# Vladislav Laikov

## Senior PHP Developer

Symfony / Laravel · distributed systems & high load

t.me/vladisdev · +375 29 163 36 69 · vladlikedev@gmail.com · linkedin.com/in/vlad-laikov-php
Remote · UTC+3

Senior backend engineer, **7+ years on PHP / Symfony**. For the last **3.5 years** my home ground has been the **money path of a product**: payments, balances and integrations; **10+ payment providers** shipped to production.

I design the money path so a payment cannot be lost, duplicated or spent twice: uniqueness is enforced by the database itself rather than by application code, lock ordering removes deadlocks instead of retries layered over the symptom, and internal and external states never diverge. Heavy operations move out of the synchronous cycle into queues, and the way the result comes back is chosen to fit the task.

I change live production without a maintenance window: major database and framework upgrades, a dialect switch and carving services out of a monolith are staged and reversible, on a ~**50 TB** cluster. Infrastructure is not someone else's job — **Docker** and **Kubernetes**, **RabbitMQ** and **Kafka**, replication and sharding, **GitLab CI** / **GitHub Actions** pipelines, alerting in **Grafana**: I carry an epic from the architectural decision to the graphs in production.

### PROFILE

**CORE COMPETENCIES: payment integrations and billing · PHP / Symfony · Laravel · PostgreSQL · high-load and distributed systems**

**DOMAINS:** **payments & billing** · **fintech core** · SaaS · MedTech · classifieds · outsourcing

### SKILLS

**Languages & frameworks: PHP 5.6—8.4**, **Symfony 3.4—7.4**, **Laravel 6—13**, Doctrine, API Platform, Symfony Messenger, **Go**, Composer / PSR, **OOP**, **SOLID** · GRASP, GoF patterns, DDD, CQRS, Hexagonal

**Data & database performance: PostgreSQL 14—18**, MySQL 5.7—8, **Redis**, **MongoDB**, **ClickHouse**, Elasticsearch, PgBouncer · query plans and composite indexes, partitioning of growing tables, sharding, logical replication, upgrades under load

**Infrastructure & operations: Docker**, **Kubernetes**, **Linux**, **AWS** (ECS, EC2, S3), Nginx, **GitLab CI**, GitHub Actions, **Grafana / Loki**, Prometheus, Sentry, ELK · metrics, logs, alerting, incident analysis down to root cause

**Distributed systems & real-time: Kafka**, **RabbitMQ**, Symfony Messenger, **Mercure**, **Centrifugo**, WebSocket, microservices, REST API / OpenAPI · webhook intake and orchestration, idempotent handlers, retries on failure

**Quality & testing: PHPUnit**, **Codeception**, **Selenium**, **PHPStan** (level 9), Rector, CS Fixer, GrumPHP · code review, staged legacy refactoring without rewrites, tests against real databases

### EXPERIENCE

Apr 2024 — Present

#### Senior PHP Developer · payments team

**systeme.io — international SaaS platform · Remote**

An international all-in-one SaaS platform for online business (email marketing, sales funnels, online courses, payment acceptance): **15M+** registered accounts and **500K+** active businesses across **170+ countries**. A high-load system: around **1B** email sends a month (**12B+** a year), ~**2B** contact records in the database, millions of published pages and funnels drawing hundreds of millions of views a month. Carving pieces of the monolith out into services — staged and reversible.

**Stack: PHP 8.3—8.5**, **Symfony 6.4—7.4**, **Doctrine**, **PostgreSQL 14—18**, **MongoDB**, **Redis**, **RabbitMQ**, **Kafka**, **Mercure**, **Docker**, **AWS**, **Grafana / Loki**, PHPStan level 9, GrumPHP

- Replaced client-side polling with push over **Mercure (SSE)**: heavy operations left the synchronous request for queues, and the interface receives an event the moment the result is ready instead of polling on a timer. Designed an observer registry for it: a new polling site plugs in as a single class, with no edits to business handlers, and an event fires only when the response actually changes. Result: **~200K redundant requests an hour** gone — **~1.5B a year**, about **90% of polling traffic**; the busiest route went from **~70K requests an hour to ~4K**.
- Owned **all 10+ payment integrations** on the platform. Delivered **Apple Pay via Stripe** end-to-end and extracted one provider **from the monolith into a standalone microservice** — its own database, its own deployment, a shared contract bundle: a new provider plugs in without touching the core.
- Designed a **partial-refund engine**: the business gave a one-line requirement — I defined the granularity (order-item level), four webhook routing scenarios, a guarded status machine and over-refund protection.
- Ran a major **PostgreSQL 14 → 18** upgrade on a ~**50 TB** cluster without pausing payment acceptance: the new database ran alongside the old one and stayed in sync through logical replication, we switched over in parts and could roll back at any step.
- Ran **PHP 8.3 → 8.5** and **Symfony 6.4 → 7.4 LTS** migrations on live production. Discipline through tooling: **PHPStan level 9**, a hard CI gate on every PR, real PostgreSQL and Redis in tests instead of mocks.

Mar 2023 — Mar 2024

### PHP Developer · CORE team

**Octopays (Mservis) — fintech platform · Remote**

An international payment provider (PSP aggregator) for the high-risk segment: payment acceptance and payouts to cards and wallets across **10+ countries**. A high-load system: ~**5M** transactions a month (**60M+** a year), **40+** payment methods and providers, **200+** active merchants. Core team, a **2 TB+** production database, real-time transaction processing, event-driven architecture (**Kafka**, millions of events a day).

**Stack: PHP 7.4—8.2**, **Laravel 9—10**, **PostgreSQL**, MySQL, **Redis**, **Kafka**, **Docker**, ELK, **PHPUnit**, **PHPStan**

- **Balances with held funds:** while an operation is still in flight its amount stays frozen on the balance, so the same money cannot be spent twice under concurrent requests — at ~**150–200K operations a day**. Payout idempotency is enforced by a **unique index in the database**: a duplicate is rejected by the database itself, not by application code; **not a single double payout since rollout**.
- **Deadlocks** on concurrent balance recalculation were removed by **lock ordering and targeted FOR UPDATE** under READ COMMITTED — prevention at the access-pattern level, not a retry wrapper over the symptom. Concurrency errors at peak hours dropped to **almost none**, and manual investigations of "stuck" operations all but disappeared.
- Financial module: a transaction moves through about ten internal statuses while the merchant sees a simple pending / success / fail; internal and external state never diverge. The result — predictable callbacks across **40+ integrations** and a **30–40% drop** in merchant queries about "unclear" statuses.
- **Laravel 9 → 10** migration and refactoring of critical payment pipelines: **PHPUnit** tests over financial scenarios (up to ~**80%** coverage), **PHPStan level 6+**, observability on **ELK** — diagnosing payment incidents went **from hours to minutes**. Took part in the **MySQL → PostgreSQL** switch on a critical contour (**2 TB+**): schema reworked for PG-specific types and indexes, staged migration with no loss of integrity and no pause in payment acceptance, legacy code synchronised with the new dialect; key transaction queries became **2–3× faster**, the slowest ones included.

Apr 2020 — Mar 2023

### PHP Developer

**Basis33 — outsourcing for EU clients · Remote**

Full-cycle custom development for EU clients, a team of **25 engineers**: high-load services with large codebases, business-process automation, a **medical SaaS platform**. I ran two products in unrelated domains, both shipped to production.

**Stack: PHP 7.4**, **Symfony 5.3**, API Platform, **MySQL**, **RabbitMQ**, Redis, Docker, GitLab CI/CD

- Grew the core of the medical platform and shipped new features. Closed the remote-appointment request with a **Zoom API integration**: doctor and patient enter the consultation straight from the booking calendar, with no links sent by hand.
- Sped up the heavy screens: query plans, **N+1** unfolded into explicit joins, composite indexes matching the real filters, schema normalisation and targeted denormalisation of hot result sets, **Redis** caching with invalidation driven by domain events. The production cluster ran with **replication and partitioning** — read paths went to the replica.
- Covered what mattered with tests: **unit tests** over domain rules, **integration tests** across the database and external services, and **functional tests** over API scenarios — regressions in booking and calendars were caught before release, not in production.

Feb 2019 — Mar 2020

### PHP Developer

**Like Systems — classifieds and data collection · Remote**

Search-driven SPA classifieds platform and an automated data collection service.

**Stack: PHP 7.1—7.4**, **Symfony 3.4 → 4.4 LTS**, Doctrine, **PostgreSQL**, Docker, Swagger, GitLab CI

- Took part in moving the platform off its legacy stack: **PHP 7.1 → 7.4**, **Symfony 3.4 → 4.4 LTS** — modules ported piece by piece, with no service downtime.
- Built the backend of a classifieds platform: a domain model shaped for search and filtering in **PostgreSQL**, a versioned **REST contract** in Swagger, automated builds and deployment through **GitLab CI**, and a collector pulling listings from external sources.

## BEYOND WORK

- I run my own products from idea to production, with no team behind me: **DDD / CQRS**, real-time over **Centrifugo**, an installable PWA.
- **Development with AI agents.** I prefer **Claude Code** and keep it on a short leash: together we built the flow from documentation and spec through to backend, frontend and deployment.

## EDUCATION

### Belarusian State University of Informatics and Radioelectronics

Minsk, Belarus

**Faculty of Computer Systems and Networks — Computer Science**

English — **B1**