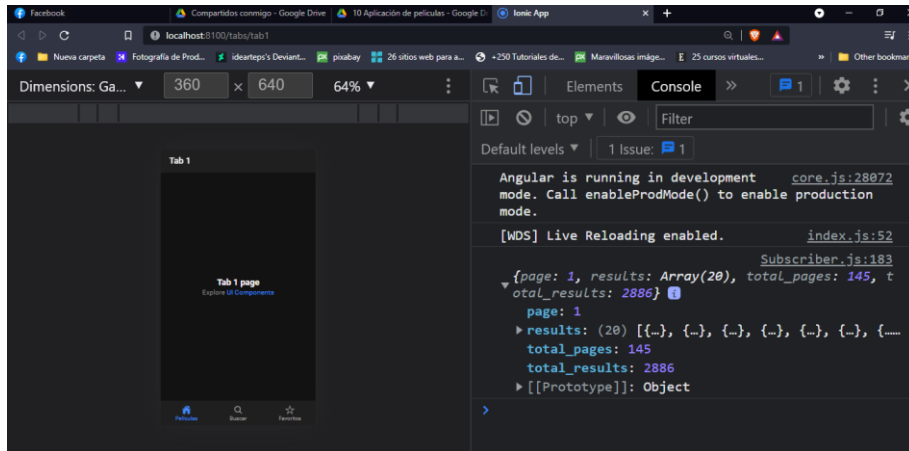


Ilustración 12: Resultado, ver console

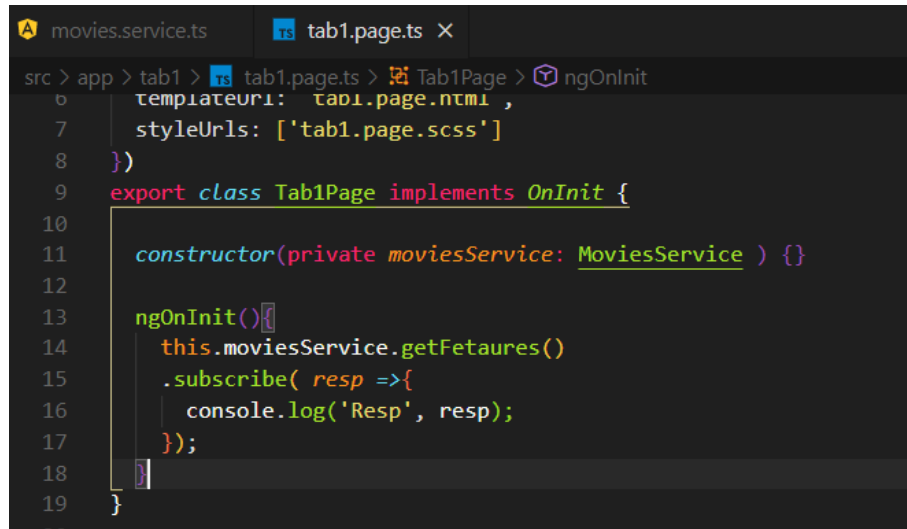
Probar que funcione correctamente hasta este punto, revisar los pasos previos y compartir resultado.

Parte II

Crear la interfaz para presentar los resultados

1. Modificaremos nuestro console.log para controlar los resultados de una manera mas sencilla

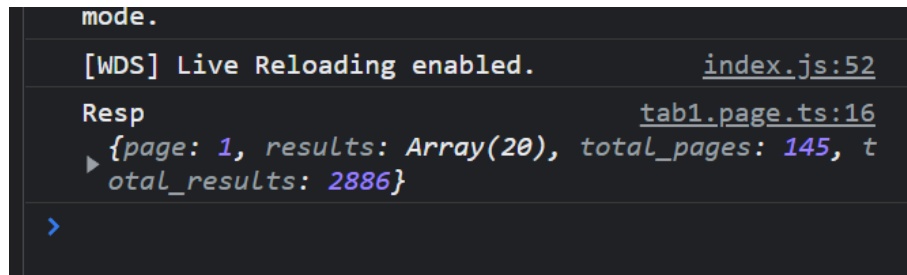
Clase 11- Proyecto práctico



```
src > app > tab1 > tab1.page.ts > Tab1Page > ngOnInit
6   templateUrl: 'tab1.page.html',
7   styleUrls: ['tab1.page.scss']
8 })
9 export class Tab1Page implements OnInit {
10
11   constructor(private moviesService: MoviesService ) {}
12
13   ngOnInit(){
14     this.moviesService.getFetaures()
15       .subscribe( resp =>{
16         console.log('Resp', resp);
17       });
18   }
19 }
```

Ilustración 13: Modificando el console.log para mejor control del resultado

Guardamos cambios y vemos resultado que debería ser el mismo que teníamos anteriormente



```
mode.
[WDS] Live Reloading enabled.      index.js:52
Resp                                tab1.page.ts:16
▶ {page: 1, results: Array(20), total_pages: 145, t
  otal_results: 2886}
>
```

Ilustración 14: resultado del código anterior

Ahora vamos a postman y copiemos toda la respuesta o resultado que nos envío la consulta.

Clase 11- Proyecto práctico

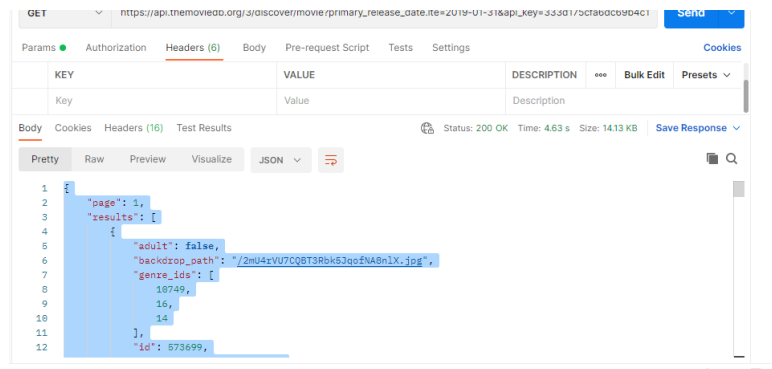


Ilustración 15: resultado Postman a copiar

Ahora, vamos a trabajar para crear nuestras interfaces

Nos vamos al árbol de ficheros y allí creamos una carpeta llamada interfaces y luego un nuevo documento llamado interface.ts de la carpeta APP

Y allí dentro del interfaces.ts, vamos a llamar el plugin JSON to TS convert to clipboard



Ilustración 16: Plugin a utilizar

Clase 11- Proyecto práctico

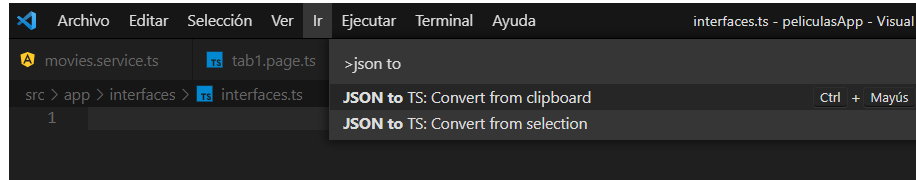
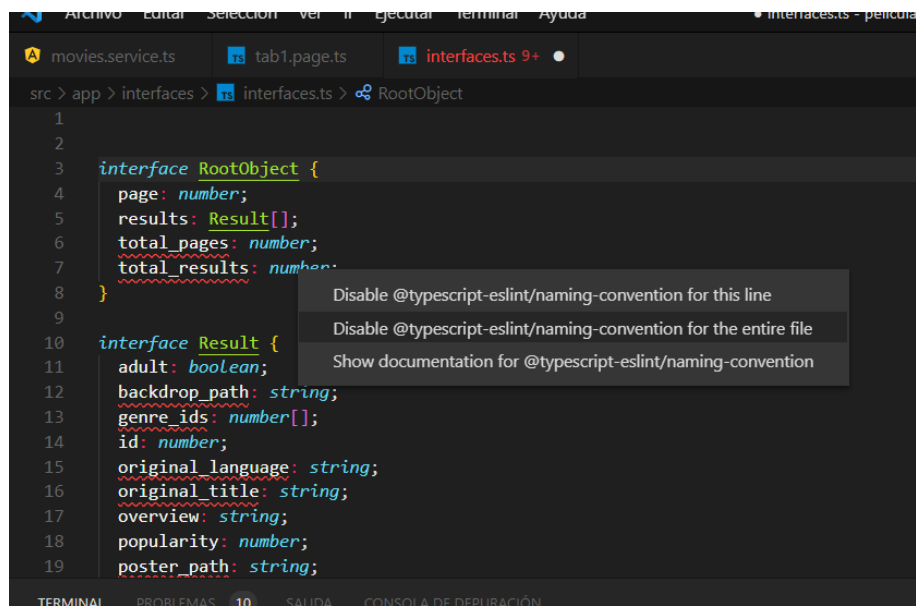


Ilustración 17: Invocamos el Plugin JSON to TS: Convert from clipboard

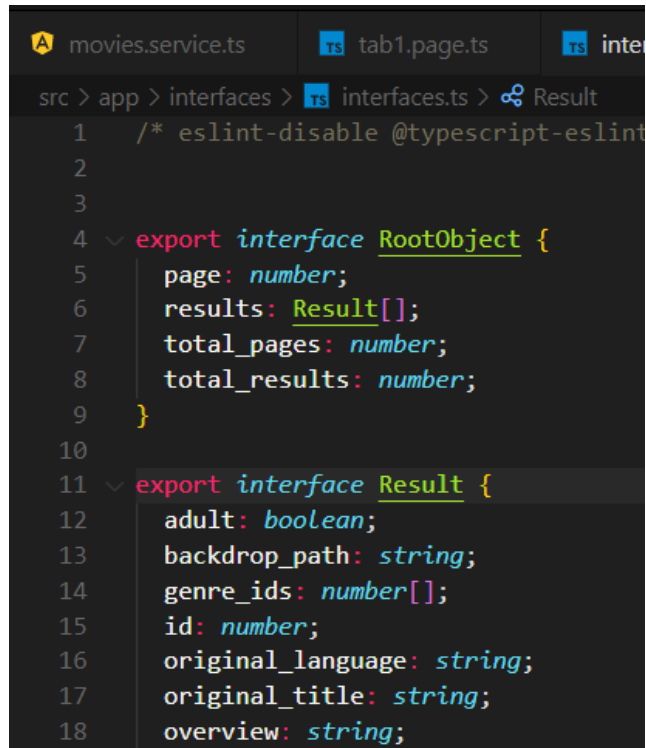
Y ya tenemos creada nuestra interfaz



Si ese error persiste, tenemos que seleccionar la opción segunda para corregir el formato

Ahora a las dos interfaces le vamos a anteponer la palabra Export

Clase 11- Proyecto práctico



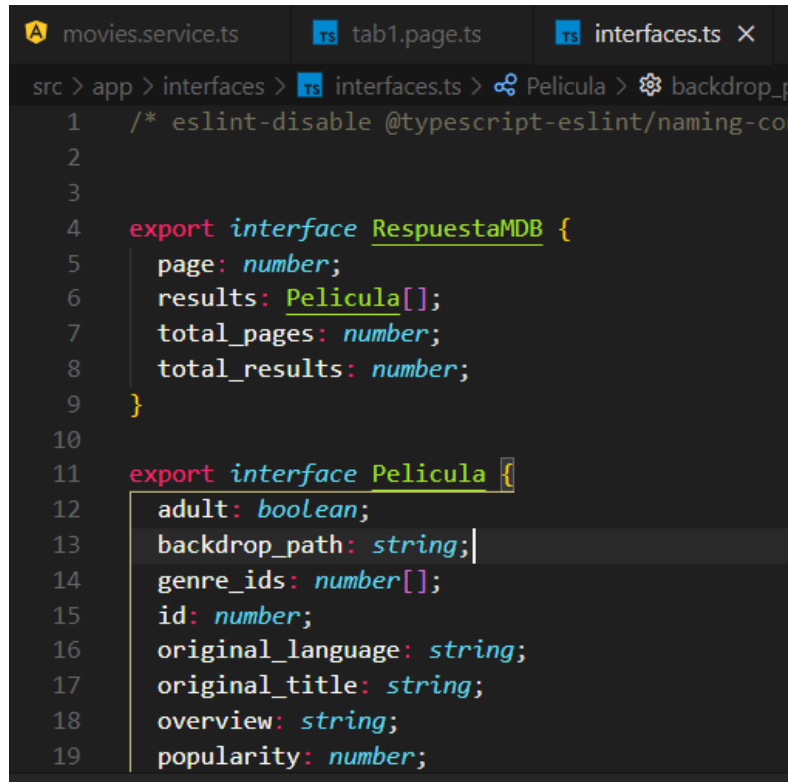
The screenshot shows a code editor with three tabs: 'movies.service.ts', 'tab1.page.ts', and 'inter'. The active tab is 'interfaces.ts', which contains the following TypeScript code:

```
src > app > interfaces > interfaces.ts > Result
1  /* eslint-disable @typescript-eslint-
2
3
4  export interface RootObject {
5      page: number;
6      results: Result[];
7      total_pages: number;
8      total_results: number;
9  }
10
11 export interface Result {
12     adult: boolean;
13     backdrop_path: string;
14     genre_ids: number[];
15     id: number;
16     original_language: string;
17     original_title: string;
18     overview: string;
```

Ilustración 18: anteponeamos la palabra Export

Acomodamos un poco el nombre al resultado que obtendremos

Clase 11- Proyecto práctico

A screenshot of a code editor with three tabs: 'movies.service.ts', 'tab1.page.ts', and 'interfaces.ts'. The 'interfaces.ts' tab is active, showing the following TypeScript code:

```
1  /* eslint-disable @typescript-eslint/naming-convention */
2
3
4  export interface RespuestaMDB {
5    page: number;
6    results: Pelicula[];
7    total_pages: number;
8    total_results: number;
9  }
10
11  export interface Pelicula {
12    adult: boolean;
13    backdrop_path: string;
14    genre_ids: number[];
15    id: number;
16    original_language: string;
17    original_title: string;
18    overview: string;
19    popularity: number;
```

Ilustración 19: Cambiamos el nombre de la interface a Pelicula

Ahora en el fichero tab1.page.ts

```
export class Tab1Page implements OnInit {

  constructor(private moviesService: MoviesService ) {}

  ngOnInit(){
    this.moviesService.getFetaures()
      .subscribe( (resp: RespuestaMDB) =>{
        console.log('Resp', resp);
      });
  }
}
```

Debemos tener este código tal cual

Clase 11- Proyecto práctico

Si nos colocamos en resp del concole.log, y colocamos punto, veremos que nos da las opciones que podemos mandar a llamar

```
8 styleUrls: [ 'tab1.page.scss' ]
9 })
10 export class Tab1Page implements OnInit {
11
12     constructor(private moviesService: MoviesService ) {}
13
14     ngOnInit(){
15         this.moviesService.getFetaures()
16         .subscribe( (resp: RespuestaMDB) =>{
17             console.log('Resp', resp.);
18         });
19     }
20 }
21
```

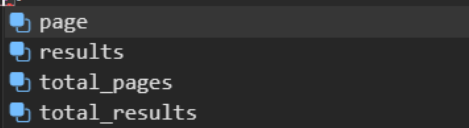


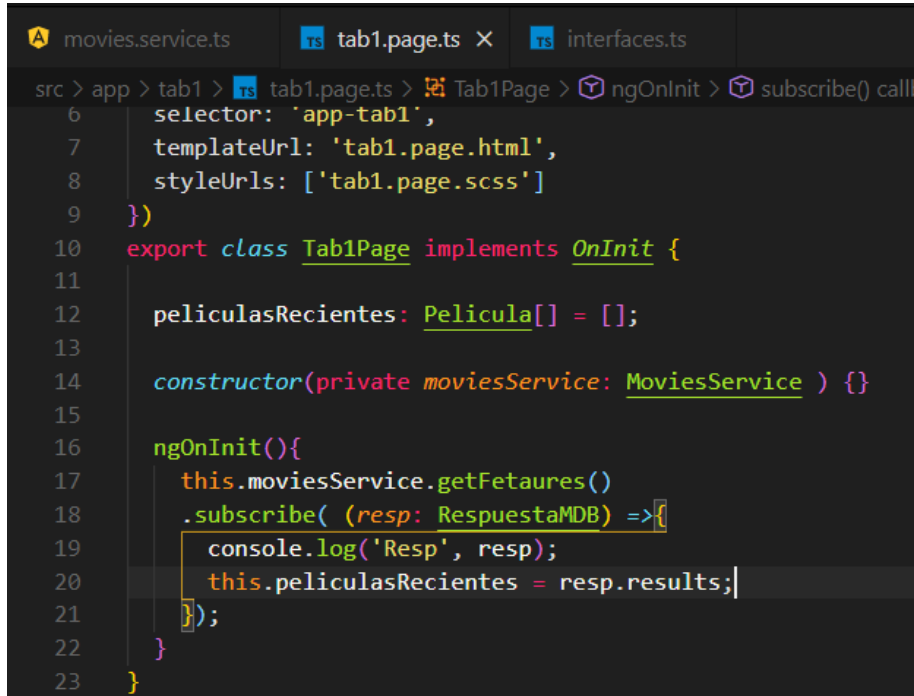
Ilustración 20: valores que podemos mandar a llamar de la propiedad

Ahora, vamos a crearnos un arreglo debajo del export class llamado peliculasRecientes, le diremos que es del tipo arreglo y lo inicializaremos vacío

```
4
5 @Component({
6     selector: 'app-tab1',
7     templateUrl: 'tab1.page.html',
8     styleUrls: ['tab1.page.scss']
9 })
10 export class Tab1Page implements OnInit {
11
12     peliculasRecientes: Pelicula[] = [];
13
14     constructor(private moviesService: MoviesService ) {}
15
16     ngOnInit(){
17         this.moviesService.getFetaures()
18         .subscribe( (resp: RespuestaMDB) =>{
19             console.log('Resp', resp.);
20         });
21     }
22 }
```

Ilustración 21: Definición del arreglo vacío películas Recientes

Clase 11- Proyecto práctico



```
src > app > tab1 > tab1.page.ts > Tab1Page > ngOnInit > subscribe() call
6   selector: 'app-tab1',
7   templateUrl: 'tab1.page.html',
8   styleUrls: ['tab1.page.scss']
9 })
10 export class Tab1Page implements OnInit {
11
12   peliculasRecientes: Pelicula[] = [];
13
14   constructor(private moviesService: MoviesService ) {}
15
16   ngOnInit(){
17     this.moviesService.getFetaures()
18     .subscribe( (resp: RespuestaMDB) =>{
19       console.log('Resp', resp);
20       this.peliculasRecientes = resp.results;
21     });
22   }
23 }
```

Ilustración 22: Obtendremos el resultado en console.log

Por el momento no hemos hecho ningún cambio significativo, pero si hemos logrado que TypeScript sepa el tipo de dato que contiene el arreglo de resultado

Guardamos cambios y todo debería seguir funcionando perfectamente.

Clase 11- Proyecto práctico

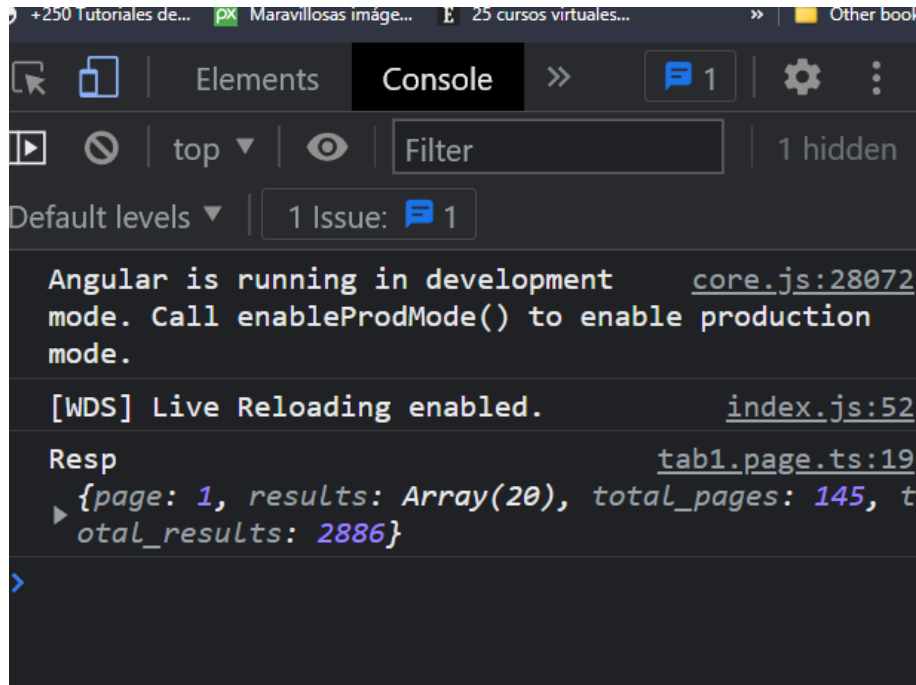


Ilustración 23: Resultado de las películas obtenidas

Notar que con esto si nos colocamos en el servicio y colocamos una posición del arreglo seguido de un punto, es posible ver el contenido del arreglo en esa posición

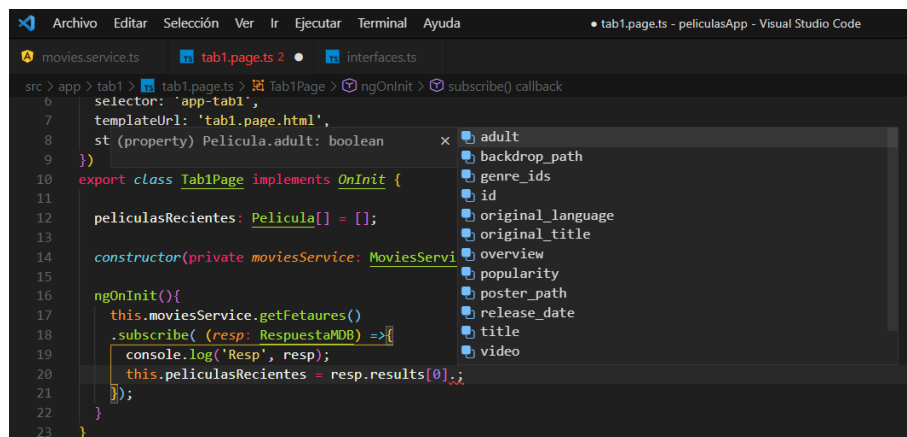


Ilustración 24: Contenido del arreglo al agregar.

Clase 11- Proyecto práctico

Parte 3:

Mostrar películas Pipe.module y pipe. Imagen

Es tiempo de mostrar información en la pagina principal

1. Abrimos el HTML de page1.page.html

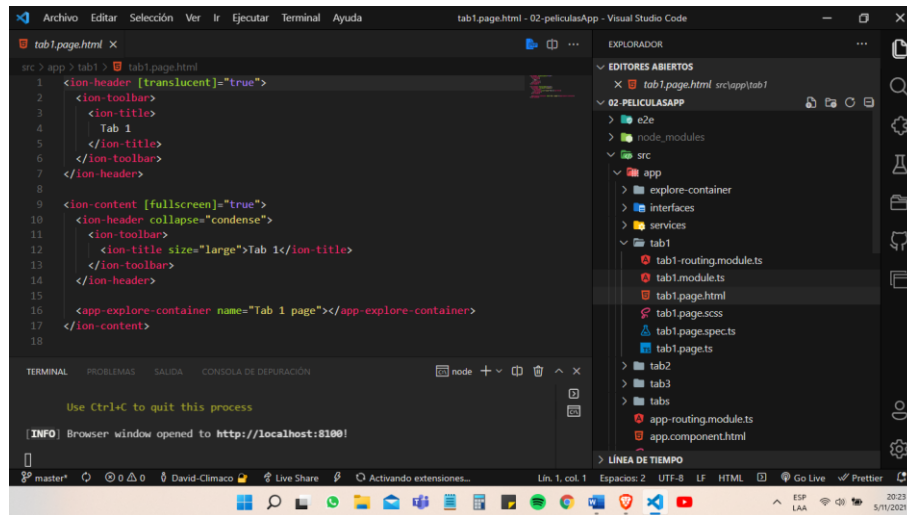


Ilustración 25: Tab1 page.html para mostrar información de las películas

Borramos todo el contenido y solamente dejaremos el <ion-content></ion-content>

Lo que vamos a crear es un slider, que nos permita ver las películas.

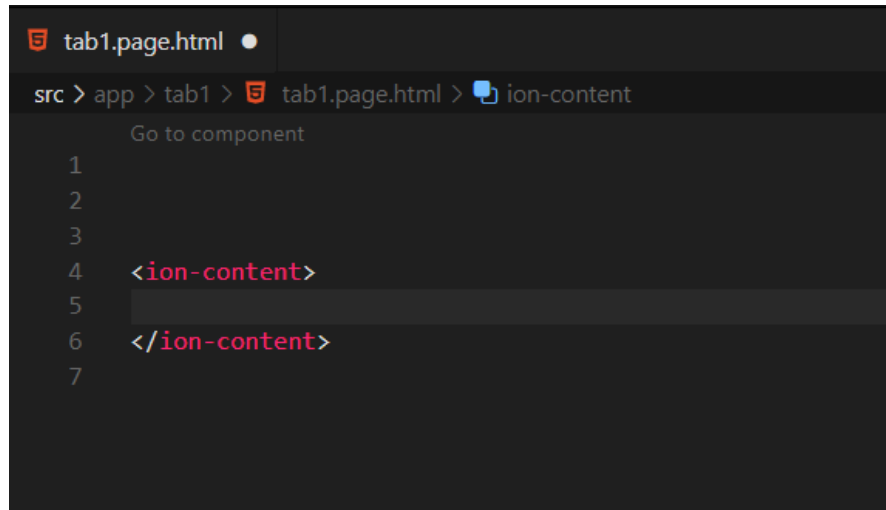
En la documentación oficial podemos revisar como hacer ese slider.

<https://ionicframework.com/docs/api/slides#usage>

<https://ionicframework.com/docs/angular/slides>

Para nuestro caso lo haremos paso a paso para mostrar por el momento mostrar al menos los nombres de las películas

Clase 11- Proyecto práctico



```
tab1.page.html
src > app > tab1 > tab1.page.html > ion-content
Go to component
1
2
3
4 <ion-content>
5
6 </ion-content>
7
```

Ilustración 26: Dejar vacía la página solamente con `ion-content`

Recuerden solamente mostraremos el nombre de la película...

Pensemos por un momento como podemos hacer esa tarea.

Clase 11- Proyecto práctico

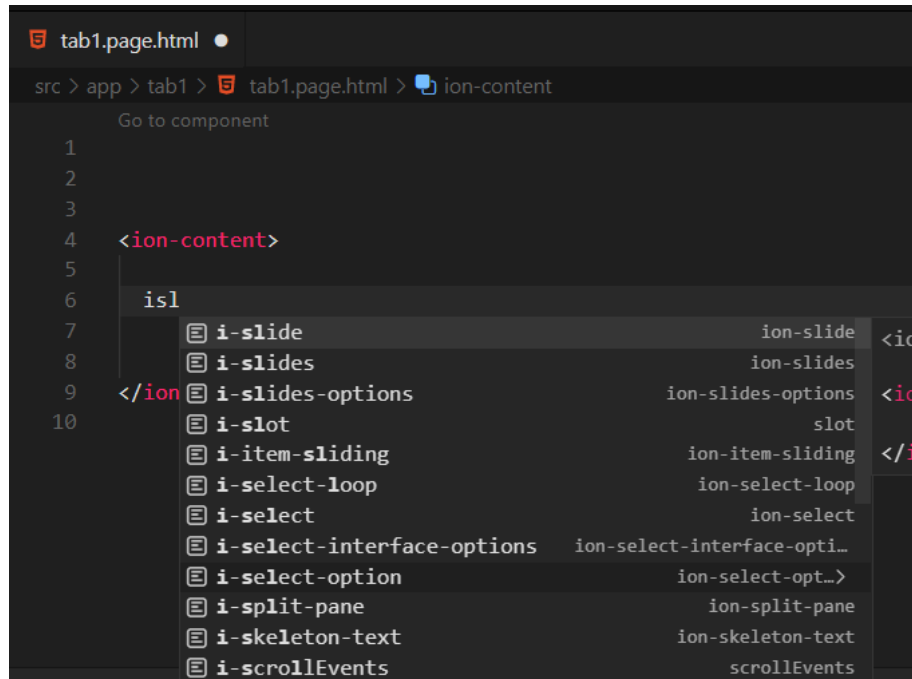


Ilustración 27: con el shortcut islides lo creamos

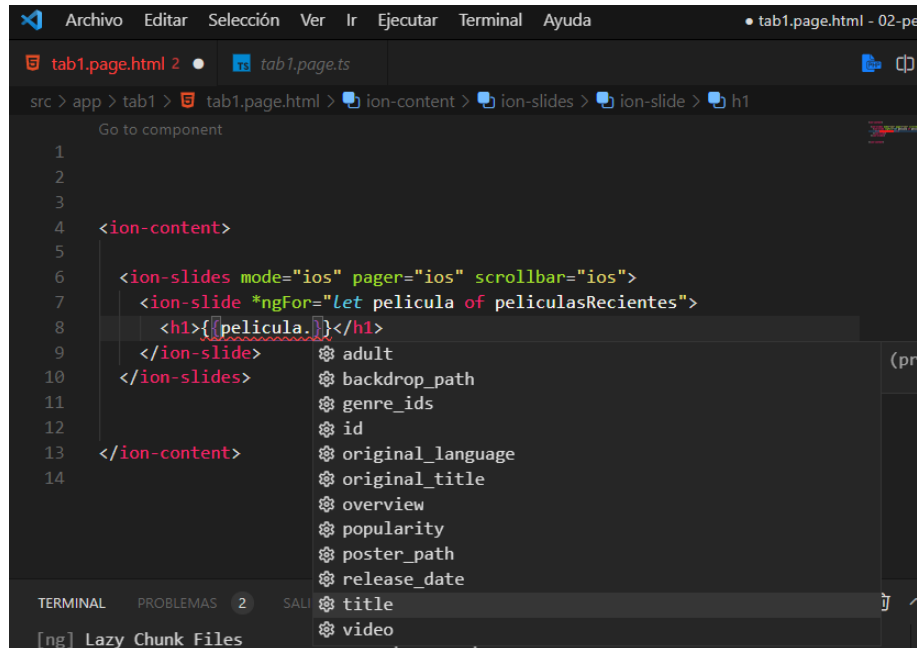
Usamos islides



Vamos a usar el ngfor para recorrer el arreglo que nos dará los títulos de las películas

Los parámetros del /ngFor serán películas of “El arreglo vacío que hicimos anteriormente”

Clase 11- Proyecto práctico

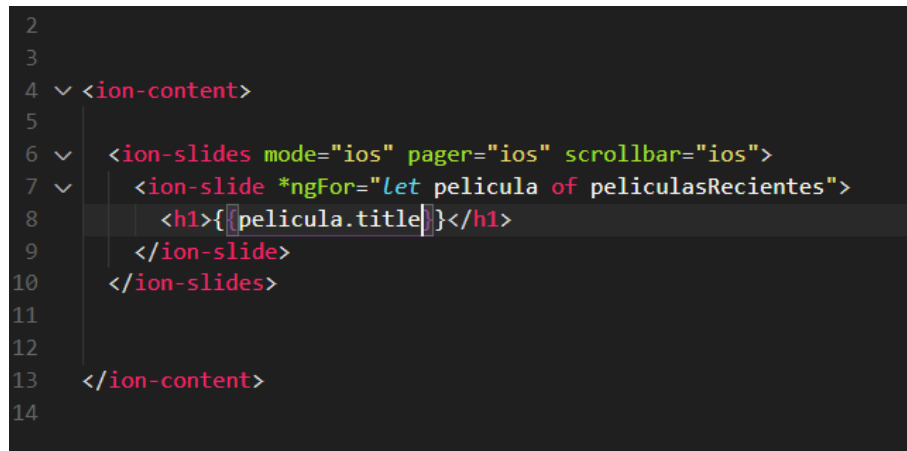


```
1
2
3
4 <ion-content>
5
6 <ion-slides mode="ios" pager="ios" scrollbar="ios">
7   <ion-slide *ngFor="let pelicula of peliculasRecientes">
8     <h1>{{pelicula.title}}</h1>
9   </ion-slide>
10 </ion-slides>
11
12 </ion-content>
13
14
```

Properties of pelicula:

- adult
- backdrop_path
- genre_ids
- id
- original_language
- original_title
- overview
- popularity
- poster_path
- release_date
- title
- video

Y en el título h1, vamos a colocar que de ese arreglo: película.title



```
2
3
4 <ion-content>
5
6 <ion-slides mode="ios" pager="ios" scrollbar="ios">
7   <ion-slide *ngFor="let pelicula of peliculasRecientes">
8     <h1>{{pelicula.title}}</h1>
9   </ion-slide>
10 </ion-slides>
11
12 </ion-content>
13
14
```

Ilustración 28: Recorremos el arreglo con ngFor

Clase 11- Proyecto práctico

Vemos el resultado

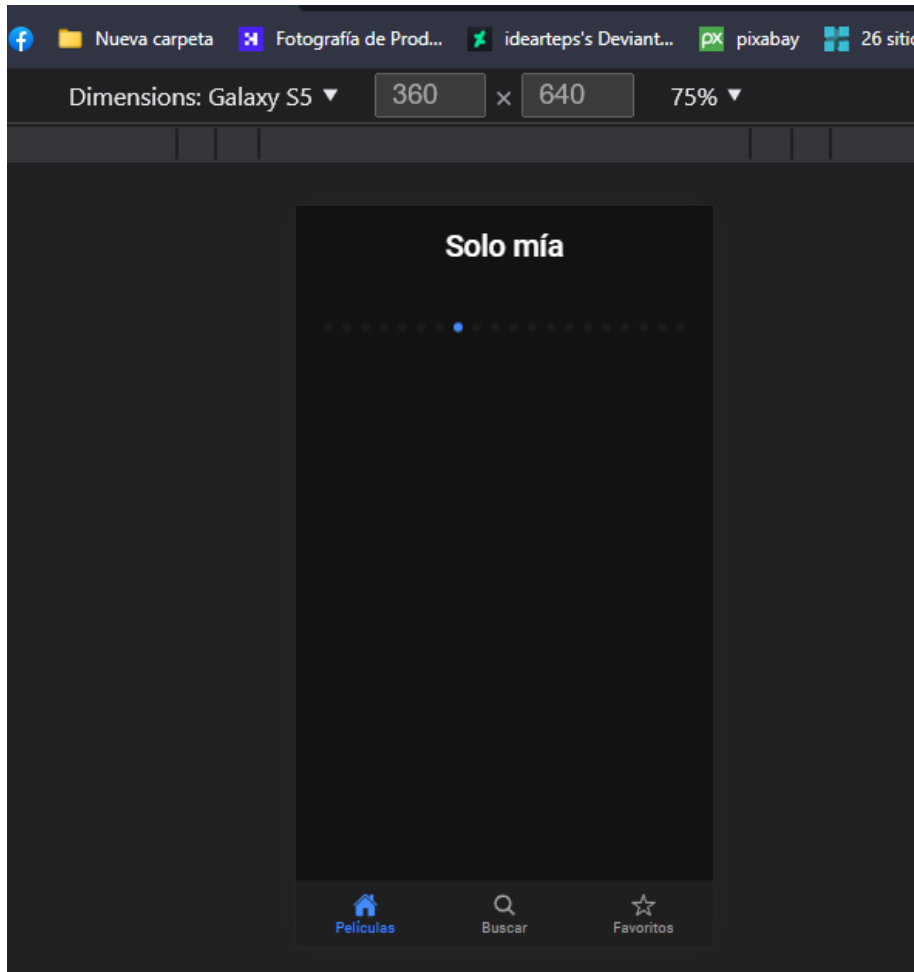


Ilustración 29: Resultado del slider

Como vemos ya nos muestra los títulos, hasta el momento un poco feo, pero en fin funciona, mas adelante podemos mejorar eso.

Mostrando las imágenes de las películas

En los resultados de consola o en postman, vean que tenemos 2 tipos de imágenes Con `backdrop_path` y con `poster_path`. Veremos cual nos conviene utilizar

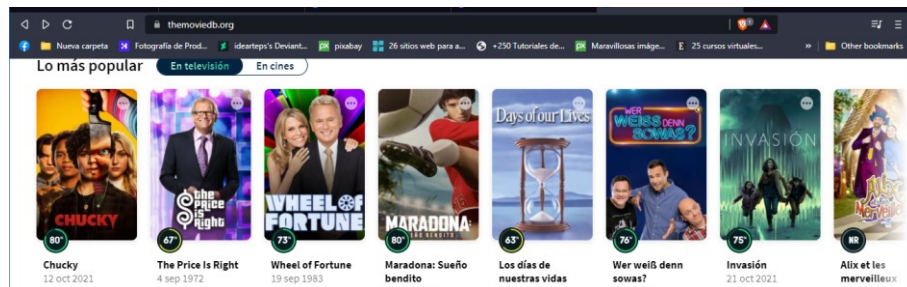


Ilustración 30: resultados de las imágenes

Si en lugar de la propiedad `title` que estamos mostrando, colocamos `backdrop_path`. Lo que nos va a colocar será un URL incompleto, como el que aparece en la consola, resultado de cada película

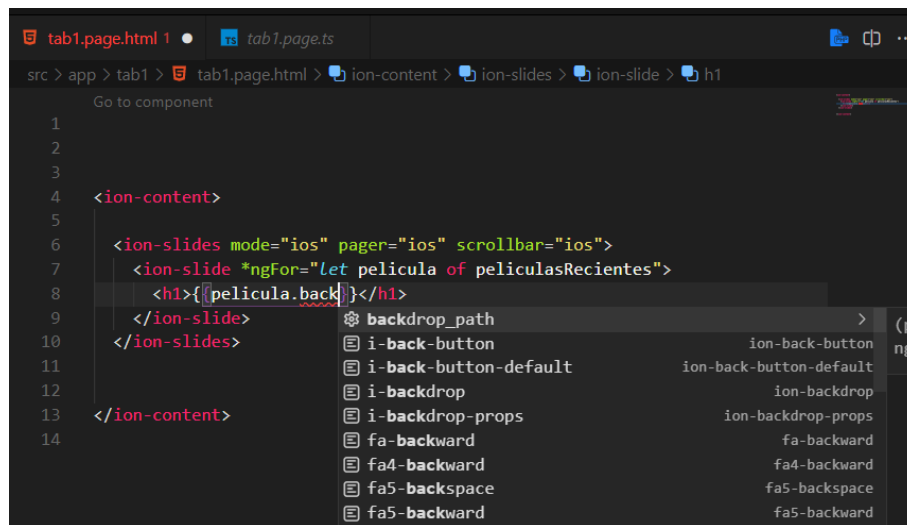


Ilustración 31: colocando propiedad `backdrop_path`

Clase 11- Proyecto práctico

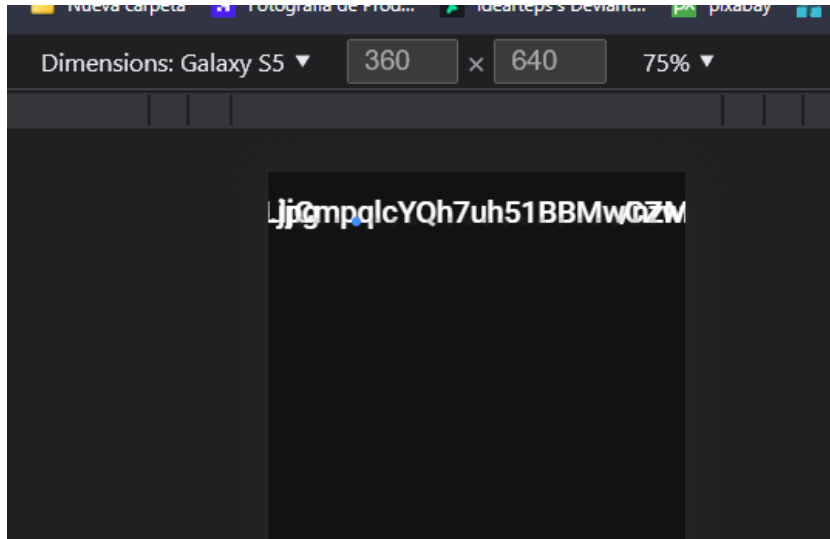


Ilustración 32: resultado de mostrar el `backdrop_path`

Ahora veamos en ThemovieDB, como vienen esas URL para mostrar resultados de imágenes.

Ojo con la construcción de esa URL, nos dirigiremos a la sección de imágenes de themoviedb.

<https://developers.themoviedb.org/3/getting-started/introduction>

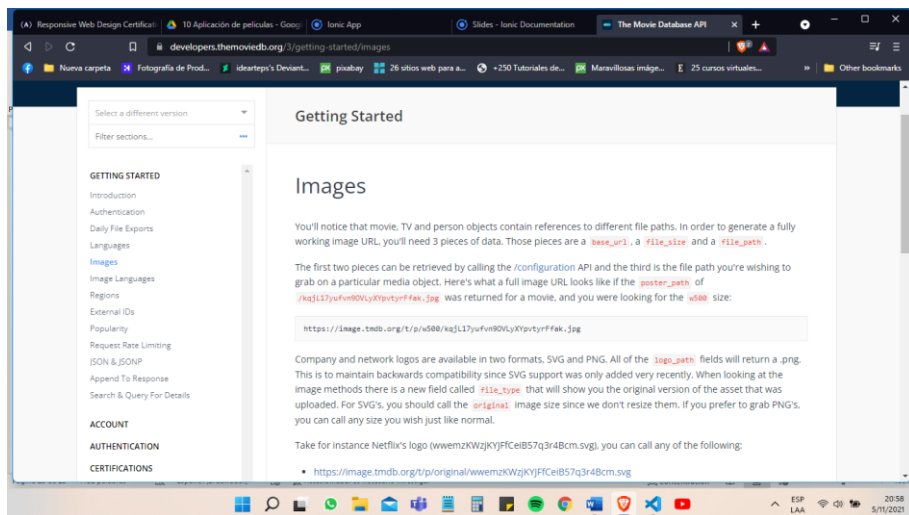


Ilustración 33: Sección de imágenes de themoviedb

Clase 11- Proyecto práctico

Copiaremos la URL que aparece en el ejemplo en postman para ver el resultado, copiamos la de Netflix que aparece al final

`https://image.tmbd.org/t/p/w500/wwemzKWzjKYJFfCeIB57q3r4Bcm.png`

Noten la construcción de la URL, la W500 es el tamaño de imagen que devolverá, seguido nos da la ruta de la imagen que cada resultado obtenido al consumir la API.

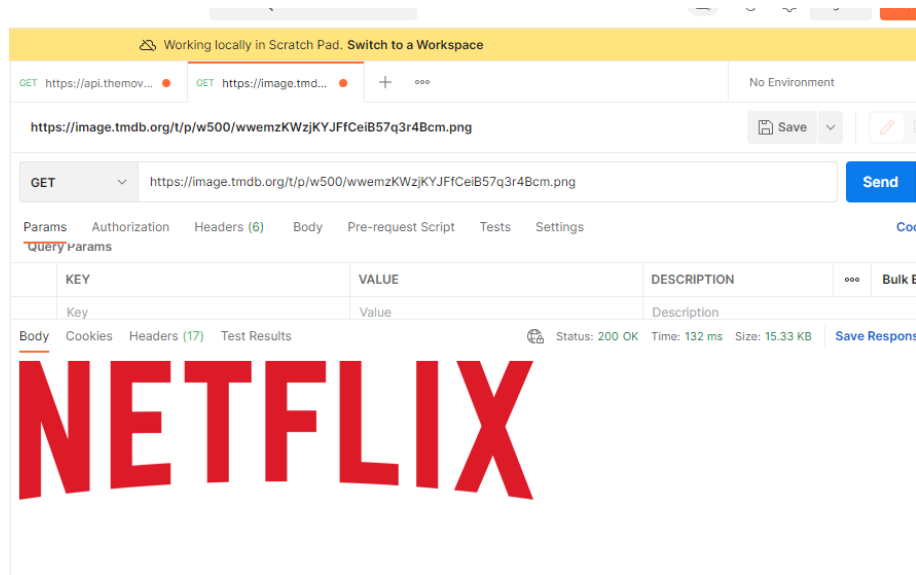


Ilustración 34: resultado en postman de la búsqueda de la imagen

Notar que la ruta resultado que nos da en console.log nuestra API, forma parte de la URL completa para mostrar el resultado...

Ahora vamos a crear en nuestra APP esa URL (Copiaremos la usada en Postman como referencia)

En este punto vamos a implementar un PIPE

¿Qué es un PIPE?

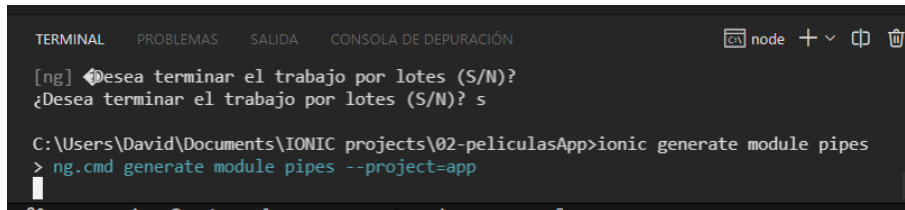
Pues un Pipe, nos ayuda a mostrar alguna información o resultado de un aspecto mas sencilla o conveniente a criterio personal, es decir si obtenemos una fecha en formato, DDMMYYHHMMSS, nosotros podemos usar un pipe para mostrar YYMMDD, o algo así.

Es mas fácil hacerlo que explicarlo...

Clase 11- Proyecto práctico

Entonces en consola vamos a crear un módulo que nos permita almacenar nuestro pipes

ionic generate module pipe



```
TERMINAL  PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  node + v  [icon] [icon] [icon]

[ng] ¿Desea terminar el trabajo por lotes (S/N)?
¿Desea terminar el trabajo por lotes (S/N)? s

C:\Users\David\Documents\IONIC projects\02-peliculasApp>ionic generate module pipes
> ng.cmd generate module pipes --project=app
```

Ilustración 35: Creando el modulo pipes

Nos creo un fichero llamado pipes

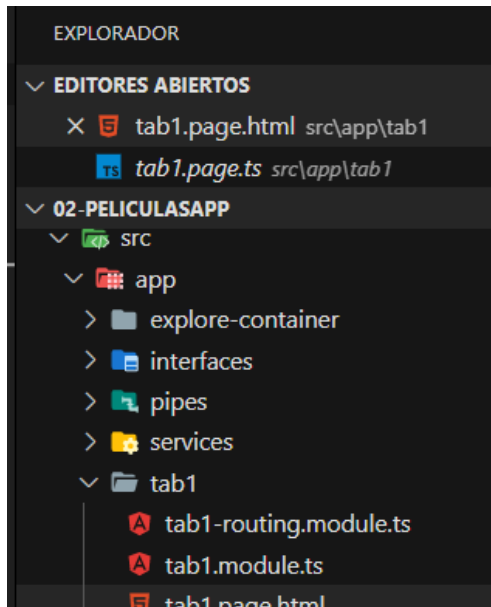
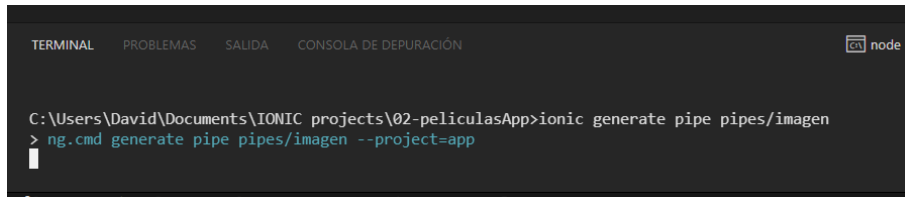


Ilustración 36: Fichero pipe creado

Clase 11- Proyecto práctico

Ahora crearemos nuestro pipe/

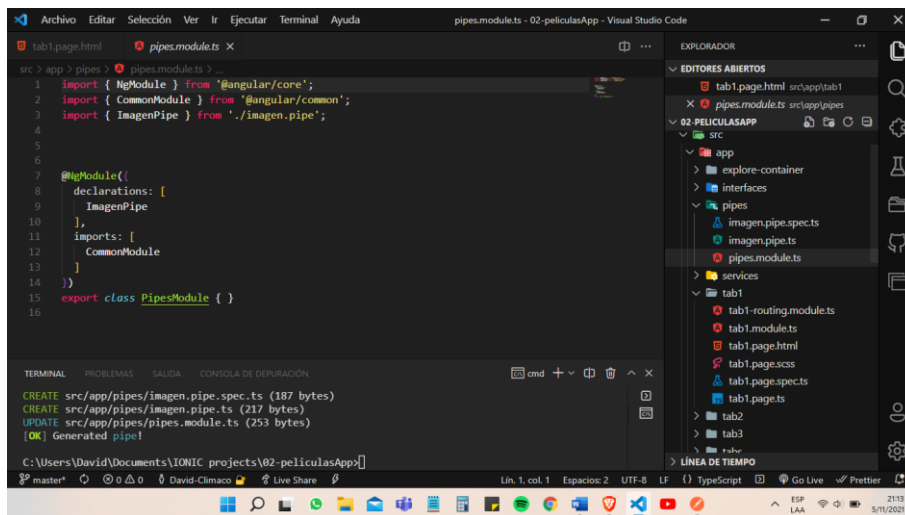


```
TERMINAL  PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  node

C:\Users\David\Documents\IONIC projects\02-peliculasApp>ionic generate pipe pipes/imagen
> ng.cmd generate pipe pipes/imagen --project=app
```

Ilustración 37: Generando el pipe/imagen dentro del fichero pipes

Ahora vamos al módulo del pipe creado



```
Archivo  Editar  Selección  Ver  Ir  Ejecutar  Terminal  Ayuda  pipes.module.ts - 02-peliculasApp - Visual Studio Code

src > app > pipes > pipes.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { ImagenPipe } from './imagen.pipe';
4
5
6
7  @NgModule({
8    declarations: [
9      ImagenPipe
10     ],
11    imports: [
12      CommonModule
13     ]
14  })
15  export class PipesModule { }
16
```

TERMINAL PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN

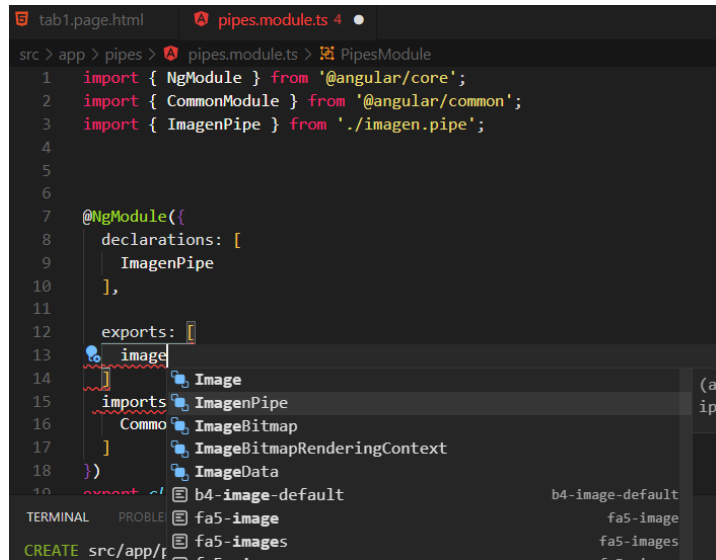
```
cmd + ↵  +  -  ^  x
CREATE src/app/pipes/imagen.pipe.spec.ts (187 bytes)
CREATE src/app/pipes/imagen.pipe.ts (217 bytes)
UPDATE src/app/pipes/pipes.module.ts (253 bytes)
[OK] Generated pipe!

C:\Users\David\Documents\IONIC projects\02-peliculasApp>
```

Ilustración 38: Módulo del pipe/image creado

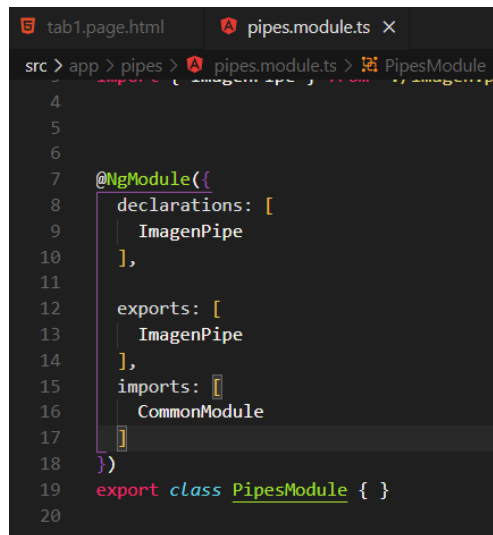
Como vamos a utilizar este pipe fuera del módulo, vamos a declarar un **Export**. El **imagenPipe**

Clase 11- Proyecto práctico



```
src > app > pipes > pipes.module.ts > PipesModule
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { ImagenPipe } from './imagen.pipe';
4
5
6
7 @NgModule({
8   declarations: [
9     ImagenPipe
10  ],
11
12   exports: [
13     ImagenPipe
14  ],
15   imports: [
16     CommonModule,
17     Image,
18     ImageBitmap,
19     ImageBitmapRenderingContext,
20     ImageData
21  ]
22 })
23 export class PipesModule { }
```

Ilustración 39: Imagen Pipe exportado

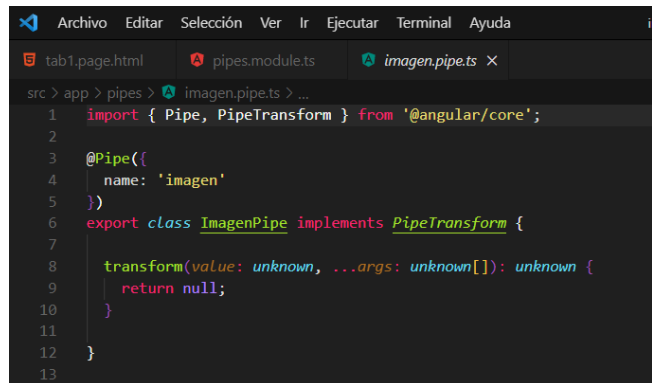


```
src > app > pipes > pipes.module.ts > PipesModule
4
5
6
7 @NgModule({
8   declarations: [
9     ImagenPipe
10  ],
11
12   exports: [
13     ImagenPipe
14  ],
15   imports: [
16     CommonModule
17  ]
18 })
19 export class PipesModule { }
```

Ilustración 40: Imagenpipe exportado

Clase 11- Proyecto práctico

Ahora vamos al fichero donde vamos a trabajar nuestra lógica: imagen.pipe.ts, aquí para retornar una imagen que tenga el tipo de estilo que deseamos

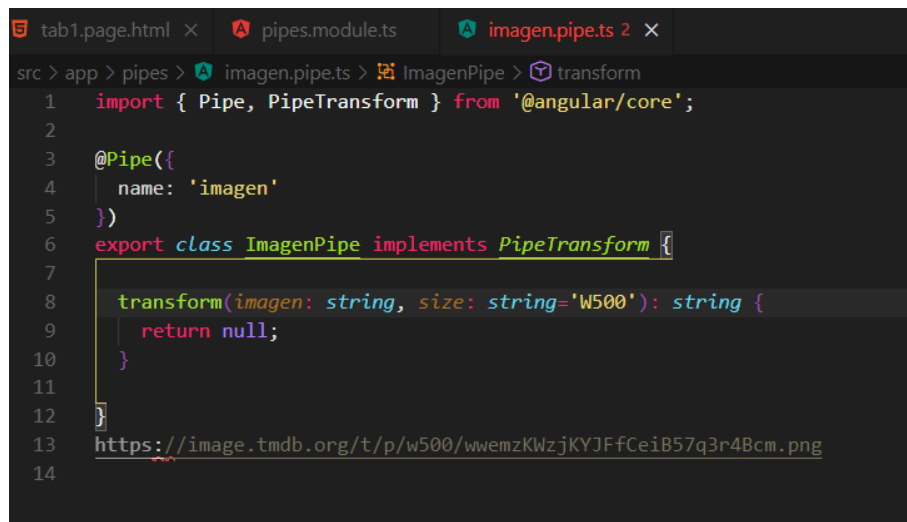


```
1 import { Pipe, PipeTransform } from '@angular/core';
2
3 @Pipe({
4   name: 'imagen'
5 })
6 export class ImagenPipe implements PipeTransform {
7
8   transform(value: unknown, ...args: unknown[]): unknown {
9     return null;
10  }
11
12 }
13
```

Ilustración 41: Fichero imagen.pipe.ts

Vamos a copiar la URL de Postman como ejemplo de lo que tenemos que construir.

1. Vamos a recibir una imagen que es del tipo string.
2. Es conveniente tomar el size. String (El tamaño de la imagen que desea el usuario)

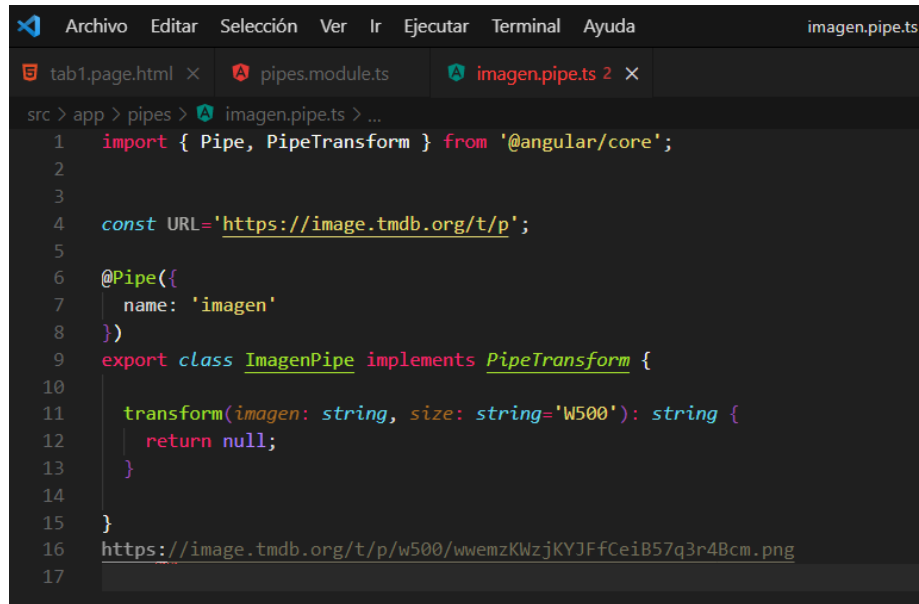


```
1 import { Pipe, PipeTransform } from '@angular/core';
2
3 @Pipe({
4   name: 'imagen'
5 })
6 export class ImagenPipe implements PipeTransform {
7
8   transform(imagen: string, size: string='W500'): string {
9     return null;
10  }
11
12 }
13 https://image.tmbd.org/t/p/w500/wwemzKWzjKYJFFCeIB57q3r4Bcm.png
14
```

Ilustración 42: Construimos la URL del resultado

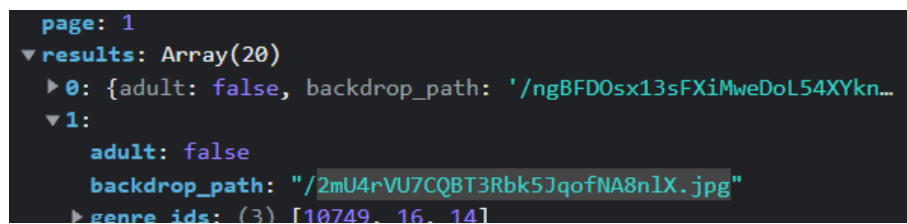
Ya que empezamos a acomodar la URL de esa imagen, declararemos una constante llamada URL

Clase 11- Proyecto práctico



```
1 import { Pipe, PipeTransform } from '@angular/core';
2
3
4 const URL='https://image.tmdb.org/t/p';
5
6 @Pipe({
7   name: 'imagen'
8 })
9 export class ImagenPipe implements PipeTransform {
10
11   transform(imagen: string, size: string='W500'): string {
12     return null;
13   }
14
15 }
16 https://image.tmdb.org/t/p/w500/wwemzKWzjKYJfCeIB57q3r4Bcm.png
17
```

Ojo con el slash, porque ya las imágenes en los resultados comienzan con slash



```
page: 1
▼ results: Array(20)
  ► 0: {adult: false, backdrop_path: '/ngBFD0sx13sFXiMweDoL54XYkn...'
    ▼ 1:
      adult: false
      backdrop_path: "/2mU4rVU7CQBT3Rbk5JqofNA8n1X.jpg"
      ► genre_ids: (3) [10749, 16, 14]
```

Ilustración 43: Direccion de backdrop_path

Ahora lo que puedo hacer en la lógica, es preguntar si existe una Imagen, sino existe una imagen que se salga y no haga nada (por el momento)

Clase 11- Proyecto práctico

```
3
4   const URL='https://image.tmbd.org/t/p';
5
6   @Pipe({
7     name: 'imagen'
8   })
9   export class ImagenPipe implements PipeTransform {
10
11     transform(imagen: string, size: string='W500'): string {
12
13       if(!imagen){
14         return;
15       }
16
17       return null;
18
19     }
20   }
```

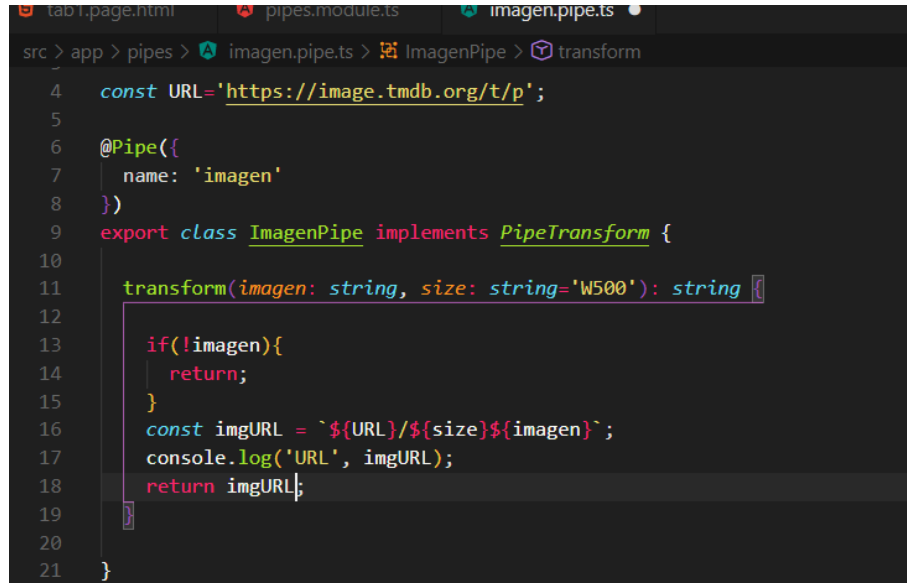
Pero si existe una imagen, voy a crearme una constante que almacene la URL construida así.

```
tab1.page.html x pipes.module.ts imagen.pipe.ts 2 x
src > app > pipes > imagen.pipe.ts > ImagenPipe > transform
1   import { Pipe, PipeTransform } from '@angular/core';
2
3
4   const URL='https://image.tmbd.org/t/p';
5
6   @Pipe({
7     name: 'imagen'
8   })
9   export class ImagenPipe implements PipeTransform {
10
11     transform(imagen: string, size: string='W500'): string {
12
13       if(!imagen){
14         return;
15       }
16       const imgURL = `${URL}/${size}${imagen}`;
17       return null;
18
19     }
20   }
```

Ilustración 44: Constuyendo la URL

Y aun podemos colocar un URL para ver los resultado en console.log y el utimo return debe ser imgURL

Clase 11- Proyecto práctico

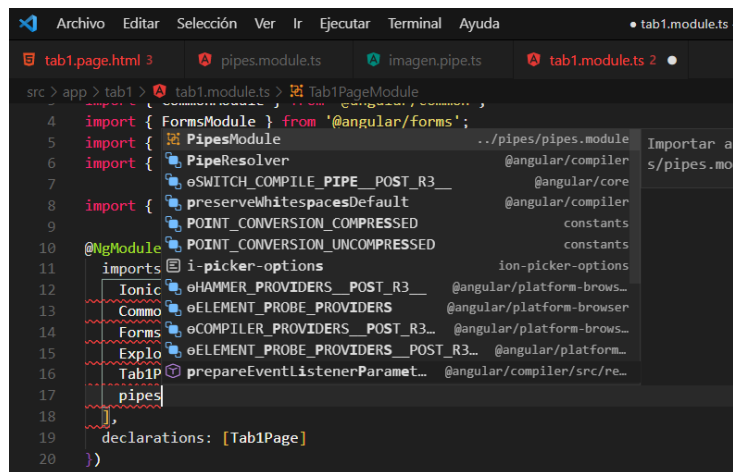


```
src > app > pipes > imagen.pipe.ts > ImagePipe > transform
4  const URL='https://image.tmbd.org/t/p';
5
6  @Pipe({
7    name: 'imagen'
8  })
9  export class ImagePipe implements PipeTransform {
10
11    transform(imagen: string, size: string='W500'): string {
12
13      if(!imagen){
14        return;
15      }
16      const imgURL = `${URL}/${size}${imagen}`;
17      console.log('URL', imgURL);
18      return imgURL;
19    }
20
21  }
```

Ilustración 45: Console log de nuestra URL

¿Ya está hecho, pero como utilizamos este pipe?

Primero en nuestro tabs1.module.ts, importemos nuestro PipesModule



```
Archivo  Editar  Selección  Ver  Ir  Ejecutar  Terminal  Ayuda  • tabs1.module.ts
tab1.page.html 3  pipes.module.ts  imagen.pipe.ts  tabs1.module.ts 2
src > app > tab1 > tabs1.module.ts > Tab1PageModule
4  import { FormsModule } from '@angular/forms';
5  import { PipesModule } from '../pipes/pipes.module';
6  import { PipeResolver } from '@angular/compiler';
7  import { eSWITCH_COMPILE_PIPE_POST_R3 } from '@angular/core';
8  import { preserveWhitespacesDefault } from '@angular/compiler';
9  import { POINT_CONVERSION_COMPRESSED } from '@angular/core';
10 import { POINT_CONVERSION_UNCOMPRESSED } from '@angular/core';
11 @NgModule({
12   imports: [i-picker-options, ion-picker-options],
13   providers: [eHAMMER_PROVIDERS_POST_R3, eELEMENT_PROBE_PROVIDERS, eCOMPILER_PROVIDERS_POST_R3, eELEMENT_PROBE_PROVIDERS_POST_R3, prepareEventListenerParamet...],
14   declarations: [Tab1Page]
15 })
16 export class Tab1PageModule {
17   pipes
18 }
19
20 })
```

Ilustración 46: Importando nuestro PipeModule dentro de tabs1.module.ts7

Nos aseguramos que se importe arriba,

Clase 11- Proyecto práctico

Guardamos los cambios

Y en la vista, en lugar del H1, crearemos un ion-card

Y allí dentro vamos a crear un img, con el src la propiedad de la película.backdrop.path.

Espacio | y le paso el pipe de la imagen.

```
<img [src]="pelicula.backdrop_path | imagen" alt="no-img">
```

Resultado

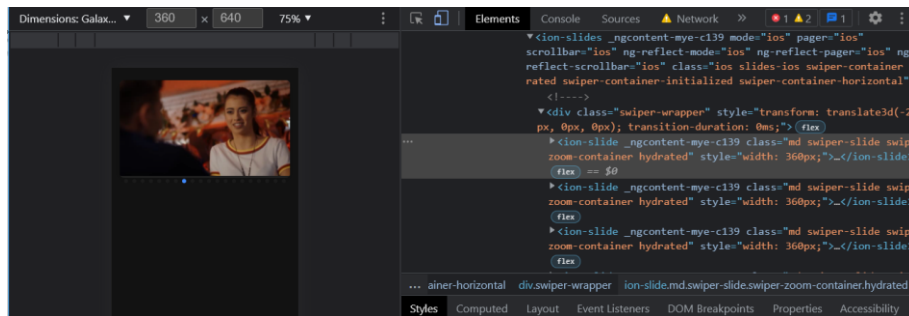


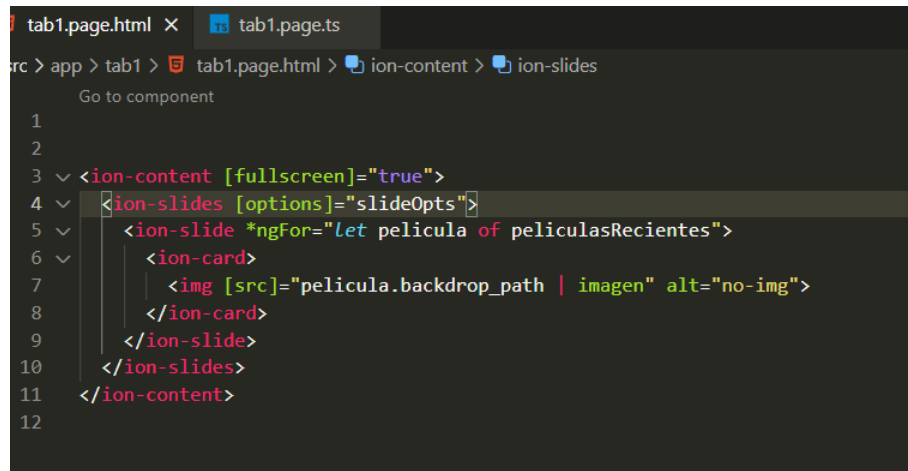
Ilustración 47: Resultado un slide

Al arrastrar a la derecha o izquierda debe poder verse las imágenes cargadas

Vean que los slides se adaptan al tamaño de pantalla perfectamente

¿Pero cómo sabría el usuario que hay otra imagen para ver?

Para ello vamos a modificar los slides un poco colocando unas opciones slideOpts



Clase 11- Proyecto práctico

Y esas opciones las vamos a configurar en tab1.page.ts y vamos a decir que slideOpts es un objeto, con slidesPerView (respetar mayúsculas y minúsculas) tendrá 1.1, y freeMode = true.

```
export class Tab1Page implements OnInit {  
  
  peliculasRecientes: Pelicula[] = [];  
  
  slideOpts = {  
    slidesPerView: 1.1,  
    freeMode: true  
  };  
};
```

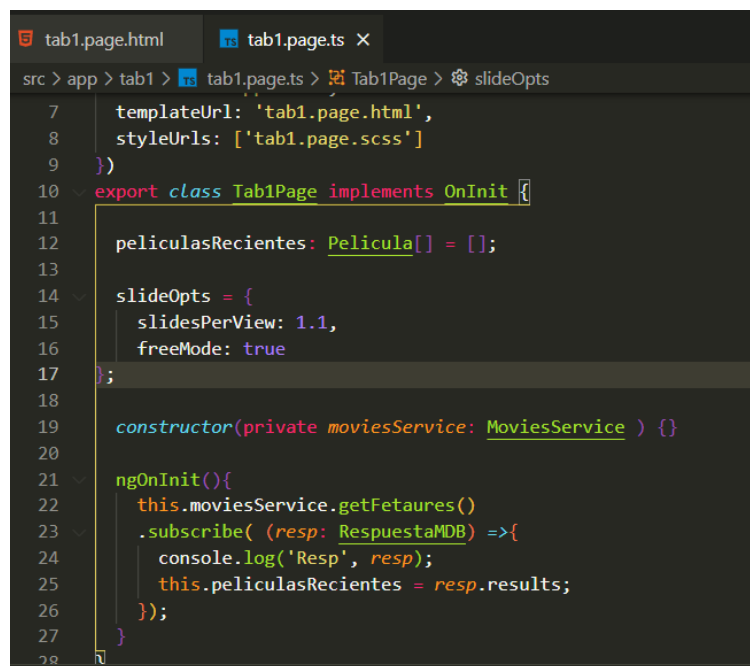


Ilustración 48; Parámetros completos en el ts

Resultado, que se ve un 10% de la próxima imagen.

Si modificamos el 1.1 por 1.3 notaremos que se visualiza mayor parte del próxima imagen

Por el momento quedamos así,-

Clase 11- Proyecto práctico