**Blockchain based Ticketing solution for Sporting Arenas**

Submitted in partial fulfillment of the requirements for the degree of

Masters of Electrical and Computer Engineering

in

Department of ECE

Sharath Srikanth Chellappa

M.S., Engineering Department, Electrical and Computer Engineering

Carnegie Mellon University Pittsburgh, PA

*May,* 2020

**ACKNOWLEDGEMENTS**

I would like to thank Professor Priya Narasimhan for helping conceive and bring to conclusion this unique problem statement combining the concepts of Blockchain and Distributed Authentication. This work is first of a kind with no previous work done in this area. Hence any work that we plan to do will be the first in this sphere. The given work has been completed by using Nexus Devices as the Peers in the Network and a One Plus 6T as the main authenticator.

**ABSTRACT**

**PROBLEM STATEMENT AND MOTIVATION**

This problem statement can be better explained with the help of an example. Let us consider the case of the Pittsburgh Penguins stadium. The seating capacity of the stadium is 19,758. Now considering an average of 3 seconds for each ticket being scanned in the stadium, this brings the total time spent on scanning all the tickets are 39516 seconds (659 minutes). This is assuming that there is only one scanner available for scanning each ticket.

Now let us consider the case where in there are 5 entrances into the stadium with 3 scanners being placed at each entrance. Now that would mean that there are 15 scanners to take care of the scanning of the total set of tickets. The total time taken to scan and authenticate each ticket holder would be 33 minutes.

This however is a stadium with a seating capacity of just 20,000. Assuming a seating capacity of 100000, with about 10 entrances and 6 scanners per entrance, it takes us approximately 84 minutes (1hr and 24 mins) for scanning all the tickets in a sold out stadium.

Although 33 minutes / 84 minutes is a fairly good time, in today’s day and age, we must understand means and ways of decreasing the time and improve user experience for the user as it is they who have to wait in order to get in to watch their favorite sport.

Our idea proposes to use the concept of Group Based Authentication in a peer distributed fashion without the involvement of a Central Server. In this case, if a scanner authenticates User 1, he is made a part of the trusted chain thereby giving him access to authenticate anyone post this. Hence the number of authenticators increase on the fly thereby greatly decreasing the time. Considering a small number of ticket holders (say 100), let us assume that we start with just 1 scanner. When 1 ticket holder has been scanned, then the number of scanners are now 2. This way let’s assume that the ratio of a ticket holder to turn into an authenticator is 0.4 (i.e. 4 out of 10 ticket holders go ahead to authenticate other people). In this case, the sequence can then be simplified to the following: Number of scanners (Number of tickets authenticated).

1 (0 authenticated), 2 (2 authenticated), 4 (4 authenticated), 4 (8 authenticated), 6 (12 authenticated), 6 (18 authenticated), 8 (24 authenticated), 12 (32 authenticated), 16 (44 authenticated), 24 (60 authenticated), 32 (84 authenticated), 40 (116 authenticated) and so on.

So if you notice in this case, the amount of time taken to authenticate 100 people (assuming 3 seconds per person), starting out at 1 scanner is going to be 36 seconds. The same scenario without our group-based authentication would be 300 seconds (5 minutes). The time taken in this approach is reduced to 1/10th in comparison to the traditional approach.

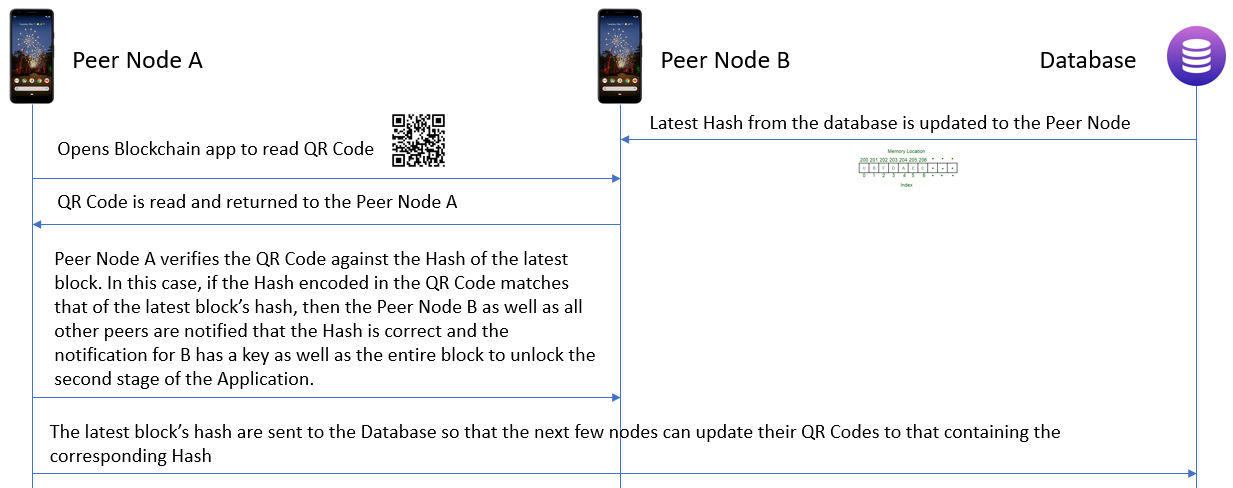
**SOLUTION PROPOSED**

Details of our solution are described in this section. We propose a blockchain based authentication technique where in each party in the system is termed as Nodes with there being 2 broad types of each. Node A and Node B. Node A is the primary authenticator in the System. What this means is that the first ever ticket to be authenticated in the system will be authenticated by a Node A peer. Please note that there is only one type of Node A peer in the system. Every node which gets authenticated and can now further authenticate fall in to the Node B category. The Node B first displays the latest block’s QR code which when authenticated allows the Node B’s application to move into the 2nd Screen (QR Code Reader). Details on the architecture are displayed in the next Section. The database used for the POC is the Firebase Database as that is the easiest Real Time Database to interface with the Android Application.

The calculation of the Hash along with the meaning of the word difficulty will be explained in the later sections. In the Project, we are using One Plus 6T as Node A, while using Google Nexus as Node Bs. In the real world scenario, Node A would be a Scanner while Node B would be a ticket holder. As soon as the scanner scans the QR code on the ticket, it would make Node B an Authenticator which can go ahead and authenticate other Node Bs.

**ARCHITECTURAL APPROACH**

For this system to work, we need to design our architectures in such a way that they do not have any central server. This made me consider the following architectures:



First Architecture:

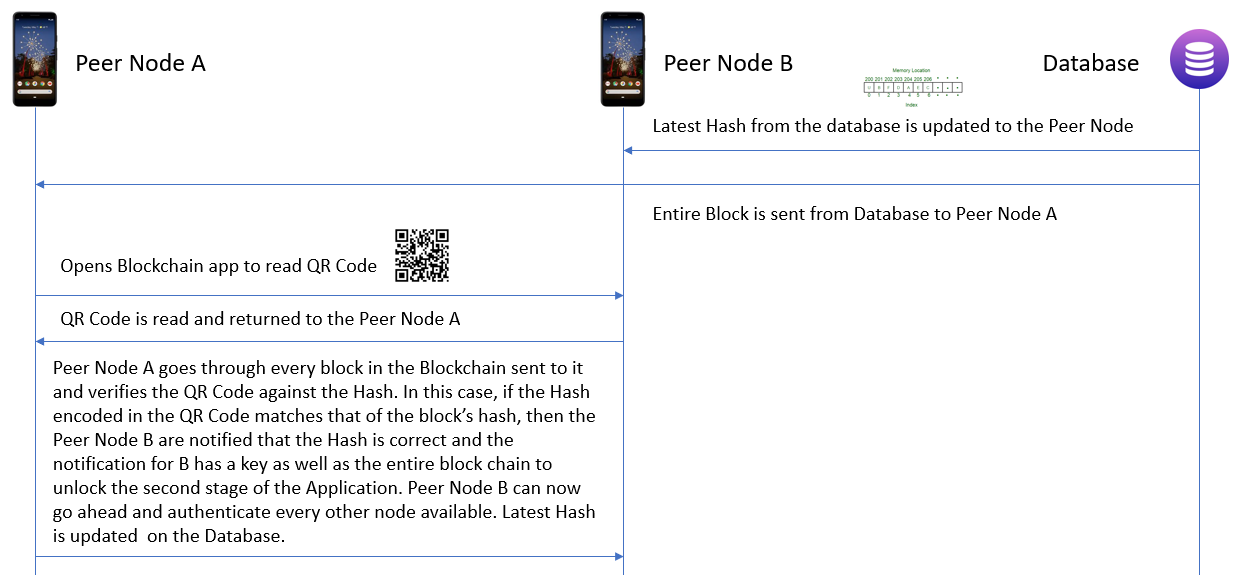
1. The latest hash (in the case of the first authentication, it is a randomized string satisfying the constraints of the difficulty), is stored in the Firebase.
2. As soon as Peer Node B sees changed value in Database, it updates the QR code to that of the latest hash.
3. Peer Node A reads the QR code of Node B.
4. Peer Node A compares the QR Code against the latest Block in the hash and verifies if the QR Code is correct or not. If it is correct, Peer B along with all other peers get notification of the new block to be added along with a key to Peer B to unlock it’s QR Code Scanner portion.
5. Peer Node A also updates the Hash in the Firebase Database so that every other peer yet to be authenticated can update their QR code.

Advantages:

* Peer-Peer system for the authenticators (Each authenticator on authentication send block to all other authenticated peers)
* Master-Slave, for nodes to be authenticated (Nodes to be authenticated pull data of the Database as and when the Value of the Hash changes on the database)

Disadvantages:

* When 2 mobile phones are authenticated at the exact same instant, then the mobile authenticated depends on the Database Update as well as the Notification received or not. This makes this decision just at this instant alone non-deterministic.



Second Architecture:

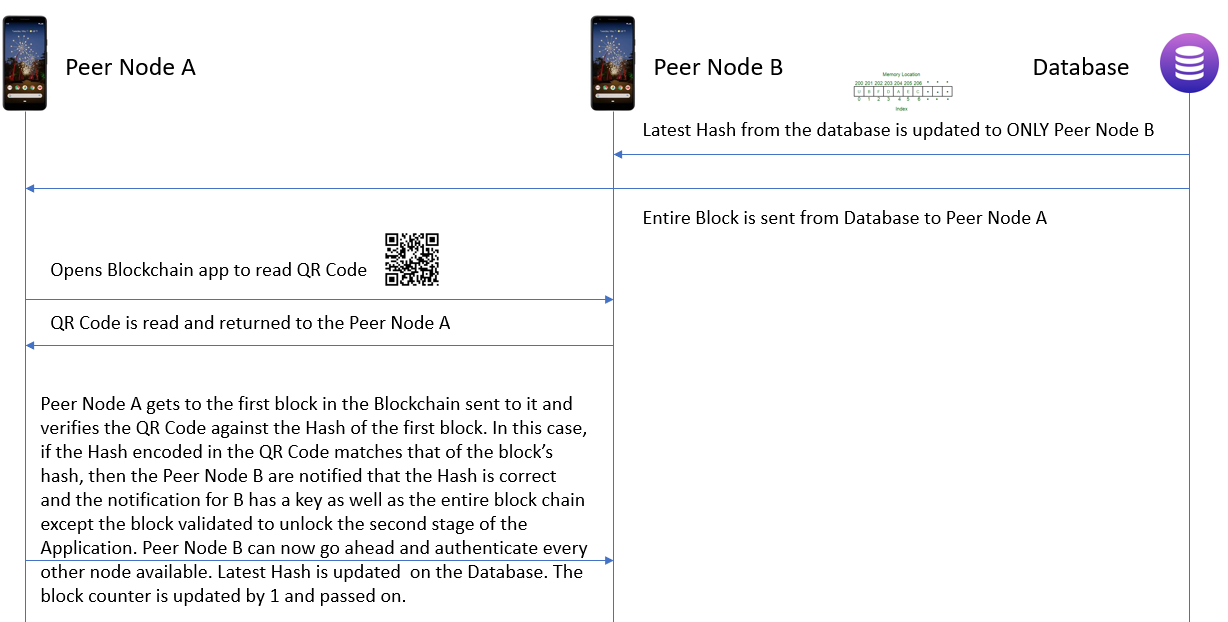
1. Latest hash along with the Entire Block chain is updated from Database to Peer Node A and B.
2. As soon as Peer Node B sees changed value in Database, it updates the QR code to that of the latest hash.
3. Peer Node A reads the QR code of Node B.
4. Peer Node A compares the QR Code against the all the Blocks that it received from the database verifies if the hash encoded in the QR Code is correct or not. If it is correct, Peer B along with all other peers get the entire block chain and Peer B can unlock it’s QR Code Scanner portion.
5. Peer Node A also updates the Hash in the Firebase Database so that every other peer yet to be authenticated can update their QR code.

Advantages:

* No collision issues.

Disadvantages:

* Man-in-the-Middle attacks can cripple the entire system
* Entire chain needs to be scanned for every iteration of the authentication.



Third Architecture:

1. The Entire Block chain is updated from Database to Peer Node A.
2. Latest Hash is updated on Peer Node B ALONE.
3. Peer Node A reads the QR code of Node B.
4. Peer Node A compares the QR Code against the block counter’s Block (block counter initially 1) that it received from the database verifies if the hash encoded in the QR Code is correct or not. If it is correct, Peer B gets the entire block chain except the current block read and Peer B can unlock it’s QR Code Scanner portion.
5. Peer Node A also updates the Hash in the Firebase Database so that next peer yet to be authenticated can update their QR code.

Advantages:

* Only Block counter’s block needs to be verified against the Hash.

Disadvantages:

* MiTM attacks can cripple entire system.
* Peer-Peer Authentcation for authenticator nodes. Pull Request for nodes undergoing authentication. (No true Peer-Peer Authentication)

**ARCHITECTURE SELECTED**

Looking at the pros and cons of each architecture, we chose to proceed ahead with the First Architecture. In this architecture, the Peer Node A is a One Plus 6T with the Peer Node B being Nexus Tablet. There are 2 versions of the application, one for Node A and other for all the other nodes to be authenticated. To simulate the warmup of the MD5 algorithm (as is the best practice), the Peer Node A carries out adding 10 blocks of random data to its blockchain before uploading the latest Block’s hash to the Database. Post this, the message exchange takes place as given in the Figure described above.

If you look at the current architectures suggested above, there is no mention of a front end server which will asynchronously collect and forward updates to the database. This is very much necessary as frequent read and writes to the Database directly from an application is not suggested. Hence we plan to include this in the future extensions with the benchmarking being done to finalise on the server to be selected.

**FRONT END SERVER SELECTION**

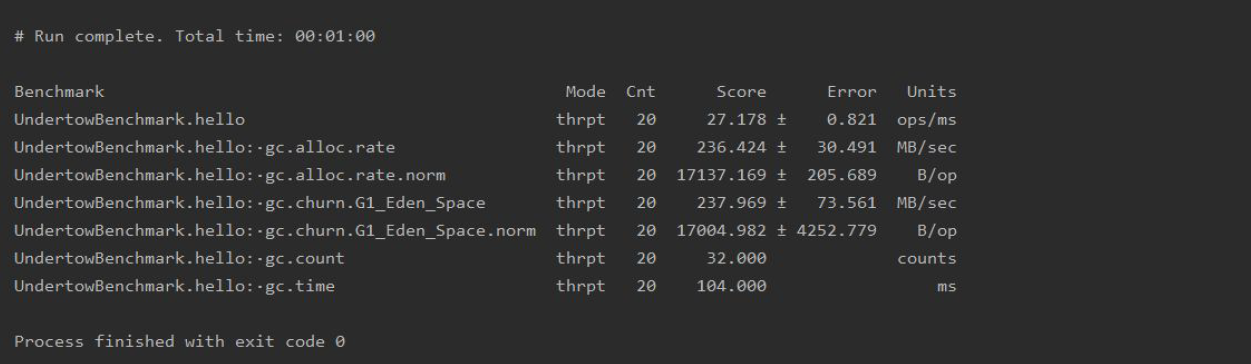
An important part of the project is to understand is the collision aspect of the requests sent. As described above, collisions are when 2 Nodes send the command to authenticate their respective Nodes at the same instant so that these nodes can go further in authentication. So in the case when 2 nodes unlock with same hash, the time stamps at which they create the new hash is different leading to undefined behaviour on who is unlocked first. Also, which latest hash will be populated in the database is again undefined.

Looking at these possible errors, it is imperative for us to keep a non-blocking server in between the database and the Nodes which will collect the Requests and update database without having an undefined behaviour.

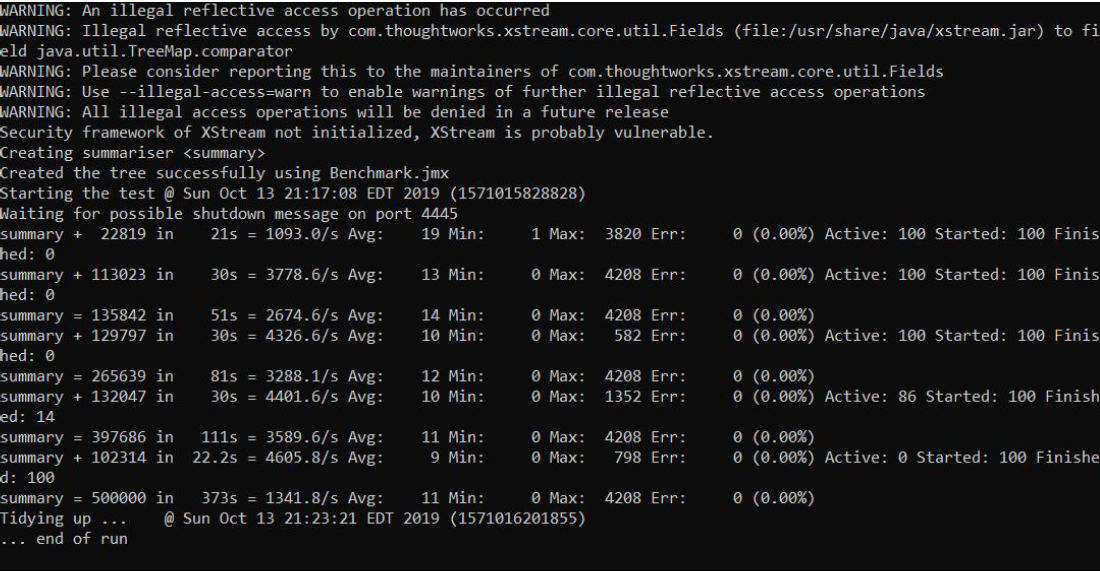
I did the benchmarking for the 3 frameworks: Vertx, Undertow or Spring boot. The reason to choose these 3 is because these support reactive programming. Reactive programming is a paradigm associated with asynchronous streams, which responds to any changes or events. Vertx uses an event bus to apply reactive programming to communicate with different parts of the application and passes the event asynchronously to handlers when available. In Spring, we need to handle synchronisation and locking manually. Undertow is a NIO server that can handle both blocking and non-blocking requests.

Benchmarking results are as follows:

Undertow Benchmarking (32 threads sending requests totalling 35000 requests against server using OkHttp.)



Spring Benchmarking (GET Requests totalling about 32000 requests against Server)



This clearly shows that we must proceed with the Undertow server as it is able to run 17137 (with an error of 205 operations) in 3 minutes. Hence for 35000 requests it is going to take 6 minutes to completely resolve all the queries.

Benchmarking Notes:

Type of Requests sent – Simple GET Requests satisfying the Hash Difficulty to update a sample Firebase Server.

Number of VMs for the benchmarking – 6 m4.large VMs in AWS

Number of requests sent – 32000 to 35000 RPS (Requests per second) depending on how much the server can scale.

Run length – Run until all the requests are complete (19 mins for Spring and 6 minutes for Undertow)

Benchmarking Tool – JRunner

One thing to remember is that it can be a maximum of 3000 queries (3000 people are **simultaneously** authenticated) which will come in a real-world scenario concurrently. This hence can be resolved by either Undertow or Spring but the Reactive programming capability of Undertow might help in any future expansions of the Project.

**DETAILS OF SOLUTION**

Important Definitions:

Difficulty: The difficulty is a number that regulates how long it takes for miners to add new blocks of transactions to the blockchain. In our case, we define difficulty as the minimum number of zeros in the hash that it needs to start with till the Miner is said to stop the mining and the hash is considered acceptable for the block.

Nonce: In cryptography, a nonce is an arbitrary number that can be used just once in a cryptographic communication. It is similar in spirit to a nonce word, hence the name. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks. This is basically the string that is added to the data to be hashed to give the acceptable hash. This value keeps getting updated by the miner till the hash is acceptable.

**IMPLEMENTATION OF THE UNDERLYING AUTHENTICATION TECHNIQUE**

UNDERLYING IMPLEMENTATION

BLOCK CLASS

BLOCKCHAIN CLASS

The underlying authentication is a simple blockchain with a difficulty of 4 which is implemented by 2 classes: Block and Blockchain. The Block class is responsible for the creation of the individual blocks while the Blockchain class contains the individual blocks as well as the functions responsible to mine new blocks.

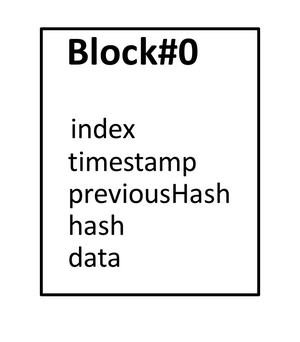
The concepts used here is based on the the following link:

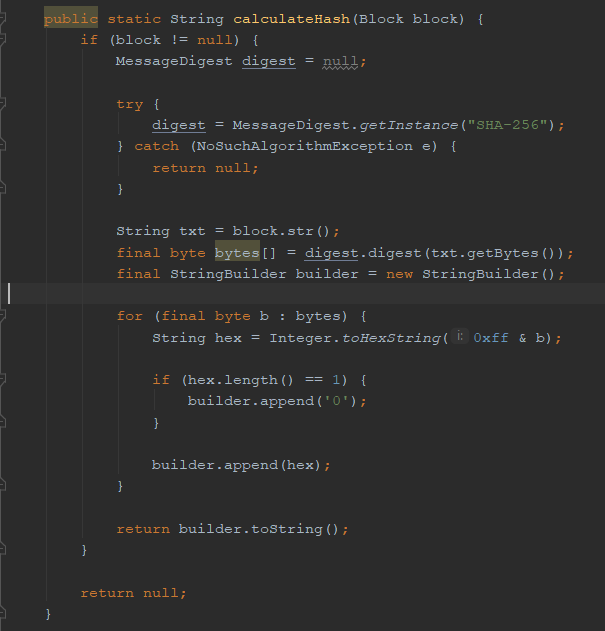
<https://www.ssaurel.com/blog/create-your-own-blockchain-in-30-minutes/>

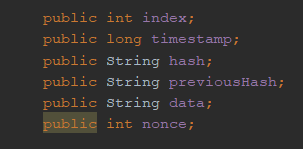
The link here explains a basic implementation of Blockchain in Java. My code has been written based on the concepts given here with a lot of changes in order to implement on the Mobile Platform (Android Studio).

**BLOCK CLASS**

**STRUCTURE OF THE BLOCK CRYPTOGRAPHIC HASH FUNCTION**

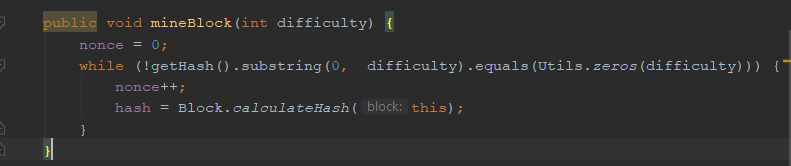
 SHA-256 Hash function is used.

 **HASH CALCULATION**



**BLOCK MINING**

**In this project the difficulty that we are using is 4. This means that the block must be mined until the hash that has been generated has starts with 4 zeros. The higher the difficulty value, the longer the process of mining and the harder it is to break the chain. For a mobile with 4 GB RAM, it takes 8 seconds to build a chain of 10 blocks with a difficulty of 4. Beyond 4, there is a trade-off with usability with the mobile phone either taking much longer or crashing down.**



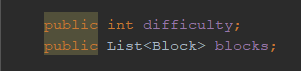
**BLOCKCHAIN CLASS**

**BLOCKCHAIN**

Block 3

Block 1

Block 2



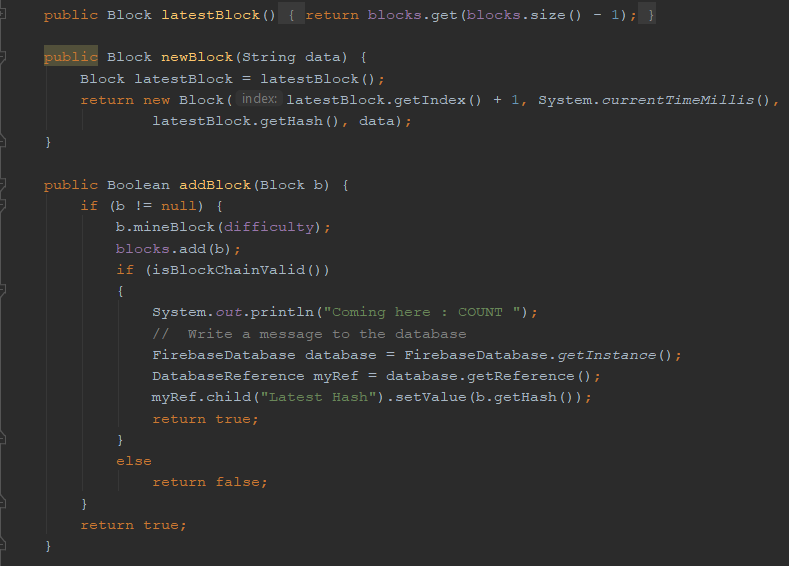
Block 5

Block 4

Block 6

**CREATING AND MINING NEW BLOCKS**

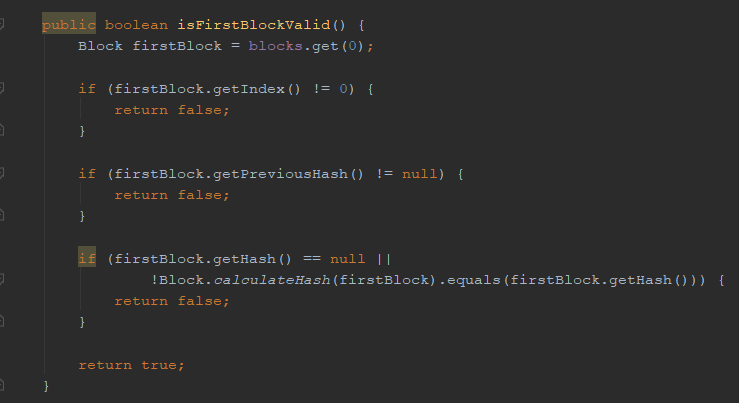
The Blockchain must allow the creation of new blocks and their mining taking into account the difficulty associated with it. So, we add a newBlock method to generate a new block associated with the last block known of the Blockchain.



**BLOCKCHAIN VALIDITY CHECK**

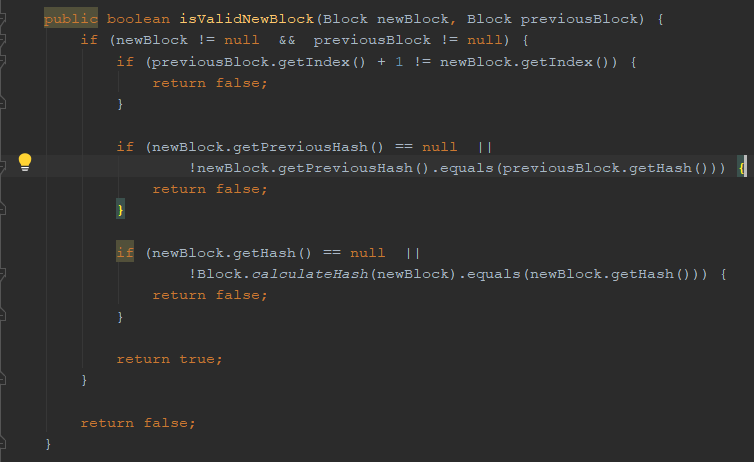
In order to verify the blockchain we need to validate the following:

* The first block is valid



* A new block is valid with the previous Block

A new block is valid next to the previous block of the Blockchain if its index is incremented by 1 compared to the index of the previous block, its previousHash field is filled with the hash of the previous block and finally if its hash is itself coherent



* Blockchain is valid

A Blockchain is valid and the integrity of its data is guaranteed when the first block is valid and each block that composes it is valid next to the block that precedes it.

