**Exam: Treasury**

Your task is to create a Solidity smart contract that implements a Treasury. It can be used by different organizations to store their funds. The owner of the treasury can initiate withdrawals of funds from the treasury. Organization stakeholders can vote on the initiated withdrawals of funds (They vote on do they agree the funds to be withdrawn and later spend for the specified use case). The voting happens through the ERC-20 token that represents a user is part of the organization (stakeholder). The voting system should lock the tokens when a user vote and allow users to unlock their tokens after the voting period is finished.

# Project Structure

Create a Hardhat project as an environment for the involved contracts, following the recommended structure and guidelines provided by Hardhat.

# Smart Contracts Functionality

## ERC-20 Token

Implement a minimal ERC-20 token contract. Use an instance of the ERC-20 contract to test the Treasury System functionalities. E.g you can initially mint X tokens for the creator of the ERC-20 contract and distribute some tokens to every user that must be represented as a stakeholder of the organization.

## Treasury

Create a contract named "Treasury" that allows the organization to set up a treasury for its funds (in ETH).

### Store Funds

* **Functionality**
* Implement a function that allows everybody to lock ETH in the contract. That way everybody can contribute to the organization.
* **Execution Process:**
* Users, who want to contribute funds to the organization, call the **storeFunds()** function and provide the amount of ETH they wish to lock in the treasury.
* The smart contract receives and stores the contributed funds in the treasury.
* The contributed funds become part of the organization's total treasury balance and can be used for future withdrawals determined by the organization's shareholders.

### Initiate withdrawal

* **Functionality**
* The owner of the treasury can initiate the withdrawal of funds by providing details such as the amount requested for withdrawal, description (what the funds will be used for) and voting duration (Up to when stakeholders can vote on the requested withdrawal. Many different withdrawals can be initiated at a moment.
* **Execution Process**
* The owner of the treasury calls the **initiateWithdrawal()** function, providing the details of the withdrawal, including the amount requested, description, and voting duration.
* The smart contract creates a new withdrawal request with a unique identifier and stores the withdrawal details.
* Stakeholders of the organization can now participate in the voting process for this withdrawal request.

### Voting

* **Functionality**
* Users can vote on an initiated withdrawal with "Yes" or "No" (**Hint: Use enum or bool**).
* When a user casts a vote, he also provides the amount of tokens he wants to vote with. The more tokens are provided - the more powerful the vote is. The tokens should be locked in the contract and the user is unable to claim them back until the voting period ends. (This is needed to ensure the same tokens would not be transferred to another address and reused for voting more than once).
* **Execution Process**
  + Users, who are stakeholders of the organization, can cast their votes on the initiated withdrawal by calling the **vote()** function.
* Each user specifies their vote as either "Yes" or "No" using an enum or boolean value.
* When casting a vote, the user also provides the amount of tokens they want to vote with, which determines the voting power.
* The tokens used for voting are locked in the contract, preventing users from claiming them back until the voting period ends.  
  The sum for the chosen option (yes/no) is increased with the amount of tokens locked by the user

### Withdrawal execution

* **Functionality**
* The result of the voting is measured by how many tokens are collected for "Yes" and how many tokens are collected for "No".
* In case the result of the voting (after the voting period ends) is "Yes", the owner of the treasury can execute the withdrawal which is in practice withdrawal of the ETH amount to an address specified by him.
* **The owner can execute the withdrawal as well in case there are no votes at all.**
* The funds can’t be withdrawn in case the result of the voting is "No" or an equal amount of tokens is collected for both "Yes" and "No".
* **Execution Process:**
  + After the voting period ends, the owner of the treasury calls the **executeWithdrawal()** function.
  + If the result of the voting is "Yes" or there are no votes at all, the owner can execute the withdrawal.
  + The ETH amount requested for withdrawal is transferred to the address specified by the owner.
  + If the result of the voting is "No" or an equal amount of tokens is collected for both "Yes" and "No" the withdrawal cannot be executed.

# General Requirements

* Smart Contracts must be implemented using the **Hardhat Development Environment.**
* Smart Contracts must be written in **Solidity**.
* The application should have **Access Control** functionality. (e.g. treasury owner).
* Unit tests must be implemented for 100% of the **Initiate Withdrawal** and **Voting** functionalities logic.
* **Deployment** task must be implemented.
* **Interaction** tasks must be implemented for the **Store Funds** Functionality.

# Other requirements

* Proper **Licensing** information should be added.
* Properly implemented business logic to avoid exploits and bugs.
* Apply **error handling** and **data validation** to avoid crashes when invalid data is entered.
* Demonstrate use of programming concepts - Smart Contracts Security, Gas Optimization, and Design Patterns.

# Bonuses

* Implement "Unlock Tokens" functionality.
  + **Functionality**
* Every stakeholder that participated in the voting can now claim back its tokens (the ones used for voting) for a given initiated withdrawal when the voting duration is finished by specifying the ID of the initiated withdrawal and an address to which the tokens are to be transferred.
  + **Execution Process:**
    - Stakeholders call the **unlockTokens()** function, providing the ID of the initiated withdrawal and the address to which the tokens should be transferred.
* The smart contract verifies that the stakeholder has participated in the voting for the specified withdrawal and that the voting duration has ended.
* If the conditions are met, the smart contract checks what is the amount locked by the user to vote for the specific withdrawal and transfers the tokens back to the stakeholder's specified address.
* Unit tests for the **Store Funds**, **Withdrawal execution** and **Unlock Tokens** functionalities logic.
* Emit proper events for the main actions.
* Implement a task that logs all historical events for "voting" for initiated withdrawal with ID = 1 in the console.
* NatSpec **documentation** on the contracts and their external and public functions.
* **Deploy and Verify the** **Smart Contracts** on the Sepolia Ethereum Testnet network (An **Alchemy** provider must be used for interaction with the Sepolia). Add information about the addresses of the contracts in the README.md.
* Additional functionality or libraries outside the general requirements, with motivated usage.

# Project Submission

Create a .zip file of the whole hardhat project excluding node\_modules, artifacts, and cache folders.

1. **Assessment Criteria**

**General Requirements – 80 %**

**Other Requirements – 20 %**

**Bonuses – up to 20 %**

**Good luck! 😊**