Contents

[Purpose: 2](#_Toc122367501)

[Setup for testing 2](#_Toc122367502)

[Deploying the contract: 2](#_Toc122367503)

[Address Information 2](#_Toc122367504)

[Admin Functions 3](#_Toc122367505)

[Create Banks 3](#_Toc122367506)

[Remove Bank 3](#_Toc122367507)

[Modify Bank isAllowedToVote 3](#_Toc122367508)

[Smart Contract Flow 4](#_Toc122367509)

[Bank Functions 4](#_Toc122367510)

# Purpose Of this document:

This document is to list the steps to test the KYC.sol contract in remix IDE.

# Setup for testing

Figure 1: Deploy Contract

## Deploying the contract:

1. Initialize the hardhat node on cmd or Visual Studio code terminal
2. Open and compile the contract in Remix
3. In Deploy and run transactions, change environment to ‘Hardhat Provider’ and click ‘OK’ on the alert that appears.
4. Click on ‘Deploy’

This will deploy the contract on the local Hardhat network.

## Address Information

1. Admin:

In above steps, if ‘ACCOUNT’ has not been changed, then the address ‘0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266’ will be the **Admin** for the application.

1. Banks:

From the default address present I am assuming the address ‘Account #1’ to ‘Account #5’ will be the bank address.

**Admin Address:**

Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)

Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

**Bank Address:**

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 (10000 ETH)

Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC (10000 ETH)

Private Key: 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

Account #3: 0x90F79bf6EB2c4f870365E785982E1f101E93b906 (10000 ETH)

Private Key: 0x7c852118294e51e653712a81e05800f419141751be58f605c371e15141b007a6

Account #4: 0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65 (10000 ETH)

Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a

Account #5: 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc (10000 ETH)

Private Key: 0x8b3a350cf5c34c9194ca85829a2df0ec3153be0318b5e2d3348e872092edffba

**Note:** I will be writing this document with the assumption that above accounts are Admin and Bank accounts. You may use other accounts, but change them in the steps that follow accordingly.

# Admin Functions

## Create Banks

Figure 2 Deployed contract's Interface functions

1. Ensure ‘Admin’ address is selected in the ‘ACCOUNT’ dropdown.
2. Format for creating banks is (string)**bankName**, (address)**bankAddress**, (string)**registrationNumber**
3. In the text box near orange box labelled ‘addBank’ enter the following one by one and click ‘addBank’ to create the banks
   1. Bank A, 0x70997970C51812dc3A010C7d01b50e0d17dc79C8,reg1
   2. Bank B, 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC,reg2
   3. Bank C, 0x90F79bf6EB2c4f870365E785982E1f101E93b906 ,reg3
   4. Bank D, 0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65 ,reg4
   5. Bank E, 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc ,reg5
4. Repeating the same bank address with different name and registration number will throw an alert ‘Gas estimation failed’.

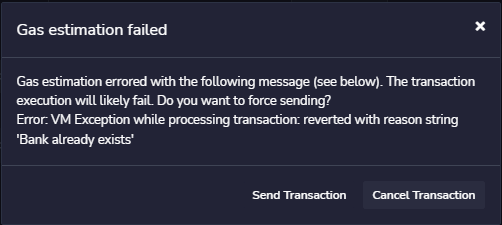


Figure 3Alert text from remix IDE

## Remove Bank

1. In the text box next to ‘removeBank’ enter the bank address to be deleted
2. Click on ‘removeBank’
3. If bank is not present in the blockchain, an error alert similar to ‘Figure 3’ will be thrown

## Modify Bank isAllowedToVote

Figure 4Bank details before modification

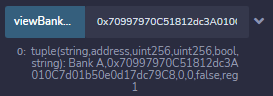
1. In the text box near the blue ‘viewBankDetails’ enter the target bank address and note the details. In Figure 4: …,true,reg1 indicates that the isAllowedToVote is true
2. In the textbox near orange bank\_isAllowedToVote enter the target bank address and false to change the voting status as: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8,false
3. Click on bank\_isAllowedToVote
4. Click on viewBankDetails from step1 with the target bank address.

Figure 5Bank details after modification

1. Note that the value has now changed to …,false,reg1. This indicates bank is not allowed to vote

# Smart Contract Flow

The sections that follow will be according to the ‘Smart Contract Flow’ as described in ‘Creation and Verification of KYC Over Blockchain‘.

**Note:**

1. Change the address in ‘ACCOUNT’ to any one of the bank address created in the previous sections. I am using the address ‘0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC’ for this document
2. This account will not be able to access the ‘Admin’ functions
3. I am using readable string instead of hashes to make testing easy

## Bank Functions

These functions can only be performed by an address designated as a bank.

### Add Customer

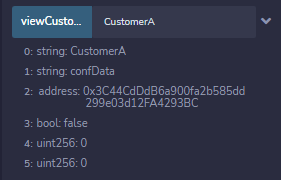
1. In the textbox next to the orange button labelled ‘addCustomer’ enter (string)customer name and (string)customer data. Eg. CustomerA,confData

Figure 6 view customer details

1. Click on addCustomer
2. If the customer exists an alert similar to Figure 3 will be thrown
3. To verify the customer data, in the text box near blue ‘viewCustomer’ enter the customer name. It will return all the customer details including votes and KYC status.
4. The output order is:

0: customer name

1: customer data

2: bank address

3: KYC status

4: Upvotes

5: Downvotes

### Create KYC Request

Figure 7Customer details and KYC information after processing request successfully

1. In the textbox next to the orange button labelled ‘addRequest’ add the KYC request information as: (string)customer name, (string)customer data. Eg. CustomerA,confData
2. Click on addRequest
3. If the kyc request already exists an alert similar to Figure 3 will be thrown
4. Passing customer name to the blue viewKYC will display KYC details

### Verify KYC

1. This may be done with the same bank address or another bank address
2. Enter (string) customer name in the textbox next to the orange verifyKYC box.
3. If KYC is verified successfully, in viewCustomer the value corresponding ‘3:’ will change to true and Upvote counter ‘4:’ will increment by 1.
4. If KYC fails, ‘3:’ will be false and downvote counter ‘4:’ will increment by 1. It wil also raise a complain on the bank and run internal function to check if bank is faulty. The admin can view the outcome of this using admin functions.

### View Customer

All banks can run this function.

1. In the textbox near the blue button ‘viewCustomer’ enter the customer name
2. Click the viewCustomer button to get the customer details.
3. The output order is:

0: customer name

1: customer data

2: bank address

3: KYC status

4: Upvotes

5: Downvotes

## Upvote and Downvote Customer

The upvote and downvote function are internal and not accessible via the Remix IDE interface.

1. The functions run when the verifyKYC function is called. When verification is successful customer is upvoted; and downvoted when verification fails
2. On failure of verifyKYC banks to choose to report the bank using the reportBank function by passing the ethAddress and bank name. On reporting a bank internal function validBank will run to check if bank is authentic and will affect the voting status accordingly
3. Finally, the internal function validateCustomer is called to modify the kyc status based on the Upvotes, Downvotes and logic provided in the requirements