

Ilya Mois

Senior Full-Stack Engineer · Remote or on-site in Tbilisi, Georgia (GMT+4)

job.offers@mois.pro · github.com/mois-ilya · linkedin.com/in/moisilya · Russian (native), English (professional)

SUMMARY

I have been building web products since 2018, most recently developer platforms and payments. At Tonkeeper, I owned Tonconsole, TonAPI's developer console, for 21 months, including its billing and documentation, and maintained TonAPI's TypeScript SDK, which has about 16k weekly downloads. In 2026, I built Diwy Split on my own, a Telegram Mini App that splits restaurant bills from receipt photos: from the interface and LLM recognition pipeline to the server it runs on. It went from prototype to launch in under a month of active work.

I build LLM pipelines and release prompt or model changes only after automated evals against ground truth show no regression. I develop with coding agents inside a workflow of test scenarios, multi-model review and screenshot tests in CI, and nothing reaches users until every check passes.

EXPERIENCE

Diwy Split — side project

Mar 2026 – Present · diwy.me · Telegram Mini App that splits restaurant bills from receipt photos · solo, live in beta

- **LLM recognition** — chose from about 20 model pairings by evals on labelled receipts: 0.991 at \$0.004 per receipt. The model only transcribes; code does the arithmetic and flags any receipt whose total doesn't add up.
- **Testing** — edits are stored as events, so concurrent changes survive and bills replay from history. Replayed events build 77 bill states for pixel-exact screenshot tests in CI; property-based tests check the settlement math against a reference implementation.
- **Operations** — kept the app reachable in Russia, where Cloudflare is throttled. GeoDNS sends users there through a TLS-passthrough relay and everyone else through Cloudflare. External probes watch both routes and alert to Telegram, and data streams continuously to R2.
- **UX** — prototyped the flows before writing code so a paper receipt becomes a screen with no dead ends, including taking only part of a dish, with a receipt editor for what recognition gets wrong.

Tonkeeper — Software Engineer, Developer Platform & Developer Experience

Jun 2024 – Mar 2026 · Remote · TON Apps: 23-person team behind the Tonkeeper wallet (12M+ MAU). Most work is public, linked below.

- **Tonconsole**, TonAPI's developer console — moved billing to one USD balance matching plan prices, with TON or USDT top-ups. Migrated state to TanStack Query incrementally, isolating caches by user so accounts in one browser never share data.
- **Console products** — shipped webhooks, dedicated liteservers, API plans and analytics. One tested pricing function made a threefold webhook price change a constants edit. Liteserver buyers download `global.config.json` without manual edits. Downgrades show the unspent balance instead of being blocked.
- **Free TonAPI limits for dApps** — took an idea I was given and designed and built it on every wallet platform, including QA's test setup. The wallet injects `window.tonapi.fetch`; the SDK uses it without configuration, giving dApps inside Tonkeeper higher limits.
- **TypeScript SDK, ~16k weekly downloads** — sole maintainer from mid-2024. Added bigint, address and cell formats to TonAPI's [OpenAPI schema](#), giving any generator exact amount types. Kept old SDK paths as deprecated wrappers when the schema was regrouped.
- **Integration docs** — generated the REST reference from the same schema. Wrote the transaction-tracking and data-signing guides and the cookbook.
- **Wallet** — replaced polling of two DEX aggregators with streamed Omniston swap quotes. Built desktop staking ([merged after I left](#)). Changed TRC20 sending errors from "Insufficient funds" to "temporarily unavailable" during TronGrid outages. Fixed slow server rendering in Tonviewer, the explorer.
- **Fees and wallet extensions** — wrote a fee estimator using TON's formulas, tested against real transactions ([unfinished when I left](#)). Worked on sponsored-fee (gasless) transfers and W5 wallet extensions, including 2FA.

Quintegro — Frontend Developer

Jul 2023 – May 2024 · Remote

- Led development of the booking-change module for a reservation platform connecting hosts and guests.
- Defined frontend-backend API contracts, generated TypeScript types from them and set up CI.

Minacu — Full Stack Engineer

Apr 2023 – Jul 2023 · Part-time, Remote

- Built streaming speech-to-text over gRPC, text analysis through the OpenAI API and a Telegram integration through tdlb — Python and NestJS on the backend, Next.js on the frontend.

Upper Echelon Products — JavaScript Developer, then Technical Team Lead

Apr 2021 – Mar 2023 · Remote

- Promoted to team lead: ran 3–5 engineers and 1–2 QA through about 11 releases, mentored five juniors.
- Built an Amazon Seller Central ingestion pipeline and moved the backend from a VPS to AWS with CI/CD. Standardised the frontend architecture and moved API documentation to Swagger.

Apptech — Software Engineer

Jul 2020 – Nov 2020 · Part-time, St Petersburg

- Built push notifications and real-time messaging on Firebase for the second version of a messenger app.

GET Information Technology — Software Engineer

Mar 2018 – Feb 2020 · St Petersburg

- Built Backbone/Marionette and ExtJS interfaces and the ColdFusion services that supplied their data.
- Introduced ESLint and Babel, including tooling that tracked their rollout across the codebase.

SELECTED OPEN SOURCE

- **TON Connect SDK**, the library dApps use to connect TON wallets — implemented [data signing](#) for text, binary and TON cell payloads across the protocol, SDK and UI. Added wallet capabilities: [required features](#) filter out unsuitable wallets, [preferred features](#) rank them and prompt a reconnect instead of failing silently.
- **TON Connect specification** — wrote a [sign-data reference implementation](#) with tests. They exposed two places where compliant wallets could sign different messages for the same request, both fixed in the spec: the response [address format](#) and [appDomain encoding](#), where the spec's own example was wrong.
- **Telegram Bot Docs skill** — an agent skill that splits Telegram's bot documentation into about 1,000 searchable sections with a freshness check, so coding agents read current docs rather than memory.

SKILLS

- **Languages & frontend** — TypeScript, JavaScript, Python; Go (reading and small fixes); React, Next.js
- **Backend & APIs** — Node.js, Fastify, NestJS, REST, OpenAPI, GraphQL, gRPC, webhooks, OAuth/OIDC, JWT
- **Data & messaging** — PostgreSQL, MySQL, MongoDB, Redis, RabbitMQ, event sourcing
- **Cloud & ops** — Cloudflare (Workers, Durable Objects, R2, Pages), AWS, Azure AI, Docker, GitHub Actions, NixOS, Prometheus, Grafana
- **Testing** — Vitest, Playwright (pixel-exact screenshot tests), fast-check, Testcontainers
- **Domains** — developer platforms and SDKs · payments and billing · Telegram Mini Apps and bots · blockchain

EDUCATION

MSc, High Technology and Economics of Innovation

2019 – 2021 · ITMO University

BSc, Information Systems and Technologies

2015 – 2019 · Herzen State Pedagogical University