# **Final Project Report**

# **Smart Sorting Using Transfer Learning for identifying rotten fruits and vegetables**

### **1. INTRODUCTION**

#### **1.1 Project Overview**

“Smart Sorting Using Transfer Learning for identifying rotten fruits and vegetables” is an AI-based project developed to classify fruits and vegetables as either healthy or rotten using image classification techniques. By leveraging transfer learning and a pretrained MobileNetV2 model, the system provides high accuracy and real-time results for image-based freshness prediction. The application is deployed via Flask as a web interface where users can upload images and get instant classification results.

#### **1.2 Purpose**

The main goal of this project is to automate post-harvest quality inspection and reduce human effort and inconsistency in sorting agricultural produce. This not only increases productivity but also minimizes food waste and ensures better food supply chain management.

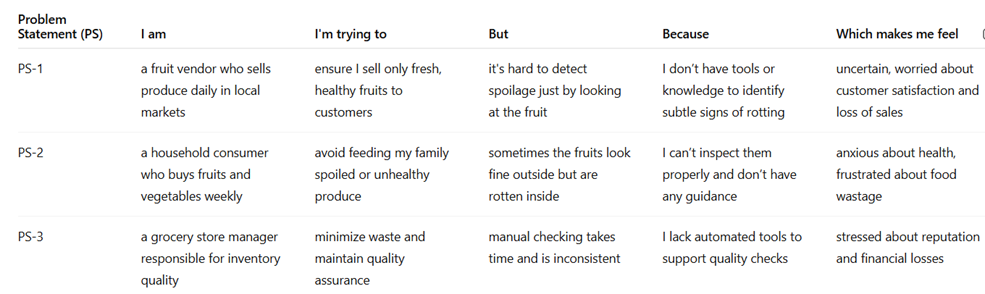
### **2. IDEATION PHASE**

#### **2.1 Problem Statement**

Manual sorting of fruits and vegetables is time-consuming, inefficient, and error-prone. Human inspection is subjective and inconsistent, leading to improper handling, loss of fresh produce, and poor market quality. There’s a need for a consistent, automated classification solution that can work quickly and efficiently.

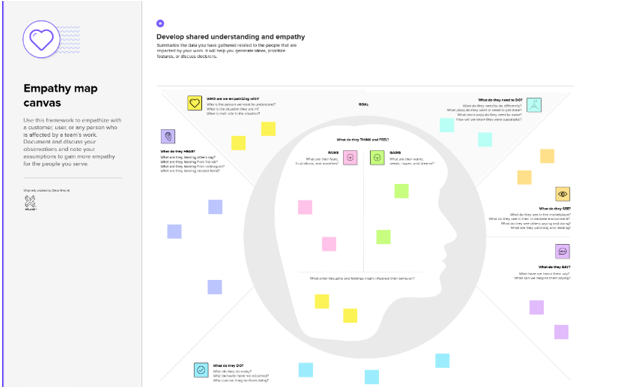
The following are some of the problem statements that have been identifying from perspectives like fruit vendor , a household consumer , a grocery store manager.

Just take a look at the problem statements.



#### **2.2 Empathy Map Canvas**

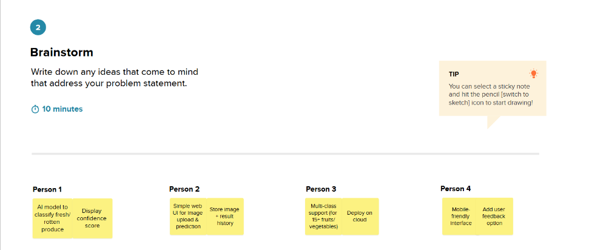
This is the Empathy Map Canvas for sorting rotten fruits and vegetables



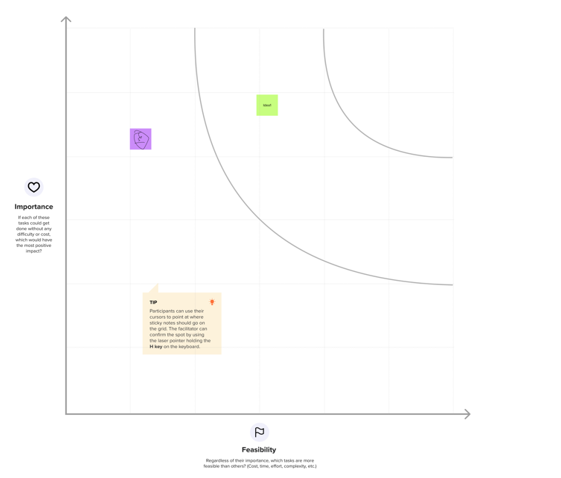
Here is the explanation of the basic terms used in the above Empathy Map Canvas:

* **Think & Feel**: “I need an easier way to detect spoilage.”
* **Hear**: Complaints about damaged produce.
* **See**: Rotten items being packed or sold.
* **Say & Do**: Attempt manual sorting, seek alternatives.
* **Pain**: Labor cost, errors, waste.
* **Gain**: Accurate, fast sorting; better market value.

#### **2.3 Brainstorming**



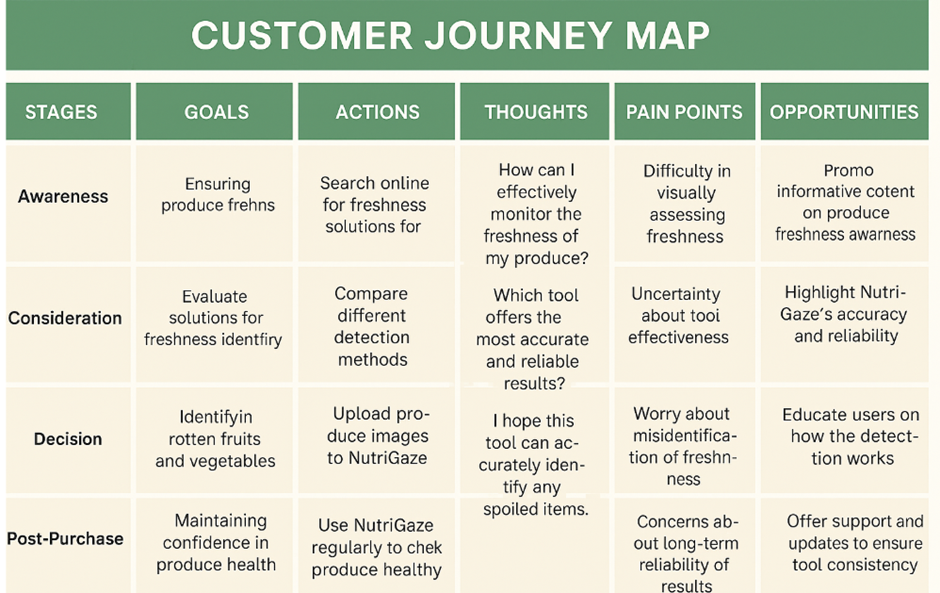
Ideas explored included:

* Sensor-based systems (costly)
* Manual checklists (slow)
* Transfer learning using MobileNetV2 (selected)
* 

Transfer learning stood out for its speed, ease of training, and deployment flexibility.

### **3. REQUIREMENT ANALYSIS**

#### **3.1 Customer Journey Map**



1. Capture/upload image
2. Server processes image
3. Model classifies (healthy/rotten)
4. Result shown to user  
    Users need simplicity, speed, and accuracy at every step.

#### **3.2 Solution Requirement**

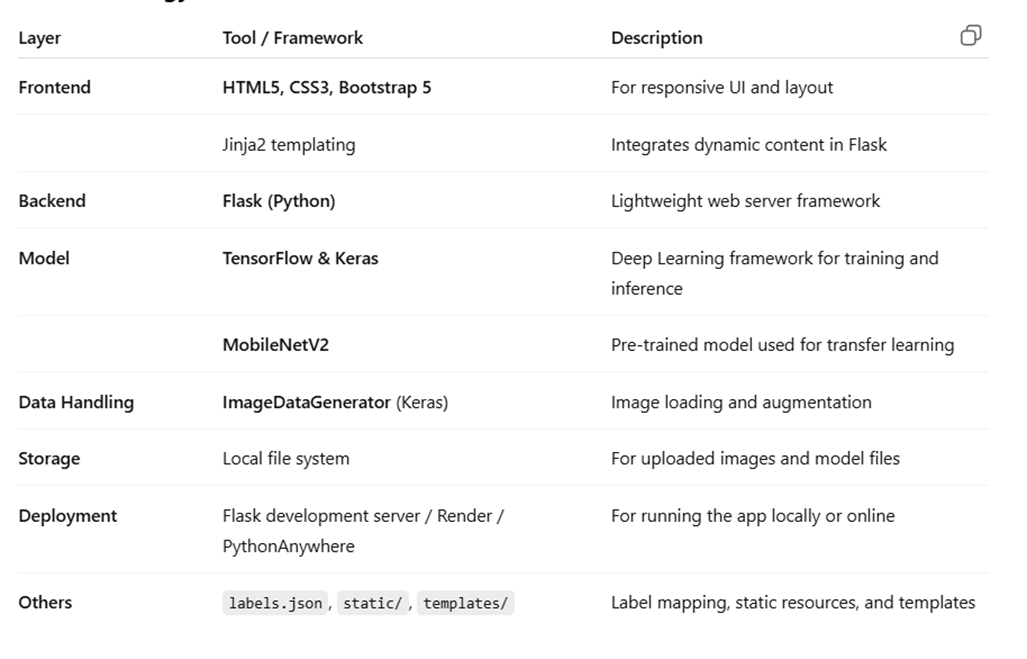
* **Functional**: Upload images, show prediction, navigate UI.
* **Non-Functional**: Fast inference, responsive UI, handle edge cases.

#### **3.3 Data Flow Diagram**

User → Upload Image → Model Prediction → Display Result



#### **3.4 Technology Stack**



### **4. PROJECT DESIGN**

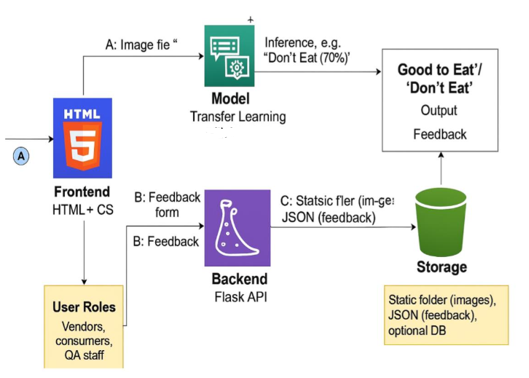
#### **4.1 Problem-Solution Fit**

Problem: Manual sorting is slow and inconsistent.  
 Solution: An AI-powered, web-based classifier using transfer learning.

#### **4.2 Proposed Solution**

Upload an image → resize and normalize → pass to MobileNetV2 → predict class → display result and confidence.

#### **4.3 Solution Architecture**

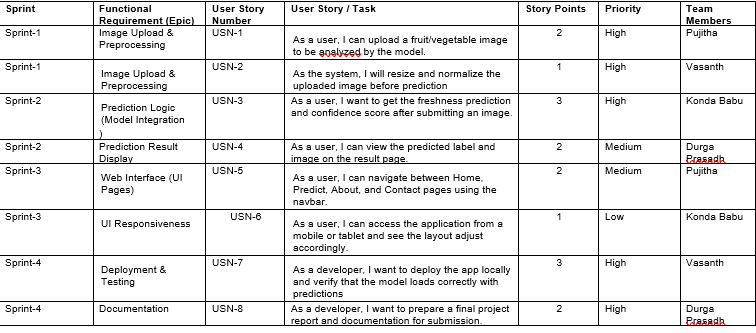


This modular structure supports easy maintenance and expansion.

### **5. PROJECT PLANNING & SCHEDULING**

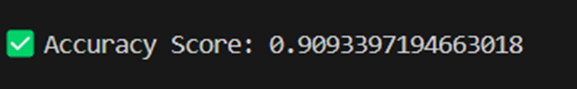
#### **5.1 Project Planning**

**Week-wise breakdown**:



### **6. FUNCTIONAL AND PERFORMANCE TESTING**

#### **6.1 Performance Testing**

* **Accuracy**: ~90%
* **Prediction Time**: ~1.4 seconds
* **File types supported**: JPG, PNG
* **Responsive UI**: Works on mobile and desktop
* 

**Test Cases**:

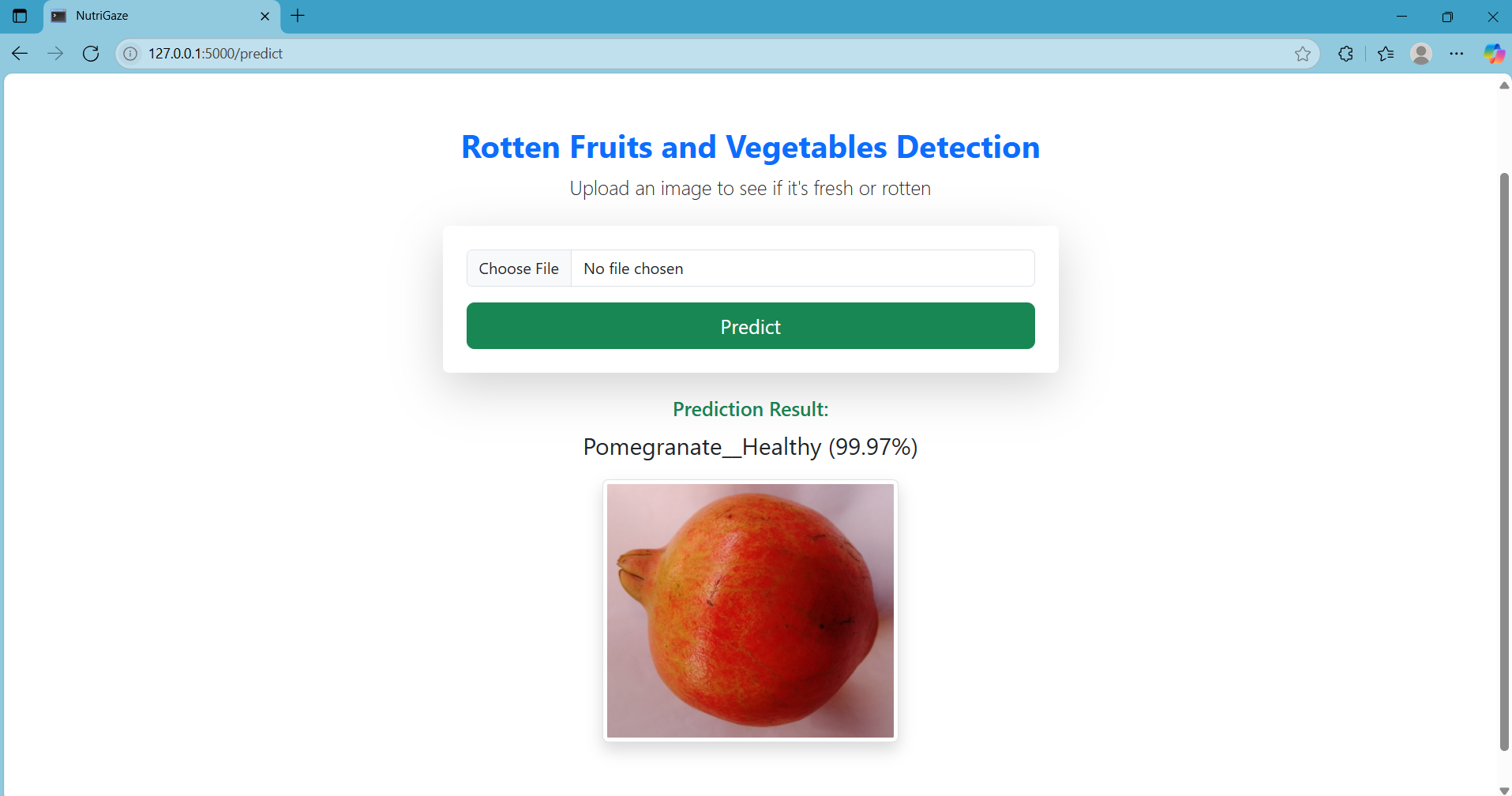
* Valid image upload → accurate result ✅
* Invalid file (PDF, text) → handled error ✅
* Empty form → shown warning ✅
* Large image (>3MB) → rescaled and classified ✅

### **7. RESULTS**

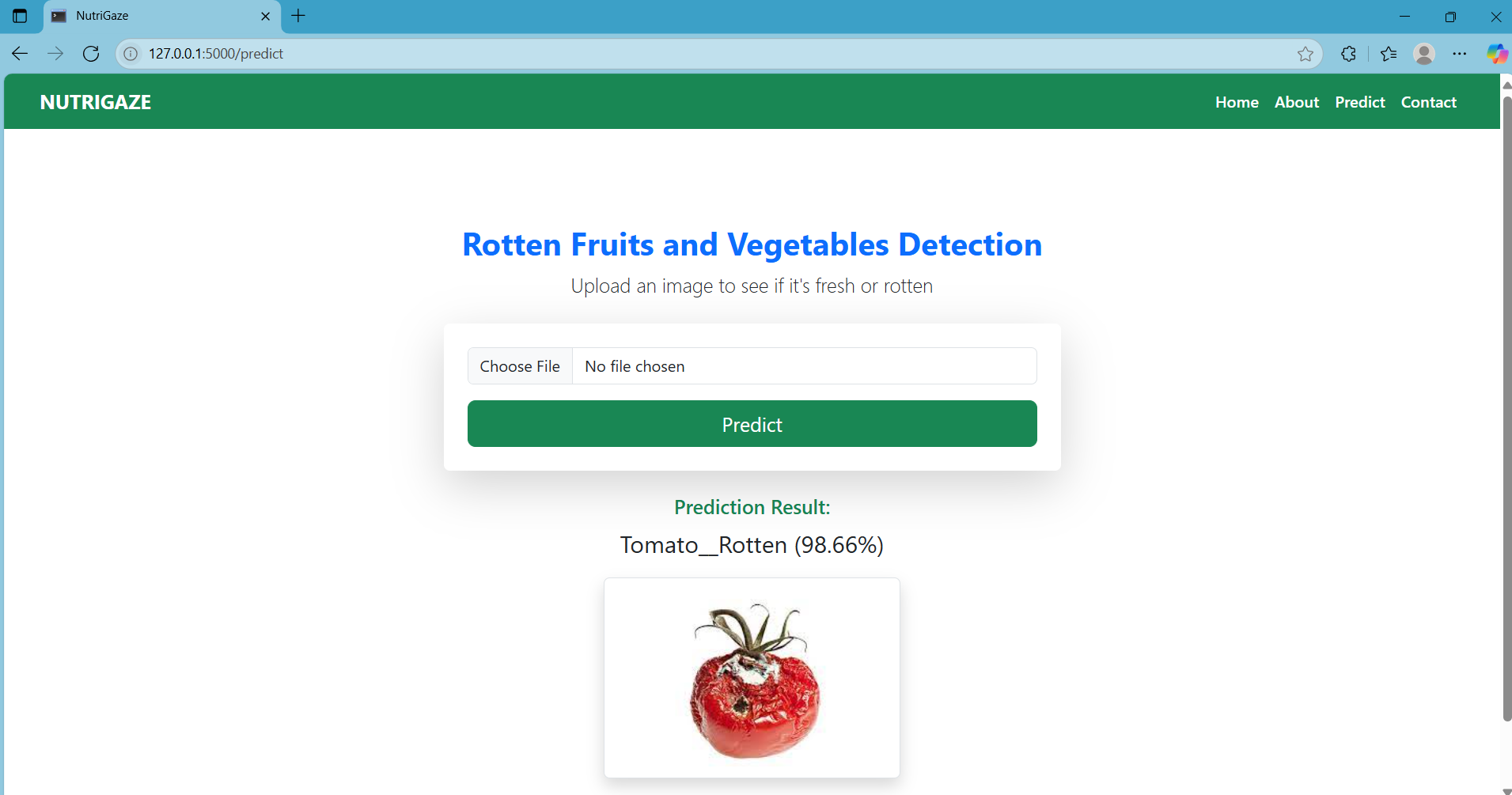
#### **7.1 Output Screenshots**

**Sample Predictions**:

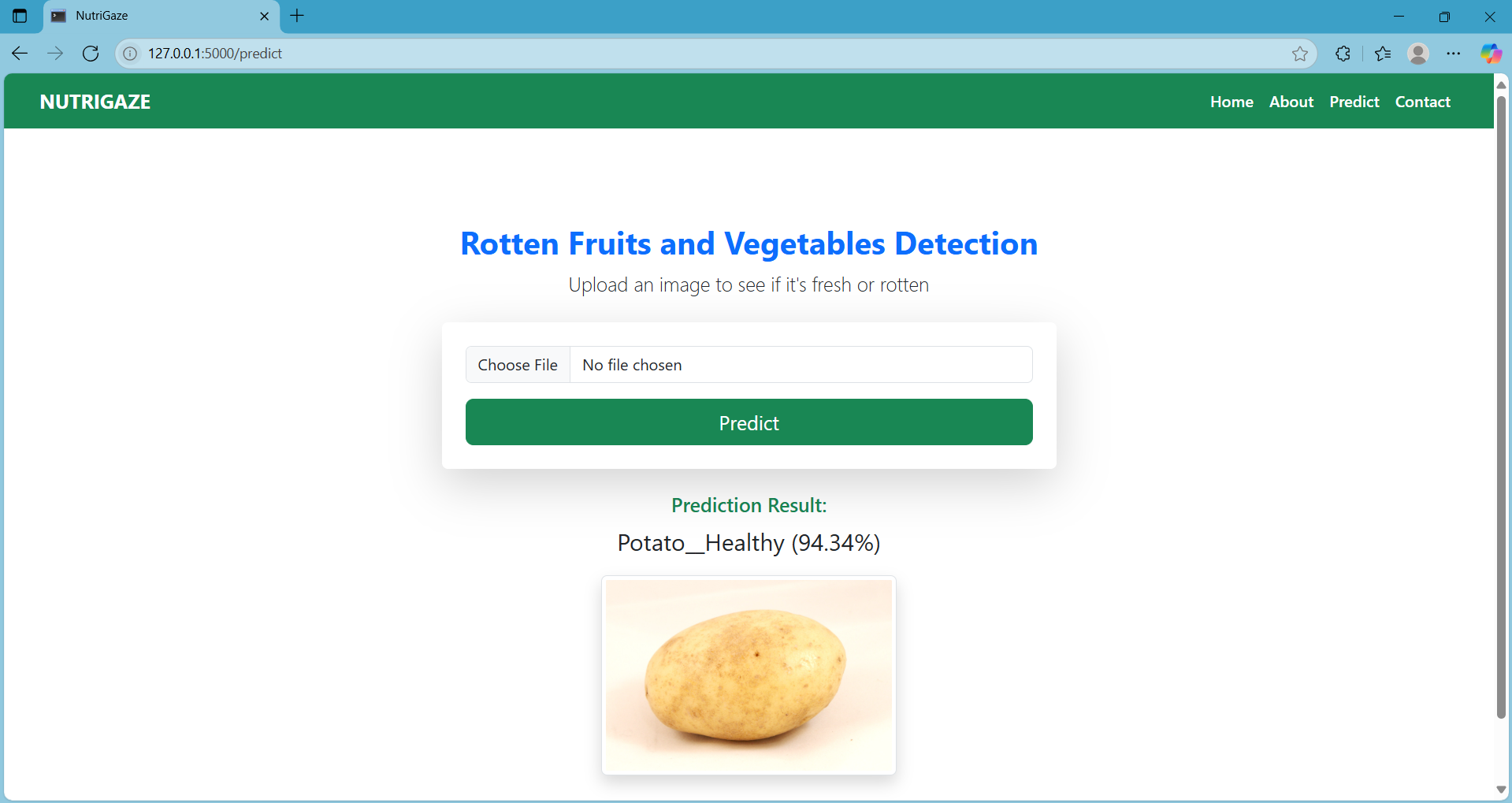
* Promogranate\_Healthy.jpg → Prediction: Apple\_Healthy (98.5%)



* Tomato\_Rotten.png → Prediction: Tomato\_Rotten (98.66%)

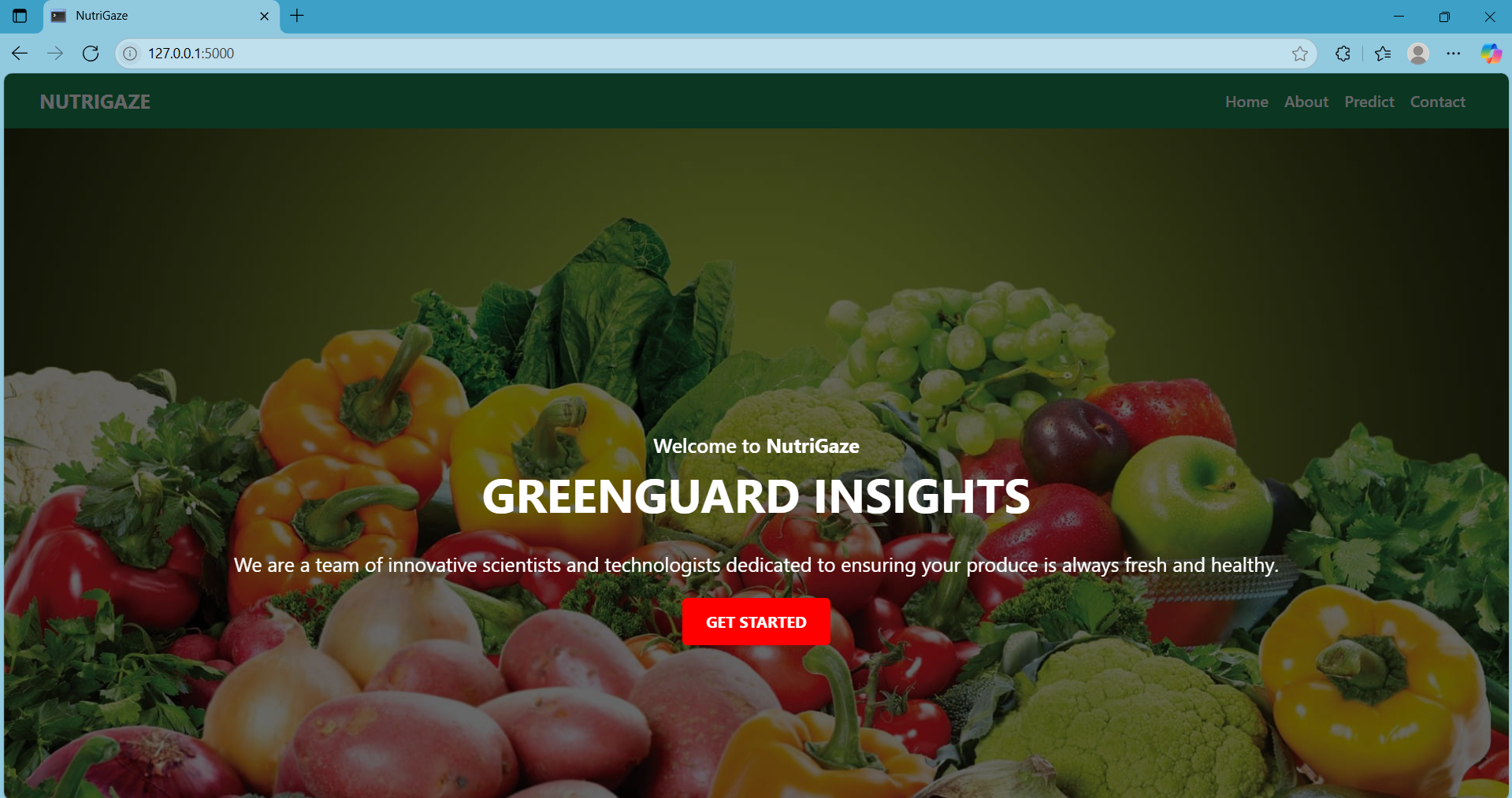


* Potato\_Healthy.jpg → Prediction: Potato\_Healthy (94.34%)

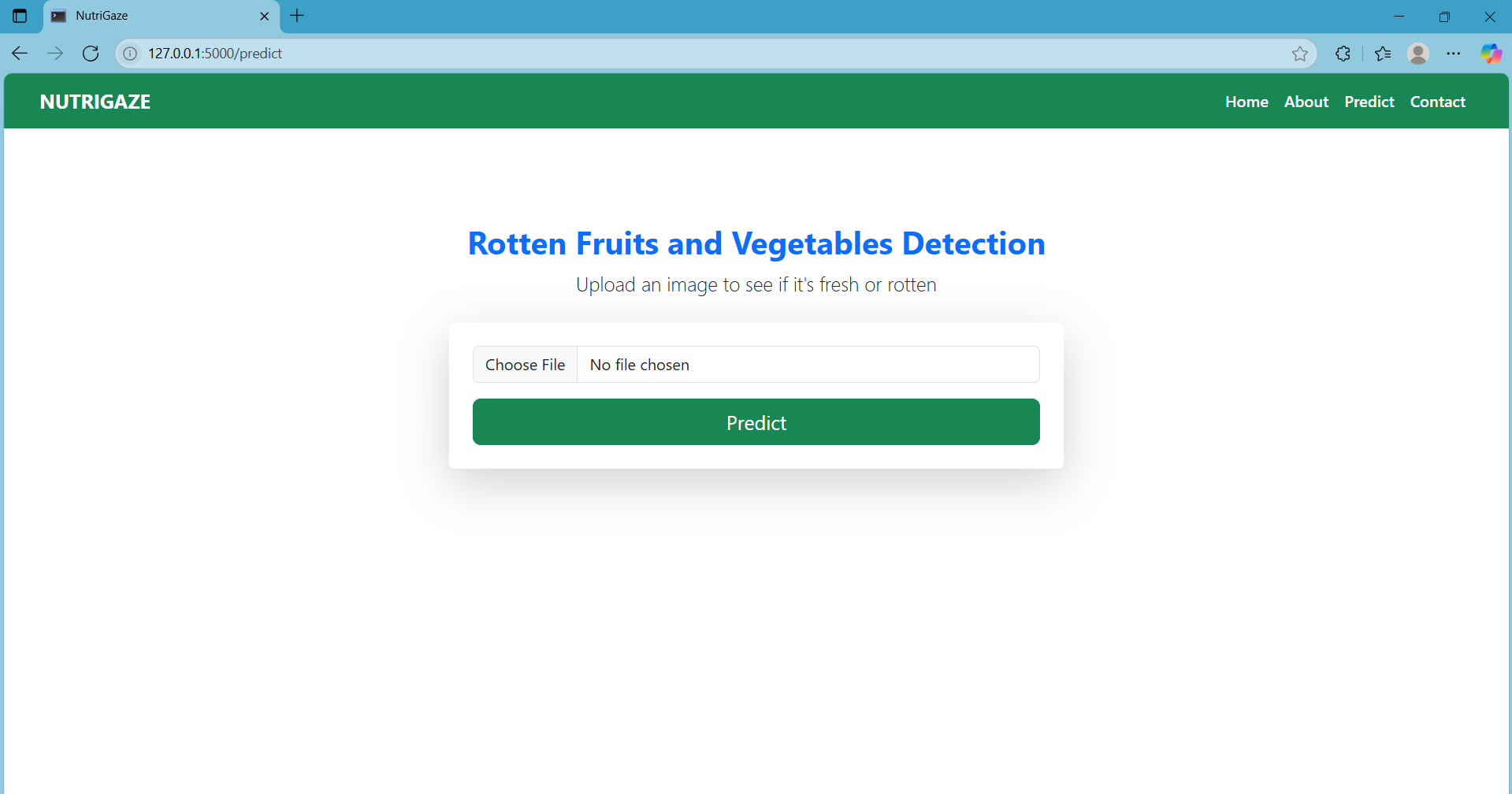


**Interface**:

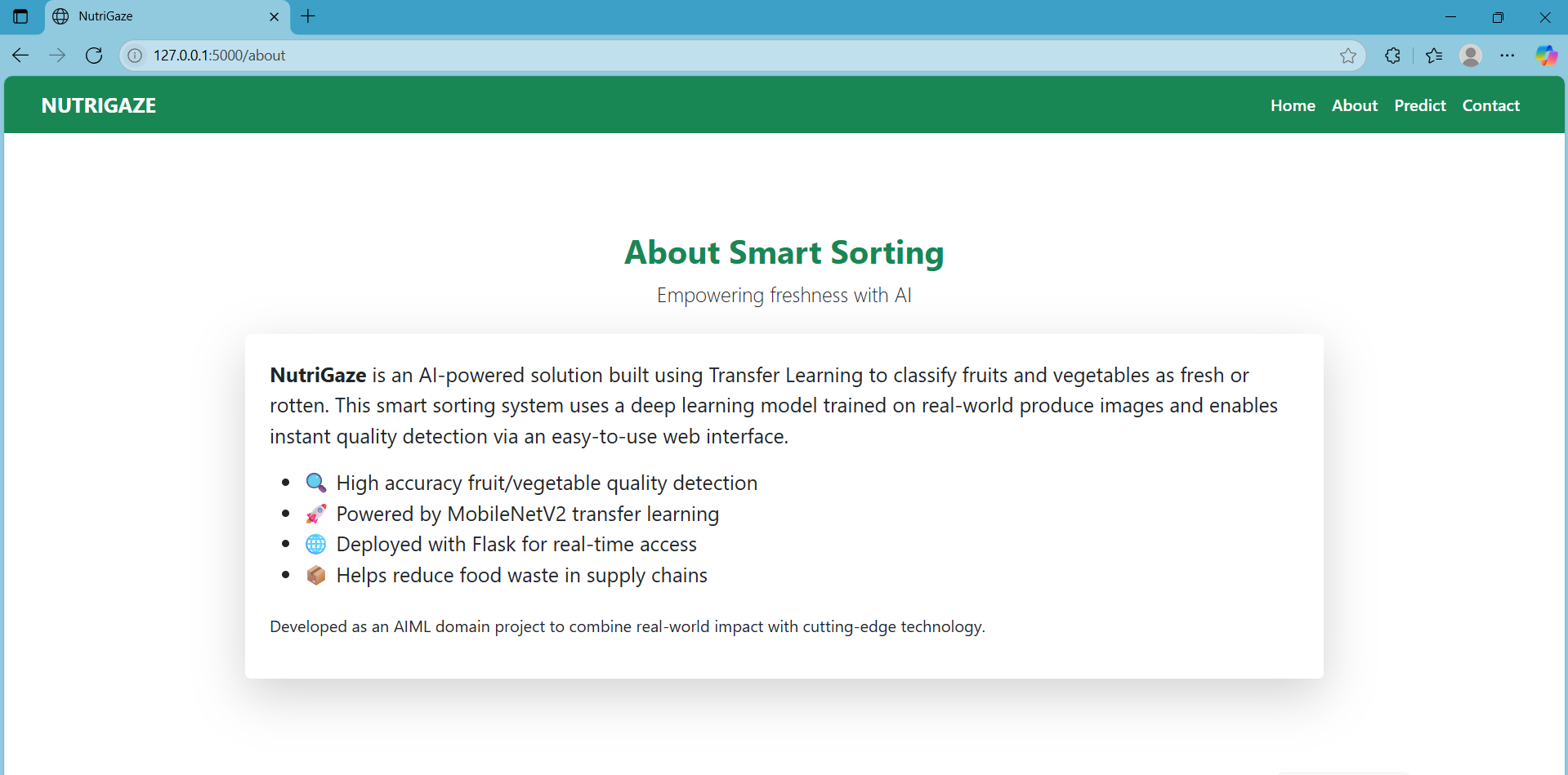
1. ***Home page***



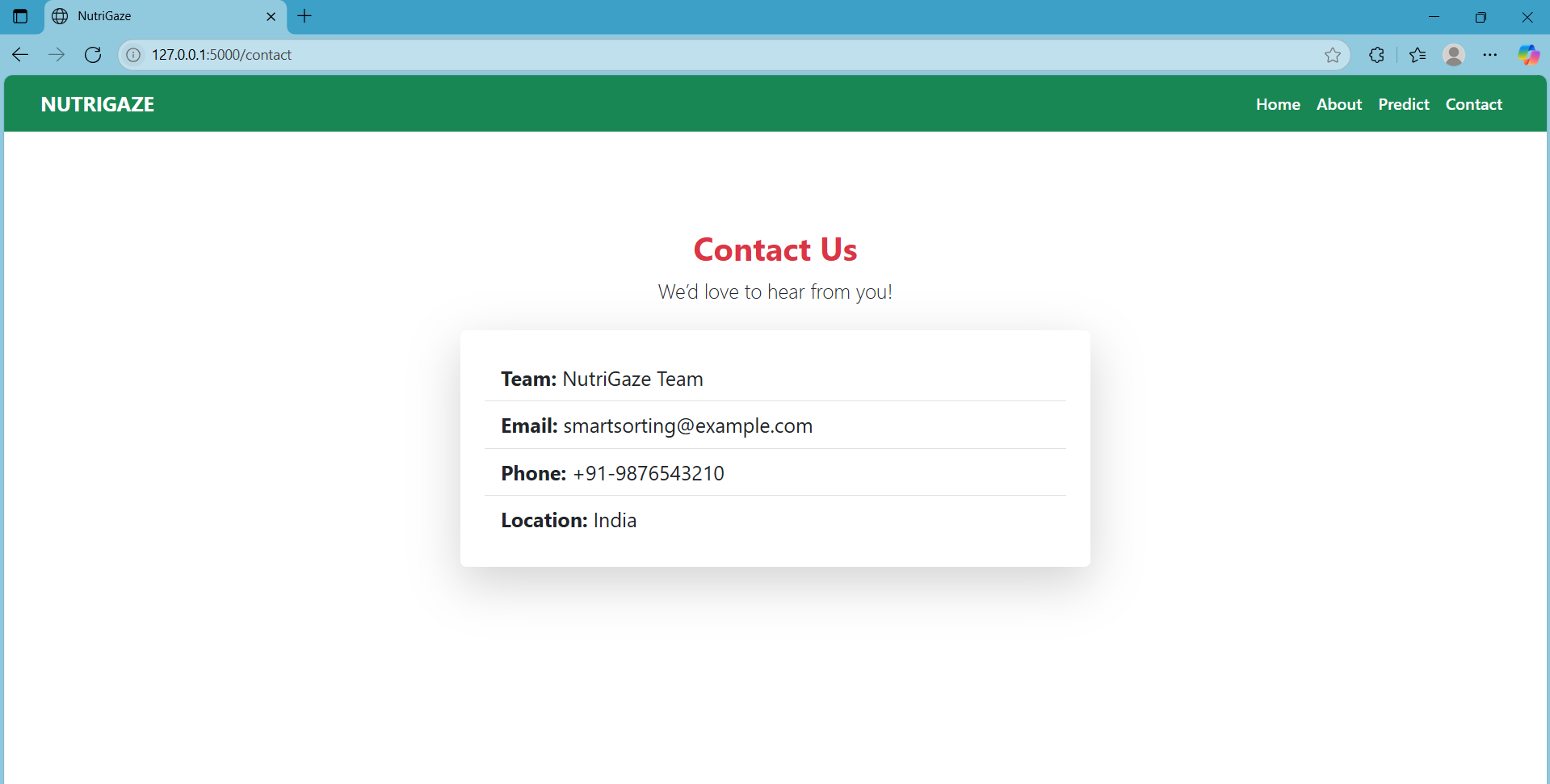
1. ***Predict page***

******

1. ***About Page***

******

1. ***Contact Page***

******

These results validated the model's reliability and usability.

### **8. ADVANTAGES & DISADVANTAGES**

**Advantages**:

* Lightweight MobileNetV2 model
* High accuracy and speed
* Easy to use Flask UI
* Works offline (local deployment)
* Modular and scalable architecture

**Disadvantages**:

* Only binary classification (healthy/rotten)
* No severity detection
* Depends on image quality
* Limited class variety based on dataset

These limitations provide direction for future upgrades.

### **9. CONCLUSION**

The project successfully achieved its objectives of building an AI-based fruit and vegetable sorting system. By combining deep learning with web technologies, we created a solution that is:

* Practical
* Efficient
* Easy to deploy and demonstrate

The system helps automate post-harvest sorting and can greatly assist small farmers, vendors, and supply chain professionals. Through this project, we gained hands-on experience in model training, Flask backend, UI development, and testing.

This solution can serve as a base for commercial applications, research publications, or startup ventures.

### **10. FUTURE SCOPE**

**Upcoming Improvements**:

* Multi-level spoilage detection
* Confidence heatmaps for explainability
* Mobile app (React Native or Flutter)
* Raspberry Pi deployment (edge AI)
* Integration with market/inventory systems
* Cloud deployment (Render, AWS, etc.)

These features will expand usability and commercial potential.

### **11. APPENDIX**

**Source Code**

**# train\_model.py**

**# train\_model.py**

**import os**

**import json**

**import numpy as np**

**import tensorflow as tf**

**from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from tensorflow.keras.preprocessing.image import ImageDataGenerator**

**from tensorflow.keras.applications import MobileNetV2**

**from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout**

**from tensorflow.keras.models import Model**

**from tensorflow.keras.optimizers import Adam**

**# Constants**

**IMG\_SIZE = 224**

**BATCH\_SIZE = 32**

**EPOCHS = 10**

**DATA\_DIR = "Fruit and Vegetable Diseases Dataset"**

**MODEL\_PATH = "model.h5"**

**LABELS\_PATH = "labels.json"**

**# Load and prepare data**

**datagen = ImageDataGenerator(**

**rescale=1./255,**

**validation\_split=0.2,**

**rotation\_range=20,**

**zoom\_range=0.2,**

**horizontal\_flip=True**

**)**

**train\_gen = datagen.flow\_from\_directory(**

**DATA\_DIR,**

**target\_size=(IMG\_SIZE, IMG\_SIZE),**

**batch\_size=BATCH\_SIZE,**

**subset="training",**

**class\_mode="categorical"**

**)**

**val\_gen = datagen.flow\_from\_directory(**

**DATA\_DIR,**

**target\_size=(IMG\_SIZE, IMG\_SIZE),**

**batch\_size=BATCH\_SIZE,**

**subset="validation",**

**class\_mode="categorical",**

**shuffle=False # Required for correct evaluation**

**)**

**# Save label mapping**

**with open(LABELS\_PATH, "w") as f:**

**json.dump(train\_gen.class\_indices, f)**

**# Build model**

**base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(IMG\_SIZE, IMG\_SIZE, 3))**

**base\_model.trainable = False**

**x = base\_model.output**

**x = GlobalAveragePooling2D()(x)**

**x = Dropout(0.3)(x)**

**predictions = Dense(train\_gen.num\_classes, activation='softmax')(x)**

**model = Model(inputs=base\_model.input, outputs=predictions)**

**model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy'])**

**# Train model**

**model.fit(train\_gen, validation\_data=val\_gen, epochs=EPOCHS)**

**# Save model**

**model.save(MODEL\_PATH)**

**print("✅ Model trained and saved as model.h5")**

**# ====================**

**# ✅ Evaluation Section**

**# ====================**

**# Predict on validation set**

**Y\_pred = model.predict(val\_gen)**

**y\_pred = np.argmax(Y\_pred, axis=1)**

**y\_true = val\_gen.classes**

**labels = list(val\_gen.class\_indices.keys())**

**# Accuracy Score**

**acc = accuracy\_score(y\_true, y\_pred)**

**print("\n✅ Accuracy Score:", acc)**

**# Classification Report**

**report = classification\_report(y\_true, y\_pred, target\_names=labels)**

**print("\n✅ Classification Report:\n", report)**

**# Confusion Matrix**

**cm = confusion\_matrix(y\_true, y\_pred)**

**plt.figure(figsize=(14, 10))**

**sns.heatmap(cm, annot=False, cmap="Blues", xticklabels=labels, yticklabels=labels)**

**plt.title("Confusion Matrix")**

**plt.xlabel("Predicted")**

**plt.ylabel("Actual")**

**plt.xticks(rotation=90)**

**plt.tight\_layout()**

**plt.savefig("confusion\_matrix.png")**

**plt.show()**

# app.py

from flask import Flask, render\_template, request

import tensorflow as tf

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

import numpy as np

import json

import os

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

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

# Load label mappings

with open("labels.json", "r") as f:

class\_indices = json.load(f)

labels = {v: k for k, v in class\_indices.items()}

def predict\_image(image\_path):

img = load\_img(image\_path, target\_size=(224, 224))

img\_array = img\_to\_array(img) / 255.0

img\_array = np.expand\_dims(img\_array, axis=0)

prediction = model.predict(img\_array)[0]

class\_index = np.argmax(prediction)

class\_name = labels[class\_index]

confidence = prediction[class\_index] \* 100

return f"{class\_name} ({confidence:.2f}%)"

@app.route('/')

def home():

return render\_template("index.html")

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

def predict():

result = None

if request.method == "POST":

image = request.files["image"]

if image:

filepath = os.path.join("static/uploads", image.filename)

image.save(filepath)

result = predict\_image(filepath)

return render\_template("predict.html", prediction=result, image\_path=filepath)

return render\_template("predict.html", prediction=result)

@app.route('/about')

def about():

return render\_template("about.html")

@app.route('/contact')

def contact():

return render\_template("contact.html")

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**Dataset**:

* [Fruit and Vegetable Disease: Healthy vs Rotten – Kaggle](https://www.kaggle.com/datasets/muhammad0subhan/fruit-and-vegetable-disease-healthy-vs-rotten)
* 28 classes (14 × 2)

**GitHub & Demo**:

* Repository: https://github.com/pujithaparasa/Smart-Sorting-Transfer-Learning-for-identifying-rotten-fruits-and-vegetables

**Directory Structure**:

