A MAIN PROJECT REPORT ON

SMART CONTRACTS USING BLOCK CHAIN

Submitted in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

In

Computer Science and Engineering

By

K. Bhagya Sri(19A81A0586)

Under the Esteemed Supervision of

M.Sree Radha Manga Mani M.Tech

Department of Computer Science and Engineering (Accredited by N.B.A.)

SRI VASAVI ENGINEERING COLLEGE (Autonomous)

(Affiliated to JNTUK, Kakinada)

Pedatadepalli, Tadepalligudem-534101, A.P 2022-23

SRI VASAVI ENGINEERING COLLEGE (Autonomous)

Department Of Computer Science and Engineering

Pedatadepalli, Tadepalligudem

Certificate

This is to certify that the Project Report entitled "**Smart Contracts Using Block Chain**" submitted by **K. Bhagya Sri (19A81A0586)** for the award of the degree of Bachelor of Technology in the Department of Computer Science and Engineering during the academic year 2022-2023.

Name of Project Guide	**Head of the Department**
M. Sree Radha Manga Mani M.Tech.	Dr. D Jaya Kumari M.Tech.,Ph.D..
	Professor & HOD.

External Examiner

DECLARATION

I hereby declare that the project report entitled “**Smart Contracts Using Block Chain**" submitted by me to Sri Vasavi Engineering College(Autonomous), Tadepalligudem, affiliated to JNTUK Kakinada in partial fulfillment of the requirement for the award of the degree of B.Tech in Computer Science and Engineering is a record of Bonafide project work carried out by me under the guidance of **M.Sree Radha Manga Mani, M.Tech** I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree in this institute or any other institute or University.

Project Associate

K. Bhagya Sri (19A81A0586)

ACKNOWLEDGEMENT

First and foremost, I sincerely salute to our esteemed institute **SRI VASAVI ENGINEERING COLLEGE,** for giving me this golden opportunity to fulfill my warm dream to become an engineer.

my sincere gratitude to my project guide **M.Sree Radha Manga Mani,** Department of Computer Science and Engineering**,** for her timely cooperation and valuable suggestions while carrying out this project.

I express my sincere thanks and heartful gratitude to **Dr. D. Jaya Kumari**, Professor &Head of the Department of Computer Science and Engineering, for permitting me to do my project.

I express my sincere thanks and heartful gratitude to **Dr. G.V.N.S.R. Ratnakara Rao**, Principal, for providing a favorable environment and supporting me during the development of this project.

My special thanks to the management and all the teaching and non-teaching staff members, Department of Computer Science and Engineering, for their support and cooperation in various ways during my project work. It is my pleasure to acknowledge the help of all those respected individuals.

I would like to express my gratitude to my parents, friends who helped to complete this project.

Project Associate

K. Bhagya Sri(19A81A0586)

ABSTRACT

Smart contracts are automated digital contracts that enable a secure and self executing agreement to be formulated. Contracting is an essential factor in doing business. A block chain-based smart contract or a "smart contract" for short, is a computer program intended to digitally facilitate the negotiation or contractual terms directly between users without third parties when certain conditions are met. These are now essential for block chain-based business. This project implements the concept of block chain technology. It can be thought of as a system that releases digital assets to all or some of the involved parties once the pre-defined rules have been met. A smart contract focuses on many specific processes such as solving financial fraud, electronic voting, combined block chain and Internet of Things (IoT). Smart contracts represent a next step in the progression of block chains from a financial transaction protocol to an all purpose utility. They are pieces of software, not contracts in the legal sense, that extend block chains' utility from simply keeping a record of financial transaction entries to automatically implementing terms of multiparty agreements.

TABLE OF CONTENTS

INTRODUCTION

1.1 Introduction

A **smart contract** is a computer program or a transaction protocol that is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement. The objectives of smart contracts are the reduction of need for trusted intermediators, arbitrations costs, fraud losses, as well as the reduction of malicious and accidental exceptions. Smart contracts are commonly associated with crypto currencies, and the smart contracts introduced by Ethereum are generally considered a fundamental building block for decentralized finance (DeFi) applications.

Smart contracts are executable codes that run on top of the block chain to facilitate, execute, and enforce an agreement between untrustworthy parties without the involvement of a trusted third-party. Smart contracts gave network automation and the ability to convert paper contracts into digital contracts. Compared to traditional contracts, smart contracts enabled users to codify their agreements and trust relations by providing automated transactions without the supervision of a central authority . In order to prevent contract tampering, smart contracts are copied to each node of the block chain network. By enabling the execution of the operations by computers and services provided by block chain platforms, human error could be reduced to avoid disputes regarding such contracts.

1.2 Motivation

The need for decentralization is the key motivation behind the block chain technology, and decentralization is achieved by distributing the computation tasks to all the nodes of the block chain network. Decentralization solves several problems of traditional systems; the single point of failure is one such problem. For example, in a centralized system such as a bank, the user would always communicate with the same third-party bank to fetch their account details. Although this transaction may be possible almost every time, 100% uptime is not guaranteed, as this server is centralized and has just a few backup servers for load balancing. There could well be a situation where all the servers could be flooded with requests, resulting in crashes and server shutdown. This downtime is something that's inevitable, even in perfectly architected servers**.** If the same scenario was faced in a decentralized network, it wouldn't be an issue, because all the transaction data would be distributed across all the nodes, meaning that each node can act as a backup node in case of failure, maintaining the integrity of the data (another key benefit of block chain-based

solutions).
This is something that's achieved by maintaining a distributed ledger of block chain data. Block chain immutability, which is a key factor in trusting the integrity of the block chain, ensures the integrity of the ledger, which is publicly accessible to all nodes.

1.3 Scope

On block chain, the goal of a smart contract is to simplify business and trade between both anonymous and identified parties, sometimes without the need for a middleman. A smart contract scales down on formality and costs associated with traditional methods, without compromising on authenticity and credibility.Smart contracts are simply programs stored on a block chain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met. Some of the main scope of this project are:

1.Real estate: Smart contracts can be used to automate real estate transactions, such as property transfers, lease agreements, and rental payments. This can reduce the need for intermediaries and speed up the transaction process.
2.Financial services: Smart contracts can be used to automate financial services, such as loans, insurance, and investment management. This can help to reduce the risk of fraud and increase transparency.
3. Supply chain management: Smart contracts can be used to automate supply chain management processes, such as inventory tracking, product delivery, and payment processing. This can help to reduce costs and increase efficiency.
4. Gaming and entertainment: Smart contracts can be used to create decentralized gaming and entertainment platforms, where users can play games and participate in events without the need for intermediaries.
5.Energy management: Smart contracts can be used to manage energy transactions, such as buying and selling electricity, and to incentivize energy conservation.

2. LITERATURE SURVEY

[1] Magazzeni 2017

Magazzeni show that Block chain in its widest sense combines three existing technologies: (1) distributed databases, (2) encryption and (3) consensus protocols. This combination of technologies makes it possible to build applications around a representation of a shared state.

The consensus protocol ensures that parties maintain an identical copy, without the need for a centralized administrator or data storage. So unlike previous automated communication protocols, Blockchain, and general ledger technology make it possible to maintain a so called "stateful shared state" of a series of transactions (Magazzeni et al., 2017). "Shared" refers to the fact that all participants maintain an identical copy, unlike current systems, in which parties have to rely on their own version of events.

[2] Mavridou 2019

The development of smart contracts has proven to be error-prone in practice, and as a result, contracts deployed on public platforms are often riddled with security vulnerabilities. This issue is exacerbated by the design of these platforms, which forbids updating contract code and rolling back malicious transactions. In light of this, it is crucial to ensure that a smart contract is secure before deploying it and trusting it with signifi-
cant amounts of cryptocurrency. To this end, we introduce the VeriSolid framework for the formal verification of contracts that are specified using a transition-system based model with rigorous operational semantics. Our model-based approach allows developers to reason about and verify contract behavior at a high level of abstraction. VeriSolid allows the generation of Solidity code from the verified models, which enables the correct-by-design development of smart contracts.

[3] Nehai 2018

This paper present a way of applying model-checking to a Blockchain Ethereum application based on smart contracts. The corner stone of the approach is to build up a model according to a three-fold modeling process, namely the kernel layer, the application layer and the environment layer. Translation rules from Solidity to NuSMV language have been provided to build the application layer. This approach has been applied to a case study coming from the energy market field: the Blockchain Energy Market Place. Finally Model-Checking technique has been exercised on the resulted NuSMV model to assess some properties of interest formalized in CTL logic.

[4] Bhargavan 2016

Karthikeyan Bhargavan, a computer science researcher who has made significant contributions to the field of smart contracts and blockchain technology. Bhargavan is well known for his work on the security and formal verification of smart contracts. He has proposed techniques for automatically analyzing smart contracts to detect vulnerabilities and prevent security breaches. In addition, he has worked on developing new programming languages and frameworks for writing smart contracts that are less prone to errors and easier to verify.One of Bhargavan's notable contributions is the development of a formal verification framework for the Tezos blockchain. This framework, called Mi-Cho-Coq, allows developers to write smart contracts in a high-level programming language and then use a proof assistant to formally verify the correctness of the contract's behavior. This significantly reduces the risk of bugs and vulnerabilities in smart contracts, which can have serious consequences in decentralized applications.Overall, Bhargavan's work has helped to improve the security and reliability of smart contracts and blockchain systems, which is essential for the widespread adoption of these technologies..

3. SYSTEM STUDY AND ANALYSIS

3.1 Problem Statement

In traditional contract management systems, there are issues such as lack of transparency, high costs, and potential for fraud or manipulation. Smart contracts offer a potential solution to these problems by providing a secure, transparent, and automated way to execute contracts using block chain technology. However, there are still challenges to be addressed in terms of scalability, privacy, and ease of use. The objective of this project is to explore the potential of smart contracts using block chain technology to improve contract management, identify and address existing challenges, and propose solutions to enable widespread adoption of this technology.

3.2 Existing System

In this system, Commercial Contract is the main document governing the deal between you and another organisation. It is the written formal agreement between the two parties. It stipulates each party's legal responsibilities, obligations, governance, contract length, financial details and liabilities within the agreement. It will include the commercial agreements and does not go into the purely technical aspects of the manufacturing, supply or outsourcing of the product or process . Where possible, sections should not be reproduced in multiple documents, as this can lead to contradictions and conflicts. Where appropriate, cross references to relevant sections in different documents should be made.

3.3 Limitations of the Existing System

1 Challenges related to confidentiality are higher as unauthorized users can access the information

2 It has low maintenance.

3 It requires a lot of physical space.

4 Referring to the older transactions is difficult.

5 Tampering the documents is quite easy.

6 Smart contracts are immutable

3.4 Proposed System

Confidentiality Challenges:

Challenges related to confidentiality can decrease the degree to which unauthorized access to information is prevented.

Determinism Challenges:

Challenges related to determinism hinder nodes in a DLT network from computing consistent results by following the same protocol.

Maintainability Challenges:

Maintainability challenges deteriorate the ease with which deployed smart contract code can be updated, for example, to add functionality, correct flaws, or improve code efficiency.

3.5 Advantages of the Proposed System

1.Decentralization: The traditional client-server model relies on a central authority or intermediary to manage and control the network. In contrast, the block chain is a distributed ledger that is not controlled by any single entity, making it more secure, transparent, and resistant to fraud.

2.Immutability: Once a transaction is recorded on the block chain, it cannot be altered or deleted. This immutability ensures that the data on the block chain is tamper-proof and secure, reducing the risk of fraud or malicious activity.

3.Efficiency: Smart contracts are automated, which means they can execute themselves without the need for intermediaries, such as lawyers or banks. This can save time, reduce costs, and increase efficiency.

4.Security: Smart contracts are secured using cryptographic technology, making them tamper-proof and resistant to hacking. This can increase trust and reduce the risk of fraud.

5.Transparency: Smart contracts are stored on a public block chain, which makes them transparent and verifiable. Parties can easily access and view the terms of the contract, and any changes made to it.

3.6 Functional Requirements

1. To provide security to the contract in between 2 people.
2. A layer for the distributed ledgers to store immutable records of data.
3. A layer to implement and run the business logic of the application.
4. The provision of immutable transactions and asset records.

Overall, the functional requirements for smart contracts should aim to provide a secure, transparent, and efficient mechanism for executing transactions and enforcing contracts between parties.

3.7 Non-functional Requirements

- Security: Smart contracts must be designed to be secure, ensuring that they are not vulnerable to hacking, tampering, or other forms of cyber attacks.
- Performance: Smart contracts must be designed to handle a large number of transactions quickly and efficiently, without slowing down the block chain network.
- Scalability: Smart contracts must be designed to scale as the number of users and transactions increases.
- Reliability: Smart contracts must be designed to work reliably and consistently, without any unexpected downtime or errors.
- Compliance: Smart contracts must comply with all relevant laws and regulations, including data protection, privacy, and anti-money laundering law.

3.8 System Requirements

- Programming language: Smart contracts are typically written in programming languages such as Solidity for Ethereum, C++ for EOS, and Java for Hyperledger. It's important to choose a language that is supported by the blockchain platform and that you are comfortable working with.
- Development environment: You will need a development environment for writing, testing, and deploying smart contracts.
- Digital Wallet: A digital wallet is required to interact with the smart contract. It holds the necessary cryptographic keys required to sign and send transactions to the blockchain network.
- System requirements is a statement that identifies the functionality that is needed by a system in order to satisfy the customer's requirements.
- System requirements are a broad and also narrow detailed statement that the customer makes in order to achieve their requirements.
- System requirements can come from the customer, the company, or even other larger groups that set the requirements for specific and whole categories.

3.8.1 Software Requirements

- Operating System: Windows 7 or above
- Languages : Python
- Domain : Block chain

3.8.2 Hardware Requirements

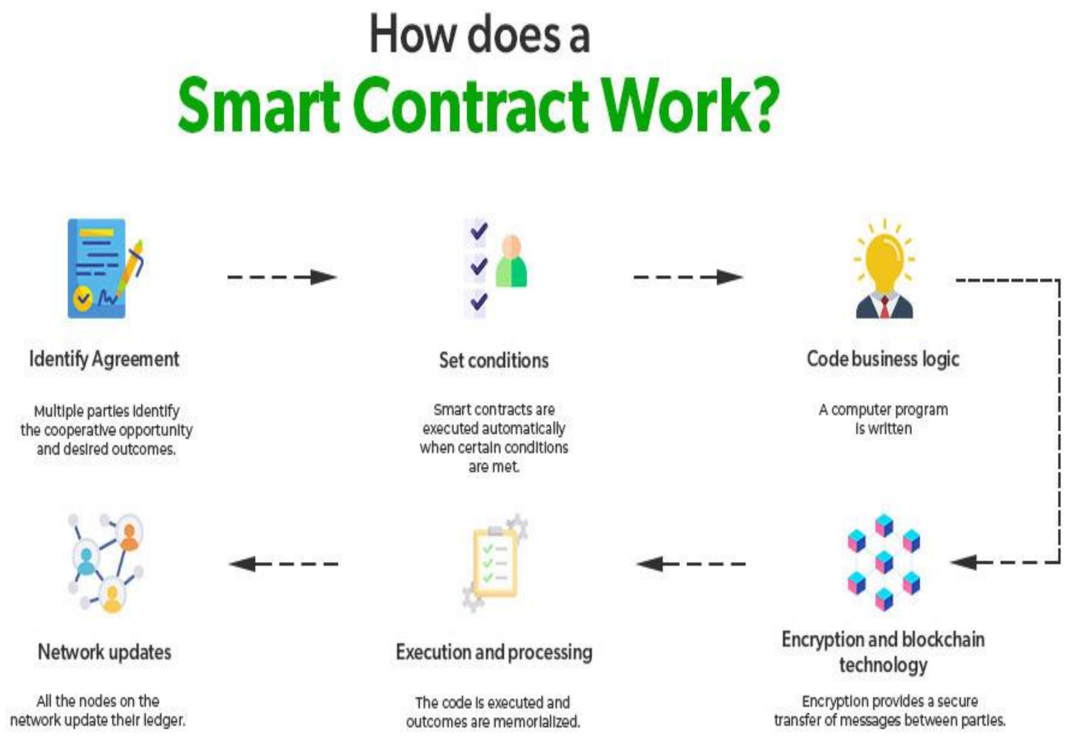
- RAM : 4 GB or above
- Processor : Intel core i3 or above
- Hard Disk : 512GB or above

4.SYSTEM DESIGN

4.1 System Architecture Design

A system Architecture is the conceptual model that defines the structure, behavior, and more views of the system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

A system architecture can consist of system components and the sub-systems developed that will work together with the overall system.


How does a
Smart Contract Work?
Identify Agreement
Multiple parties identify the cooperative opportunity and desired outcomes.
Set conditions
Smart contracts are executed automatically when certain conditions are met.
Code business logic
A computer program is written
Encryption and blockchain technology
Encryption provides a secure transfer of messages between parties.
Execution and processing
The code is executed and outcomes are memorialized.
Network updates
All the nodes on the network update their ledger.

A Smart Contract (or crypto contract) is a computer program that directly and automatically controls the transfer of digital assets between the parties under certain conditions. A smart contract works in the same way as a traditional contract while also automatically enforcing the contract. Smart contracts are programs that execute exactly as they are set up (coded, programmed) by their creators. Just like a traditional contract is enforceable by law, smart contracts are enforceable by code.

4.2 Design Diagrams using UML Approach

UML is a method for describing the system architecture in detail using the blue print. UML represents a collection of best engineering practice that has proven successful in the modeling of large and complex systems. The UML is very important parts of developing object-oriented software and the software development process.

The UML uses mostly graphical notations to express the design of software projects. Using the helps UML helps project teams communicate explore potential designs and validate the architectural design of the software.

The Unified Modeling Language (UML) was created to forge a common, semantically, and syntactically rich visual modeling language for the architecture, design, and implementation of complex software systems both structurally and behaviorally. UML has applications beyond software development, such as process flow in manufacturing.

It is analogous to the blueprints used in other fields, and consists of different types of diagrams. In the aggregate, UML diagrams describe the boundary, structure, and the behavior of the system and the objects within it.

Object-oriented concepts in UML

The objects in UML are real world entities that exist around us. In software development, objects can be used to describe, or model, the system being created in terms that are relevant to the domain. Objects also allow the decomposition of complex systems into understandable components that allow one piece to be built at a time.

Here are some fundamental concepts of an object-oriented world:

- Objects Represent an entity and the basic building block.
- Class Blue print of an object.
- Abstraction Behavior of a real world entity.
- Encapsulation Mechanism of binding the data together and hiding them from outside world.
- Inheritance Mechanism of making new classes from existing one.
- Polymorphism It defines the mechanism to exists in different forms.

Types of UML Diagrams

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation).

The two broadest categories that encompass all other types are Behavioral UML diagram and Structural UML diagram. As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas other describe the behavior of the system, its actors, and its building components.

4.2.1 Class Diagram

Class-based Modeling, or more commonly class-orientation, refers to the style of object-oriented programming in which inheritance is achieved by defining classes of objects; as opposed to the objects themselves (compare Prototype-based programming). The most popular and developed model of OOP is a class-based model, as opposed to an object-based model. In this model, objects are entities that combine state (i.e., data), behavior (i.e., procedures, or methods) and identity (unique existence among all other objects). The structure and behavior of an object are defined by a class, which is a definition, or blueprint, of all objects of a specific type. An object must be explicitly created based on a class and an object thus created is considered to be an instance of that class. An object is similar to a structure, with the addition of method pointers, member access control, and an implicit data member which locates instances of the class (i.e., actual objects of that class) in the class hierarchy (essential for runtime inheritance features). Class diagram is a static diagram. It represents the static view of an application.

4.2.2 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

4.2.3 Collaboration Diagram

A Collaboration diagram is easily represented by modeling objects in a system and representing the associations between the objects as links. The interaction between the objects is denoted by arrows. To identify the sequence of invocation of these objects, a number is placed next to each of these arrows.

4.2.4 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. Activity diagrams are constructed from a limited repertoire of shapes, connected with arrows.

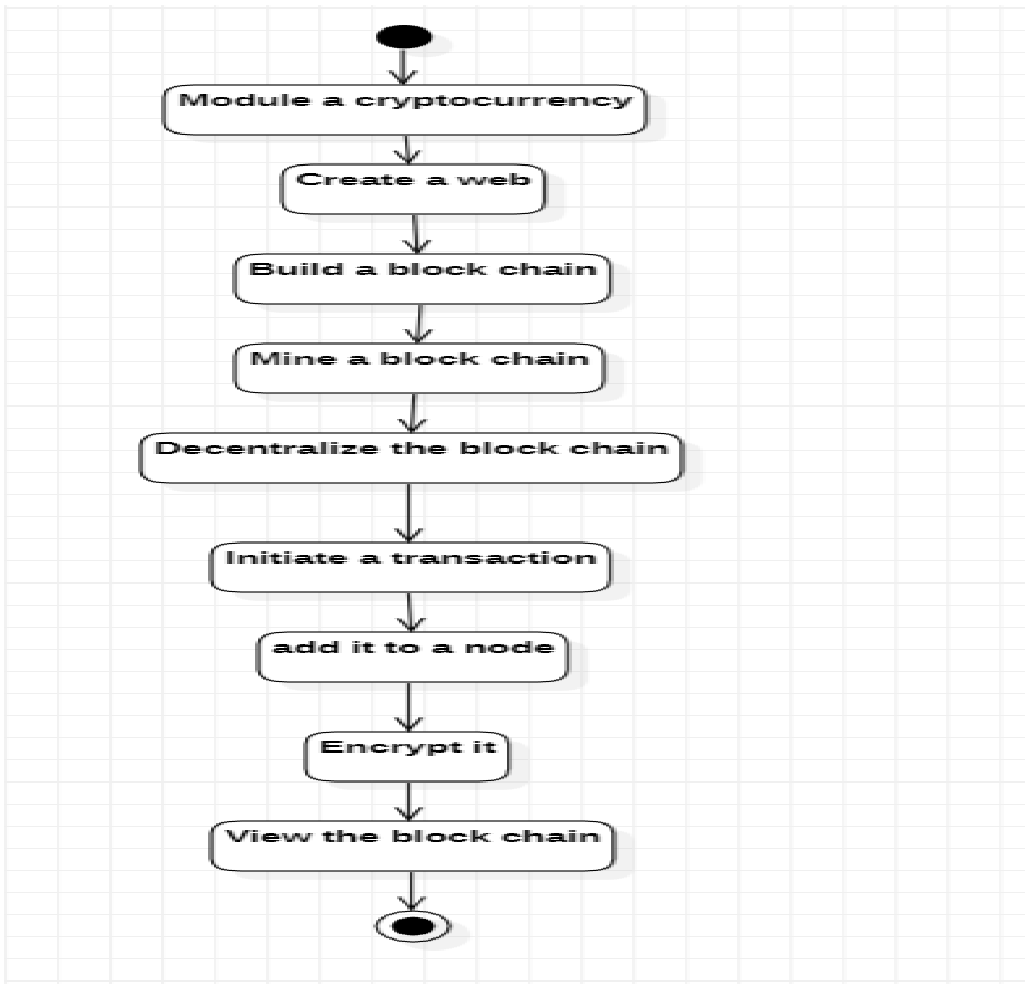
The most important shape types:

• diamonds represent decisions;

• bars represent the start (split) or end (join) of concurrent activities;

• a black circle represents the start (initial state) of the workflow;

• An encircled black circle represents the end (final state).



4.2.5 Deployment Diagram

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed. So deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships. Purpose: The name Deployment itself describes the purpose of the diagram.

Deployment diagrams are used for describing the hardware components where software components are deployed. Component diagrams and deployment diagrams are closely related. Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware. UML is mainly designed to focus on software artifacts of a system. But these two diagrams are special diagrams used to focus on software components and hardware components. So, most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on hardware topology of a system.

5. TECHNOLOGIES

5.1 Python

5.1.1 About Python:

Below are some facts about Python.

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.

Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time. Python language is being used by almost all tech giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

5.2.1 Block Chain:

The Block Chain find wide applications in Smart contracts, Supply chain management, Asset protection through an indisputable record of real-time ownership, Personal data management and Identification, Payment processing, Crowd funding through crypto-currencies, tracking drugs in pharmaceutical supply chains, verification of land records and certificates, etc.

Block chain has been heralded as the most significant innovation since the Internet itself. It initially emerged as the backbone of bit-coin and is an incorruptible digital public ledger of transactions. Block chain technology is a novel data structure that is secure, cryptography-based, and stores transactional records (known as the block) in databases (known as chains) distributed across a network through peer-to-peer nodes, allowing the transfer of digital goods.

Block chain is a combination of three leading technologies:

1. Cryptographic keys

2. A peer-to-peer network containing a shared ledger

3. A means of computing, to store the transactions and records of the network

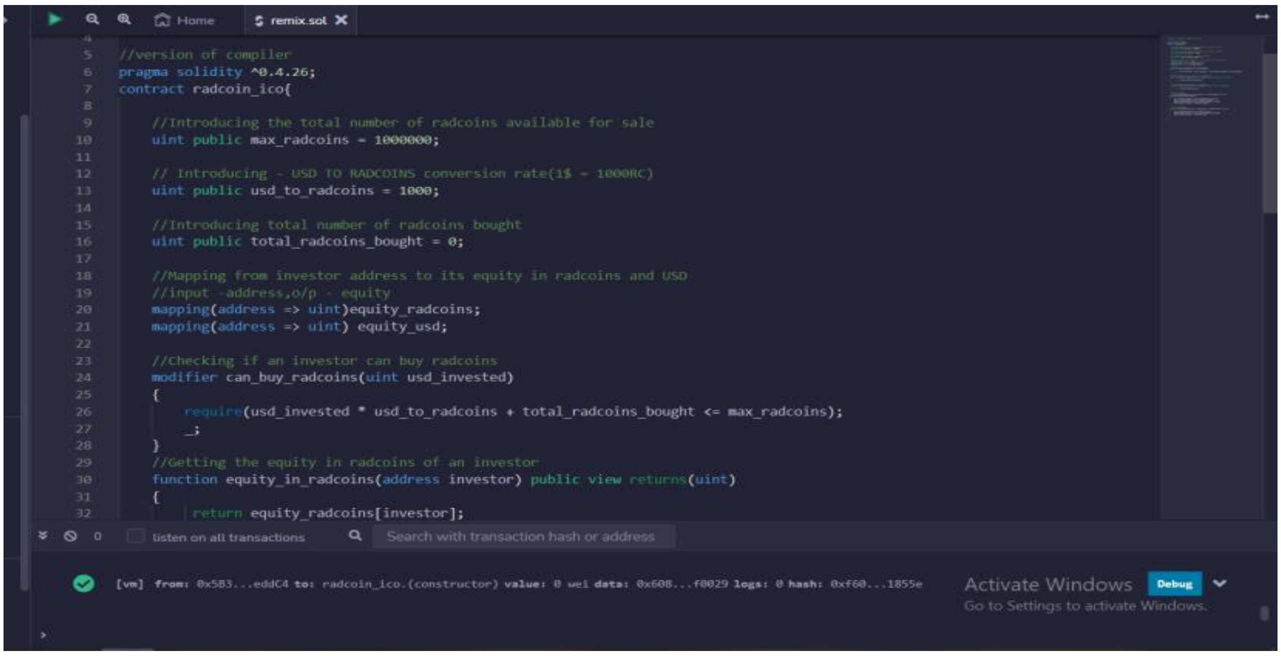
5.3.1 :Ethereum:

Ethereum is a decentralized block chain platform that establishes a peer-to-peer network that securely executes and verifies application code, called smart contracts. Smart contracts allow participants to transact with each other without a trusted central authority. Transaction records are immutable, verifiable, and securely distributed across the network, giving participants full ownership and visibility into transaction data. Transactions are sent from and received by user-created Ethereum accounts. A sender must sign transactions and spend Ether, Ethereum's native crypto-currency, as a cost of processing transactions on the network.

This block chain is validated by a network of automated programs that reach a consensus on the validity of transaction information. No changes can be made to the block chain unless the network reaches a consensus. This makes it very secure.

6. IMPLEMENTATION

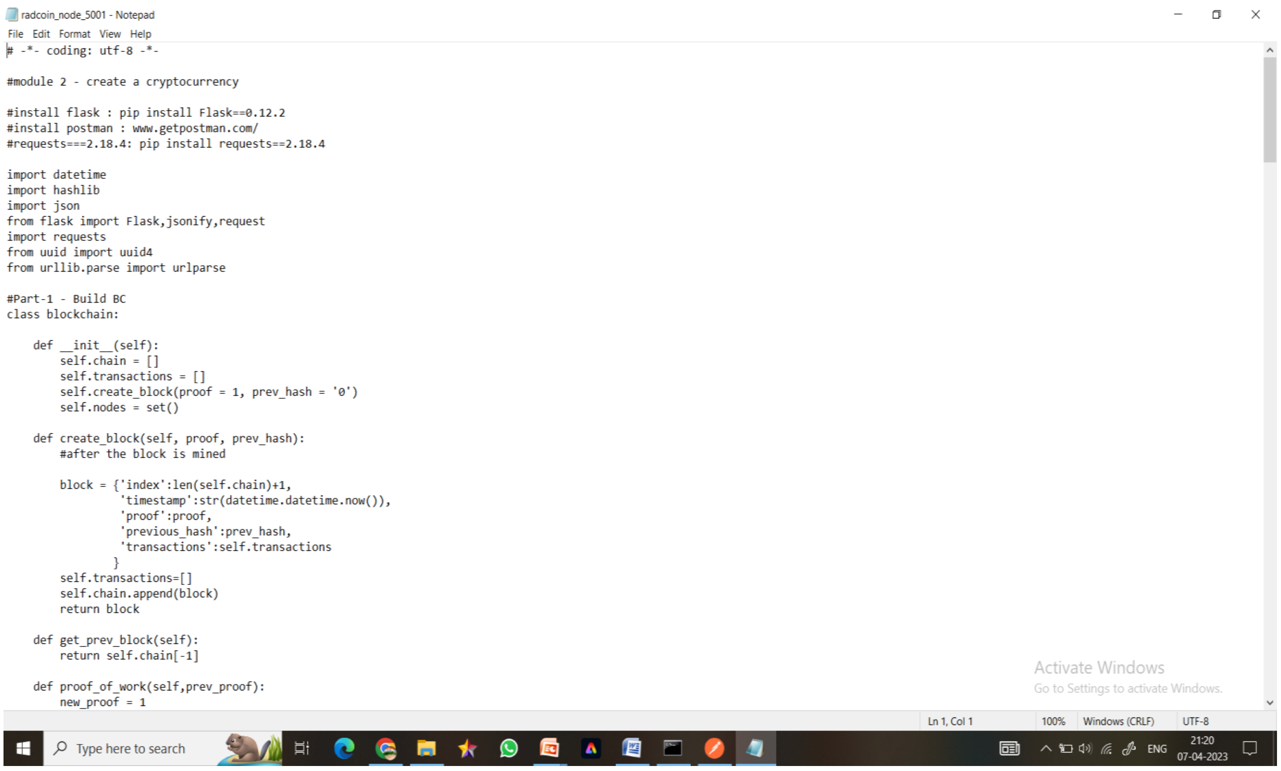
1.Ethereum Code(Contract)


Home
remix.sol
listen on all transactions
Search with transaction hash or address
Activate Windows
Go to Settings to activate Windows.
Debug

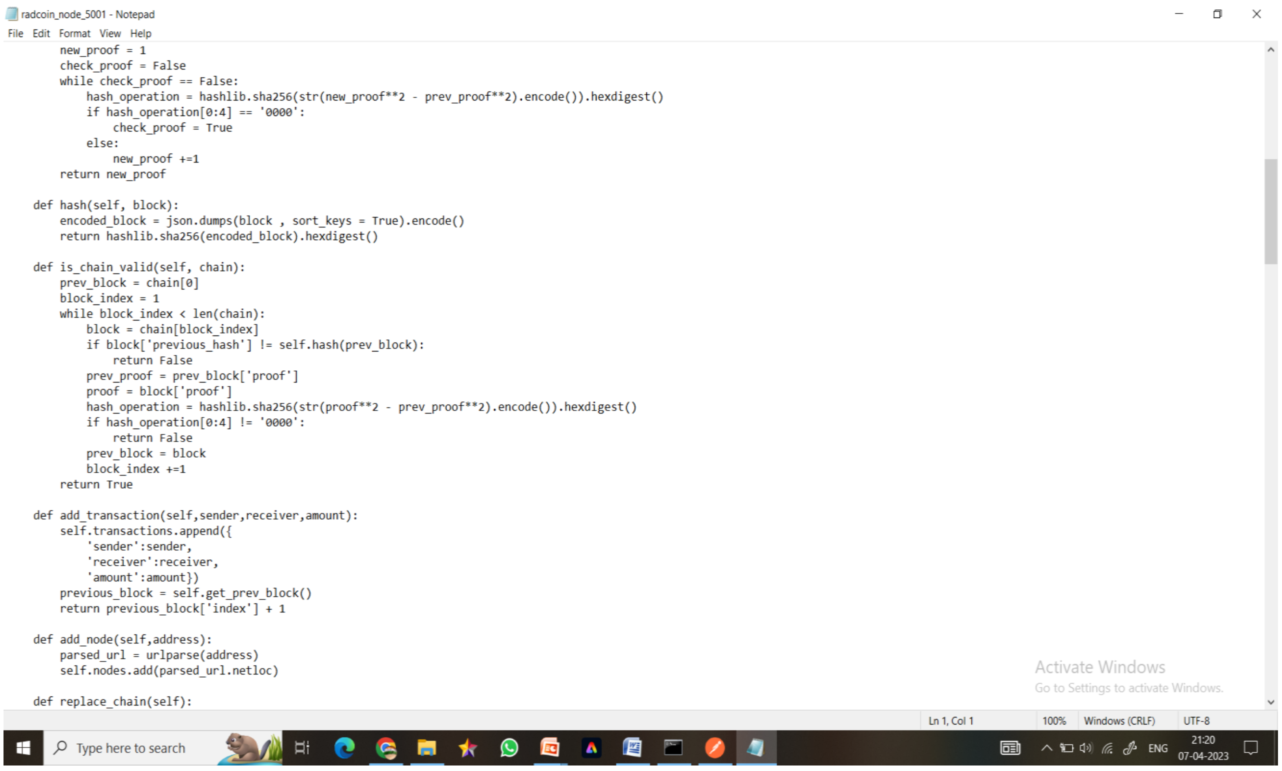
```
//version of compiler
pragma solidity ^0.4.26;
contract radcoin_ico{

    //Introducing the total number of radcoins available for sale
    uint public max_radcoins = 1000000;

    // Introducing - USD TO RADCOINS conversion rate(1$ - 1000RC)
    uint public usd_to_radcoins = 1000;

    //Introducing total number of radcoins bought
    uint public total_radcoins_bought = 0;

    //Mapping from investor address to its equity in radcoins and USD
    //input -address,o/p - equity
    mapping(address => uint)equity_radcoins;
    mapping(address => uint) equity_usd;

    //Checking if an investor can buy radcoins
    modifier can_buy_radcoins(uint usd_invested)
    {
        require(usd_invested * usd_to_radcoins + total_radcoins_bought <= max_radcoins);
        _;
    }
    //Getting the equity in radcoins of an investor
    function equity_in_radcoins(address investor) public view returns(uint)
    {
        return equity_radcoins[investor];
```

```
[vm] from: 0x5B3...eddC4 to: radcoin_ico.(constructor) value: 0 wei data: 0x608...f0029 logs: 0 hash: 0xf60...1855e
```

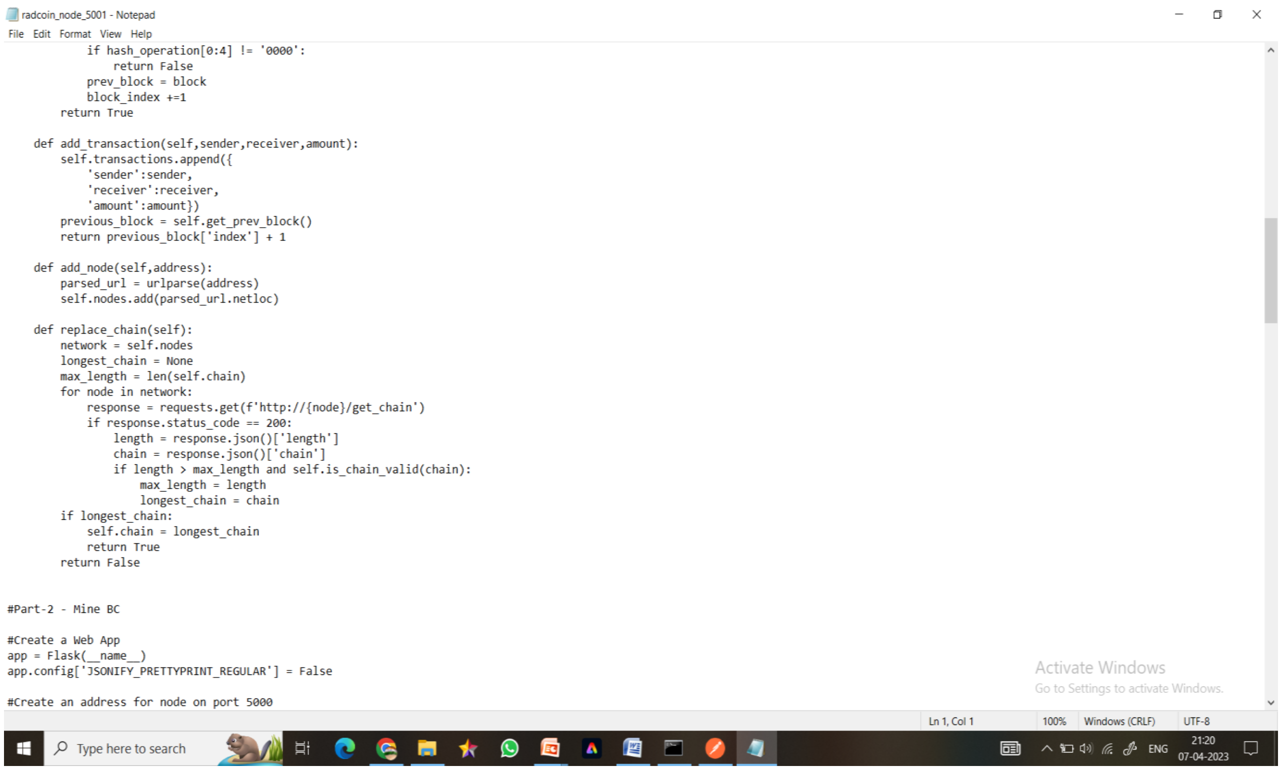
Connecting nodes, Adding Transactions, etc.


radcoin_node_5001 - Notepad

```
# -*- coding: utf-8 -*-

#module 2 - create a cryptocurrency

#install flask : pip install Flask==0.12.2
#install postman : www.getpostman.com/
#requests===2.18.4: pip install requests==2.18.4

import datetime
import hashlib
import json
from flask import Flask,jsonify,request
import requests
from uuid import uuid4
from urllib.parse import urlparse

#Part-1 - Build BC
class blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(proof = 1, prev_hash = '0')
        self.nodes = set()

    def create_block(self, proof, prev_hash):
        #after the block is mined

        block = {'index':len(self.chain)+1,
                 'timestamp':str(datetime.datetime.now()),
                 'proof':proof,
                 'previous_hash':prev_hash,
                 'transactions':self.transactions
                }
        self.transactions=[]
        self.chain.append(block)
        return block

    def get_prev_block(self):
        return self.chain[-1]

    def proof_of_work(self,prev_proof):
        new_proof = 1
```

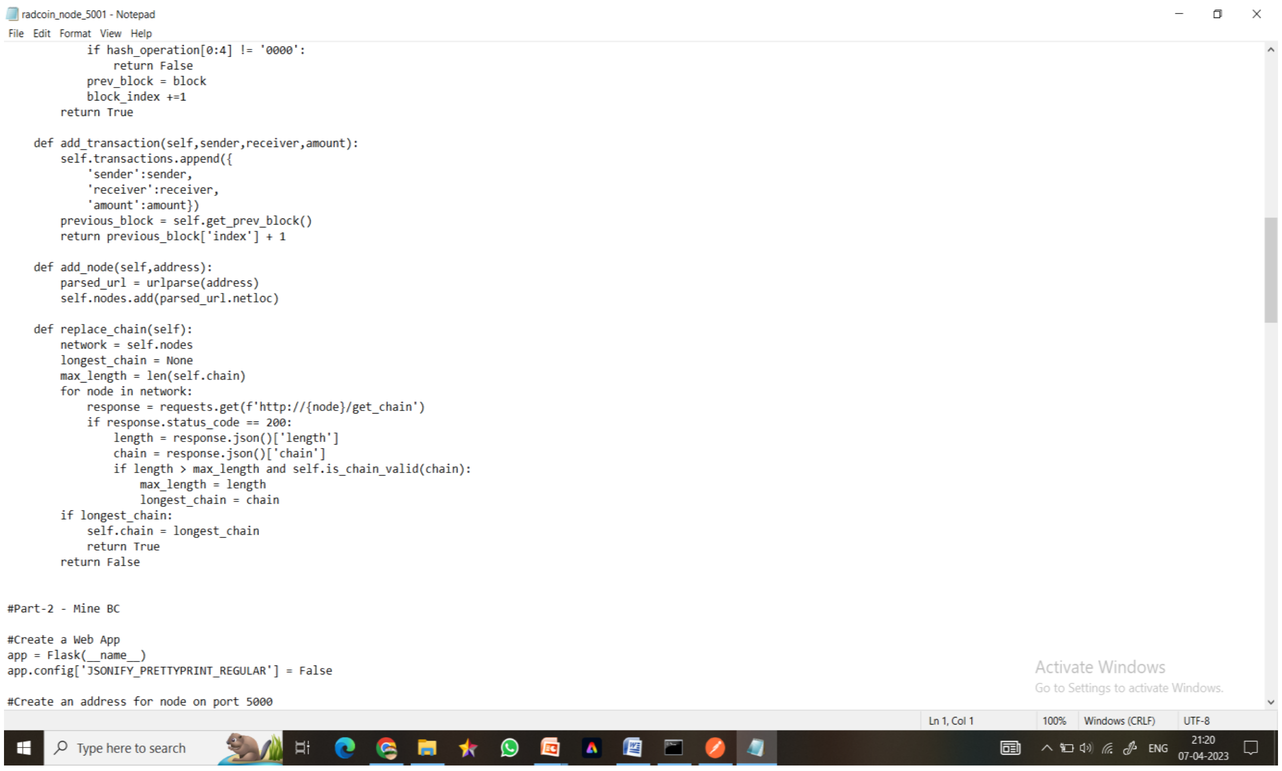

radcoin_node_5001 - Notepad

```
        new_proof = 1
        check_proof = False
        while check_proof == False:
            hash_operation = hashlib.sha256(str(new_proof**2 - prev_proof**2).encode()).hexdigest()
            if hash_operation[0:4] == '0000':
                check_proof = True
            else:
                new_proof +=1
        return new_proof

    def hash(self, block):
        encoded_block = json.dumps(block , sort_keys = True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        prev_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(prev_block):
                return False
            prev_proof = prev_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(str(proof**2 - prev_proof**2).encode()).hexdigest()
            if hash_operation[0:4] != '0000':
                return False
            prev_block = block
            block_index +=1
        return True

    def add_transaction(self,sender,receiver,amount):
        self.transactions.append({
            'sender':sender,
            'receiver':receiver,
            'amount':amount})
        previous_block = self.get_prev_block()
        return previous_block['index'] + 1

    def add_node(self,address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)

    def replace_chain(self):
```



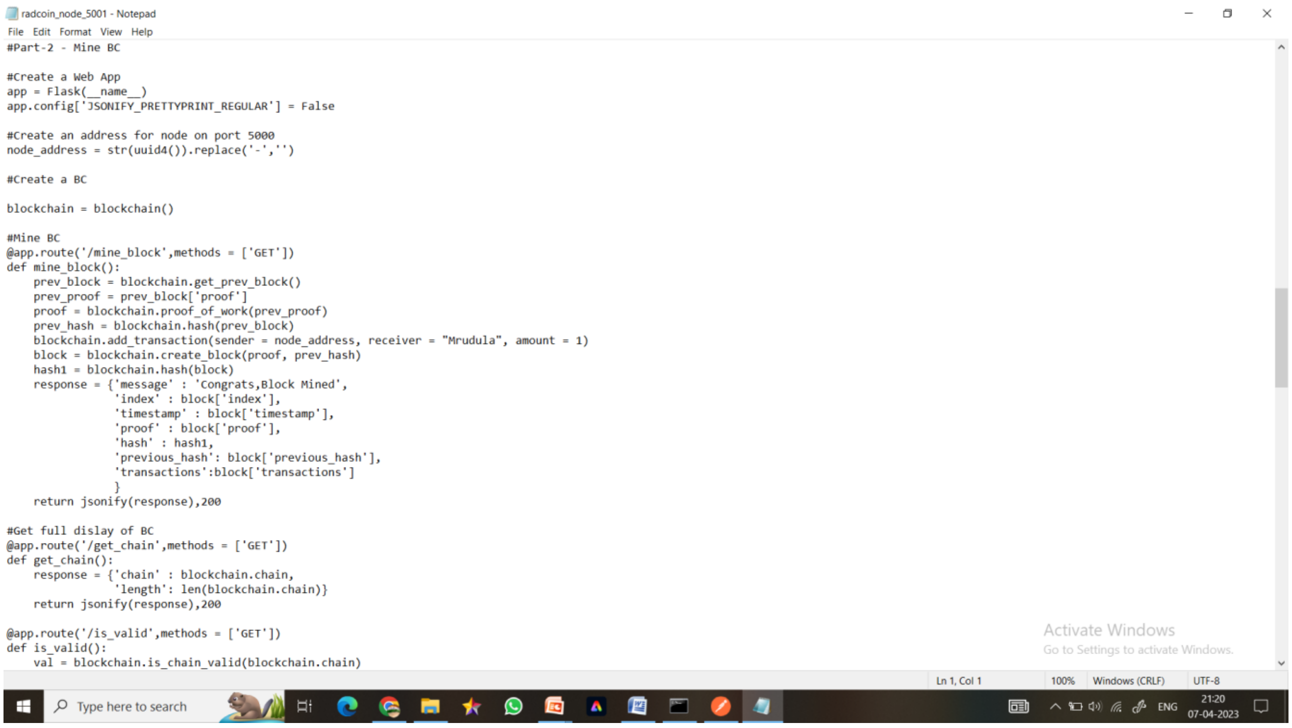
radcoin_node_5001 - Notepad

```
            if hash_operation[0:4] != '0000':
                return False
            prev_block = block
            block_index +=1
        return True

    def add_transaction(self,sender,receiver,amount):
        self.transactions.append({
            'sender':sender,
            'receiver':receiver,
            'amount':amount})
        previous_block = self.get_prev_block()
        return previous_block['index'] + 1

    def add_node(self,address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)

    def replace_chain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)
        for node in network:
            response = requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']
                if length > max_length and self.is_chain_valid(chain):
                    max_length = length
                    longest_chain = chain
        if longest_chain:
            self.chain = longest_chain
            return True
        return False


#Part-2 - Mine BC

#Create a Web App
app = Flask(__name__)
app.config['JSONIFY_PRETTYPRINT_REGULAR'] = False

#Create an address for node on port 5000
```

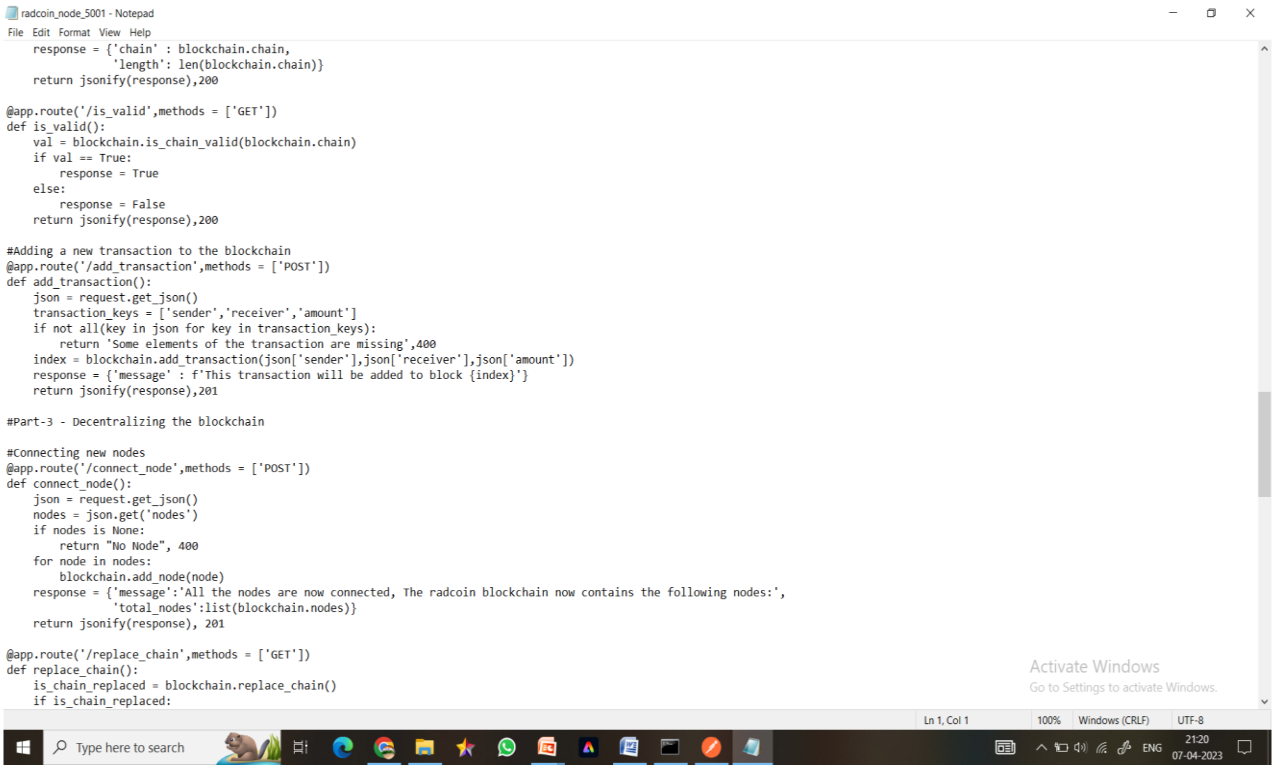


radcoin_node_5001 - Notepad

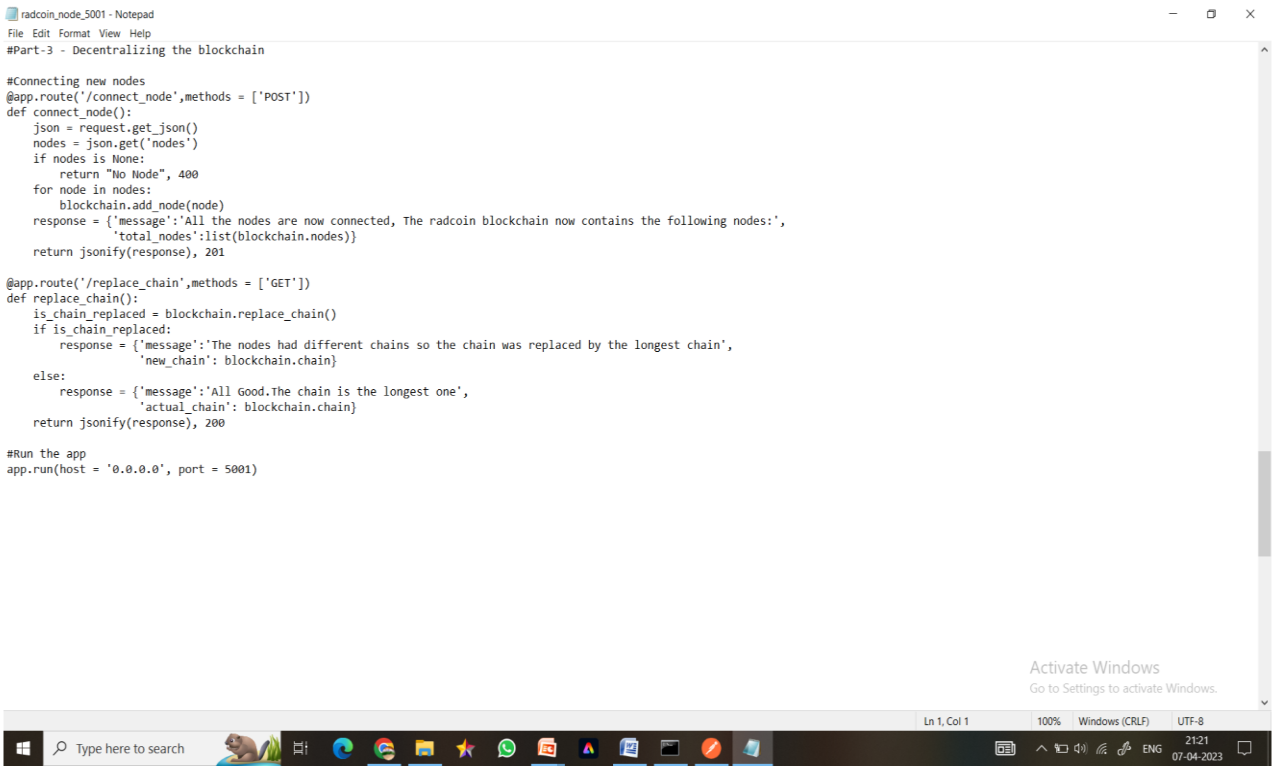
```
            if hash_operation[0:4] != '0000':
                return False
            prev_block = block
            block_index +=1
        return True

    def add_transaction(self,sender,receiver,amount):
        self.transactions.append({
            'sender':sender,
            'receiver':receiver,
            'amount':amount})
        previous_block = self.get_prev_block()
        return previous_block['index'] + 1

    def add_node(self,address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)

    def replace_chain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)
        for node in network:
            response = requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']
                if length > max_length and self.is_chain_valid(chain):
                    max_length = length
                    longest_chain = chain
        if longest_chain:
            self.chain = longest_chain
            return True
        return False


#Part-2 - Mine BC

#Create a Web App
app = Flask(__name__)
app.config['JSONIFY_PRETTYPRINT_REGULAR'] = False

#Create an address for node on port 5000
```


radcoin_node_5001 - Notepad
File Edit Format View Help
Ln 1, Col 1 100% Windows (CRLF) UTF-8
Type here to search
21:20 07-04-2023

```
#Part-2 - Mine BC

#Create a Web App
app = Flask(__name__)
app.config['JSONIFY_PRETTYPRINT_REGULAR'] = False

#Create an address for node on port 5000
node_address = str(uuid4()).replace('-','')

#Create a BC

blockchain = blockchain()

#Mine BC
@app.route('/mine_block',methods = ['GET'])
def mine_block():
    prev_block = blockchain.get_prev_block()
    prev_proof = prev_block['proof']
    proof = blockchain.proof_of_work(prev_proof)
    prev_hash = blockchain.hash(prev_block)
    blockchain.add_transaction(sender = node_address, receiver = "Mrudula", amount = 1)
    block = blockchain.create_block(proof, prev_hash)
    hash1 = blockchain.hash(block)
    response = {'message' : 'Congrats,Block Mined',
                'index' : block['index'],
                'timestamp' : block['timestamp'],
                'proof' : block['proof'],
                'hash' : hash1,
                'previous_hash': block['previous_hash'],
                'transactions':block['transactions']
                }
    return jsonify(response),200

#Get full dislay of BC
@app.route('/get_chain',methods = ['GET'])
def get_chain():
    response = {'chain' : blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response),200

@app.route('/is_valid',methods = ['GET'])
def is_valid():
    val = blockchain.is_chain_valid(blockchain.chain)
```

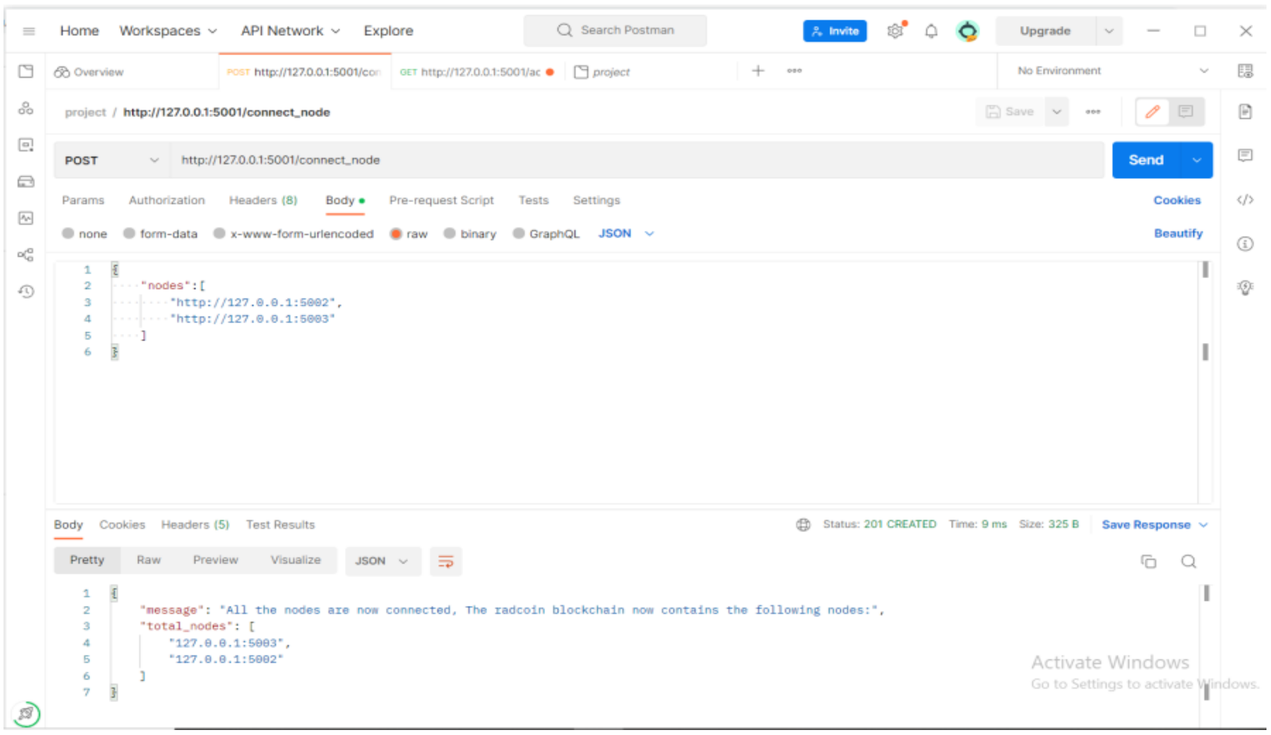

radcoin_node_5001 - Notepad
File Edit Format View Help
Ln 1, Col 1 100% Windows (CRLF) UTF-8
Type here to search
21:20 07-04-2023

```
    response = {'chain' : blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response),200

@app.route('/is_valid',methods = ['GET'])
def is_valid():
    val = blockchain.is_chain_valid(blockchain.chain)
    if val == True:
        response = True
    else:
        response = False
    return jsonify(response),200

#Adding a new transaction to the blockchain
@app.route('/add_transaction',methods = ['POST'])
def add_transaction():
    json = request.get_json()
    transaction_keys = ['sender','receiver','amount']
    if not all(key in json for key in transaction_keys):
        return 'Some elements of the transaction are missing',400
    index = blockchain.add_transaction(json['sender'],json['receiver'],json['amount'])
    response = {'message' : f'This transaction will be added to block {index}'}
    return jsonify(response),201

#Part-3 - Decentralizing the blockchain

#Connecting new nodes
@app.route('/connect_node',methods = ['POST'])
def connect_node():
    json = request.get_json()
    nodes = json.get('nodes')
    if nodes is None:
        return "No Node", 400
    for node in nodes:
        blockchain.add_node(node)
    response = {'message':'All the nodes are now connected, The radcoin blockchain now contains the following nodes:',
                'total_nodes':list(blockchain.nodes)}
    return jsonify(response), 201

@app.route('/replace_chain',methods = ['GET'])
def replace_chain():
    is_chain_replaced = blockchain.replace_chain()
    if is_chain_replaced:
```



```
#Part-3 - Decentralizing the blockchain

#Connecting new nodes
@app.route('/connect_node',methods = ['POST'])
def connect_node():
    json = request.get_json()
    nodes = json.get('nodes')
    if nodes is None:
        return "No Node", 400
    for node in nodes:
        blockchain.add_node(node)
    response = {'message':'All the nodes are now connected, The radcoin blockchain now contains the following nodes:',
             'total_nodes':list(blockchain.nodes)}
    return jsonify(response), 201

@app.route('/replace_chain',methods = ['GET'])
def replace_chain():
    is_chain_replaced = blockchain.replace_chain()
    if is_chain_replaced:
        response = {'message':'The nodes had different chains so the chain was replaced by the longest chain',
              'new_chain': blockchain.chain}
    else:
        response = {'message':'All Good.The chain is the longest one',
              'actual_chain': blockchain.chain}
    return jsonify(response), 200

#Run the app
app.run(host = '0.0.0.0', port = 5001)
```

Execution

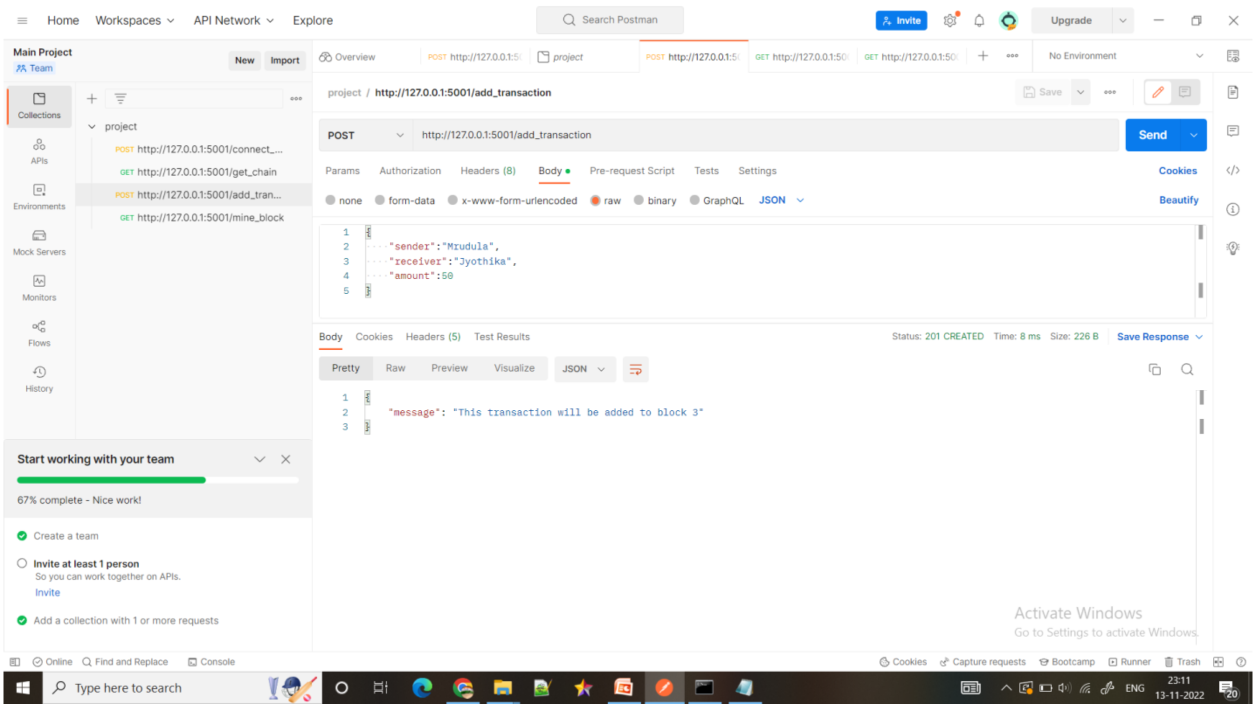


POST http://127.0.0.1:5001/connect_node

```
{
    "nodes":[
        "http://127.0.0.1:5002",
        "http://127.0.0.1:5003"
    ]
}
```

Status: 201 CREATED Time: 9 ms Size: 325 B

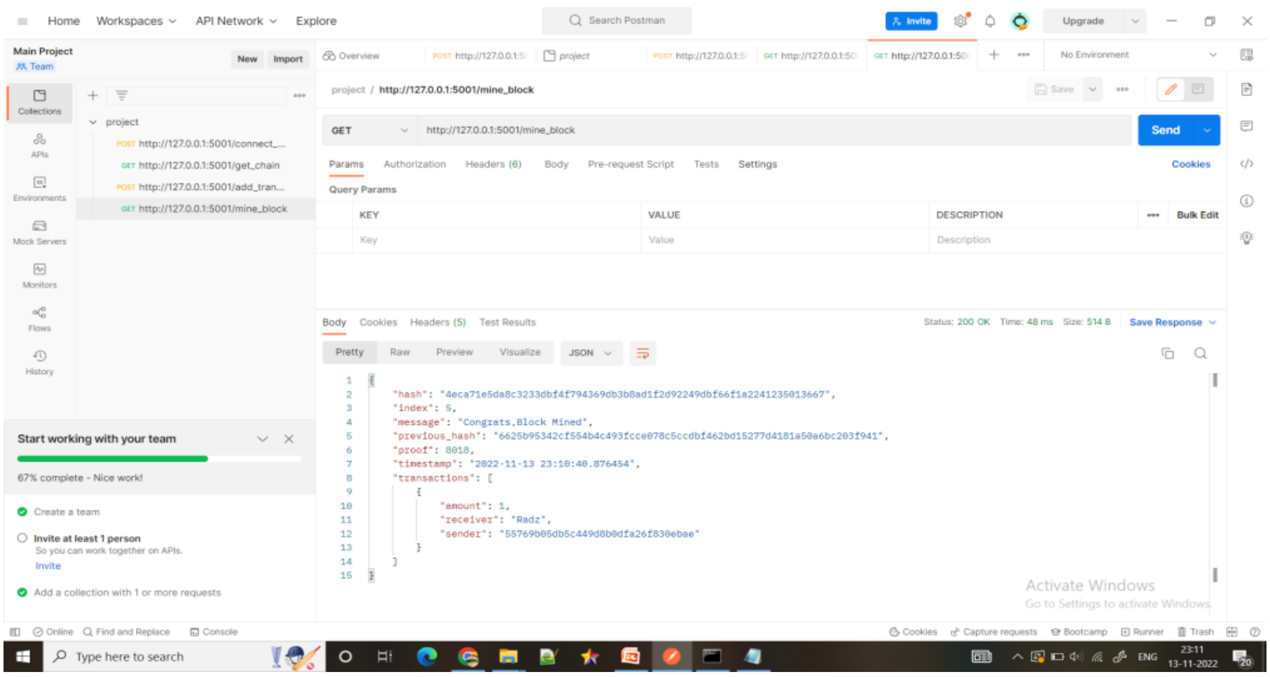
```
{
    "message": "All the nodes are now connected, The radcoin blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5003",
        "127.0.0.1:5002"
    ]
}
```


POST http://127.0.0.1:5001/add_transaction

```
{
    "sender":"Mrudula",
    "receiver":"Jyothika",
    "amount":50
}
```

Status: 201 CREATED Time: 8 ms Size: 226 B

```
{
    "message": "This transaction will be added to block 3"
}
```


GET http://127.0.0.1:5001/mine_block

Status: 200 OK Time: 48 ms Size: 514 B

```
{
    "hash": "4eca71e5da8c3233dbf4f794369db3b8ad1f2d92249dbf66f1a2241235013667",
    "index": 5,
    "message": "Congrats,Block Mined",
    "previous_hash": "6625b95342cf554b4c493fcce078c5ccdbf462bd15277d4181a50a6bc203f941",
    "proof": 8018,
    "timestamp": "2022-11-13 23:10:40.876454",
    "transactions": [
        {
            "amount": 1,
            "receiver": "Radz",
            "sender": "55769b05db5c449d8b0dfa26f830ebae"
        }
    ]
}
```

7. TESTING

7.1 Introduction To Testing :

Testing is a crucial process in software development that involves evaluating a product or system to identify defects, errors, or other issues before its release. The main objective of Testing is to ensure that the product or system functions as intended, meets its performance, Reliability, and security requirements, and satisfies the expectations of stakeholders. Testing can be performed on various levels and techniques, such as unit testing, Integration Testing, System Testing, and Automated Testing. By having a well-designed testing process, developers can improve the quality of the product or system, reduce the risk of issues or defects in the field, and ensure that the system meets the user's needs.

7.2 Testing Strategies:

When it comes to testing strategies for block chain, there are several approaches that can be taken. One common strategy is to focus on testing the smart contracts that are built on top of the Block chain. This involves creating test cases that simulate various scenarios and edge cases to ensure the contracts function as intended. Another important aspect of testing block chain is performance testing, which involves simulating high volumes of transactions to ensure the system performance testing, which involves simulating high volumes of transactions to ensure the system to attacks. Additionally, testing interoperability with other block chain systems and legacy systems can help ensure seamless integration.

7.3 Unit Testing:

Unit testing is an essential part of block chain development as it helps ensure the integrity and reliability of the code. The goal of unit testing is to isolate and test individual pieces of code, such as functions or methods, to ensure they behave as expected. For block chain, unit testing can be used to test smart contracts, which are self-executing programs that run on the block chain. Test cases can be designed to cover various scenarios and edge cases to ensure the smart contract functions as intended. Unit testing can also be used to test other components of the block chain system, such as the consensus algorithm and the peer-to-peer networking layer.

7.3.1 Integration Testing:

Integration testing is a critical part of block chain development, as it tests the interaction between various components of the system to ensure they work together seamlessly. In the context of block chain, integration testing involves testing the integration of smart contracts with the block chain platform, as well as the integration between different block chain nodes. This includes various components of the system to ensure they work together seamlessly. In the context of block chain, integration testing involves testing the integration of smart contracts with the block chain platform, as well as the integration between different block chain nodes. This includes various components of the system to ensure they work together seamlessly. In the context of block chain, integration testing involves testing the integration of smart contracts with the block chain platform, as well as the integration between different block chain nodes. This includes testing the communication and synchronization of the nodes, as well as the consensus mechanism used to validate transactions. Integration testing can also involve testing the integration of

Block chain systems with external systems, such as wallets or exchanges, to ensure interoperability. used to validate transactions. Integration testing can also involve testing the integration of Block chain systems with external systems, such as wallets or exchanges, to ensure interoperability. By testing these integrations, developers can identify and fix issues before they impact the system as a whole.

7.3.2 System Testing:

System testing is an important part of block chain development, as it tests the entire system as a whole to ensure it meets the intended requirements and performs as expected. In the context of block chain, system testing involves testing the end-to-end functionality of the system, including the consensus mechanism, smart contracts, and network communication. This can include testing the consensus mechanism, smart contracts, and network communication. This can include testing the performance and scalability of the system, as well as its ability to handle high transaction volumes.Additionally, system testing can involve testing the security of the system, including vulnerability assessments and penetration testing to identify potential attack vectors. By conducting thorough system testing, developers can ensure that the blockchain system meets the necessary quality standards before deployment.

7.3.3 Acceptance Testing:

Acceptance testing is a critical part of block chain development, as it involves testing the system from the perspective of end-users and stakeholders to ensure it meets their requirements and expectations. In the context of block chain, acceptance testing can involve testing the functionality of the system, including the user interface and user experience. It can also involve testing the security and privacy features of the system, such as encryption and access control. Additionally, acceptance testing can involve testing the interoperability of the system with external systems, such as wallets and exchanges. By conducting acceptance testing, developers can ensure that the Block chain system meets the necessary quality standards and satisfies the needs of its users.

7.4 Test Cases:

Test cases for smart contracts in block chain can vary depending on the specific requirements of the contract, but here are some general examples:

1.Test case to verify contract deployment: This test case verifies that the contract can be successfully deployed on the block chain network.

2.Test case to verify contract initialization: This test case verifies that the contract is properly initialized with the correct parameters.

3.Test case to verify contract functionality: This test case verifies that the contract performs as expected under various scenarios and edge cases.

4.Test case to verify contract security: This test case verifies that the contract is secure and cannot be exploited by malicious actors.

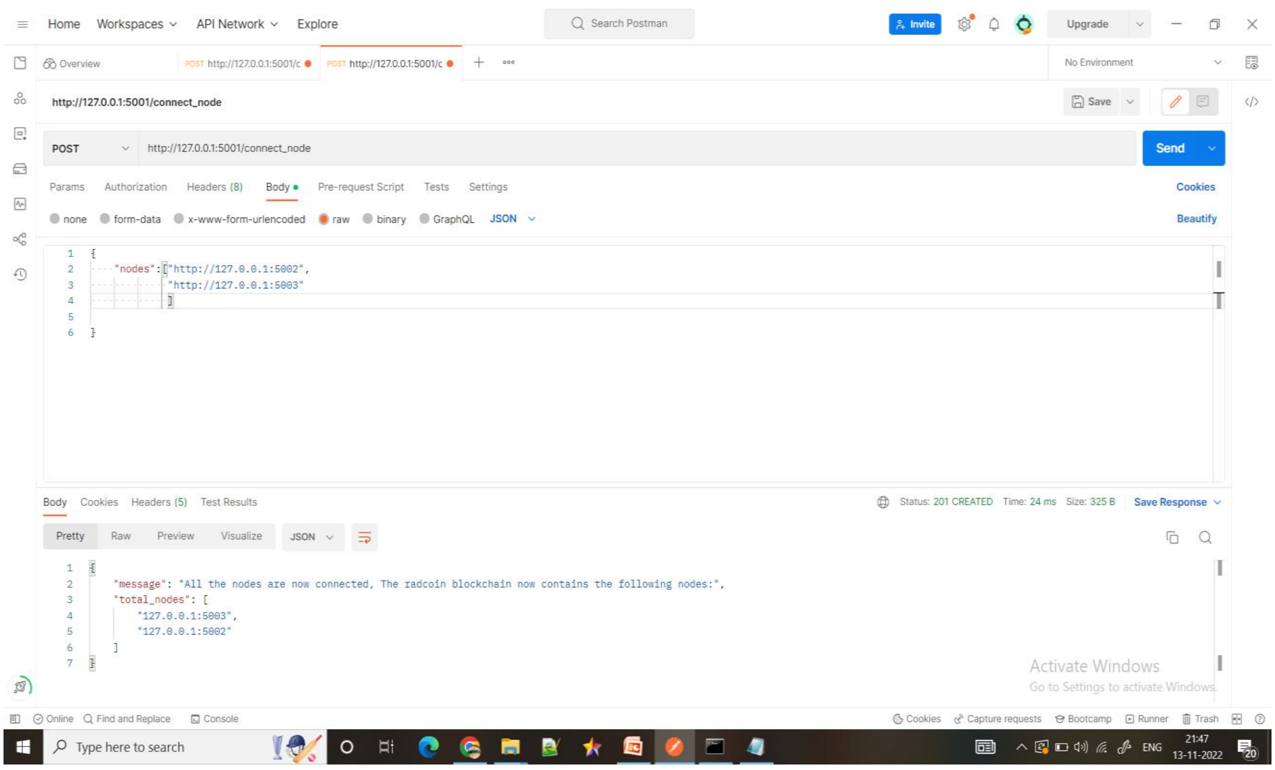
5.Test case to verify contract interaction with other contracts: This test case verifies that the contract can interact correctly with other contracts on the block chain network.

6.Test case to verify contract up gradeability: This test case verifies that the contract can be upgraded without compromising its integrity or security.

7.Test case to verify contract interoperability: This test case verifies that the contract can interact correctly with external systems, such as wallets and exchanges.

By designing and executing these and other relevant test cases, developers can ensure that the smart contract performs as intended and meets the necessary quality standards.

8. SCREEN SHOTS



http://127.0.0.1:5001/connect_node

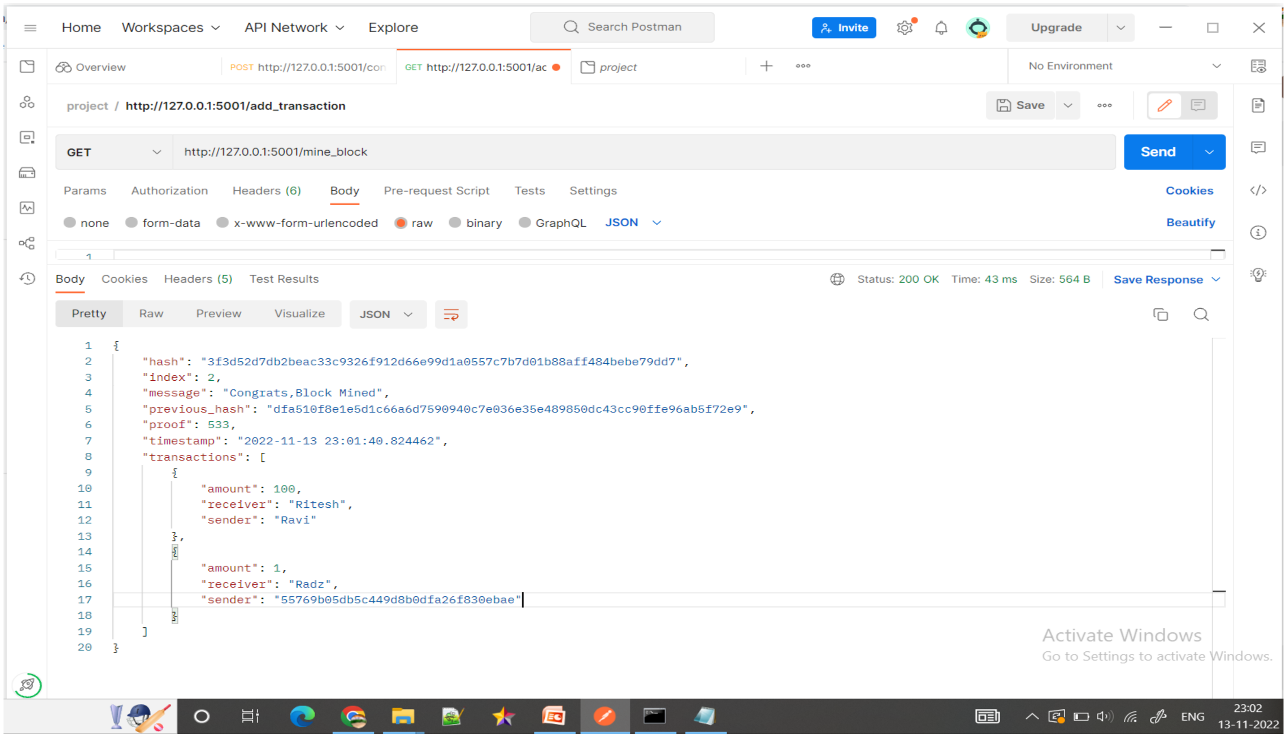
POST http://127.0.0.1:5001/connect_node

Body: ● raw JSON

```
{
    "nodes":["http://127.0.0.1:5002",
            "http://127.0.0.1:5003"
            ]

}
```

Status: 201 CREATED Time: 24 ms Size: 325 B

```
{
    "message": "All the nodes are now connected, The radcoin blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5003",
        "127.0.0.1:5002"
    ]
}
```



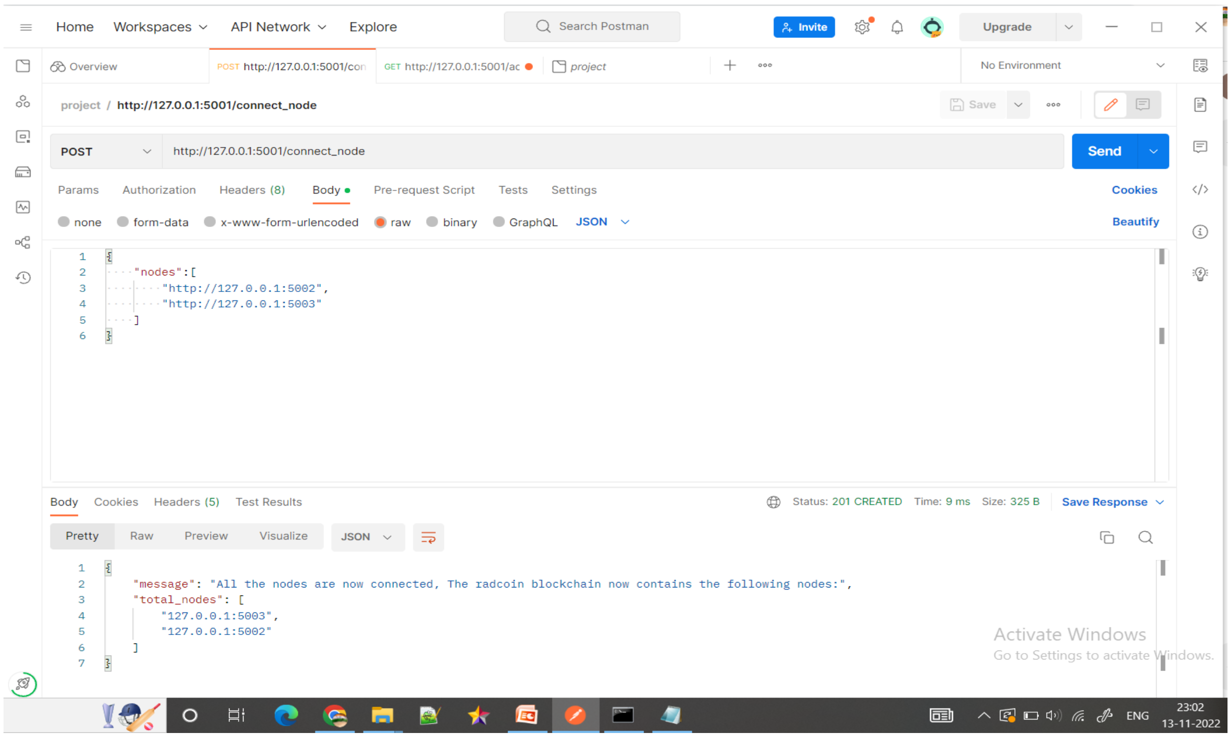
project / http://127.0.0.1:5001/add_transaction

GET http://127.0.0.1:5001/mine_block

Body: ● raw JSON

Status: 200 OK Time: 43 ms Size: 564 B

```
{
    "hash": "3f3d52d7db2beac33c9326f912d66e99d1a0557c7b7d01b88aff484bebe79dd7",
    "index": 2,
    "message": "Congrats,Block Mined",
    "previous_hash": "dfa510f8e1e5d1c66a6d7590940c7e036e35e489850dc43cc90ffe96ab5f72e9",
    "proof": 533,
    "timestamp": "2022-11-13 23:01:40.824462",
    "transactions": [
        {
            "amount": 100,
            "receiver": "Ritesh",
            "sender": "Ravi"
        },
        {
            "amount": 1,
            "receiver": "Radz",
            "sender": "55769b05db5c449d8b0dfa26f830ebae"
        }
    ]
}
```

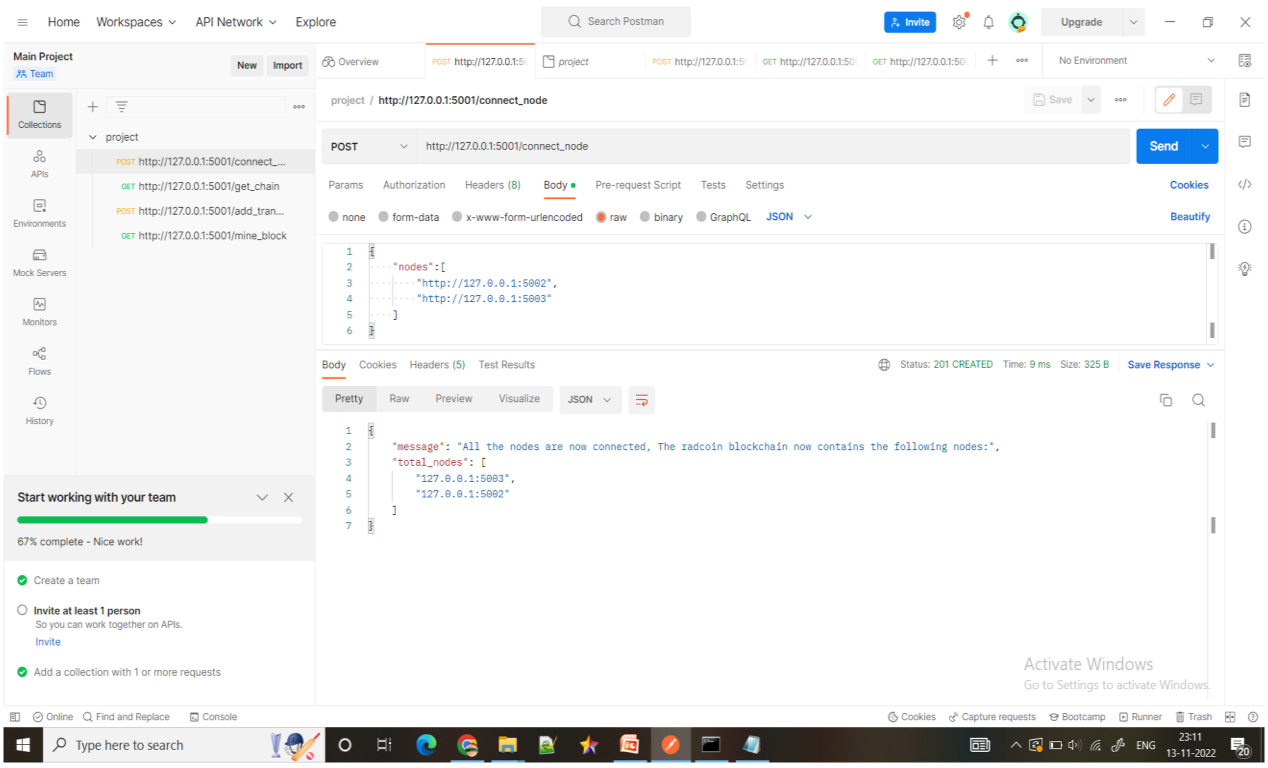

project / http://127.0.0.1:5001/connect_node

POST http://127.0.0.1:5001/connect_node

```
{
    "nodes":[
        "http://127.0.0.1:5002",
        "http://127.0.0.1:5003"
    ]
}
```

Status: 201 CREATED Time: 9 ms Size: 325 B

```
{
    "message": "All the nodes are now connected, The radcoin blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5003",
        "127.0.0.1:5002"
    ]
}
```

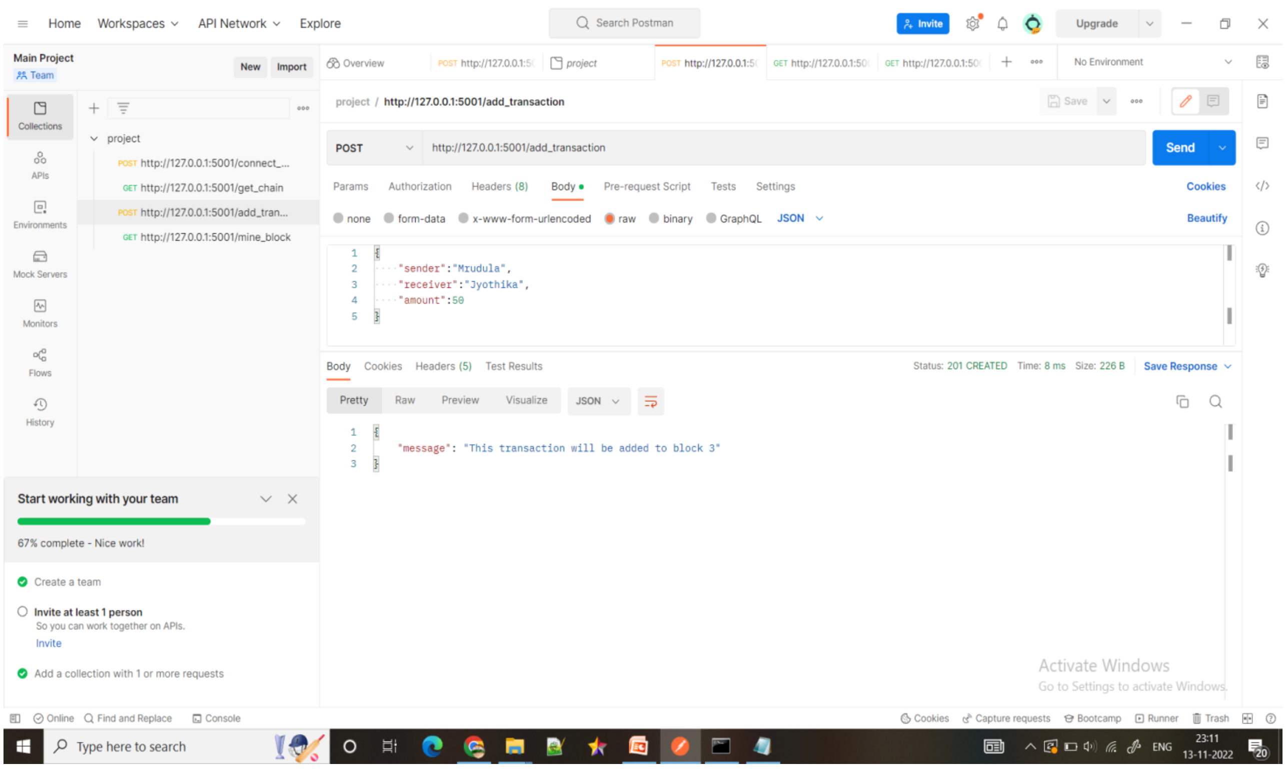

project / http://127.0.0.1:5001/connect_node

POST http://127.0.0.1:5001/connect_node

```
{
    "nodes":[
        "http://127.0.0.1:5002",
        "http://127.0.0.1:5003"
    ]
}
```

Status: 201 CREATED Time: 9 ms Size: 325 B

```
{
    "message": "All the nodes are now connected, The radcoin blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5003",
        "127.0.0.1:5002"
    ]
}
```

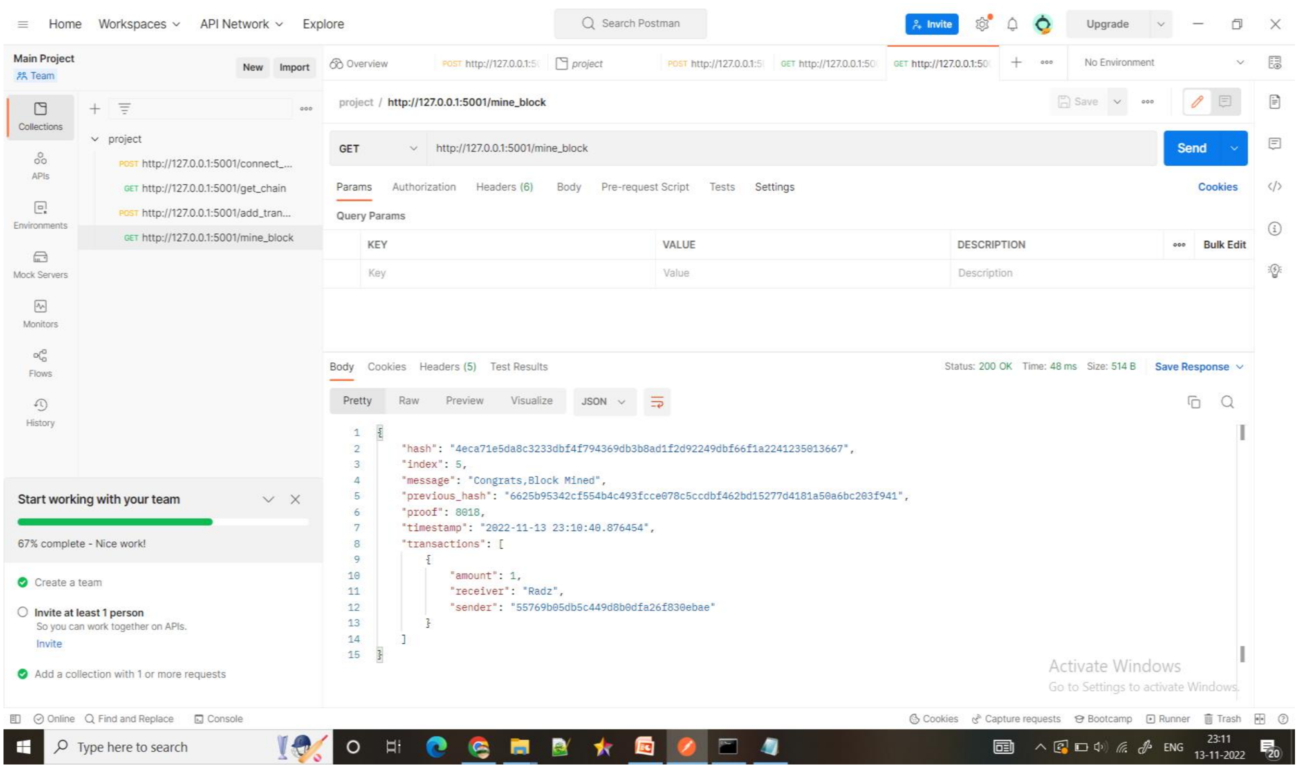


POST http://127.0.0.1:5001/add_transaction

```
{
    "sender":"Mrudula",
    "receiver":"Jyothika",
    "amount":50
}
```

Status: 201 CREATED Time: 8 ms Size: 226 B

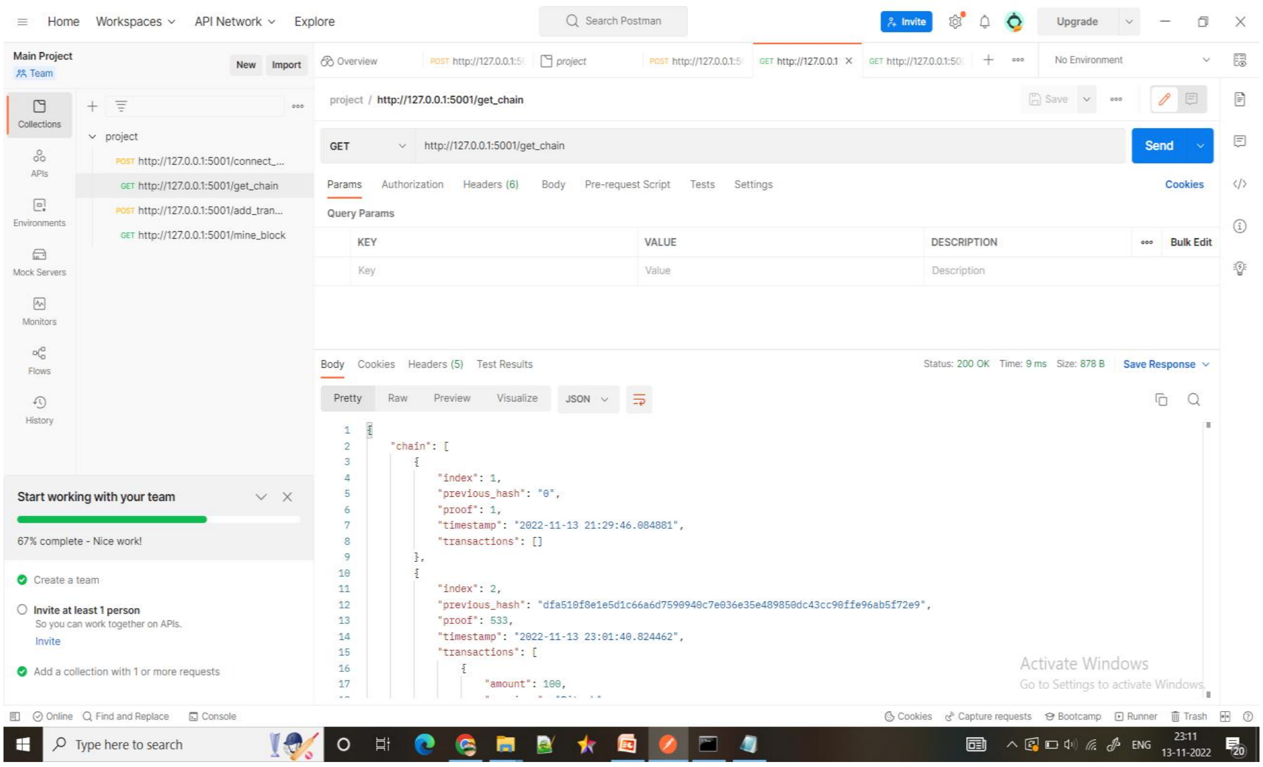
```
{
    "message": "This transaction will be added to block 3"
}
```



GET http://127.0.0.1:5001/mine_block

Status: 200 OK Time: 48 ms Size: 514 B

```
{
    "hash": "4eca71e5da8c3233dbf4f794369db3b8ad1f2d92249dbf66f1a2241235013667",
    "index": 5,
    "message": "Congrats,Block Mined",
    "previous_hash": "6625b95342cf554b4c493fcce078c5ccdbf462bd15277d4181a50a6bc203f941",
    "proof": 8018,
    "timestamp": "2022-11-13 23:10:40.876454",
    "transactions": [
        {
            "amount": 1,
            "receiver": "Radz",
            "sender": "55769b05db5c449d8b0dfa26f830ebae"
        }
    ]
}
```



project
POST http://127.0.0.1:5001/connect_...
GET http://127.0.0.1:5001/get_chain
POST http://127.0.0.1:5001/add_tran...
GET http://127.0.0.1:5001/mine_block

project / http://127.0.0.1:5001/get_chain

GET http://127.0.0.1:5001/get_chain

KEY	VALUE	DESCRIPTION
Key	Value	Description

Start working with your team

67% complete - Nice work!

- Create a team
- Invite at least 1 person
  So you can work together on APIs.
  Invite
- Add a collection with 1 or more requests

Body Cookies Headers (5) Test Results   Status: 200 OK Time: 9 ms Size: 878 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2022-11-13 21:29:46.084881",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "dfa510f8e1e5d1c66a6d7590940c7e036e35e489850dc43cc90ffe96ab5f72e9",
            "proof": 533,
            "timestamp": "2022-11-13 23:01:40.824462",
            "transactions": [
                {
                    "amount": 100,
```

Activate Windows
Go to Settings to activate Windows.

Online Find and Replace Console Cookies Capture requests Bootcamp Runner Trash

ENG 23:11 13-11-2022

9.CONCLUSION AND FUTURE WORK

9.1 Conclusion:

Smart contracts are a critical component of block chain technology, allowing for the execution of self-enforcing agreements in a transparent and decentralized manner. To ensure the integrity and reliability of smart contracts, thorough testing is essential. Test cases for smart contracts in block chain can include verifying deployment, initialization, functionality, security, interaction in block chain can include verifying deployment, initialization, functionality, security, interaction with other contracts, upgradeability, and interoperability. By designing and executing these test cases, developers can identify and fix issues before they impact the system as a whole, ensuring that the smart contract performs as intended and meets the necessary quality standards. In this way, smart contracts can enable a wide range of innovative and secure applications across various industries.

9.2 Future Work:

Looking towards the future, there are several areas of potential future work for smart contracts using block chain technology. Here are some examples:

Increased scalability: One of the biggest challenges facing block chain technology is scalability, and this is especially true for smart contracts. Future work could focus on improving the scalability of smart contracts, allowing them to handle even larger volumes of transactions and users.

Enhanced privacy: While block chain is inherently transparent, there is a growing need for privacy in certain applications. Future work could focus on developing smart contracts with enhanced privacy features, such as zero-knowledge proofs or other cryptographic techniques.

Integration with AI: As artificial intelligence (AI) continues to advance, there is potential for smart contracts to be integrated with AI algorithms, allowing for even more sophisticated and autonomous execution of contracts.

Interoperability between block chains: Currently, different block chain networks are often siloed, which can limit their functionality. Future work could focus on developing interoperability protocols to enable smart contracts to interact across different block chain networks.

Standardization: Currently, there is a lack of standardization in smart contract development, which can make it difficult for developers to create interoperable and compatible contracts. Future work could focus on developing standardization protocols and best practices for smart contract development to enhance compatibility and ease of use.

Overall, future work in smart contracts using block chain will likely focus on improving scalability, enhancing privacy, integrating with AI, improving interoperability, and standardizing development practices to enable even more innovative and secure applications.

10. REFERENCES

1. www.google.com
2. https://scholar.google.co.in/
3. https://www.python.org/
4. https://remix.ethereum.org
5. www.getpostman.com/
6. https://www.kaspersky.com/resource-center/definitions/what-is-cryptocurrency
7. https://en.wikipedia.org/wiki/Smart_contract
8. https://ethereum.org/en/whitepaper/
9. https://docs.soliditylang.org/en/v0.8.10/
10. https://www.springer.com/gp/book/9781484226032
11. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html
12. https://www.penguinrandomhouse.com/books/540144/blockchain-revolution-by-don-tapscott-and-alex-tapscott/
13. https://ethereum.org/en/whitepaper/
14. https://www.oreilly.com/library/view/decentralized-applications/9781491924549/
15. https://www.wiley.com/en-us/The+Business+Blockchain%3A+Promise%2C+Practice%2C+and+Application+of+the+Next+Internet+Technology-p-9781119300311