

Lesson 04 — Modern Web Stack

Frontend / Backend / API — what each is, why apps split that way.

IT Landscape

Module 2

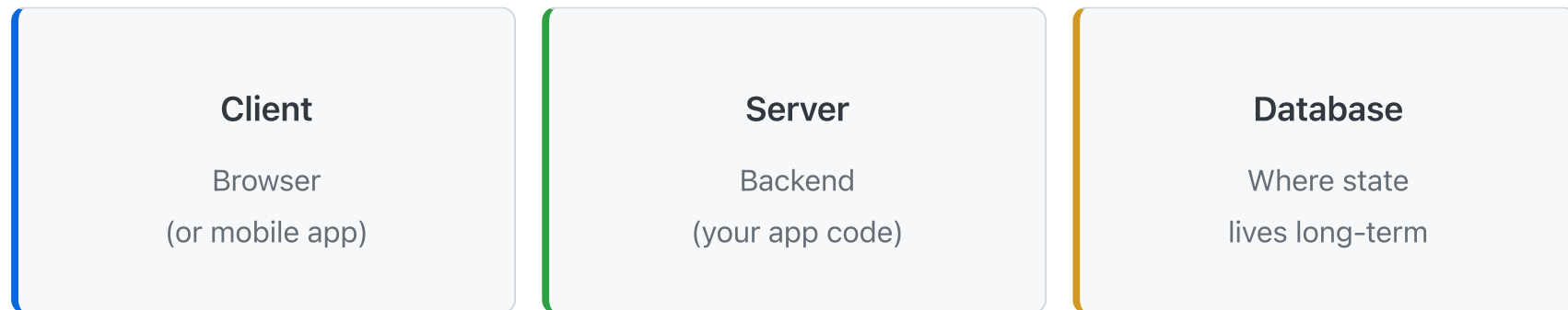
What is a "web app"?

*A website that **does** something — not just shows you something.*

Examples: **Gmail, Google Docs, Notion, Spotify Web Player, Slack.**

Contrast: a restaurant's menu page. That's a *brochure* — you read it and leave. A web app remembers you, accepts your input, talks back.

The 3-tier model in one diagram



Client — *HTTP request* → Server — *SQL query* → Database

Almost every app on the internet is some flavor of this.

Tier 1 — Frontend (the browser)

- **HTML** — structures the page.
- **CSS** — styles it (colors, layout, fonts).
- **JavaScript** — makes it interactive.

The frontend = **everything the user actually sees and clicks**. Lives in the browser. Sends HTTP requests when it needs data.

Note: "frontend" is sometimes called "client-side" — same thing.

Tier 2 — Backend (the server)

- An **HTTP server** (Nginx, Node, Django, Rails) plus your **app code**.
- Receives a request → talks to a database → returns JSON or HTML.
- Runs on a Linux box in a data center somewhere — or on serverless infra (Lesson 06).

The backend is the only thing trusted to read or write the database. The client never talks to the DB directly.

The handshake between them — APIs

An API is a contract: "ask me this way, I'll answer this way."

Most modern APIs speak **REST** (over HTTP, JSON-shaped) or **GraphQL** (single endpoint, you specify the shape).

Illustrative one-liner — you do *not* need to run this:

```
fetch('/api/users/42') // server returns { "id": 42, "name": "Ada" }
```

Real example — a modern site under the hood

What happens when you open `notion.so`:

1. Browser downloads ~100 KB of JavaScript (a React app).
2. That JS calls `/api/v3/getSpaces`.
3. Backend hits a **Postgres** database.
4. Backend returns JSON.
5. React renders the sidebar — the page lights up.

Five steps. Three tiers. **Same shape as Gmail, Slack, your bank.**

Frontend frameworks you'll hear named (as of 2026)

Name	From	What it's for
React	Meta	Biggest mindshare
Vue	Independent	Lightweight alternative
Svelte	Independent	Compiles away — small bundles
Angular	Google	Enterprise, opinionated
Next.js	Vercel	React + a backend bundled
Astro / SolidJS	Independent	Newer, niche

Most teams pick one and stick with it for years.

Backend frameworks you'll hear named (as of 2026)

Language	Common frameworks
Python	Django, FastAPI
Node.js	Express, NestJS
Ruby	Rails
Java	Spring Boot
C#	ASP.NET
PHP	Laravel
Go	<code>net/http</code> + chi / gin
Elixir	Phoenix

Many roads up the same mountain.

Static sites vs dynamic apps

Static

Built once, served to all.

- Blogs
- Documentation
- Marketing pages

Tools: Hugo, Jekyll, Astro

Dynamic

Rendered per user / per request.

- Dashboards
- Social feeds
- E-commerce

Tools: anything with a backend

Mobile fits in too

- **Native** — Swift for iOS, Kotlin for Android. Fastest, separate codebases.
- **Hybrid / cross-platform** — React Native (Meta), Flutter (Google). One codebase, both stores.
- **Mobile web** — just a website on your phone, possibly installable as a **PWA** (Progressive Web App).

All three usually talk to the **same backend + same API** as the web app.

What you'll see on a job description

- "Looking for a **React** developer with **REST API** experience."
- "Backend engineer — **Python/Django, Postgres, AWS.**"
- "Full-stack — **TypeScript** everywhere."

You now have the decoder ring. Frontend, backend, API, database — you can place every one of these words.

Recap

- **3 tiers:** client (browser) ↔ server (backend) ↔ database.
- **APIs** glue them together — REST and GraphQL are the two big shapes.
- **Frontend and backend frameworks** are tools, not religions. Many roads up the same mountain.

Coming up: Lesson 05 — what writing code actually looks like in 2026.

