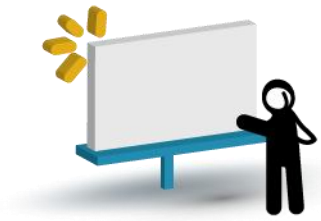
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Desarrollo con Angular

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Unidad 3: Creación de servicios



Presentación:

En la presente unidad vemos un concepto fundamental: los servicios. Estos últimos son los encargados de brindar los datos a los componentes y de consumir las conexiones externas hacia api rest.

Objetivos:

Que los participantes:

- Comprendan para qué se utilizan los servicios
- Aprendan cómo crear servicios.
- Conozcan cómo realizar una conexión con el servidor.

Bloques temáticos:

- Servicios.
- HTTPCLIENT.
- Pipes

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

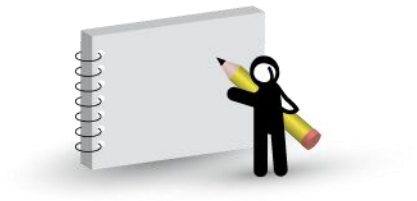
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***



Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Servicios

Definición

Los Componentes son grandes consumidores de servicios. No recuperan datos del servidor, ni validan inputs de usuario, ni logean nada directamente en consola. Delegan todo este tipo de tareas a los Servicios.

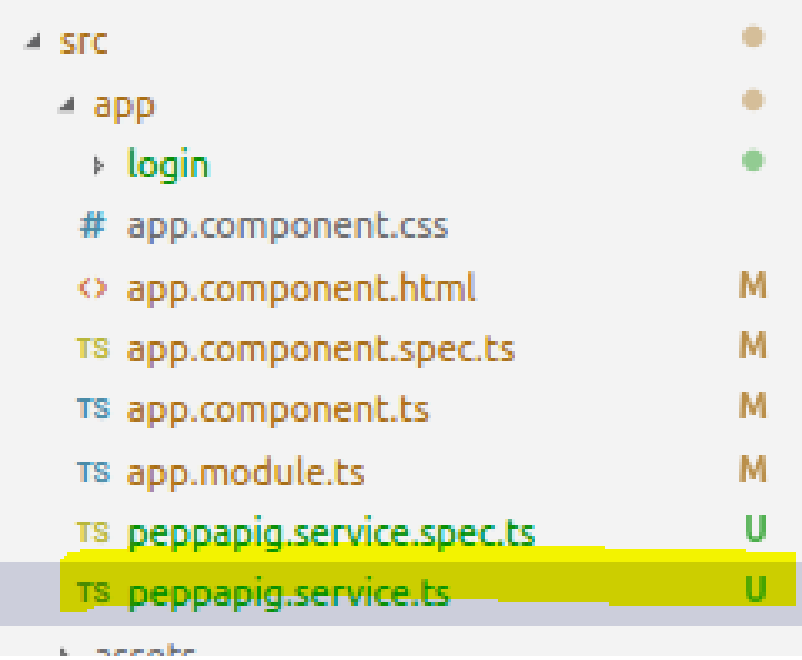
Los servicios deberían contener/hacer algo muy específico. Por ejemplo, serían susceptibles de encapsular en un servicio:

- Servicio de logging
- Servicio de datos
- Bus de mensajes
- Cálculo de Impuestos
- Configuración de la app

Crear un servicio

Ejecutar **ng generate service nombre-servicio**

Por ejemplo ejecutar **ng generate service pepapig**. Esto nos generar un archivo **peppapig.service.ts** en el directorio app



En el contenido del archivo incluimos lo siguiente:

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

@Injectable({
  providedIn: 'root'
})
export class PeppapigService {

  constructor() { }
  getPeppaFriends(){
    return ["pedro poni","madamme gazelle", "dani dog"];
  }
}
```

Como visualizamos en el ejemplo, el servicio es una clase.

En dicha clase nosotros podemos desarrollar distintos métodos, los cuales realizaran consultas a un api externo o bien guardaran las modificaciones en la misma.

Consumir un servicio desde un componente

Incluimos el servicio en el componente en el cual queramos utilizarlo. Por ejemplo en el **app.component.ts**

```
import { PeppapigService } from './peppapig.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers: [PeppapigService]
})
export class AppComponent {
  title = 'app works!';
  visible = false;
  clase="css-class";

  constructor(public peppaservice:PeppapigService) {
```

Luego llamamos al método definido del service, por ejemplo:

```
obtenerServicePeppaPig(){
  console.log(this.peppaservice.getPeppaFriends());

}
```

De esta forma consumimos los datos retornados por el servicio

HTTPCLIENT

Incluirlo en el app.module.ts

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

import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    HttpClientModule,
    ReactiveFormsModule
  ],
```

Incluimos el modulo HTTPCLIENT en el modulo raíz.

Crear un servicio que utilice el HTTPCLIENT

Para la conexión con la API REST debemos crear un servicio el cual se conectara con la misma. Por ejemplo creamos un servicio que se conecte con la API para obtener los usuarios de un sistema:

- Creamos el mismo en la carpeta **/services/**
- En este servicio vamos a importar el componente **HTTP**, **map** y **toPromise** que están en el core de angular 2

Veamos el código de ejemplo:

Consumir el servicio desde un component

Ahora vamos a consumir este servicio creado recientemente desde un controlador:

Importamos el servicio

```
import { UsuariosService } from './usuarios.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers: [UsuariosService]
})
export class AppComponent {
  title = 'app works!';
  visible = false;
  clase="css-class";

  constructor(public usrservice:UsuariosService) {
```

Vemos que en el constructor recibimos como parámetro la variable que haga referencia a ese servicio

Ahora vamos llamamos al método **getUsuarios** definido anteriormente

```
consultarApi(){
 this.usrservice.getUsuarios().subscribe(datos=>{
   console.log(datos);
 })

}
```

El método **suscribe** es un método utilizado por los observables. Un observable es un patrón utilizado para resolver conexiones asincrónicas.

Básicamente nosotros ejecutamos el método **getUsuarios** el cual realiza la consulta a la api, pero el código no queda bloqueado por lo cual el método **getUsuarios** no retorna los datos devueltos por la api sino que retorna un observable. Luego con el método **subscribe** cuando la api retorne datos se ejecutara lo especificado en este método.

Envío de datos por post

Para enviar datos por POST debemos llamar al siguiente método del modulo http client (desde el service):

```
save(data){
  return this.http.post("http://localhost:3000/productos",data)
}
```

El método POST al igual que GET nos retorna un observable, por lo cual a su retorno debemos aplicarle un subscribe desde el controller:

```
export class ProductosAbmComponent implements OnInit {
  myForm:FormGroup
  constructor(private prod:ProductosService,public fb:FormBuilder) { }
  save(){
    this.prod.save(this.myForm.value).subscribe(data=>{
      console.log(data)
    })
  }
  ngOnInit() {
    this.myForm = this.fb.group({
      name:['',[Validators.required]],
      description:['',[Validators.required]],
      sku:['',[Validators.required]],
      price:['',[Validators.required,Validators.min(0)]],
    })
  }

}
```

En el ejemplo vemos que el parámetro **data** recibe los datos cargados por el usuario desde un formulario. Este parámetro es el que se enviara en el body del request.

Envío datos por header

En el caso de tener que enviar un token a través del header de nuestra aplicación o cualquier otro dato a través del mismo debemos hacerlo de la siguiente manera:

```
getProductos(){


  return this.http.get("http://localhost:3000/productos",{headers:{
    'x-access-token':localStorage.getItem('token')
  }})
}
```

En el caso de una petición tipo GET el mismo se envía en el segundo parámetro de la llamada al método.

```
save(data){
  return this.http.post("http://localhost:3000/productos",data,{headers:{
    'x-access-token':localStorage.getItem('token')
  }})
}
```

En el caso de ser una petición de tipo POST el header se envía en el tercer parámetro del método.

El header es enviado utilizando un objeto json.

Pipes

Los pipes me sirven para mostrar con un formato determinado los datos en pantalla.

Pueden encontrar la documentación de los Pipes en el manual de Angular en

https://angular.io/guide/pipes

Los pipes disponibles son:

- currency
- date
- uppercase
- json
- limitTo
- lowercase
- async
- decimal
- percent

Veamos algunos ejemplos sencillos:

Teniendo en cuenta el ejemplo que venimmos desarrollando, si yo quisiera que el nombre del héroe apareciera siempre en mayúscula tendría que reemplazar esta línea:

```
{{ heroe.nombre }}
```

Por esta otra:

```
{{ heroe.nombre | uppercase}}
```

Para dar formato a una fecha podemos utilizar el pipe date. En donde mostrábamos la fecha, podemos hacer que solo se muestre el año reemplazando esta línea:

```
{{ heroe.aparicion }}
```

Por esta otra

```
{{ heroe.aparicion | date: 'y' }}
```

Lowercase y uppercase

Definimos una variable nombre de tipo string en el controlador (ts) de cualquier componente

```
nombre:string = 'Veronica';
```

Este es el contenido de la tabla con los pipes lowercase y uppercase utilizados:

```
<table class="table">
    <thead class="thead-dark">
    <tr>
      <th scope="col">Variable</th>
      <th scope="col">Pipe</th>
      <th scope="col">Resultado</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>{{ nombre }}</td>
      <td> uppercase </td>
      <td>{{ nombre | uppercase}}</td>
    </tr>
    <tr>
      <td>{{ nombre }}</td>
      <td> lowercase </td>
      <td>{{ nombre | lowercase}}</td>
    </tr>

  </tbody>
  </table>
```

Y este es el resultado obtenido una vez que lo probamos en el navegador:

Pipes

Variable	Pipe	Resultado
Veronica	uppercase	VERONICA
Veronica	lowercase	veronica

Aquí tienen las referencias a la documentación de estos pipes

https://angular.io/api/common/LowerCasePipe

https://angular.io/api/common/UpperCasePipe

Slice

https://angular.io/api/common/SlicePipe

El pipe slice nos permite cortar la longitud de una variable. Por ejemplo en este caso si indicamos slice:3 cortara las tres primeras letras del nombre.

Si colocamos:

```
<tr>
      <td>{{ nombre }}</td>
      <td> slice:3 </td>
      <td>{{ nombre | slice:3}}</td>
</tr>
```

El resultado obtenido es onica

Si queremos dejar las tres primeras letras y cortar las restantes:

```
<tr>
      <td>{{ nombre }}</td>
      <td> slice:0:3 </td>
      <td>{{ nombre | slice:0:3}}</td>
</tr>
```

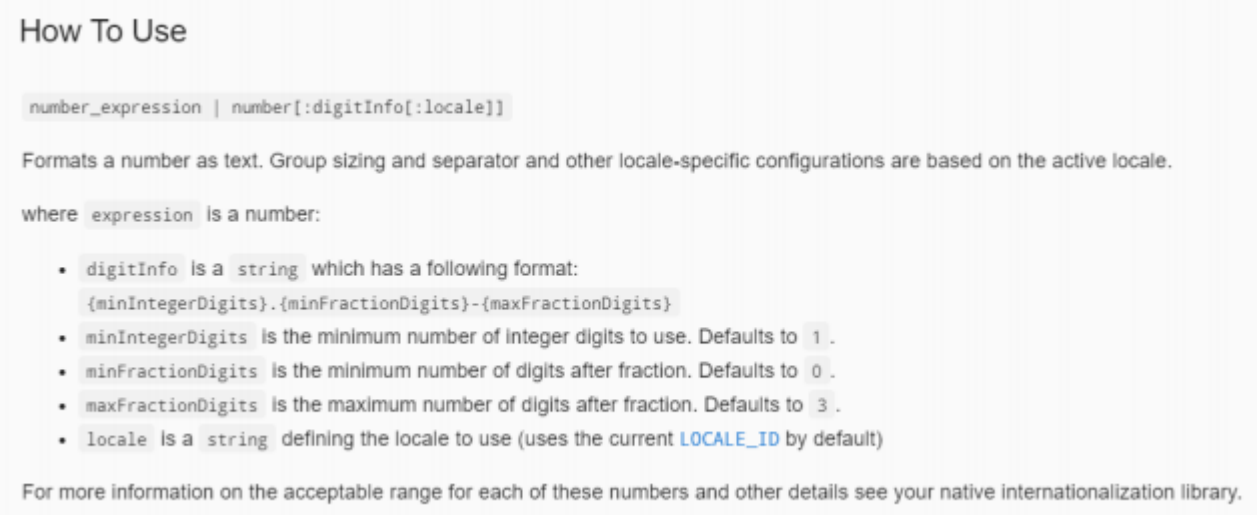
Lo mismo podemos hacer con una variable del tipo array. Si tenemos una array de 10 elementos del 0 al 9 y escribimos slice:1:5 mostrará solo los elementos del 1 al 4.

Decimal

El pipe decimal me sirve para mostrar un número en un formato en partícular. Si bien el pipe se llama decimal en realidad se utiliza con la palabra number.

La documentación de Angular nos explica cómo utilizarlo:

https://angular.io/api/common/DecimalPipe



How To Use

`number_expression | number[:digitInfo[:locale]]`

Formats a number as text. Group sizing and separator and other locale-specific configurations are based on the active locale.

where `expression` is a number:

- `digitInfo` is a `string` which has a following format:
  `{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}`
- `minIntegerDigits` is the minimum number of integer digits to use. Defaults to `1`.
- `minFractionDigits` is the minimum number of digits after fraction. Defaults to `0`.
- `maxFractionDigits` is the maximum number of digits after fraction. Defaults to `3`.
- `locale` is a `string` defining the locale to use (uses the current `LOCALE_ID` by default)

For more information on the acceptable range for each of these numbers and other details see your native internationalization library.

En la documentación también tienen varios ejemplos.

Agregamos algunos a nuestra tabla, definiendo la variable PI.

Percent

Muestra el valor con formato de porcentaje. Funciona muy parecido al pipe decimal.

https://angular.io/api/common/PercentPipe

How To Use

```
number_expression | percent[:digitInfo[:locale]]
```

Description

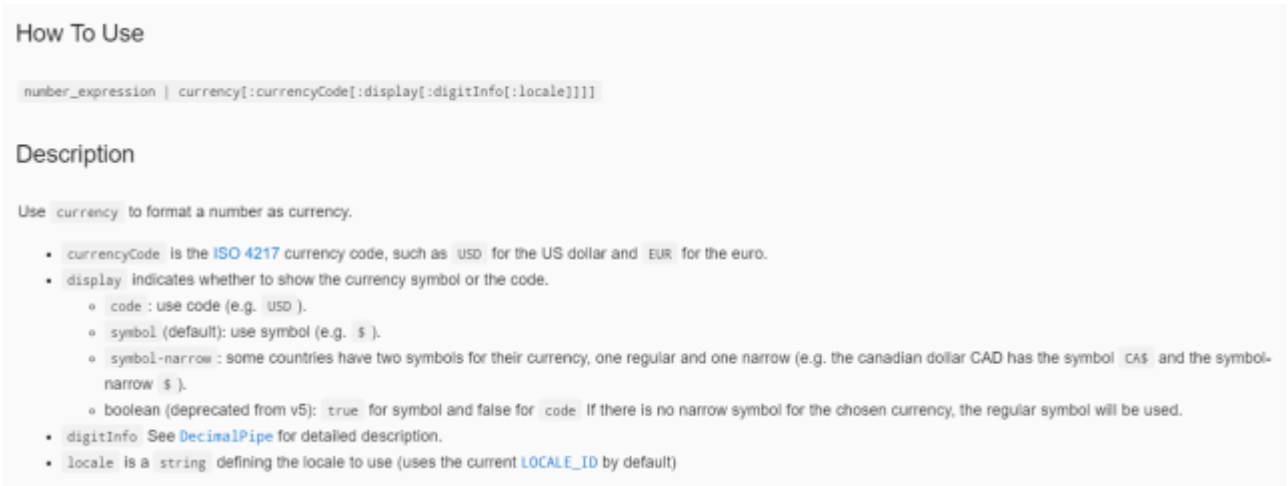
Formats a number as percentage.

- `digitInfo` See `DecimalPipe` for detailed description.
- `locale` is a `string` defining the locale to use (uses the current `LOCALE_ID` by default)

Currency

https://angular.io/api/common/CurrencyPipe

Es utilizado para mostrar formatos de moneda.



How To Use

```
number_expression | currency[:currencyCode[:display[:digitInfo[:locale]]]]
```

Description

Use `currency` to format a number as currency.

- `currencyCode` is the ISO 4217 currency code, such as `USD` for the US dollar and `EUR` for the euro.
- `display` indicates whether to show the currency symbol or the code.
  - `code` : use code (e.g. `USD`).
  - `symbol` (default): use symbol (e.g. `$`).
  - `symbol-narrow` : some countries have two symbols for their currency, one regular and one narrow (e.g. the canadian dollar CAD has the symbol `CA$` and the symbol-narrow `$`).
  - boolean (deprecated from v5): `true` for symbol and false for `code` If there is no narrow symbol for the chosen currency, the regular symbol will be used.
- `digitInfo` See `DecimalPipe` for detailed description.
- `locale` is a `string` defining the locale to use (uses the current `LOCALE_ID` by default)

Json

https://angular.io/api/common/JsonPipe

Async

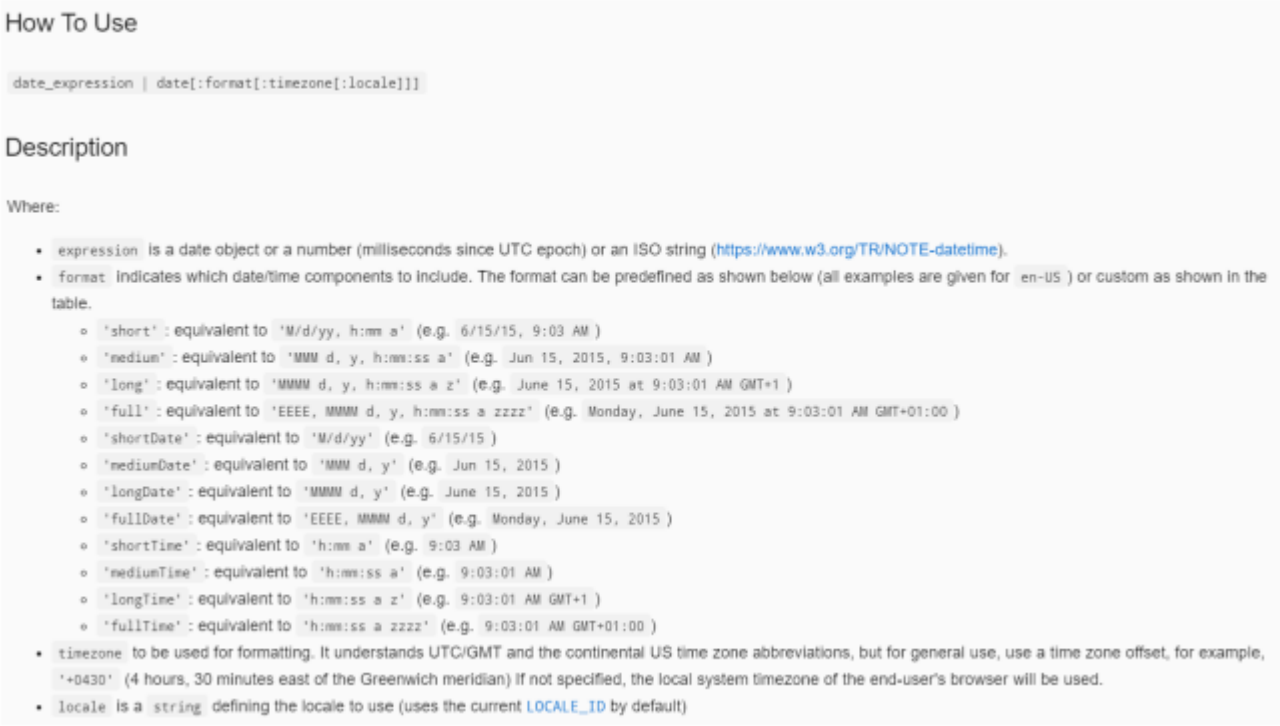
Sirve para trabajar con promesas. Muestra el valor en forma asincrónica una vez que recibe la información.

https://angular.io/api/common/AsyncPipe

Date

Vamos a ver cómo funciona el pipe de fecha y como pasarlo a español.

https://angular.io/api/common/DatePipe



How To Use

`date_expression | date[:format[:timezone[:locale]]]`

Description

Where:

- `expression` is a date object or a number (milliseconds since UTC epoch) or an ISO string (https://www.w3.org/TR/NOTE-datetime).
- `format` indicates which date/time components to include. The format can be predefined as shown below (all examples are given for `en-US`) or custom as shown in the table.
  - `'short'`: equivalent to `'M/d/yy, h:mm a'` (e.g. `6/15/15, 9:03 AM`)
  - `'medium'`: equivalent to `'MMM d, y, h:mm:ss a'` (e.g. `Jun 15, 2015, 9:03:01 AM`)
  - `'long'`: equivalent to `'MMMM d, y, h:mm:ss a z'` (e.g. `June 15, 2015 at 9:03:01 AM GMT+1`)
  - `'full'`: equivalent to `'EEEE, MMMM d, y, h:mm:ss a zzzz'` (e.g. `Monday, June 15, 2015 at 9:03:01 AM GMT+01:00`)
  - `'shortDate'`: equivalent to `'M/d/yy'` (e.g. `6/15/15`)
  - `'mediumDate'`: equivalent to `'MMM d, y'` (e.g. `Jun 15, 2015`)
  - `'longDate'`: equivalent to `'MMMM d, y'` (e.g. `June 15, 2015`)
  - `'fullDate'`: equivalent to `'EEEE, MMMM d, y'` (e.g. `Monday, June 15, 2015`)
  - `'shortTime'`: equivalent to `'h:mm a'` (e.g. `9:03 AM`)
  - `'mediumTime'`: equivalent to `'h:mm:ss a'` (e.g. `9:03:01 AM`)
  - `'longTime'`: equivalent to `'h:mm:ss a z'` (e.g. `9:03:01 AM GMT+1`)
  - `'fullTime'`: equivalent to `'h:mm:ss a zzzz'` (e.g. `9:03:01 AM GMT+01:00`)
- `timezone` to be used for formatting. It understands UTC/GMT and the continental US time zone abbreviations, but for general use, use a time zone offset, for example, `'+0430'` (4 hours, 30 minutes east of the Greenwich meridian) If not specified, the local system timezone of the end-user's browser will be used.
- `locale` is a `string` defining the locale to use (uses the current `LOCALE_ID` by default)

Para que tome el idioma español tenemos que agregar las siguientes líneas en el a

app.module.ts

```
//para que tome las fechas en español
import { LOCALE_ID } from '@angular/core';
import localeEs from '@angular/common/locales/es';
import { registerLocaleData } from '@angular/common';
registerLocaleData(localeEs);
```

Y colocar en el mismo archivo, dentro del array providers lo siguiente:

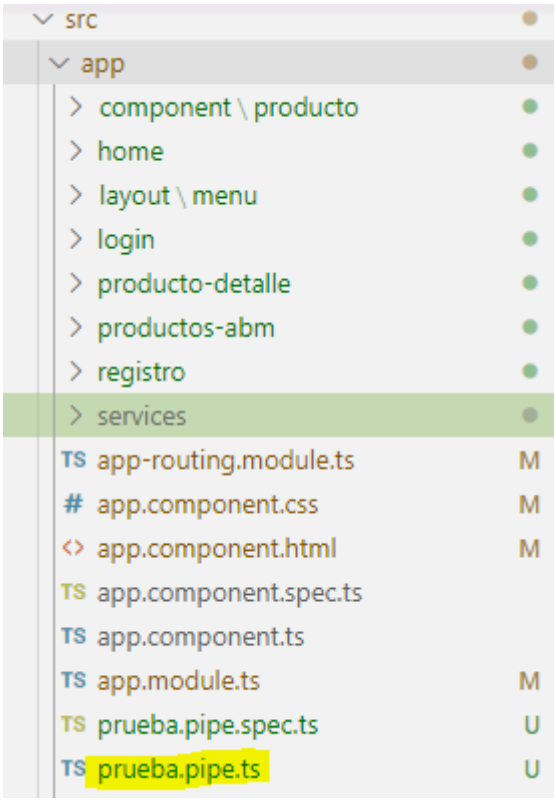
```
providers: [
    {provide: LOCALE_ID, useValue: 'es'}
  ],
```

Pipes personalizados

Ejecutar utilizando el cli de angular

Ng g pipe *nombre_pipe*

Al ejecutar el comando se creara el archivo con el nombre del pipe dentro del directorio app:



Dentro del controlador podemos agregar el código que necesitemos a la hora de realizar una transformación, por ejemplo:

```
export class capitalizadoPipe implements PipeTransform {
  transform(value: string, todas:boolean=true ): string {
    value = value.toLowerCase();

    let nombres = value.split(" ");

    if (todas){
      for ( let i in nombres ) {
        nombres [i] = nombres[i][0].toUpperCase() + nombres[i].substr(1);
      }
    }else{
      nombres[0] = nombres[0][0].toUpperCase() + nombres[0].substr(1);
    }


    return nombres.join(" ");
  }
}
```

El value es la variable sobre el cual se va a aplicar el pipe y args[] son los parámetros que se le pueden pasar. En este caso le pasamos la variable string nombre2 sin parámetros en caso de que queramos que capitalice todas las palabras, ese sería el funcionamiento por defecto y le pasamos el valor false para que capitalice solo la primer palabra.

Desde la vista de un componente lo debemos llamar de la siguiente manera:

```
<tr>
      <td>{{ nombre2 }}</td>
      <td> capitalizado </td>
      <td><pre>{{ nombre2 | capitalizado }}</pre></td>


  </tr>
  <tr>
    <td>{{ nombre2 }}</td>
    <td> capitalizado:false </td>
    <td><pre>{{ nombre2 | capitalizado:false }}</pre></td>
</tr>
```

Bibliografía utilizada y sugerida

https://cli.angular.io/

https://www.typescriptlang.org/

http://victorroblesweb.es/tag/manual-de-angular-2-en-espanol/

https://app.desarrolloweb.com/manuales/manual-angular2

https://www.youtube.com/watch?v=H9Dtgy3Fd40

https://angular.io/guide/pipes

Lo que vimos:

En esta unidad hemos aprendido qué son los servicios, cómo generarlos, cómo utilizarlos y cómo conectarlos con una API REST

Lo que viene:

En la próxima unidad veremos cómo generar una aplicación web navegable con angular.