Práctica de San Cristobal Seguros - Frontend

Libro de Laboratorios

October 12, 2023

Acerca de este documento

Este documento se genero a partir de fuentes en LaTeX en https://github.com/bootlin/training-materials.

Correcciones, sugerencias, contribuciones son bienvenidas!

Tipos básicos

Ejercicio 1.1

Dado el siguiente codigo:

1. Usando tu IDE preferido, coloca el cursor sobre las errores en rojo para inspeccionarlos
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;

```



```
6 const mensaje = agregarInventario(nombre, cantidad);
7
8 console.log('[Ejercicio 1.6]', mensaje);
9
10 function agregarInventario(nombre: string, cantidad: number): string {
11   return `Se agregaron ${cantidad} ${nombre}s al inventario.`;
12 }
```


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
```



```
53
54   return animal.species;
55 }
56
57 console.log('[Ejercicio 4.4]',
58   `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

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);
pushToCollection([], stringCollection);

pushToCollection('1', numberCollection);
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 - Clases

Objetivos: Crear clases con propiedades y métodos con tipos, Añadir modificadores de acceso a los miembros de la clase.

Ejercicio 5.1

Dado el siguiente codigo:

```
class MC {
  greet(event = 'party') {
    return `Bienvenido al ${event}`;
  }
}

const mc = new MC();
console.log('[Ejercicio 5.1]', mc.greet('espectaculo'));
```

1 Añadir tipo de parámetro de forma explícita en método ‘greet‘

2 Agregar el tipo de retorno explicito al metodo greet

Ejercicio 5.2

Dado el siguiente codigo:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}

const jane = new Person('Juan', 31);

console.log('[Ejercicio 5.2]', `El nombre de la nueva persona es ${jane.name}.`);
```

1 Añadir tipos de parámetros explícitos al constructor

2 Agregue parámetros con tipos para almacenar valores

Ejercicio 5.3

Dado el siguiente codigo:

```
class Employee {
  title: string;
  salary: number;
  constructor(title: string, salary: number) {
    this.title = title;
    this.salary = salary;
  }
```



```
}

const employee = new Employee('Ingeniero', 100000);

console.log('[Ejercicio 5.3]', `El titulo del nuevo empleado es ${employee.title} y gana $ ${employee
```

1 Hacer que las propiedades de title y salary estén explícitamente disponibles públicamente

2 Reduzca la clase a tres líneas de código manteniendo la funcionalidad

Ejercicio 5.4

Dado el siguiente codigo:

```
class Animal {
  constructor(name) { }
  move(meters) {
    console.log(`${this.name} se movio ${meters}m.`);
  }
}

class Snake {
  move(meters) {
    console.log('Deslizandose...');
    // debe invocar al metodo `move` del padre, con un deslizamiento predeterminado
    // de 5 metros
  }
}

class Pony {
  move(meters) {
    console.log('Galopando...');
    // debe invocar al metodo `move` del padre con un galope predeterminado
    // de 60 metros
  }
}

// La clase Animal no debe ser instanciable.
// Eliminar o comentar una vez que se logra el error deseado.
const andrew = new Animal("Andrew el Animal");
andrew.move(5);

const sammy = new Snake("Sammy la serpiente");
sammy.move();
console.log(sammy.name); // debe devolver error

const pokey = new Pony("Pokey el pony");
pokey.move(34);
console.log(pokey.name); // Should devolver error
```

1 Añadir tipos

2 Hacer que la clase Snake herede de Animal

2 Hacer que la clase Pony herede Animal

2 Hacer que el miembro del nombre no pueda ser accedido públicamente

Ejercicio 5.5

Dado el siguiente codigo:

```
class Furniture {
  constructor(manufacturer: string = 'IKEA') { }
}

class Desk extends Furniture {
  kind() {
    console.log('[Ejercicio 5.5]', `Este es un escritorio hecho por ${this.manufacturer}`);
  }
}

class Chair extends Furniture {
  kind() {
    console.log('[Ejercicio 5.5]', `Esta es una silla hecha por ${this.manufacturer}`);
  }
}

const desk = new Desk();
desk.kind();
desk.manufacturer; // debe devolver error

const chair = new Chair();
chair.kind();
chair.manufacturer; // debe devolver error
}
```

1 Hacer que solo las clases Desk y Chair puedan ver el miembro del fabricante

Ejercicio 5.6

Dado el siguiente codigo:

```
class Student {
  public school: string = 'Harry Herpson High School';
  constructor(private name: string) { };
  introduction() {
    console.log('[Ejercicio 5.6]', `Hola, mi nombre es ${this.name} y asisto a ${Student.school
  }
}

const student = new Student('Morty');
console.log(Student.school);
student.introduction();
```

1 Elimine el error sin cambiar las referencias a ‘Student.school‘

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)
```


Introduccion - Comenzando

Los codigos de este laboratorio se encuentran en:

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

Inicio del componente principal

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

Cada aplicación en Angular se inicia con un componente de aplicación y un módulo raíz. El componente de aplicación `(app/app.component.ts)` y el módulo raíz `(app/app.module.ts)` se crearon, pero la aplicación no inicia. Haga los cambios necesarios para que funcione de forma correcta. Para lograr esto, vamos a necesitar:

- Declarar el componente de aplicación en el módulo raíz `(app/app.module.ts)`
- Agregar el componente de aplicación como el componente de inicio en el modulo `(app/app.module.ts)`
- Utilizar el servicio `platformBrowserDynamic` para iniciar el módulo 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

Para la primer parte de este laboratorio vamos a crear un proyecto nuevo con el nombre `ejercicio03` utilizando Angular CLI. Agreguamos un nuevo componente para mostrar una lista de los próximos eventos usando el HTML y los datos que se muestrean a continuación. Luego, cargamos ese componente desde la plantilla en línea del componente principal de la aplicación `(app/app.component.ts)`. Para hacer esto necesitaremos:

- Crear el componente `event-details`
- Podemos 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 necesario para el enlace de datos mediante 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 variaría si desea 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 anterior, 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 necesitaremos:

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, por ejemplo `event-address`
- 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 al 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 hijo. `(app/child/child.component.ts)`. El componente hijo tiene un botón "*Click Me*" y una propiedad de `counter` que se actualiza constantemente a través de un temporizador. Agregue una propiedad `Output` al componente hijo y enlázelo desde el padre para que la propiedad `currentCounter` del padre se actualice al valor actual de la propiedad `counter` del hijo al presiona el botón "*Click Me*". Veremos un mensaje de felicitaciones cuando lo hayamos logrado. Para hacer esto:

- Agregar una propiedad de salida '`Output`' al componente hijo
- En el método `buttonClicked` del 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 la propiedad de salida y llame a una función de control en el componente principal, pasando 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 hijo desde el componente padre cuando haga clic en el botón "*Detener el contador*" del componente padre. Para hacer esto:

- Agregue una variable local al elemento `<child>` en la plantilla del componente padre
- Agregue un enlace de `click` al botón "*Detener el contador*" del padre. Este 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 elemento `div` y los valores están en un elemento `div` separado, pero no estan alineados. Agregue estilos al componente para hacer que el segundo elemento `div` se muestre a la derecha del primer elemento `div`. Siéntete libre de jugar con cualquier otro estilo que te gustaría agregar

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

Para ello necesitarás:

- Agregar clases a la propiedad de estilo en los metadatos del componente (no agregue directamente estilos en la plantilla)
- Agregar 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

Iterar 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 dentro del componente. Para hacer esto necesitaremos:

- Usar `ngFor` para repetir el elemento `<tr>` dentro del elemento `<tbody>` en la plantilla del componente. Se enlazará al arreglo de eventos que ya está definido dentro del componente
- Enlaza cada elemento `<td>` con las propiedades del evento (no enlace todavía la propiedad `location`)

Usando el operador de navegación segura

Continuando con el ejercicio anterior, vamos a agregar la propiedad `location` a la lista de eventos, mostrando `address`, `city` y `country`. Sin embargo, al hacer esto, apareceran algunos errores en la consola. Solucione los errores utilizando el operador **Safe-Navigator**. Para ello:

- Actualice la plantilla del componente de 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 disponibles *en línea*, lo que significa que tienen una URL asociada, pero no tienen una ubicación física. Algunos eventos no tienen ninguna de los dos opciones. 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 para expandir y contraer. Cuando se hace clic en los botones, la propiedad `hidden` del evento se activa o desactiva. Oculte los detalles del evento cuando haga clic en los botones vinculando la expresión `event.hidden` a la propiedad `hidden` del elemento `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 elemento `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 elemento `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 *"A determinar"* 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 *adeterminar*. 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='Online'`. 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. Tal vez lo mejor sea 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()` del servicio. Para ello necesitarás

- Crear el servicio y regístralo en el módulo raíz
- Eliminar los datos del componente, inyectar 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 que utilice el `router-outlet` del enrutador en lugar de mostrar el componente *lista de eventos* directamente
- Agregar un archivo de rutas con una ruta para el componente *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 una guarda de ruta que impida que la página de detalles del evento se cargue si el `id` del evento es para un evento que no existe
- Agregar una entrada inválida en la tabla de la lista de eventos

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

- Conectar la guarda de ruta a la ruta
- No olvide agregar la guarda 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 selección "*Revisada*" que alterna una propiedad `reviewed` en el controlador a `true` o `false` 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

- Agregar un método `canDeactivate` a la guarda de detalles de eventos que verificará la propiedad `reviewed` del componente detalles de eventos y se volverá `true` si fue revisado o `false` 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 de la aplicación está escrito para simular una llamada a una API que puede tardar varios cientos de milisegundos en ejecutarse Como el componente de la lista de eventos tarda un tiempo en obtener sus datos, la página se procesa parcialmente mientras espera la llegada de 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 resolución que llame al servicio de eventos y almacene los eventos en la ruta
- Actualizar `events-list.component` para que los eventos se obtenga 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

Independizar el enrutamiento en un módulo propio

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

Hasta ahora estuvimos agregando las dependencias hacia el módulo de enrutamiento dentro del módulo raíz. En este ejercicio vamos a crear un módulo propio para la gestión de las rutas. Para ello necesitaremos:

- Crear un módulo nuevo lamado `app-routing`
- Mover el módulo de ruteo al nuevo módulo
- Importar en el módulo raíz el nuevo módulo

Recolectando datos con formularios

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

Creando un formulario reactivo

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio23` del repositorio.

Esta aplicación tiene un formulario de creación de evento que actualmente no está funcionando. Para acceder al formulario de la aplicación, haga clic en el enlace **"Crear nuevo evento"** en la página de lista de eventos. Conecte el formulario utilizando *formularios reactivos (basados en modelos)* para que los datos ingresados se puedan guardar como un nuevo evento. Cuando se presiona el botón de guardar, llame a `eventService.save()` y pase el nuevo evento que guardará los datos ingresados. Después de guardar el evento, envíe al usuario a la página de la lista de eventos. El nuevo evento debería estar visible en dicha página. Para ello var a necesitar:

- Crear el formulario y los controles de formulario en el componente (`.ts`). Conecte el formulario y los elementos del formulario utilizando los enlaces apropiados.
  Nota: asegúrese de que los campos de dirección se guarden en un objeto `address` dentro del evento.
  SUGERENCIA: Puede usar `formGroupName` (es decir, `<div formGroupName="location">`) para envolver los controles de dirección anidados en un objeto ubicación en el html y puede usar `FormGroup` en el componente (es decir, `this.location=new FormGroup({...})`) al igual que lo hace con un objeto de formulario
- Conectar el `ngSubmit` en el formulario
- Conectar el `ngSubmit` a un método de guardado en el componente. Dentro de ese método debe llamar a `eventService.saveEvent` y pasar el nuevo evento con los datos del formulario
- También debe conectar el método de guardado en el componente para llevar de nuevo al usuario a la página de la lista de eventos después de guardar
- Agregue el módulo `ReactiveFormsModule` al módulo de la aplicación

Generando formularios utilizando FormBuilder

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

Modifique la aplicación para que genere los controles utilizando el servicio de `FormBuilder`. Para ello va a necesitar:

- Agregar la importación de la clase `FormBuilder`
- Injectar el servicio en el constructor del componente
- Generar los controles utilizando el servicio

Validando un formulario reactivo

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

Esta aplicación tiene un formulario de creación de eventos que ya está conectado para guardar nuevos eventos. Para acceder al formulario de la aplicación, haga clic en el enlace "Crear nuevo evento" en la página de la lista de eventos. El formulario está conectado con formularios reactivos, pero no tiene ninguna validación. Agregue validación al formulario para que muestre un mensaje de error junto a los campos que tienen un error. Desactive el botón **Guardar** cuando el formulario no sea válido para que el usuario no pueda guardar un evento inválido.

Agregue las siguientes validaciones:

1 haga todos los campos requeridos

2 haga que el país sea obligatorio y que conste de dos letras mayúsculas

3 Asegúrese de mostrar el mensaje de error apropiado para el campo del país según la activación que se active

Para hacer esto necesitaremos:

- Agregar los validadores apropiados a los campos como se definió anteriormente
- Agregar un mensaje de validación para cada campo y agregar estilos al componente para hacer que esos mensajes de validación aparezcan cómo y dónde desea que aparezcan
- Haga que los mensajes de validación aparezcan solo si los campos no son válidos y se han tocado
- Enlazar la propiedad `disabled` en el botón *Guardar* a la propiedad no válida del formulario
- (Opcional) Agregar estilos a los campos de entrada para que tengan un color de fondo diferente si no son válidos

Creación de un formulario basado en plantillas

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio23` del repositorio.

Esta aplicación tiene un formulario de creación de evento que actualmente no está conectado. Para acceder al formulario de la aplicación, haga clic en el enlace **"Crear nuevo evento"** en la página de lista de eventos. Conecte el formulario utilizando *formularios basados en plantillas* para que los datos introducidos se puedan guardar como un nuevo evento. Cuando se presiona el botón de guardar, llame a `eventService.save()` y pase el nuevo evento que guardará los datos ingresados. Después de guardar el evento, envíe al usuario a la página de la lista de eventos. El nuevo evento debería estar visible en dicha página. Para ello vamos a necesitar:

- Conectar el formulario y los elementos del formulario utilizando los enlaces apropiados
  Nota: asegúrese de que los campos de dirección se guarden en un objeto de tipo `address` en el evento
- Agregar `ngSubmit` en el formulario
- Conectar el `ngSubmit` a un método de guardado en el componente y en ese método llamar a `eventService.saveEvent` y pasar el nuevo evento desde los datos del formulario
- También deberá conectar el método de guardado en el componente para llevar de nuevo al usuario a la página de la lista de eventos después de guardar

Validar un formulario basado en plantillas

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

Esta aplicación tiene un formulario de creación de evento que se ha conectado para guardar un nuevo evento, pero le falta validación. Para acceder al formulario en la aplicación, haga clic en el enlace "Crear nuevo evento" en la página de la lista de eventos. Agregue validación al formulario que muestra un mensaje de error junto a los campos que tienen un error. Desactive el botón Guardar cuando el formulario no sea válido para que el usuario no pueda guardar un evento no válido. Agregue las siguientes validaciones: haga todos los campos requeridos; haga que el país sea obligatorio y que conste de dos letras mayúsculas. Asegúrese de mostrar el mensaje de error apropiado para el campo del país según la activación que se active. Para ello necesitaremos

- Agregar los validadores apropiados al campo como se definió anteriormente
- Agregar un mensaje de validación para cada campo y agregar estilos al componente para hacer que esos mensajes de validación aparezcan cómo y dónde deseamos que aparezcan
- Hacer que los mensajes de validación aparezcan solo si los campos no son válidos y se han tocado
- Enlazar la propiedad `disabled` en el botón Guardar a la propiedad no válida del formulario
- (Opcional) Agregue estilos a los campos de entrada para que tengan un color de fondo diferente si no son válidos

Creando un validador personalizado

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

El formulario de creación de eventos en esta aplicación ya está conectado con la validación. Para acceder al formulario en la aplicación, haga clic en el enlace "Crear nuevo evento" en la página de la lista de eventos. Nos gustaría hacerlo para que los eventos no se puedan programar los fines de semana. Agregue un validador personalizado que evite que el campo de fecha sea un fin de semana. Asegúrese de que el mensaje de validación diga *"No puede ser un sábado"* o *"No puede ser un domingo"* según si la fecha no válida es un Sábado o un Domingo. Para ello necesitarás:

- Agregar el validador personalizado. Sugerencia: puede convertir el valor del campo de fecha en una fecha y obtener el día de la semana usando esta sintaxis: `new Date([valor de cadena]).getDay()`. Si eso devuelve un `6` o un `0` es un sábado o un domingo. Convertir una cadena en una fecha utilizando `new Date([valor de cadena])` es un enfoque ingenuo, pero funcionará lo suficientemente bien para este ejercicio siempre que ingrese formatos de fecha válidos
- Hacer que el validador personalizado devuelva un objeto de error que contenga el mensaje de error apropiado
- Agregue el nuevo validador personalizado al campo de fecha
- Actualice el html para mostrar el mensaje de error devuelto por el validador si no es válido

Comunicación con el servidor mediante HTTP

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

Convirtiendo un observable en una promesa

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio41` del repositorio.

Este proyecto contiene un componente principal con un botón y propiedades para un observable, una promesa y el resultado de la promesa. El observable debe convertirse en una promesa. Cuando se hace clic en el botón, escuchará la promesa y mostrará el resultado. Aparecerá un mensaje de éxito si el resultado es correcto. Para hacer esto:

- En el controlador de eventos `ngOnInit`, convierta *"this.obs"* en una promesa
- Asigne esta promesa a la propiedad *"this.promise"*

Realizar una solicitud HTTP

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio42` del repositorio.

Este proyecto contiene un componente principal con un solo botón. Cuando se presiona el botón, el cliente debe realizar una solicitud GET a la siguiente URL:
http://swapi.co/api/films/, y luego asignar la propiedad "results" de los datos retornados a la propiedad `movieList` del componente. Esto mostrará los datos en la página. Para hacer esto:

- emita la llamada desde el objeto `http` con la URL dada
- mapear la respuesta retornada y convertir el json a JavaScript
- suscríbase al observable resultante y asigne la propiedad de resultados a la propiedad `movieList`

Tratar con un flujo de datos

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio43` del repositorio.

Este proyecto contiene un componente principal con una caja de texto. Cuando el usuario escribe en la caja de texto, cada tecla se convierte en un elemento en una secuencia observable. La salida de la secuencia se muestra debajo de la caja de texto. Asigne el flujo, de modo que cada vez que presione la tecla se convierta en su versión en mayúsculas. Para hacer esto:

- en el evento `ngOnInit` de la clase parentComponent, asigne el valor entrante a su valor en mayúscula usando la función `map`

Usando Parámetros de Cadena de Consulta

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio44` del repositorio.

Este proyecto contiene un componente principal con una caja de texto y un botón. Cuando el usuario escribe el nombre de una ciudad y hace clic en el botón, se mostrará el pronóstico actual para mañana. Necesitará construir la URL adecuada para la solicitud agregando la ciudad que proviene del cuadro de entrada y el ID de aplicación que proviene de la propiedad `appid`.

El formato de la URL es `http://api.openweathermap.org/data/2.5/weather?q=_CITY_&APPID=_APPID_`

Para hacer esto:

- en el método `makeRequest`, cree la cadena de URL correcta y pásela al método `http.get`

Manipulación de datos con pipes

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

Usando el pipe lowercase

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio30` del repositorio.

Este proyecto contiene un componente de inicio (`app/home/home.component.ts`) con una propiedad llamada *"myText"*. Utilizando un enlace a dicha propiedad y el pipe integrado correcto, muestre este texto en la página, pero cambielo para que esté en minúsculas.

- Añadir el enlace a la variable
- Agregue un pipe utilizando `lowerCase` para la unión

Usando el pipe de fecha con parámetros

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio31` del repositorio.

Este proyecto contiene un componente de inicio (`app/home/home.component.ts`). La plantilla para el componente de inicio muestra una fecha única 4 veces diferentes, sin utilizar pipes para formatear la salida. En cada línea hay un texto que indica cuál debería ser la visualización correcta del enlace. Agregar pipes a los enlaces para que cada línea coincida con el resultado esperado dado en esa línea.

- Agregar el pipe de fecha
- Agregue el parámetro correcto al pipe de fecha para crear la salida con el formato correcto
- Revise la documentación oficial para obtener ayuda: https://angular.io/docs/ts/latest/api/common/index/DatePipe-pipe.html

Creando un pipe personalizado

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio32` del repositorio.

Este proyecto contiene un componente de inicio (`app/home/home.component.ts`) con una propiedad llamada *"myText"* que usa un pipe `TitleCase`. Cree un pipe personalizado en el archivo `app/title-case.pipe.ts` de manera que el mensaje *"Este título debería tener la primer letra mayúscula de cada palabra"* se muestre en la página en una etiqueta `h1`. Para hacer esto:

- Importar `Pipe` y `PipeTransform` desde `@angular/core`
- Use el decorador `@Pipe`, pasando el nombre del pipe. Asegúrese de coincidir con el valor esperado en el componente de inicio (home)
- Crear la clase del pipe y exportarla
- Implementar la función de transformación para convertir la cadena de entrada en una cadena de título

Usando la función del ciclo de vida ngOnChanges

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio33` del repositorio.

Este proyecto contiene un componente padre e hijo. El componente padre tiene un botón llamado *"Actualizar cantidad"*. El componente hijo tiene una propiedad `quantity` de entrada que establece el padre. El elemento hijo a su vez tiene su propia propiedad llamada `squaredQuantity` que debe contener el cuadrado de la cantidad que le otorga el componente padre, y lo mantiene sincronizado a medida que la cantidad cambia en el elemento padre. Cuando se presiona el botón *"Actualizar cantidad"*, se cambia la cantidad, que actualiza la propiedad de entrada del componente hijo. Debe mantener el valor de `squaredQuantity` sincronizado mientras esto sucede. Al hacer clic en el botón `"Actualizar cantidad"`, debería ver un mensaje de *Felicitaciones*. Para hacer esto:

- Importe `OnChanges` de `@angular/core` e implementelo en la clase de componente hijo (tenga en cuenta que esto es opcional)
- Cree el método `onChanges` en el componente hijo. El nombre debe coincidir exactamente con el valor esperado
- En este método, mantenga la cantidad cuadrada sincronizada con la cantidad

Filtrando datos

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio34` del repositorio.

Este proyecto contiene un componente principal que tiene una lista de películas con calificaciones. Hay varios botones, uno para cada calificación. Implementar el método de filtro para que las películas se filtren en consecuencia. Para hacer esto:

- Crear una propiedad `displayedMovies` (sin inicializar)
- En el constructor, inicialice la propiedad `displaysMovies` para que sea un clon del arreglo de películas
- Utilice el método `slice` para crear un clon del arreglo
- En el método `filter`, llame al método de filtro del arreglo de películas y asigne el resultado al método `displayedMovies`

Ordenando datos

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio35` del repositorio.

Este proyecto contiene un componente principal que tiene una lista de películas con calificaciones. Hay dos botones, uno para clasificar la lista por clasificación ascendente, y otro para clasificarla de forma descendente. Implemente los métodos para los botones para que la lista de películas se ordene correctamente. Para hacer esto:

- Llame al método `sort` del arreglo. Este método tiene 1 parámetro que es una función con 2 parámetros, cada uno una película
- En el método de ordenación ascendente, compare las clasificaciones de películas y devuelva un número positivo si la primera clasificación es anterior a la segunda, cero si las calificaciones son las mismas, o un número negativo si la segunda clasificación viene después de la primera

- Implementar el método opuesto para el orden descendente

Creación de directivas y componentes avanzados

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

Crear una directiva

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio38` del repositorio.

Este proyecto contiene un componente principal y una directiva que permite ocultar el contenido, pero está sin implementar. El componente principal tiene varios `divs` que tienen la directiva `hide` en ellos. Cree una directiva que permita ocultar tal que cuando pase el mouse sobre cualquier elemento con esta directiva, el estilo de visibilidad del elemento esté configurado como *"oculto"*. Para hacer esto:

- Importamos `Directive` y creamos el esqueleto de la directiva
- Damos a la directiva el selector correcto para que coincida con el atributo *"ocultar"*
- Capturamos una referencia al elemento en el que se encuentra la directiva en el constructor
- En el método `Init`, agregamos un detector de eventos al elemento nativo en el evento `'mouseover'`
- En la función callback para el evento, establecemos el `style.visibility` del elemento nativo en la cadena *"oculta"*

Escuchar cambios de parámetros de ruta de una página parametrizada

El código inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio39` del repositorio.

Este proyecto contiene un componente de películas que es un componente enrutado e incluye un parámetro de identificación. El `id` corresponde a la película actual en base de datos de películas. También hay enlaces a la misma ruta con un `ID` diferente en la parte superior del componente. Su tarea es implementar el método `OnInit` para que el componente escuche los parámetros de la ruta y muestre la película actual. Para hacer esto:

- Crear un observador en los parámetros de la ruta activada en el método de inicio
- Establezca la propiedad `currentMovie` del componente en el movimiento correcto en la matriz que coincida con el `ID`

Pruebas de Unidad del código

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

Escribe una prueba simple

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio45` del repositorio.

En el archivo `simple-test.spec.ts`, cree una función `it` que simplemente pruebe que verdadero es verdadero.

Para hacer esto:

1 dentro de la función `describe`, agregue una función `it`

2 en la función `it`, escriba una expectativa, usando el comparador `toBe`

Prueba un servicio

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio46` del repositorio.

En el archivo `movie.service.spec.ts`, cree una función `it` que pruebe que al agregar un miembro al reparto de la pelicula, incrementa el numero de miembros en uno.

1 dentro de la función `describe`, agregue una función `it`

2 en la función `it`, escriba una expectativa, usando la función `toBe`

Usa un objeto espía de Jasmine

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio47` del repositorio.

El archivo `movie.service.spec.ts` contiene una prueba para verificar la función `addCastMember` del servicio de películas. El servicio de películas utiliza el servicio de actor.Necesitamos probar el servicio de películas sin probar el servicio de actor al mismo tiempo, por lo cual, el servicio del actor necesita ser simulado. Cree un servicio de actor simulado que permitirá que la prueba pase.

Para hacer esto:

1 Al comienzo de la función `it`, use `createSpyObj` para crear un servicio de actor simulado

2 Asegúrese de mirar y ver cómo se usa en el servicio de películas

3 Asegúrese de que el servicio de actor simulado devuelva el valor correcto para que la prueba pase

Simule una llamada http

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio48` del repositorio.

El archivo `movie.service.spec.ts` contiene una prueba para verificar la función `addCastMember` del servicio de películas. El servicio de películas envía una solicitud de publicación utilizando el servicio http. Tenemos que simular esa llamada. La llamada simulada debe devolver un observable. Implementar las partes de código faltantes de la prueba.

Para hacer esto:

1 En la función `beforeEach`, cree el objeto http falso que se pueda pasar al constructor del servicio de películas

2 En la función `it`, configure el objeto `mockHttp` para que el método de publicación devuelva un observable

3 Utilice `of(false)` para crear un Observable para devolver

Use toHaveBeenCalledWith

El codigo inicial para este laboratorio se encuentra disponible en la carpeta `ejercicio49` del repositorio.

El archivo `movie.service.spec.ts` contiene una prueba para verificar que la URL es correcta cuando se llama a la publicación http para que persista el nuevo miembro del reparto. La prueba tiene una expectativa fallida en este momento. Implementar una expectativa correcta para comprobar la URL.

Para hacer esto:

1 Eliminar la expectativa existente

2 Crear una nueva expectativa, usando el comparador `toHaveBeenCalledWith`

3 Usar `jasmine.any` para ignorar el segundo parámetro del método posterior

