EBook Gratis

APRENDIZAJE Angular

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angular

It is an unofficial and free Angular ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Angular.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Angular

Observaciones

Angular (comúnmente denominado " **Angular 2+** " o " **Angular 2** ") es un marco web de front-end de código abierto basado en TypeScript liderado por el Angular Team en Google y por una comunidad de individuos y corporaciones para abordar todos los Partes del flujo de trabajo del desarrollador al crear aplicaciones web complejas. Angular es una reescritura completa del mismo equipo que creó AngularJS . [1]

El marco consta de varias bibliotecas , algunas de ellas básicas (@ angular / core por ejemplo) y otras opcionales (@ angular / animations).

Las aplicaciones de Angular se escriben mediante la composición de *plantillas* HTML con marcado Angularized, la escritura de clases de *componentes* para administrar esas plantillas, la adición de la lógica de la aplicación en los *servicios* y los componentes de boxeo y los servicios en los *módulos* .

A continuación, inicie la aplicación mediante el *arranque* del *módulo raíz* . Angular asume el control, presenta el contenido de su aplicación en un navegador y responde a las interacciones de los usuarios de acuerdo con las instrucciones que ha proporcionado.

Podría decirse que la parte más fundamental del desarrollo de aplicaciones angulares son los **componentes** . Un componente es la combinación de una plantilla HTML y una clase de componente que controla una parte de la pantalla. Aquí hay un ejemplo de un componente que muestra una cadena simple:

src / app / app.component.ts

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

@Component({
    selector: 'my-app',
    template: `<h1>Hello {{name}}</h1>`
})
export class AppComponent {
    name = 'Angular';
}
```

Cada componente comienza con una función decoradora `@Component` que toma un objeto de metadatos . El objeto de metadatos describe cómo la plantilla HTML y la clase de componente trabajan juntas.

La propiedad del `selector` le dice a Angular que muestre el componente dentro de una etiqueta personalizada `<my-app>` en el archivo *index.html* .

index.html (dentro de la etiqueta del `body`)

```
<my-app>Loading AppComponent content here ...</my-app>
```

La propiedad de plantilla define un mensaje dentro de un encabezado `<h1>` . El mensaje comienza con "Hola" y termina con `{{name}}` , que es una expresión de enlace de interpolación angular. En tiempo de ejecución, Angular reemplaza `{{name}}` con el valor de la propiedad de `name` del componente. El enlace de interpolación es una de las muchas características angulares que descubrirá en esta documentación. En el ejemplo, cambie la propiedad de `name` la clase de componente de `'Angular'` a `'World'` y vea qué sucede.

Este ejemplo está escrito en **TypeScript** , un superconjunto de JavaScript. Angular utiliza TypeScript porque sus tipos facilitan el soporte de la productividad del desarrollador con herramientas. Además, **casi todo el soporte es para TypeScript** y, por lo tanto, será **difícil** usar **JavaScript plano** para escribir su aplicación. Escribir código angular en JavaScript es posible, sin embargo; Esta guía explica cómo.

Más información sobre la **arquitectura** de Angular se puede encontrar **aquí.**

Versiones

Versión	Fecha de lanzamiento
5.0.0-beta.1 (más reciente)	2017-07-27
4.3.2	2017-07-26
5.0.0-beta.0	2017-07-19
4.3.1	2017-07-19
4.3.0	2017-07-14
4.2.6	2017-07-08
4.2.5	2017-06-09
4.2.4	2017-06-21
4.2.3	2017-06-16
4.2.2	2017-06-12
4.2.1	2017-06-09
4.2.0	2017-06-08
4.2.0-rc.2	2017-06-01
4.2.0-rc.1	2017-05-26

Versión	Fecha de lanzamiento
4.2.0-rc.0	2017-05-19
4.1.3	2017-05-17
4.1.2	2017-05-10
4.1.1	2017-05-04
4.1.0	2017-04-26
4.1.0-rc.0	2017-04-21
4.0.3	2017-04-21
4.0.2	2017-04-11
4.0.1	2017-03-29
4.0.0	2017-03-23
4.0.0-rc.6	2017-03-23
4.0.0-rc.5	2017-03-17
4.0.0-rc.4	2017-03-17
2.4.10	2017-03-17
4.0.0-rc.3	2017-03-10
2.4.9	2017-03-02
4.0.0-rc.2	2017-03-02
4.0.0-rc.1	2017-02-24
2.4.8	2017-02-18
2.4.7	2017-02-09
2.4.6	2017-02-03
2.4.5	2017-01-25
2.4.4	2017-01-19
2.4.3	2017-01-11
2.4.2	2017-01-06

Versión	Fecha de lanzamiento
2.4.1	2016-12-21
2.4.0	2016-12-20
2.3.1	2016-12-15
2.3.0	2016-12-07
2.3.0-rc.0	2016-11-30
2.2.4	2016-11-30
2.2.3	2016-11-23
2.2.2	2016-11-22
2.2.1	2016-11-17
2.2.0	2016-11-14
2.2.0-rc.0	2016-11-02
2.1.2	2016-10-27
2.1.1	2016-10-20
2.1.0	2016-10-12
2.1.0-rc.0	2016-10-05
2.0.2	2016-10-05
2.0.1	2016-09-23
2.0.0	2016-09-14
2.0.0-rc.7	2016-09-13
2.0.0-rc.6	2016-08-31
2.0.0-rc.5	2016-08-09
2.0.0-rc.4	2016-06-30
2.0.0-rc.3	2016-06-21
2.0.0-rc.2	2016-06-15
2.0.0-rc.1	2016-05-03

Versión	Fecha de lanzamiento
2.0.0-rc.0	2016-05-02

Examples

Instalación de Angular utilizando angulo-cli.

Este ejemplo es una configuración rápida de Angular y cómo generar un proyecto de ejemplo rápido.

Requisitos previos:

- Node.js 6.9.0 o mayor.
- npm v3 o mayor o hilo .
- Typings v1 o mayor.

Abra un terminal y ejecute los comandos uno por uno:

`npm install -g typings` o `yarn global add typings`

`npm install -g @angular/cli` o `yarn global add @angular/cli`

El primer comando instala la biblioteca de mecanografía globalmente (y agrega las `typings` ejecutables a PATH). El segundo instala @ **angular / cli** globalmente, agregando el ejecutable `ng` a PATH.

Para configurar un nuevo proyecto

Navegue con el terminal a una carpeta donde desee configurar el nuevo proyecto.

Ejecutar los comandos:

```
ng new PROJECT_NAME
cd PROJECT_NAME
ng serve
```

Eso es todo, ahora tienes un proyecto de ejemplo simple hecho con Angular. Ahora puede navegar hasta el enlace que se muestra en la terminal y ver qué se está ejecutando.

Para agregar a un proyecto existente

Navegue a la raíz de su proyecto actual.

Ejecuta el comando:

```
ng init
```

Esto agregará los andamios necesarios para su proyecto. Los archivos se crearán en el directorio actual, así que asegúrese de ejecutar esto en un directorio vacío.

Ejecutando el proyecto localmente

Para ver e interactuar con su aplicación mientras se ejecuta en el navegador, debe iniciar un servidor de desarrollo local que aloje los archivos para su proyecto.

```
ng serve
```

Si el servidor se inició correctamente, debe mostrar una dirección en la que se ejecuta el servidor. Por lo general es esto:

```
http://localhost:4200
```

Fuera de la caja, este servidor de desarrollo local está conectado con la recarga de módulos calientes, por lo que cualquier cambio en el html, mecanografiado o css, hará que el navegador se vuelva a cargar automáticamente (pero se puede desactivar si se desea).

Generación de componentes, directivas, tuberías y servicios.

El comando `ng generate <scaffold-type> <name>` (o simplemente `ng g <scaffold-type> <name>`) le permite generar automáticamente componentes Angulares:

```
# The command below will generate a component in the folder you are currently at
ng generate component my-generated-component
# Using the alias (same outcome as above)
ng g component my-generated-component
# You can add --flat if you don't want to create new folder for a component
ng g component my-generated-component --flat
# You can add --spec false if you don't want a test file to be generated (my-generated-
component.spec.ts)
ng g component my-generated-component --spec false
```

Hay varios tipos posibles de andamios que angular-cli puede generar:

Tipo de andamio	Uso
Módulo	`ng g module my-new-module`
Componente	`ng g component my-new-component`
Directiva	`ng g directive my-new-directive`

Tipo de andamio	Uso
Tubo	`ng g pipe my-new-pipe`
Servicio	`ng g service my-new-service`
Clase	`ng g class my-new-class`
Interfaz	`ng g interface my-new-interface`
Enumerar	`ng g enum my-new-enum`

También puede reemplazar el nombre del tipo por su primera letra. Por ejemplo:

`ng gm my-new-module` para generar un nuevo módulo o `ng gc my-new-component` para crear un componente.

Construcción / Bundling

Cuando haya terminado de construir su aplicación web Angular y desee instalarla en un servidor web como Apache Tomcat, todo lo que necesita hacer es ejecutar el comando de compilación con o sin el conjunto de indicadores de producción. La producción minimiza el código y se optimiza para un entorno de producción.

```
ng build
```

o

```
ng build --prod
```

Luego busque en la carpeta raíz del proyecto una carpeta `/dist` , que contiene la compilación.

Si desea obtener los beneficios de un paquete de producción más pequeño, también puede usar la compilación de la plantilla Anticipada, que elimina el compilador de la plantilla de la compilación final:

```
ng build --prod --aot
```

Examen de la unidad

Angular proporciona pruebas unitarias integradas, y cada elemento creado por angular-cli genera una prueba unitaria básica, que puede gastarse. Las pruebas unitarias se escriben con jazmín y se ejecutan a través de Karma. Para iniciar la prueba ejecuta el siguiente comando:

```
ng test
```

Este comando ejecutará todas las pruebas en el proyecto y las volverá a ejecutar cada vez que cambie un archivo fuente, ya sea una prueba o un código de la aplicación.

Para más información también visite: página angular-cli github

Programa Angular "Hola Mundo"

Requisitos previos:

Configuración del entorno de desarrollo

Antes de comenzar, tenemos que configurar nuestro entorno.

- Instale Node.js y npm si aún no están en su máquina.

  Verifique que esté ejecutando al menos el nodo 6.9.xy npm 3.xx ejecutando los nodos -v y npm -v en una ventana de terminal / consola. Las versiones más antiguas producen errores, pero las versiones más nuevas están bien.

- Instale Angular CLI globalmente usando `npm install -g @angular/cli` .

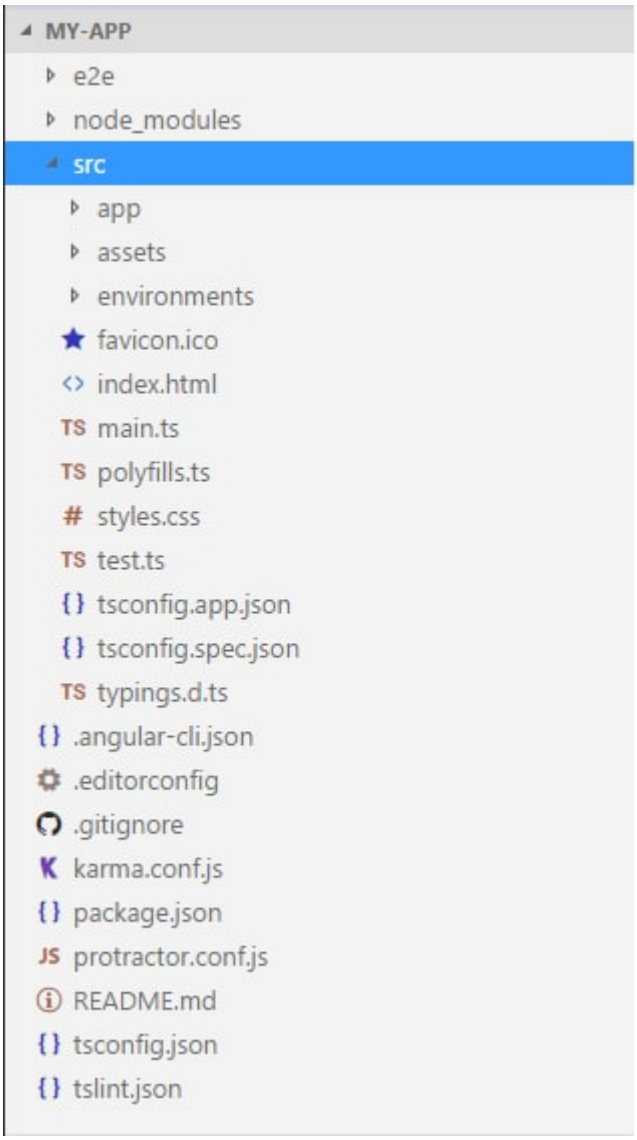
Paso 1: Creando un nuevo proyecto

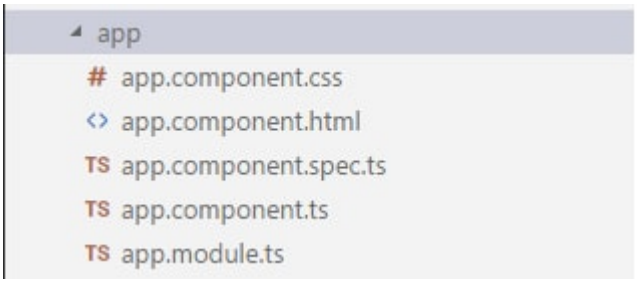
Abra una ventana de terminal (o el símbolo del sistema Node.js en Windows).

Creamos un nuevo proyecto y una aplicación de esqueleto usando el comando:

```
ng new my-app
```

Aquí la `ng` es para angular. Obtenemos una estructura de archivos algo como esto.





Hay muchos archivos. No tenemos que preocuparnos por todos ellos ahora.

Paso 2: Servir la aplicación

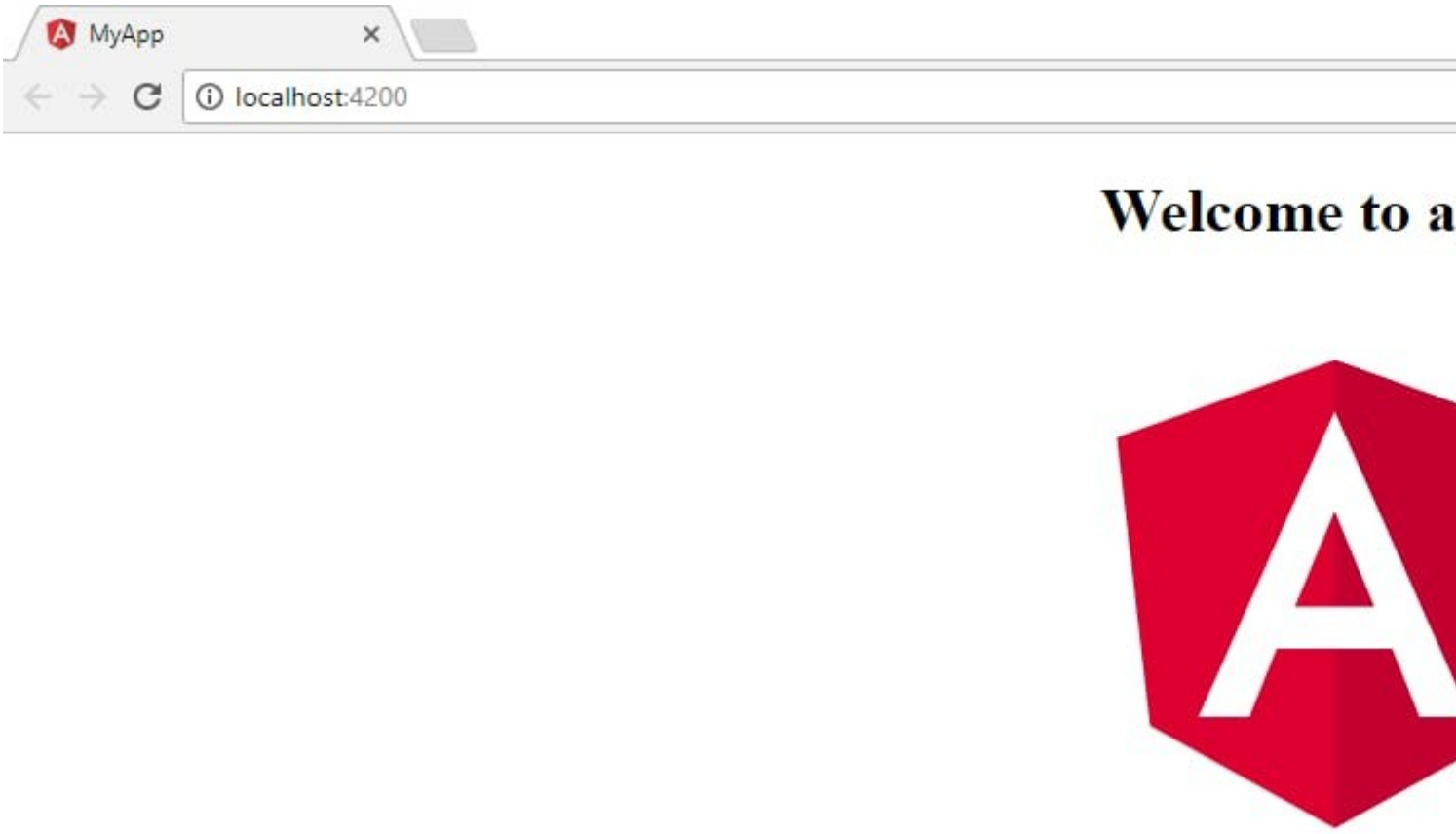
Lanzamos nuestra aplicación usando el siguiente comando:

```
ng serve
```

Podemos usar una bandera `-open` (o simplemente `-o`) que abrirá automáticamente nuestro navegador en `http://localhost:4200/`

```
ng serve --open
```

Navegue el navegador a la dirección `http://localhost:4200/` . Se ve algo como esto:



Paso 3: Editando nuestro primer componente angular.

El CLI creó el componente Angular predeterminado para nosotros. Este es el componente raíz y se llama `app-root` . Uno puede encontrarlo en `./src/app/app.component.ts` .

Abra el archivo del componente y cambie la propiedad del título de ¡ `Welcome to app!!` a `Hello World` . El navegador se vuelve a cargar automáticamente con el título revisado.

Código original: Observe el `title = 'app';`

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

Angular CLI, 20 minutes ago | 1 author (Angular CLI)
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```

Código modificado: Se modifica el valor del `title` .

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Hello World';
}
```

Del mismo modo, hay un cambio en `./src/app/app.component.html` .

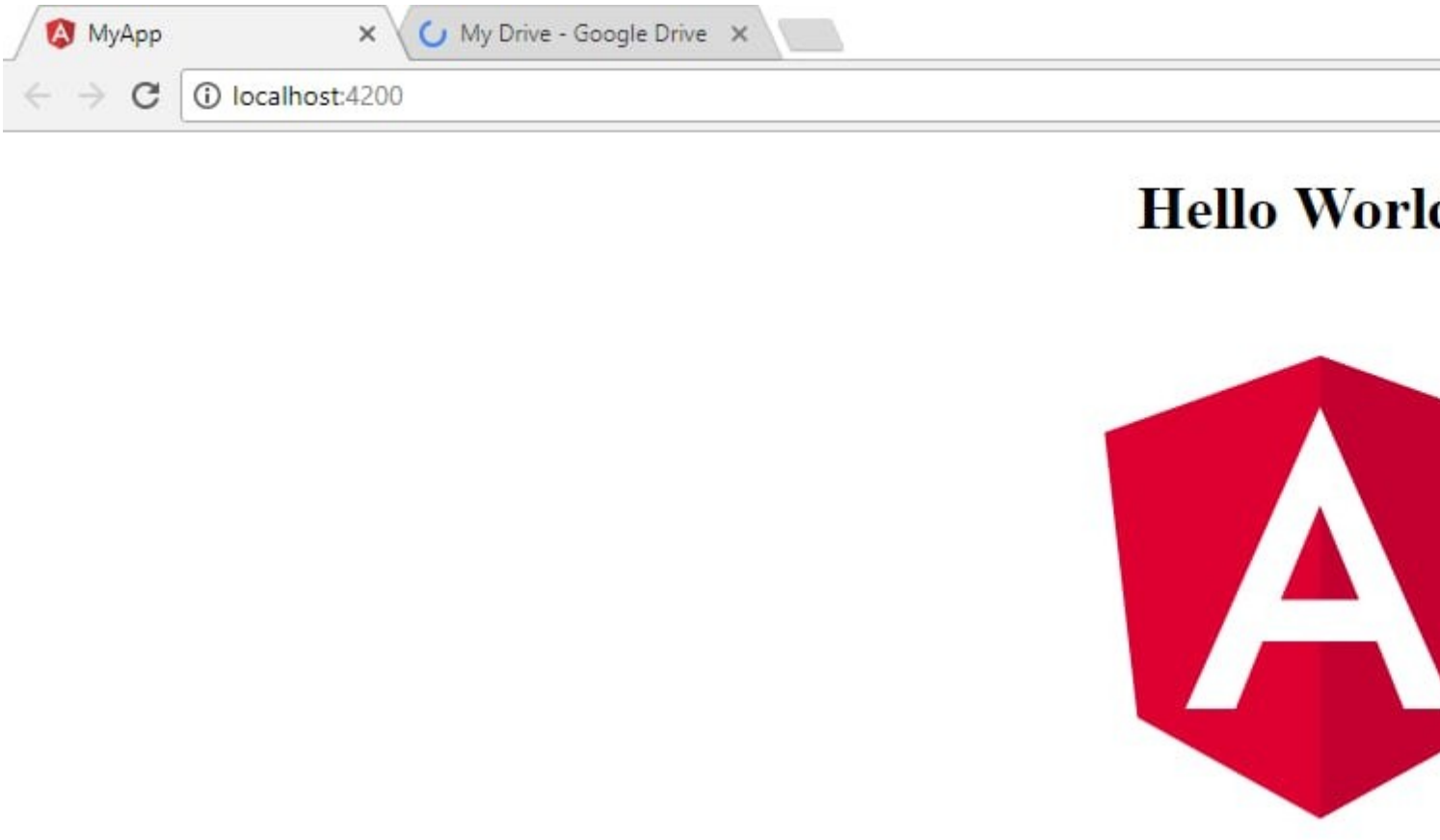
HTML original

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!!
  </h1>
  <img width="300" src="
```

HTML modificado

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
     {{title}}!!
  </h1>
  <img width="300" src="
```

Observe que se mostrará el valor del `title` de `./src/app/app.component.ts` . El navegador se vuelve a cargar automáticamente cuando se realizan los cambios. Se ve algo como esto.



Para encontrar más sobre el tema, visite este enlace aquí .

Lea Empezando con Angular en línea: https://riptutorial.com/es/angular/topic/9754/empezando-con-angular

Capítulo 2: Compartir datos entre componentes

Introducción

El objetivo de este tema es crear ejemplos simples de varias formas en que los datos se pueden compartir entre componentes a través del enlace de datos y el servicio compartido.

Observaciones

Siempre hay muchas formas de realizar una tarea en la programación. Por favor, siéntase libre de editar ejemplos actuales o agregar algunos de los suyos.

Examples

Envío de datos desde el componente principal al hijo a través del servicio compartido

servicio.ts:

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

@Injectable()
export class AppState {

  public mylist = [];

}
```

parent.ponent.ts:

```
import {Component} from '@angular/core';
import { AppState } from './shared.service';

@Component({
  selector: 'parent-example',
  templateUrl: 'parent.component.html',
})
export class ParentComponent {
  mylistFromParent = [];

  constructor(private appState: AppState){
    this.appState.mylist;
  }

  add() {
    this.appState.mylist.push({"itemName":"Something"});
  }
```

```
}
```

parent.component.html:

```
<p> Parent </p>
  <button (click)="add()">Add</button>
<div>
  <child-component></child-component>
</div>
```

child.ponent.ts:

```
import {Component, Input } from '@angular/core';
import { AppState } from './shared.service';

@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by shared service</h3>
        {{mylist | json}}
  `,
})
export class ChildComponent {
  mylist: any;

  constructor(private appState: AppState){
    this.mylist = this.appState.mylist;
  }

}
```

Envíe datos desde el componente principal al componente secundario a través del enlace de datos utilizando @Input

parent.ponent.ts:

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

@Component({
  selector: 'parent-example',
  templateUrl: 'parent.component.html',
})

export class ParentComponent {
  mylistFromParent = [];

  add() {
    this.mylistFromParent.push({"itemName":"Something"});
  }

}
```

parent.component.html:

```
<p> Parent </p>
  <button (click)="add()">Add</button>

<div>
  <child-component [mylistFromParent]="mylistFromParent"></child-component>
</div>
```

child.ponent.ts:

```
import {Component, Input } from '@angular/core';

@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by parent</h3>
        {{mylistFromParent | json}}
  `,
})

export class ChildComponent {
  @Input() mylistFromParent = [];
}
```

Envío de datos de niños a padres a través del emisor de eventos @Output

event-emitter.component.ts

```
import { Component, OnInit, EventEmitter, Output } from '@angular/core';

@Component({
  selector: 'event-emitting-child-component',
  template: `<div *ngFor="let item of data">
              <div (click)="select(item)">
                {{item.id}} = {{ item.name}}
              </div>
             </div>
            `
})

export class EventEmitterChildComponent implements OnInit{

  data;

  @Output()
  selected: EventEmitter<string> = new EventEmitter<string>();

  ngOnInit(){
    this.data = [ { "id": 1, "name": "Guy Fawkes", "rate": 25 },
                  { "id": 2, "name": "Jeremy Corbyn", "rate": 20 },
                  { "id": 3, "name": "Jamie James", "rate": 12 },
                  { "id": 4, "name": "Phillip Wilson", "rate": 13 },
                  { "id": 5, "name": "Andrew Wilson", "rate": 30 },
                  { "id": 6, "name": "Adrian Bowles", "rate": 21 },
                  { "id": 7, "name": "Martha Paul", "rate": 19 },
                  { "id": 8, "name": "Lydia James", "rate": 14 },
                  { "id": 9, "name": "Amy Pond", "rate": 22 },
                  { "id": 10, "name": "Anthony Wade", "rate": 22 } ]
  }
```

```
  select(item) {
      this.selected.emit(item);

  }

}
```

evento-receiver.component.ts:

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

@Component({
    selector: 'event-receiver-parent-component',
    template: `<event-emitting-child-component (selected)="itemSelected($event)">
                </event-emitting-child-component>
                <p *ngIf="val">Value selected</p>
                <p style="background: skyblue">{{ val | json}}</p>`
})

export class EventReceiverParentComponent{
  val;

  itemSelected(e){
    this.val = e;
  }
}
```

Envío de datos asíncronos de padres a hijos utilizando Observable y Subject

shared.service.ts:

```
import { Injectable }    from '@angular/core';
import { Headers, Http } from '@angular/http';

import 'rxjs/add/operator/toPromise';

import { Observable } from 'rxjs/Observable';
import { Observable } from 'rxjs/Rx';
import {Subject} from 'rxjs/Subject';


@Injectable()
export class AppState {

  private headers = new Headers({'Content-Type': 'application/json'});
  private apiUrl = 'api/data';

  // Observable string source
  private dataStringSource = new Subject<string>();

  // Observable string stream
  dataString$ = this.dataStringSource.asObservable();

  constructor(private http: Http) { }

  public setData(value) {
    this.dataStringSource.next(value);
```

```
  }

  fetchFilterFields() {
    console.log(this.apiUrl);
    return this.http.get(this.apiUrl)
               .delay(2000)
               .toPromise()
               .then(response => response.json().data)
               .catch(this.handleError);
  }

  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error); // for demo purposes only
    return Promise.reject(error.message || error);
  }

}
```

parent.ponent.ts:

```
import {Component, OnInit} from '@angular/core';
import 'rxjs/add/operator/toPromise';
import { AppState } from './shared.service';

@Component({
  selector: 'parent-component',
  template: `
              <h2> Parent </h2>
              <h4>{{promiseMarker}}</h4>

              <div>
                <child-component></child-component>
              </div>
            `
})
export class ParentComponent implements OnInit {

  promiseMarker = "";

  constructor(private appState: AppState){ }

  ngOnInit(){
    this.getData();
  }

  getData(): void {
    this.appState
        .fetchFilterFields()
        .then(data => {
          // console.log(data)
          this.appState.setData(data);
          this.promiseMarker = "Promise has sent Data!";
        });
  }

}
```

child.ponent.ts:

```
import {Component, Input } from '@angular/core';
import { AppState } from './shared.service';


@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by shared service</h3>
        {{fields | json}}
  `,
})
export class ChildComponent {
  fields: any;

  constructor(private appState: AppState){
    // this.mylist = this.appState.get('mylist');

    this.appState.dataString$.subscribe(
      data => {
        // console.log("Subs to child" + data);
        this.fields = data;
      });

  }

}
```

Lea Compartir datos entre componentes en línea:
https://riptutorial.com/es/angular/topic/10836/compartir-datos-entre-componentes

Capítulo 3: Emisor de eventos

Examples

Atrapando el evento

Crear un servicio-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
    navchange: EventEmitter<number> = new EventEmitter();
    constructor() {}
    emitNavChangeEvent(number) {
        this.navchange.emit(number);
    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}
```

Crear un componente para usar el servicio.

```
import {Component} from 'angular2/core';
import {NavService} from '../services/NavService';

@Component({
    selector: 'obs-comp',
    template: `obs component, item: {{item}}`
    })
    export class ObservingComponent {
    item: number = 0;
    subscription: any;
    constructor(private navService:NavService) {}
    ngOnInit() {
        this.subscription = this.navService.getNavChangeEmitter()
        .subscribe(item => this.selectedNavItem(item));
    }
    selectedNavItem(item: number) {
        this.item = item;
    }
    ngOnDestroy() {
        this.subscription.unsubscribe();
    }
}

@Component({
    selector: 'my-nav',
    template:`
        <div class="nav-item" (click)="selectedNavItem(1)">nav 1 (click me)</div>
        <div class="nav-item" (click)="selectedNavItem(2)">nav 2 (click me)</div>
    `,
})
export class Navigation {
    item = 1;
    constructor(private navService:NavService) {}
```

```
    selectedNavItem(item: number) {
        console.log('selected nav item ' + item);
        this.navService.emitNavChangeEvent(item);
    }
}
```

Lea Emisor de eventos en línea: https://riptutorial.com/es/angular/topic/9828/emisor-de-eventos

Capítulo 4: En bucle

Examples

NgFor - Markup For Loop

La directiva **NgFor crea** una instancia de una plantilla por elemento desde un iterable. El contexto para cada plantilla instanciada se hereda del contexto externo con la variable de bucle dada establecida en el elemento actual del iterable.

Para personalizar el algoritmo de seguimiento predeterminado, NgFor admite la opción **trackBy** . **trackBy** toma una función que tiene dos argumentos: índice y elemento. Si se proporciona **trackBy** , las pistas angulares cambian según el valor de retorno de la función.

```
<li *ngFor="let item of items; let i = index; trackBy: trackByFn">
    {{i}} - {{item.name}}
</li>
```

Opciones adicionales : NgFor proporciona varios valores exportados que pueden ser asignados a las variables locales:

- **El índice** se establecerá en la iteración del bucle actual para cada contexto de plantilla.
- **primero** se establecerá en un valor booleano que indica si el elemento es el primero en la iteración.
- **el último** se establecerá en un valor booleano que indica si el elemento es el último en la iteración.
- **incluso** se establecerá en un valor booleano que indica si este elemento tiene un índice par.
- **odd** se establecerá en un valor booleano que indica si este elemento tiene un índice impar.

Lea En bucle en línea: https://riptutorial.com/es/angular/topic/9826/en-bucle

Capítulo 5: Enrutamiento

Examples

Enrutamiento con niños.

Descubrí que esta es la forma de anidar correctamente las rutas de los niños dentro del archivo app.routing.ts o app.module.ts (según sus preferencias). Este enfoque funciona cuando se usa WebPack o SystemJS.

El siguiente ejemplo muestra las rutas de inicio, inicio / contador y inicio / contador / obtener datos. Las primeras y últimas rutas son ejemplos de redirecciones. Finalmente, al final del ejemplo es una forma adecuada de exportar la Ruta a importar en un archivo separado. Por ej. app.module.ts

Para explicar con más detalle, Angular requiere que tenga una ruta sin ruta en la matriz secundaria que incluye el componente principal, para representar la ruta principal. Es un poco confuso, pero si piensa en una URL en blanco para una ruta secundaria, esencialmente sería igual a la misma URL que la ruta principal.

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

import { HomeComponent } from "./components/home/home.component";
import { FetchDataComponent } from "./components/fetchdata/fetchdata.component";
import { CounterComponent } from "./components/counter/counter.component";

const appRoutes: Routes = [
    {
        path: "",
        redirectTo: "home",
        pathMatch: "full"
    },
    {
        path: "home",
        children: [
            {
                path: "",
                component: HomeComponent
            },
            {
                path: "counter",
                children: [
                    {
                        path: "",
                        component: CounterComponent
                    },
                    {
                        path: "fetch-data",
                        component: FetchDataComponent
                    }
                ]
            }
```

```
        ]
    },
    {
        path: "**",
        redirectTo: "home"
    }
];

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

Gran ejemplo y descripción a través de Siraj

Enrutamiento básico

El enrutador permite la navegación de una vista a otra según las interacciones del usuario con la aplicación.

Los siguientes son los pasos para implementar el enrutamiento básico en Angular:

NOTA : Asegúrate de tener esta etiqueta:

```
<base href='/'>
```

como el primer hijo debajo de su etiqueta de cabecera en su archivo index.html. Este elemento indica que su carpeta de aplicaciones es la raíz de la aplicación. Angular entonces sabría cómo organizar tus enlaces.

1. Compruebe si está apuntando a las dependencias de enrutamiento correctas / más recientes en package.json (usando la última versión de Angular) y si ya realizó una `npm install` :

   ```
   "dependencies": {
       "@angular/router": "^4.2.5"
   }
   ```

2. Defina la ruta según su definición de interfaz:

   ```
   interface Route {
     path?: string;
     pathMatch?: string;
     component?: Type<any>;
   }
   ```

3. En un archivo de enrutamiento (`routes/app.routing.ts`), importe todos los componentes que

necesita configurar para diferentes rutas de enrutamiento. La ruta vacía significa que la vista se carga de forma predeterminada. ":" en la ruta indica un parámetro dinámico pasado al componente cargado.

```
import { Routes, RouterModule } from '@angular/router';
import { ModuleWithProviders } from '@angular/core';
import { BarDetailComponent } from '../components/bar-detail.component';
import { DashboardComponent } from '../components/dashboard.component';
import { LoginComponent } from '../components/login.component';
import { SignupComponent } from '../components/signup.component';

export const APP_ROUTES: Routes = [
    { path: '', pathMatch: 'full', redirectTo: 'login' },
    { path: 'dashboard', component: DashboardComponent },
    { path: 'bars/:id', component: BarDetailComponent },
    { path: 'login', component: LoginComponent },
    { path: 'signup',   component: SignupComponent }
];
export const APP_ROUTING: ModuleWithProviders = RouterModule.forRoot(APP_ROUTES);
```

4. En su `app.module.ts` , coloque esto bajo `@NgModule([])` bajo `imports` :

```
// Alternatively, just import 'APP_ROUTES
import {APP_ROUTING} from '../routes/app.routing.ts';
@NgModule([
    imports: [
        APP_ROUTING
        // Or RouterModule.forRoot(APP_ROUTES)
    ]
])
```

5. Cargue / visualice los componentes del enrutador según la ruta a la que haya accedido. La directiva `<router-outlet>` se utiliza para indicar a angular dónde cargar el componente.

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

@Component({
    selector: 'demo-app',
    template: `
        <div>
            <router-outlet></router-outlet>
        </div>
    `
})
export class AppComponent {}
```

6. Enlace las otras rutas. De forma predeterminada, `RouterOutlet` cargará el componente para el cual se especifica una ruta vacía en las `Routes` . `RouterLink` directiva `RouterLink` se usa con la etiqueta de anclaje html para cargar los componentes adjuntos a las rutas. `RouterLink` genera el atributo href que se utiliza para generar enlaces. Por ejemplo:

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

@Component({
```

```
    selector: 'demo-app',
    template: `
        <a [routerLink]="['/login']">Login</a>
        <a [routerLink]="['/signup']">Signup</a>
        <a [routerLink]="['/dashboard']">Dashboard</a>
        <div>
            <router-outlet></router-outlet>
        </div>
        `
})
export class AnotherComponent { }
```

Ahora, estamos bien con el enrutamiento a rutas estáticas. `RouterLink` admite la ruta dinámica al pasar parámetros adicionales junto con la ruta.

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

@Component({
  selector: 'demo-app',
  template: `
        <ul>
          <li *ngFor="let bar of bars | async">
            <a [routerLink]="['/bars', bar.id]">
              {{bar.name}}
            </a>
          </li>
        </ul>
    <div>
      <router-outlet></router-outlet>
    </div>
  `
})
export class SecondComponent { }
```

`RouterLink` toma una matriz donde el primer parámetro es la ruta de enrutamiento y los elementos subsiguientes son para los parámetros de enrutamiento dinámico.

Lea Enrutamiento en línea: https://riptutorial.com/es/angular/topic/9827/enrutamiento

Capítulo 6: Formas

Examples

Formas reactivas

app.module.ts

Agregue estos en su archivo app.module.ts para usar formularios reactivos

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from './app.component';

@NgModule({
    imports: [
        BrowserModule,
        FormsModule,
        ReactiveFormsModule,
    ],
    declarations: [ AppComponent ]
    providers: [],
    bootstrap: [ AppComponent ]
})
export class AppModule {}
```

app.component.ts

```
import { Component,OnInit } from '@angular/core';
import template from './app.component.html';
import { FormGroup,FormBuilder,Validators } from '@angular/forms';
import { matchingPasswords } from './validators';

@Component({
    selector: 'app',
    template
})
export class AppComponent implements OnInit {
    addForm: FormGroup;

    constructor(private formBuilder: FormBuilder) {
    }

    ngOnInit() {
        this.addForm = this.formBuilder.group({
            username: ['', Validators.required],
            email: ['', Validators.required],
            role: ['', Validators.required],
            password: ['', Validators.required],
            password2: ['', Validators.required]
        }, { validator: matchingPasswords('password', 'password2') });
```

```
    };

    addUser() {
        if (this.addForm.valid) {
            var adduser = {
                username: this.addForm.controls['username'].value,
                email: this.addForm.controls['email'].value,
                password: this.addForm.controls['password'].value,
                profile: {
                    role: this.addForm.controls['role'].value,
                    name: this.addForm.controls['username'].value,
                    email: this.addForm.controls['email'].value
                }
            };

            console.log(adduser);// adduser var contains all our form values. store it where
you want
            this.addForm.reset();// this will reset our form values to null
        }
    }

}
```

app.component.html

```
<div>
    <form [formGroup]="addForm">
        <input
            type="text"
            placeholder="Enter username"
            formControlName="username" />

        <input
            type="text"
            placeholder="Enter Email Address"
            formControlName="email"/>

        <input
            type="password"
            placeholder="Enter Password"
            formControlName="password" />

        <input
            type="password"
            placeholder="Confirm Password"
            name="password2"
            formControlName="password2" />

        <div class='error' *ngIf="addForm.controls.password2.touched">
            <div
                class="alert-danger errormessageadduser"
                *ngIf="addForm.hasError('mismatchedPasswords')">
                    Passwords do not match
            </div>
        </div>
        <select name="Role" formControlName="role">
            <option value="admin" >Admin</option>
            <option value="Accounts">Accounts</option>
            <option value="guest">Guest</option>
```

```
        </select>
        <br/>
        <br/>
        <button type="submit" (click)="addUser()">
            <span>
                <i class="fa fa-user-plus" aria-hidden="true"></i>
            </span>
            Add User
        </button>
    </form>
</div>
```

validadores.ts

```
export function matchingPasswords(passwordKey: string, confirmPasswordKey: string) {
    return (group: ControlGroup): {
        [key: string]: any
    } => {
        let password = group.controls[passwordKey];
        let confirmPassword = group.controls[confirmPasswordKey];

        if (password.value !== confirmPassword.value) {
            return {
                mismatchedPasswords: true
            };
        }
    }
}
```

Formularios dirigidos por plantillas

Plantilla - `signup.component.html`

```
<form #signUpForm="ngForm" (ngSubmit)="onSubmit()">

  <div class="title">
    Sign Up
  </div>

  <div class="input-field">
    <label for="username">username</label>
    <input
      type="text"
      pattern="\w{4,20}"
      name="username"
      required="required"
      [(ngModel)]="signUpRequest.username" />
  </div>

  <div class="input-field">
    <label for="email">email</label>
    <input
      type="email"
      pattern="^\S+@\S+$"
      name="email"
      required="required"
```

```
      [(ngModel)]="signUpRequest.email" />
  </div>

  <div class="input-field">
    <label for="password">password</label>
    <input
      type="password"
      pattern=".{6,30}"
      required="required"
      name="password"
      [(ngModel)]="signUpRequest.password" />
  </div>

  <div class="status">
    {{ status }}
  </div>

  <button [disabled]="!signUpForm.form.valid" type="submit">
    <span>Sign Up</span>
  </button>

</form>
```

Componente - `signup.component.ts`

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

import { SignUpRequest } from './signup-request.model';

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.css']
})
export class SignupComponent {

  status: string;
  signUpRequest: SignUpRequest;

  constructor() {
    this.signUpRequest = new SignUpRequest();
  }

  onSubmit(value, valid) {
    this.status = `User ${this.signUpRequest.username} has successfully signed up`;
  }

}
```

Modelo - `signup-request.model.ts`

```
export class SignUpRequest {

  constructor(
    public username: string="",
    public email: string="",
```

```
    public password: string=""
  ) {}

}
```

Módulo de aplicaciones - `app.module.ts`

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { SignupComponent } from './signup/signup.component';

@NgModule({
  declarations: [
    AppComponent,
    SignupComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Componente de la aplicación - `app.component.html`

```
<app-signup></app-signup>
```

Lea Formas en línea: https://riptutorial.com/es/angular/topic/9825/formas

Capítulo 7: RXJS y Observables

Examples

Espera múltiples solicitudes

Un escenario común es esperar a que finalicen varias solicitudes antes de continuar. Esto se puede lograr utilizando el método `forkJoin` .

En el siguiente ejemplo, `forkJoin` se usa para llamar a dos métodos que devuelven `Observables` . La devolución de llamada especificada en el método `.subscribe` se llamará cuando ambos Observables se hayan completado. Los parámetros proporcionados por `.subscribe` coinciden con el orden dado en la llamada a `.forkJoin` . En este caso, primero `posts` luego `tags` .

```
loadData() : void {
    Observable.forkJoin(
        this.blogApi.getPosts(),
        this.blogApi.getTags()
    ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
        this.posts = posts;
        this.tags = tags;
    }));
}
```

Solicitud basica

El siguiente ejemplo muestra una solicitud HTTP GET simple. `http.get()` devuelve un `Observable` que tiene el método `subscribe` . Este anexa los datos devueltos a la matriz de `posts` .

```
var posts = []

getPosts(http: Http): {
    this.http.get(`https://jsonplaceholder.typicode.com/posts`)
        .subscribe(response => {
            posts.push(response.json());
        });
}
```

Lea RXJS y Observables en línea: https://riptutorial.com/es/angular/topic/9829/rxjs-y-observables

Capítulo 8: Tubería

Introducción

Las tuberías son muy similares a los filtros en AngularJS, ya que ambas ayudan a transformar los datos en un formato específico. El carácter de la tubería `|` Se utiliza para aplicar tuberías en angular.

Examples

Tubos personalizados

my.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'myPipe'})
export class MyPipe implements PipeTransform {

  transform(value:any, args?: any):string {
    let transformedValue = value; // implement your transformation logic here
    return transformedValue;
  }

}
```

my.component.ts

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

@Component({
  selector: 'my-component',
  template: `{{ value | myPipe }}`
})
export class MyComponent {

    public value:any;

}
```

my.module.ts

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

import { MyComponent } from './my.component';
import { MyPipe } from './my.pipe';

@NgModule({
  imports: [
    BrowserModule,
```

```
  ],
  declarations: [
    MyComponent,
    MyPipe
  ],
})
export class MyModule { }
```

Tubos personalizados múltiples

Tener tubos diferentes es un caso muy común, donde cada tubo hace una cosa diferente. Agregar cada tubería a cada componente puede convertirse en un código repetitivo.

Es posible agrupar todas las tuberías utilizadas con frecuencia en un `Module` e importar que el nuevo módulo en cualquier componente necesita las tuberías.

breaklines.ts

```
import { Pipe } from '@angular/core';
/**
 * pipe to convert the \r\n into <br />
 */
@Pipe({ name: 'br' })
export class BreakLine {
    transform(value: string): string {
        return value == undefined ? value :
             value.replace(new RegExp('\r\n', 'g'), '<br />')
              .replace(new RegExp('\n', 'g'), '<br />');
    }
}
```

mayúsculas.ts

```
import { Pipe } from '@angular/core';
/**
 * pipe to uppercase a string
 */
@Pipe({ name: 'upper' })
export class Uppercase{
    transform(value: string): string {
        return value == undefined ? value : value.toUpperCase( );
    }
}
```

pipe.module.ts

```
import { NgModule } from '@angular/core';
import { BreakLine } from './breakLine';
import { Uppercase} from './uppercase';

@NgModule({
    declarations: [
        BreakLine,
        Uppercase
    ],
```

```
    imports: [

    ],
    exports: [
        BreakLine,
        Uppercase
    ]
    ,
})
export class PipesModule {}
```

my.component.ts

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

@Component({
  selector: 'my-component',
  template: `{{ value | upper | br}}`
})
export class MyComponent {

    public value: string;

}
```

my.module.ts

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

import { MyComponent } from './my.component';
import { PipesModule} from './pipes.module';

@NgModule({
  imports: [
    BrowserModule,
    PipesModule,
  ],
  declarations: [
    MyComponent,
  ],
})
```

Lea Tubería en línea: https://riptutorial.com/es/angular/topic/9824/tuberia

Creditos

S. No	Capítulos	Contributors
1	Empezando con Angular	aholtry, Anup Kumar Gupta, BogdanC, Community, daddycool, Edric, Fahad Nisar, Faisal, Hendrik Brummermann, Philipp Kief, Tom
2	Compartir datos entre componentes	0mpurdy, BogdanC, JGFMK, Nehal
3	Emisor de eventos	aholtry
4	En bucle	aholtry
5	Enrutamiento	0mpurdy, aholtry, Edric
6	Formas	aholtry, Saka7
7	RXJS y Observables	aholtry
8	Tubería	aholtry, Amr ElAdawy