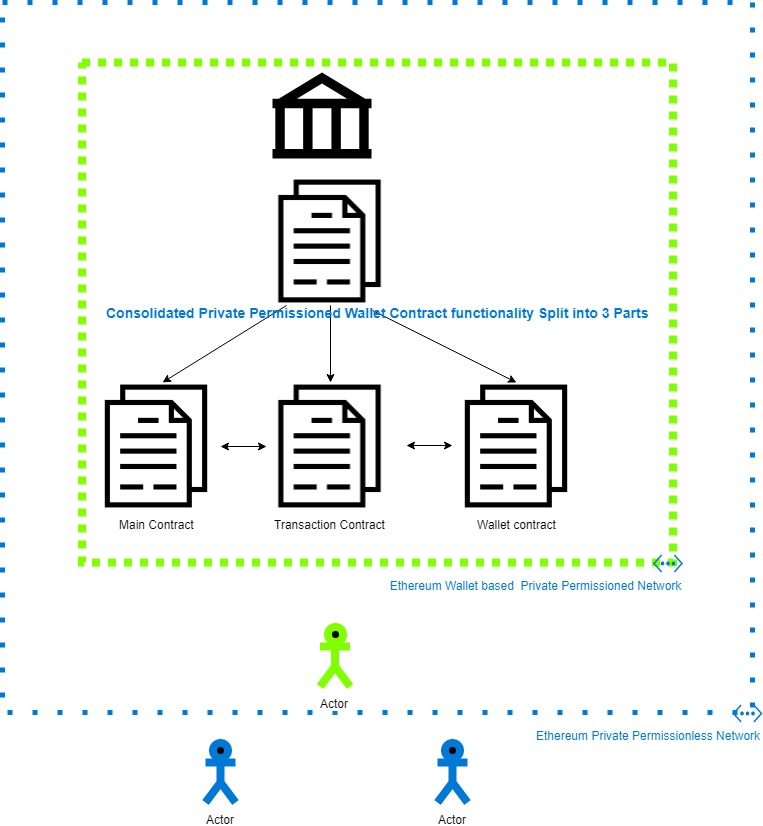
**Deploying Smart Contract on the Ethereum Network**

**Since we have on the Network all as Ethereum Node Peers as it’s a private Network which is permission less, we aim to create a Smart Contract which is based on ERC20 standard with private tokens.**

**Smart Contract Overview:**

In this illustration of the below image After creating the private permission less network, we create an internal private permissioned wallet-based system through the smart contract Approach



**Some Questions?**

**When talking about the permissioned wallet based out of the original “Private Ethereum Network”, we are creating a mechanism for each of the Car Nodes as well as Admin Nodes to participate in an ecosystem of Token which issues the Token in terms of the Odometer Values.**

**What is Smart Contract?**

According to Wiki **“A smart Contract is a computer protocol intended to digitally facilitate, verify or enforce negotiation of contract , It enables the performance of a transaction without intermediaries”**

**Here we leverage this idea to create our own wallet where we perform the following:**

1. **Segregation of the user level ==🡺 Authorization**

**We enforce two type of users inside the wallet 1) Leader or Admin 2) Users or Car Nodes**

1. **Enforce Transaction Mechanism =🡺 Consensus on the Wallet for each internal modification of state of the Wallet through Consensus based Voting Approach Approach among the Admin Nodes.**
2. **Enforce Creation of Token and Token Transfer Rights =🡺Token Mechanism to the Admins and Only Receiving Rights to the User Nodes**
3. **Change the Authorization levels, Introduction of New Nodes into the wallet, Token transfer to a node all take place though the consensus-based voting mechanism of the Admins**

**Function Listing in the Smart Contract:**

1. **CRUD Operation of Any Node**
2. **Authorization Operation on any Node**
3. **Token Issuance in the Network**
4. **Token Transfer in the Network**
5. **Transaction Mechanism in the Network**
6. **Transaction Voting through Consensus and fulfillment of each transaction**
7. **Transaction Queue**
8. **Change the consensus Percentage for any of the above operation**
9. **Store the Asset Details of each of the Node**
10. **Store the Authorization Levels of each of the Node**
11. **Store the transaction Details of each of them and garbage collect on fulfillment**

**Where does smart contract reside?**

**Smart contract are created as .sol files and then they are compiled and stored on a specific NON WALLET ADDRESS in the Private Network which we created on.**

**Eg: Contract Address 0xf9eae0f545c0d5b1f1538df4746e46bf2c90d381**

**Who can invoke the smart contract?**

**Any one can invoke the smart contract through a Node by knowing the ABI (Application Binary Interface) which is obtained on compiling of the smart contract before deploying on the network.**

* **Compile the Smart Contract**
* **Get the ABI of Smart Contract for the client Node to Interact with the contract**
* **Deploy the Smart Contract after compilation**
* **Get the Address of the Smart Contract to invoke by its address**
* **Anyone can invoke the smart contract using CALL or SEND Operation of the Ethereum Framework**
* **CALL is used to operate on Methods which don’t do any manipulation or in layman terms they are used to invoke “READ ONLY METHODS”**
* **SEND is used to operate on Methods which modify the internal state variable of the Smart Contract involving a gas cost**
* **Since Anyone can reach the smart contract who has the ABI and the Address, restriction to the invoking of the smart contract is designed internal to the smart contract as we have done in our smart contract.**
* **Our initial users or ADMINS of the smart contract are specified in the Constructor while deploying.**
* **This entails to addition of other Nodes through the Admin Nodes which were specified during the creation of the Smart contract.**

**Can I destroy the Smart contract?**

**No its also consensus based from the required amount of Admins for which we have set it up in the internal smart contract design which allows to delete a smart contract after the approval of all the Admin**

**Why Split Each Smart Contract?**

**24576** is the max byte limit for each smart contract according to the Ethereum Improvement Proposal standard EIP 170 <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md> which places a hard rule on the max size of the smart contract which can e deployed at an address.

**What Each Smart Contract Does?**

**MAIN.SOL =🡺**

* **Main Contract is the Abstract Layer of the Smart contract Interaction which hides the underlying complexities of the Smart contract distributed between other two smart contract.**
* **This contract is the first point of contact for any Node willing to place any kind of transaction on the Wallet**
* **All Interaction with the other smart contract TRANSACTION & WALLET happens through this smart contract**
* **This will be the last contract to be deployed on Network after Transaction Contract & Wallet contract as its constructor will hold the address of the Transaction & Wallet Contract Address**

**TRANSACTION.SOL 🡺**

* **This is the CORE CONSENUS & TRANSACTION QUEUE ENGINE which creates the workflow of how a transaction is place on the network for:**
* **a) Operation on Node Enrollment**

**b) Operation on Node Permission Elevation**

**c) Operation on Token Issuance**

**d) Operation on Odometer Value Requested and Transfer of Token**

**e) Creation of Voting Mechanism for each transaction in the Network requesting approvals according to the set consensus percentage level.**

**Consensus Values which are setup on a predetermined basis and which we can be modified post deployment in the Smart contract are:**

|  |  |
| --- | --- |
| **Consensus Type** | **Operation Concerned** |
| **Participant Consensus** | **CRUD on Participant Role Transaction** |
| **Leader Consensus** | **CRUD on Leader Role Transaction** |
| **Follower Consensus** | **CRUD on Follower Role Transaction** |
| **Delete Transaction Consensus** | **Remove any Transaction on Network considered invalid** |
| **Token Supply Consensus** | **Token Distribution among the Admin Nodes Transaction** |
| **Token Transfer Consensus** | **Token Transfer or Odometer Value Transfer Transaction** |
| **Token Retract Consensus** | **Token Bounce in case of any issue in Transaction** |
| **Overall Consensus** | **Consensus to be reached to change of any of the above consensus values** |

* **Then according to the transaction, it then calculates the required signature needed from the Admin Nodes and then fulfills the transaction if it receives it or else waits for it by putting the transaction in the queue**
* **Lifecyle of each of the Transaction is maintained here:**

**Creation of the transaction**

**Calculation of the required consensus signatures**

**Collection of the signatures from each admin Nodes**

**Check if the consensus is satisfied**

**If YES =🡺 FULFILL, the transaction in the WALLET Smart contract**

**Else NO 🡺 Continue to maintain the transaction in the queue**

**Other Operations:**

* **Check the Validity of each signature from each Admin Node to remove duplicity and authentication**
* **Check the pending transaction**
* **Finalize the transaction and place a secure operation by invoking the WALLET Contract**

**WALLET.SOL 🡺**

* **This Smart Contract operates closely with Transaction Contract as the outcome of any transaction are either stored or decided upon the interaction with Wallet Contract.**
* **It maintains the following storage:**

|  |
| --- |
| **List of Participants in the Wallet** |
| **List of Followers in the Wallet** |
| **List of Admin/Leader in the Wallet** |
| **Mapping of Details of Each Participant in Network with their Token Balance** |
| **Mapping of Each Participant with their permission in Network** |
| **Balance Maintenance of each Node** |

* **All the Operation Mentioned in Main Node finally finalizes in this Smart contract After the Consensus Mechanism setup in the Transaction Smart contract**

**How each smart Contract Interacts with Smart Contract?**

**Each Smart Contract Address invoke the Other through OOPS Concept of Interface present in Ethereum Virtual Machine.**

**Interface Relation are like this:**

|  |  |
| --- | --- |
| **Contract Address** | **Uses Another Contract through Interface Mechanism** |
| **Main Contract** | **Wallet & Transaction** |
| **Transaction Contract** | **Wallet** |
| **Wallet Contract** | **None** |

**Contract Dependency:**

**∏ 🡪 Depends on Relationship**

**Main Contract ∏ { Wallet Contract, Transaction Contract }**

**Transaction Contract ∏ { Wallet Contract }**

**Wallet Contract ∏ { }**

**Security Aspects of Smart Contract?**

Smart Contract are prone to security attacks as they can be invoked by anyone on the network which is not traditionally secured by any authentication mechanism.

But we handle the security in different way:

1. **External Restriction:**

External Restriction is not possible as anyone can invoke the smart contract at will if they have the requisite ABI as earlier mentioned. But this is of minor concern as no one can modify the binary which resides in the private network as it is an immutable transaction on the network

1. **Internal Restriction:**

This restriction is possible by all means as when we create the smart contract in the constructor we specify the initial admin who would be part of the initial network.

These in turn allow the other nodes to be added to the network through consensus mechanism.

As more no of Admin Nodes, the criteria of consensus becomes a tough barrier to pass though as their signatures cannot be mutilated and these unique signatures or Votes for each transaction counts very important to validate and finalize a transaction

1. **Allowing only known smart contract intra-interaction:**

Since some methods are made public for other external smart contract to reach it, this poses a serious problem as there is a probability to introduce invalid data in the network thereby circumventing the internal restriction.

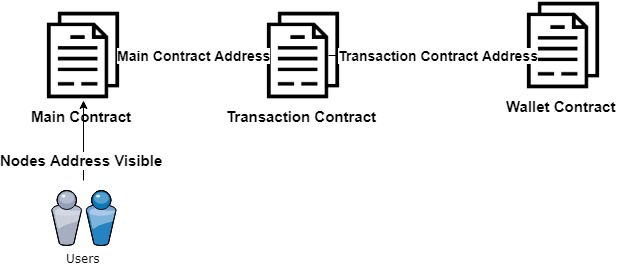
These public methods are tightened during after the deployment process, immediately to distribute the valid smart contract address to each of them as the external caller of the address will be a trusted smart contract and not a malicious node.

This value of setting the trusted external contract address callee is set as a one-time function which cannot be modified post that.

1. **Checking each Method Access against the Permission Level of the Caller (Node**):

Finally, we then place the usual authorization level check for each method invoke since the calls on the main contract will be directly from the nodes’s address and not like an external contract address.

To explain further:



Here we see that the Client Can invoke the main smart contract using its own Ethereum node Address, but to Access the other Contract it has to be done from verified Main Contract Address which restricts the Callee Just to the Verified MAIN contract Address or TRANSACTION contract Address.

**Deploying of Smart Contract Procedure:**

**Having explained the Smart Contract Overview mechanism, we then delve into the practical deployment of the Smart contract.**

**Choice of Deployment Tools:**

**In current Ethereum Technology suite we have many compilers by which you can compile and deploy the Solidity based Smart contract saved in .sol format.**

**Smart Contracts are available @** <https://gitlabee.dt.renault.com/swlabs/blockchain/ethereum_poo/tree/master/api_admin1/odometerapi/blockchainapi/ethereum/contracts>

**They are:**

1. **Main.sol**
2. **Transaction.sol**
3. **Wallet.sol**

**Tools:**

**Truffle: Truffle Framework enables deploy of small smart contracts but it fails to compile properly in standard smart contracts which are of considerable size and also is unstable and variable during compiling meaning it does not generate ABI properly and misses some functions. So we don’t use it**

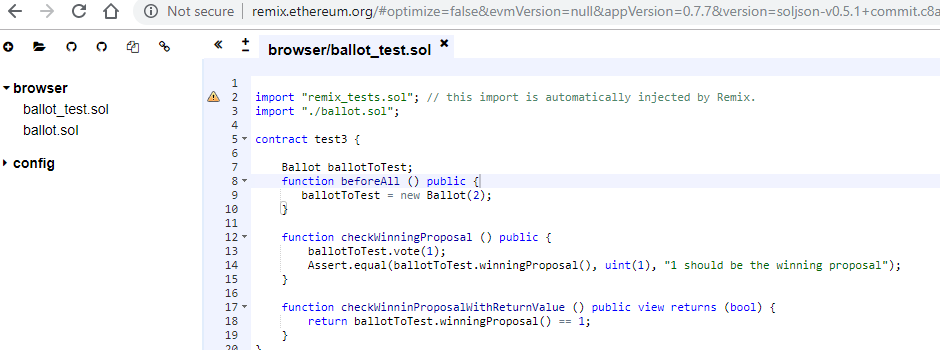
**SolcJs compiler: SolcJS compiler has also same problems as Truffle, moreover it is not updated to current Ethereum Standard and compiles based on old standard which is a problem. We avoid this as well.**

**RemixIDE : Remix IDE is the best option to go now as its easier to compile , deploy and also interact with Smart contract. I used REMIX IDE on non-https version to interact with the nodes present in the internal network as the HTTPs version restricts the communication to the internal network nodes.**

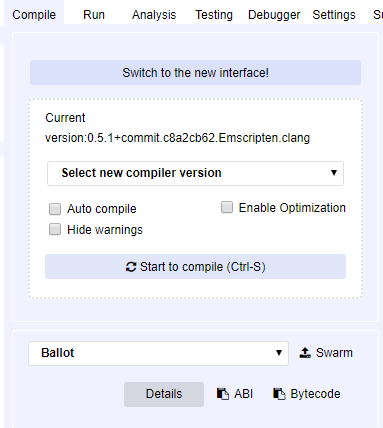
**Link:** [**http://remix.ethereum.org/#optimize=false&evmVersion=null&appVersion=0.7.7&version=soljson-v0.5.1+commit.c8a2cb62.js**](http://remix.ethereum.org/#optimize=false&evmVersion=null&appVersion=0.7.7&version=soljson-v0.5.1+commit.c8a2cb62.js)

**Usage of Tool:**

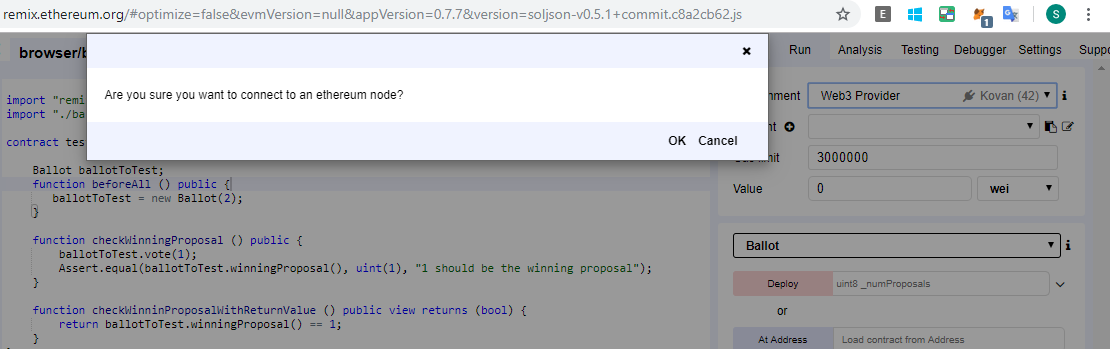
**We paste our smart contract on the browser:**



**Then we can press Ctrl+S to compile each smart contract which also generates ABI Code on the right.**

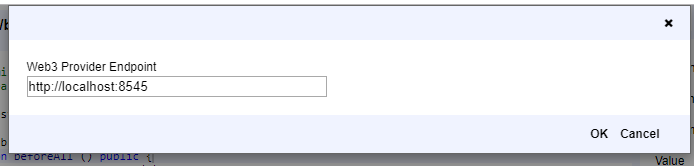


**Then we switch to the Run Tab on the Right and then select the environment under web3 provider.**



**Here it prompts to connect to the local node in your private Network, which we click OK**

**We specify the local Node URL with the port for connecting to it.**

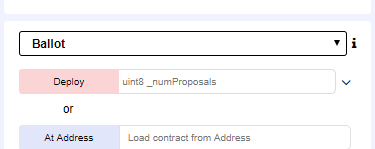


**And then we are connected to the local Node, we should notice that it should automatically connect without any problems and your Node Address ACCOUNT will be reflected in the Remix IDE Tool.**

**Step1: Deployment of Wallet contract:**

**We first deploy wallet contract as it has no dependencies on other smart contract.**

**After we compile the Smart contract using Ctrl+S , we then swtich to Run Tab and then notice that the name of smart contract is pre-filled like this.**



**Then we go ahead, and we need to specify the initial set of Admins in the constructor before we deploy the smart contract.**

**We specify Admin as an array in constructor:**

**["0xaa43515b48c36a51f158cc8b941900b8aa6fb885","0x22a8c699ff1081c31e32ef0ec9266694e415fcd5"]**

**Then we click deploy and we after some gas spent we find that the transaction is submitted to the network and then address is displayed which is the address of the smart contract.**

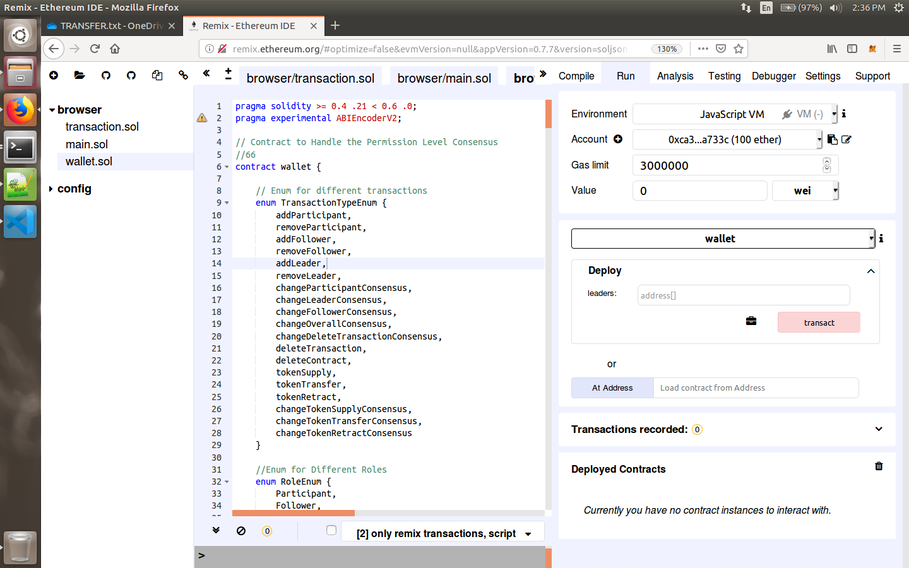
**Step 2: Deploy Transaction Contract:**

**2) We follow the same procedure for the subsequent Contracts, but the next one to deploy is Transaction Contract where it needs the address of the previously deployed WALLET Contract Address**

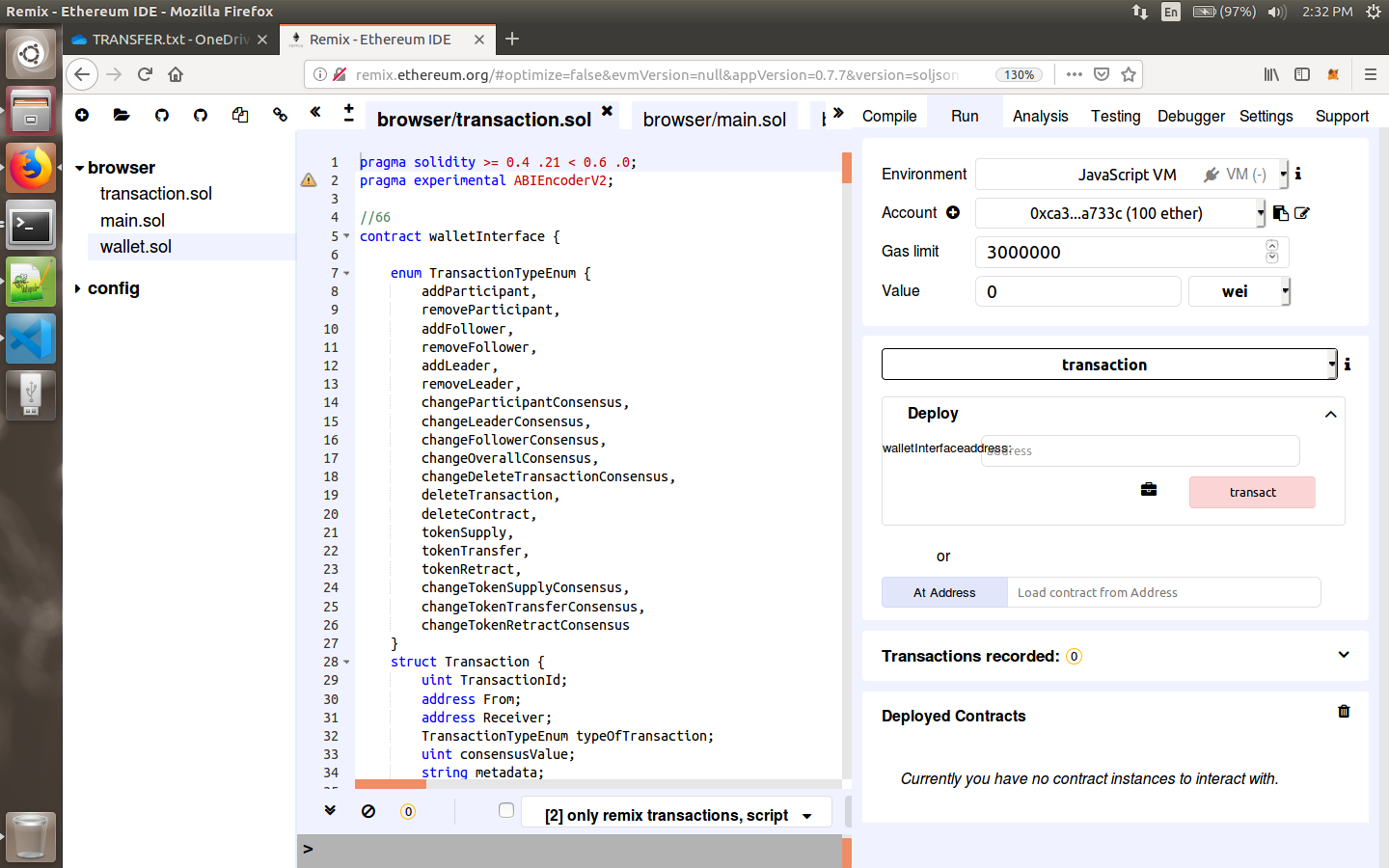
**3) We follow the similar procedure for MAIN Contract , but here we specify both the address of the WALLET & TRANSACTION contract Address in the constructor.**

**For eg: Screenshots of the RMIX Compiler showing the constructor:**

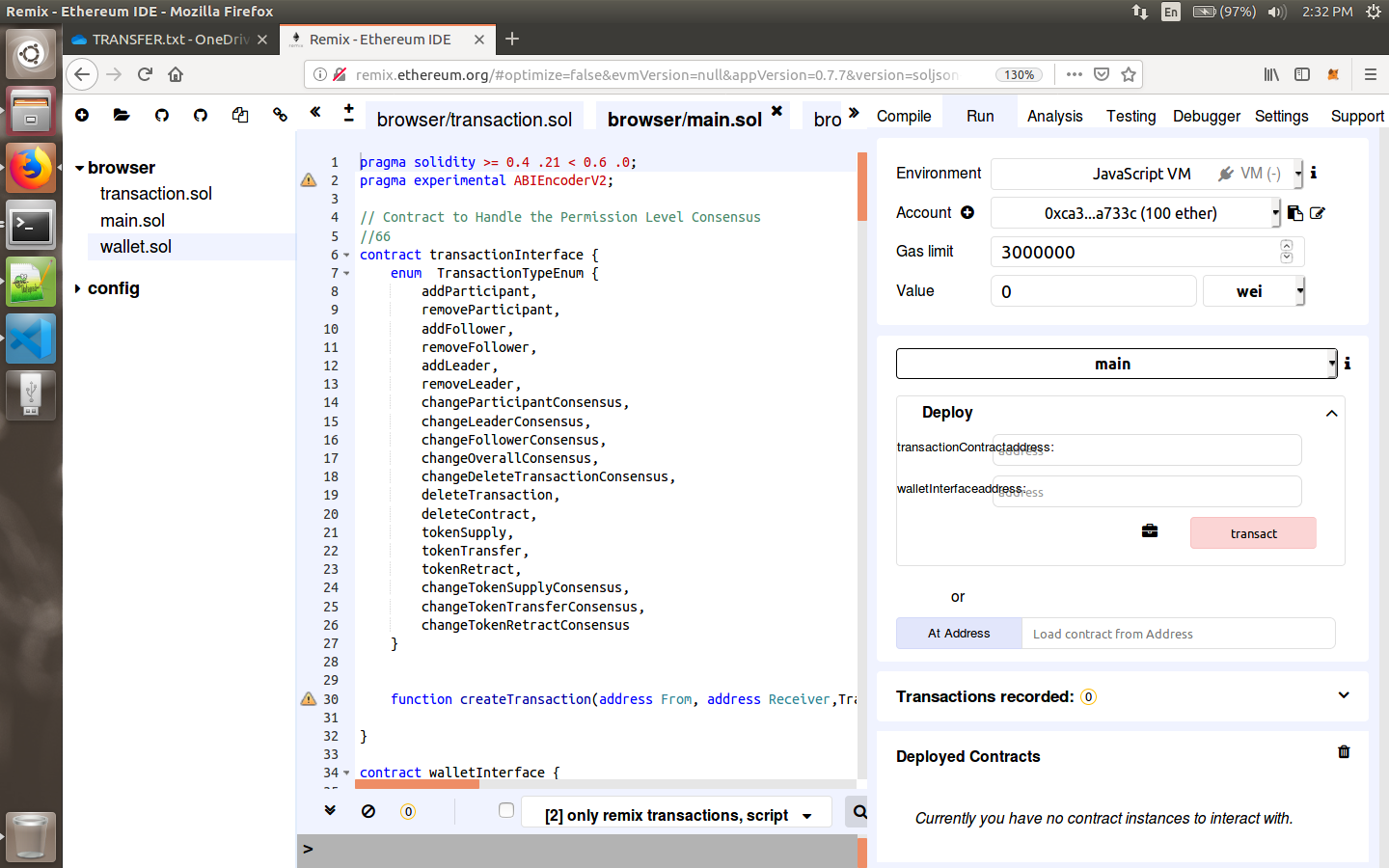
**Wallet Contract:**

****

**Transaction Contract:**

****

**Main Contract:**

****

**Finally After Deploying the 3 Smart contract we have noted the address of each of them, because they will be the ones we will be interacting with. Although I have showed how to get the ABI we don’t have to save it anywhere since it will be part of the code commit of the Middleware Repo which contains the ABI of this version of SMART CONTRACT under:**

<https://gitlabee.dt.renault.com/swlabs/blockchain/ethereum_poo/tree/master/api_admin1/odometerapi/blockchainapi/ethereum/abi>

**This will be used by the Node Middleware to communicate to the Smart contract through its corresponding Node.**

**We should get three address of smart contracts similar to the one listed below:**

ethereum\_maincontract='0xf9eae0f545c0d5b1f1538df4746e46bf2c90d381'

ethereum\_transactioncontract='0x81c3b8de05e26519970a3a150db46fad9014d4e2'

ethereum\_walletcontract='0x92e9180e02fd62c91f509686c6457066b59266ca'

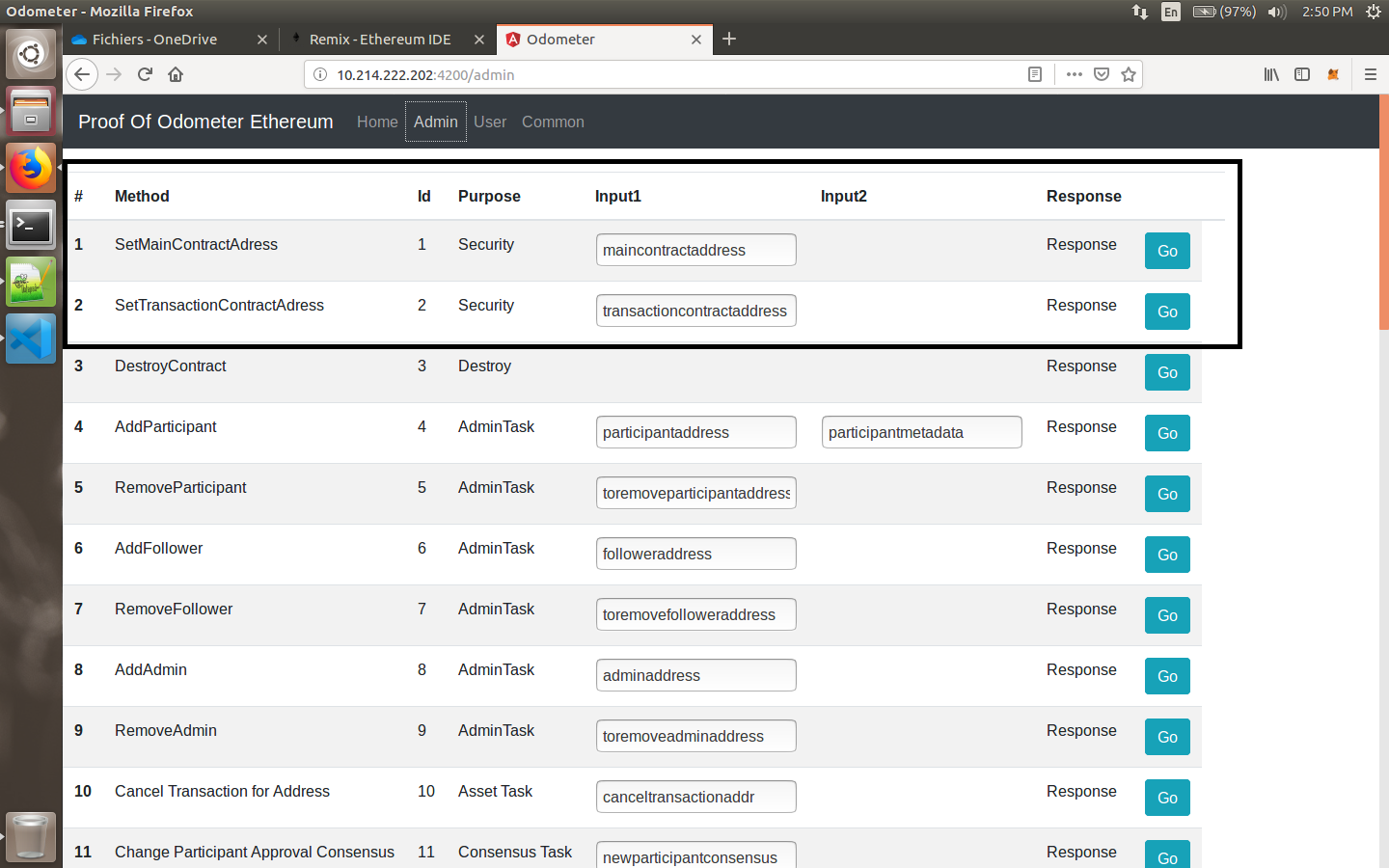
**This will be specified in the .env file of the Middleware project.**

**Step 3: Security Check and Address Distribution:**

**After the deployment of the smart contract we have to complete the address distribution among each of the smart so that it knows each of them and expects the call only from these addresses.**

**This step can be either perform through the REMIX IDE where we can interact with the methods directly or else when deploying the Middleware and corresponding Front End, we have the option of distributing the address.**

**\*\* Remember this distribution of verified address is only one-time function and cannot be invoked again, If there is a mistake of the address provision, then smart contracts have to be deployed again.**

****

**This interface will be available when we deploy the middleware as well as the angular front end corresponding to any Admin Node.**

**As highlighted in this screenshot we set the Main Contract Address and the Transaction Contract Address.**

**This completes the procedure of the SMART Contract deployment with security.**