ScopeTreasure Courses

LARAVEL COURSES · LESSON LV.8 · CODE SCOPE SERIES

Laravel Final Project + Deployment — App Ko Live Karna

The series finale of part one: the finished Students app toured once, then the 5 honest steps that take it from localhost to a live server.

Introduction

Your Students Manager is complete: routes (LV.2), Blade pages (LV.3), a real database (LV.4), guarded forms (LV.5), full CRUD (LV.6) and a login lock (LV.7). But it lives on `127.0.0.1` — an address that means "this computer only". **Deployment** is the journey from your laptop to a server, so the whole world can visit one real address and use your app.

Deployment scares beginners because it is five small steps wearing one big name. This lesson names them plainly: prepare a server with PHP and a database, send your files, write the server's own `.env`, run the migrations, and switch debug off. Do them in order and `myapp.com/students` answers exactly like your laptop did — because it is the same app, the same code, finally awake in public.

(Roman Urdu: Deployment matlab app ko apne kamray se duniya ke stage par le jana — paanch chhote qadam, aik bara naam.)

Key points

- **Localhost is a rehearsal room:** `127.0.0.1:8000` is visible only to your machine. A server is just a computer that stays on, in public, with a real address (a domain) pointing at it.
- **The app does not change — its settings do:** the same code runs on the server. What changes is `.env`: the server's database name, its passwords, `APP_ENV=production`, and a real `APP_URL`. Code is travelled luggage; `.env` is the local address written on arrival.
- **`APP_DEBUG=false` is non-negotiable live:** debug mode prints beautiful error pages — beautiful *for hackers too*, showing your code, paths and settings. Production errors must be quiet faces with the details written to the log file instead.
- **Migrations travel with the app:** you never rebuild tables by hand on the server. `php artisan migrate --force` runs the same blueprints from LV.4 and constructs identical tables. This is why migrations exist.
  (Roman Urdu: Apni mez kursi haath se dobara mat banao — migration ka naksha saath le jao, server khud bana le ga.)
- **The web root is `public/`:** the server must point its document root at the project's `public` folder — that is the only door. Pointing it at the project root exposes files that were never meant to be public.
- **Three duties begin at launch: backups** (the database, regularly — your users' data is now real), **updates** (framework and server patches), and **logs** (read `storage/logs/laravel.log` when something feels wrong — the app writes its diary there).

- **Deploy is a repeatable ritual, not an adventure:** the same five steps, every time, in order. Write them in a file once; follow the file forever.

Code examples

Example 1 — The final tour: the app you actually built

What it does: Before shipping, walk the whole system once as a user. Log in as the teacher. Open `/students`: the five students in a table (LV.3's loop, LV.4's data). Add a sixth student through the validated form (LV.5). Edit Bilal's marks and watch the flash banner (LV.6). Delete the practice row and confirm. Log out and try `/students` — bounced to login (LV.7).

What you will see: every lesson answering in one flow, with no step feeling foreign. That is the real certificate of part one: not eight small demos, but ONE app where routes, views, models, forms, CRUD and auth hold hands. Freeze this version — it is the exact app the next examples ship.

Example 2 — The five steps: from laptop to live server

```
# On the server, inside the project folder:

# Step 1 — server ready: PHP + Composer + MySQL installed (hosting panel or SSH)
# Step 2 — files sent: upload the project (or git clone), then:
composer install --no-dev --optimize-autoloader

# Step 3 — the server's own .env: real DB name/password, then:
php artisan key:generate

# Step 4 — build the tables from the travelled blueprints:
php artisan migrate --force

# Step 5 — production switches (inside .env):
# APP_ENV=production
# APP_DEBUG=false
# APP_URL=https://myapp.com
php artisan config:cache
```

What it does: Five steps, each with one job. Files travel (Git is the cleanest suitcase). Composer installs only the production packages. The server gets its OWN `.env` — you never copy your laptop's, because its database password and its secrets belong to the server. `key:generate` mints the app key that encrypts sessions and cookies there. Migrations build the tables. Debug goes off.

What you will see: the migrate step printing the same DONE lines as LV.4 — on a machine across the internet. Then `https://myapp.com/students` opens the login page. The app is live.

Example 3 — After launch: the log is the app's diary

```
tail -n 50 storage/logs/laravel.log
```

```
[2026-01-19 09:14:22] production.ERROR: SQLSTATE[HY000] [2002] Connection refused
```

What it does: When a live user reports "it broke", you do not guess — you read the diary. This real-shaped entry tells the whole story in one line: at 09:14 the app could not reach its database (the server's MySQL was down or the `.env` password had changed). Error page for the user, truth in the log for you — that split is exactly why `APP_DEBUG=false` is safe.

What you will see: timestamped entries, one per incident, each naming the failure plainly. Ninety percent of live mysteries are answered in the last fifty lines.

Quick recap

- Deployment = same code, public server, server's own `.env`, migrations run there, debug off.
- The 5 steps: server ready → files sent → `.env` + key → `migrate --force` → production switches.
- Web root points at `public/` — the only public door.
- After launch: backups, updates, and the log diary are your three standing duties.

Common mistakes

1. **Copying your laptop `.env` to the server** — wrong database, wrong passwords, debug still on, and your local secrets now live in two places. Fix: write the server's `.env` fresh on the server; only its values belong there.
2. **Leaving `APP_DEBUG=true` live** — one error page can gift an attacker your table names, file paths and config. Fix: `APP_DEBUG=false` before the first public visit, and verify by triggering a harmless 404.
3. **Building tables by hand on the server** — hand-built tables drift from the code within a week. Fix: migrations only, always — the blueprints ARE the tables' truth.
4. **No backup before the first update** — one bad update with no backup is a permanent loss. Fix: back up the database before every deploy ritual, not after the first disaster.

Practice

1. Write your own `DEPLOY.md` in the school app: the five steps in your words, with the exact commands. Then deploy to any free/cheap PHP host or a second machine — and tick each step off the file as you do it.
2. On the live copy, break one thing on purpose (rename the database in `.env`), visit the site, then find the incident in `laravel.log`. Write the log line's three facts: when, what system, what failed.

Practice quiz

1. On deployment, what changes: the code or the settings?

- The code is rewritten for the server
- The app does not change, its settings do: the server's own .env with its database name, passwords, APP_ENV=production and a real APP_URL; code is travelled luggage, .env is the local address written on arrival
- Both change completely
- Nothing changes at all

Answer: The app does not change, its settings do: the server's own .env with its database name, passwords, APP_ENV=production and a real APP_URL; code is travelled luggage, .env is the local address written on arrival

2. Why is APP_DEBUG=false non-negotiable live?

- Because debug mode is slow
- Because debug error pages show your code, paths and settings, beautiful for hackers too; production errors must be quiet faces with details written to the log file instead
- Because debug mode breaks the site
- It is not, debug may stay on

Answer: Because debug error pages show your code, paths and settings, beautiful for hackers too; production errors must be quiet faces with details written to the log file instead

3. Where must the server's web root point, and why?

- At the project root, so everything is reachable
- At the project's public folder, the only door; pointing at the project root exposes files that were never meant to be public
- At the storage folder
- It does not matter

Answer: At the project's public folder, the only door; pointing at the project root exposes files that were never meant to be public

---

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-final-project-deployment · More free tools and notes at https://scopetreasure.com