



From Schema to Shipped

Nouralden Alhouni — full-stack software engineer based in Tripoli, Libya. I design and build production systems end to end: database schema, backend, frontend, and the infrastructure that ships and runs them.

noralden.elhouni@gmail.com

github.com/NoraldenElhouni · linkedin.com/in/nouralden-alhouni

Tripoli, Libya · MIS, Eastern Mediterranean University (EMU), North Cyprus –
Sep 2025

What I build, and how

Over the past year I designed and built twelve production systems solo — enterprise ERPs, booking platforms, mobile apps, and the infrastructure behind them — most still in active, daily use. I work across the full stack: relational schema design, backend APIs and business logic, frontend UI, and the deployment pipelines that ship all of it.

13

production systems shipped

1,469+

commits across every codebase

380+

screens & pages built

5

platforms — web, desktop, iOS, Android, API

Languages

TypeScript

JavaScript

Python

PHP

Dart

SQL

Backend & Data

Supabase

PostgreSQL

Neon

FastAPI

Laravel

Prisma

SQLAlchemy

Deployment & Infra

Vercel

Cloudflare Workers

Docker

GitHub Actions

cPanel

LibyaSpider

Frontend

React

Next.js

React Native / Expo

Flutter

Electron

Tailwind CSS

Mobile & Desktop

EAS Build

Electron Forge

Auto-updaters

App Store release

Practice

Git

Zod

TanStack Query

RLS / permission design

Search Console

01 — 04

Enterprise Systems & ERPs

Multi-schema, permission-gated back-office systems running real business operations — finance, HR, procurement, contracting, and manufacturing.

EGC Management

Desktop ERP running finance, HR, operations, and contracts for a construction company

The core internal management system used daily by staff at EGC, a construction and engineering firm. Covers company finance, HR and payroll, project operations, procurement, and the full contractor-contracting lifecycle — from pricing rounds through comparison, award, and payment milestones.

Timeline

Nov 2025 – Sep 2026

Role

Solo — design to deploy

Platform

Desktop, Windows & Mac

Electron	React 19	TypeScript	Vite	Supabase / Postgres	TanStack Query	Tailwind CSS
----------	----------	------------	------	---------------------	----------------	--------------

What's built

- Company finance — treasury, payments, contracts, bookkeeping
- Full contracting workflow: pricing rounds → contractor quotes → side-by-side comparison → award → milestone payments
- Bill-of-quantities engine with a reusable project-template catalog
- HR, payroll, and departmental structure
- Procurement and vendor ordering
- A five-layer permission system governing exactly what every employee, contractor, and vendor can see and do

129

screens built

112

database tables

6

data schemas

449

commits over 10 months

Engineering note

The permission system is the standout piece of engineering here. Instead of hardcoding who can do what, every access decision is resolved through one five-layer rule engine (role → department → user → project → team) — so a screen with dozens of individual permission checks now costs one data request instead of dozens. Rolled out in phases against real, live data, with a full audit to make sure nothing was missed.

Fields — EGC Builder

Mobile companion to the desktop ERP — field access for staff, contractors, and vendors

The mobile counterpart to EGC Management, giving staff, contractors, and vendors role-appropriate access to projects, contracts, and payments from a phone — built for people working on-site rather than at a desk.

Timeline

Sep 2025 – Sep 2026

Role

Solo — design to deploy

Platform

Mobile, iOS & Android

React Native	Expo	Expo Router	NativeWind	Zustand	Supabase	EAS
--------------	------	-------------	------------	---------	----------	-----

What's built

- Three separate experiences in one app — staff, contractors, and vendors each see only what applies to them
- Contract, milestone, and payment tracking on the go
- Material ordering with cart and checkout
- One-tap contractor directory and calling
- Push notifications
- The same five-layer permission engine as the desktop app, so access rules never drift out of sync

61

screens built

197

commits, active every month

3

role-based experiences, 1 app

12

months of continuous work

Engineering note

Rather than duplicate access logic between desktop and mobile, both apps are wired into the exact same permission engine — one request resolves an entire screen's worth of what's visible, instead of checking permission-by-permission. That shared foundation is what keeps two separate apps from quietly drifting apart over time.

EGC-Solid Back Office

Multi-tenant finance & asset back office for a construction business line

A back-office system for Solid, a construction business line — tracking projects, client finances, company expenses, and equipment, with strict per-company data isolation built in from the database up.

Timeline	Role	Platform
Jul – Aug 2026	Solo — design to deploy	Web
Next.js 15	React 19	TypeScript
Supabase / RLS	shadcn/ui	Cloudflare Workers

What's built

- Multi-company structure with company-switching and scoped access
- Project finance: deposits, expenses, payments, with automatic status tracking
- Automatic revenue-percentage calculation on every payment
- Equipment and asset tracking with a full movement ledger
- Finance report PDF export

20

database tables

19

screens built

74

UI components

26

commits, intensive build

Engineering note

The finance engine does real domain modeling, not just data entry: every payment automatically calculates and logs the project's revenue-percentage cut, and every equipment movement is forced through a single database-level function so quantities can never be silently corrupted by an app-level bug. Combined with strict per-company data isolation enforced by Postgres RLS, correctness lives in the data layer, not just the interface.

FINE ERP

A five-unit manufacturing ERP, built and shipped in under two months

An enterprise ERP for a foam-to-furniture manufacturer, covering procurement, foam production, cutting, furniture assembly, and showroom sales as five connected operating units, used daily by staff across the business.

Timeline	Role	Platform
Jul – Sep 2026	Solo — design to deploy	API + Desktop
Laravel 13	PHP 8.4	Sanctum
	Electron	React 19
		TypeScript
		TanStack Query

What's built

- Five operating units on one shared system: procurement/treasury, foam manufacturing, cutting, furniture assembly, store/showroom
- Full inventory: items, warehouses, stock lots, valuation, movement tracking
- Manufacturing: production batches, work orders, bills of materials
- Accounting: chart of accounts, journal entries, trial balance, financial reports
- HR: attendance, payroll, leave requests

73

data models

166

API routes

2

apps, built in ~7 weeks

186

commits

Engineering note

With five operating units sharing financial and inventory records, the real risk is two people overwriting each other's changes at the same time. Every sensitive record — sales orders, journal entries, stock adjustments — carries a version number that's checked and updated atomically, so a conflicting update is rejected outright rather than silently lost. That's the difference between a system that only looks multi-user and one that actually is.

05 — 07

Platforms

Two-sided platforms handling real transactions — bookings, submissions, and dine-in orders — under real concurrency and deadline pressure.

Hotel Booking & Operations Platform

A full hotel booking and operations platform, multi-property and multi-tenant

Customers browse and book stays online, hotel staff run day-to-day operations and finance per property, and a platform-wide console oversees everything across every hotel on the system.

Timeline

Jan 2026 – Sep 2026

Role

Solo — design to deploy

Platform

Web

Next.js 15	React 19	TypeScript	Supabase	Postgres RPC	Recharts	Puppeteer / Chromium
------------	----------	------------	----------	--------------	----------	----------------------

What's built

- Customer booking flow with five different Libyan payment options
- Full property operations: rooms, live availability calendar, reservations, mid-stay room moves
- Finance module: expenses, salaries, income, auto-numbered invoicing
- An insights dashboard — KPIs, monthly trends, cancellation impact, and more
- Company-wide analytics across every property on the platform

236

commits — busiest codebase

28

database tables

45

screens built

8

months, sustained build

Engineering note

The booking engine solves a genuinely hard concurrency problem: when two people try to book the same room at the same moment, database-level locking makes sure only one wins, and a "busiest night" algorithm correctly reserves inventory for long stays instead of over-blocking rooms. A recent addition extends this so a guest can be moved mid-stay to a different room without corrupting the nightly revenue records — a real bug, found and fixed.

LIA Competition Platform

A full submission platform for a national student architecture competition

A registration and submission platform for "Libyan House," an architecture design competition for university students across Libya — handling team formation, multi-file design submissions, deadlines, and public results.

Timeline

Sep 2025 – Feb 2026

Role

Solo — design to deploy

Platform

Web — bilingual

Next.js 15

React 19

TypeScript

Supabase

next-intl

tus resumable upload

What's built

- Student registration, academic verification, and team formation (1–4 members)
- A multi-step submission wizard for boards, 3D models, diagrams, and reports
- Live admin dashboard tracking submissions in real time, by university
- Public results page with award tiers — first, second, and third place plus two honorable mentions
- Submitted to Google Search Console for public visibility

Launch & reach

1,000+

page views on launch day

522

visitors on launch day

19

projects submitted

6

universities submitted from

139 students registered across 19 universities over the competition's life — a wider draw than the 19 that ultimately submitted, which tracks with how open student competitions usually go.

Engineering note

The submission pipeline does real data-integrity work: before a submission is accepted, it cross-checks team composition, verifies every member's university, and confirms ownership — and if anything fails after files are already uploaded, it automatically cleans up every uploaded file so nothing is left orphaned. Paired with the live admin dashboard, this held up through a real, time-pressured competition deadline and a launch day that pulled over 500 visitors.

MEGDA QR

QR-code dine-in ordering — scan, browse, order, no app or login required

A two-sided restaurant platform: diners scan a QR code at their table to browse the menu and order from their phone, while restaurant owners and staff manage branches, menus, tables, and incoming orders from a companion dashboard. Two apps, one shared Supabase backend.

Timeline	Role	Platform
Mar 2026 – Sep 2026	Solo — design to deploy	Mobile/desktop dashboard + web ordering
Flutter	Dart	Next.js 16
	TypeScript	Supabase
		Zustand

What's built

- Manager dashboard: branch, staff, menu, table, and order management, with QR code generation per table
- Customer site: scan → live table session → menu browsing → cart → order submission, no login required
- Shared, per-table order visibility — everyone at the table sees what's been submitted, without seeing each other's private carts
- Rate-limited session creation and a signed session cookie, re-validated server-side on every request
- Subscription/plan structure for restaurant billing

Two repos at different stages: the manager dashboard is furthest along, with core flows complete end to end; the customer ordering site is newer, with checkout and payment integration still ahead. Row-level security is intentionally off project-wide as a tracked pre-launch item, not an oversight.

23

database tables

32

screens across both apps

81

commits across 2 repos

Engineering note

The shared-session, private-cart design is the standout piece here: when multiple diners at one table scan the same QR code, they're joined into a single session via a find-or-create RPC — avoiding duplicate, competing sessions — while each phone keeps its own local cart and everyone at the table sees a live, shared view of what's already been ordered. No accounts needed. It's backed by a signed session cookie that's re-validated server-side on every request rather than trusted at face value, with the rejected design alternatives documented alongside it.

08 — 11

Client-Facing Products

Public sites and authenticated portals — from marketing pages with self-service content, to secure client-only apps.

EGC Client Portal

Letting a construction firm's clients check in on their own projects, any time

A dedicated app for EGC's clients, built so they can log in and see the live status, details, and financial balance of their own projects without needing to call the office.

Timeline	Role	Platform
Built Jun 2026, resumed Aug 2026	Solo — design to deploy	iOS, Android & Web
React Native	Expo Router	react-native-web
Supabase Auth	EAS	

What's built

- Secure client login, provisioned by EGC — not self-serve sign-up
- Full Arabic/RTL interface, one codebase shipping to iOS, Android, and web
- Project list and detail view with live status and balance

Submitted to the Apple App Store; reviewer feedback asked for account deletion and more complete features before a few "coming soon" screens. A clear remediation plan is written and that work is back in progress.

6

screens built

3

platforms, one codebase

Engineering note

Login here does more than check a password — a client only gets access once their account is actually linked to a real client record in the system, so signing up alone can't grant someone access to project data. It's a deliberate safeguard, not an accident.

EGC Website

A firm's public site, migrated onto the same live backend as its internal ERP

EGC's bilingual marketing and portfolio site, showcasing projects, founders, and services to prospective clients — recently migrated so its content is managed live from the same company database as the internal systems, instead of hardcoded.

Timeline

Oct 2025, reactivated Aug-Sep 2026

Role

Solo — design to deploy

Platform

Web — bilingual, Arabic default

Next.js 16	React 19	TypeScript	next-intl	Supabase
------------	----------	------------	-----------	----------

What's built

- Full bilingual site: home, about, services, founders, projects, gallery, contact
- Projects and photos pulled live from the company database instead of hardcoded
- A locked-down, read-only public connection into that database
- Bulk content-seeding tooling for uploading bilingual project text and photos

10

page variants, 5 pages × 2 languages

4

public content tables

32

commits

Engineering note

Rather than stand up a separate database for the website, it was connected directly to the firm's existing company database — locked down so the public site can only ever read a handful of approved tables. That was verified by actually testing that write attempts get rejected at the database level, not just trusted in the app code. One source of truth, safely exposed to the public.

SOLID Construction Benghazi

A marketing site with a small, purpose-built CMS behind it

The public marketing site for Solid Construction Benghazi, showcasing completed and in-progress projects, with a built-in content editor so staff can update page text without needing a developer.

Timeline	Role	Platform
Jun – Sep 2026	Solo — design to deploy	Web
Next.js 16	React 19	TypeScript
	Prisma	Clerk
		Neon Postgres

What’s built

- Public site: home, about, services, full project catalog with detail pages
- Project leads captured via contact form and newsletter signups
- Admin CMS to edit page text directly, with a live preview before publishing
- Secure, role-gated admin access

9

data models

15

screens built

25

commits, concentrated build

Engineering note

The content system is a small, purpose-built CMS rather than a bolted-on plugin — a generic content model lets staff edit any text block on the public pages through one form, with changes going live immediately and a live preview so nothing gets published blind.

AI Shoroog.ly

A content-rich product catalog for an agricultural inputs company

A marketing and product-catalog website for AI Shoroog, an agricultural inputs company, showcasing fertilizer and crop-nutrition products with a detailed, structured product-information system for farmers and distributors.

Timeline	Role	Platform
Jun 2026	Solo — design to deploy	Web
Next.js 16	React 19	TypeScript
	Tailwind CSS	Cloudinary

What’s built

- Full product catalog — 19 products across 4 categories, with filtering
- Rich per-product detail pages: specs, application guides, compatible crops, FAQ
- An interactive, crop-based product recommendation tool
- Company info, testimonials, and contact/newsletter capture
- Submitted to Google Search Console for public visibility

19
catalog products

5
pages

2
days, start to finish

Engineering note

The product catalog is built on a genuinely rich, hand-structured dataset — every product carries full specs, an application guide, and compatible-crop data, driving a real product-information experience rather than placeholder marketing text.

12 – 13

Infrastructure & Deployment

The services and pipelines that ship everything above, and keep it running.

PDF Service

A shared document-generation microservice, built to serve every other system

A dedicated backend service generating official Arabic business documents across several of the systems above — invoices, BOQ reports, contracts and payments, expense reports, hotel reservation documents — with every request logged for accountability.

Timeline	Role	Platform
Mar 2026 – Aug 2026	Solo — design to deploy	Backend service, VPS
FastAPI	Python	SQLAlchemy
	PostgreSQL	Alembic
	WeasyPrint	Docker

What's built

- 26 distinct Arabic document types across three business lines
- Full request logging — every document generated is recorded with who requested it and when
- A shared, filterable log viewer for monitoring usage
- Consumed by every other system, rather than each building its own PDF logic

26

document templates

3

business lines served

121

commits over 6 months

Engineering note

Correct Arabic PDF output is harder than it looks — right-to-left layout, Arabic text shaping, and number formatting all have to work correctly inside an automated document renderer, across 26 different templates. On top of that, every request is logged transparently without touching each document generator individually: middleware buffers and replays the request body, so it can log the full payload and still let the route handler consume it normally.

Shipping it, and keeping it running

Building the system is half the job — the other half is getting it live and keeping it there. Across these projects that meant a wide mix of deployment targets, hosting environments, and day-to-day operational work that never shows up in a diff.

01 Desktop releases

Auto-updating Electron builds via Electron Forge and GitHub Actions, shipping signed Windows and Mac installers straight to users.

02 Mobile builds

iOS and Android builds and OTA updates through Expo Application Services (EAS), including a full Apple App Store submission and review cycle.

03 Web hosting

Deployments across Vercel, Cloudflare Workers (via OpenNext), and traditional VPS/cPanel hosting, chosen per project's needs.

04 Domains & DNS

Domain registration and DNS through LibyaSpider and other registrars, plus SSL and hosting setup for every live project in this portfolio.

05 Backend services

Dockerized services on a VPS, with Postgres migrations managed through Alembic and structured request logging for observability.

06 Search & discovery

Google Search Console submissions to get public sites indexed and discoverable.

14

Personal Projects

Smaller, self-directed builds — different constraints, same habit of owning the whole stack.

MyFinance

A personal income/expense tracker, talking straight to Postgres over HTTP

A personal Flutter app for tracking daily income and expenses in Libyan Dinar, with category-based reports and monthly/yearly summaries — built for personal use, with no backend server of its own.

Timeline

Jun 2026

Role

Solo, personal project

Platform

Mobile — also builds for web & desktop

Flutter

Dart

Neon (serverless Postgres)

Material 3

What's built

- Daily dashboard with date navigation, spend/earn summaries, and a self-adjusting "tomorrow's goal" card
- Add-transaction flow with dynamic categories per income/expense type
- Category management with custom emoji and name
- Reports across monthly, yearly, and custom date ranges with per-category breakdowns

2

database tables

4

screens

~1,700

lines of Dart

Engineering note

No backend, no ORM — every screen talks directly to Neon's serverless Postgres over its SQL-over-HTTP API, through one small parameterized-query helper. The reports screen returns two parallel maps (amounts and emoji) from a single query pass using a Dart record type, keeping the data type-safe without a wrapper class — a small, deliberate design choice documented in the project's own notes.

That's the portfolio — more added as it ships.

I'm a full-stack engineer who likes owning things end to end: schema, backend, frontend, and the infrastructure underneath. If any of this lines up with what you're building, I'd love to talk.

noralden.elhouni@gmail.com

github.com/NoraldenElhouni · linkedin.com/in/noralden-alhouni

Tripoli, Libya