Día 6

Conceptos

- Uso avanzado de Observables
- Pipes
- Internacionalización

Uso avanzado de Observables

- *of*: algo que ya tenemos → Observable de algo que ya tenemos
- *map*: resultado obtenido → resultado obtenido modificado
- *flatMap*: resultado obtenido → Observable de otra cosa
- *zip*: varios Observables → resultados obtenidos de varios Observables

Pipes

- Crear una nueva aplicación de Ionic con la template blank.
- Ilustrar el funcionamiento de algunas de las pipes más comunes que ya incluye Angular: uppercase, lowercase, date, number…
- Los pipes se pueden anidar.
- Vamos a crear nuestra propia pipe que, dada una cadena y un número n, trunca la cadena si la longitud de la cadena es menor que n y muestra 3 puntos suspensivos. Si n es 0, la cadena no se truncaría.
- En primer lugar, generamos el archivo sobre el que vamos a implementar la pipe con:

  ionic generate pipe pipes/truncate
- Al igual que con los componentes, debemos de disponer de un módulo que incluya todos los pipes que declaramos. **Elimina la declaración que se ha hecho automáticamente de TruncatePipe en AppModule** y en su lugar, declara la pipe en el nuevo módulo de pipes que creas con:

  ionic generate module pipes
- Este módulo de pipes deberás importarlo en los módulos de aquellas páginas donde quieras usar un pipe creado para el proyecto.

- Puedes usar los pipes en la lógica del controlador, por si requirieses la modificación de algún dato dentro de él. Se inyectan como los servicios, y se tiene que invocar el método *transform*. Las pipes propias debemos hacerlas *Injectable*, como los servicios.

Internacionalización

- Instalamos @ngx-translate/core y @ngx-translate/http-loader en el proyecto con:

  npm install –save @ngx-translate/core @ngx-translate/http-loader

- Importar las siguientes dependencias en *app.module.ts*:

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { TranslateModule, TranslateLoader } from '@ngx-translate/core';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';
```

- Debajo de las importaciones anteriores, declarar la función con la que se cargará el archivo de cadenas oportuno:

```
export function HttpLoaderFactory(httpClient: HttpClient) {
  return new TranslateHttpLoader(httpClient, './assets/i18n/', '.json');
}
```

- Añadir al nodo imports del *AppModule*:

```
imports: [
  …

  HttpClientModule,
  TranslateModule.forRoot({
    loader: {
      provide: TranslateLoader,
      useFactory: HttpLoaderFactory,
      deps: [HttpClient]
    }
  }),
],
```

- En *app.component.ts* hay que cargar el idioma adecuado:

```
import { TranslateService } from '@ngx-translate/core'
…
constructor(
  …
  private translate: TranslateService) {
}
ngOnInit() {
   this.translate.use('en');
}
```

- Ya hemos configurado el módulo de traducciones en el módulo y componente principal de la aplicación. Ahora, cada vez que queramos usarlo en una página, debemos importarlo, añadiéndolo al nodo *imports* de su módulo:

```
imports: [
  …
  TranslateModule
],
```

- Crear un archivo *assets/en.json* que contiene las cadenas de la aplicación en idioma inglés. Éstas podrían anidarse según funcionalidad, pantallas, comunes, etc.
- Exponer las diferentes formas de acceder a los literales. Considerar que pueden estar parametrizados.
- El idioma se puede cargar según: el idioma en que esté configurado el dispositivo, el navegador del cliente o incluso podemos ofrecer al usuario la posibilidad de que lo elija.

Ejercicio 1

Mejore el pipe *truncate* para que no trunque una palabra por la mitad, si no que trunque a partir de donde acaba.

Por ejemplo, el resultado de:

```
{{ 'El veloz murciélago hindú comía feliz cardillo y
kiwi' | truncate:40 }}
```

Sería:

```
El veloz murciélago hindú comía feliz cardillo…
```

Sin la mejora, la palabra "cardillo" se hubiese quedado en "ca".

Ejercicio 2

Mejore el pipe *truncate* para que, si recibe como segundo parámetro la palabra "words", muestre a lo sumo las *n* primeras palabras de la cadena. Si la cadena de entrada resulta tener más de *n* palabras, se deben mostrar al final puntos suspensivos. Si no se define *n* o es 0, la cadena no se truncaría.

Por ejemplo, el resultado de:

```
{{ 'El veloz murciélago hindú comía feliz cardillo y
kiwi' | truncate:5:'words' }}
```

Debe ser:

```
El veloz murciélago hindú comía…
```

Pero si n hubiera sido mayor o igual a 9, la cadena de salida hubiera sido idéntica a la de entrada.

Si el segundo parámetro en vez de "words" es cualquier otra cosa, el comportamiento sería el mismo que antes.

Ejercicio 3

Imagínese que está participando en el desarrollo de un foro, que está basado en tecnología Angular, de una empresa que distribuye zapatillas deportivas. A los directivos de la empresa no les parece bien que los usuarios del foro hablen de marcas de zapatillas deportivas que ellos no comercializan.

Se dispone de una API de *posts* que gestiona los mensajes del foro y otra de *brands* que gestiona las marcas que la empresa comercializa. Se pretende que el servicio de Angular que obtiene el detalle de un mensaje concreto del foro, *getMessage(id: number)*, devuelva el campo mensaje ya procesado, donde se haya censurado con asteriscos la mención de aquellas marcas que no comercialice la empresa (aquellas *brands* que no tienen el atributo *isSoldByUs* a *true*).

Así, por ejemplo, si un usuario escribió:

"Yo me compré unas Adidas, pero he visto las ecco de este año y me gustan más."

El mensaje resultante debería ser:

"Yo me compré unas Adidas, pero he visto las **** de este año y me gustan más."

No pretenda usar pipes para dar solución a este problema porque resultaría muy compleja. Céntrese en hacerlo a nivel de respuesta del servicio.