Facial Recognition Access Control System
Technical Manual
by

Hoda Ahmed;
Student ID: C00214991

April 19, 2020

1 Introduction

The technical manual document contains all code for the FRACS application. The FRACS application can be run on any Linux/Unix-based system, if the suitable libraries are installed.

For the desktop application, all of the files are Python files. For ease of readability and structure, the files are broken down to represent database operations and external libraries and other APIs operations.

2 Desktop Application (Raspberry Pi)

2.1 Initialization

This section of the document includes code the is used to set up the database tables and create the collection in AWS. Firstly, the constructor is called, which calls the main function, invoking the following methods.

```python
def main():
    createDB()
    createTables()
    collection_id = "FRACSUsers"
    createRekCollection(collection_id)  # create collection called FRACSUsers
    super_admin_face_id = indexSuperAdminFace(collection_id) # index super admin face & store it in FRACSUsers collection
    createAdminDBEntry(super_admin_face_id)

if __name__ == '__main__':
    main()
```

Listing 1: fracs-init.py - main()

```python
#firstly, local database needs to be set up:
def createDB():
    db_connection = mysql.connector.connect(host ="localhost", user = "hoda", password = "XXXX")
    db_cursor = db_connection.cursor()
    db_cursor.execute("CREATE DATABASE IF NOT EXISTS FracsDB;")
    db_connection.close()
```

Listing 2: fracs-init.py - Local DB Initialization

```python
#then, tables initialized
def createTables():
    db_connection = mysql.connector.connect(host = "localhost", user = "hoda", password = "XXXX", database = "FracsDB")
    db_cursor = db_connection.cursor()
    db_cursor.execute("CREATE TABLE `fracs_users` ( "
                      "`user_id` int NOT NULL AUTO_INCREMENT,"+
                      "`firstname` varchar(255) NOT NULL,"+
                      "`lastname` varchar(255) NOT NULL,"+
                      "`login_username` varchar(255) NOT NULL,"+
                      "`login_password` varchar(255) NOT NULL,"+
                      "`is_admin` tinyint(1) NOT NULL DEFAULT '0',"+
                      "`phone_number` varchar(15) DEFAULT NULL,"+
                      "`face_id` varchar(255) NOT NULL,"+
                      "`admin_registered` varchar(255) DEFAULT NULL,"+
                      "PRIMARY KEY (`user_id`))")

    db_cursor.execute("CREATE TABLE `fracs_logs` (" +
                      " `log_id` int NOT NULL AUTO_INCREMENT,"+
                      " `username_entered` varchar(255) NOT NULL,"+
                      " `face_id` varchar(255) NOT NULL,"+
                      " `attempted_user_id` int DEFAULT NULL,"+
                      " `login_datetime` datetime NOT NULL,"+
                      " `is_successful` varchar(10) NOT NULL,"+
                      " PRIMARY KEY (`log_id`))")

    db_connection.commit()
    db_connection.close()

```

Listing 3: fracs-init.py - Local DB Tables Initialization

Following the initialization and creation of the local database, the Rekognition API will be called to create a collection for the users' faces, and then a "super admin" will be registered on the collection using the index_face function.

```
def createRekCollection(collection_id):
    #Create a collection
    print('Creating collection:' + collection_id)
    response = client.create_collection(CollectionId = collection_id)

    print('Status code: ' + str(response['StatusCode']))
    if (str(response['StatusCode']) == "200"):
        print("Collection created successfully!")

    # as soon as a collection is created, an admin must register their face to be able to login & register other users
    # therefore, with the creation of the collection, the admin must also store their face in that collection i.e. call indexSuperAdminFace method
```

Listing 4: fracs-init.py - Rekognition Collection Initialization

```
def indexSuperAdminFace(collection_id):
    photo = "admin.jpg"

    with open(photo, 'rb') as image:
        response = client.index_faces(
            CollectionId = collection_id,
            Image = {
                'Bytes': image.read()
            },
            ExternalImageId = photo,
            QualityFilter = "AUTO",
            DetectionAttributes = ['ALL']
        )
    faceid = response['FaceRecords'][0]['Face']['FaceId']
    print("Admin indexed successfully!")
    return faceid
```

Listing 5: fracs-init.py - "Super Admin" creation

Finally, the admin is stored in the local database to be able to interact with the desktop app.

```
def createAdminDBEntry(face_id):
    db_connection = mysql.connector.connect(host = "localhost", user = "hoda", password = "XXXX", database = "FracsDB")
    db_cursor = db_connection.cursor()

    p = "INSERT INTO fracs_users (firstname, lastname, login_username, login_password, is_admin, face_id) VALUES ('hoda','ahmed','hdahedo','" + pbkdf2_sha256.hash("efta7") + "', 1,'" + face_id + "');"
    db_cursor.execute(p)
    db_connection.commit()
    db_connection.close()
    print("Admin user created successfully! You may now login to the FRACS system.")
```

Listing 6: fracs-init.py - Register Super Admin locally

2.2 Database Operations

The database operation python file is specified for only operations that require interaction with the local database. This can include login validation, registering a user or for logging. The following code snippet is placed at the top of the dbOperations.py file for the imports and global variables declaration.

```
import mysql.connector
from passlib.hash import pbkdf2_sha256

db_connection = mysql.connector.connect(host="localhost", user="hoda", password="
    OPENsesameFRACS202o", database="FracsDB")
db_cursor = db_connection.cursor()
```

Listing 7: dbOperations.py

The *getHashedPassword()* function takes in two parameters, a login username and a face id. A hashed password for the user trying to login is returned to check if the password they entered is correct.

```
def getHashedPassword(username, face_id):
    query = "SELECT login_password from fracs_users where login_username = '" +
    username + "'AND face_id = '" + face_id + "';"
    db_cursor.execute(query)
    password = db_cursor.fetchone()
    if password is not None:
        return password[0]
    return None
```

Listing 8: dbOperations.py - getHashedPassword()

The *getAttemptedUserId()* function returns the ID of the user whose login username has been entered in a login attempt. This is used for logging, and can be useful in the event of user trying to login as another user using their credentials. However, that login attempt will fail, as the credentials have to match with the face of the user attempting to login.

```
def getAttemptedUserId(username):
    query = "SELECT user_id from fracs_users where login_username = '" + username
    + "';"
    db_cursor.execute(query)
    userid = db_cursor.fetchone()
    if userid is not None:
        return userid[0]
    return -1
```

Listing 9: dbOperations.py - getAttemptedUserId()

The *logAccessEvent()* function logs every access attempt/every login attempt to FRACS. These logs can then be viewed by admins at a later stage on the web application.

```
def logAccessEvent(logInfo):
    query = "INSERT INTO fracs_logs (username_entered, face_id, attempted_user_id
    , login_datetime, is_successful) VALUES ('" + logInfo["username_entered"] + "
    ','" + logInfo["face_id"] + "'," +  str(logInfo["attempted_user_id"]) + ",'"
    +logInfo["login_datetime"] + "','" + logInfo["is_successful"] + "');"
    print(query)
    db_cursor.execute(query)
    db_connection.commit()
    db_connection.close()
```

Listing 10: dbOperations.py - logAccessEvent()

The *getAdminHashedPassword()* is similar to the above getHashedPassword() function, but is defined specifically for admins as the query is different.

```
def getAdminHashedPassword(username, face_id):
    query = "SELECT login_password from fracs_users where login_username = '" +
   username + "'AND face_id = '" + face_id + "' and is_admin = 1;"
    db_cursor.execute(query)
    password = db_cursor.fetchone()
    if password is not None:
        return password[0]
    return None
```

Listing 11: dbOperations.py - getAdminHashedPassword()

The *getAdminPhoneNumber()* function is used for SMS verification (2FA) during the registration process.

```
def getAdminPhoneNumber(username):
    query = "SELECT phone_number from fracs_users where login_username = '" +
   username + "' and is_admin = 1;"
    db_cursor.execute(query)
    phone_num = db_cursor.fetchone()
    if phone_num is not None:
        return phone_num[0]
    return None
```

Listing 12: dbOperations.py - getAdminPhoneNumber()

The *checkUsername()* function checks the availability of a username, and this would be invoked in the registration process when a user wished to use a certain username. If the username is not available for use, the user is notified of this.

```
def checkUsername(username):
    query = "SELECT user_id from fracs_users where login_username = '" + username
    + "';"
    db_cursor.execute(query)
    usernameTaken = db_cursor.fetchone()
    if usernameTaken is not None:
        return False
    return True
```

Listing 13: dbOperations.py - checkUsername()

The *addNewUer()* function adds a new user entry in the database, allowing them to login in the future.

```
def addNewUser(newUserInfo):
    query = "INSERT INTO fracs_users (firstname, lastname, login_username,
   login_password, is_admin, face_id, admin_registered) VALUES ('" + newUserInfo
   ["firstname"] + "','" + newUserInfo["lastname"] + "','" +  newUserInfo["
   login_username"] + "','" +newUserInfo["login_password"] +"'," +  str(
   newUserInfo["is_admin"]) + ",'" + newUserInfo["face_id"] + "', '"+
   newUserInfo["admin_registered"] +"');"
    print(query)
    db_cursor.execute(query)
    db_connection.commit()
    db_connection.close()
    print("User created successfully! You may now login to FRACS")
```

Listing 14: dbOperations.py - checkUsername()

2.3 AWS Rekognition Operations

The rekognition operation python file is specified for only operations that are carried out using AWS Rekognition API. This can include detecting faces in a picture, verifying whether a detected face is registered with the system or not and adding new faces to a specified collection.

In order for this code to run and for the "boto3" module to be imported by Python, the AWS credentials must be stored in the home directory of the computer.

```python
import boto3
from botocore.exceptions import ClientError

collection_id = "FRACSUsers"
client=boto3.client('rekognition')
```

Listing 15: rekognitionOperations.py

The *confirmUserFace()* function is used when a user/admin wishes to login.

```python
def confirmUserFace():
    fileName='user.jpg'
    threshold = 70

    '''
    return codes:   0 = no face detected
    1 = face detected but unknown (not registered with fracs)
    -1 = more than one face detected
    '''

    imgFaceNum = detectFaces(fileName)
    if (imgFaceNum == 0):
        return 0
    elif (imgFaceNum > 1):
        return -1
    else:
        #opening file & getting face
        with open(fileName, 'rb') as image:
            response=client.search_faces_by_image(
                CollectionId=collection_id,
                Image=
                    {
                    'Bytes': image.read()
                    },
                FaceMatchThreshold=threshold
            )
        # exception handling done here for "no face exists" scenario, but not for "more than one face exists" scenario
        # there shouldn't be more than one user per face as per exception handling during registration

        faceMatches=response['FaceMatches']
        if faceMatches != []: #if list of faceMatches is not empty
            return (faceMatches[0]['Face']['FaceId'])
        return 1
```

Listing 16: rekognitionOperations.py - confirmUserFace()

The *detectFaces()* function examines a given image and detects the faces in that image.

```
def detectFaces(file):
    fileName=file

    #opening file & getting face
    with open(fileName, 'rb') as image:
        response=client.detect_faces(
            Image=
            {
            'Bytes': image.read()
            }
        )
    return len(response['FaceDetails'])

    # response contains a dictionary of faces detected & their details..
    # if we get the length of that dictionary,
    # we can determine the number of faces in the picture
```

Listing 17: rekognitionOperations.py - detectFaces()

The *addNewFace()* function adds a new face to the Rekognition collection. This function is invoked when a user is registering.

```
def addNewFace():
    photo = "user.jpg"

    with open(photo, 'rb') as image:
        response=client.index_faces(
            CollectionId=collection_id,
            Image=
            {
                'Bytes': image.read()
            },
            ExternalImageId = photo,
            QualityFilter = "AUTO"
        )

    faceid = response['FaceRecords'][0]['Face']['FaceId']
    return faceid
```

Listing 18: rekognitionOperations.py - addNewFace()

```
def main():
    print("in rekOp")

if __name__ == '__main__':
    main()
```

Listing 19: rekognitionOperations.py - main()

2.4 Twilio SMS API Operations

After logging in to register a user, the admin must use their phone number to verify themselves. To do this, the external system, Twilio, is used.

```
from twilio.rest import Client
from random import randint

smscode = 0
```

Listing 20: twilioOperations.py

The *sendSMSCode()* is called when the admin authenticates themself and needs to complete two-factor authentication. Twilio sends a 6-digit code to the admin's phone number, after which they will need to enter that code into the system to verify using 2FA. It is also called when a new admin wishes to register their phone number with the system.

```
def sendSMSCode(recipient):
    global smscode

    range_start = 10**(6-1)
    range_end = (10**6)-1
    smscode = randint(range_start, range_end)

    #Your Account SID from twilio.com/console
    account_sid = "XXXXXX"

    #Your Auth Token from twilio.com/console
    auth_token = "XXXXXX"
    client = Client(account_sid, auth_token)

    message = client.messages.create(
        to= recipient,
        from_="XXXXX", # TWILIO'S REGISTERED PHONE NUMBER
        body=smscode
    )
    return True
```

Listing 21: twilioOperations.py - sendSMSCode()

The *AddNewNumber()* function is invoked when a new admin wishes to store their phone number information into the system to be able to register users at a later stage.

```

#will only work if twilio is upgraded/premium
def AddNewNumber(person, number):
    #Your Account SID from twilio.com/console
    account_sid = "XXXXXXX"

    #Your Auth Token from twilio.com/console
    auth_token = "XXXXXXX"
    client = Client(account_sid, auth_token)

    validation_request = client.validation_requests.create(
        friendly_name=person,
        phone_number=number
    )

    global smscode
    smscode = validation_request.validation_code
    return smscode
```

Listing 22: twilioOperations.py - AddNewNumber()

The *VerifyTwilioSMS()* is used when the system wishes to verify 2FA, after an admin/user enters the code they received on their mobile phones into the system to complete 2FA.

```
def VerifyTwilioSMS(code):
    global smscode
    enteredCode = int(code)

    if (enteredCode == smscode):
        return True
    else:
        return False

```

Listing 23: twilioOperations.py - VerifyTwilioSMS()

2.5 Lock Operation

The lock operation code is called when a user successfully logs in and is authorized to access the room. It is one file that consists of one block of code that carries out everything related to the solenoid lock.

```
import RPi.GPIO as GPIO
import time

'''
The GPIO.BOARD option specifies that you are referring to the pins by the number
    of the pin the the plug - i.e
the numbers printed on the board (e.g. P1) and in the middle of the diagrams
    below.

The GPIO.BCM option means that you are referring to the pins by the "Broadcom SOC
     channel" number
'''

def unlock():
    try:
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(23, GPIO.OUT)  #Connect LOCK to GPIO 23
        GPIO.output(23, True)
        time.sleep(6)
        GPIO.output(23, False)
        GPIO.cleanup()
    except:
        GPIO.cleanup()

def main():
    unlock()

if __name__ == '__main__':
    main()
```

Listing 24: lockOperation.py

2.6 Camera Operation

There is only one operation carried out by the camera - taking a picture. The *coverCounter()* function of the below code creates 3 overlays on the camera's preview and pauses the operation for 1 second, hence doing a countdown of 3 and then a picture is taken.

```
import picamera
from PIL import Image
from time import sleep

def coverCounter(camera):
    # load the image
    img1 = Image.open('images/cd3.png')
    img2 = Image.open('images/cd2.png')
    img3 = Image.open('images/cd1.png')

    # create the pad
    pad = Image.new('RGB', (
    ((img1.size[0] + 31) // 32) * 32,
    ((img1.size[1] + 15) // 16) * 16,
    ))

    # paste the overlay - 3
    # display num 3 of countdown
    pad.paste(img1, (0,0))
    o = camera.add_overlay(pad.tobytes(), size=img1.size)
    o.alpha = 128
    o.layer = 3
    sleep(1)

    # Remove previous overlay (num 3)
    camera.remove_overlay(o)

    # paste the overlay - 2
    # display num 2 of countdown
    pad.paste(img2, (0,0))
    o = camera.add_overlay(pad.tobytes(), size=img2.size)
    o.alpha = 128
    o.layer = 3

    sleep(1)

    #remove previous overlay - 2
    camera.remove_overlay(o)

    # paste the overlay - 1
    # display num 1 of countdown
    pad.paste(img3, (0,0))
    o = camera.add_overlay(pad.tobytes(), size=img3.size)
    o.alpha = 128
    o.layer = 3
    sleep(1)
    camera.remove_overlay(o)

def main():
    camera = picamera.PiCamera()
    camera.resolution = (1280,1024)
    camera.framerate = 24
    camera.start_preview()
    overlayCounter(camera)
    camera.capture('user.jpg')
    camera.stop_preview()
    camera.close()
```

```python

if __name__ == '__main__':
    main()
```

Listing 25: camCaptureOperation.py

2.7 User Interface

2.7.1 Main UI

The UI of the FRACS desktop application was designed using Qt Designer, which is a piece of software that makes designing GUI designs for PyQt and Qt for C++ easier. It offers a wide variety of options for creating a design for the Qt application.
The following code is the UI code created by Qt Designer, used for the FRACS desktop application.

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'fracsUI.ui'
#
# Created by: PyQt5 UI code generator 5.14.0


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(801, 480)
        font = QtGui.QFont()
        font.setPointSize(14)
        MainWindow.setFont(font)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.stackedWidget = QtWidgets.QStackedWidget(self.centralwidget)
        self.stackedWidget.setGeometry(QtCore.QRect(0, 0, 800, 480))
        self.stackedWidget.setStyleSheet("background-image: url(./images/cyber5.jpeg)")
        self.stackedWidget.setObjectName("stackedWidget")
        self.SPLASH = QtWidgets.QWidget()
        self.SPLASH.setStyleSheet("background-image: url(./images/newcyber.gif);")
        self.SPLASH.setObjectName("SPLASH")
        self.label = QtWidgets.QLabel(self.SPLASH)
        self.label.setGeometry(QtCore.QRect(20, 20, 341, 151))
        font = QtGui.QFont()
        font.setPointSize(50)
        self.label.setFont(font)
        self.label.setStyleSheet("color: rgb(255, 255, 255);\n"
            "background-color: rgba(0,0,0, 0);\n"
            "background-image: url(./images/transparent.png)")
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setWordWrap(True)
        self.label.setObjectName("label")
        self.stackedWidget.addWidget(self.SPLASH)
        self.MENU = QtWidgets.QWidget()
        self.MENU.setObjectName("MENU")
        self.login = QtWidgets.QPushButton(self.MENU)
        self.login.setGeometry(QtCore.QRect(110, 60, 571, 131))
        font = QtGui.QFont()
        font.setFamily("Ubuntu")
        font.setPointSize(40)
        font.setBold(False)
        font.setItalic(False)
        font.setWeight(50)
        self.login.setFont(font)
        self.login.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.login.setStyleSheet("background-color: rgb(30, 129, 196);\n"
```

```
            "background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:0, stop:0 rgba(69, 81, 136, 240), stop:1 rgba(255, 255, 255, 255));\n"
            "background-color: rgb(238, 238, 236);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.login.setObjectName("login")
        self.registration = QtWidgets.QPushButton(self.MENU)
        self.registration.setGeometry(QtCore.QRect(110, 270, 571, 141))
        font = QtGui.QFont()
        font.setPointSize(40)
        self.registration.setFont(font)
        self.registration.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.registration.setStyleSheet("background-color:rgb(30, 129, 196);\n"
            "background-color: rgb(238, 238, 236);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;")
        self.registration.setObjectName("registration")
        self.stackedWidget.addWidget(self.MENU)
        self.LOGIN = QtWidgets.QWidget()
        self.LOGIN.setObjectName("LOGIN")
        self.unameLabel = QtWidgets.QLabel(self.LOGIN)
        self.unameLabel.setGeometry(QtCore.QRect(100, 150, 101, 21))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.unameLabel.setFont(font)
        self.unameLabel.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.unameLabel.setObjectName("unameLabel")
        self.passLabel = QtWidgets.QLabel(self.LOGIN)
        self.passLabel.setGeometry(QtCore.QRect(100, 270, 101, 21))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.passLabel.setFont(font)
        self.passLabel.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.passLabel.setObjectName("passLabel")
        self.usernameInput = QtWidgets.QLineEdit(self.LOGIN)
        self.usernameInput.setGeometry(QtCore.QRect(100, 180, 591, 51))
        self.usernameInput.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.usernameInput.setAlignment(QtCore.Qt.AlignCenter)
        self.usernameInput.setObjectName("usernameInput")
        self.passwordInput = QtWidgets.QLineEdit(self.LOGIN)
        self.passwordInput.setGeometry(QtCore.QRect(100, 300, 591, 51))
        self.passwordInput.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;")
        self.passwordInput.setEchoMode(QtWidgets.QLineEdit.Password)
        self.passwordInput.setAlignment(QtCore.Qt.AlignCenter)
        self.passwordInput.setObjectName("passwordInput")
        self.goback = QtWidgets.QPushButton(self.LOGIN)
        self.goback.setGeometry(QtCore.QRect(0, 0, 111, 21))
        self.goback.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.goback.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);")
        self.goback.setObjectName("goback")
```

```
        self.loginTitle = QtWidgets.QLabel(self.LOGIN)
        self.loginTitle.setGeometry(QtCore.QRect(200, 50, 381, 51))
        font = QtGui.QFont()
        font.setPointSize(30)
        self.loginTitle.setFont(font)
        self.loginTitle.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.loginTitle.setObjectName("loginTitle")
        self.loginBtn = QtWidgets.QPushButton(self.LOGIN)
        self.loginBtn.setGeometry(QtCore.QRect(550, 400, 141, 41))
        self.loginBtn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.loginBtn.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;")
        self.loginBtn.setObjectName("loginBtn")
        self.stackedWidget.addWidget(self.LOGIN)
        self.FACE_CAPTURE = QtWidgets.QWidget()
        self.FACE_CAPTURE.setObjectName("FACE_CAPTURE")
        self.camButton = QtWidgets.QPushButton(self.FACE_CAPTURE)
        self.camButton.setGeometry(QtCore.QRect(80, 220, 601, 31))
        font = QtGui.QFont()
        font.setPointSize(20)
        self.camButton.setFont(font)
        self.camButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.camButton.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "border-radius:6px;\n"
            "color: rgb(255, 255, 255);")
        self.camButton.setObjectName("camButton")
        self.goback_2 = QtWidgets.QPushButton(self.FACE_CAPTURE)
        self.goback_2.setGeometry(QtCore.QRect(0, 0, 111, 21))
        self.goback_2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.goback_2.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);")
        self.goback_2.setObjectName("goback_2")
        self.stackedWidget.addWidget(self.FACE_CAPTURE)
        self.LOGIN_SUCCESS = QtWidgets.QWidget()
        self.LOGIN_SUCCESS.setObjectName("LOGIN_SUCCESS")
        self.successTitle = QtWidgets.QLabel(self.LOGIN_SUCCESS)
        self.successTitle.setGeometry(QtCore.QRect(170, 210, 441, 31))
        font = QtGui.QFont()
        font.setPointSize(20)
        self.successTitle.setFont(font)
        self.successTitle.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.successTitle.setObjectName("successTitle")
        self.stackedWidget.addWidget(self.LOGIN_SUCCESS)
        self.LOGIN_FAIL = QtWidgets.QWidget()
        self.LOGIN_FAIL.setObjectName("LOGIN_FAIL")
        self.failTitle = QtWidgets.QLabel(self.LOGIN_FAIL)
        self.failTitle.setGeometry(QtCore.QRect(190, 220, 391, 31))
        font = QtGui.QFont()
        font.setPointSize(20)
        self.failTitle.setFont(font)
        self.failTitle.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.failTitle.setObjectName("failTitle")
        self.stackedWidget.addWidget(self.LOGIN_FAIL)
        self.LOGIN_FAIL_ZERO = QtWidgets.QWidget()
        self.LOGIN_FAIL_ZERO.setObjectName("LOGIN_FAIL_ZERO")
```

```
        self.failTitleZero = QtWidgets.QLabel(self.LOGIN_FAIL_ZERO)
        self.failTitleZero.setGeometry(QtCore.QRect(160, 200, 421, 101))
        font = QtGui.QFont()
        font.setPointSize(20)
        self.failTitleZero.setFont(font)
        self.failTitleZero.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.failTitleZero.setAlignment(QtCore.Qt.AlignCenter)
        self.failTitleZero.setWordWrap(True)
        self.failTitleZero.setObjectName("failTitleZero")
        self.stackedWidget.addWidget(self.LOGIN_FAIL_ZERO)
        self.LOGIN_FAIL_UNAUTH = QtWidgets.QWidget()
        self.LOGIN_FAIL_UNAUTH.setObjectName("LOGIN_FAIL_UNAUTH")
        self.failTitleUnauth = QtWidgets.QLabel(self.LOGIN_FAIL_UNAUTH)
        self.failTitleUnauth.setGeometry(QtCore.QRect(150, 190, 421, 101))
        font = QtGui.QFont()
        font.setPointSize(20)
        self.failTitleUnauth.setFont(font)
        self.failTitleUnauth.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.failTitleUnauth.setAlignment(QtCore.Qt.AlignCenter)
        self.failTitleUnauth.setWordWrap(True)
        self.failTitleUnauth.setObjectName("failTitleUnauth")
        self.stackedWidget.addWidget(self.LOGIN_FAIL_UNAUTH)
        self.LOGIN_FAIL_TWO = QtWidgets.QWidget()
        self.LOGIN_FAIL_TWO.setObjectName("LOGIN_FAIL_TWO")
        self.failTitleUnauthTwo = QtWidgets.QLabel(self.LOGIN_FAIL_TWO)
        self.failTitleUnauthTwo.setGeometry(QtCore.QRect(150, 160, 431, 111))
        font = QtGui.QFont()
        font.setPointSize(20)
        self.failTitleUnauthTwo.setFont(font)
        self.failTitleUnauthTwo.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.failTitleUnauthTwo.setAlignment(QtCore.Qt.AlignCenter)
        self.failTitleUnauthTwo.setWordWrap(True)
        self.failTitleUnauthTwo.setObjectName("failTitleUnauthTwo")
        self.failTitleUnauthTwoWarning = QtWidgets.QLabel(self.LOGIN_FAIL_TWO)
        self.failTitleUnauthTwoWarning.setGeometry(QtCore.QRect(0, 430, 701, 51))
        font = QtGui.QFont()
        font.setPointSize(11)
        self.failTitleUnauthTwoWarning.setFont(font)
        self.failTitleUnauthTwoWarning.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(252, 175, 62);")
        self.failTitleUnauthTwoWarning.setAlignment(QtCore.Qt.AlignCenter)
        self.failTitleUnauthTwoWarning.setWordWrap(True)
        self.failTitleUnauthTwoWarning.setObjectName("failTitleUnauthTwoWarning")
        self.stackedWidget.addWidget(self.LOGIN_FAIL_TWO)
        self.SMS_VERIFICATION = QtWidgets.QWidget()
        self.SMS_VERIFICATION.setObjectName("SMS_VERIFICATION")
        self.smsLabel = QtWidgets.QLabel(self.SMS_VERIFICATION)
        self.smsLabel.setGeometry(QtCore.QRect(100, 150, 551, 31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.smsLabel.setFont(font)
        self.smsLabel.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.smsLabel.setObjectName("smsLabel")
        self.smsCode = QtWidgets.QLineEdit(self.SMS_VERIFICATION)
```

```
        self.smsCode.setGeometry(QtCore.QRect(210, 220, 361, 81))
        self.smsCode.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.smsCode.setAlignment(QtCore.Qt.AlignCenter)
        self.smsCode.setObjectName("smsCode")
        self.smsVerify = QtWidgets.QPushButton(self.SMS_VERIFICATION)
        self.smsVerify.setGeometry(QtCore.QRect(550, 400, 141, 41))
        font = QtGui.QFont()
        font.setPointSize(11)
        self.smsVerify.setFont(font)
        self.smsVerify.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.smsVerify.setStyleSheet("background-image: url(./images/transparent.
    png);\n"
            "")
        self.smsVerify.setObjectName("smsVerify")
        self.goback_3 = QtWidgets.QPushButton(self.SMS_VERIFICATION)
        self.goback_3.setGeometry(QtCore.QRect(0, 0, 111, 21))
        self.goback_3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.goback_3.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);")
        self.goback_3.setObjectName("goback_3")
        self.stackedWidget.addWidget(self.SMS_VERIFICATION)
        self.NEW_USER_FORM = QtWidgets.QWidget()
        self.NEW_USER_FORM.setObjectName("NEW_USER_FORM")
        self.newFirstnameLabel = QtWidgets.QLabel(self.NEW_USER_FORM)
        self.newFirstnameLabel.setGeometry(QtCore.QRect(60, 90, 91, 20))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.newFirstnameLabel.setFont(font)
        self.newFirstnameLabel.setStyleSheet("background-image: url(./images/
    transparent.png);\n"
            "color: rgb(255, 255, 255);\n"
            "")
        self.newFirstnameLabel.setObjectName("newFirstnameLabel")
        self.userInfoBtn = QtWidgets.QPushButton(self.NEW_USER_FORM)
        self.userInfoBtn.setGeometry(QtCore.QRect(600, 410, 141, 41))
        self.userInfoBtn.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.userInfoBtn.setStyleSheet("background-image: url(./images/
    transparent.png);")
        self.userInfoBtn.setObjectName("userInfoBtn")
        self.newFirstnameInput = QtWidgets.QLineEdit(self.NEW_USER_FORM)
        self.newFirstnameInput.setGeometry(QtCore.QRect(60, 110, 681, 41))
        self.newFirstnameInput.setStyleSheet("background-color: rgb(255, 255,
    255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.newFirstnameInput.setAlignment(QtCore.Qt.AlignCenter)
        self.newFirstnameInput.setObjectName("newFirstnameInput")
        self.registerTitle = QtWidgets.QLabel(self.NEW_USER_FORM)
        self.registerTitle.setGeometry(QtCore.QRect(230, 20, 341, 51))
        font = QtGui.QFont()
        font.setPointSize(30)
        self.registerTitle.setFont(font)
        self.registerTitle.setStyleSheet("background-image: url(./images/
    transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.registerTitle.setObjectName("registerTitle")
        self.newLastnameInput = QtWidgets.QLineEdit(self.NEW_USER_FORM)
        self.newLastnameInput.setGeometry(QtCore.QRect(60, 210, 681, 41))
```

```
        self.newLastnameInput.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.newLastnameInput.setEchoMode(QtWidgets.QLineEdit.Normal)
        self.newLastnameInput.setAlignment(QtCore.Qt.AlignCenter)
        self.newLastnameInput.setObjectName("newLastnameInput")
        self.newLastnameLabel = QtWidgets.QLabel(self.NEW_USER_FORM)
        self.newLastnameLabel.setGeometry(QtCore.QRect(60, 190, 81, 20))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.newLastnameLabel.setFont(font)
        self.newLastnameLabel.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.newLastnameLabel.setObjectName("newLastnameLabel")
        self.newPassInput = QtWidgets.QLineEdit(self.NEW_USER_FORM)
        self.newPassInput.setGeometry(QtCore.QRect(60, 410, 411, 41))
        self.newPassInput.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.newPassInput.setEchoMode(QtWidgets.QLineEdit.Password)
        self.newPassInput.setAlignment(QtCore.Qt.AlignCenter)
        self.newPassInput.setObjectName("newPassInput")
        self.newUsernameLabel = QtWidgets.QLabel(self.NEW_USER_FORM)
        self.newUsernameLabel.setGeometry(QtCore.QRect(60, 286, 121, 20))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.newUsernameLabel.setFont(font)
        self.newUsernameLabel.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.newUsernameLabel.setObjectName("newUsernameLabel")
        self.newUsernameInput = QtWidgets.QLineEdit(self.NEW_USER_FORM)
        self.newUsernameInput.setGeometry(QtCore.QRect(60, 310, 681, 41))
        self.newUsernameInput.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.newUsernameInput.setAlignment(QtCore.Qt.AlignCenter)
        self.newUsernameInput.setObjectName("newUsernameInput")
        self.newPassLabel = QtWidgets.QLabel(self.NEW_USER_FORM)
        self.newPassLabel.setGeometry(QtCore.QRect(60, 386, 121, 20))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.newPassLabel.setFont(font)
        self.newPassLabel.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.newPassLabel.setObjectName("newPassLabel")
        self.goback_4 = QtWidgets.QPushButton(self.NEW_USER_FORM)
        self.goback_4.setGeometry(QtCore.QRect(0, 0, 111, 21))
        self.goback_4.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.goback_4.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);")
        self.goback_4.setObjectName("goback_4")
        self.isAdminRadio = QtWidgets.QRadioButton(self.NEW_USER_FORM)
        self.isAdminRadio.setGeometry(QtCore.QRect(490, 410, 91, 41))
        font = QtGui.QFont()
        font.setPointSize(11)
```

```
        self.isAdminRadio.setFont(font)
        self.isAdminRadio.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.isAdminRadio.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;")
        self.isAdminRadio.setObjectName("isAdminRadio")
        self.stackedWidget.addWidget(self.NEW_USER_FORM)
        self.NEW_ADMIN_FORM = QtWidgets.QWidget()
        self.NEW_ADMIN_FORM.setObjectName("NEW_ADMIN_FORM")
        self.newSMSCodeInput = QtWidgets.QLineEdit(self.NEW_ADMIN_FORM)
        self.newSMSCodeInput.setGeometry(QtCore.QRect(40, 250, 481, 91))
        self.newSMSCodeInput.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.newSMSCodeInput.setAlignment(QtCore.Qt.AlignCenter)
        self.newSMSCodeInput.setObjectName("newSMSCodeInput")
        self.goback_5 = QtWidgets.QPushButton(self.NEW_ADMIN_FORM)
        self.goback_5.setGeometry(QtCore.QRect(0, 0, 111, 21))
        self.goback_5.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.goback_5.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);")
        self.goback_5.setObjectName("goback_5")
        self.newSMSVerify = QtWidgets.QPushButton(self.NEW_ADMIN_FORM)
        self.newSMSVerify.setGeometry(QtCore.QRect(570, 270, 141, 51))
        font = QtGui.QFont()
        font.setPointSize(11)
        self.newSMSVerify.setFont(font)
        self.newSMSVerify.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.newSMSVerify.setStyleSheet("background-color: rgb(190, 206, 221);\n"
            "background-image: url(./images/transparent.png);")
        self.newSMSVerify.setObjectName("newSMSVerify")
        self.newNumberInfo = QtWidgets.QTextEdit(self.NEW_ADMIN_FORM)
        self.newNumberInfo.setGeometry(QtCore.QRect(40, 30, 741, 81))
        self.newNumberInfo.setStyleSheet("background-color: rgb(190, 206, 221);\n"
            "color: rgb(52, 101, 164);")
        self.newNumberInfo.setObjectName("newNumberInfo")
        self.newPhoneNumberInput = QtWidgets.QLineEdit(self.NEW_ADMIN_FORM)
        self.newPhoneNumberInput.setGeometry(QtCore.QRect(40, 130, 481, 51))
        self.newPhoneNumberInput.setStyleSheet("background-color: rgb(255, 255, 255);\n"
            "background-image: url(./images/transparent.png);\n"
            "border-radius: 6px;\n"
            "")
        self.newPhoneNumberInput.setText("")
        self.newPhoneNumberInput.setMaxLength(14)
        self.newPhoneNumberInput.setAlignment(QtCore.Qt.AlignCenter)
        self.newPhoneNumberInput.setObjectName("newPhoneNumberInput")
        self.newSendSMSCode = QtWidgets.QPushButton(self.NEW_ADMIN_FORM)
        self.newSendSMSCode.setGeometry(QtCore.QRect(570, 130, 141, 51))
        font = QtGui.QFont()
        font.setPointSize(11)
        self.newSendSMSCode.setFont(font)
        self.newSendSMSCode.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.newSendSMSCode.setStyleSheet("background-color: rgb(190, 206, 221);\n"
            "background-image: url(./images/transparent.png);")
        self.newSendSMSCode.setObjectName("newSendSMSCode")
        self.newNumberSkipWarning = QtWidgets.QTextEdit(self.NEW_ADMIN_FORM)
        self.newNumberSkipWarning.setGeometry(QtCore.QRect(40, 360, 741, 51))
```

```
        self.newNumberSkipWarning.setStyleSheet("color: rgb(164, 0, 0);\n"
            "background-color: rgb(252, 175, 62);")
        self.newNumberSkipWarning.setObjectName("newNumberSkipWarning")
        self.skipNewNumber = QtWidgets.QPushButton(self.NEW_ADMIN_FORM)
        self.skipNewNumber.setGeometry(QtCore.QRect(570, 420, 141, 51))
        font = QtGui.QFont()
        font.setPointSize(11)
        self.skipNewNumber.setFont(font)
        self.skipNewNumber.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.skipNewNumber.setStyleSheet("background-color: rgb(164, 0, 0);\n"
            "background-color: rgb(252, 175, 62);\n"
            "background-image: url(./images/transparent.png);")
        self.skipNewNumber.setObjectName("skipNewNumber")
        self.stackedWidget.addWidget(self.NEW_ADMIN_FORM)
        self.REGISTER_SUCCESS = QtWidgets.QWidget()
        self.REGISTER_SUCCESS.setObjectName("REGISTER_SUCCESS")
        self.regSuccess = QtWidgets.QLabel(self.REGISTER_SUCCESS)
        self.regSuccess.setGeometry(QtCore.QRect(50, 210, 681, 31))
        font = QtGui.QFont()
        font.setPointSize(17)
        self.regSuccess.setFont(font)
        self.regSuccess.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.regSuccess.setObjectName("regSuccess")
        self.stackedWidget.addWidget(self.REGISTER_SUCCESS)
        self.REGISTER_FAIL = QtWidgets.QWidget()
        self.REGISTER_FAIL.setObjectName("REGISTER_FAIL")
        self.regFail = QtWidgets.QLabel(self.REGISTER_FAIL)
        self.regFail.setGeometry(QtCore.QRect(250, 210, 291, 41))
        font = QtGui.QFont()
        font.setPointSize(24)
        self.regFail.setFont(font)
        self.regFail.setStyleSheet("background-image: url(./images/transparent.png);\n"
            "color: rgb(255, 255, 255);")
        self.regFail.setObjectName("regFail")
        self.stackedWidget.addWidget(self.REGISTER_FAIL)
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        self.stackedWidget.setCurrentIndex(11)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "FRACS"))
        self.label.setText(_translate("MainWindow", "WELCOME to FRACS"))
        self.login.setText(_translate("MainWindow", "Login"))
        self.registration.setText(_translate("MainWindow", "Registration"))
        self.unameLabel.setText(_translate("MainWindow", "Username:"))
        self.passLabel.setText(_translate("MainWindow", "Password:"))
        self.usernameInput.setPlaceholderText(_translate("MainWindow", "Enter Username"))
        self.passwordInput.setPlaceholderText(_translate("MainWindow", "Enter Password"))
        self.goback.setText(_translate("MainWindow", "Main Menu"))
        self.loginTitle.setText(_translate("MainWindow", "Login to access room"))
        self.loginBtn.setText(_translate("MainWindow", "Next"))
        self.camButton.setText(_translate("MainWindow", "Click here to start camera, when you\'re ready!"))
        self.goback_2.setText(_translate("MainWindow", "Main Menu"))
```

```
        self.successTitle.setText(_translate("MainWindow", "Login Successful :D Door unlocking.."))
        self.failTitle.setText(_translate("MainWindow", "Login Failed :( Please try again.."))
        self.failTitleZero.setText(_translate("MainWindow", "Login Failed :( No face detected. Please stand in front of camera"))
        self.failTitleUnauth.setText(_translate("MainWindow", "Login Failed :( User unrecognized. Are you sure you have registered with the FRACS system?"))
        self.failTitleUnauthTwo.setText(_translate("MainWindow", "Login Failed :( More than one face detected. Please ensure that only you stand in fron of the camera"))
        self.failTitleUnauthTwoWarning.setText(_translate("MainWindow", "*Note that unregistered users are not allowed to access room. If other person(s) is /are unregistered, you will be responsible if they gain access to room"))
        self.smsLabel.setText(_translate("MainWindow", "Admin, enter the 6 digit code that was sent to your phone:"))
        self.smsCode.setPlaceholderText(_translate("MainWindow", "Enter SMS Code"))
        self.smsVerify.setText(_translate("MainWindow", "Verify"))
        self.goback_3.setText(_translate("MainWindow", "Main Menu"))
        self.newFirstnameLabel.setText(_translate("MainWindow", "First Name:"))
        self.userInfoBtn.setText(_translate("MainWindow", "Next"))
        self.newFirstnameInput.setPlaceholderText(_translate("MainWindow", "Enter First Name"))
        self.registerTitle.setText(_translate("MainWindow", "Register to FRACS"))
        self.newLastnameInput.setPlaceholderText(_translate("MainWindow", "Enter Last Name"))
        self.newLastnameLabel.setText(_translate("MainWindow", "Last Name:"))
        self.newPassInput.setPlaceholderText(_translate("MainWindow", "Enter Password"))
        self.newUsernameLabel.setText(_translate("MainWindow", "Login Username:"))
        self.newUsernameInput.setPlaceholderText(_translate("MainWindow", "Enter Username"))
        self.newPassLabel.setText(_translate("MainWindow", "Login Password:"))
        self.goback_4.setText(_translate("MainWindow", "Main Menu"))
        self.isAdminRadio.setText(_translate("MainWindow", " Admin?"))
        self.newSMSCodeInput.setPlaceholderText(_translate("MainWindow", "Enter SMS Code"))
        self.goback_5.setText(_translate("MainWindow", "Main Menu"))
        self.newSMSVerify.setText(_translate("MainWindow", "Verify + finish!"))
        self.newNumberInfo.setHtml(_translate("MainWindow", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
            "<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
            "p, li { white-space: pre-wrap; }\n"
            "</style></head><body style=\" font-family:\'Ubuntu\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
            "<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-size:14pt;\">Please enter your phone number in the below field, then click \'Send Code\'. </span></p>\n"
            "<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-size:14pt;\">You will get a 6-digit code to that phone number, which you will need to enter below.</span></p></body></html>"))
        self.newPhoneNumberInput.setPlaceholderText(_translate("MainWindow", "Enter Phone Number in this format: +353xxxxxxxxx"))
        self.newSendSMSCode.setText(_translate("MainWindow", "Send Code"))
        self.newNumberSkipWarning.setHtml(_translate("MainWindow", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
```

```
            "<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
            "p, li { white-space: pre-wrap; }\n"
            "</style></head><body style=\" font-family:\'Ubuntu\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
            "<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">Don\'t have your phone? You can skip this part and do it on the FRACS Website later. </p>\n"
            "<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">However, you will not be able to register new users until you verify your phone number!</p></body></html>"))
        self.skipNewNumber.setText(_translate("MainWindow", "Skip"))
        self.regSuccess.setText(_translate("MainWindow", "Registration Successful! You can now gain access by logging in :D"))
        self.regFail.setText(_translate("MainWindow", "Registration Failed.."))
```

Listing 26: fracsUI.py

2.7.2 Pages Fading UI

Besides the main UI used for the FRACS desktop application, there is a code snippet that was used to perform a fading effect between pages of the application. This snippet of code was written and uploaded by user DavidBoddie on the official Python Wiki website. [1]

```
import sys
from PyQt5.QtCore import QTimeLine
from PyQt5.QtGui import *
from PyQt5 import QtWidgets, QtCore, QtGui
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QWidget

class FaderWidget(QWidget):
    def __init__(self, old_widget, new_widget):
        QWidget.__init__(self, new_widget)
        self.old_pixmap = QtGui.QPixmap(new_widget.size())
        old_widget.render(self.old_pixmap)
        self.pixmap_opacity = 1.0

        self.timeline = QTimeLine()
        self.timeline.valueChanged.connect(self.animate)
        self.timeline.finished.connect(self.close)
        self.timeline.setDuration(333)
        self.timeline.start()

        self.resize(new_widget.size())
        self.show()

    def paintEvent(self, event):
        painter = QPainter()
        painter.begin(self)
        painter.setOpacity(self.pixmap_opacity)
        painter.drawPixmap(0, 0, self.old_pixmap)
        painter.end()

    def animate(self, value):
        self.pixmap_opacity = 1.0 - value
        self.repaint()
```

Listing 27: uifade.py

[1] https://wiki.python.org/moin/PyQt/Fading%20Between%20Widgets

2.8 Main Code - fracs.py

The main code file, fracs.py, is the starting point of the whole FRACS application. It imports all the described files above and starts the application, carrying out various operations using the imported files.

The *main()* function in the file initializes the app and starts the GUI. The Qt application is contained in a class called MainWindow()

```python

import os
import sys
import time

import fracsUI
import cameraCountdown
import rekognitionOperations
import dbOperations
import twilioOperations
import lockOperation

from pyfade import FaderWidget
from PyQt5 import QtWidgets, QtCore, QtGui
from PyQt5.QtCore import QTimer
from PyQt5.QtGui import QMovie, QPainter, QPixmap

from passlib.hash import pbkdf2_sha256

def main():
    app = QtWidgets.QApplication(sys.argv)
    win = MainWindow()
    win.show()

    app.exec_()

if __name__ == '__main__':
    main()
```

Listing 28: fracs.py - main() - app initialization

The *init()* function is responsible for attaching the GUI components to the UI imported. It is also used for attaching button listeners and event handlers.

```python
def __init__(self, parent=None):
    #ui setup
    super(MainWindow, self).__init__(parent)

    self.ui = fracsUI.Ui_MainWindow()
    self.ui.setupUi(self)

    #attach button event listeners/handlers
    self.attachButtonListeners()

    #splash screen appears for 3 secs then redirects to main menu
    QTimer.singleShot(3000, lambda: self.change(1))
```

Listing 29: fracs.py - MainWindow.init()

The *attachButtonListeners()* method attaches listeners to all buttons that exist in the application

```python
    def attachButtonListeners(self): #each button in the desktop app is assigned a listener/handler
    self.ui.stackedWidget.setCurrentIndex(0)
    self.ui.login.clicked.connect(lambda: self.showLoginForm("USER"))
    self.ui.loginBtn.clicked.connect(lambda: self.change(3))

    self.ui.camButton.clicked.connect(self.openCamera)

    self.ui.registration.clicked.connect(lambda: self.showLoginForm("ADMIN"))
    self.ui.smsVerify.clicked.connect(self.verifyAdminSMS)

    self.ui.userInfoBtn.clicked.connect(self.UsernameAvailability)
    self.ui.newSendSMSCode.clicked.connect(self.verifyNewAdmin)
    self.ui.skipNewNumber.clicked.connect(lambda: self.change(12))

    #self.ui.newSMSVerify.clicked.connect(self.verifyNewAdminSMS) # will not work !!!!!!!!! to register numbers using API, twilio must be upgraded!!!!!

    self.ui.goback.clicked.connect(self.restart)
    self.ui.goback_2.clicked.connect(self.restart)
    self.ui.goback_3.clicked.connect(self.restart)
    self.ui.goback_4.clicked.connect(self.restart)
    self.ui.goback_5.clicked.connect(self.restart)
```

Listing 30: fracs.py - MainWindow.init()

The *change()* function takes in a number, which indicates the index of the StackedWidget (the widget that controls the pages of the application). Each index points to a page in the StackedWidget, and when the change() function is called, the page that corresponds to the index appears.
The function also implements the fading between the pages.

```python
def change(self, index):
    self.fader_widget = FaderWidget(self.ui.stackedWidget.currentWidget(), self.ui.stackedWidget.widget(index))
    self.ui.stackedWidget.setCurrentIndex(index)
```

Listing 31: fracs.py - MainWindow.change()

The *showLoginForm()* function displays the login form depending on whether the intended action is login or register. If the role (declared as a global variable) is admin, the login form appears with a suitable message title, and it the role is user, the login form appears with another suitable message title.

```python
def showLoginForm(self, role):
    global approle
    approle = role
    if role == "ADMIN":
        self.ui.loginTitle.setText("Admin Login for Registration")
        self.ui.loginTitle.setGeometry(QtCore.QRect(120, 50, 500, 91))
        self.change(2)
    elif role == "USER":
        self.ui.loginTitle.setText("Login to access room")
        self.ui.loginTitle.setGeometry(QtCore.QRect(200, 50, 381, 51))
        self.change(2)
    else:
        self.change(2)
```

Listing 32: fracs.py - MainWindow.showLoginForm()

The *openCamera()* function opens the camera page that informs the user that an image will be taken of their face, giving them a chance to appear in front of the camera and be prepared.

```
def openCamera(self):
    cameraCountdown.main()
    if approle == "NEW": # new user i.e. registration process
        self.checkNewUserInfo()
    else:
        self.validateFaces(approle)
```

Listing 33: fracs.py - MainWindow.openCamera()

The *validateFaces()* function carries out more than one operation. It is responsible for ensuring that the image captured contains only one person, who is registered with the system. Here is also where the lock will be triggered if the user is authenticated and authorized to gain access.
It is responsible for logging all the login attempts and also for displaying success/fail messages to the screen.

```
def validateFaces(self, role):
    face_id = rekognitionOperations.confirmUserFace()
    '''
    response codes: 0 = no face detected
    1 = face detected but unknown (not registered with fracs)
    -1 = more than one face detected
    '''
    if face_id is not None:
        if face_id == 0: # no face detected
            self.logEvent("ZERO", "FAIL")
            self.change(6)
            QTimer.singleShot(3000, self.restart)
        elif face_id == -1:
            self.logEvent("MANY", "FAIL")
            self.change(8) # more than one face detected
            QTimer.singleShot(3000, self.restart)
        elif face_id == 1: # face detected, but unrecognized user
            self.logEvent("UNKNOWN", "FAIL")
            self.change(7)
            QTimer.singleShot(3000, self.restart)

        else:
            valid = self.validateLogin(face_id)
            if valid:
                if role == "ADMIN":
                self.change(9)
                self.verifyAdmin()

            elif role == "USER":
                self.change(4)
                self.unlock()

            else:
                self.change(5)
                QTimer.singleShot(3000, self.restart)

    else:
        self.change(5)
        QTimer.singleShot(3000, self.restart)
```

Listing 34: fracs.py - MainWindow.validateFaces()

The *logEvent()* function takes in parameters related to the login attempt and calls functions from the dbOperations.py file to do this.

```
def logEvent(self, faceid, status):
    userInfo = {
        "username_entered" : self.ui.usernameInput.text(),
        "face_id" : faceid,
        "attempted_user_id" : dbOperations.getAttemptedUserId(self.ui.usernameInput.text()),
        "login_datetime" : time.strftime('%Y-%m-%d %H:%M:%S'),
        "is_successful" : status
    }

    dbOperations.logAccessEvent(userInfo)
```

Listing 35: fracs.py - MainWindow.logEvent()

The *unlock()* function triggers the lockOperation.py file, which in turn triggers the lock itself, allowing access into the room.

```
def unlock(self):
    QTimer.singleShot(2000, lockOperation.main)
    QTimer.singleShot(3000, self.restart)
```

Listing 36: fracs.py - MainWindow.unlock()

The *validateLogin()* function is not to be confused with the *validateFaces()* function. This function is mainly responsible for ensuring that the password entered by the user matches the password stored in the local database and is correct, whereas the *validateFaces()* function is responsible for the biometric aspect of authentication.
This function takes place depending on the output of the *validateFaces()* function.

```
def validateLogin(self, face_id):
    global approle
    hashedPassword = ""

    if approle == "ADMIN":
        hashedPassword = dbOperations.getAdminHashedPassword(self.ui.usernameInput.text(), face_id)
    elif approle == "USER":
        hashedPassword = dbOperations.getHashedPassword(self.ui.usernameInput.text(), face_id)

    if hashedPassword is not None:
        if pbkdf2_sha256.verify(self.ui.passwordInput.text(), hashedPassword): # credentials match with face
            self.logEvent(face_id, "SUCCESS")
            return True
        else:
            self.logEvent(face_id, "FAIL")
            return False
    else:
        self.logEvent(face_id, "FAIL")
        return False
```

Listing 37: fracs.py - MainWindow.init()

The following two functions go hand-in-hand for ensuring that the admin verification process is correct.

```
def verifyAdmin(self):
    adminPhoneNum = dbOperations.getAdminPhoneNumber(self.ui.usernameInput.text()
    )
    if adminPhoneNum is not None:
        msgSent = twilioOperations.sendSMSCode(adminPhoneNum)
    else:
        ShowError("invalid admin phone number/issue")

def verifyAdminSMS(self):
    success = twilioOperations.VerifyTwilioSMS(self.ui.smsCode.text())
    if success:
        self.logEvent()
        self.change(10)
    else:
        self.ShowError("invalid code entered.")
```

Listing 38: fracs.py - MainWindow.verifyAdmin&SMS()

The *checkNewUserInfo()* function takes care of registering new users to the system by inserting their data in the local database.

```
def checkNewUserInfo(self):
    is_admin = 0

    if self.ui.isAdminRadio.isChecked():
        is_admin = 1

    faceAvailable = rekognitionOperations.confirmUserFace()

    if faceAvailable == 1:
        newFaceId= rekognitionOperations.addNewFace()
        newUser = {
            "firstname": self.ui.newFirstnameInput.text(),
            "lastname": self.ui.newLastnameInput.text(),
            "login_username": self.ui.newUsernameInput.text(),
            "login_password": pbkdf2_sha256.hash(self.ui.newPassInput.text()),
            "is_admin" : is_admin,
            "face_id" : newFaceId,
            "admin_registered" : self.ui.usernameInput.text()
        }
        dbOperations.addNewUser(newUser)
        if self.ui.isAdminRadio.isChecked():
            self.change(11)
        else:
            self.change(12)
            QTimer.singleShot(3000, self.restart)

    else:
        self.change(13)
        QTimer.singleShot(4000, self.restart)
```

Listing 39: fracs.py - MainWindow.checkNewUserInfo()

The *UsernameAvailability()* function returns an error to the currently registering user, informing them that the username they wish to register with is not available.

```python
def UsernameAvailability(self):
    global approle

    username = self.ui.newUsernameInput.text()
    availability = dbOperations.checkUsername(username)
    if not availability:
        self.ShowError("Username entered is not available.")
        return False
    else:
        approle = "NEW"
        self.change(3)
        return True
```

Listing 40: fracs.py - MainWindow.UsernameAvailability()

The following two functions do the same thing as the *verifyAdmin()* and *verifyAdminSMS()* functions described above, respectively. Except, the following two functions are specified for newly registered admins.

```python
def verifyNewAdmin(self):
    newAdminPhoneNum = self.ui.newPhoneNumberInput.text()
    if newAdminPhoneNum is not None:
        msgSent = twilioOperations.sendSMSCode(newAdminPhoneNum)
    else:
        ShowError("invalid admin phone number/issue")


def verifyNewAdminSMS(self):
    success = twilioOperations.VerifyTwilioSMS(self.ui.newSMSCodeInput.text())
    if success:
        self.change(12)
    else:
        self.change(5)
    QTimer.singleShot(3000, self.restart)
```

Listing 41: fracs.py - MainWindow.verifyNewAdmin&SMS()

Very simply, the *showError()* function displays a warning dialog with the error message, which they can close down and modify/change their provided values.

```python
def ShowError(self, errorMsg):
    QtWidgets.QMessageBox.warning(self, 'Error', errorMsg)
```

Listing 42: fracs.py - MainWindow.showError()

Finally, *restart()* is responsible for restarting the application. It does this by clearing all text inputs in the application and deleting the image captured at the start permanently.

```python
def restart(self):
    self.ui.passwordInput.clear()
    self.ui.usernameInput.clear()
    self.ui.smsCode.clear()
    self.ui.newFirstnameInput.clear()
    self.ui.newLastnameInput.clear()
    self.ui.newPassInput.clear()
    self.ui.newUsernameInput.clear()
    self.ui.newPhoneNumberInput.clear()
    self.ui.newSMSCodeInput.clear()
    self.ui.isAdminRadio.setChecked(False)
    self.change(0)
    os.remove("user.jpg") # deletes image, so that no PII remain on the system.
    QTimer.singleShot(3000, lambda: self.change(1))
```

Listing 43: fracs.py - MainWindow.restart()

3 Web Application (Flask Webserver)

For Flask to work properly, there has to be a certain directory structure put in place. The "static" and "templates" folders have to exist for Flask to identify. The directory structure is as follows:

```
flaskTut2
├── server.py
├── static
│   ├── all.css
│   ├── jquery.dataTables.min.js
│   └── signin.css
└── templates
    ├── dashboard.html
    ├── index.html
    ├── login.html
    ├── manage_users.html
    ├── my_profile.html
    └── view_user_logs.html

2 directories, 10 files
h@da:~/Desktop$
```

Figure 1: Web App Directory Structure

The main server file that contains all the webpages configuration and login security is called server.py. It does as follows:

```
from flask import Flask, render_template, redirect, url_for
from flask_bootstrap import Bootstrap
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField
from wtforms.validators import InputRequired, Length
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import cast, Date, and_
from passlib.hash import pbkdf2_sha256
from flask_login import LoginManager, UserMixin, login_user, login_required,
    current_user, logout_user
from datetime import datetime , timedelta
from dateutil.relativedelta import relativedelta


app = Flask(__name__)
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'XXXXXXX'
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://hoda:XXXX@localhost/FracsDB'
                                        #mysql://username:password@host/database

db = SQLAlchemy(app)
Bootstrap(app)
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

class FracsUsers(UserMixin, db.Model):
    __tablename__ = "fracs_users"
    user_id = db.Column(db.Integer, primary_key=True)
    firstname = db.Column(db.String(255))
    lastname = db.Column(db.String(255))
    login_username = db.Column(db.String(255), unique=True)
    login_password = db.Column(db.String(255))
    is_admin = db.Column(db.Boolean)
    phone_number = db.Column(db.String(15))
    face_id = db.Column(db.String(255))
    admin_registered = db.Column(db.String(255))

def get_id(self):
    return self.user_id

class FracsLogs(UserMixin, db.Model):
    __tablename__ = "fracs_logs"
    log_id = db.Column(db.Integer, primary_key=True)
    username_entered = db.Column(db.String(255), unique=True)
    attempted_user_id = db.Column(db.Integer)
    face_id =  db.Column(db.String(255))
    login_datetime = db.Column(db.DateTime)
    is_successful = db.Column(db.String(10))

def generatePieChart(type):
    today = datetime.now()#.replace(hour=0, minute=0, second=0, microsecond=0)
    weekAgo=0
    monthAgo=0

    if type == "weekly":
        weekAgo = (today - timedelta(days = 7)).replace(hour=0, minute=0, second
    =0, microsecond=0) # .replace gets rid of miliseconds
        result = FracsLogs.query.filter(and_(FracsLogs.login_datetime <= today ,
    FracsLogs.login_datetime >= weekAgo)).all()
```

```
    else:
        monthAgo = today+relativedelta(months=-1)
        result = FracsLogs.query.filter(and_(FracsLogs.login_datetime <= today , FracsLogs.login_datetime >= monthAgo)).all()

    successfulAttempts = 0
    failedAttempts = 0

    for attempt in result:
        if attempt.is_successful == "SUCCESS":
        successfulAttempts+=1
    else:
        failedAttempts+=1

    labels = [
        'SUCCESS', 'FAIL'
    ]

    values = [
        successfulAttempts, failedAttempts
    ]

    colors = [
        "#46BFBD", "#F7464A"
    ]

    return zip(values, labels, colors)


@app.route('/pie')
def pie():
    logs = FracsLogs.query.order_by(FracsLogs.login_datetime.desc()).all()
    return render_template('test2.html')

@login_manager.user_loader
    def load_user(user_id):
    return FracsUsers.query.get(int(user_id))

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[InputRequired(), Length(min=4, max=15)])
    password = PasswordField('Password', validators=[InputRequired()])
    remember = BooleanField('Remember Me')

@app.route('/', methods=['GET','POST'])
def index():
    if current_user.is_authenticated:
    return redirect(url_for('dashboard'))
    return render_template('index.html')

@app.route('/login', methods=['GET','POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit(): #if form is submitted & all validators returned true
        user = FracsUsers.query.filter_by(login_username=form.username.data).first() # fetch first result from results where the username in db = username provided in form
        if user: #if there IS  a result
            if pbkdf2_sha256.verify(form.password.data, user.login_password ): # and password hashes match
                login_user(user, remember=form.remember.data)
                return redirect(url_for('dashboard'))
```

```
    return render_template('login.html', form=form)

@app.route('/dashboard')
@login_required
def dashboard():
    if isAdmin() > 0:
        logs = FracsLogs.query.order_by(FracsLogs.login_datetime.desc()).limit(10).all()
        pi1 = generatePieChart(type="weekly")
        pi2 = generatePieChart(type="monthly")
        return render_template('dashboard.html', name=current_user.login_username, logs=logs, set=pi1, set2=pi2)

    else:
        return my_profile()

def isAdmin():
    return FracsUsers.query.filter_by(login_username=current_user.login_username, is_admin=1).count() #number of entries in table that have login_username=current user and is_admin=1..... if user is admin, result must have 1 entry, else 0.

@app.route('/my_profile', methods=['GET','POST'])
@login_required
def my_profile():
    userProfile = FracsUsers.query.filter_by(login_username=current_user.login_username).first()
    return render_template('my_profile.html', userprofile=userProfile)

@app.route('/change_password')
@login_required
def change_password():
    return render_template('change_password.html')

@app.route('/view_user_logs')
@login_required
def view_user_logs():
    logs = FracsLogs.query.order_by(FracsLogs.login_datetime.desc()).all()
    return render_template('view_user_logs.html', userslogs=logs)

@app.route('/all_users')
@login_required
def manage_users():
    users = FracsUsers.query.order_by(FracsUsers.firstname).all()
    return render_template('manage_users.html', allusers=users)

@app.route('/logout')
def logout():
    logout_user()
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
#host='0.0.0.0' means web app is accessible to any device on the network
```

Listing 44: server.py

```
{% extends "bootstrap/base.html" %}

{% block title %}
    FRACS
{% endblock %}

{% block styles %}
    {{super()}}
    <link rel="stylesheet" href="{{url_for('.static', filename='starter-template.
   css')}}">
{% endblock %}

{% block content %}
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed" data-toggle
   ="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar
   ">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="#">PI FRACS</a>
            </div>
            <div id="navbar" class="collapse navbar-collapse">
                <ul class="nav navbar-nav">
                    <li class="active"><a href="#">Home</a></li>
                <li><a href="{{ url_for('login') }}">Login</a></li>
                </ul>
            </div><!--/.nav-collapse -->
        </div>
    </nav>
    <div class="container">

    <div class="starter-template">
    <h1>Welcome to FRACS Web!</h1>
    <p class="lead">Here you can control FRACS<br>And its users</p>
    </div>

    </div><!-- /.container -->
{% endblock %}
```

Listing 45: index.html

```
{% extends "bootstrap/base.html" %}
{% import "bootstrap/wtf.html" as wtf %}

{% block title %}
    Login
{% endblock %}

{% block styles %}
    {{super()}}
    <link rel="stylesheet" href="{{url_for('.static', filename='signin.css')}}">
{% endblock %}

{% block content %}
    <div class="container">
        <form class="form-signin" method="POST" action="/login">
            <h2 class="form-signin-heading"> Please sign in </h2>
            {{ form.csrf_token }}
            {{ form.hidden_tag() }}
            {{ wtf.form_field(form.username) }}
            {{ wtf.form_field(form.password) }}
            {{ wtf.form_field(form.remember) }}

            <!-- OR -----
            {{wtf.quick_form}} -->
            <input class="btn btn-primary" type="submit" value="Login">
        </form>
    </div>
{% endblock %}
```

Listing 46: login.html

```
{% extends "bootstrap/base.html" %}

{% block title %}
    Dashboard
{% endblock %}

{% block styles %}
    {{super()}}
    <link rel="stylesheet" href="{{url_for('.static', filename='all.css')}}">
    <script src='https://cdnjs.cloudflare.com/ajax/libs/Chart.js/1.0.2/Chart.min.
    js'></script>
{% endblock %}

{% block content %}
    <!-- TOP NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed" data-toggle
    ="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar
    ">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="#">FRACS Web</a>
            </div>
            <div id="navbar" class="navbar-collapse collapse">
                <ul class="nav navbar-nav navbar-right">
                    <li><a href="/logout">Log Out</a></li>
                </ul>
```

```
            </div>
        </div>
    </nav>
    <!-- TOP NAV END !!!!!!!!!!!!!!!!!!!! -->

    <!-- SIDE NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
    <div class="container-fluid">
        <div class="row">
            <div class="col-sm-3 col-md-2 sidebar">
                <ul class="nav nav-sidebar">
                    <li class="active"><a href="#">Dashboard</a> <span class="sr-only">(current)</span></li>
                    <li><a href="/all_users">FRACS Users</a></li>
                    <li><a href="/view_user_logs">View Access Logs </a></li>
                </ul>
                <ul class="nav nav-sidebar">
                    <li><a href="/my_profile">User Profile</a></li>
                </ul>
            </div>

    <!-- SIDE NAV END !!!!!!!!!!!!!!!!!!!! -->

            <div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
                <h1 class="page-header">Welcome, {{name}}!</h1>

                <div class="row placeholders">
                    <div class="col-xs-3 col-sm-6 placeholder">
                        <canvas id="pieChart1" width="300" height="200"></canvas>
                        <h4>Access Attempts</h4>
                        <span class="text-muted">in previous week</span>
                    </div>
<script>
    var pieData = [
        {% for item, label, colors in set %}
            {
                value: {{item}},
                label: "{{label}}",
                color : "{{colors}}"
            },
        {% endfor %}
    ];

    // draw pie pieChart1
    new Chart(document.getElementById("pieChart1").getContext("2d")).Pie(pieData);
</script>

                    <div class="col-xs-3 col-sm-6 placeholder">
                        <canvas id="pieChart2" width="300" height="200"></canvas>

                        <h4>Access Attempts</h4>
                        <span class="text-muted">in previous month</span>
                    </div>

<script>
    var pie2Data = [
        {% for item, label, colors in set2 %}
            {
                value: {{item}},
                label: "{{label}}",
                color : "{{colors}}"
            },
        {% endfor %}
```

```
    ];

    new Chart(document.getElementById("pieChart2").getContext("2d")).Pie(pie2Data
   );
</script>
                </div>

                <!-- TABLE BEGIN!!!!!!!!!!!! -->
                <h2 class="sub-header">Latest Access Logs</h2>
                <div class="table-responsive">
                    <table class="table table-striped table-hover" border="4"
   cellpadding="5" cellspacing="5" style="border-color: black;">
                        <thead>
                            <tr class="tbl-headings">
                                <th>#</th>
                                <th>Username Entered</th>
                                <th>Actual User ID</th>
                                <th>Date & Time</th>
                                <th>Access Granted?</th>

                            </tr>
                        </thead>
                        <tbody >
                            {% for row in logs %}
                                {% if row.is_successful == "FAIL" %}
                                <tr class="danger" >
                                {% else %}
                                <tr class="success">
                                {% endif %}
                                    <td>{{ loop.index }}</td>
                                    <td>{{ row.username_entered }}</td>
                                    <td>{{ row.attempted_user_id }} </td>
                                    <td>{{ row.login_datetime }}</td>
                                    <td>{{ row.is_successful }}</td>
                                </tr>
                            {% endfor %}
                        </tbody>
                    </table>
                </div>
<!-- TABLE END!!!!!!!!!!!! -->
            </div>
        </div>
    </div>
{% endblock %}
```

Listing 47: dashboard.html

```
{% extends "bootstrap/base.html" %}

{% block title %}
Dashboard
{% endblock %}

{% block styles %}
{{super()}}
<link rel="stylesheet" href="{{url_for('.static', filename='all.css')}}">
<script src='https://cdnjs.cloudflare.com/ajax/libs/Chart.js/1.0.2/Chart.min.js'>
    </script>
{% endblock %}

{% block content %}
<!-- TOP NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
<nav class="navbar navbar-inverse navbar-fixed-top">
<div class="container-fluid">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data
    -target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">FRACS Web</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<ul class="nav navbar-nav navbar-right">
<li><a href="/logout">Log Out</a></li>
</ul>
</div>
</div>
</nav>
<!-- TOP NAV END !!!!!!!!!!!!!!!!!!!! -->

<!-- SIDE NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
<div class="container-fluid">
<div class="row">
<div class="col-sm-3 col-md-2 sidebar">
<ul class="nav nav-sidebar">
<li class="active"><a href="#">Dashboard</a> <span class="sr-only">(current)</
    span></li>
<li><a href="/all_users">FRACS Users</a></li>
<li><a href="/view_user_logs">View Access Logs </a></li>
</ul>
<ul class="nav nav-sidebar">
<li><a href="/my_profile">User Profile</a></li>
</ul>
</div>

<!-- SIDE NAV END !!!!!!!!!!!!!!!!!!!! -->


<!-- LOGS TABLE START !!!!!!!!!!!!!!!!!!!! -->

<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
<h2 class="sub-header">FRACS Users</h2>
<div class="table-responsive">
<table id="myTable" class="table table-striped table-hover" border="1"
    cellpadding="5" cellspacing="5" style="border-color: black;">
<thead>
<tr class="tbl-headings">
```

```
<th>ID</th>
<th>Name</th>
<th>Username</th>
<th>Admin?</th>
<th>Phone Number</th>
<th>Face ID</th>
<th>Admin Registered</th>
<th> </th>
</tr>
</thead>
<tbody >
{% for row in allusers %}
<td>{{ row.user_id}}</td>
<td>{{ row.firstname }} {{ row.lastname }}</td>
<td>{{ row.login_username }}</td>
<td>{{ row.is_admin }}</td>
<td>{{ row.phone_number }}</td>
<td>{{ row.face_id }}</td>
<td>{{ row.admin_registered }}</td>
<td><button type="button" data-toggle="modal" data-target="#myOutput{{row.user_id
    }}" class="btn btn-small btn-success btn-xs">?</button></td>
</tr>



{% endfor %}
</tbody>
</table>
<!-- LOGS TABLE END !!!!!!!!!!!!!!!!!!!! -->

{% for row in allusers %}
<!-- Modal -->
<div class="modal fade" id="myOutput{{row.user_id}}" role="dialog">
<div class="modal-dialog modal-lg">

<!-- Modal content-->
<div class="modal-content">
<div class="modal-header">
<button type="button" class="close" data-dismiss="modal">&times;</button>
<h4 class="modal-title">Task Output</h4>
</div>
<div class="modal-body">
<div class="form-group">
<input type="text" class="form-control" value="Log ID: {{ row.user_id }}">
</div>
<div class="form-group">
<input type="text" class="form-control" value="Name: {{row.firstname}} {{row.
    lastname}}">
</div>
<div class="form-group">
<input type="text" class="form-control" value="Login Username: {{row.
    login_username}}"  name="designation">
</div>
<div class="form-group">
<input type="text" class="form-control" value="Admin?: {{row.is_admin}}">
</div>
<div class="form-group">
<input type="text" class="form-control" value="Phone Number: {{row.phone_number}}
    ">
</div>
<div class="form-group">
<input type="text" class="form-control" value="Face ID: {{row.face_id}}">
</div>
```

```
<div class="form-group">
<input type="text" class="form-control" value="Admin Registered: {{row.
    admin_registered}}">
</div>

</div>
<div class="modal-footer">
<button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
</div>
</div>

</div>
</div>
{% endfor %}

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

Listing 48: manage_users.html

```
{% extends "bootstrap/base.html" %}

{% block title %}
View User Logs
{% endblock %}

{% block head%}
{% block scripts %} <!-- SCRIPT BLOCK START -->

{{super()}}

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></
    script> <!-- JQUERY - MUST BE FIRST -->
<script type="text/javascript" src="http://cdn.datatables.net/1.10.2/js/jquery.
    dataTables.min.js"></script> <!-- CDN DATATABLE SCRIPT -->

<script>
$(document).ready(function(){
$('#myTable').dataTable();
});
</script>

{% endblock %} <!-- SCRIPT BLOCK END -->

{% block styles %} <!-- STYLE BLOCK START -->
{{super()}}
<link rel="stylesheet" href="{{url_for('.static', filename='all.css')}}"> <!-- MY
     OWN CSS STYLING FILE -->
<link rel="stylesheet" href="http://cdn.datatables.net/1.10.2/css/jquery.
    dataTables.min.css"></style> <!-- CDN STYLING FILE -->
{% endblock %} <!-- STYLE BLOCK END -->

{% endblock %} <!-- BLOCK HEAD END -->


{% block content %}
<!-- TOP NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
<nav class="navbar navbar-inverse navbar-fixed-top">
<div class="container-fluid">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data
    -target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">FRACS Web</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<ul class="nav navbar-nav navbar-right">
<li><a href="/logout">Log Out</a></li>
</ul>
</div>
</div>
</nav>
<!-- TOP NAV END !!!!!!!!!!!!!!!!!!!! -->


<!-- SIDE NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
<div class="container-fluid">
<div class="row">
<div class="col-sm-3 col-md-2 sidebar">
```

```
<ul class="nav nav-sidebar">
<li><a href="/dashboard">Dashboard</a></li>
<li><a href="/all_users">FRACS Users</a></li>
<li class="active"><a href="#">View Access Logs <span class="sr-only">(current)</span> </a></li>
</ul>
<ul class="nav nav-sidebar">
<li ><a href="/my_profile">User Profile</a></li>
</ul>
</div>

<!-- SIDE NAV END !!!!!!!!!!!!!!!!!!!! -->

<!-- LOGS TABLE START !!!!!!!!!!!!!!!!!!!! -->

<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
<h2 class="sub-header">Access Logs</h2>
<div class="table-responsive">
<table id="myTable" class="table table-striped table-hover" border="1" cellpadding="5" cellspacing="5" style="border-color: black;">
<thead>
<tr class="tbl-headings">
<th>#</th>
<th>Username Entered</th>
<th>Actual User ID</th>
<th>Date & Time</th>
<th>Access Granted?</th>
</tr>
</thead>
<tbody >
{% for row in userslogs %}
<p> {{ row.user_id}} </p>
{% if row.is_successful == "FAIL" %}
<tr class="danger" >
{% else %}
<tr class="success">
{% endif %}
<td>{{ row.log_id}}</td>
<td>{{ row.username_entered }}</td>
<td>{{ row.attempted_user_id }} </td>
<td>{{ row.login_datetime }}</td>
<td>{{ row.is_successful }}</td>
</tr>
<!--   <tr style="background-color:#ff6666; border-color: #b30000">
<tr style="background-color:#99b3ff; border-color: #002699">

<td><a href="#" data-toggle="modal" data-target='#myModal' class="btn btn-success">?</a></td> -->

{% endfor %}
</tbody>
</table>
<!-- LOGS TABLE END !!!!!!!!!!!!!!!!!!!! -->

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

Listing 49: view_user_logs.html

```
{% extends "bootstrap/base.html" %}

{% block title %}
User Profile
{% endblock %}

{% block styles %}
{{super()}}
<link rel="stylesheet" href="{{url_for('.static', filename='all.css')}}">
<link rel="import" href="navigation.html">

{% endblock %}

{% block content %}
<!-- NAVIGATION BEGIN -->
<!-- TOP NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
<nav class="navbar navbar-inverse navbar-fixed-top">
<div class="container-fluid">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">FRACS Web</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<ul class="nav navbar-nav navbar-right">
<li><a href="/logout">Log Out</a></li>
</ul>
</div>
</div>
</nav>
<!-- TOP NAV END !!!!!!!!!!!!!!!!!!!! -->


<!-- SIDE NAV BEGIN !!!!!!!!!!!!!!!!!!!! -->
<div class="container-fluid">
<div class="row">
<div class="col-sm-3 col-md-2 sidebar">
<ul class="nav nav-sidebar">
<li><a href="/dashboard">Dashboard</a></li>
<li><a href="/all_users">FRACS Users</a></li>
<li><a href="/view_user_logs">View Access Logs </a></li>
</ul>
<ul class="nav nav-sidebar">
<li  class="active" ><a href="/my_profile">User Profile <span class="sr-only">(current)</span></a></li>
</ul>
</div>

<!-- SIDE NAV END !!!!!!!!!!!!!!!!!!!! -->
<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
<h2 class="sub-header">User Profile</h2>
<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
<form>
<div class="row">
<div class="col-md-2 pr-md-1">
<div class="form-group">
<label>User ID</label>
<input type="text" class="form-control" disabled="" value="{{ userprofile.user_id
```

```
    }}">
</div>
</div>
<div class="col-md-3 px-md-1">
<div class="form-group">
<label>Login Username</label>
<input type="text" class="form-control"disabled="" value="{{ userprofile.
    login_username }}">
</div>
</div>
<div class="col-md-7 pl-md-1">
<div class="form-group">
<label for="exampleInputEmail1">Face ID</label>
<input type="email" class="form-control"disabled="" value="{{ userprofile.face_id
    }}">
</div>
</div>
</div>
<div class="row">
<div class="col-md-6 pr-md-1">
<div class="form-group">
<label>First Name</label>
<input type="text" class="form-control"disabled="" placeholder="Company" value="
    {{userprofile.firstname}}">
</div>
</div>
<div class="col-md-6 pl-md-1">
<div class="form-group">
<label>Last Name</label>
<input type="text" class="form-control"disabled="" placeholder="Last Name" value=
    "{{userprofile.lastname}}">
</div>
</div>
</div>
<div class="row">
<div class="col-md-12">
<div class="form-group">
<label>About You</label>
<textarea rows="4" cols="80" class="form-control" disabled="" placeholder="Here
    can be your description" value="Mike">An admin/user</textarea>
</div>
</div>
</div>
</form>


</div>
</div>
</div>

{% endblock %}
```

Listing 50: my_profile.html

```
/*
* Base structure
*/

/* Move down content because we have a fixed navbar that is 50px tall */
body {
padding-top: 50px;
}


/*
* Global add-ons
*/

.sub-header {
padding-bottom: 10px;
border-bottom: 1px solid #eee;
}

/*
* Top navigation
* Hide default border to remove 1px line.
*/
.navbar-fixed-top {
border: 0;
}

/*
* Sidebar
*/

/* Hide for mobile, show later */
.sidebar {
display: none;
}
@media (min-width: 768px) {
.sidebar {
position: fixed;
top: 51px;
bottom: 0;
left: 0;
z-index: 1000;
display: block;
padding: 20px;
overflow-x: hidden;
overflow-y: auto; /* Scrollable contents if viewport is shorter than content. */
background-color: #f5f5f5;
border-right: 1px solid #eee;
}
}

/* Sidebar navigation */
.nav-sidebar {
margin-right: -21px; /* 20px padding + 1px border */
margin-bottom: 20px;
margin-left: -20px;
}
.nav-sidebar > li > a {
padding-right: 20px;
padding-left: 20px;
}
.nav-sidebar > .active > a,
.nav-sidebar > .active > a:hover,
```

```
.nav-sidebar > .active > a:focus {
color: #fff;
background-color: #428bca;
}


/*
* Main content
*/

.main {
padding: 20px;
background-color: #001a33;
color: #e6e6ff;
}
@media (min-width: 768px) {
.main {
padding-right: 40px;
padding-left: 40px;
}
}
.main .page-header {
margin-top: 0;
}

tr{
border-radius: 30px;
border: 2px solid black;
color: black;
}

.tbl-headings{
color: #e6e6ff;
}

/*
* Placeholder dashboard ideas
*/

.placeholders {
margin-bottom: 30px;
text-align: center;
}
.placeholders h4 {
margin-bottom: 0;
}
.placeholder {
margin-bottom: 20px;
}
.placeholder img {
display: inline-block;
border-radius: 50%;
}
```

Listing 51: all.css

```
body {
padding-top: 40px;
padding-bottom: 40px;
background-color: #eee;
}

.form-signin {
max-width: 330px;
padding: 15px;
margin: 0 auto;
}
.form-signin .form-signin-heading,
.form-signin .checkbox {
margin-bottom: 10px;
}
.form-signin .checkbox {
font-weight: normal;
}
.form-signin .form-control {
position: relative;
height: auto;
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
padding: 10px;
font-size: 16px;
}
.form-signin .form-control:focus {
z-index: 2;
}
.form-signin input[type="email"] {
margin-bottom: -1px;
border-bottom-right-radius: 0;
border-bottom-left-radius: 0;
}
.form-signin input[type="password"] {
margin-bottom: 10px;
border-top-left-radius: 0;
border-top-right-radius: 0;
}
```

Listing 52: all.css