predev. Academy

TRACK ROADMAP

Node.js Backend Development

Build, test, secure and deploy a production REST API with Node.js, TypeScript, NestJS and PostgreSQL.

Web Backend

DURATION	SESSIONS	SCHEDULE	LEVEL	LANGUAGE	LOCATION
10 weeks	20 live sessions, 60 hours	Twice a week, 3 hours per session, plus 1 hour of online office hours every week	Beginner, after Tech Compass	Egyptian Arabic, English technical terms	In person or online, live

Who it is for

- CS and IT students who want a backend job before or right after graduation
- Fresh graduates who know some programming but have never shipped a real API
- Career switchers who finished Tech Compass and chose backend development
- Frontend or mobile developers who want to write the API their apps call

Before day 1

- Tech Compass completed, or basic programming: variables, conditions, loops, functions
- Laptop with 8 GB RAM minimum (16 GB recommended), 20 GB free disk, Windows 10/11 with WSL 2, macOS, or Linux
- Accounts ready before day 1: GitHub, a free Postman or Bruno install, an AI coding tool account (Claude Code or Cursor) for week 8
- Installed before day 1: Node.js 24 LTS, VS Code, Git, Docker Desktop (or Docker Engine on Linux)
- Basic English reading: docs and error messages are in English

By the end you can

- Explain what happens in an HTTP request from the client to the database and back
- Write clean asynchronous TypeScript and reason about the Node.js event loop
- Design a normalized PostgreSQL schema and write SQL queries with joins and indexes
- Build a REST API in NestJS 11 with modules, DTO validation, guards and consistent error responses
- Model data with Prisma, run migrations and use transactions for operations that must succeed together
- Implement signup, login, refresh tokens, password hashing and role-based access control
- Add file uploads to S3-compatible storage, Redis caching and BullMQ background jobs
- Document an API with OpenAPI and test it with unit and end to end tests
- Run the whole stack with docker compose and deploy it to a VPS with Coolify, a domain and HTTPS
- Use AI coding tools to speed up work while catching their mistakes through review and tests

Tech stack

LANGUAGE & RUNTIME

JavaScript (ES2024)	The language of Node.js. Async code and modules are the base of everything after it.
TypeScript (strict mode)	Types catch mistakes before runtime and are the default in Egyptian and remote backend teams.
Node.js 24 LTS	Long term support release used in production; we use its fs, env and streams APIs directly.
pnpm / npm	Package managers for installing, locking and running project scripts.

FRAMEWORKS

Express 5	Small and explicit, so students see how routing, middleware and errors work under the hood.
NestJS 11	Main framework of the track: modules, dependency injection and a clear structure that teams use for large APIs.

DATA

PostgreSQL 17	Reliable relational database used by most serious products; we design schemas and write SQL by hand first.
Prisma ORM	Type-safe database access, schema-first migrations and transactions.
Redis 8	Caching, rate limit storage and the backbone for background queues.
BullMQ	Background jobs such as emails, notifications and scheduled tasks.

SECURITY & VALIDATION

JWT + refresh tokens	Stateless auth that works for web and mobile clients.
argon2 / bcrypt	Correct password hashing.
class-validator / Zod	Validate every request at the edge of the system.
helmet + rate limiting	Safe HTTP headers and protection against brute force and abuse.

QUALITY & DOCS

Jest / Vitest + Supertest	Unit and end to end tests that run against the real HTTP layer.
OpenAPI / Swagger	Live API docs that frontend and mobile developers read every day.
Pino logging	Structured logs you can search when something breaks in production.
ESLint + Prettier	One code style for the whole team, checked automatically.

DELIVERY

Git + GitHub	Branches, pull requests and code review, the same flow used at predev. Solutions.
Docker + docker compose	Same environment on every laptop and on the server.
S3-compatible storage (MinIO locally)	File uploads stored outside the server, the standard way to handle images and documents.
VPS + Coolify	Real deployment with a domain, HTTPS and environment variables, at a cost Egyptian freelancers can afford.

AI TOOLS

Claude Code or Cursor	Write code faster with AI while reviewing every change and keeping tests as the safety net.

The running project: Khedma API: a home services booking backend

Khedma is the backend for an app where customers book home services (plumbing, cleaning, AC repair) from verified providers. It mirrors the booking and ordering backends predev. Solutions builds for clients: users with roles, listings, availability, bookings with status changes, payments records, uploads, notifications and an admin panel API. Students build it step by step from week 4 to week 7 and deploy it live.

- Signup and login with JWT access tokens, refresh tokens and logout on all devices
- Three roles: customer, provider, admin, with guards on every protected route
- Service categories and provider profiles with photo upload to S3-compatible storage
- Provider availability slots and booking creation that blocks double booking inside a transaction
- Booking status flow: pending, confirmed, in progress, completed, cancelled, with rules per role
- Reviews and ratings after completed bookings
- Search, filtering, sorting and pagination on services and providers
- Redis caching for public lists and rate limits on auth routes
- BullMQ jobs for confirmation emails and booking reminders
- Real-time booking status updates over WebSockets
- Admin endpoints: manage users, approve providers, view basic stats
- OpenAPI docs at /docs, health check endpoint, structured logs
- Unit and end to end tests, Dockerfile, docker compose, deployed on a VPS with Coolify

Phase overview

WEEKS 1-3

Foundations

Understand how the web works and write solid TypeScript on Node.js with Git, SQL and problem solving habits.

WEEKS 4-7

Core framework

Build the Khedma API in NestJS 11 with Prisma, auth, uploads, caching, queues, tests and Docker.

WEEKS 8

Building with AI

Use AI coding tools to work faster on a real codebase while keeping quality under control through review and tests.

WEEKS 9

Capstone

Each student builds and deploys their own production API from a brief.

WEEKS 10

Career week

Turn the work of the bootcamp into a job or a first freelance client.

Week by week

Foundations · Weeks 1-3

Week 1 How the web works, JavaScript and Git

Session 1 HTTP, JSON and the request lifecycle

- Client, server, DNS, IP and ports: what happens when you open a URL
- HTTP methods, headers, body, and status codes (2xx, 3xx, 4xx, 5xx)
- JSON format and REST basics: resources, URLs and verbs
- Inspecting requests with browser DevTools, curl and Postman or Bruno
- Setting up the dev environment: Node.js 24 LTS, VS Code, terminal basics

In-class lab: Call a public API with curl and Postman, read every header and status code, then write a short note explaining one full request and response.

Session 2 JavaScript essentials and Git

- Variables, types, functions, arrow functions and scope
- Arrays and objects: map, filter, reduce, destructuring, spread
- Problem solving method: understand, plan, small steps, test with examples
- Git basics: init, add, commit, log, diff, .gitignore
- GitHub: create a repo, push, write a good README

In-class lab: Solve 6 small array and object problems in plain JavaScript, commit after each one with a clear message, and push the repo to GitHub.

Homework: Finish 10 JavaScript practice problems (strings, arrays, objects) in your repo with one commit per problem, and write a README that explains how to run them.

Week 2 Async JavaScript, Node.js runtime and TypeScript

Session 1 Async code and the Node.js runtime

- Callbacks, promises and async/await, with error handling using try/catch
- The event loop: call stack, task queue, microtasks, why blocking code hurts
- ES modules: import, export, package.json type and scripts
- Core modules: fs/promises, path, process.env and .env files
- Streams basics: reading a large file without loading it all in memory
- npm vs pnpm: install, lock files, semantic versioning

In-class lab: Build a CLI tool that reads a CSV file of orders with streams, calculates totals per customer, and writes a JSON report, using async/await and environment variables for the file paths.

Session 2 TypeScript for backend developers

- Why TypeScript: types, inference and the compiler
- Primitive types, arrays, union and literal types, type narrowing
- Interfaces and type aliases for API data shapes
- Generics for reusable functions and response wrappers
- tsconfig with strict mode, running TypeScript with tsx and building with tsc
- Git branches and pull requests: create a branch, open a PR, review and merge

In-class lab: Convert last session's CLI tool to strict TypeScript, add interfaces for every data shape, fix all compiler errors, and merge the change through a pull request reviewed by a classmate.

Homework: Write a typed in-memory task manager module (add, list, complete, delete, filter by status) with generics for the result type, and push it through a pull request with at least 5 small commits.

Week 3 SQL, PostgreSQL design and a first Express API

Session 1 Relational data and PostgreSQL 17

- Tables, rows, primary keys, foreign keys and constraints
- Relationships: one to many, many to many, junction tables
- Normalization basics and when to stop
- SQL: SELECT, WHERE, ORDER BY, JOIN, GROUP BY, aggregates
- Indexes: what they do and how to read EXPLAIN at a basic level
- Running PostgreSQL in Docker and connecting with psql or a GUI client

In-class lab: Design the database for a small online store on paper, create it in PostgreSQL running in Docker, insert sample data, and write 10 queries including joins and aggregates.

Session 2 Express 5: routing, middleware and errors

- Creating an Express 5 app with TypeScript
- Routes, route params, query strings and request body parsing
- Middleware: how the chain works, logging and auth placeholders
- Central error handling and consistent error JSON
- Async errors in Express 5 and returning correct status codes
- Connecting to PostgreSQL with the pg driver and parameterized queries

In-class lab: Build a notes REST API in Express 5 with full CRUD on PostgreSQL, a logging middleware, a central error handler, and correct status codes for not found and bad input.

Homework: Add search, pagination and input checks to the notes API, write a short API guide in the README with example requests, and solve 3 SQL challenges on joins and grouping.

Week 4 NestJS 11 structure and Prisma data layer

Session 1 NestJS fundamentals

- Why move from Express to NestJS: structure for team projects
- Modules, controllers and providers
- Dependency injection and how services are shared
- DTOs and validation pipes with class-validator, or Zod pipes
- Exception filters and a standard error response shape
- Project setup: Nest CLI, ESLint, Prettier, environment config module

In-class lab: Create the Khedma project with the Nest CLI, add the config module, and build a categories module with CRUD endpoints, DTO validation and in-memory storage.

Session 2 Prisma ORM with PostgreSQL

- Prisma schema: models, fields, relations and enums
- Migrations: create, apply, reset, and committing them to Git
- Prisma Client in a Nest service, typed queries
- Relations in practice: include, select and nested writes
- Seeding data for development
- Designing the Khedma schema: users, providers, services, categories, bookings, reviews

In-class lab: Write the full Khedma Prisma schema, run the first migration on PostgreSQL in Docker, write a seed script, and switch the categories and services modules to Prisma.

Homework: Finish CRUD for services and provider profiles on Prisma with validation on every input, add an ER diagram image to the README, and open one pull request per module.

Project milestone: Khedma API runs locally on NestJS 11 + Prisma + PostgreSQL 17 with seeded data and CRUD for categories, services and providers.

Week 5 Authentication, authorization and bookings

Session 1 Auth with JWT and refresh tokens

- Password hashing with argon2 or bcrypt, never storing plain passwords
- Signup and login flows, returning access and refresh tokens
- JWT structure, expiry and signing secrets from environment variables
- Refresh token rotation, storing hashed refresh tokens, logout on all devices
- Auth guards and a current-user decorator in NestJS

In-class lab: Build the auth module for Khedma: signup, login, refresh, logout and a protected /me endpoint, then test the full token flow in Postman or Bruno.

Session 2 Roles, permissions and the booking flow

- Role-based access control with a roles decorator and guard
- Ownership checks: a customer sees only their bookings
- Booking status rules per role as a small state machine
- Prisma transactions to prevent double booking of the same slot
- Pagination, filtering and sorting patterns for list endpoints

In-class lab: Add customer, provider and admin roles, build availability slots and booking creation inside a transaction, and enforce who can confirm, start, complete or cancel a booking.

Homework: Add reviews (only after a completed booking, one per booking), admin endpoints to approve providers, and search with filters and pagination on services.

Project milestone: Users can sign up, log in and book a provider; roles and ownership rules are enforced on every route.

Week 6 Files, caching, queues and real-time

Session 1 File uploads and Redis caching

- Multipart uploads in NestJS, size and type limits
- S3-compatible storage with MinIO locally: buckets, keys, public vs private files
- Presigned URLs for direct uploads and private downloads
- Redis basics: keys, expiry, data types
- Cache-aside pattern for public lists and cache invalidation on updates

In-class lab: Add provider photo upload to MinIO with validation, return presigned URLs, and cache the public services list in Redis with invalidation when a service changes.

Session 2 Background jobs and WebSockets

- Why some work must leave the request: emails, reminders, reports
- BullMQ queues, workers, retries and backoff
- Scheduled and delayed jobs for booking reminders
- WebSockets basics with a NestJS gateway
- Authenticating socket connections and sending events to one user

In-class lab: Send booking confirmation emails through a BullMQ worker (using a test mail inbox), schedule a reminder job before each booking, and push booking status changes to the customer over WebSockets.

Homework: Add retry limits and a failed jobs log, cache provider profiles with correct invalidation, and write a short README section explaining the queue and cache design.

Project milestone: Khedma supports photo uploads, cached lists, background emails and reminders, and live booking status updates.

Week 7 Testing, security, docs and Docker

Session 1 Testing and API documentation

- What to test: unit tests for services, end to end tests for routes
- Jest or Vitest setup in NestJS, mocks and test data
- Supertest end to end tests against a test database
- OpenAPI with the Nest Swagger module: decorators, auth in docs, examples
- Structured logging with Pino and request IDs

In-class lab: Write unit tests for the booking service rules and end to end tests for auth and booking routes, then publish OpenAPI docs at /docs with working bearer auth.

Session 2 Security and containers

- OWASP Top 10 applied to an API: injection, broken access control, auth failures
- helmet, CORS settings and rate limiting on login and signup
- Secrets and environment variables, never committing .env
- Dockerfile with multi-stage build for a NestJS app
- docker compose for API, PostgreSQL, Redis and MinIO
- Health check endpoint and graceful shutdown

In-class lab: Run a security checklist on Khedma and fix what fails, add helmet and rate limits, then write a multi-stage Dockerfile and a docker compose file that starts the whole stack with one command.

Homework: Reach test coverage on all booking rules and auth flows, fix every issue from the security checklist, and record a 3 minute screen video demoing the API through the Swagger docs.

Project milestone: Khedma API is tested, documented, secured and runs fully in docker compose, ready for deployment.

Building with AI · Weeks 8

Week 8 Building with AI the right way

Session 1 AI coding tools on a real codebase

- How Claude Code and Cursor work: context, prompts, agents and their limits
- Giving the tool project context: README, conventions file, small focused tasks
- Planning first: ask for a plan, review it, then let it write code
- Reading AI diffs line by line before accepting them
- Common AI mistakes in backend code: made-up APIs, missing auth checks, unsafe queries, outdated library versions

In-class lab: Use Claude Code or Cursor to add a provider earnings report endpoint to Khedma, review the diff with a checklist, and reject or fix every wrong change before merging.

Session 2 Tests as a safety net and when not to trust AI

- Write or review tests first, then let AI implement until they pass
- Using AI to write tests, and checking that the tests actually test something
- Security review of AI output: secrets, injection, access control
- When not to use AI: auth logic you do not understand, migrations on live data, copying code you cannot explain
- Licenses, privacy and not pasting client secrets or data into AI tools

In-class lab: Pair task: one student writes failing tests for a booking cancellation fee feature, the other uses AI to implement it, then both review the code together and explain every line.

Homework: Ship one new Khedma feature (for example coupon codes or provider working hours) with AI help, and write a short report: what the AI got right, what it got wrong, and how the tests caught it.

Project milestone: Khedma gains at least one AI-assisted feature, merged only after human review and passing tests.

Capstone · Weeks 9

Week 9 Capstone build and real deployment

Session 1 Capstone planning and build sprint

- Reading a product brief and turning it into endpoints and a data model
- Scoping: must-have vs nice-to-have, cutting features to finish on time
- Repo setup from the course template: lint, tests, Docker, env config
- Daily plan, GitHub issues and small pull requests
- Instructor check-ins on schema and API design

In-class lab: Each student picks a brief, gets the schema and endpoint list approved by the instructor, and has auth plus the first core module working by the end of the session.

Session 2 Deploy on a VPS with Coolify

- VPS basics: SSH, users, firewall, updates
- Coolify: projects, applications from a Git repo, databases and services
- Environment variables and secrets in production
- Domain, DNS records and HTTPS certificates
- Running migrations on deploy, health checks, reading logs, rolling back

In-class lab: Deploy the Khedma API live first as a class, then each student deploys their capstone with PostgreSQL and Redis on Coolify, a domain or subdomain, HTTPS and working OpenAPI docs.

Homework: Finish the capstone features, reach the test and docs requirements, fix production bugs found in logs, and prepare a 5 minute demo.

Project milestone: Capstone API is live on a public HTTPS URL with docs, tests and a clean Git history.

Week 10 Career week and demo day

Session 1 CV, LinkedIn, GitHub and interviews

- A one page backend CV with projects, links and measurable details
- LinkedIn profile for backend roles in Egypt and remote
- GitHub portfolio: pinned repos, READMEs with diagrams, live links and docs
- Backend interview topics: HTTP, SQL, auth, Node.js event loop, system basics
- Mock technical interviews with live coding and API design questions

In-class lab: Each student updates their CV, LinkedIn and GitHub in class with instructor review, then does a 20 minute mock technical interview with feedback.

Session 2 Freelancing first steps and demo day

- Freelancing on Upwork, Mostaql and Khamsat: profile, first proposals, pricing small API jobs
- Estimating a backend task and writing a clear proposal
- Working with clients: scope, milestones, handover and documentation
- Demo day: presenting your capstone to instructors and guests from the industry

In-class lab: Write one real freelance proposal for a backend job post, then present the capstone live on demo day: problem, data model, live API, tests and what you would do next.

Homework: Apply to at least 10 backend or junior roles or freelance jobs within two weeks after demo day, and share progress in the alumni group.

Capstone

Each student builds a production-ready REST API for a real-world product from a short brief, using the same stack and practices as the Khedma project, and deploys it on a VPS with Coolify. The work is done alone, with instructor check-ins and office hours.

Requirements

- NestJS 11 + TypeScript strict mode + Prisma + PostgreSQL 17
- At least 5 related data models with migrations and a seed script
- JWT auth with refresh tokens, hashed passwords and at least 2 roles with guards
- Validation on every input and a consistent error response format
- At least one of: file uploads to S3-compatible storage, Redis caching, or a BullMQ background job
- Unit tests for business rules and end to end tests for the main flows
- OpenAPI docs published at /docs
- Dockerfile and docker compose for local development
- Live deployment on a VPS via Coolify with HTTPS and environment variables
- README with setup steps, architecture notes, ER diagram and live links
- Clean Git history with pull requests

Example ideas

- Clinic appointments API: doctors, schedules, bookings and patient records
- Restaurant ordering API: menus, carts, orders and kitchen status updates over WebSockets
- Gym membership API: plans, subscriptions, check-ins and expiry reminders with BullMQ
- Real estate listings API: properties, photos on S3, search filters and inquiries
- Courses and tutoring API: instructors, courses, enrollments and progress tracking
- Delivery tracking API for a small shop: orders, drivers, statuses and live updates

Capstone rubric

Works end to end and is deployed live with HTTPS	25%
API design and data model quality	20%
Auth, authorization and security practices	15%
Tests: meaningful unit and end to end coverage	15%
Code quality, structure and Git history	10%
Documentation: OpenAPI and README	5%
Demo day presentation and answering questions	10%

How you are assessed

Weekly homework Graded on GitHub pull requests: works, clean code, clear commits, submitted on time.	20%
Khedma project milestones Checked at the end of weeks 4, 5, 6 and 7 against the milestone checklist.	25%
Quizzes and code reviews Short quizzes in weeks 3, 5 and 7, plus quality of reviews you give classmates.	10%
Building with AI assignment AI-assisted feature merged with passing tests and a written review report.	5%
Capstone project Scored with the capstone rubric.	30%
Career week deliverables Updated CV, LinkedIn, GitHub portfolio, mock interview and one freelance proposal.	10%

Skills checklist

Web and HTTP

- [ ] Explain the request and response cycle and pick correct status codes
- [ ] Design clean REST resources and URLs
- [ ] Debug requests with curl, Postman or Bruno and DevTools

JavaScript, TypeScript and Node.js

- [ ] Write async/await code with proper error handling
- [ ] Explain the event loop and avoid blocking it
- [ ] Use interfaces, unions and generics in strict TypeScript
- [ ] Use fs, env and streams in Node.js

Databases

- [ ] Design a normalized schema with correct relations and constraints
- [ ] Write SQL with joins, grouping and indexes
- [ ] Use Prisma migrations, relations and transactions

NestJS and API building

- [ ] Structure an app with modules, controllers and providers
- [ ] Validate input with DTOs and pipes
- [ ] Handle errors with exception filters
- [ ] Add pagination, filtering and sorting

Auth and security

- [ ] Implement JWT access and refresh tokens with hashed passwords
- [ ] Protect routes with role guards and ownership checks
- [ ] Apply helmet, CORS, rate limiting and the OWASP Top 10 basics
- [ ] Keep secrets out of Git

Performance and background work

- [ ] Cache with Redis and invalidate correctly
- [ ] Run background jobs with BullMQ, retries and scheduling
- [ ] Upload files to S3-compatible storage with presigned URLs
- [ ] Send real-time events over WebSockets

Quality and delivery

- [ ] Write unit and end to end tests with Jest or Vitest and Supertest
- [ ] Document APIs with OpenAPI
- [ ] Containerize with Docker and docker compose
- [ ] Deploy to a VPS with Coolify, a domain and HTTPS

Working with AI

- [ ] Give AI tools clear context and small tasks
- [ ] Review AI diffs and reject wrong code
- [ ] Use tests to verify AI output

After graduation

Months 1-3 after graduation

- Apply for junior backend and internship roles every week
- Polish the capstone and add one feature per week with tests
- Practice SQL and data structures problems 3 times a week
- Take a first small freelance API job, even at a low price, to build reviews

Months 4-6

- Learn deeper PostgreSQL: query plans, indexing strategy, views, full text search
- Add CI with GitHub Actions or another runner when available, and monitoring with uptime checks
- Learn one payment gateway used in Egypt such as Paymob, and webhooks
- Contribute small fixes to an open source Node.js project

Months 7-12: toward mid-level

- System design basics: scaling reads, queues, idempotency, rate limits
- Microservices vs modular monolith, message brokers and event-driven design
- GraphQL or gRPC as a second API style
- Cloud basics on AWS or GCP, and observability with metrics and tracing
- Mentor a newer student and lead code reviews

Where this track leads

Junior Backend Developer (Node.js) | Junior NestJS Developer | Node.js / TypeScript Developer | API Developer | Junior Full Stack Developer (with a frontend track later) | Backend Intern at software houses and startups | Freelance API Developer on Upwork, Mostaql and Khamsat | Remote Junior Backend Engineer for Gulf and European companies

Resources

- Node.js documentation: https://nodejs.org/docs/latest/api/
- MDN JavaScript Guide: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- MDN HTTP: https://developer.mozilla.org/en-US/docs/Web/HTTP
- TypeScript Handbook: https://www.typescriptlang.org/docs/handbook/intro.html
- Express documentation: https://expressjs.com/
- NestJS documentation: https://docs.nestjs.com/

- Prisma documentation: https://www.prisma.io/docs
- PostgreSQL 17 documentation: https://www.postgresql.org/docs/17/
- Redis documentation: https://redis.io/docs/latest/
- BullMQ documentation: https://docs.bullmq.io/
- Jest documentation: https://jestjs.io/docs/getting-started
- Vitest documentation: https://vitest.dev/guide/
- Supertest: https://github.com/ladjs/supertest
- OpenAPI Specification: https://swagger.io/specification/
- OWASP Top 10: https://owasp.org/www-project-top-ten/
- Docker documentation: https://docs.docker.com/
- Coolify documentation: https://coolify.io/docs/
- Pro Git book: https://git-scm.com/book/en/v2
- GitHub Docs: https://docs.github.com/
- pnpm documentation: https://pnpm.io/motivation