Introduction to Web Development

Outline

How the Web Works

HTML: Structure and Semantics

CSS: Styling and Layout

JavaScript: Interactivity

Tools and Best Practices

How the Web Works

Client-Server Model

- The web operates on a **client-server architecture**.
- **Client:** Browser requesting content.
- **Server:** Delivers HTML, CSS, JS, images, or data.
- Communication occurs via **HTTP/HTTPS**, a request-response protocol.
- Modern websites often use **APIs** to fetch dynamic data in JSON format.

HTTP Requests and Responses

- **HTTP methods:** GET (retrieve), POST (submit), PUT (update), DELETE (remove).
- **Response codes:** 200 OK, 301 Redirect, 404 Not Found, 500 Internal Server Error.
- **Headers:** metadata about the request/response (content-type, cache-control).
- **Body:** Contains the actual content or data being sent.

DNS and URLs

- Domain Name System (DNS) converts domain names to IP addresses.
- URL Structure:
  `protocol://domain/path?query#fragment`
- Protocol: HTTP or HTTPS
- Path: location of resource on the server
- Query: parameters passed to server (e.g., `?id=123`)
- Fragment: navigates to a section of the page

Rendering a Web Page

1. Browser parses HTML $\rightarrow$ creates the **DOM (Document Object Model)**.
2. CSS is parsed $\rightarrow$ creates the **CSSOM**.
3. DOM + CSSOM $\rightarrow$ **Render Tree**.
4. Layout painting $\rightarrow$ pixels on the screen.
5. JavaScript can dynamically modify DOM CSSOM.

HTML: Structure and Semantics

What is HTML?

- HTML: **HyperText Markup Language** — defines structure of web pages.
- Uses **tags/elements** to organize content.
- HTML5 introduced **semantic elements** for better structure and accessibility: `<header>`, `<nav>`, `<main>`, `<section>`, `<article>`, `<footer>`.
- Browsers read HTML to build the DOM tree.

Basic HTML Document

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
      initial-scale=1.0">
  <title>My First Page</title>
</head>
<body>
  <h1>Hello World!</h1>
  <p>This is a simple HTML page.</p>
</body>
</html>
```

Common HTML Elements

- **Headings:** <h1>–<h6>
- **Paragraphs:** <p>
- **Links:** <a href="">
- **Lists:** , <ol>,
- **Images:** <img src="" alt="">
- **Tables:** <table>, <tr>, <td>, <th>
- **Forms:** <form>, <input>, <button>, <textarea>, <select>
- **Semantic tags:** <section>, <article>, <header>, <footer>, <main>, <nav>

Attributes in HTML

- Attributes provide extra info about elements: `id`, `class`, `src`, `alt`, `href`, `type`, `name`, `value`, `placeholder`, `required`.
- Example: <img src="logo.png" alt="Logo" width="200">
- Used for styling (CSS selectors) or scripting (JS).

HTML Forms

```
<form action="/submit" method="POST">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username"
      required>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email">

  <button type="submit">Submit</button>
</form>
```

Form Inputs and Validation

- `type="text"`, `"email"`, `"password"`, `"number"`, `"url"`
- HTML5 validation: `required`, `pattern`, `min`, `max`
- Labels improve accessibility (`for` attribute)
- Fieldsets legends group related inputs

CSS: Styling and Layout

What is CSS?

- Cascading Style Sheets define the presentation layer.
- Controls color, font, spacing, layout, and animation.
- "Cascading" = multiple rules resolved by specificity order.
- Inline > internal > external > browser default.

CSS Syntax Example

```
selector {
  property: value;
}
p {
  color: blue;
  font-size: 16px;
  line-height: 1.5;
}
```

Adding CSS to HTML

- External: <link rel="stylesheet" href="style.css">
- Internal: <style> in <head>
- Inline: style="property: value;"

Selectors and Specificity

```
/* Element */
p { color: red; }

/* Class */
.box { padding: 10px; }

/* ID */
#main { margin: 20px; }

/* Descendant */
nav ul li { list-style: none; }

/* Pseudo-class */
a:hover { color: green; }
```

Box Model and Layout

- Every element = content + padding + border + margin.
- Control width height via CSS.
- Layout techniques:
  - Flexbox: flexible 1D layout
  - Grid: 2D layout
  - Float (legacy)

Flexbox Example

```
.container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

Responsive Design

```
/* Media Query Example */
@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
}
```

CSS Best Practices

- Use external stylesheets
- Keep class names meaningful (BEM convention)
- Minimize redundancy
- Use variables (–primary-color) for maintainability

JavaScript: Interactivity

What is JavaScript?

- JS = programming language for web interactivity.
- Runs in the browser (client-side) or on server (Node.js).
- Manipulate DOM, handle events, fetch data (AJAX/Fetch API).
- Key features: variables, functions, objects, arrays, events.

Basic JavaScript Example

```
let name = "Alice";
function greet(person) {
  console.log("Hello, " + person + "!");
}
greet(name);
```

DOM Manipulation

```
// Select element
const heading = document.querySelector('h1');
// Change content
heading.textContent = 'Welcome to My Site!';
// Change style
heading.style.color = 'blue';
```

Event Handling

```
const button = document.querySelector('button');
button.addEventListener('click', () => {
  alert('Button clicked!');
});
```

Fetch API Example

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

JS Best Practices

- Avoid global variables
- Keep functions small and reusable
- Use let/const instead of var
- Comment your code

Tools and Best Practices

Text Editors and IDEs

- VS Code, Sublime Text, Atom, WebStorm
- Features: syntax highlighting, autocomplete, debugging, extensions
- Terminal integration for Git, npm, and deployment

Version Control with Git

- Track changes, collaborate with others
- Key commands: git init, add, commit, push, pull, branch, merge
- Platforms: GitHub, GitLab, Bitbucket

Accessibility and SEO

- Semantic HTML and ARIA roles improve accessibility
- Alt text for images, labels for forms
- SEO: descriptive titles, meta descriptions, headings, fast load times

Performance Optimization

- Minify CSS/JS, compress images
- Reduce HTTP requests using bundling
- Lazy-load assets
- Use browser caching and CDN

Deployment Options

- Free: GitHub Pages, Netlify, Vercel
- Paid: AWS, DigitalOcean, Heroku
- CI/CD pipelines for automatic deployment

Next Steps for Learning Web Dev

- Build projects (portfolio, blog, e-commerce)
- Learn frameworks/libraries (React, Vue, Angular)
- Learn backend: Node.js, Express, databases
- Keep practicing and follow best practices

Thank You!

Questions?