**ScopeTreasure Courses**

LARAVEL COURSES · LESSON LV.5 · CODE SCOPE SERIES

# Laravel Forms + Validation — User Se Data Lena

Taking data safely: Blade forms with @csrf, $request->validate rules, red @error messages and old() refill — the full request cycle.

## Introduction

Pages so far only *showed* data. Real apps also *take* data: a name typed, marks entered, a button pressed. That journey — form → server → check → save — is where beginners lose the most time, because four different guards work together and each one failing looks like a different mystery.

Laravel's form system is honest and strict. The form carries a secret security token (`@csrf`). The controller catches the submission in a `$request`. Before anything is saved, **validation** checks every field against rules you declare — marks must be a number between 0 and 100, a name must be present. Bad data bounces back with clear red messages and the user's typing restored by `old()`. In this lesson you will walk the whole cycle once, slowly, and never fear a form again.

*(Roman Urdu: Form mehman hai jo darwazay par aata hai — pehle token ki pehchan, phir data ki jaanch, tab andar entry.)*

## Key points

- **Forms speak POST:** reading pages use GET; sending data uses `<form method="POST" action="...">`. The route must match: `Route::post('/students', ...)` — a POST form knocking on a GET route gets a polite 405 error.
- **`@csrf` is the security key:** every Laravel form must carry this Blade directive, which hides a secret token in the form. Submissions without the matching token are rejected with the famous **419 Page Expired**. It stops other websites from submitting forms in your users' names.
- **The `$request` is the delivery bag:** the controller method receives `Illuminate\Http\Request $request`, and `$request->input('name')` (or `$request->name`) reads what was typed. `$request->all()` shows everything that arrived.
- **Validation is one line with rules:** `$request->validate([...])` — each field gets pipe-separated rules: `'marks' => 'required|integer|min:0|max:100'`. If anything fails, Laravel stops right there — your save code never runs on bad data.
- **Errors come back by themselves:** a failed validation redirects back to the form with an error bag. Blade prints each message with `@error('marks') {{ $message }} @enderror` — exactly under the guilty field.
  *(Roman Urdu: Ghalti wahi laal nishan ke saath wapis aati hai — user ko andaza nahi lagana parta ke ghalat kya tha.)*
- **`old()` restores the typing:** `value="{{ old('name') }}"` refills the box with what the user typed before the bounce, so one wrong field never costs them the whole form.
- **The order never changes:** token check → validation → save → redirect. Learn this cycle once and every form in every future app is the same shape.

## Code examples

### Example 1 — The form itself: token, fields, old values, error spots

```
{{-- resources/views/students/create.blade.php --}}
<form method="POST" action="{{ route('students.store') }}">
    @csrf

    <label>Name</label>
    <input type="text" name="name" value="{{ old('name') }}">
    @error('name')
        <p class="error">{{ $message }}</p>
    @enderror

    <label>Marks</label>
    <input type="text" name="marks" value="{{ old('marks') }}">
    @error('marks')
        <p class="error">{{ $message }}</p>
    @enderror

    <button type="submit">Save Student</button>
</form>
```

What it does: A complete, honest form. `@csrf` hides the security token. Each input's `name` attribute is the key the controller will read. `old()` stands ready to refill, and each field keeps an `@error` slot beneath it where its personal complaint will appear.

**What you will see:** a clean form at `/students/create` — two empty boxes and a button. Nothing red yet; the error slots are empty because no complaint exists. That changes in Example 3.

### Example 2 — The receiving end: validate first, then save

```
// routes/web.php
Route::post('/students', [StudentController::class, 'store'])->name('students.store');
```

```
// StudentController
public function store(Request $request)
{
    $validated = $request->validate([
        'name'  => 'required|string|max:50',
        'city'  => 'required|string|max:50',
        'marks' => 'required|integer|min:0|max:100',
    ]);

    $student = new Student;
    $student->name  = $validated['name'];
    $student->city  = $validated['city'];
    $student->marks = $validated['marks'];
    $student->save();

    return redirect()->route('students.index');
}
```

What it does: The POST route points at `store()`. Validation runs FIRST — only data that passes every rule lands in `$validated`, and only that clean data is saved. Then the user is redirected to the list page (never leave a user staring at a submitted form — a refresh would submit it twice; LV.6 grows this into the full CRUD pattern).

**What you will see:** submit `Sara Khan / Lahore / 92` and you land on the students list with Sara in it. The database holds exactly one new row — validated, then saved.

### Example 3 — The bounce: bad data, red messages, typing restored

What it does: Now submit the form with the name left empty and marks typed as `150`. Validation fails both fields, Laravel redirects back, and Blade's error slots and `old()` helpers wake up.

**What you will see:** the same form again — but the marks box still holds `150` (old input restored), and two red messages stand under their fields:

```
The name field is required.
The marks field must not be greater than 100.
```

Nothing was saved. Fix marks to `95`, type the name, submit — and the cycle completes cleanly. This bounce is not a failure of your app; it is your app protecting itself.

## Quick recap

- Form cycle: **@csrf token → POST route → $request → validate() → save → redirect.** Always in that order.
- Validation rules are declared per field; failures bounce back automatically with messages.

- `@error` prints the complaint under its field; `old()` refills what the user typed.
- A 419 means the token was missing; a 405 means the form knocked on the wrong verb's route.

## Common mistakes

1. **Forgetting `@csrf`** — the most famous beginner error in Laravel: every submission dies with 419 Page Expired. Fix: `@csrf` is the first line inside every form, no exceptions.
2. **Validating after saving** — checking rules below the `save()` line means bad data is already in the database. Fix: `validate()` is always the first thing `store()` and `update()` do.
3. **Trusting the browser's checks** — `type="number"` and `required` in HTML are hints users (and attackers) can bypass in seconds. Fix: server-side validation is the real guard; browser checks are only kindness.
4. **Losing the user's typing** — a bounced form that comes back empty teaches users to hate your app. Fix: every input carries `value="{{ old('field') }}"` from day one.

## Practice

1. Add a `city` field to the create form with the rules from Example 2 and its own `@error` slot. Submit once with city empty and confirm its message appears under city — not anywhere else.
2. Try to break it: submit marks as `abc`, as `-5`, and as `101`. Write down the three different messages Laravel gives you — each rule speaks its own sentence.

## Practice quiz

**1. A Laravel form submission dies with 419 Page Expired. What is missing?**

- The route
- The @csrf directive, the first line inside every form, whose secret token stops other websites submitting forms in your users' names
- The validation rules
- The model

**Answer:** The @csrf directive, the first line inside every form, whose secret token stops other websites submitting forms in your users' names

**2. Where in a store() or update() method must validate() sit, and why?**

- After save(), so the user sees the result first
- First, before save: if anything fails Laravel stops right there, so your save code never runs on bad data
- It does not matter
- Only in store(), update() needs no validation

**Answer:** First, before save: if anything fails Laravel stops right there, so your save code never runs on bad data

**3. After a failed validation, how does the user's typing survive the bounce back?**

- It does not, the form comes back empty
- With value="{{ old('name') }}", which refills each box with what the user typed, so one wrong field never costs them the whole form
- With the browser's back button only
- With JavaScript only

**Answer:** With value="{{ old('name') }}", which refills each box with what the user typed, so one wrong field never costs them the whole form

---

**Watch the Code Scope channel:** https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

**Read online:** https://scopetreasure.com/notes/laravel-forms-validation · More free tools and notes at https://scopetreasure.com