

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

Status Document II

Muthukumarana M.W.A.N.C

IT20227890

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

Department of Information Technology

Sri Lanka Institute of Information Technology Sri
Lanka

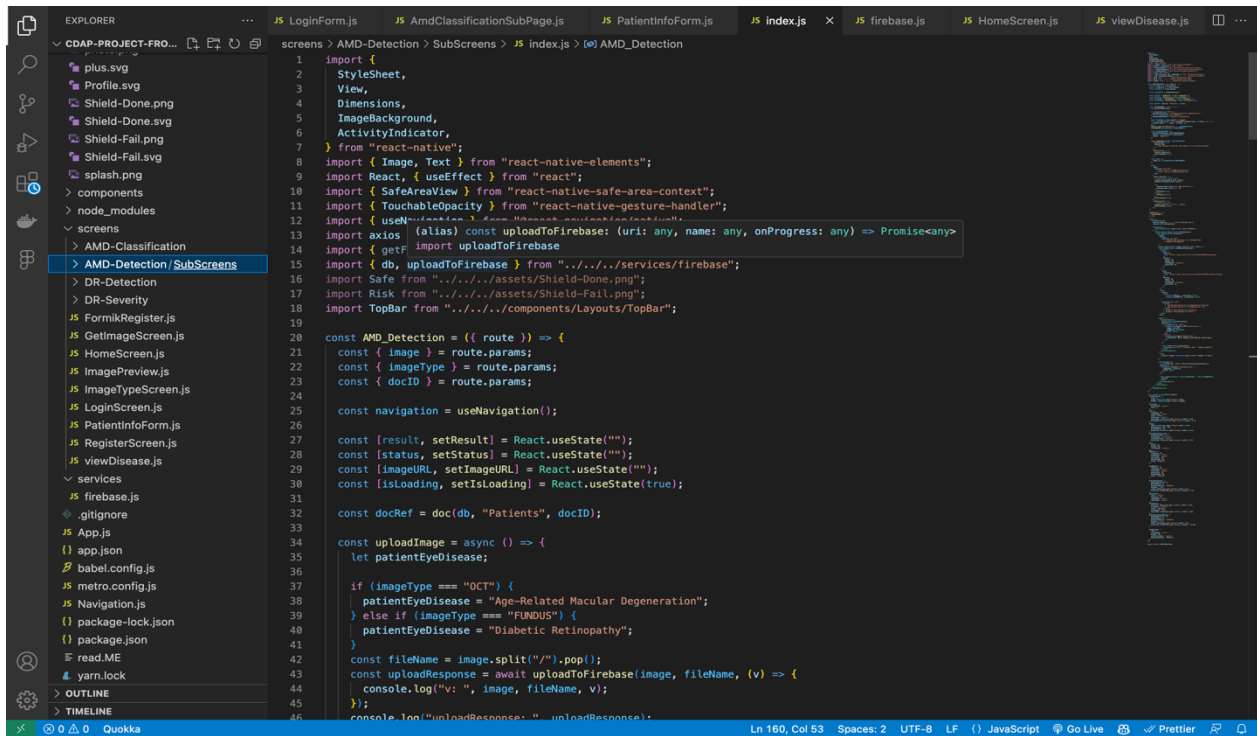
September 2023

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

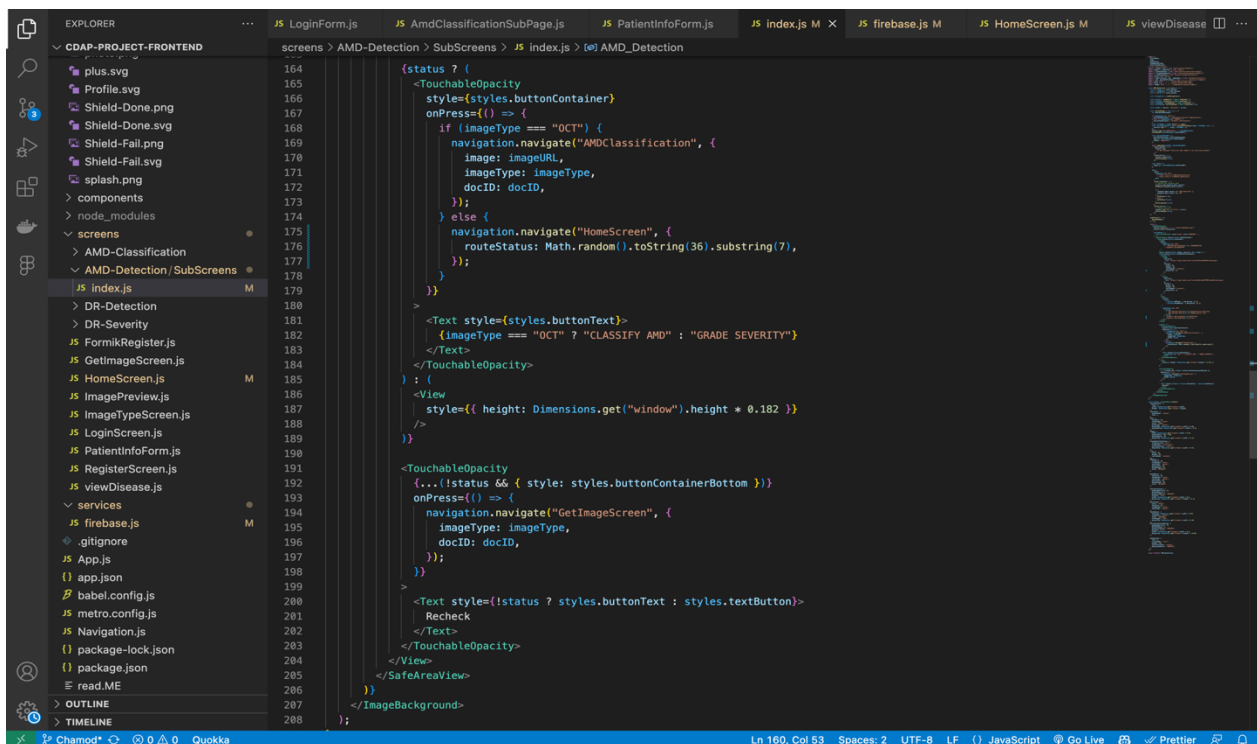
1. Project progress

1.1 Frontend Implementation



```
1 import {
2   StyleSheet,
3   View,
4   Dimensions,
5   ImageBackground,
6   ActivityIndicator,
7 } from "react-native";
8 import { Image, Text } from "react-native-elements";
9 import React, { useEffect } from "react";
10 import { SafeAreaView } from "react-native-safe-area-context";
11 import { TouchableOpacity } from "react-native-gesture-handler";
12 import { useNavigation } from "@react-navigation/native";
13 import axios (alias) const uploadToFirebase: (uri: any, name: any, onProgress: any) => Promise<any>
14 import { getF import uploadToFirebase
15 import { db, uploadToFirebase } from "../../services/firebase";
16 import Safe from "../../assets/Shield-Done.png";
17 import Risk from "../../assets/Shield-Fail.png";
18 import TopBar from "../../components/Layouts/TopBar";
19
20 const AMD_Detection = ({ route }) => {
21   const { image } = route.params;
22   const { imageType } = route.params;
23   const { docID } = route.params;
24
25   const navigation = useNavigation();
26
27   const [result, setResult] = React.useState("");
28   const [status, setStatus] = React.useState("");
29   const [imageUrl, setImageURL] = React.useState("");
30   const [isLoading, setIsLoading] = React.useState(true);
31
32   const docRef = doc(db, "Patients", docID);
33
34   const uploadImage = async () => {
35     let patientEyeDisease;
36
37     if (imageType === "OCT") {
38       patientEyeDisease = "Age-Related Macular Degeneration";
39     } else if (imageType === "FUNDUS") {
40       patientEyeDisease = "Diabetic Retinopathy";
41     }
42
43     const fileName = image.split("/").pop();
44     const uploadResponse = await uploadToFirebase(image, fileName, (v) => {
45       console.log("v: ", image, fileName, v);
46     });
47     console.log("uploadResponse: ", uploadResponse);
48   };
49 }
```

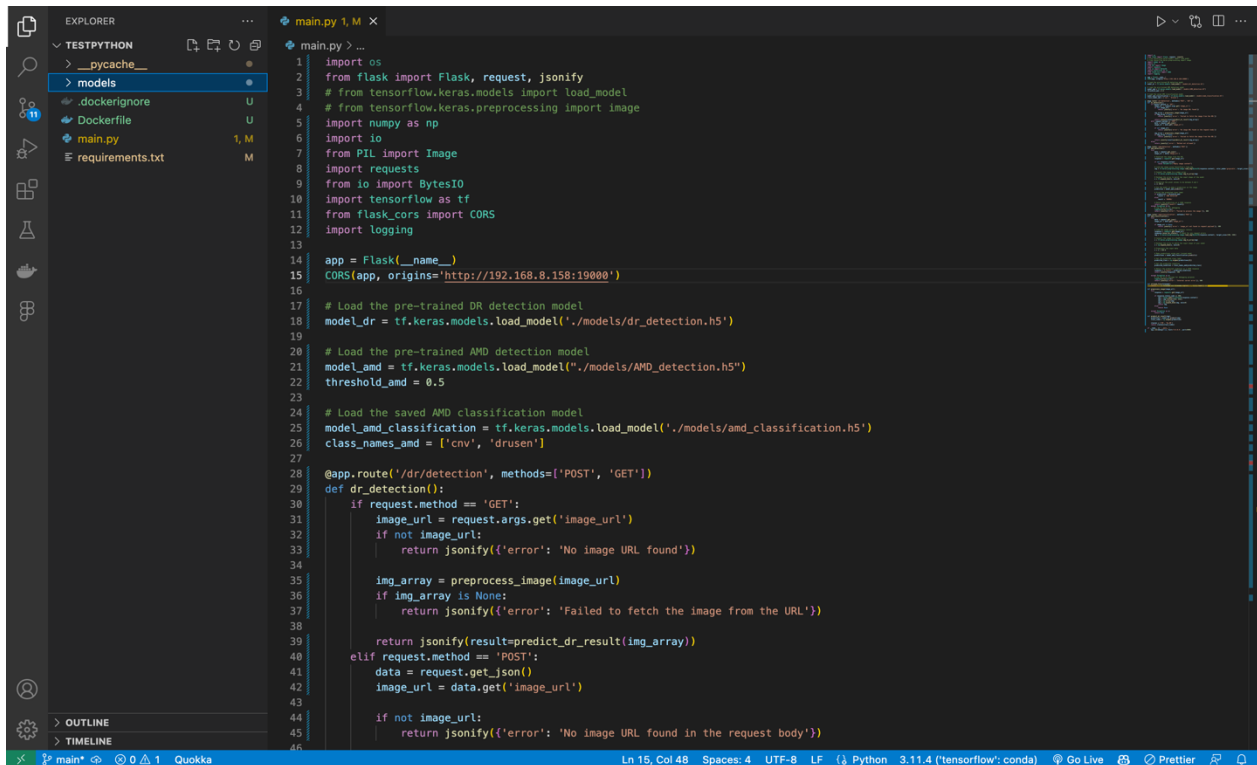
Figure 1 – Frontend Implementation



```
164 {status ? (
165   <TouchableOpacity
166     style={styles.buttonContainer}
167     onPress={() => {
168       if (imageType === "OCT") {
169         navigation.navigate("AMDClassification", {
170           image: imageUrl,
171           imageType: imageType,
172           docID: docID,
173         });
174       } else {
175         navigation.navigate("HomeScreen", {
176           routeStatus: Math.random().toString(36).substring(7),
177         });
178       }
179     }}
180   >
181     <Text style={styles.buttonText}>
182       {imageType === "OCT" ? "CLASSIFY AMD" : "GRADE SEVERITY"}
183     </Text>
184   </TouchableOpacity>
185 ) : (
186   <View
187     style={{ height: Dimensions.get("window").height * 0.182 }}
188   />
189 )
190 }
191
192 <TouchableOpacity
193   {...(status && { style: styles.buttonContainerBottom })}
194   onPress={() => {
195     navigation.navigate("GetImageScreen", {
196       imageType: imageType,
197       docID: docID,
198     });
199   }}
200 >
201   <Text style={status ? styles.buttonText : styles.textButton}>
202     Recheck
203   </Text>
204 </TouchableOpacity>
205 </View>
206 </SafeAreaView>
207 )
208 </ImageBackground>
209 }
```

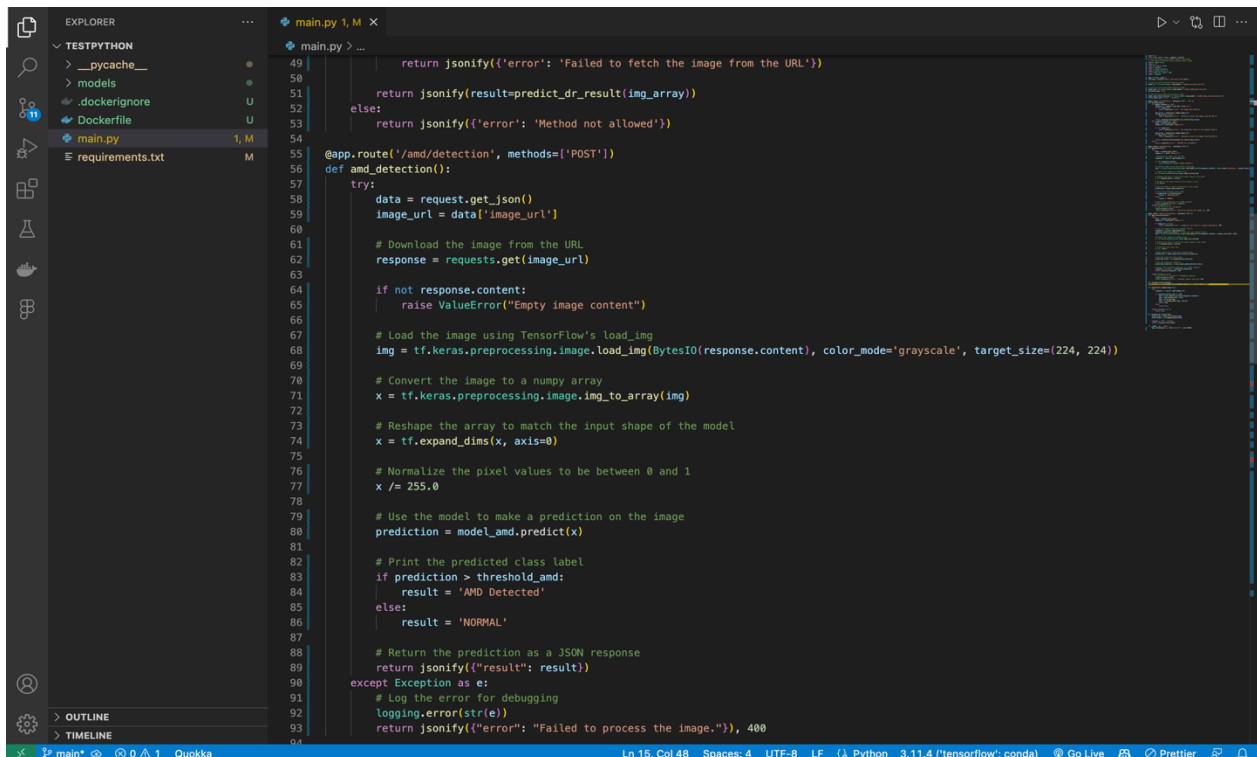
Figure 2- Frontend Implementation

1.2 Backend Implementation



```
1 import os
2 from flask import Flask, request, jsonify
3 # from tensorflow.keras.models import load_model
4 # from tensorflow.keras.preprocessing import image
5 import numpy as np
6 import io
7 from PIL import Image
8 import requests
9 from io import BytesIO
10 import tensorflow as tf
11 from flask_cors import CORS
12 import logging
13
14 app = Flask(__name__)
15 CORS(app, origins='http://192.168.8.158:19080')
16
17 # Load the pre-trained DR detection model
18 model_dr = tf.keras.models.load_model('./models/dr_detection.h5')
19
20 # Load the pre-trained AMD detection model
21 model_amd = tf.keras.models.load_model('./models/AMD_detection.h5')
22 threshold_amd = 0.5
23
24 # Load the saved AMD classification model
25 model_amd_classification = tf.keras.models.load_model('./models/amd_classification.h5')
26 class_names_amd = ['cnv', 'drusen']
27
28 @app.route('/dr/detection', methods=['POST', 'GET'])
29 def dr_detection():
30     if request.method == 'GET':
31         image_url = request.args.get('image_url')
32         if not image_url:
33             return jsonify({'error': 'No image URL found'})
34
35         img_array = preprocess_image(image_url)
36         if img_array is None:
37             return jsonify({'error': 'Failed to fetch the image from the URL'})
38
39         return jsonify(result=predict_dr_result(img_array))
40     elif request.method == 'POST':
41         data = request.get_json()
42         image_url = data.get('image_url')
43
44         if not image_url:
45             return jsonify({'error': 'No image URL found in the request body'})
46
```

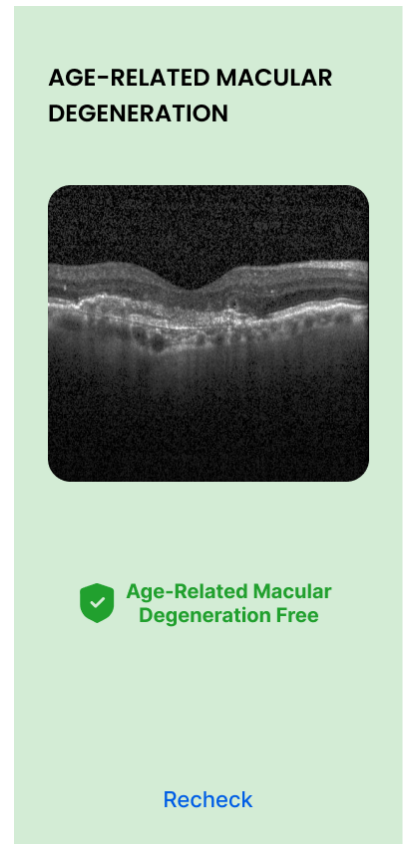
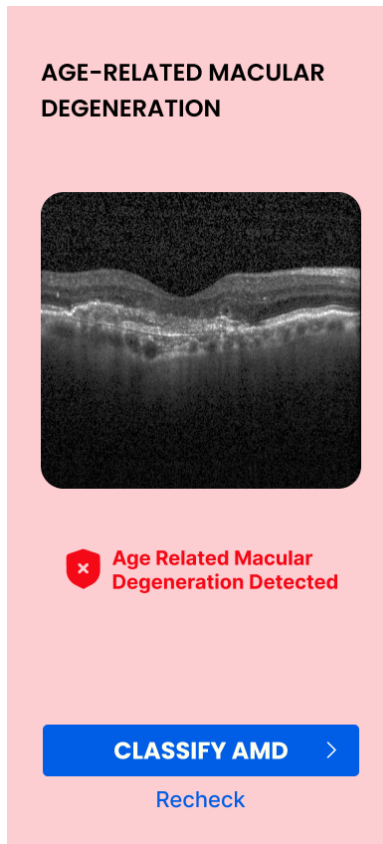
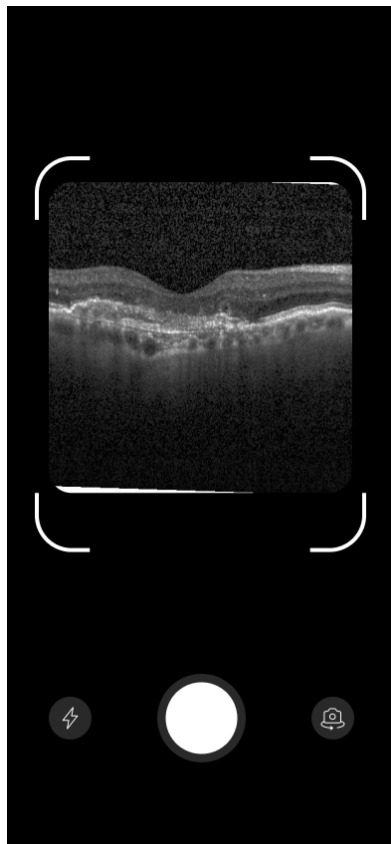
Figure 3 - Backend Implementation



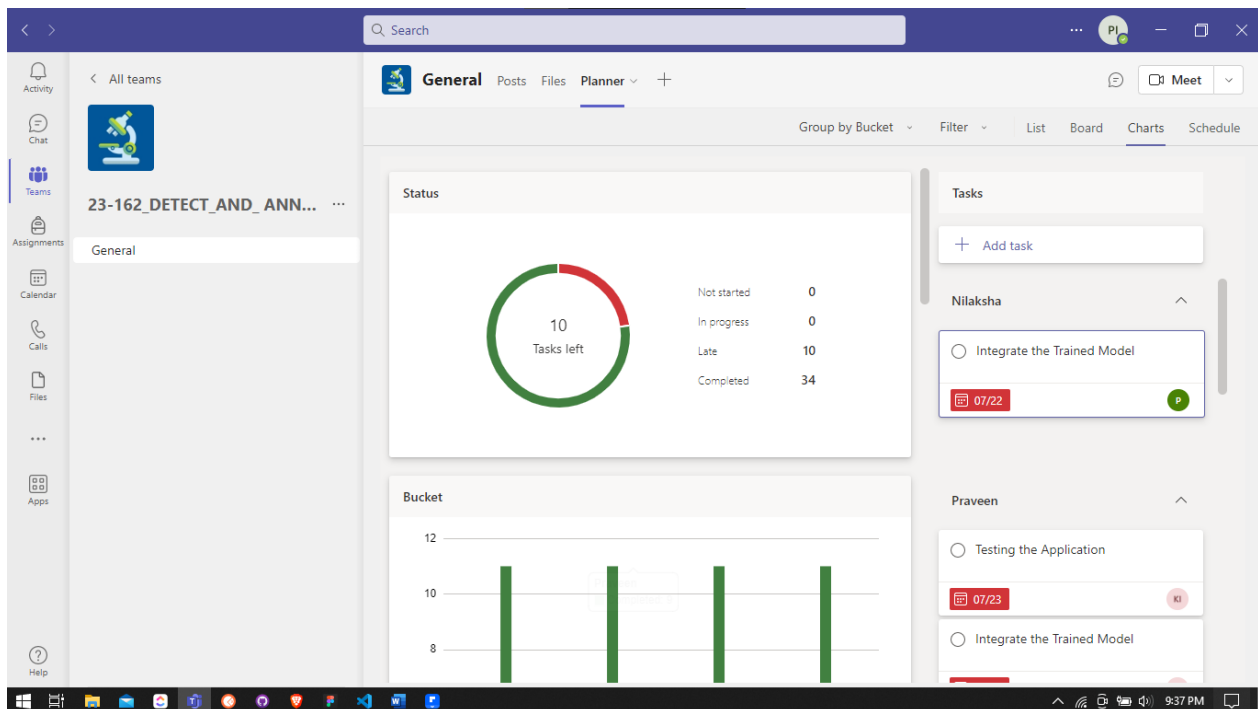
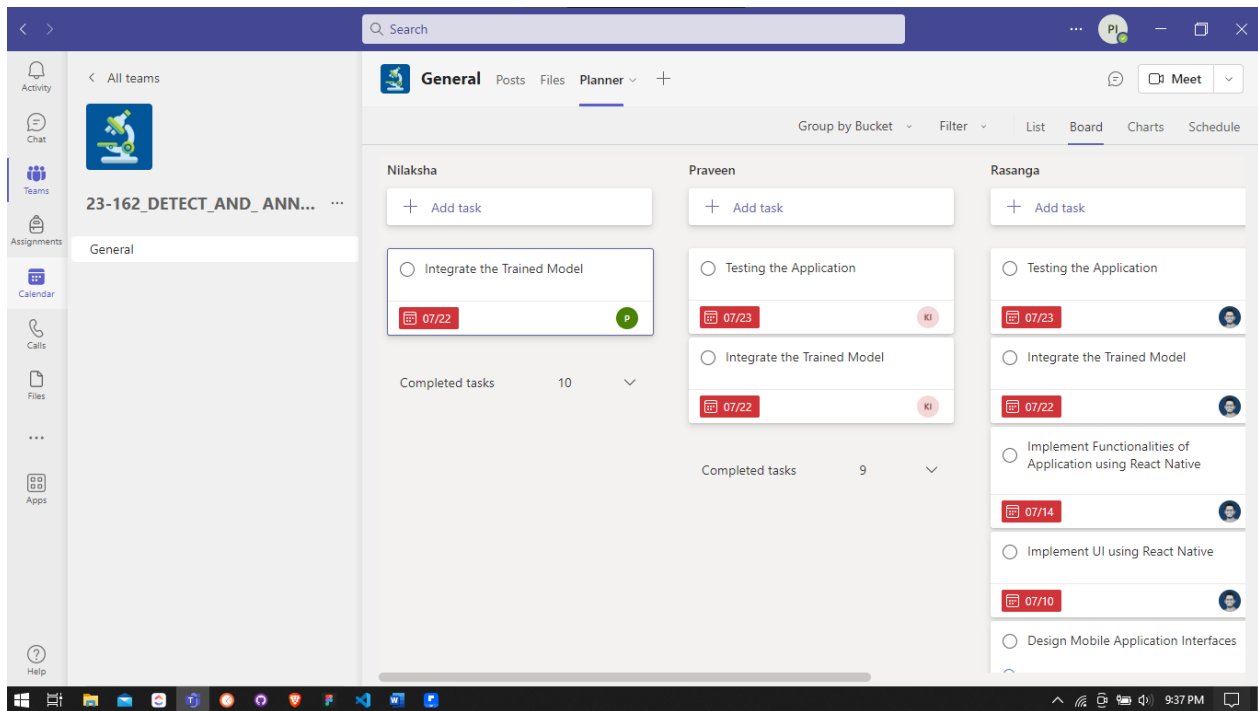
```
49         return jsonify({'error': 'Failed to fetch the image from the URL'})
50
51         return jsonify(result=predict_dr_result(img_array))
52     else:
53         return jsonify({'error': 'Method not allowed'})
54
55 @app.route('/amd/detection', methods=['POST'])
56 def amd_detection():
57     try:
58         data = request.get_json()
59         image_url = data['image_url']
60
61         # Download the image from the URL
62         response = requests.get(image_url)
63
64         if not response.content:
65             raise ValueError("Empty image content")
66
67         # Load the image using TensorFlow's load_img
68         img = tf.keras.preprocessing.image.load_img(BytesIO(response.content), color_mode='grayscale', target_size=(224, 224))
69
70         # Convert the image to a numpy array
71         x = tf.keras.preprocessing.image.img_to_array(img)
72
73         # Reshape the array to match the input shape of the model
74         x = tf.expand_dims(x, axis=0)
75
76         # Normalize the pixel values to be between 0 and 1
77         x /= 255.0
78
79         # Use the model to make a prediction on the image
80         prediction = model_amd.predict(x)
81
82         # Print the predicted class label
83         if prediction > threshold_amd:
84             result = 'AMD Detected'
85         else:
86             result = 'NORMAL'
87
88         # Return the prediction as a JSON response
89         return jsonify({"result": result})
90     except Exception as e:
91         # Log the error for debugging
92         logging.error(str(e))
93         return jsonify({"error": "Failed to process the image."}), 400
94
```

Figure 4 - Backend Implementation

1.3 Mobile App UIs



2. Project View



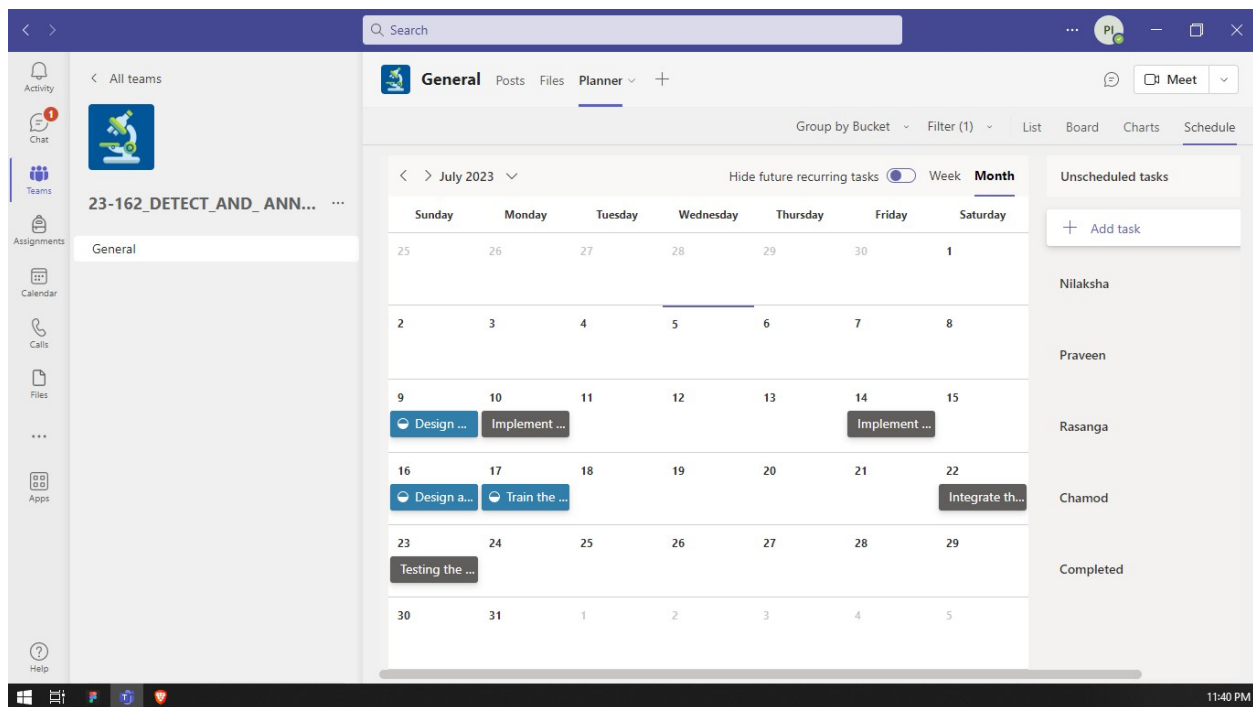


Figure 8 - Planner - Schedule View

3. Gantt chart

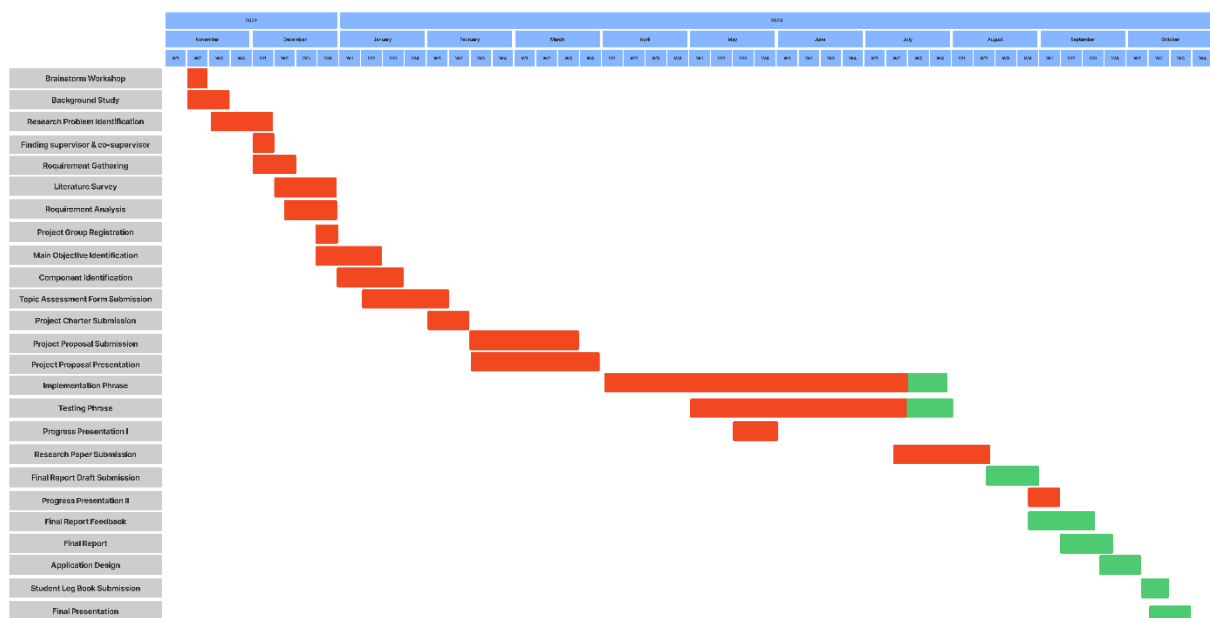


Figure 9 - Gantt Chart

4. Screenshots of Conversations and Calls - Microsoft Teams

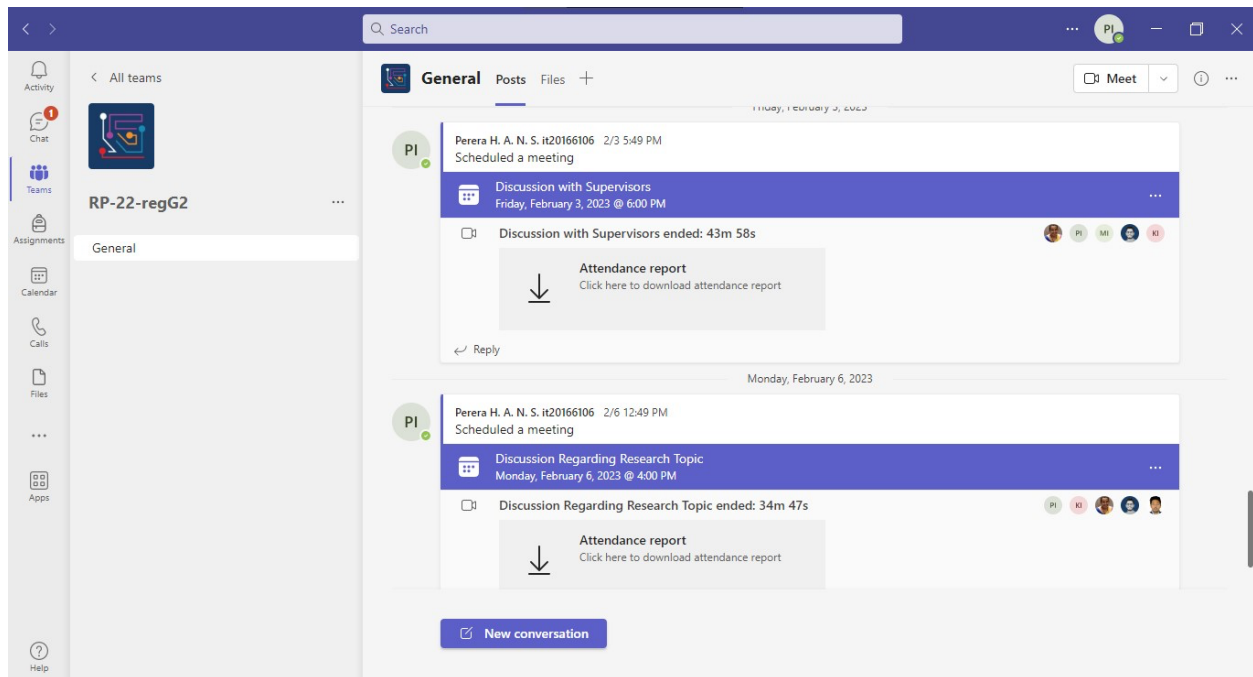


Figure 10 - MS Teams Channel

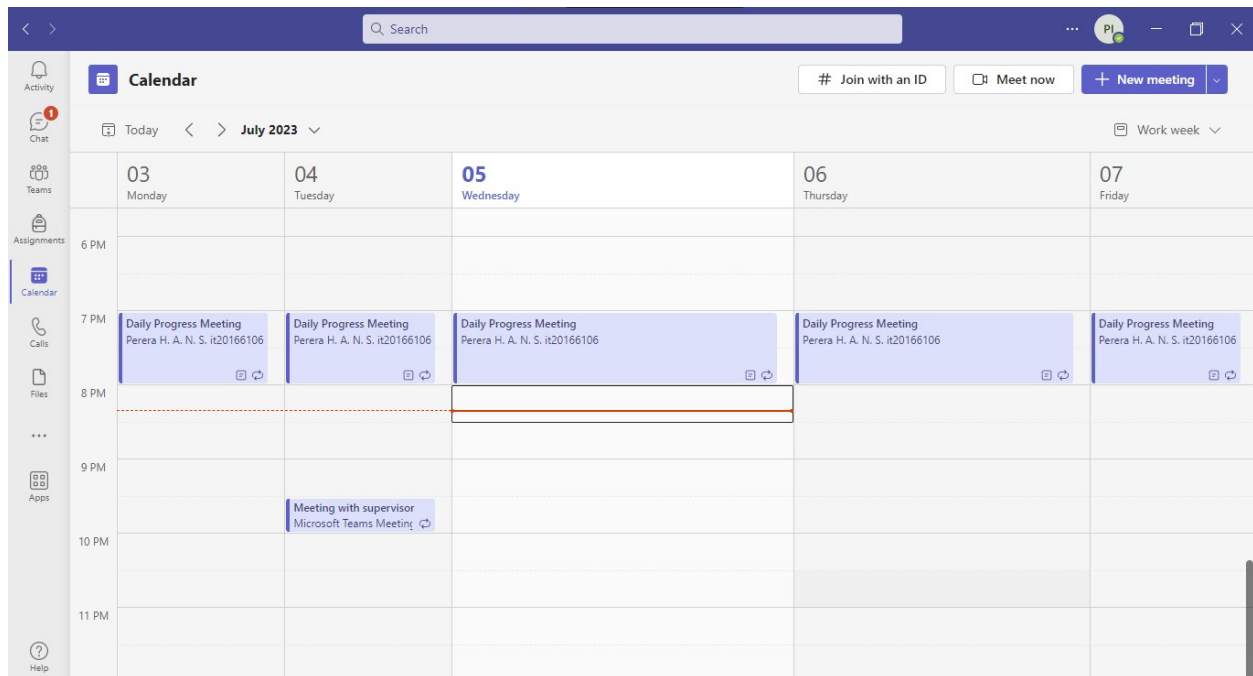


Figure 11 - Scheduled Meetings

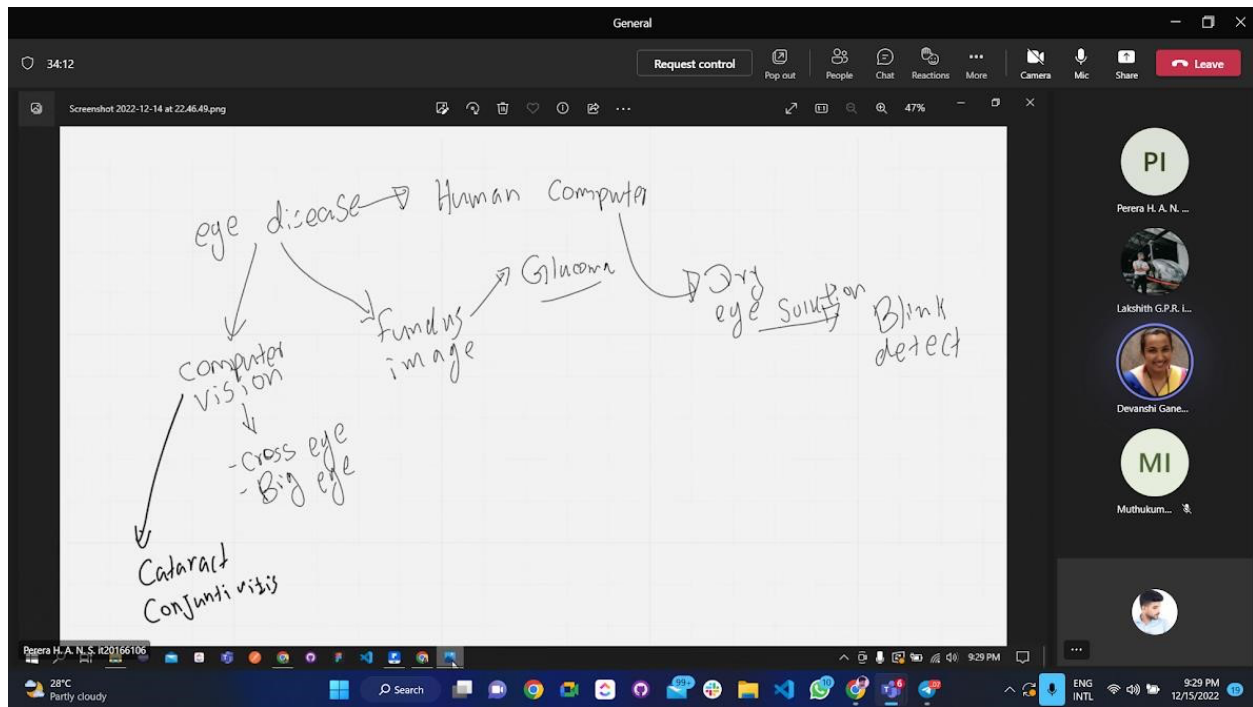


Figure 12 - Meetings with Supervisors

