ITESM CAMPUS QUERÉTARO

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DEPARTAMENT OF COMPUTER SCIENCE

Software Architecture and Design
Course Project – HLD

IndieCoin

High Level Design

Draft

Area:	<Engineering>	**Document Number:**	1
Date:	<February 11 2017>	**Document Version:**	Draft
Author:	Fernando Lobato Meeser		
Short Description:	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.		

Revision History

Version	Date	Author	Section	Description
1.0.0	02/11/2017	Fernando Lobato Meeser	All	Begin Doccument

Sign-off

Product Marketing	Date
Developer	Date
Director IS Systems Development	Date
Servers and Networks	Date
Project Manager	Date
Sr. Marketing Manager	Date
Telecom	Date
Servers and Networks	Date

Distribution List

1. <Lead Architect> Fernando Lobato Meeser
2. <Dev Manager>

TABLE OF CONTENTS

1 Purpose of this document

This document is a technical description of the requested project. This document is where a technical analyst or a developer will describe proposed development of the change or project.

2 Intended audience

This document is intended for the development and QA teams that will work on this project.

3 Functional Description

3.1 Functional Overview

IndieCoin is a lightweight implementation of a decentralized cryptocurrency mostly based upon the bitcoin protocol and the Bitcoin cryptocurrency. IndieCoin is designed to be a MVP of a blockchain based cryptocurrency. For more reference about cryptocurrencies there is a great deal in the Functional Requirements Detail.

At a glance, the system allows any two parties two transact in an efficient and anonymous way over an electronic medium without the need of a trusted third party.

IndieCoin is designed to function as a peer to peer network where any user who wants to run the application to send, receive or verify payments can join or leave the network at any time.

On keys and identities

Users are not represented with their real-world identities in the IndieCoin network. When a user downloads the application he or she can generate an asymmetric key pair. For more reference on asymmetric cryptography visit Wikipedia article or reference in Functional Requirements Document. The public key will function as the identity of the user which he can publish to anyone on the internet to receive payments. The private key is his security token with which he can later spend payments referenced to him. To keep the system, secure the generation of the private key must be random.

On the blockchain

The blockchain is a distributed public ledger that keeps track of transactions. It consists of blocks of transactions. Each node has a complete copy of the blockchain and all nodes are receiving transactions and trying to add them into blocks. The block will be kept at a specific size of 3 MB to keep the network from filling up. Inside the block there will be a timestamp, a reference to the last block that the specific node has reference and all the transactions that compose it. Once a block is published and accepted by all nodes, the transactions in that block are said to be irreversible, if all nodes agree on this. To make the system efficient a new block will be created every 5 minutes. For a node to take 5 minutes to calculate a block there is a proof-of-work that the node must complete. This proof-of-work has a value called *difficulty* which will be adjusted every 4032 blocks (two weeks). This is meant so that if the computational power in the network increases then the difficulty will increase making the proof-of-work harder and keeping the 5 minutes' threshold in block generation. At the same time if the computational power in the network decreases then the difficulty will decrease. The proof-of-work is presenting a cryptographic hash of the block (by changing a value called nonce) whose decimal representation is less than the targeted difficulty. This means proves to be computationally difficult because the characteristic of cryptographic hash (in this

sha256). For more information on hashes and cryptographic hashes refer to documentation on functional requirement document. The only way to achieve this currently is through brute force. Therefore, as the difficulty increases the probability of finding it is spread through the network.

The process of finding the hash of the block is called mining. Mining is also meant to be the issuing of coins and an incentive to keep the network and honest. While assembling a block, each node will add as first transaction something called a coinbase transaction. A coinbase transaction is how money is introduced into the network. This transaction will give the owner of that node that specific amount of coins which h can spend later.

On transactions

A transaction is a transfer of IndieCoins among users. On can only have IndieCoins if somebody in the past gave them IndieCoins or through a coinbase transaction. A transaction takes previous transactions as inputs. Each previous transaction has an amount, a reference to previous transaction (except for coinbase transaction) and public key of the user who can spend that specific transaction. A user who wants to spend previously earned transactions references them, presents a digital signature with his private key to each of the previous transactions and create a list of output transactions which will be references in the future for spending. In this list, each output transaction contains an amount and the public key of who will be able to spend that amount. The sum of the outputs must be less than the sum of the inputs and any value that is left over will be given as a fee to the miner who creates the block that records that transaction. If a user wants to return change to himself then he must specify it the output and when he wants to spend that change he will make a reference to that output.

On the network

IndieCoin is a p2p network where every single node has a complete copy of all the history of transactions. Nodes can ask other nodes for a list of peers and connect to those peers. Nodes can query other nodes on the current height of their blockchain (how many blocks they have) if the height is smaller than the one they currently have they can query specific blocks. Nodes can verify and relay new blocks and transactions. To make this work the application will be composed of a main thread that will be listening to incoming requests on a socket. A handler for each type of request will be mapped in the system. When a request comes, the system will create a new thread with the handler for that specific request.

On network race conditions

Since we are dealing with a decentralized network there are many things that can occur. A transaction that references inputs transactions that a node has not yet verified or received can appear. Each node has a queue of transactions that arrive and get verified. If the transactions references inputs that are not

yet in the blockchain then it will be logically moved to another queue that will have what we will call orphan transactions. When a new block arrives, the orphan queue will be checked to see if any transaction is referencing a transaction that just occurred in the previous block. The same can happen with blocks.

There is also the case that two nodes find one block at almost the same time and while it propagates, the network forks into two different versions of the blockchain. If this happens, each node will take the first block that arrives as valid. But will keep the secondary node that arrives in their orphan block queue. That way if the next block that arrives references an orphan block instead of the previous block, each node will take the longest proof-of-work so they will modify their blockchain to reference the orphan block. This momentary split in the network is called a fork and will eventually be solved.

3.2 Logical Flow and Business Rules

The main part of the application will be the ICPeer class. This class will be the implementation of the p2p node that handles requests, connections and sends data among peers. This class will inherit from the existing BTpeer class which serves as a framework class for our ICPeer. The BTpeer class is where the mainloop that handles request resides. The mainloop handles incoming requests to a specific port with a socket. When a request arrives, it spans a new thread and depending on the message type of the request.

Each request type will be matched to a specific function, the binding of this functions in the array handlers that lies in the ICPeer. The ICPeer will have the definition for the functions for the peers to connect. Each connection is treated as a BTConnection which is simply a wrapper around a socket to perform simple read and write operations.

There will be a database class which will interact directly with the DBMS. The objects retrieved from the database will go through class interphases to use directly in the application (Transaction, Block). When an incoming Transaction or Block arrives, they will be casted to these interphases before the application deals with them. The ICPeer will also have a miner instance which will run on a separate thread a can be interrupted in the case that a block arrives into the node.

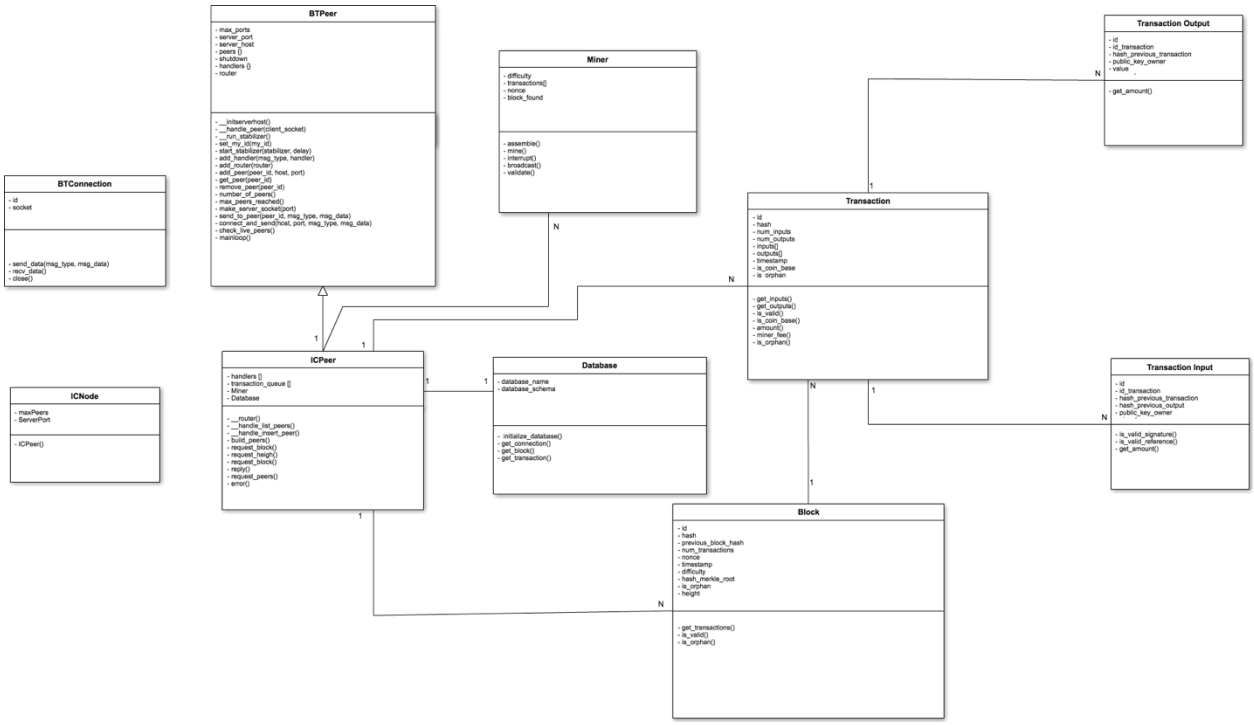


Diagram 3.0 IndieCoin Class Diagram.

3.3 Architectural Overview

The system is based on a p2p architecture where we have multiple peers running the same software. In diagram 2.5 we have a set of nodes and a zoom into one of this node to illustrate how each of them work. Each node will have an instance of an sqllite database where the blocks will be persistently stored. Each node will have a specific port over which it will listen to incoming requests and responses. Inside the node, most information will be handled on the database. But all transactions and blocks that are not yet confirmed in blocks will be queued in runtime memory. Thanks to the p2p architecture even if a peer loses information, it can request it from other peers. Each time that a request appears on the listening port, it will be handled by the interphase designed for that message. Each handler request starts a new thread with a socket for serving that request. When the request is served, the socket is closed.

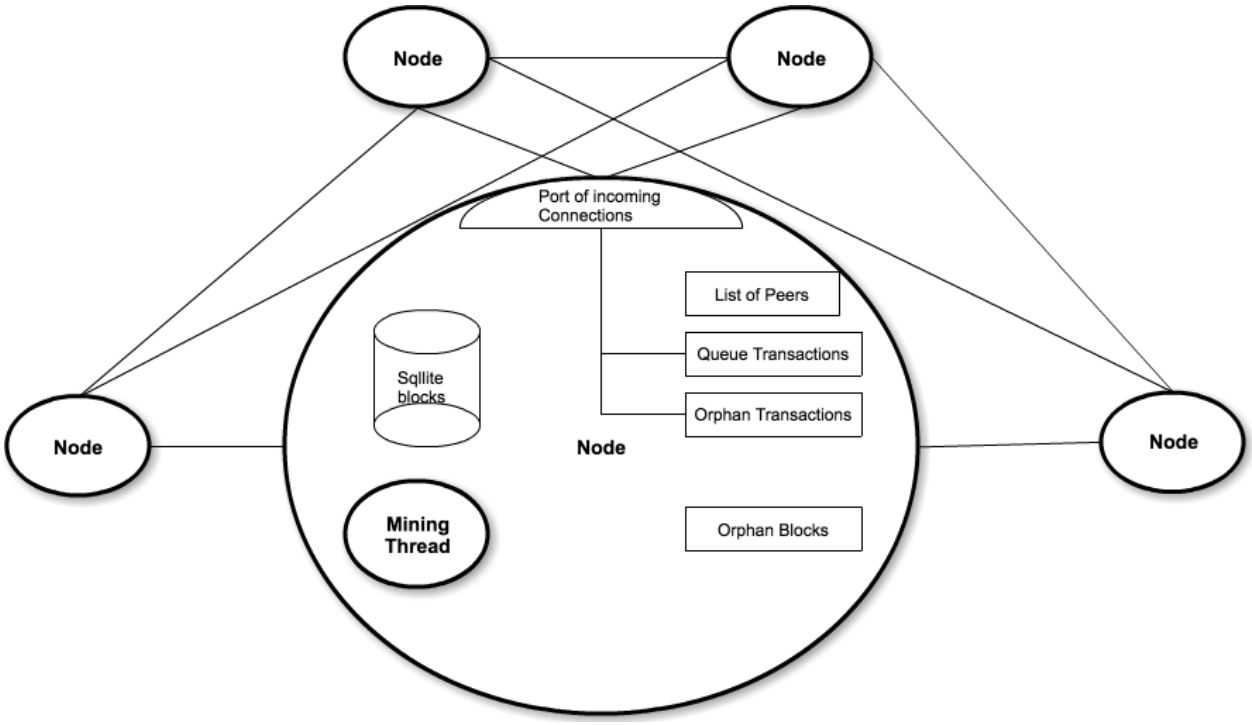
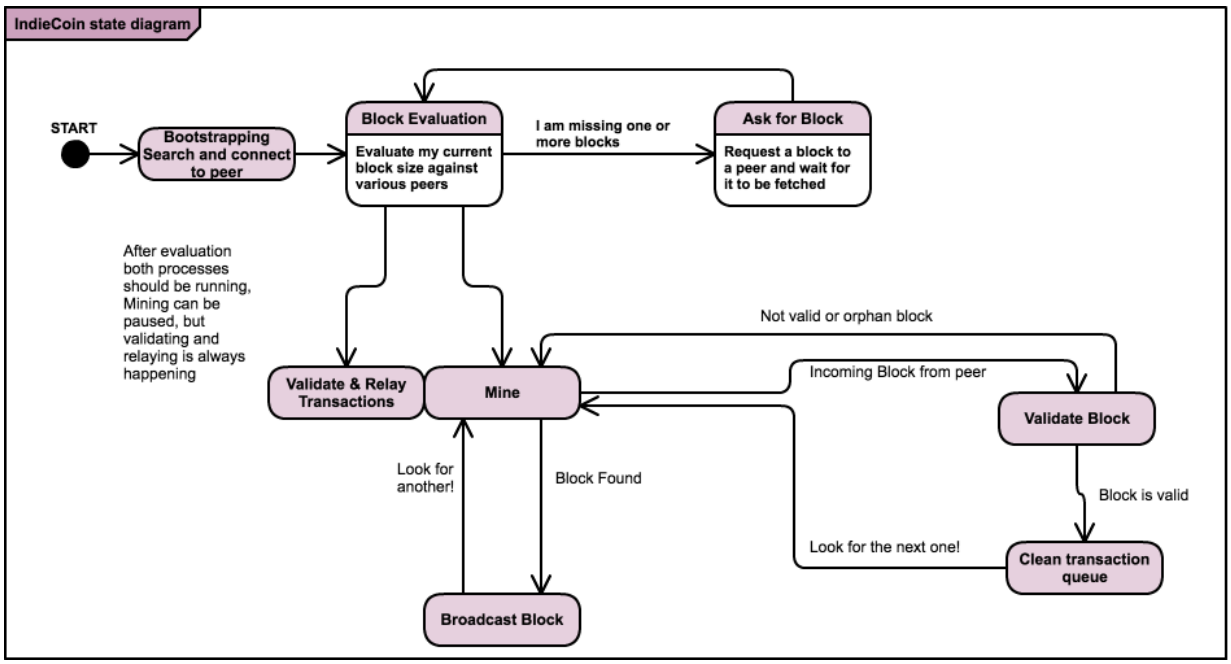


Diagram 2.5 Initial System Architecture

The following diagram shows the states through which a node goes through during its life cycle. A more detailed description can be found in the functional requirements document.



3.4 Application component additions or modifications

To have persistent data the application will use the Sqllite database management system. Sqllite is a lightweight DBMS that provides a lot flexibility and performance.

"*SQLite does not need to be "installed" before it is used. There is no "setup" procedure. There is no server process that needs to be started, stopped, or configured. There is no need for an administrator to create a new database instance or assign access permissions to users. SQLite uses no configuration files. Nothing needs to be done to tell the system that SQLite is running. No actions are required to recover after a system crash or power failure. There is nothing to troubleshoot.*[1]*"*

Additional classes will be created inside the application to serve as interphases with the objects that will be stored in the application so they can be treated at runtime.

The application will use the following python libraries directly from their source code:

(These components are for very generic use and do not need the slightest modifications)

- 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

The following software libraries will be useful as utilities for cryptographic keys, they will be added as utility classes to help the other classes.

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

  Author: Richard Moore (pyaes@ricmoo.com)

  License: MIT

- python-ecdsa

[1] "Distinctive Features Of Sqlite". *Sqlite.org*. N.p., 2017. Web. 13 Feb. 2017. https://www.sqlite.org/different.html

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

3.4.1 Business Objects

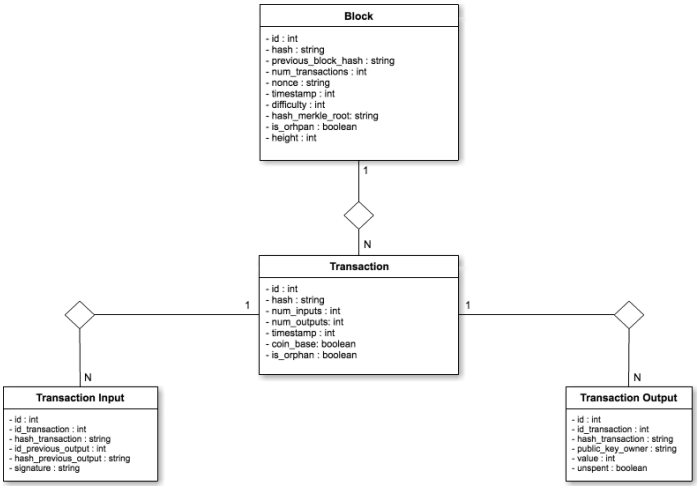
All the components listed in the previous diagrams must be created. The components that will be added that will not be created will not be modified and they can be found on a later section.

3.4.2 Pages or Forms

To limit the scope of development and focus completely on the blockchain technology, no GUI will be implemented in for this version of the application. The interaction will be handled through the command line interphase.

3.4.3 Database Modifications

The python programming language has an interphase to connect directly to sqllite which is a very lightweight DBMS. It needs no configuration and can be directly downloaded from python install package. The first time the application runs it will try to look for the existing database inside its folder, if it does not find it will create the database schema provided below.



3.4.4 Interaction with Other Modules or APIs

At this moment, the system will now interact with any other 3rd party modules or APIs.

3.5 Application Configuration Dependencies

3.5.1 Runtime Parameters

When a user runs the application, there are certain runtime parameters that will inhibit the systems functions. All parameters are optional. They will be stored inside a parameter table inside the ICNode class where the mainloop from BTPeer class will be called. The parameters will be passed upon startup to the application as CLI parameters.

Parameters:

- max_peers: define the maximum number of active peers the application should have stored in its table for connection during runtime.
    - Default value: 100
    - Max value: 300
- server_port: define a specific port over which the application should be listening for requests.
    - Default value: 0
- DNS_SEED: a list of tuples consisting of ip addresses and ports that the application should add as peers. There will be some DNS seeds hardcoded into the application (personal server).
    - Default value: [(104.131.120.174,8888)]
- miner: Boolean value determining if there should be a separate process mining.
    - Default value: False

- public_key: if the miner flag is set to true a private key can be provided for coinbase transactions. If it's set to true and no private key is provided, a key pair will be generated and the private key will be outputted.

3.5.2 Other Requirements

The application might conflict with the user's computer if there is a firewall blocking incoming and outgoing untrusted connections. If this is the case, the user must mark the application as secure or disable the firewall to allow the application to run.

3.6 Error Conditions

3.6.1 Application Exceptions

At any point during a node can could receive a message that is not mapped to any specific handler or for some reason the information could come in an inconsistent format. If this is the case, the request will be discarded and an error response will be delivered to the host that sent that message. This serves so that if the client is waiting for a response this will allow him to stop waiting.

If at any point a node loses connection to other peers which would mean the system lost connection to the network, then the system will lose all connection to the IndieCoin Network. The node must handle these exceptions and wait for internet connection to be reestablished.

3.6.2 System Exceptions

3.7 System Dependencies/Interaction

3.8 Other Technical Issues

A very big technical issue will be updating the system. Since not all nodes would update the system and we want to the network to run as complete as possible, software updates must be mostly managed through what is called soft forks. A soft fork is an update of the system with backward compatibility. It has happened in some cryptocurrencies such as Ethereum that a hard fork happens and the network splits into two different blockchain histories creating two different cryptocurrencies.

3.9 Hardware/Software Environment Overview

The system will run in the python interpreter which can be found or easily installed in more than 22 systems. For further information refer to the functional requirement document.

4 Configuration/Installation Steps

The following steps assume that a copy of the software has been downloaded form:

https://github.com/fernandolobato/IndieCoin

Verify that you have a version of python 2.7 running in the system for deployment. To verify you can open a CLI interphase and type python –V. If python is installed it will return the current installed version, if not it will return an error.

To download python, go to the follow link and follow the directions.

https://www.python.org/downloads/

Install pip.

Pip is the preferred package installer for python, if you are using any distribution of Linux or Mac OS X it is already installed, it can be upgraded with:

```
pip install -U pip setuptools
```

If the installation is going in a windows platform and pip is not installed yet run the following command:

```
python -m pip install -U pip setuptools
```

After pip is installed there are two way to go. The virtual environment will create a separate python installation for this project so that python dependencies for this project will not go into your global python and affect other installed packages or future installations.

Setup with virtual environment (Recommended):

Using the command line interface go to the root of the project and type into the cli

```
pip install –r requirements.txt
```

This will install all required libraries that are not part of the standard python library.

To run the program:

```
python indie-coin-node.py [parameters]
```

To get a list of parameters and their value:

```
python indie-coin-node.py --help
```

If the system is run without any parameters, default values will follow and the program should begin to download a copy of the blockchain from other nodes.

5 Unit Test Plan

Each functional requirement will be tested with a script that simulates being a node and queries specific information to the node that should be correctly running to verify the consistency of that information. With this we can verify that there is a correct functionality of the system. Each of this scripts shall request a specific handler function inside the node and should receive consistent information.

6 Project Timeline

Category	Detailed Description	Estimated Time
Research	Research information relate to implementation of a p2p network as well as the fundamental architecture of a cryptocurrency	2 weeks
Documentation	Document the architecture of the system and provide enough diagrams and models to understand it.	1 weeks
Code	Implement all the functionality provided in this documents.	5 weeks
Unit Testing	Test all the system components.	2 weeks
Deploy	Deploying system to server and computer to begin	1 weeks
	TOTAL estimate	11 weeks

A detailed plan can be found in Appendix 1 with a Gantt Diagram.

References

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

Style guide that will be used to code

https://www.python.org/dev/peps/pep-0008/

Appendix 1. Gantt Diagram

GANTT DIAGRAM	January	January	January	February	February	February	February	March	March	March	March
Activity	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Inception & Research											
Research											
Documentation											
P2P Implementation											
Transaction Implementation (database & interphase)											
Block Implementation (database & interphase))											
Key-pairs implementation											
Miner protocol implementation											
Unit Testing											

Deploy											