CSCE 5214.004 SDAI

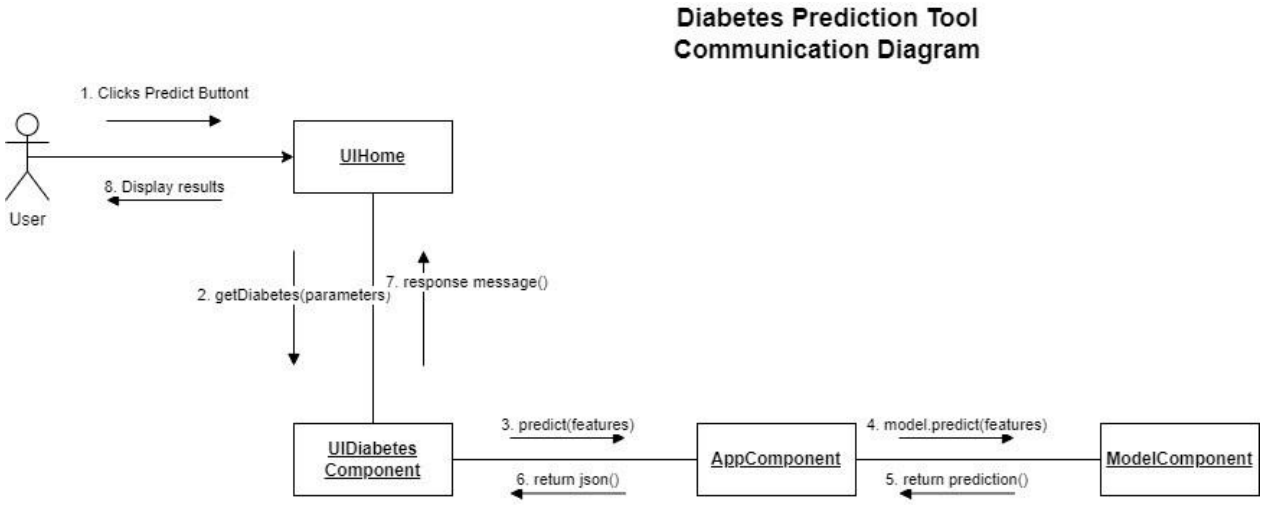
Group 9: DIABETES PREDICTION TOOL

Phase 4: Develop a User Interface

Aditya Vadrevu – 11601517
Suhas Siddarajgari Tellatakula – 11626111
Srikar Reddy Gunna - 11594012
Sai Praneeth Reddy Avula - 11582402
Indu Shashi Guda - 11618418

1. Task 1: Communication Diagram:

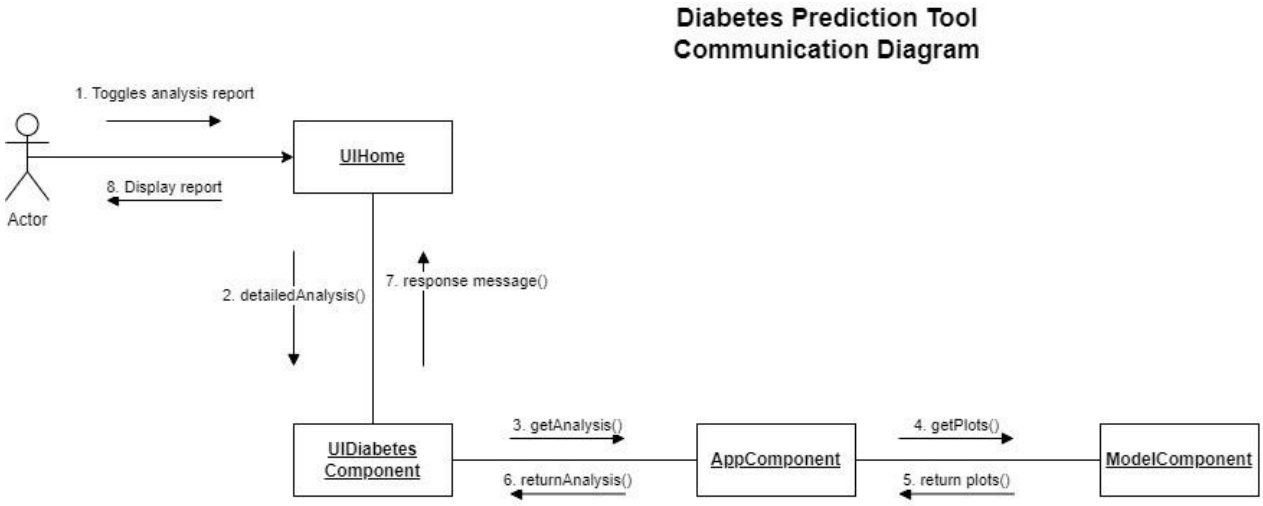
Use Case 1:



Above communication diagrams depicts the Predict Use Case. When the user provides his values and clicks on the predict button, the html component in UI Home container will pass the entered values as arguments through getDiabetes() method. This method which is defined in UI Diabetes component whenever called , will initiate http.get along with the received parameters namely glucose level, Insulin, BMI and Age. In the backend code of the tool in App.py component, function called predict will initiate whenever get method is called. It collects these 4 parameters and convert them using minmaxscalar and pass it to the backend finalized model which is already dumped and saved using joblib

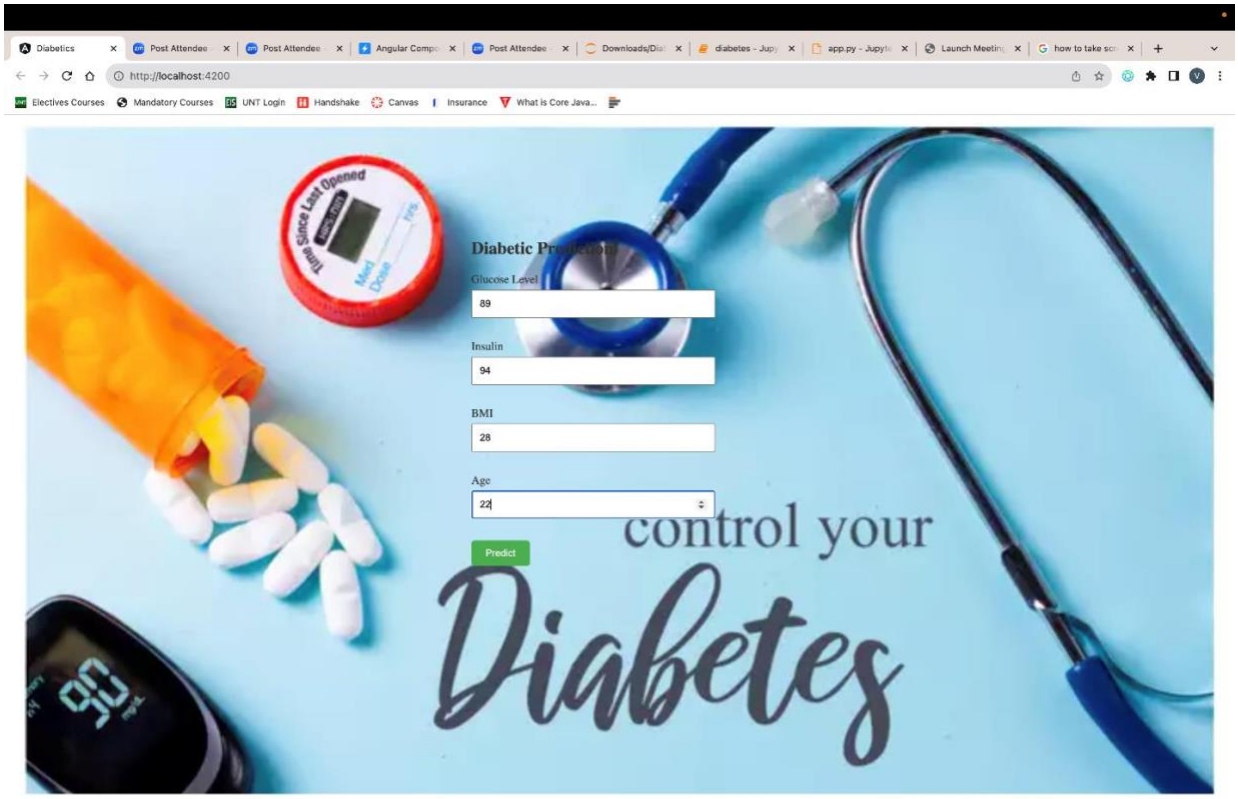
library. So , this model will take these values and predicts the output for these specific values. If the output is 0, “You don’t have Diabetes” message is assigned. Otherwise, “You have Diabetes. Please consult a doctor” message is assigned . This assigned message is returned in the form of json to the UI diabetes component which will then display the response message to user through UI Home component html web page.

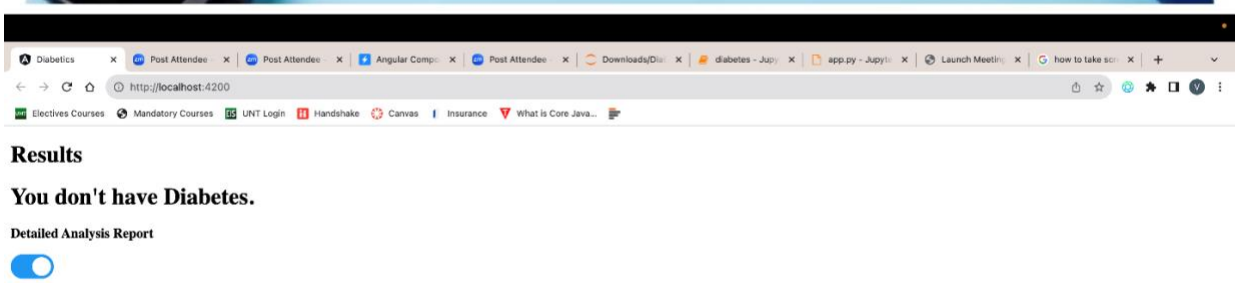
Use Case 2:

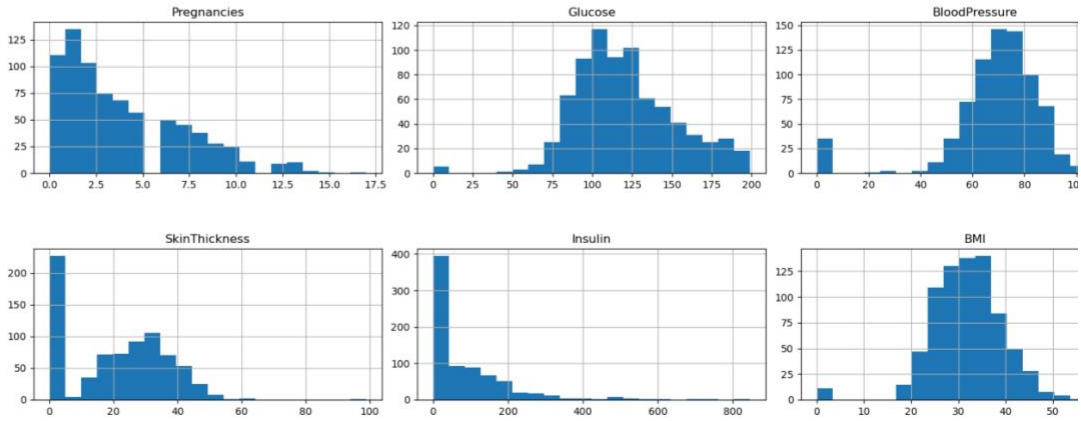


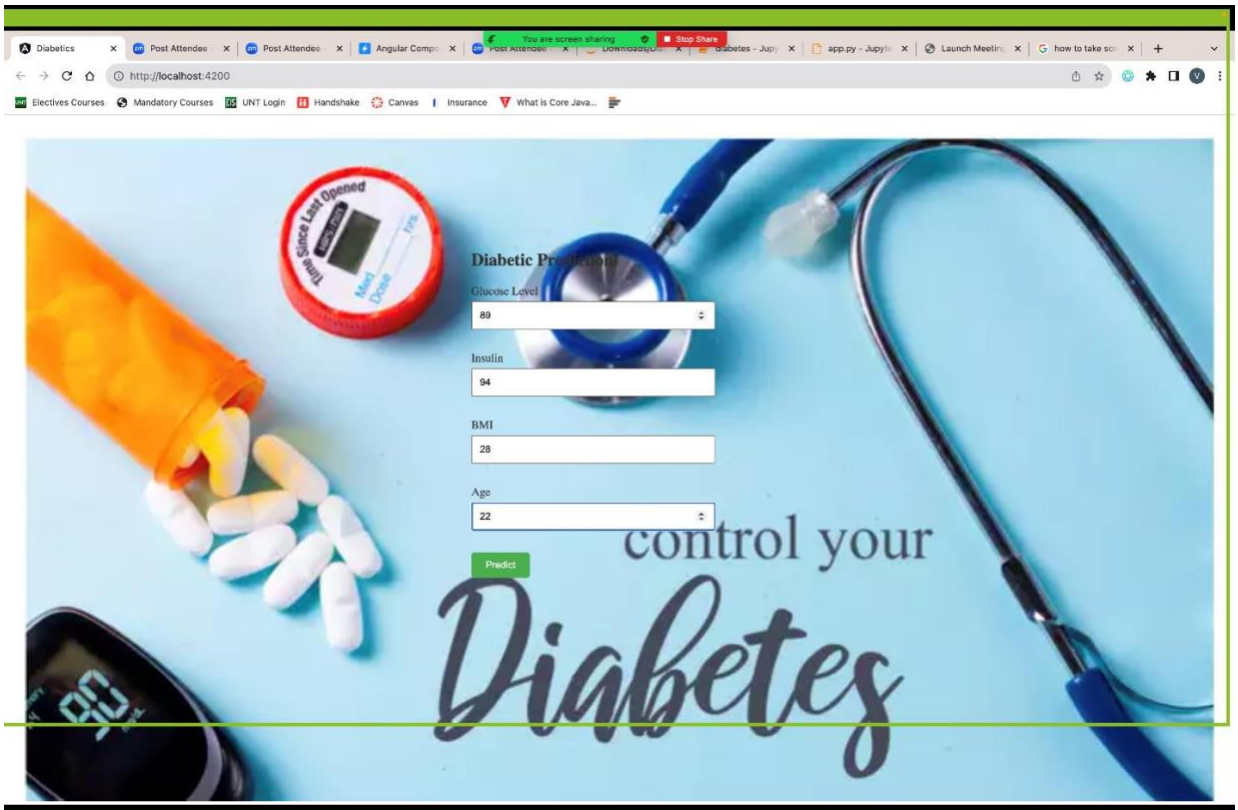
Above communication diagrams depicts the Detailed Analysis Report Use Case. Once the user can see his/her results on the results page, he can toggle the detailed analysis report which will initaiate the detailedAnalysis() method in the html component - UI Home container which will in turn call getAnalysis() in UIDiabetes Component. This method which is defined in UI Diabetes component whenever called , will initiate http.get. In the backend code of the tool in App.py component, function called getplots() will initiate whenever get method is called. Then the model will process and return the plots to app and UI Diabetes component which will then display the response message to user through UI Home component html web page.

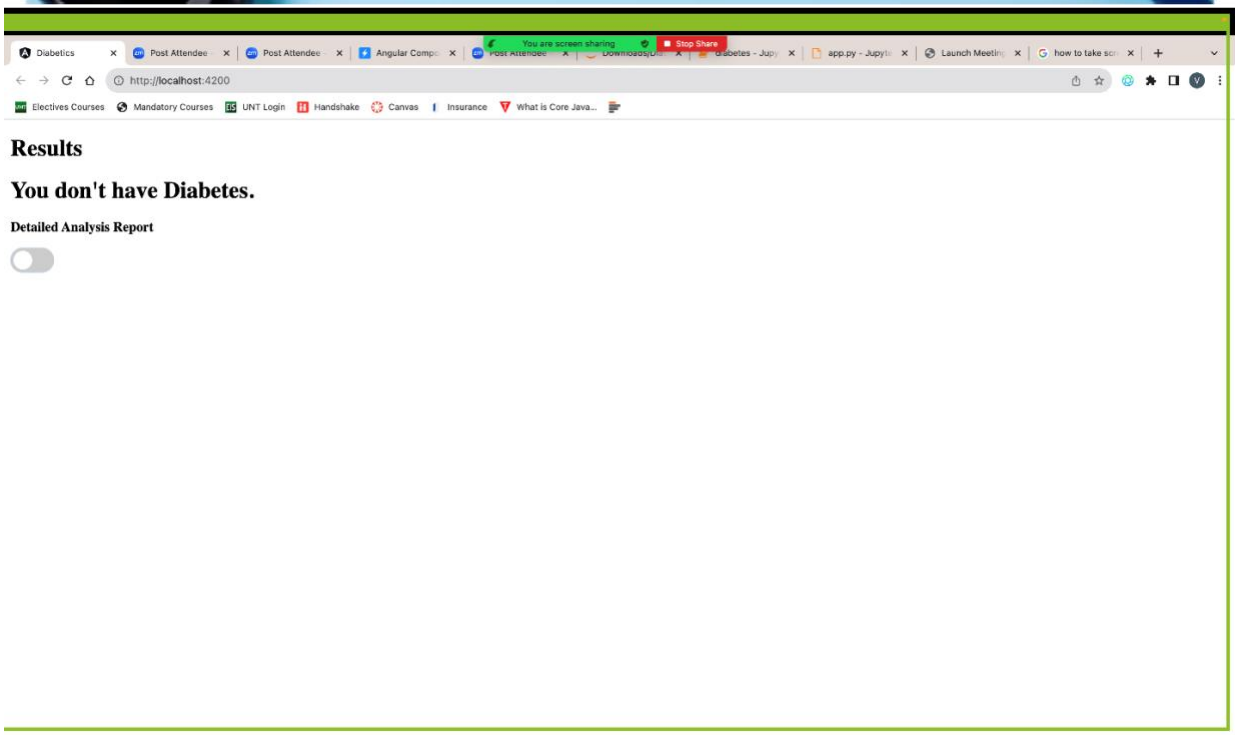
2. Task 2: UI Implementation

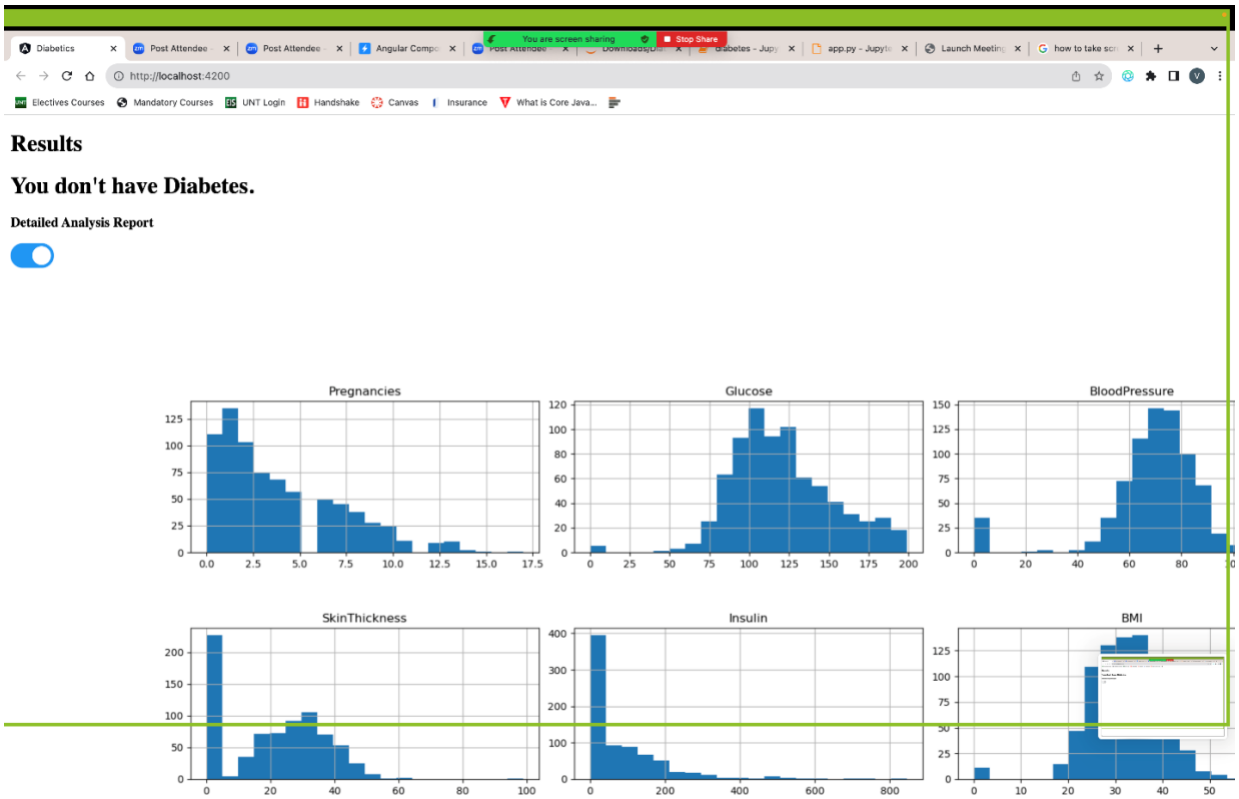


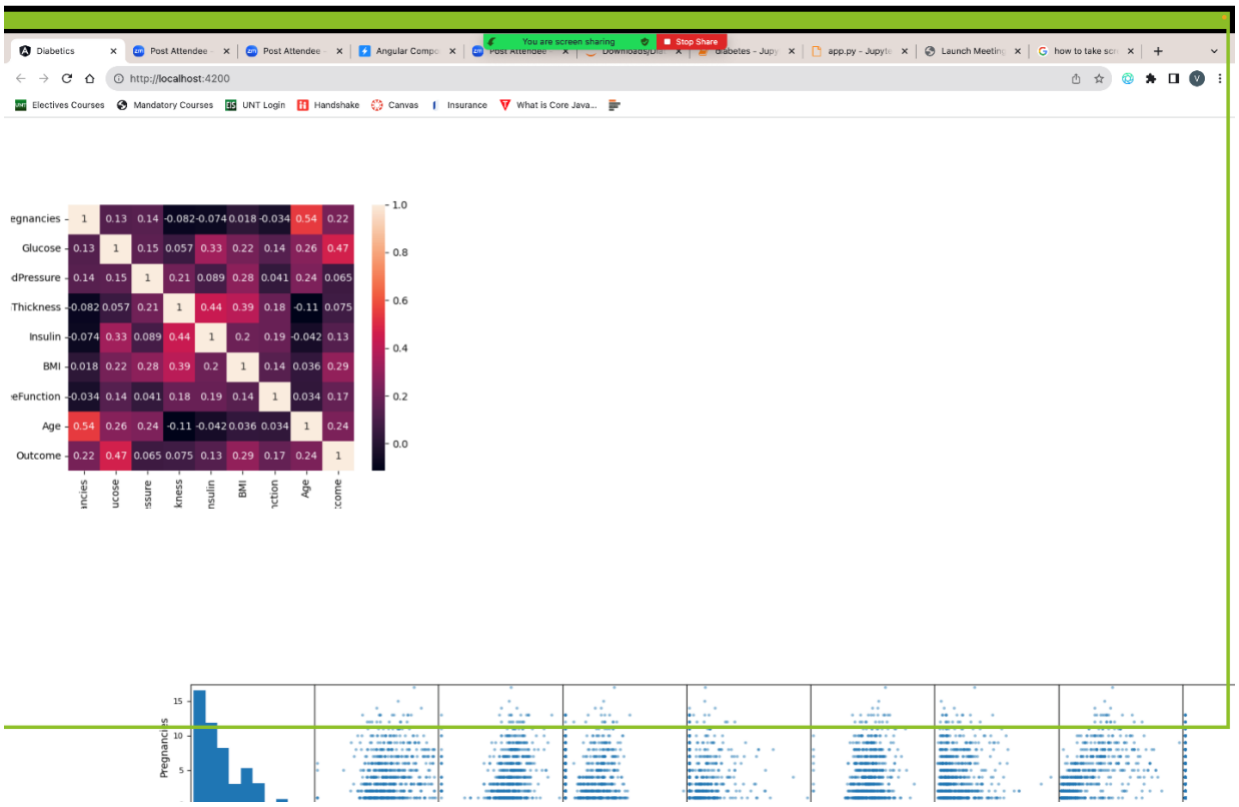


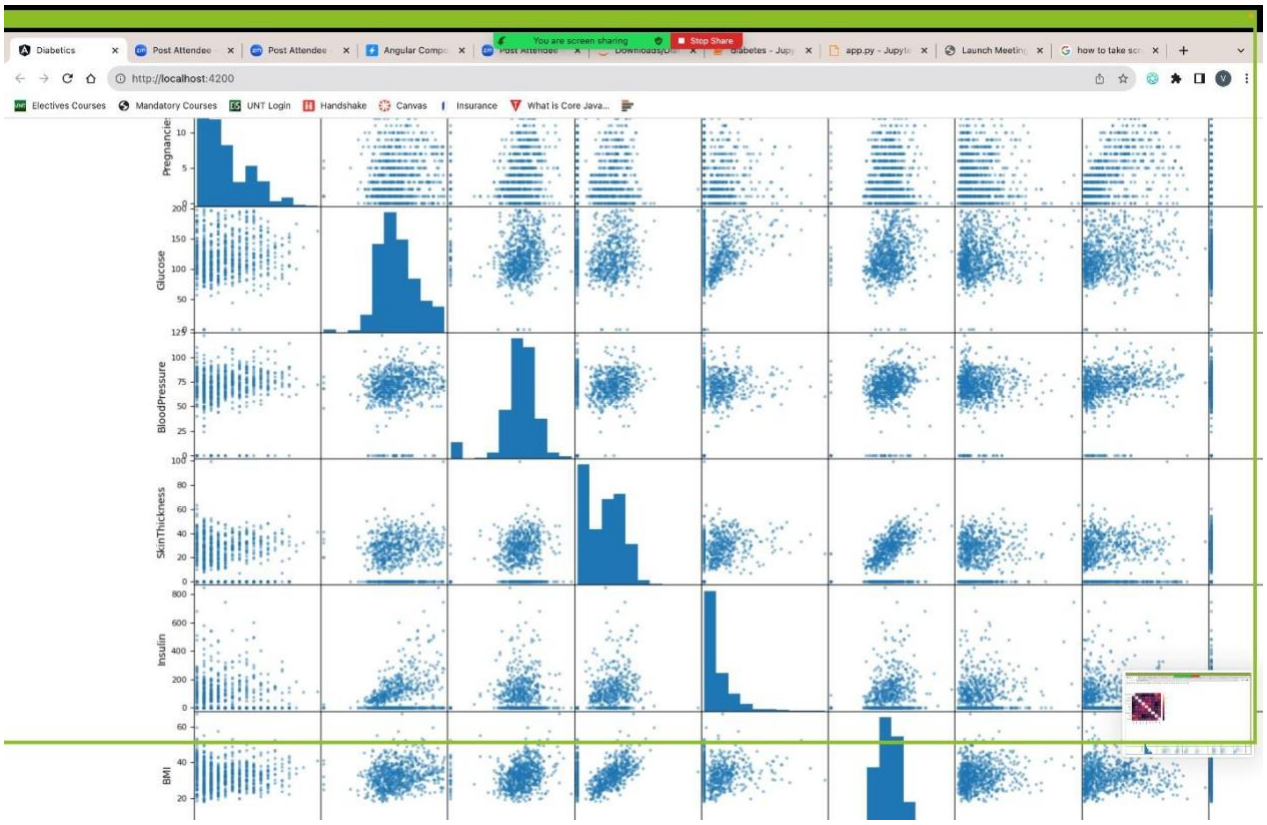



Diabetics
Post Attendee
Post Attendee
Angular Compo
You are screen sharing
Stop Share
app.py - Jupyte
Launch Meeting
how to take scr
http://localhost:4200
Electives Courses
Mandatory Courses
UNT Login
Handshake
Canvas
Insurance
What is Core Java...
Time Since Last Opened
Med Dose
Diabetic Predictor
Glucose Level
89
Insulin
94
BMI
28
Age
22
Predict
control your
Diabetes


Diabetics
Post Attendee
Post Attendee
Angular Compo
You are screen sharing
Stop Share
app.py - Jupyte
Launch Meeting
how to take scr
http://localhost:4200
Electives Courses
Mandatory Courses
UNT Login
Handshake
Canvas
Insurance
What is Core Java...
Results
You don't have Diabetes.
Detailed Analysis Report

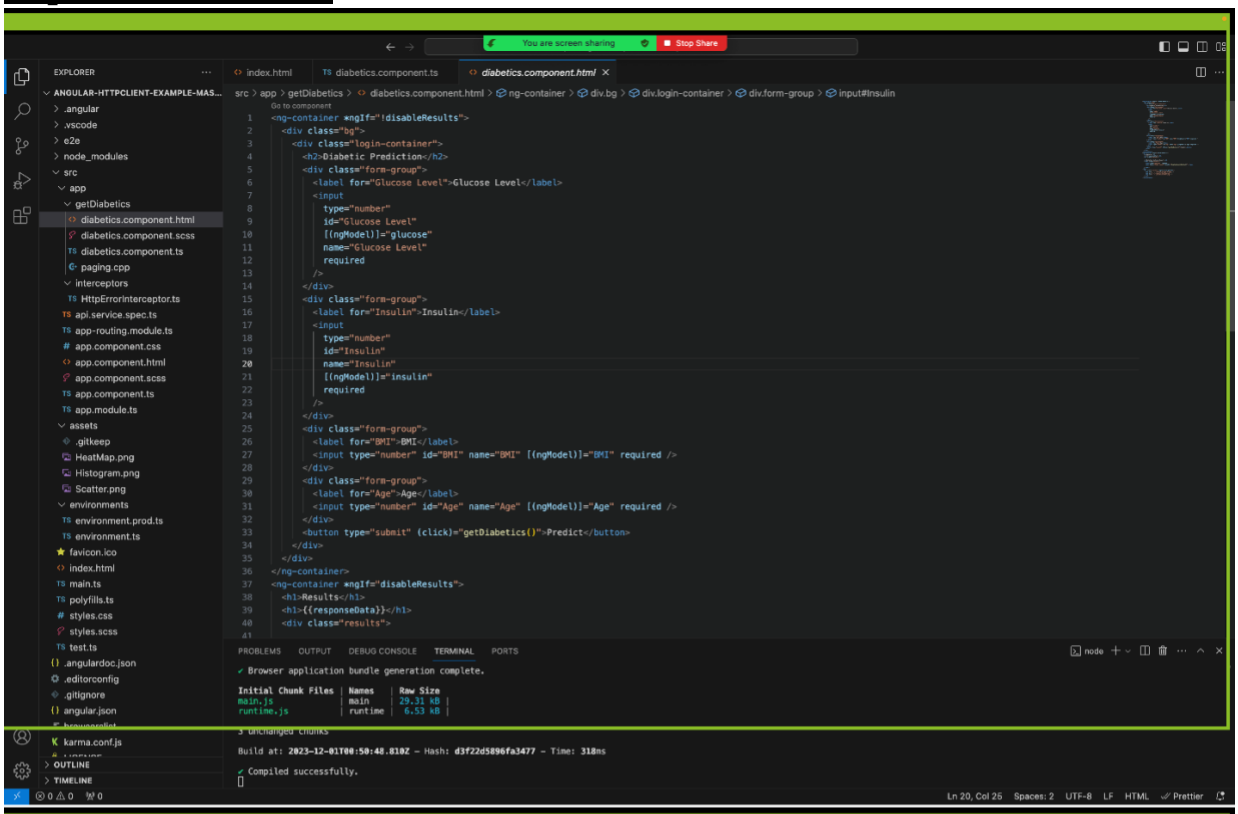

Diabetics
Post Attendee
Post Attendee
Angular Compo
You are screen sharing
Stop Share
app.py - Jupyte
Launch Meeting
how to take scr
http://localhost:4200
Electives Courses
Mandatory Courses
UNT Login
Handshake
Canvas
Insurance
What is Core Java...
Results
You don't have Diabetes.
Detailed Analysis Report
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI

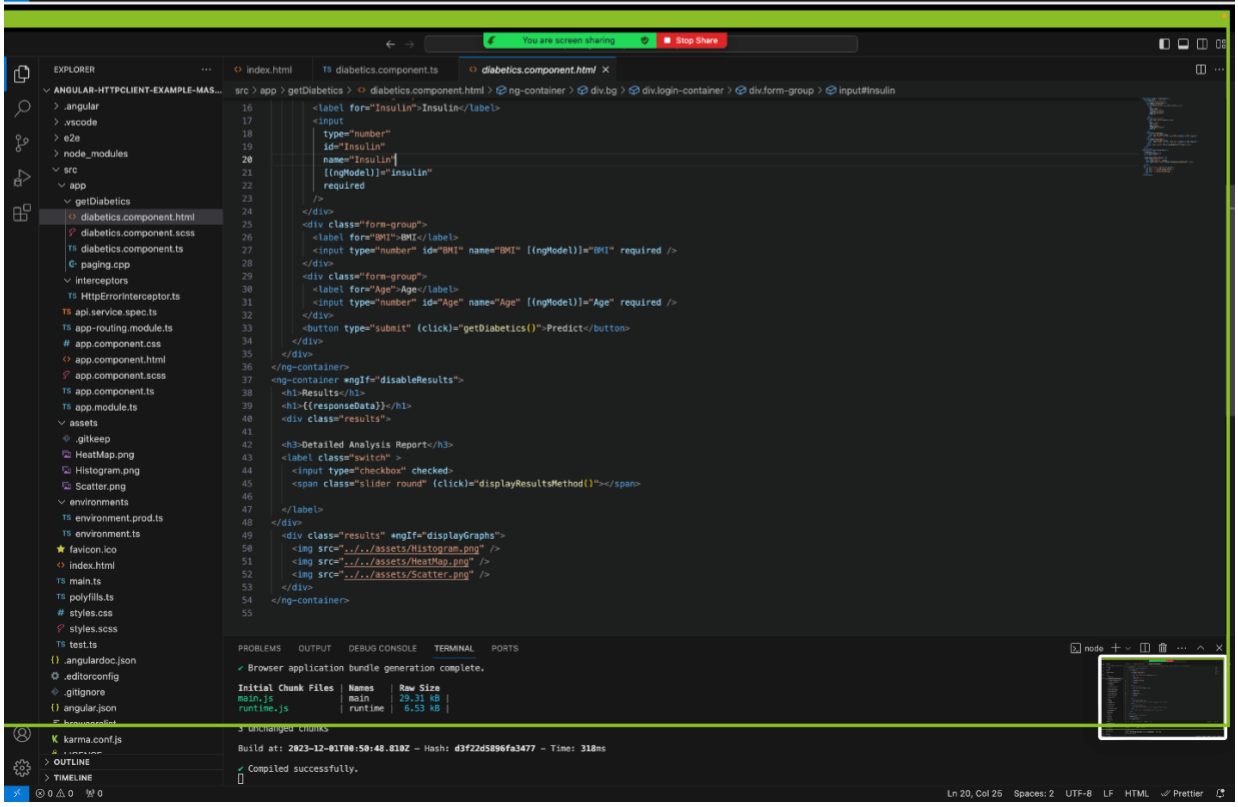

Diabetics
Post Attendee
Post Attendee
Angular Compo
You are screen sharing
Stop Share
app.py - Jupyte
Launch Meeting
how to take scr
http://localhost:4200
Electives Courses
Mandatory Courses
UNT Login
Handshake
Canvas
Insurance
What is Core Java...
egnancies
Glucose
dPressure
Thickness
Insulin
BMI
eFunction
Age
Outcome
Pregnancies



Above figures show the User Interface implemented additional to the existing Diabetes Prediction tool developed. In the earlier version, it has a very basic and minimal UI which used to take values from user and return yes/no result as pop-up. So, we have enhanced this system by developing a new UI using Angular, HTML, CSS. When the user opens the tool, he will land in the home page where he/she can enter the values like Glucose level, Insulin, BMI and Age with a relevant background image as shown above. Also, there is a predict button. Then we have a results page which shows the response message. Also, we added a detailed analysis report which shows the data correlation amongst features and the output in terms of scatter plots, heatmaps, histograms. This can be viewed using a toggle button as shown above.

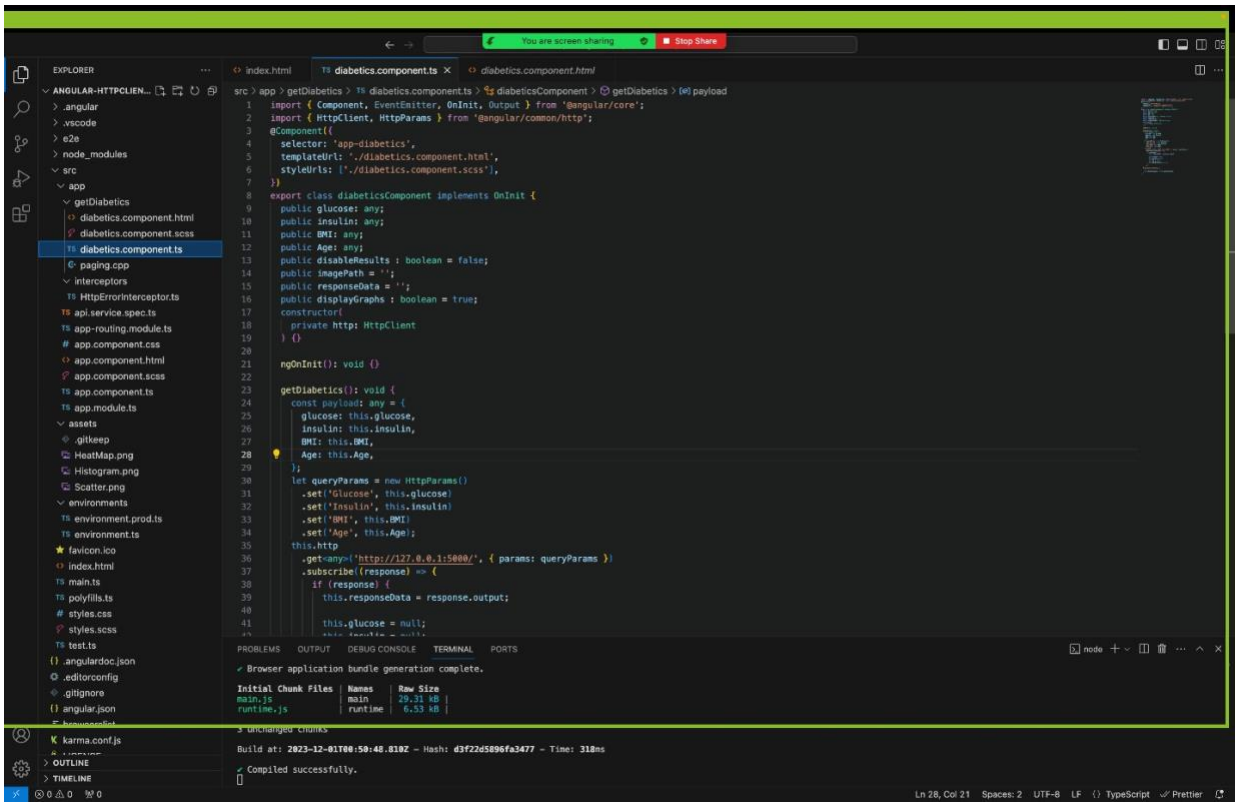
Explanation of code:





Above 2 screenshots the diabetes component html code which has both home and results webpage. Home page contains placeholders for Glucose level, Insulin, BMI and Age. Whenever the predict button is clicked, getDiabetes () method is called as shown in line number 33.

We can see the getDiabetics() function defined in the screenshot below where the entered values are paased to the backend though http.get method as seen in lines 35,36.



This will initate the predict block in backend app.py in below screenshot because it starts whenever GET is called. Inside this predict, These 4 parameters are converted into numpy array features and passed to the model.predict. We can notice that this model is already dumped and saved then called here. If the prediction turns out to be 1, ”You don’t have diabetes” message is returned. If it is 0, “You have diabetes, please consult a doctor” message is returned. This message is returned as json.

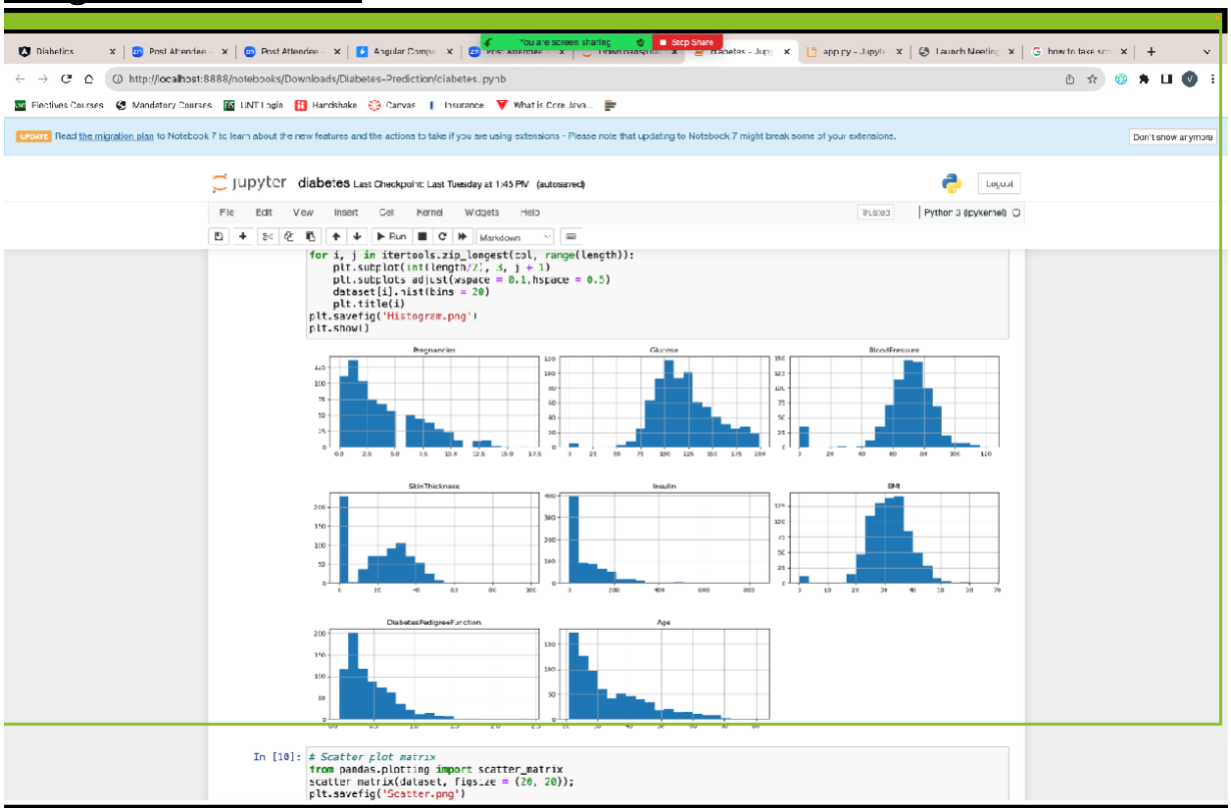
http://localhost:8888/edit/Downloads/Diabetes-Prediction/flask/app.py

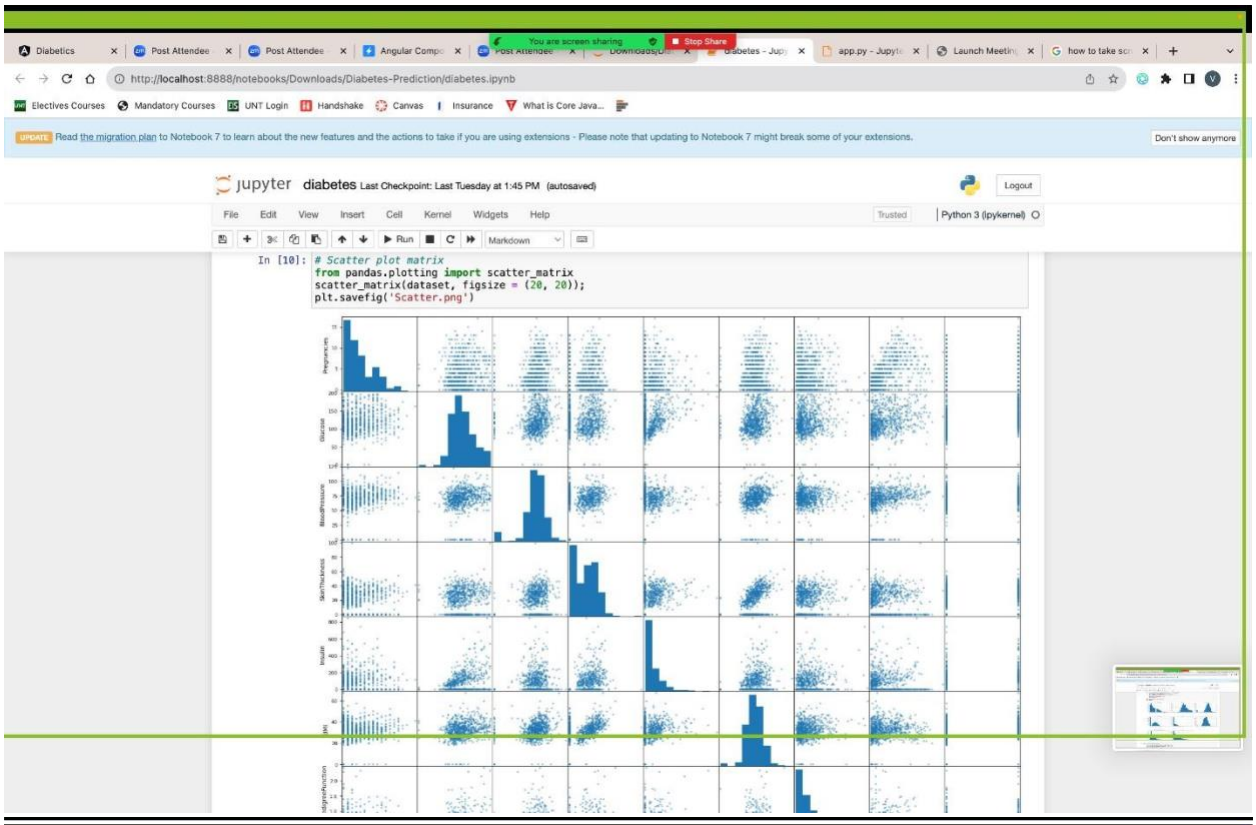
jupyter app.py a day ago

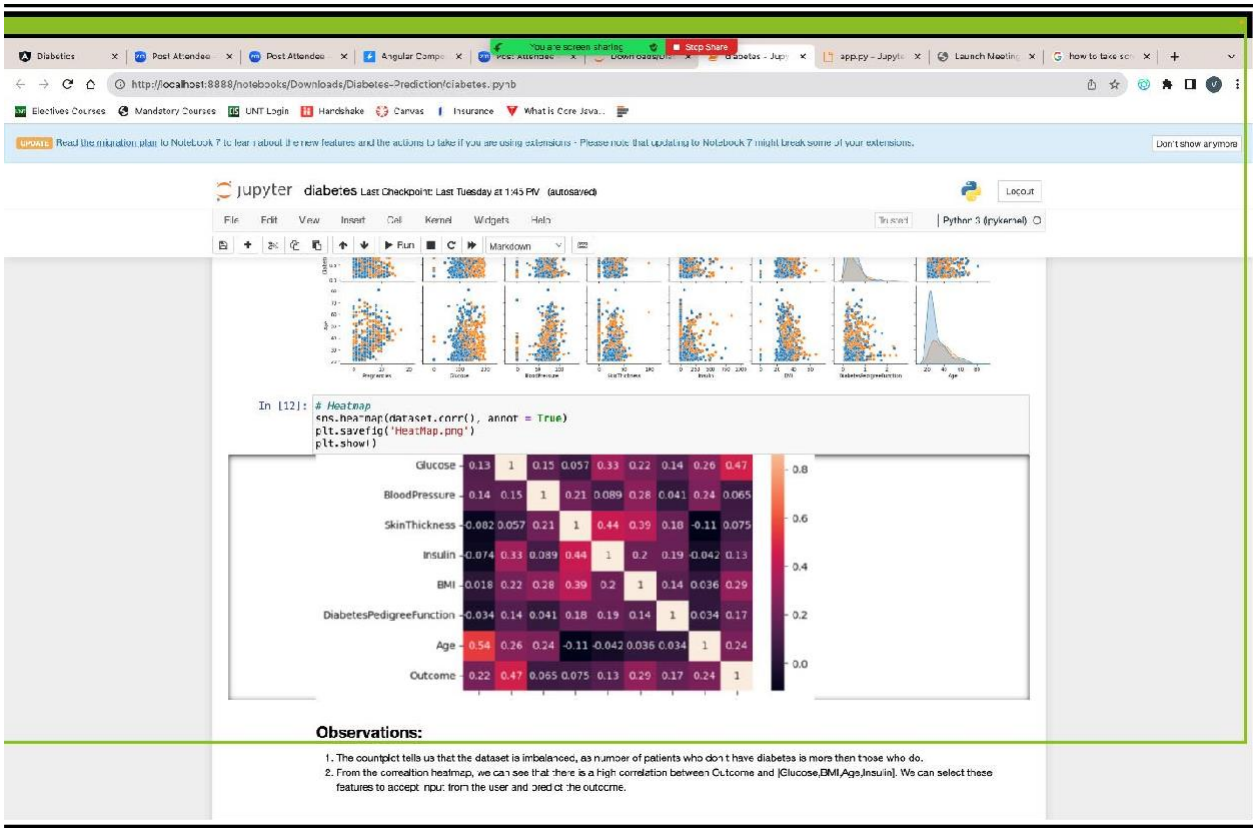
File Edit View Language Python

```python
app = Flask(__name__)
CORS(app, origins='*')

model = load('/Users/vineethreddy/Downloads/Diabetes-Prediction/model.joblib')

dataset = pd.read_csv('diabetes.csv')

dataset_X = dataset.iloc[:,[1, 2, 5, 7]].values

from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0,1))
dataset_scaled = sc.fit_transform(dataset_X)


@app.route('/', methods=['GET'])
def predict():



        # For both GET and POST requests, retrieve parameters from the query string
        glucose = request.args.get('Glucose')
        insulin = request.args.get('Insulin')
        bmi = request.args.get('BMI')
        age = request.args.get('Age')
        logging.info(f"Received parameters: Glucose={glucose}, Insulin={insulin}, BMI={bmi}, Age={age}")

        float_features = [glucose, insulin, bmi, age]

        final_features = [np.array(float_features)]
        prediction = model.predict(sc.transform(final_features))

        if prediction == 1:
            pred = "You have Diabetes, please consult a Doctor."
        elif prediction == 0:
            pred = "You don't have Diabetes."
        output = str(pred)
        logging.info(f"Final output: {'{}'.format(output)}")

        return jsonify({'output': output})


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

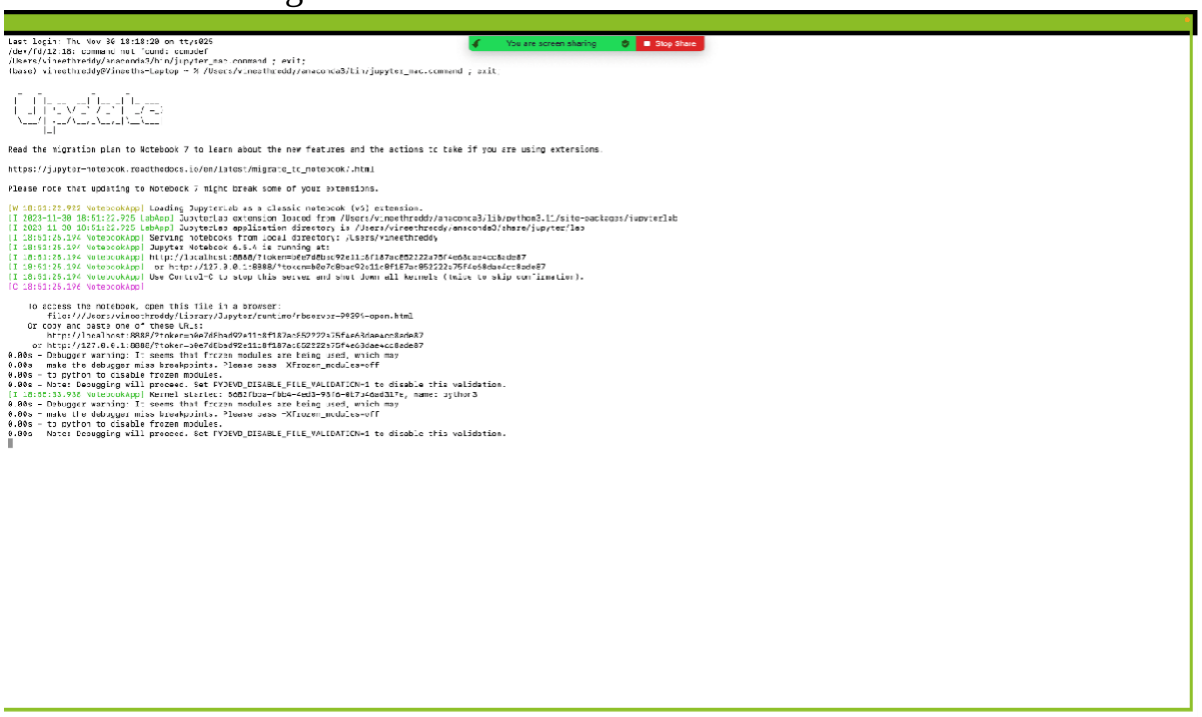
Images from backend:

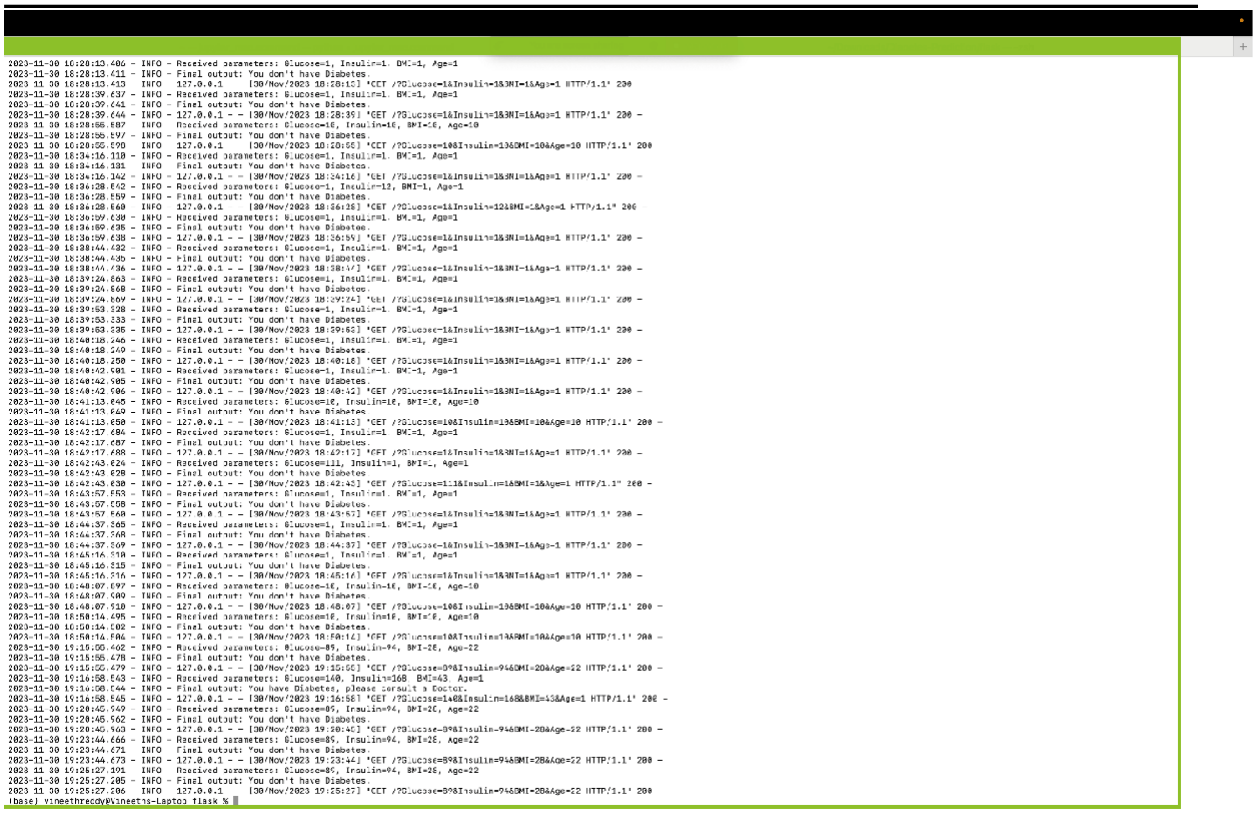



http://localhost:8888/notebooks/Downloads/Diabetes-Prediction/diabetes.ipynb
jupyter diabetes Last Checkpoint: Last Tuesday at 1:45 PM (autosaved)
In [10]: # Scatter plot matrix
from pandas.plotting import scatter_matrix
scatter_matrix(dataset, figsize = (20, 20));
plt.savefig('Scatter.png')

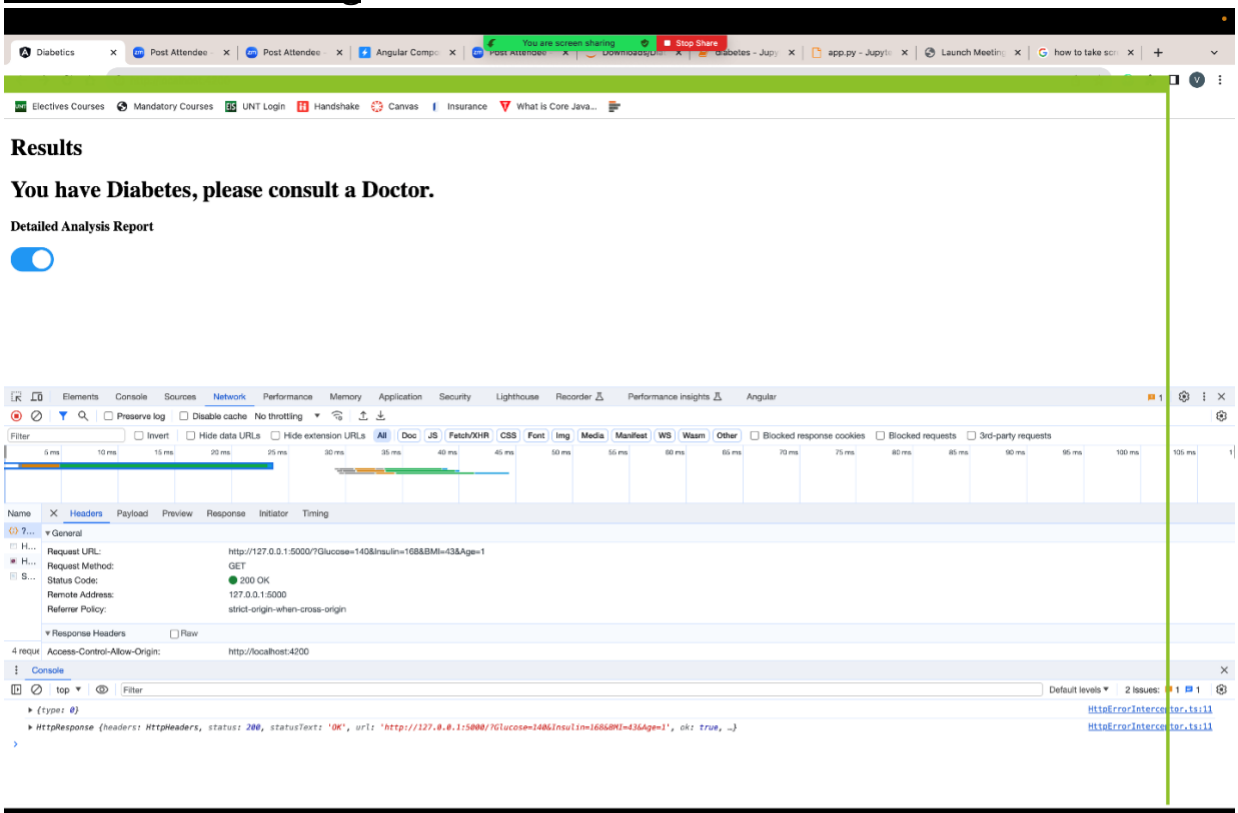

http://localhost:8888/notebooks/Downloads/Diabetes-Prediction/diabetes.ipynb
jupyter diabetes Last Checkpoint: Last Tuesday at 1:45 PM (autosaved)
In [12]: # Heatmap
sns.heatmap(dataset.corr(), annot = True)
plt.savefig('HeatMap.png')
plt.show()
Glucose 0.13 1 0.15 0.057 0.33 0.22 0.14 0.26 0.47
BloodPressure 0.14 0.15 1 0.21 0.089 0.28 0.041 0.24 0.065
SkinThickness -0.082 0.057 0.21 1 0.44 0.39 0.18 -0.11 0.075
Insulin -0.074 0.33 0.089 0.44 1 0.2 0.19 -0.042 0.13
BMI 0.018 0.22 0.28 0.39 0.2 1 0.14 0.036 0.29
DiabetesPedigreeFunction -0.034 0.14 0.041 0.18 0.19 0.14 1 0.034 0.17
Age 0.54 0.26 0.24 -0.11 -0.042 0.036 0.034 1 0.24
Outcome 0.22 0.47 0.065 0.075 0.13 0.29 0.17 0.24 1
0.8
0.6
0.4
0.2
0.0
Observations:
1. The countplot tells us that the dataset is imbalanced, as number of patients who don't have diabetes is more then those who do.
2. From the correlation heatmap, we can see that there is a high correlation between Outcome and [Glucose,BMI,Age,Insulin]. We can select these features to accept input from the user and predict the outcome.

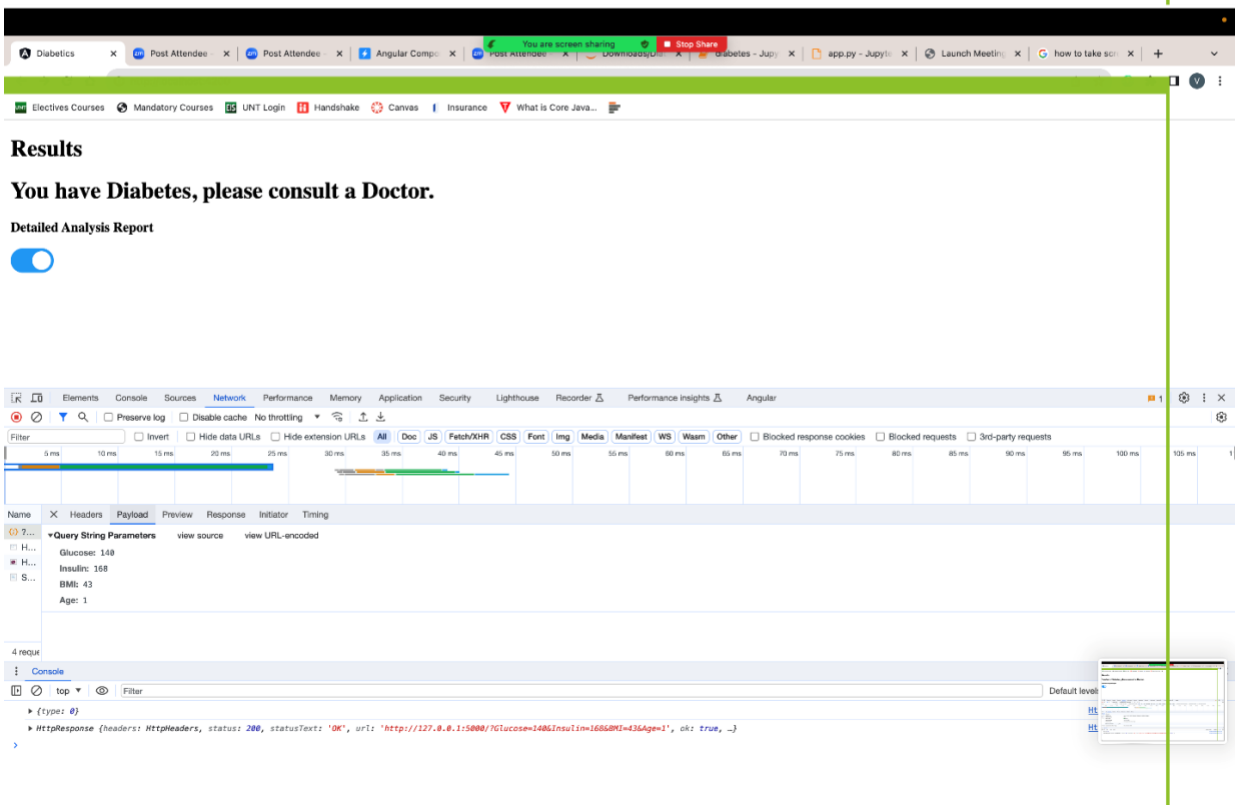
LOGS: Here, we can see how the values are traversed in the form of logs. We can see 2 sets of logs here.

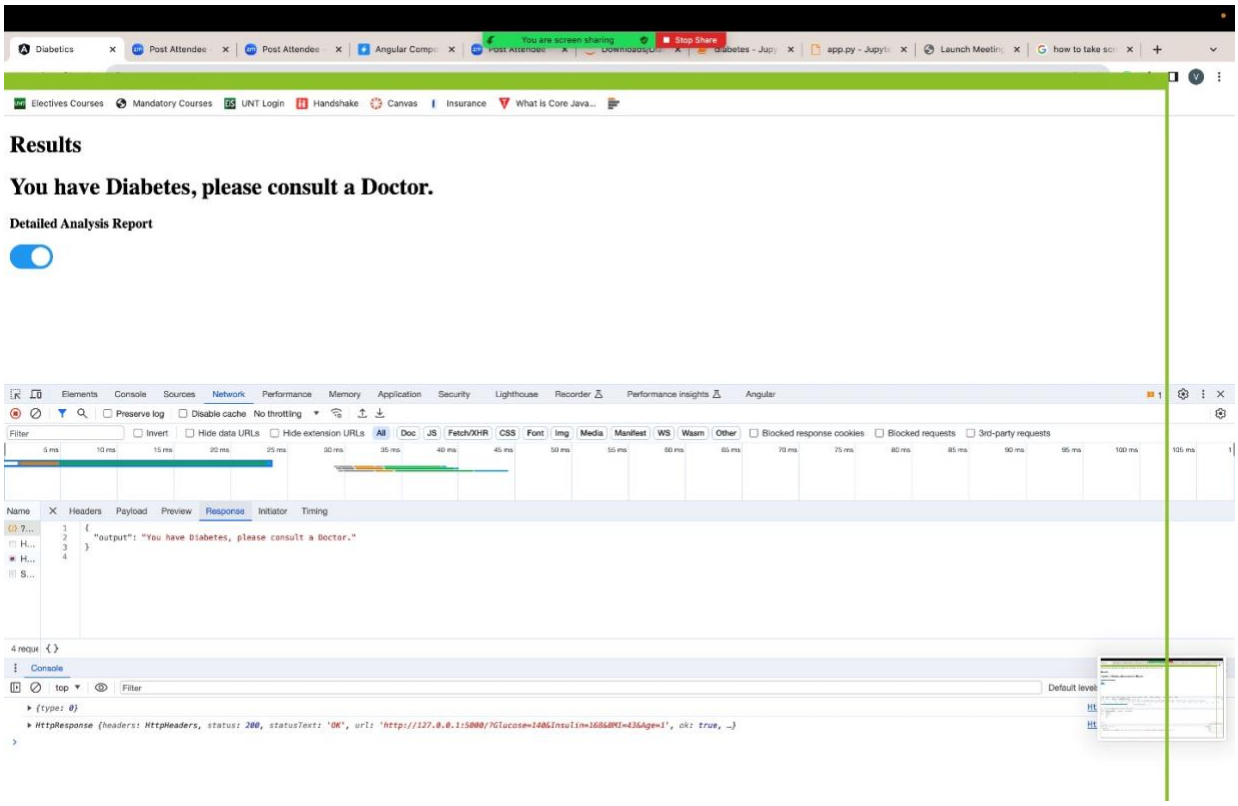




API and Data Routing







These last set of screenshots show how the data is traversed to the frontend which shows the header, payload, response.

Enhancements done are:

- Replaced pickle with joblib as a replacement for pickle to work efficiently on arbitrary Python objects containing large data, in particular large numpy arrays in the current Flask application.
- Transformed from the Basic UI to a UI having fields which having the report analysis.
- Resolved CORPS issue at the backend and front end.

3. Task 3: UI Demonstration

Link to the video demo: https://myunt-my.sharepoint.com/:v:/g/personal/suhassiddarajgaritellatakula_my_unt_edu/EbuGht170fNFv-E2KxGKV4sB9Luo87aGUSjs_Vu3CFnJrQ?e=NAQMFl