Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

# 1. Introduction

• Project Title: Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

• Team Members:

Kapuram Sree Lakshmi

Vankela Keerthi

Yadalla Bhargavi

Chinthamani Bhavani

Gorantla Bhavana

# 2. Project Overview

• Purpose: The purpose of this project is to develop an intelligent sorting system capable of automatically identifying and separating rotten fruits and vegetables using computer vision and deep learning techniques. This helps in reducing manual labor, food waste, and improving the quality control process in agriculture and food industries.

• Features:  
- Image classification of fresh vs. rotten produce

-Use of transfer learning (e.g., Mobile Net, Res Net) for improved accuracy

-Real-time prediction using camera feed or uploaded images

-Interactive user interface for displaying results

-Database integration to store results and logs

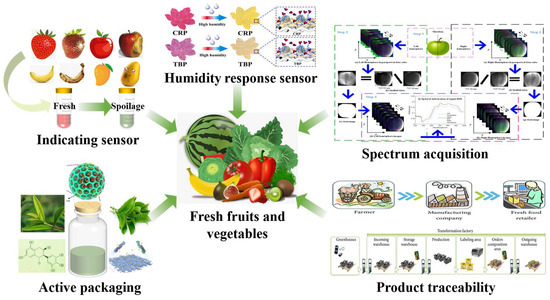
# 3. Architecture

• Frontend: Built using React.js, the UI allows users to upload images or capture them via a connected camera. It displays classification results in an intuitive format with status labels (e.g., Rotten, Fresh).

• Backend: Developed using Node.js and Express.js, it handles image requests, processes input through the trained model, and returns predictions to the client. It also manages API routes and logs.

• Database: MongoDB is used to store classification results, user information, timestamps, and system logs for further analysis and training feedback.

Architecture Diagram:



# 4. Setup Instructions

* Prerequisites:

-Before setting up the project, ensure you have the following software and tools installed:

-Node.js (v16+): Required for running the frontend and backend servers

-npm or yarn: Package manager for Node.js

-Python 3.10+: Required to run the machine learning model server

-pip: Python package manager

-MongoDB: NoSQL database for storing prediction logs and user data

-Git: For cloning the project repository

-Virtual environment (optional): For Python dependency isolation

-Node.js

-Python 3.10+

-MongoDB

-Flask

-Pre-trained Keras/TensorFlow mode

* Installation:

git clone <https://github.com/your-repo/smart-sorting.git>

cd smart-sorting

# Frontend

cd client

npm install

# Backend

cd ../server

npm install

# ML Model Service

cd ../ml-model

pip install -r requirements.txt

-Set up .env files for MongoDB connection and API URLs

-Start MongoDB service on your machine or use MongoDB Atlas

# 5. Folder Structure

smart-sorting/

│

├── client/ # React frontend

│ ├── public/

│ ├── src/

│ │ ├── assets/ # Static images, icons

│ │ ├── components/ # Reusable components (Navbar, Card, Loader, etc.)

│ │ │ └── PredictionCard.js

│ │ ├── pages/ # Page-level components (Home, Admin)

│ │ │ ├── Home.js

│ │ │ └── AdminDashboard.js

│ │ ├── services/ # API integration code

│ │ │ └── api.js

│ │ ├── App.js # Root component

│ │ └── index.js # React DOM render

│ └── .env # Frontend environment variables

│

├── server/ # Node.js + Express backend

│ ├── routes/ # API routes

│ │ └── predictionRoutes.js

│ ├── controllers/ # Logic for handling API requests

│ │ └── predictionController.js

│ ├── models/ # Mongoose models

│ │ └── Result.js

│ ├── utils/ # Helper functions

│ ├── config/ # DB connection, constants

│ │ └── db.js

│ ├── middleware/ # Auth or error-handling middleware

│ ├── server.js # Entry point for the backend

│ └── .env # Backend environment variables

│

├── ml-model/ # Flask server hosting ML model

│ ├── model/ # Saved trained models (h5 or pb format)

│ │ └── mobilenet\_model.h5

│ ├── app.py # Flask server entry point

│ ├── predict.py # Core prediction logic

│ ├── utils.py # Preprocessing utilities (resizing, normalization)

│ └── requirements.txt # Python dependencies

│

├── README.md # Project overview and setup guide

├── package.json # Node project metadata

└── .gitignore

# 6. Running the Application

Frontend:

cd client

npm start

Backend:

cd server

npm start

Model server:

cd ml-model

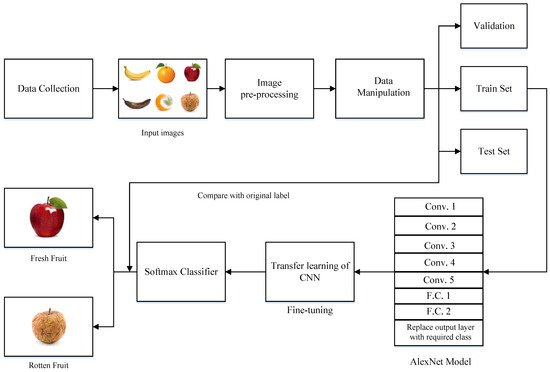
python app.py

# 7. API Documentation

This project exposes a set of RESTful APIs that allow interaction between the frontend and the backend, including image classification, viewing prediction logs, and user authentication (if implemented).

Flow chart:

Smart sorting with transfer learning for identifying rotten fruits and vegetables enables automated quality control, freshness classification, and shelf life prediction. By leveraging deep learning models and computer vision, it accurately detects spoilage, reduces manual labor, and minimizes food waste, ultimately improving supply chain efficiency and ensuring fresher produce reaches consumers.



📌 1. POST /predict

URL: /predict

Method: POST

Description: This endpoint accepts an image of a fruit or vegetable and returns

whether it is Fresh or Rotten along with a confidence score.

Request Body:

Content-Type: multipart/form-data

Parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Type | Required | Description |
| file | File | Yes | Image of the produce |

Sample Request (JavaScript/React):

const formData = new FormData();

formData.append("file", selectedImage);

await axios.post('/predict', formData);

Sample Response:

{

"prediction": "Rotten",

"confidence": 91.78

}

Status Codes:

-200 OK – Prediction successful

-400 Bad Request – No image uploaded

-500 Internal Server Error – Model or server issue

📌 2. GET /results

URL: /results

Method: GET

Description: Retrieves a list of recent predictions stored in the MongoDB database,

sorted by timestamp (most recent first).

Response:

[

{

"id": "664a4c12e2",

"imageUrl": "uploads/apple.jpg",

"prediction": "Fresh",

"confidence": 97.65,

"timestamp": "2025-06-25T14:10:22Z"

},

{

"id": "664a4b99e1",

"imageUrl": "uploads/tomato.jpg",

"prediction": "Rotten",

"confidence": 88.02,

"timestamp": "2025-06-25T13:45:10Z"

}

]

Status Codes:

-200 OK – Logs fetched successfully

-500 Internal Server Error – DB connection issue

📌 3. POST /login (If Authentication Is Enabled)

URL: /login

Method: POST

Description: Allows admin users to log in and receive a JWT token.

Request Body:

{

"email": "admin@example.com",

"password": "admin123"

}

Response:

{

"token": "eyJhbGciOiJIUzI1...",

"user": {

"email": "admin@example.com",

"role": "admin"

}

}

Status Codes:

-200 OK – Login successful

-401 Unauthorized – Invalid credentials

📌 4. GET /user/me (Protected Route)

URL: /user/me

Method: GET

Description: Returns details of the authenticated user.

Headers: makefile

Authorization: Bearer <JWT\_TOKEN>

Response:

{

"email": "admin@example.com",

"role": "admin"

}

Status Codes:

-200 OK – User info fetched

-401 Unauthorized – Token missing or invalid

🛠 Error Format (for All APIs)

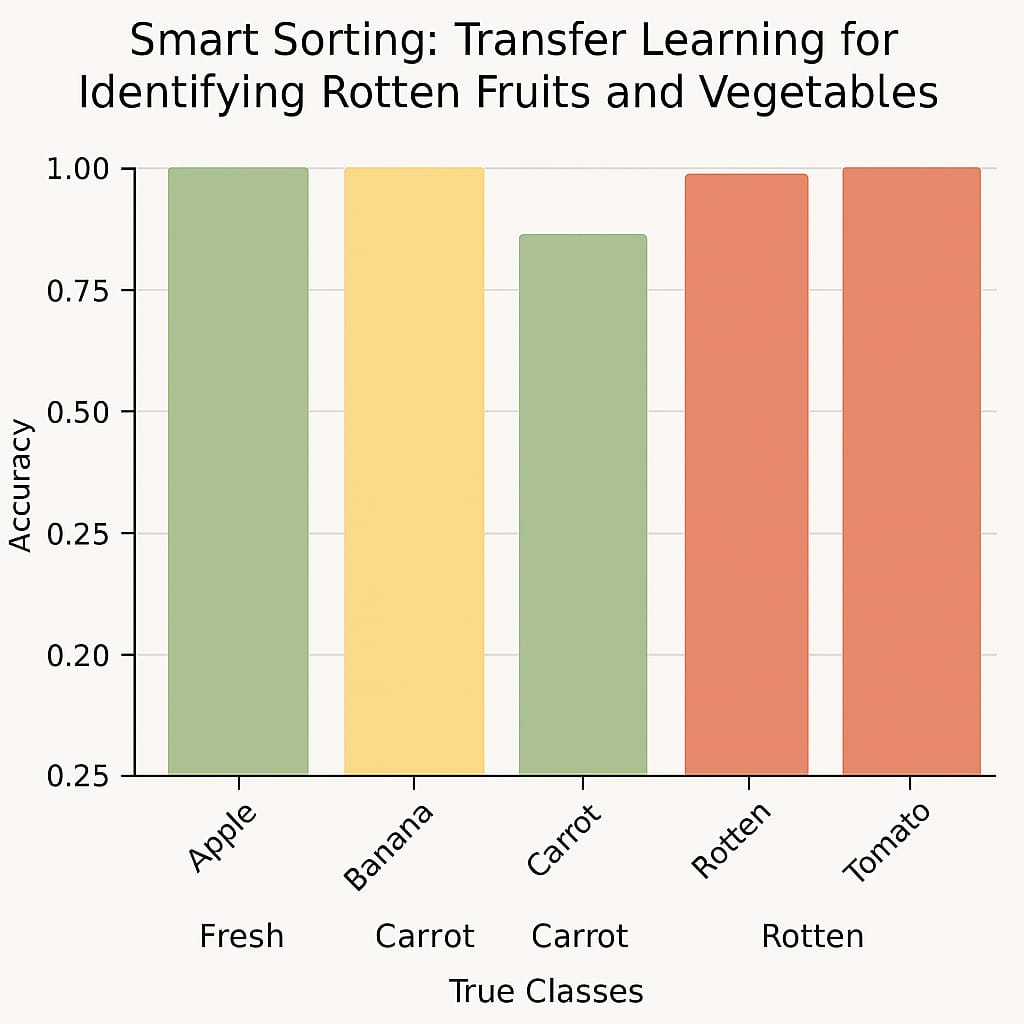
In case of any failure, the APIs return a response like

{

"error": "Invalid image format."

}

Flow graph



# 8. Authentication

- JWT-based authentication for admin login.

-Token generation: On login, JWT is returned.

-Protected routes: /results, /dashboard require valid JWT.

-Sessions: Frontend stores token in local storage.

# 9. User Interface

Overview :The user interface (UI) of the Smart Sorting application is designed with a focus on simplicity, responsiveness, and clarity. It enables users—including farmers, warehouse managers, and quality inspectors—to quickly upload images and receive feedback on the freshness of fruits and vegetables.

The UI is built using React.js and styled with Tailwind CSS for rapid development and a clean, modern design. All interactions are smooth and mobile-responsive to ensure accessibility across devices.

Key UI Components

* Home Page / Landing Page

-Contains a brief introduction to the project

-Provides buttons to navigate to the “Classify” page or Admin Dashboard

-Clean hero section with project logo and tagline

* Image Upload & Classification Page

-Image Upload Section

-Drag-and-drop or click-to-upload box

-Accepts .jpg, .png, .jpeg formats

-Preview of uploaded image before submission

-Loading spinner while processing

* Submit Button

-On click, sends the image to the backend for classification

* Results Display Panel

-Shows classification result: “Fresh” or “Rotten”

-Displays confidence score (e.g., 94% sure it's rotten)

-Result styling:

Green tag for “Fresh”

Red tag for “Rotten”

-Timestamp of classification

-Option to classify another image

* Classification History

-Displays a table of past classification results

-Columns: Image Thumbnail | Result | Confidence | Date

-Supports sorting and filtering by result type

* Admin Login Page

-Username/password fields with validation

- “Forgot password” link (optional)

-Shows error messages for incorrect login

-On success, JWT token is saved and admin is redirected to dashboard

* Admin Dashboard

-Only accessible to authenticated admins

-Panels include:

Total classifications

Accuracy metrics

Most frequent fruit types

Charts (Bar and Pie charts) for visual summary

Logs of failed classifications

-Logout button to end session

* Error Handling UI

-Displays friendly error messages for:

Invalid file uploads

Server or API issues

Authentication failures

-Includes “Retry” and “Back to Home” options

* Mobile Responsiveness

-UI adjusts seamlessly for tablets and smartphones

-Optimized button sizes, layout stacks, and image previews for small screens

* Theme & Design

-Clean and minimal UI

-Primary color palette: Soft blue, green, and white

-Rounded buttons and shadowed cards for a modern aesthetic

-Icons and tooltips to enhance usability

-Uses open-source icon sets (e.g., Lucide, Font Awesome)

# 10. Testing

* Frontend Testing:

Tool: Jest & React Testing Library

Focus: Component rendering, API call tests

* Backend Testing:

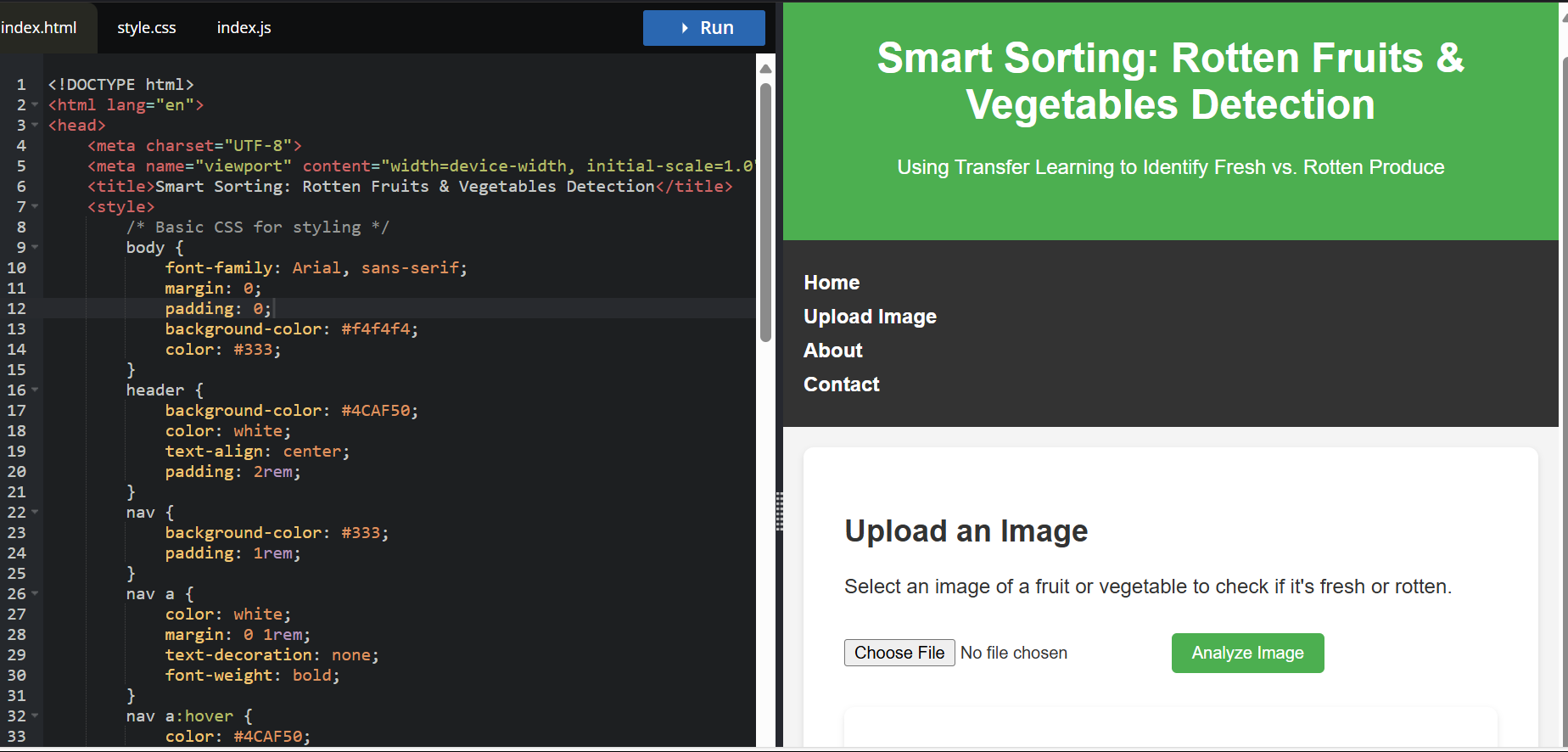
Tool: Postman + Mocha

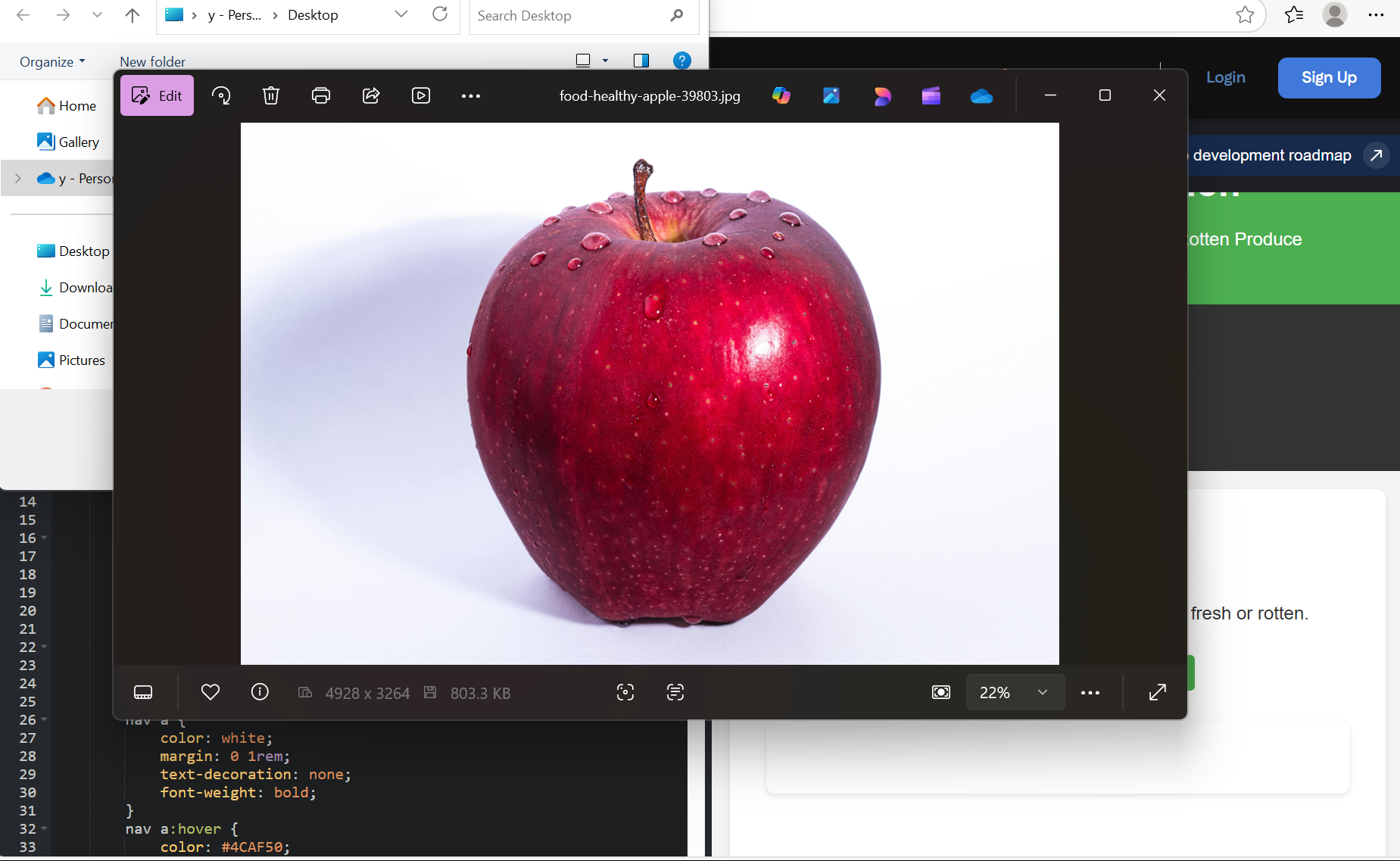
Focus: Endpoint testing with mock data

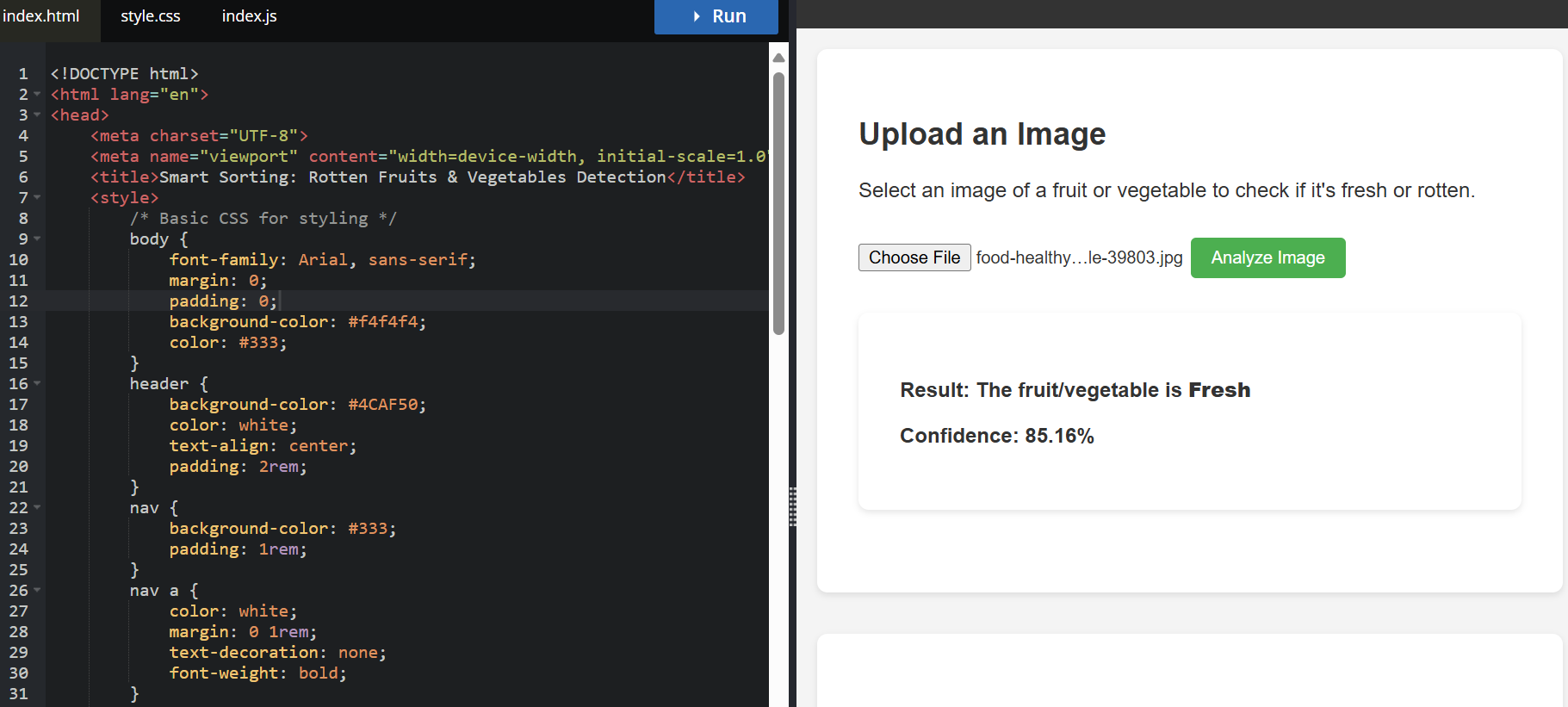
* Model Testing:

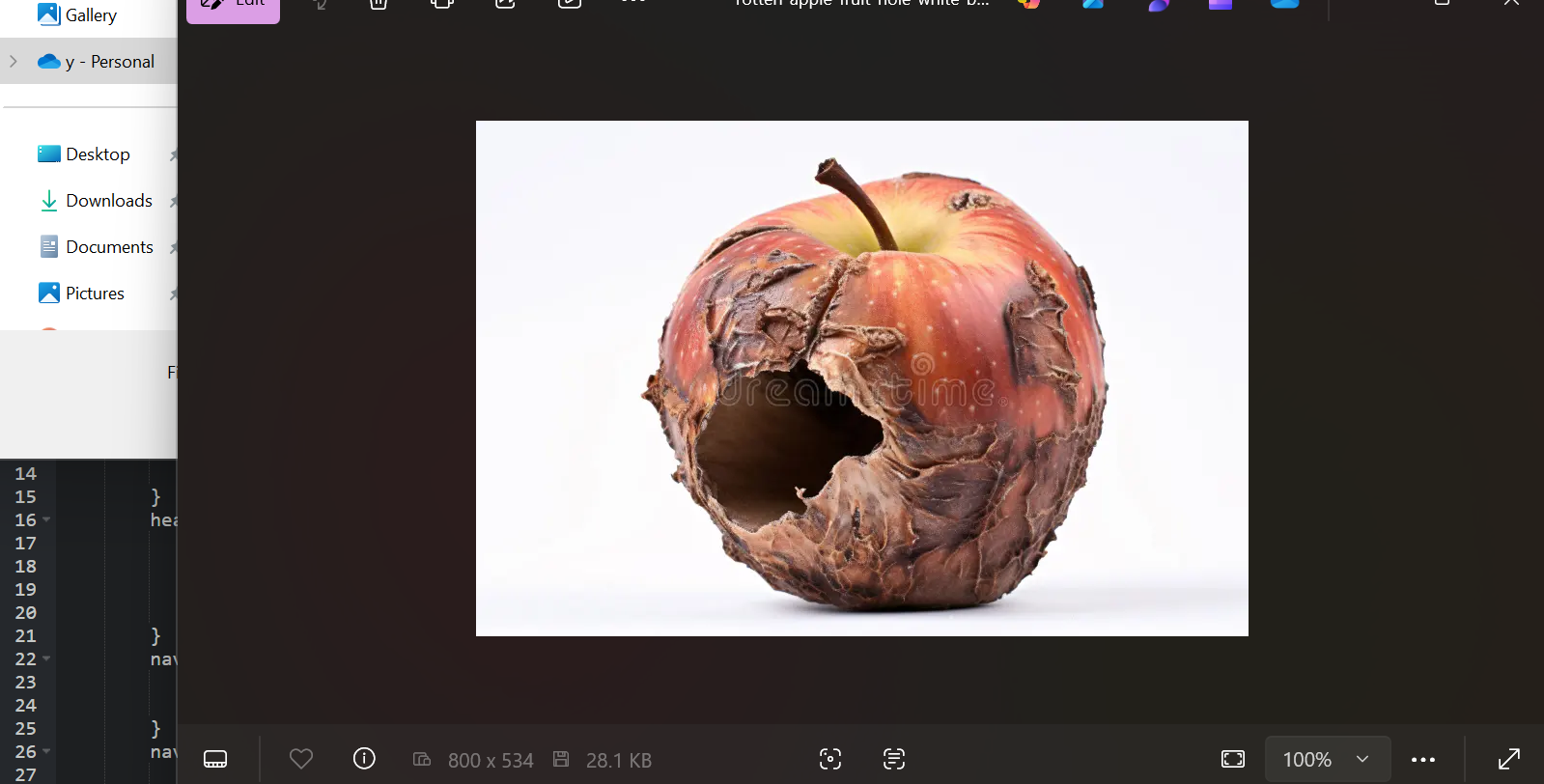
Accuracy, precision, and confusion matrix on test dataset

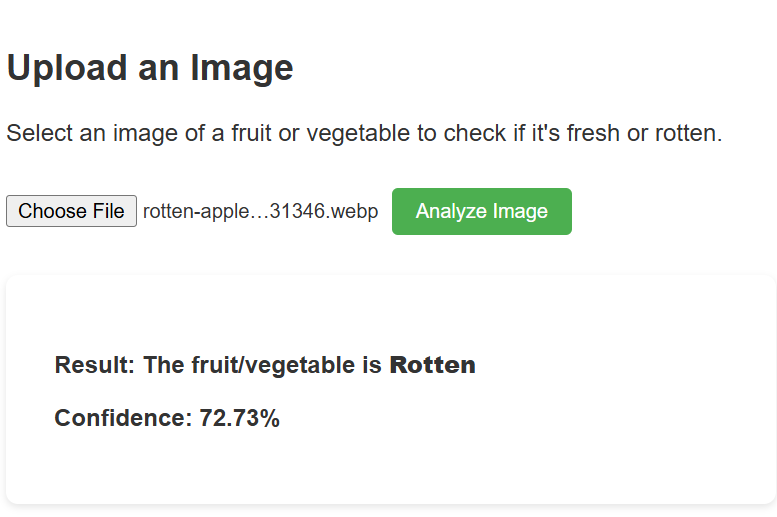
# 11. Screenshots











# 12. Known Issues

-Some fruit images with occlusions or unusual lighting may be misclassified.

- Model struggles with mixed-type batches (multiple fruits in one image).

-No offline mode currently available.

# 13. Future Enhancements

-Add support for more fruit/vegetable categories

-Improve model accuracy with more diverse dataset

-Add voice assistant and barcode scanning integration

-Implement mobile app version (React Native)