Venetsian Jakimov

AI Integrations & MCP Portfolio · November 2026

Montana, Bulgaria · remote (EET, UTC+2) · vjakimov@gmail.com · linkedin.com/in/venetsian · venetsian.com/portfolio

13	~360	9+	20+
MCP servers built	tools across them	AI models routed	years shipping software

CURRENT RESPONSIBILITIES

I run DotCOM Services, a small engineering company, as its principal engineer. Day to day I:

- set product direction and architecture for our own products (NextAI-SEM, the AI Control Center, Marketing OS, MaiaHost hosting) and for client platforms;
- design and build the MCP servers and agent workflows that let AI assistants work on live business data, and own their safety model;
- write code across the stack: React/TypeScript and Next.js front ends, PHP and Python APIs, Node.js MCP servers, MariaDB schemas;
- run production: Linux servers, deployments, monitoring, backups and security reviews;
- direct AI coding agents (Claude Code, Codex, Gemini) through static-analysis gates, review steps and runbooks, the way a lead sets standards for a team.

HOW I BUILD AI THAT TOUCHES REAL DATA

1. **Preview before write.** Every write tool defaults to a dry run. Where it matters, the preview is the real write inside a transaction that is rolled back, so it shows exactly what would happen.
2. **The server owns the invariants.** Totals are computed from line items, payments need a unique reference so a retry can't double-count, and records are voided or archived rather than deleted.
3. **Rules live where an agent can't route around them**: database triggers, row locks and stale-write hashes, not instructions in a prompt.
4. **People keep the irreversible steps.** Sending, publishing, approving and deleting stay with a person or behind an explicit confirmation.
5. **Every call is audited**: who, which token, which arguments, including refused calls.
6. **Untrusted text is data.** Text typed by clients or fetched from the web is labelled as such and never treated as instructions.

SELECTED WORK

AI Control Center: self-hosted multi-agent platform

Internal platform · my daily control plane for AI work across all projects

Task tracking, agent delegation, memory, approvals, infrastructure access and content pipelines, exposed to Claude Code through six MCP servers and run by a scheduler service.

- **6 Node.js MCP servers, ~130 tools**: core (tasks, memory, approvals, policy), agents (routing, delegation, feedback, performance), infra (SSH, SFTP, database, backups), loops, content and governance. Every tool call is logged.
- **Model routing**: 12 registered agents using 9+ models (Claude, GPT-4o and Codex, Gemini 2.5 and the Gemini CLI, DeepSeek, Grok, OpenRouter, a Bulgarian-language Gemma model) with a primary → Codex → OpenRouter fallback cascade and circuit breakers. Every attempt is logged with cost and latency.
- **Memory**: a graph in 11 tables (entities, observations, events, relationships, patterns, working memory) with a daily optimizer and weekly consolidation.
- **Evals**: declared per loop and run by deterministic checks or an LLM judge, plus a weekly golden-dataset regression across agents, a model bake-off with a fabrication-drift detector, and a harness that tests every prompt template against every model.
- **Autonomous loops** fire on events, with budgets, entity locks, a dead-letter queue (3 retries), auto-disable after 5 runs with no effect, and an approval queue whose auto-approval covers only low-risk allow-listed actions. Context is classified on five sensitivity levels, with redaction and blocking before it reaches external API providers.
- **Lessons kept in writing**: 58 documented invariants. Two examples: I removed a contradiction detector that did worse than having none, and fixed a consolidator that was turning claims into facts.

Stack: Node.js (MCP SDK), PHP API (186 endpoints), React admin (33 pages), MariaDB, Python

NextAI-SEM: AI commerce and search-marketing platform

My product · an independent install per client · 220 commits since March 2026

Catalogue, AI content generation (Gemini, Claude), Google Ads with intent detection, checkout, a sales chat agent and an MCP server, for online stores and lead-generation sites. Each client gets its own install, updated through batch rollouts.

- **53-tool MCP server** (products, pages, blog, images, chat knowledge, leads and lead forms, proposals, add-ons, site rebuilds): per-user hashed tokens, tool lists gated by enabled modules, strict schemas, and every call (including dry runs and failures) written to a JSON log and a database table. The MCP test suites hold 590+ assertions.
- **Sales chat agent** (Python FastAPI, Claude Sonnet with a fallback model and prompt caching, SSE streaming). Its single tool submits an enquiry with a schema generated from the site's published lead form, and it refuses without recorded consent. Rate limits apply per IP and per visitor.
- **Multi-site federation** coordinates Google Ads keywords between sites, signed with HMAC-SHA256 and protected by a timestamp window and nonce replay checks.
- **Fleet operations**: one registry line adds a site to the scheduler, and the build refuses to publish a collapsed catalogue.

Stack: PHP 8.3, React + TypeScript, Next.js 16 (static), Python FastAPI, MariaDB, OpenLiteSpeed

MCP servers for business systems

Production servers that let AI assistants work on live company data

One reusable pattern for MCP endpoints (stateless JSON-RPC over HTTP, protocol negotiation, hashed tokens, audit helper, sandbox procedure on a scratch database), applied to each system's own risks:

System	Tools	What the AI can do	Safety design
ICEBOX Mechanical (US HVAC contractor): ERP-style operations system	**56**	Customers, projects, invoices, payments, AR aging, statements, proposals to invoices, work orders, equipment, timesheets, crew schedule, outreach email	Superadmin-only hashed tokens; totals computed server-side from line items; payment references unique per invoice (safe retries); proposals soft-delete, invoices void, removed rows archived first; anti-spam rules and a 50/day send cap enforced in the server
Billing and contracts portal (DotCOM)	**17**	Read clients, proposals and contracts; draft proposals and contracts	Draft-only writes; dry run = real write then ROLLBACK; row lock plus content-hash guard against the owner's open editor; idempotency keys; per-token rate limits; client-typed text returned as quoted data
Global Kyokushin Budokai (karate organization, Hungary): multilingual content system	**35**	Items, append-only revisions, translations, media, forms, site builds	Translations pinned to a source revision with content-hash staleness detection; review policies and approvals by a named person; per-token serialised writes; SSRF-safe media import; concurrency test suite
Hosting platform (MaiaHost), TypeScript	**16**	Sites, DNS, domains, mailboxes, databases, files, WordPress, backups, tickets	OAuth 2.1 plus personal tokens; tools filtered by role; sessions bound to their principal; confirmation gates on NS, MX, DKIM and apex records; automatic database snapshot before WordPress updates
Google Ads write server, Python	**15**	Bids, budgets, keywords, negatives, ads, locations, new campaigns	Validate-only dry runs with before/after; CPC and daily-budget caps; new campaigns always paused; no deletions; audit log mapping each change to its inverse for rollback; 64 offline tests

- **Multi-agent cross-checking**: for ICEBOX I also built a research store where several AI assistants (Claude, ChatGPT, Grok, Copilot) record what construction drawings specify. An item needs a confirmation quorum and an adjudicated verdict. After one model overwrote another's work, I added blind takeoffs and append-only revisions so nothing can be lost.

Marketing OS: evidence-gated marketing system

Internal system running research and campaigns across several brands

Research (facts, claims) is shared per market; execution (angles, creatives, offers, posts) is per brand. Nothing generative runs inside the app: AI drafts arrive as rows, and a person approves.

- **41 MCP tools** for facts, claims, personas, keyword themes, angles, offers, creatives and social posts. Every write must state its origin and defaults to a dry run built on nested savepoints.
- **Database triggers** reject an angle whose claim isn't backed by dated evidence, and block print/export until the creative, the claim and the legal check have all passed, so no agent or UI path can skip them.
- **Two separate validation axes**: is it true (factual state) and does it work (market state). A change to a claim's state requires the operator's recorded confirmation and keeps a history.
- **Tokenless loopback MCP** hardened against DNS rebinding and CSRF, with Host and Origin checks before any database work.

- **Media studio**: Remotion rendering, ComfyUI and Veo generation, edge-tts narration.

Stack: PHP API, MariaDB 11.8 (85 tables, 37 migrations), React 19 + TypeScript 5.9 + Vite 7

Generative media pipeline and local models

Local GPU workstation (RTX 5070 Ti, 16 GB) wired into the agent platform

- **ComfyUI as MCP tools** (submit, batch, status, fetch, templates, colorize, music) with 11 workflow templates: LTX-2.5 text/image/first-last-frame to video, Krea 2 stills, Qwen image editing, Qwen text-to-image, ACE-Step and MiniMax music, DDColor colorization.
- **Queued like any other agent job**: a reachability gate, grouping by model family to avoid ~30 s model reloads, adaptive polling from per-family time estimates, bounded resubmission when ComfyUI restarts, per-family concurrency caps, and seed sweeps from a single upload.
- **Production use**: 85 on-model jewellery renders for Vanilliart's online catalogue, product-in-place composites for Peletni Kamini (a heating-stove retailer), and ad video where a person approves the stills before any video is rendered. Final renders run on Colab A100s.
- **Local LLMs**: a llama.cpp CUDA build serving open-weight models (Qwen3 32B, Qwen3-Coder 30B-A3B, gpt-oss-20b, Devstral Small 24B, Mistral Small 3.2, Gemma 3 and TranslateGemma, DeepSeek-R1 distills, MiniCPM OCR). We're trying them as a private, zero-API-cost agent target alongside the cloud models.

How I work with AI coding agents

The harness behind all of the above

- **Claude Code is my main environment**, with custom subagents routed by cost: Haiku for read-only reconnaissance, Sonnet for implementation, review and server investigation. Architecture decisions stay with me.
- **Hooks run PHPStan on every edited PHP file** and block hand edits of the baseline. `check.sh` gates (PHPStan level 5 with banned calls, plus tsc) cover ~20 repositories. I introduced them after a refactor silently broke a write path for five months.
- **Per-project analysis caches** with git-based staleness checks, plus a knowledge map of runbooks, so agents start from verified context.
- **Project Manager**: a multi-agent development tracker (208 commits since October 2026, 101 test files). It prepares specs for Claude Code, Codex, Gemini or a local model and collects their results through a mailbox. It trusts the repository over the agent's report: the app runs the gates itself, and every merge needs my approval.

OTHER PRODUCTION PLATFORMS

- **Cars4Cars.eu** (vehicle-transport marketplace): loads, bids and trips; three React back offices (admin, carrier, shipper) and an Expo driver app.
- **TaxiZ** (multi-carrier taxi dispatch, taxiz.bg): ladder dispatch, OSRM routing, Expo driver and passenger apps, push and SMS.
- **JETIS** (security-monitoring back office): migration from Delphi/MSSQL with rehearsed import rules, a rights matrix (role × module × view/read/write), a real-time alarm dispatch console fed through a Python WebSocket fan-out, and an in-app feedback system (screenshots, Q&A, triage) that feeds Claude sessions.

Code walkthroughs and live demos of any of the systems above are available on request.