Engineer Software

{ Backend & API Development }

The engine behind your app or website

The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

Still working out what you need?

Tell us what is going wrong, or what you are trying to build. We will tell you what it would take, roughly what it would cost, and whether we are the right people for it.

From 1,400 DKK an hour, excluding VAT — and less per hour on a longer booking.

Get in touch See what it costs

Selected work demonstrating this service.

Turn the service into a result

Make “Backend & API Development” serve a business outcome. We can connect the technical work to the users, operations and decisions it needs to support. The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

Get in touch

Examine the relevant work

Use the related work on this page to examine our approach. Ask about the tradeoffs and the conditions behind each result. Designed and ran 6 ETL/ELT pipelines — 6 ETL pipelines across three database engines

Designed and ran 6 ETL/ELT pipelines Get in touch →

Designed and ran 6 ETL/ELT pipelines

Designed, implemented, and administered 6 ETL/ELT pipelines, utilizing Google BigQuery, MSSQL, PostgreSQL, Shell scripting, PL/pgSQL, and Transact-SQL, integrating data for efficient Python API processing.

Delivered 8 Power BI projects with manuals

Delivered 8 Power BI projects with comprehensive manuals, integrating Microsoft Power BI tools with NodeJS API and Python FastAPI for effective data analytics and visualization.

Directed full-stack GIS map development

Directed full-stack GIS map development, overseeing PostgreSQL, Mapbox, ReactJS, and NodeJS to deliver an integrated solution.

Led 30+ GIS projects from build to support

Led the development, deployment, and support of over 30 GIS projects, demonstrating expertise in PostgreSQL, Bash, Python, JavaScript, GDAL, ArcGIS, PostGIS, and Mapbox technologies.

Developed a Data Analytics reporting system

Developed a Data Analytics reporting system, increasing quarterly software revenue by 400% through Python-based PDF reports.

A maritime platform with clear system boundaries

Architected a layered maritime platform separating a Next.js PWA frontend, a Go (Huma/Fiber) API, and a PostgreSQL function layer, keeping all business logic in the database.

Designed a JSON passthrough from Postgres to API

Designed a JSON passthrough architecture where PostgreSQL functions return complete JSON forwarded verbatim by the Go API, eliminating intermediate unmarshalling and decoupling the frontend from schema changes.

Built organization context switching for users

Designed an organization context-switching system with client localStorage and server-side cookie mirroring, letting users act as managed organizations while enforcing least-privilege authorization.

Migrated the API to Huma v2 at 100% parity

Migrated the HTTP API from Fiber to Huma v2 — 649 paths and 760 operations — reaching and holding 100% parity between the routes the server registers and the OpenAPI description it publishes.

Generated the API contract from the database out

Generated the API contract outward from the database — OpenAPI, a 44,076-line typed TypeScript client, 61 mock handlers and the limits the UI enforces — with a guard at every hop that fails on drift.

Built email with failover and delivery logging

Built email as a platform capability — three providers with failover, delivery webhooks, send and delivery logging, templating and campaigns — behind a startup check that will not boot without one.

Designed a function-first PostgreSQL data layer

Designed a PostgreSQL function-first data layer — 1,275 stored functions across 34 schemas — so every read and write goes through a function the database can grant, rather than through a table.

Implemented a catalogue-driven deep-merge

Implemented a catalogue-driven deep-merge for stored JSON preferences, preventing missing-key crashes as the schema evolves.

Hardened the application with nonce-based CSP

Hardened the application with nonce-based CSP, HSTS, SameSite cookies, least-privilege database roles and server-side entitlement re-checks.

Set the platform's founding technical decisions

Set the platform's founding decisions in the weeks after the repository opened in November 2025 — the layering, database-first data access and the zero-warnings bar — and they still hold nine months on.

Built and maintained Django apps for IPTV

Developed and maintained Django web applications for the IPTV platform, shipping new features and improving performance and stability.

Built a test suite across Go, SQL and browser

Built a layered automated test suite — 981 Go tests, 543 frontend and browser specs, 494 SQL behavioural tests — with mutation testing, property-based tests and an accessibility gate.

Built Stripe and in-app purchase entitlements

Built the payments and entitlements layer — Stripe alongside Apple and Google in-app purchase — gating the directory, search and export through an 11-table access model checked on the server.

Moved slow work onto a River job queue

Moved slow work off the request path onto a River job queue — 15 worker modules, 8 scheduled tasks and 20 pg_cron jobs — so a request returns while the work behind it carries on.

Built first-party error monitoring and tracing

Built first-party error monitoring and OpenTelemetry tracing rather than buying them — payload sanitising, spike and regression detection, symbolication and a synthetic heartbeat — behind 11 operator views.

Built fail-closed rate limiting and anti-abuse

Built fail-closed abuse controls — 22 Redis-backed rate limiters, Cloudflare Turnstile, request idempotency and an origin lock — so the platform sheds bots and floods instead of trusting its callers.

Built a loyalty wallet, leagues and badges

Built the loyalty and reputation system — 67 functions over a 31-table ledger, with leagues, badges and a redemption shop — taking a row lock on the balance to close the double-spend window.

Built the social layer: posts, feed and groups

Built the platform's social layer — posts, feed, groups, mentions, a follower graph and an occasions digest — on the same function-first data layer as the rest of the product.

Built the hiring and seafarer career workspaces

Built hiring and seafarer career workspaces; the recorded implementation snapshot contains 151 stored functions across 48 tables for vacancies, applications, certificates, rank progression and sea time.

Built a database-driven CV generator

Built a database-driven CV, references, portfolio and cover-letter generator in Python — 41 modules, 10,580 lines — rendering six output formats from one 23-table SQLite source assembled by a 19-step idempotent pipeline.

Held the generator to 981 tests and 92% coverage

Held the generator to 981 test cases at a 92% branch-coverage floor with warnings treated as failures, and asserted idempotence by running the whole build pipeline twice from an empty file and requiring the second pass to change nothing.

Enabled every linter rule as an error

Selected every rule the Python linter has as an error, working through 1,815 findings to reach zero, with each of the few exemptions carrying a written reason and two of them backed by a checker instead of a comment.

Vendored a QR encoder in 522 lines

Vendored a QR encoder — Reed-Solomon over GF(256), fixed module layout, eight mask patterns — in 522 lines rather than take a third runtime dependency, and verified it by reading the finished matrix back with an independently written decoder.

Moved every string into a content tree

Moved every user-facing string out of Python into a content tree of 874 files across three languages, after finding dead translations nobody could see were dead and a check silently grading a third of the achievements.

Adopted Swift 6 strict concurrency over C

Adopted Swift 6 complete strict concurrency with no actors, bridging a blocking C event loop to the main actor through one producer, one consumer and one ordering — after establishing that a task per event loses the ordering the interface depends on.

Wrote a parser verifying every FFI call

Wrote a parser that reads the real 7,308-line C header and verifies every call site, every enum constant and that every pointer-owning class is final, after a hand-written placeholder header let calls to three removed functions compile, link and crash.

Reached the unused half of the core

Reached the half of the messaging core the application had never used — backup transfer, disappearing messages, message editing and resending, verified invitations, proxies and encryption policy — driving every test against the real library with no mocks.

Looking for Backend & API Development?

Use the linked work to judge relevant experience. Tell us what you need to build; we can discuss the scope, delivery approach and handover.

Get in touch Full portfolio

Scope the work before commissioning it

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Talk through your requirements

Tell us what you need from “Backend & API Development”, including the deadline and the team that will maintain it. We can discuss a practical way forward.

Get in touch

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://software.engineer.company/services/backend-development/