Curso de Angular

Introducción a Angular

Empezar

Descripción general

Este curso te brindará los conocimientos necesarios para empezar a desarrollar aplicaciones web utilizando Angular. Aprenderás desde los conceptos básicos hasta las funcionalidades más avanzadas de este poderoso framework.

01 Angular

¿Qué es Angular?

01 | ¿Qué es Angular?

Angular es un framework de desarrollo de aplicaciones web de código abierto creado por Google. Es una herramienta poderosa y versátil que permite construir aplicaciones web de una manera eficiente y escalable.

Arquitectura y características principales

Una de las características más destacadas de Angular es su arquitectura basada en componentes. En lugar de construir una aplicación web como un conjunto de páginas

independientes, en Angular se construye una aplicación como una colección de componentes reutilizables.

Cada componente en Angular se compone de tres partes principales: el template, que define la estructura y la apariencia del componente; la lógica del componente, que se define en un archivo TypeScript; y los estilos, que definen la apariencia visual del componente.

Además de su arquitectura basada en componentes, Angular ofrece una serie de características adicionales que lo hacen una herramienta muy poderosa. Algunas de estas características son:

- Inyección de dependencias: en Angular, se utiliza el patrón de inyección de dependencias para administrar las dependencias entre los diferentes componentes de la aplicación. Esto permite una mayor modularidad y facilita la reutilización de código.
- Enrutamiento: Angular ofrece un sistema de enrutamiento que permite navegar entre diferentes páginas dentro de una aplicación web. Esto facilita la construcción de aplicaciones de una sola página (Single Page Applications).
- Formularios y validación: Angular cuenta con un completo sistema para la creación y validación de formularios en aplicaciones web. Esto incluye la capacidad de realizar validaciones en tiempo real, mostrar mensajes de error y realizar acciones dependiendo del estado del formulario.
- Comunicación con el servidor: Angular se integra de manera nativa con APIs RESTful, lo que facilita la comunicación entre la aplicación web y un servidor. Esto permite realizar solicitudes HTTP, recibir respuestas y manejar errores de manera sencilla.
- Pipes: Los pipes en Angular permiten realizar transformaciones y formateo de datos en el template de manera fácil y eficiente. Angular cuenta con varios pipes incorporados, como el de fecha o el de moneda, y además se pueden crear pipes personalizados para adaptarse a las necesidades específicas de la aplicación.
- Manejo de estados con RxJS: RxJS es una biblioteca que se utiliza en Angular para el manejo de flujos de datos asíncronos. Permite trabajar con observables y suscripciones, lo

que facilita el manejo del estado de la aplicación y la reactividad de la interfaz de usuario.

En resumen, Angular es un framework de desarrollo web que ofrece una arquitectura basada en componentes, junto con características poderosas como la inyección de dependencias, el enrutamiento, los formularios y validaciones, la comunicación con el servidor, los pipes y el manejo de estados con RxJS. Esta combinación de características hace que Angular sea una herramienta potente y completa para el desarrollo de aplicaciones web modernas.

Conclusión - ¿Qué es Angular?

¿Qué es Angular? En este curso, hemos explorado a fondo qué es Angular y cómo se utiliza para construir aplicaciones web robustas y escalables. Aprendimos sobre la arquitectura y las características principales de Angular, así como la instalación y configuración del entorno de desarrollo. Ahora tienes una base sólida para empezar a utilizar Angular en tus proyectos.

Componentes y Templates

02 | Componentes y Templates

Componentes y Templates

En Angular, los componentes son la base fundamental para desarrollar aplicaciones. Los componentes son bloques de construcción reutilizables que encapsulan la lógica y la interfaz de usuario de una parte específica de la aplicación. Están compuestos por un template, que define la estructura y el diseño de la interfaz de usuario, y por una clase, que contiene la lógica y los datos relacionados con el componente.

Creación de componentes

Para crear un nuevo componente en Angular, se utiliza el comando `ng generate component` seguido del nombre del componente. Esto generará automáticamente los archivos necesarios, como el archivo de clase del componente, el archivo de plantilla y el archivo de estilo.

Estructura de un componente

Un componente en Angular está compuesto por varios elementos clave:

- **Archivo de clase**: Este archivo contiene la lógica y la funcionalidad del componente. Aquí se definen las propiedades y métodos necesarios para procesar los datos y realizar acciones en el componente.
- **Archivo de plantilla**: El archivo de plantilla define la estructura y el diseño de la interfaz de usuario del componente. Se utiliza HTML para estructurar el contenido y se pueden utilizar directivas y enlaces de datos para interactuar con la lógica del componente.
- **Archivo de estilo**: El archivo de estilo define los estilos CSS para el componente. Se pueden utilizar selectores de clase, ID o etiquetas HTML para aplicar estilos específicos al componente.

Enlace de datos y eventos en el template

En Angular, se utiliza la sintaxis de doble llave `{{}}` para enlazar datos entre el componente y su template. Esto permite mostrar y actualizar dinámicamente los datos en la interfaz de usuario.

Además del enlace de datos, también se pueden vincular eventos en el template para capturar las interacciones del usuario. Esto se logra utilizando la sintaxis de paréntesis `( )` seguida del nombre del evento y el código que se ejecutará cuando ocurra el evento.

En resumen, los componentes y los templates son elementos clave en Angular para desarrollar aplicaciones. Los componentes encapsulan la lógica y la interfaz de usuario de una parte específica de la aplicación, mientras que los templates definen la estructura y el diseño de la interfaz de usuario. El enlace de datos y eventos en el template permite interactuar dinámicamente con la lógica del componente.

Conclusión - Componentes y Templates

Componentes y Templates En este módulo, hemos aprendido cómo crear componentes en Angular y cómo utilizarlos en nuestras aplicaciones. También hemos explorado la estructura de un componente y cómo enlazar datos y eventos en el template. Con esta base, podrás crear componentes reutilizables y dinámicos en tus proyectos con Angular.

Directivas

Las directivas en Angular son una herramienta poderosa que permiten manipular y controlar la apariencia y comportamiento de los elementos HTML de forma dinámica. Estas directivas son marcadores en el código HTML que le indican al framework de Angular qué acciones debe tomar en relación con esos elementos.

Existen dos tipos de directivas en Angular: las directivas incorporadas y las directivas personalizadas.

Directivas incorporadas

Angular viene pre-configurado con un conjunto de directivas incorporadas que se pueden utilizar para realizar acciones comunes en la manipulación del DOM. Algunas de las directivas incorporadas más utilizadas son:

- **ngIf**: Esta directiva permite mostrar u ocultar un elemento en función de una condición booleana.
- **ngFor**: Esta directiva se utiliza para iterar sobre una lista de elementos y generar contenido HTML dinámicamente para cada elemento.

Estas directivas incorporadas se utilizan añadiendo un atributo al elemento HTML al que se desea aplicar la directiva, seguido de un valor que cumpla con la condición correspondiente.

Directivas personalizadas

Además de las directivas incorporadas, es posible crear directivas personalizadas en Angular para abordar necesidades específicas. Estas directivas personalizadas

permiten escribir lógica personalizada que puede ser reutilizada en diferentes partes de la aplicación.

La creación de una directiva personalizada implica definir una clase, especificar metadatos y asociar la directiva a un elemento HTML utilizando el atributo correspondiente.

Una vez creada la directiva personalizada, se puede utilizar en cualquier lugar dentro de la aplicación, simplemente agregando el atributo de la directiva al elemento HTML deseado.

Manipulación del DOM con directivas

Una de las principales ventajas de las directivas en Angular es su capacidad para manipular y controlar el DOM de forma programática. Esto significa que se puede agregar, eliminar o modificar elementos HTML, clases, estilos y atributos utilizando directivas.

Para lograr esto, las directivas pueden acceder al elemento HTML al que están asociadas utilizando el objeto ElementRef y utilizar métodos y propiedades para realizar las acciones deseadas.

La manipulación del DOM se utiliza a menudo para implementar funcionalidades avanzadas en las aplicaciones Angular, como la interacción con eventos, actualización de estilos dinámicos y la creación de elementos adicionales en respuesta a ciertas acciones del usuario.

Recuerda que el uso adecuado de las directivas es fundamental para construir aplicaciones Angular eficientes y mantenibles. Es importante considerar la lógica y el rendimiento al utilizar directivas, evitando manipulaciones excesivas o innecesarias del DOM.

En resumen, las directivas en Angular son una herramienta esencial para manipular y controlar elementos HTML de forma dinámica. Ya sea utilizando las directivas incorporadas o creando directivas personalizadas, se pueden implementar fácilmente funcionalidades avanzadas y mejorar la interactividad de la aplicación.

Conclusión - Directivas

Directivas En este módulo, hemos explorado las directivas en Angular, tanto las incorporadas como las personalizadas. Aprendimos cómo utilizar directivas como ngIf y ngFor para manipular el DOM y controlar la visualización de elementos en nuestra aplicación. Con este conocimiento, podrás mejorar la interactividad y la flexibilidad de tus aplicaciones Angular.

Servicios y Dependencia

04 | Servicios y Dependencia

Los servicios en Angular son clases que se utilizan para organizar y compartir lógica o datos entre diferentes componentes. Actúan como la capa de negocio de nuestra aplicación y nos permiten encapsular la funcionalidad que utilizaremos en varios lugares.

Creación y uso de servicios

Para crear un servicio en Angular, podemos utilizar el comando `ng generate service nombre-servicio` en la línea de comandos. Esto generará automáticamente una clase con el sufijo "Service", que contendrá nuestro servicio.

Una vez que hemos creado nuestro servicio, debemos registrarlo en el módulo principal de nuestra aplicación. Esto se hace en el archivo `app.module.ts`, importando el servicio y agregándolo en la sección de `providers`.

Luego, podemos inyectar nuestro servicio en cualquier componente o servicio que lo necesite. Para hacerlo, simplemente debemos declarar una dependencia en el constructor del componente o servicio y Angular se encargará de proporcionar una instancia del servicio cuando se inicialice.

Inyección de dependencias

La inyección de dependencias es un patrón de diseño que nos permite proporcionar las dependencias necesarias a nuestros componentes o servicios en lugar de crearlas directamente dentro de ellos. Esto nos ayuda a escribir código más modular, reutilizable y fácil de probar.

En Angular, el sistema de inyección de dependencias se encarga de crear las instancias de las clases y de proporcionar las dependencias automáticamente.

Existen tres tipos de inyección de dependencias en Angular:

1. **Inyección de dependencia basada en constructor**: aquí declaramos una dependencia en el constructor de un componente o servicio y Angular se encarga de proporcionar una instancia de esta dependencia al momento de crear la instancia del componente o servicio.
   `constructor(private servicio: NombreServicio) { }`
2. **Inyección de dependencia basada en propiedad**: aquí declaramos una dependencia como una propiedad del componente o servicio y Angular se encarga de asignar la instancia de esta dependencia automáticamente.

```
@Injectable()
export class NombreComponente {
  constructor() { }

  @Input() servicio: NombreServicio;
}
```

3. **Inyección de dependencia basada en parámetro**: aquí declaramos una dependencia como un parámetro de un método y Angular se encarga de proporcionar una instancia de esta dependencia al momento de llamar al método.

```
export class NombreComponente {
  constructor() { }

  metodoServicio(@Inject(NombreServicio) servicio: NombreServicio) { }
}
```

Comunicación entre componentes usando servicios

Los servicios también nos permiten establecer una comunicación eficiente entre diferentes componentes. Esto se logra mediante la inyección de dependencias del mismo servicio en los componentes que necesitan compartir información.

Cuando un componente actualiza un dato en el servicio, todos los componentes que tienen una dependencia de ese servicio tienen acceso a la última versión del dato. Esto evita la necesidad de establecer una conexión directa entre componentes y facilita el intercambio de datos entre ellos.

En resumen, los servicios y la inyección de dependencias son componentes clave en el desarrollo de aplicaciones Angular, ya que nos permiten compartir y reutilizar código, crear componentes desacoplados y establecer una comunicación eficiente entre ellos. Su correcto uso nos ayuda a mantener nuestro código modular, legible y fácil de mantener.

Conclusión - Servicios y Dependencia

Servicios y Dependencia En este módulo, hemos aprendido cómo crear y utilizar servicios en Angular, así como la importancia de la inyección de dependencias. También exploramos cómo comunicarse entre componentes utilizando servicios. Con este conocimiento, podrás separar la lógica de negocio de tus componentes y crear aplicaciones más modulares y mantenibles.

Enrutamiento en Angular

05 | Enrutamiento en Angular

El enrutamiento es uno de los conceptos fundamentales en Angular. Permite la navegación entre diferentes páginas de una aplicación y la gestión de parámetros de ruta. En esta sección del curso, aprenderás cómo configurar rutas, cómo navegar entre páginas y cómo trabajar con rutas anidadas.

Configuración de rutas

La configuración de rutas en Angular se realiza en el módulo de la aplicación. Para cada ruta, se define un path (ruta de acceso) y un componente asociado. Al configurar las rutas, también se puede especificar un componente por defecto que se cargará cuando no se encuentre ninguna ruta definida.

A continuación, se muestra un ejemplo de configuración de rutas en el módulo de una aplicación Angular:

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

import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent }
];

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

En este ejemplo, se define una ruta vacía que carga el componente HomeComponent, una ruta '/about' que carga el componente AboutComponent y una ruta '/contact' que carga el componente ContactComponent.

Navegación entre páginas

Una vez configuradas las rutas, se puede implementar la navegación entre páginas en la aplicación. Para ello, se utiliza el servicio Router de Angular. El Router proporciona métodos para navegar a una ruta específica, navegar hacia adelante o hacia atrás en el historial de navegación, entre otros.

A continuación, se muestra un ejemplo de cómo utilizar el servicio Router para navegar a través de las rutas definidas en la aplicación:

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

@Component({
  selector: 'app-navigation',
  template: `
    <nav>
      <ul>
        <li><a (click)="navigateToHome()">Home</a></li>
        <li><a (click)="navigateToAbout()">About</a></li>
        <li><a (click)="navigateToContact()">Contact</a></li>
      </ul>
    </nav>
    <router-outlet></router-outlet>
  `
})
export class NavigationComponent {
  constructor(private router: Router) {}

  navigateToHome() {
    this.router.navigate(['/']);
  }

  navigateToAbout() {
    this.router.navigate(['/about']);
```

```
  }

  navigateToContact() {
    this.router.navigate(['/contact']);
  }
}
```

En este ejemplo, se utiliza el método navigate() del servicio Router para navegar a una ruta específica cuando se hace clic en los enlaces de navegación. El método navigate() recibe como parámetro un array que representa la ruta a la que se desea navegar.

Parámetros de ruta y rutas anidadas

En Angular, es posible pasar parámetros a través de las rutas y también crear rutas anidadas. Los parámetros de ruta se utilizan para pasar información específica a un componente, como un identificador o un filtro de búsqueda. Las rutas anidadas permiten estructurar y organizar las URL de la aplicación.

A continuación, se muestra un ejemplo de cómo definir rutas con parámetros y rutas anidadas en Angular:

```
const routes: Routes = [
 { path: '', component: HomeComponent },
 { path: 'products/:id', component: ProductDetailComponent },
 {
   path: 'admin',
   component: AdminComponent,
   children: [
     { path: 'dashboard', component: DashboardComponent },
     { path: 'settings', component: SettingsComponent },
   ]
```

```
  }
];
```

En este ejemplo, se define una ruta '/products/:id' que utiliza un parámetro de ruta llamado 'id' para identificar un producto específico en la aplicación. También se define una ruta anidada '/admin' que carga el componente AdminComponent y tiene dos rutas hijas ('/admin/dashboard' y '/admin/settings') que cargan los componentes DashboardComponent y SettingsComponent, respectivamente.

Conclusiones

El enrutamiento en Angular es un aspecto fundamental para construir aplicaciones web complejas y permite la navegación entre páginas, la gestión de parámetros de ruta y la creación de rutas anidadas. Al comprender y aplicar correctamente el enrutamiento en Angular, podrás crear aplicaciones más dinámicas y flexibles.

Recuerda siempre verificar y probar la configuración de rutas y la navegación entre páginas para asegurarte de que la experiencia del usuario sea fluida y sin errores.

Conclusión - Enrutamiento en Angular

Enrutamiento en Angular En este módulo, hemos aprendido cómo configurar las rutas en Angular y cómo navegar entre páginas. También exploramos cómo utilizar parámetros de ruta y rutas anidadas para crear una navegación más dinámica. Con

este conocimiento, podrás crear aplicaciones de una sola página (SPA) con enrutamiento efectivo en Angular.

Formularios y Validación

06 | Formularios y Validación

En esta sección del curso "Curso de Angular" aprenderás a trabajar con formularios y realizar validaciones en Angular.

Creación de formularios

Para comenzar, es necesario comprender cómo crear formularios en Angular. Los formularios son elementos fundamentales para recopilar información de los usuarios. En Angular, puedes crear formularios utilizando la sintaxis HTML tradicional o mediante la forma reactiva.

En los formularios basados en HTML, se utiliza la directiva `ngForm` para enlazar el formulario con una instancia de Angular. Además, se pueden utilizar otras directivas como `ngModel`, `ngSubmit` y `ngControl` para establecer enlaces bidireccionales de datos y capturar eventos del formulario.

Por otro lado, los formularios reactivos proporcionan un enfoque más programático para trabajar con formularios en Angular. En lugar de utilizar directivas HTML, puedes crear formularios utilizando clases TypeScript y funciones de Angular. Esto te da más control sobre la lógica del formulario y simplifica la validación de datos.

Enlace bidireccional de datos

Uno de los aspectos más importantes al trabajar con formularios es el enlace bidireccional de datos. En Angular, puedes establecer este enlace utilizando la directiva `ngModel`, que permite sincronizar los datos del formulario con una propiedad en el componente.

Al establecer el enlace bidireccional de datos, cualquier cambio realizado en los controles del formulario se reflejará automáticamente en el componente, y viceversa. Esto simplifica la gestión de los datos y proporciona una experiencia de usuario más fluida.

Validación de formularios

La validación de formularios es un paso fundamental para garantizar que los datos introducidos por el usuario sean correctos y cumplan con ciertos criterios. En Angular, puedes realizar la validación utilizando diferentes métodos.

Por un lado, puedes realizar la validación en el lado del cliente utilizando las validaciones incorporadas de Angular, como `required`, `minLength`, `maxLength`, `pattern`, entre otras. Estas validaciones se pueden aplicar directamente a los controles del formulario y proporcionan una experiencia de usuario más intuitiva al mostrar mensajes de error en tiempo real.

Por otro lado, también puedes realizar la validación en el lado del servidor, utilizando servicios y APIs RESTful. En este caso, es posible enviar el formulario al servidor y recibir una respuesta que indique si los datos introducidos son válidos o no.

Conclusiones

En esta sección del curso "Curso de Angular" has aprendido a crear formularios y realizar validaciones en Angular. Ahora eres capaz de establecer enlaces bidireccionales de datos, utilizar las validaciones incorporadas de Angular y realizar validaciones en el lado del servidor. Esta habilidad te permitirá crear formularios interactivos y asegurar la integridad de los datos en tus aplicaciones Angular.

¡Continúa con el curso y sigue aprendiendo sobre las demás características y funcionalidades de Angular!

Nota: Esta sección del curso no incluye la creación de plantillas de formularios personalizadas, pero esta es una habilidad avanzada que puedes explorar por tu cuenta.

Conclusión - Formularios y Validación

Formularios y Validación En este módulo, hemos aprendido cómo crear formularios en Angular y cómo realizar enlaces bidireccionales de datos. También exploramos cómo validar los formularios y mostrar mensajes de error. Con este conocimiento, podrás crear formularios interactivos y validar la entrada del usuario en tus aplicaciones Angular.

Comunicación con el Servidor

07 | Comunicación con el Servidor

En Angular, la comunicación con el servidor es esencial para interactuar con APIs RESTful y recuperar o enviar datos desde nuestra aplicación. Para realizar esta

comunicación de manera eficiente, Angular ofrece un conjunto de herramientas y funcionalidades que simplifican el proceso.

Integración con APIs RESTful

Al desarrollar aplicaciones web, a menudo necesitamos consumir servicios web o APIs RESTful para obtener o enviar datos. Angular proporciona el módulo HttpClient, que se utiliza para realizar solicitudes HTTP y manejar las respuestas.

La integración con APIs RESTful implica realizar solicitudes HTTP como GET, POST, PUT y DELETE para recuperar, enviar, actualizar o eliminar datos. El módulo HttpClient proporciona métodos para realizar estas solicitudes de manera sencilla, permitiendo especificar la URL del recurso, los parámetros y los encabezados necesarios.

Uso de módulo HttpClient

El módulo HttpClient de Angular es una poderosa herramienta para realizar solicitudes HTTP de manera eficiente. Para utilizarlo, primero debemos importar el módulo HttpClient en nuestro componente o servicio:

```
import { HttpClient } from '@angular/common/http';
```

Luego, podemos inyectar la instancia de HttpClient en nuestro constructor:

```
constructor(private http: HttpClient) { }
```

Una vez que tenemos acceso al objeto HttpClient, podemos utilizar sus métodos para realizar solicitudes HTTP:

```
 this.http.get('https://api.example.com/data').subscribe(response => {
  // Manejar la respuesta del servidor
}, error => {
  // Manejar el error en caso de que la solicitud falle
});
```

En este ejemplo, estamos utilizando el método `get` para realizar una solicitud GET a la URL especificada. Podemos suscribirnos al Observable devuelto por este método para recibir la respuesta del servidor y manejarla en consecuencia.

Manejo de solicitudes y respuestas

Al realizar una solicitud HTTP, es importante manejar tanto la respuesta exitosa del servidor como los posibles errores. Angular nos proporciona varias opciones para manejar estas situaciones.

En el caso de una respuesta exitosa, podemos utilizar el método `subscribe` para suscribirnos al Observable devuelto por la solicitud y manejar la respuesta:

```
 this.http.get('https://api.example.com/data').subscribe(response => {
  // Manejar la respuesta del servidor
});
```

En el caso de un error, podemos utilizar el segundo parámetro del método `subscribe` para manejar el error:

```
 this.http.get('https://api.example.com/data').subscribe(response => {
  // Manejar la respuesta del servidor
```

```
}, error => {
  // Manejar el error en caso de que la solicitud falle
});
```

Podemos realizar distintas acciones en el caso de un error, como mostrar un mensaje de error al usuario, volver a intentar la solicitud o redirigir a una página de error.

Conclusion

La comunicación con el servidor es una parte fundamental en el desarrollo de aplicaciones web. Angular proporciona el módulo HttpClient para facilitar la integración con APIs RESTful y realizar solicitudes HTTP de manera eficiente. Al utilizar el módulo HttpClient y manejar de forma adecuada las respuestas del servidor, podemos crear aplicaciones web dinámicas y responsivas que interactúen de forma eficiente con los servicios web.

Conclusión - Comunicación con el Servidor

Comunicación con el Servidor En este módulo, hemos explorado cómo integrar Angular con APIs RESTful utilizando el módulo HttpClient. Aprendimos cómo manejar las solicitudes y respuestas del servidor de manera eficiente. Con este conocimiento, podrás conectarte a servicios web y obtener y

enviar datos desde y hacia tu aplicación Angular de manera fácil y segura.

Pipes

08 | Pipes

Los *pipes* en Angular son una característica poderosa que nos permite transformar y formatear los datos en nuestros templates de manera sencilla y eficiente. Un *pipe* es básicamente una función que toma una entrada y la transforma en una salida modificada según una serie de reglas predefinidas o personalizadas.

Pipes incorporados

Angular nos ofrece una amplia gama de *pipes* incorporados que podemos utilizar de forma rápida y fácil en nuestros proyectos. Algunos ejemplos de *pipes* incorporados son:

- **Date**: nos permite formatear y mostrar fechas de diferentes maneras, como "dd/MM/yyyy" o "HH:mm:ss".
- **Currency**: nos permite formatear valores monetarios y mostrarlos en diferentes monedas y formatos.
- **UpperCase y LowerCase**: nos permiten convertir un texto a mayúsculas o minúsculas respectivamente.
- **Decimal**: nos permite formatear números decimales y ajustar la cantidad de dígitos decimales que se mostrarán.

Estos son solo algunos ejemplos de *pipes* incorporados, pero hay muchos más disponibles en Angular.

Creación de pipes personalizados

Además de los *pipes* incorporados, también podemos crear nuestros propios *pipes* personalizados para adaptarlos a las necesidades específicas de nuestros proyectos. Para crear un *pipe* personalizado, debemos seguir los siguientes pasos:

1. Crear una clase TypeScript que implemente la interfaz `PipeTransform`.
2. Definir un método `transform()` en la clase, que tomará una entrada y devolverá la salida transformada.
3. Decorar la clase con `@Pipe` y proporcionar un nombre para nuestro *pipe*.

Una vez que hayamos creado nuestro *pipe* personalizado, podremos utilizarlo en nuestros templates de la misma manera que utilizamos los *pipes* incorporados.

Formateo y transformación de datos en el template

El uso de *pipes* en nuestros templates nos permite realizar formateo y transformación de datos de manera muy conveniente. Para utilizar un *pipe* en un template, simplemente agregamos el operador `|` seguido del nombre del *pipe* y, opcionalmente, algunos parámetros en caso de que el *pipe* los requiera.

Por ejemplo, si queremos mostrar una fecha en un formato específico, podemos utilizar el *pipe* `date` de la siguiente manera:

```
<p>La fecha actual es: {{ currentDate | date: 'dd/MM/yyyy' }}</p>
```

En este ejemplo, `currentDate` es una variable en el componente que contiene la fecha actual y el *pipe* `date` se encarga de formatearla como "dd/MM/yyyy".

Conclusión

En resumen, los *pipes* en Angular nos permiten transformar y formatear los datos en nuestros templates de manera sencilla y eficiente. Podemos aprovechar los *pipes* incorporados o crear nuestros propios *pipes* personalizados para adaptarlos a las necesidades específicas de nuestros proyectos. Con el uso de *pipes*, podemos mejorar la legibilidad y funcionalidad de nuestras aplicaciones Angular.

Conclusión - Pipes

Pipes En este módulo, hemos aprendido cómo utilizar pipes incorporados en Angular, como Date y Currency, para formatear y transformar datos en el template. También

exploramos cómo crear pipes personalizados para aplicar transformaciones específicas. Con este conocimiento, podrás mostrar y manipular datos de manera efectiva en tus aplicaciones Angular.

Manejo de Estados con RxJS

En el desarrollo de aplicaciones con Angular, es muy común enfrentarse a la necesidad de gestionar y mantener el estado de la aplicación. El manejo eficiente del estado es fundamental para lograr una aplicación robusta, escalable y que ofrezca una experiencia de usuario fluida. Para abordar este desafío, Angular proporciona una poderosa biblioteca llamada RxJS (Reactive Extensions for JavaScript).

Introducción a Reactive Extensions (RxJS)

RxJS es una implementación de la programación reactiva basada en la especificación Observable. La programación reactiva se basa en el flujo de datos asíncronos y eventos, abstrayendo estos flujos como secuencias observables. Esta abstracción permite manipular y transformar los datos de una manera declarativa y funcional.

En el contexto de Angular, RxJS se utiliza para manejar el estado de la aplicación de una manera reactiva y eficiente. Permite gestionar eventos, realizar peticiones asincrónicas, manejar flujos de datos y realizar transformaciones sobre ellos de forma sencilla y elegante.

Uso de observables y suscripciones

El núcleo de RxJS es el concepto de observables y suscripciones. Un observable es una secuencia de valores o eventos que se producen en el tiempo. Puede representar eventos de teclado, llamadas API, cambios en el estado de la aplicación, entre otros. Por otro lado, una suscripción permite recibir y reaccionar a los valores emitidos por un observable.

Para utilizar RxJS en Angular, es necesario importar sus operadores y funciones necesarios. Estos operadores se utilizan para aplicar transformaciones y filtros a los observables, lo que permite manipular los datos que fluyen a través de ellos.

Gestión del estado de la aplicación

El manejo del estado de la aplicación con RxJS implica crear observables que representen el estado de diferentes partes de la aplicación y suscribirse a ellos para

recibir los cambios en el estado. Esto nos permite reaccionar y actualizar la interfaz de usuario de manera automática y eficiente.

Además, RxJS ofrece operadores como `map`, `filter`, `reduce` y `merge`, que facilitan la transformación y combinación de los datos emitidos por los observables. Estos operadores son muy útiles para realizar lógica de negocio y mantener el estado de la aplicación actualizado en todo momento.

En resumen, RxJS nos brinda las herramientas necesarias para gestionar de manera efectiva el estado de nuestra aplicación Angular. Su enfoque reactivo y las numerosas funciones y operadores disponibles nos permiten crear aplicaciones más robustas, escalables y fáciles de mantener. Domina RxJS y mejora tu habilidad para desarrollar aplicaciones profesionales con Angular.

Conclusión - Manejo de Estados con RxJS

Manejo de Estados con RxJS En este módulo, hemos introducido Reactive Extensions (RxJS) y explorado cómo utilizar observables y suscripciones para gestionar el estado de nuestra aplicación Angular. Aprendimos cómo manejar eventos y reaccionar a cambios de estado de manera efectiva. Con este conocimiento, podrás crear aplicaciones reactivas y mantener un control completo del estado de tu aplicación.

Ejercicios Practicos

Pongamos en práctica tus conocimientos

10 | Ejercicios Practicos

En esta lección, pondremos la teoría en práctica a través de actividades prácticas. Haga clic en los elementos a continuación para verificar cada ejercicio y desarrollar habilidades prácticas que lo ayudarán a tener éxito en el tema.

Introducción a Angular

En este ejercicio, aprenderás los conceptos básicos de Angular y su arquitectura. Crearás un proyecto nuevo y configurarás el entorno de desarrollo. Además, explorarás las características principales de Angular.

Creación de componentes

En este ejercicio, aprenderás a crear componentes en Angular. Crearás un componente nuevo y establecerás la estructura básica de un componente. También aprenderás a enlazar datos y eventos en el template del componente.

Uso de directivas incorporadas

En este ejercicio, explorarás las directivas incorporadas de Angular, como ngIf y ngFor. Aprenderás a utilizar estas directivas en tu proyecto para manipular el DOM y controlar la visualización de elementos en función de condiciones o iteraciones.

Creación y uso de servicios

En este ejercicio, aprenderás a crear servicios en Angular y a utilizarlos en tus componentes. Aprenderás sobre la inyección de dependencias y cómo comunicar componentes usando servicios.

Configuración de rutas

En este ejercicio, aprenderás a configurar las rutas en tu aplicación Angular. Aprenderás a navegar entre diferentes páginas y a utilizar parámetros de ruta y rutas anidadas.

Creación de formularios

En este ejercicio, aprenderás a crear formularios en Angular. Aprenderás a vincular los datos de los formularios con los componentes y a validar los datos ingresados por el usuario.

Integración con APIs RESTful

En este ejercicio, aprenderás a comunicarte con un servidor mediante APIs RESTful en Angular. Utilizarás el módulo HttpClient para realizar solicitudes HTTP y manejar las respuestas del servidor.

Uso de pipes incorporados

En este ejercicio, aprenderás a utilizar los pipes incorporados de Angular, como el pipe Date y el pipe Currency. Aprenderás a formatear y transformar los datos en el template de tu aplicación.

Uso de observables y suscripciones

En este ejercicio, aprenderás a utilizar Reactive Extensions (RxJS) para manejar el estado de tu aplicación en Angular. Aprenderás a utilizar observables y suscripciones para gestionar el flujo de datos y sincronizar componentes.

Resumen

Repasemos lo que acabamos de ver hasta ahora

11 | Resumen

- ✓ ¿Qué es Angular? En este curso, hemos explorado a fondo qué es Angular y cómo se utiliza para construir aplicaciones web robustas y escalables. Aprendimos sobre la arquitectura y las características principales de Angular, así

como la instalación y configuración del entorno de desarrollo. Ahora tienes una base sólida para empezar a utilizar Angular en tus proyectos.

- ✓ Componentes y Templates En este módulo, hemos aprendido cómo crear componentes en Angular y cómo utilizarlos en nuestras aplicaciones. También hemos explorado la estructura de un componente y cómo enlazar datos y eventos en el template. Con esta base, podrás crear componentes reutilizables y dinámicos en tus proyectos con Angular.
- ✓ Directivas En este módulo, hemos explorado las directivas en Angular, tanto las incorporadas como las personalizadas. Aprendimos cómo utilizar directivas como ngIf y ngFor para manipular el DOM y controlar la visualización de elementos en nuestra aplicación. Con este conocimiento, podrás mejorar la interactividad y la flexibilidad de tus aplicaciones Angular.
- ✓ Servicios y Dependencia En este módulo, hemos aprendido cómo crear y utilizar servicios en Angular, así como la importancia de la inyección de dependencias. También exploramos cómo comunicarse entre componentes utilizando servicios. Con este conocimiento, podrás separar la lógica de negocio de tus componentes y crear aplicaciones más modulares y mantenibles.
- ✓ Enrutamiento en Angular En este módulo, hemos aprendido cómo configurar las rutas en Angular y cómo navegar entre páginas. También exploramos cómo utilizar parámetros de ruta y rutas anidadas para crear una navegación más dinámica. Con este conocimiento, podrás crear aplicaciones de una sola página (SPA) con enrutamiento efectivo en Angular.
- ✓ Formularios y Validación En este módulo, hemos aprendido cómo crear formularios en Angular y cómo realizar enlaces bidireccionales de datos. También exploramos cómo validar los formularios y mostrar mensajes de error.

Con este conocimiento, podrás crear formularios interactivos y validar la entrada del usuario en tus aplicaciones Angular.

- ✓ Comunicación con el Servidor En este módulo, hemos explorado cómo integrar Angular con APIs RESTful utilizando el módulo HttpClient. Aprendimos cómo manejar las solicitudes y respuestas del servidor de manera eficiente. Con este conocimiento, podrás conectarte a servicios web y obtener y enviar datos desde y hacia tu aplicación Angular de manera fácil y segura.
- ✓ Pipes En este módulo, hemos aprendido cómo utilizar pipes incorporados en Angular, como Date y Currency, para formatear y transformar datos en el template. También exploramos cómo crear pipes personalizados para aplicar transformaciones específicas. Con este conocimiento, podrás mostrar y manipular datos de manera efectiva en tus aplicaciones Angular.
- ✓ Manejo de Estados con RxJS En este módulo, hemos introducido Reactive Extensions (RxJS) y explorado cómo utilizar observables y suscripciones para gestionar el estado de nuestra aplicación Angular. Aprendimos cómo manejar eventos y reaccionar a cambios de estado de manera efectiva. Con este conocimiento, podrás crear aplicaciones reactivas y mantener un control completo del estado de tu aplicación.

Prueba

Comprueba tus conocimientos respondiendo unas preguntas

12 | Prueba

1. ¿Qué es Angular?

- ◯ Un lenguaje de programación
- ◯ Una base de datos
- ◯ Un framework de desarrollo web

2. ¿Cuál es la función principal de los componentes en Angular?

- ◯ Crear la interfaz de usuario
- ◯ Gestionar el enrutamiento
- ◯ Manipular la base de datos

3. ¿Cuál de las siguientes directivas incorporadas se utiliza para mostrar u ocultar elementos en base a una expresión booleana?

- ◯ ngSwitch
- ◯ ngFor
- ◯ ngIf

---

4. ¿Qué es un servicio en Angular?

- ◯ Una función que realiza una tarea específica
- ◯ Una clase que se utiliza para compartir datos y funcionalidades entre componentes
- ◯ Un objeto que se utiliza para almacenar datos

---

5. ¿Cómo se define una ruta en Angular?

- ◯ En el archivo .ts del componente
- ◯ En el archivo de configuración de las rutas
- ◯ En el archivo .html del componente

---

6. ¿Cuál es el propósito principal de los formularios en Angular?

- ◯ Realizar peticiones HTTP
- ◯ Crear componentes
- ◯ Validar datos

---

7. ¿Cuál es el módulo de Angular utilizado para realizar solicitudes HTTP?

- ◯ RouterModule
- ◯ FormsModule
- ◯ HttpClient

8. ¿Cuál de las siguientes afirmaciones es cierta sobre los pipes en Angular?

- ◯ Se utilizan para crear animaciones
- ◯ Se utilizan para formatear y transformar datos
- ◯ Se utilizan para manejar eventos del DOM

9. ¿Qué son los Reactive Extensions (RxJS) en Angular?

- ◯ Una librería para generar gráficos
- ◯ Un método de encriptación de datos
- ◯ Una librería para manejar la programación reactiva

10. ¿Cuál de las siguientes afirmaciones es cierta sobre los observables en Angular?

- ◯ Son utilizados para recorrer listas de elementos
- ◯ Son utilizados para trabajar con flujos de datos asíncronos
- ◯ Son utilizados para almacenar datos en memoria

Entregar

Conclusión

Felicidades!

¡Felicitaciones por completar este curso! Has dado un paso importante para desbloquear todo tu potencial. Completar este curso no se trata solo de adquirir conocimientos; se trata de poner ese conocimiento en práctica y tener un impacto positivo en el mundo que te rodea.

Comparte este curso