ANGULAR

Juan García Martínez / Jose Alberto Ibáñez Marín / Pablo Sánchez Pérez

UNIVERSIDAD DE ALICANTE ADI

Angular

Introducción

¿Qué es angular?

Es un framework desarrollado en TypeScript por Google y de código abierto para crear aplicaciones web Single Page Application, tanto como aplicaciones móviles como de escritorio.

¿Qué características tiene?

- Separa el frontend y el backend
- SIgue el patrón Model-View-Controller
- Simplifica mucho el código
- Es de código abierto, como hemos comentado anteriormente
- Basado en componentes

Instalación y primer proyecto

Los requisitos de software para comenzar con Angular son los siguientes:

- Node.js, para poder ejecutar código javascript tanto del lado del servidor como del cliente.
- Npm, para administrar los paquetes necesarios de Node.js
- Angular CLI, para trabajar por línea de comandos.
- Editor de Código, en este ejemplo usaremos VS Code.

Para empezar, debemos entrar en el siguiente enlace de la página oficial de Angular que ofrece un tutorial para configurar el entorno local.

- https://angular.io/guide/setup-local

Lo primero que podemos observar es que nos recomienda antes de utilizar el framework es estar familiarizado con JavaScript, HTML y CSS. Además es recomendable conocer TypeScript pero no obligatorio.

Necesitamos antes de nada instalar Node.js, desde el siguiente enlace:

- https://nodejs.org/en/

Ahora debemos instalar Angular CLI, abriendo la consola de comandos o terminal, introducimos el siguiente comando:

- *npm install -g @angular/cli*

Una vez instalado, creamos una carpeta donde guardaremos los proyectos Angular y dentro de ella en una terminal ejecutamos el comando siguiente, donde ejemplo será el nombre de la carpeta del proyecto cuyo nombre se puede cambiar al gusto:

- *ng new ejemplo*

Con esto ya tenemos el primer proyecto creado, que es una plantilla vacía.

Lanzar el proyecto

Para lanzar el proyecto de prueba, nos situaremos en la carpeta *ejemplo* (en nuestro caso) desde la terminal, y ejecutaremos el siguiente comando:

- *ng serve -o*

Si todo sale bien obtendremos las siguientes líneas que nos indicarán el correcto funcionamiento y la dirección del servidor.

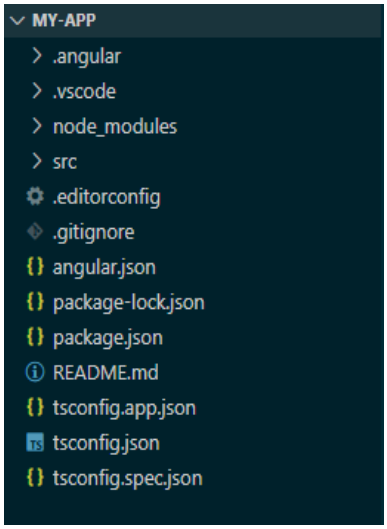
```
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/
**


✔ Compiled successfully.
```

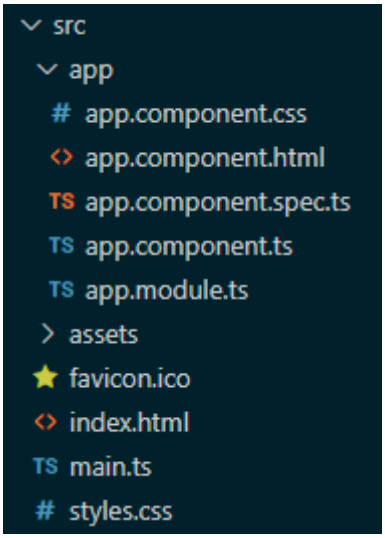
Conceptos de angular

Estructura y Flujo

La estructura básica de cualquier proyecto Angular es la siguiente:



El código para desarrollar la aplicación se encuentra en la carpeta *src* mientras que todo lo demás está relacionado con la configuración de la aplicación. La carpeta *node_modules* guarda las dependencias o paquetes que añadamos al proyecto.
Dentro de la carpeta *src* tendremos el *index* y dentro de la carpeta *app* tendremos los componentes de la aplicación y sus diferentes archivos.



Al iniciar la aplicación, se consultará en main cual será el módulo que se cargará. Este módulo es el módulo raíz, en *app.module.ts*, que define qué componente cargar. El componente principal será el *app.component.ts*. Este componente se carga en la plantilla con el mismo nombre, *app.component.html*.

Componentes

Un componente se divide en dos partes.La primera es la del Decorador, que contiene un selector, un template y un estilo. El selector nos permite insertar el componente mediante una etiqueta con el nombre de la variable, el template guarda el nombre del archivo de la plantilla y el estilo el archivo css del componente.

La segunda parte es la de la Clase, en el cual se pueden definir constructores, propiedades y métodos para usarlos dentro de la plantilla html.

También es posible declarar más componentes dentro de un componente.

Para crear un componente, lo recomendable es crear una carpeta nueva dentro de la carpeta app. Crearemos un fichero, que será donde definiremos el componente, con extensión TypeScript. Los componentes tienen como sintaxis [nombre_componente].component.[ts]. Para empezar definiremos la clase de la siguiente manera

```
import { component } from 'angular@core';
//Decorador
@Component ({
    selector: "ejemplo",
```

```
    templateUrl: "./ejemplo.component.html",
    styleUrls: ['./ejemplo.component.css']
})
//Clase
export class Ejemplo{

}
```

A la hora de insertar el componente en una plantilla html, solo tendremos que crear una etiqueta con el nombre que hayamos decidido en el selector del Decorador.

```
<ejemplo></ejemplo>
```

Cabe resaltar que si el programa es pequeño, se puede realizar el estilo y la plantilla de forma inline, insertando el código directamente entre estas dos comillas ``

```
template: `
     <p>
      ejemplo
     </p>
`
```

Interpolación

La interpolación es el mecanismo que tiene angular y otros frameworks para la sustitución de una expresión por un valor en una template. Cuando Angular ve en un template algo escrito entre dobles llaves {{}} lo evalúa y lo trata de convertir en una cadena, para luego volcarlo en el template.

Por ejemplo:

```
<p>Esto es un caso de interpolación de {{algunaCadena}}</p>
```

Eso quiere decir a Angular que debe sustituir una propiedad del componente llamada "algunaCadena" y colocar su valor tal cual en el template.

Si "algunaCadena" es una cadena, simplemente se escribirá su valor en el template, pero si eso no era una cadena tratará de colocarlo de manera que lo fuera. Por ejemplo, si tuviera un valor numérico colocará el número tal cual en el template.

Puedes usar también la interpolación como valor de propiedades de elementos HTML, como es el siguiente caso.

```
<img src="{{urlImagen}}">
```

En este caso se colocará el valor de la propiedad urlImagen como src para el elemento IMG.

Binding - Property,Event & Two Way

El enlace de datos mantiene la página actualizada automáticamente según el estado de la aplicación. Se utiliza el enlace de datos para especificar cosas como la fuente de una imagen, el estado de un botón o los datos de un usuario en particular.

Existen varios tipos de de enlaces:

- Property que le da estilo de clase de atributo con la propiedad de interpolación.

```
{{expression}}
[target]="expression"
```

Es unidireccional desde el origen de datos hasta el objetivo de visualización.

- Evento

```
(target)="statement"
```

Unidireccional desde el destino de la vista hasta la fuente de datos. Este tipo de dato se utiliza en botones.

- Bidireccional

```
[(target)]="expression"
```

Este enlace es bidireccional y la información va desde la vista hasta la fuente de datos o al revés, este enlace por ejemplo se puede usar en un input de tipo text o number para que antes de que el usuario meta los datos correspondiente muestre los datos de los componentes.

Directivas

De atributos:

- **ngModel**: Implementa el binding
- **ngClass**: permite añadir y eliminar varias clases
- **ngStyle**: permite asignar estilos inline

Estructurales:

- **ngIf**: Nos permite incluir condicionales de lógica en nuestro código,
- ***ngFor**: Permite ejecutar bucles

- **ngPlural**: es una directiva que permite agregar o remover elementos del DOM
- **ngTemplate**: esta directiva como su nombre lo indica es un template en Angular.
- **ngComponentOutlet**: nos permite crear componentes dinámicos.

Routing

En una single-page app, cambia lo que ve el usuario mostrando u ocultando partes de la pantalla que corresponden a componentes particulares, en lugar de ir al servidor para obtener una nueva página. A medida que los usuarios realizan tareas de aplicación, necesitan moverse entre las diferentes vistas que ha definido.

Para manejar la navegación de una vista habilitaremos Angular Router al crear un nuevo proyecto:

```
ng new routing-app --routing --defaults
```

Para usar el enrutador Angular, una aplicación debe tener al menos dos componentes para que pueda navegar de uno a otro. Para crear un componente usando la CLI, debemos ingresar la siguiente línea de comandos donde first es el nombre del componente:

```
ng generate component first
```

Repetimos este paso para un segundo componente pero asignándole un nombre diferente. Aquí, el nuevo nombre es second:

```
ng generate component second
```

La CLI se agrega automáticamente Component, por lo que si tuviera que escribir first-componen , su componente sería FirstComponentComponent.

Para usar los nuevos componentes, los importamos en AppRoutingModulela en la parte superior del archivo, de la siguiente manera:

```
import { FirstComponent } from './first/first.component';
import { SecondComponent } from './second/second.component';
```

Para crear una ruta básica hay tres bloques fundamentales:

La CLI de Angular realiza este paso por ti. Sin embargo, si estás creando una aplicación manualmente o trabajando con una aplicación existente que no es CLI, hay que verificar que las importaciones y la configuración sean correctas.

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

```
import { Routes, RouterModule } from '@angular/router'; // CLI imports
router

const routes: Routes = []; // sets up routes constant where you define
your routes

// configures NgModule imports and exports
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

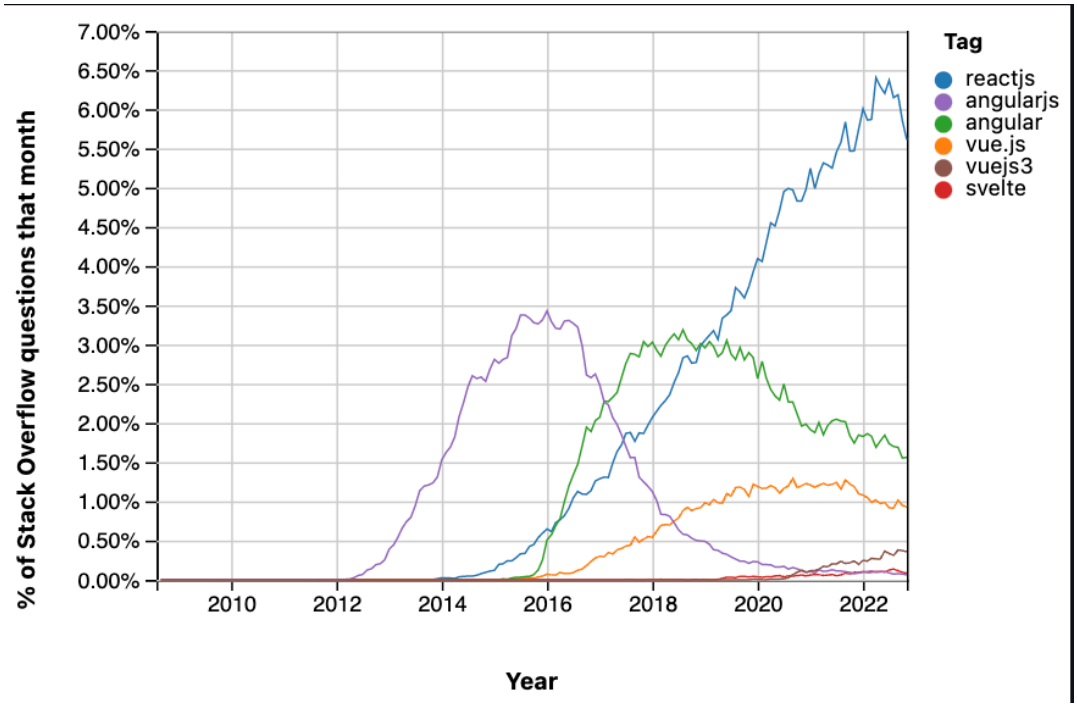
Definir las rutas:

```
const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
];
```

Para añadir las rutas primero, agregue enlaces a los dos componentes. Asigne la etiqueta de anclaje que desea agregar a la ruta al routerLink atributo. Establezca el valor del atributo en el componente para que se muestre cuando un usuario haga clic en cada enlace. A continuación, actualice la plantilla de su componente para incluir . Este elemento le informa a Angular que actualice la vista de la aplicación con el componente para la ruta seleccionada.<router-outlet>

```
<h1>Angular Router App</h1>
<!-- This nav gives you links to click, which tells the router which
route to use (defined in the routes constant in  AppRoutingModule) -->
<nav>
  <ul>
    <li><a routerLink="/first-component" routerLinkActive="active"
ariaCurrentWhenActive="page">First Component</a></li>
    <li><a routerLink="/second-component" routerLinkActive="active"
ariaCurrentWhenActive="page">Second Component</a></li>
  </ul>
</nav>
<!-- The routed views render in the <router-outlet>-->
<router-outlet></router-outlet>
```

Comparativa con otros frameworks



2022 (june 2022, +70,000 developers): https://survey.stackoverflow.co/2022/

- Popularity: React.js 42.62%, Angular 20.39%, Vue.js 18.82%, Angular.js 8.99%, Svelte 4.58%
- Loved: Svelte 75.28%, React.js 68.19%, Vue.js 63.16%, Angular 52.27%, Angular.js 21.01%
- Want: React.js 22.54%, Vue.js 14.6%, Angular 7.18%, Svelte 9.34%, Angular.js 4.32%

2021 (august 2021, +80,000 developers): https://insights.stackoverflow.com/survey/2021

- Popularity: React.js 40.14%, Angular 22.96%, Vue.js 18.97%, Angular.js 11.49%, Svelte 2.75%
- Loved: Svelte 71.47%, React.js 69.28%, Vue.js 64.41%, Angular 55.82%, Angular.js 23.18%
- Want: React.js 25.12%, Vue.js 16.69%, Angular 8.47%, Svelte 6.57%, Angular.js 5.8%

2020 (february 2020, 65,000 developers): https://insights.stackoverflow.com/survey/2020

- Popularity: React.js 35.9%, Angular 25.1%, Vue.js 17.3%, Angular.js 16.1%
- Loved: React.js 68.9%, Vue.js 66.0%, Angular 54.0%, Angular.js 24.1%
- Wanted: React.js 22.4%, Vue.js 16.4%, Angular 10.6%, Angular.js 7.7%

2019 (january 2019, +90,000 developers): https://insights.stackoverflow.com/survey/2019

- Popularity: React.js 31.3%, Angular/Angular.js 30.7%, Vue.js 15.2%
- Loved: React.js 74.5%, Vue.js 73.6%, Angular/Angular.js 57.6%
- Wanted: React.js 21.5%, Vue.js 16.1%, Angular/Angular.js 12.2%

2018 (january 2018, +100,000 developers): https://insights.stackoverflow.com/survey/2018

- Popularity: Angular 36.9%, React 27.8%
- Loved: React 69.4%, Angular 54.6%
- Wanted: React 21.3%, Angular 14.3%

Como podemos observar los lenguajes más utilizados son React y angular, en cambio, los que más desean son Svelte, React y Vue. El uso de un framework no es solo decisión de una persona sino de una empresa/organización, de la labor que se quiera implementar con estos frameworks y cual de ellos encaja mejor con los requisitos que se necesitan.

Bibliografía

https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190

- Templates and views
  https://angular.io/guide/architecture-components#templates-and-views

- Metadata
  https://angular.io/guide/architecture-components#component-metadata

- Data binding
  https://angular.io/guide/architecture-components#data-binding

- directives
  https://angular.io/guide/architecture-components#directives

- https://angular.io/guide/architecture

- Módulos
  https://angular.io/guide/architecture-modules
- Directivas
  https://medium.com/notasdeangular/directivas-en-angular-efb8a8cf78e0