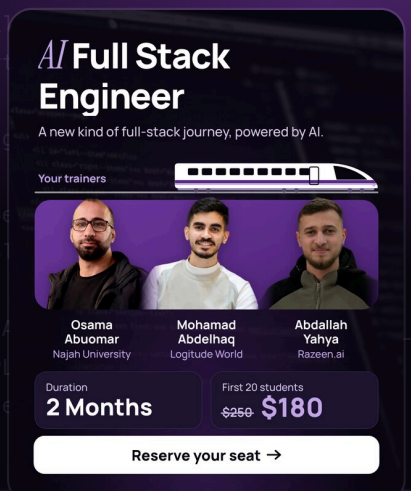
COURSE BROCHURE

AI Full Stack Engineer

A new kind of full-stack journey, powered by AI.

DURATION	FIRST 20 STUDENTS	FORMAT
2 months	~~$250~~ $180	Hands-on, project based



What this course is

A working web application has four parts: the frontend users see, the backend that answers requests, the database that holds the data, and the servers and tools that keep everything running. This course covers all four, so you understand how they fit together and can build a complete application on your own.

Every part is taught by building. Each phase ends with something you have made yourself, and by the end of the course you have working projects you can show to an employer or build on.

The four parts

1	**Frontend** Taught with React · page 2	HTML, CSS and JavaScript by hand first, then React with TypeScript. Build and deploy a tested web app.
2	**Backend** Taught with Node.js · page 2	Build a JSON API with Express, store data in PostgreSQL, add login and permissions, deploy with Docker.
3	**Databases** Taught with PostgreSQL · page 3	Write correct SQL, design clean tables, understand how Postgres works inside, make slow queries fast.
4	**DevOps & Cloud** Taught with Docker · page 3	Linux, Docker, CI with GitHub Actions, Kubernetes with GitOps, and cloud infrastructure as code.

Who it is for

- Students and graduates who want to build complete web applications, not just one piece.
- People who have tried tutorials but never finished a full project.
- Developers who know one part of the stack and want to understand the rest.

What you need

- No previous programming experience. Each part starts from the basics.
- Your own laptop, Windows, macOS or Linux.
- Some time between sessions to practise and finish the projects.

Your trainers



Osama Abuomar
Najah University



Mohamad Abdelhaq
Logitude World



Abdallah Yahya
Razeen.ai

PART 1 Frontend *Taught with React*

Everything a user sees and clicks. You write HTML, CSS and JavaScript by hand before React gets involved, then build typed, routed apps that stay fast and work from the keyboard.

PHASE	WHAT YOU LEARN	WHAT YOU HAVE AT THE END
1. The Browser	Build and style a real page before any framework hides how it works. How a page arrives · Semantic HTML and forms · The cascade and box model · Layout, Git and your first deploy	A responsive, keyboard-navigable site deployed to a real URL.
2. JavaScript	Make a page respond to a person with no framework doing it for you. The language first · Arrays and objects · DOM and events · Async and fetch · Modules, npm and Vite	A plain JavaScript app, built with Vite, that fetches live data and handles loading, empty and error states.
3. Thinking in React	Move from pages to components and put each piece of state in the right place. Components, props, JSX · Lists and keys · State and events · Lifting state and composition · Forms	An interactive multi-component React app with no behaviour you can't explain.
4. Types, Data, Routes	Talk to a real API without hand-rolled fetching, and let the types catch the rest. TypeScript where it pays · Routing with React Router · Server state with TanStack Query · Suspense and error boundaries · Client state, sparingly	A typed, routed React app running against a real API with honest loading and error UI.
5. Ship It	Get an app off your machine and prove it holds up for someone who isn't you. Tests you'll actually run · The accessibility audit · Performance you can measure · Build, deploy, hand over	A deployed app with a passing test suite, an accessibility audit and a measured performance budget.

PART 2 Backend *Taught with Node.js*

APIs, databases, authentication and the servers that answer them. You build services that hold real data, survive real traffic, and fail in ways you can explain.

PHASE	WHAT YOU LEARN	WHAT YOU HAVE AT THE END
1. The Runtime	Write, run and debug JavaScript on your own machine instead of in a browser. JavaScript, server side · Modules, npm and node_modules · Async without fear · When it breaks · The built-in toolbox	A command-line tool that reads files, calls a public API and writes a report.
2. HTTP and Your First API	Turn incoming requests into a JSON API a stranger could use without asking you. HTTP, unabstracted · Express 5 · Designing endpoints · Never trust the body (validation with Zod) · Types without a build step	A CRUD API with validated input, honest status codes and one error handler.
3. Data That Survives a Restart	Put real, related data behind the API and keep it correct while the schema changes. SQL before any ORM · Relationships and constraints · Postgres from Node · Migrations and transactions · Tests you can trust	A Postgres-backed API with migrations, seed data and tests that run on a throwaway database.
4. Users, Trust and Production	Let real people log in without handing them each other's data, then run it somewhere real. Authentication · Authorization · Attacks you'll actually meet · Your dependencies run your code · Ship it and watch it	A deployed, containerised API with sessions, ownership checks, structured logs and a health check.

PART 3 Databases *Taught with PostgreSQL*

SQL, schema design, transactions and the query planner. You write queries that are correct before they are fast, then make them fast by understanding what Postgres is actually doing.

PHASE	WHAT YOU LEARN	WHAT YOU HAVE AT THE END
1. Getting Correct Answers	Get a correct answer out of a database you did not design. The relational model · Your first queries · Joins · Duplicates, grouping and aggregates · NULL is not a value · Subqueries and EXISTS	A report answering 20 questions about a real dataset, with every row count justified.
2. Designing a Schema	Design tables where the wrong data cannot be stored in the first place. From requirements to tables · Normalization · Choosing types · Keys and constraints · Migrations	A schema built by versioned migrations that rejects bad rows on insert.
3. How Postgres Works	Explain what Postgres does between BEGIN and the bytes on disk. Pages, heaps and TOAST · Transactions and isolation · Locks and deadlocks · MVCC and vacuum	A concurrency problem reproduced in two sessions, plus the isolation level that stops it.
4. Making Queries Fast	Explain why a query is slow before adding an index. How an index works · Reading a query plan · Indexing strategy · Statistics and tuning	One slow query made ten times faster, with the plans from before and after.
5. Running It For Real	Keep the database available through failure, growth and your own mistakes. Backup and recovery · Replication and failover · Partitioning and upgrades · Postgres beyond relational (jsonb, full-text, pgvector)	A replicated cluster restored to a chosen point in time after you break it on purpose.

PART 4 DevOps & Cloud *Taught with Docker*

Linux, containers, Kubernetes and the pipelines between them. You install an app by hand first, containerise those exact steps, run it on a cluster, and ship changes without downtime.

PHASE	WHAT YOU LEARN	WHAT YOU HAVE AT THE END
1. Before Containers	Understand the machine an app runs on by running one the hard way first. The shell and the filesystem · Processes, ports and logs · Install it by hand	A web app and database installed by hand on a Linux server, with notes for every command.
2. Containers	Turn a manual install into an image that runs identically on any machine. Images are not containers · Your first Dockerfile · Layers, cache and size · Where the data goes · Compose for the whole app	A multi-container app a stranger can start with one command.
3. Ship It	Make every merge produce an image you can trust and run anywhere. Registries, tags and digests · Builds for any CPU · CI with GitHub Actions · What is actually in your image	A repository whose main branch publishes a scanned, signed image automatically.
4. Clusters	Run your images across several machines and deploy by merging instead of typing. Kubernetes without a cloud bill · Reading a broken pod · Config, secrets and storage · Getting traffic in · GitOps with Argo CD	A local Kubernetes cluster whose entire state comes from a Git repository.
5. Cloud and Operations	Create real cloud infrastructure from code, know when it breaks, and delete it before it costs you. Infrastructure as code · The bill is not a cap · Knowing when it breaks	A cloud environment defined in code with one dashboard, one alert, a short runbook and a teardown you have run.

What you will have built

Twelve projects, three per part. Each one is finished and working, not a half-done exercise, so you can show it in an interview or build on it afterwards.

PART	PROJECTS
Frontend	**Hand-built landing page.** No framework, keyboard-navigable, deployed to a real URL. **Vanilla DOM data app.** Fetch, render, filter and handle every state with zero libraries. **Typed React dashboard.** Routes, cached server state, tests and an accessibility pass.
Backend	**The bare-metal server.** HTTP by hand, before Express does it for you. **Library catalog API.** Two related tables, migrations and tests on a throwaway database. **Login you can defend.** Password hashing, server-side sessions and ownership checks that hold.
Databases	**Schema rescue.** Turn a 40-column spreadsheet into tables that reject bad rows. **The 10x query.** Explain first, index second, on a million rows. **Restore drill.** Break the database on purpose, then recover it to a chosen point in time.
DevOps & Cloud	**One command up.** App, database and cache from a single Compose file. **Merge, build, publish.** A pipeline that ships a scanned image on every merge. **Delete it all.** Wipe the cluster, rebuild it from Git, and prove it with a dashboard.

How the course works

- Sessions are practical. You build in every session and the trainers help you when you get stuck.
- The course has four parts: frontend, backend, databases, and DevOps and cloud.
- Each phase ends with something you have built, not a quiz.
- You keep all the code you write and can continue the projects after the course.

Course details

DURATION	2 months
PRICE	$180 for the first 20 students Regular price $250
PLACE	To be announced
SCHEDULE	To be announced
WHAT TO BRING	Your own laptop

Tools you will use

PART	TOOLS
Frontend	React, TypeScript, Vite, Tailwind CSS, React Router, TanStack Query, Vitest, Playwright
Backend	Node.js, TypeScript, Express, PostgreSQL, Drizzle ORM, Zod, node:test, Docker
Databases	PostgreSQL, psql, DBeaver, Docker, Flyway, pgbench, pg_stat_statements, pgvector
DevOps & Cloud	Docker, Docker Compose, Git and GitHub Actions, GitHub Container Registry, Kubernetes (k3d), Argo CD, OpenTofu, OpenTelemetry, Grafana

Reserve your *seat*

$180 for the first 20 students, then $250

WhatsApp: **+970 595 508 195**
Email: **info@xariotech.com**