1. Document Conventions

1.1 Intended audience

This documentation is mostly intended for coders. If you can use a programming language, this will help you how cryptographic currencies work, how to use them, and how to develop software that works with them. The first few topics are also suitable as an in-depth introduction to Indicoins(A type of cryptocurrency) for noncoders—those trying to understand the inner workings of Indicoin and cryptocurrencies.

1.2 A walkthrough of Blockchain

A blockchain is a digital concept to store **data**. This data comes in blocks, so imagine blocks of digital data. These blocks are chained together, and this makes their data immutable. When a block of data is chained to the other blocks, its data can never be changed again. It will be publicly available to anyone who wants to see it ever again, in exactly the way it was once added to the blockchain. That is quite revolutionary, because it allows us to keep track records of pretty much anything we can think of (to name some: property rights, identities, money balances, medical records), without being at risk of someone tampering with those records. If one buys a house right now and add a photo of the property rights to a blockchain, he/she will always and forever be able to prove that he/she owned those rights at that point. Nobody can change that piece of information once it is on a blockchain. So—Blockchain is a way to save data and make it immutable.

This documentation covers the creation, generation and deployment of a cryptographic currency and the steps are designed in each section in the following way.

a. Transaction Data
b. Chaining the block
c. Generate signature hash
d. Proof of work specifies immutable blockchain
e. Governing the chain and defining the rules
f. Cryptocurrency in action

1.3 Code Examples

The examples are illustrated in Python, C++, and using the command line of a Unix-like operating system such as Linux or macOS. All code snippets are available in the GitHub repository (*https://github.com/Adityaraj1711/indi-coin*) in the *code* subdirectory of the main repo. Fork the project code, try the code examples, or submit corrections via GitHub.

All the code snippets can be replicated on most operating systems with a minimal installation of compilers and interpreters for the corresponding languages. Where necessary, we provide basic installation instructions and step-by-step examples of the output of those instructions.

Some of the code snippets and code output have been reformatted for print. In all such cases, the lines have been split by a backslash (\) character, followed by a newline character. When transcribing the examples, remove those two characters and join the lines again and you should see identical results as shown in the example.

All the code snippets use real values and calculations where possible, so that you can build from example to example and see the same results in any code you write to calculate the same values. For example, the private keys and corresponding public keys and addresses are all real. The sample transactions, blocks, and blockchain references have all been introduced and are part of the public ledger, so you can review them after resolving their nodes.

2. Problem Domain

2.1 Problems with cryptographic currency:

- The cryptographic currency is written on Etherium and Go that creates a challenge for developers to integrate with their projects or websites.
- Widespread adoption and usage of cryptographic coins for smaller transactions and/or large stores of value is slowed/prevented by existing established payment networks and a steep learning curve for crypto currency usage.
- The Mining of cryptographic coins requires energy and computationally intensive, with increasing amounts of both.
- The modern digital currency is almost assured not to become the global currency due to low Transactions per second, 7, a very low amount for all global transactions.
- By some people Bitcoin is also viewed as a viable alternative to traditional government controlled fiat currencies. At first Bitcoin was mostly known and used by IT enthusiasts and criminals. The huge impact on growing popularity of the Bitcoin was caused by global financial crisis, which reduced people trust in banking systems and government controlled currencies. People started to look for alternative ways to store they money and cryptographic currencies started to get more attention. Investing in Bitcoin, however, seems to be risky because currently it is very unstable and it's not backed up by any government or institution.

2.2 Problems with bank note:

- The banks collect all the information of an individual that completely reveals their identity.
- The banks stores the information in a centralised ledger, on hacking their systems could reveal millions of banks records of users.
- Transactions error is faced by every user due to system failure, link disruption.
- Black money, money laundering and major theft related problems which cannot track down the transaction in cash.

2.3 Problems with other digital currency:

- ✓ Main problems with centralized currencies are that controllers can increase their supply at subjective whims and impose arbitrary rules upon their user. Furthermore, systems can be destroyed or disrupted by attacking central point of control. Being decentralized, IndiCoin solves these problems.
- ✓ Currently most of the electronic payments rely on banks and payment system such as "PayPal". While it works well enough, such systems have several disadvantages. First of all, you have to provide your personal information to third parties, your

accounts can be frozen or their balance partially or completely confiscated, payments to certain legal entities may be refused.

- ✓ Credit card or payment system transaction fee may be several percent of the transaction. Electronic funds transfer from one bank account to another is also expensive (local transfers usually cost about 1 euro while international 10-50 euros) and may take a few days in some cheap transfer cases. It also has high costs. Another problem for merchants is that completely non-reversible payments are not possible, which leads to merchants asking for more information from their customers than they would otherwise need. Despite their best efforts, certain percentage of fraud is still unavoidable. In comparison Bitcoin is potentially anonymous, has very low or no transaction cost, transfer can be made worldwide with average confirmation time of 10 minutes, no authority can freeze or deny payments to your accounts and payments are non-reversible.
- ✓ Payment systems based on a trusted third party use data encryption to protect user information. In Indi coins case, nothing is actually encrypted and all transactions are made public. The advantage of this is that there is no database or accounts that can be hacked. However this puts a lot of responsibility into the hands of the user, who has to keep his wallet (private keys) secure. Indi Coins transactions are only tied to cryptographic keys, so privacy is also responsibility of the user.

3. Introduction

3.1 Project abstract:

INDI Coin is a cryptographic currency which is developed in the Python environment. Unlike Bitcoin, It can be easily integrated in any Webapp based on Python environment or Python framework like Flask or Django.
It is a peer-to-peer payment system and digital currency and uses cryptography to control the creation and transfer of money.

3.2 Purpose:

Indi Coin major purpose is to create a complete package capable of connecting various websites based on django and flask that is used to create small projects and give a encrypted currency transaction intergration besides connecting any third party payment gateway. The project is designed by keeping in mind all the constraints required for a transaction, the possible scenarios while a transaction is made and the methods are defined well for every exception so that their could be a complete log that could be maintained on the agency's profile. Moreover every user wallet has a history panel to verify each transaction.

3.3 History:

A concept called "crypto-currency" was first described in 1998 by Wei Dai. It was an idea of new form of money that uses cryptography to control its creation and transactions instead of central authority. The first successful implementation of the idea is Bitcoin which was introduced in 2009 by Satoshi Nakamoto [1]. Bitcoin is open source completely digital currency and peer-to-peer payment network which is powered by its users. Most commonly used unit in Bitcoin is also called Bitcoin (BTC).

Main problems with centralized currencies are that controllers can increase their supply at subjective whims and impose arbitrary rules upon their user. Furthermore, systems can be destroyed or disrupted by attacking central point of control. Being decentralized, Bitcoin solves these problems.

Currently most of the electronic payments rely on banks and payment system such as "PayPal". While it works well enough, such systems have several disadvantages. First of all, you have to provide your personal information to third parties, your accounts can be frozen or their balance partially or completely confiscated, payments to certain legal entities may be refused. It also has high costs. Credit card or payment system transaction fee may be several percent of the transaction. Electronic funds transfer from one bank account to another is also expensive (local transfers usually cost about 1 euro while international 10-50 euros) and may take a few days in some cheap transfer cases. Another problem for merchants is that completely non-reversible payments are not possible, which leads to merchants asking for more information from their customers than they would otherwise need. Despite their best efforts, certain percentage of fraud is still unavoidable. In comparison Bitcoin is potentially anonymous, has very low or no transaction cost, transfer can be made worldwide with average confirmation time of 10 minutes, no authority can freeze or deny payments to your accounts and payments are non-reversible.

Payment systems based on a trusted third party use data encryption to protect user information. In Bitcoins case, nothing is actually encrypted and all transactions are made public. The advantage of this is that there is no database or accounts that can be hacked. However this puts a lot of responsibility into the hands of the user, who has to keep his wallet (private keys) secure. Bitcoin transactions are only tied to cryptographic keys, so privacy is also responsibility of the user.

By some people Bitcoin is also viewed as a viable alternative to traditional government controlled fiat currencies. At first Bitcoin was mostly known and used by IT enthusiasts and criminals. The huge impact on growing popularity of the Bitcoin was caused by global financial crisis, which reduced people trust in banking systems and government controlled currencies. People started to look for alternative ways to store they money and cryptographic currencies started to get more attention. Investing in Bitcoin, however, seems to be risky because currently it is very unstable and it’s not backed up by any government or institution.

Since introduction of Bitcoin, many similar crypto-currencies emerged, which shares most of the source code with Bitcoin. For example, Litecoin (LTC) uses different hashing algorithm, has higher maximum number of coins and shorter confirmation time. Nevertheless, Bitcoin remains the most popular crypto-currency.

The aim of this paper is to analyze and describe how Bitcoin network and protocol operates.

4. Literature Survey

4.1 Background:

Since the begging of humanity man has been trying to figure out how to get rid of excessive goods while trying to obtain goods they require. At first man resorted to a barter economy but this has several impracticalities. Later people started to trade goods in change for valuable metals and the like. After some time this was changed into trading with money in different forms. After a while commodity money appeared which later evolved into fiat money.

These money could not be given out by the common laymen and monetary authorities emerged with monopoly on supplying the people with money.

Some liberal economists have for a long time questioned states and central banks monopoly on printing and giving out currencies. It has been surrounded by harsh laws and restrictions worldwide. Libertarians argue that an unregulated market benefits the society since the market powers allocates capital and resources to where it is needed the most. The Internet has provided a new forum where you can bypass the laws and regulations created by central banks and governments regarding fiat money. The latest advance has been made through the emergence of the digital currency Bitcoins that was created in the beginning of 2009.

4.2 Problem Formulation

Bitcoin has lately attracted the medias attention partly through increased acceptance as a legitimate means of payment with some multinational corporations and partly through large private investments. (See for example Popper & Lattman, *Never mind Facebook, Winklewoss Twins Rule in Digital Currency,* New York Times, Apr 11th 2013.)

As previously stated this new form of currency is not entirely uncontroversial. It stems from the notion that no central monetary authority should be exposed to the whim of a politician or monetary economist. The system in which the money is created should only be under the control of all of the people using it.

The system is based on complex algorithms and cryptography but despite this, users that are not computer whizzes or hackers are starting to adopt the new currency.

But the question remains: is this just a short trend generated by increasing media coverage or is there a real currency behind? Can Bitcoin persevere and serve a real currency like the established ones we use in daily life?

4.3 Question formulation

Can IndiCoin become an established currency and a legitimate means of payment?

4.4 Method and delimitation

Not many scientific articles and academic papers have been published on the topic, which is delimitation in its own. However, lately media has started to cover the phenomenon more closely and that is why I have chosen to use articles published by well renowned news papers and magazines, such as The Economist and the like. There is a small amount of academic work that has been published, but most of it from a legal point of view. I have deliberately left this part out since it is beyond the scope of this thesis.

Some have highlighted the subject from an economic point of view and I intend to use them in order to create a framework for my analysis. There is of course a great spectrum of views in how a currency could or should be structured but I have chosen to limit this part to the most prevalent points of view. Furthermore I will use standard macroeconomic monetary theories and viewpoints in order to shed some light on the Bitcoin's advantages and disadvantages.

I will take a qualitative approach to the subject in order to answer my question formulation. Starting with standard economic monetary theory to later examine the currency more closely. I will also use examples of other currencies that exhibit resemblance in its structure compared to Bitcoin.

4.5 Monetary standards

4.5.1 Commodity standard

To circumvent these issues with big transactions costs involved with a barter economy we introduce a currency. The units used as currency has varied much throughout history from shark teeth to metals such as gold or silver. This has become the base for one of the two regular standards used when creating a currency. The commodity standard is based on some kind of asset/good that is scarce by nature, most commonly gold or other scarce metals. This is not to be confused with specie standard where the money is backed by an underlying asset. The physical money itself has an intrinsic value when we are talking about commodity money. (Fregert & Jonung, 2010) I will use gold as an example to simplify. If you have a gold coin you have two uses for it. Firstly it can be used as a medium of exchange with all goods having a relative price to it. Secondly it has an intrinsic value, the gold in the coin is actually worth something. It is also by nature a finite recourse, which means that the total amount available is predetermined. This in turn implies that it is scarce by nature, and if people consider it desirable and worth owning its value will be high. The downside is that technical innovations and discoveries of new sources for the good can be found. This in turn can result in supply shocks that erode the value and purchasing power of the good. (Selign, 2013)

4.5.2 Fiat standard

Fiat money has no intrinsic value more than the paper it is printed on. The marginal cost of production is close to zero and has to be made artificially scarce by its issuer and thus contingent. The issuer will have incentives to print more money since its lacking a stable equilibrium, this can go in until you have printed so much that you in the end have a paper standard. Of course this comes with the price of lost purchasing power, but in the short term it can seem tempting to make short-term gains. Its value rests upon the belief that the issuer will not print more money than that it becomes worthless. The only way to maintain an equilibrium above the paper and ink price (or the marginal production cost) is to give the issuer monopolistic rights. However, the issue remains, by merely making the issuer a monopolistic producer does not entail that the incentive to print more money is gone. The producer also have to limit its supply of money to a below profit-maximizing quantity. This entails a money authority to be surrounded by regulations and/or laws so they do not succumb to an excessive printing of money. (Selign, 2010)

Selign presents a money matrix in his paper where different base monies are presented:

<u>Money matrix (incomplete)</u>

		Nonmonetary Use?	
		Yes	**No**
Scarcity	**Absolute**	Commodity	
	Contingent		Fiat

4.5.3 Coase Durable Standard

Now we have defined the two standard base monies. As visible from the matrix we can see that the basic functions of money is whether the monies scarcity is natural or has to be artificially made, contingent, or if its scarce by nature, absolute. It also shows if the money has some kind of nonmonetary use, or in other words has an intrinsic value. For example commodity money has an intrinsic value per definition (just think of gold) and there is a limited amount we can extract from nature whereas fiat money is artificially made and has no (or a very low) intrinsic value. The scarcity is not limited but has to be contrived by the issuer.

What also become evident when studying the matrix are the two blank spots. Firstly we have to find some sort of money that has some kind of nonmonetary use but has a contingent scarcity. Selign proposes that this could be something called *Coase Durable Money.* As the name suggests this includes a monopoly player. (Selign, 2010) If the monopoly actor has monopolistic rights to a good, which is not scarce by nature, he can control the supply of the good and thus the amount of the good in circulation in an economy. It is also possible that the monopolistic player has monopolistic rights to the technology needed to reproduce it. It is essential that the good is durable. Otherwise its value would diminish rapidly and be rendered useless. A Coase-durable standard poses the same problems that Friedman pointed out with fiat standard. The monopolist has incentives to issue more money to make short run gains. This will continue until its exchange value in the economy is equal to the marginal cost of production. A rational consumer will anticipate this and will therefore not be willing to pay more then the marginal cost of production, regardless if it is the first or the last unit offered. Coase proposes that you can get rid of this problem if the monopolistic actor offers a buy-back for a slightly lower price thus ensuring that the monopolistic player has an interest in keeping the good scarce (Coase, 1972). Another way for the monopolistic issuer to ensure the consumers it will not overproduce currency is something called public destruction. This requires the monopolistic issuer to publicly destroy units of its issued good to show that they are willing to maintain a price above the marginal production cost. (Bulow, 1982) This way the issuer gains the publics trust and convince them that the good is a sustainable means of trade that fulfil the prerequisites (*unit of account*, *store of value* and *medium of exchange*) earlier listed in this paper.

4.5.4 Synthetic Commodity standard

So for the final blank space in the matrix, a monetary standard that is absolute in its scarcity but has not got any nonmonetary use. Here Selign introduce us to something he calls *Synthetic Commodity Money*. At a first glance a synthetic commodity standard might seem very similar to a fiat standard. Once again the money itself lacks an intrinsic value and rests upon the users faith in the currency. The big question is how its scarcity is absolute rather than contingent. It could be argued that synthetic commodity money is just a type of rule-bound fiat standard. Selign writes:

"The difference warranting the separate designations is that real resource costs alone limit monetary base growth in a synthetic commodity-money regime, whereas in rule-based fiat money regimes, as these are conventionally understood, base growth is limited by positive *transactions* costs, including any penalties to which rule-violating authorities are subject" (Selign, 2013)

To clarify, the difference lies in how the money is controlled. In a ruled-based fiat standard the money authorities have rules they have to follow in order to not erode the value of the money and so forth. If these rules are not followed the authority can be penalized. These rules can be self-imposed or set by an outside party, for example a government. Selign points out that self-imposed regulations are subject to bias since the authority has the mandate to change rules accordingly to their own actions beforehand and can therefore escape with impunity. A synthetic commodity standard is not bound by such rules by legislatures and the like, but has the rules built into the system. Selign refers to Buchanan discusses this as an "automatic system", which can be compared with the "managed system" that a rule-bound fiat standard constitutes (Buchanan, James, (1962) "Predictability: The Criterion of Monetary Constitution"). This implies that no central monetary authority is needed and the system works all by itself. It does not need discretionary management nor rule enforcement, no penalizing system is needed in order for it to work. But since the system is artificial it is not just controlled by nature, so the standard is not vulnerable to sudden supply shocks stemming from discoveries of new sources of the currency. We do have control when constructing the system and can incorporate tools as automatic stabilisers and so forth if we desire to do so. (Selign, 2010)

Monetary policy has for long been a topic of discussion amongst economists. Two of the more prevalent economists have expressed their views as following:

“Money is too important to be left to the central bankers.” (Centralbanking.com, 2002)

And Keynes asserting the money is too important to be “sacrificed [...] to the operation of blind forces” (Keynes, 2006).

In this regard the synthetic commodity standard can be viewed as a compromise. Money is not left to the central bankers, but is not subjected to the whim of nature. The only time man has to agree is when constructing the system. After the system is started it cannot be altered neither by man nor nature. Given that this is agreed upon beforehand.

The completed matrix can be viewed in figure 2 below.

		Nonmonetary Use?	
		Yes	**No**
Scarcity	**Absolute**	Commodity	Synthetic Commodity
	Contingent	Coase Durable	Fiat

<u>Money matrix</u>

4.5.4.1 Iraq Swiss dinars, an unexpected experiment with synthetic commodity money

Synthetic commodity money has never been implemented and/or tested in an economy by a monetary authority. However a after a historic event a new form of monetary standard emerged, the synthetic commodity money. (Selign, 2013)

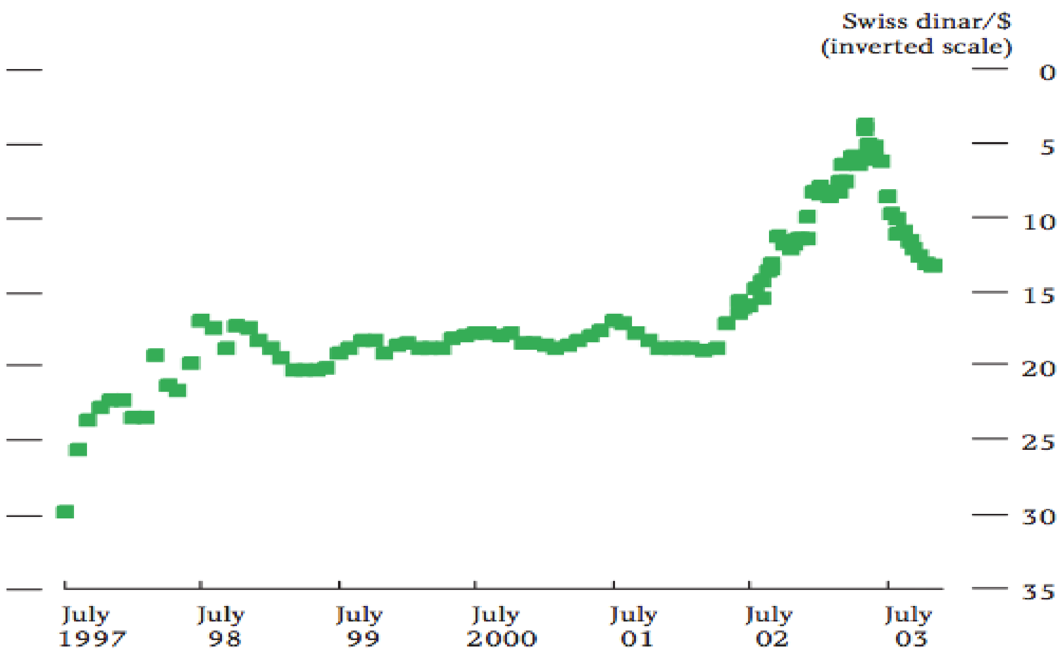
Before the invasion of Iraq in 1990s the monetary system constituted of the Iraq Dinars as a currency. The money was printed with Swiss plates, hence the name, in the U.K. and then distributed in Iraq. But during the American invasion sanctions was imposed against importing the currency into Iraq and Saddam Hussein implemented a new currency, the Saddam Dinar. He ordered for the old printing plates in Switzerland to be destroyed so no more could be printed. He also deemed the old currency illegal and forbid it to be used as a means of payment. The new currency was put into the economic system and after heavy printing of the money it soon started to depreciate. The overproduction of bills was not only by the government but also by counterfeiters. The new money was easy to forge which led to a large amount of illegal money circulating the system, worsening the depreciation of the Saddam dinar. (King, 2004)

In the north of Iraq, in Kurdistan people refused to trade with the new currency and resorted to use the Iraq Swiss dinar instead. A new system was created with a fixed amount of money in circulation and with no monetary institutions surveillance it. (King, 2004)

There was no monetary authority that could intervene in the economic system and it was entirely self-policed by its users. It had no intrinsic value and furthermore it was not based on or backed with any commodity. One issue was the deterioration of the actual bills. Since no new bills were put into the system the bills got worn and after a while, started to fall apart.

The Swiss dinar currency stabilized in 1998 and kept its value remarkably well As we can see in figure 3. Not until next American invasion in 2003 did it lose its value, and this because the Americans wanted to regulate its currency and make the whole country using only one currency. (King, 2004)

<u>Swiss dinar/Dollar exchange rate</u>

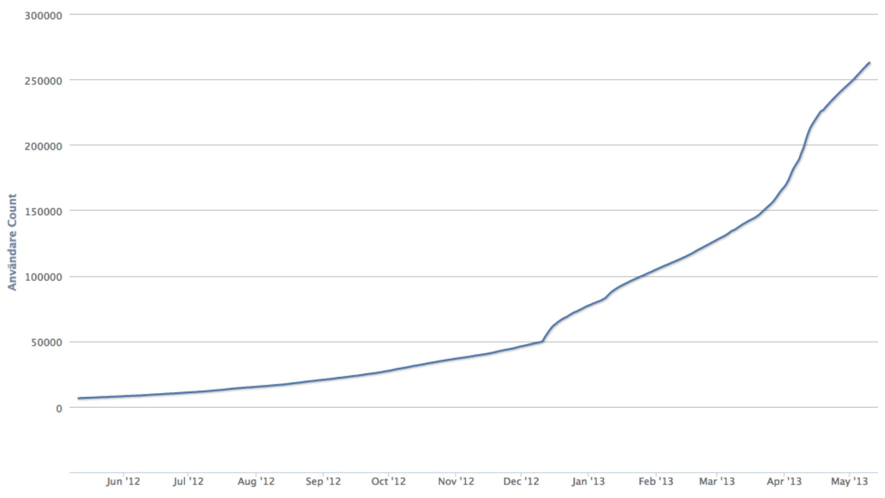


4.5.5 Brief history

In 2009 an article called "*Bitcoin: A Peer-to-Peer Electronic Cash System*" was published by the author Satoshi Nakamoto. It spoke about an electronic currency that you could only find on the Internet without any central authority. Another prerequisite for the system was that it should not rely on trust. (Nakamoto, 2009) Shortly after the publishing the system was launched, and this was the beginning of Bitcoin as a currency. Satoshi Nakamoto is now believed to be a pseudonym and he/she hasn't been heard from since the launch. (Economist, 2011)

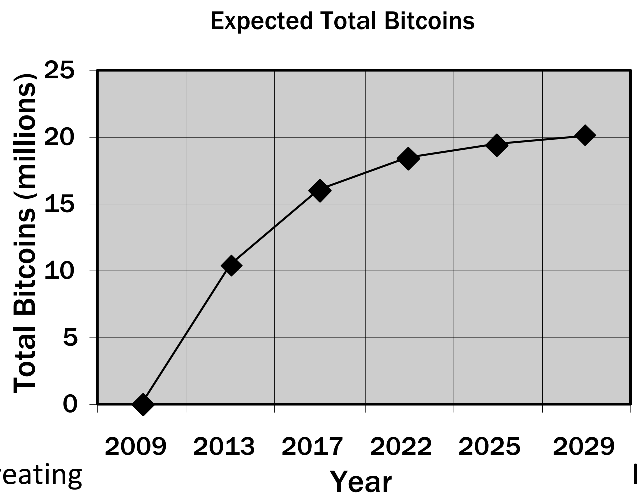
The creation of the currency was at most met with a lukewarm interest. Spikes can be seen between 2009 and 2012 but it did not gain any big amount of attention until 2013. This can also be observed as you go into statistics about the amount of users of the additional service Mywallet. Mywallet is used as a virtual wallet on your computer where you can store the currency. The growth rate of the amount of users took accelerated rapidly in December/January of 2012/13. At the moment of writing it is now at 263 100 individual users and growing. (Bitcoin Block Explorer)

Expected Amount of users of Mywallet



4.5.6 How IndiCoins are created

Indicoins are designed to mimic mining of minerals like Bitcoin. Take the extraction of gold as an example. To begin with it is easy to find new gold and the rate at which it is extracted is very high since it is easily accessible and there is a large supply. But as the extraction continues it gets harder to find since it is scarce and becomes even scarcer as more is mined. The mining continues until every piece of gold has been mined and all of it is in circulation in the economic system. Similarly with Indicoins; at the beginning the supply increases rapidly but at a decreasing rate until the creation-rate declines to zero. The rate is predetermined by algorithms and is automatically changed. Example - The creation of Bitcoin is to stop sometime in 2030 when around 21.1 million Bitcoins have been created. (Economist, 2011). Figure 5 illustrate how Bitcoins regressively are put into the system.



The process of creating Indicoins is in itself fairly technical for those not familiar with cryptography and computer programming. But I s simplified explanation is possible. The technology builds upon peer-to-peer networks, similar to downloading Bitorrents when downloading movies. (Economist, 2011) Peer-to-peer is when computer are exchanging information without a server in between. In other words when computers are exchanging information without a middleman that can be shut down.

To begin with the very first transaction the creator's computer forged 50 Bitcoins. This is announced in a network of users, also referred to as "miners" which has to authorize the creation and certify that the creation of the units is in fact legitimate. As a reward for this some of the 50 Bitcoins are distributed amongst the certifying users. After that the miners creates blocks of currency units that has to be approved and certified by the other miners. Using complex encryption techniques users are confirming each block of currency created.

This work is costly for individuals in terms of time and computer power but can be consider cheap for the network as a whole as the system grows. Making the verification-process into a forced-work task solves this i.e. all users are involved in this process. The verification process is constructed as a mathematical puzzle adapting to the current computing power i.e. if the network grows the mathematical puzzle gets more complex and conversely if it shrinks it gets easier. This is to maintain the constant growth rate of the currency creation. The time frame to create and approve one block is set to ten minutes. This means that a new block of currency units are sent into the system every ten minutes. By altering the size of the block, and thus the amount of currency created, the growth rate of the currency can regressively decline. As I mentioned earlier this is an automated process built into the algorithms. Figure 6 depicts the amount of Bitcoins that has been created since the start and it also shows the first decline in Bitcoin mining in December 2012. (Economist, 2011) At the moment of writing the amount of Bitcoins in circulation is around 11 128 000 Bitcoins. The rate at which they are created is about 300 coins per hour on average at the moment. But every four years the sizes of the blocks are halved. At the moment one block contains 50 currency units but will soon contain 25 units, after around four years one block will contain 12.5 units and so forth. (Bitcoin Block Explorer)

4.5.7 Transactions

When one user wants to transfer currency to another party he/she receives a virtual key, another key is created which is made public. The recipient using his/her private key retrieves the public key and all parties are now verified by the usage of their keys associated with their accounts or wallets. By using public-key encryption, which is often used in many other online dealings, a transfer cannot be made without the approval of both the transferring party and the recipient. This system is setup to side-step theft, and does so neatly but does not circumvent the problem with double spending. Spending money twice in this system could be said to be the cyber version of counterfeiting. In the real world funnelling the money through a single database that approves all transactions being made by checking a serial number on the money solves this. But since the whole point of Bitcoin is an open network with no centralized agent involved this has to be done another way, and this is where most other digital currencies has got stuck. This nut is cracked much like the creation of Bitcoin is, by complex mathematical algorithms and certification from the users. (Economist, 2011)

The cost related to the transactions is close to zero. Some costs are affiliated with the hardware and energy used to control the system. There are also small fees for those who whish not to be a part of the mining process but only want to exchange (read: buy) currencies such as dollars or euros into Bitcoins via websites such as Mt. Gox.

4.5.8 Lack of centralized supervision

The system itself evidently relies upon the users self-policing it. The system is constructed so that one person can not control it as long as it have not got more than half of the network's computing power. The incentive for making sure no counterfeiting, double payments or illegal creation is taking place is both the payment for helping to mine Bitcoins and making sure your own Bitcoins aren't eroding in value from an over production of illegal currency, or in economic terms: inflation. (Economist, 2011)

However, this implies that a community of users can pool their computing power and can then be able to mine more per person then they otherwise would be able to if they worked individually. This means that you actually can solve the mathematical problems faster then the algorithm has calculated, since the algorithm is based on the individual's computer power. This has to some extent occurred, but has not yet posed to be a valid concern.

5. Major Objective and Scope of project

5.1 Goals

- ✓ The main goal of this project is to create a decentralized banking system in which cryptographically verified transactions are made for small money transactions to large money transactions. The transactions made will be decentralized and the data of each transaction will be recorded all over the miners in the world.
- ✓ The prime goal is to link every website and software that includes/have a gateway of money transaction with IndiCoins. The Indi coin will be connected and each transaction can server as a single block of the chain.
- ✓ The project is comprised of a package written in Python with a very light framework called Django so that the integration will be easier to the developers.
- ✓ For now, the project focuses on the websites or softwares written in Python language. All a developer has to do is paste the package or install it by using PIP INSTALL from the command line, in the virtual environment.
- ✓ Having a decentralized storage of transaction will motivate users to share their public keys freely and will secure them from the theft of personal credentials that users usually deny to enter into small websites.
- ✓ Providing each user a wallet and the mining capability to mine the transactions that will reward them, and so in this way easy earning could be possible to users.
- ✓ For now, the developers has a lot of integration problems as it uses the Go Etherium languages and so the devlelopers could easily handle the encrypted transactions with a proper management.
- ✓ Main problems with centralized currencies are that controllers can increase their supply at subjective whims and impose arbitrary rules upon their user. Furthermore, systems can be destroyed or disrupted by attacking central point of control. Being decentralized, IndiCoin solves these problems.
- ✓ Currently most of the electronic payments rely on banks and payment system such as “PayPal”. While it works well enough, such systems have several disadvantages. First of all, you have to provide your personal information to third parties, your accounts can be frozen or their balance partially or completely confiscated, payments to certain legal entities may be refused.
- ✓ Credit card or payment system transaction fee may be several percent of the transaction. Electronic funds transfer from one bank account to another is also expensive (local transfers usually cost about 1 euro while international 10-50 euros) and may take a few days in some cheap transfer cases.

5.2 Scope of project

The scope of this project can be expanded and implemented to country level after some restrictions and norms. Legal actions and the country’s legislation could allow the package to be integrated and can be maintained by the respective organization.

The enhancement made in the project will be capable of using a decentralized currency irrespective of a user’s profile. So, problems like money laundering and disrupted behavior which includes black marketing could be solved. However, the implementation is a tedious work and so this project also requires a prosperous supervision under some regulatory organization or a third party verifier.

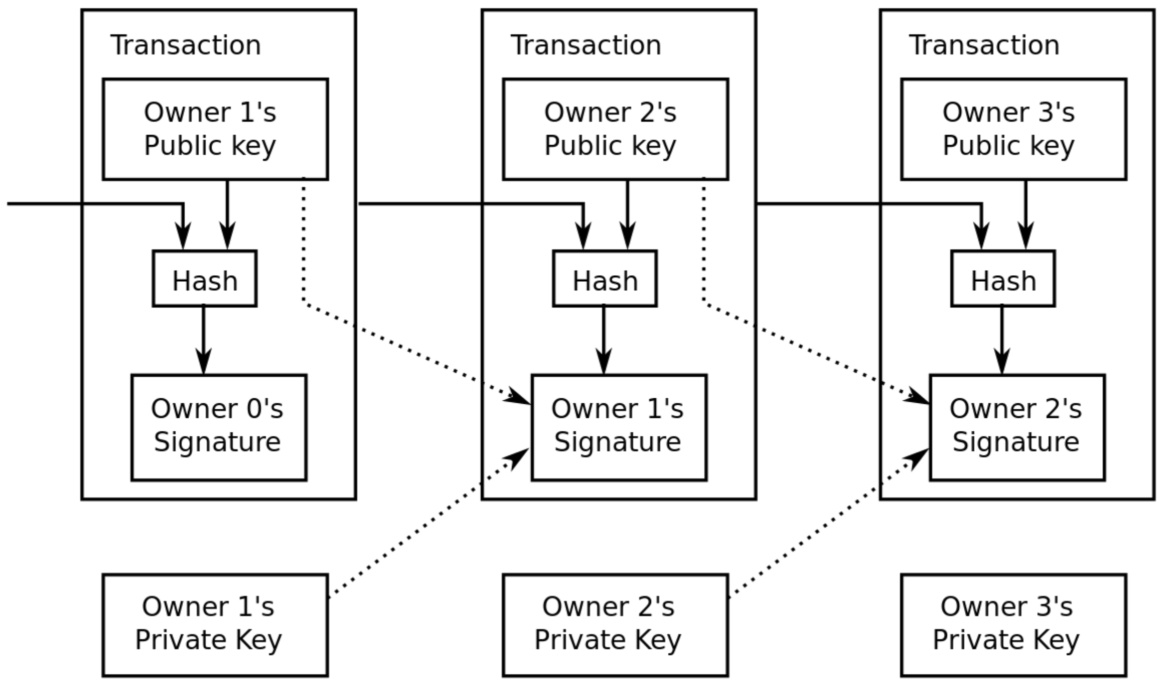
6. Problem analysis and requirement specifications

A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for nonreversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party. What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

7. Detail design/modeling

7.1 Transaction visual



The transaction is encrypted by SHA 256 encryption algorithm.

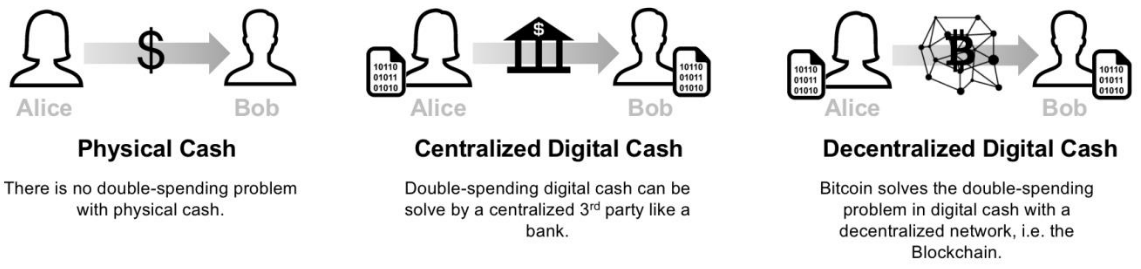
The transaction values are stored in the blocks which reside in the chain, the chain is made of the blocks and the first block is called Genesis block. The genesis block is a dummy block with a hash and the blocks appended stores the hashes of previous blocks.

7.2 A decentralized solution for double spending

To solve the double-spending problem, Satoshi proposed a public ledger, i.e., IndiCoin's blockchain to keep track of all transactions in the network. IndiCoin's blockchain has the following characteristics:

- ✓ Distributed: The ledger is replicated across a number of computers, rather than being stored on a central server. Any computer with an internet connection can download a full copy of the blockchain.
- ✓ Cryptographic: Cryptography is used to make sure that the sender owns the inidcoin that she's trying to send, and to decide how the transactions are added to the blockchain.
- ✓ Immutable: The blockchain can be changed in append only fashion. In other words, transactions can only be added to the blockchain but cannot be deleted or modified.
- ✓ Uses Proof of Work (PoW): A special type of participants in the network called miners compete on searching for the solution to a cryptographic puzzle that will

allow them to add a block of transactions to IndiCoin's blockchain. This process is called Proof of Work and it allows the system to be secure.


Alice
$
Bob
Physical Cash
There is no double-spending problem with physical cash.
Alice
Bob
Centralized Digital Cash
Double-spending digital cash can be solve by a centralized 3rd party like a bank.
Alice
Bob
Decentralized Digital Cash
Bitcoin solves the double-spending problem in digital cash with a decentralized network, i.e. the Blockchain.

Sending Indi-Coin money goes as follows:

- ✓ Step 1 (one-time effort):
  - ✓ Create a Indicoin wallet. For a person to send or receive Indicoins, she needs to create a Indicoin wallet. A Indicoin wallet stores 2 pieces of information: A private key and a public key. The private key is a secret number that allows the owner to send Indicoin to another user, or spend Indicoins on services that accept them as payment method. The public key is a number that is needed to receive Indicoins. The public key is also referred to as Indicoin address (not entirely true, but for simplicity we will assume that the public key and the Indicoin address are the same). Note that the wallet doesn't store the Indicoins themselves. Information about Indicoins balances are stored on the Indicoin's blockchain.
- ✓ Step 2:
  - ✓ Create a Indicoin transaction. If Alice wants to send 1 Coin to Bob, Alice needs to connect to her Indicoin wallet using her private key, and create a transaction that contains the amount of Indicoins she wants to send and the address where she wants to send them (in this case Bob's public address).
- ✓ Step 3:
  - ✓ Broadcast the transaction to Indicoin's network. Once Alice creates the bitcoin transaction, she needs to broadcast this transaction to the entire Indicoin's network.
- ✓ Step 4:
  - ✓ Confirm the transaction. A miner listening to Indicoin's network authenticates the transaction using Alice's public key, confirms that Alice has enough Indicoins in her wallet (in this case at least 1 BTC), and adds a new record to Indicoin's Blockchain containing the details of the transaction.
- ✓ Step 5:
  - ✓ Broadcast the blockchain change to all miners. Once the transaction is confirmed, the miner should broadcast the blockchain change to all miners to make sure that their copies of the blockchain are all in sync.

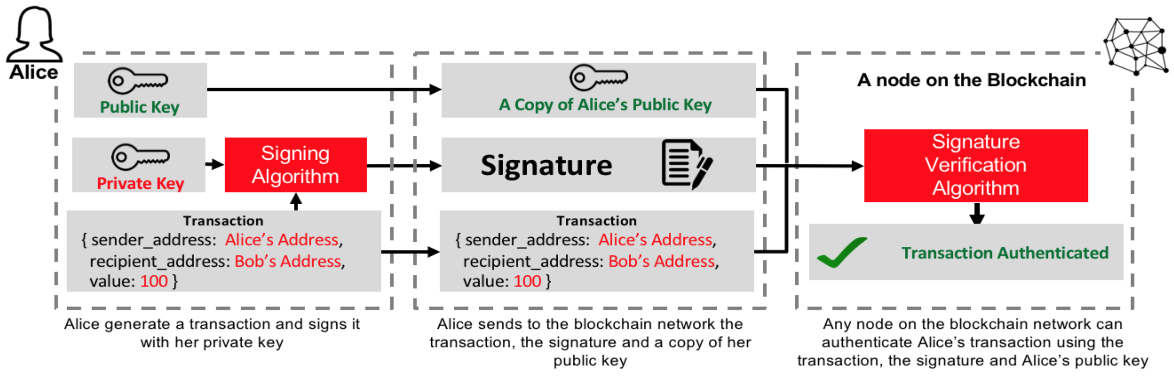
7.3 Public key cryptography

Public-key cryptography, or asymmetrical cryptography, is any cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. This accomplishes two functions: authentication, where the public key verifies a holder of the paired private key sent the message, and encryption, where only the paired private key holder can decrypt the message encrypted with the public key.

RSA and Elliptic Curve Digital Signature (ECDSA) are the most popular public-key cryptography algorithms.

In the case of Indicoin, ECDSA algorithm is used to generate Indicoin wallets. Indicoin uses a variety of keys and addresses, but for the sake of simplicity, we will assume in this blog post that each Indicoin wallet has one pair of private/public keys and that a Indicoin address is the wallet's public key.

To send or receive BTCs, a user starts by generating a wallet which contains a pair of private and public keys. If Alice wants to send Bob some BTCs, she creates a transaction in which she enters both her and Bob's public keys, and the amount of BTCs she wants to send. She then sign the transaction using her private key. A computer on the blockchain uses Alice's public key to verify that the transaction is authentic and adds the transaction to a block that will be later added to the blockchain.

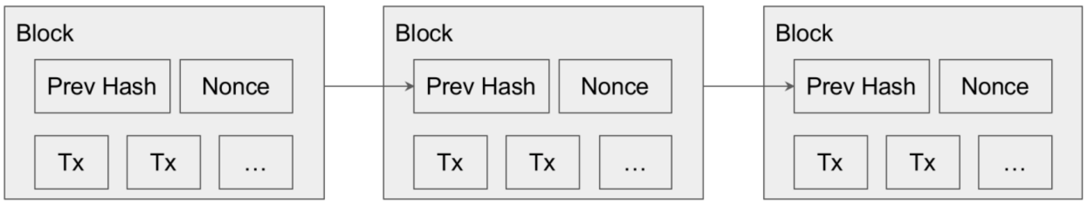

Alice
Public Key
Private Key
Signing Algorithm
Transaction
{ sender_address: Alice's Address,
recipient_address: Bob's Address,
value: 100 }
A Copy of Alice's Public Key
Signature
Transaction
{ sender_address: Alice's Address,
recipient_address: Bob's Address,
value: 100 }
A node on the Blockchain
Signature Verification Algorithm
Transaction Authenticated
Alice generate a transaction and signs it with her private key
Alice sends to the blockchain network the transaction, the signature and a copy of her public key
Any node on the blockchain network can authenticate Alice's transaction using the transaction, the signature and Alice's public key

Authentication Process for Transactions on the Blockchain

7.4 From Block to blockchain

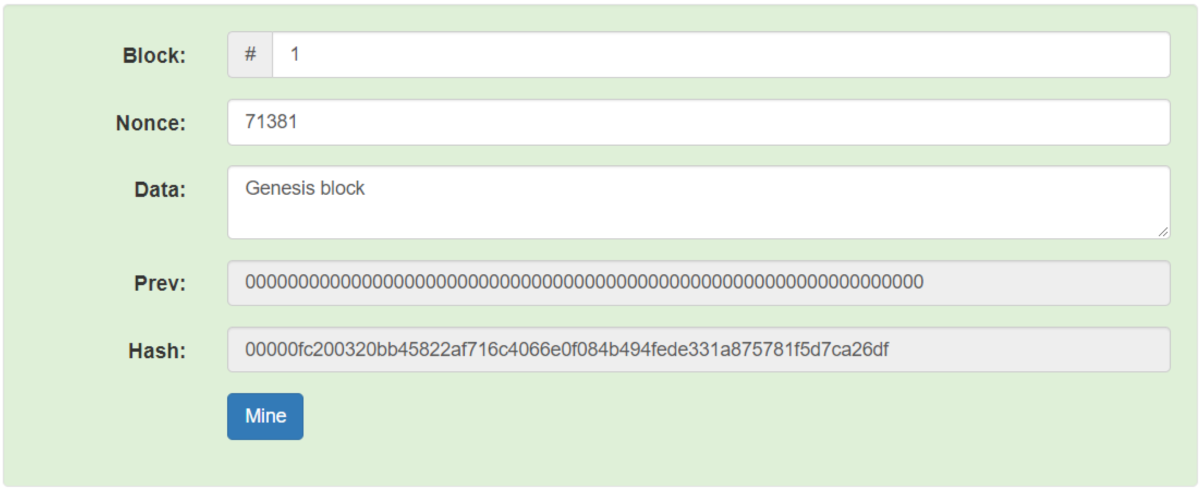
Transactions are grouped in blocks and blocks are appended to the blockchain. In order to create a chain of blocks, each new block uses the previous block's hash as part of its data. To create a new block, a miner selects a set of transactions, adds the previous block's hash and mines the block in a similar fashion described above.

Any changes to the data in any block will affect all the hash values of the blocks that come after it and they will become invalid. This gives the blockchain its immutability characteristic.

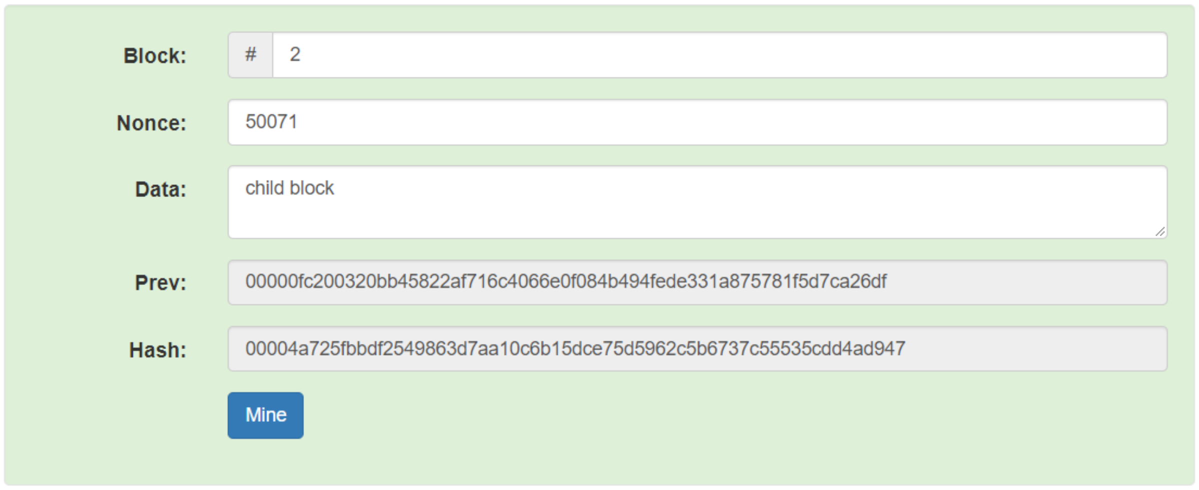


Blocks are chained together using the previous block's hash to form a Blockchain.

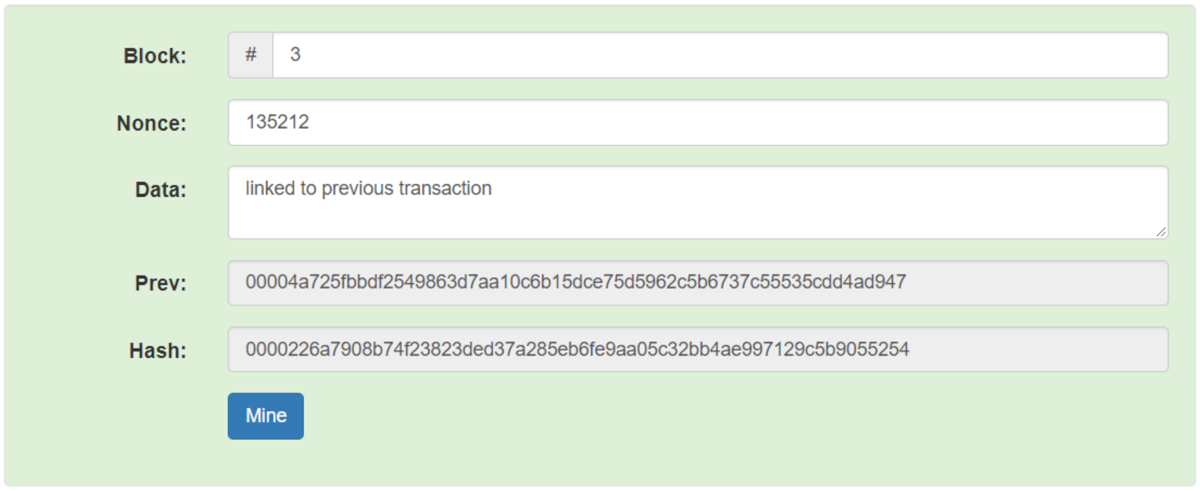
When edited data in the "Data" text box or change the nonce value, you can notice the change in the hash value and the "Prev" value (previous hash) of the next block. You can simulate the mining process by clicking on the “Mine” button of each individual block.



Block:	# 1
Nonce:	71381
Data:	Genesis block
Prev:	0000000000000000000000000000000000000000000000000000000000000000
Hash:	00000fc200320bb45822af716c4066e0f084b494fede331a875781f5d7ca26df



Block:	# 2
Nonce:	50071
Data:	child block
Prev:	00000fc200320bb45822af716c4066e0f084b494fede331a875781f5d7ca26df
Hash:	00004a725fbbdf2549863d7aa10c6b15dce75d5962c5b6737c55535cdd4ad947



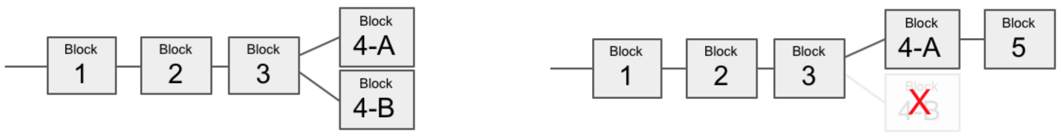
Block:	# 3
Nonce:	135212
Data:	linked to previous transaction
Prev:	00004a725fbbdf2549863d7aa10c6b15dce75d5962c5b6737c55535cdd4ad947
Hash:	0000226a7908b74f23823ded37a285eb6fe9aa05c32bb4ae997129c5b9055254

7.5 Adding Block to blockchain

All the miners in the Indicoin network compete with each other to find a valid block that will be added to the blockchain and get the reward from the network. Finding a nonce that validated a block is rare, but because of the number of miners, the probability of a miner in the network validating a block is extremely high. The first miner to submit a valid block gets his block added to the blockchain and receives the reward in Indicoins. But what happens if two miners or more submit their blocks at the same time?

Resolving Conflicts:

If 2 miners solve a block at almost the same time, then we will have 2 different blockchains in the network, and we need to wait for the next block to resolve the conflict. Some miners will decide to mine on top of blockchain 1 and others on top of blockchain 2. The first miner to find a new block resolves the conflict. If the new block was mined on top of blockchain 1, then blockchain 2 becomes invalid, the reward of the previous block goes to the miner from blockchain 1 and the transactions that were part of blockchain 2 and weren't added to the blockchain go back to the transactions pool and get added to the next blocks. In short, if there is a conflict on the blockchain, then the the longest chain wins.



Resolving conflicts - The longest chain wins

7.6 Blockchain and Double-spending

- ✓ Race Attack

An attacker sends the same coin in rapid succession to two different addresses. To prevent from this attack, it is recommended to wait for at least one block confirmation before accepting the payment.

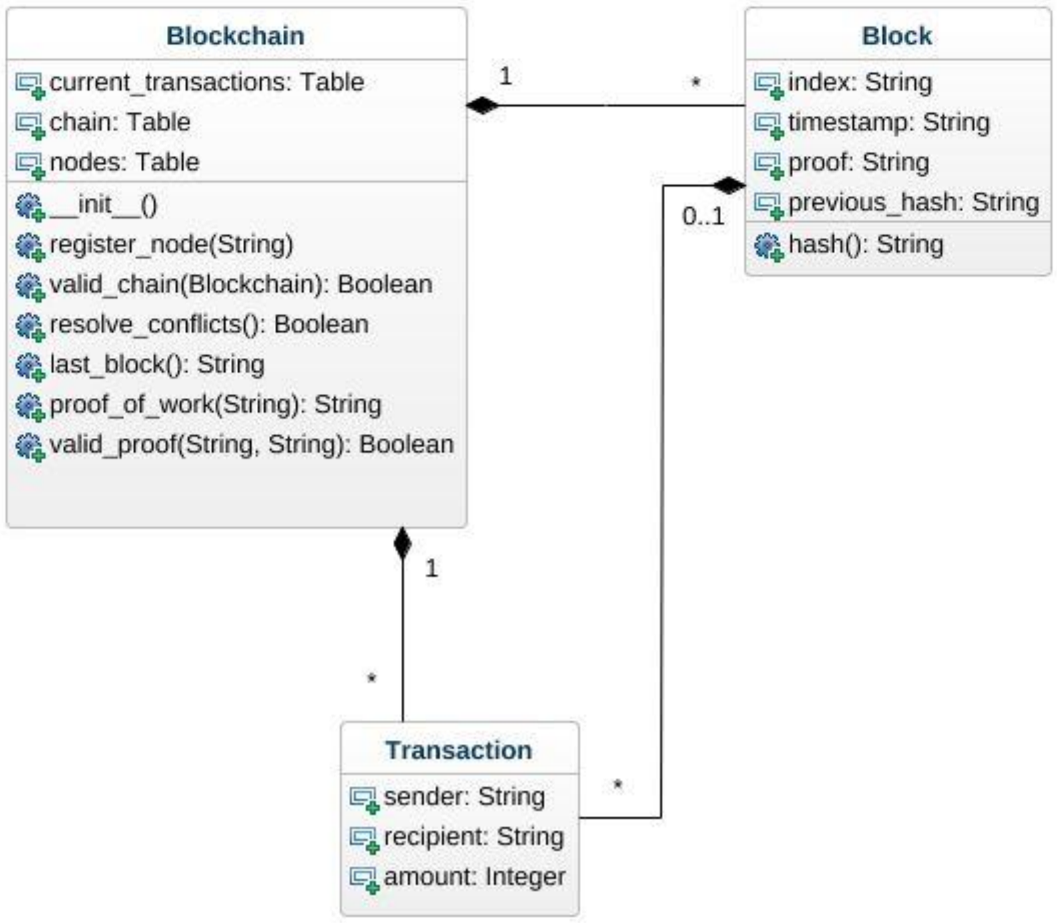
- ✓ Finney Attack

An attacker pre-mines a block with a transaction, and spends the same coins in a second transaction before releasing the block. In this scenario, the second transaction will not be validated. To prevent from this attack, it is recommended to wait for at least 6 block confirmations before accepting the payment. [3]

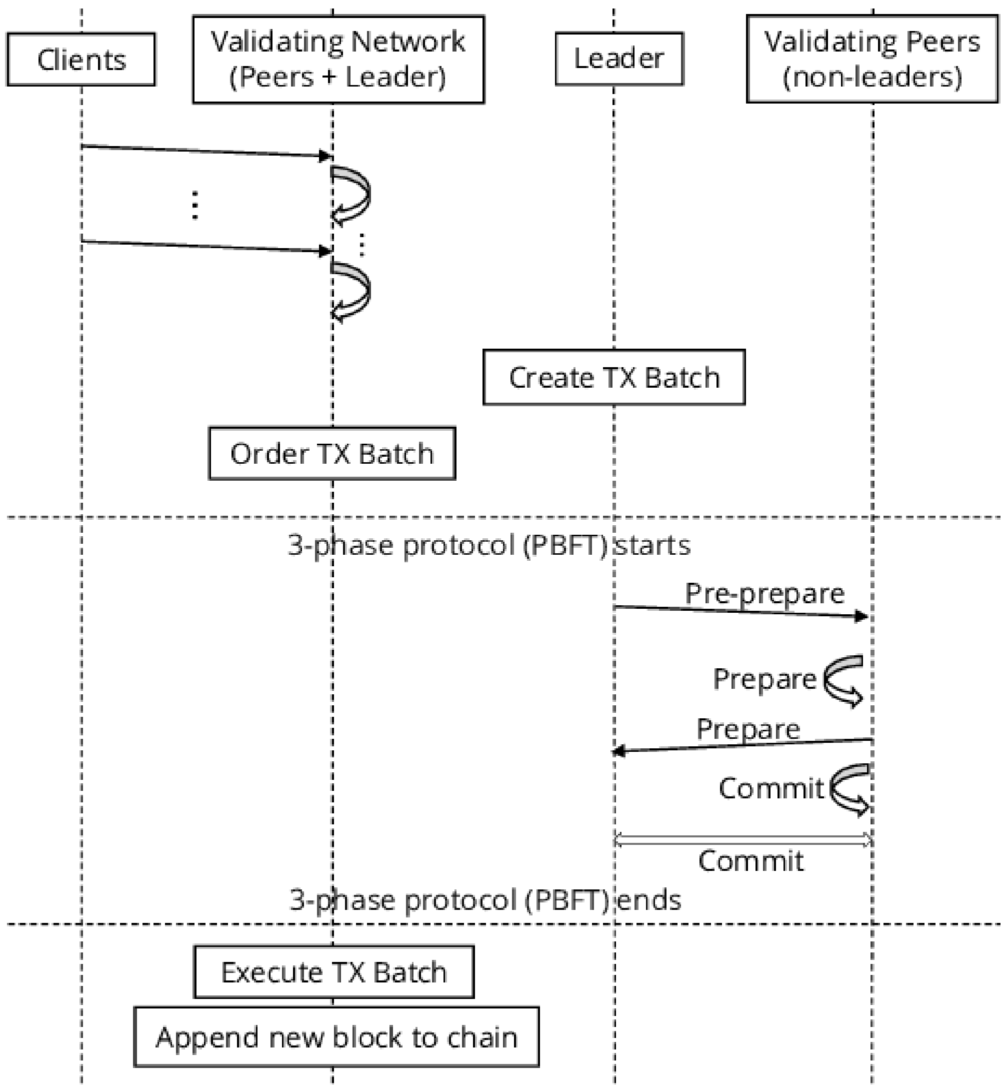
- ✓ Majority Attack (also called 51% attack)

In this attack, the attacker owns 51% of the computing power of the network. The attacker starts by making a transaction that is brodcasted to the entire network, and then mines a private blockchain where he double-spends the coins of the previous transaction. Since the attacker owns the majority of the computing power, he is guaranteed that he will have at some point a longer chain than the "honest" network. He can then release his longer blockchain that will replace the "honest" blockchain and cancel the original transaction. This attack is highly unlikely, as it's very expensive in blockchain networks like Bitcoin.

7.7 Class Diagram of Blockchain



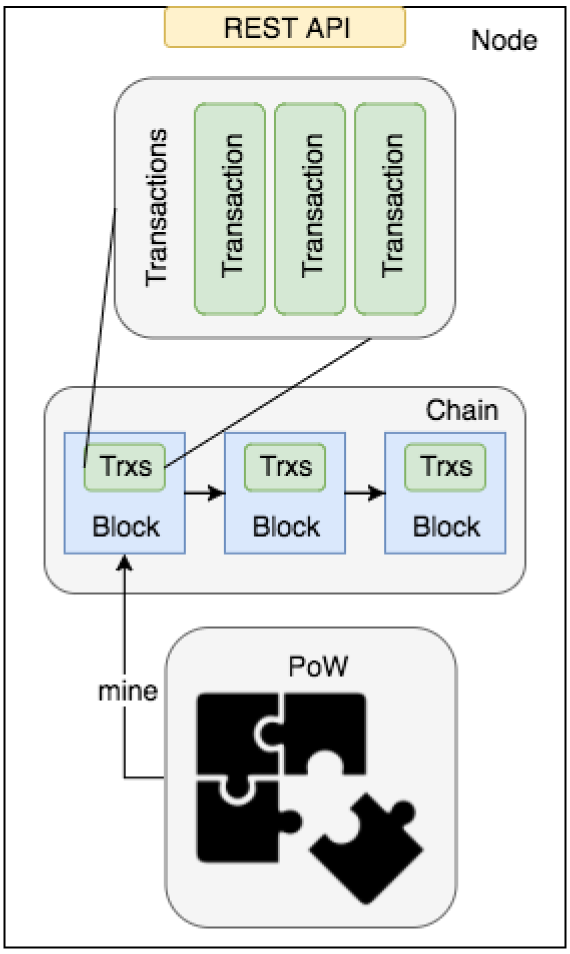
7.8 Sequence diagram of transaction


Clients
Validating Network (Peers + Leader)
Leader
Validating Peers (non-leaders)
Create TX Batch
Order TX Batch
3-phase protocol (PBFT) starts
Pre-prepare
Prepare
Prepare
Commit
Commit
3-phase protocol (PBFT) ends
Execute TX Batch
Append new block to chain

7.9 The Chain Model

Scalachain is based on a blockchain model that is a simplification of the Bitcoin one.

The main components of our blockchain model are the Transaction, the Chain, the Proof of Work (PoW) algorithm, and the Node. The transactions are stored inside the blocks of the chain, that are mined using the PoW. The node is the server that runs the blockchain.



Transaction

Transactions register the movement of coins between two entities. Every transaction is composed by a sender, a recipient, and an amount of coin. Transactions will be registered inside the blocks of our blockchain.

Chain

The chain is a linked list of blocks containing a list of transactions. Every block of the chain has an index, the proof that validates it (more on this later), the list of transactions, the hash of the previous block, the list of previous blocks, and a timestamp. Every block is chained to the previous one by its hash, that is computed converting the block to a JSON string and then hashing it through a SHA-256 hashing function.

PoW

The PoW algorithm is required to mine the blocks composing the blockchain. The idea is to solve a cryptographic puzzle that is hard to solve, but easy to verify having the proof. The PoW algorithm that is implemented in Scalachain is similar to the Bitcoin one (based on Hashcash). It consists in finding a hash with N leading zeros, that is computed starting from the hash of the last block and a number, that is the proof of our algorithm.

We can formalize it as:

NzerosHash = SHA-256(previousNodeHash + proof)

The higher is N, the harder is to find the proof. In Scalachain N=4 (It will be configurable eventually).

Node

The Node is the server running our blockchain. It provides some REST API to interact with it and perform basic operations such as send a new transaction, get the list of pending transactions, mine a block, and get the current status of the blockchain.

✓ **Blockchain implementation in Scala**

- transactions
- the chain of blocks containing lists of transactions
- the PoW algorithm to mine new blocks

These components are the essential parts of a blockchain.

Transaction
The transaction is a very simple object: it has a sender, a recipient and a value. We can implement it as a simple `case class`.

Chain

The chain is the core of our blockchain: it is a linked list of blocks containing transactions.

8. Hardware/Software Platform Environment

8.1 A blockchain implementation in Python

The project blockchain have the following features :

- ✓ Possibility of adding multiple nodes to the blockchain
- ✓ Proof of Work (PoW)
- ✓ Simple conflict resolution between nodes
- ✓ Transactions with RSA encryption

The project blockchain client will have the following features :

- ✓ Wallets generation using Public/Private key encryption (based on RSA algorithm)
- ✓ Generation of transactions with RSA encryption

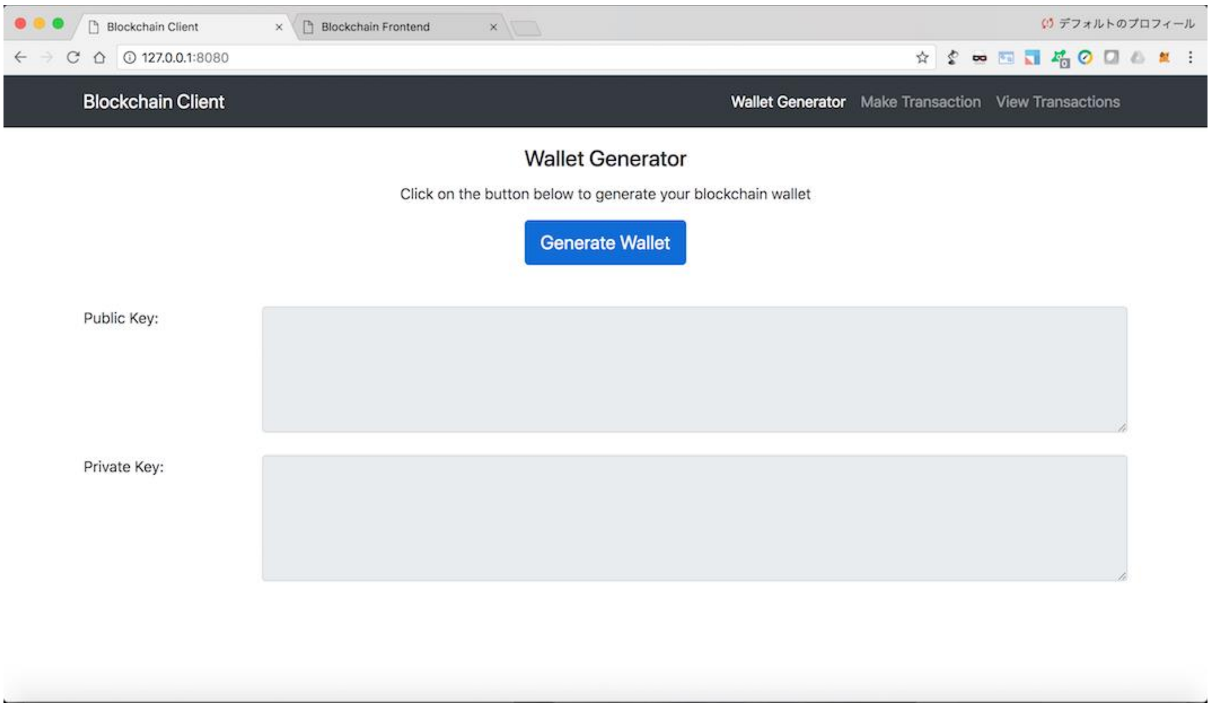
The project also implements 2 dashboards :

- ✓ "Blockchain Frontend" for miners
- ✓ "Blockchain Client" for users to generate wallets and send coins

The 2 dashboard are implemented from scratch using HTML/CSS/JS.

8.2 A blockchain client implementation

You can start the blockchain client from the terminal by going to the `blockchain_client` folder, and typing `python blockchain_client.py`. In your browser, go to `http://localhost:8080` and you'll see the dashboard below.



The dashboard has 3 tabs in the navigation bar:

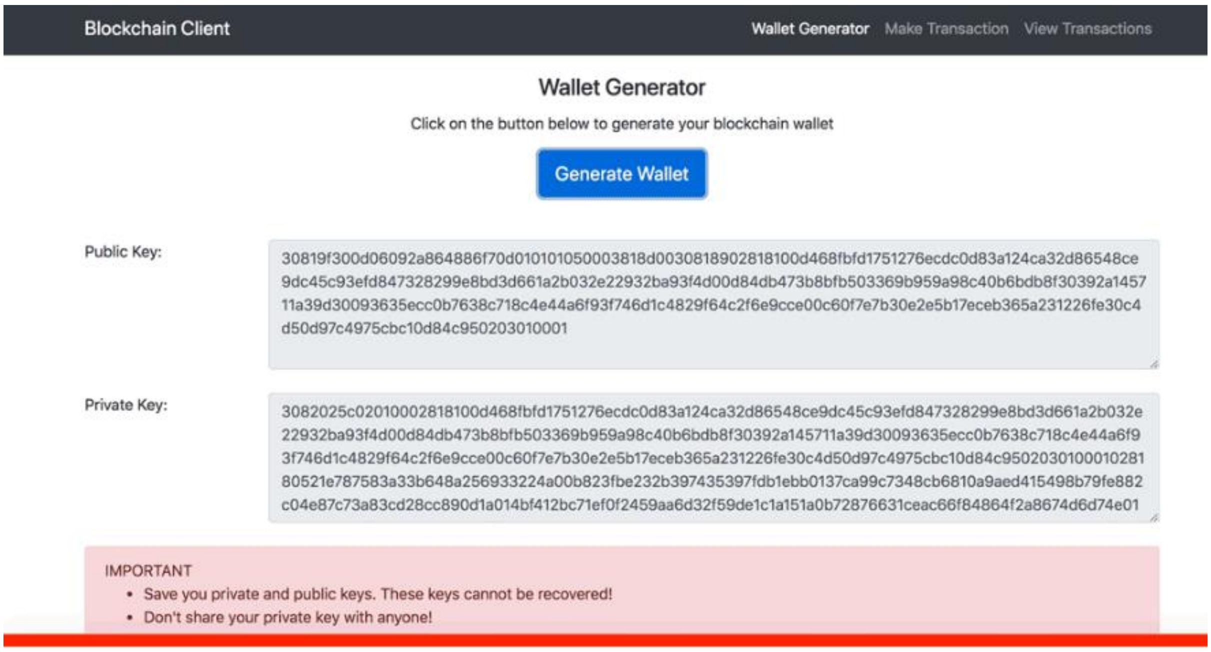
- ✓ Wallet Generator: To generate wallets (Public/Private keys pair) using RSA encryption algorithm
- ✓ Make Transaction: To generate transactions and send them to a blockchain node
- ✓ View Transactions: To view the transactions that are on the blockchain

In order to make or view transactions, you will need at least one blockchain node running (to be covered in next section).

Below is some explanation of the most important parts in the `blockchain_client.py` code.

We define a python class that we name `Transaction` that has 4 attributes `sender_address`, `sender_private_key`, `recipient_address`, `value`. These are the 4 pieces of information that a sender needs to create a transaction.

The `to_dict()` method returns the transaction information in a Python dictionary format (without the sender's private key). The `sign_transaction()` method takes the transaction information (without the sender's private key) and signs it using the sender's private key.



Wallet Generator

Click on the button below to generate your blockchain wallet

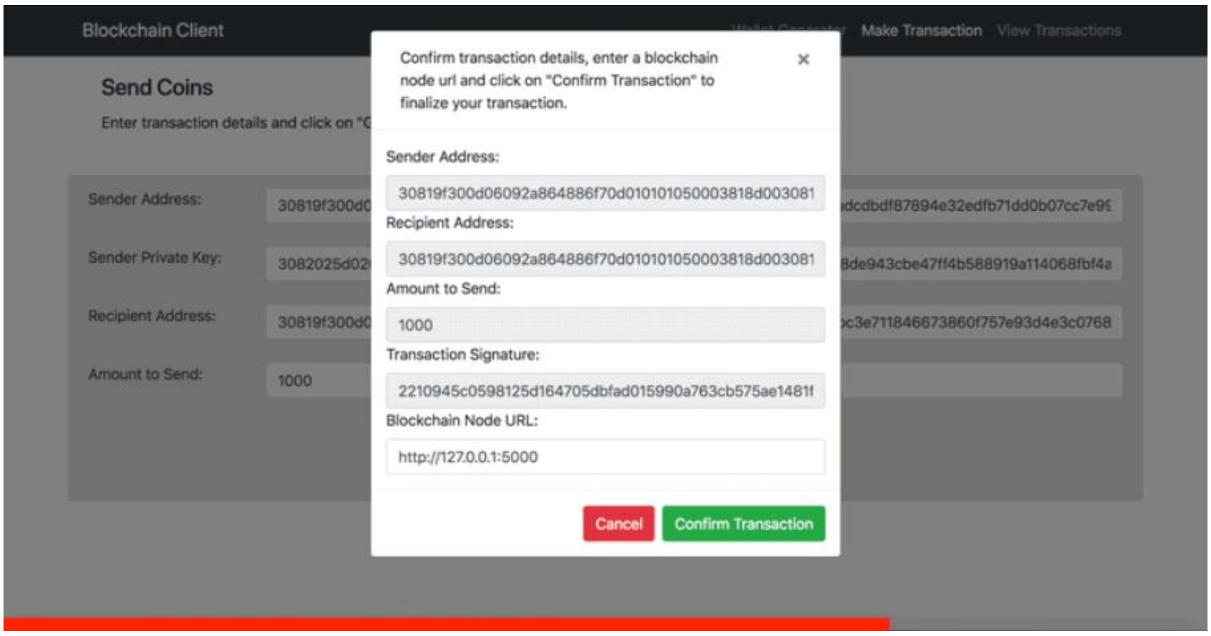
Public Key:

30819f300d06092a864886f70d010101050003818d0030818902818100d468fbfd1751276ecdc0d83a124ca32d86548ce9dc45c93efd847328299e8bd3d661a2b032e22932ba93f4d00d84db473b8bfb503369b959a98c40b6bdb8f30392a145711a39d30093635ecc0b7638c718c4e44a6f93f746d1c4829f64c2f6e9cce00c60f7e7b30e2e5b17eceb365a231226fe30c4d50d97c4975cbc10d84c950203010001

Private Key:

3082025c02010002818100d468fbfd1751276ecdc0d83a124ca32d86548ce9dc45c93efd847328299e8bd3d661a2b032e22932ba93f4d00d84db473b8bfb503369b959a98c40b6bdb8f30392a145711a39d30093635ecc0b7638c718c4e44a6f93f746d1c4829f64c2f6e9cce00c60f7e7b30e2e5b17eceb365a231226fe30c4d50d97c4975cbc10d84c950203010001028180521e787583a33b648a256933224a00b823fbe232b397435397fdb1ebb0137ca99c7348cb6810a9aed415498b79fe882c04e87c73a83cd28cc890d1a014bf412bc71ef0f2459aa6d32f59de1c1a151a0b72876631ceac66f84864f2a8674d6d74e01

IMPORTANT
- Save you private and public keys. These keys cannot be recovered!
- Don't share your private key with anyone!



Send Coins

Enter transaction details and click on "C

Sender Address: 30819f300d0 …dcdbdf87894e32edfb71dd0b07cc7e9€

Sender Private Key: 3082025d02 …8de943cbe47ff4b588919a114068fbf4a

Recipient Address: 30819f300d0 …c3e711846673860f757e93d4e3c0768

Amount to Send: 1000

Confirm transaction details, enter a blockchain node url and click on "Confirm Transaction" to finalize your transaction.

Sender Address:

30819f300d06092a864886f70d010101050003818d003081

Recipient Address:

30819f300d06092a864886f70d010101050003818d003081

Amount to Send:

1000

Transaction Signature:

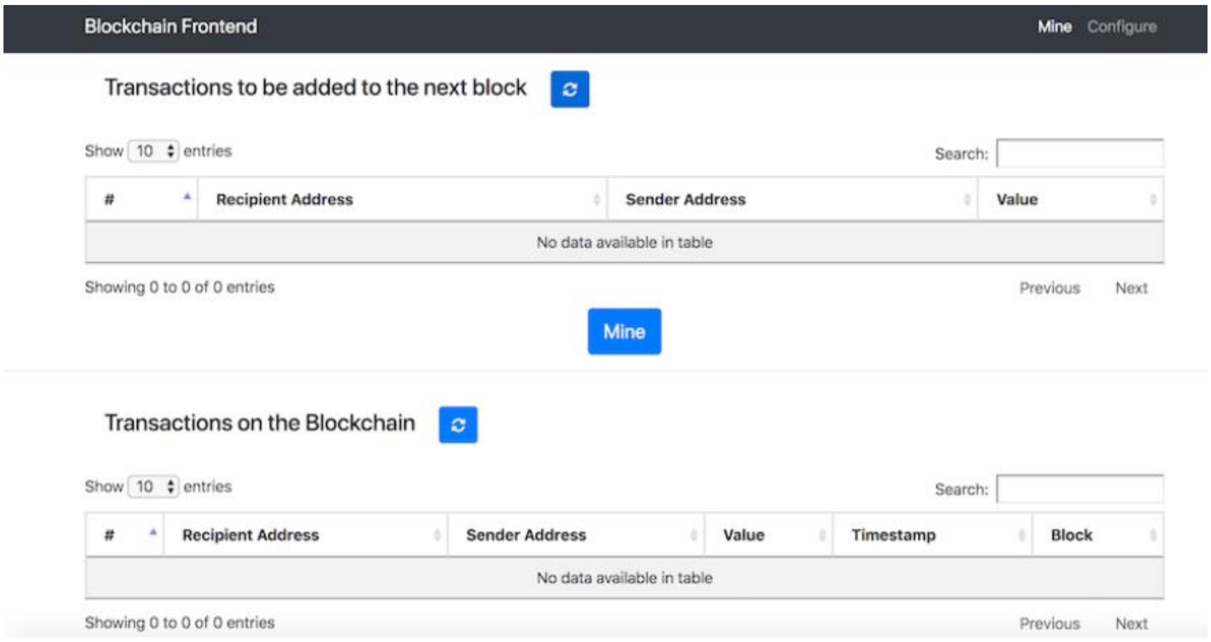
2210945c0598125d164705dbfad015990a763cb575ae1481f

Blockchain Node URL:

http://127.0.0.1:5000

8.3 A blockchain implementation

You can start a blockchain node from the terminal by going to the `blockchain` folder, and type `python blockchain_client.py` or `python blockchain_client.py -p <PORT NUMBER>`. If you don't specify a port number, it will default to port 5000. In your browser, go to `http://localhost:<PORT NUMBER>` to see the blockchain frontend dashboard.



Transactions to be added to the next block

Show 10 entries Search:

#	Recipient Address	Sender Address	Value
No data available in table			

Showing 0 to 0 of 0 entries Previous Next

Transactions on the Blockchain

Show 10 entries Search:

#	Recipient Address	Sender Address	Value	Timestamp	Block
No data available in table					

Showing 0 to 0 of 0 entries Previous Next

The dashboard has 2 tabs in the navigation bar:

- Mine: For viewing transactions and blockchain data, and for mining new blocks of transactions.
- Configure: For configuring connections between the different blockchain nodes.

Below is some explanation of the most important parts in the `blockchain.py` code.

We start by defining a `Blockchain` class that has the following attributes:

- `transactions`: List of transactions that will be added to the next block.
- `chain`: The actual blockchain which is an array of blocks.
- `nodes`: A set containing node urls. The blockchain uses these nodes to retrieve blockchain data from other nodes and updates its blockchain if they're not in sync.
- `node_id`: A random string to identify the blockchain node.

The `Blockchain` class also implements the following methods:

- `register_node(node_url)`: Adds a new blockchain node to the list of nodes.
- `verify_transaction_signature(sender_address, signature, transaction)`: Checks that the provided signature corresponds to transaction signed by the public key (sender_address).
- `submit_transaction(sender_address, recipient_address, value, signature)`: Adds a transaction to list of transactions if the signature verified.
- `create_block(nonce, previous_hash)`: Adds a block of transactions to the blockchain.
- `hash(block)`: Create a SHA-256 hash of a block.
- `proof_of_work()`: Proof of work algorithm. Looks for a nonce that satisfies the mining condition.
- `valid_proof(transactions, last_hash, nonce, difficulty=MINING_DIFFICULTY)`: Checks if a hash value satisfies the mining conditions. This function is used within the proof_of_work function.
- `valid_chain(chain)`: checks if a bockchain is valid.

- `resolve_conflicts()`: Resolves conflicts between blockchain's nodes by replacing a chain with the longest one in the network.

8.4 Block design of chain

The structure of the block in the project can be seen from here.

```
{
  "hash":"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
  "ver":1,
  "prev_block":"0000000000000000000000000000000000000000000000000000000000000000",

  "mrkl_root":"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
  "time":1231006505,
  "bits":486604799,
  "nonce":2083236893,
  "n_tx":1,
  "size":285,
  "tx":[
    {
      "hash":"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
      "ver":1,
      "vin_sz":1,
      "vout_sz":1,
      "lock_time":0,
      "size":204,
      "in":[
        {
          "prev_out":{
        "hash":"0000000000000000000000000000000000000000000000000000000000000000",

            "n":4294967295
          },
          "coinbase":"04ffff001d0104455468652054696d65732030332f4a616e2f32303039204
368616e63656c6c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f7220
62616e6b73"
        }
      ],
      "out":[
        {
          "value":"50.00000000",
          "scriptPubKey":"04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0e
a1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f OP_CHECKS
IG"
        }
      ]
    }
  ],
  "mrkl_tree":[
    "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
  ]
}
```

The 2xSHA256 hash is taken of the block's header only (not the transactions in it): 80 bytes, consisting of the version, previous block hash, tx merkle root, time, bits and nonce.

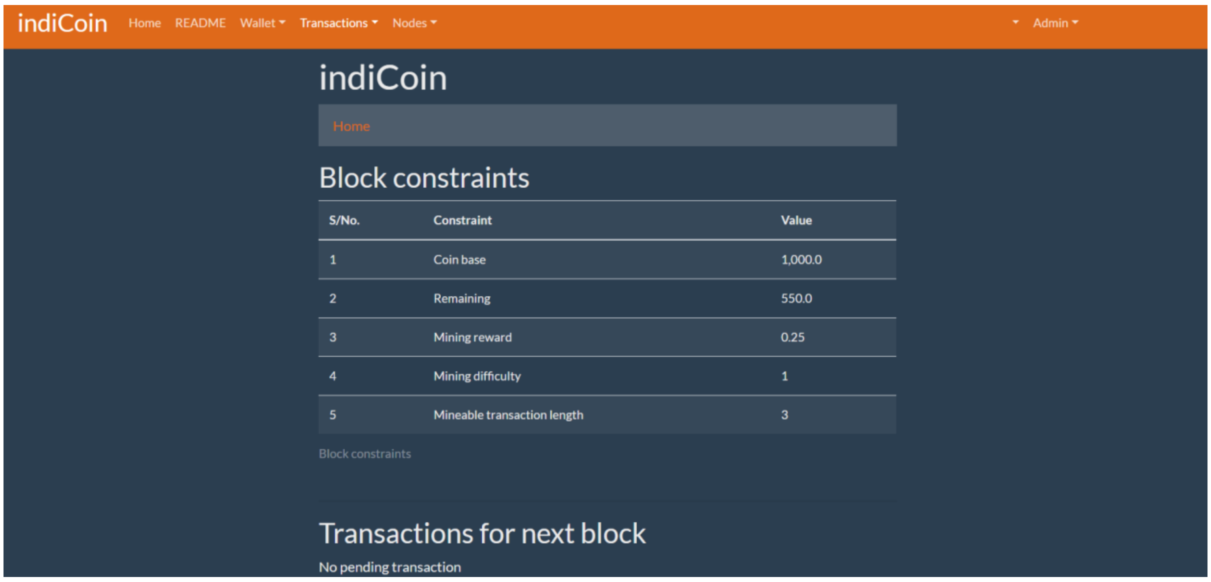
A python shell file

```
>>> import hashlib
>>> header_hex = ("01000000" +
  "81cd02ab7e569e8bcd9317e2fe99f2de44d49ab2b8851ba4a308000000000000" +
  "e320b6c2fffc8d750423db8b1eb942ae710e951ed797f7affc8892b0f1fc122b" +
  "c7f5d74d" +
  "f2b9441a" +
   "42a14695")
>>> header_bin = header_hex.decode('hex')
>>> hash = hashlib.sha256(hashlib.sha256(header_bin).digest()).digest()
>>> hash.encode('hex_codec')
'1dbd981fe6985776b644b173a4d0385ddc1aa2a829688d1e0000000000000000'
>>> hash[::-1].encode('hex_codec')
'00000000000000001e8d6829a8a21adc5d38d0a473b144b6765798e61f98bd1d'
```

A demo json file of the transaction in the chain after validation

```
{
   "ver":1,
   "inputs":[
      {
         "sequence":4294967294,
         "witness":"",
         "prev_out":{
            "spent":true,
            "spending_outpoints":[
               {
                  "tx_index":270018614,
                  "n":0
               }
            ],
            "tx_index":269632701,
            "type":0,
            "addr":"1GLctvTi81GDYZF5F6nif2MdbxUnAGHATZ",
            "value":4920000,
            "n":5,
            "script":"76a914a83fc0fb6edecb5289ef3e333520e4235bdf8a8788ac"
         },
         "script":"473044022036be6403aeb4e0e6fd54720b328d9d81bea32fb79684da0288743668fb5ef3ee02202023a71ef7217061fb9b4f35a05143de71447032e5a35b39c3d14b3210bad10b0121032725846bb7bc2e47b7b5a50670d77c8268f4d7f3243bdcf1b22174a67faaf528"
      }
   ],
   "weight":900,
   "block_height":477230,
   "relayed_by":"178.79.179.49",
   "out":[
      {
         "spent":true,
         "spending_outpoints":[
            {
               "tx_index":270020069,
               "n":0
            }
         ],
         "tx_index":270018614,
         "type":0,
         "addr":"1AG24pctoCpEMfSvEKfUZqaGYnz8ey8BfF",
         "value":3744000,
            "n":0,
            "script":"76a914659042e01e864e2f29641ea3a213c51a956d33c788ac"
         },
         {
            "spent":true,
            "spending_outpoints":[
               {
                  "tx_index":270879638,
                  "n":456
               }
            ],
            "tx_index":270018614,
            "type":0,
            "addr":"18Gj3unkApi17yh24JF6WwhN635SfABFMj",
            "value":1018920,
            "n":1,
            "script":"76a9144fc238bcda3f884ff6ce8d9feeb89b50dfd3da8888ac"
         }
      ],
      "lock_time":477228,
      "size":225,
      "double_spend":false,
      "block_index":1606071,
      "time":1500839662,
      "tx_index":270018614,
      "vin_sz":1,
      "hash":"b657e22827039461a9493ede7bdf55b01579254c1630b0bfc9185ec564fc05ab",
      "vout_sz":2
   }
```

9. Snapshots of input/output



indiCoin

Home

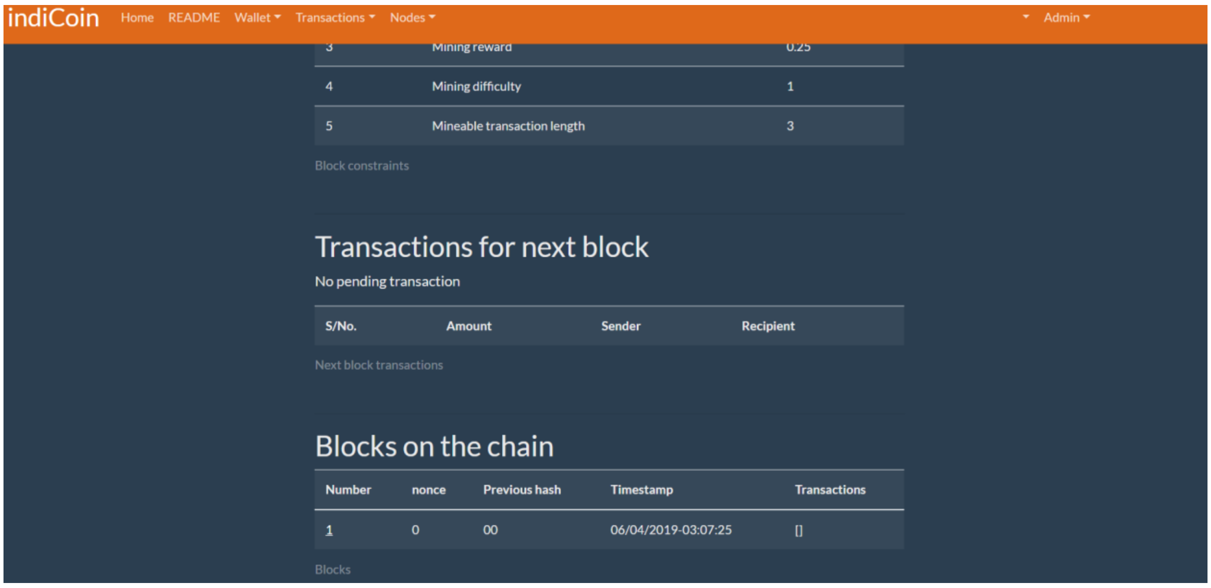
Block constraints

S/No.	Constraint	Value
1	Coin base	1,000.0
2	Remaining	550.0
3	Mining reward	0.25
4	Mining difficulty	1
5	Mineable transaction length	3

Block constraints

Transactions for next block

No pending transaction



3	Mining reward	0.25
4	Mining difficulty	1
5	Mineable transaction length	3

Block constraints

Transactions for next block

No pending transaction

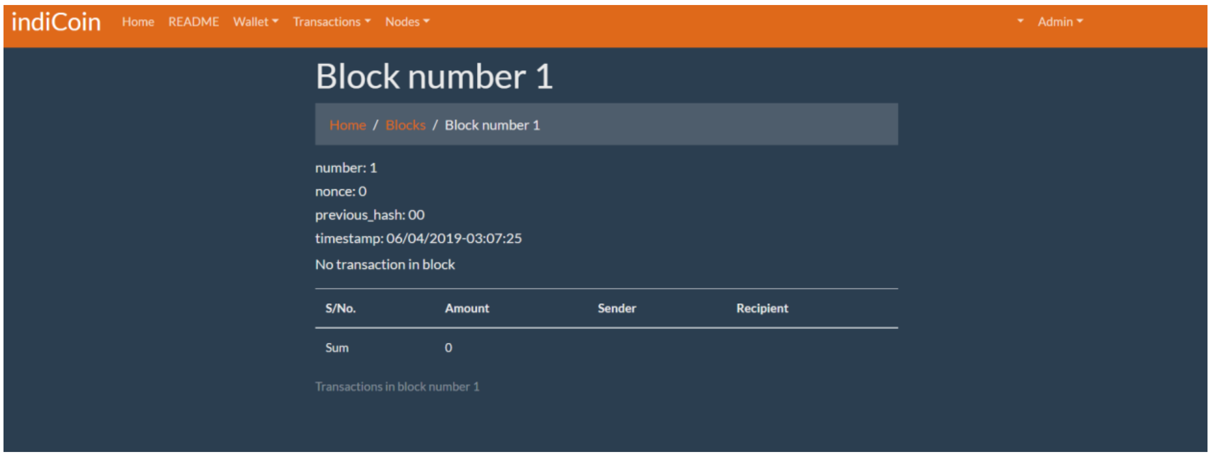
S/No.	Amount	Sender	Recipient

Next block transactions

Blocks on the chain

Number	nonce	Previous hash	Timestamp	Transactions
1	0	00	06/04/2019-03:07:25	[]

Blocks



Block number 1

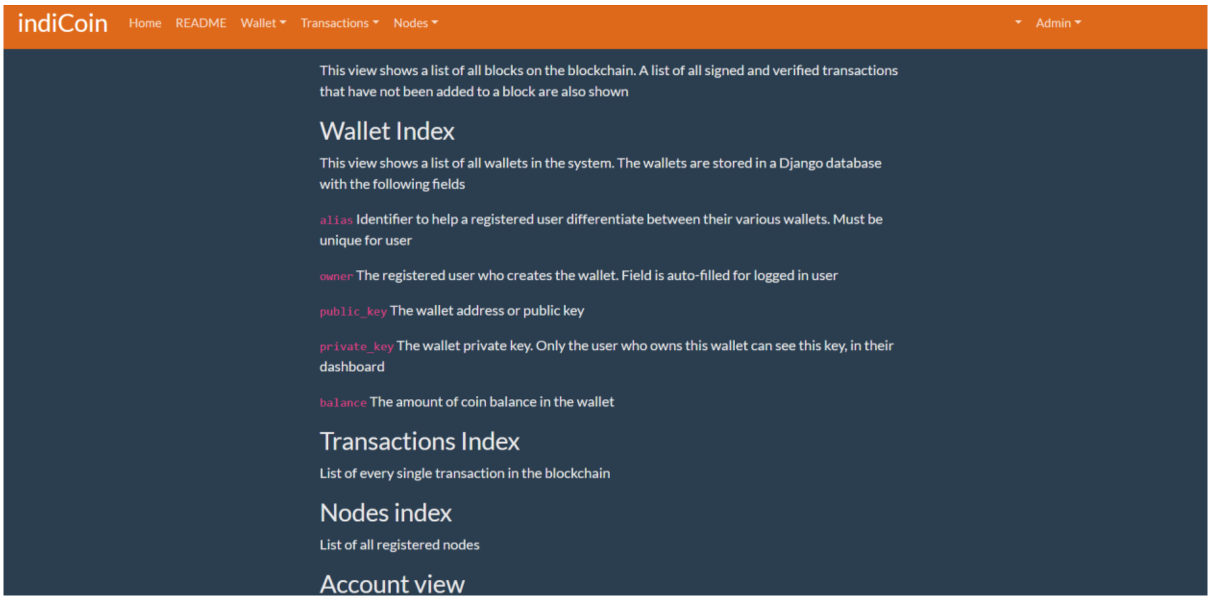
Home / Blocks / Block number 1

number: 1
nonce: 0
previous_hash: 00
timestamp: 06/04/2019-03:07:25
No transaction in block

S/No.	Amount	Sender	Recipient
Sum	0		

Transactions in block number 1

A genesis block



This view shows a list of all blocks on the blockchain. A list of all signed and verified transactions that have not been added to a block are also shown

Wallet Index

This view shows a list of all wallets in the system. The wallets are stored in a Django database with the following fields

`alias` Identifier to help a registered user differentiate between their various wallets. Must be unique for user

`owner` The registered user who creates the wallet. Field is auto-filled for logged in user

`public_key` The wallet address or public key

`private_key` The wallet private key. Only the user who owns this wallet can see this key, in their dashboard

`balance` The amount of coin balance in the wallet

Transactions Index

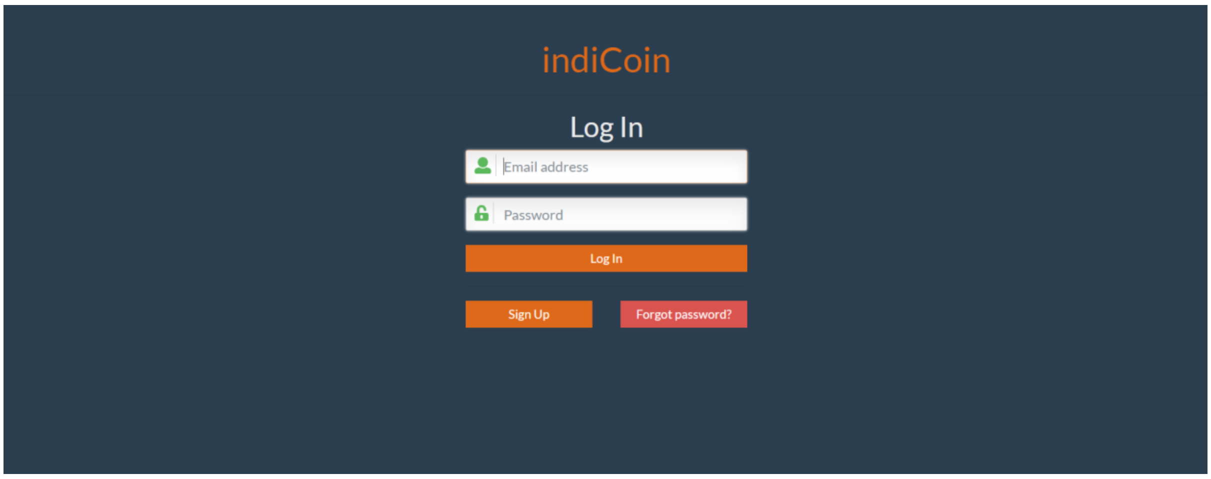
List of every single transaction in the blockchain

Nodes index

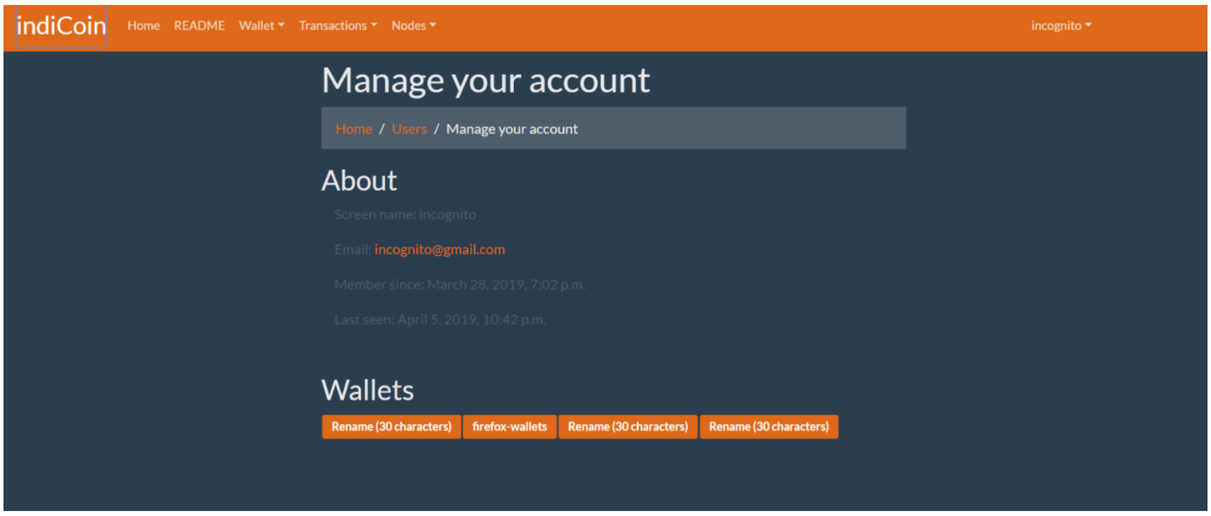
List of all registered nodes

Account view

ReadMe.md Getting started



Login Panel



Manage your account

Home / Users / Manage your account

About

Screen name: incognito

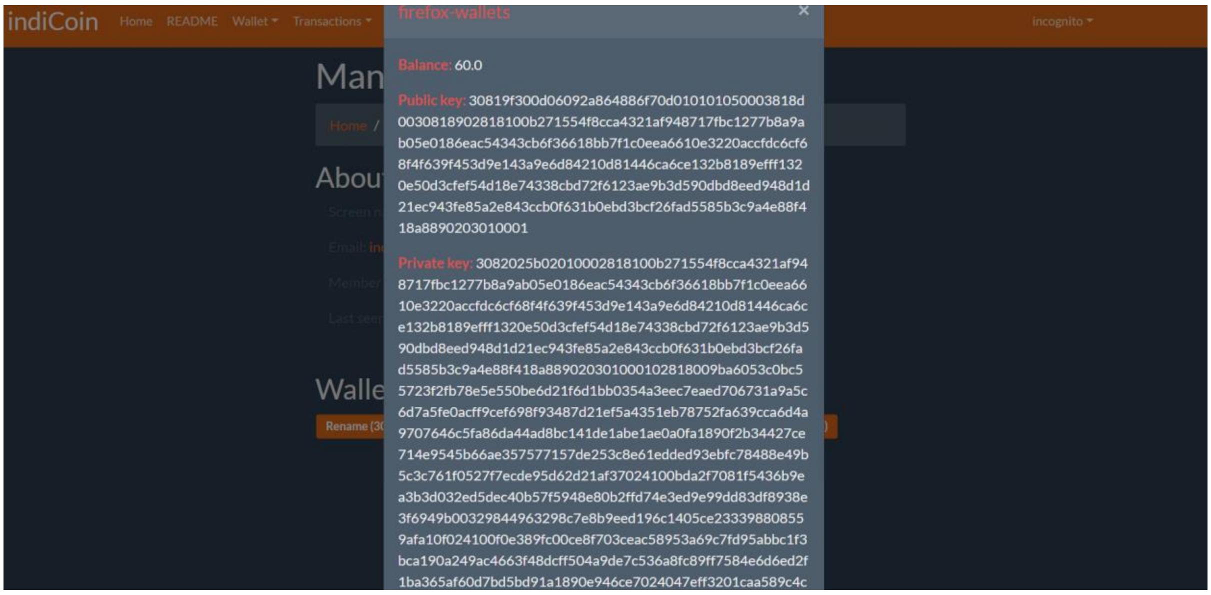
Email: incognito@gmail.com

Member since: March 28, 2019, 7:02 p.m.

Last seen: April 5, 2019, 10:42 p.m.

Wallets

Account Management



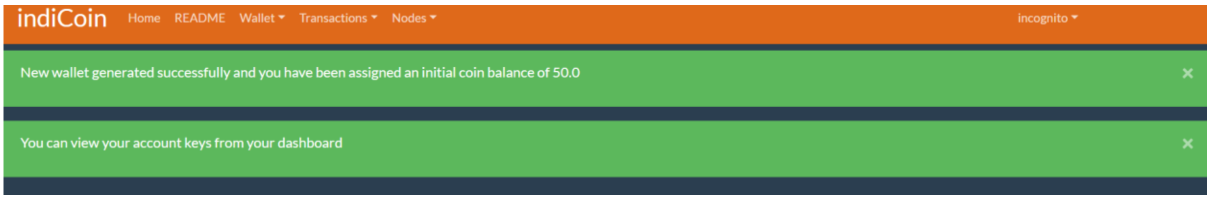
firefox-wallets

Balance: 60.0

Public key: 30819f300d06092a864886f70d010101050003818d0030818902818100b271554f8cca4321af948717fbc1277b8a9ab05e0186eac54343cb6f36618bb7f1c0eea6610e3220accfdc6cf68f4f639f453d9e143a9e6d84210d81446ca6ce132b8189efff1320e50d3cfef54d18e74338cbd72f6123ae9b3d590dbd8eed948d1d21ec943fe85a2e843ccb0f631b0ebd3bcf26fad5585b3c9a4e88f418a8890203010001

Private key: 3082025b02010002818100b271554f8cca4321af948717fbc1277b8a9ab05e0186eac54343cb6f36618bb7f1c0eea6610e3220accfdc6cf68f4f639f453d9e143a9e6d84210d81446ca6ce132b8189efff1320e50d3cfef54d18e74338cbd72f6123ae9b3d590dbd8eed948d1d21ec943fe85a2e843ccb0f631b0ebd3bcf26fad5585b3c9a4e88f418a88902030100010281809ba6053c0bc55723f2fb78e5e550be6d21f6d1bb0354a3eec7eaed706731a9a5c6d7a5fe0acff9cef698f93487d21ef5a4351eb78752fa639cca6d4a9707646c5fa86da44ad8bc141de1abe1ae0a0fa1890f2b34427ce714e9545b66ae357577157de253c8e61edded93ebfc78488e49b5c3c761f0527f7ecde95d62d21af37024100bda2f7081f5436b9ea3b3d032ed5dec40b57f5948e80b2ffd74e3ed9e99dd83df8938e3f6949b00329844963298c7e8b9eed196c1405ce233398808559afa10f024100f0e389fc00ce8f703ceac58953a69c7fd95abbc1f3bca190a249ac4663f48dcff504a9de7c536a8fc89ff7584e6d6ed2f1ba365af60d7bd5bd91a1890e946ce7024047eff3201caa589c4c

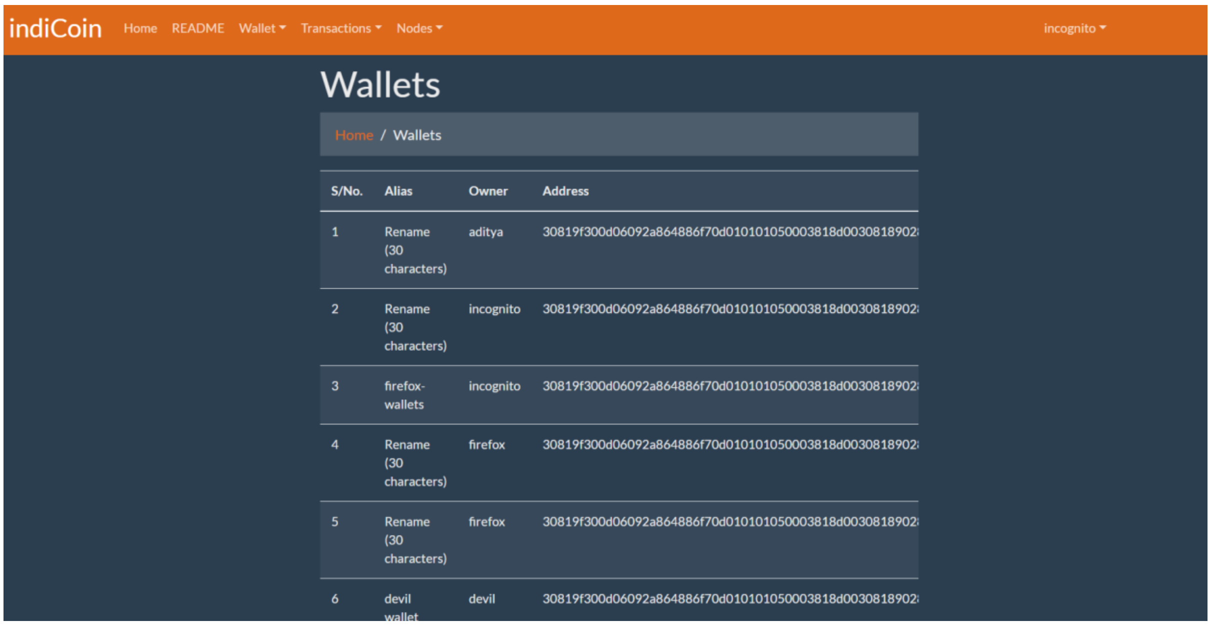
Wallet Credentials



New wallet generated successfully and you have been assigned an initial coin balance of 50.0

You can view your account keys from your dashboard

New wallet generation-auto assigned 50 IndiCoins

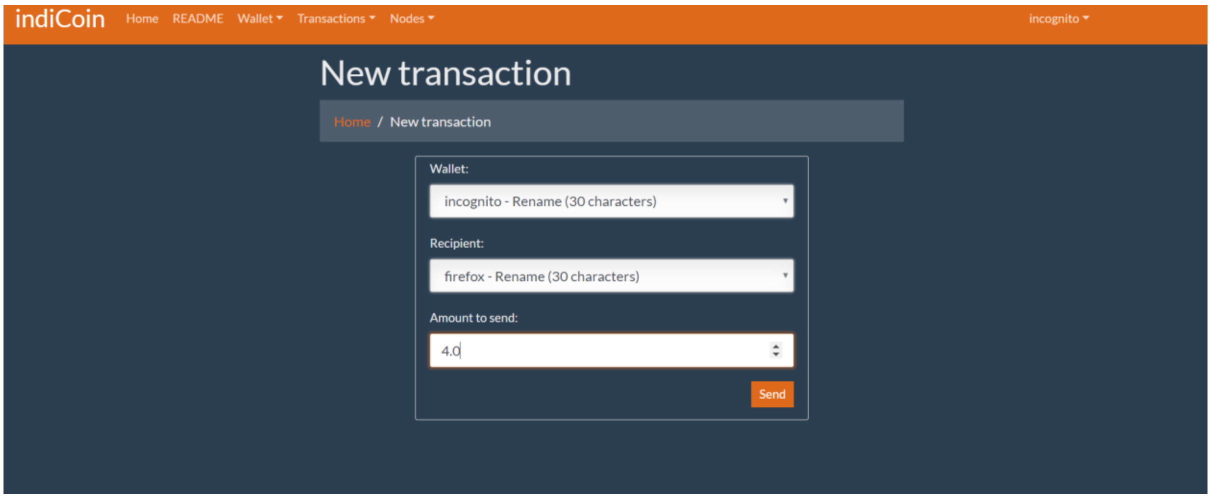


Wallets

Home / Wallets

S/No.	Alias	Owner	Address
1	Rename (30 characters)	aditya	30819f300d06092a864886f70d010101050003818d0030818902
2	Rename (30 characters)	incognito	30819f300d06092a864886f70d010101050003818d0030818902
3	firefox-wallets	incognito	30819f300d06092a864886f70d010101050003818d0030818902
4	Rename (30 characters)	firefox	30819f300d06092a864886f70d010101050003818d0030818902
5	Rename (30 characters)	firefox	30819f300d06092a864886f70d010101050003818d0030818902
6	devil wallet	devil	30819f300d06092a864886f70d010101050003818d0030818902

Wallets around the globe participating in the chain

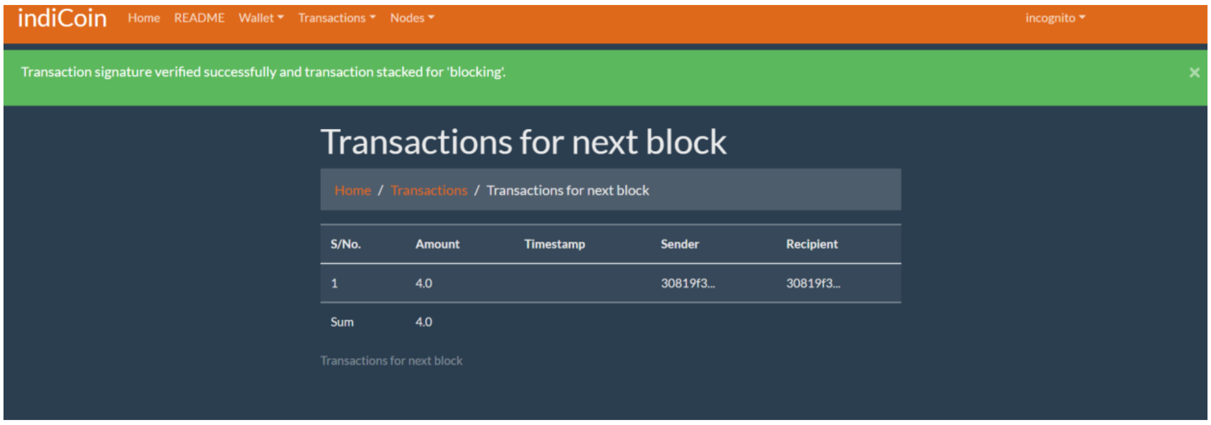


New transaction

Home / New transaction

Wallet:	incognito - Rename (30 characters)
Recipient:	firefox - Rename (30 characters)
Amount to send:	4.0

Initiating a transaction



Transaction signature verified successfully and transaction stacked for 'blocking'.

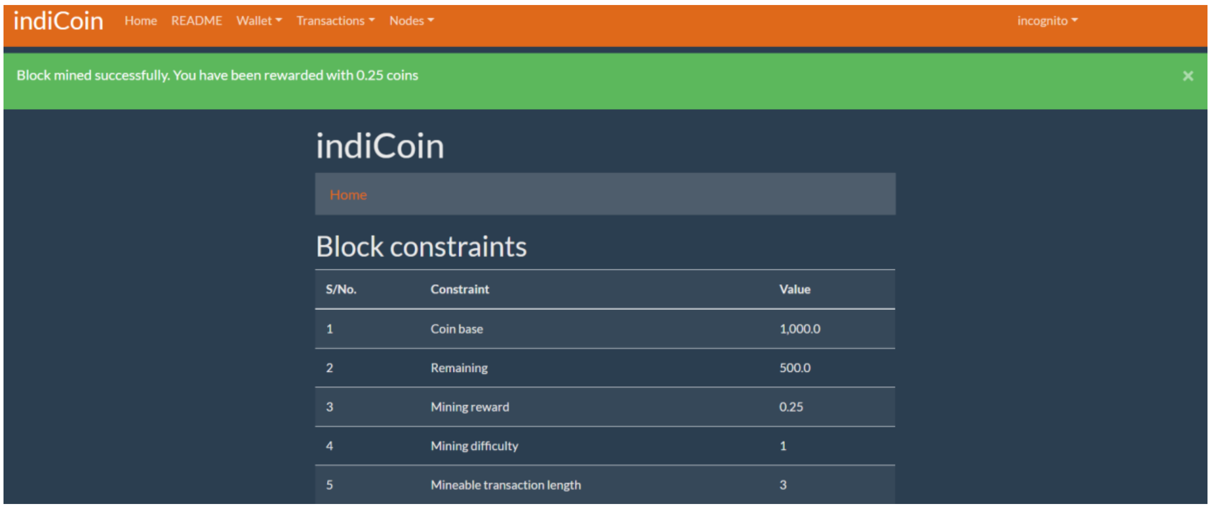
Transactions for next block

Home / Transactions / Transactions for next block

S/No.	Amount	Timestamp	Sender	Recipient
1	4.0		30819f3...	30819f3...
Sum	4.0			

Transactions for next block

On successful transaction – Receipt generated



Block mined successfully. You have been rewarded with 0.25 coins

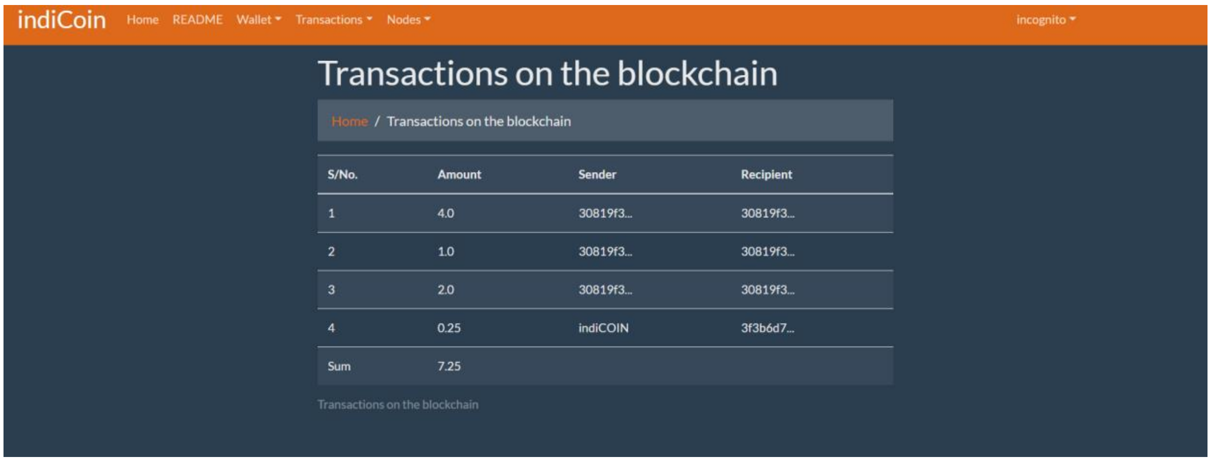
indiCoin

Home

Block constraints

S/No.	Constraint	Value
1	Coin base	1,000.0
2	Remaining	500.0
3	Mining reward	0.25
4	Mining difficulty	1
5	Mineable transaction length	3

Mining Report



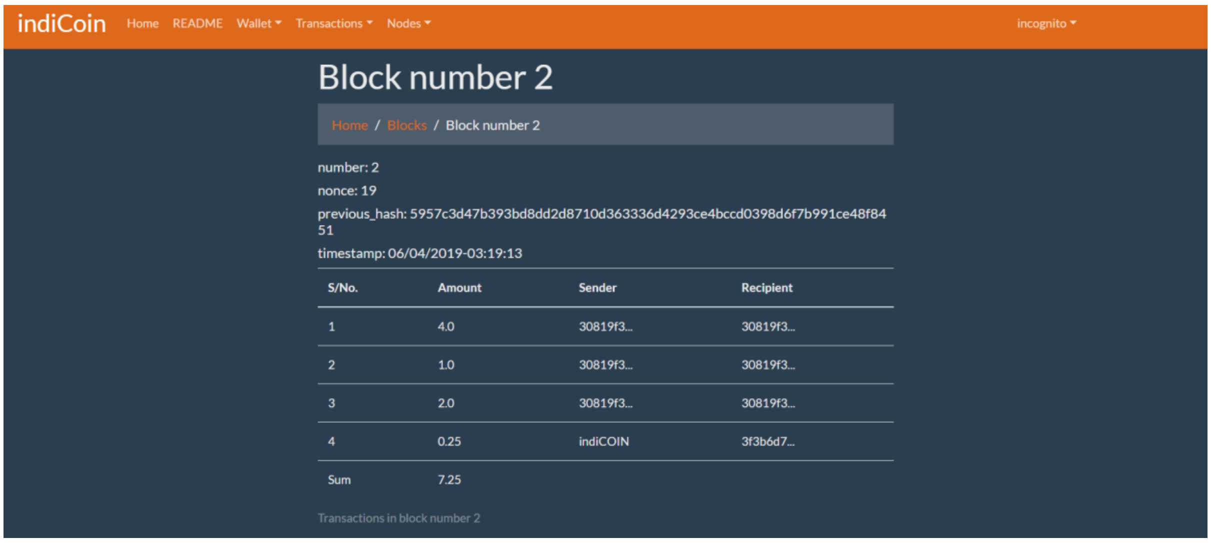
Transactions on the blockchain

Home / Transactions on the blockchain

S/No.	Amount	Sender	Recipient
1	4.0	30819f3...	30819f3...
2	1.0	30819f3...	30819f3...
3	2.0	30819f3...	30819f3...
4	0.25	indiCOIN	3f3b6d7...
Sum	7.25		

Transactions on the blockchain

Block appending to the chain



Block number 2

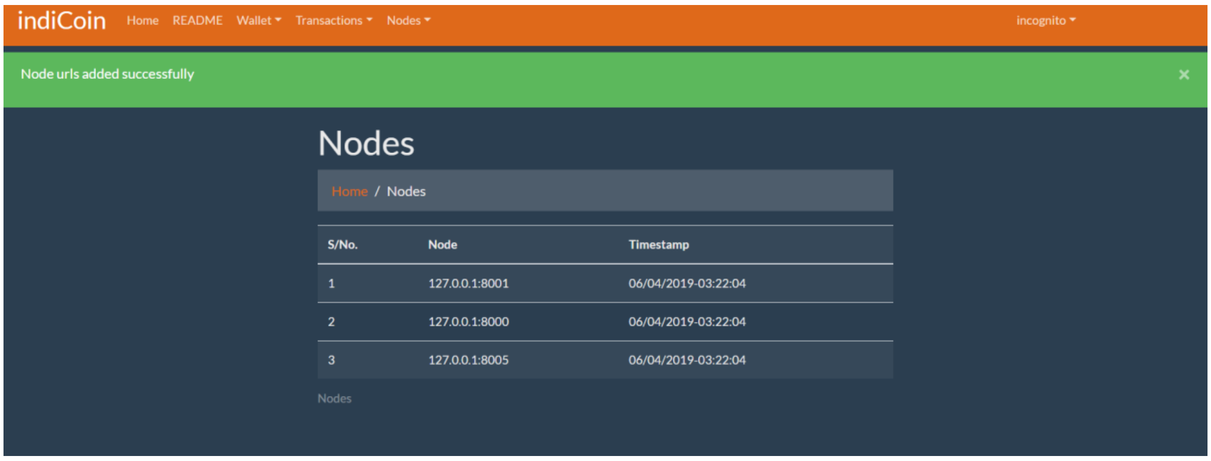
Home / Blocks / Block number 2

number: 2
nonce: 19
previous_hash: 5957c3d47b393bd8dd2d8710d363336d4293ce4bccd0398d6f7b991ce48f84 51
timestamp: 06/04/2019-03:19:13

S/No.	Amount	Sender	Recipient
1	4.0	30819f3...	30819f3...
2	1.0	30819f3...	30819f3...
3	2.0	30819f3...	30819f3...
4	0.25	indiCOIN	3f3b6d7...
Sum	7.25		

Transactions in block number 2

Visibility of blocks to the miners around the globe.



Node urls added successfully

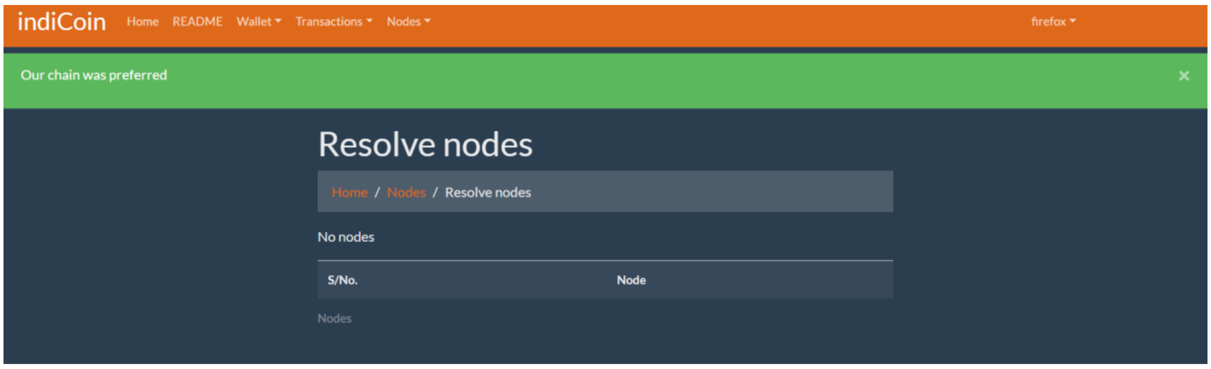
Nodes

Home / Nodes

S/No.	Node	Timestamp
1	127.0.0.1:8001	06/04/2019-03:22:04
2	127.0.0.1:8000	06/04/2019-03:22:04
3	127.0.0.1:8005	06/04/2019-03:22:04

Nodes

Adding Nodes for the network



Our chain was preferred

Resolve nodes

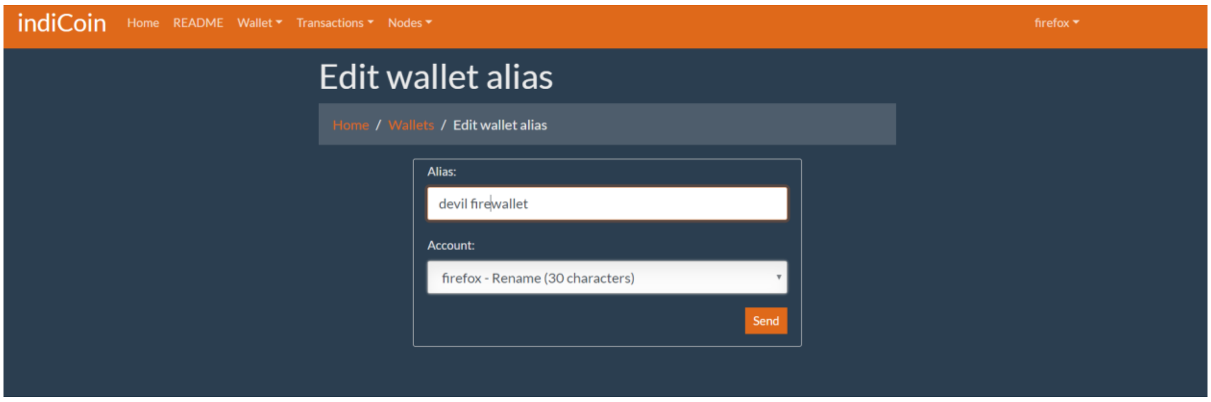
Home / Nodes / Resolve nodes

No nodes

S/No.	Node

Nodes

Preferred chain is the longest



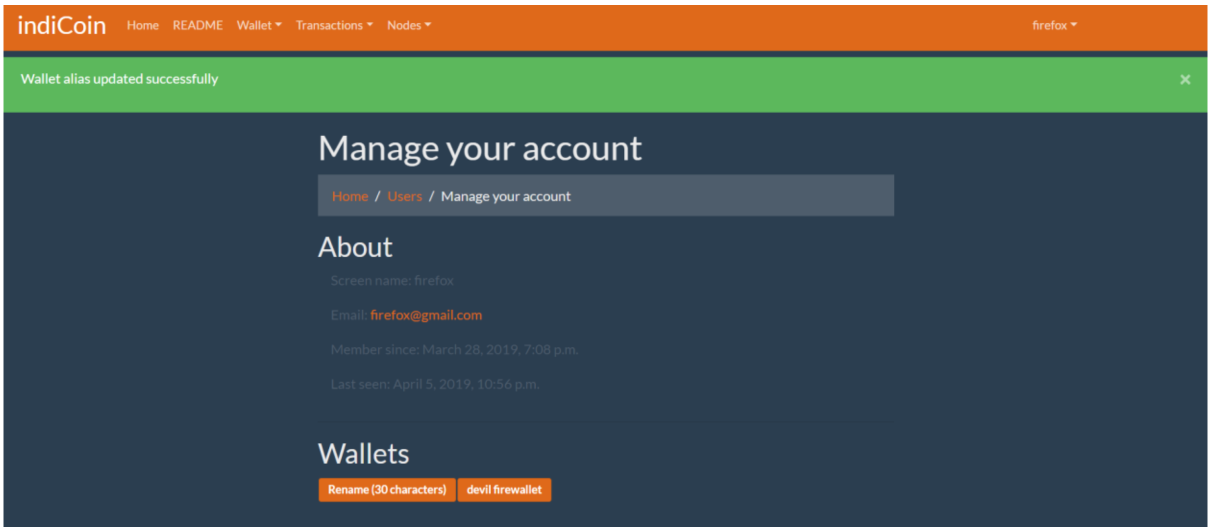
Edit wallet alias

Home / Wallets / Edit wallet alias

Alias: devil firewallet

Account: firefox - Rename (30 characters)

Editing Wallet alias



Wallet alias updated successfully

Manage your account

Home / Users / Manage your account

About

Screen name: firefox

Email: firefox@gmail.com

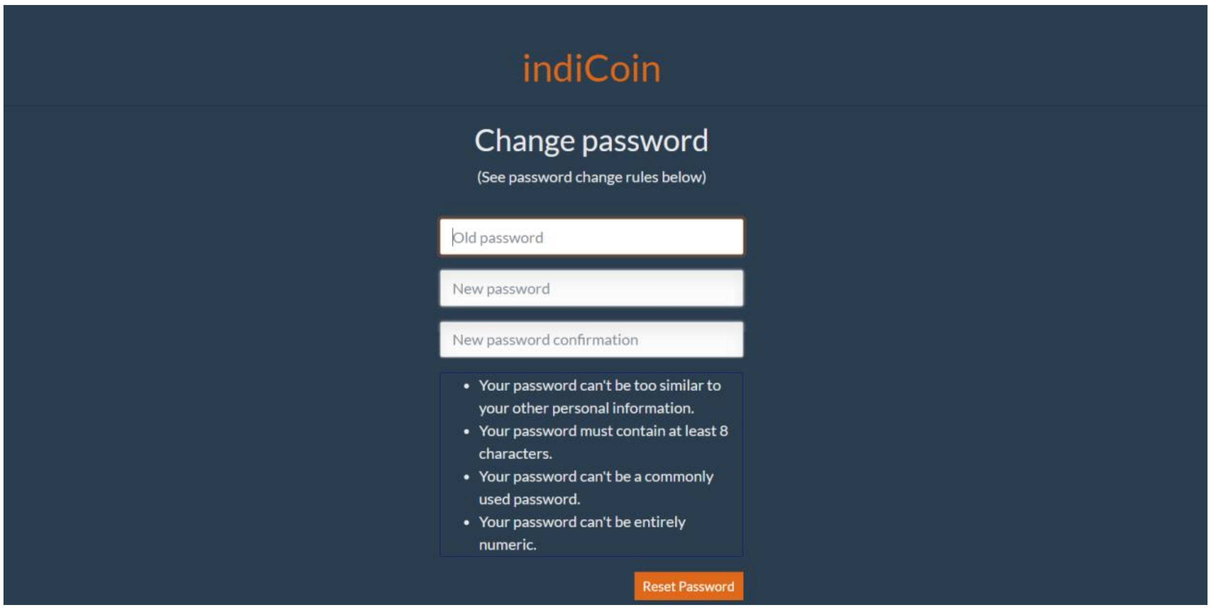
Member since: March 28, 2019, 7:08 p.m.

Last seen: April 5, 2019, 10:56 p.m.

Wallets

Rename (30 characters) devil firewallet

Successful update


indiCoin

Change password

(See password change rules below)

Old password

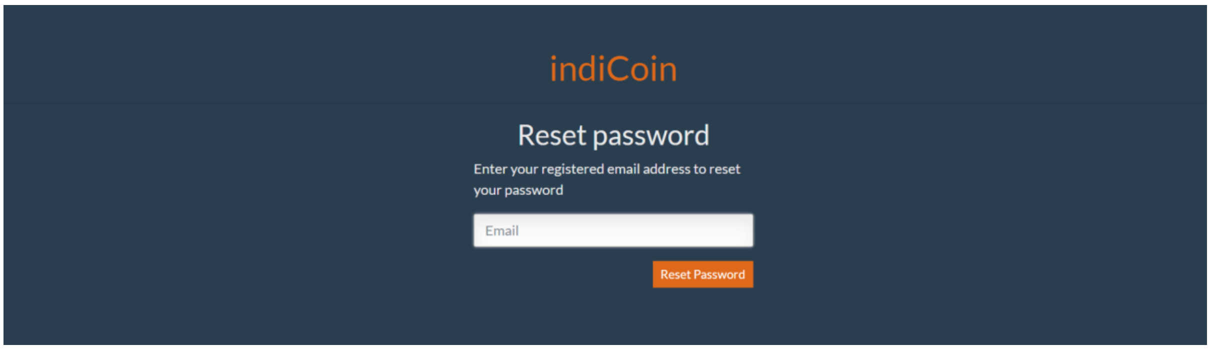
New password

New password confirmation

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Reset Password

Account Profile recovery UI


indiCoin

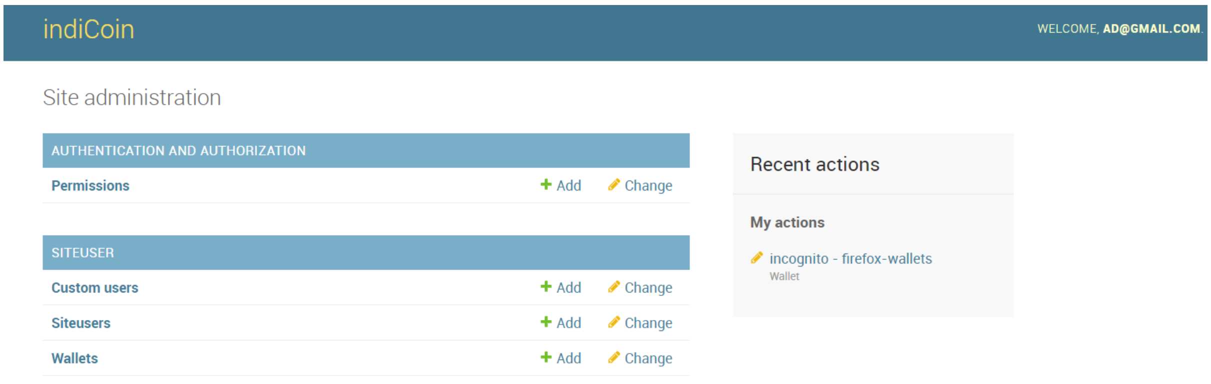
Reset password

Enter your registered email address to reset your password

Email

Reset Password

Reset Password field


indiCoin WELCOME, AD@GMAIL.COM.

Site administration

AUTHENTICATION AND AUTHORIZATION		
Permissions	+ Add	Change

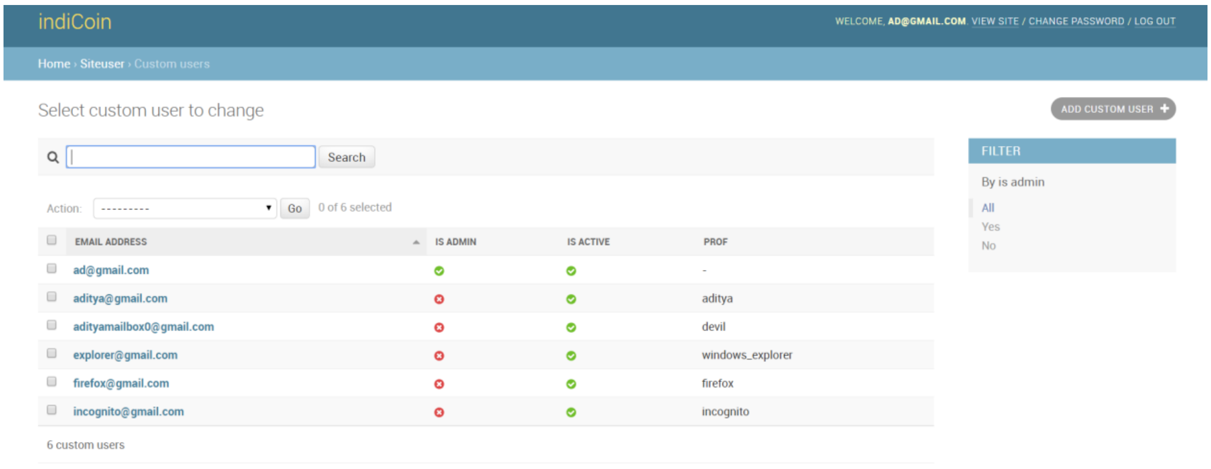
SITEUSER		
Custom users	+ Add	Change
Siteusers	+ Add	Change
Wallets	+ Add	Change

Recent actions

My actions

incognito - firefox-wallets
Wallet

Site Administration


indiCoin
WELCOME, AD@GMAIL.COM. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Siteuser › Custom users
ADD CUSTOM USER
Search
FILTER
By is admin
All
Yes
No

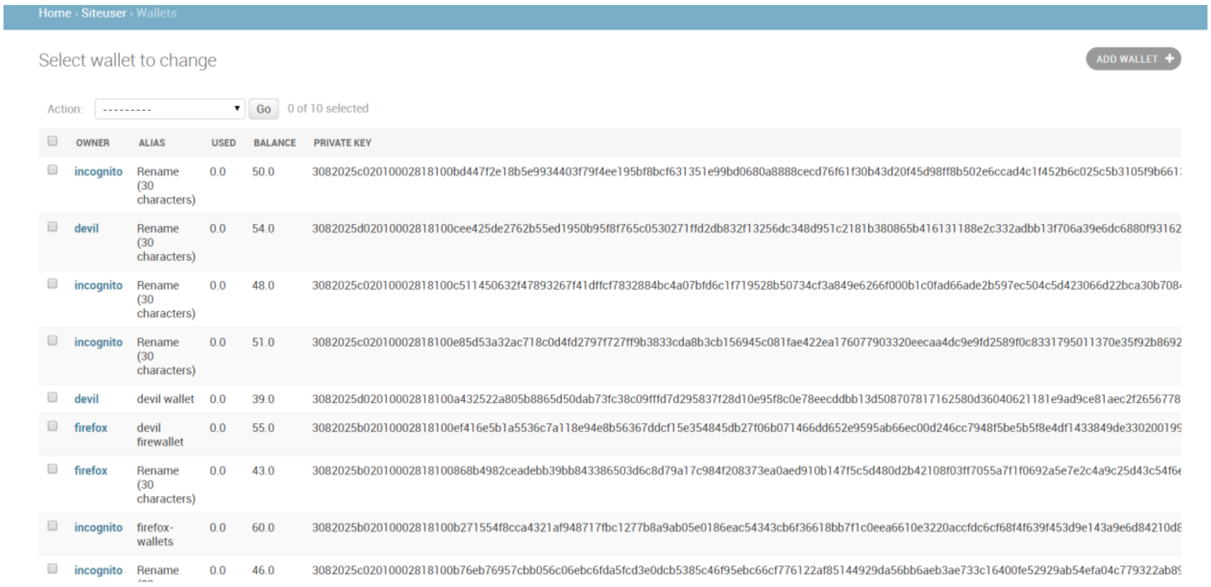
Select custom user to change

Action: --------- Go 0 of 6 selected

EMAIL ADDRESS	IS ADMIN	IS ACTIVE	PROF
ad@gmail.com	✓	✓	-
aditya@gmail.com	✗	✓	aditya
adityamailbox0@gmail.com	✗	✓	devil
explorer@gmail.com	✗	✓	windows_explorer
firefox@gmail.com	✗	✓	firefox
incognito@gmail.com	✗	✓	incognito

6 custom users

User management for organization

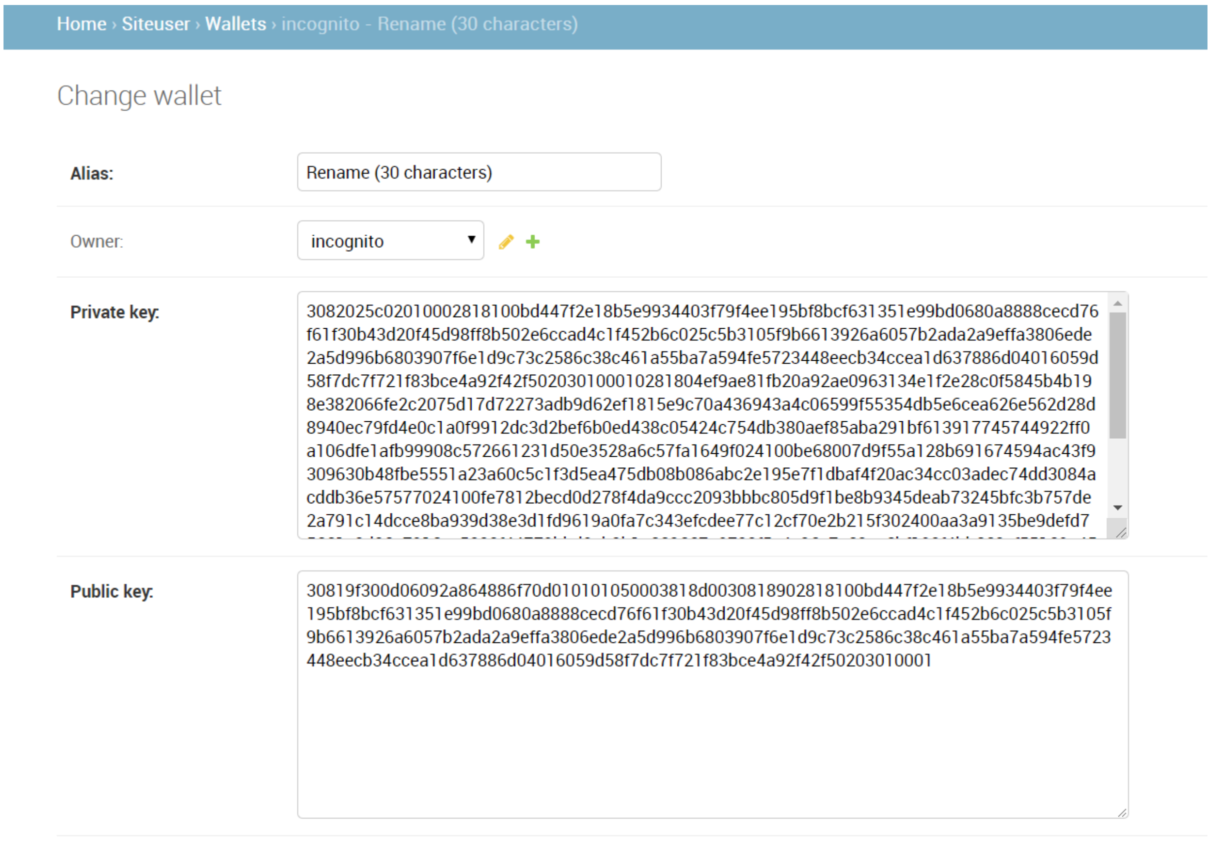


Select wallet to change

Action: --------- Go 0 of 10 selected

OWNER	ALIAS	USED	BALANCE	PRIVATE KEY
incognito	Rename (30 characters)	0.0	50.0	3082025c02010002818100bd447f2e18b5e9934403f79f4ee195bf8bcf631351e99bd0680a8888cecd76f61f30b43d20f45d98ff8b502e6ccad4c1f452b6c025c5b3105f9b661
devil	Rename (30 characters)	0.0	54.0	3082025d02010002818100cee425de2762b55ed1950b95f8f765c0530271ffd2db832f13256dc348d951c2181b380865b416131188e2c332adbb13f706a39e6dc6880f93162
incognito	Rename (30 characters)	0.0	48.0	3082025c02010002818100c511450632f47893267f41dffcf7832884bc4a07bfd6c1f719528b50734cf3a849e6266f000b1c0fad66ade2b597ec504c5d423066d22bca30b708
incognito	Rename (30 characters)	0.0	51.0	3082025c02010002818100e85d53a32ac718c0d4fd2797f727ff9b3833cda8b3cb156945c081fae422ea176077903320eecaa4dc9e9fd2589f0c8331795011370e35f92b8692
devil	devil wallet	0.0	39.0	3082025d02010002818100a432522a805b8865d50dab73fc38c09fffd7d295837f28d10e95f8c0e78eecddbb13d508707817162580d36040621181e9ad9ce81aec2f2656778
firefox	devil firewallet	0.0	55.0	3082025b02010002818100ef416e5b1a5536c7a118e94e8b56367ddcf15e354845db27f06b071466dd652e9595ab66ec00d246cc7948f5be5b5f8e4df1433849de33020019
firefox	Rename (30 characters)	0.0	43.0	3082025b02010002818100868b4982ceadebb39bb843386503d6c8d79a17c984f208373ea0aed910b147f5c5d480d2b42108f03ff7055a7f1f0692a5e7e2c4a9c25d43c54f6
incognito	firefox-wallets	0.0	60.0	3082025b02010002818100b271554f8cca4321af948717fbc1277b8a9ab05e0186eac54343cb6f36618bb7f1c0eea6610e3220accfdc6cf68f4f639f453d9e143a9e6d84210d
incognito	Rename	0.0	46.0	3082025c02010002818100b76eb76957cbb056c06ebc6fda5fcd3e0dcb5385c46f95ebc66cf776122af85144929da56bb6aeb3ae733c16400fe52929ab54efa04c779322ab8

Wallet management


Home › Siteuser › Wallets › incognito - Rename (30 characters)

Change wallet

Alias: Rename (30 characters)

Owner: incognito

Private key:

```
3082025c02010002818100bd447f2e18b5e9934403f79f4ee195bf8bcf631351e99bd0680a8888cecd76
f61f30b43d20f45d98ff8b502e6ccad4c1f452b6c025c5b3105f9b6613926a6057b2ada2a9effa3806ede
2a5d996b6803907f6e1d9c73c2586c38c461a55ba7a594fe5723448eecb34ccea1d637886d04016059d
58f7dc7f721f83bce4a92f42f50203010001028180 4ef9ae81fb20a92ae0963134e1f2e28c0f5845b4b19
8e382066fe2c2075d17d72273adb9d62ef1815e9c70a436943a4c06599f55354db5e6cea626e562d28d
8940ec79fd4e0c1a0f9912dc3d2bef6b0ed438c05424c754db380aef85aba291bf613917745744922ff0
a106dfe1afb99908c572661231d50e3528a6c57fa1649f024100be68007d9f55a128b691674594ac43f9
309630b48fbe5551a23a60c5c1f3d5ea475db08b086abc2e195e7f1dbaf4f20ac34cc03adec74dd3084a
cddb36e57577024100fe7812becd0d278f4da9ccc2093bbbc805d9f1be8b9345deab73245bfc3b757de
2a791c14dcce8ba939d38e3d1fd9619a0fa7c343efcdee77c12cf70e2b215f302400aa3a9135be9defd7
```

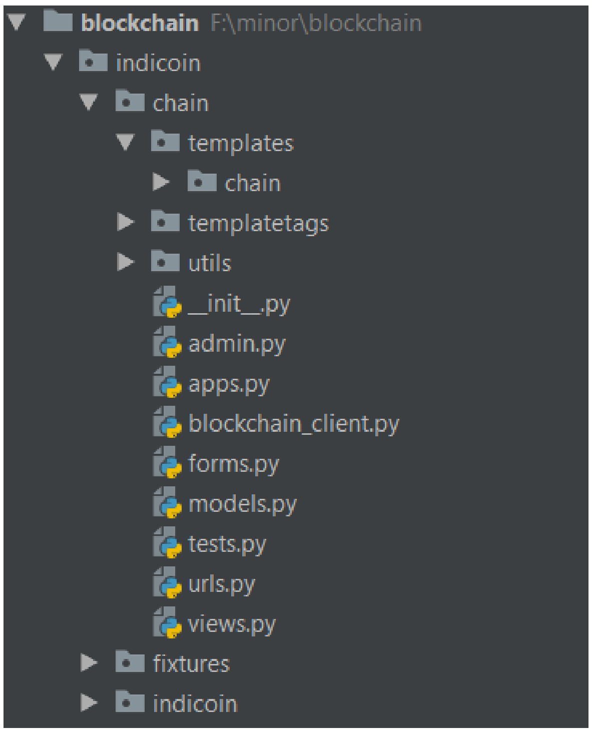
Public key:

```
30819f300d06092a864886f70d010101050003818d0030818902818100bd447f2e18b5e9934403f79f4ee
195bf8bcf631351e99bd0680a8888cecd76f61f30b43d20f45d98ff8b502e6ccad4c1f452b6c025c5b3105f
9b6613926a6057b2ada2a9effa3806ede2a5d996b6803907f6e1d9c73c2586c38c461a55ba7a594fe5723
448eecb34ccea1d637886d04016059d58f7dc7f721f83bce4a92f42f50203010001
```

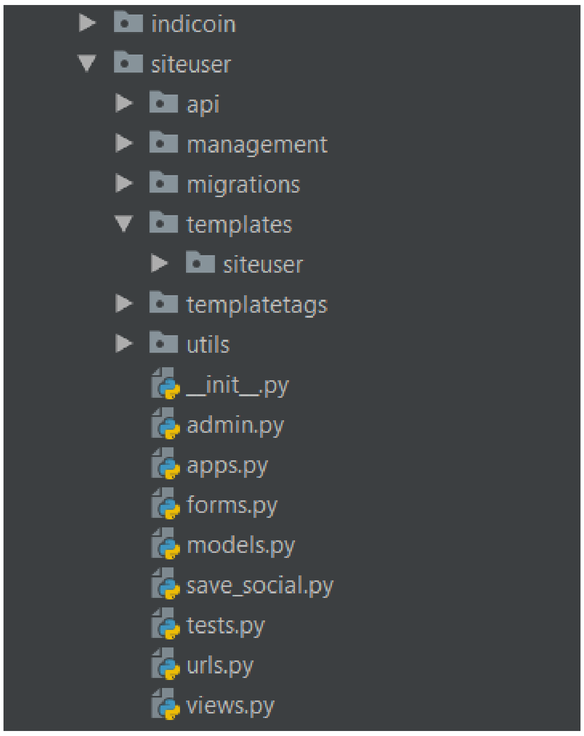
Wallet fields

10. Coding

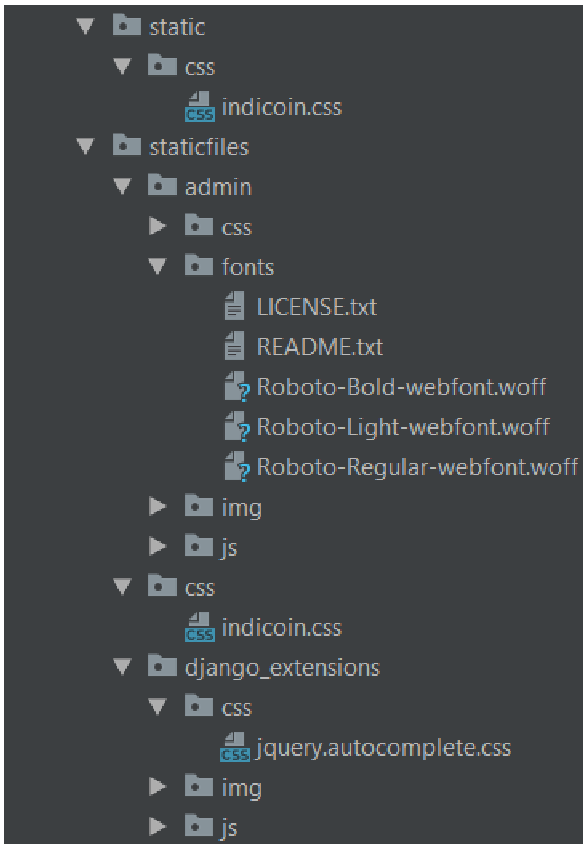
10.1 Project Hierarchy



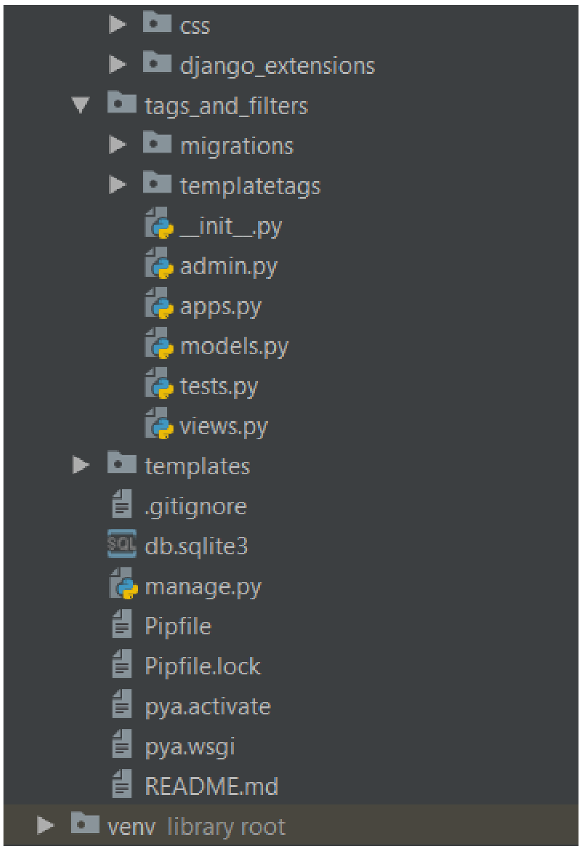
```
▼ blockchain  F:\minor\blockchain
  ▼ indicoin
    ▼ chain
      ▼ templates
        ▶ chain
      ▶ templatetags
      ▶ utils
        __init__.py
        admin.py
        apps.py
        blockchain_client.py
        forms.py
        models.py
        tests.py
        urls.py
        views.py
    ▶ fixtures
    ▶ indicoin
```



```
    ▶ indicoin
    ▼ siteuser
      ▶ api
      ▶ management
      ▶ migrations
      ▼ templates
        ▶ siteuser
      ▶ templatetags
      ▶ utils
        __init__.py
        admin.py
        apps.py
        forms.py
        models.py
        save_social.py
        tests.py
        urls.py
        views.py
```



```
    ▼ static
      ▼ css
          indicoin.css
    ▼ staticfiles
      ▼ admin
        ▶ css
        ▼ fonts
            LICENSE.txt
            README.txt
            Roboto-Bold-webfont.woff
            Roboto-Light-webfont.woff
            Roboto-Regular-webfont.woff
        ▶ img
        ▶ js
      ▼ css
          indicoin.css
      ▼ django_extensions
        ▼ css
            jquery.autocomplete.css
        ▶ img
        ▶ js
```



```
        ▶ css
        ▶ django_extensions
    ▼ tags_and_filters
        ▶ migrations
        ▶ templatetags
          __init__.py
          admin.py
          apps.py
          models.py
          tests.py
          views.py
    ▶ templates
      .gitignore
      db.sqlite3
      manage.py
      Pipfile
      Pipfile.lock
      pya.activate
      pya.wsgi
      README.md
  ▶ venv  library root
```

10.2 BlockChain Client Code base

```
import json
import time
import hashlib

from uuid import uuid4
from collections import OrderedDict
from urllib.parse import urlparse

import requests

import binascii
# import Crypto
# import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

MINING_DIFFICULTY = 1
MINING_SENDER = 'indiCOIN'
MINING_REWARD = 0.25
MINABLE_TRANSACTIONS = 3
COINBASE = 1000.00

class Transaction:
    def __init__(self, sender_address, sender_private_key, recipient_address, 
amount):
        self.sender_address = sender_address
        self.sender_private_key = sender_private_key
        self.recipient_address = recipient_address
        self.timestamp = time.strftime('%d/%m/%Y-%H:%M:%S')
        self.amount = amount

    def __getattr__(self, attr):
        return self.data[attr]

    def to_dict(self):
        return OrderedDict(
            {'sender_address': self.sender_address,
            'recipient_address': self.recipient_address,
            'amount' : self.amount,
            # 'timestamp' : self.timestamp
            })

    def sign_transaction(self):
        """Sign the transaction with sender's private key"""
        private_key = RSA.importKey(binascii.unhexlify(self.sender_private_key))
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf-8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')

class Blockchain:
    def __init__(self):
        self.transactions = []
        self.chain = []
        # self.nodes = set()
        self.nodes = OrderedDict()
        # Random number to use as node id
        self.node_id = str(uuid4()).replace('-', '')
        # Genesis block
        self.forge_block_and_add_to_chain(0, '00')

    def register_node(self, node_url):
        """
        Add new node to the nodes dictionary
        Parameters
        -----------
        node_url : str
            Node url e.g http://127.0.0.1
        """

        parsed_url = urlparse(node_url)
        if parsed_url.netloc:
            self.nodes[parsed_url.netloc] = time.strftime('%d/%m/%Y-%H:%M:%S')
        elif parsed_url.path:
            self.nodes[parsed_url.path] = time.strftime('%d/%m/%Y-%H:%M:%S')
        else:
            raise ValueError("Invalid url")

    def verify_transaction_signature(self, sender_address, signature, transaction):
        """
        Verifies that provided signature is actually signed by sender_address
         (public key)

        Parameters
        -----------
        transaction : Transaction

        Returns
        ---------
        bool
            True if transaction is valid (signature is signed by sender_address).
         False otherwise.
        """
        public_key = RSA.importKey(binascii.unhexlify(sender_address))
        verifier = PKCS1_v1_5.new(public_key)
        h = SHA.new(str(transaction).encode('utf-8'))
        return verifier.verify(h, binascii.unhexlify(signature))

    def reward_miner(self, miner_address):
        """Reward a miner

        Parameters
        -----------
        miner_address : str
            Address of miner

        Returns
        --------
        int :
            Current length of chain
        """
        transaction = OrderedDict({'sender_address': MINING_SENDER,
                                   'recipient_address': miner_address,
                                   'amount': MINING_REWARD})
        self.transactions.append(transaction)

        return len(self.chain) + 1

    def add_transaction_to_current_array(self, sender_address, recipient_address,
amount, signature):
        """
        Add transaction to the transaction array if it can be verified
        """
        transaction = OrderedDict({'sender_address': sender_address,
                                   'recipient_address': recipient_address,
                                   'amount': amount})
        verify = self.verify_transaction_signature(sender_address, signature,
transaction)
        if verify:
            self.transactions.append(transaction)
            return len(self.chain) + 1
        else:
            return False

    def forge_block_and_add_to_chain(self, nonce, previous_hash):
        """
        Add a block of transactions to the chain
        """
        block = OrderedDict()
        block['number'] = len(self.chain) + 1
        block['nonce'] = nonce
        block['previous_hash'] = previous_hash
        block['timestamp'] = time.strftime('%d/%m/%Y-%H:%M:%S')
        block['transactions'] = self.transactions

        # Reset the current list of transactions
        self.transactions = []

        self.chain.append(block)
        return block

    @staticmethod
    def hash(block):
        """
        Create SHA-256 hash of a block
        """

        # convert block from OrderedDict to regular dictionary
        block_string = json.dumps(block).encode()
        return hashlib.sha256(block_string).hexdigest()

    def proof_of_work(self):
        """
        Proof of work algorithm
        """
        last_hash = self.hash(self.last_block())
        nonce = 0
        while self.valid_proof(self.transactions, last_hash, nonce) is False:
            nonce += 1
        return nonce

    def valid_proof(self, transactions, last_hash, nonce,
difficulty=MINING_DIFFICULTY):
        """
        Check whether hash satisfies mining conditions.

        Parameters
        -----------
        transactions : list
            Current transactions
        last_hash : str
            Last hash value
        difficulty : int
            Represents how hard our mining algorithm should be

        Returns
        --------
        bool :
            True if mining condition is satisfied. False otherwise
        """
        guess = (str(transactions) + str(last_hash) + str(nonce)).encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:difficulty] == '0'*difficulty

    def valid_chain(self, chain):
        """
        Check whether blockchain is valid

        Parameters
        -----------
        chain : Blockchain
            A blockchain instance

        Returns
        ---------
        bool :
            Return True if valid. False otherwise.
        """
        last_block = chain[0]
        current_index = 1

        while current_index < len(chain):
            block = chain[current_index]

            # Check that the hash of the block is correct
            if block['previous_hash'] != self.hash(last_block):
                return False

            # Check that the Proof of Work is correct
            # Delete the reward transaction
            transactions = block['transactions'][:-1]

            # Need to make sure that the dictionary is ordered. Otherwise we'll get
a different hash
            transaction_elements = ['sender_address', 'recipient_address', 'value']
            transactions = [OrderedDict((k, transaction[k]) for k in
transaction_elements) for transaction in transactions]

            if not self.valid_proof(transactions, block['previous_hash'],
block['nonce'], MINING_DIFFICULTY):
                return False

            last_block = block
            current_index += 1

        return True

    def resolve_conflicts(self):
        """
        Resolve conflicts between blockchain's nodes
        by replacing our chain with the longest one in the network.
        """
        neighbours = [key for key, value in self.nodes.items()]
        new_chain = None
        # We're only looking for chains longer than ours
        max_length = len(self.chain)

        # Grab and verify the chains from all the nodes in our network
        for node in neighbours:
            response = requests.get('http://' + node + '/chain')

            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']

                # Check if the length is longer and the chain is valid
                if length > max_length and self.valid_chain(chain):
                    max_length = length
                    new_chain = chain

        # Replace our chain if we discovered a new, valid chain longer than ours
        if new_chain:
            self.chain = new_chain
            return True
        return False

    def last_block(self):
        return self.chain[-1]
    def mineable(self):
        if len(self.transactions) >= MINABLE_TRANSACTIONS:
            return True
        return False
```

10.3 Settings.py for configuring the chain and integrating to the webapp

```
"""
Django settings for indicoin project.
https://docs.djangoproject.com/en/2.0/ref/settings/
"""

import os
import pygments.formatters
from django.urls import reverse_lazy

from django.contrib.messages import constants as messages

from decouple import config, Csv

from chain.blockchain_client import Blockchain

BLOCKCHAIN = Blockchain()

BASE_DIR =
os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

ROOT_URLCONF = 'indicoin.urls'
SECRET_KEY = 'fe8ykj1wrt&pjewu_61ys6$b7*d4hs1^-)j_n2v0a%4)_(ryg+'
#config('SECRET_KEY', cast=Csv())
ALLOWED_HOSTS = ['*'] #config('ALLOWED_HOSTS', cast=Csv())

WSGI_APPLICATION = 'indicoin.wsgi.application'
INTERNAL_IPS = ['*']#('127.0.0.1', 'localhost')

MESSAGE_LEVEL = 10  # DEBUG
MESSAGE_TAGS = {
    messages.DEBUG: 'alert-info',
    messages.INFO: 'alert-info',
    messages.SUCCESS: 'alert-success',
    messages.WARNING: 'alert-warning',
    messages.ERROR: 'alert-danger',
}

EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 25 #config('EMAIL_PORT')
EMAIL_HOST_USER = 'indicoin@outlook.com'
EMAIL_HOST_PASSWORD = '12345' #config('EMAIL_HOST_PASSWORD')
DEFAULT_FROM_EMAIL = 'indicoin@outlook.com'

LOGIN_URL = reverse_lazy('personnel:login')
LOGOUT_URL = reverse_lazy('personnel:logout')
LOGOUT_REDIRECT_URL = reverse_lazy('personnel:login')

SITE_ID = 1
PASSWORD_RESET_TIMEOUT_DAYS = 1

LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
LOGIN_URL = reverse_lazy('siteuser:login')
LOGOUT_URL = reverse_lazy('siteuser:logout')

# SHELL_PLUS_PRINT_SQL = True
```

```
SHELL_PLUS_PYGMENTS_FORMATTER = pygments.formatters.TerminalFormatter
SHELL_PLUS_PYGMENTS_FORMATTER_KWARGS = {}
IPYTHON_KERNEL_DISPLAY_NAME = "Django Shell-Plus"
SHELL_PLUS_POST_IMPORTS = [
    ('fixtures', '*'),
    ('fixtures'),]

PREREQ_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

PROJECT_APPS = [
    'chain',
    'siteuser',
    'tags_and_filters',
]

THIRD_PARTY_APPS = [
    'django_extensions',
]

INSTALLED_APPS = PREREQ_APPS +  PROJECT_APPS + THIRD_PARTY_APPS

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

# Password validation
# https://docs.djangoproject.com/en/2.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/2.0/topics/i18n/

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'Africa/Lagos'
USE_I18N = True
USE_L10N = True
USE_TZ = True
USE_THOUSAND_SEPARATOR = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.0/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static'),)

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'

AUTH_USER_MODEL = 'siteuser.CustomUser'

DEBUG = True
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

10.4 Integrating functions to respective URLs in views.py

```
from django.db.models import Sum
from django.shortcuts import render, redirect
from django.http import  JsonResponse
from django.conf import settings
from django.contrib import messages
from django.contrib.auth.decorators import login_required

import binascii
import Crypto.Random
# from Crypto.Hash import SHA
from Crypto.PublicKey import RSA

from siteuser.models import Wallet

from .blockchain_client import Transaction, COINBASE, MINING_DIFFICULTY,
MINING_REWARD, MINABLE_TRANSACTIONS
from .forms import NodeRegistrationForm, InitiateTransactionForm,
InitiateTransactionAuthUserForm

# Instantiate a blockchain
BLOCKCHAIN = settings.BLOCKCHAIN

def index(request):
    # get balance of coins in the system
    SUM_COINS =
Wallet.objects.aggregate(total_balance=Sum('balance'))['total_balance']
    # If no wallet instance has been created, the balance returns None
    if SUM_COINS == None:
        SUM_COINS = 0.00
    template = 'chain/index.html'
    context = {}
    context['pending_transactions'] = BLOCKCHAIN.transactions
    context['chain'] = BLOCKCHAIN.chain
    context['mining_difficulty'] = MINING_DIFFICULTY
    context['mining_reward'] = MINING_REWARD
    context['coin_base'] = COINBASE
    context['unassigned'] = COINBASE - SUM_COINS
    context['mineable'] = MINABLE_TRANSACTIONS
    return render(request, template, context)

def transactions_index(request):
    """View all transactions on the blockchain"""
    template = 'chain/transactions_index.html'
    context = {}
    context['transactions'] = [transaction for block in BLOCKCHAIN.chain for
transaction in block['transactions']]
    return render(request, template, context)

def transactions_destined_for_next_block(request):
    """View all transactions to be added to the next block"""
    template = 'chain/transactions_destined_for_next_block.html'
    context = {}
    context['transactions'] = BLOCKCHAIN.transactions
    return render(request, template, context)

@login_required
def transaction_auth_user(request):
    """
    Initiate a transaction and send it to a blockchain node
    """
    user = request.user
    template = 'chain/initiate_transaction.html'

    if request.method == 'POST':
        form = InitiateTransactionAuthUserForm(request.POST, user=user)
        if form.is_valid():
            data = form.cleaned_data
            wallet = data['wallet']
            recipient = data['recipient']
            amount_to_send = data['amount_to_send']

            sender_address = wallet.public_key
            sender_private_key = wallet.private_key
            recipient_address = recipient.public_key

            transaction_object = Transaction(sender_address, sender_private_key,
recipient_address, amount_to_send)
            transaction = transaction_object.to_dict()
            signature = transaction_object.sign_transaction()
            verify = BLOCKCHAIN.verify_transaction_signature(sender_address,
signature, transaction)
            if verify:
                BLOCKCHAIN.add_transaction_to_current_array(sender_address,
recipient_address, amount_to_send, signature)
                messages.success(request, "Transaction signature verified
successfully and transaction stacked for 'blocking'.")
            else:
                messages.error(request, "Transaction rejected")

            # update wallet balances
            wallet.balance -= amount_to_send
            wallet.save()
            recipient.balance += amount_to_send
            recipient.save()

            return redirect('blockchain:transactions_destined_for_next_block')
        else:
            return render(request, template, {'form' : form})
    return render(request, template, {'form' :
InitiateTransactionAuthUserForm(user=user)})

@login_required
def transaction_anon(request):
    """
    Initiate a transaction and send it to a blockchain node
    """
    template = 'chain/initiate_transaction.html'
    context = {}

    if request.method == 'POST':
        form = InitiateTransactionForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            sender_address = data['sender_address']
            sender_private_key = data['sender_private_key']
            recipient_address = data['recipient_address']
            amount_to_send = data['amount_to_send']

            transaction_object = Transaction(sender_address, sender_private_key,
recipient_address, amount_to_send)
            context['transaction'] = transaction_object.to_dict()
            context['signature'] = transaction_object.sign_transaction()
            return JsonResponse(context, status=201)
        else:
            return render(request, template, {'form' : form})
    return render(request, template, {'form' : InitiateTransactionForm()})

def block_detail(request, index):
    """View transactions in a block"""
    template = 'chain/block_detail.html'
    context = {}
    number = int(index)-1
    context['number'] = number + 1
    context['block_items'] = BLOCKCHAIN.chain[number]
    return render(request, template, context)

def mine(request):
    if BLOCKCHAIN.mineable() is False:
        messages.error(request, "At least {} transactions are needed to forge a
block.".format(MINABLE_TRANSACTIONS))
        return redirect('blockchain:index')

    # get next proof from POW algorithm
    last_block = BLOCKCHAIN.last_block()
    nonce = BLOCKCHAIN.proof_of_work()

    # reward for finding proof
    BLOCKCHAIN.reward_miner(BLOCKCHAIN.node_id)

    # forge new block and add to chain
    previous_hash = BLOCKCHAIN.hash(last_block)
    BLOCKCHAIN.forge_block_and_add_to_chain(nonce, previous_hash)
    return redirect('blockchain:index')

def register_nodes(request):
    """Register new nodes"""
    template = 'chain/node_register.html'

    if request.method == 'POST':
        form = NodeRegistrationForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            node_urls = data['node_urls'].split(',')

            for node_url in node_urls:
                try:
                    BLOCKCHAIN.register_node(node_url)
                    continue
                except ValueError:
                    messages.error(request, "Invalid node url:
{}".format(node_url))
                    return redirect('blockchain:node_index')
            messages.success(request, "Node urls added successfully")
            return redirect('blockchain:node_index')
        else:
            return render(request, template, {'form' : form})
    return render(request, template, {'form' : NodeRegistrationForm()})

def node_index(request):
    template = 'chain/node_index.html'
    context = {}
    context['nodes'] = BLOCKCHAIN.nodes
    return render(request, template, context)

def consensus(request):
    template = 'chain/nodes_resolve.html'
    context = {}
    replaced = BLOCKCHAIN.resolve_conflicts()
    if replaced:
        msg = 'Our chain was replaced'
        context['msg'] = msg
        context['chain'] = BLOCKCHAIN.chain
        messages.error(request, msg)
    else:
        msg = 'Our chain was preferred'
        context['msg'] = msg
        context['chain'] = BLOCKCHAIN.chain
        messages.success(request, msg)

    # return redirect('blockchain:node_index')
    return render(request, template, context)
```

10.5 API and Serializers

```
from rest_framework import serializers
from ..models import CustomUser, SiteUser, Role

class CustomUserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = CustomUser
        fields = ('email', 'is_active', 'last_login') # no email, for privacy
purpose
        # fields = ('email', 'is_active', 'last_login')

class SiteUserSerializer(serializers.HyperlinkedModelSerializer):
    url = serializers.CharField(source='get_absolute_url', read_only=True)
    roles = serializers.CharField(source='get_all_roles', read_only=True)
    user = CustomUserSerializer()

    class Meta:
        model = SiteUser
        fields = ('user', 'first_name', 'last_name', 'screen_name', 'location',
'roles', 'url', 'avatar')

class RolesSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Role
        fields = ('name', )
```

```
from rest_framework import viewsets
from ..models import CustomUser, SiteUser
from .import serializers

class SiteUserViewSet(viewsets.ModelViewSet):
    queryset = SiteUser.objects.all().order_by('created')
    serializer_class = serializers.SiteUserSerializer

class CustomUserViewSet(viewsets.ModelViewSet):
    queryset = CustomUser.objects.all()
    serializer_class = serializers.CustomUserSerializer

# urls.py

router = routers.DefaultRouter()
router.register(r'users', views.CustomUserViewSet)
router.register(r'members', views.SiteUserViewSet)

user_api_urls = [
    path('users/', views.CustomUserViewSet.as_view({'get': 'list'}),
name='customuser_api'),
    path('members/', views.SiteUserViewSet.as_view({'get': 'list'}),
name='siteuser_api'),
]
```

10.6 Input field and validation forms

```
"""Forms"""

from django import forms
from django.urls import reverse_lazy
from django.utils.translation import ugettext_lazy as _
from django.contrib.auth import get_user_model
from django.contrib.auth.hashers import check_password
from django.contrib.auth.forms import ReadOnlyPasswordHashField

from .models import SiteUser, Wallet

CustomUser = get_user_model()

class UserCreationForm(forms.ModelForm):
    """Custom UCF. Takes the standard
    variables of 'email', 'password1', 'password2'
    For creating instances of 'CustomUser'."""
    password1 = forms.CharField(widget=forms.PasswordInput(attrs={'class':'form-
control', 'type':'password'}))
    password2 = forms.CharField(widget=forms.PasswordInput(attrs={'class':'form-
control', 'type':'password'}))

    class Meta:
        model = CustomUser
        fields = ('email', )

    def clean_password2(self):
        password1 = self.cleaned_data.get('password1')
        password2 = self.cleaned_data.get('password2')
        if password1 and password2 and password1 != password2:
            raise forms.ValidationError("Passwords do not match")
        return password2

    def save(self, commit=True):
        user = super(UserCreationForm, self).save(commit=False)
        user.set_password(self.cleaned_data["password1"])
        if commit:
            user.save()
        return user

class UserChangeForm(forms.ModelForm):
    password = ReadOnlyPasswordHashField

    class Meta:
        model = CustomUser
        fields = ["email", "password", "is_active", "is_admin"]

    def clean_password(self):
        return self.initial["password"]

class SiteUserMixin(forms.ModelForm):
    class Meta:
        model = SiteUser
        fields = ("screen_name", )
        widgets = {
            "screen_name" : forms.TextInput(attrs={'class':'form-control',
"placeholder" : "Display name"}),
        }

class SiteUserRegistrationForm(forms.Form):
    screen_name = forms.CharField(
        required=True,
        widget=forms.TextInput(attrs={'class':'form-control', "placeholder" :
"Screen name"}))
```

```
    email = forms.EmailField(
        required=True,
        widget=forms.TextInput(attrs={'class':'form-control', "placeholder" :
"Email address"}))

    password1 = forms.CharField(
        required=True,
        widget=forms.PasswordInput(attrs={'class':'form-control',
'type':'password', "placeholder" : "Enter password"}))

    password2 = forms.CharField(
        required=True,
        widget=forms.PasswordInput(attrs={'class':'form-control',
'type':'password', "placeholder" : "Verify password"}))

    def clean(self):
        data = self.cleaned_data
        email = data.get("email", None).strip()
        password1 = data.get('password1', None).strip()
        password2 = data.get('password2', None).strip()
        screen_name = data.get("screen_name", None).strip()

        User = get_user_model()
        if User.objects.filter(email=email).exists():
            self.add_error('email', 'Email already registered.')

        if password1 and password2 and password1 != password2:
            self.add_error('password1', "Passwords do not match")

        if SiteUser.objects.filter(screen_name=screen_name).exists():
            self.add_error('screen_name', 'Display name already taken.')

class SiteUserEditForm(forms.ModelForm):
    class Meta:
        model = SiteUser
        fields = ["screen_name",]

        widgets = {
            "screen_name" : forms.TextInput(attrs={'class' : 'form-control',
"placeholder" : "Screen name"}),
        }

class PassWordGetterForm(forms.Form):
    password = forms.CharField(
        required=True,
        widget=forms.PasswordInput(attrs={'class':'form-control',
'type':'password', "placeholder" : "Enter password"}))
    def __init__(self, *args, **kwargs):
        self.user = kwargs.pop('user')
        super(PassWordGetterForm, self).__init__(*args, **kwargs)

    def clean(self):
        password = self.cleaned_data['password']
        if check_password(password, self.user.password) is False:
            self.add_error('password', 'You entered a wrong password')

class EditAliasForm(forms.ModelForm):
    class Meta:
        model = Wallet
        fields = ('alias', )

        widgets = {'alias' : forms.TextInput(attrs={'class' : 'form-control',
'placeholder' : 'Account identifier'})}

    def __init__(self, *args, **kwargs):
        self.user = kwargs.pop("user", None)
        super(EditAliasForm, self).__init__(*args, **kwargs)
        self.fields['account'].queryset =
Wallet.objects.filter(owner__user=self.user)

    account = forms.ModelChoiceField(
        queryset=Wallet.objects.all(),
        required=True,
        widget=forms.Select(attrs={'class' : 'form-control'}))

    def clean(self):
        alias = self.cleaned_data['alias']
        aliases = Wallet.objects.filter(owner__user=self.user).values_list('alias',
flat=True)
        if alias in aliases:
            self.add_error('alias', "You already have a wallet name
{}".format(alias))
```

```
# Chain Forms

from django import forms
from siteuser.models import Wallet

class AcceptTransactionForm(forms.Form):
    sender_address = forms.CharField(widget=forms.TextInput(attrs={'class' : 'form-
control'}))
    recipient_address = forms.CharField(widget=forms.TextInput(attrs={'class' :
'form-control'}))
    signature = forms.CharField(widget=forms.TextInput(attrs={'class' : 'form-
control'}))
    amount_to_receive = forms.FloatField(widget=forms.NumberInput(attrs={'class' :
'form-control', 'step': 0.25}))

class NodeRegistrationForm(forms.Form):
    node_urls = forms.CharField(widget=forms.TextInput(attrs={'class' : 'form-
control', 'placeholder' : 'Comma-separated list e.g http://127.0.0.1:5000'}))

class InitiateTransactionForm(forms.Form):
    sender_address = forms.CharField(widget=forms.TextInput(attrs={'class' : 'form-
control', 'placeholder' : 'Sender address'}))
    sender_private_key = forms.CharField(widget=forms.TextInput(attrs={'class' :
'form-control', 'placeholder' : 'Sender private key'}))
    recipient_address = forms.CharField(widget=forms.TextInput(attrs={'class' :
'form-control', 'placeholder' : 'Recipient address'}))
    amount_to_send = forms.FloatField(widget=forms.NumberInput(attrs={'class' :
'form-control', 'step': 0.25, 'placeholder' : 'Amount to send: steps of 0.25'}))

class InitiateTransactionAuthUserForm(forms.Form):
    def __init__(self, *args, **kwargs):
        user = kwargs.pop("user", None)
        super(InitiateTransactionAuthUserForm, self).__init__(*args, **kwargs)
        self.fields['wallet'].queryset = Wallet.objects.filter(owner__user=user)

    def clean(self):
        data = self.cleaned_data
        wallet = data['wallet']
        recipient = data['recipient']
        amount_to_send = data['amount_to_send']
        if wallet == recipient:
            self.add_error('recipient', 'Wallet cannot transfer to itself')
        if wallet.balance < amount_to_send:
            self.add_error('wallet', 'Low wallet balance:
{}'.format(wallet.balance))

    wallet = forms.ModelChoiceField(
        queryset=Wallet.objects.all(),
        required=True,
        widget=forms.Select(attrs={'class' : 'form-control'}))
    recipient = forms.ModelChoiceField(
        queryset=Wallet.objects.all(),
        required=True,
        widget=forms.Select(attrs={'class' : 'form-control'}))
    amount_to_send = forms.FloatField(widget=forms.NumberInput(attrs={'class' :
'form-control', 'step': 0.25}))
```

10.7 Redirecting and node resolvers

```
"""Views"""

from django.db.models import Sum
from django.views import generic
from django.shortcuts import render, reverse, redirect
from django.contrib import messages
from django.contrib.auth import login
from django.contrib.messages.views import SuccessMessageMixin
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.decorators import login_required
from django.contrib.auth import get_user_model

from .models import SiteUser, Wallet
from chain.blockchain_client import Transaction, COINBASE, MINING_DIFFICULTY,
MINING_REWARD, MINABLE_TRANSACTIONS

from .forms import PassWordGetterForm, SiteUserRegistrationForm, SiteUserEditForm,
EditAliasForm
from django.contrib.auth.decorators import login_required

import binascii
import Crypto.Random
# from Crypto.Hash import SHA
from Crypto.PublicKey import RSA

CustomUser = get_user_model()

def new_siteuser(request):
    template = "siteuser/new.html"
    if request.method == 'POST':
        form = SiteUserRegistrationForm(request.POST)
        if form.is_valid():
            form = form.cleaned_data
            email = form['email']
            screen_name = form['screen_name']
            password1 = form['password1']

            user = CustomUser(email=email)
            user.set_password(password1)
            user.is_active=True
            user.save()

            new_user = SiteUser(user=user, screen_name=screen_name)
            new_user.save()
            login(request, user)
            return redirect('blockchain:index')
        else:
            return render(request, template, {'form' : form})
    return render(request, template, {'form' : SiteUserRegistrationForm()})

class SiteUserEdit(LoginRequiredMixin, SuccessMessageMixin, generic.UpdateView):
    model = SiteUser
    form_class = SiteUserEditForm
    template_name = 'siteuser/edit.html'
    success_message = "Profile updated successfully."
    def get_object(self):
        user = self.request.user
        return SiteUser.objects.get(pk=user.pk)

    def get_success_url(self):
        return reverse('siteuser:account_management')

def delete_account(request):
    template = 'siteuser/delete_account.html'
    user = request.user
    siteuser = user.siteuser
    if request.method == 'POST':
        form = PassWordGetterForm(request.POST, user=user)
        if form.is_valid():
            siteuser.delete()
            user.delete()
            msg = "Your account has been permanently deleted"
            messages.success(request, msg)
            return redirect('/')
        else:
            # return render(request, template, {'form' : form })
            msg = "You entered a wrong password"
            messages.error(request, msg)
            return redirect('/')
    return render(request, template, {'form' : PassWordGetterForm(user=user) })

@login_required
def account_management(request):
    template = "siteuser/account_management.html"
    context = {}
    context['siteuser'] = request.user.siteuser
    return render(request, template, context)

def generate_wallet(request):
    """Generate a new wallet"""
    random_gen = Crypto.Random.new().read
    pr_key = RSA.generate(1024, random_gen)
    pub_key = pr_key.publickey()

    private_key = binascii.hexlify(pr_key.exportKey(format='DER')).decode('ascii')
    public_key = binascii.hexlify(pub_key.exportKey(format='DER')).decode('ascii')

    # get balance of coins in the system
    SUM_COINS =
Wallet.objects.aggregate(total_balance=Sum('balance'))['total_balance']
    # If no wallet instance has been created, the balance returns None
    if SUM_COINS == None:
        SUM_COINS = 0.00
    if SUM_COINS < COINBASE:
        balance = 50.00
    else:
        balance = 0.00
    messages.success(request, "New wallet generated successfully and you have been
assigned an initial coin balance of {}".format(balance))
    if request.user.is_authenticated is False:
        messages.error(request, "Copy your public and private keys and save them in
a safe place at once as they cannot be recovered if lost")
        messages.success(request, "PUBLIC KEY: {}\n\nPRIVATE KEY:
{}".format(public_key, private_key))
        return redirect('blockchain:index')
    else:
        messages.success(request, "You can view your account keys from your
dashboard")
        # save credentials to database
        Wallet.objects.create(alias="Rename (30 characters)",
            owner=request.user.siteuser, private_key=private_key, balance=balance,
public_key=public_key)
        return redirect('siteuser:account_management')
```

```
def edit_alias(request):
    user = request.user
    template = 'siteuser/wallet_edit_alias.html'

    if request.method == 'POST':
        form = EditAliasForm(request.POST, user=user)
        if form.is_valid():
            data = form.cleaned_data
            alias = data['alias']
            account = data['account']

            account.alias = alias
            account.save(update_fields=['alias'])
            messages.success(request, "Wallet alias updated successfully")
            return redirect('siteuser:account_management')
        else:
            return render(request, template, {'form' : form})
    return render(request, template, {'form' : EditAliasForm(user=user)})

def wallet_index(request):
    template = 'siteuser/wallet_index.html'
    context = {}
    context['wallets'] = Wallet.objects.all()
    return render(request, template, context)
```

10.8 Models and database fields (Class based model)

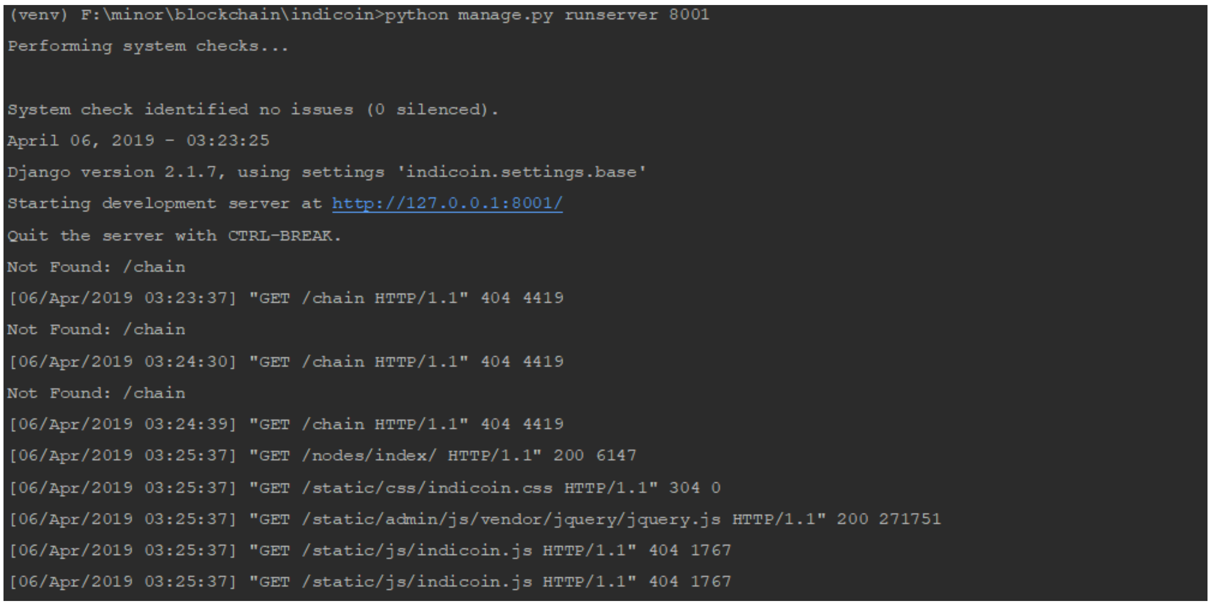
```
"""Models"""

from django.db import models
from django.urls import reverse
from django.conf import settings
# from django.contrib.auth.models import Group
from django.contrib.auth.models import BaseUserManager, AbstractBaseUser

from .utils import TimeStampedModel
from .utils import AutoSlugField

class CustomUserManager(BaseUserManager):
    def create_user(self, email, password=None):
        if not email:
            raise ValueError("You must provide an email address")
        user = self.model(email=self.normalize_email(email))
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, password):
        user = self.create_user(email, password=password)
        user.is_admin = True
        user.is_active = True
        user.save(using=self._db)
        return user

class CustomUser(AbstractBaseUser):
    email = models.EmailField(max_length=255, unique=True, verbose_name='email 
address')
    is_active = models.BooleanField(default=False) # activate by email
    is_admin = models.BooleanField(default=False)

    objects = CustomUserManager()

    USERNAME_FIELD = 'email'
    def has_module_perms(self, app_label):
        return True

    def has_perm(self, perm, obj=None):
        return True

    @property
    def is_staff(self):
        return self.is_admin

    def prof(self):
        return self.siteuser.screen_name

class SiteUser(TimeStampedModel):
    user = models.OneToOneField(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    slug = AutoSlugField(set_using="screen_name")
    screen_name = models.CharField(max_length=20, unique=True)

    class Meta:
        ordering = ['screen_name']
        verbose_name_plural = 'siteusers'

    def __str__(self):
        return self.screen_name

    def get_absolute_url(self):
        return reverse('siteuser:library', args=[str(self.id), str(self.slug)])

    def get_user_success_url(self):
        return reverse()

    def get_user_creation_url(self):
        return reverse('siteuser:new_activation', args=[str(self.user.id),
str(self.screen_name)])

class Wallet(TimeStampedModel):
    alias = models.CharField(max_length=30)
    owner = models.ForeignKey(SiteUser, null=True, blank=True,
on_delete=models.CASCADE)
    private_key = models.TextField()
    public_key = models.TextField()
    used = models.FloatField(default=0.00)
    balance = models.FloatField(default=0.00)

    def __str__(self):
        return self.owner.__str__() + " - " + self.alias

    def save(self, *args, **kwargs):
        return super(Wallet, self).save(*args, **kwargs)

from django.db import models
from django.utils.translation import ugettext_lazy as _

from .fields import AutoCreatedField, AutoLastModifiedField

class TimeStampedModel(models.Model):
    """
    An abstract base class model that provides self-updating
    ``created`` and ``modified`` fields.
    """
    created = AutoCreatedField(_('created'))
    modified = AutoLastModifiedField(_('modified'))

    class Meta:
        abstract = True
```

10.9 Social media account integration

```
# from django.core.files import File
import json
from random import randint

import requests

from django.template.defaultfilters import slugify

from django.db import IntegrityError
from django.core.exceptions import ObjectDoesNotExist
from django.core.files.base import ContentFile
from django.contrib.auth import login
from django.contrib.auth import get_user_model
from django.contrib import messages

from .models import SiteUser

CustomUser = get_user_model()

login_backends = {}
login_backends['django'] = 'django.contrib.auth.backends.ModelBackend'
login_backends['twitter'] = 'social_core.backends.twitter.TwitterOAuth'
login_backends['google_oauth2'] = 'social_core.backends.google.GoogleOAuth2'
login_backends['facebook'] = 'social_core.backends.facebook.FacebookOAuth2'
login_backends['yahoo'] = 'social_core.backends.yahoo.YahooOAuth2'

def save_avatar(image_url, model_object):
    response = requests.get(image_url)
    if response.status_code == 200:
        name = model_object.screen_name.lower()
        model_object.avatar.save(name, ContentFile(response.content), save=True)

def save_social_profile(backend, user, response, *args, **kwargs):
    request = kwargs['request']

    if backend.name == "twitter":
        # with open("response-twitter.json", "w+") as fh:
        #     json.dump(response, fh)
        screen_name = slugify(response['screen_name']) # twitter response contains
a screen_name
        image = response['profile_image_url']
        location = response['location']
        name = response['name'].split()
        first_name = name[0]
        try:
            last_name = name[1]
        except IndexError:
            last_name = ''
        email = response.get('email', None)
        if email is None:
            msg = """It appears you have no email set in your twitter account.
            We have created a dummy email {} for you for purpose of registration.
            Please be sure to change it to a real email.""".format(email)
            messages.success(request, msg)

        if CustomUser.objects.filter(email=email).exists():
            social_user = CustomUser.objects.get(email=email)

            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
```

```
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name=first_name, last_name=last_name,
location=location)
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}

        else:
            social_user = CustomUser.objects.create_user(email=email,
password=None)
            social_user.is_active = True
            social_user.save()

            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name=first_name, last_name=last_name,
location=location)
                        save_avatar(image, su)
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}

    elif backend.name == 'google-oauth2':
        # with open("response-google.json", "w+") as fh:
        #     json.dump(response, fh)
        screen_name = slugify(response['displayName'].strip())
        email = response['emails'][0]['value']
        first_name = response['name']['givenName']
        last_name = response['name']['familyName']
        image = response['image']['url'].split('?')[0]

        if CustomUser.objects.filter(email=email).exists():
            social_user = CustomUser.objects.get(email=email)

            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name=first_name, last_name=last_name)
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}

        else:
            social_user = CustomUser.objects.create_user(email=email,
password=None)
            social_user.is_active = True
            social_user.save()
            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name=first_name, last_name=last_name)
                        save_avatar(image, su)
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}

    elif backend.name == 'facebook':
        # with open("response-facebook.json", "w+") as fh:
        #     json.dump(response, fh)
        name = response['name'].split()
        first_name = name[0]
        try:
            last_name = name[1]
        except IndexError:
            last_name = ''
        screen_name = slugify("{}-{}".format(first_name, last_name))
        email = response.get('email', None)
        image =
'https://graph.facebook.com/{}/picture?type=large'.format(response['id'])

        if CustomUser.objects.filter(email=email).exists():
            social_user = CustomUser.objects.get(email=email)

            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name=first_name, last_name=last_name)
                        save_avatar(image, su)
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}

        else:
            social_user = CustomUser.objects.create_user(email=email,
password=None)
            social_user.is_active = True
            social_user.save()

            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name=first_name, last_name=last_name)
                        save_avatar(image, su)
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}

    elif backend.name == 'yahoo-oauth2':
        # with open("response-yahoo.json", "w+") as fh:
        #     json.dump(response, fh)
        image = response['image']['imageUrl']
        screen_name = slugify(response['nickname']) # not unique. check for
collisions
        email = "{}@yahoo.com".format(response['guid'].lower()) # make email from
guid
        msg = """We couldn't find your yahoo mail address so
        We have created a dummy email {} for you for purpose of registration.
        Please be sure to change it to a real email.""".format(email)
        messages.success(request, msg)

        if CustomUser.objects.filter(email=email).exists():
            social_user = CustomUser.objects.get(email=email)

            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name='', last_name='')
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}

        else:
            social_user = CustomUser.objects.create_user(email=email,
password=None)
            social_user.is_active = True
            social_user.save()

            if SiteUser.objects.filter(user=social_user).exists():
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
            else:
                while True: # keep looping until a SiteUser is successfully created
                    screen_name = "{}{}".format(screen_name, randint(10, 1000)) #
append a random string
                    try:
                        su = SiteUser.objects.create(user=social_user,
screen_name=screen_name, first_name='', last_name='')
                        save_avatar(image, su)
                        break
                    except IntegrityError:
                        continue
                login(request, social_user, backend=login_backends['django'])
                # return {'username' : screen_name}
```

10.10 Server run/trigger syntax



```
(venv) F:\minor\blockchain\indicoin>python manage.py runserver 8001
Performing system checks...

System check identified no issues (0 silenced).
April 06, 2019 - 03:23:25
Django version 2.1.7, using settings 'indicoin.settings.base'
Starting development server at http://127.0.0.1:8001/
Quit the server with CTRL-BREAK.
Not Found: /chain
[06/Apr/2019 03:23:37] "GET /chain HTTP/1.1" 404 4419
Not Found: /chain
[06/Apr/2019 03:24:30] "GET /chain HTTP/1.1" 404 4419
Not Found: /chain
[06/Apr/2019 03:24:39] "GET /chain HTTP/1.1" 404 4419
[06/Apr/2019 03:25:37] "GET /nodes/index/ HTTP/1.1" 200 6147
[06/Apr/2019 03:25:37] "GET /static/css/indicoin.css HTTP/1.1" 304 0
[06/Apr/2019 03:25:37] "GET /static/admin/js/vendor/jquery/jquery.js HTTP/1.1" 200 271751
[06/Apr/2019 03:25:37] "GET /static/js/indicoin.js HTTP/1.1" 404 1767
[06/Apr/2019 03:25:37] "GET /static/js/indicoin.js HTTP/1.1" 404 1767
```

10.11 Project Requirements

```
certifi==2019.3.9
chardet==3.0.4
Django==2.1.7
django-debug-toolbar==1.11
django-extensions==2.1.6
django-pure-pagination==0.3.0
djangorestframework==3.9.2
idna==2.8
Markdown==3.1
pycryptodome==3.8.0
Pygments==2.3.1
python-decouple==3.1
pytz==2018.9
requests==2.21.0
six==1.12.0
sqlparse==0.3.0
urllib3==1.24.1
```

11. Project limitation and future scope

When talking about limitations we need to define what the context is:

1. Indicoin technical limitations as a crypto-currency implemented on a de-centralized, fully replicated blockchain technology.
2. Indicoin limitations as a commodity that can be used as a store of value as an alternative to buying physical assets like gold or silver as a protection from debasement of your domestic fiat currency *but not a replacement for it*.
3. Indicoin limitations as an alternative currency one can purchase products and services in, and use as an alternative to your domestic fiat currency and fiat currency credit/debit...

Only 1Mb worth of transactions can be acknowledged every about 10 minutes, that translates to about 2200 transactions per 10 minutes at most. Only about ~7 transactions every 2 seconds! The next hard fork, should about double this number (1.8Mb) but that is still very constrictive.

This makes the supply very low and per laws of economics increases the price of the service (transactions).

It is still possible that Indicoin will succeed in the long-term. But it will likely require significant tradeoffs that will alienate many current core supporters, and result in a technology that is less impactful than people hoped.

After the success at the Dubai Block chain Summit 2018, Indian born startup QuickX has joined hands with the former Finance Minister of Malta to promote the use of crypto currency.

Talking about the technology, Mr. Kshitij Adhlakha, Founder and COO of QuickX Protocol, said, “We started working on block chain in 2016 and saw that the promising world of block chain has a few fatal flaws. We figured out that limitations such as Transaction speed, Cost,

Scalability and Cross block chain transfers are hindering its mass adoption.

The Company is looking forward to see the shape-shift and value addition the technology is yet to provide. It is said that QuickX Protocol will empower each and everyone to buy a brick to a castle, with its innovative technology, and revolutionize the world of transactions by using block chain assets.

With crypto currency startups like QuickX it is possible that Crypto currency becomes available reality in India.

12. References

12.1 Resources

```
1.  [Learn Blockchains by Building One](https://hackernoon.com/learn-
    blockchains-by-building-one-117428612f46)

2.  [A Practical Introduction to Blockchain with
    Python](http://adilmoujahid.com/posts/2018/03/intro-blockchain-bitcoin-
    python/)

3.  [Blockchain demo](https://anders.com/blockchain/)
```

12.2 Research paper references

✓ Satoshi Nakamoto. Bitcoin: A Peer-to-peer Electronic Cash System. Available at: http://bitcoin.org/bitcoin.pdf

✓ Yogesh Malhotra. Bitcoin Protocol: Model of 'Cryptographic Proof' Based Global Crypto-Currency & Electronic Payments System. Available at: http://yogeshmalhotra.com/BitcoinProtocolPaper_MalhotraYogesh.pdf

✓ Bitcoin Wiki: Technical. Available at: https://en.bitcoin.it/wiki/Category:Technical

✓ Wikipedia. Elliptic Curve DSA. Available at: http://en.wikipedia.org/wiki/Elliptic_Curve_DSA

✓ Avi Kak. Lecture 14: Elliptic Curve Cryptography and Digital Rights Management. Available at: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture14.pdf

✓ Standards for Efficient Cryptography. SEC 2: Recommended Elliptic Curve Domain Parameters. Available at: http://www.secg.org/collateral/sec2_final.pdf

✓ Bitcoin.org. Frequently Asked Questions. Available at: http://bitcoin.org/en/faq

✓ Wikipedia. SHA-2. Available at: http://en.wikipedia.org/wiki/SHA-2

✓ Coinbase. Accept Bitcoin using mobile device: https://coinbase.com/docs/merchant_tools/point_of_sale

✓ Coinkite. Bitcoin Merchant Terminal. https://coinkite.com/faq/terminal

12.3 Websites

- ✓ Bitcoin.org, Website, [electronic] Available at: http://bitcoin.org/en/about
- ✓ Bitcoin Block Explorer, Website, [electronic] Available at: http://www.blockchain.info/charts
- ✓ J.P., (2011) *Bits and Bobs,* http://www.economist.com/blogs/babbage/2011/06/virtual-currency
- ✓ Pringle, Robert, (2002) *Interview With Milton Friedman*, http://www.centralbanking.com/central-banking/opinion/1428941/interview-milton-friedman
- ✓ Voorhees, Erik, 2013 *Bitcoin – The Libertarian Introduction,* http://www.mises.se/2013/03/21/bitcoin-en-libertariansk-introduktion/

12.4 Articles

- ✓ Bulow, Jeremy (1982), "Durable-Goods Monopolists" *Journal of Political Economy,* vol. 90 no. 2.
- ✓ Coase, Robert (1972), "Durability and Monopoly" *Journal of Law and Economics,* vol. 15, April.
- ✓ Grinberg, Reuben (2012) "Bitcoin: An Innovative Alternative Digital Currency." *Hastings Science and Technology Law Journal,* vol. 4 no. 1.
- ✓ King, Mervyn (2004) "The Institution of Monetary Policy" *American Eagle Review, Papers an Proceedings*, vol. 92 no 2, May.
- ✓ Krugman, Paul, (2013) "The Antisocial Network", *New York Times,* 14th of April 2013

12.5 Literature and Academic work

12.5.1 Literature

- ✓ Fregert, Klas – Jounung, Lars, 2010. *Makroekonomi: Teori Politik och Institutioner*. 3rd edition, Lund.
- ✓ Keynes, John Maynard. *General Theory ofEemployment, Interest and Money*. Atlantic Publishers & Dist, 2006.
- ✓ W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.
- ✓ H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In 20th Symposium on Information Theory in the Benelux, May 1999.
- ✓ S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, pages 99-111, 1991.
- ✓ D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In Sequences II: Methods in Communication, Security and Computer Science, pages 329-334, 1993. S. Haber, W.S. Stornetta, "Secure names for bit-strings," In Proceedings of the 4th ACM Conference on Computer and Communications Security, pages 28-35, April 1997.
- ✓ Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002. [7] R.C. Merkle, "Protocols for public

key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980.

- ✓ W. Feller, "An introduction to probability theory and its applications," 1957.

12.5.2 Academic work

- ✓ Selign, George (2013), “Synthetic Commodity Money”, University of Georgia.
- ✓ Nakamoto, Satoshi (2009), “A Peer-to-peer Electronic Cash System”