

```
# Importing essential libraries and modules
```

```
from flask import Flask, render_template, request, Markup, url_for, redirect
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
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin, login_user, LoginManager, login_required, logout_user,
current_user
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import InputRequired, Length, ValidationError
from flask_bcrypt import Bcrypt
```

```
# =====
```

```
# -----LOADING THE TRAINED MODELS -----
---
```

```
# Loading plant disease classification model
```

```
disease_classes = ['Apple__Apple_scab',
                   'Apple__Black_rot',
                   'Apple__Cedar_apple_rust',
                   'Apple__healthy',
                   'Blueberry__healthy',
                   'Cherry_(including_sour)__Powdery_mildew',
                   'Cherry_(including_sour)__healthy',
                   'Corn_(maize)__Cercospora_leaf_spot Gray_leaf_spot',
                   'Corn_(maize)__Common_rust_',
                   'Corn_(maize)__Northern_Leaf_Blight',
                   'Corn_(maize)__healthy',
                   'Grape__Black_rot',
                   'Grape__Esca_(Black_Measles)',
                   'Grape__Leaf_blight_(Isariopsis_Leaf_Spot)',
                   'Grape__healthy',
                   'Orange__Haunglongbing_(Citrus_greening)',
                   'Peach__Bacterial_spot',
                   'Peach__healthy',
                   'Pepper,_bell__Bacterial_spot',
                   'Pepper,_bell__healthy',
                   'Potato__Early_blight',
                   'Potato__Late_blight',
                   'Potato__healthy',
                   'Raspberry__healthy',
                   'Soybean__healthy',
                   'Squash__Powdery_mildew',
                   'Strawberry__Leaf_scorch',
                   'Strawberry__healthy',
                   'Tomato__Bacterial_spot',
                   'Tomato__Early_blight',
                   'Tomato__Late_blight',
                   'Tomato__Leaf_Mold',
                   'Tomato__Septoria_leaf_spot',
                   'Tomato__Spider_mites Two-spotted_spider_mite',
                   'Tomato__Target_Spot',
                   'Tomato__Tomato_Yellow_Leaf_Curl_Virus',
                   'Tomato__Tomato_mosaic_virus',
                   'Tomato__healthy']
```

```
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()
```

```
# Loading crop recommendation model
```

```
crop_recommendation_model_path = 'models/RandomForest.pkl'
crop_recommendation_model = pickle.load(
    open(crop_recommendation_model_path, 'rb'))
```

```
# =====
```

```
# Custom functions for calculations
```

```
def weather_fetch(city_name):
    """
    Fetch and returns the temperature and humidity of a city
    :params: city_name
    :return: temperature, humidity
    """
    api_key = config.weather_api_key
    base_url = "http://api.openweathermap.org/data/2.5/weather?"

    complete_url = base_url + "appid=" + api_key + "&q=" + city_name
    response = requests.get(complete_url)
    x = response.json()

    if x["cod"] != "404":
        y = x["main"]

        temperature = round((y["temp"] - 273.15), 2)
        humidity = y["humidity"]
        return temperature, humidity
    else:
        return None
```

```
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
```

```
# =====
# ----- FLASK APP -----
```

```
app = Flask(__name__)
```

```

db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
app.config['SECRET_KEY'] = 'thisisasecretkey'

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), nullable=False, unique=True)
    password = db.Column(db.String(80), nullable=False)

class RegisterForm(FlaskForm):
    username = StringField(validators=[
        InputRequired(), Length(min=4, max=20)], render_kw={"placeholder":
"Username"})

    password = PasswordField(validators=[
        InputRequired(), Length(min=8, max=20)], render_kw={"placeholder":
"Password"})

    submit = SubmitField('Register')

    def validate_username(self, username):
        existing_user_username = User.query.filter_by(
            username=username.data).first()
        if existing_user_username:
            raise ValidationError(
                'That username already exists. Please choose a different one.')

class LoginForm(FlaskForm):
    username = StringField(validators=[
        InputRequired(), Length(min=4, max=20)], render_kw={"placeholder":
"Username"})

    password = PasswordField(validators=[
        InputRequired(), Length(min=8, max=20)], render_kw={"placeholder":
"Password"})

    submit = SubmitField('Login')

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user:
            if bcrypt.check_password_hash(user.password, form.password.data):
                login_user(user)
                return redirect(url_for('dashboard'))
    return render_template('login.html', form=form)

```

```
@app.route('/dashboard', methods=['GET', 'POST'])
@login_required
def dashboard():
    return render_template('dashboard.html')

@app.route('/logout', methods=['GET', 'POST'])
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

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

    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data)
        new_user = User(username=form.username.data, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('login'))

    return render_template('register.html', form=form)

# render home page

# render crop recommendation form page

@ app.route('/crop-recommend')
def crop_recommend():
    title = 'Crop Recommendation'
    return render_template('crop.html', title=title)

# render fertilizer recommendation form page

@ app.route('/fertilizer')
def fertilizer_recommendation():
    title = 'Fertilizer Suggestion'

    return render_template('fertilizer.html', title=title)

# render disease prediction input page

# =====

# RENDER PREDICTION PAGES

# render crop recommendation result page

@ app.route('/crop-predict', methods=['POST'])
def crop_prediction():
    title = 'Crop Recommendation'

    if request.method == 'POST':
        N = int(request.form['nitrogen'])
        P = int(request.form['phosphorous'])
        K = int(request.form['pottasium'])
        ph = float(request.form['ph'])
```

```

rainfall = float(request.form['rainfall'])

# state = request.form.get("stt")
city = request.form.get("city")

if weather_fetch(city) != None:
    temperature, humidity = weather_fetch(city)
    data = np.array([[N, P, K, temperature, humidity, ph, rainfall]])
    my_prediction = crop_recommendation_model.predict(data)
    final_prediction = my_prediction[0]

    return render_template('crop-result.html', prediction=final_prediction, title=title)

else:

    return render_template('try_again.html', title=title)

# render fertilizer recommendation result page

@app.route('/fertilizer-predict', methods=['POST'])
def fert_recommend():
    title = 'Fertilizer Suggestion'

    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
    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.html', recommendation=response, title=title)

# render disease prediction result page

@app.route('/disease-predict', methods=['GET', 'POST'])
def disease_prediction():
    title = 'Disease Detection'

    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=True)
```