**PROJECT REPORT**

**ON**

***“SEC Coin”***

Submitted in partial fulfillment of the

Requirement for the award of the degree of

**Bachelor of Computer Application**

** **

**Submitted To:- Submitted By:-**

Dr. Pooja Thakkar Pranav Vinayak

(09417702017)

(Batch: 2017-2020)

**Vivekananda Institute of Professional Studies**

**(Affiliated to Guru Gobind Singh Indraprastha University)**

**CERTIFICATE**

This is to certify that I, Pranav Vinayak of BCA 6th Semester from Vivekananda Institute of Professional Studies, Delhi has presented this project work entitled “SEC Coin”, a cryptocurrency application in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Applications under our supervision and guidance.

**ACKNOWLEDGEMENT**

It is our proud privilege to express our profound gratitude to the entire management of Vivekananda Institute of Professional Studies and teachers of the institute for providing us with the opportunity to avail the excellent facilities and infrastructure. The knowledge and values inculcated have proved to be of immense help at the very start of my career. Special thanks to Hon’ble Founder, Vivekananda Institute of Professional Studies, Delhi for having provided us an excellent infrastructure at VSIT.

I am grateful to Dr. Supriya Madan (Dean, VSIT), and “project guide” for their astute guidance, constant encouragement and sincere support for this project work.

Sincere thanks to all my family members, seniors and friends for their support and assistance throughout the project.

Pranav Vinayak

|  |  |
| --- | --- |
| **Table of Contents** | **Page No.** |
| 1. **INTRODUCTION**    1. **Objective of the System**    2. **Justification and need for the system**    3. **Advantage of the system**   **1.4 Previous work or related systems, how they are used.**  **2 REQUIREMENT ANALYSIS**    **2.1 Analysis Study**   * 1. **User Requirements**   2. **Discussion with IT Experts**   **2.4 Final Requirements**  **3 DESIGN OF THE SYSTEM**   * 1. **Hardware, Software requirements**   2. **System requirements**   3. **Detailed System Specification (Module Wise)**   **3.4 DFDs/Algorithms/Flow Charts**  **4 IMPLEMENTATION & CODING**   * 1. **Operating System**   2. **Languages**   3. **S/W Tools**   **4.4 Coding**  **5 TESTING & TEST RESULTS**   * 1. **Software Testing and Objective of Testing**   **.**  **6 CONCLUSION**   * 1. **Conclusion**   2. **Future Scope**   **6.3 Bibliography** | 7  7  7  9  12  13  13  15  16  17  18  18  18  20  22  24  26  27  28  53  54  57  57  58  59 |

**Table of Figures**

**3 Fig. 3.1** Actor and Use case17

**Fig.3.2** Interaction between user and admin 18 **Fig 3.3** DFD 20

**4** **Fig 4.1** Home page 21

**Fig 4.2** Blockchain 22

**Fig 4.3** Transactions 23

**Fig 4.4** Mining page 28

**Chapter 1**

**INTRODUCTION**

There is no doubt that the era of information and communication technologies has created many golden opportunities in several aspects. One of the fields that benefit from these technologies and online connections is the financial and business sector. A growing number of online users has activated virtual world concepts and created new business phenomena. Thus, new types of trading, transactions and currencies have been arising. One of the remarkable financial forms that have been emerged in the past few years is Cryptocurrency. Cryptocurrency (CC) can be defined as any medium of exchange, apart from real world money, that can be used in many financial transactions whether they are virtual or real transactions. Cryptocurrencies represent valuable and intangible objects which can be used electronically or virtually in different applications and networks such as online social networks, online social games, virtual worlds and peer to peer networks.

**1.1 OBJECTIVE OF THE SYSTEM**

This project will serve the following objectives:-

1. When Cryptocurrency is created, all confirmed transactions are stored in a public ledger. All identities of coin owners are encrypted to ensure the legitimacy of record keeping.
2. The ledger ensures that all transactions between “digital wallets” can calculate an accurate balance. All transactions are checked to make sure that the coins used are owned by the current spender.
3. here are over two billion people with access to the Internet who don't have rights to use to traditional exchange systems. These individuals are clued-in for the Cryptocurrency market.
4. There is no other electronic cash system in which your account is owned by you.
5. Every end user would be the owner as this project is completely decentralized.

**1.2 JUSTIFICATION AND NEED FOR THE SYSTEM**

The cryptocurrency revolution is also spreading to India, where Prime Minister Narendra Modi has reduced circulation of cash bills to steer the country towards electronic payment ratification. The Reserve Bank of India is now looking into the newest wave of the future -- cryptocurrency.

A year ago, the Indian government decided to take harsh measures against “black money”, funds earned on the black market on which income and other taxes have been evaded, and tax evasion by removing two of their highest value banknotes from circulation resulting in removing over 22 billion banknotes in circulation. As a result, citizens worried about losing their savings, switched to cryptocurrency to preserve their funds resulting in a trading volume spike of cryptocurrency. Since cryptocurrency is decentralized, Indian regulators are currently working on a legal framework regulating cryptocurrencies such as Bitcoin as well as the central bank of India is developing their own blockchain resulting in its very own cryptocurrency that will be called “Lakshmi”.

Blockchain technology has also enabled companies to change the way they operate digitally. Through Initial Coin offerings (ICOs) companies are offering their digital tokens for sale. While many companies raising funds through ICOs are unregulated and lack validity, companies such as 1World Online, an established Silicon Valley company, already have a working product.

**1.3 Advantages of the system**

1. Transactions

In traditional business dealings, brokers, agents, and legal representatives can add significant complication and expense to what should otherwise be a straightforward transaction. There’s paperwork, brokerage fees, commissions, and any number of other special conditions which may apply.

One of the advantages of cryptocurrency transactions is that they are one-to-one affairs, taking place on a peer-to-peer networking structure that makes “cutting out the middle man” a standard practice. This leads to greater clarity in establishing audit trails, less confusion over who should pay what to whom, and greater accountability, in that the two parties involved in a transaction each know who they are.

2. Asset Transfers

One financial analyst describes the cryptocurrency blockchain as resembling a “large property rights database,” which can on one level be used to execute and enforce two-party contracts on commodities like automobiles or real estate. But the blockchain cryptocurrency ecosystem may also be used to facilitate specialist modes of transfer.

For example, cryptocurrency contracts can be designed to add third party approvals, make reference to external facts, or be completed at a specified date or time in the future. And since you as the cryptocurrency holder have exclusive governance of your account, this minimizes the time and expense involved in making asset transfers.

3. More Confidential Transactions

Under cash/credit systems, your entire transaction history may become a reference document for the bank or credit agency involved, each time you make a transaction. At the simplest level, this might involve a check on your account balances, to ensure that sufficient funds are available. For more complex or business-critical transactions, a more thorough examination of your financial history might be required.

Another one of the great advantages of cryptocurrency is that each transaction you make is a unique exchange between two parties, the terms of which may be negotiated and agreed in each case. What’s more, the exchange of information is done on a “push” basis, whereby you can transmit exactly what you wish to send to the recipient – and nothing besides that.

This guards the privacy of your financial history and protects you from the threat of account or identity theft which is greater under the traditional system, where your information may be exposed at any point in the transaction chain.

4. Transaction Fees

You’ve no doubt read your monthly account statements from the bank or credit card company, and balked at the level of fees imposed for writing checks, transferring funds, or breathing in the general direction of the finance houses involved. Transaction fees can take a significant bite out of your assets – especially if you’re performing a lot of transactions in a month.

Since the data miners (remote and separate computer systems) that do the number crunching which generates Bitcoin and other cryptocurrencies receive their compensation from the cryptocurrency network involved, transaction fees usually don’t apply.

There may be some external fees involved if you engage the services of a third-party management service to maintain your cryptocurrency wallet, but another one of the advantages of cryptocurrency is that they are still likely to be much less than the transaction charges incurred by traditional financial systems.

5. Greater Access to Credit

Digital data transfer and the internet are the media facilitating the exchange in cryptocurrencies. So these services are potentially available to anyone who has a viable data connection, some knowledge of the cryptocurrency networks on offer, and ready access to their relevant websites and portals.

It’s estimated that there are currently 2.2 billion individuals across the world who have access to the Internet or mobile phones, but don’t currently have access to traditional systems of banking or exchange. The cryptocurrency ecosystem holds the potential to make asset transfer and transaction processing available to this vast market of willing consumers – once the required infrastructure (digital and regulatory) is put in place.

6. Easier International Trade

Though largely unrecognized as legal tender on national levels at present, cryptocurrencies by their very nature are not subject to the exchange rates, interest rates, transactions charges, or other levies imposed by a specific country.

And using the peer-to-peer mechanism of the blockchain technology, cross-border transfers and transactions may be conducted without complications over currency exchange fluctuations, and the like.

7. Individual Ownership

In a traditional banking or credit card system, you effectively turn stewardship of your funds over to a third party that can exercise the power of life or death over your assets. Accounts may be closed without notice for infringements of a financial institution’s Terms of Service – requiring you as the account holder to jump through hoops in order to get yourself back into the system.

Perhaps the greatest of all advantages of cryptocurrency is that unless you’ve delegated management of your wallet over to a third party service, you are the sole owner of the corresponding private and public encryption keys that make up your cryptocurrency network identity or address.

8. Adaptability

There are currently over 1200 unique cryptocurrencies or altcoins in circulation worldwide. Many are quite ephemeral, but a significant proportion have been created for specific use cases that illustrate the flexibility of the cryptocurrency phenomenon.

For example, there are “privacy coins” which help mask your identity on the blockchain, and supply chain tokens which can facilitate supply chain operations for various types of industries.

9. Strong Security

Once a Cryptocurrency transfer has been authorized, it can’t be reversed as in the case of the “charge-back” transactions allowed by credit card companies. This is a hedge against fraud which requires a specific agreement to be made between a buyer and seller regarding refunds in the event of a mistake or returns policy.

**1.4 Previous work or related systems; how they are used.**

Bitcoin was the first decentralized cryptocurrency introduced in 2009. Bitcoin uses the blockchain technology and has outperformed gold generating a 155% annualized gain over gold’s 6% annualized loss over the last 5 years. Its price in July 2010 at 0.06/coin USD is now worth over 4000.00/coin USD today, making it one of the biggest investment phenomenons in modern history. Since 2009 blockchain technology has gained momentum. Not only because of the tremendous spike in Bitcoin’s worth, but also through an increased awareness of its importance and greater trust among the investors.

Recently, major banking institutions and technology companies such as Intel, Barclays or Walmart have invested their time and money into the promise of cryptocurrencies like Bitcoin and Ethereum. This has led to countries with weakening currencies to adopt digital currency to take the place of traditional notes that have depreciated. Some of these early adopter countries include Brazil, Colombia, Turkey and Venezuela.

**Chapter 2**

**REQUIREMENT ANALYSIS**

Before we begin a new system it is important to study the system that will be improved or replaced (if there is one).We need to analyze how this system uses hardware, software, network and the people resources to convert data resources, such as transaction data, into information products, such as reports and displays. Thus we should document how the information system activities of input, processing, output, storage and control are accomplished.

**2.1 ANALYSIS STUDY**

1. Decentralized & No Central Authority

In traditional fiat currencies, central authorities and banks, control the financial system. However, with Bitcoin and other cryptocurrencies, these transactions can be processed and validated by a distributed and open network, that is owned by no-one. Unlike centralized banking systems, most cryptocurrencies are decentralized on distributed networks of computers that are spread around the world, also known as nodes. Transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain. The transaction is propagated across the peer-to-peer network and is replicated by every node, reaching a large percentage of the nodes within a few seconds.

2. Anonymous / Pseudo-anonymous

Since there is no need for a central authority, users do not need to identify themselves when transacting with cryptocurrency. When a transaction request is submitted, the decentralized network will check the transaction and verify it and record it on the blockchain accordingly. Cryptocurrencies, like Bitcoin, uses a private key and public key system to authenticate these transactions. This means users can create anonymous digital identities and digital wallets to transact on the decentralized system and still be able to securely authenticate their transactions.

3. Irreversible & Immutable (cannot be undone)

Cryptocurrency transactions are irreversible and immutable. The irreversible and immutable features of cryptocurrency means that it is impossible for anyone but the owner of the respective private key to move their digital assets and that transactions cannot be changed once it is recorded on the blockchain. While it is not impossible to modify the transaction, secure cryptography makes it very difficult for modification, because it requires you to alter most nodes in the blockchain. In order to prevent fraudulent transactions (that cannot be reversed), all transactions are transparently recorded on the blockchain and open to the public.

4. Limited Supply & Scarcity

Fiat currencies (e.g. dollars, euros) have an unlimited supply, as the central banks can issue as much fiat currencies as they want. Central banks often manipulate the value of the countries’ currencies as part of its economic policies. Most countries often manipulate their currency to be inflationary over a period of time. The inflationary nature of fiat currencies would mean a decrease in the value of the currency over time. Therefore, fiat currency holders might bear the cost of the decrease in value and also face the uncertainty of currency manipulation. On the other hand, most cryptocurrencies have a limited and pre-determined supply of the cryptocurrency that is coded into its underlying algorithm when it is created. For example, Bitcoin has a maximum supply of 21 million, and once this limit is reached, no new Bitcoin can be mined. Cryptocurrency intentionally creates scarcity to prevent currency manipulation and the decrease of value over time.

2.2 Feasibility Study

All projects are feasible if they have unlimited resources and infinite time. But the development of software is plagued by the scarcity of resources and difficult delivery rates. It is necessary and prudent to evaluate the feasibility of a project at the earliest possible time. The three considerations are involved in the feasibility analysis.

2.3 Technical Feasibility

Technical feasibility centers on the existing mobile system (hardware, software…etc) and to what extent it can support the proposed addition if the budget is a serious constraint, then the project is judged not feasible. The technical feasibilities are important role in my project because here I am using python 3.8.2, Java script and react. It requires visual studio code (software) to develop this application. An easily available software and easy to use.

2.4 Economical Feasibility

There is no established framework for recourse to customer problems/ disputes resolution as payments by cryptocurrencies take place on a peer-to-peer basis without an authorised central agency which regulates such payments. There exists a high possibility of its usage for illicit activities, including tax avoidance. But very recently, the supreme court has declared the trade and use of cryptocurrencies as legal

**2.5** Operational Feasibility

People with just a little bit of training can get used to the idea of cryptocurrencies. Any lehman can easily download the api to trade and mine the cryptocurrency

**2.2 USER REQUIREMENTS**

A cryptocurrency (or crypto currency) is a digital asset designed to work as a medium of exchange wherein individual coin ownership records are stored in a digital ledger or computerized database using strong cryptography to secure transaction record entries. The api for a cryptocurrency should be:-

1. Easy to use
2. Should provide a good user interface
3. Security should be maintained
4. Redundancy must be controlled
5. Understandable by a Lehman

**Inquiry Form/Interviews**

Q) What kind of system?

A) A fast and reliable system

Q) User satisfaction from the content?

The content is monetary and should be understood easily if the user has some pre requisite knowledge

Q) Data security?

Data should be in encrypted format so that no third party could access the user credentials.

Q) Using of system

A system that is easy to use and with less complications.

Q) What kind of interface?

A more user oriented interface that is basically designed for the users only

Q) Trustworthiness of the user credentials?

The users would find it impossible to tamper with the system

**2.3 Discussion with IT experts**

Creating an IT project for a beginner can at times become a challenging task. So, the discussion with the veterans in the field of IT becomes an important task which might lead to some great benefits for the developer. Some IT developers might consider it as an time wasting process but, they would be missing out on a very important lesson ignoring this step. The people who are in the IT field for a long time knows the mindset of the user pretty well and might help in giving some important pointers which in turn would help in improvement of the project. Following were the outcome of the discussion held with our IT experts:

1. Proper validations were proposed
2. User friendly template was taken
3. Security Checks were performed
4. More functionality was added

So, it was a very eventful and important step taken in the development of the project which leads to some interesting improvements in the project.

**2.4 Final Requirements**

**User Oriented:** A system should be more user friendly not of the technical point of view

**Better GUI:** All the elements used in the system should be of interactive in nature that is its look and feel is not so boring that the user could get bored while using it.

**Reliability:** The system should be reliable and fast in processing

**Data security:** Access to the organizational data is not to be granted to any unknown person which is not a part of the transaction

**Confidentiality:** Whatever the user is providing to the organization ,the user has the full rights to modify it and it could be not be accessed/modified without users permission

**Better Management of information:** All the information should be managed that is the flow of the information is to be in right track

**Presentation:** The content that is to be presented to the user is to be presented in such a way that is self explanatory to the user and he/she is satisfied with the data.

**Chapter 3**

**DESIGN OF THE SYSTEM**

**3. Software requirements**

|  |  |
| --- | --- |
| Platform | Platform Independent |
| The Operating System | Windows,MAC,Linux |
| Framework | Eclipse, Apache Tomcat Server |
| Front-End Tool | Java Script, React |
| Back-End Tool | Python 3.8.2 |

**3.1.1 Hardware Requirements**

|  |  |
| --- | --- |
| Processor | Intel Pentium IV 2.9 GHz and above |
| RAM | Minimum 512 mb |
| Graphics | Integrated graphics card |
| Hard Disk | Minimum 500 GB |

**3.2 System Requirements**

To know the detailed system requirements an SRS has to be prepared .Software requirement specification abbreviated as SRS is a means of translating the idea of files into a formal document. The main features of SRS include:

* Establishing the basis for an agreement between the client and the developer.
* Producing a reference for validation of the final product. SRS assist clients in determining if the software meets the requirements.

Mainly there are six requirements which an SRS must satisfy.

(a) It should specify the external behavior.

(b) It should specify the constraints.

(c) It should be easy to change.

(d) It should be a reference tool.

(e) It should record throughout the lifecycle.

(f) It should have the capacity of expectation of an undesired event.

Usually we come across four types of requirement specification

(a) User Interface Requirements

(b) Database Requirements

(c) Functional Requirements

(d) Non-Functional Requirements

**3.2.1 User Interface Requirements**

The application will be running on Windows, MAC and linux platforms. Besides the main menu, this program also will have several GUI structures for some of the features like purchasing and mining coins, showing best path and searching.

**3.2.2 Database Requirements**

The only database required would be a public ledger which would be in the api itself.

**3.2.3 Functional Requirements**

The various functional requirements of the system can be summarized as:-

(a) A home page that is user friendly and ambiguous.

(b) It is easy to look for the desired information.

(c) User can easily do reservation.

**3.2.4 Non-Functional Requirements**

Non-functional requirements define the system properties and constraints that arise through user needs, because of the budgeted constraints or organizational policies, or because of the need for interoperability with other software or due to the external factors such as safety regulations privacy registrations and so on.

**Design Requirements**

The main objectives of **input design** are:

(a) Controlling the amount of input

(b) Keeping the process simple.

(c) The best thing in the input design is to achieve all the objectives mentioned in the simplest manner possible.

The main objectives of **output design** are:

(a) Identifying the specific outputs.

(b) Creating reports for displaying and storing information.

The primary goal of the system analysis is to improve the efficiency of the existing system. For that the study of specification of the requirements is very essential. For the development of the new system, a preliminary survey of the existing system will be conducted. Investigation done whether the upgradation of the system into an application program could solve the problems and eradicate the inefficiency of the existing system

**3.3 Detailed System Specification (Module Wise)**

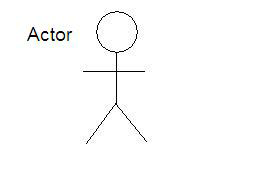
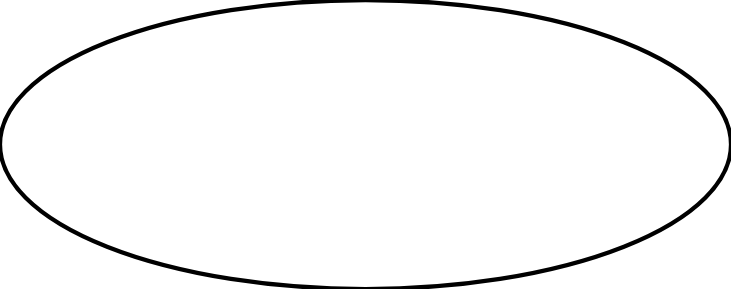
The proposed system consists of mainly 6 important modules and various other sub modules. The main modules of the system are as follows

1. **Login-** When the user opens up the api, he/she would be allotted an unique address with a starting balance of 1000 coins.
2. **Transactions**- The user has the option of transfering coins to another unique adress
3. **Mining****-** The user has an option to mine. He/she can mine the blocks by solving the proof of work puzzle. Each block mined gives the user a reward.
4. **Blockchain:** The user can view the public ledger through the api.

**3.3 THE USE CASE APPROACH**

A use case is a set of scenarios that describe an interaction between a user and a system.

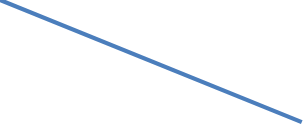
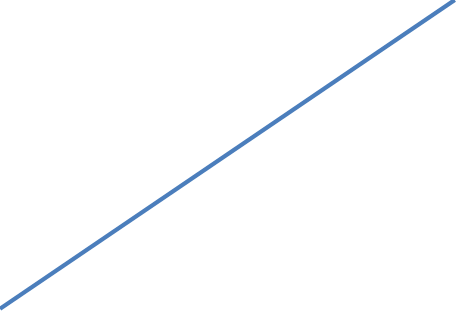
A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

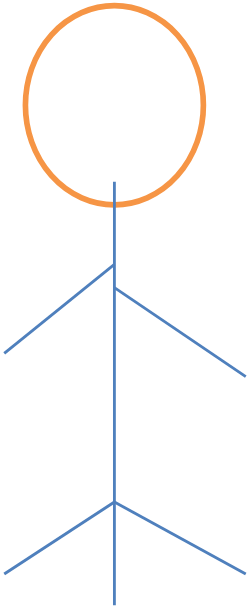
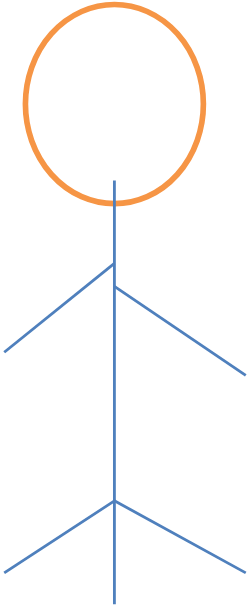
**Fig. 3.1 Actor and Usecase**

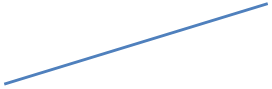
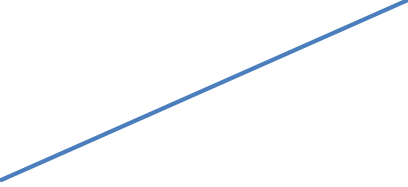
**USE CASE**

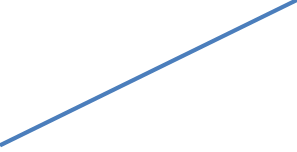
An actor is represents a user or another system that will interact with the system you are modelling.

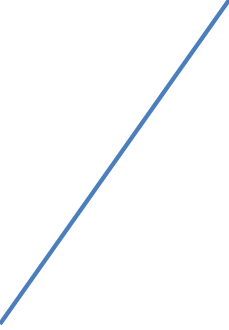
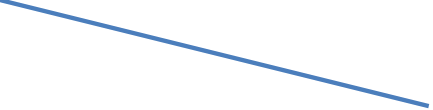
A use case is an external view of the system that represents some action the user might perform in order to complete a task.

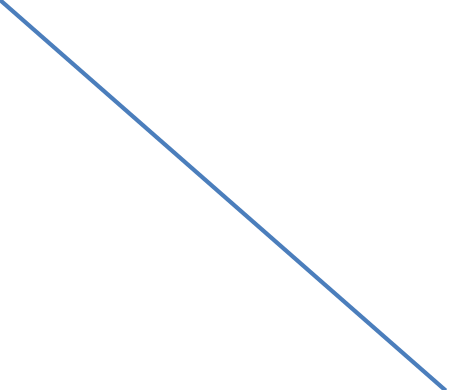
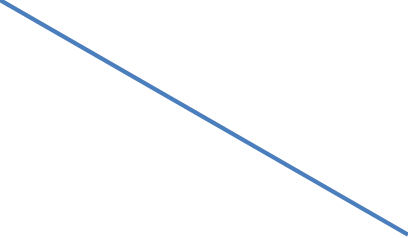












Miner

User

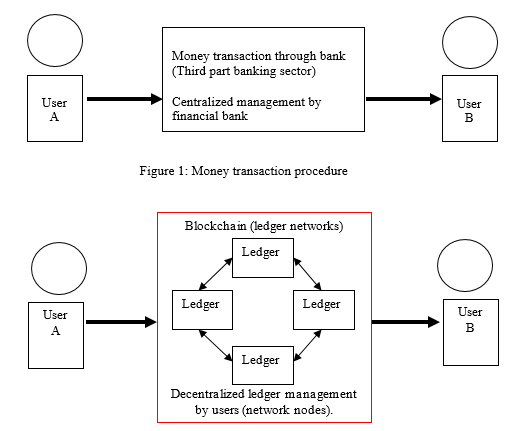
**Fig.3.2 Interaction between user and admin**

In The above diagram there are 2 actors or users who use the system. These two users are as follows:

1. User-The customer is allowed to login to the system, logout of it, make transactions, access the wallet and the public ledger.
2. Miner-The miner can also login and logout of the system. Other than that the miner mines the transactions which gives the blockchain computing power and the miner gains a reward

**3.4 DATA FLOW DIAGRAM** **(DFD)**

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).



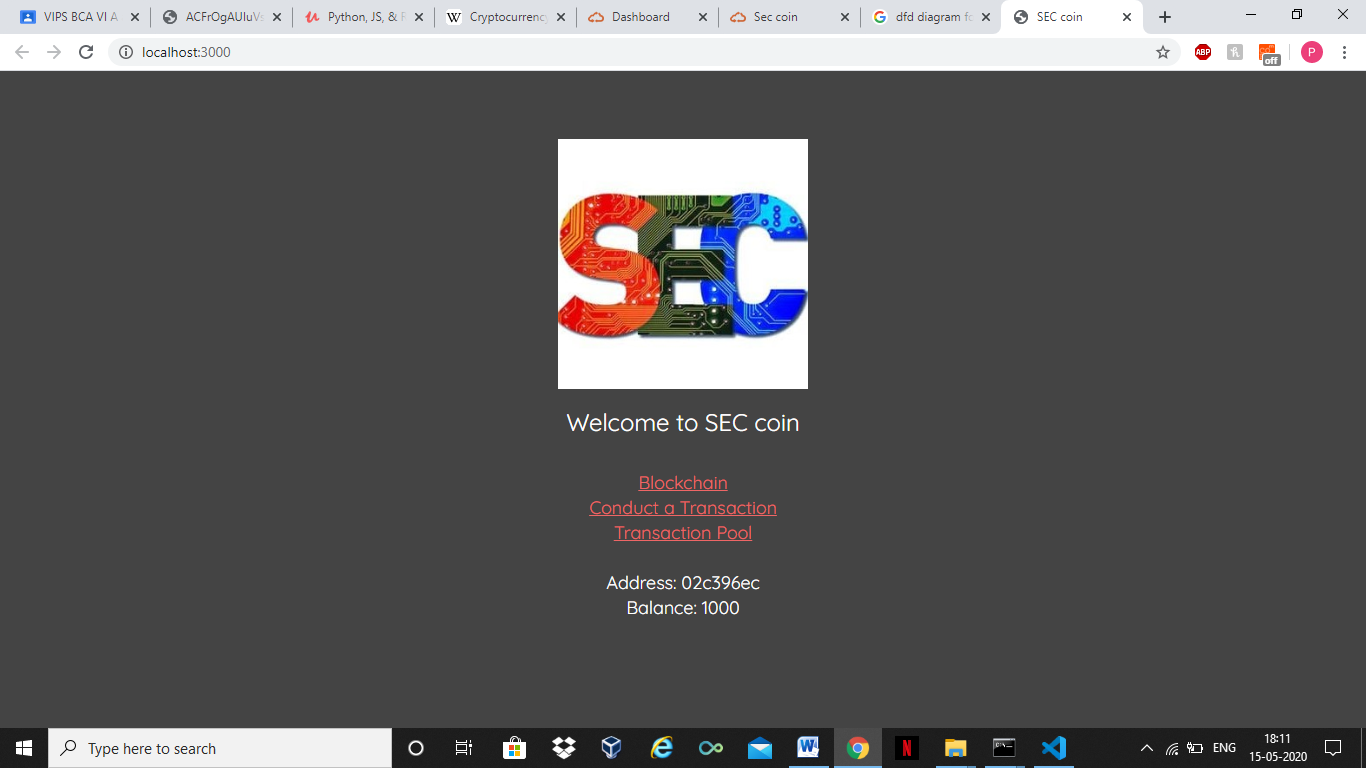
**Fig 3.4 DFD**

A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

**Chapter 4**

**IMPLEMENTATION AND CODING**

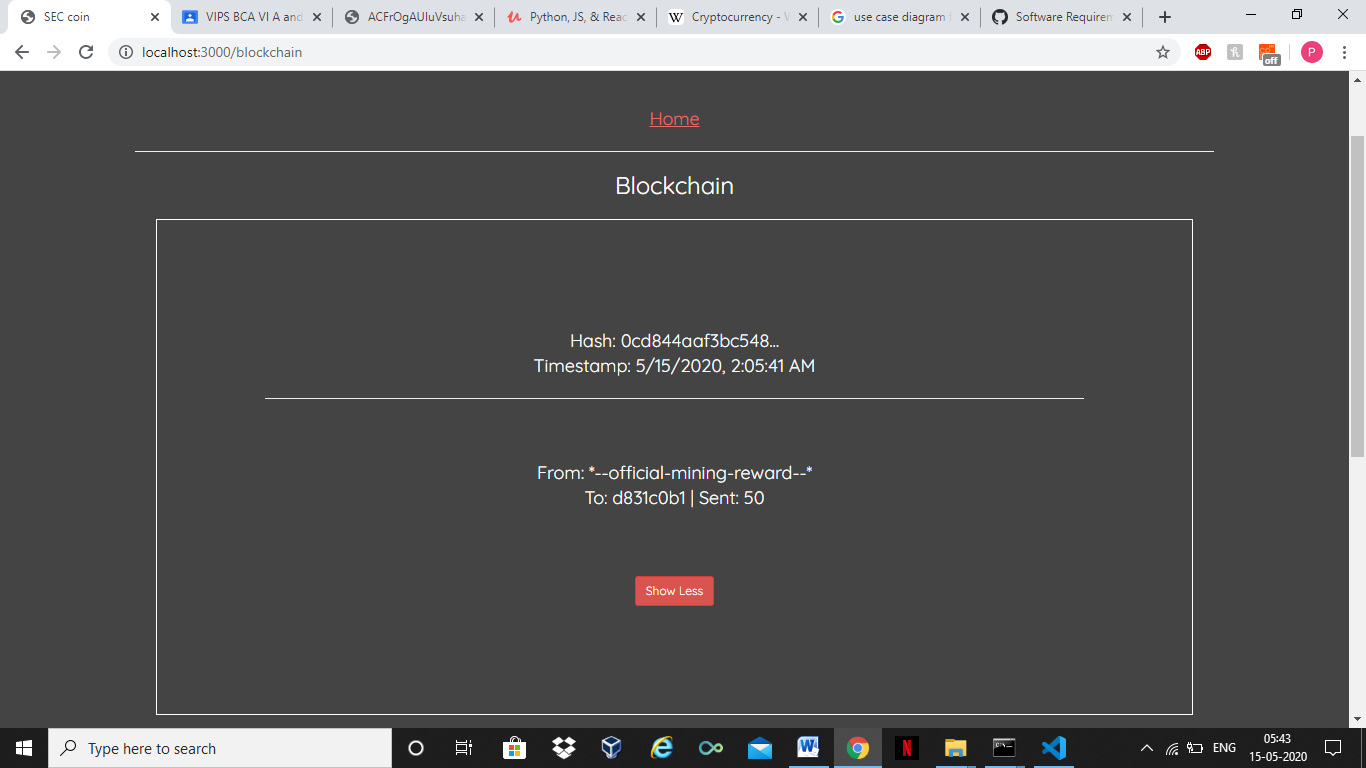
**Home Screen**



**Fig. 4.1 Home page**

This fig shows the home page. It contains the links for the various features of the api.

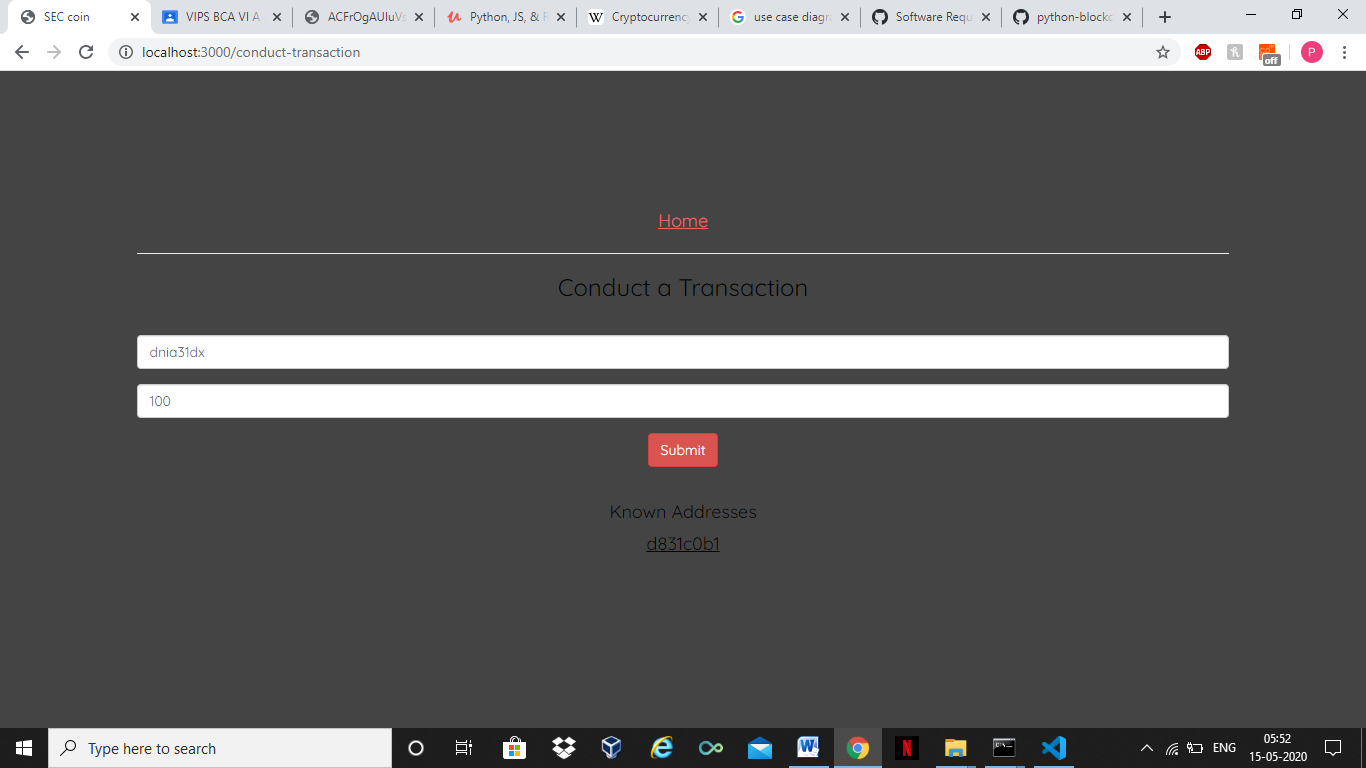
**BLOCKCHAIN**



**Fig 4.2 Blockchain**

This fig shows the public ledger. This shows the newly mined blocks and the transactions which took place.

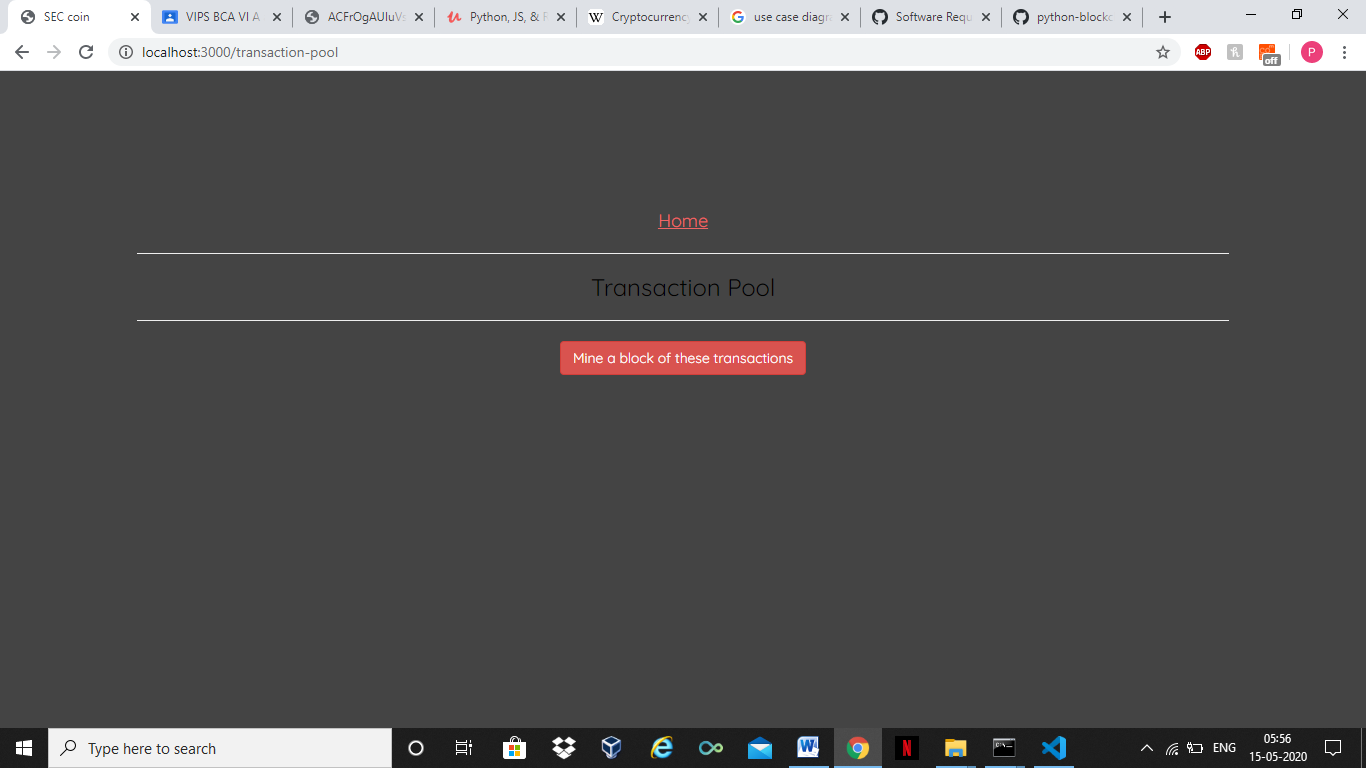
**Transaction Page**



**Fig. 4.3 Transaction page**

This fig shows the transaction page. Here we type an unique address along with the amount to transfer the desired funds to that particular address.

**Mining Interface**



**Fig 4.4**

In this figure, we see the mining interface. We can mine the blocks and earn a reward

**4.1 OPERATING SYSTEM**

**Platform Independent:** Since the project is done completely in python, it also executes main properties of language. The application is platform independent. So the client systems may have vista, Linux, Mac or any other operating system, but they can connect to server easily without any dependencies of OS.

**4.2 Languages used**

**Python 3.8.2**

**Python** is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language was officially discontinued in 2020 (first planned for 2015), and "Python 2.7.18 is the last Python 2.7 release and therefore the last Python 2 release."[30] No more security patches or other improvements will be released for it.[31][32] With Python 2's end-of-life, only Python 3.5.x and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source[34] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

**Java Script**

JavaScript often abbreviated as JS, is a programming language that conforms to the ECMA Script specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it.

**React**

React (also known as React.js or ReactJS) is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies.

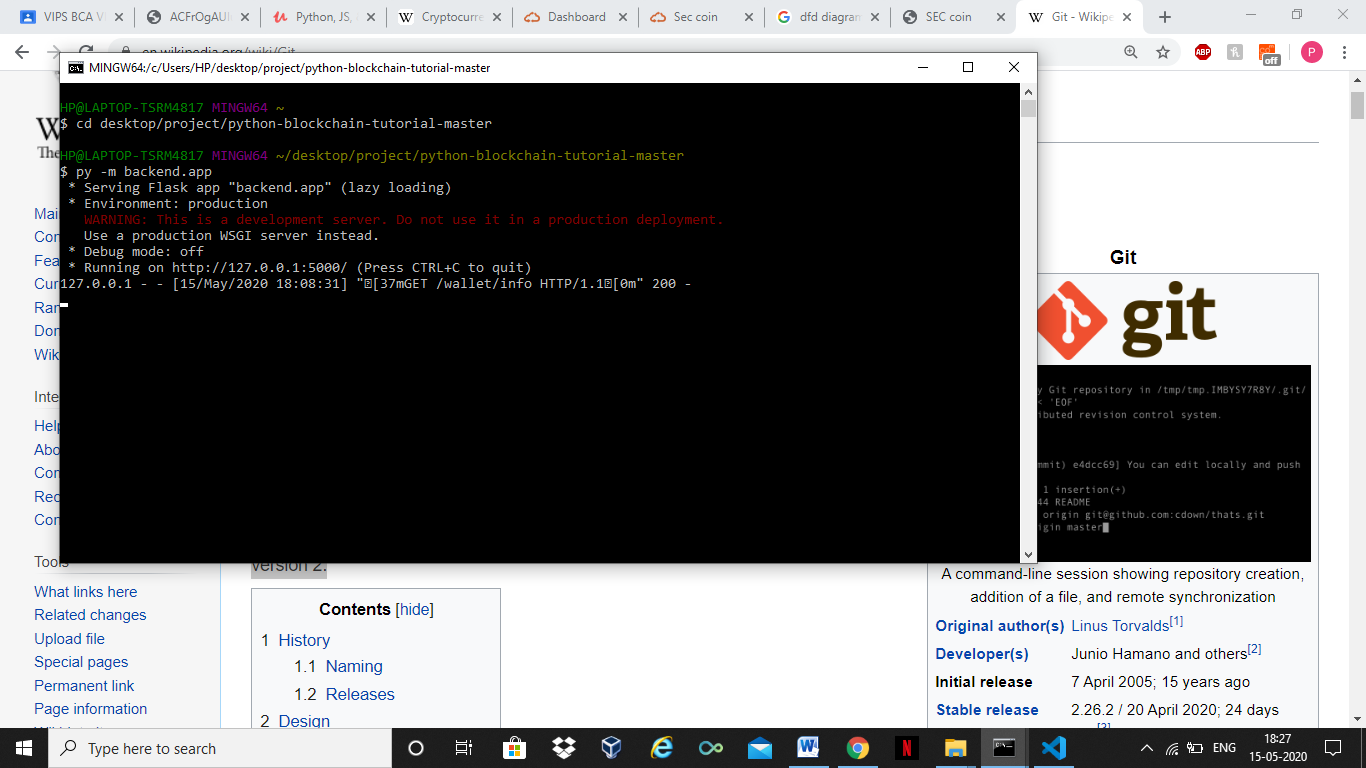
React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with rendering data to the DOM, and so creating React applications usually requires the use of additional libraries for state management and routing. Redux and React Router are respective examples of such libraries.

**4.3 S/W Tools**

**4.3.1 Git Bash**

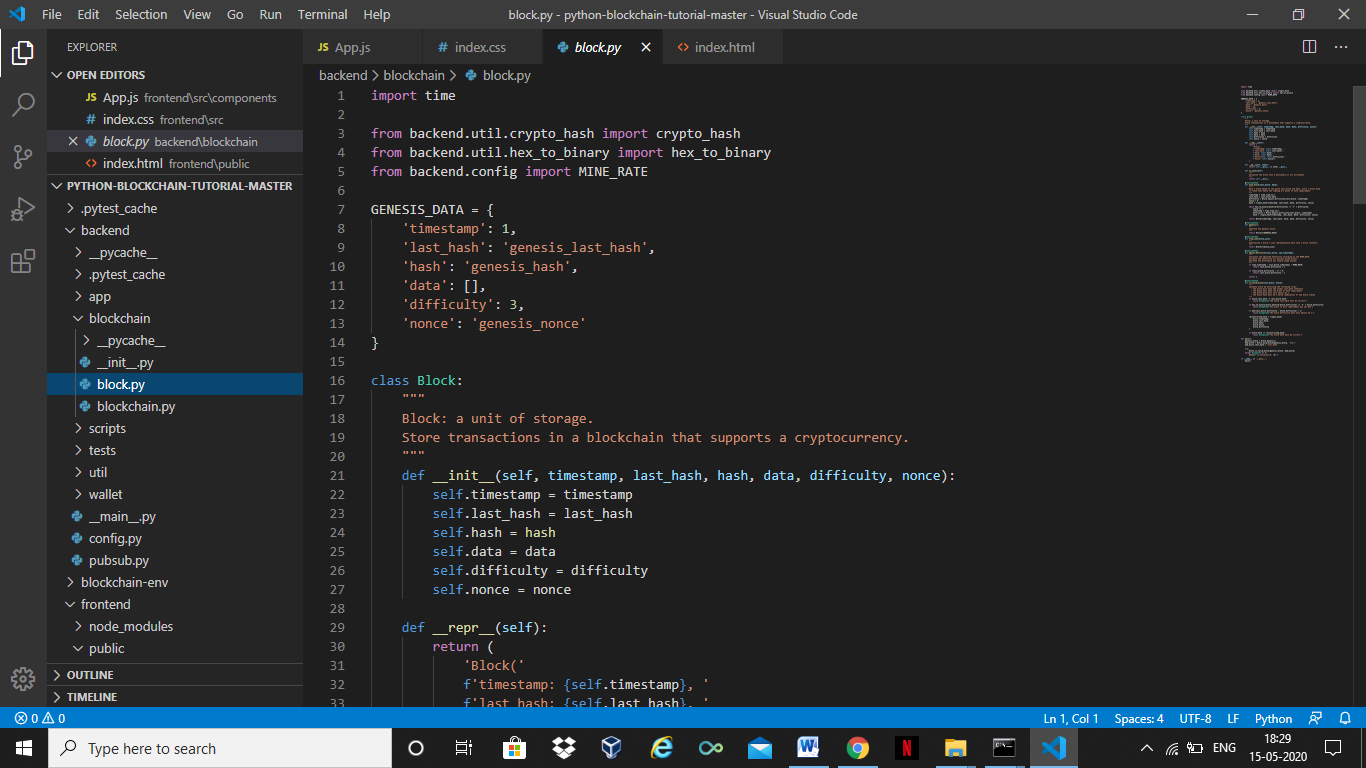
**Git** is a distributed version-control system for tracking changes in source code during software development.[8] It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed,[9] data integrity,[10] and support for distributed, non-linear workflows.[11]

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano. As with most other distributed version-control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.Git is free and open-source software distributed under the terms of the GNU General Public License version 2.4.4 CODING



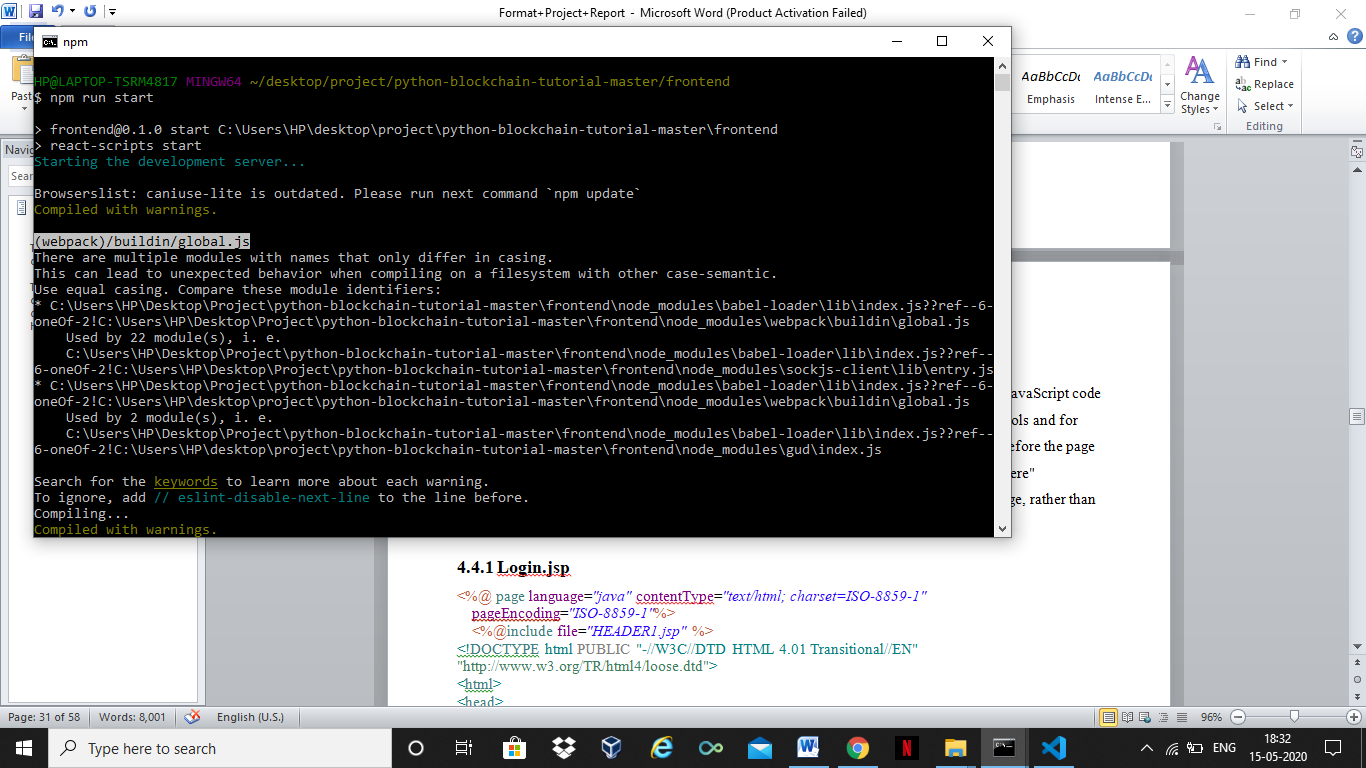
**4.3.2 Visual Studio Code**

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS.It includes embedded Git and support for debugging, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is highly customizable, allowing users to change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The source code is free and open-source, released under the permissive MIT License.The compiled binaries are freeware for any use.In the Stack Overflow 2019 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool, with 50.7% of 87,317 respondents claiming to use it.



**4.3.3 Node**

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm,[6] unifying web-application development around a single programming language, rather than different languages for server- and client-side scripts.



**4.4.1 Block.py**

import time

from backend.util.crypto\_hash import crypto\_hash

from backend.util.hex\_to\_binary import hex\_to\_binary

from backend.config import MINE\_RATE

GENESIS\_DATA = {

    'timestamp': 1,

    'last\_hash': 'genesis\_last\_hash',

    'hash': 'genesis\_hash',

    'data': [],

    'difficulty': 3,

    'nonce': 'genesis\_nonce'

}

class Block:

    def \_\_init\_\_(self, timestamp, last\_hash, hash, data, difficulty, nonce):

        self.timestamp = timestamp

        self.last\_hash = last\_hash

        self.hash = hash

        self.data = data

        self.difficulty = difficulty

        self.nonce = nonce

    def \_\_repr\_\_(self):

        return (

            'Block('

            f'timestamp: {self.timestamp}, '

            f'last\_hash: {self.last\_hash}, '

            f'hash: {self.hash}, '

            f'data: {self.data}, '

            f'difficulty: {self.difficulty}, '

            f'nonce: {self.nonce})'

        )

    def \_\_eq\_\_(self, other):

        return self.\_\_dict\_\_ == other.\_\_dict\_\_

    def to\_json(self):

        return self.\_\_dict\_\_

    @staticmethod

    def mine\_block(last\_block, data):

        timestamp = time.time\_ns()

        last\_hash = last\_block.hash

        difficulty = Block.adjust\_difficulty(last\_block, timestamp)

        nonce = 0

        hash = crypto\_hash(timestamp, last\_hash, data, difficulty, nonce)

        while hex\_to\_binary(hash)[0:difficulty] != '0' \* difficulty:

            nonce += 1

            timestamp = time.time\_ns()

            difficulty = Block.adjust\_difficulty(last\_block, timestamp)

            hash = crypto\_hash(timestamp, last\_hash, data, difficulty, nonce)

        return Block(timestamp, last\_hash, hash, data, difficulty, nonce)

    @staticmethod

    def genesis():

        """

        Generate the genesis block.

        """

        return Block(\*\*GENESIS\_DATA)

    @staticmethod

    def from\_json(block\_json):

        return Block(\*\*block\_json)

    @staticmethod

    def adjust\_difficulty(last\_block, new\_timestamp):

        if (new\_timestamp - last\_block.timestamp) < MINE\_RATE:

            return last\_block.difficulty + 1

        if (last\_block.difficulty - 1) > 0:

            return last\_block.difficulty - 1

        return 1

    @staticmethod

    def is\_valid\_block(last\_block, block):

                if block.last\_hash != last\_block.hash:

            raise Exception('The block last\_hash must be correct')

        if hex\_to\_binary(block.hash)[0:block.difficulty] != '0' \* block.difficulty:

            raise Exception('The proof of work requirement was not met')

        if abs(last\_block.difficulty - block.difficulty) > 1:

            raise Exception('The block difficulty must only adjust by 1')

        reconstructed\_hash = crypto\_hash(

            block.timestamp,

            block.last\_hash,

            block.data,

            block.nonce,

            block.difficulty

        )

        if block.hash != reconstructed\_hash:

            raise Exception('The block hash must be correct')

def main():

    genesis\_block = Block.genesis()

    bad\_block = Block.mine\_block(genesis\_block, 'foo')

    bad\_block.last\_hash = 'evil\_data'

    try:

        Block.is\_valid\_block(genesis\_block, bad\_block)

    except Exception as e:

        print(f'is\_valid\_block: {e}')

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**4.4.2 Blockchain.py**

from backend.blockchain.block import Block

from backend.wallet.transaction import Transaction

from backend.wallet.wallet import Wallet

from backend.config import MINING\_REWARD\_INPUT

class Blockchain:

    def \_\_init\_\_(self):

        self.chain = [Block.genesis()]

    def add\_block(self, data):

        self.chain.append(Block.mine\_block(self.chain[-1], data))

    def \_\_repr\_\_(self):

        return f'Blockchain: {self.chain}'

    def replace\_chain(self, chain):

        if len(chain) <= len(self.chain):

            raise Exception('Cannot replace. The incoming chain must be longer.')

        try:

            Blockchain.is\_valid\_chain(chain)

        except Exception as e:

            raise Exception(f'Cannot replace. The incoming chain is invalid: {e}')

        self.chain = chain

    def to\_json(self):

        """

        Serialize the blockchain into a list of blocks.

        """

        return list(map(lambda block: block.to\_json(), self.chain))

    @staticmethod

    def from\_json(chain\_json):

        blockchain = Blockchain()

        blockchain.chain = list(

            map(lambda block\_json: Block.from\_json(block\_json), chain\_json)

        )

        return blockchain

    @staticmethod

    def is\_valid\_chain(chain):

        if chain[0] != Block.genesis():

            raise Exception('The genesis block must be valid')

        for i in range(1, len(chain)):

            block = chain[i]

            last\_block = chain[i-1]

            Block.is\_valid\_block(last\_block, block)

        Blockchain.is\_valid\_transaction\_chain(chain)

    @staticmethod

    def is\_valid\_transaction\_chain(chain):

        transaction\_ids = set()

        for i in range(len(chain)):

            block = chain[i]

            has\_mining\_reward = False

            for transaction\_json in block.data:

                transaction = Transaction.from\_json(transaction\_json)

                if transaction.id in transaction\_ids:

                    raise Exception(f'Transaction {transaction.id} is not unique')

                transaction\_ids.add(transaction.id)

                if transaction.input == MINING\_REWARD\_INPUT:

                    if has\_mining\_reward:

                        raise Exception(

                            'There can only be one mining reward per block. '\

                            f'Check block with hash: {block.hash}'

                        )

                    has\_mining\_reward = True

                else:

                    historic\_blockchain = Blockchain()

                    historic\_blockchain.chain = chain[0:i]

                    historic\_balance = Wallet.calculate\_balance(

                        historic\_blockchain,

                        transaction.input['address']

                    )

                    if historic\_balance != transaction.input['amount']:

                        raise Exception(

                            f'Transaction {transaction.id} has an invalid '\

                            'input amount'

                        )

                Transaction.is\_valid\_transaction(transaction)

def main():

    blockchain = Blockchain()

    blockchain.add\_block('one')

    blockchain.add\_block('two')

    print(blockchain)

    print(f'blockchain.py \_\_\_name\_\_: {\_\_name\_\_}')

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**4.4.3 test\_block.py**

import pytest

import time

from backend.blockchain.block import Block, GENESIS\_DATA

from backend.config import MINE\_RATE, SECONDS

from backend.util.hex\_to\_binary import hex\_to\_binary

def test\_mine\_block():

    last\_block = Block.genesis()

    data = 'test-data'

    block = Block.mine\_block(last\_block, data)

    assert isinstance(block, Block)

    assert block.data == data

    assert block.last\_hash == last\_block.hash

    assert hex\_to\_binary(block.hash)[0:block.difficulty] == '0' \* block.difficulty

def test\_genesis():

    genesis = Block.genesis()

    assert isinstance(genesis, Block)

    for key, value in GENESIS\_DATA.items():

        getattr(genesis, key) == value

def test\_quickly\_mined\_block():

    last\_block = Block.mine\_block(Block.genesis(), 'foo')

    mined\_block = Block.mine\_block(last\_block, 'bar')

    assert mined\_block.difficulty == last\_block.difficulty + 1

def test\_slowly\_mined\_block():

    last\_block = Block.mine\_block(Block.genesis(), 'foo')

    time.sleep(MINE\_RATE / SECONDS)

    mined\_block = Block.mine\_block(last\_block, 'bar')

    assert mined\_block.difficulty == last\_block.difficulty - 1

def test\_mined\_block\_difficulty\_limits\_at\_1():

    last\_block = Block(

        time.time\_ns(),

        'test\_last\_hash',

        'test\_hash',

        'test\_data',

        1,

        0

    )

    time.sleep(MINE\_RATE / SECONDS)

    mined\_block = Block.mine\_block(last\_block, 'bar')

    assert mined\_block.difficulty == 1

@pytest.fixture

def last\_block():

    return Block.genesis()

@pytest.fixture

def block(last\_block):

    return Block.mine\_block(last\_block, 'test\_data')

def test\_is\_valid\_block(last\_block, block):

    Block.is\_valid\_block(last\_block, block)

def test\_is\_valid\_block\_bad\_last\_hash(last\_block, block):

    block.last\_hash = 'evil\_last\_hash'

    with pytest.raises(Exception, match='last\_hash must be correct'):

        Block.is\_valid\_block(last\_block, block)

def test\_is\_valid\_block\_bad\_proof\_of\_work(last\_block, block):

    block.hash = 'fff'

    with pytest.raises(Exception, match='proof of work requirement was not met'):

        Block.is\_valid\_block(last\_block, block)

def test\_is\_valid\_block\_jumped\_difficulty(last\_block, block):

    jumped\_difficulty = 10

    block.difficulty = jumped\_difficulty

    block.hash = f'{"0" \* jumped\_difficulty}111abc'

    with pytest.raises(Exception, match='difficulty must only adjust by 1'):

        Block.is\_valid\_block(last\_block, block)

def test\_is\_valid\_block\_bad\_block\_hash(last\_block, block):

    block.hash = '0000000000000000bbbabc'

    with pytest.raises(Exception, match='block hash must be correct'):

        Block.is\_valid\_block(last\_block, block)

**4.4.4 test\_blockchain.py**

import pytest

from backend.blockchain.blockchain import Blockchain

from backend.blockchain.block import GENESIS\_DATA

from backend.wallet.wallet import Wallet

from backend.wallet.transaction import Transaction

def test\_blockchain\_instance():

    blockchain = Blockchain()

    assert blockchain.chain[0].hash == GENESIS\_DATA['hash']

def test\_add\_block():

    blockchain = Blockchain()

    data = 'test-data'

    blockchain.add\_block(data)

    assert blockchain.chain[-1].data == data

@pytest.fixture

def blockchain\_three\_blocks():

    blockchain = Blockchain()

    for i in range(3):

        blockchain.add\_block([Transaction(Wallet(), 'recipient', i).to\_json()])

    return blockchain

def test\_is\_valid\_chain(blockchain\_three\_blocks):

    Blockchain.is\_valid\_chain(blockchain\_three\_blocks.chain)

def test\_is\_valid\_chain\_bad\_genesis(blockchain\_three\_blocks):

    blockchain\_three\_blocks.chain[0].hash = 'evil\_hash'

    with pytest.raises(Exception, match='genesis block must be valid'):

        Blockchain.is\_valid\_chain(blockchain\_three\_blocks.chain)

def test\_replace\_chain(blockchain\_three\_blocks):

    blockchain = Blockchain()

    blockchain.replace\_chain(blockchain\_three\_blocks.chain)

    assert blockchain.chain == blockchain\_three\_blocks.chain

def test\_replace\_chain\_not\_longer(blockchain\_three\_blocks):

    blockchain = Blockchain()

    with pytest.raises(Exception, match='The incoming chain must be longer'):

        blockchain\_three\_blocks.replace\_chain(blockchain.chain)

def test\_replace\_chain\_bad\_chain(blockchain\_three\_blocks):

    blockchain = Blockchain()

    blockchain\_three\_blocks.chain[1].hash = 'evil\_hash'

    with pytest.raises(Exception, match='The incoming chain is invalid'):

        blockchain.replace\_chain(blockchain\_three\_blocks.chain)

def test\_valid\_transaction\_chain(blockchain\_three\_blocks):

    Blockchain.is\_valid\_transaction\_chain(blockchain\_three\_blocks.chain)

def test\_is\_valid\_transaction\_chain\_duplicate\_transactions(blockchain\_three\_blocks):

    transaction = Transaction(Wallet(), 'recipient', 1).to\_json()

    blockchain\_three\_blocks.add\_block([transaction, transaction])

    with pytest.raises(Exception, match='is not unique'):

        Blockchain.is\_valid\_transaction\_chain(blockchain\_three\_blocks.chain)

def test\_is\_valid\_transaction\_chain\_multiple\_rewards(blockchain\_three\_blocks):

    reward\_1 = Transaction.reward\_transaction(Wallet()).to\_json()

    reward\_2 = Transaction.reward\_transaction(Wallet()).to\_json()

    blockchain\_three\_blocks.add\_block([reward\_1, reward\_2])

    with pytest.raises(Exception, match='one mining reward per block'):

        Blockchain.is\_valid\_transaction\_chain(blockchain\_three\_blocks.chain)

def test\_is\_valid\_transaction\_chain\_bad\_transaction(blockchain\_three\_blocks):

    bad\_transaction = Transaction(Wallet(), 'recipient', 1)

    bad\_transaction.input['signature'] = Wallet().sign(bad\_transaction.output)

    blockchain\_three\_blocks.add\_block([bad\_transaction.to\_json()])

    with pytest.raises(Exception):

        Blockchain.is\_valid\_transaction\_chain(blockchain\_three\_blocks.chain)

def test\_is\_valid\_transaction\_chain\_bad\_historic\_balance(blockchain\_three\_blocks):

    wallet = Wallet()

    bad\_transaction = Transaction(wallet, 'recipient', 1)

    bad\_transaction.output[wallet.address] = 9000

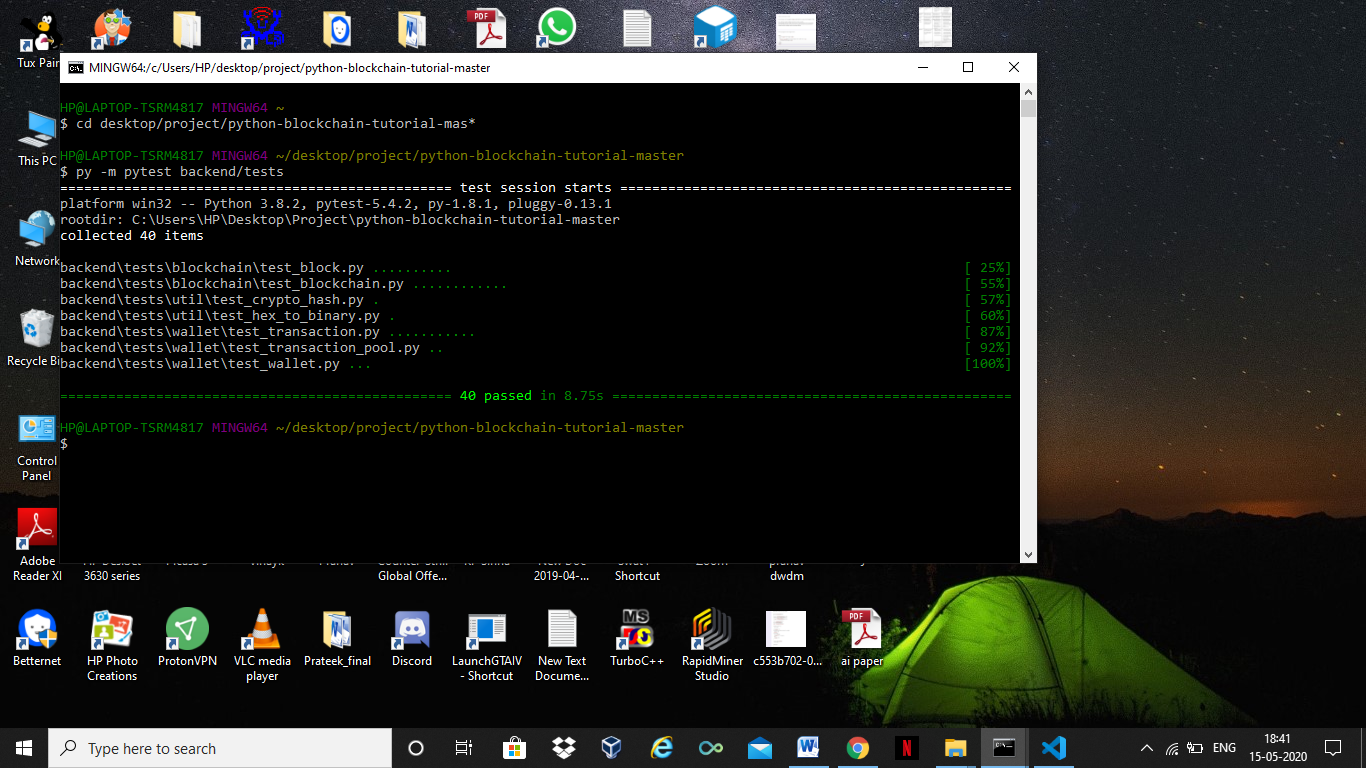
    bad\_transaction.input['amount'] = 9001

    bad\_transaction.input['signature'] = wallet.sign(bad\_transaction.output)

    blockchain\_three\_blocks.add\_block([bad\_transaction.to\_json()])

    with pytest.raises(Exception, match='has an invalid input amount'):

        Blockchain.is\_valid\_transaction\_chain(blockchain\_three\_blocks.chain)



**Fig.4.5 Testing through pytest**

This fig. show the tests we conducted through the pytest module. It tells us that there is not logical error in the code and it is ready to be implemented

**4.4.5 \_\_init\_\_.py**

import os

import requests

import random

from flask import Flask, jsonify, request

from flask\_cors import CORS

from backend.blockchain.blockchain import Blockchain

from backend.wallet.wallet import Wallet

from backend.wallet.transaction import Transaction

from backend.wallet.transaction\_pool import TransactionPool

from backend.pubsub import PubSub

app = Flask(\_\_name\_\_)

CORS(app, resources={ r'/\*': { 'origins': 'http://localhost:3000' } })

blockchain = Blockchain()

wallet = Wallet(blockchain)

transaction\_pool = TransactionPool()

pubsub = PubSub(blockchain, transaction\_pool)

@app.route('/')

def route\_default():

    return 'Welcome to the blockchain'

@app.route('/blockchain')

def route\_blockchain():

    return jsonify(blockchain.to\_json())

@app.route('/blockchain/range')

def route\_blockchain\_range():

    # http://localhost:5000/blockchain/range?start=2&end=5

    start = int(request.args.get('start'))

    end = int(request.args.get('end'))

    return jsonify(blockchain.to\_json()[::-1][start:end])

@app.route('/blockchain/length')

def route\_blockchain\_length():

    return jsonify(len(blockchain.chain))

@app.route('/blockchain/mine')

def route\_blockchain\_mine():

    transaction\_data = transaction\_pool.transaction\_data()

    transaction\_data.append(Transaction.reward\_transaction(wallet).to\_json())

    blockchain.add\_block(transaction\_data)

    block = blockchain.chain[-1]

    pubsub.broadcast\_block(block)

    transaction\_pool.clear\_blockchain\_transactions(blockchain)

    return jsonify(block.to\_json())

@app.route('/wallet/transact', methods=['POST'])

def route\_wallet\_transact():

    transaction\_data = request.get\_json()

    transaction = transaction\_pool.existing\_transaction(wallet.address)

    if transaction:

        transaction.update(

            wallet,

            transaction\_data['recipient'],

            transaction\_data['amount']

        )

    else:

        transaction = Transaction(

            wallet,

            transaction\_data['recipient'],

            transaction\_data['amount']

        )

    pubsub.broadcast\_transaction(transaction)

    return jsonify(transaction.to\_json())

@app.route('/wallet/info')

def route\_wallet\_info():

    return jsonify({ 'address': wallet.address, 'balance': wallet.balance })

@app.route('/known-addresses')

def route\_known\_addresses():

    known\_addresses = set()

    for block in blockchain.chain:

        for transaction in block.data:

            known\_addresses.update(transaction['output'].keys())

    return jsonify(list(known\_addresses))

@app.route('/transactions')

def route\_transactions():

    return jsonify(transaction\_pool.transaction\_data())

ROOT\_PORT = 5000

PORT = ROOT\_PORT

if os.environ.get('PEER') == 'True':

    PORT = random.randint(5001, 6000)

    result = requests.get(f'http://localhost:{ROOT\_PORT}/blockchain')

    result\_blockchain = Blockchain.from\_json(result.json())

    try:

        blockchain.replace\_chain(result\_blockchain.chain)

        print('\n -- Successfully synchronized the local chain')

    except Exception as e:

        print(f'\n -- Error synchronizing: {e}')

if os.environ.get('SEED\_DATA') == 'True':

    for i in range(10):

        blockchain.add\_block([

            Transaction(Wallet(), Wallet().address, random.randint(2, 50)).to\_json(),

            Transaction(Wallet(), Wallet().address, random.randint(2, 50)).to\_json()

        ])

    for i in range(3):

        transaction\_pool.set\_transaction(

            Transaction(Wallet(), Wallet().address, random.randint(2, 50))

        )

app.run(port=PORT)

The \_\_init\_\_.py file would be executed when we run the application

**4.4.6 Average\_block\_rate.py**

import time

from backend.blockchain.blockchain import Blockchain

from backend.config import SECONDS

blockchain = Blockchain()

times = []

for i in range(1000):

    start\_time = time.time\_ns()

    blockchain.add\_block(i)

    end\_time = time.time\_ns()

    time\_to\_mine = (end\_time - start\_time) / SECONDS

    times.append(time\_to\_mine)

    average\_time = sum(times) / len(times)

    print(f'New block difficulty: {blockchain.chain[-1].difficulty}')

    print(f'Time to mine new block: {time\_to\_mine}s')

    print(f'Average time to add blocks: {average\_time}s\n')

**4.4.7 test\_app.py**

import requests

import time

from backend.wallet.wallet import Wallet

BASE\_URL = 'http://localhost:5000'

def get\_blockchain():

    return requests.get(f'{BASE\_URL}/blockchain').json()

def get\_blockchain\_mine():

    return requests.get(f'{BASE\_URL}/blockchain/mine').json()

def post\_wallet\_transact(recipient, amount):

    return requests.post(

        f'{BASE\_URL}/wallet/transact',

        json={ 'recipient': recipient, 'amount': amount }

    ).json()

def get\_wallet\_info():

    return requests.get(f'{BASE\_URL}/wallet/info').json()

start\_blockchain = get\_blockchain()

print(f'start\_blockchain: {start\_blockchain}')

recipient = Wallet().address

post\_wallet\_transact\_1 = post\_wallet\_transact(recipient, 21)

print(f'\npost\_wallet\_transact\_1: {post\_wallet\_transact\_1}')

time.sleep(1)

post\_wallet\_transact\_2 = post\_wallet\_transact(recipient, 13)

print(f'\npost\_wallet\_transact\_2: {post\_wallet\_transact\_2}')

time.sleep(1)

mined\_block = get\_blockchain\_mine()

print(f'\nmined\_block: {mined\_block}')

wallet\_info = get\_wallet\_info()

print(f'\nwallet\_info: {wallet\_info}')

**4.4.8 Transactions.py**

import time

import uuid

from backend.wallet.wallet import Wallet

from backend.config import MINING\_REWARD, MINING\_REWARD\_INPUT

class Transaction:

    """

    Document of an exchange in currency from a sender to one

    or more recipients.

    """

    def \_\_init\_\_(

        self,

        sender\_wallet=None,

        recipient=None,

        amount=None,

        id=None,

        output=None,

        input=None

    ):

        self.id = id or str(uuid.uuid4())[0:8]

        self.output = output or self.create\_output(

            sender\_wallet,

            recipient,

            amount

        )

        self.input = input or self.create\_input(sender\_wallet, self.output)

    def create\_output(self, sender\_wallet, recipient, amount):

        """

        Structure the output data for the transaction.

        """

        if amount > sender\_wallet.balance:

            raise Exception('Amount exceeds balance')

        output = {}

        output[recipient] = amount

        output[sender\_wallet.address] = sender\_wallet.balance - amount

        return output

    def create\_input(self, sender\_wallet, output):

        """

        Structure the input data for the transaction.

        Sign the transaction and include the sender's public key and address

        """

        return {

            'timestamp': time.time\_ns(),

            'amount': sender\_wallet.balance,

            'address': sender\_wallet.address,

            'public\_key': sender\_wallet.public\_key,

            'signature': sender\_wallet.sign(output)

        }

    def update(self, sender\_wallet, recipient, amount):

        """

        Update the transaction with an existing or new recipient.

        """

        if amount > self.output[sender\_wallet.address]:

            raise Exception('Amount exceeds balance')

        if recipient in self.output:

            self.output[recipient] = self.output[recipient] + amount

        else:

            self.output[recipient] = amount

        self.output[sender\_wallet.address] = \

            self.output[sender\_wallet.address] - amount

        self.input = self.create\_input(sender\_wallet, self.output)

    def to\_json(self):

        """

        Serialize the transaction.

        """

        return self.\_\_dict\_\_

    @staticmethod

    def from\_json(transaction\_json):

        """

        Deserialize a transaction's json representation back into a

        Transaction instance

        """

        return Transaction(\*\*transaction\_json)

    @staticmethod

    def is\_valid\_transaction(transaction):

        """

        Validate a transaction.

        Raise an exception for invalid transactions.

        """

        if transaction.input == MINING\_REWARD\_INPUT:

            if list(transaction.output.values()) != [MINING\_REWARD]:

                raise Exception('Invalid mining reward')

            return

        output\_total = sum(transaction.output.values())

        if transaction.input['amount'] != output\_total:

            raise Exception('Invalid transaction output values')

        if not Wallet.verify(

            transaction.input['public\_key'],

            transaction.output,

            transaction.input['signature']

        ):

            raise Exception('Invalid signature')

    @staticmethod

    def reward\_transaction(miner\_wallet):

        """

        Generate a reward transaction that award the miner.

        """

        output = {}

        output[miner\_wallet.address] = MINING\_REWARD

        return Transaction(input=MINING\_REWARD\_INPUT, output=output)

def main():

    transaction = Transaction(Wallet(), 'recipient', 15)

    print(f'transaction.\_\_dict\_\_: {transaction.\_\_dict\_\_}')

    transaction\_json = transaction.to\_json()

    restored\_transaction = Transaction.from\_json(transaction\_json)

    print(f'restored\_transaction.\_\_dict\_\_: {restored\_transaction.\_\_dict\_\_}')

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**4.4.9 Wallet.py**

import json

import uuid

from backend.config import STARTING\_BALANCE

from cryptography.hazmat.backends import default\_backend

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives.asymmetric.utils import (

    encode\_dss\_signature,

    decode\_dss\_signature

)

from cryptography.hazmat.primitives import hashes, serialization

from cryptography.exceptions import InvalidSignature

class Wallet:

    """

    An individual wallet for a miner.

    Keeps track of the miner's balance.

    Allows a miner to authorize transactions.

    """

    def \_\_init\_\_(self, blockchain=None):

        self.blockchain = blockchain

        self.address = str(uuid.uuid4())[0:8]

        self.private\_key = ec.generate\_private\_key(

            ec.SECP256K1(),

            default\_backend()

        )

        self.public\_key = self.private\_key.public\_key()

        self.serialize\_public\_key()

    @property

    def balance(self):

        return Wallet.calculate\_balance(self.blockchain, self.address)

    def sign(self, data):

        """

        Generate a signature based on the data using the local private key.

        """

        return decode\_dss\_signature(self.private\_key.sign(

            json.dumps(data).encode('utf-8'),

            ec.ECDSA(hashes.SHA256())

        ))

    def serialize\_public\_key(self):

        """

        Reset the public key to its serialized version.

        """

        self.public\_key = self.public\_key.public\_bytes(

            encoding=serialization.Encoding.PEM,

            format=serialization.PublicFormat.SubjectPublicKeyInfo

        ).decode('utf-8')

    @staticmethod

    def verify(public\_key, data, signature):

        """

        Verify a signature based on the original public key and data.

        """

        deserialized\_public\_key = serialization.load\_pem\_public\_key(

            public\_key.encode('utf-8'),

            default\_backend()

        )

        (r, s) = signature

        try:

            deserialized\_public\_key.verify(

                encode\_dss\_signature(r, s),

                json.dumps(data).encode('utf-8'),

                ec.ECDSA(hashes.SHA256())

            )

            return True

        except InvalidSignature:

            return False

    @staticmethod

    def calculate\_balance(blockchain, address):

        """

        Calculate the balance of the given address considering the transaction

        data within the blockchain.

        The balance is found by adding the output values that belong to the

        address since the most recent transaction by that address.

        """

        balance = STARTING\_BALANCE

        if not blockchain:

            return balance

        for block in blockchain.chain:

            for transaction in block.data:

                if transaction['input']['address'] == address:

                    # Any time the address conducts a new transaction it resets

                    # its balance

                    balance = transaction['output'][address]

                elif address in transaction['output']:

                    balance += transaction['output'][address]

        return balance

def main():

    wallet = Wallet()

    print(f'wallet.\_\_dict\_\_: {wallet.\_\_dict\_\_}')

    data = { 'foo': 'bar' }

    signature = wallet.sign(data)

    print(f'signature: {signature}')

    should\_be\_valid = Wallet.verify(wallet.public\_key, data, signature)

    print(f'should\_be\_valid: {should\_be\_valid}')

    should\_be\_invalid = Wallet.verify(Wallet().public\_key, data, signature)

    print(f'should\_be\_invalid: {should\_be\_invalid}')

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**4.4.10 pubsub.py**

import time

from pubnub.pubnub import PubNub

from pubnub.pnconfiguration import PNConfiguration

from pubnub.callbacks import SubscribeCallback

from backend.blockchain.block import Block

from backend.wallet.transaction import Transaction

pnconfig = PNConfiguration()

pnconfig.subscribe\_key = 'sub-c-3832dbc4-9181-11ea-8e98-72774568d584'

pnconfig.publish\_key = 'pub-c-f1dd230c-561c-43fa-8007-7b8052ee4735'

CHANNELS = {

    'TEST': 'TEST',

    'BLOCK': 'BLOCK',

    'TRANSACTION': 'TRANSACTION'

}

class Listener(SubscribeCallback):

    def \_\_init\_\_(self, blockchain, transaction\_pool):

        self.blockchain = blockchain

        self.transaction\_pool = transaction\_pool

    def message(self, pubnub, message\_object):

        print(f'\n-- Channel: {message\_object.channel} | Message: {message\_object.message}')

        if message\_object.channel == CHANNELS['BLOCK']:

            block = Block.from\_json(message\_object.message)

            potential\_chain = self.blockchain.chain[:]

            potential\_chain.append(block)

            try:

                self.blockchain.replace\_chain(potential\_chain)

                self.transaction\_pool.clear\_blockchain\_transactions(

                    self.blockchain

                )

                print('\n -- Successfully replaced the local chain')

            except Exception as e:

                print(f'\n -- Did not replace chain: {e}')

        elif message\_object.channel == CHANNELS['TRANSACTION']:

            transaction = Transaction.from\_json(message\_object.message)

            self.transaction\_pool.set\_transaction(transaction)

            print('\n -- Set the new transaction in the transaction pool')

class PubSub():

    """

    Handles the publish/subscribe layer of the application.

    Provides communication between the nodes of the blockchain network.

    """

    def \_\_init\_\_(self, blockchain, transaction\_pool):

        self.pubnub = PubNub(pnconfig)

        self.pubnub.subscribe().channels(CHANNELS.values()).execute()

        self.pubnub.add\_listener(Listener(blockchain, transaction\_pool))

    def publish(self, channel, message):

        """

        Publish the message object to the channel.

        """

        self.pubnub.publish().channel(channel).message(message).sync()

    def broadcast\_block(self, block):

        """

        Broadcast a block object to all nodes.

        """

        self.publish(CHANNELS['BLOCK'], block.to\_json())

    def broadcast\_transaction(self, transaction):

        """

        Broadcast a transaction to all nodes.

        """

        self.publish(CHANNELS['TRANSACTION'], transaction.to\_json())

def main():

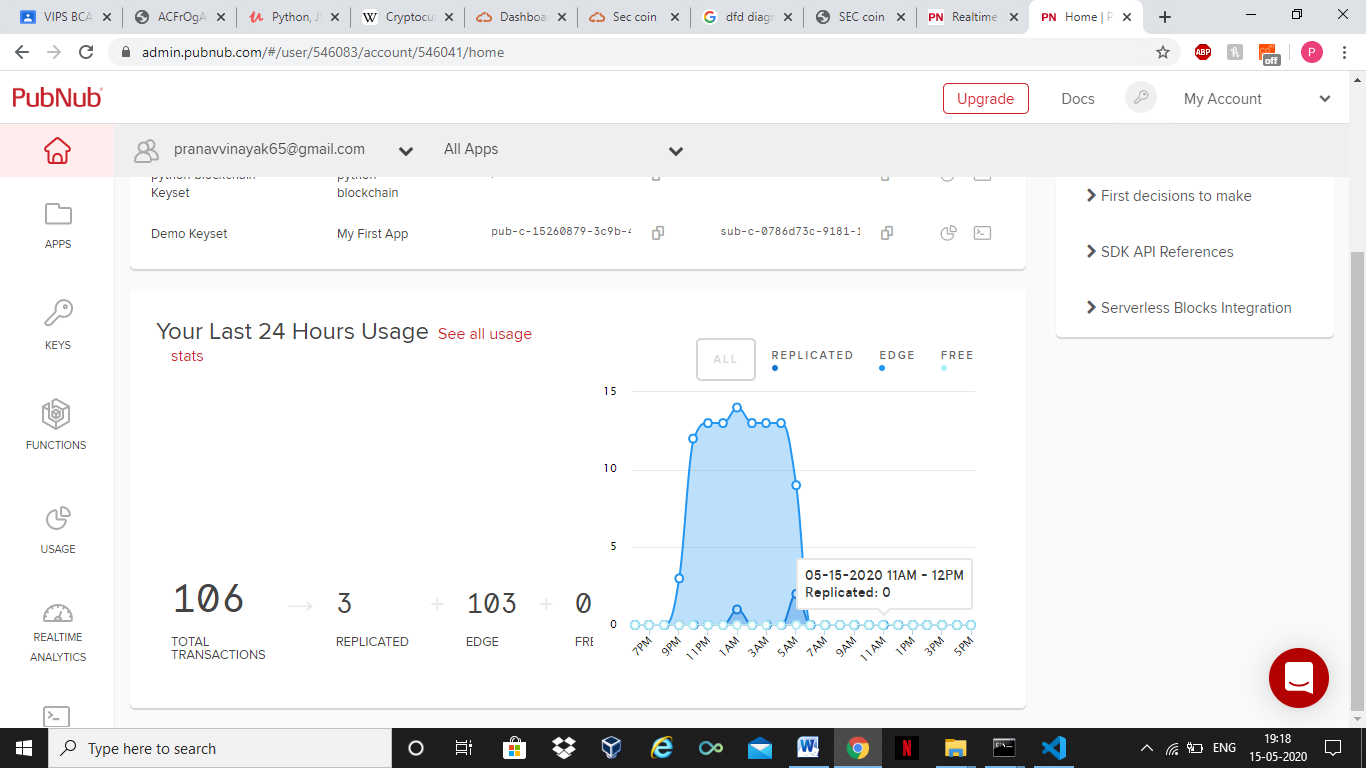
    pubsub = PubSub()

    time.sleep(1)

    pubsub.publish(CHANNELS['TEST'], { 'foo': 'bar' })

if \_\_name\_\_ == '\_\_main\_\_':

    main()



**Chapter 5**

**TESTING & TEST RESULTS**

**5.1 SOFTWARE TESTING**

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing is an exposure of a system to trial input to see whether software meets correct output. Testing cannot be determined whether software meets user’s needs, only whether it appears to confirm to requirements. Testing can show that a system is free of errors, only that it contains error. Testing finds errors, it does not correct errors. Software success is a quality product, on time and within cost. Through testing can reveal critical mistakes. Testing should therefore,

Validate Performance

Detects Errors

Identify Inconsistencies

**5.2 Test Objective**

* There is strong evidence that effective requirement management leads to overall project cost savings. The three primary reasons for this are,
* Requirement errors typically cost well over 10 times more to repair than other errors.
* Requirement errors typically comprise over 40% of all errors in a software project.
* Small reduction in the number of requirement errors pays big dividend in avoided rework costs and schedule delays.
* System are not designed as entire systems nor are they tested as single systems the analyst must perform both unit and system testing. For this different level testing are use:

**5.2.1 Unit Testing**

In unit testing Module is tested separately and the programmer simultaneously along with the coding of the module performs it.

In unit testing the analyst tests the programs making up a system. For this reason, unit testing is sometime called program testing. Unit testing gives stress on modules independently of one another, to find errors. This helps the tester in detecting errors in coding and logic that are contained within that module alone. The errors resulting from the interaction between modules are initially avoided.

Unit testing can be performed from the bottom up, Starting with smallest and lowest-level modules and proceeding one at a time., for each module in Bottom-up testing a short program is used to execute the module and provides the needed data, so that the module is asked to perform the way it will when embedded within the larger system.

**5.2.2 System Testing**

This is performed after the system is put together. The system is tested against the system requirement to check if all the requirements are met and if the system performs of specify by the requirements.

Testing is an important function to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully activated. Another reason for system testing is its utility as a user-oriented vehicle before implementation.

The function of testing is to detect the defects in the Software. The main goal testing is to uncover requirement, design and coding errors in the programs. The types of testing are discussed below:

**5.2.3 MODULE TESTING**

Module tests are typically dynamic white-box tests. This requires the execution of the software or parts of the software. The software can be executed in the target system, an emulator, simulator or any other suitable test environment.

The focus of the tests is:

* Set up of regression tests. This means the test environment once set up for a function can be re-used to check its performance e.g. after maintenance.
* Coverage of the relevant state of the art test methods like equivalence class building, boundary value analysis and condition coverage are used.

**5.2.3 INTEGRATION TESTING**

“If they all work individually, they should work when we put them together.” The problem of course is “putting them together “. This can be done in two ways:

1. Top down integration: Modules are integrated by moving downwards through the control hierarchy, beginning with main control module are incorporated into the structure in either a depth first or breadth first manner.
2. Bottom up integration: It begins with construction and testing with atomic modules i.e. modules at the lowest level of the program structure. Because modules are integrated from the bottom up, processing required for the modules subordinate to a given level is always available and the need of stubs is eliminated.

**5.2.4 BLACK-BOX TESTING**

Black-box testing is a method of [software testing](http://en.wikipedia.org/wiki/Software_testing) that tests the functionality of an application as opposed to its internal structures or workings.

The system is tested just to assure whether it is meeting all the expectations or requirements from it, tester is not concerned with the internal logic of the module or system to be tested. Some inputs are given to system and it is observed whether the system is working as per the client’s requirements or not or according to the requirements specified in SRS document. Specific knowledge of the application's code/internal structure and programming knowledge in general is not required.

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid inputs and determines the correct output. There is no knowledge of the test object's internal structure. This method of test can be applied to all levels of software testing: [unit](http://en.wikipedia.org/wiki/Unit_test), [integration](http://en.wikipedia.org/wiki/Integration_testing), [functional](http://en.wikipedia.org/wiki/Functional_testing), [system](http://en.wikipedia.org/wiki/System_testing) and [acceptance](http://en.wikipedia.org/wiki/Acceptance_test). It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well. Black box testing or functional testing is used to check that the outputs of a program, given certain inputs, conform to the functional specification of the program. The term black box indicates that the tester does not examine the internal implementation of the program being executed

**5.2.5 WHITE-BOX TESTING**

A software testing technique where by explicit knowledge of the internal workings of the item being tested are used to select the test data. Unlike black box testing, white box testing uses specific knowledge of programming code to examine outputs. The test is accurate only if the tester knows what the program is supposed to do. He or she can then see if the program diverges from its intended goal. White box testing does not account for errors caused by omission, and all visible code must also be readable.

Contrary to black-box testing, software is viewed as a white-box, or glass-box in white-box testing, as the structure and flow of the software under test are visible to the tester. Testing plans are made according to the details of the software implementation, such as programming language, logic, and styles. Test cases are derived from the program **structure**. White-box testing is also called glass-box testing, logic-driven testing or design-based testing. There are many techniques available in white-box testing, because the problem of intractability is eased by specific knowledge and attention on the structure of the software under test.

**Chapter 6**

**CONCLUSION**

**6.1CONCLUSION**

Cryptocurrency offers a new, effective and attractive model of payment methods that can boost companies and operators revenues. It also provide alternative method of payment, apart from real money, that enable users to make financial activities such as buying, selling, transferring and exchanging easily. Although cryptocurrency platforms open many channels for digital financial transactions and provide a new form of currency with different mechanisms and methods, they are not controlled and regulated as they deserved. The research analyzed cryptocurrency platforms and extracted many concerns and challenges that put such financial system under the risk. The lack of legislations is considered as the main concern in cryptocurrency systems. Almost a clear picture of the size of cryptocurrency use has been drawn from my analysis of the current cryptocurrency literature and from the conducted study. Although the pilot study has been conducted with relatively small sample, but the results showed me a preliminary perception about the use, the growth, the trust of using and future expectations of cryptocurrency. I can now realize many indications that can provide initial answers to the research questions. My analysis indicates that cryptocurrency is very likely to be the next currency platform due to the large volume of cryptocurrency that is flowing in different systems, the huge expanding and growing of using and implementing cryptocurrencies and the opportunities that cryptocurrency systems offer. Moreover, the confidence and trust rate of using cryptocurrency is noticeably high as it can be seen in several cases that have been stated in this paper besides the survey results. However, users have not realized the full picture of using cryptocurrency. In fact, many cryptocurrency forms do not deserve that much of trust yet. Many concerns, challenges and issues are existing in many cryptocurrency platforms and they are clearly outlined in the above sections of this paper. Until cryptocurrency is being well regulated and controlled, users need to take extra precautions of using such virtual money

**6.2 FUTURE SCOPE**

Some economic analysts predict a big change in crypto is forthcoming as institutional money enters the market.3﻿ Moreover, there is the possibility that crypto will be floated on the Nasdaq, which would further add credibility to blockchain and its uses as an alternative to conventional currencies.4﻿ Some predict that all that crypto needs is a verified exchange traded fund (ETF).5﻿ An ETF would definitely make it easier for people to invest in Bitcoin, but there still needs to be the demand to want to invest in crypto, which might not automatically be generated with a fund.

**BIBLOGRAPHY**

* http://www.w3schools.com
* http:// www .stackoverflow.com
* <http://wikipedia.com>
* http://udemy.com