



UA.MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE
PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Ionic v4 – 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 no exista.

- La directiva `ngIf` permite añadir un código a ejecutar en caso de que no se cumpla la condición (clausula “else”).
- Para esto tenemos que utilizar el siguiente código:

```
<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>
```

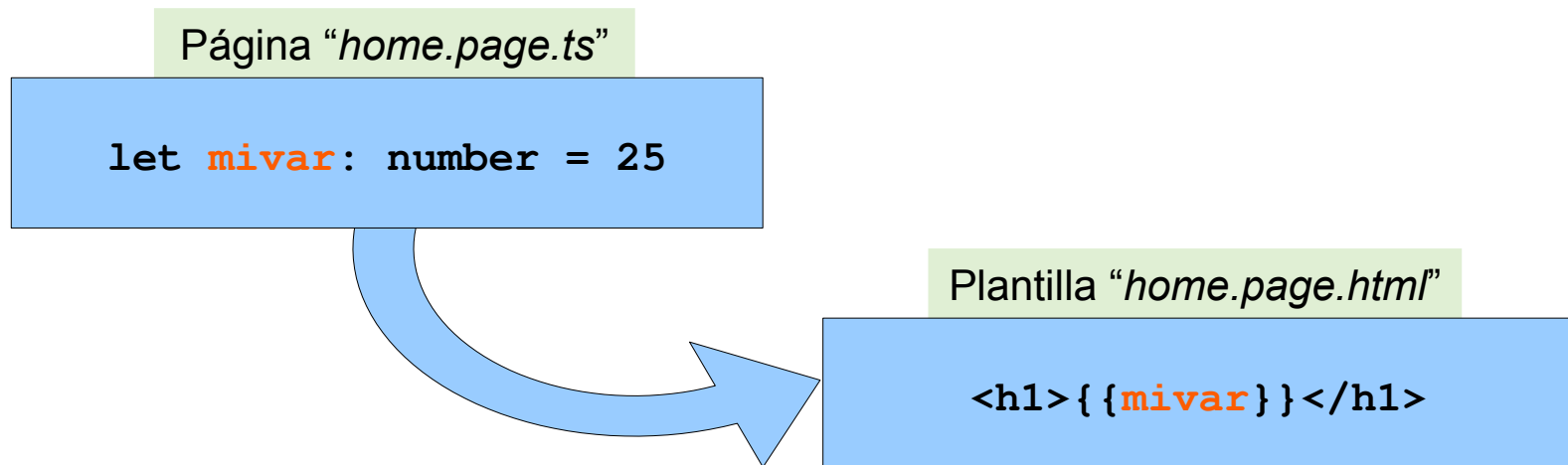
- En el controlador asociado (fichero `.ts`) tendremos que definir la variable `listItems` de tipo array.
- Esta directiva repetirá la etiqueta sobre la que se añada (la etiqueta `` en el ejemplo anterior).
- La variable `item` solo será accesible desde dentro del bucle.

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

```
<ul>
  <li *ngFor="let item of listItems; index as id">
    {{ 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>
<code>mivar=5;</code>	→ <code>{{mivar}}</code>	→ 5
	→ <code>{{otravar}}</code>	→ // OK!
<code>mivar="Hola";</code>	→ <code>{{mivar}}</code>	→ Hola
<code>mivar=true;</code>	→ <code>{{mivar}}</code>	→ true
<code>mivar=[1, 2, 3, 4];</code>	→ <code>{{mivar}}</code>	→ 1, 2, 3, 4
	→ <code>{{mivar[0]}}</code>	→ 1
	→ <code>{{mivar[50]}}</code>	→ // OK!
<code>mivar={prop: 5};</code>	→ <code>{{mivar}}</code>	→ [object Object]
	→ <code>{{mivar.prop}}</code>	→ 5
	→ <code>{{mivar.otro}}</code>	→ ERROR!!
	→ <code>{{mivar?.otro}}</code>	→ // OK!

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

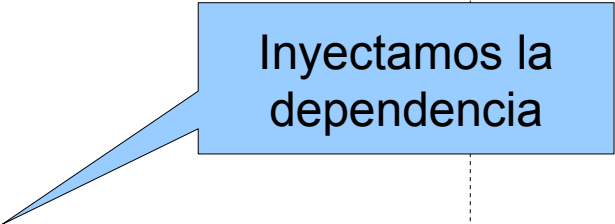
```
...
import { HttpClientModule } from '@angular/common/http';

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

Añadir al final, después
de BrowserModule

- Una vez añadido al módulo principal ya podremos utilizar esta librería desde cualquier componente, página o 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(  
    result => {  
        console.log(result);  
        this.datos = result;  
    }  
);
```

- El método “`subscribe`” permite realizar una petición asíncrona y asignar los datos cuando se obtengan.

- También podemos añadir la siguiente opción para obtener posibles errores en las peticiones:

```
this.http.get("http://...").subscribe(  
  result => {  
    console.log(result);  
  },  
  error => {  
    console.log(error);  
  }  
);
```

- Esto nos devolverá un objeto de error con el siguiente formato:
 - status: 404,
 - statusText: "Not Found",
 - url: "http:...",
 - ok: false,
 - name: "HttpErrorResponse",
 - message: "Http failure response for..."

¿PREGUNTAS?