TECHNICAL PORTFOLIO · 2026

Noushad Hossain



Lead Software Engineer
AI, Backend & Data Platforms

Dortmund, Germany · open to relocation across Germany / EU
noushad33815@gmail.com · +49 155 67816727
linkedin.com/in/noushadhossain · noushadhossain.com
Chancenkarte (Opportunity Card) holder · available immediately

6+ yrs
Python backend, data & cloud engineering

1,150+
commits as lead on a production analytics backend (2023–2026)

4
IEEE publications (robotics, computer vision, VLC, device modelling)

INSIDE THIS PORTFOLIO

01	**Hospitality analytics platform** — FastAPI / PostgreSQL / Celery backend on AWS EKS; multi-source marketing data ingestion, automated reporting, secure CI/CD. Lead · 3 yrs
02	**Autonomous coding agent & AI reporting** — tool-calling LLM agent that turns tracked tasks into tested pull requests; LLM-generated analytics narratives with retrieval over internal APIs. Built · Led
03	**InkPigeon — self-hosted email service provider** — multi-tenant SMTP/REST sending platform with its own delivery pipeline, DKIM, bounce handling, Stripe metered billing and observability. Designed & built
04	**Further case studies** — multi-tenant CRM with Postgres row-level security; measurement-driven dbt / ETL v2 evaluation for a bronze–silver–gold data platform.
05	**Complete project index** — every repository I own or lead, with my role in each, from production platforms to R&D prototypes.
06	**Capabilities & engineering approach** — evidence-based summary, startup experience, target roles.

Confidential details of employers and clients are omitted or generalised.

PROFESSIONAL PROFILE

Hands-on lead who ships production systems end to end

Python backend · data pipelines · LLM integration · cloud infrastructure · technical leadership

I am a backend, data and AI platform engineer with six years of experience, currently the engineering lead for an AI-powered analytics platform serving the hospitality industry. I joined that company as a Senior Software Engineer in 2023, when the backend was a small team's early product, became its most active contributor, and moved into the lead role in 2025 as the platform and team grew.

My whole career has been in startups and small teams, where one person has to carry a problem from idea to production: designing APIs and data models, building ingestion from third-party marketing platforms, writing the Celery workloads that keep data fresh, producing the CI/CD pipelines that deploy to Kubernetes with security scanning, and reviewing and merging the team's pull requests every sprint. I own architecture direction — a multi-year legacy-to-modern migration program and the design of a bronze/silver/gold ETL platform — and I validate decisions with measurements rather than opinions.

Outside my employer I build and operate my own products on a small VPS footprint: a self-hosted email service provider, a multi-tenant CRM and a platform of metered API tools. These are spec-first projects where I set the architecture and engineering standards and use AI coding assistants under strict review and acceptance tests — an approach I bring to teams as well. Frontend work appears in these products only where a product needs a UI; my depth is in the systems behind it.

I come from an Electrical & Electronic Engineering background with a second major in Computer Science, four IEEE conference publications and a founding role in my university's IEEE Robotics & Automation Society chapter. I am based in Dortmund, hold a German Chancenkarte and am available immediately for roles anywhere in Germany or the EU.

CAREER TIMELINE

- **Lead Software Engineer** — Cogwheel Analytics (US, remote)
  03/2025 – present · architecture, team lead, code reviews, releases
- **Senior Software Engineer** — Cogwheel Analytics
  07/2023 – 02/2025 · backend services, data ingestion, AWS/EKS, CI/CD
- **Team Lead** — Staff Asia Ltd. (Bangladesh)
  07/2022 – 06/2023 · ETL pipelines, data warehouse, Django REST
- **Data Analyst, R&D** — Staff Asia Ltd.
  11/2020 – 06/2022 · data pipelines, dashboards, RASA chatbots, Google/Meta/Microsoft APIs
- **B.Sc. Electrical & Electronic Engineering** + second major CSE — SUST
  2016 – 2020 · 4 IEEE publications · Founding Chairman, IEEE RAS student chapter

VERIFIED TECHNOLOGY FOOTPRINT

Only technologies evidenced in the projects in this portfolio.

Python 3.11/3.12 · FastAPI · SQLAlchemy 2.0 async · SQLModel · Alembic · PostgreSQL 16 · Redis · Celery · Django REST · Pandas · pytest · OpenAI / Azure OpenAI · Anthropic API · tool calling / agents · embeddings + vector search · LangChain · MCP · AWS (EKS, ECR, S3, EC2) · Docker / Compose · Kubernetes · GitHub Actions · Snyk / Trivy · Caddy / Nginx · Prometheus / Grafana · Linux ops · dbt Core · GA4 / Google Ads / Meta APIs · WeasyPrint / Plotly · Stripe · RASA · Streamlit

LANGUAGES

English — C1, professional working language · German — B1 reading/listening, A2 speaking, actively learning towards B2 · Bengali — native

PROJECT 01 · AI-DRIVEN ANALYTICS PLATFORM

Hospitality marketing analytics platform

Cogwheel Analytics · 2023 – present · Senior Engineer → Lead Software Engineer **Lead & core contributor**

Overview	A SaaS analytics platform used by hotel groups and their marketing teams to see, in one place, how paid media, web traffic, booking sources, reviews and social channels perform across portfolios of properties, with scheduled reports, scorecards and data-quality monitoring.
Problem	Each hotel's marketing data lives in different vendor systems (Google Analytics 4, Google Ads, Meta Ads, Instagram, review sites, booking data) with different identifiers, schedules and failure modes. The platform has to ingest all of it continuously, reconcile it to the right property, and turn it into reliable reports — while the codebase grew from a small team's prototype into an 18-module product.
My role	Most active backend contributor from 2023 (1,150+ commits, ~4,000 in the repository overall); since 2025 I lead the backend team, review and merge every release, and own architecture direction, infrastructure and CI/CD.

ARCHITECTURE

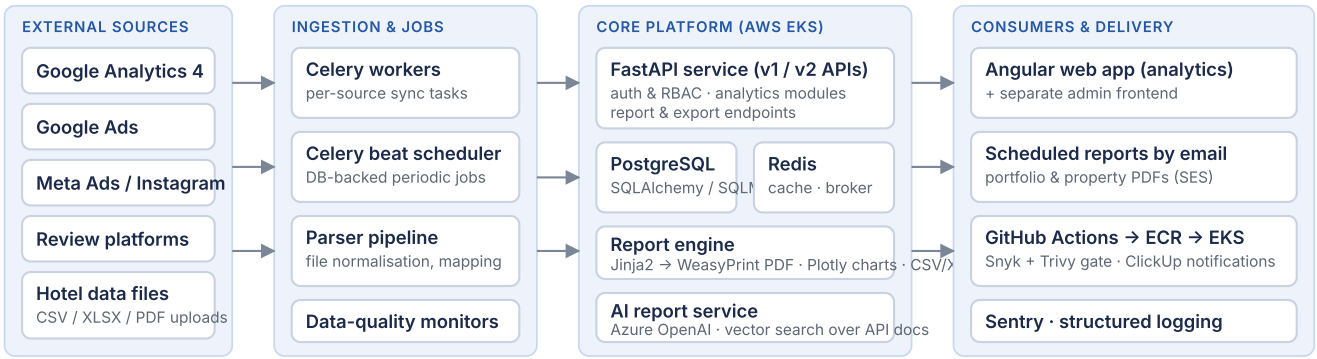


Figure 1 — Simplified system view. Source-specific sync tasks and the parser pipeline load data into PostgreSQL; the FastAPI service exposes analytics modules and a report engine; the web apps and scheduled emails consume them. Vendor names and internal module names generalised.

Python · FastAPI · SQLAlchemy / SQLModel · Alembic · PostgreSQL · Redis · Celery + beat · Pandas · WeasyPrint / Plotly · Docker · AWS EKS / ECR / S3 · GitHub Actions · Snyk / Trivy · Sentry · Azure OpenAI · pytest

MY CONTRIBUTION (INDIVIDUALLY BUILT)

- **Analytics modules end to end** — models, CRUD layer, schemas and API endpoints for channel mix, paid media, visits & revenue, source traffic and top referring domains; these are the files with the highest number of my commits in the repository.
- **Reporting API and exports** — v1/v2 report endpoints, PDF report queries (overview, paid media) and CSV/XLSX export utilities feeding scheduled property and portfolio reports.
- **Source integrations** — contributions to the Google Analytics, Google Ads and Meta ingestion code and to the persistence layer that stores vendor API responses.
- **CI/CD pipelines** — primary author of the staging workflow and co-author of the production workflow: Snyk and Trivy scans (vulnerabilities, secrets, licences, misconfiguration), a security gate with reviewer approval, findings pushed to the team's ClickUp tasks, ARM image build to ECR and rollout of the web, Celery worker and beat deployments on EKS.
- **Background processing** — Celery configuration and task wiring, including the DB-backed periodic scheduler used for all sync jobs.

PROJECT 01 · CONTINUED

What I built, decided and changed

MY CONTRIBUTION (AS LEAD, DELIVERED WITH THE TEAM)

- **Release management** — review and merge of feature, fix and sprint-release branches; ~70 sprint cycles to date.
- **Refactor program direction** — authored the May 2026 engineering brief that replaced a "migrate all 18 modules" plan (estimated at 3.5–6 years at current capacity) with a bounded target: six modules cut over, seven dead endpoint groups deleted after access-log verification, stable modules wrapped behind a proxy, the hardest rewrites deferred and planned. Each module maps to exactly one disposition.
- **ETL v2 platform** — led the design of the bronze/silver/gold ingestion architecture (idempotent runs with ingestion-run IDs, file and row hashes, versioned cleaning rules, data contracts, audit tables) and mapped it onto the team's backlog.
- **AI-powered reporting** — initiated and led the integration of LLM features into the BI product (detailed on the next page).

ENGINEERING CHALLENGES & DECISIONS

Multi-source identity and data quality

Vendors identify the same hotel differently and deliver late or partial data. We moved from ad-hoc fixes to explicit mapping tables (paid-media and source mappings, hotel-code change requests), data-completeness and data-quality monitors run as Celery tasks, and auto-resolution tasks for known failure patterns — so missing data becomes a tracked state rather than a silent gap in a report.

Security in the deploy path, without blocking delivery

Scans fail builds by default; a reviewer can approve a deployment with documented findings, and every outcome is written to the workflow summary and to the tracking task. That kept security visible without turning the pipeline into a bottleneck for a small team.

Legacy vs. rewrite

The strategic review showed the original full-rewrite plan was not deliverable. I argued for, and now run, a program that proves migration competence on low-risk modules first (auth cutover as the pilot), deletes what has no callers, and reserves rewrites for the modules where the old design actually hurts (a 3,000-line report module and a 5,000-line paid-media module).

Measure before deciding

For the transform layer of ETL v2 I ran a time-boxed dbt spike with reproducible evidence instead of a design debate (see page 8).

OUTCOME

- A production platform in continuous operation since 2023, with automated vendor syncs, scheduled reporting and a security-gated deployment path to Kubernetes.
- A written, agreed multi-year architecture direction with per-module dispositions, replacing an unbounded rewrite.
- Team process: sprint-scoped release branches, code review on every change, security findings tracked as tasks.

WHAT I WOULD IMPROVE NEXT

Finish the cutover playbook so the legacy tail is measurable; enforce data contracts at ingestion time; raise automated test coverage on the oldest modules before touching them.

PROJECT 02 · LLM INTEGRATION & AI AUTOMATION

Autonomous coding agent: tracked task → tested pull request

Personal / internal tooling · 2025 – 2026 · Python, Anthropic & OpenAI APIs, MCP, FastAPI **Designed & built**

Overview

An agent that takes implementation-ready tasks from a ClickUp list, clones the target repository, lets an LLM implement the change with tests, opens a pull request and reports back to the task. It runs as an MCP server for Claude Desktop / Cursor, as a CLI, or headless behind a FastAPI service.

Problem

A small team with a long backlog of well-specified, low-risk tickets (cutovers, small fixes, boilerplate modules). The goal was to turn those into reviewable PRs without a human babysitting an IDE session — and to know exactly what each run costs.

ARCHITECTURE & IMPLEMENTATION

- **Agentic loop on native tool calling.** The model receives the task, project instructions and a small tool set — `bash`, `read_file`, `write_file`, `str_replace` — and iterates until the change is implemented, tests are written and pass, and a PR is open. Output of each tool call is truncated to a configurable budget to keep context bounded.
- **Provider abstraction.** Claude and OpenAI providers behind one interface; switching is a single environment variable. A first version drove the Claude Code CLI from Node.js; the Python rewrite removed the external binary and calls the APIs directly.
- **Guard rails.** Priority-ordered task selection, skip-if-PR-exists, a `created_pr` tag against reprocessing, iteration/timeout limits, dry-run mode, cached per-repository instruction files.
- **Cost tracking.** Every API call records tokens and price per model/provider/day in SQLite, exposed via `/costs/*` — the cost of a PR, not just of a month.

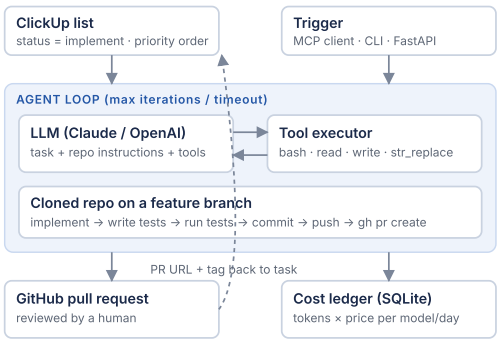


Figure 2 — The agent loop. The model only ever acts through four tools; the harness owns branching, limits, reporting and cost accounting.

OUTCOME & LESSONS

- Working end-to-end pipeline in two implementations (Node.js + CLI, then pure Python with direct API tool calling), Dockerised, with per-run logs.
- The useful unit of automation is the *well-specified* ticket: the per-repository instruction file and the test-before-PR rule mattered more than model choice.
- Next: sandboxed execution per run and automatic self-review against CI before a human sees the PR.

RELATED: LLM-GENERATED ANALYTICS REPORTS IN THE BI PRODUCT INITIATED & LED

In the analytics platform (Project 01) I led the initiative to add AI-generated narrative reports. The service embeds the platform's API documentation into Azure AI Search and uses vector search plus function calling so the model fetches property, overview and scorecard data through the platform's own authenticated endpoints before writing. The narrative is parsed into numbered sections, paired with generated charts and rendered to PDF by the existing report engine.

My part: product direction, the architecture (the API layer as the tool surface, prompts versioned as files, the report engine for output) and review. Most of the implementation was done by a team member under my lead; my own commits are in the report endpoint and prompt tooling. Lesson: grounding the model in the same endpoints the UI uses made outputs auditable — every number traces to an API call.

Python 3.11 · Anthropic API · OpenAI API · Azure OpenAI · Azure AI Search (vector) · text-embedding-3-small · MCP server · FastAPI · GitHub CLI · SQLite · Docker · Node.js / TypeScript

PROJECT 03 · BACKEND, DATA & CLOUD ARCHITECTURE

InkPigeon — a self-hosted, multi-tenant email service provider

Own product · 2026 – present · inkpigeon.com **Designed, built & operated**

Overview	A SendGrid/Mailgun-class platform that owns its entire mail pipeline with no third-party email provider: customers add a domain, verify SPF/DKIM/DMARC, then send transactional mail via SMTP relay or REST API and run campaigns and automations. The platform delivers directly to recipient MX servers, processes bounces and complaints, tracks opens/clicks, fires webhooks, offers optional hosted mailboxes, and bills through Stripe subscriptions with metered overage.
Problem	Email delivery is a hard systems problem: reputation, retries, queue semantics, tenant isolation, secrets handling and compliance all interact. The goal was a platform one engineer can operate on a single VPS, with the boring, correct choice everywhere.
My role	Sole owner: product spec (11 locked design documents covering infrastructure, conventions, a 34-table schema, API, SMTP behaviour, billing and UI), architecture, implementation, CI/CD, server operations and DNS/reputation setup.

ARCHITECTURE

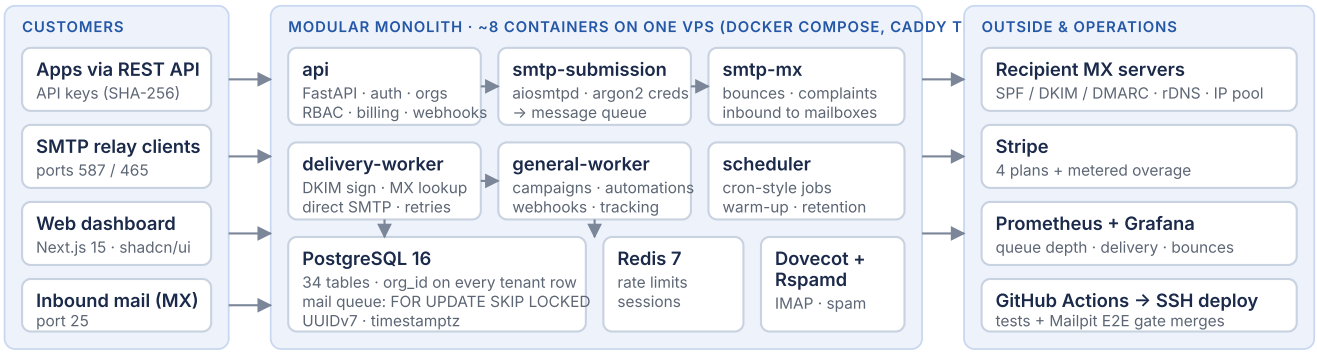


Figure 3 — One codebase deployed as independently scalable processes. The queue lives in PostgreSQL by design; Redis is reserved for rate limits and sessions.

KEY ENGINEERING DECISIONS

Modular monolith, not microservices

One Python package with strict module boundaries (modules import each other only through their `service.py`; routers stay thin), deployed as ~8 containers that scale independently. On a single VPS, microservices would add failure points without adding capacity.

PostgreSQL as the mail queue

Delivery workers claim messages with `FOR UPDATE SKIP LOCKED`. Compared with Redis/Celery this gives transactional claims, durable retry state and one fewer system to operate — at the cost of raw throughput that this product does not need yet.

Build vs. adopt

Evaluated Mail-in-a-Box, Mailu, Zimbra and Postal; all are mailbox or outbound-only products without a multi-tenant SaaS layer. Decision: reuse only the two commodity components (Dovecot for IMAP, Rspamd for spam) and build everything customer-facing.

Hosting where port 25 works

Major clouds effectively never unblock outbound SMTP; the platform runs on a Contabo VPS with Caddy for TLS, a multi-IP sending pool with reverse DNS, and Postfix configuration from an earlier iteration replaced by the in-process SMTP stack.

Tenant isolation and secrets as first-class rules

Every tenant table carries `org_id NOT NULL` and every service function takes it explicitly. Passwords and SMTP credentials use argon2id, API keys SHA-256, DKIM private keys AES-GCM; message bodies and full addresses never reach INFO logs.

Implementation, operations and outcome

IMPLEMENTATION HIGHLIGHTS

- Python 3.12, FastAPI, SQLAlchemy 2.0 async, Alembic migrations, Pydantic v2; `aiosmtpd` for inbound submission/MX, `aiosmtplib` + `dkimpy` + `dnspython` for outbound delivery.
- Capability roles (`viewer < member < admin < owner`) plus custom roles with module allow-lists enforced as router-level dependencies.
- After an adversarial review found two privilege-escalation paths in the custom-roles feature (an ungated route could mint an org-wide API key), I wrote the rule set that now governs every auth change: map the full bypass surface, no self-service on your own access, every router's gate is an explicit decision.
- 55 backend test modules; CI runs Ruff, mypy, pytest against real PostgreSQL/Redis and a Mailpit end-to-end send; merges to `main` deploy over SSH only if the suite passes.
- Prometheus metrics and Grafana dashboards; backup/restore and server bootstrap scripts; Stripe plans rotated via `lookup_key` so prices can change without code changes.

HOW IT WAS BUILT

Spec-first: the 11 design documents are the source of truth and code must match them. Implementation was carried out largely in AI-assisted coding sessions governed by a written standards file, one build-plan task per commit, and acceptance criteria that must show real test output before a task counts as done. The architecture, the rules and every review are mine; the method is one I now apply in team settings.

OUTCOME & NEXT STEPS

- Live platform with outbound ESP, hosted mailboxes and billing surfaces, operated by one person with a documented runbook.
- Pricing analysis against Resend/Postmark-class competitors showed overage rates above a defensible range; adjusting them is the next product change.
- Technical next: IP warm-up automation and a fix for a latent webhook edge case with archived Stripe prices.

Python 3.12 | FastAPI | SQLAlchemy 2.0 async | PostgreSQL 16 | Redis 7 | aiosmtpd / aiosmtplib | DKIM / SPF / DMARC | Dovecot / Rspamd | Stripe | Caddy | Docker Compose | Prometheus / Grafana | GitHub Actions | pytest / Ruff / mypy | Next.js 15 / TypeScript

FURTHER CASE STUDIES

Shorter case studies and R&D work

Intelect — multi-tenant lead CRM with defence-in-depth isolation

Designed & built

White-labelled CRM for businesses running paid ads: each client gets a subdomain, branding, team and pipeline on one shared PostgreSQL database.

- **Isolation at two layers.** An ORM hook adds the tenant filter to every query and refuses inserts for another tenant; Postgres `FORCE ROW LEVEL SECURITY` policies make even raw SQL tenant-safe. A query with no tenant context raises instead of returning everything.
- **Tests that mean something.** The suite runs against real Postgres as a non-superuser role, because superusers bypass RLS. The tenancy tests are the ones that matter: cookies don't cross tenants, reps see only their leads, raw SQL cannot read another tenant.
- **Operations.** Caddy with wildcard TLS via DNS-01, GitHub Actions → GHCR → SSH rollout, provisioning a client is a form, not a deploy; pluggable lead-source adapters (CSV first) share validation, de-duplication and audit rows.

FastAPI · SQLAlchemy 2 · Postgres RLS · Next.js 14 · Caddy · GHCR

dbt Core spike — deciding a transform layer with evidence

Built & authored

To decide build-vs-buy for the silver→gold layer of the analytics ETL v2 platform, I ran a time-boxed spike on our data shapes, our Postgres and our Celery orchestration, with every number reproducible from a runbook.

- Three gold models (daily incremental `delete+insert`, monthly, per-hotel trend) over three normalised feeds; a 5.2M-row incremental model on synthetic data at target scale.
- Found **nine** distinct ways to get silently wrong data with dbt's defaults (four serious); fixed each in a few dozen lines and pinned every one with a regression test. Enforced model contracts caught silent type widening twice.
- Clear ownership boundaries: Python owns ingestion and silver validation, dbt owns SQL and business-rule tests, Celery owns orchestration — "dbt never writes to silver, Celery never contains SQL".
- A channel-taxonomy seed (one CSV in git) resolved a long-standing reporting inconsistency independently of the dbt decision.

dbt Core 1.11 · PostgreSQL 16 · Celery · data contracts · incremental models

Roles as in git history: **owner** = sole/primary author · **prototype** = R&D, not in production.

PROTOTYPES, AGENTS & R&D

PROJECT	WHAT IT IS	STACK	ROLE
ClickUp → GitHub coding agent (2 versions)	Autonomous task-to-PR agent — Project 02.	Python / Node, Anthropic & OpenAI APIs, MCP	Owner
Insurance claim processing	Upload → OCR → field extraction → trained classifier → LangChain agent explanation over Azure OpenAI; FastAPI + SQLite.	FastAPI, OCR, scikit-learn, LangChain	Owner · prototype
Local LLM service, MCP & email agents	Small FastAPI/LangChain experiments for local model serving and agent tooling.	FastAPI, LangChain	Owner · prototype
Google Ads & analytics tooling (2023–24)	Django app and notebooks for Google Ads, GA and paid-media data migration during the platform's early phase.	Django, Google Ads API	Owner
Course-marketplace scrapers, RASA assistant, WordPress plugins	Earlier data-collection and automation work (Staff Asia era) and two small WooCommerce/WordPress plugins for a client site.	Python, RASA, PHP	Owner

COMPLETE PROJECT INDEX

Everything in my working repositories, with my actual role

Roles are stated as they appear in git history: **owner** = sole or primary author · **lead** = reviewed, merged and directed · **contributor** = commits alongside the team · **prototype** = R&D or spike, not in production.

EMPLOYER — COGWHEEL ANALYTICS (2023 – PRESENT)

PROJECT	WHAT IT IS	STACK	ROLE
Analytics backend (legacy architecture)	Core API, analytics modules, ingestion jobs, reporting, CI/CD to EKS — Project 01. ~4,000 commits, 1,150+ mine.	FastAPI, SQLModel, Celery, Postgres, AWS EKS	Lead · contributor
Analytics backend (new architecture)	Modern core the refactor program migrates modules into (auth, permissions, enterprise/brand/hotel, ownership groups). 1,000+ commits by the team.	FastAPI, Postgres	Lead · reviewer
Hotelchemy parser pipeline	Library that normalises brand-specific hotel data files (channel mix, paid media) into the platform schema.	Python, pandas, injector CLI	Lead · reviewer
AI report service	LLM narratives grounded in the platform's APIs via vector search and function calling — page 05.	Azure OpenAI, Azure AI Search	Initiated · lead
Microsoft AI foundation work	Early notebooks, LangGraph flows, evaluation flow and document-intelligence indexing that preceded the AI report service.	Azure AI, LangGraph	Owner · prototype
Cogwheel MCP server	MCP server generating tools for every endpoint in the platform's OpenAPI spec, for AI-assistant access.	Node.js, MCP	Lead (team-built)
dbt / ETL v2 spike	Evidence-based evaluation of dbt Core for the silver→gold layer — page 08.	dbt Core, Postgres, Celery	Owner
Hotel data cleaning scripts	Brand-specific cleaning and BigQuery scripts and a Streamlit tool used before the parser pipeline existed.	Python, Streamlit, BigQuery	Contributor
QA sweep, admin & web frontends, data-QA service	Team repositories I lead or review but did not author (Playwright QA suite, Angular apps, SQLModel data-QA service).	Playwright, Angular	Lead · reviewer

OWN PRODUCTS (2025 – PRESENT)

PROJECT	WHAT IT IS	STACK	ROLE
InkPigeon	Self-hosted multi-tenant email service provider — Project 03.	FastAPI, Postgres, aiosmtpd, Stripe	Owner · operator
Intelect CRM	Multi-tenant lead CRM with ORM + row-level-security isolation — page 08.	FastAPI, Postgres RLS, Caddy	Owner
Microservice tool platform	Independent metered API tools (QR, HTML→PDF, background removal) behind a Traefik gateway with a shared auth/metering/billing core.	FastAPI, Traefik, Postgres, Redis, Stripe	Owner
Education platform migrations (2)	Two WordPress/LMS sites rebuilt on a modern stack: reproducible MySQL→Postgres migration, legacy password upgrade on first login, Stripe checkout, certificates with QR verification, provider API with idempotent batch endpoints, daily link-health cron.	FastAPI, Postgres 16, Next.js, GitHub Actions	Owner
Shopify app monorepo	Queue-based Shopify→Notion order sync designed around "never lose an order", with ADRs for every infrastructure decision; ~$11/month run rate.	TypeScript, pnpm, Postgres, Dokploy	Owner · early
Email marketing backend (v1)	Multi-tenant campaign/contact backend with open/click tracking and bounce webhooks via Postal — the predecessor that led to building InkPigeon's own pipeline.	Django REST, Celery, Postal	Owner

TECHNICAL CAPABILITIES & ENGINEERING APPROACH

What I bring to a team

An evidence-based summary — every item below is backed by a project in this portfolio.

Backend & data

Python services with FastAPI and SQLAlchemy 2.0 on PostgreSQL; Celery for scheduled and background work; ingestion from vendor APIs (GA4, Google Ads, Meta) and files; reporting engines; bronze/silver/gold modelling; dbt where it earns its place.

AI & LLM engineering

Tool-calling agents with bounded loops and cost accounting; retrieval grounded in the system's own APIs; versioned prompts; LLM output rendered through existing, auditable report pipelines.

Cloud, DevOps & security

Docker everywhere; Kubernetes on AWS EKS with ECR; GitHub Actions pipelines with Snyk/Trivy gates and human approval; Caddy/Nginx TLS; Prometheus/Grafana; backups and runbooks; secrets hygiene (argon2id, SHA-256, AES-GCM) and adversarial review of authorization changes.

Leadership

Code review and release management across sprints; written architecture direction with explicit dispositions and estimates; mentoring; aligning backlog to design documents.

How I work

- **Spec first, then code.** Locked design documents beat opinions in a PR thread.
- **Measure before deciding.** Spikes with reproducible evidence, not debates.
- **Boring, correct defaults.** Postgres over a new system; a monolith until scale demands otherwise.
- **Security is a decision, never an omission.** Every route's gate is explicit.
- **Own it in production.** Deploy it, monitor it, write the runbook.

STARTUP EXPERIENCE

I joined Cogwheel Analytics when its backend was an early product and a handful of engineers, and grew with it into the lead role as the platform, customer base and team expanded. I know what it means to carry a feature from a customer conversation to a deployed, monitored endpoint in the same sprint — and when to stop and write the spec instead.

My own products are run the same way: a locked spec, the most boring correct choice, a single VPS, a runbook, and billing from day one. For a founding or early-stage role that is the combination I offer: depth in backend, data and AI systems, and the habit of owning them in production.

Target roles: Senior / Lead Backend Engineer · AI Engineer · Data Engineer · Platform Engineer · Founding Engineer — Germany or EU, on-site, hybrid or remote. Available immediately; Chancenkarte holder based in Dortmund.