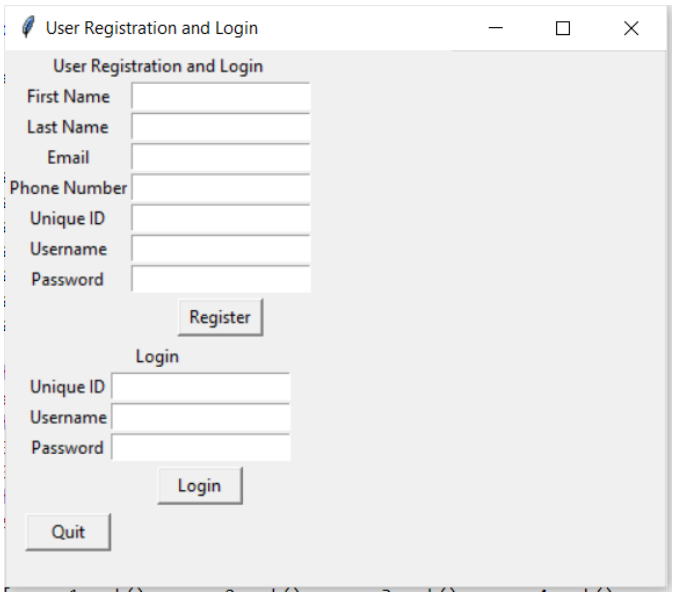
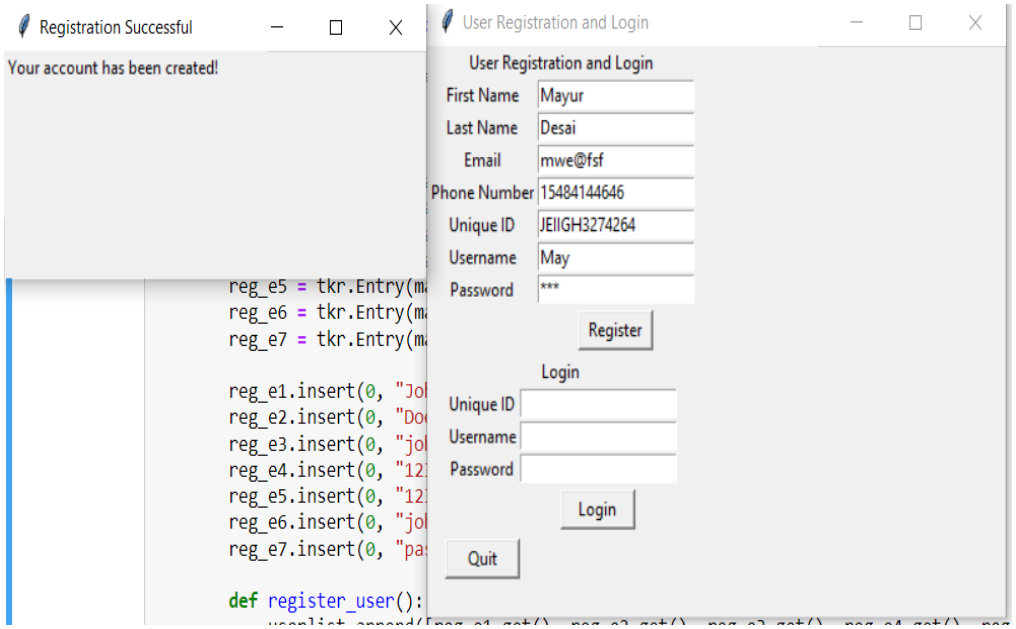
Detection Team - GlucoTech Project
BHI 504

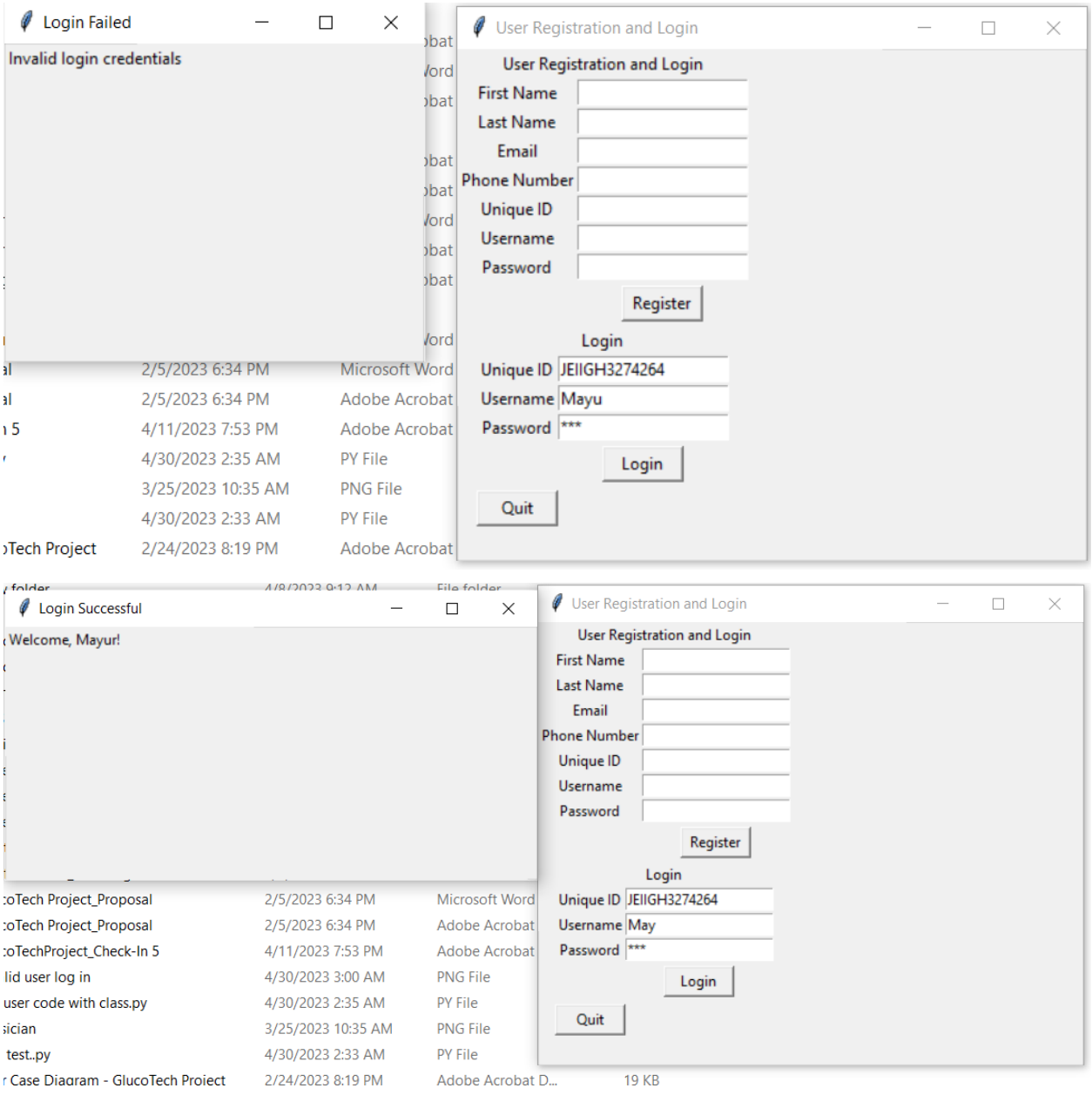
Team Member Desai - User Class

The goal is to User create(), log in() in this the verify user() is also a part of this code. Apart from this it also should save this created user for future log in. For creation of the user they have to provide first name, last name, email, phone number, unique id which will be provided, username and password, when click on register it will prompt a registration successful screen and notify that your account has been created and it will also save the registration. Then for the user login it doesn't require all of those things it only requires three things username, password and unique id, when clicked login it will prompt a welcome screen.





Detection Team - GlucoTech Project
BHI 504



Detection Team - GlucoTech Project
BHI 504

```
import tkinter as tkr

master = tkr.Tk()
userlist = []


"""Edit Window"""
master.title("User Registration and Login")
master.geometry("300x250")

tkr.Label(master, text="User Registration and Login").grid(row=0, columnspan=2)

tkr.Label(master, text="First Name").grid(row=1)
tkr.Label(master, text="Last Name").grid(row=2)
tkr.Label(master, text="Email").grid(row=3)
tkr.Label(master, text="Phone Number").grid(row=4)
tkr.Label(master, text="Unique ID").grid(row=5)
tkr.Label(master, text="Username").grid(row=6)
tkr.Label(master, text="Password").grid(row=7)

reg_e1 = tkr.Entry(master)
reg_e2 = tkr.Entry(master)
reg_e3 = tkr.Entry(master)
reg_e4 = tkr.Entry(master)
reg_e5 = tkr.Entry(master)
reg_e6 = tkr.Entry(master)
reg_e7 = tkr.Entry(master, show='*')

reg_e1.grid(row=1, column=1)
reg_e2.grid(row=2, column=1)
reg_e3.grid(row=3, column=1)
reg_e4.grid(row=4, column=1)
reg_e5.grid(row=5, column=1)
reg_e6.grid(row=6, column=1)
reg_e7.grid(row=7, column=1)
```

```
def register_user():
    userlist.append([reg_e1.get(), reg_e2.get(), reg_e3.get(), reg_e4.get(), reg_e5.get(), re
    print(userlist)
    reg_success = tkr.Tk()
    reg_success.title("Registration Successful")
    reg_success.geometry("200x100")
    tkr.Label(reg_success, text="Your account has been created!").grid(row=1)

def login_user():
    for user in userlist:
        if user[4] == login_e1.get() and user[5] == login_e2.get() and user[6] == login_e3.ge
            login_success = tkr.Tk()
            login_success.title("Login Successful")
            login_success.geometry("200x100")
            tkr.Label(login_success, text="Welcome, "+user[0]+"!").grid(row=1)
            return
    login_fail = tkr.Tk()
    login_fail.title("Login Failed")
    login_fail.geometry("200x100")
    tkr.Label(login_fail, text="Invalid login credentials").grid(row=1)

"""Registration Button"""
reg_button = tkr.Button(master, width=7, height=1, text="Register", command=register_user)
reg_button.grid(row=8, column=1, padx=3, pady=3)

"""Login Window"""
login_window = tkr.Frame(master)
login_window.grid(row=9, columnspan=2)

tkr.Label(login_window, text="Login").grid(row=0, columnspan=2)
tkr.Label(login_window, text="Unique ID").grid(row=1)
tkr.Label(login_window, text="Username").grid(row=2)
tkr.Label(login_window, text="Password").grid(row=3)

login_e1 = tkr.Entry(login_window)
login_e2 = tkr.Entry(login_window)
login_e3 = tkr.Entry(login_window, show='*')

login_e1.grid(row=1, column=1)
```

Detection Team - GlucoTech Project
BHI 504

```
login_e1.grid(row=1, column=1)
login_e2.grid(row=2, column=1)
login_e3.grid(row=3, column=1)

"""Login Button"""
login_button = tkr.Button(login_window, width=7, height=1, text="Login", command=login_user)
login_button.grid(row=4, column=1, padx=3, pady=3)

"""Quit Button"""
quit_button = tkr.Button(master, width=7, height=1, text="Quit", command=master.destroy)
quit_button.grid(row=10, column=0, padx=3, pady=3)

tkr.mainloop()

import unittest
import tkinter as tkr

class TestUserRegistrationAndLogin(unittest.TestCase):

    def test_register_user(self):
        master = tkr.Tk()
        userlist = []

        reg_e1 = tkr.Entry(master)
        reg_e2 = tkr.Entry(master)
        reg_e3 = tkr.Entry(master)
        reg_e4 = tkr.Entry(master)
        reg_e5 = tkr.Entry(master)
        reg_e6 = tkr.Entry(master)
        reg_e7 = tkr.Entry(master, show='*')

        reg_e1.insert(0, "John")
        reg_e2.insert(0, "Doe")
        reg_e3.insert(0, "johndoe@example.com")
        reg_e4.insert(0, "1234567890")
        reg_e5.insert(0, "123")
        reg_e6.insert(0, "johndoe")
        reg_e7.insert(0, "password123")

        def register_user():
```

```
        def register_user():
            userlist.append([reg_e1.get(), reg_e2.get(), reg_e3.get(), reg_e4.get(), reg_e5.g

        reg_button = tkr.Button(master, width=7, height=1, text="Register", command=register_
        reg_button.invoke()

        self.assertEqual(len(userlist), 1)
        self.assertEqual(userlist[0][0], "John")
        self.assertEqual(userlist[0][1], "Doe")
        self.assertEqual(userlist[0][2], "johndoe@example.com")
        self.assertEqual(userlist[0][3], "1234567890")
        self.assertEqual(userlist[0][4], "123")
        self.assertEqual(userlist[0][5], "johndoe")
        self.assertEqual(userlist[0][6], "password123")

    def test_login_user(self):
        master = tkr.Tk()
        userlist = [["John", "Doe", "johndoe@example.com", "1234567890", "123", "johndoe", "p

        login_e1 = tkr.Entry(master)
        login_e2 = tkr.Entry(master)
        login_e3 = tkr.Entry(master, show='*')

        login_e1.insert(0, "123")
        login_e2.insert(0, "johndoe")
        login_e3.insert(0, "password123")

        def login_user():
            for user in userlist:
                if user[4] == login_e1.get() and user[5] == login_e2.get() and user[6] == log
                    return True
            return False

        login_button = tkr.Button(master, width=7, height=1, text="Login", command=login_user
        self.assertTrue(login_button.invoke())

        login_e1.delete(0, tkr.END)
        login_e2.delete(0, tkr.END)
        login_e3.delete(0, tkr.END)
```

```
        login_e1.delete(0, tkr.END)
        login_e2.delete(0, tkr.END)
        login_e3.delete(0, tkr.END)

        login_e1.insert(0, "456")
        login_e2.insert(0, "janedoe")
        login_e3.insert(0, "password456")

        self.assertFalse(login_button.invoke())

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

```
In [1]: runfile('D:/BHI/504/Project/User code with test.py', wdir='D:/BHI/504/Project')
..
----------------------------------------------------------------------
Ran 2 tests in 0.248s

OK

In [2]:
```

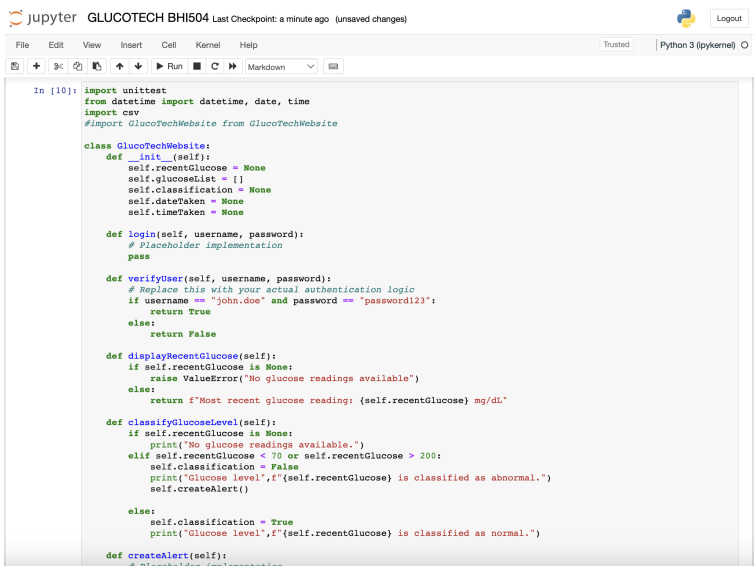
Detection Team - GlucoTech Project
BHI 504

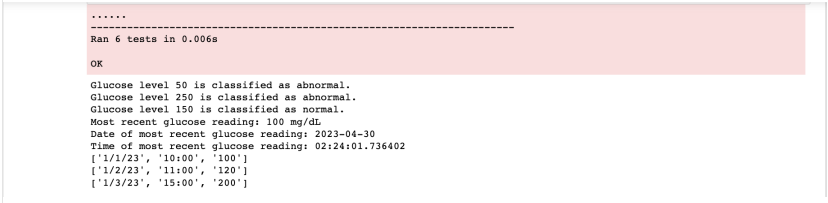
Team Member Adebiyi - GlucoTechWebsite Class

The goal is to create the Class GlucoTechWebsite, for managing glucose reading, for the GlucoTech Project.

This code will involve login(), verifyuser(), displayRecentGlucose(),classifyGlucoseLevel(), create(), displayClassification(),getDate(), getTime(), and writeFile(). The user will be able to login using their credentials and get verified as patient information is confidential. Then they will be able to view the recent blood glucose that was taken, which will be displayed. In order to classify the blood glucose if it is normal or abnormal, the glucose level will be classified, then displayed with an alert being sent. The date and time will be generated, consecutively, as the patient manually records their blood glucose with writeFile() method which is then converted to a csv file, as the most recent blood glucose reading.

It also includes a test class called "TestGlucoTechWebsite" that tests some of the methods in the "GlucoTechWebsite" class using the Python "unittest" library.





Detection Team - GlucoTech Project
BHI 504

Team Member Gaitos - GlucoTechServer Class and AlertSys Class

The GlucoTechServer Class will serve as a repository of data information obtained from the GlucoseLevelMeasurement and GlucoTechWebsite classes. This class comprises different methods and attributes. In this project, only a few methods and attributes will be utilized and displayed. The methods are readFile(), findAvg(), and createGraph(). The attributes are glucoseList, avgMonth, and classification. The images below will illustrate how the class works, how the user can utilize the class, and how the unit test is applied.

The AlertSys Class will send reminders via text message or email to registered individuals whenever the blood glucose level is abnormal.This class comprises three methods: sendText() and sendEmail().

GlucoTechServer Class
Code:

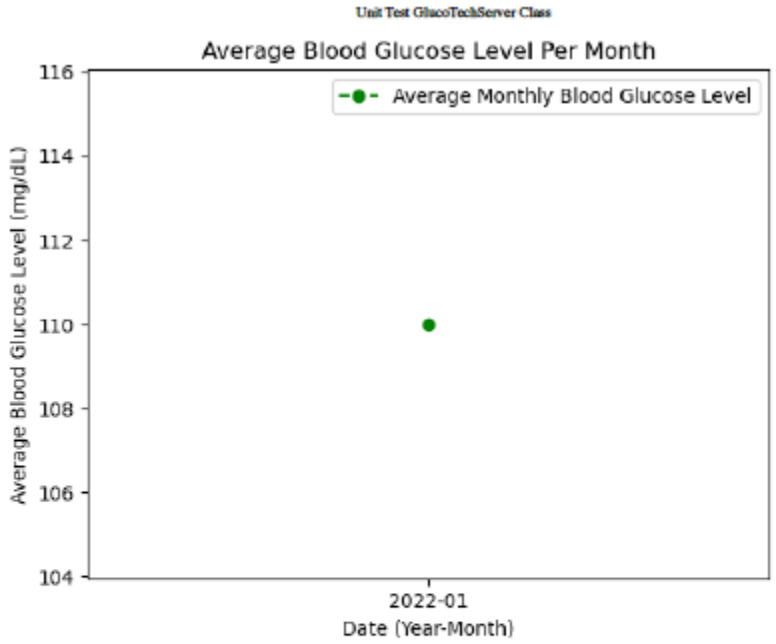
```
import pandas as pd
import matplotlib.pyplot as plt

class GlucoTechServer:
    glucoseList=[]
    avgMonth=0
    classification=False

    def __init__(self):
        self.glucoseList=[]
        print(self)

    def readFile(self, filename):
        df = pd.read_csv(filename)
        self.glucoseList.append(df)
        return df

    def findAvg(self, filename):
        df = pd.read_csv(filename)
        self.glucoseList.append(df)
        if len(self.glucoseList) == 0:
            print("Error: glucoseList is empty")
            return

        # Get most recent data
        data = self.glucoseList[-1]

        # Convert date column to datetime format and set as index
        data['Date'] = pd.to_datetime(data['Date'])
        data.set_index('Date', inplace=True)

        # Resample data to monthly average
        monthly_data = data.resample('M').mean()

        # Add Month column to the resampled data
        monthly_data['Month'] = monthly_data.index.strftime('%Y-%m')

        # Display the monthly data in a table
        return monthly_data[['Month', 'Blood Glucose Level']].rename(columns={'

    def createGraph(self, filename):
        df = pd.read_csv(filename)
        self.glucoseList.append(df)
        if len(self.glucoseList) == 0:
            print("Error: glucoseList is empty")
            return

        data = self.glucoseList[-1]  # get the most recent data
        data['Date'] = pd.to_datetime(data['Date']) # convert date to datetime
        data.set_index('Date', inplace=True) # set Date column as index

        # Resample data to monthly average and plot
        monthly_data = data.resample('M').mean() # resample data to monthly ave
        plt.plot(monthly_data.index.strftime('%Y-%m'), monthly_data['Blood Gluc

        # Set plot labels and titleserver
        plt.xlabel('Date (Year-Month)')
        plt.ylabel('Average Blood Glucose Level (mg/dL)')
        plt.title('Average Blood Glucose Level Per Month')
        plt.legend()
        plt.show()
```

Detection Team - GlucoTech Project
BHI 504

Unit Testing:

```
import unittest
import pandas as pd
import matplotlib.pyplot as plt

class TestGlucoTechServer(unittest.TestCase):

    def setUp(self):
        self.server = GlucoTechServer()
        self.test_file = 'test_data.csv'
        self.test_data = pd.DataFrame({'Date': ['2022-01-01', '2022-01-02', '20

    def test_readFile(self):
        df = self.server.readFile(self.test_file)
        self.assertTrue(df.equals(self.test_data))

    def test_findAvg(self):
        self.server.glucoseList = [self.test_data]
        expected_monthly_data = pd.DataFrame({'Month': ['2022-01'], 'Average Bl
        monthly_data = self.server.findAvg(self.test_file)
        self.assertTrue(monthly_data.equals(expected_monthly_data))

    def test_createGraph(self):
        self.server.glucoseList = [self.test_data]
        self.server.createGraph(self.test_file)
        # Check that the graph was created by ensuring that a figure exists
        self.assertTrue(plt.gcf())

if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'],exit=False)
```

```
<__main__.GlucoTechServer object at 0x116cc8c40>
```




```
...
----------------------------------------------------------------------
Ran 3 tests in 0.347s

OK
<__main__.GlucoTechServer object at 0x116cc8eb0>
<__main__.GlucoTechServer object at 0x116dbe020>
<Figure size 640x480 with 0 Axes>
```

Detection Team - GlucoTech Project
BHI 504

Executing GlucoTechServer Class:

```
server=GlucoTechServer()
```

```
<__main__.GlucoTechServer object at 0x108d63e50>
```

```
server.readFile('filename.csv')
```

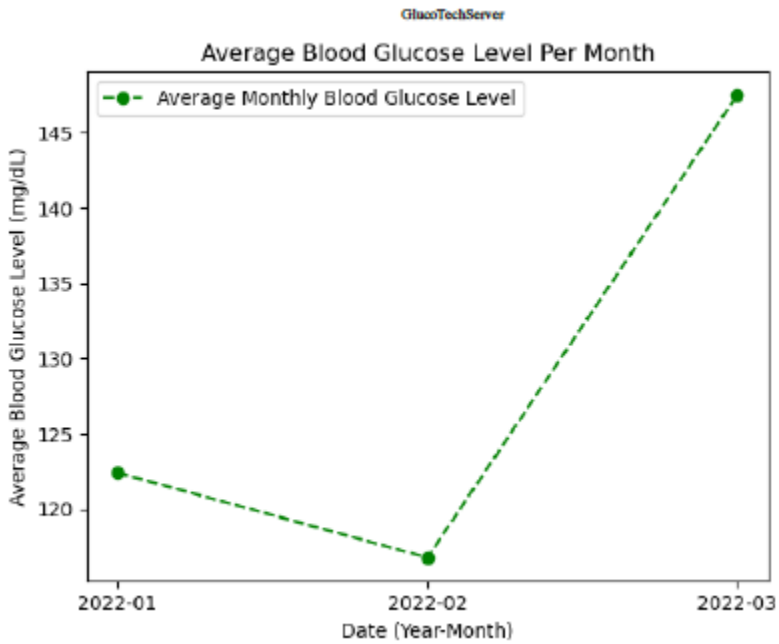
	Date	Blood Glucose Level
0	1/1/22	110
1	1/2/22	120
2	1/3/22	130
3	1/4/22	140
4	1/5/22	150
...	...	...
68	3/10/22	135
69	3/11/22	130
70	3/12/22	125
71	3/13/22	120
72	3/14/22	115

73 rows × 2 columns

```
server.findAvg('filename.csv')
```

	Month	Average Blood Glucose Level
0	2022-01	122.419355
1	2022-02	116.785714
2	2022-03	147.500000

```
server.createGraph('filename.csv')
```



Detection Team - GlucoTech Project
BHI 504

AlertSys Class
Code:

```python
class AlertSys:
    def __init__(self, phone_number, email_address):
        self.phone_number = phone_number
        self.email_address = email_address

    def sendText(self, message):
        # Use a third-party service to send a text message to the specified phone number
        print(f"Sending text message to {self.phone_number}: {message}")

    def sendEmail(self, subject, message):
        # Use a third-party email service to send an email to the specified email address
        print(f"Sending email to {self.email_address} with subject '{subject}': {message}")

    def check_blood_glucose(self, glucose_level):
        if glucose_level < 70 or glucose_level > 180:
            message = f"Blood glucose level is abnormal: {glucose_level}"
            self.sendText(message)
            self.sendEmail("Alert: Abnormal Blood Glucose Level", message)
```

Unit Testing:

```python
import unittest
from unittest.mock import patch


class TestAlertSys(unittest.TestCase):
    def setUp(self):
        # Create an AlertSys object with a mock phone number and email address
        self.alert_system = AlertSys("+15551234567", "johndoe@example.com")

    @patch.object(AlertSys, 'sendText')
    @patch.object(AlertSys, 'sendEmail')
    def test_check_blood_glucose_normal(self, mock_send_email, mock_send_text_message):
        # Check a normal blood glucose level
        glucose_level = 100
        self.alert_system.check_blood_glucose(glucose_level)

        # Ensure that no alerts were sent
        mock_send_text_message.assert_not_called()
        mock_send_email.assert_not_called()

    @patch.object(AlertSys, 'sendText')
    @patch.object(AlertSys, 'sendEmail')
    def test_check_blood_glucose_low(self, mock_send_email, mock_send_text_message):
        # Check a low blood glucose level
        glucose_level = 50
        self.alert_system.check_blood_glucose(glucose_level)

        # Ensure that a text message and email alert were sent
        mock_send_text_message.assert_called_once_with('Blood glucose level is abnormal: 50')
        mock_send_email.assert_called_once_with('Alert: Abnormal Blood Glucose Level', 'Blood glucose level is abnormal: 50')

    @patch.object(AlertSys, 'sendText')
    @patch.object(AlertSys, 'sendEmail')
    def test_check_blood_glucose_high(self, mock_send_email, mock_send_text_message):
        # Check a high blood glucose level
        glucose_level = 200
        self.alert_system.check_blood_glucose(glucose_level)

        # Ensure that a text message and email alert were sent
        mock_send_text_message.assert_called_once_with('Blood glucose level is abnormal: 200')
        mock_send_email.assert_called_once_with('Alert: Abnormal Blood Glucose Level', 'Blood glucose level is abnormal: 200')
```

Detection Team - GlucoTech Project
BHI 504

```
if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'],exit=False)
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.010s

OK
```

Executing AlertSys Class:

```
# Create an AlertSys object with a phone number and email address
alert_system = AlertSys("+15551234567", "johndoe@example.com")

# Check a blood glucose level
glucose_level = 200
alert_system.check_blood_glucose(glucose_level)
```

```
Sending text message to +15551234567: Blood glucose level is abnormal: 200
Sending email to johndoe@example.com with subject 'Alert: Abnormal Blood Glucose Level': Blood glucose level is abnor
mal: 200
```

Detection Team - GlucoTech Project
BHI 504

Team Member Tota
I will be developing the Alert System for this project. This class will receive and send alerts to users when abnormal blood glucose levels are recorded, which will help to monitor and be reminded of their blood glucose levels.

Code:def init(self, user):

self.user = user # user object

self.alerts = [] # list of alerts

def receive_alert(self, alert):

self.alerts.append(alert)

self.send_alert(alert)

def send_alert(self, alert):

if alert.level == "high":

message = "Your blood glucose levels are too high. Please check your levels and take necessary action."

elif alert.level == "low":

message = "Your blood glucose levels are too low. Please check your levels and take necessary action."

else:

message = "Your blood glucose levels are abnormal. Please check your levels and take necessary action."

send message to user via email or SMS

code for sending message goes here

def view_alerts(self):

print("Here are your alerts:")

for alert in self.alerts:

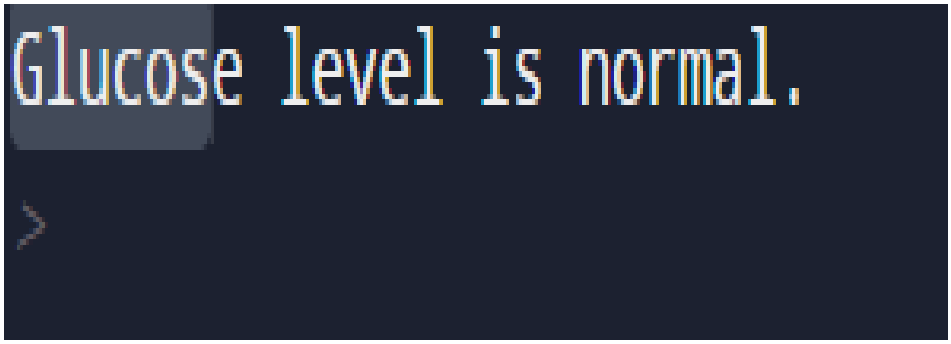
print(alert)

Testing:

```
def init(self, user):
self.user = user    # user object
self.alerts = []    # list of alerts
def receive_alert(self, alert):
    self.alerts.append(alert)
    self.send_alert(alert)

def send_alert(self, alert):
    if alert.level == "high":
        message = "Your blood glucose levels are too high. Please check your
            levels and take necessary action."
    elif alert.level == "low":
        message = "Your blood glucose levels are too low. Please check your levels
            and take necessary action."
    else:
        message = "Your blood glucose levels are abnormal. Please check your
            levels and take necessary action."

    # send message to user via email or SMS
    # code for sending message goes here

def view_alerts(self):
    print("Here are your alerts:")
    for alert in self.alerts:
        print(alert)
```

```
glucose_level = 110

if glucose_level == 110:
    print("ALERT: Glucose level is at a critical level of 200!")
else:
    print("Glucose level is normal.")
```

Detection Team - GlucoTech Project
BHI 504




```
glucose_level = 240

if glucose_level == 240:
    print("ALERT: Glucose level is at a critical level of 200!")
else:
    print("Glucose level is normal.")
```

- If the glucose level is observed to me below 120 then it indicates as normal.
- If the glucose level is observed to me above120 then it indicates as abnormal. And gives alert as ;
- Alert: Your glucose level has been measured and is outside of the target range. Please check and adjust your insulin or sugar intake as needed.

Detection Team - GlucoTech Project
BHI 504

PROJECT STATUS
USER STORIES

The interviewed users for this GlucoTech Project consist of a patient, a physician, a nurse, and a family relative/caretaker. The priority is stated below, and the estimates below are in a number of days.

Story #1 - **Yes**
As a diabetic patient, I want to be able to monitor my blood glucose levels accurately so that I can effectively and actively monitor my condition to prevent its complications, which leads to having a sense of control and engagement in my own health.

Priority 2
Estimate 2

Story #2 - **Yes**
As a diabetic patient, I want to be able to receive real-time feedback and alerts based on my blood glucose monitoring so that I can effectively utilize my anti-diabetic medications.

Priority 3
Estimate 3

Story #3 - **Yes**
As a diabetic patient, I want to be able to store my blood glucose monitoring securely in the cloud storage of GlucoTech and to be able to access them easily through their website so that I will limit having errors writing them manually and not have a hard time remembering them.

Priority 1
Estimate 3

Story #4 - **Yes**
As a diabetic patient, I want to be able to share my blood glucose monitoring with my clinicians and permitted relatives/family members so that whenever I encounter symptoms (e.g., fatigue, weakness, unconsciousness) from extremely high or low blood glucose levels, they will be able to act accordingly (e.g., provide medication if extremely high blood glucose level or provide sugar if extremely low blood glucose level).

Priority 4
Estimate 4

Story #5 - **Yes**
As a permitted relative/family member, I want to be able to access and receive blood glucose monitoring with alerts and notifications of my debilitated family members suffering from diabetes so that I can provide them with proper care (e.g., intake of medication, diet, etc.).

Detection Team - GlucoTech Project
BHI 504

Priority 3
Estimate 2

Story #6 - **Yes**
As a physician, I want to be able to track and monitor my patient's blood glucose levels and to receive alerts and notifications about their abnormal blood glucose levels remotely so that I can adjust and personalize the treatment and provide insights and recommendations (e.g., tests and referrals).

Priority 5
Estimate 2

Story #7 - **Yes**
As a physician-researcher, I want to be able to access the anonymized data information with consent about blood glucose monitoring so that I can understand and provide innovative works regarding diabetes.

Priority 8
Estimate 1

Story #8 - **Yes**
As a nurse, I want to be able to access my patient's blood glucose monitoring so that I can help them be reminded of the proper fingerstick method, the intake of their medications, and their diet.

Priority 6
Estimate 2

Story #9 - **Yes**
As a dietician, I want to be able to access my patient's blood glucose monitoring so that I can provide an individualized dietary recommendation and be able to collaborate with other clinicians regarding dietary needs.

Priority 7
Estimate 2

Detection Team - GlucoTech Project
BHI 504

BUSINESS RULES

The business rules pertaining to the GlucoTech Project are the following:

BR 1. The clinicians with access to the patient's stored blood glucose level monitoring in the website and the cloud storage are physicians, researchers, nurses, and dieticians. - **Yes**

BR 2. The non-clinicians who have access to the patient's stored blood glucose level monitoring in the website and the cloud storage are the patient and permitted relatives/family members. - **Yes**

BR 3. Individuals with access to the patient's data information have been provided with education on how to navigate the application and cloud through specific usernames and passwords. - **Yes**

BR 4. Physicians alone can modify the patient's treatment regimen, recommend new diagnostic tests, and advise/refer consultations with other specialists to prevent common diabetes mellitus complications. - **No (Did not code method to write a file)**

BR 5. Physicians and dieticians alone can recommend individualized diets to patients. - **No (Did not code method to write a file)**

BR 6. Nurses report to the physician and refer the patients with abnormal blood glucose level monitoring. - **No (Did not code method that can send communication among clinicians)**

BR 7. The system must comply with all relevant data privacy and security regulations and must maintain high standards of data protection. - **Yes**

BR 8. The GlucoTech must provide customer support to address any technical issues or concerns raised by users. - **Yes**

BR 9. An individual is considered diabetic if [please refer to the table below (Mayo Clinic, 2023)]: - **Yes**

Test	Result
HbA1C	Diabetes - more than 6.5%
	Prediabetes - 5.7% - 6.4%
	Normal - less than 5.7%
Random blood sugar	Diabetes - more than 200 mg/dL
	Prediabetes - ***
	Normal - less than 200 mg/dL
Fasting blood sugar	Diabetes - more than 126 mg/dL
	Prediabetes - 100 mg/dL - 125 mg/dL
	Normal - less than 100 mg/dL
Glucose tolerance	Diabetes - more than 200 mg/dL

Detection Team - GlucoTech Project
BHI 504

Prediabetes - 140 mg/dL - 199 mg/dL
Normal - less than 140 mg/dL

BR 10. The GlucoTech will provide alert messages to the individuals in BR 1 and BR 2 sections when the patient's blood glucose level is more than 100 mg/dL or less than 50 mg/dL. - **Yes**