



ANGULAR

Lección 4:

El sistema de

vistas

Coding Potions -4-

Angular - Vistas, sintaxis en los templates y web de notas de ejemplo



Jun 12, 2018

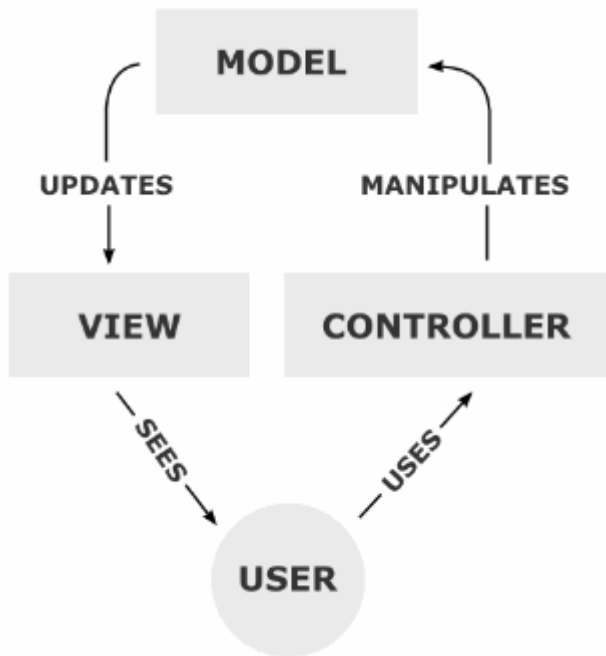
Introducción

Hasta ahora, si has seguido la serie de artículos sobre Angular, sabrás cómo crear componentes y cómo funciona (de forma básica), typescript, pero aún no sabemos cómo mostrar información en la pantalla del navegador, ni como pasar información de la vista al controlador.

En este sentido Angular es muy potente.

Con una sintaxis muy parecida a otros sistema de templates o arquitecturas MVC,

es capaz de ofrecer muchísimas funciones a los programadores. Por ejemplo podemos mostrar o no elementos de la página dependiendo de una variable boolean, o incluso podemos ejecutar directamente en la vista funciones declaradas en el componente.



Vistas

La manera más rápida de mostrar una variable contenida en un controlador es usar la sintaxis:

```
{{nombre_variable}}
```

Lo bueno de Angular es que puedes actualizar el valor de esta variable en cualquier momento y automáticamente los cambios se reflejarán en el HTML

Además, también se pueden poner operaciones o llamadas a funciones entre las llaves, por ejemplo:

<p>La suma de 5 y 4 es {{5 + 4}}</p>

O por ejemplo:

<p>getVal()</p>

Mostrando arrays

¿Qué pasa si tenemos un array de información?, para este caso Angular viene con unas etiquetas especiales para recorrer arrays o listas:


```
<li *ngFor="let item of nombre_array">
```

```
  {{ item }}
```

```
</li>
```

```
</ul>
```

Es importante poner el * antes del ngFor para que Angular lo detecte adecuadamente. Como ves, dentro del ngFor usamos sintaxis ES6, en concreto hacemos un for each. Por último mostramos la variable item que hemos creado en el bucle.

Cuando Angular sirva la página se encargara de crear un elemento `` por cada elemento del array, y dentro de cada uno, mostrará su contenido. Si en lugar de hacer el `ngFor` dentro un `` lo haces dentro de un `div`, por ejemplo, se creará un `div` por cada elemento del array.

¿Y si el array es un array de objetos, como se muestran los atributos de los objetos?

Simplemente al mostrar los elementos del array puedes poner, por ejemplo:

```
{{ item.atributo }}
```

Dentro de los bucles, también podemos declarar una variable para almacenar el valor del índice del elemento:

```
<ul>
```

```
  <li *ngFor="let item of nombre_array; let i =  
index">
```

```
    {{i}}. {{ item }}
```

```
</li>
```

```
</ul>
```

Mostrar elementos de forma condicional

En ocasiones, necesitamos que un elemento se muestre en pantalla

dependiendo de una condición. Esto se consigue por medio de ngIf:

```
<div *ngIf="array.length > 3">Esto se muestra si el  
array tiene menos de 3 elementos</div>
```

Si por ejemplo, tenemos una variable boolean (true, o false), también podemos mostrar contenido o no dependiendo del valor de esa variable:

```
<div *ngIf="condicion">Esto se muestra si la variable  
condicion es true (o distinto de null y  
undefined)</div>
```

De esta forma, por ejemplo, podemos mostrar en el navbar, un botón de login si el usuario no está registrado:

```
<nav class="navbar navbar-expand-lg navbar-light  
bg-light">
```

```
  <button class="navbar-toggler" type="button" data-  
toggle="collapse" data-  
target="#navbarTogglerDemo03" aria-  
controls="navbarTogglerDemo03" aria-  
expanded="false" aria-label="Toggle navigation">
```

```
  <span class="navbar-toggler-icon"></span>
```

```
</button>
```

```
<a class="navbar-brand" href="#">Navbar</a>
```

```
<div class="collapse navbar-collapse"
id="navbarTogglerDemo03">

  <ul class="navbar-nav mr-auto mt-2 mt-lg-0">

    <li class="nav-item active">

      <a class="nav-link" href="#">Home <span
class="sr-only">(current)</span></a>

    </li>

    <li class="nav-item">

      <a class="nav-link" href="#">Link</a>

    </li>

    <li class="nav-item">

      <a class="nav-link disabled"
href="#">Disabled</a>

    </li>

  </ul>
```

```
<div class="form-inline my-2 my-lg-0"
*ngIf="!user_is_logged">
    <button class="btn btn-outline-success my-2 my-
sm-0" type="submit">Login</button>
</div>
</div>
</nav>
```

Si te das cuenta, he puesto el ! antes de la variable de user_is_logged para negarlo, es decir, si el usuario no está registrado se mostrará el botón de login.

Dentro de los if también podemos llamar a funciones dentro de su controlador, por ejemplo:

```
<div *ngIf="getVal() < 3">Ejemplo</div>
```

Bindeando estilos y atributos html a variables en el controlador

Para bindear un atributo de una etiqueta html a una variable que tengamos en el controlador, por ejemplo el src de una imagen:

```
<img [src]="imagePath" />
```

Si queremos, por ejemplo, dinámicamente cambiar el color de fondo de una etiqueta html, pero poniendo el color que tenemos guardado en una variable, se hace así:

```
<div class="circle" [style.background]="color">
```

Como puedes observar, para los atributos de las etiquetas html se hace poniendo el atributo entre corchetes, y para los estilos también, solo que añadiendo delante style.

Eventos

Por ejemplo para controlar cuándo el usuario hace click en un botón:

```
<button (click)="onClick()">Click me!</button>
```

Después en el controlador, tenemos que crear una función con el mismo nombre, en este caso, `onClick`, que será la que se ejecute al pulsar el botón

Binding en inputs

Para recoger el valor del input mientras escribe el usuario, antes tenemos que asegurarnos que hemos importado el módulo de FormsModule de Angular, en el archivo app.mudule.ts, es decir:

```
import { FormsModule } from '@angular/forms';
```

A continuación lo importamos en la sección imports. Una vez hecho esto podemos continuar creando los inputs. Para bindear el input con la variable:

```
<input [(ngModel)]="name">
```

Al usar la sintaxis [(...)] lo que estamos haciendo es two way data binding, el paso de información se hace en dos sentidos: entre el html y el controlador y al revés. Esto quiere decir que si actualizamos el valor en el html o en el controlador en el otro sentido se actualizará automáticamente.

Para esto necesitamos tener una variable en el controlador llamada name (o con el nombre que le hayamos puesto en el html).

Si ahora muestras la variable name, debajo del input, te darás cuenta de que según escribes en el formulario se va mostrando abajo automáticamente:

```
{{name}}
```

Otro evento que podemos controlar, por ejemplo, es cuando el usuario levanta una tecla mientras escribe:

```
<input [(ngModel)]="name"  
(keyup)="onKeyUp($event)">
```

Ejemplo práctico



Para poner en práctica los conocimientos que hemos aprendido, vamos a realizar una app de prueba que consista en crear notas o recordatorios.

Para empezar creamos el proyecto con angular cli:

```
ng new notas --routing
```

Esta vez he puesto el parámetro `--routing` para que Angular se encargue de generar también el archivo de rutas. En mi caso he llamado a la aplicación `notas`.

A continuación lo que hago es `ng serve` para que la aplicación empiece a funcionar mientras desarrollo, por suerte, Angular se recompila cada vez que se guarda un

archivo por lo que cada cambio que realices se muestra en la página al instante.

Siguiente paso, generar los componentes que vayamos a necesitar.

En este caso, voy a crear únicamente un componente para mostrar la lista de notas, ya que solo van a tener un título. Si quieres que cada nota tenga una página independiente, tienes que crear otro componente para la vista detalle, con una

ruta dinámica, como vimos en este artículo, pero en este caso no lo voy a crear.

ng generate component notes

El siguiente paso es meter el componente que acabamos de crear en el archivo con las rutas, en este caso, como acabamos de crear la aplicación, viene vacío:

```
import { NgModule } from '@angular/core';
```

```
import { Routes, RouterModule } from
```

```
'@angular/router';
```

```
import { NotesComponent } from
```

```
'./notes/notes.component';
```



```
const routes: Routes = [  
  { path: "", component: NotesComponent,  
    pathMatch: 'full'},  
];
```

No hay que olvidar importar el componente de las notas. Para este tutorial he puesto que la ruta hacia las notas sea vacía, es decir, la página que aparece según abrimos la aplicación web.

Siguiente paso añadir el para que se muestren las rutas en la página, en este caso en `app.component.html`

```
<router-outlet></router-outlet>
```

Si ya viene por defecto al crear la web con angular cli, entonces no hace falta que lo añadas.

Como hemos dicho antes, tenemos que importar también el módulo de formularios (FormsModule) de Angular en el app.module.ts, dentro de la sección imports:

```
import { FormsModule } from '@angular/forms';
```

```
import { BrowserModule } from '@angular/platform-  
browser';
```

```
import { NgModule } from '@angular/core';
```

```
import { AppRoutingModule } from './app-  
routing.module';
```

```
import { AppComponent } from './app.component';
```

```
import { NotesComponent } from  
 './notes/notes.component';
```

```
@NgModule({
```

```
  declarations: [
```

```
    AppComponent,
```

```
NotesComponent

],

imports: [

  BrowserModule,

  AppRoutingModule,

  FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }
```

Creando y mostrando notas

Dentro del componente de notas, en el controlador (archivo .ts) voy a crear un

array de strings para almacenar las notas (lo inicializo en el constructor para evitar problemas) y otra variable para guardar el texto de la nota que va a ser creada, para ello:

```
export class NotesComponent implements OnInit {
```

```
  notes: string[];
```

```
  note: string;
```

```
  constructor() {
```

```
    this.notes = [];
```

```
  }
```

```
ngOnInit() {  
  
}  
  
}
```

Ahora, en el html, recorreremos y pintamos las notas y ponemos un input junto a un botón para mostrarlas:

```
<h1>Notes</h1>
```

```
<div class="notes">
```

```
<ul>
```

```
<li *ngFor="let note of notes">{{note}}</li>
```

```
</ul>
```

```
<input type="text" [(ngModel)]="note">  
<button (click)="createNote()">Create note</button>  
  
</div>
```

He bindeado con ngModel el input a la variable que he creado antes, y he creado un botón con un evento click para crear la nota.

Falta crear el método en el controlador para crear las notas y meterlas en el array:

```
createNote(){
```

```
this.notes.push(this.note);  
}
```

Si miras en la página ahora, y la pruebas, te darás cuenta de que se van creando las notas al pulsar el botón y que además no hace falta repintar o recargar el array porque Angular lo actualiza solo.

Si ahora, añades estilos css en el archivo css del componente, te puede quedar la página así:



Conclusiones

En unos minutos hemos hecho una app para crear notas (aunque no se guardan en ningún sitio y al actualizar se pierden), obviamente para este ejemplo de aplicación puede que no nos convenga utilizar Angular, ya que con Javascript puro se

puede hacer lo mismo con menos líneas de código y menos peso en los archivos, pero sirve de referencia para ver como funciona lo básico de Angular. En los siguientes artículos seguiremos profundizando en todo lo que Angular puede ofrecer.

Te dejo un par de artículos (en inglés) por si quieres profundizar un poco sobre este tema:

- [Display a Table using Components with Angular 4](#)




- Angular Template Syntax
-

¿Te ha gustado el artículo? ♥♥

Compártelo en las redes sociales para
apoyar a Coding Potions 🍷☐

¿Quieres aprender Angular gratis?

No te pierdas este curso gratuito totalmente online en el que aprenderás uno de los mejores frameworks del momento.

-  Aprende Angular desde 0 y gratis
-  Crea tus propias aplicaciones y páginas web dinámicas
-  Aprende los fundamentos de los componentes web

-  Conecta Angular un servidor
mediante una API REST
-  Depliega tus proyectos en
producción
-    Posts
-  Collections
-  Notes
-  About