Starter Kits

Introduction

To give you a head start building your new Laravel application, we are happy to offer application starter kits. These starter kits give you a head start on building your next Laravel application, and include the routes, controllers, and views you need to register and authenticate your application's users.

While you are welcome to use these starter kits, they are not required. You are free to build your own application from the ground up by simply installing a fresh copy of Laravel. Either way, we know you will build something great!

Creating an Application Using a Starter Kit

To create a new Laravel application using one of our starter kits, you should first install PHP and the Laravel CLI tool. If you already have PHP and Composer installed, you may install the Laravel installer CLI tool via Composer:

```
composer global require laravel/installer
```

Then, create a new Laravel application using the Laravel installer CLI. The Laravel installer will prompt you to select your preferred starter kit:

```
laravel new my-app
```

After creating your Laravel application, you only need to install its frontend dependencies via NPM and start the Laravel development server:

```
cd my-app
npm install && npm run build
composer run dev
```

Once you have started the Laravel development server, your application will be accessible in your web browser at http://localhost:8000.

Available Starter Kits

React

Our React starter kit provides a robust, modern starting point for building Laravel applications with a React frontend using Inertia.

Inertia allows you to build modern, single-page React applications using classic server-side routing and controllers. This lets you enjoy the frontend power of React combined with the incredible backend productivity of Laravel and lightning-fast Vite compilation.

The React starter kit utilizes React 19, TypeScript, Tailwind, and the shadcn/ui component library.

Vue

Our Vue starter kit provides a great starting point for building Laravel applications with a Vue frontend using Inertia.

Inertia allows you to build modern, single-page Vue applications using classic server-side routing and controllers. This lets you enjoy the frontend power of Vue combined with the incredible backend productivity of Laravel and lightning-fast Vite compilation.

The Vue starter kit utilizes the Vue Composition API, TypeScript, Tailwind, and the shadcn-vue component library.

Livewire

Our Livewire starter kit provides the perfect starting point for building Laravel applications with a Laravel Livewire frontend.

Livewire is a powerful way of building dynamic, reactive, frontend UIs using just PHP. It's a great fit for teams that primarily use Blade templates and are looking for a simpler alternative to JavaScript-driven SPA frameworks like React and Vue.

The Livewire starter kit utilizes Livewire, Tailwind, and the Flux UI component library.

Starter Kit Customization

React

Our React starter kit is built with Inertia 2, React 19, Tailwind 4, and shadcn/ui. As with all of our starter kits, all of the backend and frontend code exists within your application to allow for full customization.

The majority of the frontend code is located in the `resources/js` directory. You are free to modify any of the code to customize the appearance and behavior of your application:

```
resources/js/
├── components/    # Reusable React components
├── hooks/         # React hooks
├── layouts/       # Application layouts
├── lib/           # Utility functions and configuration
├── pages/         # Page components
└── types/         # TypeScript definitions
```

To publish additional shadcn components, first find the component you want to publish. Then, publish the component using `npx`:

```
npx shadcn@latest add switch
```

In this example, the command will publish the Switch component to `resources/js/components/ui/switch.tsx`. Once the component has been published, you can use it in any of your pages:

```
import { Switch } from "@/components/ui/switch"

const MyPage = () => {
  return (
    <div>
      <Switch />
    </div>
  );
};

export default MyPage;
```

Available Layouts

The React starter kit includes two different primary layouts for you to choose from: a "sidebar" layout and a "header" layout. The sidebar layout is the default, but you can switch to the header layout by modifying the layout that is imported at the top of your application's `resources/js/layouts/app-layout.tsx` file:

```
import AppLayoutTemplate from '@/layouts/app/app-sidebar-layout'; // [tl! remove]
import AppLayoutTemplate from '@/layouts/app/app-header-layout'; // [tl! add]
```

Sidebar Variants

The sidebar layout includes three different variants: the default sidebar variant, the "inset" variant, and the "floating" variant. You may choose the variant you like best by modifying the `resources/js/components/app-sidebar.tsx` component:

```
<Sidebar collapsible="icon" variant="sidebar"> [tl! remove]
<Sidebar collapsible="icon" variant="inset"> [tl! add]
```

Authentication Page Layout Variants

The authentication pages included with the React starter kit, such as the login page and registration page, also offer three different layout variants: "simple", "card", and "split".

To change your authentication layout, modify the layout that is imported at the top of your application's `resources/js/layouts/auth-layout.tsx` file:

```
import AuthLayoutTemplate from '@/layouts/auth/auth-simple-layout'; // [tl!
remove]
import AuthLayoutTemplate from '@/layouts/auth/auth-split-layout'; // [tl! add]
```

Vue

Our Vue starter kit is built with Inertia 2, Vue 3 Composition API, Tailwind, and shadcn-vue. As with all of our starter kits, all of the backend and frontend code exists within your application to allow for full customization.

The majority of the frontend code is located in the `resources/js` directory. You are free to modify any of the code to customize the appearance and behavior of your application:

```
resources/js/
├── components/    # Reusable Vue components
├── composables/   # Vue composables / hooks
├── layouts/       # Application layouts
├── lib/           # Utility functions and configuration
├── pages/         # Page components
└── types/         # TypeScript definitions
```

To publish additional shadcn-vue components, first find the component you want to publish. Then, publish the component using `npx`:

```
npx shadcn-vue@latest add switch
```

In this example, the command will publish the Switch component to `resources/js/components/ui/Switch.vue`. Once the component has been published, you can use it in any of your pages:

```
<script setup lang="ts">
import { Switch } from '@/Components/ui/switch'
</script>

<template>
    <div>
        <Switch />
    </div>
</template>
```

Available Layouts

The Vue starter kit includes two different primary layouts for you to choose from: a "sidebar" layout and a "header" layout. The sidebar layout is the default, but you can switch to the header layout by modifying the layout that is imported at the top of your application's `resources/js/layouts/AppLayout.vue` file:

```
import AppLayout from '@/layouts/app/AppSidebarLayout.vue'; // [tl! remove]
import AppLayout from '@/layouts/app/AppHeaderLayout.vue'; // [tl! add]
```

Sidebar Variants

The sidebar layout includes three different variants: the default sidebar variant, the "inset" variant, and the "floating" variant. You may choose the variant you like best by modifying the `resources/js/components/AppSidebar.vue` component:

```
<Sidebar collapsible="icon" variant="sidebar"> [tl! remove]
<Sidebar collapsible="icon" variant="inset"> [tl! add]
```

Authentication Page Layout Variants

The authentication pages included with the Vue starter kit, such as the login page and registration page, also offer three different layout variants: "simple", "card", and "split".

To change your authentication layout, modify the layout that is imported at the top of your application's `resources/js/layouts/AuthLayout.vue` file:

```
import AuthLayout from '@/layouts/auth/AuthSimpleLayout.vue'; // [tl! remove]
import AuthLayout from '@/layouts/auth/AuthSplitLayout.vue'; // [tl! add]
```

Livewire

Our Livewire starter kit is built with Livewire 3, Tailwind, and Flux UI. As with all of our starter kits, all of the backend and frontend code exists within your application to allow for full customization.

Livewire and Volt

The majority of the frontend code is located in the `resources/views` directory. You are free to modify any of the code to customize the appearance and behavior of your application:

```
resources/views
├── components            # Reusable Livewire components
├── flux                  # Customized Flux components
├── livewire              # Livewire pages
├── partials              # Reusable Blade partials
├── dashboard.blade.php   # Authenticated user dashboard
├── welcome.blade.php     # Guest user welcome page
```

Traditional Livewire Components

The frontend code is located in the `resouces/views` directory, while the `app/Livewire` directory contains the corresponding backend logic for the Livewire components.

Available Layouts

The Livewire starter kit includes two different primary layouts for you to choose from: a "sidebar" layout and a "header" layout. The sidebar layout is the default, but you can switch to the header layout by modifying the layout that is used by your application's `resources/views/components/layouts/app.blade.php` file. In addition, you should add the `container` attribute to the main Flux component:

```
<x-layouts.app.header>
    <flux:main container>
        {{ $slot }}
    </flux:main>
</x-layouts.app.header>
```

Authentication Page Layout Variants

The authentication pages included with the Livewire starter kit, such as the login page and registration page, also offer three different layout variants: "simple", "card", and "split".

To change your authentication layout, modify the layout that is used by your application's `resources/views/components/layouts/auth.blade.php` file:

```
<x-layouts.auth.split>
    {{ $slot }}
</x-layouts.auth.split>
```

WorkOS AuthKit Authentication

By default, the React, Vue, and Livewire starter kits all utilize Laravel's built-in authentication system to offer login, registration, password reset, email verification, and more. In addition, we also offer a WorkOS AuthKit powered variant of each starter kit that offers:

- Social authentication (Google, Microsoft, GitHub, and Apple)
- Passkey authentication
- Email based "Magic Auth"
- SSO

Using WorkOS as your authentication provider requires a WorkOS account. WorkOS offers free authentication for applications up to 1 million monthly active users.

To use WorkOS AuthKit as your application's authentication provider, select the WorkOS option when creating your new starter kit powered application via `laravel new`.

Configuring Your WorkOS Starter Kit

After creating a new application using a WorkOS powered starter kit, you should set the `WORKOS_CLIENT_ID`, `WORKOS_API_KEY`, and `WORKOS_REDIRECT_URL` environment variables in your application's `.env` file. These variables should match the values provided to you in the WorkOS dashboard for your application:

```
WORKOS_CLIENT_ID=your-client-id
WORKOS_API_KEY=your-api-key
WORKOS_REDIRECT_URL="${APP_URL}/authenticate"
```

Additionally, you should configure the application homepage URL in your WorkOS dashboard. This URL is where users will be redirected after they log out of your application.

Configuring AuthKit Authentication Methods

When using a WorkOS powered starter kit, we recommend that you disable "Email + Password" authentication within your application's WorkOS AuthKit configuration settings, allowing users to only authenticate via social authentication providers, passkeys, "Magic Auth", and SSO. This allows your application to totally avoid handling user passwords.

Configuring AuthKit Session Timeouts

In addition, we recommend that you configure your WorkOS AuthKit session inactivity timeout to match your Laravel application's configured session timeout threshold, which is typically two hours.

Inertia SSR

The React and Vue starter kits are compatible with Inertia's server-side rendering capabilities. To build an Inertia SSR compatible bundle for your application, run the `build:ssr` command:

```
npm run build:ssr
```

For convenience, a `composer dev:ssr` command is also available. This command will start the Laravel development server and Inertia SSR server after building an SSR compatible bundle for your application, allowing you to test your application locally using Inertia's server-side rendering engine:

```
composer dev:ssr
```

Community Maintained Starter Kits

When creating a new Laravel application using the Laravel installer, you may provide any community maintained starter kit available on Packagist to the `--using` flag:

```
laravel new my-app --using=example/starter-kit
```

Creating Starter Kits

To ensure your starter kit is available to others, you will need to publish it to Packagist. Your starter kit should define its required environment variables in its `.env.example` file, and any necessary post-installation commands should be listed in the `post-create-project-cmd` array of the starter kit's `composer.json` file.

Frequently Asked Questions

How do I upgrade?

Every starter kit gives you a solid starting point for your next application. With full ownership of the code, you can tweak, customize, and build your application exactly as you envision. However, there is no need to update the starter kit itself.

How do I enable email verification?

Email verification can be added by uncommenting the `MustVerifyEmail` import in your `App/Models/User.php` model and ensuring the model implements the `MustVerifyEmail` interface:

```
namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
// ...

class User extends Authenticatable implements MustVerifyEmail
{
    // ...
}
```

After registration, users will receive a verification email. To restrict access to certain routes until the user's email address is verified, add the `verified` middleware to the routes:

```
Route::middleware(['auth', 'verified'])->group(function () {
    Route::get('dashboard', function () {
        return Inertia::render('dashboard');
    })->name('dashboard');
});
```

> [!NOTE] Email verification is not required when using the WorkOS variant of the starter kits.

How do I modify the default email template?

You may want to customize the default email template to better align with your application's branding. To modify this template, you should publish the email views to your application with the following command:

```
php artisan vendor:publish --tag=laravel-mail
```

This will generate several files in `resources/views/vendor/mail`. You can modify any of these files as well as the `resources/views/vendor/mail/themes/default.css` file to change the look and appearance of the default email template.