

Angular

Tutorial:

**Learn Angular from
scratch step by step**

Contenido

Learn Angular from scratch step by step 1

Learn Angular with free step by step angular tutorials 5

Angular para principiantes: AngularJS vs Angular 2 vs Angular 7 6

¿Qué hay de nuevo en Angular en comparación con AngularJS? 9

Requisitos y configuración para comenzar a aprender Angular 21

**Configurar el entorno de desarrollo
angular 21**

**Ahora, comencemos a construir el
proyecto de ejemplo de aplicación
Angular..... 28**

**Construir un proyecto de ejemplo de
CRUD angular paso a paso 36**

**Arquitectura angular de aplicaciones
..... 45**

**Estructura de proyecto de aplicación
de ejemplo angular 79**

**Aplicación angular de navegación y
enrutamiento 96**

Construyendo una aplicación de ejemplo	
Angular paso a paso	102
Navegación de la aplicación usando el módulo de enrutador angular.....	102
Componentes de diseño de material angular para mejorar UI / UX	110
Agregar un backend a nuestro proyecto de ejemplo angular	117
Siguiente: ejemplo angular avanzado	131

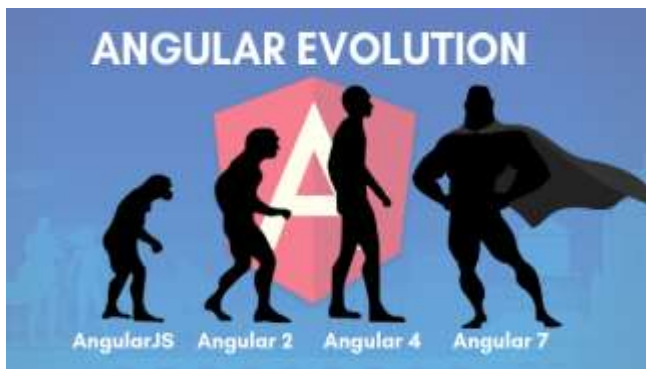
Learn Angular with free step by step angular tutorials

**All you need to learn about
Angular, the best tips and free
code examples so you can get
the most out of Angular.**

Angular para principiantes:

AngularJS vs Angular 2 vs Angular 7

Diferencias entre versiones angulares



Cuando todo comenzó, en 2010, este marco se llamaba AngularJS y alude a lo que ahora conocemos como Angular 1.x. Luego, en 2016, Angular 2 llegó como una reescritura completa del marco, mejorando de las lecciones aprendidas y prometedoras mejoras de rendimiento, y un marco más escalable y más moderno.

AngularJS estaba completamente basado en controladores y la vista se comunica usando \$scope mientras que Angular 2 es 100% un enfoque basado en componentes. En Angular 2, ya no tenemos los controladores y \$scope . Los componentes son los componentes básicos de una aplicación Angular 2.

Veremos los beneficios de este cambio en unos minutos.

La primera versión de Angular se llamó Angular 2. Más tarde, se renombró a "Angular". Entre Angular 2 y Angular 7 (la última versión estable actual) hubo Angular 4 (lanzado a principios de 2017), Angular 5 (lanzado a fines de 2017) y Angular 6 (lanzado a principios de 2018).

No te asustes con todas estas versiones. Debido a que todas las versiones de Angular 2 a Angular 7 son el mismo marco, comparten el mismo núcleo pero difieren en muchas mejoras sorprendentes.

De ahora en adelante, cada vez que usamos el término Angular nos referimos a la última versión del framework que hoy es Angular 7.

Para obtener más información sobre las últimas versiones de Angular, consulte:
<https://github.com/angular/angular/blob/master/CHANGELOG.md>

¿Qué hay de nuevo en Angular en comparación con AngularJS?

Veamos las principales diferencias entre AngularJS y Angular:

- **Angular es una reescritura completa de AngularJS**
- **Una aplicación angular y su arquitectura son diferentes de AngularJS. Los principales elementos de construcción de Angular son módulos, componentes, plantillas, metadatos, enlace de datos, directivas, servicios e inyección de dependencias.**
- **Angular no tiene un concepto o controladores de "alcance", sino que utiliza una jerarquía de componentes como su arquitectura principal**

- **Angular sigue un concepto de modularidad. Funcionalidades similares se mantienen juntas dentro de los módulos. Esto le da a Angular un núcleo más ligero optimizado**
- **El concepto de controlador, que estaba presente en AngularJS, se eliminó de Angular 2 y superior, que son IU basadas en componentes. Esto ayuda a los desarrolladores a dividir las aplicaciones en componentes con las características deseadas. Esto ayudó a mejorar la flexibilidad y la**

reutilización en comparación con AngularJS

- La sintaxis de expresión angular se centra en "[]" para el enlace de propiedad y "()" para el enlace de evento**
- Con AngularJS, crear una aplicación de página única amigable para motores de búsqueda (SEO) fue una gran dificultad. Pero este cuello de botella se eliminó con Angular 2 al habilitar el procesamiento de aplicaciones en el servidor. Estas tareas son posibles gracias al módulo Angular Universal.**

Angular también recomienda usar el lenguaje TypeScript, que presenta estas características:

- **Mecanografía Estática**
- **Programación Orientada a Objetos basada en clases**
- **Apoye la programación reactiva usando RxJS**

Además de las características de TypeScript, Angular también incluye los beneficios tomados de ES6:

- **Para / de bucles**
- **Inyección de dependencia mejorada**

- **Iteradores**
- **Reflexión**
- **Carga dinámica**
- **Compilación asincrónica de plantillas**
- **Enrutamiento más simple**

De angular 2 a angular 4

Hubo algunos cambios importantes, pero principalmente en la estructura del proyecto con muchos refactorices que hicieron que el marco fuera más estable.

- **Más pequeño y más rápido. La actualización de 2.0 a 4.0 ha reducido el tamaño del archivo**

incluido en un 60% al tiempo que mejora la velocidad de las aplicaciones.

- Angular 4 es compatible con las versiones más nuevas de TypeScript 2.1 y TypeScript 2.2.**
- Angular Universal: la gran mayoría del código Angular Universal se ha fusionado en el núcleo angular.**
- Paquete de animación: animaciones tomadas del núcleo angular y configuradas dentro de su propio paquete. Lo que significa que si no usa animaciones, el exceso de código no terminará en su aplicación.**

De angular 4 a angular 7

Angular 7 está lleno de nuevas características, correcciones de errores, mejoras de rendimiento y algunos desaprobaciones de código como una limpieza de los refactores de versiones anteriores.

- Optimizaciones para el proceso de compilación que reduce el tamaño de la aplicación al eliminar el código innecesario.**
- Componentes de diseño de materiales con renderizado del lado del servidor.**

- **Mejoras angulares universales para la asignación de código entre el servidor y las versiones del lado del cliente de la aplicación.**
- **Muchas mejoras en la CLI angular**
- **Tamaños de paquete más pequeños**
- **Compilador mejorado que admite compilación incremental, lo que significa reconstrucciones más rápidas.**
- **RxJS (biblioteca de programación reactiva) se ha actualizado a la versión 6.xo posterior.**
- **Angular ahora requiere TypeScript 3.x**

Avanzando en este tutorial angular, configuremos el entorno de desarrollo. Después de la introducción anterior sobre el estado actual de Angular Framework, ahora estamos listos para comenzar a trabajar en nuestra aplicación angular 7. La mejor manera de aprender Angular es siguiendo este tutorial paso a paso para principiantes.

En la siguiente sección de este curso gratuito angular, veremos la configuración y los requisitos necesarios para comenzar a desarrollar aplicaciones angulares.

Si está en el proceso de comenzar su proyecto Angular y necesita ayuda, eche un vistazo a nuestras Plantillas Angulares que lo ayudarán con Bootstrap 4, Angular Universal (Representación del lado del servidor), SEO, Carga diferida y mucho más.

[Angular Site Template](#) es nuestra última plantilla de Angular 7 e incluye toneladas de casos de uso implementados de forma angular, como flujos de autenticación, listado de productos, filtrado, formularios, guardias de enrutamiento y más.

Categories



New C

HOME / ALL Products

COLOR



SIZE

7	7.5	8
8.5	9	9.5
10	10.5	11

What are y



Black & W

Women

\$ 55

SALE

Requisitos y configuración para comenzar a aprender Angular

Comencemos a construir un proyecto de muestra de aplicación web completa con Angular

Configurar el entorno de desarrollo angular

En esta sección, le mostraremos cómo configurar su entorno de desarrollo local para que pueda comenzar a desarrollar aplicaciones angulares. El desarrollo de una aplicación real ocurre en un entorno

de desarrollo local que podría ser su máquina personal. Siga nuestras instrucciones de configuración para crear un nuevo proyecto angular.

Requisitos angulares: instale NodeJS y npm

Node.js y npm son fundamentales para el desarrollo web moderno utilizando Angular y otras plataformas. El nodo potencia el desarrollo del cliente y las herramientas de construcción.

Usaremos el administrador de paquetes de nodos (npm) para instalar todas las dependencias de las bibliotecas de JavaScript. [Obtenga estos ahora](#) si no están instalados en su computadora.

Nota: Verifique que está ejecutando al menos el nodo 8.xy npm 6.x ejecutando `node -v` y `npm -v` en una ventana de

terminal / consola. Las versiones anteriores pueden producir errores, pero siempre se recomiendan versiones más nuevas.

La CLI angular

Las aplicaciones angulares se crean y desarrollan principalmente a través de la CLI Angular (herramienta de interfaz de línea de comandos) que ayuda a la creación de proyectos, agregar archivos y realizar una variedad de tareas de desarrollo en curso.

Angular CLI se encarga de la configuración e inicialización de varias bibliotecas. También nos ayuda a agregar componentes, directivas, servicios, etc., a aplicaciones angulares ya existentes. También vale la pena mencionar que la CLI usa Typescript y

Webpack para la agrupación de módulos, Karma para pruebas unitarias y Protractor para una prueba de extremo a extremo. Incluye todo lo que necesita para comenzar a escribir su aplicación Angular de inmediato.

Para instalar la [CLI angular](#) globalmente, ejecute el siguiente comando en su consola

```
npm install -g @angular/cli
```

Nota: aunque no se recomienda, es posible que deba agregar "sudo" delante de estos comandos para instalar las utilidades a nivel mundial.

Nota importante: si tiene una versión anterior de la CLI instalada en su computadora, asegúrese de [actualizarla](#) correctamente [a la última CLI angular](#) .

Comencemos a construir el proyecto de ejemplo de aplicación Angular

¡Iniciar una nueva aplicación angular con la CLI es fácil! Desde su línea de comando, ejecute este comando:

ng new "my-new-angular-app"

El comando anterior creará una carpeta llamada "my-new-angular-app" y copiará todas las dependencias y configuraciones de configuración requeridas.

Angular CLI hace esto por usted:

- 1. Crea un nuevo directorio "my-new-angular-app"**
- 2. Descarga e instala bibliotecas angulares y cualquier otra dependencia**
- 3. Instala y configura TypeScript**
- 4. Instala y configura Karma y Protractor (bibliotecas de prueba)**

También puede usar el comando

ng init

La diferencia entre **ng init** y **ng new**:

- **ng new** requiere que especifique el nombre de la carpeta y creará una carpeta copiando los archivos
- **ng init** copiará los archivos a la carpeta actual.

Ahora, puedes **CD** a la carpeta creada.

Para obtener una vista previa rápida de su aplicación dentro del navegador, use el comando :

ng serve

Este comando ejecuta el compilador en modo de observación (busca cambios en el código y vuelve a compilar si es necesario), inicia el servidor, inicia la aplicación en un navegador y mantiene la aplicación ejecutándose mientras continuamos compilándola.

El servidor de desarrollo de Webpack escucha en el puerto HTTP 4200. Por lo tanto, si abre la url `http://localhost:4200/` verá que la aplicación se está ejecutando.

Uso de la CLI angular para agregar nuevas páginas

Las nuevas funciones del generador. proporcionan una manera fácil de crear páginas y servicios angulares para su aplicación. Esto hace que pasar de una aplicación básica a una aplicación web de navegación con todas las funciones sea mucho más fácil. Yo llamo a eso una curva de aprendizaje fácil :).

Para crear un nuevo componente, puede usar el siguiente comando:

```
ng generate component my-new-component
```

```
ng g component my-new-component # using  
the alias
```

```
✓ Create app/pages/my-page/my-page.html
```

```
✓ Create app/pages/my-page/my-page.ts
```

```
✓ Create app/pages/my-page/my-page.scss
```

CLI angular agregará una referencia a componentes, directivas y tuberías automáticamente en **app.module.ts**

Nota: Consulte [la documentación de CLI angular](#) para obtener más información sobre cómo agregar componentes y otros elementos a su aplicación.

Construir un proyecto de ejemplo de CRUD angular paso a paso

¿Qué vamos a construir?

Este tutorial lo lleva a través de los pasos para crear una aplicación angular con TypeScript.

La aplicación de ejemplo tiene como objetivo ayudarlo a aprender los conceptos fundamentales de Angular Framework.

Con un formato de preguntas y respuestas (Q&A), los usuarios podrán hacer preguntas sobre diferentes conceptos clave angulares y responder a los de otros. Tendrá una lista de algunos conceptos clave angulares, como CLI angular y mecanografiado, y dentro de cada una de estas categorías, una lista de preguntas con las respuestas correspondientes, la opción de votarlas (voto positivo, voto negativo) y algunos formularios para crear / eliminar / actualizar preguntas y respuestas.

La siguiente captura de pantalla es de la página de inicio donde puede ver la lista de categorías:


Categories

Learn all about Angular




Angular
Angular is a platform that makes it easy to build applications with the web.

Angular JS Angular 2 Angular 4 Angular 5



Angular CLI
Angular cli is a command line interface to scaffold and build angular apps using nodejs style modules.

Angular CLI Testing



TypeScript
TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

JavaScript Webpack TypeScript 2.x

El plan para este tutorial es crear una aplicación que lo lleve paso a paso desde la configuración hasta un ejemplo completo que sirva para demostrar las características esenciales de una aplicación profesional: una estructura de proyecto sensible, enlace de datos, servicios, solucionadores, canales, material angular, inyección de dependencia, navegación y acceso remoto a datos.

Aprenderemos suficiente Angular básico para comenzar y ganaremos la confianza de que Angular puede hacer lo que sea que necesitemos.

Cubriremos mucho terreno a nivel introductorio, pero también vincularemos muchas referencias a temas con mayor profundidad.

La aplicación consiste en un CRUD (Crear, Leer, Actualizar, Eliminar) de Preguntas y Respuestas donde las personas pueden publicar nuevas preguntas y responder las preguntas de otras personas.

En el próximo tutorial exploraremos [cómo agregar un back-end para esta aplicación Angular, usando la pila MEAN](#)

.

La aplicación tendrá las siguientes funcionalidades:

- Gestionar preguntas (crear, eliminar)**
- Gestionar respuestas (crear, actualizar, eliminar)**
- Enumere todas las preguntas de una categoría en un formato de lista**
- Enumere todas las respuestas de una pregunta en particular en un formato de lista**
- Permitir que las personas voten Preguntas y respuestas (votos positivos y negativos)**

Así es como se verá la aplicación final:

Categorias / Angular

Learn about: Angular

New Question

Where can I buy beautiful and updated Angular 5 templates?

👤 1 🗳 0 🗒 1 Answers

Internationalization (i18n) in Angular 5

👤 0 🗳 1 🗒 3 Answers

Should I care about AOT?

👤 2 🗳 0 🗒 2 Answers

Should I care about AOT?

New Answer

Yes!! It has lots of benefits such as: Faster compilation. Easier maintenance in production. Smaller Angular framework download size and more.

👍 2 🗨 0 📄 0 ✎

No, it's just the same than JIT, the

👍 0 🗨 1 📄 0 ✎

NEW ANSWER

yes !

Answer

Cancel

En esta primera parte, aprenderá a:

- **Crear clases para representar objetos del modelo.**
- **Crear servicios para crear, actualizar y eliminar objetos.**
- **Cree páginas y componentes para representar las funcionalidades y mostrar la interfaz de usuario.**

Arquitectura angular de aplicaciones

Angular es un **framework** diseñado para crear aplicaciones de una sola página (SPA) y la mayor parte de su diseño de arquitectura se centra en hacerlo de manera efectiva.

Las aplicaciones de una sola página (o SPA) son aplicaciones a las que se accede a través del navegador web como otros sitios web, pero ofrecen interacciones más dinámicas que se asemejan a las aplicaciones nativas móviles y de escritorio.

La diferencia más notable entre un sitio web normal y SPA es la cantidad reducida de actualizaciones de página.

Típicamente, el 95 por ciento del código SPA se ejecuta en el navegador; el resto funciona en el servidor cuando el usuario necesita nuevos datos o debe realizar operaciones seguras como la autenticación.

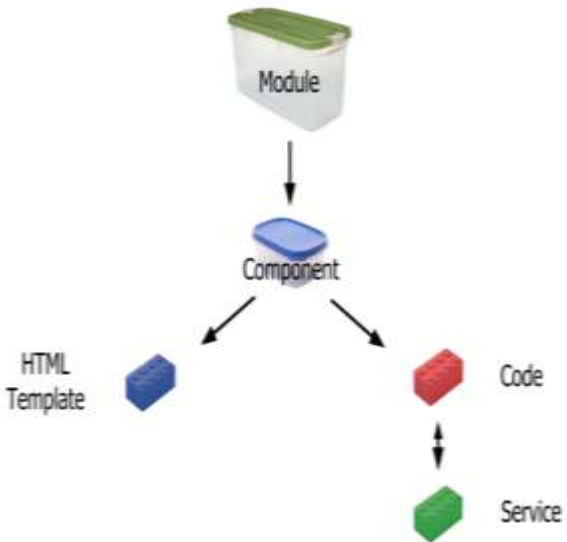
Como resultado, el proceso de representación de la página ocurre principalmente en el lado del cliente.

Módulos Angulares

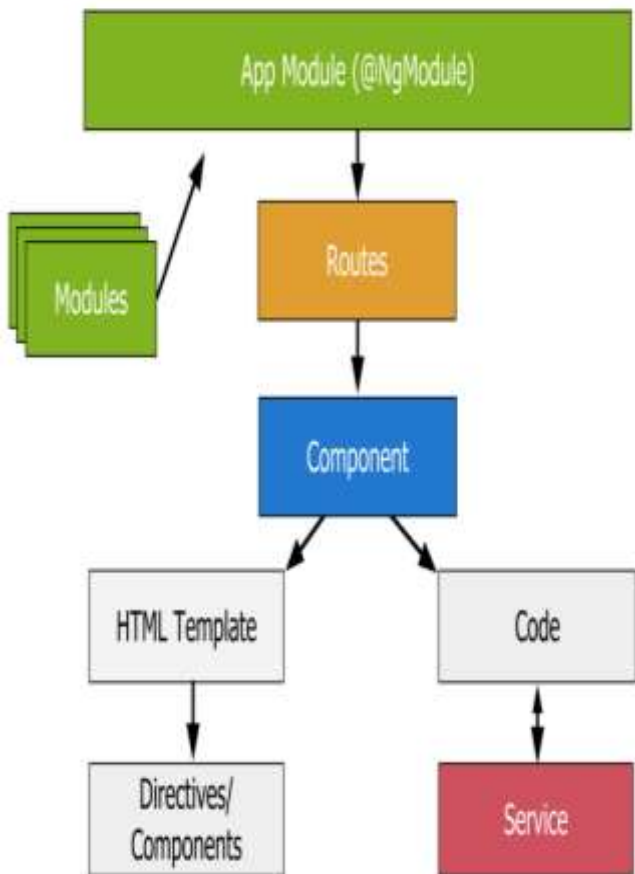
Los módulos ayudan a organizar una aplicación en bloques funcionales cohesivos al envolver

- **componentes,**
- **canales,**
- **directivas y**
- **servicios.**

Solo se trata de la ergonomía del desarrollador.



Las aplicaciones angulares son modulares. Cada aplicación angular tiene al menos un módulo: el módulo raíz, denominado convencionalmente **AppModule**.



El módulo raíz **AppModule** puede ser el único módulo en una aplicación pequeña, pero la mayoría de las aplicaciones tienen muchos más módulos. Como desarrollador, depende de usted decidir cómo usar los módulos.

Por lo general, asigna una funcionalidad principal o una característica a un módulo. Digamos que tiene cuatro áreas principales en su sistema. Cada uno tendrá su propio módulo además del módulo raíz, para un total de cinco módulos.

Ejemplo de modulo:

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

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

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

```
@NgModule({  
  declarations: [AppComponent],  
  imports: [BrowserModule],  
  providers: [],  
  bootstrap: [AppComponent]  
})
```

```
export class AppModule {}
```

decorador @NgModule.

Cualquier módulo angular es una clase con el decorador **NgModule**.

Los decoradores son funciones que modifican las clases de JavaScript.

Básicamente se utilizan para adjuntar metadatos a clases para que conozca la configuración de esas clases y cómo deberían funcionar.

```
@NgModule({  
  declarations: [AppComponent],  
  imports: [BrowserModule],  
  providers: [],  
  bootstrap: [AppComponent]  
})
```

Las propiedades del decorador

@NgModule que describen el módulo son:

- **declarations**: Las clases que pertenecen a este módulo y están relacionadas con las vistas. Hay tres clases en Angular que pueden contener vistas: componentes, directivas y tuberías.

- **exports**: las clases que deberían ser accesibles para otros componentes de los módulos.
- **imports**: módulos cuyas clases son necesarias para los componentes de este módulo.
- **providers**: Servicios presentes en uno de los módulos que se utilizarán en los otros módulos o componentes. Una vez que se incluye un servicio en los proveedores, se vuelve accesible en todas las partes de esa aplicación.

- **bootstrap**: el componente raíz que es la vista principal de la aplicación. Solo el módulo raíz tiene esta propiedad e indica el componente que se iniciará.
- **entryComponents** un componente de entrada es cualquier componente que Angular se carga imperativamente (lo que significa que no está haciendo referencia a él en la plantilla), por tipo.

Componentes angulares

Los componentes son el bloque de construcción más básico de una interfaz de usuario y aplicaciones angulares. Un componente controla una o más secciones en la pantalla (lo que llamamos vistas). Por ejemplo, en este ejemplo tenemos componentes como

`AppComponent` **(the bootstrapped component)**, `CategoriesComponent`, `CategoryQuestionsComponent`, `QuestionAnswersComponent` **etc.**

Un componente es autónomo y representa una parte reutilizable de la interfaz de usuario que generalmente está constituida por tres cosas importantes: vista / clase / selector

- Una pieza de código html que se conoce como la **vista**
- Una **clase** que encapsula todos los datos e interacciones disponibles para esa vista a través de una API de propiedades y métodos diseñados por Angular. Aquí es donde definimos la lógica de la aplicación (qué hace para admitir la vista)

- Y el elemento html mencionado anteriormente también conocido como **selector**.

Los componentes los podemos agrupar en tres tipos:

Nativos: Módulos propios de angular (http, browser, forms, etc..)

Terceros: Módulos realizados por otras personas o empresas y que pueden ser de pago o gratuitos.

Propios: Módulos creado por nosotros mismo.

Estructura del **componente**:

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

```
@Component({
```

```
  selector: "app-root",
```

```
  templateUrl: "./app.component.html",
```

```
  styleUrls: ["./app.component.css"]
```

```
})
```

```
export class AppComponent {
```

```
  title = "APP";
```

```
}
```

Los componentes están formados por código y plantilla HTML y puede tener un selector, por lo que podemos llamarlo en nuestro HTML.

`<appcomponent></appcomponent>`

Cada componente consta de:

imports

```
import { Component } from '@angular/core';  
import { DataService } from '../services/data.service';
```

decorators

```
@Component({  
  selector: 'app-customers',  
  templateUrl: './customers.component.html'  
})
```

class

```
export class CustomersComponent {  
  
}
```

Usando el decorador Angular

@Component proporcionamos

metadatos adicionales que determinan

cómo se debe procesar, instanciar y usar el componente en tiempo de ejecución. Por ejemplo, configuramos la

- plantilla html relacionada con la vista
- configuramos el selector html que vamos a usar para ese componente,
- establecemos hojas de estilo para ese componente.

El Componente pasa datos a la vista mediante un proceso llamado **Data**

Binding (Enlace de datos). Esto se

hace vinculando los elementos **DOM** a las propiedades del componente.

El enlace se puede usar para

- mostrar valores de propiedad al usuario,
- cambiar estilos de elementos,
- responder a un evento de usuario,
- etc.

Un componente debe pertenecer a un **NgModule** para que otro componente o aplicación pueda utilizarlo. Para especificar que un componente es miembro de un NgModule , debe listar NgModule en la propiedad de declaraciones de ese NgModule .

Una nota al margen sobre la importancia de los componentes desde el punto de vista de los principios de la arquitectura de software: es muy importante y se recomienda tener componentes separados, y he aquí por qué. Imagine que tenemos dos bloques de interfaz de usuario diferentes en el mismo

componente y en el mismo archivo. Al principio, pueden ser pequeños pero cada uno podría crecer. Estamos seguros de recibir nuevos requisitos para uno y no para el otro. Sin embargo, cada cambio pone en riesgo ambos componentes y duplica la carga de las pruebas sin ningún beneficio. Si tuviéramos que reutilizar algunos de esos bloques de UI en otra parte de nuestra aplicación, el otro estaría pegado.

Ese escenario viola el [principio de responsabilidad única](#) . Puede pensar que esto es solo un tutorial, pero necesitamos hacer las cosas bien, especialmente si hacerlo bien es fácil y aprendemos cómo construir aplicaciones angulares en el proceso.

Angular fomenta este principio al controlar cada **patch** de la página con su propio componente.

Una aplicación angular típica se parece a un árbol de componentes. El siguiente diagrama ilustra este concepto.

Tenga en cuenta que los componentes modales están del lado del componente principal porque son componentes imperativos que no se declaran en la plantilla html del componente.

AppComponent

BreadcrumbComponent

CategoryQuestions
Component

NewQuestion
ModalComponent

DeleteQuestion
ModalComponent

QuestionAnswers
Component

NewAnswer
ModalComponent

UpdateAnswer
Modal
Component

DeleteAnswer
Modal
Component

Categories
Component

Bloques angulares:

Templates

Los templates se utilizan para definir una vista de componente.

Una template parece HTML normal, pero también tiene algunas diferencias.

Código como * ngFor, {{hero.name}}, (clic) y [hero] usa [la sintaxis de plantilla](#) angular para mejorar las capacidades de marcado HTML. Las templates también pueden incluir componentes personalizados como <custom-element> en forma de etiquetas html no regulares.

Estos componentes se mezclan perfectamente con HTML nativo en los mismos diseños.

Bloques angulares:

servicios

Casi cualquier cosa puede ser un servicio, cualquier valor, función o característica que su aplicación necesite. Un servicio es típicamente una clase con un propósito estrecho y bien definido. Debe hacer algo específico y hacerlo bien. El objetivo principal de Angular Services es compartir recursos entre componentes.

Tome clases de componentes, deben ser esbeltas, el trabajo del componente es permitir la experiencia del usuario (mediar entre la vista y la lógica de la aplicación) y nada más. No obtienen datos del servidor, validan la entrada del usuario o inician sesión directamente en la consola. Delegan tales tareas y todo lo que no es trivial a los servicios.

Los servicios son fundamentales para cualquier aplicación Angular, y los componentes son grandes consumidores de servicios, ya que los ayudan a ser ágiles.

El escenario que acabamos de describir tiene mucho que ver con el principio de Separación de preocupaciones . Angular no aplica estos principios, pero le ayuda a seguir estos principios al facilitar la estructuración de la lógica de su aplicación en servicios y hacer que esos servicios estén disponibles para los componentes a través de la inyección de dependencia.

En nuestra aplicación de ejemplo tenemos tres servicios:

- **AnswersService ,**
- **QuestionsService ,**
- **CategoriesService .**

Cada uno de ellos tiene solo las funciones relacionadas con ellos. En este tutorial específico solo nos centraremos en CategoriesService y en las siguientes partes discutiremos los otros.

CategoriesService tiene los

siguientes métodos:

```
//gets all the question categories from  
//a local json
```

```
getCategories() {
```

```
    return
```

```
this.http.get("./assets/categories.json")
```

```
    .map((res:any) => res.json())
```

```
    .toPromise();
```

```
}
```

```
//finds a specific category by slug
```

```
getCategoryBySlug(slug: string){
```

```
    return this.getCategories()
```

```
    .then(data =>{
```

```
        return
```

```
data.categories.find((category) => {
```

```
    return category.slug == slug;
```

```
    });
```

```
    })
```

```
}
```

Bloques angulares: otros recursos

Los recursos externos como bases de datos, API, etc., son fundamentales, ya que permitirán que nuestra aplicación interactúe con el mundo exterior.

Ahora, profundicemos y asignemos la estructura del proyecto a la arquitectura de la aplicación para que podamos entender mejor cómo interactúan todas las piezas entre sí.

Si desea crear una aplicación web compleja y robusta con Angular, debe consultar [Angular Admin Template](#), que es la Angular Admin Template más completa y avanzada con muchos componentes y mejoras de rendimiento.

Incluye características como Angular Universal, AOT (compilación anticipada), Diseño de materiales, Rutas de carga diferida y muchos componentes hermosos y útiles.

The image is a promotional graphic for the Angular Admin Template. It features a dark purple background with a red horizontal band across the middle. On the left, the text 'ANGULAR' is in white and 'ADMIN TEMPLATE' is in white on a red background. Below this, a list of features is shown with red circular icons. On the right, there is a collage of three devices: a desktop monitor, a tablet, and a smartphone, all displaying the admin template's interface with various charts and tables. At the bottom left, the text '+85 screens and components' is in white on the red band. The bottom left corner has 'ANGULAR TEMPLATES' in white, and the bottom right corner has 'angular-templates.io' in white.

**ANGULAR
ADMIN TEMPLATE**

- Material Design
- Angular CLI & Angular Universal
- Lazy Modules

+85 screens and components

ANGULAR
TEMPLATES

angular-templates.io

Estructura de proyecto de aplicación de ejemplo angular

Después de seguir las instrucciones de configuración para crear un nuevo proyecto en la sección anterior, veamos la anatomía de nuestra aplicación Angular 7. Los procedimientos de configuración de cli instalan muchos archivos diferentes. La mayoría de ellos pueden ser ignorados de manera segura.

En la raíz del proyecto tenemos tres carpetas importantes y algunos archivos importantes:

- **/ src /**
 - **Esta es la carpeta más importante. Aquí tenemos todos los archivos que hacen nuestra aplicación Angular.**
- **/ e2e /**
 - **Esta carpeta es para las pruebas de extremo a extremo de la aplicación, escritas en Jazmín y ejecutadas por el transportador de pruebas**

de transportador e2e. Tenga en cuenta que no ingresaremos detalles sobre las pruebas en esta publicación.

- **nodemodules /**
 - Los paquetes npm instalados en el proyecto con el comando npm install.
- **package.json**
 - Como todas las aplicaciones web modernas, necesitamos un sistema de paquetes y un administrador de paquetes para manejar todas las

bibliotecas y módulos de terceros que utiliza nuestra aplicación. Dentro de este archivo, encontrará todas las dependencias y algunas otras cosas útiles como los scripts npm que nos ayudarán mucho a organizar el flujo de trabajo de desarrollo (agrupación / compilación).

- **tsconfig.json**
 - **Archivo de configuración principal. Debe estar en la ruta raíz, ya que es donde el**

**compilador typescript lo
buscará.**

**Dentro del directorio `/src` encontramos
nuestro código sin compilar. Aquí es
donde tendrá lugar la mayor parte del
trabajo para su aplicación Angular.**

**Cuando ejecutamos `ng serve` ,
nuestro código dentro de `/src` se agrupa
y se transpila en la versión correcta de
Javascript que el navegador entiende
(actualmente, ES5). Eso significa que
podemos trabajar a un nivel más alto
usando TypeScript, pero compilar a la
forma más antigua de Javascript que
necesita el navegador.**

Debajo de esta carpeta encontrará dos estructuras de carpetas principales.

- / app
 - tiene todos los componentes, módulos y páginas sobre los que construirá la aplicación.
- / environments
 - esta carpeta es para administrar las diferentes variables de entorno como **dev** y **prod**. Por ejemplo, podríamos tener una base de datos local para nuestro entorno de desarrollo y una

base de datos de productos para el entorno de producción. Cuando ejecutamos `ng serve`, usará por defecto el `dev env`. Si desea ejecutar en modo de producción, debe agregar el indicador **--prod** al `ng serve`.

- `index.html` /
 - Es la página de host de la aplicación, pero no modificará este archivo con frecuencia, ya que en nuestro caso solo sirve como marcador de

posición. Todos los scripts y estilos necesarios para que la aplicación funcione serán inyectados automáticamente por el proceso de agrupación de paquetes web, por lo que no tiene que hacerlo manualmente. Lo único que me viene a la mente ahora, que puede incluir en este archivo, son algunas metaetiquetas (pero también puede manejarlas a través de Angular).

Y hay otras carpetas secundarias pero también importantes

- **/assets**
 - **en esta carpeta encontrará imágenes, datos de muestra json y cualquier otro activo que pueda necesitar en su aplicación.**

Mejores prácticas angulares: la carpeta app

Este es el núcleo del proyecto. Echemos un vistazo a la estructura de esta carpeta para que tenga una idea de dónde encontrar cosas y dónde agregar sus propios módulos para adaptar este proyecto a sus necesidades particulares.

- /shared (compartido)
 - El **SharedModule** que reside en esta carpeta existe para contener los componentes, directivas y canalizaciones comunes y compartirlos con los módulos que los necesitan.
 - Importa

CommonModule

porque su componente necesita directivas comunes. Notará que reexporta otros módulos. Si revisa la aplicación, puede observar que muchos

**componentes que requieren
directivas SharedModule
también usan NgIf y NgFor de
CommonModule y se unen a
las propiedades del
componente con
[(ngModel)], una directiva
en FormsModule . Los
módulos que declaran estos
componentes tendrían que
importar CommonModule ,
FormsModule y SharedModule .**

- **Puede reducir la repetición
si SharedModule reexporta
CommonModule y FormsModule
para que los importadores**

de SharedModule obtengan CommonModule y FormsModule de forma gratuita.

SharedModule aún puede exportar FormsModule sin FormsModule entre sus importaciones.

- **/ styles (estilos)**
 - **Aquí encontrará todas las variables, mixins, estilos compartidos, etc., que harán que su aplicación sea personalizable y ampliable.**
 - **¿Quizás no conoces a Sass? En resumen, es un superconjunto de CSS que**

facilitará y acelerará sus ciclos de desarrollo increíblemente.

- **/services**

- **Aquí encontrarás todos los servicios necesarios en esta aplicación. Cada servicio tiene solo las funciones relacionadas con él.**

- **Otras carpetas**

- **Para ganar en la modularidad del código, hemos creado una carpeta para cada componente. Dentro de esas carpetas encontrará todos los**

archivos relacionados para las páginas incluidas en ese componente. Esto incluye el html para el diseño, sass para los estilos y el componente de la página principal.

- **app.component.html**
 - **Esto sirve como el esqueleto de la aplicación. Normalmente tiene un `<router-outlet>` para representar las rutas y su contenido. También se puede envolver con el contenido que desea que**

esté en cada página (por ejemplo, una barra de herramientas).

- **app.component.ts**
 - **Es el componente angular que proporciona funcionalidad al archivo app.component.html que acabo de mencionar.**
- **app-routing.module.ts**
 - **Aquí definimos las rutas principales. Estas rutas están registradas en Angular RouterModule en AppModule. Si utiliza módulos diferidos, las rutas**

secundarias de otros módulos diferidos se definen dentro de esos módulos.

- **app.module.ts**
 - **Este es el módulo principal del proyecto que iniciará la aplicación.**

A medida que avancemos en este tutorial, crearemos más páginas y realizaremos una navegación básica.

Angular app navigation and routing

Después de ver el diagrama de componentes y la estructura del proyecto, esta es la navegación que proponemos para la aplicación.

Comenzamos en la página de categorías y desde allí solo podemos navegar a las preguntas de una de las categorías.

Luego, siga las flechas de la imagen a continuación para ver las otras navegaciones disponibles en esta aplicación de ejemplo angular 7.

Tenga en cuenta que los modales para crear, actualizar y eliminar no se muestran en esta imagen porque no representan una navegación de la aplicación. Otra cosa importante a tener en cuenta es que usamos Breadcrumbs para volver a las páginas anteriores.

Categories

Learn all about Angular



Angular

Angular is a platform that makes it easy to build applica...

Angular JS

Angular 2

Angular 4

Angular 5



Angular CLI

Angular CLI is a command line interface to scaffold and ...

Angular CLI

Testing



TypeScript

TypeScript is a typed superset of JavaScript that comp...

JavaScript

ReactJS

TypeScript 2.x

Categories / Angular

Learn about: Angular

New Question

Where can I buy beautiful and updated Angular 5 templates?

👤 🗑️ 📄 1 Answer

Internationalization (i18n) in Angular 2

👤 🗑️ 📄 3 Answers

Should I care about AOT?

👤 🗑️ 📄 3 Answers

Answers

Categories / angular / Should I care about AOT?

Should I care about AOT?

New Answer

Yes! It has lots of benefits such as: Faster rendering, Fewer asynchronous requests, Smaller Angular framework download size and more.

👤 🗑️ 📄 ✎

No, it's just the same than JIT, there is no benefit at all

👤 🗑️ 📄 ✎

Antes de comenzar a pensar en la navegación, debemos considerar el tipo y la cantidad de datos que desea mostrar en su aplicación. No olvide que usará la navegación para mostrar y estructurar sus datos, es por eso que debe seguir la estructura de información de su aplicación y no al revés.

Es importante mantener las mejores prácticas para el diseño de navegación. Esto garantiza que las personas puedan usar y encontrar las funciones más valiosas en su aplicación.

La buena navegación, como el buen diseño, es invisible.

Si está buscando una navegación más compleja con menús laterales, pestañas y muchas otras configuraciones avanzadas, definitivamente debe consultar la [Plantilla de administración angular](#)

Monday,
16 of October

medRxiv preprint doi: <https://doi.org/10.1101/2020.05.14.20084000>; this version posted May 15, 2020. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted medRxiv a license to display the preprint in perpetuity. It is made available under a CC-BY 4.0 International license.



26°

London,
2:41 PM

75 88

1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

0

Learn English Online
2008-09-08

0

W.H.J. de Groot w.h.j.de.groot@uu.nl

0

Workload

WASSAG 4495.00 4495.00 4495.00

NYSE	12,046.81	↑ 100.00
------	-----------	----------

DOWJ 17,344.81

APPL	180.00	+0.35%
------	--------	--------

— 1999 included

Construyendo una aplicación de ejemplo Angular paso a paso

**Navegación de la aplicación usando
el módulo de enrutador angular**

Un poco más sobre la navegación.

Angular tiene un módulo específico dedicado a la navegación y enrutamiento, el **RouterModule**. Con este módulo puede crear rutas, lo que le permite moverse de una parte de la aplicación a otra o de una vista a otra.

Para que las rutas funcionen, necesita un ancla o elemento en la interfaz de usuario para asignar acciones (generalmente clics en elementos) a rutas (rutas URL). Usamos la directiva **routerLink** para este propósito. Por ejemplo, cuando el usuario hace clic en el nombre de una Categoría en la IU, Angular, a través de la directiva

routerLink, sabe que debe navegar a la siguiente

`url:http://localhost:4200/questions/about/category-name`

```
<a class="list-title"
```

```
[routerLink]="['/questions/about',
```

```
category.slug"> {{category.title}} </a>
```

A continuación, deberá asignar las rutas de URL a los componentes. En la misma carpeta que el módulo raíz, cree un archivo de configuración llamado **app.routes.ts**(si aún no tiene uno) con el siguiente código.


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

export const routes: Routes = [
  {
    path: '',
    component: CategoriesComponent,
    resolve: {
      data: CategoriesResolver
    }
  },
  {
    path: 'questions/about/:categorySlug',
    component: CategoryQuestionsComponent,
    resolve: {
      data: CategoryQuestionsResolver
    }
  },
  {
    path: 'question/:questionSlug',
    component: QuestionAnswersComponent,
    resolve: {
      data: QuestionAnswersResolver
    }
  }
];
```

Para cada ruta proporcionamos un **path** (también conocida como la URL) y el componente que debe representarse en esa ruta. La cadena vacía para la **CategoriesComponent's** ruta significa que el componente de categorías se representará cuando no haya una URL (también conocida como la **root path**).

Tenga en cuenta que para cada ruta también tenemos una **resolve**.

El uso de una **resolve** en nuestras rutas de navegación nos permite buscar previamente los datos del componente antes de que se active la ruta. El uso de **resolvees** una muy buena práctica para asegurarse de que todos los datos necesarios estén listos para que nuestros componentes los usen y evitar mostrar un componente en blanco mientras espera los datos.

Por ejemplo, en usamos a

CategoriesResolver para obtener la lista de categorías. Una vez que las categorías están listas, activamos la ruta.

Tenga en cuenta que si la **resolve**

Observable no se completa, la navegación no continuará.

Finalmente, el módulo raíz también debe conocer las rutas que definimos anteriormente.

Agregue una referencia a las rutas en la propiedad de importación de **AppModule**.

```
import { routes } from './app.routes';

imports: [
  RouterModule.forRoot(routes,
    { useHash: false }
  )
],
```

Observe cómo usamos **forRoot**(o eventualmente **forChild**) métodos en **RouterModule**(los documentos explican la diferencia en detalle, pero por ahora solo sepa que **forRoot** solo debe llamarse una vez en su aplicación para rutas de nivel superior).

Componentes de diseño de material angular para mejorar UI / UX

Material angular 2 vs ngx-bootstrap

Hay algunas bibliotecas que proporcionan componentes de alto nivel que le permiten construir rápidamente una interfaz agradable para su aplicación. Estos incluyen modales, ventanas emergentes, tarjetas, listas, menús, etc. Son elementos de interfaz de usuario reutilizables que sirven como bloques de construcción para su

aplicación móvil, compuestos de HTML, CSS y, a veces, JavaScript.

Dos de las bibliotecas de componentes de IU más utilizadas son Angular Material y ngx-bootstrap. Angular Material es la biblioteca oficial de UI angular y proporciona toneladas de componentes.

Por otro lado, ngx-bootstrap proporciona una serie de componentes angulares hechos sobre el marco de Twitter Bootstrap.

En este tutorial angular, vamos a utilizar material angular, pero no dude en elegir el que mejor se adapte a sus necesidades, ya que son súper completos y robustos.

En esta aplicación de ejemplo angular 7, tenemos diferentes diseños. Para cada vista necesitamos diferentes componentes de la interfaz de usuario. Aquí hay una breve lista con los componentes más importantes que utilizamos para cada vista y un enlace a los detalles de la implementación de esa vista.

- **Vista de categorías**
 - **Una lista que muestra los diferentes conceptos angulares que necesita aprender.**
 - **Componentes materiales:**
 - **Componente de lista para la lista de categorías**
 - **Componente de chips para las etiquetas de categoría**
- **Vista de preguntas de categoría**
 - **Una vista para mostrar todas las preguntas de una categoría particular.**

- **Componentes materiales:**
 - **Componente de lista para la lista de preguntas**
 - **Componente del botón**
 - **Componente de diálogo para los modales**
- **Vista de preguntas y respuestas**
 - **Una vista para mostrar todas las respuestas de una pregunta en particular.**
 - **Componentes materiales:**

- **Componente de lista para la lista de respuestas**
 - **Componente del botón**
 - **Componente de diálogo para los modales**
- **Nueva pregunta y nueva respuesta modals**
 - **Modalidades para crear / actualizar preguntas y respuestas**
 - **Componentes materiales:**

- **Componente de diálogo para gestionar el modal**

También utilizamos Componente de barra de herramientas de material angular para la navegación de migas de pan.

No dude en excavar la biblioteca de componentes de la interfaz de usuario que Angular Material tiene en la [página de documentación de](#) sus [componentes](#)

▪

Agregar un backend a nuestro proyecto de ejemplo angular

Diferentes alternativas para integraciones de datos API de back-end

La clave para una aplicación en evolución es crear servicios reutilizables para administrar todas las llamadas de datos a su back-end.

Como ya sabrás, hay muchas maneras en lo que respecta al manejo de datos y las implementaciones de back-end. En este tutorial explicaremos cómo

consumir datos de un archivo **json** estático con datos ficticios. En el siguiente tutorial [Aprenda a construir una aplicación MEAN stack](#) , aprenderá a construir y consumir datos desde una API REST con Loopback (un marco node.js perfectamente adecuado para las API REST) y MongoDB (para almacenar los datos).

Ambas implementaciones (json estática y API remota remota) deben preocuparse por el lado de la aplicación del problema, cómo manejar las llamadas de datos. Esto funciona igual y es independiente de la forma en que implemente el back-

end. Hablaremos sobre modelos y servicios y cómo trabajan juntos para lograr esto.

Alentamos el uso de modelos en combinación con servicios para el manejo de datos desde el backend hasta el flujo de presentación.

Modelos de dominio

Los modelos de dominio son importantes para definir y aplicar la lógica empresarial en las aplicaciones y son especialmente relevantes a medida que las aplicaciones se hacen más grandes y más personas trabajan en ellas. Al mismo tiempo, es importante que mantengamos nuestras aplicaciones SECAS y que se puedan mantener moviendo la lógica fuera de los componentes mismos a clases separadas (modelos) a las que se pueda recurrir. Un enfoque modular como este hace que la lógica de negocios de

nuestra aplicación sea reutilizable. Para obtener más información sobre esto, visite esta excelente publicación sobre modelos de dominio angular 2.

Servicios de datos

Angular le permite crear múltiples servicios de datos reutilizables e inyectarlos en los componentes que los necesitan. Refactorizar el acceso a datos a un servicio separado, mantiene el componente delgado y enfocado en soportar la vista. También facilita la prueba unitaria del componente con un servicio simulado. Para obtener más información sobre esto, visite la documentación de angular 2 sobre los servicios.

En nuestro caso, definimos un modelo para los datos de categorías de preguntas que extraemos del archivo json estático. Este modelo es utilizado por el `categories.service.ts`.

```
//in category.model.ts
export class CategoryModel {
  slug: string;
  title: string;
  image: string;
  description: string;
  tags: Array<Object>;
}

//in categories.service.ts
getCategories():
Promise<CategoryModel[]> {
  return
this.http.get("./assets/categories.json
")
  .toPromise()
  .then(res => res.json() as
CategoryModel[])
}
```

Y utilizamos este servicio en el `categories.resolver.ts` que buscamos las categorías para ver los datos.

```
//in categories.resolver.ts
import { Injectable } from
 '@angular/core';
import { Resolve } from
 "@angular/router";
import { CategoriesService } from
 "../services/categories.service";

@Injectable()
export class CategoriesResolver
implements Resolve<any> {

    constructor(private
categoriesService: CategoriesService) {

}

    resolve() {
        return new Promise((resolve,
reject) => {
```

```
    let breadcrumbs = [
      { url: '/', label: 'Categories' }
    ];

    //get categories from local json
file

this.categoriesService.getCategories()
  .then(
    categories => {
      return resolve({
        categories: categories,
        breadcrumbs: breadcrumbs
      });
    },
    err => {
      return resolve(null);
    }
  )
  });
}
```

Cada vez que agreguemos un nuevo servicio, recuerde que el inyector

angular no sabe cómo crear ese servicio de manera predeterminada. Si ejecutamos nuestro código ahora, Angular fallaría con un error. Después de crear servicios, tenemos que enseñarle al inyector angular cómo hacer ese Servicio registrando un proveedor de Servicios.

De acuerdo con la página de documentación angular para [la inyección de dependencias](#), hay dos formas de registrar el proveedor de servicios: en el componente mismo o en el módulo (NgModule). En nuestro caso, registramos todos los servicios en `elapp.module.ts`

```
//in app.module.ts
@NgModule({
  declarations: [
    AppComponent,
    CategoriesComponent,
    CategoryQuestionsComponent,
    NewQuestionModalComponent,
    NewAnswerModalComponent,
    UpdateAnswerModalComponent,
    QuestionAnswersComponent,
    DeleteQuestionModalComponent,
    DeleteAnswerModalComponent
  ],
  imports: [
    RouterModule.forRoot(routes,
      { useHash: false }
    ),
    SharedModule
  ],
  entryComponents: [

  ],
  providers: [
    CategoriesService,
    QuestionsService,
    AnswersService,
    CategoryQuestionsResolver,
    CategoriesResolver,
    QuestionAnswersResolver
  ],
```

```
bootstrap: [AppComponent]  
}))  
  
export class AppModule { }
```

Una nota al margen sobre la importancia de la inyección de dependencias desde el punto de los principios de la arquitectura de software: ¿ Recuerdas que acabamos de mencionar que "inyectamos" servicios de datos en los componentes que los necesitan? Bueno, este concepto se llama inyección de dependencia y es muy importante saber más sobre esto. ¿Nuevos () los Servicios? ¡De ninguna manera!

Esa es una mala idea por varias razones, que incluyen:

- Nuestro componente tiene que saber cómo crear el Servicio. Si alguna vez cambiamos el constructor del Servicio, tendremos que encontrar cada lugar donde creamos el servicio y arreglarlo. Correr alrededor del código de parche es propenso a errores y se suma a la carga de la prueba.**
- Creamos un nuevo servicio cada vez que usamos nuevo. ¿Qué pasa si el servicio debe almacenar en caché los resultados y**

compartir ese caché con otros?

No pudimos hacer eso.

- **Estamos bloqueando el Componente (donde renovamos el servicio) en una implementación específica del Servicio. Será difícil cambiar las implementaciones para diferentes escenarios.**

¿Podemos operar sin conexión?

¿Necesitaremos diferentes versiones simuladas bajo prueba?

No es fácil.

Lo entendemos. Realmente lo hacemos.

Pero es tan ridículamente fácil evitar estos problemas que no hay excusa para hacerlo mal.

Siguiente: ejemplo angular avanzado

Pensamientos finales y próximos pasos.

En este tutorial paso a paso angular, pasamos de los conceptos básicos y por qué de Angular Framework a construir una aplicación Angular 7 completa utilizando componentes de material angular. Explicamos uno por uno los principales bloques de construcción de una aplicación Angular, así como las mejores prácticas para construir una aplicación completa con Angular. Además, este tutorial muestra cómo

configurar su entorno de desarrollo para que pueda comenzar a desarrollar aplicaciones angulares en su computadora ahora mismo.

En este tutorial solo explicamos la primera parte del ejemplo de código, que es la lista de categorías obtenida de un archivo json estático. En nuestros próximos tutoriales exploraremos más a fondo los detalles de la [implementación de la API de back-end remota usando Loopback](#) y cómo implementarla en un servidor remoto.

Con suerte, no se encontró con ningún problema con este tutorial paso a paso

de Learn Angular desde cero, pero si lo hizo, no dude en publicar en la sección de comentarios a continuación.

Recuerde que puede obtener el código fuente completo de esta aplicación Angular 7 haciendo clic en el botón **OBTENER EL CÓDIGO desde el principio de esta página.**

Si desea crear una aplicación web compleja y robusta con Angular 7, debe consultar [Angular Admin Template](#), que es la plantilla Angular 7 más completa y avanzada con muchos componentes y mejoras de rendimiento. Incluye características como Angular Universal, AOT (compilación anticipada), diseño de

**materiales, rutas de carga diferida y
muchos componentes útiles y
hermosos.**