**Archiving Financial Data on Ethereum Blockchain**

Bachelor’s Thesis

CMPE492 SPRING 2018 - FINAL REPORT

4 JUNE 2018

FURKAN ERDEM PERŞEMBE, CEYHUN UZUNOĞLU

**Ph.D. Professor Can Özturan**

Contents

[1. Introduction and Motivation 3](#_Toc515639831)

[2. State of the Art 4](#_Toc515639832)

[2.1 What is Blockchain ? 4](#_Toc515639833)

[2.2 Ethereum Blockchain and its Features 5](#_Toc515639834)

[2.3 Ethereum Virtual Machine 5](#_Toc515639835)

[2.4 Smart contracts 6](#_Toc515639836)

[2.5 Applications 6](#_Toc515639837)

[3. Methods 8](#_Toc515639838)

[3.1 Currency data retrieving from trusted sources 8](#_Toc515639839)

[3.2 Ethereum private network connection 8](#_Toc515639840)

[3.3 Smart Contract functionalities 9](#_Toc515639841)

[a-) Maps as a data resource 9](#_Toc515639842)

[b-) Data insertion functions 9](#_Toc515639843)

[c-) Data retrieval functions 10](#_Toc515639844)

[3.4 Smart Contract value types in Solidity 10](#_Toc515639845)

[a-) Currency precision 10](#_Toc515639846)

[b-) Currency conversion 11](#_Toc515639847)

[4. Results - Works Done So Far 12](#_Toc515639848)

[4.1 Curency Data Retrieving from Trusted Sources 12](#_Toc515639849)

[4.2 Ethereum private network connection 12](#_Toc515639850)

[1. Maps as a data source 13](#_Toc515639851)

[2. Data insertion functions 14](#_Toc515639852)

[3. Data retrieval and converter functions 14](#_Toc515639853)

[5. Conclusion and Discussion 15](#_Toc515639854)

[6. Future Work 16](#_Toc515639855)

[7. References 17](#_Toc515639856)

# 1. Introduction and Motivation

Decentralized systems have always been a question and also a chance for humanity. As a decentralized system, blockchain had created an enormous interest in people at the present time. There are many reasons behind this interest, i.e. high price increase in crypto currencies. however from the point of computer science, blockchain is highly valuable software architecture.

In our capstone project, we are developing a smart contract that holds currency data from TCMB(The Central Bank of Turkey) and ECB(European Central Bank) in Ethereum blockchain network. Our smart contract keeps all past currency data from these two banks, so people may use these currency data by consuming our smart contract.

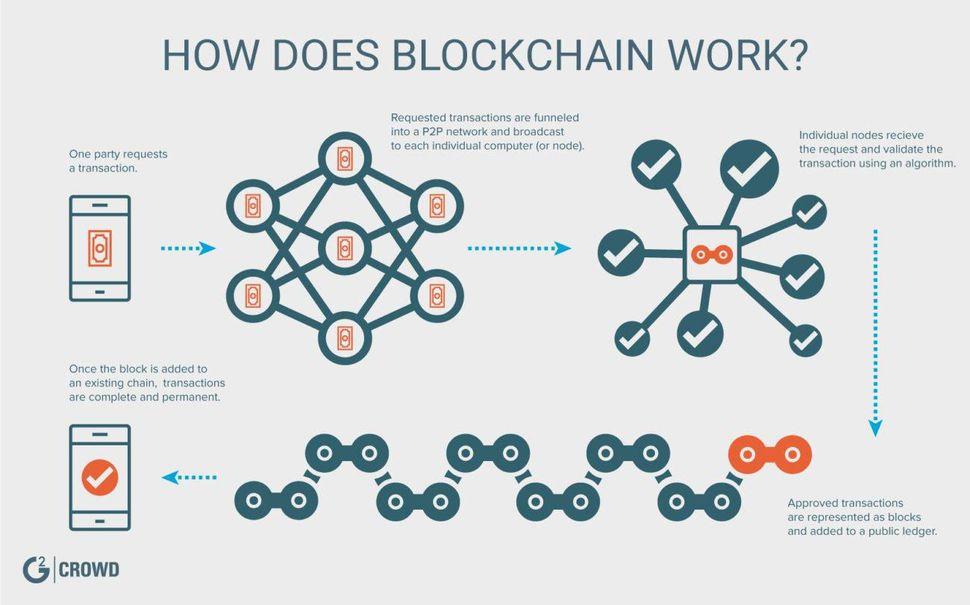
It will be mentioned later, but with the aim of clear understanding, blockchain cannot retrieve data from outside of the blockchain network. Smart contracts can only get data from other smart contracts or its creator.

Blockchain has its own currencies and people should convert their own country money to blockchain currency vise versa. Almost all transaction and operation in blockchain network related about money, hence they are all related with currency values.

Because of these reasons, in this project we want to offer trustable currency data for all people in Ethereum blockchain network with minimum fees. People can retrieve any data from 1999 to until today, can convert currencies in any date between 1999 and today. The reason of the date 1999, TCMB and ECB gives currency data till that date. Our initial focus is Turkish users, however all people in around the world can use our smart contract trustfully.

# 2. State of the Art

## 2.1 What is Blockchain ?



A blockchain is a continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block typically contains a cryptographic hash of the previous block, a timestamp and transaction data. By design, a blockchain is inherently resistant to modification of the data. It is "an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way". For use as a distributed ledger, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for validating new blocks. Once recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks, which requires collusion of the network majority.

Blockchains are secure by design and exemplify a distributed computing system with high Byzantine fault tolerance. Decentralized consensus has therefore been achieved with a blockchain. This makes blockchains potentially suitable for the recording of events, medical records, and other records management activities, such as identity management, transaction processing, documenting provenance, food traceability or voting.

Blockchain was invented by Satoshi Nakamoto in 2008 for use in the cryptocurrency bitcoin, as its public transaction ledger. The invention of the blockchain for bitcoin made it the first digital currency to solve the double spending problem without the need of a trusted authority or central server. The bitcoin design has been the inspiration for other applications.

## 2.2 Ethereum Blockchain and its Features

Ethereum is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. It supports a modified version of Nakamoto consensus transaction based state transitions. In popular discourse, the term Ethereum is often used interchangeably with Ether to refer to the cryptocurrency that is generated on the Ethereum platform.

Ether is a cryptocurrency whose blockchain is generated by the Ethereum platform. Ether can be transferred between accounts and used to compensate participant mining nodes for computations performed. Ethereum provides a decentralized Turing-complete virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts using an international network of public nodes. "Gas", an internal transaction pricing mechanism, is used to mitigate spamand allocate resources on the network.

Ethereum was proposed in late 2013 by Vitalik Buterin, a cryptocurrency researcher and programmer. Development was funded by an online crowdsale that took place between July and August 2014. The system went live on 30 July 2015, with 11.9 million coins "premined" for the crowdsale. This accounts for approximately 13 percent of the total circulating supply.

In 2016, as a result of the collapse of The DAO project, Ethereum was split into two separate blockchains – the new separate version became Ethereum (ETH), and the original continued as Ethereum Classic (ETC). The value of the Ethereum currency grew over 13,000 percent in 2017.

## 2.3 Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is the runtime environment for smart contracts in Ethereum. It is a 256-bit register stack, designed to run the same code exactly as intended. It is the fundamental consensus mechanism for Ethereum. The formal definition of the EVM is specified in the Ethereum Yellow Paper. It is sandboxed and also completely isolated from the network, filesystem or other processes of the host computer system. Every Ethereum node in the network runs an EVM implementation and executes the same instructions. In February 1, 2018, there were 27,500 nodes in the main Ethereum network. Ethereum Virtual Machines have been implemented in C++, Go, Haskell, Java, JavaScript, Python, Ruby, Rust, and WebAssembly (currently under development). The Ethereum-flavoured WebAssembly (dubbed "e-WASM") is expected to become a major component of the "Web 3.0", a World Wide Web where users interact with smart contracts through a browser.

## 2.4 Smart contracts

Ethereum's smart contracts are based on different computer language, which developers use to program their own functionalities. Smart contracts are high-level programming abstractions that are compiled down to EVM bytecode and deployed to the Ethereum blockchain for execution. They can be written in Solidity (a language library with similarities to C and JavaScript), Serpent (similar to Python, but deprecated), LLL (a low-level Lisp-like language), and Mutan (Go-based, but deprecated). There is also a research-oriented language under development called Viper (a strongly-typed Python-derived decidable language).

There is ongoing research on how to use formal verification to express and prove non-trivial properties. A Microsoft Research report noted that writing solid smart contracts can be extremely difficult in practice, using The DAO hack to illustrate this problem. The report discussed tools that Microsoft had developed for verifying contracts, and noted that a large-scale analysis of published contracts is likely to uncover widespread vulnerabilities. The report also stated that it is possible to verify the equivalence of a Solidity program and the EVM code.

## 2.5 Applications

Ethereum blockchain applications are usually referred to as DApps (decentralized application), since they are based on the decentralized Ethereum Virtual Machine, and its smart contracts. Many uses have been proposed for Ethereum platform, including ones that are impossible or unfeasible.Use case proposals have included finance, the internet-of-things, farm-to-table produce, electricity sourcing and pricing, and sports betting. Ethereum is (as of 2017) the leading blockchain platform for initial coin offering projects, with over 50% market share.

As of January 2018, there are more than 250 live DApps, with hundreds more under development. Project applications listed in this section are not exhaustive and may be outdated.

* Digital signatures that ensure authenticity and proof of existence of documents: the Luxembourg Stock Exchange has developed such a system
* Slock.It is developing smart locks
* Digital tokens pegged to fiat currencies: Dai, stablecoin pegged to US dollar. Decentralized Capital. Spanish bank Santander is also involved in such a project.
* Digital tokens pegged to gold: Digix
* Improved digital rights management for music: Imogen Heap used the technology
* Platforms for prediction markets: Augur, Gnosis Stox
* Platforms for crowdfunding: the DAO
* Social media platforms with economic incentives: Backfeed, Akasha
* Decentralized marketplaces: FreeMyVunk, TransActive Grid
* Remittance: Everex
* Online gambling: CoinPoker, Etheroll
* Electric car charging management: RWE
* Secure identity systems for the Internet: uPort
* Labour economics: Blocklancer, Ethlance
* Video Games: Cryptokitties popularity in December 2017 caused the Ethereum network to slow down.

# 3. Methods

## 3.1 Currency data retrieving from trusted sources

As it is mentioned in introduction, in this project we are creating a smart contract that keeps currency data from trusted sources and serve these data to Ethereum network. Blockchain is a decentralized network, so there is no authority to serve all country currencies to users. If someone, like us, will serve these data to users, their source of data should be trustable and non-debatable. For the purpose of this, we retrieve data from TCMB(The Central Bank of Turkey) and ECB(European Central Bank).

TCMB has its XML data source for daily currency values according to TRY(Turkish Lira) and crossrate values for USD( American Dollar). TCMB publishes 18 currencies ratio according to TRY. These values are daily and TCMB publishes daily data at 15:30 (GMT +03, Istanbul, Turkey).

ECB is European Central Bank. ECB has its XML data source for daily currency values according to ratio of EUR(Euro). ECB publishes 32 currency value in daily manner. ECB publishes these values at 14:00 (CET, Central European Time) and updates the values at 16:00 (CET, Central European Time).

Our methodology in this part of the project is that retrieving currency values from both of TCMB and ECB separately into currency classes. Each class gets all the information in these two banks. For instance, TCMB has four kind of currency values: ForexBuying, ForexSelling, BanknoteBuying, BunknoteSelling. Each classes keeps timestamps and data in the TCMB and ECb sources.

Retrieving data can be cumbersome at some points due to the sources can change their way of publishing data. Because of this, data publishing of these sources should be observed and some general decisions should be taken.

## 3.2 Ethereum private network connection

In this project, we used eBloc private Ethereum network that was created by assistants and students of Prof. Can Özturan in Boğaziçi University. The reason that we are using private Ethereum network is that smart contract creation and data insertion in to that contract needs fees. Tests of smart contract can be so much expensive in public Ethereum network.

Ethereum smart contracts are generally compiled in Remix online compiler. Remix can installed to personal computer and it can compile smart contracts offline. Also there are other smart compiling ways in other languages that are provided in Ethereum github resources of language APIs.

Thanks to Can Özturan and his assistants, we are provided eBloc private Ethereum network and enough ether money are provided to us.

## 3.3 Smart Contract functionalities

Smart Contracts have not any database or database integrations. Smart Contracts live in EVM(Ethereum Virtual Machine). All data in smart contract are preserved in Solidity data types like arrays, maps, strings, etc. .

Because of there are no kind of database in smart contracts, currency data should be kept in Solidity data type of maps. Date of data is substantially important. Keys of maps should be dates of the retrieval date of the currency data.

### a-) Maps as a data resource

TCMB has four kind of currency values as it is mentioned above. For ForexBuying, ForexSelling, BanknoteBuying, BanknoteSelling and CrossRate values there should be maps for each of them. Keys of maps should be dates of retrieval times of currency data.

ECB has one kind of data, so one map can be enough for ECB.

### b-) Data insertion functions

In Ethereum Blockchain network, only the owner of the smart contract can insert data into the smart contract. Creator of smart contract should create Solidity functions for itself. Data insertion functions should guarantee that data is inserted only by the contract creator. It is so important for *trustworthiness*.

Solidity provide “msg” object at the creation of smart contract and also all transactions in blockchain network. This “msg” object has a “owner” field. Insertion functions should check the account that is inserting data into the smart contract is owner or not. If the account that is inserting data into the smart is not the owner, function should not be executed.

Data should be inserted into maps by giving matched date of data retrieval and currency codes. Each map should have its insertion functions.

### c-) Data retrieval functions

Data should be served to users with the principle of ease of use. Each bank, TCMB and ECB should have its own data get functions. People may want to get only a value in a specific date or ay want to convert the currencies between each other in a specific date. There should be functions for all of these operations.

Mainly TCMB should have its currency get function for all four types and crossrateusd type, and also currency conversion function. Same as with TCMB, ECB should have its currency get function and also currency conversion function.

For all functions, date values are so important. Date values should be unix time or universal time conversion type.

## 3.4 Smart Contract value types in Solidity

Solidity is a restrictive programming language because of its necessaries. There is no for loops, and generic big data types. It was because of someone who is unconscious or unreliable can cause a bottleneck in blockchain network with that kind of language properties.

Solidity has so small value types and restrictive features as it is mentioned above. Blockchain network transactions fees are calculated according to how much data is used. Hence, it is crucial to use as much as less data in smart contract. Because data insertion/retrieval and keeping data in smart contract needs money. Data types in our project till midterm report are:

* Dates are keys of maps. Unix time requires uint64. Example of unix time is “1521375531”.

Universal time date requires uint32. Example of universal time only for year/month/date is “20180321”. Zeros are necessary for date understanding in function parameters.

* Currency codes requires bytes3. Currency code has a universal standard and it should be three alphabetical words long. Example of currency codes are “TRY”, “EUR”, “USD”.

### a-) Currency precision

One of the most significant and problematic situation in Solidity is that there aren’t any floating numbers in Solidity. Because of this reason, currency values should be multiplied by some big integer value before it is inserted into the smart contract. Precisions are important in currency values. Multiplying with a big integer value preserve the precision value. When it was retrieved from smart contract it should be divided to that specific big integer number for precision correction.

As solution of this problem, Currency values should be multiplied with 10^9 before inserting into the smart contract and should divided to 10^9 after retrieved from smart contract.

### b-) Currency conversion

Each bank should have its currency conversion functions for consistency.

Currency conversion algorithms requires the division operation. Division operation can cause precision values if the result is floating point number. In solidity, division operations are all integer divisions, that means no precisions are provided.

There is a generic solution to this general problem. In order to get precision values from division operations in solidity, the dividend number should be multiplied with a 10^X number. X is the number of precision that you want.

In our smart contract we always give currency value as 10’9 multiplied. In order to preserve this decision, we multiplied the dividend number in divisions operations with 10^9. Hence, users that use our smart contract get function should divide all currency conversion values by 10^9.

## 3.5 ROPSTEN Test Network and Metamask

Currency codes requires bytes3. Currency code has a universal standard and it should be three alphabetical words long. Example of currency codes are “TRY”, “EUR”, “USD”.

### a-) Currency precision

One of the most significant and problematic situation in Solidity is that there aren’t any floating numbers in Solidity. Because of this reason, currency values should be multiplied by some big integer value before it is inserted into the smart contract. Precisions are important in currency values. Multiplying with a big integer value preserve the precision value. When it was retrieved from smart contract it should be divided to that specific big integer number for precision correction.

As solution of this problem, Currency values should be multiplied with 10^9 before inserting into the smart contract and should divided to 10^9 after retrieved from smart contract.

### b-) Currency conversion

Each bank should have its currency conversion functions for consistency.

Currency conversion algorithms requires the division operation. Division operation can cause precision values if the result is floating point number. In solidity, division operations are all integer divisions, that means no precisions are provided.

There is a generic solution to this general problem. In order to get precision values from division operations in solidity, the dividend number should be multiplied with a 10^X number. X is the number of precision that you want.

In our smart contract we always give currency value as 10’9 multiplied. In order to preserve this decision, we multiplied the dividend number in divisions operations with 10^9. Hence, users that use our smart contract get function should divide all currency conversion values by 10^9.

# 4. Results - Works Done So Far

## 4.1 Curency Data Retrieving from Trusted Sources

We created two parser classes in python language for both TCMB and ECB. These classes takes data from TCMB and ECB daily data resources in different format but in a XML type. Python libraries ElementTree, urllib and json are enough for XML data retrieval from resources of these two banks. observations BankoteBuying and BunkoteSelling can be empty in TCMB source.

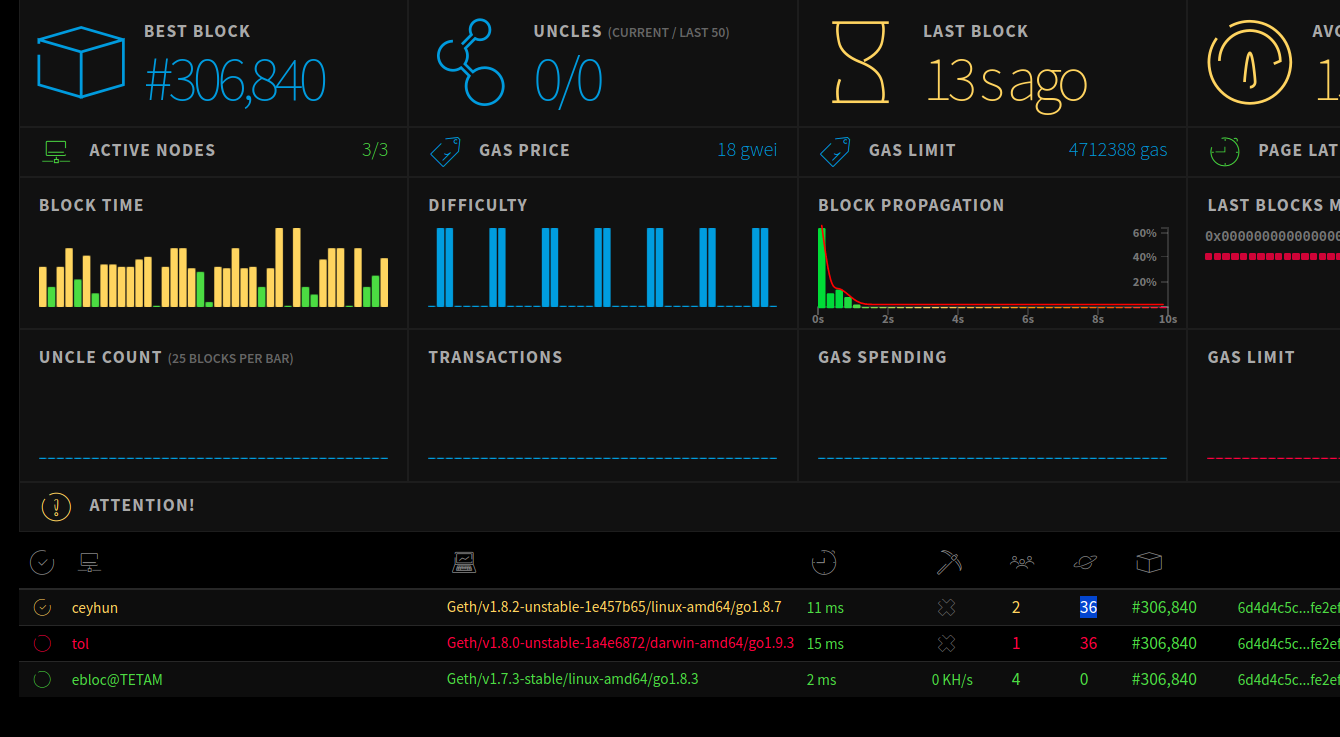
TCMB provides a XML source for daily currency data. For Turkish Lira(TRY) ratios there are four kinds of values: ForexBuying, ForexSelling, BanknoteBuying, BanknoteSelling. Also TCMB provides USD values for other than TRY currency in CrossRate column. Our TCMB parser class retrieves all these values and provides in a key-value manner.

ECB provides a simple XML source for daily currency values. Our ECB parser class gets all these values and provides in a key-value manner.

## 4.2 Ethereum private network connection

We developed our smart contract in eBloc Ethereum private network. Compilation of smart contracts are performed in Remix online ethereum compiler.

There are JavaScript bash client for Ethereum network that you can perform all necessary operations in the network. We used “web3.py” python library as a Ethereum blockchain network API. All data insertion, some smart code tests are executed via using “web3.py” library of python. Using “web3.py” we created a python language API for our smart contract.



**4.3 Smart Contract functionalities**

### Maps as a data source

We used maps as a data source in our project. Because there are no way of keeping data other than maps and arrays. Our maps are nested. Our map architecture is:

map(key: uint16\_YEAR => map(key: uint8\_MONTH => map(key: uint8\_DAY =>map(key: bytes3\_CURRENCY\_CODE => uint256\_VALUE))))

This way of mapping implementation is primitive, however we will change this way of implementing in to unix time and universal time. For ease of tests, we used this kind of solution approach to the data keeping problem.

At the beginning of our project, we designed our structure in a more simple way. We created two struct types that holds all TCMB currencies and ECB currencies. However, inserting 18 currency for TCMB and 32 currency for ECB in one shot is impossible or infeasible in Solidity. Because, a function in Solidity can take 8 parameters at most.

Using date and currency codes as keys of maps is more simple than other solutions in our point of view. Disadvantage of our architecture is there are additional transactions in data insertion, however it is not so important. Important aspect of our smart contract is ease of data retrieval.

### Data insertion functions

Only the owner of smart contract can insert data into smart contract. There is built-in “require” function in Solidity to check some condition in order to execute a smart contract function. In the creation of our smart contract, creator account is held in a “owner” variable. Each insert function in our contract checks the account of the callee of the function with “require” built-in function by comparing the owner with the calle account.

TCMB has five maps all for ForexBuying, ForexSelling, BanknoteBuying, BanknoteSelling and CrossRate. Each map has its own insertion function.

ECB has two maps for euro and dollar maps. Dollar map values are generated from euro values. Because of the reason dollar is used so much in transactions, we keeps its values in a seperated map.

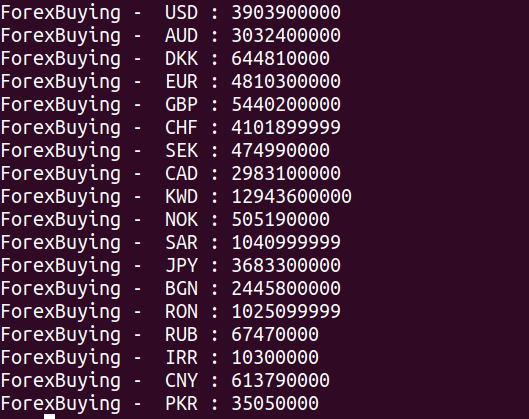
Data insertion are performed by us. Hence, we developed a python API for this purpose. This API consumes TCMB and ECB parser classes and calls smart contract functions by using “web3.py” library.

TCMB and ECB parser classes provide floating point values. Insertion API ultiplies these values with 10^9 and convert them into Solidity integer type. Python integer and Solidity integer are different. “Web3.py” library provides pleasant API for this kind of Solidity functions. As it is said before, all values in maps that holds values are the currency values that are multiplied with 10^9. Thus, we solved the precision problem.

### Data retrieval and converter functions

There are two kind of data retrieval in our smart contract. The first one is simple get function type. It accepts a date parameter and a currency code parameter, and it returns a currency value that should be divided to 10^9.

Second kind of retrieval function in our smart contract is currency converter. Currency converter functions takes a date parameter and two currency code parameter which are converted between them. Currency converter function multiplies the first currency value with 10^9 and divide it by converted currency value. Result is the 10^9 times of actual value. In our smart contract all retrieved values should be divide to 10^9, and this rule is also prevailing in currency conversion functions.



# 5. Conclusion and Discussion

Solidity is easy to use programming language. However, it has its necessary restrictions. They cause us some troubles. First of all there is no floating point numbers. There are some widely used solutions for that problem like creating a struct type with two integer field, one holds decimal part of the floating number and second one holds the precision values. In our solution to this problem is multiplying the floating point with 10^9. The reason for that solution is that there are so many mathematical operations in our smart contract like conversions. It is impractical to use other solutions for floating point problem like struct with two integer parts. Seconly, Solidity functions can take at most 8 parameters. It was a problem for our insertion functions, but we solved this problem also.

Blockchain has a great philosophical background. Decentralized systems can be a revolution for the world if it is used for the sake of humanity. All centralized systems have their backups and disaster scenarios. However, in blockchain as a decentralized system, data lives forever in network if some conditions hold.

# 6. Future Work

#### 6.1 Second smart contract that will consume the current smart contract

In the future outlook of the project, there is an implementation of another smart contract that was used for some specific purpose. This future smart contract should consume our current currency provider smart contract. It will be a client for current smart contract, it will take currency values and convert currency values.

As a result of the guidance of our instructor Can Özturan, this client smart contract will be related with insurance sector. One of the many use areas of blockchain is insurance applications. Some small or medium size insurance application smart contract will be implemented. Due to the fact that there are tremendous needs for currency data in insurance prices, this future smart contract will consume our current smart contract.

#### 6.2 A function that returns available dates of currency data

Our smart contract should be trustable and convenient. If data is not present at the given date parameter, it will cause an unnecessary transaction fees for the user of our smart contract. In order to avoid this kind of fallacies, it is essential to create a function that serves the date range that currency data are available.

#### 6.3 Some code and structure changes

Also, some code implementations will be changed. As it is mentioned before, date values as keys of maps are inconvenient. Date values will be change to unix time and universal time standards.

Unix time is important for smart contracts. Because, some smart contracts automatically consumes other smart contracts without any user intervention. Solidity provides date values in Unix time format. In fact, it was crucial to use Unix time in our smart contract.

For convenience and ease of use, we will also add universal time standard date parameters for converter and getter functions.

#### 6.4 Beyond our project horizon

It was beyond our project horizon, but it can be implemented that all financial data can be retrieved from central banks and some significant financial institutions. There are so many financial data about countries and companies in financial industry like stock exchange data. They are required in some operations.

Also these financial data can served with the needs of industry. Like currency converter, there are some financial calculations by consuming above financial data. In this smart contract, these kind of functions can be implemented.

# 7. References

* <https://solidity.readthedocs.io/en/v0.4.21/>
* <http://remix-alpha.ethereum.org>
* <https://remix.ethereum.org/>
* <https://github.com/ethereum/web3.py>
* <https://web3py.readthedocs.io/en/stable/>
* <https://github.com/ethereum>
* <https://en.wikipedia.org/wiki/Ethereum>
* <https://theethereum.wiki/w/index.php/Main_Page>