APPLYING BLOCKCHAIN TO SMART HOME SYSTEM

By

Chai Pei Zhen

A PROPOSAL

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2019

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: ____

____

____

Academic Session: _____________

I ____

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in

Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_________________________	_________________________
(Author’s signature)	(Supervisor’s signature)

Address:

____

__________________________ _________________________

__________________________ Supervisor’s name

Date: ____________________ **Date**: ____________________

DECLARATION OF ORIGINALITY

I declare that this report entitled "**APPLYING BLOCKCHAIN TO SMART HOME SYSTEM**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	
Name	:	Chai Pei Zhen
Date	:	8/4/2019

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Dr Lee Wai Kong for his support and advice throughout the project. I really appreciate his guidelines and effort when I am facing some difficulties in this project. Dr. Lee always gives me some effective suggestions and ideas so that I can do my project smoothly.

Next, I would like to thank my families and friends for their encouragement given to me. They make me feel more confident when I am facing some trouble in academic.

ABSTRACT

With the advancement of Internet of Things technology nowadays, smart home system is getting more and more popular. Before people starts to install and implement smart home system into their house, they will worry about the security issue embedded within this system. This implies that cyber security of smart home system plays a huge role and needs to be paid attention to. In this project, blockchain technology will be apply on smart home system to enhance the security of existing smart home system because blockchain technology can help to secure integrity, authenticity and auditability of personal data and information. By the nature of blockchain, each block in the chain will be linked with previous block through the help of hash function and changing of transaction records can be detected easily. Additionally, if a block was altered, the whole chain needs to be recomputed through consensus process which was too expensive and almost impossible. Other than that, blockchain also made use of public key cryptography method such that the public key of the data sender will be verified by peer nodes so only authorized parties can access. In this proposed method, it consists of a community of smart homes and only trusted parties can join into this community. Sensor data can be send to server through the aid of local gateway device within each smart homes so that these data can be query out in on-demand basis. Besides, local gateway devices will also help each another to verify a transaction before a request can be successfully enrol. Based on the data stored in the database, users can subscribe to additional service such as healthcare provided by cloud service provider through smart contract established on the consortium blockchain network.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IoT	Internet of Things
BC	Blockchain
SHS	Smart Home System
DoS	Denial of Service
DDoS	Distributed Denial of Service
BC-SHS	Blockchain based Smart Home System
BLE	Bluetooth Low Energy
EVM	Ethereum Virtual Machine
EOA	Externally Owned Account
API	Application Programming Interface
URL	Uniform Resource Locator
IPFS	InterPlanetary File System
PHP	Hypertext Preprocessor
JSON	JavaScript Object Notation
FCM	Firebase Cloud Messaging
GUI	Graphical User Interface
IDE	Integrated Development Environment
SMS	Short Message Service
cURL	Client URL
HTTP	HyperText Transfer Protocol
FTP	File Transfer Protocol
DNS	Domain Name System
SSH	Secure Shell
SDK	Software Development Kit
iOS	Iphone Operating System
PoS	Proof-of-Stake
PoW	Proof-of-Work

Chapter 1 Introduction

1.1 Background Information

The Internet of Things (IoT) is a term that originate by (Ashton 2009). This term does not have a unique definition but the general idea among people is the same. The idea was tons of physical devices in this world connect to the internet to collect and share data. It also referred to objects that are identifiable and interconnected through digital networks. This implied that an action can be trigger on these devices through internet. IoT can be applying in many areas such as security, transportation, e-Health, manufacturing, utilities, industrial provisioning, facility management and precision agriculture. However, unique IoT applications may have different requirements in terms of network and processing (Kusek 2018). For example, surveillance camera needs to have a good bandwidth to stream video in a stable and consistent way. In a typical IoT system, all collected data are stored in centralized cloud servers because each piece of device in the system requires outstanding collaboration, connectivity and coordination. This means that all IoT devices of a party should make their data accessible by the other IoT object. From (Jukić, Špeh & Heđi 2018), cloud servers for IoT purpose need to fulfil some criteria like large volume of storage, support variety of IoT data and high processing speed.

Recently, there exist of a technology which is known as blockchain (BC) technology. BC technologies are popular today largely because of the success of Bitcoin (Nakamoto 2008). However, this does not restraint usage of BC within the area of cryptocurrency only. There are also many BC platform enterprises nowadays such as Ethereum, Hyperledger, Ripple and Quorum exploring and developing how BC can contribute to the society in the future. In (Christidis & Devetsikiotis 2016) it states that BC is a distributed ledger that enables two parties who do not fully trust each other to communicate or transact in an immutable and transparent manner. The reason it is call BC because each block (except the first block) is link with previous block by using cryptographic hash function or algorithm (a mathematical algorithm that converts data of arbitrary size to a bit string of fixed size), when more and more blocks are added it forms a long chain. Each block in the BC is identifiable by a hash and it can be stored as a flat file or in a database. Hash function is used on the header of each block in order to avoid collision of header. Besides, it facilitates in protecting data in a block because a hash will be different if the key used to generate a hash is different. According to (Dinh et al. 2018), two parties interact with the BC through pair of public and private keys. When a new block needs to be created, it will broadcast by a node to its nearby one-hop peers. The neighbouring peers will ensure this new block is valid before relay it further. If the new block is valid after verification, known as mining, the mining node will broadcast this block back to the network, add this block to the chain and update the ledger. If this new block is invalid, it will be discard. BC can be divides into two categories which are public and private BC. Public BC is open to everyone who wants to participate into the network. One of the most known examples of public BC is Bitcoin. However, in private BC, a node needs to have permission in order to join the network.

Due the advancement of IoT today, Smart Home System (SHS) concept is going to be more and more popular. A home can be considering as a smart home if it can achieve several aspects such as home automation and entertainment system, security, convenience, comfort and efficient energy management. A smart home can be categories into local and cloud control types. Local SHS only allows local connections of devices within the home but cloud SHS allows devices to communicate remotely. There are several pros and cons for these two SHS in terms of response time, reliability, security, control distance and functionalities. For local SHS, it has a faster response because information does not need to pass through cloud server but directly to the device. It also has a better reliability compare to cloud SHS since devices will work even when there is no internet connection. Furthermore, local SHS provides a better security environment as it will not expose personal information to the internet. However, the idea of SHS could be more powerful if digital devices could be control and manage from a distance. Therefore, SHS should be able to link with the cloud server through the internet in order to monitor and control all smart devices remotely through smart phones, laptop or tablet. For example, owner can turn on their air conditioner before reaching home or see the real time video from the surveillance camera. Cloud SHS brings great convenience but in the same time it also points out several problems.

1.2 Problem Statement and Motivation

Since cloud SHS is accessible to the internet and closely related to daily life, it raised lots of security issues which can be exploited for illegal activities. As mentioned in (Abdur et al. 2017), there are four major security threads embedded within an IoT based system which includes trespass, monitoring and personal information leakage, denial of service (DoS) or Distributed denial of service (DDoS) and falsification. Hence, the security matter needs to be solve so that cloud SHS can be deploy in a safe and widespread manner. These security issues mentioned is going to be tackled throughout this project.

1.3 Project Scope

In terms of security goals that can be achieve by using BC-SHS are integrity, authenticity and auditability. In BC network, few data transactions are packaged into a block and linked with previous adjacent block through cryptographic hash function. By utilizing this ability integrity of BC can be ensured such that any attempt to manipulate transaction records will be detected easily by neighbouring nodes. If one of the blocks is tampered in BC, the following blocks had to be recomputed through expensive mining process. However, this is almost impossible to achieve. Thus, integrity for data transactions in BC-SHS is guaranteed. Next, the use of asymmetric cryptography in BC-SHS can help with authenticity goals. Private and public keys are part of each trusted node, which is the gateway device, in BC-SHS and they are meant for signature generation and verification. The data sender signs a transaction with its own private key and the transaction is verified by its peer nodes through its public key. Data transaction within BC deploys digital signatures to authenticate identity of nodes so it is difficult for malicious party to generate group of imitate nodes to initiate DDoS attack. Subsequently, consortium members of the SHS can audit their transaction data at any time given that every transaction data record is logged in the BC. In order to review data stored in BC, one has to achieve agreement by majority nodes and go through mining process. However, it is tough due to overwhelming cost and time needed.

1.4 Project Objective

In this project, a blockchain based Smart Home System (BC-SHS) that offers an enhancement for SHS regarding its security issues and matters is proposed. In practical, there should be a community of smart home connected to the cloud server through the gateway device (node) but in this project it might not be. A computer can be used in order to stimulate the scenario to a certain extend. This BC-SHS is expected to provide a more invulnerable and safe environment when SHS and cloud servers are communicating and transmitting data to each another. Only authenticated members are allowed to access their data in the server. Moreover, the world expects SHS in the future will be link to the internet so the cyber security aspect must be highlight and tackle as soon as possible. There are three main objectives in this proposal wish to achieve:

1) To develop a consortium BC network that secure data transactions between multiple smart homes and cloud server.
2) To develop a cloud service that is able to receive, return and store data to and from the smart home community.
3) To develop a smart home gateway that is able to perform multiple functions such as mining, divert data and record data transactions.

1.5 Impact, significance and contribution

This project is one of the pioneering efforts in utilizing BC to practical use in Malaysia. Experience gained in developing this product can be a reference for other industries which may consider adopting BC solution. Based on research, there is no such product in the market yet. Hence, this project can be a role model for the players in this industry to follow.

Chapter 2 Literature Review

In (Santoso & Vun 2015) a deployment of gateway on Wi-Fi based IoT SHS was used to secure end to end communication of IoT devices. It is a project that exploits and utilizes benefits of Elliptic Curve Cryptography protocol (Amara & Siad 2011) to facilitate two rounds authentication processes. In the first round, the gateway device will send identity and pre-secret key to IoT devices and automatically connects both of them using Wi-Fi in order to validate mutually. In second round, IoT devices need to return identity and pre-shared key back to gateway. Before any data exchange between them, messages were encrypted through symmetric cryptography. In the end, shared key will be generated and used in subsequent communication between two parties. It was claimed that the project brings better security and convenience for SHS users. Other than what (Santoso & Vun 2015) mentioned, (Wang, Xu & Yang 2018) considered linking between gateways to cloud server. There is a similarity which is encryption needs to be done before sending of data but this time it uses random distinct keys to encode for each messages it relays. (Sridhar & Smys 2017) tells some problems from asymmetric key encryption. They say asymmetric key encryption have high overhead and could not provide all time security. It only provides protection for a session of time of communication. (Sridhar & Smys 2017) then proposed a solution named intelligent security framework which consist of dual mutual authentication. Instead of using one type cryptography method, it used two which were asymmetric key and lattice-based cryptography. There are several steps in their proposed algorithm.

Step 1: Encryption using Master key repository’s public key

Step 2: Creation of IoT nodes key pair

Step 3: Generation of secret device session key

Step 4: Generation of secret service session key

(After acquire secret device session key)

Step 5: Data transfer from gateway and IoT nodes

(After acquire secret service session key)

Step 6: Data transfer from gateway and IoT nodes

Step 7: End-To-End secure transaction

By using the double and mutual authentication schemes, it decreases traffic by removing fake and fault packets. From a higher perspective, encryption itself is insufficient to protect these data. Although nowadays decryption without knowing the key can be considered as a tough job but it can be further enhance into a more invulnerable environment where it is able to defend from various kinds of attack or sabotage. Rather than only using symmetric cryptography, BC can be applied for improvement because a hash value of the information it sends will be generate based on the previous block's hash and validation will be done by mining. Besides, any new transactions who wanted to add on to the chain need to get at least 51 per cent of agreement among all nodes and also two parties interact by using public and private keys. BC itself can also adapt to any cryptographic method if necessarily.

In the project developed by (Huh, Cho & Kim 2017), they used a BC platform called Ethereum to manage IoT devices. Etheruem itself provides flexibility such that different set of codes can be implemented to achieve certain functions. In the suggested model, every device will be connected to the Ethereum network and they will consist parts of the BC. The researchers also used three Raspberry Pis and a smart phone to simulate a BC IoT environment. Each Raspberry Pi contains Ethereum account and act as an IoT device. The smartphone acts as a tool that can configure desired policy in the environment. A centralized server will not exist in this case as devices will only update or make transactions through the Ethereum network. Due to the embedded consensus algorithm, attacker cannot simply forge or modify any data in this network. Furthermore, RSA algorithm including public key and digital signature are used to validate sender and receiver of a transaction to strengthen the protection of network. There are a few weaknesses can be seen in this architecture. As mentioned earlier, there was no server and each device need to store partial transactions of Ethereum. This causes a problem such that each device needs to have immense amount of storage space due to huge amount of transactions from all connected IoT devices in future. The next limitation is that all IoT devices need to build by specialized components in order to generate sufficient computational power to calculate and solve consensus algorithm. It is infeasible and too expensive for small IoT devices to have a large storage space and fast computational ability at the same time. To overcome these limitations, adding a cloud server and local gateway into the architecture would be a way. The cloud server should not be part of the BC network as it plays a role to store IoT data transactions and provide services only. A smart contract can be added into the BC to redirect request to the server. The local gateway suggested here is to limit amount of devices that connects to the BC network. Rather than including all devices into the BC, only the gateway will be involved in it to compute the consensus algorithm so that only the gateway has to be a specialized gadget and other smart devices could just connect to it to get service.

As (Chapade, Pandey & Bhade 2013) mentioned, a DoS or DDoS attack on cloud server could be detected and mitigated by using distance estimation based DDoS detection technique which is simple but effective. Previous studies and methods mostly focused on reducing timeout period and expand length of queues. However, it is not an ideal way because there are many queues in a TCP server system such as HTTP, SMTP, and FTP. Enormous amount of memory in an operating system is required when queues were lengthened. When timeouts were shortened, remote users with slow connection speed would never get to link with the server and it also affects new outgoing connections. Due to these limitations of former techniques, distance estimation based DDoS detection technique is introduced to resolve these problems. It is a way such that it uses exponential smoothing estimation to forecast the distance mean value in the subsequent session. The detection of anomaly depends on setting of normality and deviation. A simulation was done to generate the rate of detecting an attack and rate of false positive happened and it shown a high detecting rate and low false positive rate. After detection of strange request or packet, actions like blocking and time out can be taken on the suspicious network. However, the cloud still stands a chance by getting overwhelmed by malicious users if description of normality and deviation is incorrect. This problem can be solved by only permitting certain network to be connected to the cloud server. By using this method, any unknown networks must not be able to link with the server. There exists a type of BC network which is called consortium BC network. Consortium BC is partly private in the sense that it only allows predetermined participants to send and retrieve data to or from the cloud server. It prevents unauthorized users to append new blocks by setting policies in the miner machine itself.

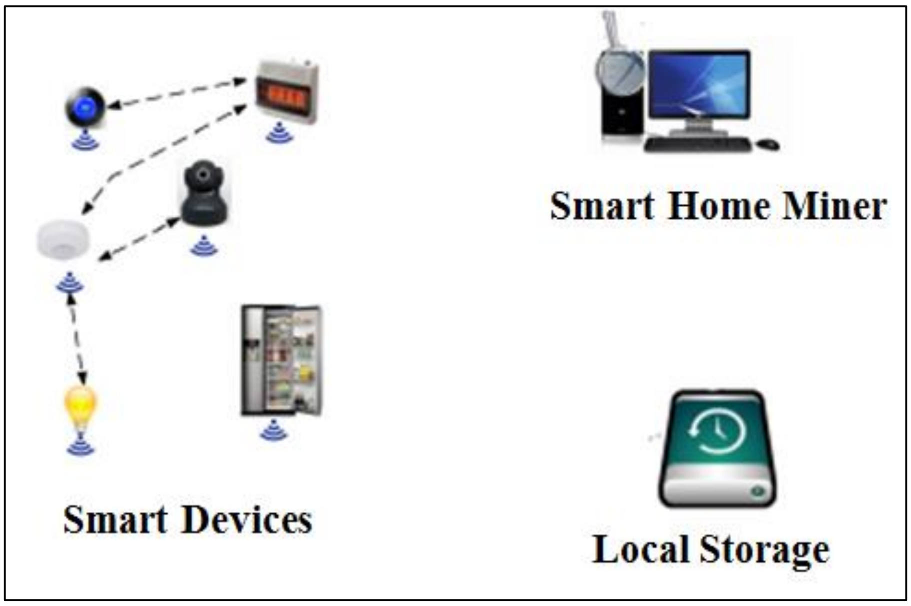


Figure 2.1: Components in local private BC

Then, in (Dorri et al. 2017) it uses a local private BC within each smart home itself. Three types of devices are included in a local private BC which is smart home miner, local storage and IoT smart devices as shown in Figure 2.1. The smart home miner will process input and output transactions. It has similar abilities as other normal miners like granting authorization, authentication, and inspecting transactions. The miner also manages a local storage. The local storage can also be known as backup drive that stores data locally. It can be a stand-alone device or integrate with the smart home miner. In this research, smart devices in the house can communicate and request data directly with each other. A device can store data either in the local or cloud storage. To store local data, miner needs to assign a shared key between device and storage as a starting point of transmission. To store data remotely, miner will first authorize the device, extract the last block's hash and send them together with the data to the cloud storage. After that, the cloud storage will return a new block-number to miner for future use. In this paper, it refers to (Dorri, Kanhere, & Jurdak 2016) which declare that it is an anonymous process when storing data into the cloud. Research paper (Aung & Tantidham 2017) is another study which is quite similar to (Dorri et al. 2017). It also uses private local BC, smart home miner and local storage in the SHS. The only different is that it deploys Ethereum in its smart home miner. With this Ethereum private BC, owner can log into their account to check all transactions history made within the SHS before. Other than that, rules and policies can be set easily to manipulate and restrict certain transactions. From (Dorri et al. 2017) it brings confidentiality because only authorized user can interpret his or her data. By the nature of BC, integrity is achieved in the sense that data or information has never been altered by anyone else other than the user from start to end. Moreover, these smart home devices are safe from malicious request since only devices with the shared key published by the miner can take part in a transaction. Nevertheless, there are also some flaws too. One of the weaknesses here is the overloading work for the home miner. A miner needs to process tons of transactions from each device in the smart home and in the same time it may needs to communicate with the cloud server. A device needs to wait for a longer time until its turn. Meanwhile, there is a relationship between number of jobs and energy consumption. The more jobs need to be done, the greater the energy consumption will be. There is a suggestion which is removing the local storage and keep all SHS data into the cloud server since cloud server can perform all functions like local storage and at the same time it can conduct more operations and services. Home miner do not has to decide which storage to store and this can reduce jobs greatly when there are lots of transactions. Meanwhile, data increases massively in the long run of SHS and this means more storage space is needed to store them. Without the local storage, cost used to build SHS will reduces since local storage no longer exists.

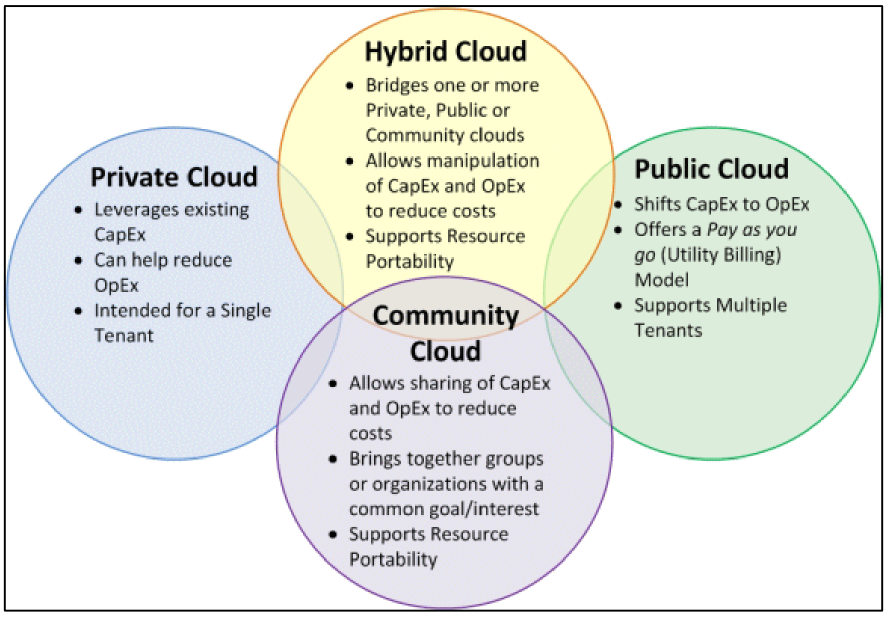

Hybrid Cloud
• Bridges one or more Private, Public or Community clouds
• Allows manipulation of CapEx and OpEx to reduce costs
• Supports Resource Portability
Private Cloud
• Leverages existing CapEx
• Can help reduce OpEx
• Intended for a Single Tenant
Public Cloud
• Shifts CapEx to OpEx
• Offers a Pay as you go (Utility Billing) Model
• Supports Multiple Tenants
Community Cloud
• Allows sharing of CapEx and OpEx to reduce costs
• Brings together groups or organizations with a common goal/interest
• Supports Resource Portability

Figure 2.2: (Cloud deployment models 2017)

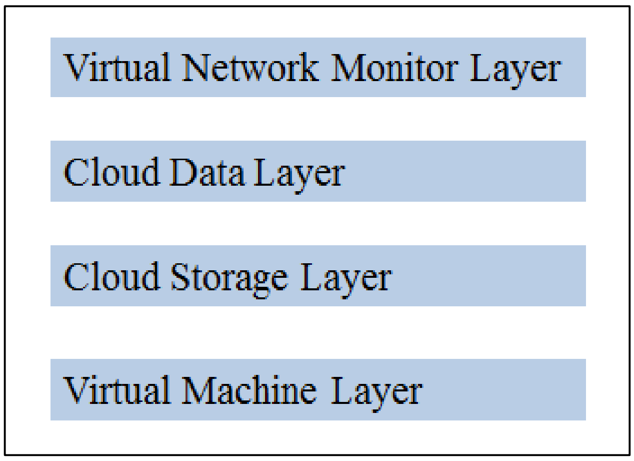


Figure 2.3: Layers of Cloud

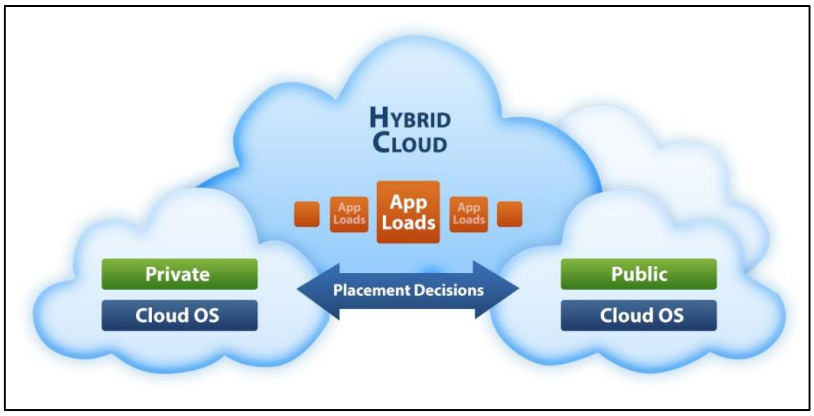


Figure 2.4: (Hybrid Cloud 2016)

According to (Sharma et al. 2017), there are several cloud deployment models such as public, private, community and hybrid as shown in Figure 2.2. It explained numerous and different kinds of embedded threats that a cloud possesses. Figure 2.3 shows the architecture of cloud network. There are several methods in this paper to secure cloud. One of them will be securing cloud layers by layers instead of protecting it as whole. It says that layered security is of supreme level and ensures all hypervisors to work perfectly. It also tells different types of clouds need to have different way to secure it. For public cloud, strong firewall should be set up. For private cloud, a virtual network invasion detection system must build in each virtual machine. Then, a response mechanism should be prepared so that it can report to user when there are intruders. For hybrid cloud (Figure 2.4), multi-tenant technologies must be applied after separating users at network level. Cloud encryption and Split-key encryption (splitting key into half and keep by different person) can be deploy as a defend mechanism for hybrid cloud too. All of these solutions are changes made from existing technologies so that they can be more trustworthy and effective but this paper did not state any authentication method as one of the solution. There should be only permitted individual be able to access the cloud no matter it is which type of cloud. This issue can be solve by setting up and apply policies header in the BC into gateway device of the user so that certain regulation and management can be enforce

Chapter 3 System Design

3.1 Reasons of Introducing BC-SHS

Before introducing BC into the SHS, SHS had the ability to manipulate data or activities in the smart home directly and with little security checking and monitoring only. This statement was declared because usually all of these IoT devices will be connected to the internet leaving a huge security hole on every single smart home device. Even if a smart home hub was installed which it can collect all sensor's data and be the only device connects to the internet, someone else can still spoof and pretend to be it. However, with BC implemented into the system, the smart home hub needed to go through many levels of security before it could reach the cloud server. For instance, the hub needed to be one of the node of the consortium BC network in order to utilize the cloud service and it also needed to went through the consensus process which was it needed to be verified by most of its peers. On the other hand, smart contract was also one of the reason of implementing BC into the system. As smart contract automated the validation of users and data, it had provided another layer of security which was blocking any unauthorized users or invalid of data from approaching the cloud server.

3.2 System Architecture

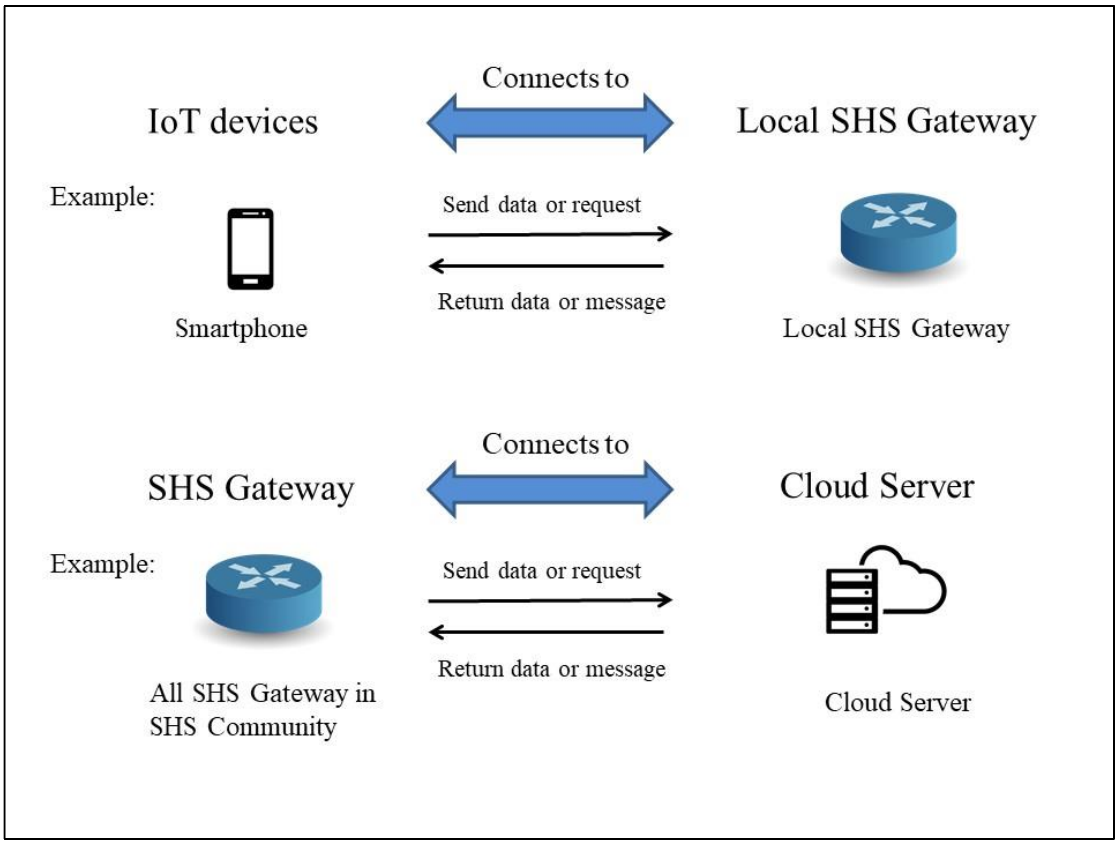


Figure 3.1: Network flow of cloud-connected SHS

First of all, there will be a community of SHS connected through cloud server and lots of IoT devices will connect to local SHS gateway device as shown in Figure 3.1. Within each smart home, there is a local gateway device act as main controller in SHS that could connect to the cloud server. There are three categories of devices attached with sensor nodes which are home appliances, user interface and monitoring and control devices. Smart devices and sensor nodes can communicate with the local gateway device through wired protocol like RS-485 or wireless technology such as ZigBee and Bluetooth Low Energy (BLE). The SHS gateway device is responsible to collect data from sensor nodes, perform limited data processing such as compression and analysis, and then forward the data to cloud server. It is the only node that can access to the internet in the proposed BC-SHS as well.

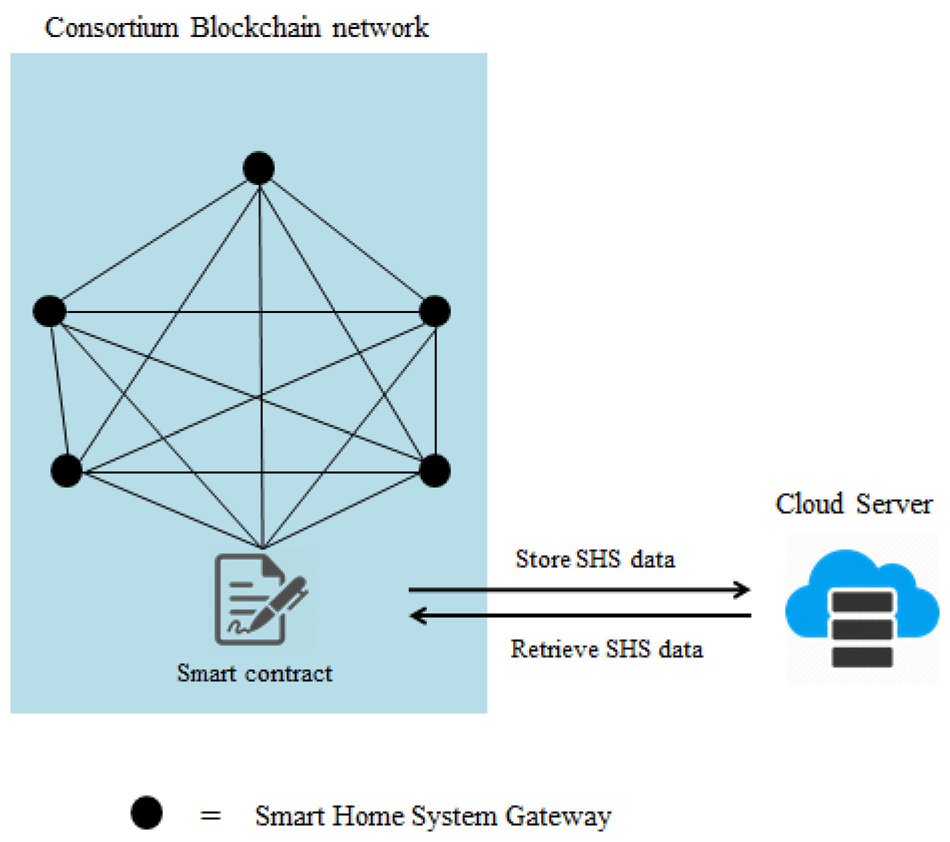


Figure 3.2: Architecture of consortium network in BC-SHS

In the BC-SHS, there are five main components which are consortium BC network, IoT sensor nodes, SHS gateway device, cloud server and smart contract. Each component has its own function to perform. For consortium BC network, it permits the trusted nodes to initiate transaction, participate in block verification and conduct consensus or mining. The role of IoT sensor nodes is to collect relevant sensor data and send it to the local SHS gateway device. They communicate through wireless or wired protocol but IoT sensor nodes will not be connect to the internet. Note that IoT sensor nodes are not part of BC network due to their limited memory space and low computational capability. Without these two abilities, they could not keep the public ledger, perform verification and mining. Next, SHS gateway device will be representing trusted node to carry out actions such as request and store data, perform block validation and consensus in the consortium BC network. This device has to perform great amount of computational and communication work all the time so it must be a specialized embedded device with vast computational power. In this project, Raspberry Pi is a better choice to simulate a SHS gateway device. Then, cloud server is here to deal with requests from SHSs and to store SHS data for further query and processing intention. In BC-SHS, several cloud servers may exist when each SHS subscribes to different services contained in multiple cloud servers. All cloud servers also own and publish a specific smart contract which has its own specific address in order for SHS gateway to interact with it. A smart contract is a set of policies or conditions that a SHS gateway needs to achieve before it can proceed with desired actions within the cloud server. In order to interact with smart contract, SHS gateway has to target the smart contract's address, then start a transaction and provoke certain function in smart contract. After the transaction has been accepted, it will undergo the normal mining process. The BC will then keep valid transactions that triggered under this process.

During this project, Ethereum BC network will be chosen as the BC environment to be deploy due to its mature development and community support. It is an open software platform and programmable BC which allows users to construct their own operations they wish rather than choosing from pre-defined operations. Inside the core of it there is a runtime environment for smart contracts named Ethereum Virtual Machine (EVM) that is able to execute code of arbitrary algorithmic complexity. EVM is totally isolated and sandboxed in the sense that codes within it do not access to any network, file system or other processes. By utilizing the peer-to-peer network protocol in Ethereum, every node connected to the network will maintain and update the BC database. Note that in BC-SHS, nodes only record data transaction and it does not store IoT data collected from sensor nodes. Other than that, accounts also act as an important role in Ethereum. In general, there are two kinds of accounts, known as contract accounts and also externally owned accounts (EOAs). EOAs require human users control the Ethereum BC throughout the transactions and are controlled by private keys. On the other hand, contract accounts' embedded codes are similar to smart contract and it can run with or without users. Contract accounts will be a better choice because it is an automated helper in IoT environment. The mining algorithm that were to use in this project was PoW (Proof-of-Work). It was an algorithm that block will get mined and miners will get the reward depending on who solved the mathematical problem in the first place.

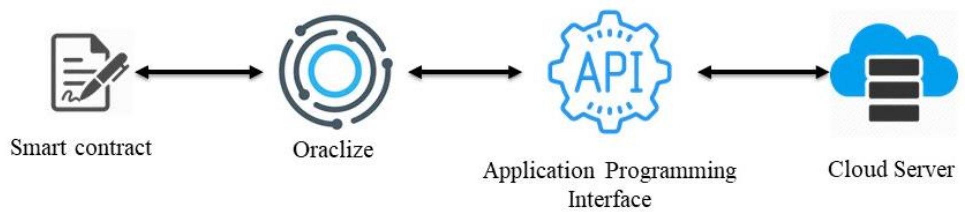


Figure 3.3: Inner architecture between smart contract and server

Figure 3.3 shows the inner architecture between smart contract and the server. In order for the BC environment, which was isolated from the outside world, to send and receive data to and from the server, smart contracts need to make use of a service named Oraclize. Oraclize provides oracle service for smart contracts and BC applications. It accepts a few data sources such as Uniform Resource Locator (URL), InterPlanetary File System (IPFS) and WolframAlpha. In this project, URL will be deployed instead of the others. This was because it enables access to any API or web page on the Internet. Moreover, the application programming interface (API) which fetches data to and from the server will be written in Hypertext Preprocessor (PHP) which is a server-side scripting language.

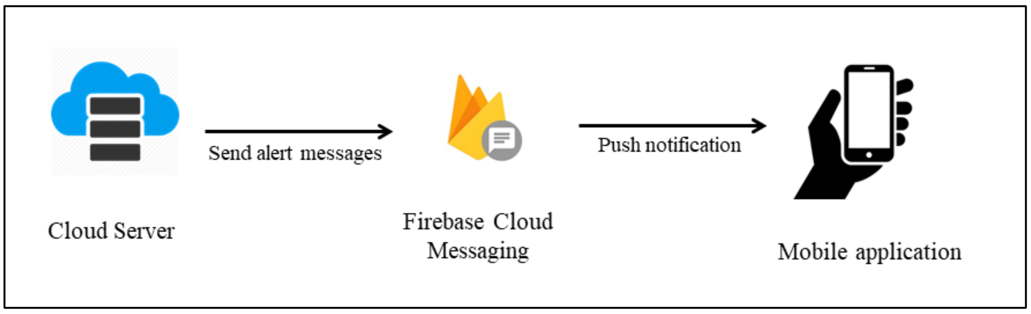


Figure 3.4: Data flow from server to smart phone

On the other hand, Firebase Cloud Messaging (FCM) service was used to facilitate the flow of data as shown in Figure 3.4. FCM was invented by Google that allows trusted parties to send notifications from their servers to end users' mobile device. FCM possessed a strong feature which allows servers to inform their users in a real time manner. Based on different settings, messages from the server can be send to an individual or through a particular topic that users possessed or subscribed. On top of that, a mobile application must be installed into the particular device in order for the FCM server to locate it. Other than sending notification from the cloud server, FCM also provides a notification console graphic user interface (GUI) that can perform multiple functions such as check status of a notification, check number of daily active users, check users' location and many other analytic functions. FCM supports both Android, iOS and chrome web apps but in this project, only Android application was installed and used.

Tool	Functions
go-ethereum/ geth	command line interface implemented in Go programming language and is used for running full Ethereum node
Puppeth	a tool to create private Ethereum network
Truffle	Ethereum based development environment and testing framework
Oraclize	an oracle service for smart contracts
Ethereum-bridge	a tool enables any non-public BC instance to interact with the Oraclize service
MySQL	a relational database management system
Android Studio	integrated development environment (IDE) for android development
VirtualBox	A virtual environment to simulates another operating system

Table 3.1: Tools and their functions

Table 3.1 shows a list of essential tools that will be used throughout the development of this project and their functions respectively.

Most of the time, this project was conducted under Linux environment. One of the Linux based operating system called Ubuntu was the first choice in terms of friendliness and complexity. Linux based operating system was preferred because there are more tools available and more developer friendly. By using this operating system, packages and modules can be handled and customized easier.

3.3 Design Specification

- A BC system that can support 100 users
- Database that stores data such as timestamp, service type and sensor data
- Database that can hold maximum 50GB of data in total
- Smart contracts that can check whether an account meets certain condition, enable or blocks an account to perform smart contract’s function, change the state of BC and send request to server through Oraclize service
- APIs that can perform actions such as create database, store data, retrieve data, calculate heart rate condition, communicate with firebase server and send push notification
- Mobile application that is able to receive push notification sent from FCM

3.4 Development Lifecycle

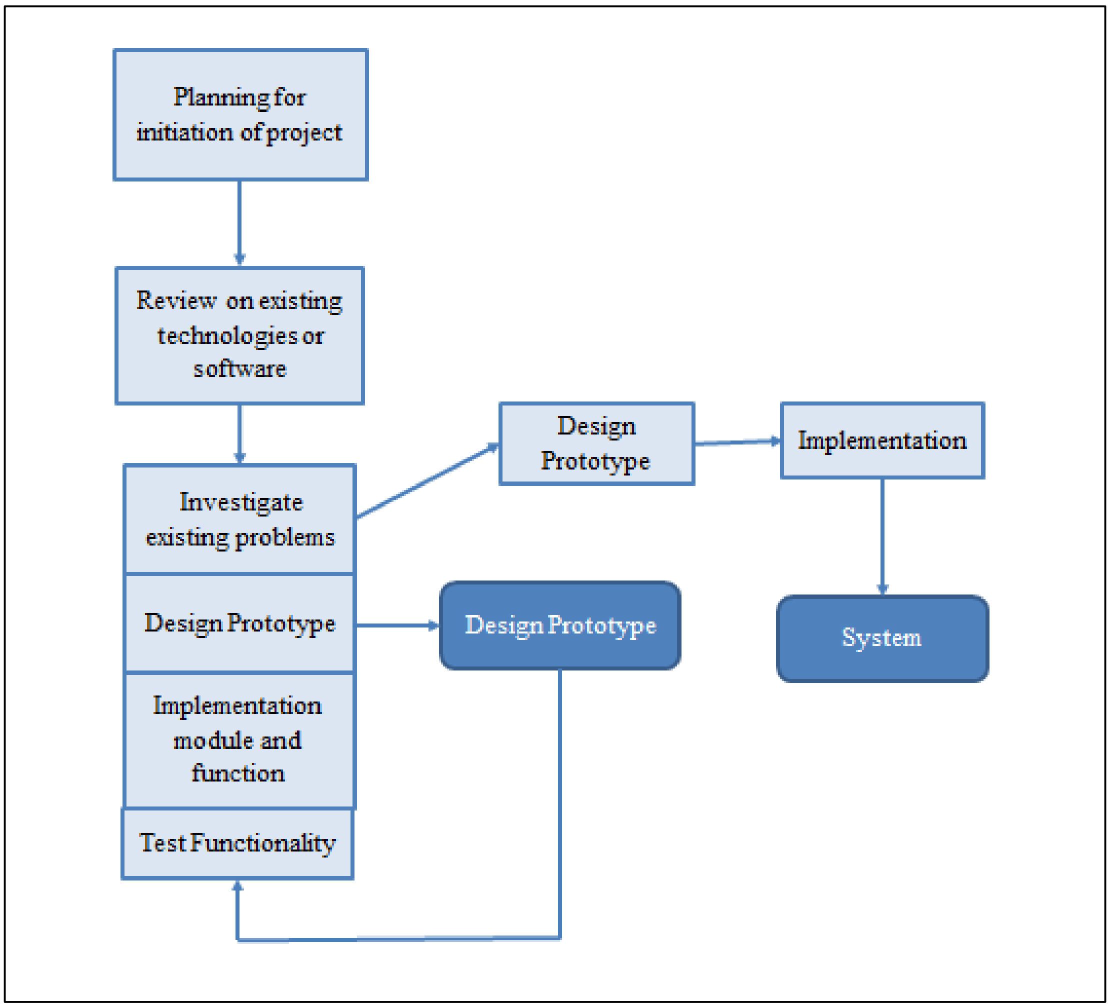

Planning for initiation of project
Review on existing technologies or software
Investigate existing problems
Design Prototype
Implementation module and function
Test Functionality
Design Prototype
Design Prototype
Implementation
System

Figure 3.5: Project development lifecycle

According to the plan as shown in Figure 3.5, throwaway prototyping lifecycle model will be the guide throughout development of this project. Throwaway prototyping is also known as rapid prototyping which refers to development of a model that finally will be disposed rather than becoming the final software. According to this model, the whole system will be building up piece by piece and every time a smart part from it will get evaluated. After improvement is made then the prototype will be discarded or thrown away. Throwaway prototyping can deliver a better quality system as it is evaluated and enhanced once in a while throughout the whole system development process. Other than that, any problems could identify easier in the early stage when the system is incomplete.

Chapter 4 Implementation

4.1 Setting up private Ethereum BC environment

The Ethereum BC was created by using Puppeth to generate the genesis file and define the network name. A few commands with all the required parameters were defined to create a private node. In order to speed up the process of mining in this private BC environment, the difficulty of the consensus algorithm was set relatively easy compared to the original complexity. Other than that, the gas limit of the BC had set relatively high due to the high complexity of smart contracts. To avoid error such as insufficient funds when calling Oraclize service, some ethers were funded into the smart contract. To achieve the scenario such that multiple nodes running on a single machine, Ethereum provided a way which was create different nodes in different directories. It meant that each various directory represented each Ethereum node.

Command to start genesis file in node:

```
geth --datadir data1 init genesis.json
```

Command to create a node:

```
geth --datadir data1 --port 30301 --networkid 123 --rpc --rpcport 8101 --ipcdisable
--nodiscover console 2>>data1/01.log
```

Command to connect two peer nodes (in Geth console):

```
admin.addPeer([executing “admin.nodeInfo.enode” from another peers])
```

Command to fund ether into the smart contract (in Geth console):

```
eth.sendTransaction ({from:eth.coinbase, to: [Smart Contract Address], value:
web3.toWei(10, “ether”)})
```

In the meantime, ethereum-bridge needed to be initiated in another console before using the smart contract’s function. Noted that the same account that used to initialize the ethereum-bridge cannot be used to deploy other contracts as it will result in unexpected behaviour. This was because the callback address will be used to deploy those contracts which was against the rules set in the connector side from the Oraclize service.

Command to start the ethereum-bridge:

```
node bridge –a 0 -H 127.0.0.1:8101
```

Before interacting with the smart contract through Truffle, the port in the truffle.js needed to be changed into the desired port. For example, if the node was started with port 8101, then the port attributes in the file must be turned into 8101 as well.

```
module.exports ={
    networks:{
          development{
               host: “127.0.01”,
               port: 8101,
               network_id: “*”
          }
    }
};
```

Command to interact with smart contract (in Truffle console):

To sign up:

```
manageData.deployed().then(inst=>inst.signUp.sendTransaction({from:eth.coinba
se}))
```

To subscribe manage data service:

```
manageData.deployed().then(inst=>inst.subMngData.sendTransaction({from:eth.c
oinbase}))
```

To store data:

```
manageData.deployed().then(inst=>inst.storeData.sendTransaction(“ServiceName
”, “100”, {from:eth.coinbase}))
```

To get data from server:

```
manageData.deployed().then(inst=>inst.getFromDatabase.sendTransaction({from:
eth.coinbase}))
```

4.2 Smart Contract related APIs

MySQL database was used and three APIs was created. The first API was made for registration purpose which will accept a parameter which was the account number of the sender. It was used to create a database named by the account number passed in. After that, it will create a table called storage with three attributes in it. Those attributes are id, service and data as shown in appendix A.1. Additionally, this API had some extra steps apart from the other two APIs because creating databases from a PHP script was not supported directly in this web hosting service.

The second API was made for the purpose of storing data. This API will accept a few parameters such as account number, timestamp, service type and data. The account number is used to connect to the respective existing database for each smart home. Timestamp will be the ID, which was the primary key for the storage, so that it will be always differ for each row. When there were more than ten rows of data stored, a title of heart condition will be set according to the average of the latest ten heartbeat stored. In the meantime, the title will be sent to the particular device through the aid of FCM. This API was shown in appendix A.2.

The third API was made to fetch data and it accepts only parameter, which is the account number. Upon request, it will return its data to the requester with a JavaScript Object Notation (JSON) encoded format. This API was shown in appendix A.3.

4.3 Firebase related APIs

Appendix A.4 was a configuration file that defines the firebase server key that retrieved from the firebase console. By declaring the key inside the server, it authorizes the application to access into Google services such as send messages from FCM legacy protocols. Without this key, firebase backend will not be able to find the respective account and server. The key can be get from the cloud messaging settings. This API was used in later stage by Appendix A.6 as an input of HTTP (HyperText Transfer Protocol) header to pass to firebase server.

Before sending any notification, all required elements need to be constructed. Appendix A.5 was a class that played a role in declaring push notification elements such as title, message, image and data. These elements were declared in private so that only functions within can alter its value and to enhance their security. In order to access and change their value, setters of these elements were built. Later on in Appendix A.6, all these elements will be encoded into JSON format and sent by HTTP POST operation.

Appendix A.6 was created to communicate with firebase servers and make use of FCM services. In this API, Client URL (cURL) was used to achieve the aim. cURL is a command-line tool that aids in transmitting data by utilizing various type of protocols such as HTTP, File Transfer Protocol (FTP), Telnet and more. In this case, HTTP was used to send POST messages which consist of push notification elements to firebase servers through the help of cURL. Other than that, this API also had another function which was to decide who will be the receiver of a particular notification. There were three types of grouping method in total. The first type will be single user only which means only one of the registered device will be receiving the notification. The second type will be based on the topic subscribed which means only users that subscribed to a particular topic will be receiving that notification. These topics can be created and customized by developer based on their requirements. The third type will be multiple user which means a group of registered users will be receiving the notification. In this project, there were only a single user type grouping method used.

4.4 Smart contract development

There were total of three smart contracts developed as shown in appendix B and were linked together through import function. Moreover, the OraclizeAPI contract was also deployed and imported in these contract so that Oraclize knows where should it return the values or data after it had successfully retrieved from a source. The contract ‘register’ in appendix B.1 was used to interact with the API for registration in appendix A.1. When calling the contract’s function, it will send a HTTP POST request in JSON format to the targeting URL through the help of Oraclize service. In appendix B.2, it is a contract named ‘UserConfig’. It keeps track whether a user had registered or subscribed to a service. It also stores the latest result data returned from the server. Then, in appendix B.3, it is a contract named ‘manageData’. It provides services for users to store and retrieve their data from their respective database but only users that had been subscribed to this service will be able to utilize it.

4.5 Mobile Application development

Android Studio was used for the development of the mobile application. In appendix C.1, all constant values were defined and these values were used across the entire application. In appendix C.2, it served for showing the message which includes title, image, message and also timestamp in the notification tray. In appendix C.3, the class was used to receive a unique firebase registration id for each different device. The registration id or token received was used in the server to send notification to a particular device. In appendix C.4, the class was used to receive messages from firebase. In firebase, there are two types of messages or payload which is notification and data. For notification message type, predefined data elements needed to be used. For data message type, those data elements can be customized based on various requirements. In appendix C.5, FCM information was added so that the application can utilize the service. Appendix C.6 and C.7 shows the main interface’s code together with the registration id string received.

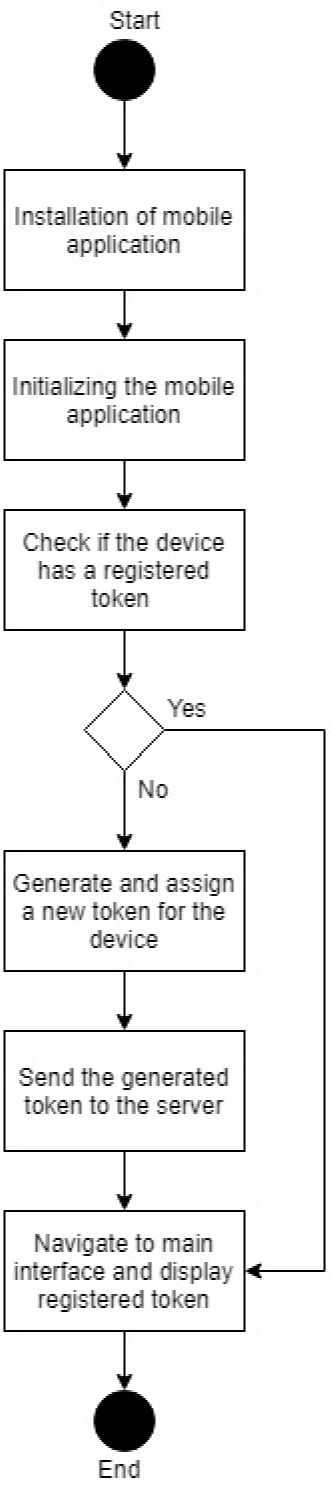

Start
Installation of mobile application
Initializing the mobile application
Check if the device has a registered token
Yes
No
Generate and assign a new token for the device
Send the generated token to the server
Navigate to main interface and display registered token
End

Figure 4.1: Events happened during installation

In Figure 4.1, events happened during the installation of mobile application was shown. After the installation of the mobile application, the mobile application will initialize and check if the mobile device was the first time installing the mobile application or the application cache data had been cleared before. If that was the case, the application will generate a new and unique registration token or id for the device. In the same time, the registration token will be stored in shared preferences. Shared preferences was a way of storing and retrieving data in a key-value pair manner which meant every value had a unique identifier known as key. After the token was generated, it will be sent to the targeted server application that was set in the build gradle file of the project. Then, the main interface as well as the registration id will be shown.

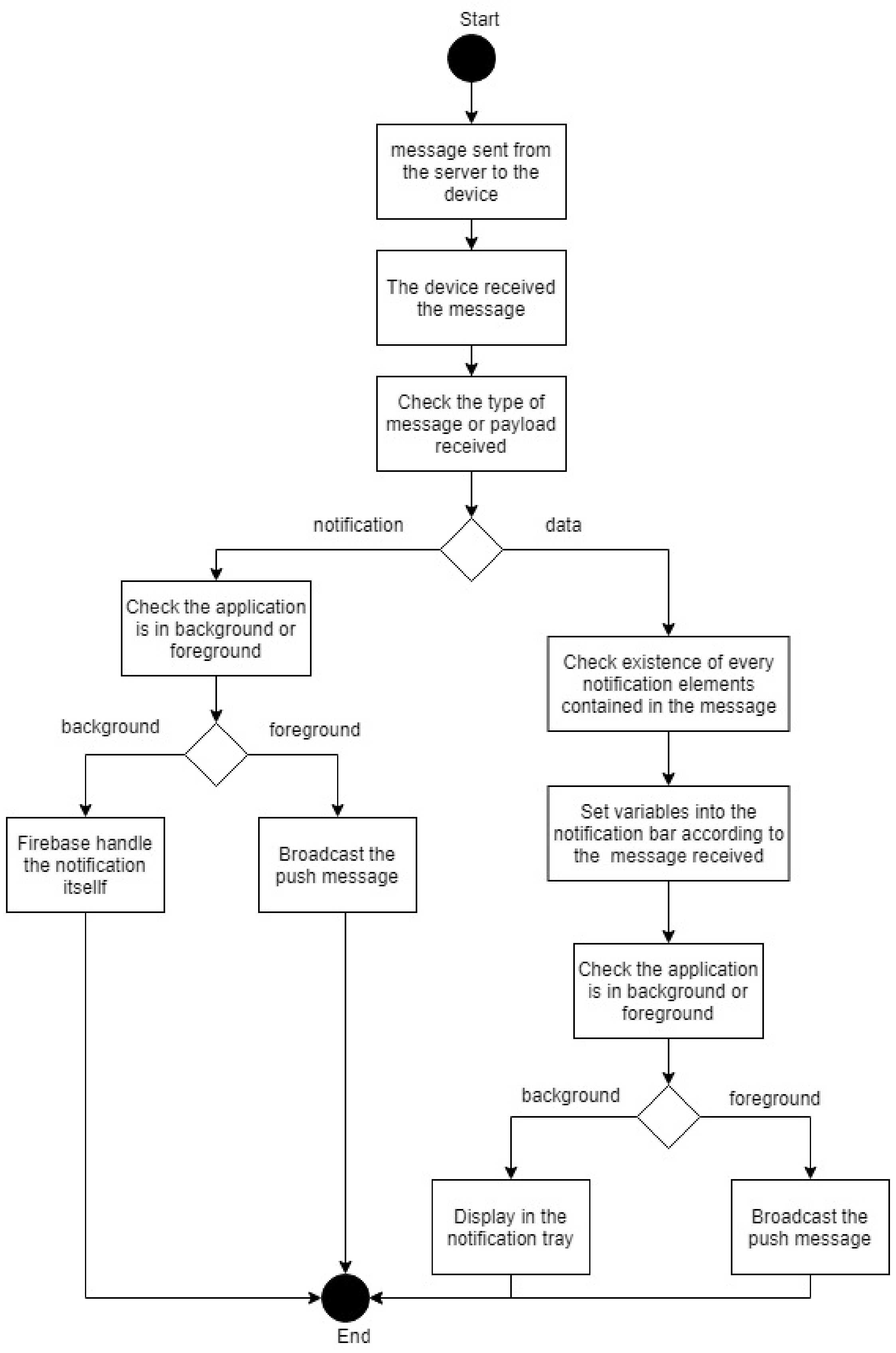
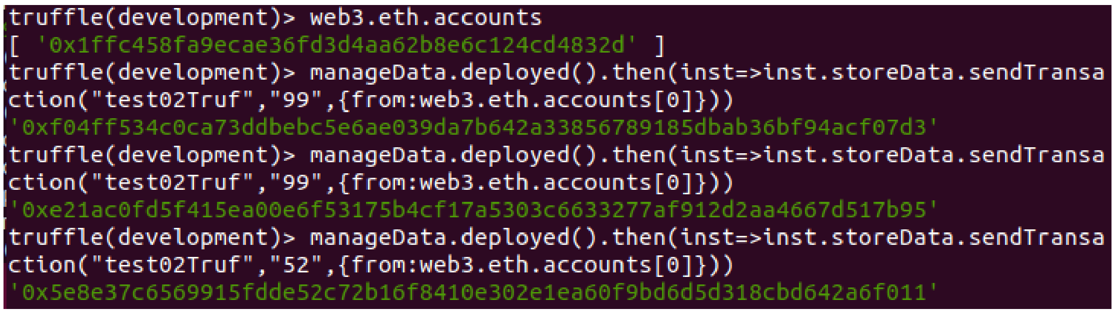

Start
message sent from the server to the device
The device received the message
Check the type of message or payload received
notification
data
Check the application is in background or foreground
background
foreground
Firebase handle the notification itsellf
Broadcast the push message
Check existence of every notification elements contained in the message
Set variables into the notification bar according to the message received
Check the application is in background or foreground
background
foreground
Display in the notification tray
Broadcast the push message
End

Figure 4.2: Events happened when receiving message

Figure 4.2 shown events happened when receiving a message from the firebase server. After a device had received the message, the type of the message or payload will be checked to decide whether it was a data or notification payload. Basically notification messages were taken care by firebase SDK (Software Development Kit) itself. Notification messages can be sent through firebase console user interface. This meant that this type of messages had less control and customization permission provided. For data payload, it must be handled by the application and firebase console user interface cannot send this kind of messages. Server side logic was needed to convey notification through firebase API in this message type. One of the activity on the figure above was broadcast the push message and it meant that notifying the message received to other listening parts of the application such as user interface objects.

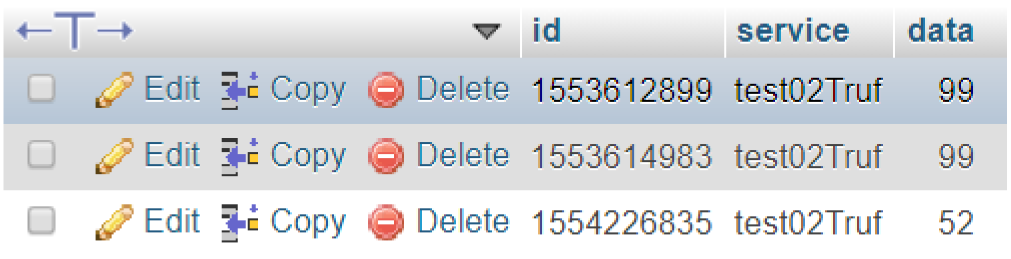
Chapter 5 Results and Discussions

5.1 Verification of Defined Objectives



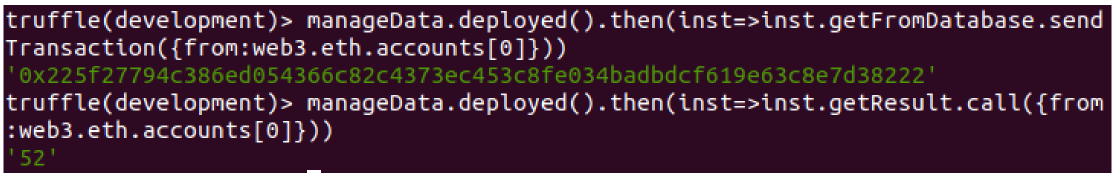
```
truffle(development)> web3.eth.accounts
[ '0x1ffc458fa9ecae36fd3d4aa62b8e6c124cd4832d' ]
truffle(development)> manageData.deployed().then(inst=>inst.storeData.sendTransa
ction("test02Truf","99",{from:web3.eth.accounts[0]}))
'0xf04ff534c0ca73ddbebc5e6ae039da7b642a33856789185dbab36bf94acf07d3'
truffle(development)> manageData.deployed().then(inst=>inst.storeData.sendTransa
ction("test02Truf","99",{from:web3.eth.accounts[0]}))
'0xe21ac0fd5f415ea00e6f53175b4cf17a5303c6633277af912d2aa4667d517b95'
truffle(development)> manageData.deployed().then(inst=>inst.storeData.sendTransa
ction("test02Truf","52",{from:web3.eth.accounts[0]}))
'0x5e8e37c6569915fdde52c72b16f8410e302e1ea60f9bd6d5d318cbd642a6f011'
```

Figure 5.1: Data sent from the gateway to cloud



				id	service	data
☐	Edit	Copy	Delete	1553612899	test02Truf	99
☐	Edit	Copy	Delete	1553614983	test02Truf	99
☐	Edit	Copy	Delete	1554226835	test02Truf	52

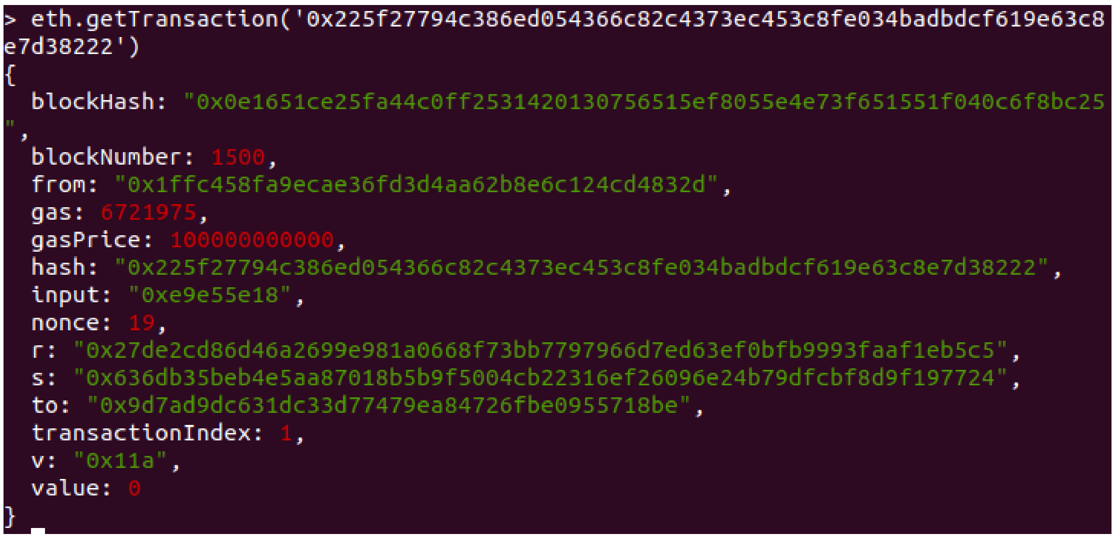
Figure 5.2: Data received in the cloud database



```
truffle(development)> manageData.deployed().then(inst=>inst.getFromDatabase.send
Transaction({from:web3.eth.accounts[0]}))
'0x225f27794c386ed054366c82c4373ec453c8fe034badbdcf619e63c8e7d38222'
truffle(development)> manageData.deployed().then(inst=>inst.getResult.call({from
:web3.eth.accounts[0]}))
'52'
```

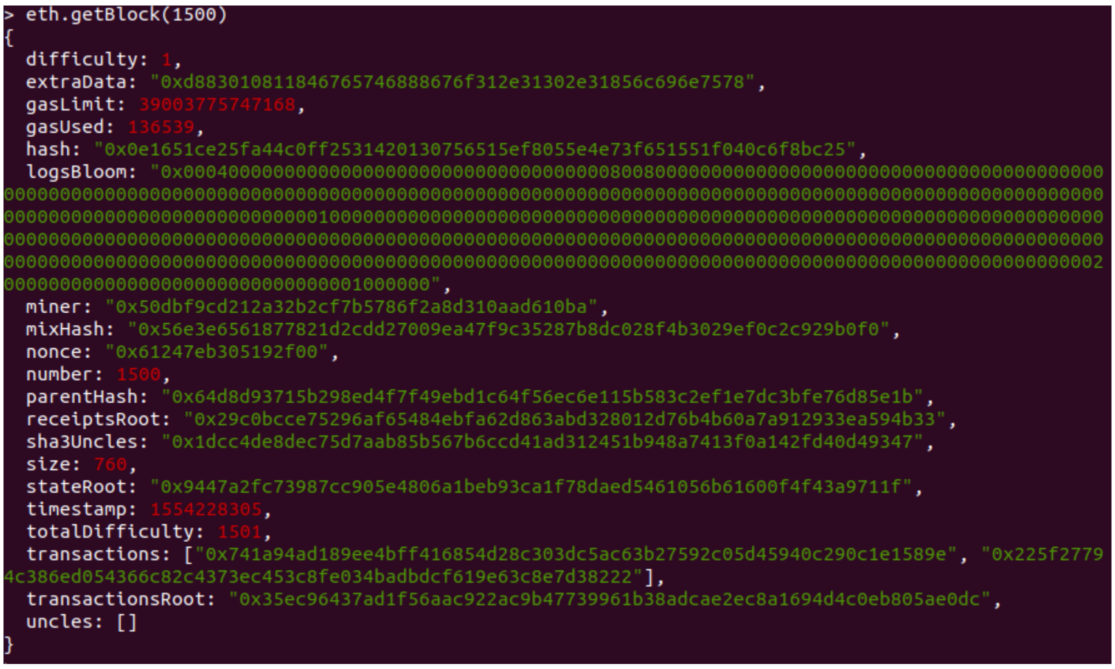
Figure 5.3: Data retrieved from the cloud database from the gateway

Figure 5.1 above shows an account sent a few dummy data to the cloud service from the smart home gateway and these dummy data will be 99, 99 and 52. Figure 5.2 was the table from the cloud database and it shows it had received and stored those dummy data sent from the smart home gateway. In Figure 5.3, the smart home gateway had sent a request to retrieve data from the cloud and it also shows successful retrieval of the last row or the most recent data stored into cloud previously. Hence, this had proven that the objective of developing a cloud service that is able to receive, return and store data to and from the smart home community had achieved. In the meantime, it also verified that the smart home gateway has the ability to request, send and retrieve data which met one of the objective stated. For the mining function, entering the command miner.start() in the Geth console will initiate the mining process in the smart home gateway and miner.stop() command to stop mining.



```
> eth.getTransaction('0x225f27794c386ed054366c82c4373ec453c8fe034badbdcf619e63c8
e7d38222')
{
  blockHash: "0x0e1651ce25fa44c0ff2531420130756515ef8055e4e73f651551f040c6f8bc25
",
  blockNumber: 1500,
  from: "0x1ffc458fa9ecae36fd3d4aa62b8e6c124cd4832d",
  gas: 6721975,
  gasPrice: 100000000000,
  hash: "0x225f27794c386ed054366c82c4373ec453c8fe034badbdcf619e63c8e7d38222",
  input: "0xe9e55e18",
  nonce: 19,
  r: "0x27de2cd86d46a2699e981a0668f73bb7797966d7ed63ef0bfb9993faaf1eb5c5",
  s: "0x636db35beb4e5aa87018b5b9f5004cb22316ef26096e24b79dfcbf8d9f197724",
  to: "0x9d7ad9dc631dc33d77479ea84726fbe0955718be",
  transactionIndex: 1,
  v: "0x11a",
  value: 0
}
```

Figure 5.4: Example transaction details retrieved using transaction hash

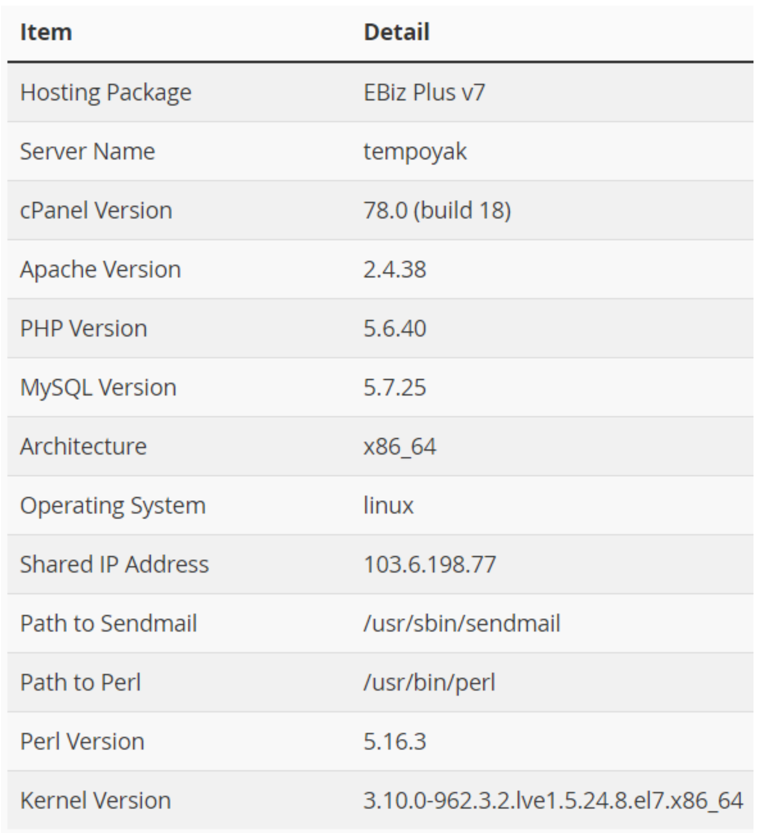


```
> eth.getBlock(1500)
{
  difficulty: 1,
  extraData: "0xd883010811846765746888676f312e31302e31856c696e7578",
  gasLimit: 39003775747168,
  gasUsed: 136539,
  hash: "0x0e1651ce25fa44c0ff2531420130756515ef8055e4e73f651551f040c6f8bc25",
  logsBloom: "0x000400000000000000000000000000000808000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000002
00000000000000000000000000000001000000",
  miner: "0x50dbf9cd212a32b2cf7b5786f2a8d310aad610ba",
  mixHash: "0x56e3e6561877821d2cdd27009ea47f9c35287b8dc028f4b3029ef0c2c929b0f0",
  nonce: "0x61247eb305192f00",
  number: 1500,
  parentHash: "0x64d8d93715b298ed4f7f49ebd1c64f56ec6e115b583c2ef1e7dc3bfe76d85e1b",
  receiptsRoot: "0x29c0bcce75296af65484ebfa62d863abd328012d76b4b60a7a912933ea594b33",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  size: 760,
  stateRoot: "0x9447a2fc73987cc905e4806a1beb93ca1f78daed5461056b61600f4f43a9711f",
  timestamp: 1554228305,
  totalDifficulty: 1501,
  transactions: ["0x741a94ad189ee4bff416854d28c303dc5ac63b27592c05d45940c290c1e1589e", "0x225f2779
4c386ed054366c82c4373ec453c8fe034badbdcf619e63c8e7d38222"],
  transactionsRoot: "0x35ec96437ad1f56aac922ac9b47739961b38adcae2ec8a1694d4c0eb805ae0dc",
  uncles: []
}
```

Figure 5.5: Example transaction details retrieved using block number

In BC-SHS, all transactions were recorded into the BC. By utilizing a specific transaction hash, which was generated when a transaction was initiated, its details such as block number, initiator, target and others can be retrieved. By executing the Geth command eth.getTransaction([Transaction Hash]), transaction details will be shown as in Figure 5.4. Moreover, some other transaction details such as timestamp, gas used and more can also be retrieved by utilizing the command eth.getBlock([Block Number]) as shown in Figure 5.5, provided that block number was recognized.

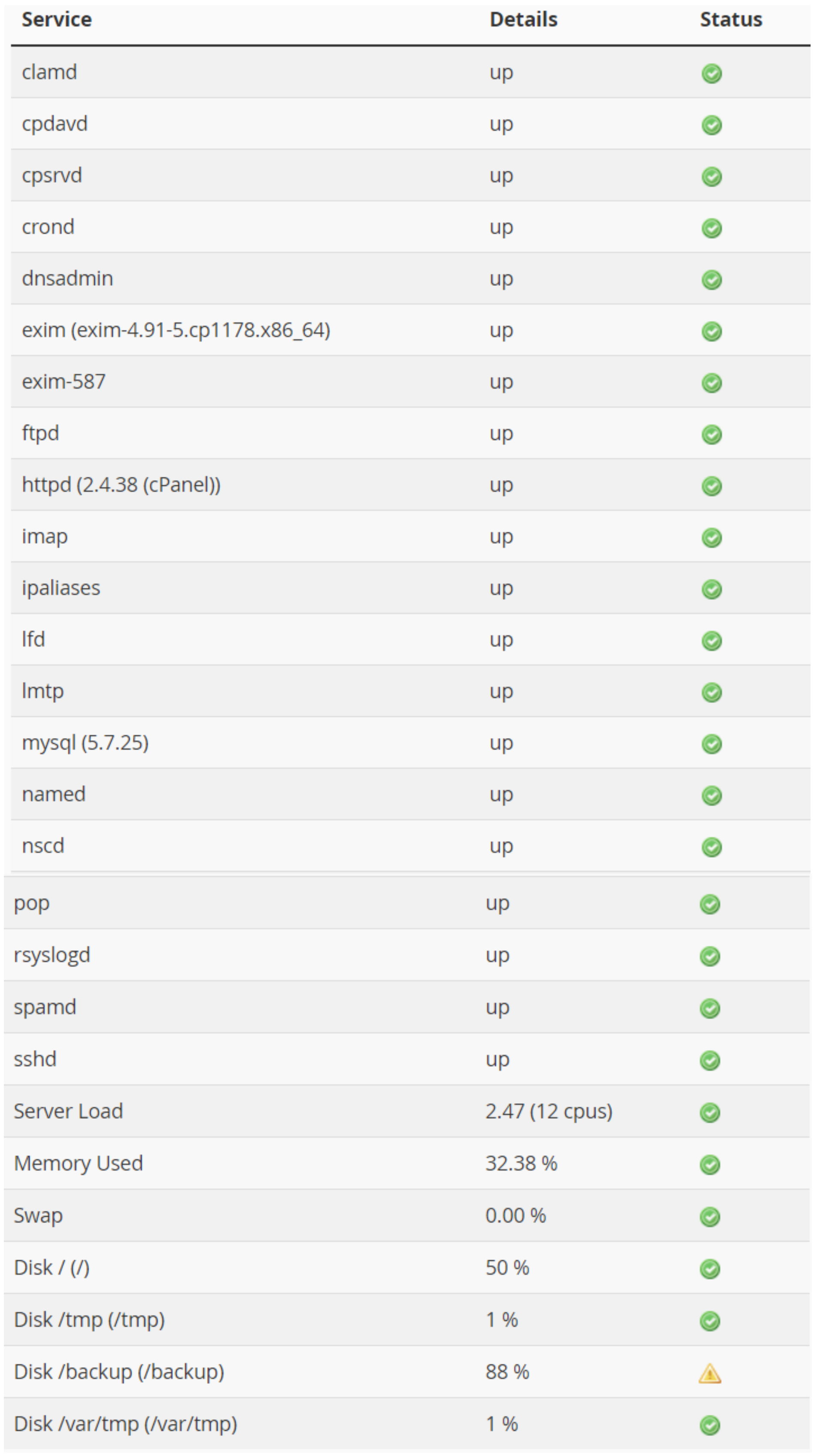
5.2 Server Information and Limitations



Item	Detail
Hosting Package	EBiz Plus v7
Server Name	tempoyak
cPanel Version	78.0 (build 18)
Apache Version	2.4.38
PHP Version	5.6.40
MySQL Version	5.7.25
Architecture	x86_64
Operating System	linux
Shared IP Address	103.6.198.77
Path to Sendmail	/usr/sbin/sendmail
Path to Perl	/usr/bin/perl
Perl Version	5.16.3
Kernel Version	3.10.0-962.3.2.lve1.5.24.8.el7.x86_64

Figure 5.6: Server information

In this session, the subscribed server information, services provided and its limitations will be displayed. This web hosting package was bought from Exabytes Network Sdn. Bhd. and it was named EBiz Plus v7 which was shown in Figure 5.6. There were several reasons of why this package was chosen. First of all, Oraclize needed to have to a URL that can be remotely accessible. Although it can be done by using tools like localTunnel to expose local server but it was unstable and lack of maintenance by the author. In order to expose this system more towards the industry, web hosting was one of the way to do so. Then, in this package it consisted half of the number of database that were needed to achieve and that was a good start for experimental purpose. It was also understood that the upgrade of package in the future can be done easily and without any hassle as only few clicking and payments needed. Last but not least, the customer service and support by this company was outstanding and helpful. 24-hours online technical support was also provided to solve any migration or implementation issues in the server. An online web hosting control panel named cPanel was included in this package. It was Linux-based and provided GUI and automation tools to facilitate users to host a web. Moreover, tempoyak will be the name of the server that cPanel used. Other than that, the version of apache, PHP and MySQL were shown too. The architecture of x86_64 means that it was 64-bit version of x86 instruction set. The shared IP address was the address that used for several sites on the server. Perl was a type text processing programming language which was also supported by this server.



Service	Details	Status
clamd	up	✓
cpdavd	up	✓
cpsrvd	up	✓
crond	up	✓
dnsadmin	up	✓
exim (exim-4.91-5.cp1178.x86_64)	up	✓
exim-587	up	✓
ftpd	up	✓
httpd (2.4.38 (cPanel))	up	✓
imap	up	✓
ipaliases	up	✓
lfd	up	✓
lmtp	up	✓
mysql (5.7.25)	up	✓
named	up	✓
nscd	up	✓
pop	up	✓
rsyslogd	up	✓
spamd	up	✓
sshd	up	✓
Server Load	2.47 (12 cpus)	✓
Memory Used	32.38 %	✓
Swap	0.00 %	✓
Disk / (/)	50 %	✓
Disk /tmp (/tmp)	1 %	✓
Disk /backup (/backup)	88 %	⚠
Disk /var/tmp (/var/tmp)	1 %	✓

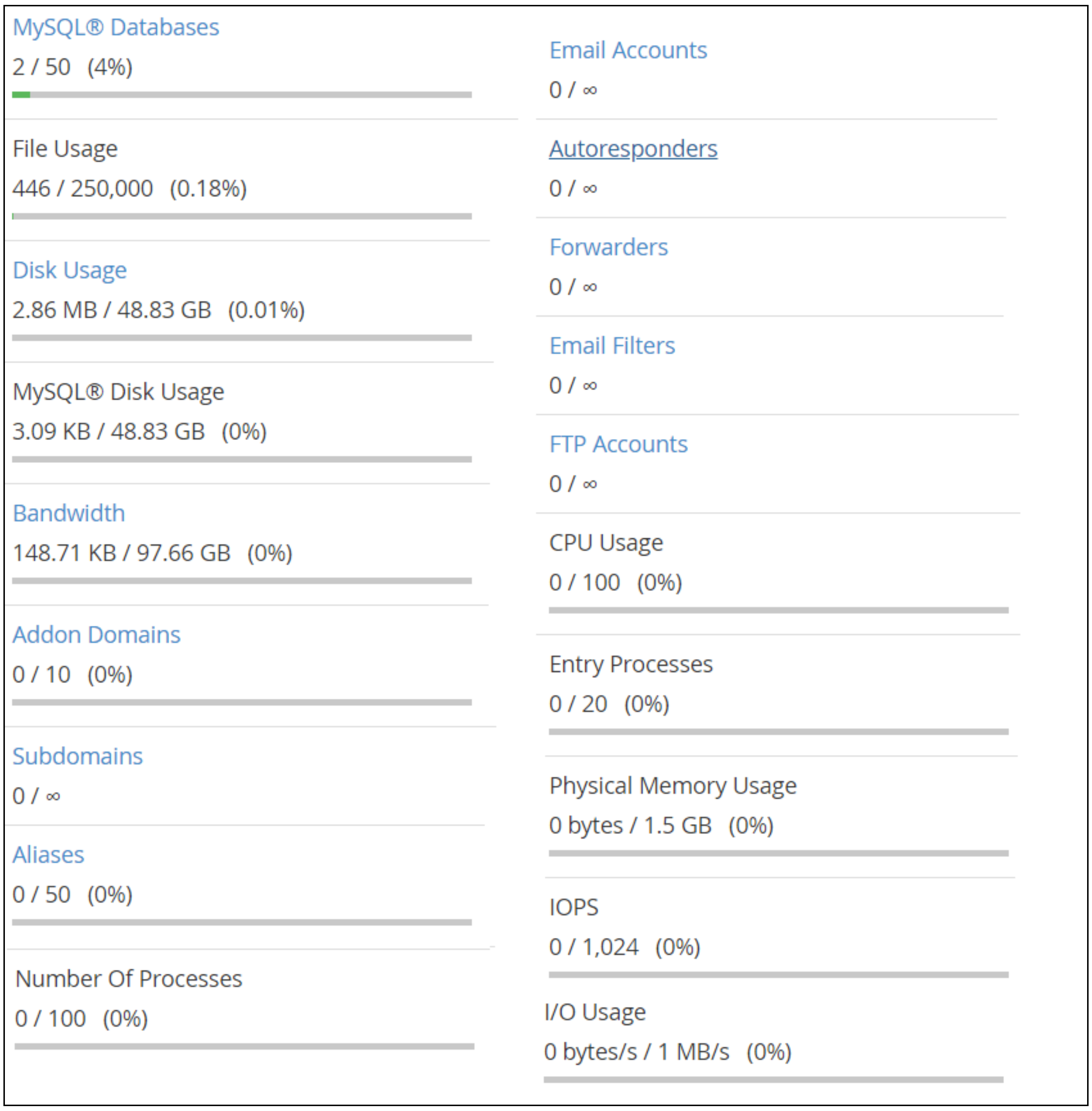
Figure 5.7: Services provided

Service Provided	Function
ClamAV Daemon (clamd)	Detect malicious software and viruses in the server
cPanel DAV Daemon (cpdavd)	Provides framework to create, alter, move documents remotely on the server
cPanel Server Daemon (cpsrvd)	Act as application server for cpanel
Cron Daemon (crond)	Aids in scheduling system task
cPanel DNS Admin Cache (dnsadmin)	Increase the speed andperformance on DNS update
Exim Mail Server (exim)	Enables the server to send and receive email
Exim Mail Server-port number (exim-587)	Mail server that listens to another port
FTP Server (ftpd)	Enables FTP service on the server
Web Server (httpd)	Processes HTTP client's request
Internet Mail Access Protocol Server (imap)	Processes mail download operations
Internet Protocol Aliases (ipaliases)	Allows adding of more than one IP address to the network adapter
Login Failure Daemon (lfd)	Scans the log file to prevent any brute-force attacks from outsiders
Local Mail Transport Protocol Server (lmtp)	Enables Exim to link with another mail server
DNS Server (named)	Runs customized name server
Name Service Cache Daemon (nscd)	Manages cache for name service requests
Post Office Protocol 3 Server (pop)	Manages how users download their email
System Logger Daemon (rsyslogd)	Log system activities and monitors web server
Apache SpamAssassin (spamd)	Filter spam messages
Secure Shell Daemon (sshd)	Allows SSH connection to the server
Server Load	Defines how many processes waiting to be process in the queue

Swap	Extra memory that can be used when physical memory was used up

Table 5.1: Service name and functions

In Figure 5.7, a series of services that was provided in this package was shown and Table 5.1 showed their names and functions. There were several terms needed to be explained on the table above. Daemon meant computer programs that execute continuously as background processes. Cron was a job scheduler in Linux-based operating system. Exim was a mail transfer agent that used simple mail transfer protocol to send mails from one to another device.



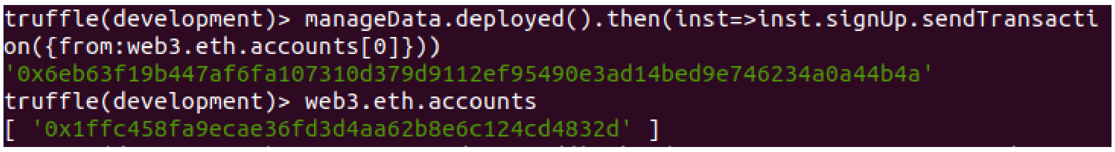
MySQL® Databases	2 / 50 (4%)
File Usage	446 / 250,000 (0.18%)
Disk Usage	2.86 MB / 48.83 GB (0.01%)
MySQL® Disk Usage	3.09 KB / 48.83 GB (0%)
Bandwidth	148.71 KB / 97.66 GB (0%)
Addon Domains	0 / 10 (0%)
Subdomains	0 / ∞
Aliases	0 / 50 (0%)
Number Of Processes	0 / 100 (0%)
Email Accounts	0 / ∞
Autoresponders	0 / ∞
Forwarders	0 / ∞
Email Filters	0 / ∞
FTP Accounts	0 / ∞
CPU Usage	0 / 100 (0%)
Entry Processes	0 / 20 (0%)
Physical Memory Usage	0 bytes / 1.5 GB (0%)
IOPS	0 / 1,024 (0%)
I/O Usage	0 bytes/s / 1 MB/s (0%)

Figure 5.8: Limitations of the subscribed service

Although there were lots of services provided under this package but there were also limitations on this subscribed service as shown in Figure 5.8. The total number of databases that can be created under this package was 50 only. The maximum file usage was 250,000 which meant only this amount of files can be stored on this server. On the other hand, disk usage was the total space occupied by all files and databases on the server. The MySQL disk usage however was total space occupied by databases only. Both of them had an upper limit of 50 gigabytes of free spaces. Then, the bandwidth usage had its maximum amount too which was 100 gigabytes. It was the amount of data transferred to and from the server. It was calculated and refreshed in every month. Aliases were additional domains that can be used to point to the same website and in this package only 50 of it was allowed. Number of processes were all processes created by the site but entry processes were the amount of processes accessing the site. The limits will be 100 and 20 respectively. Other than that, the maximum physical memory usage was 1.5 gigabyte. IOPS was stand for input/output operations per second which can measure the performance of the server to retrieve and store data into the database. The others services such as subdomains, email accounts, auto responders, forwarders, email filters and FTP accounts had no limitations.

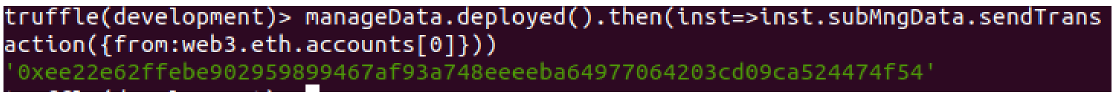
5.3 Testing

5.3.1 Test on Normal Usage



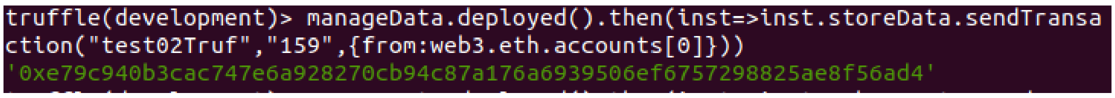
```
truffle(development)> manageData.deployed().then(inst=>inst.signUp.sendTransacti
on({from:web3.eth.accounts[0]}))
'0x6eb63f19b447af6fa107310d379d9112ef95490e3ad14bed9e746234a0a44b4a'
truffle(development)> web3.eth.accounts
[ '0x1ffc458fa9ecae36fd3d4aa62b8e6c124cd4832d' ]
```

Figure 5.9: Sign up



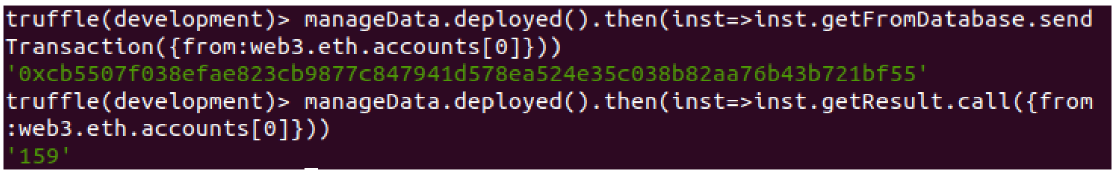
```
truffle(development)> manageData.deployed().then(inst=>inst.subMngData.sendTrans
action({from:web3.eth.accounts[0]}))
'0xee22e62ffebe902959899467af93a748eeeeba64977064203cd09ca524474f54'
```

Figure 5.10: Subscribe to manage data service



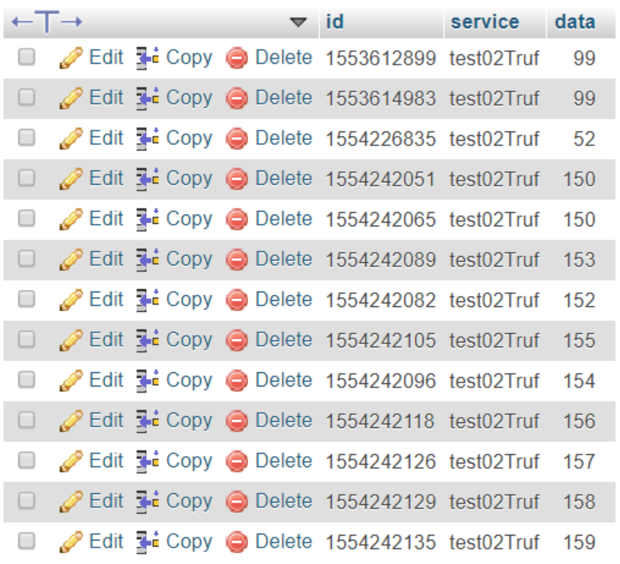
```
truffle(development)> manageData.deployed().then(inst=>inst.storeData.sendTransa
ction("test02Truf","159",{from:web3.eth.accounts[0]}))
'0xe79c940b3cac747e6a928270cb94c87a176a6939506ef6757298825ae8f56ad4'
```

Figure 5.11: Store data



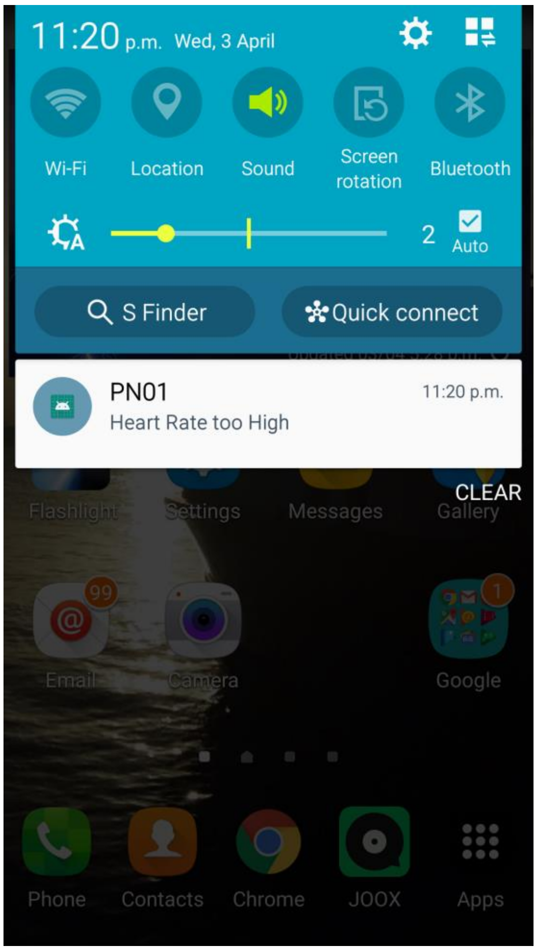
```
truffle(development)> manageData.deployed().then(inst=>inst.getFromDatabase.send
Transaction({from:web3.eth.accounts[0]}))
'0xcb5507f038efae823cb9877c847941d578ea524e35c038b82aa76b43b721bf55'
truffle(development)> manageData.deployed().then(inst=>inst.getResult.call({from
:web3.eth.accounts[0]}))
'159'
```

Figure 5.12: Retrieve data



←T→				id	service	data
☐	Edit	Copy	Delete	1553612899	test02Truf	99
☐	Edit	Copy	Delete	1553614983	test02Truf	99
☐	Edit	Copy	Delete	1554226835	test02Truf	52
☐	Edit	Copy	Delete	1554242051	test02Truf	150
☐	Edit	Copy	Delete	1554242065	test02Truf	150
☐	Edit	Copy	Delete	1554242089	test02Truf	153
☐	Edit	Copy	Delete	1554242082	test02Truf	152
☐	Edit	Copy	Delete	1554242105	test02Truf	155
☐	Edit	Copy	Delete	1554242096	test02Truf	154
☐	Edit	Copy	Delete	1554242118	test02Truf	156
☐	Edit	Copy	Delete	1554242126	test02Truf	157
☐	Edit	Copy	Delete	1554242129	test02Truf	158
☐	Edit	Copy	Delete	1554242135	test02Truf	159

Figure 5.13: Table in the cloud database

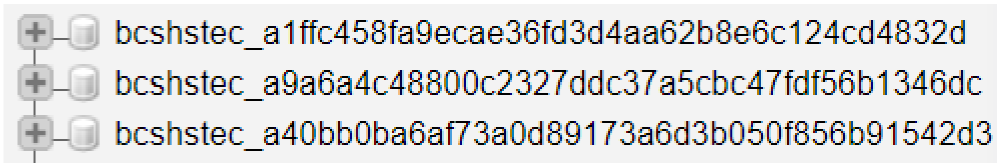


PN01 11:20 p.m.
Heart Rate too High

Figure 5.14: Notification received

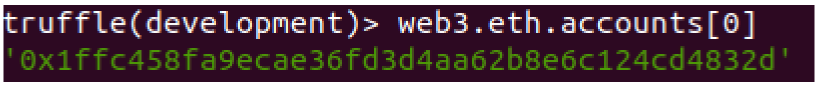
For the normal usage on this system, the first step will be signing up as a part of the SHS through the smart contract's function as in Figure 5.9. Then, users had to subscribe to the manage data service in order to divert data to and from the smart home to cloud server as shown in Figure 5.10. After that, the registered account can begin to store data into the cloud database as in Figure 5.11. The particular account can also retrieve the latest data from the cloud server as in Figure 5.12. For the push notification part, if the database table had exceeded ten rows of data, then it will evaluate whether the heart rate was too high or low. In Figure 5.13, the average of latest ten data was higher than the normal range of heart beat rate and the server sent notification to user's device as shown in Figure 5.14.

5.3.2 Accessing data with unregistered account



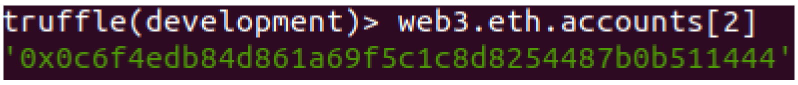
```
bcshstec_a1ffc458fa9ecae36fd3d4aa62b8e6c124cd4832d
bcshstec_a9a6a4c48800c2327ddc37a5cbc47fdf56b1346dc
bcshstec_a40bb0ba6af73a0d89173a6d3b050f856b91542d3
```

Figure 5.15: Existing databases in cloud



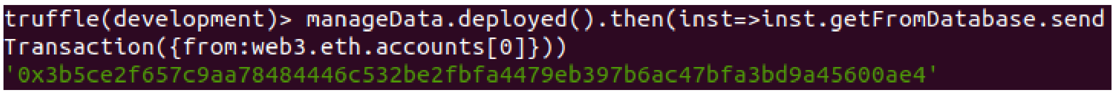
```
truffle(development)> web3.eth.accounts[0]
'0x1ffc458fa9ecae36fd3d4aa62b8e6c124cd4832d'
```

Figure 5.16: Authorized account



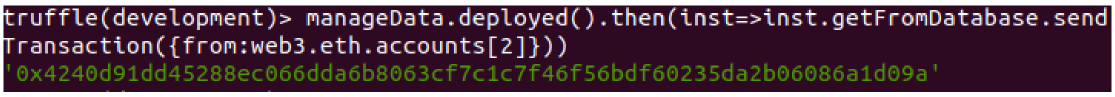
```
truffle(development)> web3.eth.accounts[2]
'0x0c6f4edb84d861a69f5c1c8d8254487b0b511444'
```

Figure 5.17: Unauthorized account



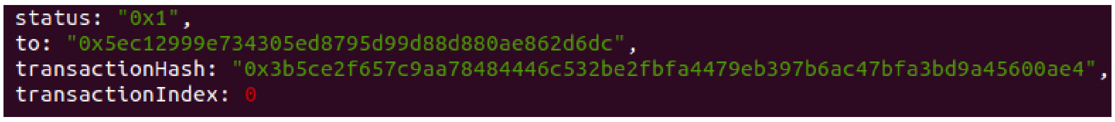
```
truffle(development)> manageData.deployed().then(inst=>inst.getFromDatabase.send
Transaction({from:web3.eth.accounts[0]}))
'0x3b5ce2f657c9aa78484446c532be2fbfa4479eb397b6ac47bfa3bd9a45600ae4'
```

Figure 5.18: Registered account requesting data



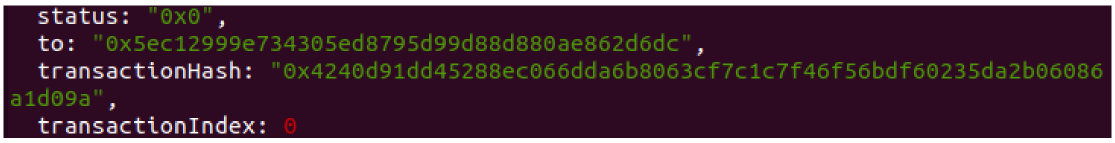
```
truffle(development)> manageData.deployed().then(inst=>inst.getFromDatabase.send
Transaction({from:web3.eth.accounts[2]}))
'0x4240d91dd45288ec066dda6b8063cf7c1c7f46f56bdf60235da2b06086a1d09a'
```

Figure 5.19: Unauthorized account requesting data



```
status: "0x1",
to: "0x5ec12999e734305ed8795d99d88d880ae862d6dc",
transactionHash: "0x3b5ce2f657c9aa78484446c532be2fbfa4479eb397b6ac47bfa3bd9a45600ae4",
transactionIndex: 0
```

Figure 5.20: Status of successful transaction



```
  status: "0x0",
  to: "0x5ec12999e734305ed8795d99d88d880ae862d6dc",
  transactionHash: "0x4240d91dd45288ec066dda6b8063cf7c1c7f46f56bdf60235da2b06086
a1d09a",
  transactionIndex: 0
```

Figure 5.21: Status of unsuccessful transaction

In this subsection, a test on accessing data from cloud server using an unregistered or unauthorized account was performed. From Figure 5.15, there was a list of existing database to show all of those registered account, including the one displayed on Figure 5.16. Moreover, Figure 5.17 was the unauthorized account created to perform this test. Then, both accounts tried to request data from the cloud server as shown in

Figure 5.18 and Figure 5.19. A command was entered to retrieve the transaction details by using the respective transaction hash generated.

Command to retrieve transaction details (in Geth console):

```
eth.getTransactionReceipt([Transaction Hash])
```

After the command was executed based on the transaction hash generated previously, the status of the transaction can be seen as in Figure 5.20 and Figure 5.21. If a transaction was executed successfully, the status of the transaction will be 0x1. Otherwise, the status will be 0x0 as an indication of failed transaction. However, if a transaction was still yet to be confirmed or mined, the command above will return null. Note that the transaction hash in Figure 5.20 was the one created in Figure 5.18 which was requested by the registered account and the transaction hash in Figure 5.21 was the one generated in Figure 5.19 which was requested by the unauthorized account.

Chapter 6 Conclusion

SHS which is closely related to one's daily life can bring lots of conveniences to an individual but in the same time it could also potentially leak personal information and private data too. The leaking of personal information may lead to serious, unrecoverable consequences and damage to an individual or even to a family. Although local SHS had been introduced as a sub-optimum solution however local SHS does not allows remote access of IoT and which greatly reduces the advantage of SHS. Cloud SHS with security issues needs to be solve or mitigate to an extend such that it can be deployed in a safe and secure environment.

In this project, BC was used as it is one of the decent solution in resolving the problem faced by cloud SHS because it can achieve several security aspects such as integrity, authenticity and auditability. Hence, BC-SHS was proposed. In BC-SHS there exist of a community of smart homes joined a consortium BC network. Each smart homes consisted a local gateway which could collect sensor data from IoT sensors nodes, communicate with cloud server through smart contract and perform consensus. The security became tighter as local SHS gateway was the only device that get to connect to the internet. Other than that, smart contract was also introduced to provide automation of checking and validating on users and data which further improves the overall security.

In the future, IoT devices or sensors can be implement into the existing system to complete the entire flow of the system as for now only self-generated or dummy data was used. The number of nodes in the BC environment can also be expand and increase which was not limited to the number of database provided by the service subscribed. The gateway device can also be replaced by the real intended device like raspberry pi as mentioned before to greatly enhance the computational power and overall performance. Other than that, multiple nodes can be set up, connect and link across several and different machines instead of all nodes in the same machine. In order to develop this system more towards the real world situation, the cost of deploying smart contracts should be calculated wisely so that it will not be too expensive to use a function. As for now users needed to install the application and open mobile data or connect to Wi-Fi in order to receive the notification. Subsequently, SMS can be implement instead so that users still can get the notification even without connection of internet. This is because most of the healthcare service users will be elderly people which has less knowledge to use a mobile phone. Moreover, the mobile application can be implemented with more functions such as send emergency messages or calls to the service provider other than just receiving notifications. Besides, a version of web application and also iOS version of mobile application can be implemented as well.

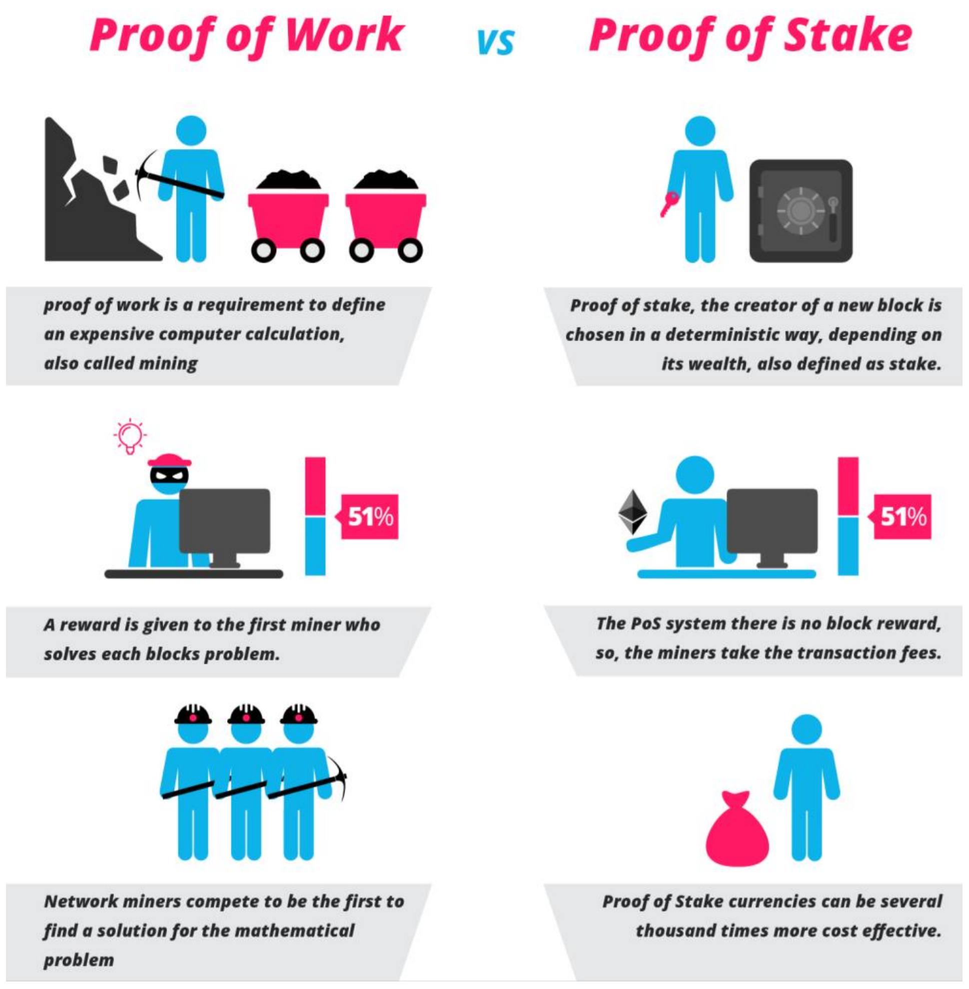

Proof of Work
vs
Proof of Stake
proof of work is a requirement to define an expensive computer calculation, also called mining
Proof of stake, the creator of a new block is chosen in a deterministic way, depending on its wealth, also defined as stake.
51%
51%
A reward is given to the first miner who solves each blocks problem.
The PoS system there is no block reward, so, the miners take the transaction fees.
Network miners compete to be the first to find a solution for the mathematical problem
Proof of Stake currencies can be several thousand times more cost effective.

Figure 6.1: (PoW vs PoS 2017)

Additionally, a more suitable and satisfying mining algorithm such as Proof-of-Stake (PoS) can be applied afterwards. There were many differences between PoW and PoS. Unlike PoW, which a new block will be added by the miner who solves the mathematical puzzle, in PoS, blocks were added into the chain by determining its wealth or known as stake. The way of proving a valid block does not depend on the computational power or how much work had been done anymore instead it depends on how much stake was placed. There will be no block rewards and miners (known as validator) will get the transaction fees in PoS system.

One of the challenges faced when developing this project was learning the contract-oriented language, Solidity. Although this language was influenced by C++, Python and JavaScript but the nature of this language was not like other programming languages and it was confusing in the beginning. Besides that, EVM is isolated from the outside world and some work need to be done so that it can communicate with sources and data from outside. Oraclize service was used to aid in this situation however it needed to be studied carefully in order to implement it successfully. The other challenge will be deploying all those APIs up to the purchased web hosting server. All those configurations needed to be set again and it was very different from the local apache server. In order to communicate from the server to the mobile application, firebase needed to be learnt and set up. The journey of learning FCM and how it worked consumes lots of time as there were many documents regarding this function. Furthermore, Android Studio was not an easy platform for newbies to create a mobile application.

In conclusion, SHS became more reliable and trustworthy after applied with BC. This project can be used as reference for industry that may want to adopt BC as their solution. Nowadays, many existing SHS does not consider cyber security as one of the important aspect and this project could be a light in dim for them.

References

A. Dorri, S. S. Kanhere, and R. Jurdak 2016, ‘*Blockchain in internet of things: Challenges and solutions*’, Cornell University Library. Available from: < https://arxiv.org/ftp/arxiv/papers/1608/1608.05187.pdf>. [1 August 2018].

A. Dorri, S. S. Kanhere, R. Jurdak and P. Gauravaram 2017, ‘Blockchain for IoT security and privacy: The case study of a smart home’, *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops),* pp. 618-623. Available from: IEEE Xplore Digital Library. [1 August 2018].

Abdur, M., Habib, S., Ali, M. and Ullah, S. 2017, ‘Security Issues in the Internet of Things (IoT): A Comprehensive Study’, *International Journal of Advanced Computer Science and Applications,* vol. 8, no. 6, pp.385-386.

Ashton, K. 2009. ‘*That 'Internet of Things' Thing*’, RFID Journal. Available from: <http://www.rfidjournal.com/articles/pdf?4986>. [5 Jul. 2018].

Christidis, K. and Devetsikiotis, M. 2016, ‘Blockchains and Smart Contracts for the Internet of Things’, *IEEE Access,* vol. 4, pp.2292-2303. Available from: IEEE Xplore Digital Library. [19 July 2018].

Cloud deployment model, 2017. Available from: <https://qph.ec.quoracdn.net/main-qimg-34fbe07d9311be4084f7b7b8a502a307>. [8 August 2018].

Dinh, T., Liu, R., Zhang, M., Chen, G., Ooi, B. and Wang, J. 2018, ‘Untangling Blockchain: A Data Processing View of Blockchain Systems’, *IEEE Transactions on Knowledge and Data Engineering,* vol. 30, no. 7, pp.1366-1385.

F. K. Santoso and N. C. H. Vun 2015, ‘Securing IoT for smart home system’, *International Symposium on Consumer Electronics (ISCE),* pp. 1-2. Available from: IEEE Xplore Digital Library. [27 July 2018].

Hybrid cloud, 2016. Available from: <https://itviconsultants.com/wp-content/uploads/2016/10/hybrid_cloud_graphic1.png>. [8 August 2018].

Kusek, M 2018, ‘Internet of Things: Today and tomorrow’, *International Convention on Information and Communication Technology, Electronics and Microelectronics*

(MIPRO), pp. 0335-0338. Available from: IEEE Xplore Digital Library. [15 July 2018].

M. Amara and A. Siad 2011, 'Elliptic Curve Cryptography and its applications', *International Workshop on Systems, Signal Processing and their Applications, WOSSPA*, pp. 247-250. Available from: IEEE Xplore Digital Library. [27 July 2018].

O. Jukić, I. Špeh and I. Heđi 2018, 'Cloud-based services for the Internet of Things', *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 0372-0377. Available from: IEEE Xplore Digital Library. [15 July 2018].

PoW vs PoS, 2017. Available from: <https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/>. [8 April 2019].

R. Tamada 2017, Android Push Notifications using Firebase Cloud Messaging FCM & PHP. Available from: < https://www.androidhive.info/2012/10/android-push-notifications-using-google-cloud-messaging-gcm-php-and-mysql/>. [7 April 2019].

S, Nakamoto 2008, '*Bitcoin: A peer-to-peer electronic cash system*', Bitcoin Project. Available from: <https://bitcoin.org/bitcoin.pdf>. [5 Jul. 2018].

S. Huh, S. Cho and S. Kim 2017, 'Managing IoT devices using blockchain platform', *International Conference on Advanced Communication Technology (ICACT),* pp. 464-467. Available from: IEEE Xplore Digital Library. [8 August 2018].

S. S. Chapade, K. U. Pandey and D. S. Bhade 2013, 'Securing Cloud Servers Against Flooding Based DDOS Attacks', *International Conference on Communication Systems and Network Technologies,* pp. 524-528. Available from: IEEE Xplore Digital Library. [27 July 2018].

S. Sridhar and S. Smys 2017, 'Intelligent security framework for iot devices cryptography based end-to-end security architecture' *International Conference on Inventive Systems and Control (ICISC)*, pp. 1-5. Available from: IEEE Xplore Digital Library. [8 August 2018].

V. D. Sharma, S. Agarwai, S. S. Moin and M. A. Qadeer 2017, 'Security in cloud computing', *International Conference on Communication Systems and Network*

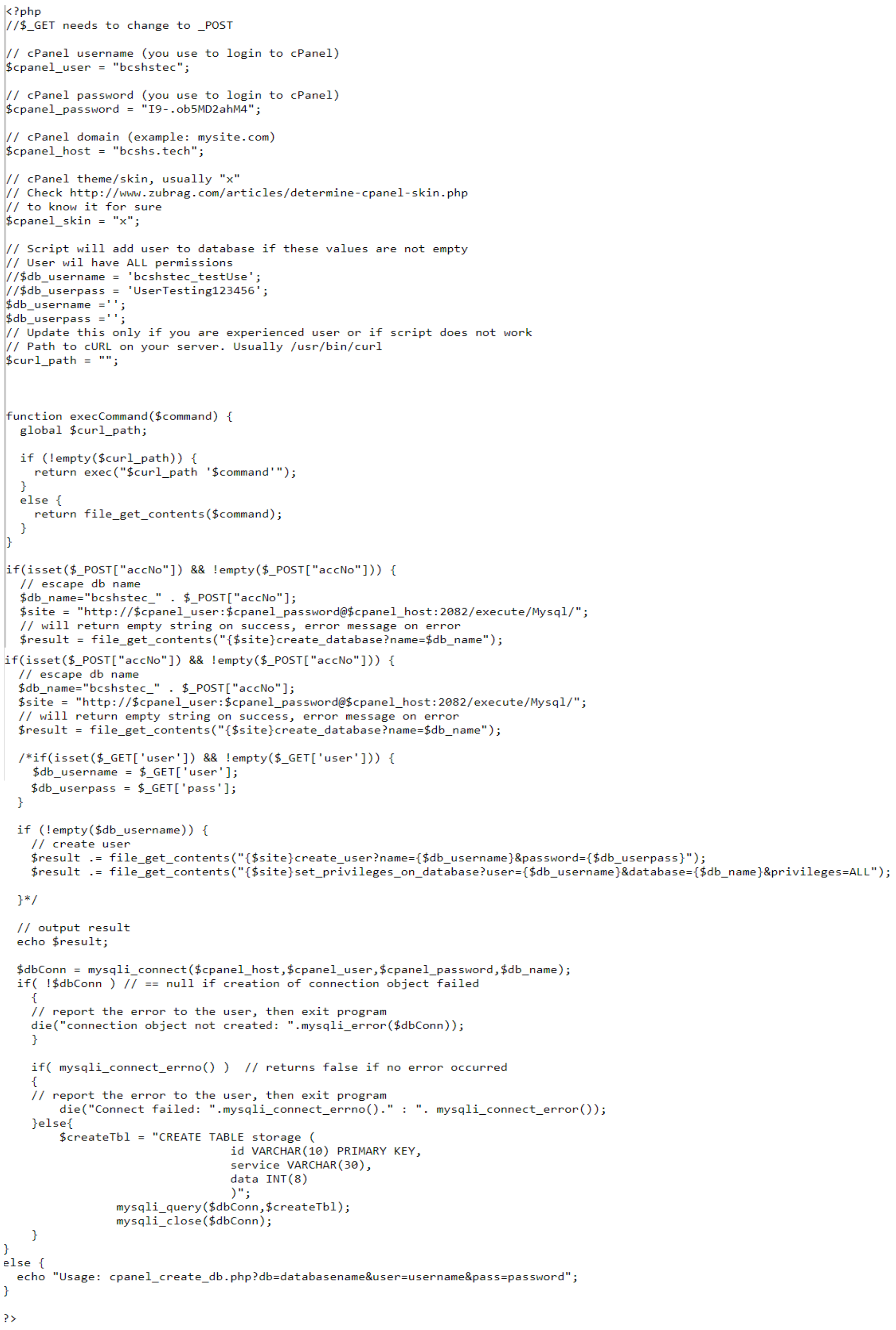
Technologies (CSNT), pp. 234-239. Available from: IEEE Xplore Digital Library. [8 August 2018].

Wang, W., Xu, P. and Yang, L. 2018, 'Secure Data Collection, Storage and Access in Cloud-Assisted IoT', *IEEE Cloud Computing*, pp.1-1. Available from: IEEE Xplore Digital Library. [1 August 2018].

Y. N. Aung and T. Tantidham 2017, 'Review of Ethereum: Smart home case study', *International Conference on Information Technology (INCIT)*, pp. 1-4. Available from: IEEE Xplore Digital Library. [1 August 2018].

APPENDIX A: API

A.1 Registration API

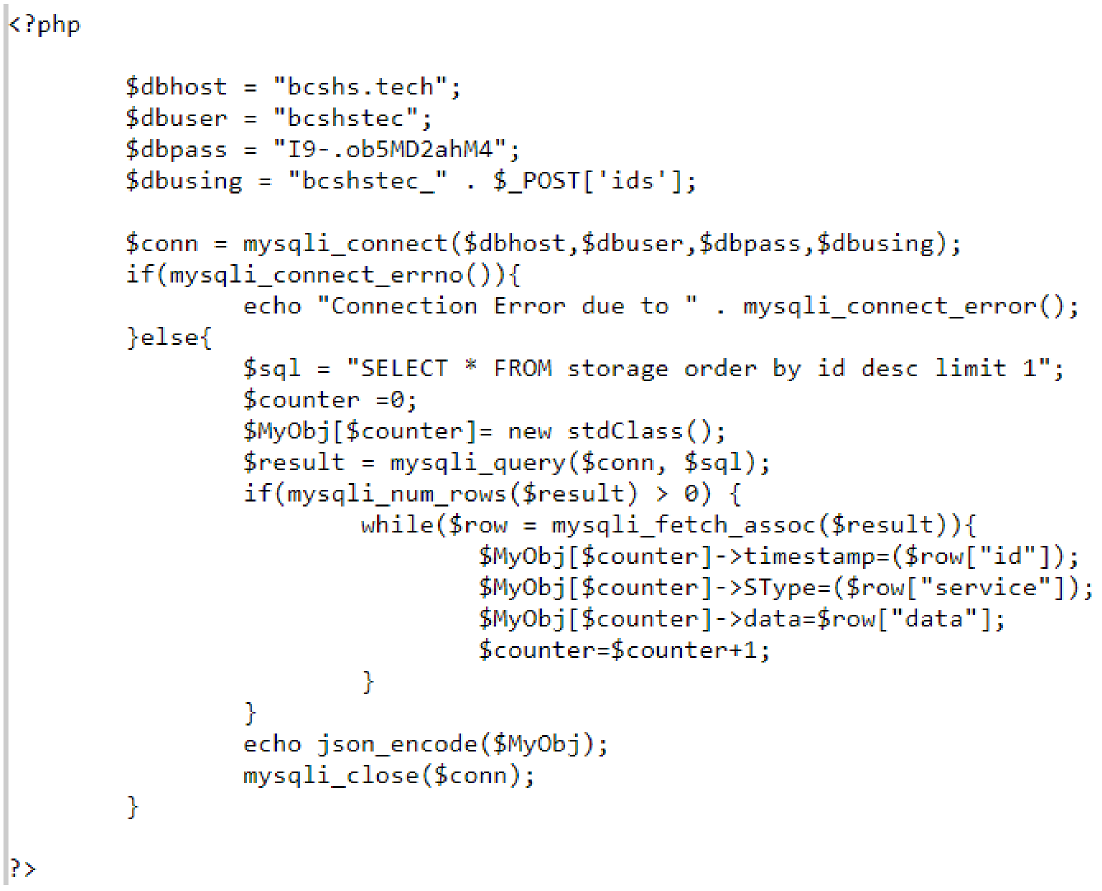


```
<?php
//$_GET needs to change to _POST

// cPanel username (you use to login to cPanel)
$cpanel_user = "bcshstec";

// cPanel password (you use to login to cPanel)
$cpanel_password = "I9-.ob5MD2ahM4";

// cPanel domain (example: mysite.com)
$cpanel_host = "bcshs.tech";

// cPanel theme/skin, usually "x"
// Check http://www.zubrag.com/articles/determine-cpanel-skin.php
// to know it for sure
$cpanel_skin = "x";

// Script will add user to database if these values are not empty
// User wil have ALL permissions
//$db_username = 'bcshstec_testUse';
//$db_userpass = 'UserTesting123456';
$db_username ='';
$db_userpass ='';
// Update this only if you are experienced user or if script does not work
// Path to cURL on your server. Usually /usr/bin/curl
$curl_path = "";


function execCommand($command) {
  global $curl_path;

  if (!empty($curl_path)) {
    return exec("$curl_path '$command'");
  }
  else {
    return file_get_contents($command);
  }
}

if(isset($_POST["accNo"]) && !empty($_POST["accNo"])) {
  // escape db name
  $db_name="bcshstec_" . $_POST["accNo"];
  $site = "http://$cpanel_user:$cpanel_password@$cpanel_host:2082/execute/Mysql/";
  // will return empty string on success, error message on error
  $result = file_get_contents("{$site}create_database?name=$db_name");
if(isset($_POST["accNo"]) && !empty($_POST["accNo"])) {
  // escape db name
  $db_name="bcshstec_" . $_POST["accNo"];
  $site = "http://$cpanel_user:$cpanel_password@$cpanel_host:2082/execute/Mysql/";
  // will return empty string on success, error message on error
  $result = file_get_contents("{$site}create_database?name=$db_name");

  /*if(isset($_GET['user']) && !empty($_GET['user'])) {
    $db_username = $_GET['user'];
    $db_userpass = $_GET['pass'];
  }

  if (!empty($db_username)) {
    // create user
    $result .= file_get_contents("{$site}create_user?name={$db_username}&password={$db_userpass}");
    $result .= file_get_contents("{$site}set_privileges_on_database?user={$db_username}&database={$db_name}&privileges=ALL");

  }*/

  // output result
  echo $result;

  $dbConn = mysqli_connect($cpanel_host,$cpanel_user,$cpanel_password,$db_name);
  if( !$dbConn ) // == null if creation of connection object failed
    {
    // report the error to the user, then exit program
    die("connection object not created: ".mysqli_error($dbConn));
    }

    if( mysqli_connect_errno() )  // returns false if no error occurred
    {
    // report the error to the user, then exit program
        die("Connect failed: ".mysqli_connect_errno()." : ". mysqli_connect_error());
    }else{
        $createTbl = "CREATE TABLE storage (
                                id VARCHAR(10) PRIMARY KEY,
                                service VARCHAR(30),
                                data INT(8)
                                )";
                mysqli_query($dbConn,$createTbl);
                mysqli_close($dbConn);
    }
}
else {
  echo "Usage: cpanel_create_db.php?db=databasename&user=username&pass=password";
}

?>
```

A.2 Store data API

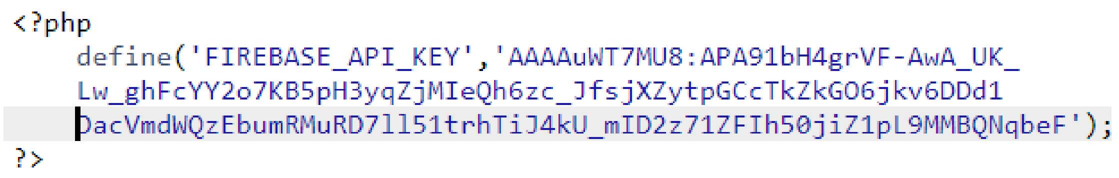
```
<?php
    $dbhost = "bcshs.tech";
        $dbuser = "bcshstec";
        $dbpass = "I9-.ob5MD2ahM4";
        $dbusing = "bcshstec_" . $_POST["accNo"];

        $conn = mysqli_connect($dbhost,$dbuser,$dbpass,$dbusing);
        $flag=false;
        if(mysqli_connect_errno()){
                echo "Connection Error due to " . mysqli_connect_error();
        }else{
                $sql = "INSERT INTO storage VALUES(?,?,?)";
                $stmt=mysqli_prepare($conn,$sql);
                mysqli_stmt_bind_param($stmt,'ssi',$_POST["timestamp"],$_POST["SType"],$_POST["Data"]);
                mysqli_stmt_execute($stmt);
                mysqli_stmt_close($stmt);


                $sql = "SELECT data from storage order by id desc limit 10";
                $result = mysqli_query($conn, $sql);
                $MyObj= new stdClass();
                if(mysqli_num_rows($result) >= 10) {

                        $sum=0;
                        $flag=true;
                        while ($row=mysqli_fetch_row($result)){
                            $sum =$sum+$row[0];
                        }
                        $avg=$sum/10;


                        if($avg>60 && $avg<101){
                                $MyObj->alertMsg="Good condition";
                                $title= 'Good condition';
                        }else if ($avg<60){
                                $MyObj->alertMsg="Heart Rate too Low";
                                $title= 'Heart Rate too Low';
                        }
                        else{
                                $MyObj->alertMsg="Heart Rate too High";
                                $title= 'Heart Rate too High';
                        }
                }
                echo json_encode($MyObj);
                mysqli_close($conn);
        }
       if($flag){
       require_once 'firebase/firebase.php';
       require_once 'firebase/push.php';
       $firebase = new Firebase();
       $push = new Push();

       // notification message
       $message = 'Hello World from PHP';

       // push type - single user / topic
       $push_type = 'individual';

       // whether to include to image or not
       $include_image = FALSE;

       $push->setTitle($title);
       $push->setMessage($message);
       if ($include_image) {
           $push->setImage('');
       } else {
           $push->setImage('');
       }
       $push->setIsBackground(FALSE);
       //$push->setPayload($payload);


       $json = '';
       $response = '';

       if ($push_type == 'topic') {
               $json = $push->getPush();
               $response = $firebase->sendToTopic('global', $json);
           } else if ($push_type == 'individual') {
               $json = $push->getPush();
               $regId = 'fBYLKQi2SWo:APA91bH87plvPlyveUpVbem1raibmXstF1HiFAQp6z7_-dR
                       DhFAq48wggRxLbqpPUbci_tcBVAratYHNa_KRhepd9RRGFWK5qh5owzOVRqPHv
                       tbchsdcrJzbIIcGiG0c6XPMrfBnRjX3';
               $response = $firebase->send($regId, $json);
           }
       }
>
```

A.3 Fetch data API



```
<?php

        $dbhost = "bcshs.tech";
        $dbuser = "bcshstec";
        $dbpass = "I9-.ob5MD2ahM4";
        $dbusing = "bcshstec_" . $_POST['ids'];

        $conn = mysqli_connect($dbhost,$dbuser,$dbpass,$dbusing);
        if(mysqli_connect_errno()){
                echo "Connection Error due to " . mysqli_connect_error();
        }else{
                $sql = "SELECT * FROM storage order by id desc limit 1";
                $counter =0;
                $MyObj[$counter]= new stdClass();
                $result = mysqli_query($conn, $sql);
                if(mysqli_num_rows($result) > 0) {
                        while($row = mysqli_fetch_assoc($result)){
                                $MyObj[$counter]->timestamp=($row["id"]);
                                $MyObj[$counter]->SType=($row["service"]);
                                $MyObj[$counter]->data=$row["data"];
                                $counter=$counter+1;
                        }
                }
                echo json_encode($MyObj);
                mysqli_close($conn);
        }

?>
```

A.4 Firebase Configuration API

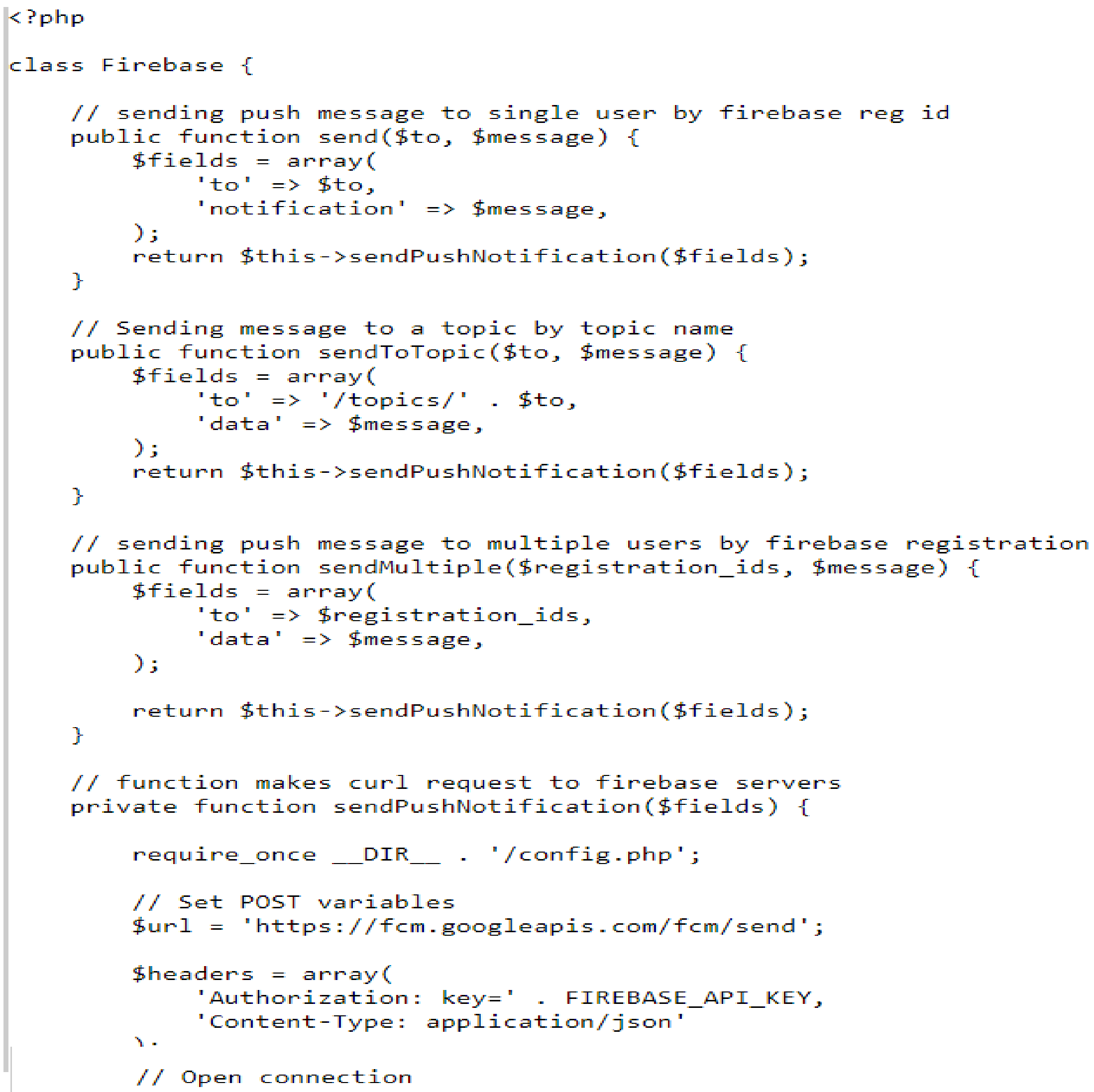


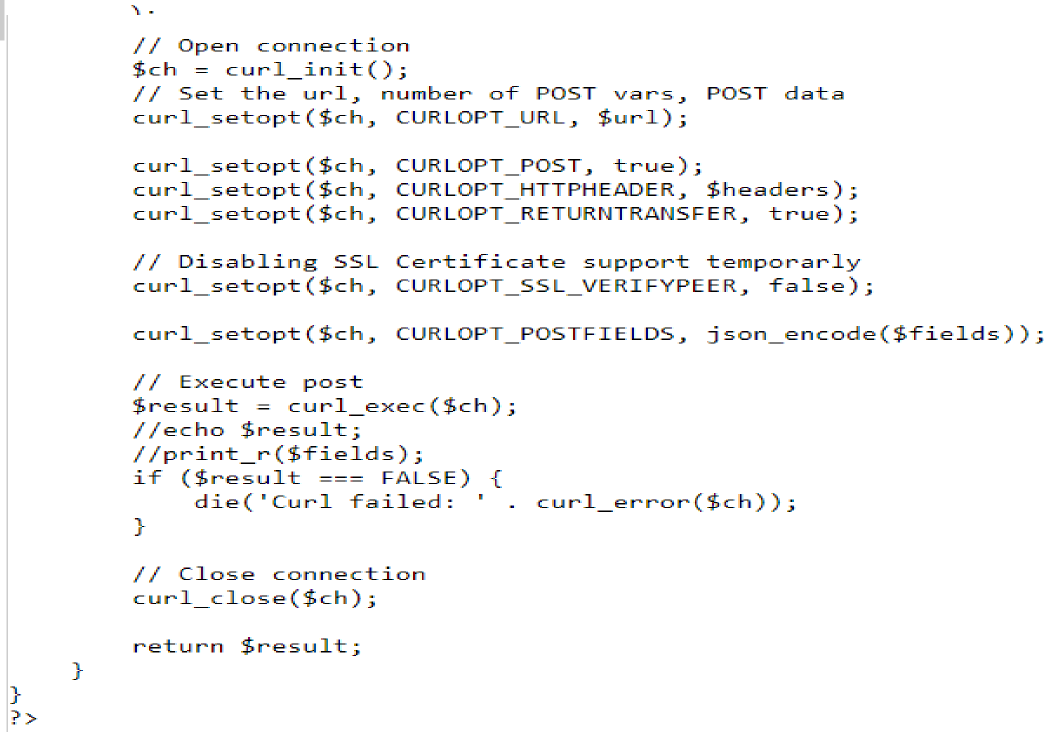
```
<?php
    define('FIREBASE_API_KEY','AAAAuWT7MU8:APA91bH4grVF-AwA_UK_
    Lw_ghFcYY2o7KB5pH3yqZjMIeQh6zc_JfsjXZytpGCcTkZkGO6jkv6DDd1
    DacVmdWQzEbumRMuRD7ll51trhTiJ4kU_mID2z71ZFIh50jiZ1pL9MMBQNqbeF');
?>
```

A.5 Firebase Set Elements Class API

```php
<?php


class Push {

    // push message title
    private $title;
    private $message;
    private $image;
    // push message payload
    private $data;
    // flag indicating whether to show the push
    // notification or not
    // this flag will be useful when perform some opertation
    // in background when push is recevied
    private $is_background;

    function __construct() {

    }

    public function setTitle($title) {
        $this->title = $title;
    }

    public function setMessage($message) {
        $this->message = $message;
    }

    public function setImage($imageUrl) {
        $this->image = $imageUrl;
    }

    public function setPayload($data) {
        $this->data = $data;
    }

    public function setIsBackground($is_background) {
        $this->is_background = $is_background;
    }
 public function getPush() {
     $res = array();
     $res['body'] = $this->title;
     //$res['data']['is_background'] = $this->is_background;
     //$res['notification']['body'] = $this->message;
     //$res['data']['image'] = $this->image;
     //$res['data']['payload'] = $this->data;
     //$res['data']['timestamp'] = date('Y-m-d G:i:s');
     return $res;
 }
```

A.6 Firebase Push Notification API





```
<?php

class Firebase {

    // sending push message to single user by firebase reg id
    public function send($to, $message) {
        $fields = array(
            'to' => $to,
            'notification' => $message,
        );
        return $this->sendPushNotification($fields);
    }

    // Sending message to a topic by topic name
    public function sendToTopic($to, $message) {
        $fields = array(
            'to' => '/topics/' . $to,
            'data' => $message,
        );
        return $this->sendPushNotification($fields);
    }

    // sending push message to multiple users by firebase registration
    public function sendMultiple($registration_ids, $message) {
        $fields = array(
            'to' => $registration_ids,
            'data' => $message,
        );

        return $this->sendPushNotification($fields);
    }

    // function makes curl request to firebase servers
    private function sendPushNotification($fields) {

        require_once __DIR__ . '/config.php';

        // Set POST variables
        $url = 'https://fcm.googleapis.com/fcm/send';

        $headers = array(
            'Authorization: key=' . FIREBASE_API_KEY,
            'Content-Type: application/json'
        );
        // Open connection
        $ch = curl_init();
        // Set the url, number of POST vars, POST data
        curl_setopt($ch, CURLOPT_URL, $url);

        curl_setopt($ch, CURLOPT_POST, true);
        curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

        // Disabling SSL Certificate support temporarly
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

        curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));

        // Execute post
        $result = curl_exec($ch);
        //echo $result;
        //print_r($fields);
        if ($result === FALSE) {
            die('Curl failed: ' . curl_error($ch));
        }

        // Close connection
        curl_close($ch);

        return $result;
    }
}
?>
```

APPENDIX B: Smart Contracts

B.1 contract 'Registration'

```
pragma solidity ^0.4.24;
import "installed_contracts/oraclize-api/contracts/usingOraclize.sol";


contract register is usingOraclize {
   constructor ()
   public {
          OAR =
OraclizeAddrResolverI(0xd6dDfA20dAB0F82ff94e16ed866a8Dd72a67cAe9);
   }
   function __callback(bytes32 id, string _result)
   public {
          require(msg.sender == oraclize_cbAddress());
       }
       function toAsciiString(address x) returns (string) {
          bytes memory s = new bytes(40);
          for (uint i = 0; i < 20; i++) {
                    byte b = byte(uint8(uint(x) / (2**(8*(19 - i)))));
                    byte hi = byte(uint8(b) / 16);
                    byte lo = byte(uint8(b) - 16 * uint8(hi));
                    s[2*i] = char(hi);
                    s[2*i+1] = char(lo);
          }
          return string(s);
          }


          function char(byte b) returns (byte c) {
          if (b < 10) return byte(uint8(b) + 0x30);
           else return byte(uint8(b) + 0x57);
          }
function registration(address _userAddr) public payable{
    string memory str = toAsciiString(_userAddr);


    string memory str1 = strConcat('{', '"accNo":' , '"' ,'a',str);
    string memory str2 = strConcat('"','}');
    string memory str3 = strConcat(str1,str2);
    oraclize_query("URL","https://bcshs.tech/registration.php",str3);
    }


}
```

B.2 contract ‘UserConfig’

```
pragma solidity ^0.4.24;
import "installed_contracts/oraclize-api/contracts/usingOraclize.sol";
import "./register.sol";

contract UserConfig{
  struct User {
        bool manageDataSub;
        bool registered;
        //bool HRservice;
        string resultData;
        //string HRmsg;
  }

  address public owner;
  mapping(address=>User) users;
     constructor ()  payable public {}
     function createUser(address _userAddr) public{
        User storage user = users[_userAddr];
        // Check that the user did not already exist:
        require(!user.registered);
        //Store the user
        users[msg.sender] = User({
          manageDataSub: false,
          registered: true,
          //HRservice:false,
          resultData: ""
          //HRmsg:""
         });
        register reg= new register();
        reg.registration(_userAddr);
        }
function subManageData(address _userAddress) public {
    users[_userAddress].manageDataSub = true;
    }

    function unSubManageData(address _userAddress) public{
    users[_userAddress].manageDataSub = false;
    }
    function getManageSub(address _userAddress) public returns(bool){
    return users[_userAddress].manageDataSub;
    }
    function setResultData(address _userAddress, string data) public{
    users[_userAddress].resultData = data;
    }
    function getResultData(address _userAddress) public returns(string){
    return users[_userAddress].resultData;
    }
}
```

B.3 contract ‘ManageData’

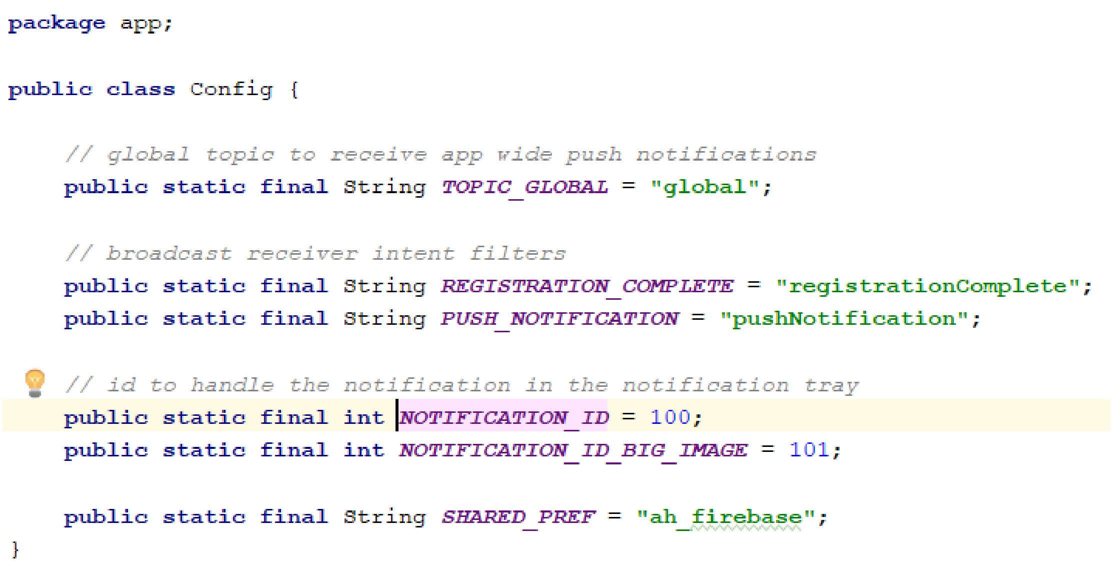
```
pragma solidity ^0.4.24;


import "installed_contracts/oraclize-api/contracts/usingOraclize.sol";
import "./register.sol";
import "./UserConfig.sol";


contract manageData is usingOraclize,UserConfig,register {
   enum oraclizeState { ToGet, AltMsg}
   struct oraclizeCallback{ oraclizeState oState;}
   mapping (bytes32=> oraclizeCallback) public oraclizeCallbacks;
   mapping (bytes32=> address) public callbackSenders;


   constructor ()
   public payable {


    OAR =
OraclizeAddrResolverI(0xd6dDfA20dAB0F82ff94e16ed866a8Dd72a67cAe9);


    }
function signUp()public{
    UserConfig.createUser(msg.sender);
    }
   function subMngData()public{
    UserConfig.subManageData(msg.sender);
    }
function __callback(bytes32 id, string _result) public {
    require(msg.sender == oraclize_cbAddress());
    oraclizeCallback memory o= oraclizeCallbacks[id];
    if(o.oState == oraclizeState.ToGet){
            UserConfig.setResultData(callbackSenders[id],_result);
    }
}
```

```
        function storeData(string serviceType, string data) public payable{
           if(UserConfig.getManageSub(msg.sender)){


                    string memory str = register.toAsciiString(msg.sender);
                    string memory str1 = strConcat('{','"timestamp":','"',uint2str(now),'"');
                    string memory str2 = strConcat(',','"SType":');
                    string memory str3 = strConcat('"',serviceType,'"',',','"Data":');
                    string memory str4 = strConcat('"',data,'"',',','"accNo":' );
                    string memory str5 = strConcat('"' ,'a',str,'"','}');
                    string memory str6 = strConcat(str1,str2,str3,str4,str5);


                    oraclize_query("URL","https://bcshs.tech/storeData.php" ,str6);


           }
          }
        function getFromDatabase() public payable{
           if(UserConfig.getManageSub(msg.sender)){


                    string memory str = register.toAsciiString(msg.sender);
                    string memory str1 = strConcat('{','"ids":','"a',str,'"}');



                    bytes32 queryId = oraclize_query("URL",
 "json(https://bcshs.tech/fetchData.php).0.data",str1);
                    oraclizeCallbacks[queryId] = oraclizeCallback(oraclizeState.ToGet);
                    callbackSenders[queryId] = msg.sender;
           }
           }
```

```
function getResult() public returns (string){


    return UserConfig.getResultData(msg.sender);


    }
    function ()external payable{
    }
    function getContBalance() public returns(uint){
    return address(this).balance;
    }
}
```

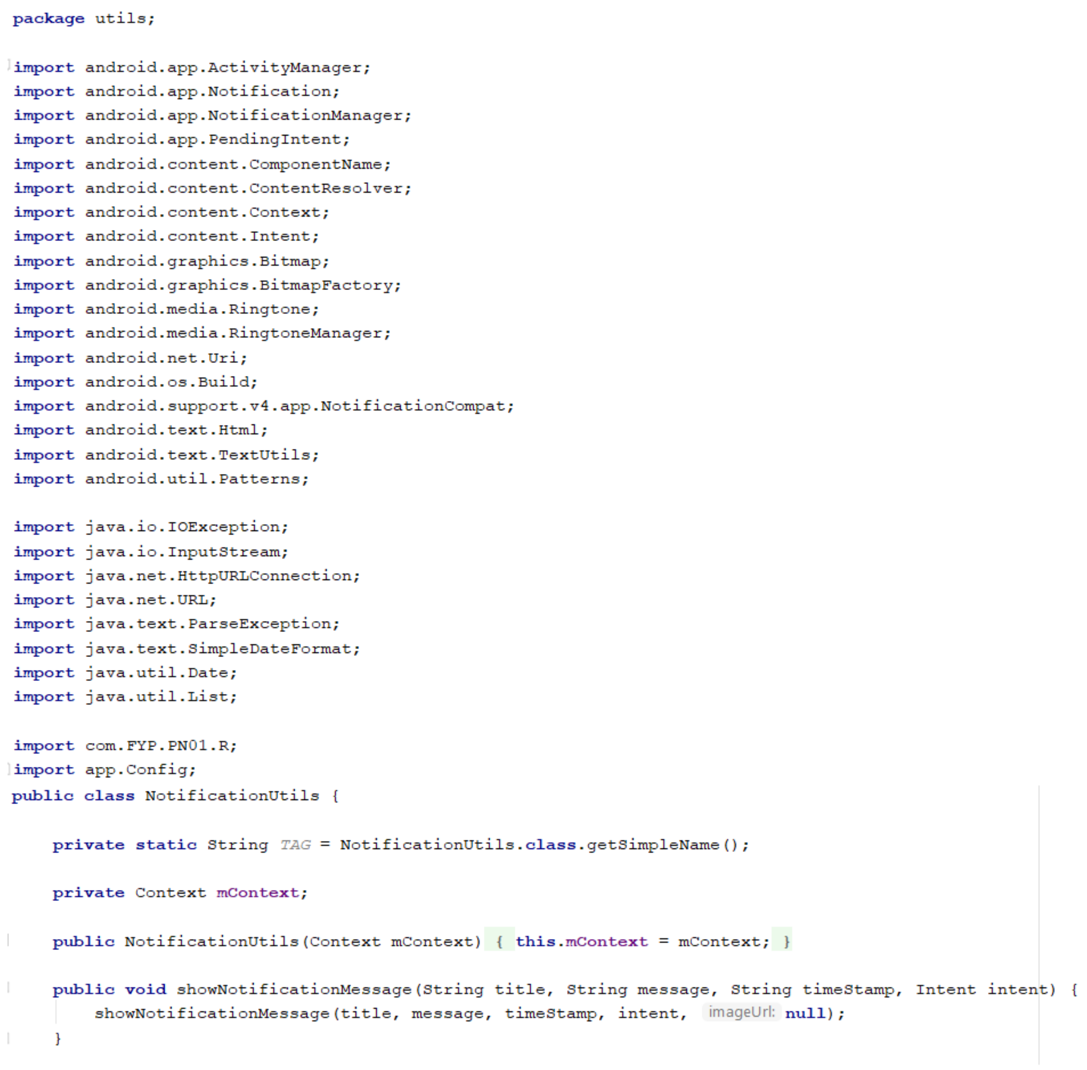
APPENDIX C: Mobile Application files

C.1 Config.java

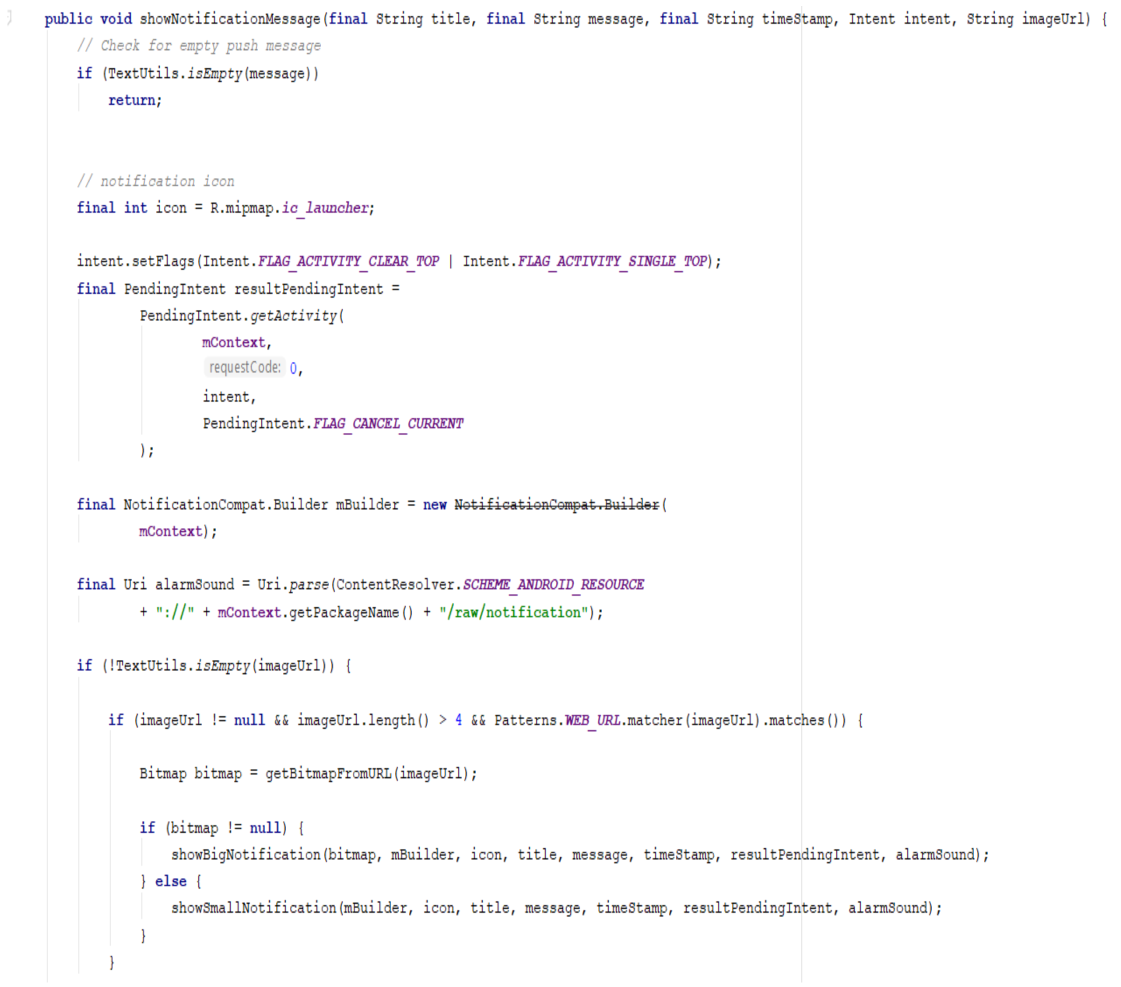


```java
package app;

public class Config {

    // global topic to receive app wide push notifications
    public static final String TOPIC_GLOBAL = "global";

    // broadcast receiver intent filters
    public static final String REGISTRATION_COMPLETE = "registrationComplete";
    public static final String PUSH_NOTIFICATION = "pushNotification";

    // id to handle the notification in the notification tray
    public static final int NOTIFICATION_ID = 100;
    public static final int NOTIFICATION_ID_BIG_IMAGE = 101;

    public static final String SHARED_PREF = "ah_firebase";
}
```

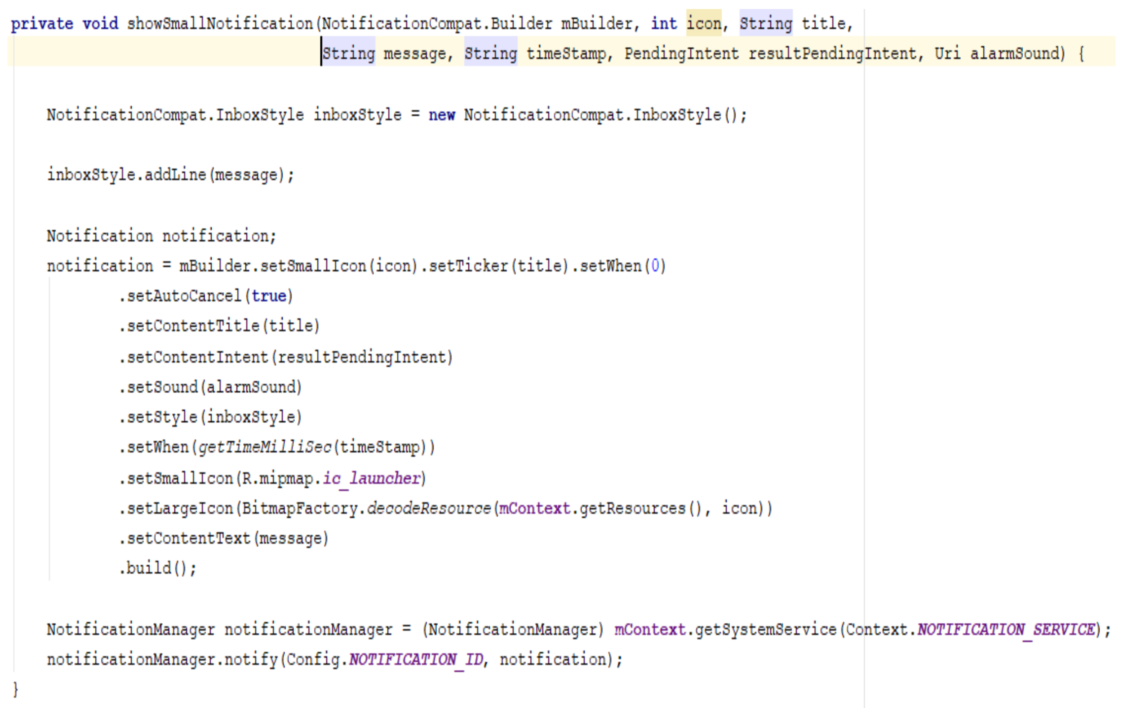
C.2 NotificationUtils.java



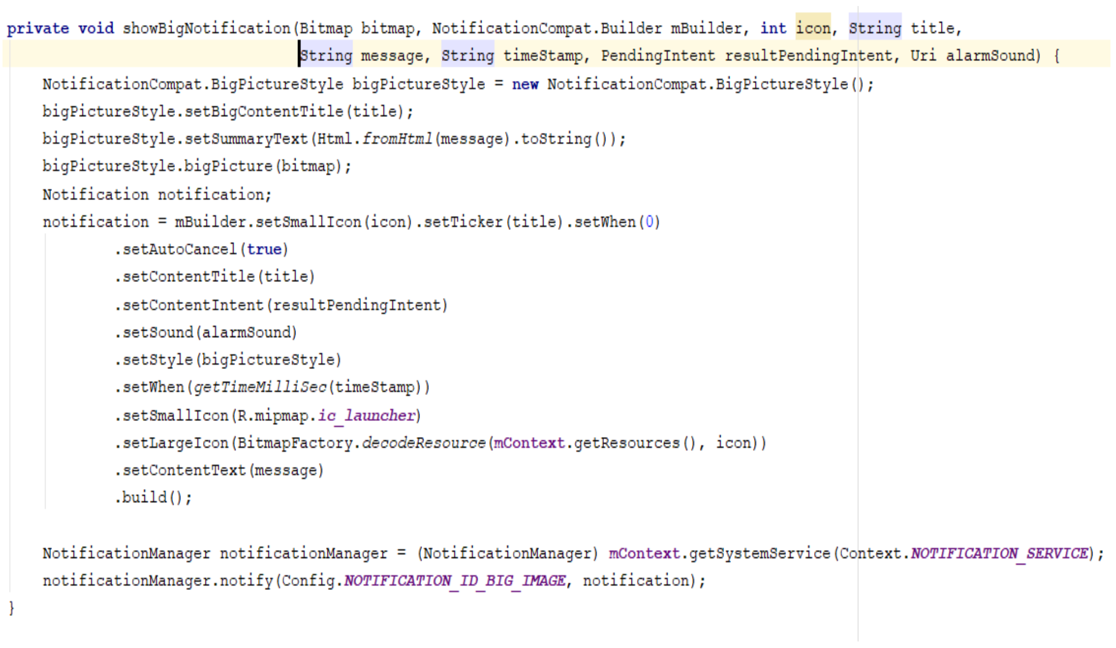
```java
package utils;

import android.app.ActivityManager;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.ComponentName;
import android.content.ContentResolver;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.media.Ringtone;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Build;
import android.support.v4.app.NotificationCompat;
import android.text.Html;
import android.text.TextUtils;
import android.util.Patterns;

import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import com.FYP.PN01.R;
import app.Config;
public class NotificationUtils {

    private static String TAG = NotificationUtils.class.getSimpleName();

    private Context mContext;

    public NotificationUtils(Context mContext) { this.mContext = mContext; }

    public void showNotificationMessage(String title, String message, String timeStamp, Intent intent) {
        showNotificationMessage(title, message, timeStamp, intent,  imageUrl: null);
    }
```



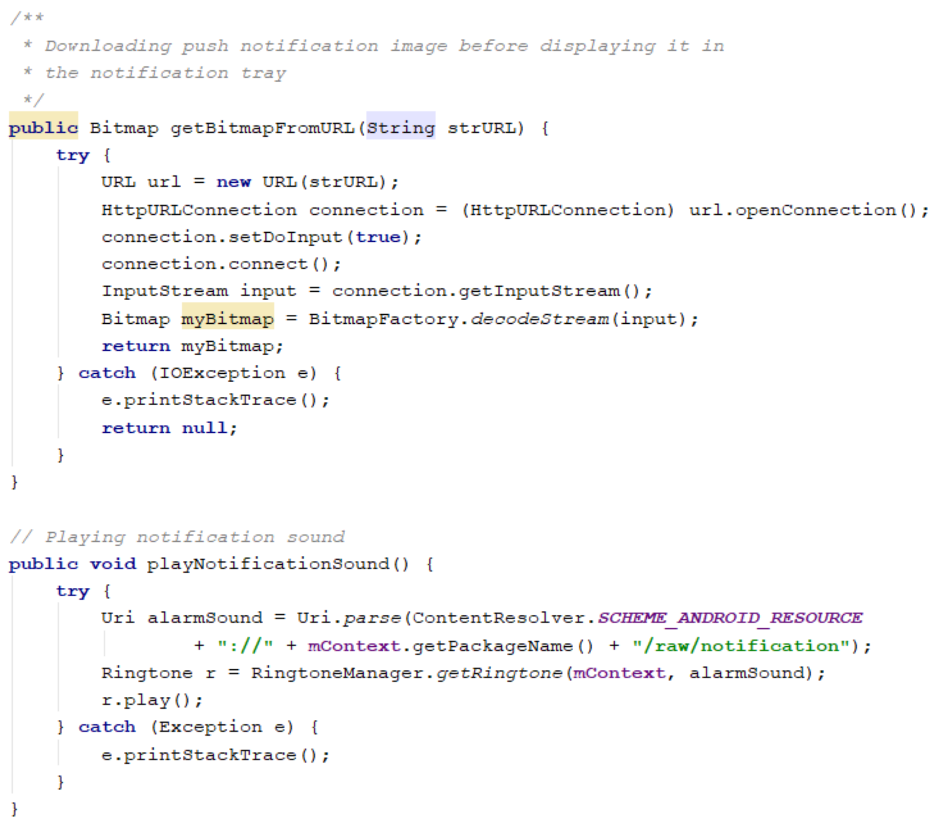
```
public void showNotificationMessage(final String title, final String message, final String timeStamp, Intent intent, String imageUrl) {
    // Check for empty push message
    if (TextUtils.isEmpty(message))
        return;


    // notification icon
    final int icon = R.mipmap.ic_launcher;

    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_SINGLE_TOP);
    final PendingIntent resultPendingIntent =
            PendingIntent.getActivity(
                    mContext,
                    requestCode: 0,
                    intent,
                    PendingIntent.FLAG_CANCEL_CURRENT
            );

    final NotificationCompat.Builder mBuilder = new ~~NotificationCompat.Builder~~(
            mContext);

    final Uri alarmSound = Uri.parse(ContentResolver.SCHEME_ANDROID_RESOURCE
            + "://" + mContext.getPackageName() + "/raw/notification");

    if (!TextUtils.isEmpty(imageUrl)) {

        if (imageUrl != null && imageUrl.length() > 4 && Patterns.WEB_URL.matcher(imageUrl).matches()) {

            Bitmap bitmap = getBitmapFromURL(imageUrl);

            if (bitmap != null) {
                showBigNotification(bitmap, mBuilder, icon, title, message, timeStamp, resultPendingIntent, alarmSound);
            } else {
                showSmallNotification(mBuilder, icon, title, message, timeStamp, resultPendingIntent, alarmSound);
            }
        }
```



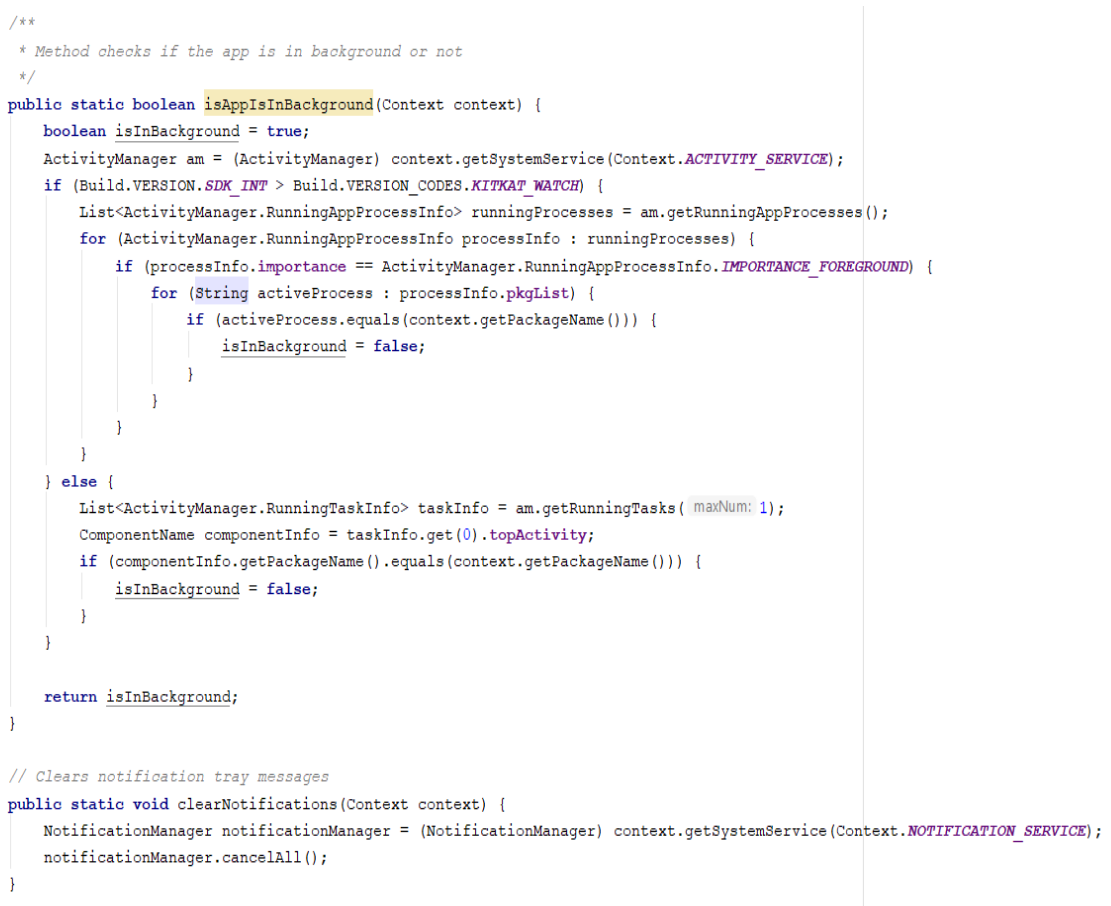
```
private void showSmallNotification(NotificationCompat.Builder mBuilder, int icon, String title,
                                   String message, String timeStamp, PendingIntent resultPendingIntent, Uri alarmSound) {

    NotificationCompat.InboxStyle inboxStyle = new NotificationCompat.InboxStyle();

    inboxStyle.addLine(message);

    Notification notification;
    notification = mBuilder.setSmallIcon(icon).setTicker(title).setWhen(0)
            .setAutoCancel(true)
            .setContentTitle(title)
            .setContentIntent(resultPendingIntent)
            .setSound(alarmSound)
            .setStyle(inboxStyle)
            .setWhen(getTimeMilliSec(timeStamp))
            .setSmallIcon(R.mipmap.ic_launcher)
            .setLargeIcon(BitmapFactory.decodeResource(mContext.getResources(), icon))
            .setContentText(message)
            .build();

    NotificationManager notificationManager = (NotificationManager) mContext.getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(Config.NOTIFICATION_ID, notification);
}
```



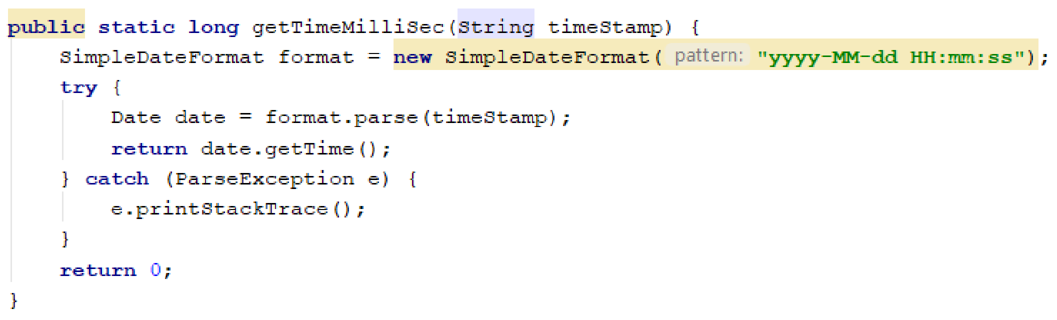
```java
private void showBigNotification(Bitmap bitmap, NotificationCompat.Builder mBuilder, int icon, String title,
                                 String message, String timeStamp, PendingIntent resultPendingIntent, Uri alarmSound) {
    NotificationCompat.BigPictureStyle bigPictureStyle = new NotificationCompat.BigPictureStyle();
    bigPictureStyle.setBigContentTitle(title);
    bigPictureStyle.setSummaryText(Html.fromHtml(message).toString());
    bigPictureStyle.bigPicture(bitmap);
    Notification notification;
    notification = mBuilder.setSmallIcon(icon).setTicker(title).setWhen(0)
            .setAutoCancel(true)
            .setContentTitle(title)
            .setContentIntent(resultPendingIntent)
            .setSound(alarmSound)
            .setStyle(bigPictureStyle)
            .setWhen(getTimeMilliSec(timeStamp))
            .setSmallIcon(R.mipmap.ic_launcher)
            .setLargeIcon(BitmapFactory.decodeResource(mContext.getResources(), icon))
            .setContentText(message)
            .build();

    NotificationManager notificationManager = (NotificationManager) mContext.getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(Config.NOTIFICATION_ID_BIG_IMAGE, notification);
}
```



```java
/**
 * Downloading push notification image before displaying it in
 * the notification tray
 */
public Bitmap getBitmapFromURL(String strURL) {
    try {
        URL url = new URL(strURL);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setDoInput(true);
        connection.connect();
        InputStream input = connection.getInputStream();
        Bitmap myBitmap = BitmapFactory.decodeStream(input);
        return myBitmap;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}

// Playing notification sound
public void playNotificationSound() {
    try {
        Uri alarmSound = Uri.parse(ContentResolver.SCHEME_ANDROID_RESOURCE
                + "://" + mContext.getPackageName() + "/raw/notification");
        Ringtone r = RingtoneManager.getRingtone(mContext, alarmSound);
        r.play();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

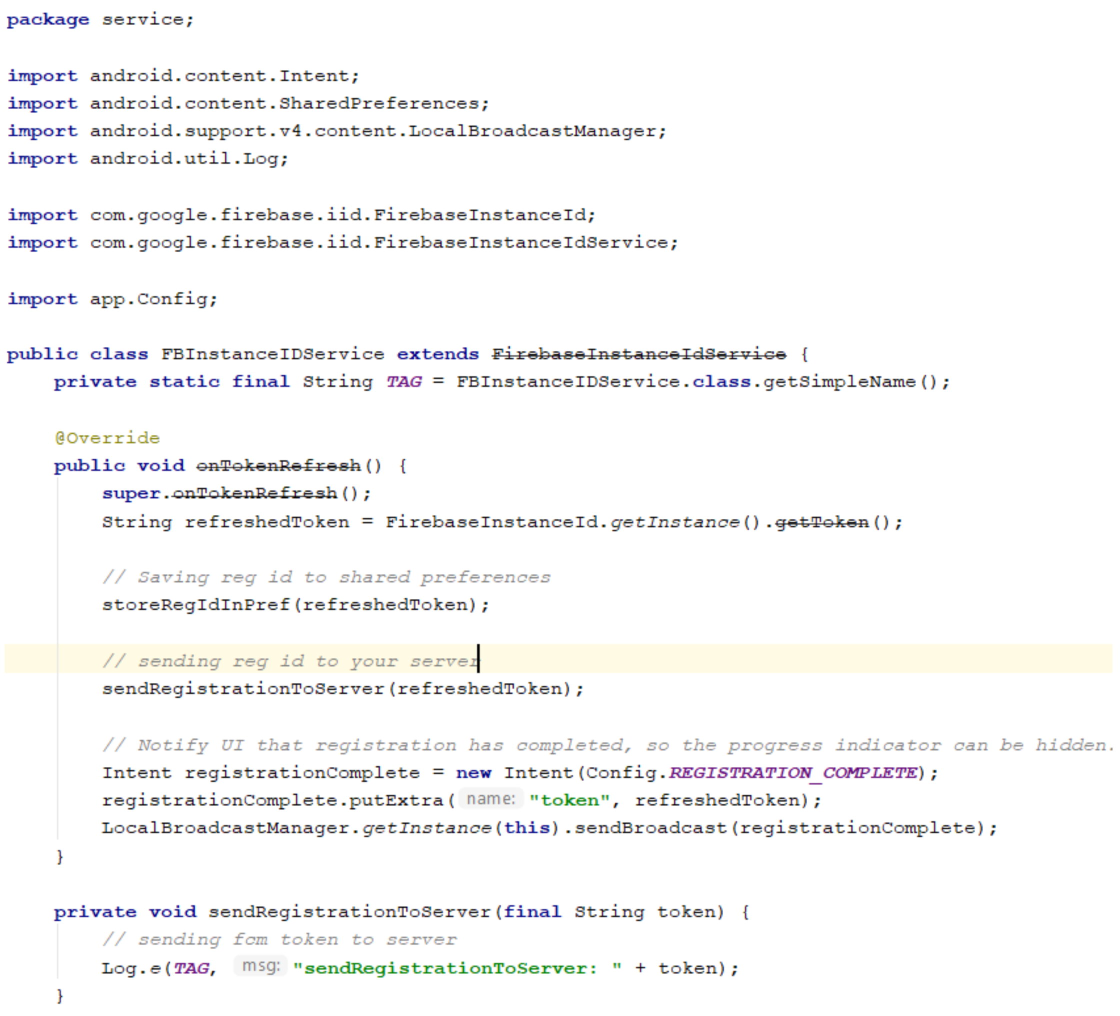


```java
/**
 * Method checks if the app is in background or not
 */
public static boolean isAppIsInBackground(Context context) {
    boolean isInBackground = true;
    ActivityManager am = (ActivityManager) context.getSystemService(Context.ACTIVITY_SERVICE);
    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.KITKAT_WATCH) {
        List<ActivityManager.RunningAppProcessInfo> runningProcesses = am.getRunningAppProcesses();
        for (ActivityManager.RunningAppProcessInfo processInfo : runningProcesses) {
            if (processInfo.importance == ActivityManager.RunningAppProcessInfo.IMPORTANCE_FOREGROUND) {
                for (String activeProcess : processInfo.pkgList) {
                    if (activeProcess.equals(context.getPackageName())) {
                        isInBackground = false;
                    }
                }
            }
        }
    } else {
        List<ActivityManager.RunningTaskInfo> taskInfo = am.getRunningTasks( maxNum: 1);
        ComponentName componentInfo = taskInfo.get(0).topActivity;
        if (componentInfo.getPackageName().equals(context.getPackageName())) {
            isInBackground = false;
        }
    }

    return isInBackground;
}

// Clears notification tray messages
public static void clearNotifications(Context context) {
    NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.cancelAll();
}
```

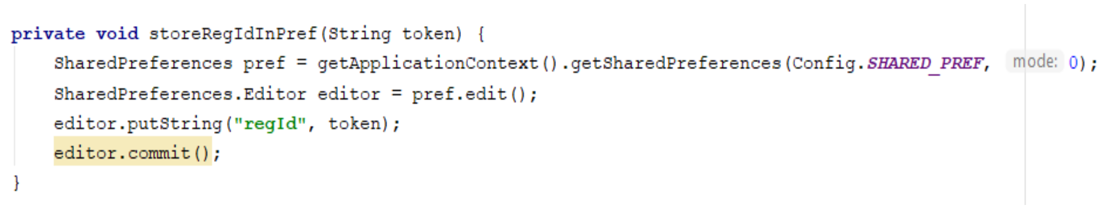


```java
public static long getTimeMilliSec(String timeStamp) {
    SimpleDateFormat format = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss");
    try {
        Date date = format.parse(timeStamp);
        return date.getTime();
    } catch (ParseException e) {
        e.printStackTrace();
    }
    return 0;
}
```

C.3 FBInstanceIDService.java

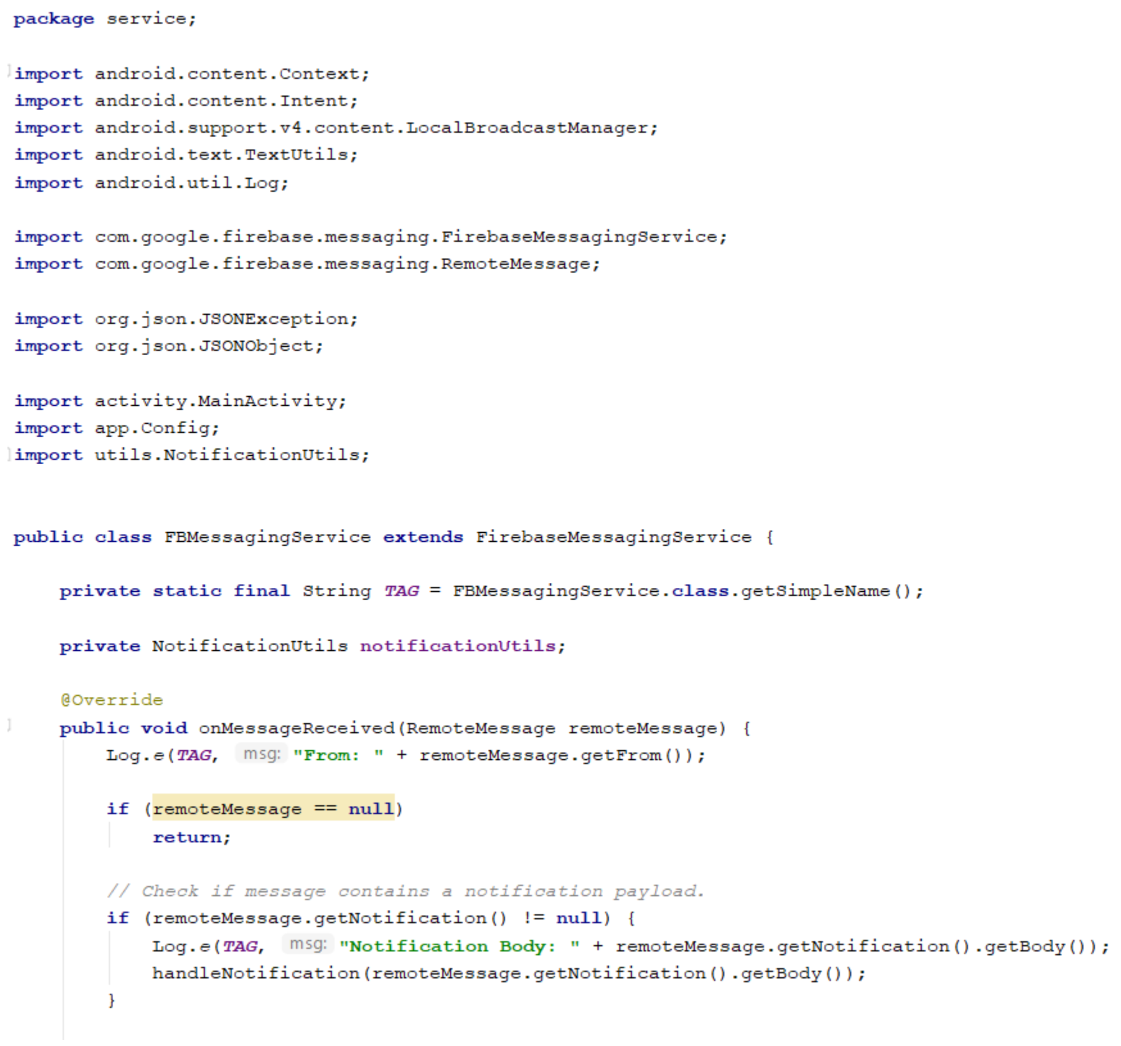


```
package service;

import android.content.Intent;
import android.content.SharedPreferences;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.FirebaseInstanceIdService;

import app.Config;

public class FBInstanceIDService extends ~~FirebaseInstanceIdService~~ {
    private static final String TAG = FBInstanceIDService.class.getSimpleName();

    @Override
    public void ~~onTokenRefresh~~() {
        super.~~onTokenRefresh~~();
        String refreshedToken = FirebaseInstanceId.getInstance().~~getToken~~();

        // Saving reg id to shared preferences
        storeRegIdInPref(refreshedToken);

        // sending reg id to your server
        sendRegistrationToServer(refreshedToken);

        // Notify UI that registration has completed, so the progress indicator can be hidden.
        Intent registrationComplete = new Intent(Config.REGISTRATION_COMPLETE);
        registrationComplete.putExtra( name: "token", refreshedToken);
        LocalBroadcastManager.getInstance(this).sendBroadcast(registrationComplete);
    }

    private void sendRegistrationToServer(final String token) {
        // sending fcm token to server
        Log.e(TAG,  msg: "sendRegistrationToServer: " + token);
    }
```

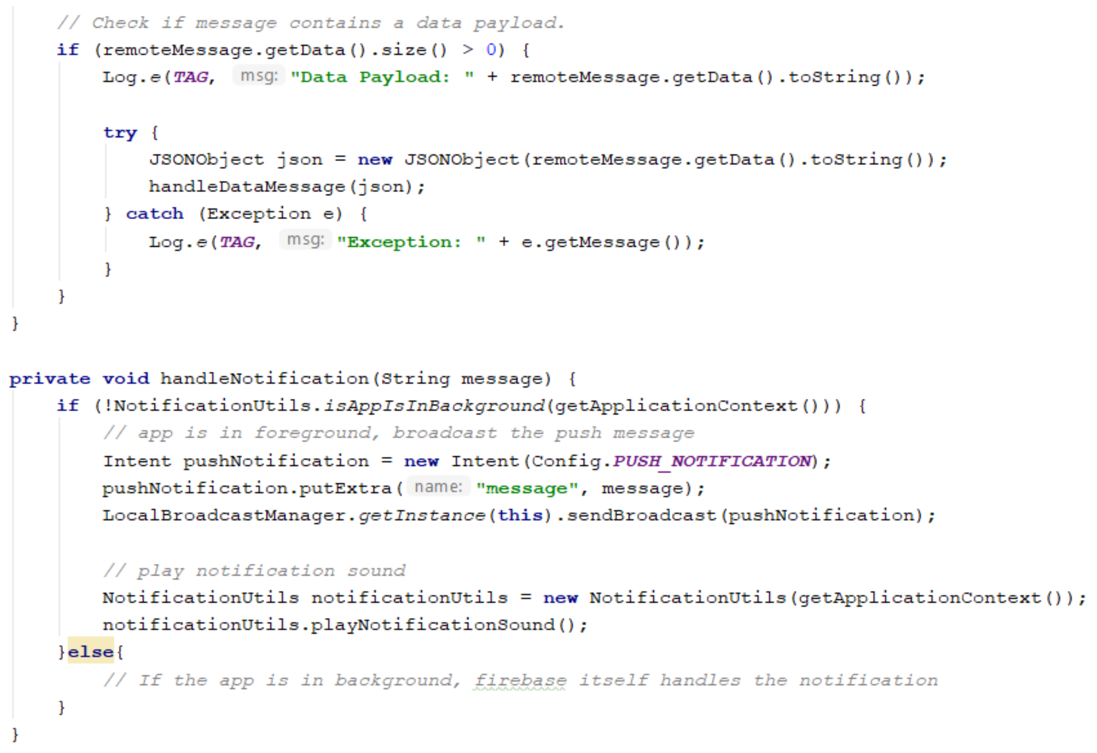


```
private void storeRegIdInPref(String token) {
    SharedPreferences pref = getApplicationContext().getSharedPreferences(Config.SHARED_PREF,  mode: 0);
    SharedPreferences.Editor editor = pref.edit();
    editor.putString("regId", token);
    editor.commit();
}
```

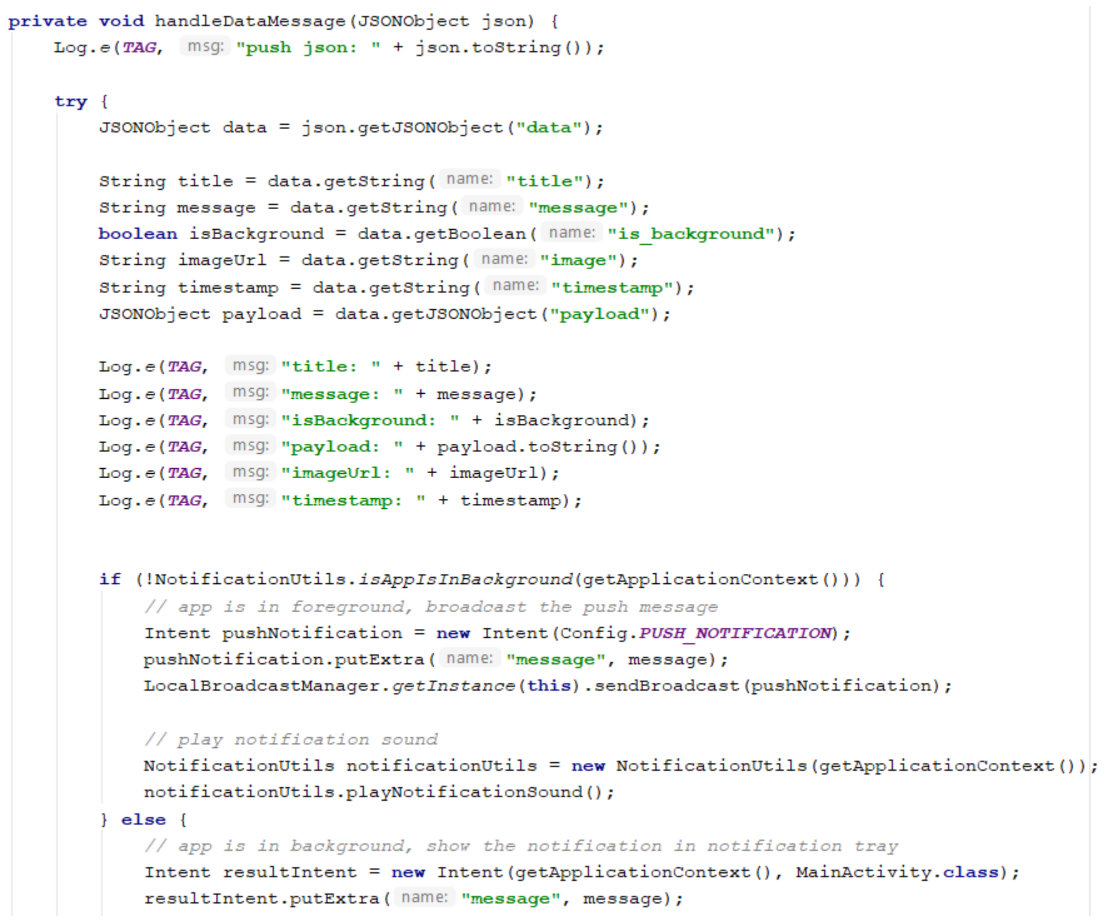
C.4 FBMessagingService.java



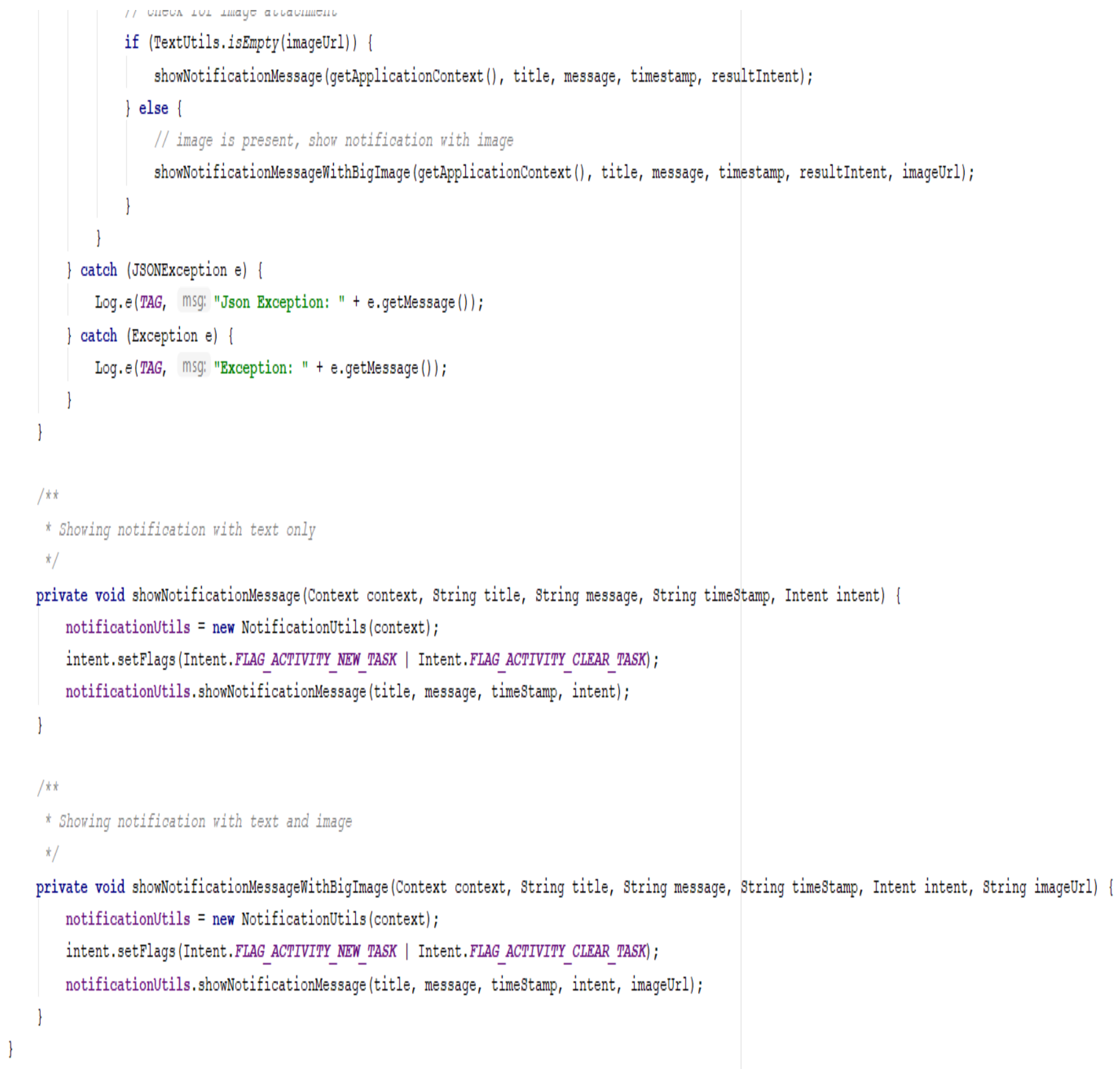
```java
package service;

import android.content.Context;
import android.content.Intent;
import android.support.v4.content.LocalBroadcastManager;
import android.text.TextUtils;
import android.util.Log;

import com.google.firebase.messaging.FirebaseMessagingService;
import com.google.firebase.messaging.RemoteMessage;

import org.json.JSONException;
import org.json.JSONObject;

import activity.MainActivity;
import app.Config;
import utils.NotificationUtils;


public class FBMessagingService extends FirebaseMessagingService {

    private static final String TAG = FBMessagingService.class.getSimpleName();

    private NotificationUtils notificationUtils;

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        Log.e(TAG,  msg: "From: " + remoteMessage.getFrom());

        if (remoteMessage == null)
            return;

        // Check if message contains a notification payload.
        if (remoteMessage.getNotification() != null) {
            Log.e(TAG,  msg: "Notification Body: " + remoteMessage.getNotification().getBody());
            handleNotification(remoteMessage.getNotification().getBody());
        }
```



```
    // Check if message contains a data payload.
    if (remoteMessage.getData().size() > 0) {
        Log.e(TAG, msg: "Data Payload: " + remoteMessage.getData().toString());

        try {
            JSONObject json = new JSONObject(remoteMessage.getData().toString());
            handleDataMessage(json);
        } catch (Exception e) {
            Log.e(TAG, msg: "Exception: " + e.getMessage());
        }
    }
}

private void handleNotification(String message) {
    if (!NotificationUtils.isAppIsInBackground(getApplicationContext())) {
        // app is in foreground, broadcast the push message
        Intent pushNotification = new Intent(Config.PUSH_NOTIFICATION);
        pushNotification.putExtra( name: "message", message);
        LocalBroadcastManager.getInstance(this).sendBroadcast(pushNotification);

        // play notification sound
        NotificationUtils notificationUtils = new NotificationUtils(getApplicationContext());
        notificationUtils.playNotificationSound();
    }else{
        // If the app is in background, firebase itself handles the notification
    }
}
```

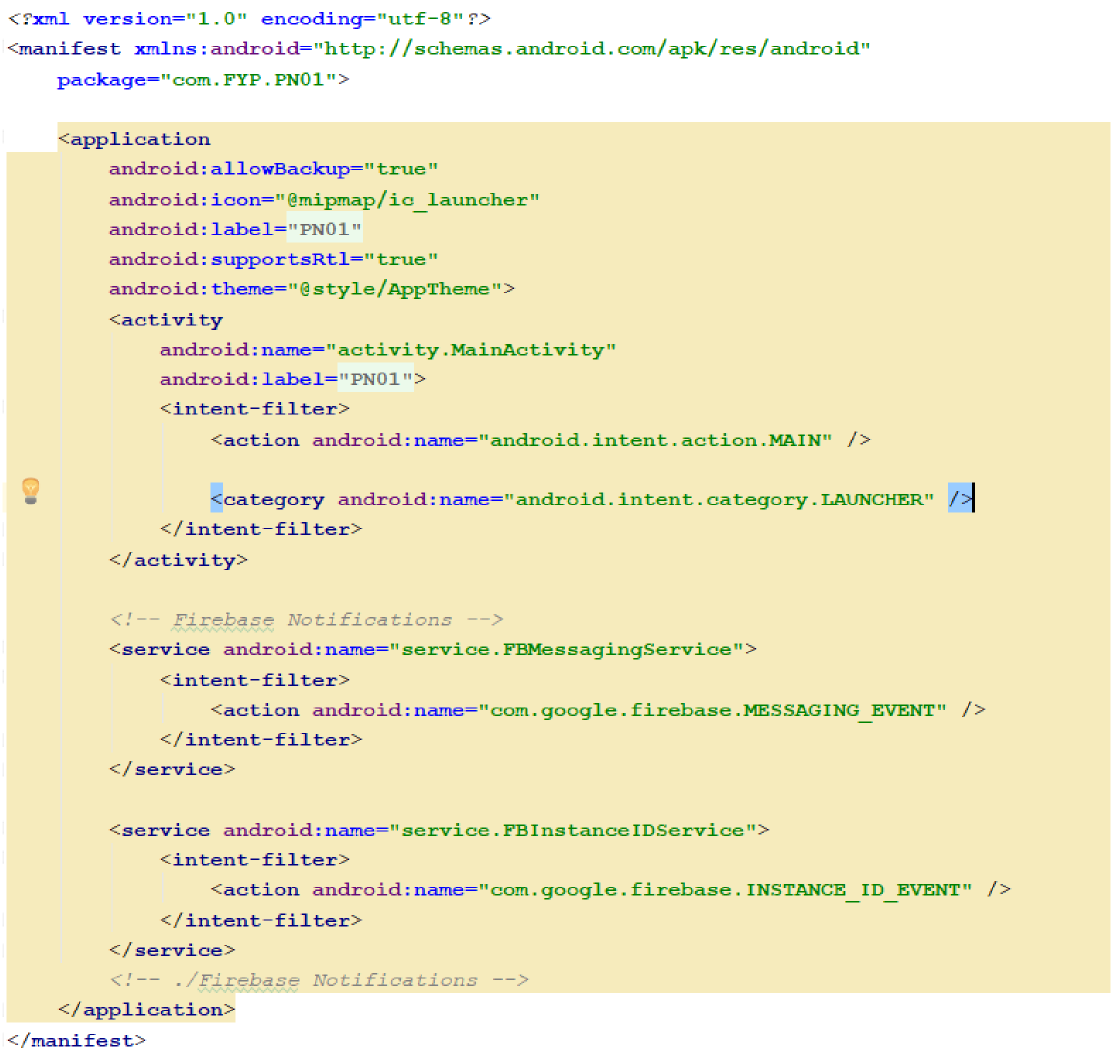


```
private void handleDataMessage(JSONObject json) {
    Log.e(TAG, msg: "push json: " + json.toString());

    try {
        JSONObject data = json.getJSONObject("data");

        String title = data.getString( name: "title");
        String message = data.getString( name: "message");
        boolean isBackground = data.getBoolean( name: "is_background");
        String imageUrl = data.getString( name: "image");
        String timestamp = data.getString( name: "timestamp");
        JSONObject payload = data.getJSONObject("payload");

        Log.e(TAG, msg: "title: " + title);
        Log.e(TAG, msg: "message: " + message);
        Log.e(TAG, msg: "isBackground: " + isBackground);
        Log.e(TAG, msg: "payload: " + payload.toString());
        Log.e(TAG, msg: "imageUrl: " + imageUrl);
        Log.e(TAG, msg: "timestamp: " + timestamp);


        if (!NotificationUtils.isAppIsInBackground(getApplicationContext())) {
            // app is in foreground, broadcast the push message
            Intent pushNotification = new Intent(Config.PUSH_NOTIFICATION);
            pushNotification.putExtra( name: "message", message);
            LocalBroadcastManager.getInstance(this).sendBroadcast(pushNotification);

            // play notification sound
            NotificationUtils notificationUtils = new NotificationUtils(getApplicationContext());
            notificationUtils.playNotificationSound();
        } else {
            // app is in background, show the notification in notification tray
            Intent resultIntent = new Intent(getApplicationContext(), MainActivity.class);
            resultIntent.putExtra( name: "message", message);
```



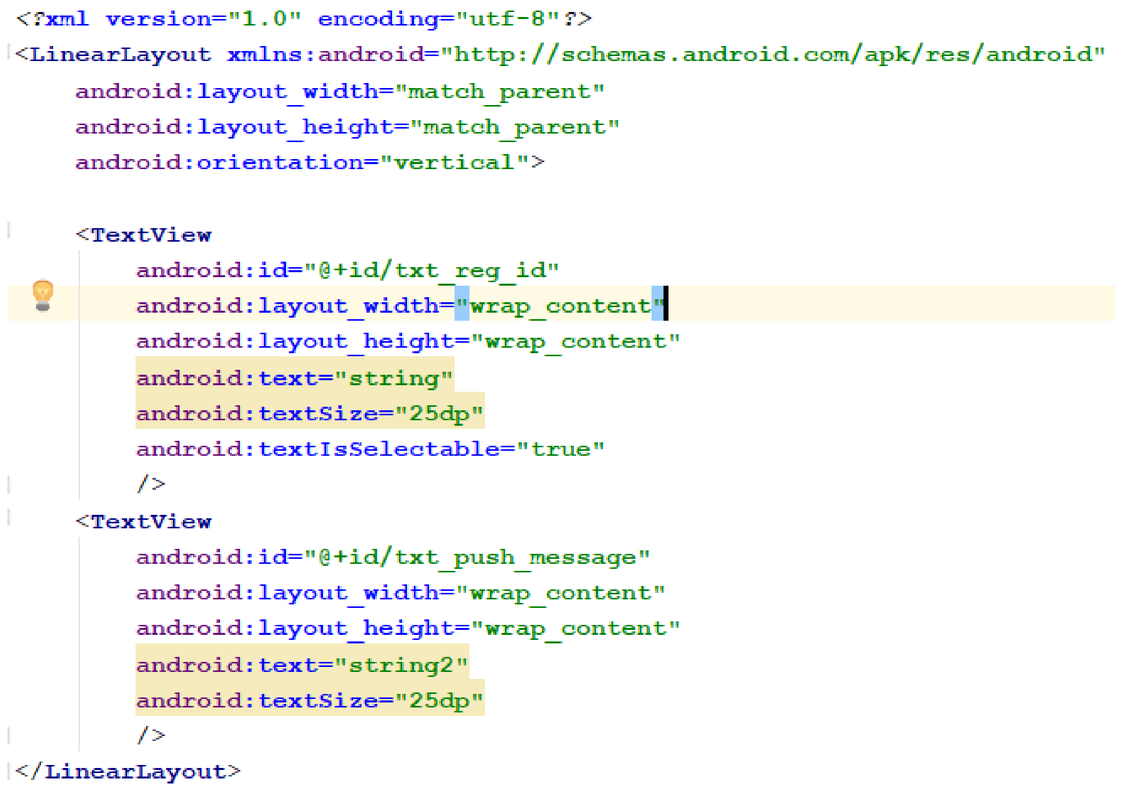
```
                // check for image attachment
                if (TextUtils.isEmpty(imageUrl)) {
                    showNotificationMessage(getApplicationContext(), title, message, timestamp, resultIntent);
                } else {
                    // image is present, show notification with image
                    showNotificationMessageWithBigImage(getApplicationContext(), title, message, timestamp, resultIntent, imageUrl);
                }
            }
        } catch (JSONException e) {
            Log.e(TAG,  msg: "Json Exception: " + e.getMessage());
        } catch (Exception e) {
            Log.e(TAG,  msg: "Exception: " + e.getMessage());
        }
    }

    /**
     * Showing notification with text only
     */
    private void showNotificationMessage(Context context, String title, String message, String timeStamp, Intent intent) {
        notificationUtils = new NotificationUtils(context);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
        notificationUtils.showNotificationMessage(title, message, timeStamp, intent);
    }

    /**
     * Showing notification with text and image
     */
    private void showNotificationMessageWithBigImage(Context context, String title, String message, String timeStamp, Intent intent, String imageUrl) {
        notificationUtils = new NotificationUtils(context);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
        notificationUtils.showNotificationMessage(title, message, timeStamp, intent, imageUrl);
    }
}
```

C.5 AndroidManifest.xml



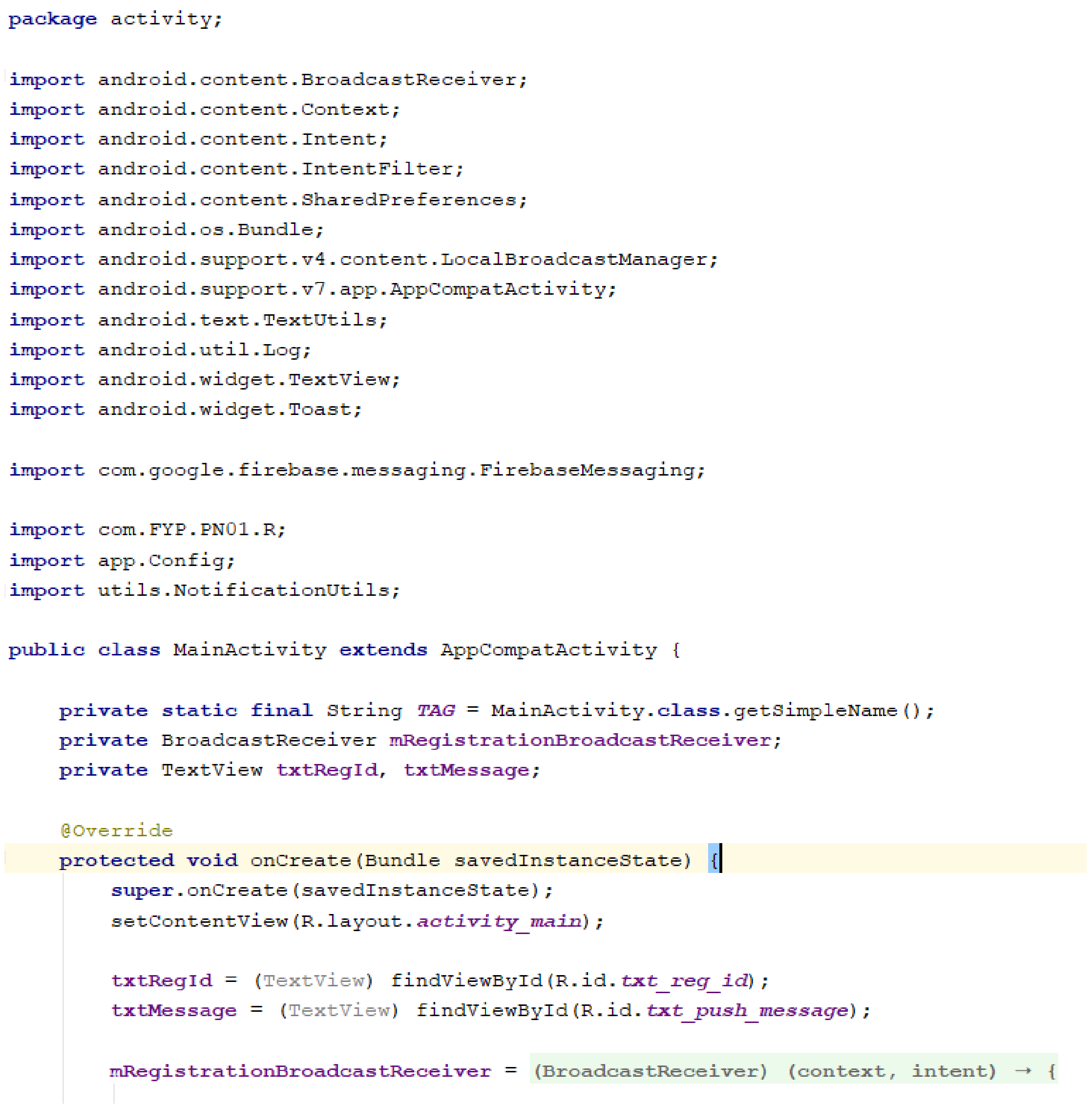
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.FYP.PN01">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="PN01"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name="activity.MainActivity"
            android:label="PN01">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Firebase Notifications -->
        <service android:name="service.FBMessagingService">
            <intent-filter>
                <action android:name="com.google.firebase.MESSAGING_EVENT" />
            </intent-filter>
        </service>

        <service android:name="service.FBInstanceIDService">
            <intent-filter>
                <action android:name="com.google.firebase.INSTANCE_ID_EVENT" />
            </intent-filter>
        </service>
        <!-- ./Firebase Notifications -->
    </application>
</manifest>
```

C.6 activity_main.xml

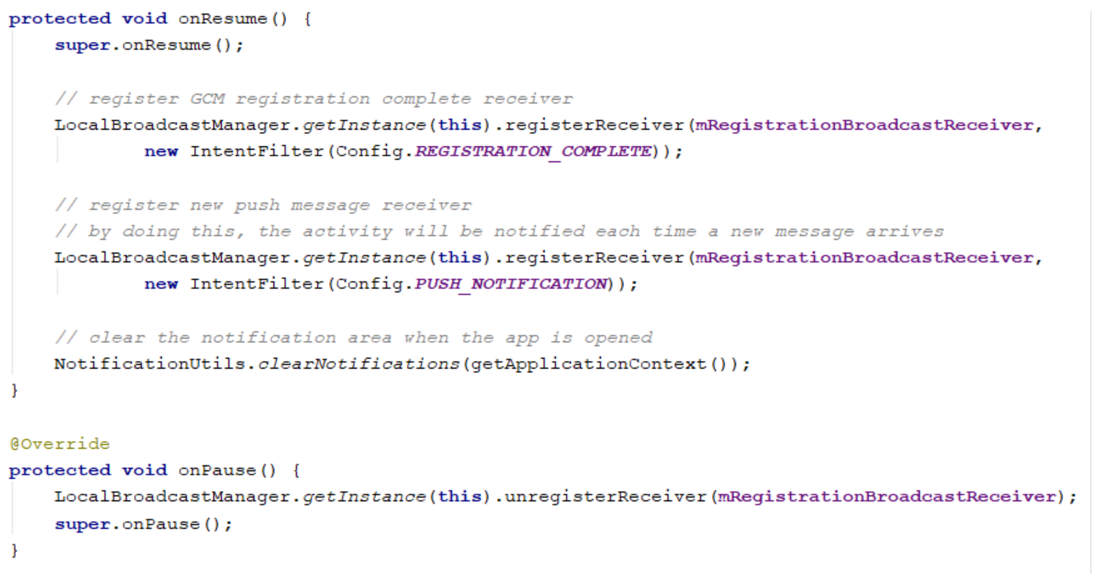


```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/txt_reg_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="string"
        android:textSize="25dp"
        android:textIsSelectable="true"
        />
    <TextView
        android:id="@+id/txt_push_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="string2"
        android:textSize="25dp"
        />
</LinearLayout>
```

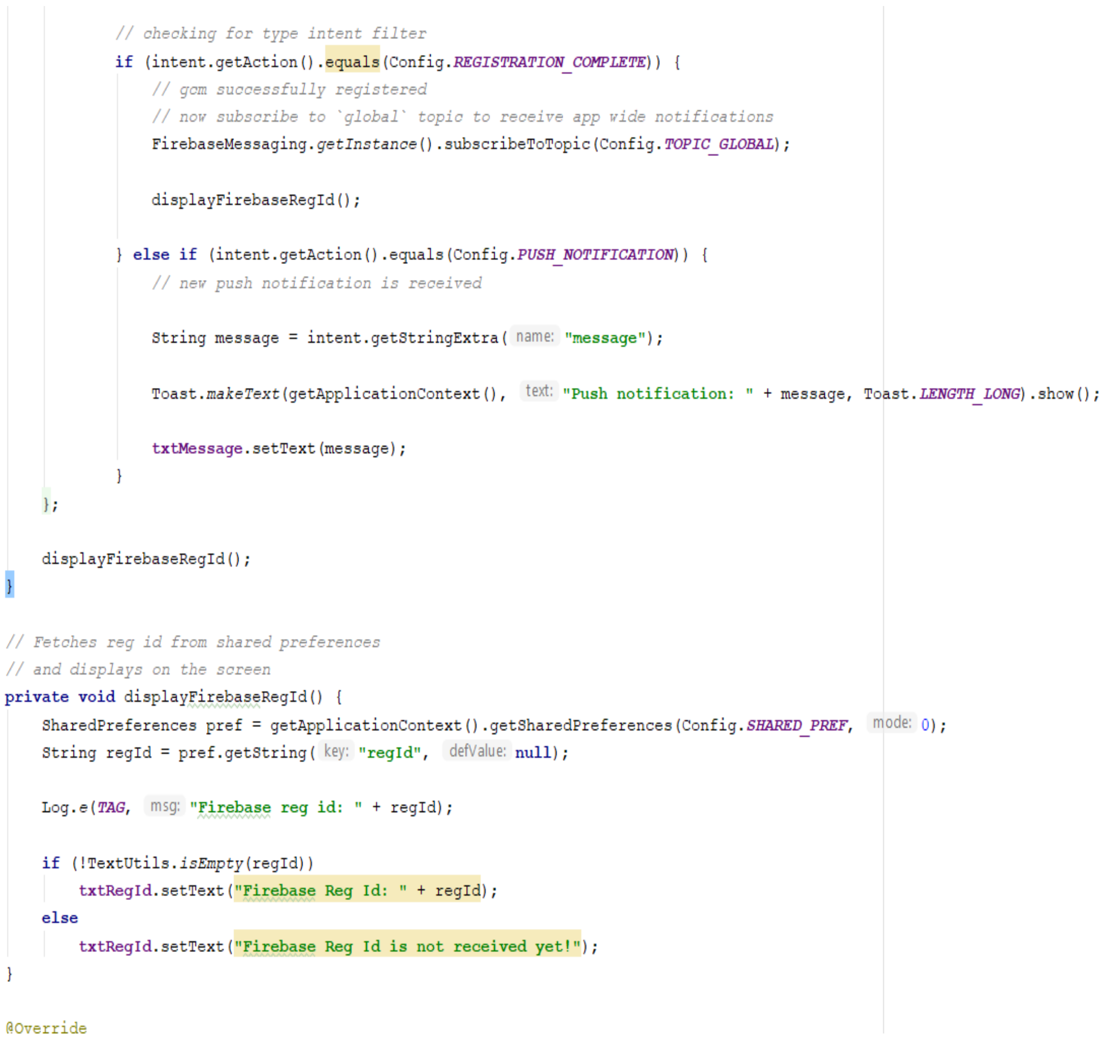
C.7 MainActivity.java



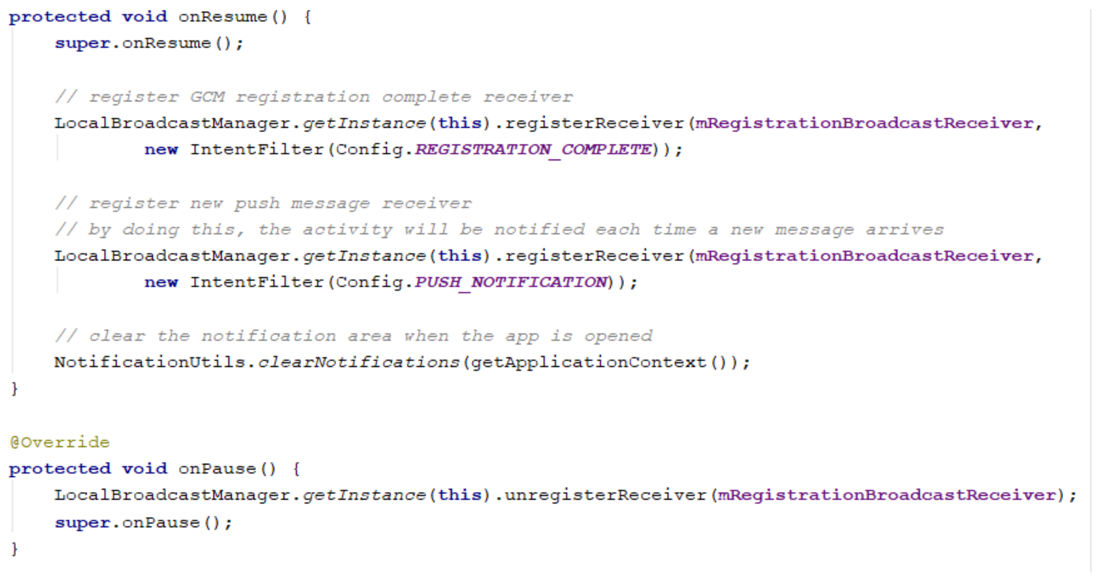
```
package activity;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.AppCompatActivity;
import android.text.TextUtils;
import android.util.Log;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.messaging.FirebaseMessaging;

import com.FYP.PN01.R;
import app.Config;
import utils.NotificationUtils;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = MainActivity.class.getSimpleName();
    private BroadcastReceiver mRegistrationBroadcastReceiver;
    private TextView txtRegId, txtMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        txtRegId = (TextView) findViewById(R.id.txt_reg_id);
        txtMessage = (TextView) findViewById(R.id.txt_push_message);

        mRegistrationBroadcastReceiver = (BroadcastReceiver) (context, intent) → {
```



```
protected void onResume() {
    super.onResume();

    // register GCM registration complete receiver
    LocalBroadcastManager.getInstance(this).registerReceiver(mRegistrationBroadcastReceiver,
            new IntentFilter(Config.REGISTRATION_COMPLETE));

    // register new push message receiver
    // by doing this, the activity will be notified each time a new message arrives
    LocalBroadcastManager.getInstance(this).registerReceiver(mRegistrationBroadcastReceiver,
            new IntentFilter(Config.PUSH_NOTIFICATION));

    // clear the notification area when the app is opened
    NotificationUtils.clearNotifications(getApplicationContext());
}

@Override
protected void onPause() {
    LocalBroadcastManager.getInstance(this).unregisterReceiver(mRegistrationBroadcastReceiver);
    super.onPause();
}
```

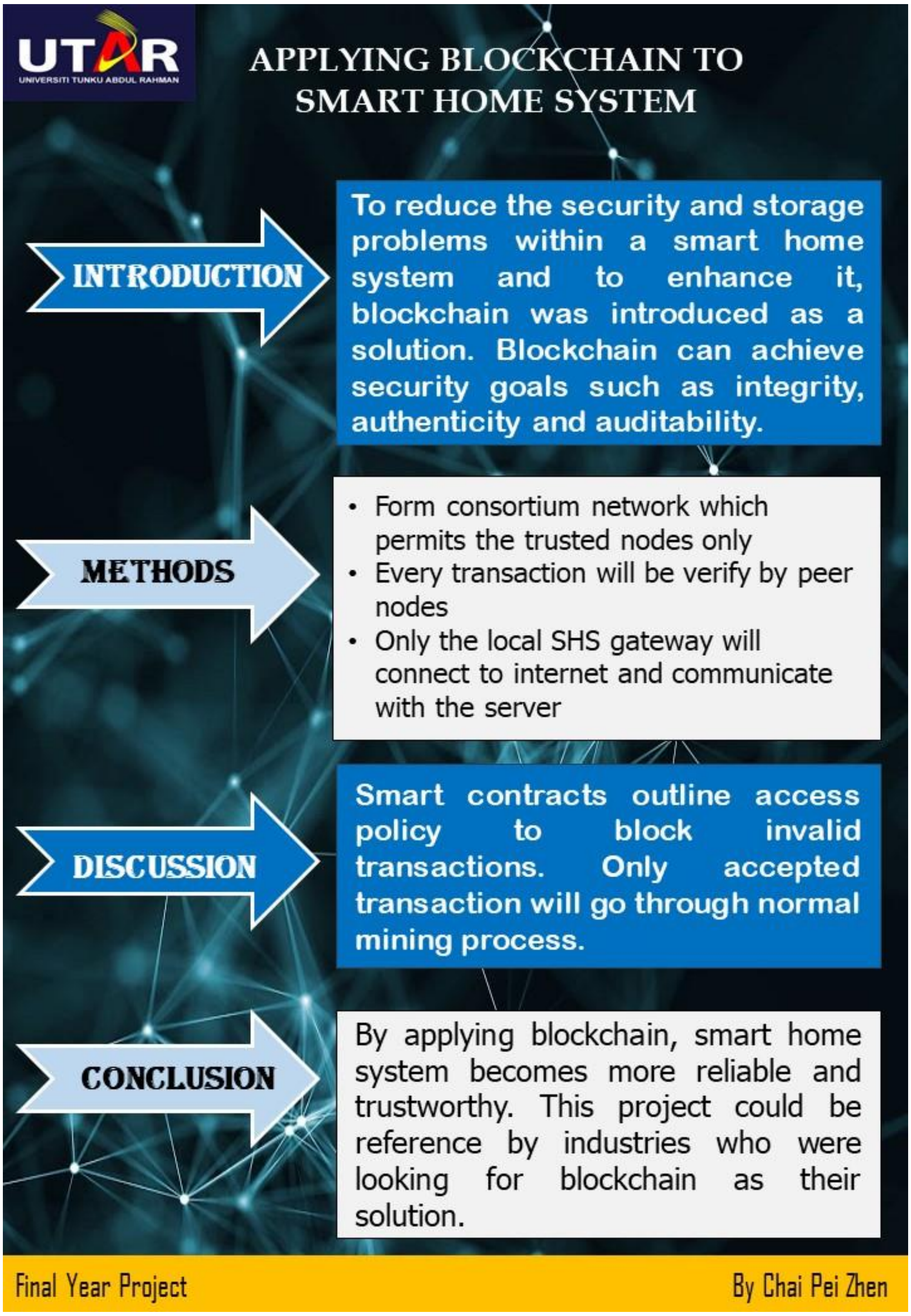


```java
            // checking for type intent filter
            if (intent.getAction().equals(Config.REGISTRATION_COMPLETE)) {
                // gcm successfully registered
                // now subscribe to `global` topic to receive app wide notifications
                FirebaseMessaging.getInstance().subscribeToTopic(Config.TOPIC_GLOBAL);

                displayFirebaseRegId();

            } else if (intent.getAction().equals(Config.PUSH_NOTIFICATION)) {
                // new push notification is received

                String message = intent.getStringExtra( name: "message");

                Toast.makeText(getApplicationContext(),  text: "Push notification: " + message, Toast.LENGTH_LONG).show();

                txtMessage.setText(message);
            }
    };

    displayFirebaseRegId();
}

// Fetches reg id from shared preferences
// and displays on the screen
private void displayFirebaseRegId() {
    SharedPreferences pref = getApplicationContext().getSharedPreferences(Config.SHARED_PREF,  mode: 0);
    String regId = pref.getString( key: "regId",  defValue: null);

    Log.e(TAG,  msg: "Firebase reg id: " + regId);

    if (!TextUtils.isEmpty(regId))
        txtRegId.setText("Firebase Reg Id: " + regId);
    else
        txtRegId.setText("Firebase Reg Id is not received yet!");
}

@Override
```



```java
protected void onResume() {
    super.onResume();

    // register GCM registration complete receiver
    LocalBroadcastManager.getInstance(this).registerReceiver(mRegistrationBroadcastReceiver,
            new IntentFilter(Config.REGISTRATION_COMPLETE));

    // register new push message receiver
    // by doing this, the activity will be notified each time a new message arrives
    LocalBroadcastManager.getInstance(this).registerReceiver(mRegistrationBroadcastReceiver,
            new IntentFilter(Config.PUSH_NOTIFICATION));

    // clear the notification area when the app is opened
    NotificationUtils.clearNotifications(getApplicationContext());
}

@Override
protected void onPause() {
    LocalBroadcastManager.getInstance(this).unregisterReceiver(mRegistrationBroadcastReceiver);
    super.onPause();
}
```

APPENDIX D: Poster



UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

APPLYING BLOCKCHAIN TO SMART HOME SYSTEM

INTRODUCTION

To reduce the security and storage problems within a smart home system and to enhance it, blockchain was introduced as a solution. Blockchain can achieve security goals such as integrity, authenticity and auditability.

METHODS

- Form consortium network which permits the trusted nodes only
- Every transaction will be verify by peer nodes
- Only the local SHS gateway will connect to internet and communicate with the server

DISCUSSION

Smart contracts outline access policy to block invalid transactions. Only accepted transaction will go through normal mining process.

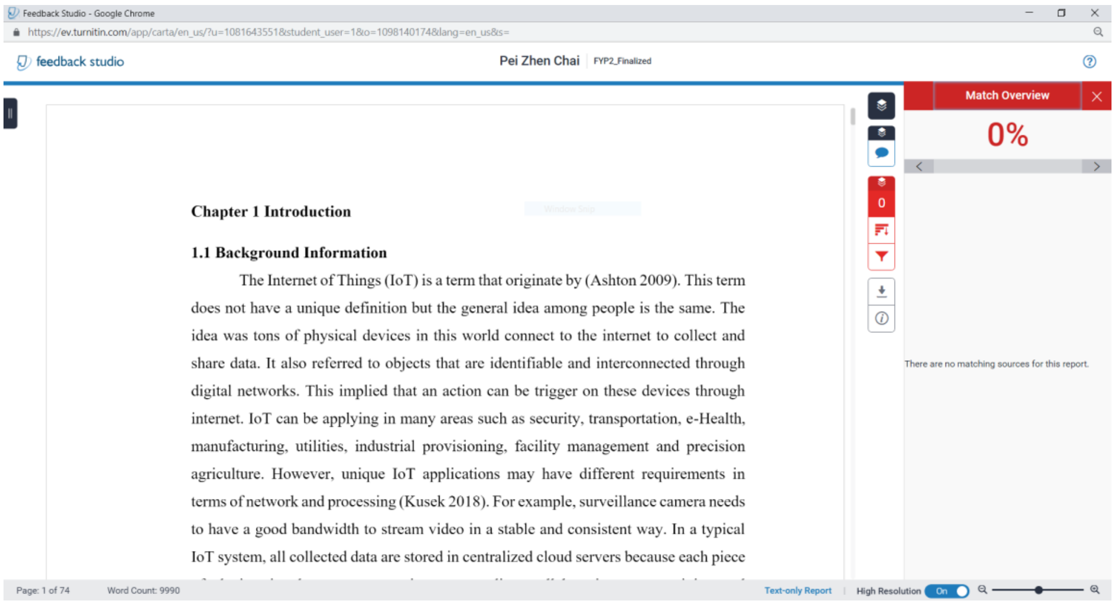
CONCLUSION

By applying blockchain, smart home system becomes more reliable and trustworthy. This project could be reference by industries who were looking for blockchain as their solution.

Final Year Project

By Chai Pei Zhen

APPENDIX E: Plagiarism Check Result



Feedback Studio - Google Chrome

https://ev.turnitin.com/app/carta/en_us/?u=1081643551&student_user=1&o=1098140174&lang=en_us&s=

feedback studio — Pei Zhen Chai | FYP2_Finalized

Match Overview

0%

There are no matching sources for this report.

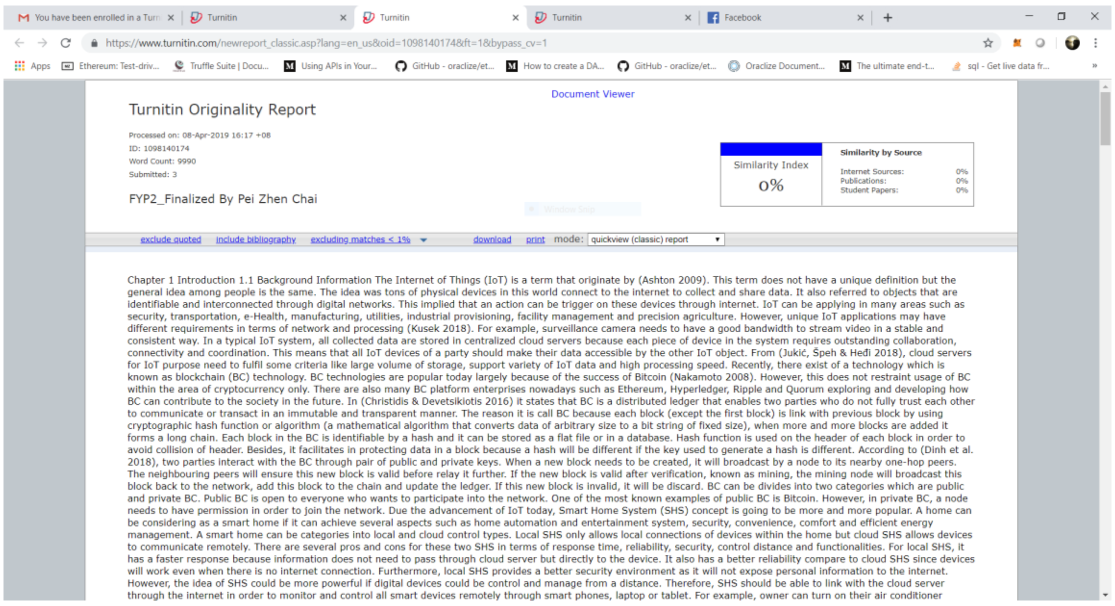
Chapter 1 Introduction

1.1 Background Information

The Internet of Things (IoT) is a term that originate by (Ashton 2009). This term does not have a unique definition but the general idea among people is the same. The idea was tons of physical devices in this world connect to the internet to collect and share data. It also referred to objects that are identifiable and interconnected through digital networks. This implied that an action can be trigger on these devices through internet. IoT can be applying in many areas such as security, transportation, e-Health, manufacturing, utilities, industrial provisioning, facility management and precision agriculture. However, unique IoT applications may have different requirements in terms of network and processing (Kusek 2018). For example, surveillance camera needs to have a good bandwidth to stream video in a stable and consistent way. In a typical IoT system, all collected data are stored in centralized cloud servers because each piece

Page: 1 of 74 Word Count: 9990

Text-only Report | High Resolution On



https://www.turnitin.com/newreport_classic.asp?lang=en_us&oid=1098140174&ft=1&bypass_cv=1

Document Viewer

Turnitin Originality Report

Processed on: 08-Apr-2019 16:17 +08
ID: 1098140174
Word Count: 9990
Submitted: 3

FYP2_Finalized By Pei Zhen Chai

Similarity Index	Similarity by Source	
0%	Internet Sources:	0%
	Publications:	0%
	Student Papers:	0%

exclude quoted include bibliography excluding matches < 1% download print mode: quickview (classic) report

Chapter 1 Introduction 1.1 Background Information The Internet of Things (IoT) is a term that originate by (Ashton 2009). This term does not have a unique definition but the general idea among people is the same. The idea was tons of physical devices in this world connect to the internet to collect and share data. It also referred to objects that are identifiable and interconnected through digital networks. This implied that an action can be trigger on these devices through internet. IoT can be applying in many areas such as security, transportation, e-Health, manufacturing, utilities, industrial provisioning, facility management and precision agriculture. However, unique IoT applications may have different requirements in terms of network and processing (Kusek 2018). For example, surveillance camera needs to have a good bandwidth to stream video in a stable and consistent way. In a typical IoT system, all collected data are stored in centralized cloud servers because each piece of device in the system requires outstanding collaboration, connectivity and coordination. This means that all IoT devices of a party should make their data accessible by the other IoT object. From (Jukić, Špeh & Heđi 2018), cloud servers for IoT purpose need to fulfil some criteria like large volume of storage, support variety of IoT data and high processing speed. Recently, there exist of a technology which is known as blockchain (BC) technology. BC technologies are popular today largely because of the success of Bitcoin (Nakamoto 2008). However, this does not restraint usage of BC within the area of cryptocurrency only. There are also many BC platform enterprises nowadays such as Ethereum, Hyperledger, Ripple and Quorum exploring and developing how BC can contribute to the society in the future. In (Christidis & Devetsikiotis 2016) it states that BC is a distributed ledger that enables two parties who do not fully trust each other to communicate or transact in an immutable and transparent manner. The reason it is call BC because each block (except the first block) is link with previous block by using cryptographic hash function or algorithm (a mathematical algorithm that converts data of arbitrary size to a bit string of fixed size), when more and more blocks are added it forms a long chain. Each block in the BC is identifiable by a hash and it can be stored as a flat file or in a database. Hash function is used on the header of each block in order to avoid collision of header. Besides, it facilitates in protecting data in a block because a hash will be different if the key used to generate a hash is different. According to (Dinh et al. 2018), two parties interact with the BC through pair of public and private keys. When a new block needs to be created, it will broadcast by a node to its nearby one-hop peers. The neighbouring peers will ensure this new block is valid before relay it further. If the new block is valid after verification, known as mining, the mining node will broadcast this block back to the network, add this block to the chain and update the ledger. If this new block is invalid, it will be discard. BC can be divides into two categories which are public and private BC. Public BC is open to everyone who wants to participate into the network. One of the most known examples of public BC is Bitcoin. However, in private BC, a node needs to have permission in order to join the network. Due the advancement of IoT today, Smart Home System (SHS) concept is going to be more and more popular. A home can be considering as a smart home if it can achieve several aspects such as home automation and entertainment system, security, convenience, comfort and efficient energy management. A smart home can be categories into local and cloud control types. Local SHS only allows local connections of devices within the home but cloud SHS allows devices to communicate remotely. There are several pros and cons for these two SHS in terms of response time, reliability, security, control distance and functionalities. For local SHS, it has a faster response because information does not need to pass through cloud server but directly to the device. It also has a better reliability compare to cloud SHS since devices will work even when there is no internet connection. Furthermore, local SHS provides a better security environment as it will not expose personal information to the internet. However, the idea of SHS could be more powerful if digital devices could be control and manage from a distance. Therefore, SHS should be able to link with the cloud server through the internet in order to monitor and control all smart devices remotely through smart phones, laptop or tablet. For example, owner can turn on their air conditioner

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Chai Pei Zhen
ID Number(s)	1502713
Programme / Course	Computer Science
Title of Final Year Project	Applying Blockchain To Smart Home System

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: ______ % Similarity by source Internet Sources: _______% Publications: _______ % Student Papers: _______%	
Number of individual sources listed of more than 3% similarity:______________	
Parameters of originality required and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor	Signature of Co-Supervisor
Name: ___________________________	Name: ___________________________
Date: ___________________________	Date: ___________________________

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	1502713
Student Name	Chai Pei Zhen
Supervisor Name	Dr. Lee Wai Kong

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Cover
	Signed Report Status Declaration Form
	Title Page
	Signed form of the Declaration of Originality
	Acknowledgement
	Abstract
	Table of Contents
	List of Figures (if applicable)
	List of Tables (if applicable)
	List of Symbols (if applicable)
	List of Abbreviations (if applicable)
	Chapters / Content
	Bibliography (or References)
	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
	Poster
	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report. ________________________ (Signature of Student) Date:	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction. ________________________ (Signature of Supervisor) Date: