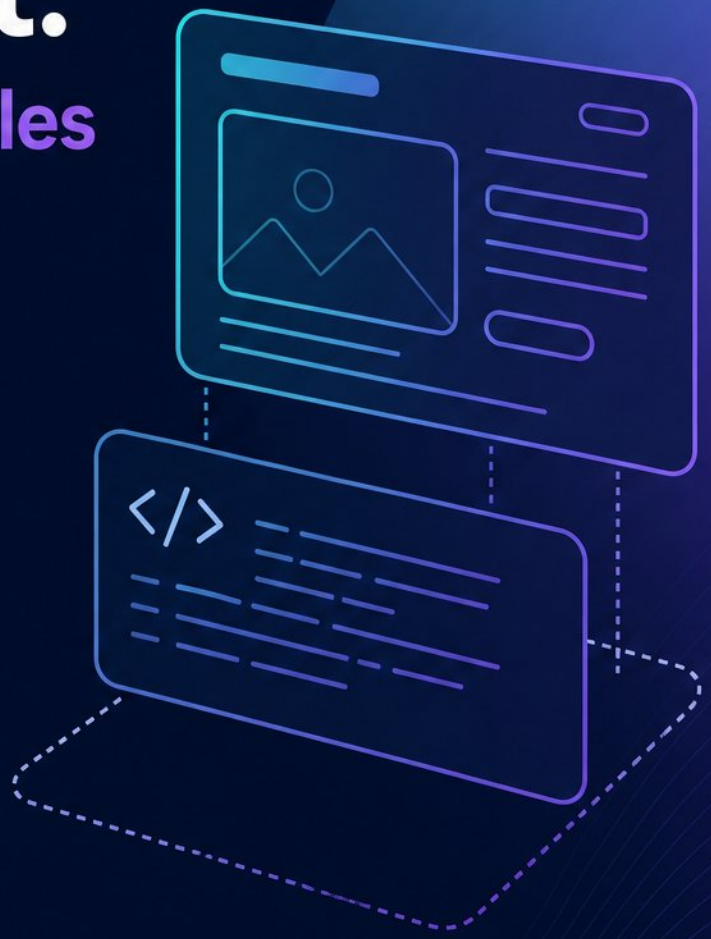
Front End Web Development:

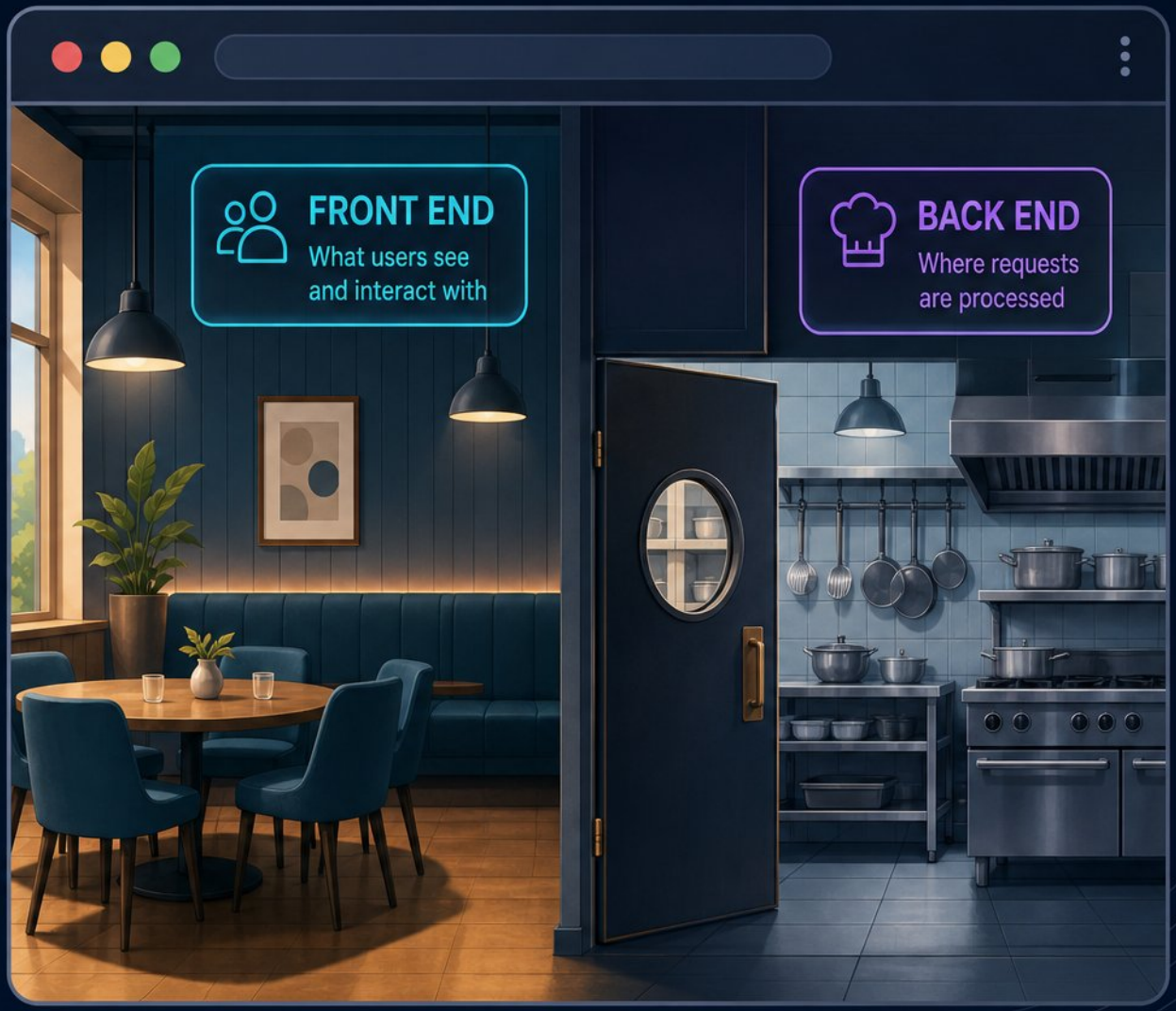
Concepts, Purpose, and Examples

- Building everything users see, click, type into, and navigate in a browser
- Roadmap: core concepts, why it matters, hands-on examples, learning paths
- For beginners, career switchers, designers, students, educators, and product pros
- By the end: read front end code, explain how pages are assembled
- No prior coding required—we build vocabulary and mental models first



What Front End Development Actually Is

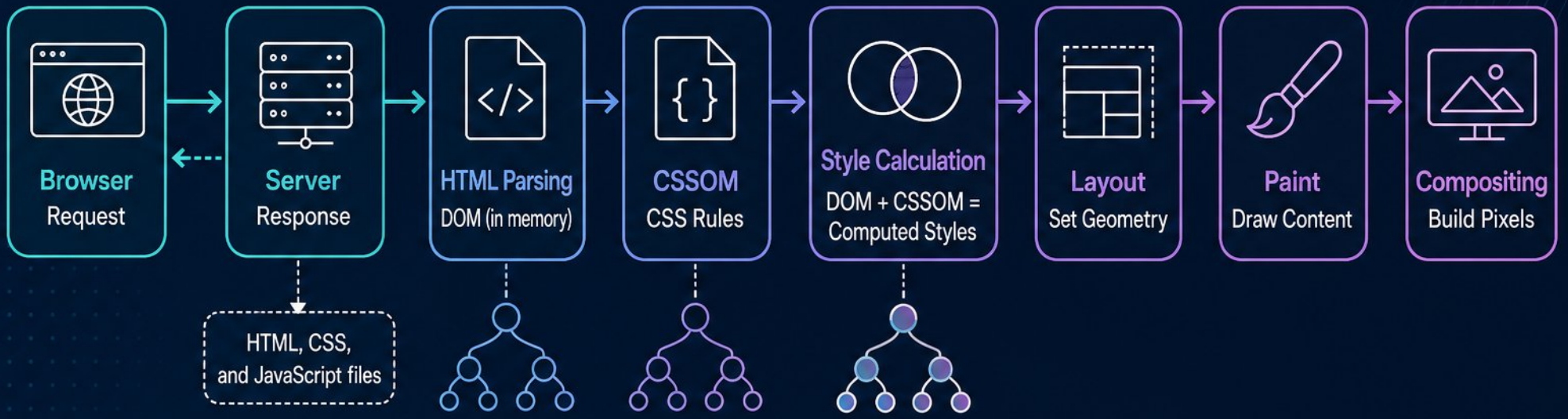
- Builds the visual, interactive parts users see and touch
- The browser is the runtime—code downloads and runs locally
- Front end owns structure, appearance, behavior, and experience
- Back end handles everything after a request leaves the browser
- Like a restaurant: dining area is front end, kitchen is back end



The Three Building Blocks: HTML, CSS, and JavaScript

- **HTML** is structure: what elements exist, their order, and content
- **CSS** is appearance: colors, fonts, spacing, layout, and screen-size adaptation
- **JavaScript** is behavior: clicks, form validation, data fetching, animation
- **HTML** alone is rough; **HTML** + **CSS** is polished but static
- All three together make a real front end — not optional

From a Web Request to a Visible Page

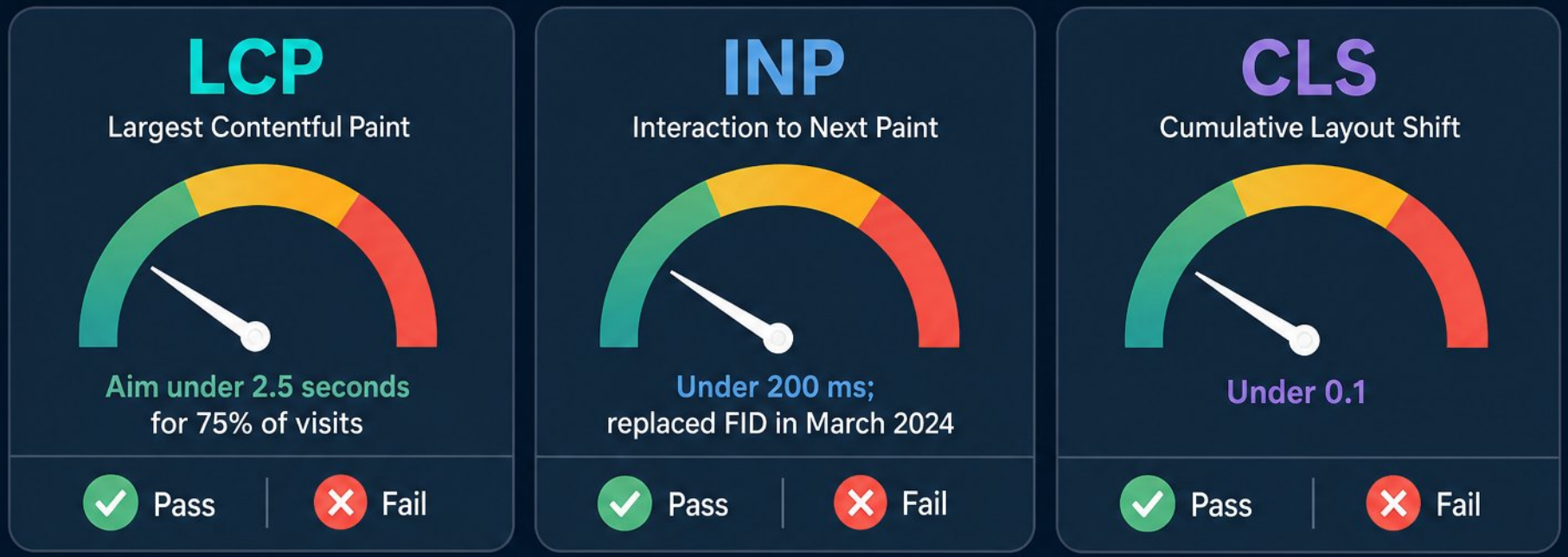


- Browser requests and receives HTML, CSS, and JavaScript files
- HTML is parsed into the DOM, a tree of nodes in memory
- CSS becomes the CSSOM; DOM plus CSSOM computes final styles
- Layout sets geometry, paint draws, compositing builds the pixels
- JavaScript edits the live DOM, not your original HTML file

Why Front End Quality Matters to the Business

- UX decides whether people can actually use the product
- Conversion, retention, trust, and brand rise or fall on the interface
- Speed pays off in conversion — SEO lift is a bonus, not the headline
- Accessibility is legal and ethical: EU Accessibility Act, rising lawsuits
- Page experience signals act as tiebreakers between similar-quality pages

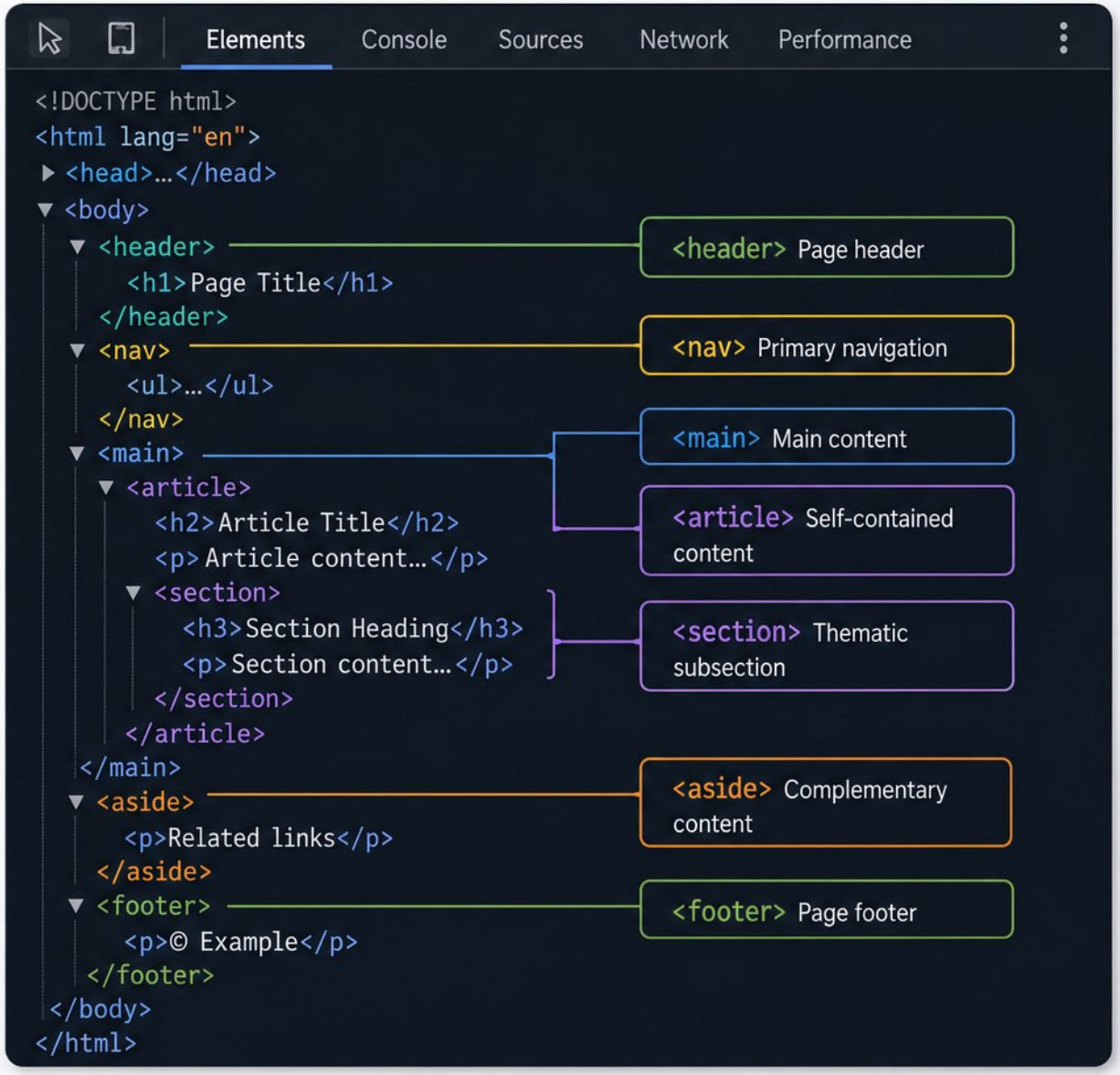
The Language of Performance: LCP, INP, and CLS



- Core Web Vitals: three user metrics, confirmed Google ranking signal since June 2021
- **LCP** measures loading: aim under 2.5 seconds for 75% of visits
- **INP** measures responsiveness: under 200 ms; replaced FID in March 2024
- **CLS** measures visual stability: under 0.1
- All three must pass, based on CrUX field data over 28 days

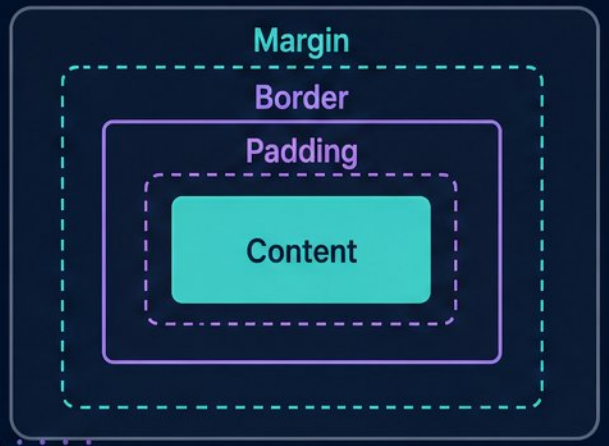
Structuring Content with Semantic HTML and the DOM

- Use elements that describe meaning, not appearance: header, nav, main, article, aside, footer
- Semantic HTML helps accessibility, SEO, and long-term maintainability
- Avoid div soup: nested divs with no meaning confuse screen readers
- Build a solid HTML foundation before styling to prevent rework
- The DOM is the browser's live markup; DevTools' Elements panel shows it in real time



Styling and Layout: Box Model, Flexbox, and Grid

- Every element is a box: margin, border, padding, content
- Set box-sizing: border-box as your first line of CSS
- Flexbox is one-dimensional: navbars, card rows, centering
- Grid is two-dimensional: page layouts, dashboards, galleries
- Rule of thumb: Grid for page structure, Flexbox for component alignment
- Pitfalls: confusing justify-content with align-items, skipping flex-wrap



FLEXBOX
One-Dimensional Layout

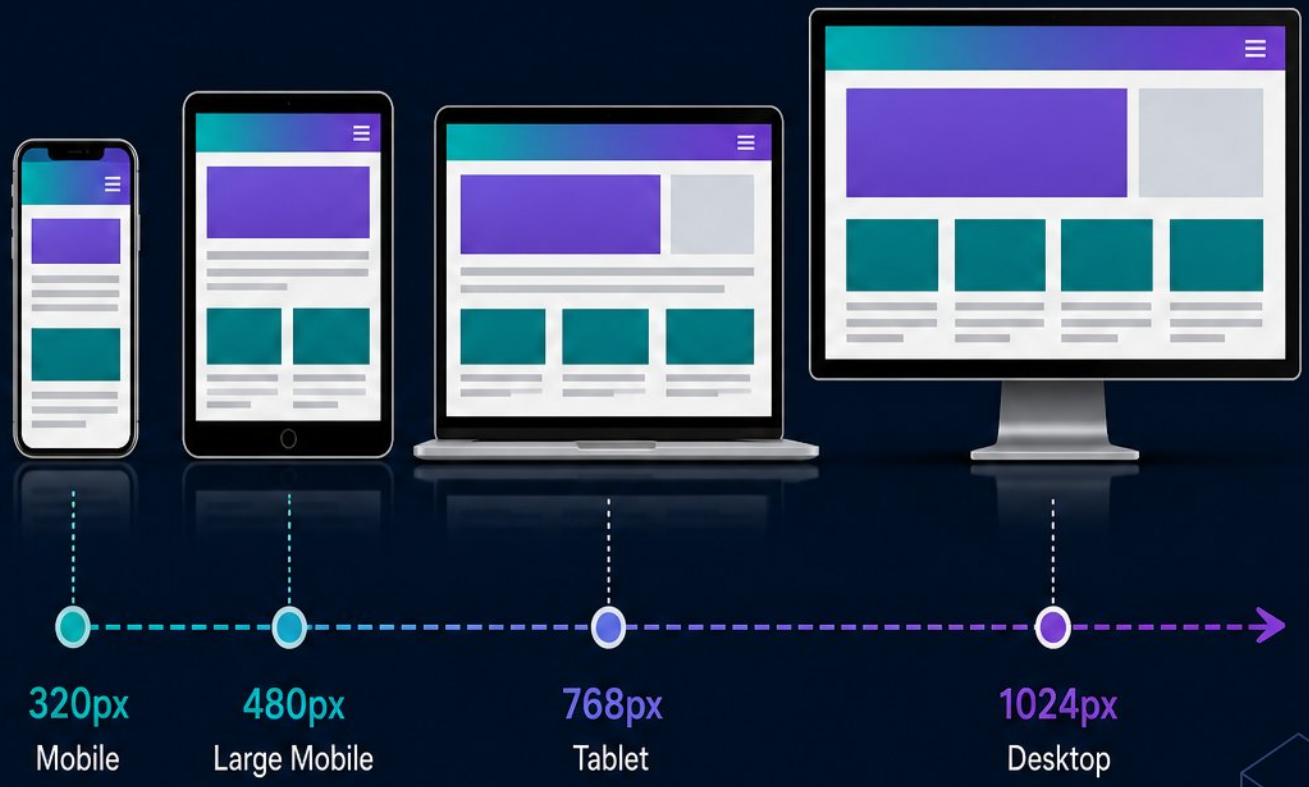
Organizes items along a single axis (either row or column). Ideal for navbars, card rows, and centering content.

CSS GRID
Two-Dimensional Layout

Organizes items in rows and columns simultaneously. Ideal for page layouts, dashboards, and galleries.

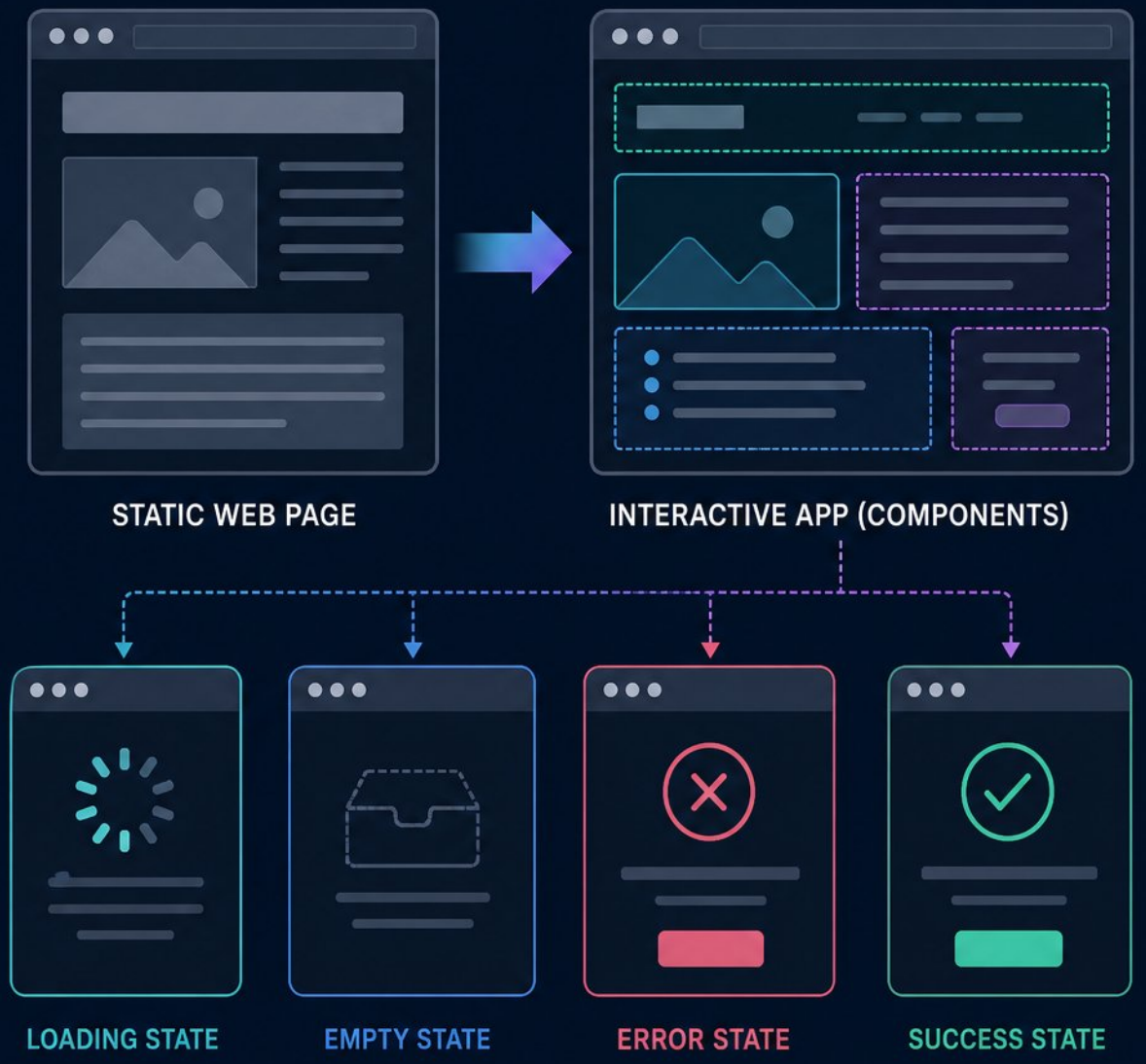
Responsive Design: Reaching Every Screen Size

- Responsive design renders well on all screen sizes and resolutions
- The viewport meta tag is essential: without it, mobile shows desktop at ~980px
- Mobile-first: base styles for small screens, then enhance upward with min-width queries
- Modern CSS cuts breakpoint dependence: clamp(), relative units, auto-fit grid
- Test real widths: 320px, 480px, 768px, and 1024px

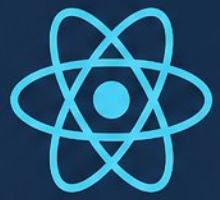


From Static Pages to Interactive Applications

- State and events: the UI updates without full page reloads
- Components break the interface into independent, reusable pieces
- Frameworks (React, Vue, Angular) handle rendering, state, and forms
- APIs fetch data; show loading, empty, error, and success states
- Real projects handle small screens, invalid input, and slow requests



Choosing a First Framework: React, Vue, and Angular



React

Largest ecosystem and hiring pool — safest default for broad job reach

Vue

Gentlest learning curve (5–10 days) and excellent docs — best pure learning start

Angular

Steepest curve (21+ days), strongest in enterprise and TypeScript teams

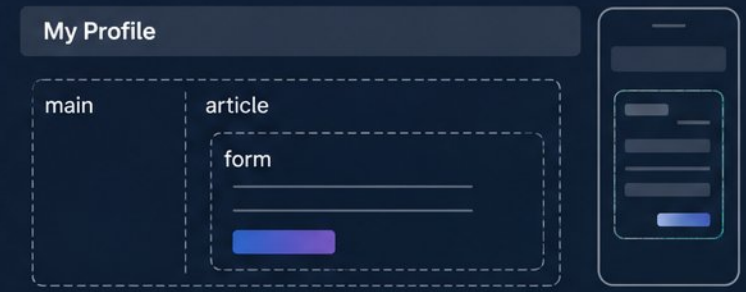
- React ~7–14 days to productivity; Angular requires TypeScript, DI, and RxJS upfront
- The bad choice is learning syntax without forms, routing, data loading, and error states
- Pick by target: React for optionality, Angular for enterprise, Vue for your existing team

A Practical Example: Building a Simple Profile Page

1. **Structure with semantic HTML:** header, main, article, form
2. **Style with CSS:** Flexbox header, Grid card layout
3. Add one JavaScript interaction: toggle or form validation
4. Show empty, error, and success states visibly
5. Then rebuild as a component to see the framework's value

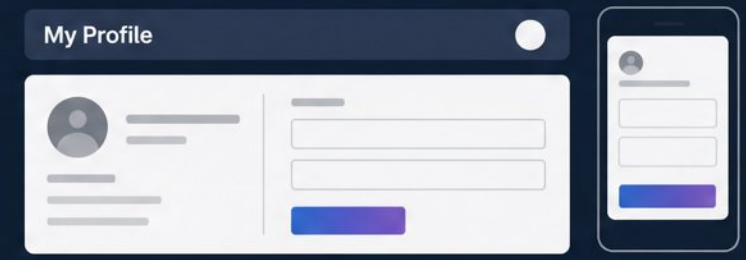
HTML

```
<header class="site-header">
  <h1>My Profile</h1>
</header>
<main>
  <article class="profile-card">
    <h2>Profile</h2>
    <form id="profile-form"> … </form>
  </article>
</main>
```



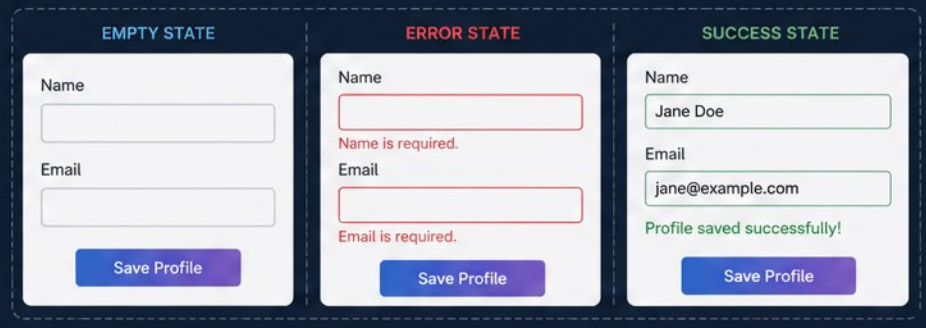
CSS

```
.site-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
.profile-card {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 1rem;
}
```



JS

```
const form = document.getElementById('profile-form');
form.addEventListener('submit', (e) => {
  e.preventDefault();
  const name = form.name.value.trim();
  const email = form.email.value.trim();
  if (!name || !email) {
    showState('error');
  } else {
    showState('success');
    form.reset();
  }
});
```

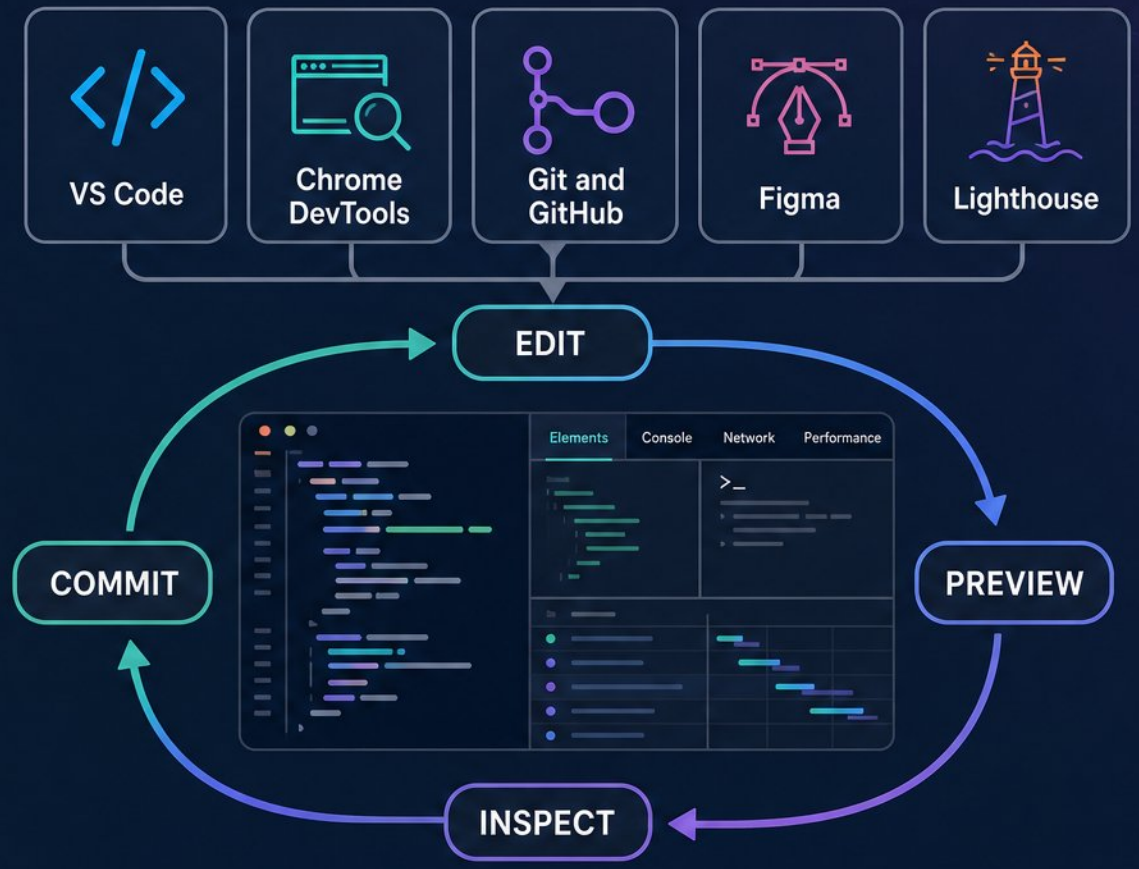


COMPONENT

```
<ProfileCard
  initialData={{ name: '', email: '' }}
  onSave={handleSave}
/>
```

Tools, Workflow, and the Debugging Habit

- Essential five: VS Code, Chrome DevTools, Git and GitHub, Figma, Lighthouse
- DevTools panels: Elements for HTML/CSS, Console for JS, Network for APIs, Performance for bottlenecks
- Vite has replaced Create React App as the 2026 standard build tool
- Commit every learning-phase project to GitHub; employers review profiles
- Practice the loop: edit, preview, inspect, commit—audit performance and accessibility as you build

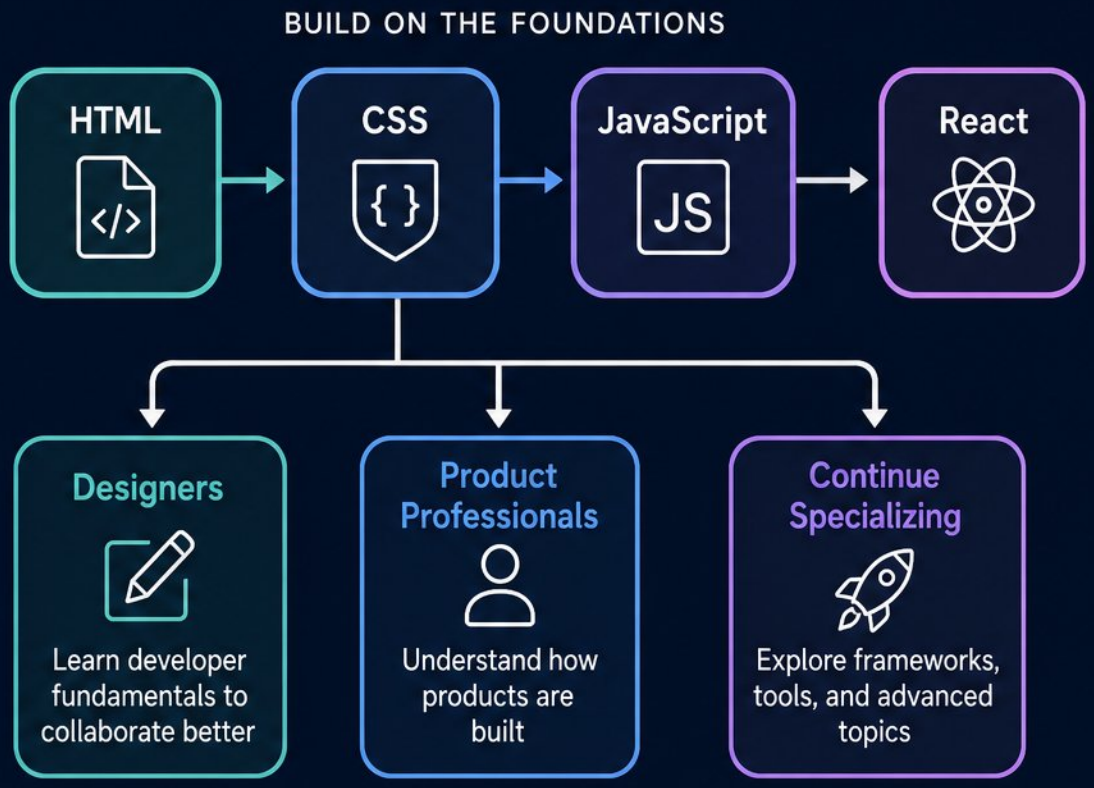


DevTools panels:
Elements for HTML/CSS, Console for JS, Network for APIs, Performance for bottlenecks

Audit performance and accessibility as you build

Key Terms, Learning Paths, and Next Steps

- SPA loads once and updates via JavaScript; SSR renders HTML per request
- Hydration attaches JavaScript to server-rendered HTML to make it interactive
- React is a library; Angular is a full framework—SPA differs from SSR/SSG
- Free paths: MDN, freeCodeCamp, The Odin Project, roadmap.sh, official docs
- Job readiness: 6–12 months at 15–20 hours weekly; projects beat certificates



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/2135bca0/public