

Machine Learning Approach to Detect & Annotate Eye Diseases using  
Retinal Images 2023-162

Status Document

Perera H. A. N. S.

IT20166106

B.Sc. (Hons) Degree in Information Technology Specializing in  
Software Engineering

Department of Information Technology

Sri Lanka Institute of Information Technology Sri  
Lanka

September 2023

## Table of Contents

1.	Project progress.....	3
1.1	Frontend Implementation.....	3
1.2	Backend Implementation .....	4
1.3	Mobile App UIs.....	5
2.	Project View.....	6
3.	Gantt chart.....	7
4.	Screenshots of Conversations and Calls - Microsoft Teams .....	8

# 1. Project progress

## 1.1 Frontend Implementation

The screenshot shows the Visual Studio Code interface with the file `index.js` open. The code is part of the `AMD-Detection` component, specifically the `uploadImage` function. The code uses React hooks like `useState` and `useNavigation` to handle file uploads to Firebase storage. It also interacts with Firestore to update a document with the uploaded image URL. The code editor has syntax highlighting for JavaScript and includes comments explaining the logic. The sidebar shows the project structure with files like `index.js`, `firebase.js`, and various `.js` files for different screens.

```
const { image } = route.params;
const { imageType } = route.params;
const { docID } = route.params;

const navigation = useNavigation();

const [result, setResult] = React.useState("");
const [status, setStatus] = React.useState("");
const [imageURL, setImageURL] = React.useState("");
const [isLoading, setIsLoading] = React.useState(true);

const docRef = doc(db, "Patients", docID);

const uploadImage = async () => {
  let patientEyeDisease;

  if (imageType === "OCT") {
    patientEyeDisease = "Age-Related Macular Degeneration";
  } else if (imageType === "FUNDUS") {
    patientEyeDisease = "Diabetic Retinopathy";
  }

  const fileName = image.split("/").pop();
  const uploadResponse = await uploadToFirebase(image, fileName, v) =>
    console.log(`v: ${v}`);
};

console.log("uploadResponse: ", uploadResponse);
setImageURL(uploadResponse.downloadUrl);
```

Figure 1 – Frontend Implementation

The screenshot shows the continuation of the `index.js` file from Figure 1. The code now handles the response from the axios post request to the backend. It logs the new document field added to the existing document and then sets the loading state to false. The code also handles errors and updates the `image_url` state with the download URL from the response. The sidebar shows the same project structure as Figure 1.

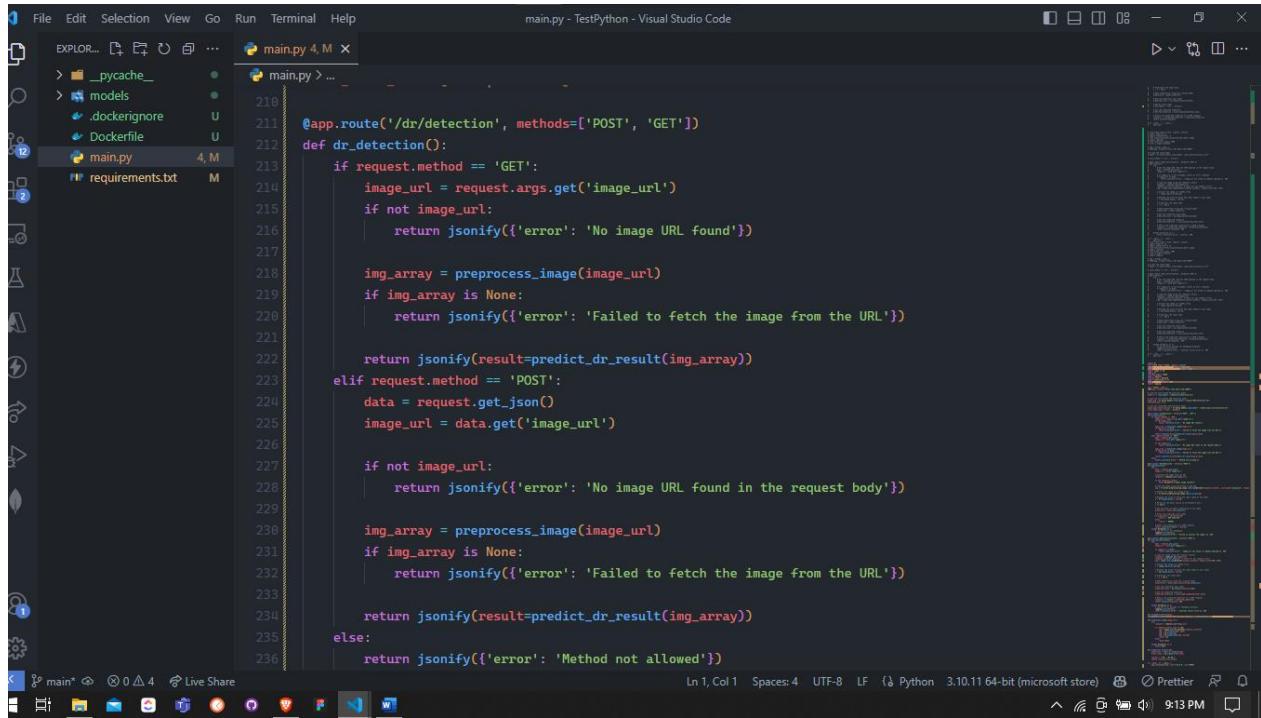
```
await updateDoc(docRef, patientDataRef)
  .then((docRef) => {
    console.log(`A New Document Field has been added to an existing document`);
  })
  .catch((error) => {
    console.log(error);
    setIsLoading(false);
  });

const data = {
  image_url: uploadResponse.downloadUrl,
};

axios
  .post(
    imageType === "OCT"
      ? "http://10.0.2.2:5000/amd/detection"
      : "http://10.0.2.2:5000/dr/detection",
    data
  )
  .then((response) => {
    // Handle the response data
    console.log(response.data.result);
    setResult(response.data.result);
  })
```

Figure 2- Frontend Implementation

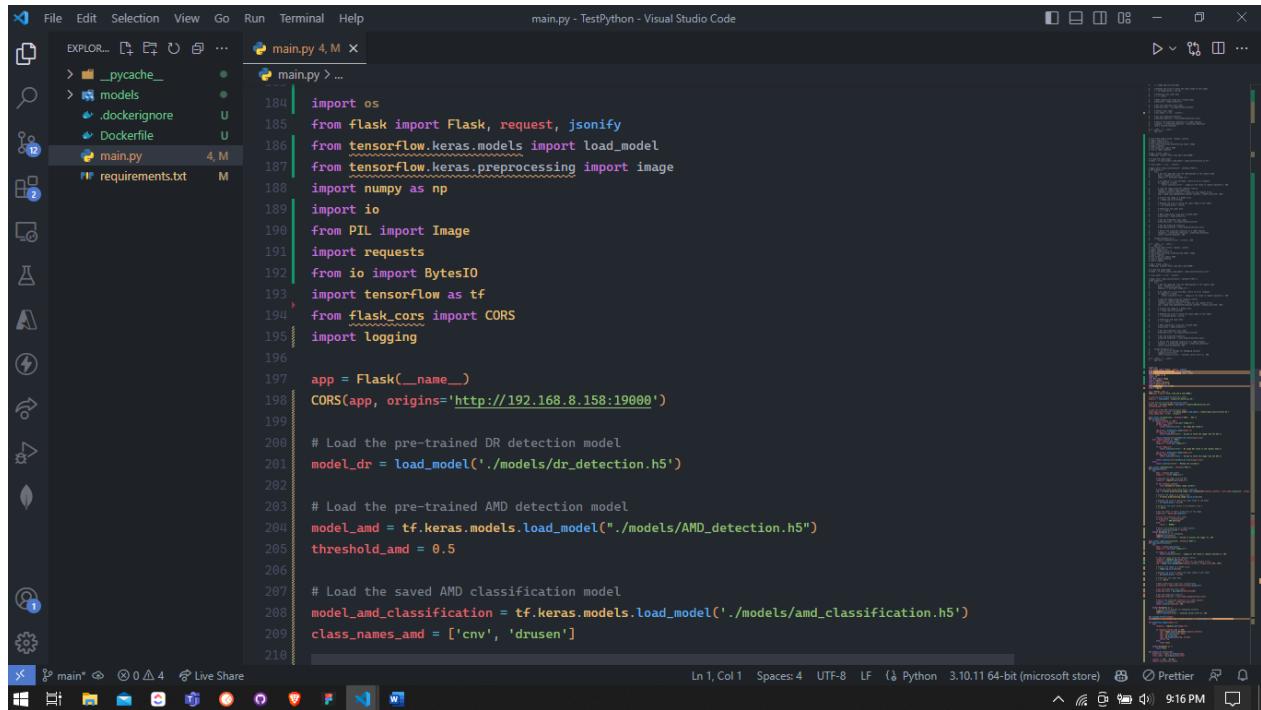
## 1.2 Backend Implementation



The screenshot shows the Visual Studio Code interface with the main.py file open. The code implements a Flask API endpoint for DR detection. It handles both GET and POST requests. For GET requests, it checks if an image URL is provided in the query parameters. If not, it returns an error. It then preprocesses the image and makes a prediction using a pre-trained DR detection model. For POST requests, it reads the JSON data from the request body, extracts the image URL, preprocesses the image, and makes a prediction. It also includes error handling for missing image URLs in the request body.

```
File Edit Selection View Go Run Terminal Help
EXPLORER main.py 4, M ...
main.py > ...
210
211 @app.route('/dr/detection', methods=['POST', 'GET'])
212 def dr_detection():
213     if request.method == 'GET':
214         image_url = request.args.get('image_url')
215         if not image_url:
216             return jsonify({'error': 'No image URL found'})
217
218         img_array = preprocess_image(image_url)
219         if img_array is None:
220             return jsonify({'error': 'Failed to fetch the image from the URL'})
221
222         return jsonify(result=predict_dr_result(img_array))
223     elif request.method == 'POST':
224         data = request.get_json()
225         image_url = data.get('image_url')
226
227         if not image_url:
228             return jsonify({'error': 'No image URL found in the request body'})
229
230         img_array = preprocess_image(image_url)
231         if img_array is None:
232             return jsonify({'error': 'Failed to fetch the image from the URL'})
233
234         return jsonify(result=predict_dr_result(img_array))
235     else:
236         return jsonify({'error': 'Method not allowed'})
Ln 1, Col 1 Spaces:4 UTF-8 LF (à Python 3.10.11 64-bit (microsoft store) Prettier 9:13 PM
```

Figure 3 - Backend Implementation

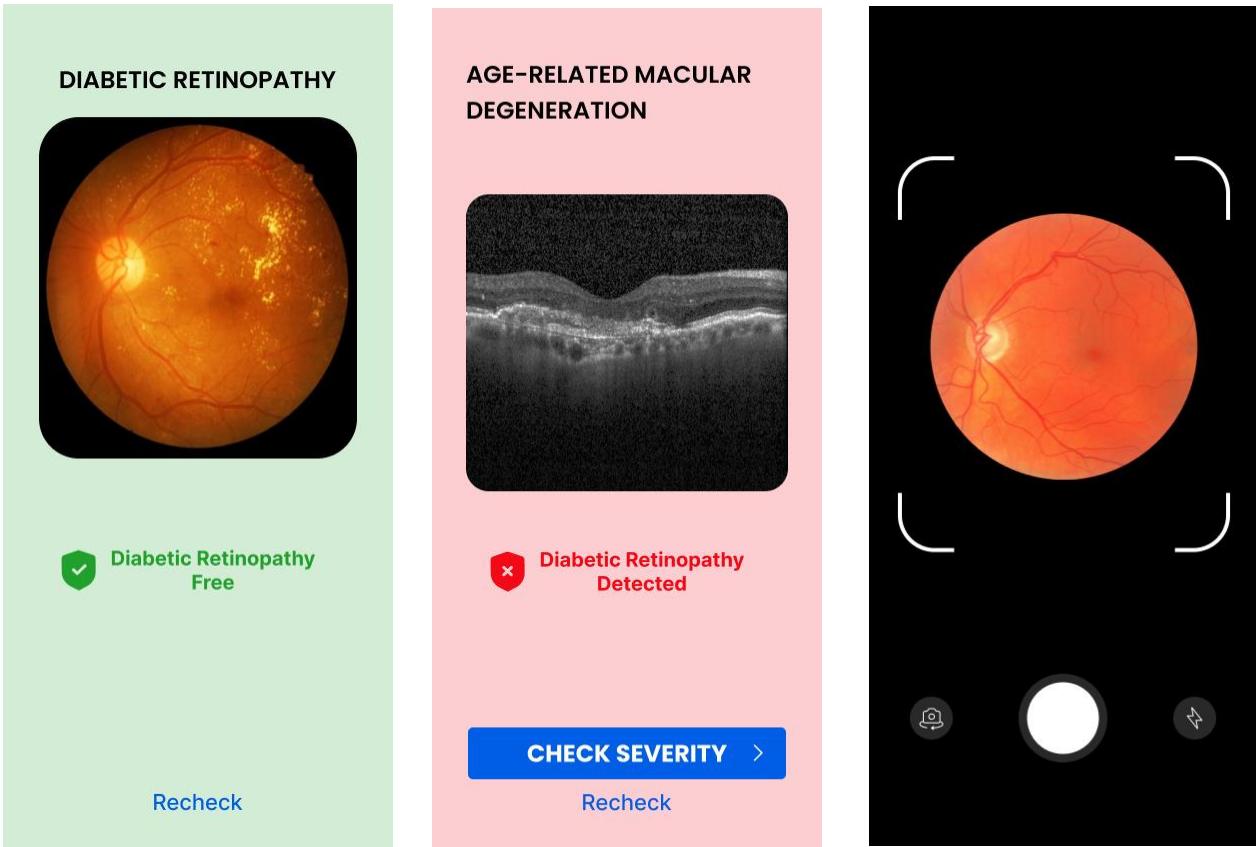


The screenshot shows the Visual Studio Code interface with the main.py file open. The code imports various libraries including Flask, requests, tensorflow.keras, and PIL. It defines a Flask application and enables CORS. The application loads three pre-trained models: a DR detection model (dr\_detection.h5), an AMD detection model (AMD\_detection.h5), and an AMD classification model (amd\_classification.h5). The AMD classification model's class names are set to ['cnv', 'drusen'].

```
File Edit Selection View Go Run Terminal Help
EXPLORER main.py 4, M ...
main.py > ...
184 import os
185 from flask import Flask, request, jsonify
186 from tensorflow.keras.models import load_model
187 from tensorflow.keras.preprocessing import image
188 import numpy as np
189 import io
190 from PIL import Image
191 import requests
192 from io import BytesIO
193 import tensorflow as tf
194 from flask_cors import CORS
195 import logging
196
197 app = Flask(__name__)
198 CORS(app, origins='http://192.168.8.158:19000')
199
200 # Load the pre-trained DR detection model
201 model_dr = load_model('./models/dr_detection.h5')
202
203 # Load the pre-trained AMD detection model
204 model_amd = tf.keras.models.load_model("./models/AMD_detection.h5")
205 threshold_amd = 0.5
206
207 # Load the saved AMD classification model
208 model_amd_classification = tf.keras.models.load_model('./models/amd_classification.h5')
209 class_names_amd = ['cnv', 'drusen']
Ln 1, Col 1 Spaces:4 UTF-8 LF (à Python 3.10.11 64-bit (microsoft store) Prettier 9:16 PM
```

Figure 4 - Backend Implementation

## 1.3 Mobile App UIs



## 2. Project View

The screenshot shows the Microsoft Planner interface in 'List' view. It displays three columns of tasks assigned to team members:

- Nilaksha:**
  - + Add task
  - Integrate the Trained Model (Due 07/22)
- Praveen:**
  - + Add task
  - Testing the Application (Due 07/23)
  - Integrate the Trained Model (Due 07/22)
- Rasanga:**
  - + Add task
  - Testing the Application (Due 07/23)
  - Integrate the Trained Model (Due 07/22)
  - Implement Functionalities of Application using React Native (Due 07/14)
  - Implement UI using React Native (Due 07/10)
  - Design Mobile Application Interfaces

Completed tasks counts are shown below each column: Nilaksha (10), Praveen (9), and Rasanga (0).

Figure 6 – Planner - Task List View

The screenshot shows the Microsoft Planner interface in 'Charts' view. It displays two main charts:

- Status:** A donut chart showing the distribution of tasks.
  - 10 Tasks left (Green)
  - Not started: 0
  - In progress: 0
  - Late: 10
  - Completed: 34
- Bucket:** A bar chart showing the distribution of tasks across four buckets.
  - Bucket 1: 12
  - Bucket 2: 10
  - Bucket 3: 10
  - Bucket 4: 10

Task lists for individual team members are also visible on the right side of the interface.

Figure 7 - Planner - Chart View

The screenshot shows the Microsoft Planner interface in 'Schedule' view for July 2023. The calendar grid displays tasks assigned to different team members. The tasks are color-coded and labeled with their descriptions. The sidebar on the left shows various team communication and management links like Activity, Chat, Teams, Assignments, Calendar, Calls, Files, Apps, and Help. The top navigation bar includes a search bar, a user profile icon, and buttons for Posts, Files, Planner, Meet, and other collaboration tools.

Figure 8 - Planner - Schedule View

### 3. Gantt chart

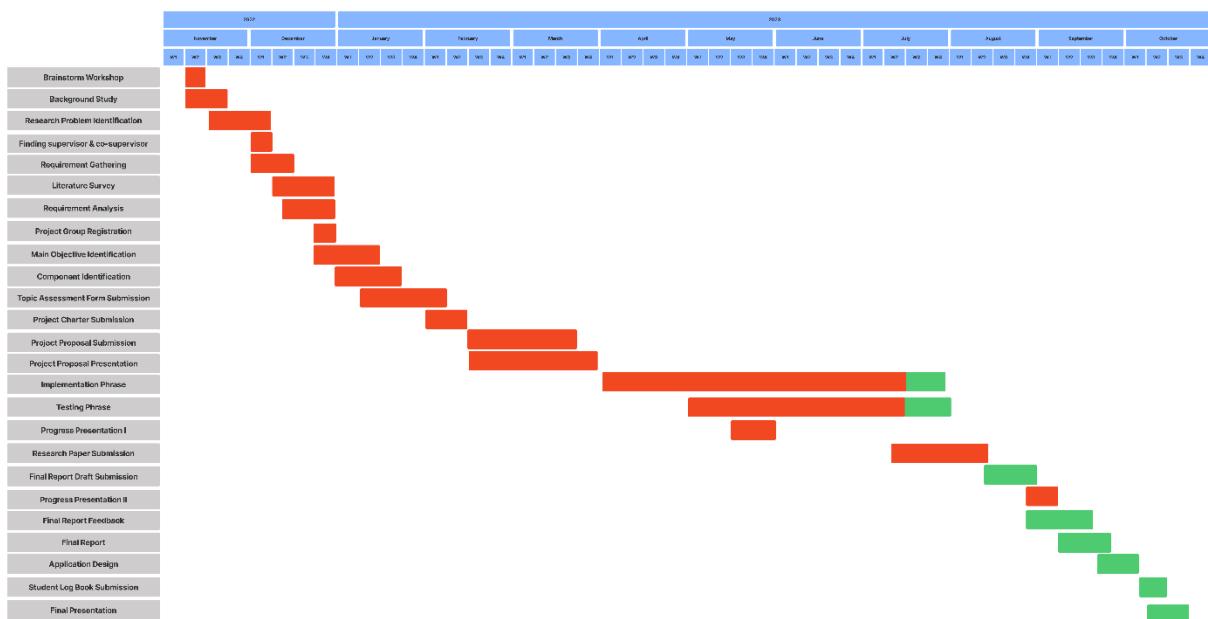


Figure 9 - Gantt Chart

## 4. Screenshots of Conversations and Calls - Microsoft Teams

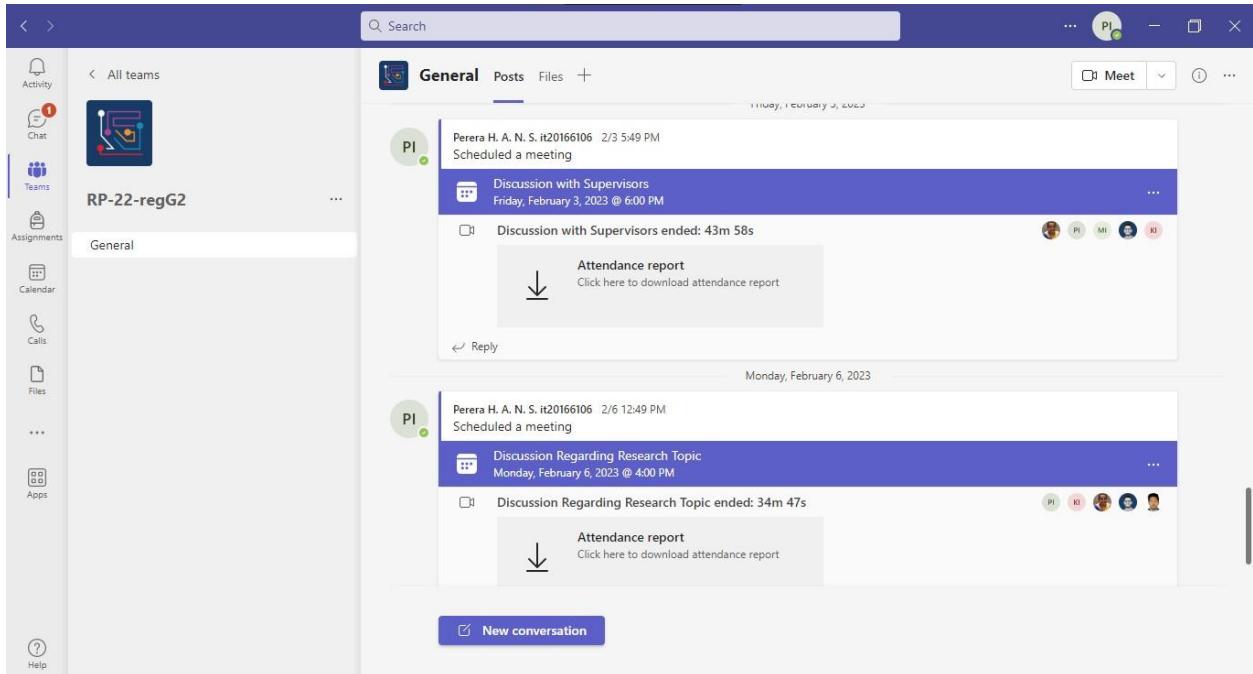


Figure 10 - MS Teams Channel

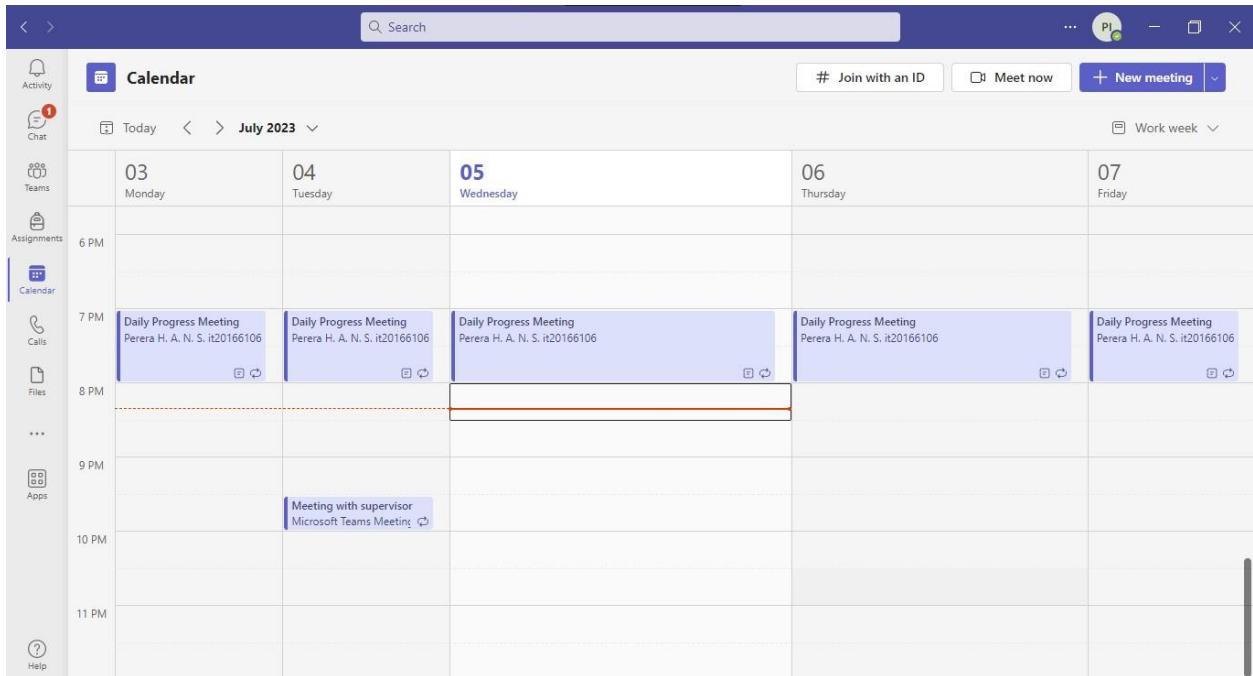


Figure 11 -Scheduled Meetings

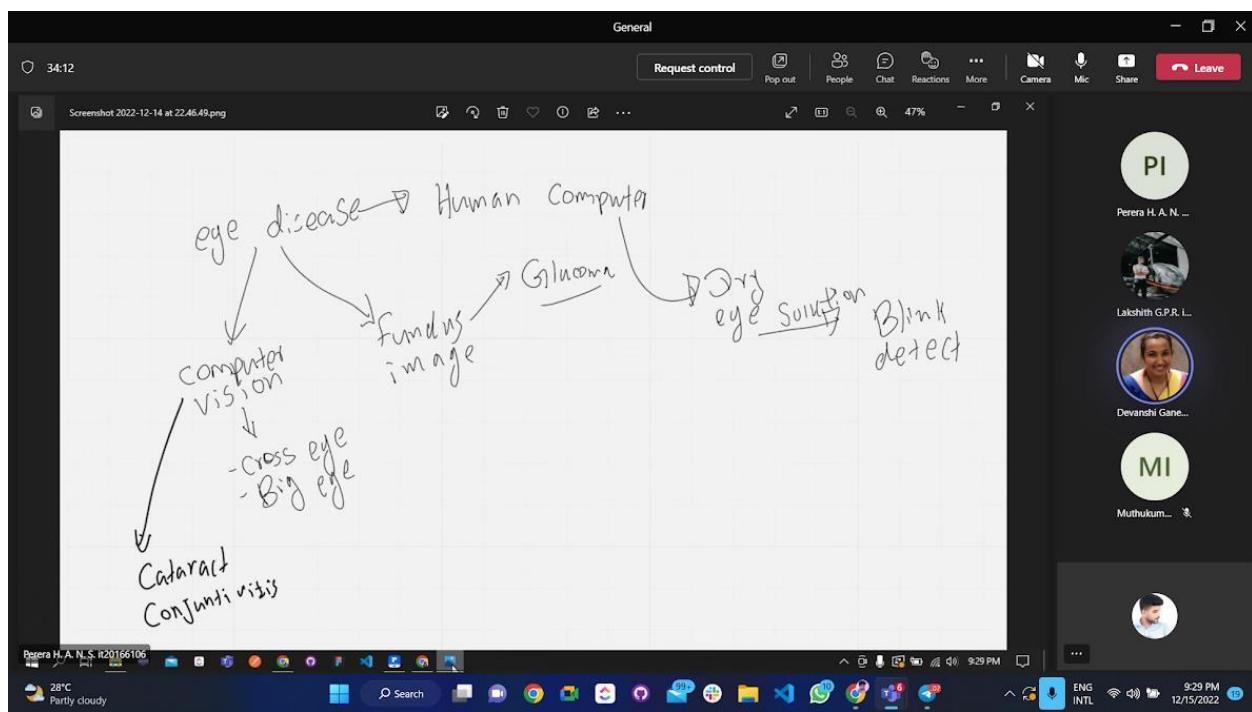


Figure 12 -Meetings with Supervisors

