React & Next.js Project Structure (CRA vs Next)

How files map to UI, routes, and API endpoints

What is this section about?

This section explains:

- How a **React** project created with **Create React App (CRA)** is structured
- What `src/` and `public/` actually do
- Why you often need a **separate backend** for React
- How **Next.js** solves this by supporting **frontend + backend** in one project
- How **routing** and **API endpoints** work in Next.js via the `pages/` folder

React File Structure (Create React App / CRA)

What is CRA?

Create React App (CRA) is a command-line tool that scaffolds a new React project and installs the required dependencies so you don't have to write boilerplate from scratch.

Typical CRA Project Layout

- `package.json` — project metadata + dependencies (Node.js ecosystem)
- `src/` — all React code (components, CSS, logic)
- `public/` — static assets served directly (images, icons, fonts), plus `index.html`

What is the role of `src/`?

`src/` contains the **React application code**. The two most important files are:

- `App.js` — the main (autogenerated) top-level component you edit frequently
- `index.js` — the entry point that mounts your React app into the page (rarely edited)

What is the role of `public/`?

`public/` contains files that should be **directly accessible** on the website.

- CRA includes `public/index.html` (a minimal HTML shell)
- React injects your compiled JS into this page and renders into a root element
- Static assets like images/fonts can live here and be referenced by URL paths

Example: referencing an image in `public/`

```
<img src="/avatar.png" />
```

Why does `/avatar.png` work from inside `src/`?

Even though your component lives in `src/`, the browser loads assets relative to the **final HTML page**. Since assets in `public/` are served at the site root, `/avatar.png` maps to `public/avatar.png`.

Common confusion

`public/` is not "React code". React components belong in `src/`. `public/` is for static files served as-is.

Backend Reality: React is Frontend-only

React is not a backend framework

React only handles the **UI layer**. If you need an API server (database calls, auth, blockchain indexing, etc.), you usually build it separately using something like **Node.js + Express**.

Why two projects can be annoying

- Duplicate setup and deployment pipelines
- Harder to share types/utilities across frontend and backend
- More moving parts = more maintenance

Next.js: React + extra "app framework" features

What is Next.js? (Meta-framework)

Next.js is a framework built **on top of React**. If you know React, you already know most of Next. Next adds features React intentionally does not include by default.

Key Next.js features (high-yield)

- **Frontend + Backend in one project** (API routes)
- **File-based routing** (routes from filenames)
- Optional rendering modes: **SSR** (Server-Side Rendering) and **SSG** (Static Site Generation)

Practical takeaway

If you want a full-stack web app without maintaining 2 separate repos, **Next.js is a common default choice**.

Routing in Next.js (Pages Router mental model)

Setup tool

Next.js projects are often scaffolded with:

- `create-next-app`

Typical Next.js project folders

- `public/` — static assets (same concept as React), but usually no `index.html`
- `styles/` — dedicated folder for CSS
- `pages/` — **special**: defines routes and API endpoints

File-based routing (the magic)

Each file under `pages/` becomes a route:

- `pages/index.js` → `/` (homepage)
- `pages/about.js` → `/about`
- `pages/tracks/freshman.js` → `/tracks/freshman`

High-yield naming rule

`index.js` inside any folder becomes that folder's root route. Example: `pages/tracks/index.js` → `/tracks`

What is `pages/_app.js`?

`pages/_app.js` is an autogenerated wrapper that lets Next initialize pages consistently (global CSS, layouts, providers). You usually touch it only for app-wide setup.

Writing APIs in Next.js (pages/api)

`pages/api` = backend endpoints

Files under `pages/api/` do **not render HTML**. They behave like API routes and return data (usually JSON).

Example: `pages/api/hello.js`

```
export default function handler(req, res) {
  res.status(200).json({ name: "John Doe" });
}
```

How it behaves

Visiting `/api/hello` returns JSON instead of a webpage:

- Response: { `name: "John Doe"` }

API routing rule

Just like UI pages, API endpoints are file-based:

- `pages/api/hello.js` → `/api/hello`
- `pages/api/user.js` → `/api/user`
- `pages/api/auth/login.js` → `/api/auth/login`

Must-know: Client vs Server code

Code in `pages/api/*` runs on the **server** (Node environment). Code in `pages/*` (non-api) runs as **frontend UI**. This separation is powerful, but keep secrets (keys) only on the server side.

Final Takeaways

- CRA React apps: `src/` for UI code, `public/` for static assets and `index.html`
- React alone is frontend-only; backends require a separate server (e.g., Express)
- Next.js extends React with file-based routing and built-in API routes
- `pages/*` defines UI routes; `pages/api/*` defines backend endpoints

Rule of thumb: Want a simple SPA? CRA is fine. Want full-stack + easy routing? Next.js is often the better default.