UD10.11 – Laravel
Aplicación multiidioma

2º CFGS
Desarrollo de Aplicaciones Web

2022-23

1.- Aplicaciones multiidioma

Laravel también ofrece un sistema para poder implementar una aplicación multiidioma de una manera muy sencilla.

Se ofrecen dos maneras para manejar las cadenas con las traducciones:

- Archivos **php** que devuelven un array asociativo clave → traducción.
- Archivos **json** con pares clave → traducción.

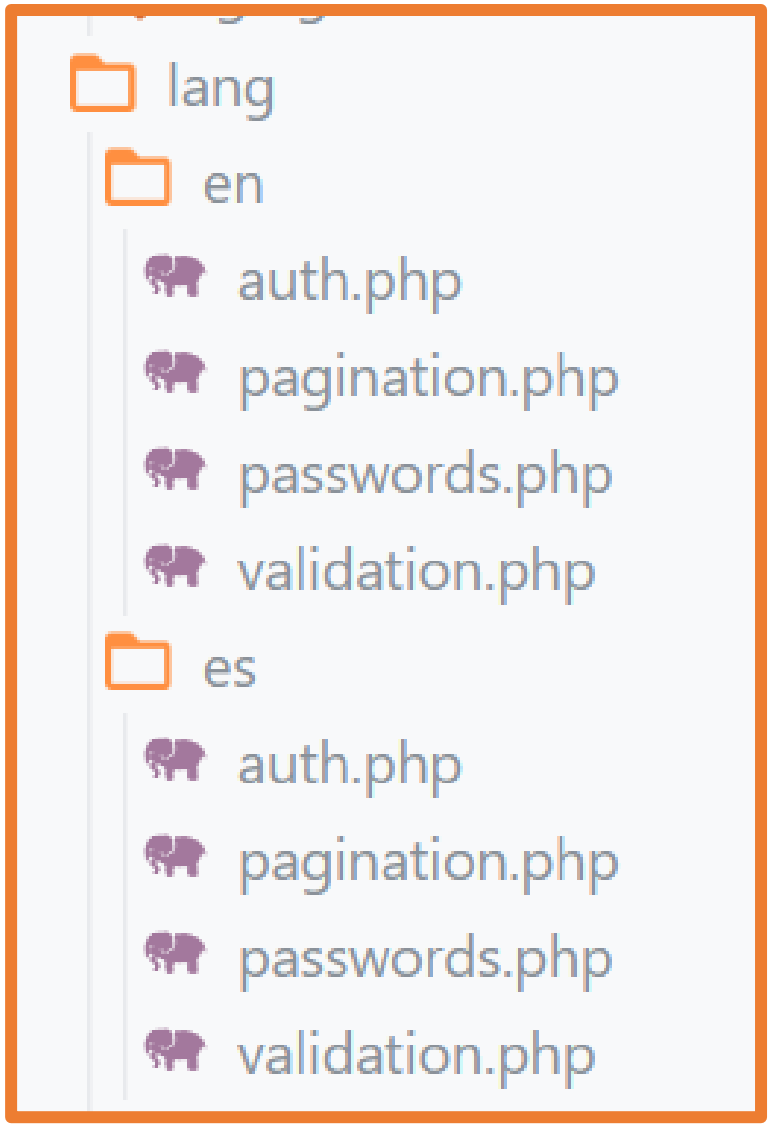
Se pueden usar los dos métodos indistintamente e incluso de manera simultánea, pero se debe tener en cuenta que no debe haber claves que coincidan con nombres de archivos de traducción.

2.- Archivos PHP

Archivos **php** que devuelven un array asociativo clave → traducción

Cada archivo php de traducción se debe ubicar en la carpeta **lang** dentro de una carpeta con el código de 2 letras del idioma al que representan.

Para cada idioma puede haber varios archivos de traducción de manera que mantendrá una organización en las traducciones.



2.- Archivos PHP

Todos los archivos de idiomas deben contener el mismo array en el que solo cambiarán las traducciones según el idioma.

lang/**en**/pagination.php

lang/**es**/pagination.php

3.- Archivos JSON

Archivos **json** con pares clave → traducción

Cada archivo json de traducción se debe ubicar directamente en la carpeta **Lang**.

Para cada idioma solo debe haber un archivo de traducción.

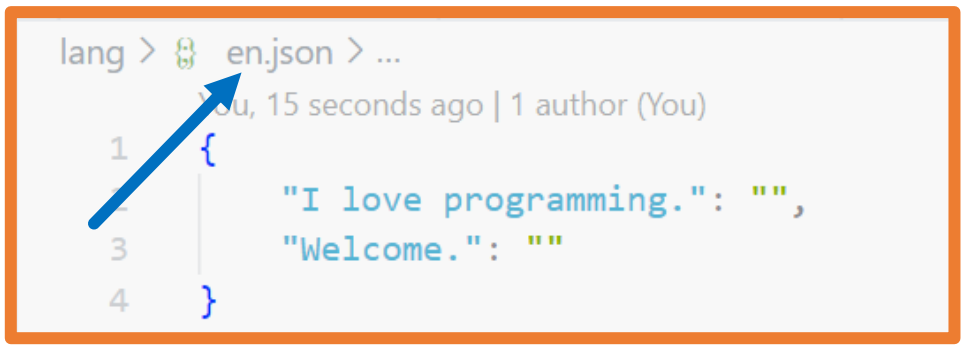
Cada archivo se debe llamar con las dos letras del idioma que representan y tener la extensión .json.

3.- Archivos JSON

Como solo existe un archivo json por cada idioma si se utiliza una "clave corta" para cada traducción las traducciones pueden ser difíciles de gestionar.

Por esta razón se recomienda elegir un **idioma base** y que en él las claves sean las traducciones en ese idioma y dejar las traducciones en blanco.

Así si se tienen que cargar las traducciones del idioma base Laravel automáticamente cargará las claves y si se tienen que cargar las traducciones de otro idioma se cargará la traducción.



4.- Uso de traducciones

Para usar las traducciones se usa el operador **_ _('*clave*')**

Si se están usando archivos PHP la clave debe ir precedida del nombre del archivo donde se encuentra:

```
{{__('pagination.previous')}}
<br>
{{__('pagination.next')}}
```

Si se están usando archivos JSON simplemente se debe poner la clave del idioma base.

```
{{__('Welcome.')}}
<br>
{{__('I love programming.')}}
```

5.- Parámetros en las traducciones

Si es necesario se pueden añadir parámetros en las traducciones.

Para ello se utilizan los dos puntos **:** seguidos del **nombre** que se le da al parámetro:

```
{
    "I love programming.": "Me encanta programar.",
    "Welcome": "Bienvenido :Name"
}
```

En las plantillas **Blade** para pasar los parámetros a la traducción se utiliza un array asociativo donde la clave es el nombre que tiene el parámetro en el archivo de traducción:

```
{{__('Welcome', ['name' => 'rick'])}}
```

5.- Parámetros en las traducciones

Si se utilizan archivos **PHP** el uso de parámetros es similar:

lang > es > auth.php

You, 8 seconds ago | 1 author (You)

```php
<?php

return [

    'usuario' => 'Usuario :Name',

];
```

En la plantilla **Blade**:

```
{{__('auth.usuario', ['name' => 'rick'])}}
```

5.- Parámetros en las traducciones

Dependiendo de cómo se indique el nombre del parámetro en el archivo de traducción se mostrará el valor pasado a la traducción.

Existen tres opciones:

```
"Welcome": "Bienvenido :Name",
```
→ Bienvenido Rick

```
"Welcome": "Bienvenido :name",
```
→ Bienvenido rick

```
"Welcome": "Bienvenido :NAME",
```
→ Bienvenido RICK

6.- Plural en las traducciones

Si es necesario se pueden definir traducciones que permitan el uso de nombres en plural, para ello se utiliza el carácter **|** como separador. Se pueden indicar el singular y el plural o bien crear reglas más complejas.

Para utilizar la traducción se utiliza la función **trans_choice()** a la que se le pasan la clave y la cantidad.

En este caso, por la manera de definir y usar, es recomendable usarla con archivos PHP.

Archivos PHP:

lang/es.php

```php
<?php

return [

    'producto' => "Hay un producto|Hay muchos productos",
    'cantidad' => "{0} No hay|[1,5] Hay algunos|[6,*] Hay muchos",

];
```

en la vista

```
{{trans_choice('plural.producto', 10)}}
{{trans_choice('plural.cantidad', 3)}}
```

6.- Plural en las traducciones

En este caso, por la manera de definir y usar, es recomendable usarla con archivos PHP.

Archivos JSON:

lang/es.json

```
{
    "I love programming.": "Me encanta programar.",
    "Welcome": "Bienvenido :Name",

    "There is one product|There are many products": "Hay un producto|Hay muchos productos",
    "{0} There are none|[1,5] There are some|[6,*] There are many": "{0} No hay|[1,5] Hay algunos|[6,*] Hay muchos"
}
```

En la vista

```
{{trans_choice('There is one product|There are many products', 2)}}
{{trans_choice('{0} There are none|[1,5] There are some|[6,*] There are many', 10)}}
```

7.- Configurar idioma por defecto en Laravel

En la configuración de Laravel se puede indicar el idioma principal de la aplicación web.

En el archivo **config/app.php** se encuentran las siguientes opciones:

```
'locale' => 'en',
```

Idioma principal de la aplicación.

```
'fallback_locale' => 'en',
```

Idioma de respaldo por si no existe traducción para el idioma principal.

8.- Configurar idioma en tiempo de ejecución.

Configurando la opción **'locale'** todas las personas que accedan a la aplicación web verán esa traducción.

Para permitir que los usuarios puedan elegir entre los diferentes idiomas disponibles se pueden utilizar las mismas técnicas vistas previamente en la unidad **"UD7 Localización e internacionalización"**:

- Obtener mediante PHP el idioma del navegador:
  $_SERVER['HTTP_ACCEPT_LANGUAGE']
- Ofrecer al usuario un seleccionable con los idiomas disponibles.
- Obtener de la base de datos la opción del usuario guardada previamente.

8.- Configurar idioma en tiempo de ejecución.

Una vez que se tiene el idioma deseado, se debe almacenar, por ejemplo, en una variable de sesión para así tenerlo disponible desde cualquier punto de la aplicación.

Para crear/consultar variables de sesión se puede utilizar:

- La variable súper global $_SESSION vista en la "UD6 Sesiones y seguridad"
- La función **session()** que ofrece Laravel

8.- Configurar idioma en tiempo de ejecución.

Uso de la función **session()**

Crear variable de sesión con la función **session()**:

```php
session(['idioma' => 'es']);
```

Obtener el valor de una variable de sesión con la función **session()**:

```php
session('idioma');
```

Borrar una variable de sesión con la función **session()**:

```php
session()->forget('idioma')
```

Comprobar si una variable de sesión existe:

en PHP

```php
if (session()->has('idioma')) {

}
```

en Blade

```
@if(session()->has('idioma'))
    {{session('idioma')}}
@endif
```

8.- Configurar idioma en tiempo de ejecución.

La mejor manera de aplicar el idioma en las páginas web de la aplicación es utilizando un **middleware** que haciendo uso de una variable de sesión con el idioma cambie en tiempo de ejecución el idioma configurado.

```
php artis1an make:middleware Translations
```

El comando anterior creará el archivo **app\Http\Middleware\Translations.php** y en la función **handle** que incorpora se debe añadir el código que se ejecutará:

```php
public function handle(Request $request, Closure $next)
{
    if (session()->has('idioma')) {
        App::setLocale(session()->get('idioma'));
    }
    return $next($request);
}
```

Deberás añadir en la parte superior la línea:

```
use Illuminate\Support\Facades\App;
```

8.- Configurar idioma en tiempo de ejecución.

Por último se debe **registrar el middleware en la aplicación**.

Para ello en el archivo **app\Http\Kernel.php** se debe:

Añadir dentro de la propiedad **$middlewareGroups** la siguiente línea

```
\App\Http\Middleware\Translations::class,
```

Añadir dentro de la propiedad **$routeMiddleware** la siguiente línea

```
'translate' => \App\Http\Middleware\Translations::class,
```