Web Development Practice Projects:

From Tutorials to Real Skills

- Building projects beats watching tutorials for demonstrable skill
- A practice project needs clear scope, constraints, and one deliverable
- The project ladder: static page → interactive UI → data-driven app → full-stack
- Match projects to your level; finish in days, not weeks
- Each module is a build-along sequence you can follow

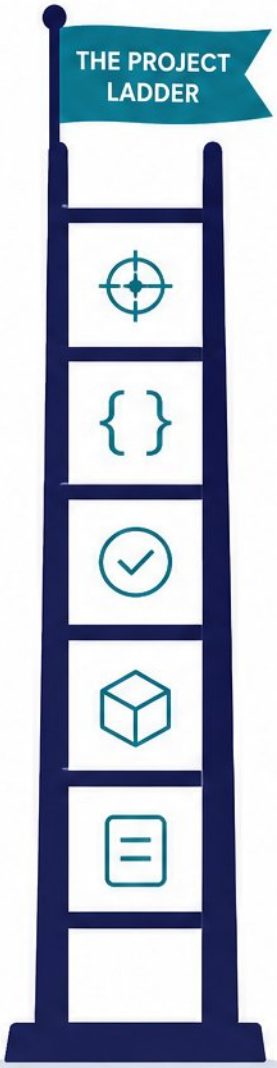
Choosing the Right Project for Your Level

- **Rule:** pick a project one small step above what you can already do
- **Categories:** static portfolio, to-do and quiz apps, weather/movie apps, full-stack task manager
- **What each proves:** styled layout, interactivity, API integration, CRUD, end-to-end build
- Three to five polished projects beat fifteen half-finished repos
- Cloning a familiar site is fine — going beyond the tutorial makes it portfolio-worthy

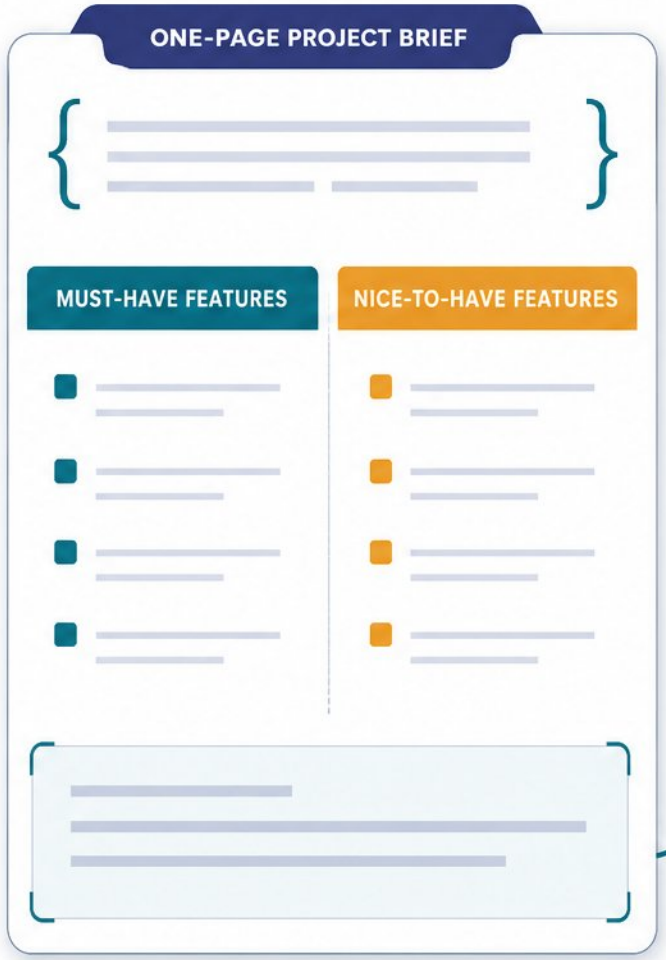
Setting Up Your Beginner Toolkit

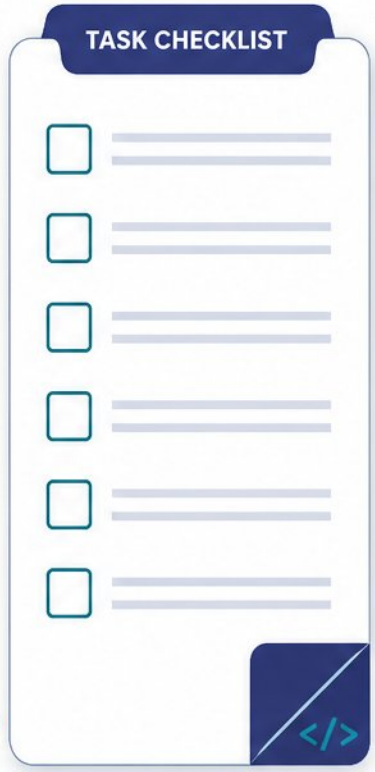
- VS Code plus a focused set of extensions, not hundreds
- ESLint and Prettier for lint-on-save and format-on-save
- Live Server, GitLens, Path Intellisense, EditorConfig, Auto Rename Tag
- Browser DevTools for inspecting elements, console, and responsive testing
- Git and GitHub: init, commit, branch, push, host your repo
- Node.js LTS and npm — install a build tool only when needed

Project Planning and Scope Control



- Write a one-page brief: must-have vs. nice-to-have features
- Break work into small, shippable tasks — not one giant milestone
- Define "done": works locally, works deployed, okay on mobile
- Prioritize features by learning value before visual polish
- Keep a lightweight README and a running checklist as you build





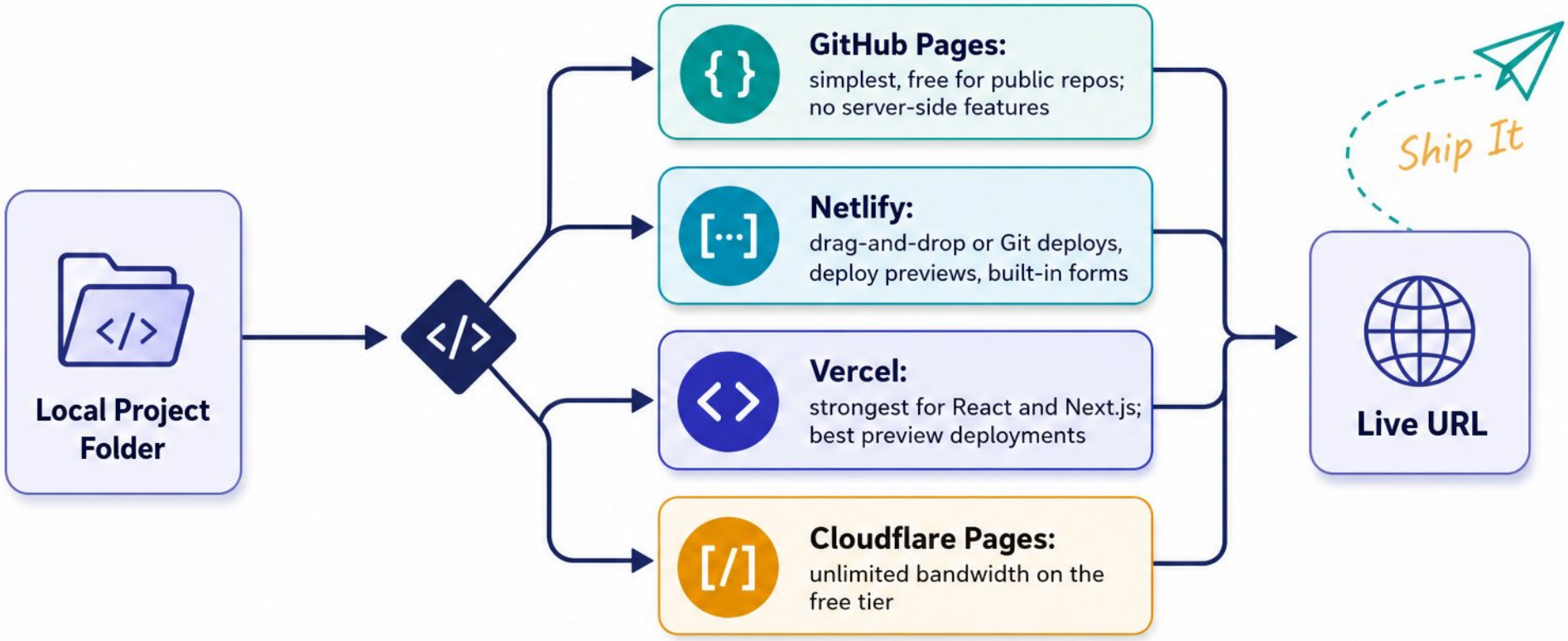
Project 1: A Static Portfolio Page

- **Semantic HTML:** header, about, projects, contact sections
- **Flexbox** and **Grid** layout, then mobile-first breakpoints
- **Accessibility:** alt text, form labels, visible focus states
- **Small JS touch:** menu toggle or smooth scrolling
- Deploy free on GitHub Pages, Netlify, or Vercel for a live URL

```
<header> ... </header>
<section id="about"> ... </section>
<section id="projects"> ... </section>
<section id="contact"> ... </section>
<script> ... </script>
```

One page. Four sections. Clean, semantic, and accessible.

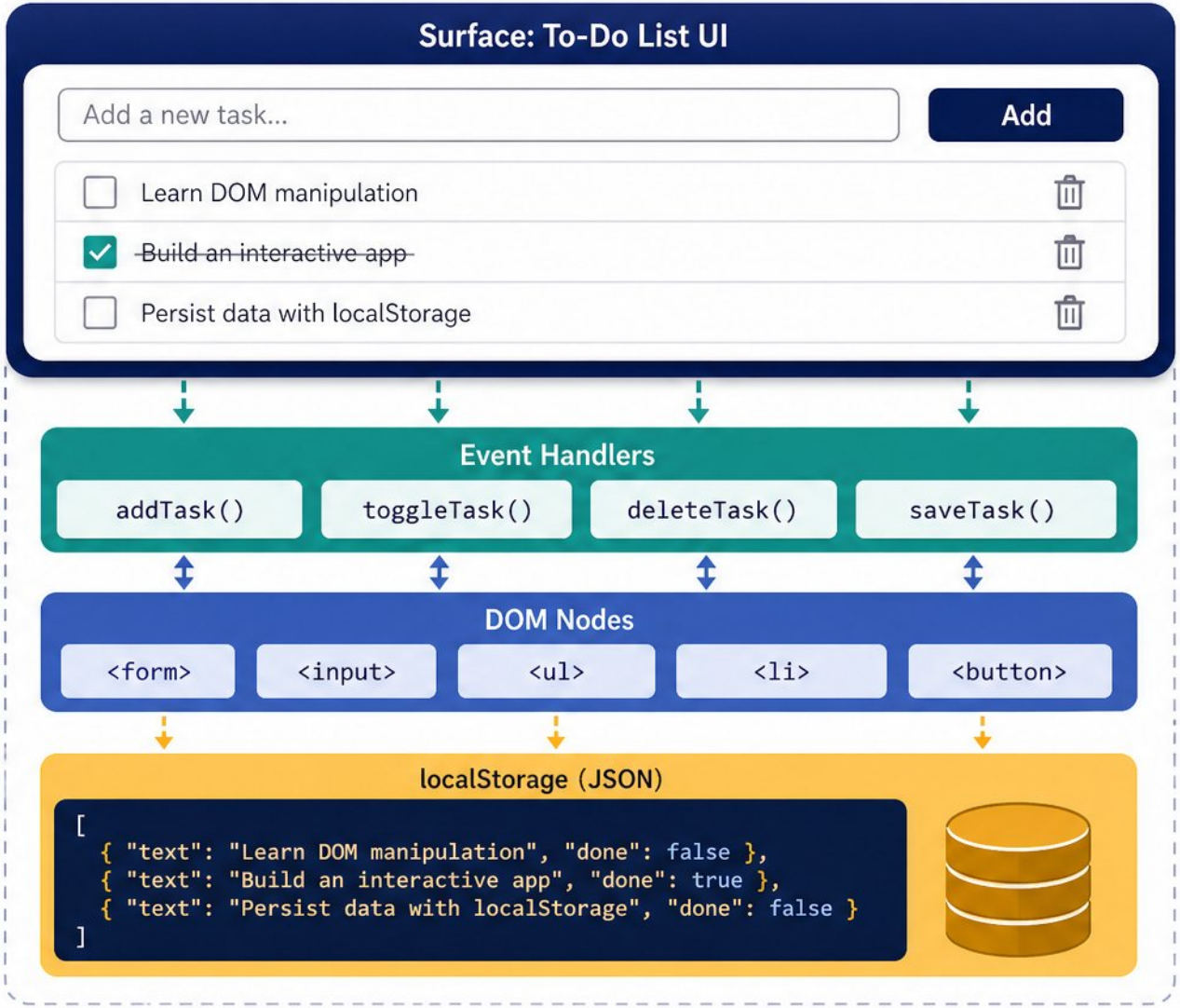
Deploying Your First Static Site



All four: custom domains and free SSL on the free tier

Project 2: An Interactive App with Vanilla JavaScript

- Pick a small, finishable app: to-do list, quiz, unit converter, expense tracker
- Core skills: DOM manipulation, event handling, form validation
- Persist state with localStorage; serialize with JSON and read back safely
- Handle empty and corrupt data without breaking the page
- Organize code into small functions; keep data separate from rendering
- Debug with console logs and DevTools breakpoints, not guessing



Working with Data and Public APIs

- An API is a data service you query by URL; read its docs first
- Use fetch with async/await and try/catch to handle responses
- Show loading, success, empty, and error states so apps feel reliable
- Keep API keys out of source control; respect rate limits
- Start keyless: Open-Meteo, JSONPlaceholder, REST Countries, Dog CEO, Trivia API
- JSONPlaceholder supports GET, POST, PUT, PATCH, DELETE for practice

Fetch-and-Async Workflow

Request → Process → Handle

```
async function getData(url) {
  try {
    const res = await fetch(url);
    if (!res.ok) throw new Error(
     'Request failed');
    const data = await res.json();
    return data;
  } catch (err) {
    throw err;
  }
}
```

- Loading
- Success
- Empty
- Error

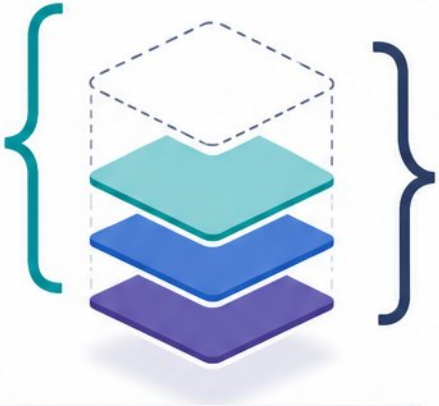
Keyless Starter APIs (Specimens)

- Open-Meteo
- JSONPlaceholder
- REST Countries
- Dog CEO
- Trivia API

JSONPlaceholder supports GET, POST, PUT, PATCH, DELETE for practice

Project 3: A Data-Driven App

- Build a weather dashboard, movie search, or public data explorer
- Know HTTP basics: methods, status codes, headers, JSON response shape
- CORS errors mean the browser blocked the request, not your code
- Handle loading, success, empty, and error states gracefully
- Shape fetched JSON into your own model before rendering



A Data-Driven App

Weather
Get current conditions for any location.

Movie Search
Find movies and explore details.

Public Data
Explore open data from public sources.

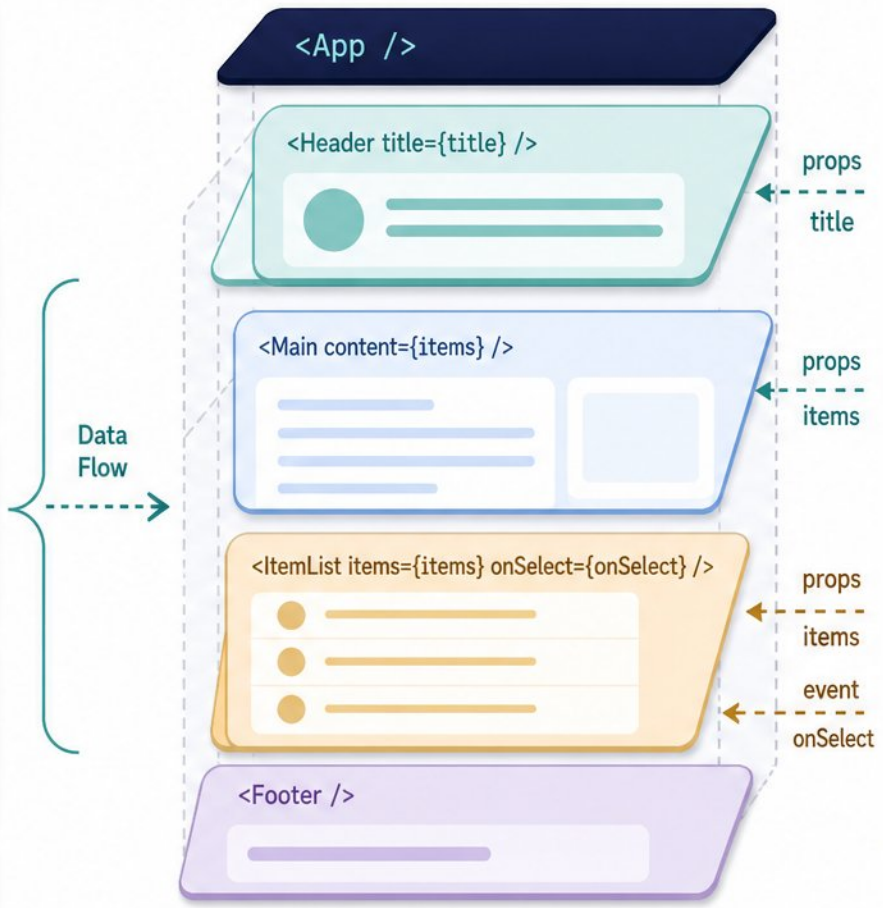
LOADING STATE	SUCCESS STATE	EMPTY STATE	ERROR STATE

```
fetch(endpoint, options)
  .then(response => response.json())
  .then(data => mapToModel(data))
  .catch(handleError)
```

Introducing a Framework: A Small React or Vue Build

- Frameworks add component reuse, managed state, and predictable re-rendering
- Scaffold with Vite: npm create vitelatest, then read the folder structure
- Build components, pass data with props, and handle user input
- React: bigger job market. Vue: gentler learning curve, HTML-like templates
- Refactor your vanilla JavaScript app into a component-based version

Component Breakdown (Exploded View)



Vite Project Structure

```
my-app/
├── node_modules/
├── public/
├── src/
│   ├── components/
│   │   ├── Header.vue
│   │   ├── Main.vue
│   │   ├── ItemList.vue
│   │   └── Footer.vue
│   ├── App.vue
│   └── main.js
├── .gitignore
├── index.html
├── package.json
└── vite.config.js
```

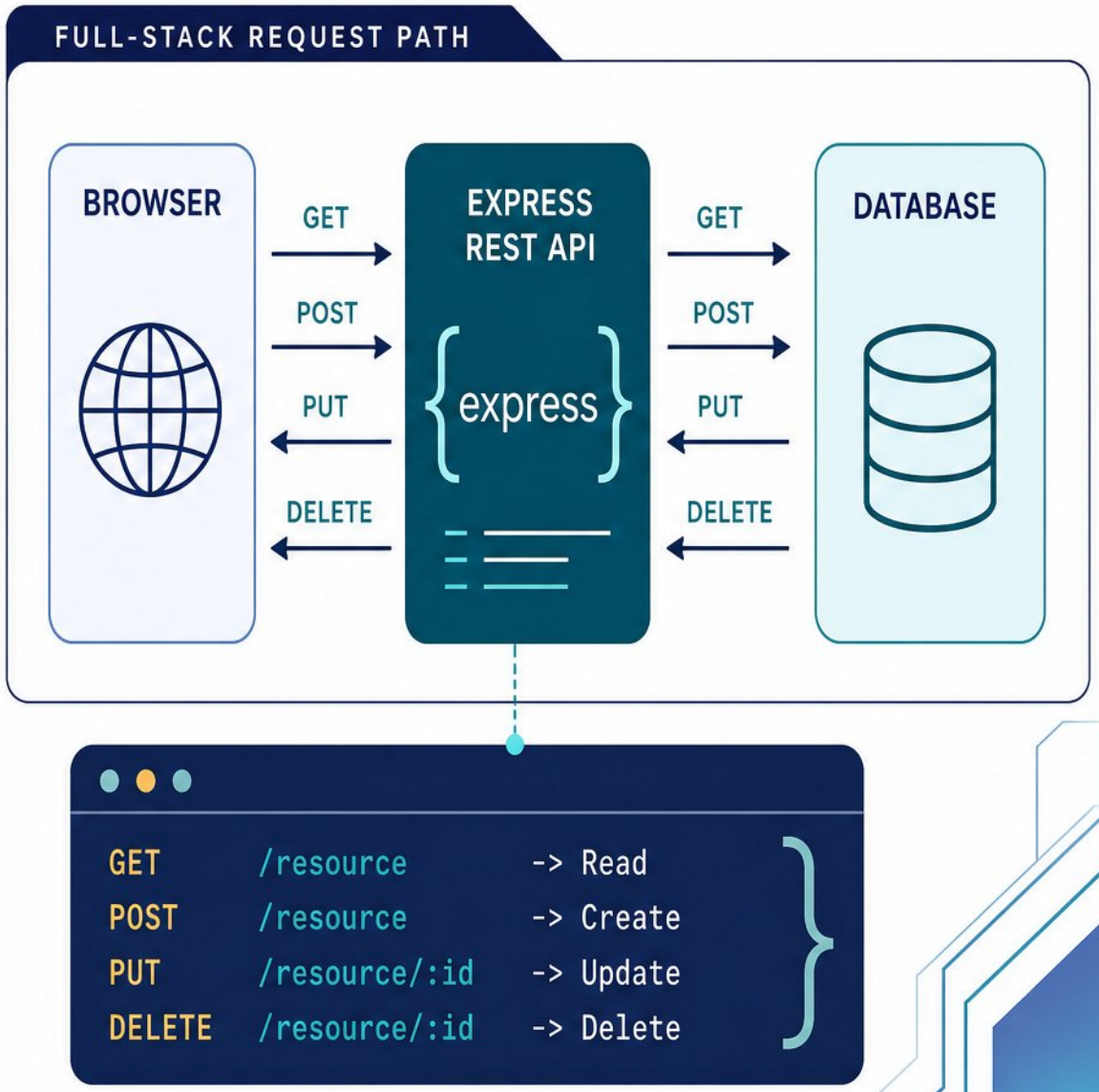
Components are reusable pieces.
Props pass data down. **Events** send data up.

THE PROJECT LADDER

4

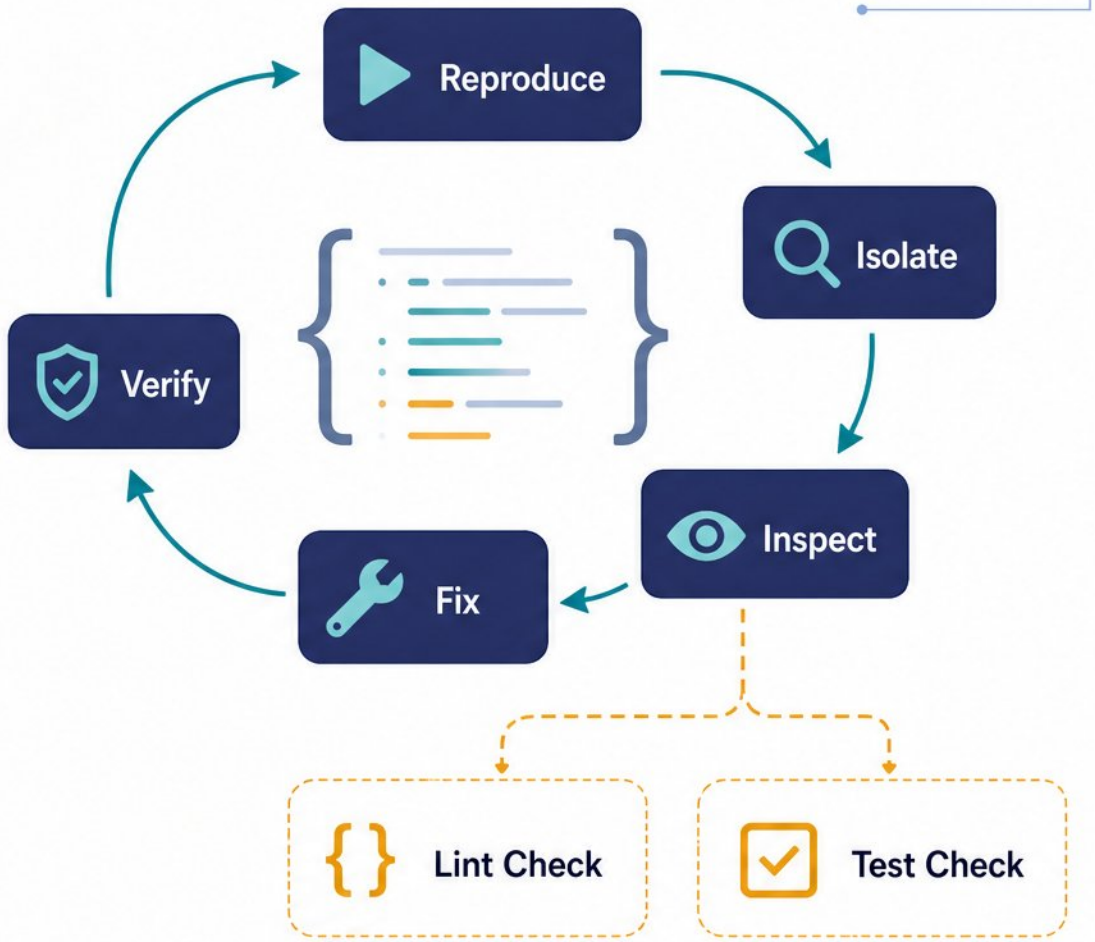
Project 4: When You Need a Backend and Database

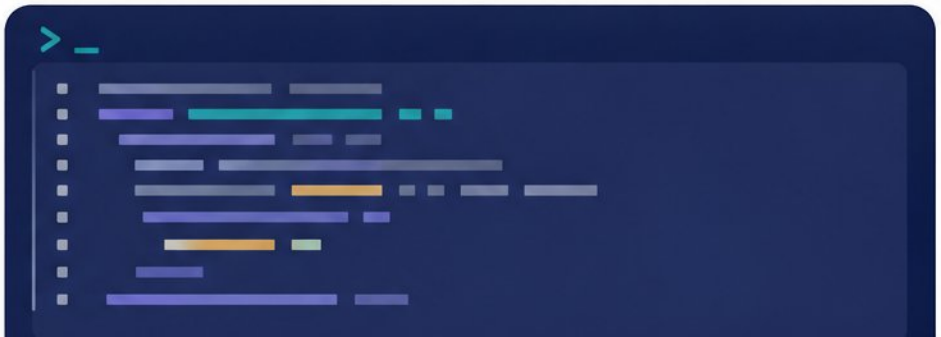
- Add a server only when localStorage or a static site cannot store shared data
- Model real features first: users, posts, or tasks with clear fields
- Build a minimal Express REST API mapping CRUD to GET, POST, PUT, DELETE
- Connect SQLite or hosted Postgres with a pool and parameterized queries
- Keep credentials in environment variables, never in source control



Testing, Debugging, and Code Quality Habits

- Debug systematically: reproduce, isolate, inspect, fix, verify
- Write a few meaningful tests; Vitest 4 is the modern default
- Set up ESLint 9 flat config plus Prettier to catch mistakes early
- Read the first line of an error — it usually names the problem
- Refactor for readability after the feature works, not mid-build

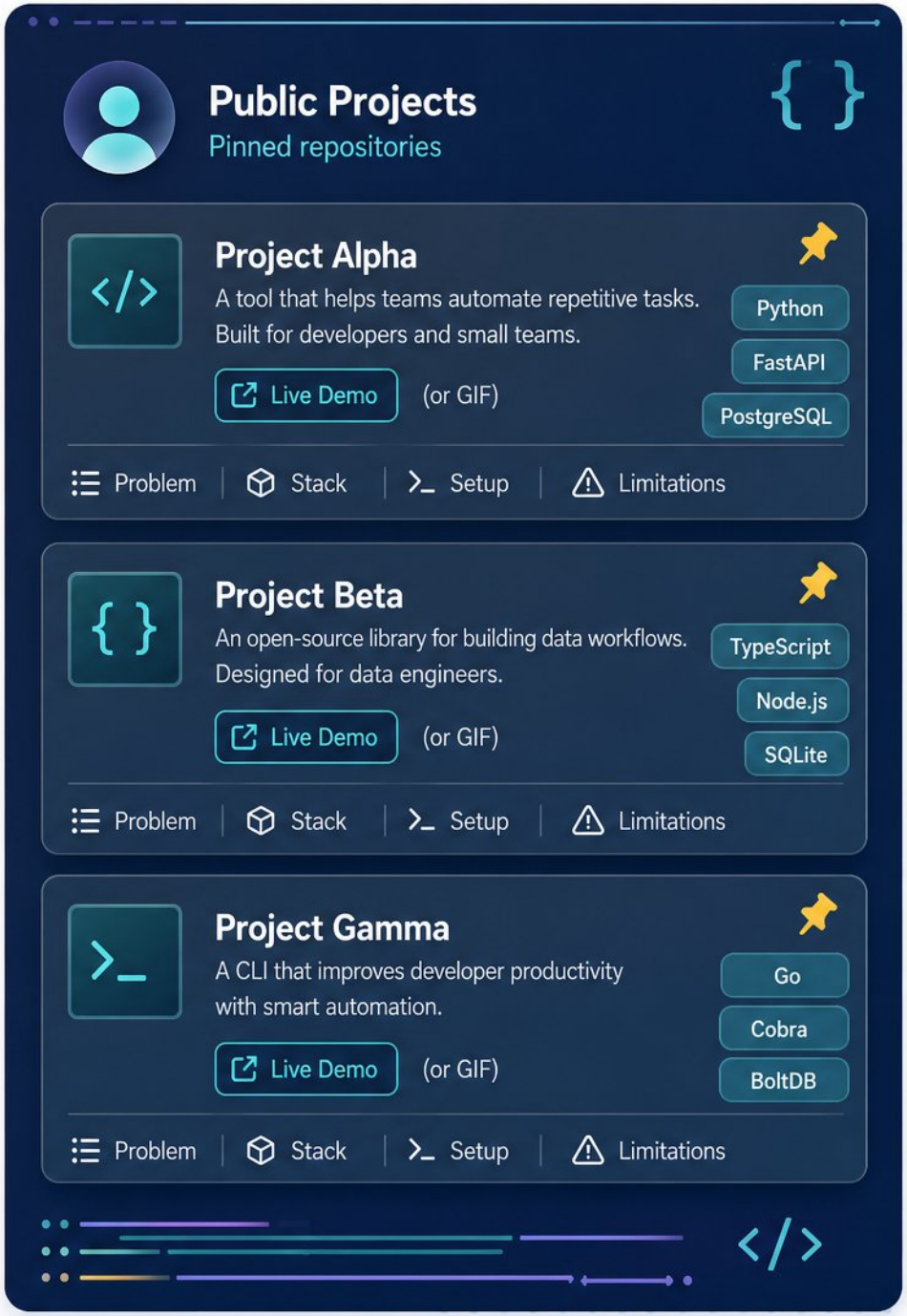




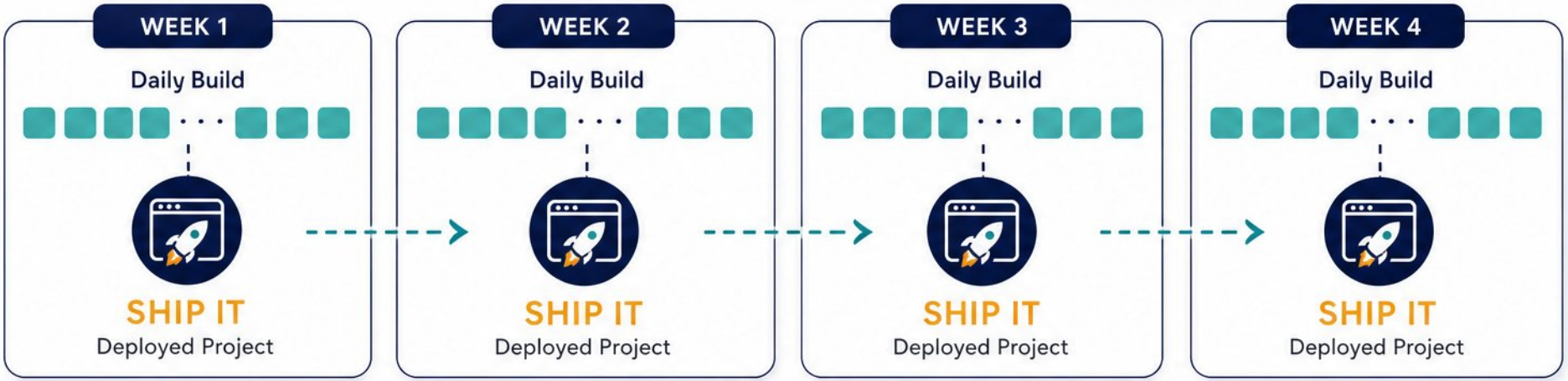
Documenting and Showcasing Your Projects

- One sentence: what it does and who it is for
- Demo link or GIF near the top of the README
- Problem, stack, setup commands, and known limitations
- Reviewers open the live demo first, then the README
- Show your specific contribution, tradeoffs, and challenges
- Avoid unfinished, tutorial-cloned, or buzzword-heavy repos
- Pin your three to five strongest projects

Ship It
Build. Document. Ship.



A 30-Day Plan for Keeping Momentum



- Daily goal: even 30–60 minutes keeps the build moving
- Four weeks: finish and deploy one project per level
- Ship before you polish — a deployed imperfect project teaches more
- Ask classmates, communities, or a mentor for feedback after each deploy
- Keep a running list of next project ideas so you never stall

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/4857d3c5/public