A Project/Dissertation Review-2 Report

on

ETHEREUM BLOCKCHAIN

Submitted in partial fulfillment of the requirement for the award of the degree of

B. Tech Computer Science Engineering

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Mr. Tarun Kumar
Assistant Professor

Submitted By:-

Nikhil Kumar

19SCSE1010851

Hargun Singh Dhall

19SCSE1010910

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA October, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"Deploying Smart Contract Using Ethereum Block chain"** in partial fulfilment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**

submitted in the

School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of **July, 2021 to December, 2021**, under the supervision of **Mr. Tarun Kumar, Assistant Professor, Department of Computer Science and Engineering** of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

Nikhil Kumar– 19SCSE1010851

Hargun Singh Dhall- 19SCSE1010910

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor (Mr Tarun Kumar

Assistant Professor)

CERTIFICATE

The Project Viva-Voce examination of **Nikhil Kumar – 19SCSE1010851, Hargun Singh Dhall -19SCSE1010910** has been held on____________________and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING.**

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021

Place: Greater Noida

Abstract

In today's growing period, blockchain is playing a very important role in holding the large set of data which is distributed over the entire network. It is mostly simply defined as a decentralized, distributed ledger technology that records the provenance of a digital asset. Advantage of blockchain is that it increases trust, security , transparency and the traceability of the data shared across a business network. Ethereum is a blockchain platform with its own cryptocurrency, called Ether (ETH) OR Ethereum , and its own programming language , called Solidity .

In this project we will apply certain alogorithm . There will be participants which are in some number which will send ether to contract address . For this project the minimum criteria is atleast 3 or more than three participants can participate for lottery system . Then there is a manager who will take care of the process . The one who wins the lottery which is selected by the manager randomly will get all contract balance which was there in the contract address.

Some of the tools that we are using to make this project are Remix id , and the programming language that we are using is the solidity . Remix id contains all the feature that is relate to blockchain technology.

In conclusion , we hope to create in a transparent way of selecting the winner of lottery tickets. It is secured because it will operate from the server.

List of Tables

Table No.	Table Name	Page Number
1.	Table for Student Data	3
2.	Table for Faculty Data	4

List of Figures

Figure No.	Table Name	Page Number
1.	UML Diagram	7
2.	Data Flow Diagram	8
3.	Functionality Flow Chart	8

Acronyms

B.Tech.	Bachelor of Technology
M.Tech.	Master of Technology
BCA	Bachelor of Computer Applications
MCA	Master of Computer Applications
B.Sc. (CS)	Bachelor of Science in Computer Science
M.Sc. (CS)	Master of Science in Computer Science
SCSE	School of Computing Science and Engineering

Table of Contents

Title		Page No.
Abstract		**I**
List of Table		**II**
List of Figures		**III**
Chapter 1	**Introduction**	**1**
	1.1 Introduction	**2**
	1.2 Formulation of Problem	**3**
	1.2.1 Tool and Technology Used	
Chapter 2	**Literature Survey/Project Design**	**5**

CHAPTER-1
Introduction

Bitcoin and blockchain technology have begun to shape and define new aspects in the computer science and information technology. The need for a decentralized money has been exploited more as a theoretical concept, but in the past decade, it became viable, all thanks to the famous paper of Satoshi Nakamoto in 2008, introducing Bitcoin and blockchain technology.

While there are controversies about Nakamoto's true identity, one is for sure: he brought something revolutionary to the world, and it is up to the users to decide what they want to do with it. Some will take this opportunity and develop their own application for solving various problems in the society, others will invest money in those ideas or simply trade with ups and downs of the cryptocurrencies' values at the market.
In this paper, we thought of bringing a small introduction to the matter of blockchain and cryptocurrencies. We begin with a quick retrospective of some of the most famous solutions for decentralized digital money before Bitcoin, and then we go into the very core of its function, together with Ethereum. These two cryptocurrencies hold majority of the cryptocurrency market capitalization. Of course, as it happens with new technologies, some limitations and problems emerged, and we described them as well.

Smart contracts are used to exchange money or ownership, store data, make decisions, and interact with other contracts. They are pieces of computer code that are programmed to perform operations on an outcome of an event based on predetermined criteria set by the creators of the contract. Often these smart contracts exist on distributed ledger technology, as of now, they primarily use blockchain.

Ethereum is a well-known blockchain network that uses smart contracts to create decentralized applications, issue tokens, and manage governance. While at first smart contracts may seem mysterious, at a second glance their operations and applications start to make a little more sense.

CHAPTER-1
Introduction

Formulation of Problem:

We already know that there has been a problem for selecting the winner of the lottery participants. Problems can be of different types such as:

1. It can be on the basis of money.
2. It can be on the basis of torture.
3. It can be on the basis of contacts.

So it is important that we make all the process digitally and transparent to others so that it’s process cannot destroy by others.

Tools and technology used are:

REMIX IDE: It is an open source web and desktop application. It is the powerful open source tool that helps you to write solidity contracts straight from the browser. It is written in javascript and supports both usage in the browser, in the browser but run locally and in a desktop version.

JavaScript- JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive.

Solidity: It is an object oriented programming language for writing smart contracts. It is used for implementation smart contracts on various blockchain platform, most notably, Ethereum.

CHAPTER-2
Literature Survey

This work provides a systematic literature review of blockchain-based applications across multiple domains. The aim is to investigate the current state of blockchain technology and its applications and to highlight how specific characteristics of this disruptive technology can revolutionise “business-as-usual” practices. To this end, the theoretical underpinnings of numerous research papers published in high ranked scientific journals during the last decade, along with several reports from grey literature as a means of streamlining our assessment and capturing the continuously expanding blockchain domain, are included in this review.

Based on a structured, systematic review and thematic content analysis of the discovered literature, we present a comprehensive classification of blockchain-enabled applications across diverse sectors such as supply chain, business, healthcare, IoT, privacy, and data management, and we establish key themes, trends and emerging areas for research. We also point to the shortcomings identified in the relevant literature, particularly limitations the blockchain technology presents and how these limitations spawn across different sectors and industries.

Building on these findings, we identify various research gaps and future exploratory directions that are anticipated to be of significant value both for academics and practitioners.
In today’s growing period, blockchain is playing a very important role in holding the large set of data which is distributed over the entire network. It is mostly simply defined as a decentralized, distributed ledger technology that records the provenance of a digital asset. Advantage of blockchain is that it increases trust, security, transparency and the traceability of the data shared across a business network.

Chapter-3

A blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.

One key difference between a typical database and a blockchain is how the data is structured. A blockchain collects information together in groups, known as blocks, that hold sets of information. Blocks have certain storage capacities and, when filled, are closed and linked to the previously filled block, forming a chain of data known as the blockchain. All new information that follows that freshly added block is compiled into a newly formed block that will then also be added to the chain once filled.

A database usually structures its data into tables, whereas a blockchain, like its name implies, structures its data into chunks (blocks) that are strung together. This data structure inherently makes an irreversible time line of data when implemented in a decentralized nature. When a block is filled, it is set in stone and becomes a part of this time line. Each block in the chain is given an exact time stamp when it is added to the chain.

A **blockchain** is a growing list of records, called *blocks*, that are linked together using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree). The timestamp proves that the transaction data existed when the block was published in order to get into its hash. As blocks each contain information about the block previous to it, they form a chain, with each additional block reinforcing the ones before it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks.

Blockchains are typically managed by a peer to peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks. Although blockchain records are not unalterable as forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high fault tolerance.

The blockchain was popularized by a person (or group of people) using the name in 2008 to serve as the public transaction ledger of the , based on work by Stuart Haber, W. Scott Stornetta, and Dave Bayer. The identity of Satoshi Nakamoto remains unknown to date. The implementation of the blockchain within bitcoin made it the first digital currency to solve the problem without the need of a trusted authority or central . The bitcoin design has inspired other applications and blockchains that are readable by the public and are widely used by . The blockchain is considered a type of cryptocurrency.

Private blockchains have been proposed for business use. *Computerworld* called the marketing of such privatized blockchains without a proper security model " cryptocurrency" however, others have argued that permissioned blockchains, if carefully designed, may be more decentralized and therefore more secure in practice than permissionless ones.

Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An *asset* can be tangible (a house, car, cash, land) or intangible (intellectual property, patents, copyrights, branding). Virtually anything of value can be tracked and traded on a blockchain network, reducing risk and cutting costs for all involved.

Is Blockchain Secure?

Blockchain technology achieves decentralized security and trust in several ways. To begin with, new blocks are always stored linearly and chronologically. That is, they are always added to the “end” of the blockchain. After a block has been added to the end of the blockchain, it is extremely difficult to go back and alter the contents of the block unless a majority of the network has reached a consensus to do so. That’s because each block contains its own hash, along with the hash of the block before it, as well as the previously mentioned time stamp. Hash codes are created by a mathematical function that turns digital information into a string of numbers and letters. If that information is edited in any way, then the hash code changes as well.

Let’s say that a hacker, who also runs a node on a blockchain network, wants to alter a blockchain and steal cryptocurrency from everyone else. If they were to alter their own single copy, it would no longer align with everyone else’s copy. When everyone else cross-references their copies against each other, they would see this one copy stand out, and that hacker’s version of the chain would be cast away as illegitimate.

Succeeding with such a hack would require that the hacker simultaneously control and alter 51% or more of the copies of the blockchain so that their new copy becomes the majority copy and, thus, the agreed-upon chain. Such an attack would also require an immense amount of money and resources, as they would need to redo all of the blocks because they would now have different time stamps and hash codes.

Due to the size of many cryptocurrency networks and how fast they are growing, the cost to pull off such a feat probably would be insurmountable. This would be not only extremely expensive but also likely fruitless. Doing such a thing would not go unnoticed, as network members would see such drastic alterations to the blockchain. The network members would then hard fork off to a new version of the chain that has not been affected. This would cause the attacked version of the token to plummet in value, making the attack ultimately pointless, as the bad actor has control of a worthless asset. The same would occur if the bad actor were to attack the new fork of Bitcoin. It is built this way so that taking part in the network is far more economically incentivized than attacking it.

Blockchain vs. Banks
Blockchains have been heralded as being a disruptive force to the finance sector, and especially with the functions of payments and banking. However, banks and decentralized blockchains are vastly different.

To see how a bank differs from blockchain, let's compare the banking system to Bitcoin's implementation of blockchain.

Blockchain vs. Banks

Feature	Banks	Bitcoin
Hours open	Typical brick-and-mortar banks are open from 9:00 am to 5:00 pm on weekdays. Some banks are open on weekends but with limited hours. All banks are closed on banking holidays.	No set hours; open 24/7, 365 days a year.
Transaction Fees	•Card payments: This fee varies based on the card and is not paid by the user directly. Fees are paid to the payment processors by stores and are usually charged per transaction. The effect of this fee can sometimes make the cost of goods and services rise. •Checks: can cost between $1	Bitcoin has variable transaction fees determined by miners and users. This fee can range between $0 and $50 but users have the ability to determine how much of a fee they are willing to pay. This creates an open marketplace where if the user sets their fee too low their

Feature	Banks	Bitcoin
	and $30 depending on your bank. •ACH: ACH transfers can cost up to $3 when sending to external accounts. •Wire: Outgoing domestic wire transfers can cost as much as $25. Outgoing international wire transfers can cost as much as $45.	transaction may not be processed.
Transaction Speed	•Card payments: 24-48 hours •Checks: 24-72 hours to clear •ACH: 24-48 hours •Wire: Within 24 hours unless international *Bank transfers are typically not processed on weekends or bank holidays	Bitcoin transactions can take as little as 15 minutes and as much as over an hour depending on network congestion.
Know Your Customer Rules	Bank accounts and other banking products require "Know Your Customer" (KYC) procedures. This means it is legally required for banks to record a customer's identification prior to opening an account.	Anyone or anything can participate in Bitcoin’s network with no identification. In theory, even an entity equipped with artificial intelligence could participate.

Feature	Banks	Bitcoin
Ease of Transfers	Government-issued identification, a bank account, and a mobile phone are the minimum requirements for digital transfers.	An internet connection and a mobile phone are the minimum requirements.
Privacy	Bank account information is stored on the bank's private servers and held by the client. Bank account privacy is limited to how secure the bank's servers are and how well the individual user secures their own information. If the bank's servers were to be compromised then the individual's account would be as well.	Bitcoin can be as private as the user wishes. All Bitcoin is traceable but it is impossible to establish who has ownership of Bitcoin if it was purchased anonymously. If Bitcoin is purchased on a KYC exchange then the Bitcoin is directly tied to the holder of the KYC exchange account.
Security	Assuming the client practices solid internet security measures like using secure passwords and two-factor authentication, a bank account's information is only as secure as the bank's server that contains client account information.	The larger the Bitcoin network grows the more secure it gets. The level of security a Bitcoin holder has with their own Bitcoin is entirely up to them. For this reason it is recommended that people use cold storage for larger quantities of Bitcoin

Feature	Banks	Bitcoin
		or any amount that is intended to be held for a long period of time.
Approved Transactions	Banks reserve the right to deny transactions for a variety of reasons. Banks also reserve the right to freeze accounts. If your bank notices purchases in unusual locations or for unusual items they can be denied.	The Bitcoin network itself does not dictate how Bitcoin is used in any shape or form. Users can transact Bitcoin how they see fit but should also adhere to the guidelines of their country or region.
Account Seizures	Due to KYC laws, governments can easily track people's banks accounts and seize the assets within them for a variety of reasons.	If Bitcoin is used anonymously governments would have a hard time tracking it down to seize it.

Blockchain Decentralization

Imagine that a company owns a server farm with 10,000 computers used to maintain a database holding all of its client's account information. This company owns a warehouse building that contains all of these computers under one roof and has full control of each of these computers and all of the information contained within them. This, however, provides a single point of failure. What happens if the electricity at that location goes out? What if its Internet connection is severed? What if it burns to the ground? What if a bad actor erases everything with a single keystroke? In any case, the data is lost or corrupted.

What a blockchain does is to allow the data held in that database to be spread out among several network nodes at various locations. This not only creates redundancy but also maintains the fidelity of the data stored therein—if somebody tries to alter a record at one instance of the database, the other nodes would not be altered and thus would prevent a bad actor from doing so. If one user tampers with Bitcoin's record of transactions, all other nodes would cross-reference each other and easily pinpoint the node with the incorrect information. This system helps to establish an exact and transparent order of events. This way, no single node within the network can alter information held within it.

Because of this, the information and history (such as of transactions of a cryptocurrency) are irreversible. Such a record could be a list of transactions (such as with a cryptocurrency), but it also is possible for a blockchain to hold a variety of other information like legal contracts, state identifications, or a company's product inventory.

Types of blockchain networks

There are several ways to build a blockchain network. They can be public, private, permissioned or built by a consortium.

Public blockchain networks

A public blockchain is one that anyone can join and participate in, such as Bitcoin. Drawbacks might include substantial computational power required, little or no privacy for transactions, and weak security. These are important considerations for enterprise use cases of blockchain.

If one desires to create a completely open blockchain, similar to Bitcoin, which enables anyone and everyone to join and contribute to the network, they can go for a public blockchain. In a public blockchain, anyone is free to join and participate in the core activities of the blockchain network. Anyone can read, write, and audit the ongoing activities on the public blockchain network, which helps a public blockchain maintain its self-governed nature.

The public network operates on an incentivizing scheme that encourages new participants to join and keep the network agile. Public blockchains offer a particularly valuable solution from the point of view of a truly decentralized, democratized, and authority-free operation.

There are a few disadvantages to a public blockchain. The primary one is the heavy power consumption that is necessary to maintain the distributed public ledger. Other issues include the lack of complete privacy and anonymity. This can lead to weaker security of the network and of the participant's identity. Along with genuine contributors, at times the participants may also include fraudulent members who may be involved in malicious activities like hacking, token stealing, and network clogging.

Private blockchain networks

A private blockchain network, similar to a public blockchain network, is a decentralized peer-to-peer network. However, one organization governs the network, controlling who is allowed to participate, execute a consensus protocol and maintain the shared ledger. Depending on the use case, this can significantly boost trust and confidence between participants.

A private blockchain can be run behind a corporate firewall and even be hosted on premises.

If one needs to run a private blockchain that allows only selected entry of verified participants, like those for a private business, one can opt for a private blockchain implementation. A participant can join such a private network only through an authentic and verified invitation. A validation is also necessary either by the network operator(s) or by a clearly defined set protocol implemented by the network.

The primary distinction between the public and private blockchains is that private blockchains control who is allowed to participate in the network, execute the consensus protocol that decides the mining rights and rewards, and maintain the shared ledger. The owner or operator has the right to override, edit, or delete the necessary entries on the blockchain as required.

In the truest sense, a private blockchain is not decentralized and is a distributed ledger that operates as a closed, secure database based on cryptography concepts. Technically speaking, not everyone can run a full node on the private blockchain, make transactions, or validate/authenticate the blockchain changes.

Permissioned blockchain networks

Businesses who set up a private blockchain will generally set up a permissioned blockchain network. It is important to note that public blockchain networks can also be permissioned. This places restrictions on who is allowed to participate in the network and in what transactions. Participants need to obtain an invitation or permission to join.
The third category of blockchains is permissioned blockchains. Permissioned blockchains allow for a mixed bag between the public and private blockchains and support many customization options. These include allowing anyone to join the permissioned network after suitable verification of their identity, and allocation of select and designated permissions to perform only certain activities on the network. For example, Ripple, one of the largest cryptocurrencies, supports permission-based roles for participants.

Such blockchains are built so that they grant special permissions to each participant. This allows participants the ability to perform specific functions such as read, access, and write information on the blockchains. Businesses are increasingly opting for permissioned blockchain networks, as this allows them to selectively place restrictions while configuring the networks, and control the activities of the various participants in the desired roles.

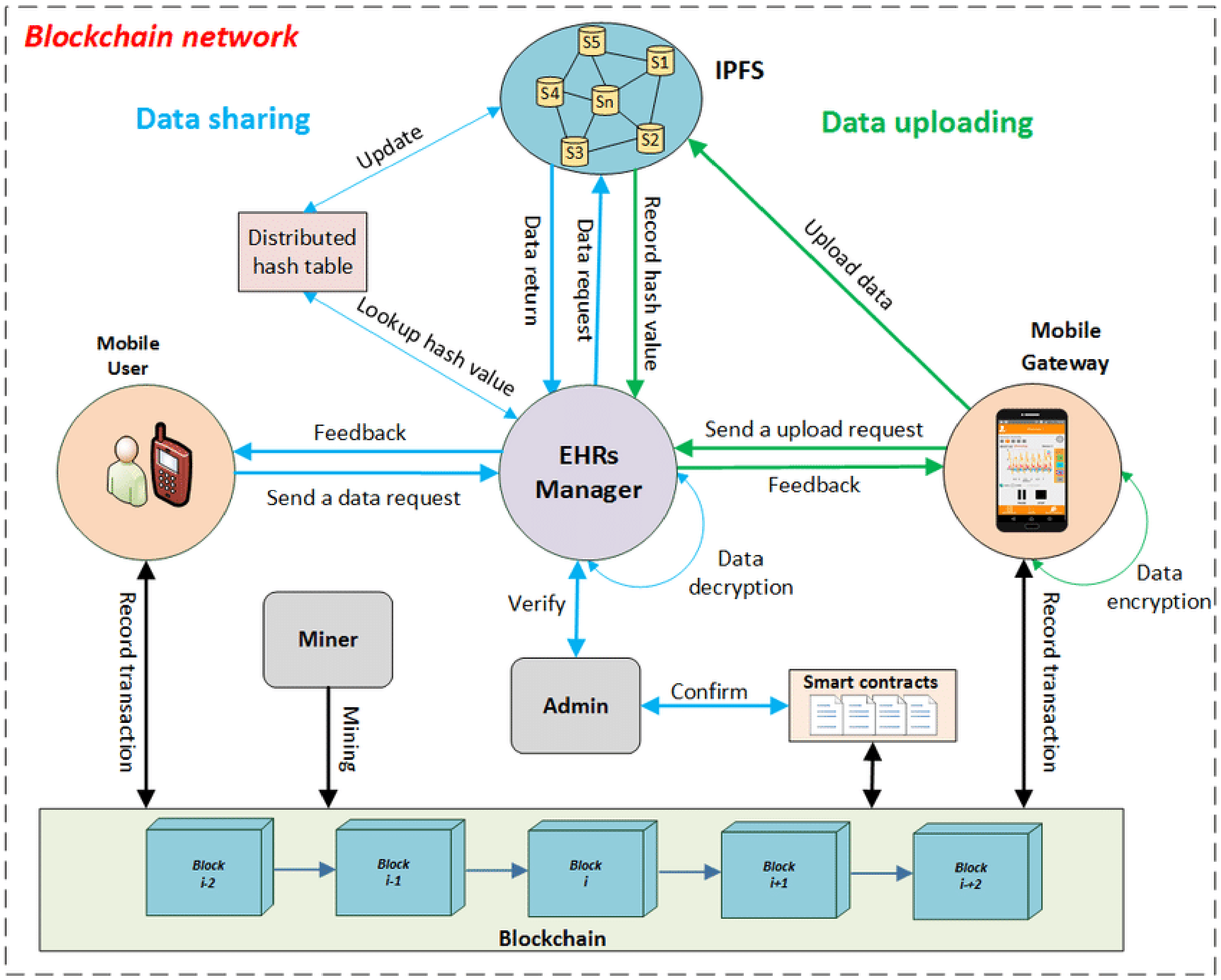
For example, if a blockchain network is used for managing dealings in farm produce from its origin (the farm) to the end customer (the market), the process involves multiple entities. Say a farmer cultivates a medicinal plant that he ships to multiple markets across the globe. In this case, multiple parties—the customs authorities who clear the produce to enter their respective nation, the shipping companies who move the produce, and the warehouse operators who need to maintain the product within a specified temperature range—all serve a vital but specific function in the farmer's transaction. In this case, permissioned networks may offer the best fit.

The farmer may finalize a particular price and quantity for selling his produce to a buyer in America and another price and quantity to another buyer in Europe. The other involved entities such as the warehouse operator may not necessarily need information about the agreed prices between the farmer and his various buyers. They may simply need access to limited information—like quantity and quality specifications—to perform their function in supporting such deals.

Consortium blockchains

Multiple organizations can share the responsibilities of maintaining a blockchain. These pre-selected organizations determine who may submit transactions or access the data. A consortium blockchain is ideal for business when all participants need to be permissioned and have a shared responsibility for the blockchain.

While in the past public and private blockchains have been the most popular variety, the permissioned blockchain, which offers the middle path between the two, is seeing increasing usage as it allows for enabling limited activities even by external vendors and providers.



What Is Ethereum?

Ethereum is a blockchain platform with its own cryptocurrency, called Ether (ETH) or Ethereum, and its own programming language, called Solidity.

As a blockchain network, Ethereum is a decentralized public ledger for verifying and recording transactions. The network's users can create, publish, monetize, and use applications on the platform, and use its Ether cryptocurrency as payment. Insiders call the decentralized applications on the network "dApps."

As a cryptocurrency, Ethereum is second in market value only to Bitcoin, as of May 2021.
Ethereum is an open-source blockchain-based platform that creates and shares business, financial services, and entertainment applications.

Ethereum users pay fees to use dApps. The fees are called "gas" because they vary depending on the amount of computational power required.

Ethereum has its own associated cryptocurrency, Ether or ETH.

Its cryptocurrency is now second only to Bitcoin in market value.

Ethereum's aim is to create an alternate protocol for constructing decentralized applications. These applications provision distinct tradeoffs which are very effective for extensive division of decentralized applications. Specific importance is given on conditions where applications require rapid development time, efficient interaction, and security for moderate and seldom utilized applications

Understanding Ethereum

Ethereum was created to enable developers to build and publish smart contracts and distributed applications (dApps) that can be used without the risks of downtime, fraud, or interference from a third party.

Ethereum describes itself as "the world's programmable blockchain." It distinguishes itself from Bitcoin as a programmable network that serves as a marketplace for financial services, games, and apps, all of which can be paid for in Ether cryptocurrency and are safe from fraud, theft, or censorship.

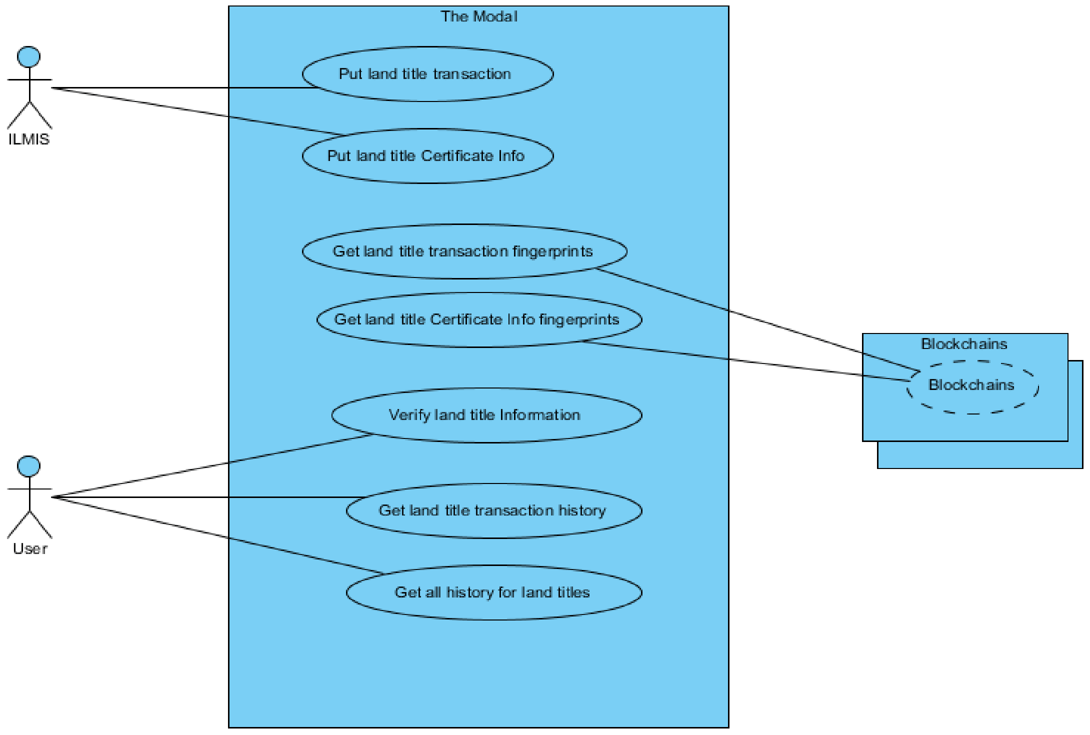
Ethereum's Founders

Ethereum was launched in July 2015 by a small group of blockchain enthusiasts. They included Joe Lubin, founder of ConsenSys, a blockchain applications developer that uses the Ethereum network. Another co-founder, Vitalik Buterin, is credited with originating the Ethereum concept and now serves as its CEO and public face. Buterin is sometimes described as the world's youngest crypto billionaire. (He was born in 1994.)

The Ether cryptocurrency was designed to be used within the Ethereum network. However, like Bitcoin, Ether is now an accepted form of payment by some merchants and service vendors. Overstock, Shopify, and CheapAir are among the online sites that accept Ether as payment.

Ethereum Classic

Ethereum Classic (ETC) is an open-source, blockchain-based distributed cryptocurrency platform that runs smart contracts. It has emerged as a split version of the Ethereum's blockchain. The split has occurred on Ethereum in 2016 when $50 million worth of funds were stolen. This resulted in the two versions existing simultaneously. The newer one was called as ETH, and the older one was renamed as ETC [46]. As mentioned in the facts of the report [47], ETC had a market cap of $745 million and a per token value of $6.41. The icon represents to ETC cryptocurrency.


The Modal
ILMIS
Put land title transaction
Put land title Certificate Info
Get land title transaction fingerprints
Get land title Certificate Info fingerprints
Blockchains
Blockchains
Verify land title Information
User
Get land title transaction history
Get all history for land titles

What Is Ethereum in Simple Terms?

Ethereum, like any blockchain, is a database of information that is designed to be unhackable. Ether, or ETH, is the cryptocurrency used to complete transactions on the blockchain.

Unlike in a traditional database, information in a blockchain is organized as a chronological "chain" made up of "blocks" of data. For instance, every transaction using an Ether coin must be verified and recorded as an additional block on that coin's unique blockchain. This process of recording every transaction in a sequence is the reason that a blockchain is often compared to a ledger.

The Ethereum blockchain stores more than transaction records for Ether currency. It allows software developers to create games and business applications, called dApps, and market them to users. Those users want to take advantage of the relative lack of risks that come with storing sensitive information on the World Wide Web.

Is Ethereum Better Than Bitcoin?

Unlike the Bitcoin blockchain, the Ethereum blockchain was not created to support a cryptocurrency. The Ether cryptocurrency was created to provide an in-house currency for applications built on the Ethereum blockchain.

In other words, Ethereum has wider ambitions. It wants to be a platform for all kinds of applications that can store information safely.

Despite their differences, the two are the creators of virtual currencies that have become rivals in the investing world. And virtual currencies are just that: They are coins that have no physical existence but are represented by a string of codes that can be exchanged at a price agreed upon by a buyer and a seller.

Implementations of DAO

This section introduces a prospective solution employing blockchain Ethereum, [28,29]) which incorporates a Turing complete programming language with smart contract computing functionality. A solution is elaborated that permits the formation of organizations where participants preserve straight real-time check of contributed collects and governance policies are formalized, automatized and imposed using software.

Distinctly, basic code for smart contract [30,31] is composed to make a Decentralized Autonomous Organization (DAO) on the Ethereum blockchain. Next, we explain the working of DAOs code, centering on fundamental establishment and governance characteristics, which includes organization, formation and voting rights.

Ethereum blockchain constructs a conceptual structure layer with a built-in Turing-complete programming language. This layer permits any person to write smart contracts and decentralized applications where they can produce their own discretional laws for ownership, transaction formats and state transition functions [24].

(a)Ethereum accounts

In Ethereum, the state consists of objects known as “accounts,” where every account has an address of 20-byte size and state transitions accounting for straight transfers of value and information between accounts.

An Ethereum account comprises of four values:

▪A counter **nonce** to ensure every transaction is processed only once
▪Up to date account's **ether balance**
▪**Contract Code** of account
▪**Storage** of account
“Ether” is the cryptocurrency of Ethereum, used for paying transaction fees. Generally, two kinds of accounts exist, one is account which is owned externally and other is contact account. The external accounts are operated by private keys and have no code whereas the **contract accounts** are operated by contract code.

(b)Messages and transactions

“Transaction” in Ethereum denote digitally signed data package holding a message. Transactions contain [24]:

▪Receiver of the message

▪Digital signature by the sender

▪The total number of ether that need to transfer from the sender to the receiver

▪Optional data field

▪A *Startgas* value, denoting the upper limit of computational steps for executing the transaction.

▪A *Gasprice* value, denoting the fee paid by sender per computational step.

Cryptocurrencies generally consists of the first three fields. The optional data field by default has no operation. However, there is an OPCODE in the virtual machine which is used by a contract to retrieve the data.

For Ethereum's anti-denial service model, the *Startgas* and *Gasprice* fields are essential.
Every transaction adjusts number of computational step, a code requires during execution to avoid computational loss. The computation is measured in gas unit. Generally, a single computation step costs a gas. Multiple gas fees are charged for several frames of data transaction.

(c)Messages

Contracts are able to send messages to other contracts. Messages are non-serializable virtual objects that subsist in the execution environment of Ethereum. A message comprises of:

•Message sender (absolute)

•Message receiver

•The total number of ether to transfer with the message

•Optional data field

•*Startgas* value

The message is generated by a contract during execution of CALL OPCODE during code execution. Contracts have relationships with other contracts.

The gas allowance allotted by a contract utilizes the total gas consumed by all transaction and all sub-executions.

For example, if an external agent X sends a transaction to Y with 1200 gas, and Y takes 700 gas before sending a transaction to Z, and the internal execution of Z takes 300 gas before repaying, then Y can expend another 200 gas before ceasing of gas.

The state transition function for Ethereum is *EMPLOY (R, TS)* $\rightarrow$ *R′*. Function is defined as follows (Fig. 5):

1.Examine if the transaction is correct and the signature is legal, and the nonce corresponds the nonce in the sender's account. Return ERROR if the condition is not met.

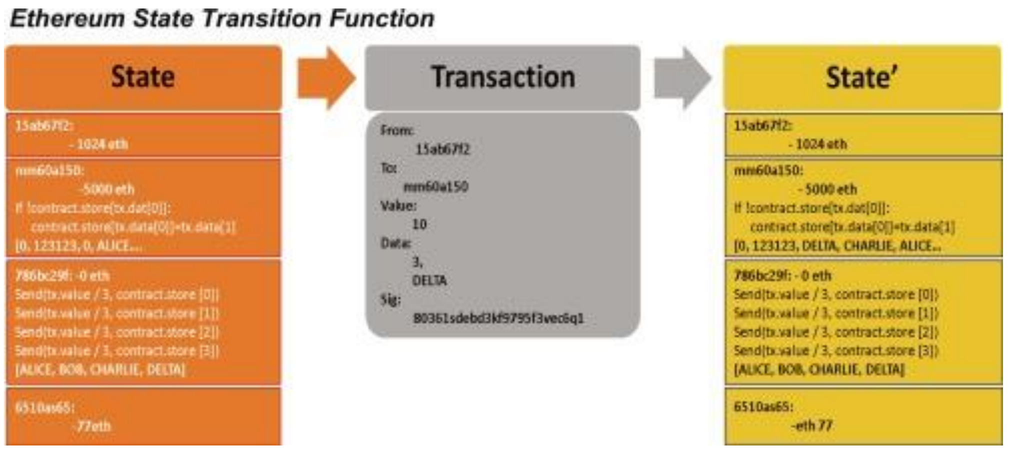
2.Compute the transaction fee as *Startgas* * *Gasprice*, and from the signature find out the broadcasting address. Deduct the fees, maintain the balance of the sender's account and increase the sender's nonce. Return an ERROR message if adequate balance is not there.

3.Assign *Gas* = *Startgas* and deduct a definite amount of gas per byte to pay for the transaction bytes.

4.Move the transaction amount from the sender's account to the receiver's account. Make the receiver's account if it is not there. If the receiver's account is a contract, execute contract's code till end or till the execution consumes the gas.

5.If the transfer value fails due to sender not having adequate balance, or the code execution consumes the gas, then roll back all state alterations excluding the fees payment and add up the fees to the miner's account.

6.Else, repay the fees for all leftover gas to the sender, and transfer the fees to the miner for gas consumed.


Ethereum State Transition Function
State
15ab67f2:
- 1024 eth
mm60a150:
-5000 eth
if !contract.store[tx.dat[0]]:
contract.store[tx.data[0]]=tx.data[1]
[0, 123123, 0, ALICE...
786bc29f: -0 eth
Send(tx.value / 3, contract.store [0])
Send(tx.value / 3, contract.store [1])
Send(tx.value / 3, contract.store [2])
Send(tx.value / 3, contract.store [3])
[ALICE, BOB, CHARLIE, DELTA]
6510as65:
-77eth
Transaction
From:
15ab67f2
To:
mm60a150
Value:
10
Data:
3,
DELTA
Sig:
80361sdebd3kf9795f3ved6q1
State'
15ab67f2:
- 1024 eth
mm60a150:
- 5000 eth
if !contract.store[tx.dat[0]]:
contract.store[tx.data[0]]=tx.data[1]
[0, 123123, DELTA, CHARLIE, ALICE...
786bc29f: - 0 eth
Send(tx.value / 3, contract.store [0])
Send(tx.value / 3, contract.store [1])
Send(tx.value / 3, contract.store [2])
Send(tx.value / 3, contract.store [3])
[ALICE, BOB, CHARLIE, DELTA]
6510as65:
-eth 77

For example, assume that the contract's code is:

if!auto.store[assigndatapayload(0)]:

auto.store[assigndatapayload(0)] = assigndatapayload (64).

This is high-level language Serpent code and this code can be compiled to EVM code. The contract code is scripted in the low-level EVM code.

Assume the contract's store kickoffs vacant, and a transaction is transmitted with 20 ether value, 3000 gas, 0.001 ether *Gasprice*, and 64 bytes of data, with bytes 0–31 expressing the number 2, bytes 32–63 expressing the string CHARLIE and bytes 64–127 expressing the string DELTA. The state transition function process in this situation is described as follows:

▪Examine that the transaction is legal and correct.

▪Examine that the transaction sender has at least 3000 * 0.001 = 3 ether. If true, then deduct three ethers from the sender's account.

▪Assign gas = 3000; supposing the transaction is 200 bytes of length and the byte-fee is 6, deduct 1200 so remaining left is 1800 gas.

▪Deduct 20 more ether from the sender's account and add up to the contract's account.

▪Execute the code. The code examines if the contract's storage at index 3 is occupied, finds if it is not occupied, then it fixes the storage at index 3 to the value DELTA. Assume it takes 287 gas, so the remaining left amount of gas is 1800 − 287 = 1513.

▪Add 1513 * 0.001 = 1.513 ether back to the sender's account and return the resulting state.

The total transaction fee is equal to the supplied *Gasprice* multiplied by the transaction length in bytes, in case if there is no contract at the receiver's side of the transaction.

The working of messages is equivalent to transactions with regard to returns. If some message execution ceases the gas, then execution of that message and other messages activated by that message returns, but parent executions don't require returning. It is harmless for one contract to call another contract. If X calls Y with G gas, then X's execution is assured to consume at most G gas. CREATE OPCODE creates a contract and its working is similar to CALL.

(d)Code execution

Ethereum Virtual Machine Code or EVM code is a low level, stack-based bytecode language. Ethereum contracts are written in EVM code. Every byte in the code expresses a function.

When the code executes, it enters in an infinite loop that consists of several times execution of the function at the current program counter. The program counter starts at zero and increment by one till the code is terminated or in case of an error or if instructions like STOP or RETURN is noticed [32]. The functions can retrieve to three kinds of space to store data:

▪**Stack**, LIFO (last-in-first-out) data structure in which data can be pushed and popped

▪**Memory**, an array of bytes

▪The contract's long-run reset key and value **storage**.

The conventional execution model of EVM code is amazingly simple. EVM is determined by a tuple having eight attributes such as tuple (block_state, transaction, message, code, memory, stack, pc, gas), where block_state is the state containing all accounts globally and permits balances and storage.

In each round of execution, at the PCth byte of code we find the current instruction. For example, the ADD operation in stack pops two data values from the stack and pushes their sum in the stack. Further, the operation reduces gas by 1 and the PC is incremented by 1. The STORE operation pops two top data values from the stack and appends the second data value at the contract's storage at the address mentioned in the first data value in the stack.

(e)Blockchain and Mining

In many ways the Ethereum blockchain is alike the Bitcoin blockchain, but still there are few dissimilarities. With respect to the architecture of Blockchain, the blocks in Ethereum comprise a copy of transaction list and the latest state whereas Bitcoin only contains a copy of the transaction list .

The fundamental block validation algorithm in Ethereum is as follows:

▪Examine that the previous block referenced is legal and is present.

▪Examine that the timestamp of the block is more than the previously referenced block and less than 15 min into the future.

▪Examine the legality of the block number, difficultness, transaction root, uncle root and gas limit.

▪Examine that the Proof of Work (PoW) is legal on the block.

▪Let state is S [0] at the end of the previous block.

▪Let the block transaction list is TL, having n transactions. For all i in 0 … n − 1, set S [i + 1] = APPLY(S[i], TL[i]). Return an ERROR if total gas spent in the block overreach the GASLIMIT or the application returns an error.

▪Let the final state be S[n], after adding the block incentive point paid to the miner.

▪To check the validity of the block, examine if the Merkle tree root of the final state S_FINAL is equal to the final state root provided in the block header otherwise, it is not valid.

As the complete state is required to store in each block, the approach may appear extremely ineffective at first glimpse. The efficiency of the approach ought to be Conforming to that of Bitcoin.

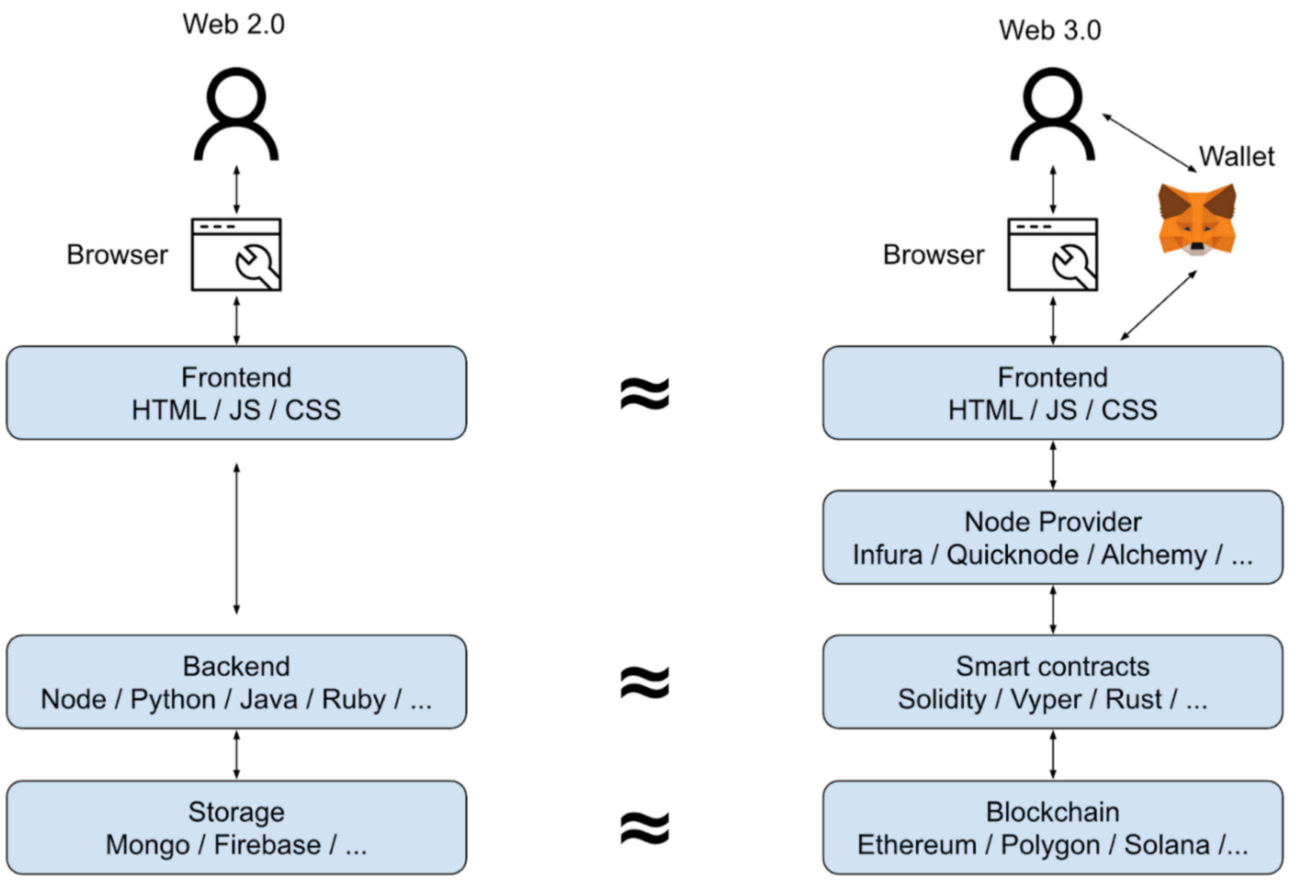
This is because the state is stored in tree structure, and only a small portion of tree requires modification. Therefore, the tree structure are quite alike for two neighboring blocks and data can be stored once and referenced twice using pointers. To do so, a “Patricia tree” is used, considering an adjustment to the Merkle tree conception which allows for not only effective change of nodes but also insertion and deletion of nodes.

Ethereum smart contracts using solidity

For Ethereum smart contracts, solidity and serpent are the two primary languages that can be used and is describes as follows.

• Solidity is a contract oriented high-level language with syntax similar to the javascript and is designed to target EVM.

•Serpern is a high-level language similar to python to write Ethereum contracts. But, solidity is the preferred language for the development of Ethereum-based applications.



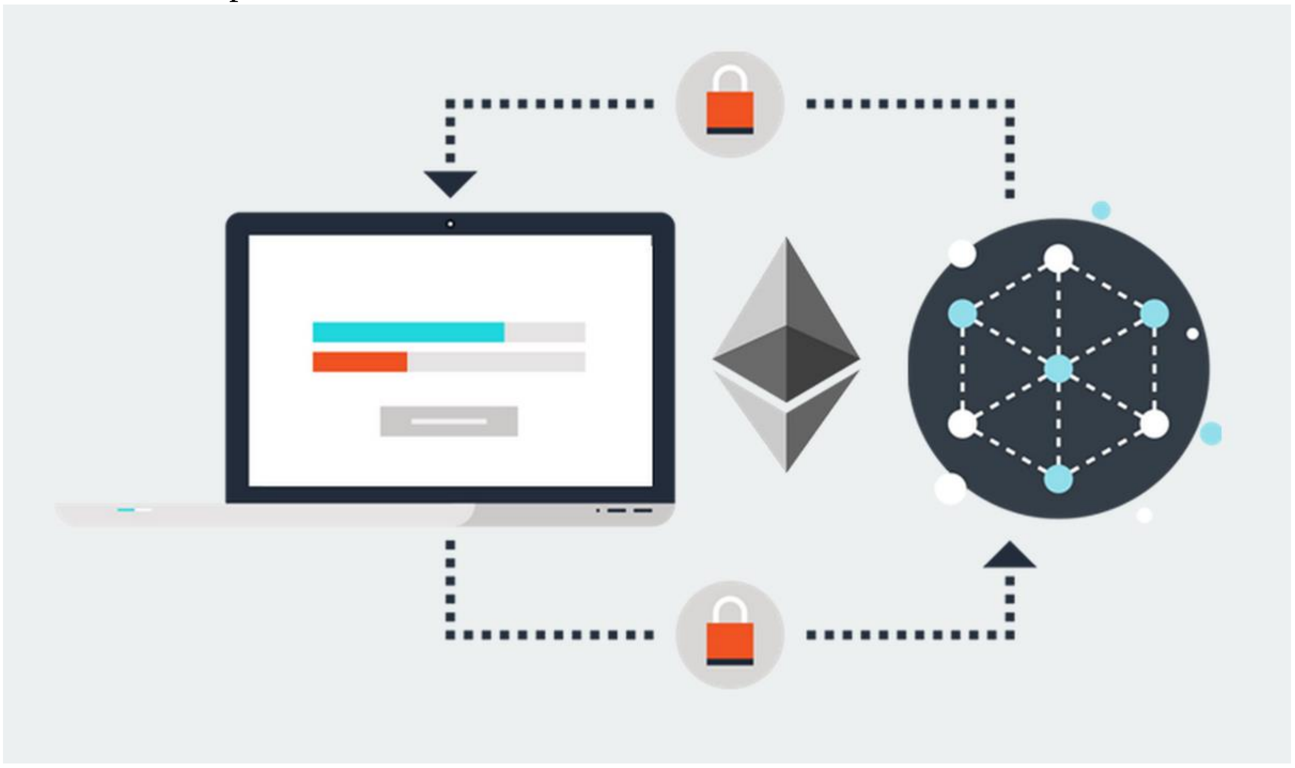
4.1 Defining a smart contract using solidity

Here, we introduce solidity for writing smart contracts. The basic steps followed for writing the smart contract using solidity are as follows.

1.*Build a smart contract*: A simple purchase order smart contract developed with the help of solidity.

2.*Structure of smart contract*: The smart contract is a collection of two main entities, i.e., data and function. Data maintains the current state of the contract and function is a logic that applies to the transition state of the contract. Each smart contract follows a standard structure. It begins with the following statements.

•*Pragama directive*: The *pragama* keyword is used to enable certain features of the compiler as shown in Listing 1. The compiler does not compile the smart contract statements with a compiler version not earlier than 0.4.0 and not after the compiler version 0.6.0. This statement does not introduce any unintended behavior when a new version of compiler launched.



Listing 1

Pragama directive

```
pragma solidity >=0.4.0 <=0.6.0
```

•**Contract declaration:** The contract declaration in the smart contract declared by using the keyword *contract*. Listing 2 declares an empty contract and identified by *Purchase order*.

Listing 2

Contract declaration

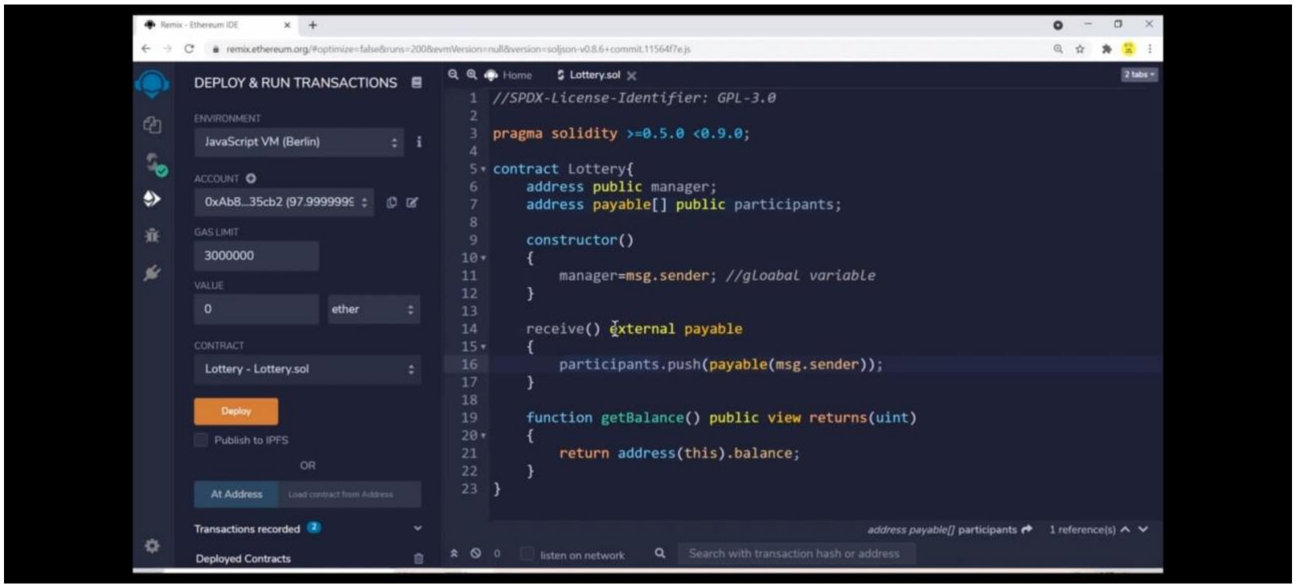
```
contract Purchase Order
{
}
```

•*Store relevant data to the contract*:

Every contract may require some data to store in smart contracts. Storing data provides a level of flexibility. The moving away from hard-coded values of data to user-provided values is an important feature. The variable used in the smart contract allows to store, label, retrieve, and manipulate the data. The variables used in the smart contract are of two types and described as follows.

(a) *Value type*: These types of variables are passed by value, i.e., they copied at the time of function arguments. For example, integers and booleans.
(b) *Reference type*: These type of variables are passed by reference. These have to be managed and controlled carefully at these are not fit into 256-bit.

•*Add data to the smart contract*: Here, add some data to the smart contract. We introduce a variable *Product_quantity* of unsigned integer of storage 256-bits and is referred as *uint256*. Its minimum value can be assigned as 0 and maximum value as 2^{256-1}. It is shown in Listing 3.


Remix - Ethereum IDE
remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.6+commit.11564f7e.js
DEPLOY & RUN TRANSACTIONS
Home
Lottery.sol
2 tabs
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 2
Deployed Contracts
address payable[] participants
1 reference(s)
listen on network
Search with transaction hash or address

ENVIRONMENT: JavaScript VM (Berlin)
ACCOUNT: 0xAb8...35cb2 (97.9999999
GAS LIMIT: 3000000
VALUE: 0 ether
CONTRACT: Lottery - Lottery.sol

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.5.0 <0.9.0;

contract Lottery{
    address public manager;
    address payable[] public participants;

    constructor()
    {
        manager=msg.sender; //gloabal variable
    }

    receive() external payable
    {
        participants.push(payable(msg.sender));
    }

    function getBalance() public view returns(uint)
    {
        return address(this).balance;
    }
}
```

Listing 3

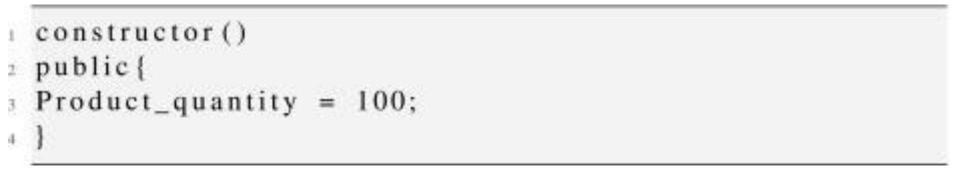
Add data to the smart contract

```
contract Purchase Order
{
uint256 Product_quantity;
}
```

•*Define the constructor*: The constructor is called when the contract is used. It initializes with some integer values as shown in Listing 4.

Listing 4

Define the constructor

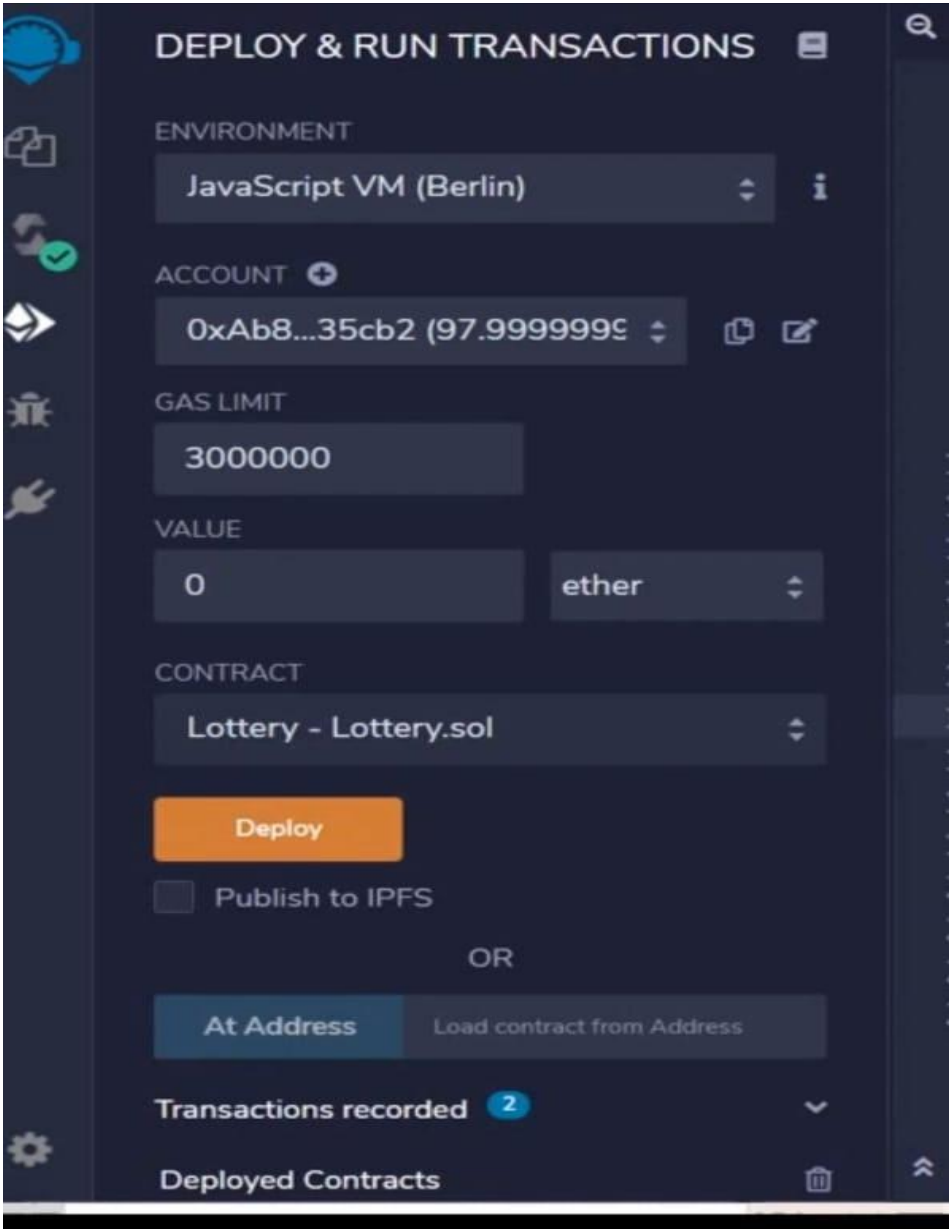


```
constructor()
public{
Product_quantity = 100;
}
```

In this Listing 4, the *public*keyword refers that anyone can access this information, it is not a restricted function.

•*Add functions to the smart contract*: The addition of function makes the contract more interactive. The function can be declared as *function <function_name><access_modified><state_mutator><return_value>*.

(a) Get function: To read the function, read_function or get_function is added in the contract is as shown in Listing 5.



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

JavaScript VM (Berlin)

ACCOUNT

0xAb8...35cb2 (97.9999999

GAS LIMIT

3000000

VALUE

0 ether

CONTRACT

Lottery - Lottery.sol

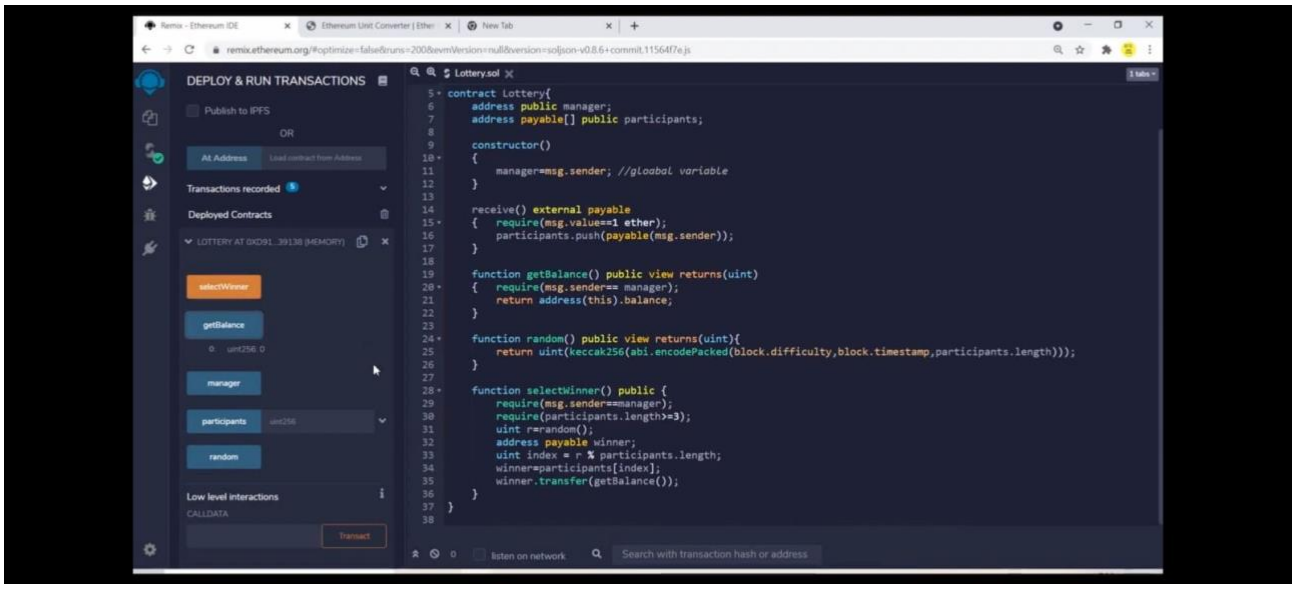
Listing 5

Get function

```
function get_quantity()
public view returns(uint256)
{
return Product_quantity;
}
```

Here, <function_name> is get_quantity() (where no arguments are passed), <access_modified> is public (anyone can access the function), <state_mutator> is view (signifies only read the contract, does not change the state of the contract), and <return_value> is returns(uint256).

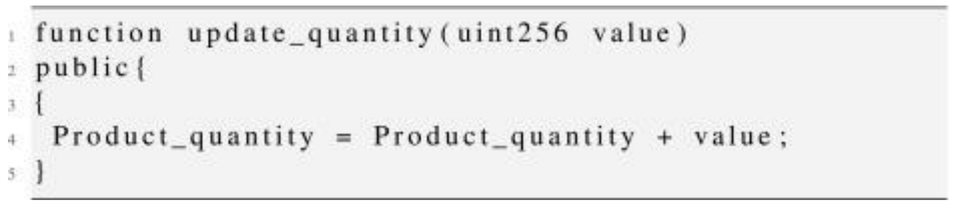
(a) Set function: Read the data is necessary but if requires to update or to write the data as well. Then, use the set_function in the contract that takes value from the user as input parameter and update the variable with that value as shown in Listing 6.



```
contract Lottery{
    address public manager;
    address payable[] public participants;

    constructor()
    {
        manager=msg.sender; //gloabal variable
    }

    receive() external payable
    {   require(msg.value==1 ether);
        participants.push(payable(msg.sender));
    }

    function getBalance() public view returns(uint)
    {   require(msg.sender== manager);
        return address(this).balance;
    }

    function random() public view returns(uint){
        return uint(keccak256(abi.encodePacked(block.difficulty,block.timestamp,participants.length)));
    }

    function selectWinner() public {
        require(msg.sender==manager);
        require(participants.length>=3);
        uint r=random();
        address payable winner;
        uint index = r % participants.length;
        winner=participants[index];
        winner.transfer(getBalance());
    }
}
```

Listing 6

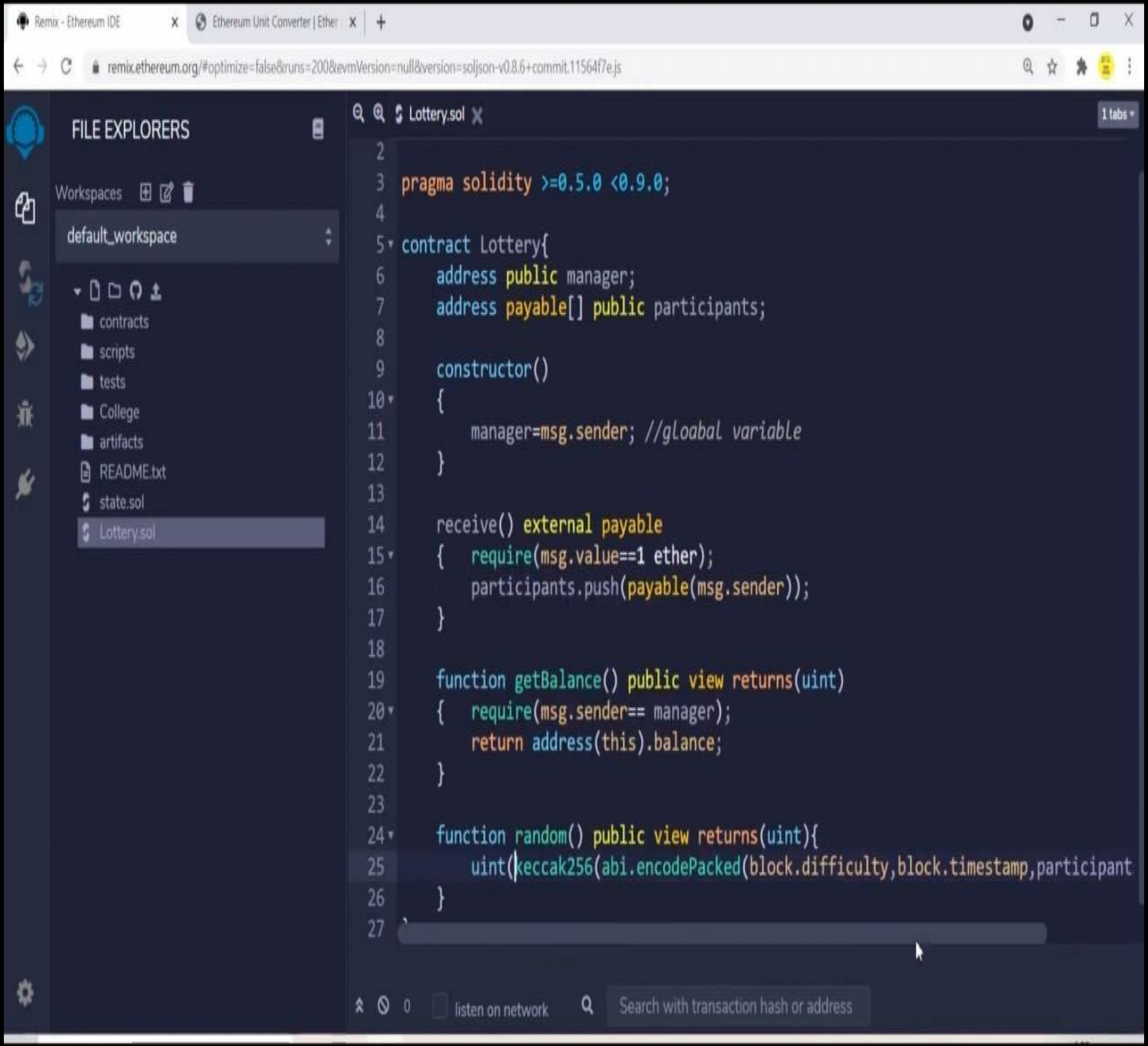
Set function



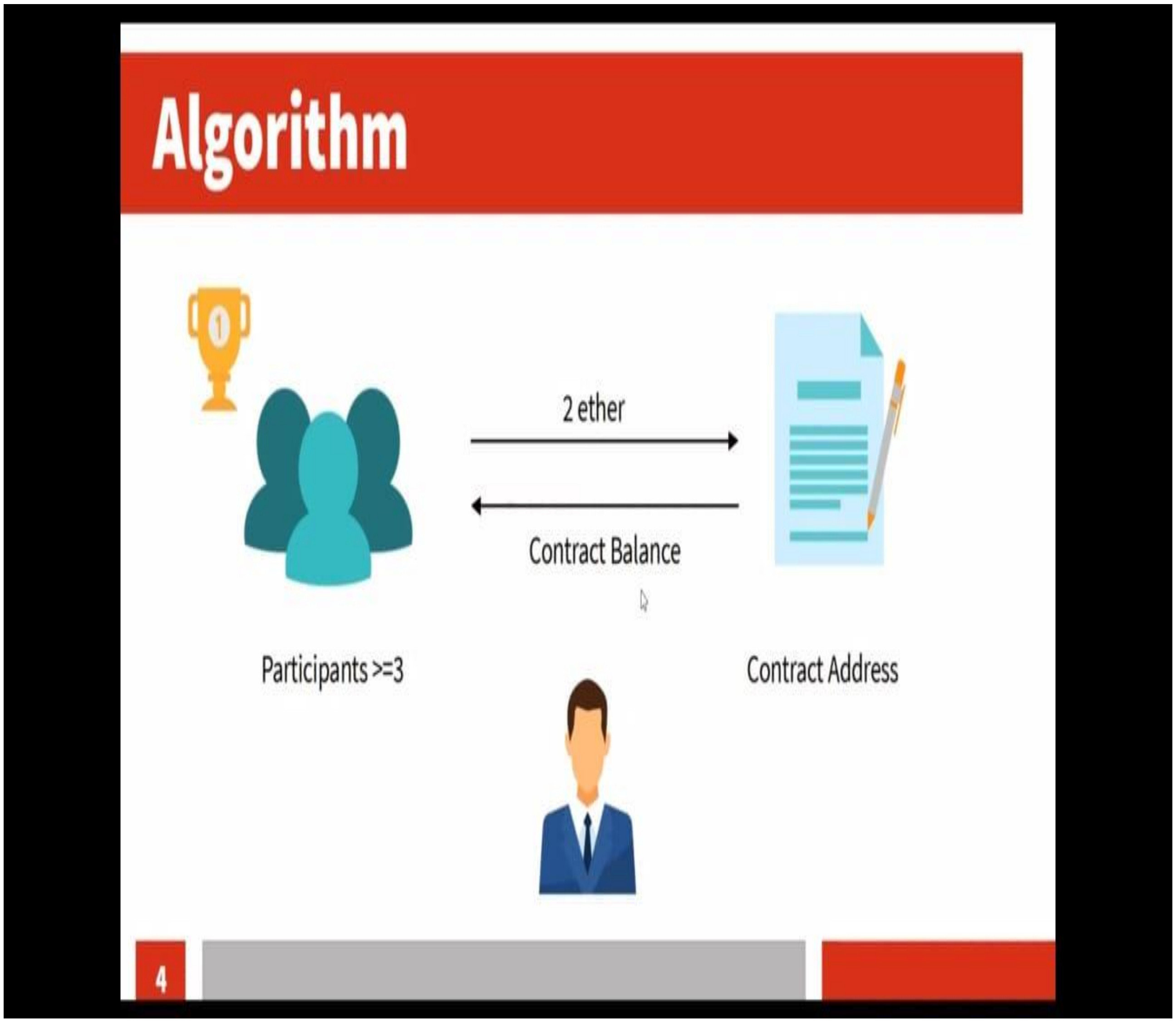
```
function update_quantity(uint256 value)
public{
{
 Product_quantity = Product_quantity + value;
}
```

Here, <function_name> is update_quantity(uint256 value), <access_modified> is public, <state_mutator> is not required, and <return_value> is returns a variable to type uint256.

3.*Defining the smart contract*: Combine all foreknown steps to make the smart contract. Listing 7 defines the purchase order smart contract.



```
pragma solidity >=0.5.0 <0.9.0;

contract Lottery{
    address public manager;
    address payable[] public participants;

    constructor()
    {
        manager=msg.sender; //gloabal variable
    }

    receive() external payable
    {   require(msg.value==1 ether);
        participants.push(payable(msg.sender));
    }

    function getBalance() public view returns(uint)
    {   require(msg.sender== manager);
        return address(this).balance;
    }

    function random() public view returns(uint){
        uint(keccak256(abi.encodePacked(block.difficulty,block.timestamp,participant
    }
```



Algorithm

Listing 7

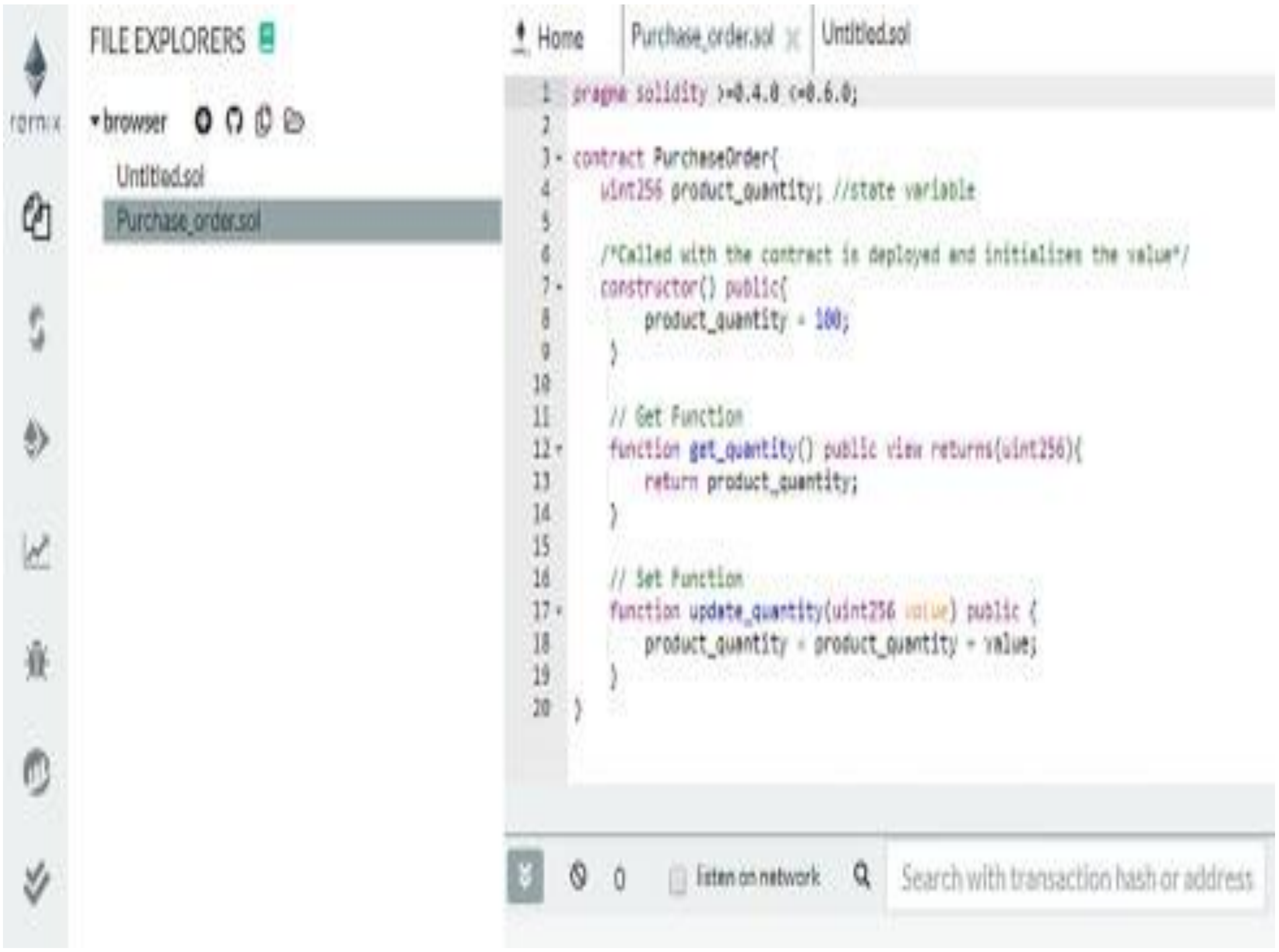
Defining the smart contract

```
pragma solidity >=0.4.0 <=0.6.0;

contract PurchaseOrder{
uint256 product_quantity; //state variable

/*Called with the contract is deployed and initializes
   the value*/
constructor() public{
product_quantity = 100;
}

// Get Function
function get_quantity() public view returns(uint256){
return product_quantity;
}

// Set Function
function update_quantity(uint256 value) public {
product_quantity = product_quantity + value;
}
}

```

4.2 Deploy and run smart contract

Run the program shown in Listing 7 at remix online integrated development environment (IDE) [2]. It is a toolset to start the development of a smart contract and to test the smart contract without any installation on a local computer.

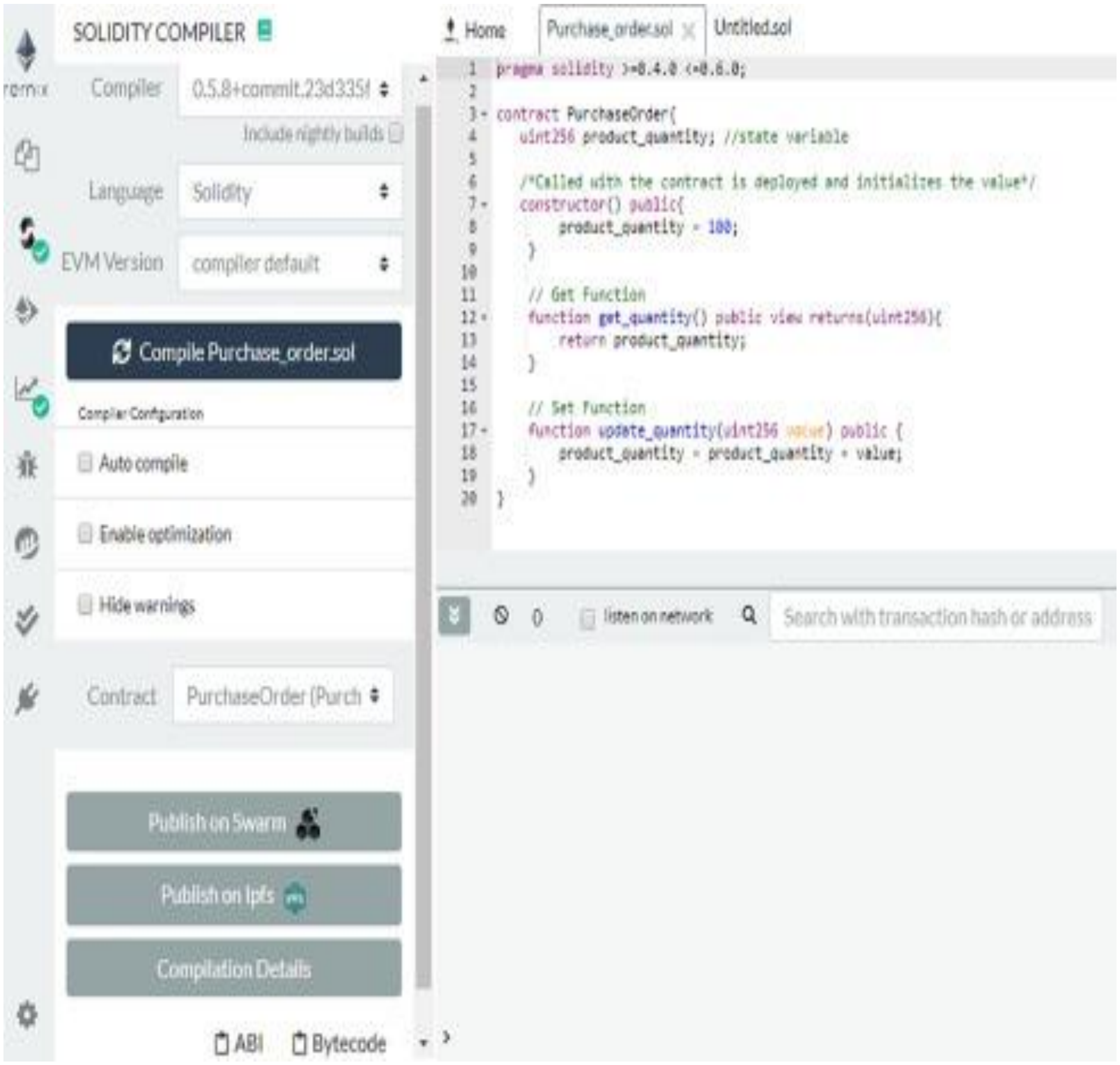
The following steps are used to deploy and run smart contracts at remix online IDE.
1.Click on the plus icon to add a new file named “Purchase_order.sol”.
2.Write the code of the contract in Purchase_order.sol as shown in Fig. 11.



```
pragma solidity >=0.4.0 <=0.6.0;

contract PurchaseOrder{
    uint256 product_quantity; //state variable

    /*Called with the contract is deployed and initializes the value*/
    constructor() public{
        product_quantity = 100;
     }

     // Get Function
     function get_quantity() public view returns(uint256){
         return product_quantity;
     }

     // Set Function
     function update_quantity(uint256 value) public {
         product_quantity = product_quantity + value;
     }
}
```

3.Click on the icon "Solidity compiler" present on the left side to compute the contract. Under this, set the compiler version is 0.5.8 and then, compile the Purchase_order.sol as shown.

Once the smart contract compilation is done successful. The, click on the compilation details that provides two key formation, i.e., application binary interface (ABI) that details all the methods explored in the contract and bytecode is EVM operation code that converts smart logic gets into bytecode on compilation.



Compiler: 0.5.8+commit.23d335f

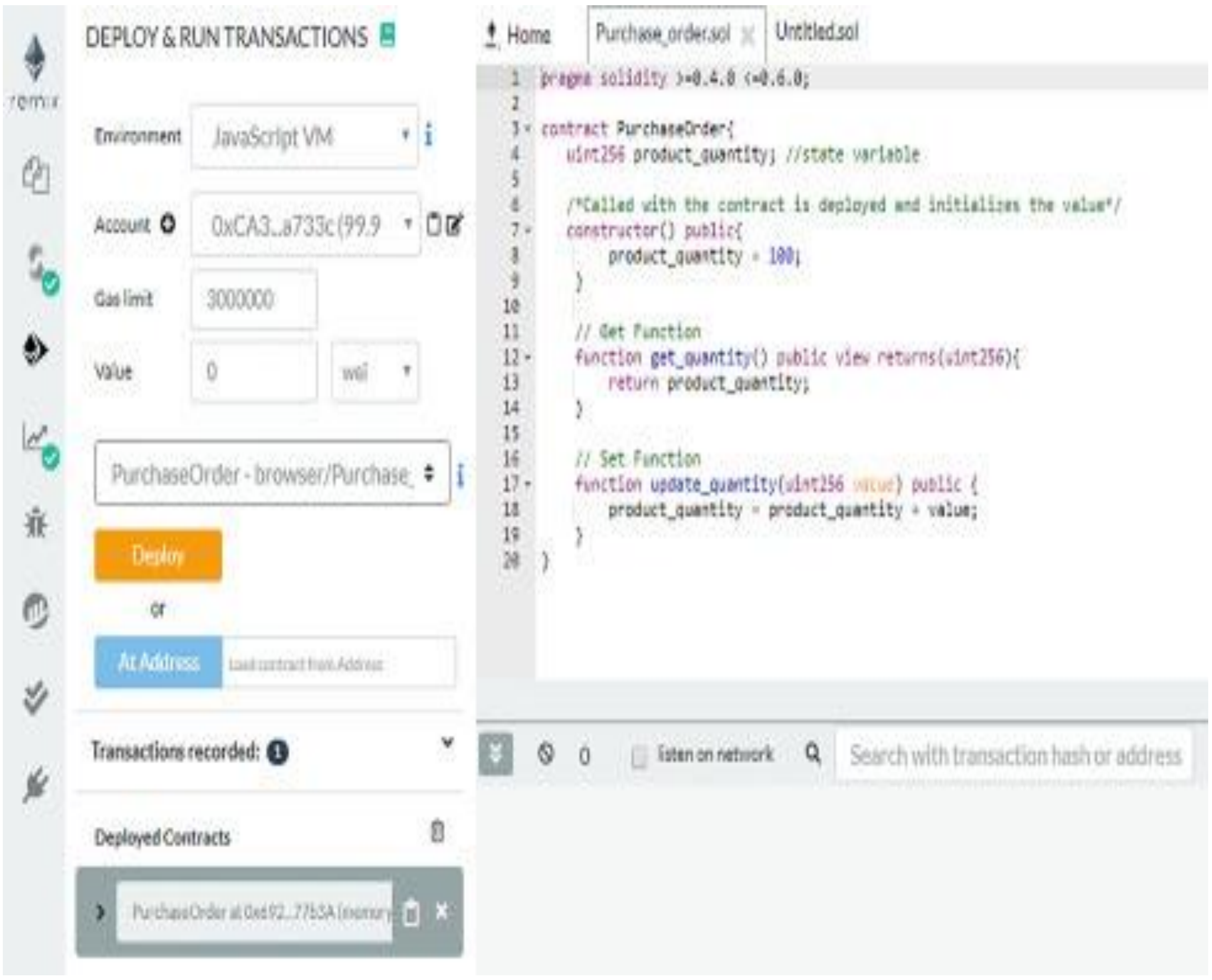
Language: Solidity

EVM Version: compiler default

Contract: PurchaseOrder (Purch

```
pragma solidity >=0.4.0 <=0.6.0;

contract PurchaseOrder{
    uint256 product_quantity; //state variable

    /*Called with the contract is deployed and initializes the value*/
    constructor() public{
        product_quantity = 100;
     }

     // Get Function
     function get_quantity() public view returns(uint256){
         return product_quantity;
     }

     // Set Function
     function update_quantity(uint256 value) public {
         product_quantity = product_quantity + value;
     }
}
```

4.In order to test the smart contract, click on the “deploy and run transactions” icon on the left side. It provides the deployment option environment (JavaScript virtual machine, injected web3 provide, web3 provider), accounts (based on environment selection), gas limit (maximum amount to spend on transactions), and value (required to send across while using smart contract) as shown.



DEPLOY & RUN TRANSACTIONS

Environment: JavaScript VM

Account: 0xCA3...a733c (99.9

Gas limit: 3000000

Value: 0 wei

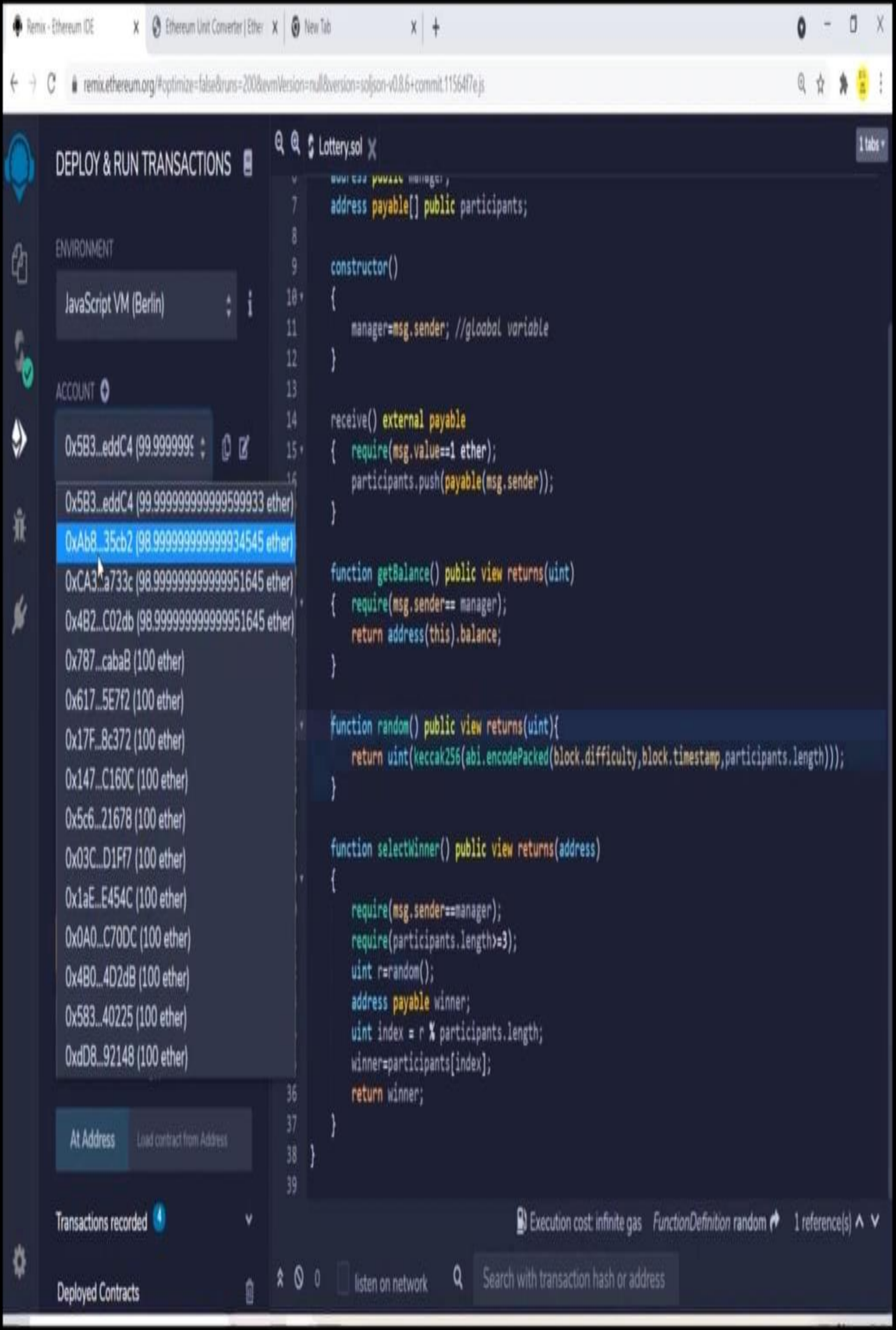
PurchaseOrder - browser/Purchase_

Transactions recorded: 1

Deployed Contracts

PurchaseOrder at 0xd92...7753A (memory)

```
pragma solidity >=0.4.0 <=0.6.0;

contract PurchaseOrder{
   uint256 product_quantity; //state variable

   /*Called with the contract is deployed and initializes the value*/
   constructor() public{
       product_quantity = 100;
    }

    // Get Function
    function get_quantity() public view returns(uint256){
       return product_quantity;
    }

    // Set Function
    function update_quantity(uint256 value) public {
       product_quantity = product_quantity + value;
    }
}
```



Remix - Ethereum IDE
Ethereum Unit Converter | Ether
New Tab
remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.6+commit.11564f7e.js
DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
ACCOUNT
Lottery.sol
1 tabs
At Address
Load contract from Address
Transactions recorded 4
Deployed Contracts
listen on network
Search with transaction hash or address

Environment: JavaScript VM (Berlin)

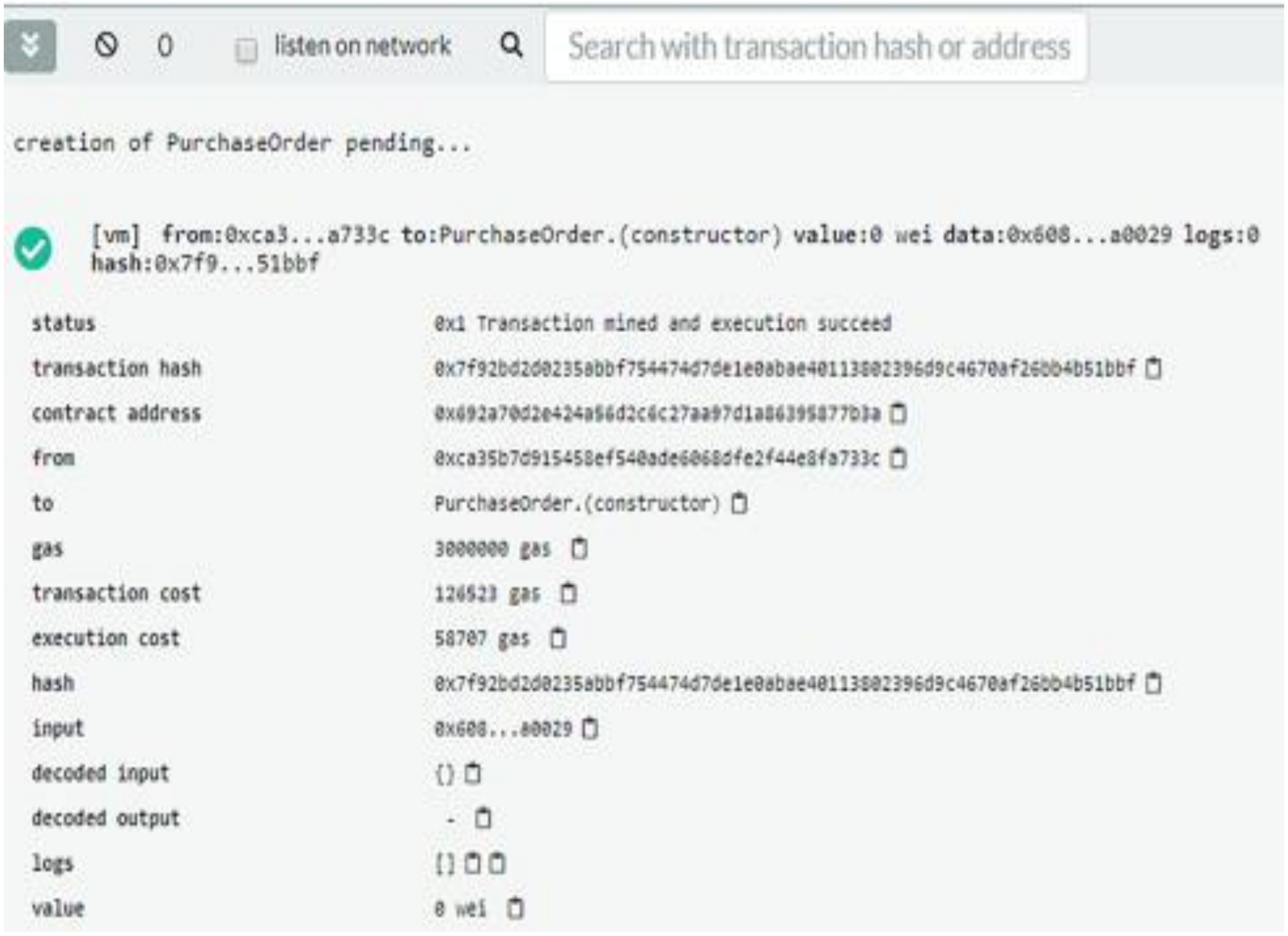
Account: 0x5B3...eddC4 (99.999999

```
0x5B3...eddC4 (99.99999999999599933 ether)
0xAb8...35cb2 (98.99999999999934545 ether)
0xCA3...a733c (98.99999999999951645 ether)
0x4B2...C02db (98.99999999999951645 ether)
0x787...cabaB (100 ether)
0x617...5E7f2 (100 ether)
0x17F...8c372 (100 ether)
0x147...C160C (100 ether)
0x5c6...21678 (100 ether)
0x03C...D1Ff7 (100 ether)
0x1aE...E454C (100 ether)
0x0A0...C70DC (100 ether)
0x4B0...4D2dB (100 ether)
0x583...40225 (100 ether)
0xdD8...92148 (100 ether)
```

```solidity
    address public manager;
    address payable[] public participants;

    constructor()
    {
        manager=msg.sender; //global variable
    }

    receive() external payable
    {   require(msg.value==1 ether);
        participants.push(payable(msg.sender));
    }

    function getBalance() public view returns(uint)
    {   require(msg.sender== manager);
        return address(this).balance;
    }

    function random() public view returns(uint){
        return uint(keccak256(abi.encodePacked(block.difficulty,block.timestamp,participants.length)));
    }

    function selectWinner() public view returns(address)
    {
        require(msg.sender==manager);
        require(participants.length>=3);
        uint r=random();
        address payable winner;
        uint index = r % participants.length;
        winner=participants[index];
        return winner;
    }
}
```

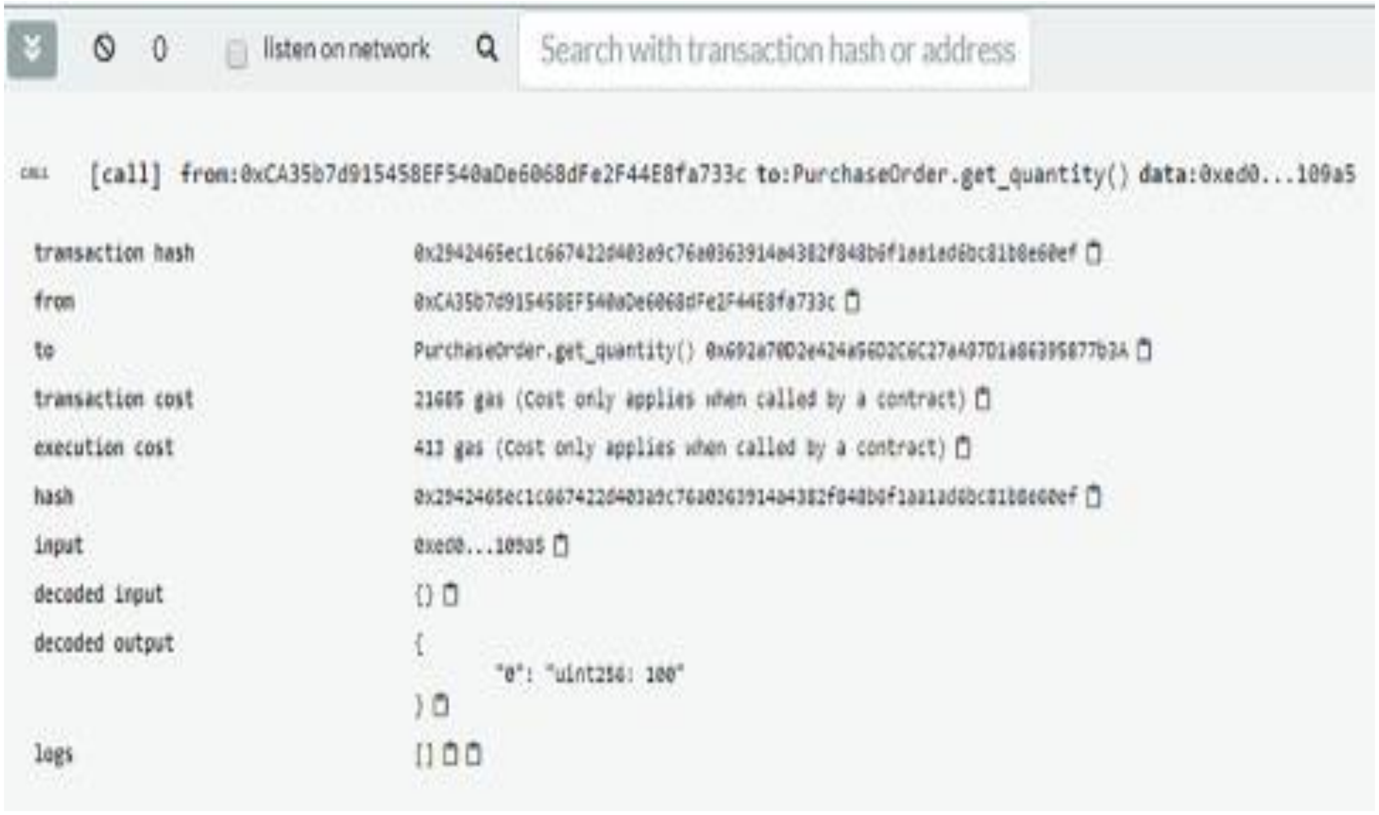
Execution cost: infinite gas   *FunctionDefinition* random   1 reference(s)

5.The results of the smart contract with transaction hash, contract address, transaction cost, and execution cost are as shown.

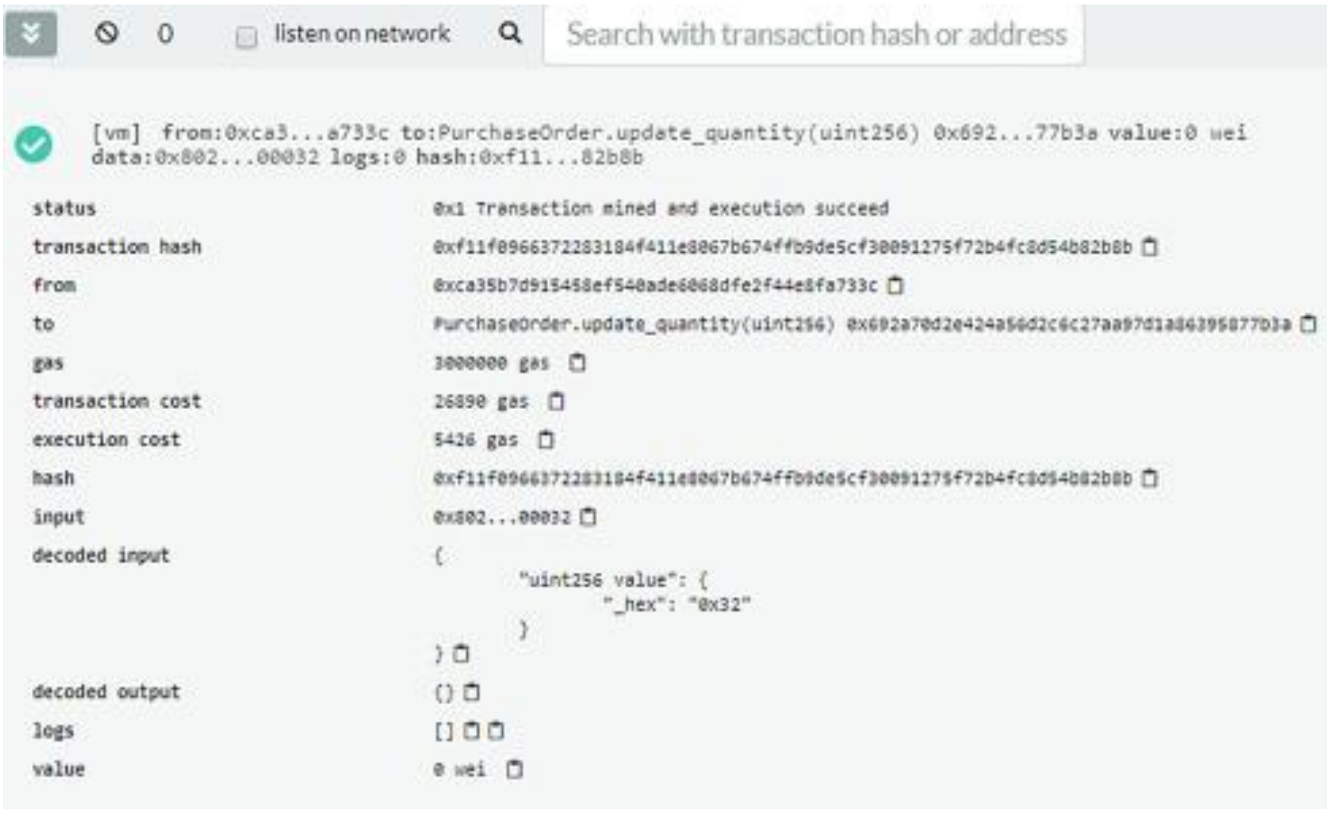


```
creation of PurchaseOrder pending...

[vm]  from:0xca3...a733c to:PurchaseOrder.(constructor) value:0 wei data:0x608...a0029 logs:0
hash:0x7f9...51bbf

status              0x1 Transaction mined and execution succeed
transaction hash    0x7f92bd2d0235abbf754474d7de1e0abae40113802396d9c4670af26bb4b51bbf
contract address    0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
from                0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to                  PurchaseOrder.(constructor)
gas                 3000000 gas
transaction cost    126523 gas
execution cost      58707 gas
hash                0x7f92bd2d0235abbf754474d7de1e0abae40113802396d9c4670af26bb4b51bbf
input               0x608...a0029
decoded input       {}
decoded output       -
logs                []
value               0 wei
```

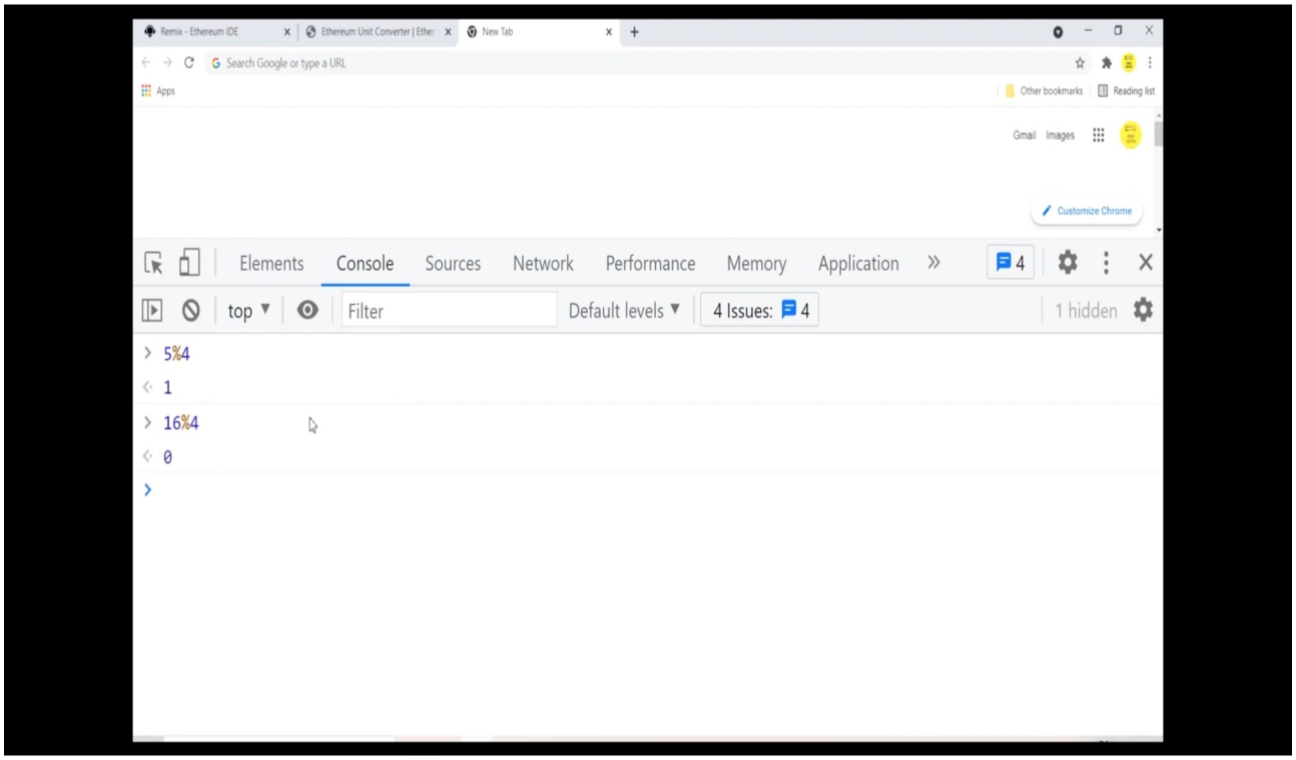
6.After this, run the set and get functions introduced in the smart contract. During get function, the "*get_quantity*" method retrieves the value 100 as shown in Fig. 15. Similarly, during get function, the "*update_quantity*" method retrieves the value (user input, 50) is as shown in Fig. 16. The set function causes a transaction to happen but get function does not.



```
CALL [call] from:0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c to:PurchaseOrder.get_quantity() data:0xed0...109a5

transaction hash     0x2942465ec1c667422d403a9c76a0363914a4382f848b6f1aa1ad6bc81b8e60ef
from                 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c
to                   PurchaseOrder.get_quantity() 0x692a70D2e424a56D2C6C27aA97D1a86395877b3A
transaction cost     21685 gas (Cost only applies when called by a contract)
execution cost       413 gas (Cost only applies when called by a contract)
hash                 0x2942465ec1c667422d403a9c76a0363914a4382f848b6f1aa1ad6bc81b8e60ef
input                0xed0...109a5
decoded input        {}
decoded output       {
                             "0": "uint256: 100"
                     }
logs                 []
```



```
[vm] from:0xca3...a733c to:PurchaseOrder.update_quantity(uint256) 0x692...77b3a value:0 wei
data:0x802...00032 logs:0 hash:0xf11...82b8b
status              0x1 Transaction mined and execution succeed
transaction hash    0xf11f0966372283184f411e8067b674ffb9de5cf30091275f72b4fc8d54b82b8b
from                0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to                  PurchaseOrder.update_quantity(uint256) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
gas                 3000000 gas
transaction cost    26890 gas
execution cost      5426 gas
hash                0xf11f0966372283184f411e8067b674ffb9de5cf30091275f72b4fc8d54b82b8b
input               0x802...00032
decoded input       {
                            "uint256 value": {
                                    "_hex": "0x32"
                            }
                    }
decoded output      {}
logs                []
value               0 wei
```



```
> 5%4
< 1
> 16%4
< 0
>
```

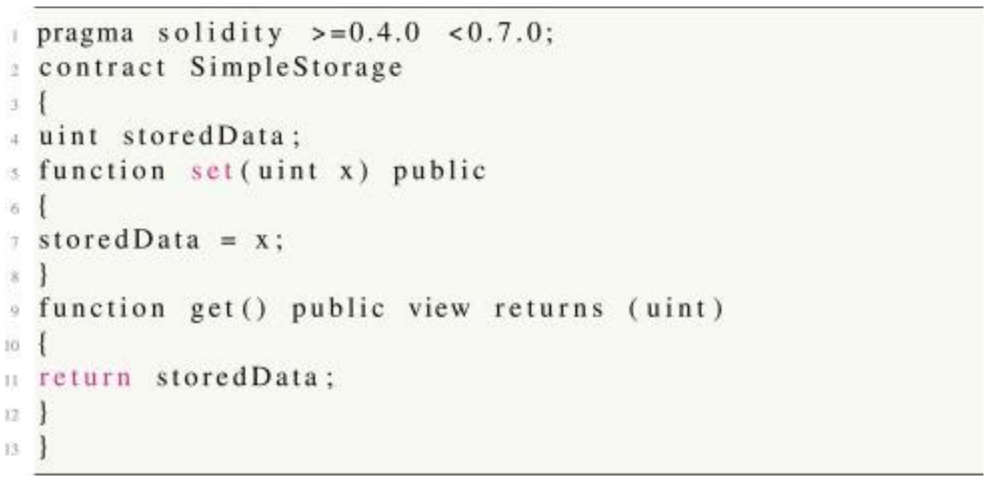
4.3 Examples of smart contract

The more examples of smart contract are as follows.

1.A simple example of smart contract for storage of data with get and set functions is as shown in Listing 8.

Listing 8

Storage example

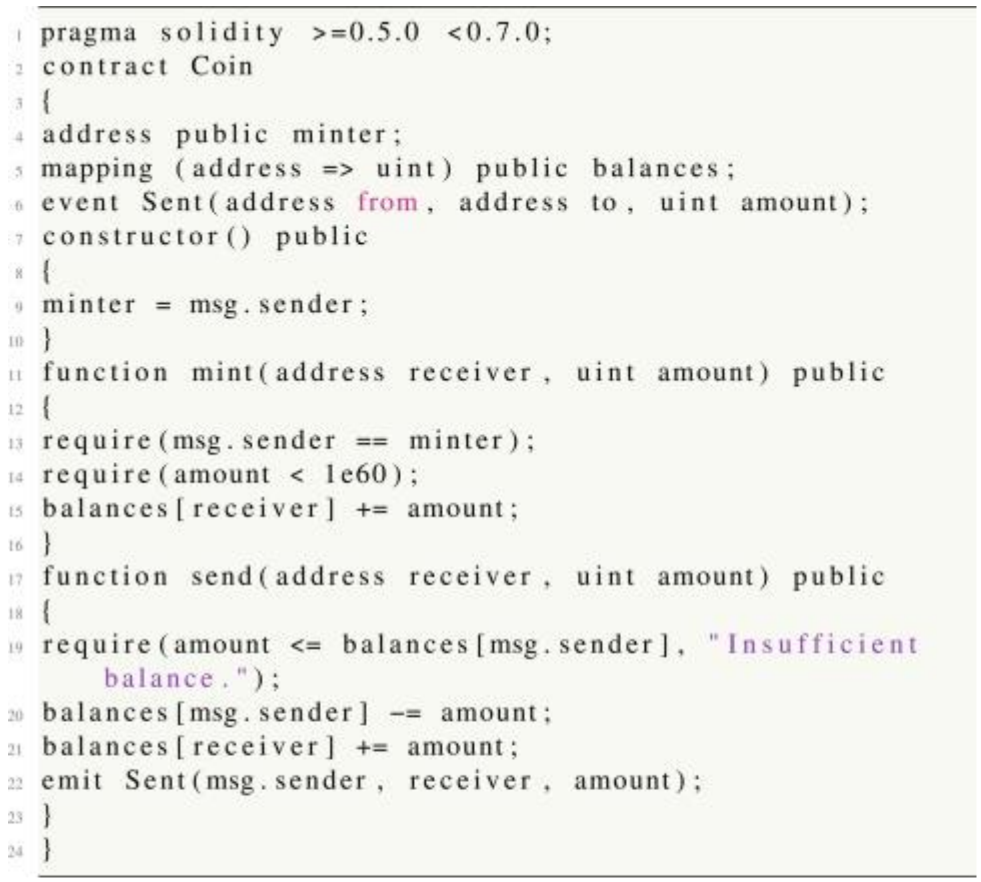


```
pragma solidity >=0.4.0 <0.7.0;
contract SimpleStorage
{
uint storedData;
function set(uint x) public
{
storedData = x;
}
function get() public view returns (uint)
{
return storedData;
}
}
```

2.The following smart contract implements the simplest form of a cryptocurrency as shown in Listing 9. In this contract, anyone can send coins without any registration process. There is only a need for an Ethereum key-pair.

Listing 9

Subcurrency example



```
pragma solidity >=0.5.0 <0.7.0;
contract Coin
{
address public minter;
mapping (address => uint) public balances;
event Sent(address from, address to, uint amount);
constructor() public
{
minter = msg.sender;
}
function mint(address receiver, uint amount) public
{
require(msg.sender == minter);
require(amount < 1e60);
balances[receiver] += amount;
}
function send(address receiver, uint amount) public
{
require(amount <= balances[msg.sender], "Insufficient balance.");
balances[msg.sender] -= amount;
balances[receiver] += amount;
emit Sent(msg.sender, receiver, amount);
}
}
```