Aula 24 - Ethereum: Introdução ao Web3

Prof. Rogério Aparecido Gonçalves *Universidade Tecnológica Federal do Paraná (UTFPR)*

Nesta aula são apresentadas algumas ferramentas de Desenvolvimento e Frameworks para o desenvolvimento e implantação de Contratos Inteligentes. Apresenta uma introdução ao `Web3`, métodos de desenvolvimento, teste e verificação de contratos inteligentes com `Ganache`, console do cliente `Geth` e Remix IDE. Introduz o `Truffle` *framework*, que também pode ser usado para testar, migrar contratos inteligentes e o Drizzle, para criar *frontends* de `DApps` de maneira mais fácil, com IPFS, para hospedar as páginas web da aplicação.

Sumário

1 Introdução

1.1 Objetivos

- Apresentação de Ferramentas de Desenvolvimento e *Frameworks*.

- Explorar a biblioteca `Web3` com o cliente `Geth`, desenvolvimento de contratos, interação com contratos via *frontends*.

1.2 Explorando Web3 com Geth

- `Web3` é uma biblioteca `JavaScript` que pode ser usada na comunicação com um Nó *Ethereum* via comunicação `RPC`. `Web3` expõe métodos que o acesso está disponível sobre `RPC`.
- A interação com o cliente `Geth` é possível via *Geth JavaScript Console*, que expõe vários métodos de consulta e gerenciamento do blockchain.
- Vimos os comandos para execução do `Geth` e do Console JavaScript na **Aula 021 - Prática sobre Ethereum: Ambiente de Desenvolvimento**.
- Iniciar o Nó `Geth` com suporte ao `web3`:

```
$ geth --datadir ~/.etherprivate/ --allow-insecure-unlock --networkid 786 --http
    --http.addr 127.0.0.1 --http.port 8559 --http.api
    "eth,net,web3,personal,engine,admin,debug" --keystore ~/.etherprivate/keystore
    --authrpc.addr localhost --authrpc.port 8551 --authrpc.vhosts localhost
    --authrpc.jwtsecret ~/.etherprivate/geth/jwtsecret --nodiscover --maxpeers 15
```

- Iniciar um console para a interação com a execução:

```
$ geth attach ~/.etherprivate/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.26-stable-e5eb32ac/linux-amd64/go1.19.3
coinbase: 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
at block: 0 (Wed Dec 31 1969 21:00:00 GMT-0300 (-03))
 datadir: /home/rag/.etherprivate
 modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0
     personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
```

- Verificando se os recursos `web3` estão disponíveis:

```
> web3.version
{
  api: "0.20.1",
  ethereum: undefined,
  network: "786",
  node: "Geth/v1.10.26-stable-e5eb32ac/linux-amd64/go1.19.3",
  whisper: undefined,
  getEthereum: function(callback),
  getNetwork: function(callback),
  getNode: function(callback),
  getWhisper: function(callback)
}
>
```

1.3 *Web3 deployment*

- Faremos um *deploy* usando o *Geth console*.
- O passo a passo pode ser visto no livro e iremos reproduzir aqui, seguindo a sequência de passos:
  - Executar o *Geth client*.
  - Criar um script de *deployment*, usando a `ABI` e o *bytecode*, e algum código `JavaScript`.
  - Faremos o *deploy* do contrato via linha de comando pelo *Geth console*.
  - Interagir com o contrato via um *frontend* web.

1.4 *Web3 deployment: Executar o* Geth client

- Executar o *Geth client*. [✓]
- Executar o *Geth console*. [✓]

1.5 *Web3 deployment: Criar um script de* deployment

- Compile o contrato com o `solc` ou utilizando o `Remix IDE`, gerando o binário e a `ABI`:

```
$ solc --bin --abi -o bin ValueChecker.sol
$ ls
bin deploy.js ValueChecker.sol
$ cd bin
$ ls
valueChecker.abi valueChecker.bin
$ cat valueChecker.bin
6080604052600a600055348015610015576000 80fd5b5061018b8061002560003960 00f3fe60806040523480
156100105760 0080fd5b506004361061002b5760003560e01c8063f9d55e2114610030575b600080fd5b6100
4a6004803603810190610045919061 00f2565b610060565b604051610057919061013a565b60405180910390
f35b600080548260ff16106100ae577f3eb1a229ff7995457774a4bd31ef7b13b6f4491ad1ebb8961af120b8
b4b6239c600160405161009d919061013a565b60405180910390a1600190506100af565b5b919050565b6000
80fd5b600060ff82169050919050565b6100cf816100b9565b81146100da57600080fd5b50565b6000813590
506100ec816100c6565b92915050565b60006020828403121561010857610076100b4565b5b600061011684
8285016100dd565b91505092915050565b60008115159050919050565b610134816101 1f565b82525050565b
60006020820190506101 4f60008301846101 2b565b9291505056fea2646970667358221220 88a7e63726327b
857c0d0a6d073976f05d5073826c629671c857a375db35d51c64736f6c63430008110033

$ cat valueChecker.abi
[{"anonymous":false,"inputs":[{"indexed":false,"internalType":"bool","name":"returnValue",
"type":"bool"}],"name":"valueEvent","type":"event"},{"inputs":[{"internalType":"uint8",
"name":"x","type":"uint8"}],"name":"Matcher","outputs":[{"internalType":"bool",
"name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"}]
```

- Preparação do código *JavaScript*:

```
var valuecheckerContract = web3.eth.contract([{ "anonymous": false, "inputs": [{
    "indexed": false, "internalType": "bool", "name": "returnValue", "type": "bool"
    }], "name": "valueEvent", "type": "event" }, { "inputs": [{ "internalType":
    "uint8", "name": "x", "type": "uint8" }], "name": "Matcher", "outputs": [{
    "internalType": "bool", "name": "", "type": "bool" }], "stateMutability":
    "nonpayable", "type": "function" }]);
```

```
var valuechecker = valuecheckerContract.new({
    from: web3.eth.accounts[0],
    data:
        '0x6080604052600a600055348015610015576000 80fd5b5061010d806100256000396000f3006080
         6040526004361061 9080803560ff16906020019092919050505060895 65b60405180821515151581
         5260200191505060405180910390f35b600080548260ff1610151560db577f3eb1a229ff79954577
         74a4bd31ef7b13b6f4491ad1ebb8961af120b8b4b6239c60016040518082151515158152602001 91
         505060405180910390a16001905060dc565b5b9190505600a165627a7a723058209ff756514f1ef4
         6f5650d800506c4eb6be2d8d71c0e2c8b0ca50660fde82c7680029', gas: '4700000'
},
    function (e, contract) {
        console.log(e, contract);
        if (typeof contract.address !== 'undefined') {
            console.log('Contract mined! address: ' + contract.address +
                'transactionHash: ' + contract.transactionHash);
        }
    })
```

1.6 *Web3 deployment: Fazendo o* deploy *pelo* Geth console

- No *Geth console* dê um *unclock* na conta:

```
> personal.listAccounts[0]
"0xedbc36d74d5a1cd64db36e53798bd1781f0c4955"
> personal.unlockAccount(personal.listAccounts[0])
Unlock account 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
Passphrase:
true
>
```

- Cole o código `JavaScript` para fazer o *deploy*:

```
> var valuecheckerContract = web3.eth.contract([{ "anonymous": false, "inputs": [{
    "indexed": false, "internalType": "bool", "name": "returnValue", "type": "bool"
    }], "name": "valueEvent", "type": "event" }, { "inputs": [{ "internalType":
    "uint8", "name": "x", "type": "uint8" }], "name": "Matcher", "outputs": [{
    "internalType": "bool", "name": "", "type": "bool" }], "stateMutability":
    "nonpayable", "type": "function" }]);
undefined
> var valuechecker = valuecheckerContract.new({
......from: web3.eth.accounts[0],
......data:
    '0x6080604052600a600055348015610015576000 80fd5b5061010d806100256000396000f30060806040
5260043610603f576000357c0100000000000000000000000000000000000000000000000000000000900463f
fffffff168063f9d55e21146044575b600080fd5b348015604f57600080fd5b50606f6004803603810190808080
3560ff16906020019092919050505060895 65b60405180821515151581526020019150506040518091039 0f35
b600080548260ff1610151560db577f3eb1a229ff7995457774a4bd31ef7b13b6f4491ad1ebb8961af120b8b4
b6239c60016040518082151515158152602001915050604051809103 90a16001905060dc565b5b9190505600a
165627a7a723058209ff756514f1ef46f5650d800506c4eb6be2d8d71c0e2c8b0ca50660fde82c7680029',
```

```
gas: '4700000'
...... },
... function (e, contract) {
...... console.log(e, contract);
...... if (typeof contract.address !== 'undefined') {
......... console.log('Contract mined! address: ' + contract.address +
    'transactionHash: ' + contract.transactionHash);
......... }
...... })
Error: insufficient funds for gas * price + value undefined
undefined
> miner.start()
null
>
```

- Na execução do *deploy* deu uma mensagem de erro `Error: insufficient funds for gas * price + value undefined`, pois a carteira da conta selecionada não tem saldo suficiente. É necessário minerar para ganhar algum saldo:

```
> miner.start()
null
> miner.stop()
> null
```

- Repetindo o processo de *deploy*:

```
> personal.unlockAccount(personal.listAccounts[0])
Unlock account 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
Passphrase:
true
> var valuecheckerContract = web3.eth.contract([{ "anonymous": false, "inputs": [{
    "indexed": false, "internalType": "bool", "name": "returnValue", "type": "bool"
    }], "name": "valueEvent", "type": "event" }, { "inputs": [{ "internalType":
    "uint8", "name": "x", "type": "uint8" }], "name": "Matcher", "outputs": [{
    "internalType": "bool", "name": "", "type": "bool" }], "stateMutability":
    "nonpayable", "type": "function" }]);
undefined
> var valuechecker = valuecheckerContract.new({
...... from: web3.eth.accounts[0],
......data:
    '0x6080604052600a600055348015610015576000 80fd5b5061010d806100256000396000f30060806040
5260043610603f576000357c0100000000000000000000000000000000000000000000000000000000900463f
fffffff168063f9d55e21146044575b600080fd5b348015604f57600080fd5b50606f6004803603810190808 0
3560ff1690602001909291905050506089565b604051808215151515815260200191505060405180910390f35
b600080548260ff1610151560db577f3eb1a229ff7995457774a4bd31ef7b13b6f4491ad1ebb8961af120b8b4
b6239c60016040518082151515158152602001915050604051809103 90a16001905060dc565b5b9190505600a
165627a7a723058209ff756514f1ef46f5650d800506c4eb6be2d8d71c0e2c8b0ca50660fde82c7680029',
gas: '4700000'
...... },
... function (e, contract) {
...... console.log(e, contract);
```

```
20 ...... if (typeof contract.address !== 'undefined') {
21 ......... console.log('Contract mined! address: ' + contract.address +
       'transactionHash: ' + contract.transactionHash);
22 ......... }
23 ...... })
24 null [object Object]
25 undefined
```

- Nos logs do Nó `Geth` irá aparecer a mensagem de que o contrato foi submetido:

```
1 INFO [11-24|12:44:17.115] Submitted contract creation
      hash=0x975501f4b6c24a46d13ead5840f40bc03460c6be4139cbd6c3d902e73790796c
      from=0xeDBc36d74d5a1Cd64DB36E53798bd1781f0C4955 nonce=1
      contract=0xfbe4899126470AF8dd4d37e878f0De486a6CFA71 value=0
```

- Iniciando a mineração o contrato será minerado:

```
1 > miner.start()
2 null
3 > null [object Object]
4 Contract mined! address: 0xfbe4899126470af8dd4d37e878f0de486a6cfa71 transactionHash:
      0x975501f4b6c24a46d13ead5840f40bc03460c6be4139cbd6c3d902e73790796c
5 > miner.stop()
```

1.7 *Web3 deployment: Interagindo com o contrato*

- Interagir com o contrato via *Geth console*
  - Após o *deployment* através da sua `ABI` o contrato estará disponível no *console*:

```
1 > valuechecker.
2 valuechecker.Matcher valuechecker.address valuechecker.transactionHash
3 valuechecker._eth valuechecker.allEvents valuechecker.valueEvent
4 valuechecker.abi valuechecker.constructor
5 > valuechecker.address
6 "0xfbe4899126470af8dd4d37e878f0de486a6cfa71"
7 > valuechecker.transactionHash
8 "0x975501f4b6c24a46d13ead5840f40bc03460c6be4139cbd6c3d902e73790796c"
```

Percebam o mesmo *address* e *transactionHash* que foram devolvidos no processo de *deploy*.

- A `ABI` do `valuechecker` está disponível:

```
1 > valuechecker.abi
2 [{
3     anonymous: false,
4     inputs: [{
```

```
        indexed: false,
        internalType: "bool",
        name: "returnValue",
        type: "bool"
    }],
    name: "valueEvent",
    type: "event"
}, {
    inputs: [{
        internalType: "uint8",
        name: "x",
        type: "uint8"
    }],
    name: "Matcher",
    outputs: [{
        internalType: "bool",
        name: "",
        type: "bool"
    }],
    stateMutability: "nonpayable",
    type: "function"
}]
>
```

- A função `Matcher` pode ser invocada para a verificação de valores:

```
> eth.getBalance(valuechecker.address)
0
> valuechecker.Matcher.call(12)
true
> valuechecker.Matcher.call(10)
true
> valuechecker.Matcher.call(5)
false
>
```

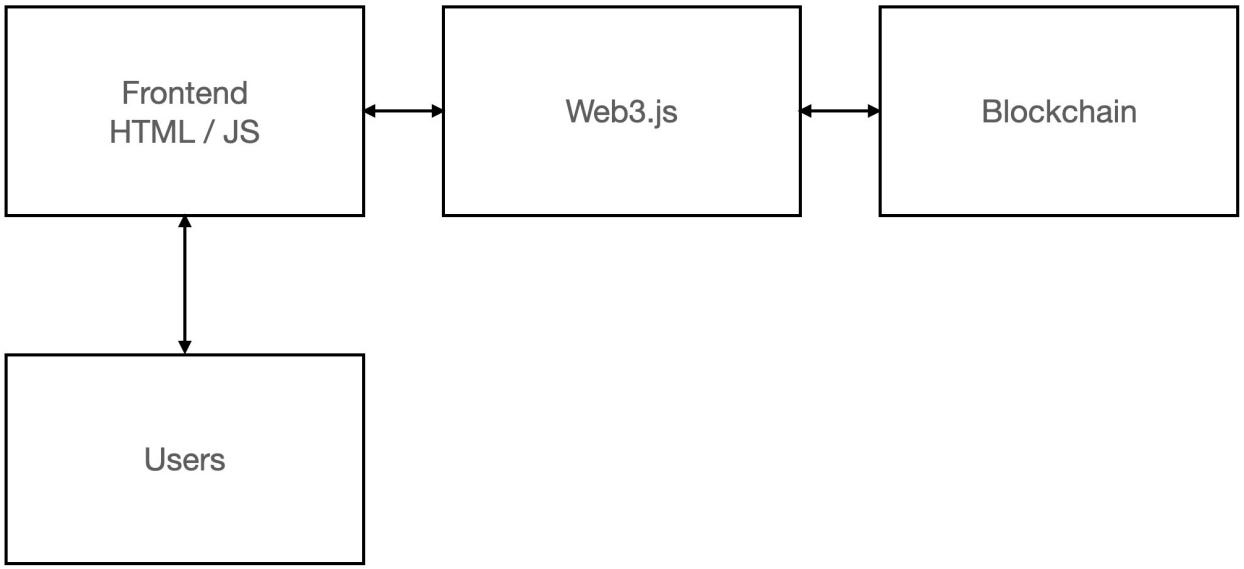
- Interagir com o contrato via um *frontend* web.
- *POST requests* É possível interagir com o `Geth` via `JSON RPC` sobre o `HTTP`. Para esse teste utilizaremos o `curl`. Lembrando que a porta utilizando foi a 8559.
- **Recuperando a lista de contas:** A lista de contas pode ser obtida utilizando o método `personal_listAccounts`, conforme o comando:

```
$ curl --request POST --data
    '{"jsonrpc":"2.0","method":"personal_listAccounts","params": [],"id":4}'
    localhost:8559 -H "Content-Type: application/json"
{"jsonrpc":"2.0","id":4,"result":["0xedbc36d74d5a1cd64db36e53798bd1781f0c4955",
"0x1478d95f8754b3ba7127100dd0bb46578fe7d22a"]}
```

Um objeto `JSON` é retornado com a lista de contas. No comando `curl`, o parâmetro `--request` é usado para especificar que o comando é uma requisição do tipo `POST` e `--data` é usado para especificar os parâmetros e valores. Finalmente, o `localhost:8559` é usando para indicar o endereço que o `HTTP endpoint` do `Geth` está aberto.

1.8 Interagindo com contratos via frontends web

- A interação com *smart contracts* como parte de uma `DApps` é normalmente feito usando uma interface web desenvolvida utilizando `HTML/JS/CSS`. Algumas bibliotecas e *frameworks* como `React`, `Redux`, e `Drizzle`, podem também ser usadas.



1.9 Biblioteca Javascript Web3.js

- Se ainda não instalou a biblioteca `web3.js`, pode instalá-la via `npm` com o comando:

```
1 $ npm install web3
```

A biblioteca `Web.js` disponibiliza alguns módulos, sendo eles:

- **web3-eth:** Ethereum blockchain e smart contracts.
- **web3-shh:** Protocolo `Whisper` (Comunicação e *broadcast* P2P).
- **web3-bzz:** Protocolo `Swarm`, que fornece armazenamento descentralizado.
- **web3-utils:** Fornece funções úteis para o desenvolvimento de `DApps`.
- Criando um servidor http para testar a app.

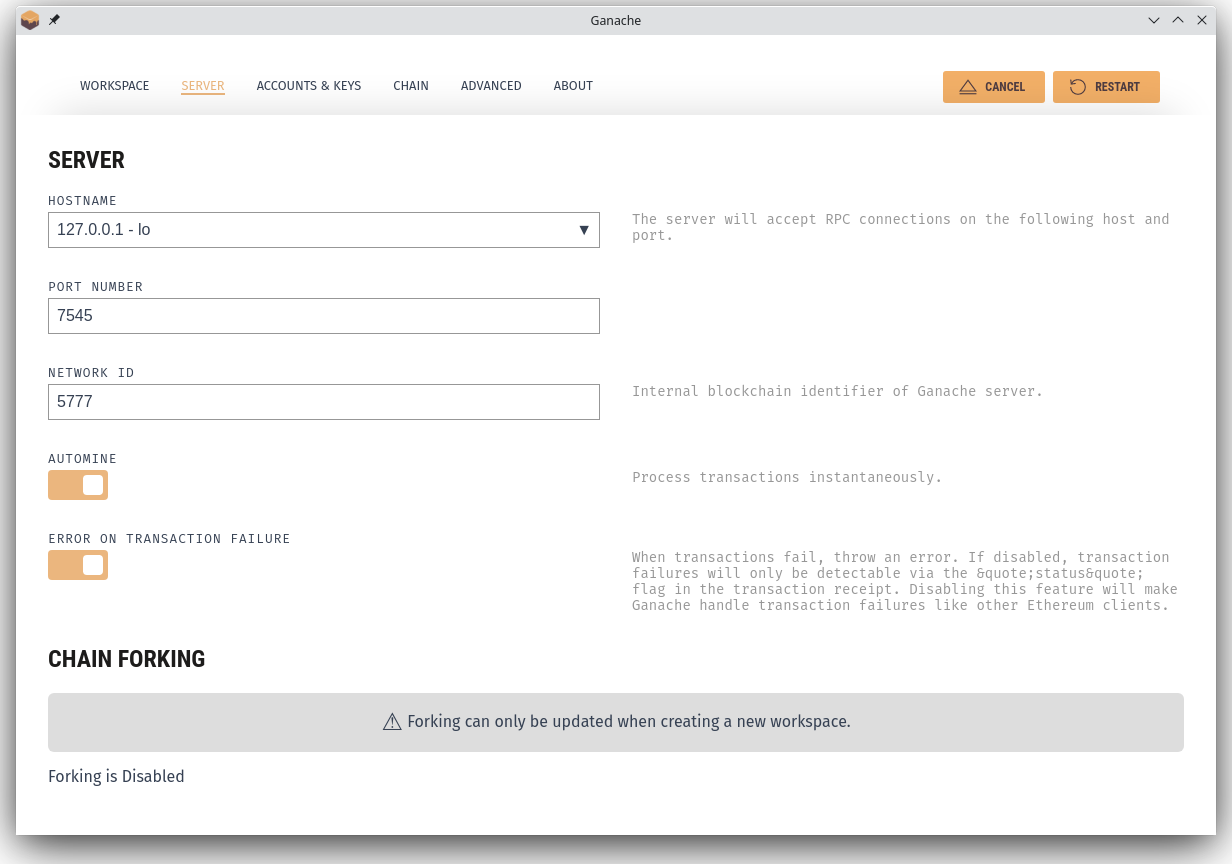
```
# Python 3.x
python3 -m http.server 7777
# If Python version returned above is 2.X
python -m SimpleHTTPServer 7777
```

1.10 Development frameworks

- Installing and initializing Truffle
  - Truffle initialization is perfumed using the Truffle init command, which generates a skeleton structure for a project
- Compiling, testing, and migrating using Truffle
  - Several commands available in Truffle can be used to compile, test and deploy smart contracts

1.11 Configuração do Ganache

- We can use Ganache as a local blockchain to provide the RPC interface.



1.12 *Interagindo com um contrato*

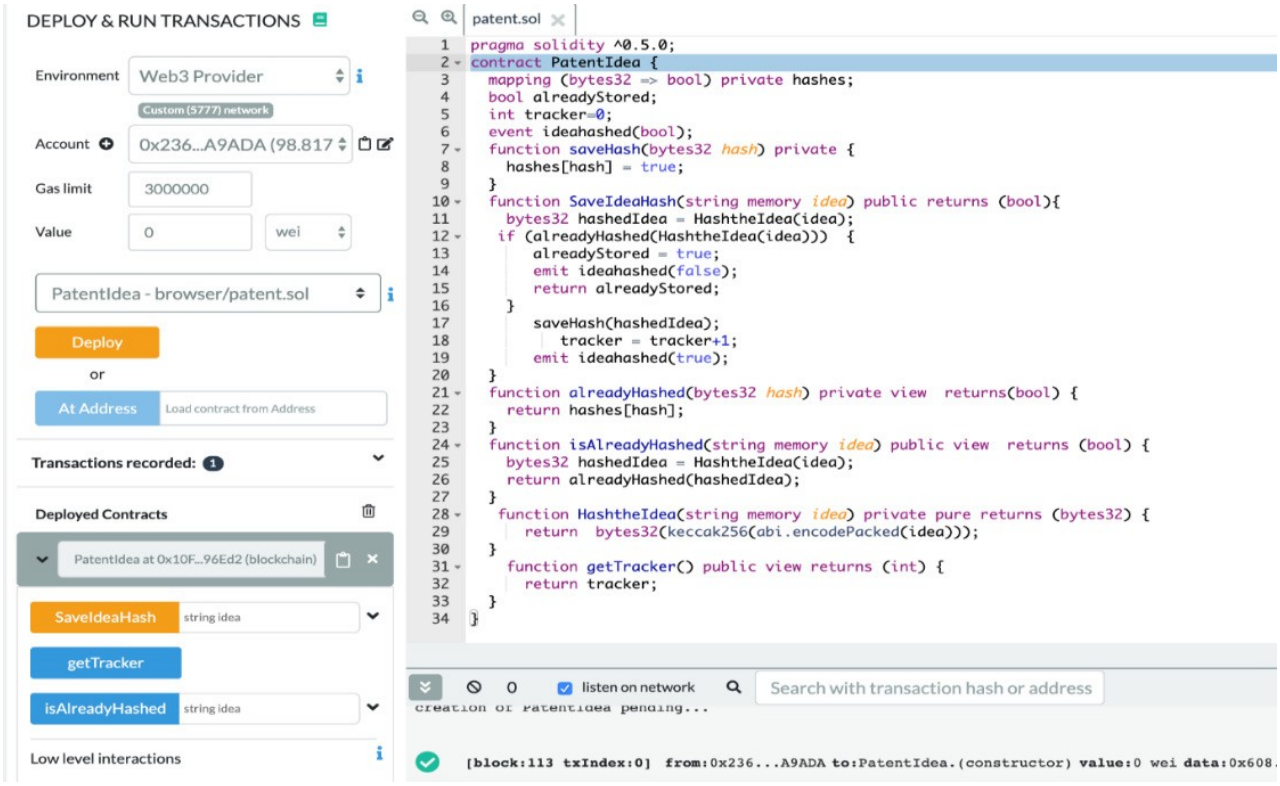
- O console do `Truffle` expõe vários métodos que podem ser usados para interagir com contratos.

```
truffle(development)> MetaCoin.
MetaCoin.__defineGetter__       MetaCoin.__defineSetter__       MetaCoin.__lookupGetter__       MetaCoin.__lookupSetter__
MetaCoin.__proto__              MetaCoin.hasOwnProperty         MetaCoin.isPrototypeOf          MetaCoin.propertyIsEnumerable
MetaCoin.toLocaleString         MetaCoin.valueOf

MetaCoin.apply                  MetaCoin.bind                   MetaCoin.call                   MetaCoin.constructor
MetaCoin.toString

MetaCoin._constructorMethods    MetaCoin._json                  MetaCoin._properties            MetaCoin._property_values
MetaCoin.abi                    MetaCoin.addProp                MetaCoin.address                MetaCoin.arguments
MetaCoin.ast                    MetaCoin.at                     MetaCoin.autoGas                MetaCoin.binary
MetaCoin.bytecode               MetaCoin.caller                 MetaCoin.class_defaults         MetaCoin.clone
MetaCoin.compiler               MetaCoin.configureNetwork       MetaCoin.contractName           MetaCoin.contract_name
MetaCoin.currentProvider        MetaCoin.decodeLogs             MetaCoin.defaults               MetaCoin.deployed
MetaCoin.deployedBinary         MetaCoin.deployedBytecode       MetaCoin.deployedSourceMap      MetaCoin.detectNetwork
MetaCoin.devdoc                 MetaCoin.ens                    MetaCoin.events                 MetaCoin.gasMultiplier
MetaCoin.hasNetwork             MetaCoin.interfaceAdapter       MetaCoin.isDeployed             MetaCoin.legacyAST
MetaCoin.length                 MetaCoin.link                   MetaCoin.links                  MetaCoin.metadata
MetaCoin.name                   MetaCoin.network                MetaCoin.networkType            MetaCoin.network_id
MetaCoin.networks               MetaCoin.new                    MetaCoin.numberFormat           MetaCoin.prototype
MetaCoin.resetAddress           MetaCoin.schemaVersion          MetaCoin.schema_version         MetaCoin.setNetwork
MetaCoin.setNetworkType         MetaCoin.setProvider            MetaCoin.setWallet              MetaCoin.source
MetaCoin.sourceMap              MetaCoin.sourcePath             MetaCoin.timeoutBlocks          MetaCoin.toJSON
MetaCoin.transactionHash        MetaCoin.unlinked_binary        MetaCoin.updatedAt              MetaCoin.updated_at
MetaCoin.userdoc                MetaCoin.web3
```

1.13 *Developing a proof of idea project*



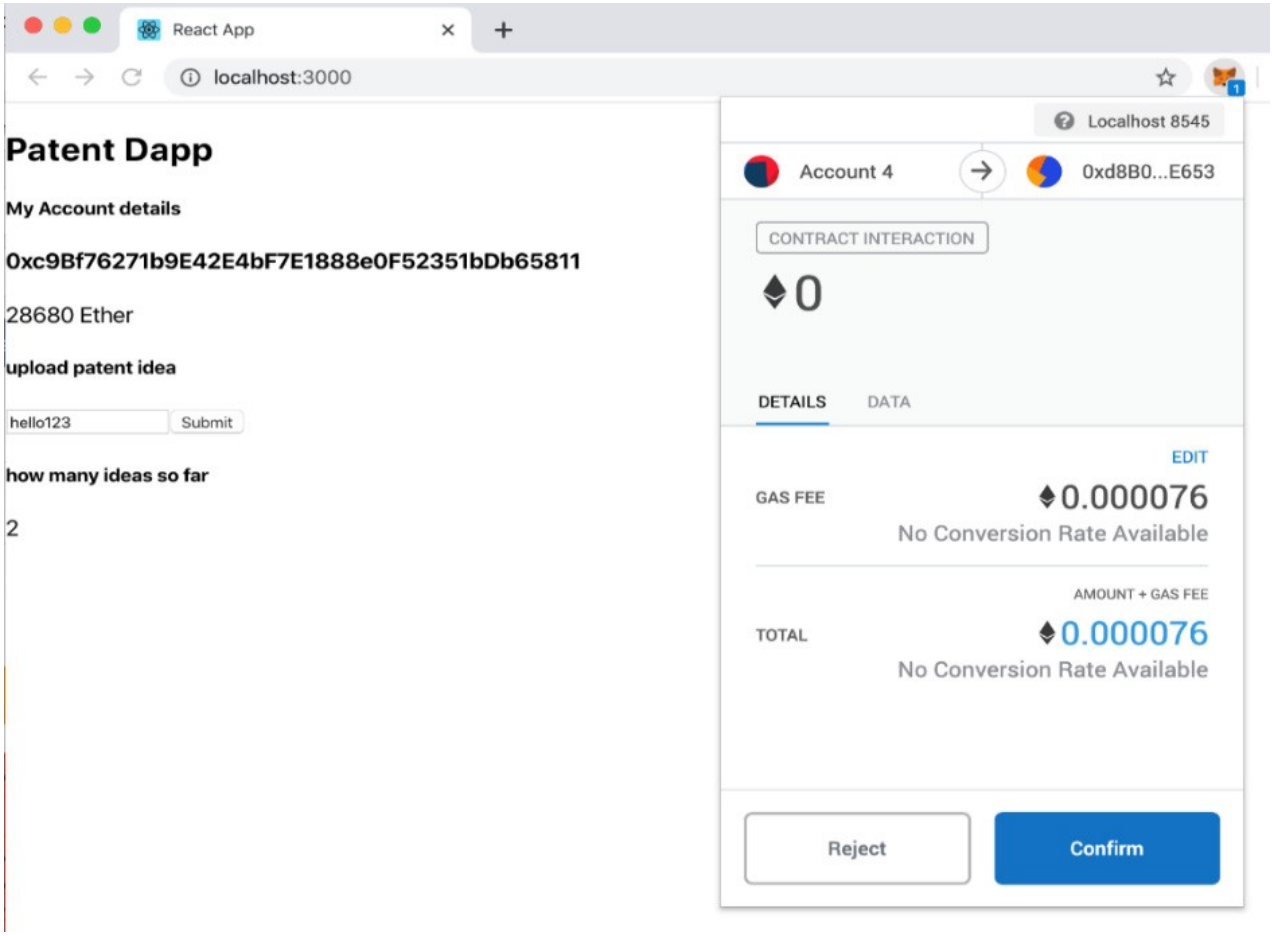
1.14 Creating the ideap project

The necessary steps to create a proof of idea project, as detailed in the core Mastering Blockchain book, are as follows:

- Write the ideap smart contract
- Compile and test it in the Remix IDE
- Deploy to Ganache using Truffle
- Deploy to your network of choice (this is optional)
- Build a web frontend using Drizzle
- Run the DApp!

1.15 Patent DApp

This is the resulting interactive frontend of the proof of idea DApp.



1.16 IPFS

- Traditionally, storage is centralized.
- In order to decentralize the entire blockchain ecosystem, storage services should also be decentralized, and serve as decentralized storage layer of the blockchain.

- DApps can benefit from decentralized storage, where backend data can be stored without fear of censorship or centralized control.

1.17 Atividade

- Instalar as ferramentas do Capítulo e implementar os projetinhos de exemplos.
- Utilizando o `Truffle` baixar o exemplo de projeto `MetaCoin` e fazer o `deploy` no `Ganache`.

1.18 Leitura Recomendada

Leitura Recomendada

Capítulo 15: Introducing Web3
Livro: IMRAN BASHIR. Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

2 Próximas Aulas

2.1 Próximas Aulas

- Desenvolvimento do Projeto.

3 Referências

3.1 Referências

Imran, Bashir. 2018. *Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition*. Packt Publishing. https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1789486&lang=pt-br&site=eds-live&scope=site.