Angular 6

Libro de Laboratorios

Polo Tecnologico
http://www.diamand.com.ar

November 2, 2018

Acerca de este documento

Actualizaciones de este documento se pueden encontrar en http://dev.diamand.com.ar/doc/training/angular/.

Este documento se genero a partir de fuentes en LaTeX en http://git.free-electrons.com/training-materials.

Mas detalles de nuestras sesiones se pueden encontrar en http://free-electrons.com/training.

Copying this document

Corrections, suggestions, contributions and translations are welcome!

TypeScript - Tipos básicos

Objetivos: Anotar tipos primitivos, arreglos y cualquier tipo. Identificar cuando ocurre la verificación de tipos. Luego analizaremos la salida del transpiler

Ejercicio 1.1

Dado el siguiente codigo:

1. Coloca el cursor sobre los garabatos rojos para inspeccionar los errores de TS
2. Coloca el cursor sobre las variables para inspeccionar sus tipos
3. Arregle el error en la linea 2 cambiando el valor de pi al tipo esperado

```
let pi = '3.14159';
let tau = pi * 2;

console.log('[Ejercicio 1.1]', `${tau} es ${pi} veces el dos.`);
```

Ejercicio 1.2

Dado el siguiente codigo:

1. Inspeccionar el tipo de ‘torta‘
2. Añadir una anotación de tipo explícito a ‘torta‘
3. Intenta asignar tipos inválidos, por diversión

```
let torta;
torta = 'arandanos';

console.log('[Ejercicio 1.2]', `Me gusta comer torta con sabor a ${torta}.`);
```

Ejercicio 1.3

Dado el siguiente codigo:

1. Inspeccione el error, luego corríjalo

```
let esPablo: boolean;

console.log('[Ejercicio 1.3]', `${esPablo ? 'Oh, hola Pablo' : 'Quien sos vos?'}`);
```


Ejercicio 1.4

Dado el siguiente codigo:

1 Añadir anotaciones de tipo (lo más explícitas posible)

2 Solucionar errores (si corresponde)

```
const entero = 6;
const decimal = 6.66;
const hexadecimal = 0xf00d;
const binario = 0b1010011010;
const octal = 0o744;
const ceroNegativo = -0;
const enRealiadadNumero = NaN;
const mayorNumero = Number.MAX_VALUE;
const elNumeroMasGrande = Infinity;

const miembros: any[] = [
  entero,
  decimal,
  hexadecimal,
  binario,
  octal,
  ceroNegativo,
  enRealiadadNumero,
  mayorNumero,
  elNumeroMasGrande
];

miembros[0] = '12345';

console.log('[Ejercicio 1.4]', miembros);
```

Ejercicio 1.5

Dado el siguiente codigo:

1 Añadir anotaciones de tipo (lo más explícitas posible)

2 Solucionar errores (si corresponde)

```
const secuencia = Array.from(Array(10).keys());
const animales = ['pinguino', 'oso hormiguero', 'zorro', 'jirafa'];
const cadenasYNumeros = [1, 'uno', 2, 'dos', 3, 'tres'];
const todosMisArreglos = [secuencia, animales, cadenasYNumeros];

console.log('Ejercicio 1.5', todosMisArreglos);
```


Ejercicio 1.6

Queremos representar un elemento de inventario como una estructura donde la primera entrada es el nombre del artículo y la segunda es la cantidad.

Dado el siguiente codigo:

1. Añadir anotaciones de tipo (lo más explícitas posible)
2. Solucionar errores (si corresponde)

```
const elementoInventario = ['tuerca', 11];

// despues lo desestructuramos
const [nombre, cantidad] = elementoInventario;

const mensaje = agregarInventario(nombre, cantidad);

console.log('[Ejercicio 1.6]', mensaje);

function agregarInventario(nombre: string, cantidad: number): string {
  return `Se agregaron ${cantidad} ${nombre}s al inventario.`;
}
```


TypeScript - Interfaces

Objetivos: Demostrar la tipificación estructural (duck typing), Crear una interfaz e implementarla en una clase, Diferenciar los alias de tipo de las interfaces.

Ejercicio 2.1

Dado el siguiente codigo:

```
function agregarAlCarro(item: { id: number, titulo: string, idVariante: number }) {
  console.log('[Ejercicio 2.1]', `Agregando "${item.titulo}" al carro de compras.`);
}

agregarAlCarro({ id: 1, titulo: 'Zapatos de cuero' });
```

1 Crea una interfaz 'CartItem' y reemplaza el tipo de parametros con ella

2 Hacer idVariante opcional

Ejercicio 2.2

Dado el siguiente codigo:

```
class Persona {
  constructor(public nombre: string, public edad: number) { }
}

const juan = new Persona('Juan', 31);

console.log('[Ejercicio 2.2]', `${juan.nombre} tiene ${juan.edad} anios.`);
```

1 Cree e implemente una interfaz en 'Persona' para asegurarse de que siempre tenga acceso a las propiedades miembros 'nombre' y 'edad'

Ejercicio 2.3

Dado el siguiente codigo:

```
// [no editar] (pretender que esto proviene de una version externa de la
// biblioteca `foo.d.ts`)
interface Ciudad {
  nombre: string;
}
// [/no editar]

const montreal = {
  coords: {
```



```
    latitud: 42.332,
    longitud: -73.324,
  },
  nombre: 'Montreal',
};

const tampa = {
  coords: {
    latitud: '27.9478',
    longitud: '-82.4584',
  },
  nombre: 'Tampa',
};

function informacionCiudad(ciudad: Ciudad) {
  const coords =
    `(${ciudad.coords.latitud.toFixed(3)}, ${ciudad.coords.longitud.toFixed(3)})`;
  return `${ciudad.nombre.toUpperCase()} se encuentra en ${coords}.`;
}

console.log('[Ejercicio 2.3]',
  `${informacionCiudad(montreal)} \n\n ${informacionCiudad(tampa)}`);
```

1. Cree una interfaz 'Coords' que tenga las propiedades numéricas 'latitud' y 'longitud'
2. Extienda la interfaz existente 'Ciudad' (sin modificarla en línea) agregando una propiedad 'coords' de tipo 'Coords'
3. Corregir lo que está mal con 'tampa'

Ejercicio 2.4

El propósito de este ejercicio es simplemente ilustrar el uso de 'readonly':

```
interface EsquemaUsuario {
  readonly id: number;
  nombre: string;
}

class Usuario implements EsquemaUsuario {
  constructor(public nombre: string, readonly id: number) { }
}

const usuario = new Usuario('Perro', 1);

console.log(usuario.id); // legible

usuario.nombre = 'Harold'; // asignable
usuario.id = 5; // no asignable

console.log('[Ejercicio 2.4]', `Usuario:`, usuario)
```


TypeScript - Functiones

Objetivos: Convertir las funciones existentes de JavaScript a TypeScript, Entender las funciones como tipos, Convierte funciones específicamente tipificadas a funciones más flexibles genéricas

Ejercicio 3.1

Dado el siguiente codigo:

```
function add(x, y) {
  return x + y;
}

function sumArray(numbers) {
  return numbers.reduce(add, 0);
}

const someSum = sumArray(['3', '6', '9']);

console.log('[Ejercicio 3.1]', `3 + 6 + 9 === ${someSum}`);
```

1 Agregue tipos explícitos a los parámetros y el tipo de retorno

2 Solucione cualquier error resultante de tipos inválidos

Ejercicio 3.2

Dado el siguiente codigo:

```
const bankAccount = {
  money: 0,
  deposit(value, message) {
    this.money += value;
    if (message) {
      console.log(message);
    }
  }
};

bankAccount.deposit(20);
bankAccount.deposit(10, 'Deposit received')

console.log('[Exercise 3.2]', `Account value: $${bankAccount.money}`);
```

1 Agregue tipos explícitos a los parámetros y el tipo de retorno a la funcion ‘deposit‘

2 Haz que el parámetro de ‘message‘ sea *optional*

Ejercicio 3.3

Para una palabra dada, calculamos su puntuación en Scrabble®

```
function computeScore(word) {
  const letters = word.toUpperCase().split('');
  return letters.reduce((accum, curr) => accum += getPointsFor(curr), 0);
}

function getPointsFor(letter) {
  const lettersAndPoints = [
    ['AEOIULNRST', 1],
    ['DG', 2],
    ['BCMP', 3],
    ['FHVWY', 4],
    ['K', 5],
    ['JX', 8],
    ['QZ', 10],
  ];

  return lettersAndPoints.reduce((computedScore, pointsTuple) => {
    const [letters, score] = pointsTuple;
    if (letters.split('').find((ll) => ll === letter)) {
      return computedScore += score;
    }
    return computedScore;
  }, 0);
}

console.log('[Ejercicio 3.3]', `zoologico vale ${computeScore('zoo')} puntos.`);
```

1 Añadir anotaciones de tipo siempre que sea posible

Ejercicio 3.4

Dado el siguiente codigo:

```
function greet(greeting) {
  return greeting.toUpperCase();
}

const defaultGreeting = greet();
const portugueseGreeting = greet('Oi como vai!');

console.log('[Ejercicio 3.4]', defaultGreeting, portugueseGreeting);
```

1 Añadir tipos explícitos a los parámetros y tipo de retorno

2 Añadir un saludo predeterminado: ”hola”

Ejercicio 3.5

```
function layEggs(quantity, color) {
```



```
  console.log(
    `[Ejercicio 3.5] Acabas de poner ${quantity} huevos ${color}. Buen trabajo!`);
}

layEggs();
```

1 Añadir anotación de tipo de parámetro

2 A pesar de que esta función no vuelve, agregue un tipo de retorno explícito

Ejercicio 3.6

Aquí hemos inicializado dos variables con tipos de funciones. Posteriormente les asignamos funciones.

```
let multiply: (val1: number, val2: number) => number;
let capitalize: (val: string) => string;

multiply = function (value: string): string {
  return `${value.charAt(0).toUpperCase()}${value.slice(1)}`;
}

capitalize = function (x: number, y: number): number {
  return x * y;
}

console.log('[Ejercicio 3.6]', capitalize(`habil ${multiply(5, 10)}`));
```

1 Arreglar los errores

Ejercicio 3.7

Actualmente, nuestra función 'pushToCollection' acepta *cualquier* elemento y lo agrega, (indiscriminadamente) a *cualquier* tipo de matriz.

Un par de problemas con esto:

1 El tipo 'any' hace que perdamos toda la información de tipos en nuestros parámetros.

2 Esta holgura se ha vuelto en nuestra contra durante el tiempo de ejecución (mira a 'incrementByTwo')

Dado el siguiente codigo:

```
const numberCollection: number[] = [];
const stringCollection: string[] = [];

function pushToCollection(item, collection) {
  collection.push(item);
  return collection;
}

// Anadir algunas cosas a las colecciones
pushToCollection(false, stringCollection);
pushToCollection('hi', stringCollection);
```



```
12 pushToCollection([], stringCollection);
13
14 pushToCollection('1', numberCollection);
15 pushToCollection('2', numberCollection);
16 pushToCollection('3', numberCollection);
17
18 const incrementedByTwo = numberCollection.map((num) => num + 2);
19
20 console.log('[Ejercicio 3.7]', `[${incrementedByTwo}] debe ser igual a [3,4,5]`);
```

1 Implementar 'pushToCollection' como una función genérica. (Esto debería crear errores en tiempo de compilación en lugares donde se agregan valores incorrectos a una colección determinada. Fije estos valores a los tipos correctos)

2 Una vez hecho genérico, 'pushToCollection' debe ser suficientemente *generic* para operar en artículos y colecciones de cualquier tipo mientras se continúa aplicando que conicidan

TypeScript - Tipos

Objetivos: Comprender cómo TypeScript realiza análisis de flujo de código, Crear y aplicar tipos de unión e intersección, Utilice protecciones de tipo básico (tipos de estrechamiento con: typeof, instanceof, etc.)

Ejercicio 4.0

TypeScript es inteligente sobre los posibles tipos de una variable, dependiendo de la ruta del código.

Dado el siguiente codigo:

```
function doStuff(value: any): void {
  if (typeof value === 'string') {
    console.log(value.toUpperCase().split('').join(' '));
  } else if (typeof value === 'number') {
    console.log(value.toPrecision(5));
  }

  value; // coloque el cursor sobre `valor` aqui
}

doStuff(2);
doStuff(22);
doStuff(222);
doStuff('hello');
doStuff(true);
doStuff({});

console.log('[Ejercicio 4.1]');
```

1 Simplemente inspeccione los tipos posibles moviéndose sobre el 'texto' para ver cómo cambia el tipo inferido si se pueden hacer suposiciones de forma segura sobre los tipos posibles dentro de la ruta del código dado

Ejercicio 4.1

Dado el siguiente codigo:

```
interface EggLayer {
  layEggs(): void;
}

interface Flyer {
  fly(height: number): void;
}

```



```
interface Swimmer {
  swim(depth: number): void;
}

// agregar alias de tipo(s) aqui

class Bird implements BirdLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de aves.');
  }

  fly(height: number): void {
    console.log(`Volando a la altura de ${height} metros.`);
  }
};

class Fish implements FishLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de pescado.');
  }

  swim(depth: number): void {
    console.log(`Nadando a una profundidad de ${depth} metros.`);
  }
}

function getRandomAnimal() {
  const animals = [
    new Bird('puffin'),
    new Bird('kittiwake'),
    new Fish('sea robin'),
    new Fish('leafy seadragon'),
  ];

  return animals[Math.floor(Math.random() * animals.length)];
}

function interrogateAnimal(animal = getRandomAnimal()) {
  animal.swim(10) // se llama solo si es un pez
  animal.fly(10); // se llama solo si es un pajaro

  return animal.species;
}

console.log('[Ejercicio 4.4]',
  `Tenemos un ${interrogateAnimal()} en nuestras manos!`);
```

1 Restrinja el tipo de 'valor' a 'string o number'

2 Solucione cualquier error resultante

Ejercicio 4.2

Dado el siguiente codigo:

```
1
2 function padLeft(value: string, padding: number | string): string {
3   // si padding es un numero, return `${Array(padding + 1).join(' ')}${value}`
4   // si padding es una cadena, return padding + value
5 }
6
7 console.log('[Ejercicio 4.2]', `
8       ${padLeft('', 0)}
9       ${padLeft('', '')}
10      ${padLeft('', '')}
11      ${padLeft('', '')}
12      ${padLeft('', '')}
13    `);
```

1 Use un protector de tipo para completar el cuerpo de la función 'padLeft'

Ejercicio 4.3

Dado el siguiente codigo:

```
1 const numbers = [1, 2, 3, [44, 55], 6, [77, 88], 9, 10];
2
3 function flatten(array) {
4   const flattened = [];
5
6   for (const element of array) {
7     if (Array.isArray(element)) {
8       element; // any[]
9       flattened.push(...element);
10    } else {
11      element; // any
12      flattened.push(element);
13    }
14  }
15
16  return flattened;
17 }
18
19 const flattenedNumbers = flatten(numbers);
20
21 console.log('[Ejercicio 4.3]', flattenedNumbers);
```

1 Añadir anotaciones de tipo ('any' excluido)

2 Inspeccione el tipo de 'element' en diferentes ramas de código

3 Bonificación: convertir 'flatten' en una función genérica

Ejercicio 4.4

Dado el siguiente codigo:

```
interface EggLayer {
  layEggs(): void;
}

interface Flyer {
  fly(height: number): void;
}

interface Swimmer {
  swim(depth: number): void;
}

// agregar alias de tipo(s) aqui

class Bird implements BirdLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de aves.');
  }

  fly(height: number): void {
    console.log(`Volando a la altura de ${height} metros.`);
  }
};

class Fish implements FishLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de pescado.');
  }

  swim(depth: number): void {
    console.log(`Nadando a una profundidad de ${depth} metros.`);
  }
}

function getRandomAnimal() {
  const animals = [
    new Bird('puffin'),
    new Bird('kittiwake'),
    new Fish('sea robin'),
    new Fish('leafy seadragon'),
  ];

  return animals[Math.floor(Math.random() * animals.length)];
}

```



```
function interrogateAnimal(animal = getRandomAnimal()) {
  animal.swim(10) // se llama solo si es un pez
  animal.fly(10); // se llama solo si es un pajaro

  return animal.species;
}

console.log('[Ejercicio 4.4]',
  `Tenemos un ${interrogateAnimal()} en nuestras manos!`);
```

1 Las aves y los peces ponen huevos. Sólo los pájaros vuelan. Sólo los peces nadan. Defina dos tipos nuevos: 'BirdLike' y 'FishLike' basados en estos rasgos

2 Crea un alias de tipo para 'Bird OR Fish' llamado 'Animal'

3 Use 'instanceof' en 'interrogateAnimal' para permitir a los peces nadar y a los pájaros volar

4 Agregue cualquier anotación de tipo faltante, siendo lo más explícito posible

Introduccion - Comenzando

Los codigos de este laboratorio se encuentran en:

https://github.com/ldiamand/curso_angular.git

Arranque del componente principal

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio02` del repositorio.

Cada aplicacion en Angular se inicia con un componente de aplicacion y un modulo raiz. El componente de aplicacion `(app/app.component.ts)` y el modulo raiz `(app/app.module.ts)` se crearon, pero la apilcacion no arranca. Haga los cambios necesarios para que la aplicacion arranque. Para lograr esto, va a necesitar:

- Declarar el componente de aplicacion en el modulo raiz `(app/app.module.ts)`
- Agregar el componente de aplicacion como el componente de arranque en el modulo `(app/app.module.ts)`
- Utilizar el servicio `platformBrowserDynamic` para arrancar el modulo en el ejecutable principal `(app/main.ts)`

Creando y Comunicando Componentes entre si

Los codigos de este laboratorio se encuentran en: https://github.com/ldiamand/curso_angular.git

Creación de un componente enlazado a datos

Cree un proyecto nuevo utilizando Angular CLI. Agregue un nuevo componente para mostrar una lista de los próximos eventos usando el HTML y los datos a continuación. Luego, cargue ese componente desde la plantilla en línea del componente principal de la aplicación (app/app.component.ts). Para hacer esto, necesitarás:

- Crear el componente
- Puede utilizar una plantilla en línea o un archivo de plantilla separado. El codigo HTML sin enlaces de datos para la plantilla se encuentran a continuación
- Añadir una propiedad en el componente para contener los datos
- Los datos para el componente también están abajo
- Agregue el código de enlace de datos necesario mediante la interpolación a la plantilla del componente
- Verifique que Angular CLI agrego el componente al módulo de aplicación (app/app.module.ts)
- Cargue el componente desde la plantilla del componente de aplicación (app/app.component.html) (Como seria la variacion para hacerlo usando la plantilla en linea?)

Codigo HTML:

```
<!-- Aqui esta el HTML de inicio para la plantilla -->
<div>
  <h1>Felicitaciones!</h1>
  <h4>Has conseguido que tu componente se muestre!</h4>
  <hr />
  <h5>Como se ve el evento?</h5>

  <div style="margin-top:30px">
    Evento:
  </div>
  <div>
    Fecha:
  </div>
  <div>
    Hora:
  </div>
  <div>
```



```
    Direccion:
  </div>
 </div>
```

Datos:

```
{
  name: 'ngConf 2025',
  date: '3/1/2025',
  time: '8am',
  location: {
    address: '123 Main St',
    city: 'Salt Lake City, UT',
    country: 'USA'
  }
}
```

Comunicacion con componentes hijos

Continuando con el proyecto anteriro, la página de detalles del evento muestra información sobre un evento, incluyendo la dirección. Cree un componente hijo que se encargará de mostrar la direccion y pase la ubicacion del evento al nuevo componente direccion desde el componente event-details. Para hacer esto necesitarás:

Consejo: No llame al elemento `<address>`. `<address>` es un elemento existente en **HTML 5**. Si bien funcionará, puede encontrar problemas de estilo u otros.

- Cree un nuevo componente de dirección que tenga una propiedad de entrada para los datos de dirección
- Verifique que Angular CLI agrego el componente al módulo de aplicación `(app/app.module.ts)`
- Actualice el componente de detalles del evento `(app/event-details.component.ts)` para pasar la dirección a su nuevo componente

Comunicación con el componente padre

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio05` del repositorio.

Este proyecto contiene un componente padre `(app/parent/parent.component.ts)` y un componente secundario `(app/child/child.component.ts)`. El componente hijo tiene un botón "Click Me" y una propiedad de contador que se actualiza constantemente por un temporizador. Agregue una propiedad de salida al componente hijo y enlazelo desde el padre para que el la propiedad 'currentCounter' se actualice al valor actual de la propiedad 'counter' del hijo cuando se presiona el botón "Click Me". Verás un mensaje de felicitaciones cuando lo hayas logrado. Para hacer esto:

- Agregar una propiedad de salida al componente hijo
- En el método buttonClicked en el componente hijo, emita un evento usando la propiedad de salida con el valor actual de 'counter'

- En el elemento `'<child>'` en la plantilla del componente principal, enlace a la propiedad de salida y llame a una función de control en el componente principal, pasando en el valor emitido por el componente hijo
- En la función de control que creó en el paso anterior, asigne a la propiedad 'currentCounter' el valor recibido desde componente hijo

Uso de variables locales para interactuar con componentes secundarios

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio06` del repositorio.

Este proyecto contiene un componente padre `(app/parent/parent.component.ts)` y un componente hijo `(app/child/child.ponent.ts)`. El componente hijo tiene un temporizador que se ejecuta constantemente y un método `stopTimer()` que detendrá la ejecución del temporizador. Utilizando variables locales, llame al método `stopTimer()` del componente secundario desde el componente principal cuando se haga clic en el botón "Detener temporizador" del componente principal. Para hacer esto:

- Agregue una variable local al elemento `<child>` en la plantilla del componente padre
- Agregue un enlace de click al botón "Stop Timer" del padre que usa la variable local para llamar al método `stopTimer()` del hijo

Agregando estilos a Componentes

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio07` del repositorio.

El componente de detalles del evento muestra información sobre un evento, las etiquetas están en un `div` y los valores están en un `div` separado, pero los dos `divs` no estan alineados. Agregue estilos al componente para hacer que el segundo `div` se muestre a la derecha del primer `div`. Siéntete libre de jugar con cualquier otro estilo que te gustaría agregar

Consejo: Configura los dos `divs` en: `display: inline-block`

Para ello necesitarás:

- Agregue clases a la propiedad de estilo en los metadatos del componente (no agregue estilos en línea directamente en la plantilla)
- Agrega las clases a los `divs` en la plantilla

Directivas incorporadas

Los códigos de este laboratorio se encuentran en: https://github.com/ldiamand/curso_angular.git

Repetir datos con ngFor

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio08` del repositorio.

Actualice el componente de la lista de eventos para mostrar los eventos que están almacenados en un arreglo en el componente. Para hacer esto necesitarás:

- Usar `ngFor` para repetir el elemento `<tr>` en el elemento `<tbody>` en la plantilla del componente Se enlazará al arreglo de eventos que ya está definida en el componente
- Agrega enlaces dentro de los elementos `<td>` para enlazar a las propiedades en cada evento

Usando el operador de navegación segura

Continuando con el ejercicio anterior, el componente de la lista de eventos parece estar representándose normalmente en este proyecto, sin embargo, está generando algunos errores en la consola. Solucione los errores utilizando el operador **Safe-Navigator**. Para hacer esto:

- Actualice la plantilla del componente de la lista de eventos utilizando el operador de navegación segura

Ocultar elementos con ngIf

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio10` del repositorio.

Algunos de los eventos que se muestran en el componente de la lista de eventos solo están en línea, lo que significa que tienen una URL asociada, pero no tienen una ubicación física. Algunos eventos no tienen ninguno de los dos. Desafortunadamente, el mensaje "Solo en línea" se muestra para todos los eventos. Use `ngIf` para mostrar el mensaje "Solo en línea" para aquellos eventos que tengan una URL asociada, pero no una ubicación. Además, oculte la ubicación y los elementos `onlineUrl` completamente si un evento no tiene estos datos. Para hacer esto:

- Agregue las directivas y expresiones `ngIf` apropiadas a los elementos correctos

Elementos ocultos con hidden

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio11` del repositorio.

Los eventos que se muestran en el componente de lista de eventos tienen botones de expandir y contraer. Cuando se hace clic en los botones, la propiedad `hidden` del evento se activa o de-

sactiva. Oculte los detalles del evento cuando haga clic en los botones vinculando la expresión `event.hidden` a la propiedad `hidden` del `div` que rodea los detalles del evento (fecha, hora y ubicación). También oculte los botones *Contraer/Expandir* según sea necesario para que solo se muestre el botón apropiado según el estado actual. Para hacer esto:

- Vincule una expresión al `div` que rodea la fecha, la hora y la ubicación para que se oculte cuando la propiedad `hidden` del evento sea verdadera
- Vincule una expresión a la propiedad `hidden` del `div` que rodea a los botones *Contraer/-Expandir* para que solo se vea uno a la vez

Ocultar y mostrar elementos con ngSwitch

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio12` del repositorio.

Algunos de los eventos que se muestran en el componente de la lista de eventos están solo en línea `(event.format='Online')`, otros solo en persona `(event.format='InPerson')`. Algunos eventos no tienen ninguno de los dos, pero no hay eventos que tengan ambos. Desafortunadamente, los mensajes *"En persona"*, *"Solo en línea"* y *"TBD"* se muestran para todos los eventos. Use `ngSwitch` para mostrar solo el mensaje apropiado para cada evento. Cada evento debe tener un solo mensaje. Para hacer esto:

- Utilice `ngSwitch` para mostrar y ocultar los mensajes apropiados en función del evento (Deberá agregar un `span` alrededor de los mensajes actuales para agregar la directiva `ngSwitch`).

Añadiendo estilo con ngClass

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio13` del repositorio.

Algunos de los eventos que se muestran en el componente de la lista de eventos solo están en línea `(event.format='Online')`, otros solo en persona `(event.format='InPerson')`. Algunos eventos no tienen ninguno de los dos, pero no hay eventos que tengan ambos. Use `ngClass` para agregar la clase *in-person* al título del evento `(<h2>)` si `evento.format='InPerson'`, o la clase *online* si `evento.format='OnLine'`. Si `event.format` no es "InPerson" ni "Online", aplique la clase *tbd*. Para hacer esto:

- Puede usar una expresión `ngClass` en línea o vincular `ngClass` a una función. Puede ser mejor usar una función, ya que la expresión puede ser larga.

Añadiendo estilo con ngStyle

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio14` del repositorio.

Algunos de los eventos que se muestran en el componente de la lista de eventos solo están en línea `(event.format='Online')`, otros solo en persona `(event.format='InPerson')`. Algunos eventos no tienen ninguno de los dos, pero no hay eventos que tengan ambos. Use `ngStyle` para hacer que el título del evento `(<h2>)` sea verde si el `evento.formato='InPerson'`, o rojo si el `evento.formato='En lnea'`. Si `event.format` no es "InPerson" ni "Online", haga que el título aparezca en gris `(#aaa)`. Para hacer esto:

- Puede usar una expresión `ngStyle` en línea o vincular `ngStyle` a una función. Puede ser mejor usar una función ya que la expresión puede ser larga.

Creación de servicios reutilizables

Los códigos de este laboratorio se encuentran en: https://github.com/ldiamand/curso_angular.git

Creación e inyección de servicios

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio15` del repositorio.

Los datos para el componente de lista de eventos están actualmente codificados en el componente. Cree un servicio para almacenar estos datos y devuélvalo desde un método `getEvents()` en el servicio. Para ello necesitarás

- Crear el servicio y regístralo en el módulo raíz
- Eliminar los datos del componente, inyectarle el servicio y llamar al mismo para obtener los datos

Enrutamiento y navegación entre páginas

Creando una Ruta

Para este laboratorio vamos a continuar con el código del laboratorio anterior.

La página de lista de eventos de la aplicación se está cargando directamente desde el componente raíz de la aplicación. Cambie la aplicación para que el componente de la lista de eventos se cargue a través de una ruta. Para hacer esto necesitarás:

- Cambiar el componente de la aplicación para usar una salida del enrutador en lugar de mostrar el componente de lista de eventos directamente
- Agregar un archivo de rutas con una ruta para el componente de lista de eventos
- Cargar las rutas en el `app.module`

Enlace a Rutas

Para este laboratorio vamos a continuar con el código del laboratorio anterior.

En primer lugar vamos a crear un componente nuevo que muestre un detalle de un evento particular.

Luego, en la página de la lista de eventos en la aplicación vamos a agregar enlaces a la página de detalles del evento, enlazando para ello la ruta correcta. Actualíce los vinculos correctamente a la página de detalles del evento.Para ello necesitarás

- Agregar la ruta respectiva
- Agregar el enlace `routerLink` apropiado a los enlaces en el componente *lista de evento*

Usando Parámetros de Ruta

Para este laboratorio vamos a continuar con el código del laboratorio anterior.

La página de lista de eventos de la aplicación enlaza con la página de detalles del evento. Si ejecuta esta aplicación puede ver que al hacer clic en el nombre del evento se carga la página de detalles del evento, pero la página de detalles del evento siempre muestra el mismo evento. Puede que no sea obvio que los enlaces pasen la información de identificación del evento en la URL (por ejemplo, *"ng-conf 2037"* enlaza a `/events/3`). Actualice el componente detalles del evento para que utilice este parámetro cuando recupere el evento del servicio de eventos. Para hacer esto necesitará:

- Agregar la propiedad id a los datos del servicio
- Agregar a la ruta el parámetro `id` del evento
- Actualice el componente detalles del evento para usar el parámetro al llamar al servicio de eventos

Navegando desde el código

Para este laboratorio vamos a continuar con el código del laboratorio anterior.

Agregar a la página de detalles del evento (a la que se accede haciendo clic en los enlaces de la página de la lista de eventos) un botón *"Volver a eventos"*. El botón estará vinculado a un método `returnToEvents` que volverá a la página de lista de eventos. Para ello necesitarás

- Agregar un parámetro `Router` al constructor de detalle de eventos
- Conectar el método `returnToEvents` en el componente de detalles del evento para navegar a la página de la lista de eventos

Nota: Se podrían definir dos rutas posibles que se podrían utilizar para volver a la página de lista de eventos.

Cómo evitar que una ruta se active

Para este laboratorio vamos a continuar con el código del laboratorio anterior.

Vamos a agregar un evento no válido en la página de la lista de eventos (el último en la lista). Al hacer clic en el enlace de este evento, irá a la página de detalles del evento para un evento no existente lo cual no se ve muy bien. Agregue una guarda `canActivate` a la ruta de detalles del evento que impida que la página se cargue si el evento no existe. Para ello necesitarás

- Agregar un servicio de activación de ruta que impida que la página de detalles del evento se cargue si la identificación del evento es para un evento que no existe
- Agregar una entrada en la tabla de la lista de eventos inválida

```
<tr>
  <td><a [routerLink]="['/events', 42]">Evento invalido</a></td>
  <td>N/A</td>
  <td>N/A</td>
  <td>N/A</td>
</tr>
```

- Conecta el servicio activación de ruta a la ruta
- No olvide agregar su nuevo servicio activador como proveedor en el módulo

Cómo evitar que una ruta se desactive

Para este laboratorio vamos a continuar con el código del laboratorio anterior.

En esta aplicación, puede acceder a la página de detalles del evento haciendo clic en uno de los enlaces en la página de la lista de eventos. La página de detalles del evento tiene una casilla de verificación "Revisada" que alterna una propiedad seleccionada en el controlador a verdadero o falso cuando se verifica o desmarca. Agregue una guarda a `canDeactivate` a la ruta de detalles del evento que evite que el usuario salga de la página de detalles del evento si el evento no se ha revisado. Para ello necesitarás

- Agreguar un método `canDeactivate` al servicio activador de detalles de eventos que verificará la propiedad revisada del componente detalles de eventos y se volverá verdadero si fue revisado o falso si no
- Conectar la propiedad `canDeactivate` de la ruta de detalles del evento a la guarda `canDeactivate`

Carga previa de datos para un componente

Para este laboratorio vamos a continuar con el código del laboratorio anterior.

El servicio de eventos en esta aplicación está escrito para simular una llamada de API de larga ejecución. Como el componente de la lista de eventos tarda un tiempo en obtener sus datos, la página se procesa parcialmente mientras espera los datos. Agregue un programa de resolución de rutas que espere a que los datos se carguen antes de reproducir cualquier parte del componente de la lista de eventos. Para hacer esto necesitarás:

- Crear un servicio de resolución de lista de eventos con un método de resolución que llame al servicio de eventos y almacene los eventos en la ruta
- Actualizar `events-list.component` para que los eventos salgan de la ruta
- Actualiar la ruta para que el componente de la lista de eventos use la resolución que creó
- No olvides añadir tu servicio de resolución al módulo de la aplicación

