



UA.MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE
PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Ionic v8 – Http Client

1. Directiva *ngIf*
2. Directiva *ngFor*
3. *Binding* de variables
4. *Http Client*

- La directiva `*ngIf` es una directiva estructural que permite validar una condición.

```
<div *ngIf="isShown">  
  <h1>Welcome!</h1>  
</div>
```

En la clase `.ts` tendríamos que definir la variable `"isShown"` de tipo booleano

- Esta directiva modifica la estructura de la página añadiendo o **quitando** elementos:
 - Si la condición se cumple se añadirá un bloque de código a la vista.
 - Si la condición no se cumple se **quitará** el bloque de código.
- La condición no se cumplirá cuando:
 - La condición se evalúe a falso o la variable tenga un valor falso.
 - La variable sea `"undefined"`.

- La directiva `ngIf` permite añadir un código a ejecutar en caso de que **no se cumpla** la condición (cláusula “else”).
- Para esto utilizaremos la etiqueta “`ng-template`”:

```
<div *ngIf="isShown; else otroCaso">
  <h1>¡Se ha cumplido el if!</h1>
</div>

<ng-template #otroCaso>
  <h1>No se ha cumplido la condición</h1>
</ng-template>
```

- Mediante la directiva ***ngFor** podemos crear bucles que repitan un bloque de código:

```
<ul>
  <li *ngFor="let item of listItems">
    {{ item }}
  </li>
</ul>
```

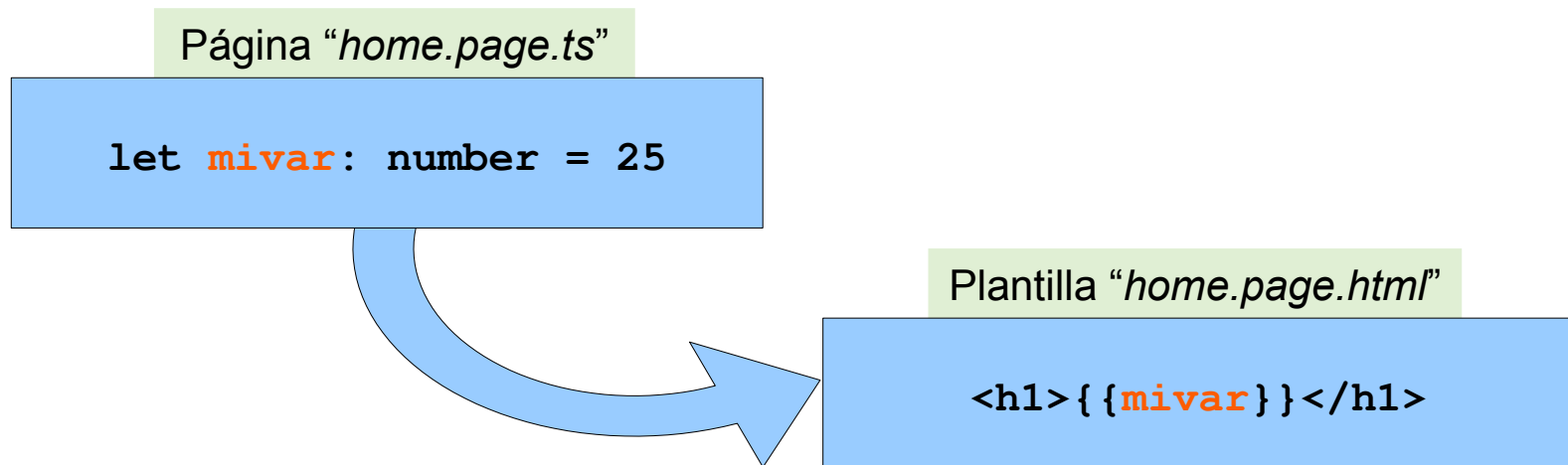
- En el controlador asociado (fichero `.ts`) tendremos que definir la variable `listItems` de tipo array.
- Esta directiva **repetirá la propia etiqueta sobre la que se añada** (la etiqueta `` en el ejemplo anterior).
- La variable `item` solo será accesible desde dentro del bucle.

- Si lo necesitamos podemos obtener el índice de la iteración:

```
<ul>
  <li *ngFor="let item of listItems; index as id">
    {{ id }} - {{ item }}
  </li>
</ul>
```

- En este ejemplo dentro del bucle podemos consultar tanto la variable “`item`” como la variable “`id`” con el índice.
- El índice empezará en 0.

- El *binding* por **interpolación** nos permite mostrar datos del componente en la plantilla.
- Simplemente tenemos que indicar en la plantilla el mismo nombre de variable entre llaves dobles.



La variable definida en el controlador podrá ser de **cualquier tipo**, incluso de tipos complejos:

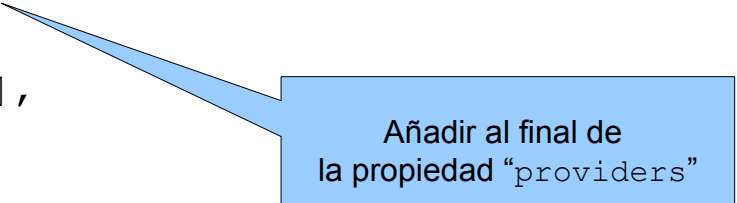
<u>Controlador</u>	<u>Vista</u>	<u>Resultado</u>
mivar=5;	→ {{mivar}}	→ 5
	→ {{otravar}}	→ ¡¡ERROR!!
mivar="Hola";	→ {{mivar}}	→ Hola
	→ {{mivar[0]}}	→ H
mivar=true;	→ {{mivar}}	→ true
mivar=[1, 2, 3, 4];	→ {{mivar}}	→ 1, 2, 3, 4
	→ {{mivar[0]}}	→ 1
	→ {{mivar[50]}}	→ // OK!
mivar={prop: 5};	→ {{mivar}}	→ [object Object]
	→ {{mivar.prop}}	→ 5
	→ {{mivar.otraprop}}	→ // OK!
	→ {{otravar.prop}}	→ ¡¡ERROR!!
	→ {{mivar?.prop}}	→ // OK!

HTTP CLIENT

- Para poder utilizar la librería **HttpClient** primero tenemos que añadirla al módulo principal de la aplicación (`src/app/app.module.ts`):

```
...
import { provideHttpClient } from '@angular/common/http';

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



Añadir al final de la propiedad "providers"

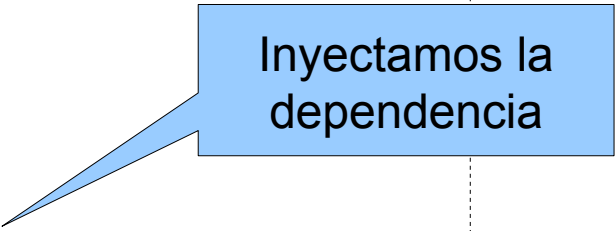
- Una vez añadido al módulo principal ya podremos utilizar esta librería desde cualquier componente, página o servicio.
- Por ejemplo, en un servicio:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class PeliculasAPIService {

  constructor(public http: HttpClient) { }

}
```



Inyectamos la
dependencia

- Una vez inyectada la clase `HttpClient` ya podremos utilizarla.
- Para solicitar contenido a una URL de Internet utilizaremos su método “get”:

```
this.http.get("http://...").subscribe ({  
    next: (result:any) => {  
        console.log(result);  
        this.datos = result  
    }  
});
```

- El método “subscribe” permite realizar una petición **asíncrona** y asignar los datos cuando se obtengan.
- Problemas con CORS en navegadores → Extensión “CORS everywhere”.

- También podemos añadir la opción “complete” para monitorizar el fin de la petición y “error” para obtener posibles errores en las peticiones:

```
this.http.get("http://...").subscribe({  
  next: (result:any) => console.log(result),  
  error: (err:any) => console.log(err),  
  complete: () => console.info('Petición finalizada')  
});
```

- La variable “err” contendrá una serie de propiedades con información del error:
 - status: 404,
 - statusText: "Not Found",
 - url: "http:...",
 - ok: false,
 - name: "HttpErrorResponse",
 - message: "Http failure response for..."

¿PREGUNTAS?