Ethereum DAPP development
With Javascript (2021)

Andrea Lisi, andrealisi.12lj@gmail.com
Decentralized Applications

Thanks to web3 library it is possible to write a Decentralized Application (DApp), i.e. an application whose main logic is not executed by a single server but by a decentralized network like Ethereum

- The Backend is (partially) composed by smart contracts
- The Frontend is any software application with Web3
- In a realistic scenario, rely the entire backend code on the smart contracts is unfeasible
  - Smart contracts should code only the logic to decentralize
Decentralized Applications

The most popular Ethereum DApp: Cryptokitties

● Buy, sell and trade unique digital cats
● It became so popular that on December 2017 the Ethereum network suffered a slowdown
Part 3

Web DApp

A simple web application with help of Metamask
State of the project

- Init a project
- Write contracts
- Write tests
- Is the contract ready?
  - Yes
  - No
- Deploy / Migrate
  - Local network?
    - Yes
    - Use Ganache
    - No
    - Use Infura
  - No
- Do you need a Proxy?
  - Yes
  - Use Geth / Parity
Tools: Metamask

Metamask is an Ethereum **wallet** implemented as a browser extension
- Specifically, it is a Hierarchical Deterministic (HD) wallet
- [https://metamask.io/](https://metamask.io/)

It is possible to create accounts for different Ethereum networks: the main network, or test networks like Ropsten and Rinkeby
- It is also possible to import Ganache accounts if we test the DApp with a local blockchain
Walkthrough

● Setup the environment
  ○ Create a server for the DApp
    ■ Configure it
  ○ Link the smart contract libraries
    ■ Web3 and truffle-contract

● Develop the core of the DApp

● Run the DApp
  ○ Connect it to the local network
Create HTML project

In the Truffle root directory create a src/ folder inside the Truffle folder, and create js/, css/ and index.html in src/

- src/
  - js/, css/, index.html

The web page needs to get the contract json files from the build/ folder

To do this, we need a local server, serving the DApp. We can install one with NodeJs
Lite server

In the Truffle root directory initialize a node project creating package.json file with

- $ npm init

We need a server serving the contracts to the DApp

- lite-server, for single-page apps:
  - $ npm install --save lite-server
  - It automatically updates package.json

- https://www.npmjs.com/package/lite-server
Lite server

In `package.json` insert the pair “dev”: “lite-server” inside “scripts”

- To execute the lite server with `npm run dev`

Now `package.json` should look like this:

```json
"scripts": {
  "dev": "lite-server",
},
"dependencies": {
  "lite-server": "^2.6.1"
},
```
Lite server

Create the configuration file, `bs-config.json`, inside the Truffle project root directory to tell lite-server the folders it needs to look at to serve the web application

- `bs` stands for BrowserSync, the tool lite-server is built on top of

```json
{
  "server": {
    "baseDir": ["./src", ".build/contracts", ".node_modules/@truffle/contract", ".node_modules/web3"]
  }
}
```
Web3 and truffle-contract

We need to import the `web3` and `truffle-contract` libraries

Install `truffle-contract` with npm

- `$ npm install --save web3`
- `$ npm install --save @truffle/contract`
  - Check its [browser usage](#)
Web3 and truffle-contract

Include the scripts in index.html

- web3.min.js & truffle-contract.js
- app.js, the DApp script
Develop the core of the DApp

Now we have all the requirements for the DApp

In the following steps we are going to:

- Take an example contract
  - Remember to compile it
- Code `app.js` that is going to implement the frontend and call the smart contract
- Try the DApp locally
contract MyContract {

    uint public value;
    event click();

    constructor() public {
        value = 42;
    }

    function pressClick() public {
        emit click();
    }
}
DApp initialization

Create the DApp script called *app.js* in *src/js/*

This script should:
1. Init web3
2. Init smart contracts (read json files)
3. Activate event listeners
4. Render page (call smart contract functions useful for initialization)
5. Implement a onclick function
app.js, Overall structure

App = {
  // Attributes

  init: function() { return App.initWeb3(); },
  // Functions

} // Call init whenever the window loads
$(function() {
  $(window).on('load', function () {
    App.init();
  });
});
app.js, App object

```javascript
App = {
    contracts: {}, // Store contract abstractions
    web3Provider: null, // Web3 provider
    account: '0x0', // current ethereum account

    init: function() { return App.initWeb3(); },

    initWeb3: function() { /* initialize Web3 */ return App.initContract(); },
    initContract: function() { /* Upload the contract's */ return App.listenForEvents(); },
    listenForEvents: function() { /* Activate event listeners */ return App.render(); },
    render: function() { /* Render page */ }
}
```
app.js, 1. Init web3

```javascript
initWeb3: function() {
    if(typeof web3 !== 'undefined') { // Check whether exists a provider, e.g Metamask
        App.web3Provider = window.ethereum; // standard since 2/11/18
        web3 = new Web3(App.web3Provider);
        try { // Permission popup
            ethereum.enable().then(async() => { console.log("DApp connected"); });
        } catch(error) { console.log(error); }
    } else { // Otherwise, create a new local instance of Web3
        App.web3Provider = new Web3.providers.HttpProvider(App.url); // <==
        web3 = new Web3(App.web3Provider);
    }
    return App.initContract();
},
```
app.js, 2. Init contracts

```javascript
initContract: function() {

    // Store ETH current account
    web3.eth.get_coinbase(function(err, account) {
        if (err === null) {
            App.account = account;
            console.log(account);
            $('#accountId').html('Your address: ' + account);
        }
    });

    // Init contracts
}
```
app.js, 2. Init contracts

```javascript
initContract: function() {

    // Store ETH current account
    // ...

    // Init contracts
    $.getJSON("MyContract.json").done(function(c) {
        App.contracts["MyContract"] = TruffleContract(c);
        App.contracts["MyContract"].setProvider(App.web3Provider);

        return App.listenForEvents();
    });
},
```
app.js, 3. Activate event listeners

```javascript
listenForEvents: function() {
  App.contracts["MyContract"].deployed().then(async (instance) => {
    // click is the Solidity event
    instance.click().on('data', function (event) {
      $("#eventId").html("Event catched!");
      console.log("Event catched");
      console.log(event);
      // If event has parameters: event.returnValues.*paramName*
    });
  });
  return App.render();
},
```
app.js, 3. Activate event listeners

```javascript
listenForEvents: function() {
    App.contracts["MyContract"].deployed().then(async (instance) => {
        web3.eth.getBlockNumber(function (error, block) {
            // click is the Solidity event
            instance.click().on('data', function (event) {
                $("#eventId").html("Event catched!");
                console.log("Event catched");
                console.log(event);
                console.log(block);  // If you want to get the block
            });
        });
    });
    return App.render();
},
```
app.js, 3. Activate event listeners

Usually online we find solutions involving `event().watch(callback)`, but the `watch` function is not anymore supported by the most recent versions of web3Js

Source, Github Issue

More on events:

https://web3js.readthedocs.io/en/1.0/web3-eth-contract.html#contract-events

https://ethereum.stackexchange.com/questions/64872/truffle-how-to-get-event
```javascript
render: function() {

    // Retrieve contract instance
    App.contracts["MyContract"].deployed().then(async(instance) => {

        // Call the value function (value is a public attribute)
        const v = await instance.value();
        console.log(v);
        $("#valueId").html("" + v);
    });
},
```
// Call a function of a smart contract
// The function send an event that triggers a transaction: Metamask pops up and ask the user to confirm the transaction

pressClick: function() {

    App.contracts["Contract"].deployed().then(async(instance) => {
        await instance.pressClick({from: App.account});
    });
}
DApp, try it

- Start lite server with
  `$ npm run dev`

- Migrate the contract on Ganache
- Open the browser, open Metamask and select “Private network”
- Import an account from Ganache by copying its private key and pasting it into Metamask
  - If the account was already imported and previously used, it may need to **reset it**, otherwise its nonce conflicts with the new Ganache instance
Resources


Dapp, full tutorial (2018): https://www.youtube.com/watch?v=3681ZYbDSSk
  ● Update: https://www.youtube.com/watch?v=X6DzzeoRTS0

On lite-server:
Extra

More on DApp development
Truffle: workflow

1. Init a project
2. Write contracts
3. Write tests

Is the contract ready?

- Yes: Deploy / Migrate
- No: Local network?

- Yes: Use Ganache
- No: Do you need a Proxy?

- Yes: Use Infura
- No: Use Geth / Parity

- Use Geth / Parity

- Use Infura
- Do you need a Proxy?

- Is the contract ready?
- Write tests
- Write contracts
- Init a project
**Tools: Infura**

Infura is a hosted Ethereum node cluster that lets users run your application without requiring them to set up their own Ethereum node or wallet.

Infura can be used to migrate a DApp to a supported Ethereum network.

It is necessary to register to the service and create a project that generates an ID and an API key.
Migrate to an Ethereum network

With Truffle is very easy to connect to an Ethereum network exploiting the Infura service:

● In this way you do not need to download the blockchain
● You need to register to Infura and get an API token
● You need a wallet like Metamask
   ○ Create an account on the chosen network
   ○ And get Ether for that account to pay for the gas
● You need to modify truffle_config.js
   ○ Here how: https://www.trufflesuite.com/guides/using-infura-custom-provider
Decentralized Applications

Truffle provides DApp boilerplates (project stubs) in its “boxes”

- $ truffle unbox pet-shop
- https://truffleframework.com/boxes

Truffle provides a tool called Drizzle to help the development of a DApp with ReactJs

- https://truffleframework.com/drizzle