predev. Academy

TRACK ROADMAP

Backend Development with PHP and Laravel

Build, test, document, and deploy a production REST API with a Filament admin panel using Laravel 13 and PHP 8.4.

Web Backend

DURATION	SESSIONS	SCHEDULE	FORMAT	LEVEL	LANGUAGE
10 weeks	20 live sessions, 60 hours	Twice a week, 3 hours per session, plus 1 hour of online office hours every week	In person or online, live	Beginner, after Tech Compass	Egyptian Arabic, English technical terms

Who it is for

- CS and IT students who want a backend skill they can get hired or freelance with
- Fresh graduates who know some programming but have never built and shipped a real API
- Career switchers who finished Tech Compass and chose web backend
- Frontend or mobile developers who want to write the API their apps talk to

Before day 1

- Tech Compass completed, or basic comfort with variables, conditions, and loops in any language
- Laptop with 8 GB RAM minimum (16 GB recommended), 20 GB free disk, Windows 11, macOS, or Linux
- Installed before day 1: VS Code or PhpStorm, Git, PHP 8.4 with Composer (Laravel Herd on Windows and macOS, or php.new on Linux), and a database client such as TablePlus or DBeaver
- Accounts: GitHub, and a free API client such as Postman or Bruno
- Reading basic English documentation, with a translator if needed

By the end you can

- Explain what happens during an HTTP request and pick the correct method and status code for each endpoint
- Write clean, typed PHP 8.4 with classes, interfaces, traits, enums, and namespaces, managed with Composer
- Design a relational schema, write SQL joins and aggregates, and translate the schema into Laravel migrations and Eloquent relations
- Build a versioned REST API with Form Request validation, API Resources, pagination, filtering, and consistent error responses
- Secure the API with Sanctum tokens, policies, and rate limits, and avoid common holes such as mass assignment and broken authorization

- Move slow work to Redis queues, send mail and notifications, store uploaded files, and schedule recurring tasks
- Write Pest feature tests for every important endpoint and business rule
- Publish OpenAPI documentation and build a Filament admin panel for the operations team
- Use Claude Code or Cursor to speed up work, then review and test the output before merging
- Deploy a Laravel app with Docker and Coolify on a VPS with Nginx, SSL, workers, scheduler, and database backups

Tech stack

LANGUAGE & FRAMEWORK

PHP 8.4	Typed, object oriented, and still the language behind a large share of the web and many Egyptian companies
Laravel 13	The most used PHP framework, with routing, ORM, queues, mail, and testing built in
Composer	Installs packages and autoloads your classes, every PHP project uses it

API & AUTH

Laravel Sanctum	Token auth for mobile and SPA clients without OAuth complexity
API Resources and Form Requests	Keep response shape and input validation out of controllers
OpenAPI with Scramble	Generates API docs from your code so frontend and mobile teams can use the API

TOOLING & DEPLOY

Git and GitHub	Version control, pull requests, and your public portfolio
Docker	Runs the same environment on your laptop and on the server
Coolify on a VPS	Self hosted deploys with Nginx, SSL, env management, and backups, the way small teams ship in Egypt
Claude Code or Cursor	AI coding assistants, used with tests and code review

DATA

MySQL 8 or PostgreSQL 17	The relational databases you will meet in almost every Laravel job
Eloquent ORM	Maps tables to PHP models and relations so queries stay readable
Redis	Backs queues and cache in production Laravel apps

ADMIN & TESTING

Filament 5	Builds a full admin panel on top of your models in hours
Pest 4	Readable feature tests that protect the API from regressions

The running project: Hagz: bookings API and admin panel

Hagz is a booking backend for clinics, salons, and padel courts: customers find a venue, pick a service and a time slot, and book and pay; venue owners and admins manage everything from a Filament panel. It mirrors the booking and scheduling backends predev. Solutions builds for clients, and it covers every core Laravel topic: relations, auth roles, business rules, queues, notifications, and payments.

- Customer registration and login with Sanctum tokens, plus roles for customer, venue owner, and admin

- Venues, branches, services, staff, working hours, and generated time slots
- Search and filter venues by area, category, price, and availability, with pagination
- Booking flow with double booking protection using database transactions and row locks
- Booking status lifecycle (pending, confirmed, cancelled, completed) modelled with a PHP enum
- Payment checkout and webhook handling against a sandbox gateway such as Paymob
- Email and database notifications for booking confirmation and cancellation, sent through Redis queues
- Scheduled reminder job one day before each booking
- Venue image upload to local or S3 compatible storage
- Reviews and ratings per venue, with cached average rating
- Versioned API (/api/v1) with OpenAPI docs generated by Scramble
- Filament admin panel with venue, booking, and user management plus a revenue dashboard
- Pest feature test suite covering auth, booking rules, and permissions
- Live deployment on a VPS through Coolify with HTTPS, queue worker, scheduler, and daily backups

Phase overview

WEEKS 1-3

Foundations

Understand how the web works, write typed object oriented PHP, use Git and Composer, and design and query a relational database.

WEEKS 4-7

Core framework

Build the Hagz bookings API and its Filament admin panel step by step with Laravel 13, covered by Pest tests.

WEEKS 8

Building with AI

Use Claude Code or Cursor to move faster on a real codebase while keeping quality under control through specs, tests, and review.

WEEKS 9

Capstone

Build your own backend from a brief and deploy it live on a VPS.

WEEKS 10

Career week

Present your work and prepare to get your first backend job or freelance client.

Week by week

Foundations · Weeks 1-3

Week 1 How the web works and first steps in PHP

Session 1 HTTP, APIs, and your developer setup

- Client and server, DNS, IP addresses, and ports in plain terms
- Anatomy of an HTTP request and response: method, URL, headers, body
- HTTP methods (GET, POST, PUT, PATCH, DELETE) and when to use each
- Status codes: 2xx, 3xx, 4xx, 5xx, and the ones you will use daily
- JSON as the language of APIs, and what makes an API REST style
- Tool setup check: PHP 8.4, Composer, VS Code extensions, terminal basics

In-class lab: Open the browser DevTools Network tab on a real site and read five requests. Then call a public API with curl and with Bruno or Postman, and write a 15 line PHP script served with the built in server that returns JSON with the correct Content-Type and status code.

Session 2 PHP 8.4 basics and problem solving

- Variables, scalar types, strings, and declare(strict_types=1)
- Arrays (indexed and associative) and the array functions you will use most
- Conditions, loops, and the match expression
- Functions with typed parameters and return types, default and named arguments
- Null handling: nullable types, null coalescing, and the nullsafe operator
- A method for solving problems: read, restate, example, plan, code, test

In-class lab: Solve six warm up problems in PHP (FizzBuzz variant, word frequency counter, Egyptian phone number validator, price with VAT calculator, array grouping, palindrome check), each with a small set of manual test cases.

Homework: Finish the six problems, then solve five easy problems on a judge site such as Exercism's PHP track. Write a short note listing every status code you saw in the Network tab exercise and what each means.

Week 2 Object oriented PHP, Composer, and Git

Session 1 OOP in modern PHP

- Classes, objects, constructor property promotion, and readonly properties
- Visibility, including asymmetric visibility and property hooks in PHP 8.4
- Interfaces and abstract classes, and when each one fits
- Traits for shared behaviour
- Backed enums with methods, used for statuses and roles
- Exceptions: throwing, catching, and writing custom exception classes

In-class lab: Model a small booking domain in plain PHP: Venue, Service, TimeSlot, and Booking classes, a BookingStatus enum, a PaymentGateway interface with a FakeGateway implementation, and a DoubleBookingException thrown when two bookings overlap.

Session 2 Namespaces, Composer, PSR, and Git workflow

- Namespaces and use statements
- Composer: composer.json, require, autoload with PSR-4, and composer.lock
- PHP standards: PSR-4 autoloading and PSR-12 coding style, enforced with Laravel Pint
- Git basics: init, add, commit, log, diff, branches, and merge
- GitHub: remote repos, pull requests, code review comments, and a useful README
- Writing clear commit messages

In-class lab: Turn the previous session's booking domain into a Composer project with PSR-4 autoloading, add a package such as nesbot/carbon for dates, format it with Pint, push it to GitHub, and open a pull request that a classmate reviews.

Homework: Add a Customer class and a Coupon class with a discount enum to the domain, on a feature branch, and merge it through a reviewed pull request. Solve four problems that use classes and arrays.

Week 3 Relational databases and SQL

Session 1 Relational design

- Tables, rows, columns, and data types in MySQL and PostgreSQL
- Primary keys, foreign keys, unique and not null constraints
- One to many and many to many relations, and pivot tables
- Normalization up to third normal form, and when to break it
- Indexes: what they speed up and what they cost
- Drawing an ERD before writing code

In-class lab: In groups, design the Hagz database as an ERD (users, venues, branches, services, staff, time slots, bookings, payments, reviews), then create it with SQL DDL in MySQL 8 or PostgreSQL 17 and insert sample data.

Session 2 Querying data and talking to it from PHP

- SELECT with WHERE, ORDER BY, and LIMIT
- INNER and LEFT JOIN across several tables
- GROUP BY, HAVING, and aggregates such as COUNT, SUM, and AVG
- Transactions and why booking a slot must be atomic
- PDO with prepared statements, and how SQL injection works
- Reading a query plan with EXPLAIN

In-class lab: Write 12 SQL queries against the Hagz schema (bookings per venue this month, top rated venues, free slots for a service on a date, revenue per area). Then write a PHP script with PDO that books a slot inside a transaction and rejects a second booking for the same slot.

Homework: Add a reviews table and write five more queries, including one that needs a subquery. Show a SQL injection attack on a deliberately unsafe query and fix it with a prepared statement, in a short README.

Week 4 Laravel fundamentals and Eloquent

Session 1 Laravel 13 project structure, routing, and controllers

- Creating a project with the Laravel installer and reading the folder structure
- The request lifecycle: public/index.php, the HTTP kernel, middleware, router, controller
- Artisan commands you will use daily
- Routes, route parameters, named routes, groups, and the api.php file
- Controllers, single action controllers, and resource controllers
- Configuration and .env, and why secrets never go into Git

In-class lab: Create the Hagz repository, add a health check endpoint, and build read only endpoints for venues and services that return hard coded JSON through controllers grouped under /api/v1.

Session 2 Migrations, models, relations, seeders, and factories

- Migrations: creating and changing tables, foreign keys, and indexes
- Eloquent models, fillable attributes, and casts (including enum casts)
- Relations: belongsTo, hasMany, belongsToMany with pivot data, and hasManyThrough
- Query scopes and eager loading to avoid N+1 queries
- Factories with states, and seeders for realistic local data
- Soft deletes and timestamps

In-class lab: Translate the week 3 ERD into Laravel migrations and models with all relations, write factories and a seeder that fills the database with Egypt venues in areas such as Nasr City, Heliopolis, Maadi, and Sheikh Zayed, and switch the venue endpoints to real Eloquent queries.

Homework: Add staff and working hours tables with their relations, and an Artisan command that generates time slots for the next 14 days from working hours. Use Laravel Debugbar or query logging to show there are no N+1 queries on the venue list.

Project milestone: Hagz repo on GitHub with full schema, relations, seeded data, and working read endpoints for venues and services.

Week 5 API design, validation, and authentication

Session 1 REST API design with Form Requests and API Resources

- REST conventions: resource naming, nesting, and versioning with /api/v1
- Form Requests for validation and authorization of input
- API Resources and Resource Collections to control response shape
- Pagination, filtering, and sorting through query parameters
- A consistent error format and handling exceptions in bootstrap/app.php
- Route model binding

In-class lab: Build full CRUD for venues and services plus a public search endpoint that filters by area, category, price range, and date availability, returns paginated API Resources, and returns 422 responses in one consistent format.

Session 2 Authentication and authorization with Sanctum and policies

- Hashing passwords and why you never store plain text
- Sanctum API tokens: register, login, logout, and token abilities
- The auth:sanctum middleware and writing your own middleware
- Roles for customer, venue owner, and admin using an enum
- Gates and policies for record level authorization
- Rate limiting login and public endpoints

In-class lab: Add register, login, logout, and profile endpoints with Sanctum. Write policies so only a venue's owner can edit it and only the booking's customer or the venue owner can view a booking, then try to break them with requests from the wrong user.

Homework: Add password reset by email (logged to the log driver for now) and an owner endpoint that lists only that owner's venues. Write the API conventions for Hagz in the README: URL style, error format, pagination format.

Project milestone: Authenticated, validated, versioned API with roles and policies. Venue owners manage their own venues and services.

Week 6 Business rules, testing, and background work

Session 1 The booking flow and feature tests with Pest

- Putting business logic in action or service classes, out of controllers
- Preventing double booking with database transactions and lockForUpdate
- Booking status transitions with an enum and guard methods
- Pest setup, test structure, and RefreshDatabase
- Feature tests with factories, Sanctum::actingAs, and JSON assertions
- Testing the unhappy paths: 401, 403, 404, and 422

In-class lab: Build the create, confirm, and cancel booking endpoints with a CreateBooking action. Write Pest feature tests first for the main rules: a slot cannot be booked twice, a customer cannot cancel someone else's booking, and cancellation is blocked less than 2 hours before the slot.

Session 2 Queues, mail, notifications, storage, and scheduling

- Why slow work belongs in a queue, and running Redis locally
- Jobs, dispatching, retries, failed jobs, and the queue worker
- Mailables and Markdown mail templates
- Notifications over mail and the database channel
- File storage: local and S3 compatible disks, validating and storing uploads
- Task scheduling for recurring jobs

In-class lab: Send a booking confirmation email and a database notification through a Redis queue, add venue image upload with validation and storage, and schedule a daily job that sends a reminder for every booking happening tomorrow. Use Mailpit to see the emails.

Homework: Add payment checkout against a sandbox gateway (Paymob test mode or a fake gateway) with a webhook endpoint that verifies the signature and marks the booking paid. Cover it with at least four Pest tests, using Http::fake for the gateway.

Project milestone: Full booking flow with payments, queued notifications, image uploads, scheduled reminders, and a green Pest suite.

Week 7 Performance, API docs, and the admin panel

Session 1 Caching, performance, and OpenAPI documentation

- Finding slow queries and N+1 problems, and adding the right index
- Caching with Redis: remember, cache keys, and cache invalidation on model events
- Rate limiters per user and per IP
- Security checklist: mass assignment, authorization on every route, CORS, and secrets
- Generating OpenAPI docs with Scramble and keeping them accurate
- Writing a Postman or Bruno collection for the frontend and mobile team

In-class lab: Profile the venue search, fix the slow parts with eager loading and indexes, cache venue ratings and invalidate the cache when a review is added, and publish the API docs at /docs/api with examples for every endpoint.

Session 2 Admin panel with Filament

- Installing a Filament panel and protecting it to admins only
- Resources: forms, tables, columns, filters, and actions
- Relation managers for venue services and bookings
- Custom actions such as confirm booking or refund payment
- Dashboard widgets with stats and charts
- Reusing policies so the panel respects the same permissions as the API

In-class lab: Build the Hagz admin panel: manage users, venues, services, and bookings, filter bookings by status and date, add a confirm and cancel action that reuses the booking action class, and add a dashboard with bookings today and revenue this month.

Homework: Add reviews (only after a completed booking) with API endpoints, tests, and a Filament resource for moderation. Clean up the README so a new developer can clone, seed, and run Hagz in under 10 minutes.

Project milestone: Hagz feature complete: documented API, Filament admin, caching, and tests for every main endpoint.

Week 8 Building with AI the right way

Session 1 AI assistants on a real Laravel codebase

- How coding assistants work and why they make confident mistakes
- Setting up Claude Code or Cursor, with a project rules file (CLAUDE.md or Cursor rules) describing Hagz conventions
- Giving the assistant Laravel context, for example with the Laravel Boost MCP server
- Writing a short spec and acceptance tests before asking for code
- Working in small steps on a branch, one reviewable diff at a time
- Using AI to explain unfamiliar code and error messages

In-class lab: Add a coupons feature to Hagz with AI help: write the spec and the Pest tests yourself, let the assistant write the migration, model, endpoints, and Filament resource, then run the tests and fix what fails until all pass.

Session 2 Reviewing AI output and knowing when to stop trusting it

- A review checklist for generated Laravel code: validation, authorization, mass assignment, N+1, transactions
- Spotting invented methods, outdated syntax, and packages that do not exist
- Tests as the safety net: what a test suite catches and what it misses
- Areas to treat with care: auth, payments, migrations on live data, and anything that deletes
- Keeping secrets and client data out of prompts
- Being able to explain every line you commit

In-class lab: Review a pull request generated by AI that contains planted bugs (a missing policy check, a mass assignment hole, an N+1 query, and a non atomic booking). Find and fix each one, add a test that would have caught it, and write a short review summary.

Homework: Pick your capstone idea, write a one page spec (users, endpoints, data model, out of scope) and an ERD, and get it approved in office hours. Note one place this week where the AI was wrong and how you caught it.

Project milestone: Coupons feature merged into Hagz through a reviewed pull request, written with AI and covered by tests you wrote.

Week 9 Capstone build and real deployment

Session 1 Docker and deploying with Coolify

- Docker basics: images, containers, volumes, and a Laravel Dockerfile
- Docker Compose for local MySQL or PostgreSQL, Redis, and Mailpit
- Coolify on a VPS: connecting GitHub, building, and deploying on push
- Nginx and PHP-FPM in front of Laravel, and the production .env
- SSL with Let's Encrypt, a custom domain, and running migrations safely
- Running the queue worker and scheduler in production, logs, and daily database backups

In-class lab: Deploy Hagz to the class VPS through Coolify with a domain and HTTPS, a Redis service, a queue worker, the scheduler, and scheduled backups. Then start your capstone repo from the approved spec.

Session 2 Capstone build day

- Turning the spec into a task list and cutting scope to fit the week
- Schema, auth, and the main endpoints first
- Tests for the main business rule before polishing
- Mid day code review with the instructor
- Deploying the capstone through Coolify
- Writing the README and API docs

In-class lab: Build the core of your capstone in class with instructor reviews every hour, and deploy the first working version to your own Coolify app with HTTPS before the session ends.

Homework: Finish the capstone: all must have requirements, green tests, OpenAPI docs, Filament admin, and a live HTTPS URL. Prepare a 5 minute demo for demo day.

Project milestone: Capstone live on a VPS with HTTPS, tests, docs, and an admin panel.

Week 10 Career preparation and demo day

Session 1 CV, LinkedIn, GitHub portfolio, and freelancing

- A one page backend developer CV built around projects and results
- LinkedIn profile, headline, and posting about what you built
- GitHub portfolio: pinned repos, READMEs with live links, API docs, and screenshots
- How Laravel hiring works in Egypt, the Gulf, and remote roles
- Freelancing first steps on Upwork, Mostaql, and Khamsat: profile, first proposals, and scoping
- Estimating and pricing a small API or admin panel job

In-class lab: Rewrite your CV and LinkedIn profile with instructor feedback, polish the Hagz and capstone READMEs, and write a sample proposal for a real Laravel job post.

Session 2 Mock interviews and demo day

- Common PHP and Laravel interview questions: request lifecycle, Eloquent, queues, and service container
- SQL and database design questions
- A live coding problem solved out loud
- Explaining your project's design choices and tradeoffs
- Demo day: a 5 minute capstone demo to guests and hiring partners
- Feedback and a personal plan for the next 3 months

In-class lab: Each student does a 20 minute mock technical interview with feedback, then presents the capstone live at demo day in front of guests from the industry.

Homework: Apply to at least five Laravel jobs or freelance projects in the next two weeks, act on the interview feedback, and share your capstone post on LinkedIn.

Capstone

Each student builds and deploys their own backend from a brief: a REST API for a real local use case, with authentication, at least one non trivial business rule, background work, a Filament admin panel, Pest tests, OpenAPI docs, and a live deployment on a VPS through Coolify.

Requirements

- Approved one page spec and ERD before coding starts
- At least 5 related tables with migrations, factories, and seeders
- Sanctum authentication with at least two roles and policies on every protected resource
- Form Request validation and API Resources on every endpoint, under /api/v1
- At least one business rule protected by a transaction or a lock
- At least one queued job and one notification (mail or database)
- Filament admin panel for the main resources
- Pest feature tests for auth, the main business rule, and permission checks, all passing
- OpenAPI docs reachable from the live URL
- Live deployment on a VPS through Coolify with HTTPS, queue worker, and scheduler running
- README with setup steps, live link, docs link, and a short design notes section

Example ideas

- Restaurant ordering backend with menus, carts, delivery zones across Egypt, and order status tracking
- Tutoring center platform: groups, sessions, attendance, and monthly fee tracking for teachers
- Gym membership system with plans, class bookings, check-ins, and renewal reminders
- Real estate listings API with search by compound and area, viewing requests, and an agent dashboard
- Small pharmacy inventory and orders with stock alerts and supplier purchase orders
- Event ticketing for university clubs with QR tickets, capacity limits, and check-in

Capstone rubric

Works end to end and is live on the VPS with HTTPS	**25%**
API design, validation, and consistent responses	**15%**
Data model, migrations, and Eloquent usage	**15%**
Authentication, authorization, and security	**15%**
Pest tests covering the main rules	**15%**
Code quality, Git history, README, and API docs	**10%**
Demo day presentation and answers	**5%**

How you are assessed

Weekly homework Submitted as a GitHub pull request, reviewed against a short checklist each week	**20%**
Running project milestones Hagz milestones in weeks 4 to 7, checked live in office hours: endpoints work, tests pass, code is readable	**25%**
AI week review exercise Bugs found and fixed in the AI generated pull request, plus the quality of the tests added	**10%**
Capstone project Graded with the capstone rubric	**30%**
Mock interview and demo day Technical answers, clear explanation of your own code, and the live demo	**10%**
Attendance and participation Attending sessions, pair work, and reviewing classmates' pull requests	**5%**

Skills checklist

Web and HTTP

- [ ] Read and explain any HTTP request and response
- [ ] Choose the right method and status code for an endpoint
- [ ] Test APIs with curl and Postman or Bruno

PHP

- [ ] Write typed PHP 8.4 with strict types
- [ ] Use classes, interfaces, traits, and enums appropriately
- [ ] Manage dependencies and autoloading with Composer
- [ ] Follow PSR-12 style with Laravel Pint

Databases

- [ ] Design a normalized schema and draw its ERD
- [ ] Write joins, aggregates, and subqueries
- [ ] Use transactions and indexes correctly
- [ ] Prevent SQL injection

Laravel

- [ ] Routing, controllers, and middleware
- [ ] Migrations, Eloquent relations, factories, and seeders
- [ ] Form Requests and API Resources
- [ ] Sanctum auth, gates, and policies
- [ ] Queues, jobs, mail, notifications, storage, and scheduling
- [ ] Caching with Redis and avoiding N+1 queries

Quality

- [ ] Write Pest feature tests for happy and unhappy paths
- [ ] Review code for security and performance problems
- [ ] Use AI assistants with specs, tests, and review

Delivery

- [ ] Work with Git branches and pull requests
- [ ] Write OpenAPI docs and a clear README
- [ ] Build a Filament admin panel
- [ ] Deploy with Docker and Coolify, with SSL, workers, and backups

After graduation

Months 1-3 after graduation

- Apply for junior Laravel and PHP backend roles and internships every week
- Take one or two small freelance jobs on Mostaql, Khamsat, or Upwork to build reviews
- Add one more deployed project to your GitHub, ideally for a real small business
- Read the Laravel docs sections not covered in class: events, broadcasting, and Octane

Months 4-6

- Learn a frontend for your APIs: Livewire, Inertia with Vue or React, or a Flutter client
- Go deeper on databases: query optimization, indexes, and PostgreSQL features
- Add static analysis with Larastan and CI with GitHub Actions
- Contribute a small fix or docs improvement to an open source Laravel package

Months 7-12: toward mid-level

- Software design: SOLID, domain driven structure for larger Laravel apps, and design patterns
- Real time features with Laravel Reverb and broadcasting
- Production operations: monitoring, logging, queue scaling with Horizon, and zero downtime deploys
- Second backend skill for range: Node.js with NestJS, or Go basics
- Mentor newer students and lead a feature from spec to deploy

Where this track leads

Junior Backend Developer (PHP / Laravel) | Laravel Developer at software houses and startups in Egypt | Junior Full Stack Developer (Laravel with Livewire, Vue, or React) | API Developer for mobile app teams | Remote Laravel Developer for Gulf, European, and US companies | Freelance Laravel Developer on Upwork, Mostaql, and Khamsat | Junior DevOps or Deployment Engineer for PHP teams

Resources

- Laravel documentation: https://laravel.com/docs
- Laravel Bootcamp: https://bootcamp.laravel.com
- Laracasts: https://laracasts.com
- PHP manual: https://www.php.net/manual/en/
- PHP: The Right Way: https://phptherightway.com
- Composer documentation: https://getcomposer.org/doc/
- PHP-FIG PSR standards: https://www.php-fig.org/psr/
- MDN: HTTP: https://developer.mozilla.org/en-US/docs/Web/HTTP
- MySQL documentation: https://dev.mysql.com/doc/
- PostgreSQL documentation: https://www.postgresql.org/docs/
- Pest documentation: https://pestphp.com/docs
- Filament documentation: https://filamentphp.com/docs

- Scramble (OpenAPI for Laravel): https://scramble.dedoc.co
- OpenAPI Specification: https://spec.openapis.org/oas/latest.html
- Redis documentation: https://redis.io/docs/latest/
- Docker documentation: https://docs.docker.com
- Coolify documentation: https://coolify.io/docs
- Git documentation: https://git-scm.com/doc
- Exercism PHP track: https://exercism.org/tracks/php