

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

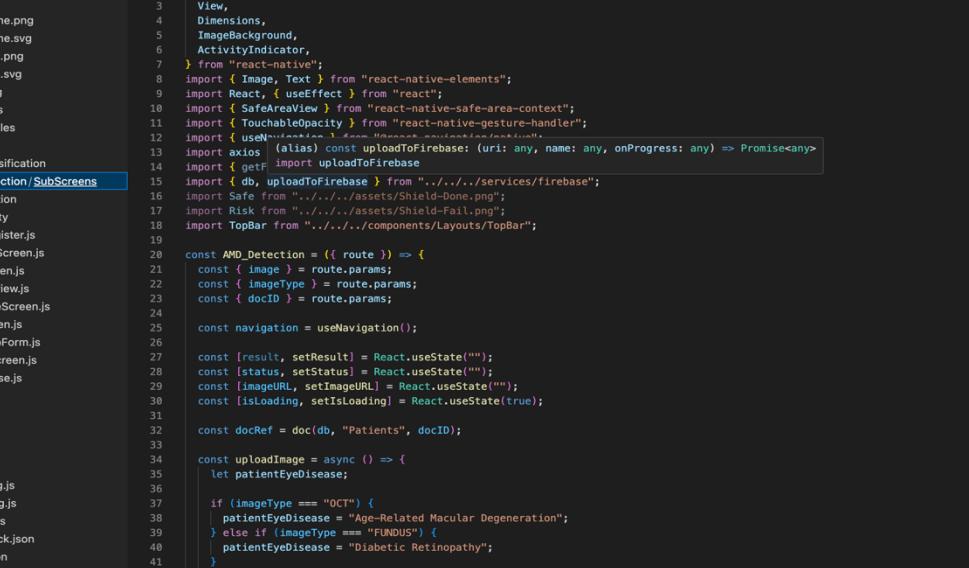
September 2023

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 { get } 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: ${v}, image, fileName, v`);
46     });
47     console.log(`UploadResponse: ${uploadResponse}`);
48   }
49
50   const handleImageUpload = () => {
51     uploadImage();
52     navigation.navigate("Result", { result, status, imageURL });
53   }
54
55   return (
56     <SafeAreaView style={styles.container}>
57       <ImageBackground source={image} style={styles.image}>
58         <View style={styles.overlay}>
59           <Text style={styles.title}>${patientEyeDisease}</Text>
60           <Text style={styles.message}>${status}</Text>
61           <Text style={styles.message}>${result}</Text>
62           <Text style={styles.message}>${imageURL}</Text>
63           <Text style={styles.message}>${isLoading}</Text>
64           <Text style={styles.message}>${uploadResponse}</Text>
65           <Text style={styles.message}>${uploadImage}</Text>
66           <Text style={styles.message}>${handleImageUpload}</Text>
67         </View>
68       </ImageBackground>
69     </SafeAreaView>
70   );
71 }
72
73 export default AMD_Detection;
```

Figure 1 – Frontend Implementation



The screenshot shows a code editor with the following details:

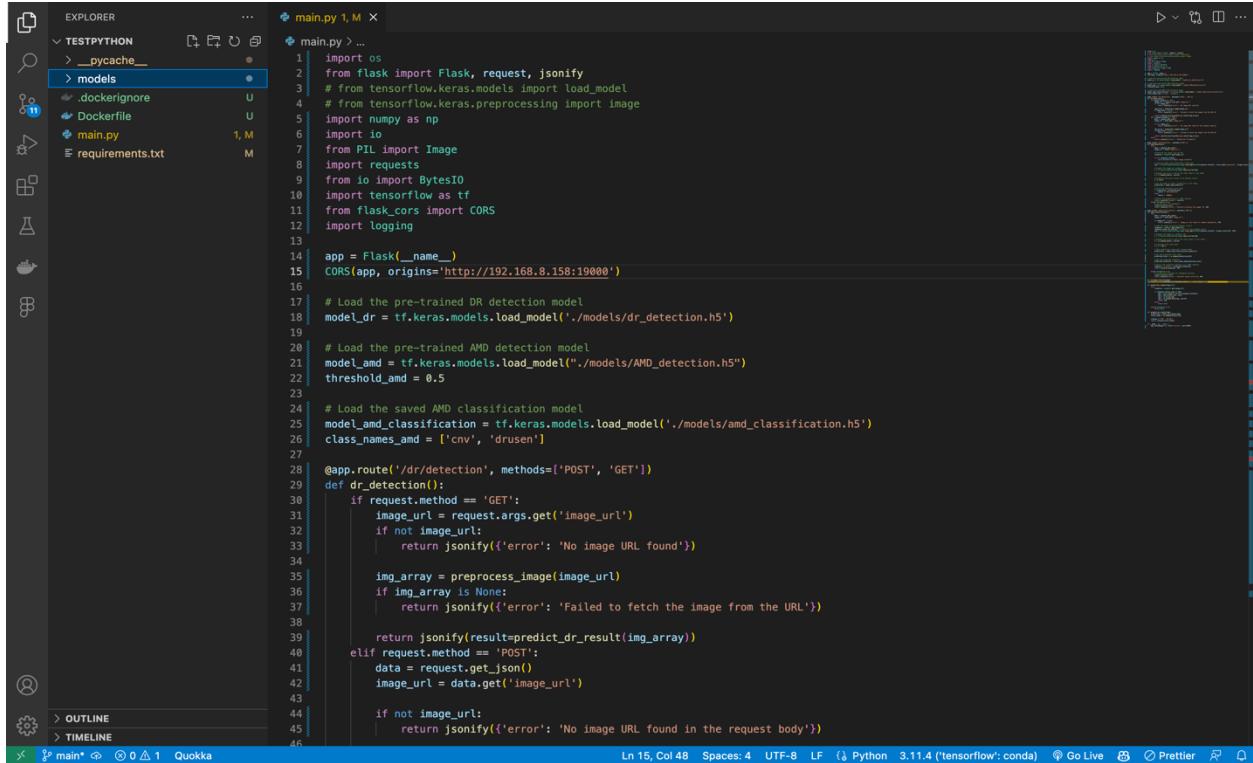
- Left Sidebar (Explorer):** Lists the project structure. Key files include `CDAP-PROJECT-FRONTEND`, `index.js`, `DR-Detection`, `DR-Severity`, `FormikRegister.js`, `GetImageScreen.js`, `HomeScreen.js`, `ImagePreview.js`, `ImageTypeScreen.js`, `LoginScreen.js`, `PatientInfoForm.js`, `RegisterScreen.js`, `viewDisease.js`, `services`, `firebase.js`, `.gitignore`, `App.js`, `app.json`, `babel.config.js`, `metro.config.js`, `Navigation.js`, `package-lock.json`, `package.json`, `read.ME`, `OUTLINE`, and `TIMELINE`.
- Top Bar:** Shows tabs for various files: `LoginForm.js`, `AmdClassificationSubPage.js`, `PatientInfoForm.js`, `index.js`, `firebase.js`, `HomeScreen.js`, and `viewDisease.js`.
- Current File:** `index.js` under `AMD-Detection/SubScreens`. The code is a functional component for `AMD_Detection` with the following structure:

```
status ? (
  <TouchableOpacity
    style={styles.buttonContainer}
    onPress={() => {
      if (imageType === "OCT") {
        navigation.navigate("AMDClassification", {
          image: imageURL,
          imageType: imageType,
          docID: docID,
        });
      } else {
        navigation.navigate("HomeScreen", {
          routeStatus: Math.random().toString(36).substr(7),
        });
      }
    }}
  >
    <Text style={styles.buttonText}>
      {imageType === "OCT" ? "CLASSIFY AMD" : "GRADE SEVERITY"}
    </Text>
  </TouchableOpacity>
) : (
  <View
    style={{ height: Dimensions.get("window").height * 0.182 }}
  />
)

<Touchableopacity
  {...status && { style: styles.buttonContainerBottom }}
  onPress={() => {
    navigation.navigate("GetImageScreen", {
      imageType: imageType,
      docID: docID,
    });
  }}
>
  <Text style={!status ? styles.buttonText : styles.textButton}>
    Recheck
  </Text>
  </Touchableopacity>
</View>
<SafeAreaView>
  </SafeAreaView>
</ImageBackground>
```

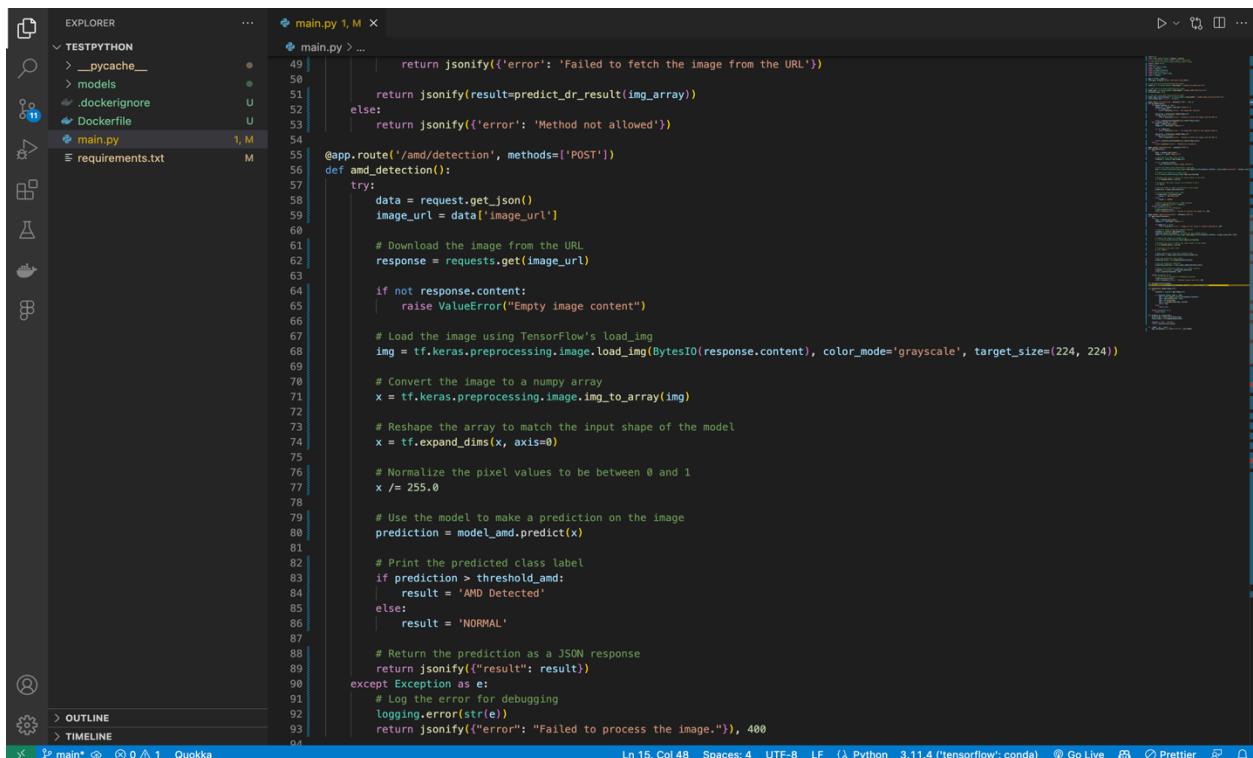
Figure 2- Frontend Implementation

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://192.168.8.158:19000')
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
47         img_array = preprocess_image(image_url)
48         if img_array is None:
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
```

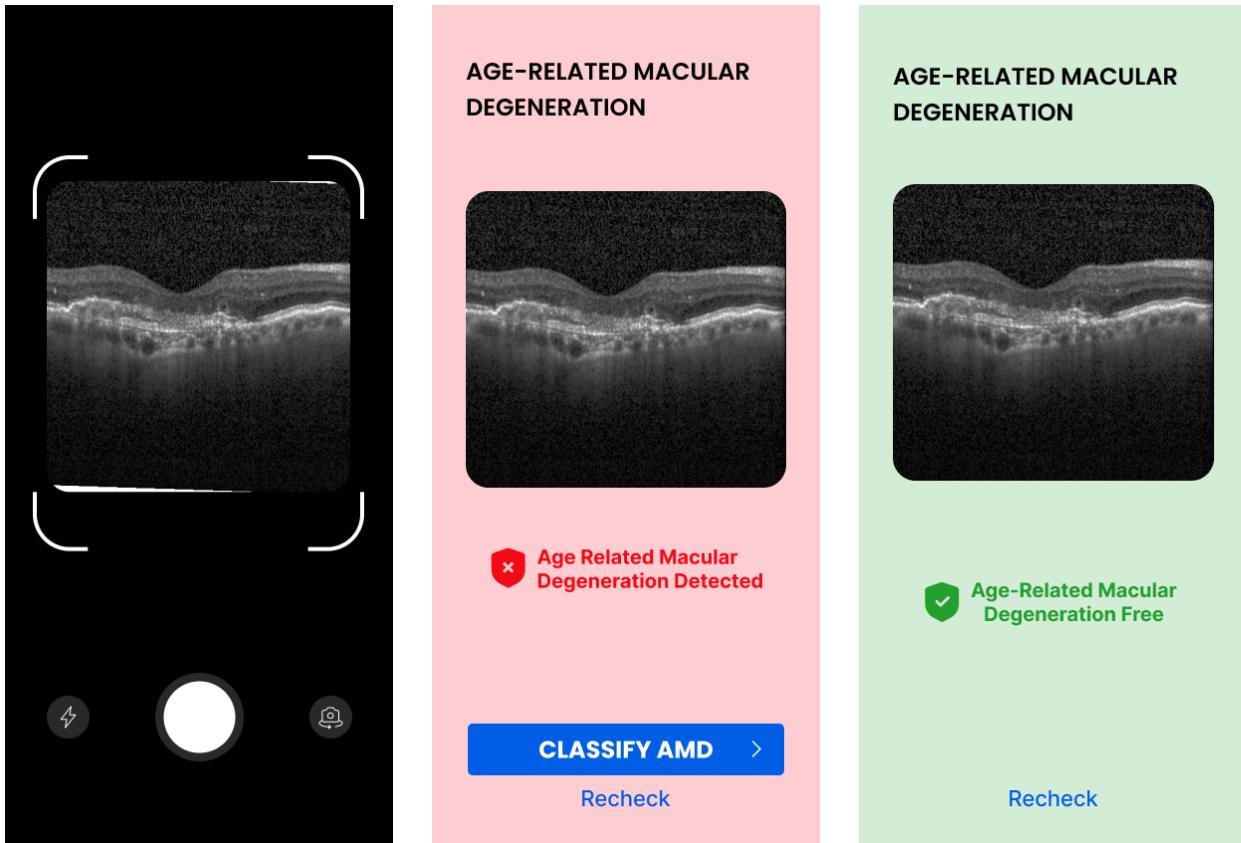
Figure 3 - Backend Implementation



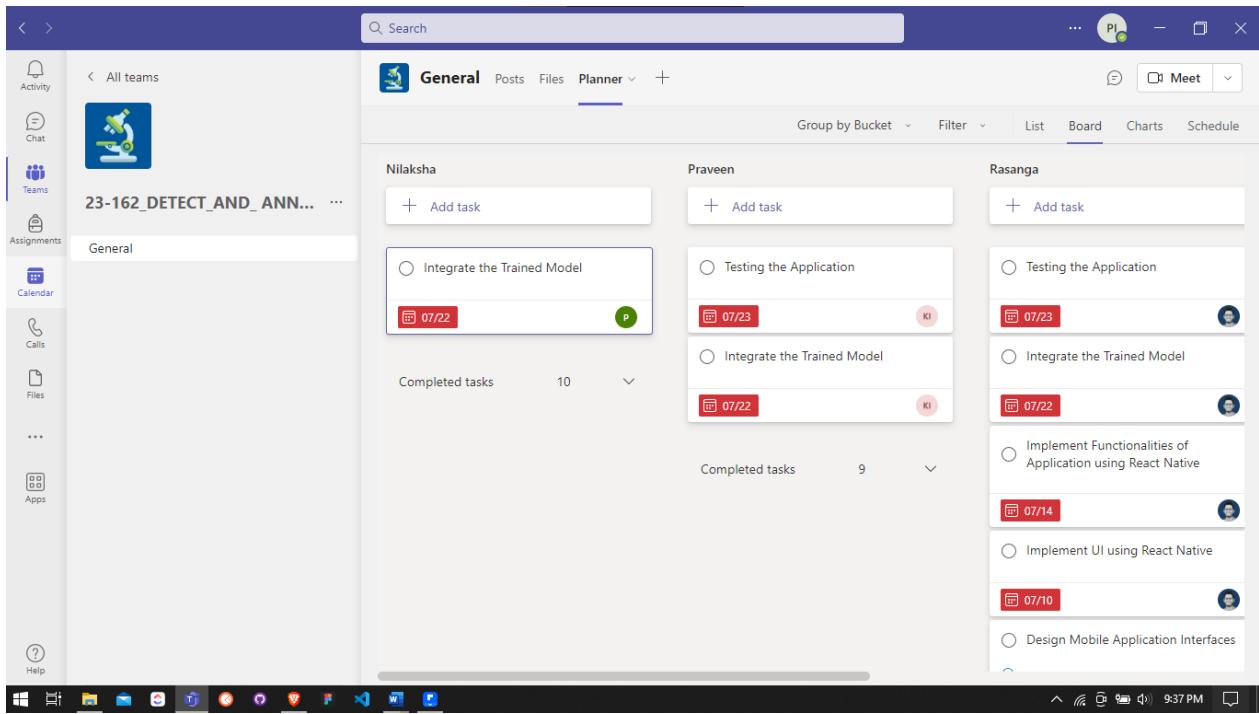
```
1         return jsonify({'error': 'Failed to fetch the image from the URL'})
2
3     else:
4         return jsonify(result=predict_dr_result(img_array))
5
6     else:
7         return jsonify({'error': 'Method not allowed'})
8
9     try:
10        data = request.get_json()
11        image_url = data['image_url']
12
13        # Download the image from the URL
14        response = requests.get(image_url)
15
16        if not response.content:
17            raise ValueError("Empty image content")
18
19        # Load the image using TensorFlow's load_img
20        img = tf.keras.preprocessing.image.load_img(BytesIO(response.content), color_mode='grayscale', target_size=(224, 224))
21
22        # Convert the image to a numpy array
23        x = tf.keras.preprocessing.image.img_to_array(img)
24
25        # Reshape the array to match the input shape of the model
26        x = tf.expand_dims(x, axis=0)
27
28        # Normalize the pixel values to be between 0 and 1
29        x /= 255.0
30
31        # Use the model to make a prediction on the image
32        prediction = model_amd.predict(x)
33
34        # Print the predicted class label
35        if prediction > threshold_amd:
36            result = 'AMD Detected'
37        else:
38            result = 'NORMAL'
39
40        # Return the prediction as a JSON response
41        return jsonify({"result": result})
42    except Exception as e:
43        # Log the error for debugging
44        logging.error(str(e))
45        return jsonify({"error": "Failed to process the image."}), 400
46
```

Figure 4 - Backend Implementation

1.3 Mobile App UIs



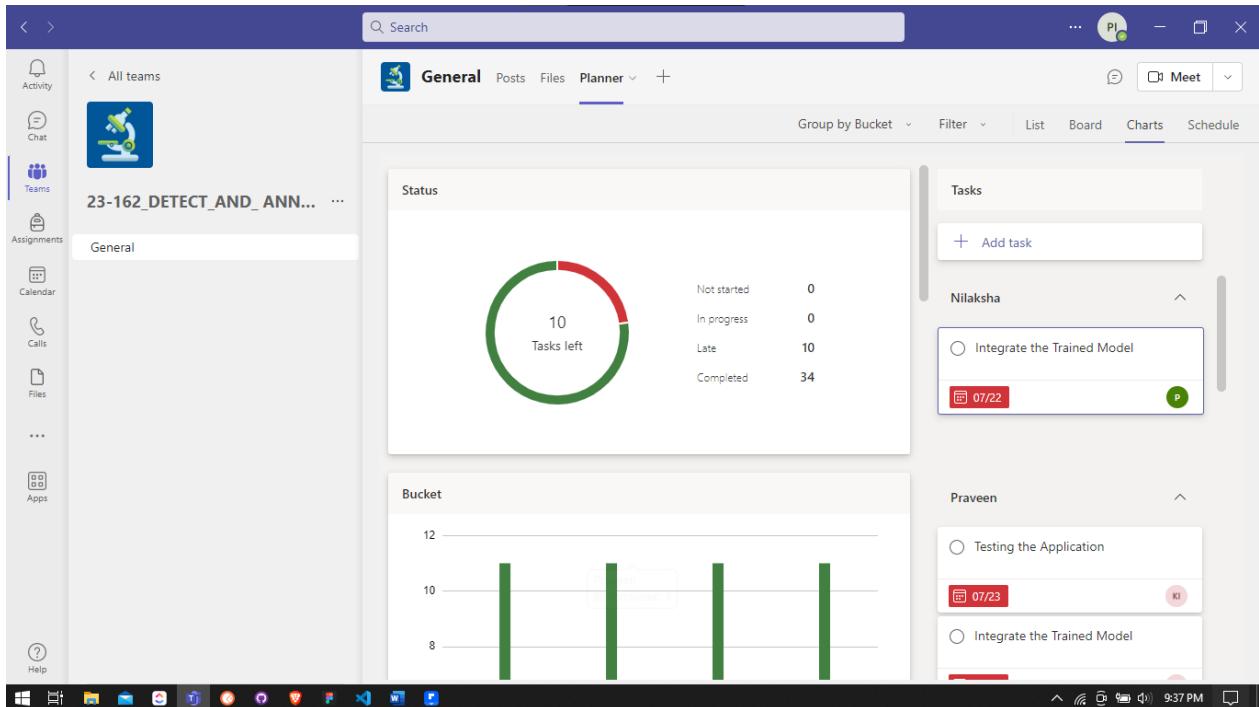
2. Project View



The screenshot shows the Microsoft Planner Task List View. The interface is divided into three main columns, each representing a team member: Nilaksha, Praveen, and Rasanga. Each column has a header with a 'General' tab and a 'Planner' tab. Below the headers are sections for 'Completed tasks' and a list of tasks with due dates and assignees.

| Team Member | Task | Due Date | Assignee |
|-------------|---|----------|----------|
| Nilaksha | Integrate the Trained Model | 07/22 | P |
| | Testing the Application | 07/23 | KI |
| Praveen | Integrate the Trained Model | 07/22 | KI |
| | Testing the Application | 07/23 | KI |
| Rasanga | Testing the Application | 07/23 | KI |
| | Integrate the Trained Model | 07/22 | KI |
| | Implement Functionalities of Application using React Native | 07/14 | KI |
| | Implement UI using React Native | 07/10 | KI |
| | Design Mobile Application Interfaces | 07/10 | KI |

Figure 6 – Planner - Task List View



The screenshot shows the Microsoft Planner Chart View. The interface includes a 'Status' section with a donut chart showing the distribution of tasks (10 tasks left), and a 'Bucket' section with a bar chart showing task distribution across four buckets. On the right, there are sections for individual team members (Nilaksha, Praveen) showing their tasks and due dates.

| Status | Count |
|-------------|-------|
| Not started | 0 |
| In progress | 0 |
| Late | 10 |
| Completed | 34 |

| Bucket | Count |
|--------|-------|
| 12 | 1 |
| 10 | 1 |
| 8 | 1 |
| 6 | 1 |

Figure 7 - Planner - Chart View

The screenshot shows the Microsoft Planner interface in 'Schedule' view for the month of July 2023. The calendar is grouped by bucket, with tasks for different team members. The tasks are as follows:

- Nilaksha:** Design ... (July 10)
- Praveen:** Implement ... (July 14)
- Rasanga:** Design a... (July 16), Train the ... (July 17)
- Chamod:** Implement ... (July 22)
- Completed:** Testing the ... (July 23)

The calendar shows the following days of the week: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. The days of the month are: 25, 26, 27, 28, 29, 30, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31.

Figure 8 - Planner - Schedule View

3. Gantt chart

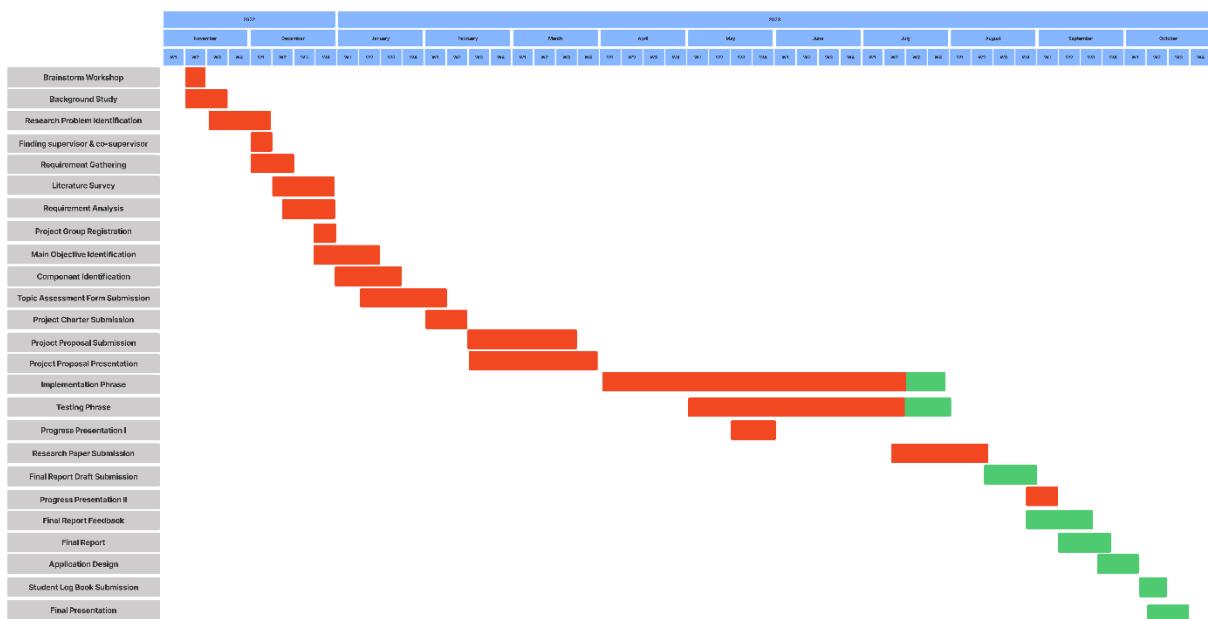


Figure 9 - Gantt Chart

4. Screenshots of Conversations and Calls - Microsoft Teams

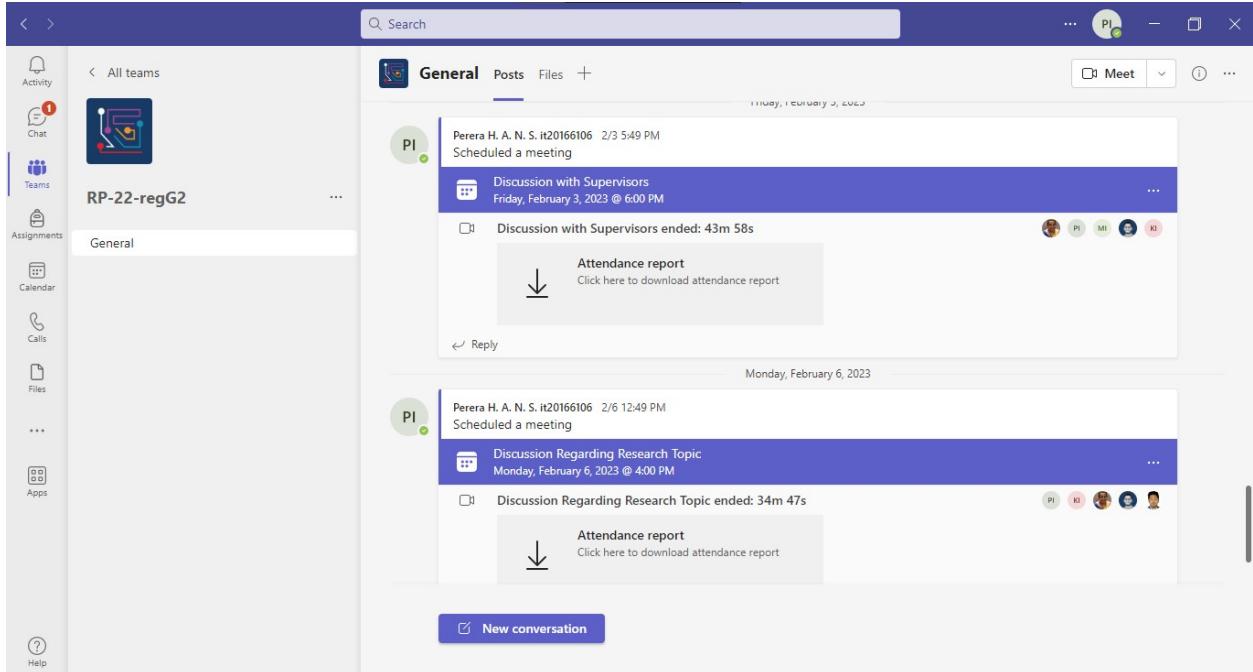


Figure 10 - MS Teams Channel

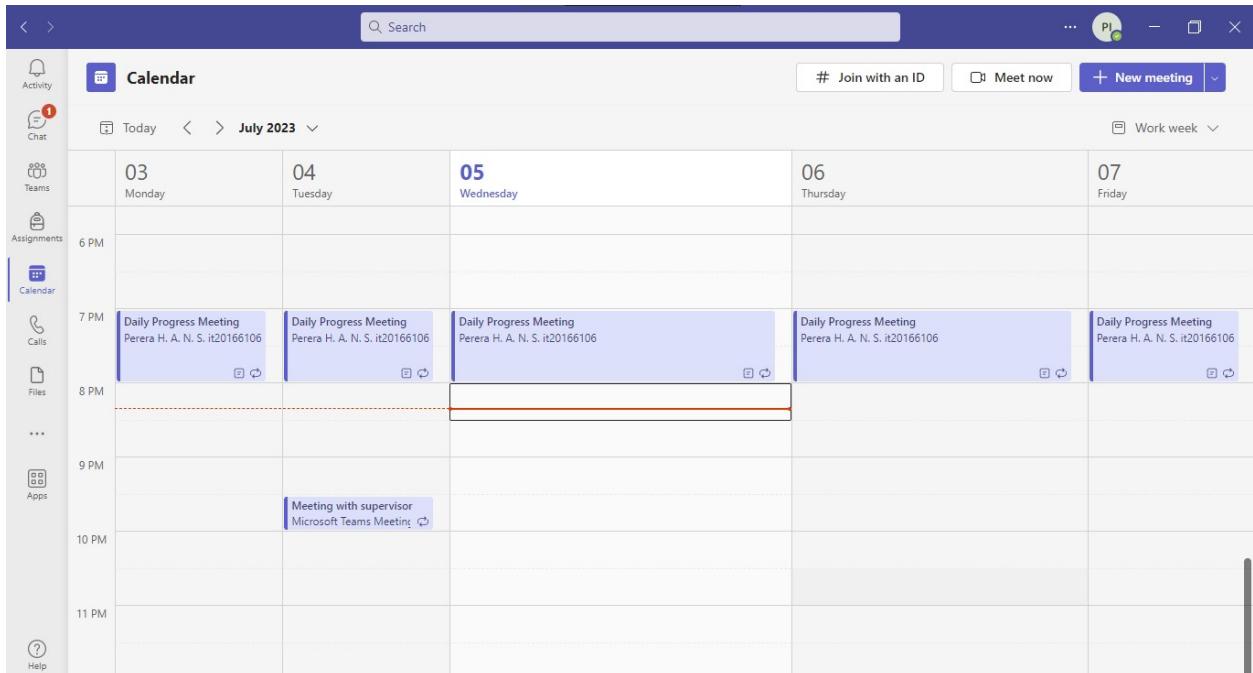


Figure 11 -Scheduled Meetings

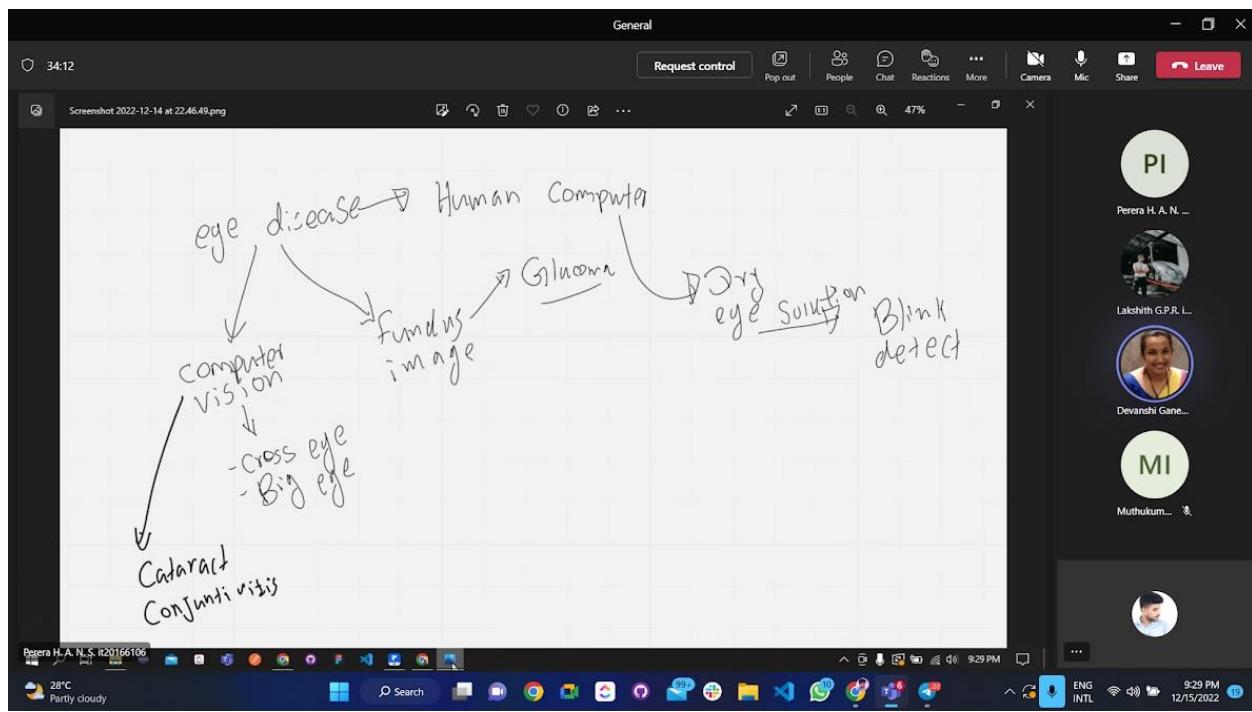


Figure 12 -Meetings with Supervisors

