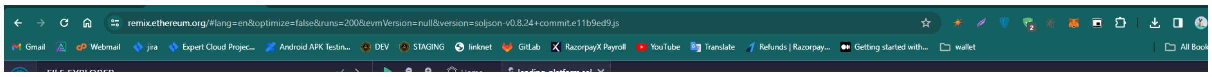
Deploying a smart contract using Remix is a straightforward process. Remix is an online Solidity IDE that allows you to write, compile, and deploy smart contracts directly from your web browser. Below is a step-by-step process to deploy a smart contract using Remix:

Step 1: Open Remix

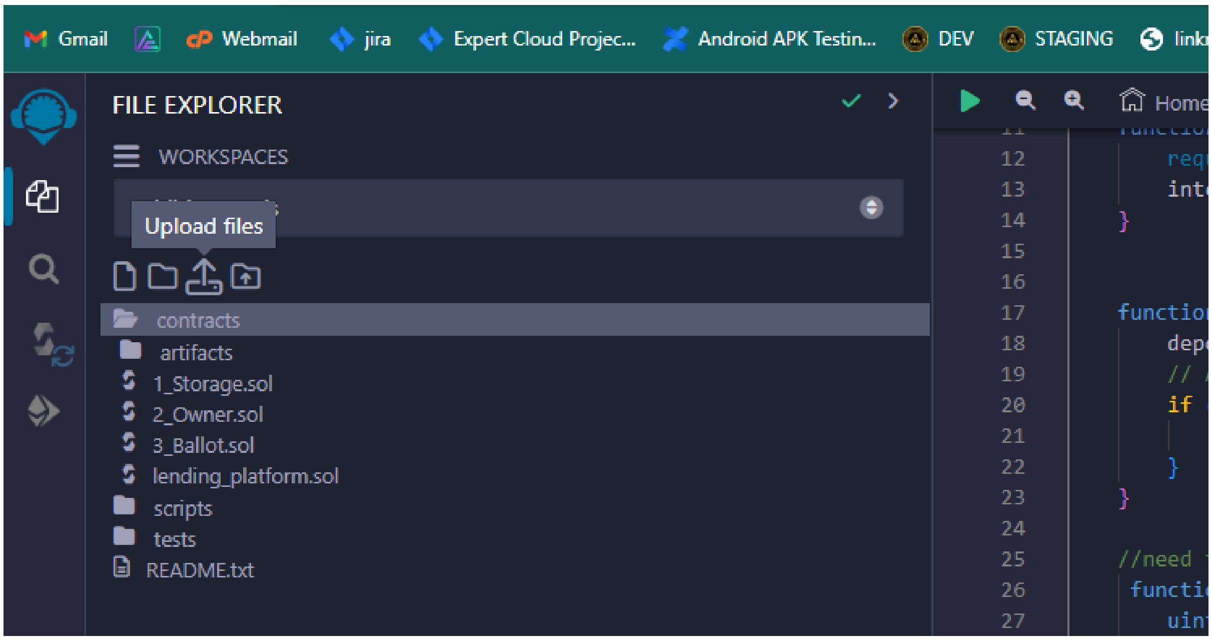
1. Open your web browser and navigate to [Remix IDE](https://remix.ethereum.org/).



remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.24+commit.e11b9ed9.js

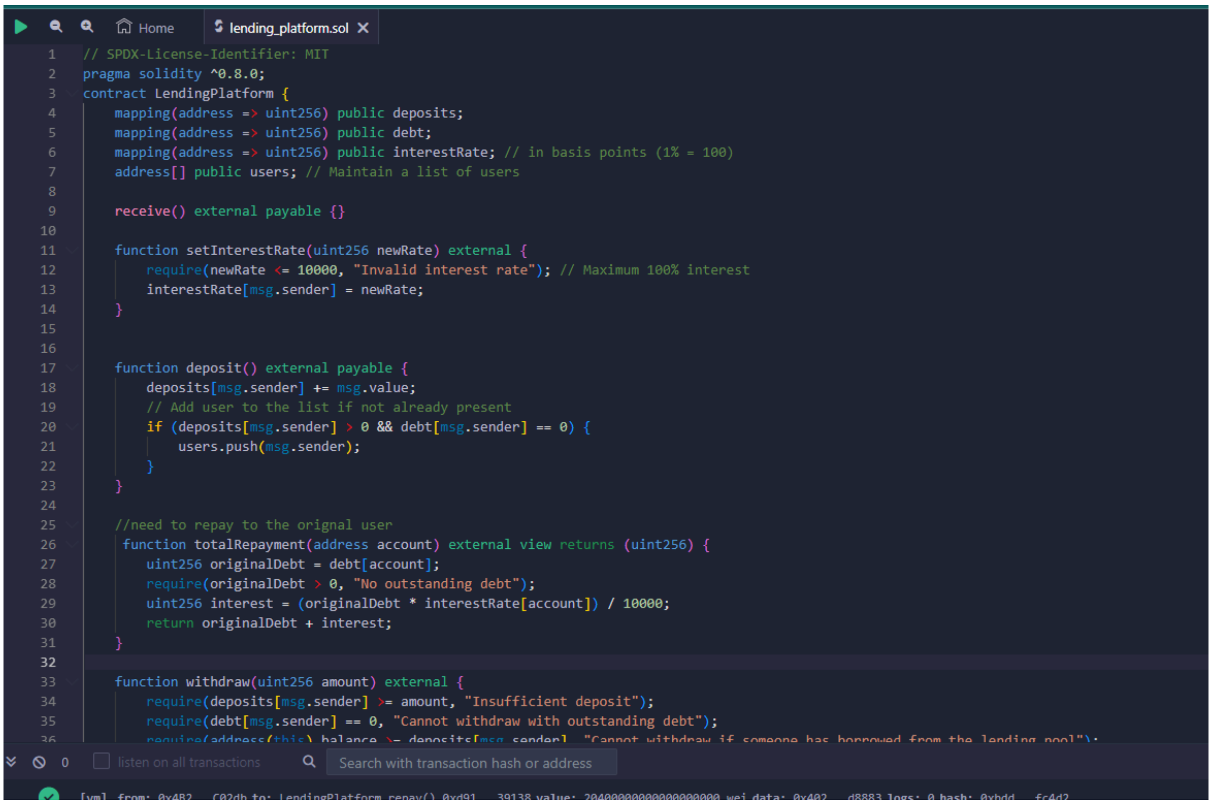
Step 2: Create or Import Smart Contract

1. Create a new file or import an existing smart contract file into Remix. You can use the "+" button on the left sidebar to add a new file or import from GitHub, Gist, or your local machine.



Step 3: Write or Paste Smart Contract Code

1. Write your smart contract code in the editor or paste the code if you're importing from an existing source.


lending_platform.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract LendingPlatform {
    mapping(address => uint256) public deposits;
    mapping(address => uint256) public debt;
    mapping(address => uint256) public interestRate; // in basis points (1% = 100)
    address[] public users; // Maintain a list of users

    receive() external payable {}

    function setInterestRate(uint256 newRate) external {
        require(newRate <= 10000, "Invalid interest rate"); // Maximum 100% interest
        interestRate[msg.sender] = newRate;
    }


    function deposit() external payable {
        deposits[msg.sender] += msg.value;
        // Add user to the list if not already present
        if (deposits[msg.sender] > 0 && debt[msg.sender] == 0) {
            users.push(msg.sender);
        }
    }

    //need to repay to the orignal user
     function totalRepayment(address account) external view returns (uint256) {
        uint256 originalDebt = debt[account];
        require(originalDebt > 0, "No outstanding debt");
        uint256 interest = (originalDebt * interestRate[account]) / 10000;
        return originalDebt + interest;
    }

    function withdraw(uint256 amount) external {
        require(deposits[msg.sender] >= amount, "Insufficient deposit");
        require(debt[msg.sender] == 0, "Cannot withdraw with outstanding debt");
        require(address(this).balance >= deposits[msg.sender], "Cannot withdraw if someone has borrowed from the lending pool");
```

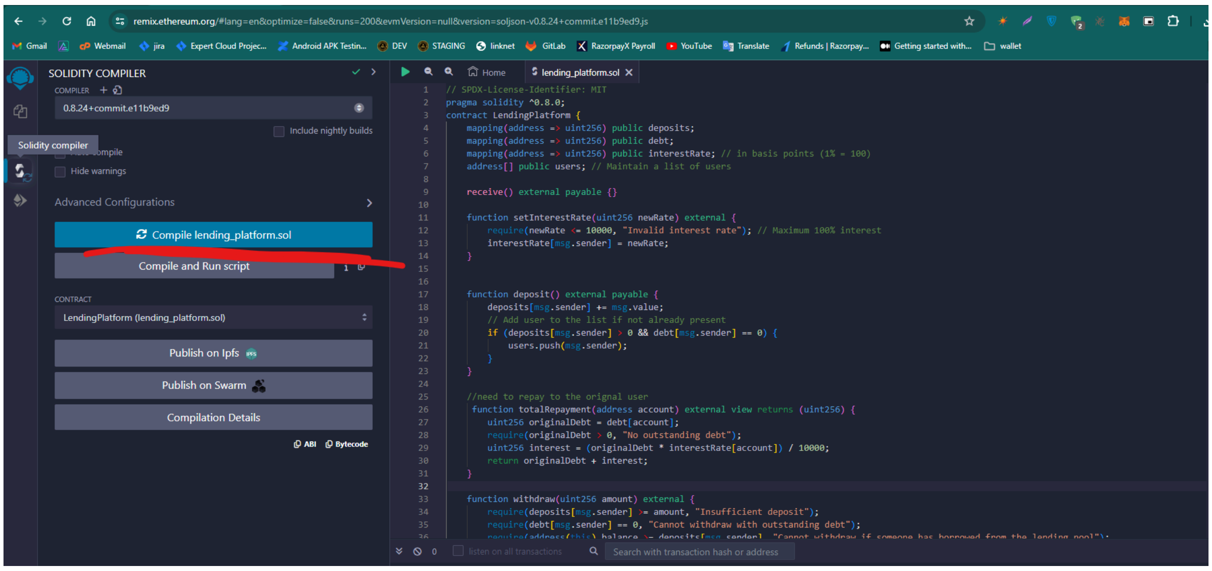
```
0    listen on all transactions    Search with transaction hash or address
[vm] from: 0x4B2...C02db to: LendingPlatform.repay() 0xd91...39138 value: 20400000000000000000 wei data: 0x402...d8883 logs: 0 hash: 0xbdd...fc4d2
```

Step 4: Compile Smart Contract

1. In the left sidebar, go to the "Solidity Compiler" tab.

2. Choose the version of Solidity you want to use from the dropdown menu.

3. Click the "Compile" button to compile your smart contract. Verify that there are no compilation errors in the "Compile" tab.


remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.24+commit.e11b9ed9.js

SOLIDITY COMPILER

COMPILER: 0.8.24+commit.e11b9ed9

Include nightly builds

Auto compile

Hide warnings

Advanced Configurations

Compile lending_platform.sol

Compile and Run script

CONTRACT: LendingPlatform (lending_platform.sol)

Publish on Ipfs

Publish on Swarm

Compilation Details

ABI Bytecode

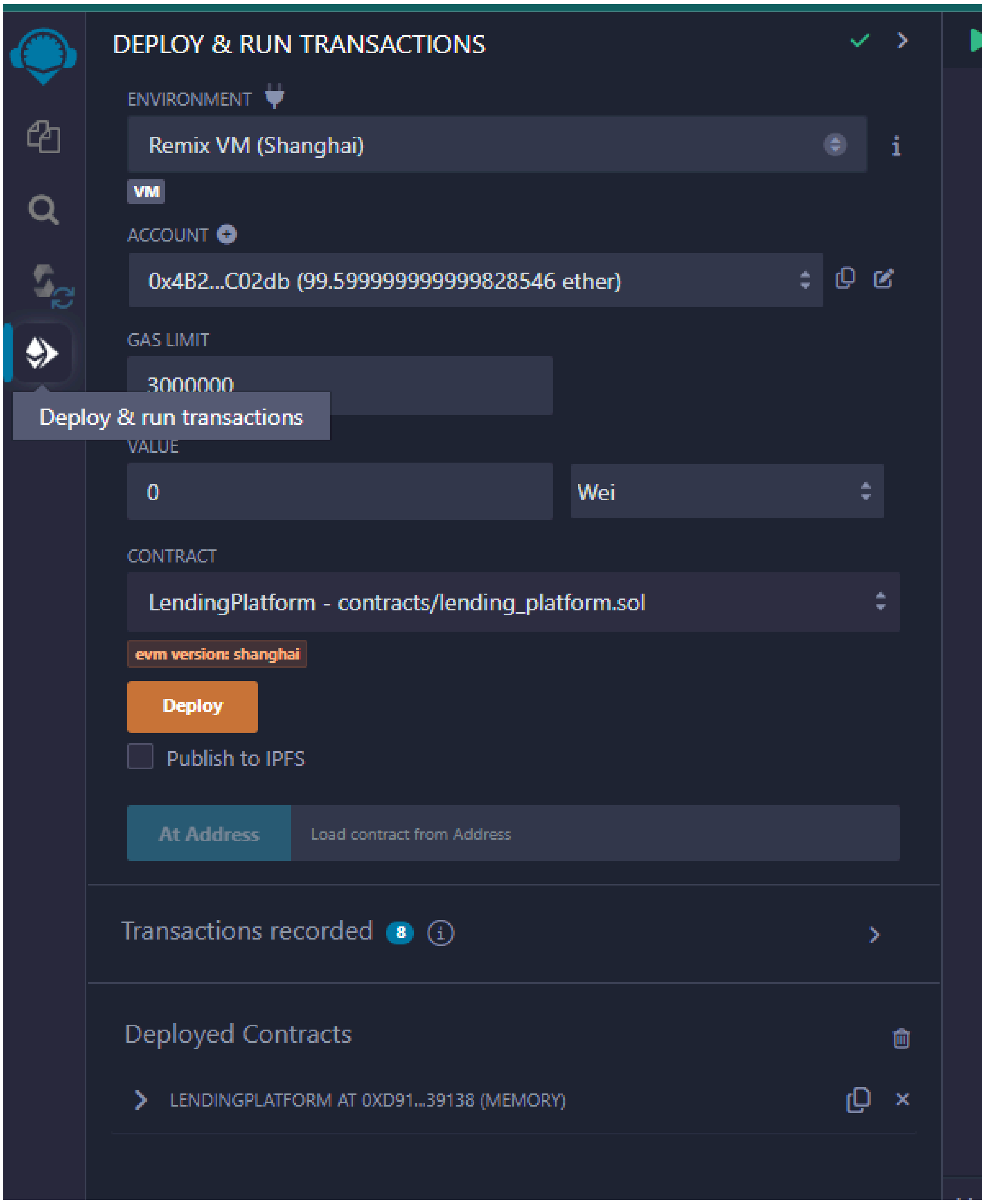
```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract LendingPlatform {
    mapping(address => uint256) public deposits;
    mapping(address => uint256) public debt;
    mapping(address => uint256) public interestRate; // in basis points (1% = 100)
    address[] public users; // Maintain a list of users

    receive() external payable {}

    function setInterestRate(uint256 newRate) external {
        require(newRate <= 10000, "Invalid interest rate"); // Maximum 100% interest
        interestRate[msg.sender] = newRate;
    }


    function deposit() external payable {
        deposits[msg.sender] += msg.value;
        // Add user to the list if not already present
        if (deposits[msg.sender] > 0 && debt[msg.sender] == 0) {
            users.push(msg.sender);
        }
    }

    //need to repay to the orignal user
     function totalRepayment(address account) external view returns (uint256) {
        uint256 originalDebt = debt[account];
        require(originalDebt > 0, "No outstanding debt");
        uint256 interest = (originalDebt * interestRate[account]) / 10000;
        return originalDebt + interest;
    }

    function withdraw(uint256 amount) external {
        require(deposits[msg.sender] >= amount, "Insufficient deposit");
        require(debt[msg.sender] == 0, "Cannot withdraw with outstanding debt");
        require(address(this).balance >= deposits[msg.sender], "Cannot withdraw if someone has borrowed from the lending pool");
```

Step 5: Deploy Smart Contract

1. Switch to the "Deploy & Run Transactions" tab in the left sidebar.

2. Select the environment where you want to deploy your smart contract (JavaScript VM, Injected Web3, etc.). For beginners, using the "JavaScript VM" is recommended for testing.

3. Click the "Deploy" button.


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

Remix VM (Shanghai)

VM

ACCOUNT

0x4B2...C02db (99.5999999999999828546 ether)

GAS LIMIT

3000000

VALUE

0 Wei

CONTRACT

LendingPlatform - contracts/lending_platform.sol

evm version: shanghai

Transactions recorded 8

Deployed Contracts

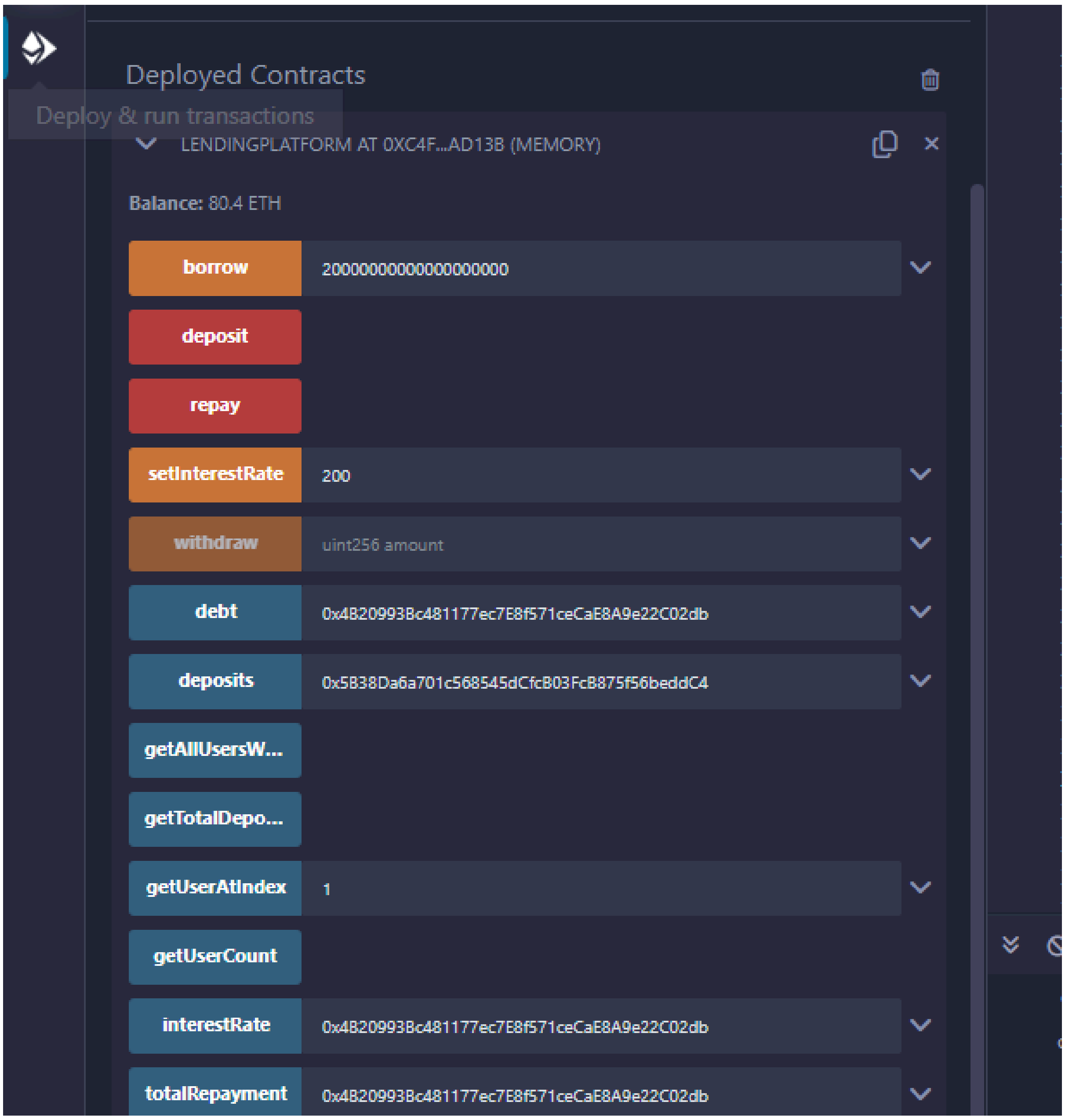
LENDINGPLATFORM AT 0XD91...39138 (MEMORY)

4. Remix will provide you with the deployed contract address, and you'll see your contract in the "Deployed Contracts" section.

Step 6: Interact with Smart Contract

1. Once the contract is deployed, you can interact with it using the provided user interface in the "Deployed Contracts" section.

2. Click on the deployed contract under "Deployed Contracts" to reveal its functions and interact with them.


Deployed Contracts

Deploy & run transactions

LENDINGPLATFORM AT 0XC4F...AD13B (MEMORY)

Balance: 80.4 ETH

borrow	20000000000000000000
deposit	
repay	
setInterestRate	200
withdraw	uint256 amount
debt	0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
deposits	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
getAllUsersW...	
getTotalDepo...	
getUserAtIndex	1
getUserCount	
interestRate	0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
totalRepayment	0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db