Run private Ethereum network with React interface

In this guide, I will show the steps to run a Ethereum private network on your computer, install smart contract and interact with them via a React interface.

The document is based on two tutorials:

How to develop on Ethereum using Remix, Metamask and React:

https://levelup.gitconnected.com/https-medium-com-zubairnahmed-react-ethereum-getting-started-with-the-minimum-toolset-required-part-1-of-4-9562efa23d18

Set up of the private Ethereum network

https://medium.com/mercuryprotocol/how-to-create-your-own-private-ethereum-blockchain-dad6af82fc9f

The document have 4 parts:

1. Installation of the tools
2. Genesis block definition and Ethereum node start up
3. Connect Remix and metamask and deploy smart contract
4. Use smart contract via React

1. Installation of the tools

The tools necessary are :

Tool	Description	Where	Version at document creation
getch	CLI for Ethereum network	https://www.ethereum.org/cli	1.8.13-stable
remix	Online IDE to develop solidity	http://remix.ethereum.org	
metamask	Browser extension for Ethereum dApps and wallets mangement tool	https://metamask.io/	4.9.2
Google Chrome	Internet browser (used for remix and metamask)	https://www.google.com/chrome/	Version 67.0.3396.99 (Official Build) (64-bit)
MyEtherWallet	Online tool to generate Ethereum wallet amd downloadable UTC wallet	https://www.myetherwallet.com/#generate-wallet	3.21.21
React	User interface JS library	https://reactjs.org/	1.5.2

Remark : Be sure to connected on remix HTTP and not HTTPS, otherwise you will have error while trying to connect to your local network.

Please follow instructions to install the tools as given by official websites.

Create 2 Ethereum wallets on MyEtherWallet, note down your passphrases, the private keys in a secure place and download the corresponding UTC files.

2. Genesis block definition and Ethereum node start up

A. Genesis block

In this chapter I will provide the steps to create the Genesis block and start up your private Ethereum node on your computer.

1. Create a directory to hold your network files and go it
2. Create your genesis file, called myGenesis in this guide
3. Paste the following code

   And replace FIRST_WALLET_ADDRESS and SECOND_WALLET_ADDRESS by the public addresses (like e2148925f4a356bfb52d2047698f329c48ae5846) of the wallets created in first section of this document. (The second wallet will be used to start a second node on the network).

   ```
   {
   ```

```
    "config": {
       "chainId": 1994,
       "homesteadBlock": 0,
       "eip155Block": 0,
       "eip158Block": 0,
       "byzantiumBlock": 0
    },
    "difficulty": "400",
    "gasLimit": "2000000",
    "alloc": {
       "FIRST_WALLET_ADDRESS": {
           "balance": "100000000000000000000000"
       },
       "SECOND_WALLET_ADDRESS": {
           "balance": "120000000000000000000000"
       }
    }
 }
```

config

- chainId — this is your chain's identifier, and is used in replay protection.
- homesteadBlock, eip155Block, eip158Block, byzantiumBlock — these relate to chain forking and versioning, so in our case lets leave them 0 since we're starting a new blockchain.

difficulty

This dictates how difficult it is to mine a block. Setting this value low (~10–10000) is helpful in a private blockchain as it lets you mine blocks quickly, which equals fast transactions, and plenty of ETH to test with. For comparison, the Ethereum mainnet Genesis file defines a difficulty of 17179869184.

gasLimit

This is the the total amount of gas that can be used in each block. With such a low mining difficulty, blocks will be moving pretty quick, but you should still set this value pretty high to avoid hitting the limit and slowing down your network.

alloc

Here you can allocate ETH to specific addresses. This won't create the account for you, so make sure its an account you already have control of. You will need to add the account to your private chain in order to use it, and to do that you need access to the keystore/utc file. For example, Geth and MyEtherWalletgive you access to this file when you create an account, but Metamask and Coinbase do not. Here we allocate 100,000 and 120,000 ETH respectively.

B. Start the node

Instantiate the data directory:

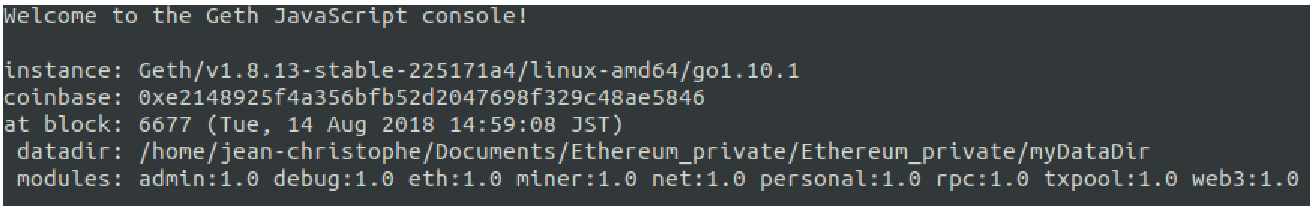
`geth --datadir ./myDataDir init ./myGenesis.json`

Start the Ethereum node

```
geth --rpc --rpcport "8545" --rpcapi "db,eth,net,web3,personal,web3,debug" --rpccorsdomain="*" --datadir ./myDataDir --networkid 1994 console 2>> myEth.log
```

These parameters will allow us to connect via remix later, note that the networkid parameter here is important and must be the same as the the chainId provided in the genesis block.

If the node started correctly, the ouput is like this:



```
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.13-stable-225171a4/linux-amd64/go1.10.1
coinbase: 0xe2148925f4a356bfb52d2047698f329c48ae5846
at block: 6677 (Tue, 14 Aug 2018 14:59:08 JST)
 datadir: /home/jean-christophe/Documents/Ethereum_private/Ethereum_private/myDataDir
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

You can display the logs on another terminal:

in folder my-eth-chain: `tail -f myEth.log`

Import wallets

Put the UTC files of the two wallets created in the first chapter in the `myDataDir/keystore` directory.

Set default wallet

Check default wallet: eth.coinbase

If the address is the same that one of the wallet imported it's ok, if not :

`miner.setEtherbase(web3.eth.accounts[0])`

Start mining

Check your balance : `eth.getBalance(eth.coinbase)`

Run miner : `miner.start()`

Stop miner : `miner.stop()`

Add other peers

Instantiate new data dir, using the same genesis block:

`geth --datadir ./peer2DataDir init ./myGenesis.json`

launch the peer on a different port:

`geth --datadir ./peer2DataDir --networkid 1994 --port 30304 console 2>> myEth2.log`

Join the first peer

On the console of the first peer : `admin.nodeInfo.enode`

The ouput will look like this :

```
“enode://dcff6d9dcb14eeb1d1b7575b0653fa1025ad1b7722c6d652d0449f0966e97931bdf037e5542086e7b9e0bec056566522c6c0cc4d73d8e4186a35da8aa5988e15@[::]:30303”
```

On the second peer:

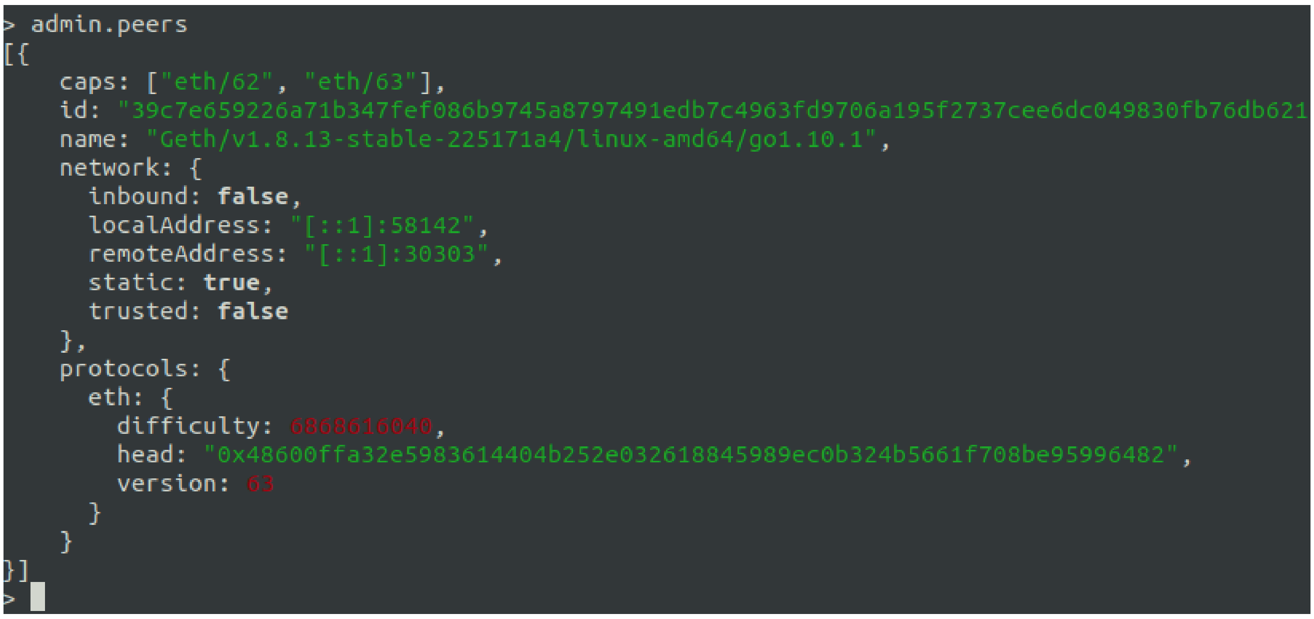
```
admin.addPeer(
“enode://b56882b93f4f97dd69f98c1dca1bd751c72374b5a8b5852288a56059a8cbc63614afd57274cfa7695178
cf292aad9e682f0117044f9a28c9e098da688c8dfd89@127.0.0.1:30303” )
```

Make sure you replace “enode://...@” above with the output from admin.nodeInfo.enode which is specific to you. As shown above, the “[::]” is replaced with “127.0.0.1:30303” which is the IP:Port of the 1st peer.

Verify the peers are communicating

On the second peer : `admin.peers`

Output should show that peer 2 is connected to 127.0.0.1:30303:



```
> admin.peers
[{
    caps: ["eth/62", "eth/63"],
    id: "39c7e659226a71b347fef086b9745a8797491edb7c4963fd9706a195f2737cee6dc049830fb76db621
    name: "Geth/v1.8.13-stable-225171a4/linux-amd64/go1.10.1",
    network: {
      inbound: false,
      localAddress: "[::1]:58142",
      remoteAddress: "[::1]:30303",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 6868616040,
        head: "0x48600ffa32e5983614404b252e032618845989ec0b324b5661f708be95996482",
        version: 63
      }
    }
}]
> 
```

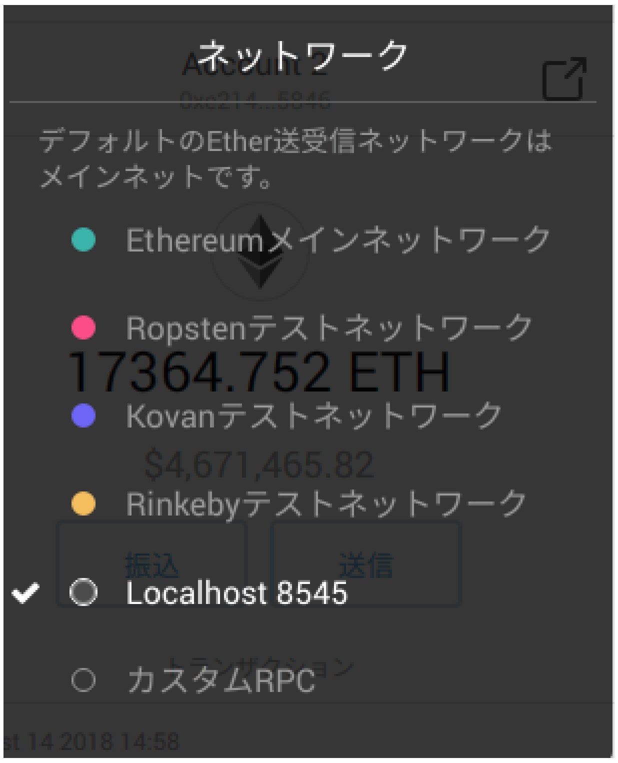
3. Connect Remix and metamask and deploy smart contract

Metamask set up

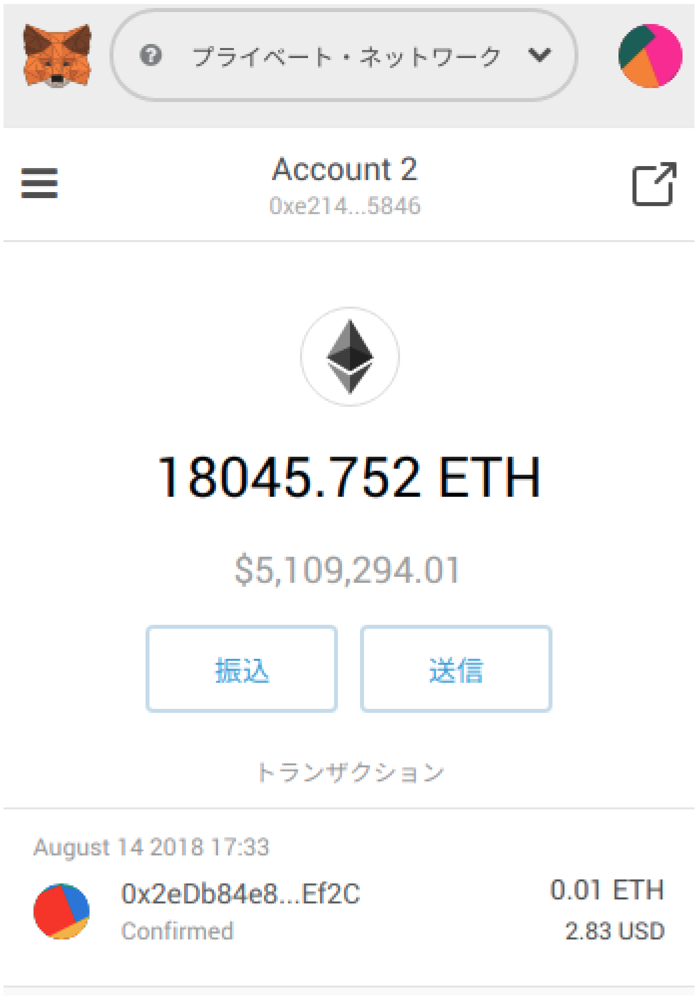
Click on the metamask icon on your browser.

Add the account of your first wallet, by providing the private key.

Then choose the network local to connect to your private Ethereum node "Localhost 8545" :


ネットワーク
デフォルトのEther送受信ネットワークはメインネットです。
Ethereumメインネットワーク
Ropstenテストネットワーク
Kovanテストネットワーク
Rinkebyテストネットワーク
✓ Localhost 8545
カスタムRPC

You should see your fund appear in your wallet.



Remix set up

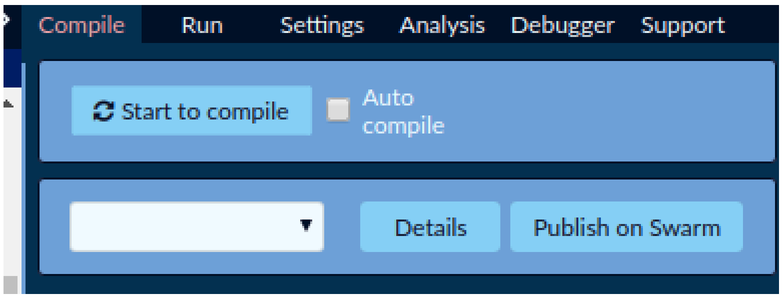
Connect on http://remix.ethereum.org

Create a file that we will call ReactExample.sol and paste the following :

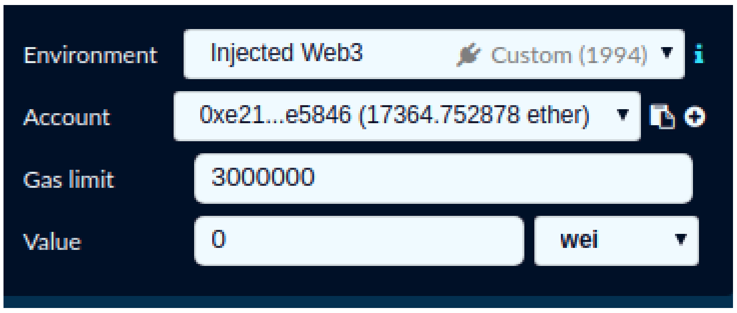
```
pragma solidity ^0.4.11;

contract ReactExample {
    address private owner;
    string public you_awesome;
    string private secret;
    string private state;
    bool public pseudoRandomResult;
    event ExperimentComplete (bool result);

constructor() public {   // constructor of the contract
    owner = msg.sender;
    you_awesome = "You're awesome";
    secret = "secret data";
    state = "initial_state";
}

function getSecret () public view returns (string) {
    return secret;
}

function getState() public view returns (string) {
    return state;
}

function setState (string newState) public payable {
    state = newState;
}

function kill () public {       // destruct smart contract
    require (msg.sender == owner);  // error if false
    selfdestruct(owner);
}

function setExperimentInMotion () public payable returns (bool) {
    bytes32 _pseudoRandomResult = keccak256 (msg.sender, msg.data, msg.value);
    if (_pseudoRandomResult > bytes32(10)) pseudoRandomResult = true;
    else pseudoRandomResult = false;

    emit ExperimentComplete (pseudoRandomResult);
}

function () public payable {        // fall back function
    revert ();
}
}
```

This is our smart contract with several variables and functions.

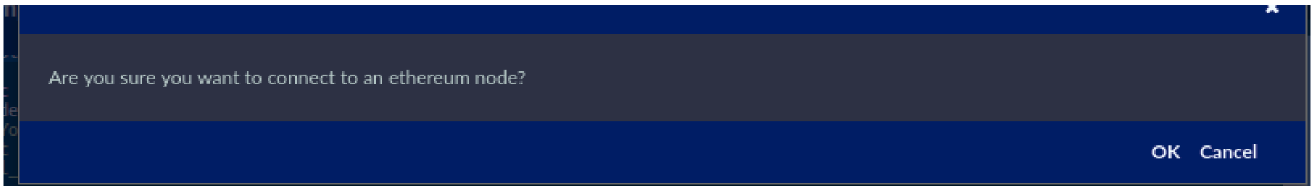
You can click on `Start to Compile:`



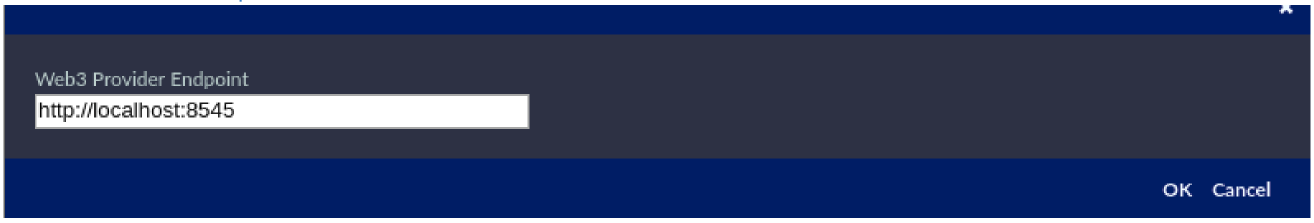
Then go on the Run tab and select Web3 Provider :


Environment: Injected Web3 — Custom (1994)
Account: 0xe21...e5846 (17364.752878 ether)
Gas limit: 3000000
Value: 0 wei

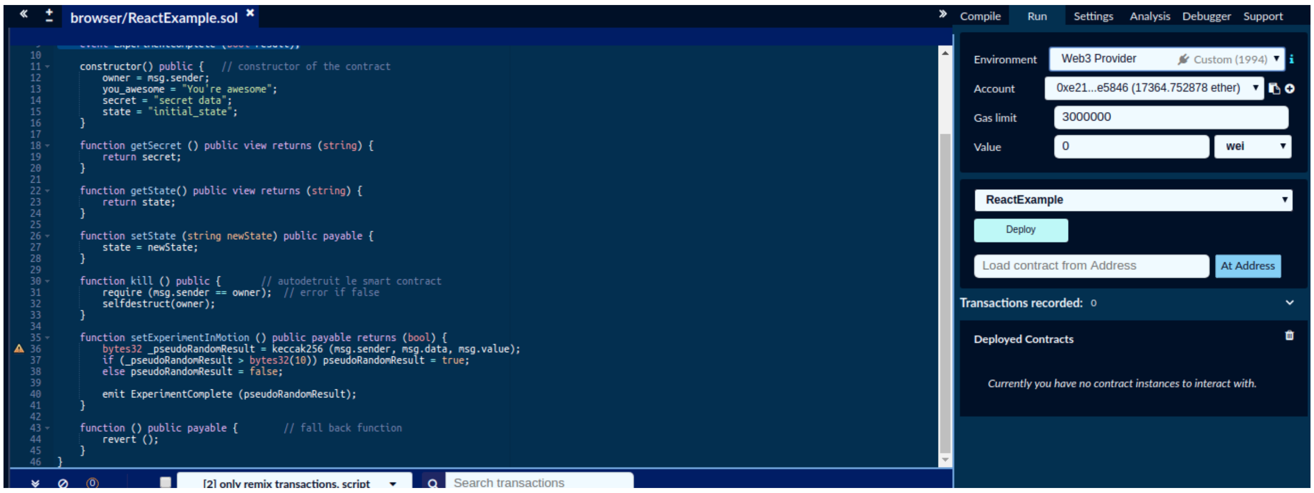
Click OK :


Are you sure you want to connect to an ethereum node?

This is our node http://localhost:8545, click OK:


Web3 Provider Endpoint: http://localhost:8545

Remix should look like this (or other color depending of your theme):



```
    constructor() public {   // constructor of the contract
        owner = msg.sender;
        you_awesome = "You're awesome";
        secret = "secret data";
        state = "initial_state";
    }

    function getSecret () public view returns (string) {
        return secret;
    }

    function getState() public view returns (string) {
        return state;
    }

    function setState (string newState) public payable {
        state = newState;
    }

    function kill () public {       // autodetruit le smart contract
        require (msg.sender == owner);  // error if false
        selfdestruct(owner);
    }

    function setExperimentInMotion () public payable returns (bool) {
        bytes32 _pseudoRandomResult = keccak256 (msg.sender, msg.data, msg.value);
        if (_pseudoRandomResult > bytes32(10)) pseudoRandomResult = true;
        else pseudoRandomResult = false;

        emit ExperimentComplete (pseudoRandomResult);
    }

    function () public payable {        // fall back function
        revert ();
    }
}
```

Environment: Web3 Provider — Custom (1994)
Account: 0xe21...e5846 (17364.752878 ether)
Gas limit: 3000000
Value: 0 wei

ReactExample

Deploy

Load contract from Address — At Address

Transactions recorded: 0

Deployed Contracts

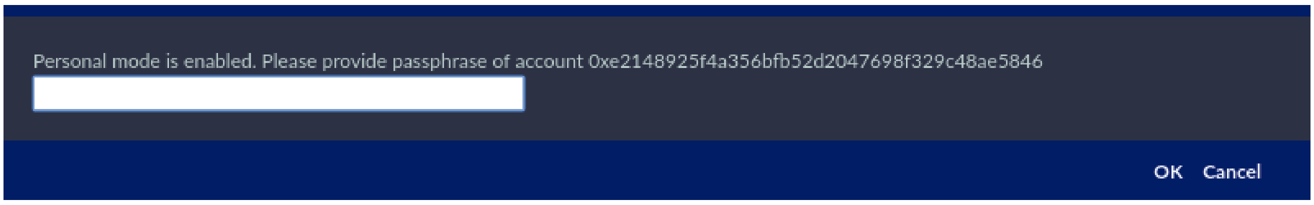
Currently you have no contract instances to interact with.

[2] only remix transactions, script

Search transactions

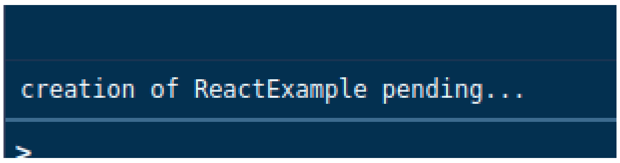
you can now click on Deploy

Provide the passphrase you used to initiate your wallet (password of MyEtherWallet) during Chapter 1 and click OK :



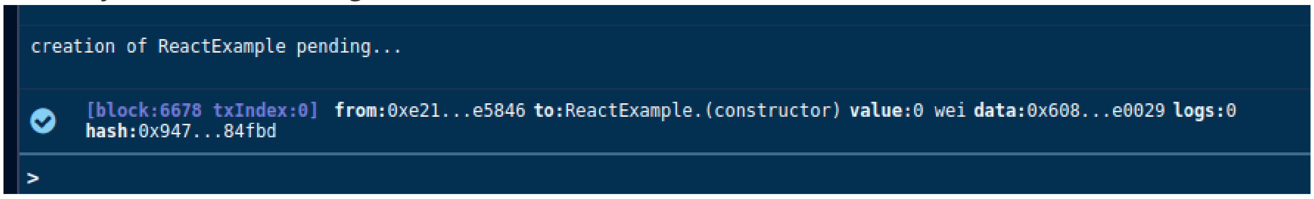
Personal mode is enabled. Please provide passphrase of account 0xe2148925f4a356bfb52d2047698f329c48ae5846

Wait during smart contract creation :



```
creation of ReactExample pending...
>
```

Be sure your miner is running, when done it should look like this :



```
creation of ReactExample pending...

[block:6678 txIndex:0] from:0xe21...e5846 to:ReactExample.(constructor) value:0 wei data:0x608...e0029 logs:0
hash:0x947...84fbd
>
```

4. Use smart contract via React

In this example we will deploy a react app and use functions of our smart contract deployed before.

For full understanding please refer to the tutorial : https://levelup.gitconnected.com/react-ethereum-getting-started-with-the-minimum-toolset-required-part-2-of-4-ad4d258ebe53

Install create-react-app

`npm install -g create-react-app`

Create React application

`create-react-app react_ethereum_private`

Go in the directory : `cd react_ethereum_private`

Start the react app

`npm run start`

Code the react app

Open your favorite IDE or text editor and paste the following code :

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  constructor(props) {
    super (props);
```

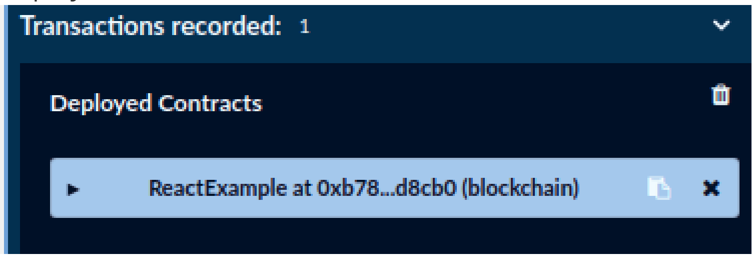
```
const MyContract = window.web3.eth.contract([
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "name": "result",
        "type": "bool"
      }
    ],
    "name": "ExperimentComplete",
    "type": "event"
  },
  {
    "constant": false,
    "inputs": [],
    "name": "kill",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [],
    "name": "setExperimentInMotion",
    "outputs": [
      {
        "name": "",
        "type": "bool"
      }
    ],
    "payable": true,
    "stateMutability": "payable",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "newState",
        "type": "string"
      }
    ],
    "name": "setState",
    "outputs": [],
    "payable": true,
    "stateMutability": "payable",
    "type": "function"
  },
  {
    "payable": true,
        "stateMutability": "payable",
        "type": "fallback"
      },
      {
        "inputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "constructor"
      },
      {
        "constant": true,
        "inputs": [],
        "name": "getSecret",
        "outputs": [
          {
            "name": "",
            "type": "string"
          }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
      },
      {
        "constant": true,
        "inputs": [],
        "name": "getState",
        "outputs": [
          {
            "name": "",
            "type": "string"
          }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
      },
      {
        "constant": true,
        "inputs": [],
        "name": "pseudoRandomResult",
        "outputs": [
          {
            "name": "",
            "type": "bool"
          }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
      },
      {
        "constant": true,
        "inputs": [],
        "name": "you_awesome",
        "outputs": [
          {
            "name": "",
            "type": "string"
          }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
      }
    ]);

    this.state = {
      ContractInstance: MyContract.at ('0x2edb84e826f93e1075fbb8c34c6f3300bd6aef2c'),
      contractState: ''
    }
    this.querySecret = this.querySecret.bind (this);
    this.queryContractState = this.queryContractState.bind (this);
    this.handleContractStateSubmit = this.handleContractStateSubmit.bind (this);
    this.queryConditionResult = this.queryConditionResult.bind (this);
    this.activateExperiment = this.activateExperiment.bind (this);
    this.state.event = this.state.ContractInstance.ExperimentComplete();
  }

  queryConditionResult () {
    const { pseudoRandomResult } = this.state.ContractInstance;

    pseudoRandomResult ((err, result) => {
      console.log ('This is the smart contract conditional::::', result);
    })
  }

  activateExperiment () {
    const { setExperimentInMotion } = this.state.ContractInstance;

    setExperimentInMotion ({
      gas: 300000,
      from: window.web3.eth.accounts[0],
      value: window.web3.toWei (0.01, 'ether')
    }, (err,result) => {
      console.log ('Experiment to determine true or false set in motion. ');
    })
  }

  querySecret () {
    const { getSecret } = this.state.ContractInstance;

    getSecret ((err, secret)=> {
      if (err) console.error ('An error occured::::', err);
      console.log ('This is our contract\'s secret::::', secret);
    })
    }

  queryContractState () {
    const { getState } = this.state.ContractInstance;

    getState ((err, state) => {
      if (err) console.error ('An error occured::::', err);
      console.log ('This is our contract\'s state::::', state);
    })
  }

  handleContractStateSubmit (event) {
    event.preventDefault();

    const { setState } = this.state.ContractInstance;
    const { contractState: newState } = this.state;

    var rawTx = {
      chainId: 1994,

      gas: 300000,
      from: window.web3.eth.accounts[0],
      value: window.web3.toWei (0.01, 'ether')
        }

    setState (newState, rawTx,
      (err, result) => {
        console.log('Smart contract state is changing.');
      }
    )
  }

  render () {
    this.state.event.watch ((err, event) => {
      if (err) console.error ('An error occured::::', err);
      console.log ('This is the event::::', event);
      console.log ('This is the experiment result::::', event.args.result);
    })

    return (
      <div className="App">
        <header className="App-header">
          <img src={ logo } className="App-logo" alt="logo" />
          <h1 className="App-title"> React and Ethereum private app </h1>
        </header>

        <br />
        <br />
        <button onClick={ this.querySecret }> Query Smart Contract's Secret </button>
        <br />
        <br />
        <button onClick={ this.queryContractState }> Query Smart Contract's State
</button>
```

```
        <br />
        <br />


        <form onSubmit={ this.handleContractStateSubmit }>
          <input
            type="text"
            name="state-change"
            placeholder="Enter new state..."
            value ={ this.state.contractState }
            onChange={ event => this.setState ({ contractState: event.target.value }) }
/>
          <button type="submit"> Submit </button>
        </form>

        <br />
        <br />
        <button onClick={ this.queryConditionResult }> Query Smart Contract's
Conditional Result </button>
        <br />
        <br />
        <button onClick={ this.activateExperiment }> Start Experiment on Smart Contract
</button>
        <br />
        <br />

      </div>
    );




  }


}

export default App;
```

Replace the smart contract address, on the line

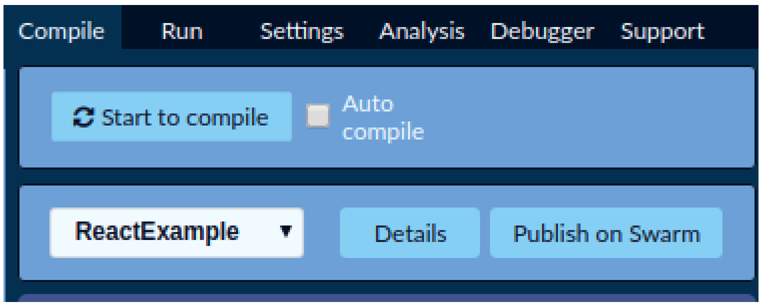
`ContractInstance: MyContract.at ('0x2edb84e826f93e1075fbb8c34c6f3300bd6aef2c')`

Replace the address the one we generated in the previous section. On remix, run tab, click on the copy icon next to the star in the "Deployed contracts" area :

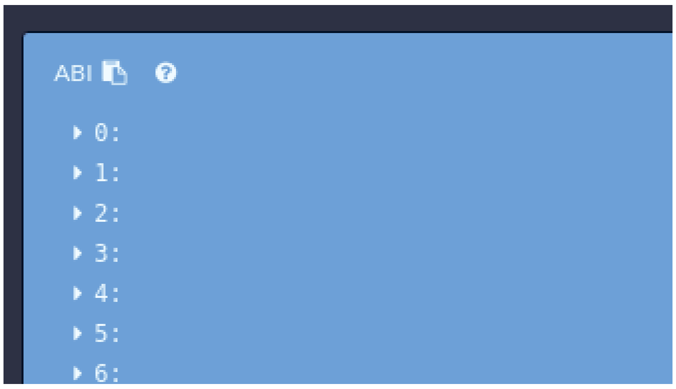


Each time the smart contract is redeployed, this step is to be reproduced with the new address as well as copy paste the structure in const MyContract = window.web3.eth.contract([ ])

To do this, go to compile tab, and click on Details button on the right of your selected file :

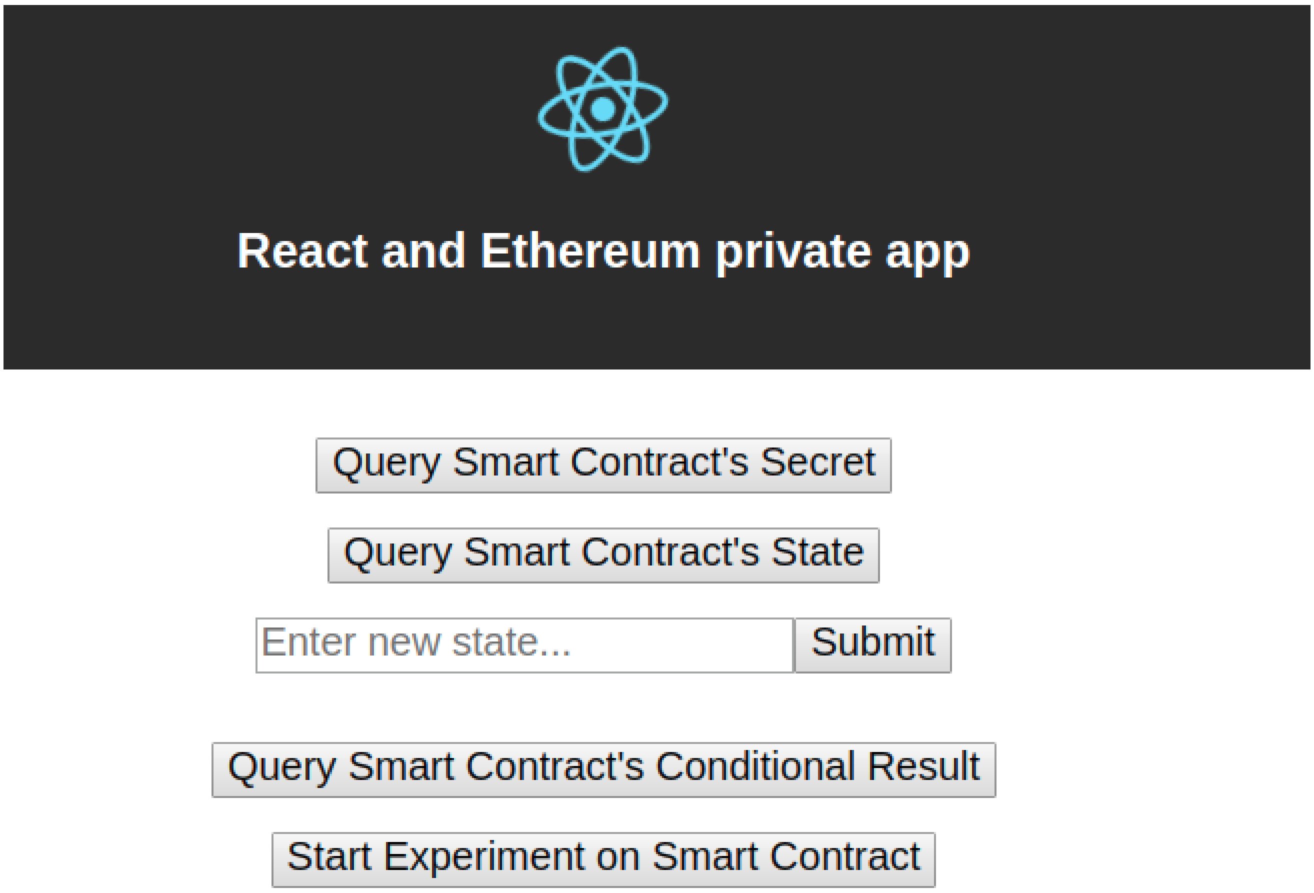


Then click on the copy button next to the ABI section :



Test the smart contract via React

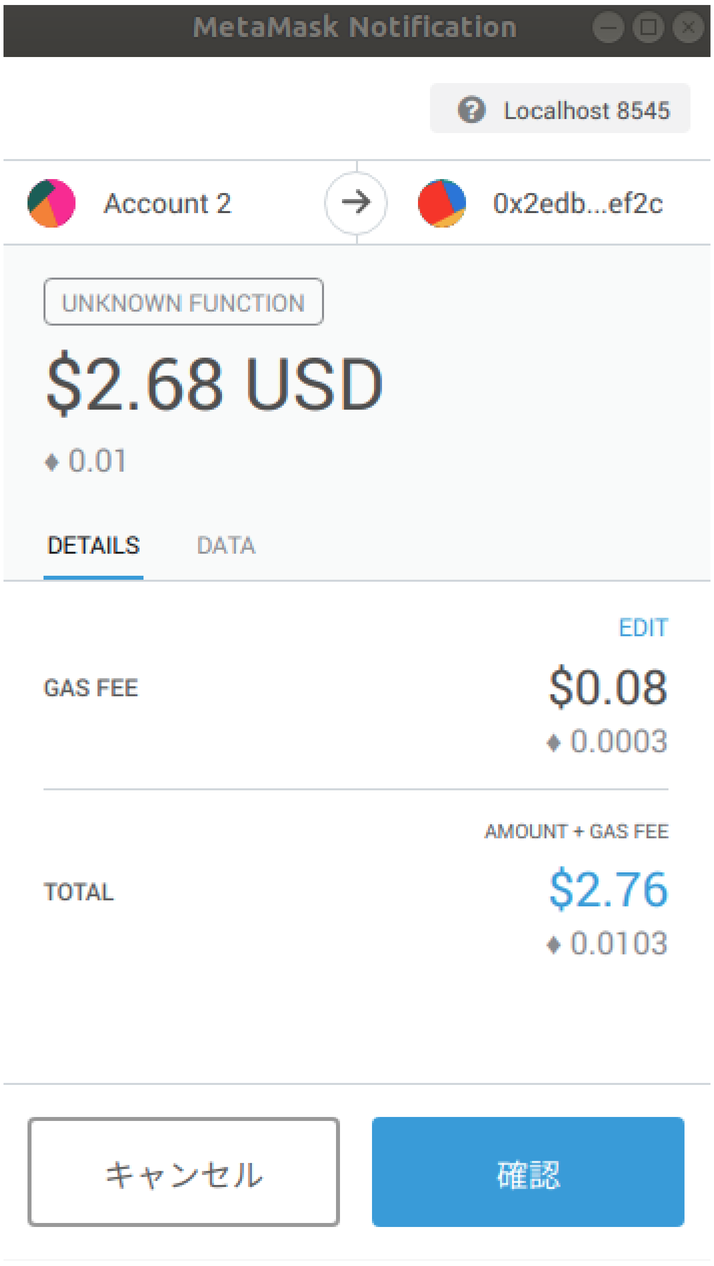
Once this is done, your react app should look like this :


React and Ethereum private app
Query Smart Contract's Secret
Query Smart Contract's State
Enter new state...
Submit
Query Smart Contract's Conditional Result
Start Experiment on Smart Contract

You can inspect the code to see what is doing each button and functions :

- Query Smart Contract's Secret : Read a private string via a function
- Query Smart Contract State: Read the private string state of the smart contract
- New state + Submit : Change the state of the smart contract
- Query Smart Contract's Conditional Result : Read the PseudoRandomResult boolean
- Start Experiment on Smart Contract : Trigger an event that activate setExperimentInMotion function to generate a pseudo random result, readable via the abode button

You will remark that each time a function need gas to work, Metamask will ask you to validate the transaction first :



By validating you authorize the transaction and the transfer of the funds.

A. Helpful: geth console commands

admin.nodeInfo.enode net.listening net.peerCount admin.peers eth.coinbase eth.getBalance(eth.coinbase) personal eth.accounts miner.setEtherbase(web3.eth.accounts[0]) miner.setEtherbase(“0xae13d41d66af28380c7af6d825ab557eb271ffff”) miner.start() miner.stop() miner.hashrate eth.getBlock(0) eth.getBlock(“latest”) eth.blockNumber web3.eth.getBlock(BLOCK_NUMBER).hash eth.syncing debug.verbosity(6) // highest logging level, 3 is default