A practical view of web3.js

TOMMASO AZZALIN

7 DECEMBER 2021

SEMINAR FOR THE COURSE “BLOCKCHAIN & CRYPTOCURRENCIES”

A.Y. 2021/2022 – UNIVERSITY OF BOLOGNA

Presentation topics

Presentation topics

- What is **web3.js**?

Presentation topics

- What is **web3.js**?
- How to install, import and use the library

Presentation topics

- What is **web3.js**?
- How to install, import and use the library
- A quick overview of the main features of the library

Presentation topics

- What is **web3.js**?
- How to install, import and use the library
- A quick overview of the main features of the library
- A quick and practical demo of how to build a Dapp with **web3.js**, **Truffle** and other libraries

What is web3.js?

What is web3.js?

- It is a JavaScript library for connecting to the Ethereum blockchain

What is web3.js?

- It is a JavaScript library for connecting to the Ethereum blockchain
- This is performed by connecting to an Ethereum node

What is web3.js?

- It is a JavaScript library for connecting to the Ethereum blockchain
- This is performed by connecting to an Ethereum node
- It hides the complexity of handling the connection to the Ethereum node

What is web3.js?

- It is a JavaScript library for connecting to the Ethereum blockchain
- This is performed by connecting to an Ethereum node
- It hides the complexity of handling the connection to the Ethereum node
- It hides the actual JSON-RPC API calls to the node

What is web3.js?

- It is a JavaScript library for connecting to the Ethereum blockchain
- This is performed by connecting to an Ethereum node
- It hides the complexity of handling the connection to the Ethereum node
- It hides the actual JSON-RPC API calls to the node
- Otherwise, one could also connect with Ethereum's own JSON-RPC API (*de gustibus non disputandum est*)

Installing web3.js

Installing web3.js

- **Prerequisite**: Node.js installed

Installing web3.js

- **Prerequisite**: Node.js installed
- Install (globally): `npm install -g web3`

Installing web3.js

- **Prerequisite**: Node.js installed
- Install (globally): `npm install -g web3`
- Install (inside a Node.js project): `npm install --save web3`

Installing web3.js

- **Prerequisite**: Node.js installed
- Install (globally): `npm install -g web3`
- Install (inside a Node.js project): `npm install --save web3`
- If our code is not client-side, we are done…

Installing web3.js

- **Prerequisite**: Node.js installed
- Install (globally): `npm install -g web3`
- Install (inside a Node.js project): `npm install --save web3`
- If our code is not client-side, we are done…
- …but what if we are creating a Dapp?
  - We will see in the next slides one extra step (which will be needed during the demo)

Installing web3.js

- **Prerequisite**: Node.js installed
- Install (globally): `npm install -g web3`
- Install (inside a Node.js project): `npm install --save web3`
- If our code is not client-side, we are done…
- …but what if we are creating a Dapp?
  - We will see in the next slides one extra step (which will be needed during the demo)
- What if one needs types for static type checking?
  - Install them with `npm install typescript ts-node @types/node typechain @typechain/web3-v1 --save-dev`

Importing web3.js

Importing web3.js

- Import in a .js file (to be executed with Node.js): `const web3 = require('web3');`

Importing web3.js

- Import in a .js file (to be executed with Node.js): `const web3 = require('web3');`
- Import in a .js file (to be executed by the browser):

Importing web3.js

- Import in a .js file (to be executed with Node.js): `const web3 = require('web3');`
- Import in a .js file (to be executed by the browser):
  - There is no import directive needed inside the JavaScript code…

Importing web3.js

- Import in a .js file (to be executed with Node.js): `const web3 = require('web3');`
- Import in a .js file (to be executed by the browser):
  - There is no import directive needed inside the JavaScript code…
  - …but the library must be added to the same HTML file:
    - `<script src="https://cdnjs.cloudflare.com/ajax/libs/web3/1.6.1/web3.min.js"></script>`
    - `<script src="js/app.js"></script> <!-- Here the library is used, without the need to import it -->`

Using web3.js

Using web3.js

- We have to create an instance of the class `Web3`

Using web3.js

- We have to create an instance of the class `Web3`
- The constructor of `Web3` requires a provider (i.e., an object which handles the connection to the Ethereum node) and it is used as follows: `new Web3(provider)`

Using web3.js

- We have to create an instance of the class `Web3`
- The constructor of `Web3` requires a provider (i.e., an object which handles the connection to the Ethereum node) and it is used as follows: `new Web3(provider)`
- The `provider` instance can be created in three main ways:

Using web3.js

- We have to create an instance of the class `Web3`
- The constructor of `Web3` requires a provider (i.e., an object which handles the connection to the Ethereum node) and it is used as follows: `new Web3(provider)`
- The `provider` instance can be created in three main ways:
  - `ethereum` which is injected inside the `window` object by the crypto wallets, such as MetaMask, therefore accessible via `window.ethereum`
    - Before creating an instance of `Web3`, we have to wait the user to enable the connection: `await window.ethereum.request({ method: "eth_requestAccounts"});`

Using web3.js

- We have to create an instance of the class `Web3`
- The constructor of `Web3` requires a provider (i.e., an object which handles the connection to the Ethereum node) and it is used as follows: `new Web3(provider)`
- The `provider` instance can be created in three main ways:
  - `ethereum` which is injected inside the `window` object by the crypto wallets, such as MetaMask, therefore accessible via `window.ethereum`
    - Before creating an instance of `Web3`, we have to wait the user to enable the connection: `await window.ethereum.request({ method: "eth_requestAccounts"});`
  - `web3.currentProvider` is deprecated, but it was used by previous versions of the same wallets, accessible via `window.web3.currentProvider`

Using web3.js

- We have to create an instance of the class `Web3`
- The constructor of `Web3` requires a provider (i.e., an object which handles the connection to the Ethereum node) and it is used as follows: `new Web3(provider)`
- The `provider` instance can be created in three main ways:
  - `ethereum` which is injected inside the `window` object by the crypto wallets, such as MetaMask, therefore accessible via `window.ethereum`
    - Before creating an instance of `Web3`, we have to wait the user to enable the connection: `await window.ethereum.request({ method: "eth_requestAccounts"});`
  - `web3.currentProvider` is deprecated, but it was used by previous versions of the same wallets, accessible via `window.web3.currentProvider`
  - `Web3.providers.HttpProvider(url)` is used for instantiating providers directly connected to the nodes (skipping the wallets part), `url` is where the node can be found and it is used as follows: new `Web3.providers.HttpProvider(url)`

Using web3.js

Using web3.js

- We have created an instance of the class `Web3`, let's say this object is called `web3`

Using web3.js

- We have created an instance of the class `Web3`, let's say this object is called `web3`
- Let's assume we connected to a smart contract SC through an instance `sc` (its type is `web3.eth.Contract`)

Using web3.js

- We have created an instance of the class `Web3`, let's say this object is called `web3`
- Let's assume we connected to a smart contract SC through an instance `sc` (its type is `web3.eth.Contract`)
  - Let's assume it has two methods: `set()` and `get()`, the former modifies the state of the contract, the latter does not

Using web3.js

- We have created an instance of the class `Web3`, let's say this object is called `web3`
- Let's assume we connected to a smart contract SC through an instance `sc` (its type is `web3.eth.Contract`)
  - Let's assume it has two methods: `set()` and `get()`, the former modifies the state of the contract, the latter does not
  - `set()` requires a transaction to be sent, hence
    - `sc.methods.set().send()`

Using web3.js

- We have created an instance of the class `Web3`, let's say this object is called `web3`
- Let's assume we connected to a smart contract SC through an instance `sc` (its type is `web3.eth.Contract`)
  - Let's assume it has two methods: `set()` and `get()`, the former modifies the state of the contract, the latter does not
  - `set()` requires a transaction to be sent, hence
    - `sc.methods.set().send()`
  - `get()` instead requires only the blockchain to be read, hence
    - `sc.methods.get().call()`

Using web3.js

- We have created an instance of the class `Web3`, let's say this object is called `web3`
- Let's assume we connected to a smart contract SC through an instance `sc` (its type is `web3.eth.Contract`)
  - Do we need to deploy it to the **Mainnet** or other test networks?
    - Our `web3` instance has to be created with an `HttpProvider`
    - `sc = await sc.deploy({data: SC.bytecode}).send({from: address});`

Demo

Demo

- Let’s now see how to build a To-Do list Dapp with **web3.js** and other libraries

Demo

- Let’s now see how to build a To-Do list Dapp with **web3.js** and other libraries
- Other than **web3.js**, we will need **Truffle**

Demo

- Let’s now see how to build a To-Do list Dapp with **web3.js** and other libraries
- Other than **web3.js**, we will need **Truffle**
- Install **Truffle** with `npm install -g truffle`

Demo

- Let’s now see how to build a To-Do list Dapp with **web3.js** and other libraries
- Other than **web3.js**, we will need **Truffle**
- Install **Truffle** with `npm install -g truffle`
- Link to the repository: https://github.com/TommasoAzz/web3-demo

Demo

- Let's now see how to build a To-Do list Dapp with **web3.js** and other libraries
- Other than **web3.js**, we will need **Truffle**
- Install **Truffle** with `npm install -g truffle`
- Link to the repository: https://github.com/TommasoAzz/web3-demo
- Link to download the code (if you want to follow along the demo): https://github.com/TommasoAzz/web3-demo/tree/demo

Thanks for your attention!

Please ask questions, if you have some!

References

- Web3.js – Ethereum JavaScript API – Web3.js
- JSON-RPC API - Ethereum
- Pet shop tutorial – Truffle Suite
- Adding TypeScript to Truffle and Buidle – Solidity Developer