Sprint – 4

**Front end web application development**

| **Team ID** | PNT2022TMID13366 |
| --- | --- |
| **Project Name** | Fertilizer recommendation system for disease prediction |

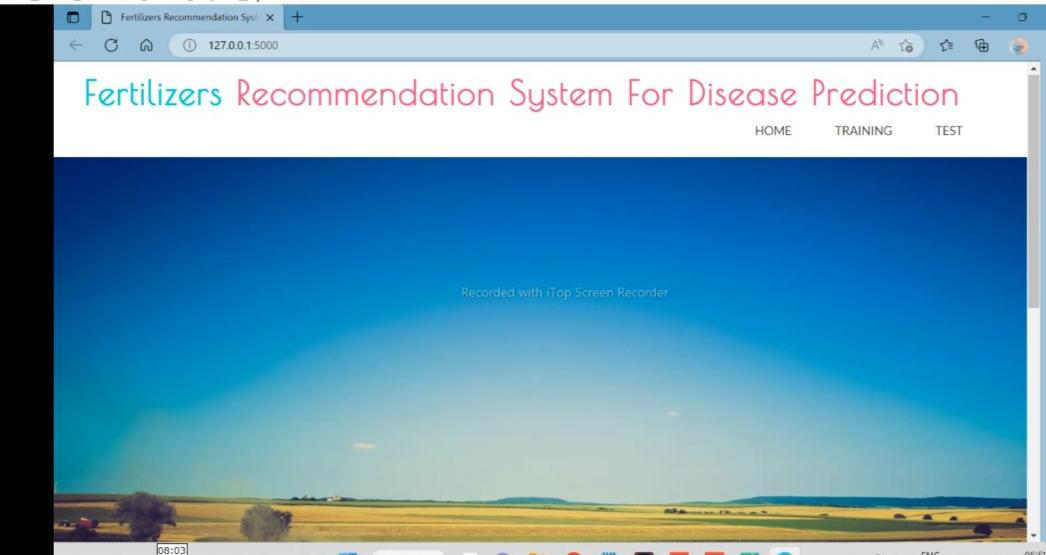
**Build HTML Pages**

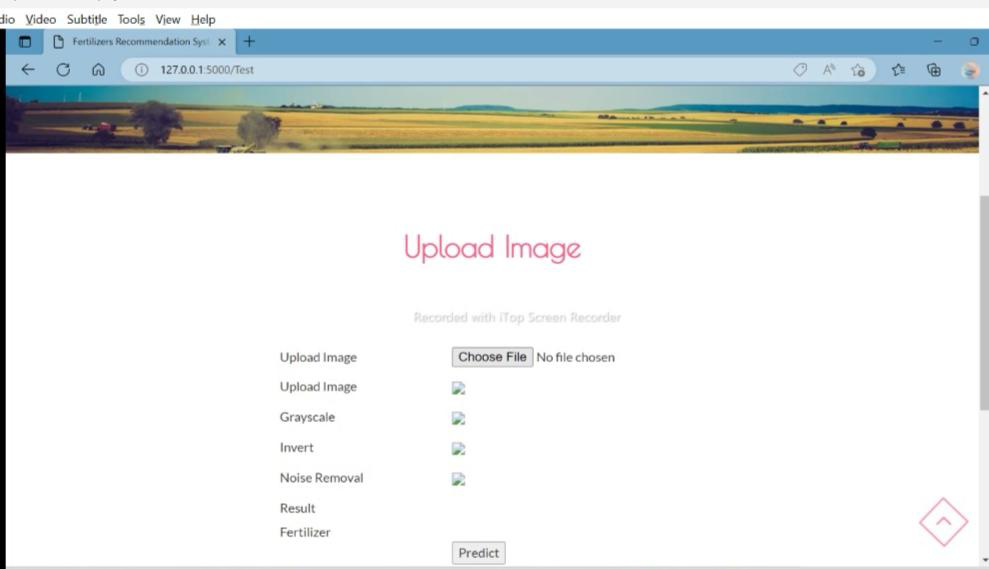
Build the UI where a home page will have details about the application, a prediction page where a user is allowed to browse an image and get the predictions.

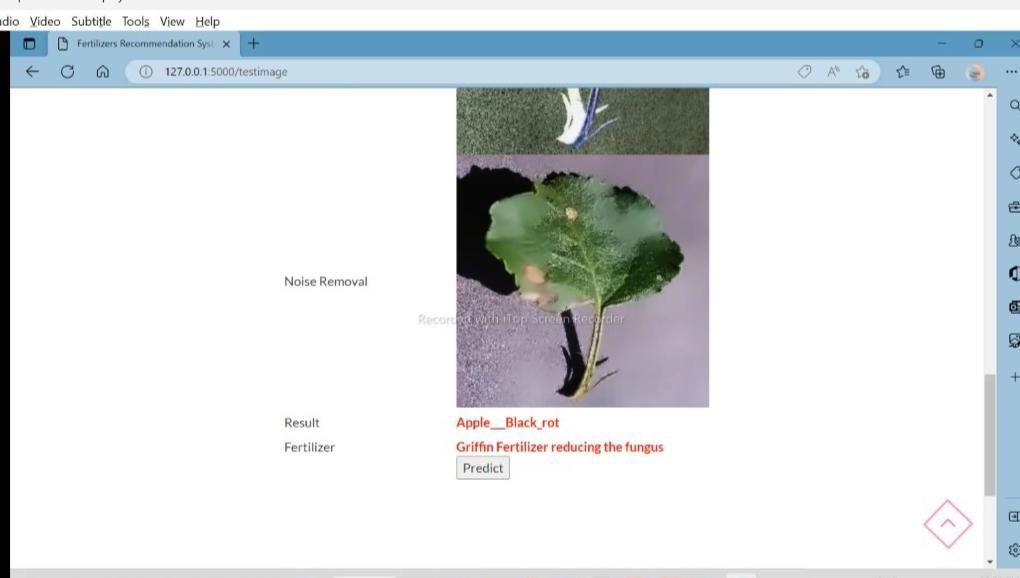
# Open the browser and navigate to localhost to check your web application.

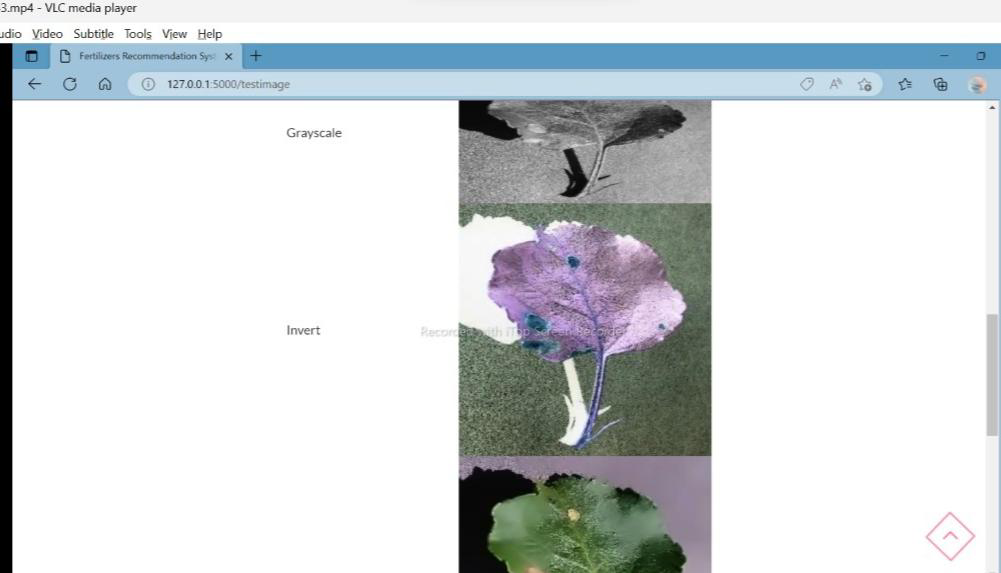
**The home page looks like this.**

# \* Running the application on http://127.0.0.1:5000



****

****

****

**Index.html**

from flask import Flask, render\_template, flash, request, session,send\_file

from flask import render\_template, redirect, url\_for, request

import warnings

import datetime

import cv2

app = Flask(\_\_name\_\_)

app.config['DEBUG']

app.config['SECRET\_KEY'] = '7d441f27d441f27567d441f2b6176a'

@app.route("/")

def homepage():

return render\_template('index.html')

@app.route("/Training")

def Training():

return render\_template('Tranning.html')

@app.route("/Test")

def Test():

return render\_template('Test.html')

@app.route("/train", methods=['GET', 'POST'])

def train():

if request.method == 'POST':

import model as model

return render\_template('Tranning.html')

@app.route("/testimage", methods=['GET', 'POST'])

def testimage():

if request.method == 'POST':

file = request.files['fileupload']

file.save('static/Out/Test.jpg')

img = cv2.imread('static/Out/Test.jpg')

if img is None:

print('no data')

img1 = cv2.imread('static/Out/Test.jpg')

print(img.shape)

img = cv2.resize(img, ((int)(img.shape[1] / 5), (int)(img.shape[0] / 5)))

original = img.copy()

neworiginal = img.copy()

cv2.imshow('original', img1)

gray = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

img1S = cv2.resize(img1, (960, 540))

cv2.imshow('Original image', img1S)

grayS = cv2.resize(gray, (960, 540))

cv2.imshow('Gray image', grayS)

gry = 'static/Out/gry.jpg'

cv2.imwrite(gry, grayS)

from PIL import ImageOps,Image

im = Image.open(file)

im\_invert = ImageOps.invert(im)

inv = 'static/Out/inv.jpg'

im\_invert.save(inv, quality=95)

dst = cv2.fastNlMeansDenoisingColored(img1, None, 10, 10, 7, 21)

cv2.imshow("Nosie Removal", dst)

noi = 'static/Out/noi.jpg'

cv2.imwrite(noi, dst)

import warnings

warnings.filterwarnings('ignore')

import tensorflow as tf

classifierLoad = tf.keras.models.load\_model('model.h5')

import numpy as np

from keras.preprocessing import image

test\_image = image.load\_img('static/Out/Test.jpg', target\_size=(200, 200))

img1 = cv2.imread('static/Out/Test.jpg')

# test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis=0)

result = classifierLoad.predict(test\_image)

out = ''

fer = ''

if result[0][0] == 1:

out = "Apple\_\_\_Black\_rot"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][1] == 1:

out = "Apple\_\_\_healthy"

elif result[0][2] == 1:

out = "Corn\_(maize)\_\_\_healthy"

elif result[0][3] == 1:

out = "Corn\_(maize)\_\_\_Northern\_Leaf\_Blight"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][4] == 1:

out = "Peach\_\_\_Bacterial\_spot"

fer = 'Compounds available for use on peach and nectarine for bacterial spot include copper, oxytetracycline (Mycoshield and generic equivalents), and syllit+captan; however, repeated applications are typically necessary for even minimal disease control.'

elif result[0][5] == 1:

out = "Peach\_\_\_healthy"

if result[0][6] == 1:

out = "Pepper\_bell\_\_\_Bacterial\_spot"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][7] == 1:

out = "Pepper\_bell\_\_\_healthy"

elif result[0][8] == 1:

out = "Potato\_\_\_Early\_blight"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][9] == 1:

out = "Potato\_\_\_healthy"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][10] == 1:

out = "Potato\_\_\_Late\_blight"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][11] == 1:

out = "Tomato\_\_\_Bacterial\_spot"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][12] == 1:

out = "Tomato\_\_\_Late\_blight"

fer = 'Spraying fungicides is the most effective way to prevent late bligh'

elif result[0][13] == 1:

out = "Tomato\_\_\_Leaf\_Mold"

fer = 'Griffin Fertilizer reducing the fungus'

elif result[0][14] == 1:

out = "Tomato\_\_\_Septoria\_leaf\_spot"

fer = 'Griffin Fertilizer reducing the fungus'

org = 'static/Out/Test.jpg'

gry ='static/Out/gry.jpg'

inv = 'static/Out/inv.jpg'

noi = 'static/Out/noi.jpg'

return render\_template('Test.html',fer=fer,result=out,org=org,gry=gry,inv=inv,noi=noi)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, use\_reloader=True)

**Base.html**

from keras.models import Sequential

from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense

from keras.models import model\_from\_json

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

batch\_size = 32

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255

train\_datagen = ImageDataGenerator(rescale=1/255)

# Flow training images in batches of 128 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(

'DataSet', # This is the source directory for training images

target\_size=(200, 200), # All images will be resized to 200 x 200

batch\_size=batch\_size,

# Specify the classes explicitly

classes = ['Apple\_\_\_Black\_rot','Apple\_\_\_healthy','Corn\_(maize)\_\_\_healthy','Corn\_(maize)\_\_\_Northern\_Leaf\_Blight',

'Peach\_\_\_Bacterial\_spot','Peach\_\_\_healthy','Pepper\_bell\_\_\_Bacterial\_spot','Pepper\_bell\_\_\_healthy','Potato\_\_\_Early\_blight',

'Potato\_\_\_healthy','Potato\_\_\_Late\_blight','Tomato\_\_\_Bacterial\_spot','Tomato\_\_\_Late\_blight',

'Tomato\_\_\_Leaf\_Mold','Tomato\_\_\_Septoria\_leaf\_spot'],

# Since we use categorical\_crossentropy loss, we need categorical labels

class\_mode='categorical')

import tensorflow as tf

model = tf.keras.models.Sequential([

# Note the input shape is the desired size of the image 200x 200 with 3 bytes color

# The first convolution

tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(200, 200, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

# The second convolution

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The third convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The fourth convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The fifth convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# Flatten the results to feed into a dense layer

tf.keras.layers.Flatten(),

# 128 neuron in the fully-connected layer

tf.keras.layers.Dense(128, activation='relu'),

# 5 output neurons for 5 classes with the softmax activation

tf.keras.layers.Dense(15, activation='softmax')

])

model.summary()

from tensorflow.keras.optimizers import RMSprop

early = tf.keras.callbacks.EarlyStopping(monitor='val\_loss',patience=5)

model.compile(loss='categorical\_crossentropy',

optimizer=RMSprop(lr=0.001),

metrics=['accuracy'])

total\_sample=train\_generator.n

n\_epochs = 10

history = model.fit\_generator(

train\_generator,

steps\_per\_epoch=int(total\_sample/batch\_size),

epochs=n\_epochs,

verbose=1)

model.save('model.h5')

acc = history.history['accuracy']

loss = history.history['loss']

epochs = range(1, len(acc) + 1)

# Train and validation accuracy

plt.plot(epochs, acc, 'b', label=' accurarcy')

plt.title(' accurarcy')

plt.legend()

plt.figure()

# Train and validation loss

plt.plot(epochs, loss, 'b', label=' loss')

plt.title(' loss')

plt.legend()

plt.show()mport tensorflow as tf

import numpy as np

from tkinter import \*

import os

from tkinter import filedialog

import cv2

import time

from matplotlib import pyplot as plt

from tkinter import messagebox

def endprogram():

print ("\nProgram terminated!")

sys.ex

def fulltraining():

import model as mm

def testing():

global testing\_screen

testing\_screen = Toplevel(main\_screen)

testing\_screen.title("Testing")

# login\_screen.geometry("400x300")

testing\_screen.geometry("600x450+650+150")

testing\_screen.minsize(120, 1)

testing\_screen.maxsize(1604, 881)

testing\_screen.resizable(1, 1)

testing\_screen.configure(bg='green')

# login\_screen.title("New Toplevel")

Label(testing\_screen, text='''Upload Image''', disabledforeground="#a3a3a3",

foreground="#000000", width="300", height="2", font=("Calibri", 16)).pack()

Label(testing\_screen, text="").pack()

Label(testing\_screen, text="").pack()

Label(testing\_screen, text="").pack()

Button(testing\_screen, text='''Upload Image''', font=(

'Verdana', 15), height="2", width="30", command=imgtest).pack()

global affect

def imgtest():

import\_file\_path = filedialog.askopenfilename()

image = cv2.imread(import\_file\_path)

print(import\_file\_path)

filename = 'Output/Out/Test.jpg'

cv2.imwrite(filename, image)

print("After saving image:")

#result()

#import\_file\_path = filedialog.askopenfilename()

print(import\_file\_path)

fnm = os.path.basename(import\_file\_path)

print(os.path.basename(import\_file\_path))

# file\_sucess()

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nImage : " + fnm + "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

img = cv2.imread(import\_file\_path)

if img is None:

print('no data')

img1 = cv2.imread(import\_file\_path)

print(img.shape)

img = cv2.resize(img, ((int)(img.shape[1] / 5), (int)(img.shape[0] / 5)))

original = img.copy()

neworiginal = img.copy()

cv2.imshow('original', img1)

gray = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

img1S = cv2.resize(img1, (960, 540))

cv2.imshow('Original image', img1S)

grayS = cv2.resize(gray, (960, 540))

cv2.imshow('Gray image', grayS)

dst = cv2.fastNlMeansDenoisingColored(img1, None, 10, 10, 7, 21)

cv2.imshow("Nosie Removal", dst)

thresh = 127

im\_bw = cv2.threshold(grayS, thresh, 255, cv2.THRESH\_BINARY)[1]

#cv2.imshow("affect Removal", im\_bw)

number\_of\_black\_pix = np.sum(im\_bw == 0)

#print(number\_of\_black\_pix)

#if(number\_of\_black\_pix<5000):

#affect =result()

def result():

import warnings

warnings.filterwarnings('ignore')

import tensorflow as tf

classifierLoad = tf.keras.models.load\_model('firemodel.h5')

import numpy as np

from keras.preprocessing import image

test\_image = image.load\_img('Output/Out/Test.jpg', target\_size=(200, 200))

img1 = cv2.imread('Output/Out/Test.jpg')

# test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis=0)

result = classifierLoad.predict(test\_image)

out = ''

pre=''

if result[0][0] == 1:

out="Fire"

elif result[0][1] == 1:

out="Nofire"

messagebox.showinfo("Result", "Classfication Rssult : "+str(out)

def main\_account\_screen():

global main\_screen

main\_screen = Tk()

width = 600

height = 600

screen\_width = main\_screen.winfo\_screenwidth()

screen\_height = main\_screen.winfo\_screenheight()

x = (screen\_width / 2) - (width / 2)

y = (screen\_height / 2) - (height / 2)

main\_screen.geometry("%dx%d+%d+%d" % (width, height, x, y))

main\_screen.resizable(0, 0)

# main\_screen.geometry("300x250")

main\_screen.configure(bg='green')

main\_screen.title("Forest Fire Detection ")

Label(text="Forest Fire Detection", width="300", height="5", bg='green', font=("Calibri", 16)).pack()

Button(text="Training", font=( 'Verdana', 15), height="2", width="30", bg='green',command=fulltraining,highlightcolor="black").pack(side=TOP)

Label(text="").pack()

Button(text="Testing", font=( 'Verdana', 15), height="2", width="30",bg='green', command=testing).pack(side=TOP)

Label(text="").pack()

main\_screen.mainloop()

main\_account\_screen()

|  |  |
| --- | --- |

|  |  |
| --- | --- |