WEB DESIGN FULL COURSE

PRE-REQUISITE

Important

Click here to go to the complete Web Design Course Outline Explorer Website.

🌐 Week 1: INTRODUCTION & SETUP

📘 Day 1: Introduction to Web Design

✨ Key Concepts:

How the web works

`Request-Response model` **(browser ⇄ server)**

1. Client Sends a Request

> The client is usually a **web browser** (e.g., Chrome, Firefox).
> When you type a URL (e.g., `https://google.com`) or click a link, the browser sends a **request** to the server.

> This request includes:
>
> 1. The `type of request` (usually **GET** or **POST**).
> 2. The `URL` or `path` to the resource.
> 3. `Headers` (like browser info, language, etc.).
> 4. `Optional data` (in **POST** requests, such as form data).

2. Server Processes the Request

> The `server` is *a computer running web server software* (like Apache, Nginx, or Node.js).
> It receives the request, processes it, and prepares a response.

> The server may:
>
> 1. Fetch a file (e.g., HTML, CSS, JavaScript).
> 2. Run a script (e.g., PHP, Python, Java).

> 3. Query a database and create dynamic content.

3. Server Sends a Response

> The server sends back an **HTTP response**.

> The response contains:
>
> 1. A **status code** (e.g., 200 OK, 404 Not Found).
> 2. Headers (e.g., content type, caching info).
> 3. A **body** (usually HTML, JSON, or other content).

4. Client Receives the Response

> The browser reads the response.
> If it's an HTML page, it displays it to the user.
> If it’s data (e.g., JSON), it might be handled by JavaScript.

🧠 Practical Example

You visit: `https://www.youtube.com/home`

1. **Client (Browser) → Server**

```
GET /about HTTP/1.1 Host: youtube.com
```

2. **Server → Client**

```
HTTP/1.1 200 OK Content-Type: text/html
<html>
   <head><title>About</title></head>
   <body>
      <h1>About Us</h1>
   </body>
</html>
```

🔍 Key Concepts

Concept	Description
HTTP	Hypertext Transfer Protocol; the foundation of communication on the Web.
Request	A message sent by the client to ask for a resource.
Response	A message sent by the server with the result of the request.
Status Code	Tells whether the request was successful (e.g., 200, 404, 500).
GET	Request type to **retrieve** data.
POST	Request type to **send** data to the server (e.g., form submission).

📦 Analogy: Ordering Food

Web Component	Restaurant Analogy
Browser (Client)	Customer placing an order
Server	Chef in the kitchen
Request	The food order
Response	The prepared dish
URL	The menu item name
HTTP	The waiter's notepad and communication

🧠 Summary

Note

A web page is a file served to your browser. The browser (client) makes a request to a server (e.g., via a URL), which returns HTML, CSS, and JS files.
The web runs on a **request-response model**.
Clients send requests for content, and **servers** respond with the data.
This process happens billions of times a day to make websites load, APIs work, and the Internet function.

Important

Frontend Developer: Builds what users see (Interface)
Backend Developer: Handles server-side logic (Databases)
Full-Stack Developer: Does both (Front and Backend development)

🛠️ Day 2: Digital Notetaking with Obsidian

✨ Key Concepts:

> Use **Markdown** for writing notes (`.md` files)
> Organize your files in folders
> Use **internal links** (`[[page-name]]`) to connect thoughts

Tip

Keep a **Web Design Vault** in Obsidian to log:

1. Project ideas
2. Lessons learned
3. Code snippets

💻 Day 3: SETTING UP THE DEVELOPEMENT ENVIRONMENT

✅ Tools:

> 1. **Code Editor**: Notepad, VS Code, Sublime Text etc.,
> 2. **Browser**: Chrome (with DevTools)
> 3. **Live Server Extension**: For live preview

📝 Using **Notepad** (Simple and Built-in)

✅ Steps:

1. **Open Notepad**

> Click **Start Menu** → search **Notepad** → Open it.

2. **Write Basic HTML Code**

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

3. **Save the File**

> Name it like: `index.html`

> **Important**
>
> 1. In the **"Save as type"** box, choose **All Files**.
> 2. Make sure the file ends with `.html`

4. **Open in Browser**

> Open `File Explorer`, find your file then double-click it.
> It will open in your default browser (Chrome, Edge, etc.)

> **Note**
>
> **Notepad is good for quick practice**, but has no extra features like highlighting, autocomplete, or live preview.

💻 Using Visual Studio Code (VS Code)

> **VS Code** is a free, professional code editor by Microsoft — recommended for serious web development.

✅ Step-by-step Setup

1. Download & Install VS Code

> Go to https://code.visualstudio.com
> Download for Windows/Mac/Linux and install it.

2. Create a Project Folder

> Create a new folder on your computer, e.g., `MyWebsite` for instance inside the Desktop.

3. Open the Folder in VS Code

> Open VS Code.
> You can also open VS Code using the Terminal:

```
. code
```

> Click **File** → **Open Folder**, then select your folder.

4. Create a New HTML File

> Click **File** → **New File**
> Save it as `index.html`

5. Write Your Code

> Inside the `index.html` file, add the code below:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My VS Code Page</title>
  </head>
  <body>
    <h1>Welcome to my website!</h1>
  </body>
</html>
```

6. Install Live Server (Optional but Recommended)

> Go to the **Extensions** tab on the left (or press `Ctrl+Shift+X`)
> Search for **Live Server** and click **Install**

7. Run Your Page in the Browser

> Right-click inside your HTML file → Click **"Open with Live Server"**
> Your page will open in the browser and auto-update when you save.

🔥 **Important**

Turn on `Auto save` in VS Code to ensure you don't have to keep saving every time you make changes., thus making your development seamless, fast and effective.

⚖️ Notepad vs VS Code:

Feature	Notepad	VS Code
Syntax highlighting	❌ No	✅ Yes
Code suggestions	❌ No	✅ Yes
Live preview	❌ No	✅ With Live Server extension
Best for	Beginners, quick edit	All levels of development
Project management	❌ Basic only	✅ Strong project tools
Weight/Computer RAM usage	✅ Lightweight/Low usage	❌ Heavyweight/high usage

---

🧠 Summary

🔥 **Tip**

1. **Notepad**: Easy for beginners to get started. Just save as `.html` and open in a browser.
2. **VS Code**: More powerful. Great for long-term learning and real-world development.
3. You can start with **Notepad**, then switch to **VS Code** as you become more comfortable.

🧰 Folder Structure:

```
/web-design-course/
     index.html
   /css/
      styles.css
   /js/
      script.js
```

🧾 Day 4: HTML Basics

🧱 HTML Skeleton

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

💡 Explanation:

> `<!DOCTYPE html>` : Declares HTML5
> `<head>` : Metadata, styles, scripts
> `<body>` : What the user sees (headings, paragraphs, images, links)

🧑‍💻 Day 5: Simple Web page Project — Bio Page

Objective:

> Create a personal bio page using plain HTML

Example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My Bio</title>
  </head>
  <body>
    <h1>Jane Doe</h1>
    <p>I’m an aspiring web developer learning HTML and CSS!</p>
    <ul>
      <li>📍 Location: Nairobi</li>
      <li>💼 Interest: Web Design</li>
      <li>🎓 Goal: Become a Frontend Developer</li>
    </ul>
  </body>
</html>
```

🧠 Learning Outcomes:

> Practice HTML tags
> Understand file saving (`.html`)
> View output in the browser

✅ Week 1 Summary:

Topic	Outcome
Web Basics	Understand how web pages work
Obsidian	Organize notes efficiently
Environment Setup	Run code locally
HTML	Write structured content
Mini Project	Apply your knowledge

🧩 Week 2: DEEP DIVE INTO HTML & INTRO TO CSS

📄 Day 1: HTML Continued — Multimedia & Navigation

✅ Key Concepts:

> Embedding **images**, **audio**, and **video**
> Adding **navigation links** with `<a>`

🧠 Example:

```
<body class="p-6">
  <h1 class="text-2xl font-bold mb-4">My Portfolio</h1>

  <img src="profile.jpg" alt="Profile Photo" class="w-32 rounded-full mb-4">

  <video controls class="w-full mb-4">
    <source src="intro.mp4" type="video/mp4">
    Your browser does not support the video tag.
  </video>

  <nav class="mb-4">
    <a href="index.html" class="text-blue-500 hover:underline mr-4">Home</a>
    <a href="about.html" class="text-blue-500 hover:underline">About</a>
  </nav>
</body>
```

🧠 Explanation:

> `img` displays pictures
> `video` and `audio` play media
> `a href` adds hyperlinks (internal or external)

📥 Day 2: Forms & Tables

✅ Key Concepts:

1. **Forms** collect input
2. **Tables** organize data

🧠 Example:

```
<form class="space-y-4 p-4 border rounded">
  <input type="text" placeholder="Your Name" class="w-full border p-2 rounded"
/>    <input type="email" placeholder="Your Email" class="w-full border p-2
rounded" />
  <button type="submit" class="bg-blue-600 text-white px-4 py-2
```

```
rounded">Submit
  </button>
</form>

<table class="table-auto border-collapse mt-6 w-full">
  <thead>
    <tr class="bg-gray-200">
      <th class="border px-4 py-2">Name</th>
      <th class="border px-4 py-2">Email</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td class="border px-4 py-2">Jane</td>
      <td class="border px-4 py-2">jane@example.com</td>
    </tr>
  </tbody>
</table>
```

🎨 Day 3: CSS Fundamentals

✅ Key Concepts:

> Styling with **external CSS** (for instance from a website)
> Using **Tailwind CSS** for utility-first design

💡 Tailwind Setup:

Add Tailwind CDN (Content Delivery Network) to `<head>` :

```
<script src="https://cdn.tailwindcss.com"></script>
```

🧠 Example:

```
<p class="text-lg text-gray-700 font-semibold">
  Welcome to my styled webpage!
</p>
```

✍️ Day 4: CSS Selectors & Text Styling

✅ Key Concepts:

> Tailwind handles most styling via **utility classes**
> Text styling: `text-xl`, `font-bold`, `text-blue-500`, `underline`

🧠 Example:

```
<h2 class="text-2xl font-bold text-center underline text-indigo-600">
  This is a Heading
</h2>
<p class="text-sm text-gray-600 italic mt-2">
  Styled with Tailwind CSS
</p>
```

📦 Day 5: Box Model

✅ Key Concepts:

> Each HTML element is a **box**: content + padding + border + margin
> Tailwind classes:
> `p-*`: padding
> `m-*`: margin
> `border`: adds border

> The **CSS Box Model** is a fundamental concept in web design and layout. It describes how every HTML element on a webpage is structured and how space is calculated around it.

> When you create an element like a `<div>`, `<p>`, or `<h1>`, the browser treats it as a **rectangular box**, and that box has **4 parts**:

🧱 The 4 Parts of the Box Model

```
+-------------------------------+
|          Margin (outer)       |
|   +-------------------------+ |
|   |      Border             | |
|   |   +-----------------+   | |
|   |   |    Padding         |   |  |
|   |   |   +--------------+   |   |  |
|   |   |   |   Content     |   |   |  |
|   |   |   +--------------+   |   |  |
|   |   +--------------------+   |  |
|   +--------------------------+  |
+--------------------------------+
```

1. Content

> This is *where your text, image, or other content lives*.
> You set its size using properties like `width` and `height`.

2. Padding

> *Space between the content and the border*.
> Makes content look less cramped.
> Example: `padding: 20px;`

3. Border

> *A line that wraps around the padding (and content)*.
> You can set its thickness, style, and color.
> Example: `border: 2px solid black;`

4. Margin

> The space **outside the border**.
> Creates distance between elements.
> Example: `margin: 15px;`

🧮 Box Model Formula

> To calculate the **total space** an element takes:

```
Total Width = margin + border + padding + content width Total
Height = margin + border + padding + content height
```

For example:

```
div {   width: 200px;   padding: 10px;   border: 5px solid black;   margin:
20px; }
```

- The total width is:

```
200 (content) + 10*2 (padding) + 5*2 (border) + 20*2 (margin) = 270px
```

💡 Tip: `box-sizing`

> By default, the `width` and `height` apply only to the **content**.
> If you want the total width to include **padding and border**, use:

```
* {   box-sizing: border-box; }
```

This makes layouts easier to manage.

🧠 Example:

```
<div class="p-6 m-4 border border-gray-300 rounded shadow">
  <h3 class="text-xl font-semibold">Box Model Example</h3>
  <p>This box has padding, margin, and a border.</p>
</div>
```

✅ Week 2 Summary:

Topic	Skills Developed
Multimedia	Embed videos, images, and audio
Forms	Create input fields, buttons
Tables	Display structured data
CSS	Start using Tailwind utility classes
Box Model	Understand element spacing and layout

📐 Week 3: LAYOUT AND RESPONSIVENESS

📊 Day 1: Display & Positioning

✅ Key Concepts:

> `display` controls layout behavior (`block` , `inline` , `flex` , etc.)
> Tailwind classes:

```
`block`, `inline-block`, `flex`, `grid` `relative`, `absolute`, `fixed`, `z-
*`
```

🧠 Example:

```
<div class="relative bg-gray-100 p-6">
  <p class="text-lg">This is normal text.</p>
  <span class="absolute top-2 right-2 text-sm text-red-600">Absolute!</span>
</div>
```

💡 Explanation:

> `relative` acts as a reference for absolutely positioned children
> `absolute top-2 right-2` positions the span near the top right

📏 Day 2: Flexbox Layout

✅ Key Concepts:

> Flexbox *allows responsive, one-dimensional layouts*

> Tailwind classes:
> `flex` , `flex-row` , `flex-col` `justify-*` , `items-*`

🧠 Example:

```
<div class="flex justify-between items-center bg-gray-100 p-4">
  <div class="text-lg font-bold">Logo</div>
  <nav class="flex space-x-4">
    <a href="#" class="text-blue-500 hover:underline">Home</a>
    <a href="#" class="text-blue-500 hover:underline">About</a>
  </nav>
</div>
```

💡 Explanation:

> `justify-between` : space between items
> `items-center` : vertically centers content

🧪 Day 3: Mini Layout Project (Card or Blog)

✅ Objective:

> Build a blog card component with image, title, text.

🧠 Example:

```
<div class="max-w-sm rounded overflow-hidden shadow-lg m-4">
  <img class="w-full" src="blog.jpg" alt="Blog image" />
  <div class="p-4">
    <h2 class="font-bold text-xl mb-2">My Blog Post</h2>
    <p class="text-gray-700 text-base">This is a simple blog card using
Tailwind
    </p>
  </div>
</div>
```

📱 Day 4: Media Queries with Tailwind

✅ Key Concepts:

> Tailwind uses **breakpoint prefixes** for responsive design:
> `sm:` , `md:` , `lg:` , `xl:` , `2xl:`

🧠 Example:

```
<div class="bg-blue-200 p-4 text-center sm:bg-green-200 md:bg-yellow-200
lg:bg-red-200">   Resize the screen to see the background color change!
</div>
```

💡 Explanation:

- `sm:` activates ≥640px

- `md:` ≥768px
- `lg:` ≥1024px, etc.

🧑‍💻 Day 5: Build Day — Responsive Bio/Profile Page

✅ Objective:

Build a fully responsive profile page using Tailwind utility classes.

🧠 Layout Example:

```
<div class="max-w-4xl mx-auto p-4 flex flex-col md:flex-row items-center">
  <img src="me.jpg" class="w-32 h-32 rounded-full mb-4 md:mb-0 md:mr-6"
alt="My Photo" />
    <div>
      <h1 class="text-2xl font-bold mb-2">Jane Doe</h1>
      <p class="text-gray-600">Frontend Developer passionate about responsive
design and user experience.</p>
    </div>
</div>
```

💡 Explanation:

- `flex-col` on small screens, `flex-row` on medium and up
- `max-w-4xl mx-auto`: centered content with max width

✅ Summary for Week 3:

Topic	Skills Developed
Display & Positioning	Use `flex`, `block`, `absolute`, `relative`
Flexbox	Create responsive navigation and cards
Project	Hands-on layout with real components
Media Queries	Adapt design to mobile/tablet/desktop
Responsive Design	Use Tailwind to adjust layout based on screen size

🧱 Week 4: Advanced HTML/CSS + Personal Portfolio

🧩 Day 1: Semantic HTML & Accessibility

✅ Key Concepts:

- **Semantic HTML**: Tags that describe their meaning (e.g., `<header>`, `<main>`, `<footer>`)
- **Accessibility (a11y)**: Making websites usable for all (e.g., screen readers)

🧠 Example:

```
<header class="bg-blue-600 text-white p-4">
   <h1 class="text-2xl font-bold">Jane Doe Portfolio</h1>
</header>
<main class="p-4">
    <section aria-labelledby="about">
        <h2 id="about" class="text-xl font-semibold mb-2">About Me</h2>
        <p>I’m a frontend developer with a focus on responsive and accessible
design.</p>
    </section>
</main>
<footer class="bg-gray-800 text-white p-4 text-center">   &copy; 2025
EutopianCJ </footer>
```

💡 Explanation:

- Use ARIA attributes (`aria-*`) to enhance accessibility
- Screen readers benefit from clear structure and landmarks

🎨 Day 2: CSS Pseudo-classes & Transitions

✅ Key Concepts:

- **Pseudo-classes**: `hover:`, `focus:`, `active:`
- **Transitions**: Smooth changes between states

🧠 Example:

```
<a href="#" class="text-blue-600 hover:text-blue-800 transition-colors
duration-300">   Hover me
</a>
```

💡 Explanation:

- `hover:text-blue-800` : changes color on hover
- `transition-colors` : adds smooth color change effect

📐 Day 3: UI/UX Basics + Project Planning

✅ Key Concepts:

- **UI (User Interface)**: Visual layout (colors, fonts, spacing)
- **UX (User Experience)**: How easy/intuitive it is to use
- **Wireframes**: Sketch layouts before coding

💡 Tools:

- Pen & paper
- Figma for digital mockups

🧠 Example Planning:

- Sections: Hero, About, Projects, Contact
- Pages: `index.html` , `about.html` , `contact.html`

🏠 Day 4: Portfolio Homepage Layout

✅ Objective:

Create a homepage layout with Tailwind and semantic tags

🧠 Example:

```
<section class="bg-white text-center p-10">
   <h1 class="text-4xl font-bold mb-4">Hi, I'm Jane</h1>
   <p class="text-gray-600">I build responsive, accessible websites with
```

```
Tailwind CSS and JavaScript.</p>
   <a href="#contact" class="mt-6 inline-block bg-blue-600 text-white px-4 py-
2 rounded hover:bg-blue-700 transition">Contact Me
   </a>
</section>
```

📄 Day 5: Portfolio Subpages (About & Contact)

✅ Key Concepts:

- Use **navigation** to switch between pages
- Use **forms** on Contact page

🧠 Example Navigation:

```
<nav class="flex space-x-6 justify-center bg-gray-200 p-4">
  <a href="index.html" class="hover:underline text-blue-700">Home</a>
  <a href="about.html" class="hover:underline text-blue-700">About</a>
  <a href="contact.html" class="hover:underline text-blue-700">Contact</a>
</nav>
```

🧠 Example Contact Form:

```
<form class="max-w-md mx-auto mt-8 space-y-4">
   <input type="text" placeholder="Your Name" class="w-full border p-2
rounded" required />
   <input type="email" placeholder="Your Email" class="w-full border p-2
rounded" required />
   <textarea placeholder="Your Message" class="w-full border p-2 rounded"
rows="4"></textarea>
   <button class="bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700
transition">Send</button>
</form>
```

✅ Summary for Week 4:

Topic	Skills Developed
Semantic HTML	Use `<main>`, `<section>`, `<footer>` correctly
Accessibility	Add ARIA labels and improve screen-reader support
UI/UX	Design wireframes and understand structure
Transitions	Animate hover and focus states
Portfolio	Start building and linking pages

🧠 Week 5: JavaScript Basics

🧪 Day 1: Introduction to JavaScript

✅ Key Concepts:

- **JavaScript** runs in the browser to make pages interactive
- It's a **programming language** with variables, functions, and logic

🧠 Example:

```
<script>
  const greeting = "Hello, JavaScript!";
  console.log(greeting); // Prints to the browser console
</script>
```

💡 Explanation:

- JavaScript can be added in a `<script>` tag
- You can also place scripts in a separate `.js` file

```
<script src="script.js"></script>
```

🧮 Day 2: Functions & Events

✅ Key Concepts:

- **Functions** = reusable blocks of code
- **Events** = user actions like `click`, `hover`, `input`

🧠 Example (Inline & External):

```
<button onclick="sayHi()" class="bg-blue-600 text-white px-4 py-2
rounded">Click Me
</button>

<script>
  function sayHi() {
    alert("Hello from JavaScript!");
  }
</script>
```

Or using **addEventListener**:

```
document.querySelector('button').addEventListener('click', () => {
  alert("Hello from JS event listener!");
});
```

🌐 Day 3: DOM Manipulation

✅ Key Concepts:

- **DOM** = Document Object Model (HTML as objects)
- You can **select** and **modify** elements dynamically

🧠 Example:

```
<h1 id="title" class="text-2xl">Original Title</h1>
<button id="changeBtn" class="mt-4 bg-green-600 text-white px-4 py-2">Change
Title
</button>  <script>
document.getElementById("changeBtn").addEventListener("click", () => {
document.getElementById("title").textContent = "Updated Title!";   });
</script>
```

🧾 Day 4: Form Validation

✅ Key Concepts:

- Prevent form submission if fields are empty or incorrect
- Use `.value`, `if` conditions, and `alert()`

🧠 Example:

```
<form id="contactForm" class="space-y-4">
    <input type="text" id="name" placeholder="Your Name" class="border p-2 w-
full rounded" />
    <input type="email" id="email" placeholder="Your Email" class="border p-2
w-full rounded" />
    <button type="submit" class="bg-blue-600 text-white px-4 py-2
rounded">Submit</button>
</form>

<script>
document.getElementById("contactForm").addEventListener("submit", function (e)
{     const name = document.getElementById("name").value.trim();
  const email = document.getElementById("email").value.trim();
      if (name === "" || email === "") {
         e.preventDefault();
         alert("Please fill in all fields.");
      }
    });
</script>
```

💡 Explanation:

- `.trim()` removes spaces
- `e.preventDefault()` stops form from submitting if validation fails

✏️ Day 5: Mini Project – Interactive Contact Form

✅ Objective:

Make a real, user-friendly form with:

- Input validation

- Visual feedback

🧠 Example with Feedback:

```
<p id="msg" class="text-sm text-red-600 mb-2 hidden"></p>
<form id="myForm" class="space-y-4">
    <input type="text" id="user" placeholder="Username" class="w-full border
p-2 rounded" />
    <button type="submit" class="bg-indigo-600 text-white px-4 py-2
rounded">Send
    </button>
</form>

<script>
    const form = document.getElementById("myForm");
    const msg = document.getElementById("msg");
    form.addEventListener("submit", (e) => {
       const user = document.getElementById("user").value;
           if (user.length < 3) {
              e.preventDefault();
              msg.textContent = "Username must be at least 3 characters.";
msg.classList.remove("hidden");
           } else {
              msg.textContent = "";
              msg.classList.add("hidden");
           }
    });
</script>
```

✅ Summary for Week 5:

Topic	Skills Developed
JavaScript Basics	Write functions, use variables, and log output
Events	Handle clicks and form submissions
DOM Manipulation	Dynamically change page content
Form Validation	Improve UX by checking inputs
Mini Project	Combine form, JS, and styling into a real feature

🚀 Week 6: Project Management & Deployment

📁 Day 1: Introduction to Git & GitHub

✅ Key Concepts:

- **Git**: Tracks changes in code (version control)
- **GitHub**: Cloud hosting for Git repositories

🔧 Common Git Commands:

```
git init                    # Start a new Git repo
git status                  # Check file changes
git add .                   # Stage all changes
git commit -m "message"     # Save snapshot
git remote add origin <repo-url> git push -u origin main    # Upload code
```

🧠 GitHub Flow Summary:

1. Make a change locally
2. Commit it with a message
3. Push to GitHub
4. Others can view or contribute

🌍 Day 2: Hosting with GitHub Pages

✅ Steps:

1. Push your project to a GitHub repository
2. Go to **Settings > Pages**
3. Select branch (e.g. `main`) and folder (`/root`)
4. GitHub will give you a public URL 🎉

🧠 Example:

```
<!-- index.html -->
```

```html
<h1 class="text-3xl font-bold text-center mt-10">Welcome to My Portfolio</h1>
```

💡 Tip:

- Your homepage **must** be named `index.html`

🖼️ Day 3: Image Optimization & File Formats

✅ Key Concepts:

- Use optimized formats like **WebP**, **JPEG (compressed)**, **SVG** for icons
- Reduce file size to **improve load speed**

🔧 Tools:

- tinypng.com
- squoosh.app

🧠 Tailwind Example:

```html
<img src="profile.webp" alt="Profile" class="w-32 h-32 rounded-full"
loading="lazy" />
```

💡 Best Practices:

- Use `loading="lazy"` for better performance
- Compress large images before uploading

🔍 Day 4: SEO Basics for Web Designers

✅ SEO = Search Engine Optimization

- Helps your site appear in **Google search results**

🧠 HTML Example:

```html
<head>
   <meta charset="UTF-8" />
```

```
    <meta name="description" content="Jane Doe – Frontend Developer Portfolio">
<meta name="keywords" content="HTML, CSS, JavaScript, Web Design">
    <meta name="author" content="Jane Doe">
    <title>Jane Doe | Portfolio</title>
</head>
```

💡 SEO Tips:

- Use **descriptive titles**
- Include **alt text** on images
- Use semantic tags like `<main>`, `<nav>`, `<footer>`

🎯 Day 5: Final Touches to Portfolio Project

✅ Checklist:

- ✅ Files named properly (`index.html`, etc.)
- ✅ Navigation works across pages
- ✅ Responsiveness verified on all devices
- ✅ Add favicon (`favicon.ico`)
- ✅ Site is live via GitHub Pages

🧠 Deployment Workflow:

1. `git add .`
2. `git commit -m "Final version"`
3. `git push`
4. Verify site on GitHub Pages

✅ Summary for Week 6:

Topic	Skills Developed
Git & GitHub	Track changes and collaborate
Hosting	Share your site publicly
Optimization	Speed up load time with smaller files

Topic	Skills Developed
SEO	Make sites discoverable
Polish & Deploy	Finalize and publish projects

🎨 Week 7: CSS Frameworks & Components

🧰 Day 1: Introduction to CSS Frameworks

✅ Key Concepts:

- **CSS Frameworks** offer pre-built styles and utilities for faster development
- Examples:
  - **Tailwind CSS** (utility-first)
  - **Bootstrap** (component-based)

💡 Why Tailwind?

- Customizable and minimal
- Encourages responsive, consistent design

🧠 Tailwind Setup (CDN):

```
<!-- Include Tailwind CSS from CDN -->
<script src="https://cdn.tailwindcss.com"></script>
```

🧱 Day 2: Components & Grid Layouts

✅ Key Concepts:

- **Components** are reusable UI blocks (cards, navbars, buttons)
- **Grids** manage two-dimensional layouts

🧠 Component Example – Card:

```
<div class="max-w-sm mx-auto bg-white shadow-lg rounded-lg overflow-hidden">
  <img class="w-full h-48 object-cover" src="project.jpg" alt="Project Image"
```

```html
/>
  <div class="p-4">     <h2 class="text-xl font-semibold mb-2">Project
Title</h2>       <p class="text-gray-600 text-sm">Brief description of the
project goes here.
     </p>
  </div>
</div>
```

🧠 Grid Layout Example:

```html
<div class="grid grid-cols-1 md:grid-cols-3 gap-6 p-6">
   <div class="bg-blue-100 p-4 rounded">Card 1</div>
   <div class="bg-blue-100 p-4 rounded">Card 2</div>
   <div class="bg-blue-100 p-4 rounded">Card 3</div>
</div>
```

🛠️ Day 3: Portfolio Redesign with Tailwind

✅ Objective:

- Rebuild your portfolio using **Tailwind CSS**
- Refactor your layout using `flex`, `grid`, and reusable components

💡 Tips:

- Extract common sections like **navbars**, **footers**
- Use `@apply` in custom CSS (when using Tailwind CLI)

```html
<!-- Navbar Component -->
<nav class="bg-gray-800 text-white p-4 flex justify-between items-center">
  <span class="font-bold text-lg">Jane Dev</span>
  <div class="space-x-4">
    <a href="index.html" class="hover:underline">Home</a>
    <a href="projects.html" class="hover:underline">Projects</a>
  </div>
</nav>
```

✨ Day 4: Animations & Scroll Effects

✅ Key Concepts:

- Tailwind provides basic transitions and animations
- Use `transition`, `transform`, `duration-300`, etc.

🧠 Example – Hover Card Animation:

```
<div class="transition-transform transform hover:scale-105 duration-300 p-4
bg-white shadow rounded">
  <h3 class="text-lg font-semibold">Hover Me</h3>
  <p class="text-gray-600 text-sm">This card zooms on hover.</p>
</div>
```

📚 Day 5: Optional JS Libraries (AOS, Swiper.js)

✅ Key Concepts:

- **AOS (Animate On Scroll)** for scroll-triggered animations
- **Swiper.js** for sliders/carousels

🧠 AOS Integration:

1. Add CDN to your `<head>`:

```
<link href="https://unpkg.com/aos@2.3.1/dist/aos.css" rel="stylesheet">
<script src="https://unpkg.com/aos@2.3.1/dist/aos.js"></script>
<script>AOS.init();</script>
```

2. Add `data-aos` attributes:

```
<div data-aos="fade-up" class="p-6 bg-gray-100 rounded">   Scroll to see me
fade in! </div>
```

🧠 Swiper.js Carousel:

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/swiper/swiper-
bundle.min.css"/>
<script src="https://cdn.jsdelivr.net/npm/swiper/swiper-bundle.min.js">
</script>
```

```html
<div class="swiper mySwiper">
  <div class="swiper-wrapper">
    <div class="swiper-slide">Slide 1</div>
    <div class="swiper-slide">Slide 2</div>
  </div>
</div>

<script>
const swiper = new Swiper(".mySwiper", { loop: true });
</script>
```

✅ Summary for Week 7:

Topic	Skills Developed
Tailwind Components	Reusable blocks (cards, buttons, navs)
Grid Layout	2D responsive layout with `grid-cols-*`
Portfolio Redesign	Professional structure with Tailwind
Animations	Visual appeal through hover/scroll effects
JS Libraries	Add carousels and scroll animations easily

🏁 Week 8: Capstone Project & Presentation

🧭 Day 1: Capstone Planning & Wireframing

✅ Key Concepts:

- Define **purpose** and **goals** of your website
- Plan sections: Home, About, Projects, Contact
- Sketch layout using **wireframes** before coding

📌 Planning Tips:

Step	Description
🎯 Goal	Portfolio, blog, or business website
📁 Pages	`index.html`, `about.html`, `projects.html`, `contact.html`
🧱 Layout	Navbar, Hero, Services, Testimonials, Footer
🖼️ Tools	Paper sketch, Figma, or Wireframe.cc

💻 Days 2–4: Build Your Capstone Website

✅ Tasks:

- Set up folder structure
- Create responsive layouts using Tailwind
- Use semantic HTML and interactivity with JavaScript

🧠 Sample Folder Structure:

```
/capstone-project/
    index.html
    about.html
    contact.html
    /css/
       styles.css
       /js/
       script.js
    /images/
       image.jpg
```

🧠 Example: Hero Section

```
<section class="bg-gray-100 text-center p-10">
  <h1 class="text-4xl font-bold mb-4">Hi, I’m Jane – Frontend Developer</h1>
  <p class="text-lg text-gray-600 mb-6">I design and code beautifully simple
websites.
  </p>
  <a href="#projects" class="bg-blue-600 text-white px-6 py-3 rounded
hover:bg-blue-700 transition">     View My Work
```

```
  </a>
</section>
```

🧠 Example: Contact Form (with JS Validation)

```
<form id="contactForm" class="max-w-md mx-auto space-y-4">
   <input type="text" id="name" placeholder="Name" class="w-full border p-2
rounded" />
   <input type="email" id="email" placeholder="Email" class="w-full border p-2
rounded" />
   <textarea id="message" placeholder="Message" class="w-full border p-2
rounded" rows="4">
   </textarea>
   <button type="submit" class="bg-green-600 text-white px-4 py-2
rounded">Send
   </button>
</form>

<script>
  document.getElementById("contactForm").addEventListener("submit", (e) => {
const name = document.getElementById("name").value.trim();
     const email = document.getElementById("email").value.trim();
        if (!name || !email) {
           e.preventDefault();
           alert("Please fill out all required fields.");
        }
    });
</script>
```

🎤 Day 5: Present & Reflect

✅ Presentation Tips:

- Use a **slide deck** (Google Slides or PowerPoint)
- Tell a story: Problem → Solution → Demo
- Include:
    - Overview of the project
    - Live demo or screen recording
    - Code highlights
    - What you learned

    - What you'd improve

💡 Reflection Prompts:

- What were your biggest challenges?
- Which tools or skills helped the most?
- What features are you proud of?

✅ Summary for Week 8:

Topic	Skills Applied
Planning	Wireframing, layout structuring, project roadmap
Building	Responsive design, semantic HTML, interactive JS
Styling	Clean UI with Tailwind CSS
Debugging	Real-world problem solving
Presentation	Professional communication and showcase

🧪 Week 9: Practical Project – Phase 1: Concept & Planning

🧭 Day 1: Define Project Scope, User Stories, Basic Structure

✅ Key Concepts:

- Define **what** your site will do
- Write **user stories** to guide features
- Plan **page structure**

🧠 Project Scope Example:

> "I want to build a recipe-sharing website where users can browse, search, and save favorite recipes."

📜 User Story Format:

> "As a [type of user], I want to [goal] so that I can [benefit]."

🧠 Examples:

- As a visitor, I want to filter recipes by category so I can quickly find dinner ideas.
- As a user, I want to view recipe details so I can follow the steps easily.

📁 Structure Plan:

```
/index.html
/about.html
/recipes.html
/contact.html
```

✏️ Day 2: Wireframing & Mockups

✅ Key Concepts:

- Visualize your layout using **wireframes**
- Create basic **mockups** (low or high fidelity)

🛠 Tools:

- Pen & paper
- Figma or Canva for digital mockups

🧠 Tip:

Include:

- Header + Navbar
- Hero section
- Feature cards
- Footer

Example Sketch:

```
[ Header / Nav ]
[ Hero Section ]
[ Categories Grid ]
[ Featured Recipes ]
[ Footer ]
```

⚙️ Day 3: Technology Stack Selection

✅ Key Concepts:

- Choose your **front-end stack**
- You'll likely use:
    - HTML + Tailwind CSS
    - Vanilla JavaScript
    - Git & GitHub for version control

✅ Optional Enhancements:

Tool	Use
AOS.js	Scroll animations
Swiper.js	Image sliders
Google Fonts	Custom typography
Formspree or Netlify	Handle form submissions without backend

📝 Days 4–5: Project Documentation – Introduction & Planning

✅ Documentation Contents:

- Project title + one-line description
- Purpose and goals
- User personas
- Page and section breakdown
- Planned features
- Technology choices
- Timeline

🧠 Example:

```
# Project Title:
QuickRecipes

## Purpose
Help users quickly discover and save easy-to-make recipes.

## Target User
Busy students and working adults looking for quick meals.

## Pages
- Home
- Categories
- Single Recipe
- About
- Contact

## Features
- Filter by category
- View recipe details
- Responsive design
```

💡 Bonus Tip:

Use Obsidian or Google Docs to organize project notes and plan content flow.

✅ Summary for Week 9:

Task	Outcome
Define Scope	Clarify site purpose and user goals
User Stories	Translate needs into site features
Wireframes	Plan layout and interface visually
Tech Stack	Choose tools you'll use
Docs	Start writing a real project spec

🔨 Week 10: Practical Project – Phase 2: Development (Frontend Focus)

🎯 Days 1–2: Coding the UI (HTML Structure & Tailwind Styling)

🧱 Task

Translate your wireframes into a working layout.

🧠 Example: Homepage Layout

```html
<!-- index.html --> <!DOCTYPE html>
<html lang="en">

   <head>
      <meta charset="UTF-8" />
      <meta name="viewport" content="width=device-width,initial-scale=1" />
      <title>QuickRecipes</title>
      <script src="https://cdn.tailwindcss.com"></script>
   </head>
   <body class="bg-gray-50 text-gray-800">

     <!-- Navbar -->
     <header class="bg-white shadow">
         <nav class="max-w-4xl mx-auto p-4 flex justify-between items-center">
            <a href="index.html" class="text-2xl font-bold">QuickRecipes</a>

            <ul class="flex space-x-6">
               <li><a href="index.html" class="hover:text-blue-600">Home</a>
               </li>
               <li><a href="recipes.html" class="hover:text-blue-
600">Recipes</a>
               </li>
               <li><a href="about.html" class="hover:text-blue-600">About</a>
               </li>
               <li><a href="contact.html" class="hover:text-blue-
600">Contact</a>
               </li>
            </ul>
        </nav>
    </header>

    <!-- Hero Section -->
    <section class="bg-green-100 p-10 text-center">
       <h1 class="text-4xl font-bold mb-4">Find Your Next Favorite Recipe</h1>
<p class="text-lg mb-6">Browse, search, and save delicious recipes in seconds.
</p>
       <a href="recipes.html" class="inline-block bg-green-600 text-white px-6
py-3 rounded hover:bg-green-700 transition">Browse Recipes</a>
    </section>
    </body>
</html>
```

- **Structure**: Clear `<header>` vs `<section>`.
- **Styling**: Utility classes for spacing (`p-4`), typography (`text-4xl`), and responsive colors.
- **Responsiveness**: By default Tailwind utilities work mobile-first; you can add breakpoints later.

🎨 Days 3–4: Styling Components & Responsive Tweaks

🏷️ Card Component Example (Recipe Preview)

```
<div class="max-w-sm bg-white rounded-lg overflow-hidden shadow-lg m-4">
   <img src="images/spaghetti.jpg" alt="Spaghetti" class="w-full h-48 object-
cover">
   <div class="p-4">
      <h2 class="text-xl font-semibold mb-2">Spaghetti Bolognese</h2>
      <p class="text-gray-600 text-sm mb-4">A classic Italian pasta dish with
rich meat sauce.
      </p>
      <a href="recipe.html" class="text-green-600 hover:underline font-
medium">View Recipe →
      </a>
    </div>
</div>
```

- **Flexibility**: Place multiple cards inside a grid:

```
<div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6 p-6">   <!--
repeat card -->
</div>
```

- **Responsive**: `sm:grid-cols-2` activates at ≥640px, `lg:grid-cols-3` at ≥1024px.

💻 Day 5: Basic JavaScript Functionality

✅ Implement Search Filter

1. **HTML**:

```
<input id="search" type="text" placeholder="Search recipes..."
class="border p-2 rounded w-full mb-6">
<div id="cards" class="grid grid-cols-1 sm:grid-cols-2 gap-4">
   <!-- Card items with data-name attributes -->
   <div class="card" data-name="spaghetti-bolognese">…</div>
   <div class="card" data-name="pancakes">…</div>   <!-- etc. -->
</div>
```

2. **JavaScript** (`script.js`):

```
const searchInput = document.getElementById("search");
const cards = document.querySelectorAll(".card");

searchInput.addEventListener("input", () => {
  const query = searchInput.value.toLowerCase();
  cards.forEach(card => {
  const name = card.getAttribute("data-name");
  card.style.display = name.includes(query) ? "block" : "none";
    });
});
```

- **Explanation**:
  - `.querySelectorAll(".card")` returns all recipe cards.
  - On each keypress (`input`), compare `data-name` to query.
  - Show/hide cards via `style.display` .

🚀 Week 11: Practical Project – Phase 3: Advanced Features

🔌 Days 1–2: Integrations & Enhancements

✅ Add Lightbox for Images

1. **Include Library** (e.g., SimpleLightbox):

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/simplelightbox/2.1.3/simple-
lightbox.min.css" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/simplelightbox/2.1.3/simple-
lightbox.min.js">
</script>
```

2. **Initialize**:

```
document.addEventListener("DOMContentLoaded", () => {
   const lightbox = new SimpleLightbox('.card img', { /* options */ });
});
```

- **Benefit**: Clicking a recipe image opens a full-screen preview.

🛒 Day 3: Local Storage for Favorites

✅ Save & Retrieve Favorites

1. **HTML** (“Save” button inside each card):

```
<button class="favorite-btn" data-id="spaghetti-bolognese">♥</button>
```

2. **JavaScript**:

```
const favButtons = document.querySelectorAll(".favorite-btn");
favButtons.forEach(btn => {
  const id = btn.dataset.id;   // On page load, mark saved favorites
  if (localStorage.getItem(id)) btn.classList.add("text-red-500");
btn.addEventListener("click", () => {
  if (localStorage.getItem(id)) {
     localStorage.removeItem(id);
     btn.classList.remove("text-red-500");
  }
  else {
  localStorage.setItem(id, true);
  btn.classList.add("text-red-500");
  }
  });
});
```

- **Explanation**:
  - Use `localStorage` to persist favorites across sessions.
  - Toggle CSS class to provide visual feedback.

🛠️ Days 4–5: Polishing & Testing Advanced Features

- **Error Handling**: Wrap JS in `try…catch` blocks when interacting with storage.
- **Accessibility**: Ensure buttons have `aria-label="Favorite recipe"`.
- **Performance**: Debounce search/filter functions to limit rapid DOM updates.

📋 Week 12: Project Documentation, Testing & Deployment

🧪 Day 1: Document Testing Process

✅ Testing Checklist:

- **Functional**: All links, forms, and buttons work
- **Responsive**: Test on mobile (≤640px), tablet (641–1024px), desktop (>1024px)
- **Accessibility**:
  - All images have `alt` text
  - Form fields have associated `<label>`s
  - Keyboard navigation (tab order) works

📝 Example Test Log:

Component	Test	Result
Navbar links	Click each link	Pass
Search filter	Type “spa”	Filters to Spaghetti card
Contact form	Submit empty form	Shows alert, prevents submit

🚢 Day 2: Deployment Steps & Final Checks

1. **Git Commit**: `git commit -am "Project complete and tested"`
2. **Push**: `git push`
3. **GitHub Pages**:
   - Ensure `index.html` at root
   - Verify live URL
4. **Custom Domain** (optional): Add `CNAME` file if you have one
5. **SSL**: GitHub Pages provides HTTPS by default

📊 Day 3: Presentation Preparation

✅ Slide Deck Outline:

1. **Title Slide**: Project name, your name, Registration number, date
2. **Problem Statement**: Why you built this site (Problem you tried to solve)
3. **Demo**: Live site or recording
4. **Key Features**: Search, favorites, responsive design
5. **Code Highlights**: Show snippets (e.g., search JS)
6. **Challenges & Learnings**
7. **Next Steps**: Potential improvements

📚 Days 4–5: Revision & Final Theory Exam

✅ Revision Topics:

- HTML semantics & accessibility
- Tailwind utility classes & responsive breakpoints
- JavaScript basics: events, DOM, localStorage
- Git/GitHub workflow
- SEO & performance best practices

📝 Sample Exam Questions:

1. **HTML**: Explain the purpose of semantic tags like `<article>`, `<section>`.

2. **Tailwind**: What does `md:grid-cols-2` do?
3. **JavaScript**: How does `e.preventDefault()` work in event handlers?
4. **Git**: Describe the difference between `git pull` and `git fetch`.
5. **SEO**: List three meta tags important for SEO.