Angular Internationalization - i18n

TABLE OF CONTENTS

Introduction 3

Internalization. 4

What is NGX-Translate? 5

Implementation with NGX-Translate 5

Create an angular project 5

Create pages and services 6

Generate Home Page 7

Routing Configuration 8

Install translate plugins 10

Language Assets 11

Configure with i18n files 12

Create translateConfig Service 13

Localizing Application 14

Result 15

Implementation with i18n-library 16

Create an angular project 16

Add localize to angular project 16

Setting up the app component 17

Create a translation source file 17

Translating the content 18

Configure the app to serve in multiple languages 20

Execution Demo 22

Conclusion 23

# Logo, company name Description automatically generated

# Introduction

Modern web and mobile user experience is a worldwide thing. Localization of our application (supporting multiple languages) will help us to reach worldwide people. Angular is offering Internationalization(i18n) plugins to enrich your application with multiple languages. This can be accomplished in an Angular application through third party libraries, such as ngx-translate, or you can use the built-in i18n functionality. This documentation gives a brief overview of how to translate and use internalization-i18n to convert into multiple languages and by using built-in i18n library, by using these two libraries we can make our website view in different languages.

In this document, we will discuss the implementation of how translator use packages as well as built-in library in supporting multiple language. In this angular translator, we will discuss step by step procedure of how to create an angular website and extract the text from template into JSON (JavaScript Object Notation). Then, we will install the necessary packages from NPM (Node Package Manager). And finally, we will explain how to serve our website in multiple languages by changing the language.

**Source Code of NGX-translator**: [**https://github.com/pavitharani/i18n-angular-demo.git**](https://github.com/pavitharani/i18n-angular-demo.git)

**Source Code of i18n library:** [**https://github.com/pavitharani/i18n\_library\_Demo.git**](https://github.com/pavitharani/i18n_library_Demo.git)

# INTERNALISATION

Internalization is the process of designing our app so that it can support various language. It is also known as i18n. Localization is the process of building versions of your project for different locales. The localization process includes the following actions.

* Extract text for translation into different languages
* Format data for a specific locale

Building a user-friendly application is every product owner’s goal; various factors enhance the user experience of an app. To entertain a user, an application must be usable, findable, credible, desirable, and valuable, but most importantly, it should be accessible to worldwide users.

Every text on the application should be localized so that anyone can access the information provided on the app. Various methods help in translating an Angular app, such as using the built-in i18n tool or by using the ngx-translate plugin

“i18n” is a numeronym where “18” represents the number of letters between the first letter (“I”) and the last letter (“N”) in the word “internationalization”.

# what is NGX-TRANSLATe? Tech Startup That Makes Mind-

* [NGX-Translate](https://github.com/ngx-translate/core) is an internationalization library for Angular.
* It allows you to Internationalize the Angular app in multiple languages.
* We can easily convert static or dynamic data into various languages.
* It provides you useful service, a directive, and a pipe to manipulate any data.
* NGX-Translate is also extremely modular.

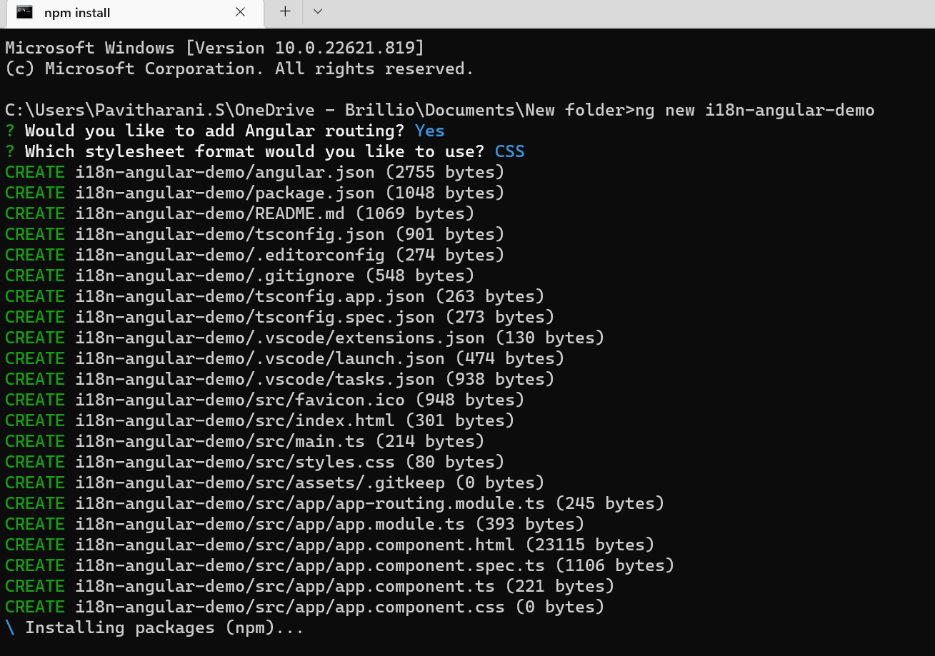
to Buy Tech

# IMPLEMENTATION with ngx-translate

## Step-1 Create an Angular Project

Use the below **ng command** to generate the project. After that, the CLI is going to deliver a prompt to you, which asks you if you would like the Angular routing addition. And that is your choice to accept/decline. After that, the CLI sends you another prompt, which will help us to decide upon a stylesheet format you would like to use. Pick the [CSS](https://flatlogic.com/blog/what-is-material-ui/) option here. The combination of these two decisions will allow Angular to generate the files and folders required and establish the needed packages from npm.

ng new i18n-angular-demo

****

After that, we need to proceed to the root folder and run the local development server using these commands:

ng serve

That will make our application available at the following address:

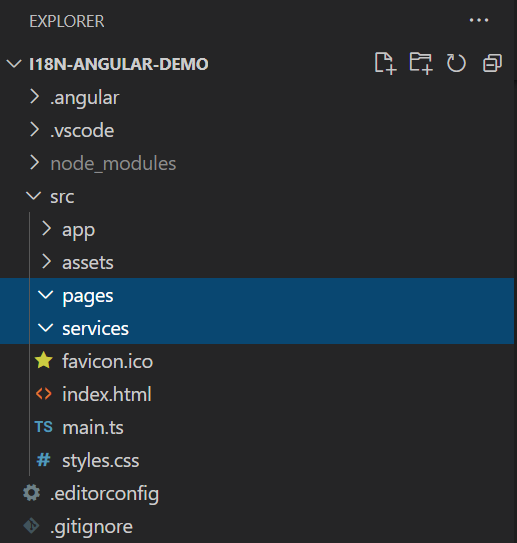
[http://localhost:4200/](http://localhost:4200/)

Now, what we need to do is to go to this address using a web browsing of our choosing and see a beautiful, yet utterly incomplete page like this:

Graphical user interface, application

Description automatically generated

## Step-2 Create Pages and Services folder

****

## Step-3 Generate Home Page

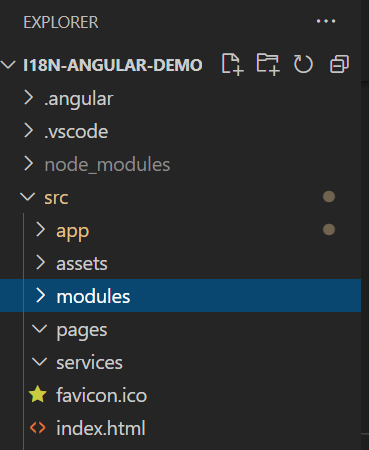
Create a home page using a ng command.

ng generate component pages/home

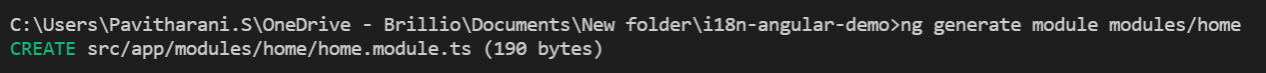
Text

Description automatically generated

Create a module folder and generate home module



ng generate modules modules/home



**File Structure**

We will find the auto generated files in the following file structure. And manually create a routes file for Home pages.

Text

Description automatically generated

## Step-4 Routing Configuration

* **home.routes.ts**  
   Create home.routes.ts inside the pages component and provide the route.

import { Route } from '@angular/router';

import { HomeComponent } from './home.component';

export const HomeRoutes: Route[] = [

  {

    path: '',

    component: HomeComponent,

  },

];

* **home.module.ts**  
  Connect routes with RouterModule.

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { HomeComponent } from '../pages/home/home.component';

import { HomeRoutes } from '../pages/home/home.routes';

import { RouterModule } from '@angular/router';

@NgModule({

  declarations: [HomeComponent],

  imports: [

    CommonModule,

    RouterModule.forChild(HomeRoutes),

  ],

})

export class HomeModule { }

* **app.component.html**  
  Clean up the default code except the following router-outlet tag.

<router-outlet></router-outlet>

* **app.module.ts**  
  Remove auto imported Home components.

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

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

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

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

@NgModule({

declarations: [AppComponent],

imports: [

HttpClientModule,

BrowserModule,

AppRoutingModule

],

providers: [],

bootstrap: [AppComponent],

})

export class AppModule {}

* **app-routing.module.ts**  
  App routes connect with Home modules. We must specify the page path name. Here, you have configured home page is default page with empty path value.

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [

{

path: '',

loadChildren: () =>

import('./modules/home.module').then((m) => m.HomeModule),

},

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

})

export class AppRoutingModule {}

* **Restart Angular Server**

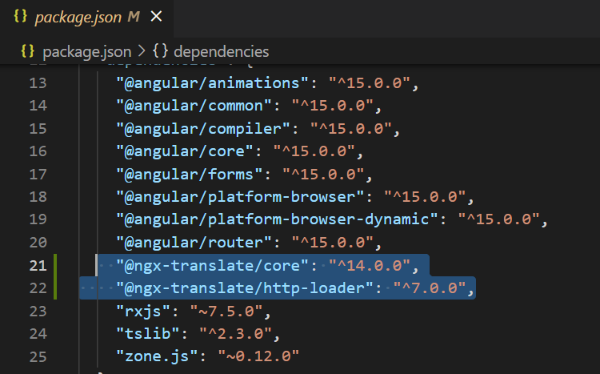
You can access http://localhost:4200/ using ng serve command

## Step-5 Install Translate Plugins

Use the below npm command to install the translate plugins.

npm install @ngx/translate/core

npm install @ngx/translate/http-loader

You will see the packages downloaded in package.json

## Step-6 Language Assets

Create language files in JSON format. Inside the assets folder, create i18n folder and create the json files. Here, we are creating en(english), es(spanish) and ta(tamil).

**Json files:**

Text

Description automatically generated with medium confidence

**en.json:**

English version language file.

{

    "home":

{

        "title": "Home page",

        "heading":"Angular Translate Demo",

        "welcome\_text": "Stay Positive better days are on the way.",

        "choose\_language": "Choose language"

    }

}

**es.json:**

Spanish version language file.

{

    "home":

{

        "title": "Página de inicio",

        "heading":"Demostración de traducción angular",

        "welcome\_text": "Manténgase positivo mejores días están en camino.",

        "choose\_language": "Elige lengua"

    }

}

**ta.json:**

Tamil version language file.

{

    "home":

{

        "title": "முகப்பு பக்கம்",

        "heading":"கோண மொழிபெயர்ப்பு டெமோ",

        "welcome\_text": "நேர்மறையாக இருங்கள் நல்ல நாட்கள் வரும்.",

        "choose\_language": "மொழியைத் தேர்வுசெய்க"

    }

}

## Step-7 Configure with i18n files

* **app.module.ts**

Import TranslateModules and configured it with language assets.

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

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { TranslateLoader, TranslateModule } from '@ngx-translate/core';

import { TranslateHttpLoader } from '@ngx-translate/http-loader';

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

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

export function rootLoaderFactory(http: HttpClient) {

  return new TranslateHttpLoader(http, 'assets/i18n/', '.json');

}

@NgModule({

  declarations: [AppComponent],

  imports: [

    HttpClientModule,

    BrowserModule,

    AppRoutingModule,

    TranslateModule.forRoot({

      loader: {

        provide: TranslateLoader,

        useFactory: rootLoaderFactory,

        deps: [HttpClient],

      },

    }),

  ],

  providers: [],

  bootstrap: [AppComponent],

})

export class AppModule {}

* **home.module.ts**  
  Import TranslateModule.

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { HomeComponent } from '../pages/home/home.component';

import { HomeRoutes } from '../pages/home/home.routes';

import { RouterModule } from '@angular/router';

import { TranslateModule } from '@ngx-translate/core';

@NgModule({

  declarations: [HomeComponent],

  imports: [

    CommonModule,

    RouterModule.forChild(HomeRoutes),

    TranslateModule,

  ],

})

export class HomeModule { }

## Step-7 Create a TranslateConfig Service

Generate a service for changing the language type.

ng generate service services/translateConfig

You will find in services directory.

Graphical user interface, text, application

Description automatically generated

**translate-config.ts**

Here the default language is constructed with English(en) language. And changeLanguage method will help you to switch with other supported languages.

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

import { TranslateService } from '@ngx-translate/core';

@Injectable({

  providedIn: 'root'

})

export class TranslateConfigService {

  constructor(private translateService:TranslateService) {

    this.translateService.use('en');

  }

  changeLanguage(type:string){

    this.translateService.use(type);

  }

}

## Step-8 Localizing Application

**home.component.html**  
Use translate pipe for HTML content with JSON content reference. Add button to change Language and by using event binding property call the method in your typescript(ts) file.

<div class="demo">

<h1 translate>home.heading</h1>

<h4 translate>home.choose\_language</h4>

<div>

    <button (click)="changeLanguage('en')">English</button>

    <button (click)="changeLanguage('es')">Spanish</button>

    <button (click)="changeLanguage('ta')">Tamil</button>

</div>

<h3 translate>home.welcome\_text</h3>

</div>

**home.component.ts**

Import translateConfig service.

import { Component, OnInit } from '@angular/core';

import { TranslateConfigService } from 'src/app/services/translate-config.service';

@Component({

  selector: 'app-home',

  templateUrl: './home.component.html',

  styleUrls: ['./home.component.css']

})

export class HomeComponent  implements OnInit{

  constructor(private translateConfigService:TranslateConfigService){}

  ngOnInit():void{

  }

  changeLanguage(type: string) {

    this.translateConfigService.changeLanguage(type);

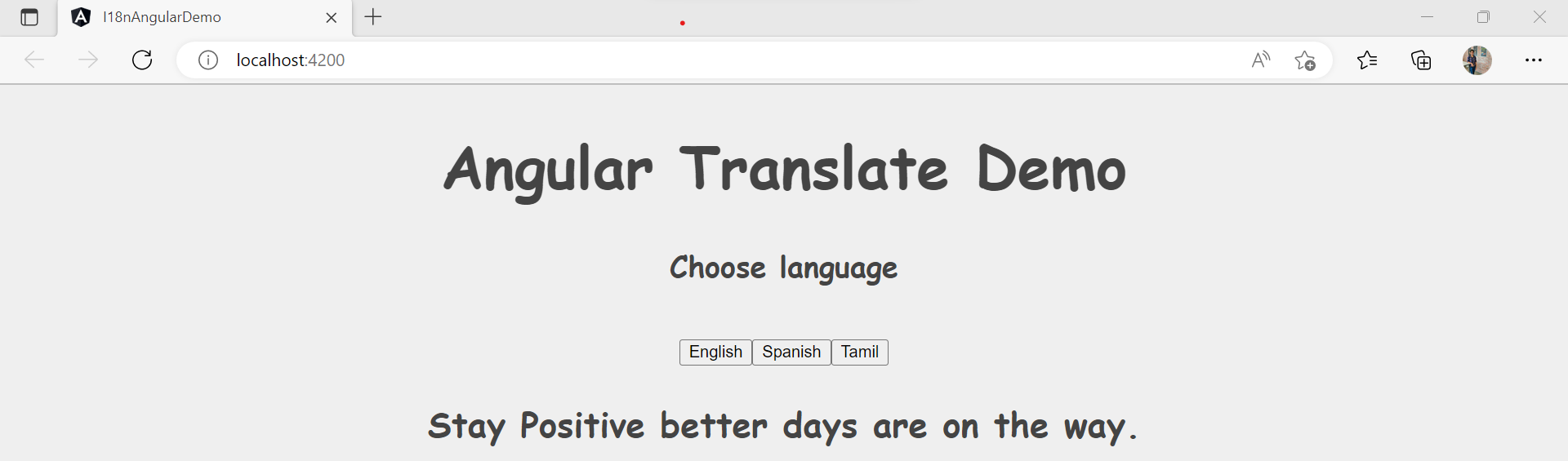
  }

}

Finally, add your own styles in your code, to see the results.

## RESULT

**English Language:**



**Spanish Language:**



**Tamil Language:**

Graphical user interface, application, Word

Description automatically generated

Thus, translate plugins help how to build a multilingual Angular app using ngx-translate plugin easily.

# Implementation with i18n-library

## Step-1: Create an angular project

Run the following command to create the app.

ng new i18nDemo

Open the **i18nDemo** app using VS code.

## Step-2: Add localize to Angular Project

To add the localization features provided by Angular, we need to add the **@angular/localise** package to our project:

ng add @angular/localise

This command:

* Updates **package.json** and installs the package.
* Updates **polyfills.ts** to import the **@angular/localize** pack

## Step-3: Setting up the app component

Open **app.component.html** file. Replace the already existing text with the following code.

<h1 i18n>

  Localization Demo in Angular using i18n

</h1>

<h3 i18n="@@myName">

  Hello, My name is Pavitharani

</h3>

<hr/>

You can observe that we have marked **<h1>** and **<h3>** tags with i18n attribute. This is a way to tell the Angular to consider this text as translatable content. We will explore i18n attribute in detail.

## Step-4: Create a translation source file

Run the following command in the CLI to create a translation source file.

ng extract i18n

It will create a **messages.xlf** file. Create translate folder inside src and place the messages.xlf inside it. Then, Open the file and you can observe the following XML code inside it.

<?xml version="1.0" encoding="UTF-8" ?>

<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">

  <file source-language="en-US" datatype="plaintext" original="ng2.template">

    <body>

      <trans-unit id="1208749786232006981" datatype="html">

        <source> Localization Demo in Angular using i18n

</source>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">1,3</context>

        </context-group>

      </trans-unit>

      <trans-unit id="myName" datatype="html">

        <source> Hello, My name is Pavitharani</source>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">5,7</context>

        </context-group>

      </trans-unit>

    </body>

  </file>

</xliff>

This file contains a list of **<trans-unit>** tags. These tags will have all the content that was marked for translation using i18n attribute. You can also observe that each **<trans-unit>** tag has an id property associated with it. This unique id will be generated by default for each tag that was marked with i18n attribute. We can also customize the id by providing a name prefixed with **@@**as we have done with **<h3>** tag in previous section. Hence, the id for **<h3>** tag is “myName” as we defined it

If you change the text for any tag in your HTML file, you need to regenerate the translation file. Regenerating the file will override the default id of **<trans-unit>** tags. Hence, it is advisable to provide custom ids to each translatable tag to maintain consistency.

Hence, we have successfully implemented i18n to our app. Next, we will extend it to make it available to different languages.

## Step-5: Translating the content

We will translate our application in two new languages apart from English, which are Spanish and Hindi. Make three copies of the messages.xlf file and rename them to **messages.en.xlf, messages.es.xlf** and **messages.hi.xlf**. These file names can be customized as per your choice but the extension should be .xlf**.**

Open **messages.es.xlf** and put the following content in it.

<?xml version="1.0" encoding="UTF-8" ?>

<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">

  <file source-language="en-US" datatype="plaintext" original="ng2.template">

    <body>

      <trans-unit id="1208749786232006981" datatype="html">

        <source> Localization Demo in Angular using i18n</source>

          <target>Demostración de localización en angular usando i18n</target>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">1,3</context>

        </context-group>

      </trans-unit>

      <trans-unit id="myName" datatype="html">

        <source> Hello, My name is Pavitharani</source>

         <target>Hola, mi nombre es Pavitharani</target>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">5,7</context>

        </context-group>

      </trans-unit>

    </body>

  </file>

</xliff>

This is the same content as the original messages.xlf file, but we have added a **<target>** tag corresponding to each **<source>** tag. The **<target>** tag contains the translated text for the content inside the **<source>** tag. Here I am using Google translate for the translation but in real time applications, a language expert will translate the contents from **messages.xlf** file.

Similarly open the **messages.hi.xlf** and put in the following content in it.

<?xml version="1.0" encoding="UTF-8" ?>

<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">

  <file source-language="en-US" datatype="plaintext" original="ng2.template">

    <body>

      <trans-unit id="1208749786232006981" datatype="html">

        <source> Localization Demo in Angular using i18n</source>

  <target>I18n का उपयोग कर कोणीय में स्थानीयकरण डेमो</target>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">1,3</context>

        </context-group>

      </trans-unit>

      <trans-unit id="myName" datatype="html">

        <source> Hello, My name is Pavitharani</source>

<target>हेलो, मेरा नाम पवित्रारानी है</target>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">5,7</context>

        </context-group>

      </trans-unit>

    </body>

  </file>

</xliff>

Finally, we will make English translation file. Open **messages.en.xlf** and put in the following content in it.

<?xml version="1.0" encoding="UTF-8" ?>

<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">

  <file source-language="en-US" datatype="plaintext" original="ng2.template">

    <body>

      <trans-unit id="1208749786232006981" datatype="html">

        <source> Localization Demo in Angular using i18n</source>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">1,3</context>

        </context-group>

      </trans-unit>

      <trans-unit id="myName" datatype="html">

        <source> Hello, My name is Ankit</source>

        <context-group purpose="location">

          <context context-type="sourcefile">src/app/app.component.html</context>

          <context context-type="linenumber">5,7</context>

        </context-group>

      </trans-unit>

    </body>

  </file>

</xliff>

## Step-6: Configure the app to serve in multiple languages

Open **angular.json** file and add the following configuration.

🡪Configure the locale and the path of translated file.

  "projects": {

    "i18nDemo": {

      "i18n":{

      "locales":{

        "hi":"src/translate/messages.hi.xlf",

        "es":"src/translate/messages.es.xlf",

        "en":"src/translate/messages.en.xlf"

      }

    },

🡪Custom locale-specific configuration using a single locale

Text

Description automatically generated

🡪Configuration for locale definition in the build configurationText

Description automatically generated

Here we have added the configuration for the specified language. We have provided the path and format for i18n file and set the locale to “es”, ”en” and “hi”. When we execute the application, app content will be served from the i18n file path provided.

## Step-7: Execution Demo

Once you have added the configuration for all the languages in angular.json file, run the following command to start the server.

ng serve –configuration=es

This will launch the application in “es” configuration and our app will show the Spanish language translations.

Refer to the output screen as shown below:

# Graphical user interface, text, application, email Description automatically generated

Similarly, we can configure for Hindi and English. Run the following command for Hindi and English.

ng serve –configuration=hi

It will open in new port number since we are already using the port 4200.

# Graphical user interface, text, application, Word Description automatically generated

ng serve –configuration=en

Graphical user interface, text, application, Word

Description automatically generated

Thus, the configurations that we have defined will run in different ports.

# Conclusion

Thus, we have implemented both the ways of internalisation(i18n). NGX-translator helps you to create a multilingual Angular App. Since, translation files are stored in JSON format, which is lightweight and easy to work with. Another way that we have implemented is by using built-in i18n module where a special tool to extract strings into translation files (where the actual translation takes place). We also looked at how to set up an environment for translating an Angular app, how to access translate service’s methods. Therefore, built-in internationalization has huge advantage when the translations are provided by a third party.