Plant Disease Detection Software

UCS503 Software Engineering Project Report

Submitted by:
Jai Dalmotra - 102103716
Sheetal Ahuja - 102103700
(3CO25)

Submitted to: Arwinder Dhillon

Computer Science and Engineering Department
TIET, Patiala

Project Selection Phase

Name	Roll Number	Project Experience	Programming languages used
Jai Dalmotra	102103716	Weed detection Model, Portfolio Website	HTML,CSS, JS, Python
Sheetal	102103700	Roll the dice app, frameworks with UI/UX(Figma)	Flutter, Dart

Programming Language / Environment Experience

1. HTML, CSS, JAVASCRIPT

- Proficient in developing interactive web interfaces using HTML, CSS, and JavaScript.

2. SQL

- Experienced in designing and managing databases using SQL for a previous project.

3. PYTHON

- Proficient knowledge of Python and its potential for diverse application development.Experienced in machine learning and deep learning

Choice Of Project

First Choice	Plant Disease detection
Second Choice	LeetCode Clone

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose of this Document

The general purpose of a Software Requirements Specification (SRS) document for "**Plant Disease Detection Software**" is to serve as a comprehensive and detailed blueprint that outlines the requirements, functionalities, and constraints of the software system. This document serves as the cornerstone for achieving the platform's vision of fostering innovation, facilitating collaboration, and ensuring the successful development, seamless implementation, and robust user engagement of the system.

1.2. Scope of the Development Project

The scope of a "Plant Disease Detection Development Project" encompasses various aspects related to the creation and implementation of a plant disease detection system. This project aims to develop a solution that can accurately identify diseases in plants using machine learning and deep learning techniques. Below is an outline of the key components within the scope of such a project:

1. Problem Definition:

- Define the scope of the specific plant diseases to be detected (e.g., common crop diseases like rust, blight, or powdery mildew).
- Identify the target plant species or crops for disease detection.

2. Data Collection and Preparation:

- Gather a diverse and representative dataset of images of healthy and diseased plants for the selected plant species.
- Annotate and preprocess the data, ensuring it is suitable for model training.

3. Model Selection and Development:

- Choose appropriate machine learning or deep learning models for image classification.
- Develop and train the disease detection model using the collected data.
- Implement data augmentation techniques to improve model generalization.

4. User Interface and Integration:

- Design a user-friendly interface for users to input plant images (live or static) for disease detection.
- Integrate the model into the user interface for real-time or batch processing.

5. Disease Identification and Reporting:

- Implement algorithms to identify the type and severity of plant diseases.
- Provide clear and informative disease diagnosis, including the disease name, confidence score, and recommendations.

6. Visual Aids and Interpretation:
- Create heat maps or 3D images to highlight affected areas on the plant.
- Generate visual aids to help users understand the disease and its impact.

7. Database of Diseases:
- Develop a comprehensive database containing information about various plant diseases.
- Include details about symptoms, causes, prevention, and treatment methods for each disease.

8. Accuracy and Performance:
- Set performance benchmarks and accuracy targets for disease detection.
- Continuously monitor and improve the model's accuracy and speed.

9. Real-time and Mobile Capabilities:
- Explore the feasibility of enabling real-time disease detection through mobile applications or portable devices.

10. Ethical Considerations:
- Address ethical concerns related to data privacy and consent.
- Ensure responsible and unbiased AI algorithms.

11. Deployment and Scalability:
- Determine the deployment strategy (cloud-based, on-premises, edge devices).
- Ensure scalability to accommodate increasing users and data.

12. Testing and Validation:
- Conduct rigorous testing, including model validation, accuracy assessment, and usability testing.

13. Timeline and Milestones:
- Develop a project timeline with clear milestones and deliverables.

14. Risk Assessment:
- Identify potential risks, such as data quality, model accuracy, or hardware failures, and develop mitigation strategies.

15. Scalability and Future Enhancements:
- Consider future enhancements, such as expanding the range of detectable diseases or supporting multiple plant species.

This scope provides a comprehensive framework for developing a plant disease detection system. The specific scope may vary depending on project goals, available resources, and the

complexity of the plant diseases involved. A well-defined scope is essential for successful project planning and execution.

1.3. Definitions and abbreviations:

Plant disease detection software often uses various definitions and abbreviations to describe its features, processes, and terminology. Here are some common definitions and abbreviations used in the context of plant disease detection software:

No.	Term	Definition
1	Plant Disease Detection	The process of identifying and diagnosing diseases, pests, or abnormalities in plants using software, usually based on image analysis or other data sources.
2	Image Analysis	The technique of examining and processing digital images to extract relevant information for disease detection, including color, texture, and shape features.
3	Machine Learning	A subset of artificial intelligence (AI) that involves training algorithms to recognize patterns in data, often used in plant disease detection to improve accuracy.
4	Deep Learning	A type of machine learning that uses neural networks with multiple layers to learn complex patterns and representations from data.
5	Convolutional Neural Network (CNN)	A type of deep learning model commonly used in image analysis tasks, including plant disease detection.
6	Dataset	A collection of images or data used for training and testing machine learning models in plant disease detection.
7	Feature Extraction	The process of selecting and extracting relevant information or features from images, which are then used for classification or analysis.
8	Classification	The task of categorizing images or data into predefined classes or categories, such as healthy or diseased plants.

9	Segmentation	The process of dividing an image into distinct regions or segments, often used to isolate plant parts or disease symptoms.
10	False Positive	When the software incorrectly identifies a healthy plant as diseased.
11	False Negative	When the software fails to detect a disease in a diseased plant.
12	Sensitivity (True Positive Rate)	The percentage of actual diseased plants correctly identified by the software.
13	Specificity (True Negative Rate	The percentage of actual healthy plants correctly identified by the software.
14	Accuracy	The overall correctness of the disease detection results, is often measured as the percentage of correct classifications.
15	Threshold	A predefined value is used to determine whether an image or data point is classified as diseased or healthy.

Abbreviations:

No.	**Mnemonic**	**Full Form**
1	CNN	Convolutional Neural Network
2	ML	Machine Learning
3	DL	Deep Learning
4	AI	Artificial Intelligence
5	GUI	Graphical User Interface
6	ROI	Region of Interest
7	CSV	Comma-separated values (a common file format for storing data)

8	API	Application Programming Interface
9	IoT	Internet of Things
10	FP	False Positive
11	FN	False Negative
12	TPR	True Positive Rate (Sensitivity)
13	TNR	True Negative Rate (Specificity)
14	ROI	Region of Interest

These definitions and abbreviations can help users and developers communicate effectively when discussing and working with plant disease detection software.

1.4 References:

[1]Flutter. (n.d.). Retrieved from OReilly - Flutter and Dart Cookbook.
[2] Machine Learning. (n.d.). Retrieved from Hands-on Machine Learning with Scikit Learn Keras and TensorFlow 2nd Edition.
[3]Python. (n.d.). Retrieved from https://www.python.org/
[4]Flask.https://flask.palletsprojects.com/en/3.0.x/

1.5 Overview

The remaining sections of this document provide a general description, including the characteristics of the users of this project, the product's hardware, and the functional and data requirements of the product. A general description of the project is discussed in section 2 of this document. Section 2 gives the functional requirements, data requirements and constraints, and assumptions made while designing the multi-utility system. It also gives the user a viewpoint of product use. Section 3 gives the specific requirements of the product. Section 3.0 also discusses the external interface requirements and gives a detailed description of functional requirements.

2. OVERALL DESCRIPTION

2.1 Product Prescriptive:

Plant Disease Detection Software is an advanced plant disease detection system designed to assist farmers and agricultural professionals in identifying and managing diseases in various plant species. It combines cutting-edge machine learning and deep learning technologies with user-friendly features to provide accurate and timely disease diagnosis and recommendations.

Creating a product prescription for a plant disease detection system involves defining the product's features, functionalities, and specifications. Below is a product prescriptive for a plant disease detection system:

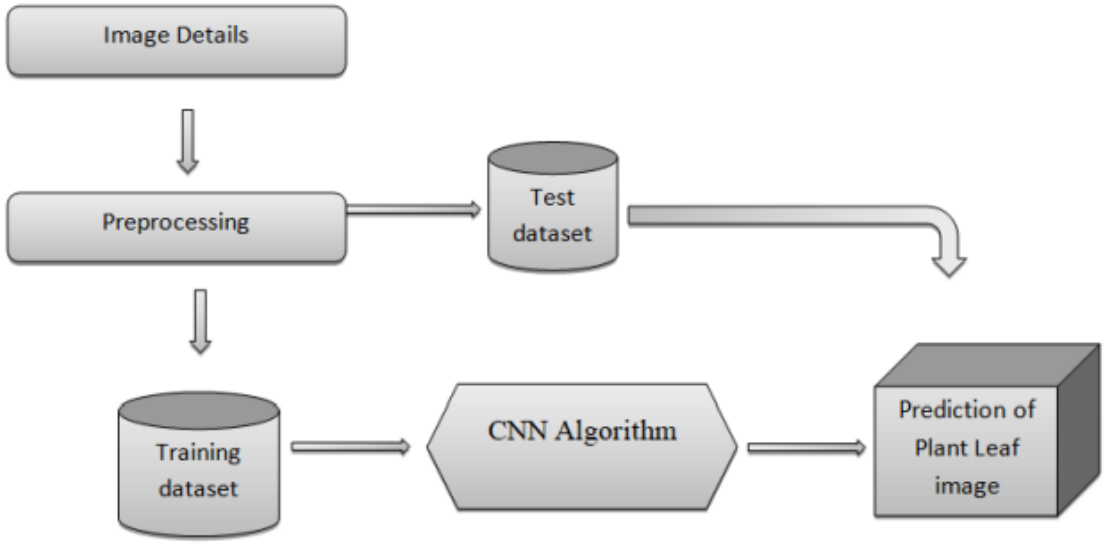


Fig. Overview Of the System

Deployment Options:

- Mobile application (iOS and Android)
- Web-based platform

Licensing:

PlantDoc Pro offers both free and premium versions. The premium version provides advanced features, such as real-time disease detection and customizable alerts, on a subscription basis.

Future Enhancements:

- Expansion of the range of detectable plant species and diseases.
- Integration with Internet of Things (IoT) devices for automated disease monitoring.

- Multilingual support for wider accessibility.

PlantDoc Pro aims to revolutionize the way plant diseases are detected and managed, ultimately contributing to increased crop yield and sustainable agriculture practices.

2.2 Product Functions:

1. User-Friendly Interface:
- Intuitive and easy-to-use interface for both novice and experienced users.
- Support for input in the form of live pictures or static images.

2. Comprehensive Disease Library:
- A vast database containing information on a wide range of plant diseases, including symptoms, causes, and recommended treatments.

3. Real-time Disease Detection:
- Capability for real-time disease detection through a mobile application or web platform.
- Quick and accurate diagnosis of plant diseases.

4. Accurate Disease Identification:
- State-of-the-art convolutional neural networks (CNNs) for precise disease identification.
- Confidence scores for disease predictions.

5. Visual Aids and Heatmaps:
- Generation of heatmaps and 3D imaging to highlight affected areas on the plant.
- Visual representation of disease severity.

6. Detailed Disease Reports:
- Clear and informative disease reports that include:
  - Disease name and type.
  - Confidence score for disease identification.
  - Recommendations for disease management.
  - Images with marked diseased areas.

7. Customizable Alerts
- Option to set up customizable alerts for specific plant species or regions.
- Receive notifications when diseases are detected.

8. Ethical Data Handling
- Strict adherence to data privacy and consent regulations.
- Transparent data collection and storage practices.

9. Scalability and Compatibility:
- Compatibility with a wide range of devices, including smartphones and tablets.
- Scalable infrastructure for accommodating growing user bases.

10. Cloud Integration:
- Cloud-based storage and processing for enhanced accessibility and scalability.
- Secure data backup and synchronization.

11. Analytics and Insights:
- Access to analytics and insights regarding disease prevalence, hotspots, and trends.
- Data-driven decision support for farmers and agricultural experts.

2.3 User Characteristics:

The user characteristics of a detection site designed for a plant disease encompass the diverse group of individuals who engage in this field. These users typically include:

1. User
- Farmers
- Agricultural professionals and consultants
- Agricultural researchers and institutions.

2. Admin

For the smooth conduct and backend management of the project.

2.4 General Constraints, Assumptions, and Dependencies:

When developing an image-based plant disease detection software, it's important to consider various constraints, assumptions, and dependencies to ensure the successful design, development, and deployment of the application. Here are some general constraints, assumptions, and dependencies to keep in mind:

Constraints:

1. Hardware Constraints: The software's performance may be constrained by the hardware it runs on, including processing power, memory, and storage. High-resolution images or complex algorithms may require more powerful hardware.

2. Data Availability: Availability and quality of plant disease image datasets can be a constraint. The software's accuracy and generalizability depend on the quality and diversity of the training data.

3. Processing Time: Real-time or near-real-time processing may be required for certain applications (e.g., in-field disease detection), which imposes constraints on algorithm efficiency and speed.

4. Network Connectivity: For cloud-based solutions or remote data sharing, a stable internet connection is essential, which may not always be available in all locations.

5. Privacy and Security: Handling sensitive plant data or images may require adherence to privacy regulations and security measures to protect user data and proprietary information.

6. Compatibility: The software may need to be compatible with various image formats, operating systems, and devices, which can be a constraint during development and maintenance.

7. Resource Availability: The availability of skilled personnel, funding, and development tools can constrain the development process.

Assumptions:

1. Image Quality: The software assumes that the input images are of sufficient quality and resolution for accurate disease detection. Poor-quality images may lead to inaccurate results.

2. Static Background: Assumption that the background in the images is relatively static and does not interfere with disease detection.

3. No Hardware Limitations: Assumes users have access to devices (e.g., smartphones, cameras) capable of capturing images suitable for analysis.

4. Data Labelling: Assumes that labeled datasets for training machine learning models are available or can be created.

5. Consistency in Plant Species: Assumes that the plant species under consideration exhibit consistent disease symptoms across different regions and environmental conditions.

Dependencies:

1. Image Processing Libraries: Dependencies on external image processing libraries and frameworks, such as OpenCV, TensorFlow, or PyTorch.

2. Machine Learning Libraries: Dependencies on machine learning libraries and frameworks for model development and training.

3. Data Sources: Dependencies on external sources for collecting and updating disease datasets.

4. APIs: Dependencies on APIs for weather data, geolocation, or other external services that may enhance disease predictions.

5. Training Data: Dependencies on access to labeled training data to develop and fine-tune machine learning models.

6. Cloud Services: Dependencies on cloud-based infrastructure for scalability, storage, or remote data access.

7. Operating System Compatibility: Dependencies on specific operating systems or libraries that may affect deployment options.

8. Development Tools: Dependencies on integrated development environments (IDEs), version control systems, and testing frameworks.

9. User Interfaces: Dependencies on graphical user interface (GUI) libraries and frameworks for creating user-friendly interfaces.

10. Regulatory Compliance: Dependencies on regulatory requirements and guidelines for handling agricultural or environmental data.

It's crucial to document and consider these constraints, assumptions, and dependencies throughout the software development lifecycle to ensure a well-designed and functional plant disease detection system. Additionally, ongoing monitoring and adaptation may be necessary as conditions and requirements change over time.

3. SPECIFIC REQUIREMENTS

3.1.External Interface Requirements:

External interface requirements for a"Plant Disease Detection software " can include interactions with external systems, services, and stakeholders. Here are some external interface requirements to consider:

1. User Interfaces:

- Web interface for users to interact with the website.
- Mobile app interface for users who prefer to use a mobile device.

2. User Authentication:

- Integration with third-party authentication providers (e.g., Google, Facebook) for user login and registration.

3. Reviews and Ratings:

- Allow users to leave reviews and ratings using external services like Disqus or a custom-built system.

3.2 Detailed Description of Functional Requirements

Functional requirements for plant disease detection software define the specific features and capabilities that the software must have to fulfill its intended purpose effectively. Below is a detailed description of functional requirements for plant disease detection software:

1. User Authentication and Management:

- User Registration: Users should be able to create accounts with basic information.
- User Login: Registered users can log in securely with their credentials.
- User Roles: Differentiate between roles (e.g., farmers, agricultural experts, administrators) with varying access levels.

2. Image Input and Processing:

- Image Upload: Allow users to upload images of plants with suspected diseases.
- Live Image Capture: Enable real-time image capture through device cameras.
- Image Preprocessing: Automatically preprocess uploaded images (e.g., resizing, normalization) for analysis.

3. Disease Detection:

- Machine Learning Models: Implement machine learning and deep learning models for disease detection.
- Disease Identification: Identify the type and severity of plant diseases from input images.

4. Disease Database:

- Comprehensive Database: Maintain a comprehensive database of known plant diseases.
- Symptom Information: Include detailed information on symptoms, causes, and management strategies for each disease.
- Regular Updates: Ensure that the disease database is regularly updated with the latest information.

5. Reporting and Recommendations:

- Disease Reports: Generate detailed disease reports including:
  - Disease name and type.
  - Confidence score for disease identification.

- Recommendations for disease management.
- Visual Aids: Create visual aids, such as heatmaps or marked images, to highlight affected areas on plants.

6. User Feedback:

- Feedback Submission: Provide a mechanism for users to submit feedback and report issues.
- Feedback Analysis: Monitor and analyze user feedback for system improvement.

7. Privacy and Security:

- Data Privacy: Ensure that user data and images are handled with strict privacy and consent.
- Secure Storage: Use encryption and secure storage practices to protect sensitive information.
- Authentication Security: Implement robust security measures for user authentication.

8. Compatibility:

- Cross-Platform Compatibility: Support various platforms, including web browsers, iOS, and Android devices.
- Device Compatibility: Ensure compatibility with a wide range of devices and screen sizes.

9. Performance and Scalability:

- Efficient Processing: Optimise the software for fast and efficient disease detection.
- Scalability: Design the system to handle a growing number of users and images.

10. Analytics and Insights:

- Analytics Dashboard: Provide users with access to analytics and insights regarding disease prevalence, hotspots, and trends.
- Data Visualization: Present data in a visually informative way, such as charts and graphs.

These functional requirements ensure that the plant disease detection software is equipped with the necessary features to accurately identify plant diseases, provide actionable recommendations, and offer a user-friendly experience. The software should also prioritize data privacy, security, and scalability to meet the needs of users in the agricultural sector.

3.3 Performance Requirements:

1. The system should be able to handle multiple concurrent users
Requests.

2. The system should specify a maximum acceptable page load time for different pages on the website. Common industry standards suggest that web pages should load within 2-3 seconds.
3. The system should be compatible with different search engines to ensure a consistent user experience.

3.4 Quality Attributes:
Quality attributes, also known as non-functional requirements, are essential for ensuring that Plant Disease Detection software operates effectively and provides a positive user experience. Here are the key quality attributes for the platform:

Usability: The site would be easy to navigate, with an intuitive user interface that allows users to browse, search, and post listings without confusion.

Performance: The site would load quickly, respond promptly to user interactions, and handle high traffic volumes efficiently, ensuring a responsive user experience.

Reliability: Users would be able to rely on the site for consistent availability without frequent errors or downtime.

Scalability: The site would be scalable to accommodate increasing user traffic and adapting to demand without performance degradation.

Security: Robust security measures, including data encryption, user authentication, and protection against fraud, would be in place to safeguard user information and transactions.

Privacy: Respect user privacy by adhering to data protection regulations and providing transparent privacy settings for users to control their data.

3.5 Other Requirements:
None at this time.

4. Methodology

Preprocessing and Training the model (CNN): The dataset is preprocessed such as Image reshaping, resizing, and conversion to an array form. Similar processing is also done on the test image. A dataset consists of about 10 different classes of leaf, out of which any image can be used as a test image for the software.

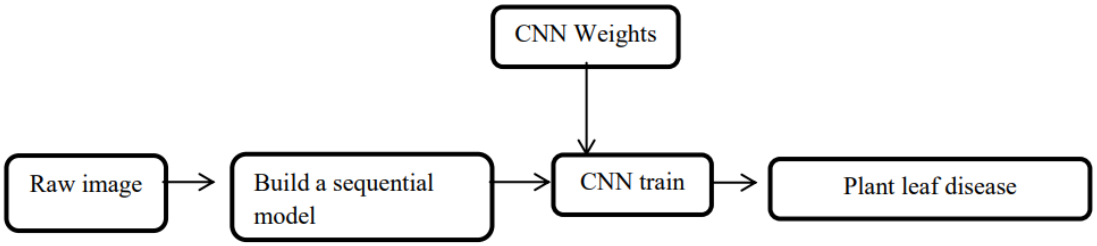


Fig 4: Methodology of the system

The training dataset is used to train the model (CNN) so that it can identify the test image and the disease it has. CNN has different layers which are Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. After the model is trained successfully, the software can identify the Plant leaf disease prediction image contained in the dataset. After successful training and preprocessing, a comparison of the test image and trained model takes place to predict the Sign language.

CNN Model steps:

Conv2d:

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel "slides" over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The output features are essentially, the weighted sums (with the weights being the values of the kernel itself) of the input features located roughly in the same location of the output pixel on the input layer.

Whether or not an input feature falls within this "roughly same location", gets determined directly by whether it"is in the area of the kernel that produced the output or not. This means the size of the kernel directly determines how many (or few) input features get combined in the production of a new output feature.

This is all in pretty stark contrast to a fully connected layer. In the above example, we have 5×5=25 input features and 3×3=9 output features. If this were a standard fully connected layer, you""d have a weight matrix of 25×9 = 225 parameters, with every output feature being the weighted sum of every single input feature. Convolutions allow us to do this transformation with only 9 parameters, with each output feature, instead of "looking at" every input feature, only getting to "look" at input features coming from roughly the same location. Do take note of this, as it"ll be critical to our later discussion.

MaxPooling2D layer
Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool_size) for each input channel. The window is shifted by strides along each dimension

The resulting output, when using the "valid" padding option, has a spatial shape (number of rows or columns) of output_shape = math.floor((input_shape - pool_size) / strides) + 1 (when input_shape >= pool_size)

The resulting output shape when using the "same" padding option is: output_shape = math.floor((input_shape - 1) / strides) + 1

Input shape

• If data_format='channels_last': 4D tensor with shape (batch_size, rows, cols, channels).
• If data_format='channels_first': 4D tensor with shape (batch_size, channels, rows, cols).

Output shape

•If data_format='channels_last': 4D tensor with shape (batch_size, pooled_rows, pooled_cols, channels).
• If data_format='channels_first': 4D tensor with shape (batch_size, channels, pooled_rows, pooled_cols)

Flatten layer
It is used to flatten the dimensions of the image obtained after convolving it. Dense: It is used to make this a fully connected model and is the hidden layer. Dropout: It is used to avoid overfitting on the dataset and is dense if the output layer contains only one neuron which decides to which category image belongs.

Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4)

Flatten has one argument as follows

keras.layers.Flatten(data_format = None)

data_format is an optional argument and it is used to preserve weight ordering when switching from one data format to another data format. It accepts either channels_last or channels_first as value. channels_last is the default one and it identifies the input shape as (batch_size, ..., channels) whereas channels_first identifies the input shape as (batch_size, channels, ...)

Dense layer
Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is

a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True). These are all attributes of Dense

Note: If the input to the layer has a rank greater than 2, then Dense computes the dot product between the inputs and the kernel along the last axis of the inputs and axis 0 of the kernel (using tf.tensordot). For example, if input has dimensions (batch_size, d0, d1), then we create a kernel with shape (d1, units), and the kernel operates along axis 2 of the input, on every sub-tensor of shape (1, 1, d1) (there are batch_size * d0 such sub-tensors). The output in this case will have shape (batch_size, d0, units).

Besides, layer attributes cannot be modified after the layer has been called once (except the trainable attribute). When a popular kwarg input_shape is passed, then keras will create an input layer to insert before the current layer. This can be treated equivalent to explicitly defining an InputLayer.

Input shape
N-D tensor with shape: (batch_size, ..., input_dim). The most common situation would be a 2D input with shape (batch_size, input_dim).

Output shape
N-D tensor with shape: (batch_size, ..., units). For instance, for a 2D input with shape (batch_size, input_dim), the output would have shape (batch_size, units).

Dropout layer
The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using model.fit, training will be appropriately set to True automatically, and in other contexts, you can set the kwarg explicitly to True when calling the layer.

(This is in contrast to setting trainable=False for a Dropout layer. trainable does not affect the layer's behaviour, as Dropout does not have any variables/weights that can be frozen during training.)

seed: A Python integer to use as random seed.

Image Data Generator:
It resizes the image, applies shear in some range, zooms the image and does horizontal flipping with the image. This Image Data Generator includes all possible orientation of the image.

Training Process:
train_datagen.flow_from_directory is the function that is used to prepare data from the train_dataset directory Target_size specifies the target size of the image. Test_datagen.flow_from_directory is used to prepare test data for the model and all is similar as above. fit_generator is used to fit the data into the model made above, other factors used are steps_per_epochs tells us about the number of times the model will execute for the training data.

Epochs:
It tells us the number of times the model will be trained in forward and backward pass.

Validation process:
Validation_data is used to feed the validation/test data into the model. Validation_steps denotes the number of validation/test samples.

Model Used
InceptionV3 is a convolutional neural network architecture that is part of the Inception family, developed by Google. It was introduced in 2015 and is an improvement over its predecessor, InceptionV1 (GoogLeNet). InceptionV3 is specifically designed for image recognition and classification tasks.
The key innovation in the Inception series of models, including InceptionV3, is the utilization of the "Inception module." This module is a building block that allows the network to capture features at multiple scales by performing convolutions of different sizes (1x1, 3x3, 5x5) simultaneously and concatenating their outputs. This enables the network to efficiently learn features at various levels of abstraction.
InceptionV3 introduced several improvements over its predecessors, including:
Factorization: InceptionV3 replaced large convolutions (e.g., 5x5) with factorized smaller convolutions (two consecutive 3x3 convolutions), reducing the computational load while maintaining expressiveness.
Reduction in Parameters: By using factorized convolutions and carefully designed architecture, InceptionV3 reduced the number of parameters compared to the original Inception model (GoogLeNet).
Auxiliary Classifiers: It also introduced auxiliary classifiers at intermediate layers during training, aiding in combating the vanishing gradient problem and improving convergence.
InceptionV3, like its predecessors, was primarily trained on the ImageNet dataset, which includes millions of labeled images across thousands of classes. It achieved strong performance on various image classification benchmarks.

Final Testing Accuracy Report

```
Found 8339 images belonging to 41 classes.
66/66 [==============================] - 82s 1s/step - loss: 0.1489 - accuracy: 0.9738
Time taken to evaluate the model: 82.84821139144897
Test Loss: 0.1408977735953331
Test Accuracy: 0.9738183482170105
66/66 [==============================] - 44s 654ms/step
              precision    recall  f1-score   support

           0     1.0000    0.9524    0.9756       126
           1     1.0000    1.0000    1.0000       124
           2     1.0000    1.0000    1.0000        55
           3     0.9734    1.0000    0.9865       329
           4     1.0000    0.9952    0.9976       210
           5     1.0000    0.9882    0.9941       170
           6     0.3684    0.7000    0.4828        10
           7     0.5000    1.0000    0.6667        10
           8     0.9000    0.9000    0.9000        10
           9     0.8182    0.9000    0.8571        10
          10     1.0000    0.4000    0.5714        10
          11     0.8111    0.6293    0.7087       116
          12     0.9423    0.6405    0.7626       153
          13     0.1368    0.4194    0.2063        31
          14     0.9259    0.9804    0.9524       102
          15     1.0000    1.0000    1.0000       230
          16     0.9895    0.9594    0.9742       197
          17     0.9872    1.0000    0.9936       232
          18     1.0000    0.9746    0.9871       236
          19     0.9583    1.0000    0.9787       276
          20     1.0000    1.0000    1.0000       215
          21     0.9882    1.0000    0.9941        84
          22     1.0000    1.0000    1.0000       459
          23     1.0000    0.9722    0.9859        72
          24     1.0000    0.9950    0.9975       199
          25     1.0000    0.9966    0.9983       295
          26     1.0000    1.0000    1.0000       200
          27     1.0000    0.9600    0.9796       200
          28     1.0000    1.0000    1.0000        30
          29     1.0000    1.0000    1.0000       221
          30     0.9785    1.0000    0.9891        91
          31     0.9953    0.9882    0.9917       425
          32     1.0000    0.9400    0.9691       200
          33     0.9615    0.9843    0.9728       381
          34     0.9947    0.9895    0.9921       190
          35     0.9831    0.9859    0.9845       354
          36     0.9937    0.9463    0.9694       335
          37     1.0000    0.9536    0.9762       200
          38     0.9888    1.0000    0.9903      1071
          39     0.9867    1.0000    0.9933        74
          40     0.9695    1.0000    0.9845       310

    accuracy                         0.9730      8339
   macro avg     0.9303    0.9305    0.9211      8339
weighted avg     0.9803    0.9730    0.9752      8339
```

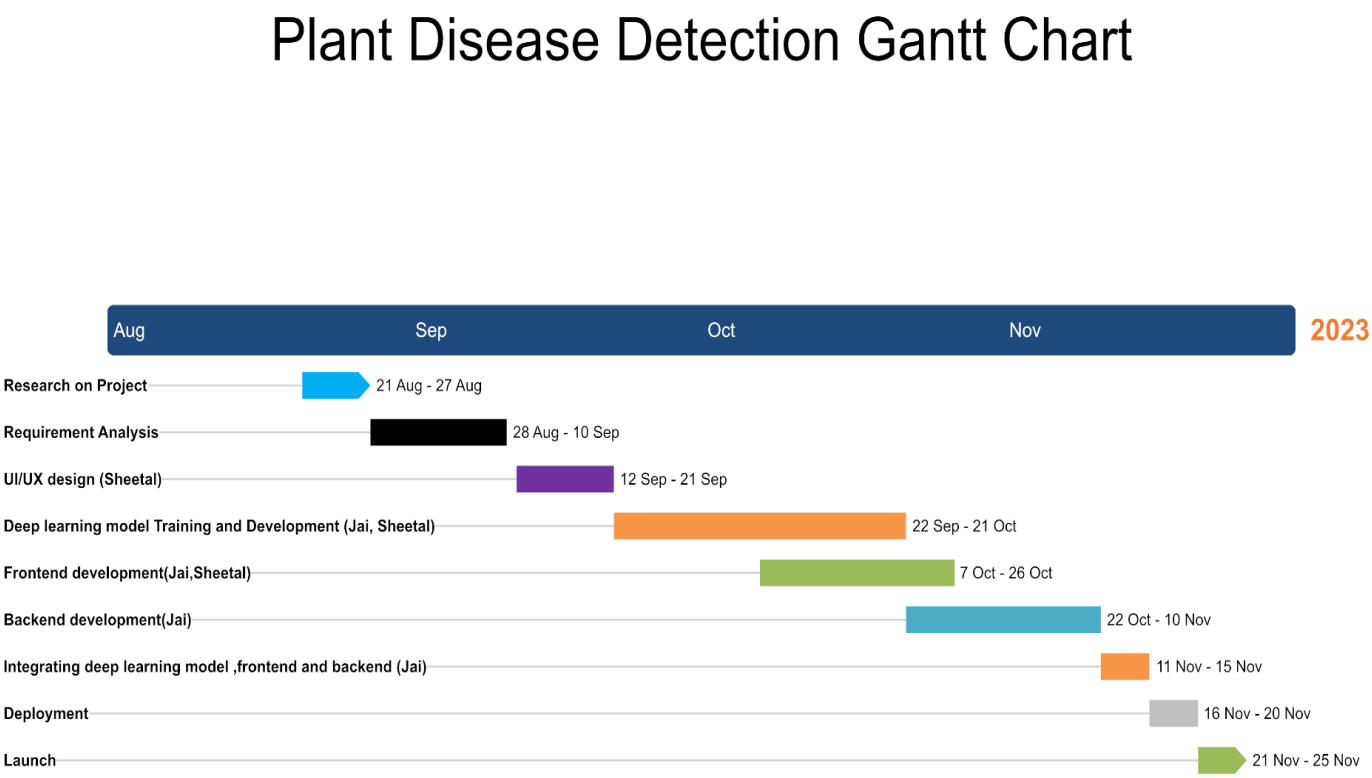
5. Document Approves:

SRS for Plant Disease Detection Software approved by:
Arwinder Dhillon

(name)
Designation: Miss
Date: 22 September,2023

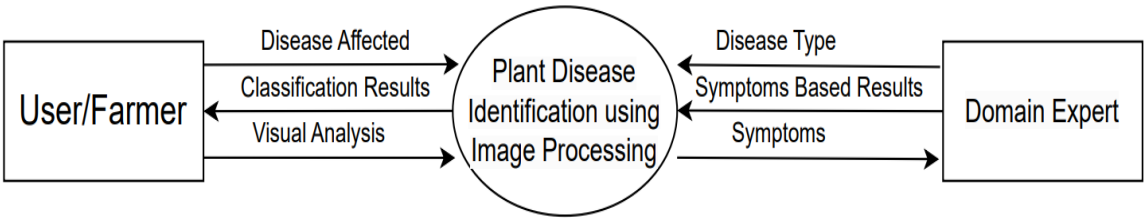
6.Diagrams

Gantt Chart

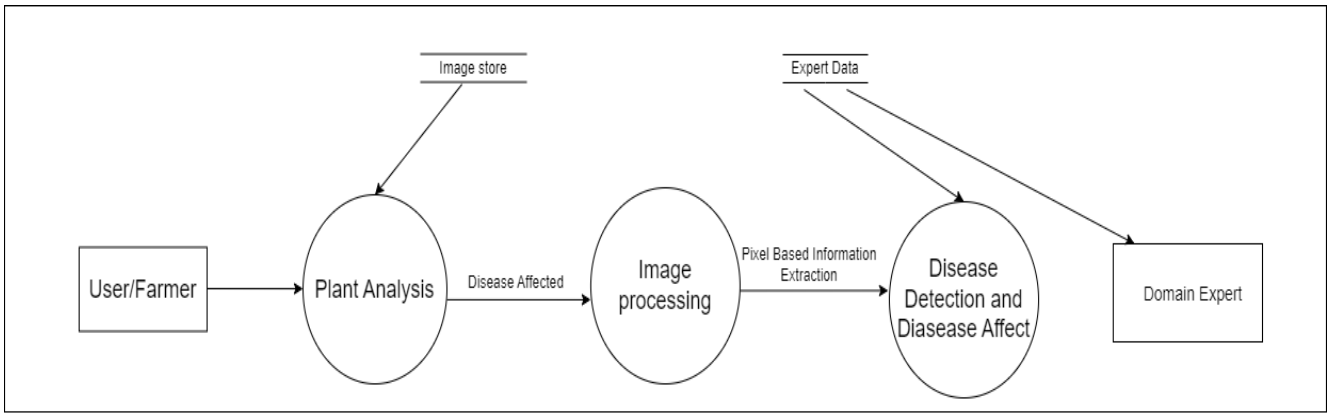


Data Flow Diagrams

Level 0

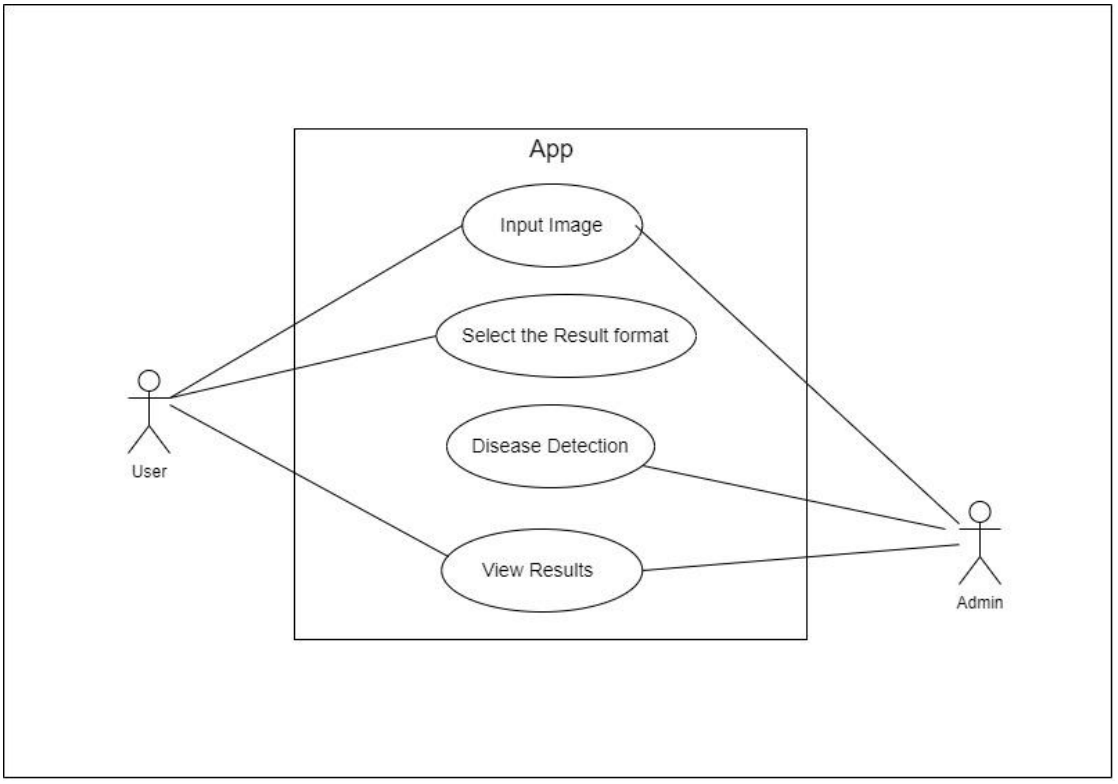


Level 1



Use Case Diagram

App use case diagram



CNN model use case diagram

Use Case Scenario:

Diagram 1

1.Use Case Title	Image-based Plant Disease Detection software
2.Abbreviated Title	Plant Disease detector
3.Use case ID	1
4.Actors	User,Admin
5. Description: The User will give the input. The input will be an image, after this the user will input the type of output he or she demands and the output will be displayed. 5.1 Pre Conditions: User must have the image of the plant 5.2 Task sequence: 1. Open the Application. 2. The User will input the image. 3. Choose the output type. 4. Agree to the terms and conditions of the application. 5. Click on the "Submit" button. 5.3 Post Conditions: 1. User must have a proper image of the plant	
6. Modification History: Date 18-Sep-2023	
7.Author: 1.Jai 2.Sheetal	

Diagram 2

1.Use Case Title	Image-based Plant Disease Detection software
2.Abbreviated Title	Plant Disease detector
3.Use case ID	2
4.Actors	User,Actor
5. Description: Model is trained using an infected planet disease image dataset.The model's primary function is to classify images of plants either as healthy or diseased and,when diseased, to specify the type of diseases. After training the model is tested using test data. 5.1 Pre Conditions: User must have the image of the plant 5.2 Task sequence: 1. Go to the website's page. 2. The User will input the image. 3. Choose the output type. 4. Agree to the terms and conditions of the website. 5. Click on the "Submit" button. 5.3 Post Conditions: 1. User must have a proper image of the plant	
6. Modification History: Date 18-Sep-2023	
7.Author: 1.Jai 2.Sheetal	

Use Case Templates

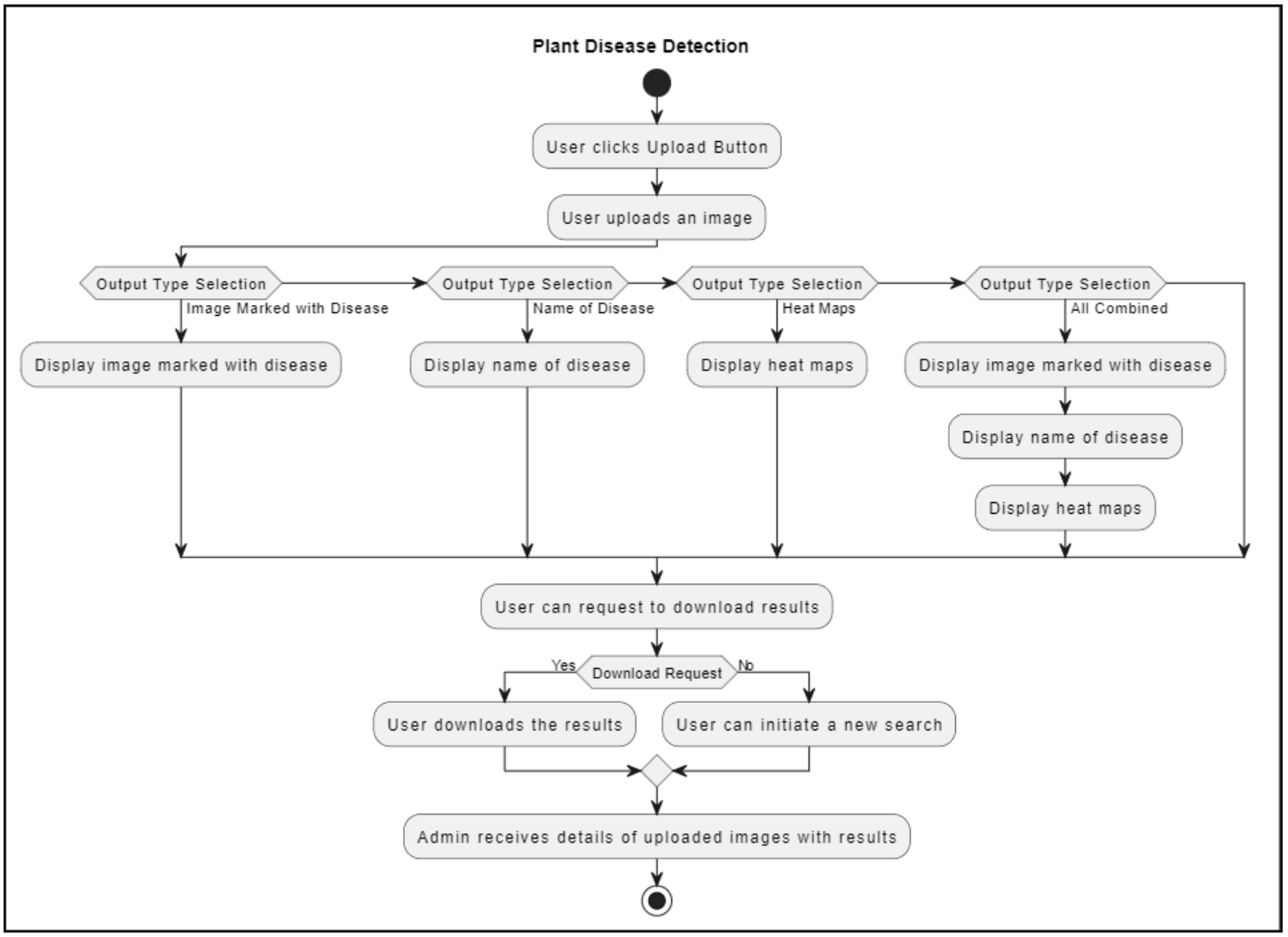
Table 1 : Use case Templates

Trigger	The event or action that initiates a specific requirement
Precondition	The state or condition that must be true before the requirement can be executed.
Basic path	The sequence of steps that describe the normal or successful execution of the requirement.
Post condition	The state or condition that must be true after the requirement has been successfully executed.
Exception paths	The alternative sequence of steps that occur when the basic path encounters errors or exceptions.
others	Additional notes or details relevant to the requirement but not covered explicitly by the above categories.

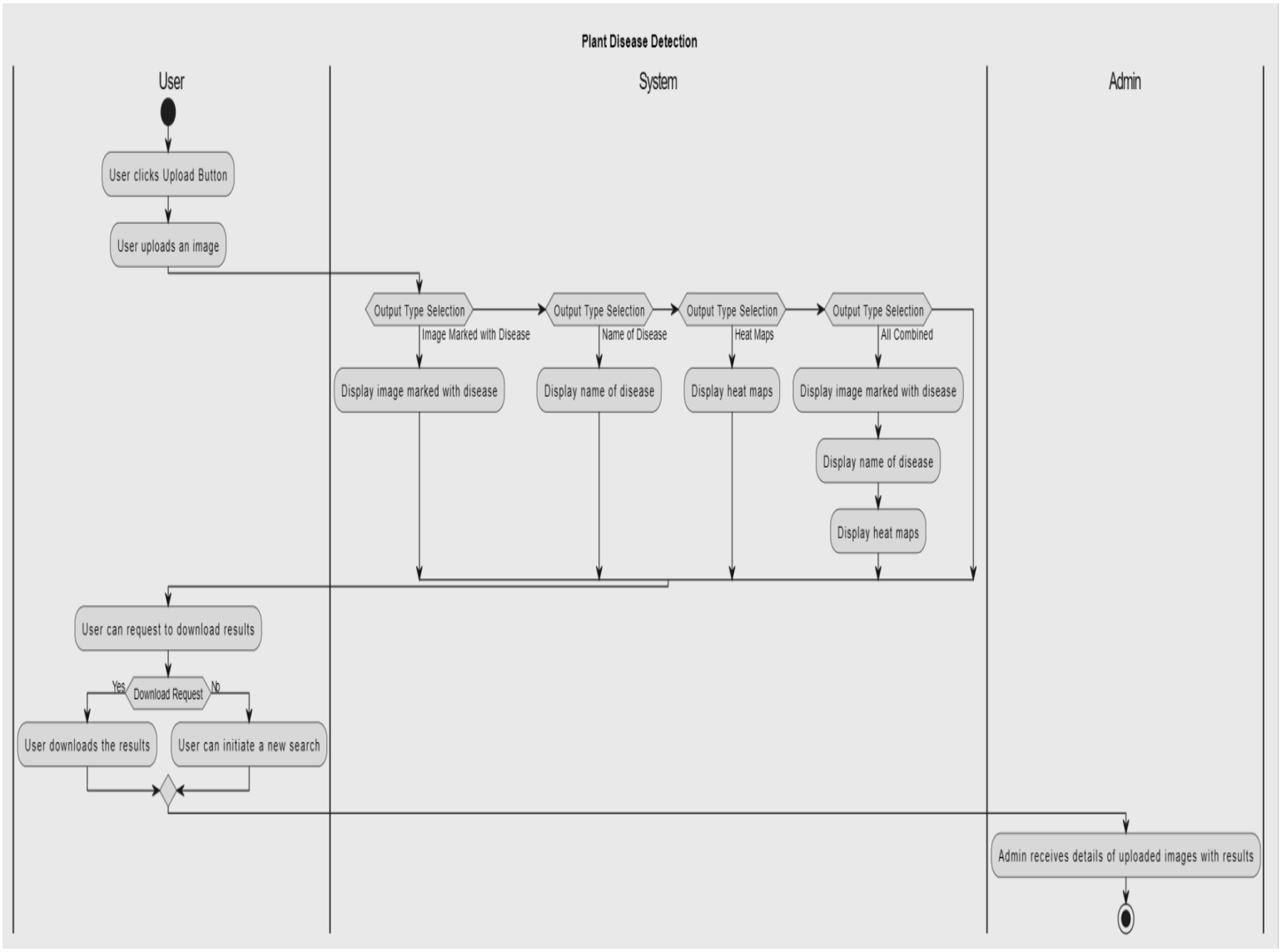
Table 2 :Plant detection

Trigger	User clicks on the upload icon
Precondition	User has access to upload
Basic Path	User provides the image System runs the model System returns the results
Post Condition	User is taken to the prediction page
Exception Path	Invalid input led to error messages
Others	Admin can store the history of the searches

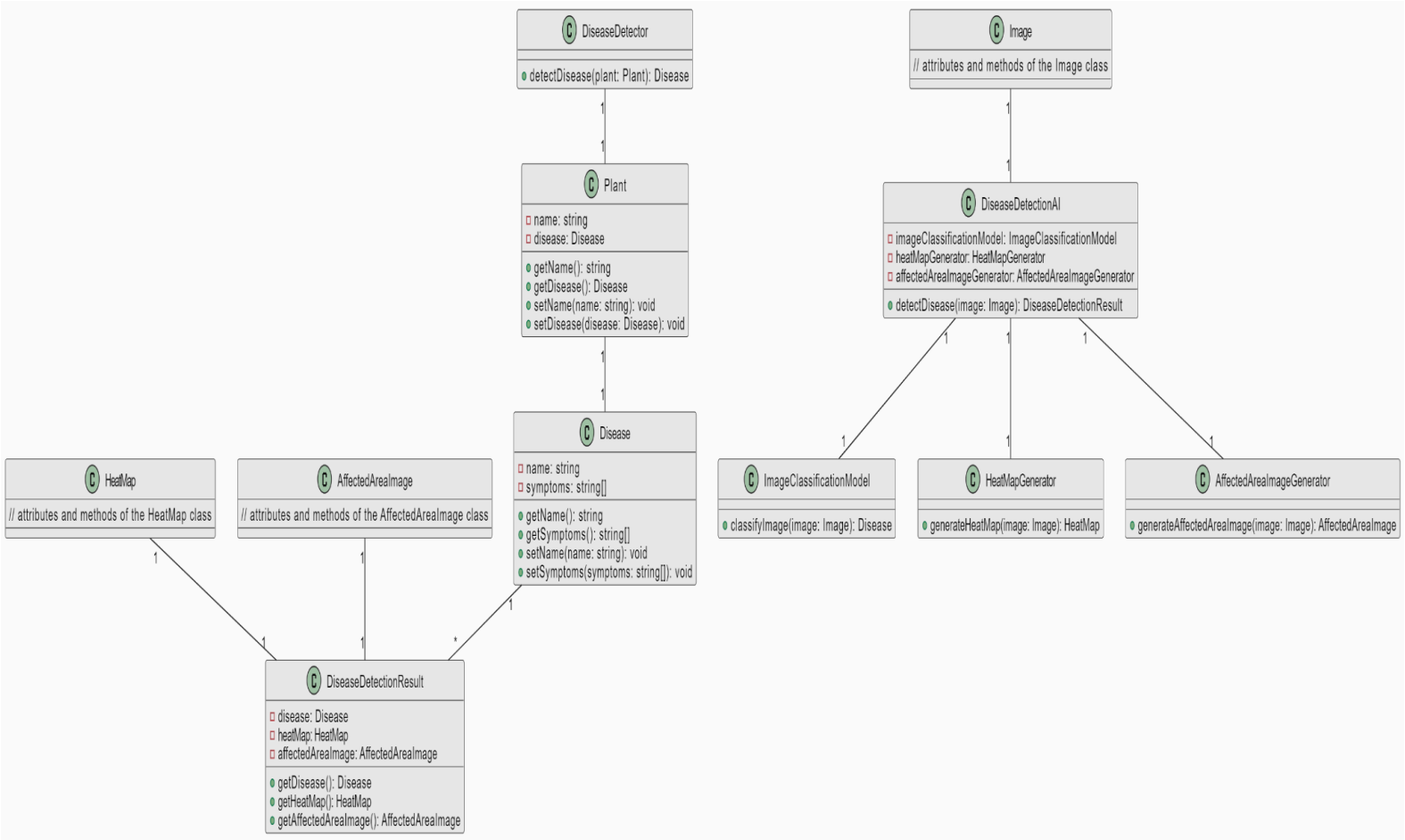
Activity Diagram



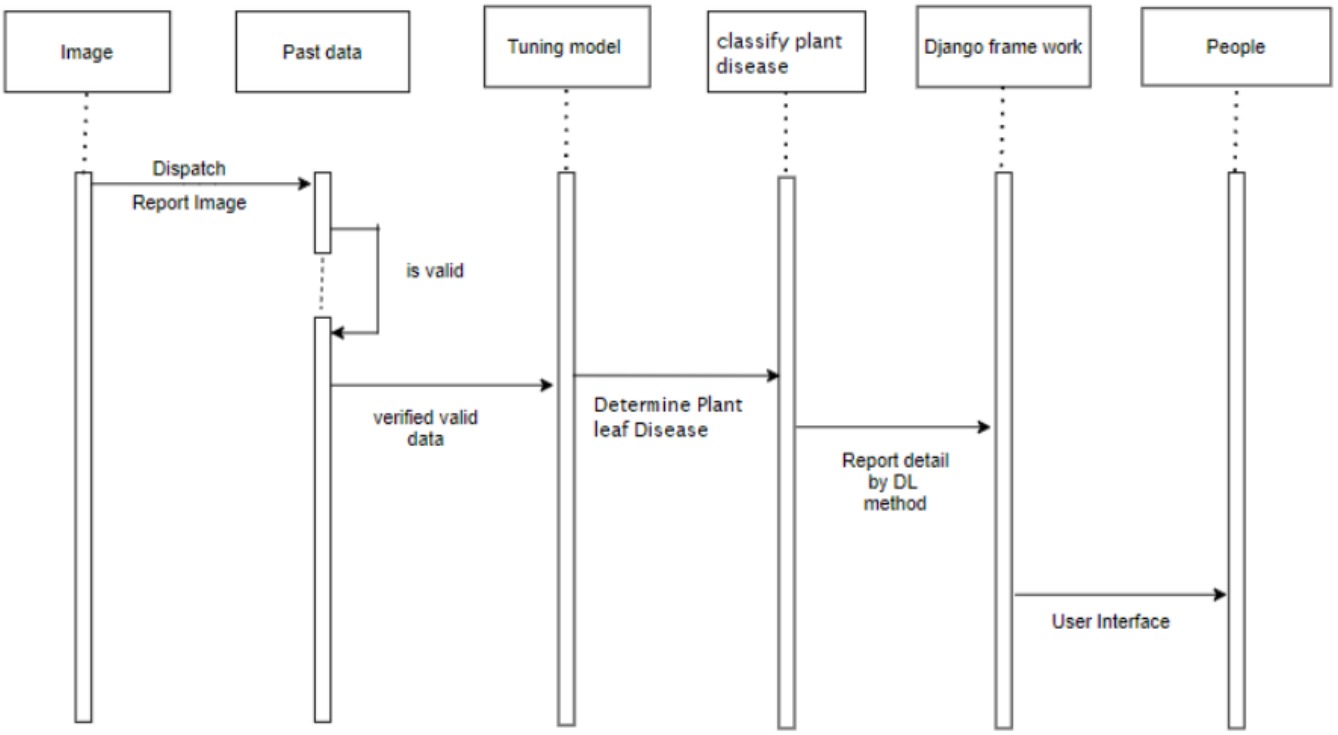
Swimlane Diagram:



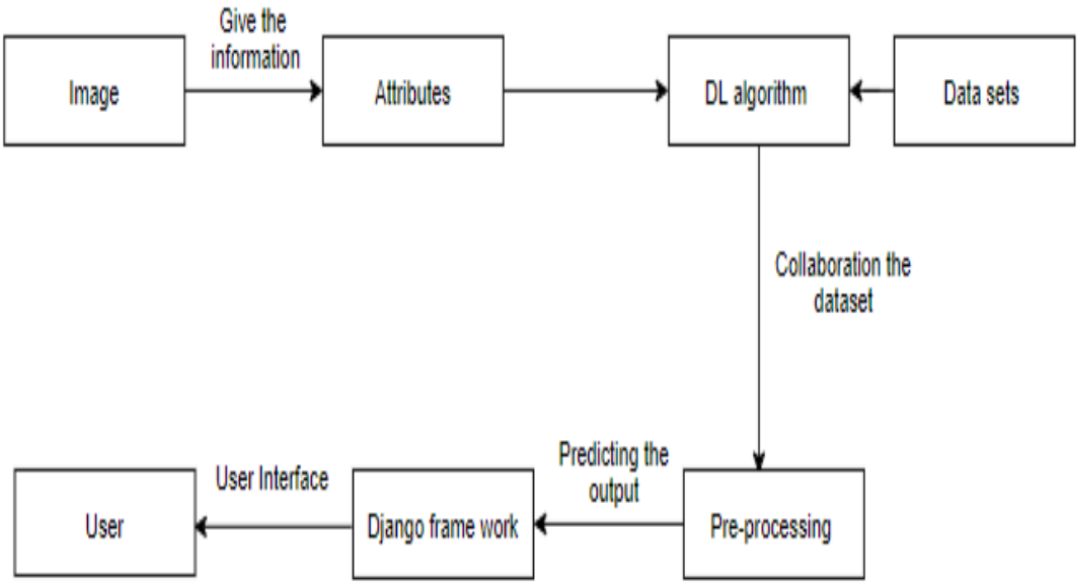
Class Diagram:



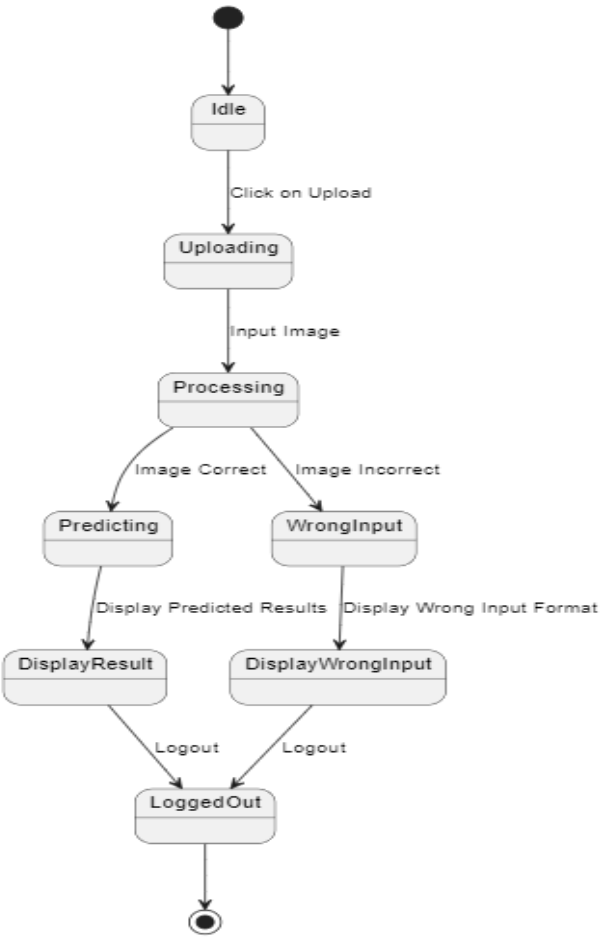
Sequence Diagram



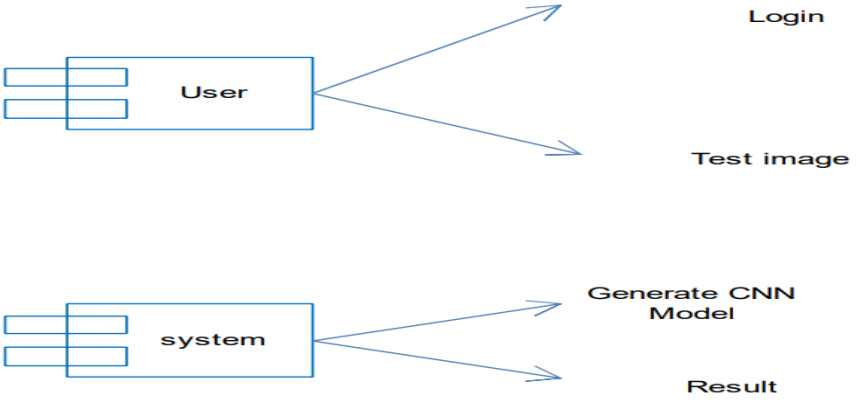
Collaboration diagram



State space diagram



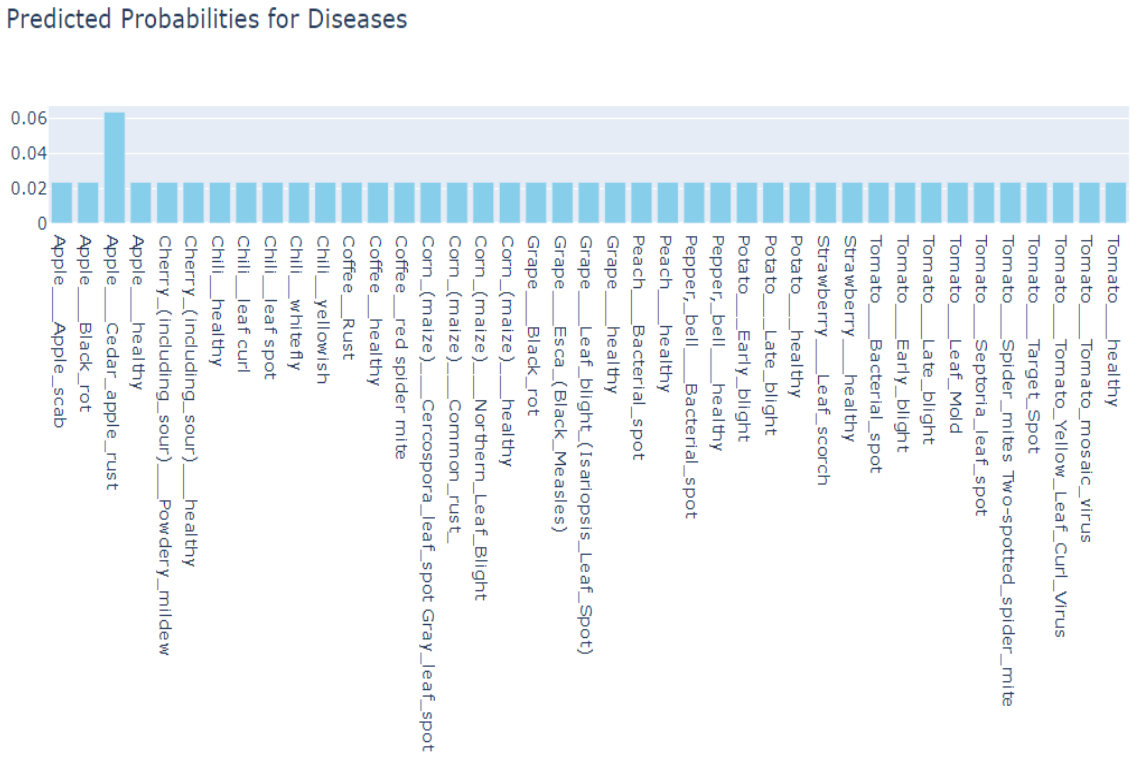
Component diagram



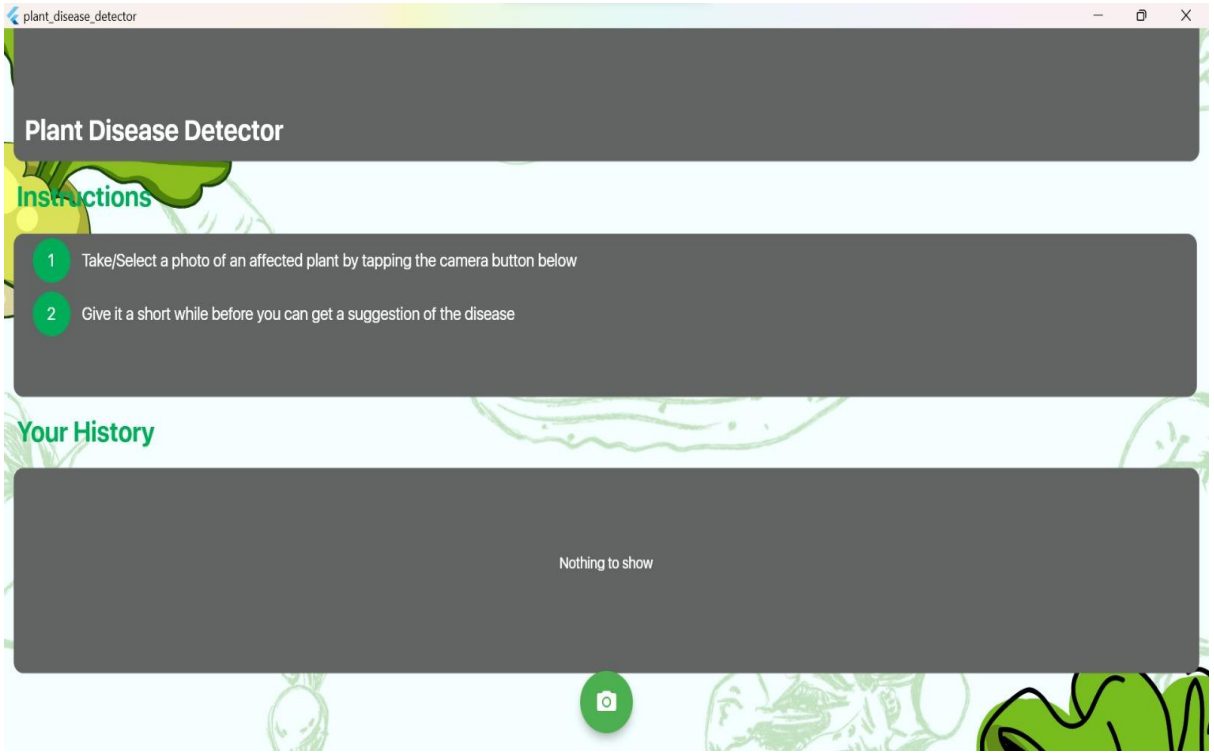
ScreenShots of the Working Project

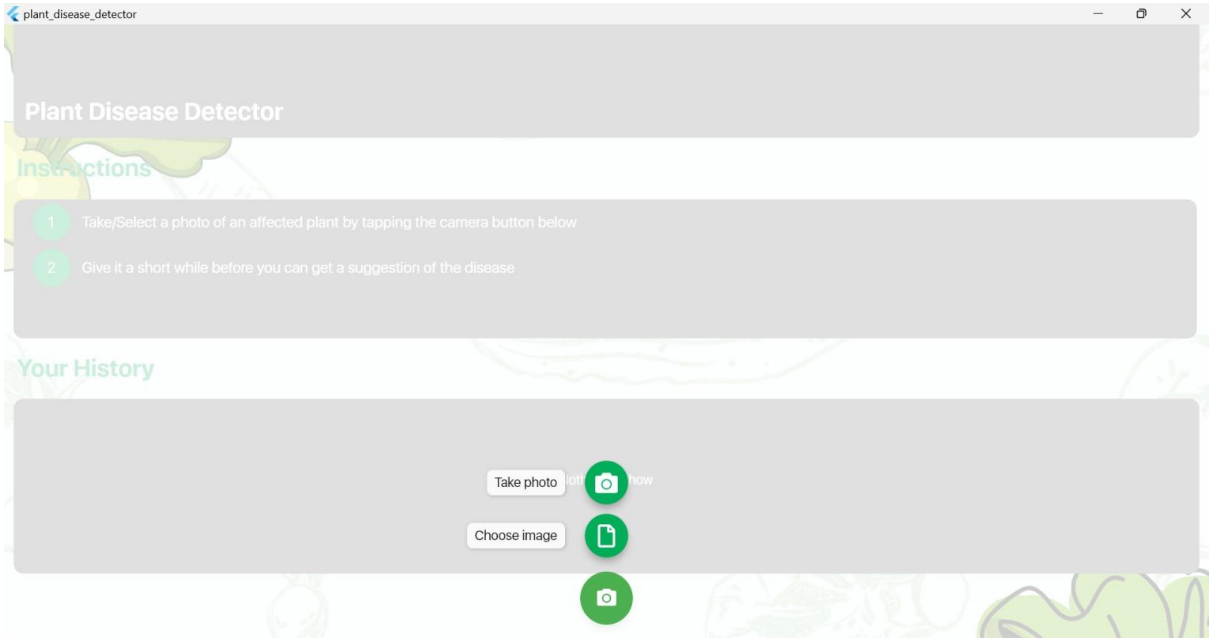
Model Working


```
Confidence Score: 0.9993820190429688
```


Predicted Probabilities for Diseases
0.06
0.04
0.02
0
Apple___Apple_scab
Apple___Black_rot
Apple___Cedar_apple_rust
Apple___healthy
Cherry_(including_sour)___Powdery_mildew
Cherry_(including_sour)___healthy
Chili__healthy
Chili__leaf curl
Chili__leaf spot
Chili__whitefly
Chili__yellowish
Coffee__Rust
Coffee__healthy
Coffee__red spider mite
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot
Corn_(maize)___Common_rust_
Corn_(maize)___Northern_Leaf_Blight
Corn_(maize)___healthy
Grape___Black_rot
Grape___Esca_(Black_Measles)
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)
Grape___healthy
Peach___Bacterial_spot
Peach___healthy
Pepper,_bell___Bacterial_spot
Pepper,_bell___healthy
Potato___Early_blight
Potato___Late_blight
Potato___healthy
Strawberry___Leaf_scorch
Strawberry___healthy
Tomato___Bacterial_spot
Tomato___Early_blight
Tomato___Late_blight
Tomato___Leaf_Mold
Tomato___Septoria_leaf_spot
Tomato___Spider_mites Two-spotted_spider_mite
Tomato___Target_Spot
Tomato___Tomato_Yellow_Leaf_Curl_Virus
Tomato___Tomato_mosaic_virus
Tomato___healthy

Working App


plant_disease_detector
Plant Disease Detector
Instructions
1 Take/Select a photo of an affected plant by tapping the camera button below
2 Give it a short while before you can get a suggestion of the disease
Your History
Nothing to show


plant_disease_detector
Plant Disease Detector
Instructions
1 Take/Select a photo of an affected plant by tapping the camera button below
2 Give it a short while before you can get a suggestion of the disease
Your History
Take photo
Choose image


Suggestions
Disease name
Tomato Septoria Leaf Spot
Possible causes
It is a fungus and spreads by spores most rapidly in wet or humid weather. Attacks plants in the nightshade family, and can be harbored on weeds within this family.
Possible solution
Remove infected leaves immediately, use organic fungicide options.