BLOCKCHAIN AUTHENTICATOR AND MACHINE LEARNING RECOMMENDER SYSTEM

A PROJECT REPORT

Submitted by

JASKIRAT SINGH BHATIA [Reg No: RA1511003010011]

Under the Guidance of

Dr. M. PRAKASH

(Associate Professor, Department of Computer Science and Engineering)

In Partial Fulfillment of the Requirements

for the Degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM UNIVERSITY, KATTANKULATHUR- 603 203

APRIL 2019

Department of Computer Science and Engineering
SRM Institute of Science & Technology
Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course	**: BTech / Computer Science Engineering(CSE)**
Student Name	**: Jaskirat Singh Bhatia**
Registration Number	**: RA1511003010011**
Title of Work	**: BlockChain Authenticator and Machine Learning Recommender System**

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalised in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

SRM UNIVERSITY, KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this thesis titled "**BLOCKCHAIN AUTHENTICATOR AND MCHINE LEARNING RECOMMENDER SYSTEM"** is the bonafide work of **Mr. JASKIRAT SINGH BHATIA AND Mr. ANOOP SINGH LABANA** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. M. PRAKASH	Dr. B. AMUTHA
SUPERVISOR	**HEAD OF THE DEPARTMENT**
Associate Professor	Department of Computer Science and Engineering
Department of Computer Science and Engineering	

ABSTRACT

The biggest concern in this era is trust; No one is willing to trust anyone. The latest trend in development of trust is the Blockchain technology and we now bring it into the medical branch. EHRs (Electronic Health Records) are the basis of storing the health records for the patients these days, but they are not decentralised and they can be modified or deleted by an authorised person or by a hacker easily. The base for this project is to give all the power to the patient by implementing the EHRs System on blockchain, making the records unmodifiable and not deletable show-ing transparency with whomsoever we want to share our records. The other part of our project is based on Machine Learning which focusses on predicting a disease based on the symptoms shown by the patient. We will be using SVM, Logistic Regression and Decision Tree algorithm to predict the disease and in the end, we will develop a voting system which takes the votes of the disease predicted by the 3 algorithms and select the majority. Then as per the disease predicted, a doctor for the same will be recommended using a recommender system.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Dr. M. PRAKASH his valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to me for completing this research work.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

EHR	Electronic Health Record
ML	Machine Learning
RSA	Rivest-Shamir-Adleman
UI	User Interface
API	Application Program Interface

CHAPTER 1

INTRODUCTION

1.1 Blockchain and Machine Learning Recommending System

Now a day's Blockchain and Machine Learning (ML), plays a great role in different fields or areas among the Health Care System. This leads to various studies and re-searches being conducted to selected health care facilities. It is necessary to ensure a technologically appropriate system designed to fully utilize the Blockchain and Ma-chine Learning technology for the maximum benefit in the health care society.

Here computers have great applicability on storing data's robustly and ease the access on them in brief burst of time.

In order to store the consumer information securely, a blockchain is being build. The blockchain is a robust, integrated, secure and trustworthy technology. The blockchain system encrypts the data stored in each layer with the data stored in the next layer, making it almost impossible to crack it without the key which brings us to the next technology in Blockchain that is the public/private key system. The public/private key of Blockchain works like any other system, that is all the hospitals have the public key of the patient and can use it to enter the details of the patient, and the patient will have his private key which he and only he can use to access his data, hence making it very secure.

ML using the algorithms of Deep Learning will be used to a high level of extent in our project. Deep Learning algorithms on clustering to figure out the type of disease the patient has and informing the patient about it. Also using Machine Learning Recom-mending algorithms to recommend the most suitable doctor for that particular disease nearest to the patient.

1.2 Problem Statement

Data is one thing in this world everyone is after. The ley man is afraid to give up his personal details into the world for privacy reasons. The lay man in this world is also very busy and doesn’t has enough time to spend on his health and take care of it properly. Thus, our aim is to develop a system which is secure and also one which saves the time of the user by predicting his disease from his symptoms and recommend the most suitable doctor accordingly.

CHAPTER 2

LITERATURE SURVEY

2.1 Literature Survey

Human services might be an immense application circumstance of blockchain, and blockchains used in help region unit known as wellbeing blockchain. As a rule, blockchain squares zone unit open and consequently the exchanges in them region unit open. On the off chance that some security information are engaged with these exchanges, they will be spilled. Inferable from help framework including a phenomenal arrangement of protection data, certain security components ought to be intended to shield this protection data in wellbeing blockchain. Besides, in light of the fact that the center of security components is that the key administration plots, the reasonable key oversee ment plans should be planned before blockchains might be used in help framework. Here, as per the highlights of wellbeing blockchain, the creators utilize a body identifier net-work to style a light-weight reinforcement and prudent recuperation topic for keys of wellbeing blockchain. The creator's examinations demonstrate that the plan has high security and perfor-mance, and it tends to be utilized to ensure protection messages on wellbeing blockchain adequately and to showcase the apparatus of wellbeing blockchain.

In view of the customary lexicon learning techniques, that ignores the connection be-tween the example and subsequently the wordbook iota, we tend to propose the weighted component to join the example with the wordbook molecule amid this dad per. Then, the customary word reference learning strategy is inclined to cause over-tting for patient classication of the constrained preparing informational collection. Subsequently, this paper embraces l2-standard regularization requirement, which understands the confinement of the model space, and upgrades the speculation capacity of the model and maintains a strategic distance from over-tting somewhat. Contrasted and the past shallow word reference learning, this paper proposed the insatiable profound lexicon learning.

In light of the conventional lexicon learning techniques, that ignores the connection be-tween the example and in this manner the wordbook molecule, we tend to propose

the weighted component to connect the example with the wordbook molecule amid this dad per. In the mean time, the customary word reference learning strategy is inclined to cause over-tting for patient classication of the restricted preparing informational index. Subsequently, this paper receives l2-standard regularization imperative, which understands the constraint of the model space, and improves the speculation capacity of the model and maintains a strategic distance from over-tting somewhat. Contrasted and the past shallow lexicon learning, this paper proposed the avaricious profound word reference learning.

Persistent similitude learning means to discover fitting separation measurements to quantify dad tient sets for a particular errand. To catch the verifiable data of patient' record, a legitimate method to speak to longitudinal EHR is important. Additionally, we need an approach to gain proficiency with the likeness degree or separation between each pair of patients. In this paper, we propose two patient comparability learning systems on EHR dataset. The crude EHRs are feed into a CNN model which catches the back to back consecutive data to get familiar with a vector portrayal. At that point delicate max based administered grouping strategy and triplet misfortune based separation metric learning technique are utilized to get familiar with the comparable ity of patient sets. Exploratory outcomes on sickness forecast and patient bunching demonstrate that CNN can more readily speak to the longitudinal EHR successions, and our endtoend closeness structures beat best in class separate measurement learning strategies.

The huge capability of this innovation shows up wherever, as of recently, a believed outsider was fundamental for the settlement of market administrations. With Blockchain, direct exchanges abruptly turned out to be conceivable, whereby a focal on-screen character, who controlled the information, earned commission or even interceded in a blue penciling design, can be killed. This troublesome character, which underlies Blockchain innovation, will firmly influence the level of influence between existing business sector players in human services. It will likewise advance new computerized plans of action and advanced wellbeing activities. Because of the way that, later on, (information) go-betweens can be kept away from, this innovation opens new entryways concerning how showcase communications in social insurance can be directed. Blockchain in this manner has an enormous potential for the future and will indicate troublesome changes in the social insurance industry.

2.2 Inference From The Survey

Blockchain and Machine Learning Recommending System Now a day’s Blockchain and ML, plays a big role in different fields or areas among the Health Care System. This leads to various studies being conducted to selected health care facilities. It is im-portant to ensure a technologically appropriate, affordable, efficient, and environmen-tally adaptable and consumer friendly system, designed to fully utilize the Blockchain and Machine Learning technology for the maximum benefit in the health care soci-ety. Here computers have great relevant on storing data’s securely and ease access on them in short period of time. In order to store the consumer information securely, a blockchain is being build. The blockchain is a robust, integrated, secure and trustwor-thy technology.

The blockchain system encrypts the data stored in each layer with the data stored in the next layer, making it almost impossible to crack it without the key which brings us to the next technology in Blockchain that is the public/private key sys-tem. The public/private key of Blockchain works like any other system, that is all the hospitals have the public key of the patient and can use it to enter the details of the patient, and the patient will have his private key which he and only he can use to access his data, hence making it very secure. Healthcare may be a huge application situation of blockchain, and blockchains utilized in aid area unit known as health blockchain. In general, blockchain blocks area unit open and therefore the transactions in them area unit public. If some privacy data are involved in these transactions, they will be leaked. Owing to aid system involving an excellent deal of privacy information, certain security mechanisms should be designed to shield this privacy information in health blockchain.

Furthermore, because the core of security mechanisms is that the key man-agement schemes, the suitable key management schemes ought to be designed before blockchains may be utilized in aid system. Here, according to the features of health blockchain, the authors use a body detector network to style a light- weight backup and economical recovery theme for keys of health blockchain.

CHAPTER 3

BLOCKCHAIN

A blockchain comprises of a lot of records called blocks where each block is directly linked with the block before it with by hash calculates by Secure Hash Algorithm. A block contains a tstamp, data in the transaction and the hash of the block before it. It is based on a peer to peer basis, that is when three or more PC's are interconnected and contribute resources without going through an independent server computer.

With the rapid development in technology, new inventions keep on developing and we must exploit the new technologies to do good for the world. The Blockchain tech-nology focuses on a distributed ledger, hence decentralising the data. The advantage of this is that if one of the ledgers is corrupted or hacked, we do not lose any data as rest of the ledgers support the uncorrupted data. Blockchain also encrypts the next block of the chain with its predecessor, making the data unmodifiable; the pro of this is that insurance companies would trust on the patient more as all his records would be orig-inal and showing transparency with the insurance companies. As the Health Record will be accessible by the patient unlike the existing system where the hospitals and the doctors access the records for the patient and each hospital has a separate record, hence providing a common record of the patient for any hospital.

3.1 Transaction

An electronic coin is termed as a group of digital signatures. Each holder, when he wants to transfer the coin to the next person, digitally signs the earlier transaction with the public ey of the later holder, adding all these to the end of the coin. The chain of ownership can be easily verified by the consignee which can be done by simply verifying the signatures affixed in the transaction

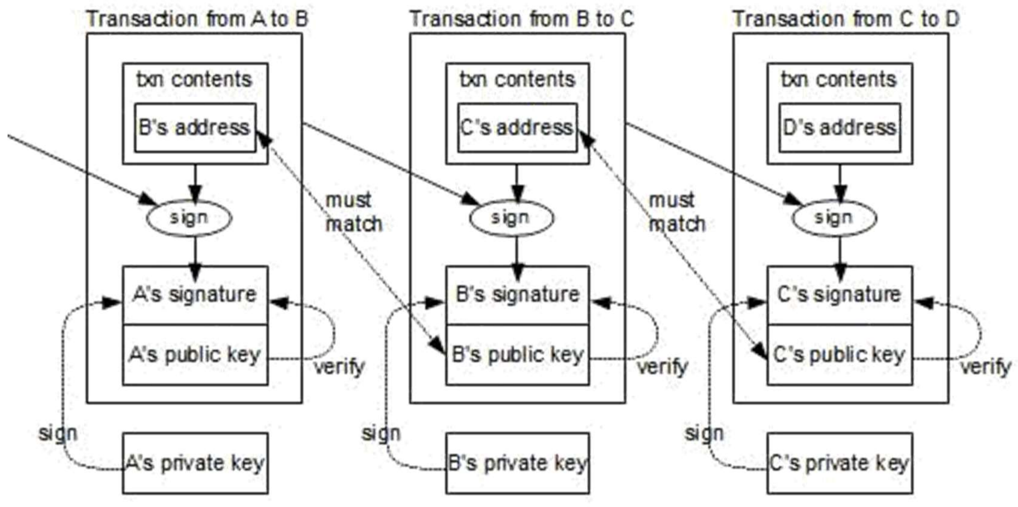


Figure 3.1: Transaction

3.2 Proof Of Work

The proof of work is used to figure out if the block has actually being mined or not. It comprises a value which when hashed with a hashing algorithm, the hash begins with a required minimum number of zeros. The median work required is the time required to mine a particular block which is the exponentials of the number of zeros mandatory and can be verified by executing a single hash. Once the amount of CPU power which needs to be a lot has been expended to make it satisfy the required proof-of-work, then the contents of the block cannot be changed without performing that very work again. As the blocks which arrive later are appended onto it, the work done to change the block would combine redoing all the work (or mining) blocks after it which sums up to be a lot.

Proof of work is also responsible for the blockchain to only continue in the original or unmodified chain. The decision of which chain to select s made by the longest chain; as that will be the chain with the maximum proof of work expended in it. If the majority of the nodes in the chain are honest nodes, the authentic chain will grow the most rapidly and will be definitely faster than any competing chains. To update a previous block, a hacker would have to mine the block he is attacking along with all blocks which are after it and then hook up with and surpass the work of all the honest nodes considering the fact that the honest nodes grow very rapidly; the probability of which is next to zero.

3.3 Transparency and Privacy

As Blockchain uses Public-Private key cryptography, each user has his own pair of keys. When a transaction is made, the user can share his public key and make his transaction visible to the person with whom he shared his key. If the user wishes privacy, he can choose to keep his public key anonymous and all the other people will notice that a transaction took place. Hence giving the user full power for his transparency or privacy.

CHAPTER 4

DISEASE PREDICTOR

Our Disease Predictor will take in various amounts of symptoms as input and predict a disease based on the symptoms. It will be based on three machine learning algorithms

: 1) Logistic Regression

2) Naive Bayes

3) Decision Tree Classification

4.1 Creating the Data-Set

We had a data-set w̆hich mapped each symptom to a disease along with its TF-IDF (Term Frequency âAS¸ Inverse Document Frequency) with 321 symptoms and 4219 diseases. So, in order to get the symptoms for a particular disease, we had to run a for loop on all the symptoms looking which one had the corresponding disease; and once we found out all the symptoms, we made a new dataset which had the symptoms for each disease.

Once we had created our dataset, it was time to convert it into a dataset which the machine could understand. The new dataset had the rows as all the diseases and the columns as all the symptoms. So if a particular disease had a particular symptom, it was marked as 1 and if not, it was marked as 0. In the end we had a dataset of dimensions: 4921 X 321 (diseases X symptoms).

4.2 Logistic Regression

Logistic regression can be used to anticipate the probability of any outcome which can only have two values(0 or 1, a or b). The anticipation is based on the usage of one or more predictors (they can be numerical or categorical). A linear regressor is not appropriate to anticipate the value of a two value variable for two reasons: A linear regressor can predict values which are outside the acceptance range and hence not advisable to use(e.g. predicting values outside the range of 0 to 100).

Since the operations can only have one of two possible values(0 or 1, a or b) for each operation, the residuals and results will not be normally or gausianlally distributed around the predicted line.

On the other hand, a logistic regression produces a logistic curve or a sigmoid curve, whose values are within 1 and 0. Logistic regression is somewhat similar to linear regression, but the curve is constructed/made using the natural logarithm or the sigmoid of the "odds" of the target variable, rather than the estimation. Moreover, the predictors don’t need to be gausianally distributed or normally or even have equal variances in each and every group.

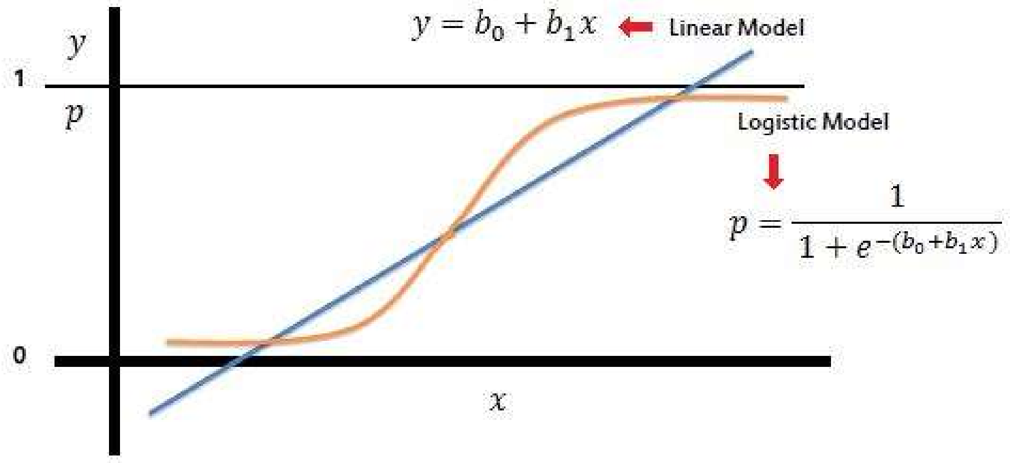
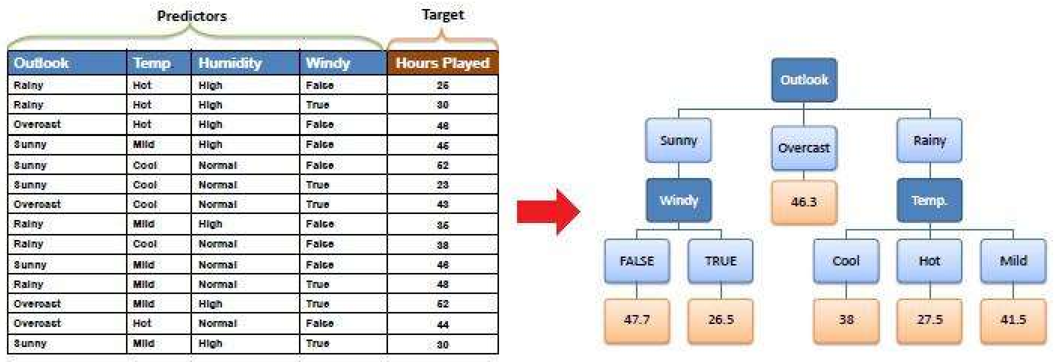


Figure 4.1: Logistic Regression

4.3 Decision Tree

Decision tree is used to create classification or regression models/algorithms in the fashion of a distributed tree structure. It divides a data-set into smaller and even smaller subsets wheras at that very time a correlated decision tree is developed which keeps on increasing as the nodes are developed. The final result is a decision tree with nodes and leaf leaves (decision nodes and nodes). A decision tree node (e.g., Outbook) can have more than two branches (e.g., Left, Right and Top), each representing corresponding values for the respective attribute tested. Leaf node (e.g., Time Played) representing a decision for the respective analytical target. The topmost/root node decision node generally corresponds to the best predictor in the tree is called the root node. Decision trees, generally can handle categorical data easily and in some cases even numerical data is handled easily.



Predictors				Target
Outlook	Temp	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30

Figure 4.2: Decision Tree

4.3.1 Algorithm

The core-deep design for building decision trees is called ID-3 by R. Quinlan which employs a top to down, greedy search algorithm through the area of possible branches without any kind of back-tracking. The ID-3 design can be efficiently used to construct a decision tree in a very small time for regression by replacing Standard Deviation Reduction with Information Gain.

4.4 Naive Bayes

The Naive Bayes is kind of a classifier which uses the famous Bayes Theorem and the Naive assumption. It predicts the participation estimations for each class. The class which has the highest estimation or possibility is considered to be as the best class or the most likely class. This is known as Maximum A Posteriori or (MAP). Map of the hypothesis is:
MAP (H)

= max (P(H|E))

= max ((P(E|H) * P(H)) / P(E))

= max (P(E|H) * P(H))

P(E) is probability of evidence, and it is usually used to normalize the result. It remains the same so, removing it doesn't really affect the output, but changes it slightly which is negligible. Naive Bayes classifier expects none of the features to be analogous to each other or one and another. Presence or absence of one or a few features does not influence the presence or absence of any other feature.

4.4.1 Example

A fruit may be treated to be an Orange if it is orange, round, and about 5" in diameter. Even if these features bank on another or even upon the inclusion of any other features, the naive Bayes classifier acknowledges all of these attributes to individually devote to the estimation that this particular fruit is an orange. In real datasets, we test a hypothesis given many evidences(features). So, calculations become a little complicated. To simplify the work, the feature to individualize approach is used to separate multiple evidence and treat all of them as an individual one.

P(H|Multiple Evidences) = P(E1| H)* P(E2|H)...*P(En|H) * P(H) / P(Multiple Evidences)

4.5 Support Vector Machine

A Support Vector Machine (SVM) is a classification algorithm performs classification by finding a hyperplane that maximizes the margins between two classes. Support Vectors are the vectors that are used to define the hyperplane. SVM is very effective in classifying datasets which have 2 or more categories. It generally takes more tome to train a dataset on an SVM algorithm, but the answer is very crisp and accurate.

4.5.1 Algorithm

1) Determine an optimal hyperplane to maximize the margin

2) Use the above definition for non-linearly separable problems that is have a

3) penalty class for misclassifications.

4) Map data to a higher dimensional space(if the data is not linearly separable) where it is relatively easier to classify with linear

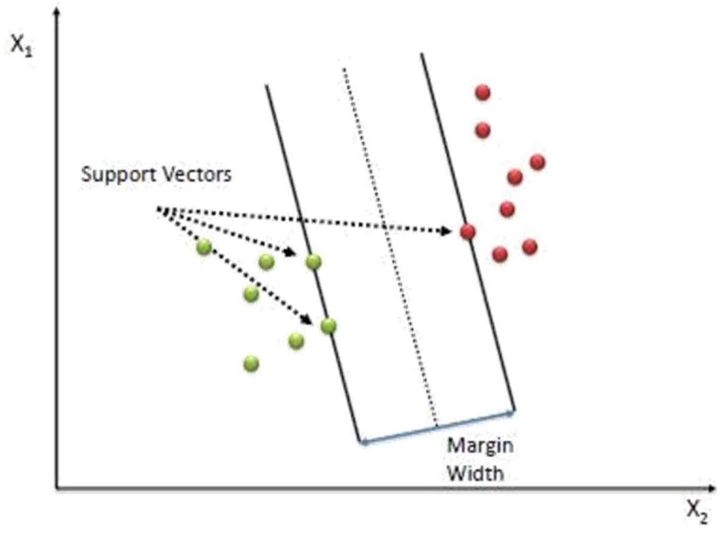


Figure 4.3: SVM

decision surfaces of the svm: redefine the problem so that data now explicitly maps to this space.

In the end, the data is separated linearly with a linear plane. The data can be classi-fied as 1) In the current category or 2) Not in current category.

4.6 Training and Testing Data

The dataset developed in is entirely used as the sole training data for out algorithms. This training data is fit in each of the algorithms separately and then the trained algo-rithm is pickled to save it. Pickling is used so that we do not have to train the algorithm again and again. And can be used directly from the hard disk of my PC which saves a lot of time. Each algorithm has its own pickle. When we want to load the data, we just read the pickle and load tit as our model which can then be used to predict diseases normally. Our testing data in this case was the same as our training data as an algorithm can not predict diseases which it doesn’t know already.

4.7 Final Result

Whenever the symptoms were input in our user interface, we first had to convert it into a format which our algorithm could read. So, we used binary just as we did in our dataset. If the symptom entered was in the list, it was entered as 1 and if it was not in the list, it was entered as 0. We had a list of 0s and 1s as a list of length 321 (number of symptoms). Then this data was used to get our predictions for each algorithm.

CHAPTER 5

RECOMMENDER SYSTEM

A recommender System will be used to recommend a doctor for the patient who is best suitable for that particular disease.

A recommender system or a recommendation system (sometimes "system" is replaced with a equivalent such as "platform" or "engine") is a tract of information percolating systems which seek to estimate the "rating" or "preference" a user would give to a peculiar aspect. Recommender systems mostly functions with two types of general information: Characteristic information: This is the information about items (categories, keywords, etc.) and users (profiles, preferences, etc.) which uniquely define each user. User - Item interactions: This is information such as likes, number of purchases, ratings, etc which are uniquely provided by each user. Based on the information above, we arrive at a first type of recommending systems: content-based, which uses idiosyncratic information, and the second type of recommendation systems: collaborative filtering, which is based on user - item interactions. The third type is a mix of both: Hybrid systems combine both types of knowledge with the aim of averting complications which are generated when working with just one kind.

5.1 Content Based Systems

These strategies make recommendations using a user's item and profile features. They hypothesize that if a user was interested in a particular item in the past, they will most probably be interested in it sometime in the future. Similar items are usually grouped together based on their feature - set. User profiles are generated using archival interactions or by explicitly questioning users about their enthusiasm. One kind of systems utilize the data from the user's personal or social life; they are not considered to be purely content - based. One issue which arises is making evident recommendations for a user because of exorbitant specialization (user 1 is only interested in categories 2, 3, and 4, and the system is unable to recommend items outside the range of those categories, even though those categories could not be really interesting to user 1). Another common problem which arises is that new users lack a detailed profile unless they are explicitly asked for more and more information. Nevertheless, it is relatively very simple to add new items or categories to the system. We just need to assure that we assign them a relevant group according to their feature-set.

5.2 Collaborative Filtering

These systems use some special kinds of techniques to study the user interaction with a particular group of items. A matrix can be used to easily visualize the set of interactions where each entry (i,j)(i, j)(i,j) is used to represent the interaction between user iii and item jjj respectively. A very interesting way we can gain insight on collaborative filtering is by not looking at it as a recommendation problem, but by considering it a generalization of classification and regression. In general cases we aim to predict a variable which is depended directly on other variables or features, whereas in collaborative filtering, distinctions such as feature variables or class variables do not exist.

5.3 Data-Set

We had successfully created a data set of all the doctors in Delhi. Our dataset comprises of 3000 doctors, their speciality, degrees and their addresses. After transforming the data-set into an excel file, we had used google Application Program Interface (API) to find the latitude and longitude of each doctors address so that it could be plotted on the map and also so that we could find the distance of the doctor from the user. And we had given a random rating (out of 10), and a random Cost per Meeting to each doctor.

5.4 Implementation

After finishing the dataset, we had created a user interface where the user would enter the speciality of the doctor he needed. Then our recommendation algorithm would filter out results as per the user requirements. The user could choose whether to filter each doctor by Rate, Rating or Distance. The resulting Doctors were then displayed on the map along with a table showing the information about each doctor.

CHAPTER 6

ENCRYPTION-DECRYPTION

Rivest-Shamir-Adleman (RSA) is a public - private key cryptosystem. RSA s based on the factoring problem that is the difficulty of factorizing the product of two very large prime numbers; making it almost impossible to crack. In RSA, two keys are generated that is the public and private key such that neither can be derived from one of them. The private key is kept a secret and no one else knows it but the user, on the other hand, the public key is known to anyone or everyone. During encryption, the public key is used to encrypt the document and the private key to decrypt it. For a Digital Signature, the private key is used to digitally signa document.

In our App, we have two different classes a Patient and a Doctor. The size of the keys of a doctor is 2048 bits and the size of the keys of a Patient is 2048*4 bits. The size of the doctor keys is small because he will be signing data which is relatively small in size whereas the Patient will be encrypting large sized data.

Whenever a patient or a doctor signs in, his public- private keys are generated, his private key is encrypted and both are stored in our database and can be accessed by the user upon login.

6.1 Implementation

The keys when generated, are stored in a database in a string format. Whenever we had to load a particular key set, The User ID was required to decrypt the private key of the user and the public key could be read directly from the database. Whenever a Doctor entered an EHR for a Patient, each block of the EHR was digitally signed by the private key of the user and then it was encrypted by the public key of the user. Then stored in the database. Whenever we wanted to retrieve data from the database to show on the user's screen, the data was first decrypted using the private key of the user, then the digital signature of the doctor was verified and if the signature was valid, it was displayed on the screen

CHAPTER 7

CONCLUSION

To conclude, We made a web User Interface (UI) using Flask (Python). The UI had a connection with a database with tables: 1)Patient 2)Doctor

3)Keys
4)EHR
5)Blockchain
Whenever a Doctor or a Patient Signed Up, A new entry was made in the respected table and a pair of keys were generated which were stored in the keys table. Whenever the Doctor created a new Electronic Health Record (EHR), it was stored in the EHR table and a new block in the blockchain was created linked to the previous block. Only the Doctor has access to this page. The Disease Predictor, which used Logistic Regression, Decision Tree, SVM and Naive Bayes was 87 % accurate. The Doctor Recommender System produced excellent results in recommending a doctor which could be sorted by distance, Rate or Rating.

The Problems of having all the records safe, encrypted, unmodifiable, undeletable were succesfully resolved by the blockchain and RSA encryption.

CHAPTER 8

CODE

8.1 about.html

```
{% extends "layout.html" %}
{% block content %}
  <div class="jumbotron jumbotron-fluid text-center" style="background: #FFFACD">
    <h1>Patient About Page</h1>
    <h3>Welcome To Patient About Page</h3>
 </div>
    <div class = "jumbotron jumbotron-fluid" style="background: #FFFACD">
        <h3>As a Patient, you can do several things like:</h3><br>
        <ul>
                <li>Securely store all your medical records on a BlockChain making all the
records stored undeletable and unmodifiable.</li>
                <li>Make use of our specialized Disease Predictor which uses Machine Learning
Algorims to detect which disease you have just by taking an input of your symptoms.</li>
                <li>If you're still not satisfied, we also recommend you a doctor according to your
convenience.</li>
                <li>We use RSA Public-Private key to encrypt and store all your data</li>
                <li>A big advantage of our Rabadiom is that the doctor who writes the record for
you digitally signs it. That means that the doctor can not deny the fact that he had given that
record.</li>
        </ul>
        <br>
        <h2 class= "text-center">SO BYE BYE FAKE RESULTS AND A CHAOTIC
LIFESTYLE</h2>
    </div>
{% endblock content %}
```

8.2 account.html

```
{% extends "layout.html" %}
{% block content %}

  <div class="content-section" style="background: #2F4F4F;color: #FFFACD">
    <div class="media">
      <img class="rounded-circle account-img" src="{{ image_file }}">
      <div class="media-body">
        <h2 class="account-heading" style="color: #FFFACD">{{ current_user.username }}</h2>
        <p class="text-secondary" style="color: #FFFACD">{{ current_user.email }}</p>
      </div>
    </div>
      <form method="POST" action="" enctype="multipart/form-data">
          {{ form.hidden_tag() }}
          <fieldset class="form-group">
              <legend class="border-bottom mb-4">Account Info</legend>
              <div class="form-group">
                  {{ form.username.label(class="form-control-label") }}

                  {% if form.username.errors %}
                      {{ form.username(class="form-control form-control-lg is-invalid") }}
                      <div class="invalid-feedback">
                          {% for error in form.username.errors %}
                              <span>{{ error }}</span>
                          {% endfor %}
                      </div>
                  {% else %}
                      {{ form.username(class="form-control form-control-lg") }}
                  {% endif %}
              </div>
              <div class="form-group">
                  {{ form.email.label(class="form-control-label") }}
                  {% if form.email.errors %}
                      {{ form.email(class="form-control form-control-lg is-invalid") }}
                      <div class="invalid-feedback">
                          {% for error in form.email.errors %}
```

```
                <span>{{ error }}</span>
              {% endfor %}
            </div>
          {% else %}
            {{ form.email(class="form-control form-control-lg") }}
          {% endif %}
        </div>
        <div class="form-group">
          {{ form.picture.label() }}
          {{ form.picture(class="form-control-file") }}
          {% if form.picture.errors %}
            {% for error in form.picture.errors %}
              <span class="text-danger">{{ error }}</span></br>
            {% endfor %}
          {% endif %}
        </div>
      </fieldset>
      <div class="form-group">
        {{ form.submit(class="btn btn-outline-info") }}
      </div>
    </form>
  </div>
{% endblock content %}
```

8.3 create_post.html

```
{% extends "doc_layout.html" %}
{% block content %}
<div class="content-section" style="background: #2F4F4F;color: #FFFACD">
  <form method="POST" action="">
    {{ form.hidden_tag() }}
    <fieldset class="form-group">
      <legend class="border-bottom mb-4">{{ legend }}</legend>
      <div class="form-group">
        {{ form.userid.label(class="form-control-label") }}
        {% if form.userid.errors %}
```

```
                {{ form.userid(class="form-control form-control-lg is-invalid") }}
                <div class="invalid-feedback">
                    {% for error in form.userid.errors %}
                        <span>{{ error }}</span>
                    {% endfor %}
                </div>
            {% else %}
                {{ form.userid(class="form-control form-control-lg") }}
            {% endif %}
        </div>
        <div class="form-group">
            {% for i in range(7) %}
                {{ form.test_or_med[i].label(class="form-control-label") }}


                {{ form.test_or_med[i](class="form-control form-control-lg") }}

                {{ form.causes[i].label(class="form-control-label") }}


                {{ form.causes[i](class="form-control form-control-lg") }}
            {% endfor %}
        </div>
        <div>
            {% for disease in form.diseases %}
                {{ disease.label(class="form-control-label") }}


                {{ disease(class="form-control form-control-lg") }}
            {% endfor %}
        </div>
    </fieldset>
    <div class="form-group">
        {{ form.submit(class="btn btn-outline-info") }}
    </div>
</form>
```

```
</div>
{% endblock content %}
```

8.4 layout.html

```
<!DOCTYPE html>
<html>
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

  <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='main.css') }}">

  {% if title %}
    <title>eClinic - {{ title }}</title>
  {% else %}
    <title>eClinic</title>
  {% endif %}
 <style>
  #map{
   height:400px;
   width:100%;
  }
 </style>
</head>
<body class="text-steel" style="background: #FFFACD">
  <header class="site-header">
```

```
   <nav class="navbar navbar-expand-md fixed-top" style="background:   #2F4F4F">
    <div class="container">
      <a class="navbar-brand mr-4" href="{{ url_for('main.home') }}"><span style="color: 
red">e+</span><span style="color: white">Clinic</span></a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarToggle" aria-controls="navbarToggle" aria-expanded="false" aria-label="Toggle 
navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarToggle">
        <div class="navbar-nav mr-auto">
          <a class="nav-item nav-link" href="{{ url_for('main.home') }}" style="color: 
#FFFACD">Home</a>
          <a class="nav-item nav-link" href="{{ url_for('main.about') }}" style="color: 
#FFFACD">About</a>
        </div>
        <!-- Navbar Right Side -->
        <div class="navbar-nav">
          {% if current_user.is_authenticated %}
            <a class="nav-item nav-link" href="{{ url_for('predictor.disease_predictor') }}" 
style="color: #FFFACD">Disease Predictor</a>
            <a class="nav-item nav-link" href="{{ url_for('recommend.doctor_recommender') }}" 
style="color: #FFFACD">Doctor Recommender</a>
            <a class="nav-item nav-link" href="{{ url_for('users.user_posts') }}" style="color: 
#FFFACD">My EHRs</a>
            <a class="nav-item nav-link" href="{{ url_for('users.account') }}" style="color: 
#FFFACD">Account</a>
            <a class="nav-item nav-link" href="{{ url_for('users.logout') }}" style="color: 
#FFFACD">Logout</a>
          {% else %}
            <a class="nav-item nav-link" href="{{ url_for('bchain.show_chain') }}" style="color: 
#FFFACD">BlockChain</a>
            <a class="nav-item nav-link" href="{{ url_for('doc_main.home') }}" style="color: 
#FFFACD">Switch To Doctor's Page</a>
            <a class="nav-item nav-link" href="{{ url_for('users.login') }}" style="color: 
#FFFACD">Login</a>
```

```
                <a class="nav-item nav-link" href="{{ url_for('users.register') }}" style="color:
#FFFACD">Register</a>
              {% endif %}
            </div>
          </div>
        </div>
      </nav>
    </header>
    <main role="main" class="container">
      <div class="row">
        <div class="col-md-8">
          {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
              {% for category, message in messages %}
                <div class="alert alert-{{ category }}">
                  {{ message }}
                </div>
              {% endfor %}
            {% endif %}
          {% endwith %}
          {% block content %}{% endblock %}
          <div class="jumbotron jumbotron-fluid" style="background: #FFFACD">
            <h2 class="text-center">WELCOME TO THE FUTURE</h2>
            <h1 class="text-center" style="text-shadow: 3px 2px grey"><span style="color: red;font-
size: 60px">e</span><span style="color: red;font-size: 50px">+</span><span style="color:
#2F4F4F">CLINIC </span></h1>
          </div>
        </div>
        {% if current_user.is_authenticated %}
          <div class="col-md-4">
            <div class="content-section" style="background: #2F4F4F">
              <h3 style="color: #FFFACD">Currently Logged In as A Patient</h3>
              <p class='text-muted'>Your General Information.
                <ul class="list-group">
                  <li class="list-group-item" style="background: #FFFACD">Name: {{
current_user.name }}</li>
```

```
                <li class="list-group-item" style="background: #FFFACD">Username: {{
current_user.username }}</li>
                <li class="list-group-item" style="background: #FFFACD">Email: {{
current_user.email }}</li>
              </ul>
            </p>
          </div>
        </div>
      {% else %}
        <div class="col-md-4">
          <div class="content-section" style="background: #2F4F4F">
            <h3 style="color: #FFFACD">Currently not Logged In</h3>
            <p class='text-muted'>Login to get all the benefits of our App.
              <ul class="list-group">
                <li class="list-group-item" style="background: #FFFACD">Saving EHRs on a
BlockChain</li>
                <li class="list-group-item" style="background: #FFFACD">Predicting Diseases</li>
                <li class="list-group-item" style="background: #FFFACD">High Level Security</li>
                <li class="list-group-item" style="background: #FFFACD">Doctor
Reccomendations</li>
              </ul>
            </p>
          </div>
        </div>
      {% endif %}
    </div>
  </main>



  <!-- Optional JavaScript -->
  <!-- jQuery first, then Popper.js, then Bootstrap JS -->
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-
```

```
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
</body>
</html>
```

8.5 Show Map.html

```
{% extends "layout.html" %}
{% block content %}

<div id="map"></div>
 <script>
  function initMap(){
   // Map options
   var options = {
    zoom:11,
    center:{lat:{{ lat }},lng:{{ lng }}}
   }

   // New map
   var map = new google.maps.Map(document.getElementById('map'), options);

  var markers = [
    {
     coords:{ lat: {{ lat }}, lng:{{ lng }} },
     content:'<h5>{{ current_user }}</h5>'
    },
    {
     coords:{ lat: {{ doctors[0]["lat"] }}, lng:{{ doctors[0]["lng"] }} },
iconImage:'https://developers.google.com/maps/documentation/javascript/examples/full/ima
ges/beachflag.png',
          content:'<h5>{{ doctors[0]["Name"] }} <br>Rate: {{ doctors[0]["Rate"] }} <br>Rating: {{
doctors[0]["Rating"] }}</h5>'
        },
        {
          coords:{ lat: {{ doctors[1]["lat"] }}, lng:{{ doctors[1]["lng"] }} },

iconImage:'https://developers.google.com/maps/documentation/javascript/examples/full/ima
ges/beachflag.png',
          content:'<h5>{{ doctors[1]["Name"] }} <br>Rate: {{ doctors[1]["Rate"] }} <br>Rating: {{
doctors[1]["Rating"] }}</h5>'
        },
        {
          coords:{ lat: {{ doctors[2]["lat"] }}, lng:{{ doctors[2]["lng"] }} },

iconImage:'https://developers.google.com/maps/documentation/javascript/examples/full/ima
ges/beachflag.png',
          content:'<h5>{{ doctors[2]["Name"] }} <br>Rate: {{ doctors[2]["Rate"] }} <br>Rating: {{
doctors[2]["Rating"] }}</h5>'
        },
        {
          coords:{ lat: {{ doctors[3]["lat"] }}, lng:{{ doctors[3]["lng"] }} },

iconImage:'https://developers.google.com/maps/documentation/javascript/examples/full/ima
ges/beachflag.png',
          content:'<h5>{{ doctors[3]["Name"] }} <br>Rate: {{ doctors[3]["Rate"] }} <br>Rating: {{
doctors[3]["Rating"] }}</h5>'
        },
        {
          coords:{ lat: {{ doctors[4]["lat"] }}, lng:{{ doctors[4]["lng"] }} },
iconImage:'https://developers.google.com/maps/documentation/javascript/examples/full/ima
ges/beachflag.png',
          content:'<h5>{{ doctors[4]["Name"] }} <br>Rate: {{ doctors[4]["Rate"] }} <br>Rating: {{
doctors[4]["Rating"] }}</h5>'
        }
        ];

        for(var i = 0;i < markers.length;i++){
        // Add marker
        addMarker(markers[i]);
      }

      // Add Marker Function
      function addMarker(props){
        var marker = new google.maps.Marker({
          position:props.coords,
          map:map,
          //icon:props.iconImage
        });

        // Check for customicon
        if(props.iconImage){
          // Set icon image
          marker.setIcon(props.iconImage);
        }

        // Check content
        if(props.content){
          var infoWindow = new google.maps.InfoWindow({
            content:props.content
          });

          marker.addListener('click', function(){
```

```
          infoWindow.open(map, marker);
        });
      }
    }
  }
</script>
<script async defer

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCukArbjvhnW2JjGBedKNUR5fBf
bY2KzWg&callback=initMap">
  </script>
  <br>
  <br>
        <div class="container">
          <h2>Doctor Table</h2>
          <p>Table which shows the doctors recommended</p>
          <table class="table table-striped" style="background: #2F4F4F;color:
#FFFACD">
            <thead>
              <tr>
                <th>Name</th>
                <th>Rate</th>
                <th>Rating</th>
                {% if doctors[0]["distance"] %}
                  <th>Distance</th>
                {% endif %}
              </tr>
            </thead>
            <tbody>
              {% for d in doctors %}
              <tr>
                <td> {{ d["Name"] }} </td>
                <td> {{ d["Rate"] }} </td>
                <td> {{ d["Rating"] }} </td>
                {% if d["distance"] %}
                  <td>{{ d["distance"] }}</td>
                {% endif %}
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>

{% endblock content %}
```

8.5 Show Diseases.html

```
{% extends "layout.html" %}
{% block content %}
        {% if diseases %}
        <div class="container">
          <h2>Disease Table</h2>
          <p>Table which shows the diseases predicted</p>
          <table class="table table-striped" style="background: #2F4F4F;color:
#FFFACD">
            <thead>
              <tr>
                <th>Disease</th>
                <th>Algorith Used</th>
              </tr>
            </thead>
            <tbody>
              {% for d in diseases %}
              <tr>
                <td> {{ d[0] }} </td>
                <td> {{ d[1] }} </td>
              </tr>
              {% endfor %}
```

```
                </tbody>
              </table>
            </div>
            {% else %}
            <div class="jumbotron jumbotron-fluid">
                <h1>No Symptom Entered</h1>
                <h3>Please Go Back And Enter Some Symptoms</h3>
            </div>
            {% endif %}
            <div>
    <button class="btn" style="background:   #2F4F4F"><a class="nav-item nav-link"
href="{{ url_for('recommend.doctor_recommender') }}" style="color: #FFFACD">Doctor
Recommender</a></button>
   </div>
{% endblock content %}
```

BLOCKCHAIN

8.6 Routes

```
from flask import render_template, Blueprint, flash, request

from rabadiom.blockchain.utils import BlockChain, Patient, Doctor, Transaction, EHR, Block
from rabadiom.models import Blockchain, User, Keys, Ehr

from rabadiom.blockchain import blockchain
#blockchain = BlockChain()

bchain = Blueprint('bchain', __name__)



@bchain.route("/BlockChain", methods=['GET', 'POST'])
def show_chain():
#    with app.app_context():
        rows = Blockchain.query.all()
```

```
    chain = []
    if rows:
        #chain = []
        for row in rows:
            user = User.query.filter_by(id=row.user_id).first()
            doc = User.query.filter_by(id=row.doctor_id).first()

            pat_keys = Keys.query.filter_by(user_id=user.id).first()
            pat_keys = pat_keys.private_key
            doc_keys = Keys.query.filter_by(user_id=doc.id).first()
            doc_keys = doc_keys.private_key
            patient = Patient(name=user.name, private_key=pat_keys)
            doctor = Doctor(name=doc.name, private_key=doc_keys)
            ehr = Ehr.query.filter_by(id=row.node).first()
            data = ""
            data += ehr.diseases1 + ehr.diseases2 + ehr.diseases3 + ehr.test_or_med1 +
ehr.test_or_med2 + ehr.test_or_med3 + ehr.test_or_med4 + ehr.test_or_med5 +
ehr.test_or_med6 + ehr.test_or_med7 + ehr.causes1 + ehr.causes2 + ehr.causes3 +
ehr.causes4 + ehr.causes5 + ehr.causes6 + ehr.causes7
            ehr = EHR(patient=user, doctor=doc, data=data)
            transaction = Transaction(patient=user, doctor=doctor, ehr=ehr.ToHash())
            block = Block(transaction=transaction, hash=row.hash,
prev_hash=row.prev_hash, nonce=row.nonce, tstamp=row.tstamp)
            #blockchain.newBlock(Block)
            #if blockchain.isChainValid(chain):
            chain.append(block)

    #if blockchain.isChainValid(chain):
    #  flash("Chain is Valid", "success")
    #  return render_template('show_chain.html', title='BlockChain', chain = chain)
    #else:
    #  flash("Chain is Invalid", "danger")
    #  return render_template('show_chain.html', title='BlockChain', chain = chain)
    last_block = chain[0]
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]
        # Check that the hash of the block is correct
        last_block_hash = last_block.hash
        if block.prev_hash != last_block_hash:
            flash("prev hash != last block hash" + str(current_index))
            flash("Chain is invalid","danger")
            return render_template('show_chain.html', title='BlockChain', chain = chain)
        last_block = block
        current_index += 1
    flash("Chain is valid", "success")
    return render_template('show_chain.html', title='BlockChain', chain = chain)
```

8.7 Utils.py

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes

from flask import flash

import hashlib
import json

from flask import Flask, jsonify, request
import requests
from time import time
from urllib.parse import urlparse
from uuid import uuid4

from datetime import datetime
import collections



class Patient:
    def __init__(self, name, private_key=""):
        if private_key == "":
            self.private_key = rsa.generate_private_key(
                public_exponent=65537,
                key_size=2048 * 4,
                backend=default_backend()
                )
        else:
            self.private_key = private_key = serialization.load_pem_private_key(
                                                                private_key,
                                                                password=None,
                                                                backend=default_backend())

        self.public_key = self.private_key.public_key()

        self.name = name



    def Encrypt(self, message):
        ciphertext = self.public_key.encrypt(
        message,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
            )
        )
        return ciphertext



    def Decrypt(self, ciphertext):
        plaintext = self.private_key.decrypt(
        ciphertext,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
            )
        )
        return plaintext

    def RetName(self):
        return self.name



class Doctor:
    def __init__(self, name, private_key=""):
        self.name = name

        if private_key == "":
            self.private_key = rsa.generate_private_key(
                public_exponent=65537,
                key_size=2048,
                backend=default_backend()
                )

        else:
            self.private_key = private_key = serialization.load_pem_private_key(
                                                                private_key,
                                                                password=None,
                                                                backend=default_backend())
        self.public_key = self.private_key.public_key()

    def Sign(self, message):
        message = bytes(message, "utf-8")
        signature = self.private_key.sign(
        message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
            ),
        hashes.SHA256()
        )
        return signature

    def Verify(self, message, signature):
        message = bytes(message, "utf-8")
        try:
            self.public_key.verify(
            signature,
            message,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
                ),
            hashes.SHA256()
            )
            return True
        except:
            print("invalid signature")
            return False

    def RetName(self):
        return self.name
```

```
class EHR:
    def __init__(self, doctor = Doctor, patient = Patient, data = ""):
        self.patient = patient
        self.doctor = doctor
        self.data = data


    def ToDict(self):
        dict1 = {"doctor": self.doctor.RetName(), "patient": self.patient.RetName(), "sign":
self.sign}
        return dict1

    def ToHash(self):
        block_string = json.dumps({"patient":self.patient, "doctor": self.doctor, "data":self.data},
default=str).encode()
        return hashlib.sha256(block_string).hexdigest()




class Transaction:
    def __init__(self, doctor = Doctor, patient = Patient, ehr = ""):
        self.doctor = doctor
        self.patient = patient
        self.ehr = ehr
    def ToDict(self):
        transaction = {"doctor": self.doctor.name, "patient": self.patient.name, "ehr": self.ehr}
        return transaction



class Block:
    def __init__(self, nonce = 0, tstamp = None, prev_hash = None, hash = None, transaction
= Transaction):
        self.nonce = nonce
        self.tstamp = tstamp
        self.prev_hash = prev_hash
        self.transaction = transaction
        if hash == None:
            self.hash = self.calcHash()
        else:
            self.hash = hash
    def calcHash(self):
        block_string = json.dumps({"nonce":self.nonce, "tstamp":self.tstamp,
"transaction":self.transaction.ToDict(),
                                 "prev_hash":self.prev_hash}, default=str).encode()
        return hashlib.sha256(block_string).hexdigest()

    def mine(self, difficulty = 4):
        compare = "0" * difficulty
        self.hash = self.calcHash()
        while str(self.hash[:difficulty]) != compare:
            self.nonce += 1
            self.hash = self.calcHash()
        return self.hash

    def ToDict(self):
        block = {"nonce":self.nonce,
                 "tstamp":self.tstamp,
                 "transaction":self.transaction.ToDict(),
                 "prev_hash":self.prev_hash,
                 "hash":self.hash}
        return block
```

```
class BlockChain:
    def __init__(self):
        self. pending_transactions = []
        self.chain = []
        self.generateGenesisBlock()
        self.difficulty = 4
        self.reward = 100
        self.nodes = set()



    def isChainValid(self, chain):
        """
        Determine if a given blockchain is valid
        :param chain: A blockchain
        :return: True if valid, False if not
        """

        last_block = chain[0]
        current_index = 1

        while current_index < len(chain):
            block = chain[current_index]
            # Check that the hash of the block is correct
            last_block_hash = last_block.calcHash()
            if block.prev_hash != last_block_hash:
                flash("prev hash != last block hash")
                return False

            # Check that the Proof of Work is correct
            if str(last_block.calcHash()[:self.difficulty]) != "0" * self.difficulty:
                print("proof not valid")
                return False

            last_block = block
            current_index += 1

        return True




    def replaceChain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)
        for node in network:
            response = requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']
                bchain = []
                for temp in chain:
                    doctor = Doctor(name = temp["transaction"]["doctor"])
                    patient = Patient(name = temp["transaction"]["patient"])
                    ehr = temp["transaction"]["ehr"]
                    transaction = Transaction(doctor, patient, ehr)
                    block = Block(hash = temp["hash"], nonce = temp["nonce"], prev_hash =
temp["prev_hash"], transaction = transaction, tstamp = temp["tstamp"])
                    bchain.append(block)
                if length > max_length and self.isChainValid(bchain):
                    max_length = length
                    longest_chain = bchain
        if longest_chain:
            self.chain = longest_chain
            return True
        return False

    def add_node(self, address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)









    def PrintChain(self):
        for block in self.chain:
            pprint.pprint(block.ToDict())

    def newBlock(self, block = Block):
        #if block.hash == None:

        #if block.prev_hash == None:
        block.prev_hash = self.chain[-1].hash
        #if block.tstamp == None:
        block.tstamp = datetime.utcnow()
        block.mine()
        #else:
        #block.tstamp = tstamp
        self.chain.append(block)
        #transaction = Transaction(patient=miner_address, doctor="Genesis", ehr="Genesis")
        #self.pending_transactions.append(transaction)
        return block

    def generateGenesisBlock(self):
        doctor = Doctor(name = "Genesis")
        patient = Patient(name = "Genesis")
        ehr = EHR(doctor, patient, "Genesis")
        transaction = Transaction(doctor=doctor, patient=patient, ehr=ehr.ToHash())
        block = Block(transaction=transaction)
        block.mine()
        self.chain.append(block)
    def FinishPendingTransactions(self):
        pass



    @property
    def last_block(self):
        return self.chain[-1]



    def valid_proof(block = Block):
        """
        Validates the Proof
        :param last_proof: <int> Previous Proof
        :param proof: <int> Current Proof
        :param last_hash: <str> The hash of the Previous Block
        :return: <bool> True if correct, False if not.
        """

        guess_hash = block.calcHash()
        return str(guess_hash[:4]) == "0000"

def GetPrivateKey(key):
    private_key = key.private_bytes(
                            encoding=serialization.Encoding.PEM,
                            format=serialization.PrivateFormat.TraditionalOpenSSL,
                            encryption_algorithm = serialization.NoEncryption())
    return private_key

def GetPublicKey(key):
    public_key = key.public_bytes(
                        encoding=serialization.Encoding.PEM,
                        format=serialization.PublicFormat.SubjectPublicKeyInfo)
    return public_key
```

DISEASE PREDICTOR

8.8 forms.py

```
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed
from wtforms import StringField, PasswordField, SubmitField, BooleanField, FieldList,
SelectField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from rabadiom.disease_predictor.utils import unique_symptoms


class SymptomForm(FlaskForm):
    symptom1 = SelectField('Symptom - 1', choices=unique_symptoms)
    symptom2 = SelectField('Symptom - 2', choices=unique_symptoms)
    symptom3 = SelectField('Symptom - 3', choices=unique_symptoms)
    symptom4 = SelectField('Symptom - 4', choices=unique_symptoms)
    symptom5 = SelectField('Symptom - 5', choices=unique_symptoms)
    symptom6 = SelectField('Symptom - 6', choices=unique_symptoms)
    submit = SubmitField('Submit')

    def validate_symptom1(self, idd):
      if idd.data == None:
        raise ValidationError("First Symptom can not be Empty")

class SymptomFinalForm(FlaskForm):
  #symptoms = StringField[]
  #symptom = StringField('Symptom')
  #symptoms.append(symptom)

  symptom = FieldList(StringField('Symptom'), min_entries=6, max_entries=6)
  add_symptom = SubmitField("Add another Symptom")
  submit = SubmitField("Submit")
```

8.9 Routes.py

```
from flask import render_template, Blueprint, flash, request
from rabadiom.disease_predictor.forms import SymptomForm, SymptomFinalForm
from flask_login import login_user, current_user, logout_user, login_required
from rabadiom import login_manager
from rabadiom.disease_predictor.utils import predict_disease, predict_using_ml
from wtforms import StringField
from wtforms.validators import DataRequired
from functools import wraps
import math



def login_required(role):
    def wrapper(fn):
        @wraps(fn)
        def decorated_view(*args, **kwargs):
            if not current_user.is_authenticated:
              return login_manager.unauthorized()
            if (current_user.role != role):
                return login_manager.unauthorized()
            return fn(*args, **kwargs)
        return decorated_view
    return wrapper



predictor = Blueprint('predictor', __name__)

@predictor.route("/Disease Predictor", methods=['GET', 'POST'])
@login_required("User")
def disease_predictor():
        form = SymptomForm()
        if form.validate_on_submit():
                if form.symptom1.data == "Select":
                    return render_template('show_diseases.html', title='Disease Predictor
Result')
                symptoms = [form.symptom1.data, form.symptom2.data,
form.symptom3.data, form.symptom4.data, form.symptom5.data, form.symptom6.data]
                diseases = predict_using_ml(symptoms)
                #diseases = ',\r\r\n'.join(["disease :" + d[0] + " \nscore :" + str(math.ceil(d[1]))
for d in diseases])
                #flash(diseases,"success")
                return render_template('show_diseases.html', title='Disease Predictor Result',
diseases=diseases)
        elif request.method == "GET":
                form.symptom1.data = "None"
                form.symptom2.data = "None"
                form.symptom3.data = "None"
                form.symptom4.data = "None"
                form.symptom5.data = "None"
                form.symptom6.data = "None"
        return render_template('disease_predictor.html', title='Disease Predictor', form=form)
```

8.11 utils.py

```
from flask import render_template, Blueprint, flash, request
from rabadiom.disease_predictor.forms import SymptomForm, SymptomFinalForm
from flask_login import login_user, current_user, logout_user, login_required
from rabadiom import login_manager
from rabadiom.disease_predictor.utils import predict_disease, predict_using_ml
from wtforms import StringField
from wtforms.validators import DataRequired
from functools import wraps
import math


def login_required(role):
    def wrapper(fn):
        @wraps(fn)
        def decorated_view(*args, **kwargs):
            if not current_user.is_authenticated:
              return login_manager.unauthorized()
            if (current_user.role != role):
                return login_manager.unauthorized()
            return fn(*args, **kwargs)
        return decorated_view
    return wrapper



predictor = Blueprint('predictor', __name__)


@predictor.route("/Disease Predictor", methods=['GET', 'POST'])
@login_required("User")
def disease_predictor():
            form = SymptomForm()
            if form.validate_on_submit():
                        if form.symptom1.data == "Select":
                                    return render_template('show_diseases.html', title='Disease Predictor
Result')
                        symptoms = [form.symptom1.data, form.symptom2.data,
form.symptom3.data, form.symptom4.data, form.symptom5.data, form.symptom6.data]
                        diseases = predict_using_ml(symptoms)
                        #diseases = ',\r\r\n'.join(["disease :" + d[0] + " \nscore :" + str(math.ceil(d[1]))
for d in diseases])
                        #flash(diseases,"success")
                        return render_template('show_diseases.html', title='Disease Predictor Result',
diseases=diseases)
            elif request.method == "GET":
                        form.symptom1.data = "None"
                        form.symptom2.data = "None"
                        form.symptom3.data = "None"
                        form.symptom4.data = "None"
                        form.symptom5.data = "None"
                        form.symptom6.data = "None"
            return render_template('disease_predictor.html', title='Disease Predictor', form=form)
```

MAIN

8.12 Routes.py

```
from flask import render_template, request, Blueprint
from rabadiom.models import Post
from flask_login import login_user, current_user, logout_user, login_required

main = Blueprint('main', __name__)



@main.route("/")
@main.route("/Patient/home")
def home():
    page = request.args.get('page', 1, type=int)
    posts = Post.query.order_by(Post.date_posted.desc()).paginate(page=page, per_page=5)
    return render_template('home.html', posts=posts)



@main.route("/Patient/about")
def about():
    return render_template('about.html', title='About')
```

POSTS

8.13 Routes.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, TextAreaField, FieldList
from wtforms.validators import DataRequired, ValidationError
from rabadiom.models import User



class PostForm(FlaskForm):

    userid = StringField('User ID', validators=[DataRequired()])
    diseases = FieldList(StringField('Probable Disease'), min_entries=3, max_entries=3)
    test_or_med = FieldList(StringField('Suggested Medicine or Test'), min_entries=7,
max_entries=7)
    causes = FieldList(StringField('Why?'), min_entries=7, max_entries=7)

    submit = SubmitField('Post')


    def validate_userid(self, userid):
        user = User.query.filter_by(username=userid.data).first()
        if not user:
            raise ValidationError('No Such User Exists')
        elif user.role == "Doctor":
            raise ValidationError('The ID Provided is the ID of a Doctor')
```

8.14 forms.py

```
from flask import (render_template, url_for, flash,
                   redirect, request, abort, Blueprint)
from flask_login import current_user, login_required
from rabadiom import db, login_manager, bcrypt
from rabadiom.models import Post, User, Ehr, Keys, Blockchain, SignedEhr
from rabadiom.posts.forms import PostForm
from functools import wraps
from rabadiom.blockchain.utils import Patient, Doctor, Block, EHR, Transaction
from rabadiom.blockchain import blockchain
```

```
def login_is_required(role = "ANY"):
    def wrapper(fn):
        @wraps(fn)
        def decorated_view(*args, **kwargs):
            if not current_user.is_authenticated:
              return login_manager.unauthorized()
            if ( (current_user.role != role) and (role != "ANY")):
                return login_manager.unauthorized()
            return fn(*args, **kwargs)
        return decorated_view
    return wrapper

posts = Blueprint('posts', __name__)



@posts.route("/post/new", methods=['GET', 'POST'])
@login_is_required("Doctor")
def new_post():
    form = PostForm()
    if form.validate_on_submit():
        if blockchain.isChainValid(blockchain.chain):
            user = User.query.filter_by(username=form.userid.data).first()
            post = Ehr(doctor_id=current_user.id, user_id=int(user.id), diseases1 =
form.diseases[0].data, diseases2 = form.diseases[1].data, diseases3 =
form.diseases[2].data,
                        test_or_med1 = form.test_or_med[0].data, test_or_med2 =
form.test_or_med[1].data, test_or_med3 = form.test_or_med[2].data,
                        test_or_med4 = form.test_or_med[3].data, test_or_med5 =
form.test_or_med[4].data, test_or_med6 = form.test_or_med[5].data,
                        test_or_med7 = form.test_or_med[6].data,
                        causes1 = form.causes[0].data, causes2 = form.causes[1].data, causes3 =
form.causes[2].data,
                        causes4 = form.causes[3].data, causes5 = form.causes[4].data, causes6 =
form.causes[5].data,
                        causes7 = form.causes[6].data)
            ehr = post
            doc_keys = Keys.query.filter_by(user_id=current_user.id).first()
            doc_keys = doc_keys.private_key
            pat_keys = Keys.query.filter_by(user_id=user.id).first()
            pat_keys = pat_keys.private_key
            patient = Patient(name=user.name, private_key=pat_keys)
            doctor = Doctor(name=current_user.name, private_key=doc_keys)
            data = ""
            data += ehr.diseases1 + ehr.diseases2 + ehr.diseases3 + ehr.test_or_med1 +
ehr.test_or_med2 + ehr.test_or_med3 + ehr.test_or_med4 + ehr.test_or_med5 +
ehr.test_or_med6 + ehr.test_or_med7 + ehr.causes1 + ehr.causes2 + ehr.causes3 +
ehr.causes4 + ehr.causes5 + ehr.causes6 + ehr.causes7
            ehr = EHR(patient=patient, doctor=doctor, data=data)
            transaction = Transaction(patient=patient, doctor=doctor, ehr=ehr.ToHash())
            block = Block(transaction=transaction)
            block = blockchain.newBlock(block)
            try:
                chain_block = Blockchain.query.order_by(Blockchain.node.desc()).first()
                if len(blockchain.chain) == 2:
                    chain_user = User.query.filter_by(id=chain_user.user_id).first()
                    chain_doc = User.query.filter_by(id=chain_doc.user_id).first()
                    doc_keys = Keys.query.filter_by(user_id=chain_doc.id).first()
                    doc_keys = doc_keys.private_key
                    pat_keys = Keys.query.filter_by(user_id=chain_user.id).first()
                    pat_keys = pat_keys.private_key
                    patient = Patient(name=chain_user.name, private_key=pat_keys)
                    doctor = Doctor(name=chain_doc.name, private_key=doc_keys)
                    transaction = Transaction(patient=patient, doctor=doctor, ehr=chain_block.ehr)
                    block = Block(transaction=transaction)
                    blockchain.chain[0] = block
                block.prev_hash = chain_block.hash
            except:
                pass
            db_block = Blockchain(user_id=user.id, doctor_id=current_user.id,
ehr=ehr.ToHash(), hash=block.hash,
                prev_hash=block.prev_hash, nonce=block.nonce, tstamp=block.tstamp)
            diseases1 = doctor.Sign(form.diseases[0].data)
            diseases2 = doctor.Sign(form.diseases[1].data)
            diseases3 = doctor.Sign(form.diseases[2].data)
            test_or_med1 = doctor.Sign(form.test_or_med[0].data)
            causes1 = doctor.Sign(form.causes[0].data)
            test_or_med2 = doctor.Sign(form.test_or_med[1].data)
            causes2 = doctor.Sign(form.causes[1].data)
            test_or_med3 = doctor.Sign(form.test_or_med[2].data)
            causes3 = doctor.Sign(form.causes[2].data)
            test_or_med4 = doctor.Sign(form.test_or_med[3].data)
            causes4 = doctor.Sign(form.causes[3].data)
            test_or_med5 = doctor.Sign(form.test_or_med[4].data)
            causes5 = doctor.Sign(form.causes[4].data)
            test_or_med6 = doctor.Sign(form.test_or_med[5].data)
            causes6 = doctor.Sign(form.causes[5].data)
            test_or_med7 = doctor.Sign(form.test_or_med[6].data)
            causes7 = doctor.Sign(form.causes[6].data)
            diseases1 = patient.Encrypt(diseases1)
            diseases2 = patient.Encrypt(diseases2)
            diseases3 = patient.Encrypt(diseases3)
            test_or_med1 = patient.Encrypt(test_or_med1)
            causes1 = patient.Encrypt(causes1)
            test_or_med2 = patient.Encrypt(test_or_med2)
            causes2 = patient.Encrypt(causes2)
            test_or_med3 = patient.Encrypt(test_or_med3)
            causes3 = patient.Encrypt(causes3)
            test_or_med4 = patient.Encrypt(test_or_med4)
            causes4 = patient.Encrypt(causes4)
            test_or_med5 = patient.Encrypt(test_or_med5)
            causes5 = patient.Encrypt(causes5)
            test_or_med6 = patient.Encrypt(test_or_med6)
            causes6 = patient.Encrypt(causes6)
            test_or_med7 = patient.Encrypt(test_or_med7)
            causes7 = patient.Encrypt(causes7)

            encrypted = SignedEhr(diseases1 = diseases1, diseases2 = diseases2, diseases3 =
diseases3,
                    test_or_med1 = test_or_med1, test_or_med2 = test_or_med2, test_or_med3 =
test_or_med3,
                    test_or_med4 = test_or_med4, test_or_med5 = test_or_med5, test_or_med6 =
test_or_med6,
                    test_or_med7 = test_or_med7,
                    causes1 = causes1, causes2 = causes2, causes3 = causes3,
                    causes4 = causes4, causes5 = causes5, causes6 = causes6,
                    causes7 = causes7)

            db.session.add(db_block)
            db.session.add(post)
            db.session.add(encrypted)
            db.session.commit()

            flash('Your post has been created!', 'success')
            return redirect(url_for('doc_main.home'))
        else:
            flash(str(blockchain.chain[-1].ToDict()))
            flash("BlockChain is not Valid", "danger")
    return render_template('create_post.html', title='New Post',
                           form=form, legend='New Post')

@login_required
@posts.route("/post/<int:post_id>")
def post(post_id):
    post = Ehr.query.get_or_404(post_id)
```

```
doctor = User.query.filter_by(id=post.doctor_id).first()
encrypted = SignedEhr.query.filter_by(id=post.id).first()
doc_keys = Keys.query.filter_by(user_id=doctor.id).first()
doc_keys = doc_keys.private_key
pat_keys = Keys.query.filter_by(user_id=current_user.id).first()
pat_keys = pat_keys.private_key
patient = Patient(name=current_user.name, private_key=pat_keys)
doc = Doctor(name=doctor.name, private_key=doc_keys)
diseases1 = patient.Decrypt(encrypted.diseases1)
diseases2 = patient.Decrypt(encrypted.diseases2)
diseases3 = patient.Decrypt(encrypted.diseases3)
test_or_med1 = patient.Decrypt(encrypted.test_or_med1)
causes1 = patient.Decrypt(encrypted.causes1)
test_or_med2 = patient.Decrypt(encrypted.test_or_med2)
causes2 = patient.Decrypt(encrypted.causes2)
test_or_med3 = patient.Decrypt(encrypted.test_or_med3)
causes3 = patient.Decrypt(encrypted.causes3)
test_or_med4 = patient.Decrypt(encrypted.test_or_med4)
causes4 = patient.Decrypt(encrypted.causes4)
test_or_med5 = patient.Decrypt(encrypted.test_or_med5)
causes5 = patient.Decrypt(encrypted.causes5)
test_or_med6 = patient.Decrypt(encrypted.test_or_med6)
causes6 = patient.Decrypt(encrypted.causes6)
test_or_med7 = patient.Decrypt(encrypted.test_or_med7)
causes7 = patient.Decrypt(encrypted.causes7)

diseases1 = doc.Verify(post.diseases1, diseases1)
diseases2 = doc.Verify(post.diseases2, diseases2)
diseases3 = doc.Verify(post.diseases3, diseases3)
test_or_med1 = doc.Verify(post.test_or_med1, test_or_med1)
causes1 = doc.Verify(post.causes1, causes1)
test_or_med2 = doc.Verify(post.test_or_med2, test_or_med2)
causes2 = doc.Verify(post.causes2, causes2)
test_or_med3 = doc.Verify(post.test_or_med3, test_or_med3)
```

```
    causes3 = doc.Verify(post.causes3, causes3)
    test_or_med4 = doc.Verify(post.test_or_med4, test_or_med4)
    causes4 = doc.Verify(post.causes4, causes4)
    test_or_med5 = doc.Verify(post.test_or_med5, test_or_med5)
    causes5 = doc.Verify(post.causes5, causes5)
    test_or_med6 = doc.Verify(post.test_or_med6, test_or_med6)
    causes6 = doc.Verify(post.causes6, causes6)
    test_or_med7 = doc.Verify(post.test_or_med7, test_or_med7)
    causes7 = doc.Verify(post.causes7, causes7)

    if diseases3 and diseases2 and diseases1 and test_or_med1 and test_or_med2 and
test_or_med3 and test_or_med4 and test_or_med5 and test_or_med6 and test_or_med7 \
        and causes7 and causes6 and causes5 and causes4 and causes3 and causes2 and
causes1:
        flash("OKAY ALL GOOD, DATA HAS NOT BEEN TAMPERED WITH", "success")
    else:
        flash("DANGER CONTENT IS INVALID")

    return render_template('post.html', title=post.id, post=post, doctor=doctor)



@posts.route("/post/<int:post_id>/update", methods=['GET', 'POST'])
@login_required
def update_post(post_id):
    flash("POST CAN NOT BE UPDATED ONCE ENTERED", "danger")
    return render_template('errors/403.html')



@posts.route("/post/<int:post_id>/delete", methods=['POST'])
@login_required
def delete_post(post_id):
    flash("POST CAN NOT BE DELETED ONCE ENTERED", "danger")
    return render_template('errors/403.html')
```

RECOMMENDER

8.15 forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, BooleanField, FieldList,
SelectField, RadioField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from rabadiom.recommender.utils import unique_specialities


class Recommend(FlaskForm):
    speciality = SelectField('Speciality', choices=unique_specialities)
    filter = RadioField('Filter By', choices=[('rating', 'Rating'),('rate', 'Rate'),('distance',
'Distance')], default='rating')
    #a_or_d = RadioField('Ascending or Descending', choices=[('asc', 'Ascending'),('desc',
'Descending')], default='asc')
    address = StringField('Address')
    submit = SubmitField('Submit')

    def address_not_valid(self):
        raise ValidationError("Address needs to be filled")
```

8.16 Routes.py

```
from flask import render_template, Blueprint, flash, request, redirect, url_for
from rabadiom.recommender.forms import Recommend
from flask_login import login_user, current_user, logout_user
from rabadiom import login_manager
from rabadiom.recommender.utils import sort_by_distance, sort_by_rating, sort_by_rate
import requests, json
from functools import wraps
```

```
def login_is_required(role):
    def wrapper(fn):
        @wraps(fn)
        def decorated_view(*args, **kwargs):
            if not current_user.is_authenticated:
              return login_manager.unauthorized()
            if (current_user.role != role):
                return login_manager.unauthorized()
            return fn(*args, **kwargs)
        return decorated_view
    return wrapper



recommend = Blueprint('recommend', __name__)



#@login_is_required("User")
@recommend.route("/Doctor Recommender", methods=['GET', 'POST'])
def doctor_recommender():
	if(current_user.is_authenticated and current_user.role == "User"):
		pass
	else:
		flash("Please login to access this page", "success")
		return redirect(url_for('users.login'))
	form = Recommend()
	if form.validate_on_submit():
		#flash(form.filter.data,"success")
		doctors = []
		if form.speciality.data == "None":
			flash("Please Enter A Speciality", "Danger")
		if form.address.data == "None":
			flash("Please Enter your Address", "Danger")
			return render_template('doctor_recommender.html', title='Doctor
Recommender', form=form)
```

```
                address = form.address.data
                api_key = 'AIzaSyCukArbjvhnW2JjGBedKNUR5fBfbY2KzWg'
                url = 'https://maps.googleapis.com/maps/api/geocode/json?'
                res_ob = requests.get(url + 'address=' + address + '&key=' + api_key)
                x = res_ob.json()
                try:
                        lat = x['results'][0]['geometry']['location']['lat']
                        lng = x['results'][0]['geometry']['location']['lng']
                except:
                        lat = 0
                        lng = 0
                latlang = [lat, lng]
                #flash(str(lat) + str(lng))
                if latlang == [0,0]:
                        flash("Enter A Valid Address to Filter by Address", "danger")
                        return render_template('doctor_recommender.html', title='Doctor
Recommender', form=form)
                if form.filter.data == 'distance':
                        doctors = sort_by_distance(latlang[0], latlang[1], form.speciality.data)
                elif form.filter.data == 'rating':
                        doctors = sort_by_rating(form.speciality.data)
                else:
                        doctors = sort_by_rate(form.speciality.data)
                return render_template('show_map.html', title='Doctor Recommender',
doctors=doctors, lat=lat, lng=lng)
        elif request.method == "GET":
                #form.a_or_d = 'asc'
                #form.filter = 'rating'
                #form.speciality = 'None'
                form.address.data = "None"
        return render_template('doctor_recommender.html', title='Doctor Recommender',
form=form)
```

8.16 utils.py

```
import pandas as pd
from math import sin, cos, sqrt, atan2, radians
import requests, json




df = pd.read_csv("C:/Users/jaski/OneDrive/Desktop/major project/doctor_dataset.csv")

unique = sorted(list(df["Speciality"].unique()))

unique_specialities = []
unique_specialities.append(('Select',"None"))
for u in unique:
	unique_specialities.append((u,u))



def calculate_distance(lat1, long1, lat2, long2):
	R = 6373.0
	dlon = long2 - long1
	dlat = lat2 - lat1
	a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
	c = 2 * atan2(sqrt(a), sqrt(1 - a))

	distance = R * c

	return distance




def sort_by_distance(lat, lng, speciality):
	df = pd.read_csv("C:/Users/jaski/OneDrive/Desktop/major
project/doctor_dataset.csv")
```

```
    newdf = df.where(df["Speciality"] == speciality).dropna()
    dist = []
    for i in range(len(newdf)):
        dist.append(calculate_distance(lat, lng, newdf["lat"].iloc[i], newdf["lng"].iloc[i]))
    newdf["distance"] = dist
    newdf = newdf.sort_values(by=["distance"])
    result = []
    for i in range(5):
        try:
            result.append(newdf.iloc[i].dropna().to_dict())
        except:
            pass

    return result

def sort_by_rating(speciality):
    df = pd.read_csv("C:/Users/jaski/OneDrive/Desktop/major
project/doctor_dataset.csv")
    newdf = df.where(df["Speciality"] == speciality).dropna()
    newdf = newdf.sort_values(by=["Rating"], ascending=False)
    result = []
    for i in range(5):
        result.append(newdf.iloc[i].dropna().to_dict())

    return result

def sort_by_rate(speciality):
    df = pd.read_csv("C:/Users/jaski/OneDrive/Desktop/major
project/doctor_dataset.csv")
    newdf = df.where(df["Speciality"] == speciality).dropna()
    newdf = newdf.sort_values(by=["Rate"])
    result = []
    for i in range(5):
        result.append(newdf.iloc[i].dropna().to_dict())
        return result
```

REFERENCES

[1] Mettler M.A. HSG, IEEE 18th International Conference, e-Health Networking Matthias

[2] Arpita Nayak, Kaustubh Dutta, International Conference on Intelligent Computing and Control (I2C2)

[3] CHUNXUE WU1, CHONG LUO, IEEE Access

[4] Qiuling Suo, Fenglong Ma, Ye Yuan, Mengdi Huai, IEEE TRANSACTIONS ON NANOBIOSCIENCE, JUNE 2018

[5] Pranav Shinde, Sanjay Jhadhav, International Journal of ComputerScience and Information Technologies

[6] Huawei Zhao, Peidong Bai, IET Journal

[7] Guy Zyskind, Oz Nathan, Alex ’Sandy’ Pentland, IEEE CS Security and Privacy Workshops