CHAPTER 1

INTRODUCTION

1.1ABOUT ANDROID

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance and commercially sponsored by Google.

It is free and open source software; its source code is known as Android Open Source Project, which is primarily licensed under the Apache License. However most Android devices ship with additional proprietary software pre-installed, most notably Google Mobile Services, which includes core apps such as Google Chrome, the digital distribution platform Google Play and associated Google Play Services development platform. About 70 percent of Android smartphones run Google's ecosystem.

The versions on android are as follows:

- Android 1.0: Android Alpha
- Android 1.1: Android Beta
- Android 1.5: Android Cupcake
- Android 1.6: Android Donut
- Android 2.0 - 2.1: Android Éclair
- Android 2.2 – 2.2.3: Android Froyo
- Android 2.3.3 – 2.3.7: Android Gingerbread
- Android 3.0 – 3.2.6: Android Honeycomb
- Android 4.0 – 4.o.4: Android Ice Cream Sandwich
- Android 4.1 – 4.3.1: Android Jelly bean
- Android 4.4 – 4.4.4: Android KitKat

- Android 5.0 – 5.1.1: Lollipop
- Android 6.0 – 6.o.1: Android Marshmallow
- Android 7.0 – 7.1.2: Android Nougat
- Android 8.0 – 8.1.o: Android oreo
- Android 9.0: Android Pie
- Android 10

1.2ANDROID ARCHITECTURE

Android architecture contains different number of components to support any android device needs. Android software contains an open-source Linux Kernel having collection of number of C/C++ libraries which are exposed through an application framework services.

Among all the components Linux Kernel provides main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide platform for running an android application.

The main components of android architecture are following: -

- Applications
- Application Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

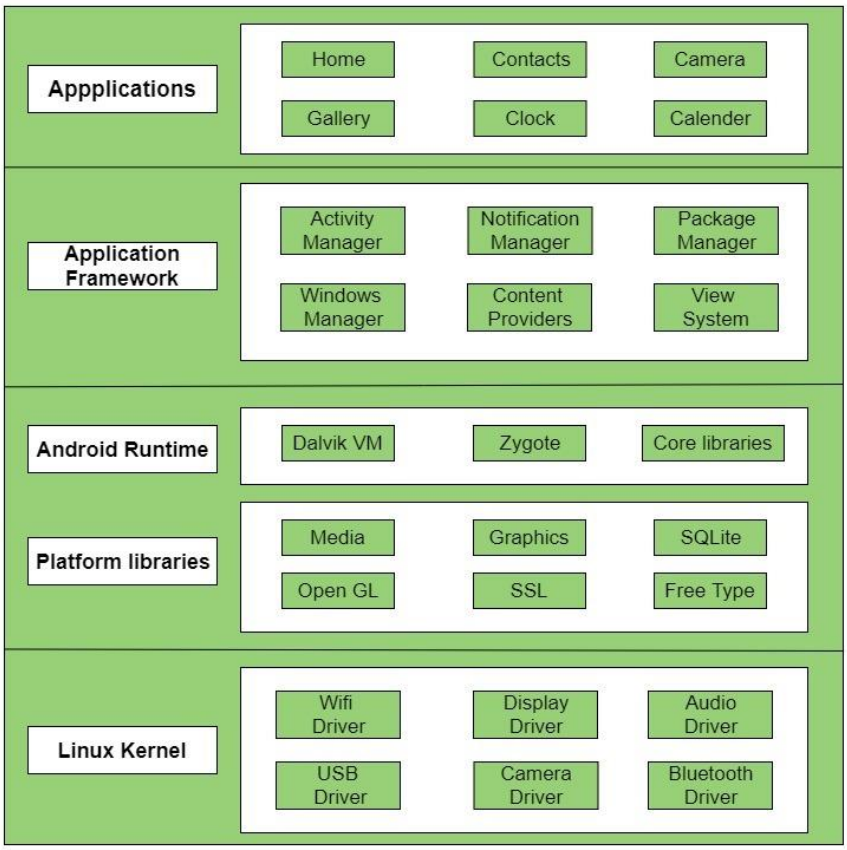

Fig 1.2.1 Android Architecture

Applications:

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc. and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only. It runs within the Android run time with the help of the classes and services provided by the application framework.

Application framework:

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources. Generally, it provides the

services with the help of which we can create a particular class and make that class helpful for the Applications creation.

Application runtime:

Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.

Like Java Virtual Machine (JVM), **Dalvik Virtual Machine (DVM)** is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management.

Platform libraries:

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

- **Media** library provides support to play and record an audio and video formats.
- **Surface manager** responsible for managing access to the display subsystem.
- **RecyclerView** is used to display content after its fetched from the server.
- **Volley** This open source networking library is used to fetch data from a backend server. It is primary used for login authentication.

Linux Kernel:

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime. The Linux Kernel will provide an abstraction layer

between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc.

The features of Linux kernel are:

- **Security:** The Linux kernel handles the security between the application and the system.
- **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
- **Process Management:** It manages the process well, allocates resources to processes whenever they need them.

1.3 ABOUT MINI PROJECT

Most of our health data are eating dust in hospital's record or databases. With the invention of advance data analytics tools like machine learning, we now have the ability of make sense of this data, find patters in it and even make prediction using it.

The patient can be given control of this data which will solve many privacy issues and machine learning can help them understand it in ways it has never been possible. This data can be used by doctors and researcher, with patient's permission of course to discover new patterns and drive medical research further into the future. Earlier a trained and experienced healthcare professional was needed to perform make sense of this data, but now with advance AI algorithms, we can capture their experience and knowledge to automate it.

DISCLAIMER AND CAUTION: This is not designed to replace any healthcare professional or supervise them. It is in its very early stages and therefore is prone to error. The system in the future can help the healthcare professionals reduce their strain. Just like any other technology AI is a tool and it depends on the user how it will be used, and like every other intelligent information processing system, it can and will make mistakes. This system is

not a replacement but an example on how collaboration between humans and machines can make the world a better place.

India have one doctor for every 10,189 patients which is well below the recommended ratio of 1:1000 by World Health Organization. Diagnostic system like the one presented in this report can help reduce and save the very expensive time of the Indian doctors.

1.4 OBJECTIVES OF THE MINI PROJECT

- The main objective of this project is to an easy way to understand your health data by an android application.
- The ease of usage is the main objective of this course.
- The software dose all the heavy calculation on the cloud and therefore is not constrained by the android architecture or underlying hardware.

1.5 ADVANTAGES OF THE MINI PROJECT

- The advantage of the mini project is the use of an android application to complete a task that is otherwise considered time consuming and may even required some level of domain expertise.
- It bring the power of AI into an android application without constraining the phone's hardware.

CHAPTER 2

REQUIREMENT SPECIFICATIONS

2.1 MOBILE REQUIREMENTS

- **Operating System:** Android Marshmallow or above.
- **RAM**: 4 GB
- **Other Requirements:** Internet access
- **Libraries:** Volley (For networking)

2.2 COMPUTER REQUIREMENTS

- **RAM**: 8 GB or more (16 GB recommended)
- **Processor**: Intel i7 or greater
- **GPU**: Nvidia GTX 1050 Ti
- **Operating System**: Windows
- **Developing Language**: Java
- **Tools used**: Android Studio
- **Backend Database**: MySQL
- **Backend Language**: Python
- **Backend Frameworks**: PIL, NumPy, Pandas, TensorFlow
- **Networking Framework**: Flask Microframework

CHAPTER 3

ANALYSIS AND DESIGN

3.1. DESIGN GOALS

The software is going to use a client server architecture. The front-end of the system is written in java and the graphical user interface is to build using Swing framework in java. The system is using a database for storing and retrieving data that exits on a server which the client software connects for data driven transaction.

In order for data analysis, the system need to utilize python's very powerful machine learning and deep learning frameworks like sklearn and TensorFlow. This can be done on the client machine but it may be slow as it requires a substantial amount of compute. On top of that certain client system won't be able to run deep learning framework like TensorFlow.

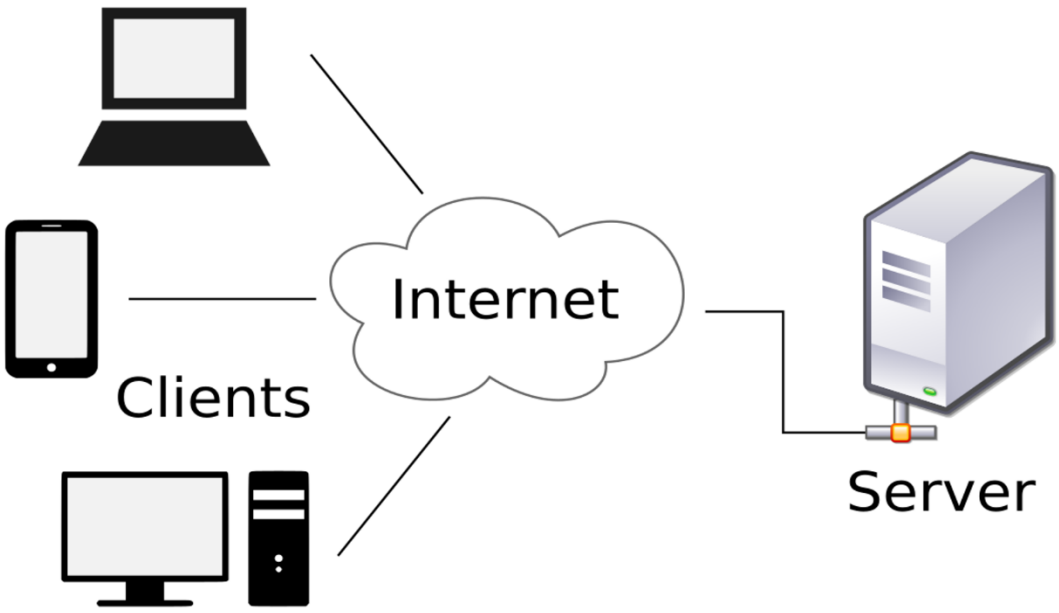

Fig 3.1.1 Client Server Architecture

The above-mentioned design is implemented in one computer but the design of the software is such that, it will require very less effort to scale it to network level. This is achieved by using tailored API that talk between the java code and the python code and using MySQL software's local host feature.

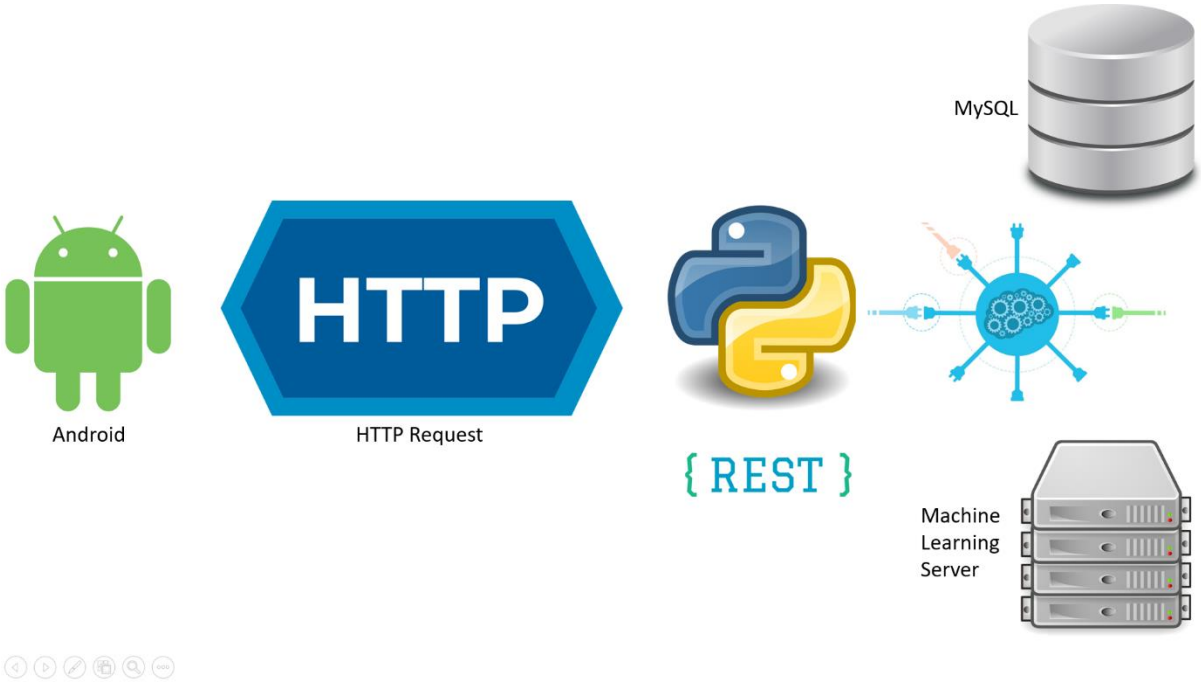

Fig 3.1.2 API Design

3.2. ALGORITHM/PSEUDOCODE

The algorithm used for analysis of the medical data to predict weather or not the patient have chronic kidney disease, is known as the random forest algorithm.

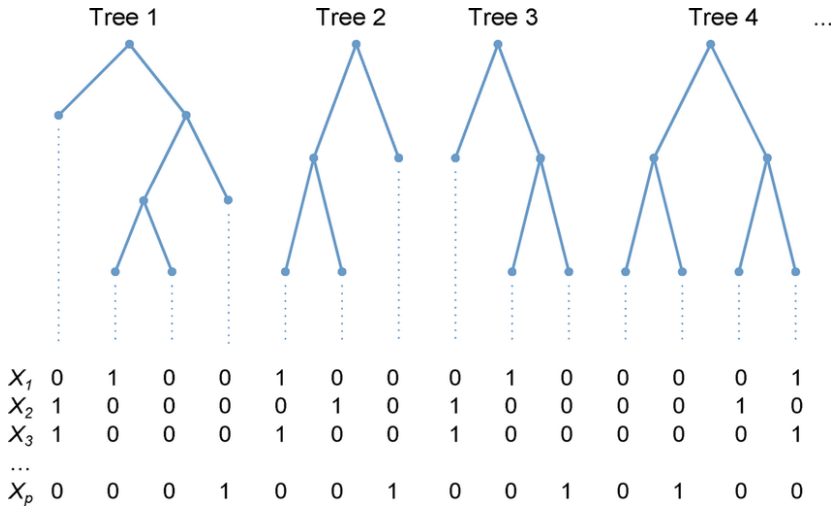

Fig 3.2.1 Random Forest

The random forest classifier uses multiple decision trees to classify. Then it sums ups the output and the class label with the maximum votes is given as the final answer.

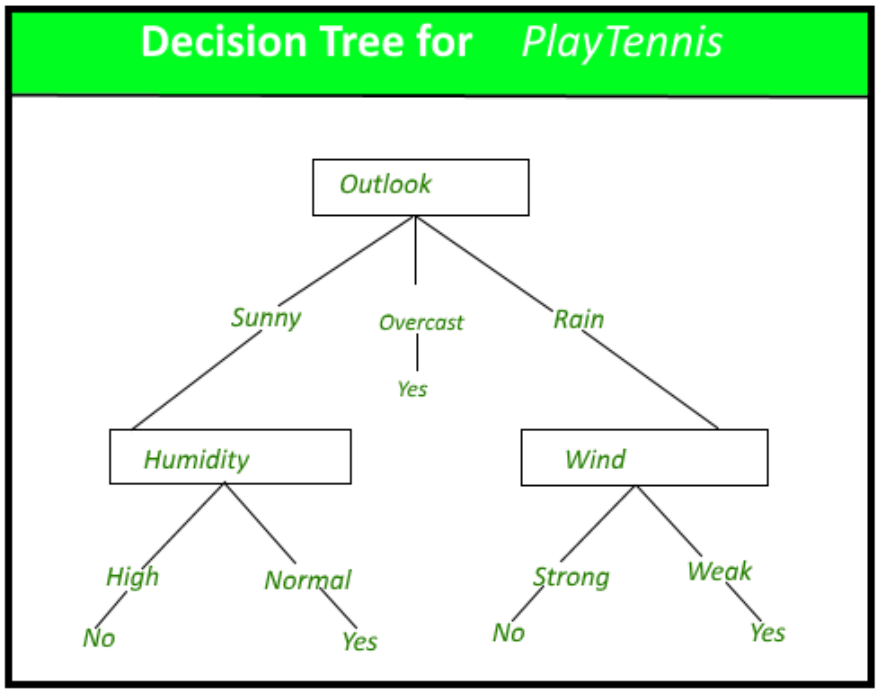

Fig 3.2.2 Decision Tree

The above is an example of a single simple decision tree to help decide a system weather the condition is suitable for playing tennis or not. In case of healthcare data or any real-world scenario for that matter, the tree is much for complex.

The decision tree algorithm repeatedly checks for every feature to see if classifying the data based on it will reduce entropy (degree of disorder). It performs this calculation for every feature. Once it settles on one feature to classify with, it does the same thing on the all the nodes recursively.

3.3 OBJECT-ORIENTED PROGRAMMING

Oops stands of object-oriented programming and in this programming style we have our main focus on objects rather than on the functions and the usual flow of non-OOPs

1. Data Abstraction: It is also known as data hiding, which gives only the essential details of function being used.
2. Encapsulation: It is defined as wrapping up of data and function into one single entity.
3. inheritance: It is defined as the property of inheriting the features and data members of one class to another class. There are four types of inheritance single inheritance, multilevel inheritance, multiple inheritance, hierarchical inheritance.
    a. Single inheritance: ln this type of inheritance there is only one class deriving from the parent.
    b. Multilevel inheritance: ln this type of inheritance we one class inheriting features of the parent class. Now again this class is further derived by other classes.
    c. Multiple inheritance: ln this type of inheritance we have a class inheriting of the features of two or more classes.
    d. Hierarchical inheritance: ln this type of inheritance we have many classes inheriting from the same base class.

4. Polymorphism: It is the ability to represent a given function in more than one form. We can do polymorphism either by running the overloaded functions or by over ridding them at the time of call.

3.4 FLOWCHART

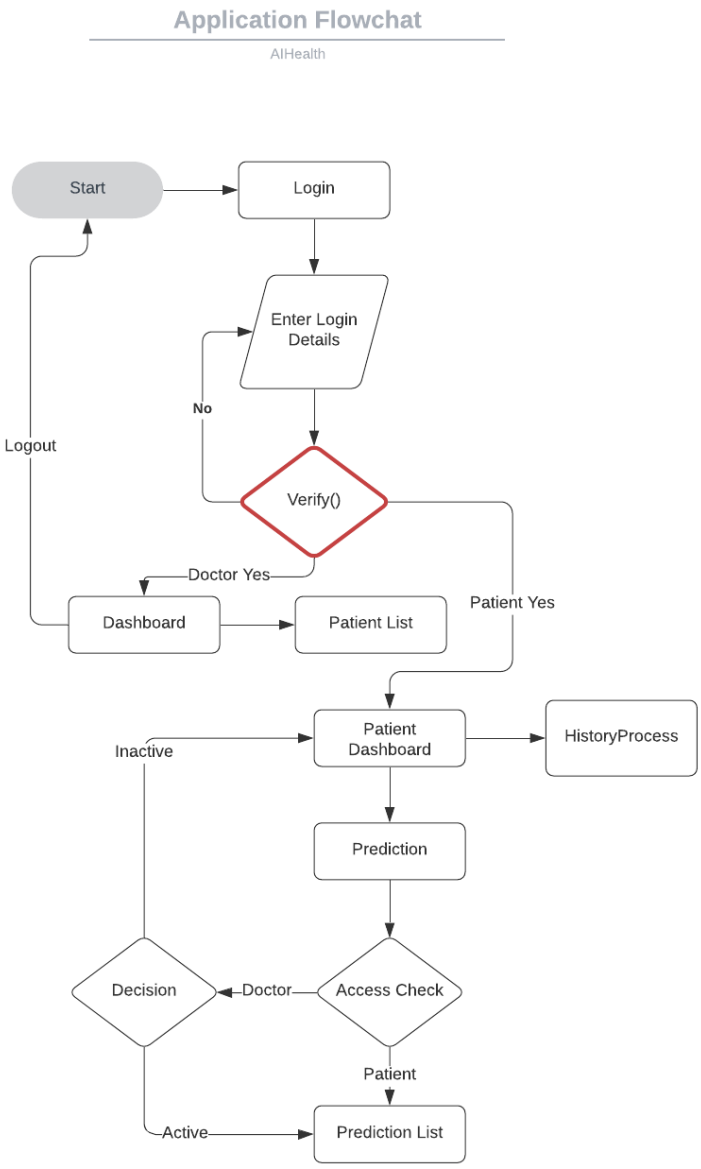

Figure 3.4.1 Flowchart

CHAPTER 4

IMPLEMENTATION

4.1. ANDROID CONCEPTS USED

The project has been development using the Android Framework which is similar to wing framework. ln this framework there are usually two parts FXML files and Controller classes. The FXML file has Panes like Anchor Panes, Stack Pane etc. which is loaded into the scene and put onto the user screen(Stage).The FXML file is used for designing the front end using Scene Builder application and we give IDs to the various fields which we use in the Controller class to call it and give functionality to the various fields. ln short this follows the similar concept of website where we use the HTML, CSS to give the Ul design and some backend language like PHP, JavaScript etc. to add the functionality.

- The first screen has the login where we select the user type and then enter user name and password and go to the next page. If the username and the corresponding password matches with the database on the backend server, the user go its respective dashboard.
- If it's a patient, then the patient dashboard shows details like name, age, etc. but if it's the doctor dashboard then it displays details like name, specialization but also have the option to navigate to patient list.
- The patient list will list the patient using a RecyclerView and upon selection it will take the user to the respective patient dashboard.
- The patient has the option to go to the prediction which show what prediction services is available to the patient.
- The prediction service list depends on the user data and the availability of the service itself.

The concepts use in this project are as follows:

- Volley: Volley is a networking library that can make server request and the parse JSON data retrieved from the server. As the app is retrieving almost all of it data and is functionality depends completely on the a server (cloud), volley is extensively used in every activity.

  Fox example: Volley is used to validate username for password for login.

```
public void verifyLogin(String ip, final String LoginType, final String usrname, String passwd) {
    String url = "http://" + ip + ":5000/" + LoginType + "/" + usrname + "/" + passwd + "/";
    Log.d( tag: "devout", url);

    final String ip_address = ip;

    RequestQueue queue = Volley.newRequestQueue( context: this);

    final JsonObjectRequest request = new JsonObjectRequest(Request.Method.GET, url, jsonRequest: null
            Log.d( tag: "devout", msg: "Response Received");
            try {
                String results = response.getString( name: "code");
                Log.d( tag: "devout", msg: "Request code is " + results);
                if (results.equals("200")) {
                    Log.d( tag: "devout", msg: "Login Verified");
                    String name = response.getString( name: "fname") + " " + response.getString( name:
                    Log.d( tag: "devout", name);
                    if (LoginType.equals("doc_log")) {
                        Doctor doctor = new Doctor(
                                usrname,
                                name,
                                Integer.parseInt(response.getString( name: "age")),
                                response.getString( name: "specialization"),
                                response.getString( name: "address"),
                                response.getString( name: "ph_no")
                        );
                        Intent intent =  new Intent(getApplicationContext(), DoctorDashboard.class);
                        intent.putExtra( name: "Doctor", doctor);
                        intent.putExtra( name: "ip", ip_address);
                        startActivity(intent);
                        Log.d( tag: "devout", msg: "Doctor");
                    }
                    else {
                        Patient patient = new Patient(
```

```
                        usrname,
                        name,
                        Integer.parseInt(response.getString( name: "age")),
                        response.getString( name: "ph_no")
                );
                Intent intent =  new Intent(getApplicationContext(), PatientDashboard.class);
                intent.putExtra( name: "Patient", patient);
                intent.putExtra( name: "ip", ip_address);
                startActivity(intent);
                Log.d( tag: "devout",  msg: "Patient");
            }
        }
        else {
            Toast toast = Toast.makeText(getApplicationContext(),
                    text: "Incorrect username/password",
                    Toast.LENGTH_SHORT);

            toast.show();
        }
    } catch (JSONException e) {
        Log.e( tag: "devout", msg: "JSON error");
    }
}, (error) -> {
    Toast toast = Toast.makeText(getApplicationContext(),
            text: "Network Error",
            Toast.LENGTH_SHORT);

    toast.show();
    Log.e( tag: "devout",  msg: "Listener error " + error.getMessage());
});

queue.add(request);
}
```

The verifyLogin() function take in the entered username and password, then it create a RequestQueue. Then the JSONObjectRequest() take the context and the URL for the login verification server. It makes the request and gets a JSON response, which is then parsed for user data if the validation is successful. Then this is added to the RequestQueue which runs on a separate network thread.

- Multithreading: The volley library used a separate network thread to run the request. This allows for the background thread for the activity to run without waiting for the server response. For example, the dashboard activity thread start the patient list activity thread with it functionality, while the network thread tuns on the background. The network takes the response from the sever and populated the patient list activity with patient name.

- RecyclerView: RecyclerView is used in the patient list activity to display the data retrieved from the backend server in a list.

  The Adapter class of the RecyclerView is defined for this functionality.

```
package org.scorp.aihealth;

import ...

public class PatientAdapter extends RecyclerView.Adapter<PatientAdapter.PatientViewHolder> {

    private Context context;
    private ArrayList<Patient> patientArrayList;
    private OnItemClickListener listener;

    public interface OnItemClickListener {
        void onItemClick(int position);
    }

    public void setOnItemClickListener(OnItemClickListener listener) {
        this.listener = listener;

    }

    public PatientAdapter(Context context, ArrayList<Patient> patientArrayList) {
        this.context = context;
        this.patientArrayList = patientArrayList;
    }

    @NonNull
    @Override
    public PatientViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(context).inflate(R.layout.patient_element, parent, attachToRo
        return new PatientViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull PatientViewHolder holder, int position) {
        Patient currentPatient = patientArrayList.get(position);

        String name = currentPatient.getName();
```

```
        String username = currentPatient.getUsername();

        holder.p_username.setText(username);
        holder.p_name.setText(name);
    }

    @Override
    public int getItemCount() { return patientArrayList.size(); }

    public class PatientViewHolder extends RecyclerView.ViewHolder {

        public TextView p_username;
        public TextView p_name;

        public PatientViewHolder(@NonNull View itemView) {
            super(itemView);
            p_username = itemView.findViewById(R.id.p_username);
            p_name = itemView.findViewById(R.id.p_name);
            itemView.setOnClickListener((v) -> {
                    if (listener != null) {
                        int position = getAdapterPosition();
                        if (position != RecyclerView.NO_POSITION) {
                            listener.onItemClick(position);
                        }
                    }
            });
        }
    }
}
```

The adapter class defines the functionality of the RecyclerView. The response is a JSONArray which is then looped through to create a object array of patient. The RecyclerView takes in this object array and fit the data to the view. In a **M**odel **V**iew **C**ontroller (MVC), the adapter class would be the controller while the RecyclerView itself would be the view and the response would be the model. The controller models the data to the view. The adapter also add the functionality of the selecting a patient and then opening the respective patient dashboard.

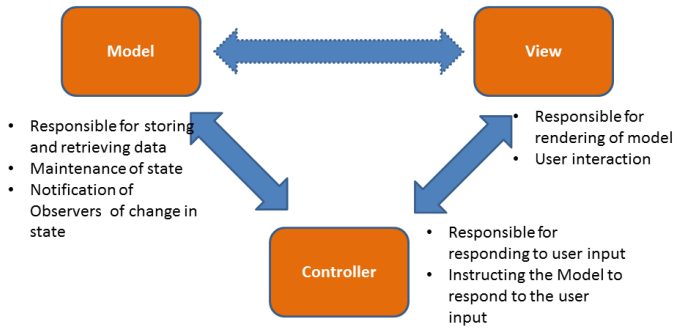

Figure 4.1.1 MVP Design

4.2. FUNCTIONALITY OF THE PROJECT

The main functionality of the project is the prediction services. The user has the option to perform ML based prediction on the user data. This is one of the most import function of the android application.

```
private void predictCKD(String ip, String username) {
    String url = "http://" + ip + ":5000/pat_prediction/ckd/" + username + "/";
    Log.d( tag: "devout", url);
    RequestQueue queue = Volley.newRequestQueue( context: this);
    final JsonObjectRequest request = new JsonObjectRequest(Request.Method.GET, url, jsonRequest: null,
            Log.d( tag: "devout", msg: "Response Received");
            try {
                String result = "Server Error";
                String code = response.getString( name: "code");
                if (code.equals("200")) {
                    result = response.getString( name: "result");
                }
                else{
                    result = response.getString( name: "error");
                }
                Toast toast = Toast.makeText(getApplicationContext(),
                        result,
                        Toast.LENGTH_SHORT);
                toast.show();
            } catch (JSONException e) {
                Log.e( tag: "devout", msg: "JSON error");
                Toast toast = Toast.makeText(getApplicationContext(),
                        text: "JSON Error",
                        Toast.LENGTH_SHORT);
                toast.show();
            }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast toast = Toast.makeText(getApplicationContext(),
                    text: "Network Error",
                    Toast.LENGTH_SHORT);

            toast.show();
            Log.e( tag: "devout", msg: "Listener error " + error.getMessage());
            Toast toast = Toast.makeText(getApplicationContext(),
                    text: "Network Error",
                    Toast.LENGTH_SHORT);

            toast.show();
            Log.e( tag: "devout", msg: "Listener error " + error.getMessage());
        }
    });

    queue.add(request);
}
```

4.3. SQL DATABASE CONNECTIVITY

The SQL database is in the backend server which is accessed by a python Flask web server. the Flask microframework is used to create a web server which then connects with the MySQL local server.

```python
from flask import Flask
from flask import jsonify
from verifyLogin import *
from DBPatientList import *
from MLServer import ChronicKidneyDisease

app = Flask(__name__)

@app.route("/doc_log/<string:username>/<string:password>/", methods=['GET'])
def docLog(username, password):
    loginType = "doc_log"
    code = verifyLogin(loginType, username, password)
    response = {}

    if (code == "200"):
        response = retiveDocData(username)
    elif (code == "201"):
        response = {
            "code": code,
            "error": "Incorrect username/password"
        }
    else:
        response = {
            "code": code,
            "error": "Database error"
        }

    print(response)
    return jsonify(response)

#http://192.168.0.112:5000/pat_log/ruman@scorp.com/ruman18/
@app.route("/pat_log/<string:username>/<string:password>/", methods=['GET'])
def patLog(username, password):
    loginType = "pat_log"
    code = verifyLogin(loginType, username, password)
    response = {}

    if (code == "200"):
        response = retivePatData(username)
    elif (code == "201"):
        response = {
            "code": code,
```

```
            "error": "Incorrect username/password"
        }
    else:
        response = {
            "code": code,
            "error": "Database error"
        }

    print(response)
    return jsonify(response)

@app.route("/doc/patlist/<string:username>/", methods=['GET'])
def running(username):
    response = retrivePatList(username)
    print(response)
    return jsonify(response)

@app.route("/pat_prediction/ckd/<string:username>/", methods=['GET'])
def ckdPrediction(username):
    response = ChronicKidneyDisease(username)
    print(response)
    return jsonify(response)

# start flask app
app.run(host="0.0.0.0", port=5000)
```

This web server allows to android client to utilize the backend database through different URLs. Each URL performs a specific database retinal by connecting to the database, forming the query and executing it.

CHAPTER 5

SAMPLE OUTPUT

The sample output of the activity screen along with the server output are as follows:

Android Screenshots (Client Output):

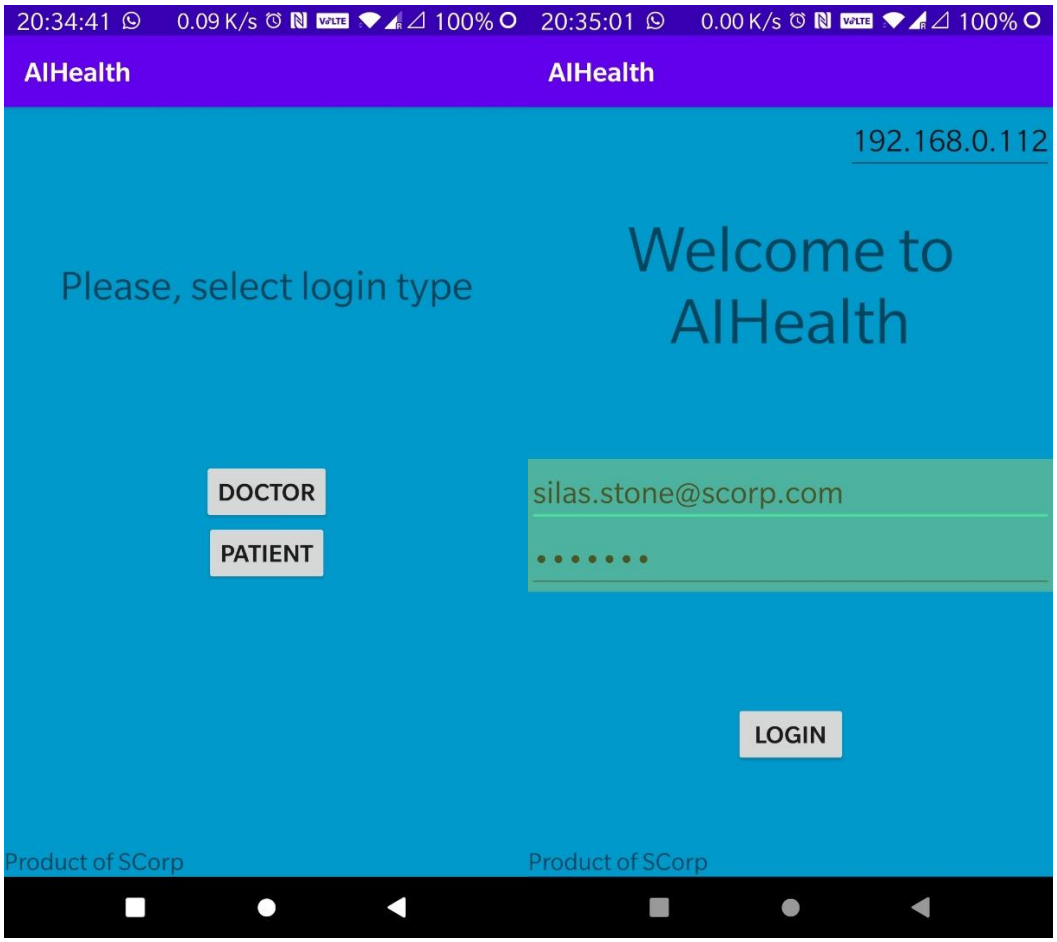

Figure 5.1.1 Login Screen

Server Output after tapping “Login”:

```
select * from doc where username = "silas.stone@scorp.com";
{'code': '200', 'lanme': 'Stone', 'fname': 'Silas', 'age': 52, 'address': 'Detroit', 'specialization': 'Prosthesis', '
ph_no': 2442833174, 'institute_id': 1}
192.168.0.101 - - [06/Dec/2020 20:35:07] "←[37mGET /doc_log/silas.stone@scorp.com/stone52/ HTTP/1.1←[0m" 200 -
```

Android Screenshots (Client Output):

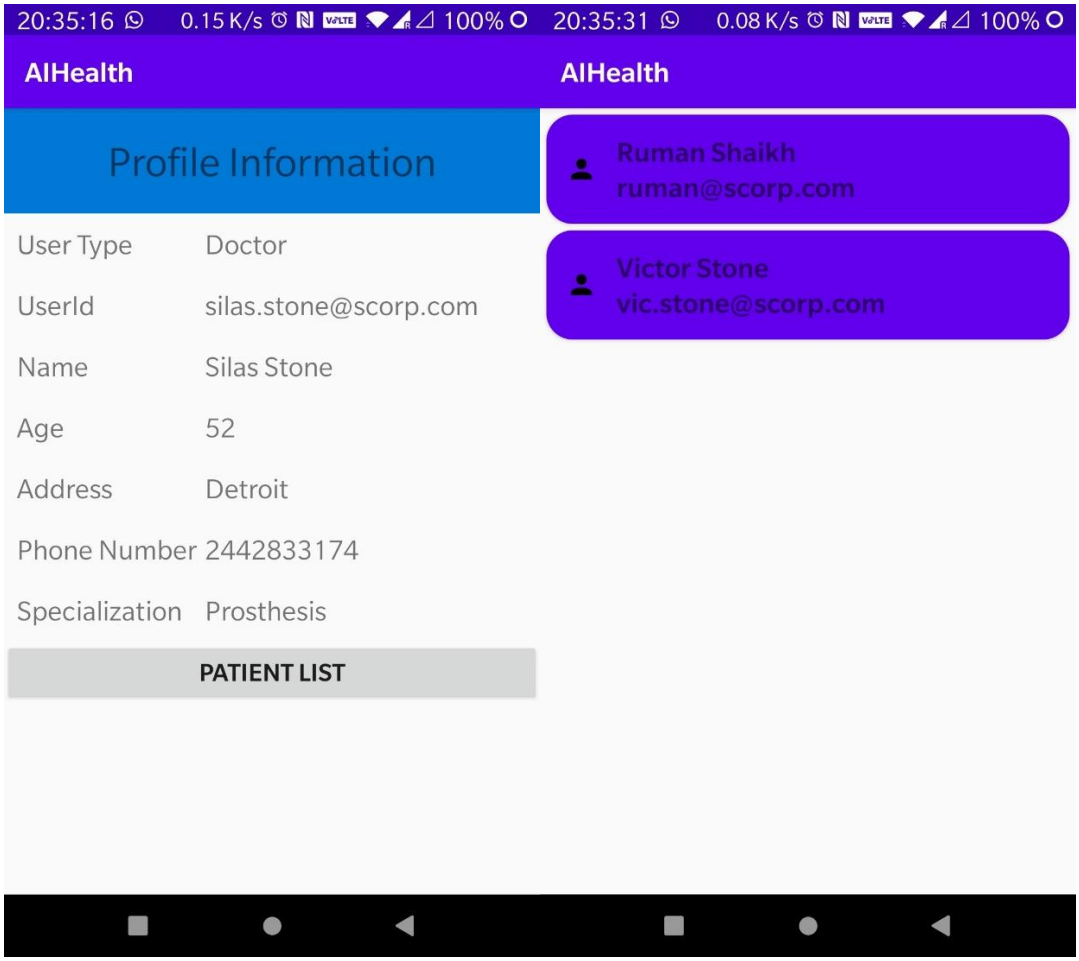

Figure 5.1.2 Doctor Profile

Server Output after tapping a patient:

```
select * from doc where username = "silas.stone@scorp.com";
{'code': '200', 'lanme': 'Stone', 'fname': 'Silas', 'age': 52, 'address': 'Detroit', 'specialization': 'Prosthesis', '
ph_no': 2442833174, 'institute_id': 1}
192.168.0.101 - - [06/Dec/2020 20:35:07] "←[37mGET /doc_log/silas.stone@scorp.com/stone52/ HTTP/1.1←[0m" 200 -
{'code': '200', 'message': 'Successful Retrieval', 'size': 2, 'patientList': [{'username': 'ruman@scorp.com', 'lname':
 'Shaikh', 'fname': 'Ruman', 'age': 22, 'ph_no': 9538772702}, {'username': 'vic.stone@scorp.com', 'lname': 'Stone', 'f
name': 'Victor', 'age': 22, 'ph_no': 6418378416}]}
192.168.0.101 - - [06/Dec/2020 20:35:25] "←[37mGET /doc/patlist/silas.stone@scorp.com/ HTTP/1.1←[0m" 200 -
```

Android Screenshots (Client Output):

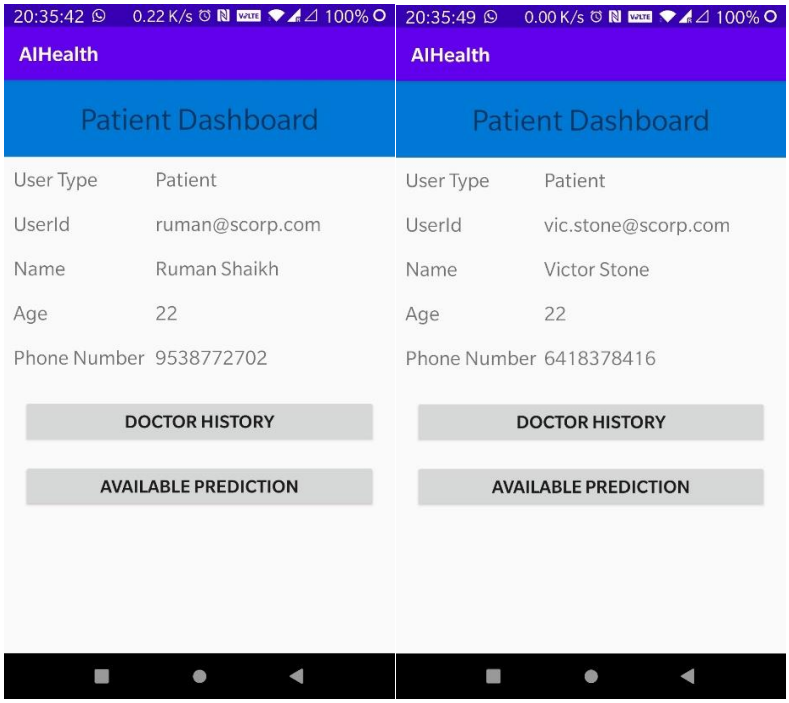

Figure 5.1.3 Patient Profile 1

Android Screenshots (Client Output):

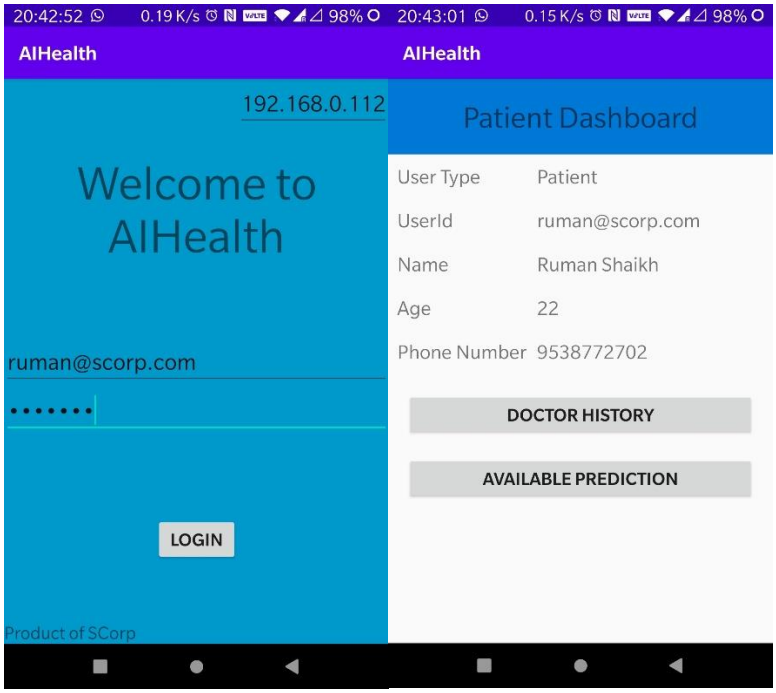

Figure 5.1.4 Patient Profile 2

Server Output after tapping a patient:

```
select * from pat where username = "ruman@scorp.com";
{'code': '200', 'lanme': 'Shaikh', 'fname': 'Ruman', 'age': 22, 'ph_no': 9538772702}
192.168.0.101 - - [06/Dec/2020 20:42:57] "←[37mGET /pat_log/ruman@scorp.com/ruman18/ HTTP/1.1←[0m" 200 -
0
{'code': '200', 'result': 'Negative'}
192.168.0.101 - - [06/Dec/2020 20:43:13] "←[37mGET /pat_prediction/ckd/ruman@scorp.com/ HTTP/1.1←[0m" 200 -
0
{'code': '200', 'result': 'Negative'}
192.168.0.101 - - [06/Dec/2020 20:43:19] "←[37mGET /pat_prediction/ckd/ruman@scorp.com/ HTTP/1.1←[0m" 200 -
```

Android Screenshots (Client Output):

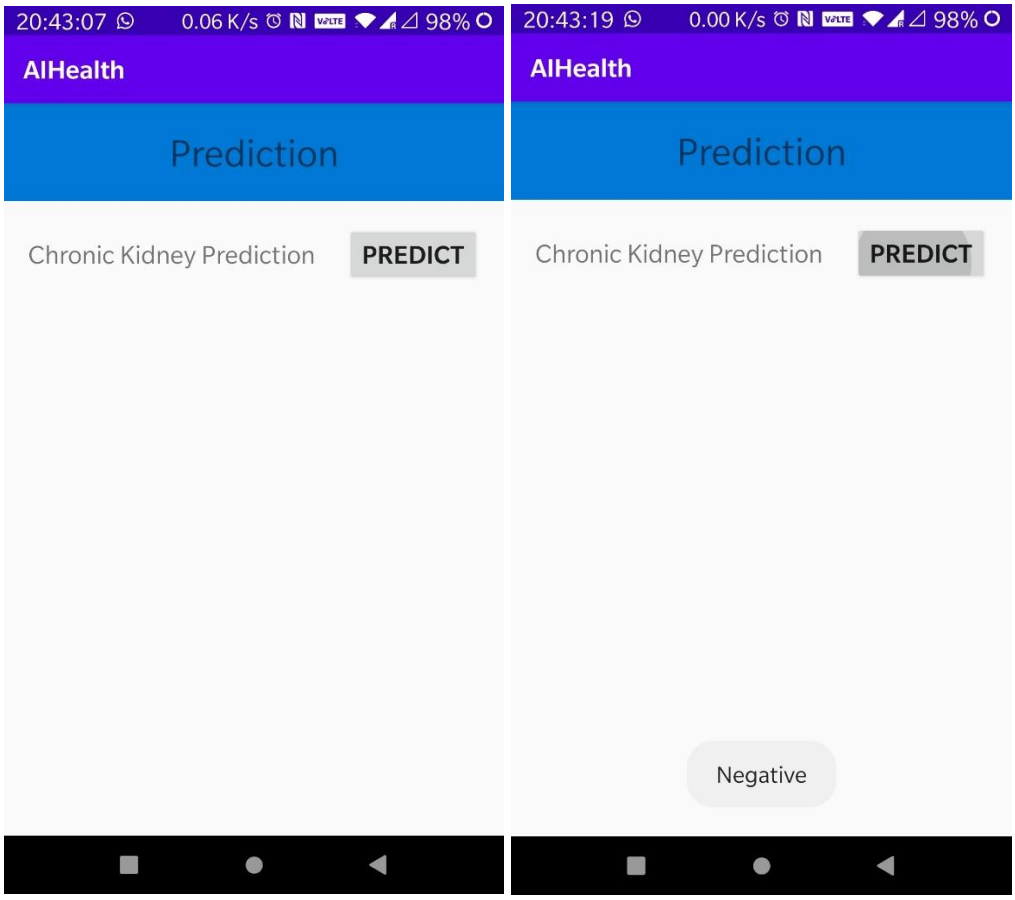

Figure 5.1.5 Prediction Screen

Server Output after tapping a patient:

```
select * from doc where username = "silas.stone@scorp.com";
{'code': '200', 'lanme': 'Stone', 'fname': 'Silas', 'age': 52, 'address': 'Detroit', 'specialization': 'Prosthesis', '
ph_no': 2442833174, 'institute_id': 1}
192.168.0.101 - - [06/Dec/2020 20:41:29] "←[37mGET /doc_log/silas.stone@scorp.com/stone52/ HTTP/1.1←[0m" 200 -
{'code': '200', 'message': 'Successful Retrieval', 'size': 2, 'patientList': [{'username': 'ruman@scorp.com', 'lname':
 'Shaikh', 'fname': 'Ruman', 'age': 22, 'ph_no': 9538772702}, {'username': 'vic.stone@scorp.com', 'lname': 'Stone', 'f
name': 'Victor', 'age': 22, 'ph_no': 6418378416}]}
192.168.0.101 - - [06/Dec/2020 20:41:32] "←[37mGET /doc/patlist/silas.stone@scorp.com/ HTTP/1.1←[0m" 200 -
0
{'code': '200', 'result': 'Negative'}
192.168.0.101 - - [06/Dec/2020 20:41:36] "←[37mGET /pat_prediction/ckd/ruman@scorp.com/ HTTP/1.1←[0m" 200 -
```

CHAPTER 6

CONCLUSION

This project shows the use of server client architecture that help in implementing an android app to perform prediction on healthcare data. This app demonstrates the use of AI in healthcare in a very accessible package. The app uses a number of object-oriented programming principle, advance view in android, flask, etc.

As most app now a days need an internet connection to perform their essential functionality like WhatsApp, Facebook, etc. it's time to not just fetch data from the cloud but also perform compute heavy operation on the cloud and return the result back to the user.

REFERENCES

The references for this project are as follows:

- https://en.wikipedia.org/wiki/Android_(operating_system)
- https://developer.android.com/training/volley
- https://developer.android.com/reference/kotlin/androidx/recyclerview/widget/RecyclerView
- https://developer.android.com/jetpack/androidx/releases/cardview
- https://www.youtube.com/playlist?list=PLhQjrBD2T381qULidYDKP55-4u1piASC1
- https://www.youtube.com/playlist?list=PLrnPJCHvNZuBCiCxN8JPFI57Zhr5SusRL
- https://flask.palletsprojects.com/en/1.1.x/