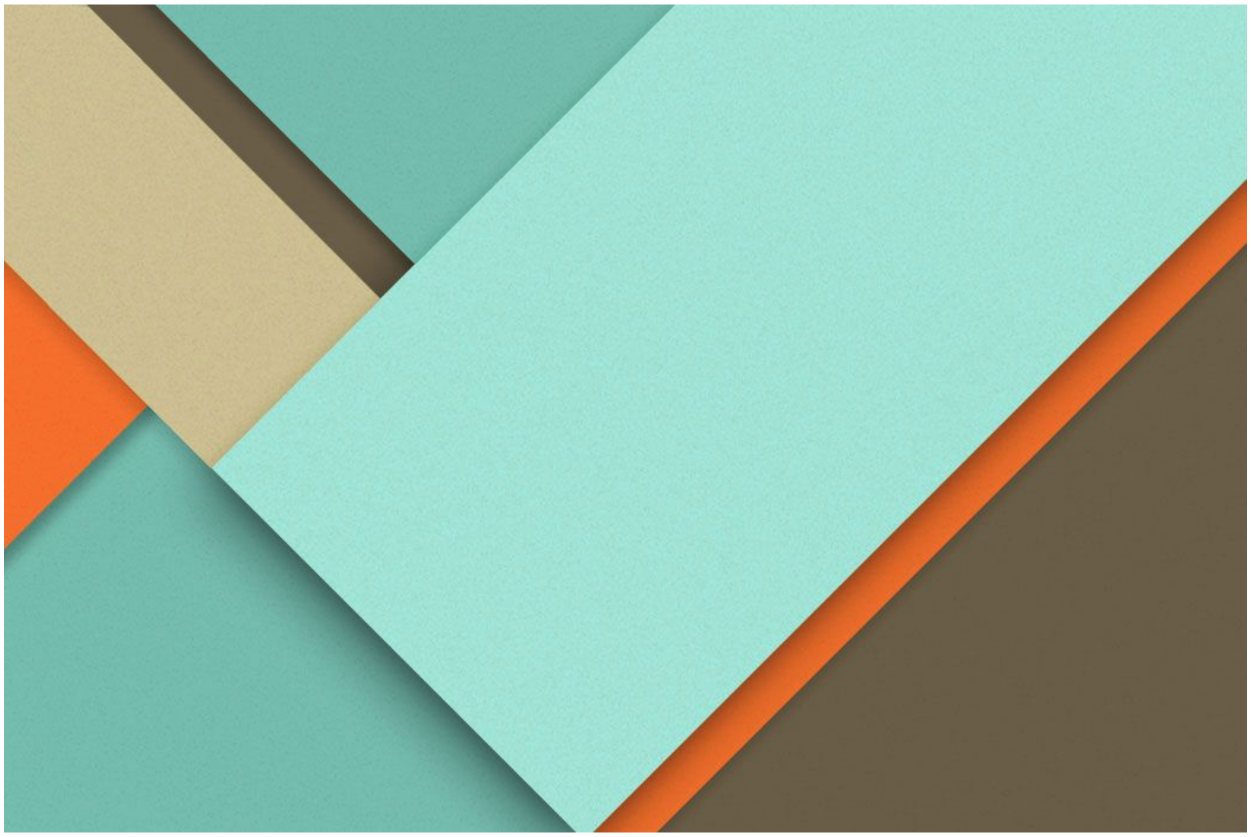


DIGITAL IDENTITY - KYC/AML

ETHEREUM SMART CONTRACT

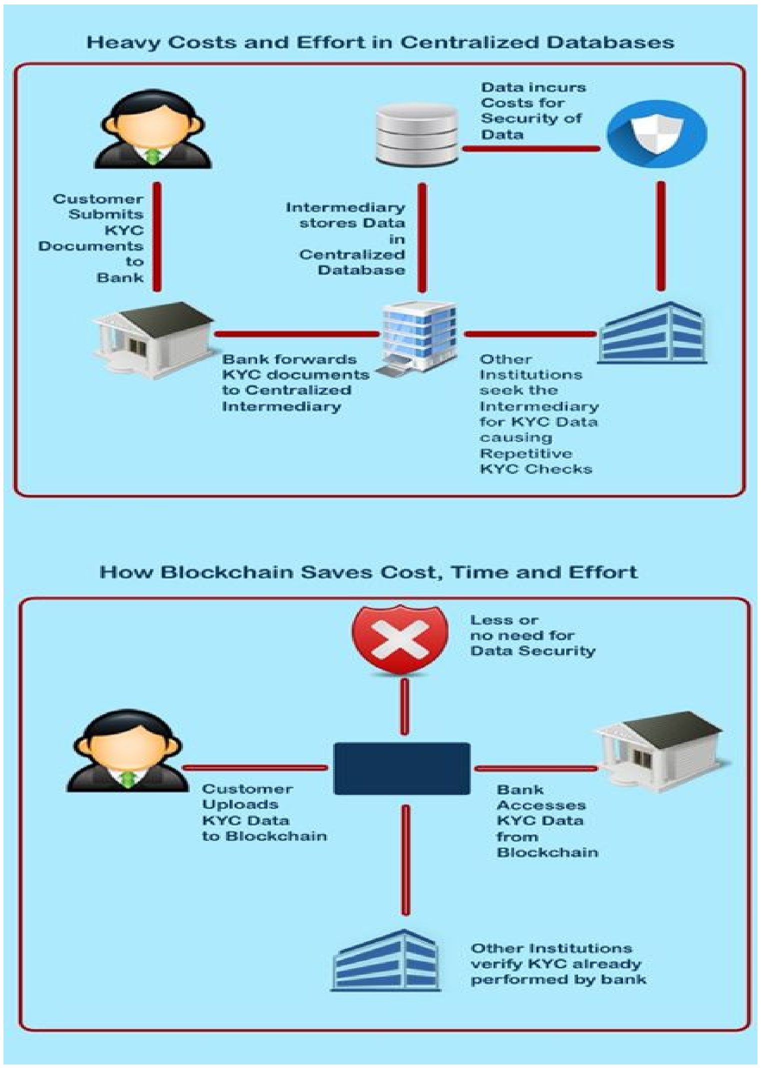
Submitted by
Hemant Gupta

Remix IDE is used for writing & testing the smart contract for this project

Overview

The project aims to provide a solution to store the KYC data in a standardized form on blockchain where banks and other institutions can directly access the KYC data to reduce redundancy and high cost for security of data.

Through this smart contract, the user can upload the KYC data which will be validated by concerned authority (may be UIDAI or IT department). After validation, the user can share the blockchain account with the banks and other institutions for direct retrieval of KYC data.


Heavy Costs and Effort in Centralized Databases
Data incurs Costs for Security of Data
Customer Submits KYC Documents to Bank
Intermediary stores Data in Centralized Database
Bank forwards KYC documents to Centralized Intermediary
Other Institutions seek the Intermediary for KYC Data causing Repetitive KYC Checks
How Blockchain Saves Cost, Time and Effort
Less or no need for Data Security
Customer Uploads KYC Data to Blockchain
Bank Accesses KYC Data from Blockchain
Other Institutions verify KYC already performed by bank

Concept

1. Initially, User have to submit KYC data for verification.
2. Concerned Authority validates the KYC data and mark the status as “Verified” if the data matches with their records, otherwise mark as “Verification Failed” so that user can again apply for verification after submitting correct & latest data.
3. User can now share the Account ID ,i.e, Ethereum address directly to financial institutions and other organizations so that they can access the verified KYC data eliminating the need of collecting physical proofs.
4. At any point of time, user can apply for modification in the KYC data. Please keep in mind that applying for modification will lead to “Unverified” status on that account. This will also send the modified KYC data for re-verification.

Some Edge Cases Covered

1. User can not submit the KYC data more than once.
2. Aadhar and PAN Card Number Validation Included.
3. User can not verify the KYC data of any account.
4. There is a special Ethereum Address reserved exclusively for approving KYC data known as KYC Approver Address.
5. KYC Approver Address can not submit the KYC data.

References Used

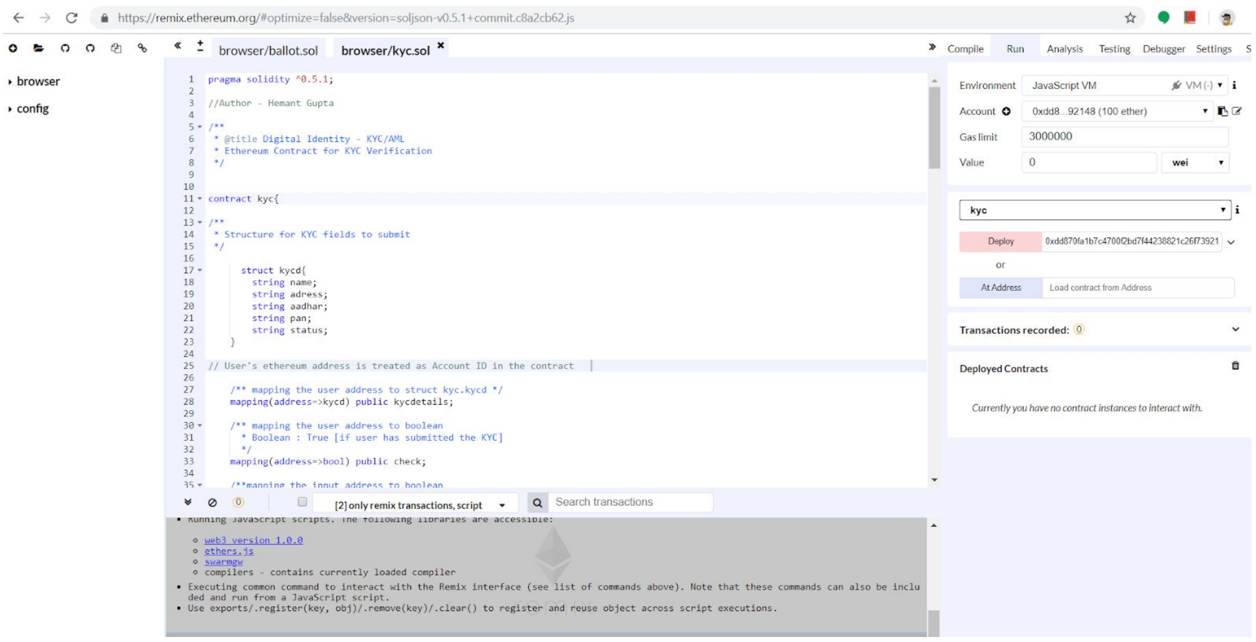
I. https://www.pluralsight.com/courses/ethereum-blockchain-developing-applications

II. https://medium.com/coinmonks/solidity-tutorial-returning-structs-from-public-functions-e78e48efb378

III. https://solidity.readthedocs.io/en/v0.5.1/

ScreenShots

1. Deploying the Smart Contract with KYC Approver Address as input



```
pragma solidity ^0.5.1;

//Author - Hemant Gupta

/**
 * @title Digital Identity - KYC/AML
 * Ethereum Contract for KYC Verification
 */


contract kyc{

/**
 * Structure for KYC fields to submit
 */

      struct kycd{
        string name;
        string adress;
        string aadhar;
        string pan;
        string status;
    }

// User's ethereum address is treated as Account ID in the contract

    /** mapping the user address to struct kyc.kycd */
    mapping(address=>kycd) public kycdetails;

    /** mapping the user address to boolean
      * Boolean : True [if user has submitted the KYC]
      */
    mapping(address=>bool) public check;

    /**mapping the input address to boolean
```

Environment: JavaScript VM   VM (-)
Account: 0xdd8...92148 (100 ether)
Gas limit: 3000000
Value: 0 wei

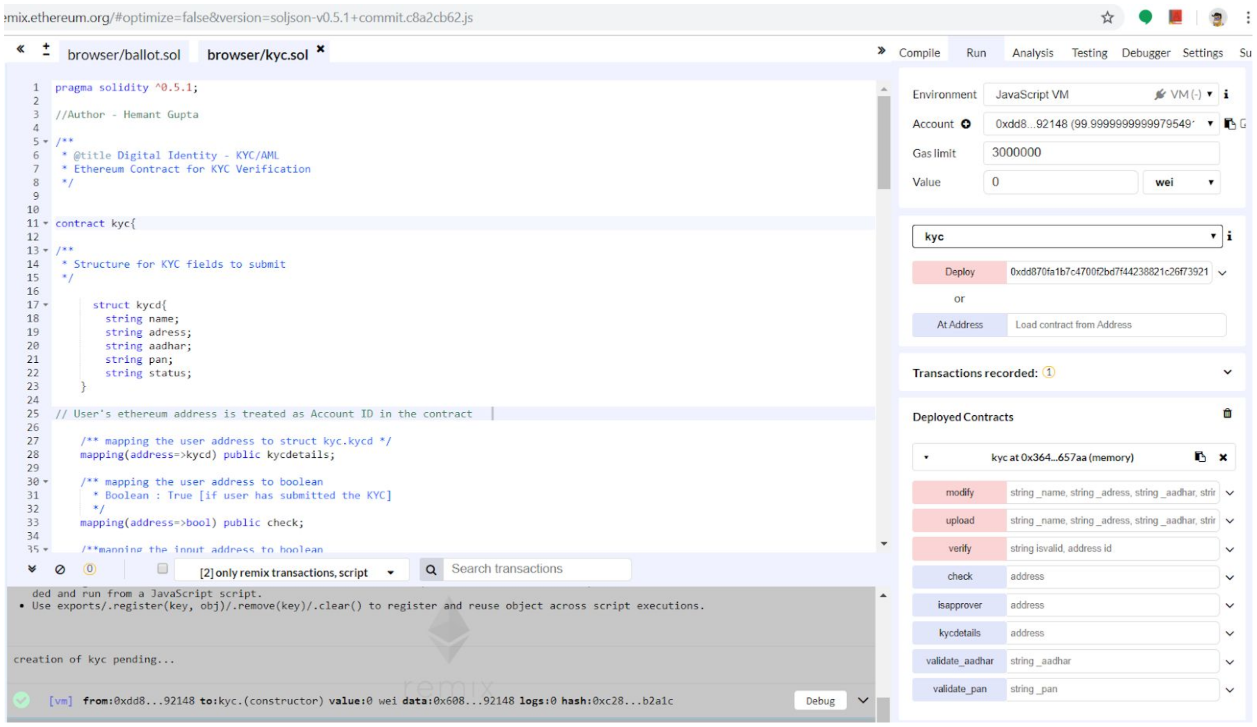
kyc

Deploy: 0xdd870fa1b7c4700f2bd7f44238821c26f73921

Currently you have no contract instances to interact with.

```
• Running JavaScript scripts. The following libraries are accessible:
   ○ web3 version 1.0.0
   ○ ethers.js
   ○ swarmgw
   ○ compilers - contains currently loaded compiler
• Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be inclu
  ded and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

2. After deploying, a set of functions are visible on the screen



```
pragma solidity ^0.5.1;

//Author - Hemant Gupta

/**
 * @title Digital Identity - KYC/AML
 * Ethereum Contract for KYC Verification
 */

contract kyc{

/**
 * Structure for KYC fields to submit
 */

    struct kycd{
        string name;
        string adress;
        string aadhar;
        string pan;
        string status;
    }

// User's ethereum address is treated as Account ID in the contract

    /** mapping the user address to struct kyc.kycd */
    mapping(address=>kycd) public kycdetails;

    /** mapping the user address to boolean
     * Boolean : True [if user has submitted the KYC]
     */
    mapping(address=>bool) public check;

    /**mapping the input address to boolean
```

```
ded and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.

creation of kyc pending...

[vm] from:0xdd8...92148 to:kyc.(constructor) value:0 wei data:0x608...92148 logs:0 hash:0xc28...b2a1c
```

Environment: JavaScript VM
Account: 0xdd8...92148 (99.9999999999979549
Gas limit: 3000000
Value: 0 wei

kyc
Deploy: 0xdd870fa1b7c4700f2bd7f44238821c26f73921
or
At Address: Load contract from Address

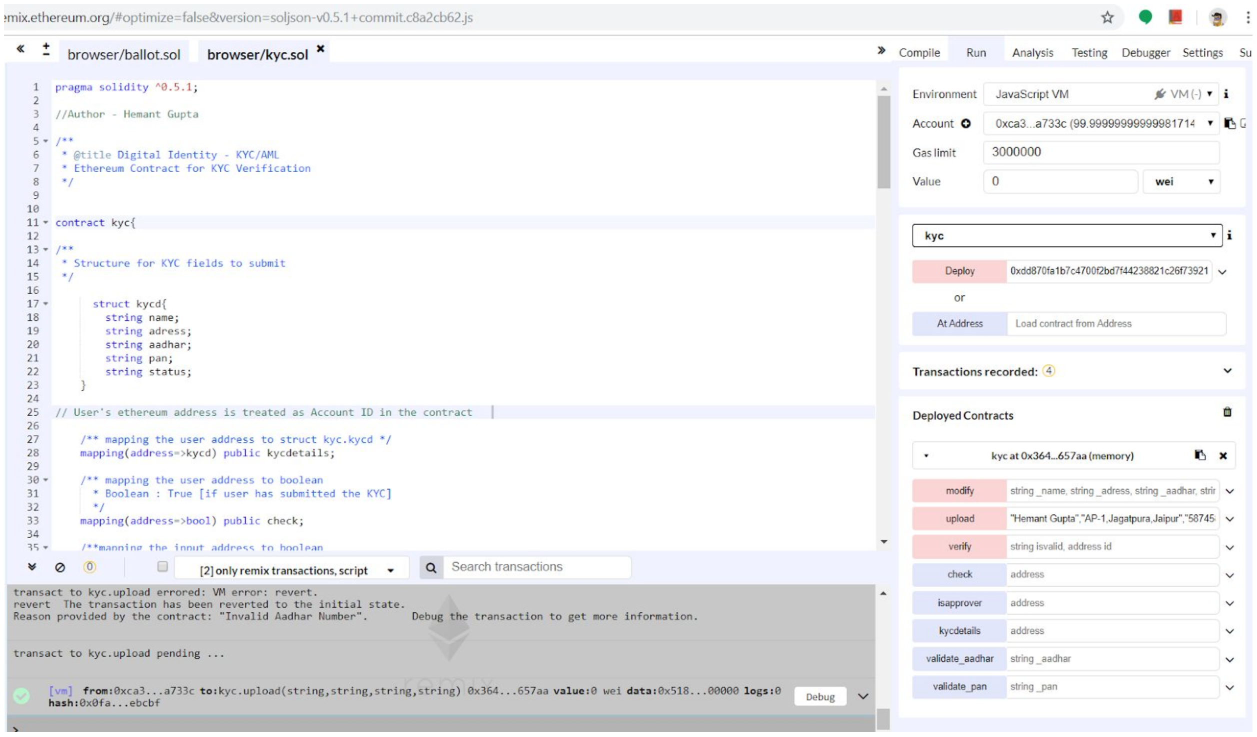
Transactions recorded: 1

Deployed Contracts
kyc at 0x364...657aa (memory)

modify	string _name, string _adress, string _aadhar, strir
upload	string _name, string _adress, string _aadhar, strir
verify	string isvalid, address id
check	address
isapprover	address
kycdetails	address
validate_aadhar	string _aadhar
validate_pan	string _pan

mix.ethereum.org/#optimize=false&version=soljson-v0.5.1+commit.c8a2cb62.js
browser/ballot.sol
browser/kyc.sol
Compile
Run
Analysis
Testing
Debugger
Settings
Su
[2] only remix transactions, script
Search transactions
Debug

3. Uploading the KYC data with user's ethereum address



```
pragma solidity ^0.5.1;

//Author - Hemant Gupta

/**
 * @title Digital Identity - KYC/AML
 * Ethereum Contract for KYC Verification
 */

contract kyc{

/**
 * Structure for KYC fields to submit
 */

    struct kycd{
        string name;
        string adress;
        string aadhar;
        string pan;
        string status;
    }

// User's ethereum address is treated as Account ID in the contract

    /** mapping the user address to struct kyc.kycd */
    mapping(address=>kycd) public kycdetails;

    /** mapping the user address to boolean
     * Boolean : True [if user has submitted the KYC]
     */
    mapping(address=>bool) public check;

    /**mapping the input address to boolean
```

```
transact to kyc.upload errored: VM error: revert.
revert  The transaction has been reverted to the initial state.
Reason provided by the contract: "Invalid Aadhar Number".      Debug the transaction to get more information.

transact to kyc.upload pending ...

[vm] from:0xca3...a733c to:kyc.upload(string,string,string,string) 0x364...657aa value:0 wei data:0x518...00000 logs:0
hash:0x0fa...ebcbf
```

Environment: JavaScript VM
Account: 0xca3...a733c (99.99999999999981714
Gas limit: 3000000
Value: 0 wei

kyc
Deploy: 0xdd870fa1b7c4700f2bd7f44238821c26f73921
or
At Address: Load contract from Address

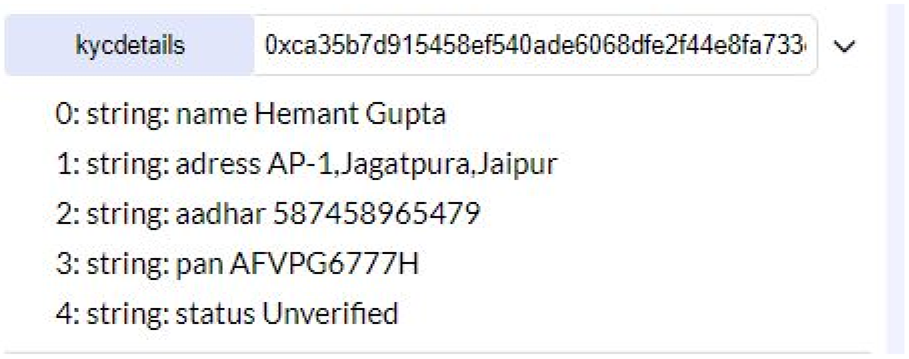
Transactions recorded: 4

Deployed Contracts
kyc at 0x364...657aa (memory)

modify	string _name, string _adress, string _aadhar, strir
upload	"Hemant Gupta","AP-1,Jagatpura,Jaipur","58745
verify	string isvalid, address id
check	address
isapprover	address
kycdetails	address
validate_aadhar	string _aadhar
validate_pan	string _pan

mix.ethereum.org/#optimize=false&version=soljson-v0.5.1+commit.c8a2cb62.js
browser/ballot.sol
browser/kyc.sol
Compile
Run
Analysis
Testing
Debugger
Settings
Su
[2] only remix transactions, script
Search transactions
Debug

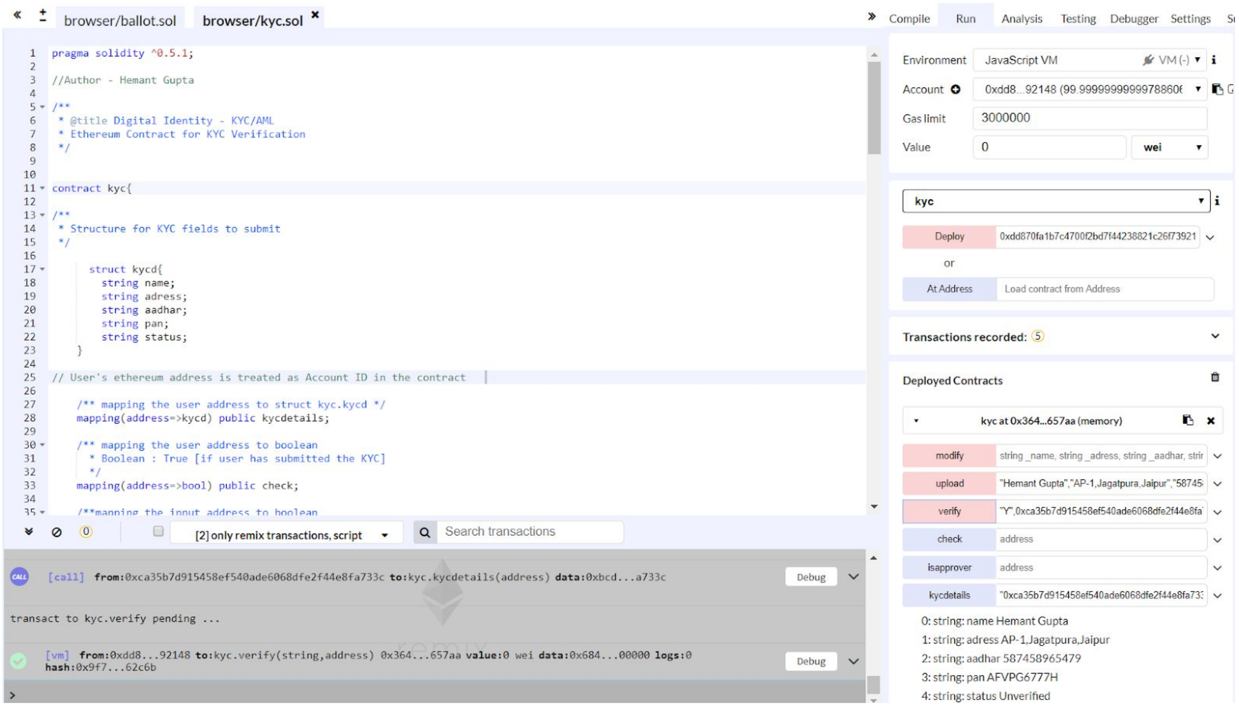
4. Getting the KYC data stored with user’s ethereum address as parameter



kycdetails 0xca35b7d915458ef540ade6068dfe2f44e8fa733

```
0: string: name Hemant Gupta
1: string: adress AP-1,Jagatpura,Jaipur
2: string: aadhar 587458965479
3: string: pan AFVPG6777H
4: string: status Unverified
```

5. Validating the KYC data with KYC Approver Ethereum Address



browser/ballot.sol | browser/kyc.sol

```
pragma solidity ^0.5.1;

//Author - Hemant Gupta

/**
 * @title Digital Identity - KYC/AML
 * Ethereum Contract for KYC Verification
 */


contract kyc{

/**
 * Structure for KYC fields to submit
 */

    struct kycd{
      string name;
      string adress;
      string aadhar;
      string pan;
      string status;
    }

// User's ethereum address is treated as Account ID in the contract

    /** mapping the user address to struct kyc.kycd */
    mapping(address=>kycd) public kycdetails;

    /** mapping the user address to boolean
      * Boolean : True [if user has submitted the KYC]
      */
    mapping(address=>bool) public check;

    /**mapping the input address to boolean
```

```
[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c to:kyc.kycdetails(address) data:0xbcd...a733c

transact to kyc.verify pending ...

[vm] from:0xdd8...92148 to:kyc.verify(string,address) 0x364...657aa value:0 wei data:0x684...00000 logs:0
hash:0x9f7...62c6b
```

Environment: JavaScript VM
Account: 0xdd8...92148 (99.9999999999978860
Gas limit: 3000000
Value: 0 wei

kyc
Deploy: 0xdd870fa1b7c4700f2bd7f44238821c26f73921
At Address: Load contract from Address

Transactions recorded: 5

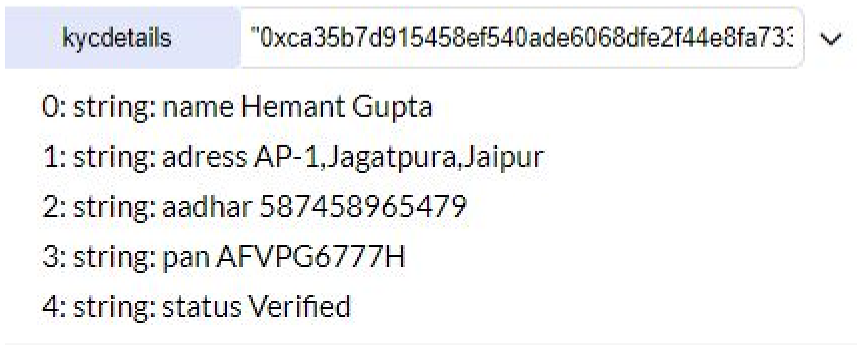
Deployed Contracts

kyc at 0x364...657aa (memory)

modify	string _name, string _adress, string _aadhar, strir
upload	"Hemant Gupta","AP-1,Jagatpura,Jaipur","58745
verify	"Y",0xca35b7d915458ef540ade6068dfe2f44e8fa
check	address
isapprover	address
kycdetails	"0xca35b7d915458ef540ade6068dfe2f44e8fa73

```
0: string: name Hemant Gupta
1: string: adress AP-1,Jagatpura,Jaipur
2: string: aadhar 587458965479
3: string: pan AFVPG6777H
4: string: status Unverified
```

6. Modifying the KYC data with user's ethereum address as parameter



kycdetails "0xca35b7d915458ef540ade6068dfe2f44e8fa733

```
0: string: name Hemant Gupta
1: string: adress AP-1,Jagatpura,Jaipur
2: string: aadhar 587458965479
3: string: pan AFVPG6777H
4: string: status Verified
```