UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER (UOW)

BEng (Hons) in Software Engineering

5COSC021C: Software Development Group Project

Module Leader : Banuka Athuraliya

Detection of Early Prognosis of Diabetic Retinopathy,Macular Edema and Glaucoma

Demo Video link- Demo Video

Group Name: Acetech Labs
Project Name : OculoGuard
Acetech Labs Members:

Student Name	UOW No:	IIT Student ID:
Anjula Jayashanka	w18100026	20200245
Maheswaram Harrisagar	w1810587	2019791
Chamodh Samaranayake	w1810008	20200267
Thidas Jayawardane	w18098145	2019717
Achintha Jayatilake	w1761374	2019530

Declaration Page

ALL OF OUR GROUP MEMBERS DECLARE THAT THIS ASSIGNMENT IS AN ORIGINAL WORK SUBMITTED BY THE MEMBERS OF THE **Acetech Labs-SE-46.** HAVE ACTIVELY CONTRIBUTED TO PREPARE THE REPORT OF **OculoGuard** , ANY OTHER WORK OF A SIMILAR NATURE HAS BEEN PROPERLY REFERENCED.

Abstract

Over 450 million people suffer from diabetes mellitus around the world,where nearly half of the cases remain undiagnosed.The amount of diabetes patients are expected to rise exponentially in the next two decades.Diabetic Eye Diseases(DED) are one of the dominant complications of diabetes which leads to a wide range of issues from permanent vision impairment to blindness.Diabetic Retinopathy,Diabetic Macular Edema and Glaucoma are three of the predominant forms of DEDs accounting for more than 60 percent of the cause of blindness in the working population.Early detection is crucial in the next upcoming decade as the world population faces a vast array of challenges in the form of complications,lack of resources and the exponential rise of DED with a dangerous emphasis on low and middle income countries.The proposed initiative, named Oculo-Guard is a software solution that accurately detects the Early Prognosis of Diabetic Retinopathy,Diabetic Macular Edema and Glaucoma that can be deployed to enhance earlier detection of DEDs cutting costs,saving resources and reducing complications and casualties.

Subject Descriptors:

Machine Learning

Learning Paradigm

Supervised Learning

Supervised learning by classification

Key Words: Machine Learning,Deep Learning,k-nearest neighbors,Support Vector Machine

Acknowledgement

Team Acetech Labs would like to express our thanks and gratitude to a number of people First and foremost our thanks and heartfelt gratitude goes to Mr.Banu Athuraliya,our module coordinator, Mrs.Janani Harischandra ,our mentor and the rest of the module team.Their support,feedback and guidance was crucial for this project to be a possibility

We would like to thank each and every member of Acetech Labs who have been supportive and understanding of each other and helping each other succeed through challenges and hard times.

Special thanks goes to the domain and legal experts and survey respondents who played a key role in making this report a success by giving their valuable feedback.

Last but not least we would like to thank our colleagues outside the team and our family members that lended their support at any time we needed assistance.

Acetech Labs,

SE 2nd year,

Group - 46

Table of Contents

List of Tables

List of Figures

Abbreviations table

Abbreviation	Explanation
ML	Machine Learning
DL	Deep Learning
KNN	k-nearest neighbors
SVM	Support Vector Machine
AI	Artificial Intelligence
CNN	Convolutional neural network
DR	Diabetic Retinopathy
DME	Diabetic Macular Edema
DED	Diabetic Eye Diseases

Chapter1:Implementation

1.1 Chapter Overview

The previous SRS report we discussed the design and architecture of the OculoGurd software. This chapter will discuss in detail the programming languages used to implement the project, technologies, frameworks, and libraries. A detailed discussion of the elements of the OculoGuard prototype describes how each component was developed, its intended use, relevant code snippets and screenshots, and the challenges and solutions encountered in implementing each component.

1.2 Technology selections

NextJS and React
React is a front-end Library used to build single page web applications. Introduced by Facebook it has gained the most popularity throughout these years. React uses Javascript ES6 which has a very less learning curve. Next JS is like an extension for React and mostly used in production environments. It has built in Router which follows the file hierarchy. Server-side rendering and static site generation are the features which highlight Next JS as a production grade framework. With the newly introduced API routes this framework can be considered as a full stack framework

Flutter
Flutter is a UI library developed by Google for cross platform development. Introduced in 2018 since then it has become one of the most loved frameworks among the developers. Flutter Follows the same design pattern as React but with class widgets. Learning curve is very low due to the continuous support from the flutter team and excellent documentation. Also the integration with different technologies like cloud is much easier than following native approach. Sound Null Safety and continuous documentation along the API flutter is the best framework one can choose to implement hybrid applications.

AWS Amplify
Amazon Web Services are among the pioneers in the cloud computing sector. Amplify is basically a wrapper introduced to sum up the 200 plus different aws cloud services. It’s easy to integrate with different tech stacks like React and Flutter and at the same time provide flexibility to be used as a single backend for multiple applications. From Authentication to server less deployment of applications, amplify provides and wraps up the backend which every developer needs.

TailwindCSS
Tailwinds is utility based css framework which accelerates the ui development time. No need to use css files, only need to specify classes for the components.

NumPy
NumPy, Python's most fundamental package, is a general-purpose array configuration package. It includes high-performance multitasking array objects as well as tools for working with them. NumPy is a fast directory for standard multidimensional data.

Flask
This web application was created using the Flask microframework. Flasks were used due to their ease of use and simple structure. Flask is much lighter and more explicit than many web frameworks.

TensorFlow
TensorFlow is an open-source machine learning platform that runs from end to end. It can be used for a variety of tasks, but it is most used for training and guessing deep neural networks. TensorFlow is an open-source library developed by the Google Brain team.

1.3 Implementation of the data science component

Since our system mainly depends on image processing we are dealing with images with pixels and random forest is basically depend on decision trees so since that we can't do image processing with random forest, so we need a Convolutional neural network(CNN) and for that we have two options one is we have to train a model or else we have to go with transfer learning. So we went with mobilenet lightweight,easy to train and Optimize Convolutional neural networks(CNNs) best suited for mobile and embedded vision applications. We input one layer on the input side to input data then we chopout some couples of layers from the bottom and insert our Calcification layer for that.We didn't go with an own model because lack of data

Datasets

Collected datasets contain images of glaucoma from three different sources;
MESSIDOR
Bin Rushed
Magrabi Eye center
Obtained data sets from Hamilton Eye Institute Macular Edema which contains DME images.

ML Model
For the Model 1 ,we used 4165 fundus images as jpg format and all images represent 7 classes respectively dr_Mild, dr_Moderate, dr_Poliferate, dr_Severe, Glaucoma_Positive, Macular_Edema and Normal.

For the Model 2 ,we used 4165 fundus images as jpg format and all images represent 4 classes respectively DR, Glaucoma_Positive ,Macular_Edema and Normal.

For the Model 3 ,we used 1447 fundus images as jpg format and all images represent 4 classes respectively dr_Mild, dr_Moderate, dr_Proliferate and Normal.

Importing Required environments

```
import numpy as np
np.random.seed(2017)

import os
import time
from keras.preprocessing import image
from keras.applications.imagenet_utils import preprocess_input
from keras.layers import Input
from keras.models import Model
from keras.models import load_model
import tkinter as tk
from tkinter import filedialog

import os
import itertools
import shutil
import random

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Dense, Activation,Reshape, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.applications import imagenet_utils
 from tensorflow.keras.models import load_model

 from sklearn.metrics import confusion_matrix
 from sklearn.metrics import classification_report

 import numpy as np
 import pandas as pd
 from glob import glob
 import shutil
 import matplotlib.pyplot as plt
 %matplotlib inline
 import matplotlib.pyplot as plt
 print(tf.__version__)
 import cv2
```

Figure 1:Snippets for importing Environments

Imported Keras, Tensorflow ,Open-CV,Sklearn, tkinter and numpy for implementing machine learning Environment.

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Figure 2:Code for plotmatrix function

This code function prints and plots the confusion matrix of the representative model. Normalization is also applied. We divide each row element by the total of the entire row to produce the normalized version. The final normalized matrix will show us the percentage since each row contains the entire number of real values for each class label.

```
os.chdir('E:\OculoGuard_Mobile_app-main\Backend\Model 02\Data')
if os.path.isdir('\Data') is False:
    os.mkdir('train')
#     os.mkdir('valid')
    os.mkdir('test')
    filenames = ["DR","Glaucoma_Positive","Macular_Edema","Normal"]
    for i in filenames:
        shutil.move(f'{i}', 'train')
#         os.mkdir(f'valid/{i}')
        os.mkdir(f'test/{i}')

#         valid_samples = random.sample(os.listdir(f'train/{i}'), int(len(os.listdir(f'train/{i}'))*0.2))
#         for j in valid_samples:
#             shutil.move(f'train/{i}/{j}', f'valid/{i}')

        test_samples = random.sample(os.listdir(f'train/{i}'), int(len(os.listdir(f'train/{i}'))*0.3))
        for k in test_samples:
            shutil.move(f'train/{i}/{k}', f'test/{i}')
os.chdir('..')
```

Figure 3:Code for dividing data sets as test and train

All Dataset were divided into 2 folders as “train” and “test”

```
train_path = 'E:/OculoGuard_Mobile_app-main/Backend/Model 02/Data/train'
# valid_path = 'E:\leaf-disease-classification-main\Research - noise\Dataset/valid'
test_path = 'E:/OculoGuard_Mobile_app-main/Backend/Model 02/Data/test'

# train_batches = ImageDataGenerator(
#                                 preprocessing_function=tf.keras.applications.mobilenet.preprocess_input,
#                                 rotation_range=10,
#                                 width_shift_range=0.1,
#                                 height_shift_range=0.1,
#                                 shear_range=0.15,
#                                 zoom_range=0.1,
#                                 channel_shift_range=10.,
#                                 horizontal_flip=True
#                                 ).flow_from_directory(
#                                                     directory=train_path,
#                                                     target_size=(224,224),
#                                                     batch_size=16
#                                                    )

train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
    directory=train_path, target_size=(224,224), batch_size=1000 )

# valid_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
#     directory=valid_path, target_size=(224,224), batch_size=795 )

test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
    directory=test_path, target_size=(224,224), batch_size=1248 , shuffle=False)
```

Figure 4:Train path code

Added train path and called trained baches

```
train_batches.class_indices
```

```
mobile = tf.keras.applications.mobilenet.MobileNet()
mobile.summary()
print(len(mobile.layers))
```

```
x = mobile.layers[-6].output

GVP =  tf.keras.layers.GlobalAveragePooling2D()(x)
output = Dense(units=4, activation='softmax')(GVP)

model_1 = Model(inputs=mobile.input, outputs=output)

model_2 = Model(inputs=mobile.input, outputs=output)

for layer in model_1.layers[:-23]:
    layer.trainable = False

for layer in model_2.layers[:-23]:
    layer.trainable = False

model_1.summary()
print(len(model_1.layers))
```

Figure 5:Code for printing model summary and class indices

Model was trained with 40 epochs to get better accuracy.

```
## ====================================================================
#        Train with the Noised set and noised validation and Test
#=====================================================================
tf.random.set_seed(959)
class_weight = {0: 1/5,
                1: 1,
                2: 2,
                3: 1/5}
model_1.compile(optimizer=Adam(lr=0.00003), loss='categorical_crossentropy', metrics=['accuracy'])
history = model_1.fit(
    train_batches,
    epochs=40,
    steps_per_epoch=len(train_batches),
    validation_data = test_batches,
    verbose = 1,
    validation_steps=len(test_batches),
    class_weight=class_weight)

model_1.save('models/model.h5')
```

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```

Figure 6:Code for model history

This code snippet summarizes and prints model history. The accuracy and validation loss will be printed in a graph when this code is implemented.

```
#model = load_model('models/model.h5')

test_labels = test_batches.classes
predictions = model_1.predict(x=test_batches, steps=len(test_batches), verbose=1)
cm = confusion_matrix(y_true=test_labels, y_pred=predictions.argmax(axis=1))
```

```
train_batches.class_indices
```

```
y_pred_bool = np.argmax(predictions, axis=1)

print(classification_report(test_labels, y_pred_bool))
```

```
test_batches.class_indices

cm_plot_labels = [i for i in train_batches.class_indices]
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

```
model_1.save('models/model_final.h5')
```

```
plt.hist(y_pred_bool, bins = 7)
plt.show()
```

Figure 7:Code for saving model and print histogram

Models were saved with data after training and saved as h5 files.(Data files)
And it prints histograms with model classes.

Prediction

```
dict={0:'Glaucoma_Positive',1:'Macular_Edema',2:'Normal',3:'dr_Mild',4:'dr_Moderate',5:'dr_Proliferate',6:'dr_Severe'}
```

```
model_1=('C:/Users/Harrisagar/Desktop/FinalClassification/Final_Prediction/Model 01 New/models/model.h5')
```

Figure 8:Code for prediction

Added a dictionary with values as, {'Glaucoma_Positive': 0, 'Macular_Edema': 1,'Normal': 2, 'dr_Mild': 3, 'dr_Moderate': 4, 'dr_Proliferate': 5,'dr_Severe': 6} Since the train batches were implemented with these class indices

```
mobile = tf.keras.applications.mobilenet.MobileNet()
# mobile.summary()
# print(len(mobile.layers))

x = mobile.layers[-6].output

GVP =  tf.keras.layers.GlobalAveragePooling2D()(x)
output = Dense(units=7, activation='softmax')(GVP)

model_1 = Model(inputs=mobile.input, outputs=output)

model_2 = Model(inputs=mobile.input, outputs=output)

for layer in model_1.layers[:-23]:
    layer.trainable = False

for layer in model_2.layers[:-23]:
    layer.trainable = False

# model_1.summary()
# print(len(model_1.layers))

model_1.compile(optimizer=Adam(lr=0.0003), loss='categorical_crossentropy', metrics=['accuracy'])
model_1.load_weights('C:/Users/Harrisagar/Desktop/FinalClassification/Final_Prediction/Model 01 New/models/model.h5')
```

Figure 9:Code to load model weights

```
root = tk.Tk()
root.withdraw()
img_path = filedialog.askopenfilename()
```

```
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
# print('Input image shape:', x.shape)
```

```
x = tf.keras.applications.mobilenet.preprocess_input(x)
pr=model_1.predict(x)
pr1=np.argmax(pr)
# print('Result=',dict[pr1])
```

Figure 10:Code to get input image

```
if dict[pr1] == ("Glaucoma_Positive"):
    print("Glaucoma_Positive")
elif dict[pr1] ==("Macular_Edema"):
    print("Macular_Edema")
else:
    print("No Disease")
```

Figure 11:Code for printing prediction results

Prints disease when the image inputs.

1.4 Implementation of the backend component

Web Application

The project backend implementation process has been broken down into 3 feature which are,

- Login process
- Taking input files to Machine learning
- Taking output from ml and passing in to the frontend

Data Modeling and Database Design

For the designing of the database, an Entity diagram was prepared and accordingly a schema is created using the GraphQL query language.

```
type S3Object @model @auth(rules: [{allow: public}, {allow: private}, {allow: groups, groups: ["us-east-1_oEs5LyGNF_Google"], operations: [read, creat
  id: ID!
  bucket: String
  region: String
  key: String
}

enum PredictionTypes {
  PROGNOSIS
  DIABETICRETINOPATHY
  MACULOEDEMA
  GLAUCOMA
}

type Record @model @auth(rules: [{allow: public}, {allow: private}, {allow: groups, groups: ["us-east-1_oEs5LyGNF_Google"], operations: [read, create,
  id: ID!
  userID: ID! @index(name: "byUser")
  details: String
  diseaseType: PredictionTypes
  isVerified: Boolean
  verifiedBy: String
  S3Object: S3Object @hasOne
}

type User @model @auth(rules: [{allow: public}, {allow: private}, {allow: groups, groups: ["us-east-1_oEs5LyGNF_Google"], operations: [read, create, u
  id: ID!
  name: String
  age: Int
  ophthalmologistID: ID @index(name: "byOphthalmologist")
  Records: [Record] @hasMany(indexName: "byUser", fields: ["id"])
  address: String
}

type Ophthalmologist @model @auth(rules: [{allow: public}, {allow: private}, {allow: groups, groups: ["us-east-1_oEs5LyGNF_Google"], operations: [read,
  id: ID!
  name: String
  Users: [User] @hasMany(indexName: "byOphthalmologist", fields: ["id"])
  location: String
}
```

Figure 12:code for Data Modeling and Database Design

1.5 Implementation of the front end component

Mobile Application

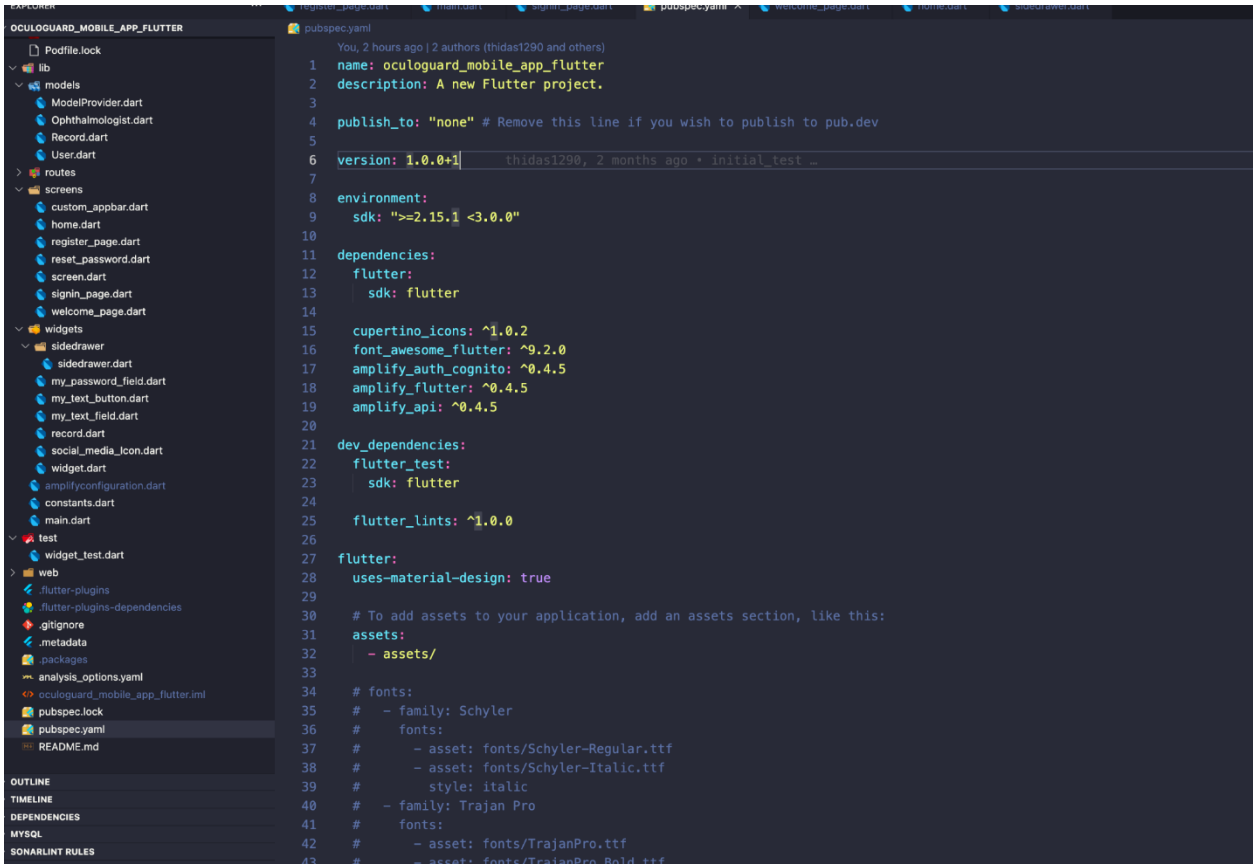


Figure 13:Code for amplified libraries

For the implementation, amplified libraries
are defined and added as dependencies for the project These libraries are responsible for the User Authentication and for the use of GRPHQL API.

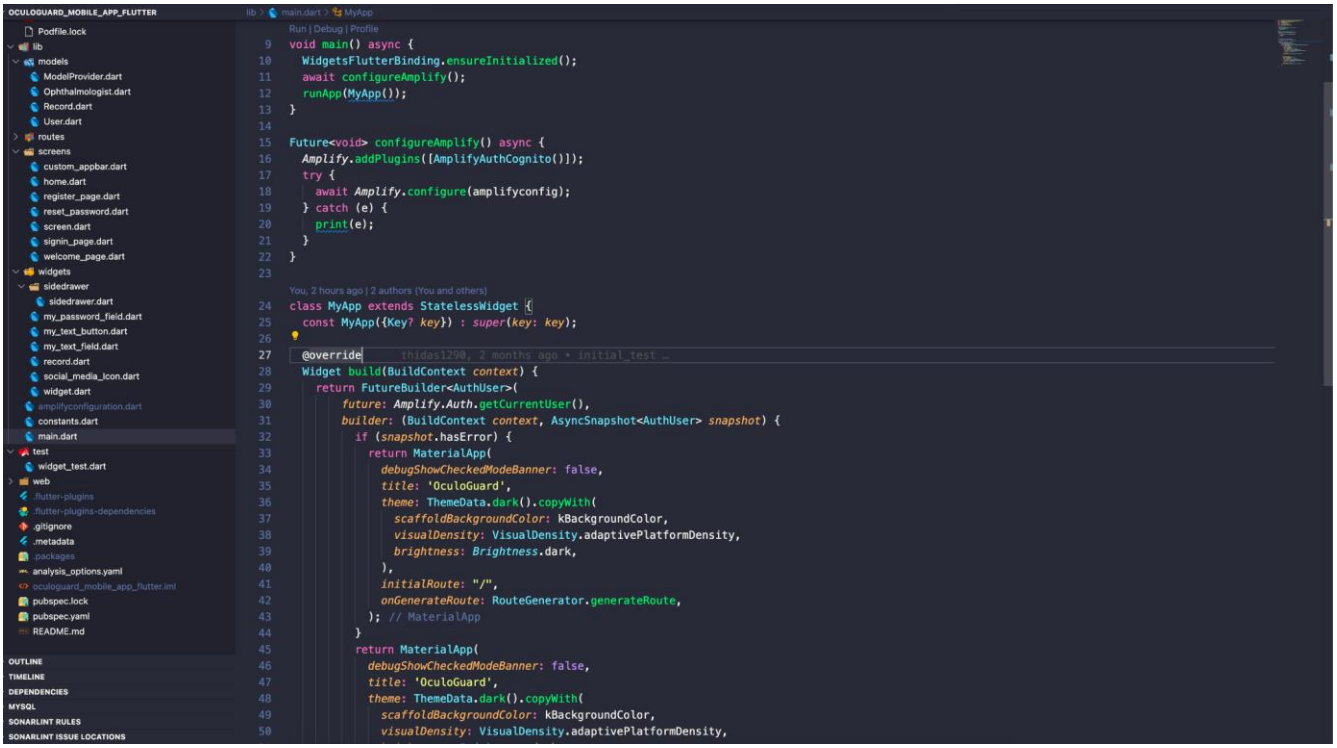


Figure 14:Code for Main widget

Next the main method has been initialized along with initialization of Amplify dependencies.
MyApp widget is the first in the widget tree which defines the MaterialApp Widget. Inside it a future builder has been initialized to check whether there's a user who has already logged in, if so then the app will navigate directly to the home page and if not, the user has to go through sign and signup process. Future builder is used because checking the use state using Amplify.Auth will take around some time and these are regarded as futures or asynchronous functions, therefore use of future builder is mandatory.

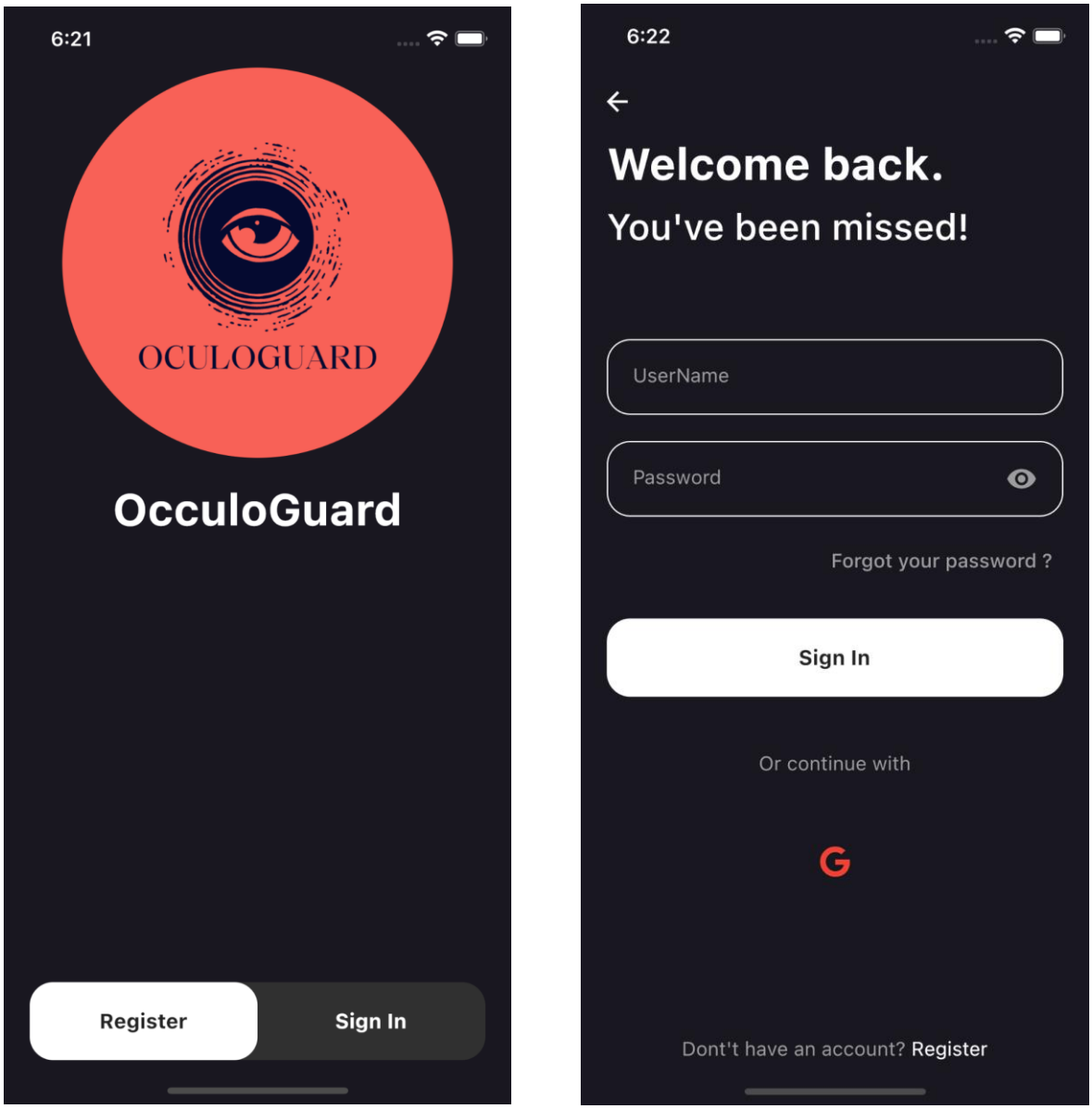


Figure 15:Login page

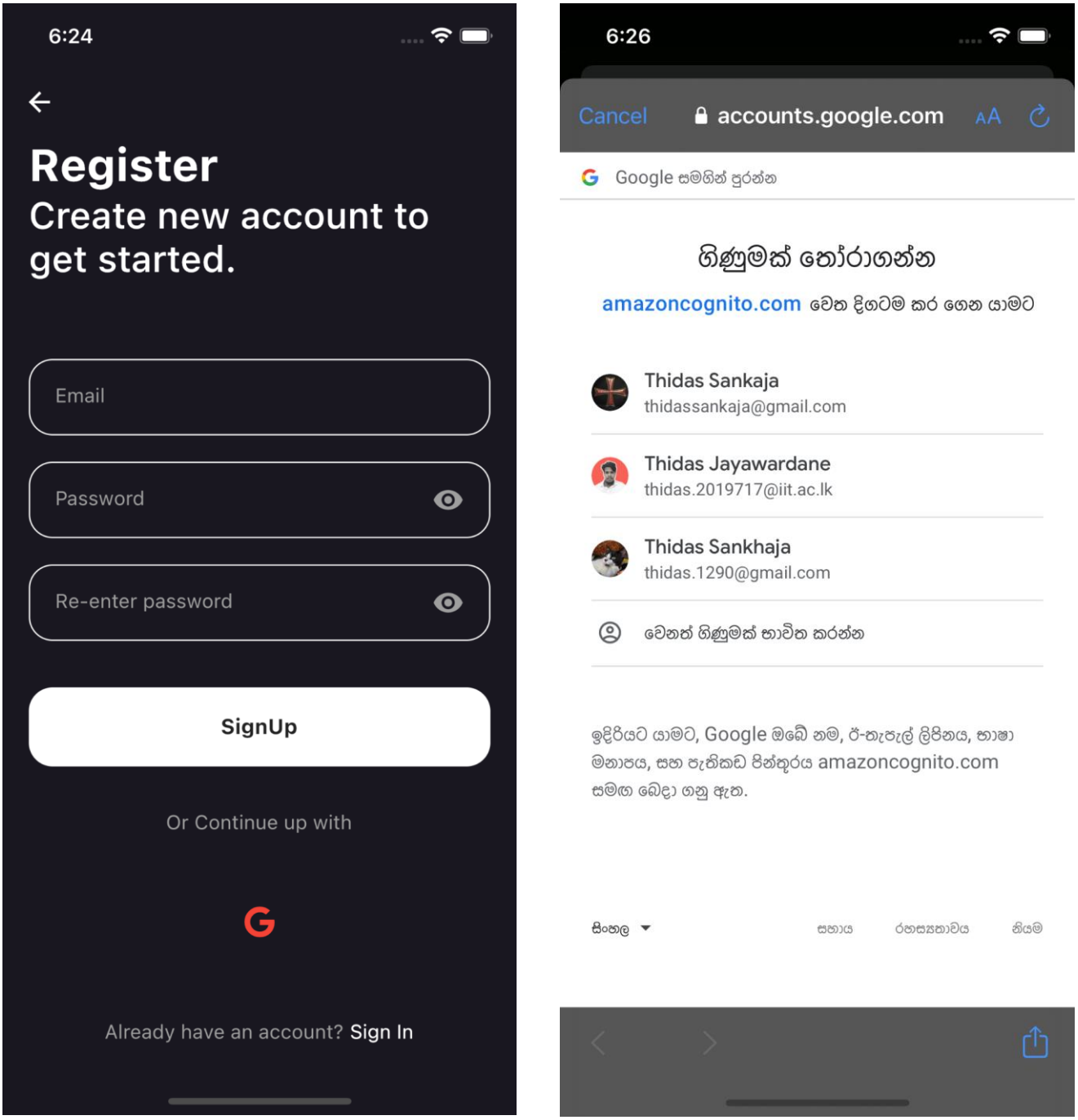


Figure 16:Sign up page

Web Application

For the web application Next Js was used with React, Next js is basically like a super set of react with enhanced functionalities. For the implementation React hooks were used to manipulate and design the ui rendering.

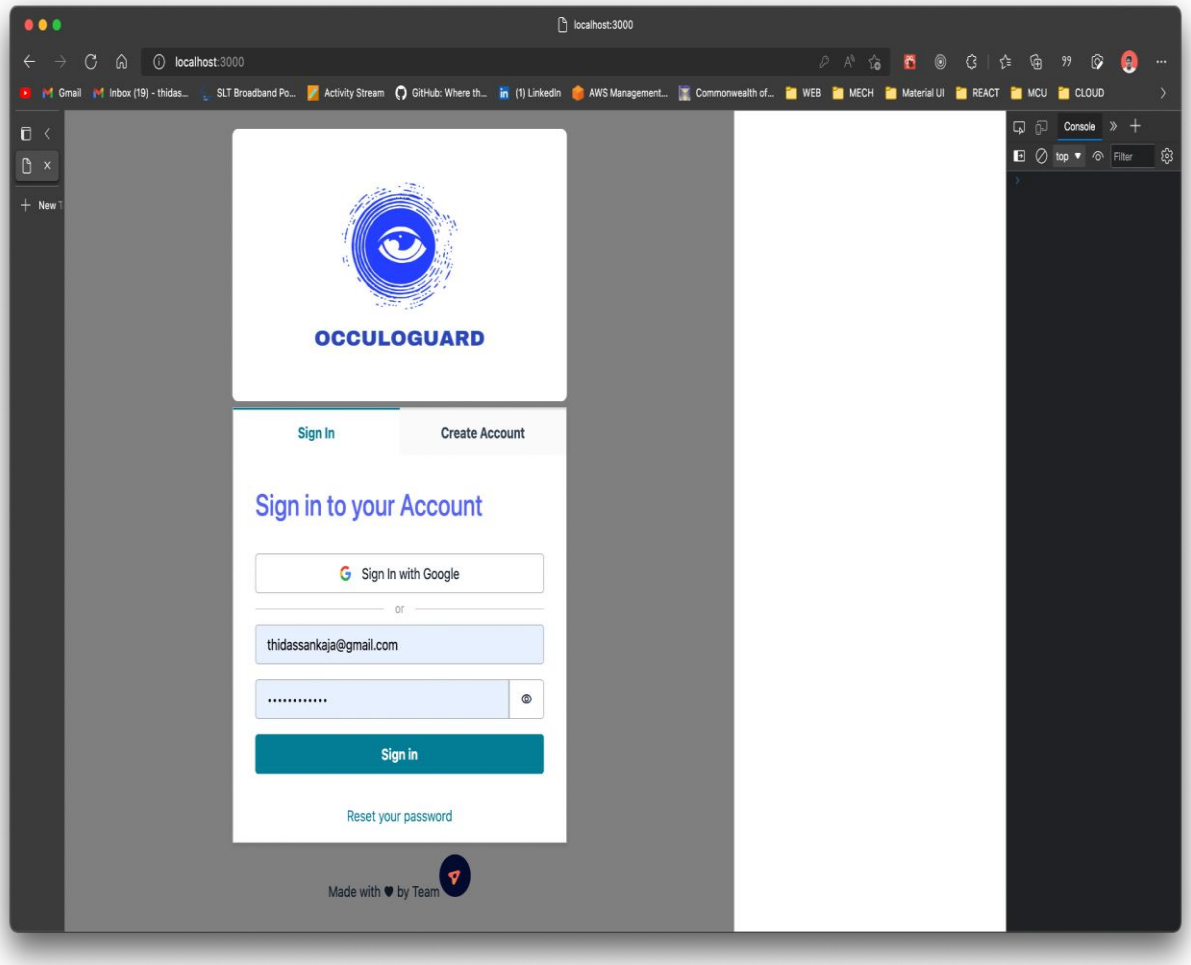


Figure 17:web app sign up page

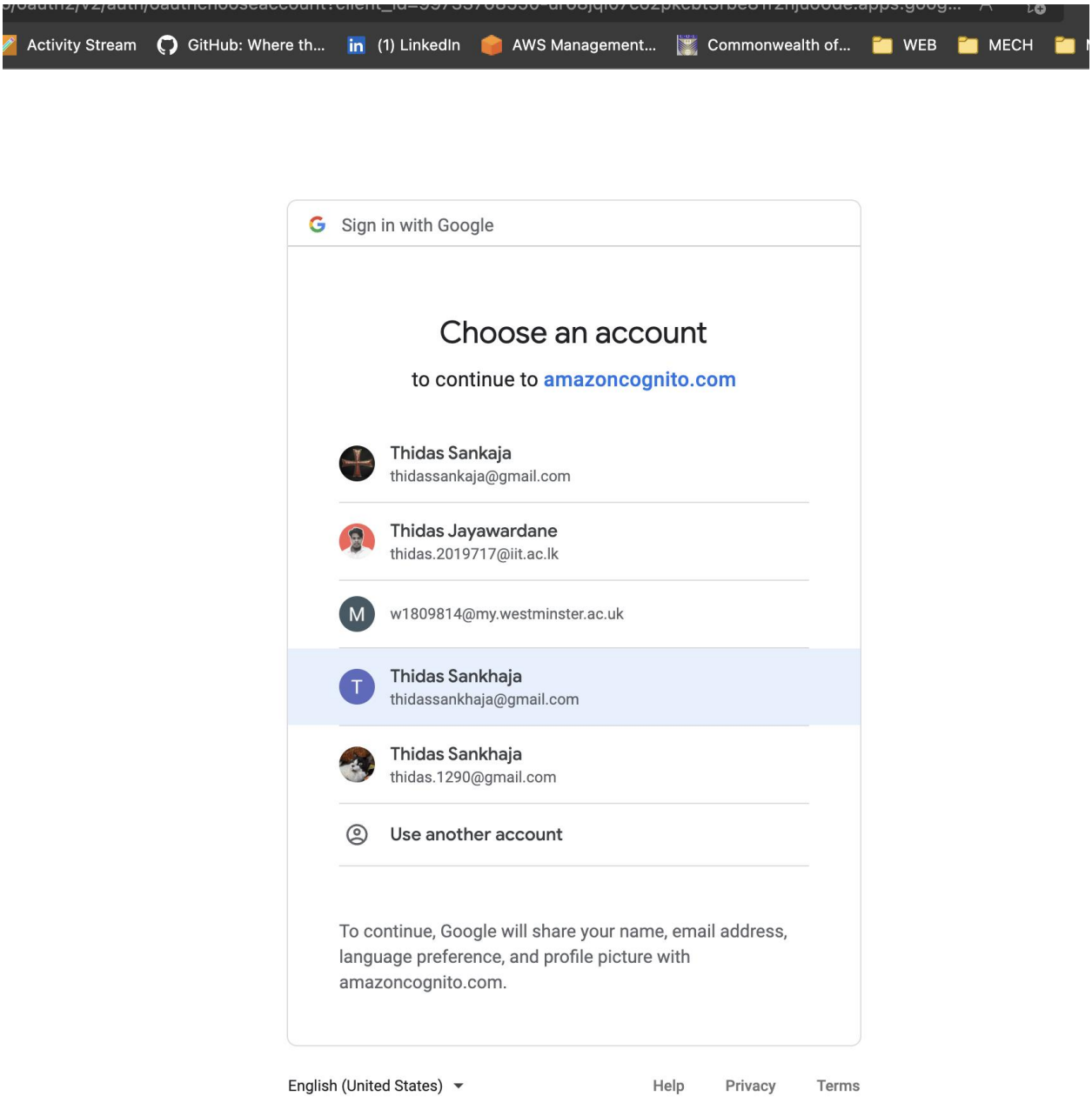


Figure 18:Account choosing page

app.jsx file is the main file in NEXT js where component tree starts in there Amplify export were configured and initialized which will be used in the entire project to communicate to the backend database. Also ssr (server side rendering) has also been initialized as in some routes before the client side rendering rendered code from server side is fetched to the client(next js in this case). This is really effective and SEo friendly since rendered code can directly interact.

Inside the MyApp function the main component is wrapped around the Layout of the web application which holds the navigation for every page inside the component.

Then when exporting the component from the page it's again wrapped with Amplify Authenticator in order to add authentication around the web application and also some custom Sign in header and signup headers were added in order to keep up the design. Since the implementing custom sign up login design and logic

```
You, 2 days ago | 1 author (You)
1  import "../styles/globals.css";
2  import awsExports from "../src/aws-exports";
3  import { Amplify } from "aws-amplify";
4  import Header from "../components/signupflow/Header";
5  import Footer from "../components/signupflow/Footer";
6  import SignInHeader from "../components/signupflow/SignInHeader";
7  import SignInFooter from "../components/signupflow/SignInFooter";
8  import { Router } from "next/router";      You, 2 days ago • default …
9  import { withAuthenticator } from "@aws-amplify/ui-react";
10 import "@aws-amplify/ui-react/styles.css";
11 import Layout from "../components/layout/Layout";
12
13 Amplify.configure({ ...awsExports, ssr: true });
14
15 function MyApp({ Component, pageProps }) {
16   return (
17     <Layout>
18       <Component {...pageProps} />
19     </Layout>
20   );
21 }
22
23 export default withAuthenticator(MyApp, {
24   components: {
25     Header,
26     SignIn: {
27       Header: SignInHeader,
28       Footer: SignInFooter,
29     },
30     Footer,
31   },
32 });
```

Figure 19:Initialization of root component with Authenticator

Next implemented the main home page where the dashboard is implemented

The logic behind is Ophthalmologists can't create new accounts but patients can so when a user logs into the system it checks whether he’s already an Ophthalmologist or a patient by running queries using graphql api in order to determine identity. If all these fail it means the user is a new user and in the dashboard it’ll show to create a record inside the db.

If the user is already a member and a Ophthalmologist then the dashboard renders differently as it will show the number of patients the Ophthalmologist have and unassigned patients in the system. If clicks on a patient he has he can perform eye disease predictions using the API by uploading the required images and can save the report of that in the name of the selected user.

If the logged in user is a patient then only the disease prediction can be directly done in there and can save to the record list.

```
You, 1 hour ago | 1 author (You)
import { useState, useEffect } from "react";
import Head from "next/head";
import { API, Auth } from "aws-amplify";
import { listUsers, getOphthalmologist, getUser } from "../src/graphql/queries";
import Button from "../components/buttons/Button";
import Link from "next/link";
import Card from "../components/card/Card";
import { data } from "autoprefixer";
import HeadDisplay from "../components/HeadDisplay";
import { useRouter } from "next/router";

const defaultAuthMode = "AMAZON_COGNITO_USER_POOLS";
let predictor = "/predictor/";

export default function Home() {
  const [userName, setUserName] = useState(null);
  const [users, setUsers] = useState([]);
  const [assignedUsers, setAssignedUsers] = useState([]);
  const [user, setUser] = useState(null);      You, 1 hour ago • updated compatible
  const [userId, setUserId] = useState(null);
  const router = useRouter();

  useEffect(() => {
    checkOphthalamologist();
    getAllUsers();
    getAssignedUsers();
  }, []);

  // useEffect(() => {}, [user]);

  const checkOphthalamologist = async () => {
    const userInfo = await Auth.currentAuthenticatedUser();
    console.log(userInfo.attributes.sub);
    try {
      const { data } = await API.graphql({
        query: getOphthalmologist,
        variables: {
          id: userInfo.attributes.sub,
        },
        authMode: defaultAuthMode,
```

```
      if (data.getOphthalmologist.name === "") {
        setUserName("NEW");
      } else {
        setUserName(data.getOphthalmologist.name);
      }
      return;
    }
  } catch (e) {
    console.log(e);
  }

  try {
    const { data } = await API.graphql({
      query: getUser,
      variables: {
        id: userInfo.attributes.sub,
      },
      authMode: defaultAuthMode,
    });
    console.log("data = ", data);
    if (data.getUser === null) {
      setUserName("NEW");
    } else {
      setUserName(data.getUser.name);
      setUserId(userInfo.attributes.sub);
    }
  } catch (e) {
    console.log(e);
  }
};

const getAllUsers = async () => {
  try {
    const { data } = await API.graphql({
      query: listUsers,
      authMode: defaultAuthMode,
    });
    setUsers([
      ...data.listUsers.items.filter((user) => {
        return user.ophthalmologistID === null;
      }),
```

```
      authMode: defaultAuthMode,
    });
    setUsers([
      ...data.listUsers.items.filter((user) => {
        return user.ophthalmologistID === null;
      }),
    ]);
  } catch (e) {
    console.log(e);
  }
};

const getAssignedUsers = async () => {
  const currentUser = await Auth.currentAuthenticatedUser();
  try {
    const { data } = await API.graphql({
      query: listUsers,
      authMode: defaultAuthMode,
    });
    setAssignedUsers([
      ...data.listUsers.items.filter((user) => {
        return user.ophthalmologistID === currentUser.attributes.sub;
      }),
    ]);
  } catch (e) {
    console.log(e);
  }
};

if (user === "P") {
  return (
    <div>
      <HeadDisplay>
        Welcome back  
        {userName === "NEW" ? (
          <span>
            Seems like you are new to the System please create account from
            <Link href={"/profile"}> here</Link>
          </span>
        ) : (
          userName
        )}
```

Figure 20:Initialization of dashboard

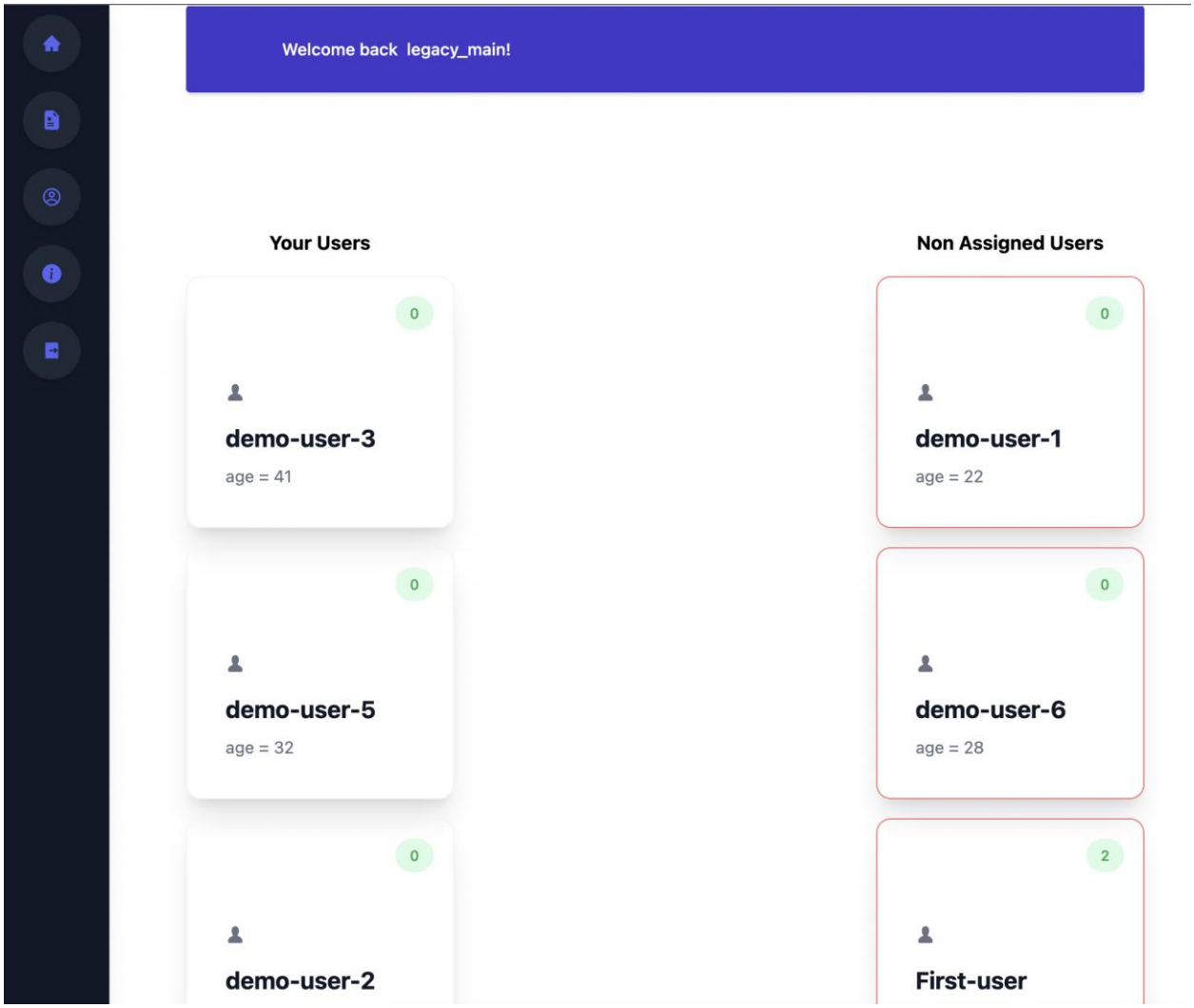


Figure 21:Screen shot of dash board

The above image is for Ophthalmologist

The card shaped ui directs Ophthalmologist to ML API page where predictions can perform

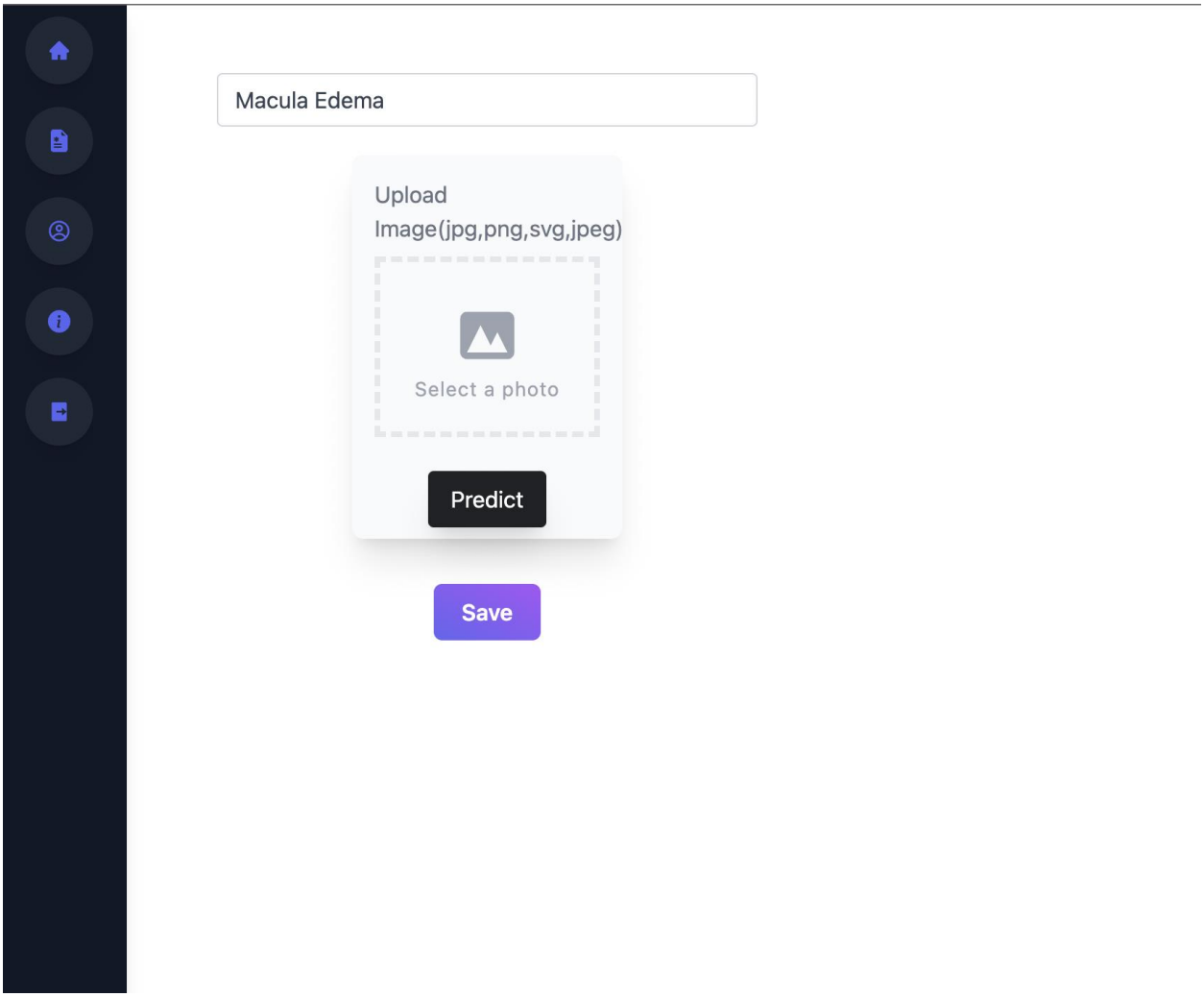


Figure 22:Screen shot of prediction page

This is the page rout for prediction process where user or ophthalmologist can upload images and predict the save

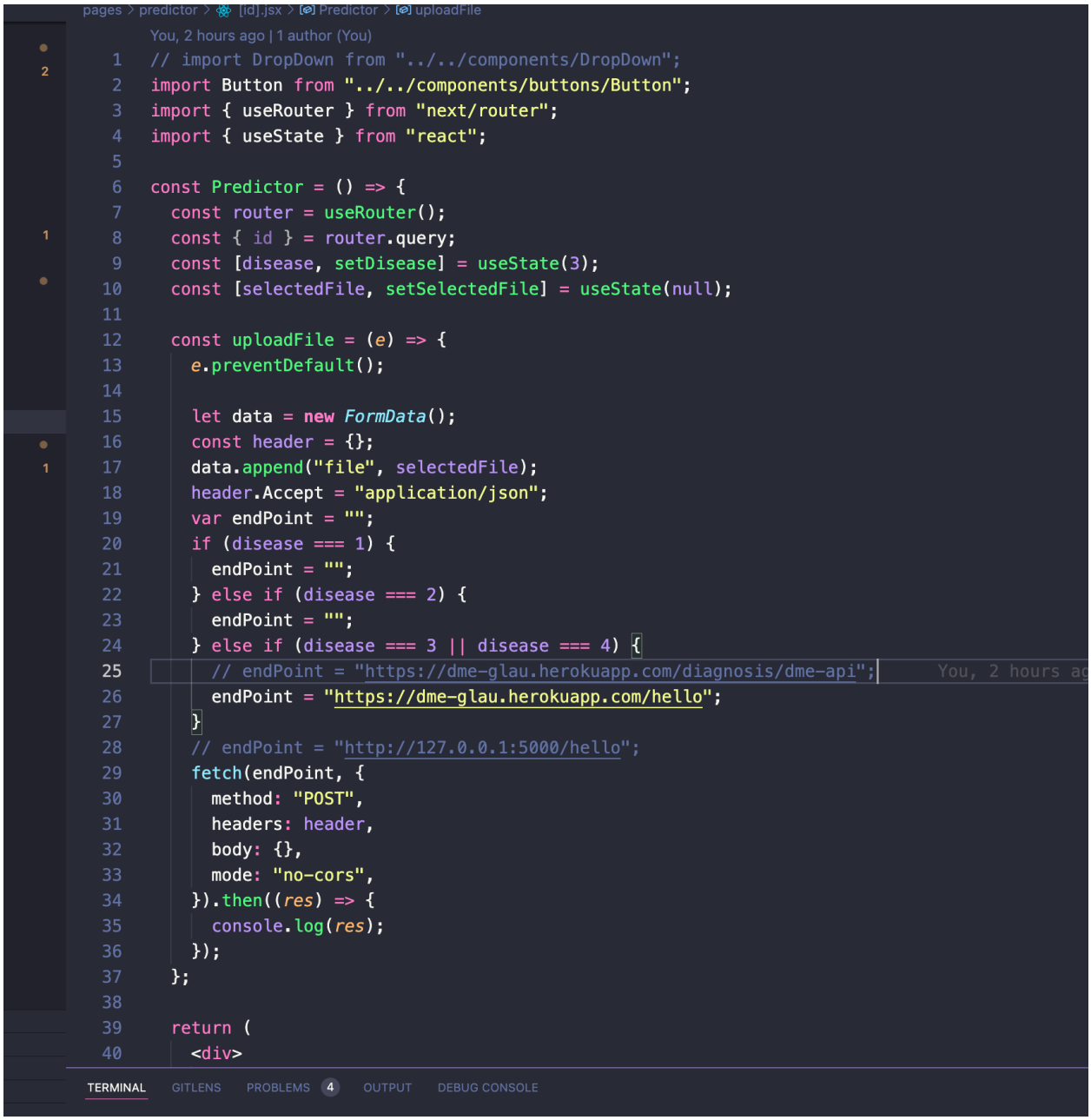


Figure 23:Code for connecting with ML api

The Predictor route uses a dynamic route to capture the designated patient id. The dynamic route can be accessed by using the next useRouter hook and from that the patient identification can be performed. Also in this page ML apis are called using fetch api and image is captured by the formData object which can be passed through the body of the function call.

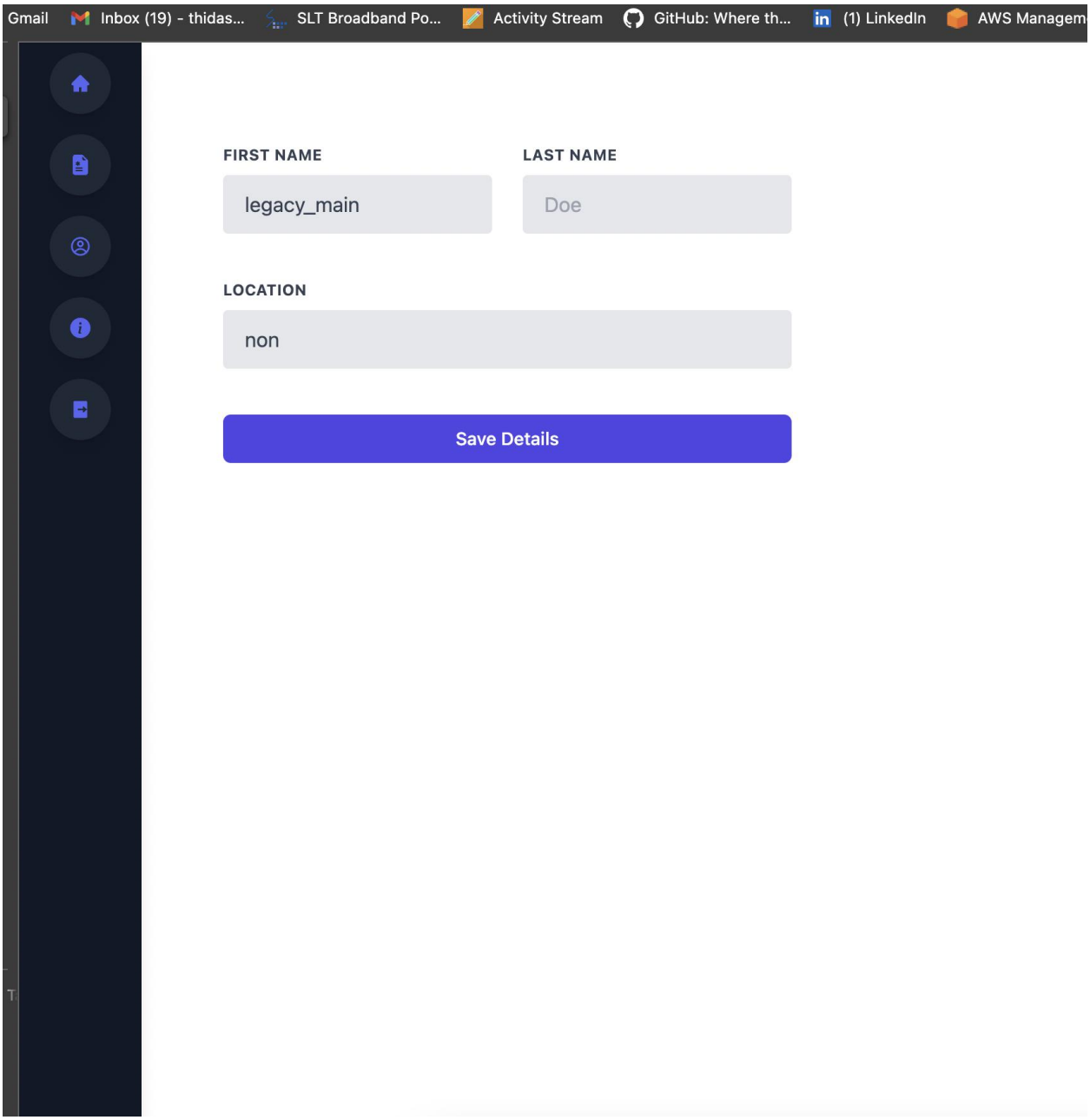
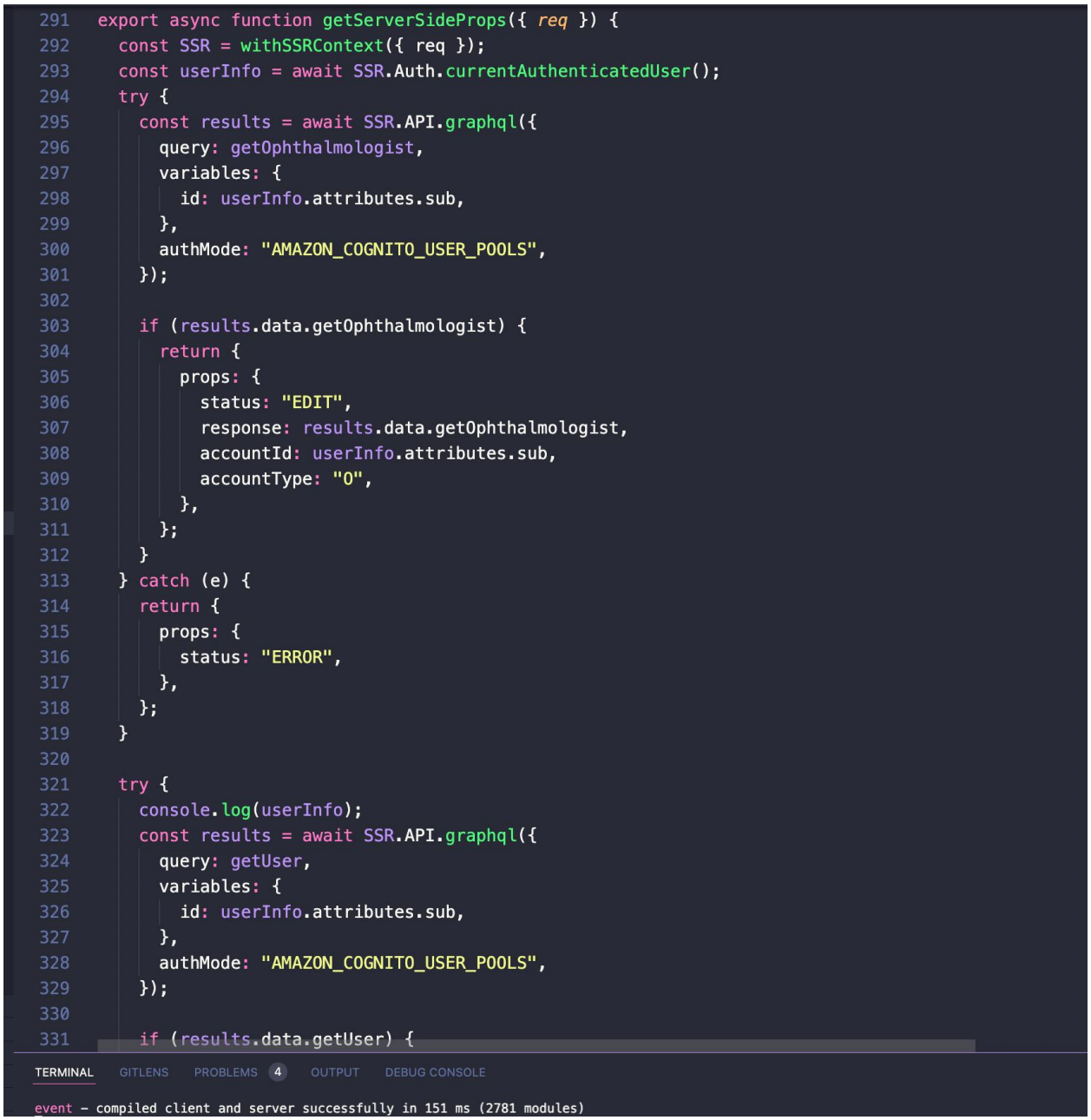


Figure 24:Profile page

Next in the profile page Ophthalmologist and patients can change their name and also new users can register with details.



```
export async function getServerSideProps({ req }) {
  const SSR = withSSRContext({ req });
  const userInfo = await SSR.Auth.currentAuthenticatedUser();
  try {
    const results = await SSR.API.graphql({
      query: getOphthalmologist,
      variables: {
        id: userInfo.attributes.sub,
      },
      authMode: "AMAZON_COGNITO_USER_POOLS",
    });

    if (results.data.getOphthalmologist) {
      return {
        props: {
          status: "EDIT",
          response: results.data.getOphthalmologist,
          accountId: userInfo.attributes.sub,
          accountType: "O",
        },
      };
    }
  } catch (e) {
    return {
      props: {
        status: "ERROR",
      },
    };
  }

  try {
    console.log(userInfo);
    const results = await SSR.API.graphql({
      query: getUser,
      variables: {
        id: userInfo.attributes.sub,
      },
      authMode: "AMAZON_COGNITO_USER_POOLS",
    });

    if (results.data.getUser) {
```

TERMINAL GITLENS PROBLEMS 4 OUTPUT DEBUG CONSOLE

event - compiled client and server successfully in 151 ms (2781 modules)

The profile page implementation

Figure 25:The profile page implementation

1.6 GIT Repository

GitHub and Git was mainly used in collaboration work with multiple team members and also when implementing CI/CD pipe Line.An organization in the name of our team has created been creed and our working repositories are created on them.

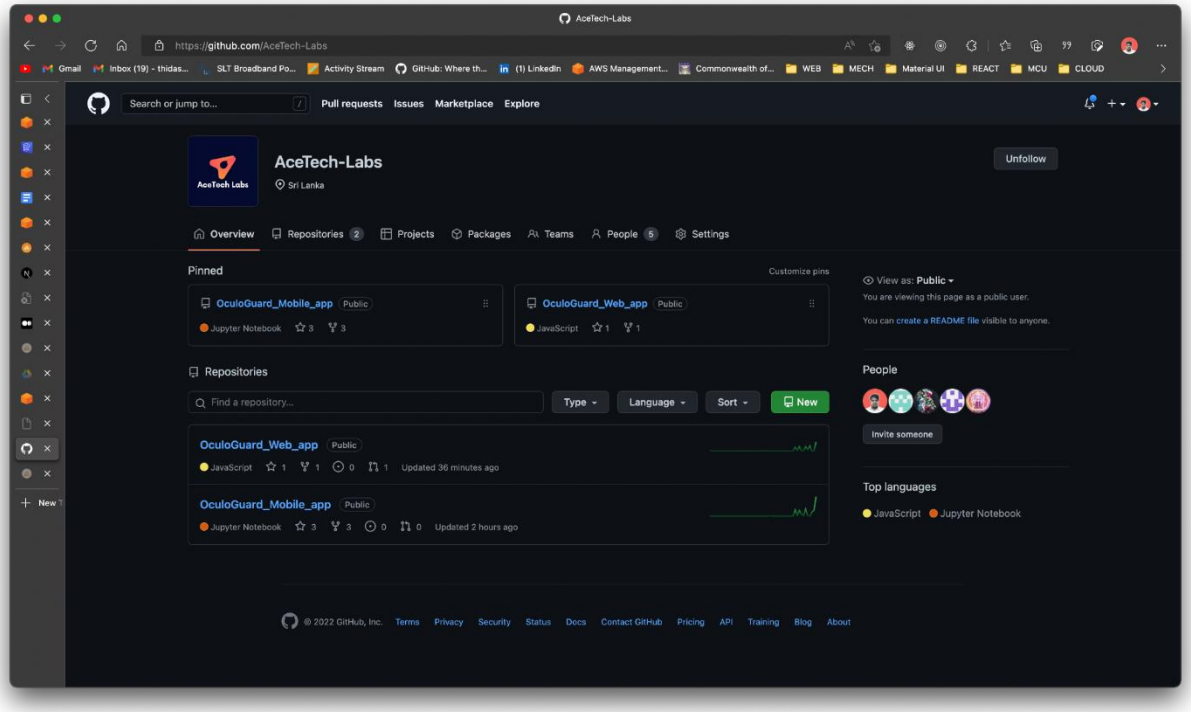


Figure 26:Github repository

Each contributor to the repository created a sub branch on the repository and committed code to the main branch and merged them accordingly.

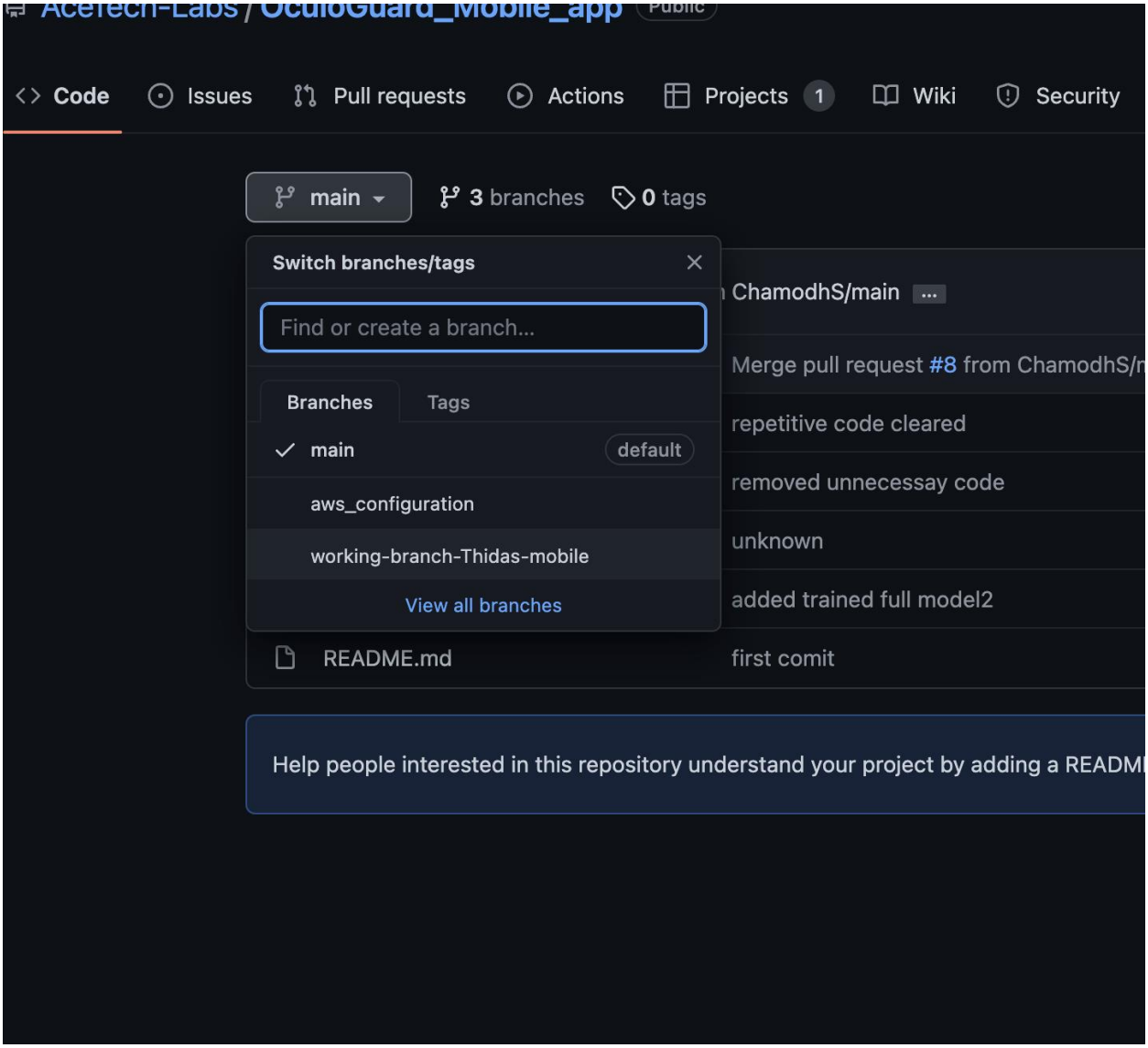


Figure 27:GitHub branches

Also when moving from one technology to another a separate branch was created and implemented and then merged to the main branch.

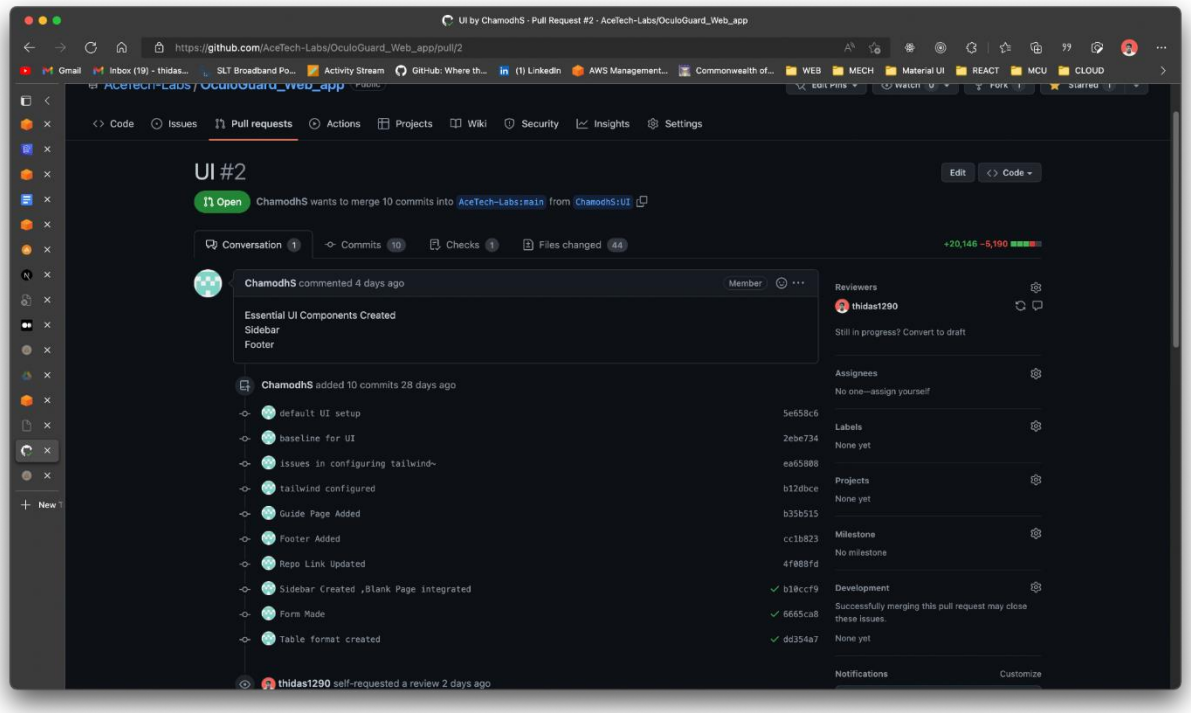


Figure 28:Screenshots of pull requests

Pull requests and code reviews were also committed when merging to the main branch.

1.7 Deployments/CI-CD Pipeline

Continuous Integration and Development pipelines are deployed using AWS Amplify, the same cloud service we used in the rest of this applications. The Github account and organization has given the authorization to connect with aws cloud with the main branch such a way that on every commit to the main branch, amplify will automatically run test and build the application on cloud.

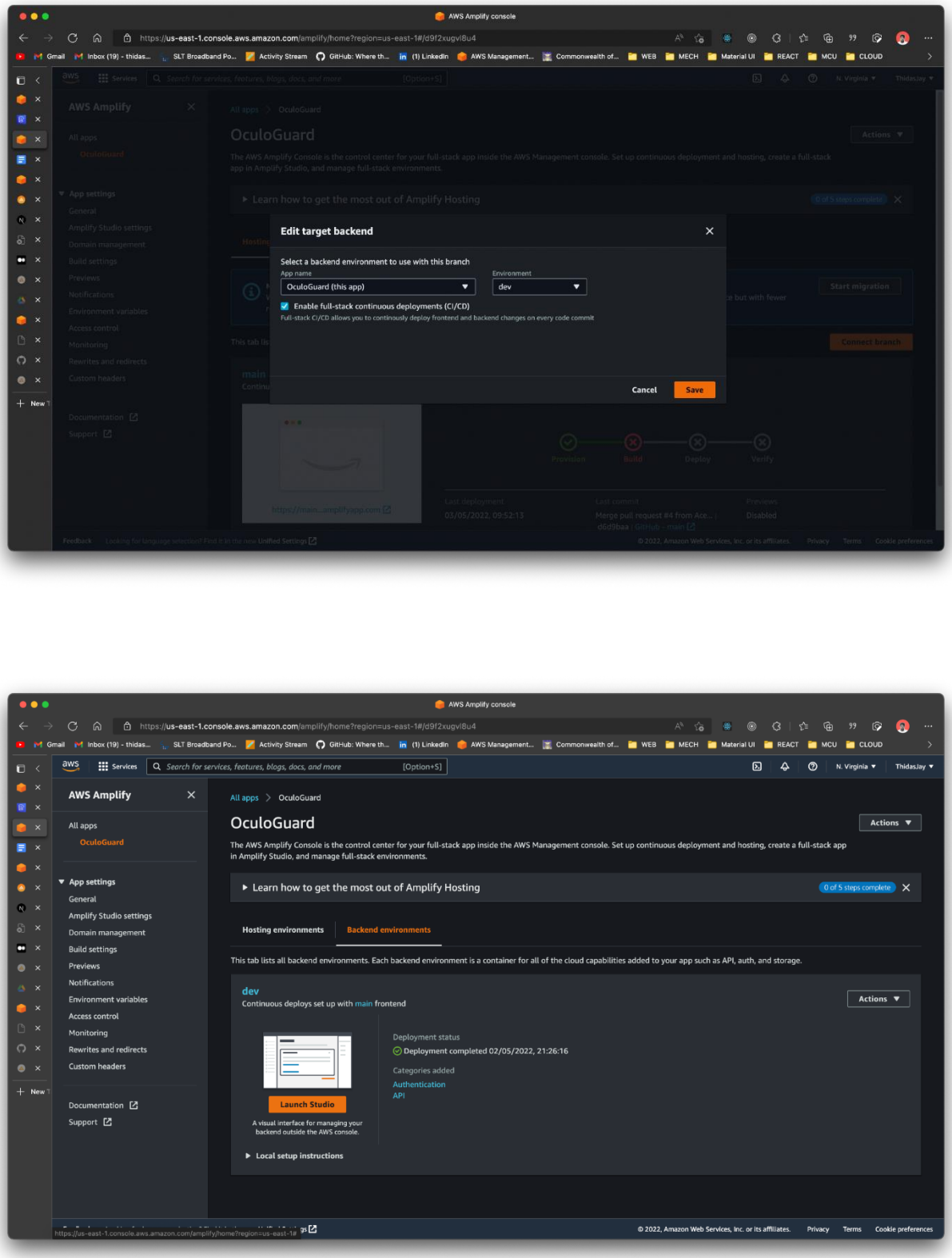


Figure 29:CI-CD Pipeline

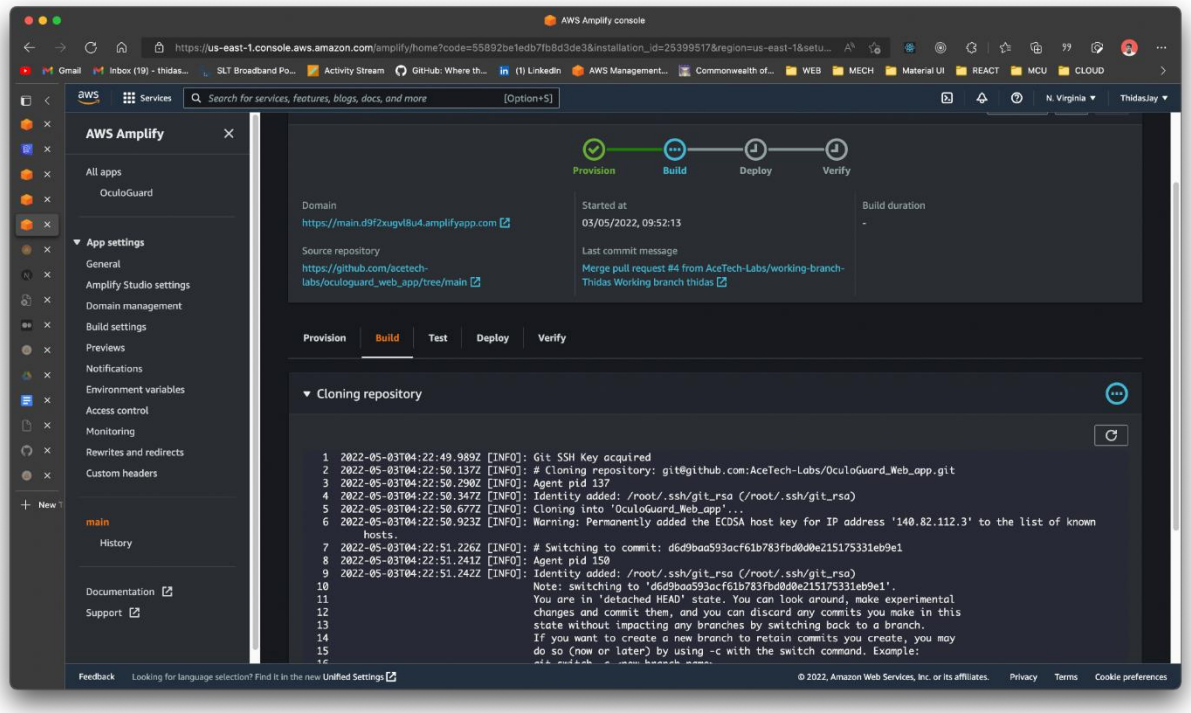


Figure 30:CI-CD Pipeline

Build specification for cloud deployment

File: amplify.yml

```
version: 1
backend:
  phases:
    build:
      commands:
        - '# Execute Amplify CLI with the helper script'
        - amplifyPush --simple
frontend:
  phases:
    preBuild:
      commands:
        - yarn install
    build:
      commands:
        - yarn run build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
```

Figure 31:CI-CD Pipeline

1.8 Chapter Summary

In this chapter,we discussed the OculoGuard software's prototype feature implementation process. This chapter discusses specific technologies,libraries and frameworks that were used. This chapter also described how each feature of the solution was implemented, including explanations through code snippets and screenshots of the implementation as well as the problems encountered and solutions used.The user manual was also covered in detail at the end of this chapter. The next chapter will go over OculoGuard's testing phase.

Chapter 2: Testing

2.1 Chapter Introduction

This chapter is about the Testing phase of OculoGuard. First it will discuss testing criteria and functional/non-functional requirements. and after it will explain Unit, Performance, Usability and Compatibility. Also this chapter will briefly explain about limitations we faced during the testing.

2.2 Testing Criteria

The IEEE 829 test plan was used to carry functional and non-functional testing of the OcularGuard project.We mainly focused on test cases according to the functional and non-functional requirements. These test cases will ensure OculoGuard functions are working as expected.

2.3 Testing functional requirements

this is the table of the black box testing done and its results

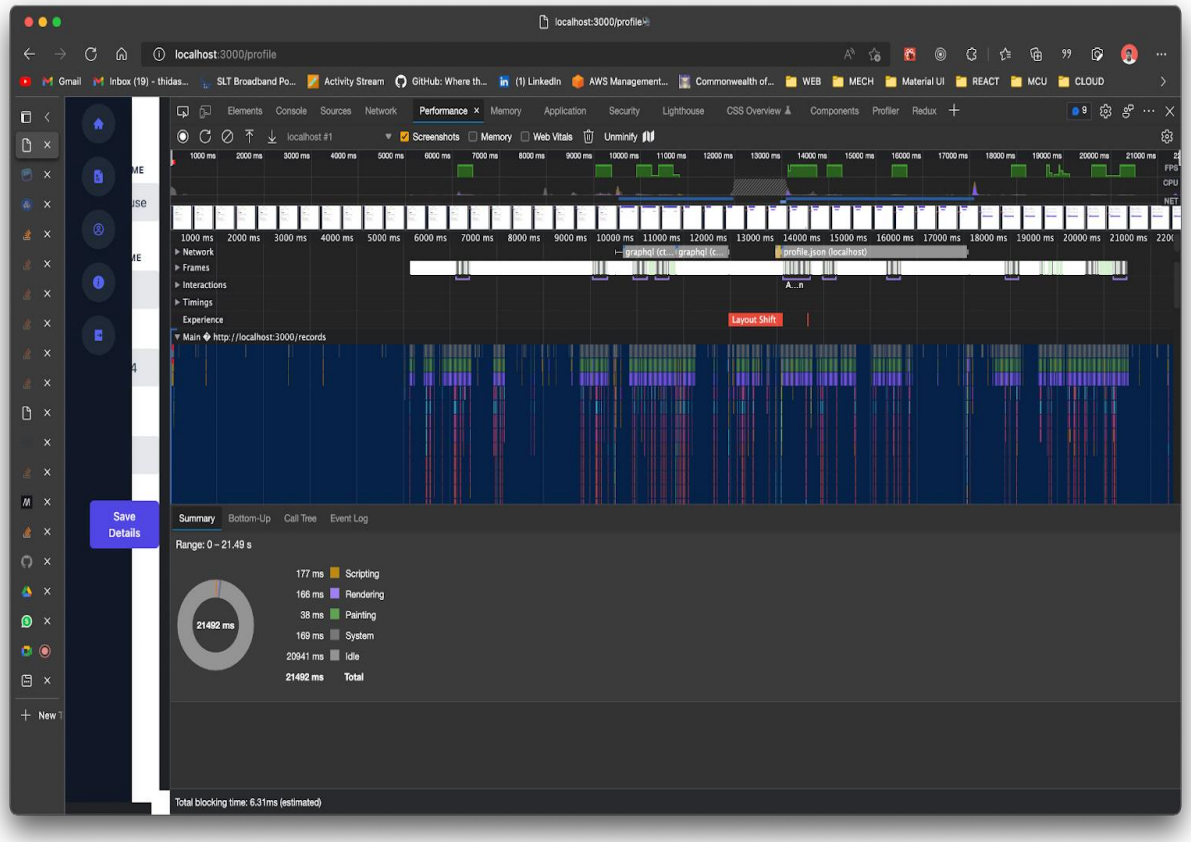
Test Case No.	Feature tested	Priority Level	Description	Expected Result	Actual Result	Status
1.0	FR1	Critical	Opthaamolgists and patients should be able to register and log into the system	Both parties should be able successfully sign in and log into the system	Both Parties can sign in and log in to the system	Pass
2.0	FR2	Critical	Should be able to input fundus images and get accept input and preprocess	System should pre process images before classifying.	System will resize input image and classify	Pass
3.0	FR5	Luxury	View History of Patients	An ophthalmologist can see patient details	ophthalmologist can access to Patients details	Pass

Table 1:Functionalities

2.4 Unit testing

Unit tests were done during developing and errors which identified were fixed

2.5 Performance testing



2.6 Usability testing

Usability testing checks the ease of access and navigation of the OculoGuard applications to users,with the all design of UI. It has a Unique logo to ensure the uniqueness of the project. In addition,the GUI of OculoGuard mobile app and web app works on every suitable device

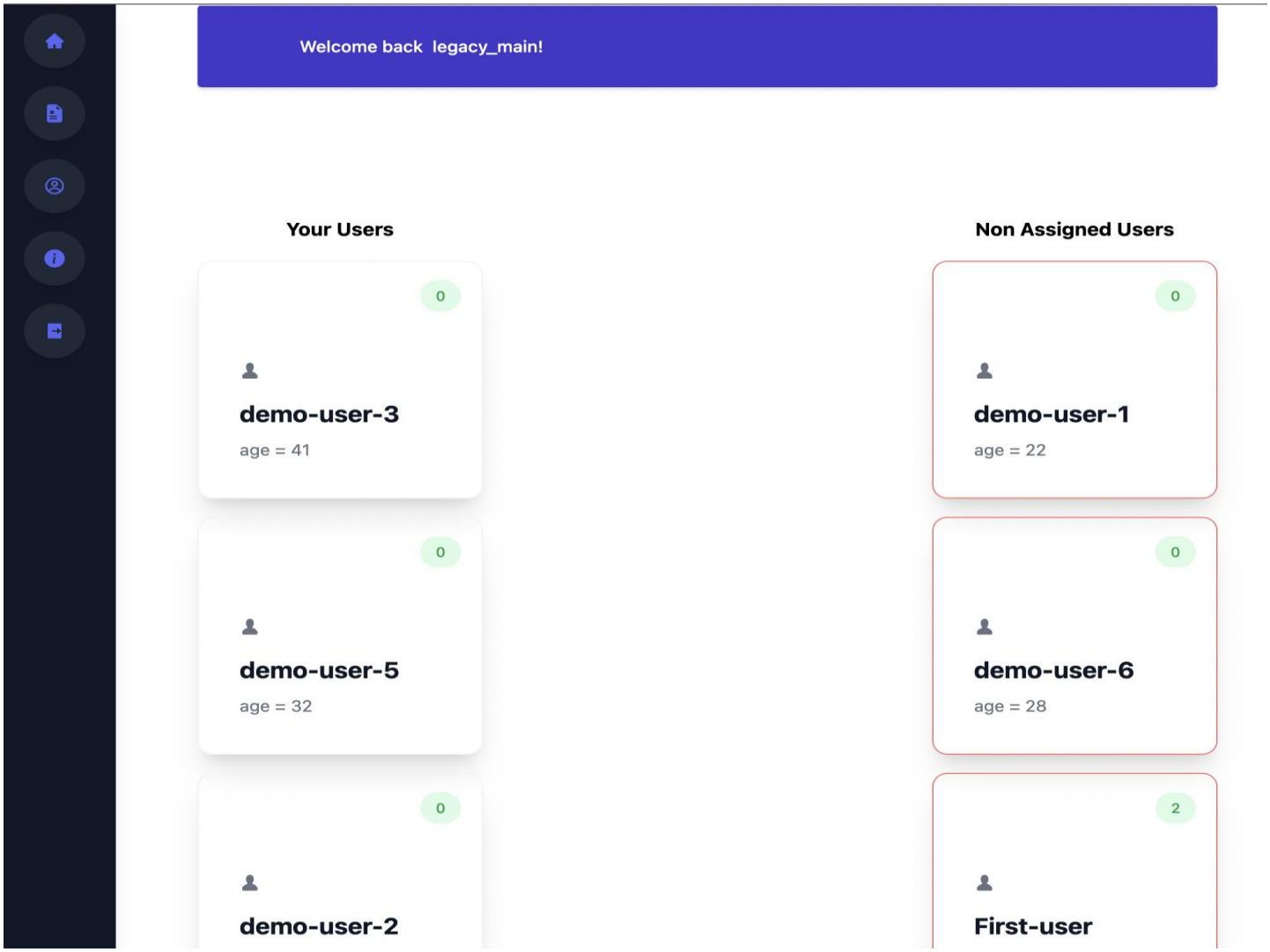


Figure 32Dashboard:

2.7 Compatibility testing

Compatibility ensures that an application can work on different platforms.
the OculoGuard web application was tested in Google chrome, Safari, Mozilla Firefox and Microsoft Edge.Oculus Guard mobile app was tested in many devices. All the functionalities of OculoGuard applications worked well.

2.8 Chapter Summary

In this Chapter we discussed testing methods, functional testing, Performance testing, Usability testing and Compatibility testing of OculoGuard both mobile and web applications.

Chapter 3: Evaluation

3.1 Chapter Overview

This chapter explains how evaluating project accomplishments aids decision-making. The chapter defines roles and duties as well as the methods for project evaluation management. A systematic and impartial examination of a current or finished project is known as project evaluation. Lessons learnt from evaluations are also fed into the decision-making process. The goal is to identify the project's relevance and degree of accomplishment, as well as its development effectiveness, efficiency, impact, and long-term viability.

3.2 Evaluation methods

The purpose of evaluation is to provide you an indication of how successfully the program ideas were implemented. This might involve determining if the volume and frequency of services you anticipated were given, whether program participants were happy with their experience, and the degree to which the program was carried out with fidelity, or adherence to the program developers' concept.

3.3 Quantitative evaluation

The goal of quantitative evaluation is to get a certain result. For your project, you'll need to have predetermined results. Then, using numerical data, you'll assess how well your program is performing in terms of these outcomes. Various measures such as true positive (TP), false positive (FP), false negative (FN), and true negative (TN) rates have been used to evaluate the performance for diagnosing the disease and determining the severity of the condition.

Code for calculating accuracy, precession, and recall

```
import numpy as np
```
✓ 0.4s

```
cm = np.array(
[[531,   2,   0,  24],
 [  0,  50,   0,   0],
 [ 31,   6,  54,   9],
 [ 12,   0,   0, 529]])
```
✓ 0.1s

```
def precision(label, confusion_matrix):
    col = confusion_matrix[:, label]
    return confusion_matrix[label, label] / col.sum()

def recall(label, confusion_matrix):
    row = confusion_matrix[label, :]
    return confusion_matrix[label, label] / row.sum()

def precision_macro_average(confusion_matrix):
    rows, columns = confusion_matrix.shape
    sum_of_precisions = 0
    for label in range(rows):
        sum_of_precisions += precision(label, confusion_matrix)
    return sum_of_precisions / rows

def recall_macro_average(confusion_matrix):
    rows, columns = confusion_matrix.shape
```

```
    rows, columns = confusion_matrix.shape
    sum_of_recalls = 0
    for label in range(columns):
        sum_of_recalls += recall(label, confusion_matrix)
    return sum_of_recalls / columns
# print("label precision recall")
# for label in range(4):
#     print(f"{label:5d} {precision(label, cm):9.3f} {recall(label, cm):6.3f}")
```

✓ 1.9s

```
print("label precision recall")
for label in range(4):
    print(f"{label:5d} {precision(label, cm):9.3f} {recall(label, cm):6.3f}")
def accuracy(confusion_matrix):
    diagonal_sum = confusion_matrix.trace()
    sum_of_all_elements = confusion_matrix.sum()
    return diagonal_sum / sum_of_all_elements
print("Accuracy: ",accuracy(cm))
print("precision total:", precision_macro_average(cm))
print("recall total:", recall_macro_average(cm))
```

✓ 0.2s

Figure 33:Code for calculating accuracy, precession, and recall

```
label precision recall
    0     1.000  1.000
    1     1.000  0.930
    2     0.952  0.985
    3     0.663  0.586
    4     0.648  0.893
    5     0.682  0.170
    6     0.727  0.140
Accuracy:  0.8274478330658106
precision total: 0.810314311703751
recall total: 0.6720828808825655
```

Output Model 1

Figure 34:Model 1 Evaluation

```
label precision recall
    0     0.925  0.953
    1     0.862  1.000
    2     1.000  0.540
    3     0.941  0.978
Accuracy:  0.9326923076923077
precision total: 0.9321093030803035
recall total: 0.8677850546066365
```

Output Model 2

Figure 35:Model 2 Evaluation

```
label precision recall
    0     0.622  0.622
    1     0.709  0.676
    2     0.467  0.568
    3     0.423  0.386
Accuracy:  0.618018018018018
precision total: 0.555190048537318 6
recall total: 0.5628384090912695
```

Output Model 3

Figure 36:Model 3 Evaluation

Chapter 3 - Evaluation

3.4 Qualitative evaluation (Feedback from end users, domain experts and industry experts)

As a Qualitative evaluation a review with the domain expert has been done to gather feedback and the feedback was used to improve, enhance and evaluate the system.

3.5 Self evaluation

OculoGuard is a software solution that detects the Early Prognosis of Diabetic Retinopathy, Diabetic Macular Edema and Glaucoma that can be deployed to enhance earlier detection of DEDs cutting costs, saving resources, and reducing complications and casualties. Given the present state of the project, there are a few capabilities that might be enhanced. Prognosis can be done more effectively, but it takes months of training to develop a model that is both efficient and accurate. Since this is a second-year project and considering the time allocated to the project the percentage of accuracy and development seems to be sufficient. This project was initially planned to use a random forest algorithm for the machine learning part but considering the disease categories and available resources we moved to a mobile net which seemed to be best fit for our system as we are detecting four diseases.

3.6 Chapter Summary

This chapter discussed evaluation methodologies and addressed the most appropriate approaches for the project OculoGuard. By researching many evaluation methods, the most appropriate one which suits this project was taken. All methods used for evaluation are briefly described with examples and results. Also, this chapter has the evaluation conducted by all group members. Hope this chapter will be helpful for decision making, effectiveness, efficiency, impact, and long-term viability.

Chapter 4: Conclusion

4.1 Chapter Overview

This chapter emphasizes upon the achievement of the relevant aims and objectives of the Oculo-Guard project while bringing forth the limitations the team had to face and the intended future enhancements and integrations the team is planning to add to the existing prototype

4.2 Achievements of aims and objectives

The primary goal was to design,implement and develop a software solution that not only focus on the diagnosis of the most prevalent diabetic diseases (Diabetic Retinopathy,Glaucoma,Diabetic Macular Edema) but as well as prognosis of Diabetic Retinopathy which was proposed as an initiative in order to minimize cost and promote prevention of diabetic side effects.In addition the proposal of an efficient patient management system created using both the web application and the mobile application was also a key aim of the project.

Functionalities	Achievement
Should be able to input fundus images,accept inputs and preprocess	Successful, input fundus images and undergoes preprocessing
Should be able to extract features from fundus images and early prognose DR and diagnose DR, Macula Edema and Glaucoma	Successful, Features are filtered and extracted. Models are successful in accurately detecting the proposed eye diseases and carries out prognosis for Diabetic Retinopathy

Table 2:Achievement table

The aims of Oculo-Guard have been fulfilled during the given time,and successfully provides a single solution for the detection of multiple prominent diabetic eye diseases .

4.3 Limitations of the research

The limitation of the research mainly focused on the prognosis sector of the project.As the datasets used for the prognosis model was relatively smaller and barely adequate it was difficult to obtain precise accuracy on the prognosis model.

There was a relatively difficult limitation the team faced when it became evident that most of the said diseases have identical features which complicated the differentiating process.

Chapter 4-Conclusion

4.4 Future enhancements

Building a smartphone fundus photography compatible model

Currently the inputs for the diagnosis models are in the form of high quality fundus images obtained from an ophthalmoscope using a direct ophthalmoscopy. However one of the main initiatives that was proposed was to make the model compatible to an indirect ophthalmoscopy which can be simply done using a smartphone and a magnifying lens.The model being compatible for indirect ophthalmoscopy makes Oculo-Guard widely accessible even in environments with fewer resources and medical equipment

Increasing the coverage of diseases and reducing the timeline for the earliest possible diagnosis

The prototype only focus on 3 prominent diabetic eye diseases in terms of diagnosis.One of the main future enhancements will focus on increasing the coverage of diseases by the created diagnosis system (Eg-Cataracts).Exploring the possibility of modifying the model to cover non diabetic eye complications is also one of the initiatives discussed for future enhancements.

Further improvements in accuracy and performance of prognosis

Due to the limited availability of the relevant datasets the accuracy of the prognosis model for Diabetic Retinopathy was somewhat hindered. Initiatives were proposed for more accurate pre-processing mechanisms and increasing the sample size of the datasets and training a more precise and accurate model for even better results .

4.5 Concluding remarks

Diabetes related eye complications is already a major issue in the modern day world with appropriate medical resources needed for prevention of these diseases being unproportionately distributed throughout the world.

In a situation where the number of diabetes patients is going to witness an exponential increase we have successfully suggested an approach which focuses on prevention rather than cure which minimizes the resources that need to be utilized for combating this issue.

Team Acetech-Labs is proud to have made a significant contribution to a solution for a major world problem and thankful for the contributions and the dedication of each team member as well as the tremendous support received from the module team.

Appendix

Each section under the appendix should be named “Appendix A”, “Appendix B” etc.
For this the page numbering should be Uppercase roman numerals

Appendix A