ASHESI UNIVERSITY

AN INTELLIGENT TELEHEALTH SYSTEM TO MANAGE THE SPREAD OF COVID-19 AT HOSPITALS

APPLIED PROJECT

B.Sc. Computer Science

Jeffrey Kafui Adorkor

2022

ASHESI UNIVERSITY

AN INTELLIGENT TELEHEALTH SYSTEM TO MANAGE THE SPREAD OF COVID-19 AT HOSPITALS

Applied Project

Applied Project submitted to the Department of Computer Science, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science.

Jeffrey Kafui Adorkor

2022

DECLARATION

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

I hereby declare that preparation and presentation of this Applied Project were supervised in accordance with the guidelines on supervision of Applied Project laid down by Ashesi University.

Supervisor's Signature:

……...

Supervisor's Name:

……...

Date:

……...

Acknowledgement

Special thanks to God, my family and my supervisor, Mr. Francis Gatsi for seeing me through this project.

Abstract

With the outbreak of the Coronavirus, people who visit hospitals are put at risk of contracting the virus in addition to their existing condition. As a result of the risks associated with the COVID-19 virus, unique and innovative solutions must be introduced to help reduce the spread of the virus. This project proposes an Intelligent Telehealth system to help manage the spread of the COVID-19 virus by reducing the number of physical interactions at hospitals. This system would help reduce the number of physical interactions at hospitals by instituting virtual consultations between doctors and patients. These consultations would include special features such as a Computer Vision component to monitor facial expressions. These special features would ensure that the gap between physical healthcare and virtual healthcare is bridged. The system was implemented as a mobile application for patients to create and manage bookings with doctors and to manage their medical records as well. A web application was also built to handle video consultations and to give doctors the opportunity to update the patient's medical records during consultations. After the system was implemented, it was tested with three patients to measure its usability and usefulness. Scores were measured on a scale of 1 - 5 and all patients had scores ranging from 3-5 thus showing the usefulness of the system.

Table of Contents

Chapter 1: Introduction

Healthcare is an essential facet of our lives. However, with the outbreak of the Coronavirus, people who visit the hospital are put at risk of contracting the virus in addition to their existing condition. As a result of the risks associated with the COVID-19 virus, unique and innovative solutions must be introduced to help reduce the spread of the virus.

One of these possible solutions is a telehealth system. A telehealth system is a system that employs the use of digital information and communication technologies to help access health care services remotely. Some of these technologies include computers and mobile devices, which are used to improve or support current health care services.

1.1 Background

Coronaviruses are a genus of the Coronaviridae family which cause illness in animals or humans [1]. When humans acquire coronaviruses, they may develop infections of the respiratory system ranging from the common cold to more serious infections [1]. Recently, the world was taken aback by the COVID-19 coronavirus. The disease associated with this virus originated in Wuhan, China and has kept spreading widely to other regions. Even though there seems to be a calm after the storm, the world was not ready when the virus spread like wildfire.

One of the structures that struggled with handling the virus was the health sector. Patients who visited hospitals for different conditions ended up acquiring the COVID-19 virus from hospitals. The rate at which COVID infections which were officially classified as hospital acquired are yet to fall [2].

A significant factor in slowing down the transmission of the virus is social distancing which reduces the amount of person-to-person contact [1]. Therefore, unique, and innovative solutions

are needed to reduce person-to-person contact at hospitals. One of these unique and innovative solutions is Telemedicine.

The use of telehealth technology is a twenty-first century approach that is both patient-centered and protects patients, physicians, as well as others [1]. Telehealth is the delivery of health care services by health care professionals through the use of Information and Communication Technology [1]. Some of these technologies include computers and mobile devices.

1.**2 Motivation**

According to Oliver [2], the rates of COVID infections officially classified as hospital acquired are yet to fall. Patients acquire the virus from hospitals where they expect safety. Formal complaints have been lodged against hospitals from people who are angry and distressed that they may have contracted the virus from hospitals [2]. With the slow reduction of hospital acquired COVID infections, it is important for the number of physical interactions at hospitals to be reduced especially with patients who can easily be treated at home. The introduction of a telehealth system would be beneficial for the reduction of hospital acquired COVID infections.

1.3 Project's Focus

The focus of this project is to design a solution that would help manage the spread of the COVID-19 virus at hospitals by reducing the number of physical interactions at hospitals.

1.4 Related Work

It is possible to find existing Telehealth systems that seek to provide remote healthcare to patients. One of these systems is Medbot which is a conversational artificial intelligence chatbot used for delivering telehealth [3]. Medbot provides free primary healthcare to patients; however,

it is a virtual artificial intelligence chatbot, and its diagnosis does not match up to that of an actual doctor.

Telehealth Mobile system, another Telehealth system, establishes connectivity between patients and doctors. The Telehealth Mobile system seeks to deliver healthcare services to patients by taking advantage of cell communication networks [4]. Although this system connects patients to doctors, it does not include a video conferencing feature for virtual consultations.

The proposed system for this work seeks to connect patients to actual doctors while ensuring that all interactions between patients and doctors are live and in real-time.

1.5 Project Objectives and Significance

This project's main objective is to provide an Intelligent Telehealth system to help manage the spread of the COVID-19 virus. The project seeks to make the following contributions:

- A telehealth system that allows patients to have virtual consultations with doctors.
- A telehealth system that employs Natural Language Processing to help identify a patient's condition after an assessment.
- A telehealth system that incorporates Computer Vision during video consultation to help doctors diagnose a patient's body language.

Chapter 2: User Requirements Specifications

This chapter outlines the functionality and goals of the Telehealth system. It gives an insight into the process of acquiring these goals and outlines the functional and non-functional requirements of the Telehealth system.

2.1 Requirements Elicitation Technique & Procedure

Requirements were derived after conducting a survey on the possible user groups of the Telehealth system. These user groups were identified as patients and doctors. Each of the user groups had different questionnaires that contained a mix of close-ended and open-ended questions. The results of these questionnaires are analyzed in Section 2.1.1 to Section 2.1.2.

2.1.1 Patient Questionnaire Analysis

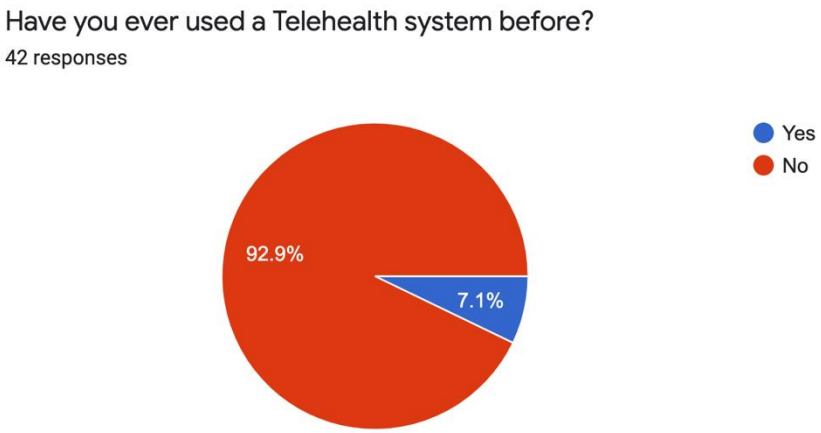


Figure 2.1: Proportion of respondents who have used a Telehealth system to those who have not used one

Out of the 42 responses in Figure 2.1, approximately 93% of respondents have not experienced the use of a Telehealth system before. This statistic informs the need to provide users with an onboarding process to enlighten them on how a Telehealth system functions.

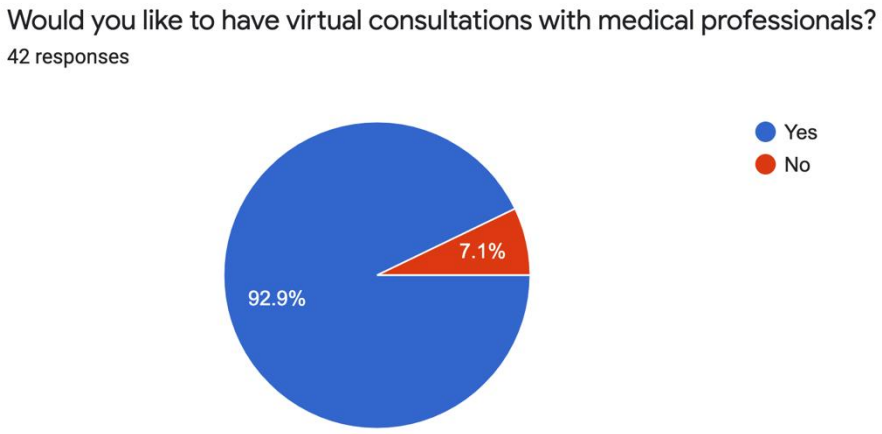


Figure 2.2: Respondents and their desire to have virtual consultations

In Figure 2.2, about 93% of respondents desire to have virtual consultations with medical professionals. This response could be as a result of the lockdown restrictions that occurred during the outbreak of the COVID-19 pandemic. The lockdown period tremendously changed the way people interacted with each other. Meetings and events were moved online to help manage the spread of the virus. This shift caused a spike in daily meeting participants on Zoom, a video conferencing platform. Zoom's daily participants rose from 10 million in December 2019, to over 300 million in April 2020 [5]. It is likely that as the world adapts to the effects of the virus and advancements in technology, people are willing to have virtual consultations with doctors. This could be the reason for their choice in Figure 2.2. Hence the need for a Telehealth system to provide virtual consultations to patients.

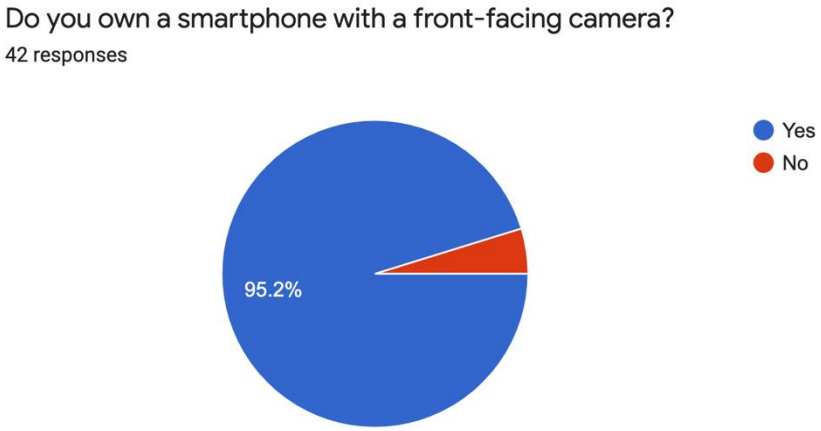


Figure 2.3: Respondents and whether they own smartphones with a front-facing camera

Furthermore, Figure 2.3 supports the reason for the choices made by respondents in Figure 2.2. From the results, it is evident that approximately 95% of the stakeholders who responded to the questionnaire own a smartphone with a front-facing camera.

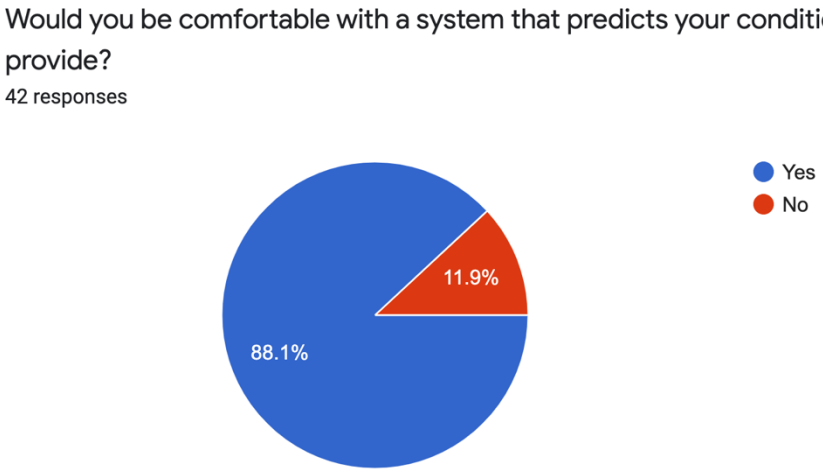


Figure 2.4: Respondents and their comfortability with a symptom prediction system

Of the 42 responses gathered, the majority (approximately 88%) indicated that they were comfortable with a system that predicts their condition based on their symptoms. This shows the importance of a symptom checker in the Telehealth system.

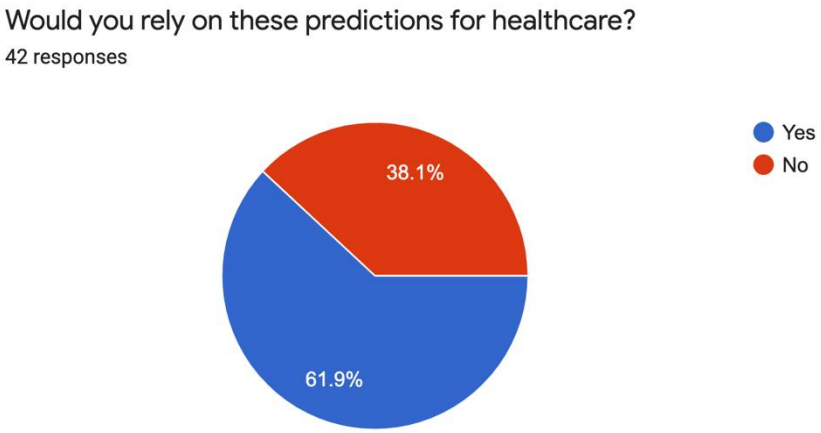


Figure 2.5: Respondents and their comfortability with a symptom prediction system

Approximately 62% of respondents in Figure 2.5 stated that they will rely on these predictions for healthcare. Although this is a healthy number, 38% of these respondents will not rely on the predicted conditions for healthcare. These respondents had concerns about the accuracy and validity of the prediction. They also had concerns about the absence of a medical professional during this prediction stage. A detailed overview of these responses can be found in Appendix A2.

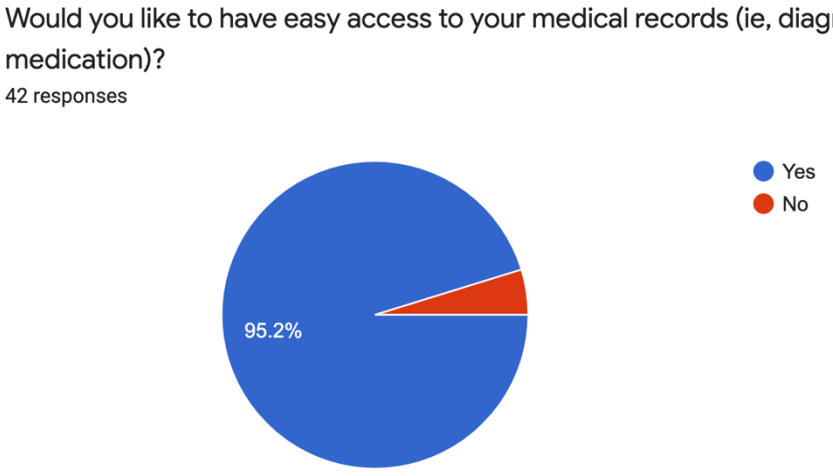


Figure 2.6: Respondents and whether they desire to have easy access to their medical records

In Figure 2.6, about 95% of respondents desired to have easy access to their medical records. This could be the case where they want some transparency, where they have direct access to these medical records.

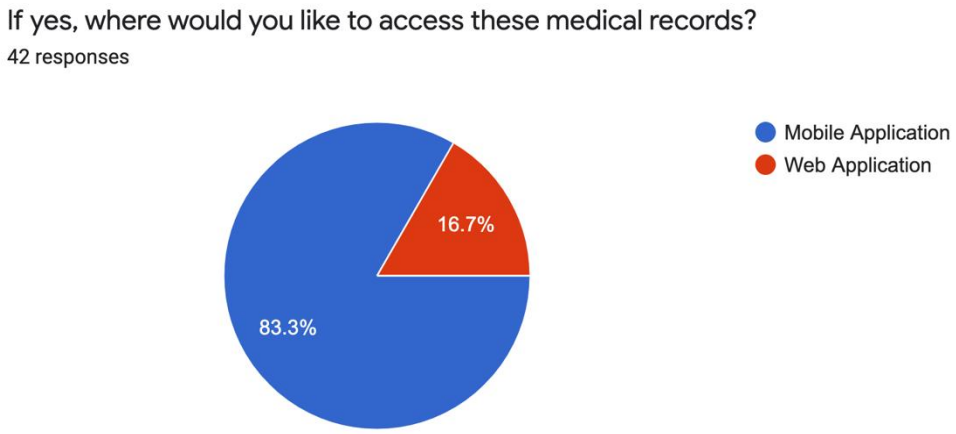


Figure 2.7: Respondents and whether they prefer a Mobile or Web Application

A healthy number of respondents (approximately 83%) opted for a solution that will give them access to their medical records via a Mobile Application. The reason for this choice could be the portability and functionality of the mobile application. These medical records can be accessed on a mobile application anywhere, at any time, whereas a Web Application will require the patient to keep a URL in memory. This usually, will deter patients from accessing the application via the web.

2.1.2 Doctor Questionnaire Analysis

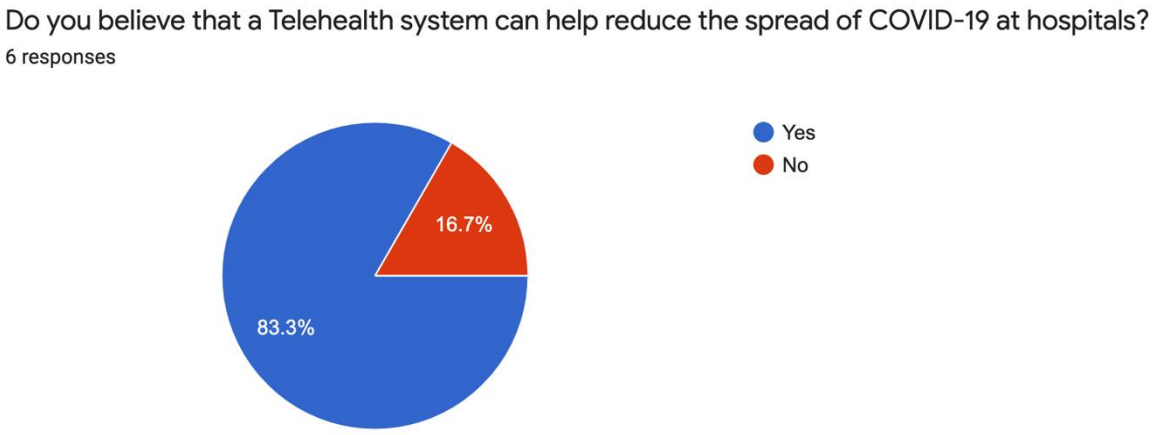


Figure 2.8: Doctors' view on a Telehealth system to reduce the spread of COVID-19

Out of the 6 responses gathered from doctors, approximately 83% of them indicated that a Telehealth system can help reduce the spread of COVID-19 at hospitals.

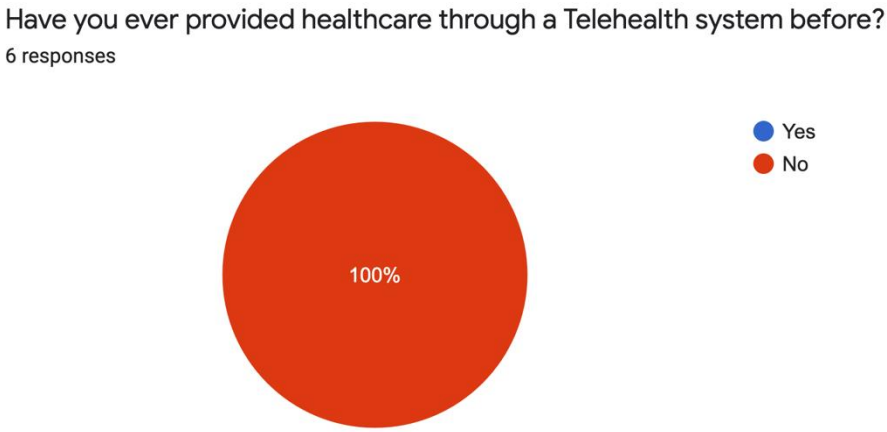


Figure 2.9: Doctors and whether they have provided healthcare through a Telehealth System

In Figure 2.9, all respondents indicated that they have never provided healthcare through a Telehealth system before.

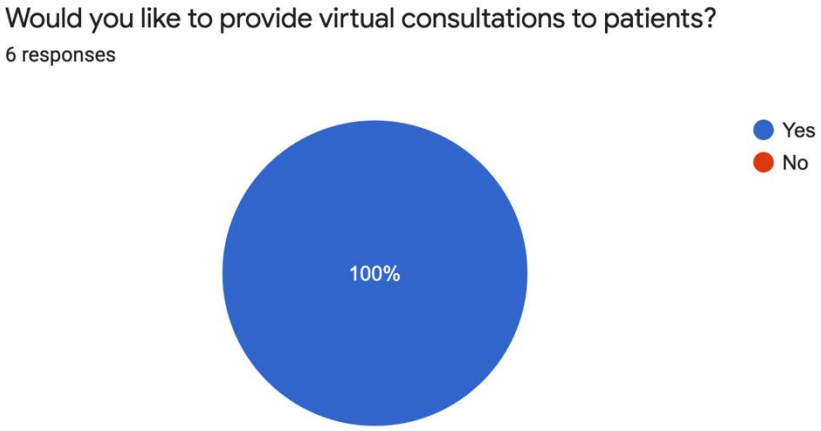


Figure 2.10: Doctors and their desire to provide virtual consultations to patients

Although the doctors in Figure 2.9 had not provided healthcare through a Telehealth system, the evidence from Figure 2.10 indicates that they will all like to offer virtual consultations to patients. This point further goes on to support their responses gathered in Figure 2.8.

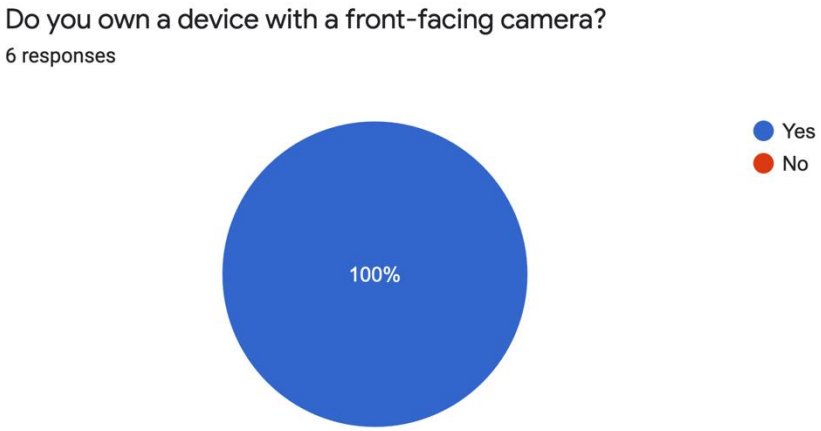


Figure 2.10: Ownership of a device with a front-facing camera among doctors

The results from Figure 2.10 suggest that 100% of the doctors interviewed own a device with a front-facing camera. This statistic supports their desire to provide virtual consultations to patients. The doctors would not face any difficulty in providing virtual consultation services to patients.

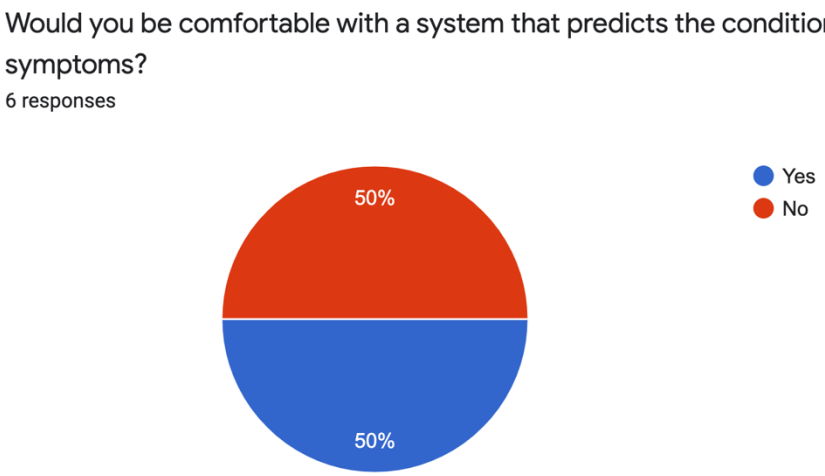


Figure 2.11: Doctors and their comfort level with a condition predictor

Figure 2.11 presents a split decision on how comfortable doctors are with a system that predicts conditions based on a patient's symptoms. Although there seems to be some positivity among 50% of the doctors the remaining 50% had concerns with the system providing accurate results. Some doctors feared that users of a condition predictor may not be literate enough to understand the results presented.

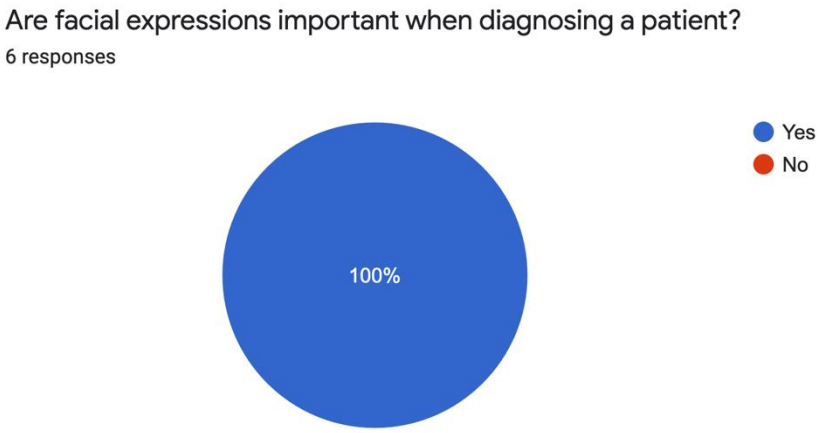


Figure 2.12: The importance of facial expressions to doctors

100% of the doctors stated that facial expressions are important when diagnosing a patient. This statistic could support the importance of a tool that could help doctors analyze the facial expression of a patient during consultations.

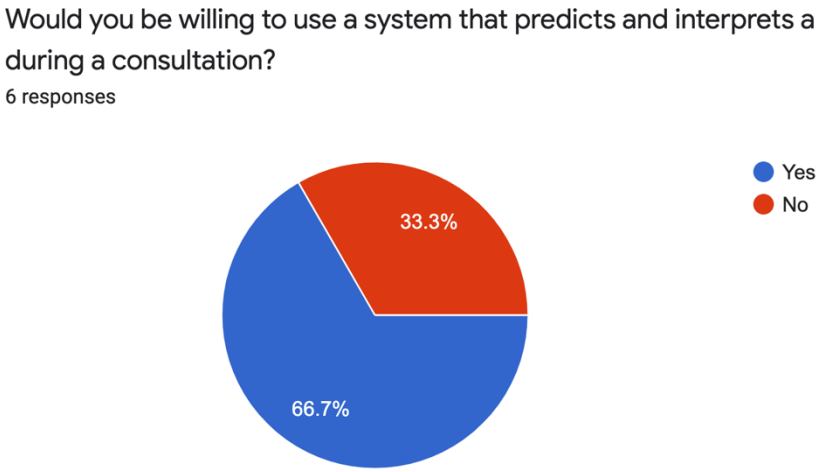


Figure 2.13: Willingness of doctors to use a tool that predicts and interprets a patient's facial expression

Although 100% of the doctors expressed the importance of facial expressions during the diagnosis of a patient in Figure 2.12, 66.7% of them were willing to use a tool that predicted and interpreted a patient's facial expression.

2.2 Functional Requirements

Functional Requirements describe the features that must be implemented in the Intelligent Telehealth system to allow users to meet their goals. The critical components of the Intelligent Telehealth system include the following:

- Medical Record Manager
- Computer Vision Component
- Symptom Checker

This section outlines the functional requirements of these critical components.

2.2.1 Medical Record Manager

The medical record manager is the component responsible for managing the medical records of patients. This component has two users, namely the doctor and the patient.

Doctor

The doctor is responsible for the management of a patient's medical records.

- The doctor must be able to create medical records.
- The doctor must be able to manage medical records.
- The doctor must be able to view a patient's medical records.

Patient

The patient is responsible for viewing their medical records.

- The patient must be able to view their medical records.

2.2.2 Computer Vision Component

The computer vision component is responsible for analyzing the facial expression of patients during virtual consultations. This component has the following functional requirements:

- The Computer Vision component must be able to analyze a patient's facial expression

during virtual consultations.

- The Computer Vision component must be able to record the analysis of a facial expression to a patient's medical record.
- The Computer Vision component must use the camera of a patient's existing device.

2.2.3 Symptom Checker

The symptom checker is responsible for predicting a patient's medical condition after conducting an interview. This component has the following functional requirements:

- The symptom checker must be able to conduct a symptom interview for patients.
- The symptom checker must be able to predict a patient's condition from the symptom interview.

2.3 Non-functional Requirements

Non-Functional Requirements define the attributes of the Telehealth system which are not directly related to the functions that the system must perform [6]. These requirements are defined as follows:

- Availability: The telehealth system would be required to be up and running at all times.
- Security: The telehealth system would be dealing with sensitive information such as the medical records of patients. Hence, it must be designed to prevent malicious attacks and unauthorized access.
- Usability: The telehealth system should follow design principles to ensure that users find the system easy to use.
- Performant: The computer vision component must be quick in returning the analysis of a patient's facial expressions. Furthermore, the complete system should be quick to respond under different workloads.

2.4 Use Case Diagram

The use case diagram defines the interactions between users and a system [6].

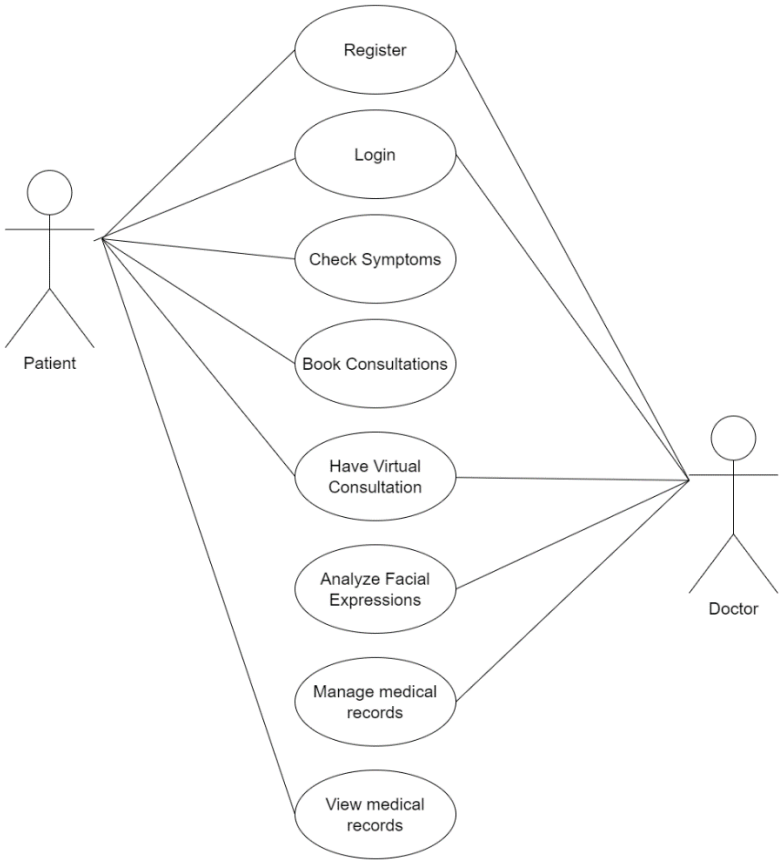


Figure 2.8: Use Case Diagram for the Telehealth system

This diagram defines the two groups of users of the Intelligent Telehealth system. Patients can check their symptoms, book consultations, and view medical records. Doctors can analyze facial expressions and manage medical records. Both users can register, login and have virtual consultations.

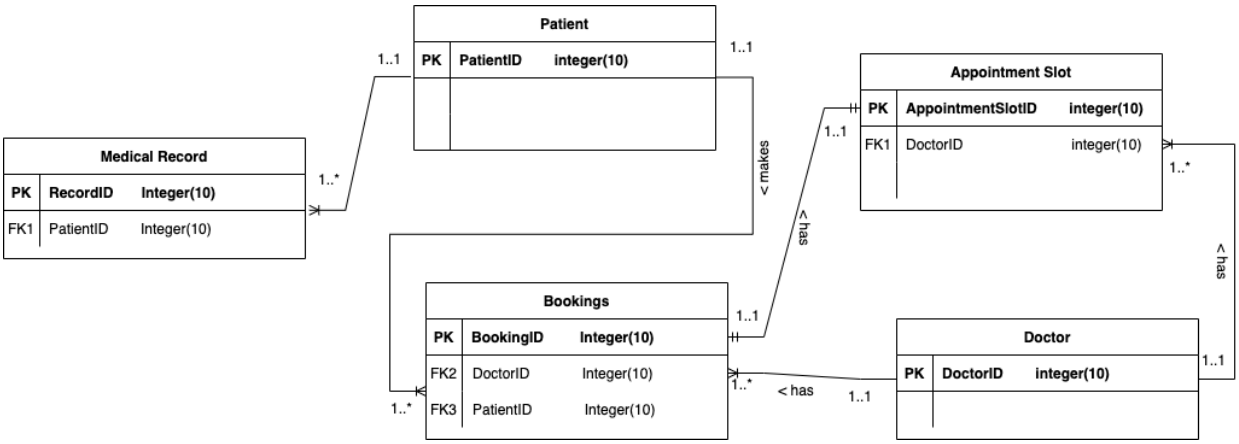


Figure 2.9: Entity-Relationship Diagram

The Entity-Relationship diagram in Figure 2.9 shows how the entities of the Telehealth system relate to each other. This diagram aided in the modeling and design of the relational database for the Telehealth system. The diagram, alongside the responses, gathered from the questionnaire, aided in the requirement specification for the Telehealth system.

2.5 Design Decisions

From the requirements defined above, the Intelligent Telehealth system consisted of a mobile application for patients and a web application for doctors. The mobile application facilitated the processes of the patients outlined in the use case diagram above, while the web application facilitated the processes of the doctors.

Chapter 3: Architecture and Design

This chapter outlines the architecture of the main components of the Intelligent Telehealth system. As stated in Chapter 2, the Intelligent Telehealth system consists of a mobile application and a web application. The architecture pattern chosen for both components is the Model View Controller architecture pattern.

3.1 Model-View-Controller

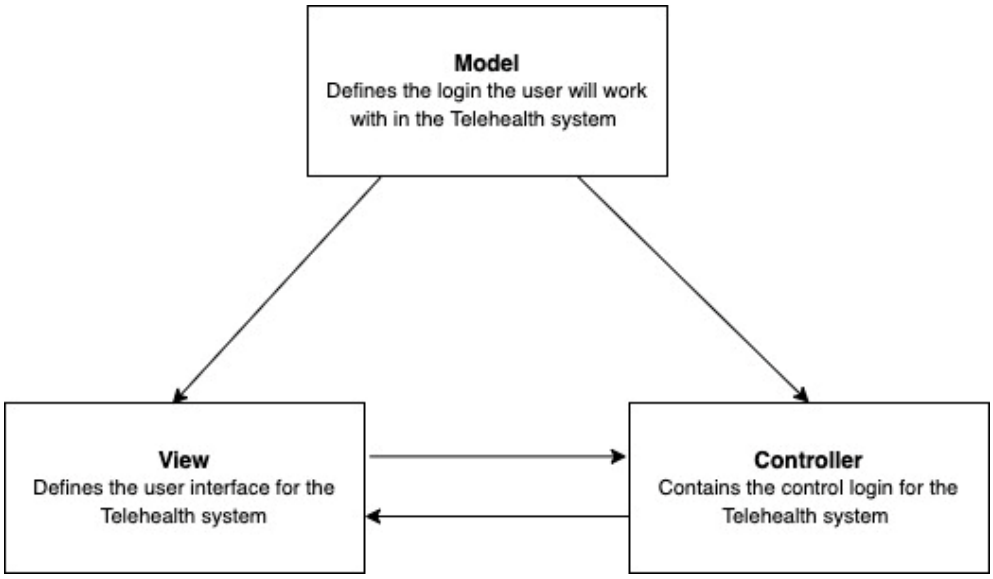


Figure 3.1: MVC Design Pattern

The Model-View-Controller pattern is an architectural pattern that separates presentation and interaction from the system data [6]. It is structured into three logical components that interact with each other [6]. These logical components are:

- Model
- View
- Controller

3.1.1 Model

The model manages the system data and associated operations on that data [6]. For the Intelligent Telehealth system, the model contains business logic and information on the data stored in the database of the system. For example, the model contains the business logic and information on the Medical Records entity of the Intelligent Telehealth system.

3.1.2 View

The view component defines and manages how the data is presented to the user [6]. For the Intelligent Telehealth system, the view describes how data and results from the business logic are displayed to both patients and doctors. For example, the view manages how Medical Records are presented to both the doctor and the patient.

3.1.3 Controller

The controller component manages user interaction and passes these interactions to the View and the Model [6]. For the Intelligent Telehealth system, the controller would interpret user input, and inform the model and the view to update if necessary. For example, the controller would interpret input from doctors on a patient's medical record and update the view and model if necessary.

3.1.4 Justification of Architecture

The Model-View-Controller pattern allows data to change independently of its representation [6]. Since the Model-View-Controller pattern allowed data to change independently, it made it easy to modify the data without worrying about breaking its representation.

Chapter 4: Implementation

This chapter outlines the functionalities that were implemented in the Intelligent Telehealth system. It provides information on the frameworks, programming languages and tools that were employed while developing the Intelligent Telehealth system.

4.1 Technology Stack

To develop the patient's mobile application for the Intelligent Telehealth system, React Native was employed. The mobile application consumed data provided by a REST API built using Django and the Django Rest Framework. The frames of a patient's video were sent to a server running the Face-API to return information on a patient's facial expression to the doctor.

The web application for doctors was also developed using Django and the natural language processing component of the Symptom Checker was developed using Python.

4.1.1 React Native

React Native is a JavaScript framework for building natively rendering mobile applications for iOS and Android [7]. React Native was chosen for this project because it provides the opportunity to build cross-platform applications from a single codebase thus reducing the time spend during development. React Native also translates markup to the native UI elements and therefore it was preferred to other cross-platform solutions which try to mimic the native UI elements.

4.1.2 Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design [8]. In this project, Django is used to build the REST API which serves the mobile application of the patients with data. It is also used to handle the views that handle the virtual consultations and the processes of the doctor's web application.

4.1.3 Django Rest Framework

The Django Rest Framework is an extension of Django, which allows for building Web APIs [9]. In this project, the Django Rest Framework is used to provide REST API views for serving the patient's mobile application.

4.1.4 Face-API

Face-API is a JavaScript API for face detection and face recognition in the browser implemented on top of the tensorflow.js core API [10]. Face-API is used in this project to detect the patient's face and their facial expressions.

4.1.5 Twilio Programmable Video

Twilio Programmable Video is a Software Development Kit which is built on WebRTC to provide video conferencing capabilities to applications [11]. Twiliio Programmable Video is the backbone of the Telehealth system's video conferencing capabilities. It provides API routes which allow the system to create rooms and verify a user's entry into a room.

4.1.6 PostgreSQL

PostgreSQL is a powerful, open-source object-relational database system with a strong reputation in reliability, feature robustness and performance [12]. In this project, PostgreSQL serves as the database for storing entities and their records.

4.1.7 Axios

With a small package and an extensible interface, Axios offers a simple to use library [13]. Axios boasts of a simple promise-based HTTP client for node.js and the browser. For this project it allowed for HTTP requests to be made to the Django backend.

4.2 Project File Structure

Django was used to develop the backend of the Intelligent Telehealth system. After the project was setup, the various applications within the Django project were created leading to the following project structure in Figure 4.1.

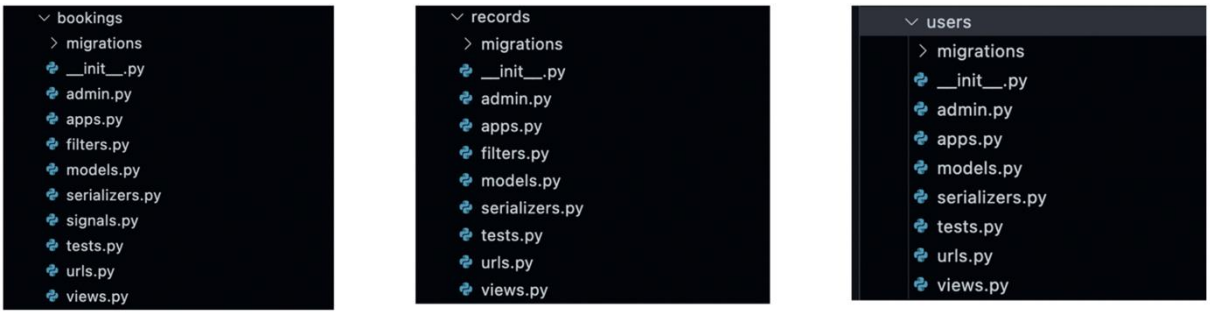


Figure 4.1: Project file structure for backend

Figure 4.1 shows the various applications within the Django project. To follow the MVC design pattern that was outlined in Chapter 3, each application was set up using a models.py file to handle the Model and a views.py file to handle the View and Controller aspect of the MVC design pattern.

4.3 Key Components

4.3.1 Booking Manager

The Booking component handles scheduling virtual consultations between doctors and patients. Fundamentally, the Booking Manager performs CRUD operations to Create, Retrieve, Update and Delete consultations. The process begins in the views.py file of the Booking Manager shown in Figure 4.2.

```python
# Create your views here.

class BookingList(ListCreateAPIView):
    queryset = Booking.objects.all()
    serializer_class = BookingSerializer
    filter_class = BookingFilter

    def list(self, request):
        queryset = self.get_queryset()
        serializer = BookingListSerializer(queryset, many=True, context={"request": request})
        return Response(serializer.data)

    def perform_create(self, serializer):
        patient = get_object_or_404(Patient, patient=self.request.user.id)
        serializer.save(patient=patient)

class BookingDetail(RetrieveUpdateDestroyAPIView):
    queryset = Booking.objects.all()
    serializer_class = BookingSerializer

    def retrieve(self, request, *args, **kwargs):
        instance = self.get_object()
        serializer = BookingListSerializer(instance)
        return Response(serializer.data)

    def perform_update(self, serializer):
        patient = get_object_or_404(Patient, patient=self.request.user.id)
        serializer.save(patient=patient)

class SlotList(ListCreateAPIView):
    queryset = AppointmentSlot.objects.all()
    serializer_class = SlotSerializer
    filter_class = SlotFilter

class SlotDetail(RetrieveUpdateDestroyAPIView):
    queryset = AppointmentSlot.objects.all()
    serializer_class = SlotSerializer
```

Figure 4.2: Booking Manager views.py

The views used in the Booking Manager, are BookingList for listing and creating bookings for a specific user, BookingDetail for retrieving, updating and deleting a Booking. Two other views which facilitate the processes of the Booking Manager are SlotList and SlotDetail. These views perform operations on the AppointmentSlot model which is used by a Booking to specify the time for a consultation.

When a booking is created, the associated AppointmentSlot is marked as close to prevent multiple bookings for the specific time.

The mobile application relies on specific routes to perform the operations of the Booking Manager. These routes are defined in the urls.py file shown in Figure 4.3

```python
from django.urls import path
from .views import BookingList, BookingDetail, SlotList, SlotDetail

urlpatterns = [
    path('', BookingList.as_view(), name='booking-list'),
    path('<int:pk>', BookingDetail.as_view(), name='booking-detail'),
    path('slots', SlotList.as_view(), name='slot-list'),
    path('slots/<int:pk>', SlotDetail.as_view(), name='slot-list')
]
```

Figure 4.3: Booking Manager urls.py

The mobile application accesses these routes using the Axios library and the respective HTTP request for the routes. These API calls are made in the bookings.ts file shown in Figure 4.4.

```typescript
export const getSlots = async ({start_gte, start_lte, doctor}: any) => {
  const slots = await client
    .get("bookings/slots", {
      params: {
        start_gte: start_gte,
        start_lte: start_lte,
        doctor: doctor,
        status: "OP",
      },
    })
    .then((res) => {
      return res.data;
    })
    .catch((error) => {
      if (error.response) {
        console.log(error.response.data);
      } else if (error.request) {
        console.log(error.request);
      } else {
        console.log("Error", error.messsage);
      }
      return [];
    });

  return slots;
};

export const getBookings = async (patient: any) => {
  const bookings = await client
    .get("bookings", { params: { patient: patient } })
    .then((res) => {
      return res.data;
    })
    .catch((error) => {
      if (error.response) {
        console.log(error.response.data);
      } else if (error.request) {
        console.log(error.request);
      } else {
        console.log("Error", error.messsage);
      }
      return [];
    });

  return bookings;
};
```

Figure 4.4: bookings.ts file for making API requests

Figure 4.5 shows the processes the user goes through when performing CRUD operations on a Booking.

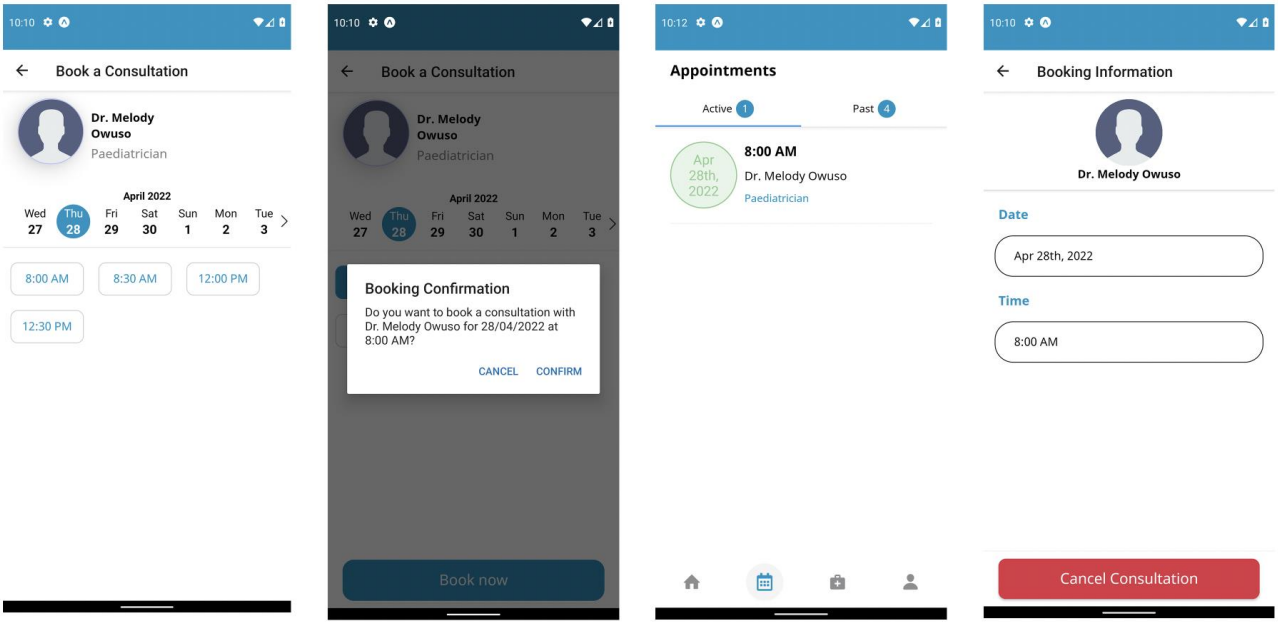


Figure 4.5: Appointment Booking flow

The user selects the slot for the doctor they would want to book a consultation with. The system alerts the user to confirm the time and date for the consultation. After confirming the consultation, the booking is added to the user's list of appointments. Should the need to cancel a consultation arise, the user can tap on an appointment and cancel the consultation.

4.3.2 Video Consultation

On the Django server, a CRON job runs every minute to determine when it's time for a consultation. This CRON job is shown in Figure 4.6.

```
def create_meeting_rooms():
    current_time = datetime.datetime.now().replace(second=0, microsecond=0)
    print(current_time)
    bookings = Booking.objects.filter(booking_slot__start_time = current_time).exclude(booking_status='CO')
    for booking in bookings:
        try:
            room_name = namegenerator.gen()
            twilio_room = create_room(room_name)
            booking.room_name = room_name
            booking.room_sid = str(twilio_room.sid)
            booking.save()
        except:
            print("failed to create room for:", booking.id)
```

Figure 4.6: CRON job for creating meeting rooms

It determines if it is time for a consultation by comparing the current time with the start time of bookings. After deriving the bookings for the current time, a random name is generated to serve as the name for the video consultation room. When the room name is generated, it is passed to a function which uses Twilio Programmable Video to create a room for video consultations on Twilio's servers. This function is shown in Figure 4.7.

```
def create_room(room_name):
    try:
        twilio_room = twilio_client.video.rooms.create(unique_name=room_name)
        return twilio_room
    except:
        print("failed to create room for booking")
```

Figure 4.7: Function to create a room on Twilio servers

If the room is created successfully, a room object with the room name and a unique id is returned. This room name and unique id are assigned to the Booking object, and it is saved. When the time for a consultation is due, a text message is sent to both patient and patient with a unique link to join the meeting room. When both user groups visit the link, the get function of the JoinRoomView is called. This function can be seen in Figure 4.8.

```python
class JoinRoom(View):

    def get(self, request, ref):
        booking = get_object_or_404(Booking, booking_reference=ref)
        room_name = booking.room_name
        room_sid = booking.room_sid
        room_data = {
            'room_name': room_name,
            'room_sid': room_sid
        }
        patient_name = booking.patient.patient.first_name
        doctor_name = booking.doctor.doctor.first_name
        email_form = VideoForm()
        request.session['room_data'] = room_data
        context = {'room_name': room_data['room_name'],
                   'room_sid': room_data['room_sid'],
                   'patient_name': patient_name,
                   'doctor_name': doctor_name,
                   'form': email_form }
        return render(request, 'video_chat/join.html', context)
```

Figure 4.8: Get function of JoinRoom class view

The get function, gets the related booking object, the room name and the room identifier. It also renders a form which would take the patient's and doctor's email. This form authenticates the users and makes a post request to the same view to perform the functions shown in Figure 4.9.

```python
def post(self, request, ref):
    booking = Booking.objects.get(booking_reference=ref)
    email_form = VideoForm(request.POST)
    if email_form.is_valid():
        user_email = email_form.cleaned_data['participant']
        if User.objects.filter(email = user_email).exists():
            room_data = request.session['room_data']
            room_name = room_data['room_name']
            user_token = get_access_token(room_data['room_name'])
            user = User.objects.get(email = user_email)
            request.session['booking_id'] = booking.id
            request.session['patient_id'] = booking.patient.id
            request.session['doctor_id'] = booking.doctor.id
            request.session['user_id'] = user.id
            request.session['is_doctor'] = user.is_doctor
            request.session['participant_data'] = {'token': user_token.to_jwt()}
            return redirect('video_chat:room-view', room_name)
        else:
            return redirect('video_chat:join-room', ref)
```

Figure 4.9: Post function of JoinRoom class view

The function shown in Figure 4.9, gets the associated booking and the email value. If the email does not exist, the user is redirected to the form to try again. If the email is valid, a token is generated using the function shown in Figure 4.10.

```
def get_access_token(room_name):
    access_token = twilio.jwt.access_token.AccessToken(account_sid, api_key, api_secret, identity=uuid.uuid4().int)
    video_grant = twilio.jwt.access_token.grants.VideoGrant(room=room_name)
    access_token.add_grant(video_grant)
    return access_token
```

Figure 4.10: Generate access token function

In Figure 4.10, an access token is generated using Twilio's token generator API. This token is required to access a room. The token is returned to the Post function in Figure 4.9 and stored in the session variable. After the required details are stored in the session variable, the view redirects to the room view. Figure 4.11 shows the function for the room view. This view gets the details from the session variable and passes it as a context to the HTML file that renders the video call screen.

```python
def room_view(request, room_name):
    room_data = request.session['room_data']
    participant_data = request.session['participant_data']
    patient_id = request.session['patient_id']
    doctor_id = request.session['doctor_id']
    booking_id = request.session['booking_id']
    is_doctor = request.session['is_doctor']
    doctor = Doctor.objects.get(id=doctor_id)
    patient = Patient.objects.get(id=patient_id)
    booking = Booking.objects.get(id=booking_id)
    context = {
        "context": {
            'room_name': room_name,
            'room_sid': room_data['room_sid'],
            'participant_token': participant_data['token'],
            'doctor': doctor_id,
            'patient': booking_id,
            'booking': booking_id,
            'is_doctor': 'true' if is_doctor else 'false'
        }
    }

    return render(request, 'video_chat/room.html', context)
```

Figure 4.11: Function for the room view

Figure 4.12 shows what users see upon successfully joining the consultation room.

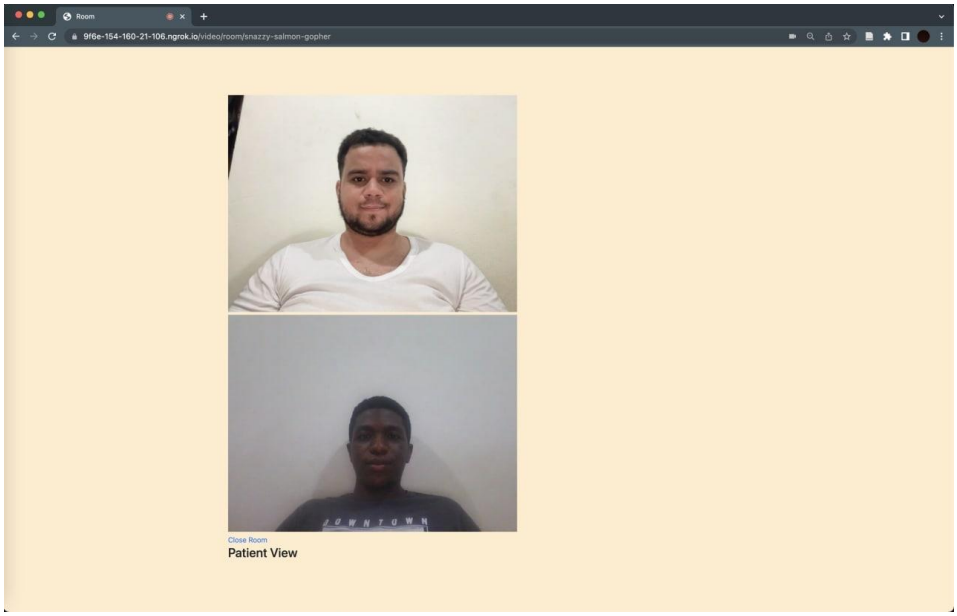


Figure 4.12: Video call screen with both patient and doctor

This video call screen is made possible via the processes on the Django backend and the Twilio JavaScript script shown in Figure 4.13 and a custom script to handle the video and audio tracks in Figure 4.14.

```
<script src="https://sdk.twilio.com/js/video/releases/2.15.2/twilio-video.min.js"></script>
```

Figure 4.13: Twilio SDK import

```
const container = document.getElementById("video-container");

document.addEventListener('DOMContentLoaded', async function(){
    const room = await joinVideoRoom();

    handleConnectedParticipant(room.localParticipant);
    room.participants.forEach(handleConnectedParticipant);
    room.on("participantConnected", handleConnectedParticipant);

    room.on("participantDisconnected", handleDisconnectedParticipant);
    window.addEventListener("pagehide", () => room.disconnect());
    window.addEventListener("beforeunload", () => room.disconnect());
}, false)

const handleConnectedParticipant = (participant) => {
    const participantDiv = document.createElement("div");
    participantDiv.setAttribute("id", participant.identity);
    container.append(participantDiv);

    participant.tracks.forEach((trackPublication) => {
        handleTrackPublication(trackPublication, participant);
    });

    participant.on("trackPublished", handleTrackPublication)
};

const handleTrackPublication = (trackPublication, participant) => {
    function displayTrack(track) {

    const participantDiv = document.getElementById(participant.identity);

      participantDiv.append(track.attach());
    }

    if (trackPublication.track) {
      displayTrack(trackPublication.track);
    }

    trackPublication.on("subscribed", displayTrack);
  };


const handleDisconnectedParticipant = (participant) => {
    participant.removeAllListeners();
    const participantDiv = document.getElementById(participant.identity);
    participantDiv.remove()
}
```

Figure 4.14: Custom function to handle

4.3.3 Computer Vision

One of the functional requirements of the system was to detect the facial expression of a patient during consultation. This requirement was fulfilled using Face-API. Face-API relies on face detection models to detect faces in the browser. For this project, the models that were implemented were Tiny Face Detector, 68 Point Face Landmark Detection Models, Face Recognition Model and Face Expression Model.

Tiny Face Detector

Tiny Face Detector is a performant, realtime face detector, which is extremely mobile and web friendly [10]. This model was chosen as the Telehealth system must be performant.

68 Point Face Landmark Detection Models

This package is a very lightweight and fast, yet accurate 68 point face landmark detector [10]. It can detect 68 facial landmarks on a face.

Face Recognition Model

This model implements a ResNet-34 like architecture to compute a face descriptor from any given face image [10]. The Face Recognition Model is important because it can detect a face without the need for a training set.

Face Expression Recognition Model

The face expression recognition model is lightweight, fast and provides reasonable accuracy [10]. To identify facial expressions, the Face-API uses all the models that have been described above. These models are stored locally as shown in Figure 4.15.

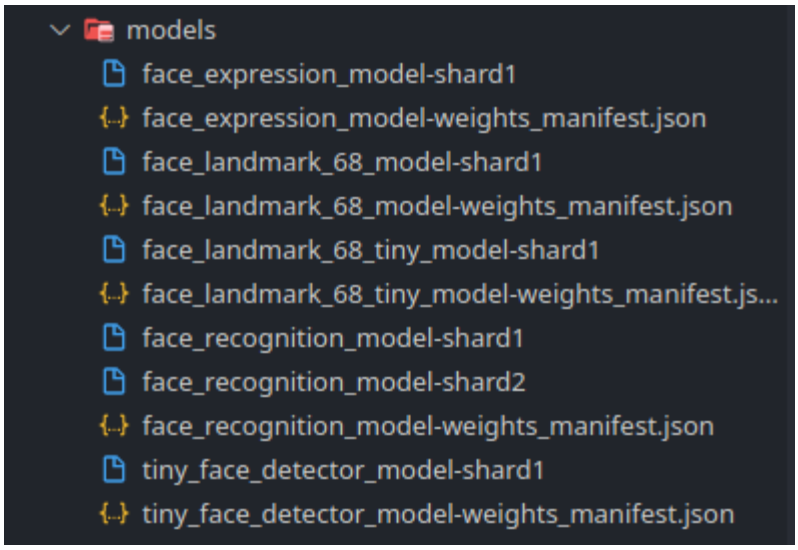


Figure 4.15: Face-API models stored locally

To make use of these models, they are loaded from the disk and initialized with Face-API. Figure 4.16 shows the process of loading these models and initializing them.

```
Promise.all([
  faceapi.nets.tinyFaceDetector.loadFromUri(MODEL_URL),
  faceapi.nets.faceLandmark68Net.loadFromUri(MODEL_URL),
  faceapi.nets.faceRecognitionNet.loadFromUri(MODEL_URL),
  faceapi.nets.faceExpressionNet.loadFromUri(MODEL_URL),
]);
```

Figure 4.16: Initializing the Face-API models

After initializing the models, the patient's video track is derived from the Document Object Model (DOM) as shown in Figure 4.17.

```
async function doMagic() {
  if (context.is_doctor == "true") {
    const parent_div = document.querySelector('.magic-container');
    const patient_view = parent_div.lastChild;
    var input = '';

    var children = patient_view.children;
    for (let index = 0; index < children.length; index++) {
        const element = children[index];
        if(element.tagName == 'VIDEO'){
            input = element;
        }
    }
    console.log(input);
    // const input = patient_view.lastChild;
    input.setAttribute("id", "patient_video");
    const video = document.getElementById("patient_video");
    detectFaces(video);
  } else {
    console.log("not a doctor");
  }
};
```

Figure 4.17: Finding the patient's video track

After the patient's video track is derived, the video track is passed to a custom function which triggers the Face-API to detect the patient's facial expressions. This function is shown in Figure 4.18.

```
function detectFaces(input) {
  const canvas = faceapi.createCanvasFromMedia(input);
  document.body.append(canvas);
  const displaySize = { width: input.videoWidth, height: input.videoHeight };
  faceapi.matchDimensions(canvas, displaySize);
  setInterval(async () => {
    const detections = await faceapi
      .detectAllFaces(input, new faceapi.TinyFaceDetectorOptions())
      .withFaceExpressions();
    const resizedDetections = faceapi.resizeResults(detections, displaySize);
    canvas.getContext("2d").clearRect(0, 0, canvas.width, canvas.height);
    faceapi.draw.drawDetections(canvas, resizedDetections);
    faceapi.draw.drawFaceExpressions(canvas, resizedDetections);
  }, 500);
}
```

Figure 4.18: Detecting a facial expression

In Figure 4.18, a canvas is created to house the facial expressions. Once the canvas has been created, the canvas is added to the HTML. Every 500 milliseconds, it detects a patient's face and

gives a prompt on the facial expression detected. Figure 4.19 to Figure 4.22 show different facial expressions identified on a face.

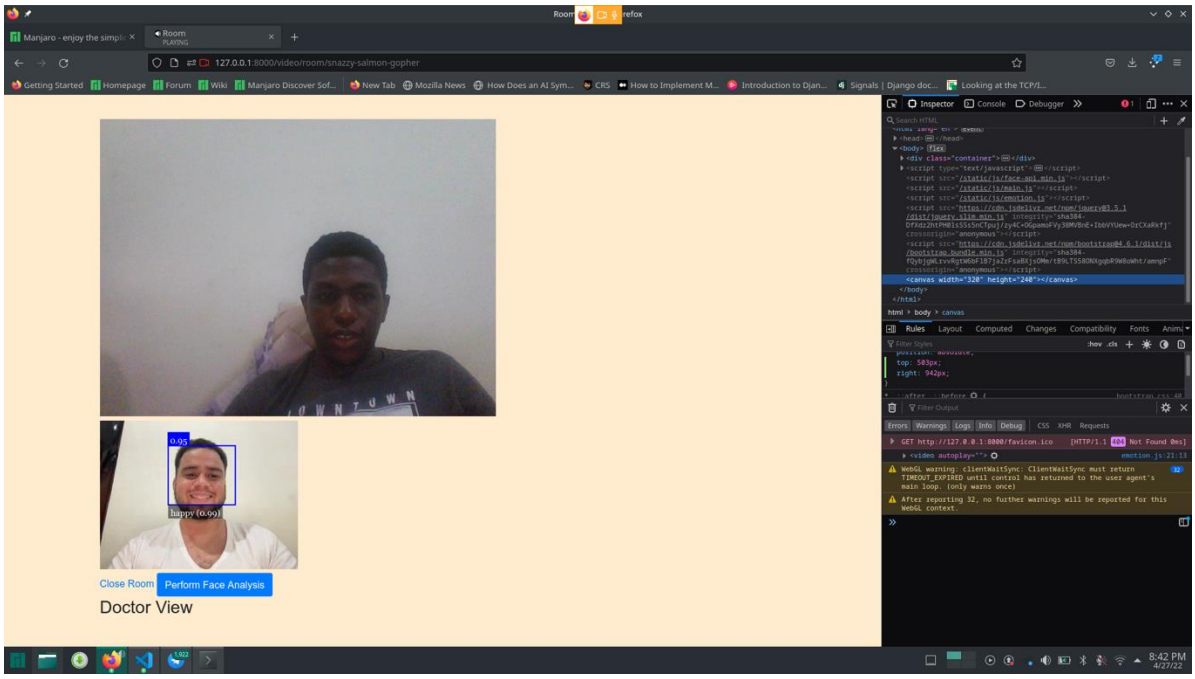


Figure 4.19: Happy facial expression detected on a patient's face

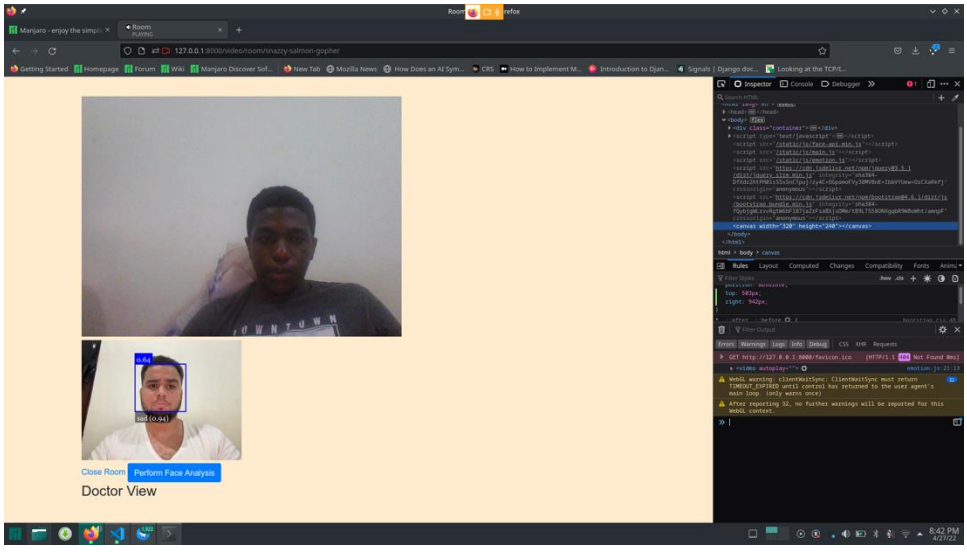


Figure 4.20: Sad facial expression detected on a patient's face

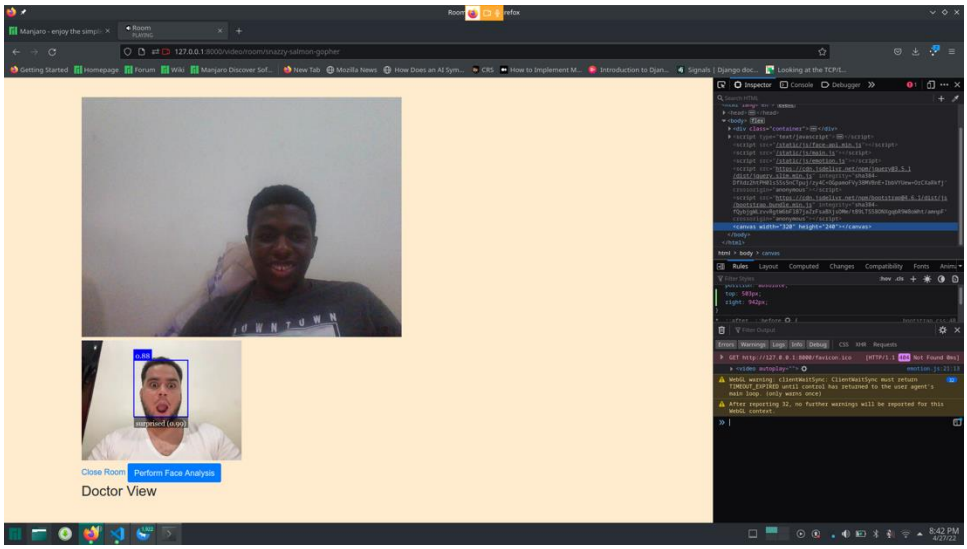


Figure 4.21: Surprised facial expression detected on a patient's face

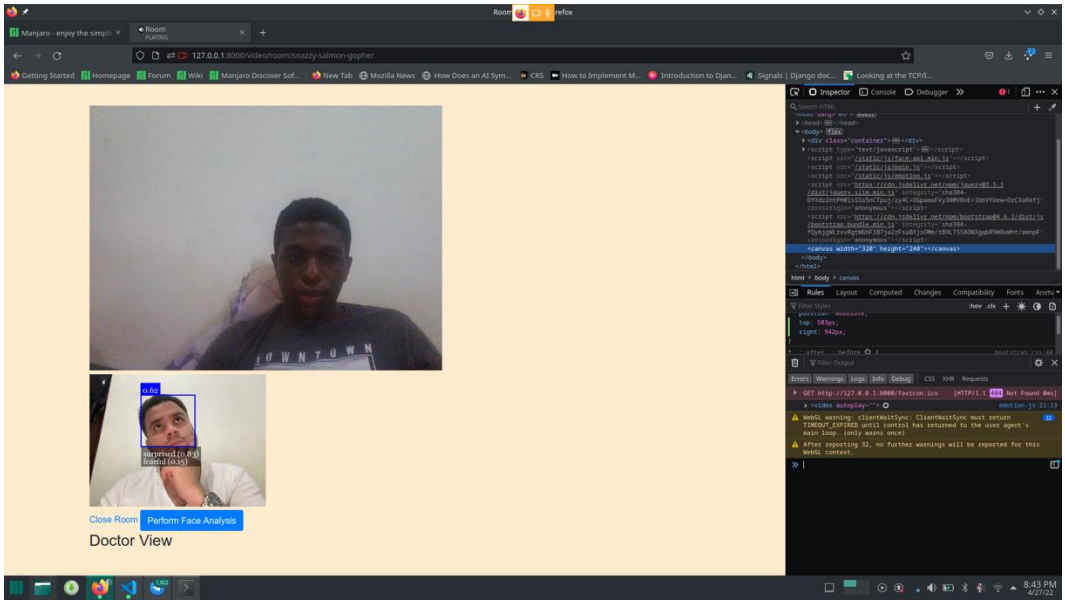


Figure 4.22: Surprised and fearful facial expressions detected on a patient's face

Chapter 5: Testing and Results

This chapter outlines the various tests that were executed to verify that the Intelligent Telehealth system met all requirements and that the various functionalities worked as expected. The following tests were executed:

- Unit Testing
- User Testing

5.1 Unit Testing

Unit testing is the process of testing program components, such as methods or object classes [6].

5.2 User Testing

User testing is the stage in the testing process in which stakeholders involved in the project provide input and advice on the system [6]. The questionnaire in Appendix A3 was used on a group of 3 patients to evaluate the performance of the Telehealth System. Table 5.1 provides a summary of the scores given to each question by the patients.

Table 5.1: Summary of User Testing

Patients	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**
Patient 1	4	4	5	4	5
Patient 2	4	5	5	5	5
Patient 3	3	4	4	4	4

Generally, the feedback was good from all testers. Issues arose with how the video call screen looked on some screen sizes however the users appreciated the effectiveness of the facial analysis.

Chapter 6: Conclusion and Recommendations

This chapter outlines challenges that were faced during the project's lifecycle and some recommendations that can be implemented to improve the system's performance.

Functional Requirements	**Status**
The doctor must be able to create medical records	Achieved
The doctor must be able to manage medical records	Achieved
The doctor must be able to view a patient's medical record	Achieved
The patient must be able to view their medical records	Achieved
The Computer Vision component must be able to analyze a patient's facial expression during virtual consultations	Achieved
The Computer Vision component must be able to record the analysis of a facial expression to the patient's medical record	Not Achieved
The Computer Vision component must use the camera of a patient's existing device	Achieved
The Symptom Checker must be able to conduct a symptom interview for patients	Not achieved
The Symptom Checker must be able to predict a patient's condition from the symptom interview	Not achieved

6.1 Challenges

One of the major challenges faced during the development of the Computer Vision component of the Intelligent Telehealth system was the lack of a commercial body language dataset for building

a custom model. The only solid body language dataset was available for research purposes and had limited access. This predicament forced the use of an already built API for analyzing facial expressions.

6.2 Recommendations for Future Work

Healthcare is a cycle and does not end after one visit. Currently, the system has no feature in place to handle the cyclical nature of healthcare. Going forward, it is recommended to emulate this healthcare cycle. Doctors should be given the opportunity to recommend what the next stage would be after a consultation. Some of these recommendations could include prescribing medication, referring the patient to another hospital, or asking the patient to come in for a physical consultation. A follow-up feature could also be implemented so doctors can follow up on their patients.

Furthermore, the experience of doctors could be improved by perfecting the doctor web application. The symptom checker could also be implemented to support doctors during consultations.

References

[1] Elham Monaghesh and Alireza Hajizadeh. 2020. The role of telehealth during COVID-19 outbreak: a systematic review based on current evidence. *BMC Public Health* 20, (August 2020), 1193. DOI:https://doi.org/10.1186/s12889-020-09301-4

[2] David Oliver. 2021. David Oliver: Could we do better on hospital acquired covid-19 in a future wave? *BMJ* 372, (January 2021), n70. DOI:https://doi.org/10.1136/bmj.n70

[3] Urmil Bharti, Deepali Bajaj, Hunar Batra, Shreya Lalit, Shweta Lalit, and Aayushi Gangwani. 2020. Medbot: Conversational Artificial Intelligence Powered Chatbot for Delivering Tele-Health after COVID-19. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 870–875. DOI:https://doi.org/10.1109/ICCES48766.2020.9137944

[4] J. E. García and R. A. Torres. 2013. Telehealth mobile system. In *2013 Pan American Health Care Exchanges (PAHCE)*, 1–5. DOI:https://doi.org/10.1109/PAHCE.2013.6568315

[5] Katherine A. Karl, Joy V. Peluchette, and Navid Aghakhani. 2022. Virtual Work Meetings During the COVID-19 Pandemic: The Good, Bad, and Ugly. *Small Group Research* 53, 3 (June 2022), 343–365. DOI:https://doi.org/10.1177/10464964211015286

[6] Ian Sommerville. 2011. Software engineering (9th ed ed.). Pearson, Boston.

[7] React Native. Retrieved from https://reactnative.dev/.

[8] The web framework for perfectionists with deadlines. Django. Retrieved from https://www.djangoproject.com/

[9] Django REST framework. Retrieved from https://www.django-rest-framework.org/

[10] face-api.js. Retrieved from https://justadudewhohacks.github.io/face-api.js/docs/index.html

[11] Twilio Video. Twilio Docs. Retrieved from https://www.twilio.com/docs/video

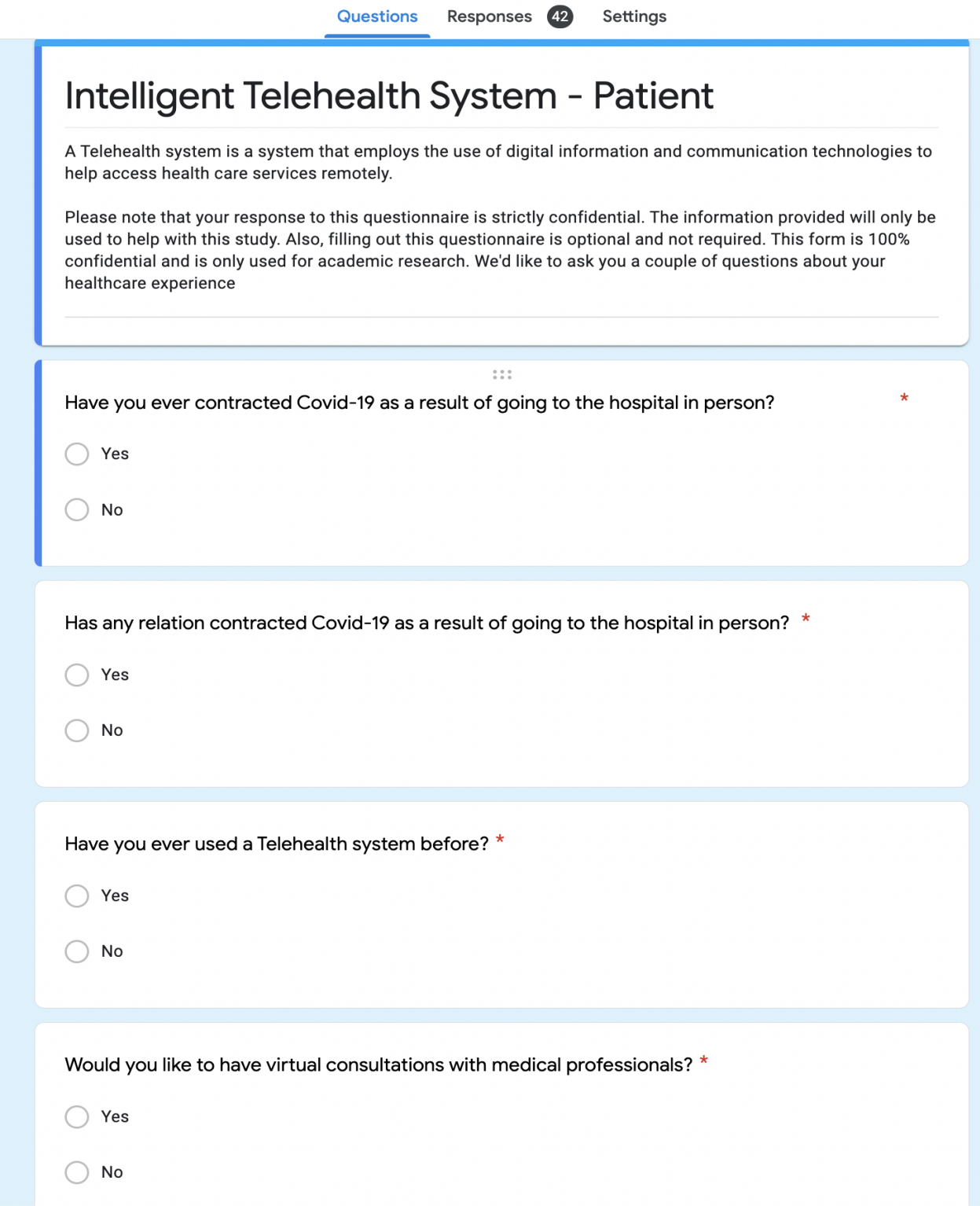
[12] PostgreSQL: The World's Most Advanced Open Source Relational Database. PostgreSQL. Retrived from https://www.postgresql.org/

[13] Getting Started. Axios. Retrieved from https://axios-http.com/docs/intro

Appendix

Appendix A [Requirements Gathering]

A1 [Patient Questionnaire]



Do you own a smartphone with a front-facing camera? *

- Yes
- No

Would you be comfortable with a system that predicts your condition based on symptoms you provide? *

- Yes
- No

Would you rely on these predictions for healthcare? *

- Yes
- No

If no, why would you not rely on these predictions?

Long-answer text

Would you like to have easy access to your medical records (ie, diagnosis, recommendation and medication)? *

- Yes
- No

If yes, where would you like to access these medical records?

- Mobile Application
- Web Application

A2 [Patient Questionnaire Result]

If no, why would you not rely on these predictions?

13 responses

- Trust issues for the technology
- If it is not approved by a medical professional, i may have some trust issues in relying on them
- Might not be very accurate
- The predictions might not be accurate and might give me false predictions
- How valid are these predictions? Would they mislead me to believe a condition I may not have?
- what kind of predictions will they be? would they have results to complex diagnosis?
- I do not believe that the system is intelligent enough to make such predictions.
- What kind of predictions will this system give?
- I dont trust these systems
- The predictions may not always point towards the right condition
- It might not be the actual health problem
- Cause it might malfunction
- Because you said prediction and I don't trust that words
  Its not sure

A2 [Doctor Questionnaire]

Intelligent Telehealth System - Doctor

A Telehealth system is a system that employs the use of digital information and communication technologies to help access health care services remotely.

Please note that your response to this questionnaire is strictly confidential. The information provided will only be used to help with this study. Also, filling out this questionnaire is optional and not required. This form is 100% confidential and is only used for academic research. We'd like to ask you a couple of questions about your healthcare experience

Have you ever contracted COVID-19 in the line of duty? *

- ◯ Yes
- ◯ No

Has any relation contracted COVID-19 as a result of going to the hospital in person? *

- ◯ Yes
- ◯ No

Do you believe that a Telehealth system can help reduce the spread of COVID-19 at hospitals? *

- ◯ Yes
- ◯ No

Have you ever provided healthcare through a Telehealth system before? *

- ◯ Yes
- ◯ No

If yes, how was your experience?

Long-answer text

Would you like to provide virtual consultations to patients? *

- Yes
- No

Do you own a device with a front-facing camera? *

- Yes
- No

Would you be comfortable with a system that predicts the condition of patients based on their symptoms? *

- Yes
- No

If no, what concerns do you have with the system predicting conditions?

Long-answer text

Are facial expressions important when diagnosing a patient? *

- Yes
- No

Would you be willing to use a system that predicts and interprets a patient's facial expression during a consultation? *

- Yes
- No

During a consultation, what information is taken when filling a patient's medical record?

Long-answer text

A3 [User Testing Questionnaire]

1. How was your interaction with the interface of the Telehealth System?
2. Was it easy to navigate through the Telehealth system?
3. Did the application achieve the goal of providing a Telehealth system to you?
4. How likely are you to use the Telehealth system in the future?
5. How likely are you to recommend the Telehealth system?