## **TASK**

1. Create a smart contract with the Sui package to establish an on-chain IDO. The smart contract should include three functions:

* Create IDO: With fields such as StartDateTime, EndDateTime, cap (limit of amount), whitelisting address (wallet address which can participate in the IDO), and Owner.
* Fund IDO: Only whitelisted addresses are permitted to fund the IDO until it reaches the cap amount.
* Transfer IDO fund: Only the Owner can transfer funds to a specific address.

1. Deploy the smart contract.
2. Simply call the contract from the frontend application using three buttons:

* Create IDO: On clicking this button, the create IDO function should be called with static information and should display the transaction ID.
* Fund IDO: On clicking this button, the Fund IDO function should be called with static information and should display the transaction ID.
* Transfer IDO fund: On clicking this button, the Transfer IDO fund function should be called with static information and should display the transaction ID.

## **IDO.SOL**

* To manage contract ownership, the smart contract incorporates the OpenZeppelin library's Ownable contract.
* The variables cap, whitelist, owner, startDateTime, and endDateTime are all defined as public state variables.
* The UniswapV2Router02 contract from the SushiSwap core library is used to define the router variable.
* The constructor accepts the start and end dates and times, the cap, the whitelist of addresses, the owner's address, and the address of the Uniswap router in order to create an IDO.
* The primary function for constructing the IDO is createIDO. The start and finish dates, the cap, and whether the sender is authorised to participate in the IDO are among the many things that are checked to make sure the IDO is being formed correctly.
* A utility function called isInWhitelist determines whether or not an address is on the whitelist.
* The contract's owner can update the cap and whitelist using the updateCap and updateWhitelist functions, respectively.

## 

## 

## **Fund\_ido.sol**

* The import statements at the top of the code import other contracts from external libraries.
* The contract IDO is declared, which inherits from the Ownable contract from the OpenZeppelin library.
* Various state variables are declared to store information about the IDO, such as the start and end times, the maximum amount that can be raised, the whitelist of addresses allowed to participate, and a mapping of addresses to their contribution amounts.
* The router variable is declared to store an instance of the Uniswap router contract from the SushiSwap core library.
* The constructor function is defined, which takes in various parameters to set the state variables of the contract, including the start and end times of the IDO, the maximum amount that can be raised, the whitelist of addresses allowed to participate, the owner of the contract, and the address of the Uniswap router contract.
* The fundIDO function is defined, which allows users to fund the IDO by sending ETH to the contract. This function first checks that the IDO has started and has not ended, and that the user has not already contributed more than the cap amount. It then checks if the user is whitelisted to participate in the IDO. If they are, it adds their contribution amount to the contributions mapping. If the user's contribution amount exceeds the cap, it removes them from the whitelist. Finally, it transfers the ETH to the owner of the contract.
* The require statements are used to check that certain conditions are met before executing the code. If the conditions are not met, the function will revert and any changes made to the state will be rolled back.

## **transferidofund.sol**

* contract IDO is Ownable { declares the IDO contract and inherits from the Ownable contract, allowing us to use the onlyOwner modifier later on
* function transferIDOFund(address payable \_recipient, uint256 \_amount) external onlyOwner { declares the transferIDOFund function, which takes in two parameters: \_recipient (the address to transfer funds to) and \_amount (the amount to transfer). The external keyword means that the function can be called from outside the contract, and the onlyOwner modifier restricts access to the function to the contract owner.
* require(address(this).balance >= \_amount, "IDO: insufficient funds"); checks if the contract has enough balance to transfer the specified amount of funds.
* require(\_recipient != address(0), "IDO: invalid recipient"); checks if the recipient address is valid.
* \_recipient.transfer(\_amount); transfers the specified amount of funds to the recipient address.

## **React Part:**

1. This is a React functional component that defines a simple UI with 3 buttons and some text.
2. The useState hook is used to define 3 separate states for holding the data returned from 3 different API calls made using the axios library.
3. The 3 button elements have a shared style defined by the buttonStyle object, which is used to give them a consistent appearance.
4. Each button has an onClick event handler that makes a GET request to a specific URL and sets the corresponding state with the response data returned by the API call.
5. The URL used for the API call is a Polygonscan URL for a specific address.
6. The response data is stored in the state using the setIDO1Data, setFundIDOData, and setTransferIDOFundsData functions, respectively, depending on which button was clicked.
7. The UI will display the response data for each button click, if available, using conditional rendering with the && operator to check if the state variable has a value before rendering the corresponding text.

## TestNet Link : <https://mumbai.polygonscan.com/address/0xDC821916B26d32F8fAf77BE7FCe79e88E38bceA2>



