ScopeTreasure Courses

LARAVEL COURSES · LESSON LV.13 · CODE SCOPE SERIES

REST API in Laravel — Dusri Apps Se Guftagu

Your app learns a second language: apiResource routes, honest JSON answers, 201 and 422 status codes, Postman testing and API Resources that hide secrets.

Introduction

Your Students app speaks HTML — pages for human eyes. A phone app, a JavaScript front-end, or another company's system cannot eat HTML; they want the **data itself**, in JSON. An **API** is that second mouth: the same students, the same rules, answered as JSON over the same honest verbs — GET to read, POST to create, PUT to update, DELETE to remove. That style — resources, verbs, status codes — is **REST**.

The beautiful part: almost nothing new is needed. The same models, the same validation, the same `findOrFail` manners — even LV.5's validation errors return as tidy JSON automatically. This lesson builds the students API, tests it with Postman, reads its status codes like a professional, and finishes with **API Resources** — the translation layer that decides exactly which fields the outside world may see.

(Roman Urdu: Website insaanon ke liye sajti hai; API mashinon ke liye sach bolti hai — wahi data, bas JSON ki zubaan mein.)

Key points

- **API routes live in `routes/api.php`** and automatically wear the `/api` prefix: `Route::apiResource('students', ...)` answers at `/api/students`. Web routes serve pages; api routes serve data — separate files, separate manners.
- **`apiResource` makes 5 routes, not 7:** index, store, show, update, destroy. The two missing ones — create and edit — exist to show HTML forms, and APIs have no forms to show. Your LV.6 verb knowledge transfers unchanged.
- **Return models, receive JSON:** when a controller method returns an Eloquent model or collection, Laravel converts it to JSON and sets the right headers by itself. No `json_encode`, no manual headers.
- **Status codes are the API's tone of voice:** `200` OK (reads, updates), `201` Created (a successful POST), `404` Not Found, `422` your data failed validation. Clients — apps you will never meet — steer entirely by these numbers, so say them correctly.

  (Roman Urdu: API mein chehra nahi hota, lehja hota hai — status code hi us ka lehja hai. 201 khushi, 422 narazgi, 404 "nahi mila".)
- **`findOrFail` speaks JSON here:** on the web it showed a 404 page; asked for JSON, the same call returns a 404 JSON body. One habit, both worlds.
- **Postman is the API's browser:** it sends GET/POST/PUT/DELETE with bodies and headers you control, and shows the raw JSON and status. Add the header `Accept: application/json` so Laravel answers errors as JSON instead of redirecting like a website.

- **API Resources guard the guest list of fields:** a Resource class decides what leaves the building — formatting dates, renaming keys, and *hiding* internals. A user model poured raw into JSON would leak its password hash to anyone who asks; a Resource never includes it.

Code examples

Example 1 — The API's five routes and its first JSON answer

```
// routes/api.php
use App\Http\Controllers\Api\StudentApiController;

Route::apiResource('students', StudentApiController::class);
```

```
// The whole index() — the web version's quiet twin
public function index()
{
    return Student::all();
}
```

What it does: `apiResource` registers the five data routes under `/api`. The controller's `index()` simply returns the collection; Laravel's JSON machinery does the dressing.

What you will see: Postman (or the browser) GET `http://127.0.0.1:8000/api/students` → status `200 OK` and the class in JSON:

```
[
  {"id": 1, "name": "Ali Raza", "city": "Sahiwal", "marks": 88, "teacher_id": 1},
  {"id": 2, "name": "Sara Khan", "city": "Lahore", "marks": 92, "teacher_id": 1},
  {"id": 3, "name": "Bilal Ahmed", "city": "Sahiwal", "marks": 76, "teacher_id": 1}
]
```

(The real answer also carries `photo`, `created_at` and `updated_at` per row — Example 3 exists precisely to tidy that.)

Example 2 — Create, fail, and not-found: 201, 422, 404

```
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',
    ]);

    return Student::create($validated);
}

public function show(Student $student)
{
    return $student;
}
```

What it does: `store()` reuses LV.5's exact rules — validation was never a "web" feature. Route model binding on `show()` fetches or 404s. A POST that passes returns the newborn student; Laravel marks a creation model with status **201** automatically.

What you will see: POST valid data → `201 Created` with the new student's JSON (id 6). POST `"marks": 150` with the `Accept: application/json` header → `422` and:

```
{
  "message": "The marks field must not be greater than 100.",
  "errors": { "marks": ["The marks field must not be greater than 100."] }
}
```

GET `/api/students/99` → `404` with a JSON message naming the missing model. Three codes, three honest answers — that is a professional API's accent.

Example 3 — The API Resource: only what guests may see

```
php artisan make:resource StudentResource
```

```
// app/Http/Resources/StudentResource.php
public function toArray(Request $request): array
{
    return [
        'id'        => $this->id,
        'name'      => $this->name,
        'city'      => $this->city,
        'marks'     => $this->marks,
        'is_topper' => $this->marks >= 90,
        'joined'    => $this->created_at->format('d M Y'),
    ];
}
```

```
// controller
return StudentResource::collection(Student::all());
```

What it does: `toArray()` is the guest list. Fields not written here — `teacher_id`, the raw timestamps, the photo path — simply never leave. A computed kindness (`is_topper`) and a human date (`joined`) are added in the same breath. Applied to a User model, this is where the password hash and remember-token get left behind, every single time.

What you will see: the same GET now answers trimmed and friendly:

```
[
  {"id": 1, "name": "Ali Raza", "city": "Sahiwal", "marks": 88, "is_topper": false, "joined":
"19 Jan 2026"},
  {"id": 2, "name": "Sara Khan", "city": "Lahore", "marks": 92, "is_topper": true, "joined":
"19 Jan 2026"}
]
```

wrapped in a `data` key — the Resource collection's standard envelope.

Quick recap

- `routes/api.php` + `apiResource` = 5 data routes under `/api`; models returned become JSON automatically.
- Status codes are the voice: 200 reads, 201 created, 404 missing, 422 invalid — always honest.
- Postman + `Accept: application/json` is how APIs are tested and how errors stay JSON.
- API Resources decide which fields leave the building — secrets never do.

Common mistakes

1. **Pouring models raw into public JSON** — password hashes, tokens and internal flags ride along because "the model has them". Fix: a Resource on every public endpoint; the guest list is written, never assumed.
2. **Returning 200 for everything** — creations and failures wearing the success code teach clients to misread your API. Fix: 201 for created, 422 for invalid, 404 for missing — Laravel already picks most of them; do not flatten them.
3. **Testing APIs only in the browser** — browsers only GET, and error answers come back as web pages, hiding the JSON truth. Fix: Postman with the `Accept: application/json` header from the first test onward.
4. **Writing a second validation for the API** — the web rules and API rules drift apart within a month. Fix: the same `validate()` rules (later, the same FormRequest class) serve both mouths of the app.

Practice

1. Build the full students API (all five methods) and run the four-code drill in Postman: a 200 list, a 201 create, a 422 bad-marks create, and a 404 show. Save the collection — it is your API's test diary.
2. Add a `TeacherResource` that embeds its students (hint: a Resource may contain `StudentResource::collection($this->students)`) and return it from a `/api/teachers/1` endpoint. Read the nested JSON and name one field you deliberately left out.

Practice quiz

1. How many routes does apiResource make, and which two are missing compared with resource?

- 7, the same as resource
- 5: index, store, show, update, destroy; create and edit are missing because they exist to show HTML forms, and APIs have no forms to show
- 3: index, show, destroy
- 5, missing store and update

Answer: 5: index, store, show, update, destroy; create and edit are missing because they exist to show HTML forms, and APIs have no forms to show

2. Which status code says what in an API?

- 200 for everything
- 200 OK for reads and updates, 201 Created for a successful POST, 404 Not Found, 422 when data failed validation
- 201 for reads, 200 for creates
- 404 for success, 200 for failure

Answer: 200 OK for reads and updates, 201 Created for a successful POST, 404 Not Found, 422 when data failed validation

3. Why must public API endpoints use an API Resource instead of pouring models raw into JSON?

- For speed only
- Because a user model poured raw into JSON would leak its password hash to anyone who asks; a Resource decides what leaves the building and hides internals
- Resources are required by PHP
- Raw models cannot become JSON

Answer: Because a user model poured raw into JSON would leak its password hash to anyone who asks; a Resource decides what leaves the building and hides internals

---

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-rest-api · More free tools and notes at https://scopetreasure.com