

HANDBOOK

Rapid Web Dev using Copilot/Cursor



Table of Contents

Table of Contents

.....	0
Table of Contents	1
Course Objectives	4
Introduction to AI Enhanced Web Development	5
Overview of Modern Web Development Workflows	5
How web development workflows evolved	5
Key stages in a modern workflow	5
The role of automation and tools	7
AI-assisted development.....	7
Introduction to Copilot	8
Benefits of using Copilot	8
Limitations and precautions	9
Copilot in a typical workflow	9
Environment Setup: VS Code	9
Why a proper environment setup is important?	9
What is VS Code?	9
Key features of VS Code for web development.....	10
Steps to set up VS Code for web development	10
Cursor Editor	11
What is Cursor Editor?	12
How Cursor Editor is different from traditional editors	12
Key features of Cursor Editor.....	12
GitHub Integration	13
Why version control is necessary.....	13
What is GitHub?	13
Key steps for integrating an editor with GitHub.....	14
Prompt Engineering Basics and Approaches	15
What is prompt engineering?	15
Why prompt engineering matters in AI-assisted web development?	15
Components of a good prompt.....	15

Example prompts in a coding context.....	16
Techniques for effective prompt engineering	16
HTML Template Building with Copilot.....	16
Semantic structure and best practices	16
What is semantic HTML?	17
Why semantic HTML is important?.....	17
Best practices for semantic HTML	17
Role of Copilot in enforcing semantic structure	17
Building an HTML skeleton template with Copilot	18
What is an HTML Skeleton Template?.....	18
AI-generated Templates – Landing Pages, Blog Layout, Resume Site.....	20
What is an AI-generated HTML template?	20
Example 1: Landing Page Template	21
Try Hands-on:.....	21
Blog Layout Template	21
Resume Site Template	21
Styling with CSS using Gemini & Copilot	22
Creating reusable CSS templates with Copilot.....	22
Why reusable CSS matters?	22
How Copilot helps?	22
Interactivity with JavaScript.....	24
JS Fundamentals:	25
Variables.....	25
Functions.....	25
Control flow	26
DOM Manipulation:.....	27
What is the DOM?.....	27
Element selection.....	27
Event handling	28
Adding an event listener	28
Combining Selection and Events.....	28
AI-Suggested Scripts:.....	28
Form validation	29
What is form validation?.....	29
Types of validation	29

How to use Copilot for Form Validation	29
Role of AI	31
References	32

Course Objectives

After completing this handbook, learner will be able to

- Apply modern web development skills using AI-assisted tools like GitHub Copilot and Cursor AI.
- Enable rapid prototyping and data visualization through Streamlit for building interactive web applications.
- Introduce AI integration techniques using tools like Gemini API for smart feature development (e.g., chatbots, content generation).
- Develop proficiency in version control and cloud deployment using Git, GitHub, and Streamlit Community Cloud.
- Empower faculty to transfer industry-relevant skills to students, enhancing employability and innovation in academic projects.

Chapter 1: Rapid Web Dev using Copilot/Cursor

Learning Outcomes:

- Describe modern web development workflows and AI-assisted tools.
- Set up VS Code, Cursor Editor, and GitHub integration.
- Construct semantic HTML structures and templates using Copilot.
- Apply CSS styling and refactor styles with AI prompts.
- Develop interactive features using JavaScript and DOM manipulation.
- Implement AI-assisted scripts for validation and basic UI behavior.

Introduction to AI Enhanced Web Development

Overview of Modern Web Development Workflows

Web development is the process of creating websites and web applications that work across devices. A workflow refers to the series of steps and tools that a developer follows from the idea stage to a live, functioning application. Over time, these workflows have matured from simple coding in a text editor to structured, automated, and collaborative processes.

How web development workflows evolved

In the early stages, a developer wrote HTML pages directly in a text editor. There was no clear separation of roles or tasks. If an error occurred, everything had to be fixed manually. Over the years, the complexity of applications increased. Now websites not only display information but also process data, perform calculations, and provide interactive experiences similar to desktop applications.

Modern workflows emerged to handle this complexity. They bring structure, speed, and automation to development. They also allow multiple developers to work together efficiently. These workflows include several steps such as planning, coding, testing, deployment, and maintenance.

Key stages in a modern workflow

1. **Planning and Requirement Analysis:** The process begins with defining what needs to be built. For example, a faculty team may plan to create a website for

online assignment submissions. They will list what features are required, such as a student login, file upload, and teacher dashboard.

2. **Designing the User Interface:** The next step is to plan how the website will look and feel. Wireframes or simple sketches are created. For example, teachers might sketch how the login page, assignment upload form, and dashboard will appear.
3. **Setting up the Development Environment:** A development environment is a combination of software tools that allow a developer to write, test, and run the code. This includes a code editor, web browser, local server, and version control system. Modern editors such as VS Code help manage all these in one place.
4. **Coding and Collaboration:** Developers write code using HTML, CSS, JavaScript, and other frameworks. Version control tools such as Git allow many developers to work on the same project without overwriting each other's work. Think of it as a shared register where everyone writes but can see all changes.
5. **Testing and Debugging:** Before a website is launched, it must be tested. Testing ensures that all pages work correctly, links are functional, and there are no errors in the logic.
6. **Deployment and Maintenance:** Once the application is ready, it is deployed to a live server so that students and teachers can access it. Maintenance continues after deployment. Changes and improvements are made as needed.

Key Stages in a Modern Workflow



AI-Generated Image

The role of automation and tools

Modern workflows rely heavily on tools to automate repetitive tasks. For example, in the earlier example of an assignment submission portal:

- Automation tools can automatically check whether the uploaded files meet size and format requirements.
- Build tools prepare the application to run smoothly on different devices and browsers.
- Version control systems keep track of every change made by the developers.

This automation saves time and reduces human errors.

AI-assisted development

In recent years, AI tools such as Copilot have become part of the workflow. These tools can suggest code snippets, detect possible errors, and help developers write clean code faster. This reduces the time spent on writing repetitive code and allows developers to focus on the main logic of the application.

For example, while working on the online assignment submission portal, a developer can ask an AI assistant to generate the code for file upload validation. Instead of writing every line manually, the assistant provides a ready-to-use structure that can be customized.

Continuous Integration and Deployment

Many organizations now use Continuous Integration and Continuous Deployment (CI/CD). This means that every time a developer makes a change, automated systems test the code and, if everything works correctly, deploy the updated application. This process ensures that updates reach users faster and with fewer mistakes.

Introduction to Copilot

What is Copilot

Copilot is an AI-powered tool that works inside a code editor. It observes what a developer is typing and suggests complete lines of code or even entire functions. It does not replace the developer but acts like a pair of intelligent helping hands. It is similar to a junior assistant in a classroom who anticipates the next step and provides materials before the teacher asks.

Why Copilot is important for modern web development

When building web applications, developers often write code that is similar to patterns used in previous projects. For example, almost every web project includes user login forms, database connections, and input validation. These patterns are well known, but they take time to write from scratch. Copilot saves time by generating these standard pieces of code. This allows developers to focus on designing unique features instead of spending time on routine parts.

How Copilot works

Copilot uses machine learning models trained on a large collection of publicly available code. When a developer starts typing in an editor, Copilot analyses the context of the code and predicts what code is likely to be needed next. It then displays a suggestion that the developer can accept, reject, or modify. For example, if a developer writes the line `function addNumbers(`, Copilot can suggest the remaining code that completes the function by adding two numbers.

Benefits of using Copilot

1. Faster coding
2. Learning by example
3. Improved accuracy
4. Focus on problem-solving

Limitations and precautions

- Suggestions may be incorrect
- Not a replacement for understanding
- Privacy and responsibility

Copilot in a typical workflow

In a modern workflow, Copilot acts at the coding stage. It does not plan the project, manage versions, or deploy the application. Its primary purpose is to speed up the writing of code during development. For example, when building the online assignment submission portal discussed earlier, a developer can:

- Use Copilot to quickly generate the login page form elements.
- Use Copilot to draft the code that checks file upload limits.
- Modify and improve the suggestions to fit the project's unique requirements.

This process saves significant time compared to manually typing everything.

Environment Setup: VS Code

A good development environment helps a developer write, organize, run, and test code efficiently. One of the most widely used environments today is a code editor called VS Code. Setting up a development environment is similar to arranging a classroom before a lesson. The teacher needs a board, chalk, charts, and lesson plans ready so that the teaching process flows smoothly. In the same way, a web developer needs certain tools and configurations ready before writing the first line of code.

Why a proper environment setup is important?

Without a proper setup, coding can become disorganized. Developers may spend time fixing configuration issues instead of focusing on solving the problem at hand. A good setup ensures that:

- Code is written in a structured and readable way.
- Errors can be quickly detected and fixed.
- Testing and running applications becomes simple.
- Collaboration with other developers is smooth.

What is VS Code?

VS Code is a modern source-code editor. It is lightweight, fast, and supports multiple programming languages. It also allows developers to install extensions that add new features. VS Code is not just a text editor. It combines the functionality of an editor,

debugger, and file manager. For a developer, this means they can perform most of their work in a single window without switching between different tools.

Key features of VS Code for web development

1. Integrated file management
2. Syntax highlighting and error detection.
3. Built-in terminal
4. Extensions and customization
5. Integrated debugging.
6. Version control integration

Steps to set up VS Code for web development

Step 1: Installation

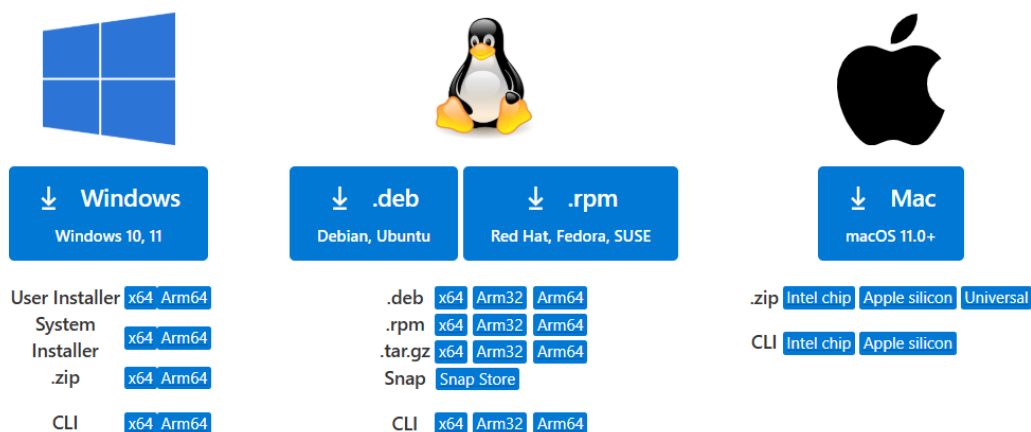
Download and install VS Code from its official source:

<https://code.visualstudio.com/download>

During installation, select options to add it to your system's path so that you can open files directly from the terminal.

Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.



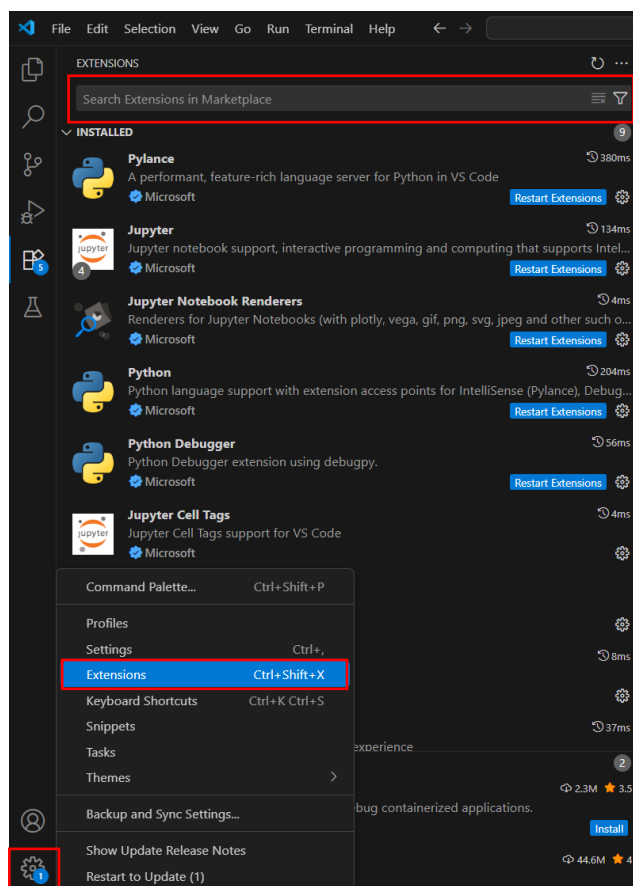
The image shows the download page for Visual Studio Code. It features three main sections: Windows, Linux, and Mac. Each section has a logo at the top, a download button, and a list of available download methods and architectures.

Platform	Download Method	Architecture
Windows	User Installer	x64, Arm64
	System Installer	x64, Arm64
	.zip	x64, Arm64
	CLI	x64, Arm64
	Windows 10, 11	
Linux	.deb	x64, Arm32, Arm64
	.rpm	x64, Arm32, Arm64
	.tar.gz	x64, Arm32, Arm64
	Snap	Snap Store
	CLI	x64, Arm32, Arm64
	Debian, Ubuntu	
	Red Hat, Fedora, SUSE	
Mac	.zip	Intel chip, Apple silicon, Universal
	CLI	Intel chip, Apple silicon
	macOS 11.0+	

Source: <https://code.visualstudio.com/download>

Step 2: Extensions

After installation, open VS Code and go to the Extensions Marketplace. Search for and install extensions that support HTML, CSS, JavaScript, and frameworks commonly used in web development. Some useful extensions also provide live preview of the webpage you are building.



Source: Screenshot

Step 3: Integrating AI tools

Install the extension for AI assistants like Copilot. Once configured, suggestions will start appearing automatically while you write code.

NOTE: To enable Copilot in VS Code, press the following keys: **Ctrl + Alt + I**. This enables the Copilot chat option within VS Code.

Cursor Editor

In the earlier sections, we learned how modern workflows depend on well-organized environments and tools like VS Code. While VS Code is widely used, new editors are emerging that focus on integrating Artificial Intelligence more deeply into the coding process. One such modern editor is Cursor Editor. It is designed from the ground up with AI assistance as a central feature, making it particularly useful in rapid development scenarios.

What is Cursor Editor?

Cursor Editor is a code editor that builds on the capabilities of VS Code but adds more advanced AI features. While traditional editors require the user to install and configure AI extensions separately, Cursor Editor comes with AI-powered features tightly integrated. This means the coding experience is more conversational, context-aware, and optimized for quick prototyping.

How Cursor Editor is different from traditional editors

1. **AI deeply integrated:** Unlike VS Code, where AI support is added through extensions, Cursor has AI as a built-in component.
2. **Natural language commands:** Cursor allows developers to type plain instructions (for example, "create a login form with two fields and a submit button") and it automatically generates the relevant code.
3. **Inline explanations:** If a developer does not understand a part of the code, they can ask Cursor to explain it in simple terms without leaving the editor.
4. **Faster iteration:** Cursor is optimized for quick cycles of code writing, testing, and revision. This is very useful during early stages of a project or in hackathon-like settings.

Key features of Cursor Editor

1. AI-assisted Code Generation
2. Code Explanation
3. Code Refactoring
4. Context Awareness
5. Collaboration Assistance

Steps to start using Cursor Editor

1. Installation.

Download and install Cursor Editor from its official source: <https://cursor.com/>



Source: <https://cursor.com/>

2. Initial configuration.

Log in and enable AI features. Most of these features are already active by default.

3. **Open or create a project.**

Import existing code or create a new folder.

4. **Interacting with AI.**

Use the dedicated AI panel or natural language commands to request suggestions, explanations, or new code.

5. **Testing and refining.**

Review the generated code carefully, test it, and make changes as necessary.

GitHub Integration

In the earlier sub-topics, we explored how modern editors such as VS Code and Cursor help developers write and manage code effectively. Once code is written, it is important to store it in a way that supports collaboration, version tracking, and backup. This is where version control systems come in. One of the most widely used version control platforms is GitHub. Integrating your editor with a version control platform helps maintain a smooth and professional workflow.

Why version control is necessary

When multiple people work on a project, keeping track of changes becomes challenging. Without a structured system:

- Developers may accidentally overwrite each other's work.
- It becomes difficult to find which version of code worked correctly.
- Recovering an older version of a file can be time-consuming.

Version control systems solve these problems by recording every change made to the codebase, along with the author and time of the change. They allow developers to return to earlier versions if needed.

A simple analogy is the way teachers keep multiple versions of question papers or assignments. Each version is dated and stored separately, so they can review or reuse earlier versions when required.

What is GitHub?

GitHub is an online platform that stores projects managed by Git (a version control system). Git records every change made to files, while GitHub provides a shared, cloud-based place for teams to work together on those projects.

GitHub allows:

- **Collaboration:** Multiple developers can work on the same project without overwriting each other's work.
- **Version history:** Every change is saved, so older versions can be restored if necessary.

- **Branching:** Developers can create separate branches of a project to test new features before merging them into the main code.
- **Backup:** Projects stored on GitHub are safe even if something happens to the local system.

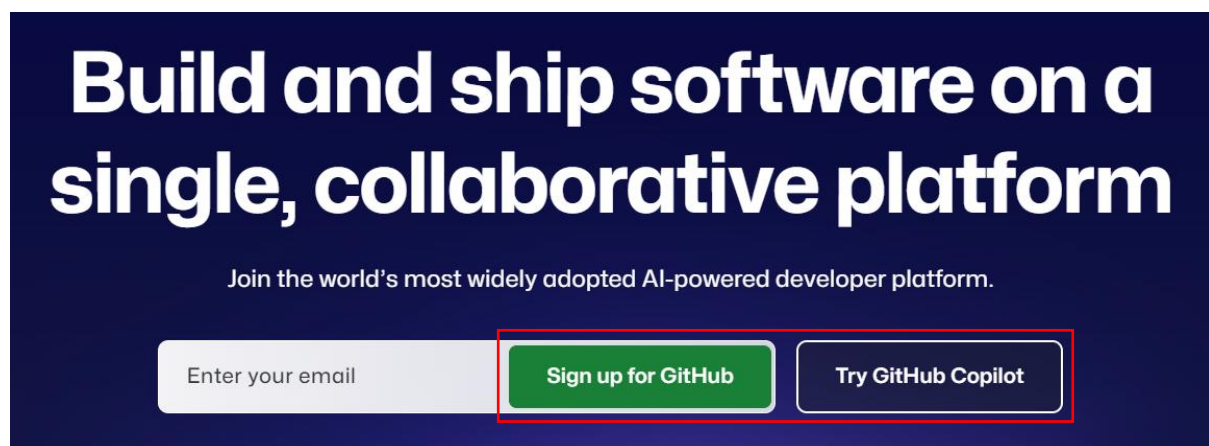
Key steps for integrating an editor with GitHub

Step 1: Install Git

Install Git on the local machine from the official source: <https://git-scm.com/downloads/win>. This tool tracks changes in your files.

Step 2: Sign in to GitHub

Create an account on GitHub: <https://github.com/> and sign in from within your code editor.



Source: <https://github.com/>

Step 3: Initialize a repository

In the editor, initialize a repository. This prepares the local project for version tracking.

Step 4: Stage and commit changes

After making changes to code, the editor shows modified files. These changes can be staged (selected) and committed (saved with a note describing the changes).

Step 5: Push changes to GitHub

Once committed, the changes can be pushed to the GitHub repository. Other developers can pull these changes to their systems.

Step 6: Branching and merging

Create branches for experimenting with new features. Once tested, branches can be merged back into the main code. This avoids disturbing the stable version.

Prompt Engineering Basics and Approaches

In earlier sub-topics, we explored how AI assistants such as Copilot and editors like Cursor are changing the way web development is done. These AI-powered tools rely on user input to understand what needs to be generated or explained. This input is called a prompt. The way a prompt is written directly affects the quality and relevance of the AI's output. Therefore, understanding how to design good prompts—known as prompt engineering—has become an important skill in modern workflows.

What is prompt engineering?

Prompt engineering is the process of framing clear, specific, and structured instructions so that an AI assistant can produce useful results. A poorly written prompt may result in vague or incorrect suggestions, while a well-designed prompt can guide the AI to produce code or explanations that meet the user's exact needs.

Prompt engineering is similar to giving instructions to students. If a teacher gives incomplete instructions such as “Do the project,” students may be confused and produce very different outcomes. But if the teacher gives clear instructions about topics, format, and expected outcomes, students are more likely to produce work that meets expectations.

Why prompt engineering matters in AI-assisted web development?

In web development, especially when using AI-powered tools like Copilot or Cursor, the AI depends on prompts to:

- Generate new code.
- Refactor existing code.
- Explain complex code segments.
- Suggest improvements.

If the prompt is unclear, the AI may generate suggestions that do not match the requirements. Good prompt engineering reduces the time spent rewriting or discarding irrelevant suggestions.

Components of a good prompt

1. **Context:** Provide enough background so that the AI understands the task. For example, instead of saying “Create a form,” you might say, “Create an HTML form with two text fields for username and password and a submit button.”
2. **Specific details:** Mention the technologies, frameworks, or design requirements. For example, specify “responsive layout using HTML and CSS” instead of just “layout.”
3. **Desired output format:** If you want the code in a specific structure or language, include that in the prompt.

4. **Constraints or limitations:** Mention if there are restrictions, such as “avoid using external libraries” or “keep the code within 20 lines.”

Example prompts in a coding context

Poor prompt: Make a page.

Improved prompt: Generate an HTML page with a login form that includes fields for username and password, a submit button, and a simple CSS style. The layout should be responsive and fit both mobile and desktop screens.

The improved prompt clearly states what is expected, which increases the chance of receiving accurate code suggestions.

Techniques for effective prompt engineering

1. **Think step by step:** Break complex tasks into smaller prompts. For example, first prompt for the structure of a page, then prompt for styling, and later prompt for validation scripts.
2. **Iterate:** If the output is not correct the first time, refine the prompt with more detail. Each prompt can build upon the previous response.
3. **Use natural language:** AI assistants understand plain English. Write prompts as if you are giving instructions to a junior developer.
4. **Experiment:** Different prompts can result in different solutions. Do not hesitate to try different approaches.
5. **Review and correct:** Always verify the output. Treat AI suggestions as a starting point, not a final solution.

HTML Template Building with Copilot

HTML (HyperText Markup Language) forms the backbone of every website. It defines the structure and layout of a webpage, while CSS and JavaScript add design and interactivity. In modern web development, the task of building HTML templates has become faster and more reliable with the support of AI-powered assistants such as Copilot. Traditionally, developers created HTML pages manually, starting from a blank file. They had to remember tag names, attributes, and correct structure. This approach was time-consuming and often led to small but frequent errors like missing tags or incorrect nesting. Today, tools such as Copilot assist developers by suggesting standard structures, generating repetitive elements, and even creating entire templates for different types of websites.

Semantic structure and best practices

HTML is not only about creating something that displays correctly on a browser; it is about structuring content in a way that is meaningful and accessible. A well-structured HTML page improves readability for humans and machines.

What is semantic HTML?

Semantic HTML uses tags that have a specific meaning. For example:

- `<header>` indicates the header section of a page.
- `<main>` defines the main content area.
- `<article>` is used for self-contained content like a blog post.
- `<footer>` is used for the page footer.

These tags convey the purpose of each section, which makes it easier for browsers and tools to understand the content. In contrast, non-semantic tags like `<div>` and `` do not carry meaning by themselves and are used mainly for styling and grouping.

Why semantic HTML is important?

1. **Accessibility:** Screen readers and other assistive technologies can navigate semantic pages better, making content accessible to everyone.
2. **SEO (Search Engine Optimization):** Search engines can interpret semantic HTML more accurately, improving the visibility of the website.
3. **Readability and maintainability:** Semantic structure makes it easier for other developers to understand the layout and purpose of sections when maintaining or updating the code.
4. **Professional standards:** Using semantic tags is part of good coding practices followed in the industry.

Best practices for semantic HTML

1. **Use correct tags for correct purposes:** For example, use `<nav>` for navigation menus instead of generic `<div>` containers.
2. **Organize content hierarchically:** Use heading tags (`<h1>` to `<h6>`) in proper order, similar to chapters and subheadings in a book.
3. **Avoid unnecessary nesting:** Adding too many layers of `<div>` or `` tags makes the code harder to manage.
4. **Keep structure separate from styling:** Use HTML for structure and CSS for design. Do not mix inline styling unnecessarily.

Role of Copilot in enforcing semantic structure

Copilot helps ensure that semantic structure is followed correctly. When a developer starts typing, Copilot suggests the right tags. For example:

- If the developer begins typing `<header>`, Copilot may automatically suggest the closing `</header>` and prompt for navigation links.

- If the developer writes “main layout” in a comment, Copilot often suggests a complete structure including `<main>`, `<section>`, and `<footer>` tags.

This guidance reduces the likelihood of forgetting important elements or using the wrong tags.

Building an HTML skeleton template with Copilot

Once the concept of structure is clear, the next step is to build a basic HTML skeleton template. This skeleton serves as a starting point for any webpage. It includes the essential sections of a page but without detailed content or styling. In a traditional workflow, developers create this skeleton manually by typing each tag. With Copilot, this process becomes faster because it can generate the basic template automatically as soon as the developer begins.

What is an HTML Skeleton Template?

An HTML skeleton is a minimal structure that includes:

1. **DOCTYPE declaration.**
2. **`<html>`** root element with language attributes.
3. **`<head>`** contains metadata, title, and links to stylesheets.
4. **`<body>`** will later hold the main content.

Before adding text, images, and design, you need a skeleton structure for the webpage.

Steps to create an HTML skeleton template with Copilot

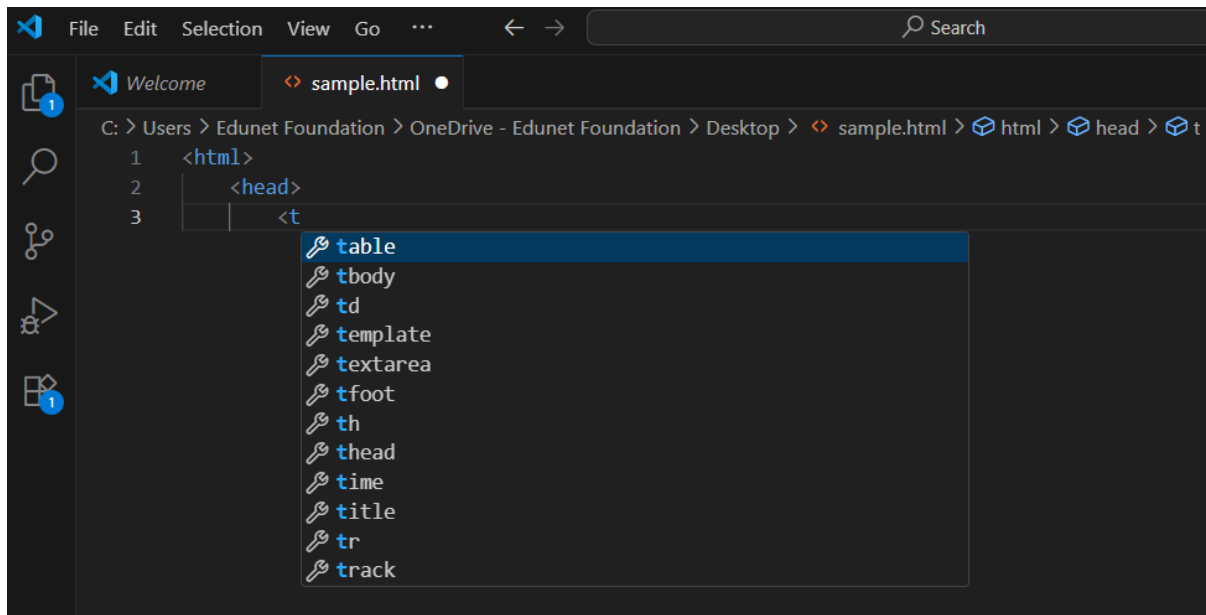
Step 1: Create a new file

Open your editor (VS Code or Cursor) and create a new file with the extension `.html`.

File > New File > Type the filename.html

Step 2: Start typing '!' or the word html

Most editors, especially with Copilot enabled, automatically suggest a complete HTML boilerplate.



Step 3: Add the head section

Include important metadata:

- `<meta charset="UTF-8">`
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`
- `<title>` for the page title

```
1 <html>
2   <head>
3     <meta charset="UTF-8">
4     <meta name="viewport" content="width=device-width, initial-scale=1.0">
5     <title> SAMPLE WEB PAGE </title>
6   </head>
7 </html>
```

Copilot often fills these in automatically once you start typing.

Step 4: Define the body structure

Add the `<body>` tag and leave placeholders for sections like header, main, and footer. Copilot frequently offers suggestions here, such as:

Example workflow with Copilot: Consider you are building a **Sample homepage**. After typing:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

<head>

Copilot might generate:

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sample HTML Document</title>

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Sample HTML Document</title>
7    </head>
8  </html>
```

This saves time, eliminates the need to remember every tag, and encourages developers to begin with a complete, valid structure.

Advantages of using Copilot for skeleton templates

1. Speed
2. Consistency
3. Error reduction
4. Focus on logic

AI-generated Templates – Landing Pages, Blog Layout, Resume Site

Once the basic skeleton of an HTML page is in place, the next step is to build complete templates that include structured sections, content placeholders, and sometimes basic styling. Traditionally, developers would spend considerable time writing repetitive code for common layouts. With the help of Copilot, this process has become faster because AI can generate these templates from natural language descriptions.

What is an AI-generated HTML template?

An AI-generated template is a block of HTML code (often including semantic structure) automatically suggested by AI tools such as Copilot when a developer describes the structure in plain language. These templates provide:

- A starting point with pre-defined sections.
- Clean and readable structure.
- Time savings by avoiding repetitive typing.

Instead of building from scratch, developers can prompt the AI with instructions.

Example 1: Landing Page Template

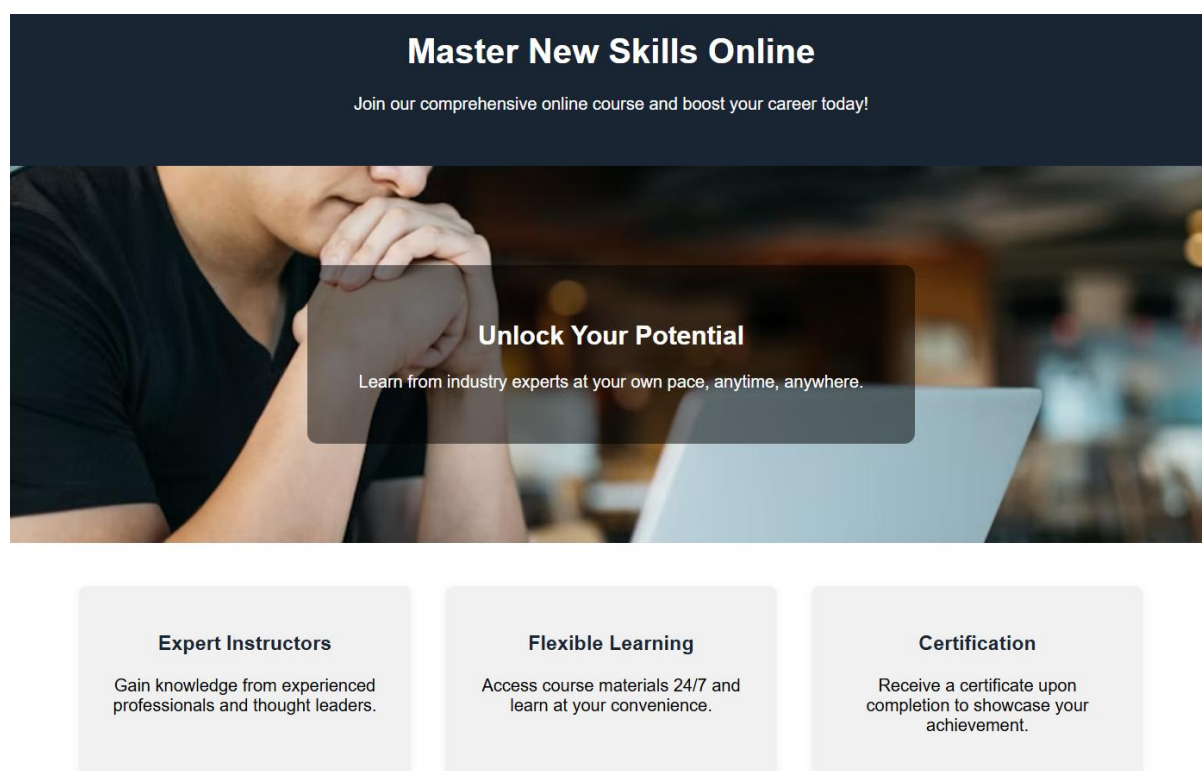
A landing page is a single web page designed to focus on a specific purpose, such as introducing a course, promoting an event, or showcasing a product.

Example Copilot Prompt:

“Build a simple landing page for an online course with a hero image, 3 features, and a contact form”

Once the code is generated, click on **Run** on the top panel

Output:



Try Hands-on:

Blog Layout Template

Copilot Prompt: “Create a blog layout with a list of articles. Each article should have a title, author, date, and a short summary. Include a sidebar with links.”

Resume Site Template

Copilot Prompt: “Create a personal resume webpage with sections for About, Education, Skills, Projects, and Contact”.

Benefits of using Copilot for Template generation

1. Rapid Prototyping
2. Consistency
3. Encourages experimentation
4. Saves repetitive effort

Styling with CSS using Gemini & Copilot

After creating the HTML structure of a webpage, the next step is to make it visually appealing. This is achieved using CSS (Cascading Style Sheets). CSS defines colors, fonts, spacing, alignment, and overall presentation. While HTML focuses on structure, CSS ensures that the page is attractive and easy to use.

Traditionally, developers manually write CSS rules for each element or section. This is time-consuming and prone to repetitive work, particularly when the same styles are applied across multiple pages. Today, AI tools such as Copilot and Gemini can assist by generating reusable CSS templates, refactoring existing styles, and even suggesting improvements based on best practices.

Creating reusable CSS templates with Copilot

Why reusable CSS matters?

In modern websites, a consistent look across pages is essential. Instead of rewriting similar CSS rules multiple times, developers create reusable style sheets. These style sheets define:

- Color schemes
- Typography (fonts and sizes)
- Common layouts (grids and flexboxes)
- Reusable components like buttons and cards

By centralizing these styles in one place, changes can be applied to the entire website by updating a single file.

How Copilot helps?

Copilot can generate reusable CSS templates by analyzing the HTML structure and understanding the developer's prompt. For example, when a developer types: "Create a basic CSS template for a responsive layout with a header, main content, and footer"

Copilot can suggest:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

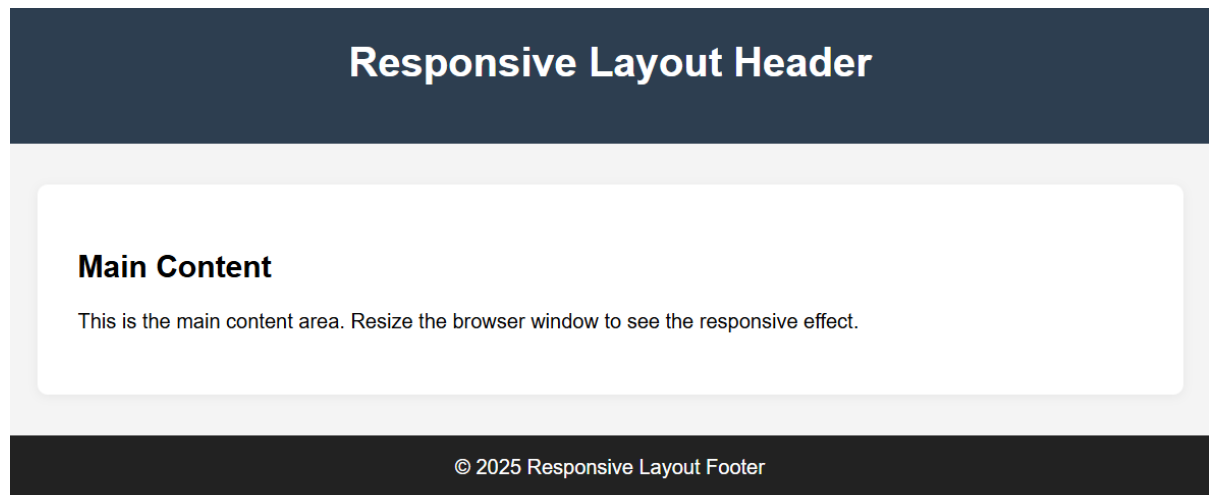
```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Responsive Layout Template</title>
<style>
  * { box-sizing: border-box; }
  body { margin: 0; font-family: Arial, sans-serif; background: #f4f4f4;
}

  header {
    background: #2d3e50;
    color: #fff;
    padding: 1.5rem 0;
    text-align: center;
  }
  main {
    max-width: 900px;
    margin: 2rem auto;
    background: #fff;
    padding: 2rem;
    border-radius: 8px;
    box-shadow: 0 2px 8px rgba(0,0,0,0.05);
  }
  footer {
    background: #222;
    color: #fff;
    text-align: center;
    padding: 1rem 0;
    position: relative;
    bottom: 0;
    width: 100%;
  }
  @media (max-width: 600px) {
    main {
      padding: 1rem;
      margin: 1rem;
    }
    header, footer {
      padding: 1rem 0;
    }
  }
</style>
</head>
<body>
  <header>
    <h1>Responsive Layout Header</h1>
  </header>
  <main>
    <h2>Main Content</h2>
    <p>This is the main content area. Resize the browser window to see the
responsive effect.</p>
```



```
</main>
<footer>
    &copy; 2025 Responsive Layout Footer
</footer>
</body>
</html>
```

Output



Advantages of using Copilot for CSS

1. Speeds up repetitive tasks
2. Encourages consistency
3. Introduces responsive patterns

Interactivity with JavaScript

HTML and CSS define the structure and presentation of a webpage, but modern websites require interactivity. This is achieved through JavaScript (JS), a scripting language that runs in the browser. JavaScript can respond to user actions, validate inputs, display messages, and update the page without requiring a full reload. In traditional development, JavaScript is typed manually by the developer. With AI tools like Copilot, many repetitive scripts can be suggested automatically, reducing development time. However, it is still essential to understand the basics of JavaScript before relying on AI assistance.

JS Fundamentals:

Before adding interactivity, a developer needs to understand the building blocks of JavaScript. These building blocks allow developers to store data, perform actions, and make decisions based on conditions.

Variables

Variables are used to store data such as numbers, text, or any information that the script needs. In JavaScript, variables can be declared using `let`, `const`, or `var`.

- **let** is used when a value may change during execution.
- **const** is used when a value should not change.
- **var** is the older form of variable declaration and is less commonly used in modern code.

NOTE: Before trying the below given example, make sure `node.js` is installed in the system

Example:

```
let studentName = "Rahul";
const maxMarks = 100;

console.log("Before:", studentName, maxMarks);

// Changing the value of studentName (allowed)
studentName = "Priya";
console.log("After changing name:", studentName, maxMarks);

// Changing maxMarks (NOT allowed)
// maxMarks = 200; // This will cause an error
```

Output

```
C:\Program Files\nodejs\node.exe --experimental-network-inspection .\trial.js
Before: Rahul 100
After changing name: Priya 100
```

Here, **studentName** can change, but **maxMarks** remains constant.

Functions

What are functions?

Functions are blocks of code that perform specific tasks. They allow reuse of code, which reduces duplication.

Example:

```
function calculatePercentage(marksObtained, totalMarks) {  
    return (marksObtained / totalMarks) * 100;  
}  
  
console.log(calculatePercentage(80, 100)); // Output: 80
```

Output

```
C:\Program Files\nodejs\node.exe --experimental-network-inspection .\trial.js  
80
```

Control flow

Control flow allows scripts to make decisions and repeat actions. Three common concepts are:

Conditional statements (if, else if, else).**Example:**

```
let marks = 80; // You can change this value as needed  
let grade;  
if (marks >= 90) {  
    grade = "A";  
} else if (marks >= 75) {  
    grade = "B";  
} else {  
    grade = "C";  
}  
console.log("Grade:", grade);
```

Output

```
C:\Program Files\nodejs\node.exe --experimental-network-inspection .\trial.js  
Grade: B
```

Loops (for, while): Loops are used to repeat a set of instructions.

Example:

```
for (let i = 1; i <= 5; i++) {  
    console.log("Student " + i);  
}
```

Output

```
C:\Program Files\nodejs\node.exe --experimental-network-inspection .\trial.js
Student 1
Student 2
Student 3
Student 4
Student 5
```

This prints “Student 1” to “Student 5”.

Switch statements: Switch helps when there are multiple conditions to check.

How AI assists with JS fundamentals

When a developer starts typing JavaScript code, Copilot can suggest:

- Pre-written functions based on variable names.
- Common structures like loops or conditionals.
- Explanations for complex syntax.

This allows faculty and students to focus more on understanding the logic rather than typing every line.

DOM Manipulation:

Once the basics of JavaScript are clear, the next important concept is interacting with the Document Object Model (DOM). The DOM is the structured representation of a webpage that browsers create from HTML. Using JavaScript, developers can access, modify, and control parts of this structure dynamically, which makes the page interactive.

What is the DOM?

The DOM represents a webpage as a tree of elements. Each tag in HTML (like <div>, <p>, <button>) becomes a node in this tree. JavaScript can be used to:

- Select elements in this tree.
- Change their content or styling.
- Add or remove elements dynamically.

Element selection

Purpose: To access a specific part of the page.

Common methods to select elements:

getElementById(): Selects a single element using its id attribute.

Example: let heading = document.getElementById("mainTitle");

1. **getElementsByClassName()**
Selects elements with a given class.
2. **querySelector() and querySelectorAll()**
More flexible methods that use CSS-like selectors.

Example:

```
let button = document.querySelector(".submit-button");
```

Event handling

Purpose: To respond when a user interacts with the page.

Events include:

- Clicking a button.
- Typing in a text box.
- Moving the mouse over an element.

Adding an event listener**Example:**

```
let button = document.getElementById("submitButton");  
button.addEventListener("click", function() {  
    alert("Form submitted successfully!");  
});
```

When the user clicks the button, the function inside `addEventListener` is executed.

Combining Selection and Events

Together, element selection and event handling allow developers to create rich interactivity. For example:

- Show or hide a message when a button is clicked.
- Change the background color when a user hovers over a section.
- Display a student's details when a teacher selects their name from a list.

AI-Suggested Scripts:

One of the most common interactive tasks on a webpage is validating user input. When a user fills out a form, such as entering their name, email, or marks, it is essential to ensure that the information is correct before it is submitted. JavaScript is widely used to perform this validation in the browser. Traditionally, developers wrote validation scripts manually. With AI tools like Copilot, these scripts can be generated quickly by

writing a clear prompt or comment. This not only saves time but also ensures that best practices are followed.

Form validation

What is form validation?

Form validation is the process of checking whether user input meets specific requirements before submitting it to the server.

Examples of validation:

- Ensuring that a text field is not left empty.
- Confirming that an email address is in the correct format.
- Making sure that numerical values are within an acceptable range.

Validation improves data quality and provides immediate feedback to the user.

Types of validation

1. **Client-side validation:** Done using JavaScript in the browser before the form is submitted. Example: Displaying a message immediately if the email field is blank.
2. **Server-side validation:** Done after the form data reaches the server. Even if client-side validation is used, server-side validation must still be performed for security.

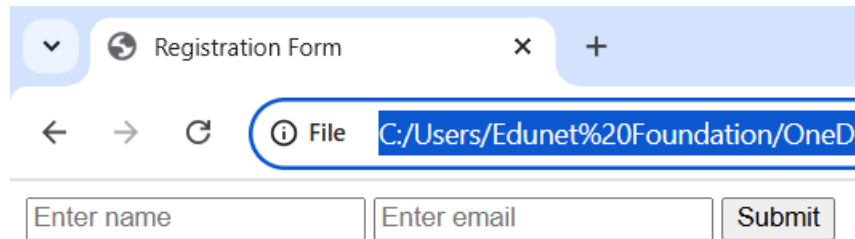
How to use Copilot for Form Validation

Step 1: Create a form

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Registration Form</title>
</head>
<body>
  <form id="registrationForm">
    <input type="text" id="name" placeholder="Enter name">
    <input type="email" id="email" placeholder="Enter email">
    <button type="submit">Submit</button>
  </form>
```

Output



Registration Form

File C:/Users/Edunet%20Foundation/OneD

Enter name Enter email Submit

Step 2: Add a comment

Press Enter and type `// validate form fields before submission`

Now Press **Ctrl + I** so that Copilot recommends the code

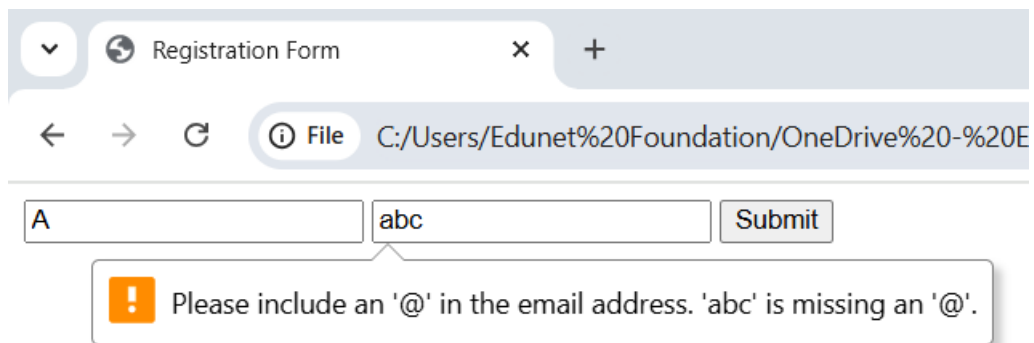
Step 3: Accept Copilot suggestions

Copilot may suggest:

```
<script>
    document.getElementById('registrationForm').addEventListener('submit',
function(event) {
    var name = document.getElementById('name').value.trim();
    var email = document.getElementById('email').value.trim();
    if (name === '' || email === '') {
        alert('Please fill in all fields.');
```

This script prevents submission if the fields are empty or invalid.

Output



The screenshot shows a web browser window with a single tab titled "Registration Form". The address bar displays a local file path: "C:/Users/Edunet%20Foundation/OneDrive%20-%20E". Below the address bar, there is a form with two input fields and a "Submit" button. The first input field contains the letter "A", and the second input field contains "abc". A yellow error message box is positioned below the "abc" field, stating: "Please include an '@' in the email address. 'abc' is missing an '@'." The error message is preceded by a yellow square icon containing a black exclamation mark.

Role of AI

Copilot:

- Detects that the task involves form validation.
- Suggests common checks like empty fields or incorrect email formats.
- Reduces repetitive code for error handling.

References

- Brains, E. (2024, November 18). Code less, Achieve More: Automating workflows in modern web development. Medium.
<https://medium.com/@emperorbrains/code-less-achieve-more-automating-workflows-in-modern-web-development-a47a0fbbab4b>
- Cursor. (n.d.). Cursor - the AI code editor. <https://cursor.com/>
- Holcombe, J. (2025, May 23). What Is GitHub? A Beginner's Introduction to GitHub. Kinsta®. <https://kinsta.com/blog/what-is-github/>
- Jarzynski, P. (2021, December 23). A modern web development workflow explained - the startup - medium. Medium. <https://medium.com/swlh/a-modern-web-development-workflow-explained-c96bd68ff79c>
- Microsoft. (2024, October 1). What is Copilot, and how can you use it? Microsoft Copilot. <https://www.microsoft.com/en-us/microsoft-copilot/for-individuals/do-more-with-ai/general-ai/what-is-copilot?form=MA13KP>
- Muchmore, M. (2025, July 30). What is Copilot? Microsoft's AI assistant explained. PCMAG. <https://www.pcmag.com/explainers/what-is-microsoft-copilot>
- Mzizi, S., & Mzizi, S. (2024, June 14). HTML, CSS, and JavaScript: Essential Front-End languages explained. ITonlinelearning.
<https://www.itonlinelearning.com/blog/html-css-and-javascript-essential-front-end-languages-explained/>
- Prompt Engineering for AI Guide. (n.d.). Google Cloud.
<https://cloud.google.com/discover/what-is-prompt-engineering>
- Python environments in VS Code. (2021, November 3).
<https://code.visualstudio.com/docs/python/environments>
- Setting up Visual Studio Code. (2021, November 3).
<https://code.visualstudio.com/docs/setup/setup-overview>
- Understanding HTML, CSS, and JavaScript | Documentation. (n.d.).
<https://docs.wweb.io/web-development-basics/understanding-html-css-javascript.html>
- What is a copilot and how does it work? | Microsoft Copilot. (n.d.).
<https://www.microsoft.com/en-us/microsoft-copilot/copilot-101/what-is-copilot>
- What is prompt engineering? (2024, March 22). McKinsey & Company.
<https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-prompt-engineering>
- Working with GitHub in VS Code. (2021, November 3).
<https://code.visualstudio.com/docs/sourcecontrol/github>