

# Disease Prediction System

## Pneumonia Detector

Comprehensive Project Documentation

Version 1.0

Generated on: November 10, 2025

## Abstract

This document provides comprehensive documentation for the Disease Prediction System, specifically designed for pneumonia detection using chest X-ray images. The system is built using Django web framework and employs a deep learning model (TensorFlow/Keras) to classify chest X-ray images as either "Normal" or "Pneumonia". The application features user authentication, image upload functionality, real-time prediction capabilities, and a comprehensive dashboard for tracking prediction history and statistics. This documentation covers all modules, their functionality, architecture, implementation details, deployment guidelines, security best practices, and troubleshooting. **Target Audience:** Developers, system administrators, healthcare researchers, and technical stakeholders who need to understand, deploy, or maintain the system.

# **Table of Contents**

1. Introduction
2. System Architecture
3. Project Structure
4. Core Module (disease\_prediction)
5. Pneumonia Predictor Module
6. Authentication Module
7. Database Models
8. URL Routing and Views
9. Templates and User Interface
10. Installation and Setup
11. Usage Instructions
12. API Endpoints
13. Security Best Practices
14. Deployment Guide
15. Testing
16. Troubleshooting and FAQ
17. Dependencies
18. Future Enhancements
19. Conclusion

Appendix A: Glossary

Appendix B: References

# 1. Introduction

The Disease Prediction System is a web-based application that uses artificial intelligence to detect pneumonia from chest X-ray images. The system is designed to assist healthcare professionals and researchers in analyzing medical images efficiently. **Purpose:** This system provides an automated tool for preliminary pneumonia detection in chest X-ray images, helping healthcare professionals make faster and more accurate diagnoses. However, it is important to note that this tool is intended for educational and research purposes and should not replace professional medical judgment. **Key Features:**

- User authentication and authorization
- Image upload and preprocessing
- Real-time pneumonia prediction using deep learning
- Prediction history and statistics dashboard
- User-friendly web interface
- Secure file handling and storage
- User-specific prediction tracking

## 1.1 Technology Stack

**Backend Framework:** Django 4.2.25

**Machine Learning:** TensorFlow 2.15.0, Keras 2.15.0

**Database:** SQLite3 (development), PostgreSQL/MySQL (production recommended)

**Image Processing:** Pillow 11.3.0, NumPy 1.26.4

**Frontend:** HTML5, CSS3, JavaScript

**Server:** Django Development Server (development), Gunicorn/uWSGI (production)

**Reverse Proxy:** Nginx (production recommended)

## 1.2 System Requirements

### Minimum Requirements:

- Python 3.8 or higher
- 4GB RAM
- 2GB free disk space
- Internet connection (for initial setup)

### Recommended Requirements:

- Python 3.10 or higher
- 8GB RAM or more
- 5GB free disk space
- GPU support (optional, for faster predictions)
- Ubuntu 20.04+ / Windows 10+ / macOS 10.15+

## 2. System Architecture

The Disease Prediction System follows a three-tier architecture pattern:

1. **Presentation Layer:** HTML templates with CSS and JavaScript for user interface
2. **Application Layer:** Django views and business logic for processing requests
3. **Data Layer:** SQLite database for storing predictions and user data

### 2.1 System Flow

#### User Request Flow:

1. User accesses the web application through a browser
2. User authenticates (login/signup) if not already logged in
3. User uploads a chest X-ray image
4. Image is validated and saved to media directory
5. Image is preprocessed (resize, normalize)
6. Preprocessed image is fed to the TensorFlow model
7. Model returns prediction (Normal/Pneumonia) with confidence score
8. Prediction result is saved to database
9. Result is displayed to the user
10. User can view prediction history in the dashboard

### 2.2 Component Diagram

#### Main Components:

- **Django Web Framework:** Handles HTTP requests, routing, and templating
- **TensorFlow/Keras Model:** Pre-trained deep learning model for pneumonia detection
- **Database (SQLite):** Stores user accounts and prediction history
- **Media Storage:** File system storage for uploaded images
- **Authentication System:** Django's built-in authentication for user management

## 3. Project Structure

The project follows Django's standard project structure with the following key directories:

```
disease_prediction/
└── disease_prediction/
    ├── __init__.py                                # Core project configuration
    ├── settings.py                               # Django settings
    ├── urls.py                                   # Main URL configuration
    ├── wsgi.py                                    # WSGI configuration
    └── asgi.py                                    # ASGI configuration
        └── __init__.py                            # Main application
            ├── __init__.py                          # Database models
            ├── models.py                           # View functions
            ├── views.py                            # Form definitions
            ├── forms.py                            # Admin configuration
            ├── admin.py                            # App configuration
            ├── apps.py                             # Unit tests
            ├── tests.py                            # HTML templates
            └── templates/
                ├── base.html
                ├── upload.html
                ├── dashboard.html
                └── includes/
                    └── navbar.html
                └── registration/
                    ├── login.html
                    └── signup.html
            └── migrations/                         # Database migrations
                └── 0001_initial.py
            └── models/
                └── pneumonia_model.h5             # ML model storage
        └── media/                                 # Uploaded media files
        └── static/                                # Static files (CSS, JS, images)
        └── db.sqlite3                            # SQLite database
    └── manage.py                                # Django management script
    └── requirements.txt                         # Python dependencies
    └── README.md                                # Project README
```

### 3.1 Directory Descriptions

**disease\_prediction/**: Core Django project configuration files

**pneumonia\_predictor/**: Main application containing business logic

**media/**: User-uploaded images (not version controlled)

**static/**: Static files like CSS and JavaScript

**migrations/**: Database migration files

**models/**: Pre-trained machine learning model files

## 4. Core Module (disease\_prediction)

The core module contains the main Django project configuration.

### 4.1 settings.py

The settings.py file contains all Django project configuration including:

- **INSTALLED\_APPS**: Lists all installed Django applications including 'pneumonia\_predictor'
- **MIDDLEWARE**: Configures security, session, authentication, and CSRF middleware
- **DATABASES**: SQLite3 database configuration
- **STATIC\_URL** and **MEDIA\_URL**: Configuration for static and media files
- **AUTH\_PASSWORD\_VALIDATORS**: Password validation rules for user authentication
- **LOGIN\_URL**, **LOGIN\_REDIRECT\_URL**, **LOGOUT\_REDIRECT\_URL**: Authentication URL configuration
- **SECRET\_KEY**: Secret key for cryptographic signing (should be kept secure in production)
- **DEBUG**: Debug mode (set to False in production)
- **ALLOWED\_HOSTS**: List of allowed hostnames

#### 4.1.1 Key Settings Configuration

```
# settings.py - Key Configuration

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'pneumonia_predictor', # Our main application
]

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'

LOGIN_URL = '/accounts/login/'
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/accounts/login/'
```

### 4.2 urls.py

The main URL configuration file routes URLs to their respective views:

- **'/admin/'**: Django admin interface
- **'/'**: Main upload and prediction page (upload\_and\_predict view)
- **'/dashboard/'**: Dashboard view showing prediction statistics
- **'/accounts/signup/'**: User registration page
- **'/accounts/'**: Includes Django's built-in authentication URLs (login, logout, password reset)

- **Media files:** Serves uploaded media files during development

#### 4.2.1 URL Configuration Code

```
# urls.py - URL Configuration

from django.contrib import admin
from django.urls import path, include
from pneumonia_predictor import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.upload_and_predict, name='upload_and_predict'),
    path('dashboard/', views.dashboard, name='dashboard'),
    path('accounts/signup/', views.signup, name='signup'),
    path('accounts/', include('django.contrib.auth.urls')),
]

# Serve media files in development
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
                          document_root=settings.MEDIA_ROOT)
```

#### 4.3 wsgi.py and asgi.py

**wsgi.py:** Web Server Gateway Interface configuration for deployment with WSGI servers (e.g., Gunicorn, uWSGI)

**asgi.py:** Asynchronous Server Gateway Interface configuration for deployment with ASGI servers (e.g., Daphne, Uvicorn) supporting WebSockets and async features

## 5. Pneumonia Predictor Module

The pneumonia\_predictor module is the main application containing the core functionality.

### 5.1 views.py

The views.py file contains all view functions that handle HTTP requests and business logic.

#### 5.1.1 predict\_pneumonia()

**Function:** predict\_pneumonia(img\_path)

**Purpose:** Predicts whether a chest X-ray shows pneumonia or is normal

**Parameters:** img\_path (str) - Path to the image file

**Returns:** Dictionary with 'label' ('Normal' or 'Pneumonia') and 'probability' (float)

**Process:**

1. Loads and preprocesses the image (resize to 224x224, normalize to [0,1])
2. Uses the loaded TensorFlow/Keras model to make predictions
3. Returns prediction label and confidence score

**Model:** Loaded once at application startup from pneumonia\_model.h5

#### 5.1.1.1 Code Implementation

```
# views.py - Prediction Function

import os
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from django.conf import settings

# Load model once at startup
MODEL_PATH = os.path.join(settings.BASE_DIR,
                           'pneumonia_predictor',
                           'models',
                           'pneumonia_model.h5')
model = load_model(MODEL_PATH)

def predict_pneumonia(img_path):
    """
    Predicts pneumonia from chest X-ray image.

    Args:
        img_path (str): Path to the image file

    Returns:
        dict: Dictionary with 'label' and 'probability'
    """
    # Load and preprocess image
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    # Make prediction
```

```

prediction = model.predict(img_array)
prob = float(prediction[0][0])

# Determine label
label = "Normal" if prob > 0.5 else "Pneumonia"

return {"label": label, "probability": prob}

```

### 5.1.2 upload\_and\_predict()

**Function:** upload\_and\_predict(request) **Decorator:** @login\_required(login\_url='login')

**Purpose:** Handles image upload and prediction requests

**Process:**

1. Validates uploaded image using UploadImageForm
2. Saves uploaded image to media directory
3. Calls predict\_pneumonia() to get prediction
4. Saves prediction results to database (Prediction model)
5. Renders upload.html template with results

**Template:** upload.html

#### 5.1.2.1 Code Implementation

```

# views.py - Upload and Predict View

from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from .forms import UploadImageForm
from .models import Prediction

@login_required(login_url='login')
def upload_and_predict(request):
    """
    Handles image upload and prediction.
    Requires user authentication.
    """
    result = None
    image_url = None

    if request.method == "POST":
        form = UploadImageForm(request.POST, request.FILES)
        if form.is_valid():
            img = form.cleaned_data['image']

            # Save uploaded image
            os.makedirs(settings.MEDIA_ROOT, exist_ok=True)
            img_path = os.path.join(settings.MEDIA_ROOT, img.name)
            with open(img_path, 'wb+') as f:
                for chunk in img.chunks():
                    f.write(chunk)

            # Make prediction
            result = predict_pneumonia(img_path)
            image_url = settings.MEDIA_URL + img.name

            # Save prediction to database
            try:
                Prediction.objects.create(
                    user=request.user,
                    image_name=img.name,
                    label=result['label'],

```

```

        probability=result[ 'probability' ]
    )
except Exception as e:
    print("Prediction save failed:", e)
else:
    form = UploadImageForm()

return render(request, 'upload.html', {
    'form': form,
    'result': result,
    'image_url': image_url
})

```

### 5.1.3 dashboard()

**Function:** dashboard(request) **Decorator:** @login\_required(login\_url='login')

**Purpose:** Displays prediction statistics and history

**Data Provided:**

- Total number of predictions
- Number of pneumonia cases detected
- Number of normal cases
- Recent predictions for the current user (last 20)

**Template:** dashboard.html

#### 5.1.3.1 Code Implementation

```

# views.py - Dashboard View

@login_required(login_url='login')
def dashboard(request):
    """
    Displays prediction statistics and history.
    Requires user authentication.
    """
    # Get statistics
    total = Prediction.objects.count()
    pneumonia = Prediction.objects.filter(label='Pneumonia').count()
    normal = Prediction.objects.filter(label='Normal').count()

    # Get recent predictions for current user
    recent = Prediction.objects.filter(user=request.user)[:20]

    return render(request, 'dashboard.html', {
        'total': total,
        'pneumonia': pneumonia,
        'normal': normal,
        'recent': recent,
    })

```

### 5.1.4 signup()

**Function:** signup(request)

**Purpose:** Handles user registration

**Process:**

1. Uses Django's UserCreationForm for user registration

2. Validates form data
  3. Creates new user account
  4. Logs in the user automatically after registration
  5. Redirects to upload\_and\_predict page
- Template:** registration/signup.html

#### 5.1.4.1 Code Implementation

```
# views.py - Signup View

from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import login as auth_login

def signup(request):
    """
    Handles user registration.
    Automatically logs in user after successful registration.
    """
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            auth_login(request, user)
            return redirect('upload_and_predict')
    else:
        form = UserCreationForm()

    return render(request, 'registration/signup.html', {'form': form})
```

## 5.2 models.py

The models.py file defines the database models used in the application.

#### 5.2.1 Prediction Model

**Model:** Prediction

**Fields:**

- user (ForeignKey): Links prediction to a user account
- image\_name (CharField): Name of the uploaded image file
- label (CharField): Prediction result ('Normal' or 'Pneumonia')
- probability (FloatField): Confidence score from the model
- created\_at (DateTimeField): Timestamp of when prediction was made (auto-generated)

**Meta Options:**

- ordering: ['-created\_at'] - Orders predictions by most recent first

**Methods:**

- image\_url(): Returns the URL path to the uploaded image

### 5.2.1.1 Code Implementation

```
# models.py - Prediction Model

from django.db import models
from django.conf import settings

class Prediction(models.Model):
    """
    Stores prediction results for each user.
    """
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE
    )
    image_name = models.CharField(max_length=255)
    label = models.CharField(max_length=32)
    probability = models.FloatField()
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']

    def image_url(self):
        """
        Returns the URL path to the uploaded image.
        """
        return f'{settings.MEDIA_URL}{self.image_name}'

    def __str__(self):
        return f'{self.user.username} - {self.label} - {self.created_at}'
```

## 5.3 forms.py

The forms.py file defines form classes for user input validation.

### 5.3.1 UploadImageForm

**Form:** UploadImageForm

**Fields:**

- image (ImageField): Field for uploading chest X-ray images

**Validation:** Django automatically validates that uploaded file is a valid image format

**Usage:** Used in upload\_and\_predict view for image upload functionality

### 5.3.1.1 Code Implementation

```
# forms.py - Upload Image Form

from django import forms

class UploadImageForm(forms.Form):
    """
    Form for uploading chest X-ray images.
    """
```

## 5.4 admin.py

The admin.py file is used to register models with Django's admin interface. Currently, no models are registered, but the Prediction model can be registered for administrative management of prediction records.

# 13. Security Best Practices

Security is crucial for any web application, especially one handling medical data. This section outlines security best practices for the Disease Prediction System.

## 13.1 Production Settings

### Critical Security Settings for Production:

- Set DEBUG = False in settings.py
- Use a strong, unique SECRET\_KEY (generate new one for production)
- Configure ALLOWED\_HOSTS with your domain name
- Use HTTPS for all communications
- Store sensitive data in environment variables
- Use a production-grade database (PostgreSQL recommended)
- Enable CSRF protection (already enabled by default)
- Configure proper CORS settings if needed

## 13.2 Environment Variables

### Recommended Environment Variables:

- SECRET\_KEY: Django secret key
- DEBUG: Debug mode (False for production)
- ALLOWED\_HOSTS: Comma-separated list of allowed hosts
- DATABASE\_URL: Database connection string
- EMAIL\_HOST: SMTP server for email
- EMAIL\_PORT: SMTP port
- EMAIL\_HOST\_USER: SMTP username
- EMAIL\_HOST\_PASSWORD: SMTP password

## 13.3 File Upload Security

### File Upload Security Measures:

- Validate file types (only allow image formats)
- Limit file size (set MAX\_UPLOAD\_SIZE in settings)
- Scan uploaded files for malware (optional)
- Store uploaded files outside web root
- Use unique filenames to prevent overwrites
- Implement file cleanup for old uploads

## 13.4 Authentication Security

### **Authentication Security Best Practices:**

- Use strong password validation (already configured)
- Implement rate limiting for login attempts
- Use session timeout for inactive users
- Enable two-factor authentication (optional enhancement)
- Hash passwords securely (Django does this by default)
- Use HTTPS for authentication pages

# 14. Deployment Guide

This section provides guidance on deploying the Disease Prediction System to a production environment.

## 14.1 Pre-Deployment Checklist

### **Before Deployment:**

- Set DEBUG = False
- Generate new SECRET\_KEY
- Configure ALLOWED\_HOSTS
- Set up production database (PostgreSQL recommended)
- Configure static files collection
- Set up media files storage
- Configure email settings
- Set up logging
- Test all functionality
- Review security settings

## 14.2 Deployment with Gunicorn

### **Steps for Gunicorn Deployment:**

1. Install Gunicorn: pip install gunicorn
2. Collect static files: python manage.py collectstatic
3. Run migrations: python manage.py migrate
4. Start Gunicorn: gunicorn disease\_prediction.wsgi:application
5. Configure reverse proxy (Nginx) to forward requests to Gunicorn
6. Set up process manager (systemd) for auto-restart

## 14.3 Nginx Configuration

### **Basic Nginx Configuration:**

- Configure upstream to Gunicorn socket
- Set up server block for your domain
- Configure SSL/TLS certificates
- Set up static files serving
- Configure media files serving
- Set proper security headers

# 15. Testing

This section covers testing strategies and test cases for the Disease Prediction System.

## 15.1 Unit Tests

### **Recommended Unit Tests:**

- Test prediction function with sample images
- Test model loading and initialization
- Test image preprocessing
- Test database model creation
- Test form validation
- Test view functions

## 15.2 Integration Tests

### **Recommended Integration Tests:**

- Test user registration flow
- Test user login flow
- Test image upload and prediction flow
- Test dashboard data retrieval
- Test authentication required views

## 15.3 Running Tests

### **To run tests:**

```
python manage.py test
```

### **To run specific test:**

```
python manage.py test pneumonia_predictor.tests.TestPrediction
```

### **To run with coverage:**

```
coverage run --source='.' manage.py test
```

```
coverage report
```

# 16. Troubleshooting and FAQ

This section addresses common issues and frequently asked questions about the Disease Prediction System.

## 16.1 Common Issues

### 16.1.1 Model Not Loading

**Problem:** Model file not found or cannot be loaded

**Solution:**

- Verify pneumonia\_model.h5 exists in pneumonia\_predictor/models/ directory
- Check file permissions
- Verify TensorFlow and Keras are installed correctly
- Check model file integrity

### 16.1.2 Image Upload Fails

**Problem:** Image upload fails or returns error

**Solution:**

- Check MEDIA\_ROOT and MEDIA\_URL settings
- Verify media directory exists and is writable
- Check file size limits
- Verify image format is supported (JPEG, PNG, etc.)
- Check file permissions

### 16.1.3 Database Errors

**Problem:** Database errors or migrations fail

**Solution:**

- Run migrations: python manage.py migrate
- Check database file permissions
- Verify database settings in settings.py
- Check for migration conflicts
- Create superuser if needed: python manage.py createsuperuser

### 16.1.4 Authentication Issues

**Problem:** User cannot login or session expires

**Solution:**

- Verify LOGIN\_URL and LOGIN\_REDIRECT\_URL settings
- Check session configuration
- Verify user account is active
- Check password validation rules
- Clear browser cookies and cache

## 16.2 Frequently Asked Questions

### 16.2.1 What image formats are supported?

**Answer:** The system supports common image formats including JPEG, PNG, GIF, and BMP. The image is automatically converted to the required format (224x224 pixels) for model prediction.

### 16.2.2 How accurate is the prediction?

**Answer:** The accuracy depends on the trained model. The system displays a confidence score (probability) for each prediction. For medical use, always consult with healthcare professionals. This tool is for educational and research purposes only.

### 16.2.3 Can I use my own model?

**Answer:** Yes, you can replace pneumonia\_model.h5 with your own trained model. Ensure the model expects the same input format (224x224 RGB images) and outputs a single probability value.

### 16.2.4 How do I backup the database?

**Answer:** For SQLite, simply copy the db.sqlite3 file. For PostgreSQL or MySQL, use the respective database backup tools (pg\_dump, mysqldump).

# 17. Dependencies

The following are the main Python packages required for this project. A complete list is available in requirements.txt. Install all dependencies using: pip install -r requirements.txt

## 17.1 Core Dependencies

**Django 4.2.25:** Web framework for building the application

**TensorFlow 2.15.0:** Machine learning framework for loading and running the pneumonia prediction model

**Keras 2.15.0:** High-level neural networks API (included with TensorFlow)

**NumPy 1.26.4:** Numerical computing library for array operations

**Pillow 11.3.0:** Python Imaging Library for image processing and manipulation

**h5py 3.14.0:** Python interface to the HDF5 binary data format (required for loading .h5 model files)

## 17.2 Installation Notes

### System Requirements:

- Python 3.8 or higher
- TensorFlow supports both CPU and GPU. For GPU support, install TensorFlow-GPU and CUDA toolkit
- Minimum 4GB RAM recommended (8GB+ for better performance)
- Disk space: At least 2GB for dependencies and model files

### Installation Tips:

- Use a virtual environment to avoid conflicts with other projects
- On Windows, you may need to install Visual C++ Redistributable for TensorFlow
- For production, consider using a lighter deployment option or containerization (Docker)
- Regularly update dependencies for security patches: pip install --upgrade -r requirements.txt

## 18. Future Enhancements

This section outlines potential enhancements and improvements for future versions of the system.

### 18.1 Planned Features

#### **Short-term Enhancements:**

- REST API for programmatic access
- Batch image processing
- Enhanced dashboard with charts and graphs
- Export prediction history to CSV/PDF
- Email notifications for predictions

#### **Long-term Enhancements:**

- Support for multiple disease detection
- Model retraining capabilities
- Advanced visualization and analytics
- Integration with healthcare systems (HL7, DICOM)
- Mobile application support (iOS/Android)
- Real-time prediction via WebSocket
- Multi-language support
- Advanced user roles and permissions

### 18.2 Performance Optimizations

#### **Performance Improvements:**

- Implement model caching and lazy loading
- Add image compression and optimization
- Implement database query optimization
- Add Redis caching for predictions
- Implement asynchronous task processing (Celery)
- Add CDN for static and media files
- Implement database connection pooling
- Add gzip compression for responses
- Optimize TensorFlow model inference
- Implement batch processing for multiple images

### 18.3 Scalability Improvements

**Scalability Enhancements:**

- Horizontal scaling with load balancers
- Microservices architecture
- Containerization with Docker
- Kubernetes orchestration
- Database replication and sharding
- Message queue for async processing
- Auto-scaling based on load
- Geographic distribution

## 19. Conclusion

This documentation provides a comprehensive overview of the Disease Prediction System (Pneumonia Detector). The system leverages deep learning and web technologies to provide an accessible platform for chest X-ray image analysis.

### **Key Takeaways:**

- The system uses a pre-trained TensorFlow/Keras model for pneumonia detection
- Django framework provides a robust and secure web interface
- User authentication ensures data privacy and user-specific prediction history
- The dashboard provides valuable insights into prediction statistics
- The system is designed for extensibility and future enhancements

### **System Strengths:**

- Easy to deploy and configure
- User-friendly interface
- Secure authentication system
- Comprehensive prediction tracking
- Extensible architecture

### **Important Disclaimer:**

This system is intended for educational and research purposes only. It should not be used as a substitute for professional medical advice, diagnosis, or treatment. Always consult with qualified healthcare professionals for medical decisions. The predictions generated by this system are not intended for clinical use and should be verified by licensed medical professionals.

### **Support and Contributions:**

For issues, questions, or contributions, please refer to the project repository or contact the development team. We welcome feedback and contributions to improve the system.

## Appendix A: Glossary

### **Terminology:**

**API:** Application Programming Interface - a set of protocols for building software applications

**CNN:** Convolutional Neural Network - a type of deep learning model used for image recognition

**Django:** A high-level Python web framework

**Keras:** A high-level neural networks API

**ML Model:** Machine Learning Model - a mathematical representation of patterns in data

**TensorFlow:** An open-source machine learning framework

**WSGI:** Web Server Gateway Interface - a specification for web servers and applications

**ASGI:** Asynchronous Server Gateway Interface - an extension of WSGI for async applications

**ORM:** Object-Relational Mapping - a technique for accessing databases

**REST API:** Representational State Transfer API - a web service architecture

## Appendix B: References

### **Documentation:**

- Django Documentation: <https://docs.djangoproject.com/>
- TensorFlow Documentation: [https://www.tensorflow.org/api\\_docs](https://www.tensorflow.org/api_docs)
- Keras Documentation: <https://keras.io/api/>
- Python Documentation: <https://docs.python.org/>

### **Libraries:**

- ReportLab: <https://www.reportlab.com/docs/reportlab-userguide.pdf>
- Pillow: <https://pillow.readthedocs.io/>
- NumPy: <https://numpy.org/doc/>

### **Best Practices:**

- Django Security: <https://docs.djangoproject.com/en/stable/topics/security/>
- PEP 8 Style Guide: <https://www.python.org/dev/peps/pep-0008/>
- Web Security: <https://owasp.org/www-project-top-ten/>