**Challenges**

**Challenge: Querying Smart Contract Data from the Frontend**

Having successfully set up the basic HTML and JavaScript frontend for our "Buy Me A Coffee" dApp and wired up the core funding functionality, it's time to solidify your understanding with a practical challenge. You've already completed the first introductory task (reflecting on your learning goals); this next challenge delves deeper into frontend-to-smart-contract interaction, specifically focusing on reading data.

This challenge is a bit more involved than the first. Your task is to implement a new feature on the frontend:

1. **Create a New Button:** Add a new HTML button element to your index.html page. You can label it something descriptive like "Check Funding Amount" or "Query My Donations".
2. **Add Click Functionality:** Using JavaScript (in your index.js or equivalent file), add an event listener to this new button.
3. **Interact with the Smart Contract:** When the button is clicked, your JavaScript code should interact with our deployed Fundme.sol smart contract (currently running on your local Anvil chain). Specifically, it needs to call the getAddressToAmountFunded function.
4. **Log the Output:** The value returned by the getAddressToAmountFunded function should be logged to the browser's developer console using console.log().

**Understanding the Target Function:**getAddressToAmountFunded

To successfully complete this challenge, you need to understand the specific smart contract function you'll be calling. Let's look at its definition within Fundme.sol (located at foundry-fund-me-cu/src/Fundme.sol in the course repository):

/\*\*

\* @notice Gets the amount that an address has funded

\* @param fundingAddress the address of the funder

\* @return the amount funded

\*/

function getAddressToAmountFunded(address fundingAddress) public view returns (uint256) {

return s\_addressToAmountFunded[fundingAddress];

}

Key points about this function:

* **Purpose:** It checks the internal mapping s\_addressToAmountFunded to see how much Ether (in Wei) a specific address has donated to the contract.
* **Argument:** It requires one argument: fundingAddress, which is the Ethereum address you want to query.
* **Return Value:** It returns a uint256 representing the total amount funded by the specified fundingAddress.
* view**Function:** Notice the view keyword. This is crucial. It signifies that this function only reads data from the blockchain state; it does not modify it.

**Key Concept: Read vs. Write Operations**

This challenge introduces a fundamental concept: the difference between *read* and *write* operations when interacting with smart contracts.

* **Write Operation (e.g., Funding):** When a user clicked the "Buy Coffee" button, they initiated a *write* operation. This involved sending a transaction to the fund function, which modified the blockchain's state (updated the contract's balance and the s\_addressToAmountFunded mapping). Write operations require gas fees and need to be signed by a user's wallet (like Metamask) to authorize the state change.
* **Read Operation (e.g., Calling**getAddressToAmountFunded**):** Calling a view function like getAddressToAmountFunded is a *read* operation. It simply queries the current state of the blockchain without changing anything. Because no state is modified, read operations:
  + Do not require signing a transaction with a wallet.
  + Do not cost gas (when called via a node's read interface, as is typical from a frontend).
  + Are generally much faster than write operations.

**Implementation Guidance**

Here are some steps and hints to guide you:

1. **Modify HTML:** Add a <button> element with a unique ID in your index.html.
2. **Modify JavaScript:**
   * Get a reference to your new button using its ID.
   * Attach an event listener (e.g., 'click').
   * Inside the event handler function (make it async):
     + Ensure you have access to your ethers.js Contract object instance.
     + Determine which fundingAddress you want to query. **This is part of the challenge!** You might hardcode your own address for testing, use the currently connected wallet address (if available), or even add an input field to the HTML to let the user specify an address.
     + Call the getAddressToAmountFunded method on your contract object, passing the chosen address as an argument. Remember that contract function calls are asynchronous, so use await.
     + Store the returned value (which will likely be a BigNumber object from ethers.js).
     + Use console.log() to display the result. You might want to convert the BigNumber to a string for easier reading (result.toString()).

**Before Proceeding**

This challenge provides valuable hands-on experience interacting with smart contracts in a read-only context. Take your time:

* **Pause the course** and dedicate time to implementing this feature.
* **Consult documentation:** Refer to the documentation for the web3 library you are using (e.g., ethers.js) to understand how to call contract functions, especially view functions that require arguments.
* **Leverage AI:** Don't hesitate to use AI assistants (like ChatGPT, Claude) to help you understand specific code snippets, debug errors, or explore different ways to achieve the goal.

Give it your best shot! We'll review a potential solution after you've had a chance to work through it.