Workbook: Ethereum Basics

Document Purpose

This document is a ***workbook*** to help accomplish the Smart-Contract demo activity showcased during the Women Who Code Blockchain - Ethereum Basics session on @October 20, 2021

The audience of this event are expected to be participants who are fairly new to the Blockchain development world, hence the workbook covers the smart-contract implementation using an online IDE.

Github

The source code of this demo smart-contract along with the web UI can be found at - https://github.com/crpdev/wwcb-ethereum-basics

Presentation Deck

The presentation deck used during the event can be found at wwc-ethereum-basics-deck

Pre-requisites

The pre-requisites to proceed further with this workbook are

- Desktop or Laptop with a stable Internet connection
- Access to the online IDE - Remix

Writing a Smart-Contract

To get started with smart-contract development in Solidity, let's assume a hypothetical use-case scenario and implement it.

Requirements

The Admin or Moderator of Women Who Code Blockchain community approaches you to develop a DApp (Decentralised Application) to manage Events organised in the community. They approach you so you could help them in writing a smart-contract and a front-end that wires to the smart-contract.

Actors

The actors who interact with the smart-contract are the Admin or Moderator of the community and the community members who vote for events proposed.

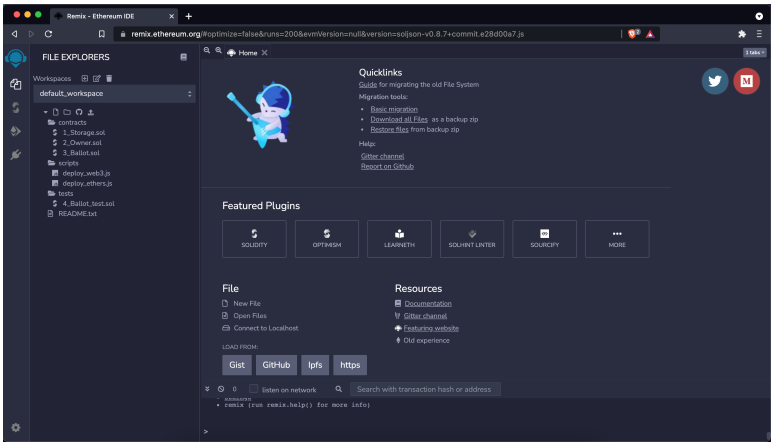
Scenarios

The smart-contract should cover the following scenarios. This is however the initial version and could be extended as needed.

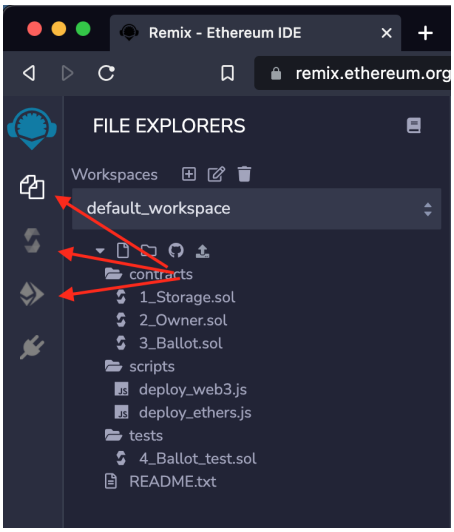
- Only the Admin or Moderator should be able to add event proposals
- The Admin or Moderator should not have the permission to vote for a proposed event
- Participants should be able to vote for an event proposal
- Participant can vote for an event only once
- Anyone should be able to view the total votes
- The smart-contract address and owner details must be available to the public

Let's start coding our smart-contract

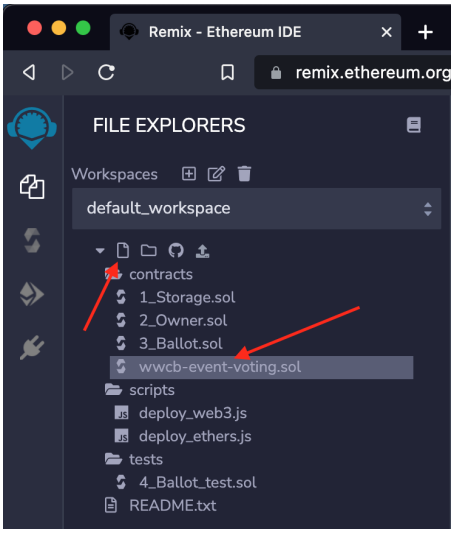
1. Head over to https://remix.ethereum.org/ - an online IDE that helps you write smart contracts, compile to check for errors and test



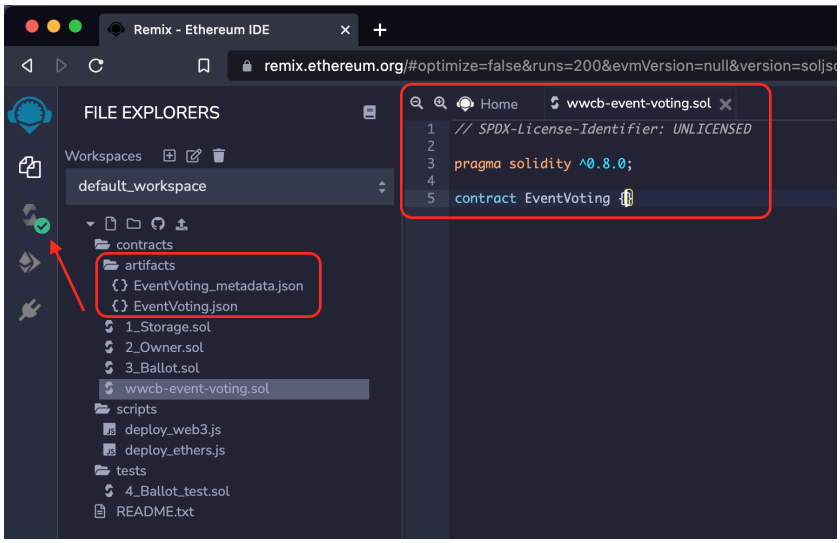
2. On the left panel, you'll find the explorer, solidity compiler and deploy & run windows



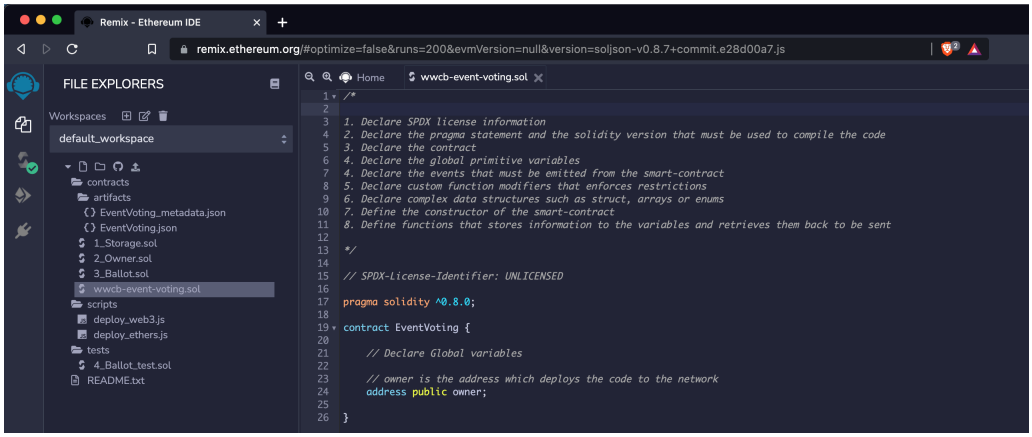
3. In the File Explorer panel, click the new file icon or right click on "contracts" folder create a new file "wwcb-event-voting.sol"



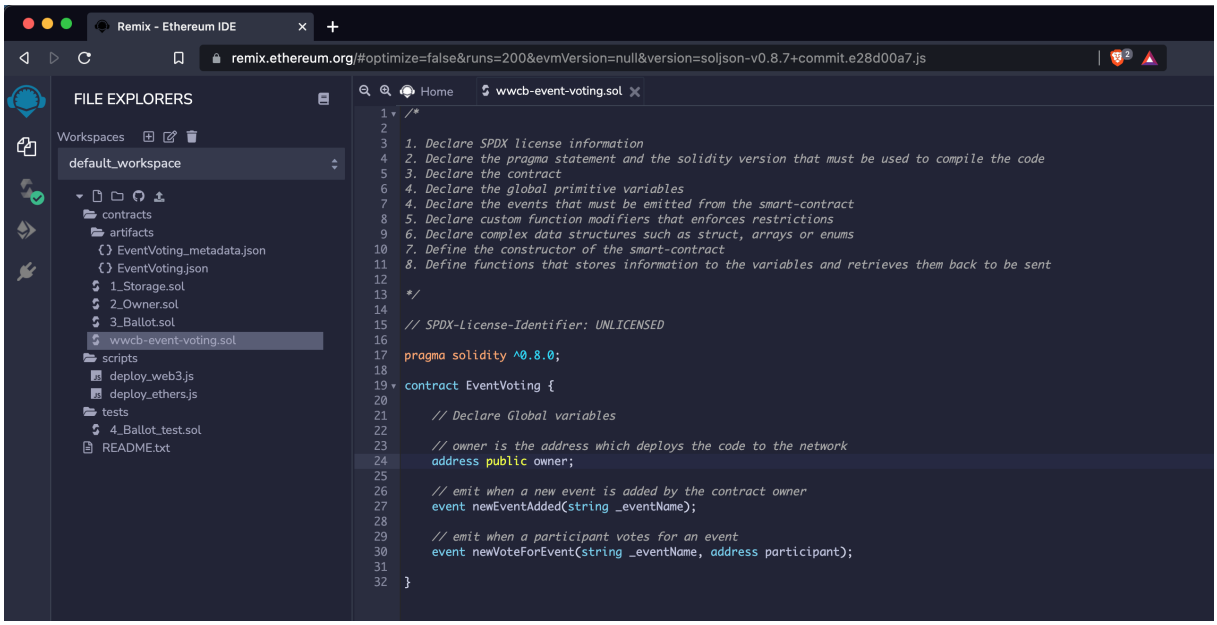
4. We start by declaring the SPDX and the pragma declarations, followed by the contract name. For the purposes of this workbook, we'll use the contract name as "EventVoting", however you're free to choose any. As you write the declarations and save the file, you'll notice some changes in the IDE:
    - An artifacts folder pops up with json files. These are the compiled contract ABIs
    - The compiler checks the contract and displays the result



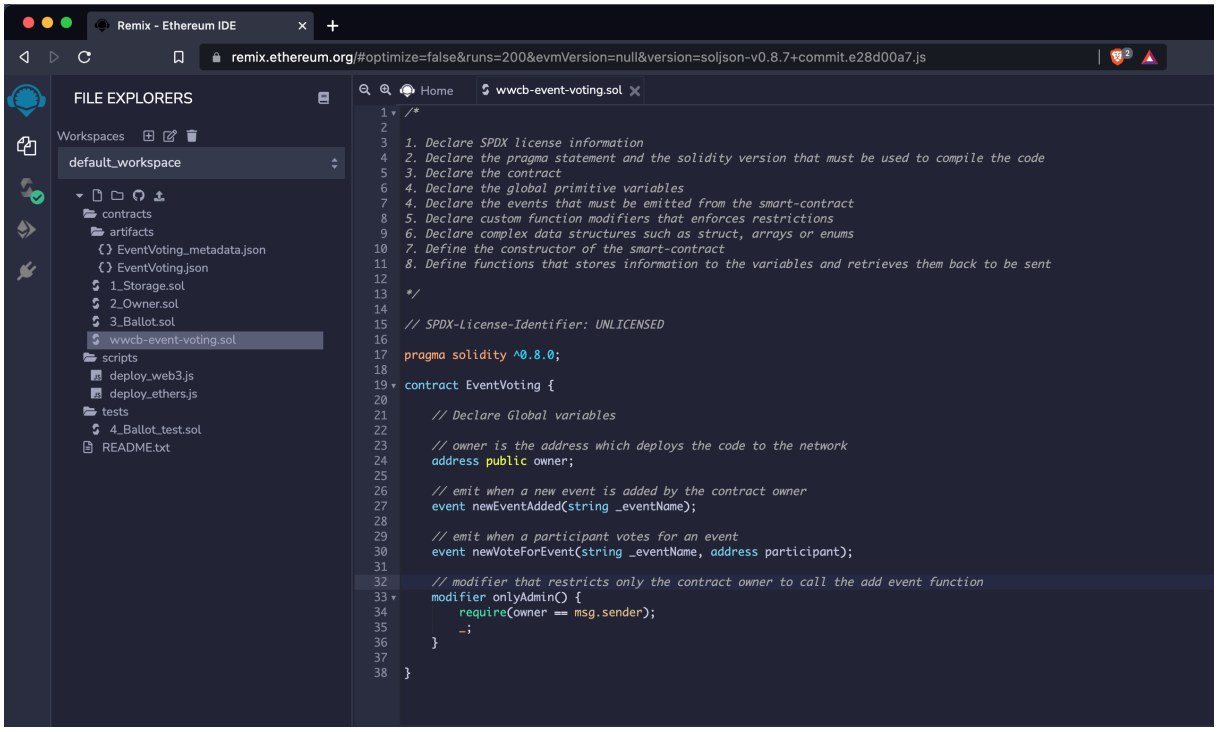
5. We now start writing the global primitive variables that will be used within the smart-contract
    a. owner - Address of the contract owner who deploys the smart-contract



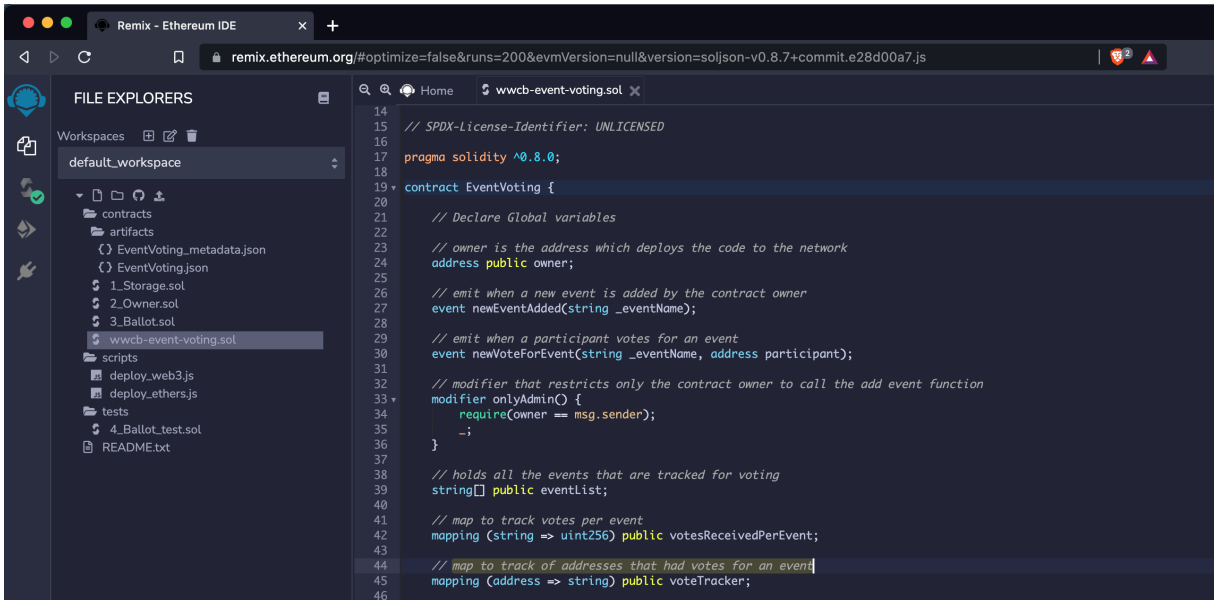
6. Declare the events that will be emitted from the smart-contract
    a. newEventAdded - emit when a new event is added by the Admin
    b. newVoteForEvent - emit when a participant votes for an event



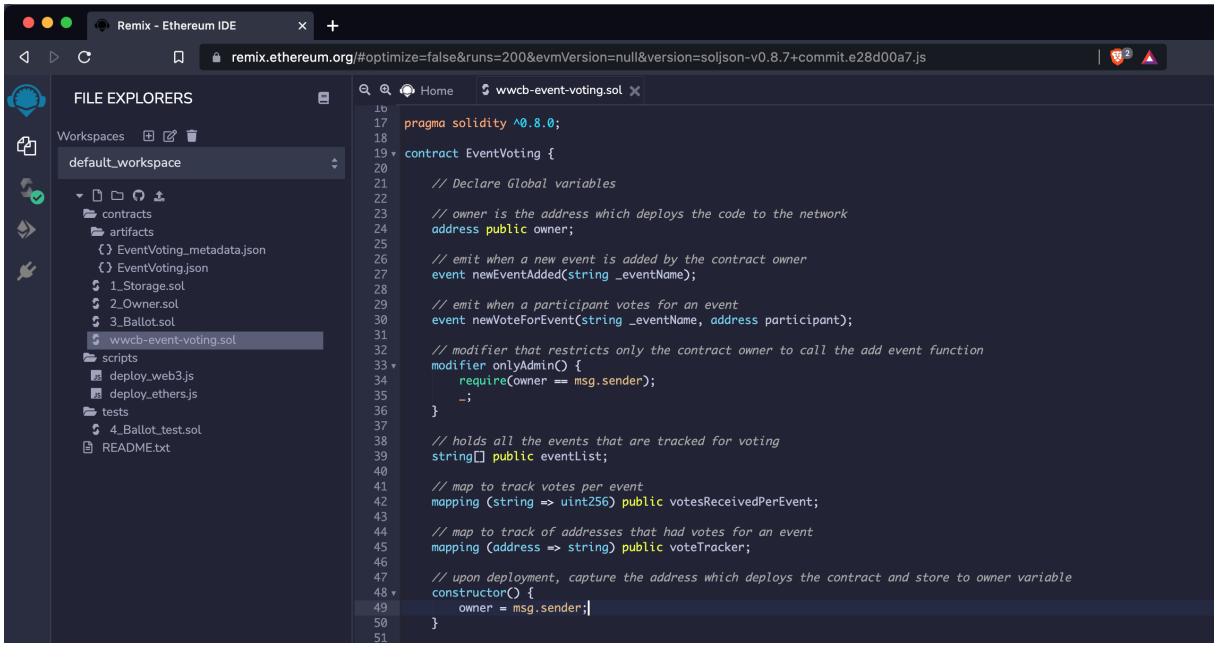
7. Declare modifiers that enforces restrictions on function calls
    a. onlyAdmin - modifier that restricts only the contract owner to call the add event function



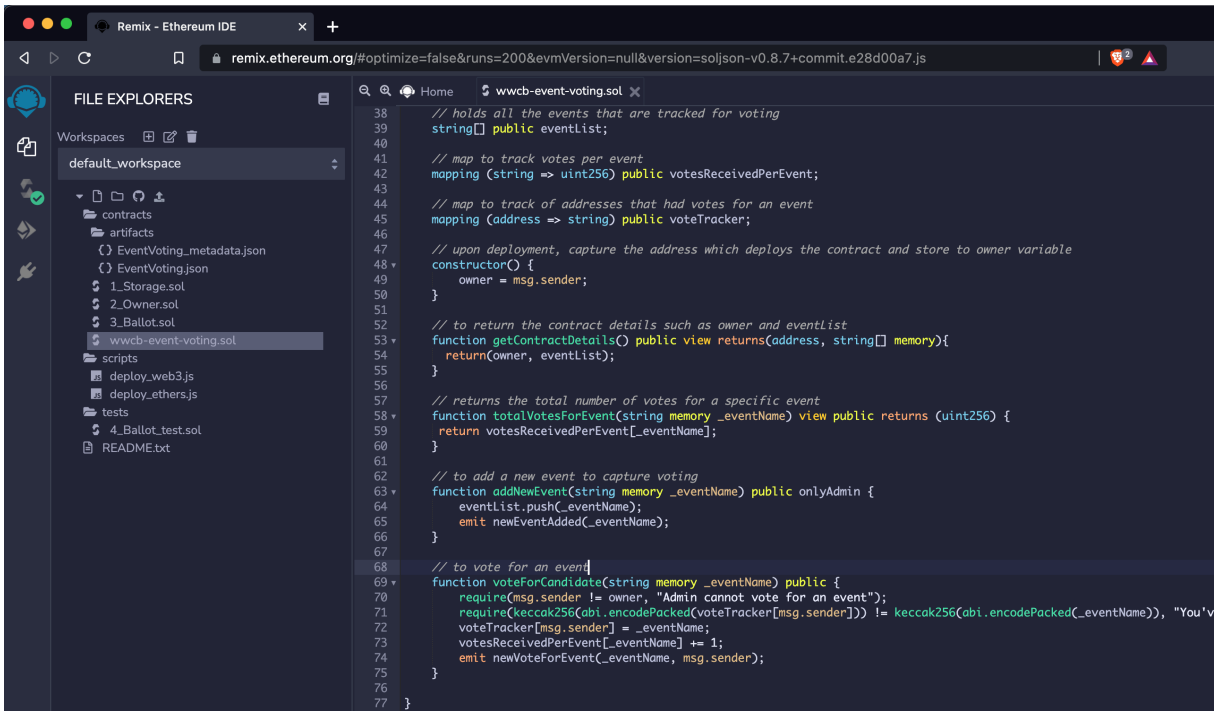
8. Declare complex data structures such as structs, arrays or enums to be used
    a. eventList - holds all the events that are tracked for voting
    b. votesReceivedPerEvent - map to track votes per event
    c. voteTracker - map to track of addresses that had votes for an event



9. Define the constructor that will be called during contract creation. Constructor will be called only once in the contract life-cycle. In our smart-contract, we store the owner variable with the address that deploys the smart-contract



10. Define the functions
    a. getContractDetails - to return the contract details such as owner and eventList
    b. totalVotesForEvent - returns the total number of votes for a specific event
    c. addNewEvent - called only by admin to add a new event to capture voting
    d. voteForCandidate - called only by the participants to vote for an event



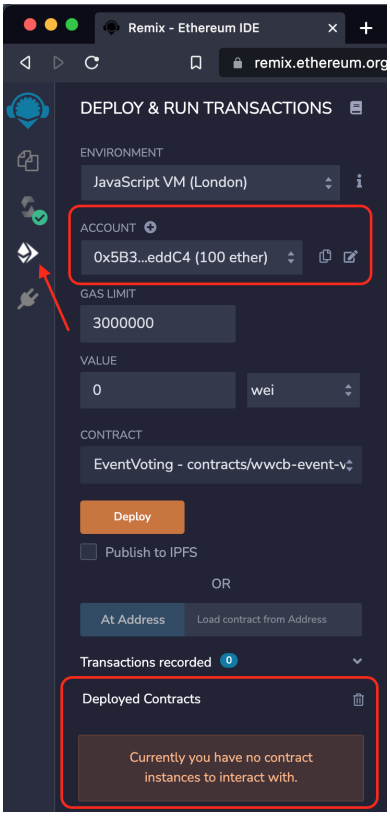
11. As you save, the smart-contract code is compiled and the status is display on the compiler menu.

    Code

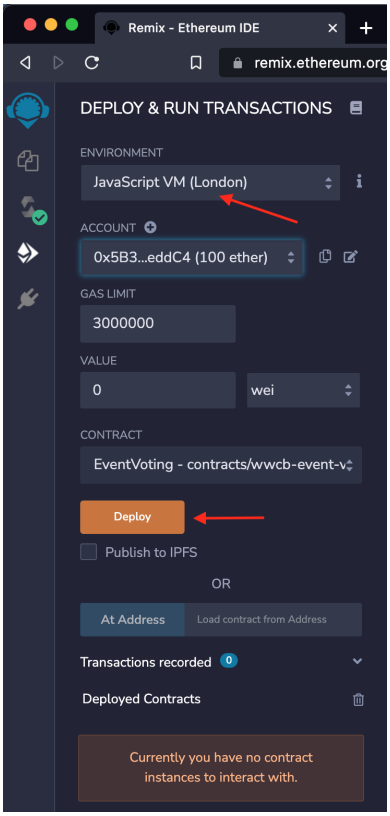
```
/*

1. Declare SPDX license information
2. Declare the pragma statement and the solidity version that must be used to compile the code
3. Declare the contract
4. Declare the global primitive variables
4. Declare the events that must be emitted from the smart-contract
5. Declare custom function modifiers that enforces restrictions
6. Declare complex data structures such as struct, arrays or enums
7. Define the constructor of the smart-contract
8. Define functions that stores information to the variables and retrieves them back to be sent

*/

// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.0;

contract EventVoting {

    // Declare Global variables

    // owner is the address which deploys the code to the network
    address public owner;

    // emit when a new event is added by the contract owner
    event newEventAdded(string _eventName);

    // emit when a participant votes for an event
    event newVoteForEvent(string _eventName, address particpant);

    // modifier that restricts only the contract owner to call the add event function
    modifier onlyAdmin() {
        require(owner == msg.sender);
        _;
    }

    // holds all the events that are tracked for voting
    string[] public eventList;
```

```
    // map to track votes per event
    mapping (string => uint256) private votesReceivedPerEvent;

    // map to track of addresses that had votes for an event
    mapping (address => string) private voteTracker;

    // upon deployment, capture the address which deploys the contract and store to owner variable
    constructor() {
        owner = msg.sender;
    }

    // to return the contract details such as owner and eventList
    function getContractDetails() public view returns(address, string[] memory){
      return(owner, eventList);
    }

    // returns the total number of votes for a specific event
    function totalVotesForEvent(string memory _eventName) view public returns (uint256) {
     return votesReceivedPerEvent[_eventName];
    }

    // to add a new event to capture voting
    function addNewEvent(string memory _eventName) public onlyAdmin {
        eventList.push(_eventName);
        emit newEventAdded(_eventName);
    }

    // to vote for an event
    function voteForCandidate(string memory _eventName) public {
        require(msg.sender != owner, "Admin cannot vote for an event");
        require(keccak256(abi.encodePacked(voteTracker[msg.sender])) != keccak256(abi.encodePacked(_eventName)), "You've already vo
        voteTracker[msg.sender] = _eventName;
        votesReceivedPerEvent[_eventName] += 1;
        emit newVoteForEvent(_eventName, msg.sender);
    }

}
```

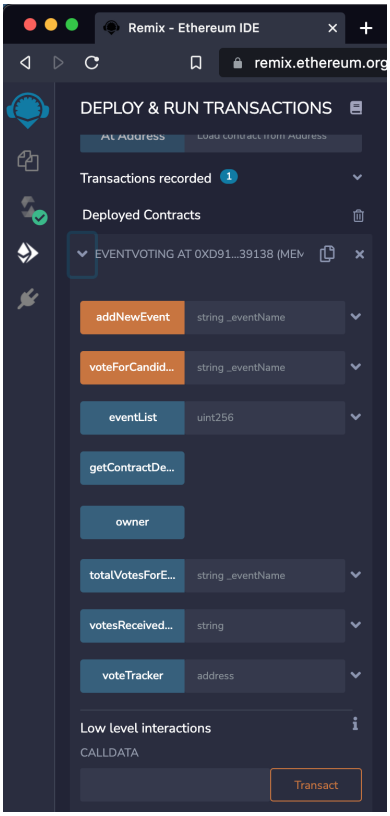
12. It's now time to deploy and test the smart-contract
13. Remix IDE comes with features that provides 10 test accounts that can be used to verify the smart-contract. Click on the "Deploy and run transactions" menu. Right now there are no contracts deployed



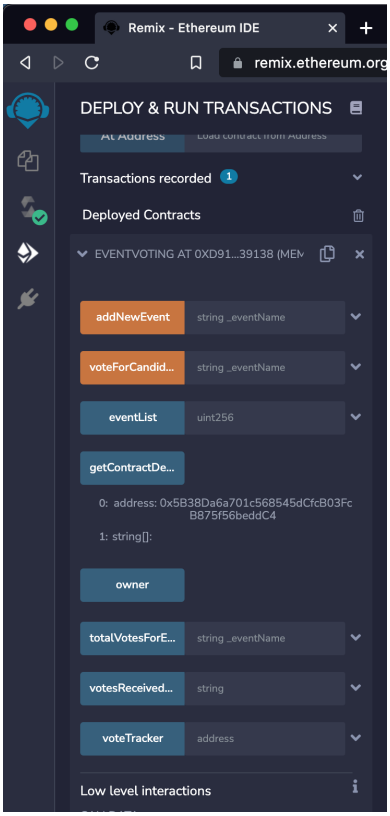
14. Make sure the JavaScript VM environment is selected. Select the 1st account - which will be the owner of the contract and click "Deploy"



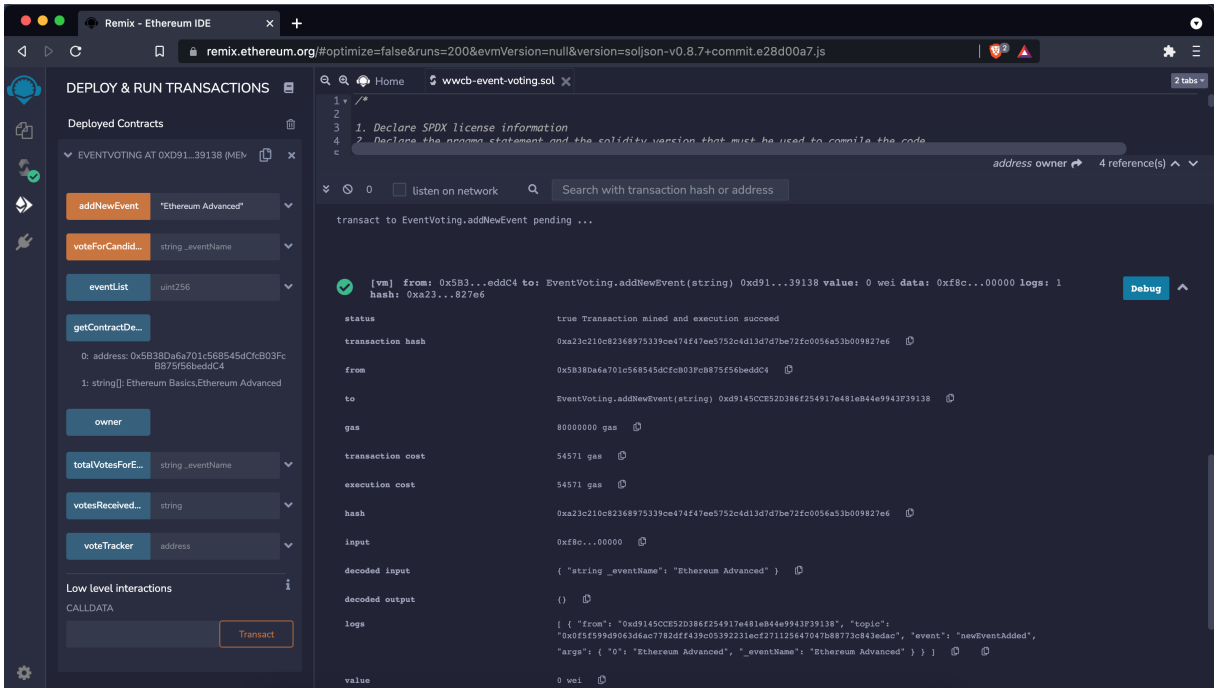
15. You'll notice that the Deployed Contracts tab now has the details of the newly deployed contract. You'll also see the functions available to perform testing



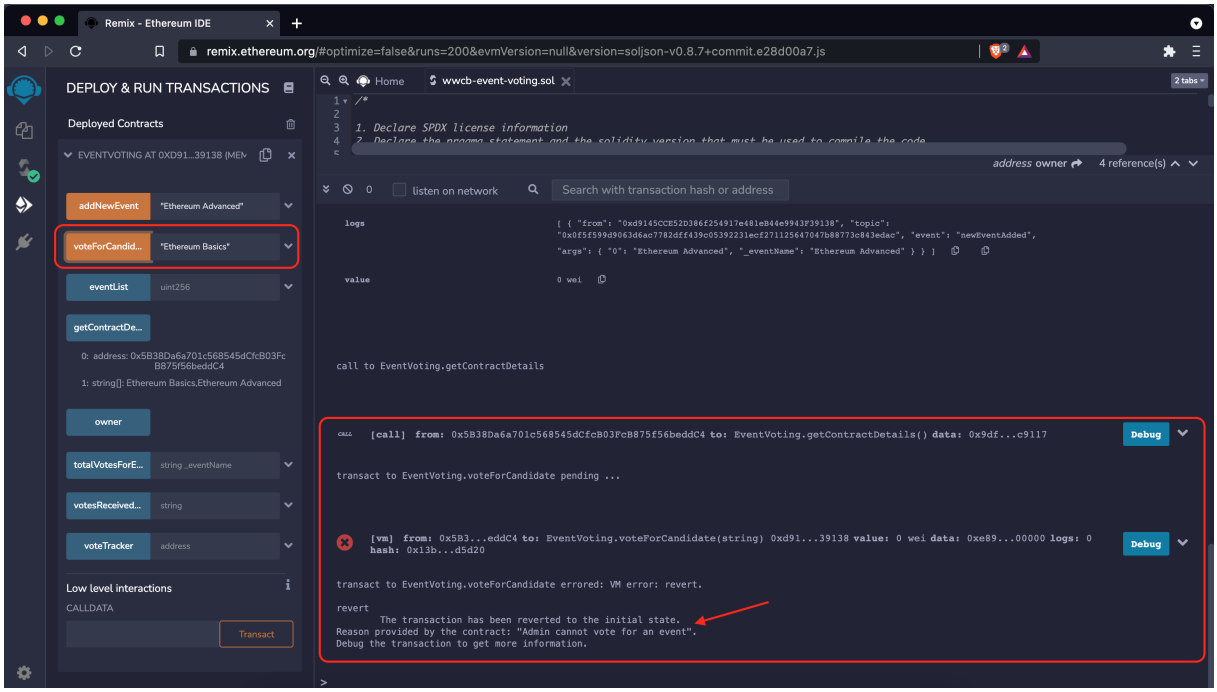
16. Click on getContractDetails method
    a. getContractDetails returns only the owner as there are no events yet



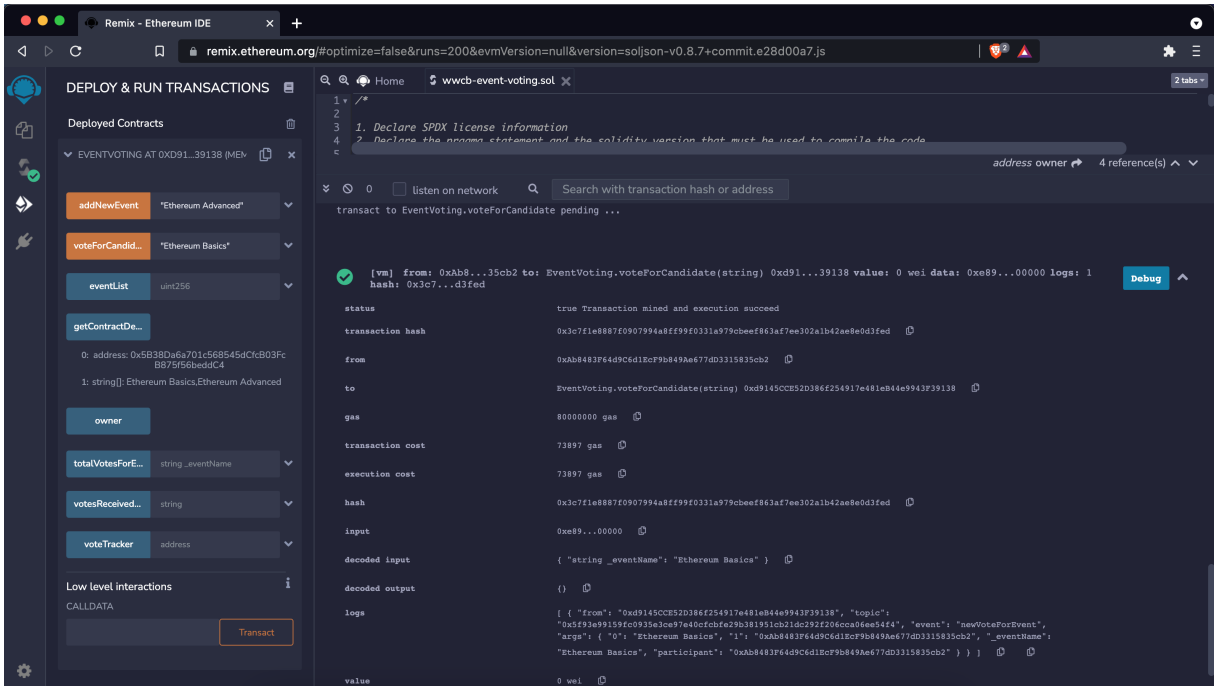
17. With the 1st account selected (owner), you can add new events. As new events are added, the details of the transaction can be viewed in the console window. As part of the addNewEvent function, we also emit an event and the same is visible in the console log



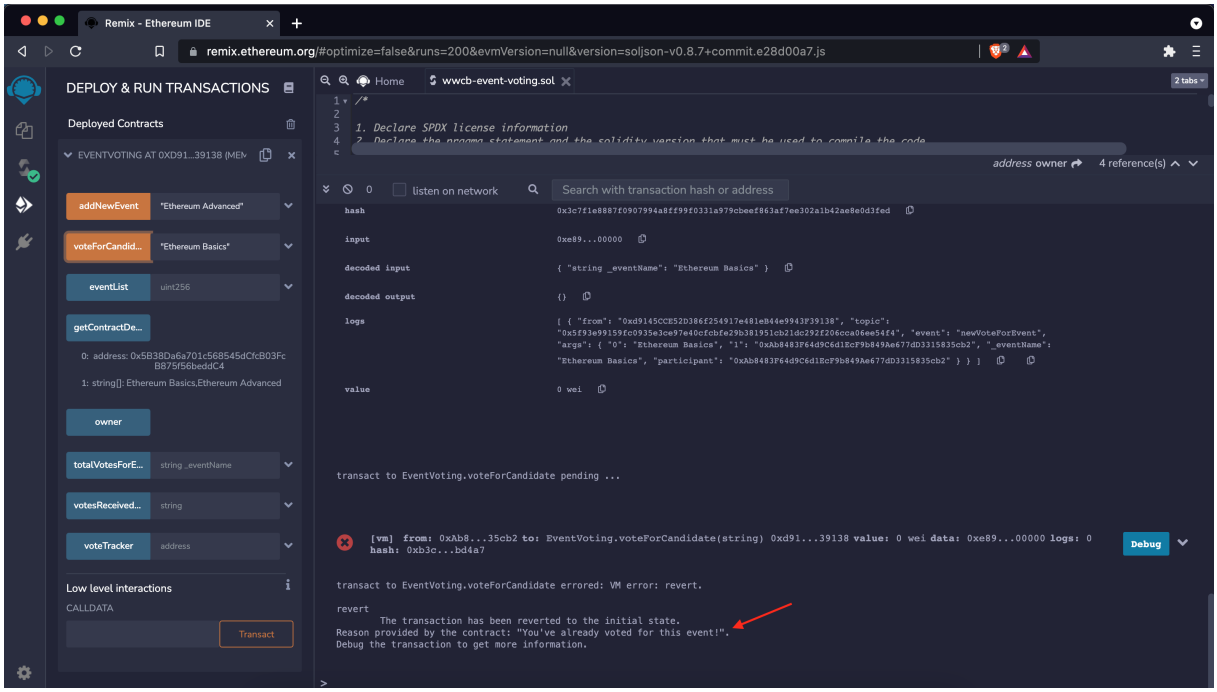
18. With the 1st account selected, if you try to vote for an event, you must get an error



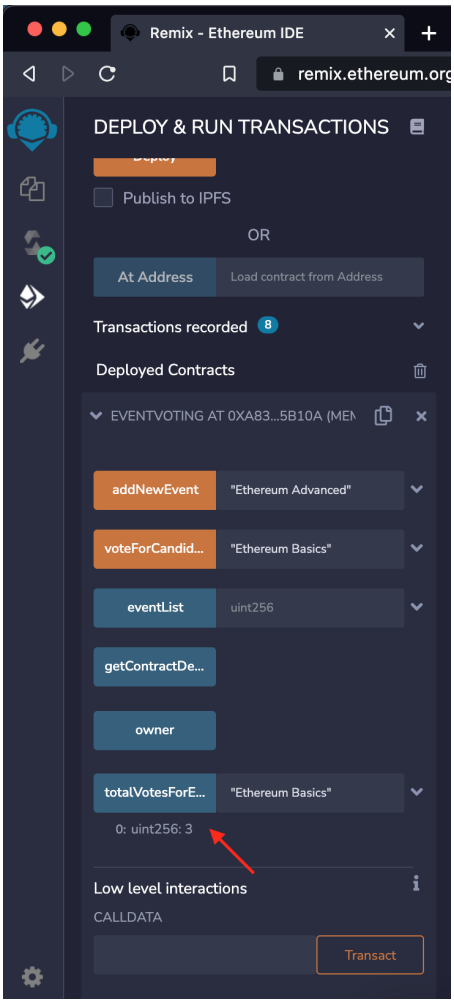
19. Select any account from the other 9 accounts in the Account drop-down list and vote for an event and now the transaction must pass



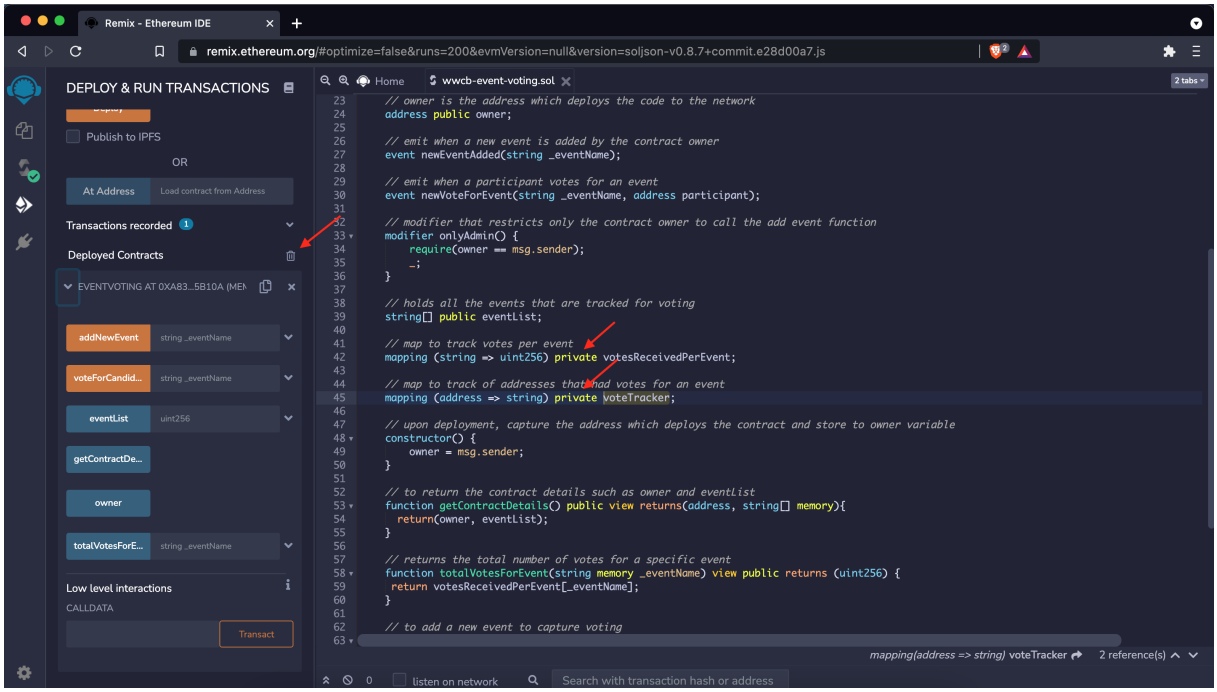
20. When you try to vote for the same event again, it should fail with an error



21. Add new events and cast votes using multiple accounts. Call totalVotesForEvent method to check the votes received for an event



22. Notice that the mappings votesReceivedPerEvent and voteTracker which has the visibility "public" is open for everyone, delete the contract from the Deployed Contracts menu, change the visibility to private and redeploy the contract. Be aware that, when you re-deploy the contract, you'll lose all previous data



23. The tests conclude that the smart-contract is implemented with all the requirements