



Introducción a Angular

¿Qué es una SPA?

A veces los nombres en el desarrollo de software no están bien elegidos, y eso puede llevar a mucha confusión. Ese no es el caso del término SPA: una aplicación de una sola página tiene literalmente una sola página.

Si quieres ver una aplicación de una sola página en acción, te invito a que te dirijas a la página web de

<https://angular-university.io/>

¿Qué es una SPA?

Si empiezas a navegar, verás que la página no se recarga completamente - sólo se envían nuevos datos mientras el usuario navega por la aplicación - ese es un ejemplo de una aplicación de una sola página.

¿Qué es una SPA?

Esta página está casi vacía, a excepción de la etiqueta, no hay mucho que hacer aquí.

En el sitio web real la página descargada también tiene HTML que fue pre-renderizado en el servidor usando Angular Universal.

¿Qué es una SPA?

Fíjate en los nombres de los paquetes de CSS y Javascript: Todo el CSS e incluso todo el Javascript de la aplicación (que incluye Angular) ni siquiera viene del mismo servidor que el index.html.

Además, observe que el nombre de los paquetes está versionado: contiene la marca de tiempo en la que se ejecutó la compilación de despliegue que desplegó esta versión concreta de la aplicación.

¿Qué es una SPA?

Una aplicación de una sola página es súper sencilla de desplegar si la comparamos con las aplicaciones más tradicionales renderizadas en el lado del servidor: en realidad es sólo un archivo `index.html`, con un paquete de CSS y un paquete de Javascript.

Estos 3 archivos estáticos se pueden subir a cualquier servidor de contenido estático como Apache, Nginx, Amazon S3 o Firebase Hosting.

¿Qué es una SPA?

Por supuesto, la aplicación tendrá que hacer llamadas al backend para obtener sus datos, pero eso es un servidor separado que puede ser construido si es necesario con una tecnología completamente diferente: como Node, Java PHP, etc.

Versionado

Versionado

Otra ventaja de desplegar nuestro frontend como una aplicación de una sola página es el versionado y el rollback.

Podemos configurar el servidor que está sirviendo nuestra SPA con un parámetro que especifica qué versión de la aplicación frontend construir.

UX

UX

Si alguna vez ha utilizado una aplicación web que está constantemente recargando todo desde el servidor en casi cada interacción con el usuario, sabrá que ese tipo de aplicación da una mala experiencia al usuario debido a:

1. Las constantes recargas de páginas completas.
2. también debido a los viajes de ida y vuelta de la red al servidor para obtener todo ese HTML.

UX

En una aplicación de página única, hemos resuelto este problema, utilizando un enfoque arquitectónico fundamentalmente diferente:

En una SPA, después de la carga inicial de la página, no se envía más HTML por la red. En su lugar, sólo se solicitan datos al servidor (o se envían al servidor).

UX

Por lo tanto, mientras se ejecuta una SPA, sólo se envían datos a través de la red, lo que requiere mucho menos tiempo y ancho de banda que el envío constante de HTML.

Con esto, tenemos una segunda gran ventaja de una aplicación de una sola página: una experiencia de usuario muy mejorada debido a menos recargas de páginas completas y un mejor rendimiento general porque se necesita menos ancho de banda.

Problemas SPA

Problemas SPA

Hasta hace relativamente poco, los motores de búsqueda como Google tenían dificultades para indexar correctamente una aplicación de una sola página. ¿Pero qué pasa hoy en día?

En el pasado, había algunas recomendaciones para utilizar un esquema especial de rastreo de Ajax que, entretanto, ha quedado obsoleto. Entonces, ¿es Google capaz ahora de renderizar completamente Ajax?

Angular

Angular

Angular es una plataforma de desarrollo, construida sobre TypeScript. Como plataforma, Angular incluye:

1. Un marco de trabajo basado en componentes para construir aplicaciones web escalables.
2. Una colección de bibliotecas bien integradas que cubren una amplia variedad de características, incluyendo enrutamiento, gestión de formularios, comunicación cliente-servidor, y más.
3. Un conjunto de herramientas para desarrolladores que le ayudan a desarrollar, construir, probar y actualizar su código.

Angular CLI

Angular CLI

La CLI de Angular es la forma más rápida, sencilla y recomendada de desarrollar aplicaciones de Angular. Aquí hay algunos ejemplos:

1. **ng build:** Compila una aplicación Angular en un directorio de salida.
2. **ng serve:** Construye y sirve tu aplicación, reconstruyendo en los cambios de archivos.
3. **ng generate:** Genera o modifica archivos basados en un esquema.
4. **ng test:** Ejecuta pruebas unitarias en un proyecto determinado.
5. **ng e2e:** Construye y sirve una aplicación Angular, y luego ejecuta pruebas de extremo a extremo.

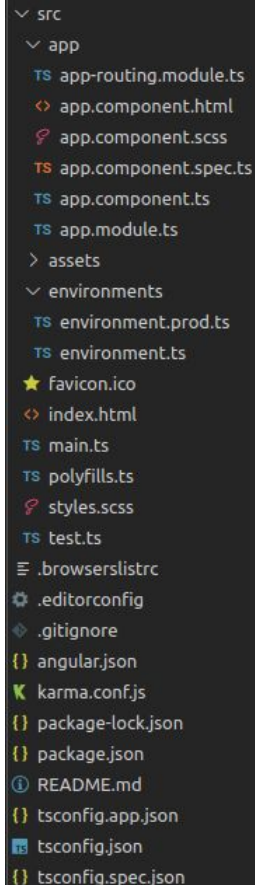
Estructura Básica



Estructura básica

Angular se basa en módulos.

1. Cada módulo puede contener una funcionalidad concreta dentro de nuestra aplicación.
2. Cada módulo puede contener componentes, servicios, directivas, etc.
3. Toda aplicación contiene un módulo raíz que realiza la función de cargar todo lo necesario para nuestra aplicación



```
src
├── app
│   ├── app-routing.module.ts
│   ├── app.component.html
│   ├── app.component.scss
│   ├── app.component.spec.ts
│   ├── app.component.ts
│   ├── app.module.ts
│   └── assets
├── environments
│   ├── environment.prod.ts
│   └── environment.ts
├── favicon.ico
├── index.html
├── main.ts
├── polyfills.ts
├── styles.scss
├── test.ts
├── .browserslistrc
├── .editorconfig
├── .gitignore
├── angular.json
├── karma.conf.js
├── package-lock.json
├── package.json
├── README.md
├── tsconfig.app.json
├── tsconfig.json
└── tsconfig.spec.json
```

Estructura básica: Módulos

Cada app de Angular contiene al menos un NgModule, el módulo raíz. Si bien una aplicación pequeña puede tener solo un NgModule, la mayoría de apps tienen muchos más módulos.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [];

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

Estructura básica: Módulos

Metadatos de un módulo:

1. **Declarations:** componentes, directivas y pipes.
2. **Exports:** para exportar componentes específicos de este módulo.
3. **Imports:** otros módulos o librerías.
4. **Providers:** servicios.
5. **Bootstrap:** Solo el módulo raíz deberá almacenar esta propiedad. Enlaza el componente principal.

Componentes



Componente

Los componentes son los bloques de construcción que componen una aplicación. Un componente incluye una clase TypeScript con un decorador `@Component()`, una plantilla HTML y estilos. El decorador `@Component()` especifica la siguiente información específica de Angular:

1. Un selector CSS que define cómo se utiliza el componente en una plantilla. Los elementos HTML de la plantilla que coinciden con este selector se convierten en instancias del componente.
2. Una plantilla HTML que instruye a Angular sobre cómo renderizar el componente.
3. Un conjunto opcional de estilos CSS que definen la apariencia de los elementos HTML de la plantilla.

Componente

```
app.component.ts x
1  import { Component } from
   '@angular/core';
2
3  @Component({
4    selector: 'my-app',
5    templateUrl: './app.component.html',
6    styleUrls: [ './app.component.css' ]
7  })
8  export class AppComponent {
9    name = 'Angular';
10 }
11
```

Creación de componente

Para crear un componente ejecutaremos en la consola el siguiente comando:

```
ng generate component nombre_componente
```

Ciclo de vida de un componente

Una instancia de componente tiene un ciclo de vida que comienza cuando Angular instancia la clase del componente y renderiza la vista del componente junto con sus vistas hijas.

El ciclo de vida continúa con la detección de cambios, ya que Angular comprueba si las propiedades vinculadas a los datos cambian, y actualiza tanto la vista como la instancia del componente según sea necesario.

Ciclo de vida de un componente

El ciclo de vida termina cuando Angular destruye la instancia del componente y elimina su plantilla renderizada del DOM.

Su aplicación puede utilizar métodos del ciclo de vida de un componente o directiva para inicializar nuevas instancias, iniciar la detección de cambios cuando sea necesario, responder a las actualizaciones durante la detección de cambios y limpiar antes de la eliminación de instancias.

Ciclo de vida de un componente

Ordenados:

1. Constructor
2. OnChanges
3. OnInit
4. DoCheck
5. AfterContentInit
6. AfterContentChecked
7. AfterViewInit
8. AfterViewChecked
9. OnDestroy

Templates

Templates

En Angular, una plantilla es un trozo de HTML. Utiliza una sintaxis especial dentro de una plantilla para aprovechar muchas de las características de Angular.

Cada plantilla de Angular en su aplicación es una sección de HTML para incluir como parte de la página que el navegador muestra. Una plantilla HTML de Angular renderiza una vista, o interfaz de usuario, en el navegador, igual que el HTML normal, pero con mucha más funcionalidad.

Interpolación de textos

La interpolación de texto le permite incorporar valores de cadena dinámicos en sus plantillas HTML.

La interpolación para cambiar dinámicamente lo que aparece en una vista de la aplicación, como por ejemplo mostrar un saludo personalizado que incluya el nombre del usuario.

Pipes

Utilice las tuberías para transformar cadenas, cantidades de moneda, fechas y otros datos para su visualización. Las tuberías son funciones sencillas que se utilizan en las expresiones de plantilla para aceptar un valor de entrada y devolver un valor transformado. Las tuberías son útiles porque puedes utilizarlas en toda tu aplicación, declarando cada tubería sólo una vez. Por ejemplo, puede utilizar una tubería para mostrar una fecha como 15 de abril de 1988 en lugar de su formato de cadena sin procesar.

Síntaxis de un Template

Un Template tendrá lo siguiente:

Etiquetas HTML: `<h2>`, `<p>`, `<div>`, etc.

Elementos de Angular:

- `{{ usuario.nombre }}`, `(click)` y `[usuario]` enlazan los datos a mostrar en el DOM.
- `<app-usuario-detalles>`: selector de otro componente
- `*ngFor`: interactúa con una lista.
- `*ngIf`: muestra/oculta la etiqueta.

Binding

Binding

Sin un framework, seríamos responsables de enlazar la información en el HTML con el componente, lo que supondría un trabajo pesado. Esta tarea en Angular se resuelve con el concepto de databinding.

1. Desde el componente al DOM con `{{ }}`
2. Desde el DOM al componente con `()`
3. Two-way data-binding `[()]`

Property binding

Desde el componente al DOM Se mostrará el valor del atributo o método en el DOM.

```
<p>My color is {{ fontColor }}</p>
```

Property binding

Lanza eventos del DOM y ejecuta un método cuando se lance dicho evento.

```
<button  
  [disabled]="canClick"  
  (click)="sayMessage()">  
  Trigger alert message  
</button>
```

Property binding

En el Double data-binding se utiliza la Propiedad [(ngModel)], generalmente es usado en formularios.

```
<input type="text" class="form-control" id="name"
  required
  [(ngModel)]="data.name" />

<p>Hello {{ data.name }}!</p>
```


Routing

Routing

El enrutamiento le permite mostrar vistas específicas de su aplicación dependiendo de la ruta de la URL.

Para añadir esta funcionalidad a tu aplicación necesitas actualizar el archivo `app.module.ts` para utilizar el módulo, `RouterModule`. Este módulo se importa desde `@angular/router`.

Routing

Una definición de ruta es un objeto JavaScript. Cada ruta suele tener dos propiedades. La primera propiedad, `path`, es una cadena que especifica la ruta URL de la ruta. La segunda propiedad, `component`, es una cadena que especifica qué componente debe mostrar la aplicación para esa ruta.

Routing

```
import { RouterModule } from '@angular/router';

@NgModule({
  declarations: [
    AppComponent,
    HelloWorldComponent,
    HelloWorldTemplateComponent,
    HelloWorldNgIfComponent,
    HelloWorldDependencyInjectionComponent,
    HelloWorldInterpolationComponent,
    HelloWorldBindingsComponent,
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot([
      { path: 'bindings', component: HelloWorldBindingsComponent },
      { path: 'ngIf', component: HelloWorldNgIfComponent },
    ]),
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Routing

En este punto, has definido dos rutas para tu aplicación. Sin embargo, su aplicación todavía tiene los componentes codificados en su plantilla `app.component.html`.

Para que sus rutas funcionen, necesita actualizar su plantilla para cargar dinámicamente un componente basado en la ruta de la URL.

Para implementar esta funcionalidad, debes añadir la directiva `router-outlet` a tu archivo de plantilla.

Routing

```
import { RouterModule } from '@angular/router';

@NgModule({
  declarations: [
    AppComponent,
    HelloWorldComponent,
    HelloWorldTemplateComponent,
    HelloWorldNgIfComponent,
    HelloWorldDependencyInjectionComponent,
    HelloWorldInterpolationComponent,
    HelloWorldBindingsComponent,
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot([
      { path: 'bindings', component: HelloWorldBindingsComponent },
      { path: 'ngIf', component: HelloWorldNgIfComponent },
    ]),
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Forms

Forms

El manejo de entrada de datos de los usuarios con formularios es la piedra angular de muchas aplicaciones comunes. Las aplicaciones utilizan formularios para permitir a los usuarios iniciar sesión, actualizar un perfil, introducir información sensible y realizar muchas otras tareas de entrada de datos.

Angular proporciona dos enfoques diferentes para manejar la entrada del usuario a través de los formularios: **reactivo** y **basado en plantillas**.

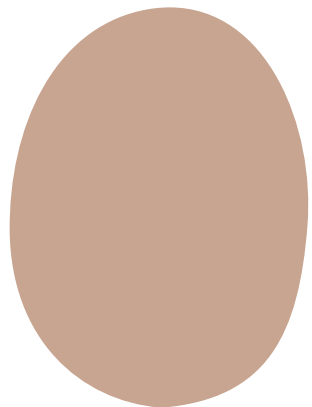
Formularios reactivos

Los formularios reactivos y los formularios basados en plantillas procesan y gestionan los datos del formulario de forma diferente. Cada enfoque ofrece diferentes ventajas.

Los **formularios reactivos** proporcionan un acceso directo y explícito al modelo de objetos de formularios subyacente. En comparación con los formularios basados en plantillas, son más robustos: son más escalables, reutilizables y comprobables. Si los formularios son una parte clave de su aplicación, o si ya está utilizando patrones reactivos para construir su aplicación, utilice formularios reactivos.

Formularios basados en plantillas

Los **formularios basados en plantillas** se basan en las directivas de la plantilla para crear y manipular el modelo de objetos subyacente. Son útiles para añadir un formulario simple a una aplicación, como un formulario de inscripción a una lista de correo electrónico. Son fáciles de añadir a una aplicación, pero no escalan tan bien como los formularios reactivos. Si tienes unos requisitos de formulario muy básicos y una lógica que puede gestionarse únicamente en la plantilla, los formularios impulsados por plantillas podrían ser una buena opción.



FIN

