Functional Requirements Document

IndieCoin

Version	**Description of Change**	**Author**	**Date**
1.0.0	Initial Requirements Detailing	Fernando Lobato Meeser	02/09/2017
1.0.1	Functional Requirements detailing	Fernando Lobato Meeser	02/11/2017

CONTENTS

1 INTRODUCTION

IndieCoin is a light weight implementation of a decentralized cryptocurrency. A cryptocurrency is a digital asset that can work as a medium of exchange using cryptographic principles to ensure security.

A decentralized cryptocurrency allows to parties to transact over an electronic medium without a trusted third party. In today's world if Alice want to send some form of value to Bob, she either must be physically in the same room with Bob to give him some form of cash or asset; or she must trust a third party to send money to Bob (Credit card, bank, western union, friend who will meet Bob). Not all people in the world have access to financial services. Decentralized cryptocurrencies allow any two parties to transact over an electronic medium without the need to trust a third party. Since there is no third party, there is a great deal of anonymity that does not exist in any other electronic medium. Currently carrying transactions over electronic mediums results in very high fees for the parties transacting because of the need to mediate disputes for the third party. This prevents a lot of markets from being accessible outside of their physical realm. Decentralized currencies present an economic scheme that brings these fees to a minimum.

While the field of digital currency and money is not new; the idea of having a decentralized system while keeping consensus could not be achieved until 2009 with Bitcoin. Bitcoin implements a concept called a blockchain. A blockchain is a distributed database where everybody keeps a copy of the records so anybody can verify the consistency of new transactions. In this case, it works as a p2p network. Every specific amount of time a new block is created (mined in cryptocurrency slang) by putting together several transactions. Then each node in the network must solve a proof of work, the first node to solve the proof of work broadcasts the block containing the newly validated transactions and the solution to the proof of work to all other nodes on the network. As nodes receive blocks they validate all transactions inside them to the previous copy of the block chain, incorporate the new block into their copy of the blockchain and begin looking a new block. Since the proof of work in the network is automatically adjusted to the computational power of the network and if someone in the network does not have the majority of the computing power, the network is safe from double spending. This is just an overview of how consensus is kept in a blockchain, in further documents there will be a detailed explanation of all aspects involved in the architecture of such a system. For simplification, we will call the process of finding and publishing a new block mining.

Another important aspect is the generation of coins. Each node is racing all other nodes for a period of time to have their block included into the blockchain. This happens because this is how the coin is generated. Each node includes a transaction at the beginning of every node sending a specific amount of non-existing coins to themselves. The miner who get his block published into the blockchain first get everybody to agree that he now owns a specific amount of coins that did not exist before. This adds an incentive to miners to keep the network honest. The more people that trust the network the more valuable their asset becomes.

The last important high level aspect is how is value transferred and accounted for in transactions. The blockchain is just a reference of attributing value to this abstract coins. When a coin is generated through the process described earlier called mining, this coin is tied up to the identity of the miner (person who owns the computer who found the new block). The network is anonymous and the way we bind identities is with the use of Asymmetric Cryptography. Every user has a public key and a private key. When they are going to receive money from someone else they send the other party their public key. That who will be sending payment takes some of his previous coins and creates a transaction. A transaction can be seen as an announcement where a party references previously owns coins and says that now the person who presents a digital signature to a specific public key will be able to spend those coins. At the same time, the person spending those coins must present a signature to the public key that was referenced in the previous transaction of those coins. This way the system is kept secure. Only the person who holds a private key referencing certain coins can spend them.

Cryptocurrencies have sparked a great deal of innovation in the Fintech community because of its technological breakthroughs. Since the release of Bitcoin there have been a great of different cryptocurrencies (alt coins). Most of this coins are simply a fork from the original bitcoin source code (which is open source). Being a relatively new field there is still a lot of innovation and research to be done. Nevertheless, being such a young field it lacks a great deal of experts and developers working in ways to bring better payment systems to our digital world. The technology briefly described earlier (blockchain) can be a starting point for creating a lot more things that just currencies. It is the first time in history we can have trust without the need of a central authority.

The actual bitcoin blockchain implementation is a very big and complex project for someone completely new into this are to understand. To understand it one must master c++, and need a lot of theory into GNU build system. While they are the current tools to implement the bitcoin blockchain, they are not needed in theory to understand and begin to build and test new blockchain applications. The purpose of this project is to create a scaled down version of a cryptocurrency, following the footprints of many existing currencies and implementing a simple blockchain in a p2p network in the python programming language, that allows any newcomer into the blockchain to easily understand the technical concepts and can build and modify simple prototypes.

As of now (2017) there exist a great deal of online resources of how blockchain works. Very few of this online resources cover any technical details that would empower developers to be able to understand the blockchain in a deeper sense.

1.1 Purpose

The purpose of this project is to create a solid starting point for any curious developer or engineer that wants to understand the technical aspects of blockchain implementation in cryptocurrencies. This will give them a bigger insight into blockchain itself and can complement the existing books and articles that already explain the theory without diving much into technical aspects.

The system should have the basic characteristics inside a cryptocurrency to make it a solid introduction and didactical technical tool. This features for a user running a node include:

- Nodes connect and talk to each other
- Nodes requesting and sending blocks
- Nodes validating and relaying transactions
- Nodes mining new blocks
- Nodes generating Public and Private Keys
- Nodes signing transactions and broadcasting them to other nodes
- Nodes listening to incoming transactions with the public key assigned to them to notify a user.

1.2 Scope

Cryptocurrencies and blockchains are a new field, therefore innovation is coming up from all places with very interesting proposals for new features. This new features will be out of scope for this project. This project’s purpose is to be a swift introduction into how a blockchain for a cryptocurrency work. Therefore, we are only concerned in having a blockchain where two parties can transact on a digital coin. No special type of transactions will be implemented and no simplified payment verification solution.

To limit the scope of the project and comply with deadlines, this version of the application will not implement a GUI to focus completely on the implementation of the blockchain. The program will be a CLI program that will be ran from terminal.

To keep the project simple and understandable there will be some drawbacks into the performance of the system. The system will be implemented in the python programming language which because of its interpreted nature is not very efficient but is very easy to write and understand.

1.3 Background

The idea for this document comes after I (Fernando Lobato) became curious into how the cryptocurrencies worked. I felt that a lot people were talking about it (especially technical people), but I realized very few knew how it worked (I personally did not). So, I began my investigation into the theory of how the worked. Understanding the basic ideas of how this system works proved to be a simple task, although when I wanted to dive deeper into certain technical aspects or had more questions about specific areas of how it worked I saw that since the field is relatively new, some things are still not documented. After going through more references (some awful, some that provided great insight) I decided to go into the actual implementation details of the system. This is where I was stuck for a long time because there are practically no artifacts detailing how the architecture works.

Since the purpose of software architecture is to create structures that help us reason about a system I realized that this is a system that needs structure to help more people reason about it easier. That way I can kill two stones with one bird. Learn more about solid software architecture principles and help document a system that I believe could help the world.

1.4 References

The following references are listed in order of how much knowledge someone has about the underlying system. The first to resources provide a very basic introduction into many of the concepts that will be used in all the documents. The other resources are lengthy, detailed and technical. However, they are useful as reference for specific subjects.

A swift introduction onto what bitcoin is, what it means and how it works. This videos show all the necessary prior knowledge needed to understand how a decentralized currency like the one that will be built works.
Concepts such as digital signatures, hashes, proof-of-work and certain cryptographic principles. While the explanations are not very technical they are great reference.
Khan Academy
https://www.khanacademy.org/economics-finance-domain/core-finance/money-and-banking/bitcoin/v/bitcoin-what-is-it
Created by: Zulfikar Ramzan

The initial bitcoin whitepaper which outlines the architecture for a distributed cryptocurrency.
Bitcoin: a peer to peer electronic cash system
https://bitcoin.org/bitcoin.pdf
By: Satoshi Nakamoto

Detailed explanation of all bitcoin features and their interactions.
Mastering Bitcoin
https://uplib.fr/w/images/8/83/Mastering_Bitcoin-Antonopoulos.pdf
By: Andreas M. Antonopoulos

The actual bitcoin open source repo implementation
https://github.com/bitcoin/bitcoin

A complete database schema for a bitcoin.
https://webbtc.com/api/schema

1.5 Assumptions and Constraints

1.5.1 Assumptions

This project is assuming that blockchain technologies will be a driven and innovative force in the technology field in the years to come. Any new tools could come tomorrow and offer a new alternative just as blockchain did. This project also assumes that the internet will keep its current domain name system structure and client server model, the application will be built on top of that model. There are however new designs for these systems using the blockchain (which are outside of the scope of this document).

1.5.2 Constraints

To comply with the purpose of the system, it should be kept to a minimum functionality while complying with the overall objective of a decentralized currency. This decision is to keep the learning curve as small as possible. The performance of the system will also be limited by the tools chosen to build the system. The process of mining in a blockchain is a very CPU or GPU intensive process that needs to be highly optimized. That level of specific optimization for a task is out of scope. We constraint the system to a minimum mining performance to keep our main objective of having a technical didactical solution.

3 FUNCTIONAL REQUIREMENTS

3.1 Context

IndieCoin is designed to be a peer-to-peer electronic digital currency. The program consists of a node (client-server) application that will be made in the python programming language to allow portability among operating systems.

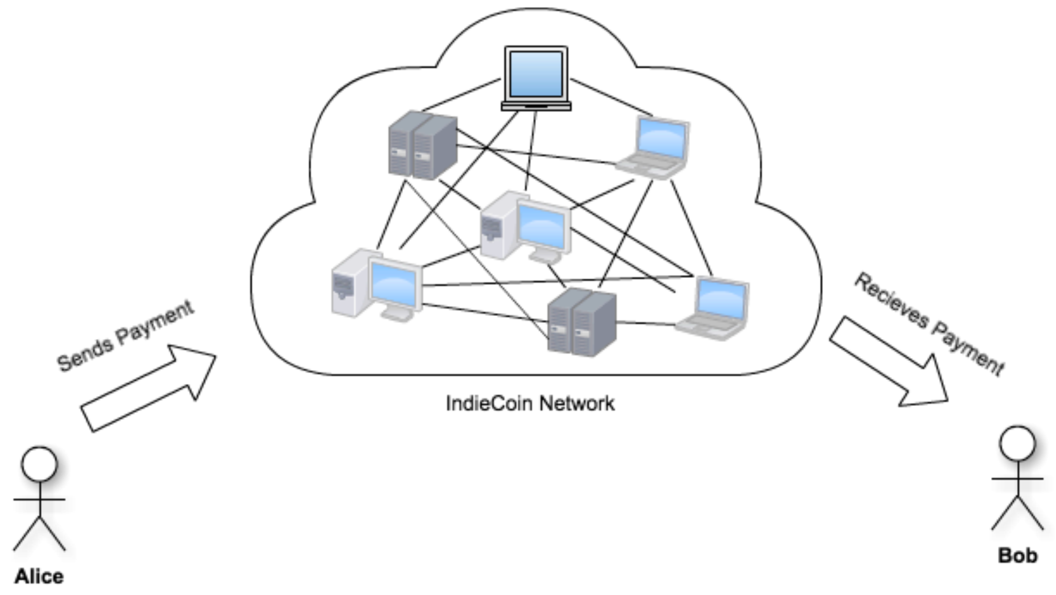

Diagram 1.1 Context of IndieCoin

The main context of the system is having any two parties transact over an electronic medium. In diagram 1.1, we have two fictitious parties called Alice and Bob. Alice want to send money to Bob and she chooses to use the IndieCoin network.

Each node inside the IndieCoin network is a computer (can be a server, desktop or laptop) that is running the IndieCoin application.

Each node is responsible for the verification and propagation of transactions through the network.

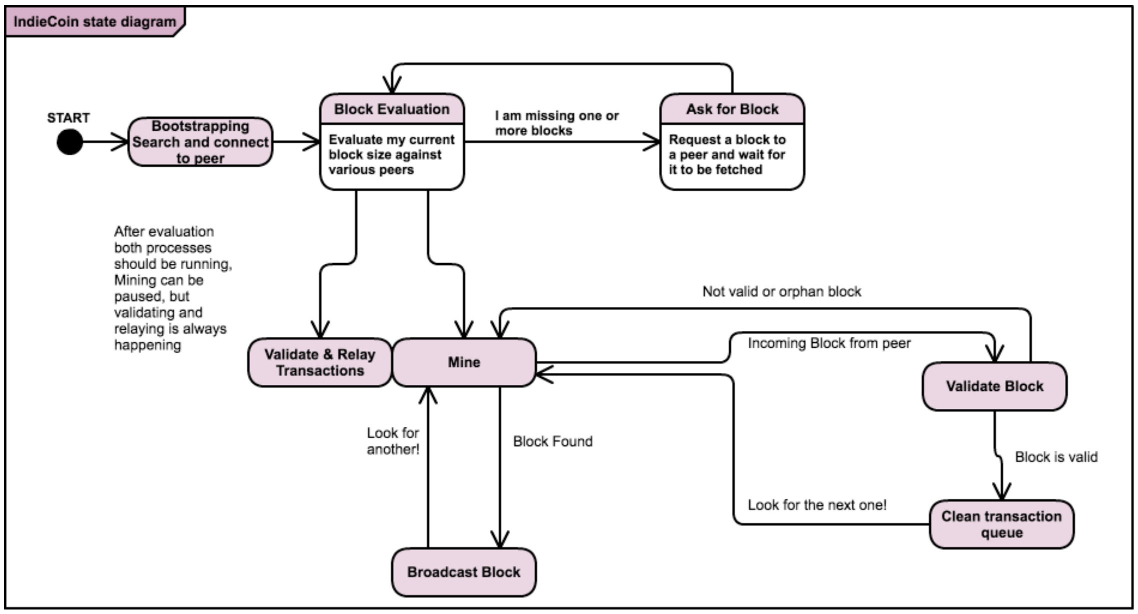
IndieCoin state diagram
START
Bootstrapping Search and connect to peer
Block Evaluation
Evaluate my current block size against various peers
I am missing one or more blocks
Ask for Block
Request a block to a peer and wait for it to be fetched
After evaluation both processes should be running, Mining can be paused, but validating and relaying is always happening
Not valid or orphan block
Incoming Block from peer
Validate & Relay Transactions
Mine
Validate Block
Look for another!
Block Found
Block is valid
Look for the next one!
Clean transaction queue
Broadcast Block

Diagram 1.2 IndieCoin Node state diagram

In diagram 1.2 we zoom into the main states and action that a node can have to complete our context objective. The diagram does not have an exit state because the system is meant to be kept running always. The system however can be stopped at any time from any state. A stop state and a transition from every state to this state was excluded to avoid overwhelming with information. This way nodes can join and leave the system at any time. When they come back the will just take the longest chain of proof-of-work.

Diagram 1.2 gives an overview of the process the system needs to follow to achieve its purpose. In the following sections, there will be a detail of how each of this states and transitions will be designed.

3.2 User Requirements

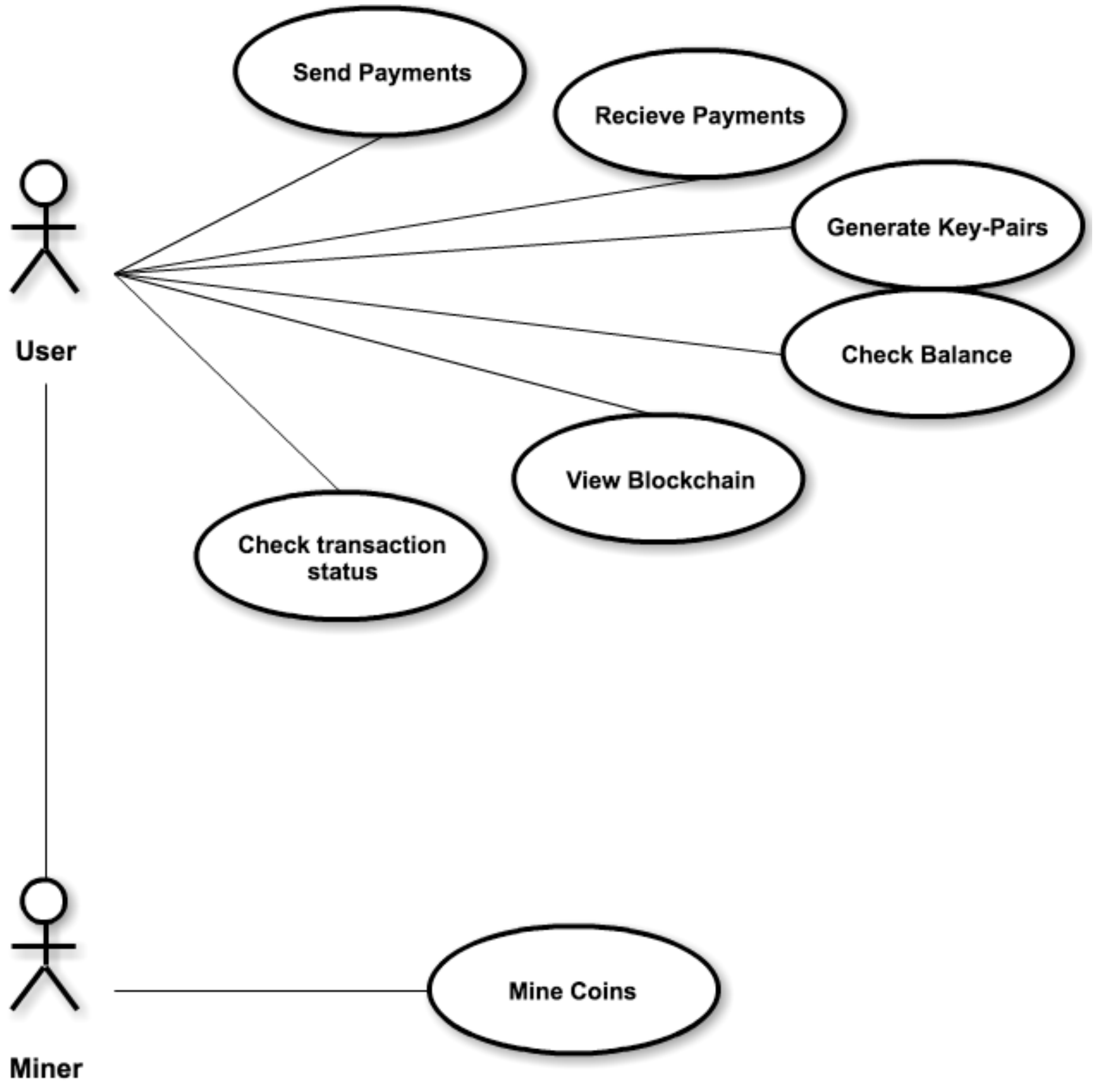

Diagram 2.1 System Requirement

This are the requirements need from a user perspective to fulfil the objective of having a distributed crypto-currency. Table 1.0 shows each system requirement in detail with a user story to describe it.

Table 1.0 Detailed System Requirements

	User	System Requirements	User Story
1	Normal User	Send Payments	As an IndieCoin user I want to send a payment to another user in the IndieCoin Network.
2		Receive Payments	Aa an IndieCoin user I want to receive a payment to from another user in the IndieCoin Network.
3		Generate Key-Pairs	As an Indie Coin user, I want to generate a new private and public key where so that I can receive payments.
4		View Blockchain	As an Indie Coin user, I want to be able to check any information in the blockchain so I can make sure IndieCoin works correctly or clear any doubt regarding transaction.
5		Check Transaction Status	As an Indie Coin user, I want to be able to check the status of a transaction so I know where my payment is.
6		Check Balance	As an IndieCoin user I want to be able to check the balance related to my public key to verify my available funds.
7	Miner	Mine	Aa an IndieCoin miner I want to be able to mine to generate new IndieCoins to have more money.

4.1 Logical Data Model/Data Dictionary

There are 5 main entities needed to run the system. This persistent data model allows us to keep track of all the transactions in the system.

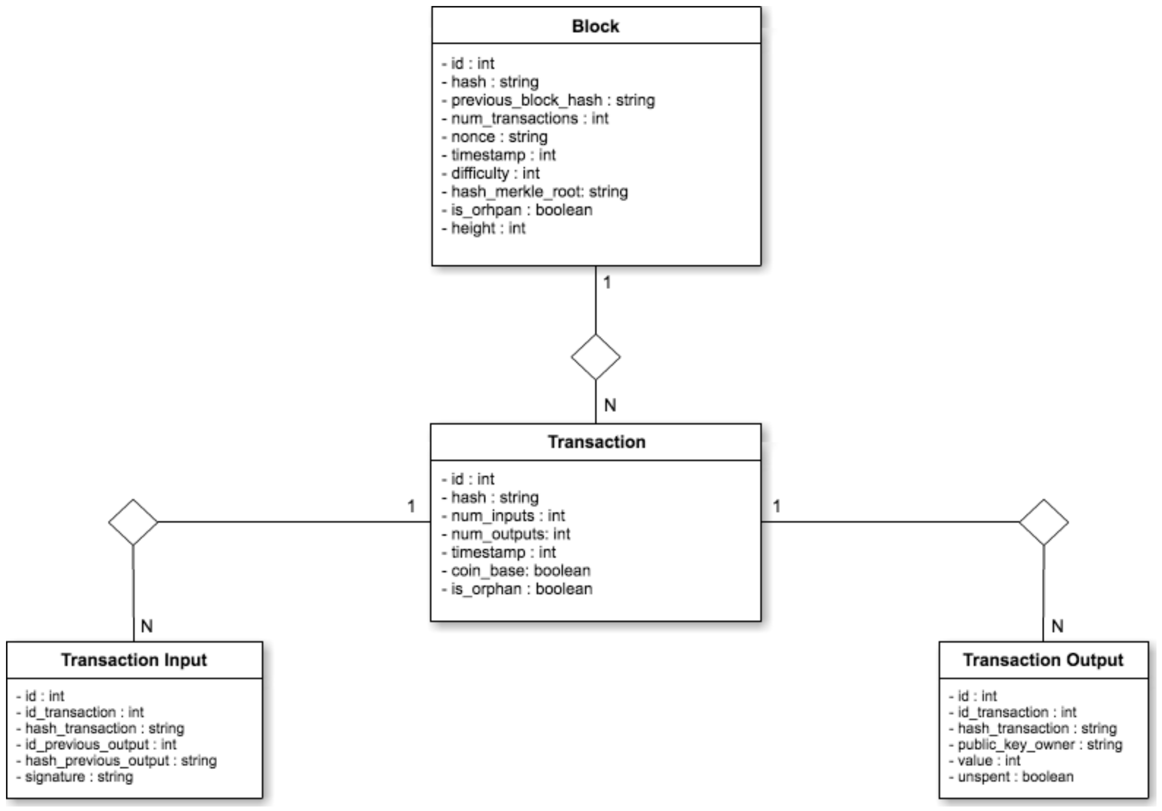

Diagram 3 Entity Relationship Diagram

The basic structure required for our blockchain is to keep a database of block where each block references a previous block. Each block consists of transaction and each transaction references previous transactions in order to spend money.

4.2 Functional Requirements

No.	Functional Requirement	Description

1	Create Transaction	Reference previous inputs, add digital signature, create outputs and propagate through network
2	Validate Transaction	Validate that for each input in the transaction the digital signature matches, get the sum of inputs value and verify that it's less than the sum of outputs value.
3	Relay Transaction	Spread a transaction through the network.
4	Sign Transaction	Create a digital signature of a transaction.
5	List Peers	Display a list of known peers.
6	Connect to Peers	Request a connection to a peer.
7	Request Peers	Request a list of peers to a know peers.
8	Receive Peers	Return a list of peers to a know peer.
9	Request Block	Request a block of specific height to a peer.
10	Receive Block	Return a requested block of specific height to a peer.
11	Fill Block	Take transactions from transaction queue and put them into a block.
12	Mine	Hash the block with all the transaction changing the nonce until the hash is less than the difficulty (proof-of-work)
13	Broadcast Block	Send a newly found block to all peers in network.

14	Validate Block	Validate all transaction inside a block and that it references an existing block.
15	Clean transaction Queue	Once a block is accepted, remove all transactions that are in that block that are also in my transaction queue.
16	Generate Key - Pairs	Generate a public and private key using elliptic curve cryptography.
17	Adjust difficulty	Adjust the difficulty for the proof of work based on the amount to time that some period of blocks were mined
18	Request Height	Ask to certain random peers the current height of their blockchain.

Table 1.2 Functional Requirements.

4.2.1 Functional Requirements Group 1

Communicating - Requirements Group 1

Section/ Requirement ID	**Requirement Definition**
FR 5	The system shall list peers.
FR 6	The system shall connect to peers.
FR 7	The system shall request peers.
FR 8	The system shall receive peers.
FR 9	The system shall request blocks.
FR 10	The system shall receive blocks.
FR 18	The system shall request height.
FR 3	The system shall relay transactions.
FR 13	The system shall broadcast blocks

4.2.2 Functional Requirements Group 2

Wallet Operations - Requirements Group 2

Section/ Requirement ID	**Requirement Definition**
FR 2	The system shall validate transactions.
FR 4	The system shall sign transactions.
FR 16	The system shall generate key-pairs.

4.2.3 Functional Requirements Group 3

Verification - Requirements Group 3

Section/ Requirement ID	**Requirement Definition**
FR 14	The system shall validate blocks.
FR 15	The system shall clean transaction queue.
FR 17	The system shall adjust difficulty.

4.3 Interface Requirements

To limit the scope of the system and focus on the actual implementation of the blockchain no GUI will be generated for this version. The program will be ran from the command line. Documentation will be provided to calibrate certain parameters from the command line.

4.3.1 Hardware Interfaces

The system runs purely as a software interphase.

4.3.2 Software Interfaces

The application will not directly interact with any other application. It will however run on top of the python interpreter and will make use of the following libraries:

- BTPeer

  Description: A Pure-Python implementation of a peer to peer framework for general purpose.

  Author: Nadeem Abdul Hamid

  Original Source can be found at:

  http://cs.berry.edu/~nhamid/p2p/btpeer.py

- pyaes

  Description: Pure-Python implementation of AES block-cipher and common modes of operation.

  Author: Richard Moore (pyaes@ricmoo.com)

  License: MIT

- python-ecdsa

  Description: This is an easy-to-use implementation of ECDSA cryptography (Elliptic Curve Digital Signature Algorithm), implemented purely in Python, released under the MIT license. With this library, you can quickly create key pairs (signing key and verifying key), sign messages, and verify the signatures. The keys and signatures are very short, making them easy to handle and incorporate into other protocols.

  Author: Brian Warner (warner-github@lothar.com)

  License: MIT

  Original source can be found at:

https://github.com/warner/python-ecdsa

4.3.3 Communications Interfaces

Nodes inside the IndieCoin network need to communicate with each other to relay transactions, peers and blocks. Each node should be running a listening thread that can handle different types of request. When an incoming connection comes, a new thread should be spanned with the handler for that specific action.

The application will communicate over the internet protocol using the transport control protocol (TCP/IP).

IndieCoin can be seen as a communication protocol for keeping track of value among peers.

The following communication actions will be performed by the system.

1. Request/Return Peers

   Nodes should be able to ask other nodes for the list of their known peers so that the network can grow.

2. Request/Return Blocks

   Nodes may join and leave the network at any time. When a node comes into the existing network it can request any block that was mined during its absence. If a node is new to the network, it can one by one for all the blocks in the network.

3. Broadcast Block

   When a node mines a new block, it broadcasts it to all its peers.

4. Relay Block

   When a node receives and validates a block it will send it to other peers.

5. Relay/Receive Transaction

When a node receives and validates a transaction it will send it to other peers.

6. Request/Return Height

A node may ask other nodes their current block height to know if it is up to date or should request some blocks before going operational.

4.4 Data Conversion Requirements

The system transmits will all data through IP/TCP protocol using JSON format. When the data arrives into a node it will be de serialized and verified for consistency using JSON format. If the information is not in the correct format it will be ignored. It will also be ignored if it is in the correct format but is not consistent with the protocol.

If the information transmitted is a block or a transaction it need to be stored persistently in the system. Before data is stored persistently we will have an interphase class to deal with it and verify the business rules that apply to each form of data. After that the data will be written into the database where it will be kept persistent.

4.5 Hardware/Software Requirements

The system will be implemented on the python programming language version 2.7. It will be recommended and documented how to run a virtual environment so that all project dependencies do not conflict with existing dependencies inside a user's system. All the dependencies will be managed using python requirements scheme.

The python interpreter for python 2.7 comes included in Mac OS X and Linux operating system. It can be installed for windows and other 22 platforms.

The complete list of available platforms and installation instructions can be found:

https://www.python.org/download/other/

4.6 Operational Requirements

The system is meant to be a decentralized system. Once the system is running it should not receive intervention from an organizational perspective, no organization should control it.

4.6.1 Security and Privacy

IndieCoin is an electronic payment system. This puts a lot of pressure in keeping the system save. The first part of security inside system comes from digital signatures to keep authentication of users. As long as private keys are generated in a random manner and users store them safely, their funds should be kept save from malicious use.

The second big problem when dealing with security in a payment network is double sepnding. Double spending is when a user tries to spend a digital token twice to confuse the system. This problem is mitigated in the blockchain with the hashbased proof-of-work. As long as the amjority of CPU power inside IndieCoin is acting honest then nobody can double spend any coin. Because all other nodes will simply ignore any node that tries to insert fake transactions.

To reduce latency and throughput it's not necessary for the system to send information encrypted. All information can travel completely open, since the security of the system lies in keeping private keys secure. All the information in the blockhchain is available for anyone to look.

4.6.2 Recoverability

A. In the event the application is unavailable to users (down) because of a system failure, how soon after the failure is detected must function be restored?

Peer to peer networks have proved to be very resistant to complete system failures, because for the system to completely fail it need all connections among all nodes to be down. Or all nodes inside the system must suffer from some catastrophic failure. The probability of this happening decreases with the number of peers in the system. At the same time if there are not any peers in the system then it means that it is not very popular and no user would be there to detect a lack of service. If there are few users in the system then as long as they are online, the system will be online. As popularity increases, availability increases. The system should not need to recover at any time. A specific node can retreat from the system at any time and when it gets back, it only has to trust the longest block of work in the network to get back in sync.

B. In the event the database is corrupted, to what level of currency must it be restored? For example "The database must be capable of being restored to its condition of no more than 1 hour before the corruption occurred".

   Since the system is dealing with a distributed system with high redundancy, in the case of a node database failure, the systems information would not be compromised and the information of that node can be restored without any loss of information. The database must be capable of being restored to its exact condition before being corrupted.

C. If the processing site (hardware, data, and onsite backup) is destroyed, how soon must the application be able to be restored?]

   Being a peer to peer network in which every single node keeps a complete copy of the records, then it would require every single node to be destroyed. An individual node can be destroyed, but thanks to the systems high redundancy it would go completely unnoticed.

Recoverability is the ability to restore function and data in the event of a failure.

4.6.3 System Availability

IndieCoin should have a 99.999% availability for all users. The way that this specific system will achieve this is through a massive redundancy. The system is basically a P2P network where every node has a complete copy of the blockchain, therefore if one node stay alive, the system can still be running. Obviously, it would require more than one node to serve a lot of people. That is the beauty of P2P networks, the more people that use the system, the more available the system is.

4.6.4 General Performance

[Specific performance requirements, related to a specific functional requirement, should be listed with that functional requirement.]

In real life applications, there should be a great deal of optimization in terms of throughput, latency and database writing times. Nevertheless, the nature of this project is didactical. The python programming language is a relatively slow language for this task, but it suits well for understanding easily what the code does.

4.6.5 Capacity

As a blockchain becomes mainstream, the data starts to grow exponentially because every node must keep a complete history of the records in order to maintain decentralization. This project is not intended to scale at the transaction level of existing cryptocurrencies. There are however various techniques on how to optimize the blockchain to keep it small. All of those are out of the scope of this project.

In terms of memory, the blockchain proves to be a very inefficient system because every single node must maintain a complete copy of all the records. This is however the price of not trusting anybody, the price of decentralization.

4.6.6 Data Retention

Once a block or a transaction are validated they become part of the blockchain history. At this point no information will be deleted from the system at any time. The blockchain should allow any user to navigate through all the transactions that have occurred in the history of transactions all the way to the first transaction which must be hard coded into the system.

4.6.7 Error Handling

Whenever a message or a request is sent that is not in the correct format or is not part of the protocol, each node will automatically answer with a standard error message.