Client-Side Web Development

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- ➔ Evolution of Web Client-Side Development
- ➔ Web Architecture and Technologies
- ➔ Core Technologies
- ➔ Development Tools
- ➔ Progressive Web Applications

Evolution of Client-Side Web Development

Static Web Pages (Early 1990s)

- ➔ HTML files served directly from the server's filesystem without processing.
- ➔ Development centered on HTML markup, with content created at design time.
- ➔ Browser's "View Source" feature served as the primary learning tool for developers.
- ➔ Content consisted of text, images, and hyperlinks between documents.
- ➔ Communication followed a request-response model where servers delivered complete HTML pages.
- ➔ URLs mapped directly to files stored on the web server's disk.

Dynamic Pages (Mid 1990s)

- ➔ Servers generated HTML content through CGI programs and server-side scripts.
- ➔ HTML forms enabled users to submit data and interact with server applications.
- ➔ Table-based layouts provided structure for page content organization.
- ➔ CSS specification introduced separation of content from presentation.
- ➔ JavaScript brought interactivity and client-side form validation to browsers.
- ➔ Session management enabled state preservation between page requests.

Web Applications (early 2000s)

- ➔ Server-side frameworks established patterns for application development.
- ➔ CSS gained consistent implementation across major browsers.
- ➔ JavaScript evolved to handle sophisticated DOM manipulation.
- ➔ XMLHttpRequest enabled asynchronous server communication.
- ➔ Browser developer tools improved debugging capabilities.
- ➔ Web standards convergence reduced cross-browser inconsistencies.

Rich Web Applications (2010s)

- ➔ Asynchronous requests enabled partial page updates without reloads (AJAX).
- ➔ JavaScript frameworks like jQuery standardized DOM manipulation.
- ➔ Single Page Applications (SPAs) introduced new patterns for web architecture.
- ➔ Responsive design addressed the proliferation of mobile devices.
- ➔ HTML5 and CSS3 brought native support for multimedia and animations.
- ➔ Build tools automated asset compilation and optimization.
- ➔ Package managers centralized dependency management.

Modern Architecture (2020s)

- ➔ React, Vue, and Angular established component architectures.
- ➔ State management libraries address complex data flows.
- ➔ TypeScript adds static typing to JavaScript development.
- ➔ Server components blend client and server rendering.
- ➔ Edge computing moves processing closer to users.
- ➔ Web Assembly enables near-native performance.

Evolution Summary

- **Client-Side Web Development Evolution**:
  - Shift from document-centric to application-centric development
  - Growing importance of client-side processing
  - Increasing focus on user experience and performance
- **Modern Challenges**:
  - Balancing performance and functionality
  - Managing growing application complexity
  - Supporting diverse devices and browsers
  - Ensuring security and privacy

Technologies and Architectures

Client-Server Model

- ➔ Web browsers issue requests to web servers for resources.
- ➔ Servers process requests and return appropriate responses.
- ➔ HTTP protocol governs communication between clients and servers.
- ➔ Each request-response cycle is independent (stateless).
- ➔ Browsers handle rendering and user interaction.
- ➔ Servers manage data storage and business logic.

Client-Side Technology Stack

➔ **HTML** provides document structure and content.

➔ **CSS** controls layout and visual presentation.

➔ **JavaScript** enables interactivity and dynamic behavior.

➔ Web APIs offer access to browser features and capabilities.

➔ Build tools optimize code for production deployment.

➔ Package managers handle dependency management.

Browser Rendering Pipeline

- ➔ HTML parsing creates the Document Object Model (DOM).
- ➔ CSS parsing builds the CSS Object Model (CSSOM).
- ➔ Render tree combines DOM and CSSOM.
- ➔ Layout process determines element positions and sizes.
- ➔ Paint stage renders pixels to the screen.
- ➔ Compositing layers optimizes rendering performance.

Server-Based Architecture

- ➔ Server generates complete HTML pages.
- ➔ Each user action triggers a page reload.
- ➔ Session state maintained on server.
- ➔ JavaScript enhances existing functionality.
- ➔ Forms submit data to server endpoints.
- ➔ Links navigate between distinct pages.

Client-Based Architecture

- ➔ Client maintains application state.
- ➔ Server provides data through APIs.
- ➔ Updates occur without page reloads.
- ➔ Components manage their own logic and presentation.
- ➔ Client-side routing handles navigation.
- ➔ Data synchronizes bidirectionally.

Web Application Architectures

- ➔ **Multi-Page Applications** (MPA) use server-side rendering.
- ➔ **Single Page Applications** (SPA) handle rendering on the client.
- ➔ **Progressive Web Apps** (PWA) add native application features.
- ➔ **Static Site Generators** (SSG) pre-renders pages at build time.

- ➔ **Server-Side Rendered** (SSR) apps combine client and server rendering.
- ➔ **Client-Side Rendered** (CSR) apps build the interface in the browser.

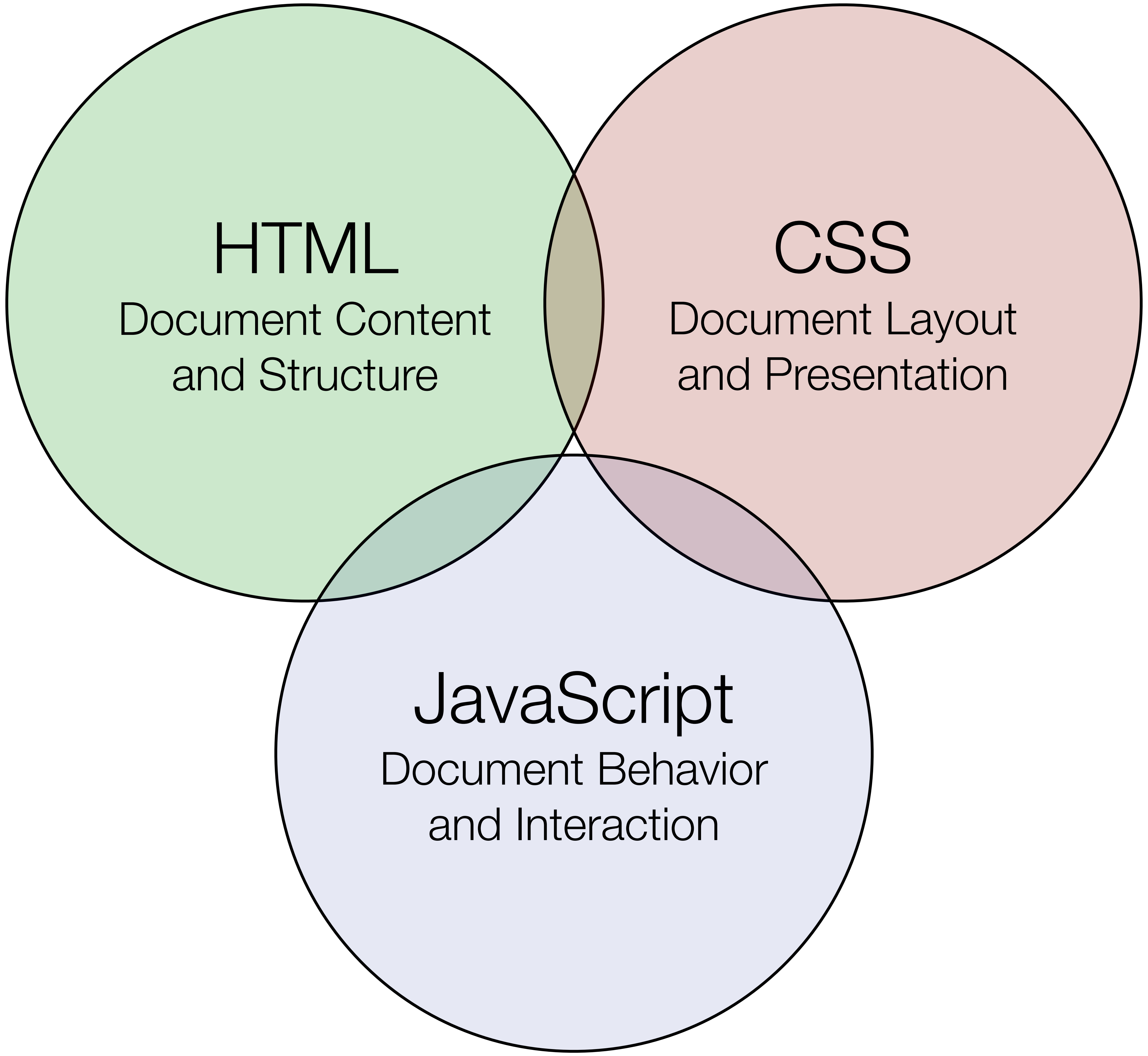
Client-Side Core Technologies

Three Pillars of Client-Side Development

- ➔ **HyperText Markup Language** (HTML) is a markup language for structuring and linking web documents — defines content and structure.
- ➔ **Cascading Style Sheets** (CSS) is a declarative language for controlling document presentation — defines layout and visual presentation.
- ➔ **JavaScript** is a programming language designed to create interactive web applications — defines behavior and functionality.

- ➔ W3C and WHATWG maintain HTML and CSS standards.
- ➔ ECMA International standardizes JavaScript through ECMAScript specifications.

Three Pillars of Client-Side Development



HTML

Document Content and Structure

CSS

Document Layout and Presentation

JavaScript

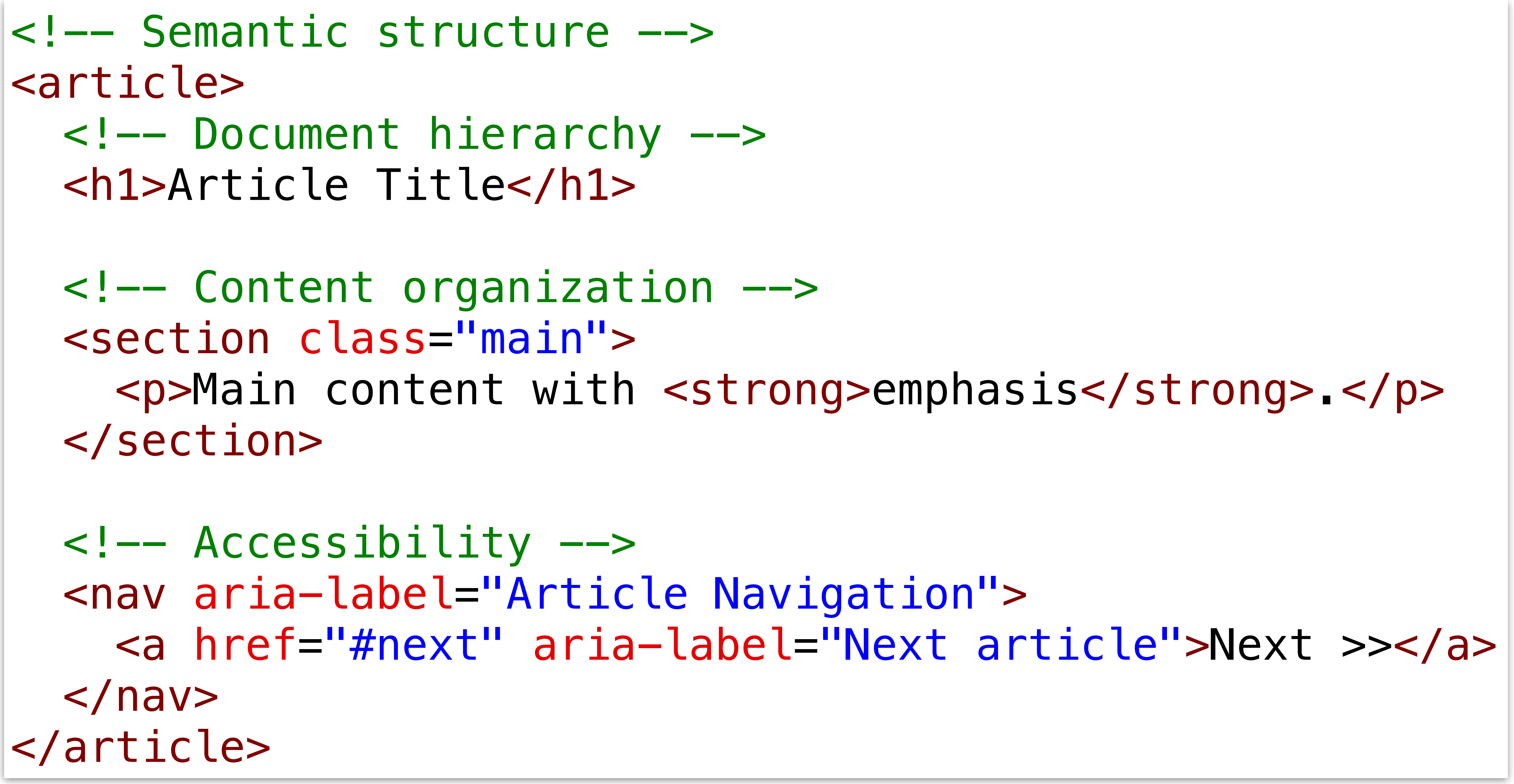
Document Behavior and Interaction

HTML: Content + Structure

- ➔ **Elements** are the building blocks of HTML documents, using tags to structure content.
- ➔ **Attributes** provide additional properties that customize element behavior.
- ➔ **Hyperlinks** connect documents, allowing for web navigation.
- ➔ **Semantic Structure** organizes content into meaningful sections and hierarchies.

- ➔ WHATWG develops and maintains the HTML Living Standard (html.spec.whatwg.org).
- ➔ W3C provides HTML recommendations and guidelines for implementation.

HTML: Content + Structure



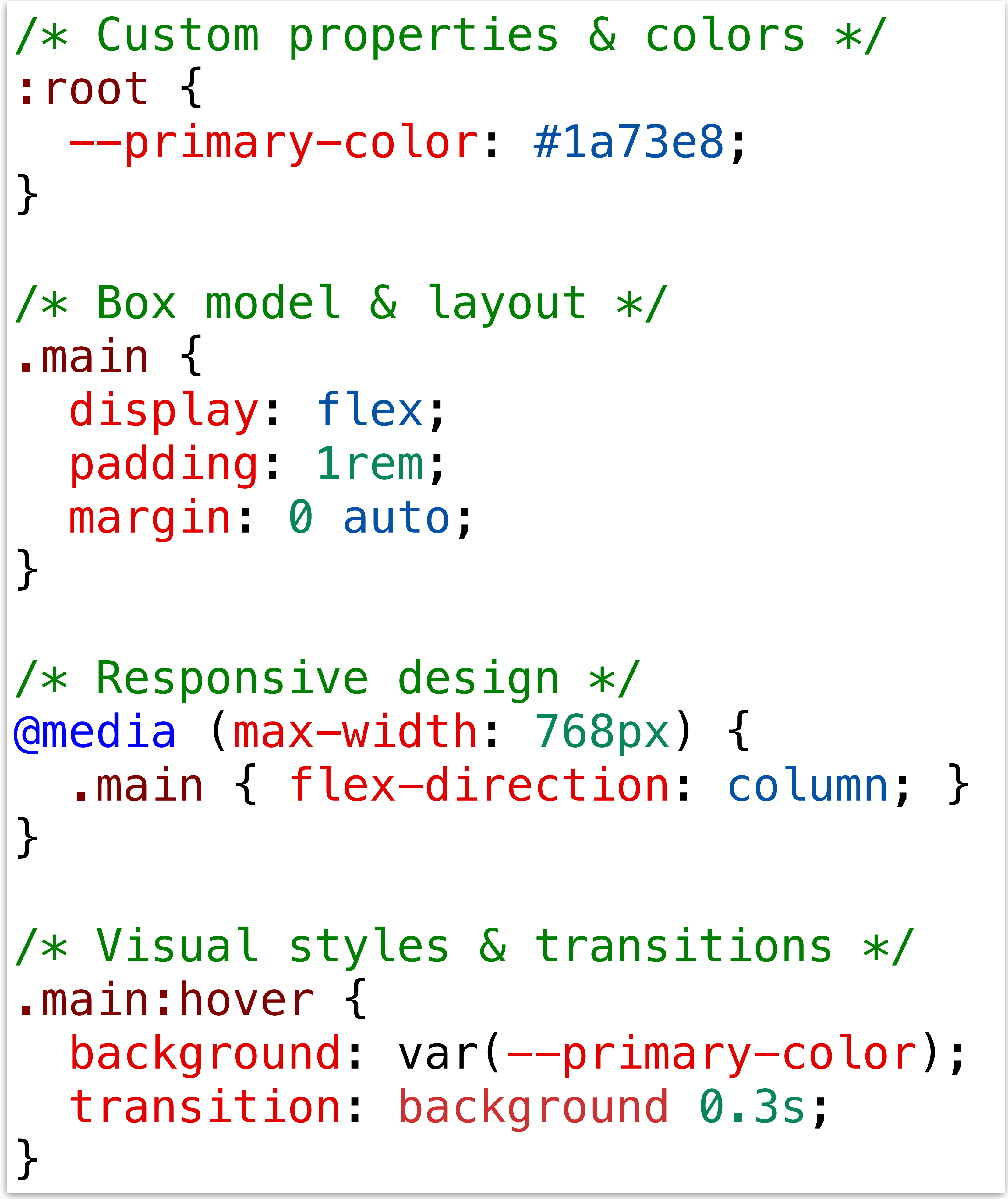
```html
<!-- Semantic structure -->
<article>
  <!-- Document hierarchy -->
  <h1>Article Title</h1>

  <!-- Content organization -->
  <section class="main">
    <p>Main content with <strong>emphasis</strong>.</p>
  </section>

  <!-- Accessibility -->
  <nav aria-label="Article Navigation">
    <a href="#next" aria-label="Next article">Next >></a>
  </nav>
</article>
```

CSS: Presentation + Layout

- ➔ **Selectors** identify elements to style using pattern matching expressions.
- ➔ **Properties** define visual aspects like colors, sizes, and layouts.
- ➔ **Box Model** handles spacing and sizing of elements on the page.
- ➔ **Media Queries** enable responsive layouts across different devices.

- ➔ W3C maintains the CSS specifications and recommendations.

CSS: Presentation + Layout



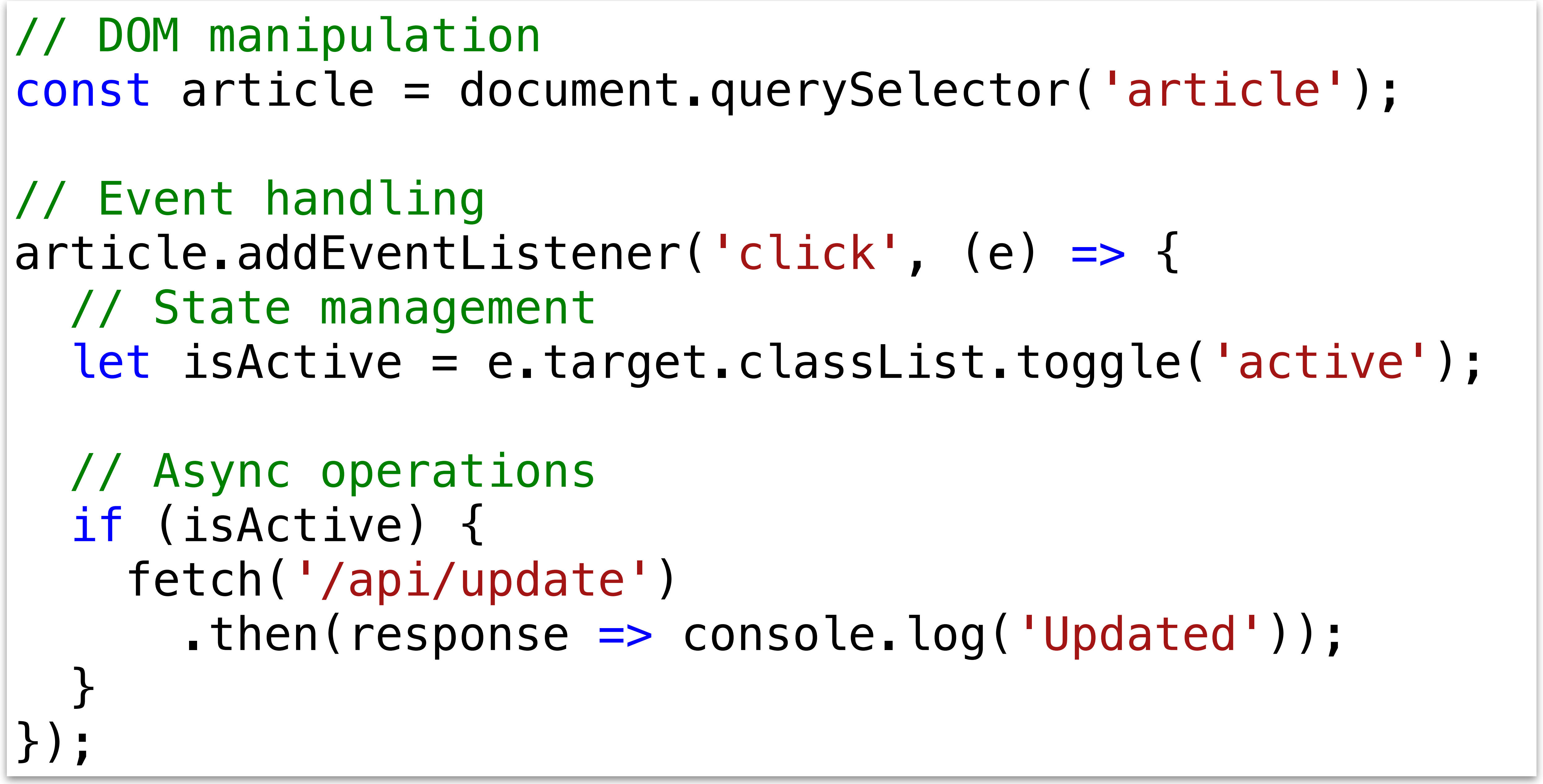
```css
/* Custom properties & colors */
:root {
  --primary-color: #1a73e8;
}

/* Box model & layout */
.main {
  display: flex;
  padding: 1rem;
  margin: 0 auto;
}

/* Responsive design */
@media (max-width: 768px) {
  .main { flex-direction: column; }
}

/* Visual styles & transitions */
.main:hover {
  background: var(--primary-color);
  transition: background 0.3s;
}
```

JavaScript: Behavior + Interaction

- ➔ **Programming Language** designed specifically for client-side web applications.
- ➔ **Event System** responds to user actions and browser changes.
- ➔ **DOM APIs** allow dynamic manipulation of web documents.
- ➔ **Asynchronous Operations** handle timing and external data fetching.
- ➔ **Web APIs** provide access to browser functionality.

- ➔ ECMA International defines the ECMAScript standard.

JavaScript: Behavior + Interaction



```
// DOM manipulation
const article = document.querySelector('article');

// Event handling
article.addEventListener('click', (e) => {
  // State management
  let isActive = e.target.classList.toggle('active');

  // Async operations
  if (isActive) {
    fetch('/api/update')
      .then(response => console.log('Updated'));
  }
});
```

Browser Rendering

- ➔ **HTML Parser** converts markup into Document Object Model.
- ➔ **CSS Engine** applies styles to the document elements.
- ➔ **Layout Engine** calculates positions and dimensions.
- ➔ **Paint System** renders visual elements to the screen.
- ➔ **JavaScript Engine** executes program code.

- ➔ Each browser implements its own rendering engine.

Web APIs

- ➔ **DOM Manipulation** interfaces with the document structure.
- ➔ **Network Requests** handle data exchange with servers.
- ➔ **Storage manages** client-side data persistence.
- ➔ **Graphics** enable 2D and 3D rendering.
- ➔ **Media** controls audio and video playback.
- ➔ …

Development Tools

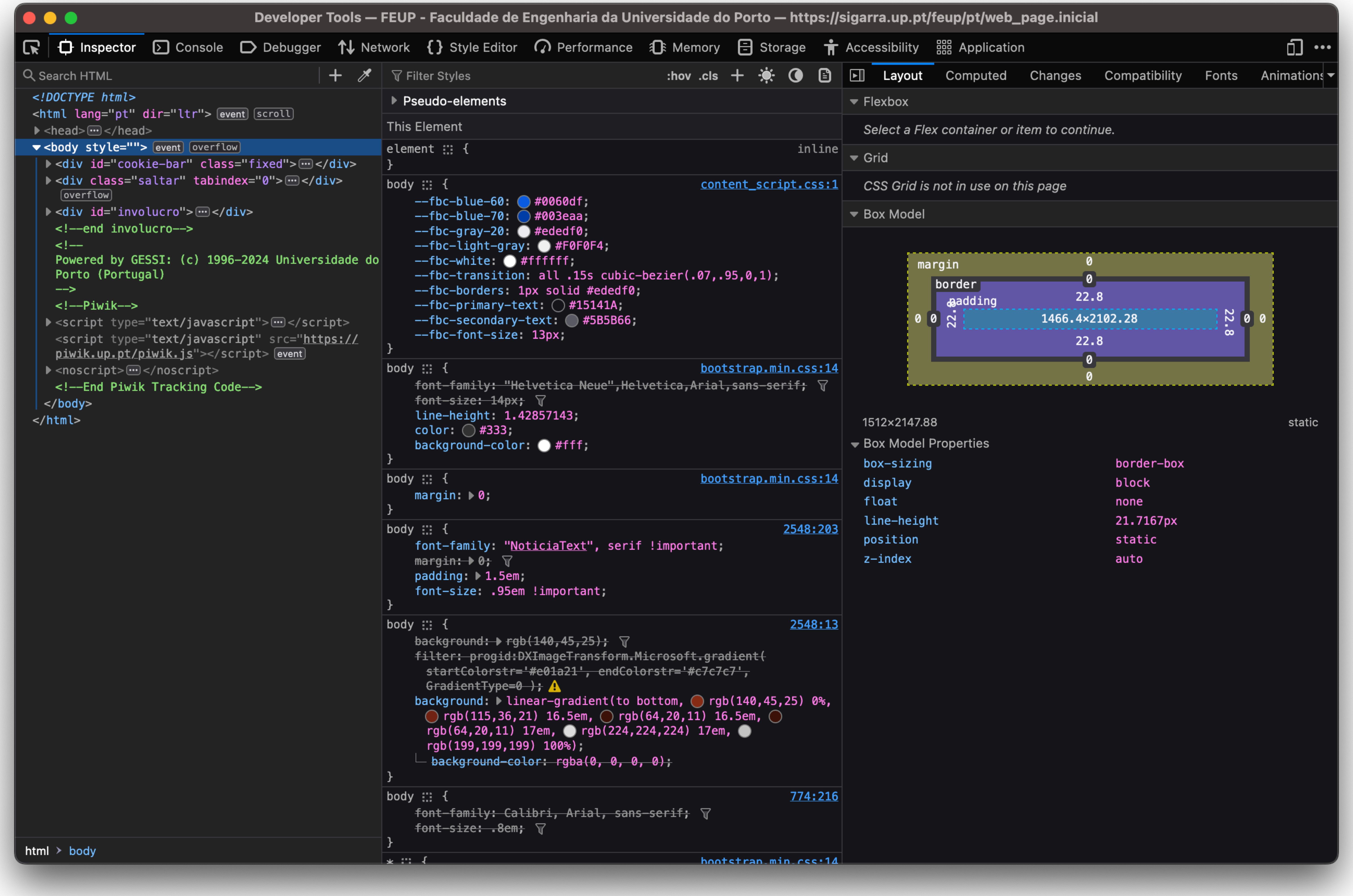
Client-Side Web Development

- ➔ Modern web development requires more than HTML, CSS, and JavaScript.
- ➔ Modern interfaces require tools for development, building, testing, and deployment.
- ➔ Package managers handle growing ecosystem of frontend dependencies.
- ➔ Component architectures organize interfaces into reusable parts.
- ➔ Browser compatibility remains a constant development concern.

Development Tools

- ➔ Browser DevTools provide inspection and debugging capabilities for live applications.
- ➔ Code Editors offer specialized features for web technology development.
- ➔ Development Servers enable local testing and provide instant feedback.
- ➔ Version Control manages code changes and team collaboration.
- ➔ Testing Frameworks validate application behavior.

Browser Developer Tools



Developer Tools — FEUP - Faculdade de Engenharia da Universidade do Porto — https://sigarra.up.pt/feup/pt/web_page.inicial

```
<!DOCTYPE html>
<html lang="pt" dir="ltr"> event scroll
  <head>⋯</head>
  <body style=""> event overflow
    <div id="cookie-bar" class="fixed">⋯</div>
    <div class="saltar" tabindex="0">⋯</div>
     overflow
    <div id="involucro">⋯</div>
    <!--end involucro-->
    <!--
    Powered by GESSI: (c) 1996-2024 Universidade do
    Porto (Portugal)
    -->
    <!--Piwik-->
    <script type="text/javascript">⋯</script>
    <script type="text/javascript" src="https://
    piwik.up.pt/piwik.js"></script> event
    <noscript>⋯</noscript>
    <!--End Piwik Tracking Code-->
  </body>
</html>
```

html › body

Pseudo-elements

This Element

```
element {                                                      inline
}
body {                                            content_script.css:1
    --fbc-blue-60: #0060df;
    --fbc-blue-70: #003eaa;
    --fbc-gray-20: #ededf0;
    --fbc-light-gray: #F0F0F4;
    --fbc-white: #ffffff;
    --fbc-transition: all .15s cubic-bezier(.07,.95,0,1);
    --fbc-borders: 1px solid #ededf0;
    --fbc-primary-text: #15141A;
    --fbc-secondary-text: #5B5B66;
    --fbc-font-size: 13px;
}
body {                                           bootstrap.min.css:14
    ~~font-family: "Helvetica Neue",Helvetica,Arial,sans-serif;~~
    ~~font-size: 14px;~~
    line-height: 1.42857143;
    color: #333;
    background-color: #fff;
}
body {                                           bootstrap.min.css:14
    margin: 0;
}
body {                                                       2548:203
    font-family: "NoticiaText", serif !important;
    ~~margin: 0;~~
    padding: 1.5em;
    font-size: .95em !important;
}
body {                                                        2548:13
    ~~background: rgb(140,45,25);~~
    ~~filter: progid:DXImageTransform.Microsoft.gradient(~~
      ~~startColorstr='#e01a21', endColorstr='#c7c7c7',~~
      ~~GradientType=0 );~~ ⚠
    background: linear-gradient(to bottom, rgb(140,45,25) 0%,
      rgb(115,36,21) 16.5em, rgb(64,20,11) 16.5em,
      rgb(64,20,11) 17em, rgb(224,224,224) 17em,
      rgb(199,199,199) 100%);
    └ ~~background-color: rgba(0, 0, 0, 0);~~
}
body {                                                        774:216
    ~~font-family: Calibri, Arial, sans-serif;~~
    ~~font-size: .8em;~~
}
* {                                              bootstrap.min.css:14
```

Flexbox

Select a Flex container or item to continue.

Grid

CSS Grid is not in use on this page

Box Model

1512×2147.88 static

Box Model Properties

box-sizing	border-box
display	block
float	none
line-height	21.7167px
position	static
z-index	auto

Why Build Tools

- ➔ Development code uses modern features not supported directly by browsers.
- ➔ Multiple JavaScript files need management and dependency tracking.
- ➔ CSS requires preprocessing for maintainable stylesheets.
- ➔ Assets need optimization for production delivery.
- ➔ Source code requires transformation for browser compatibility.

Build Pipeline

- **Development Flow**
    - Source code written in TypeScript/modern JavaScript
    - CSS preprocessing with Sass/Less
    - Asset management through build tools
    - Hot reloading for rapid development
- **Production Build**
    - Code transpilation and minification
    - Asset optimization and compression
    - Source maps for debugging
    - Cache control through file hashing

Build Tools: Overview

- Common Build Tasks
  - **Bundling**: combine multiple files into optimized packages
  - **Transpiling**: convert modern code to browser-compatible versions
  - **Optimization**: minify, compress, tree-shake (remove unused code)
  - **Asset Processing**: optimize images, compile styles, load fonts
- Package Managers
  - Manage dependencies and build scripts
  - Define project configuration and run development tasks
- Key Benefits
  - Use latest language features while maintaining compatibility
  - Improved loading performance, automated optimization, and processing
  - Development productivity and code quality (linting, type checking)

Package Managers

- Packages that manage dependencies and orchestrate development workflows.
- Common Features
  - Dependency management (install, update, remove)
  - Script execution (build, test, dev)
  - Lock file generation (versions control)
- Popular Options
  - **npm**: default Node.js package manager, https://www.npmjs.com
  - **yarn**: Facebook alternative to npm, https://yarnpkg.com

Bundling Tools

- ➔ Bundling tools combine source files into optimized bundles for deployment.
- ➔ **webpack**: comprehensive but more complex
  - ➔ https://webpack.js.org
- ➔ **esbuild**: fast Go-based bundler
  - ➔ https://esbuild.github.io
- ➔ **Rollup**: tree-shaking focused
  - ➔ https://rollupjs.org

Code Transpilation

- Code transpilation converts JavaScript/TypeScript into browser-compatible code, e.g.:
    - Modern JS (ES2024) → Compatible JS (ES5)
    - TypeScript → JavaScript
    - JSX (XML-like syntax used in React) → JavaScript
- **Babel**: popular JavaScript transpiler
    - https://babeljs.io
- **SWC**: modern Rust-based alternative
    - https://swc.rs
- **TypeScript compiler** (tsc): native TS compilation
    - https://www.typescriptlang.org

Development Servers

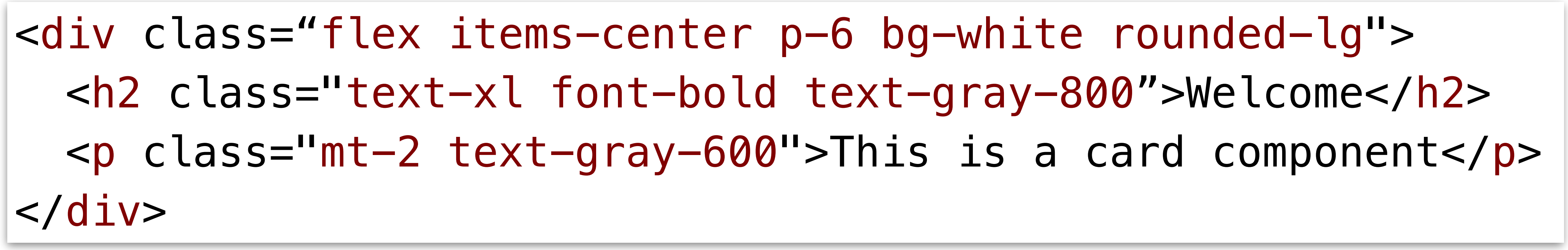
- Web servers optimized for frontend development, providing instant feedback and code updates without manual browser refresh.
- Key Features
  - Hot Module Replacement (HMR): update code without page reload
  - Fast refresh: preserve component state during updates
  - Source maps: debug your original code instead of minified bundles
  - Live reload: automatic browser refresh on changes
- Example Tools:
  - **Vite**: fast ES modules-based server, https://vitejs.dev
  - **webpack-dev-server**: webpack's development server

Asset Processing

- Asset processing optimizes and transforms web assets during development and build.
- Common Tasks
  - **Images**: compress, resize, convert formats
  - **Styles**: compile Sass/Less, autoprefix CSS
  - **Fonts**: subset, convert formats, optimize loading
  - **SVG**: optimize, sprite generation
- Example Tools:
  - **PostCSS**: transform and optimize CSS, https://postcss.org

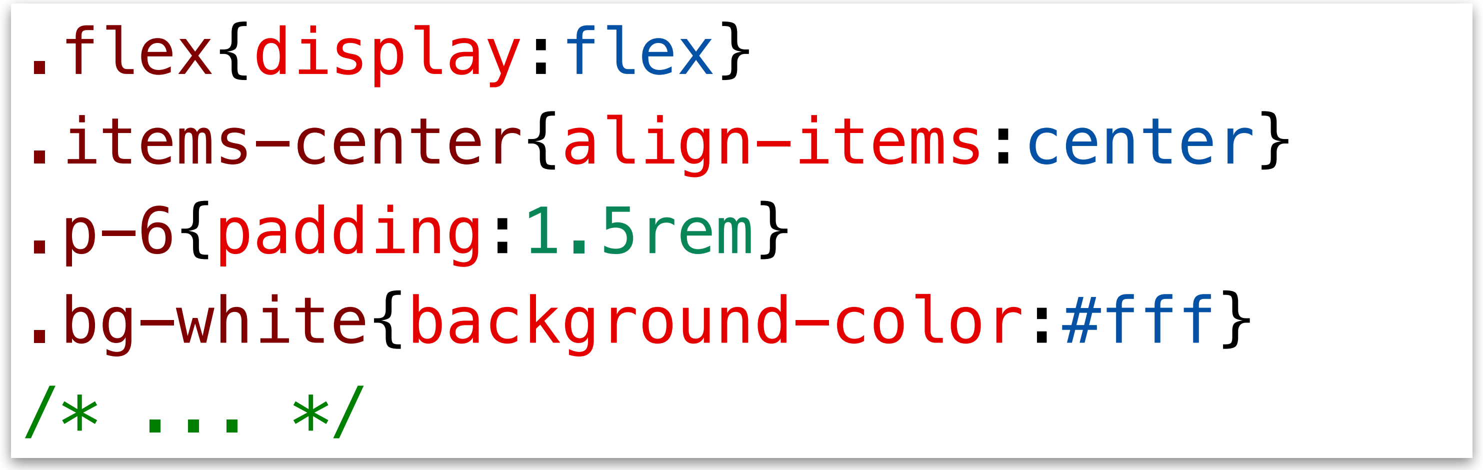
Build Example: Tailwind

- ➔ Source File contains utility class names.



```
<div class=“flex items-center p-6 bg-white rounded-lg">
  <h2 class="text-xl font-bold text-gray-800”>Welcome</h2>
  <p class="mt-2 text-gray-600">This is a card component</p>
</div>
```

- ➔ **Build Process**:
  - ➔ PostCSS processes Tailwind directives
  - ➔ Tree shaking removes unused utilities
  - ➔ CSS is minified for production
  - ➔ Similar process for JavaScript modules
  - ➔ Final bundle only includes used code

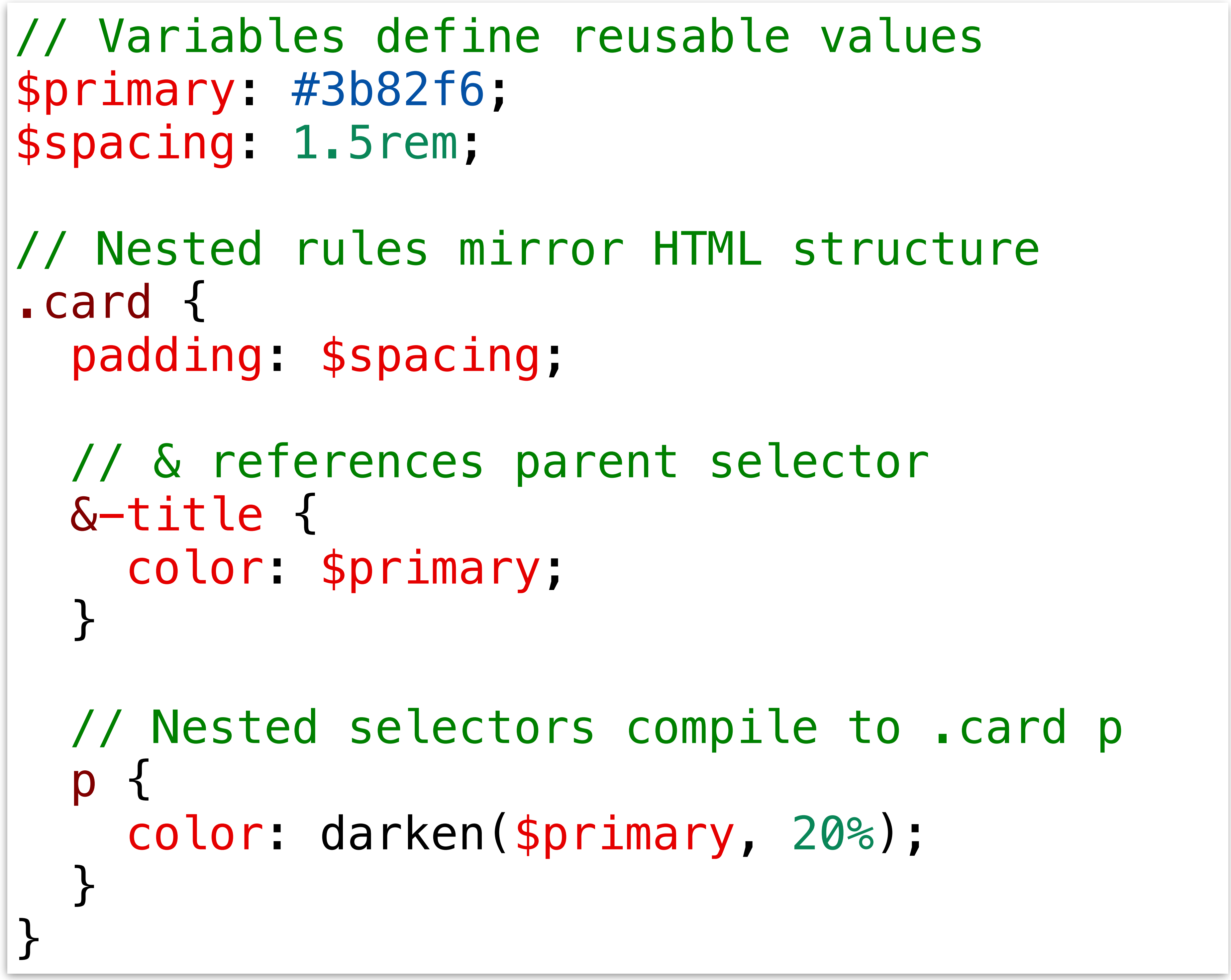


```
.flex{display:flex}
.items-center{align-items:center}
.p-6{padding:1.5rem}
.bg-white{background-color:#fff}
/* ... */
```

- ➔ **Production CSS** contains only the necessary utility classes in minified form.

Example: CSS Preprocessor

- ➔ Source File uses preprocessing features. E.g., Sass https://sass-lang.com
- ➔ **Build Process**:
  - ➔ Preprocessor compiles to standard CSS
  - ➔ Variables and functions are resolved
  - ➔ Nested rules are flattened
  - ➔ Rules are minified and optimized
  - ➔ Development maintains source maps
- ➔ **Production CSS** contains standard CSS rules in compressed format.



```scss
// Variables define reusable values
$primary: #3b82f6;
$spacing: 1.5rem;

// Nested rules mirror HTML structure
.card {
  padding: $spacing;

  // & references parent selector
  &-title {
    color: $primary;
  }

  // Nested selectors compile to .card p
  p {
    color: darken($primary, 20%);
  }
}
```

Progressive Web Applications

Progressive Web Applications

- **Native Applications** provide features like offline access, push notifications, and device integration.
- **Web Applications** run on any device with a browser but traditionally lacked these capabilities.
- **Progressive Web Apps** bridge this gap by enhancing web apps with native-like features.
- Google Chrome team introduced PWA concept in 2015.
  - Original proposal: Chrome Dev Summit 2015
  - https://web.dev/explore/progressive-web-apps
- Browser support varies significantly.

PWA Capabilities

- ➔ **Native Features** include offline access, push notifications, and device integration.
- ➔ **Installation** allows apps to run outside the browser window.
- ➔ **Network Independence** works with poor or no connectivity.
- ➔ **Automatic Updates** deliver new versions without user intervention.
- ➔ **Secure Context** requires HTTPS for enhanced security.

PWA Core Technologies

- ➔ **Service Workers** acts as a network proxy between browser, network, and cache.
- ➔ **Web App Manifest** enables "Add to Home Screen" with icons and metadata.
- ➔ **HTTPS** required as a security prerequisite for PWA features.
- ➔ **Cache API** stores resources locally for offline access.
- ➔ **IndexedDB** provides client-side storage for application data.

PWA Architecture

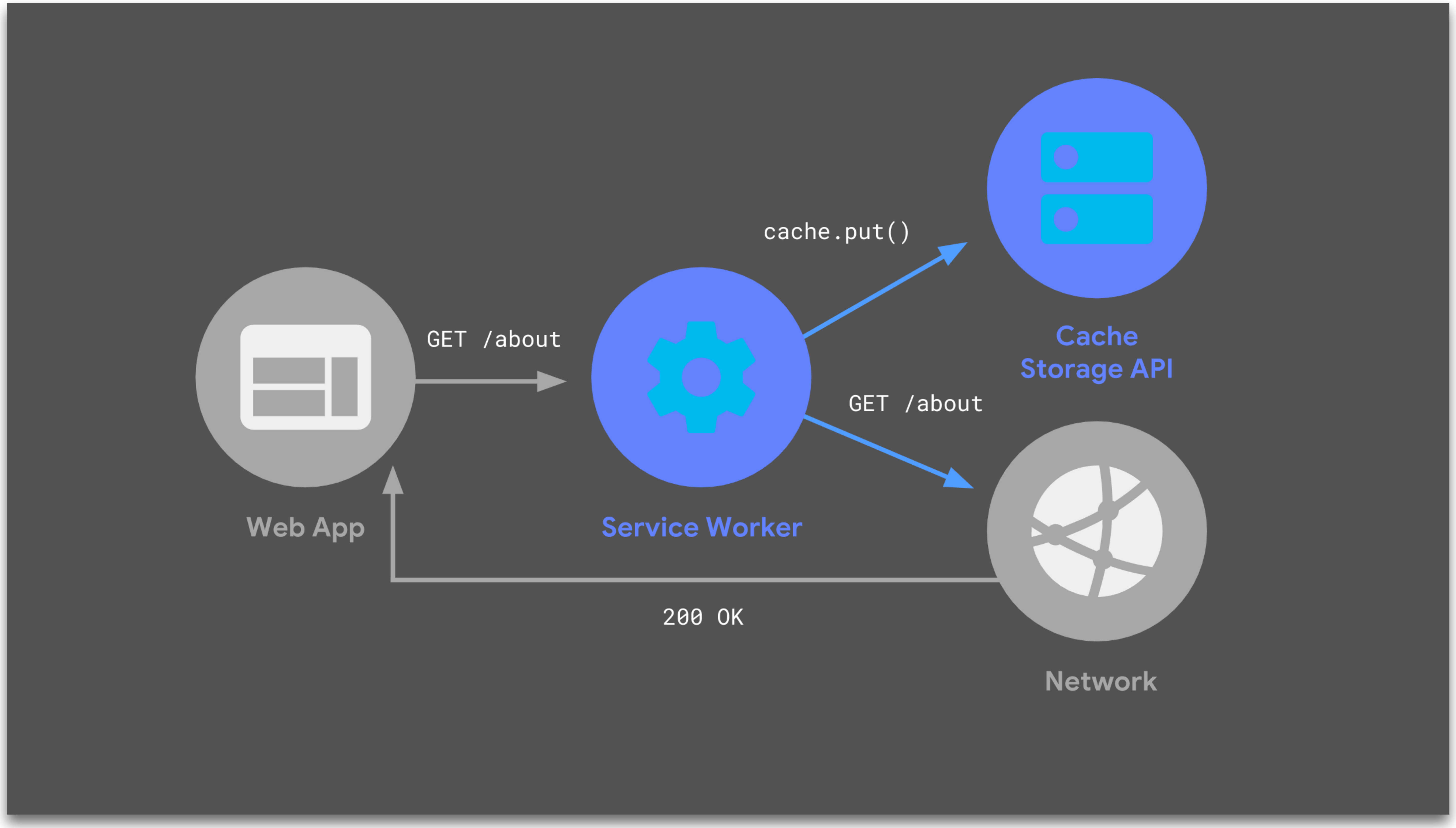


Image from Beyond SPAs: alternative architectures for your PWA (2018)
https://developers.google.com/web/updates/2018/05/beyond-spa

PWA Resources

- ➔ **Learning**
  - ➔ https://web.dev/learn/pwa
  - ➔ https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
- ➔ **Demos**
  - ➔ https://whatpwacando.today
  - ➔ Microsoft Progressive Web Apps Demos
    - ➔ https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/demo-pwas
    - ➔ https://github.com/mdn/pwa-examples

Web Development Resources

Web Development Resources (1)

- **Browser Compatibility**
  - caniuse.com — Feature support across browsers
  - MDN Browser Compatibility — Detailed compatibility tables
  - webhint.io — Browser compatibility testing tool
- **Documentation & Learning**
  - MDN Web Docs (developer.mozilla.org) — Reference web documentation
  - web.dev — Articles and resources on web development practices
  - javascript.info — JavaScript tutorials

Web Development Resources (2)

- **Development Tools**
  - CodePen (codepen.io) — Frontend code playground
  - JSFiddle (jsfiddle.net) — JavaScript testing environment
- **Performance and Testing**
  - PageSpeed Insights — Performance analysis
  - Lighthouse — Web app auditing tool
- **Code Quality**
  - ESLint — JavaScript code quality
  - validator.w3.org — HTML validation
  - CSS Validator (jigsaw.w3.org/css-validator) — CSS validation

Client-Side Web Development: Summary

- **Client-Side Web Development has evolved from static pages to complex applications**
  - Core technologies: HTML, CSS, and JavaScript remain fundamental
  - Modern frameworks and tools enhance development capabilities
- **Development Process involves multiple tools and workflows**
  - Build tools optimize code for production
  - Development tools support efficient workflows
- **Success in web development requires**
  - Understanding core technologies
  - Mastering development tools