Introduction

Declarative Forms, Seamless Rendering

Anonymous · 09/09/2026

Declarative Forms, Seamless Rendering

A modern PHP form library that leverages a declarative, schema-based approach to form definition and rendering, compatible with JSON Forms while providing a powerful backend-centric workflow.

WIP

Work In Progress.

Key Features

- **Schema-Based Form Definition**: Define forms using structured arrays that clearly separate data schemas from UI layouts.
- **Backend-First Approach**: Built for PHP developers who want control over form generation with minimum effort.
- **Symfony-Compatible Twig Extensions**: Familiar syntax for Symfony developers, minimizing learning curve.
- **Decoupled Rendering System**: Separate your form logic from its presentation with specialized renderers.
- **JSON Forms Compatibility**: Define once, use anywhere - same schema can be used with frontend JSON Forms library.
- **Automatic Form Generation**: Generate form UI schemas automatically from your data models.
- **Extensible Architecture**: Easily add custom types, layouts and renderers.
- **Translatable, Optionally**: Validation error messages and input action labels (show/hide password, copy, etc.) can be translated by wiring an optional translator — see Translations.

Why Choose This Library?

Compared to Traditional PHP Form Libraries

Traditional PHP form libraries often tightly couple form definition with HTML generation, making it difficult to separate concerns. This library takes a different approach:

1. **Declarative Rather Than Imperative**: Define *what* your form should be rather than *how* it should be built, step by step.
2. **Clear Separation of Concerns**: Schema (data structure) is separate from UI schema (presentation) and data (values).
3. **Powerful Layouts Without HTML Knowledge**: Create complex, multi-column layouts without writing HTML.

Compared to Frontend-Only Solutions

Unlike frontend-only form builders, this library:

1. **Keeps Validation Logic Server-Side**: Where it belongs for security-critical applications.
2. **Provides PHP-Native Form Definition**: No need to write JavaScript to define your forms.
3. **Works With or Without JavaScript**: Generate the same forms for both traditional and SPA applications.

Installation

Install the library using Composer:

```
composer require derafu/form
```

Basic Usage

Creating a Simple Form

```
// Define the form structure.
$definition = [
    'schema' => [
        'type' => 'object',
        'properties' => [
            'name' => [
                'type' => 'string',
                'title' => 'Full Name',
            ],
            'email' => [
                'type' => 'string',
                'format' => 'email',
                'title' => 'Email Address',
```

```
            ],
        ],
        'required' => ['name', 'email'],
    ],
    'uischema' => [
        'type' => 'VerticalLayout',
        'elements' => [
            [
                'type' => 'Control',
                'scope' => '#/properties/name',
            ],
            [
                'type' => 'Control',
                'scope' => '#/properties/email',
            ],
        ],
    ],
    'data' => [
        'name' => '',
        'email' => '',
    ],
];

// Create the form.
$form = $formFactory->create($definition);
```

Rendering with Twig

```
{# Simple rendering of the entire form. #}
{{ form(form) }}

{# Or with more control over individual components. #}
{{ form_start(form) }}
    {{ form_row(form.fields.name) }}
    {{ form_row(form.fields.email) }}
    <button type="submit" class="btn btn-primary">Submit</button>
{{ form_end(form) }}
```

Automatic Schema Generation

The library can automatically generate a schema from your data:

```
$definition = [
    'data' => [
        'name' => 'John Doe',
        'email' => 'john.doe@example.com',
        'age' => 34,
```

```
        'birthdate' => "1990-01-01",
    ],
];

// Schema and UI schema will be automatically generated.
$form = $formFactory->create($definition);
```

Integration with JSON Forms

This library's form definitions are compatible with the JSON Forms JavaScript library, allowing you to use the same definitions for both server-side and client-side rendering:

```
<div id="json-form-container"></div>
```

```
//
https://raw.githubusercontent.com/derafu/form/refs/heads/main/assets/js/jsonforms-viewer.js
import './jsonforms-viewer.js';

// Render the form and save the reference.
const formDefinition = {{ jsonFormsDefinition | json_encode | raw }};
const formInstance = window.renderJsonForm('json-form-container', formDefinition);
// You can now use `formInstance` to get the form data or its errors.
```

Development Requirements

To render forms using JSON Forms you must install:

```
npm install @jsonforms/core @jsonforms/react @jsonforms/material-renderers \
    react react-dom @mui/material @emotion/react @emotion/styled
```

Form Layouts

The UI schema supports various layout types by default:

- **VerticalLayout**: Fields stacked vertically.
- **HorizontalLayout**: Fields arranged horizontally.
- **Group**: Logical grouping of fields, often with a label.
- **Categorization**: Tab-based layouts for complex forms.

Example:

```
$uischema = [
```

```
    'type' => 'VerticalLayout',
    'elements' => [
        [
            'type' => 'Group',
            'label' => 'Personal Information',
            'elements' => [
                ['type' => 'Control', 'scope' => '#/properties/name'],
                ['type' => 'Control', 'scope' => '#/properties/email'],
            ],
        ],
        [
            'type' => 'Group',
            'label' => 'Address',
            'elements' => [
                ['type' => 'Control', 'scope' => '#/properties/street'],
                ['type' => 'Control', 'scope' => '#/properties/city'],
            ],
        ],
    ],
];
```

Last updated on 09/09/2026