Doctor's waiting room assistant robot

Team #Engineers

Bakhtiyorov Firdavs, Gulomov Nozimjon, Laura Cesar, Dadajonov Bakhtiyorjon, Mohamed Alhammadi

Jump Together, Fly Farther!

School of Global Convergence Studies

인하대학교 국제학부

Final Presentation & Demo

June 13, 2024

2024-1 VIP Course

Professors: Mehdi Pirahandeh, Kakani Vijay, and Serrao Pruthvi Loy Rozario

Contents

Team contributions

- **Firdavs_12214762**
  - **Team leader,Software Developer, Hardware Specialist** *(Developing codes, implementing interaction logic and database management. Ensuring smooth communication between hardware and software.)*
- **Nozimjon_12204507**
  - **UI/UX & 3D Modeling Designer***(Iterating on the design to improve user experience.Creating visual assets and final presentation.3D Modeling)*
- **Laura_122400022**
  - **Software Developer***(Integrating AI models and algorithms into software applications to solve complex problems and enhance user experiences.)*
- **Bakhtiyorjon_12200288**
  - **Testing and Quality Assurance, database** *(Analyzing feedback and performance data.Ensuring the system meets all functional and performance requirements.)*
- **Alhammadi_12200182**
  - **Educating team members and users about data protection practices.**

Project Overview

Doctor’s waiting room assistant robot

Our project is the development of a Doctor’s Waiting Room Assistant Robot using a Raspberry Pi. The robot enhances patient experience and using facial recognition for patient check-ins, providing information, and streamlining the waiting process.

Existing Problems

- Long wait times and inefficient check-in processes.
- Limited patient access to information.
- Overburdened healthcare staff with administrative tasks.

Possible Solutions

- Automated check-ins and patient notifications.
- Improved information accessibility.

Practical Influences

- Higher patient satisfaction and reduced operational costs.
- Improved staff efficiency and regulatory compliance

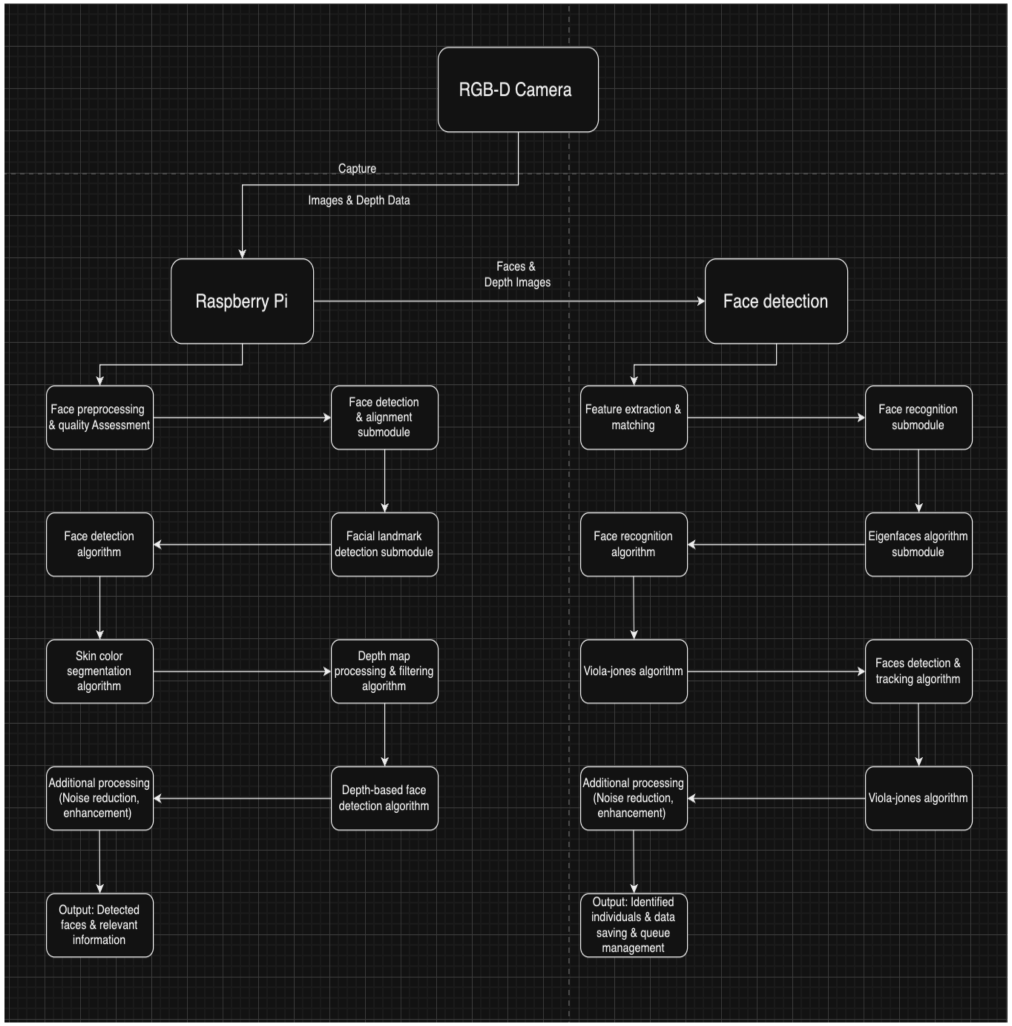
Project Overview

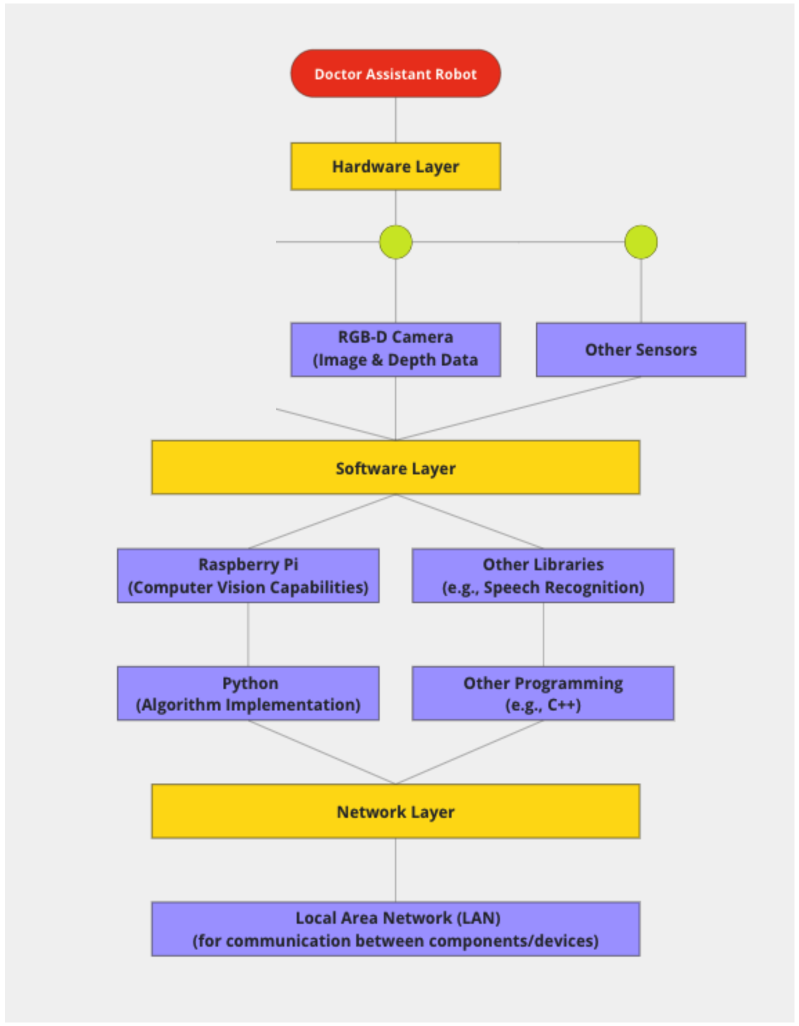
Inspiration

Main Goals

- Reduce patient wait times.
- Provide information and guidance to patients.
- Improve the overall efficiency of the waiting room.
- Enhance patient satisfaction and comfort.
- Provide valid database for institutions

System Architecture

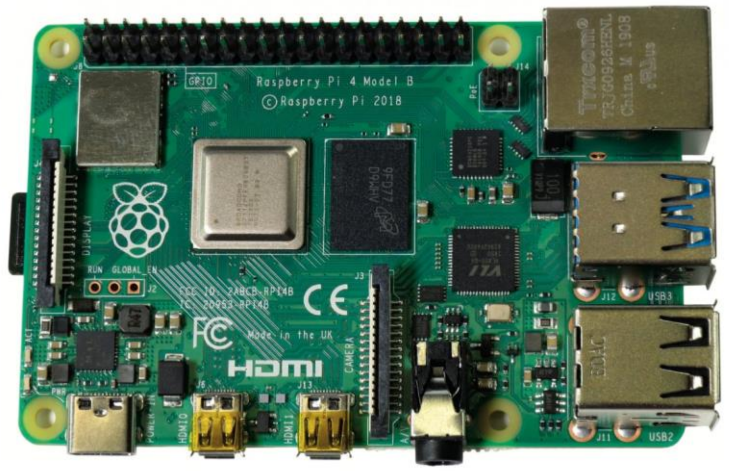

RGB-D Camera
Capture
Images & Depth Data
Raspberry Pi
Faces & Depth Images
Face detection
Face preprocessing & quality Assessment
Face detection & alignment submodule
Feature extraction & matching
Face recognition submodule
Face detection algorithm
Facial landmark detection submodule
Face recognition algorithm
Eigenfaces algorithm submodule
Skin color segmentation algorithm
Depth map processing & filtering algorithm
Viola-jones algorithm
Faces detection & tracking algorithm
Additional processing (Noise reduction, enhancement)
Depth-based face detection algorithm
Additional processing (Noise reduction, enhancement)
Viola-jones algorithm
Output: Detected faces & relevant information
Output: Identified individuals & data saving & queue management


Doctor Assistant Robot
Hardware Layer
RGB-D Camera (Image & Depth Data
Other Sensors
Software Layer
Raspberry Pi (Computer Vision Capabilities)
Other Libraries (e.g., Speech Recognition)
Python (Algorithm Implementation)
Other Programming (e.g., C++)
Network Layer
Local Area Network (LAN) (for communication between components/devices)

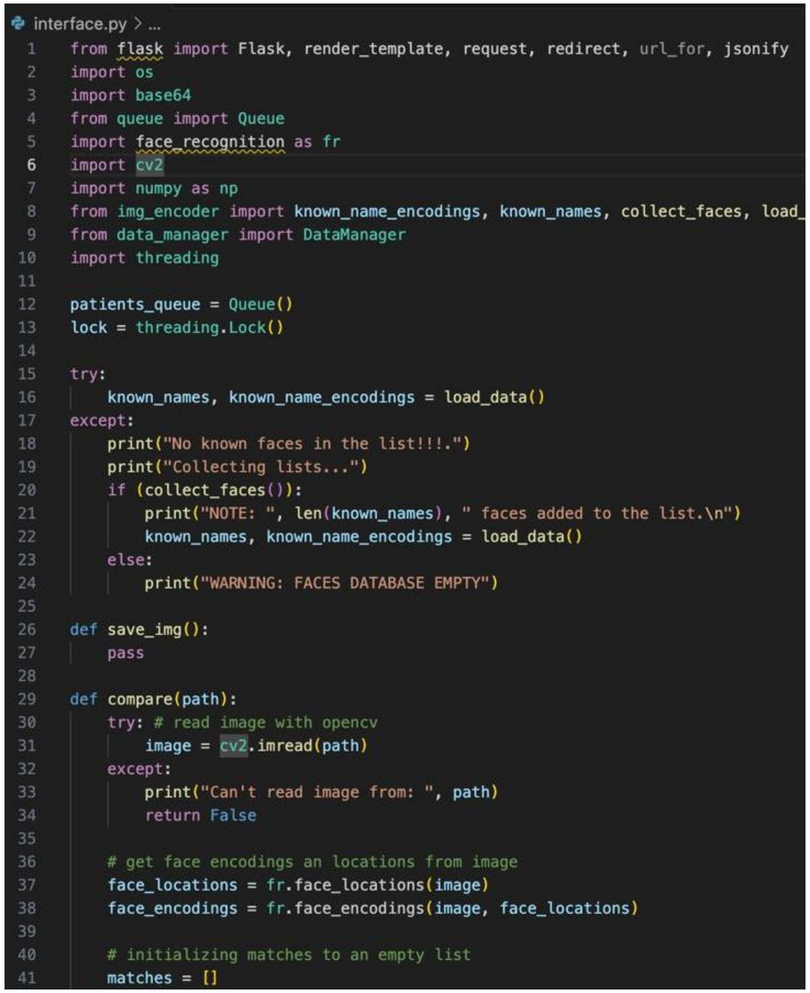
Future Enhancements

- **“Advanced AI”:**
  More personalized patient interactions.
- **Integration with Wearables:**
  Syncing with health devices for real-time monitoring.
- **“Enhanced Voice Recognition”:**
  Improved voice command accuracy.
- **“Mobility”:**
  Enhanced autonomous navigation for better interaction throughout the waiting room and potentially other areas.
- **“Expansion”:**
  Deploying the robot in other hospital departments.

Hardware Implementation



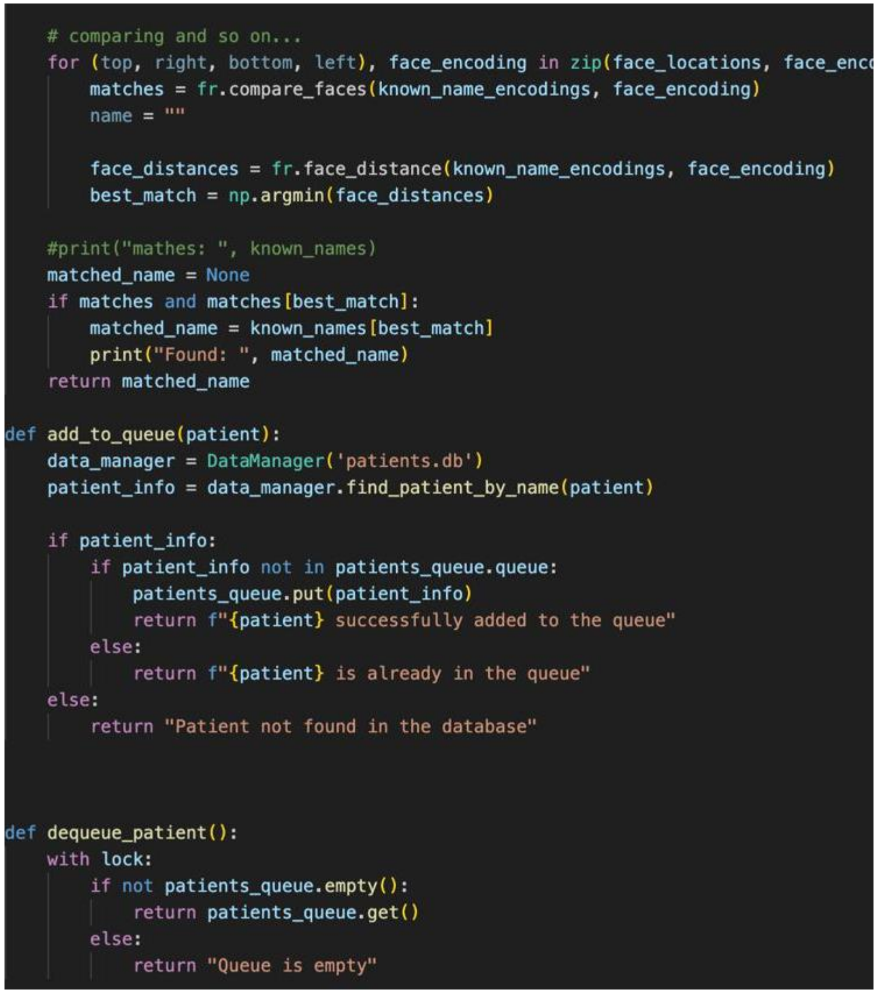
Software Development



interface.py > ...

```python
from flask import Flask, render_template, request, redirect, url_for, jsonify
import os
import base64
from queue import Queue
import face_recognition as fr
import cv2
import numpy as np
from img_encoder import known_name_encodings, known_names, collect_faces, load_
from data_manager import DataManager
import threading

patients_queue = Queue()
lock = threading.Lock()

try:
    known_names, known_name_encodings = load_data()
except:
    print("No known faces in the list!!!.")
    print("Collecting lists...")
    if (collect_faces()):
        print("NOTE: ", len(known_names), " faces added to the list.\n")
        known_names, known_name_encodings = load_data()
    else:
        print("WARNING: FACES DATABASE EMPTY")

def save_img():
    pass

def compare(path):
    try: # read image with opencv
        image = cv2.imread(path)
    except:
        print("Can't read image from: ", path)
        return False

    # get face encodings an locations from image
    face_locations = fr.face_locations(image)
    face_encodings = fr.face_encodings(image, face_locations)

    # initializing matches to an empty list
    matches = []
```

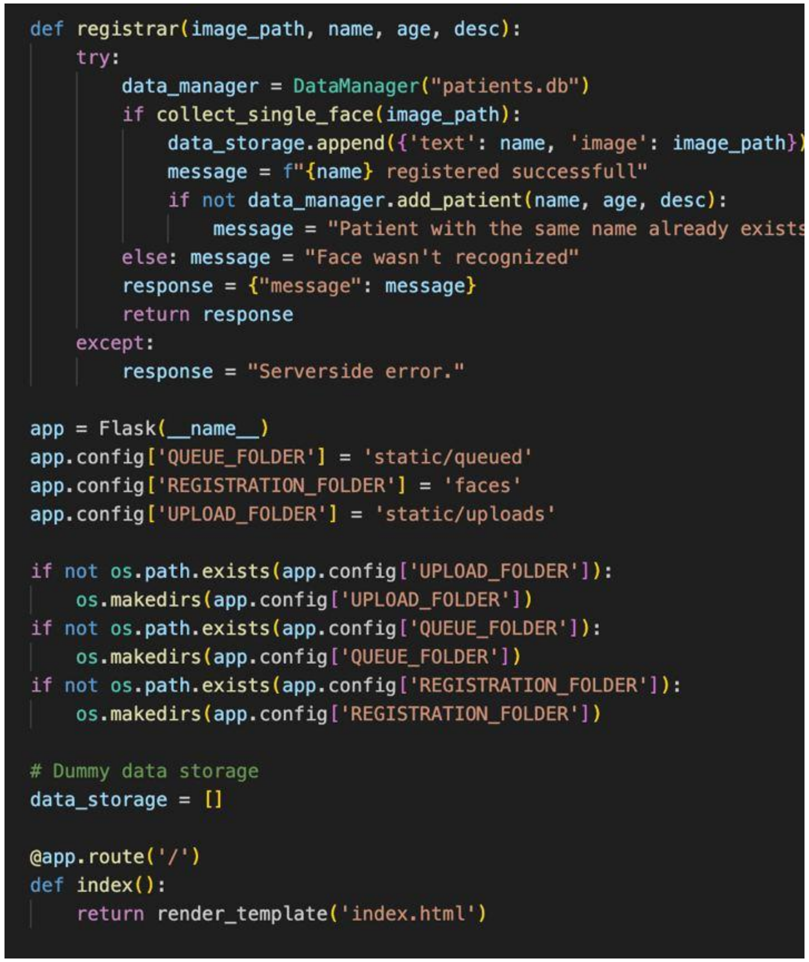
#1



```python
    # comparing and so on...
    for (top, right, bottom, left), face_encoding in zip(face_locations, face_enc
        matches = fr.compare_faces(known_name_encodings, face_encoding)
        name = ""

        face_distances = fr.face_distance(known_name_encodings, face_encoding)
        best_match = np.argmin(face_distances)

    #print("mathes: ", known_names)
    matched_name = None
    if matches and matches[best_match]:
        matched_name = known_names[best_match]
        print("Found: ", matched_name)
    return matched_name

def add_to_queue(patient):
    data_manager = DataManager('patients.db')
    patient_info = data_manager.find_patient_by_name(patient)

    if patient_info:
        if patient_info not in patients_queue.queue:
            patients_queue.put(patient_info)
            return f"{patient} successfully added to the queue"
        else:
            return f"{patient} is already in the queue"
    else:
        return "Patient not found in the database"


def dequeue_patient():
    with lock:
        if not patients_queue.empty():
            return patients_queue.get()
        else:
            return "Queue is empty"
```

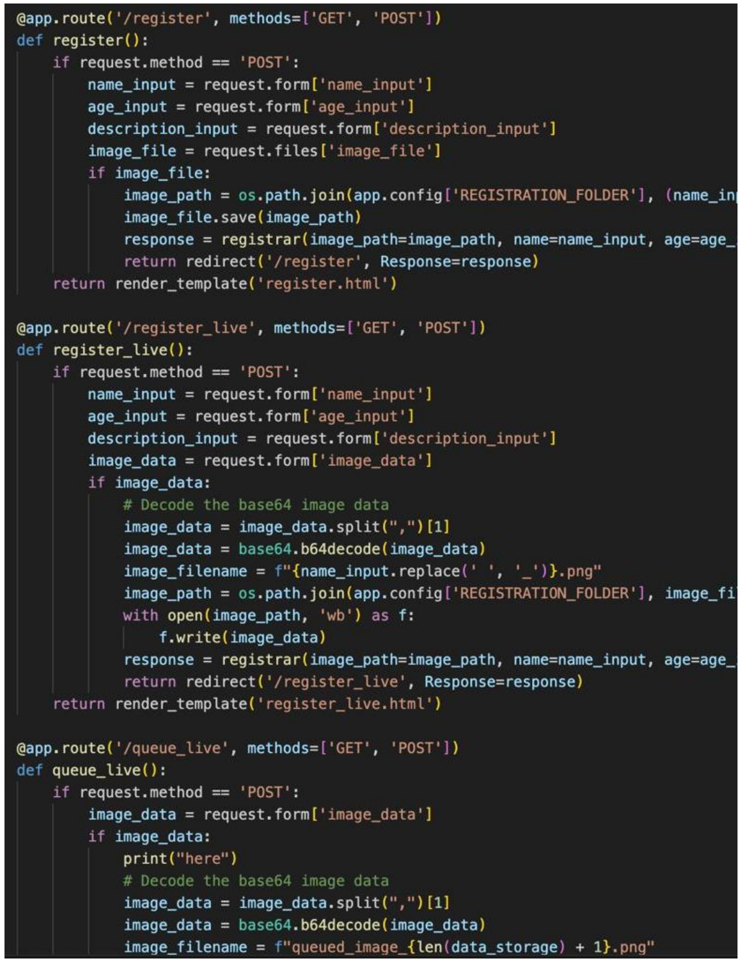
#2

Software Development



```python
def registrar(image_path, name, age, desc):
    try:
        data_manager = DataManager("patients.db")
        if collect_single_face(image_path):
            data_storage.append({'text': name, 'image': image_path})
            message = f"{name} registered successfull"
            if not data_manager.add_patient(name, age, desc):
                message = "Patient with the same name already exists
        else: message = "Face wasn't recognized"
        response = {"message": message}
        return response
    except:
        response = "Serverside error."

app = Flask(__name__)
app.config['QUEUE_FOLDER'] = 'static/queued'
app.config['REGISTRATION_FOLDER'] = 'faces'
app.config['UPLOAD_FOLDER'] = 'static/uploads'

if not os.path.exists(app.config['UPLOAD_FOLDER']):
    os.makedirs(app.config['UPLOAD_FOLDER'])
if not os.path.exists(app.config['QUEUE_FOLDER']):
    os.makedirs(app.config['QUEUE_FOLDER'])
if not os.path.exists(app.config['REGISTRATION_FOLDER']):
    os.makedirs(app.config['REGISTRATION_FOLDER'])

# Dummy data storage
data_storage = []

@app.route('/')
def index():
    return render_template('index.html')
```

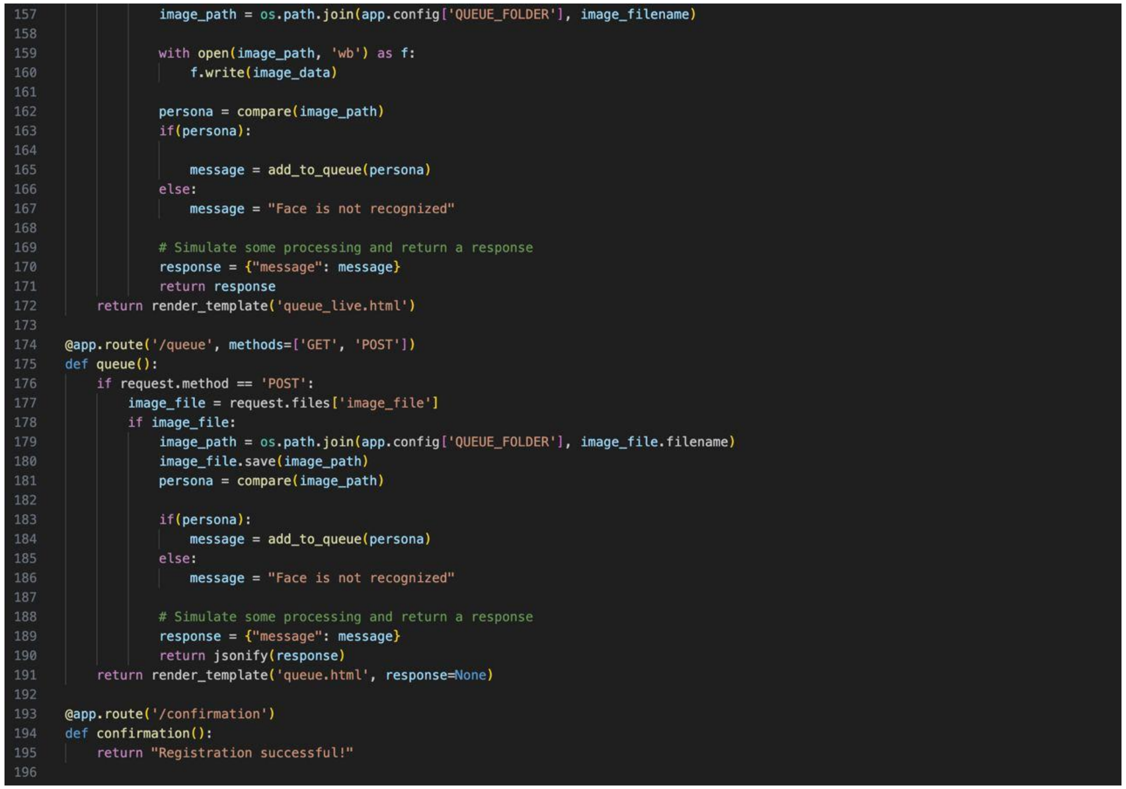
#3



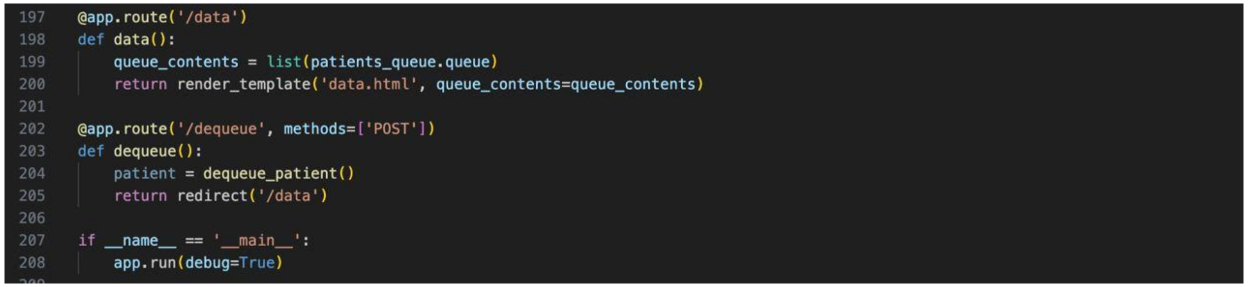
```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name_input = request.form['name_input']
        age_input = request.form['age_input']
        description_input = request.form['description_input']
        image_file = request.files['image_file']
        if image_file:
            image_path = os.path.join(app.config['REGISTRATION_FOLDER'], (name_in
            image_file.save(image_path)
            response = registrar(image_path=image_path, name=name_input, age=age_
            return redirect('/register', Response=response)
    return render_template('register.html')

@app.route('/register_live', methods=['GET', 'POST'])
def register_live():
    if request.method == 'POST':
        name_input = request.form['name_input']
        age_input = request.form['age_input']
        description_input = request.form['description_input']
        image_data = request.form['image_data']
        if image_data:
            # Decode the base64 image data
            image_data = image_data.split(",")[1]
            image_data = base64.b64decode(image_data)
            image_filename = f"{name_input.replace(' ', '_')}.png"
            image_path = os.path.join(app.config['REGISTRATION_FOLDER'], image_fi
            with open(image_path, 'wb') as f:
                f.write(image_data)
            response = registrar(image_path=image_path, name=name_input, age=age_
            return redirect('/register_live', Response=response)
    return render_template('register_live.html')

@app.route('/queue_live', methods=['GET', 'POST'])
def queue_live():
    if request.method == 'POST':
        image_data = request.form['image_data']
        if image_data:
            print("here")
            # Decode the base64 image data
            image_data = image_data.split(",")[1]
            image_data = base64.b64decode(image_data)
            image_filename = f"queued_image_{len(data_storage) + 1}.png"
```

#4

Software Development



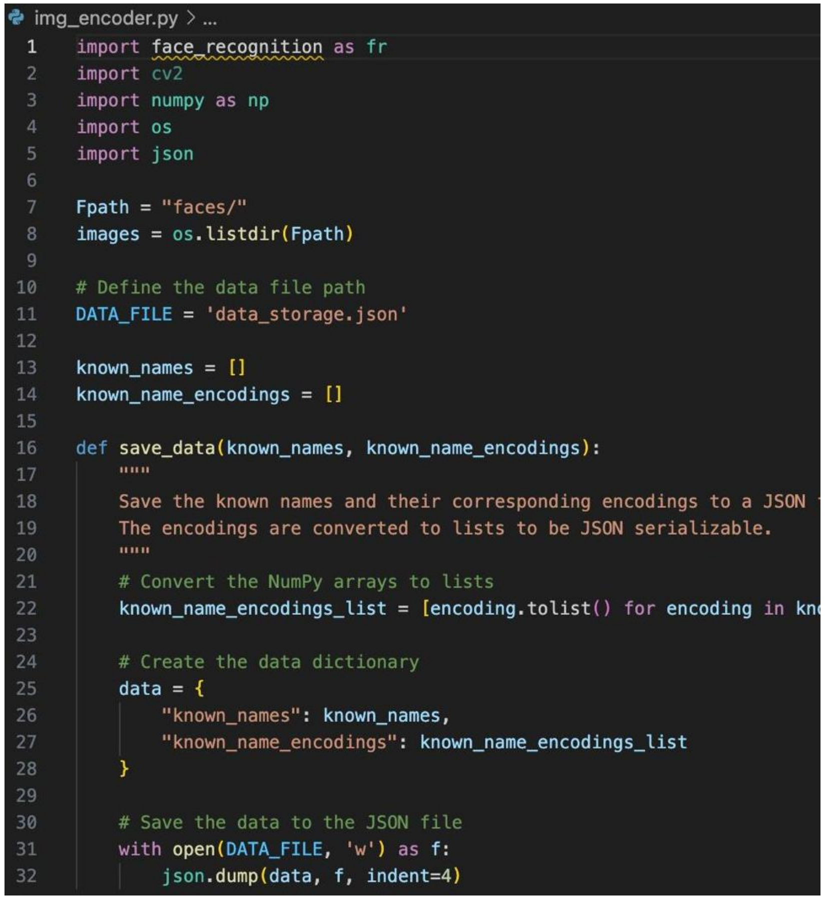
```
            image_path = os.path.join(app.config['QUEUE_FOLDER'], image_filename)

            with open(image_path, 'wb') as f:
                f.write(image_data)

            persona = compare(image_path)
            if(persona):

                message = add_to_queue(persona)
            else:
                message = "Face is not recognized"

            # Simulate some processing and return a response
            response = {"message": message}
            return response
    return render_template('queue_live.html')

@app.route('/queue', methods=['GET', 'POST'])
def queue():
    if request.method == 'POST':
        image_file = request.files['image_file']
        if image_file:
            image_path = os.path.join(app.config['QUEUE_FOLDER'], image_file.filename)
            image_file.save(image_path)
            persona = compare(image_path)

            if(persona):
                message = add_to_queue(persona)
            else:
                message = "Face is not recognized"

            # Simulate some processing and return a response
            response = {"message": message}
            return jsonify(response)
    return render_template('queue.html', response=None)

@app.route('/confirmation')
def confirmation():
    return "Registration successful!"

```



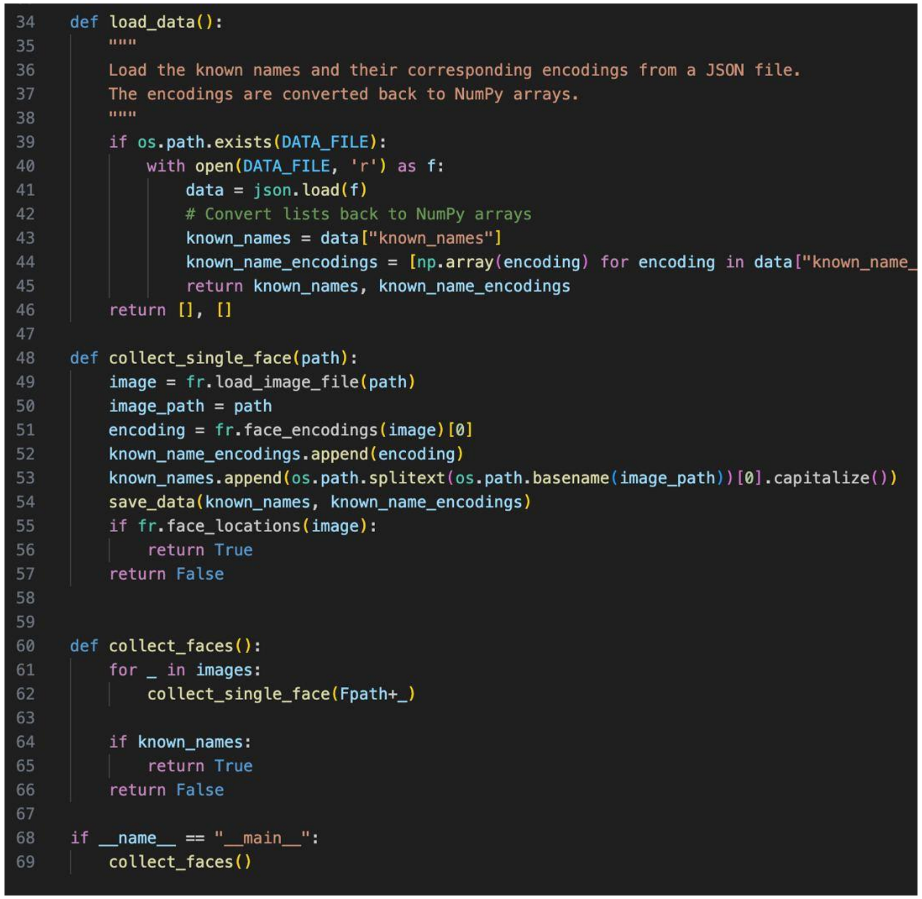
```
@app.route('/data')
def data():
    queue_contents = list(patients_queue.queue)
    return render_template('data.html', queue_contents=queue_contents)

@app.route('/dequeue', methods=['POST'])
def dequeue():
    patient = dequeue_patient()
    return redirect('/data')

if __name__ == '__main__':
    app.run(debug=True)
```

Software Development

인하대학교 INHA UNIVERSITY


img_encoder.py > ...

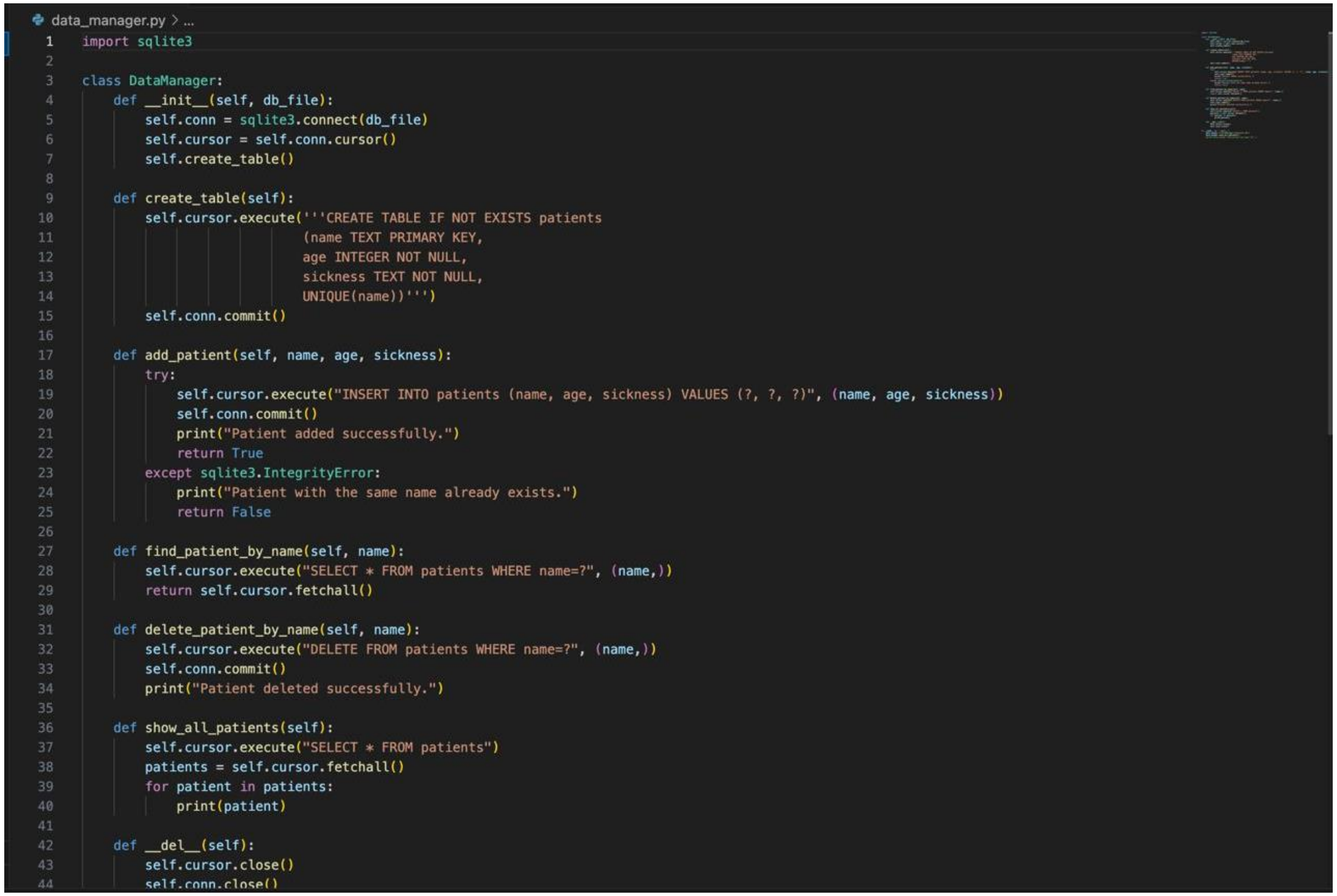
```python
import face_recognition as fr
import cv2
import numpy as np
import os
import json

Fpath = "faces/"
images = os.listdir(Fpath)

# Define the data file path
DATA_FILE = 'data_storage.json'

known_names = []
known_name_encodings = []

def save_data(known_names, known_name_encodings):
    """
    Save the known names and their corresponding encodings to a JSON f
    The encodings are converted to lists to be JSON serializable.
    """
    # Convert the NumPy arrays to lists
    known_name_encodings_list = [encoding.tolist() for encoding in kno

    # Create the data dictionary
    data = {
        "known_names": known_names,
        "known_name_encodings": known_name_encodings_list
    }

    # Save the data to the JSON file
    with open(DATA_FILE, 'w') as f:
        json.dump(data, f, indent=4)
```



```python
def load_data():
    """
    Load the known names and their corresponding encodings from a JSON file.
    The encodings are converted back to NumPy arrays.
    """
    if os.path.exists(DATA_FILE):
        with open(DATA_FILE, 'r') as f:
            data = json.load(f)
            # Convert lists back to NumPy arrays
            known_names = data["known_names"]
            known_name_encodings = [np.array(encoding) for encoding in data["known_name_
            return known_names, known_name_encodings
    return [], []

def collect_single_face(path):
    image = fr.load_image_file(path)
    image_path = path
    encoding = fr.face_encodings(image)[0]
    known_name_encodings.append(encoding)
    known_names.append(os.path.splitext(os.path.basename(image_path))[0].capitalize())
    save_data(known_names, known_name_encodings)
    if fr.face_locations(image):
        return True
    return False


def collect_faces():
    for _ in images:
        collect_single_face(Fpath+_)

    if known_names:
        return True
    return False

if __name__ == "__main__":
    collect_faces()
```

Image encoder

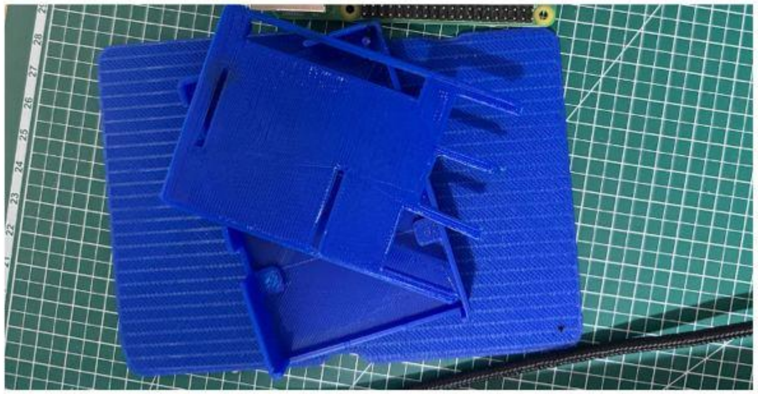
Software Development

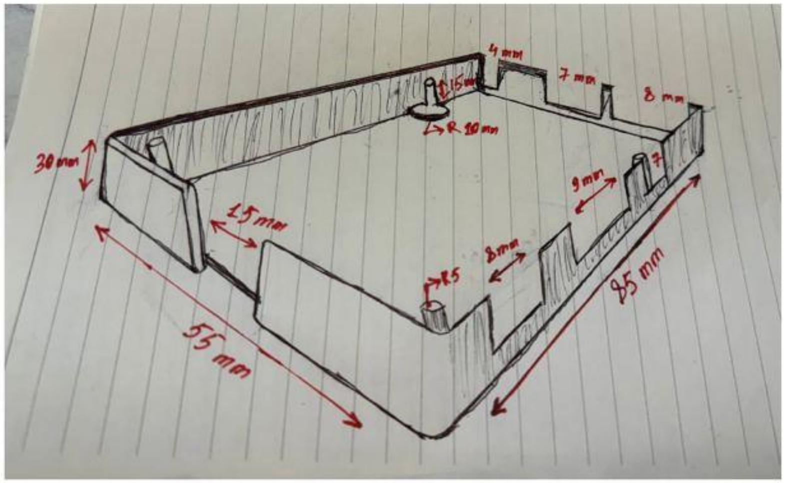


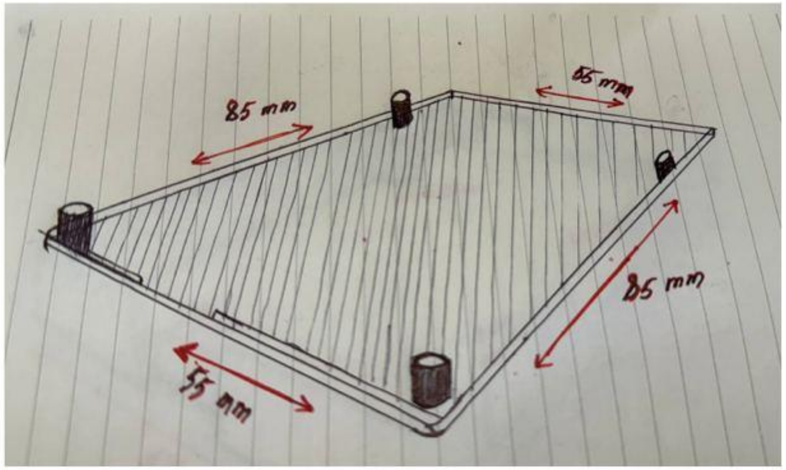
```python
import sqlite3

class DataManager:
    def __init__(self, db_file):
        self.conn = sqlite3.connect(db_file)
        self.cursor = self.conn.cursor()
        self.create_table()

    def create_table(self):
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS patients
                            (name TEXT PRIMARY KEY,
                            age INTEGER NOT NULL,
                            sickness TEXT NOT NULL,
                            UNIQUE(name))''')
        self.conn.commit()

    def add_patient(self, name, age, sickness):
        try:
            self.cursor.execute("INSERT INTO patients (name, age, sickness) VALUES (?, ?, ?)", (name, age, sickness))
            self.conn.commit()
            print("Patient added successfully.")
            return True
        except sqlite3.IntegrityError:
            print("Patient with the same name already exists.")
            return False

    def find_patient_by_name(self, name):
        self.cursor.execute("SELECT * FROM patients WHERE name=?", (name,))
        return self.cursor.fetchall()

    def delete_patient_by_name(self, name):
        self.cursor.execute("DELETE FROM patients WHERE name=?", (name,))
        self.conn.commit()
        print("Patient deleted successfully.")

    def show_all_patients(self):
        self.cursor.execute("SELECT * FROM patients")
        patients = self.cursor.fetchall()
        for patient in patients:
            print(patient)

    def __del__(self):
        self.cursor.close()
        self.conn.close()
```

Data manager

3D MODELING

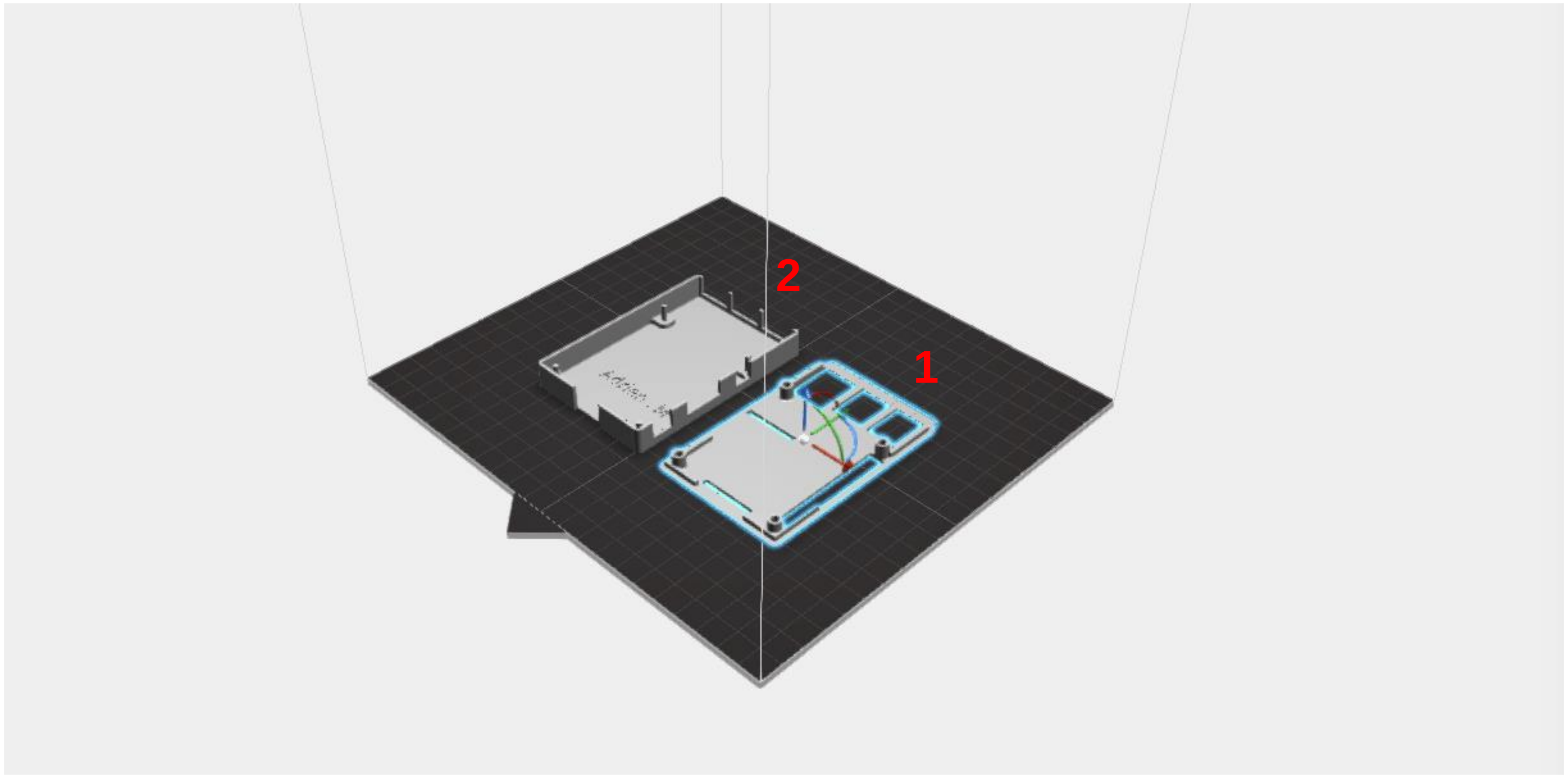



4 mm
7 mm
8 mm
15 mm
R 10 mm
30 mm
9 mm
7
15 mm
8 mm
R5
55 mm
85 mm

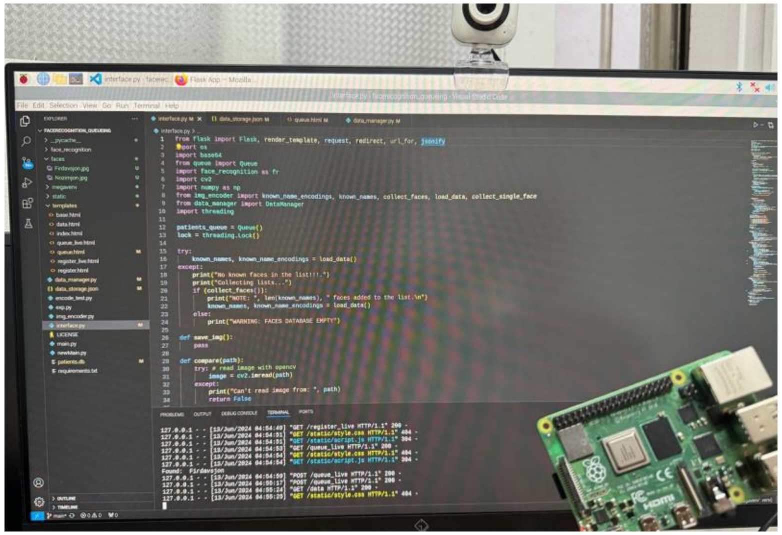

85 mm
55 mm
85 mm
55 mm

3D MODELING

• 1	X	60mm	Y	7mm	Z	94mm
• 2	X	60mm	Y	11.5mm	Z	89mm

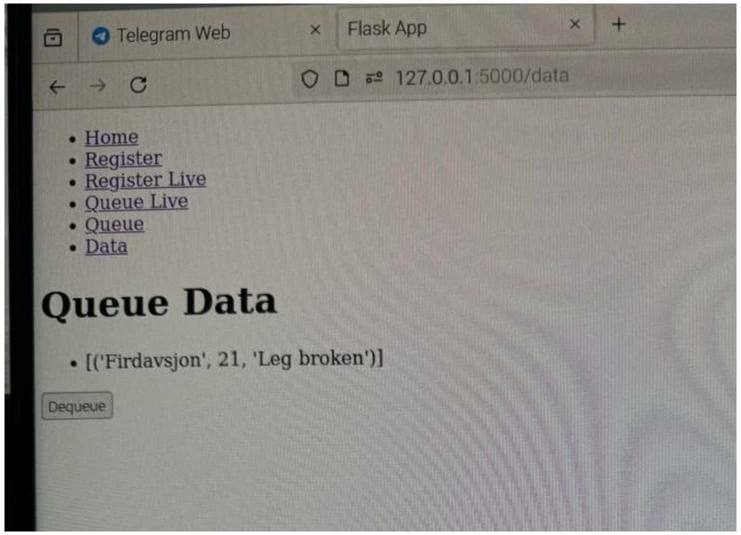


User Interface Design



```python
from flask import Flask, render_template, request, redirect, url_for, jsonify
import os
import base64
from queue import Queue
import face_recognition as fr
import cv2
import numpy as np
from img_encoder import known_name_encodings, known_names, collect_faces, load_data, collect_single_face
from data_manager import DataManager
import threading

patients_queue = Queue()
lock = threading.Lock()

try:
    known_names, known_name_encodings = load_data()
except:
    print("No known faces in the list!!!.")
    print("Collecting lists...")
    if (collect_faces()):
        print("NOTE: ", len(known_names), " faces added to the list.\n")
        known_names, known_name_encodings = load_data()
    else:
        print("WARNING: FACES DATABASE EMPTY")

def save_img():
    pass

def compare(path):
    try: # read image with opencv
        image = cv2.imread(path)
    except:
        print("Can't read image from: ", path)
        return False
```

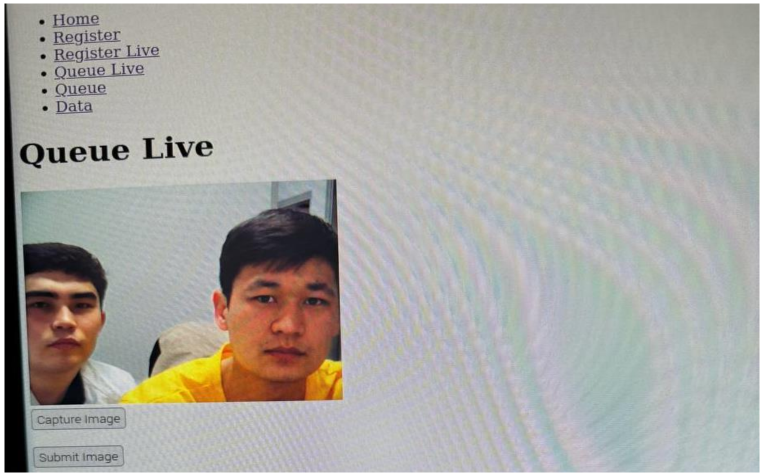
```
127.0.0.1 - - [13/Jun/2024 04:54:49] "GET /register_live HTTP/1.1" 200 -
127.0.0.1 - - [13/Jun/2024 04:54:51] "GET /static/style.css HTTP/1.1" 404 -
127.0.0.1 - - [13/Jun/2024 04:54:51] "GET /static/script.js HTTP/1.1" 304 -
127.0.0.1 - - [13/Jun/2024 04:54:53] "GET /queue_live HTTP/1.1" 200 -
127.0.0.1 - - [13/Jun/2024 04:54:54] "GET /static/style.css HTTP/1.1" 404 -
127.0.0.1 - - [13/Jun/2024 04:54:54] "GET /static/script.js HTTP/1.1" 304 -
Found:  Firdavsjon
127.0.0.1 - - [13/Jun/2024 04:54:59] "POST /queue_live HTTP/1.1" 200 -
127.0.0.1 - - [13/Jun/2024 04:55:17] "POST /queue_live HTTP/1.1" 200 -
127.0.0.1 - - [13/Jun/2024 04:55:24] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [13/Jun/2024 04:55:29] "GET /static/style.css HTTP/1.1" 404 -
```



- Home
- Register
- Register Live
- Queue Live
- Queue
- Data

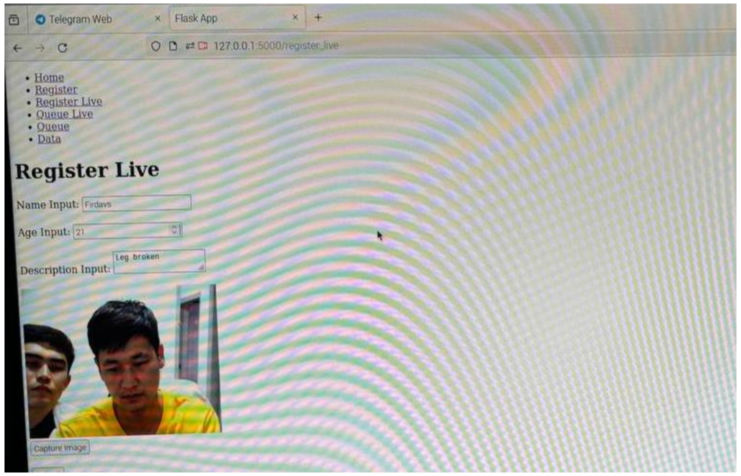
Queue Data

- [('Firdavsjon', 21, 'Leg broken')]



- Home
- Register
- Register Live
- Queue Live
- Queue
- Data

Queue Live



- Home
- Register
- Register Live
- Queue Live
- Queue
- Data

Register Live

Name Input: Firdavs

Age Input: 21

Description Input: Leg broken

User Interface Design

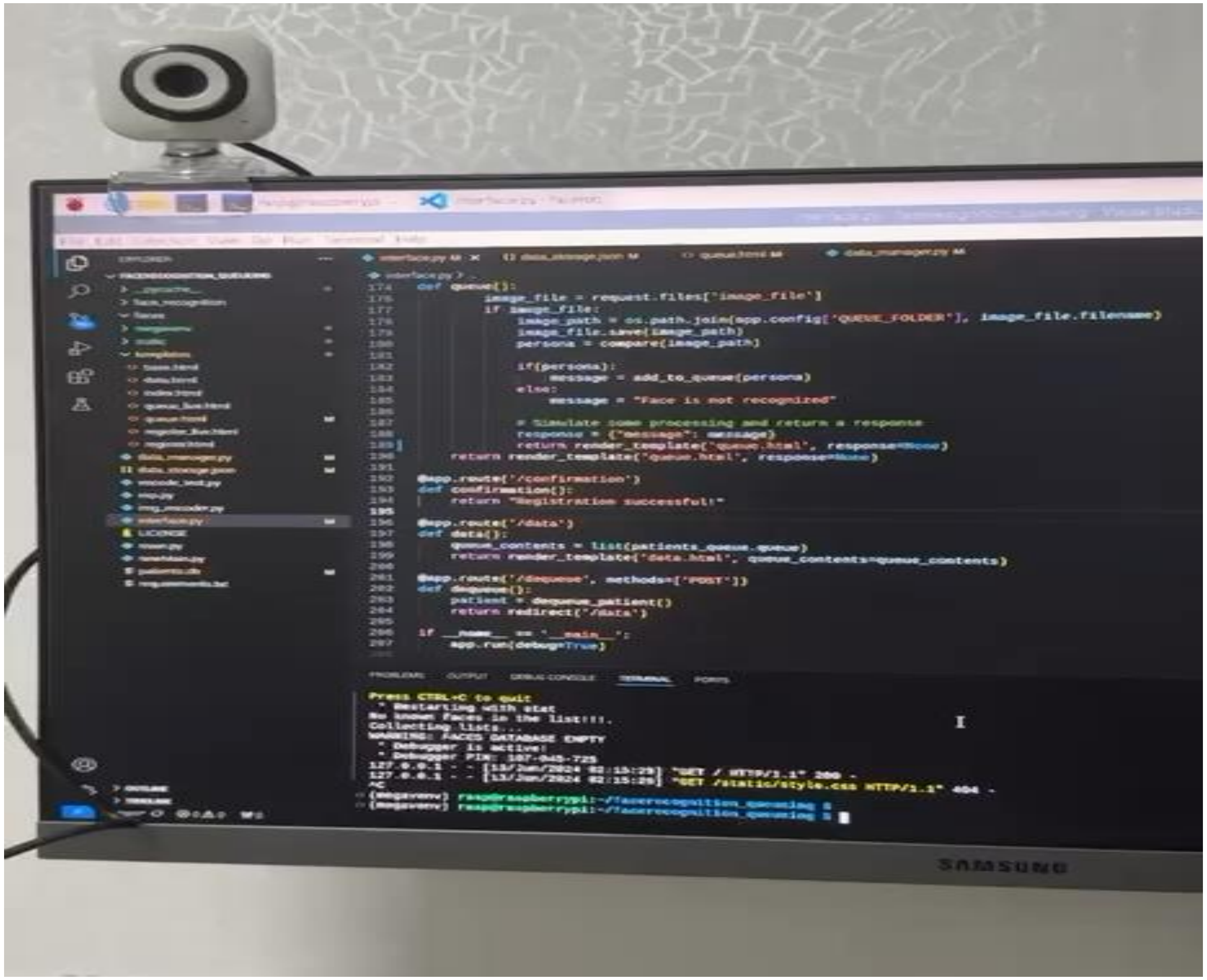
UI/UX Design

•**Layout and Design Principles**:
•Clean, intuitive, and user-friendly design.

•**Ease of Use**:
•Accessible and simple for all patient demographics.

•**Screenshots**
• Visual examples of the main interface screens.

Video Demo



```
def queue():
        image_file = request.files['image_file']
        if image_file:
            image_path = os.path.join(app.config['QUEUE_FOLDER'], image_file.filename)
            image_file.save(image_path)
            persona = compare(image_path)

            if(persona):
                message = add_to_queue(persona)
            else:
                message = "Face is not recognized"

            # Simulate some processing and return a response
            response = {"message": message}
            return render_template('queue.html', response=None)
    return render_template('queue.html', response=None)

@app.route('/confirmation')
def confirmation():
    return "Registration successful!"

@app.route('/data')
def data():
    queue_contents = list(patients_queue.queue)
    return render_template('data.html', queue_contents=queue_contents)

@app.route('/dequeue', methods=['POST'])
def dequeue():
    patient = dequeue_patient()
    return redirect('/data')

if __name__ == '__main__':
    app.run(debug=True)
```

```
Press CTRL+C to quit
 * Restarting with stat
No known faces in the list!!!.
Collecting lists...
WARNING: FACES DATABASE EMPTY
 * Debugger is active!
 * Debugger PIN: 107-045-725
127.0.0.1 - - [13/Jun/2024 02:15:29] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [13/Jun/2024 02:15:29] "GET /static/style.css HTTP/1.1" 404 -
^C
(megavenv) rasp@raspberrypi:~/facerecognition_queueing $
(megavenv) rasp@raspberrypi:~/facerecognition_queueing $
```

References

https://keras.io/api/datasets/

https://streamlit.io/

https://chatgpt.com

https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html

https://docs.ros.org/en/foxy/index.html

https://inha343.autodesk360.com/g/projects/20240506763917551/data/dXJuOmFkc2sud2lwcHJvZDpmcy5mb2xkZXI6Y28ud1ZxWm5VaXdRTE9zM2ZMRTJ1N0FHUQ

https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

https://github.com/freedom99/SOD---An-Embedded-OpenCV-Alternative?tab=readme-ov-file#programming-interfaces

https://learnopencv.com/embedded-computer-vision-which-device-should-you-choose/

https://www.tensorflow.org/api_docs/python/tf/

https://cloudprint.makerbot.com/workspace/cbc818de-cc68-455b-8859-ed60e431f647/jobs

https://www.3dforprint.com (3D Modeling)

Github link: https://github.com/baxtiyorov3407/Face_Recognation.git

Q&A