How to Build Multi-Language (i18n) Support in Laravel

How to Build Multi-Language (i18n) Support in Laravel 12

Modern applications often need to serve users in multiple languages. Laravel 12 makes it easy to implement **multi-language (i18n) support** with built-in localization features. In this article, you'll configure locales, create translation files, switch languages from controllers and middleware, and add a simple language switcher UI for your Blade views.

Configure Locales in Laravel

Set the default application language and add a fallback language in your `.env` file:

```
APP_LOCALE=en
APP_FALLBACK_LOCALE=enCode language: Bash (bash)
```

`APP_LOCALE` defines the default language (English here). `APP_FALLBACK_LOCALE` is used if a translation is missing in the chosen language.

Create Translation Files

Laravel stores translations under the `lang` directory. Each locale has its own folder. For example, create `lang/en/messages.php` and `lang/es/messages.php`:

```
// lang/en/messages.php
return [
    'welcome' => 'Welcome to our application!',
    'login' => 'Login',
    'register' => 'Register',
];Code language: PHP (php)
```

This is the English version. Now create the Spanish version:

```
// lang/es/messages.php
return [
    'welcome' => '¡Bienvenido a nuestra aplicación!',
    'login' => 'Iniciar sesión',
    'register' => 'Registrarse',
];Code language: PHP (php)
```

Each key should exist in all languages. Laravel automatically looks up the key in the current locale's file.

Using Translations in Blade Templates

Use the `__()` helper or `@lang` directive to fetch translations inside your views:

```
<h1>{{ __('messages.welcome') }}</h1>

<a href="/login">@lang('messages.login')</a> |
<a href="/register">@lang('messages.register')</a>Code language: PHP (php)
```

When the locale is `en`, this outputs English. When `es`, it displays Spanish text. This keeps Blade templates clean and reusable.

Switching Locales in a Controller

You can set the current locale dynamically in a controller action, for example, when a user chooses a language preference:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\App;

class LanguageController extends Controller
{
    public function switch($locale)
    {
        if (in_array($locale, ['en', 'es'])) {
            App::setLocale($locale);
            session(['locale' => $locale]);
        }

        return redirect()->back();
    }
}Code language: PHP (php)
```

The app's locale is updated and stored in the session. Add a route like `/lang/{locale}` to switch languages dynamically.

Persisting Locale with Middleware

Create a middleware that applies the user's preferred locale on every request:

```
php artisan make:middleware SetLocaleCode language: Bash (bash)
```

Then implement it like this:

```
namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\App;

class SetLocale
{
    public function handle($request, Closure $next)
    {
        $locale = session('locale', config('app.locale'));
        App::setLocale($locale);

        return $next($request);
    }
}Code language: PHP (php)
```

Register this middleware in `app/Http/Kernel.php`, and Laravel will automatically use the stored session locale for every request.

Building a Language Switcher UI

Add a simple dropdown in your Blade template to let users pick a language:

```
<form action="/lang" method="GET">
    <select name="locale" onchange="this.form.submit()">
```

```
        <option value="en" {{ app()->getLocale() == 'en' ? 'selected'
: '' }}>English</option>
        <option value="es" {{ app()->getLocale() == 'es' ? 'selected'
: '' }}>Español</option>
    </select>
</form>Code language: PHP (php)
```

This dropdown reloads the page with the chosen locale applied. You can enhance it with Bootstrap or Tailwind to match your design system.

Using Events for Language Switching

For audit logs or analytics, trigger an event whenever the user switches their language:

```
// app/Events/LanguageChanged.php
namespace App\Events;

use Illuminate\Foundation\Events\Dispatchable;

class LanguageChanged
{
    use Dispatchable;

    public $locale;

    public function __construct($locale)
    {
        $this->locale = $locale;
    }
}Code language: PHP (php)
```

Dispatch the event from your controller when switching locales, and attach listeners to record logs or update persistent user settings.

Wrapping Up

In this guide, you configured Laravel 12 for multiple languages, created translation files, used translations in Blade, switched locales via controllers and middleware, built a dropdown UI, and fired events for language changes. Multi-language support makes your app accessible to global users and improves overall UX.

FAQ: Multi-Language (i18n) in Laravel

- **How do I change the default language in Laravel?**
  Update `APP_LOCALE` in your `.env` file.
- **Can I use JSON files for translations?**
  Yes. Laravel supports `lang/{locale}.json` files for single-line key/value translations.
- **How do I remember the user's preferred language?**
  Store the selected locale in the session (via middleware) or in the database tied to the user profile.

1v0 Ship v1.0 Faster

What’s Next

Continue improving your app’s UX and SEO with these related guides:

- Creating a User-Friendly Roles & Permissions UI in Laravel
- Mastering Validation Rules in Laravel 12
- Laravel SEO Guide: Optimizing Meta, Slugs, and Sitemaps