New York Institute of Technology

School of Engineering and Computing Sciences

INCS 870

CAR RENTAL SYSTEM USING BLOCKCHAIN FOR TRANSACTIONS

By

Group-2

Michael Jonathan Injeti - 1247011
Nikhil Prakash Kammar - 1236529
Prabhjot Singh - 1200037

Guide
Prof. Wei Li

Date Submitted:
April 22nd 2019

Car Rental System Using Blockchain for Transactions

Michael Jonathan Injeti, Nikhil Prakash Kammar, Prabhjot Singh

Department of Information, Network and Computer Security,
School of Engineering and Computing Sciences,
NYIT University, Vancouver, BC

Abstract:

"Advancements of the internet are radical in the last couple of decades, especially in the consumer business field. It has dramatically influenced on business process and has facilitated easy communication between business and its customers. The Car Rental System that we are proposing will help the dealers manage the transactions and renting process. Additionally, it helps the customer to view available cars, register, and book a car. We are using Blockchain technology to implement an online wallet system for secure transactions".

Keywords— Car rental, Blockchain, Security.

Introduction:

We are familiar with the car renting process where the customer borrows a car for a period of time in return they pay the dealer. It helps the people who don't own a vehicle or doesn't have access to their own vehicle to get around. In this process, the customer has to contact the dealer which can be made easy with our project by providing an online platform. It also helps the people to surf the available cars, reserve a car for a period of time, make payments securely, and return the car. Additionally, it helps the dealer to manage cars and transactions.

Our objective is to create an online system where a customer can easily register and reserve a car online, and a dealer can manage the cars and transactions effectively and securely. This system enhances the communication between customer and dealer with more accuracy, and it also helps to maintain the log of every car and customer. The transaction is done through the online wallet which uses the Blockchain technology to ensure security.

This car rental system also enhances the business process by allowing for advance bookings from anywhere in the world; it helps plane your trip better. It gives a wide range of available cars with different price ranges from which customer can choose according to their needs. It takes an identity verification document to avoid malicious users. Additionally, it sends a confirmation email to the customer to enhance this security. As everything is done online, it saves the usage of paper. Hence it's eco-friendly. This service is available 24/7, so it is beneficial to customers.

Scope:

During the process of the development of this project we have researched wide range of topic like business concepts to computational field, Blockchain, servers and many other topics which are listed below.

- Car rental industry: Research on current trends in car renting business, where the customers are facing difficulties, how can we help with that issues and systems that we can automate.
- Blockchain to build a virtual wallet: We are using Blockchain technology to build the virtual wallet which will be integrated with the customer's account so we had to research on Blockchain technology.
- Apache maven servers: We are using Apache maven 3.6 servers to host our system so we had to research on this field to gain more control on the subject.
- Java backend: All the backend programming is done on java platform. We had to learn new set of libraries related to web development and about hosting the it on the apache servers.
- Online platform which is available 24/7: we have to provide 24/7 service to our customers so we had to research on suitable technology that can support this.
- Customer service: We researched on how to give better service to the customers and added the feedback form to the system.
- Mango Data base: We are using mango DB to access, manipulate and delete data so we had to learn all the basic and advance concepts related to this.

Functional and Non-Functional Requirements:

Requirement analysis is a part of planning of a software which is comprised of different tasks that determines the needs and conditions that has to be met for the smooth running of a software.

Functional requirements are those requirements which involves internal working of a software, description of each module and its sub-module. It describes tasks that should be performed by each system, the process involved in it and the resources that it uses to get the results. The functional requirements that we found are:

- Customer registration: The system should allow new member registrations online, update the database with new member’s information and generate a unique ID for them.
- Reservation of cars: The system should allow the customers to make bookings and online reservations.
- Updating the database: Whenever there is a new registration of a customer or of a customer makes a new car reservation there are number of variables that needs to be updated on the data base. The system should allow these updates and make the required changes in the database.
- Handling transactions: This system is using Blockchain to handle the transactions. The system should be able to make the changes whenever a customer makes a payment or adds value to the virtual wallet.

Non-functional requirements are the aspects which is concerned with how the system can deliver the functional requirements. These are the essential for smooth and effective working of software. The non-functional requirements that we found are:

- Security: The system must provide security and integrity for all sub systems and the data which is stored in the data base. Only authorized candidate should be able to access and manipulate the data on company side of the server. Customers are supposed to have valid username and passwords to login to their respective accounts.

- Performance: The system should have quick and accurate response time for all the processes. It shouldn’t keep the customer waiting for results or crash in the middle of a process.
- Error handling: Errors that occur in the process must be minimized, if user run into an error appropriate message must be displayed so that user can recover from error. Error handling time should be considerably less. User input validation is highly recommended to avoid hacking attacks from a malicious user.
- Availability: The access to the system must be available 24/7. If there is any occurrence of a major system malfunctions, the system should be able recover in as short time as possible.
- Ease of use: Consideration of level of users of system should be taken into account before building the interface. The user interface should be easy to work with by no or little training.

Methodology:

Three main components that we have to consider in this system are customer, admin (dealer) and the car rental software. In this chapter we explain the how everything works inside the system by using the data flow diagrams. They are the graphical representation of the data flow and transformation from input to output.

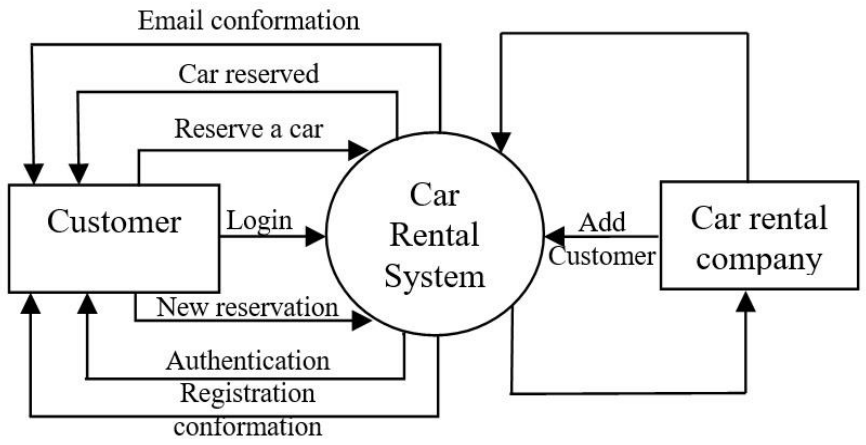


Fig. 1. Data flow diagram level 0.

The figure 1 shows the level 0 data flow diagram of the car rental system. The Customer has to input all the details to start with a new registration in the car rental application. Admin will now send an authentication email with the link to the customer after cross-checking all the details. Customer has to respond to the authentication link for successful registration After successful registration, the customer will login into the car rental application and start new reservation or car rental service The system will automatically reserve the car taking all the inputs into consideration given by the customer. The admin will monitor all the car reservations and also the transaction reports from the database. Finally, an email will be sent showing full details to the customer confirming the new reservation of the car.

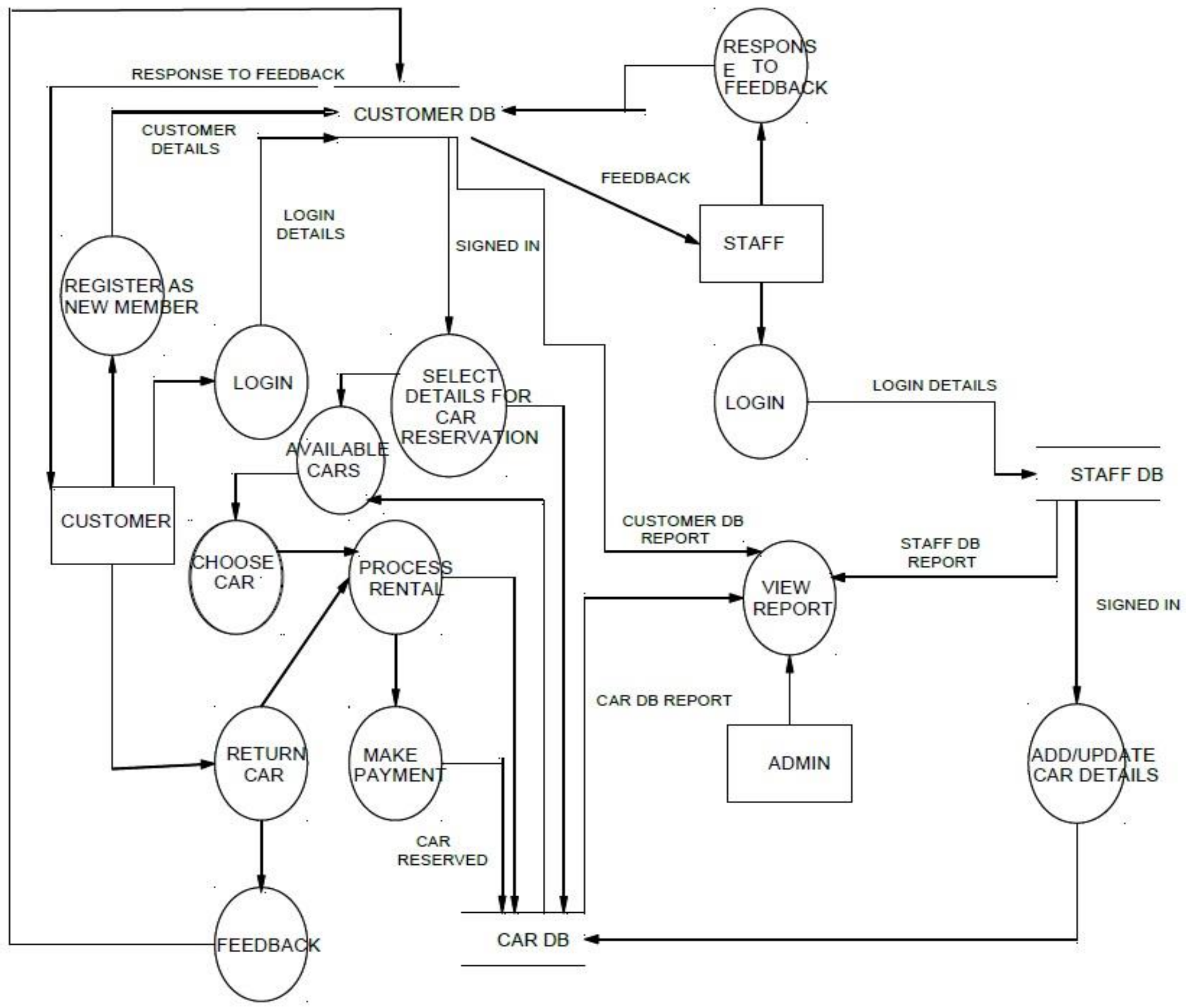


Fig. 2. Data flow diagram level 1.

The customer registers themselves as new members in the car rental application. Every customer will be given a unique ID/Token and their details will be stored in the customer data which can be accessed by admin/staff. After successful registration, the customer will login and start a new reservation by selecting details given in the application. The system will provide all the details to customer from the car data base. The car database and reports from the customer database will be monitored by the admin. The staff can access all the details in the application and be given the admin rights to change/update the cars and other details in the backend. The customer will be shown the available cars to choose. After choosing the car the system will start processing the rental and reserves the car. The next step will be the returning of car by customer, the system will process the rental and generate the bill for the customer to make payment. After payment is done successfully the system will ask for a feedback from the customer and it will be stored the customer database. The staff has the access to respond to any feedback or reports given by the customer and handles all the issues related to the car rentals and provide the customer service.

Blockchain:

[1] A Blockchain is a growing list of records, called blocks, which are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and a transaction data which is generally represented as a Markel tree.

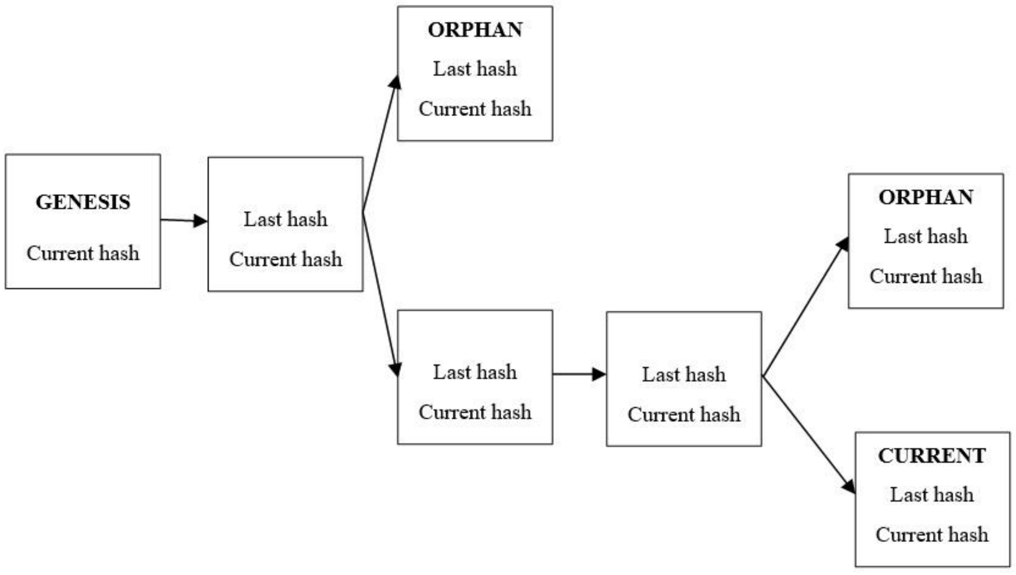


Fig.3. Blockchain formation

Design:

[2] Blockchain restricts the modification of the data. It is an open and distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way. For use as a distributed ledger, a Blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks.

Function:

Once recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks, which requires a consensus of the network majority. [3] Although Blockchain records are not unalterable, Blockchain may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance. Decentralized consensus has therefore been claimed with a Blockchain.

Hashes and Hash Functions

In the first place, Blockchain is informational before being economic or monetary, conducive to many emerging and increasingly popular token-free Blockchain. [4] It relies extensively on hashes and hash functions. A hash which is output is the result of a transformation of the original information (input). A hash function is a mathematical algorithm that takes an input and transforms it into an output. A cryptographic hash function is characterized by its extreme difficulty to revert, in other words, to recreate the input data from its hash value alone. This is called the collision resistance.

Payment Finality:

Payment finality is the discharge of an obligation by a transfer of funds and a transfer of securities that have become irrevocable and unconditional. [5] In a world of fiat money, payment finality is conceptualized about bank money within a triangular payment structure involving a payer, a payee, and a bank. Payment finality in a crypto-economy differs from the traditional banking system. A transaction is final once it is included in the Blockchain, thereby becoming simultaneously verifiable by many sources.

Implementation:

A. Membership Registration

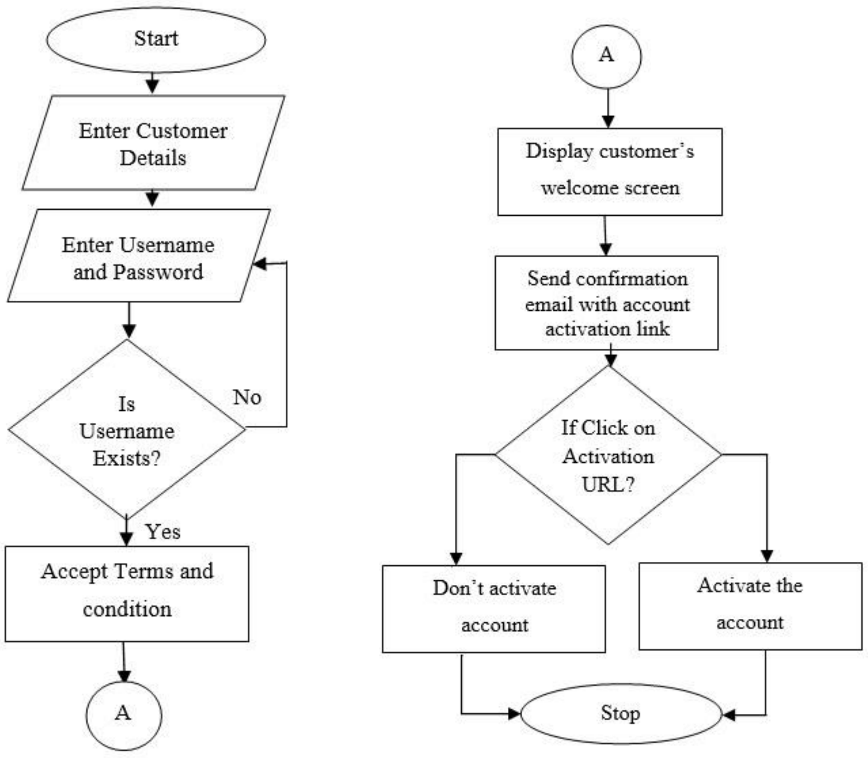

Start
Enter Customer Details
Enter Username and Password
Is Username Exists?
No
Yes
Accept Terms and condition
A
A
Display customer's welcome screen
Send confirmation email with account activation link
If Click on Activation URL?
Don't activate account
Activate the account
Stop

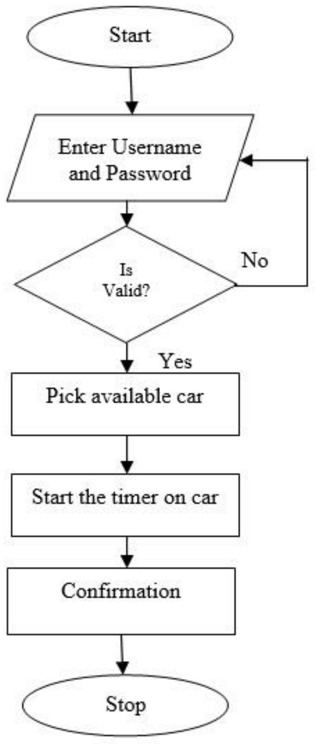
Fig. 3. Flow Chart for membership registration.

Above figure shows the flow chart of membership registration. It involves following steps:

- The customer will start the new registration online.
- The customer should input all the details i.e., Name, Phone, Number, ID- driving license or any government issued photo ID.
- They have to enter the Username and Password and the system will check if it exists and suggest to enter unique username.
- The customers had to read all the terms and conditions and should accept for to use the applications.
- The system will now accept all the details given by the customer and provide them with the welcome screen.
- The system will automatically send the confirmation email with the account activation link to customer's email ID

- The customer should accept the confirmation and should click on the link to verify their Email ID.
- Now the registration is successfully done.

B. Car Reservation and Return



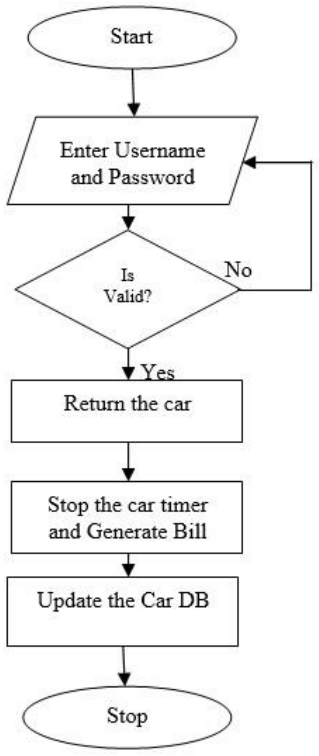

Start
Enter Username and Password
Is Valid?
No
Yes
Return the car
Stop the car timer and Generate Bill
Update the Car DB
Stop

Fig. 4. Flow Chart for Car registration.

Fig. 5. Flow Chart for Car return.

Above figure shows the flow chart of car registration and return. Steps for car registrations:

- User will login with valid credentials to get in their profile.
- Next, user will pick from available cars.
- The system starts the timer on the car and gives the confirmation on booking.

Steps for car return are:

- User will login with valid credentials to get in their profile.
- Next, the customer will return the car which stops the timer on that car.
- The system generates the bill, deducts from customer’s wallet and updates the car database.

C. Blockchain transactions

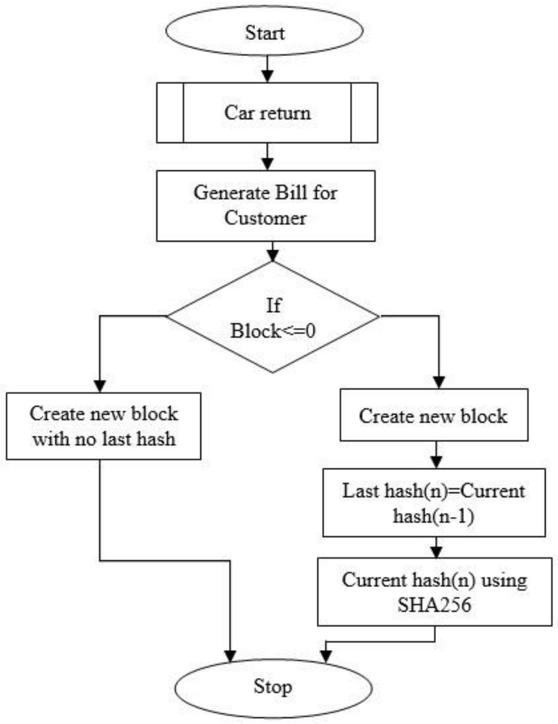

Start
Car return
Generate Bill for Customer
If Block<=0
Create new block with no last hash
Create new block
Last hash(n)=Current hash(n-1)
Current hash(n) using SHA256
Stop

Fig. 6. Flow Chart for Blockchain transactions.

- The figure 6 shows the flow chart for Blockchain transactions. It involves following steps:
- Whenever the Car is returned by the customer successfully, bill is generated for that particular transaction.
- When bill is generated a new block is added to the Blockchain.
- If there is no block in the database, then the system creates a new block with no last hash.
- If there is pre-existing blocks, then it creates a new block linking with last block's hash.
- The system uses SHA256 hash function and it takes current time, transaction, Last hash and nonce. Finally, it is authenticated by both public and private key.

Working:

A. System requirements:

Hardware requirements:

- Intel I5 5th Generation or above.
- Hard disk: 1 TB.
- RAM: 8 GB or above.
- OS: Windows 8 and above.

Software requirements:

- Java Development Kit with Eclipse oxygen.
- Apache maven.
- Postman API development environment.

B. Initial setup:

Initial setup for java development:

- Download Java SE Development Kit version 8 and install both JDK and JRE.
- Add a new environment variable with JDK location path address.
- Download eclipse oxygen and install it.
- Import all the modules.
- Start the system by running configuration setting and eureka modules to set the configurations for the system and make a link between apache server.
- Then start rest of the modules by running them as an application.

Initial setup for apache tomcat server:

- Download Apache maven 3.6 and install it on your system.
- Download MongoDB and install it on your system.
- Add a new environment variable with Apache maven's location path address.
- Now open the Command prompt and build all the modules by using the command:
  >mvn clean install -DskipTests=true
- Now the Car rental system is ready to work with.

C. Working of modules

Create new customer:

We use postman API development environment to test the outputs. To test the working of create new customer we pass the following variables through postman.

"name":"Mike",

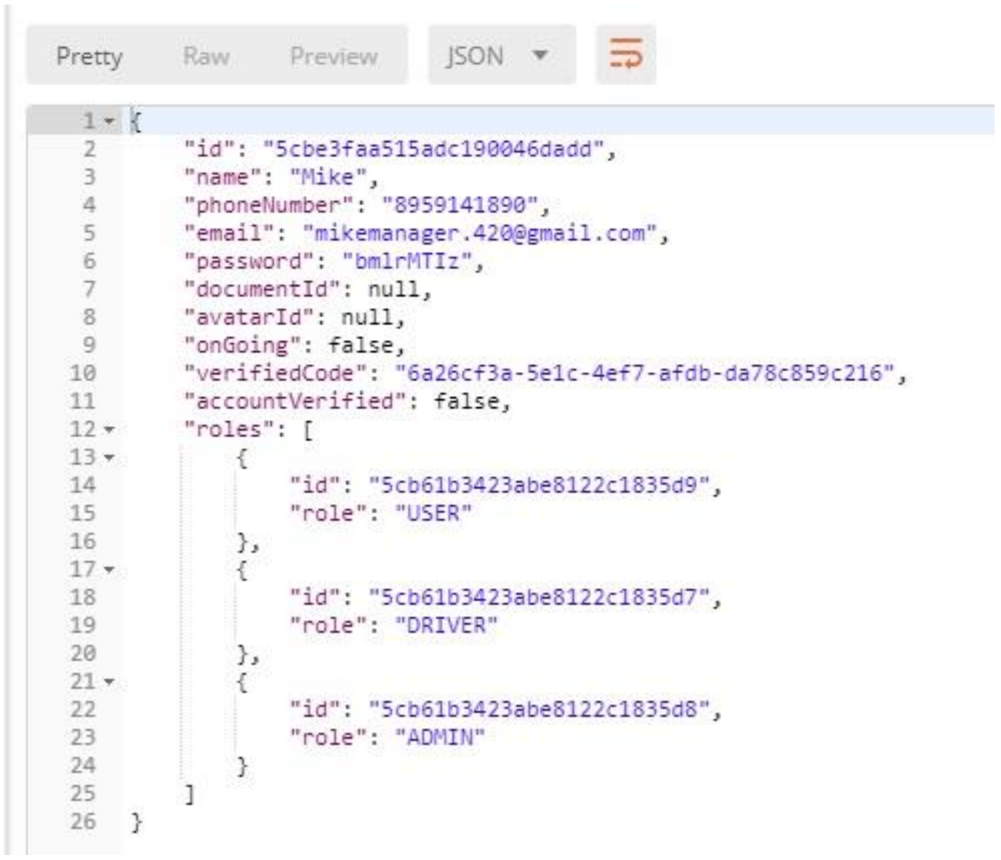
"phoneNumber":"8959141890",

"email":"mikemanager.420@gmail.com",

"roleTypes":["DRIVER","USER","ADMIN"],

"password":"nik123"

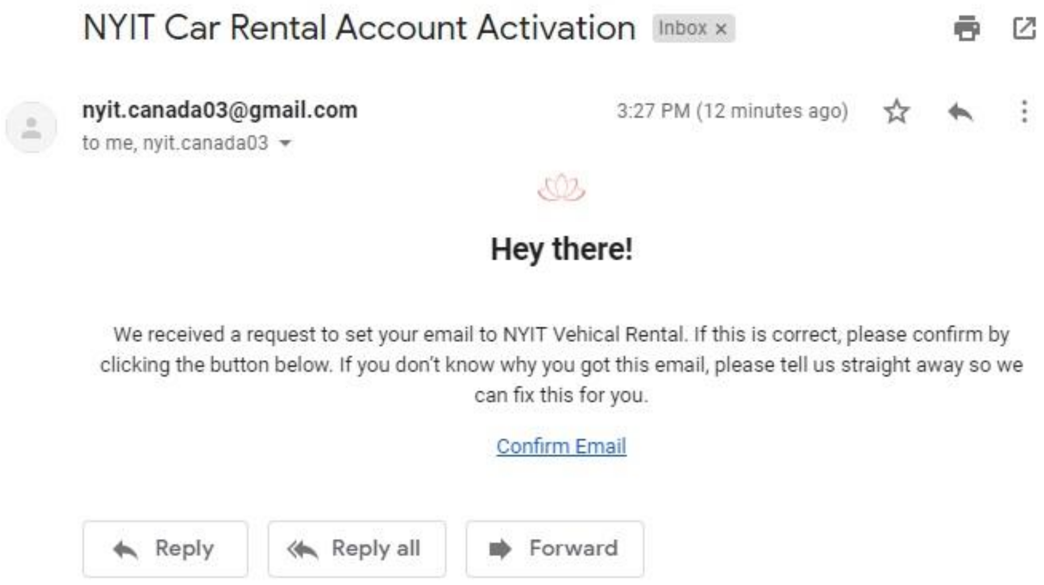
This creates a new user with unique ID number and encodes the password and saves it in the database.



```
{
    "id": "5cbe3faa515adc190046dadd",
    "name": "Mike",
    "phoneNumber": "8959141890",
    "email": "mikemanager.420@gmail.com",
    "password": "bmlrMTIz",
    "documentId": null,
    "avatarId": null,
    "onGoing": false,
    "verifiedCode": "6a26cf3a-5e1c-4ef7-afdb-da78c859c216",
    "accountVerified": false,
    "roles": [
        {
            "id": "5cb61b3423abe8122c1835d9",
            "role": "USER"
        },
        {
            "id": "5cb61b3423abe8122c1835d7",
            "role": "DRIVER"
        },
        {
            "id": "5cb61b3423abe8122c1835d8",
            "role": "ADMIN"
        }
    ]
}
```

Fig.7.Screenshot of created user.

Verification email is sent to the new customer's email address and customer has to click on the verification link to activate the account.


NYIT Car Rental Account Activation

nyit.canada03@gmail.com
to me, nyit.canada03
3:27 PM (12 minutes ago)

Hey there!

We received a request to set your email to NYIT Vehical Rental. If this is correct, please confirm by clicking the button below. If you don't know why you got this email, please tell us straight away so we can fix this for you.

Confirm Email

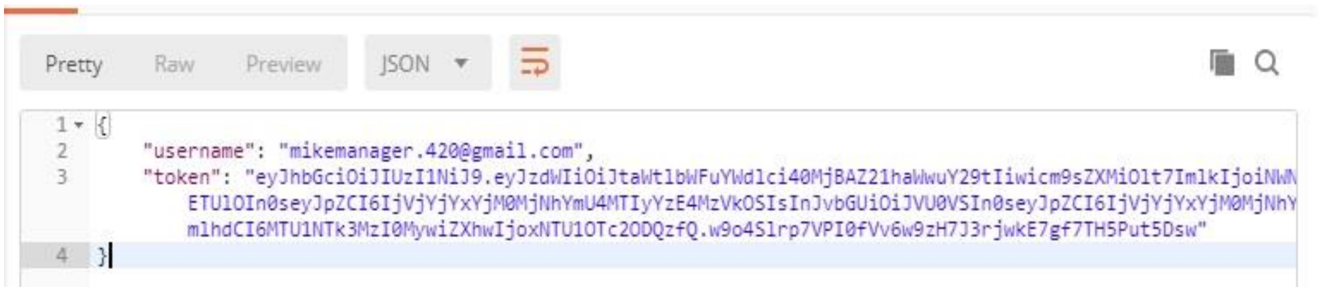
Fig. 8. Screenshot of verification email

Customer login:

User will login with valid credentials to get in their profile and customer is given a session token for an hour. To test the working of customer login we pass the following variables through postman.

"email":"mikemanager.420@gmail.com",

"password":"nik123"

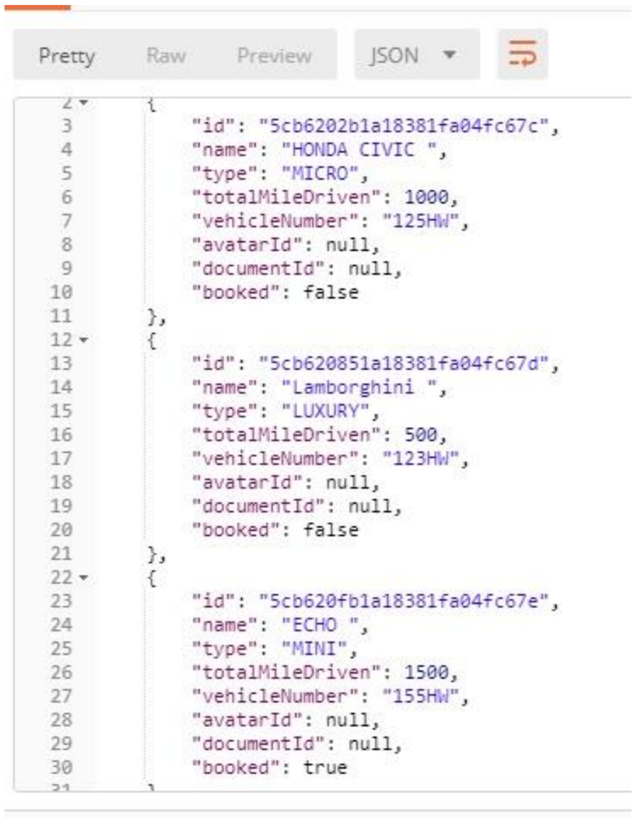


```
{
    "username": "mikemanager.420@gmail.com",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJtaWtlbWFuYWdlci40MjBAZ21haWwuY29tIiwicm9sZXMiOlt7ImlkIjoiNWN
        ETU1OIn0seyJpZCI6IjVjYjYxYjM0MjNhYmU4MTIyYzE4MzVkOSIsInJvbGUiOiJVU0VSIn0seyJpZCI6IjVjYjYxYjM0MjNhY
        mlhdCI6MTU1NTk3MzI0MywiZXhwIjoxNTU1OTc2ODQzfQ.w9o4Slrp7VPI0fVv6w9zH7J3rjwkE7gf7TH5Put5Dsw"
}
```

Fig. 9. Screenshot of Session token.

Car rent:

User will login with valid credentials get in their profile. Next, user will pick from available cars.



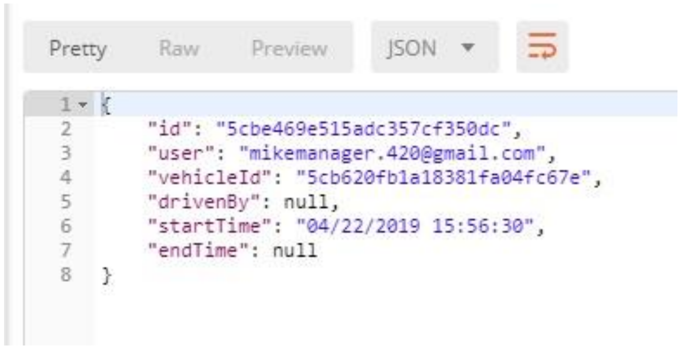
```
    {
        "id": "5cb6202b1a18381fa04fc67c",
        "name": "HONDA CIVIC ",
        "type": "MICRO",
        "totalMileDriven": 1000,
        "vehicleNumber": "125HW",
        "avatarId": null,
        "documentId": null,
        "booked": false
    },
    {
        "id": "5cb620851a18381fa04fc67d",
        "name": "Lamborghini ",
        "type": "LUXURY",
        "totalMileDriven": 500,
        "vehicleNumber": "123HW",
        "avatarId": null,
        "documentId": null,
        "booked": false
    },
    {
        "id": "5cb620fb1a18381fa04fc67e",
        "name": "ECHO ",
        "type": "MINI",
        "totalMileDriven": 1500,
        "vehicleNumber": "155HW",
        "avatarId": null,
        "documentId": null,
        "booked": true
```

Fig. 10. Screenshot of available cars.

To test the working of renting car module we pass the following variables through postman to get the confirmation. The system starts the timer on the car and gives the confirmation on booking.

"user": "mikemanager.420@gmail.com",

"vehicleId":"5cb620fb1a18381fa04fc67e"

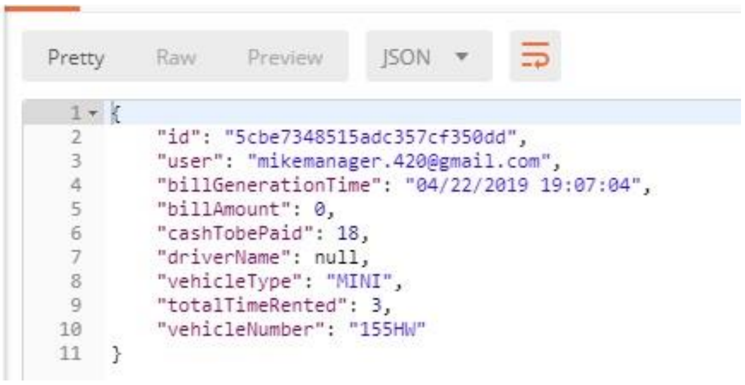


```
{
    "id": "5cbe469e515adc357cf350dc",
    "user": "mikemanager.420@gmail.com",
    "vehicleId": "5cb620fb1a18381fa04fc67e",
    "drivenBy": null,
    "startTime": "04/22/2019 15:56:30",
    "endTime": null
}
```

Fig.11. Screenshot of booking confirmation.

Return Car:

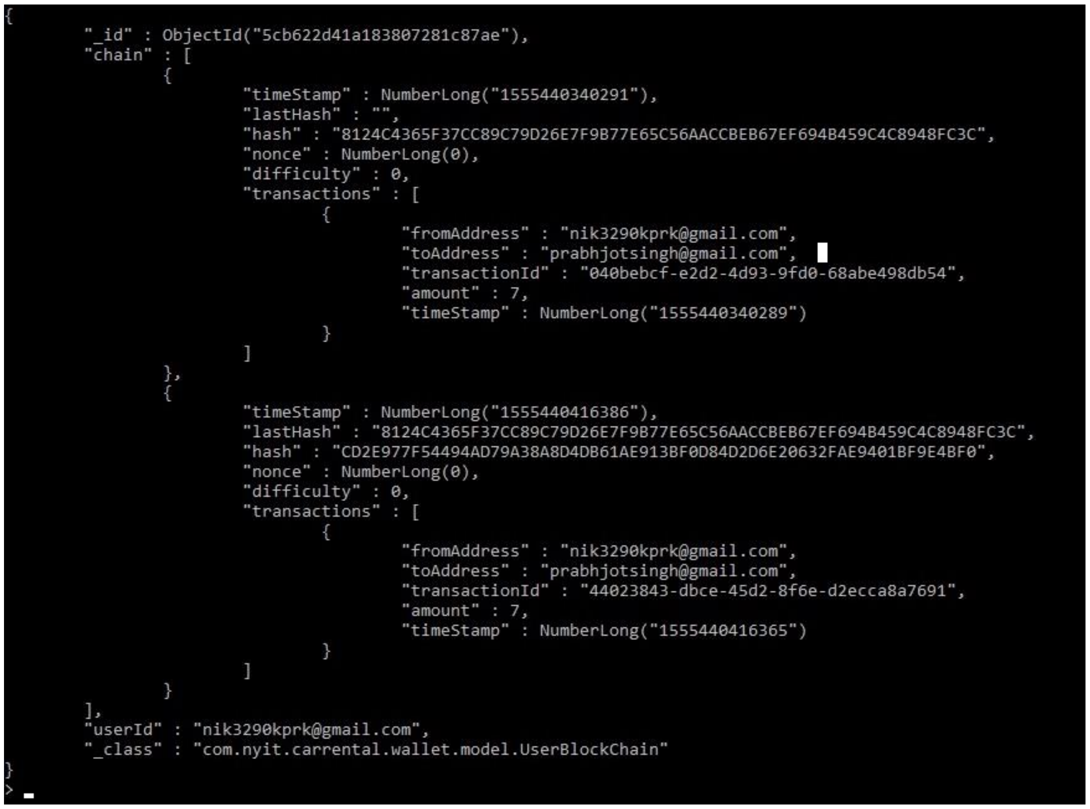
User will login with valid credentials get in their profile. Next, the customer will return the car which stops the timer on that car and generates the bill.



```
{
    "id": "5cbe7348515adc357cf350dd",
    "user": "mikemanager.420@gmail.com",
    "billGenerationTime": "04/22/2019 19:07:04",
    "billAmount": 0,
    "cashTobePaid": 18,
    "driverName": null,
    "vehicleType": "MINI",
    "totalTimeRented": 3,
    "vehicleNumber": "155HW"
}
```

Fig. 12. Screenshot of bill generated after car return.

When the bill is generated, the system creates a new block in Blockchain and updates the database. Transaction is validated by the admin (miner). We can see the block in the Blockchain database.



```
{
        "_id" : ObjectId("5cb622d41a183807281c87ae"),
        "chain" : [
                {
                        "timeStamp" : NumberLong("1555440340291"),
                        "lastHash" : "",
                        "hash" : "8124C4365F37CC89C79D26E7F9B77E65C56AACCBEB67EF694B459C4C8948FC3C",
                        "nonce" : NumberLong(0),
                        "difficulty" : 0,
                        "transactions" : [
                                {
                                        "fromAddress" : "nik3290kprk@gmail.com",
                                        "toAddress" : "prabhjotsingh@gmail.com",
                                        "transactionId" : "040bebcf-e2d2-4d93-9fd0-68abe498db54",
                                        "amount" : 7,
                                        "timeStamp" : NumberLong("1555440340289")
                                }
                        ]
                },
                {
                        "timeStamp" : NumberLong("1555440416386"),
                        "lastHash" : "8124C4365F37CC89C79D26E7F9B77E65C56AACCBEB67EF694B459C4C8948FC3C",
                        "hash" : "CD2E977F54494AD79A38A8D4DB61AE913BF0D84D2D6E20632FAE9401BF9E4BF0",
                        "nonce" : NumberLong(0),
                        "difficulty" : 0,
                        "transactions" : [
                                {
                                        "fromAddress" : "nik3290kprk@gmail.com",
                                        "toAddress" : "prabhjotsingh@gmail.com",
                                        "transactionId" : "44023843-dbce-45d2-8f6e-d2ecca8a7691",
                                        "amount" : 7,
                                        "timeStamp" : NumberLong("1555440416365")
                                }
                        ]
                }
        ],
        "userId" : "nik3290kprk@gmail.com",
        "_class" : "com.nyit.carrental.wallet.model.UserBlockChain"
}
> _
```

Fig. 13. First two blocks in the Blockchain.

Conclusion:

Car rental business has been updated with new technology compared to the experience where every activity concerning car rental business is limited to a physical location. Now a day cars are being taken as rent and being customers reserve cars, and even the cars are taken to their doorsteps. Customers who are registered members hire the car to go to their office. The web-based car rental system has offered an advantage to both customers as well as Car Rental Company to efficiently and effectively manage the business and satisfy the customer at the click of a button. The payment system has been totally controlled by the technology of the Blockchain. A Blockchain is a growing list of records, called blocks, which are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data which is generally represented as a Merkle tree. The customers are provided with the wallet where they can store the digital money and pay the fee for car rental directly from the wallet. The block chain is very secure for the transaction and hence will provide the assured security for the transactions. Hence the car rental system is the booming business now in the major cities which can provide employment to many and also useful for the good economic private service to the customers. This car rental system can be linked with either frontend website or to a mobile application.

References

[1] https://www.tutorialspoint.com/blockchain/index.htm

[2] https://liblockchain.org/

[3] https://en.wikipedia.org/wiki/Blockchain_(database)

[4] https://www.coursehero.com/file/28521534/SSRN-id2662660pdf/

[5] https://www.coursehero.com/file/p4oj8qr/The-crypto-economy-is-an-economic-system-which-is-not-defined-by-geographic/

[6] https://towardsdatascience.com/blockchains-the-technology-of-transactions-9d40e8e41216

Appendix:

Code to create each block:

```
package com.nyit.carrental.wallet.domain;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import org.springframework.stereotype.Component;
import com.nyit.carrental.wallet.util.BlockChainUtils;
import com.nyit.carrental.wallet.util.Configuration;

public class Block {
    private long timeStamp; //Time of this block generated
    private String lastHash; //Hash of previous block
    private String hash; //Hash of this block
    private long nonce;
    private int difficulty; //Difficulty of generating hash
    private List<Transaction> transactions; //Transactions

    public static boolean minable;

    public Block(long timeStamp,List<Transaction> transactions, String lastHash) {
        this.timeStamp = timeStamp;
        this.transactions = transactions;
        this.lastHash = lastHash;
        this.nonce = 0;
        this.hash = calculateHash(timeStamp, String.valueOf(transactions),lastHash, nonce);
    }

    // TODO: consider the security of calling this method
    // TODO: how to put nonce and difficulty into this without hardcode but in configure way

    /**
     * Create genesis block
     * @return Block
     */
    public static Block genesis() {
        long timeStamp = new Date().getTime();
        String lastHash = "";
        ArrayList<Transaction> transactions = new ArrayList<>();
        return new Block(timeStamp, transactions,lastHash);
    }

    /**
     * Mining block
     * @param lastBlock
     * @param transactions
     * @return Block
     */
    public static Block mineBlock(Block lastBlock, List<Transaction> transactions) {
        long timeStamp = new Date().getTime();
        long nonce = 0;
        String lastHash = lastBlock.getHash();
        int difficulty = lastBlock.getDifficulty();
        String hash = "";
        while (hash.length() < difficulty || !hash.substring(0, difficulty).equals(new String(new
char[difficulty]).replace('\0', '0'))) {
            timeStamp = new Date().getTime();
            difficulty = Block.adjustDifficulty(lastBlock, timeStamp);
            hash = Block.calculateHash(timeStamp, String.valueOf(transactions),lastHash, nonce);
            nonce++;
            if (!minable) {
                return null;
            }
        }
        return new Block(timeStamp, transactions, lastHash);
    }

    /**
     * Adjust mining difficulty according to control the mining speed
     * @param lastBlock
     * @param currentTime
     * @return
     */
    //TODO: Fix the hardcoded mine rate
    public static int adjustDifficulty(Block lastBlock, long currentTime) {
        int difficulty = lastBlock.getDifficulty();
        difficulty = lastBlock.getTimeStamp() + Configuration.MINE_RATE > currentTime ? ++difficulty : --
difficulty;
        return difficulty;
    }

    /**
     * Calculate SHA256 of block
     * @param timeStamp
     * @param lastHash
     * @param difficulty
     * @param nonce
     * @param merkleRoot
     * @return String
     */
    public static String calculateHash(long timeStamp,String transactions, String lastHash, long nonce) {
        return BlockChainUtils.getSHA256Hash(Long.toString(timeStamp) +
                                                                                  transactions +
                                lastHash +
                                Long.toString(nonce));
    }


    @Override
    public String toString() {
        return "Block -" +
                "\n timeStamp: " + String.valueOf(timeStamp) +
                "\n lastHash: " + String.valueOf(lastHash) +
                "\n hash: " + String.valueOf(hash) +
                "\n difficulty: " + String.valueOf(difficulty) +
                "\n transactions: " + String.valueOf(transactions);
    }
}
```

Code to create links to Blockchain:

```
package com.nyit.carrental.wallet.domain;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class BlockChain {

    private List<Block> chain;

    /**
     * Create a new block chain with a genesis block
     * @return BlockChain
     */
    public static BlockChain newBlockchain() {
        List<Block> chain = new LinkedList<>();
        chain.add(Block.genesis());
        return new BlockChain(chain);
    }

    public Transaction findTransaction(String transactionId) {
        for(Block block : chain) {
            for(Transaction tx : block.getTransactions()) {
                if(tx.getTransactionId().equals(transactionId)) return tx;
            }
        }
        return null;
    }

    /**
     * Get all transactions contained in block chain
     * @param bcs
     * @return
     */
    public List<Transaction> getAllTransactions(Block bcs) {
        ArrayList<Transaction> transactions = new ArrayList<>();
        transactions.addAll(bcs.getTransactions());
        return transactions;
    }

    /**
     * Add a new mined block into block chain
     * @param transactions
     * @return Block
     */
    public Block addBlock(ArrayList<Transaction> transactions) {
        Block.minable = true;
        Block block = Block.mineBlock(chain.get(chain.size() - 1), transactions);
        if (block != null) {
            chain.add(block);
        }
        return block;
    }
    /**
     * Validate all blocks in this block chain
     * @param bc
     * @return
     */
    public boolean isValidChain(List<Block> bc) {
        for (int i = 1; i < bc.size(); i++) {
            Block curtBlock = bc.get(i);
            Block prevBlock = bc.get(i - 1);
            String curtHash = Block.calculateHash(curtBlock.getTimeStamp(),
String.valueOf(getAllTransactions(curtBlock)),curtBlock.getLastHash(),
                    curtBlock.getNonce());
            if (!curtBlock.getLastHash().equals(prevBlock.getHash()) || !curtBlock.getHash().equals(curtHash)) {
                return false;
            }
        }
        return true;
    }

    /**
     * replace current chain with new chain
     * @param newChain
     */
    public boolean replaceChain(List<Block> newChain) {
        if (newChain.size() < chain.size()) {
            return false;
        } else if (!isValidChain(newChain)) {
            return false;
        }
        chain = newChain;
        Block.minable = false;
        return true;
    }

    /**
     * Get the last block
     * @return Block
     */
    public Block getLastBlock() {
        return this.chain.get(chain.size() - 1);
    }

}
```