TEST PLAN FOR

PLANT DISEASE DETECTION SYSTEM USING DEEP LEARNING

Made By:
Shakti Maddeshiya 2000290120141
Saurabh Mishra 2000290120
Shivanshu Mishra 2000290120

Test Manager:
Prof. Shreela Pareek

Guide:
PROF. RAJ KUMAR

ChangeLog

Version	**Change Date**	**By**	**Description**
version number	Date of Change	Name of person who made changes	Description of the changes made
001	05-11-2023	Shakti Maddeshiya	Initial Draft

1 Introduction

Plants play a crucial role in people's daily life in the form of vegetables, fruits and various consumer goods. So, identifying plant leaf diseases is very important for farmers in producing quality crops and also to enhance agricultural productivity. But, due to wide variety of diseases, leaf disease identification by human eye is time-consuming, difficult and also less accurate. Therefore, there is a need for Automatic Leaf Disease Identification techniques which helps farmers to identify various diseases with less effort, less time and more accurate. These techniques also helps in healthy monitoring of fields that ensures quality agricultural products. For this case study, different diseased-leaf images of plant are collected. Our task is to classify a given leaf among various diseases with high accuracy.

1.1 Scope

1.1.1 In Scope

The scope for a "PLANT DISEASE DETECTION SYSTEM USING DEEP LEARNING" system would encompass various features and requirements that need to be tested to ensure the system functions correctly and meets user needs. Here is a broad outline of potential features and requirements that might be included in the scope:

<u>Functional Requirements:</u>

1. Image Acquisition and Input:

- The system shall be able to accept input images of plant leaves captured by various devices such as cameras or smartphones.
- Supported image formats should include commonly used formats like JPEG and PNG.

2.Image Preprocessing:

- The system shall preprocess input images to enhance features relevant to disease detection, including but not limited to resizing, normalization, and noise reduction.
- It should handle variations in lighting conditions and backgrounds.

3.Disease Classification:

- The system shall employ a deep learning model for classifying plant diseases based on input images.
- It should support multiple classes corresponding to different types of plant diseases.

4.Model Training:

- There should be a mechanism to train and fine-tune the deep learning model using a dataset of labeled images.
- The training process should be configurable, allowing the use of different hyperparameters and architectures.

5.Model Evaluation:

- The system shall provide metrics for evaluating the performance of the trained model, such as accuracy, precision, recall, and F1 score.
- Users should be able to assess the model's effectiveness and reliability.

6.Real-time Inference:

- The system shall perform real-time inference, providing rapid results for disease detection on new input images.
- Inference response time should be optimized for user efficiency.

7.User Interface:

- The system shall have a user-friendly interface to upload images, initiate disease detection, and display the results.
- It should provide clear and understandable visualizations of the detected diseases and confidence scores.

8.Integration with External Systems:

- The system shall allow integration with external systems or databases for data sharing and storage.
- It should support common APIs or file formats for seamless interaction with other applic ations.

9.Security and Privacy:

- The system shall implement security measures to protect user data and ensure the privacy of sensitive information.
- Access controls should be in place to manage user permissions.

10.Scalability:

- The system should be designed to handle a scalable number of users and a growing dataset.

- It should be able to accommodate increased computational demands as the dataset and user base expand.

11.Documentation:

- The system shall come with comprehensive documentation, including user manuals, API documentation, and a guide for model retraining.

Non-Functional Requirements:

1.Performance:

- The system shall be capable of processing a minimum of X images per second during real-time inference.

- Inference response time shall not exceed Y seconds for any given image.

2.Reliability:

- The system shall have an uptime of at least 99% over any given month.

- It should be resilient to temporary disruptions, with the ability to recover gracefully.

3.Scalability:

- The system should be designed to scale horizontally to handle an increasing number of simultaneous users.

- It should be able to handle a growing dataset without significant degradation in performance.

4.Usability:

- The user interface shall be intuitive and easy to use, requiring minimal training for end-users.

- The system shall provide clear and understandable error messages to assist users in troubleshooting.

5.Maintainability:

- The system shall be modular and well-documented to facilitate ease of maintenance.

- Updates and patches shall be deployable with minimal downtime.

Testing these features and requirements would involve a series of steps, including unit testing, integration testing, system testing, and acceptance testing, to validate that the system works as intended, is user-friendly, secure, and reliable. It's also essential to test the system under different conditions and loads to ensure it performs well when deployed in a real-world environment.

1.1.2 Out of Scope

Out Of Scope defines the features, functional or non-functional requirements of the software that **will NOT be** tested

1.Climate and Weather Monitoring: Monitoring weather conditions, such as temperature, humidity, or precipitation, might be out of scope unless explicitly specified in the project requirements.

2.Soil Analysis: If the project focuses solely on leaf disease detection, analyzing soil conditions or composition might be considered out of scope.

3.Geographic Information System (GIS) Integration: Incorporating geographic data or mapping functionalities might be considered beyond the project's scope unless explicitly stated.

4.Multi-Language Support: If the project is initially designed for a specific language or region, adding support for multiple languages might be considered an out-of-scope enhancement.

5.Hardware Integration: If the project is intended as a software application, integration with specific hardware components (e.g., specialized sensors) might be considered out of scope unless stated otherwise.

6.User Authentication and Authorization: Depending on the project's focus, implementing complex user authentication and authorization mechanisms might be considered beyond the primary scope.

7.Real-time Data Streaming: If the project is not explicitly designed for real-time data processing, implementing real-time data streaming features might be considered out of scope.

8.Mobile Application Development: If the project is scoped for web applications, developing a mobile application might be considered out of scope unless it's specifically included in the project requirements.

1.2 Quality Objective

The overall objective of testing for a " PLANT DISEASE DETECTION SYSTEM USING DEEP LEARNING " system is to ensure that the system is reliable, efficient, user-friendly, and fulfills all the specified requirements. Here's a more detailed breakdown of the objectives:

1.Accuracy: Achieve a minimum accuracy rate of 90% in identifying and classifying plant diseases using the deep learning model.

2.Precision and Recall: Attain a balance between precision and recall, ensuring that the model can correctly identify diseased plants (precision) while minimizing false negatives (recall).

3.Robustness: Ensure that the deep learning model performs reliably across different environmental conditions, such as variations in lighting, plant types, and image qualities.

4.Scalability: Design the system to handle a scalable amount of data, ensuring that the model's performance remains consistent as the dataset size increases.

5.User-Friendly Interface: Create an intuitive and user-friendly interface for the application, allowing users to easily interact with and interpret the results of the plant disease detection.

6.Cross-Browser and Cross-Device Compatibility: Ensure that the application works seamlessly across various web browsers and devices, providing a consistent user experience.

7.Security: Implement measures to secure user data and maintain the confidentiality and integrity of the plant disease information.

8.Performance: Optimize the performance of the application, aiming for fast response times and minimal latency in delivering results.

9.Documentation: Provide comprehensive documentation for the project, including code documentation, user manuals, and system architecture documentation.

10.Feedback Mechanism: Implement a feedback mechanism to gather user feedback, allowing continuous improvement of the system based on user input and evolving requirements.

11.Adherence to Standards: Ensure that the project adheres to relevant coding standards, ethical guidelines, and industry best practices in machine learning and plant pathology.

12.Maintainability: Design the system in a way that facilitates easy maintenance, updates, and future enhancements to adapt to evolving technologies and requirements.

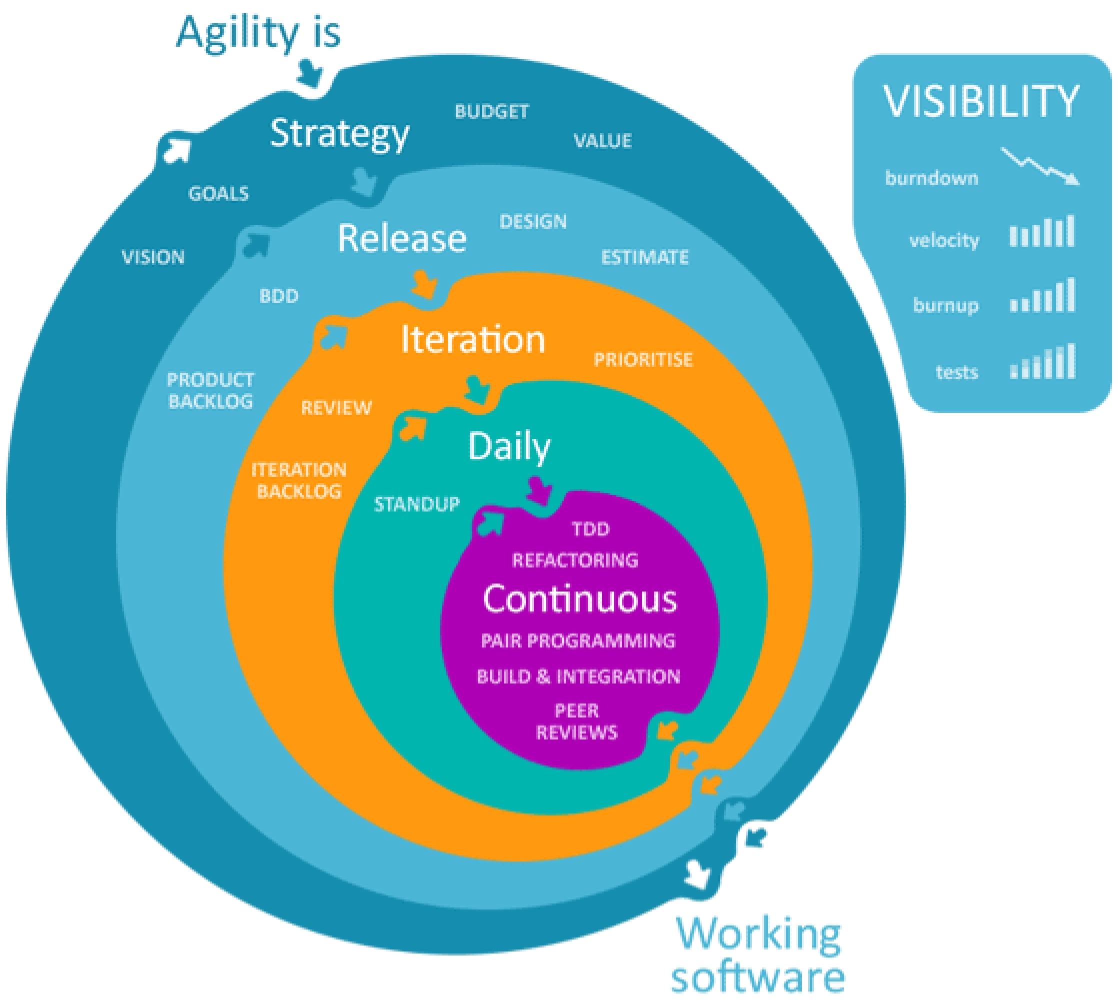
1.3 Roles and Responsibilities

Detail description of the Roles and responsibilities of different team members like

- QA Analyst: Shakti Maddeshiya
- Test Manager: Mrs. Shreela Parikh
- Configuration Manager: Mrs. Neha Shukla
- Developers: Shakti Maddeshiya
- Installation Team: Shakti Maddeshiya, Saurabh Mishra, Shivanshu Singh

2 Test Methodology

2.1 Overview



Agile Software Development is a software development methodology that values flexibility, collaboration, and customer satisfaction. It is based on the Agile Manifesto, a set of principles for software development that prioritize individuals and interactions, working software, customer collaboration, and responding to change.
Agile Software Development is an iterative and incremental approach to software development that emphasizes the importance of delivering a working product quickly and frequently. It involves close collaboration between the development team and the customer to ensure that the product meets their needs and expectations.

The Agile Software Development process typically consists of the following steps:

1. Requirements Gathering: The customer's requirements for the software are gathered and prioritized.

2. Planning: The development team creates a plan for delivering the software, including the features that will be delivered in each iteration.

3. Development: The development team works to build the software, using frequent and rapid iterations.

4. Testing: The software is thoroughly tested to ensure that it meets the customer's requirements and is of high quality.

5. Deployment: The software is deployed and put into use.
6. Maintenance: The software is maintained to ensure that it continues to meet the customer's needs and expectations.

2.2 Test Levels

For a "PLANT DISEASE DETECTION SYSTEM USING DEEP LEARNING " system, the test levels can be organized to ensure thorough validation at different stages of development, from individual components to the full, integrated system. Here's how the test levels could be structured:

1.Unit Testing:
Objective:
- Verify the correctness of individual components and functions within the system.

Scope:
- Test individual functions responsible for image preprocessing.
- Validate the correctness of disease classification algorithms.

2.Integration Testing:
Objective:
- Validate the interaction and collaboration between different modules and components.

Scope:
- Test the integration of image acquisition and preprocessing modules.
- Validate the integration of the deep learning model with the overall system.

3.Component Testing:
Objective:
- Test the behavior of specific components or subsystems in isolation.

Scope:
- Test the image acquisition module to ensure it correctly captures and reads images.
- Verify the correctness of the preprocessing module.

4.System Testing:
Objective:
- Assess the functionality of the entire system.

Scope:
- Test the end-to-end process from image acquisition to disease detection.
- Validate system behavior under normal and abnormal conditions.

5.Performance Testing:

Objective:
- Evaluate the responsiveness, scalability, and resource usage of the system.

Scope:
- Test the system's ability to handle a high volume of image processing requests.
- Measure the response time under different load conditions.

6.Regression Testing:

Objective:
- Ensure that new changes do not negatively impact existing functionalities.

Scope:
- Perform regression tests after updates to the deep learning model or changes to the user interface.
- Confirm that new features or bug fixes do not introduce unintended side effects.

2.3 Test Completeness

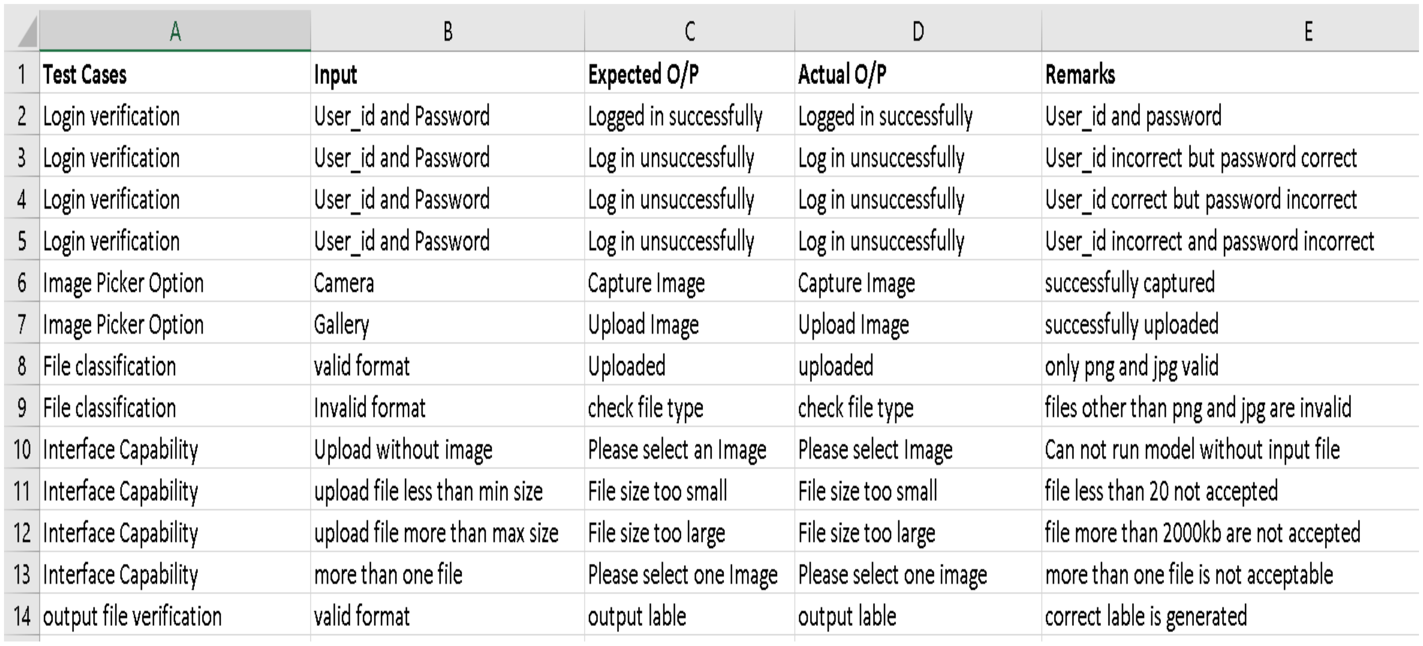
For instance, a few criteria to check Test Completeness would be

- 100% test coverage
- All Manual & Automated Test cases executed
- All open bugs are fixed or will be fixed in next release

3 Test Deliverables

Here mention all the Test Artifacts that will be delivered during different phases of the testing lifecycle.

3.1 Test Cases



	A	B	C	D	E
1	**Test Cases**	**Input**	**Expected O/P**	**Actual O/P**	**Remarks**
2	Login verification	User_id and Password	Logged in successfully	Logged in successfully	User_id and password
3	Login verification	User_id and Password	Log in unsuccessfully	Log in unsuccessfully	User_id incorrect but password correct
4	Login verification	User_id and Password	Log in unsuccessfully	Log in unsuccessfully	User_id correct but password incorrect
5	Login verification	User_id and Password	Log in unsuccessfully	Log in unsuccessfully	User_id incorrect and password incorrect
6	Image Picker Option	Camera	Capture Image	Capture Image	successfully captured
7	Image Picker Option	Gallery	Upload Image	Upload Image	successfully uploaded
8	File classification	valid format	Uploaded	uploaded	only png and jpg valid
9	File classification	Invalid format	check file type	check file type	files other than png and jpg are invalid
10	Interface Capability	Upload without image	Please select an Image	Please select Image	Can not run model without input file
11	Interface Capability	upload file less than min size	File size too small	File size too small	file less than 20 not accepted
12	Interface Capability	upload file more than max size	File size too large	File size too large	file more than 2000kb are not accepted
13	Interface Capability	more than one file	Please select one Image	Please select one image	more than one file is not acceptable
14	output file verification	valid format	output lable	output lable	correct lable is generated

Login Decision Table:

Conditions	Rule1	Rule2	Rule3	Rule4
UserID(T/F)	F	T	F	T
Password(T/F)	F	F	T	T
Output(E/H)	E	E	E	H

T – Correct username/password

F – Wrong username/password

E – Error message is displayed

H – Home screen is displayed

Case 1 – UserId and password both were wrong. The user is shown an error message.

Case 2 – UserId was correct, but the password was wrong. The user is shown an error message.

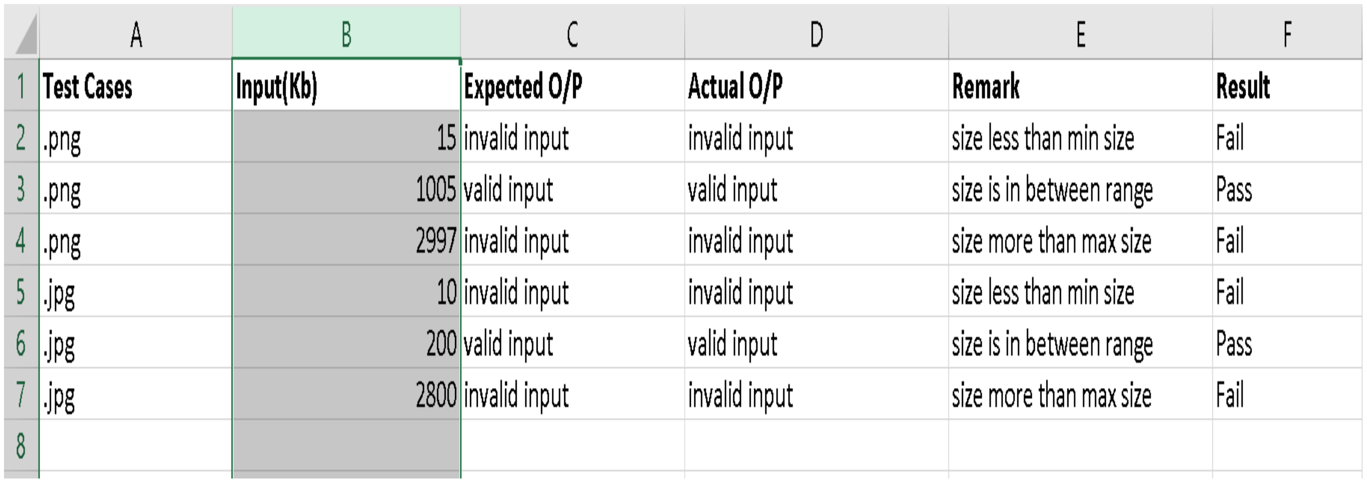
Case 3 – UserId was wrong, but the password was correct. The user is shown an error message.

Case 4 – Username and password both were correct, and the user navigated to the homepage.

Boundary Value analysis: Interface Capability

File size must be between 20kb to 2000kb

Invalid (min-1)	Valid (min,min+1,nominal,max-1,max)	Invalid (max+1)
19kb	20kb,21kb,1000kb,1999kb,2000kb	2001kb



	A	B	C	D	E	F
1	**Test Cases**	**Input(Kb)**	**Expected O/P**	**Actual O/P**	**Remark**	**Result**
2	.png	15	invalid input	invalid input	size less than min size	Fail
3	.png	1005	valid input	valid input	size is in between range	Pass
4	.png	2997	invalid input	invalid input	size more than max size	Fail
5	.jpg	10	invalid input	invalid input	size less than min size	Fail
6	.jpg	200	valid input	valid input	size is in between range	Pass
7	.jpg	2800	invalid input	invalid input	size more than max size	Fail
8						

4 Resource & Environment Needs

4.1 Testing Tools

A list of Tools like

● Selenium

● Postman API

4.2 Test Environment

Following **software’s** are required in addition to client-specific software.

● Windows 8 and above
● MS Excel
● VSCode
● Python
● Firebase

5 Terms/Acronyms

Make a mention of any terms or acronyms used in the project

TERM/ACRONYM	**DEFINITION**
API	Application Program Interface
AUT	Application Under Test