

# Smart Contracts Exercise 02: Decentralized Voting System

## 1 Introduction

In this exercise, you will implement a smart contract of a decentralized voting system on the blockchain. The goal of this exercise is to familiarize yourself with the basics of the Solidity language.

### 1.1 Prerequisites

Ensure that you have already installed the following on your system:

- **Node.js** - <https://nodejs.org/en/> An open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
- **NPM**: Node Package Manager, which comes with Node.js.

Open your terminal and run the following commands to verify the installations:

```
$ node -v  
$ npm -v
```

Both commands should return the installed version numbers of Node.js and NPM respectively. Node.js provides the runtime environment required to execute JavaScript-based tools like Hardhat, while NPM is used to manage the packages and dependencies needed for development. It is recommended that you use NPM 7 or higher.

### 1.2 Project Set Up

To get started, visit the following [GitLab repository](#) and clone it to your local machine. This repository contains a template in which you will complete this exercise. After you clone the repository start with the following command within your project folder:

```
$ npm install
```

This will install all the necessary dependencies for the project. Your implementation will be in the file *contracts/Voting.sol*. In this file, there are #TODO comments where you should implement the required functionality. To fulfill this task you need to pass all the provided tests. You can run the tests with the following command:

```
$ npx hardhat test
```

There is also a deployment script in the *scripts* folder. You can deploy the contract to the local hardhat network with the following command:

```
$ npx hardhat run scripts/deploy.js
```

## 2 Specification: Voting Contract

### 2.1 Overview

The **Voting** contract is a simple implementation of a voting system using Solidity. It allows the contract owner to add candidates, and any address to vote exactly once for a candidate. The contract includes the following functionalities:

- The contract owner can add candidates.
- Any address can vote exactly once for a candidate.
- The contract tracks the number of votes each candidate has received.
- The contract tracks whether an address has already voted.
- A function to get the total number of candidates.
- A function to retrieve a candidate's name and vote count by index.
- A function to get the index of the winning candidate.

### 2.2 Solidity Crash Course

The **Voting** contract is designed to facilitate a decentralized voting system. Below are some solidity code snippets that you might find useful for implementing the contract and explanations of the key components.

#### 2.2.1 State Variables

**State variables** are used to store data permanently on the blockchain. They represent the contract's state and can be accessed and modified by the contract's functions.

```
// Address of the contract owner
address public owner;

// Dynamic array to store all candidates
Candidate[] public candidates;

// Mapping to track whether an address has already voted
mapping(address => bool) public hasVoted;
```

### 2.2.2 Structs

**Structs** are custom data types that allow you to group related data together. They are useful for organizing complex data structures within the contract.

```
/**
 * @dev Struct to represent a candidate.
 * @param name The name of the candidate.
 * @param voteCount The number of votes the candidate has received.
 */
struct Candidate {
    string name;
    uint voteCount;
}
```

### 2.2.3 Modifiers

**Modifiers** are used to change the behavior of functions in a declarative way. They can enforce rules or conditions before executing a function's code.

```
// Modifier to restrict access to the contract owner
modifier onlyOwner() {
    require(msg.sender == owner, "Only the owner can call this function");
    _; // Continue executing the function
}

function addCandidate(string memory _name) public onlyOwner {
    // Only the contract owner can call this function
}
```

### 2.2.4 Functions

**Functions** define the behavior of the contract. They can read and modify the contract's state, perform computations, and interact with other contracts or external systems.

### 2.2.5 Events

**Events** are used to log information on the blockchain that can be accessed by off-chain applications. They are essential for tracking contract activities and facilitating interactions with the user interface.

```
/**
 * @dev Event emitted when a vote is cast.
 * @param voter The address of the voter.
 * @param candidateIndex The index of the candidate voted for.
 */
event Voted(address indexed voter, uint indexed candidateIndex);

/**
 * @dev Event emitted when a new candidate is added.
 * @param name The name of the candidate to be added.
 */
event CandidateAdded(string name);
```

### 2.2.6 Functionality Provided by Solidity

Here are some useful code snippets you might need:

```
// Sender of the transaction
address sender = msg.sender;

// Amount sent with the transaction
uint amount = msg.value;

// Enforcing conditions
require(condition, "Error message");

// Casting arbitrary data to uint
uint number = uint(data);

// Empty address
address emptyAddress = address(0);

// Emit an event
emit eventName(parameters);
```