**from** flask **import** Flask, render\_template, flash, request, session,send\_file  
**from** flask **import** render\_template, redirect, url\_for, request,Markup  
*#from wtforms import Form, TextField, TextAreaField, validators, StringField, SubmitField***from** werkzeug.utils **import** secure\_filename  
**import** datetime  
**import** pandas **as** pd  
**import** pickle  
**import** numpy **as** np  
**import** mysql.connector  
**from** utils.soildata **import** soil\_dic  
**from** utils.fertilizer **import** fertilizer\_dic  
**import** sys  
**import** tensorflow **as** tf  
  
**from** PIL **import** Image, ImageOps  
  
**from** tkinter **import** \*  
**import** os  
**from** tkinter **import** filedialog  
**import** cv2  
**import** time  
**from** matplotlib **import** pyplot **as** plt  
**from** tkinter **import** messagebox  
app = Flask(\_\_name\_\_)  
app.config[**'DEBUG'**]  
app.config[**'SECRET\_KEY'**] = **'7d441f27d441f27567d441f2b6176a'**UPLOAD\_FOLDER = **'static/uploads/'**app.config[**'UPLOAD\_FOLDER'**] = UPLOAD\_FOLDER  
  
  
*# =========================================================================================***import** tensorflow **as** tf  
classifierLoad = tf.keras.models.load\_model(**'SoilNet\_93\_86.h5'**)  
**import** numpy **as** np  
**from** tensorflow.keras.models **import** load\_model  
**from** tensorflow.keras.preprocessing.image **import** load\_img, img\_to\_array  
**import** pickle  
model\_path = **"SoilNet\_93\_86.h5"**SoilNet = load\_model(model\_path)  
ideal\_df = pd.read\_csv(**"fertilizer.csv"**)  
ideal\_df.set\_index(**'Crop'**, inplace=**True**)  
  
classes = {0: **"Alluvial Soil:-{ Rice,Wheat,Sugarcane,Maize,Cotton,Soyabean,Jute }"**,  
 1: **"Black Soil:-{ Virginia, Wheat , Jowar,Millets,Linseed,Castor,Sunflower} "**,  
 2: **"Clay Soil:-{ Rice,Lettuce,Chard,Broccoli,Cabbage,Snap Beans }"**,  
 3: **"Red Soil:{ Cotton,Wheat,Pilses,Millets,OilSeeds,Potatoes }"**}  
  
  
**def** model\_predict(image\_path, model):  
 print(**"Predicted"**)  
 image = load\_img(image\_path, target\_size=(224, 224))  
 image = img\_to\_array(image)  
 image = image / 255  
 image = np.expand\_dims(image, axis=0)  
  
 result = np.argmax(model.predict(image))  
 prediction = classes[result]  
  
 **if** result == 0:  
 print(**"Alluvial.html"**)  
  
 **return "Alluvial"**, **"Alluvial.html"  
 elif** result == 1:  
 print(**"Black.html"**)  
  
 **return "Black"**, **"Black.html"  
 elif** result == 2:  
 print(**"Clay.html"**)  
  
 **return "Clay"**, **"Clay.html"  
 elif** result == 3:  
 print(**"Red.html"**)  
  
 **return "Red"**, **"Red.html"**@app.route(**"/"**)  
**def** homepage():  
  
 **return** render\_template(**'index.html'**)  
  
@app.route(**"/AdminLogin"**)  
**def** AdminLogin():  
  
 **return** render\_template(**'AdminLogin.html'**)  
  
@app.route(**"/about"**)  
**def** about():  
  
 **return** render\_template(**'About.html'**)  
  
  
@app.route(**"/soildata"**)  
**def** soildata():  
  
 **return** render\_template(**'soildata.html'**)  
  
  
@app.route(**"/sdataset"**)  
**def** sdataset():  
  
 **return** render\_template(**'sdataset.html'**)  
  
  
  
  
@app.route(**"/UserLogin"**)  
**def** UserLogin():  
 **return** render\_template(**'UserLogin.html'**)  
  
@app.route(**"/NewUser"**)  
**def** NewUser():  
 **return** render\_template(**'NewUser.html'**)  
  
  
  
  
@app.route(**"/UserHome"**)  
**def** UserHome():  
 user = session[**'uname'**]  
  
 conn = mysql.connector.connect(user=**'root'**, password=**''**, host=**'localhost'**, database=**'1leafdb'**)  
 *# cursor = conn.cursor()* cur = conn.cursor()  
 cur.execute(**"SELECT \* FROM regtb where username='"** + user + **"'"**)  
 data = cur.fetchall()  
 **return** render\_template(**'UserHome.html'**,data=data)  
  
  
@app.route(**"/userlogin"**, methods=[**'GET'**, **'POST'**])  
**def** userlogin():  
  
 **if** request.method == **'POST'**:  
 username = request.form[**'uname'**]  
 password = request.form[**'password'**]  
 session[**'uname'**] = request.form[**'uname'**]  
  
 conn = mysql.connector.connect(user=**'root'**, password=**''**, host=**'localhost'**, database=**'1leafdb'**)  
 cursor = conn.cursor()  
 cursor.execute(**"SELECT \* from regtb where username='"** + username + **"' and Password='"** + password + **"'"**)  
 data = cursor.fetchone()  
 **if** data **is None**:  
  
 alert = **'Username or Password is wrong'  
 return** render\_template(**'goback.html'**, data=alert)  
  
  
  
 **else**:  
 user = session[**'uname'**]  
  
 conn = mysql.connector.connect(user=**'root'**, password=**''**, host=**'localhost'**, database=**'1leafdb'**)  
 *# cursor = conn.cursor()* cur = conn.cursor()  
 cur.execute(**"SELECT \* FROM regtb where username='"** + user + **"'"**)  
 data = cur.fetchall()  
 **return** render\_template(**'UserHome.html'**, data=data)  
  
@app.route(**"/newuser"**, methods=[**'GET'**, **'POST'**])  
**def** newuser():  
 **if** request.method == **'POST'**:  
  
 name1 = request.form[**'name'**]  
 gender1 = request.form[**'gender'**]  
 Age = request.form[**'age'**]  
 email = request.form[**'email'**]  
 pnumber = request.form[**'phone'**]  
 address = request.form[**'address'**]  
  
 uname = request.form[**'uname'**]  
 password = request.form[**'psw'**]  
  
  
 conn = mysql.connector.connect(user=**'root'**, password=**''**, host=**'localhost'**, database=**'1leafdb'**)  
 cursor = conn.cursor()  
 cursor.execute(  
 **"INSERT INTO regtb VALUES ('"** + name1 + **"','"** + gender1 + **"','"** + Age + **"','"** + email + **"','"** + pnumber + **"','"** + address + **"','"** + uname + **"','"** + password + **"')"**)  
 conn.commit()  
 conn.close()  
 *# return 'file register successfully'* **return** render\_template(**'UserLogin.html'**)  
  
@app.route(**"/result"**, methods=[**'GET'**, **'POST'**])  
**def** result():  
  
 **if** request.method == **'POST'**:  
 title = **'Soil Classification and Crop Recommendation '** f = request.files[**'file'**]  
 print(f.filename)  
  
 f.filename = **"test.jpg"** f.save(secure\_filename(f.filename))  
 filename = **'Output/Out/Test.jpg'  
 import** warnings  
  
 warnings.filterwarnings(**'ignore'**)  
  
 **import** tensorflow **as** tf  
  
 *#classifierLoad = tf.keras.models.load\_model('brain.h5')* **import** numpy **as** np  
  
 **from** tensorflow.keras.utils **import** load\_img, img\_to\_array  
 test\_image = load\_img(**'test.jpg'**, target\_size=(128, 128))  
 img1 = cv2.imread(**'test.jpg'**)  
 *# m=classes.index(r[1])* img = cv2.imread(**'test.jpg'**)  
 **if** img **is None**:  
 print(**'no data'**)  
  
 img1 = cv2.imread(**'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)* cv2.imwrite(**'img.jpg'**, img1)  
 gray = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)  
  
 img1S = cv2.resize(img1, (960, 540))  
  
 *#cv2.imshow('Original image', img1S)* cv2.imwrite(**'static/Original\_image.jpg'**, img1S)  
 grayS = cv2.resize(gray, (960, 540))  
 *#cv2.imshow('Gray image', grayS)* cv2.imwrite(**'static/Gray\_image.jpg'**, grayS)  
  
 dst = cv2.fastNlMeansDenoisingColored(img1, **None**, 10, 10, 7, 21)  
 *#cv2.imshow("Nosie Removal", dst)* cv2.imwrite(**'static/Nosie\_Removal.jpg'**, dst)  
  
 thresh = 127  
 im\_bw = cv2.threshold(grayS, thresh, 255, cv2.THRESH\_BINARY)[1]  
 *# cv2.imshow("affect Removal", im\_bw)* cv2.imwrite(**'static/affect\_Removal.jpg'**, im\_bw)  
 img1S\_inverted = cv2.bitwise\_not(img1S)  
 cv2.imwrite(**'static/Original\_image\_inverted.jpg'**, img1S\_inverted)  
  
 *#file = request.files.get('file')  
 #print(file.filename)* img = f.read()  
 file\_path = **"test.jpg"** print(**"@@ Predicting class......"**)  
 pred, output\_page = model\_predict(file\_path, SoilNet)  
 print(pred)  
 prediction = Markup(str(soil\_dic[pred]))  
 **return** render\_template(**'soil-result.html'**, prediction=prediction, title=title)  
  
  
  
  
  
  
  
@app.route(**"/adminlogin"**, methods=[**'GET'**, **'POST'**])  
**def** adminlogin():  
  
 **if** request.method == **'POST'**:  
 username = request.form[**'uname'**]  
 password = request.form[**'password'**]  
 session[**'uname'**] = request.form[**'uname'**]  
  
 conn = mysql.connector.connect(user=**'root'**, password=**''**, host=**'localhost'**, database=**'1leafdb'**)  
 cursor = conn.cursor()  
 cursor.execute(**"SELECT \* from admin where username='"** + username + **"' and Password='"** + password + **"'"**)  
 data = cursor.fetchone()  
 **if** data **is None**:  
  
 alert = **'Username or Password is wrong'  
 return** render\_template(**'goback.html'**, data=alert)  
  
  
  
 **else**:  
 conn = mysql.connector.connect(user=**'root'**, password=**''**, host=**'localhost'**, database=**'1leafdb'**)  
 cursor = conn.cursor()  
 cursor.execute(**"SELECT \* from regtb"**)  
 data = cursor.fetchall()  
  
 **return** render\_template(**'AdminHome.html'**,data=data)  
  
  
  
@app.route(**"/tdataset"**)  
**def** tdataset():  
  
 **return** render\_template(**'tdataset.html'**)  
  
  
@app.route(**"/AdminHome"**)  
**def** AdminHome():  
 conn = mysql.connector.connect(user=**'root'**, password=**''**, host=**'localhost'**, database=**'1leafdb'**)  
 cursor = conn.cursor()  
 cursor.execute(**"SELECT \* from regtb"**)  
 data = cursor.fetchall()  
  
 **return** render\_template(**'AdminHome.html'**,data=data)  
  
@app.route(**"/train"**)  
**def** train():  
 **import** mainmodel  
  
 **return "Train SuccessFull"**@app.route(**"/train1"**)  
**def** train1():  
 **import** pandas **as** pd  
 **from** sklearn.model\_selection **import** train\_test\_split  
 **from** sklearn.ensemble **import** RandomForestClassifier  
 **from** sklearn.metrics **import** classification\_report, confusion\_matrix  
 **import** pickle  
 **import** matplotlib.pyplot **as** plt  
 **import** seaborn **as** sns  
  
 *# Load dataset* df = pd.read\_csv(**'crop\_recommendation.csv'**)  
  
 *# Features and target* X = df.drop(**'label'**, axis=1)  
 y = df[**'label'**]  
  
 *# Train-test split* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
 *# Train model* model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
 model.fit(X\_train, y\_train)  
  
 *# Save model* **with** open(**'crop\_model.pkl'**, **'wb'**) **as** f:  
 pickle.dump(model, f)  
  
 *# Predictions* y\_pred = model.predict(X\_test)  
  
 *# Classification report as text* report = classification\_report(y\_test, y\_pred)  
  
 *# Confusion matrix* cm = confusion\_matrix(y\_test, y\_pred, labels=model.classes\_)  
  
 *# Plot classification report* plt.figure(figsize=(12, 6))  
 plt.axis(**'off'**)  
 plt.title(**"Classification Report"**, fontsize=16, weight=**'bold'**)  
 plt.text(0.01, 0.05, report, {**'fontsize'**: 12}, fontproperties=**'monospace'**)  
 plt.show()  
  
 *# Plot confusion matrix* plt.figure(figsize=(14, 10))  
 sns.heatmap(cm, annot=**True**, fmt=**'d'**, cmap=**'YlGnBu'**,  
 xticklabels=model.classes\_, yticklabels=model.classes\_)  
 plt.title(**"Confusion Matrix for Crop Recommendation"**)  
 plt.xlabel(**"Predicted Crop"**)  
 plt.ylabel(**"Actual Crop"**)  
 plt.xticks(rotation=90)  
 plt.tight\_layout()  
 plt.show()  
  
 **return "Train SuccessFull"**@app.route(**"/upimage"**)  
**def** upimage():  
  
  
 **return** render\_template(**'upimage.html'**)  
  
  
@app.route(**'/predict'**, methods=[**'POST'**])  
**def** predict():  
 model = pickle.load(open(**'crop\_model.pkl'**, **'rb'**))  
 **try**:  
 *# Extract form inputs* features = [float(request.form[x]) **for** x **in** [**'N'**, **'P'**, **'K'**, **'temperature'**, **'humidity'**, **'ph'**, **'rainfall'**]]  
 data = np.array([features])  
  
 *# Predict crop* prediction = model.predict(data)[0]  
 predicted\_crop=prediction  
 ideal\_df = pd.read\_csv(**"fertilizer.csv"**)  
 ideal\_df.set\_index(**'Crop'**, inplace=**True**)  
  
  
  
  
  
  
  
  
  
 **return** render\_template(**'cropresult.html'**, prediction\_text=**f'Recommended Crop: 🌱 {**prediction**}'**,fertilizer\_text=**f'Recommended Crop : 🌱 '**)  
 **except** Exception **as** e:  
 **return** render\_template(**'cropresult.html'**, prediction\_text=**f'Error: {**e**}'**,fertilizer\_text=**''**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app.run(debug=**True**, use\_reloader=**True**)