How to Use Laravel Livewire for Interactive UIs

Laravel Livewire allows you to build dynamic, reactive interfaces without writing JavaScript. Instead, Livewire components are written in PHP and Blade, and Livewire handles the DOM updates via AJAX under the hood. This makes it perfect for developers who prefer Laravel's syntax but still need interactivity. In this article, we'll set up Livewire, create components, connect them with controllers, trigger events, and build a practical login form with validation.

Installing Laravel Livewire

```
composer require livewire/livewireCode language: Bash (bash)
```

After installing, include Livewire's scripts and styles in your Blade layout:

```
<!-- resources/views/layouts/app.blade.php -->
<html lang="en">
<head>
  @livewireStyles
</head>
<body>
  <div class="container">
    @yield('content')
  </div>
  @livewireScripts
</body>
</html>Code language: PHP (php)
```

Now Livewire is available throughout your Laravel project.

Creating Your First Livewire Component

```
php artisan make:livewire CounterCode language: Bash (bash)
```

This generates two files:

```
app/Http/Livewire/Counter.php     // Component logic (PHP)
resources/views/livewire/counter.blade.php // Component view (Blade)
Code language: plaintext (plaintext)
```

```
// app/Http/Livewire/Counter.php
namespace App\Http\Livewire;

use Livewire\Component;

class Counter extends Component
{
    public $count = 0;

    public function increment()
    {
        $this->count++;
    }

    public function render()
    {
        return view('livewire.counter');
    }
}Code language: PHP (php)
```

```
<!-- resources/views/livewire/counter.blade.php -->
<div>
  <h2>Count: {{ $count }}</h2>
```

```
  <button wire:click="increment">Increment</button>
</div>Code language: PHP (php)
```

The PHP property `$count` syncs with Blade automatically. Clicking the button calls the `increment` method, Livewire updates the DOM, and no JavaScript is required.

Embedding Livewire in Blade

```
<!-- resources/views/welcome.blade.php -->
@extends('layouts.app')

@section('content')
  <h1>Welcome</h1>
  @livewire('counter')
@endsectionCode language: PHP (php)
```

Use the `@livewire('name')` directive to embed Livewire components into Blade views.

Using Livewire with Controllers

You can fetch data inside Livewire components the same way you would in a controller. For example, listing posts:

```
// app/Http/Livewire/PostList.php
namespace App\Http\Livewire;

use Livewire\Component;
```

```
use App\Models\Post;

class PostList extends Component
{
    public $posts = [];

    public function mount()
    {
        $this->posts = Post::latest()->get();
    }

    public function render()
    {
        return view('livewire.post-list');
    }
}Code language: PHP (php)
```

```
<!-- resources/views/livewire/post-list.blade.php -->
<div>
  <h2>Posts</h2>
  <ul>
    @foreach($posts as $post)
      <li>{{ $post->title }}</li>
    @endforeach
  </ul>
</div>Code language: PHP (php)
```

The `mount()` method acts like a controller constructor, loading data before rendering. Livewire keeps it reactive when the component updates.

Triggering Events with Livewire

Livewire supports emitting and listening to events between components, just like Laravel events. This enables parent-child communication without JavaScript.

```
// app/Http/Livewire/ChildComponent.php
namespace App\Http\Livewire;

use Livewire\Component;

class ChildComponent extends Component
{
    public function notifyParent()
    {
        $this->emit('childNotified', 'Child says hello!');
    }

    public function render()
    {
        return view('livewire.child-component');
    }
}Code language: PHP (php)
```

```
<!-- resources/views/livewire/child-component.blade.php -->
<button wire:click="notifyParent">Notify Parent</button>Code language: PHP (php)
```

```
// app/Http/Livewire/ParentComponent.php
namespace App\Http\Livewire;

use Livewire\Component;

class ParentComponent extends Component
{
    protected $listeners = ['childNotified'];

    public function childNotified($message)
    {
        session()->flash('info', $message);
```

```
    }

    public function render()
    {
        return view('livewire.parent-component');
    }
}Code language: PHP (php)
```

The parent listens to the `childNotified` event and reacts by flashing a session message. This mimics event-driven UIs with only PHP and Blade.

Building a Livewire Login Form

Let's build a login form UI with Livewire that handles validation and authentication directly.

```
// app/Http/Livewire/LoginForm.php
namespace App\Http\Livewire;

use Livewire\Component;
use Illuminate\Support\Facades\Auth;

class LoginForm extends Component
{
    public $email = '';
    public $password = '';

    protected $rules = [
        'email' => 'required|email',
        'password' => 'required|min:6',
    ];

    public function login()
    {
```

```
        $this->validate();

        if (Auth::attempt(['email' => $this->email, 'password' =>
$this->password])) {
            session()->regenerate();
            return redirect()->intended('/dashboard');
        }

        $this->addError('email', 'Invalid credentials.');
    }

    public function render()
    {
        return view('livewire.login-form');
    }
}Code language: PHP (php)
```

```
<!-- resources/views/livewire/login-form.blade.php -->
<form wire:submit.prevent="login">
  <div>
    <label for="email">Email</label>
    <input id="email" type="email" wire:model="email">
    @error('email') <span class="text-danger">{{ $message }}</span>
@enderror
  </div>

  <div>
    <label for="password">Password</label>
    <input id="password" type="password" wire:model="password">
    @error('password') <span class="text-danger">{{ $message }}</span>
@enderror
  </div>

  <button type="submit">Login</button>
</form>Code language: PHP (php)
```

The component handles input binding, validation, and authentication logic in PHP. No JavaScript is needed, but the UI feels dynamic thanks to Livewire.

Wrapping Up

Livewire bridges the gap between Laravel Blade and modern JavaScript frameworks. You can build interactive components, handle events, fetch data, and even implement authentication UIs without leaving PHP. It’s ideal for developers who want interactivity while staying in the Laravel ecosystem.

Livewire vs Vue: Which Should You Choose?

Laravel supports both Livewire and Vue 3. Which one you use depends on your project’s needs. Here’s a quick comparison:

Feature	Livewire	Vue 3
Learning Curve	Low — stays in PHP and Blade.	Higher — requires JS and Vue ecosystem knowledge.
Setup	Composer install, Blade directives.	Install via NPM + Vite config.
Use Cases	Forms, dashboards, CRUD, admin panels.	SPAs, real-time UIs, highly interactive apps.
Performance	Good, but frequent AJAX calls.	Excellent, reactive updates in the browser.
Flexibility	Bound to Laravel ecosystem.	Can integrate with any backend API.
Developer Audience	Laravel devs who prefer PHP-only solutions.	Teams with frontend specialists.

Recommendation: Use Livewire if you want Laravel-only productivity without diving deep into JavaScript. Use Vue 3 if your app needs SPA features, heavy interactivity, or has a dedicated frontend team.

What's Next

Want to go further with Livewire and interactive UIs? Try these guides:

- Creating a Role-Specific Dashboard in Laravel 12
- Creating a User-Friendly Roles & Permissions UI in Laravel
- Mastering Validation Rules in Laravel 12