DEVELOPMENT NOTEBOOK

Blockchain development, Blockchain Minor, HBO-ICT

Abstract

In this document you can read about my development throughout the course Blockchain development week by week.

Thomas Bronsveld 500757630

Thomas.Bronsveld@hva.nl

Contents

1. Introduction

The blockchain development course teaches us the fundamentals about blockchain and how to develop a blockchain. It touches upon topics like smart contracts, double spending, forking and setting up a blockchain.

I've had an interest in blockchain for a while now, however, I never decided to look up about Blockchain on the internet. While I was quite tempted to do it, I knew that I would follow the Blockchain minor and thus decided it would be better to not look anything up. I was of the opinion that the Blockchain minor would be a much better place to set up a foundation than the internet with all the scattered information and, quite often, wrong information.

Blockchain development was one of the courses I was looking forward to. As a Software Engineering student, I really wanted to know the ins and outs of the blockchain technology and how to develop one.

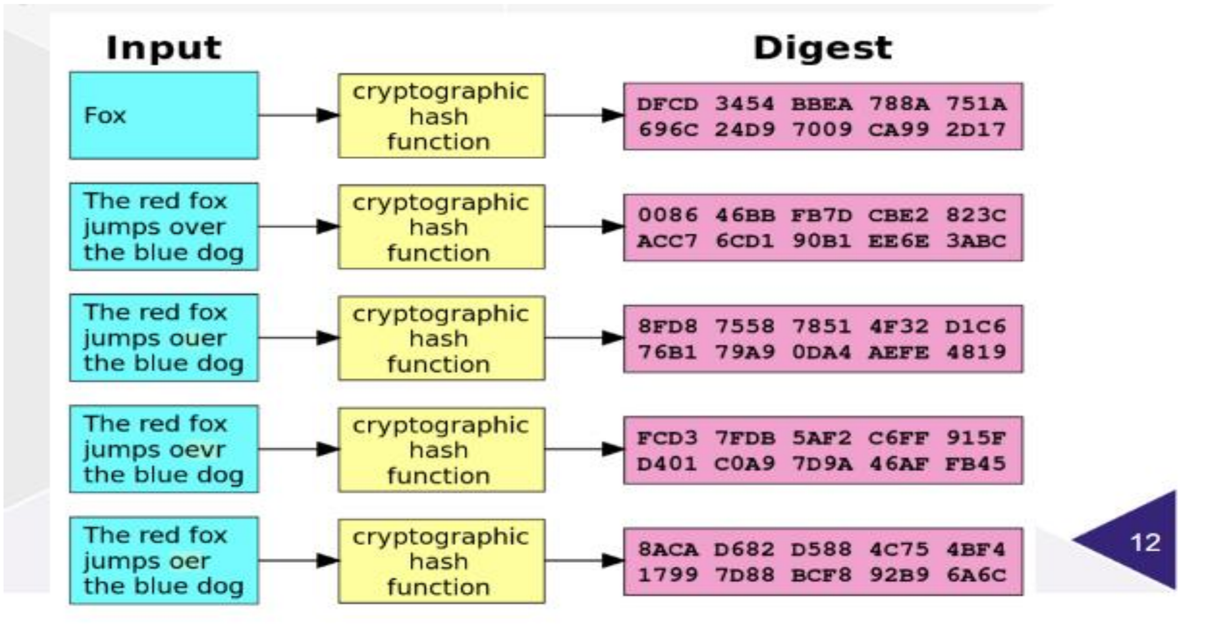
2. Week 1: The building blocks of Blockchain

This week started with the general course introduction followed up about the basics of blockchain. The topics briefly touched upon were the following:

- How Blockchain is secure
- Decentralized network
- The components of a Blockchain

2.1 How Blockchain is so secure

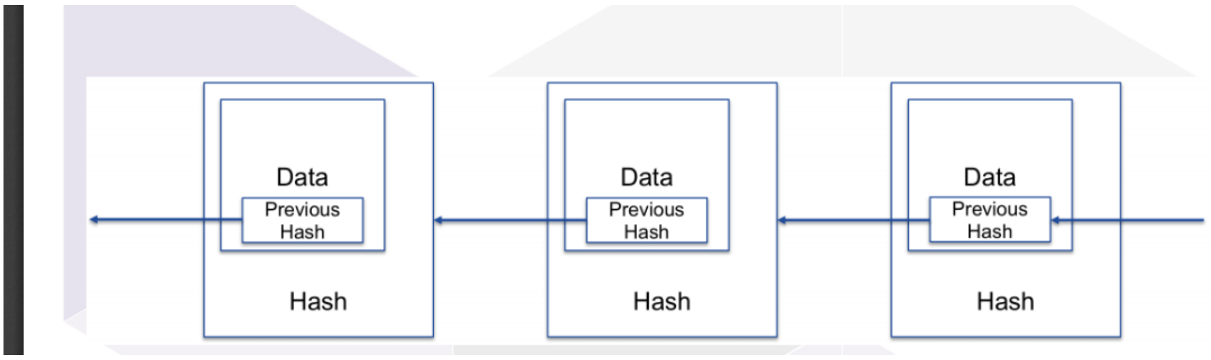
We started this course with the explanation of how Blockchain is so secure. Blockchain makes use of the SHA256 cryptographic method. The SHA256 cryptographic method creates a mix of both characters and numbers based on the data that you assign as the input. What this means is that any input doesn't generate the same series of characters and numbers. This is the crux of the Blockchain infrastructure, because if you change just a single letter in the data, the outcome of the SHA256 algorithm will be completely different. This is what we call Hashing. Image 2.1.1 showcases that there is a large difference between the output of the hashing algorithm with only a single letter change.


Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C
12

2.1.1: How hashing works https://simple.wikipedia.org/wiki/Cryptographic_hash_function

Hashing is used because it is a one way encryption and it is extremely rare to generate the same hash with different inputs. The hash of the block takes the input of the block's data, timestamp, index, and the hash of the previous block. If the block is the genesis block, aka the first block in the blockchain, there is no previous hash and you can set it to null or give it a bunch of zero's.
The hash of the previous is one of the two measures to achieve security, with the other being the hash of the latest block. The hash of the latest block will of course be different if someone changes the data of the previous block and thus cause a change in the hash value that will become the hash of the

previous block. This connection of two hash values creates the security of Blockchain. Image 2.1.2 illustrates the connection between the blocks of a Blockchain.



2.1.2: https://anders.com/blockchain/blockchain.html the connection between the blocks illustrated.

2.2 The building blocks of the Blockchain

Now that we know about the hashing functionality of the blockchain and that it is set up in a decentralized manner, we can talk about the what the Blockchain consists of. The fundamental part of the blockchain is the block. The block contains several attributes of information. The first and most important part is the hash. The hash is the representation of the block and is the crux of the entire Blockchain. The hash is generated by the input, the input consists of the following parts:

- Data, this can be anything from transactions to documents.
- Index, the number of position of the block in the Blockchain.
- Timestamp, when the block is generated.
- The hash of the previous block.

These inputs generate the hash for the block that is added to the blockchain. Once the hash is generated the block is fully generated and thus can be seen as closed.
Something to mention is that when a Blockchain is initially setup, the first block of the Blockchain is named the Genesis block. This block has as previous hash a bunch of zeros, the data can be something generic like a statement about the goals of the Blockchain and why it is started and the index is 0. The timestamp is automatically generated and the hash is generated based on the previous talked about input.

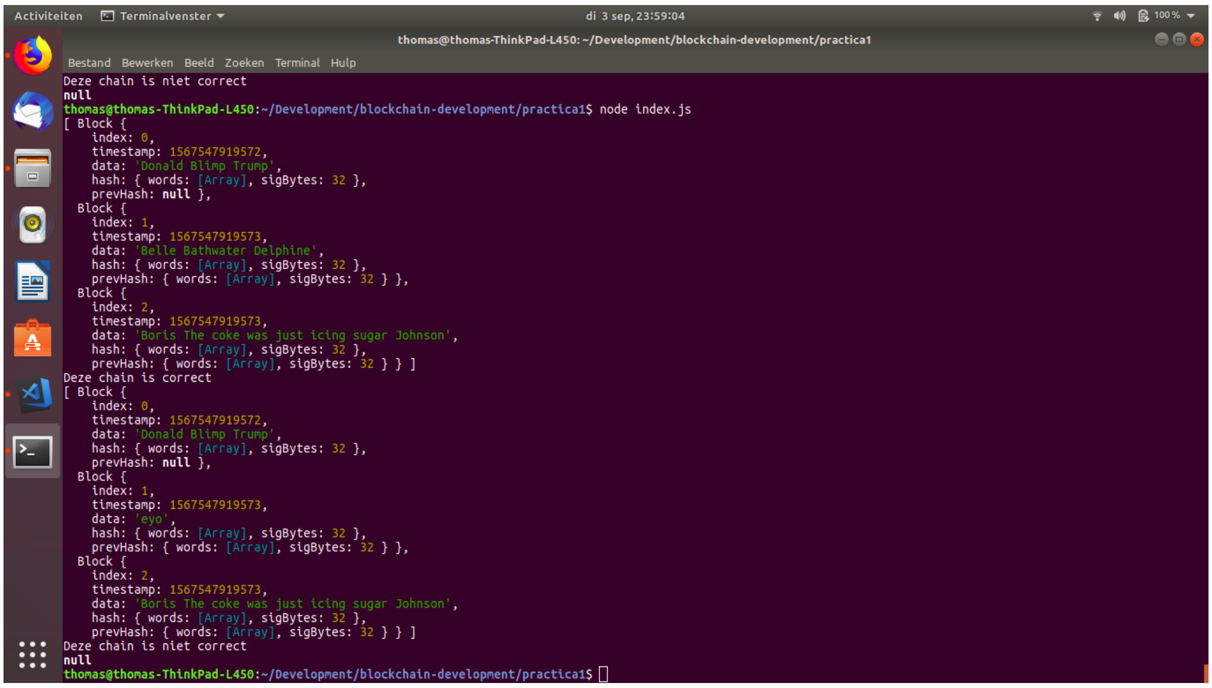
2.3 Assignment of the week

Create a self-written Blockchain in your language of choice. The team I am part of decided to write the Blockchain in Javascript.

```
let sha256 = require('crypto-js/sha256');



let dataList = ['Dit is een test',

  'Dit is een andere test',
  'Ik ben Thomas Bronsveld',
  'Wij zijn team logisch'
]
//Simple Class in order to make Block objects.
class Block{

  constructor(index){
    this.index = sha256(index);
    this.timestamp = new Date().toISOString().replace(/T/, ' ').     // replace T with a space
    replace(/\..+/, '');
    this.data = [];
    this.hash;
    this.prevHash;
  }
 //function to calculate the hash of the block with all the inputs.
  calculateHash(){
    this.hash = sha256(sha256(this.data, this.index, this.timestamp, this.prevHash));
  }
}
class Blockchain { //Class to make a blockchain object.
  constructor(){
    this.chain = [];
  }

  addBlock(block){ //adds a block object to the chain.
    if(block instanceof Block){
      this.chain.push(block);
```

```
            this.checkValidity();

            return;

        }

        console.log("Er is geprobeerd iets raars toe te voegen aan de chain");

        return;

    }

    createGenesisBlock(){ //Creates a genesis block and adds it to the chain

        let genesisBlock = new Block(this.chain.length);

        genesisBlock.data.push(dataList[this.chain.length]);

        genesisBlock.prevHash = null;

        genesisBlock.calculateHash();

        // this.addBlock(genesisBlock);

        this.chain.push(genesisBlock);

    }

    checkValidity(){ //checks the validity of every block in the chain

        for(let i = 1; i < this.chain.length; i++){

            if(this.chain[i].prevHash !== this.chain[i - 1].hash){

                console.log("Deze chain is niet correct. Gemanipuleerd bij block: " + (i -1));

                return;

            }

        }

        console.log("Deze chain is correct");

        return;

    }

}

//Zet de chain op.

let blockChain = new Blockchain();
```

```
blockChain.createGenesisBlock();


setInterval(function(){ //check the chain validity every 3 seconds.
  blockChain.checkValidity();
},3000);


//Voeg nieuwe Block toe.
let testBlock = new Block(blockChain.chain.length);
testBlock.data.push(dataList[testBlock.index]);
testBlock.prevHash = blockChain.chain[blockChain.chain.length - 1].hash;
testBlock.calculateHash();
blockChain.addBlock(testBlock);


let grn = "test";
blockChain.addBlock(grn);


//3de block.
let testBlock3 = new Block(blockChain.chain.length);
testBlock3.data.push(dataList[testBlock3.index]);
testBlock3.prevHash = blockChain.chain[blockChain.chain.length - 1].hash;
testBlock3.calculateHash();
blockChain.addBlock(testBlock3);



//Manipuleer data 2de block.=3-3
blockChain.chain[1].data = sha256(dataList[dataList.length - 1]);
blockChain.chain[1].calculateHash(;
```


Activiteiten
Terminalvenster
di 3 sep, 23:59:04
100 %
thomas@thomas-ThinkPad-L450: ~/Development/blockchain-development/practica1
Bestand Bewerken Beeld Zoeken Terminal Hulp

```
Deze chain is niet correct
null
thomas@thomas-ThinkPad-L450:~/Development/blockchain-development/practica1$ node index.js
[ Block {
    index: 0,
    timestamp: 1567547919572,
    data: 'Donald Blimp Trump',
    hash: { words: [Array], sigBytes: 32 },
    prevHash: null },
  Block {
    index: 1,
    timestamp: 1567547919573,
    data: 'Belle Bathwater Delphine',
    hash: { words: [Array], sigBytes: 32 },
    prevHash: { words: [Array], sigBytes: 32 } },
  Block {
    index: 2,
    timestamp: 1567547919573,
    data: 'Boris The coke was just icing sugar Johnson',
    hash: { words: [Array], sigBytes: 32 },
    prevHash: { words: [Array], sigBytes: 32 } } ]
Deze chain is correct
[ Block {
    index: 0,
    timestamp: 1567547919572,
    data: 'Donald Blimp Trump',
    hash: { words: [Array], sigBytes: 32 },
    prevHash: null },
  Block {
    index: 1,
    timestamp: 1567547919573,
    data: 'eyo',
    hash: { words: [Array], sigBytes: 32 },
    prevHash: { words: [Array], sigBytes: 32 } },
  Block {
    index: 2,
    timestamp: 1567547919573,
    data: 'Boris The coke was just icing sugar Johnson',
    hash: { words: [Array], sigBytes: 32 },
    prevHash: { words: [Array], sigBytes: 32 } } ]
Deze chain is niet correct
null
thomas@thomas-ThinkPad-L450:~/Development/blockchain-development/practica1$ 
```

Figure 2.3.1: The terminal output of the code snippet above.

2.4 What I learned from this assignment

From this assignment I learned how to create a Blockchain in code. While it is basic Javascript, it was quite interesting to see how a blockchain would look like in the terminal.

3. Week 2

This week we expanded upon the basis of Blockchain. We discussed the benefits of Distributed network vs a centralized network and we talked a short bit about distributed ledgers. After that we build further upon the basis of Blockchain with Consensus, miners and the 51% concept.

3.1. Distributed vs centralized & distributed Ledgers

I knew what decentralized network meant before I followed the course. Decentralized network means that not a single entity is in control of the network and that every person participating in the network as a collective is in control. People check each other's information and state of the blockchain. This setup makes sure that every single individual node is up to date with the latest version of the blockchain and because of this it is difficult to manipulate data. The reason behind that is because every node has the entire information, changing data means that every node has to validate it as true and in a blockchain this is extremely difficult to accomplish.
"A decentralized network offers increased system reliability, scale and privacy" (Solarwinds msp, 2018).

A centralized network means that a single or extremely small amount of entities are in control of the network. "Some key advantages of a centralized network or authority are consistency, efficiency and affordability" (Solarwinds msp, 2018) .

Distributed ledgers are a type of database that is spread across multiple sites, countries or institutions, and is typically public. Records are stored one after the other in a continuous ledger, rather than sorted into blocks, but they can only be added when the participants reach a quorum. A distributed ledger requires greater trust in the validators or operators of the ledger. For example, the global financial transactions system Ripple selects a list of validators (known as Unique Node Validators) from up to 200 known, unknown or partially known validators who are trusted not to collude in defrauding the actors in a transaction. This process provides a digital signature that is considered less censorship resistant than Bitcoin's, but is significantly faster (UK Government Chief Scientific Adviser, 2016).

3.2. What is consensus?

Consensus is the mechanism used to validate the Blocks. As mentioned in the previous chapter, each block has a couple of inputs that generate the hash for the block. The consensus comes into play when the hash is generated. Each node of the Blockchain has the same information, unless manipulated, and once a nonce is found it generates a specific hash. This nonce gets found by miners. Miners are the ones who deliver computational power, which costs electricity, to find this nonce. Because finding this nonce costs electricity and computational power, just the finding of a nonce can take a lot of work from the computers that take part in mining. This is the principle upon which Proof of Work works.

Through consensus the nonce gets checked by every node and see if the hash they generate with the found nonce is equal to the hash that the one who claims to have found the nonce claims to be. If the hash found by the entirety of the Blockchain is not the same, then there is a dispute in the Blockchain.

A dispute can be solved through multiple manners. The first manner is that the nonce is not valid and the entirety of the Blockchain moves on to find a new nonce.

The second manner is that the one who claims to have found the nonce, has altered their data. If this happens the founder of the so called nonce either needs to change their data back to the original data or he can decide to continue on his/her own. This is called forking. We will discuss this topic broader in the 51% concept paragraph.

3.3. Consensus example

The most well known example is the Proof of Work consensus method. With this consensus method people are trying to find a specific nonce, a value, that generates a hash that consists of a predetermined configuration. Usually this configuration has a certain amount of zeros in it, set by the difficulty of the Blockchain. If the difficulty is set to four, then the miners need to find a nonce that generates a hash value with four zeros at the start of it. Once a nonce is found, this nonce gets distributed to all nodes within the Blockchain and validated. The validation is a simply check where the nodes check if the hash they generate, with the nonce, also comes out with four zeros in it and if the hashes match.

3.4. 51% concept.

In Blockchain there is something called forking. Forking is done when either the one who forks is in disagreement with the Blockchain or someone is trying to do double spending.

If someone is in disagreement, he can fork the Blockchain and continue on with his own rules. You can kind of see this as two shareholders of a company splitting up and one starting his/her own company with the experiences that they have from the company that they were originally a shareholder of. With the experiences in Blockchain being the old blocks up until the part where they had a disagreement.

The last part of forking is where the 51% comes into. In Blockchain someone can also double spend their monetary assets. So let us say we are at block 50 in the Blockchain. The one who wants to double spend their money, spends their money on block 50, gets what they want for their money and then put the Blockchain on their node on hold. This means that their version of the Blockchain stays on block 50 while the others on the Blockchain continue. If the person who has stopped at Block 50 of the Blockchain has 51% or more of the computational power, they can play catch up with the main Blockchain and eventually pass them with the amount of blocks. When that happens, because his chain is longer and that means his chain is the truth, he still has the money that he spend on the original Blockchain.

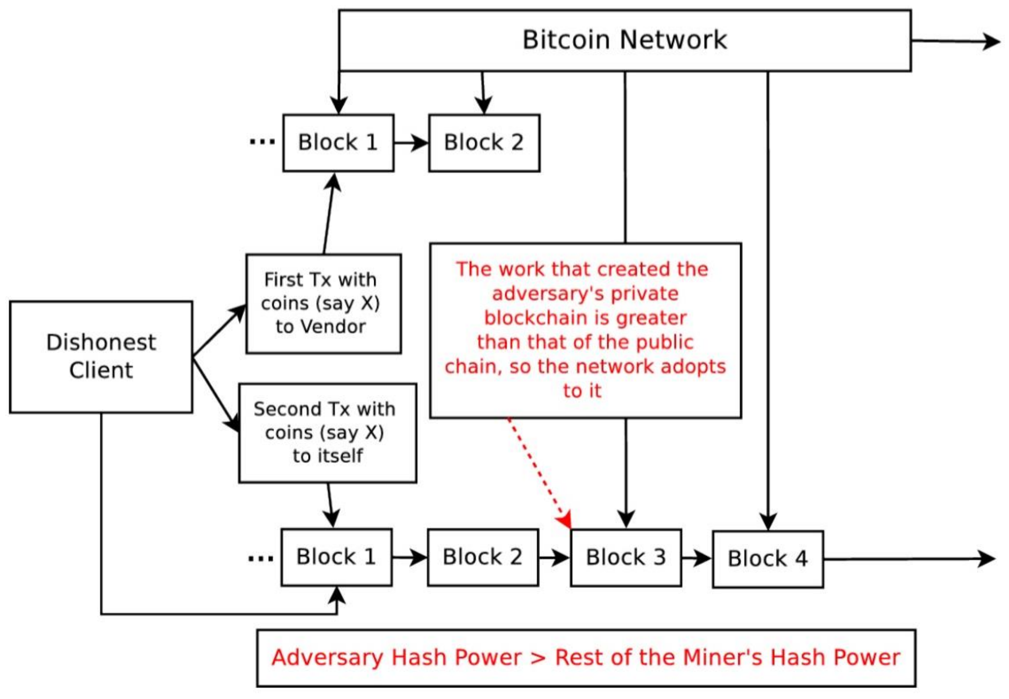

Bitcoin Network
Block 1
Block 2
Dishonest Client
First Tx with coins (say X) to Vendor
Second Tx with coins (say X) to itself
The work that created the adversary's private blockchain is greater than that of the public chain, so the network adopts to it
Block 1
Block 2
Block 3
Block 4
Adversary Hash Power > Rest of the Miner's Hash Power

Fig. 5. Double Spending Attack

Figure 3.4.1: Double spending. Source: http://casanova.al/bull-cardano/what-is-s-bitcoin-double-spending-problem-in-bitcoin/

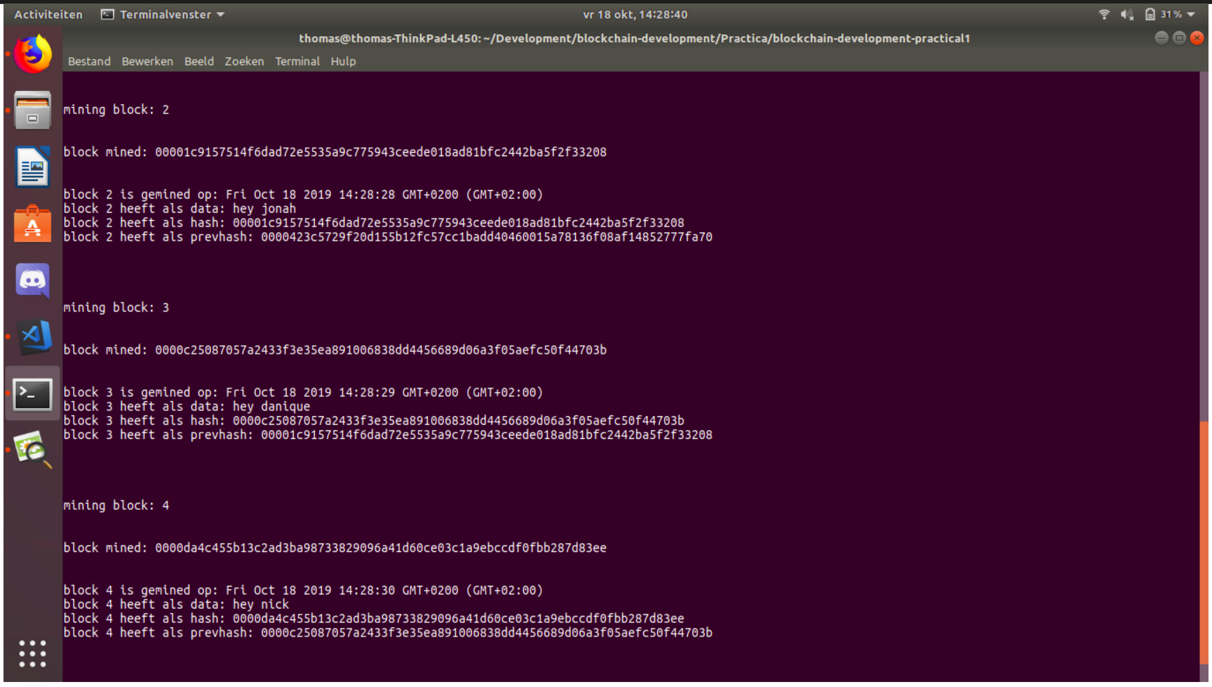
3.5 Assignment week 2

This week we had to add a nonce and difficulty to our own created Blockchain of lesson 1.

```
class Block {

   constructor(index, timestamp, data, prevhash){

       this.index = index;

       this.timestamp = timestamp;

       this.data = data;

       this.prevhash = prevhash;

       this.hash = this.calculatehash();

       this.nonce = 0;
```

```
    }



    calculatehash(){

        return SHA256(SHA256(this.index + this.prevhash + this.timestamp +
this.data + this.nonce))

    }

    mineBlock(difficulty) {

        console.log("mining block: " + this.index);

        console.log("\n");

        while (this.hash.toString().substring(0, difficulty) !==
Array(difficulty + 1).join("0")) {

            this.nonce++;

            this.hash = this.calculatehash();

        }



        console.log("block mined: " + this.hash);

        console.log("\n");

        console.log("block "+this.index+" is gemined op: "+this.timestamp);

        console.log("block "+this.index+" heeft als data: "+this.data);

        console.log("block "+this.index+" heeft als hash: "+this.hash);

        console.log("block "+this.index+" heeft als prevhash:
"+this.prevhash);

        console.log("\n");

        console.log("\n");

    }

}
```

```
class Blockchain {

   constructor(){

       this.chain = [this.createGenesisBlock()];

       this.difficulty = 4;

   }

let coin = new Blockchain();

coin.addBlock(new Block(coin.chain.length, Date(Date.now()), "hey neal",
coin.chain[coin.chain.length-1].hash));

coin.addBlock(new Block(coin.chain.length, Date(Date.now()), "hey jonah",
coin.chain[coin.chain.length-1].hash));

coin.addBlock(new Block(coin.chain.length, Date(Date.now()), "hey
danique", coin.chain[coin.chain.length-1].hash));

coin.addBlock(new Block(coin.chain.length, Date(Date.now()), "hey nick",
coin.chain[coin.chain.length-1].hash));
```


Activiteiten
Terminalvenster
vr 18 okt, 14:28:40
31 %
thomas@thomas-ThinkPad-L450: ~/Development/blockchain-development/Practica/blockchain-development-practical1
Bestand
Bewerken
Beeld
Zoeken
Terminal
Hulp

```
mining block: 2

block mined: 00001c9157514f6dad72e5535a9c775943ceede018ad81bfc2442ba5f2f33208

block 2 is gemined op: Fri Oct 18 2019 14:28:28 GMT+0200 (GMT+02:00)
block 2 heeft als data: hey jonah
block 2 heeft als hash: 00001c9157514f6dad72e5535a9c775943ceede018ad81bfc2442ba5f2f33208
block 2 heeft als prevhash: 0000423c5729f20d155b12fc57cc1badd40460015a78136f08af14852777fa70



mining block: 3

block mined: 0000c25087057a2433f3e35ea891006838dd4456689d06a3f05aefc50f44703b

block 3 is gemined op: Fri Oct 18 2019 14:28:29 GMT+0200 (GMT+02:00)
block 3 heeft als data: hey danique
block 3 heeft als hash: 0000c25087057a2433f3e35ea891006838dd4456689d06a3f05aefc50f44703b
block 3 heeft als prevhash: 00001c9157514f6dad72e5535a9c775943ceede018ad81bfc2442ba5f2f33208



mining block: 4

block mined: 0000da4c455b13c2ad3ba98733829096a41d60ce03c1a9ebccdf0fbb287d83ee

block 4 is gemined op: Fri Oct 18 2019 14:28:30 GMT+0200 (GMT+02:00)
block 4 heeft als data: hey nick
block 4 heeft als hash: 0000da4c455b13c2ad3ba98733829096a41d60ce03c1a9ebccdf0fbb287d83ee
block 4 heeft als prevhash: 0000c25087057a2433f3e35ea891006838dd4456689d06a3f05aefc50f44703b
```

Figure 3.5.1: The terminal output of the code snippet above.

3.6 What I learned from the assignment

I learned how to implement the nonce and difficulty to generate a hash value that has a set amount of zeros equal to the difficulty.

4. Week 3

This week once again started with presenting our blockchain application with a PoW implementation.

The topics of this week are focused on Proof of Stake, smart contracts and Ethereum.

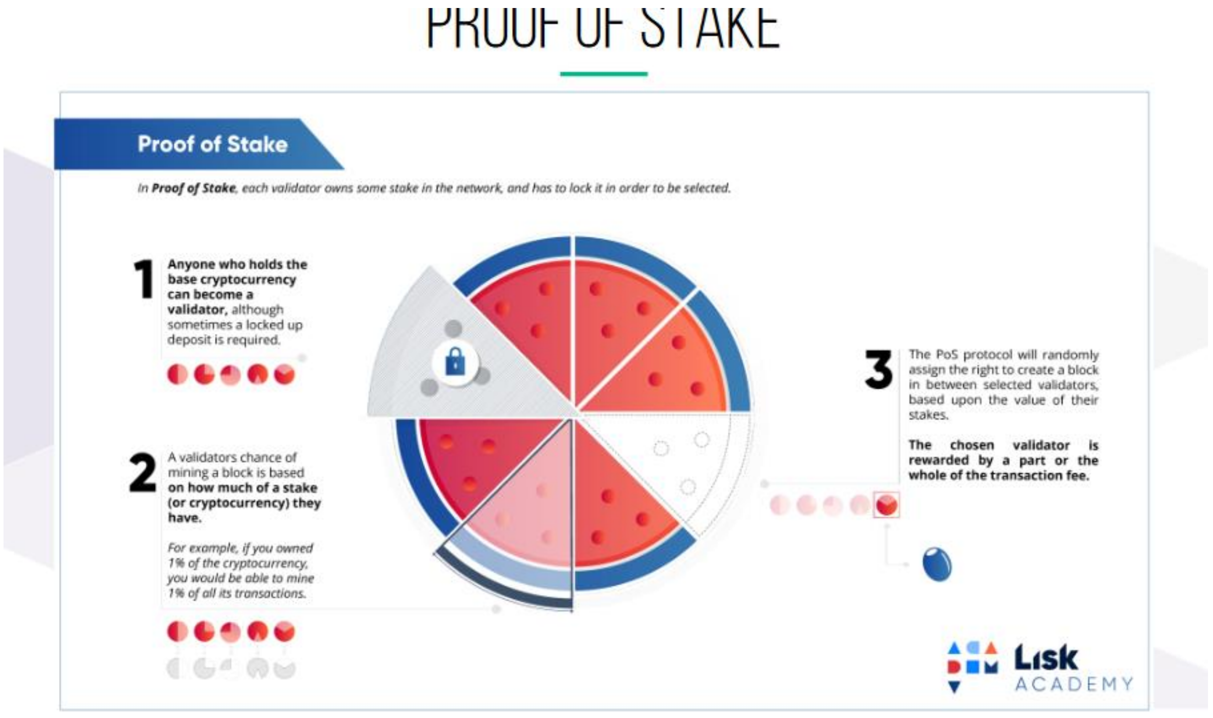
4.1 Proof of stake

Proof of Stake is a consensus algorithm where the blockchain aims to achieve distributed consensus. This is done through various manners like random selection, wealth and age or some other blockchain specific value to form a stake. The larger the stake, the higher the chance is that you get selected to validate the block. If someone stakes nine euro and someone else stakes a single euro, the one who staked nine euro's got a chance of ninety percent to get chosen while the one who staked a single euro ten percent. This can of course lead to a monopolization of the ability to validate a block. Someone with a lot of wealth has of course a higher chance to get chosen if he stakes a lot. To prevent this a lot of blockchains which use Proof of Stake make sure that when someone get chosen, he doesn't get entered into the pool of potential validators for the next block.

Once the creator of the next block is selected, they create the block and receive network fees as payment for their stake. Proof of stake has the following advantages over PoW:

- Energy consumption is far lower
- Discourages creation of centralized cartels like what happens with Bitcoin.

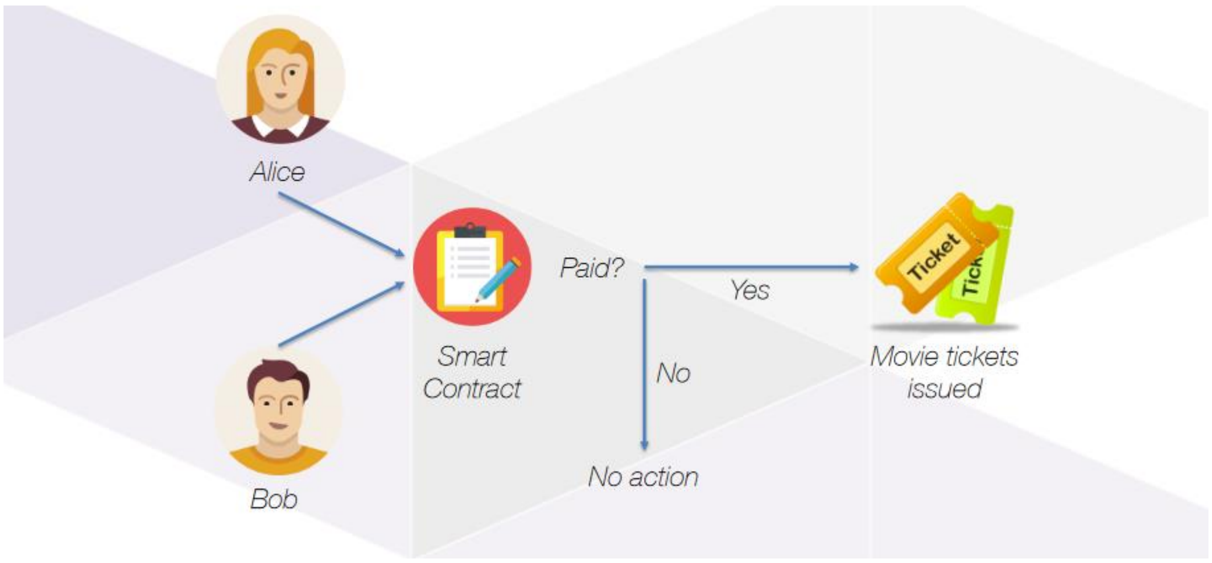
Image 4.1.1 showcases Proof of Stake


PROOF OF STAKE
Proof of Stake
In Proof of Stake, each validator owns some stake in the network, and has to lock it in order to be selected.
1 Anyone who holds the base cryptocurrency can become a validator, although sometimes a locked up deposit is required.
2 A validators chance of mining a block is based on how much of a stake (or cryptocurrency) they have.
For example, if you owned 1% of the cryptocurrency, you would be able to mine 1% of all its transactions.
3 The PoS protocol will randomly assign the right to create a block in between selected validators, based upon the value of their stakes.
The chosen validator is rewarded by a part or the whole of the transaction fee.
Lisk ACADEMY

4.1.1: Imagery of Proof of Stake. source: https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/proof-of-stake

4.2 Smart Contract

Smart contract is an agreement between two or more people in the form of code that is immutably stored in the blockchain as part of a block. The responsibility of the smart contract is to facilitate transactions inside the blockchain. The smart contract starts working once a block is mined and it will then execute the transactions. In order to update a smart contract, it seems that a new smart contract gets put on the blockchain and connects to the old contract.



4.2.1: The workings of a smart contract. Source: Lecture 3 slides.

What you can see is that Smart Contracts are basically coded agreements, like the name implies, that occur between people when they would meet with each other.

4.2.1 Use cases of a smart contract

A couple of use cases of smart contracts given by the lecturer.

Trade finance	Smart contracts can be set up as escrow accounts that monitor an exchange between two parties. It can track the location of the goods and when ownership has been transferred it can trigger payments
P2P insurance	Insurance firms can automate the insurance policy by writing it into a smart contract. This technology can enable P2P insurance business models through templated smart contracts.
Loyalty and rewards	E-commerce, retail and travel & tourism are some of the industries that can create a smart contract-driven loyalty and rewards system stored on a distributed ledger allowing interoperability.

Land registry	Store the ownership of land/property on a distributed database and create safeguards for secure updates to this registry when transferring ownership through involvement of government/central body.
Post-trade services	Smart contracts are triggered to ensure regulatory compliance of trades, ensure trade is executed as per the requirements and take corrective steps as needed.

4.3: Ethereum

Ethereum is an open source public Blockchain cryptocurrency with no ownership. Ethereum uses PoW consensus algorithm as well to ensure the integrity of the network. Behind the scenes it uses the Ethereum Virtual Machine (EVM) to execute code and transactions. The EVM's general purpose is to make Bitcoin-like value transactions, like 10 ETH from Thomas to Charlie, or more complex applications. It facilitates transactions to be programmed to do different things and it gives an UI to make smart contracts interactable for the average person.

An Ethereum account has the following properties:

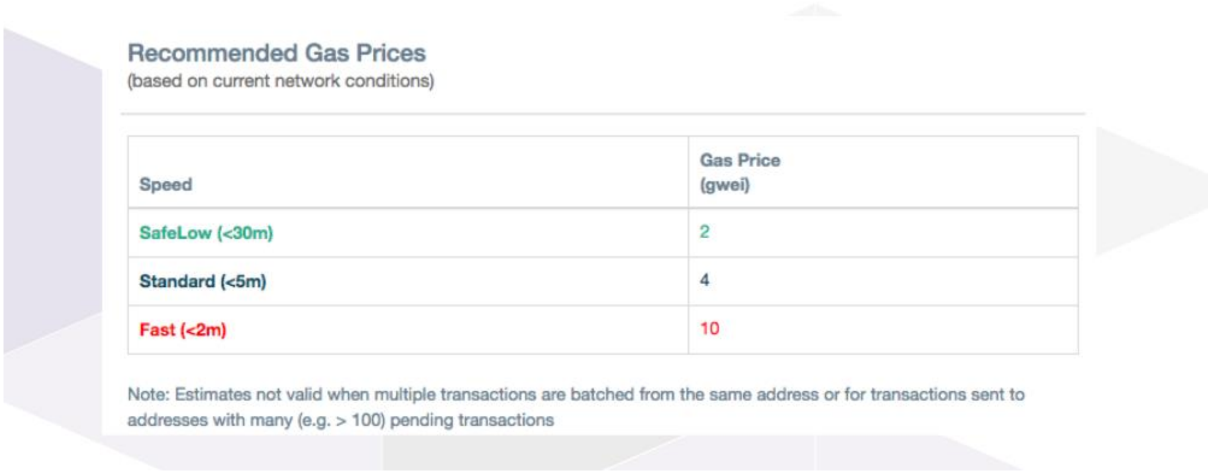
- 20 byte address
- 'State'
- Nonce
- Ether amount
- Contract code
- Memory

You can send a transaction to an external account by creating and signing the transaction. Extern accounts are just other people on the network.
There are also the so called "contract" accounts, these account are controlled by smart contract code.

Making a transaction or submitting contracts in Ethereum cost you something called "gas". Gas is a unit that measures the amount of computational effort that it will take to execute certain operations https://blockgeeks.com/guides/ethereum-gas/

The main purpose of gas is to incentivize miners and discourage storage and attacks. Miners get paid an amount in Ether which is equivalent to the total amount of gas it took them to execute a complete operation.

In image 4.3.1 you can see a couple of options that a user can choose to set the speed of the completion of their transaction.



Recommended Gas Prices
(based on current network conditions)

Speed	Gas Price (gwei)
SafeLow (<30m)	2
Standard (<5m)	4
Fast (<2m)	10

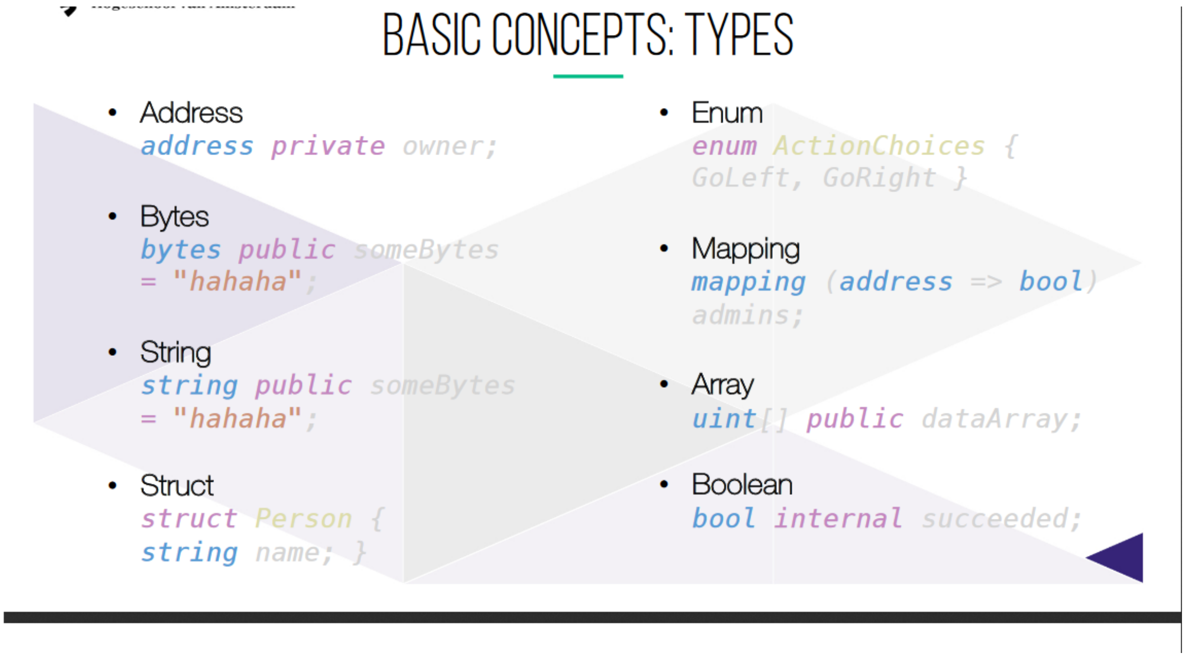
Note: Estimates not valid when multiple transactions are batched from the same address or for transactions sent to addresses with many (e.g. > 100) pending transactions

4.3.1: Gas prices source: http://ethgasstation.info

4.4 Solidity

Smart contracts in Ethereum are written in Solidity. Solidity is a turing-complete language based on C++, Python and Javascript. It has the possibility of inheritance, libraries and complex custom types.

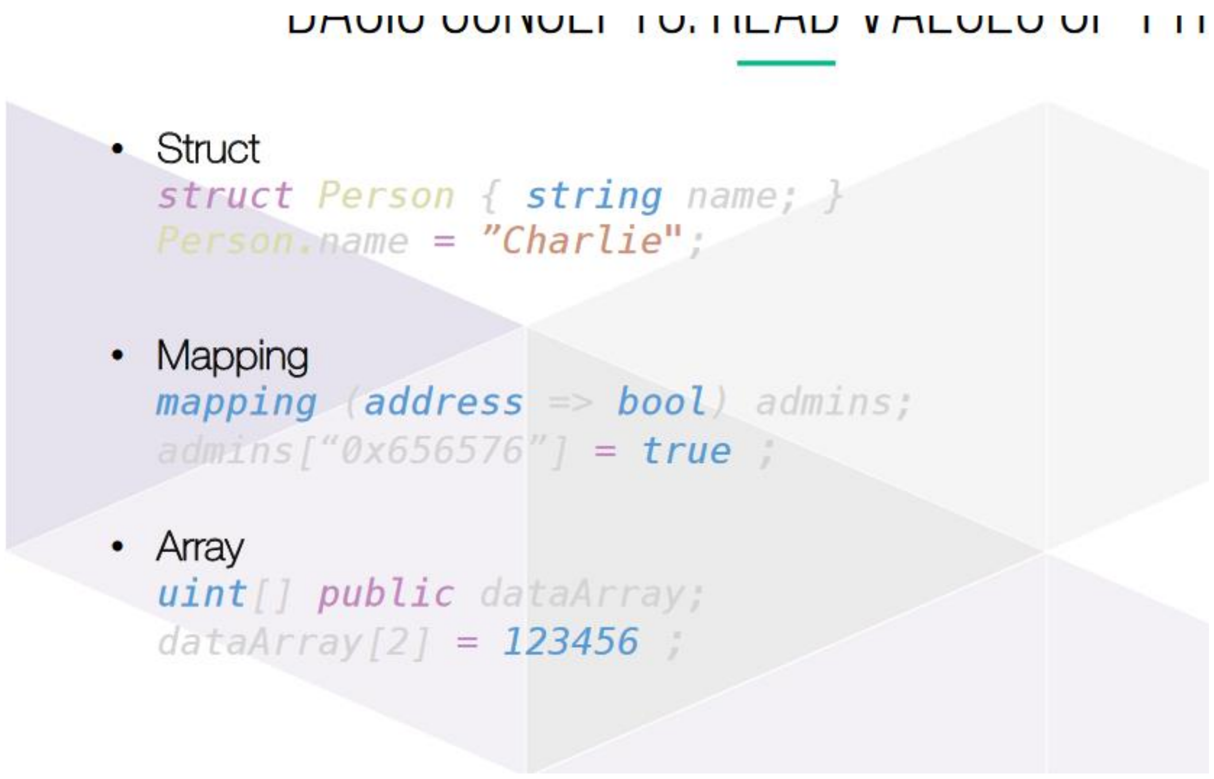
Below you can see a couple of images that showcase some part of Solidity.



BASIC CONCEPTS: TYPES

- Address
```
address private owner;
```
- Bytes
```
bytes public someBytes
= "hahaha";
```
- String
```
string public someBytes
= "hahaha";
```
- Struct
```
struct Person {
string name; }
```
- Enum
```
enum ActionChoices {
GoLeft, GoRight }
```
- Mapping
```
mapping (address => bool)
admins;
```
- Array
```
uint[] public dataArray;
```
- Boolean
```
bool internal succeeded;
```

4.4.1: Data types of Solidity. Source: Lecture 3 slides.



- Struct

```
struct Person { string name; }
Person.name = "Charlie";
```

- Mapping

```
mapping (address => bool) admins;
admins["0x656576"] = true ;
```

- Array

```
uint[] public dataArray;
dataArray[2] = 123456 ;
```

4.4.2: Examples of implementations. Source: Lecture 3 slides.

4.5 Assignment of the week

The assignment for this week was completing the first two lessons of Cryptozombies. The final code can be found below

```
pragma solidity ^0.4.25;

import "./zombiefactory.sol";

contract KittyInterface {
  function getKitty(uint256 _id) external view returns (
    bool isGestating,
    bool isReady,
    uint256 cooldownIndex,
    uint256 nextActionAt,
    uint256 siringWithId,
    uint256 birthTime,
    uint256 matronId,
    uint256 sireId,
    uint256 generation,
    uint256 genes
  );
}
```

```
contract ZombieFeeding is ZombieFactory {

 // 1. Remove this:
 address ckAddress = 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d;
 // 2. Change this to just a declaration:
 KittyInterface kittyContract = KittyInterface(ckAddress);

 // 3. Add setKittyContractAddress method here

 function feedAndMultiply(uint _zombieId, uint _targetDna, string _species) public {
  require(msg.sender == zombieToOwner[_zombieId]);
  Zombie storage myZombie = zombies[_zombieId];
  _targetDna = _targetDna % dnaModulus;
  uint newDna = (myZombie.dna + _targetDna) / 2;
  if (keccak256(abi.encodePacked(_species)) == keccak256(abi.encodePacked("kitty"))) {
   newDna = newDna - newDna % 100 + 99;
  }
  _createZombie("NoName", newDna);
 }

 function feedOnKitty(uint _zombieId, uint _kittyId) public {
  uint kittyDna;
  (,,,,,,,,,kittyDna) = kittyContract.getKitty(_kittyId);
  feedAndMultiply(_zombieId, kittyDna, "kitty");
 }

}
```

```
pragma solidity ^0.4.25;


contract ZombieFactory {

   event NewZombie(uint zombieId, string name, uint dna);

   uint dnaDigits = 16;

   uint dnaModulus = 10 ** dnaDigits;


   struct Zombie {

       string name;

       uint dna;
```

```
        uint year;

    }

    Zombie[] public zombies;

    mapping (uint => address) public zombieToOwner;

    mapping (address => uint) ownerZombieCount;

    function _createZombie(string _name, uint _dna) internal {

        uint id = zombies.push(Zombie(_name, _dna,
(uint(keccak256(abi.encodePacked(_str)) / _dna + _dna)

        / 2));

        zombieToOwner[id] = msg.sender;

        ownerZombieCount[msg.sender]++;

        emit NewZombie(id, _name, _dna);

    }

    function _generateRandomDna(string _str) private view returns (uint) {

        uint rand = uint(keccak256(abi.encodePacked(_str)));

        return rand % dnaModulus;

    }

    function createRandomZombie(string _name) public {

        require(ownerZombieCount[msg.sender] == 0);

        uint randDna = _generateRandomDna(_name);

        randDna = randDna - randDna % 100;

        _createZombie(_name, randDna);

    }

}
```

I added a simple modification to the struct of the Zombie. I included it to have an age which is calculated in the babylonian method of finding the square root of a value.

Screenshot of completion.

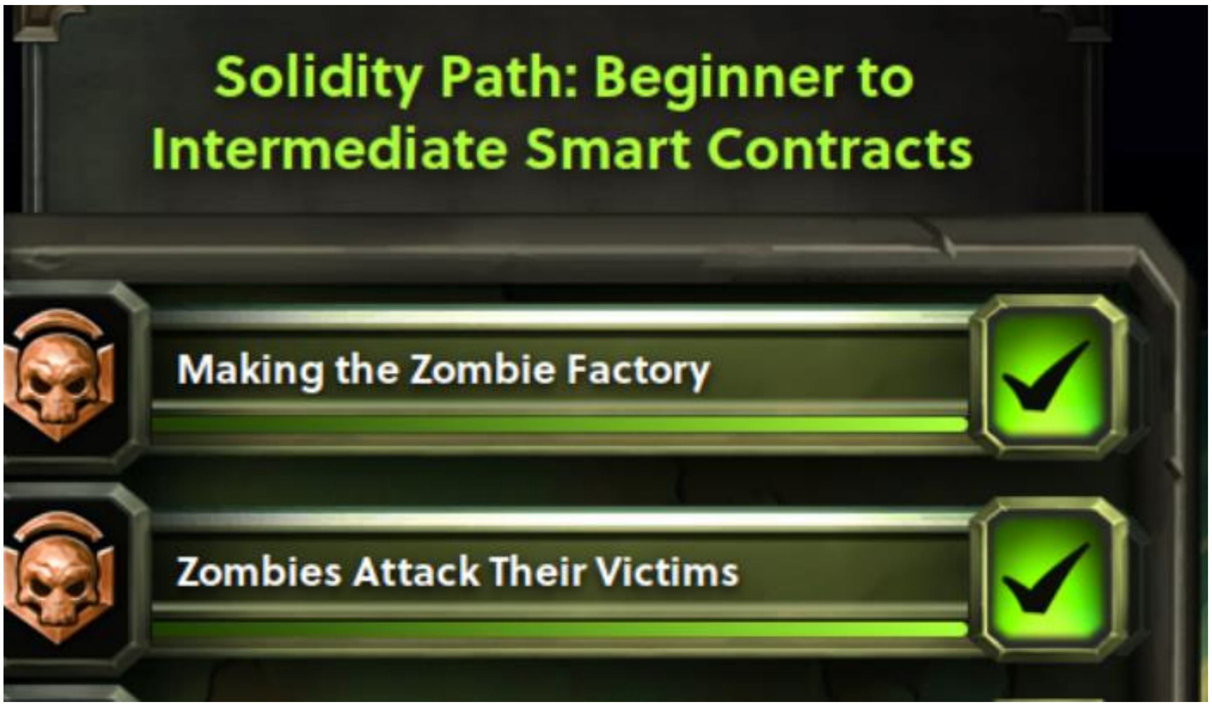

Solidity Path: Beginner to Intermediate Smart Contracts
Making the Zombie Factory
Zombies Attack Their Victims

Figure 4.5.1: Completion of the first two lessons of cryptozombies

4.6 What I learned from this assignment

The differences between the data types of Java and Solidity and how solidity operates. What was quite interesting is the way it handles values specific to that of the sender with msg.sender where you can assign a value to a mapping where the key is the address of the sender.

5. Week 4: Smart contract & Self Sovereign Identity

5.1 Setting up a smart contract on the internet.

This week we started with putting a smart contract online on the public test network of Etherium. First installed the browser extension called Metamask and created an account. Metamask acts as our wallet on the test network[1]. We requested a single Etherium from Ropsten network that we will use to put our smart contract online.
In order to properly test our Smart contract we used the Remix IDE[2] (Integrated Development Environment) for simple testing.

The code we put online and test was a simple contract that allowed us to set and get an integer. The fun part was being able to use the contract of the other classmates and setting the value of their contract to a number we wanted.
While this is quite simple, I accidently send my private key of my wallet to my classmate. Guess I won't be cut out to own a couple BitCoin, as I will likely lose it by leaking my private key.

An interesting fact, requesting information on a Blockchain doesn't cost you anything. It is putting something on the Blockchain, like an integer in a smart contract, that costs you money.

5.2 Self sovereign identity

Alongside putting a simple smart contract online, we also talked about Self Sovereign Identity (SSI). SSI is quite powerful and can be the solution for quite some use cases, especially in Third world countries. According to our lecturer, SSI comes along in three phases, with the first phase being a centralized Identity. This phase occurred in the early days of the internet and saw centralized authorities issuing and authenticate the digital Identity. The most well known example would be a government like the dutch one, where they issues something called DigID to its populace. This allowed you to see into your personal data on the internet and make requests/upload data to the government.

The second phase is called Federated Identity. This phase started somewhere around the turn of the century when a couple of commercial organizations moved towards defragment online identity. This phase sits in between the previous, centralized and the next one, decentralized. The reason for this is because while you can use a single identity for access to multiple website where your preferences are saved per website, the identity comes from a single organization and is only valid for websites that work with the identity. On other websites you will either have to make an account or use another identity/account like your Facebook and Google account.
The most common example for this phase is Google and Youtube. You can use your Google account to login on Youtube and see your subscriptions, liked videos and uploads, while also using the same account for your Gmail and Google Drive.

The third and final phase is the Decentralized Identity. This is where centralized identities got turned into interoperable identities with centralized control through federated design, while respecting a certain level of user consent about how to share an identity and with who. This development was driven

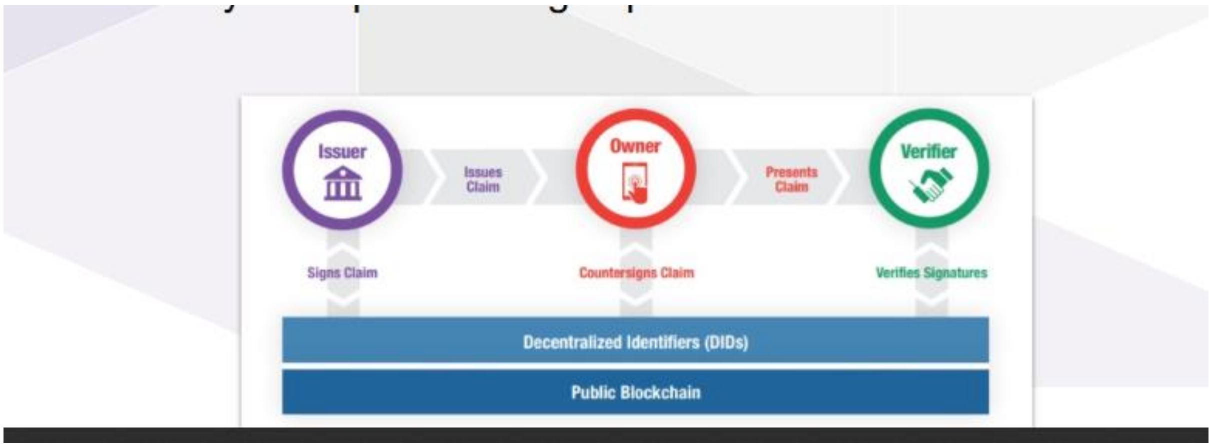
[1] https://faucet.metamask.io/

[2] http://remix.ethereum.org/

by the refugee crisis in Europe where people couldn’t be identified or didn’t even exist according to the documents or rather due to the lack of documents. Due to the refugee crisis we took a step towards true user control of identity, but just a step. The next step is to include user autonomy. When users can be the rulers of their own identity is when we have stepped into the realm of Self Sovereign Identity.

To summarise Self Sovereign Identity consists of the following properties:

- True user control of digital identity
- Interoperability
- Allow users to make claims (like being 18 or older than 18)



5.2.1: Example of the flow of a claim. Source: Lecture 4 slides.

The 10 principles of Self Sovereign Identity:

1. Existence must be independent
2. The user must have control over their identity
3. The user must have access to their own data
4. System and Algorithms that work with SSI must be transparent
5. The identities must be persistent and thus long lived
6. Information and services about the identity must be transportable
7. The identities should be used as widely used as possible, this is interoperability
8. The users must consent themself to the use of their identity
9. The disclosure of claims must be minimized
10. The right of the user must be protected

5.3 A simple use-case for why we should use SSI

Here in the Netherlands, when we want to buy a beer or alcohol, we have to show our identity-card to the seller. While this card contains our age, as needed to purchase the alcohol, it also contains our address, name and our citizen number. So the seller sees far more than he needs to see. This simple action can be the cause of a robbery, burglary or even rape if the seller has malicious intent. The seller just need to know our age, not everything else. So by using SSI, we can simply only allow our age to be visible and minimize the occurrence of such events.

5.4 Potential flow of credentials

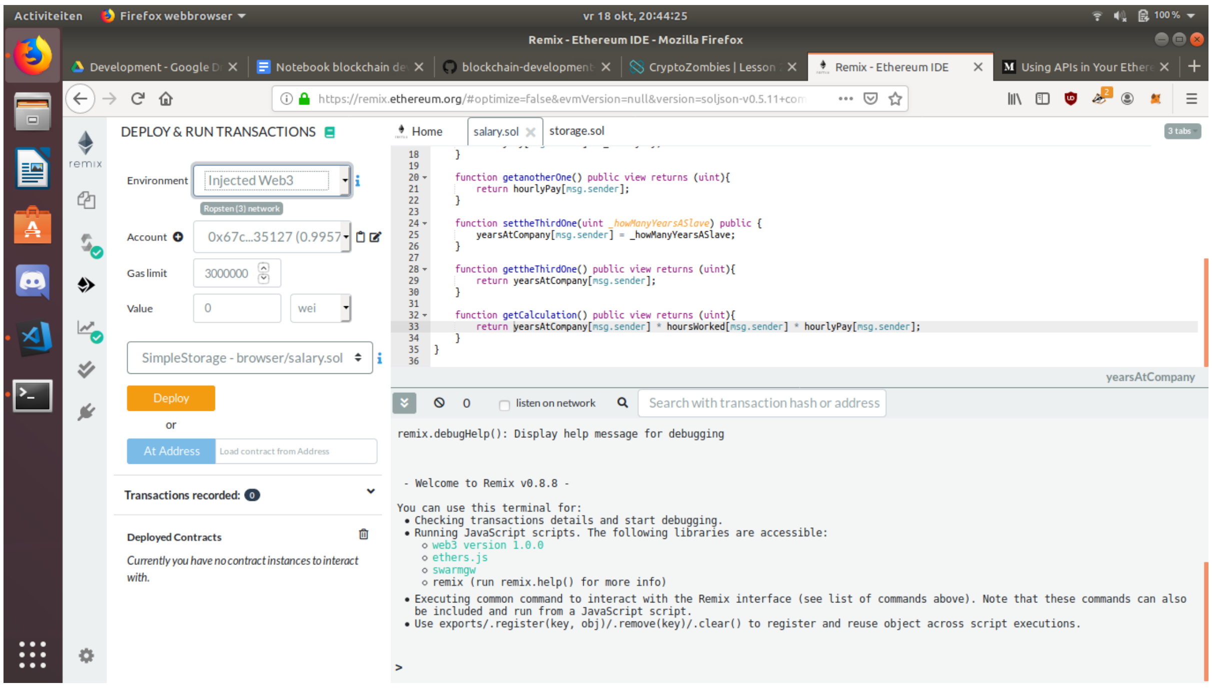
Step	Action
1: Government	Gives your online identity
2: College/University	Confirms your claim that you finished graduating there
3: Company	Job Application
4: Company	Salary stroke
5: ING BANK	Loan application

5.5 Assignment of the week

This week we can either make a live smart contract on the Ropsten testnet or research use cases of smart contracts. I decided to create a live smart contract.

```
pragma solidity >=0.4.22 <0.6.0;


contract Salary {

   mapping (address=> uint) hoursWorked;

   mapping (address=> uint) hourlyPay;

   mapping (address=> uint) yearsAtCompany;


   function setRandomInt(uint _hours) public {

       hoursWorked[msg.sender] = _hours;

   }
```

```
    function getRandomInt() public view returns (uint){

        return hoursWorked[msg.sender];

    }


    function setanotherOne(uint _hourlyPay) public {

        hourlyPay[msg.sender] = _hourlyPay;

    }


    function getanotherOne() public view returns (uint){

        return hourlyPay[msg.sender];

    }


    function settheThirdOne(uint _howManyYearsASlave) public {

        yearsAtCompany[msg.sender] = _howManyYearsASlave;

    }


    function gettheThirdOne() public view returns (uint){

        return yearsAtCompany[msg.sender];

    }


    function getCalculation() public view returns (uint){

        return yearsAtCompany[msg.sender] * hoursWorked[msg.sender] *
hourlyPay[msg.sender];

    }

}
```


DEPLOY & RUN TRANSACTIONS

Environment: Injected Web3 (Ropsten (3) network)
Account: 0x67c...35127 (0.9957
Gas limit: 3000000
Value: 0 wei
SimpleStorage - browser/salary.sol

Transactions recorded: 0

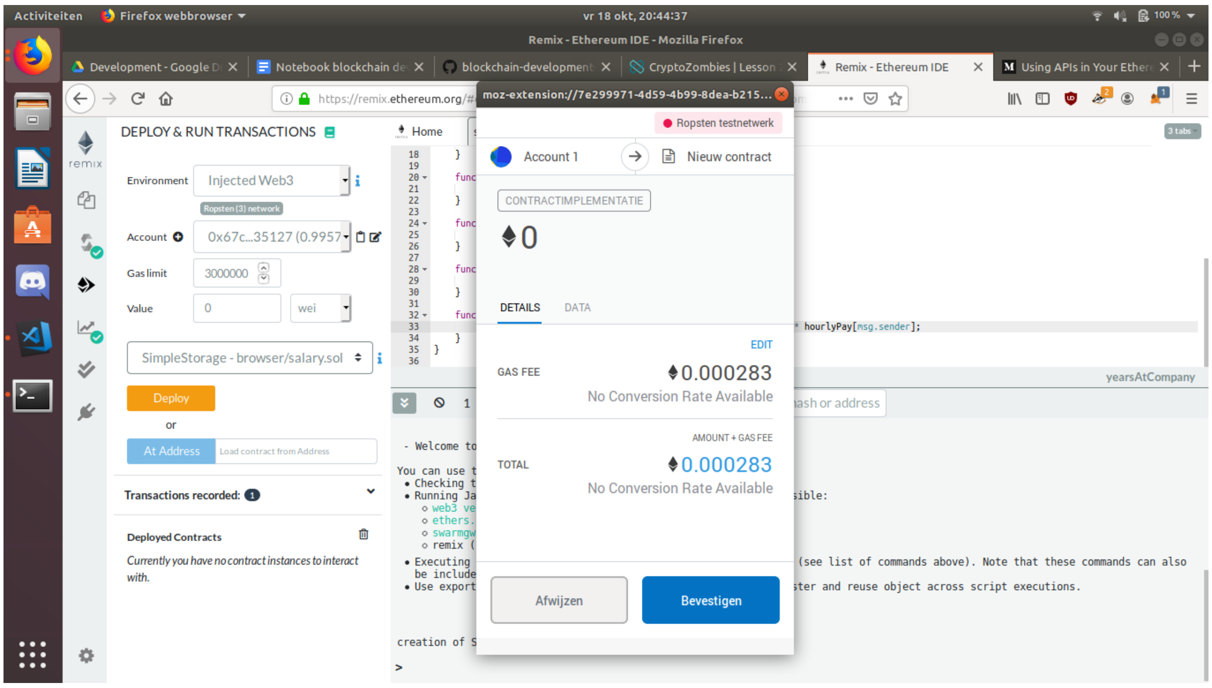
Deployed Contracts

Currently you have no contract instances to interact with.

```
    }

    function getanotherOne() public view returns (uint){
        return hourlyPay[msg.sender];
    }

    function settheThirdOne(uint _howManyYearsASlave) public {
        yearsAtCompany[msg.sender] = _howManyYearsASlave;
    }

    function gettheThirdOne() public view returns (uint){
        return yearsAtCompany[msg.sender];
    }

    function getCalculation() public view returns (uint){
        return yearsAtCompany[msg.sender] * hoursWorked[msg.sender] * hourlyPay[msg.sender];
    }
}
```

```
remix.debugHelp(): Display help message for debugging


 - Welcome to Remix v0.8.8 -

You can use this terminal for:
• Checking transactions details and start debugging.
• Running JavaScript scripts. The following libraries are accessible:
  o web3 version 1.0.0
  o ethers.js
  o swarmgw
  o remix (run remix.help() for more info)
• Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also
  be included and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.

>
```

Figure 5.5.1: Deploying the smart contract.


moz-extension://7e299971-4d59-4b99-8dea-b215...

Ropsten testnetwerk

Account 1 → Nieuw contract

CONTRACTIMPLEMENTATIE

0

DETAILS DATA

EDIT

GAS FEE: 0.000283 — No Conversion Rate Available

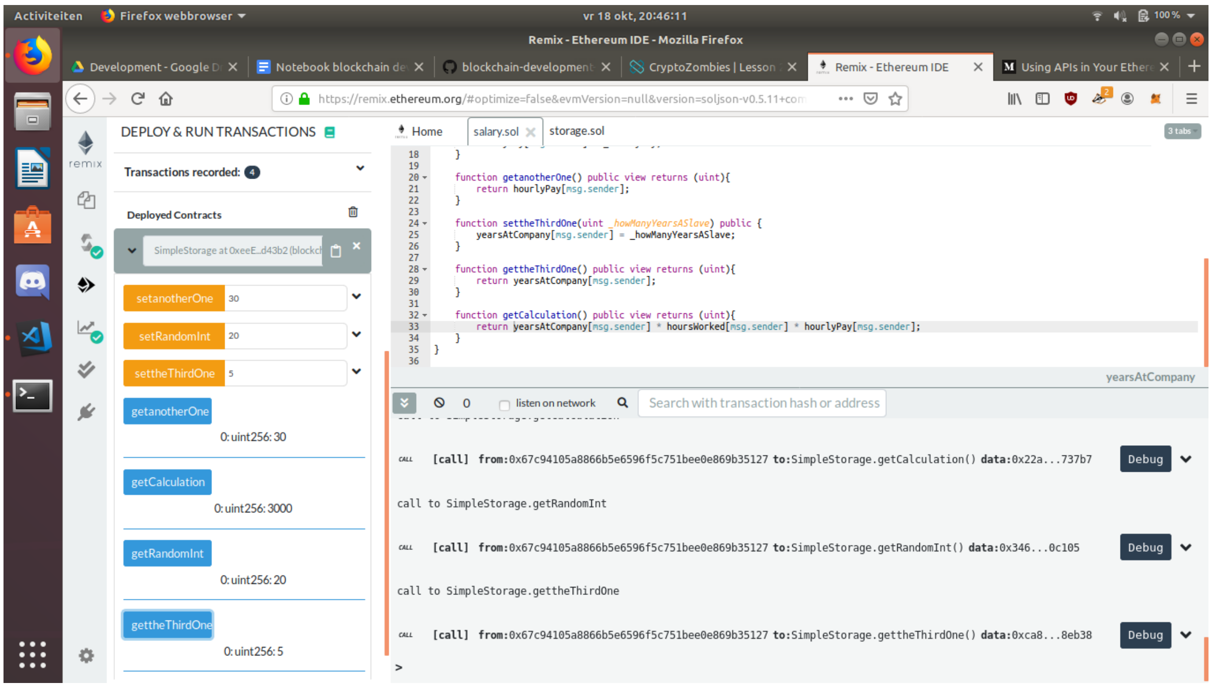
AMOUNT + GAS FEE

TOTAL: 0.000283 — No Conversion Rate Available

Afwijzen | Bevestigen

Transactions recorded: 1

Figure 5.5.2: Paying to put it on the testnet



```
    }

    function getanotherOne() public view returns (uint){
        return hourlyPay[msg.sender];
    }

    function settheThirdOne(uint _howManyYearsASlave) public {
        yearsAtCompany[msg.sender] = _howManyYearsASlave;
    }

    function gettheThirdOne() public view returns (uint){
        return yearsAtCompany[msg.sender];
    }

    function getCalculation() public view returns (uint){
        return yearsAtCompany[msg.sender] * hoursWorked[msg.sender] * hourlyPay[msg.sender];
    }
}
```

```
CALL  [call] from:0x67c94105a8866b5e6596f5c751bee0e869b35127 to:SimpleStorage.getCalculation() data:0x22a...737b7

call to SimpleStorage.getRandomInt

CALL  [call] from:0x67c94105a8866b5e6596f5c751bee0e869b35127 to:SimpleStorage.getRandomInt() data:0x346...0c105

call to SimpleStorage.gettheThirdOne

CALL  [call] from:0x67c94105a8866b5e6596f5c751bee0e869b35127 to:SimpleStorage.gettheThirdOne() data:0xca8...8eb38
>
```

Figure 5.5.3: Proof that it works

5.6 What I learned from the assignment

The main takeaway from this assignment was how to deploy a smart contract on a Blockchain network. I had never done this before and while it wasn't that difficult, it was quite cool to actually deploy something live that other users can use.

6. Introduction to Hyperledger Fabric week 5.

So far we have mainly discussed open public Blockchain, but this week we are learning about Hyperledger Fabric, a private blockchain technology. First we introduce a problem faced by many distributed computer systems, the Byzantine General Problem and have Blockchain handles it. Afterwards we discuss the two types of Blockchain and finally we will talk about Hyperledger.

6.1 Byzantine General problem

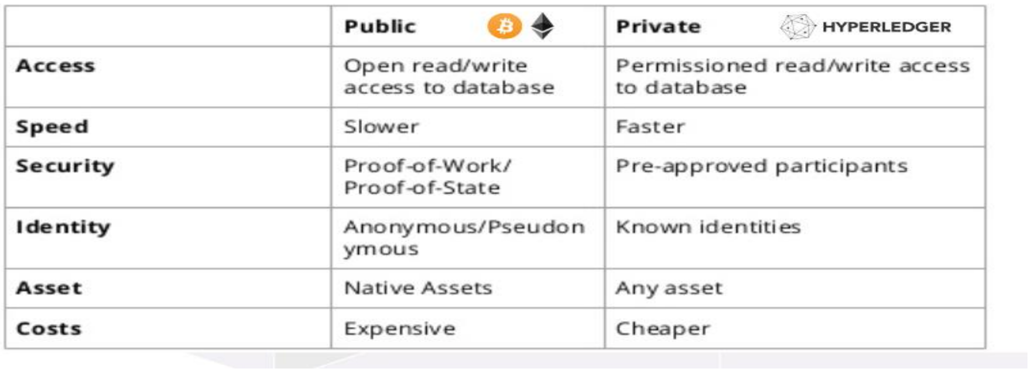
Every single distributed computer system network faces one common problem: the Byzantine general problem. When you got a group of four people giving different answers to a question that you posed, which one do you trust to be correct?
This is the Byzantine general problem. This is something that distributed networks face every day and Blockchain fixes this issue by saying that the entire network must agree upon an answer before it continues. If a few nodes are corrupt, the rest of the nodes are the ones who counteract this by saying that they do not agree, because their data is different. This situation can cause a fork to occur between the nodes. Either the ones who are corrupt create a new chain and hope to drag as many people as possible with them onto the fork or the non-corrupt nodes create a fork and say that if you want to join this fork, you have to agree with the fact that what the corrupt nodes said is not true.

In order to reach consensus, the number of nodes that agree must be 2F+1 in a system containing 3F+1 nodes, where F is the number of faults.
At most (N-1) / 3 out of N amount of nodes are simultaneously faulty.

6.2 Two types of Blockchain

The types of Blockchain can be grouped together into two, namely the open public Blockchain and the permissioned enterprise Blockchains, also known as private Blockchains.
The open and public Blockchain is accessible to everyone and anyone can participate and leave whenever they want. The data of the Blockchain is viewable by everyone and anyone can trace where the transactions came from. Sustaining an open public Blockchain is generally expensive, mainly because of the in chapter 2 discussed Proof of Work, but are secure in the fact that you got a lot of people participating who check everything concerning the Blockchain. As the old saying goes: Safety in numbers.

Permissioned enterprise Blockchains are only accessible through some form of authentication, usually a key. These types of Blockchains are used by organisations to share data amongst themselves when they don’t necessarily trust each other and are quite cheap to maintain.



	Public	Private HYPERLEDGER
Access	Open read/write access to database	Permissioned read/write access to database
Speed	Slower	Faster
Security	Proof-of-Work/ Proof-of-State	Pre-approved participants
Identity	Anonymous/Pseudonymous	Known identities
Asset	Native Assets	Any asset
Costs	Expensive	Cheaper

Figure 6.2.1: An overview of public vs private. Source: Lecture 5 slides

There are circumstances when you do not want to use a blockchain:

- When the process involves confidential data
- The process stores a lot of static data
- The rules of transaction change frequently or in unexpected ways

The first reason speaks for itself, a Blockchain is usually between two or more parties, you don't want those parties to see the confidential data. The second reason is, if the data is static, aka not changing, then why not store it in a file. Blockchain is a great tool when you want the ability to track the changes in data, but not for just storing a fixed dataset.
The third one is a bit more complicated. Rules of a transaction are the basis of Blockchain, they kind of determine the value of the data stored whether it is cryptocurrency or some pdf files, if the rules of transactions change frequently then you can ask if those transactions were ever valid in the first place.

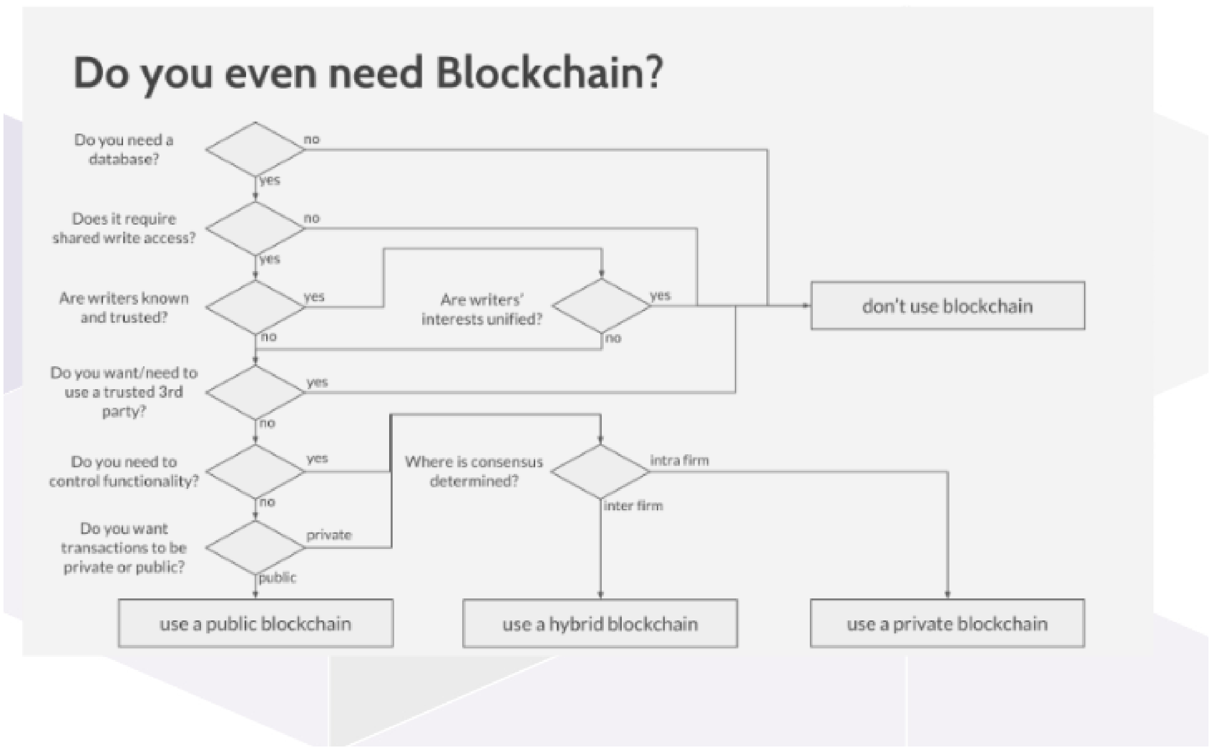

Do you even need Blockchain?
Do you need a database?
no
yes
Does it require shared write access?
no
yes
Are writers known and trusted?
yes
no
Are writers' interests unified?
yes
no
don't use blockchain
Do you want/need to use a trusted 3rd party?
yes
no
Do you need to control functionality?
yes
no
Where is consensus determined?
intra firm
inter firm
Do you want transactions to be private or public?
private
public
use a public blockchain
use a hybrid blockchain
use a private blockchain

Figure 6.2.2: Decision tree if you need a Blockchain or not. Source: Lecture 5 slides

6.3 Introduction to Hyperledger technologies

Hyperledger Fabric is an enterprise-grade permissioned distributed ledger framework for developing solutions and applications. Its modular and versatile design satisfies a broad range of industry use cases. It offers a unique approach to consensus that enables performance at scale while preserving privacy.

6.3.1 What is distributed ledger technology?

Source: https://searchcio.techtarget.com/definition/distributed-ledger

Distributed ledger technology (DLT) is a digital system for recording the transaction of assets in which the transactions and their details are recorded in multiple places at the same time. Unlike traditional databases, distributed ledgers have no central data store or administration functionality.

In a distributed ledger, each node processes and verifies every item, thereby generating a record of each item and creating a consensus on each item's veracity. A distributed ledger can be used to record static data, such as a registry, and dynamic data, i.e., transactions.

Some of the consensus protocols that can be used by these technologies are:

- Proof of work
- Proof of stake
- Proof of activity
- Proof of authority
- Proof of burn

The first two have been discussed in this report already, so I will give a short explanation to the other three.

Proof of activity is actually a hybrid of Proof of Work and Proof of Stake. In Proof of Activity the mining process starts as a standard Proof of Work process with the miners racing against each other to find a new block. When a new Block is mined, the system switches to Proof of Stake, with the mined block containing only a header and the miners reward address.

Based on the header details, a new random group of validators from the blockchain network is selected who are required to validate or sign the new block. The more cryptocoins a validator owns, the more chances he or she has for being selected as a signer.

Once all validators sign the newly found block, it gains the status of a complete block, gets identified and added to the blockchain network, and transactions start getting recorded on it.

Source: https://www.investopedia.com/terms/p/proof-activity-cryptocurrency.asp

Proof of Authority is basically what its name implies. The transactions and blocks are validated by approved accounts. Usually these accounts earn the right to become a validator by their reputation or contribution to the Blockchain. This acts as a deterrent to corruption, because they don’t want their reputation to be tarnished or their work to become useless.

Proof of Burn is similar to Proof of Stake, but instead people send coins to a verifiably unspendable address. The underlying idea is that the consensus methods of Proof of Work, Proof of Stake or Proof of Activity all basically tone down to the miner doing something that the miner finds expensive to do for a new block. So sending money to an unspendable address is the same as the coins can never be used again, this is where the name Proof of Burn comes from.

6.3.2 The different Hyperledger projects and modules

Project	what does it do?
Indy	Identity manager that you can plug into Hyperledger project or other decentral systems
Sawtooth	Modular platform for building, deploying and running distributed ledgers. Has various consensus algorithms. Default is proof of elapsed time. Provides high scalability without high energy usage.
Fabric	Consensus and membership services uses container technology to host chaincode
Burrow	Permissible smart contract machine that provides a modular blockchain client with a permissioned smart contract interpreter
Cello	On-demand deployment of blockchain
Explorer	First Blockchain explorer for permissioned ledger. Operates without compromising member’s security
Composer	Provides a suite of tools for building blockchain business networks. Data modelling language is Java. Generate rest API for interacting Generate a skeleton angular app

6.3.3 Hyperledger Fabric

Hyperledger Fabric has a couple of reason why you would use it. The first reason is the permissioned membership. As explained earlier all participants in Hyperledger Fabric has their identities known to the other participants. It also complies very well with data protection laws that require knowing who the members of the network are and who is accessing specific data.

The consensus protocol is pluggable, meaning you can swap from Proof of Authority to Proof of Burn without too much effort. It also allows you to bring your own built-in identity management or use your company identity.

The transaction process is separated in three phases:

- chaincode processing and agreement
- Transaction ordening
- Transaction validation and commitment

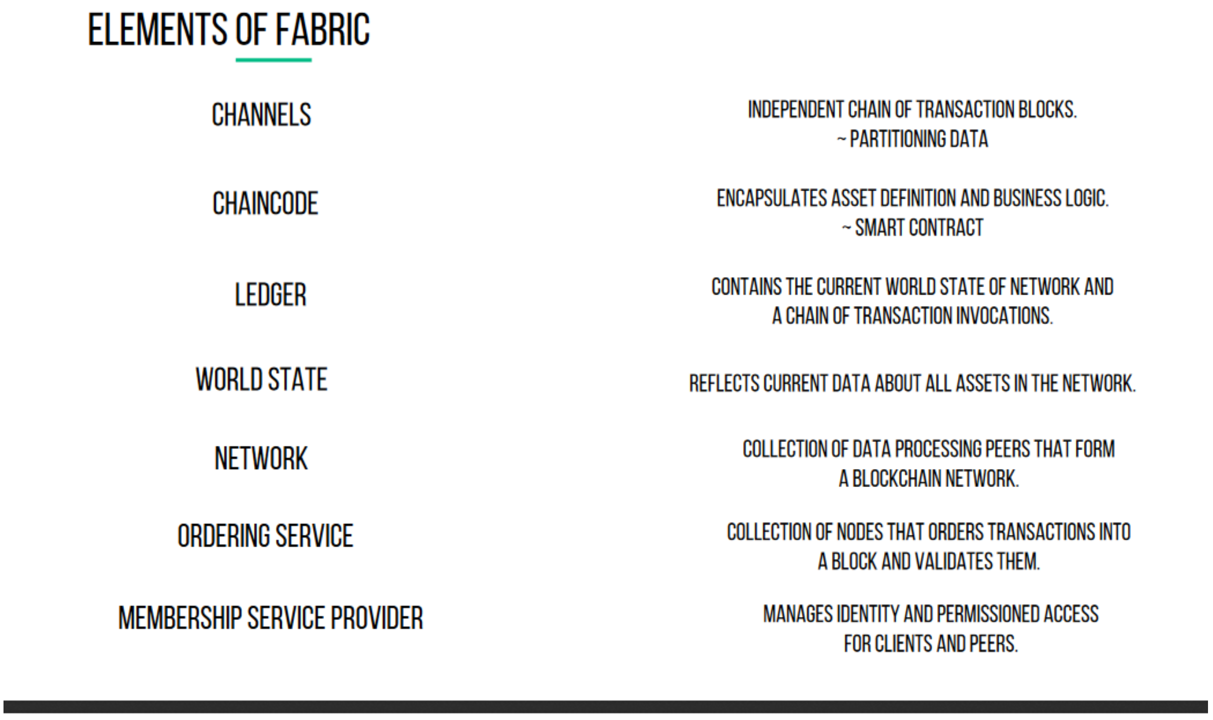
This separation has a couple of advantages:

- Fewer levels of trust and verification are required across nodes
- Network scalability and performance optimized

The data is on a need-to-know basis. This is achieved by partitioning data on the Blockchain. Hyperledger Fabric makes use of channels that enables data to only go towards parties that need to know.

The transactions are committed to the ledger in a set of asset key-value pairs. The advantage of this is that you can look for both keys and values in the so called rich queries. The data is stored in a JSON document and queries are pushed as well in a JSON document style.

Below you can find an image of the elements that Fabric consists out of.



ELEMENTS OF FABRIC

CHANNELS	INDEPENDENT CHAIN OF TRANSACTION BLOCKS. ~ PARTITIONING DATA
CHAINCODE	ENCAPSULATES ASSET DEFINITION AND BUSINESS LOGIC. ~ SMART CONTRACT
LEDGER	CONTAINS THE CURRENT WORLD STATE OF NETWORK AND A CHAIN OF TRANSACTION INVOCATIONS.
WORLD STATE	REFLECTS CURRENT DATA ABOUT ALL ASSETS IN THE NETWORK.
NETWORK	COLLECTION OF DATA PROCESSING PEERS THAT FORM A BLOCKCHAIN NETWORK.
ORDERING SERVICE	COLLECTION OF NODES THAT ORDERS TRANSACTIONS INTO A BLOCK AND VALIDATES THEM.
MEMBERSHIP SERVICE PROVIDER	MANAGES IDENTITY AND PERMISSIONED ACCESS FOR CLIENTS AND PEERS.

Figure 6.3.3.1: The elements that make up the Fabric Blockchain. Source: Lecture 5 slides

6.3.4 Transaction flow of Hyperledger Fabric

The transaction flow will be shown through the use of slides from the lecture of lesson 5.

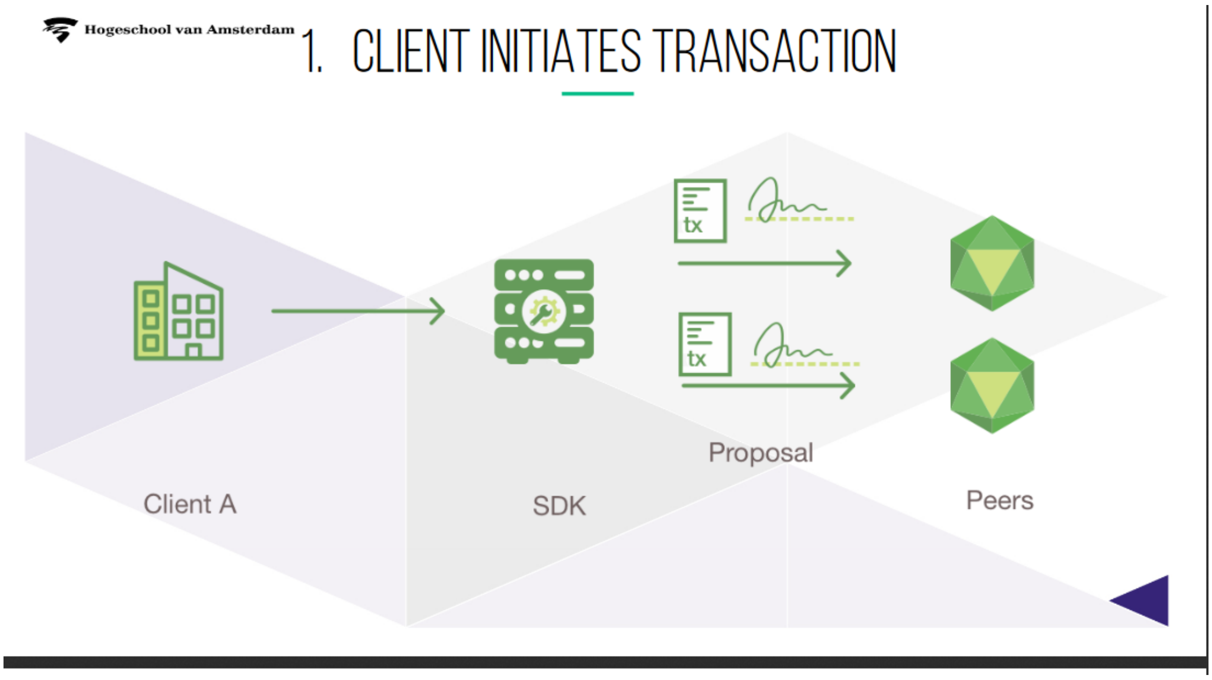


Figure 6.3.4.1: Step 1 of the transaction flow. Source: lecture 5 slides.

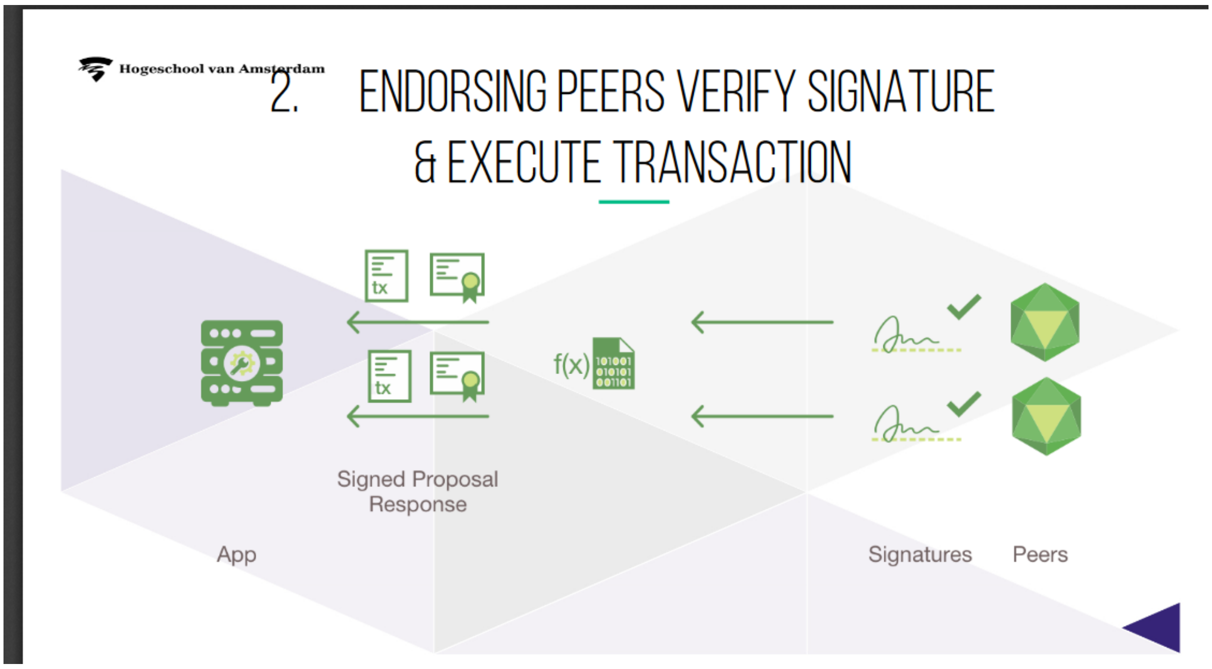


Figure 6.3.4.2: Step 2 of the transaction flow. Source: lecture 5 slides.

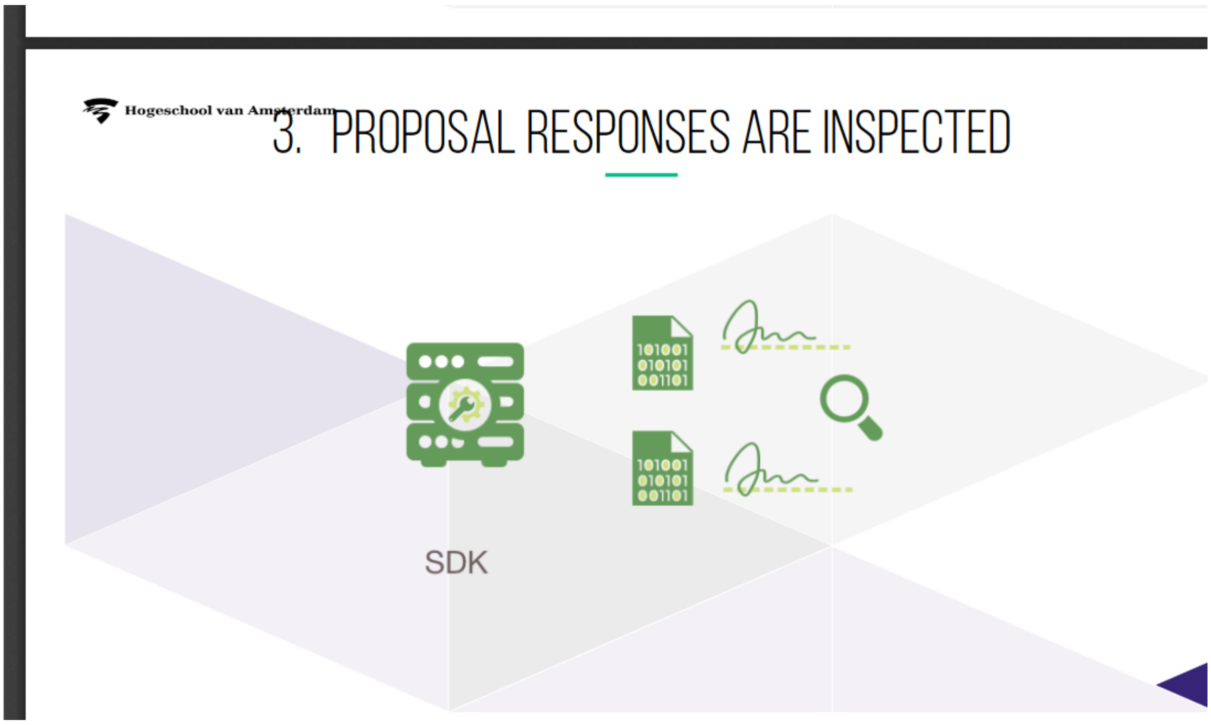


Figure 6.3.4.3: Step 3 of the transaction flow. Source: lecture 5 slides.

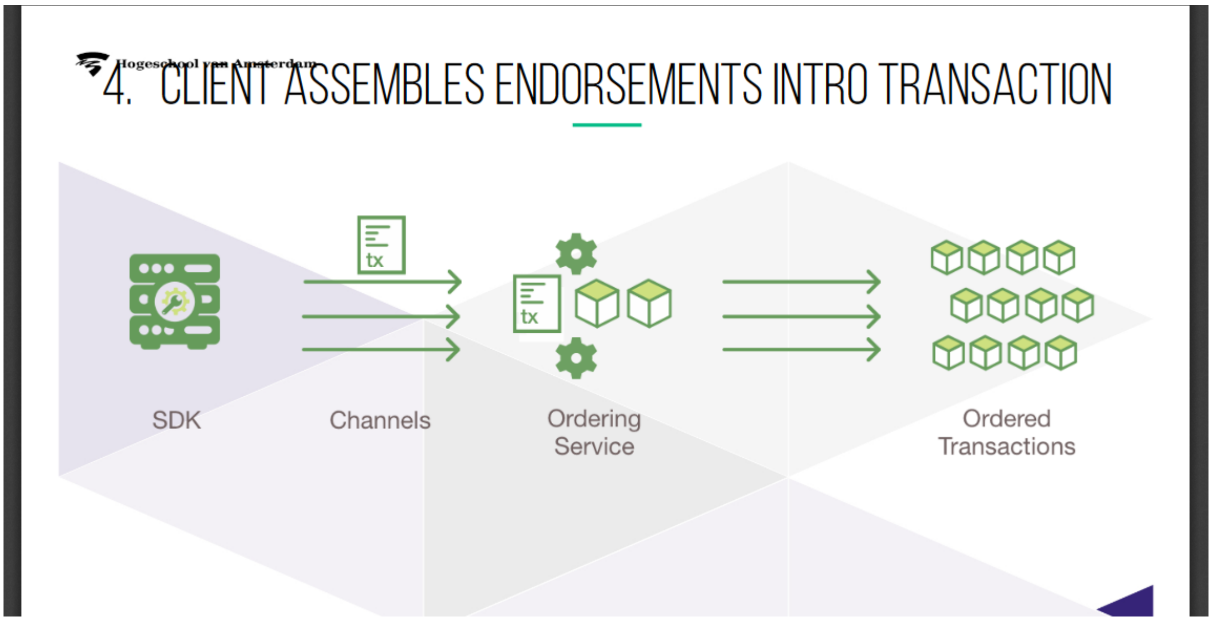


Figure 6.3.4.4: Step 4 of the transaction flow. Source: lecture 5 slides.

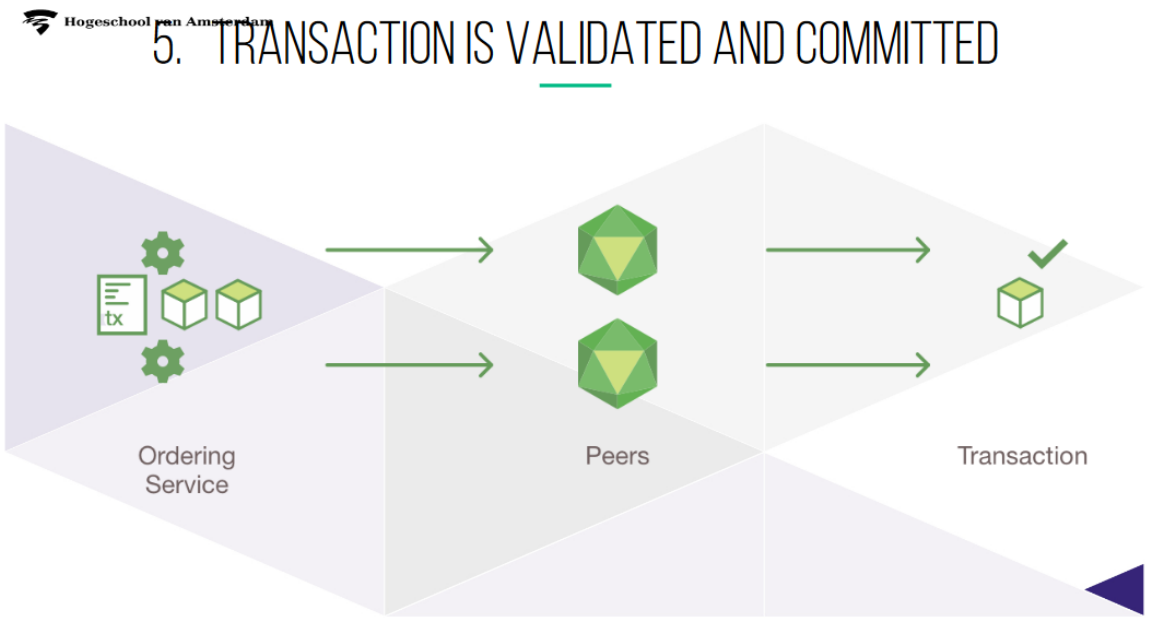


Figure 6.3.4.5: Step 5 of the transaction flow. Source: lecture 5 slides.

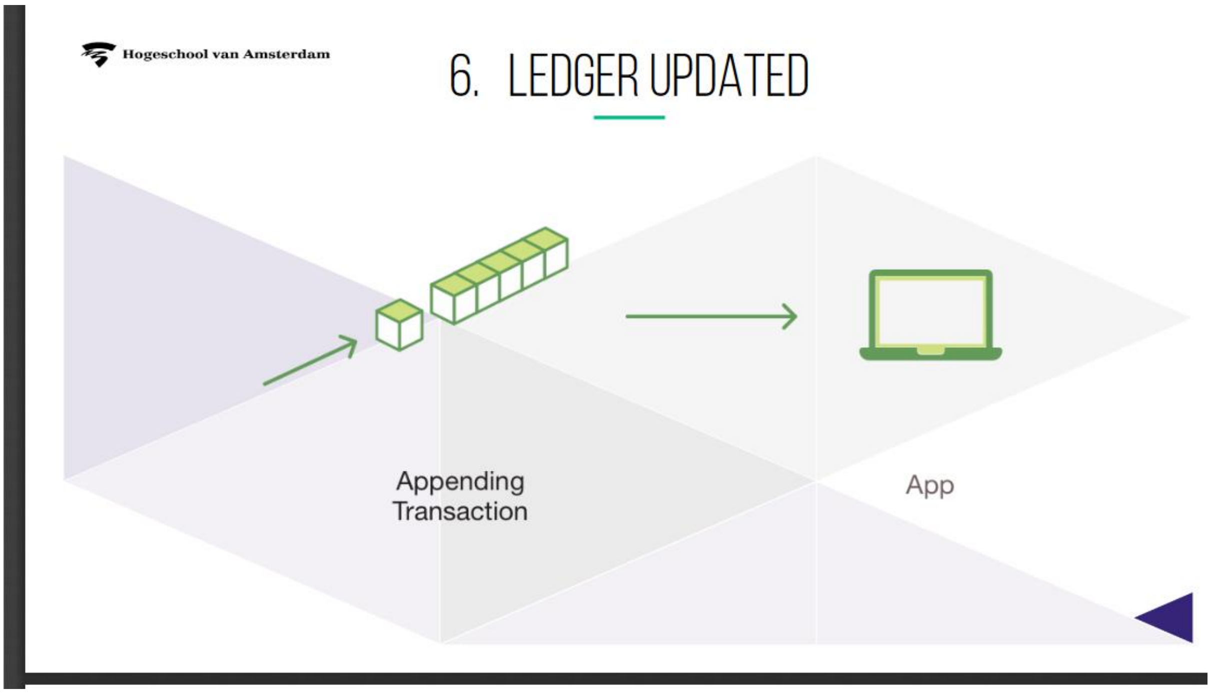


Figure 6.3.4.6: Step 6 of the transaction flow. Source: lecture 5 slides.

6.4 Assignment of the week

This week comes with another assignment. We have to come up with a casus and a solution to that casus. We have to form a theoretical solution with either Ethereum if we want to go with a public Blockchain or Hyperledger Fabric if we go with a private Blockchain.

6.4.1 Casus

We're company KekW and we are a fortune 500 company. Our main product consists of IT applications providing a solution to our customer needs and we offer advice to other companies as well as governments about Big Data. We operate mainly in the US and Europe, but we are expanding into Asia and Africa as well. As a large company we have an enormous amount of contracts from catering and cleaning to temporary employees and office rents. We're looking into the Blockchain technology because we think it is a possible solution to solving our issues with the contracts and payments for the contracts. An example would be the following situation: we order a certain amount of food from our catering service, the catering services deliver the order but the amount is incorrect. The issue could be miscommunication, forgetfulness or the catering service and our office manager differ on the quality of the given delivery or the catering service delivers and places it into the specific location, however, there is no one to verify if they're the ones who delivered the food.

This is just one example, but these issues also happen with the cleaning service or our temporary employees or with payments for our services or payment for services catering to us.

As we have no employees who are knowledgeable about Blockchain, we ask that you write a report for us about the potential Blockchain we can use and advice on other issues that you might come across.

6.4.2 Analysis

As this use case involves a large amount of contracts that contain sensitive information the correct blockchain to use would be a Hyperledger Fabric blockchain. Hyperledger Fabric is an enterprise-grade permissioned distributed ledger framework for developing solutions and applications. Its modular and versatile design satisfies a broad range of industry use cases. It offers a unique approach to consensus that enables performance at scale while preserving privacy.

The data is on a need-to-know basis. This is achieved by partitioning data on the Blockchain. Hyperledger Fabric makes use of channels that enables data to only go towards parties that need to know.

6.4.3 Choice of Blockchain

For this issue we find that a private Blockchain will be the better solution compared to a public Blockchain. The reason we recommend a private blockchain is due to the fact that the actors describes in the casus are all known companies or single entities and there is no need for the public to know about cooperation between yourself and these companies/entities.

A private blockchain offers privacy, as indicated by the name, but it also offers protection within the group of users. As you have a heavy reliance on contracts between yourself and the companies, there is a need to make sure the contracts are shown as the original and agreed upon contracts. Both Blockchain types offer this solution through the inner workings of blockchain, however, as the contracts contains sensitive information that you don’t want everyone to view, it goes in the total opposite direction of what a public blockchain does and thus a private blockchain is the best choice. A public blockchain is a type of blockchain where anyone can join whenever they want and view the data in the blockchain from the beginning till the latest block. Thus a public blockchain is not recommended at all.

Another great reason to pick a private Hyperledger Fabric Blockchain is that it allows you to split the contracts amongst certain peers. For example you can have the contracts that temporary employees have split off from the other companies that take part in the Blockchain in order to prevent potential poaching of these temporary employees.

Another reason that we have chosen for a Hyperledger Fabric blockchain is because of scalability. Scalability in Blockchain has to do with the nodes, which each contain a copy of the blockchain. In a public blockchain once a new block of transactions gets added, every single other node needs to verify this block. This can take an enormous amount of time depending on the scale of the blockchain, but that is already without adding in the PoW (Proof of Work). In traditional public blockchain you need a PoW in order to close a block with transaction, this process can take a considerable amount of time and would reduce the performance of the blockchain by a significant amount.

1. The transaction processing has 3 separate phases:
    1. Distributed logic processing and agreements involving chain codes;
    2. Transaction ordering;
    3. Transaction validation and commit.
2. This ensures fewer levels of trust and validation across different types of nodes, thus reducing overhead;
3. A transaction lifecycle is as follows:
    1. A requester submits a transaction proposal to an endorser;
    2. The endorsement policy specifies the number and combination of endorsers required for this transaction;
    3. The endorser executes chain codes to simulate the proposal to the peers through a ’read/write set‘;
    4. The endorser sends back the signed proposal responses, also called ’endorsements‘;
    5. The client submits a transaction to the orderer with digital signatures;
    6. The orderer creates a block of transactions and sends it to the peers;
    7. The peer checks whether endorsement policy was met and checks for conflicting transactions. When both checks are successful the peer commits the block in the ledger.

6.4.4 Handling the Byzantine Fault Tolerance

In public blockchain there is a term called the Byzantine fault. The term is a condition of a computer system where components may fail and there is imperfect information on whether a component has failed. A couple examples would be:

- Network fail
- Hardware or software components can break or crash
- Malicious components can send malicious messages.

In a blockchain network every non-malicious entity has the same blockchain state and they agree on the state of the blockchain. The implication for Hyperledger Fabric is that the orderer service should be jointly controlled by the network's members using a BFT algorithm that resists malicious activities by bad actors. It's insufficient for one organization to control the orderer, because that organization itself may not be trustworthy. After all, one of the motivations to use blockchain is so that organizations can cooperate while only partially trusting one another (Kynan Rilee, 2018).

The Hyperledger Fabric architecture lets users choose an orderer service that implements a consensus algorithm that fits their application. One desirable property is Byzantine fault tolerance (BFT), which says the orderer can do its job even in the presence of malicious actors (Kynan Rilee, 2018).

It's important to note that BFT, however it's implemented, only applies to the *ordering* of transactions in Hyperledger Fabric. Its job is to ensure that every peer has the same list of transactions on its ledger (Kynan Rilee, 2018).

In order to get on the ledger in the first place, the transactions first have to get past endorsement policies and the endorsing peers that ran the transactions' chain code. Then after they're on the ledger, the transactions are checked a final time by each peer's *validation* step. Every step of the way, Fabric has a system in place to prevent bad things from happening (Kynan Rilee, 2018).

6.4.5 Which consensus protocol

Hyperledger Fabric allows you to plug-in any consensus protocol that you want. I think the best consensus protocol for this casus is the Proof of Authority algorithm. Proof of Authority is basically what its name implies. The transactions and blocks are validated by approved accounts. Usually these accounts earn the right to become a validator by their reputation or contribution to the Blockchain. But for this casus the accounts can be mid-level managers that check the data of their own company as well as the data that the others put on the ledger.

Another possibility is the hashgraph algorithm. With hashgraph the transactions on the nodes get shared among random nodes at random intervals. This leads to all transactions eventually being shared around.

6.4.6 Potential bottlenecks

One potential bottleneck is that if a lot of transactions occur quite quick, it will take a couple seconds for the ledger to be updated. According to research, Hyperledger Fabric 1.0 cannot handle more than 10.000 concurrent transactions when the number of nodes exceeds 6 (Nasir Q, 2018).

Another potential bottleneck is that upgrades to Hyperledger Fabric take time, so if your desire to add more clients to the Blockchain ramps up a lot and the transactions follow that, the Blockchain won't be able to keep up. While I don't foresee 10.000 transactions happening simultaneously, it is a good idea to keep this in the back of your head.

6.4.7 Greatest weakness

As explained in the previous chapter, Hyperledger Fabric 1.x cannot handle 10.000 concurrent transactions with more than 6 nodes. This means that if you plan to have this many transactions happen concurrently, you might run into the problem that people will find out there are only 6 nodes. With Blockchain it is the case that the more nodes you have, the safer it is. Thus people might claim that the Blockchain is not safe and leave or refuse to work with the Blockchain.

6.4.8 Summary

The casus has a number of entities that can be somewhat trusted and the data that the blockchain will contain is sensitive data. As we don't want people without permission to view the data, we need to make sure only entities who we want to enter the blockchain can be inside the blockchain. These conditions make it quite clear the best type of blockchain to use is a private blockchain.

The private blockchain that we recommend is Hyperledger Fabric. Hyperledger Fabric allows you to create channels to make sure people who need to know the data can see it. Compared to a public blockchain, a Hyperledger Fabric private blockchain has good scalability. This comes in the form of the option of choosing your own PoW algorithm and the verification of transactions is done by endorsement policies and the endorsing peers that ran the transactions chain code rather than by every node on the blockchain. After they're added to the ledger they get checked one final time Hyperledger Fabric through the validation steps.

6.5 What I learned from this assignment

How to apply the knowledge I have so far acquired to write an advisory text about which type of Blockchain to use. It was quite difficult and even now I am not sure if I've done the assignment correctly.

7. Hyperledger fabric development week 6.

This week was setting up a Hyperledger Fabric network to perform our last assignment. As such, this week I will include the commands that help set this up and I will shortly explain what they do.

Before we could setup the network we needed to have the following installations: Docker and Node.

7.1 commands

Commands	What it does
"curl -sSL http://bit.ly/2ysbOF E | bash -s"	This command does the following: ● Clone the Hyperledger/Fabric-samples repository ● Checkout the appropriate version tag ● Install all the Hyperledger fabric platform-specific binaries and config files for the version specific into the /bin and /config directories of fabric-samples ● Download the Hyperledger Fabric docker image
CD Fabric-samples/first-network	Change to the directory
Byfn.sh	Leverages the docker images to bootstrap a hyperledger fabric network that by default is comprised of four representing two different organizations, and an orderer node. It will also launch a container to run a scripted execution that will join peers to a channel, deploy a chaincode and drive execution of transactions against the deployed chaincode.
./byfn.sh generate	generates the artifacts required for the network. The artifacts are the: ● Ledger ● Peers (4) ● 2 Organisations ● The orderer ● World state database (CouchDB in this case) ● The Blockchain ● The chaincode
./byfn.sh up -l node	Brings up the network

./byfn.sh down	Brings the network down
./startFabric.sh javascript	Spins up a blockchain network and install and instantiate a javascript version of the fabcar smart contract which will be used by our application to access the ledger
docker ps	checks the running containers
npm install	Will install all dependencies when you switch to the Fabcar/javascript subdirectory in the fabcar-samples repo
node enrollAdmin.js	Stored the CA administrators credentials in the wallet directory
node registerUser.js	Enroll a user now that you have admin credentials in a wallet
node query.js	Applications read data from the ledger using a query. The most common queries involve the current values of data in theledger–its world state. Run the query.js program to return a listing of all the cars on the ledger. This program uses the User1 identity to access theledger

7.2 Imagery of the slides

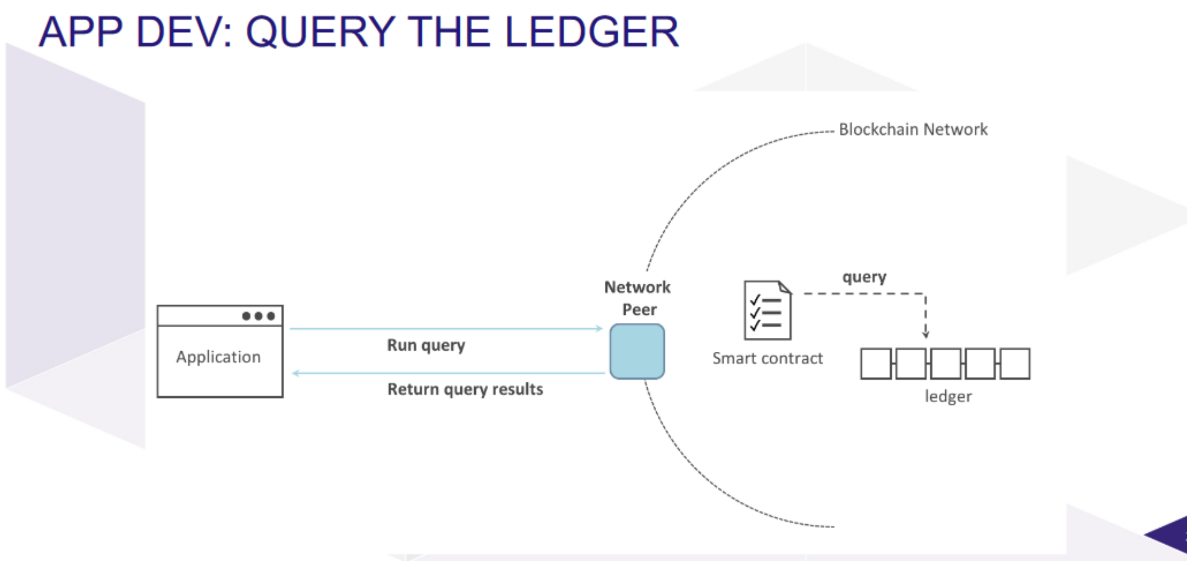
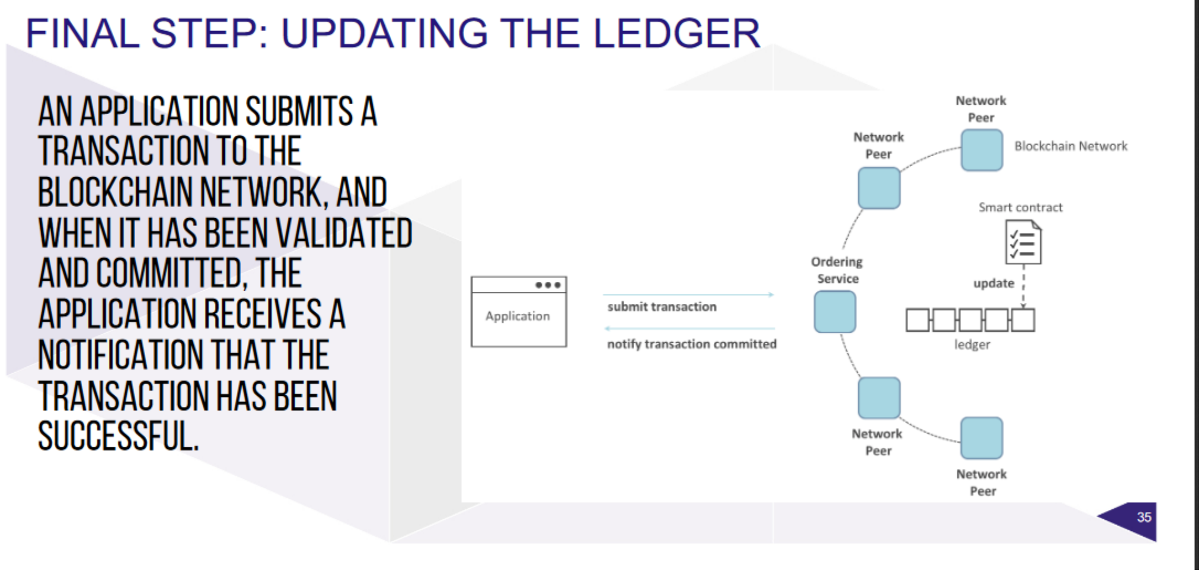
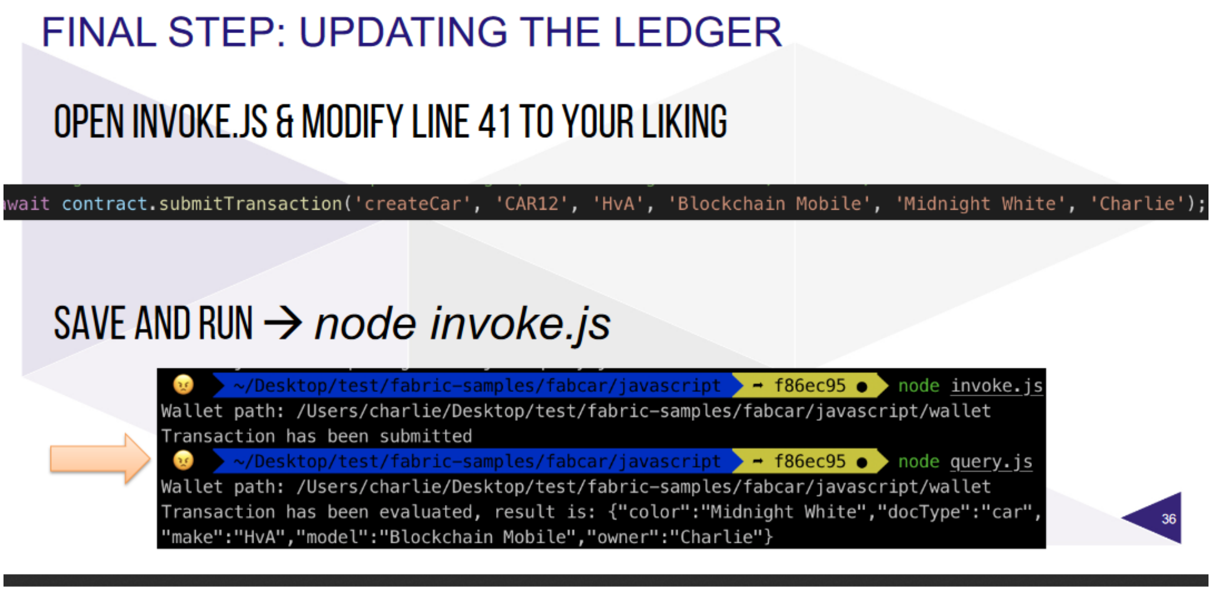

APP DEV: QUERY THE LEDGER

Figure 7.2.1: Flow of the query. Source: Slides lecture 6


FINAL STEP: UPDATING THE LEDGER

AN APPLICATION SUBMITS A TRANSACTION TO THE BLOCKCHAIN NETWORK, AND WHEN IT HAS BEEN VALIDATED AND COMMITTED, THE APPLICATION RECEIVES A NOTIFICATION THAT THE TRANSACTION HAS BEEN SUCCESSFUL.

Figure 7.2.3: Flow of updating the ledger. Source: Slides lecture 6


FINAL STEP: UPDATING THE LEDGER

OPEN INVOKE.JS & MODIFY LINE 41 TO YOUR LIKING

```
wait contract.submitTransaction('createCar', 'CAR12', 'HvA', 'Blockchain Mobile', 'Midnight White', 'Charlie');
```

SAVE AND RUN → *node invoke.js*

```
 ~/Desktop/test/fabric-samples/fabcar/javascript  f86ec95 ●  node invoke.js
Wallet path: /Users/charlie/Desktop/test/fabric-samples/fabcar/javascript/wallet
Transaction has been submitted
 ~/Desktop/test/fabric-samples/fabcar/javascript  f86ec95 ●  node query.js
Wallet path: /Users/charlie/Desktop/test/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"color":"Midnight White","docType":"car",
"make":"HvA","model":"Blockchain Mobile","owner":"Charlie"}
```

36

Figure 7.2.3: Adding a transaction and checking it with query.js Source: Slides lecture 6

7.3 Assignment of the week

Add a second user to the hypFab network

modify the fabcar.js smart contract for the following:

- Create a method that returns all cars of a certain color.
- Add a“year” car property, along with a method to query cars between a year range
- Use thechangeCarOwner() method to switch car ownership from user1 to user2 (Hint:invoke.js)

7.3.1 Adding a user to the network

```
const secret = await ca.register({ affiliation: 'org1.department1',
enrollmentID: 'user2', role: 'client' }, adminIdentity);

        const enrollment = await ca.enroll({ enrollmentID: 'user2',
enrollmentSecret: secret });

        const userIdentity = X509WalletMixin.createIdentity('Org1MSP',
enrollment.certificate, enrollment.key.toBytes());

        await wallet.import('user2', userIdentity);

        console.log('Successfully registered and enrolled admin user
"user2" and imported it into the wallet');
```

I do think that this code should work, however, I get the following error: Failed to register user "user2": Error: fabric-ca request register failed with errors [[{"code":20,"message":"Authentication failure"}]

What I did was basically changing user1 to user2 everywhere. I am not sure how in-depth this assignment should go into configuration and security setup of the network, but I think there is something drastically wrong.

7.3.2 Find cars of specific colour

First we add another car to the ledger with

```
       await contract.submitTransaction('createCar', 'CAR13', 'Honda',
'Accord', 'Black', 'Thomas');
```

```
async findCarsWithColour(ctx, carColour){

       const carAsBytes = await ctx.stub.getState(carColour); // get the
car from chaincode state

       if (!carAsBytes || carAsBytes.length === 0) {

           throw new Error(`${carColour} does not exist`);

       }

       console.log(carAsBytes.toString());

       return carAsBytes.toString();

   }
```

The possible method to find cars with a specific colour. I couldn't test this due to errors. Basically we look through the JSON with the getState, which finds any car that has a value that matches the car colour.

```
async createCar(ctx, carNumber, make, model, color, owner, year) {

       console.info('============= START : Create Car ===========');


       const car = {

           color,

           docType: 'car',

           make,

           model,
           owner,

           year,

       };


       await contract.submitTransaction('createCar', 'CAR13', 'Honda',
'Accord', 'Black', 'Thomas', '1994');
```

Added the year attribute to the car const and a submission of a car that has a year value.

```
async findCarsWithYear(ctx, carYear){

        const carAsBytes = await ctx.stub.getState(carYear); // get the car
from chaincode state

        if (!carAsBytes || carAsBytes.length === 0) {

            throw new Error(`${carYear} does not exist`);

        }

        console.log(carAsBytes.toString());

        return carAsBytes.toString();

    }
```

Find any car that has a value that matches with the given carYear parameter.

```
await contract.submitTransaction('createCar', 'CAR15', 'Honda', 'Accord',
'Black', 'user1');

        await contract.submitTransaction('changeCarOwner', 'CAR15',
'user2');
```

First we add a car that belongs to user1 and then call upon the changeCarOwner within invoke.js with the corresponding values.

Unfortunately I cannot test anything, because I get the following errors:

2019-10-18T17:56:00.665Z - error: [Channel.js]: Channel:mychannel received discovery error:access denied

```
2019-10-18T17:56:00.666Z - error: [Channel.js]: Error: Channel:mychannel Discovery error:access
denied

2019-10-18T17:56:00.676Z - error: [Channel.js]: Channel:mychannel received discovery error:access
denied

2019-10-18T17:56:00.676Z - error: [Channel.js]: Error: Channel:mychannel Discovery error:access
denied

2019-10-18T17:56:00.676Z - error: [Network]: _initializeInternalChannel: Unable to initialize channel.
Attempted to contact 2 Peers. Last error was Error: Channel:mychannel Discovery error:access denied

Failed to evaluate transaction: Error: Unable to initialize channel. Attempted to contact 2 Peers. Last
error was Error: Channel:mychannel Discovery error:access denied
```

8. My opinion/feedback about the course blockchain development.

I think the Blockchain development course was very informative about how Blockchain works. I think the biggest downside of this course was the slow pace at the development side due to the minor being open to non-ICT students. If the minor becomes more popular, perhaps you can look into splitting a class into two groups, where one group is full of non-ICT students and the other is full of them. This does bring some problems with it, but I really think this course could be a bit tougher on the ICT-students.

If we're talking about teaching the course, I think Charlie did an amazing job being excited about the course and subject, informative and generally his explanation of the subjects.

Bibliography

Kynan Rilee. (2018, February 15). *kokster/understanding-hyperledger-fabric-byzantine-fault-tolerance-cf106146ef43*. Retrieved from https://medium.com: https://medium.com/kokster/understanding-hyperledger-fabric-byzantine-fault-tolerance-cf106146ef43

Nasir Q, Q. A. (2018). *Performance Analysis of Hyperledger Fabric Platforms.* Wiley Hindawi.

Solarwinds msp. (2018, November 30). *blog/centralized-vs-decentralized-network*. Retrieved from solarwindsmsp.com: https://www.solarwindsmsp.com/blog/centralized-vs-decentralized-network

UK Government Chief Scientific Adviser. (2016). *Distributed Ledger Technology: beyond block chain.* UK government.