Artificial Intelligence And Machine Learning

Project Documentation

1. Introduction

• Project Title: Waste Warriors

Waste Warriors is an AI-powered CleanTech web application that classifies food or waste items as *Healthy* or *Rotten* using image analysis. It leverages transfer learning to support sustainable waste management practices.

• Team Members:

- **Gouthami Boddu[Member 1]** – Full Stack Developer (Frontend + Backend Integration)
- **Maniprabha Bhuvanasi[Member 2]** – AI/ML Developer (Model Training and Prediction)
- **Allam raju[Member 3]** – UI/UX Designer (Frontend Components and Layout)
- **Yalla Satya Venkata Sri Sai rishi [Member 4]** – Database Engineer (MongoDB Management and Integration)

2. Project Overview

• Purpose:

This project is aimed at modernizing the waste classification process using machine learning. It enables users to upload images of food or waste and receive instant classification results (Healthy or Rotten), helping individuals, businesses, and municipal systems make informed disposal decisions.

This AI-based solution addresses real-world challenges in:

- Recycling centers (automated waste sorting),
- Smart cities (smart bins),

- Industrial sectors (real-time classification of factory waste).

• Features:

- Upload and preview food or waste images.
- AI-based classification using transfer learning.
- Real-time prediction with accuracy confidence.
- MongoDB storage for predictions and feedback.
- Clean and responsive user interface.
- Eco-friendly suggestions based on predictions.

3. Architecture

• Frontend:

The frontend is developed using **React.js** with a component-based architecture. Each section (navbar, upload, result) is modular and reusable. Axios is used to send HTTP requests to the backend. Tailwind CSS ensures responsiveness and clean styling across devices. The UI updates dynamically based on the backend's response.

• Backend:

The backend is powered by **Flask (Python)** instead of Node.js for easier integration with TensorFlow. It handles:

- Receiving images from the frontend.
- Preprocessing images for the model.
- Running predictions using a pre-trained model (e.g., MobileNet).
- Returning the predicted label and confidence score.
- Connecting to MongoDB for storing results.

• Database:

MongoDB is used as a NoSQL database for flexible document storage. Collections include:

- predictions – Stores image metadata, prediction result, and timestamp.

4. Setup Instructions

• Prerequisites:

To run this project, you need the following installed:

- **Node.js** (for frontend)
- **Python 3.8+** (for backend)
- **MongoDB** (local or MongoDB Atlas)
- **npm** (Node package manager)
- **pip** (Python package manager)

• Installation:

1. Clone the project repository from GitHub.
2. Navigate to the frontend/ folder:
    - Install dependencies using npm install.
3. Navigate to the backend/ folder:
    - Create and activate a Python virtual environment.
    - Install dependencies using pip install -r requirements.txt.
4. Set environment variables (e.g., MONGO_URI, FLASK_APP).
5. Ensure MongoDB is running locally or use an Atlas connection string.

5. Folder Structure

• Client:

frontend/

src/

components/

Navbar.jsx

UploadForm.jsx

ResultDisplay.jsx

App.jsx

index.js

public/

index.html

package.json

- components/: Contains reusable React components.
- App.jsx: Main app structure and routes.
- index.js: Entry point for rendering the React app.
- package.json: Lists all frontend dependencies.

• **Server:**

backend/

app.py

model/

model.h5

utils/

image_preprocessor.py

model_loader.py

routes/

predict_route.py

database/

mongo_connection.py

uploads/

.env

requirements.txt

- app.py: Main Flask server script.
- model/: Contains the pre-trained model used for predictions.
- utils/: Helper scripts for image processing and model handling.
- routes/: API endpoints like /predict.
- database/: MongoDB connection configuration.
- uploads/: Temporary storage for user-uploaded images.

6. Running the Application

To run the full application locally:

• Frontend:

- Navigate to the frontend directory.
- Run npm install (once) to install dependencies.
- Run npm start to launch the development server.

Accessible at: http://localhost:3000

• Backend:

- Navigate to the backend directory.
- Activate your Python environment.
- Run python app.py or flask run to start the server.

7. API Documentation

• POST /predict

- **Description:** Accepts an uploaded image and returns prediction.
- **Method:** POST

- **Request Format:** multipart/form-data
- **Parameters:**
    - file: image file to be classified
- **Response Example:**


```
{
 "prediction": "Rotten",
 "confidence": 0.92
}
```


8. Authentication

Authentication is optional in the current implementation. If enabled:

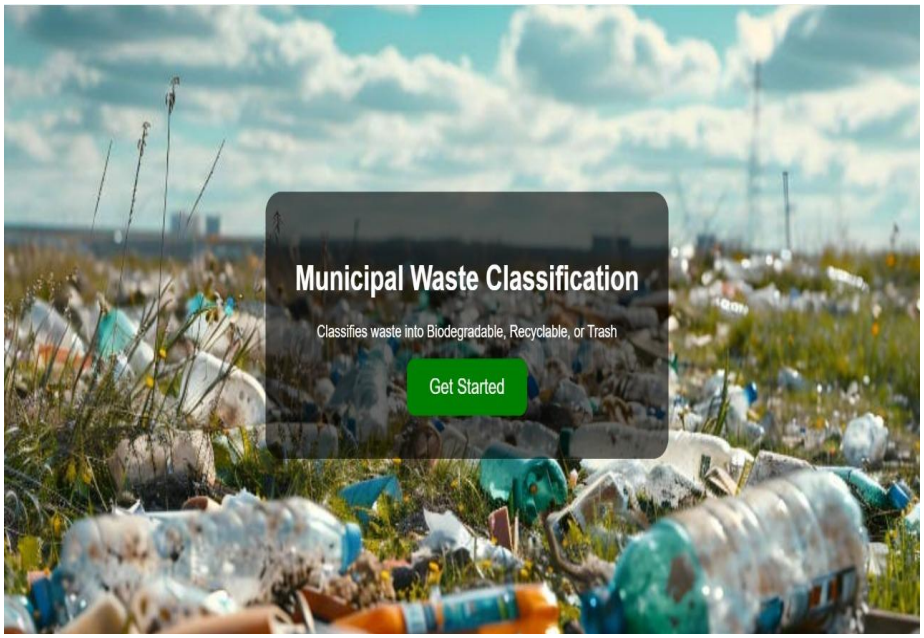
- **Method:** JSON Web Token (JWT)
- **Flow:**
    - User logs in with email and password.
    - Server generates a token and returns it.
    - Token is stored in localStorage on the frontend.
    - Protected API routes validate the token before granting access.
- **Security:** Token expires after a set time to prevent misuse.

Benefits of Adding Authentication:

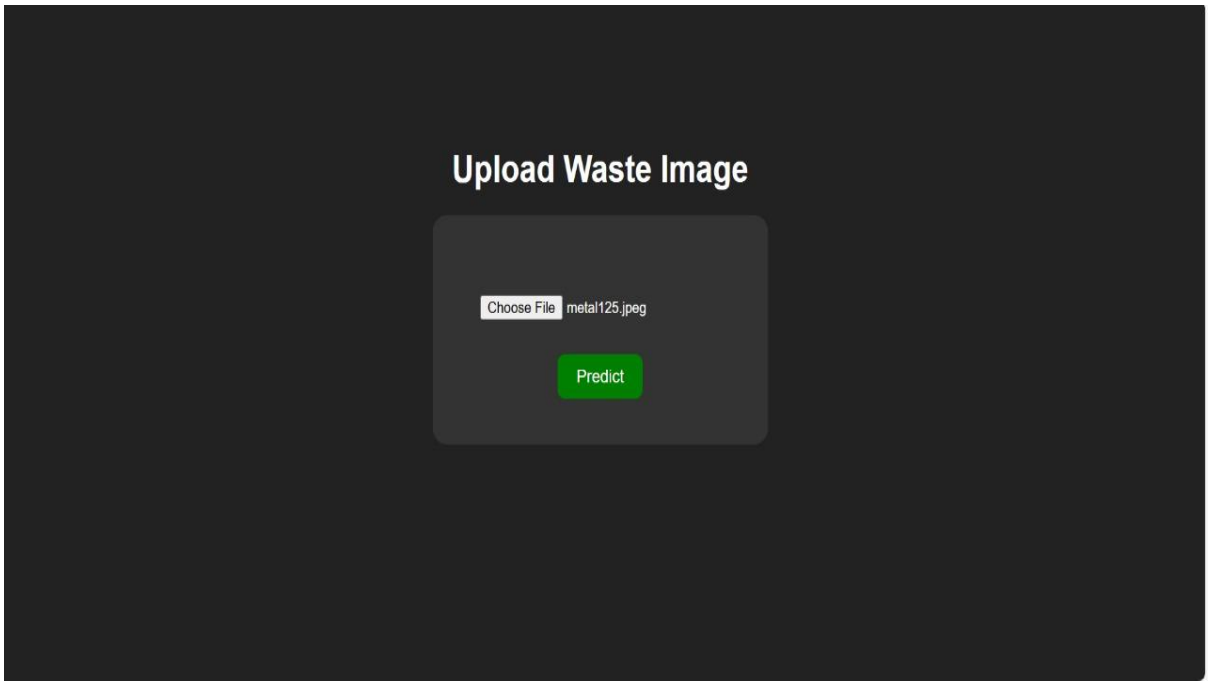
- Personalized prediction history
- Feedback tracking and learning per user
- Access control for sensitive data
- Better data security and accountability
- User-specific dashboard and insights

9.User Interface:

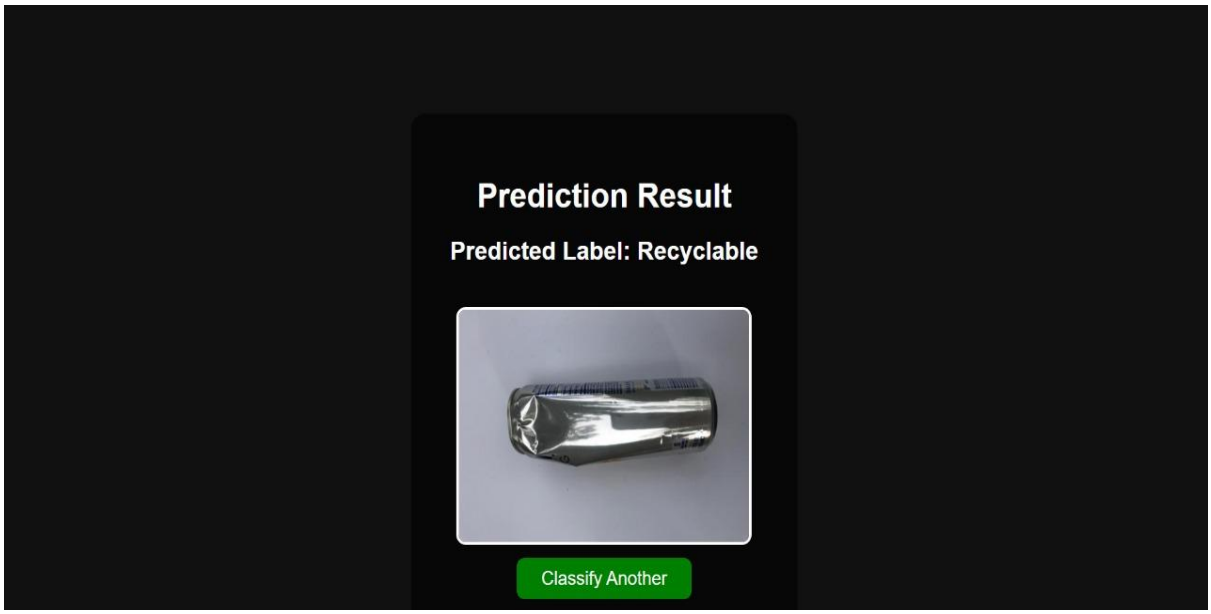
Website Interface – CleanTech



Homepage Interface:



Prediction Output:



10. Testing

• Testing Strategy:

To ensure the stability, accuracy, and reliability of the *HealthyVsRotten – CleanTech* web application, a combination of manual and automated testing strategies was used across both frontend and backend components.

Unit Testing:

- Individual Python functions such as image preprocessing, model loading, and prediction logic were tested in isolation.
- Unit tests verified that the image input is correctly resized, normalized, and passed to the model.

Integration Testing:

- Tested the integration between frontend and backend by simulating full prediction cycles:
    - Upload image → API call → Model prediction → Display result.
- Ensured MongoDB correctly stores prediction results after a successful prediction.

End-to-End Testing:

- Manually tested the entire workflow from image upload to result display across different browsers (Chrome, Edge, Firefox).
- Ensured consistent behavior and correct response even under slow network conditions.

Functional Testing:

- Verified each feature against its requirement:
  - Image upload
  - Prediction result display
  - Error messages for invalid formats or missing inputs
  - Loading indicators and UI responsiveness

• Tools Used:

Tool/Library	**Purpose**
Postman	Testing API endpoints with different inputs
Vscode	For writing the code
PyTest / unittest	Backend unit testing (Flask app)
Jest (if React)	Component-level testing for React frontend
MongoDB Compass	Verifying backend DB entries manually
Browser DevTools	Inspecting frontend behavior and performance

11.Demo link:

link to a demo to showcase the application:

: [“https://drive.google.com/file/d/1eiSA1r-gIwkmo6G2UbCBnbhyZhwsmCaB/view?usp=drive_link “]

12. Known Issues

Despite thorough testing, the *HealthyVsRotten – CleanTech* application has a few known limitations and areas for improvement:

- Limited Dataset Accuracy:
  The model's accuracy is dependent on the quality and diversity of the training dataset. In real-world scenarios, it may misclassify images due to poor lighting, complex backgrounds, or uncommon waste types.
- **Image Upload Size Limitations:**
  Currently, only images under a certain size threshold are accepted. Larger files may cause timeouts or crashes in some environments.
- **No Real-Time Camera Feed:**
  The application only works with manually uploaded images. Real-time camera integration (e.g., for smart bins or conveyor belts) is not implemented yet..
- **Limited Mobile Optimization:**
  Although responsive, certain UI elements may not render perfectly on very small screens or older devices.

13. Future Enhancements

Several enhancements can be made to improve the functionality, scalability, and impact of this project:

- **Real-Time Camera Integration:**
  Extend the application to work with live camera feeds for real-time waste classification in smart bins, factories, or recycling plants.
- **Advanced Classification Categories:**
  Expand the model to classify waste into multiple categories (e.g., Organic, Recyclable, Hazardous) instead of just Healthy/Rotten.

- **Authentication & User Dashboard:**
  Implement secure login, user roles (Admin, User), and dashboards for viewing personal prediction history and uploading training data.
- **Model Retraining via Feedback Loop:**
  Allow users to submit corrections to predictions, which can be used to retrain and improve the model over time.
- **Mobile App Version:**
  Create a companion mobile app to make the tool more accessible for household and field use.
- **Analytics and Reporting:**
  Provide analytics to administrators showing trends in predictions, types of waste most frequently misclassified, and improvement metrics.

---------- **THANK YOU** ----------