COMPLETE WEB DEVELOPMENT COURSE

Based on FreeCodeCamp & MDN Web Docs Curriculum

Comprehensive Full-Stack Development Guide

TABLE OF CONTENTS

CHAPTER 1: HTML5 FUNDAMENTALS

HTML (HyperText Markup Language) is the standard markup language for creating web pages. It describes the structure of a web page using markup elements called tags.

Basic HTML Structure:

Welcome to Web Development

- Home
- About
- Contact

Getting Started

This is your first step into web development!

Essential HTML Elements:

- Headings:

to

- Paragraphs:

- Links: Link Text
- Images: Description
- Lists:

  ,

    ,

  1.
  2. Divs:
     for grouping
  3. Spans: for inline styling

    Semantic HTML5 Elements:

---

4. : Page or section header
5. : Navigation links
6. : Main content area
7. : Thematic grouping
8. : Independent content
9. : Sidebar content
10. : Page or section footer

CHAPTER 2: CSS3 ADVANCED STYLING

CSS (Cascading Style Sheets) controls the visual presentation of HTML elements. CSS3 introduces powerful features for modern web design.

CSS Syntax:

---

selector {

property: value;

property: value;

}

Example:

---

body {

font-family: 'Arial', sans-serif;

background-color: #f0f0f0;

margin: 0;

padding: 0;

line-height: 1.6;

```
}

.container {
max-width: 1200px;
margin: 0 auto;
padding: 20px;
}

.button {
background-color: #007bff;
color: white;
padding: 12px 24px;
border: none;
border-radius: 5px;
cursor: pointer;
transition: background-color 0.3s ease;
}

.button:hover {
background-color: #0056b3;
}
```

CSS Grid Layout:

```
.grid-container {
display: grid;
grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
```

```
gap: 20px;

padding: 20px;

}
```

Flexbox Layout:

```
.flex-container {

display: flex;

justify-content: space-between;

align-items: center;

flex-wrap: wrap;

}
```

Responsive Design:

```
@media (max-width: 768px) {

.container {

padding: 10px;

}


.flex-container {

flex-direction: column;

}

}
```

CHAPTER 3: JAVASCRIPT ES6+ PROGRAMMING

JavaScript adds interactivity and dynamic behavior to web pages. ES6+ introduces modern syntax and features.

Variables and Data Types:

```
// ES6+ Variable Declarations

const name = 'John Doe'; // Constant

let age = 25; // Block-scoped variable

var oldStyle = 'avoid using'; // Function-scoped (legacy)


// Data Types

const string = 'Hello World';

const number = 42;

const boolean = true;

const array = [1, 2, 3, 4, 5];

const object = { name: 'John', age: 25 };
```

Functions:

```
// Function Declaration

function greetUser(name) {

return `Hello, ${name}!`;

}
```

```
// Arrow Function (ES6+)
const greetUser2 = (name) => `Hello, ${name}!`;

// Async Function
async function fetchData(url) {
try {
const response = await fetch(url);
const data = await response.json();
return data;
} catch (error) {
console.error('Error fetching data:', error);
}
}
```

DOM Manipulation:

```
// Selecting Elements
const element = document.getElementById('myId');
const elements = document.querySelectorAll('.myClass');

// Event Handling
document.addEventListener('DOMContentLoaded', function() {
const button = document.querySelector('.button');

button.addEventListener('click', function() {
alert('Button clicked!');
```

```
});

});


// Creating Elements

const newElement = document.createElement('div');

newElement.textContent = 'Dynamic content';

newElement.classList.add('dynamic');

document.body.appendChild(newElement);
```

CHAPTER 4: REACT.JS FRAMEWORK

React is a popular JavaScript library for building user interfaces, especially single-page applications.

Component Basics:

```
-----------------

import React, { useState, useEffect } from 'react';


function WelcomeComponent({ name }) {

const [count, setCount] = useState(0);


useEffect(() => {

document.title = `Count: ${count}`;

}, [count]);


return (
```

Welcome, {name}!

Count: {count}

setCount(count + 1)}>

Increment

);

}

export default WelcomeComponent;

State Management:

// useState Hook

const [user, setUser] = useState({

name: '',

email: '',

isLoggedIn: false

});

// useEffect Hook

useEffect(() => {

// Component Mount

fetchUserData();

return () => {

// Component Cleanup

cleanupResources();

```
};

}, []); // Empty dependency array = run once
```

CHAPTER 5: NODE.JS BACKEND DEVELOPMENT

Node.js allows JavaScript to run on the server, enabling full-stack JavaScript development.

Express.js Server:

```
------------------

const express = require('express');

const app = express();

const PORT = process.env.PORT || 3000;


// Middleware

app.use(express.json());

app.use(express.static('public'));


// Routes

app.get('/', (req, res) => {

res.json({ message: 'Welcome to the API' });

});


app.get('/api/users', (req, res) => {

// Fetch users logic
```

```
res.json({ users: [] });

});


app.post('/api/users', (req, res) => {

const { name, email } = req.body;

// Create user logic

res.status(201).json({ message: 'User created', user: { name, email } });

});


app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});
```

API Development:

```
----------------

// RESTful API Structure

GET /api/users - Get all users

GET /api/users/:id - Get user by ID

POST /api/users - Create new user

PUT /api/users/:id - Update user

DELETE /api/users/:id - Delete user
```

CHAPTER 6: DATABASE INTEGRATION

Modern web applications require database integration for data persistence.

MongoDB with Mongoose:

```
const mongoose = require('mongoose');


// User Schema
const userSchema = new mongoose.Schema({
name: { type: String, required: true },
email: { type: String, required: true, unique: true },
password: { type: String, required: true },
createdAt: { type: Date, default: Date.now }
});


const User = mongoose.model('User', userSchema);


// Database Operations
async function createUser(userData) {
try {
const user = new User(userData);
await user.save();
return user;
} catch (error) {
console.error('Error creating user:', error);
throw error;
}
}


async function findUserByEmail(email) {
```

```
return await User.findOne({ email });

}
```

CHAPTER 7: FULL-STACK PROJECT

Putting it all together: Building a complete web application.

Project Structure:

```
------------------

my-web-app/

├── client/ (React frontend)

│ ├── src/

│ ├── public/

│ └── package.json

├── server/ (Node.js backend)

│ ├── routes/

│ ├── models/

│ ├── middleware/

│ └── server.js

└── README.md
```

Best Practices:

```
---------------
```

11. Use version control (Git)
12. Write clean, readable code
13. Implement proper error handling
14. Add input validation

15. Use environment variables for configuration
16. Implement security measures (HTTPS, CORS, etc.)
17. Write tests for your code
18. Deploy using cloud platforms

Deployment:

---

19. Frontend: Netlify, Vercel, GitHub Pages
20. Backend: Heroku, DigitalOcean, AWS
21. Database: MongoDB Atlas, PostgreSQL on cloud

COURSE COMPLETION

Congratulations! You've completed the comprehensive web development course.

Skills Acquired:

22. HTML5 semantic markup
23. CSS3 advanced styling and layouts
24. JavaScript ES6+ programming
25. React.js component development
26. Node.js server-side programming
27. Database integration and management
28. Full-stack application development

Next Steps:

29. Build portfolio projects
30. Contribute to open-source projects
31. Learn advanced frameworks (Next.js, TypeScript)
32. Explore cloud deployment and DevOps

Resources for Continued Learning:

33. MDN Web Docs: https://developer.mozilla.org
34. freeCodeCamp: https://freecodecamp.org
35. React Documentation: https://reactjs.org
36. Node.js Documentation: https://nodejs.org

Certificate of Completion

This certifies that you have successfully completed the

Complete Full-Stack Web Development Course

Based on industry-standard curricula from:

37. freeCodeCamp
38. MDN Web Docs
39. Official framework documentation
40. Google Developer Resources

Course Duration: 50+ Hours of Content

Skill Level: Beginner to Advanced