

# THE DEFINITIVE GUIDE TO BECOME A BLOCKCHAIN DEVELOPER



**TAKE YOUR FIRST STEPS ON THE BLOCKCHAIN, LEARN  
HOW TO WALK WITH BITCOIN, AND START RUNNING  
WITH ETHEREUM AND ITS SMART CONTRACTS.**

---

# Table of Contents

Title Page	1.1
Introduction	1.2
Milestone I: Blockchain 101	1.3
Milestone II: A Bitcoin Primer	1.4
Milestone III: Bitcoin Node Setup	1.5
Milestone IV: Transaction Lifecycle	1.6
Milestone V: Ethereum	1.7
Milestone VI: Smart Contracts	1.8

# **The Definitive Guide to Becoming a Blockchain Developer**

**by Piotr Pasich**

**© 2018 X-Team**  
**Licensed under CC BY-NC-ND**

# Why We Wrote This Guide

There is no doubt that the blockchain, is playing a big role in the modern world. From cryptocurrencies and smart contracts, to e-voting systems and decentralized databases, the blockchain is revolutionizing how we think about many concepts we used to take for granted. I hope that this guide will enable you to dip your toes into the great sea of possibilities which the blockchain offers. See you there!

-- Piotr Pasich, Senior Developer, X-Team

---

The guide consists of 6 milestones.

## **Milestone I: Blockchain 101**

What is the blockchain? Where did it come from and how does it work? This milestone will give you a historic overview of one of the hottest techs today.

## **Milestone II: A Bitcoin Primer**

What makes Bitcoin tick? Starting with a short historical and terminological overview, you will then dive straight into what's going on under the hood.

## **Milestone III: Bitcoin Node Setup**

Learn how to set up your own Bitcoin node, how to work with the various nets the BTC Blockchain supports, and how to set up your first wallet.

## **Milestone IV: Transaction Lifecycle**

Learn about the JavaScript API used for blockchain operations, best practices, and the transaction lifecycle so you can build a scalable system.

## **Milestone V: Ethereum**

Learn how to work with Ethereum, the cryptocurrency for Smart Contracts, a revolutionary technology, home of the cryptokittens, and BTC's main competitor.

## **Milestone VI: Smart Contracts**

It's one thing to buy and sell crypto, but Ethereum lets you implement your business logic straight on the blockchain with its Smart Contracts. Here's how.

Satoshi mentioned that Bitcoin's purpose is to handle micro-transactions without any broker between the sender and the receiver.

Bitcoin, Ethereum, and more than 1000 other cryptocurrencies are changing the way we think about money. Even more importantly — from a programmer's perspective — they have also created an extraordinary opportunity for new businesses based not only on cryptocurrencies but also on the Blockchain itself. This technology may be considered as a new, highly effective and secure system for exchanging information between banks, healthcare services, or applications distributed around the world.

In this series of articles, I will present the basics of the Blockchain, Bitcoin, Ethereum, and contracts from a developer's point of view. They will enable you to understand and start playing with the mechanisms behind the Blockchain and cryptocurrencies. Finally, you will be able to implement your own integration, system, or even a game.

This course is divided into several sections in an order which I would have liked when I first started my Blockchain journey. Each main section will be called a Milestone. Every Milestone will build on the knowledge from the previous one.

Albert Einstein said: "If you can't explain it simply, you don't understand it well enough." I will attempt to explain all the parts as simply as I can and refer to real life as often as I can because I remember how many problems I had with understanding all the mechanisms and algorithms at the beginning.

## What Is the Blockchain?

Some say that the Blockchain is a chain of blocks. It sounds funny in the beginning, but after the first impression, this sentence brings many questions: "What is a chain?", "What is a block?", "What kind of information does the block contain?", etc.

## Historical Overview

Imagine this is 1998, and you have just read a paper published by an engineer called Wei Dai, which describes a new electronic cash system that is based on a protocol which does not require any real money or checks to transfer funds from one account to another. This was an era before the credit cards became so popular and before PayPal (PayPal was established in December 1998 as Confinity).

## B-Money

Wei Dai claims that a community should own the money, not one centralized organization, company, or individual, and the owners should be anonymous. All parties are identified only by a number or a public key.

The system described by Wei Dai and inspired by Tim May's crypto-anarchy should cover a couple of aspects in his opinion:

- No one owns the system (neither the government nor a single institution)

In a traditional banking system, one organization — the bank — manages the funds that others put into their account. At this point, the Bank as the owner permits to transfer money from one account to another and confirms this transfer. The bank calculates the current money balance available on the account.

In the result, the Bank can manage the funds in whichever way they choose to — it is possible to reverse or cancel the transfer, block your account, or add some limitations. The question is, if the Bank can do all of this stuff with your account, does it mean that you still own the money?

A decentralized financial system may be a solution for this problem. If nobody owns the system itself, nobody can change the rules or agreements alone.

- It should provide an efficient medium of exchange the money

The main purpose of having money is to exchange it for some goods. Transferring funds from one account into another is one of the fundamental features of any financial system.

Moreover, making a transfer should be as easy and secure as possible. Nobody likes to write a check with all the mandatory information and then confirm in the Bank that this specific check is valid, and the owner has enough funds on his account.

In the best case scenario, the transfer should be a short message that says: "The X-number of B-money, should be transferred from Account A to Account B. Transfer signed by A-Account Owner."

- It should provide a way to enforce contracts

In the real world, all rules or laws from a specific country apply between the transaction parties. In the Blockchain there is no room for interpretation or misinterpretation, it should provide a mechanism based on conditions and numbers. If two or more sides agree to the agreement conditions and all requirements are qualified then the payment or the transfer should be done automatically.

- The system exists because of cooperation of all participants

This rule is an implication from the first one — no one owns the system. That also means that anybody who wants to participate in the system needs to cooperate to keep this system running. All transfers, contracts, and actions should be computed and confirmed by the participants.

- It should be fully anonymous

Because the system is distributed and every participant has a copy of all information, that makes the system quite transparent. However, nobody wants to share all the information about their financial transactions with the world. That means the system should also be anonymous.

The proposed solution is to change the names into numbers and do not add any additional fields to transfers. As a result, the accounts or the transfers cannot be identified, because they are just numbers, i.e., some amount of B-money are transfers from account 1111 to account 2222 signed by 3333.

The protocol did not describe how it should be done, but the general idea, which is to create a decentralized system where all nodes have their own copy of the same database keeping information about all users, transactions and the current state of this system. The question is how to keep those nodes or servers continually working, how to encourage their owners to participate in the system and how to secure the information from all vulnerabilities. And the answer is always the same — money.

## Where the Money Comes From

Usually, in a traditional financial system, the government is responsible for printing and distributing money. In the system where everything should stay decentralized and none should be able to make or generate the money without a clear agreement with other participants, the question where the money comes from becomes pretty solid.

The system cannot be opened with one address that owns and distributes all funds. Wei Dai proposed to reward the users for solving specific, complex calculations with some amount of virtual units — B-money.

The computations should be quite complicated to perform, but easy to confirm by others. Moreover, all the work done by the servers should be useless and worthless, so there may be only one reward for doing them — the B-money or any other virtual currency. I.e., let us imagine a problem to compute that takes 100 hours to solve on a typical machine. Those 100 hours may be rewarded with 1 B-money coin, the same amount of funds that were required to perform the computations.

# Bitcoin — The First Blockchain Implementation

In 2008, an individual or an organization called Satoshi Nakamoto published a manifesto that described a possible implementation of a new virtual currency. In January 2009, this cryptocurrency was presented to the world — the Bitcoin. The idea behind Bitcoin was to create a system that meets Dai Wei's B-money description and provides a solution for all the problems pointed out 10 years prior.

Because Bitcoin was the very first fully-functional and widely accepted blockchain implementation, and because most of the new systems and applications that are working with the blockchain are based on the Bitcoin mechanism, all further information about the blockchain will apply to Bitcoin.

## What Kind of Blockchain Problems Does Bitcoin Solve?

### Decentralization and Distribution

Bitcoin is designed as a peer-to-peer network where every node (server or individual computer that has installed a piece of software able to communicate properly with the Bitcoin network) has its own copy of the database that stores information about accounts and transactions between those accounts.

The nodes communicate with each other to synchronize the current state of this information and determine if all pieces of information have been broadcasted properly or whether they should be removed or omitted from the database.

As a result, the Blockchain as a service is owned by the community, nor any organization, government, or company,

### Security

One of the reasons why the blockchain is decentralized is security. No single entity, organization, or company is in the possession of the blockchain, so nobody except for the owner should be able to manage the funds.



This approach has pros and cons. From one point of view, nobody can block the account, take some money from it (because they can change the database or the code) or cancel the transfer. On the other hand, if you make a mistake you will not be able to undo it. The money may be lost.

Bitcoin as the first blockchain implementation cryptographically secures the data in the ledger. We will explain exactly how in the next part.

## Anonymity

The blockchain idea does not specify if the data inside should be public or not. As far as the people are concerned about the personal information that may be published, it should always provide the option to stay anonymous. Just imagine that somebody publishes your name with a full list of your transactions. That is how the blockchain would look like if it asked for your name.

In most (if not all) implementations that I have worked with, the blockchain keeps only numbers — addresses of wallets, blocks, or transactions. The wallet is not assigned to any user. Everybody who has the password to the selected wallet can access it by passing this password and the address. The result is that one user can have access to many addresses and only they would know to which ones.

All transactions included in the blocks preserve this anonymity too. A transaction can be explained with a simple sentence: "Some amount of funds is sent from address A to address B and signed by address A's owner." This provides all necessary information and does not publish any sensitive data at the same time.

## Availability

If the network is distributed around the world and nobody owns it, how to keep it working continually? How to persuade the servers' owners that they should keep their machines up and running? Promise an income that may compensate the cost of those machines and electricity.

Every block has to be confirmed by one of the nodes that are available in the network. The node calculates the hash from the block and publishes this hash across the blockchain. If other nodes confirm that the block hash is correct, the node which made the calculations is rewarded with some constant amount of funds (these days it is 12,5 BTC per block) and with all fees that were added to transactions in this block (I would say approximately 1-3 BTC per block).

# Different Blockchain Use Cases

Currently, the blockchain supports mostly the financial industry. Almost 1000 different cryptocurrencies are available to share (the full list can be found at <https://coinmarketcap.com/all/views/all/>). The two most important cryptocurrencies are Bitcoin and Ethereum. They have some differences, but both are working on the same principles.

However, the Blockchain as a technology can support other industries. Imagine services that need to share information with each other, but do not share a common protocol — like the banks and transfers between them. All banks use some third party software to transfer money from one to another, like SWIFT or Fedwire. This increases the fee. The Blockchain solves that problem by unifying the information that can be shared between banks; imagine that every bank server is a node connected to some Blockchain (not Bitcoin, but a dedicated Blockchain). If one bank shares a piece of information about a transfer, every other bank will know about that in a matter of few minutes, without any broker in the middle. Insurance, gift cards, and loyalty programs. Government and Public records, and other systems that are divided and communicating with each other, but do not have a common protocol.

## Healthcare Industry

There are 26, if not more, electronic medical records options just in Boston. On the other hand, in my country, the main part of the health record is still kept on paper. Maintaining your own health record may be a huge challenge and just think about all the people living in your district.

Now, imagine that if the doctor writes a new prescription, it is immediately published to other facilities.

## Voting System

The voting process can be supported by the Blockchain technology. Every citizen may be able to vote from their home with a blockchain client. The votes will be automatically published to the network and counted after every mined block.

## Academic Certificates

This is what everybody puts on the resume when looking for a new job opportunity or when they want to continue their education. Build apps that issue and verify blockchain-based certificates for academic credentials, professional certifications, workforce development, and civic records.

## IP

Intellectual Property during the Internet era may be a tough topic. Spreading information about the copyrights to a movie, song, or piece of art between countries, or registering a worldwide patent is a long, expensive process because the information about intellectual rights or new inventions needs to be shared across many different databases and systems.

Putting that information in the Blockchain would solve the distribution problem and help with commonizing a knowledge source.

## Smart Contracts

This is one of the most-mentioned features of the blockchain. The blockchain allows one to write simple (in Bitcoin) or more sophisticated (in Ethereum) contracts between parties (two or many). Those contracts are based only on true/false conditions and numbers. It means that if all conditions meet the numbers, the Blockchain automatically transfers the funds.

## ICO and Tokens

Another method to use the smart contracts is to distribute tokens. ICO (Initial Coin Offering) projects are meant to raise funds for companies for future investments. The ICO presents white papers where the project is described, and investors can buy tokens (usually called by the ICO name). The tokens are distributed in the selected Blockchain. The most popular one for this usage is the Ethereum.

You can think about the tokens like a sub-currency in another cryptocurrency. If somebody publishes a new token, you can exchange the tokens between investors or sell them, but every transaction requires a fee paid in ETH.

## Cat Contracts and Games

Another type of smart contracts is what I like to call Cat Contracts. Some time ago, a new token type was published on Ethereum, and it caused many problems in a particular Blockchain, namely [CryptoKitties](#). CryptoKitties is a game that is based on a smart contract

which allows users to buy virtual Kitties, keep them, sell them or reproduce them. The game became so popular that it generates more than 10% of transactions in the Ethereum network.

## Blockchain Security

On the Internet, the biggest challenge will always be the security. From my own professional experience, I can say that there will always be somebody who wants to crack your code. The first thing that comes to mind when you think about a hacker that can change the state of the Blockchain is stealing the money. On the other hand, let us take another example — the gun-tracking system — and imagine what may happen if somebody unregisters or re-registers a pistol that was used to commit a crime.

## Double Spending

One of the main concerns regarding the Blockchain is the double spending problem — some amount of money is sent from one account to another, and then, the same money is sent to a third account. Bitcoin solved this problem by cryptographically connecting the newest block with the previous one and with a mining mechanism which calculates a hash number from all transactions included in a block. This mechanism will be described in the chapter dedicated to Bitcoin.

One of the methods that allow one to omit the cryptographical bridge is to possess more than 51% of all nodes in the network which will confirm the blocks. This may be pretty hard to achieve if you consider that the current hash rate is [13,523,349.50 TH/s](#) and a mining machine which handles approximately 14 TH/s costs around \$8000.

The complexity of calculations in the mining mechanism also prevents the brute-force attack vector. If somebody tried to publish all the possible hashes that may be accepted by the Blockchain and expect a miner's reward, I would say that:

- Supposing you could generate a billion ( $2^{30}$ ) per second, you need  $2^{130}$  seconds.
- Doing this in parallel using a billion machines requires only  $2^{100}$  seconds.
- Getting a billion of your richest friends to join you gets it down to only  $2^{70}$  seconds.
- There are about  $2^{25}$  seconds per year, so you need  $2^{45}$  years.
- The age of the Universe is about  $2^{34}$  years so far — better get cracking!
- [Is it possible to brute-force bitcoin address creation in order to steal money?](#)

On the other hand, there is a running project called a [Large Bitcoin Collider](#), which tries to guess or brute force the private keys to selected wallets containing some Bitcoin, and they have already broken some of them!

## Conclusion

In this part, you learned about the basic concept of the Blockchain and its general idea, and got a historical overview. In the next section, I will explain how Bitcoin works and describe all terms necessary to start working with it.

Bitcoin can be described as a ledger of transactions. Every transaction can be explained by a simple sentence.

Bitcoin is a form of digital currency based on the Blockchain and allows the owners to mine, transfer and store funds in their digital wallets. The network grows every day and more and more shops, services, and investors are accepting this kind of payment in their business. In this chapter, I will give you an introduction to the Bitcoin world, explain the basic terms and how it works. If you want to become a blockchain developer, this is the perfect point to start — from the very beginning, with the most popular cryptocurrency.

## Bitcoin — A Historical Overview

Bitcoin was mentioned for the very first time in October 2008, by an individual or organization called Satoshi Nakamoto. He or she published a paper on a Cryptography Mailing list explaining a new idea of a cryptocurrency called Bitcoin that was decentralized, transparent, anonymous, and cryptographically secured. On the 3rd January 2009, they published the Bitcoin software and mined block 0. Until mid-January, all mined funds remained untouched.

Satoshi mentioned that Bitcoin's purpose is to handle micro-transactions without any broker between the sender and the receiver, i.e., without a bank or a government that could block your assets, cancel the transfer, or freeze the account. With Bitcoin, you are the money owner with all the rights and obligations. It is a huge responsibility. If you transfer funds to a wrong account or even to the wrong network (that happens surprisingly often), it will be gone.

## How Bitcoin Works

Bitcoin can be described as a ledger of transactions. Every transaction can be explained by a simple sentence.

Some amount of funds is transferred from address A to address B and signed by A's owner with their private key.

This sentence introduces a couple of terms that are fundamental for Bitcoin — the address, the transfer, and the private key. There are a couple more that I will explain — a block, a confirmation, etc.

Imagine that you write this sentence on a piece of paper and give it to your friend. He also wants to make a transfer, so he writes his own sentence below yours in the same format — word for word — and changes only the A, B, and private key variables. The paper goes to the next person until there will be no more place to write or until the time frame will end. After 10 minutes, the paper is put on the stack, on the top of other similar papers.

This stack is the blockchain, a single piece of paper is a block, and the sentence you wrote is a transaction.

Bitcoin is a digital currency, and the general perception of the money transferring mechanism seems to be pretty similar to Internet Banks. I bet you have one or two open accounts in some Internet Bank. Those accounts are just numbers. To get access to those accounts and manage your money, you need to enter your username and password.

When you log in to the bank's website, it is possible to make a transfer by filling the amount, a title and a receiver's address. In most banks, it is not mandatory to put all information about the receiver, like the name or address.

Bitcoin generally works in the same way and may be compared to the Internet Banks, but with few differences.

## Dictionary

### The Address

If you want to start playing with Bitcoin you need to open an address or a wallet — that is how many addresses are called. Because Bitcoin keeps the users anonymous, some of the information has to be omitted — like the name or the transfer title.

Opening a new address can be as easy as registering a new email address. There are many services and applications that have an interface for Bitcoin wallets, like bitcoin.info or blockchain.info. A new address can be opened from the web browser without installing any additional software on your computer. Usually, the account will be protected only by a password, and if you would like to use this address from other services, a private key should be generated.

Another method to open a new address is to install one of the node applications like the Bitcore, the btcd, or the Bcoin. The difference between the software installed on your computer and services available on the Internet is that, for the former, no one keeps the

password to your account but you. However, installing the Bitcoin node requires 160-200 GB of free space on the disk and a couple of days to download and synchronize your database with the public Blockchain.

## The Wallet Address

The Wallet Address is a specific kind of address that you can login into. For every transaction, a new address should be generated (see The Transaction section). It does not mean that you should keep all the addresses and private keys for them to manage your funds. All new addresses will be automatically assigned to your wallet. All funds received by any of the assigned addresses will automatically be counted into the wallet balance.

The wallet number should not be used for any transaction and should not be shared with anybody else.

## The Transaction

As I described before, the transaction can be described as a statement that some number of Bitcoins are moved from address A to address B and signed by address A's owner's private key. This is the simplest situation that may occur in the blockchain. If you run an e-commerce shop that sells much stuff in one day, you will receive many transfers. Usually, every transfer will have a defined title related to the order number in your shop. Because of the anonymity that the Blockchain provides, there will be no title in the transfer, and it would be irrational to ask your customers to paste their Bitcoin address into every order.

## Requesting Payment

In Bitcoin, if you would like to request a payment from someone else, you generate a unique address for this specific payment or a specific person. So, you, as a wallet owner, can have many addresses and generate a new one every time somebody needs to pay you with Bitcoin.

In the e-commerce shop, when the user selects the Bitcoin payment method, a newly generated address should be presented to him. A QR code is usually generated with the address because many users are using their smartphones to pay with Bitcoin. It is recommended to avoid rewriting the address by hand, as every mistake will cause the money lost.



The generated address should be saved in the database and assigned to the order. When the user makes the transfer, the funds appear at the generated address, and the user or the order can be recognized by this address.

There is no need to generate a new address for every order. They can be generated for a specific user or for a specific case. That depends on your business needs — if you run an e-commerce platform, then payments should be related to orders. If you run an investment or ICO project, then maybe it is best to generate an address for a user, and when you are raising money by an application similar to the Kickstarter, you may consider keeping only one address per project.

## **Making a Transfer**

Let us consider another point of view — the returns. We will stick to the e-commerce example for a little bit longer. If somebody orders a dress and it does not fit, they should be able to return the order and receive the money back. The application should ask for the pay-back address. This address should be generated by the user, but it is not necessary.

The system prepares the transfer and publish it to the blockchain. This transfer will not be confirmed until it is mined with the next block. Moreover, there should be a fee added to the transfer which will be received by the miner. The fee is set by the sender and should be balanced to keep it low, but not too low, because it may happen that the transfer remains waiting on the pending list forever.

I described that a transfer could be described as moving funds from address A to address B. But, as I wrote before, the funds are counted to the Wallet. If you received 10\$ to 10 different addresses and would like to make a payment for 10\$ you do not need to make 10 different transfers. In Bitcoin, every unit is assigned to your wallet, and you make a payment from your wallet and sign it with your wallet's private key. In the Blockchain, the transfer will be presented as a transfer from those 10 different addresses to the receiver address.

This transfer is done by one user that received his funds to a couple of different addresses that are listed on the left side. If the amount of money does not match the amount of money that is sent to the receiver, the rest is transferred to a new address that will be assigned to your wallet. If the numbers match, there will be only one address on the right site.

## **The Fee**

Every transaction needs to be sent with the fee. The miner will be rewarded with this fee when the transaction is added to the next block. All miners want to earn as much Bitcoin as they can so transactions with the highest fees will be mined first. The Bitcoin algorithm does

not specify the lowest fee, so it is possible to send a transfer without any fee, and some of the miners accept transactions without fees, but this is a really rare situation. Most likely, this transaction will remain stuck on a pending list.

## The Block

The block is a set of information with a list of transactions. The block is defined by its hash number, the previous block's hash number, and the list of transactions. Every block is published in approximately 10 minutes. That means if you send a proper transaction with the fee high enough to be mined in the next block, the block with this transaction may be published in next 2 minutes, next 10 minutes, or next 60 minutes (with another 6 blocks).

When the transaction is added to the first block, it is already confirmed.

## The Blockchain

The Blockchain is a ledger containing all published and confirmed blocks in a specific, chronological order. All blocks that are added to the blockchain are safe and cryptographically confirmed. The question is, what happens if somebody wants to add a wrong block to the blockchain? Bitcoin provides a mechanism that secures this state called mining or proof of work. Those two terms define the same thing.

## Confirmations

When you receive a transfer to your bank account, you trust that that money can be spent. Bitcoin does not provide a true/false value that says the transaction was successfully processed, but gives a number of confirmations. Because some of the blocks may be orphaned (for different reasons) the number which you trust should be balanced with the time of confirmation.

That happens because while every block is mined in approximately 10 minutes, it depends on the fee and the blockchain load. So, if you would like to be pretty sure that the payment is secure, then it has to have more confirmations. This implies that it requires more time — 1 confirmation takes 10 minutes, 6 confirmations take 60 minutes.

My professional experience says that we can divide the payment types into 3 different groups, depending on how much we trust the sender and how much money we are willing to risk losing in the worst case:

- Small transactions — usually e-commerce websites have a lot of them. Most of those

transactions are below 50-100 USD. Moreover, the time between the transaction and the moment when an order will be sent is a day or two, so only 1 confirmation is enough to trust the payment.

- Medium transactions — in this case, let us say you raise the payments for 1,000 USD or more. Losing this kind of money will not cause huge damage to your business, but it is a good idea to secure them better than 1-2 USD payments. It is quite unusual for 1 miner to be able to confirm 3 blocks in a row, so 3 confirmations.
- Big transactions — the ICO projects became more popular lately. Even if some say that the trend is calming down, those project still raise significant amounts of money. A single investment can be more than 500,000 USD, and in selected projects, the tokens are distributed just after the transaction is confirmed. For those transactions, it is necessary to make sure, that it is 100% confirmed, but the confirmation time is not so relevant. Mining 6 blocks in a row is virtually impossible, so I prefer to wait for 6 or more confirmations from the blockchain.

## The Mining Mechanism

Much trust is put in the mining mechanism when we think about the Blockchain, Bitcoin, or other cryptocurrencies. The algorithm should secure the network and prevent double spending. To ensure that can happen, the blocks should be published in correct order. To keep this order and propagate blocks to all nodes, Bitcoin allows publishing a block in approximately 10 minutes. This time or the process does not depend on the computational power of a computer or a computer's network. If the computation power allows for mining the block faster than in 10 minutes, then calculations for the next block will be harder and will take more time. On the other hand, if the block were to be published after 10 minutes, that means it was too hard to calculate, and the difficulty level should be adjusted.

Mining should be a hard process that takes a lot of resources and time. However, confirming that the block is correctly mined should be fast and easy.

## Discard the Dice

Imagine that you have a 12-sided dice in your hand and you have to discard all values lower than 6. You roll the dice until you have an expected result. Moreover, let us agree that we will be doing that over and over again, and each time that we get a correct result, we will call it a block. Our job is to make sure that every block is mined in approximately 6 discards.

If you need to discard the dice more than 6 times per block, you adjust the number to be lower than 7 (6 plus 1) for the next block. In the opposite situation, when it took less than 6 rolls, we make the process harder by rolling a number less than 5 (6 minus 1).

In our game, the mining mechanism requires a simple action to be repeated several times until the expected result is achieved.

However, if somebody else looks at the table, they may easily confirm if the last result is correct or not.

## How It Works in Bitcoin

Of course, the previous example is simplified to show the general idea of the mining mechanism — that the block should be hard to generate and easy to confirm. The Bitcoin node has access to a list of pending transactions and creates a block that contains all accepted transactions (usually in a fee order, from the biggest to the lowest). All the information included in transactions have to take up less than 1MB and the calculations complexity has to be lower than the computationally difficult level set with the previous block.

The node takes some selected amount of transactions and calculates the SHA-256 hash for them. The difficulty level requires a minimum number of zeros at the beginning of this hash. If the number is bigger, the block is accepted and can be published to the blockchain. If not, the node takes another set of transactions and starts the calculations from very beginning. This process is repeated for the next 10 minutes until a new block is mined.

Proof of work is not only used by the Bitcoin blockchain but also by Ethereum's and many other blockchains.

## Proof of Work vs Proof of Stake

Proof of Work refers to cryptocurrencies such as Bitcoin and means that the miner is rewarded for the calculations needed to create a new block.

Proof of Stake is a different way to validate the transactions. The creator of a new block is chosen in a deterministic way, depending on its wealth, also defined as stake. So, he takes only the fee from all transactions included in a block with no pre-defined reward. The miners in the Proof Of Stake networks are called forgers.

The next difference is that, in the Proof of Stake mechanisms, all the coins are generated initially, and their number never changes.

The Proof of Stake algorithm is meant to limit the electricity that is consumed during the mining process. A single Bitcoin transaction requires electricity worth \$8, and in one block, there are about 2000 transactions included. Because the PoS does not reward the fastest miner, but one of the forgers from the validation pool, the need to use high-powered computer networks will be limited.

## Conclusion

Now, you know all the basics about the Blockchain and the Bitcoin. These sections defined all the most important terms and explained the mechanisms. You need to remember what the blockchain, the block, the transaction, the confirmation, the fee, and the mining mechanism are. In the next part, we will set up a Bitcoin node, make and simulate the first transaction and explain step-by-step what happens.

We have already gained Bitcoins for development purposes, and we are ready to start programming.

This is the part where the real fun with the Blockchain starts. I will show you how to set up a Bitcoin node, explain what the differences between modes are, show you how to send a simple transaction and describe the transaction lifecycle. This will be a prelude to integrating a JavaScript application with Bitcoin payment.

**Requirements:** For the purpose of our examples, I will assume that you are familiar with JavaScript and the npm package manager.

## What Is a Node

The node is a client and a server kept in one software application. It can either run as both or one can be disabled. If you want to integrate your application with Bitcoin payments, then maybe it is not the best idea to waste computation power for mining.

The node provides all necessary features that support mining and synchronizing blocks, sending transactions, etc. Once you install the node, it will look for other nodes to connect with and download the latest state of the Blockchain. Be prepared, this process takes a while. The current Bitcoin database is about 160GB, and it grows every day.

## Peer-to-Peer

The Bitcoin network is based on the peer-to-peer solution. There is no central point or a server that manages how the data is transferred between the users, and there is no central database where the data is kept. So, the whole blockchain needs to be downloaded and synchronized by every node.

Once the node starts running, it checks the list of last connected IP addresses to synchronize with them. This list is broadcasted with the last block. If no list is saved, the node pings well-known list of IPs that are connected to the network. This list has been hardcoded since 2010. The last step of seeking a node in the network is DNS seeding.

## Which Node Should I Choose

Bitcoin is not a specific piece of software or an application. I would say that it is a full description of a protocol that can be implemented by anyone, and anyone can join the network with their own software. It does not mean that if you implement your own client, it

will be able to broadcast corrupted data and get it accepted. The Blockchain secured transactions and blocks from such a situation.

That is why there are so many available clients on the Internet. There is `btcd` (with `btwallet`), `Bitcore`, `Bcoin`, `decred`, etc. There are a couple of differences between them — some separate the Blockchain management from transactions, some provide additional indexing features, and I will stick to `Bitcore` because it supports the JavaScript and is used by enterprise businesses like `Trezor`, `Bitpay`, or `Streamium`. It keeps all features in one application, so that will make our examples easier to work with. Some add support for unspent transactions lists. However, all of them are fully-working nodes that can be used in our example.

I will stick to `Bitcore` because it supports the JavaScript and is used by enterprise businesses like `Trezor`, `Bitpay`, or `Streamium`. It keeps all features in one application, so that will make our examples easier to work with.

## Modes

When we run the node without any additional flag, it will connect to the main Bitcoin Blockchain and download approximately 160 GB of data or more. This is expected behavior when you want to run the node in the production environment, start mining or make real transactions. But, during the development process, it would be much easier and cheaper to mine blocks faster and avoid wasting funds on the fees. Most of the nodes available on the Internet can connect to other Bitcoin networks or simulate own network, and each network mode has own purpose.

There are times when people get confused with the modes and transfer Bitcoins from the test mode to the main network. It is not possible to transfer coins between different networks. They will be lost.

In every mode, the addresses start with a different identifier. In the `TestNet` it is `0x6F` or `0x9F` rather than `0x00` in the `MainNet`. If you will transfer coins from a `TestNet` address starting with `0x6F` to an address starting with `0x00`, the node will not inform you about any problems and will just make this transaction. The funds, however, will be moved to an address that does not exist. This happens because the Blockchain supports full anonymity. It does not store a list of available addresses, only a list of transactions. Bitcoin does not know whether this address does or does not exist — it gives full freedom to the money owner, with all responsibilities and punishments.

## MainNet or LiveNet

This is the main ledger that everybody operates on. If somebody is talking about the value of Bitcoin, she refers to this particular network and all Bitcoins stored there have real value. The name may depend on the node you are using. In the btcwallet, it will be MainNet, while in Bitcore, it will be the LiveNet.

The default listening port is 8333, and the RPC port is 8332.

## TestNet

This mode is intended for test transactions and the development process. Twice before, people started trading these funds for real money, so the TestNet was reset twice, and all assets were lost. Bitcoins stored in the TestNet do not and should not have any real value.

The TestNet allows to fully simulate the behavior of the MainNet without risking real money or assets. Bitcoins are free and easy to mine. The minimal difficulty level is half of the minimal difficulty level of the MainNet, and if no block is mined in 20 minutes, the difficulty level is reset to the minimum value.

Also, ports are different — 18333 is for the listening protocol and 18332 for the RPC API. The Blockchain size is approximately 4-16GB.

## SimNet

The Simulation Network is intended for tests on a local machine. This mode allows us to run your own, private network and to force the Bitcoin node to mine or generate blocks whenever we want. Every action is done manually. If a transaction is sent, it will not be mined or confirmed until you run a specific command.

I use this method to speed up the development and debug the transaction process, but because of the number of transactions that I can manually generate and confirm in one block, its use is limited, and I often receive empty blocks. I strongly recommend to re-test all features with the TestNet or even the MainNet.

## Installing Bitcore

The Bitcore installation process is fast if you know npm. The full operation is described at <https://bitcore.io/start>, and it contains only 3 points, but we will extend it a little bit because bitcore also requires additional libraries which are only mentioned in the official installation guide.



## Install Required Software

Bitcore requires you to install node.js (version 4 or higher) and ZeroMq without root privileges.

**For linux users:**

```
apt-get install libzmq3-dev build-essential
```

**For mac users:**

```
brew install zeromq
```

## Install Bitcore with NPM

Bitcore is installed as an npm package with the following command. It will be installed globally.

```
npm install -g bitcore
```

## Start your node

To synchronize Bitcore with MainNet, run it with the bitcored command. This process will take approximately a week or so because the Bitcoin blockchain is about 160 Gb at the moment.

```
bitcored
```

But for development purposes, we would like to proceed a little bit faster, omit this process and go straight to the next steps of our example. I would recommend switching to the TestNet mode (Test network). Bitcore does not support the SimNet mode (Simulation network) yet. TestNet is about 10-16 Gb and will synchronize in at maximum a couple of hours if not faster.

Because the TestNet does not operate on real money, it is better for development and test purposes — there is no need to have real Bitcoins or spend them. Thus, the possibility of losing funds is limited.

To run Bitcore in the TestNet mode, an additional node needs to be created by running the command:

```
bitcore create mynode --testnet
```

After that, a new directory will be created at `mynode`, where the Blockchain files will be stored. Another important thing is that the configuration, which can be found at `~/.bitcoin/bitcoin.conf`, will describe the method and credentials for the API endpoint.

```
{
  "network": "testnet",
  "port": 3001,
  "services": [
    "bitcoind",
    "web"
  ],
  "servicesConfig": {
    "bitcoind": {
      "spawn": {
        "datadir": "./data",
        "exec": "/home/papi/.nvm/bitcore-node/bin/bitcoind"
      }
    }
  }
}
```

After the node is created with the config file above, it needs to be synchronized with the blockchain. Enter the `mynode` directory which was created for TestNet and run Bitcore.

```
cd mynode
bitcored
```

A list of downloaded blocks will appear. They should be grouped in bunches of around 2000 each. At the very end of each information line, there will be progress information. Each block can be described with the hash and also a number, and these numbers are in a specific

order. The currently accepted block number can be previewed on every website dedicated to Bitcoin which also supports TestNet, e.g., <https://live.blockcypher.com/btc-testnet/>. So, at least you will know how many blocks there are to download and can calculate how much time your node needs to synchronize.

## Block Explorer

Bitcore can be connected with a user interface where all the information about the current state of the Blockchain can be presented. I am pretty sure that you are familiar with this interface. Maybe the colors are different, or the place where buttons are located, but if you have even the most basic knowledge about Bitcoin, you probably visited sites like [blockchain.info](http://blockchain.info).

Running commands on the command line can make the learning process faster, but copy-pasting the block and transaction hashes may be time-consuming during everyday work. Therefore, I prefer to use the web interface to control what is published in the blockchain.

Why do I not recommend you just visit an external page that will present all this information? It may happen that you will publish a transaction and want to see what happens with it before every other node on the internet will get information about it. Also, your node may be desynchronized (has not downloaded all the blocks from the Blockchain). So, if the transaction is on the Blockchain and your node doesn't know about it, you may miss this piece of information.

To install the bitcore user interface, you just need to run:

```
bitcore install insight-api insight-ui
```

This will download the packages `insight-api` and `insight-ui` and run the npm installation process. The next time you start your node, the user interface will be available at <http://localhost:3001/insight>.

## Setting Up Your Wallet

After the node is synchronized, to operate with Bitcoin, a Wallet needs to be created. This will allow you to not only review what is stored in the Blockchain but also send and receive transactions, create new addresses, etc. Most of the next steps will be done from the command line, but the information about the Blockchain state will be visible in the User Interface. After a wallet or a new address is created, it can be previewed on the website.

## Installing the Wallet Service

For now, Bitcore is able only to read the data from the Blockchain and does not support sending and receiving funds or managing wallets and addresses. We need to install additional software called the Wallet Service. The Wallet Service's main responsibility is to publish data to the blockchain and maintain your wallet.

The Wallet Service requires a MongoDB database to run. Other nodes do not need to operate with the local database but also do not provide a web interface, only the command line interface.

**Mac OS X** The easiest way to install MongoDB is to use brew:

```
brew install update
brew install mongodb
mkdir -p /data/db
sudo chown -R `whoami` /data/db #this assumes that the next step
will be run by the current user
Mongod
```

Ubuntu/Debian MongoDB is included in the standard repository and can be installed from aptitude:

```
sudo apt-get install mongodb
```

This should automatically start the mongod service in the background.

Install the Wallet Service to the MyNode The Wallet Service can be installed from the command line, like the User Interface.

```
cd mynode
bitcore install bitcore-wallet-service
bitcore install insight-api
npm install -g bitcore-wallet
```

Those steps will install the insight-api plugin for bitcore, the wallet-service, and the command line available from npm. Now, Bitcore should be restarted.

```
bitcored
```

These steps provide us with the ability to operate on the wallets and addresses, but we still need to open a new wallet.

## Opening a New Wallet

The `bitcore-wallet` npm package adds two commands to the CLI — the `wallet-create` and the `wallet`. They allow us to operate on funds, addresses, and transactions, but first of all, the Wallet should be created.

The Wallet is the primary place where the funds are collected. The mechanism of Bitcoin assigns many addresses to a single Wallet.

```
wallet-create -h http://localhost:3232/bws/api --testnet  
'myWallet' 1-1
```

Now, `bitcored` can manage the funds that will be sent to one of its addresses. Because Bitcoin provides full anonymity and the transaction title, the sender's name or any additional information cannot be included in the transfer; we need to create a new address for each payment or sender. All the funds received at the addresses that we will create will be automatically assigned to the wallet.

To create a new address, we can use the `address` method from the API which is available from the CLI:

```
wallet -h http://localhost:3232/bws/api address
```

The new address will be output. This address can be used to transfer funds to your wallet. However, the question is how to get the funds into TestNet. Mining requires a lot of computational power, time and does not give you any guarantees that it will succeed. Fortunately, Bitcoins available in the TestNet are free, and people like to share them. You can use one of the websites that will send you Bitcoins when you enter the address, e.g., <http://tpfaucet.appspot.com/> or <https://testnet.coinfaucet.eu/en/>.

Information about my test transaction can be found at the address <https://testnet.blockchain.info/address/mjm7r1GZDnKUFu5NdtiPg49gyRrwwuhV9g> or in our Insight application at <http://localhost:3001/insight/address/mjm7r1GZDnKUFu5NdtiPg49gyRrwwuhV9g>.

Remember that TestNet addresses and LiveNet addresses are different. If you enter the LiveNet address into one of those services, the funds will be lost, because the networks are totally separated and do not know about each other.

# Receiving Bitcoins

Now, we have to wait for the Bitcoins to be transferred to our wallet. The first step that will be done by the external application is to broadcast the information about a new transaction to the TestNet Blockchain. You can find this transaction by entering the transaction hash number into the User Interface that we installed. It should have 0 confirmations. Don't worry if the UI cannot find this transaction. Check if the node is fully synchronized with the blockchain. It may take a little bit.

The second thing that will happen is that the transaction will receive the first confirmation. Now, the transaction is included in the last block and can be easily found in the Blockchain. From now on, the funds are in your wallet, assigned to the address that they were sent to.

The third step is to get more confirmations. I described in the previous parts that the number of confirmations is strictly related to the number of blocks that are mined after the block that includes the transaction. The more confirmations a transaction has, the more confident you can be that the transfer is legit.

## Checking the Wallet Balance

When the transaction has at least one confirmation, the Bitcoins will be assigned to your wallet. To check the current wallet balance, we can use an API method:

```
wallet -h http://localhost:3232/bws/api status
```

This will display the current state of the funds that are assigned to all the addresses you manage.

Now, it may be time for the first exercise for you. Please create a few new addresses, send Bitcoins from TestNet to them, and see what happens. All Bitcoins sent to all the different addresses will be summed in the wallet. However, if you transfer the funds from your wallet to another wallet, Bitcoin will get the funds from selected addresses, sum them, and send them to the address.

If the sum is different than the amount of funds that you want to transfer and the Blockchain fee, the difference will be transferred to newly created address. Do not worry; this address will be automatically assigned to your wallet.

On the left side of the screenshot, you can see all the addresses that are assigned to the wallet. The transfer could not be handled by only one address, so the Bitcoin mechanism created a transfer from three addresses assigned to your wallet to the final address.

However, the sum of the funds does not match the amount of Bitcoins that should be transferred, so a new address was created (the 1ErMJj176E46irGo8vn6hrqaSpJ3yow4Sx) which is assigned to the initial wallet and the funds are back.

As you can see, you do not need to worry about the addresses that you created. After a while, there will be plenty of them, and you will probably not remember which was created for what purpose. But it does not matter, as long as you can transfer the funds from a single wallet.

## Conclusion

After this chapter, our fully working node is installed, configured, and set to TestNet. We have already gained Bitcoins for development purposes, and we are ready to start programming. In the next part, I will explain the transaction lifecycle in details and we will set up a Node.js application that will communicate with the Blockchain API.

Every integration seemed a little bit different, but after I created my very first payment integration with the blockchain technology, I realized that it was something new.

I have implemented dozens of different kinds of payment methods like PayPal, DotPay, or even credit cards. Every integration seemed a little bit different, but after I created my very first payment integration with the blockchain technology, I realized that it was something new for me; that I cannot just send the user in my application to the third party service and wait for a response with confirmation or rejected status. Using the blockchain makes me responsible for everything I may or may not have been aware of during the payment process.

In this part, I would like to present one way in which you can integrate your application, exchange system, ICO, or e-commerce shop and give your users the ability to pay with, buy or donate Bitcoins. I will start with a general overview of the transaction lifecycle and then implement this process with the JavaScript application.

## Why?

The main question is always "why". Why should you integrate your application with Bitcoin payments or even consider integrating with the Blockchain and avoid using ready-to-use services? There are many platforms that provide you with a workflow similar to PayPal's, but they all have something in common — fees.

Implementing your own solution lets you avoid paying a few percent of fees to other companies (the buyer still needs to pay the Blockchain fees, but she has to do it anyway, in every solution). If you want to keep this money in your pocket, this article is for you.

## Goal

After this part, you will know how to integrate the Bitcoin payment method with your application. The purpose you use it for is up to you. You can make a system to exchange crypto and fiat currency, make an investment platform or open an ICO; the possibilities are endless, but all of them include you accepting Bitcoin payments.

For our purposes, we will assume that we have an e-commerce business and we want to give the clients the option to pay with Bitcoins. This is the easiest example that I can imagine because it does not require us to discuss any additional business logic. During the order process, the user can choose between different payment processes including Bitcoin. We will focus only on this one.



# Transaction Lifecycle

In the usual payment system like PayPal, your shop redirects the user to a payment processor's website and sends all required information to process the transaction, like the amount, recipient, and a transaction title which often refers to the order ID.

The user is asked to choose a bank or enter a credit card number to finish the transaction, and the third party system takes care of the validation process, etc.

In the next step, the application just waits for a callback response with the information that the payment is accepted, rejected, or pending. If it is accepted, you can be pretty confident that the money is yours. The advantage of this solution is that the responsibility for the process is taken by the third party. If they give your application the information that the payment has been successful, but it was not, they need to cover the difference. On the other hand, all payment systems I was working with always had a way to withdraw money.

The Blockchain and traditional Internet payment methods are different, and all differences start at the very beginning. Any payment system that works with the Blockchain can deliver the same API ex.bitpay.com. But minus the fees.

## User Flow

Let's start from the beginning when the user is on the page where he chooses the payment method and picks Bitcoin. We cannot redirect the user to any external service because we do not know where he keeps his wallet. He could be managing his wallet on his own computer like we are. Moreover, because all payments have to be anonymous in the Blockchain, the transaction cannot provide or store any information like the sender name, receiver name, or the title. This forces us to take a different approach and recognize the payer not with the order number or ID but rather with the final address in Bitcoin.

The transaction can be presented as a short message that says: "Some amount of BTC is sent from address A to address B and signed by the owner of address A and identified by a hash number C"

Because the Blockchain is decentralized, the payment needs to be done manually, by the user like with a traditional fiat transfer from the bank account. But, the transfer should be completed in approximately 10 to 30 minutes, depending on the fee which is defined by the payer. The system presents a price without this fee, but if the funds should be transferred to the cold wallet, which may happen in ICO-projects or with bigger transactions, you need to be aware that there will be another fee which needs to be paid with the new transfer.

Then, the application needs to synchronize with the Bitcoin Blockchain and check if the transaction is included in the last block. This is the very first confirmation that it has. The last step is to gather as many confirmations from the Blockchain as we need.

## The API Client Implementation

I will show you how to connect your JavaScript application with the Blockchain and how to process the full transaction lifecycle from the very beginning, when the user requests Bitcoin payment, to the very end, when the transfer is confirmed. The library that we are going to use is called bitcore-lib and is published at <https://github.com/bitpay/bitcore-lib> by bitpay.

## Prerequisites: bitcore-lib and bitcore-explorers

We will use the node to support the development and the npm or bower package managers, depend on the operating system. At this point, I assume that you are already familiar with those technologies. If not, I strongly recommend one of the Node.js courses that are recommended by X-Team and can be found at <https://x-team.com/nodejs-resources/>.

Bitcore-lib is a package that allows you to connect to the Bitcoin node and call the API endpoints. Installation is handled by the npm package manager for linux/windows machines:

```
npm install bitcore-lib
```

or, bower for mac users:

```
bower install bitcore-lib
```

Bitcore-explorers is responsible for making bitcoin transactions and can be installed with:

```
npm install bitcore-explorers
```

and for the mac users:

```
bower install bitcore-explorers
```

## Codebase — Connect to the Bitcoin Node

When the required packages are installed, we can start to write the code. We will start by creating a connection with the Bitcoin node and its API.

First of all, we need to generate a WIF hash key (Wallet Import Format). The WIF hash number is a way of encoding a private ECDSA key to make it easier to copy. To do it, we can use the following script:

```
var bitcore = require('bitcore-lib')
var Insight = require('bitcore-explorers').Insight

var privateKey = bitcore.PrivateKey('testnet').toWIF()
console.log(privateKey)
```

The presented WIF number should be saved in the configuration. This is a private key that we will use to authorize the transaction. Now, we will use it to connect to the wallet:

```
var bitcore = require('bitcore-lib');
var Insight = require('bitcore-explorers').Insight;

var privateWIFkey =
  'cUw66adsvt6mrpkzqtPueZ6uQQsPURGcc5iK34DUnGqxsKt3FcKh';

var decodedPrivateKey =
  bitcore.PrivateKey.fromWIF(privateWIFkey)
var senderBitcoinAddress = decodedPrivateKey.toAddress()

console.log(senderBitcoinAddress)
```

And finally, we have generated a new address for the wallet that we will use for our application.

## Step 1 — Request Payment

When the user wants to pay with Bitcoin, the system should present him an address that he should send the funds to. The best approach is to present the address and the QR code, because many people use mobile applications which can scan the QR codes.

The new address should be generated for each user or even for each order, because all the information that the transaction payload contains about the sender in the Blockchain is limited only to the address that the funds are transferred from, amount and timestamp. Because this information is fully anonymous, the sender cannot be identified by the sender address. But, the Blockchain does not limit the number of addresses assigned to your wallet, so for every new buyer on your website, new address should be generated. This is the application's responsibility to save this address and create a relationship in the database between it and the order.

### Example

1. A user U1 places an order, the system generates a special address for him — A1 — and presents it.
2. A user U2 places an order, the system generates a special address for him — A2 — and presents it.
3. And so on...

If any funds are transferred and confirmed to the address A1, we will know that this is a payment which should be assigned to user U1, or somebody else wants to pay for him.

### Generate a New Wallet

Before we generate a new address for the user, we need to create a totally new Wallet for our application. We can do it from the code level, by using the REST API which is delivered with BWS at the http address <http://localhost:3232/bws/api>.

```
var Client = require('bitcore-wallet-client');

var fs = require('fs');
var BWS_INSTANCE_URL = 'http://localhost:3232/bws/api'

var client = new Client({
  baseUrl: BWS_INSTANCE_URL,
  verbose: false,
});

client.createWallet("My Wallet", "MyApplication", 2, 2,
{network: 'testnet'}, function(err, secret) {
  if (err) {
    console.log('error: ',err);
    return
  };
  // Handle err
  console.log('Wallet Created. Share this secret with your
copayers: ' + secret);
  fs.writeFileSync(wallet.dat', client.export());
});
```

We will work on this piece of code from now on. I create a Bitcoin-wallet-client object in line 6, which connects to the BWS API. Then, we can execute any call to this API using this client.

The `createWallet` method accepts a couple of parameters — a wallet name, copayer name, two random numbers, advanced options (in our case we specify that we will use TestNet), and a callback function.

As a result, we will receive a secret hash which we will use to operate our wallet. Also important — we can dump the wallet's information into one file, which is done in the 18th line. Information about the type of cryptocurrency, network, private and public keys, etc. will be saved in JSON format.

This step can and should be executed only once, and the secret key can be copied to the configuration file in your application or kept in another safe place.

## Generate a New Address

Now it is time to generate new random and unique address for our user. We will need to save this address and connect it with a user's account to be able to recognize if the payment is dedicated to a specific order and which user made the transfer. As I mentioned before, Bitcoin does not allow adding additional fields or information to the transaction. Thus we can recognize the payer only by this unique address. This address will be connected with our Wallet, which we created a moment ago, but this connection will not be visible to anybody else.

```
var Client = require('bitcore-wallet-client');

var fs = require('fs');
var BWS_INSTANCE_URL = 'http://localhost:3232/bws/api'

//secret saved from the previous exercise
var secret =
  'RMAYDCHnV7RoF9FPs8c7qVL3Eny64yFAFG57suLnwFdCjKHQ9uU5efYbDuHxgME
  oWtGQtKwRQPTbtc'

var client = new Client({
  baseUrl: BWS_INSTANCE_URL,
  verbose: false,
});

client.joinWallet(secret, "MyApplication", {}, function(err,
wallet) {
  if (err) {
    console.log('error: ', err);
    return
  };

  console.log('Joined ' + wallet.name + '!');
  fs.writeFileSync(address.dat', client.export());
```

```
client.openWallet(function(err, ret) {
  if (err) {
    console.log('error: ', err);
    return
  };
  if (ret.wallet.status == 'complete') {
    client.createAddress({}, function(err, addr){
      if (err) {
        console.log('error: ', err);
        return;
      };

      //new address
      console.log('
Return:', addr)
    });
  }
});
```

As in the previous example, I connect to the REST API by creating a Client object. Creating an address is an action that saves something to the blockchain, so we need to unblock our wallet first. The same happens when a transaction is sent.

In the 14th line, we call the `joinWallet` method, which authenticates the application with the secret key. That is the same key we generated in the previous example.

Another method, `openWallet`, in line 24, opens this Wallet, and from now on, we will be able to write to the blockchain. As the only argument, it accepts the callback function where we will create a new Address.

Line 30 is responsible for creating the new address. Thus, we will receive an Address object with the field `address`, which should be presented to the user and saved in the database for future reference. We couldn't call this method before because it would throw the error that we are not joined to any Wallet, or the Wallet is not opened.

## Step 2 — Payment (User Side)

The second step is totally on the user's side, and we cannot automate that process easily. We can just improve the way how or when we present payment information. At this point, the user scans the QR code or copies the address to his wallet and makes the payment.

When that happens, the payment is published to the Blockchain but has not been included in any block yet. It is on the pending list, and we can get information about its status. Until the block with this transaction is mined, the payment is not secure, so we cannot trust it. But, we can give the user the feedback that we noticed the payment.

To do it, we can call an API method that delivers a full list of all pending transactions. Once, the transaction is mined, it will be moved from this list to the block.

At this point, we can check if any new pending transaction for the address generated for our user has appeared. When the Steam platform allowed Bitcoin payments, they did the same. The first step was to present a unique address for the user, and if she did not close the page and transferred coins to this address, the Steam platform displayed the current payment status as Pending.

To approach the same behavior, we can list all transactions for the address, and before being mined, they will have 0 confirmations. At this point, the transaction is Pending and cannot be treated as safe. However, we can always inform the user that we immediately noticed the payment.

We will use the 'bitcore-explorers' library and Insights API. This pair of tools allows us to review the blockchain without having a wallet. This is safe. I strongly recommend you connect and open the Wallet only and only when it is necessary.

```
var explorers = require('bitcore-explorers');
var insight = new
explorers.Insight('http://localhost:3001/insight-api/',
'testnet');

const exampleAddress = 'myqfgu5fZKgPBiqv7gghBPETc4yEnCdocU'

insight.requestGet('txs/?address=' + exampleAddress,
function(err, response) {
  console.log(response.body)
});
```



In this piece of code, I used an address where I transferred some amount of coins, but they have not been mined yet. Therefore, I can see that the txs method returns a list of transactions with only one element which has no confirmations.

## Step 3 — Get the Transaction From the First Block

There are two ways of getting information about the transaction now — the first one is to save the transaction id from the pending list and track its status with a cronjob or another system, or listen to the last published block and get a list of transactions included in it. Because the second approach requires us to use new API methods, I'll stick to it. It does not mean it is better, just different.

### Get the Latest Block

This is one of the approaches for getting information about the current payment status. If you already got the transaction ID, then you can omit this step and go directly to step four.

But sometimes, it happens that you are not able to observe all addresses. When you run a really successful business, you probably open hundreds of addresses every day, and checking pending transactions for all of them may be a time- and resource-consuming process. That's why I prefer to observe the latest block in the blockchain. The block includes a list of mined transactions. We will review those transactions and compare receivers with our saved addresses. If we find any of the addresses in our database, we have a match; we can save the transaction ID and just wait for confirmations.

To get information about a block, we can use the block method in the Insight's API. This will return a full set of data including a list of transactions.

```
var explorers = require('bitcore-explorers');
var insight = new
explorers.Insight('http://localhost:3001/insight-api/',
'testnet');

const exampleBlock =
'000000000000007435e1e4c72e17ae47fbdd63132950468d3dbd5d732e7fc32d
8'

insight.requestGet('block/' + exampleBlock, function(err,
response) {
  console.log(response.body)
});
```

Unfortunately, the list of transactions includes only the IDs, so we need to go through all those IDs and get the transaction data from another method called tx. I prepared the code:

```
var explorers = require('bitcore-explorers');
var insight = new
explorers.Insight('http://localhost:3001/insight-api/',
'testnet');

const exampleBlock =
'00000000000000592e07543f0da2c21a93095827c4dc03f178da1557057b9795
f'

insight.requestGet('block/' + exampleBlock, function(err,
response) {
  JSON.parse(response.body).tx.forEach((transaction) => {
    insight.requestGet('tx/' + transaction, function(err,
response) {
      const transactionObject = JSON.parse(response.body)
      console.log(transactionObject)
    });
  })
});
```

As you can see, the latest block has two transactions, but the receivers are pretty well hidden. This is because they are stored in the vout section. You can notice that the response includes two sections — vin (value in), and vout (value out). Those are arrays because Bitcoin does not make a transfer from one address to another, but from a bunch of addresses in the same wallet to whatever is specified.

**TIP:** You can create your own transaction and send it to how many receivers you want to. To do so, you need to create a raw transaction, sign it with your private key, and publish to the blockchain. This operation requires three different steps, or four if you want to verify the generated hash. In this case, you can compose your own transaction. You need to specify which addresses you want to use to send money from, and where exactly it will be sent — you can transfer funds to one or many addresses. The funny thing here is that there could be 5 recipient addresses, all of them the same. Bitcoin cannot stop you.

The code below lists all receiver addresses with the amount of funds that were transferred to it. This one list is included into one specific transaction:

```
var explorers = require('bitcore-explorers');
var insight = new
explorers.Insight('http://localhost:3001/insight-api/',
'testnet');

const exampleBlock =
'000000000000000592e07543f0da2c21a93095827c4dc03f178da1557057b9795
f'

insight.requestGet('block/' + exampleBlock, function(err,
response) {
  JSON.parse(response.body).tx.forEach((transaction) => {
    insight.requestGet('tx/' + transaction, function(err,
response) {
      const transactionObject = JSON.parse(response.body)
      transactionObject.vout.forEach((vout) => {
        console.log('TRANSFER ' + vout.value + ' BTC, to ' +
vout.scriptPubKey.addresses)
      })
    });
  });
});
```

## Get the Block Hash

Now we can get information about the block, but we need to know the block hash, which is pretty hard to guess. That is why blocks also have order numbers. My approach is to save the number of the last block that I checked to the database and increase it every time I run the cronjob. In this case, I can check transactions in many blocks, one after another, not only in the last one.

The API provides a method that returns a block hash for a specific order number. It is called

`block-index` :

```
var explorers = require('bitcore-explorers');
var insight = new
explorers.Insight('http://localhost:3001/insight-api/',
'testnet');

const exampleBlock = 1261415

insight.requestGet('block-index/' + exampleBlock, function(err,
response) {
  console.log('BLOCK HASH: ' +
JSON.parse(response.body).blockHash)
});
```

After the transaction is mined in the block, it has the very first confirmation and can be treated with limited trust. I call it limited trust because there is a slight chance that somebody can publish a block that should not be accepted by the Blockchain, but somehow our node accepted it. Some of those blocks may be dropped when the next is mined. Those blocks which have not been accepted after the next block was mined are called orphaned blocks. This is why one confirmation may not be enough to trust the payment.

## Step 4 — Confirm the Transaction

The last step is to gather as many confirmations from the blockchain as we need to mark the transaction as secured or trusted. After the transaction is mined for the first time and added to the block, every next block on the stack will add one confirmation to the transaction. The confirmation number is nothing more than a piece of information about how many blocks have been mined after the block with this transaction.

The Blockchain algorithm makes the mining process extremely difficult and mining few blocks in a row by the same miner is almost impossible or impossible. The level of trust can be considered as a good balance between the waiting time for the final confirmation and the certainty level. Because each block is mined in approximately 10 minutes, the more confirmations we need to trust the transaction the more time we need to wait for it. I prefer to use the following thresholds:

- 1 confirmation (10 minutes) for small transactions, max 10-50 USD, frequent transactions or from trusted users. This limits the risk, and I keep the level of risk to a low level, where nobody will be seriously hurt if the Bitcoins are gone. This threshold can be used for food or game ordering (The Steam platform used a single-confirmation algorithm with BitPay).
- 3 confirmations (30 minutes) for usual transactions with bigger value. It is quite unusual that one miner will confirm 3 blocks in a row, but that may happen. This kind of threshold is often used with the internet exchange systems that allow purchasing more than 2000 USD.
- 6 confirmations (60 minutes) for big transactions, mostly used for investing process in ICO or other Blockchain projects, where the users operate with several tens of thousands of dollars. In this case, time is not as important as security and confidence are.

## Get Transaction Data

To get information about received confirmation number, we need to know the transaction ID. The `tx` method is meant to return complete information about the transaction:

```
var explorers = require('bitcore-explorers');
var insight = new
explorers.Insight('http://localhost:3001/insight-api/',
'testnet');

const transactionId =
'84ab8aa62753a29f11adc15b6415c6fc58799e7b6f74b80bc31db2a72f6dd10
5'

insight.requestGet('tx/' + transactionId, function(err,
response) {
  console.log('number of confirmations for ' + transactionId + '
is ' + JSON.parse(response.body).confirmations)
});
```

After the number of confirmations meets our expectations, we can mark the transaction as done.

## Step 5 (Extra) — Secure Funds

I did not specify this step at the beginning of this chapter because it is not necessary and the process of the payment confirmation ends with the previous step. However, in some cases, moving funds to the cold wallet or to some other wallet that is not connected to the application may be a good idea for the projects that operate with bigger numbers, like ICOs, investing platforms, or loans systems.

When the investor transfers 200 000 USD to our system in Bitcoins, we do not want to keep it unsecured for too long. Why do I claim that that money is unsecured when everybody is talking about the Blockchain's high security? My previous projects and experience taught me that somebody will always try to break your application. It does not matter if you have a billion dollar wallet or you create a Facebook contest with a 50 USD prize for fuel. If there is any money at stake, there will always be some attempt to take it.

People like to think that a hacker's job is strictly related to hacking the software and breaking all technical barriers. But it is not. If the code is written right, it is the hardest part to break, thus the attack will be launched somewhere else. And the weakest point is the human factor.

### Send a Transfer

After we have received the funds, but before we have collected all the confirmations, we can transfer the funds to a secured cold wallet or at least to some specific address, which is not connected directly to the application or whose secret hash is not saved in the database or the configuration file.

I must admit, sending a transfer with `bitcore-wallet-client` is a little bit overcomplicated:

```
var Client = require('bitcore-wallet-client')

var fs = require('fs')
const BWS_INSTANCE_URL = 'http://localhost:3232/bws/api'

//secret saved from the previous exercise
const secret =
  'RMAYDCHnV7RoF9FPs8c7qVL3Eny64yFAFG57suLnwFdCjKHQ9uU5efYbDuHxgME
  oWtGQtKwRQPTbtc'
const receiverAddress = ''
```

```
const amountOfBtc = 1

var client = new Client({
  baseUrl: BWS_INSTANCE_URL,
  verbose: false,
})

client.joinWallet(secret, "MyApplication", {}, function (err,
wallet) {
  client.openWallet(function (err, ret) {
    if (ret.wallet.status == 'complete') {
      client.createTxProposal(
        {
          outputs: [{
            toAddress: receiverAddress,
            amount: amountOfBtc,
          }],
          feePerKb: 10000,
          excludeUnconfirmedUtxos: false,
        },
        function (err, tx) {
          wallet.publishTxProposal({txp: tx}, function (err) {
            wallet.getTxProposals({}, function (err, txps) {
              var txp = txps[0]
              wallet.signTxProposal(txp, function (err, txs){
                wallet.broadcastTxProposal(txs, displayend)
              })
            })
          })
        })
      })
    }
  })
})
})
```

I omitted all the parts with catching errors so that you can find only the success line in the code. Let's go from the top and discuss what is going on:

- `joinWallet` — we need to connect with the Wallet. This method authorizes us to use it, but it is still not opened.

- `openWallet` — from now on, we will be able to use the Wallet. We need to open it to send the transaction.
- `createTxProposal` — the blockchain keeps the security at the highest level, so we do not create a transaction, but a transaction proposal. This method returns an object from the specified array. We do not need to enter all the information, like a full list of addresses that the funds are sent from. This will be automatically generated.
- `publishTxProposal` — the transaction proposal is published to the blockchain, but it cannot be mined until we sign it with the private key.
- `signTxProposal` — we do not need to provide the private key because we have already opened the wallet, so the software knows that we operate on this specific wallet, but we need to pass a specific list of transaction IDs that we want to sign.
- `broadcastTxProposal` — after we call this method, the transaction is broadcasted to the internet and can finally be mined.

**TIP:** If you need to keep some funds for returns, I recommend you check the balance and create an additional address that the application can manage. However, the main part of the money should be secured.

## Conclusion

At this point, you know exactly how to handle Bitcoin payments, how to confirm them and secure the funds. Now you can create a payments system with the Bitcoin node. In the next chapter, I will present another Blockchain cryptocurrency — Ethereum. It has some differences in comparison to Bitcoin.



The CryptoKitties is a Smart Contract that allows users to collect or buy and breed digital cats.

If you would like to become a Blockchain developer, sooner or later, you will work with Ethereum. It is the second-largest cryptocurrency at the moment. According to coinmarketcap, its market capital is half the size of Bitcoin and twice the size of Ripple, the third-largest cryptocurrency. Ethereum, like Bitcoin, is a Blockchain solution that provides sharing coins' features. So, why did it become so popular? All the small differences between Ethereum and the first cryptocurrency make it so popular — the smart contracts, quickly mined blocks, and low fees.

## What is Ethereum?

Like Bitcoin, Ethereum is a Blockchain solution. The general functional principle is the same — currency owners can exchange the funds by sending the transactions which are broadcasted with the decentralized Blockchain. But a couple of differences make Ethereum so special

## Blocks are Mined More Often

Every block is mined in approximately 10 minutes in Bitcoin. In Ethereum, it happens 4-5 times per minute. Thus, making microtransactions, ordering pizza, or paying for goods does not require one to wait for 10 minutes until the transaction receives the first confirmation.

## Smaller Blocks

In Bitcoin, the block size is currently specified at 1MB. The Ethereum block size is calculated in a totally different way and depends on the complexity of transactions included in this block. Every block can have a complexity of around 8,000,029 Gas. Every transaction has its own complexity called the Gas and calculated from the code, smart contracts, and information that the transaction includes. Let's say that a single, basic transaction has a complexity of 21,000. There may be 380 simple transactions in the block (Bitcoin can contain about 2000). However, if any of those transactions included in this block is more complex and requires more Gas, the total number of transactions will be lower.

## Smart Contracts

The Smart Contracts are what makes Ethereum so unique. Imagine an ordinary contract in real life. It says that if a person A delivers some goods, products or work, the other contracting party B will pay. In the programmers' world, we would say that the payment will be done if all conditions are met. In Ethereum, we can write such a contract in the primitive scripting language called Solidity, which is executed by Ethereum Virtual Machine.

## Mining Rewards

In Bitcoin, the Miner is rewarded by 12.5 BTC now, and this number will decrease every year. Ethereum rewards every mined block by 5 ETH, and this number is constant — that adds up to around 11,5 million ETH per year. Moreover, the Miner is rewarded with the Gas from contracts (like the fee in Bitcoins), and references to 2 recent uncles with  $\frac{1}{32}$  of a block reward — 0.15625 ETH per uncle.

## Smart Contracts

The Smart Contracts are the most important feature provided by Ethereum. What is Smart Contract? In a real world, the contracts between parties are executed after all conditions have been met. After that, the service provider receives payment. Ethereum provides a basic programming language in which you can write your own conditions. If all of them are fulfilled, the contract will automatically execute.

The difference between Bitcoin and Ethereum is that Ethereum supports more sophisticated conditions and methods in Smart Contracts than Bitcoin. The options are limited only by the imagination, which can be extremely creative. In late 2017, one of the Smart Contracts blocked the Blockchain for a few days and was generating about 10% of transactions in Ethereum for a couple of weeks. What was it about? Virtual Cats.

The CryptoKitties is a Smart Contract that allows users to collect or buy and breed digital cats. Each Cat is one-of-a-kind and 100% owned until you sell it. And the price is double the price that you paid for the cat. It goes down until somebody will buy the cat. Thus, Ethereum allows users to write games with the Smart Contracts.

## Solidity

Solidity is a contract-oriented programming language for implementing Smart Contracts. It was influenced by C++, Python, and JavaScript and is designed to target the Ethereum Virtual Machine.

Solidity is quite similar to JavaScript in my opinion. Easy to learn and execute. But there is one thing that is really, really hard during the development process — testing. Compared to other applications I used to develop, here, maintenance is impossible. Therefore, top-level quality is a must-have. If a Smart Contract is published to the Blockchain, it can never be changed. Every change requires a new Smart Contract. If you find a bug in your code, it is a huge decision to make — leave it or move all users to a new Smart Contract.

## Digital Tokens

You have probably heard about the Digital Tokens in Ethereum. It is a special kind of the Smart Contract that allows users to exchange the funds/goods/cats other than Ethereum, but using Ethereum's Blockchain. It is commonly used in ICO projects (Initial Coin Offerings). The ICO is something between the Kickstarter and a stock exchange. Already existing companies or newly opened startups are looking to gather funds for the development. In order to do that they offer investors Digital Tokens which are stored in the Blockchain. Once the tokens are withdrawn, they can be exchanged, bought or sold by the owners. The price is or may be related to the company value.

## How to Run the Ethereum Geth Node

The node that I would recommend for connecting with Ethereum is called Geth. It is the command line interface with a full Ethereum node implemented in Go. Geth is an open source project and can be found in the GitHub repository at <https://github.com/ethereum>.

The Ethereum software is divided into three parts responsible for different parts of the work:

- the Ethereum Virtual Machine, responsible for computations,
- the Swarm, responsible for peer-to-peer file sharing,
- Whisper, a messaging protocol that allows sending messages to other nodes.

## Geth Installation

Because the Geth software is available on the windows, mac os, and linux platforms, I will stick to the Ubuntu environment. If you're using another, please see the instructions available at <https://github.com/ethereum/go-ethereum/wiki/Building-Ethereum>.

Geth can be obtained in two ways — compiled from the source or installed from the PPA. PPA is easier and does not require one to install additional software or the Go language.

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install ethereum
```

After executing this code on the command line, a new geth application should be available. You can check with the `geth -h`.

## Start the Geth Node

Running the geth software is not as obvious as it may look like. When you execute geth on the command line, it will immediately start synchronizing with the Blockchain. In our case, we will need to enable a couple of features that are not included with the standard command:

```
geth --rpc --rpcapi="db,eth,net,web3,personal,web3"
```

This will run `geth` and enable the RPC API, including selected features that will allow our application to download the data we need. Like Bitcoin, Ethereum, too, has a TestNet or three. The Ropsten for testing Proof Of Work concept, Kovan — for Parity purposes and Rinkeby for geth. We will use the last one (<https://rinkeby.etherscan.io/>).

Running geth in TestNet mode requires only one additional flag: `--rinkeby` :

```
geth --rinkeby --rpc --rpcapi="db,eth,net,web3,personal,web3" --
rpccorsdomain "http://localhost:3000"
```

Now, the API is broadcasted at the address <http://127.0.0.1:8545>. The `rpccorsdomain` flag will allow us to connect to the RPC API from this specific domain. I set this to the localhost:3000 because it is the default address for React's boilerplate.

**WARNING:** In this case, the RPC API is enabled, and the communication between your application and the Ethereum's node should be well secured. At the very least, it should not be available from the Internet otherwise it may get hacked.

## Geth's Console

Geth as a tool delivers a console that allows us to perform some actions on the Ethereum's Blockchain, like managing accounts, addresses, reviewing blocks or transactions. The majority of the commands available in this console are the same methods that we will use in the API.

Geth's console is included as an `attach` argument. To connect to it, we need two things — a running Ethereum node and a specified ipc file, which is an inter-process file. We will need to use the file from the rinkeby folder:

```
geth attach ipc:~/.ethereum/rinkeby/geth.ipc
```

Afterwards, we can check if everything works correctly. Try to run the command `eth.getBlock()` within the console. It should return the highest block in the chain.

**HINT:** If the highest block in the chain is 0, please double-check whether the node is synchronized with the Blockchain. You can do it by checking the log of the running node, or by typing `eth.syncing` in the console. This command displays all information about current state of the node.

## Create an Account

Our next step is creating an Account. The Account is similar to the Wallet in Bitcoin and is mandatory to operate on the assets. Geth delivers a special argument that allows for account management. To create a new account you can execute:

```
geth --rinkeby account new
```

This command will ask you for the passphrase that protects your account and will return an Ethereum address that will be used for further work.

Now, we can check currently opened accounts on our node by calling:

```
geth --rinkeby account list
```

The presented list has two columns — the address and keystone file, which can be used as a backup file for the account. If we lose the node or would like to use another piece of software to manage the account, e.g. a web interface. In that case, we can generate a private key to the account or use the keystone file.

It is also possible to check the list of accounts in the Attach console. The `eth.accounts` method will present all created addresses.

# Transaction Lifecycle

As in the previous part, I would like to present a full transaction lifecycle. Let's go back to our previous example with the e-commerce website where the user can buy goods and pay with the fiat currency and already implemented Bitcoin. Now, we would like to add a new case — the Ethereum payment method.

Even with the huge list of differences between them, Bitcoin and Ethereum are based on the same cryptocurrency principles — anonymity, transparency and information sharing. So, the transaction lifecycle will look exactly the same as in part #4 — the application will generate an address for the order, the user will send funds to this address, and we will watch the Blockchain for the latest blocks and list all transactions inside that will be sent to this address. After that, we will just decide how many confirmations we need to trust the transaction.

## Step #0 — Connect to the Geth

Our first step is to connect to the Geth node from the JavaScript application. Fortunately, geth delivers an RPC API endpoint (a real-time connection) which can be handled by the web3.js library — <https://github.com/ethereum/web3.js>. To install the web3 library, we can use npm package manager:

```
npm install web3
```

Now we should be able to connect to the node and download basic information. Let's start with the list of accounts:

```
var Web3 = require('web3');  
var web3 = new Web3(new  
Web3.providers.HttpProvider("http://localhost:8545"));  
console.log(web3.eth.accounts)
```

As you can see, the web3 packager delivers commands really similar to the ones available in the console.

I know that the code does not look pretty, and the configuration should be moved to the environment variables, etc., but in this tutorial, I would like to focus exclusively on the connection between a JavaScript application and the Ethereum node. If you would like to polish your React's or JavaScript's skills, I can recommend the resources shared at <https://x-team.com/javascript-resources/>.

## Step #1 — Generate a New Address

Let's move to the e-commerce shop page and imagine how the user orders items. She adds new products to the order, and when she is done and ready, she enters the delivery data and the method of payment. We would like to add a new method here — the Ethereum. After the user accepts this method, a new address should be generated and presented to the user — preferably as a QR code and a piece of text that she can copy-paste to her wallet. It should be displayed in both ways because some users use mobile applications to pay with Ethereum and others use web applications or even geth's console.

At this point, Ethereum works a little bit different than Bitcoin, because it is not possible to add more than one address to the account as each account needs to be secured with a passphrase. Moreover, that is why Ether's transactions only have one sender and one receiver. The funds are not kept in one wallet. However, the transaction fees are pretty low in comparison to Bitcoin and the confirmation time is short, so that is not a problem.

Thus, we will need to create a new account for each user and keep a randomly generated passphrase in the database. I would recommend automating the process of transferring funds, after they have been confirmed on each wallet, to some other address that will not have its passphrase saved in the database. I will explain how to do it in step #5.

To generate a new address from the JavaScript code, we can use the method available in the `personal` namespace, `web3.personal.newAccount`, which will accept one parameter with the passphrase. This passphrase should be encrypted and saved with the address for further purposes, otherwise, you will lose access to this account.

```
var newAccount = web3.personal.newAccount('testpassphrase')
console.log(newAccount)
```

A new address will be generated, which can be presented to the user.

## Step #2 — Wait for Payment

Now, we are waiting for the user's action. She needs to copy-paste or scan the address and transfer funds to our wallet.

## Step #3 — Get the Transaction from the Latest Block

In this step, we will watch the last published blocks and compare the transaction receivers with our list of saved addresses. I chose this method because it is much quicker than the alternatives I took into consideration. We could watch all the addresses and check the current balance or a list of transactions assigned to specific addresses, but both methods will generate an enormous number of API calls because, after some time, I expect to have hundreds or thousands of newly generated addresses saved in the database I'll be checking against.

So, I prefer to get the last published block and compare the transactions' receivers with the list of addresses generated by our application. We can use `getBlock` method, which accepts one parameter — the block hash. It does not have a default parameter set to the latest block, but we can use the word "latest" which will be properly resolved by the interpreter.

```
web3.eth.getBlock("latest")
```

This method will return all the information about the block, including the hash, number, size, timestamp, and list of transaction ids. Unfortunately, that's only ids for transactions. To get complete information about each transaction, we will need to call another method, `getTransaction`, which will return the transaction object.

To automate this process a little bit and get only receivers from transactions we can use the following code:

```
var block = web3.eth.getBlock('1644382')
block.transactions.forEach((transaction) => {
  console.log(web3.eth.getTransaction(transaction).to)
})
```

Now, if the receiver is found in our database, we save the first confirmation for the payment. In the example, I passed the block number into the `getBlock` method, because that is also acceptable besides "latest" or the hash. This is helpful because Ethereum's Blockchain publishes blocks every 12 seconds, so if you would like to run a cronjob, which will be executed every minute, you will need to check at least the last five blocks. My solution for that case is to save the number of the last checked block and increase it when the next cronjob executes.

## Step #4 — Gather Confirmations



Ethereum does not return a confirmation number as Bitcoin does. Each new block that is published on the ledger can be treated as a new confirmation for already mined transactions. In this case, we can use two different approaches. Let's assume that we need five confirmations — the transaction will be secured in about 1 minute, and this is just enough to ensure that we are not in the uncle tree.

The first approach is watching the block like in the previous step. However, not the latest block or blocks, but rather the block which is five confirmations behind the latest block which we checked or the last published block.

```
var block = web3.eth.getBlock(web3.eth.blockNumber - 5)

block.transactions.forEach((transaction) => {
  console.log(web3.eth.getTransaction(transaction).to)
})
```

The second approach includes listing all pending transactions from our database and getting the block number from the `getTransaction` method.

```
var transactionIdFromDB =
'0x2b22e4604bbaa4e3570c9d08121d224790561bfa50f203e1dcc43cc9c90da
758';
var transaction = web3.eth.getTransaction(transactionIdFromDB)

if ((transaction.blockNumber - web3.eth.blockNumber) > 5) {
  //confirm transaction
}
```

## Step #5 — Transferring Funds to an External Wallet

When the system notices that the funds have been transferred, I strongly recommend moving them to an external account which is not connected to the application. Just in case. Some applications require you to keep coins for pay-back, so 90% of funds can be transferred to a safe wallet and the rest to the pay-back wallet, which can be connected to the application.

The Web3 package delivers an easy-to-use method for transferring funds from one account to another — `sendTransaction`. Unfortunately, this method does not accept the passphrase which is required to send the transaction. Before we try to make a transfer, we need to unlock the account and go through authorization with the passphrase.

To do it, we need to use the `unlockAccount` method in the `personal` namespace:

```
web3.personal.unlockAccount('0x63e707fce38c59267838ff91090b5616a  
ae7e26b', '1234', 3)
```

This method accepts three arguments — the account, the passphrase, and the number of seconds after which the account will be locked again. It returns a boolean value that tells if the operation succeeded or not.

After that, we are able to send the transfer using the method `sendTransfer` :

```
var transactionIdFromDB =
'0xc810577c36e4f5cd106b7040759554b11b4c83bb8acb730fb989b294aae99
f17';
var transaction = web3.eth.getTransaction(transactionIdFromDB)
var safeAccountAddress =
'0x63e707fce38c51132317497192734917239419bb'

var transactionObject = {
  from: transaction.to,
  to: safeAccountAddress,
  value: transaction.value
}

web3.eth.estimateGas(transactionObject, function(error, gas) {
  web3.eth.getGasPrice(function (error, gasPrice) {
    var gasPrice = Number(gasPrice);
    var transactionFee = gasPrice * gas;

    transactionObject.value = -(transactionObject.value -
transactionFee);
    web3.eth.sendTransaction(transactionObject);
  })
});
```

In the code, I create a new transaction object which defines a transfer from the account that received funds, to the safe account with the value of transfer minus the calculated fee that is required to send the transaction. The fee is calculated by the formula  $\text{GasPrice} * \text{RequiredGas}$ .

## Conclusion

In this chapter, we have learned about Ethereum, which is both similar to and different from Bitcoin. We have also been able to successfully implement a full transaction lifecycle for our e-commerce shop. In the next part, I will show you how to use and write Smart Contracts.



I must admit, you can learn by hacking.

Smart Contracts are one of the hottest things that are currently available on the Blockchains. Bitcoin enables you to implement only a simple Smart Contract with basic conditions, but Ethereum delivers a fully operative programming language that you can use to manage a business.

In this chapter, I would like to explain what a Smart Contract is, how it works, how to write one, and where you can find ready to use solutions. I'm going to assume that you are already a programmer who knows JavaScript and has no problems with reading similar code. I will skip some code explanations like private/public modifiers or string typing.

## What is a Smart Contract

In the real world, we sign many contracts — when we set up cable television, buy a new phone or even a house or flat. Every one of those contracts has something in common — we pay to get something, or we give something to be paid. But, there is a huge group of contracts — I would call them agreements — we are so familiar with that we do not even notice that we are talking about them.

One of my favorite examples of simple cash distribution is when you give your friend 10 USD and ask him to share it with five colleagues evenly. He says 'OK', and you expect that everybody will receive an equal 2 USD. That is an agreement between you and the broker.

How hard could it be? As always, something may go wrong. One of the receivers can be favored by the broker and receive 3 USD, so another one receives only 1 USD. You did not agree to that.

Smart Contracts solve this kind of problems (and more) by automation. The contract's creator can specify all the rules in the contract, and nobody can change it — literally nobody. If you want to change any condition, a new Smart Contract needs to be created.

In our specific case, the Smart Contract works as the broker between you and all interested parties. Every transaction made to the contract will be evenly distributed between saved addresses (in our case 5). Thus, if you create a contract with 5 saved addresses, or if you send those addresses as additional information to the contract, and transfer 10 ETH to this Smart Contract, it will immediately transfer 2 ETH to each receiver.

Because this guide is intended for developers, here is an example we can find in the real world. Some applications that help during the software development process, like IDEs or a database management systems, require you to buy annual access to it. Every year, I have to

sign some kind of a contract between me and the software company to extend the license for another year.

So, the definition of this contract has a couple of conditions: If I do not have a license, or if its expiry date is close or passed, and if I agree with the license agreement, and I have paid, either I will receive a new one or my current license will be extended for another year. This looks pretty simple to a developer — just a couple of `if` conditions. And it is so simple. When you think like a programmer, most of the questions are of a true/false nature, and so are the contracts.

Smart Contracts in Ethereum work in the same way. Imagine that you can accept ETH payments for the software license and automate the process of extending the due date for next year.

## How Smart Contracts Work

In Ethereum, there are two different types of accounts — the one you already know and use, that is a personal account, and the Smart Contract account which, as the personal one, has its own address and balance. But the Smart Contract does not have a private key that allows you to control the funds on it. Whatever happens in this account is controlled by the code inside. Moreover, this code cannot be changed — ever. So, if you haven't noticed a bug in your Smart Contract, and it is published, you cannot just fix it and deploy. A new Smart Contract needs to be created. Smart Contracts are immutable.

Smart Contracts can do the same things a personal account can — they can send or receive Ethers or some other virtual tokens or information. But remember, every transaction in Ethereum requires Gas to be mined, and the Gas costs Ethers. So, each Smart contract transaction requires Ethers (which is paid not by the Smart Contract itself but rather by the users).

One interesting property of Smart Contracts is that they can communicate with each other and interact. However, those actions and interactions are usually executed on the Blockchain. However, we may require that some of the information necessary to resolve some conditions in a Smart Contract are retrieved from the Internet.

Smart Contracts are designed to operate on data provided by the blockchain and usually do neither communicate nor allow communication with the external world. However, one type of Smart Contracts allows the use of external APIs — the Oracle Smart Contract.

## Digital Tokens

Digital Tokens are something I would call a cryptocurrency inside another cryptocurrency. Many ICO projects (Initial Coin Offering) offer tokens in exchange for funds for their businesses. An ICO is something between a stock exchange and Kickstarter. Newly created or existing businesses are looking for opportunities for raising money to start operating or expand their operations. In exchange, they offer tokens. The tokens can be shared, exchanged, sold or bought as stock papers. Usually, the tokens start with an initial value, e.g., 1 USD per token, and after the company is developed, the token value can rise or fall.

Writing token contracts is hard, because there are many edge cases to handle, and there is no room for mistakes. Parity is one of the biggest software companies that provide solutions for Smart Contracts. They have published a contract called ERC20, which is widely used to share and distribute digital tokens. A list of those tokens can be found at <https://etherscan.io/tokens>.

## Getting Started with Smart Contracts

### Solidity

Solidity is a statically typed programming language specifically designed to write Smart Contracts that can run on the Ethereum Virtual Machine. It is quite similar to JavaScript to make it easier to learn for web developers.

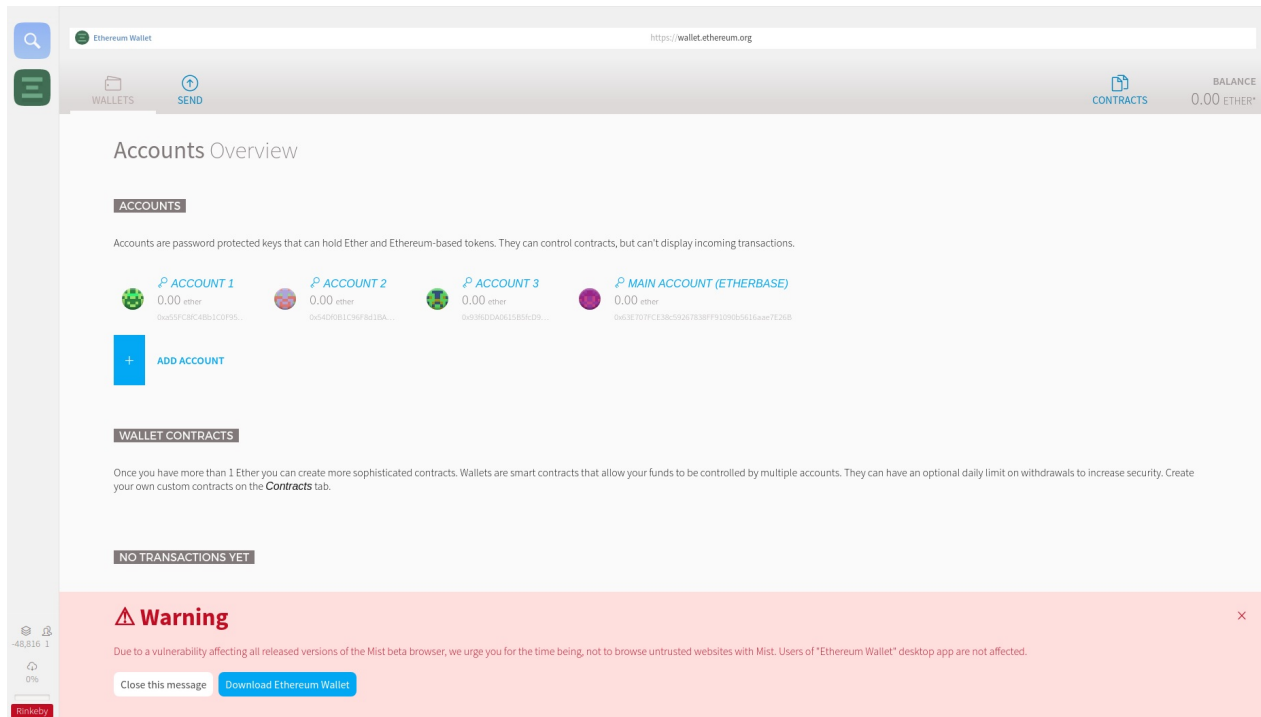
### Mist

The Mist browser is the tool of choice to browse and use Dapps founded by Ethereum. We will need this piece of software to publish our first Smart Contract. To download Mist, we need to go to the release page at <https://github.com/ethereum/mist/releases> and download the latest version available for our operating system. I use Ubuntu, so that'll be Mist 0.9.3 (at the time of writing) in a deb-package.

**TIP:** At the release page you will find two applications — the Ethereum Wallet, which is designed for personal usage, and Mist, which includes the Ethereum Wallet and additional features, like publishing Smart Contracts. Please remember to download Mist.

When you open Mist for the first time, it will ask you which Ethereum network you would like to use — let's stick to Rinkeby TestNet, because it will synchronize with the Blockchain a lot faster than LiveNet, and we will avoid losing Ethers during development. To switch networks,

you need to go to the top menu, select Develop, then Network, and chose the network you are interested in. You can find the currently selected network in the bottom left corner of the application.



## Remix IDE

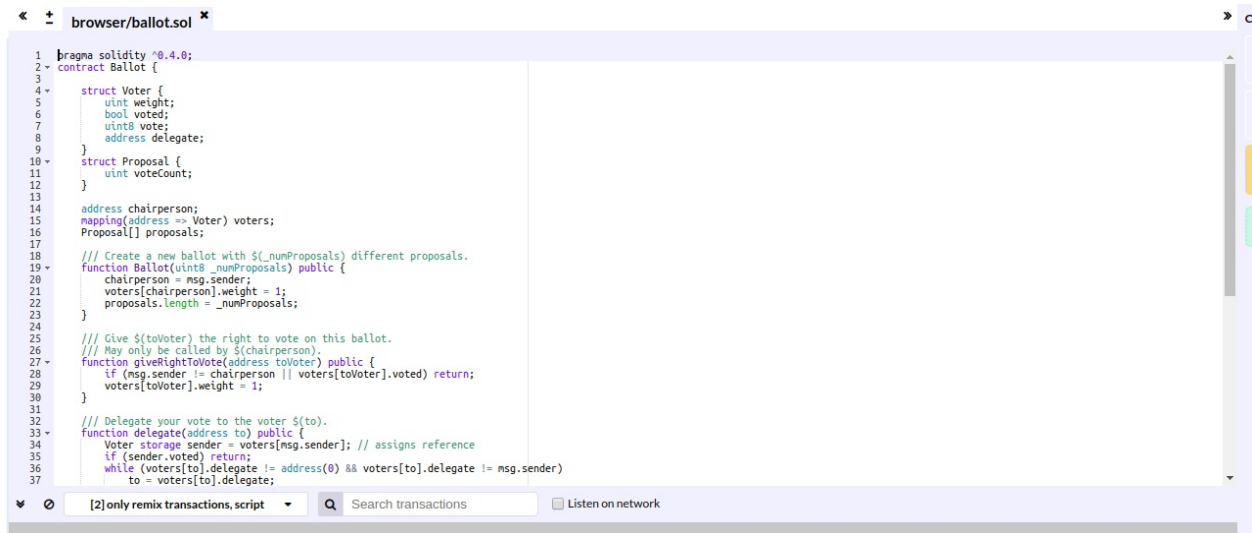
Writing Smart Contracts can be much easier with a proper IDE (Integrated Development Environment). Solidity documentation strongly recommends using the Remix IDE, which is available online at <https://remix.ethereum.org/>. Also, Remix can be installed on a local machine. Full instructions are available at <https://solidity.readthedocs.io/en/develop/installing-solidity.html#remix>.

**TIP:** Ethereum also offers plugins for other development tools. A full list of those can be found at <http://solidity.readthedocs.io/en/develop/>.

This IDE provides a bunch of useful tools, such as a compiler, analysis tool or debugger. We can easily track what happens in the contract during every test without publishing it to the blockchain. When you open the web page for the first time, a Smart Contract example called Ballot will be presented to you. It implements a voting system.

**Exercise:** Please review the Ballot Smart Contract. Do you recognize some structures, the constructor, or methods? Take a look at the syntax and the methods names, and explain how this Smart Contract works.





## Your first Smart Contract

```

pragma solidity ^0.4.0;
contract Payroll {
    function Payroll() payable{

    }

    function () payable{

    }

    function withdraw() payable {

    }
}

```

## Constructor Function

A constructor function is called only once when the Contract is Created. It sets up the prerequisites for the Smart Contract, like setting the initial token amount, etc.

## Fallback Function

Whenever somebody sends Ethers to your Smart Contracts without any additional information and without calling any specific method in the Smart Contracts, this method will be called. You can return funds to the sender, use a default method or distribute them as you wish.

## Withdraw Function

This function is used to make a transfer and collect goods, Ethers or tokens from the Smart Contract. The Smart Contract does not need to send new transaction right away when you push Ethers into it. It can work as a database. Consider digital tokens and how they work. You can buy tokens by sending Ethers to the Smart Contract and exchange them back by calling the withdraw method.

Without this method, withdrawing Ethers from a Contract is impossible, because it does not have a private key that can be used to make a transfer.

## Payable Keyword

You may notice some functions have an optional keyword `payable` after the name. If you add this keyword, the Smart Contract will be able to operate with Ethers. You can attach Ethers to the Smart Contract, send a transfer with Ethers to it or withdraw Ethers. Without this keyword, Mist would throw an exception if you tried to create a non-payable Smart Contract with attached Ethers.

By default, Contracts do not accept any money. The `payable` keyword should be added after every function that should be able to operate with money.

## Business Logic

Let me explain the business logic from this example. I have created an example Smart Contract that allows you, as the owner, to manage the company's salaries, which will be distributed by Ethereum. The general idea is to make the Smart Contract owner able to send only one transfer to the Smart Contract and then, every employee, should be able to withdraw their salary.

It works a little bit different than with fiat currency because when you receive a traditional bank transfer with salary, you usually do not need to ask for it once again. If we try to move our example to traditional banking, we end up with one account where the company sends a

big transfer with all the salaries, and you need to go to the bank and ask them to pay you from this account when you need your money. Moreover, you will need to pay the fee. Sounds awful, huh?

But it does make sense when you take into consideration a couple of factors and possibilities that Smart Contracts provide. Let's assume that we run a website strictly focused on IT, programming languages, or even the Blockchain. Everyone can become an author on this service and each article is paid an equal \$300 in Ethers. So far, it is a real-life example. But now, let me demonstrate how a Smart Contract would handle the authors' payments.

As the owner of the Smart Contracts and the website, we should be able to add a new author's address to the database of earned payments. All saved addresses should be able to withdraw their payments whenever they want to.

```
pragma solidity ^0.4.0;

contract Payroll {
    uint totalReceived = 0;
    address owner;
    mapping (address => uint) public salaryAmount;
    mapping (address => uint) public withdrawnSalary;
    function Payroll() payable public {
        updateTotalReceived();
        owner = msg.sender;
    }

    function () payable public {
        updateTotalReceived();
    }

    function updateTotalReceived() internal {
        totalReceived += msg.value;
    }

    function addAddress(address _salaryAddress, uint _salary)
    isOwner public {
        if (msg.sender == owner) {
            salaryAmount[_salaryAddress] = _salary;
        }
    }
}
```

```
modifier isOwner() {
    require(msg.sender == owner);
    _;
}

modifier canWithdraw() {
    require(salaryAmount[msg.sender] > 0);
    _;
}

function withdraw() canWithdraw public {
    uint amountPaid = withdrawnSalary[msg.sender];
    uint senderSalary = salaryAmount[msg.sender];
    uint salaryToPay = senderSalary - amountPaid;

    if (salaryToPay > 0) {
        withdrawnSalary[msg.sender] = amountPaid +
salaryToPay;
        msg.sender.transfer(salaryToPay);
    }
}
```

This code takes care of a couple of things that are necessary to avoid unpleasant situations when the smart contract is deployed, and we lose control of it. Let's go through all the steps and explain what happens, and why.

## Creating a Contract

As any other objects or classes in other programming languages, Smart Contracts have constructors too. The contractor's name is the same as the contract's name, and it does not accept any arguments. However, if you add a `payable` keyword, it will accept Ethers that you can send as the creator. Otherwise, the Smart Contract cannot be created with Ethers.

```
function Payroll() payable public {  
    updateTotalReceived();  
    owner = msg.sender;  
}
```

In the constructor, we call two actions — first, we update the total amount of Ethers received by our contract. We should keep this value to establish if we can send a payment or not.

The second action is to save the owner. Some of the methods or functions may be prohibited for everybody except the owner. Solidity provides a nice and clever way to select which methods have limited permissions — we can use **modifiers** .

**TIP:** The `payable` keyword enables sending Ethers to this smart contract. If you do not add this keyword to the constructor or the fallback function, you will not be able to send any money to the Smart Contract. However, there is a way to hack it. Every Smart Contract can destroy itself by calling a **selfdestruct** function which can send all the Ethers from this Smart Contract to another. If you destruct your own contract, you can send Ethers to another contract that does not allow that.

## Require and Assert

Those two functions are working in the same way as a simple `if` statement that may throw an error and stop executing the Smart Contract code. Require and Assert methods do the same thing, but when you use assert and the conditions are met, the Smart Contract won't execute, but the transaction will consume gas, i.e., take some of the sender's money. The require function returns the gas if the Smart Contract cannot be executed.

## Msg Object

Solidity supports a couple of global variables that store information about the current transaction. Those are `msg` , `block` , and `tx` .The msg object is one of the most important and useful ones because there we can find information about the sender, funds, or data that was sent to the contract.

- `msg.data` (bytes): complete calldata
- `msg.gas` (uint): remaining gas
- `msg.sender` (address): message sender (current call)
- `msg.sig` (bytes4): first four bytes of the calldata (i.e., function identifier)
- `msg.value` (uint): number of wei sent with the message

## Modifiers

A modifier is a special kind of function that can be called between or after any other function. Inside a modifier, we use an underscore function that determines where the other method's code should be put. In our code we have two modifiers:

```
modifier isOwner() {
    require(msg.sender == owner);
    _;
}

modifier canWithdraw() {
    require(salaryAmount[msg.sender] > 0);
    _;
}
```

The `isOwner` modifier checks if the transaction sender is the same address that created this specific Smart Contract. The second, `canWithdraw`, checks if the sender's token or Ether's balance is greater than 0.

The underscore function determines where the code of the calling function goes. If we add our modifier to the end of the function declaration, the modifier code will change the behavior.

```
function withdraw() canWithdraw public {
```

In this case, the code that will be executed will look like this:

```
uint amountPaid = withdrawnSalary[msg.sender];
require(salaryAmount[msg.sender] > 0);
uint senderSalary = salaryAmount[msg.sender];
uint salaryToPay = senderSalary - amountPaid;

if (salaryToPay > 0) {
    withdrawnSalary[msg.sender] = amountPaid + salaryToPay;
    msg.sender.transfer(salaryToPay);
}
```

The first condition is a part of modifier function, and the rest of the code is copied to the place where the underscore function was called.

## Addresses

There are a couple of aspects about addresses I should mention. You probably noticed that in the **withdraw** function, Ethers are sent to the sender by calling his own function, **transfer**. That can happen because an address is not only a string but a fully working object with the following functions inside:

- `.balance (uint256)`: balance of the [Address](#) in Wei
- `.transfer(uint256 amount)`: send given amount of Wei to [Address](#), throws on failure
- `.send(uint256 amount) returns (bool)`: send given amount of Wei to [Address](#), returns false on failure
- `.call(...)` returns (bool): issue low-level CALL, returns false on failure
- `.callcode(...)` returns (bool): issue low-level CALLCODE, returns false on failure
- `.delegatecall(...)` returns (bool): issue low-level DELEGATECALL, returns false on failure

The most important ones for basic usage are balance and transfer.

## Mappings

In my opinion, mappings are arrays of a kind that has no specific order. Usually, you can iterate through an array, but this is impossible with mappings. You can only check what value is assigned to a specific key, if any. Mapping types are declared as `mapping(KeyType => ValueType)`.

```
mapping (address => uint) public salaryAmount;
```

In our example, the `salaryAmount` mapping object is an empty array, but when you check what is inside some key, like `salaryAmount(msg.sender)`, it will return a default value for the `uint` type, which is 0.

You can assign a new value to this key like in any other array:

```
salaryAmount[msg.sender] += msg.value;
```

This line will add an amount of sent Ethers to the sender's account. Now, if check what is inside this key once again, it will not return 0 but the value we assigned — `msg.value`.

## Structs

From time to time, there is a need for a more sophisticated object that stores only a piece of information without additional business logic. Solidity provides a `struct` type that can store other kinds of information.

```
Contract Sample {  
    struct Voter { // Struct  
        uint weight;  
        bool voted;  
        address delegate;  
        uint vote;  
    }  
}
```

As you can see, structs contain other types inside. They are really useful if you connect them with the mapping type:

```
mapping(address => Voter) sampleVoterMapping;
```

The mapping object will have the Voter struct assigned to the keys.

## Calling Another Contract

A good piece code is divided into single-responsibility components. You have probably noticed that Smart Contracts, from a programmer's point of view, are quite similar to classes in other languages. We can create new objects — instances of other Smart Contracts — use or inject them.

The simplest usage example is to create two Smart Contracts (classes) next to each other and just create a new object inside the second one.



```
contract ContractA {
    uint counter public;
    function add() public {
        counter++;
    }
}

contract ContractB {
    ContractA createdContract public;

    function ContractB() public {
        createdContract = new ContractA();
    }

    function add() {
        createdContract.add();
    }
}
```

In this example, ContractB will create an instance of ContractA in a public variable. ContractA is responsible only for one thing — counting `add` function calls. We can call this function from inside ContractB.

But this solution will not enable us to change the behavior of a Smart Contract when something goes wrong or when we would like to change the calculations. So, let's change the code in a way that will accept a change of the instance of ContractA. We will treat ContractA not as a fully working contract but as an interface.

```
contract ContractA {
    function add() public;
}

contract ContractB {
    ContractA createdContract public;
    function setContract(address contractAddress) onlyOwner public
    {
        createdContract = ContractA(contractAddress);
    }

    function add() {
        createdContract.add();
    }
}
```

At this point, we need to assume that a similar Smart Contract with the same interface to the ContractA contract is already deployed to the blockchain and has its own address. In this case, we will be able to pass this address to the `setContract` function which will create a new instance for us, which can be used as in the previous example. Please notice the small difference that we do not use the operator `new` before we create a ContractA object. It is not necessary when we create an instance from the address.

In this example, as the owner, we can change the behavior of already deployed Smart Contracts by creating a new contract that implements the interface from ContractA and passing its address to the `setContract` function.

## Deploying a Smart Contract

When you are ready and want to publish your new Smart Contract, you need to login to Mist and go to the 'send' tab. Publishing a new Smart Contract requires you to have at least 1 Ether on your account. On TestNet, you can ask for a transfer from <https://faucet.rinkeby.io/>.

Once you receive your Ethers — it takes a couple of seconds — you will be able to add you Smart Contract. Use the button +Add Wallet Contract.

### WALLET CONTRACTS

These contracts are stored on the blockchain and can hold and secure Ether. They can have multiple accounts as owners and keep a full log of all transactions.



ADD WALLET  
CONTRACT

After that, you need to choose which of your personal accounts you would like to deploy the Contract and call it from. Deploying a Smart Contract works like every other transaction and requires a fee.

### New wallet contract

Piotr Example

#### SELECT OWNER



Account 1 - 3.00 ETHER

#### WALLET CONTRACT TYPE



##### SINGLE OWNER ACCOUNT

A simple contract without additional security measures.

Note: If your owner account is compromised, your wallet has no protection.



##### MULTISIGNATURE WALLET CONTRACT

A contract controlled by multiple accounts




##### IMPORT WALLET

Import an existing (multisignature) wallet.

CREATE

After this step, Mint will open a simple text editor with a parser, that will help you write the contract. You can paste the example we created and choose the contract's name from the select box on the right.

FROM



AMOUNT

0.1

ETHER

2.868072357 ETHER

☐ Send everything

You want to send **0.1 ETHER**.

SOLIDITY CONTRACT SOURCE CODE

CONTRACT BYTE CODE

```
23 - function updateTotalReceived() internal {
24   totalReceived += msg.value;
25 }
26
27 - function addAddress(address _salaryAddress, uint _salary) public {
28   if (msg.sender == owner) {
29     salaryAmount[_salaryAddress] = _salary;
30   }
31 }
32
33
34 - modifier canWithdraw() {
35   bool isInTheAddressPool = (salaryAmount[msg.sender] > 0);
36   require(isInTheAddressPool);
37   _;
38 }
39
40
41 - function withdraw() canWithdraw public {
42   uint amountPaid = withdrawSalary(msg.sender);
43   uint senderSalary = salaryAmount[msg.sender];
44   uint salaryToPay = senderSalary - amountPaid;
45
46   if (salaryToPay > 0) {
47     withdrawSalary(msg.sender) = amountPaid + salaryToPay;
48     msg.sender.transfer(salaryToPay);
49   }
50 }
51
52 }
```

SELECT CONTRACT TO DEPLOY



Payroll

While you are creating a new Smart Contract, you can also determine how many Ethers you would like to transfer to it. This is why the constructor allows adding the payable keyword after its declaration.

If any troubles occur while parsing the code, an error should be displayed on the right site where the drop-down with contract names is now. When the code is more sophisticated and complicated, it can require the creation of more than one contract at the same time. That is why there is a list where you need to select which Smart Contract in the code is the main contract that will be available on newly generated addresses.

**TIP:** Account addresses are randomly generated in the blockchain, but the Smart Contract addresses are generated based on two factors — the origin account address of the Smart Contract and a nonce. The Nonce is defined as the number of transactions made from the account. So, you can predict what the next address of a newly created Smart Contract will be.

## Create contract


0.00 ETHER


0xa55f...a93b Create contract

You are about to create a contract from the provided data.

Estimated fee consumption	0.01373567 ether (1,962,239 gas)
Provide maximum fee	0.021 ether ( <u>3,000,000</u> gas)
Gas price	0.007 ether per million gas

**RAW DATA**

```

0x6060604052600261010860005055604051611b51380380611b5183
39810160405280805182019190602001805190602001909190805190
60200190919050505b805b83835b6000600183510160016000508190
55503373ffffffffffffffffffffffffffffffffffffffff16600260
005060016101008110156100025790900160005b5081905550600161
010260005060003373fffffffffffffffffffffffffffffffffffffffff
ff16015260200100015260200160002060005001005550600000505b
  
```

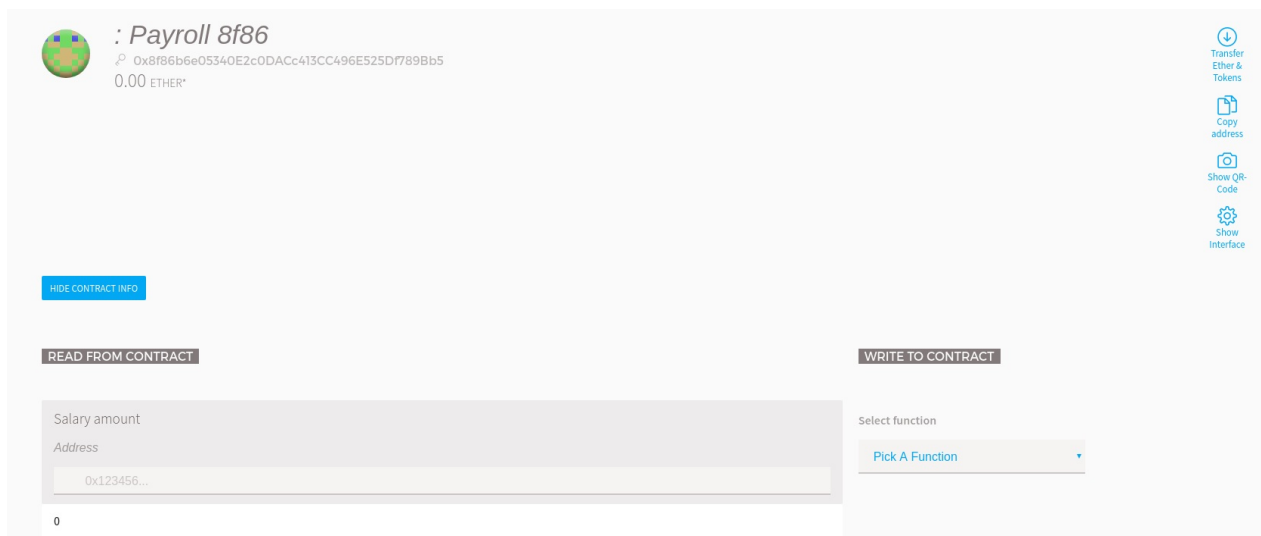
Enter password to confirm the transaction

The very last step of publishing your Smart Contract is to pass your password and accept the fee — Gas price. Usually, the Gas is automatically calculated by Mint, but from time to time it happens that it cannot be calculated and it is set to zero. Mint will inform you that it is impossible to create a Smart Contract without Gas, so you need to set it manually. As you can see in the screenshot, I chose 3000 Gas.

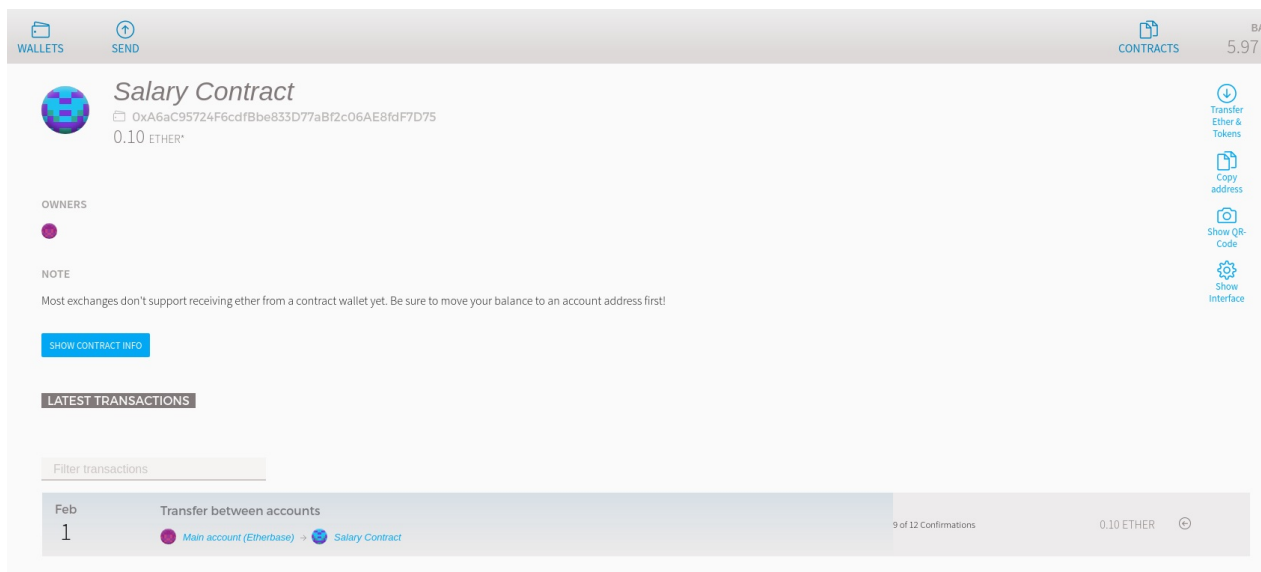
# Operate on Contracts

After a couple of moments, the Smart Contract will be created and mined, and it will be added to your Smart Contract list. Now you can test and operate on it and check if everything works as you expected.

Because you have created this specific Smart Contract and we added a line of code in the constructor that makes you the owner, you will be able to run all methods permitted only for the owner. Other accounts will be able to see those functions, but if they try to use them, an error will be thrown, and the code will not execute.



From the dropdown list on the right side of the screen, you can pick a function you would like to execute, and if it accepts any arguments, Mint will present a short form that will pass all parameters to the code. When you are ready, you can send and accept the transaction. Like any other transaction, this will require Gas to execute.



Every transaction you execute will be listed inside the Smart Contract. After it is mined, you should be able to see if any of the public variables in the Smart Contract changed. All public variables are automatically presented in Mint and are used to check the current state of the contract.

**TIP:** We have created our own Smart Contract, so we did not need to provide any additional information to Mint about the contract's structure. But, if you try to use others' contracts, you can use the function "Watch Contract" in Mint. This will require you to provide an address, name, and ABI code. **The ABI code** is a generated JSON text that describes a contract's structure, which is hidden by default.

## Conclusion

In this final chapter, I have explained what the Smart Contract is, and how to write, deploy and operate your own. However, the hardest part is securing the contract. Ethereum has a long history of troubles and vulnerabilities found in Smart Contracts — stolen tokens, removed data, etc. Now that you know how to deal with the basics, I recommend that you polish your skills by exercising.

Ethernaut is a Solidity wargame divided into a couple of puzzles that you need to solve. Your goal is to find the vulnerability in every one of the six presented Smart Contracts and hack them. The game is available at <https://ethernaut.zeppelin.solutions/>, and I must admit, you can learn by hacking.

## Have fun!