**Setup**

**Setting Up Your web3 Frontend Project**

Welcome! This lesson guides you through setting up the foundational structure for a simple web application designed to interact with a Solidity smart contract. We'll create the necessary files and build the basic HTML layout using standard web development practices.

**Creating the Project Environment**

First, we need to establish a dedicated workspace for our project. This ensures all related files are neatly organized.

1. **Choose Your Base Directory:** Open your terminal or command prompt and navigate to the directory where you typically store your coding projects. In this example, we'll assume a base directory named full-stack-web3-cu.
2. **Create the Project Folder:** Use the mkdir (make directory) command to create a new folder specifically for this lesson. We'll name it html-ts-coffee-cu to reflect its content (HTML, eventually TypeScript, and the "Buy Me a Coffee" contract focus).

mkdir html-ts-coffee-cu

1. **Navigate into the Project Folder:** Change your current directory to the newly created one.

cd html-ts-coffee-cu

1. **Open in Code Editor:** Open this folder in your preferred code editor. If you're using Visual Studio Code (VS Code), you can often do this directly from the terminal:

code .

Alternatively, you can open VS Code and use the File > Open Folder... menu option.

1. **Verify Location:** It's good practice to confirm you're working in the correct directory. Use the pwd (print working directory) command in your editor's integrated terminal:

pwd

The output should show the full path ending with /html-ts-coffee-cu.

**Defining the Application Scope with README.md**

Before writing code, it's beneficial to define what we're building. A README.md file serves as an excellent place for this "product specification."

1. **Create the File:** In the root of your html-ts-coffee-cu directory, create a new file named README.md.
2. **Add Content:** Open README.md and outline the application's goals. Using Markdown, we can list the key features:

# What are we making?

- Minimal HTML/JS site

- That has the following buttons which map to the solidity smart contract:

- Connect

- Buy Coffee

- Get Balance

- Withdraw

This simple specification clearly states our objective: build a basic frontend with four specific buttons, each intended to trigger a function on our smart contract. We'll start with JavaScript (JS) and introduce TypeScript (TS) later.

*(Aside: AI for Code Generation)*  
This README.md specification is clear enough that you could potentially use an AI coding assistant (like DeepSeek at chat.deepseek.com, as shown in the video) to generate initial code. You would typically copy the Markdown content, paste it into the AI chat interface (often enclosed in backticks ```), and prompt it, for example: "Here is my product spec... Could you please make me a basic HTML template with this?". While AI can generate functional HTML, CSS, and JavaScript, it often includes more complexity (like styling) than needed for initial learning. For this tutorial, we'll build the components manually step-by-step to ensure a clear understanding, using AI more as a helper later if needed.

**Building the Basic HTML Structure**

Now, let's create the main HTML file for our application.

1. **Create**index.html**:** In the project root, create a file named index.html. This is the default file web servers look for.
2. **Basic HTML Document:** Start with the fundamental HTML structure:

<!DOCTYPE html>

<html>

</html>

* + <!DOCTYPE html>: Declares the document type to the browser.
  + <html>: The root element enclosing all other HTML content. HTML tags generally come in pairs: an opening tag (<tag>) and a closing tag (</tag>), surrounding the content they affect.

1. **Add the Head Section:** The <head> contains meta-information about the page, not the visible content itself. Let's add it, including a <title>:

<!DOCTYPE html>

<html>

<head>

<title>Buy me a coffee</title>

</head>

</html>

* + <head>: Section for metadata, scripts, styles, etc.
  + <title>: Defines the text shown in the browser's title bar or tab. You can view this by opening index.html in a browser. Using a tool like the "Live Server" extension in VS Code is highly recommended, as it automatically refreshes the browser when you save changes.

1. **Add the Body Section:** The <body> contains the actual content visible to the user.

<!DOCTYPE html>

<html>

<head>

<title>Buy me a coffee</title>

</head>

<body>

<!-- Visible page content goes here -->

</body>

</html>

* + <body>: Holds elements like text, headings, images, and buttons that form the user interface.

**Adding Initial User Interface Elements**

Let's add the first button based on our README.md specification.

1. **Create a Simple Button:** Inside the <body> tags, add a basic <button> element:

<body>

<button>Hi!</button>

</body>

If you view this in your browser (using Live Server or by opening the file directly), you'll see a clickable button labeled "Hi!".

1. **Create the "Connect" Button with an ID:** Now, let's create the specific "Connect" button from our plan. Crucially, we'll add an id attribute:

<body>

<button id="connectButton">Connect</button>

</body>

* + <button>: Creates a clickable button element.
  + id="connectButton": The id attribute provides a unique identifier for this specific HTML element. This is essential because it allows our JavaScript code (which we'll add later) to easily find and interact with this particular button, enabling us to add functionality like connecting to a user's web3 wallet when it's clicked.

With the project directory created, the scope defined in README.md, and the basic index.html structure including our first functional element placeholder, we have successfully set the foundation for our web3 frontend application. The next steps will involve adding the remaining buttons and implementing the JavaScript logic to interact with them.