Ira A. Fulton Schools of Engineering
Arizona State University

Course Project1 - Fall B 2022

CSE 598: Engineering Blockchain Applications

Project 1: Creating an ERC-20 Token Standard Smart Contract on Ethereum

Welcome to project 1 of the Engineering Blockchain Applications course. This project will help you familiarize yourself with the Ethereum ecosystem by writing, testing, and deploying an ERC-20 Token Standard smart contract on the Goerli Ethereum Testnet in the Solidity programming language. Through this project, you will learn foundational knowledge to begin your journey creating Ethereum-based Web3 projects that utilize token mechanisms including Decentralized Autonomous Organizations (DAOs) and Non-Fungible Tokens (NFTs).

Background

Tokens are widely used in the Ethereum ecosystem to represent many things including governance power, in-game currencies, reputation points, and much more. The Ethereum ecosystem uses Ethereum Improvement Proposals (EIPs) for the community to propose standards specifying potential new features or processes for the network. EIPs play a central role in how changes happen and are documented on Ethereum. They are the way for people to propose, debate and adopt changes. There are different types of EIPs including core EIPs for low-level protocol changes that affect consensus and require a network upgrade as well as Ethereum Request for Comments (ERCs) for application standards. One such type of ERC standards are token standards, like the ERC-20 Token Standard, which allow applications interacting with the specific ERC standard token to treat all other tokens using the same rules, which makes it easier to create interoperable applications. The ERC standard this project will use is the ERC-20 Token Standard, the most widely used fungible token standard currently on the Ethereum network. Fungibility refers to the ability for each token to be treated exactly the same, in contrast to non-fungible tokens in which the token has properties that make it distinct from other sets of tokens. A smart contract on the Ethereum Network can be considered an ERC-20 Token Contract if it implements the following methods and events. Please read and see the following links for an understanding of what the standard and contract is:

1. https://ethereum.org/en/developers/docs/standards/tokens/erc-20/
2. https://eips.ethereum.org/EIPS/eip-20

Methods

function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool

success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256 remaining)

Events

event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)

ERC-20 Token Smart Contract High Level Logic Description: An ERC-20 Token Smart Contract is a smart contract that creates an ecosystem for a given token you define. At minimum, the token must include the methods and events defined in the eip-20 specification.

The first 3 methods (name(), symbol(), and decimals()) are getter functions that return back basic information that you define for your contract. For your token ecosystem, your contract needs to keep track of the balances of tokens different addresses have (explore the solidity documentation to better understand how balances are represented). The rest of the methods relate to those balances.

The totalSupply() function returns the total token supply that your contract has created (sum of all balances for all addresses that have a balance). The balanceOf() method returns back the token balance of a given address. The transfer() method allows an address to transfer a portion of their token balance to another address. The approve() function allows the caller of the function to approve another address to spend some of the function caller's balance. The allowance() function returns how much an input spender address is still allowed to spend from an input approver address. The transferFrom() function transfers tokens from address1 to address2, given the caller of the function was allowed to transfer address1's tokens and address1 had enough balance left to make the transfer. The two events emit a log to the blockchain whenever a transfer occurs (event Transfer) or whenever an approval occurs (event Approval) (see Solidity documentation to learn about events).

All together, these methods and events create a token ecosystem that is easily interoperable with other smart contracts and can be easily understood and traced by the blockchain. You can create more methods and events on top of the minimum ones specified in the standard. A minting function may be useful, for example.

Project
The rest of this project documentation will walk you through the steps you will need to take to get acquainted with the tools and resources that will allow you to write, test, and deploy your own ERC-20 Token Standard Smart Contract on the Goerli Ethereum Testnet. The tasks below are separated into two categories: Set-Up Tasks and Graded Tasks. Set-Up

Tasks guide you to the resources, tools, and documentation that will help you successfully complete the project. Graded Tasks define how we expect the smart contract you submit to function for us to grade. The tools and resources recommended in the Set-Up tasks will allow you to test that your smart

contract is working properly so that you know what grade to expect upon submission (please ask us questions along the way, we want all smart contracts submitted to be properly functioning ERC-20 Token Standard Smart Contracts that you can be proud of)!

Happy web3 development!

Set-Up Tasks

- *Set-Up Task 1 - MetaMask*

For us to deploy a smart contract on an Ethereum Network, we will first need a way to connect to an Ethereum Network and pay the gas fees associated with deploying your smart contract (it costs the network computational power and storage to execute transitions, those engaging in transactions pay for these costs using "gas").
We recommend using the MetaMask browser extension (supported browsers include Chrome, Firefox, Brave, and Edge), as it gives us an easy-to-use interface into Ethereum Networks that can be used for this project, as well as providing you a wallet and wide-ranging interoperability with other smart contracts and ecosystems you may be interested in exploring in the future.

You can download the browser extension by going to this link: https://metamask.io/

Once you've downloaded the browser extension and gone through the set-up process, you are ready for the next task.

- *Set-Up Task 2 - Goerli Ethereum Testnet*

We will be deploying our smart contracts to the Goerli Ethereum Testnet. The Goerli testnet allows us to deploy our smart contract in a live setting, without the need for real ETH and other mainnet security considerations as we are engaging in development. Testnets are widely used and it is highly recommended to test and deploy smart contracts on a testnet before you deploy on the Ethereum mainnet.

You can learn more about Ethereum Networks by going to this link:https://ethereum.org/en/developers/docs/networks/

To set up the Goerli Testnet on MetaMask, click the browser extension icon for MetaMask (pin the browser extension into your browser bar if you haven't already done so), click the Ethereum Mainnet option at the top of MetaMask pop-out, and click show/hide test networks. This will bring you to the advanced settings. Toggle on the show testnets setting. On the main pop-up again, if you click the Ethereum Mainnet option you can now switch to the Goerli Test Network. Once switched, you have now completed the steps necessary in this task to be able to connect to the Goerli Test Network via your browser.

- *Set-Up Task 3 - Goerli Testnet Faucet*

The on-chain currency for the *Goerli* Testnet is Goerli ETH. You will need Goerli ETH to pay gas fees when deploying your smart contract and transacting on the Goerli network. Similar to many developer

testnets, the Goerli network has faucets that give you Goerli ETH for free (usually up to a daily limit per address). Working faucets as of writing this project documentation can be found at these links:
1)https://faucet.egorfine.com/,
2) https://faucet.dimensions.network/
3) https://Goerli.oregonctf.org/

(last link seems to give the most eth in the quickest amount of time)

Once on the website, go to your MetaMask extension and copy your account address by clicking address (will prompt you to copy to clipboard). Note: you need to do this when the network you are pointing to on MetaMask is the *Goerli* Testnet Network (not the Ethereum Mainnet). Once copied, paste your address on the faucet's website and follow the website instructions to receive Goerli ETH. Once the faucet sends the transaction you will be able to see your balance update on your MetaMask extension.

Note: these faucets are public and many developers globally are using it to request Goerli ETH. If the Goerli ETH does not show up in your account immediately, do not panic. You will be able to internally write and test your smart contract using the tools that come to follow. If you successfully requested Goerli ETH and your balance doesn't update in 24 hours, please reach out to the course team and we will resolve the issue.

- *Set-Up Task 4 - Remix IDE and Solidity*

We recommend using the Remix IDE for this project (https://remix.ethereum.org). Remix is a browser-based IDE that easily connects to MetaMask, allowing you to develop, test, and deploy your Solidity smart contract, all directly in your browser. The best way to get familiar with any new development platform is to explore and try it out for yourself! Remix provides you with example contracts for you to explore in the default workspace (found on the left in your File Explorers tab) and detailed documentation here: https://remix-ide.readthedocs.io/en/latest/#

We will be helping you navigate and use Remix through our Solidity development live sessions.

If you would like to explore Solidity development on your own, the Solidity documentation is well detailed and can be found here: https://docs.soliditylang.org/en/v0.8.12/

Graded Tasks

You will be graded by submitting your deployed ERC-20 Smart Contract to the project submission link in Canvas. We will be testing your smart contract by calling the required ERC-20 methods and querying the required ERC-20 events over the public *Goerli* Ethereum Testnet. In total, the ERC-20 standard defines nine (9) methods and two (2) events. Only one (1) required method, approve(), will not be directly graded since it will be graded indirectly through a different method and one of the events (resulting in 10 total graded methods/events). Each successful method/event will be equally weighted for the final project grade (that is, each successful method/event results in a contribution of 10% towards the final project grade, if all methods/events work properly it will result in a final grade of 100% for Project).

Ira A. Fulton Schools of Engineering
Arizona State University

A *parameter spreadsheet* will give you the parameters we expect you to use so that each method returns to us our desired output (the spreadsheet file will have a row for each student). For example, we will give you the strings we want your smart contract to return to us when we call the name() method.

The following graded tasks define what the expected behavior for the functions and events are, as well as how we will be grading your contract after you submit it to the autograder. All functions should have the same function signatures as given in the instructions. The Solidity documentation is your friend as you learn solidity and how different functionality is represented in the language. I highly recommend spending time with solidity and remix before diving into the project-specific tasks.

➔ Graded Task 1 - function name() public view returns (string memory)

Expected Behavior of Function:

● This method returns the name of your token
● The name of your token is of type string
● The return can either be hardcoded for your token name or can return a variable you set with the name of your token

✔ How We Are Grading:

● Our grader calls the name() function for your contract, and evaluates whether or not the string returned is equivalent to the token name we provide in the project 1 parameter sheet.
● If equivalent, the task passes, if not, the task fails.

➔ Graded Task 2 - function symbol() public view returns (string memory)

Expected Behavior of Function:

● This method returns the symbol of your token
● The symbol of your token is of type string
● The return can either be hardcoded for your token symbol or can return a variable you set with the symbol of your token

✔ How We Are Grading:

● Our grader calls the symbol() function for your contract, and evaluates whether or not the string returned is equivalent to the symbol name we provide in the project 1 parameter sheet.
● If equivalent, the task passes, if not, the task fails.

➔ Graded Task 3 - function decimals() public view returns (uint8)

Expected Behavior of Function:

● This method returns the number of decimal places your token has, the decimals value of your token is of type uint8
● The return can either be hardcoded for your decimals value or can return a variable you set with the decimals value your token has

✔ How We Are Grading:

● Our grader calls the decimals() function for your contract, and evaluates whether or not the number returned is equal to the number we provide in the project 1 parameter sheet.
● If equal, the task passes, if not, the task fails.

● The decimal value we are requiring for all students is 0, so we are expecting a return of 0 when the function is called, and evaluating the response is equal to 0.

➔ Graded Task 4 - function totalSupply() public view returns (uint256)

Expected Behavior of Function:

● This method returns the total supply of your token, the total supply of your token is of type uint256
● The total supply of your token is equal to the amount of tokens your contract has in circulation
● Anytime your contract increases or decreases total token balances, including minting new tokens and initializing token balances in the constructor, the total supply should change to reflect the increase/decrease
● Transfers between addresses do not increase or decrease total supply, as token balances are changing addresses not increasing/decreasing the total supply of tokens in circulation

✔ How We Are Grading:

● Our grader calls the totalSupply() function for your contract, and evaluates whether or not the number returned is greater than or equal to the required balances for all other tasks.
● If the value returned is greater than or equal to the required balances for all of your other tasks combined, the task passes, if not, the task fails.

➔ Graded Task 5 - function transfer(address _to, uint256 _value) public returns (bool success)

Expected Behavior of Function:

● The function takes in as inputs 1) the address to send tokens to, and 2) the amount of tokens to send.
● When called, the function allows a message sender to transfer a specific number of tokens to another address, decrementing the message sender's balance and incrementing the input address's balance by the amount of tokens.
● The function must ensure the message sender has a token balance greater than or equal to the amount of tokens attempted to be sent. If the token balance of the sender is less than the amount they are attempting to send, return false.
● The function returns true or false based on whether the transfer was successful or not.

✔ How We Are Grading:

● Our grader calls the transfer function with the address provided in the task 5 column of the parameter sheet and attempts to make a transfer with a value less than 100 tokens.
● For our grader to be able to grade this function, the address provided in the task 5 column needs to have a balance of at least 100 tokens. You can transfer the tokens in the following way:
○ In your contract constructor, give yourself an amount of tokens or create a mint function that allows you to get tokens.
○ Then, once you deploy your contract on the *Goerli* Testnet, call the transfer function yourself and transfer our address at least 100 tokens. You will be able to interact with your deployed contract directly in Remix (the contract needs to be included in a block before interactions, so it may take some seconds before the ability to interact with it appears. You can track progress using a block explorer.).
● If the transfer is successful (had enough balance to send the _value amount, balance of our address decreases, balance of the _to address increases), the task passes, if not, the task fails.

➔ Graded Task 6 - function balanceOf(address _owner) public view returns (uint256 balance)

Expected Behavior of Function:

● When called, the function takes in an address as input and returns the token balance of the address.
● The token balance value returned is of type uint256.

✔ How We Are Grading:

● Our grader calls the balanceOf function with the address provided in the task 6 – address column of the parameter sheet as input, and checks that it has the balance specified in the task 6 - amount column of the parameter sheet.
● Similar to task 5, this task requires you to ensure the address we provide in the sheet has the required balance before you submit for grading. Thus, once you deploy your contract on the *Goerli* test network, use the transfer function to transfer the tokens to our address.
● If the balance value returned from the function call equals the value we provided in the parameter sheet, the task passes, fails otherwise.

➔ Non-Graded Task 7 - function approve(address _spender, uint256 _value) public returns (bool success)

Expected Behavior of Function:

● When called, this function allows a message sender to give another address the right to transfer tokens from their balance, up to a specific amount.
● The function takes in as inputs 1) the address the message sender wants to approve as an allowed spender of their balance, and 2) the amount of tokens that address is allowed to spend from the message sender's balance.
● Message sender's are allowed to approve a token amount greater than their current balance of tokens.
● The function returns true or false based on whether the approval was successful or not.

✔ How We Are Grading:

● Using the address you used to deploy the erc-20 contract with, you are to approve the address given in the parameter sheet's task 7 - address column (input 1) the amount given in the parameter sheet's task 7 - amount column (input 2).
● Our grader will use task 8 to assess whether or not task 7 functions properly.

➔ Graded Task 8 - function allowance(address _owner, address _spender) public view returns (uint256 remaining)

Expected Behavior of Function:

● This function takes 2 addresses as inputs.
● When called, this function returns the remaining number of tokens a given spender address (input 2) is allowed to spend from a given owner address (input 1).
● The function returns the remaining balance allowed as a number of type uint256.

✔ How We Are Grading:

● Our grader will call the allowance function and give the address you used to deploy your smart contract with as the owner address (input 1) and will give the address found in the parameter sheet we asked you to approve in task 7 as the spender address (input 2).
● We will evaluate whether or not the number returned is equal to the number we asked you to approve in task 7 (found in the task 7 - amount column in the parameter sheet).

● If equal, the task passes, if not, the task fails.

➔ Graded Task 9 - function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)

Expected Behavior of Function:

● When called, this function allows a message sender to transfer a given amount of tokens (input 3) from an address (input 1) to another address (input 2), given the following requirements are met:

○ The _value transfer amount of tokens is less than or equal to the remaining allowance the message sender has been approved to transfer from the _from address.

○ The _value transfer amount of tokens is less than or equal to the balance of tokens the _from address has remaining.

● A successful transferFrom call must decrement the _from address's balance of tokens and increment the _to address's balance of tokens by the _value amount.

● A successful transferFrom call must decrement the allowed amount msg.sender is approved to transfer from the _from address in the future by the _value amount.

● The function returns true or false based on whether the transferFrom call was successful or not.

✔ How We Are Grading:

● Our grader will call the transferFrom function and uses the address you used to deploy your smart contract with as the _from address, the address you approved in task 7 to spend from your address (found in task 7 - address column of the parameter sheet) as the _to address, and a number less than the amount we asked you to approve our address to spend as the _value.

● If the transfer is successful (the message sender has enough left allowed by the _from address to transfer the _value amount, the _from address has enough balance to send the _value amount, balance of _from address decreases, balance of the _to address increases, remaining allowance amount decreases), the task passes, if not, the task fails.

● The transferFrom call is successful if:

○ The address you approved in task 7 (found in parameter sheet) has enough left approved by the address you used to deploy your contract with to transfer the _value amount.

○ The address you used to deploy your contract with has enough of a balance left to transfer the _value amount.

○ The balance of the address you used to deploy your contract with decreases by the _value amount.

○ The balance of the arbitrary address we send tokens to increases by the _value amount.

○ The amount approved by the address you used to deploy your contract with for the address in task 7 to transfer decreases by the _value amount.

○ decreases),

○ If the transferFrom call is successful, the task passes, if not, the task fails.

➔ Graded Task 10 - event Transfer(address indexed _from, address indexed _to, uint256 _value)

This event allows other contracts and interfaces the ability to track when a token transfer occurs, detailing from which address, to which address, and how much. We will be querying this event after we transfer tokens to an arbitrary address (see Graded Task 5).

➔ Graded Task 11 - event Approval(address indexed _owner, address indexed _spender, uint256 _value)

This event allows other contracts and interfaces the ability to track when a successful approval() call occurs, detailing the approving address, the address that was approved to transfer tokens, and how much they are approved to transfer. We will be querying this event to assess the successful completion of Non-Graded Task 7.

Submission Steps

The autograder should not be used to debug or test function-by-function as you develop your erc-20 token. All testing of erc-20 functionality can happen through remix, and does not require deployment to the *Goerli* test network (which costs gas). The autograder should be used when you believe you have a working erc-20 contract and have gone through all the necessary steps to be graded.

You will be graded by going to the autograder link in week 8 and doing the following:

1) submitting the name of your token (found in the parameter sheet for task 1)
2) submitting your smart contract address,and
3) submitting the account you used to deploy the contract.

We will be grading by directly calling the methods and events on your live contract that is deployed on the testnet as specified in the instructions above. The output will be your grade, and which methods/events passed or failed. The grader automatically saves your highest score on our backend. The grader is not linked to Canvas's grading backend, so we will upload the grades all at once. Submissions are unlimited, once you are happy with the highest score you have received from the autograder your work is done.