A Simple Login System using Ethereum

The login process will be as follows:

• 1. A user accesses a website that requires him or her to login. When the user is not logged in, the website requests the user to enter his or her Ethereum address.

• 2. The backend for the website receives the address for the user and creates a challenge string and a JSON Web Token (JWT). Both of these are sent back to the user.

• 3. The user sends the challenge string to the Login contract and stores the JWT for later use locally.

• 4. The backend listens for login attempts using the challenge string at the Ethereum network. When an attempt with the challenge string for the right user is seen, it can assume the user has proved his or her identity. The only person that can send a message with an Ethereum address is the holder of the private key, and the only user that knows the challenge string isthe user that received the challenge through the login website.

• 5. The user gets notified or polls the website backend for confirmation of his or her successful login. The user then proceeds to use the JWT

Code :

Login.sol

```
pragma solidity ^0.5.16;

contract Login {
    event LoginAttempt(address sender, string challenge);

    function login(string memory challenge) public {
        emit LoginAttempt(msg.sender, challenge);
    }
}
```

app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const LoginContract = require('./login_contract.js');
const jwt = require('jsonwebtoken');
const cuid = require('cuid');
const cors = require('cors');

// LoginAttempt is the name of the event that signals logins in the
```

```
// Login contract. This is specified in the login.sol file.

const challenges = {};
const successfulLogins = {};

const events = LoginContract.events.LoginAttempt();

events._emitter.on("connected", function(subscriptionId){
    console.log(subscriptionId);
})
.on('data', function(event){
    console.log(event); // same results as the optional callback above
    // If the challenge sent through Ethereum matches the one we generated,
    // mark the login attempt as valid, otherwise ignore it.
    const { sender, challenge } = event.returnValues;
    console.log(challenges);

    console.log('Sender:', sender);
    console.log('Challenge:', challenge);
    if(challenges[sender.toLowerCase()] === challenge) {
        successfulLogins[sender.toLowerCase()] = true;
    }
})
.on('error', function(error, receipt) { // If the transaction was rejected by the network with a receipt,
the second parameter will be the receipt.
    console.log(error);
});


// From here on it's just express.js
const secret = process.env.JWT_SECRET || "my super secret passcode";

const app = express();
// WARNING: CHANGE IN PRODUCTION
app.use(cors({
    origin: 'http://localhost:8080'
```

```
}))
app.use(bodyParser.json({ type: () => true }));

function validateJwt(req, res, next) {
    try {
        req.jwt = jwt.verify(req.body.jwt, secret, {
            algorithms: ['HS256']
        });
        next();
    } catch(e) {
        res.sendStatus(401); //Unauthorized
    }
}

app.post('/login', (req, res) => {
    // All Ethereum addresses are 42 characters long
    if(!req.body.address || req.body.address.length !== 42) {
        res.sendStatus(400);
        return;
    }

    req.body.address = req.body.address.toLowerCase();

    const challenge = cuid();
    challenges[req.body.address] = challenge;

    const token = jwt.sign({
        address: req.body.address,
        access: 'finishLogin'
    }, secret);

    res.json({
        challenge: challenge,
        jwt: token
    });
});
```

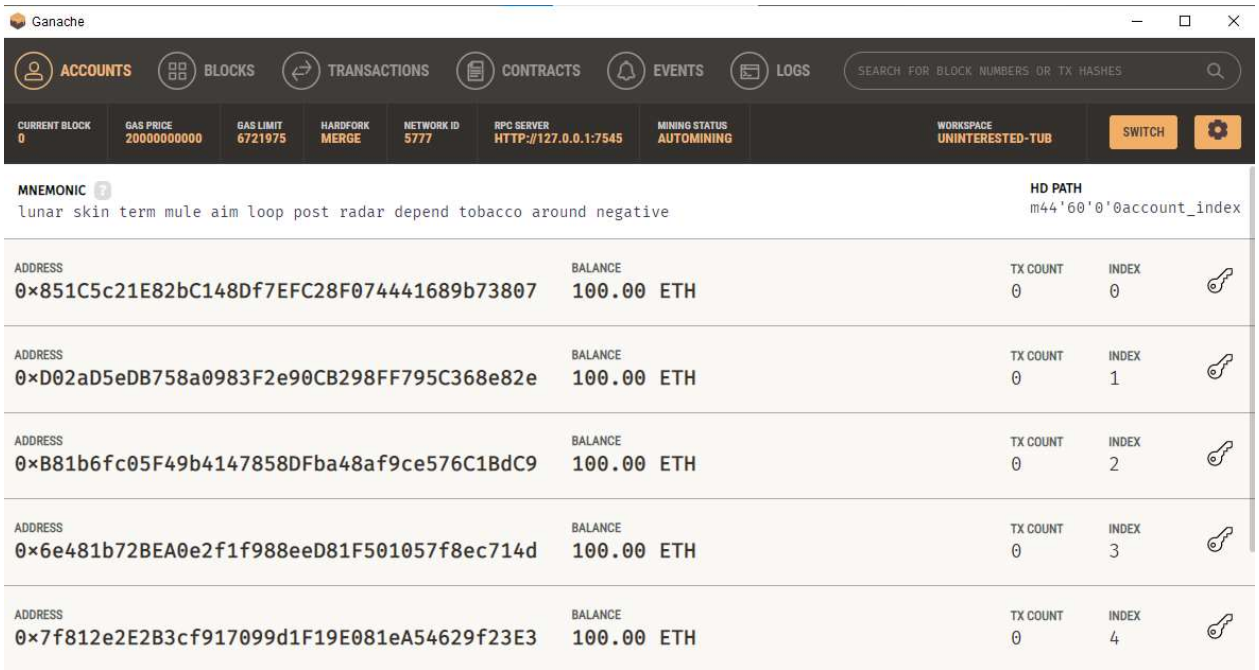
```
app.post('/finishLogin', validateJwt, (req, res) => {
    if(!req.jwt || !req.jwt.address || req.jwt.access !== 'finishLogin') {
        res.sendStatus(400);
        return;
    }

    if(successfulLogins[req.jwt.address]) {
        delete successfulLogins[req.jwt.address];
        delete challenges[req.jwt.address];

        const token = jwt.sign({
            address: req.jwt.address,
            access: 'full'
        }, secret);

        res.json({
            jwt: token,
            address: req.jwt.address
        });
    } else {
        // HTTP Accepted (not completed)
        res.sendStatus(202);
    }
});

app.post('/apiTest', validateJwt, (req, res) => {
    if(req.jwt.access !== 'full') {
        res.sendStatus(401); //Unauthorized
        return;
    }

    res.json({
        message: 'It works!'
    });
});
```

```
app.listen(process.env.PORT || 3000);
```

Setup :

```
D:\Development\Eth_Login>cd solidity

D:\Development\Eth_Login\solidity>truffle init
contracts already exists in this directory...
? Overwrite contracts? (y/N)
Starting init...
================
? Overwrite contracts? Yes
migrations already exists in this directory...
? Overwrite migrations? Yes

> Copying project files to D:\Development\Eth_Login\solidity

Init successful, sweet!

Try our scaffold commands to get started:
  $ truffle create contract YourContractName # scaffold a contract
  $ truffle create test YourTestName         # scaffold a test

http://trufflesuite.com/docs
```

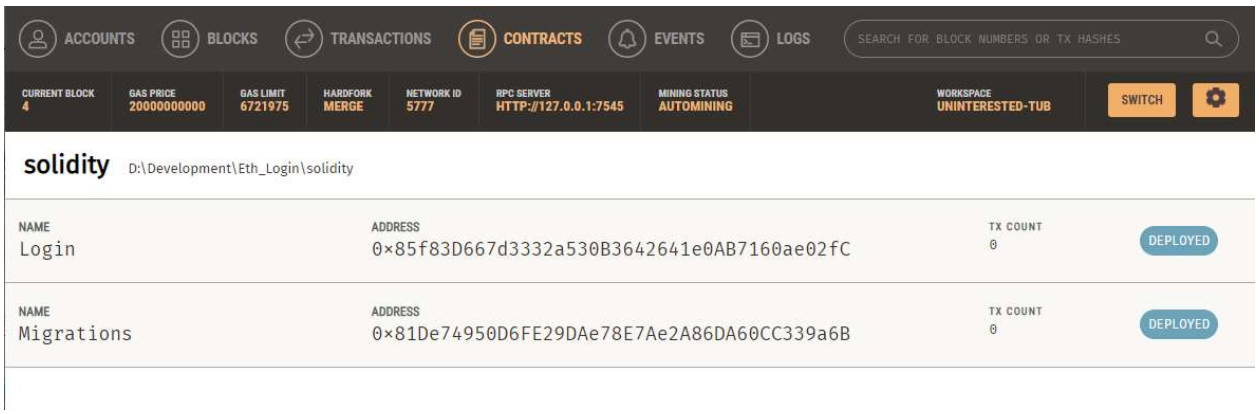
Start ganache for creating blockchain :



Deploy our Smart Contract to Ganache Ethereum blockchain

```
D:\Development\Eth_Login\solidity>truffle migrate

Compiling your contracts...
===========================
> Compiling .\contracts\Login.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to D:\Development\Eth_Login\solidity\build\contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
Network up to date.
```

```
Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0xe89d31730f5f7347dbe87d4d8d25066532f02004de44367755bb5a50ce8af1fd
   > Blocks: 0            Seconds: 0
   > contract address:    0x25bE38beEF8aE525Fa93dA02A29936813a7A4b7D
   > block number:        5
   > block timestamp:     1632729967
   > account:             0x2B0203dc4a866D763C787e1BD8Fc23086641a95f
   > balance:             99.99059872
   > gas used:            191943 (0x2edc7)
   > gas price:           20 gwei
```



Starting our backend :

```
D:\Development\Eth_Login\backend>npm i express web3 body-parser cors
cuid express-jwt jsonwebtoken
```

```
D:\Development\Eth_Login\backend>node app.js
Latest Ethereum Block is  7n
0x8
{
  address: '0x85f83d667d3332a530b3642641e0ab7160ae02fc',
  blockHash: '0x74da410ab6c3a3263f11024723ca1d699a7add66779e77ae11a0b5a20b3d6ca6',
  blockNumber: 8n,
  data: '0x000000000000000000000000851c5c21e82bc148df7efc28f074441689b7380700000000000000000000000000000000000000000000000000
000000000000000000400000000000000000000000000000000000000000000000000000000000000000000000000000000019636c70333063786b39303030306c38387a386
36636376e333500000000000000',
  logIndex: 0n,
  removed: false,
  topics: [
    '0xdeb017956c894eba079e6a630f5cfec256256dc05616ad89f1eb825abef45963'
  ],
  transactionHash: '0x3c976f4c50088ed77755222cc4e7107156a50c3b04bf30a7e4182990c904df8b',
  transactionIndex: 0n,
  returnValues: {
    '0': '0x851C5c21E82bC148Df7EFC28F074441689b73807',
    '1': 'clp30cxk90000l88z8cf67n35',
    __length__: 2,
    sender: '0x851C5c21E82bC148Df7EFC28F074441689b73807',
    challenge: 'clp30cxk90000l88z8cf67n35'
  },
  event: 'LoginAttempt',
```

Frontend :

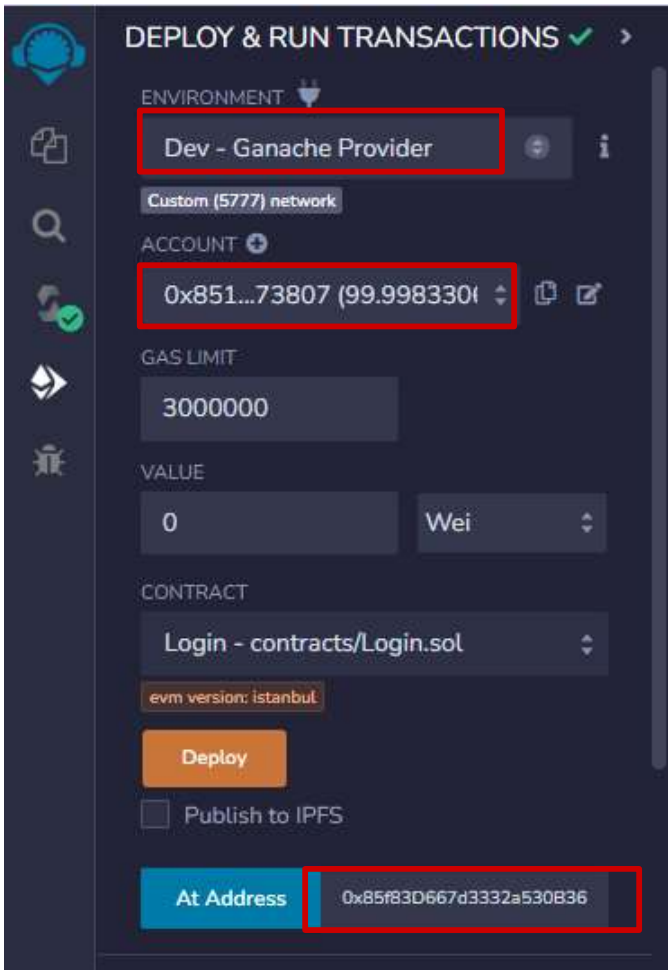
```
D:\Development\Eth_Login\frontend>npm init
D:\Development\Eth_Login\frontend>npm i webpack jquery
D:\Development\Eth_Login\frontend>npm i serve
D:\Development\Eth_Login\frontend> serve -p 8080

Serve -> to static serve a Html page
```

Ethereum address | Login

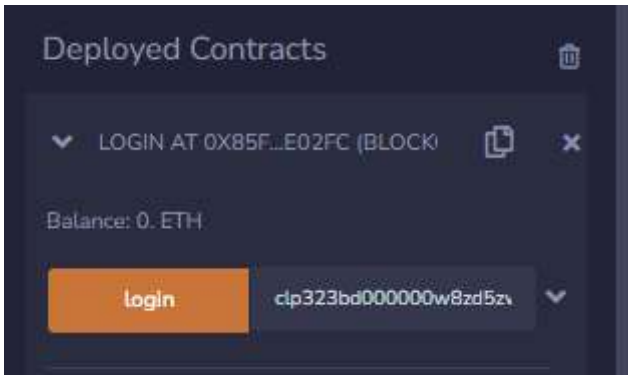
Send the following text to the login contract using an Ethereum client or wallet and then wait:

clp2uteuv0000y48zeqwadk6b

Open Remix IDE



Add Login text to interact with smart contract



Logged in as 0x851c5c21e82bc148df7efc28f074441689b73807

Testing API access: It works!

Transaction mined :

← BACK

TX 0×3c976f4c50088ed77755222cc4e7107156a50c3b04bf30a7e4182990c904df8b

SENDER ADDRESS
0×851C5c21E82bC148Df7EFC28F074441689b73807

TO CONTRACT ADDRESS
0×85f83D667d3332a530B3642641e0AB7160ae02fC

CONTRACT CALL

VALUE	GAS USED	GAS PRICE	GAS LIMIT	MINED IN BLOCK
0.00 ETH	24519	2852170687	24519	8

TX DATA
0×3a7d280c0000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000000000001
9636c70333063786b39303030306c38387a38636636376e333500000000000000

CONTRACT

CONTRACT
Login

ADDRESS
0×85f83D667d3332a530B3642641e0AB7160ae02fC

FUNCTION
login(challenge: string)

INPUTS
clp30cxk90000l88z8cf67n35

Conclusion : We successfully implemented a Smart Contract based Login system in Ethereum.

https://github.com/auth0-blog/ethereum-login-sample/tree/master
https://auth0.com/blog/an-introduction-to-ethereum-and-smart-contracts-part-2/