BUILD PYTHON CODE

```
from __future__ import division, print_function
import os
import numpy as np
import cv2
# Keras
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
# Flask utils
from flask import Flask, request, render_template
from werkzeug.utils import secure_filename
```

Initialization: flask applications must create an application instance. The web server passes all the requests it receives from clients to objects for handling using a protocol for WSG from flask import Flask app = Flask (__name__) (An application instance is an object of class Flask.)

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

MODEL LOADING

model = load_model(“vegetable.h5”)
model1=load_model(“fruit.h5”)

```
model.make_predict_function()
default_image_size = (128, 128)
labels = ["Apple___Black_rot", "Apple___healthy", "Corn_(maize)___healthy",
"Corn_(maize)___Northern_Leaf_Blight", "Peach___Bacterial_spot","Peach___healthy"]
```

```
labels=["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" ]


def convert_image_to_array(image_dir):
 try:
 image = cv2.imread(image_dir)
 if image is not None:
 image = cv2.resize(image, default_image_size)
 return img_to_array(image)
 else:
 return np.array([])
 except Exception as e:
 print(f"Error : {e}")
 return None
def model_predict(file_path, model):
 x = convert_image_to_array(file_path)
 x = np.expand_dims(x, axis=0)
 preds = model.predict(x)
 return preds
```

Routes and View Functions in Flask Framework Instance

Clients send requests to the webserver, in turn, sends them to the Flask application instance. The instance needs to know what code needs to run for each URL requested and map URLs to Python functions. The association between a URL and the function that handles it is called a route. The most convenient way to define a route in a Flask application is through the (app.route). Decorator exposed by the application instance, which registers the ‘decorated function,’ decorators are python feature that modifies the behavior of a function.

```
@app.route("/", methods=['GET'])
def index():
 return render_template("index.html", query="")
```

Request

To process incoming data in Flask, you need to use the request object, including mime-type, IP address, and data. HEAD: Un-encrypted data sent to server w/o response.

GET

Sends data to the server requesting a response body.

POST

Read form inputs and register a user, send HTML data to the server are methods handled by the route. Flask attaches methods to each route so that different view functions can handle different request methods to the same URL.

```
@app.route("/", methods=['GET', 'POST'])
def upload():
 if (request.method == 'POST'):
 f = request.files['file']
 basepath = os.path.dirname(__file__)
 file_path = os.path.join(basepath, 'uploads', secure_filename(f.filename))
 f.save(file_path)
 preds = model_predict(file_path, model)
 preds = np.argmax(preds)
 result = labels[preds]
 return render_template('index.html', prediction_text=result)
 return None
```

Server Startup - The application instance has a ‘run’ method that launches flask’s integrated development webserver -

if __name__ == "__main__":

```
app.run(debug=True)
```