Essa semana eu me peguei precisando fazer uma coisa que nunca havia feito antes no Ionic que é portar uma aplicação para outras línguas além do português.

Dei uma pesquisada na internet e vi que o Angular tem uma forma um pouco rudimentar para lidar com isso e foi aí que eu achei uma biblioteca chamada [@ngx-translate](https://www.npmjs.com/package/@ngx-translate/core).

Essa biblioteca é muito simples de se usar, ela implementa um *service*, uma *directive* e um *pipe* que vão interagir com as configurações que você vai definir através de arquivos *json* para trazer as traduções de acordo com o que você passar como parâmetro de língua.

No **Ionic** você usa esta biblioteca aliada ao plugin **[Globalization](https://ionicframework.com/docs/native/globalization/" \t "_blank)** que vai lhe dizer qual é a língua padrão usada no dispositivo em que seu aplicativo foi instalado. E a partir desta informação a biblioteca puxa, através dos arquivos *json* pré-definidos, os textos referentes àquela língua.

É tudo muito simples de fazer, não têm complicação, são pequenas configurações. O único trabalho que você vai ter é de fazer as traduções nos arquivos *json* para cada língua que você escolher dar suporte.

**Pré-requisitos**

Para esse exemplo do artigo vamos usar ainda o **Ionic V3,** porque a versão 4 ainda tem algumas incompatibilidades com biblioteca **rxjs**.

Como a ideia do artigo é transformar um aplicativo que já existe em um aplicativo com várias línguas, não vamos começar com um aplicativo em branco, vamos supor que você já tem o aplicativo pronto, com algumas telas, com algumas páginas, e vamos a partir daí implementar a internacionalização desse aplicativo.

**Começando**

Para trabalhar no Ionic V3, nós não podemos usar as últimas versões das bibliotecas @ngx-translate, porque as últimas versões desta biblioteca são para serem usadas com a **versão 6 do Angular**, e o Ionic V3 utiliza ainda a versão 5.

https://miro.medium.com/max/30/1*g0xHv_2XUSc-W6ejlQm2xQ.png?q=20

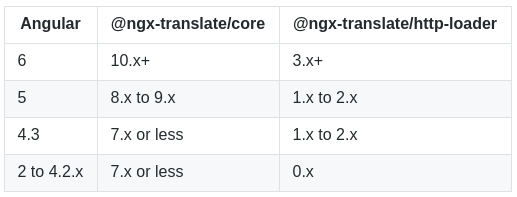


Tabela de relacionamento entre as versões, retirada do repositório oficial do @ngx-translate

Então como visto na tabela, vamos usar a versão 9.1.1 do core e a versão 2.0.1 do http-loader.

npm i --save @ngx-translate/core@9.1.1 @ngx-translate/http-loader@2.0.1

Feito isso, nós vamos abrir o nosso *app.module.ts* e vamos configurar a nossa biblioteca para ser executada ao iniciar o nosso aplicativo. É aqui também que nós vamos chamar as configurações iniciais e o plugin *Globalization*.

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

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

import {IonicApp, IonicErrorHandler, IonicModule, Platform} from 'ionic-angular';

import {SplashScreen} from '@ionic-native/splash-screen';

import {StatusBar} from '@ionic-native/status-bar';

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

import {Globalization} from "@ionic-native/globalization";

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

import {HomeModule} from "../pages/home/home.module";

@NgModule({

declarations: [

MyApp

],

imports: [

BrowserModule,

TranslateModule.forRoot(),

IonicModule.forRoot(MyApp),

HomeModule

],

bootstrap: [IonicApp],

entryComponents: [

MyApp

],

providers: [

StatusBar,

SplashScreen,

{provide: ErrorHandler, useClass: IonicErrorHandler},

Globalization

]

})

export class AppModule { }

view rawapp.module.ts hosted with ? by GitHub

Basicamente o que fizemos aqui foi adicionar à sessão imports o **TranslateModule** e à sessão providers o plugin Globalization. O TranslateModule deve ser importando usando o método forRoot() ou forChild(). Estes métodos também aceitam parâmetros, mas isso veremos mais pra frente.

Como o @ngx-translate é um módulo nativamente desenvolvido para Angular e a estrutura de pastas do Ionic V3 é diferente, vamos ter de dizer explicitamente à biblioteca onde nossos arquivos json de traduções estão. Em nosso caso, será em /assests/i18n/\*.json. E para fazermos isso vamos configurar uma função que atuará como fábrica e a incluiremos no forRoot().

Para criar essa função fábrica, usaremos o [**@ngx-translate/http-loader**](https://www.npmjs.com/package/@ngx-translate/http-loader), pois carregaremos nossos arquivos json de traduções seletivamente via http. Também importamos o HttpClient (se seu aplicativo já não o estiver importando) que é usado para carregar estes arquivos.

Em seguida, passamos essa fábrica como parâmetro para o TranslateModule.forRoot(), bem como o HttpClient como a dependência que será injetado na função fábrica.

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

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

import {IonicApp, IonicErrorHandler, IonicModule, Platform} from 'ionic-angular';

import {SplashScreen} from '@ionic-native/splash-screen';

import {StatusBar} from '@ionic-native/status-bar';

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

import {Globalization} from "@ionic-native/globalization";

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

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

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

import {HomeModule} from "../pages/home/home.module";

export function createTranslateLoader(http: HttpClient) {

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

}

@NgModule({

declarations: [

MyApp

],

imports: [

BrowserModule,

HttpClientModule,

TranslateModule.forRoot({

loader: {

provide: TranslateLoader,

useFactory: (createTranslateLoader),

deps: [HttpClient]

}

}),

IonicModule.forRoot(MyApp),

HomeModule

],

bootstrap: [IonicApp],

entryComponents: [

MyApp

],

providers: [

StatusBar,

SplashScreen,

{provide: ErrorHandler, useClass: IonicErrorHandler},

Globalization

]

})

export class AppModule { }

view rawapp.module.ts hosted with ? by GitHub

Por fim (das configurações), vamos fazer o plugin Globalization trabalhar nos dizendo qual a linguagem padrão definida no dispositivo.

Mas antes fazemos um teste para saber se nossa aplicação está rodando como aplicativo mesmo ou num browser (como PWA, por exemplo). Testando a presença da biblioteca cordova, descobriremos se é um aplicativo ou se está rodando no browser. A ausência do cordova indica que estamos num browser, então usamos a língua default ou consultamos a própria biblioteca Translate para tentar identificar; caso o cordova esteja presente, então usaremos o Globalization para isso.

Tendo posse da língua selecionada, usaremos então o translate.use() para definirmos qual língua a biblioteca usará como base. É tudo bem transparente, nesse momento não há muito o que se explicar desse código.

O que acontece em seguida é que a biblioteca, tomando como base o código da linguagem escolhida, vai até a pasta e i18n que nós definimos na configuração, e tenta localizar um arquivo com mesmo nome do código.

Nós também definimos a opção translate.setDefaultLanguage(), indicando para a biblioteca que caso não exista um arquivo com o código selecionado, então a biblioteca pode exibir o resultado com a linguagem padrão, que definimos numa constante logo no começo no código.

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

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

import {IonicApp, IonicErrorHandler, IonicModule, Platform} from 'ionic-angular';

import {SplashScreen} from '@ionic-native/splash-screen';

import {StatusBar} from '@ionic-native/status-bar';

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

import {Globalization} from "@ionic-native/globalization";

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

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

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

import {HomeModule} from "../pages/home/home.module";

export function createTranslateLoader(http: HttpClient) {

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

}

export const DEFAULT\_LANGUAGE = 'en-us';

@NgModule({

declarations: [

MyApp

],

imports: [

BrowserModule,

HttpClientModule,

TranslateModule.forRoot({

loader: {

provide: TranslateLoader,

useFactory: (createTranslateLoader),

deps: [HttpClient]

}

}),

IonicModule.forRoot(MyApp),

HomeModule

],

bootstrap: [IonicApp],

entryComponents: [

MyApp

],

providers: [

StatusBar,

SplashScreen,

{provide: ErrorHandler, useClass: IonicErrorHandler},

Globalization

]

})

export class AppModule {

constructor(platform: Platform, translate: TranslateService, private globalization: Globalization) {

platform.ready().then( () => {

translate.setDefaultLang(DEFAULT\_LANGUAGE);

if ((<any>window).cordova) {

this.globalization.getPreferredLanguage().then( result => {

translate.use(result.value.toLowerCase());

});

} else {

let browserLanguage = translate.getBrowserLang() || DEFAULT\_LANGUAGE;

translate.use(browserLanguage.toLowerCase());

}

});

}

}

app.module.ts hosted with ? by GitHub

## Criando os arquivos de tradução

Com a configuração concluída, chegou a hora de trabalharmos em cima dos arquivos de tradução, que são basicamente os arquivos json, que devem ser nomeados de acordo com os códigos das linguagens que o aplicativo dará suporte.

Digamos que nós queremos que nosso aplicativo dê suporte as línguas: português do Brasil, inglês dos Estados Unidos, inglês da grã-bretanha e espanhol da Espanha, então nós teríamos que ter os seguintes arquivos na nossa pasta i18n:

* pt-br.json
* en-us.json
* en-gb.json
* es-es.json

Como exemplo vou exibir somente aqui o arquivo português do Brasil e inglês dos Estados Unidos:

[**en-us.json**](https://gist.github.com/wgbn/79f32f343b7c2719f278ce8a6e30d8f8#file-en-us-json)

|  |
| --- |
| { |
|  | "LAYOUT\_APP\_NOME": "Obesity Calculator", |
|  | "HOME\_PRIMEIRO\_PASSO": "First we need to calculate your BMI", |
|  | "HOME\_LABEL\_PESO": "Your weight (kg)", |
|  | "HOME\_LABEL\_ALTURA": "Your height (m)", |
|  | "HOME\_BUTTON\_CALCULAR": "Calculate" |
|  | } |

[**pt-br.json**](https://gist.github.com/wgbn/79f32f343b7c2719f278ce8a6e30d8f8#file-pt-br-json)

|  |
| --- |
| { |
|  | "LAYOUT\_APP\_NOME": "cálculo de obesidade", |
|  | "HOME\_PRIMEIRO\_PASSO": "Primeiro precisamos calcular seu IMC", |
|  | "HOME\_LABEL\_PESO": "Seu peso (kg)", |
|  | "HOME\_LABEL\_ALTURA": "Seu altura (m)", |
|  | "HOME\_BUTTON\_CALCULAR": "Calcular" |
|  | } |

Os arquivos de tradução são extremamente simples também, basicamente são campos chave = valor, onde a chave é o nome do item e o valor é a tradução. Eu preferi padronizar os nomes das chaves colocando todos em maiúsculas e separados por underline como se fossem constantes, mas isso foi uma escolha minha e no seu aplicativo você pode fazer do jeito que você achar melhor. O importante é que cada arquivo possuam exatamente as mesmas chaves.

## Aplicando a tradução dos templates

Agora que nós já temos a configuração necessária e os arquivos de tradução, vamos ver como aplicar essas traduções no template HTML do nosso aplicativo.

Para isso nós vamos usar o pipe translate fornecido pela biblioteca:

home.html

<div class="home-page">

<form novalidate #imc="ngForm" (ngSubmit)="calcularImc(imc.value)">

<p>{{ 'HOME\_PRIMEIRO\_PASSO' | translate }}</p>

<p>

<label for="peso">{{ 'HOME\_LABEL\_PESO' | translate }}</label>

<input type="tel" id="peso" name="peso" [(ngModel)]="peso" required>

</p>

<p>

<label for="altura">{{ 'HOME\_LABEL\_ALTURA' | translate }}</label>

<input type="tel" id="altura" name="altura" [(ngModel)]="altura" required>

</p>

<p>

<button type="submit" botao-padrao [disabled]="!imc.form.valid">

<span \*ngIf="!showLoad else load">{{ 'HOME\_BUTTON\_CALCULAR' | translate }}</span>

<ng-template #load>

<i class="fa fa-spin fa-circle-o-notch"></i>

</ng-template>

</button>

</p>

</form>

</div>

Como é possível ver no código acima, é muito simples aplicar tradução. Como qualquer outro *pipe* no Angular, você passa um dado para o *pipe* e ele retorna um resultado formatando aquele dado. No nosso caso, nós passamos como dado o nome da chave que queremos acessar no arquivo de tradução e automaticamente a biblioteca busca o conteúdo da chave no arquivo de tradução correto, dentre todos os arquivos de tradução que criamos.

Além do *pipe*, a biblioteca também nos disponibiliza uma **directive** e um **service** para que possamos traduzir alguns termos diretamente no nosso código *typescript*, pois nem sempre usaremos a tradução diretamente no template.

Para este tutorial eu não precisei usar o *service* nem a *directive*, mas vou deixar abaixo um exemplo da documentação da biblioteca ensinando como usá-los:

<div [translate]="'HELLO'"></div>

view rawdirective-no-template.html hosted with ? by GitHub

translate.get('HELLO').subscribe((res: string) => {

console.log(res);

//=> 'hello world'

});

view rawservice-no-typescript.ts hosted with ? by GitHub

É possível também, no momento a tradução de um termo, definir um parâmetro que será injetado nesta tradução, mas essa opção vou deixar para que vocês pesquisem na documentação oficial da biblioteca, é tão simples como os exemplos mostrados anteriormente.