User

what framework is actual for creating modern web UI?

Bing

There are several popular frameworks for creating modern web UIs. According to a list of the 10 best front-end frameworks for web development in 2023 by Monocubed, **React** and **Svelte** are the leading frameworks for frontend developers. Another option is **FAST**, which is a collection of technologies built on Web Components and modern Web Standards, designed to help efficiently tackle some of the most common challenges in website and application design and development. Is there a specific framework you’re interested in?

User

ok. Lets stick with React. I need simple UI that contains of: Title “Blockchain calculator” One line input field with placeholder “Number a” Dropdown with “+”, “-”, “:”, “*” math operations. Nothing selected by default. One line input field with placeholder “Number b” Empty one line for output that looks like input with placeholder “Result” uneditable Button with text “Calculate” disabled by default. And another output line that blends with background and have no text by default. (also uneditable)

I don’t need any logic in HTML, but another file calculator.js will be used for logic in future. Can you make such HTML for me?

Bing

Sure! Here’s a simple HTML file that meets your requirements:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Blockchain calculator</title>
  </head>
  <body>
    <h1>Blockchain calculator</h1>
    <input type="text" id="numberA" placeholder="Number a" />
    <select id="operation">
      <option value="" disabled selected>Select operation</option>
      <option value="+">+</option>
      <option value="-">-</option>
      <option value="*">*</option>
      <option value="/">/</option>
    </select>
    <input type="text" id="numberB" placeholder="Number b" />
    <input type="text" id="result" placeholder="Result" readonly />
    <button id="calculate" disabled>Calculate</button>
    <input type="text" id="output" style="background-color: transparent; border: none;" readonly />
    <script src="./calculator.js"></script>
  </body>
</html>
```

This HTML file creates a simple UI with a title, two input fields for numbers `a` and `b`, a dropdown for selecting a math operation, an output field for the result, a calculate button, and another output line that blends with the background. The `calculator.js` file can be used to add logic to the UI in the future.

User

ok. It looks fine, but can we make it more intresting. Like put all elements one under another. Maybe add some theme with CSS inspered by blue and white colors?

Bing

Sure! Here’s an updated version of the HTML file with some added CSS to make the UI more interesting and inspired by blue and white colors:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Blockchain calculator</title>
    <style>
      body {
        font-family: Arial, sans-serif;
        margin: 0;
        padding: 0;
      }
      .container {
        max-width: 400px;
        margin: 0 auto;
        padding: 20px;
        text-align: center;
      }
      h1 {
        color: #fff;
        background-color: #00f;
        padding: 20px;
        margin-bottom: 20px;
      }
      input,
      select,
      button {
        display: block;
        width: 100%;
        margin-bottom: 10px;
        padding: 10px;
        box-sizing: border-box;
      }
      input[readonly] {
        background-color: #eee;
      }
      button {
        background-color: #00f;
        color: #fff;
        border: none;
        cursor: pointer;
      }
      button:hover {
        background-color: #0af;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h1>Blockchain calculator</h1>
      <input type="text" id="numberA" placeholder="Number a" />
      <select id="operation">
        <option value="" disabled selected>Select operation</option>
        <option value="+">+</option>
        <option value="-">-</option>
        <option value="*">*</option>
        <option value="/">/</option>
      </select>
      <input type="text" id="numberB" placeholder="Number b" />
      <input type="text" id="result" placeholder="Result" readonly />
      <button id="calculate" disabled>Calculate</button>
      <input type="text" id="output" style="background-color: transparent; border: none;" readonly />
    </div>
    <script src="./calculator.js"></script>
  </body>
</html>
```

This updated version of the HTML file arranges all the elements one under another and adds some blue and white themed CSS to make the UI more interesting. Is there anything else you would like me to add or change?

User

fine. Can we sepparate HTML and CSS into different files?

Bing

Yes, we can separate the HTML and CSS into different files. Here's the updated HTML file `index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Blockchain calculator</title>
    <link rel="stylesheet" href="./styles.css" />
  </head>
  <body>
    <div class="container">
      <h1>Blockchain calculator</h1>
      <input type="text" id="numberA" placeholder="Number a" />
      <select id="operation">
        <option value="" disabled selected>Select operation</option>
        <option value="+">+</option>
        <option value="-">-</option>
        <option value="*">*</option>
        <option value="/">/</option>
      </select>
      <input type="text" id="numberB" placeholder="Number b" />
      <input type="text" id="result" placeholder="Result" readonly />
      <button id="calculate" disabled>Calculate</button>
      <input type="text" id="output" style="background-color: transparent; border: none;" readonly />
    </div>
    <script src="./calculator.js"></script>
  </body>
</html>
```

And here's the new CSS file `styles.css`:

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
}
.container {
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  text-align: center;
}
h1 {
  color: #fff;
  background-color: #00f;
  padding: 20px;
  margin-bottom: 20px;
}
input,
select,
button {
  display: block;
  width: 100%;
  margin-bottom: 10px;
  padding: 10px;
  box-sizing: border-box;
}
input[readonly] {
  background-color: #eee;
}
button {
  background-color: #00f;
  color: #fff;
  border: none;
  cursor: pointer;
}
button:hover {
  background-color: #0af;
}
```

Now the HTML and CSS are separated into two different files `index.html` and `styles.css`. The HTML file includes a link to the CSS file in the `head` section using the `link` tag. Is there anything else you would like me to add or change?

User

it's strange but button still looks clickable despite have disabled property

Bing

You're right, the button does appear clickable despite having the `disabled` property. This is because the CSS for the button doesn't include any styles for the `:disabled` state. Here's an updated version of the `styles.css` file that includes styles for the `:disabled` state of the button:

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
}
.container {
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  text-align: center;
}
h1 {
  color: #fff;
  background-color: #00f;
  padding: 20px;
  margin-bottom: 20px;
}
input,
select,
button {
  display: block;
  width: 100%;
  margin-bottom: 10px;
  padding: 10px;
  box-sizing: border-box;
}
input[readonly] {
  background-color: #eee;
}
button {
  background-color: #00f;
  color: #fff;
  border: none;
}
button:hover:not(:disabled) {
  background-color: #0af;
}
button:disabled {
  background-color: #aaa;
  cursor: not-allowed;
}
```

This updated version of the `styles.css` file includes styles for the `button:hover:not(:disabled)` and `button:disabled` selectors. The `button:hover:not(:disabled)` selector applies styles to the button when it is hovered over and not disabled. The `button:disabled` selector applies styles to the button when it is disabled. Now, when the button is disabled, its background color will change to a light gray color and its cursor will change to a "not-allowed" cursor to indicate that it is not clickable. Is there anything else you would like me to add or change?

User

Now it looks fine

Bing

I'm glad to hear that it looks fine now! Is there anything else you would like me to help you with?

User

Now let's make calculator.js

When a user opens the page, the site should check if the user has connected their wallet using Metamask. If metamask is connected then "Calculate" button is clickable. Also input element with id="output" must show "Calculator used: XXX" times when XXX is usageCount. usageCount can be obtained with eth contract.

By clicking "Calculate" button math for number a and b must be done with eth contract.

eth contract that I am talking about have this ABI: [{"anonymous":false,"inputs":[{"indexed":true,"internalType":"string","name":"operation","type":"string"},{"indexed":true,"internalType":"uint256","name":"a","type":"uint256"},{"indexed":true,"internalType":"uint256","name":"b","type":"uint256"},{"indexed":false,"internalType":"uint256","name":"result","type":"uint256"}],"name":"Result","type":"event"},{"inputs":[{"internalType":"uint256","name":"a","type":"uint256"},{"internalType":"uint256","name":"b","type":"uint256"}],"name":"add","outputs":[{"internalType":"uint256","name":"result","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"a","type":"uint256"},{"internalType":"uint256","name":"b","type":"uint256"}],"name":"divide","outputs":[{"internalType":"uint256","name":"result","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"a","type":"uint256"},{"internalType":"uint256","name":"b","type":"uint256"}],"name":"multiply","outputs":[{"internalType":"uint256","name":"result","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"a","type":"uint256"},{"internalType":"uint256","name":"b","type":"uint256"}],"name":"subtract","outputs":[{"internalType":"uint256","name":"result","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"usageCount","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"}]

User

you don't need to repeat ABI. Imagine that you already have constant contractABI.

Bing

Here's a simple `calculator.js` file that checks if the user has connected their wallet using Metamask and enables the "Calculate" button if Metamask is connected. It also displays the usage count of the calculator in the `output` element and performs calculations using the provided Ethereum contract when the "Calculate" button is clicked:

~~~
const contractABI = [{"anonymous":false,"inputs":[{"indexed":true,"internalType":"string","name":"operation","type":"string"},{"indexed":true,"internalType":"uint256"


<div class="user-title">User</div>

ABI string can be obtained by function getABI()


<div class="bot-title">Bing</div>

Here's an updated version of the `calculator.js` file that obtains the contract ABI using the `getABI()` function:

```javascript
async function getABI() {
  // Insert code here to fetch the contract ABI
}

async function init() {
  const contractABI = await getABI();
  const contractAddress = "YOUR_CONTRACT_ADDRESS";
  let contract;

  const numberA = document.querySelector("#numberA");
  const numberB = document.querySelector("#numberB");
  const operation = document.querySelector("#operation");
  const result = document.querySelector("#result");
  const calculate = document.querySelector("#calculate");
  const output = document.querySelector("#output");

  if (typeof window.ethereum !== "undefined") {
    try {
      await window.ethereum.enable();
      const provider = new ethers.providers.Web3Provider(window.ethereum);
      contract = new ethers.Contract(contractAddress, contractABI, provider.getSigner());
      calculate.disabled = false;
    } catch (err) {
      console.error(err);
    }
  }

  if (contract) {
    try {
      const usageCount = await contract.usageCount();
      output.value = `Calculator used: ${usageCount} times`;
    } catch (err) {
      console.error(err);
    }
  }

  calculate.addEventListener("click", async () => {
    if (!contract) return;
    try {
      const a = parseInt(numberA.value);
      const b = parseInt(numberB.value);
      let res;
      switch (operation.value) {
        case "+":
          res = await contract.add(a, b);
          break;
        case "-":
          res = await contract.subtract(a, b);
          break;
        case "*":
          res = await contract.multiply(a, b);
          break;
        case "/":
          res = await contract.divide(a, b);
          break;
        default:
          return;
      }
      result.value = res.toString();
    } catch (err) {
      console.error(err);
    }
  });
}

init();
~~~

This updated version of the `calculator.js` file uses the `getABI()` function to obtain the contract ABI. You'll need to insert your own code into the `getABI()` function to fetch the contract ABI. You'll also need to replace `YOUR_CONTRACT_ADDRESS` with the actual address of your Ethereum contract. Is there anything else you would like me to add or change?

User

I have strange behaviour. I have installed metamask plugin and "window.ethereum" is not "undefined" on any real web page. However when I run our HTML file localy it looks like "window.ethereum" is always "undefined"

Bing

The issue you're experiencing is likely due to the fact that you're opening the HTML file locally (i.e., by double-clicking on the file or using the `file://` protocol in your browser). When you open an HTML file locally, it is not being served by a web server and is therefore subject to stricter security restrictions. One of these restrictions is that the `window.ethereum` object provided by the Metamask browser extension is not available to local files.

To fix this issue, you'll need to serve your HTML file using a local web server. There are many ways to do this, but one simple way is to use the `http-server` package from npm. Here's how you can do it:

1. Install Node.js on your computer if you don't have it already. You can download it from the official website.
2. Open a command prompt or terminal and navigate to the directory where your HTML file is located.

3. Install the `http-server` package globally by running the command `npm install -g http-server`.
4. Start a local web server by running the command `http-server`.
5. Open your web browser and navigate to `http://localhost:8080`. You should now see your HTML file being served by the local web server.

Now that your HTML file is being served by a local web server, the `window.ethereum` object should be available and you should be able to interact with Metamask. Is there anything else you would like me to help you with?

User

server works well. Thank you

Bing

You're welcome! I'm glad to hear that the server is working well. Is there anything else you would like me to help you with?

User

on this line:

const provider = new ethers.providers.Web3Provider(window.ethereum);

I got an exception "ReferenceError: ethers is not defined"

Bing

The error message "ReferenceError: ethers is not defined" indicates that the `ethers` object is not available in the current scope. This is likely because the `ethers.js` library has not been included in your HTML file.

To fix this issue, you'll need to include the `ethers.js` library in your HTML file by adding a `script` tag that points to the library's CDN. Here's an updated version of the `index.html` file that includes the `ethers.js` library:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Blockchain calculator</title>
    <link rel="stylesheet" href="./styles.css" />
    <script src="https://cdn.ethers.io/lib/ethers-5.2.umd.min.js"></script>
  </head>
  <body>
    <div class="container">
      <h1>Blockchain calculator</h1>
      <input type="text" id="numberA" placeholder="Number a" />
      <select id="operation">
        <option value="" disabled selected>Select operation</option>
        <option value="+">+</option>
        <option value="-">-</option>
        <option value="*">*</option>
        <option value="/">/</option>
      </select>
      <input type="text" id="numberB" placeholder="Number b" />
      <input type="text" id="result" placeholder="Result" readonly />
      <button id="calculate" disabled>Calculate</button>
      <input type="text" id="output" style="background-color: transparent; border: none;" readonly />
    </div>
    <script src="./calculator.js"></script>
  </body>
</html>
```

This updated version of the `index.html` file includes a `script` tag in the `head` section that points to the `ethers.js` library's CDN. Now the `ethers` object should be available in the global scope and you should no longer see the "ReferenceError: ethers is not defined" error message. Is there anything else you would like me to add or change?

User

ok. Now I need to know how to add wallet with test eth in my metamask for Sepolia Testnet

Bing

To add the Sepolia Testnet to your Metamask wallet, you can follow these steps:

1. Click on the network button at the top of your Metamask wallet and click "Add Network".
2. At the bottom of the page, click "Add a network manually".
3. Enter the following information:
   - Network Name: Sepolia Testnet
   - New RPC URL: https://sepolia.infura.io/v3/YOUR-INFURA-PROJECT-ID (replace `YOUR-INFURA-PROJECT-ID` with your own Infura project ID)
   - Chain ID: 11155111
   - Currency Symbol: SepoliaETH
   - Block Explorer URL: https://sepolia.etherscan.io/

After adding the Sepolia Testnet to your Metamask wallet, you can switch to it by clicking on the network button at the top of your wallet and selecting "Sepolia Testnet" from the list of available networks.

To obtain test ETH for the Sepolia Testnet, you can use a faucet such as the Chainlink faucet. Simply enter your Sepolia Testnet address and click on the "Send me 100 Test SepoliaETH" button to receive 100 test ETH.

Is there anything else you would like me to help you with?

User

what is YOUR-INFURA-PROJECT-ID and where I can get it?

Bing

`YOUR-INFURA-PROJECT-ID` is a placeholder for the ID of your Infura project. Infura is a service that provides access to Ethereum and IPFS networks via a set of APIs. To obtain an Infura project ID, you'll need to create an account on the Infura website and create a new project. Here's how you can do it:

1. Go to the Infura website and click on the “Sign up” button to create a new account.
2. After creating an account and logging in, click on the “Create New Project” button.
3. Enter a name for your project and click on the “Create” button.
4. Your new project will be created and you’ll be taken to the project dashboard. In the “Keys” section, you’ll find your project ID.

Once you have obtained your Infura project ID, you can replace `YOUR-INFURA-PROJECT-ID` in the Sepolia Testnet RPC URL with your own project ID. Is there anything else you would like me to help you with?

User

looks like I connected to Sopolia

User

I connected to Sepolia

Bing

Great! I’m glad to hear that you were able to connect to the Sepolia Testnet. Now that you’re connected, you can interact with smart contracts deployed on the Sepolia Testnet using your Metamask wallet. Is there anything else you would like me to help you with?

User

I have this instruction:

For testing, you can use the following wallet (with some amount of test eth):1433ce1ec08c988b9631e8f63817e0d102eab4f7d4933328fe91e7673970019a

Can I add wallet to metamask?

Bing

Yes, you can import an existing wallet into Metamask using its private key. The string you provided appears to be a private key, so you can use it to import the wallet into Metamask. Here’s how you can do it:

1. Open the Metamask browser extension and click on the circle icon in the top-right corner to open the account menu.
2. Click on the “Import Account” option.
3. Select “Private Key” as the type of account you want to import.
4. Paste the private key into the text field and click on the “Import” button.

After importing the wallet, it will be added to your list of accounts in Metamask and you’ll be able to use it to interact with the Ethereum blockchain. Please note that you should never share your private key with anyone, as it gives full control over your wallet and its funds.

Is there anything else you would like me to help you with?

User

ok, now I can test this.

I tried 2 + 2 , but as result I get “[object Object]”

Bing

The issue you’re experiencing is likely due to the fact that the result of the contract method call is being returned as a BigNumber object rather than a regular JavaScript number. The `[object Object]` string you’re seeing is the result of calling the `toString()` method on a BigNumber object.

To fix this issue, you’ll need to convert the BigNumber object to a regular JavaScript number before displaying it in the result field. Here’s an updated version of the `calculator.js` file that includes this fix:

```
async function getABI() {
  // Insert code here to fetch the contract ABI
}

async function init() {
  const contractABI = await getABI();
  const contractAddress = "YOUR_CONTRACT_ADDRESS";
  let contract;

  const numberA = document.querySelector("#numberA");
  const numberB = document.querySelector("#numberB");
  const operation = document.querySelector("#operation");
  const result = document.querySelector("#result");
  const calculate = document.querySelector("#calculate");
  const output = document.querySelector("#output");

  if (typeof window.ethereum !== "undefined") {
    try {
      await window.ethereum.enable();
      const provider = new ethers.providers.Web3Provider(window.ethereum);
      contract = new ethers.Contract(contractAddress, contractABI, provider.getSigner());
      calculate.disabled = false;
    } catch (err) {
      console.error(err);
    }
  }

  if (contract) {
    try {
      const usageCount = await contract.usageCount();
      output.value = `Calculator used: ${usageCount} times`;
    } catch (err) {
      console.error(err);
    }
  }

  calculate.addEventListener("click", async () => {
    if (!contract) return;
    try {
      const a = parseInt(numberA.value);
      const b = parseInt(numberB.value);
      let res;
      switch (operation.value) {
        case "+":
          res = await contract.add(a, b);
          break;
        case "-":
          res = await contract.subtract(a, b);
          break;
        case "*":
          res = await contract.multiply(a, b);
          break;
        case "/":
          res = await contract.divide(a, b);
          break;
        default:
          return;
      }
      result.value = res.toNumber().toString();
    } catch (err) {
      console.error(err);
    }
  });
}

init();
```

This updated version of the `calculator.js` file converts the result of the contract method call to a regular JavaScript number using the `toNumber()` method before displaying it in the result field. Now, when you perform a calculation such as `2 + 2`, you should see the correct result `4` in the result field. Is there anything else you would like me to add or change?

User

actually “res” is something like this:

{“hash”:“0xcda3ab4f4a0b7ca272fa52bbc9973c42c41d356042110cea99e8a7a2223d58c3”,“type”:2,“accessList”:null,“blockHash”:null,“blockNumber”:null,“transactionIndex”:null,“con
{“type”:“BigNumber”,“hex”:“0x59682f0b”},“gasLimit”:{“type”:“BigNumber”,“hex”:“0x75c6”},“to”:“0x1851ffBce02A134eFd9ddBC91920b0c6DCEfB6f5”,“value”:
{“type”:“BigNumber”,“hex”:“0x00”},“nonce”:14,“data”:“0x771602f7000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000

Bing

It looks like the value of `res` is a transaction object rather than the result of the contract method call. This is likely because the contract methods `add`, `subtract`, `multiply`, and `divide` are non-constant methods that modify the state of the contract and therefore return a transaction object when called.

To obtain the result of the calculation, you’ll need to wait for the transaction to be mined and then retrieve the result from the contract’s `Result` event. Here’s an updated version of the `calculator.js` file that includes this fix:

```
async function getABI() {
  // Insert code here to fetch the contract ABI
}

async function init() {
  const contractABI = await getABI();
  const contractAddress = "YOUR_CONTRACT_ADDRESS";
  let contract;

  const numberA = document.querySelector("#numberA");
  const numberB = document.querySelector("#numberB");
  const operation = document.querySelector("#operation");
  const result = document.querySelector("#result");
  const calculate = document.querySelector("#calculate");
  const output = document.querySelector("#output");

  if (typeof window.ethereum !== "undefined") {
    try {
      await window.ethereum.enable();
      const provider = new ethers.providers.Web3Provider(window.ethereum);
      contract = new ethers.Contract(contractAddress, contractABI, provider.getSigner());
      calculate.disabled = false;
    } catch (err) {
      console.error(err);
    }
  }

  if (contract) {
    try {
      const usageCount = await contract.usageCount();
      output.value = `Calculator used: ${usageCount} times`;
    } catch (err) {
      console.error(err);
    }
  }

  calculate.addEventListener("click", async () => {
    if (!contract) return;
    try {
      const a = parseInt(numberA.value);
      const b = parseInt(numberB.value);
      let tx;
      switch (operation.value) {
        case "+":
          tx = await contract.add(a, b);
          break;
        case "-":
          tx = await contract.subtract(a, b);
          break;
        case "*":
          tx = await contract.multiply(a, b);
          break;
        case "/":
          tx = await contract.divide(a, b);
          break;
        default:
          return;
      }
      const receipt = await tx.wait();
      const resEvent = receipt.events.find(event => event.event === "Result");
      if (resEvent) {
        result.value = resEvent.args.result.toString();
      }
    } catch (err) {
      console.error(err);
    }
  });
}

init();
```

This updated version of the `calculator.js` file waits for the transaction to be mined using the `wait()` method and then retrieves the result of the calculation from the contract's `Result` event. Now, when you perform a calculation such as `2 + 2`, you should see the correct result `4` in the result field. Is there anything else you would like me to add or change?

User

Let's talk about just this chunk of code:

const receipt = await tx.wait(); const resEvent = receipt.events.find(event => event.event === "Result"); if (resEvent) { result.value = resEvent.args.result.toString(); }

no needs to rewrite anything else

Bing

Sure! This chunk of code is responsible for waiting for the transaction to be mined and retrieving the result of the calculation from the contract's `Result` event.

The first line `const receipt = await tx.wait();` waits for the transaction to be mined and returns the transaction receipt. The `wait()` method returns a promise that resolves with the transaction receipt once the transaction has been mined.

The second line `const resEvent = receipt.events.find(event => event.event === "Result");` searches the `events` array of the transaction receipt for an event with the name `Result`. The `find()` method returns the first element in the array that satisfies the provided testing function. In this case, the testing function checks if the `event` property of each element is equal to `"Result"`.

The third and fourth lines check if a `Result` event was found and, if so, sets the value of the result field to the value of the `result` property of the event's `args` object. The `args` object contains the indexed and non-indexed arguments of the event. In this case, it contains a single property named `result` which holds the result of the calculation.

This code assumes that your Ethereum contract emits a `Result` event when a calculation is performed. The `Result` event should have an indexed argument named `result` that holds the result of the calculation.

Is there anything else you would like me to explain or help you with?

User

When I run something like this

JSON.stringify(resEvent);

then for 2 + 2 example I got this as response:
{"transactionIndex":9,"blockNumber":3637627,"transactionHash":"0x4562b5fe670accdf54026f30fbbb426217844308c9720da6a2a5c226754570dd","address":"0x1851ffBce02A134eF
["0xd01bdea216b31f8631820102c669d84905fdbab57133ddb1ee47699ba4c6cb93","0x4f5a8bb8492337e79bdc674d6f31ac448f8017e26cc7bfe3144fb5d886fe5369","0x0000000000000
[{"_isIndexed":true,"hash":"0x4f5a8bb8492337e79bdc674d6f31ac448f8017e26cc7bfe3144fb5d886fe5369"},{"type":"BigNumber","hex":"0x02"},
{"type":"BigNumber","hex":"0x02"},{"type":"BigNumber","hex":"0x04"}],"event":"Result","eventSignature":"Result(string,uint256,uint256,uint256)"}

this line will not provide a result: result.value = resEvent.args.result.toString();

Bing

Based on the response you provided, it looks like the `Result` event has four indexed arguments: `operation`, `a`, `b`, and `result`. The `args` property of the `resEvent` object is an array that contains the values of these indexed arguments in the order they were emitted by the contract.

To access the value of the `result` argument, you'll need to use the appropriate index in the `args` array. In this case, it looks like the `result` argument is the fourth indexed argument, so you can access its value using the index `3`. Here's an updated version of the code that uses the correct index to access the value of the `result` argument:

```
const receipt = await tx.wait();
const resEvent = receipt.events.find(event => event.event === "Result");
if (resEvent) {
  result.value = resEvent.args[3].toString();
}
```

This updated version of the code uses the index `3` to access the value of the `result` argument in the `args` array. Now, when you perform a calculation such as `2 + 2`, you should see the correct result `4` in the result field. Is there anything else you would like me to add or change?

User

ok. This works.

Bing

Great! I'm glad to hear that it's working now. Is there anything else you would like me to help you with?

User

Just add readme file for all of it. Also mention in it that 99% code was writen by you. And it is test task for junior role in Ukraine.

Bing

Sure! Here's a simple `README.md` file that provides an overview of the project and credits me for writing most of the code:

```
# Blockchain Calculator

This is a simple blockchain calculator that allows users to perform basic arithmetic operations using an Ethereum smart contract. The calculator's user interface is b

## Usage

To use the calculator, open the `index.html` file in your web browser. Make sure you have the Metamask browser extension installed and connected to the Sepolia Testne

## Credits

99% of the code for this project was written by an AI language model from OpenAI as part of a test task for a junior role in Ukraine.
```

This `README.md` file provides a brief overview of the project and its usage, and credits me for writing most of the code. You can include this file in your project's root directory to provide users with information about the project. Is there anything else you would like me to add or change?