AIM: Elements of Distributed Computing.

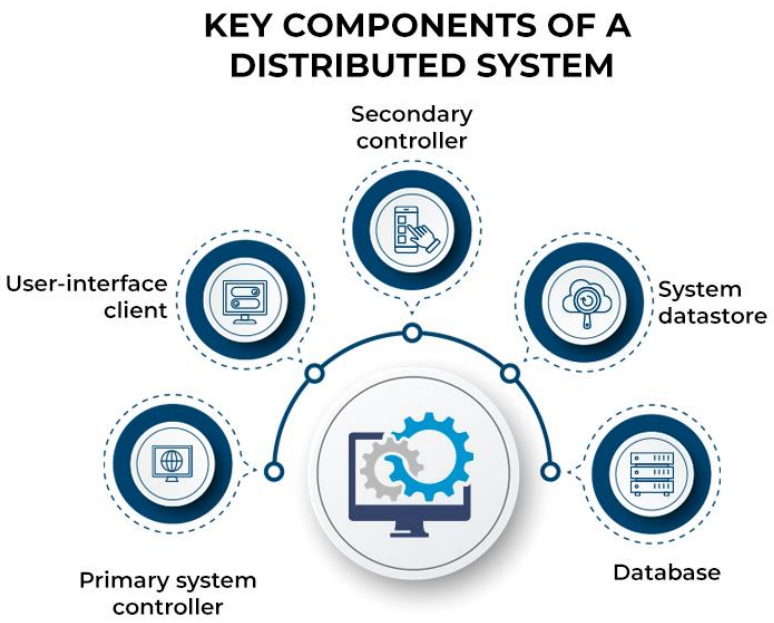
DESCRIPTION:

Distributed computing refers to a system where processing and data storage is distributed across multiple devices or systems, rather than being handled by a single central device. In a distributed system, each device or system has its own processing capabilities and may also store and manage its own data. These devices or systems work together to perform tasks and share resources, with no single device serving as the central hub.

There are several key components of a Distributed Computing System

Devices or Systems: The devices or systems in a distributed system have their own processing capabilities and may also store and manage their own data.
Network: The network connects the devices or systems in the distributed system, allowing them to communicate and exchange data.
Resource Management: Distributed systems often have some type of resource management system in place to allocate and manage shared resources such as computing power, storage, and networking.

The architecture of a Distributed Computing System is typically a Peer-to-Peer Architecture, where devices or systems can act as both clients and servers and communicate directly with each other.



Characteristics

There are several characteristics that define a Distributed Computing System

Multiple Devices or Systems: Processing and data storage is distributed across multiple devices or systems.

Peer-to-Peer Architecture: Devices or systems in a distributed system can act as both clients and servers, as they can both request and provide services to other devices or systems in the network.

Shared Resources: Resources such as computing power, storage, and networking are shared among the devices or systems in the network.

Horizontal Scaling: Scaling a distributed computing system typically involves adding more devices or systems to the network to increase processing and storage capacity. This can be done through hardware upgrades or by adding additional devices or systems to the network..

Advantages and Disadvantages

Advantages of the Distributed Computing System are:

Scalability: Distributed systems are generally more scalable than centralized systems, as they can easily add new devices or systems to the network to increase processing and storage capacity.

Reliability: Distributed systems are often more reliable than centralized systems, as they can continue to operate even if one device or system fails.

Flexibility: Distributed systems are generally more flexible than centralized systems, as they can be configured and reconfigured more easily to meet changing computing needs.

There are a few limitations to Distributed Computing System

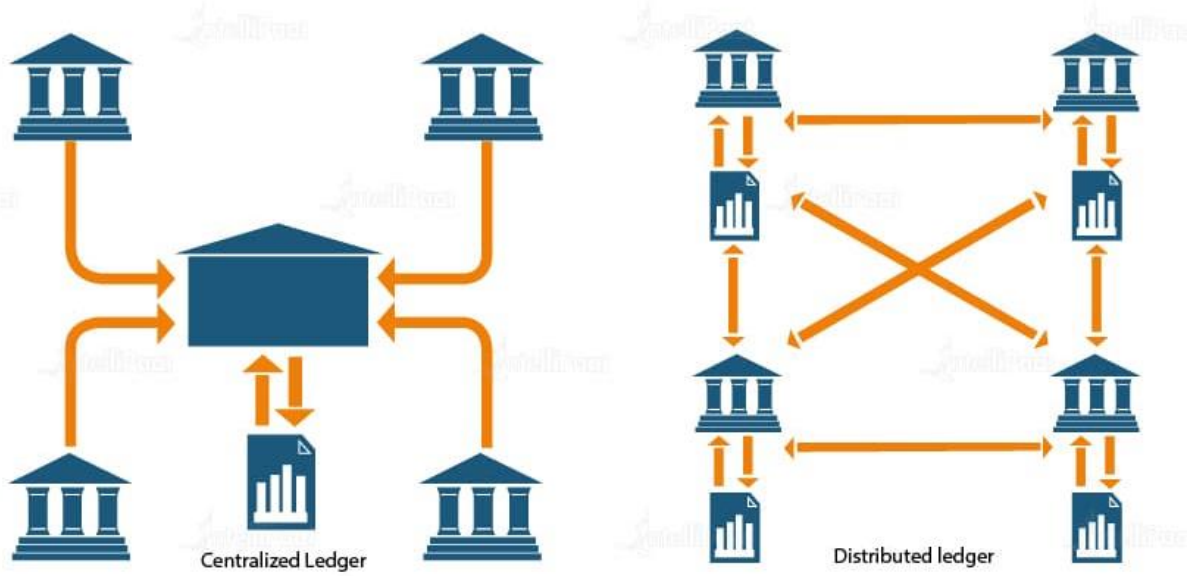
Complexity: Distributed systems can be more complex than centralized systems, as they involve multiple devices or systems that need to be coordinated and managed.

Performance: Distributed systems may not offer the same level of performance as centralized systems, as processing and data storage is distributed across multiple devices or systems.

Blockchain as a Distributed System:

Blockchain is a decentralized network that stores data on numerous servers, fostering trust among unknown peers. Initially associated with cryptocurrencies like Bitcoin, it now finds applications in Global Trade, Real Estate, Capital Markets, and Asset Management. It relies on a distributed immutable ledger where all transactions are recorded, ensuring transparency and security as transactions cannot be deleted or altered.

Distributed Computing is integral to managing the immutable ledger, ensuring every node maintains an up-to-date copy of the blockchain in real-time. This eliminates the need for a central authority, enhancing security and transparency.



Maintaining the ledger's current state is crucial in the distributed network of Blockchain. Miners validate transactions and create blocks, with one miner ultimately adding a block to the blockchain. This process relies on a consensus protocol, ensuring reliability and trust among peers. Distributed Computing plays a pivotal role in facilitating communication between nodes to reach a consensus.

In essence, Blockchain is a Distributed System that heavily relies on Distributed Systems' concepts and elements, making every computation within it a form of Distributed System Computing.

Cryptography in Blockchain:

Cryptography is a method of securing data from unauthorized access. In the blockchain, cryptography is used to secure transactions taking place between two nodes in a blockchain network. As discussed above, in a blockchain there are two main concepts cryptography and hashing. Cryptography is used to encrypt messages in a P2P network and hashing is used to secure the block information and the link blocks in a blockchain.

Cryptography is a technique or a set of protocols that secure information from any third party during a process of communication. It is also made up of two Greek terms, Kryptos term meaning "hidden" and Graphein, a term meaning "to write". Some terminologies related to Cryptography:

- **Encryption:** Conversion of normal text to a random sequence of bits.
- **Key:** Some amount of information is required to get the information of the cryptographic algorithm.
- **Decryption:** The inverse process of encryption, conversion of a Random sequence of bits to plaintext.
- **Cipher:** The mathematical function, i.e. a cryptographic algorithm which is used to convert plaintext to ciphertext(Random sequence of bits).

Types of Cryptography

The two types of cryptography are:

1. Symmetric-key Encryption: It focuses on a similar key for encryption as well as decryption. Most importantly, the symmetric key encryption method is also applicable to secure website connections or encryption of data. It is also referred to as secret-key cryptography. The only problem is that the sender and receiver exchange keys in a secure manner. The popular symmetric-key cryptography system is Data Encryption System(DES).

2. Asymmetric-key Encryption: This cryptographic method uses different keys for the encryption and decryption process. This encryption method uses public and private key methods. This public key method help completely unknown parties to share information between them like email id. private key helps to decrypt the messages and it also helps in the verification of the digital signature.

Wallets and Digital Signatures

A blockchain wallet is special software or a hardware device that is used to keep the transaction information and personal information of the user. Blockchain wallets do not contain the actual currency. The wallets are used to keep private keys and maintain a transaction balance. Wallets are only a communication tool to communicate to carry out transactions with other users. The real data or currency is stored in blocks in the blockchain.

Digital signatures are like proofs that the user gives to the recipient and other nodes in the network to prove that it is a legitimate node in the network to carry out transactions. While initiating a transaction with other nodes in the blockchain network, the user first has to create a unique digital signature by combining the transaction data with the user's private key using a special algorithm. This process will guarantee the authenticity of the node and the integrity of the data.

Hash Functions in Blockchain:

A hash function is a mathematical function that takes an input string of any length and converts it to a fixed-length output string. The fixed-length output is known as the hash value. To be cryptographically secure and useful, a hash function should have the following properties:

- **Collision resistant:** Give two messages m1 and m2, it is difficult to find a hash value such that hash(k, m1) = hash(k, m2) where k is the key value.
- **Preimage resistance:** Given a hash value h, it is difficult to find a message m such that h = hash(k, m).
- **Second preimage resistance:** Given a message m1, it is difficult to find another message m2 such that hash(k, m1) = hash(k, m2).
- **Large output space:** The only way to find a hash collision is via a brute force search, which requires checking as many inputs as the hash function has possible outputs.
- **Deterministic:** A hash function must be deterministic, which means that for any given input a hash function must always give the same result.
- **Avalanche Effect:** This means for a small change in the input, the output will change significantly.
- **Puzzle Friendliness:** This means even if one gets to know the first 200 bytes, one cannot guess or determine the next 56 bytes.
- **Fixed-length Mapping:** For any input of fixed length, the hash function will always generate the output of the same length.

Types of Cryptographic Hash Functions :

There are several different classes of hash functions. Some of the popular classes are:

1. RACE Integrity Primitives Evaluation Message Digest (RIPEMD): This set includes RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320.

2. Message-Digest Algorithm: This family comprises hash functions MD2, MD4, MD5, and MD6.

3. BLAKE2: BLAKE2 is a cryptographic hash function based on BLAKE, designed with the aim to replace MD5 and SHA-1 algorithms in applications requiring high performance in software.

4. BLAKE3: BLAKE3 is a cryptographic function based on Bao and BLAKE2. It is a few times faster than BLAKE2. This algorithm provides many features like parallelism, XOF, KDF, etc.

5. Whirlpool: It is a cryptographic hash function, first described in 2000. It is a modified version of the Advanced Encryption Standard (AES). Whirlpool produces a hash of 512 bits.

6. Secure Hashing Algorithm: The family of SHA comprises four SHA algorithms: SHA-0, SHA-1, SHA-2, and SHA-3.

Uses of Hash Functions in Blockchain :

The blockchain has a number of different uses for hash functions. Some of the most common uses of the hash function in blockchain are:

- **Merkle Tree:** This uses hash functions to make sure that it is infeasible to find two Merkle trees with the same root hash.
- **Proof of Work Consensus:** This algorithm defines a valid block as the one whose block header has a hash value less than the threshold value.
- **Digital signatures:** Hash functions are the vital part of digital signatures that ensures data integrity and are used for authentication for blockchain transactions.
- **The chain of blocks:** Each block header in a block in the blockchain contains the hash of the previous block header. This ensures that it is not possible to change even a single block in a blockchain without being detected.
-

Deprecation of Hash Functions:

Deprecated cryptographic algorithms are encryption and hashing techniques that are no longer considered secure due to vulnerabilities or weaknesses that make them unsuitable for modern cryptographic applications. As security research advances and computing power increases, older cryptographic algorithms can become vulnerable to attacks, which is why they are deprecated in favor of more robust and secure alternatives.

Some examples of deprecated cryptographic algorithms include - MD5 (Message Digest Algorithm 5, SHA-1 (Secure Hash Algorithm 1), DES (Data Encryption Standard), RC4 (Rivest Cipher 4), RSA with Small Key Sizes), ECDSA with Weak Curves, SHA-224

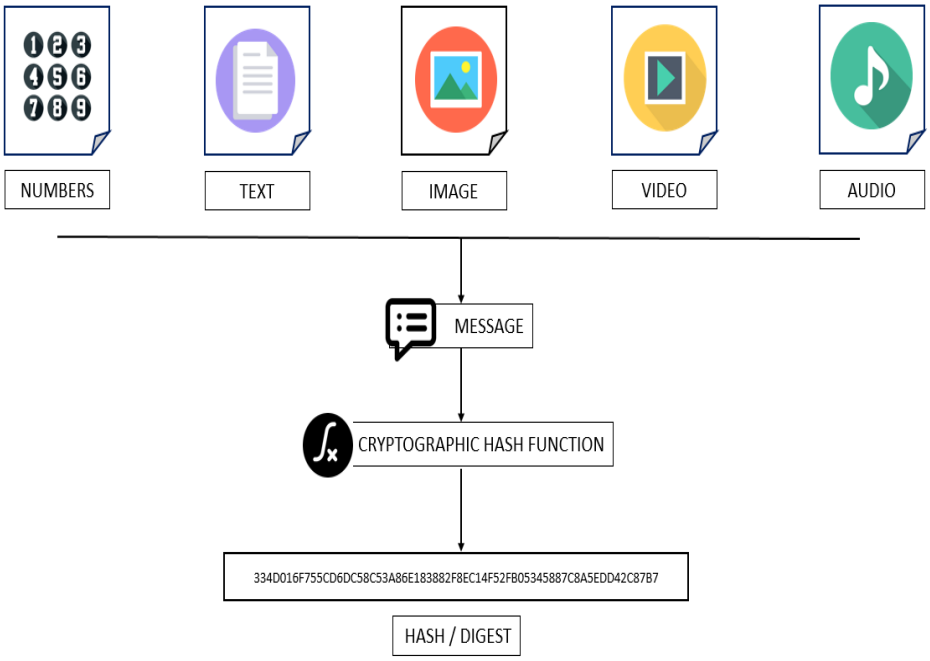
AIM: To learn and demonstrate different Cryptographic Hashing.

DESCRIPTION:

A cryptographic hash is a digest or digital fingerprints of a certain amount of data. In cryptographic hash functions, the transactions are taken as an input and run through a hashing algorithm which gives an output of a fixed size.

Cryptography is "Hashing" where any plain set of text can be turned into a cryptographic hash (a unique string of data) by using cryptographic algorithms. After sending data through a cryptographic hashing, it can't be reversed and hence it holds a distinguished significance other than the above two cryptography techniques. The hashing process can also be used to compress the data into a small string of text and hence can help in reducing a large chunk of data.

A hash function takes an input string (numbers, alphabets, media files) of any length and transforms it into a fixed length. The fixed bit length can vary (like 32-bit or 64-bit or 128-bit or 256-bit) depending on the hash function which is being used. The fixed-length output is called a hash. This hash is also the cryptographic byproduct of a hash algorithm.



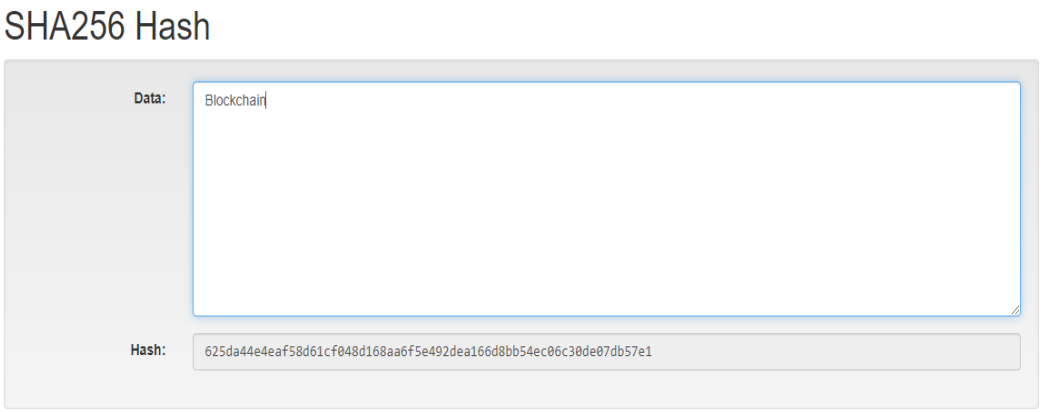
Properties of Cryptographic Hash Functions:

Deterministic:

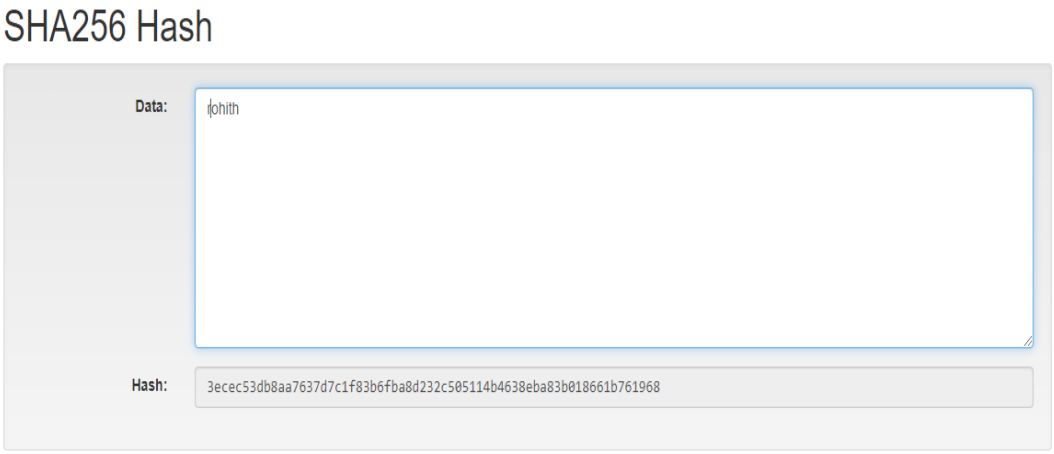
Cryptographic hash functions are deterministic. It always generates the same hash for the same input data. However, even if a single letter changes, the hash changes dramatically.

Impossible to go back:

Hash function is a one-way function, there is no way to get back entire text from the generated hash. This is different from traditional cryptographic functions like encryption where you can encrypt something using the key and by using decryption, you can decrypt the message to its original form.



Avalanche Effect: This means for a small change in the input, the output will change significantly. The avalanche effect in cryptography refers to the phenomenon where a small change in the input of a cryptographic function, such as a hash function, results in a significant and seemingly unrelated change in the output.

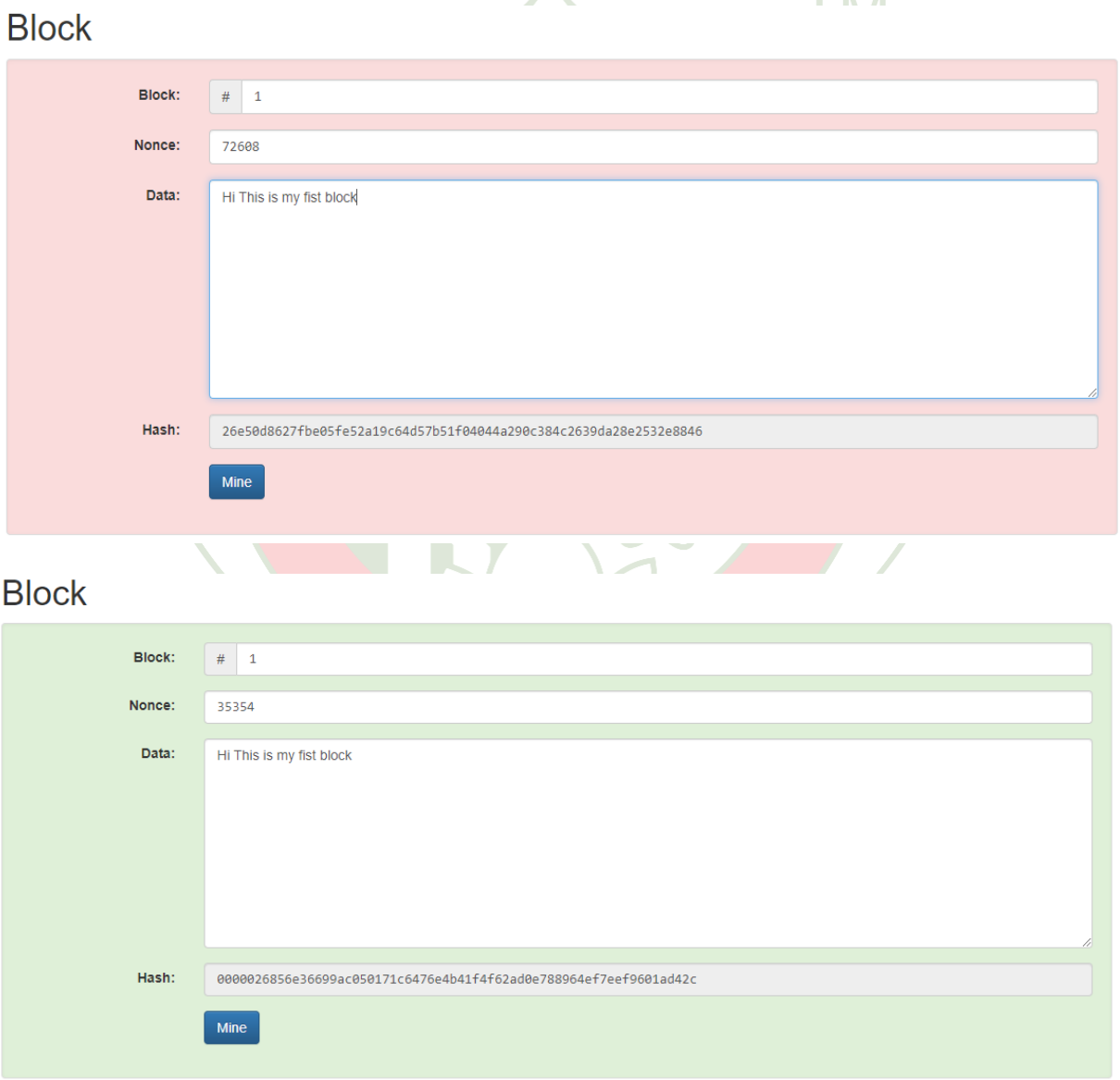


The hash value changes significantly with small change in the input value.

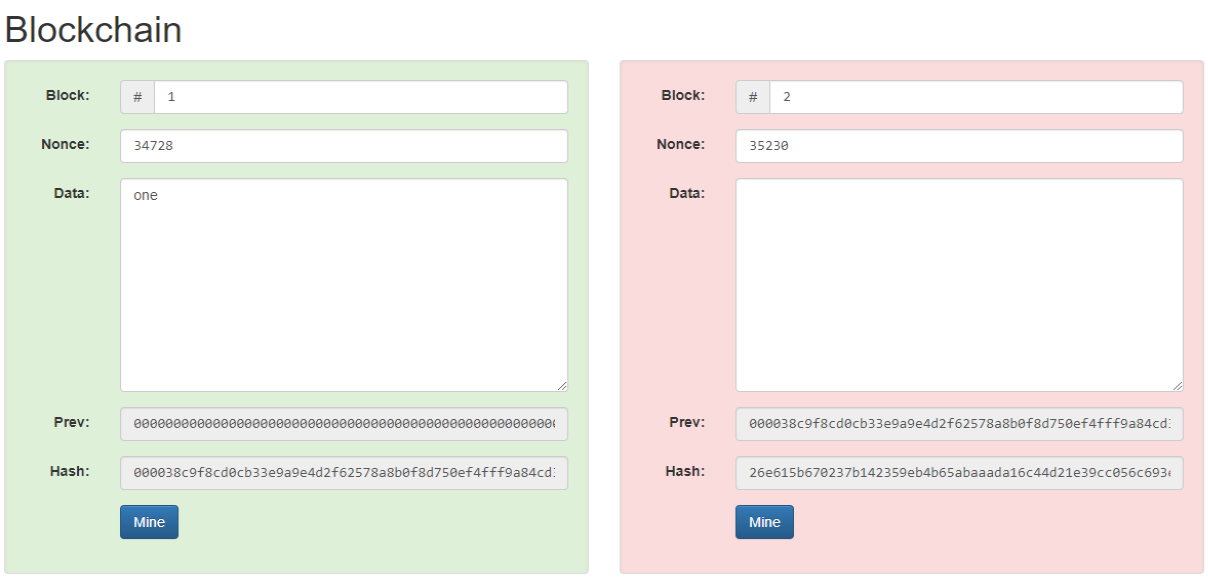
The length of the hash function remains the same for any input value.

Collision resistant: Give two messages m1 and m2, it is difficult to find a hash value such that hash(k, m1) = hash(k, m2) where k is the key value.

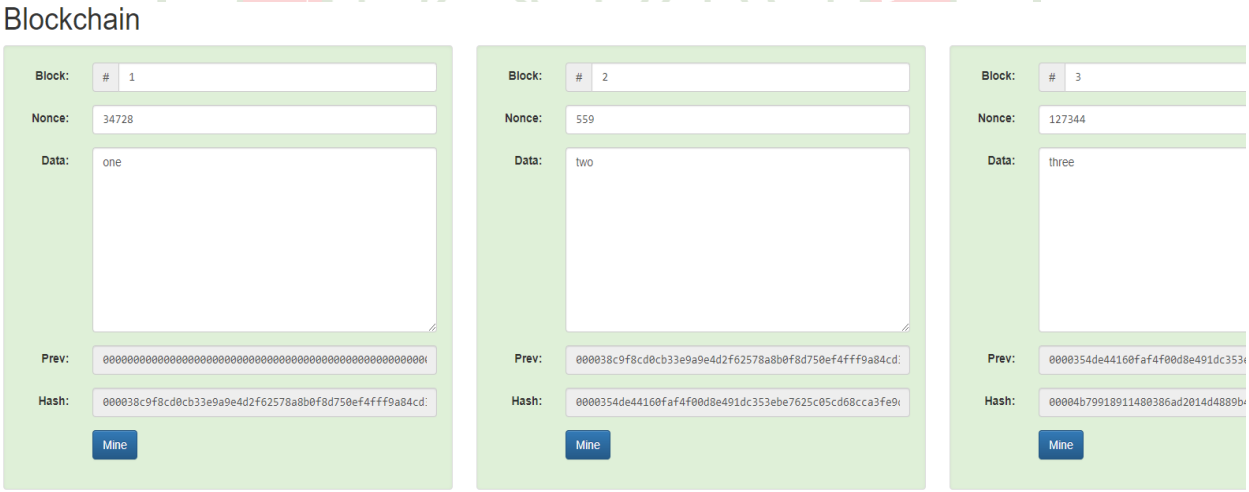
OUTPUT:



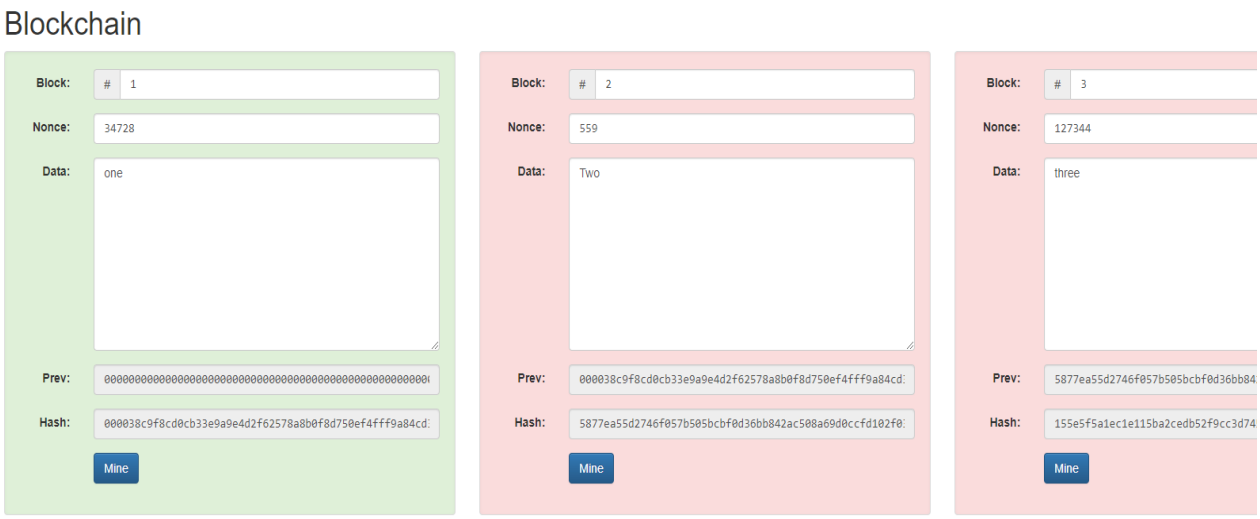
Once data is written onto a block, the hash function pertaining to the block changes which makes the block inadmissible. To validate the new block, a miner must mine the block.



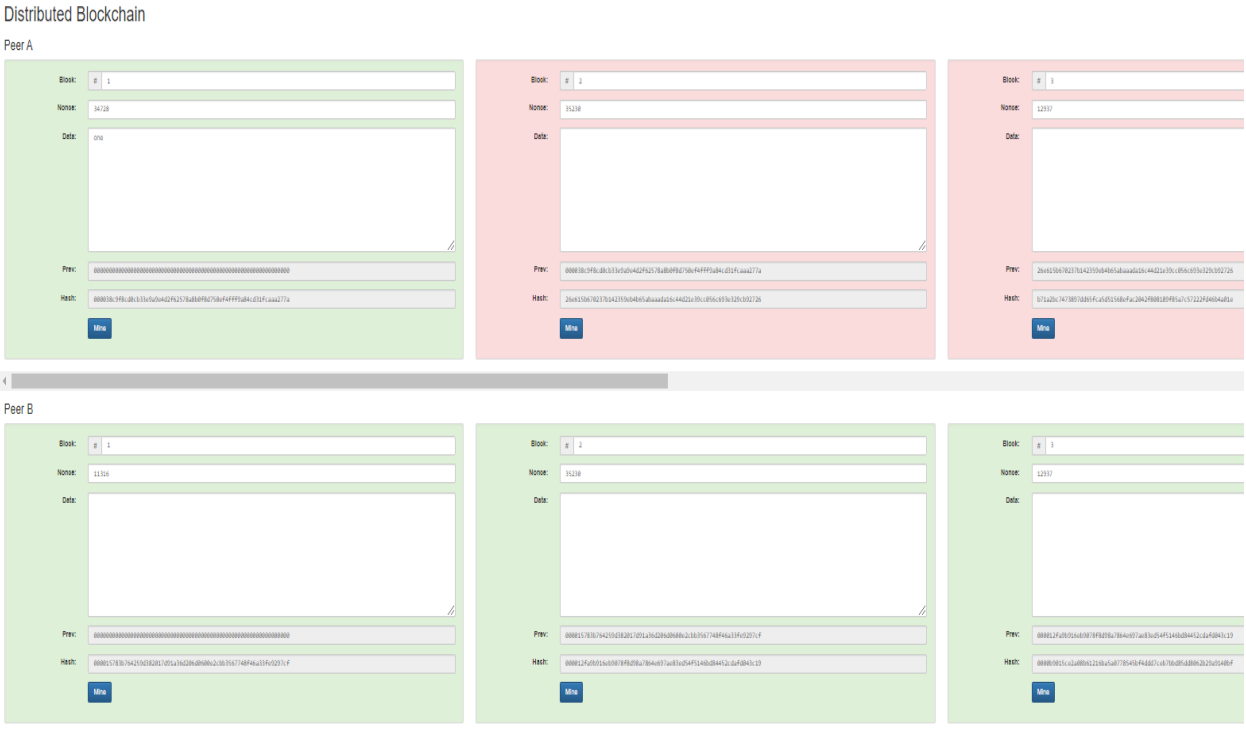
For every block, once the block is mined, it turns green colour. The shift from red to green colour indicates that the block is validated.

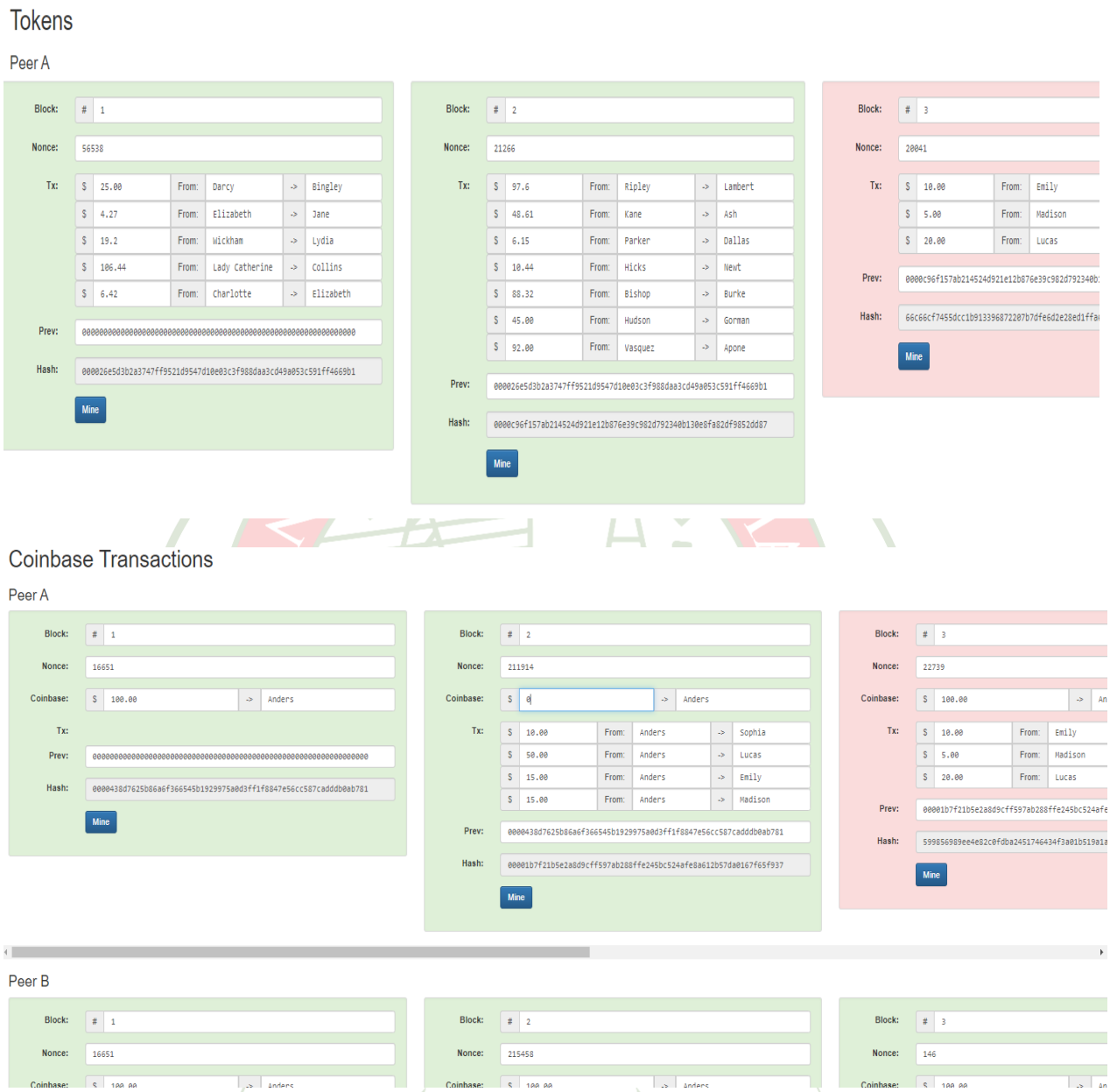


This process of validation must happen for all the blocks within that node as the previous hash for the next series of block changes one after another. So, to validate every block within the node, all the red blocks must be mined. This functionality of blockchain makes it highly secured and cyber-attack resistant. For an attacker to change the contents of a particular block, they must mine every other block on the chain. This makes it a time taking process for any intruder and any unauthentic change made on the chain is easily noticed.



Slight change in the value will make the block and all the subsequent blocks invalid indicating with red colour.





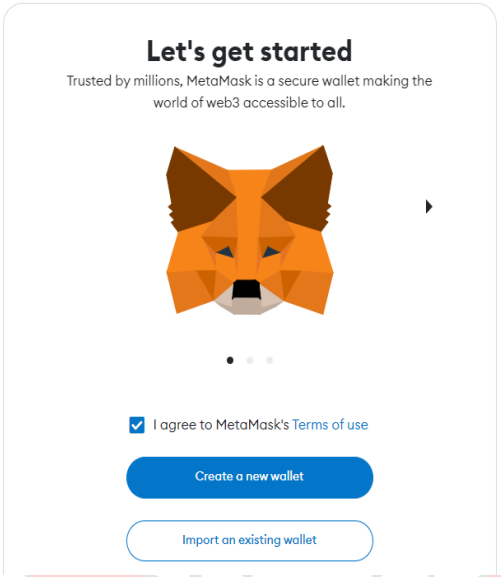
This is the representation of the transactions in the form of the blockchain. The above logic is applied to transactions between end users as well. If 10.0$ is sent from Anders to Sophia then every other block on the node must validate that transaction. Only then the transaction is considered valid. This validation is done through a distributed ledger that every user is by default provided with. This ledger helps to keep the track of all the transactions happening within the node thereby decreasing the risk of theft and variation in balances.

AIM: To create a MetaMask valet and perform transactions.

DESCRIPTION: MetaMask is a software cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications.

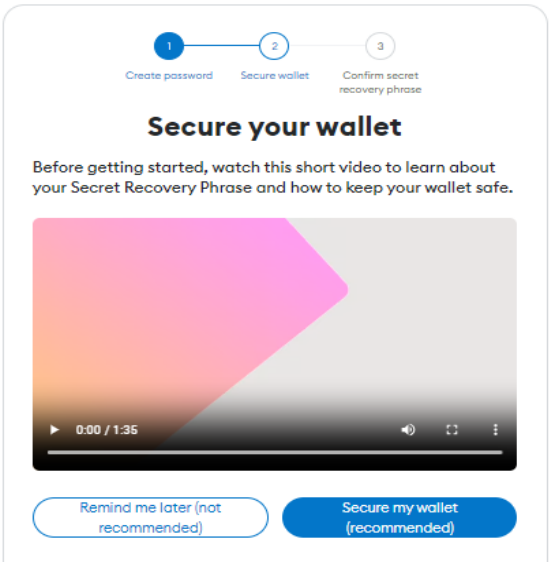
PROCEDURE:

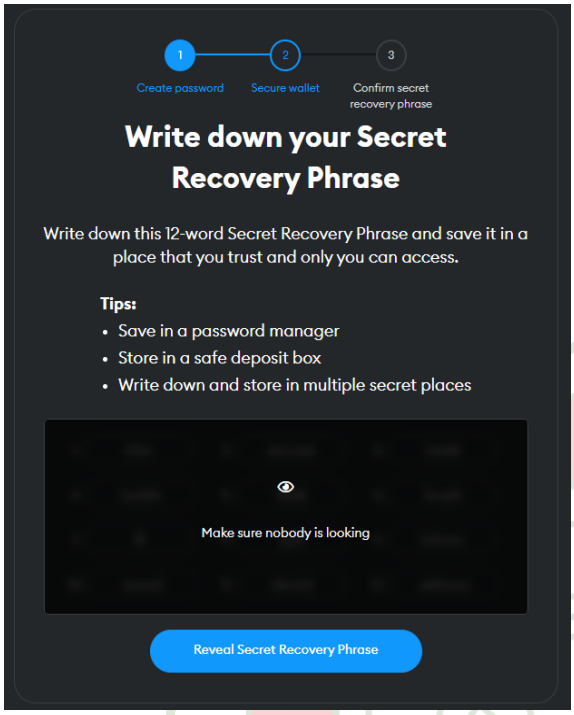
1.Add the MetaMask chrome extension to the chrome browser.

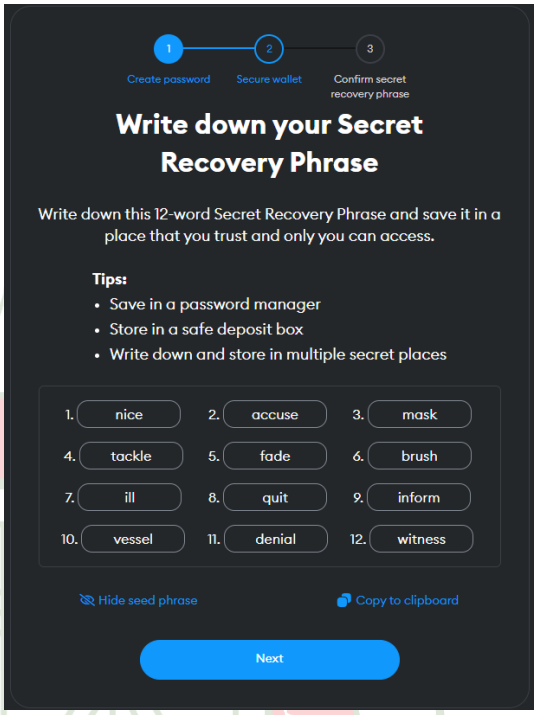


2.Click on create a new wallet.

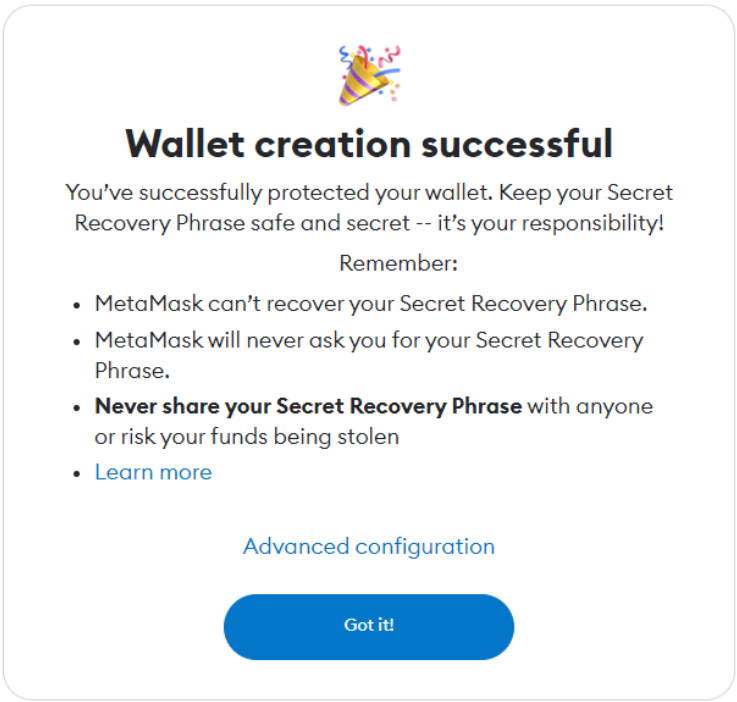
To create a MetaMask wallet, install the extension, set a strong password, secure your seed phrase, and access decentralized finance effortlessly.

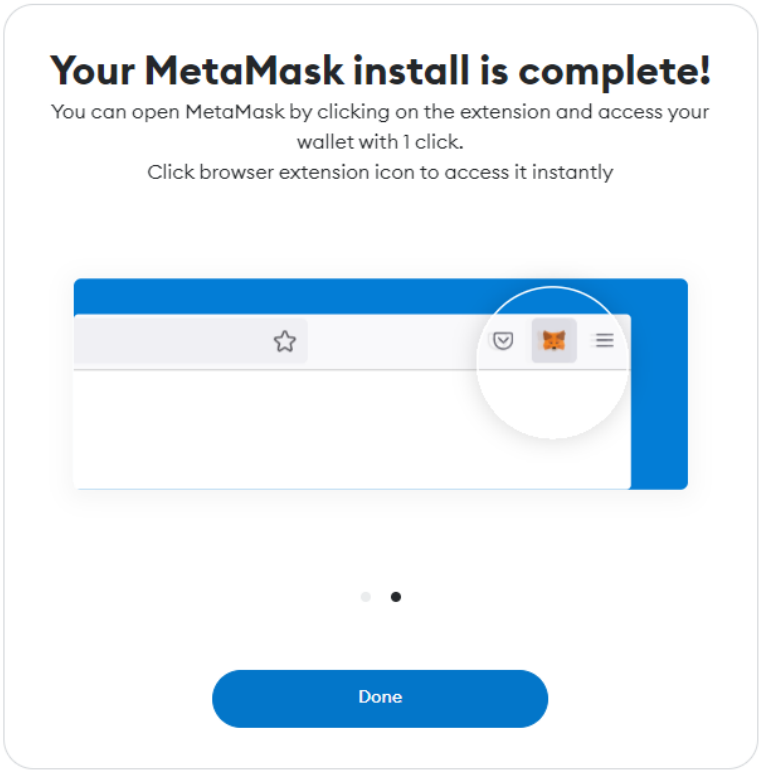




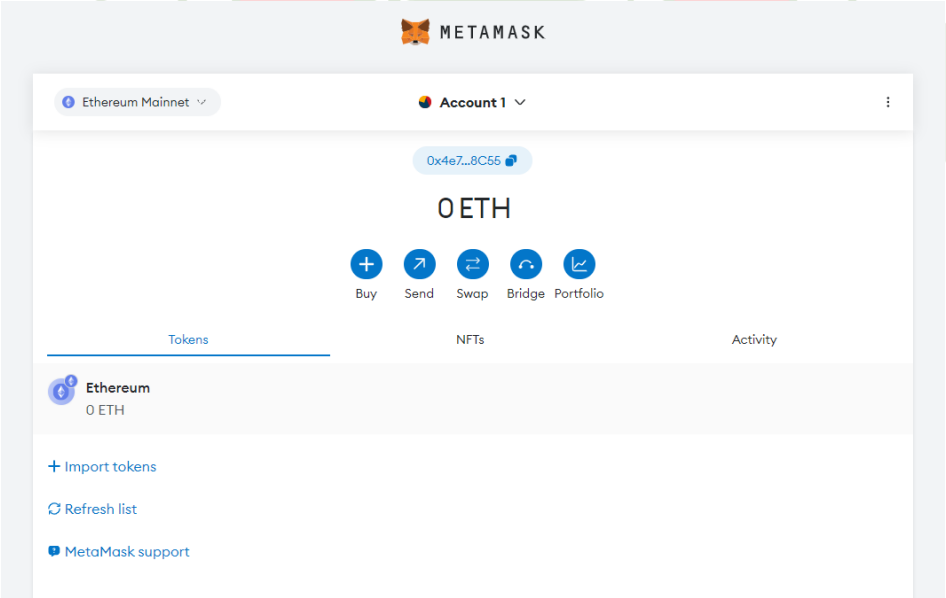


3.The MetaMask valet will be created.

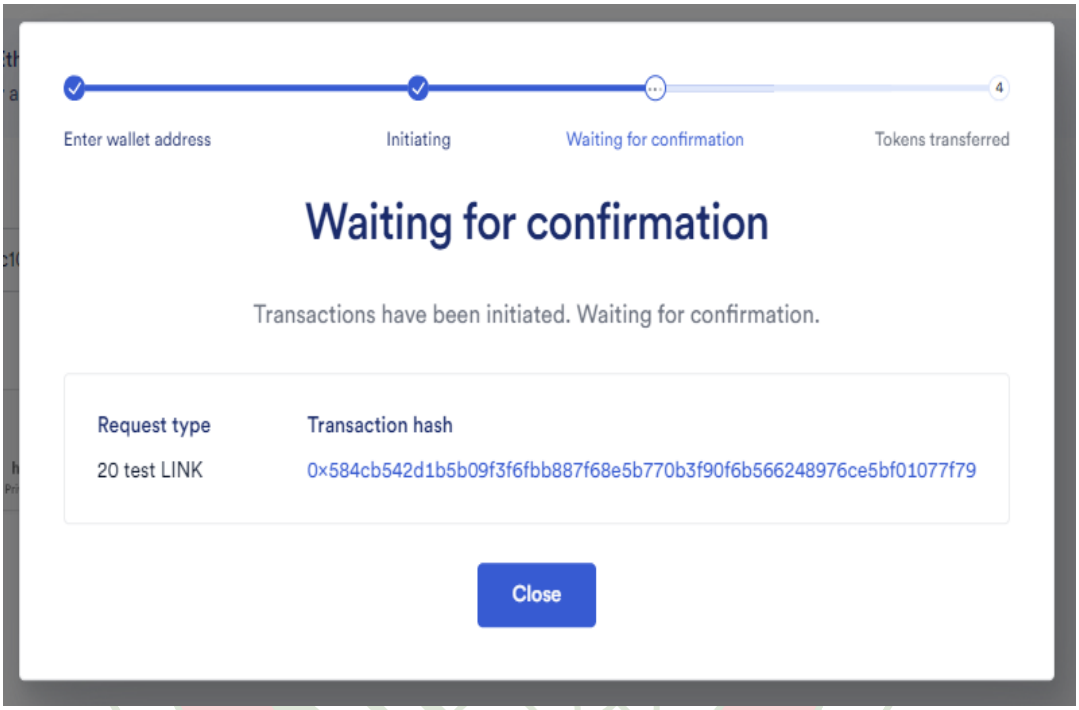


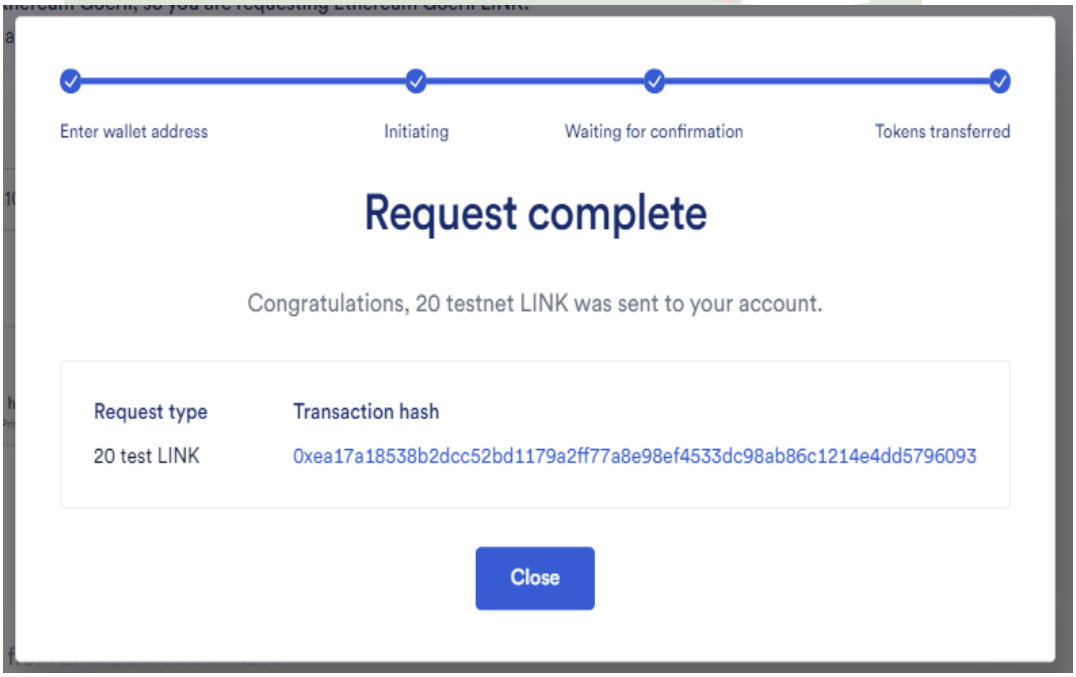


4.Metamask Home screen



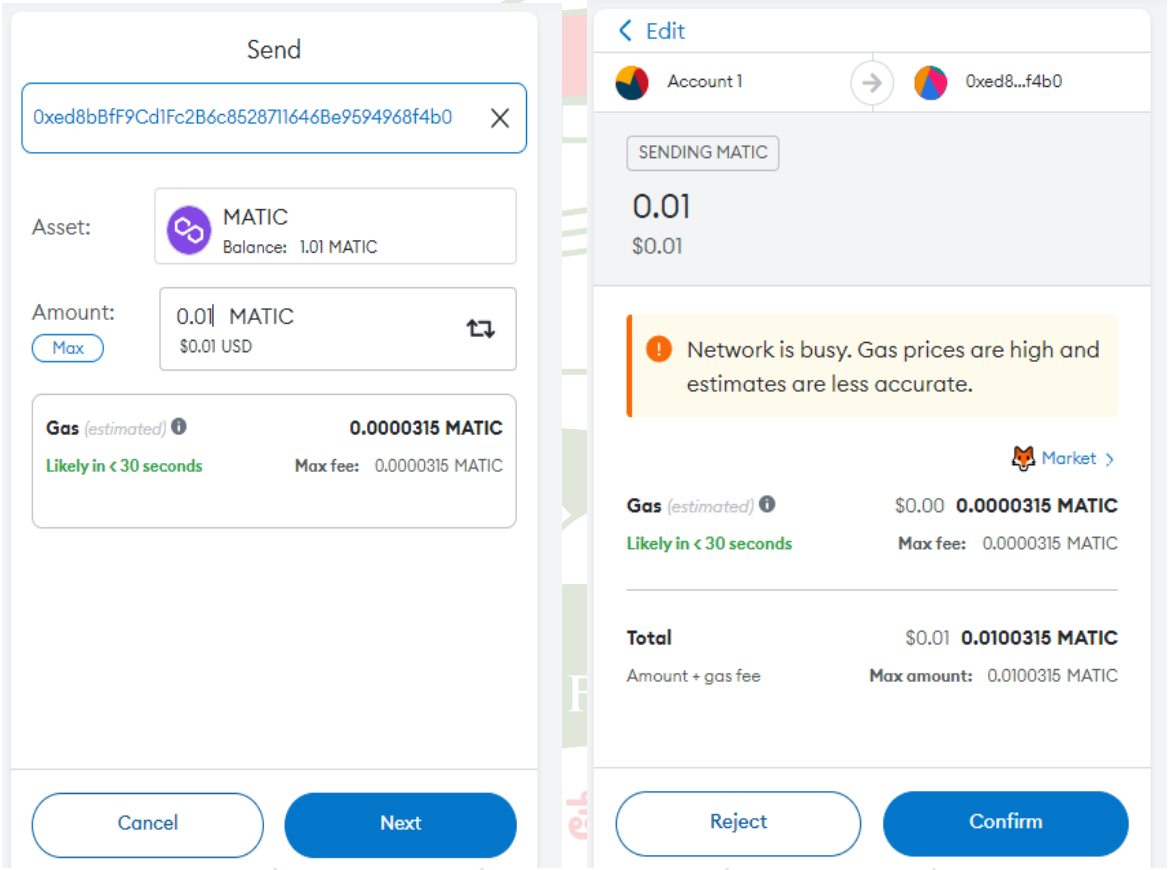
5. Go to https://faucets.chain.link/mumbai, and enter the MetaMask address and click on get test tokens.



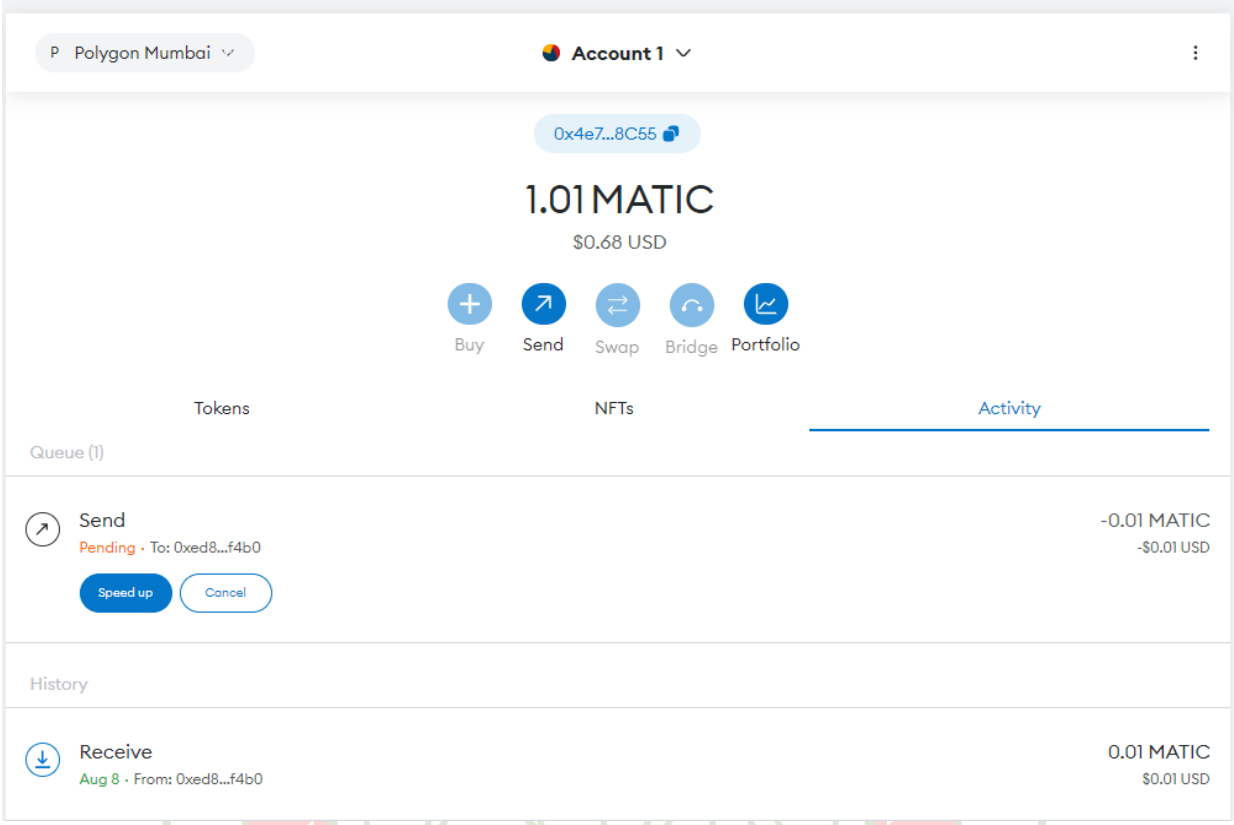


Performing transactions:

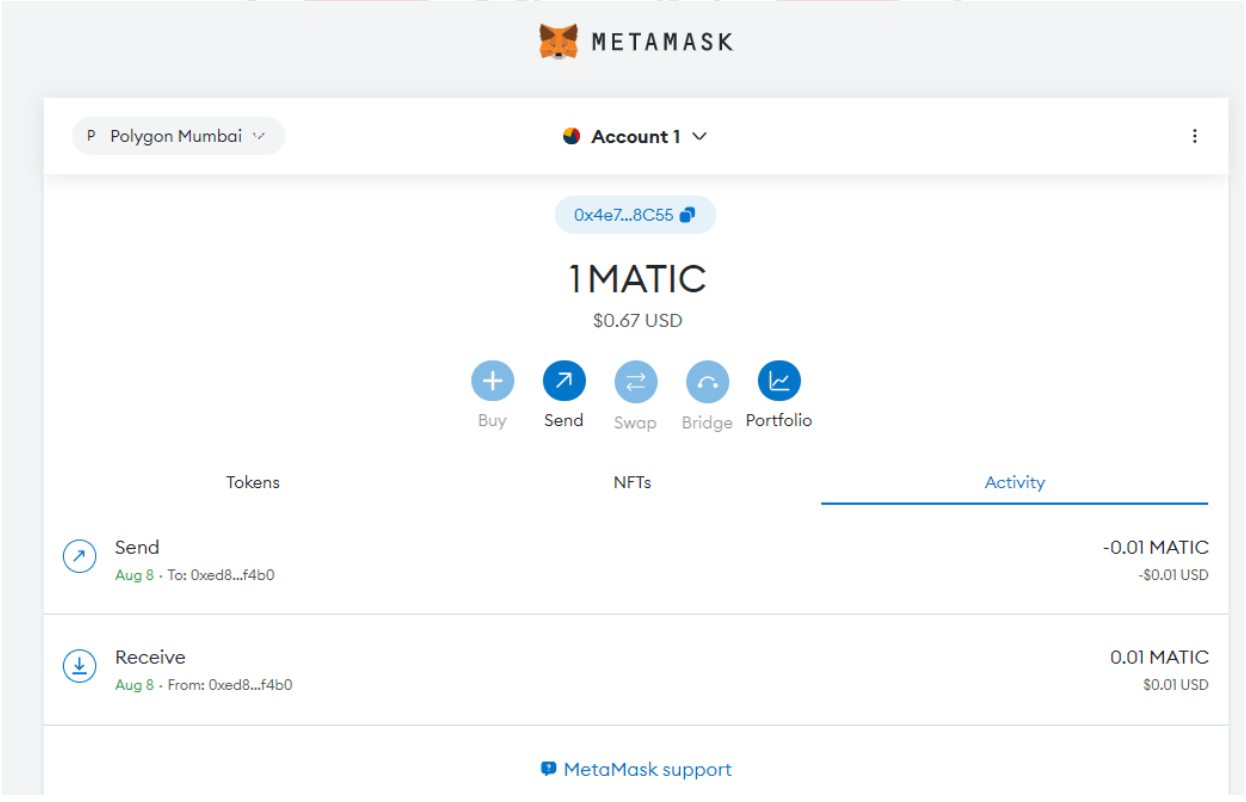
1. Click on the send button.
2. Enter the receiver address and click on next.
3. Enter the amount to be transferred.
4. The total amount with the gas fee will be displayed click on send.



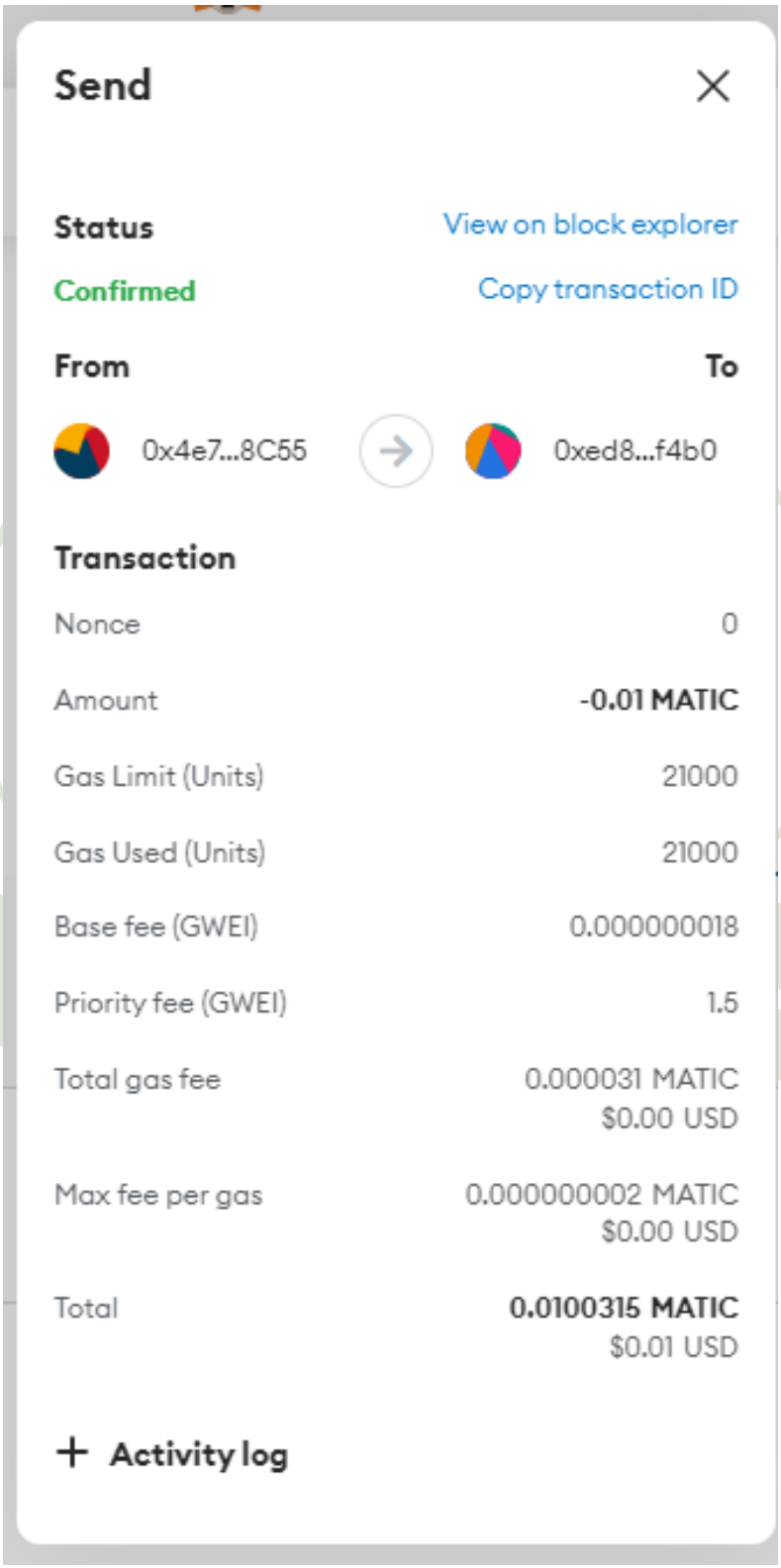
Gas fees in Ethereum are transaction costs paid to miners for processing transactions. When network congestion surges, fees rise due to heightened demand for limited resources, causing slower transaction confirmations and increased costs for users, impacting the overall user experience and the cost-effectiveness of using the network.



The transaction speed is directly proportional to the traffic on the chain. The average transaction speed in Ethereum varies, but it typically ranges from 10 to 30 seconds, depending on network congestion.



Transaction Details:



All the necessary details will be displayed in the transactions.

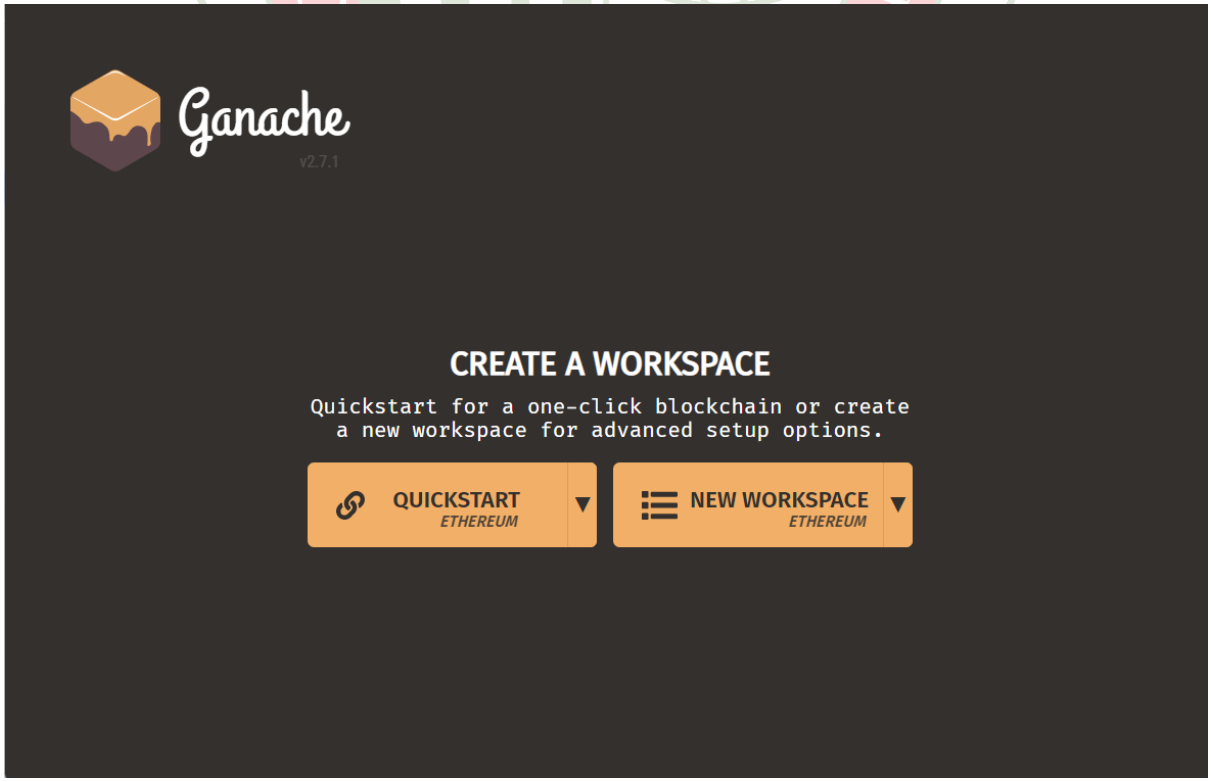
AIM: To setup Ganache environment and import ethers into MetaMask wallet.

DESCRIPTION:

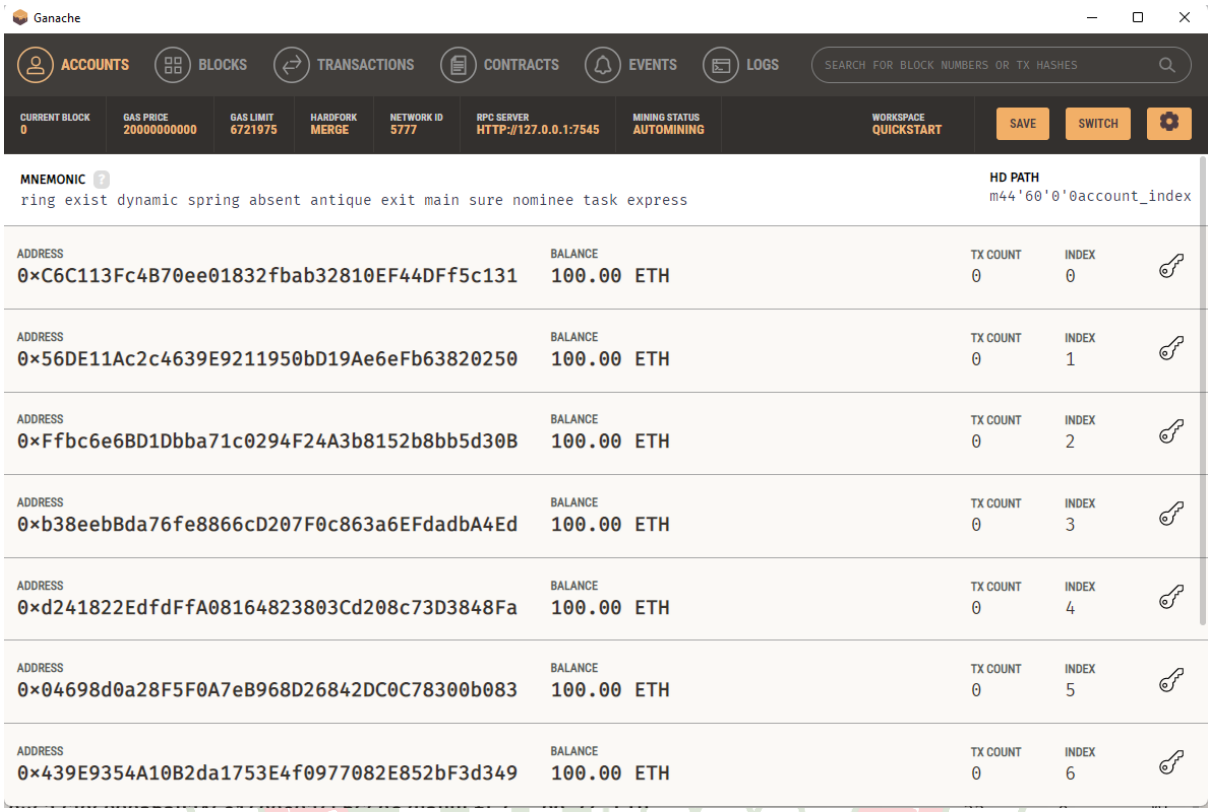
Ganache is a private Ethereum blockchain environment that allows to you emulate the Ethereum blockchain so that you can interact with smart contracts in your own private blockchain.

Ganache comes in two Flavors: a UI and CLI. Ganache UI is a desktop application supporting Ethereum and File coin technology. Our more robust command-line tool, ganache, is available for Ethereum development. It offers:

- console.log in Solidity
- Zero-config Mainnet and testnet forking
- Fork any Ethereum network without waiting to sync
- Ethereum JSON-RPC support
- Snapshot/revert state
- Mine blocks instantly, on demand, or at an interval
- Fast-forward time
- Impersonate any account (no private keys required!)
- Listens for JSON-RPC 2.0 requests over HTTP/WebSockets
- Programmatic use in Node.js
- Pending Transactions

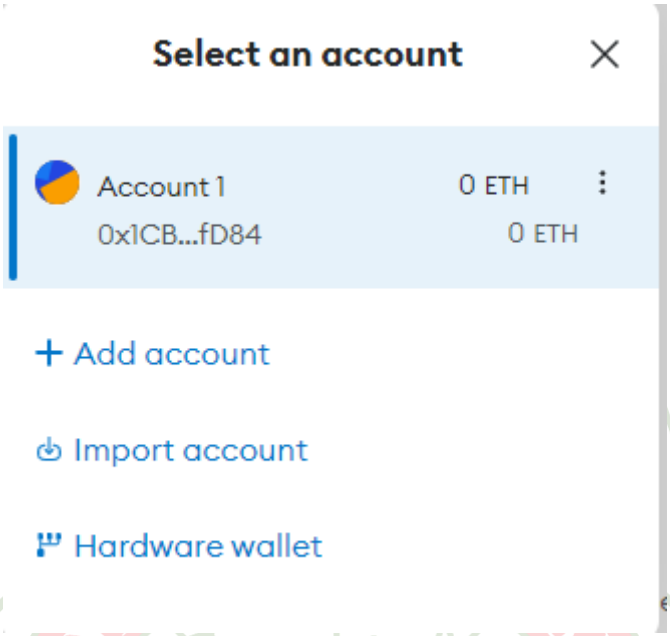


This is a home page of the ganache. Select the QuickStart option to start the Ethereum network.

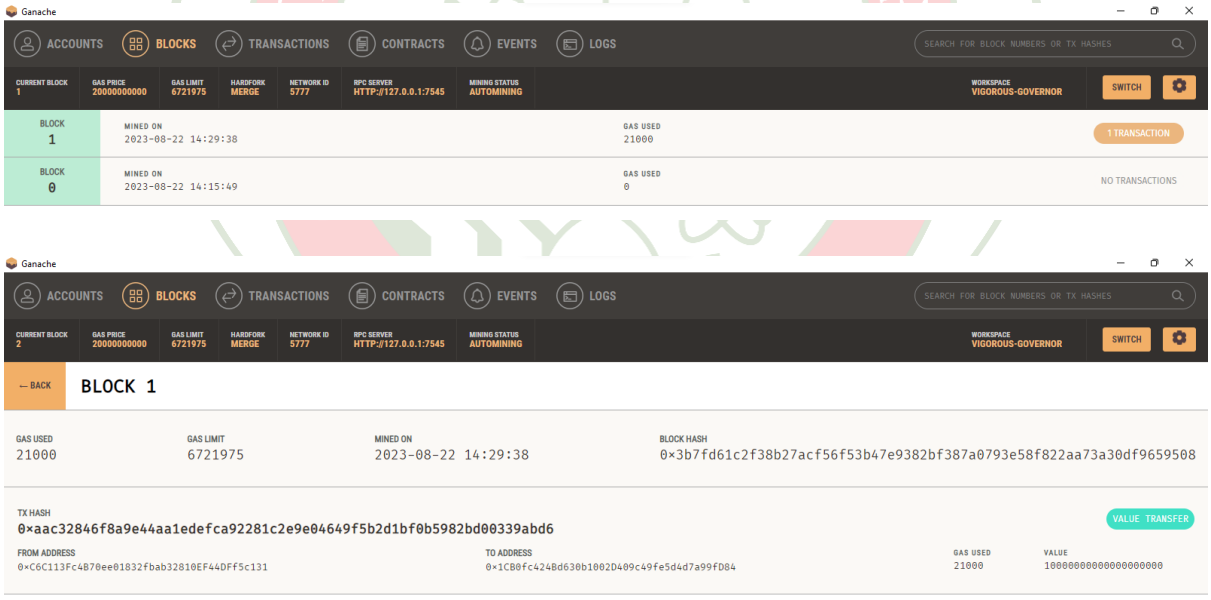


These are the free test accounts which can be imported and used. It gives 100 ETH for each wallet.

Copy the private key of the test account and paste it in the import account section. This private key enables us to import the 100 ETH that are present in the wallet to be transferred into out wallet.



The ethers are transferred to the account. These ethers are test ethers and can be used to run smart contracts and deploy real-time projects.



The transactions are reflected in the ganache interface in the Blocks section. Transactions on Ganache are synthetic, local Ethereum blockchain activities for testing and debugging smart contracts and decentralized apps (dApps) safely.

Usefull Ghanache-CLI commands:

- `-a` or `--accounts`: Specify the number of accounts to generate at startup.

```
C:\Users\CBIT>ganache-cli -a
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x8A22162Ad2E8eFD3F25d6D3Ea488E35551002444 (100 ETH)
(1) 0x48866d10dAEEA77bd3993794936a929B602BCC99 (100 ETH)
(2) 0xa311d510D7e27BF87fc6E0052F489f386B33930a (100 ETH)
(3) 0xd2F871f06D96d1AEA4676B442b551b0afBd63D65 (100 ETH)
(4) 0xF4f686ed1a4f44987BA68676faeaAa587bBcDc38 (100 ETH)
(5) 0x7675422e8FDF7461c8c4a9811f9725bE61ED3464 (100 ETH)
(6) 0x3AE9Fd21729a5e040600017648C11D5745957b44 (100 ETH)
(7) 0x25F9dafCEc4Abce8998403797eE3D3BFc648E98f (100 ETH)
(8) 0x0a3a5b73f0367aFDA927aD18dC3209328B073bE2 (100 ETH)
(9) 0xafE8262f1ACF438E869BA6Bd89AC1DfC7A876426 (100 ETH)

Private Keys
==================
(0) 0x64342fc838cb8e67d304aaf9e0669f24eed53758cf1fc60cc1c3b30656d1588d
(1) 0x9da6f23f5b90b67f7dc275ef92f8fd50a6b54160c01e311dccef10de885773e5
(2) 0x5b986a674c18e0ba0dc7e1864039c03ae83c7750da22472e7350c14ceb40f110
(3) 0xb5ba089a83700e6c49214ed6706285c8e478df81500ce351250b0c126dda6514
(4) 0x8c11931acadb587cad516e3b9059c2dfaf799943d2c141883736ff894d0d3f30
(5) 0x6bfc9f93060aea21723358061cfa54437b73b5e51b13c793b19f8f0bedbf2f3e
(6) 0x039c40df0a9615af2f13bfa435ddca861783db9a011af520151ab54da27a780f
(7) 0xe7f9e488195a90773e952dcc2541b3ba96c116973498a0f86384d80df160b7a7
(8) 0x5574277d6f96b37da3228fe7c1f1f49fc1817c6a5d05ee1965a21ffc0d7ec737
(9) 0xfc4b5d43c9235c7a39410216d030d95fb0573abd16b72f34f94133a0dc2ac6d7

HD Wallet
==================
Mnemonic:      huge evolve recipe common taste fat stuff social opinion ramp loan grocery
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
>
```

The ‘ganache-cli -a’ command allows developers to specify the number of accounts, offering flexibility for custom testing environments and precise simulation of blockchain scenarios.

- `-b` or `--blocktime`: Specify blocktime in seconds for automatic mining. Default is 0 and no auto-mining.

```
C:\Users\CBIT>ganache-cli -b
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0xcD62406f312bf74A6Dee08111c45E56D6e19182F (100 ETH)
(1) 0xe277c8861838D4c039BFd7A8d6a365E2539efB99 (100 ETH)
(2) 0x469f6E194d87aBF371b7b2A028B5B3e8EEa009Fe (100 ETH)
(3) 0xf419A682085CeC1E3CE4DeCd447A4902A640e233 (100 ETH)
(4) 0x1386943ba934320A8Ae4bE53654f814F48779bFC (100 ETH)
(5) 0xaB531CcAf2128728976214B20B009134250cF5fB (100 ETH)
(6) 0x748C6f97CD4193B6b4C02375c1b1bb0599C42146 (100 ETH)
(7) 0x9CB7fae29e4bb370Be9fA20336C8cD16eAA85117 (100 ETH)
(8) 0x5C5DF9b701dA76B9Bc3a76FAe0ae6837A46a8E83 (100 ETH)
(9) 0x070D5b2FD0212493E090235d69A226b24D396AfB (100 ETH)

Private Keys
==================
(0) 0x796f0132cef4d2cd51f5978762e94c39f96490c6bf2748549bea2675cd263b70
(1) 0x37efe6655759418ea88ba33e57f57014ad8ee25d531aaa46c747d5e0df828810
(2) 0xc3dff9ea08cb0456306fb725e3eadb04be7d914469128ff489026f627308f365
(3) 0xddaa44d5db237b12e4c671696bc11ddc69736850ccd8700240a8127054532f18
(4) 0x063cc73996787e7bacdff909526bcf98ee4d9353b442b66cc0dc60f8a0655db7
(5) 0x912b96a23535972e4ae2eae27bd3916f8151f6008c2918cb6f0de62750ce4b72
(6) 0xf0f47f0abac7e641248cc3a34b9244d66106e064ff771607e60d661bdff25939
(7) 0x2fc59249f9b7608d0d3d18a65466fd998db991d2cb0fdf4570f9c7f33a4ff1a0
(8) 0x644b70e668fab1e09eec5afddd102ee3ed301c5eea045479091e6930a784e6f8
(9) 0x86bc3e133e781e98825eb28d244a22bc98dff00c64f1db95e1ffca086fd81441

HD Wallet
==================
Mnemonic:      over episode weather shoulder busy run truck business arm arrow grant spike
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
```

The ‘ganache-cli -b’ command sets the initial account balance, granting developers the ability to control resources and emulate real-world scenarios while testing and refining blockchain applications.

- `-d` or `--deterministic`: Generate deterministic addresses based on a pre-defined mnemonic.

```
C:\Users\CBIT>ganache-cli -d
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (100 ETH)
(1) 0xFFcf8FDEE72ac11b5c542428B35EEF5769C409f0 (100 ETH)
(2) 0x22d491Bde2303f2f43325b2108D26f1eAbA1e32b (100 ETH)
(3) 0xE11BA2b4D45Eaed5996Cd0823791E0C93114882d (100 ETH)
(4) 0xd03ea8624C8C5987235048901fB614fDcA89b117 (100 ETH)
(5) 0x95cED938F7991cd0dFcb48F0a06a40FA1aF46EBC (100 ETH)
(6) 0x3E5e9111Ae8eB78Fe1CC3bb8915d5D461F3Ef9A9 (100 ETH)
(7) 0x28a8746e75304c0780E011BEd21C72cD78cd535E (100 ETH)
(8) 0xACa94ef8bD5ffEE41947b4585a84BdA5a3d3DA6E (100 ETH)
(9) 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e (100 ETH)

Private Keys
==================
(0) 0x4f3edf983ac636a65a842ce7c78d9aa706d3b113bce9c46f30d7d21715b23b1d
(1) 0x6cbed15c793ce57650b9877cf6fa156fbef513c4e6134f022a85b1ffdd59b2a1
(2) 0x6370fd033278c143179d81c5526140625662b8daa446c22ee2d73db3707e620c
(3) 0x646f1ce2fdad0e6deeeb5c7e8e5543bdde65e86029e2fd9fc169899c440a7913
(4) 0xadd53f9a7e588d003326d1cbf9e4a43c061aadd9bc938c843a79e7b4fd2ad743
(5) 0x395df67f0c2d2d9fe1ad08d1bc8b6627011959b79c53d7dd6a3536a33ab8a4fd
(6) 0xe485d098507f54e7733a205420dfddbe58db035fa577fc294ebd14db90767a52
(7) 0xa453611d9419d0e56f499079478fd72c37b251a94bfde4d19872c44cf65386e3
(8) 0x829e924fdf021ba3dbbc4225edfece9aca04b929d6e75613329ca6f1d31c0bb4
(9) 0xb0057716d5917badaf911b193b12b910811c1497b5bada8d7711f758981c3773

HD Wallet
==================
Mnemonic:      myth like bonus scare over problem client lizard pioneer submit female collect
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
```

The ‘ganache-cli -d’ command adjusts the default mining delay, enabling developers to fine-tune transaction speeds and explore blockchain performance under different conditions during testing and development.

- `-g` or `--gasPrice`: Use a custom Gas Price (defaults to 20000000000)

```
C:\Users\CBIT>ganache-cli -g
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x42dDFA0fB471C226C4311E5aDB513c3c045DF875 (100 ETH)
(1) 0x35d044a33597fcF2D5948bB30Ea539c99049dE6b (100 ETH)
(2) 0x04cCf0dB89839C590b7Ee84C088066e338cf4066 (100 ETH)
(3) 0x904CA5B0Ef24E8197024B9bF6688AD5677D8E61F (100 ETH)
(4) 0x3fCcc0dAFcda9ad8d4910Ea8ec04f19e90B3aC85 (100 ETH)
(5) 0x10F195Ac9f5b94654D9B054FB36c3e7A7802F776 (100 ETH)
(6) 0x118Cc7C233595c979859B61677702129797480f7 (100 ETH)
(7) 0x006d4baF0d6ABF5552c0bf9E6851DAb9E848F67F (100 ETH)
(8) 0xb02E66fc66b0b3EC2743865D94149bF9a04F66EF (100 ETH)
(9) 0x71BF4FE910b757e7b1Dbde0418e2969Fa56De53a (100 ETH)

Private Keys
==================
(0) 0x19b2d7bfea96efa9e8e875feb533377d2a4cccad60e2630caba66ad390ca94c7
(1) 0xec837edf2ac2c6f180ab45b726c6dc25d11ed26ce523b177d2e315eba84988bd
(2) 0xf0deafa06e19df1c86fa30f9d52e518fdba8ed6a7235883d5eb838642731db4c
(3) 0xd323f0a3d863ca77571620ceab305846e4c5fc4913947134c03f6a63fd6ffe58
(4) 0x00012641527df6da763775c51208493831462ae328e751b680d4ad61e4114409
(5) 0xbf0550b4698edb3ce4c06d4bb777b159552b6629271b97974a6b469236069d24
(6) 0x8f57dbd507c89d865c926ac0491b5389f91d924b4e2ece2c80593591ea35731f
(7) 0xe7c20a3c7717c13f024e220d5d4e8fcb10a8a61c87a87f05a9deeb8912928430
(8) 0xbc9f799cf254d29d5509e11ea78dd0813b609801aaab83e45d46713f4b0317bb
(9) 0xf68da53d47f93c5a9e983e3de54b6fda24c8eb2043b6a685ee53eb20efbbf3f7

HD Wallet
==================
Mnemonic:      badge label weird cloth adult secret action magic praise metal guilt renew
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
```

The ‘ganache-cli -g’ command customizes the gas price for transactions, providing developers with control over transaction cost simulations, essential for precise smart contract testing and optimization.

- `-l` or `--gasLimit`: Use a custom Gas Limit (defaults to 90000)

```
C:\Users\CBIT>ganache-cli -l
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x57821c8Cf47fB520Fa40dEa405aA681F688690e0 (100 ETH)
(1) 0x3337Fe7cfb05eDD9518Bf94f59326a4315f05b87 (100 ETH)
(2) 0x6a4a05f3E4A0A6f95cd58dc4FF9B83C4eFd4AAD4 (100 ETH)
(3) 0xb5E0fBE2e36022AFF96fC6B46f79ac4579156444 (100 ETH)
(4) 0x946FE2FBFc76e8544B71927861a382E07D9ED261 (100 ETH)
(5) 0x6d7F2bb8865dEDd3a2458f269cCB666c90984C33 (100 ETH)
(6) 0x920EB7a2850A3Cf66116e2a9707eE98B08968dA4 (100 ETH)
(7) 0x6A5E82FB4FaB8906E2118A04ED1Cccc062F293c2 (100 ETH)
(8) 0x59F17D1998E3Db6Ac87633A1b9EC1B516593260D (100 ETH)
(9) 0x75a6350985EbdF74773C754f4224b8C9F6454885 (100 ETH)

Private Keys
==================
(0) 0x54eb1bf6781f5fcddf9351ede5e497ca3e34974e819a3e0bbdd3212a43a63f6b
(1) 0x233b31ed6b75b26782ca637d94ac2b272ea2efb2cfe3efc9e21d6d3ec92f7669
(2) 0xf5d8adbd5ed75ffd410fee7503377adfbcd6689325ec991044ae47185609ed80
(3) 0x3b55b176d82547def162e5c1f06327f2f307f63be5c00bce75a8831cb1a1667a
(4) 0x5e78b35bfc071b23f6cf66393849eab0dc1443788b187f741337af8acb47095f
(5) 0x335100fc6a38d82998de2ea6da2bfd583aa40a4de6f85d603f96f94a5541deca
(6) 0xe2df60527e316401c01c4070c3a8dc798aa052261d3daaf2b71ccf869b2072e0
(7) 0x5e586ca3837bd7518f30137e9e24c78fac3669615729231cce29ba4737217d59
(8) 0xa5bde68edcbbac9aa9c40921956a6b465fce44d3bbcb1a1be2867a63be337cb6
(9) 0xbaacaa39371607393d1b8873dcbf67da639f422c7ca399542cc09b336bd088bf

HD Wallet
==================
Mnemonic:      barely fortune rich fork razor smile piece reopen fat fork valve innocent
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
```

The ‘ganache-cli -l’ command sets the gas limit, allowing developers to simulate diverse Ethereum network conditions and refine smart contracts for optimal performance and cost-efficiency during testing.

AIM: Introduction to Remix env. and basic solidity programs

DESCRIPTION:

Remix IDE is a no-setup tool with a GUI for developing smart contracts. Used by experts and beginners alike, Remix will get you going in double time. Remix plays well with other tools, and allows for a simple deployment process to the chain of your choice. Remix is famous for its visual debugger.

In Remix, everything is a plugin. Core plugins are preloaded for essential operations. Additional plugins offer an expanded palette of tools.The Remix team builds all core plugins and some additional plugins, all of which are designated by the green checkmark.

Remix IDE is used for the entire journey of smart contract development by users at every knowledge level. It requires no setup, fosters a fast development cycle, and has a rich set of plugins with intuitive GUIs. The IDE comes in two flavors (web app or desktop app) and as a VSCode extension.

Remix Online IDE, see: https://remix.ethereum.org

Supported browsers: Firefox, Chrome, Brave. We do not support use of Remix on tablets or mobile devices.

Solic commands :

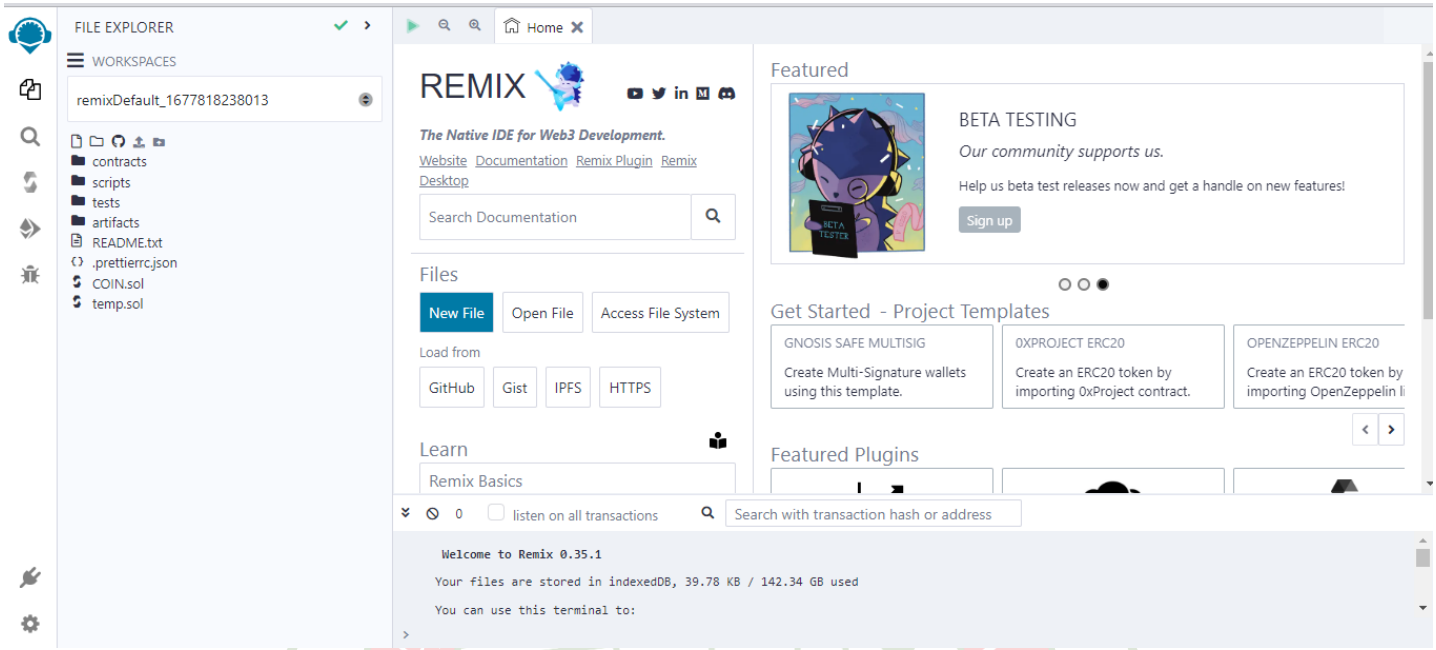
npm install –g solc

```
C:\Users\CBIT>npm install -g solc

added 9 packages in 6s

1 package is looking for funding
  run `npm fund` for details
```

solc –-version

```
C:\Users\CBIT>solcjs --version
0.8.21+commit.d9974bed.Emscripten.clang
```

Home Page :

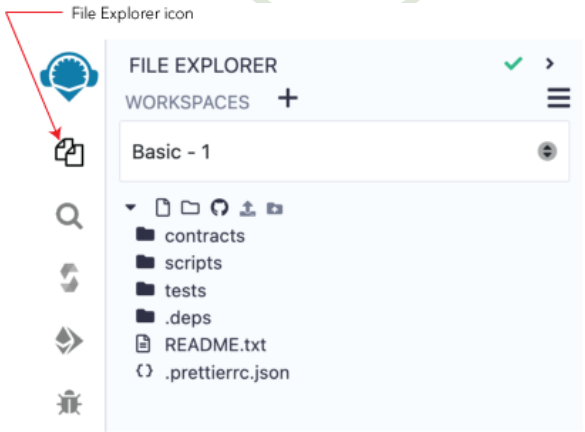


This the home page of the remix online IDE. Left side it has all the necessary folders and navigation icons. At the bottom of the page it has the terminal to see the output and verify the transactions.

The default folder structure consists of four folders namely, contracts, scripts, tests, and artifacts. It also consists of a README.txt file and a prettierc.json file.

The File Explorer is for managing workspaces and files. This plugin also contains many shortcuts and commands. For a quick tour, right-click on a file to get a pop-up menu and also check the hamburger menu at the top right of the plugin.

To find the File Explorer module - click the File Explorer icon.



Editor

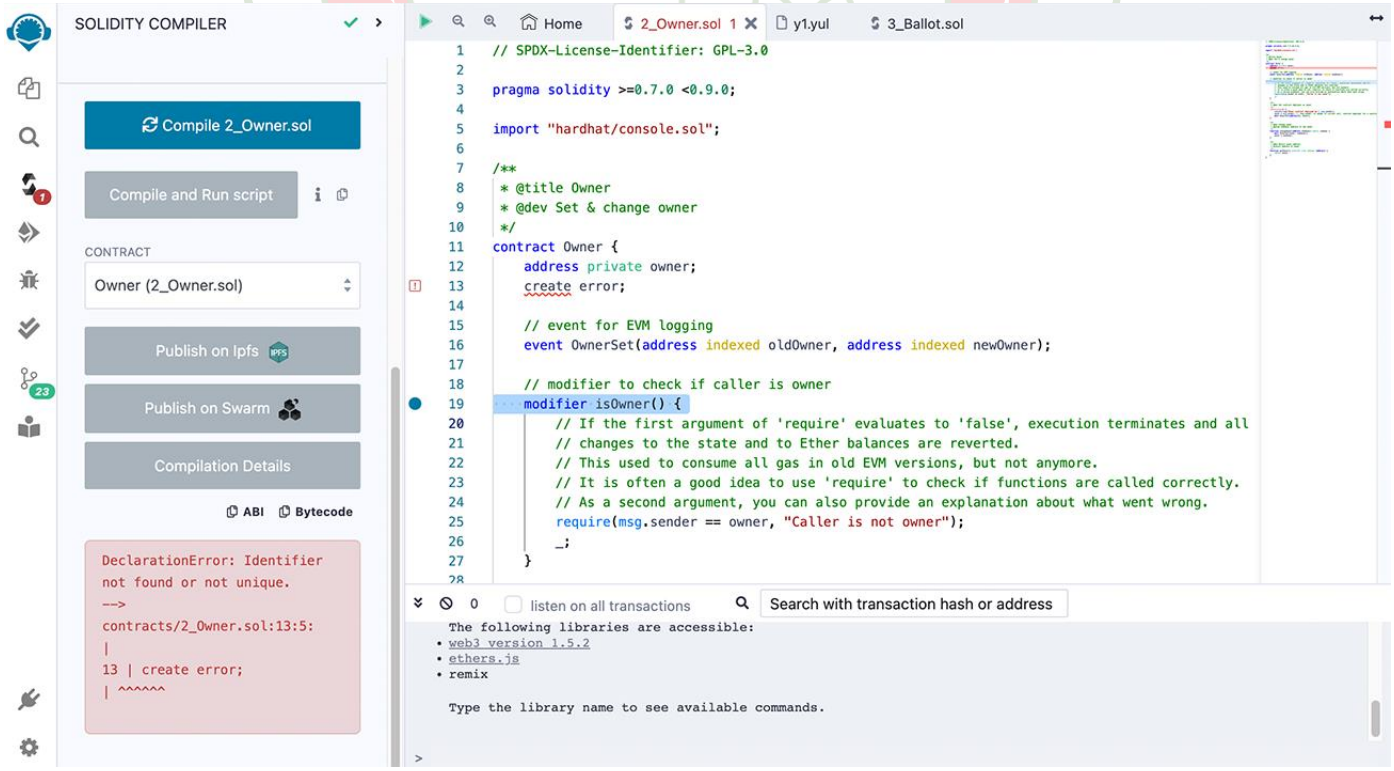
Remix uses the Monaco editor. This is the same editor used by VSCode.

Remix saves the current file every 5 seconds.

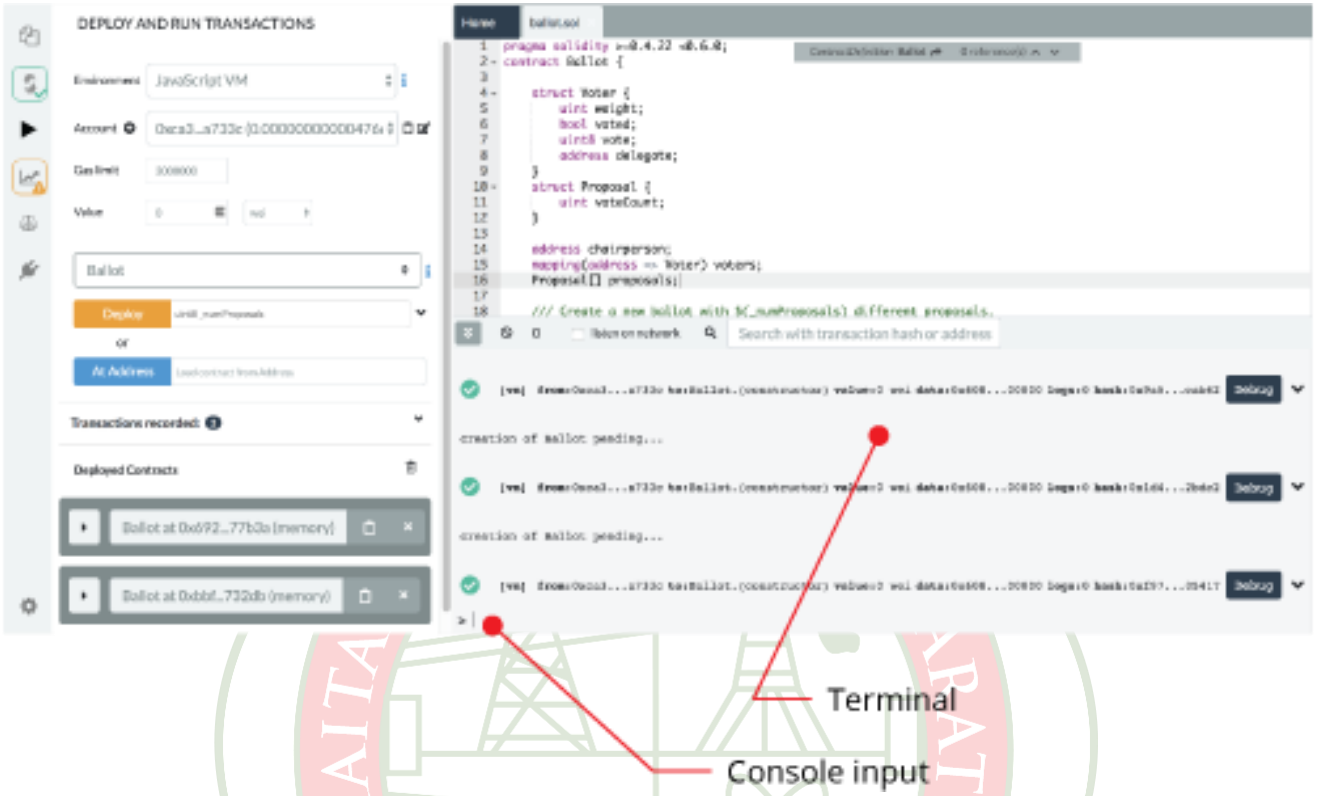
The Remix Editor will highlight keywords in Solidity, JS, and TS.

Editor displays information from other plugins

- The main purpose of the Editor is, of course, to edit code. But it also works with other plugins, notably, the Solidity Compiler and the Debugger.
- The Solidity Compiler will display warnings and errors in the Editor's gutter at the problematic line.
- Breakpoints for the Debugger are input in the Editor's gutter.
- When stepping through code in the Debugger, the relevant code will be highlighted in the Editor.



Terminal



Features, available in the terminal:

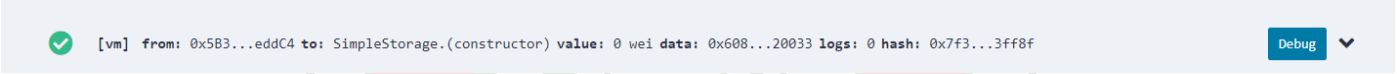
- It integrates a JavaScript interpreter and the web3 object. It enables the execution of the JavaScript script which interacts with the current context. (note that web3 is only available if the web provider or injected provider mode is selected).
- It displays important actions made while interacting with the Remix IDE (i.e. sending a new transaction).
- It displays transactions that are mined in the current context. You can choose to display all transactions or only transactions that refers to the contracts Remix knows (e.g transaction created from the Remix IDE).
- It allows searching for the data and clearing the logs from the terminal.

Write a program in solidity to assign values to a variable and display the same.

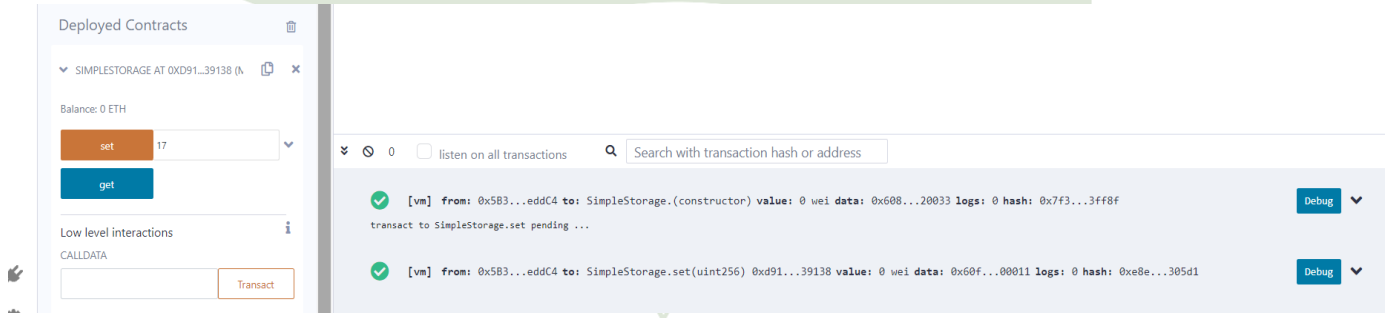
PROGRAM:

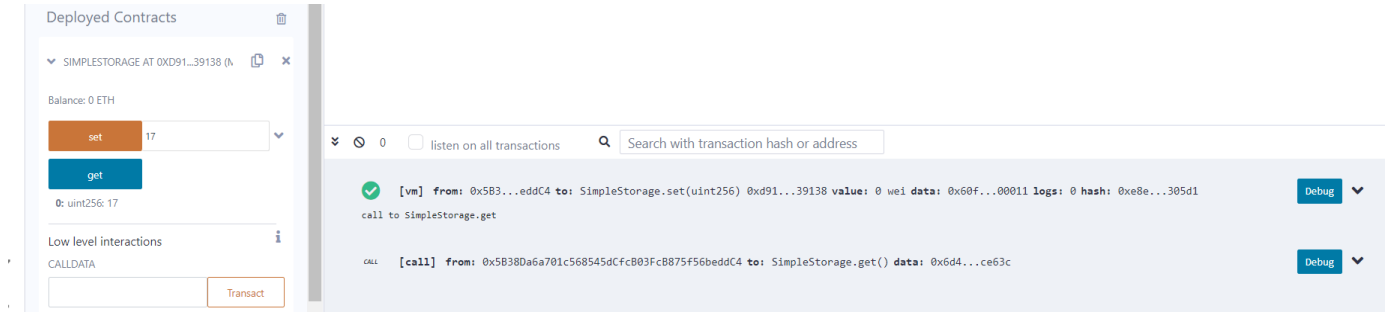
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract SimpleStorage{
    uint storedData;
    function set(uint x) public {
        storedData = x;
    }
    function get() public view returns (uint){
        return storedData;
    }
}
```

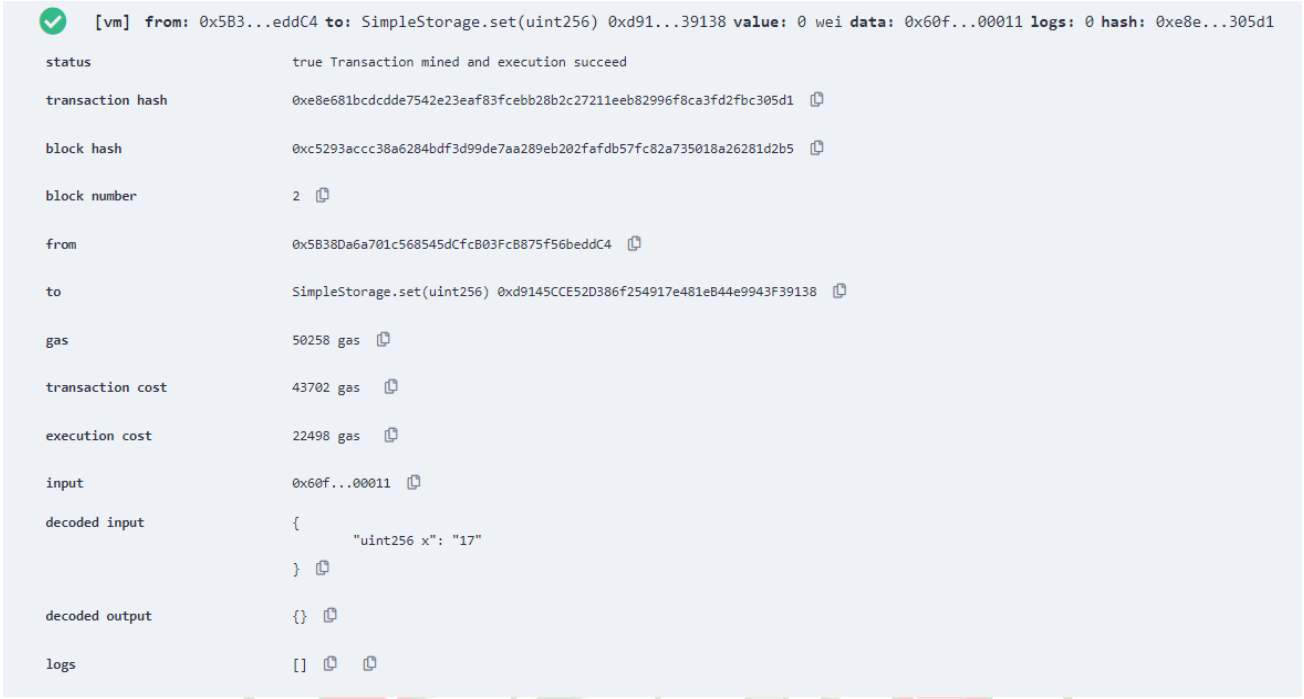
After clicking on the Deploy button the following will be displayed on the terminal if there are no errors in the code.



OUTPUT:







The screenshot shows the status of transaction and transaction hash along with from and to where the transaction has happened in the network, total gas fee, decode input

AIM : To write a program understanding the different data types in solidity

DESCRIPTION :

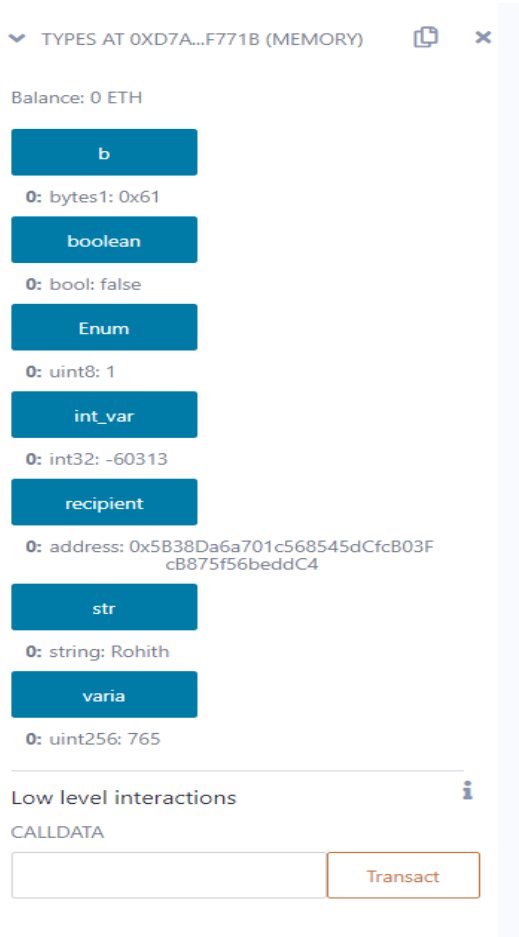
Solidity, the programming language for Ethereum smart contracts, supports various data types to represent and manipulate different kinds of data. These data types include bool, integers(int, uint), address, string, bytes, enum, bytesX etc, Data types in Solidity are used for a variety of purposes within smart contracts. They allow you to define how data should be stored and interpreted in your contract's storage and memory. These data types serve various purposes within Solidity smart contracts, such as storing data, defining custom data structures, managing addresses, handling bytes, and more. Choosing the appropriate data type depends on the nature of the data and the requirements of your contract.

PROCEDURE :

Code :

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.22 <0.9.0;
/// @title A contract for demonstrate Value types
/// @author Jitendra Kumar
/// @notice For now, this contract just show how Value types works in solidity
contract Types {
    // Initializing Bool variable
    bool public boolean = false;

    // Initializing Integer variable
    int32 public int_var = -60313;
    uint public varia = 765;
    // Initializing String variable
    string public str = "Rohith";
    bytes1 public b = "a";
    address public recipient = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
    // Defining an enumerator
    enum my_enum{ geeks_, _for, _geeks, manjusha }
    // Defining a function to return
    // values stored in an enumerator
    function Enum() public pure returns(my_enum) {
        return my_enum._for;
    }
}
```

OUTPUT :



Different variables with various data types were defined which when deployed displays the value of each variable in the Ethereum network.

Calling function :

```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Types.str() data: 0xc15...bae84

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                      Types.str() 0xD7ACd2a9FD159E69Bb102A1ca21C9a3e3A5F771B

execution cost          3445 gas (Cost only applies when called by a contract)

input                   0xc15...bae84

decoded input           {}

decoded output          {
                            "0": "string: Rohith"
                        }

logs                    []

call to Types.varia
```

Calling each function would give you information about each execution, from and to addresses, execution cost and decoded output.

2.AIM: To understand different keywords in solidity programming.

DESCRIPTION:

Solidity, the programming language for Ethereum smart contracts, has several keywords and reserved words that serve specific purposes within the language. Here are some of the most commonly used keywords in Solidity:

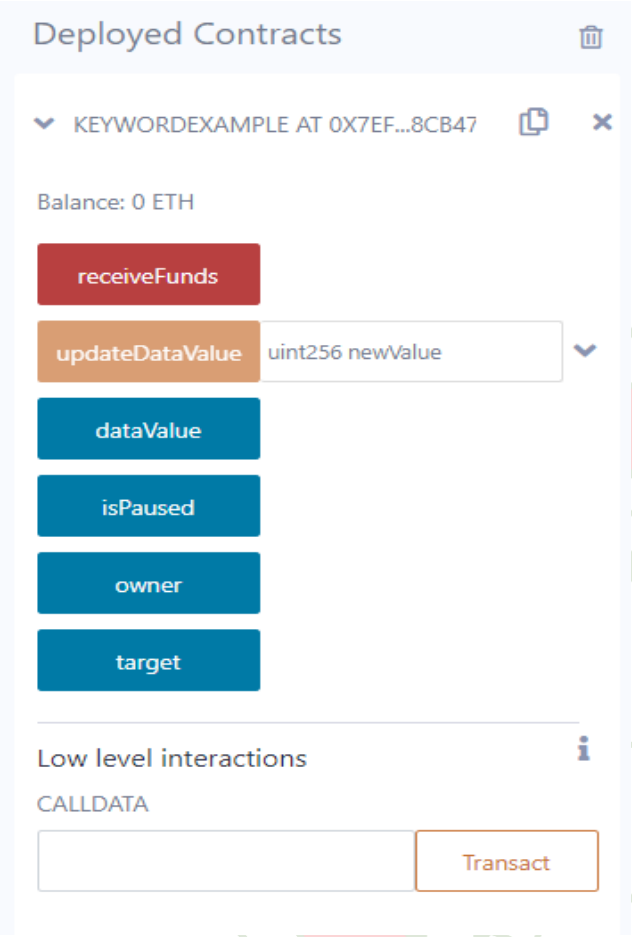
- contract: Defines a smart contract.
- function: Declares a function within a contract.
- constructor: Specifies the constructor function, which is executed only once during contract deployment.
- modifier: Defines a custom function modifier that can be used to add conditions to functions.
- event: Declares an event that allows contracts to log important information that can be monitored by external applications.
- mapping: Declares a mapping data structure, used for key-value storage.
- enum: Defines an enumeration, a user-defined data type with a finite set of values.
- struct: Declares a custom data structure with multiple fields.
- return: Specifies the value to be returned from a function.
- storage: Indicates that a variable is stored in contract storage.
- memory: Indicates that a variable is stored in memory, which is temporary and cleared after the function execution.
- calldata: Indicates that a variable refers to the transaction input data.
- payable: Used to indicate that a function can receive Ether.
- address: Specifies the data type for Ethereum addresses.

CODE :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract KeywordExample {
    // State variables
    address public owner;
    address public target;
    uint256 public dataValue;
    bool public isPaused;
    // Event
```

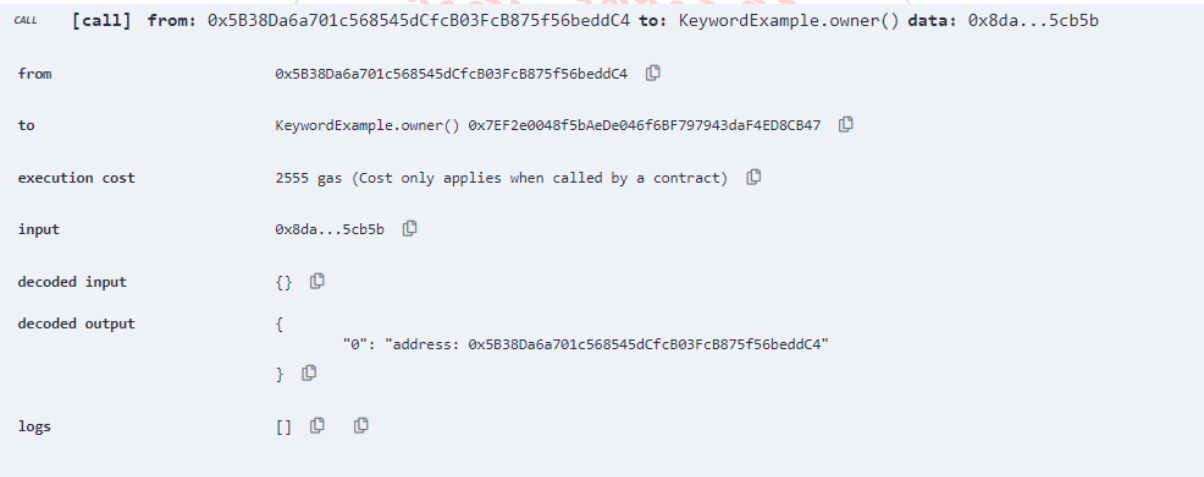
```
    event LogEvent(address indexed sender, uint256 value);
    // Enum
    enum State { Inactive, Active }
    // Struct
    struct Person {
        string name;
        uint256 age;
    }
    // Constructor
    constructor() {
        owner = msg.sender;
        dataValue = 0;
        isPaused = false;
    }
    // Modifier
    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can call this function");
        _;
    }
    // Function
    function updateDataValue(uint256 newValue) public onlyOwner {
        dataValue = newValue;
        emit LogEvent(msg.sender, newValue);
    }

    // Function that accepts Ether (payable)
    function receiveFunds() public payable {
        require(msg.value > 0, "Value must be greater than 0");
    }

    // Fallback function
    receive() external payable {
        // This function is called when the contract receives Ether
    }
}
```

OUTPUT :



updateDataVariable() sets the number to a variable and is used in different functions which when deployed displays the results.

Calling function :



It contains the transaction details like from ad to addresses decode output, decode input, execution cost, gas fee of the contract.

3.

AIM: To write a program to perform various arithmetic operations, logical operations, relational operations, bitwise operations, conditional operations.

DESCRIPTION:

Solidity is an object-oriented programming language for implementing smart contracts on various blockchain platforms, most notably, Ethereum. Solidity is statically typed, supports inheritance, libraries, and complex user-defined types, among other features. Solidity, the programming language for Ethereum smart contracts, supports a wide range of operators for performing various operations on data.

Solidity supports the following operations:

Arithmetic operators -

- Addition
- Subtraction
- Multiplication
- Division
- Modulus
- Increment
- Decrement

Logical operators -

- && - returns true only if both operands are true
- || - returns true if at least one operand is true
- ! – returns the negation of the operand

Relational operators –

- == - checks if two operands are equal
- != - checks if two operands are not equal
- > - checks if operand on the left side is greater than other operand
- < - checks if operand on the left side is less than other operand
- >= - checks if operand on the left side is greater than or equal to other operand

- <= - checks if operand on the left side is less than or equal to other operand

Bitwise operators –

- & - returns bitwise AND operation
- | - returns bitwise OR operation
- ^ - returns bitwise XOR operation
- ~ - returns bitwise NOT operation

Conditional operations –

- ? : - it allows you to express conditional statements concisely
- If-else statements - used for more complex conditional logic

PROCEDURE:

1. Arithmetic Operations -

CODE :

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;
contract SimpleStorage
{
    uint storedData;
    uint firstno;
    uint secondno;
    function setfirst(uint x) public{
        firstno=x;
    }
    function setsecond(uint x) public{
        secondno=x;
    }
    function add() public view returns (uint){
        return firstno+secondno;
    }
    function sub() public view returns (uint){
        return firstno-secondno;
    }
    function multiply() public view returns (uint){
        return firstno*secondno;
    }
    function div() public view returns (uint){
        return firstno/secondno;
    }
    function mod() public view returns (uint){
        return firstno%secondno;
```

```
    }
    function incre() public returns(uint){
        return ++firstno;
}
    function decre() public returns(uint){
        return --firstno;
}
}
```

OUTPUT :

The two set functions are made to set two variables which were then used for arithmetic operations – addition, subtraction, division, multiplication, division, modulus and two separate functions incre() and decre() were made to take input from user and increment, decrement numbers respectively.

Calling each function :

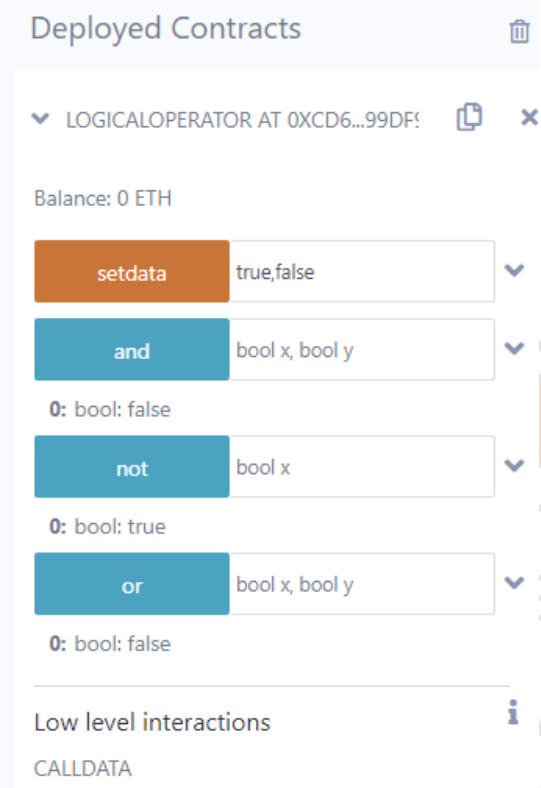
For example when you call multiply function, it would give you information about each execution, from and to addresses, execution cost and decoded output. Similarily, calling other functions displays their respective information.

2. Logical operations –

CODE -

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;
// Creating a contract
contract logicalOperator{

    // Defining function to demonstrate
    // Logical operator
    bool a;
    bool b;
    function setdata(bool x,bool y) public{
        a=x;
        b=y;
    }
    function and(bool x, bool y) pure public returns(bool){
        return x&&y;
    }
    function or(bool x, bool y) pure public returns(bool){
        return x||y;
    }
    function not(bool x) pure public returns(bool){
        return !x;
    }
}
```

OUTPUT –



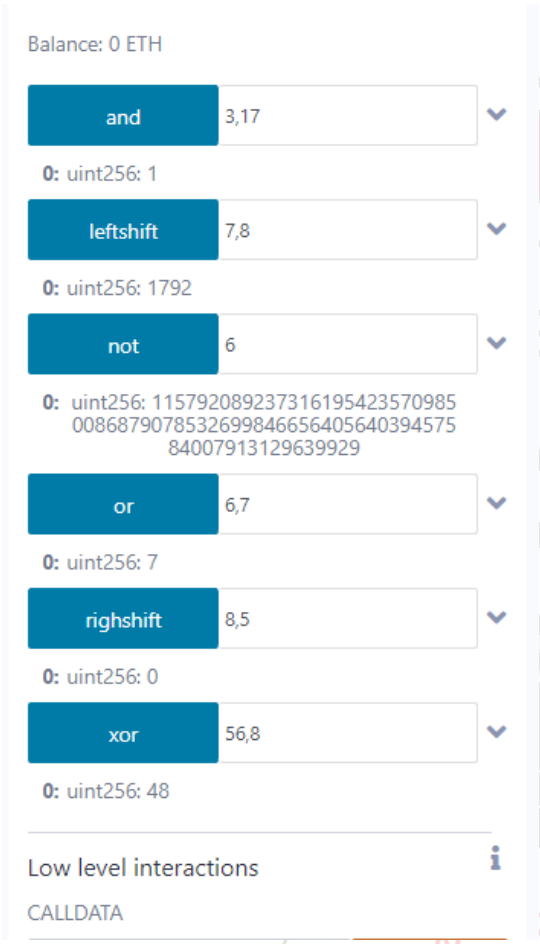
The setdata sets two boolean values to two variables and performs AND, OR, NOT operations for the operands and the results are displayed in the deployed contracts.

3. Bitwise operations –

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;
// Creating a contract
contract logicalOperator{
    function and(uint a,uint b) pure public returns(uint){
        return a&b;
    }
    function or(uint x,uint y) pure public returns(uint){
        return x|y;
    }
    function xor(uint a,uint b) pure public returns(uint){
        return a^b;
    }
    function not(uint a) pure public returns(uint){
        return ~a;
    }
    function leftshift(uint a,uint b) pure public returns(uint){
```

```
        return a<<b;
    }
    function righshift(uint a,uint b) pure public returns(uint){
        return a>>b;
    }
}
```

OUTPUT :



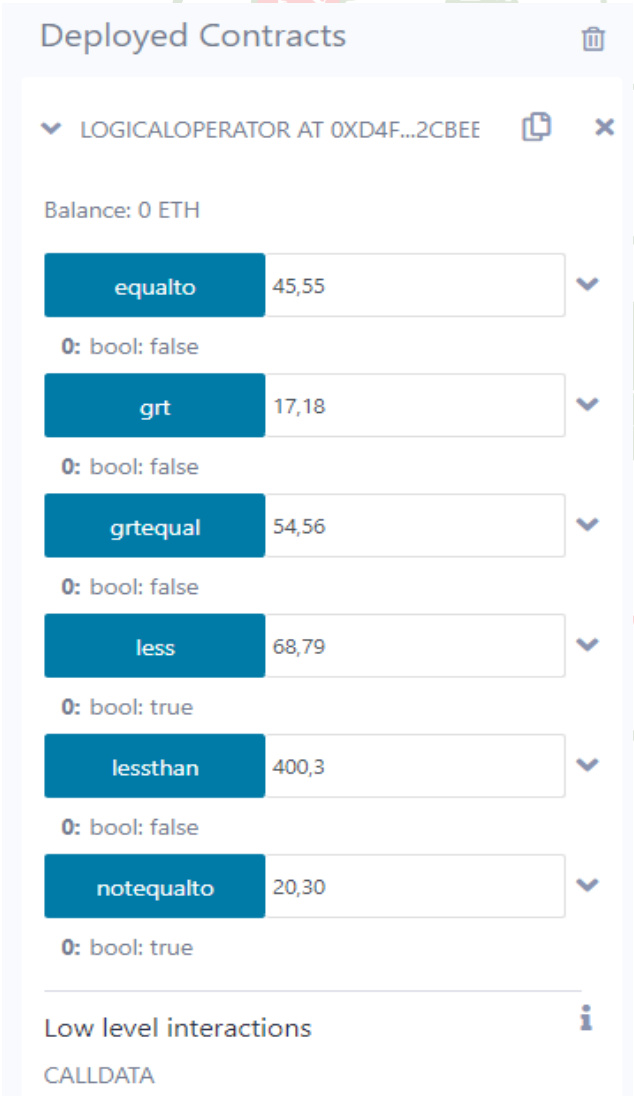
Each function takes two parameters and performs bitwise and, or, not, leftshift, rightshift operations on the operands and the results are displayed after deploying it.

4. Relational operations -

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;
// Creating a contract
contract logicalOperator{
    function equalto(uint a,uint b) pure public returns(bool){
        return a==b;
    }
```

```
    function notequalto(uint x,uint y) pure public returns(bool){
        return x!=y;
    }
    function grt(uint a,uint b) pure public returns(bool){
        return a>b;
    }
    function less(uint a,uint b) pure public returns(bool){
        return a<b;
    }
    function grtequal(uint a,uint b) pure public returns(bool){
        return a>=b;
    }
    function lessthan(uint a,uint b) pure public returns(bool){
        return a<=b;
    }
}
```

OUTPUT :

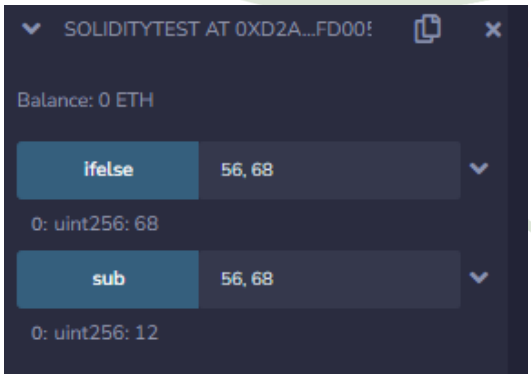


Each function takes two parameters and performs relational operations on the operands which checks the condition between operands and the returns boolean values which are displayed after deploying it.

5. Conditional operations -

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;

// Creating a contract
contract SolidityTest{
    // Defining function to demonstrate
    // conditional operator
    function sub(uint a, uint b) pure public returns(uint){
    uint result = (a > b? a-b : b-a);
    return result;
    }
    function ifelse(uint a, uint b) pure public returns(uint){
        if(a>b){
            return a;
        }
        else{
            return b;
        }
    }
}
```


OUTPUT :



There are two conditional operators - ifelse statements and ternary operators which returns the value based on the given condition and displayed in the deployed contracts.

1. AIM: Write a program to demonstrate state variable, local variable and global variable.

DESCRIPTION:

Solidity supports three types of variables.

- **State Variables** − Variables whose values are permanently stored in a contract storage.
- **Local Variables** − Variables whose values are present till function is executing.
- **Global Variables** − Special variables exists in the global namespace used to get information about the blockchain.

Variable scope is an essential concept in Solidity, the programming language for Ethereum smart contracts. Solidity has various types of variables with different scopes, such as local, state, and global variables.

Local Variable
Local variables are declared within a function and are only accessible within that function. Their scope is limited, and their lifetime ends when the function execution is completed.

State Variable
State variables are declared at the contract level and represent the contract's state on the blockchain. They are stored on the Ethereum network and are accessible within the entire contract.

Global Variable
Global variables are special variables provided by the Solidity language. They are available throughout the contract and provide information about the blockchain, transaction, and execution context.

Examples of Global Variables:

- block.timestamp (current block timestamp)
- msg.sender (address of the sender of the current function call)
- msg.value (amount of ether sent in the current function call)

Variable Scope

There are three types of variable scopes in Solidity:

1. Public

Public variables are accessible from within the contract and can be accessed from external contracts as well. Solidity automatically generates a getter function for public state variables.

2. Private

Private variables are only accessible within the contract they are defined in. They are not accessible from derived contracts or external contracts.

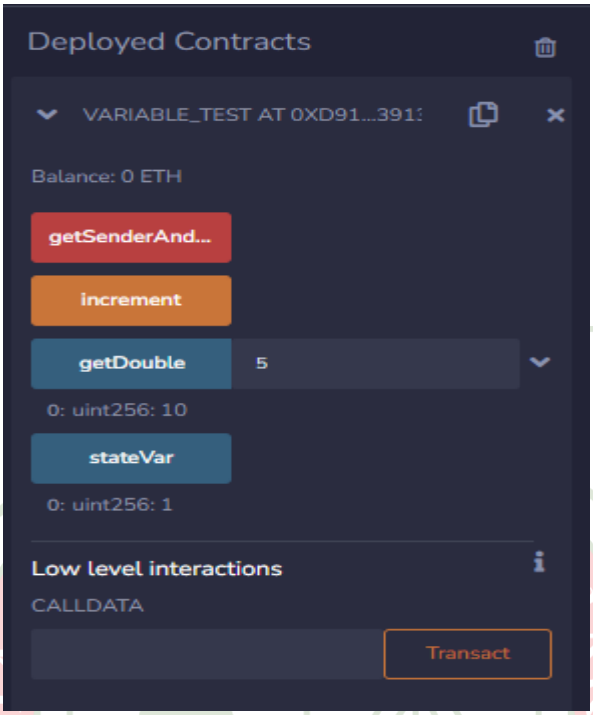
3. Internal

Internal variables are accessible within the contract they are defined in and derived from contracts. They are not accessible from external contracts.

PROGRAM:

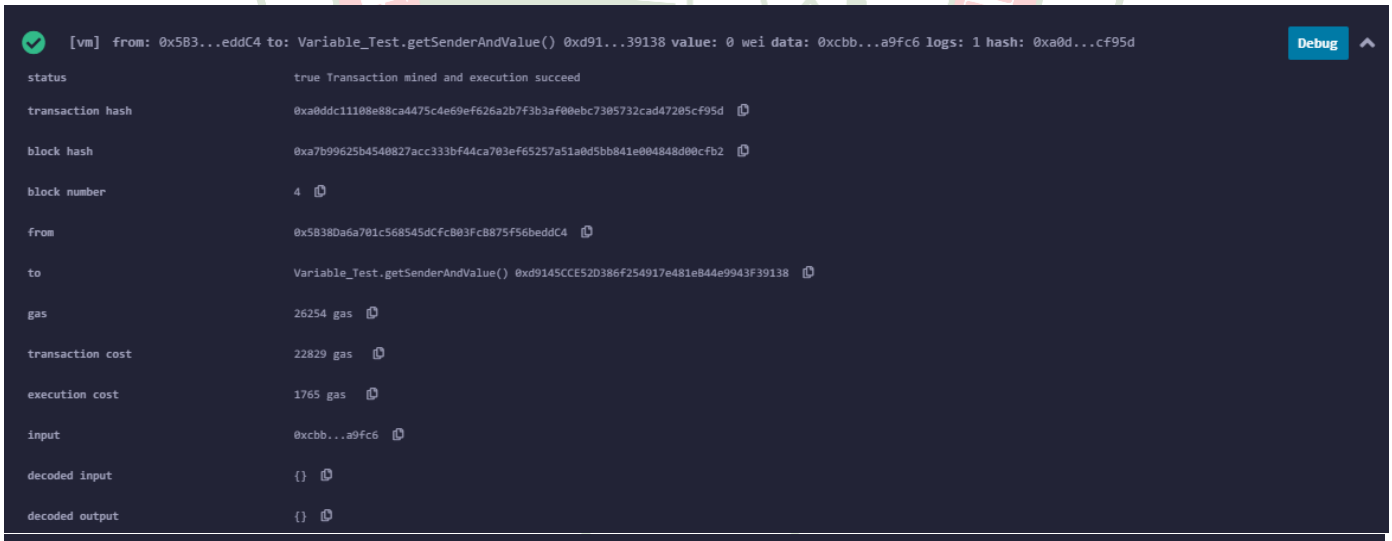
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract Variable_Test {

  uint public stateVar = 0;
  event SenderAndValue(address sender, uint value);

  function getDouble(uint value) public pure returns (uint)
  {
    uint localVar = value * 2;
    return localVar;
  }
  function increment() public
  {
    stateVar += 1;
  }

  //demonstarte global variable
  function getSenderAndValue() public payable
  {
    address sender = msg.sender;
    uint value = msg.value;
    emit SenderAndValue(sender, value);
  }
}
```

DEPLOYMENT:



OUTPUT:

Global variable



State varibale

2. AIM: Write a program to demonstrate all the special variables.

DESCRIPTION:

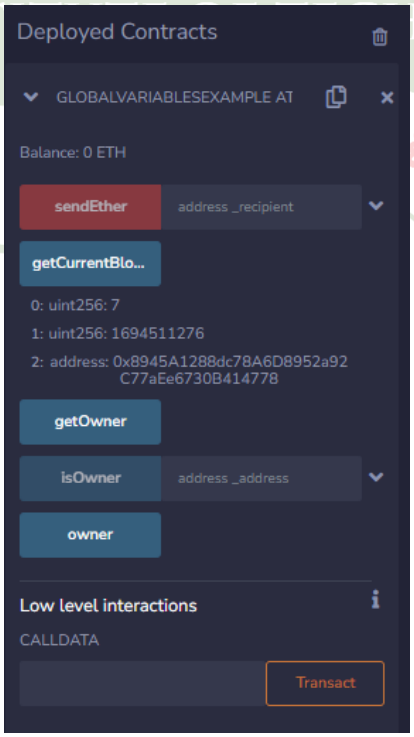
There exist special variables and functions in solidity which exist in the global namespace and are mainly used to provide information about the blockchain or utility functions. They are of two types:

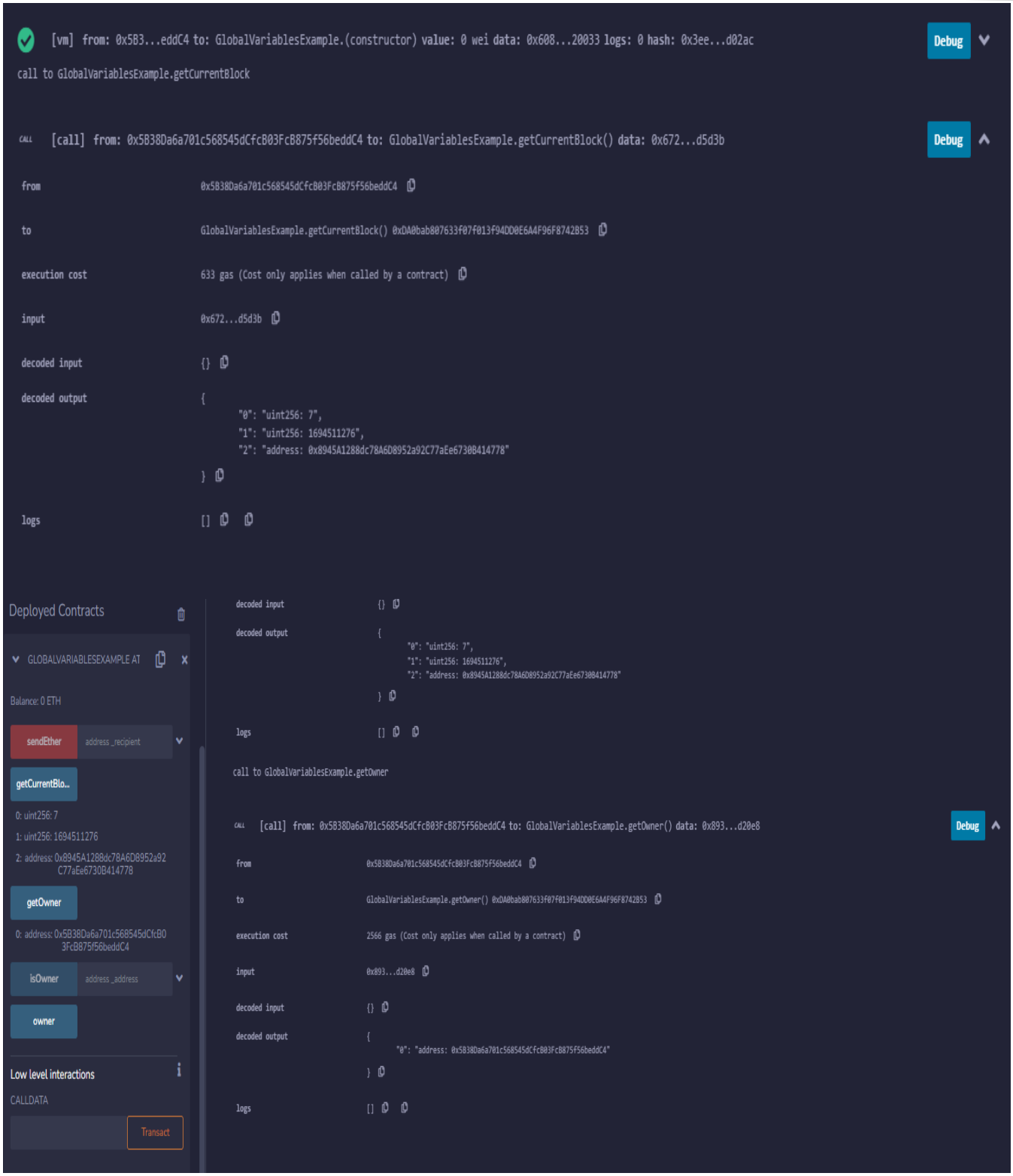
Block	Transaction Properties
block.coinbase (address payable)	Current block miner's address
block.difficulty (uint)	Current block difficulty
msg.value (uint)	Number of wei sent with the message
block.number (uint):	Current block number
blockhash(uint blockNumber) returns (bytes32)	Gives hash of the given block and will only work for the 256 most recent block due to the reason of scalability.
block.timestamp:	Current block timestamp as seconds since unix epoch
gasleft() returns (uint256):	Remaining gas
msg.sender (address payable)	Sender of the message (current call)
msg.sig (bytes4)	First four bytes of the calldata (i.e. function identifier)
now (uint)	Current block timestamp (alias for block.timestamp)
tx.gasprice (uint)	Gas price of the transaction
block.gaslimit (uint)	Current block gaslimit
tx.origin (address payable)	Sender of the transaction (full call chain)

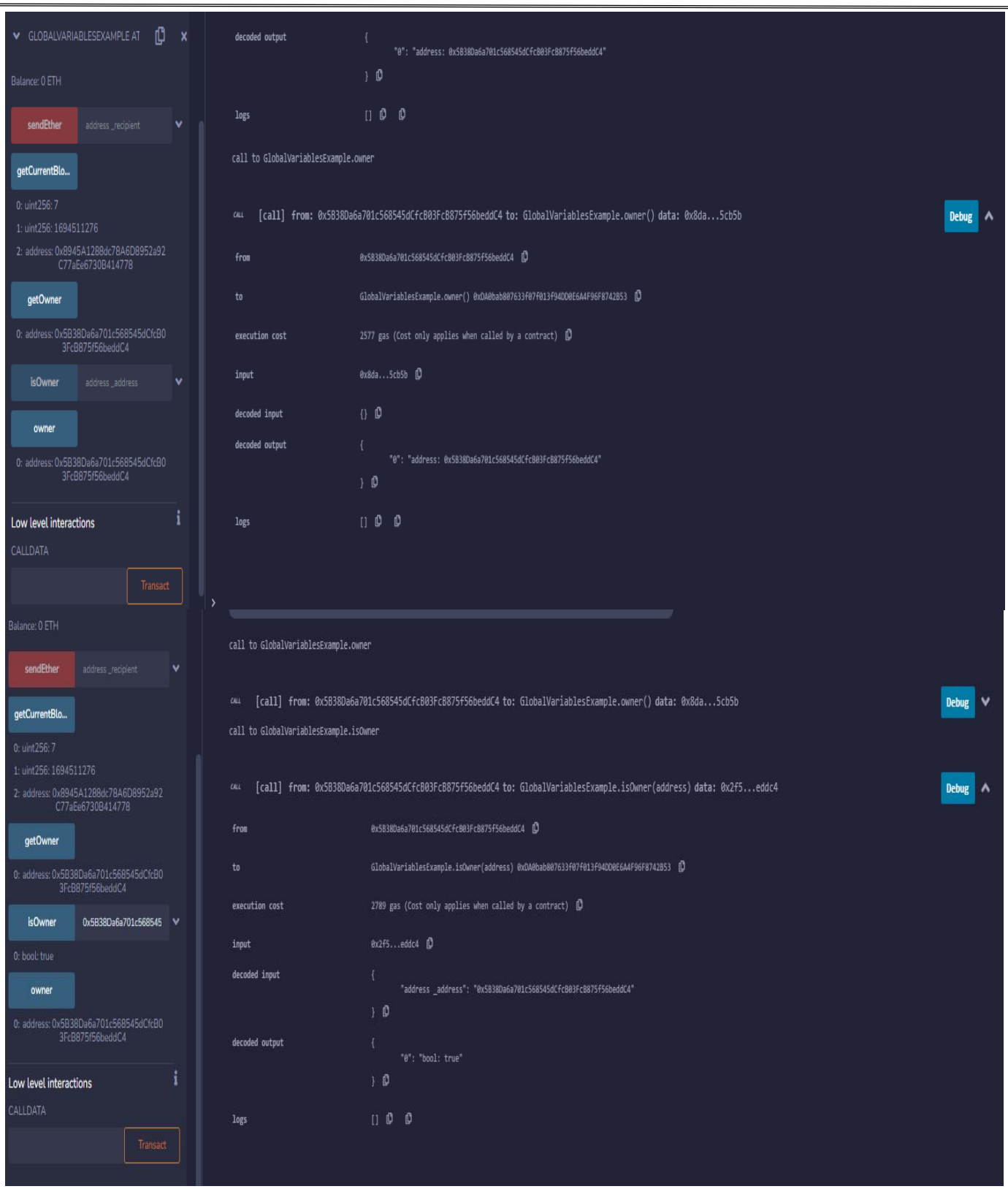
PROGRAM:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract GlobalVariablesExample {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function getOwner() public view returns (address) {
        return owner;
    }

    function isOwner(address _address) public view returns (bool) {
        return _address == owner;
    }

    function sendEther(address payable _recipient) public payable {
        require(msg.sender == owner, "Only the contract owner can send
ether.");
        _recipient.transfer(msg.value);
    }

    function getCurrentBlock() public view returns
(uint256,uint256,address){
        return (block.number, block.timestamp, block.coinbase);
    }
}
```

OUTPUT:




[vm] from: 0x5B3...eddC4 to: GlobalVariablesExample.(constructor) value: 0 wei data: 0x608...20033 logs: 0 hash: 0x3ee...d02ac
Debug
call to GlobalVariablesExample.getCurrentBlock
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: GlobalVariablesExample.getCurrentBlock() data: 0x672...d5d3b
Debug
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to GlobalVariablesExample.getCurrentBlock() 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
execution cost 633 gas (Cost only applies when called by a contract)
input 0x672...d5d3b
decoded input {}
decoded output { "0": "uint256: 7", "1": "uint256: 1694511276", "2": "address: 0x8945A1288dc78A6D8952a92C77aEe6730B414778" }
logs []
Deployed Contracts
GLOBALVARIABLESEXAMPLE AT
Balance: 0 ETH
sendEther
address _recipient
getCurrentBlo...
0: uint256: 7
1: uint256: 1694511276
2: address: 0x8945A1288dc78A6D8952a92C77aEe6730B414778
getOwner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
isOwner
address _address
owner
Low level interactions
CALLDATA
Transact
call to GlobalVariablesExample.getOwner
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: GlobalVariablesExample.getOwner() data: 0x893...d20e8
Debug
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to GlobalVariablesExample.getOwner() 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
execution cost 2566 gas (Cost only applies when called by a contract)
input 0x893...d20e8
decoded input {}
decoded output { "0": "address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4" }
logs []


GLOBALVARIABLESEXAMPLE AT
Balance: 0 ETH
sendEther
address _recipient
getCurrentBlo...
0: uint256: 7
1: uint256: 1694511276
2: address: 0x8945A1288dc78A6D8952a92C77aEe6730B414778
getOwner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
isOwner
address _address
owner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
Low level interactions
CALLDATA
Transact
decoded output
{ "0": "address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4" }
logs
[]
call to GlobalVariablesExample.owner
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: GlobalVariablesExample.owner() data: 0x8da...5cb5b
Debug
from
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to
GlobalVariablesExample.owner() 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
execution cost
2577 gas (Cost only applies when called by a contract)
input
0x8da...5cb5b
decoded input
{}
decoded output
{ "0": "address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4" }
logs
[]
call to GlobalVariablesExample.owner
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: GlobalVariablesExample.owner() data: 0x8da...5cb5b
Debug
call to GlobalVariablesExample.isOwner
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: GlobalVariablesExample.isOwner(address) data: 0x2f5...eddc4
Debug
from
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to
GlobalVariablesExample.isOwner(address) 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
execution cost
2789 gas (Cost only applies when called by a contract)
input
0x2f5...eddc4
decoded input
{ "address _address": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4" }
decoded output
{ "0": "bool: true" }
logs
[]

3. AIM: Write a program to demonstrate IF condition statement.

DESCRIPTION: If statement is a type of conditional statement. It is used to execute a certain block of code or statements only if a certain condition is true else no statement is executed.

Syntax:

```
if (condition) {
  // code is executed if condition is true
}
```

Here,

if: keyword used to initiate the conditional statement.
(condition): to check the condition it can be any expression that evaluates to a boolean value (true or false).
{ }: enclose the block of code to be executed if the condition is true.

PROGRAM:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract demo_If{
    uint i = 10;
    function greater_than_ten(uint _var) public view returns(bool){
        if(_var > i){
            return true;
        }
        return false;
    }
}
```

4. AIM: Write a program to demonstrate IF-ELSE condition statement.

DESCRIPTION:

If-else statement is a type of conditional statement that allows the program to execute one block of code if a certain condition is true and an else block of code is executed if the condition is false. It contains two blocks of code and depending on the condition any one of the blocks is getting executed.

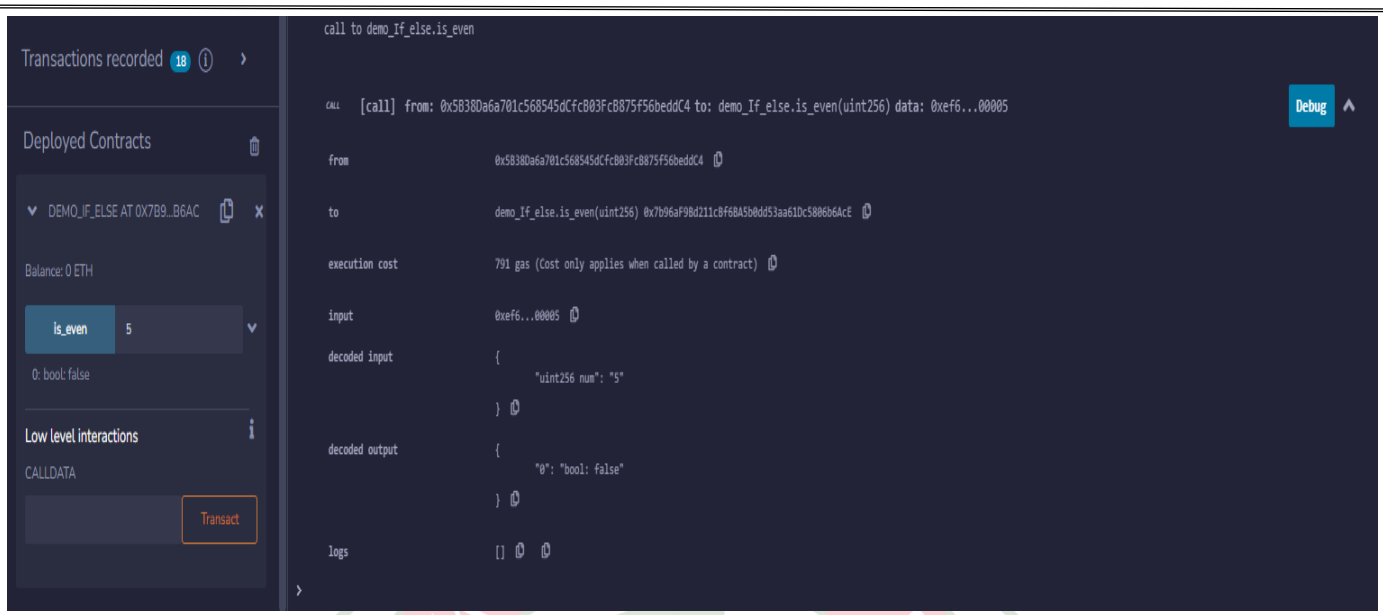
Syntax:

```
if (condition) {
  //  this block of code is executed if  condition is true
} else {
  // this block of code is executed if condition is false
}
```

Here,
if: keyword used to initiate the conditional statement.
(condition): to check the condition it can be any expression that evaluates to a boolean value (true or false).
{ }: enclose the block of code to be executed if the condition is true.
else: keyword used to execute the block of statement if condition fails.
{ }: enclose the block of code to be executed if the condition is false.

PROGRAM:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract demo_If_else{
    function is_even(uint num) public pure returns(bool){
        if(num%2==0){
            return true;
        }
        else {
            return false;
        }
    }
}
```



5 AIM: Write a program to demonstrate IF- ELSE IF -ELSE condition statement.

DESCRIPTION:

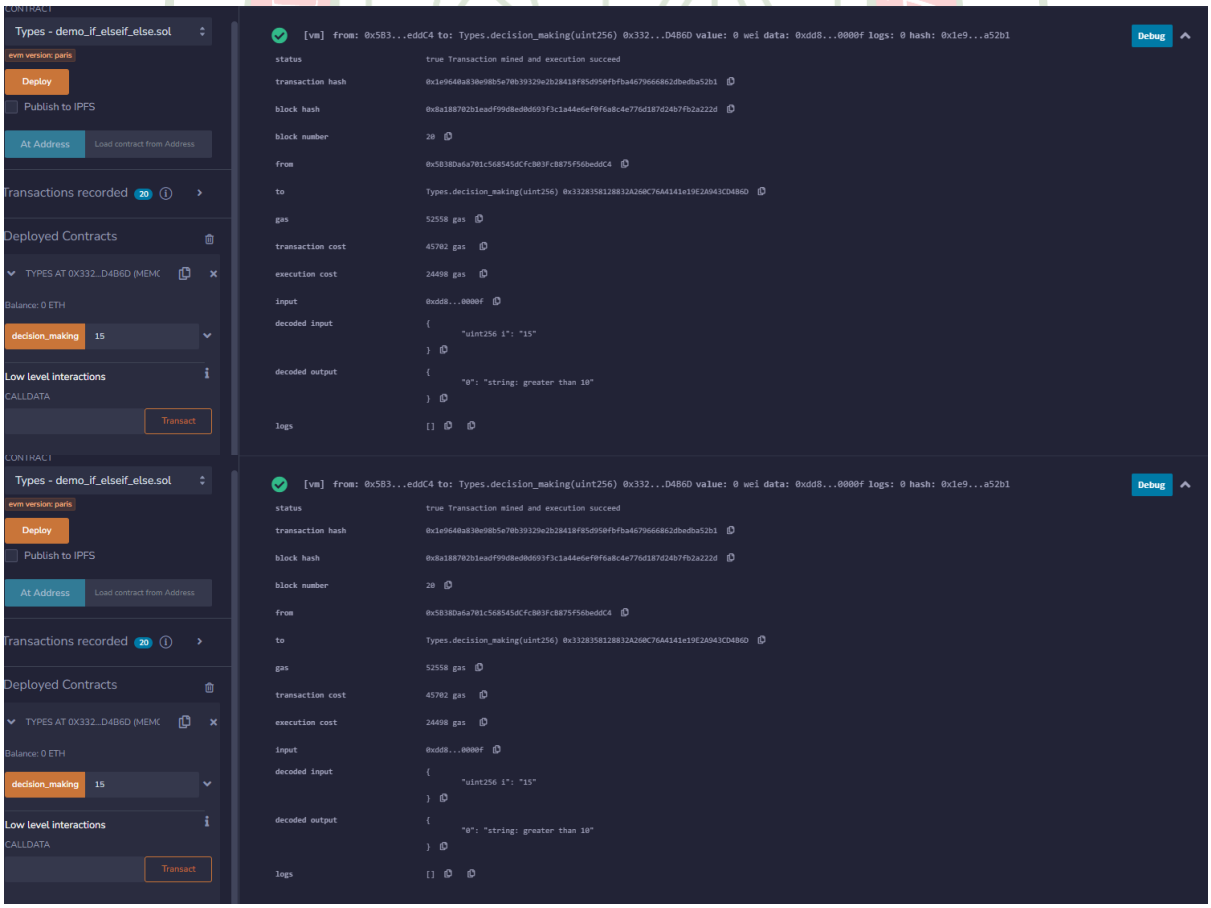
This is a modified form of if…else conditional statement which is used to make a decision among several options. The statements start execution from if statement and as the condition of any if block is true the block of code associated with it is executed and rest if are skipped, and if none of the condition is true then else block executes.

Syntax:

```
if (condition) {
  statement or block of code to be executed if the condition is True
} else if (condition 2) {
  statement or block of code to be executed if the condition of
else...if is True
} else {
  statement or block of code to be executed if none of the condition is
True
}
```

PROGRAM:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract Types {
    string result;

    function decision_making(uint i) public returns(string memory){
        if(i<10){
            result = "less than 10";
        }
        else if(i == 10){
            result = "equal to 10";
        }
        else{
            result = "greater than 10";
        }
        return result;
    }

}
```



AIM: To get started with the Solidity programming language and deploy smart contracts.

DESCRIPTION:

Smart Contract:

A Smart Contract is a computer program that has been deployed onto a blockchain. These programs automatically execute, control or document events and actions according to the terms of a contract or an agreement. The main goal of smart contracts are to eliminate trusted intermediaries and avoid accidental exceptions and mishandling.

Solidity:

Solidity is an object-oriented programming language for implementing smart contracts on various blockchain platforms, most notably Ethereum. Programs written in Solidity run on the Ethereum Virtual Machine.

Ganache, Truffle and Node.js:

Ganache provides an easy one-click solution for creating a local blockchain testnet onto which a tool such as Truffle which runs on top of node.js can be used to deploy and interact with Smart Contracts.

Procedure:

1. Download ganache for your OS.
2. Install latest version of Node.js

   For debian-based OS use the command:

   ```
   sudo snap install node
   ```

3. Install truffle using npm

   ```
   npm install truffle -g
   ```

4. Launch ganache, click on the Quickstart button and note the port number used. 5. Run the following command in your workspace folder.

   ```
   truffle init
   ```

6. Edit `truffle-config.js` and uncomment the following lines:

Replace the port with the one displayed in ganache.

7. Write your smart contracts using your favourite text editor in the `contracts` directory.

8. Create a new `.js` file in the migrations directory and add the following lines for each smart contract file in `contracts/`.

```
1 var MyContract = artifacts.require("MyContract"); 2
3 module.exports = function(deployer) {
```

```
4 // deployment steps
5 deployer.deploy(MyContract);
6 };
```

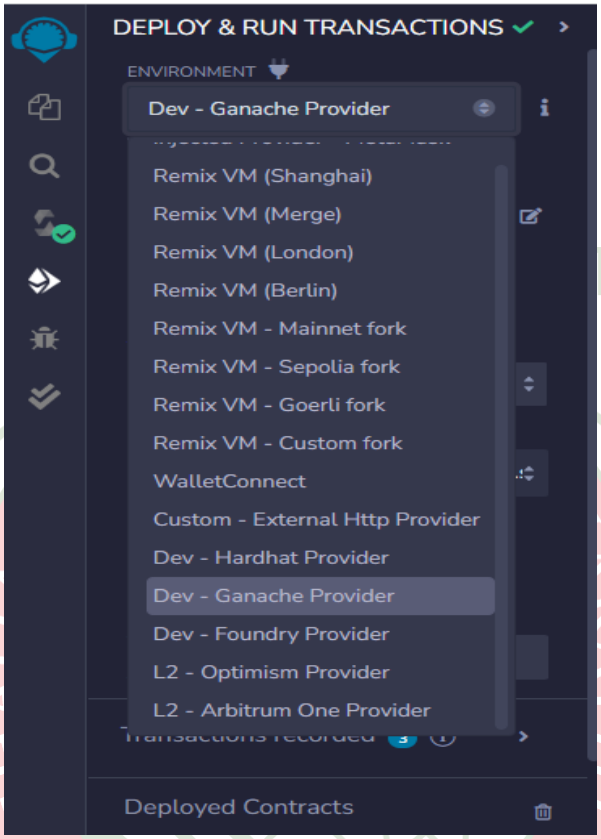
9. Deploy the contracts using "`truffle migrate`" or "`truffle deploy`" or "`truffle compile`".

10. Interact with the smart contracts by running "`truffle console`".

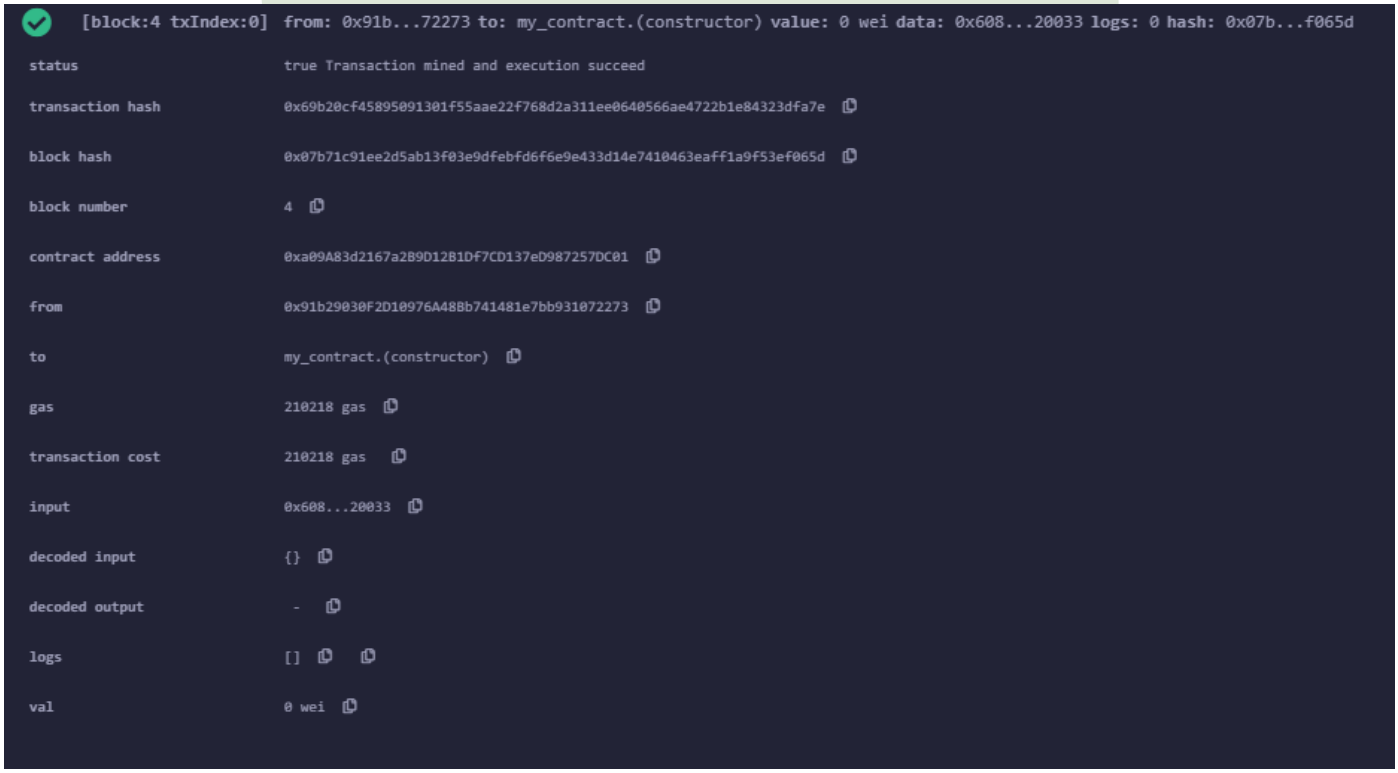
PROGRAMS AND OUTPUTS:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
// Creating a contract
contract my_contract{
    // Defining a function
    function get_output() public pure
returns (string memory){
        return ("Hi, your contract ran
successfully");
    }
    function is_even(uint num) public pure
returns(bool){
        if(num%2==0){
            return true;
        }
        else {
            return false;
        }
    }
}
```

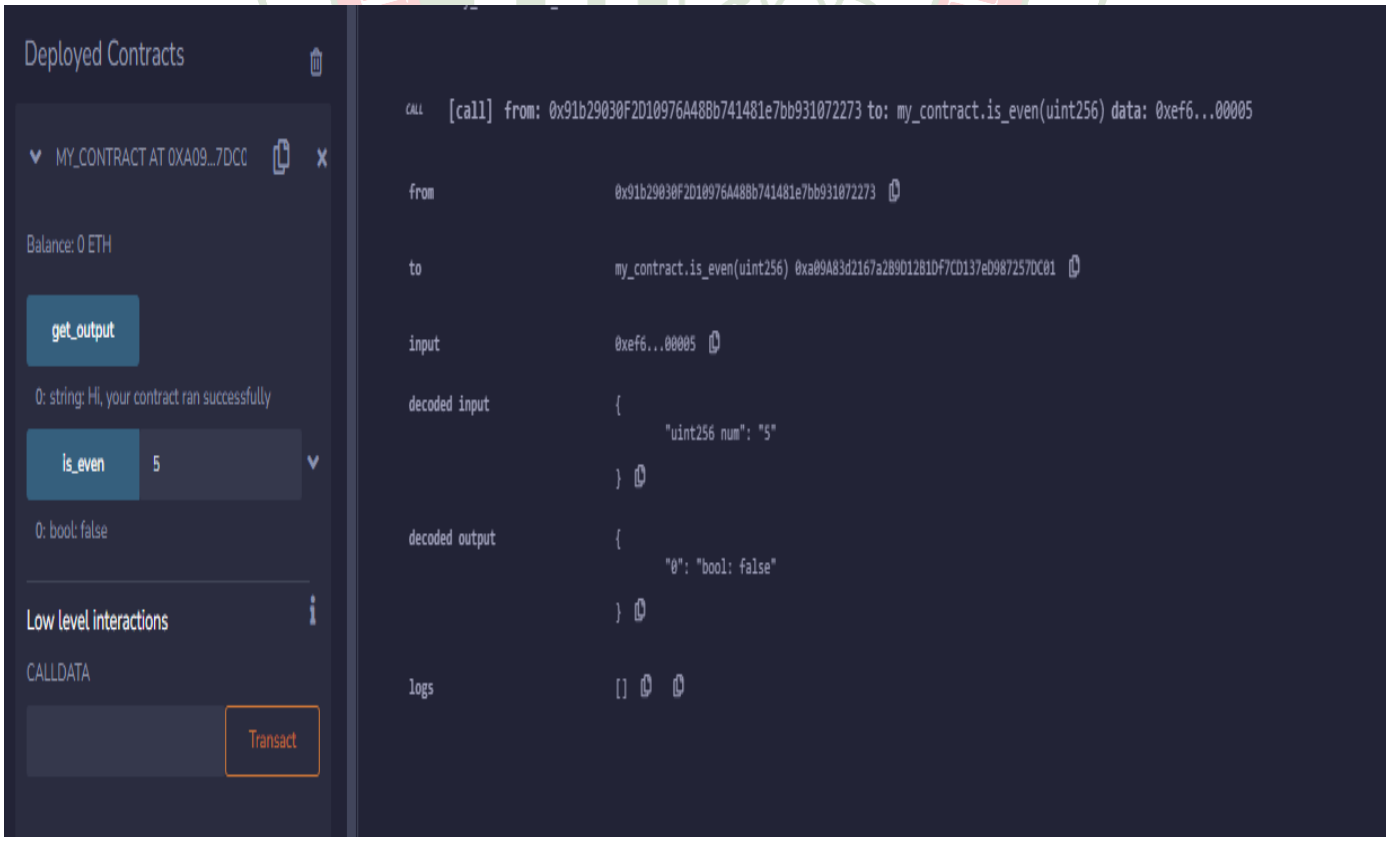
Interacting with smart contract:

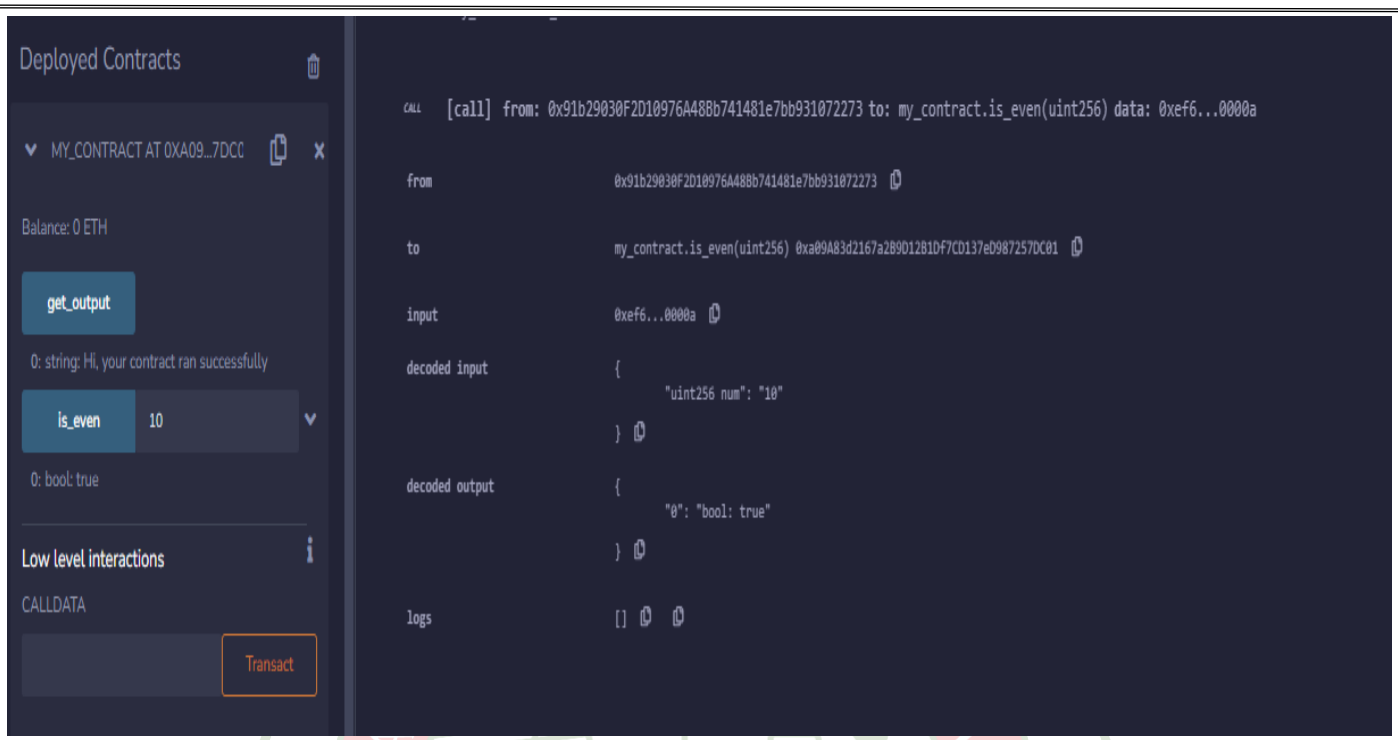


In the remix environment , after compiling the code In the deployment section select Dev-Ganache Provider in the



After deployment we can execute the smart contract and call the various member functions in the contract.





Viewing the deployed transactions on Ganache:

BLOCK 4	MINED ON 2023-09-26 14:47:35	GAS USED 210218	1 TRANSACTION
BLOCK 3	MINED ON 2023-09-26 14:41:34	GAS USED 210218	1 TRANSACTION
BLOCK 2	MINED ON 2023-09-26 14:41:22	GAS USED 210218	1 TRANSACTION
BLOCK 1	MINED ON 2023-09-26 14:32:36	GAS USED 134419	1 TRANSACTION
BLOCK 0	MINED ON 2023-09-26 14:13:56	GAS USED 0	NO TRANSACTIONS

Above is the list of blocks having the transactions that have being executed on the local blockchain. It consists of all the data related to the transactions likes, the gas fee used and the timestamp.

← BACK **BLOCK 4**

GAS USED	GAS LIMIT	MINED ON	BLOCK HASH
210218	6721975	2023-09-26 14:47:35	0×07b71c91ee2d5ab13f03e9dfebfd6f6e9e433d14e7410463eaff1a9f53ef065d

TX HASH
0×69b20cf45895091301f55aae22f768d2a311ee0640566ae4722b1e84323dfa7e — CONTRACT CREATION

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0×91b29030F2D10976A48Bb741481e7bb931072273	0×a09A83d2167a2B9D12B1Df7CD137eD987257DC01	210218	0

Above are the details in each block . It consists of various transactions executed. It consists of the gas used, gas limit, timestamp, the block hash, and the transaction's hash value.

← BACK **TX 0×69b20cf45895091301f55aae22f768d2a311ee0640566ae4722b1e84323dfa7e**

SENDER ADDRESS	CREATED CONTRACT ADDRESS	
0×91b29030F2D10976A48Bb741481e7bb931072273	0×a09A83d2167a2B9D12B1Df7CD137eD987257DC01	CONTRACT CREATION

VALUE	GAS USED	GAS PRICE	GAS LIMIT	MINED IN BLOCK
0.00 ETH	210218	3100112976	210218	4

TX DATA

0×6080604052348015610010576000080fd5b506102d9806100206000396000f3fe6080604052348015610010576000080fd5b506004361061003657600035600e01c806354f8a2f21461003b578063ef6c9d8a14610059575b600080fd5b
6100436100895656b6040516100509190610161565b6040518090390f35b6100736004803603810190610006e91906101be565b6100a9565b6040516100809190610206565b6040518090390f35b6006060405180606001604052806022
8152602001610282602291399050905656b6000806002836100b9919061025656b036100c757600190506100cc565b600090505b919050565b60000815190509190505656b60008282526020820190509291505056b60005b8381101561
010b578082015181840152602081019050610100f0565b6000848401525050505050565b6000601f19601f830116905091905056b6000610133826100d1565b61013d81856100dc565b935061014d818560208601610ed565b6101568161
0117565b840191505092915050565b6000602082019050818103600083015261017b8184610128565b905092915050565b600080fd5b6000819050919050565b61019b81610188565b81146101a657600080fd5b50565b600081359050
6101b881610192565b9291505056b6000602082840312156101d4576101d3610183565b5b60006101e2848285016101a9565b915050929150505656b6000811515905091905056b610200816101eb565b82525050565b600060208201
905061021b6000830184610f7565b92915050565b7f4e487b7100000000000000000000000000000000000000000000000000000000600052601260045260246000fd5b600061025b82610188565b9150610266836101885656b925082
610276576102756102215656b5b828206905092915050565fe48692c20796f757220636f6e7472616374207616e207375636365737366756c6c79a2646970667358221220d464489856559245672 47b75ebf2406db9e582759e28e2b6c8
4efe46d609714064736f6c63430008120033

EVENTS

TX HASH
0×69b20cf45895091301f55aae22f768d2a311ee0640566ae4722b1e84323dfa7e — CONTRACT CREATION

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0×91b29030F2D10976A48Bb741481e7bb931072273	0×a09A83d2167a2B9D12B1Df7CD137eD987257DC01	210218	0

TX HASH
0×fa056ba939fc01dd7b27c8c2ee14e38d0dc6ed3d3ac143a9ed533a76971d1270 — CONTRACT CREATION

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0×91b29030F2D10976A48Bb741481e7bb931072273	0×ec3B0994655F39F1278943A0a63C6dBE8Cf2e5Bd	210218	0

TX HASH
0×e447923b93003aad63a7b1b2c5c9ac7aa84cc3806dc794095b42378ab39978d3 — CONTRACT CREATION

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0×91b29030F2D10976A48Bb741481e7bb931072273	0×458e966090304B5b9A54B93b6172589a7EA591C5	210218	0

TX HASH
0×d78fcf838409eff97cc5c1d8b82d0df6e83b3d9cad774d702d698e53829795f8 — CONTRACT CREATION

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0×91b29030F2D10976A48Bb741481e7bb931072273	0×fC23a632E15d46DC1F44bEaC9013fb0D2FAC0560	134419	0

EXPERIMENT – 9

AIM: To understand and implement the concept of arrays and structures in solidity.

DESCRIPTION:

Arrays:

- Arrays are data structures that store the fixed collection of elements of the same data types in which each and every element has a specific location called index.
- Instead of creating numerous individual variables of the same type, we just declare one array of the required size and store the elements in the array and can be accessed using the index.
- In Solidity, an array can be of fixed size or dynamic size.
  Arrays have a continuous memory location, where the lowest index corresponds to the first element while the highest represents the last
- **1 . Fixed size array:** the size of an array is fixed (predefined) at the time of compilation
- data_type[size] array_name = <elements>;
- **2 . Dynamic size array:** dynamic means the size of an array can change
- data_type[] array_name = <elements>;

Structures:

- Structs in Solidity allows you to create more complicated data types that have multiple properties. You can define your own type by creating a **struct**.
- They are useful for grouping together related data.
- Structs can be declared outside of a contract and imported in another contract. Generally, it is used to represent a record.
- To define a structure *struct* keyword is used, which creates a new data type.

Syntax:

```
struct <structure_name> {
   <data type> variable_1;
   <data type> variable_2;
}
```

Example:

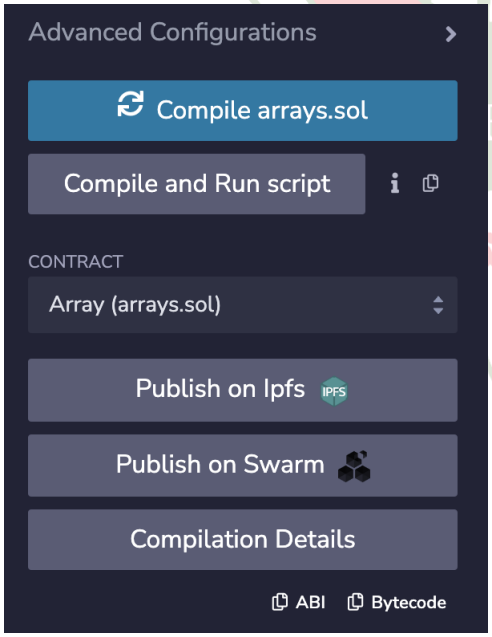
```
struct Book {
  string title;
  string author;
  uint book_id;

}
```

PROCEDURE:

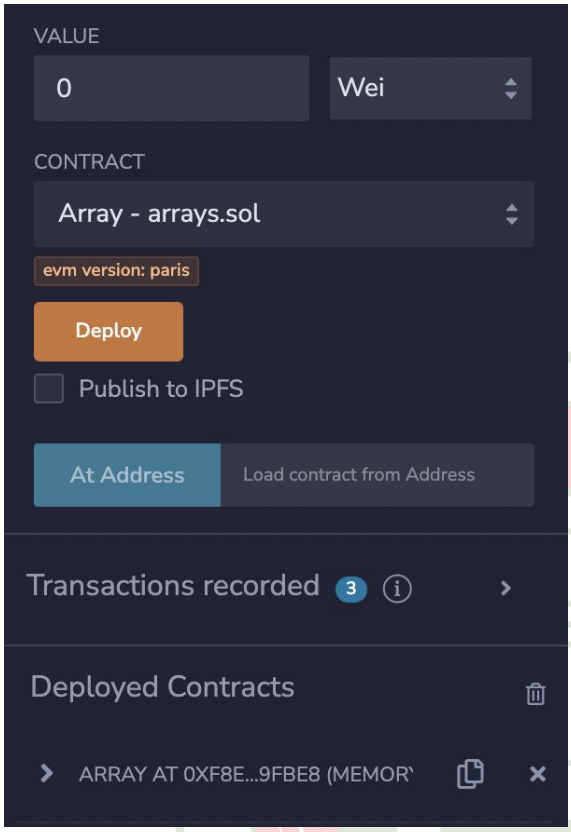
(ARRAYS)

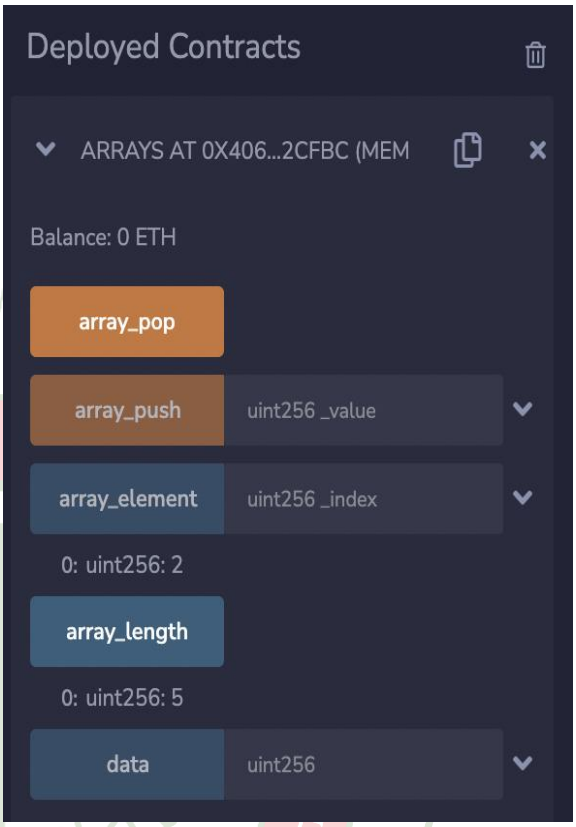
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;
// Creating a contract
contract Arrays {
    uint[] public data;

  function array_element(uint _index) public view returns (uint) {    infinite gas
        require(_index < data.length, "Index out of bounds");
        return data[_index];
        }

  function array_length(    2436 gas
  ) public view returns(uint) {
        uint x = data.length;
        return x;
    }

    function array_push(uint _value) public {    46873 gas
        data.push(_value);
    }

    function array_pop(    infinite gas
    ) public returns(uint[] memory){
        data.pop();
        return data;
    }
}
```

Compiling the smart contract :



Deploying the smart contract :





OUTPUT:

Pushing an element into array

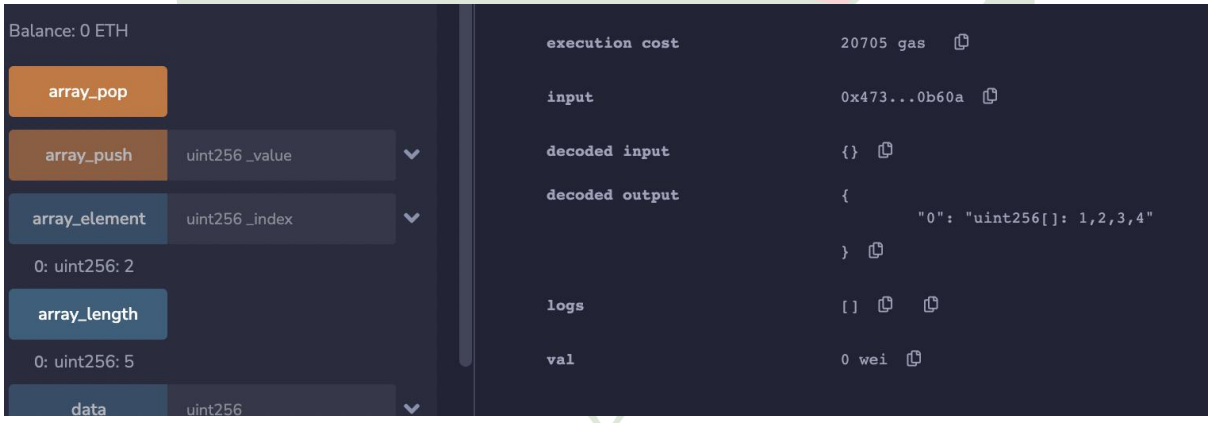
Here the element is being pushed into the array using the push() method and it is inserted at the last of the array.

Accessing an element

Each element in the array can be accessed by the index.

Length of array

Pop element from an array



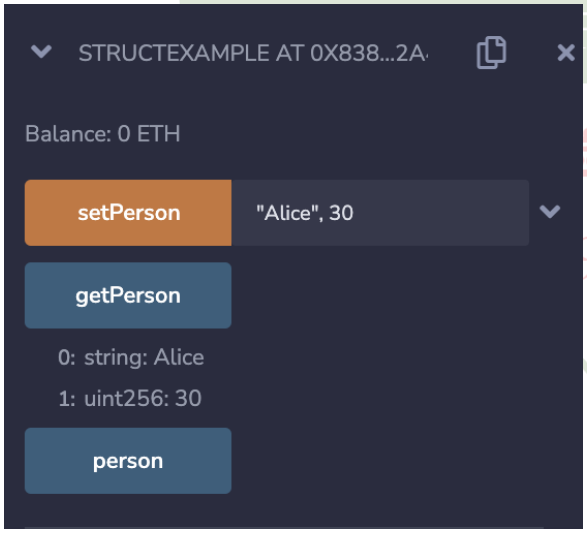
The elements in the array can be popped out using pop() method. The element at the last of the array will be popped out of the array.

PROCEDURE:

(STRUCTURES)

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract StructExample {
    // Define a struct to represent a Person with two fields: name and age.
    struct Person {
        string name;
        uint age;
    }

    // Declare a state variable of the Person struct type.
    Person public person;

    // Function to set the values of the Person struct.
    function setPerson(string memory _name, uint _age) public {    infinite gas
        person.name = _name;
        person.age = _age;
    }

    // Function to get the values of the Person struct.
    function getPerson() public view returns (string memory, uint) {    infinite gas
        return (person.name, person.age);
    }
}
```

Deploying smart contract:



OUTPUT:

```
[vm] from: 0x5B3...eddC4 to: StructExample.setPerson(string,uint256) 0x838...2A4DC value: 0 wei data: 0x2fd...00000
logs: 0 hash: 0x6af...39fe0
status                      true Transaction mined and execution succeed
transaction hash            0x6af9a848b46bfd44482f453d67ebc6e2964e3bf4ac163c5d7fe491e85b239fe0
block hash                  0x7cd7c8ca9a77abef3a3de041f587aaddf7f24891d9b50860b42edecb1833b1e6
block number                40
from                        0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                          StructExample.setPerson(string,uint256) 0x838F9b8228a5C95a7c431bcDAb58E289f5D2A4DC
gas                         77755 gas
transaction cost            67613 gas
execution cost              45941 gas
input                       0x2fd...00000
decoded input               {
                                "string _name": "Alice",
                                "uint256 _age": "30"
                            }
```

```
call to StructExample.getPerson

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: StructExample.getPerson() data: 0x8ec...4dc95
from                        0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                          StructExample.getPerson() 0x838F9b8228a5C95a7c431bcDAb58E289f5D2A4DC
execution cost              5687 gas (Cost only applies when called by a contract)
input                       0x8ec...4dc95
decoded input               {}
decoded output              {
                                "0": "string: Alice",
                                "1": "uint256: 30"
                            }
logs                        []
```

AIM : To explore mathematical and cryptographic functions in Solidity.

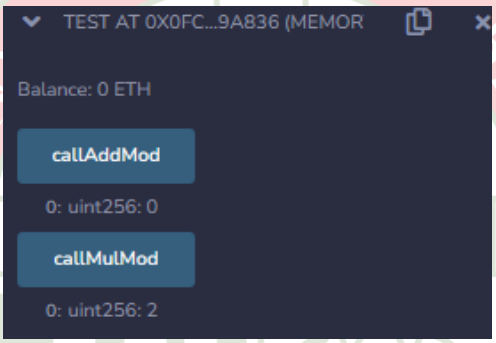
DESCRIPTION :

1) **Mathematical Functions:**
   Solidity provides inbuilt mathematical functions as well. Following are heavily used methods –
   - addmod(uint x, uint y, uint k) returns (uint) – computes (x + y) % k where the addition is performed with arbitrary precision and does not wrap around at 2256.
   - mulmod(uint x, uint y, uint k) returns (uint) – computes (x * y) % k where the addition is performed with arbitrary precision and does not wrap around at 2256.
2) **Cryptographic Functions:**
   Solidity provides inbuilt cryptographic functions as well. Following are important methods –
   - keccak256(bytes memory) returns (bytes32) – computes the Keccak-256 hash of the input.
   - ripemd160(bytes memory) returns (bytes20) – compute RIPEMD-160 hash of the input.
   - sha256(bytes memory) returns (bytes32) – computes the SHA-256 hash of the input.
   - ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address) – recover the address associated with the public key from elliptic curve signature or return zero on error. The function parameters correspond to ECDSA values of the signature: r - first 32 bytes of signature; s: second 32 bytes of signature; v: final 1 byte of signature. This method returns an address.

CODE-1:

```
pragma solidity ^0.5.0;+
contract Test {
  function callAddMod() public pure returns(uint){
    return addmod(4, 5, 3);
  }
  function callMulMod() public pure returns(uint){
    return mulmod(4, 5, 3);
  }
}
```

OUTPUT-1:



CODE-2:

Keccak256

```
pragma solidity ^0.5.0;
contract Test {
  function callKeccak256() public pure returns(bytes32 result){
    return keccak256("ABC");
  }
}
```

OUTPUT-2:

CODE-3:

SHA256

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract HashExample {
   function getHash() public pure returns (bytes32) {
      return sha256(“ABC”);
   }
}
```

OUTPUT-3:

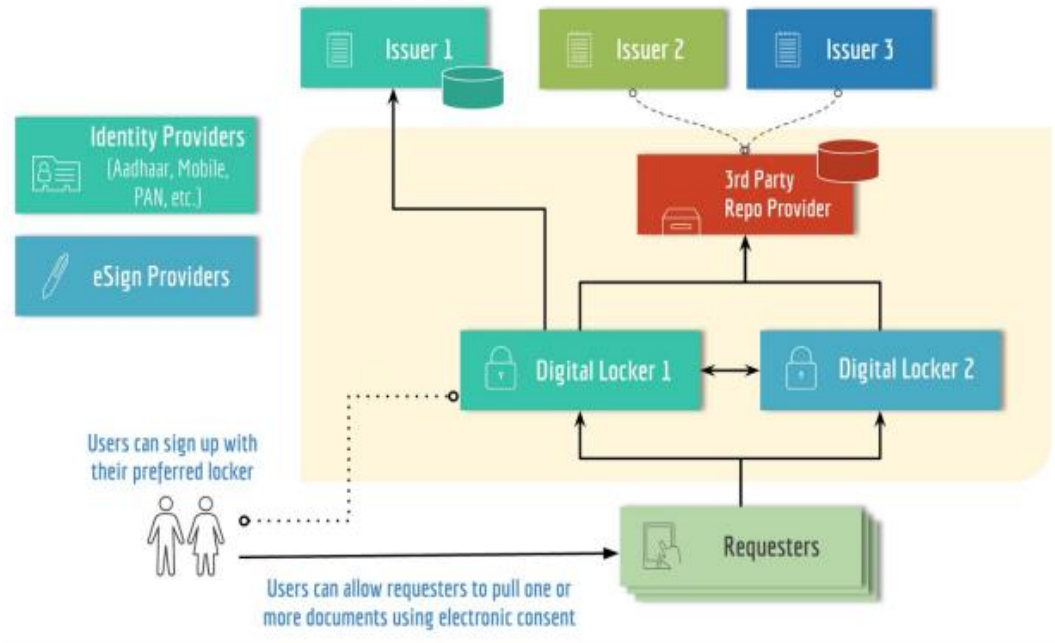
AIM: To Implement Digi Locker using smart contracts and Blockchain.

DESCRIPTION:

DigiLocker is a flagship initiative of Ministry of Electronics & IT (MeitY) under Digital India programme. DigiLocker aims at 'Digital Empowerment' of citizen by providing access to authentic digital documents to citizen's digital document wallet. The issued documents in DigiLocker system are deemed to be at par with original physical documents as per Rule 9A of the Information Technology (Preservation and Retention of Information by Intermediaries providing Digital Locker facilities) Rules, 2016 notified on February 8, 2017 vide G.S.R. 711(E).DigiLocker allows access to digital versions of various documents including drivers licenses, vehicle registration certificates and academic mark sheets. It also provides 1 GB storage space to each account to upload scanned copies of legacy documents.

BENEFITS TO CITIZENS

1. Important Documents Anytime, Anywhere!
2. Authentic Documents, Legally at Par with Originals.
3. Digital Document Exchange with the consent of the citizen.
4. Faster service Delivery- Government Benefits, Employment, Financial
    a. Inclusion, Education, Health.



PROGRAM:

```
pragma solidity ^0.5.0;
pragma experimental ABIEncoderV2;

contract digiLocker {
    //structures and other variable declrn

    struct Document{
        bytes32 docid; //doc id
        string docName;
        string timestamp; //
        bytes32 docHash; //doc hash
    }
    struct UserDetails{
        string firstName;
        string lastName;
        string email;
        string contact;
    }

    struct User{
        userType utype;
        bool valid;
        UserDetails details;
        address _useraddress;
        bytes32 accessKey; // user master key Hash
        string pubKey; // Public key of user
    }
    uint usercount = 0;
    address[] _glbluseraddress;
    ////////////////////////-- enums here --
////////////////////////////////////
    enum userType { Issuer, Resident, Requester, Admin }
    enum Permission {READ, MODIFY}


    ////////////////////////-- events here --
////////////////////////////////////
    event registeredUserEvent(string _email,userType utype,address indexed
_useraddress);
    event uploadDocumentEvent(bytes32 indexed docid, bytes32 docHash, address
indexed user_addr);
    event sharedDocumentEvent(bytes32 indexed docid, address indexed
docOwner,address indexed sharedWith, uint8 permission);
    event verifyDocumentEvent(bytes32 indexed docid, address indexed docOwner,
address indexed sharedWith);


    ////////////////////////-- mapping here --
////////////////////////////////////
    mapping(address => User) registerUsers;
    mapping (address => Document[])  ownerDocuments;
    mapping (string => address)  emailAddressMapping;
```

```
    ////////////////////-- functions here --
////////////////////////////////
    function isalreadyRegisteredUser() public view returns(bool){
        if(registerUsers[msg.sender]._useraddress ==
0x0000000000000000000000000000000000000000){
            return false;
        }
        else{
            return true;
        }
    }
    // temp function
    function getUseraccessKey() public view returns(bytes32){
       return (registerUsers[msg.sender].accessKey);
    }

    function getUserType() public view returns(uint8){
        return uint8(registerUsers[msg.sender].utype);
    }

    function getRegisteredUser() public view returns(bytes32, address, string
memory){
       return (
           registerUsers[msg.sender].accessKey,
           registerUsers[msg.sender]._useraddress,
           registerUsers[msg.sender].details.firstName
       );
    }

    //register user
    function registerUser(string memory _firstName,
            string memory _lastName,
            string memory _email, uint8 _utype,
            string memory _contact, bytes32 accessKey,
            string memory pubKey) public returns(bool) {
            UserDetails memory d = UserDetails(_firstName,_lastName, _email,
_contact);
                User memory newuser = User(
                    userType(_utype), true, d,
                    msg.sender, accessKey, pubKey
                );

                emailAddressMapping[_email] = msg.sender;

                registerUsers[msg.sender] = newuser;
                emit registeredUserEvent(_email, userType(_utype),
msg.sender);
                _glbluseraddress.push(msg.sender);
                usercount++;
                return true;
    }

    function checkAlreadyUpload(bytes32 docId)public view returns(bool){
        for(uint i = 0; i<ownerDocuments[msg.sender].length; i++)
            if(ownerDocuments[msg.sender][i].docid == docId)
                return true;
        return false;
```

```
    }

    function getDocCountByUserId() public view returns(uint256){
        return ownerDocuments[msg.sender].length;
    }

    function uploadDocument(string memory docName, bytes32 docId, bytes32
docHash, string memory timestamp) public returns(bool){
        Document memory d = Document(docId, docName, timestamp, docHash);
        ownerDocuments[msg.sender].push(d);
        emit uploadDocumentEvent(docId, docHash, msg.sender);
        return true;
    }

    function shareDocumentwithUser(bytes32 docid, uint8 permission,address
_requester) public{
        //  address  _owner, = msg.sender
        emit sharedDocumentEvent(docid, msg.sender, _requester, permission);
    }

    function verifyUserDocument(bytes32 docid, address  _owner) public{
        // , address _requester = msg.sender
        emit verifyDocumentEvent(docid,_owner,msg.sender);
    }

    function isValidSharableUser(string memory email_) public view
returns(bool){
        if(emailAddressMapping[email_] !=
0x0000000000000000000000000000000000000000 &&
            uint8(registerUsers[emailAddressMapping[email_]].utype) == 2){
            return true;
        }
        else{
            return false;
        }
    }

    function getOwnerDocInfoByDocId(bytes32 docId)public view returns (string
memory, string memory){

        for(uint i = 0; i < ownerDocuments[msg.sender].length; i++){
            if(ownerDocuments[msg.sender][i].docid == docId)
                return (ownerDocuments[msg.sender][i].docName,
                    ownerDocuments[msg.sender][i].timestamp
                );

        }
    }

    function getOwnerDocumetList()public view returns (
        string[] memory, string[] memory, bytes32[] memory) {
        return getDocumetList(msg.sender);
    }

    function getDocumetList(address _useradd)public view returns (
        string[] memory, string[] memory, bytes32[] memory) {
```

```
      string[] memory _docName = new
string[](ownerDocuments[_useradd].length);
      string[] memory _timestamp = new
string[](ownerDocuments[_useradd].length);
      bytes32[] memory _docid = new
bytes32[](ownerDocuments[_useradd].length);


      for(uint i=0;i<ownerDocuments[_useradd].length;i++){
         _timestamp[i] = ownerDocuments[_useradd][i].timestamp;
         _docName[i] = ownerDocuments[_useradd][i].docName;
         _docid[i] = ownerDocuments[_useradd][i].docid;

      }

      return (_docName,_timestamp, _docid);
    }


    function getDocumentListbyDocId(bytes32 _docId) public view returns(
        bytes32,
        string memory,
        string memory,
        bytes32,
        string memory,
        string memory,
        string memory,
        string memory){
        for(uint i=0;i<usercount;i++){
            for(uint j=0;j < ownerDocuments[_glbluseraddress[i]].length;j++){

                if(ownerDocuments[_glbluseraddress[i]][j].docid == _docId)

                    return (
                        ownerDocuments[_glbluseraddress[i]][j].docid,
                        ownerDocuments[_glbluseraddress[i]][j].docName,
                        ownerDocuments[_glbluseraddress[i]][j].timestamp,
                        ownerDocuments[_glbluseraddress[i]][j].docHash,
                        registerUsers[_glbluseraddress[i]].details.firstName,
                        registerUsers[_glbluseraddress[i]].details.email,
                        registerUsers[_glbluseraddress[i]].details.lastName,
                        registerUsers[_glbluseraddress[i]].details.contact
                    );

            }

        }

    }

    function getEmailIdByAddrss()public view returns(string memory,string
memory,string memory){
        return (registerUsers[msg.sender].details.email,
                registerUsers[msg.sender].details.firstName,
                registerUsers[msg.sender].details.lastName);
    }
```

```
    function getAddressByEmail(string memory _email)public view
returns(address){
        return emailAddressMapping[_email];
    }

    //To find owners details
    function getEmailIdByUsrAddr(address _usraddrs)public view returns(string
memory,string memory,string memory){
        return (registerUsers[_usraddrs].details.email,
                registerUsers[_usraddrs].details.firstName,
                registerUsers[_usraddrs].details.lastName);
    }

    function getDocIndex(bytes32 _docid_,address _uaddr_)public view
returns(uint256){

       for(uint i=0;i < ownerDocuments[_uaddr_].length ;i++){
           if(ownerDocuments[_uaddr_][i].docid == _docid_){
               return i;
           }
       }
    }

    function getPublicKey(address _uaddr_)public view returns(string memory){
        return registerUsers[_uaddr_].pubKey;
    }


    function getDocumentName(bytes32 _docId, address docOwner) public view
returns(string memory){
        for(uint j=0;j < ownerDocuments[docOwner].length;j++){
            if(ownerDocuments[docOwner][j].docid == _docId)
                return (ownerDocuments[docOwner][j].docName);
        }
    }

}
```


EXPLANATION

1. Contract Structure: The contract is named "DigiLocker" and is written in Solidity version 0.5.0. It uses experimental ABIEncoderV2, which is used for encoding and decoding data structures.

2. Data Structures: The contract defines three main data structures: `Document`, `UserDetails`, and `User`. These structures hold information about documents, user details, and user accounts, respectively. The `userType` and `Permission` enums define user types (e.g., Issuer, Resident) and document permission levels (READ, MODIFY).

3. Events: The contract declares several events, such as `registeredUserEvent`, `uploadDocumentEvent`, `sharedDocumentEvent`, and `verifyDocumentEvent`, to log important contract interactions.

4. Mappings: The contract uses mappings to store data, including registered users, user documents, and an email address mapping for quick lookups.

5. User Registration: Users can register by providing personal details, including name, email, user type, contact, access key, and a public key. This information is stored in the contract.

6. Document Management: Users can upload documents with associated metadata, such as document name, ID, hash, and timestamp. These documents are stored in the contract and associated with the user who uploaded them. Users can check if a document with a given ID has already been uploaded. Users can retrieve their own documents and document counts.

7. Sharing Documents: Users can share documents with other users, specifying the document's ID, the recipient's address, and the permission level (READ or MODIFY).

8. Document Verification: Users can verify documents shared with them.

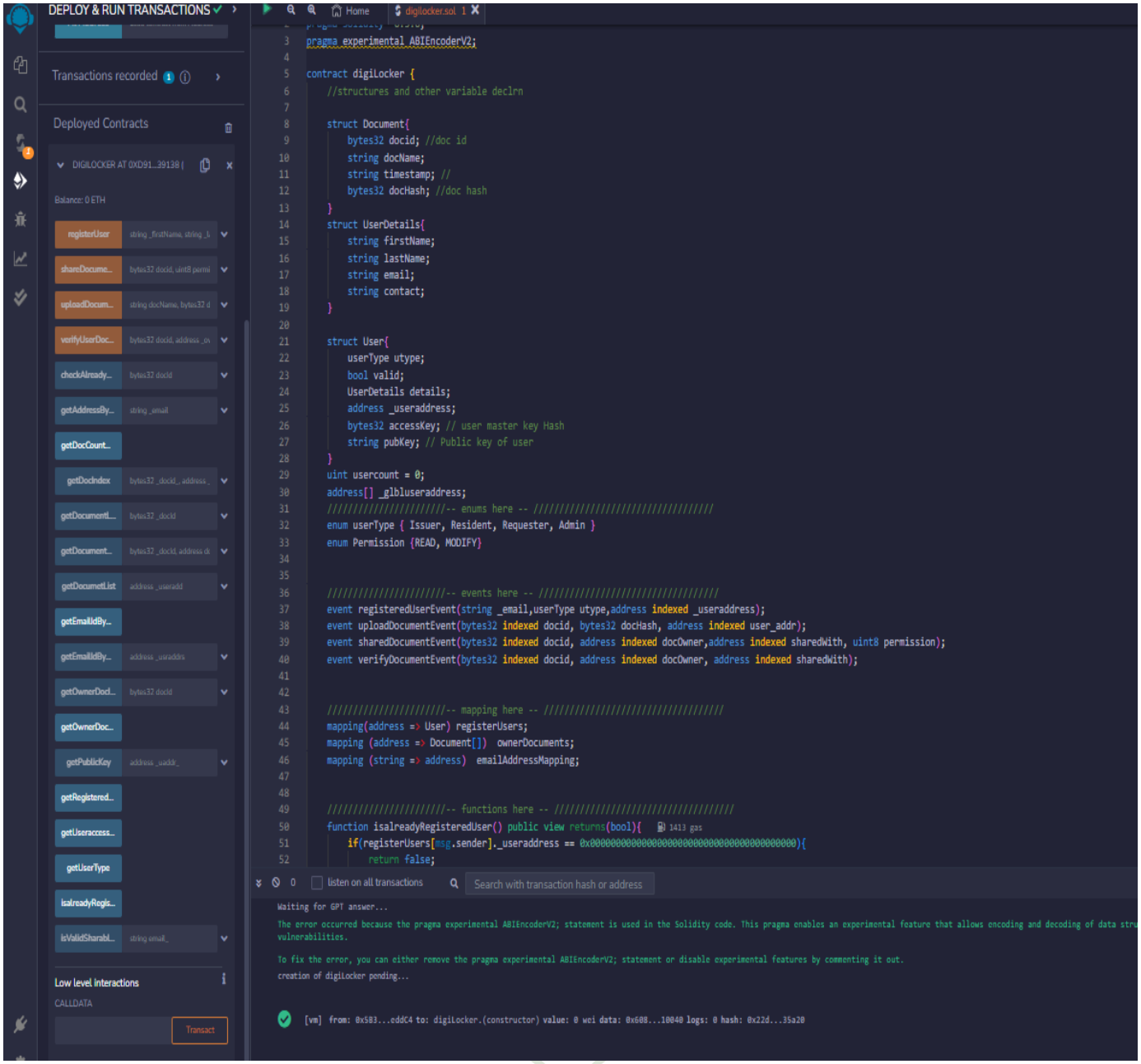
9. User and Document Listing: Functions are provided to list a user's documents and details, as well as to retrieve details about documents by their ID.Functions allow users to retrieve their email address, first name, last name, and find an address by email.

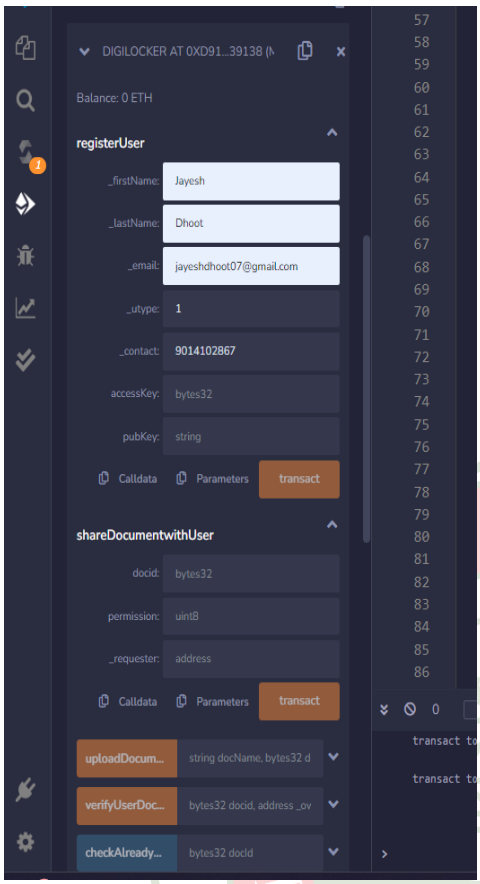
10. Document Index and Public Key: Functions enable users to find the index of a document by its ID and retrieve the public key of a user by their address.

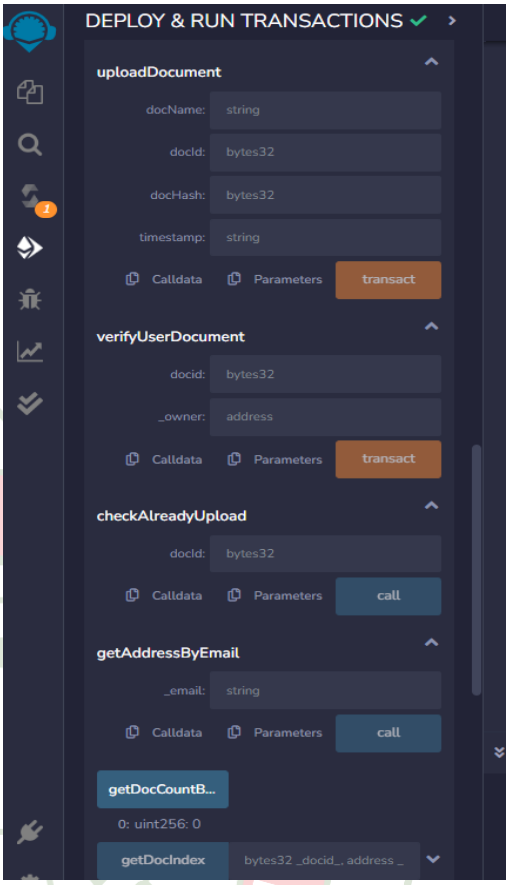
11. Document Name Retrieval: A function is provided to retrieve the name of a document by its ID and its owner's address.

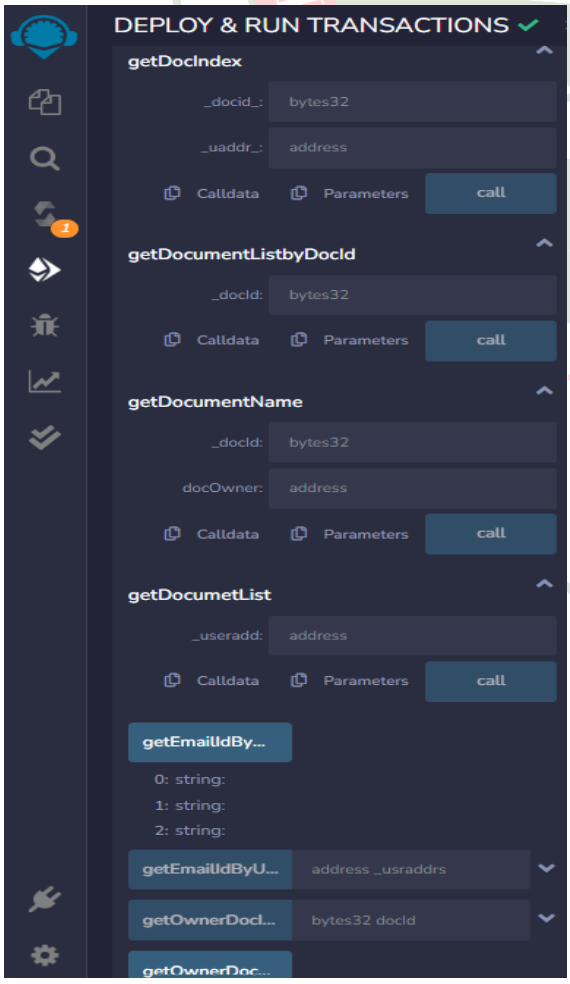
The contract aims to serve as a secure and auditable digital locker system where users can store and manage their documents, share them with others, and verify document authenticity. It also provides user registration and user details management functionality. The use of blockchain technology ensures data immutability and transparency.

SAMPLE OUTPUTS




DIGILOCKER AT 0XD91...39138 (M
Balance: 0 ETH
registerUser
_firstName: Jayesh
_lastName: Dhoot
_email: jayeshdhoot07@gmail.com
_utype: 1
_contact: 9014102867
accessKey: bytes32
pubKey: string
Calldata
Parameters
transact
shareDocumentwithUser
docId: bytes32
permission: uint8
_requester: address
Calldata
Parameters
transact
uploadDocum...
string docName, bytes32 d
verifyUserDoc...
bytes32 docId, address _ov
checkAlready...
bytes32 docId


DEPLOY & RUN TRANSACTIONS
uploadDocument
docName: string
docId: bytes32
docHash: bytes32
timestamp: string
Calldata
Parameters
transact
verifyUserDocument
docId: bytes32
_owner: address
Calldata
Parameters
transact
checkAlreadyUpload
docId: bytes32
Calldata
Parameters
call
getAddressByEmail
_email: string
Calldata
Parameters
call
getDocCountB...
0: uint256: 0
getDocIndex
bytes32 _docId_, address _


DEPLOY & RUN TRANSACTIONS
getDocIndex
docId: bytes32
uaddr: address
Calldata
Parameters
call
getDocumentListbyDocId
_docId: bytes32
Calldata
Parameters
call
getDocumentName
_docId: bytes32
docOwner: address
Calldata
Parameters
call
getDocumetList
_useradd: address
Calldata
Parameters
call
getEmailIdBy...
0: string:
1: string:
2: string:
getEmailIdByU...
address _usraddrs
getOwnerDocI...
bytes32 docId

AIM: To setup Hyperledger fabric in windows

DESCRIPTION:

Hyperledger Fabric is an open-source platform for building distributed ledger solutions, with a modular architecture that delivers high degrees of confidentiality, flexibility, resiliency, and scalability. This enables solutions developed with fabric to be adapted for any industry. This is a private and confidential blockchain framework managed by the Linux Foundation.

Hyperledger Fabric is designed for use in enterprise-level applications, and it is characterized by its modular architecture, permissioned network, and smart contract functionality, known as “chaincode”.

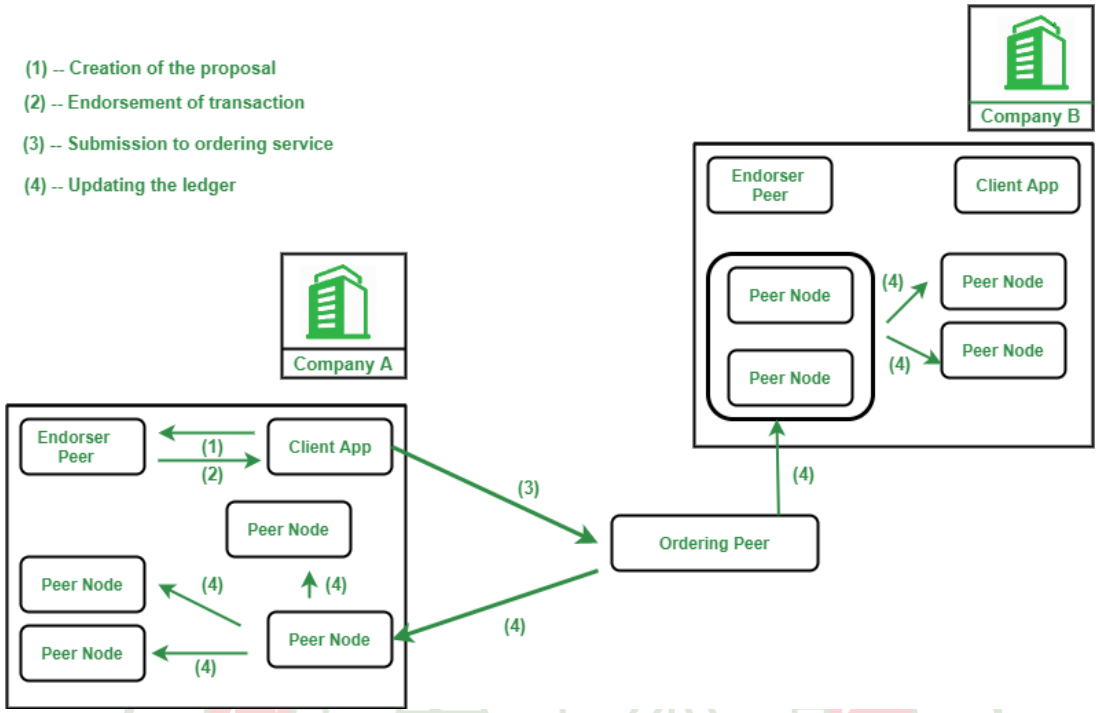
- The platform provides a high degree of security, privacy, and scalability, and it supports the development of custom blockchain solutions for various use cases across industries such as finance, supply chain, and healthcare.
- Hyperledger Fabric operates as a network of nodes, where each node performs a specific function, such as validating transactions, maintaining the ledger, and executing chaincode.
- Transactions are validated and ordered by a consensus mechanism, which ensures the integrity and consistency of the ledger.

Components:

Hyperledger fabric is an enterprise-level permission blockchain network. It is made up of various unique organizations or members that interact with each other to serve a specific purpose. For example, these organizations can be a bank, financial institution, or a supply chain network. Each organization is identified and they have a fabric certificate authority. These organizations are called members.

Each member of the fabric can set up one or more authorized peers to participate in the network using the fabric certificate authority. All of these peers must be authorized properly.

There is a client-side application connected to the network written with the software development kit (SDK) of any particular programming language.



Steps to setup Fabric in Windows:

1) Open windows power shell in administrator mode

```
wsl -- install
```

```
wsl -l -v
```

2) Open Windows feature -> enable Windows subsystem for Linux and Virtual machine platform.

3) Download Linux Kernel Update Package from `https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update _x64.msi`

4) To open Ubuntu,

```
run wsl command in powershell
```

5) Update and Upgrade Ubuntu

Run the following commands in the Ubuntu terminal:

- `sudo apt update`
- `sudo apt upgrade`

6) Download Docker Desktop for Windows

```
https://desktop.docker.com/win/stable/amd64/Docker%20Desktop%20I nstaller.exe
```

7) Enable Docker for Ubuntu

In Docker Desktop settings,

go to Resources > WSL Integration and enable Ubuntu.

Hit Apply and Restart to enable Docker for the Ubuntu command line.

8) Install Go and Verify Versions In the Ubuntu terminal,

run the following commands:

- `sudo apt-get install curl`
- `curl -V`
- `sudo wget https://golang.org/dl/go1.16.3.linux-amd64.tar.gz`
- `tar xvf go1.16.3.linux-amd64.tar.gz`
- `export GOPATH=$HOME/go`
- `export PATH=$PATH:$GOPATH/bin`
- `go version`

9) Verify Git Installation

Run the following command to verify Git installation:

```
git --version
```

10) Install Hyperledger Fabric and Fabric Samples

Navigate to the directory where you want to install Hyperledger Fabric and Fabric Samples.

In bash

- `mkdir -p $HOME/go/src/github.com/`
- `cd $HOME/go/src/github.com/`
- `curl -sSL https://bit.ly/2ysbOFE | bash -s`

11) Set up Hyperledger Fabric Test Network

Navigate to the test-network directory and bring up the Hyperledger Fabric test network:

In bash:

- `cd $wsl`
- `cd go/src/github.com/fabric-samples/test-network`
- `./network.sh down`
- `./network.sh up`
- `docker images`
- `docker ps -a`

12) Check Components of the Test Network

To check the components of the test network, run:

In bash:

- `docker ps -a`

Hyperledger Fabric is now setup and is ready for development and testing