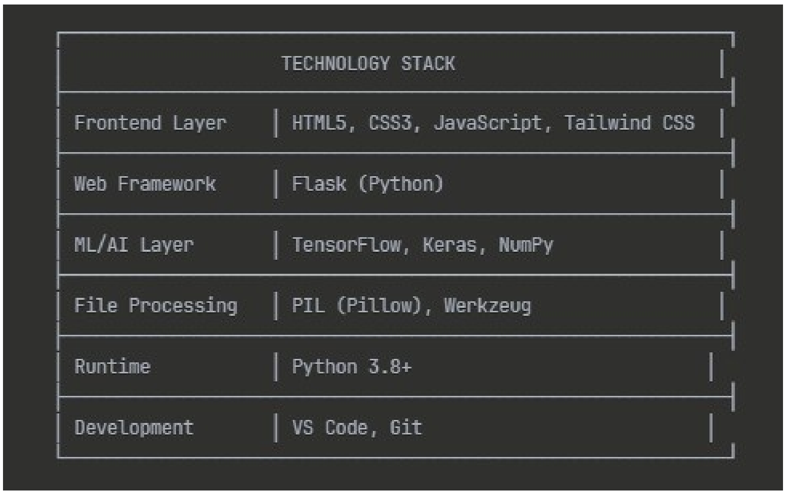
Technology Stack Documentation

Project: "Transfer Learning-Based Classification of Poultry Diseases for Enhanced Health Management"
Location: Ongole, Andhra Pradesh
Date: June 2025
Team ID: LTVIP2025TMID42969
Team Members: P. Srinivasa Kalyan, M. Karthik Reddy

1. Technology Stack Overview

1.1 Architecture Pattern

Pattern: Model-View-Controller (MVC) with AI/ML Integration **Deployment:** Single-tier web application with integrated ML model



TECHNOLOGY STACK	
Frontend Layer	HTML5, CSS3, JavaScript, Tailwind CSS
Web Framework	Flask (Python)
ML/AI Layer	TensorFlow, Keras, NumPy
File Processing	PIL (Pillow), Werkzeug
Runtime	Python 3.8+
Development	VS Code, Git

2. Backend Technologies 2.1 Core

Framework

Flask Web Framework

- **Version:** 2.3.0+
- **Purpose:** Web application framework and routing
- **Key Features:**
  - Lightweight and flexible ○ Jinja2 templating engine ○ Built-in development server
  - RESTful request dispatching

Implementation Details:

```
from flask import Flask, render_template, request


app = Flask(__name__)
```

```
@app.route('/') def
index():
   return render_template('index.html')


@app.route('/predict', methods=['POST']) def
upload():
   # File processing and prediction logic
   pass
```

Advantages:

- Minimal setup and configuration
- Excellent for prototyping and small applications
- Strong community support
- Easy integration with Python ML libraries

2.2 Machine Learning Stack

TensorFlow/Keras

- **Version:** TensorFlow 2.12.0, Keras 2.12.0
- **Purpose:** Deep learning model inference
- **Model Type:** Convolutional Neural Network (CNN)
- **Model File:** healthy_vs_rotten.h5 (pre-trained)

Key Capabilities:

```
from keras.models import load_model from
keras.preprocessing import image import
numpy as np


# Model loading model =
load_model("healthy_vs_rotten.h5")


# Image preprocessing img = image.load_img(img_path,
target_size=(224, 224)) arr = image.img_to_array(img) /
255.0
```

arr = np.expand_dims(arr, axis=0)

# Prediction pred =

model.predict(arr)[0]

NumPy

- **Version:** 1.24.0+
- **Purpose:** Numerical computations and array operations
- **Use Cases:**
    - Image array manipulation ○ Model prediction processing
    - Mathematical operations

2.3 File Processing

Werkzeug

- **Version:** 2.3.0+
- **Purpose:** WSGI utility library and file handling
- **Key Features:**
    - Secure filename sanitization
    - File upload handling ○ HTTP utilities

PIL (Pillow)

- **Version:** 9.5.0+
- **Purpose:** Image processing and manipulation
- **Features:**
    - Image format conversion
    - Resizing and cropping ○ Format validation

Implementation: from werkzeug.utils

import secure_filename from PIL import

Image

# Secure file handling

```
filename = secure_filename(file.filename)


# Image processing img =
Image.open(img_path) img
= img.resize((224, 224))
```

3. Frontend Technologies

3.1 Core Web Technologies

HTML5

- **Purpose:** Semantic markup and structure
- **Key Features:**

  ○ File input for image uploads ○
  Semantic elements for accessibility ○
  Canvas support for image display ○
  Form validation

CSS3

- **Purpose:** Styling and visual presentation
- **Advanced Features:**

  ○ Flexbox and Grid layouts ○ CSS
  animations and transitions ○
  Backdrop filters for glass effects
  ○ Responsive design media queries

JavaScript (Vanilla)

- **Purpose:** Client-side interactivity
- **Features:**

  ○ Form submission handling ○
  File validation ○ Dynamic
  content updates ○ Animation
  controls

3.2 CSS Framework

Tailwind CSS

- **Version:** 3.3.0 (CDN)

-
  Purpose: Utility-first CSS framework
- **CDN Integration:**

```
<script src="https://cdn.tailwindcss.com"></script>
```

Key Benefits:

- Rapid UI development
- Consistent design system
- Mobile-first responsive design
- Minimal custom CSS required

Usage Example:

```
<div class="bg-white/70 backdrop-blur-md rounded-2xl p-8 shadow-lg">
  <h1 class="text-4xl font-bold text-green-700 mb-4">PoultryDetect</h1>
</div>
```

3.3 UI Enhancement Libraries

Lottie Animations

- **Source:** @lottiefiles/lottie-player
- **Purpose:** Vector animations
- **CDN:** unpkg.com/@lottiefiles/lottie-player@latest
- **Implementation:**

```
<lottie-player src="animation.json"
 background="transparent"
 speed="1" loop autoplay>
</lottie-player>
```

Custom CSS Animations

- **Purpose:** Interactive elements and visual feedback
- **Examples:** ○ Walking hen animation ○ Hover effects
  - Loading indicators

4. Development Tools 4.1 Code Editor

Visual Studio Code

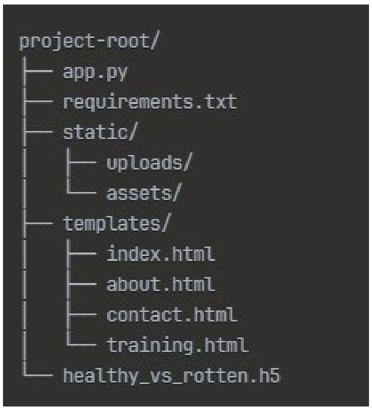
Extensions:

-
    - Python extension pack ○ HTML/CSS/JS language support ○ Git integration
    - Live Server for development

4.2 Version Control

Git

- **Purpose:** Source code management
- **Repository Structure:**



```
project-root/
├── app.py
├── requirements.txt
├── static/
│   ├── uploads/
│   └── assets/
├── templates/
│   ├── index.html
│   ├── about.html
│   ├── contact.html
│   └── training.html
└── healthy_vs_rotten.h5
```

4.3 Package Management pip (Python Package Installer) • **Requirements File:**

requirements.txt

- **Key Dependencies:** Flask==2.3.2 tensorflow==2.12.0

keras==2.12.0 numpy==1.24.3

Pillow==9.5.0

Werkzeug==2.3.6

5. External Services & CDNs

5.1 Content Delivery Networks

Tailwind CSS CDN

- **URL:** https://cdn.tailwindcss.com
- **Purpose:** CSS framework delivery

Fallback: Local Tailwind build if CDN fails

Lottie Files CDN

- **URL:** https://unpkg.com/@lottiefiles/lottie-player@latest
- **Purpose:** Animation player library

-
  - **Alternative:** Local animation files

5.2 External APIs

Google Scholar Integration

- **Purpose:** Research paper access
- **Implementation:** Direct URL construction research_url = f"https://scholar.google.com/scholar?q={disease_name}+in+Poultry"

6. System Requirements

6.1 Development Environment

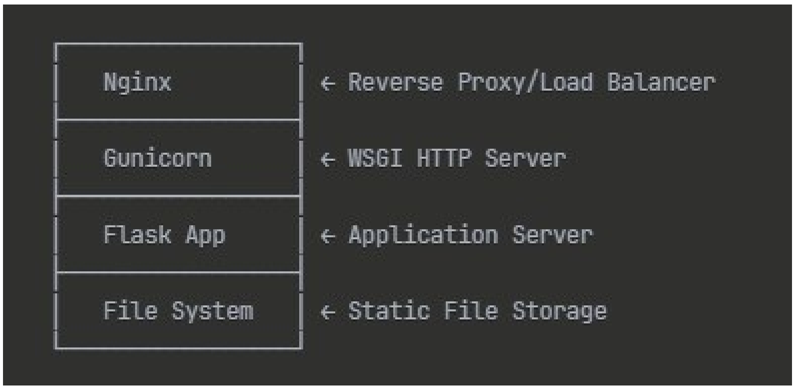
Hardware Requirements:

- RAM: 8GB minimum, 16GB recommended
- Storage: 10GB available space
- Processor: Multi-core CPU (Intel i5/AMD Ryzen 5 or better)
- Network: Stable internet connection for CDN resources

Software Requirements:

- Operating System: Windows 10/11, macOS 10.15+, or Linux Ubuntu 18.04+
- Python: 3.8 or higher
- Web Browser: Chrome, Firefox, Safari, or Edge (latest versions) **6.2 Production Environment Server Specifications:**
- RAM: 4GB minimum
- Storage: 20GB available space
- CPU: 2 cores minimum
- Network: Reliable internet connection

Deployment Stack:



```
┌──────────────────┐
│  Nginx           │ ← Reverse Proxy/Load Balancer
├──────────────────┤
│  Gunicorn        │ ← WSGI HTTP Server
├──────────────────┤
│  Flask App       │ ← Application Server
├──────────────────┤
│  File System     │ ← Static File Storage
└──────────────────┘
```

7. Performance Optimizations

7.1 Frontend Optimizations

CSS Optimization:

- Tailwind CSS purging for production
- Critical CSS inlining
- Image optimization and compression
- Lazy loading for non-critical assets

JavaScript Optimization:

- Minification for production
- Asynchronous loading
- Event delegation
- Debounced input handling **7.2 Backend**

Optimizations Flask Optimizations:

- Template caching
- Static file serving optimization
- Gzip compression
- Request routing optimization

ML Model Optimizations:

- Model preloading on application start
- Image preprocessing optimization
- Batch prediction capability
- Memory management for large images

7.3 Caching Strategy

Browser Caching:

- Static asset caching headers
- CDN resource caching
- Application cache for offline capability

Server-Side Caching:

- Template fragment caching
- Model prediction result caching
- Static file caching

8. Security Considerations

8.1 Input Validation

File Upload Security:

```
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
MAX_FILE_SIZE = 10 * 1024 * 1024  # 10MB


def allowed_file(filename):     return '.' in filename and \
filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

8.2 Security Headers

HTTP Security Headers:

- Content Security Policy (CSP)
- X-Frame-Options
- X-Content-Type-Options
- Secure file upload handling

8.3 Data Privacy

Privacy Measures:

- No persistent user data storage
- Automatic file cleanup
- Minimal logging
- No tracking or analytics

9. Testing Framework

9.1 Testing Strategy

Manual Testing:

- Cross-browser compatibility testing
- Responsive design testing
- File upload functionality testing
- ML model prediction accuracy testing

Automated Testing (Future):

- Unit tests for Flask routes
- Integration tests for ML pipeline
- Performance testing for file uploads
- Security testing for file handling

9.2 Quality Assurance

Code Quality:

- Python PEP 8 style compliance
- HTML/CSS validation
- JavaScript linting
- Documentation coverage **Performance**

Monitoring:

- Response time measurement
- Memory usage tracking
- File storage monitoring
- Error rate tracking