# PROJECT DEVELOPMENT PHASE

**Sprint - IV**

|  |  |
| --- | --- |
| Date | 17 November 2022 |
| Team ID | PNT2022TMID49457 |
| Project Name | Fertilizers Recommendation System for Disease Prediction |
| Maximum Marks | 8 Marks |

# Integration the Deployed Model with Flask

**Web Application**

**App.py**

# Importing essential libraries and modules

from flask import Flask, render\_template, request, Markup

import numpy as np

import pandas as pd

from utils.disease import disease\_dic

from utils.fertilizer import fertilizer\_dic

import requests

import config

import pickle

import io

import torch

from torchvision import transforms

from PIL import Image

from utils.model import ResNet9

import mysql.connector

conn=mysql.connector.connect(host="localhost", user="root", password="", database="login")

cursor=conn.cursor()

disease\_classes = ['Apple\_\_\_Black\_rot',

'Apple\_\_\_healthy',

'Corn\_(maize)\_\_\_Northern\_Leaf\_Blight',

'Corn\_(maize)\_\_\_healthy',

'Peach\_\_\_Bacterial\_spot',

'Peach\_\_\_healthy',

'Pepper,bell\_\_Bacterial\_spot',

'Pepper,bell\_\_healthy',

'Potato\_\_\_Early\_blight',

'Potato\_\_\_Late\_blight',

'Potato\_\_\_healthy',

'Tomato\_\_\_Bacterial\_spot',

'Tomato\_\_\_Late\_blight',

'Tomato\_\_\_Leaf\_Mold',

'Tomato\_\_\_Septoria\_leaf\_spot']

disease\_model\_path = 'models/plant-disease-model.pth'

disease\_model = ResNet9(3, len(disease\_classes))

disease\_model.load\_state\_dict(torch.load(

disease\_model\_path, map\_location=torch.device('cpu')))

disease\_model.eval()

def predict\_image(img, model=disease\_model):

"""

Transforms image to tensor and predicts disease label

:params: image

:return: prediction (string)

"""

transform = transforms.Compose([

transforms.Resize(256),

transforms.ToTensor(),

])

image = Image.open(io.BytesIO(img))

img\_t = transform(image)

img\_u = torch.unsqueeze(img\_t, 0)

# Get predictions from model

yb = model(img\_u)

# Pick index with highest probability

\_, preds = torch.max(yb, dim=1)

prediction = disease\_classes[preds[0].item()]

# Retrieve the class label

return prediction

app = Flask(\_name\_)

# render home page

@app.route('/')

def home():

title = 'Farmer Buddy - Home'

return render\_template('index.html', title=title)

@app.route('/fertilizer')

def fertilizer\_recommendation():

title = 'Harvestify - Fertilizer Suggestion'

return render\_template('fertilizer.html', title=title)

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

def login(): # put application's code here

return render\_template('login.html')

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

def register():

return render\_template('register.html')

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

def success():

return render\_template('success.html')

@app.route('/login\_validation', methods=['POST'])

def login\_validation():

email=request.form.get('email')

password=request.form.get('password')

cursor.execute("""SELECT \* FROM users WHERE email LIKE'{}' AND password LIKE '{}'""".format(email,password))

users = cursor.fetchall()

if len(users)>0:

return render\_template('success.html')

else:

return render\_template('login.html', prediction\_text = "1" )

@app.route('/add\_user', methods=['POST'])

def add\_user():

name= request.form.get('name')

email = request.form.get('email')

password = request.form.get('password')

cursor.execute("""INSERT INTO users(id, name, email, password) VALUES (NULL,'{}','{}','{}')""".format(name,email,password))

conn.commit()

return render\_template('login.html', prediction\_text = "0")

@app.route('/fertilizer-predict', methods=['POST'])

def fert\_recommend():

title = 'Farmer Buddy'

crop\_name = str(request.form['cropname'])

N = int(request.form['nitrogen'])

P = int(request.form['phosphorous'])

K = int(request.form['pottasium'])

# ph = float(request.form['ph'])

df = pd.read\_csv('Data/fertilizer.csv')

nr = df[df['Crop'] == crop\_name]['N'].iloc[0]

pr = df[df['Crop'] == crop\_name]['P'].iloc[0]

kr = df[df['Crop'] == crop\_name]['K'].iloc[0]

n = nr - N

p = pr - P

k = kr – K

k = kr - K

temp = {abs(n): "N", abs(p): "P", abs(k): "K"}

max\_value = temp[max(temp.keys())]

if max\_value == "N":

if n < 0:

key = 'NHigh'

else:

key = "Nlow"

elif max\_value == "P":

if p < 0:

key = 'PHigh'

else:

key = "Plow"

else:

if k < 0:

key = 'KHigh'

else:

key = "Klow"

response = Markup(str(fertilizer\_dic[key]))

return render\_template('fertilizer-result.ht,ml', recommendation=response, title=title)

# render disease prediction result page

@app.route('/disease-predict', methods=['GET', 'POST'])

def disease\_prediction():

title = 'Farmer Buddy'

if request.method == 'POST':

if 'file' not in request.files:

return redirect(request.url)

file = request.files.get('file')

if not file:

return render\_template('disease.html', title=title)

try:

img = file.read()

prediction = predict\_image(img)

prediction = Markup(str(disease\_dic[prediction]))

return render\_template('disease-result.html', prediction=prediction, title=title)

except:

pass

return render\_template('disease.html', title=title)

# if name == '\_main\_':

app.run(debug=False)