1v0 Ship v1.0 Faster

Building a Multi-Step Form Wizard in Laravel

Multi-step (“wizard”) forms improve completion rates by breaking long forms into smaller, focused steps. In this guide, you’ll build a robust Laravel multi-step form with **session-backed state**, per-step **validation**, guarded navigation (no step skipping), and a clean **Blade UI** including a progress indicator. We’ll also cover optional file handling, edit mode, and testing hooks so your wizard is reliable in production.

Routes & Controller Skeleton

```
// routes/web.php
use App\Http\Controllers\RegistrationWizardController;

Route::prefix('register-wizard')->name('wizard.')->group(function () {
    Route::get('/step/{step}', [RegistrationWizardController::class,
'show'])->name('show');
    Route::post('/step/{step}', [RegistrationWizardController::class,
'store'])->name('store');
    Route::post('/back/{step}', [RegistrationWizardController::class,
'back'])->name('back');
    Route::post('/reset', [RegistrationWizardController::class,
'reset'])->name('reset');
});Code language: PHP (php)
```

We’ll use a single controller with dynamic step routes to keep the flow predictable. Each step has its own GET for display and POST for validation + persistence to session. A *back* action supports safe navigation without losing data, and *reset* clears the flow.

```
// app/Http/Controllers/RegistrationWizardController.php
namespace App\Http\Controllers;
```

```
use App\Http\Requests\Wizard\StepOneRequest;
use App\Http\Requests\Wizard\StepTwoRequest;
use App\Http\Requests\Wizard\StepThreeRequest;
use Illuminate\Http\Request;

class RegistrationWizardController extends Controller
{
    // Define the ordered steps for guard checks
    private array $steps = ['1', '2', '3', 'confirm'];

    public function show(Request $request, string $step)
    {
        $this->guardStep($request, $step);

        return view('wizard.step-'.$step, [
            'data' => $request->session()->get('wizard', []),
            'current' => $step,
            'steps' => $this->steps,
        ]);
    }

    public function store(Request $request, string $step)
    {
        $this->guardStep($request, $step);

        $validated = match ($step) {
            '1' => app(StepOneRequest::class)->validated(),
            '2' => app(StepTwoRequest::class)->validated(),
            '3' => app(StepThreeRequest::class)->validated(),
            'confirm' => [],
            default => abort(404),
        };

        // Merge this step's validated data into the session "wizard"
payload
        $payload = array_merge(
            $request->session()->get('wizard', []),
            [$step => $validated]
```

```
        );
        $request->session()->put('wizard', $payload);

        // When user confirms, persist & finish
        if ($step === 'confirm') {
            return $this->finish($request);
        }

        // Advance to next step
        $next = $this->nextStep($step);
        return redirect()->route('wizard.show', ['step' => $next]);
    }

    public function back(Request $request, string $step)
    {
        $prev = $this->previousStep($step);
        return redirect()->route('wizard.show', ['step' => $prev]);
    }

    public function reset(Request $request)
    {
        $request->session()->forget('wizard');
        return redirect()->route('wizard.show', ['step' => '1']);
    }

    private function finish(Request $request)
    {
        $all = $request->session()->get('wizard', []);

        // Example: flatten and persist into your domain model(s)
        $data = array_merge($all['1'] ?? [], $all['2'] ?? [],
$all['3'] ?? []);
        // \App\Models\User::create($data); // adapt for your case

        $request->session()->forget('wizard');

        return redirect('/')->with('success', 'Registration
complete!');
```

```
    }

    private function guardStep(Request $request, string $step): void
    {
        abort_unless(in_array($step, $this->steps, true), 404);

        // Prevent skipping ahead: ensure all prior steps exist in
session
        $idx = array_search($step, $this->steps, true);
        for ($i = 0; $i < $idx; $i++) {
            $required = $this->steps[$i];
            if ($required !== 'confirm' &&
empty($request->session()->get('wizard.'.$required))) {
                redirect()->route('wizard.show', ['step' =>
$required])->send();
                exit;
            }
        }
    }

    private function nextStep(string $step): string
    {
        $i = array_search($step, $this->steps, true);
        return $this->steps[min($i + 1, count($this->steps) - 1)];
    }

    private function previousStep(string $step): string
    {
        $i = array_search($step, $this->steps, true);
        return $this->steps[max($i - 1, 0)];
    }
}Code language: PHP (php)
```

Key ideas: session stores each step's validated subset; *guardStep()* blocks direct navigation to later steps; *finish()* consolidates session data into your database and clears the wizard state.

Per-Step Validation with Form Requests

```
php artisan make:request Wizard/StepOneRequest
php artisan make:request Wizard/StepTwoRequest
php artisan make:request Wizard/StepThreeRequestCode language: Bash (bash)
```

Using dedicated *FormRequest* classes keeps validation clean and testable. Each step validates only its own fields and messages.

```
// app/Http/Requests/Wizard/StepOneRequest.php
namespace App\Http\Requests\Wizard;

use Illuminate\Foundation\Http\FormRequest;

class StepOneRequest extends FormRequest
{
    public function authorize(): bool { return true; }

    public function rules(): array
    {
        return [
            'first_name' => ['required','string','max:80'],
            'last_name'  => ['required','string','max:80'],
            'email'      => ['required','email','max:255'],
        ];
    }
}Code language: PHP (php)
```

```
// app/Http/Requests/Wizard/StepTwoRequest.php
class StepTwoRequest extends FormRequest
{
    public function authorize(): bool { return true; }
```

```
    public function rules(): array
    {
        return [
            'address' => ['required','string','max:255'],
            'city'    => ['required','string','max:100'],
            'country' => ['required','string','max:2'], // ISO code
        ];
    }
}Code language: PHP (php)
```

```
// app/Http/Requests/Wizard/StepThreeRequest.php
class StepThreeRequest extends FormRequest
{
    public function authorize(): bool { return true; }

    public function rules(): array
    {
        return [
            'password' => ['required','confirmed','min:8'],
            'terms'    => ['accepted'],
        ];
    }
}Code language: PHP (php)
```

Each step populates only its slice of the session payload, reducing coupling and improving clarity in error states.

Blade UI: Layout, Progress Bar, and Step Views

```
<!-- resources/views/layouts/wizard.blade.php -->
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Multi-Step Wizard</title>
    <style>
        .progress { display:flex; gap:.5rem; margin-bottom:1rem; }
        .dot { width:12px; height:12px; border-radius:999px;
background:#ddd; }
        .dot.active { background:#4f46e5; }
        .step-card { max-width:720px; margin:auto; padding:1.25rem;
border:1px solid #eee; border-radius:12px; }
        .actions { display:flex; gap:.5rem; margin-top:1rem; }
        .error { color:#b91c1c; font-size:.9rem; }
    </style>
</head>
<body>
    <div class="progress">
        @foreach($steps as $s)
            <div class="dot {{ $s === $current ? 'active' : '' }}"
title="Step {{ $s }}"></div>
        @endforeach
    </div>

    <div class="step-card">
        @yield('content')
    </div>
</body>
</html>Code language: PHP (php)
```

A minimal layout with a *dot-based* progress bar communicates the user's position without heavy CSS frameworks. Feel free to swap with Tailwind or Bootstrap if your project already uses them.

```
<!-- resources/views/wizard/step-1.blade.php -->
@extends('layouts.wizard')

@section('content')
<h2>Step 1: Your Details</h2>
```

```
<form method="POST" action="{{ route('wizard.store',['step' => '1']) }}">
    @csrf
    <label>First name</label>
    <input name="first_name" value="{{ old('first_name', $data['1']['first_name'] ?? '') }}" />
    @error('first_name') <div class="error">{{ $message }}</div> @enderror

    <label>Last name</label>
    <input name="last_name" value="{{ old('last_name', $data['1']['last_name'] ?? '') }}" />
    @error('last_name') <div class="error">{{ $message }}</div> @enderror

    <label>Email</label>
    <input name="email" type="email" value="{{ old('email', $data['1']['email'] ?? '') }}" />
    @error('email') <div class="error">{{ $message }}</div> @enderror

    <div class="actions">
        <button type="submit">Next</button>
        <form method="POST" action="{{ route('wizard.reset') }}">@csrf <button>Reset</button> </form>
    </div>
</form>
@endsectionCode language: PHP (php)
```

```
<!-- resources/views/wizard/step-2.blade.php -->
@extends('layouts.wizard')

@section('content')
<h2>Step 2: Address</h2>

<form method="POST" action="{{ route('wizard.store',['step' => '2']) }}">
    @csrf
    <label>Address</label>
    <input name="address" value="{{ old('address',
```

```
$data['2']['address'] ?? '') }}" />
    @error('address') <div class="error">{{ $message }}</div>
@enderror

    <label>City</label>
    <input name="city" value="{{ old('city', $data['2']['city'] ?? '')
}}" />
    @error('city') <div class="error">{{ $message }}</div> @enderror

    <label>Country (ISO)</label>
    <input name="country" maxlength="2" value="{{ old('country',
$data['2']['country'] ?? '') }}" />
    @error('country') <div class="error">{{ $message }}</div>
@enderror

    <div class="actions">
        <form method="POST" action="{{ route('wizard.back',['step' =>
'2']) }}"> @csrf <button>Back</button> </form>
        <button type="submit">Next</button>
    </div>
</form>
@endsectionCode language: PHP (php)

<!-- resources/views/wizard/step-3.blade.php -->
@extends('layouts.wizard')

@section('content')
<h2>Step 3: Security</h2>

<form method="POST" action="{{ route('wizard.store',['step' => '3'])
}}">
    @csrf
    <label>Password</label>
    <input type="password" name="password" />
    @error('password') <div class="error">{{ $message }}</div>
@enderror

    <label>Confirm Password</label>
    <input type="password" name="password_confirmation" />
```

```
    <label><input type="checkbox" name="terms" value="1" {{
old('terms', $data['3']['terms'] ?? false) ? 'checked' : '' }} /> I
accept terms</label>
    @error('terms') <div class="error">{{ $message }}</div> @enderror

    <div class="actions">
        <form method="POST" action="{{ route('wizard.back',['step' =>
'3']) }}"> @csrf <button>Back</button> </form>
        <button type="submit">Next</button>
    </div>
</form>
@endsectionCode language: PHP (php)
```

```
<!-- resources/views/wizard/step-confirm.blade.php -->
@extends('layouts.wizard')

@section('content')
<h2>Confirm & Submit</h2>

@php($all = $data ?? [])
<pre>{{ json_encode($all, JSON_PRETTY_PRINT) }}</pre>

<form method="POST" action="{{ route('wizard.store',['step' =>
'confirm']) }}">
    @csrf
    <div class="actions">
        <form method="POST" action="{{ route('wizard.back',['step' =>
'confirm']) }}"> @csrf <button>Back</button> </form>
        <button type="submit">Confirm</button>
    </div>
</form>
@endsectionCode language: PHP (php)
```

Each view rehydrates fields from session so users don’t lose progress. The confirmation screen shows a summary; in real apps, present a styled review instead of raw JSON.

Preventing Step Skips & Handling Expiry

```
// Optionally, add a timestamp to invalidate long-idle sessions
// In store(): after merging payload
$payload['_meta']['touched_at'] = now()->timestamp;
$request->session()->put('wizard', $payload);

// In guardStep(): invalidate after 60 minutes of inactivity
$wizard = $request->session()->get('wizard', []);
if (($wizard['_meta']['touched_at'] ?? 0) <
now()->subMinutes(60)->timestamp) {
    $request->session()->forget('wizard');
    redirect()->route('wizard.show', ['step' =>
'1'])->with('warning','Your session expired.')->send();
    exit;
}Code language: PHP (php)
```

Guard clauses ensure users can't land on later steps without completing earlier ones. Adding an inactivity expiry avoids stale multi-day flows or abandoned sessions.

Optional: File Uploads in a Step

```
// Example rule inside a FormRequest for a file step:
'avatar' => ['nullable','image','mimes:jpg,jpeg,png','max:2048'];

// Controller store() for that step:
if ($request->hasFile('avatar')) {
```

```
    $path = $request->file('avatar')->store('wizard-
avatars','public');
    $validated['avatar_path'] = $path;
}Code language: PHP (php)
```

When a step includes files, store them immediately and keep only the path in session (not raw file data). Serve back previews from disk and re-validate on confirmation.

Edit Mode (Reopen Wizard with Existing Data)

```
// Preload existing profile data into session then redirect to step 1
public function edit(Request $request)
{
    $profile = auth()->user()->profile;
    $request->session()->put('wizard', [
        '1' => ['first_name' => $profile->first_name, 'last_name' =>
$profile->last_name, 'email' => $profile->email],
        '2' => ['address' => $profile->address, 'city' =>
$profile->city, 'country' => $profile->country],
        '3' => [],
    ]);
    return redirect()->route('wizard.show', ['step' => '1']);
}Code language: PHP (php)
```

For editing, hydrate the session with existing values, then let users move through steps and resubmit. On finish, update rather than create new rows.

Feature Testing the Wizard

```
// tests/Feature/RegistrationWizardTest.php
use Tests\TestCase;

class RegistrationWizardTest extends TestCase
{
    public function test_user_cannot_skip_steps(): void
    {
        $this->get('/register-
wizard/step/3')->assertRedirect('/register-wizard/step/1');
    }

    public function test_step_one_validates_and_persists(): void
    {
        $this->post('/register-wizard/step/1', [
            'first_name' => 'Jane',
            'last_name'  => 'Doe',
            'email'      => 'jane@example.com',
        ])->assertRedirect('/register-wizard/step/2');

        $this->get('/register-wizard/step/2')->assertOk();
    }
}Code language: PHP (php)
```

Tests should assert redirect behavior on invalid navigation and confirm that each step stores data and advances the flow. Add more cases for expiries and file steps as needed.

Livewire / Vue Comparison (When to Choose What)

Aspect	Blade + Sessions (This Article)	Livewire	Vue 3 (SPA)

1v0 Ship v1.0 Faster

Aspect	Blade + Sessions (This Article)	Livewire	Vue 3 (SPA)
Complexity	Low	Medium	High
State Handling	Server session	Livewire component state	Client store (Pinia/Vuex)
UX Smoothness	Full-page reloads	Partial (AJAX, no reload)	SPA-smooth
Best For	Simple to moderate wizards	Dynamic forms without full SPA	Rich UIs, complex flows

Start with Blade + sessions for speed and simplicity. Move to Livewire for AJAX-powered steps without building an SPA. Choose Vue for complex, highly interactive multi-step flows where client-side routing/state is a win.

Wrapping Up

You built a production-ready multi-step form wizard using Laravel, Blade, sessions, and per-step FormRequest validation. The guard rules prevent skipping steps; the UI keeps users oriented; and optional features like expiry, file handling, and edit mode make it practical for real apps. Expand with Livewire or Vue when UX demands it.

What’s Next

Keep leveling up your forms and UI:

- Mastering Validation Rules in Laravel 12
- Handling File Uploads and Image Storage in Laravel

- How to Use Laravel Livewire for Interactive UIs