encode CLUB

Solidity Foundations

Lesson 3

Anatomy of a dApp - Part 1
Smart Contracts and Interfaces

Topics

1. Review of HelloWorld.sol
2. Definition of a dApp
3. Interacting with HelloWorld.sol with Interfaces
4. Implementing Access Control with *Ownable*
5. Using an ERC20 with interfaces

HelloWorld.sol

Review

Running HelloWorld.sol on a public testnet

- Connect to Metamask
- Deploy
- Interact

HelloWorld.sol

Who is **hosting** the **code**?

Who is **storing** the **data**?

Who is **managing** the **features**?

dApps

A **Decentralized application** is an application that can operate **autonomously**, typically through the use of **smart contracts**, that run on a **decentralized computing**, **blockchain** or other **distributed ledger** system.

dApps

Like traditional applications, dApps provide some **function** or **utility** to its users.

However, unlike traditional applications, dApps **operate without human intervention** and are not owned nor hosted by any centralized entity.

dApps

Who is **hosting** the **code**?

The smart contract address of the **dApp** points to an account inside the blockchain that has a storage area for **Data** and **Code**.

Whomever sends a **transaction** to that address may trigger a **code execution** that possibly leads to a **state change** in the blockchain.

dAPPs

Who is **storing** the **data**?

All the data used by the **dApp** must be accessible from inside the blockchain.

To access the data outside the blockchain, **dApps** must bring that information inside the blockchain first, usually through an **Oracle** pattern.

dApps

Who is **managing** the **features**?

Anyone can use and even fork all features from a **dApp** at any time.

In most cases, the **dApps** are managed by **Communities** and **Token Holders**.

Some **dApps** are maintained and/or operated by private companies or other centralized organizations. Even in those cases, nothing stops the community from taking over the operation and management of those **dApps**.

Interfaces

Let’s implement an **Interface** for HelloWorld.sol.

Contents:

- *interface* object;
- Abstract contracts;
- Attaching objects to addresses;
- Call data;
- Function selectors.

Practical example

Try it out

Let’s use the features from HelloWorld.sol in the testnet using interfaces.

Practical example

What we’ve learned

- The contract is public and “free” for anyone to use;
- To use the contract, you don’t even need to have access to the code;
- The contract features are the same, regardless of who you are;
- The contract code can’t be changed after deployment.

Access control

Experimentation

New Feature: Only the deployer of the contract should be able to set text.

Contents:

- *owner* state variable;
- *msg.sender* global variable;
- *require* statement;
- Reverted transactions.

Practical example

Try it out

Let’s deploy the HelloWorldOwnable.sol in the testnet.

Practical example

What we’ve learned

- The contract can implement logic to replicate some of the centralized application’s capabilities;
- New implementations of a Smart Contract must be deployed to new addresses [1];
- Everyone can still interact with the previous implementations;
- The same *interface* can be compatible with different contracts.

Footnote 1: Upgradability patterns

Conclusion

Any Smart Contract in the Blockchain can be interacted with, so long we use an **Interface** to do so.

Practical example

Try it out

Let’s pick an interface for ERC20 and try to use it in a deployed address.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol

https://github.com/bokkypoobah/WeenusTokenFaucet

https://goerli.etherscan.io/token/0x022E292b44B5a146F2e8ee36Ff44D3dd863C915c#writeContract

Practical example

Next lesson

Let's use a smart contract by interface, but let's do it using a better UX for non developers