IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, November 2026
Impact Factor: 8.2

Implementation and Testing of Blockchain Smart Contracts for Storage and Decentralized Polling

Dr. K. Kasturi[1] and Aamir Ahmed[2]
[1] Professor, Department of Applied Computing and Emerging Technologies
[2]Student, Department of Applied Computing and Emerging Technologies
School of Computing Sciences, VISTAS, Chennai, Tamil Nadu, India
[1] kasturi.scs@vistas.ac.in,[2] aamirahmed2704@gmail.com

Abstract : *Blockchain technology makes it possible to store data and run program logic in a way that is transparent and tamper-resistant, without depending on a single trusted server. On the Ethereum platform, such programs are called smart contracts and are written mainly in the Solidity language. This paper presents two smart-contract tasks that were written, compiled, deployed and tested in Remix IDE, a free browser-based development environment, using its built-in simulated blockchain, so that no wallet, real cryptocurrency or software installation was needed. Task 1, Simple Storage, is a contract that keeps one signed integer in permanent storage. It provides increment and decrement functions that change the value by one, makes the value readable from outside through a public variable and a getValue function, and logs every change through a ValueChanged event. The contract was compiled with Solidity 0.8.28 and tested on Remix VM (Cancun); the output (result) show the value changing from 0 to -1 and back to 0, with matching event logs. Task 2, Polling System, is a decentralized voting contract. Any user can create a poll with a title, at least two options and a deadline. Each address may vote only once per poll and only before the deadline, which is enforced by mappings and a timestamp check, and a getWinner function declares the winning option, with a tie flag, once the poll has ended. Custom errors, events, a modifier and read-only view functions make every rule explicit and auditable. The output(result) show a poll being created and invalid votes being rejected by the contract.Together, the two tasks cover the core ideas of smart-contract development: state variables, functions, constructors, mappings, structures, events, access rules based on time and sender, and the complete write, compile, deploy, interact and verify workflow.*

Keywords: Blockchain, Ethereum, Solidity, Smart Contract, Remix IDE, Mapping, Event, Decentralized Voting.

I. INTRODUCTION

A blockchain is a ledger that is copied across many independent computers called nodes. Records are grouped into blocks, and each block is cryptographically linked to the previous one, so a recorded entry cannot be changed silently. Ethereum extends this idea: besides recording transfers of its currency, it can store and run programs. A smart contract is such a program. It lives at an address on the blockchain, owns permanent data (its state) and exposes functions that anyone can call. Because its code and data are public and are executed identically by every node, nobody can secretly change its rules.

1.1 Task 1: Simple Storage Smart Contract

Task 1 isolates the basic elements of every contract (stored data, functions that change it, functions that read it) in the smallest useful example. The requirements are:

- declare an integer variable inside a Solidity contract to store a value;
- write an increment function that increases the value by 1 and a decrement function that decreases it by 1;
- make the value readable from outside the contract, by a public variable or a read function;

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, November 2026
Impact Factor: 8.2

- compile, deploy and test the contract to confirm that both functions work correctly.

1.2 Task 2: Polling System Smart Contract

In a traditional online poll, the votes sit in a database owned by one organisation, whose administrator can edit counts, allow repeated voting or move the deadline, and voters have no way to verify the result. Task 2 solves this by placing the polling rules in a smart contract. The requirements are:

- create a structure for a poll containing a title, a list of options, an end time and a vote count for every option, using mappings;
- allow any user to create a poll by providing the options and a deadline;
- allow each address to vote only once, and only before the deadline;
- use mappings to store votes securely and prevent double voting;
- add a function that determines and returns the winning option after the poll ends.

II. TECHNOLOGIES AND TOOLS USED

Both tasks use only free tools that run in a web browser.

Technology / Tool	Used in	Purpose
Solidity 0.8.28	Task 1	Language and compiler version used for SimpleStorage.sol.
Solidity ^0.8.20	Task 2	Language version for PollingSystem.sol (0.8.20 or a later compatible 0.8.x release).
Remix IDE 2.6.9 (remix.ethereum.org)	Both	Browser editor, compiler interface, deployment panel and terminal.
Remix VM (Cancun / Osaka)	Task 1 / Task 2	Simulated in-browser blockchain with pre-funded test accounts; Cancun was selected for Task 1 and Osaka appears in the Task 2 output (Result)
Ethereum Virtual Machine (EVM)	Both	Runtime that executes the compiled bytecode and measures cost in gas.

Table 1: Technologies and tools

2.1 Solidity

Solidity is a statically typed language for Ethereum contracts. The features used in this report are: state variables (permanent storage); constructors (run once at deployment); events (searchable log entries); struct and mapping (grouping data and constant-time key-value lookup); custom errors and modifiers (cheap, reusable validation); and block.timestamp (current block time, used for deadlines). Since version 0.8, arithmetic is checked automatically, so an overflow or underflow cancels the transaction instead of giving a wrong result.

2.2 Remix IDE and Remix VM

Remix IDE combines a file explorer, Solidity compiler, deployment panel and terminal in one browser window. Its Deploy & Run Transactions panel reads the compiler's interface description (the ABI) and creates a button for every public function: orange buttons send transactions that change the state and cost gas, while blue buttons only read data. The Remix VM supplies several test accounts with fake ether, so contracts can be deployed and tested instantly and free of cost, and the account list allows several voters to be imitated. A state-changing call is a transaction that consumes gas; a view function only reads data and is free when called from outside.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, November 2026
Impact Factor: 8.2

III. DESCRIPTION

3.1 Task 1: Simple Storage Contract

3.1.1 Design

The contract holds exactly one piece of data, the signed integer storedValue, and offers a constructor, two functions that change it, two ways to read it and an event that reports every change. The type int256 was chosen because it can hold negative numbers, so decrementing below zero is valid. The source code is given in the above Listing 1.

3.1.2 Steps to Compile, Deploy and Test

1. Open remix.ethereum.org in a browser.
2. In the File Explorer, create contracts/SimpleStorage.sol and paste Listing 1.
3. Open the Solidity Compiler, select version 0.8.28 and click Compile; a green checkmark confirms success.
4. Open Deploy & Run Transactions, choose Remix VM (Cancun), enter 0 in the
5. _initialValue box and click Deploy.
6. Under Deployed Contracts, press decrement and increment (orange), then getValue
7. and storedValue (blue), and read the logs in the terminal.

3.2 Task 2: Polling System Contract

3.2.1 Overview

One contract, PollingSystem, stores all polls in a mapping from a poll id to a Poll structure. A creator defines a question, options and a deadline; a voter picks one option while the poll is open; and anyone can read the poll, the vote counts and, after the deadline, the winner. There is no administrator and no function that edits a poll after creation. A poll passes through three stages: it is created and receives an id; votes are accepted until the deadline; after the deadline (block.timestamp >= endTime) voting is closed and the winner can be requested.

3.2.2 Data Structure and Modifier

A poll is a struct holding its title, a dynamic array of options, the end time, the creator's address and two mappings: voteCount (option index to number of votes) and hasVoted (voter address to true/false). The counter pollCount is public and also serves as the next poll id. The polls mapping is private; access is provided through view functions. The modifier pollExists rejects any poll id that has not been created.

3.2.3 Creating a Poll: createPoll()

The function validates its input (non-empty title, at least two options, deadline later than the current block time), gives the poll the id stored in pollCount, increments the counter, stores the title, end time and creator (msg.sender), copies the options into storage, emits PollCreated and returns the id. After creation nobody can add, remove or rename an option or move the deadline.

3.2.4 Casting a Vote: vote()

The function takes the poll id and the zero-based option index and performs three checks in a fixed order: (1) the current time must be earlier than the end time, which is the time-based restriction; (2) the caller's address must not already be marked in hasVoted, which prevents double voting; (3) the option index must be within the options array. Only then does it mark the caller as having voted, add one to the chosen option's counter and emit Voted. If any check fails the whole transaction is reverted, so the voter record and the count are always updated together or not at all.

3.2.5 Finding the Winner: getWinner()

After the deadline anyone can call getWinner. It reverts with VotingStillOpen if the poll has not ended, so a partial result is never shown as final. It then scans the options once,
remembering the highest count and its index. A larger count becomes the new leader and clears the tie flag; an equal count from a different option sets the tie flag. In a tie the lowest index is returned and isTie is true. If nobody voted, option 0 is returned with isTie true, correctly showing that there is no real winner.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 6, November 2026 Impact Factor: 8.2

Custom error	Raised when
PollDoesNotExist	The poll id is greater than or equal to pollCount.
EmptyTitle / NeedAtLeastTwoOptions	A poll is created with an empty title or fewer than two options.
DeadlineMustBeInFuture	The end time is not later than the current block timestamp.
VotingClosed / VotingStillOpen	A vote is cast at or after the deadline / the winner is requested before it.
AlreadyVoted / InvalidOption	The same address votes twice / the option index is out of range.

Table 2: Custom errors defined in the contract

3.2.6 Steps to Compile, Deploy and Use

1. In Remix IDE create PollingSystem.sol and paste the complete contract.
2. In the Solidity Compiler select version 0.8.20 or higher and compile; a green checkmark confirms success.
3. In Deploy & Run Transactions choose Remix VM, select PollingSystem and click
4. Deploy.
5. Call createPoll with a title, an options array and a future Unix timestamp. For example, the arguments used in the test were "Best programming language ?", ["Python", "JavaScript", "Rust"] and 1798761600. The first poll receives id 0.
6. Call vote(pollId, optionIndex), changing the account in the account drop-down to imitate several voters. After the deadline call getWinner(pollId).

IV. OUTCOMES

4.1 Task 1: Simple Storage Results

The contract compiled without errors and was deployed on Remix VM (Cancun) with the initial value 0. Figure 1 shows the successful deployment (status Transaction mined and execution completed, block 2) and the value 0 read from storedValue. The value was then changed by calling decrement and increment, and the logs in the terminal were inspected.

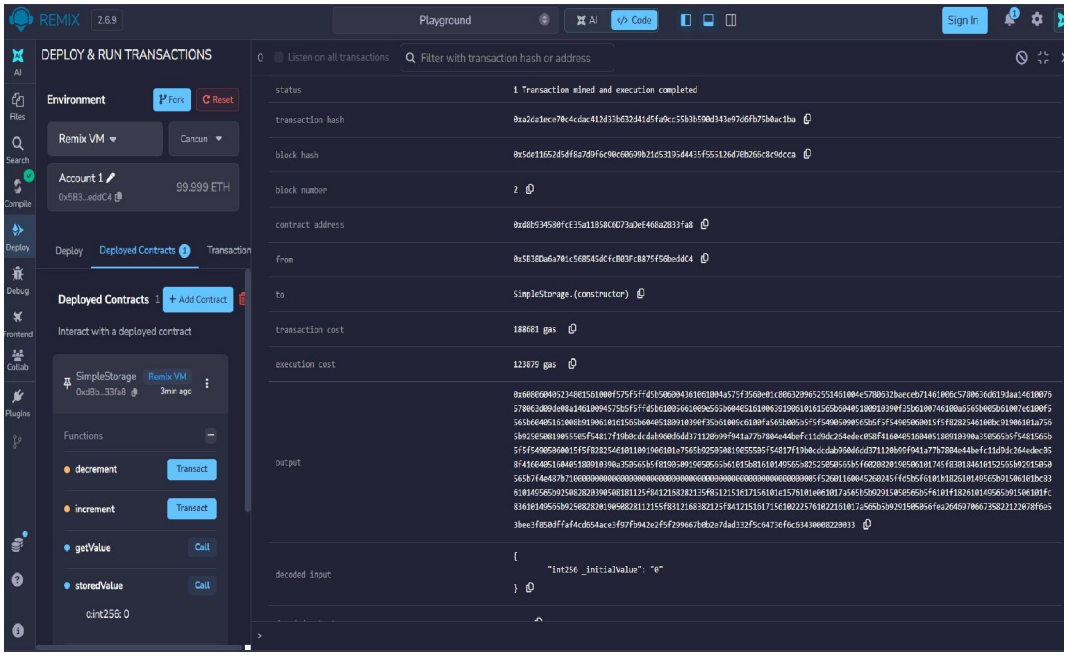


Figure 1: Deployment of SimpleStorage with _initialValue = 0; storedValue reads 0

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, November 2026
Impact Factor: 8.2

4.2 Task 2: Polling System Results

The contract compiled and deployed on Remix VM shows the createPoll transaction with the arguments "Best programming language ?", ["Python", "JavaScript", "Rust"] and the deadline 1798761600: the status is Transaction mined and execution completed, the transaction cost is 209,346 gas (185,630 execution cost) and one event log was produced, which is the PollCreated event.The validation rules were also demonstrated. When a vote transaction was sent for a poll id that did not exist, the contract refused it with the custom error PollDoesNotExist and reverted to the initial state (Fig. 2). The Transaction History in Fig. 3 lists the constructor (deploy), successful createPoll transactions and the rejected vote transactions, shown with red marks. A rejected transaction changes nothing, which is exactly the atomic behaviour required for a trustworthy voting system.



Figure 2: vote rejected with the custom error PollDoesNotExist

Figure 3: Transaction History showing deploy, successful createPoll and rejected vote transactions

Table 3 lists the planned test scenarios. The first and the last rows were observed in the transaction history the remaining behaviour follows directly and can be confirmed by repeating the steps with other accounts.

No.	Action	Result	Status
1	createPoll with valid title, three options, future deadline	Succeeds; id 0; PollCreated emitted	Observed
2	createPoll with empty title / one option / past deadline	EmptyTitle / NeedAtLeastTwoOptions / DeadlineMustBeInFuture	Expected
3	Account 2 votes for option 1	Succeeds; Voted emitted; count 1	Expected
4	Account 2 votes again; invalid option index	AlreadyVoted; InvalidOption	Expected
5	getWinner before the deadline; vote after the deadline	VotingStillOpen; VotingClosed	Expected
6	getWinner after the deadline	Winning option, its votes, isTie flag	Expected
7	vote with a non-existent poll id	Reverts with PollDoesNotExist	Observed (Fig. 2)

Table 3: Task 2 test scenarios

The five requirements of Task 2 are implemented: a Poll structure with mappings, poll creation by any user, one vote per address before the deadline, secure vote storage with double-vote prevention, and a getWinner function with a tie flag.

4.3 Limitations and Future Scope

- Task 1: anyone can call increment and decrement because there is no access control; an owner restriction, a reset function or a step size could be added.
- Task 2: one address is not one person, so identity checks (a whitelist or token ownership) are needed; votes are public, so secret ballots need commit-and-reveal or zero-knowledge methods; block timestamps can differ

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, November 2026
Impact Factor: 8.2

from real time by seconds.

Both: testing was manual on a simulated network. Future work includes automated tests (Hardhat or Foundry), deployment to a public test network such as Sepolia, a React and ethers.js front-end and a security audit.

4.4 Conclusion

Both tasks were completed successfully in Remix IDE. Simple Storage demonstrated state, functions, a constructor and an event in the smallest possible contract, and its output (result)confirm that the value changed from 0 to -1 and back to 0 with matching logs. The Polling System extended these ideas with structures, mappings, time-based restrictions and custom errors to build a transparent voting contract in which the rules, not a person, decide who may vote and who wins. Together they provide a firm foundation for building larger decentralized applications.

REFERENCES

[1] Solidity Documentation. Solidity Programming Language Documentation. https://docs.soliditylang.org
[2] Remix IDE. Remix IDE Documentation. https://remix.ethereum.org and https://remix-ide.readthedocs.io
[3] Ethereum Foundation. Ethereum Developer Documentation. https://ethereum.org/en/developers/docs/
[4] Buterin, V. (2014). Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. https://ethereum.org/whitepaper/
[5] Wood, G. (2025). Ethereum: A Secure Decentralised Generalised Transaction Ledger (Shanghai version). https://ethereum.github.io/yellowpaper/paper.pdf
[6] Luu, L., Chu, D.-H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making Smart Contracts Smarter. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 254–269. https://doi.org/10.1145/2976749.2978309
[7] Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A Survey of Attacks on Ethereum Smart Contracts (SoK). Principles of Security and Trust, 164–186. https://doi.org/10.1007/978-3-662-54455-6_8
[8] Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2018). Blockchain Technology Overview (NIST IR 8202). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8202
[9] OWASP Foundation. (2024). Smart Contract Security Verification Standard (SCSVS), version 0.0.1. https://owasp.github.io/www-project-smart-contract-security-verification-standard/
[10] Solidity Documentation. Security Considerations. https://docs.soliditylang.org/en/latest/security-considerations.html.