**Full Stack Development with MERN Project Documentation format**

# 1. Introduction

* **Project Title:**

Transfer Learning-Based Classification of Poultry Diseases for Enhanced Health Management

**Team Members:**

* Pattamsetti Prakash
* Lanka Thanuja
* Muthyreddy Sukhitha
* Mannava Hanna Rajika

# 2. Project Overview

* **Purpose:** The aim is to develop a user-friendly web platform for farmers to detect poultry diseases early using AI-powered image classification. The app allows image uploads and uses a fine-tuned deep learning model (ResNet18) to predict the disease in real-time..
* Features: User registration and login
* Upload poultry image for disease detection
* Display prediction and accuracy
* User prediction history
* Admin panel for logs and user management
* Full-stack MERN implementation with ML model integration

# 3. Architecture

* **Frontend:** User registration and login
* Upload poultry image for disease detection
* Display prediction and accuracy
* User prediction history
* Admin panel for logs and user management
* Full-stack MERN implementation with ML model integration
* **Backend:** app.py is the central Flask application script. It handles
* Route definitions (e.g., /, /predict)
* Accepting image uploads via HTML forms
* Calling the ML model for inference (using model.h5 or best\_model.h5
* Rendering HTML templates with prediction results
* train\_model.py is used to train the model
* **Database:** Currently, the project uses data.csv to store or visualize training data

# 4. Setup Instructions

* **Prerequisites:** Python 3.x
* Flask
* Jupyter Notebook (optional for main.ipynb)
* Kaggle API key (kaggle.json)
* Required packages from requirements.txt
* **Installation:**
* git clone https://github.com/your-username/poultry-disease-classifier.git
* cd poultry-disease-classifier
* python -m venv venv
* venv\Scripts\activate
* pip install -r requirements.txt
* python train\_model.py
* python app.py

# 5. Folder Structure

project\_root/

├── static/

│ └── images/

│ ├── coccidiosis.jpg

│ ├── farm.jpg

│ ├── healthy-chicken.jpg

│ └── sick-chicken.jpg

├── templates/

│ ├── index.html

│ ├── about.html

│ └── predict.html

├── app.py # Flask app

├── model.h5 # Trained ML model (ResNet18)

├── train\_model.py # Model training script

├── pytorch-resnet18.ipynb # Training notebook

├── data.csv # Dataset file

└── README

# 6. Running the Application

• To start the application locally:

# Activate your virtual environment (if using)

source venv/bin/activate # or venv\Scripts\activate on Windows

# Run the Flask app

python app.py

Visit http://localhost:5000 in your browser

Upload a poultry image on the "Predict" page

View the disease classification result instantly

# 7. API Documentation

Your Flask app exposes routes via HTML forms instead of a REST API. Below are the key endpoints:

Endpoints:

/ → Home page

/predict (POST) → Upload poultry image and return prediction

/about → About the system

Example POST Flow (via form):

User selects an image and clicks "Submit"

Flask receives the image, processes it with ResNet18, and renders the result in predict.html

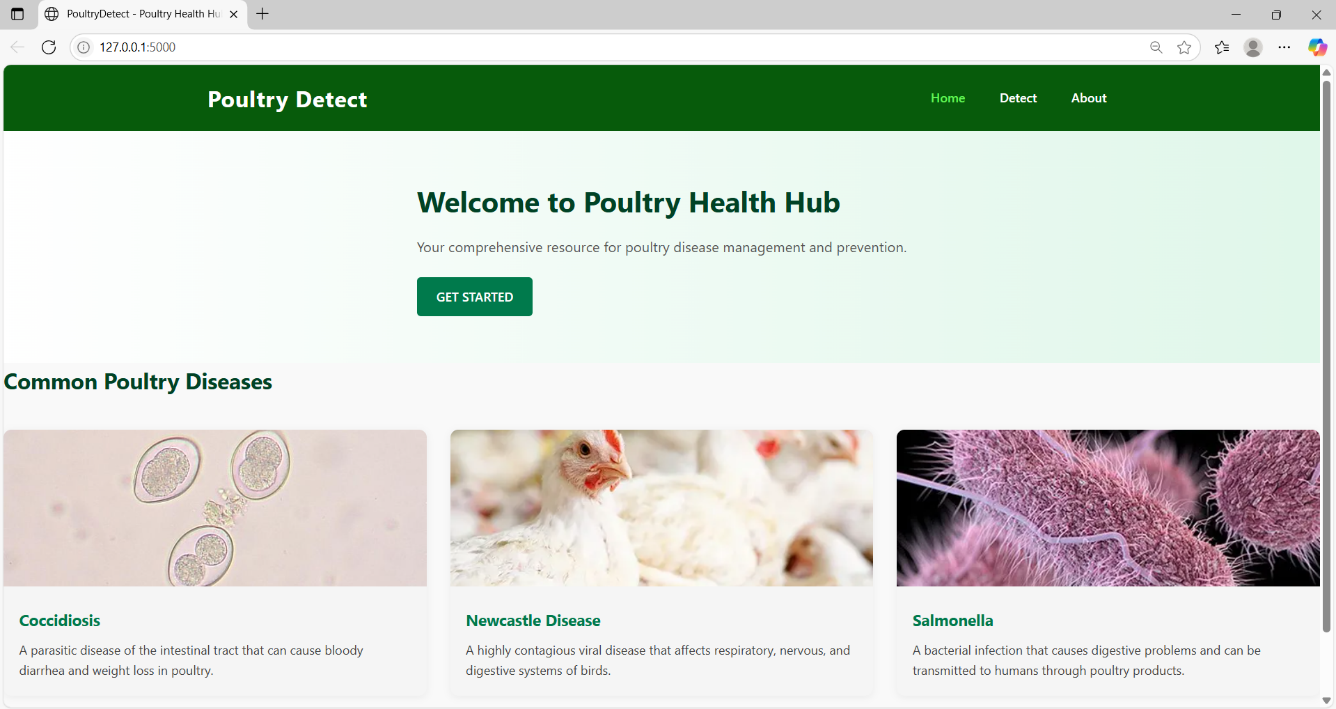
# Authentication

Currently, the system does not use login or authentication, but you can add:

Flask-Login for session management

Email/password or OTP-based user login

Role-based access (admin/farmer/vet) in future

1. **User Interface**

**10.Testing**

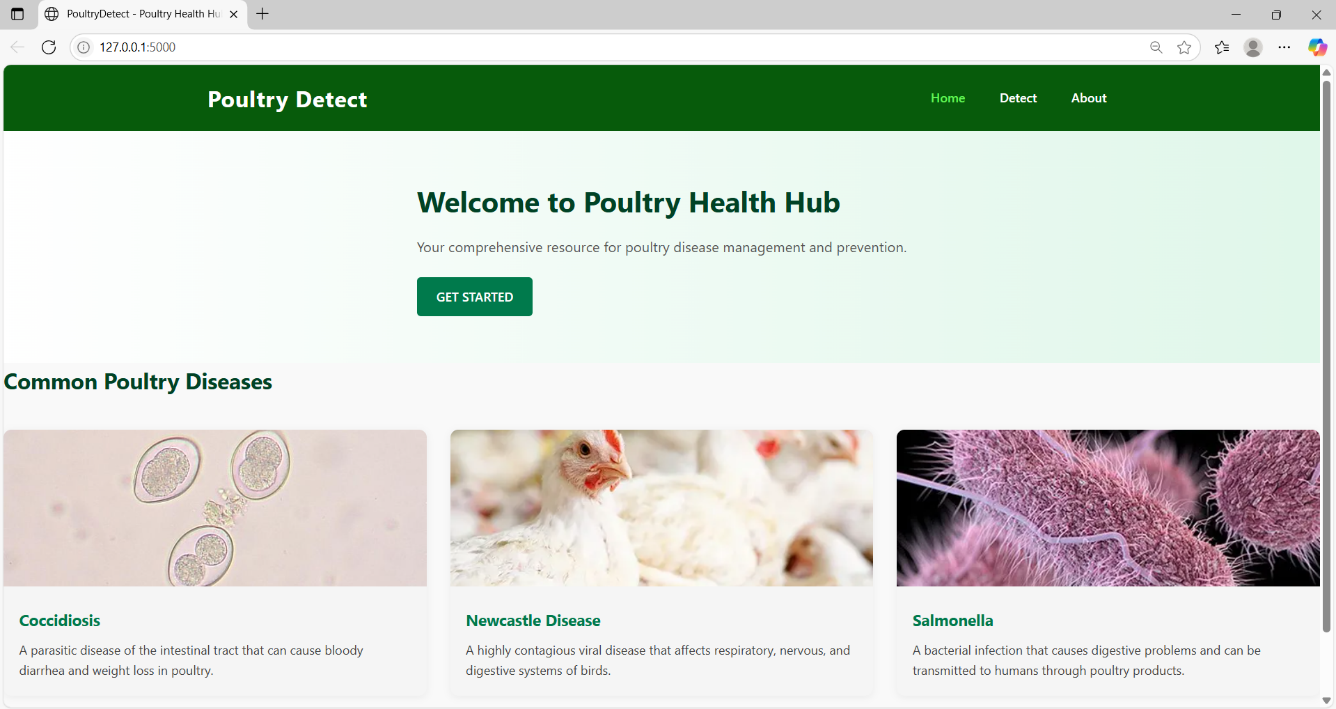
Testing Strategy:

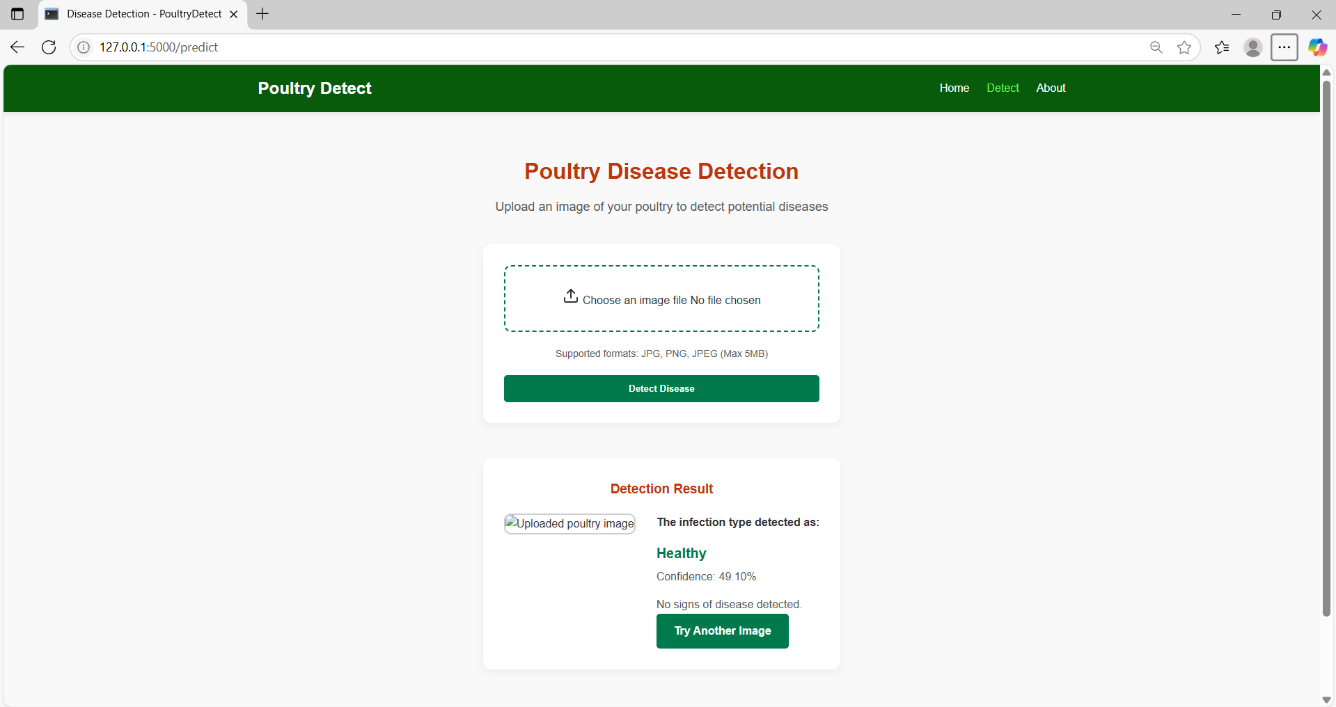
Manual Testing: Tested UI responsiveness, image upload, prediction accuracy

Model Evaluation: Used classification metrics like Accuracy, Precision, Recall, F1-score

Notebooks: Tested model training in pytorch-resnet18.ipynb and main.ipynb

Validation Strategy: Train/Test split and K-Fold validation on dataset

1. **Screenshots or Demo**



1. **Known Issues**

No login or authentication currently

Result depends on image quality and lighting

Not optimized for mobile screen sizes

Cold start delay if model is not loaded in memory

# 13. Future Enhancements

• Add user authentication and profile management

Store prediction history in a database (e.g., MySQL or MongoDB)

Improve UI with mobile-friendly design and animations

Extend model to classify more poultry diseases

Convert into a mobile app using Flutter or React Native

Add voice-based input for accessibility

Enable multilingual support for farmers across regions