Menu

Join the RSK Open Slack Community to get the latest updates from the RSK Ecosystem!

How to create your first frontend for smart contracts

In this tutorial I will show you step-by-step how to create your first front end to interact with a smart contract deployed at RSK local network, using only Javascript and HTML and connected to a wallet using a web3 provider.

Overview

Below is a summary of the steps to be taken to build our front end:

1. Configure Metamask to connect to RSK testnet;
2. Get some testnet RBTCs at faucet;
3. Connect Remix with RSK Testnet;
4. Create, compile and deploy a smart contract on RSK Testnet using Remix;
5. Initialise the project;
6. Install web3.js;
7. Create a javascript file;
8. Create a html file;
9. Install and run a local server;
10. Interact with the smart contract.

Steps 1 to 4 are explained in detail in the tutorial link below:

- **using Remix and Metamask with RSK testnet**

Webinar

We have run a **webinar** in which we run through this tutorial:

Serie de Webinar: Armá tu primer frontend para Smart Contracts

Go to top

The same webinar is also available in **Español** and **Português**.

Check out our **other webinars**.

Requirements

- Metamask
- Remix - web tool, online
- Node.js and NPM (Node Package Manager)
- Visual Studio Code (VSCode) or any other editor of your choice
- HTTP server: express
- web3.js

As earlier mentioned, to install Metamask and connect to RSK testnet and to connect Remix with RSK Testnet are explained in detail in the tutorial link below:

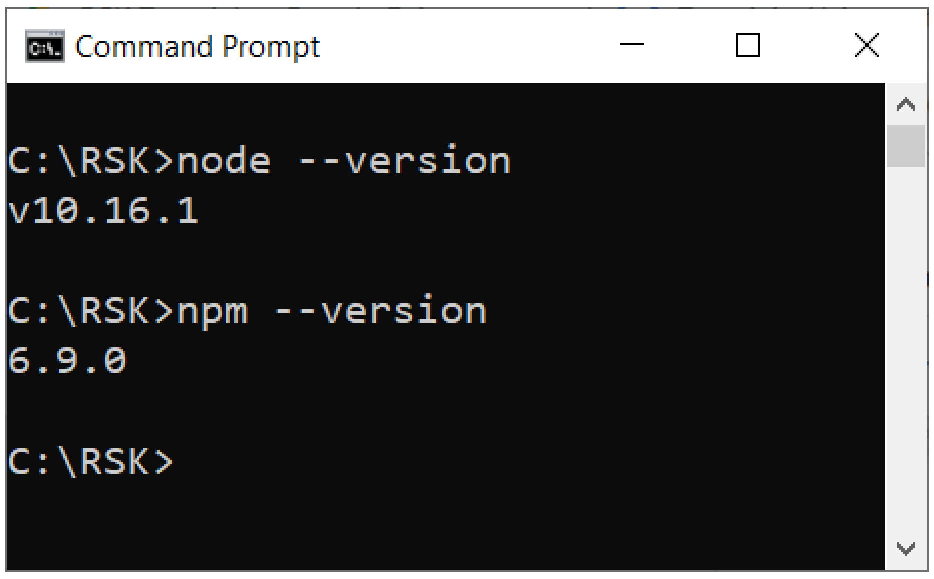
- **using Remix and Metamask with RSK testnet**

Node.js and NPM

Node.js and NPM are needed, though both are usually installed at once.

NB: To check if Node.js and NPM is already installed, input the following commands in the terminal:

```
node --version
npm --version
```



```
C:\RSK>node --version
v10.16.1

C:\RSK>npm --version
6.9.0

C:\RSK>
```

Go to **Node.js** if you need to install it.

Visual Studio Code (VSCode)

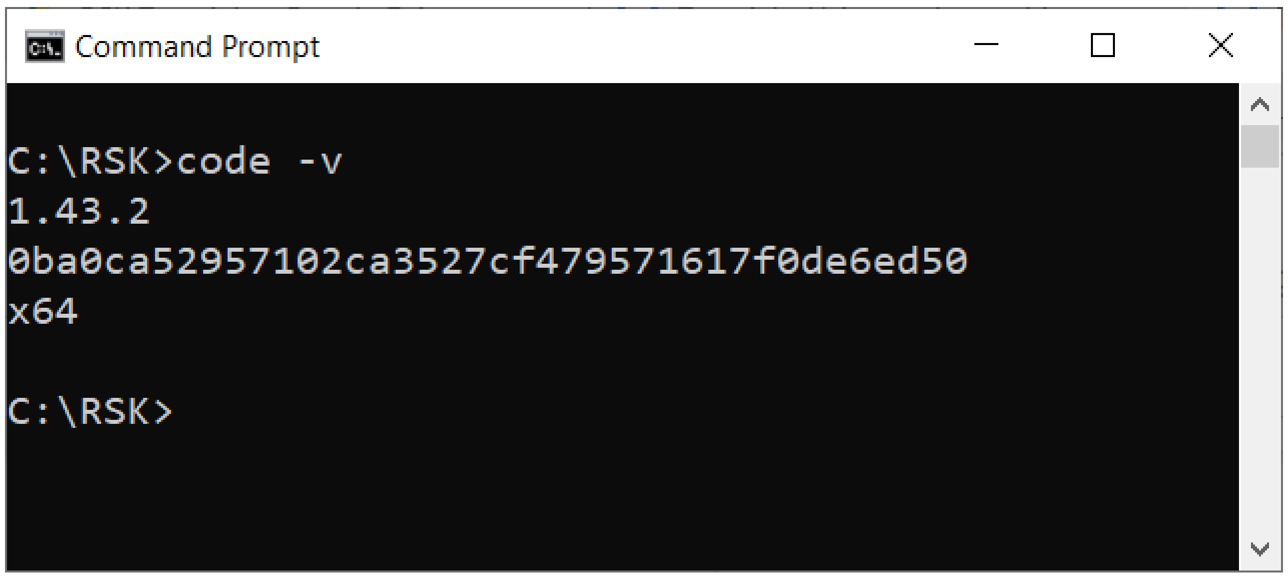
In this tutorial, We would use VSCode to create and deploy our project.

To use VSCode **download it here**.

Verify if your VS code installation was successful by typing the following command into the terminal:

```
code -v
```

Go to top



```
C:\RSK>code -v
1.43.2
0ba0ca52957102ca3527cf479571617f0de6ed50
x64

C:\RSK>
```

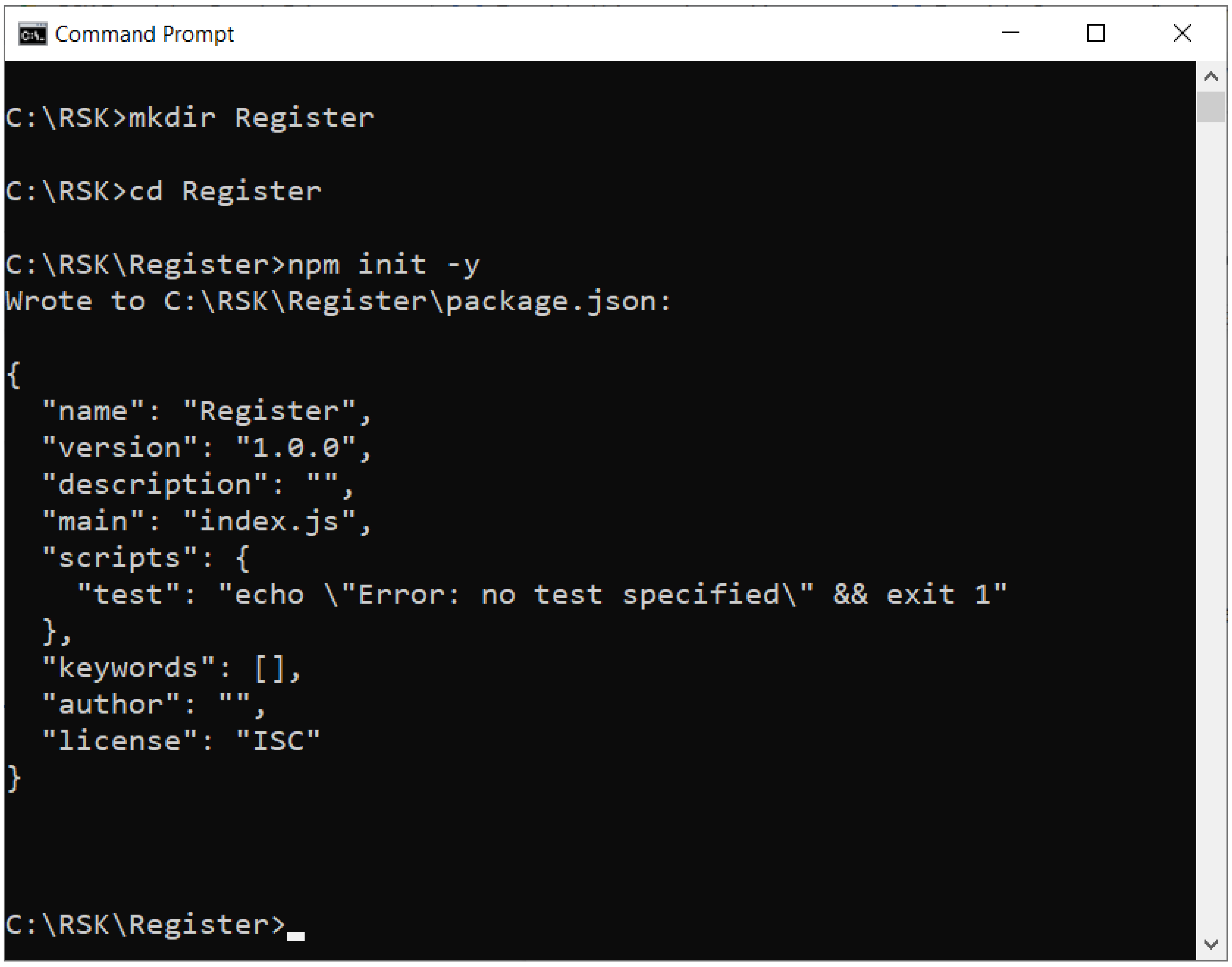
Create the Register project

Create a folder named Register, and start an npm project in the Register folder by typing the following commands below into the terminal:

```
mkdir Register
cd Register
npm init -y
```

For example, I will create a folder at this location - `C:\RSK\`

My project can be located in the folder `C:\RSK\Register`.



```
C:\RSK>mkdir Register

C:\RSK>cd Register

C:\RSK\Register>npm init -y
Wrote to C:\RSK\Register\package.json:

{
  "name": "Register",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}



C:\RSK\Register>
```

Express

Express is a Node.js web application framework that helps to develop web applications. It is a minimalist HTTP server.

To install Express, input the command below into the terminal and press `enter` at your project location:

```
npm install express --save
```

More info:

- http://expressjs.com/
- https://www.npmjs.com/package/express

Web3.js

Web3.js helps us to develop websites or clients that interact with the blockchain - writing code that reads and writes data from the blockchain with smart contracts.

The web3.js library is an Ethereum Javascript API which connects using the generic JSON-RPC spec. As RSK's virtual machine implementation is compatible with the Ethereum Virtual Machine (EVM), it is possible to use web3.js to interact with the front end and the RSK local node.

To install web3.js, input the command below into the terminal and press `enter` at your project location:

```
npm install web3 --save
```

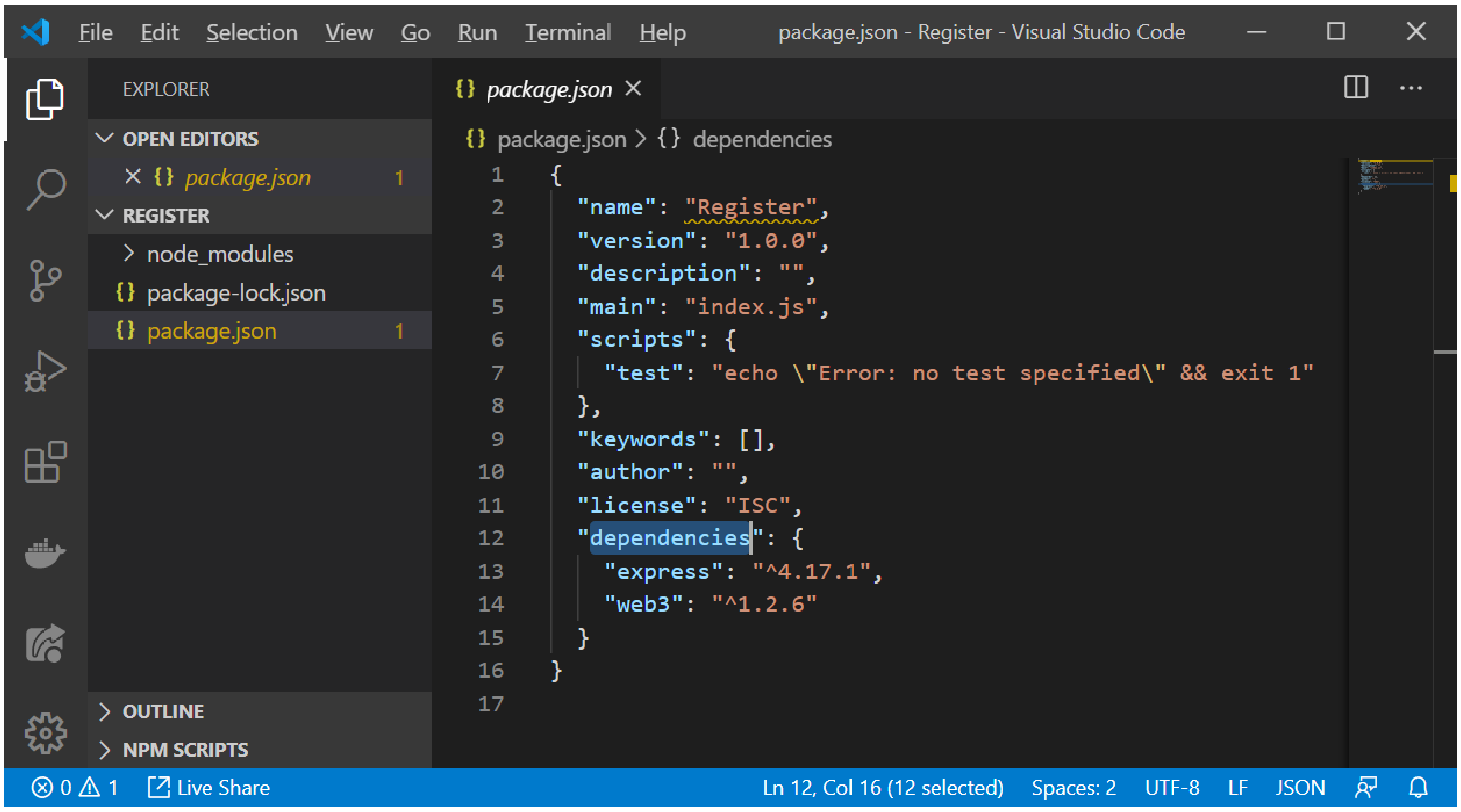
More info:

- https://web3js.readthedocs.io/

Check package.json

`package.json` is a file created by npm with some configurations, including the packages which we installed before using the command `npm init -y`.

After the installations, I will open the project folder named Register in VSCode and verify the file package.json. Let's take a look at the dependencies in the file:


File Edit Selection View Go Run Terminal Help
package.json - Register - Visual Studio Code
EXPLORER
OPEN EDITORS
package.json 1
REGISTER
node_modules
package-lock.json
package.json 1
OUTLINE
NPM SCRIPTS
package.json > {} dependencies
0 ⚠ 1 Live Share
Ln 12, Col 16 (12 selected) Spaces: 2 UTF-8 LF JSON

```json
{
  "name": "Register",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "web3": "^1.2.6"
  }
}
```

You will find the previously installed packages:

Go to top

```
  "dependencies": {
    "express": "^4.17.1",
    "web3": "^1.2.6"
  }
```

Deploy a smart contract

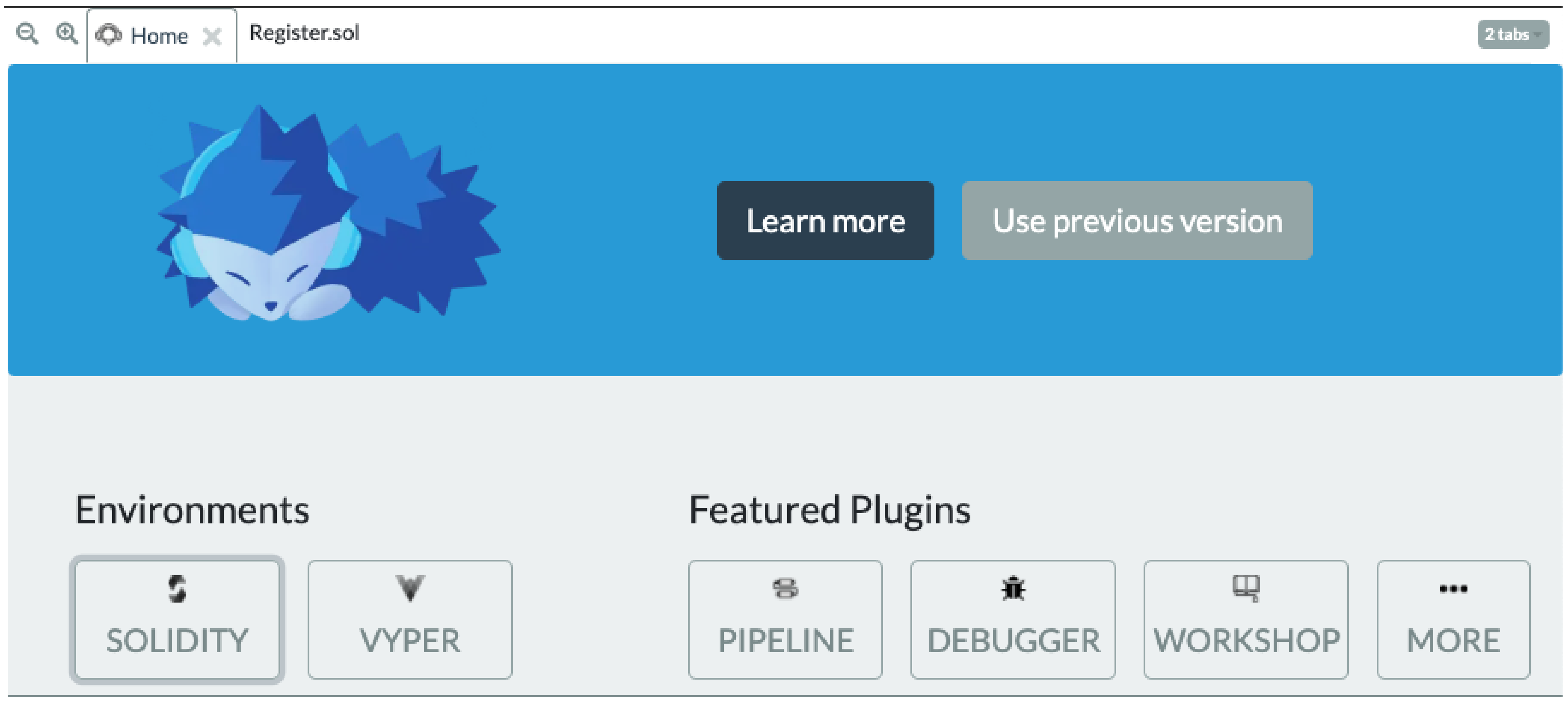
This is explained in detail in the tutorial:

- using Remix and Metamask with RSK testnet

We'll do the relevant parts of that tutorial here.

Remix

Go to http://remix.ethereum.org/



NB: Be sure to click the **Solidity** button under **Environment**, as seen in the image above.

We need this in order to compile our smart contract as our code is written in the Solidity language.

Create a smart contract

Create a new file

File name: `Register.sol`

Copy and paste the smart contract from the following gist, or copy and paste the code below:

https://raw.githubusercontent.com/solangegueiros/dapp-register-rsk/master/register-rsk-web3-injected/register.sol

Go to top

```
pragma solidity 0.5.4;

contract Register {
    string private info;

    function setInfo(string memory _info) public {
        info = _info;
    }

    function getInfo() public view returns (string memory) {
        return info;
    }
}
```

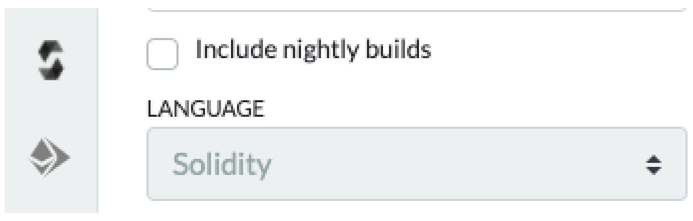
Register.sol

This smart contract has:

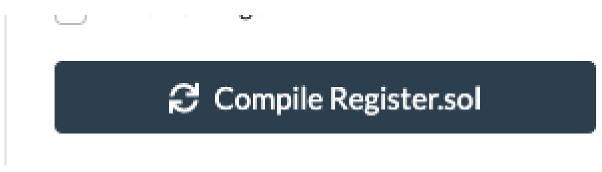
- A variable `info` to store a string
- A function `getInfo()` to return the string stored at variable info
- A function `setInfo()` to change the string stored at variable info

Compile a smart contract

In the 3rd button at the left side select Solidity compiler.



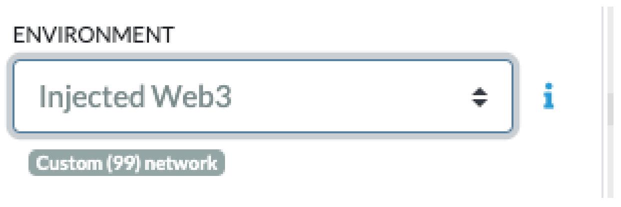
Click the button **Compile Register.sol**.



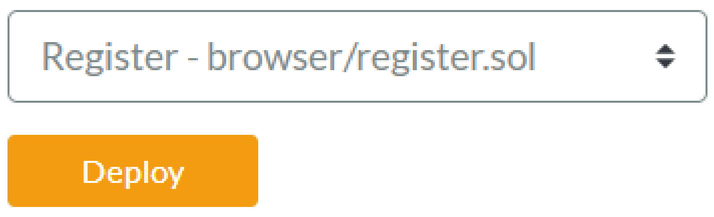
Deploy a smart contract

In the left side panel, go to the button Deploy and run transactions.

Under `Environment`, make sure you have selected the `Injected Web3` option, as this tells Remix to use the Web3 provider injected by a browser plugin such as **MetaMask** or **Nifty**.



Then click the button `Deploy`.

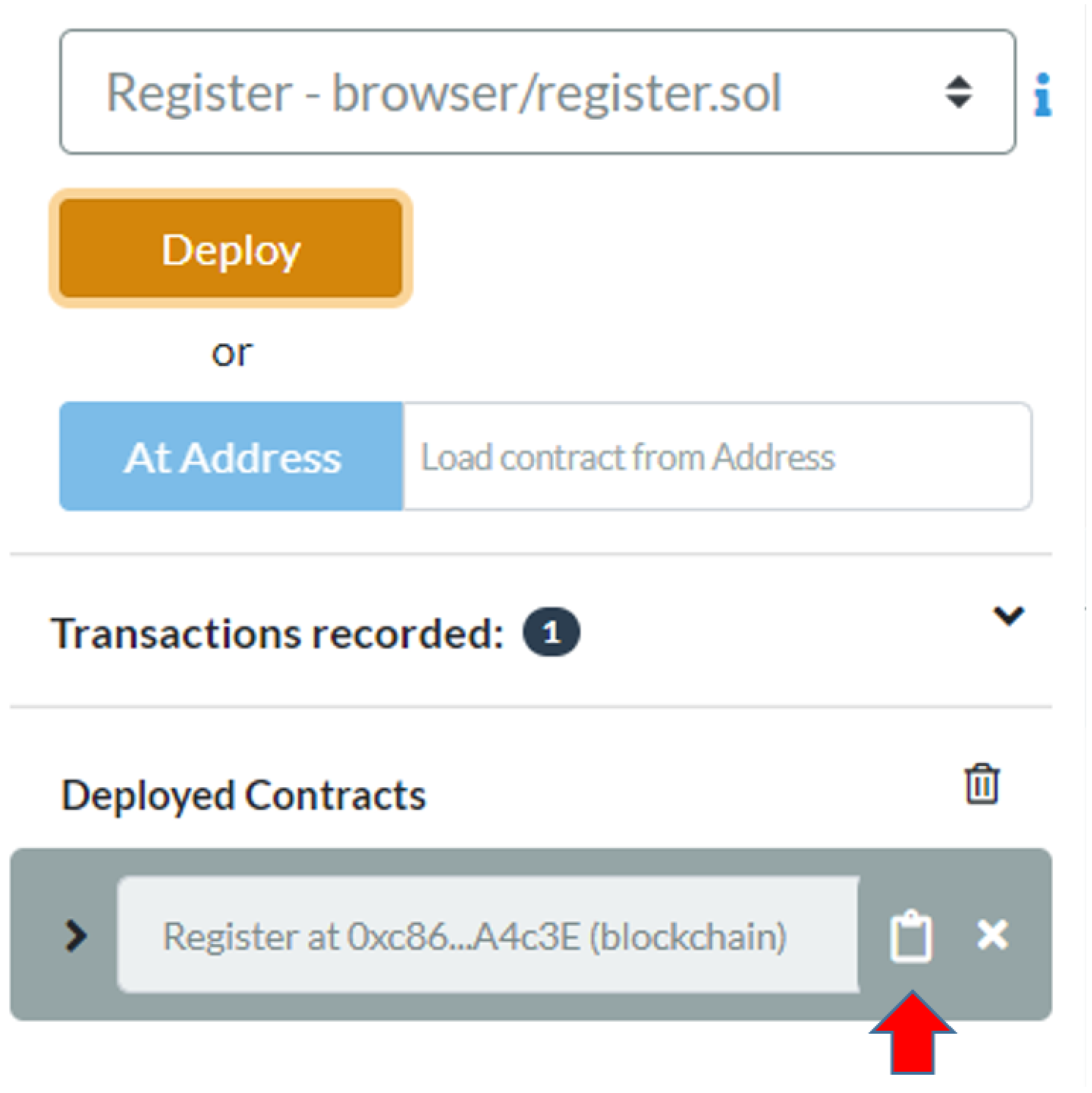


Go to top

Ensure that you have tRBTC in your wallet, as this is needed to pay gas fees when deploying the smart contracts. To do so, please follow this guide: Using Remix and Metamask with RSK testnet.

When a smart contract is deployed with Remix, we can see it in the left panel under deploy and run transactions.

Click on the copy button at the right side of the smart contract to copy the address of the smart contract. We will need it for use on the front end.


Register - browser/register.sol
Deploy
or
At Address
Load contract from Address
Transactions recorded: 1
Deployed Contracts
Register at 0xc86...A4c3E (blockchain)

In my example, the contract address is `0xc864D0fef177A69aFa8E302A1b90e450910A4c3E`.

Client-Side Application - The front end

Now let's start building out the front end that will interact with the smart contract.

We have only 2 files in the front end:

1. index.html
2. index.js

Index.html

In Register folder, create a file named `index.html`.

Copy and paste the HTML from the following gist, or copy and paste the code below:

Go to top

https://raw.githubusercontent.com/solangegueiros/dapp-register-rsk/master/register-rsk-web3-injected/index.html

```html
<!DOCTYPE html>
<html >
  <head>
    <title>Register information at Blockchain</title>

    <link href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css'
rel='stylesheet' type='text/css'>
    <script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
    <script src="./node_modules/web3/dist/web3.min.js"></script>
    <script src="./index.js"></script>
  </head>

  <body class="container">

    <h1 class="page-header">Register information at Blockchain - RSK network</h1>

    <div class="row">
      <div>
        <h3 class="sub-header">Set information</h3>
        <form class="form-inline" role="form">
          <div class="form-group">
            <table>
              <tr>
                <td><label for="newInfo">Info:</label> </td>
                <td>
                  <input class="form-control" id="newInfo">
                </td>
              </tr>
            </table>
          </div>
          <a href="#" onclick="registerSetInfo()" class="btn btn-primary">Set</a>
        </form>
      </div>
    </div>

    <div class="row">
      <div>
        <h3 class="sub-header">Get last information saved</h3>
        <form class="form-inline" role="form">
          <a href="#" onclick="registerGetInfo()" class="btn btn-primary">Get</a>
          <div class="form-group">
            <table>
              <tr>
                <td>Info:</td>
                <td>
                  <label id="lastInfo">
                </td>
              </tr>
            </table>
          </div>
        </form>
      </div>
    </div>

  </body>
</html>
```

Go to top

Index.js

In the `Register` folder, create the file `index.js`.

Copy and paste the JavaScript from the following gist, or copy and paste the code below:

https://raw.githubusercontent.com/solangegueiros/dapp-register-rsk/master/register-rsk-web3-injected/index.js

Go to top

```
// Source code to interact with smart contract

// web3 provider with fallback for old version
window.addEventListener('load', async () => {
  // New web3 provider
  if (window.ethereum) {
      window.web3 = new Web3(ethereum);
      try {
          // ask user for permission
          await ethereum.enable();
          // user approved permission
      } catch (error) {
          // user rejected permission
          console.log('user rejected permission');
      }
  }
  // Old web3 provider
  else if (window.web3) {
      window.web3 = new Web3(web3.currentProvider);
      // no need to ask for permission
  }
  // No web3 provider
  else {
      console.log('No web3 provider detected');
  }
});
console.log (window.web3.currentProvider)

// contractAddress and abi are setted after contract deploy
var contractAddress = '0xc864D0fef177A69aFa8E302A1b90e450910A4c3E';
var abi = JSON.parse( '[{"constant":true,"inputs":[],"name":"getInfo","outputs":
[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},
{"constant":false,"inputs":[{"name":"_info","type":"string"}],"name":"setInfo","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"}]' );

//contract instance
contract = new web3.eth.Contract(abi, contractAddress);

// Accounts
var account;

web3.eth.getAccounts(function(err, accounts) {
  if (err != null) {
    alert("Error retrieving accounts.");
    return;
  }
  if (accounts.length == 0) {
    alert("No account found! Make sure the Ethereum client is configured properly.");
    return;
  }
  account = accounts[0];
  console.log('Account: ' + account);
  web3.eth.defaultAccount = account;
});

//Smart contract functions
function registerSetInfo() {
```

Go to top

```
    info = $("#newInfo").val();
    contract.methods.setInfo (info).send( {from: account}).then( function(tx) {
      console.log("Transaction: ", tx);
    });
    $("#newInfo").val('');
  }

  function registerGetInfo() {
    contract.methods.getInfo().call().then( function( info ) {
      console.log("info: ", info);
      document.getElementById('lastInfo').innerHTML = info;
    });
  }
```

This part connected to RSK Local node using the wallet injected, in our case, Metamask:

```
// web3 provider with fallback for old version
window.addEventListener('load', async () => {
  // New web3 provider
  if (window.ethereum) {
      window.web3 = new Web3(ethereum);
      try {
          // ask user for permission
          await ethereum.enable();
          // user approved permission
      } catch (error) {
          // user rejected permission
          console.log('user rejected permission');
      }
  }
  // Old web3 provider
  else if (window.web3) {
      window.web3 = new Web3(web3.currentProvider);
      // no need to ask for permission
  }
  // No web3 provider
  else {
      console.log('No web3 provider detected');
  }
});
```

Update index.js

Did you remember the address of the smart contract that you copied after deploying it?

It will be updated here:

```
var contractAddress = '0xc864D0fef177A69aFa8E302A1b90e450910A4c3E';
```

HTML server

In the `Register` folder, create a file named `server.js`.

Go to top

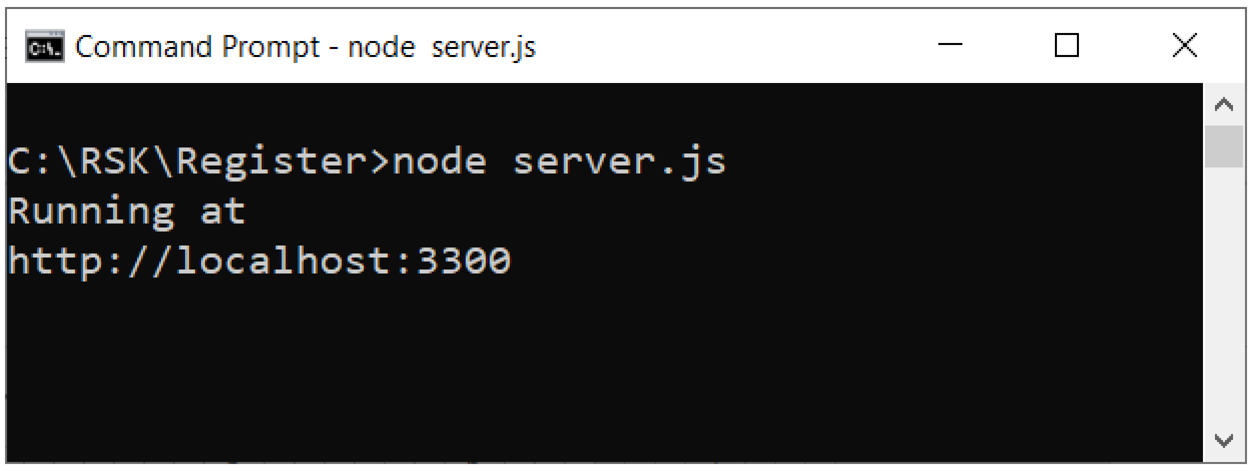
```
var express = require('express');
var app = express();
app.use(express.static(__dirname));
app.listen('3300');
console.log('Running at\nhttp://localhost:3300');
```

This file configures the express HTML server.

Run Server

The last step is to execute the express server. Input the command below into the terminal.

```
node server.js
```



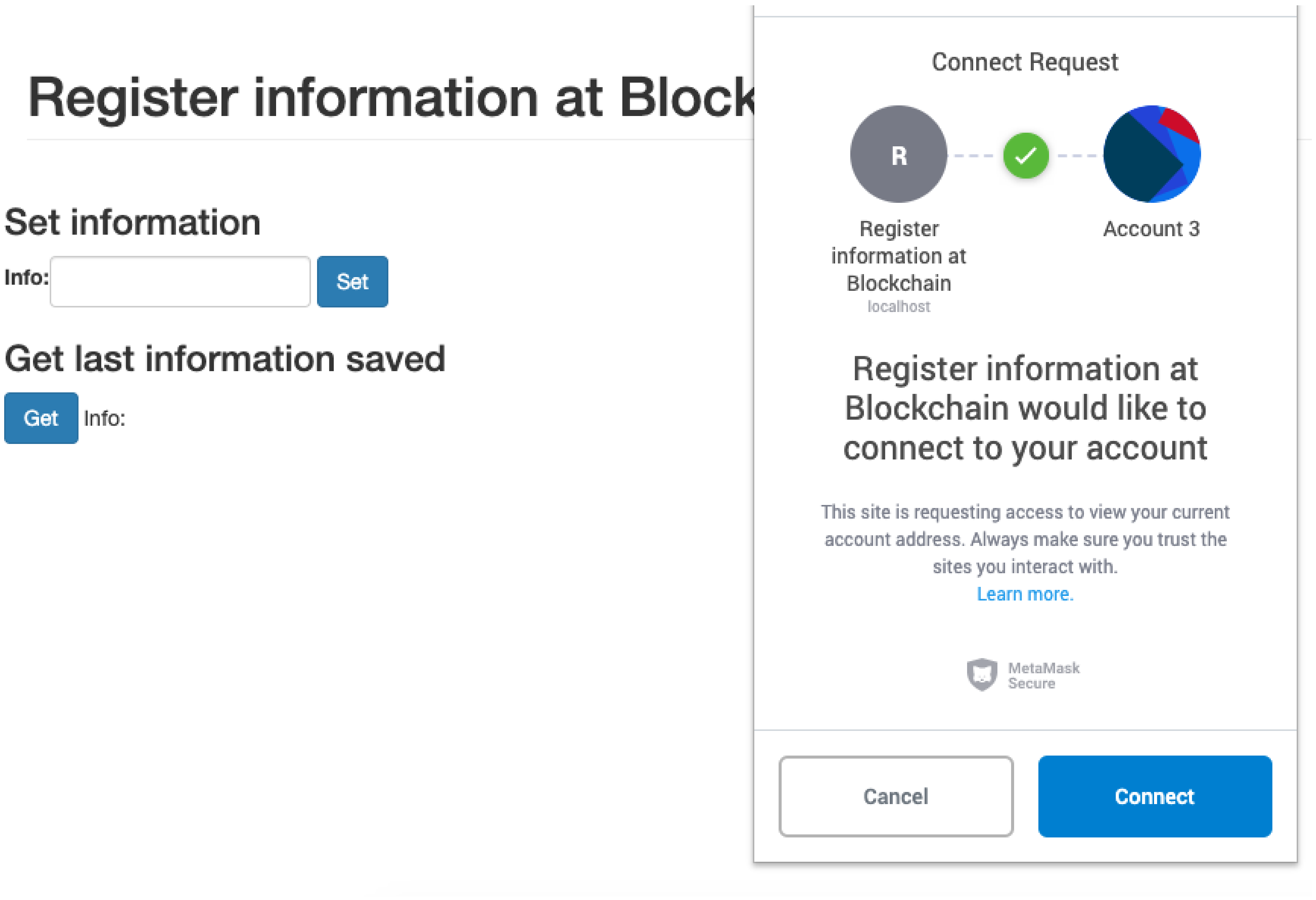
```
C:\RSK\Register>node server.js
Running at
http://localhost:3300
```

In your browser, go to:

```
http://localhost:3300
```

Click on the **Connect** button when the Metamask Wallet pops up

Go to top

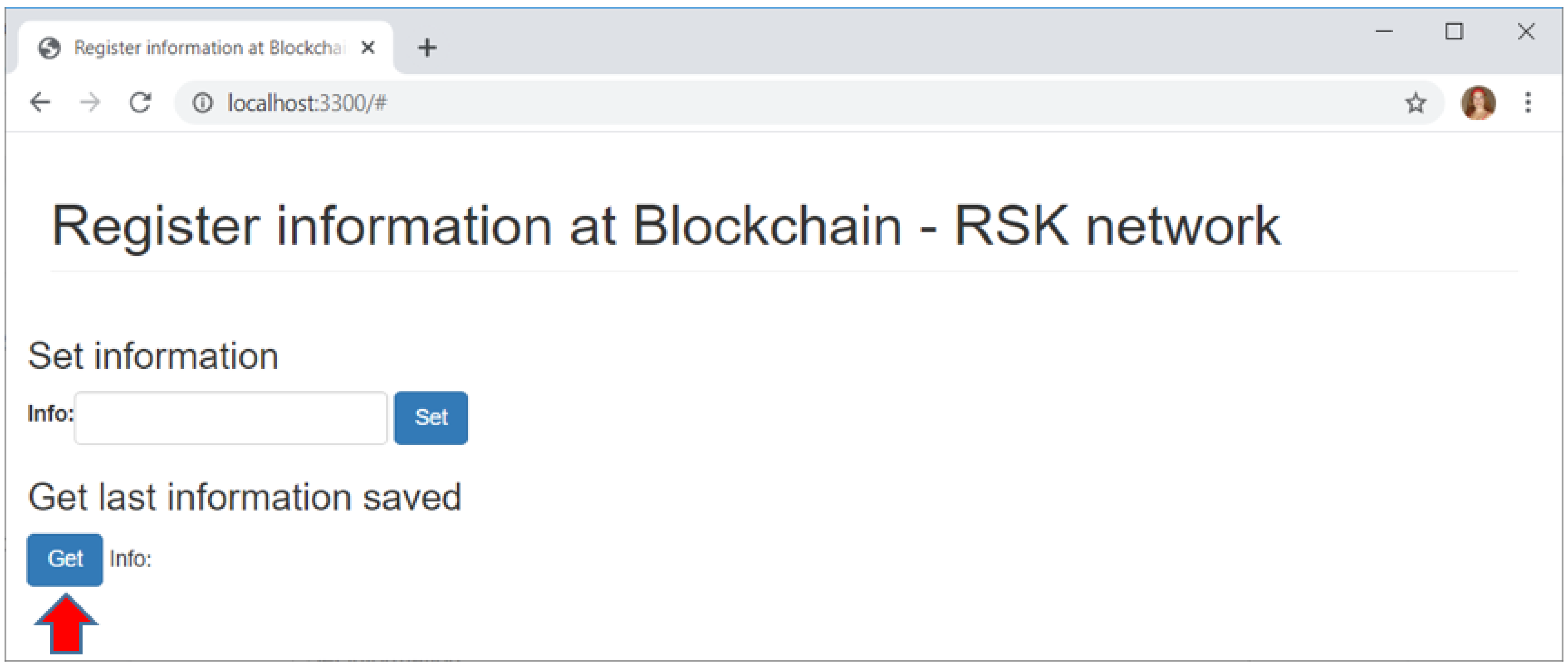

Register information at Block
Set information
Info:
Set
Get last information saved
Get
Info:
Connect Request
R
Register information at Blockchain
localhost
Account 3
Register information at Blockchain would like to connect to your account
This site is requesting access to view your current account address. Always make sure you trust the sites you interact with.
Learn more.
MetaMask Secure
Cancel
Connect

Interact with the smart contract

Get

Click on the **Get** button.

The web3 instance will call the `setInfo()` function on the `register` smart contract instance, with the info that you entered into the text field.


Register information at Blockcha
localhost:3300/#
Register information at Blockchain - RSK network
Set information
Info:
Set
Get last information saved
Get
Info:

We do not have any information stored, because we did not specify an initial value in the smart contract.

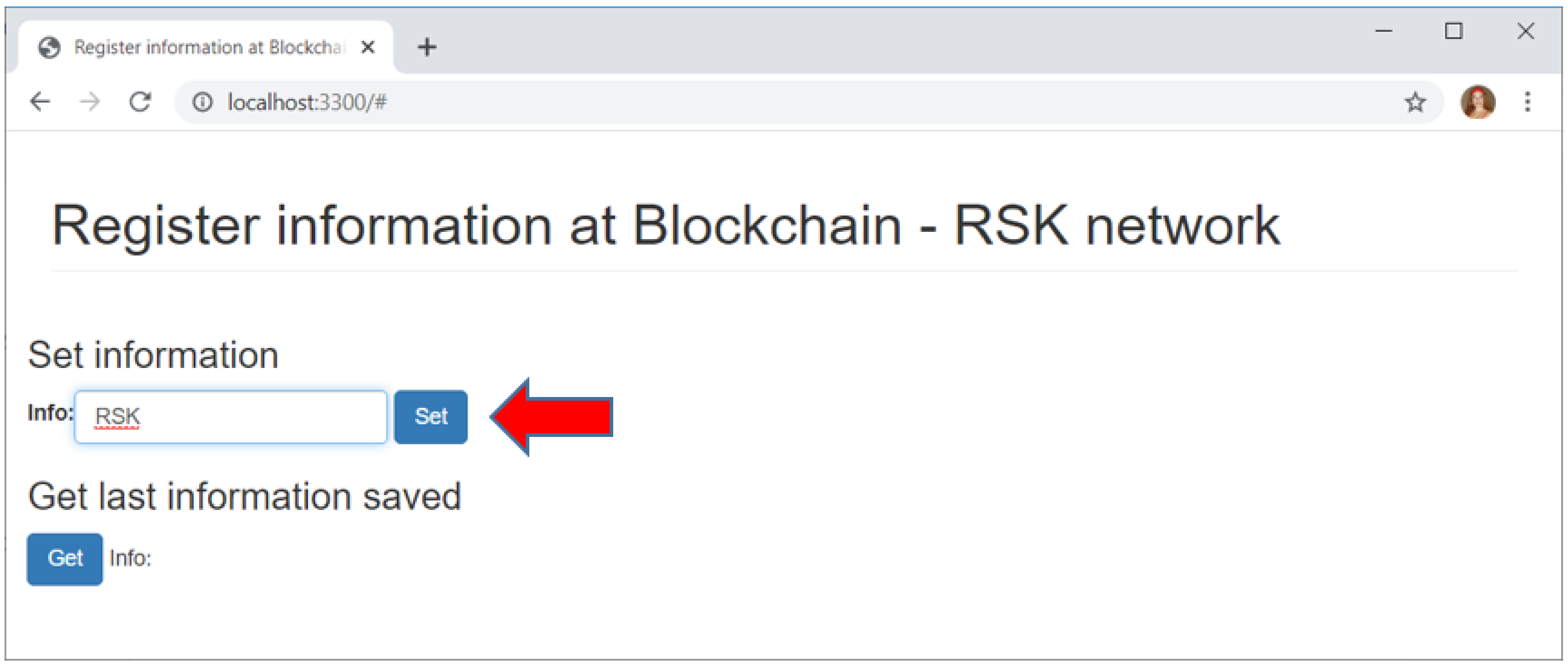
Set

Enter any value in the info text field, and click on the Set button.

The web3 instance would call the `setInfo()` function at the smart contract instance register, with the info that you defined.

I will enter the value “RSK”.

Go to top


Register information at Blockchain - RSK network

Set information

Info: RSK Set

Get last information saved

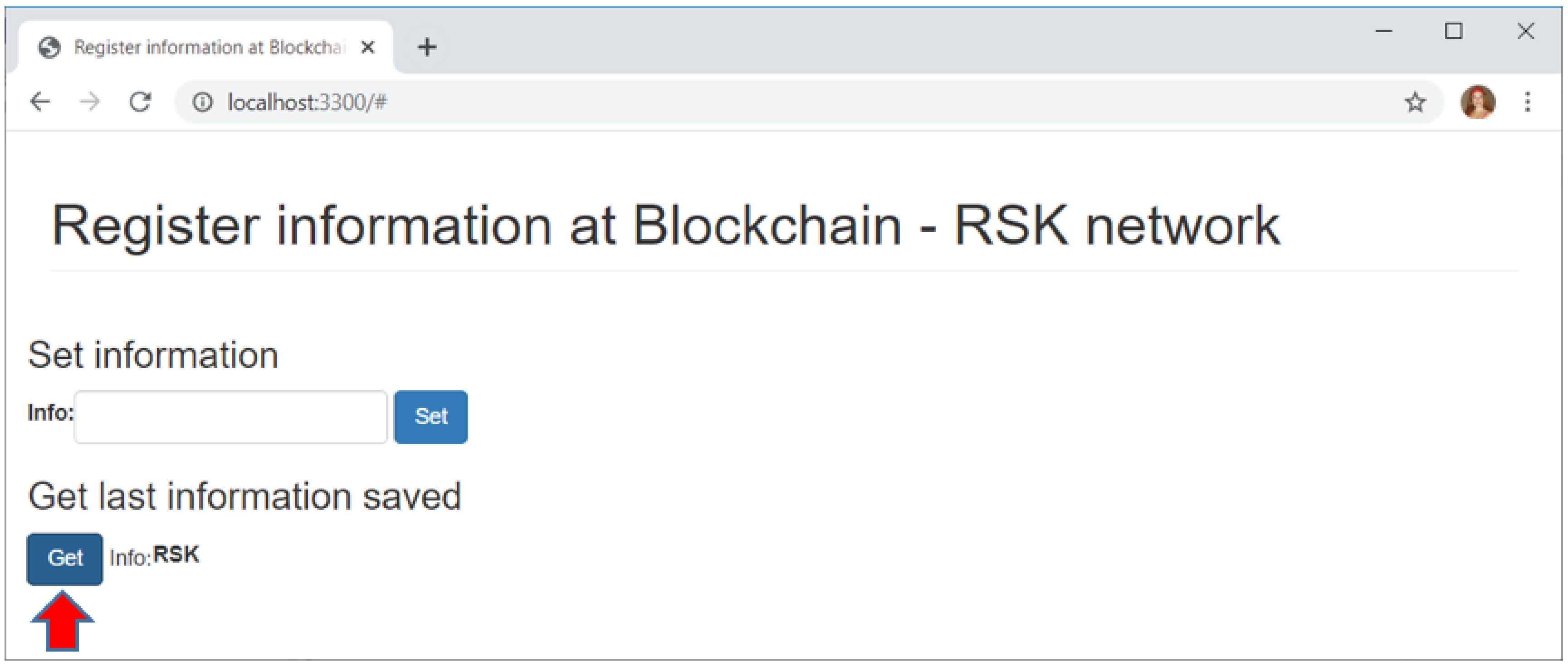
Get Info:

Wait a few seconds for the transaction to be included in a block…

getInfo (again)

Now we have the value “RSK” saved, and we can check it.

Click on the Get button again


Register information at Blockchain - RSK network

Set information

Info: Set

Get last information saved

Get Info:**RSK**

And it returned the info “RSK”.

Great! Now we have an information stored in our smart contract, and we can retrieve it!

Note

> *We have a smart contract with a public function that has no restrictions about who is allowed to call it, but that’s just for demonstration purposes.*
>
> *In a real world smart contract, such checks would be performed in any functions that alter the smart contract state.*

Congratulations!

You have successfully built and deployed your first decentralized application (DApp) powered by RSK smart contracts!

You can download the full source code to this tutorial here:

https://github.com/solangegueiros/dapp-register-rsk/tree/master/register-rsk-web3-injected

Satisfied with this tutorial? I’d appreciate your feedback and you can share it :)

Go to top

Subscribe to our Newsletter

E-Mail Address Subscribe

Rsk is the most secure smart contract network in the world and enables decentralized applications secured by the Bitcoin Network to empower people and improve the quality of life of millions.

Start now

Whitepapers
Original
Updated
RIF
Roadmap
Explorer

Merged Mining
Bounty Program
Grants Program
Ecosystem Fund
Innovation Studio

Open Finance
Use cases
Faqs
Blog
Brand Guidelines

Terms & Conditions
Privacy Policy
About IOVlabs
Contact IOVlabs

Documentation

RSK Public Key (2ED3 E888 0384 D3D9 70B6 A612 BEBC A6A9 63F6 1479)

Go to top