through Web3.js (Tutorial)

Yang Nana Follow
Mar 22, 2018 · 7 min read

My OS is Mac OS High Sierra Version 10.13.3 (17D47)

Source: https://coursetro.com/posts/code/99/Interacting-with-a-Smart-Contract-through-Web3.js-(Tutorial)

Installing & Running the Ethereum TestRPC

The Ganache is a Node.js Ethereum client for the testing and developing smart contracts. Because it's based on Node.js, we need Node.js installed along with NPM (Node Package Manager) to install it.

Open up your command line or console and run the following 2 commands:

```
> node -v
> npm -v
```

(my node version is v9.7.1 and my npm version is 5.6.0)

If either of these commands go unrecognized, visit Nodejs.org and download the appropriate installer. Run it through all of the default options.

Once finished, close and reload your console and re-run the commands above. They should now provide you with version numbers.

Next, let's use NPM to install the Ganache:

```
> npm install -g ganache-cli
```

(My ganache-cli version is Ganache CLI v6.1.0 (ganache-core: 2.1.0))

Once finished, run the following command to start it:

```
> ganache-cli
```

This provides you with 10 different accounts and private keys, along with a local server at localhost:8545.

Installing Web3.js

Web3.js is the official Ethereum Javascript API. You use it to interact with your Ethereum smart contracts.

Before we can install it, let’s create a project folder in a new console window:

```
> mkdir coursetro-eth
> cd coursetro-eth
```

Next, run the npm init command to create a package.json file, which will store project dependencies:

```
> npm init
```

Hit enter through all of the prompts. Next, run the following command to install web3.js:

```
> npm install ethereum/web3.js --save
```

Changing the Environment in Remix

Switch over to the **Remix IDE,** click on the Run tab, and then change the Environment dropdown from Javascript VM to Web3 Provider.

Hit "OK" and then specify the testrpc localhost address (by default, it's **http://localhost:8545**)

This means that instead of deploying and testing in the Javascript VM, we're now using the Ganache client on your computer.

If you haven't been following along since the previous lesson, paste in this contract in a new solidity file called "Coursetro.sol":

```
pragma solidity ^0.4.18;

contract Coursetro {

   string fName;
   uint age;

   function setInstructor(string _fName, uint _age) public {
       fName = _fName;
       age = _age;
   }

   function getInstructor() public constant returns (string,
uint) {
       return (fName, age);
   }

}
```

Hit **Create**. We will need the address of this contract shortly, so leave this window open.

Creating the UI

Open up your preferred code editor (I use Visual Studio Code) with the project folder we created. Here, you'll notice a node_modules folder, which includes web3 that we installed via npm earlier.

Let's create an index.html in the project folder.

We're not going to create anything too fancy in terms of a UI, but we'll have some limited CSS, and a UI that consists of a place that retrieves the Instructor's name and age from the **getInstructor()** function, and a form with 2 input fields for a name and age, which will be set via jQuery from 2 input textfields.

To get started, paste the following contents into the empty index.html file:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>

    <link rel="stylesheet" type="text/css" href="main.css">

    <script src="node_modules/web3/dist/web3.min.js">
</script>

</head>
<body>
    <div class="container">

        <h1>Coursetro Instructor</h1>

        <h2 id="instructor"></h2>

        <label for="name" class="col-lg-2 control-
label">Instructor Name</label>
        <input id="name" type="text">

        <label for="name" class="col-lg-2 control-
label">Instructor Age</label>
        <input id="age" type="text">

        <button id="button">Update Instructor</button>


    </div>

    <script src="https://code.jquery.com/jquery-
```

```
3.2.1.slim.min.js"></script>

    <script>
       // Our future code here..
    </script>

</body>
</html>
```

As you can see, we're referencing a **main.css** file, so create that file and paste in the following rulesets real quickly:

```
body {
    background-color:#F0F0F0;
    padding: 2em;
    font-family: 'Raleway','Source Sans Pro', 'Arial';
}
.container {
    width: 50%;
    margin: 0 auto;
}
label {
    display:block;
    margin-bottom:10px;
}
input {
    padding:10px;
    width: 50%;
    margin-bottom: 1em;
}
button {
    margin: 2em 0;
    padding: 1em 4em;
    display:block;
}

#instructor {
    padding:1em;
    background-color:#fff;
    margin: 1em 0;
}
```

Save it.

Using Web3.js to Connect & Interact with the Smart Contract

Going back to the index.html, at the bottom of the file we have an empty **<script>** tag. This is where we will write the necessary code to work with our smart contract.

Normally I would never use jQuery (I'm a big Angular fan), but this keeps things more simple.

In the head tags, we're already importing the Web3.js library, so now, let's use it to connect to our testrpc client:

```
<script>

        if (typeof web3 !== 'undefined') {
            web3 = new Web3(web3.currentProvider);
        } else {
            // set the provider you want from Web3.providers
            web3 = new Web3(new
Web3.providers.HttpProvider("http://localhost:8545"));
        }

    </script>
```

This code comes directly from the **Web3.js Github** page.

It's saying that if web3 is not undefined, then we'll use that as our provider. If it's undefined (else), we can manually specify the provider ourselves.

You may be wondering, how would web3 be defined? Well, if you're using the Chrome extension **MetaMask**(which we will use later in this course) or an Ethereum browser like **Mist**, the provider is automatically injected.

Next, we have to specify a default ethereum account to use through the **web3.eth.defaultAccount** method:

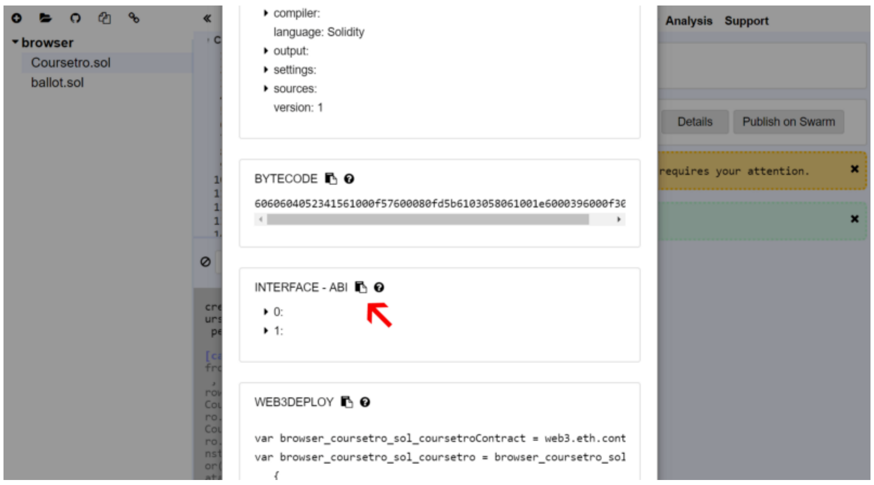
```
<script>

        // Previous if/else statement removed for brevity

        web3.eth.defaultAccount = web3.eth.accounts[0];
        </script>
```

Remember when we ran the **ganache** console command? It provided us with 10 accounts. We're simply choosing the first account here to use.

Next, we need to use the **web3.eth.contract()** method to initiatlize (or create) the contract on an address. It accepts one parameter, which is referred to as the ABI (Application Binary Interface).

This ABI allows you to call functions and receive data from your smart contract.

If you switch back to the Remix IDE, click on the **Compile** tab and click **Details**. Scroll down until you see the **Interface—ABI** section and click the copy icon as shown below:



```
compiler:
language: Solidity
output:
settings:
sources:
version: 1
```

```
6060604052341561000f57600080fd5b6103058061001e6000396000f3
```

requires your attention.

```
0:
1:
```

```
var browser_coursetro_sol_coursetroContract = web3.eth.cont
var browser_coursetro_sol_coursetro = browser_coursetro_sol
   {
```

Going back to **index.html** paste the following code:

```
<script>

        // Previous if/else statement removed for brevity

        web3.eth.defaultAccount = web3.eth.accounts[0];

        var CoursetroContract = web3.eth.contract(PASTE ABI
HERE!);
```

```
    </script>
```

Great. Now that we have the interface for interacting with our contract through the **CoursetroContract** variable, the last thing to do is to define the actual contract address.

We used Remix to create the contract earlier, and it has an associated address.

Go back to Remix and click the **Run** tab, and click on the copy icon next to the contract that we created earlier on the right column.

Back in **index.html** add the following line:

```
<script>

        // Previous if/else statement removed for brevity

        web3.eth.defaultAccount = web3.eth.accounts[0];

        var CoursetroContract = web3.eth.contract(YOUR ABI);

        var Coursetro = CoursetroContract.at('PASTE CONTRACT
ADDRESS HERE');
        console.log(Coursetro);

    </script>
```

Great. Let's save this, and then (in Visual Studio Code) you can right-click on the **index.html** and Reveal in Explorer. Double click the index.html to run it in the browser.

CTRL-SHIFT-I (i) will show the console. You will see something similar to the following:

Notice our 2 functions! **getInstructor** and **setInstructor**.

If you want, in the console window within the inspector, you can type:

```
> Coursetro.setInstructor('Brutis', 44) // Hit Enter
"0x894..."                            // This is the response
(address)

> Coursetro.getInstructor()          // Hit Enter
(2) ["brutis", e]                    // An array containing
our data
```

Awesome! But let's use jQuery to make these calls for us based on our form:

```
<script>

        // Previous code removed for brevity

        Coursetro.getInstructor(function(error, result){
            if(!error)
                {
                    $("#instructor").html(result[0]+'
('+result[1]+' years old)');
                    console.log(result);
                }
            else
                console.error(error);
        });

        $("#button").click(function() {
            Coursetro.setInstructor($("#name").val(),
$("#age").val());
        });
```

```
    </script>
```

We’re simply calling **.getInstructor** and passing the error and result through a callback function. If the error isn’t present, we set the html of an h2 element with the id of **#instructor** to the returned result array (0 = the name, 1 = the age).

Next, on click, we call **.setInstructor** to the name and age values from the input fields in the form.

Save it, refresh and give it a go!

Here is the whole index.html file (including my example data)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>

<link rel="stylesheet" type="text/css" href="main.css">

<script src="node_modules/web3/dist/web3.min.js"></script>

</head>
<body>
    <div class="container">

<h1>Coursetro Instructor</h1>

<h2 id="instructor"></h2>

<label for="name" class="col-lg-2 control-label">Instructor
Name</label>
        <input id="name" type="text">

<label for="name" class="col-lg-2 control-label">Instructor
Age</label>
        <input id="age" type="text">

<button id="button">Update Instructor</button>

</div>
```

```
<script src="https://code.jquery.com/jquery-
3.2.1.slim.min.js"></script>


<script>


if (typeof web3 !== 'undefined') {
            web3 = new Web3(web3.currentProvider);
        } else {
            // set the provider you want from Web3.providers
            web3 = new Web3(new
Web3.providers.HttpProvider("http://localhost:8545"));
        }

  web3.eth.defaultAccount = web3.eth.accounts[0];
  var CoursetroContract = web3.eth.contract([ { "constant":
false, "inputs": [ { "name": "_fName", "type": "string" }, {
"name": "_age", "type": "uint256" } ], "name":
"setInstructor", "outputs": [], "payable": false,
"stateMutability": "nonpayable", "type": "function" }, {
"constant": true, "inputs": [], "name": "getInstructor",
"outputs": [ { "name": "", "type": "string" }, { "name": "",
"type": "uint256" } ], "payable": false, "stateMutability":
"view", "type": "function" } ]);


var Coursetro =
CoursetroContract.at('0x7c74fa5e63b9599550131fc921c1f2748260
4236');
  //console.log(Coursetro);

  Coursetro.getInstructor(function(error, result){
            if(!error)
                {
                    $("#instructor").html(result[0]+'
('+result[1]+' years old)');
                    console.log(result);
                }
            else
                console.error(error);
        });


$("#button").click(function() {
            Coursetro.setInstructor($("#name").val(),
$("#age").val());
        });


</script>


</body>
</html>
```