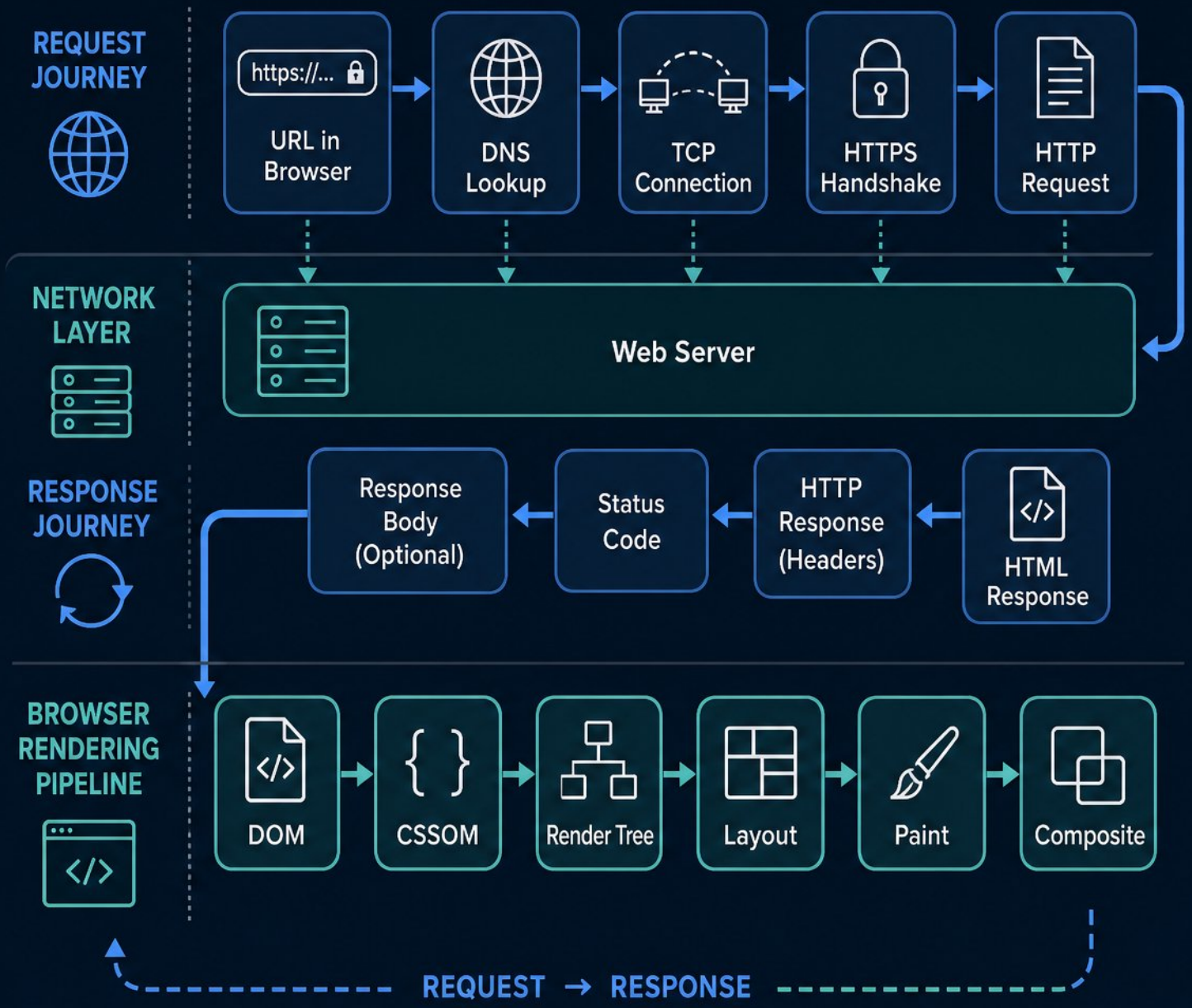
Introduction to Web Application Development

- Web apps are interactive and stateful — login, create, edit, and save data
- Static sites return the same hard-coded page to every visitor
- Client-server model: browser requests, server processes logic, database stores data
- Core roles: frontend, backend, full-stack, DevOps
- You'll build a task-list app with CRUD, auth basics, and a live deploy

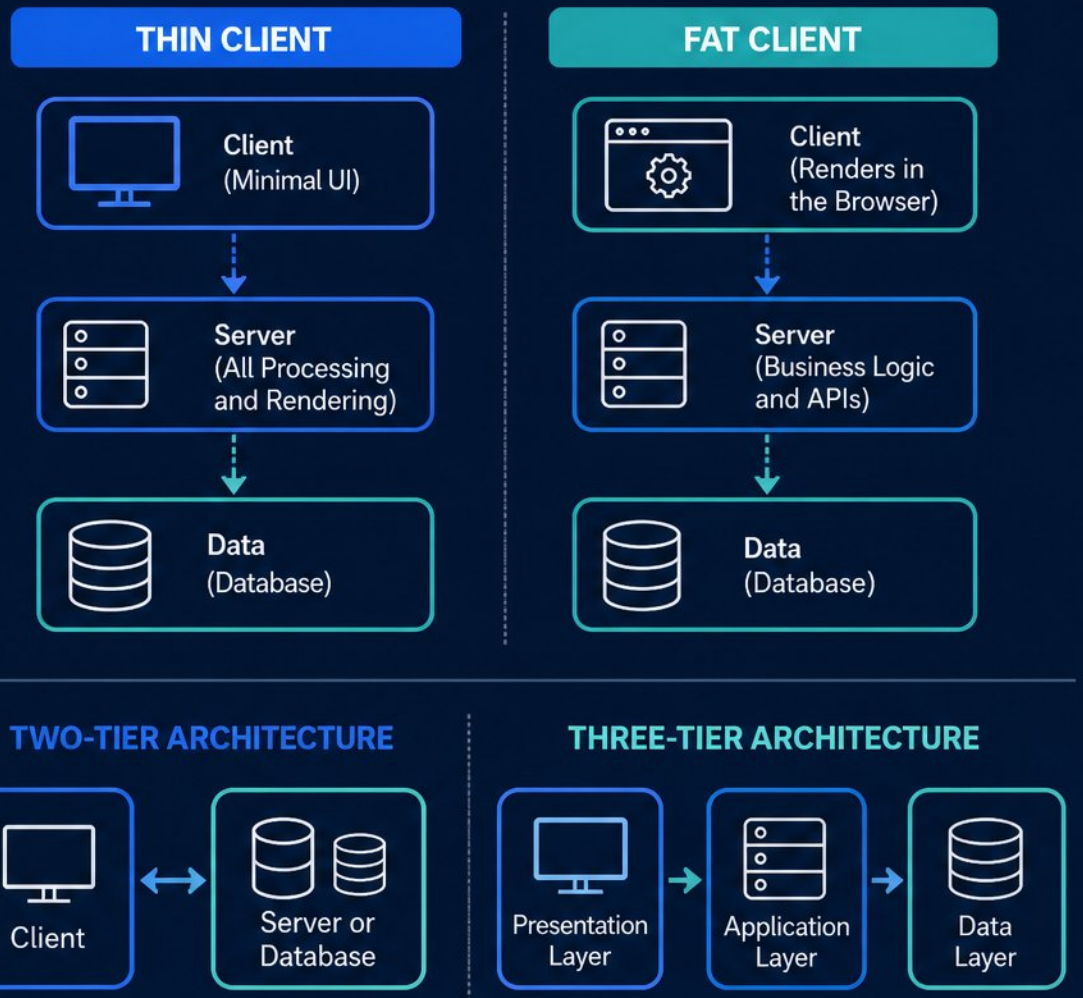
How the Web Works: Browsers, HTTP, and the Rendering Pipeline



- URL to page: DNS lookup, TCP and HTTPS handshake, HTTP request, HTML response
- HTTP message anatomy: start line, headers, blank line, optional body
- Status code families: 2xx success, 3xx redirect, 4xx client error, 5xx server error
- Rendering pipeline: DOM, CSSOM, render tree, layout, paint, composite
- CSS is render-blocking; synchronous scripts are parser-blocking — use defer and async
- DevTools: inspect the Network tab, then trace layout and paint in Performance

Client-Server Architecture and Application Tiers

- Clients request, servers respond: an asymmetry shaping every design choice
- Thin clients rely on the server; fat clients render in the browser
- 2026 hybrid: server-rendered first load, client-side interactivity after
- Two-tier: client talks straight to the server or database
- Three-tier: presentation, application, and data layers separated
- Statelessness lets any server handle any request, enabling horizontal scaling
- Golden rule: client presents, server enforces rules and access control

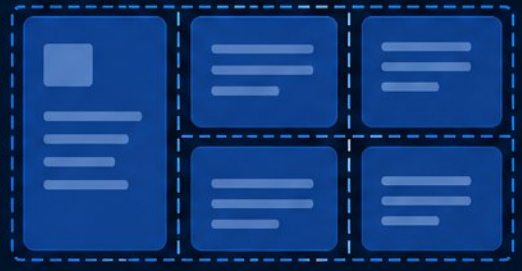


Frontend Fundamentals: Semantic HTML and Modern CSS Layout

- Semantic HTML builds structure that screen readers, search engines, and AI can parse
- WCAG 2.2 AA essentials: descriptive alt text, keyboard operability, visible focus, 4.5:1 contrast
- Use CSS Grid for two-dimensional page scaffolding, Flexbox for one-dimensional alignment
- `repeat(auto-fit, minmax(280px, 1fr))` creates responsive grids with zero media queries
- Use gap instead of margin hacks; animate with transform and opacity to skip layout and paint

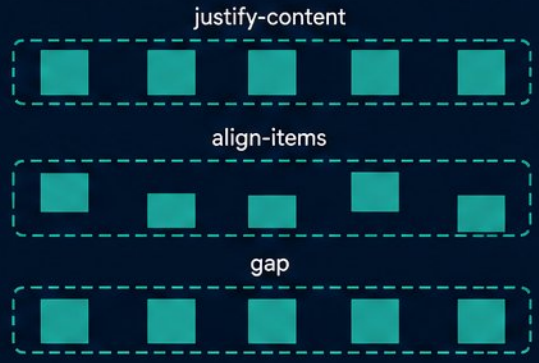
SEMANTIC HTML LANDMARKS

CSS GRID (TWO-DIMENSIONAL)



```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
  gap: 1.5rem;
}
```

FLEXBOX (ONE-DIMENSIONAL)

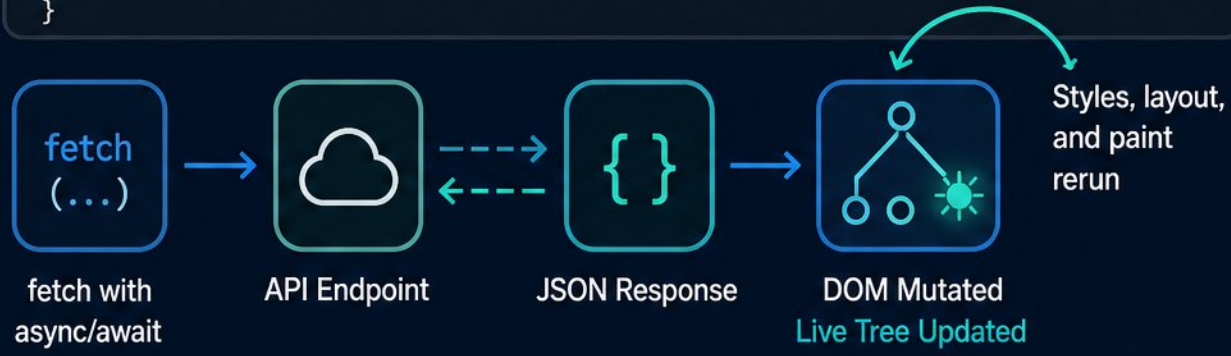


JavaScript in the Browser:
DOM, Events, and Data Fetching

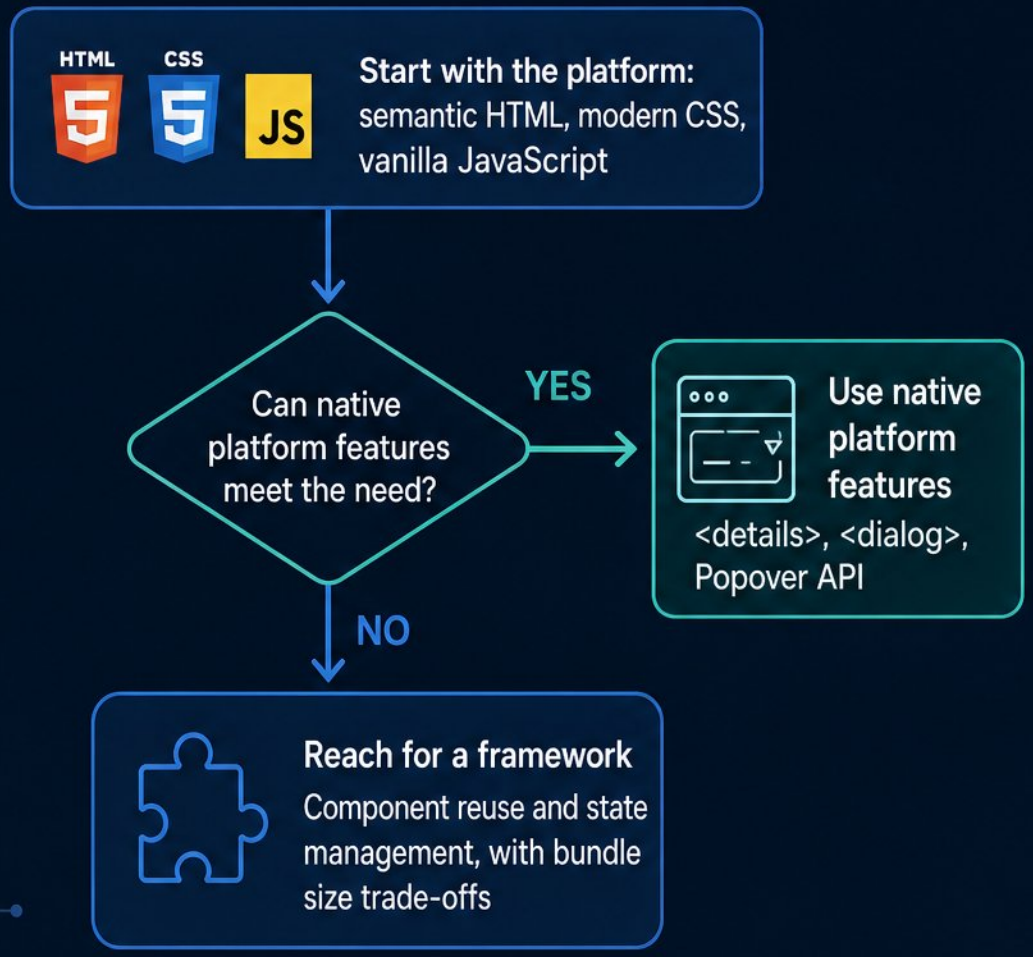
- Core JS applied to the page: variables, functions, arrays, objects, event listeners
- The DOM is a live tree — mutate it and styles, layout, and paint rerun
- Call APIs with fetch and async/await; POST needs Content-Type and JSON.stringify
- Gotcha: fetch resolves on 404 and 500 — always check response.ok before parsing
- Abort stale requests with AbortController and AbortSignal.timeout

```
async function loadData() {
    const controller = new AbortController();
    const timeout = AbortSignal.timeout(5000);
    const signal = AbortSignal.any([controller.signal, timeout]);

    try {
      const res = await fetch('/api/data', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ query: 'example' }),
        signal
      });
      if (!res.ok) throw new Error('Network response was not ok');
      const data = await res.json();
      updateDOM(data);
    } catch (err) {
      if (err.name === 'AbortError') return;
      console.error(err);
    }
}
```



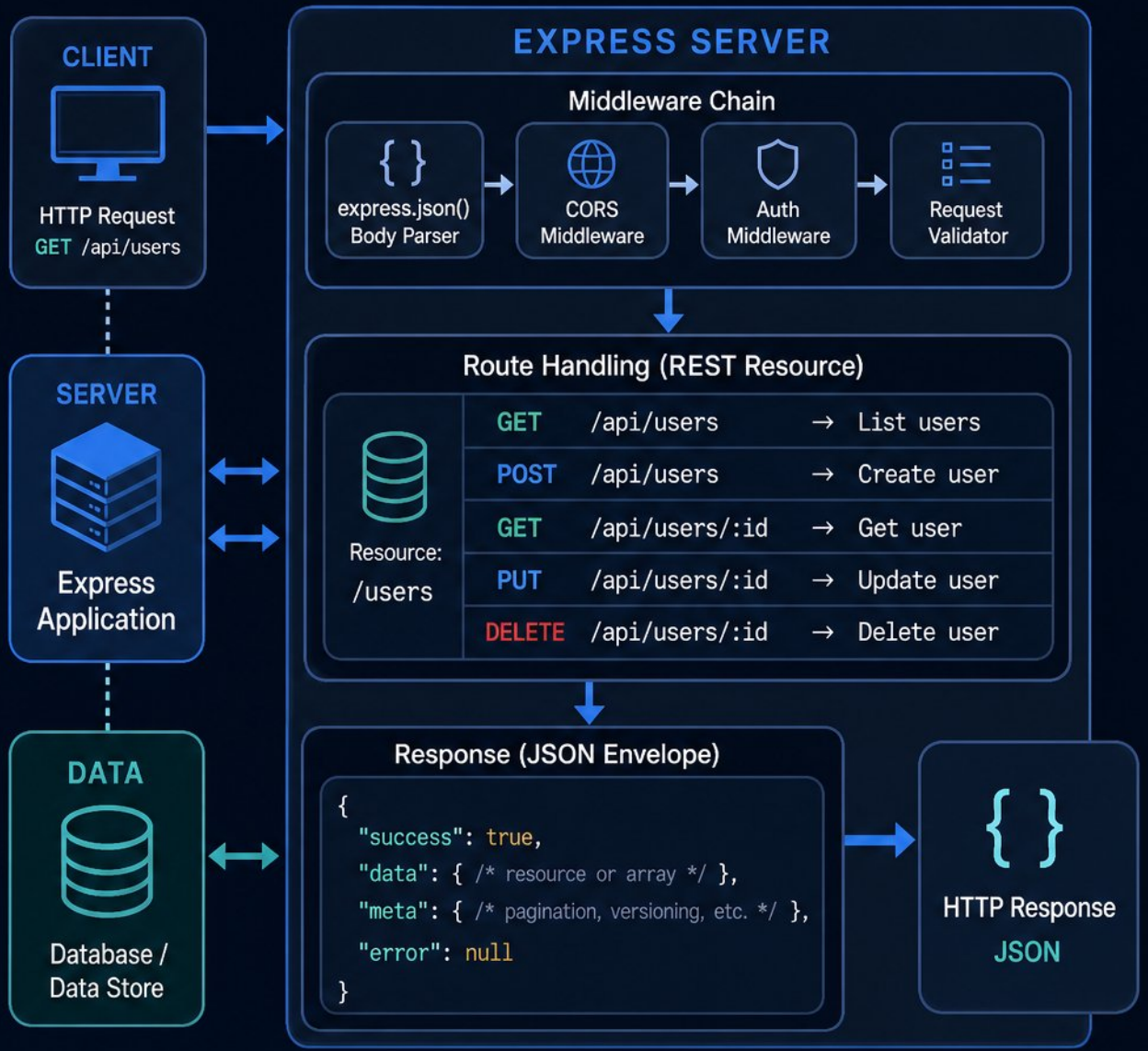
When Plain JavaScript Is Enough and When to Reach for a Framework



- Start with the platform: semantic HTML, modern CSS, vanilla JavaScript
- Native HTML and CSS replace old JS patterns: <details>, <dialog>, Popover API
- Frameworks add component reuse and state management, but cost bundle size
- Decision guide: static sites, dashboards, and highly interactive SPAs differ
- Progressive enhancement layers interactivity only where UX demands it

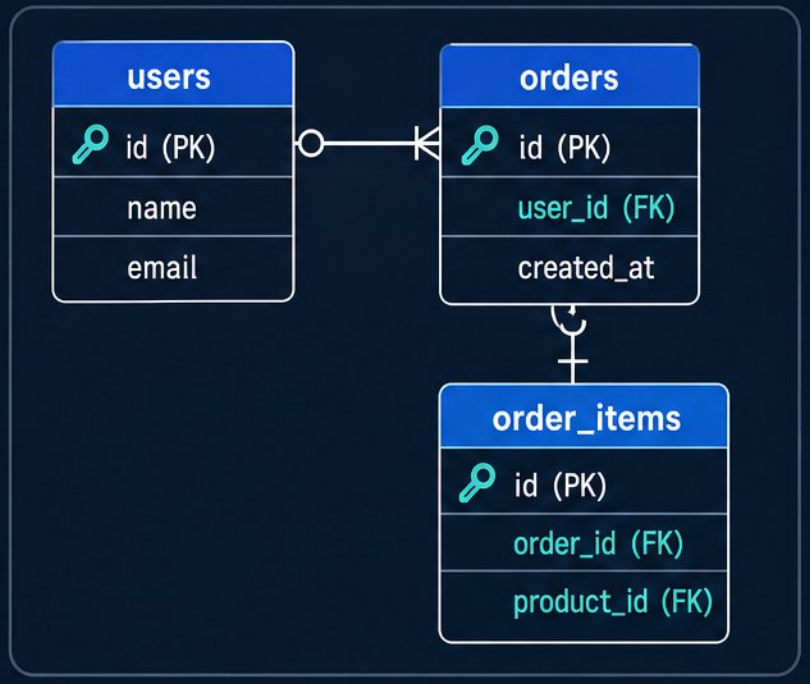
Backend Fundamentals: Servers, REST APIs, and JSON

- Web servers accept requests, route them, apply middleware, run logic, return responses
- Minimal Express app: routes, express.json(), CORS config, and a /health endpoint
- REST design: plural nouns in paths, HTTP methods as verbs, clean CRUD mapping
- Conventions that hold up: precise status codes, structured error envelopes, cursor pagination, versioning from day one
- Auth basics: hash passwords with bcrypt, issue and verify tokens, validate input server-side



Databases: Persistence, Relational Modeling, and CRUD

- Server memory disappears on restart; data belongs in a database
- Tables store rows and columns, linked by primary and foreign keys
- Core SQL: SELECT, INSERT, UPDATE, DELETE, plus JOIN for related data
- Indexes speed lookups like a book's index, with write overhead
- Start with PostgreSQL; add Redis for caching later
- Migrations version your schema; parameterized queries block SQL injection



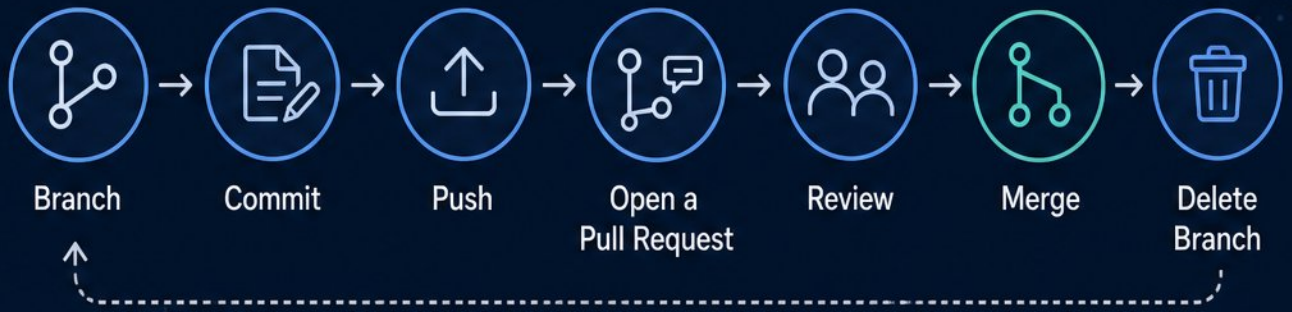
SQL

```
SELECT u.name, o.created_at, oi.product_id
FROM users u
JOIN orders o ON o.user_id = u.id
JOIN order_items oi ON oi.order_id = o.id
WHERE u.id = ?
ORDER BY o.created_at;
```

Development Workflow: Git, GitHub, and Modern Tooling

- Git and GitHub basics: repositories, commits, branches, and remotes
- Daily loop: status, add, commit, pull, push
- Branch, commit, push, open a pull request, review, merge, delete branch
- Vite: instant server start and lightning-fast hot module replacement
- Manage dependencies: read lockfiles, audit what you install
- Debug methodically: read errors, reproduce the bug, then fix it

The collaborative Git and GitHub loop




```
$ git status
$ git add .
$ git commit -m "Update feature"
$ git pull
$ git push
```

Dev Server (Vite)

Instant server start and lightning-fast hot module replacement

Package Manifest

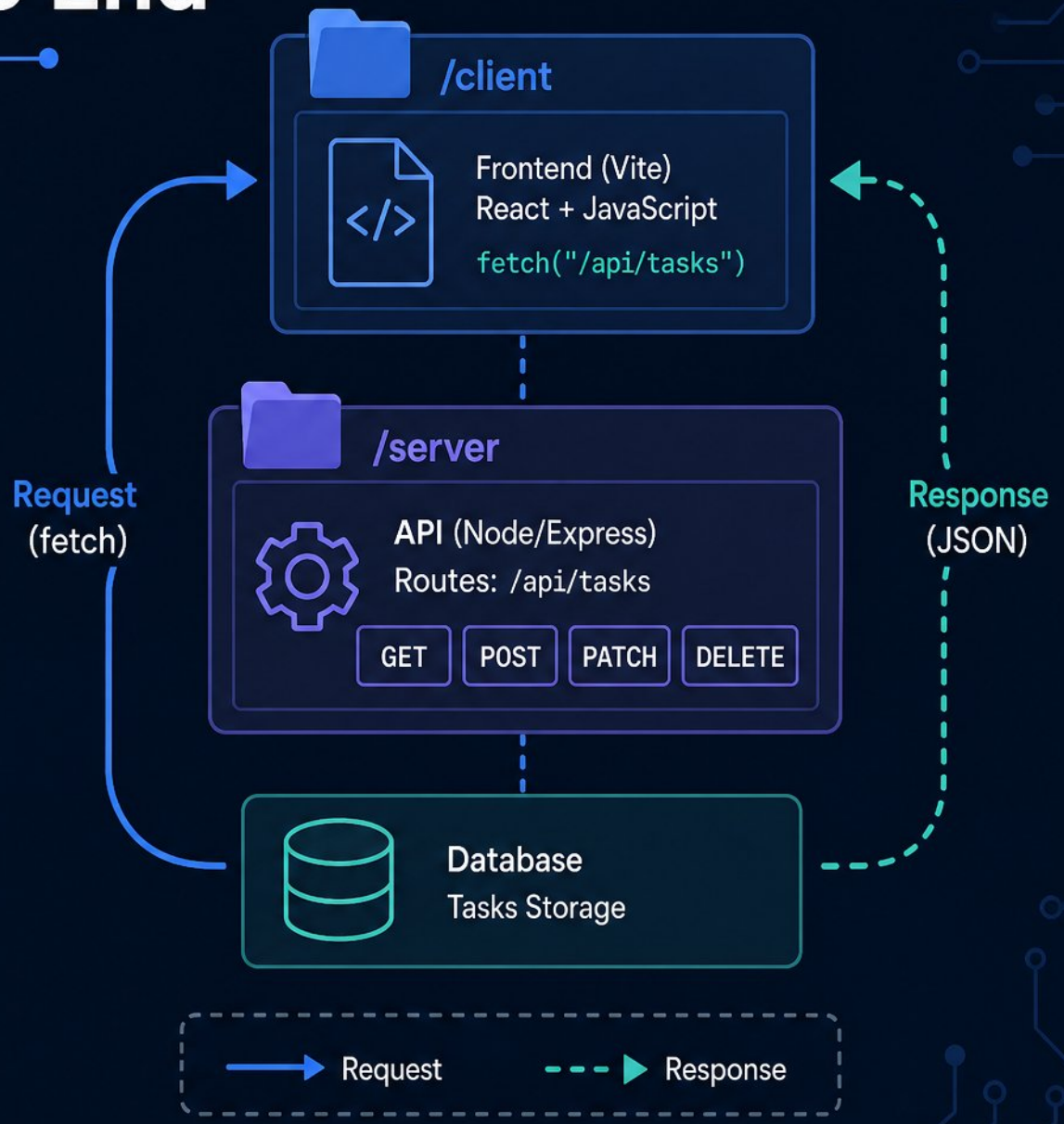
package.json

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {
    "vite": "^latest"
  }
}
```

Manage dependencies: read lockfiles, audit what you install

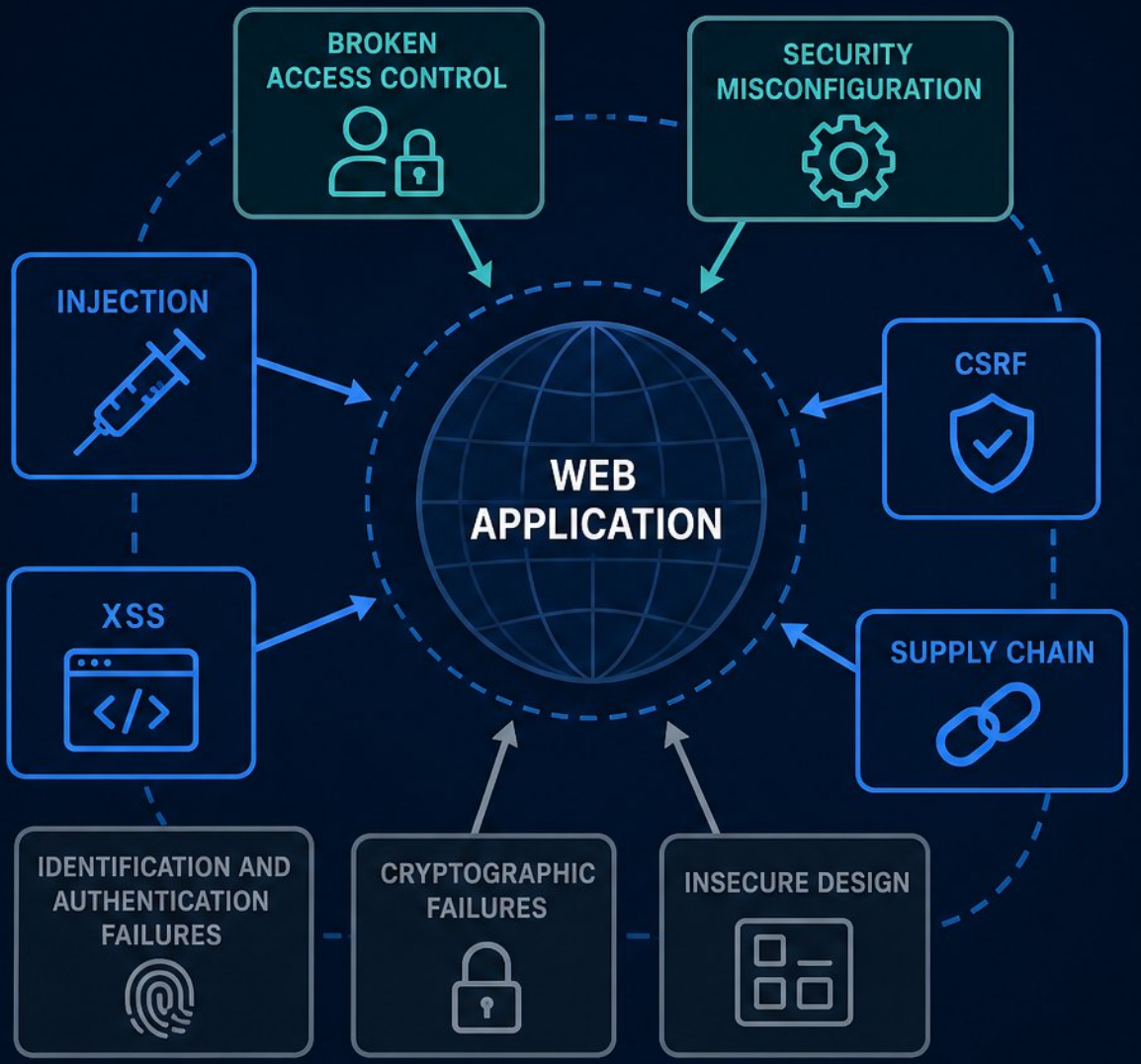
Building Your First Full-Stack Application End to End

- Split project into `/client` and `/server` folders; run both with one command
- Build backend first: task resource with `GET`, `POST`, `PATCH`, `DELETE`
- Wire frontend with fetch; handle loading, empty, and error states
- Validate in both places: client for feedback, server as real boundary
- Debug pitfalls: CORS, Vite proxy, and never let frontend touch the database



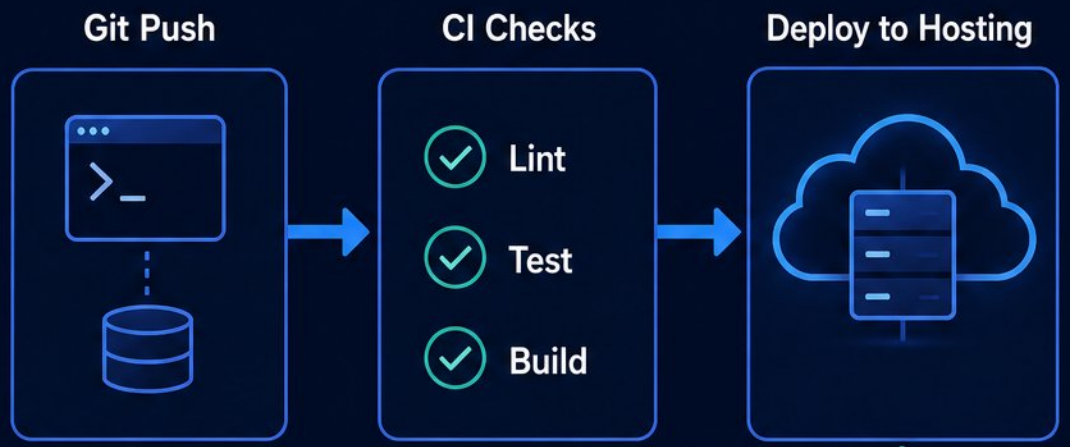
Security Essentials
Every Web Developer Should Know

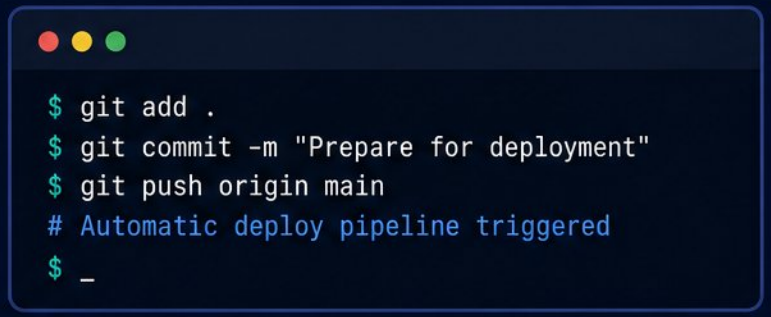
- OWASP Top 10:2025 checklist: Broken Access Control, Misconfiguration, Supply Chain, Injection
- SQL injection stopped by parameterized queries; XSS by output encoding and CSP
- CSRF defended with synchronizer tokens, SameSite cookies, and custom API headers
- HTTPS everywhere: HttpOnly and Secure cookies, bcrypt hashing, secrets in environment variables
- Watch new risks: supply chain failures and failing-open error handling

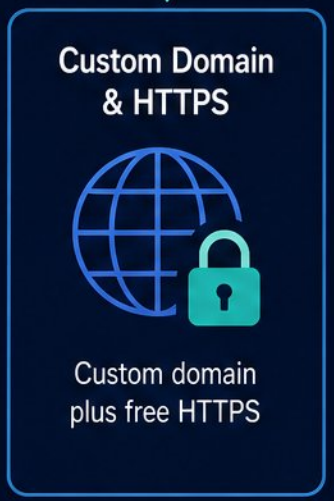


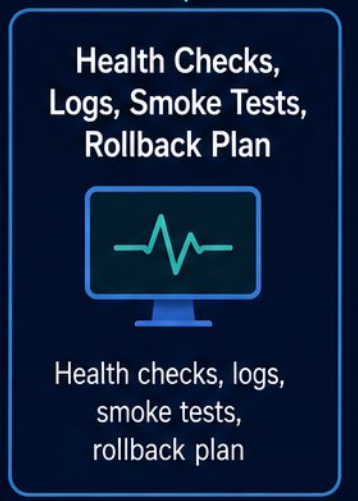
Deployment and Going Live

- Match the project to the platform: static host, framework platform, PaaS, or VPS
- Connect your Git repo: automatic deploys on every push, live in minutes
- Keep secrets in environment variables; never commit .env files
- GitHub Actions: lint, test, and build on PRs; deploy only from main
- Custom domain plus free HTTPS; then health checks, logs, smoke tests, rollback plan







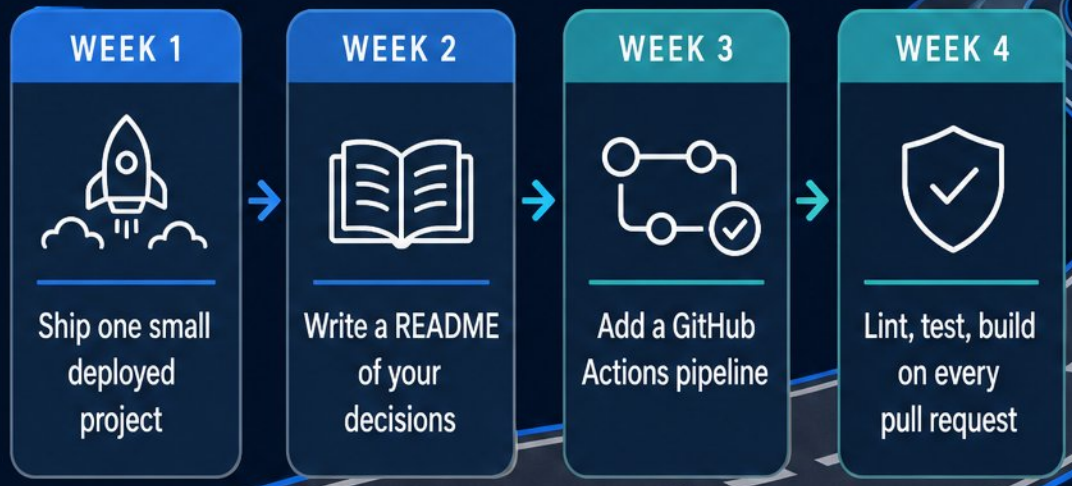


Continuing to Learn: Portfolio, Open Source, and Staying Current

- Choose a direction: frontend, backend, full-stack, DevOps, accessibility, or AI integration
- Portfolio: three to five deployed projects with live URLs and clear READMEs
- One non-tutorial feature proves you can build, not just follow
- Open source: read CONTRIBUTING.md, start from good first issues
- Small focused PRs, sustained monthly, beat a burst of activity
- Resources: MDN, The Odin Project, freeCodeCamp, official framework docs
- Stay current via Baseline, Core Web Vitals, WCAG; review AI output
- Keep building: finished small projects beat collected certificates

Course Wrap-Up: Your Next 30 Days

- Recap: platform, client-server, frontend, APIs, databases, workflow, security, deployment
- Week 1-2: ship one small deployed project with a README of your decisions
- Week 3-4: add a GitHub Actions pipeline — lint, test, build on every pull request
- Steady weekly contributions beat one big burst of activity
- Go deeper next: authentication, Vitest and Playwright, Core Web Vitals, accessibility audits



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/5b7c13ed/public