APPANGULAR MULTI IDIOMA

1.- Internacionalización / Localización — ¿Qué son?

Ahora que ya tenemos claro lo que vamos a hacer en esta serie de artículos, voy a explicar de forma lo más resumida posible que es la Internacionalización y la Localización.

1.1. Internacionalización.

Es el proceso mediante el cual se prepara un elemento o producto para permitir su adaptación a diferentes regiones. En el caso de los programas informáticos implica prepararlo para poder ser traducido a varios idiomas, utilizar monedas diferentes o usar distintos formatos de fecha, por citar unos ejemplos. En algunos casos puede implicar pantallas o procesos de negocio diferentes. Así se puede decir que un programa informático esta internacionalizado cuando permite su adaptación a diferentes regiones.

1.2. Localización.

Es el proceso mediante el cual un producto internacionalizado se configura para una determinada región, aprovechando las opciones que la internacionalización previa de este producto ha permitido. Por ejemplo, la internacionalización puede permitir usar distintos formatos de fecha, y la localización es seleccionar el adecuado para una región. Así no tiene sentido decir «un programa internacionalizado a España», ya que la internacionalización es genérica, o decir «un programa localizado» sin indicar a qué región se ha localizado.

2.- Creando el proyecto de Angular.

Una vez que ya tenemos lo necesario para publicar un proyecto, comenzamos creando el mismo.

Para crear el proyecto de Angular, teniendo el CLI de Angular instalado en nuestro equipo ejecutamos la siguiente orden, donde vamos a crear nuestro proyecto con los estilos definidos como CSS y la configuración de las rutas añadidas:

```
jsersan@iMac-de-Jose angular % ng new appMultilingual --style=css --routing
CREATE appMultilingual/README.md (1061 bytes)
CREATE appMultilingual/.editorconfig (274 bytes)
CREATE appMultilingual/.gitignore (620 bytes)
CREATE appMultilingual/angular.json (3094 bytes)
CREATE appMultilingual/package.json (1080 bytes)
CREATE appMultilingual/tsconfig.json (863 bytes)
CREATE appMultilingual/.browserslistrc (600 bytes)
CREATE appMultilingual/karma.conf.js (1433 bytes)
CREATE appMultilingual/tsconfig.app.json (287 bytes)
CREATE appMultilingual/tsconfig.spec.json (333 bytes)
CREATE appMultilingual/.vscode/extensions.json (130 bytes)
CREATE appMultilingual/.vscode/launch.json (474 bytes)
CREATE appMultilingual/.vscode/tasks.json (938 bytes)
CREATE appMultilingual/src/favicon.ico (948 bytes)
CREATE appMultilingual/src/index.html (301 bytes)
CREATE appMultilingual/src/main.ts (372 bytes)
```

Al finalizar:

```
CREATE appMultilingual/src/app/app-routing.module.ts (245 bytes)
CREATE appMultilingual/src/app/app.module.ts (393 bytes)
CREATE appMultilingual/src/app/app.component.css (0 bytes)
CREATE appMultilingual/src/app/app.component.html (23364 bytes)
CREATE appMultilingual/src/app/app.component.spec.ts (1100 bytes)
CREATE appMultilingual/src/app/app.component.ts (219 bytes)
✔ Packages installed successfully.
    Successfully initialized git.
```

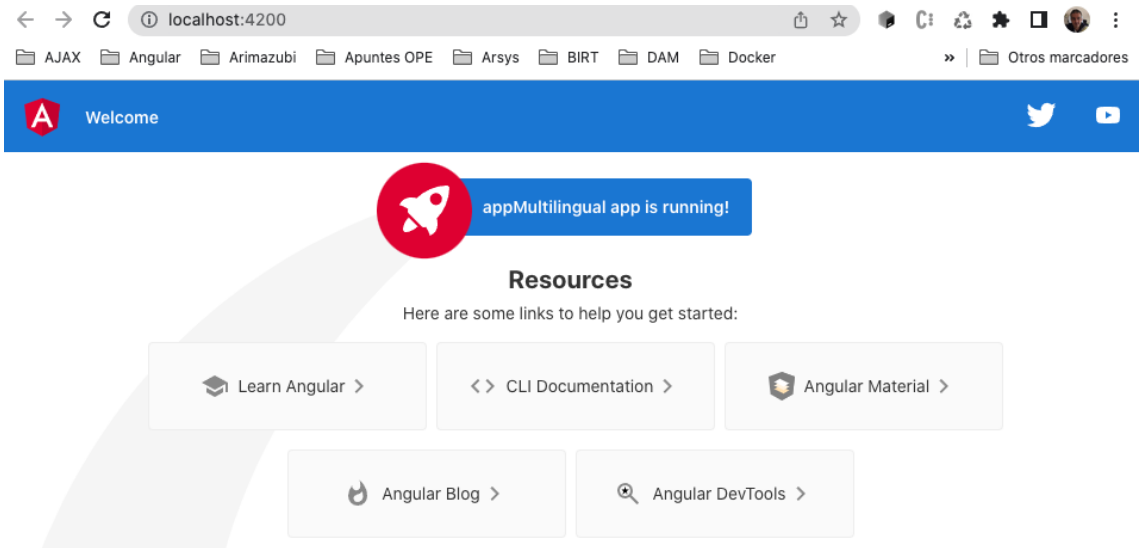
Recuerden:

- ng: Para ejecutar un comando del CLI de Angular.
- new: Para hacer referencia que queremos crear un nuevo proyecto.
- angular-i18n-ngx-translate: Nombre del proyecto (esto podeís poner lo que queráis).
- -- styles=css: Para especificar que vamos a usar estilos CSS.
- -- routing: Para añadir configuración de rutas.

Levantamos:

$ ng serve -o
$ npm start

Por defecto se inicializa en el puerto 4200, por lo que para acceder al contenido de la aplicación debemos de acceder mediante la URL:



3.- ngx translate — ¿Qué es?

Es una librería que permite la carga dinámica de archivos de traducciones en nuestras aplicaciones angular, cuya capacidad de carga dinámica de archivos nos permite cambiar el idioma de la aplicación sin necesidad de recargar la pantalla. Trabajaremos con ficheros de extensión JSON.

Para trabajar con esta librería, vamos a trabajar haciendo uso de la documentación oficial: http://www.ngx-translate.com/

4.- ngx translate — Instalación

Para descargar el módulo de traducciones y el cargador de traducciones de ngx-translate, desde nuestro terminal ejecutamos el siguiente comando añadiendo las dependencias de producción en nuestro package.json.

```
jsersan@iMac-de-Jose appMultilingual % npm install @ngx-translate/core @ngx-translate/http-loader

added 2 packages, removed 1 package, and audited 918 packages in 4s

116 packages are looking for funding
  run `npm fund` for details

5 vulnerabilities (4 moderate, 1 high)

To address all issues, run:
  npm audit fix --force
```

5.- Crear archivos de los idiomas

Vamos a crear tres ficheros para añadir las traducciones en Español, Euskara e Inglés.

Para ello, nos dirigimos a la carpeta "**assets**" de nuestro proyecto Angular en "**src/assets**", creamos un directorio llamado "i18n" y dentro de él tres ficheros JSON con los nombres **es.json**, **en.json** y **eus.json**.

Añadimos el siguiente contenido. Al de inglés:

src > assets > i18n > { } en.json > { } HOME > EUSKARA

```json
{
    "HOME": {
        "TITLE": "i18n (Internazionalization) example to Angular Project",
        "ENGLISH": "English",
        "SPANISH": "Spanish",
        "EUSKARA": "Euskara"
    }
}
```

Español:

src > assets > i18n > { } es.json > { } HOME > EUSKARA

```
{
    "HOME": {
        "TITLE": "Ejemplo de Proyecto Angular con i18n (Internacionalizatión)",
        "ENGLISH": "Inglés",
        "SPANISH": "Español",
        "EUSKARA": "Euskara"
    }
}
```

Euskera:

src > assets > i18n > { } eus.json > { } HOME > TITLE

```
{
    "HOME": {
        "TITLE": "I18n bidezko Angular Proiektuaren Adibidea (Nazioartekotzea)",
        "ENGLISH": "Inglés",
        "SPANISH": "Español",
        "EUSKARA": "Euskara"
    }
}
```

6.- Configuración de ngx-translate

Una vez que ya disponemos de los ficheros con los textos en los dos idiomas, lo que tenemos que hacer es configurar el servicio de traducciones para lo cual importaremos los módulos de traducción dentro del **app.module.ts**, necesitando importar también el módulo y el cliente http, para el correcto funcionamiento del servicio de traducciones que vamos a usar.

Crearemos también dentro del archivo **app.module.ts** una factoría para cargar los archivos de traducciones que lo vamos a llamar “createTranslateLoader”:

src > app > app.module.ts > createTranslateLoader

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule, HttpClient } from '@angular/common/http';
import { TranslateModule, TranslateLoader } from '@ngx-translate/core';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';
import { FormsModule } from '@angular/forms';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

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

Esta librería se encargará de cargar en la aplicación los distintos ficheros de idiomas de la carpeta i18n dependiendo del idioma seleccionado en cada momento, por ejemplo, si el idioma seleccionado es inglés (en), la librería cargara el fichero **assets/i18n/en.json.**

Para acabar la configuración de la aplicación, añadimos al import del **app.module.ts** la configuración de carga del servicio de traducciones, quedando el fichero **app.module.ts** de la siguiente manera.

src > app > app.module.ts > AppModule

```
1   import { NgModule } from '@angular/core';
2   import { BrowserModule } from '@angular/platform-browser';
3   import { HttpClientModule, HttpClient } from '@angular/common/http';
4   import { TranslateModule, TranslateLoader } from '@ngx-translate/core';
5   import { TranslateHttpLoader } from '@ngx-translate/http-loader';
6   import { FormsModule } from '@angular/forms';
7
8   import { AppRoutingModule } from './app-routing.module';
9   import { AppComponent } from './app.component';
10
11  export function createTranslateLoader(http: HttpClient) {
12    return new TranslateHttpLoader(http, './assets/i18n/', '.json');
13  }
14
15  @NgModule({
16    declarations: [
17      AppComponent
18    ],
19    imports: [
20      BrowserModule,
21      AppRoutingModule,
22      FormsModule,
23      HttpClientModule,
24      TranslateModule.forRoot({
25        loader: {
26          provide: TranslateLoader,
27          useFactory: createTranslateLoader,
28          deps: [HttpClient]
29        }
30      })
31    ],
32    providers: [],
33    bootstrap: [AppComponent]
34  })
35  export class AppModule { }
```

7.- Usando las traducciones — Componente.

Ahora que ya tenemos configurado el apartado para gestionar las traducciones, vamos a implementar el uso de las traducciones en un componente, que en este caso lo haremos en el componente inicial que es el componente typescript “App Component”:

src > app > app.component.ts > AppComponent

```typescript
import { Component } from '@angular/core';
// Importamos el servicio de traducciones para hacer uso de él
import { TranslateService } from '@ngx-translate/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'angular-i18n-ngx-translate';
  // Esto es lo que añadimos
  selectedLanguage = 'es';

  constructor(private translateService: TranslateService) {
      this.translateService.setDefaultLang(this.selectedLanguage);
      this.translateService.use(this.selectedLanguage);
  }

  selectLanguage(lang: string) {
      this.translateService.use(lang);
  }
  // Hsta aquí
}
```

Lo primero será cambiar el comportamiento del componente, para lo cual añadimos una variable selectedLanguage inicializada a español (es) para que guarde el idioma seleccionado.

Dentro del constructor del componente inyectamos el servicio de traducciones que nos permitirá cambiar el lenguaje seleccionado, además estableceremos el idioma por defecto mediante la función setDefaultLang() y el idioma actual con use().

Por último, creamos una función para cambiar el idioma selectLanguage ().

8.- Usando las traducciones — Template

Después de realizar las configuraciones y añadir las funcionalidades dentro del componente lo que tenemos que hacer es crear en el template (**app.componente.html**) un título y un selector para cambiar el idioma, que servirá para traducir los textos usamos el pipe translate al que le vamos a pasar como parámetro la referencia a la traducción de los archivos json, en el caso del título 'HOME.TITLE' en app.component.html:

src > app > app.component.html > div
Go to component

```
<div style="text-align:center">
  <h1 (click)="selectLanguage">
    {{ 'HOME.TITLE' | translate }}
  </h1>
  <select
    [ngModel]="selectedLanguage"
    (ngModelChange)="selectLanguage($event)">
    <option value="es">
      {{ 'HOME.SPANISH' | translate }}
    </option>
    <option value="en">
      {{ 'HOME.ENGLISH' | translate }}
    </option>
    <option value="eus">
      {{ 'HOME.EUSKARA' | translate }}
    </option>
  </select>
</div>
```

Lanzamos el proyecto. La primera pantalla es en castellano:

Si hacemos click en el selector de idiomas, tenemos la opción de seleccionar el idioma Inglés, si así lo deseamos. Vemos como el título de nuestra app cambia al que hemos especificado en la referencia "HOME.TITLE" para el idioma inglés, en el fichero en.json:

Y en euskara: