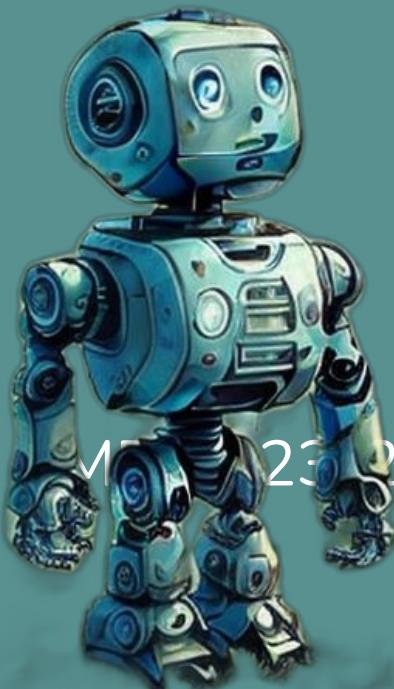


Advanced Plant Disease Analyzer: Automated Detection and Management System Quality Measurement - Vanilla



MAY 23 2024-105



Team Member

Supervisor: Dr. Sanika Wijayasekara

Co Supervisor: Ms. Tharushi Rubasinghe

Student Name

W.M.Janith chathuranga	IT20658854
Madurasingha M.A.C.A.	IT20619244
Gunawardhana O.W.	IT20263980
Herath H.M.R.K.	IT20665548

Presentation Outline

1. Overview of the Project
2. Development progress

Understanding the Research Area

Process of UI Design

Wireframe User Interface Design

Implimented Models

Technologies used

3. Future works
4. Individual Process

INTRODUCTION

Problem: Vanilla farmers need better tools

- ❑ Vanilla farming is becoming more complex due to new technologies.
- ❑ Growers struggle with diseases, inefficient techniques, and inconsistent quality.

Solution: Data visualization and decision support

- ❑ A research project called "Data Visualization and Decision Support" is underway.
- ❑ This project will provide real-time data and smart recommendations to farmers.

Benefits: Healthier plants, faster decisions, higher yields

- ❑ Farmers will have access to insights that improve plant health.
- ❑ Easier decision-making will lead to quicker responses to problems.
- ❑ The project aims to increase vanilla production through these improvements.

Background

High Disease Susceptibility: Vanilla cultivation is highly vulnerable to diseases, which can significantly affect both yield and quality.

Inefficiency of Traditional Methods: Current methods for disease detection and quality assessment are labor-intensive, time-consuming, and prone to human error, making them inefficient.

Need for Automation: An integrated automated system for early disease detection and accurate quality assessment would be invaluable for vanilla farmers, allowing timely interventions to reduce losses and enhance product quality.

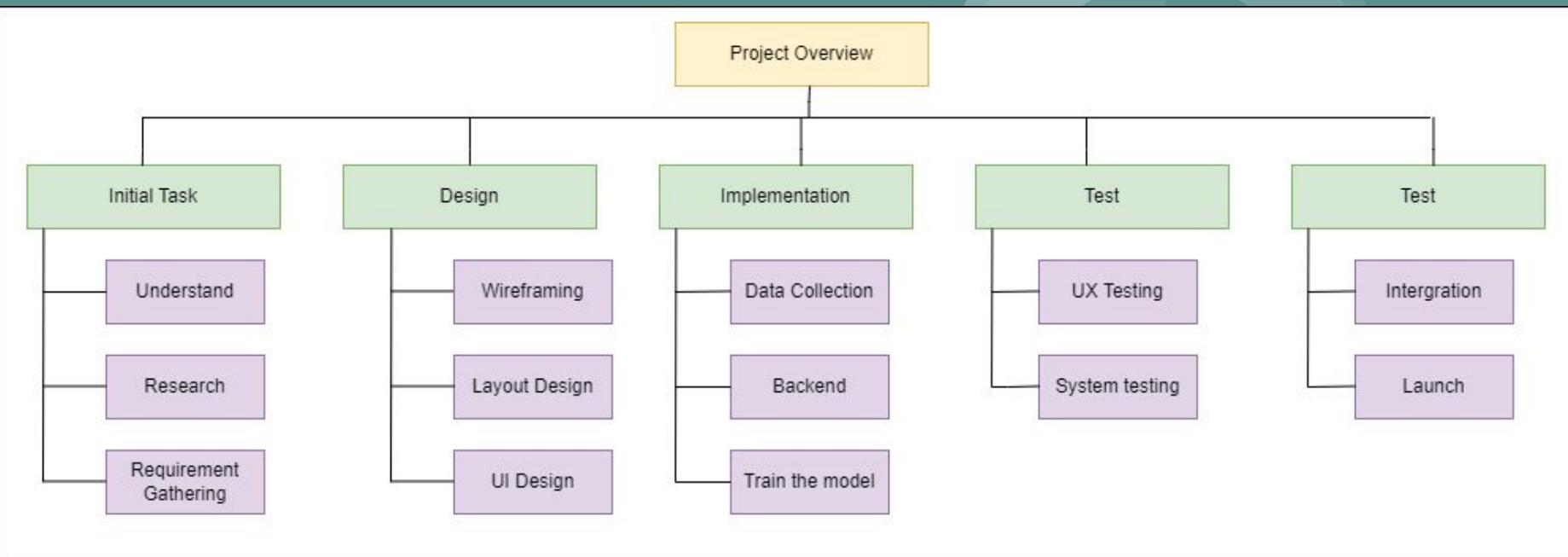
Research Question

How can image processing and machine learning be utilized to detect and classify diseases in vanilla plants with high accuracy?

What are the key attributes that determine vanilla pod quality, and how can these be measured automatically?

What are preferred data visualization formats and user communication channels?

Project Overview



Overall progress



Data Collection



Backend



Model Training



Front End



Software Quality Testing



Launch product



Understanding Research Area



Wireframing

Sign In Page

SIGNIN	Sign In	Sign Up
Name	<input type="text"/>	
Password	<input type="password"/>	
Forgot Password?		
Sign In		
To Home		

Sign Up Page

SIGNUP	Sign In	Sign Up
User Name	<input type="text"/>	
Role	<input type="text"/>	
Password	<input type="password"/>	
Comfirm Password	<input type="password"/>	
Sign Up		

Home Page

The image shows a user interface design. At the top left is a circular header containing the word "Logo". To its right is a dark grey horizontal bar with two white rectangular sections. On the far right of the header are two icons: a bell and a person. The main content area features a large grey box containing a 2x5 grid of small square icons with a black 'X' in the center. To the left of this main box is a vertical sidebar with a light brown background, displaying seven similar square icons with 'X's. On the far right, there is another grey box containing three horizontal grey bars.

Quality Inspection Section

The screenshot displays a user interface (UI) design application. On the left, there's a vertical toolbar with several icons: a circular 'Logo' icon at the top, followed by seven square icons with diagonal cross patterns of varying sizes. To the right of the toolbar is a dark grey rectangular area containing a white horizontal bar. Further right is a light grey rectangular section labeled 'Quality Inspection Section' with a 'Button' icon. Below this is a large rectangular box containing a smaller box with a large 'X' inside it. In the top right corner of the main window, there are two small square icons: one with a bell and another with a person profile.

Harvest Reporting

The screenshot shows a mobile application interface. At the top left is a circular "Logo" icon. To its right is a dark grey horizontal bar. On the far right are two icons: a bell and a user profile. The main content area has a light grey header with the text "Harvest Reporting". Below the header are two buttons: "Form" and "Chart". The "Form" button is currently active, indicated by a darker grey background. The "Chart" button is white with black text. A large rectangular input field contains three text entries: "Name", "Harvest (Weight/Kg)", and "Revenue (Rs.)", each in its own row. At the bottom of this field is a dark grey "Update" button. To the left of the main content area is a vertical sidebar with six icons arranged vertically, each consisting of a square with a diagonal line.

User Interface Design

SIGNUP

Sign In Sign Up

Username: XXXhackmail

Role: Administrator

Password: *****

Confirm Password: Confirm Password*

SIGN UP

SIGNIN

Sign In Sign Up

Name: XXXhackmail

Password: *****

Forgot Password?

SIGN IN

TO HOME

TODAY: 2024-3-15 28.9°C Clouds

Dashboard

GREEN MIND

Notice: Navigate using button panel.

Home Quality Inspection Disease Inspection Statistics IoT Panel

TODAY: 2024-3-15 27.7°C Clouds

Quality Inspection Section

Quality Inspection

Notice: Quality Information.

Upload or Drop Image HERE

Uploading

TODAY: 2024-3-15 27.7°C Clouds

Harvest Reporting

Form Chart

Notice: Stats Information.

Month: 01-2024

Harvest (Weight / Kg): Harvest from Kilograms*

Revenue (Rs.): Revenue*

Update

TODAY: 2024-3-15 27.7°C Clouds

Disease Statistics

10
Total Recorded Diseases
(Last 30 days)

13
Total Damaged Cases
(Last 30 days)

All Reported Cases

ID	Reported Date	Condition
2023-10-11_XN0X0T0dC0BzZs0oG1	2023-10-11	Fungal Disease
2023-10-17_J579pWk7M1pJZRtDs	2023-10-17	Fungal Disease
2023-11-17_ZFElabu9h875euf1Twm8	2023-11-17	Damaged
2023-11-20_KdhUuInUlj3qOmzZYb	2023-11-20	Damaged
2023-11-25_a9p12HfYNNzhYFrVyy2P	2023-11-25	Damaged

© 2024 Venka Quality Inspection Site.

TODAY: 2024-3-15 27.7°C Clouds

Disease Inpection

Upload or Drop Image HERE

© 2024 Venka Quality Inspection Site.

TODAY: 2024-3-15 27.7°C Clouds

Disease Chart

Number of Cases

Month	Number of Cases
10-2023	2
11-2023	4
1-2024	3
3-2024	14

© 2024 Venka Quality Inspection Site.

TODAY: 2024-3-15 27.7°C Clouds

IOT Control Panel

Manual Guided

Temperature 31 °C
Humidity 83

NPK 17-17-17 NPK 20-20-20 NPK 30-10-10 Water Readings

Water (Main)
NPK 17-17-17
NPK 20-20-20

© 2024 Venka Quality Inspection Site.

**W.M.Janith chathuranga
IT20658854**

BSC (HONS) DEGREE IN INFORMATION TECHNOLOGY (SPECIALIZATION IN
INFORMATION TECHNOLOGY)



Data Visualization and Decision Support

Background

Vanilla Cultivation Challenges:

- Disease Risks
- Complex Cultivation
- Stringent Quality Demands

These challenges limit real-time insights for farmers, hindering informed decision-making.

Our Solution:

"Data Visualization & Decision Support" empowers farmers with:

- Detailed Harvest & Crop Data (Tables & Graphs)
- AI-Powered Predictions for informed cultivation choices.

Research Problem

Challenges:

- ❑ Intuitive Interface: Catering to diverse tech skills of vanilla farmers (beginner to advanced).
- ❑ Data Integration: Seamlessly combining data from various sources (Plant Disease Analyzer, Growing/Fertilization) for accurate, real-time insights.
- ❑ AI Prediction Accuracy: Enhancing speed and precision of machine learning predictions for informed decision-making.

Main Objective:

- ❑ Empower Vanilla Farmers with Real-Time Insights

Through:

- ❑ Data Visualization Techniques (Tables, Graphs)
- ❑ Intelligent Decision Support Systems (AI Predictions)

Outcomes:

- ❑ Informed Decision-Making
- ❑ Optimized Cultivation Practices
- ❑ Increased Productivity & Success

Sub-Objectives:

Machine Learning Integration:

- Analyze data from fault detection, condition measurement, and disease detection.
- Seamlessly integrate AI predictions for real-time insights.

Data Visualization Design:

- Visually appealing representations of harvest and crop data.

Data Flow Optimization:

- Ensure smooth data exchange between components.

Requirements

Functional Requirements

- The system must feature an intuitive and user-friendly interface to ensure easy accessibility for vanilla farmers.
- Incorporate machine learning algorithms to process and analyze data, enabling the generation of real-time predictions for farmers.
- Use of tables and graphs.

Non - Functional Requirements

- The interface shall adhere to accessibility standards (e.g., WCAG 2.0), ensuring that it is usable by individuals with disabilities.
- The codebase and design of the interface should adhere to best practices, facilitating ease of maintenance and future enhancements.

Technologies Used



Gunawardhana O.W.

IT20263980



BSC (HONS) DEGREE IN INFORMATION TECHNOLOGY (SPECIALIZATION IN
INFORMATION TECHNOLOGY)

Quantitative Quality Measurement

Background

- "Quantitative Quality Measurement" is indispensable in smart vanilla cultivation, akin to the significance of disease management in crop health.
- Similar to disease detection, timely intervention based on quantitative quality measurements is vital for ensuring optimal vanilla pod quality.
- Vanilla pod quality assessment holds immense importance for market value and desirability.
- Accurate measurements and standardization of attributes such as size, color intensity, and surface texture are pivotal for determining vanilla pod quality.

Research Problem

- Ensuring the accuracy and precision of automated measurements for attributes like size, color intensity, and surface texture in vanilla pods.
- Developing methods to effectively integrate multiple attributes into a comprehensive quality assessment framework for vanilla pods.
- Addressing variability in vanilla pod attributes due to environmental conditions, genetic variation, and post-harvest factors to improve measurement reliability.

Main Objectives

- ❑ Establish standardized protocols for consistent measurement of vanilla pod attributes across different environments and cultivars.
- ❑ Develop automated systems for precise measurement of vanilla pod attributes like size, color intensity, and surface texture.
- ❑ Validate and calibrate measurement techniques against established standards to ensure accuracy in assessing vanilla pod quality.

Sub Objectives

- ❑ Conduct comparative analyses of vanilla pods assessed with traditional methods and the quantitative measurement system.
- ❑ Identify relevant environmental parameters affecting quality attributes (e.g., humidity, light).
- ❑ Collect a diverse dataset of vanilla plant images for algorithm training and validation.

Requirements

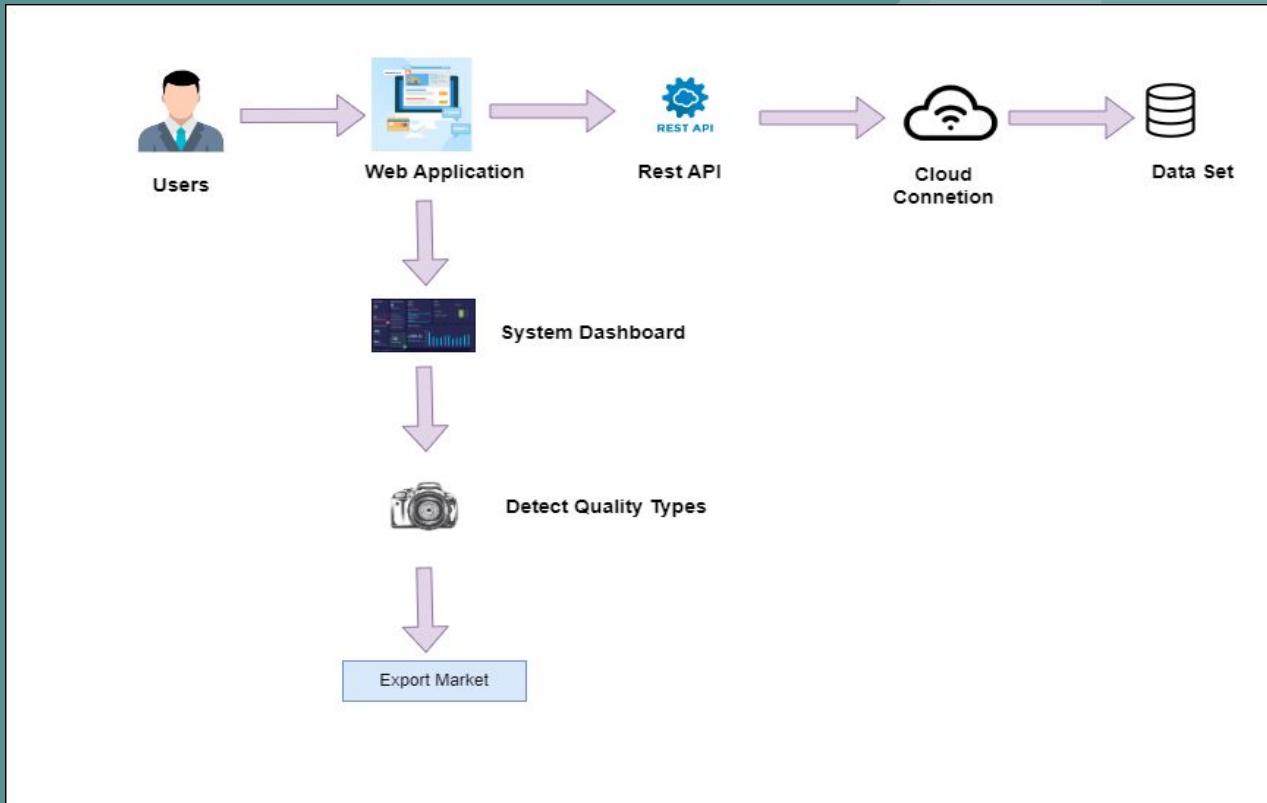
Functional Requirements

- Standardized Measurements:
- Image Processing Algorithms:
- Cultivar Adaptability:

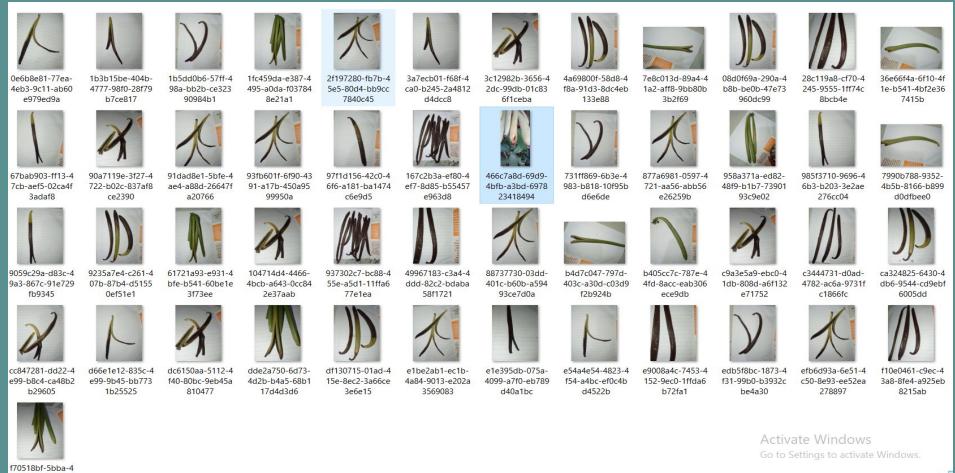
Non - Functional Requirements

- Accuracy and Precision:
- Usability and User-Friendly Interface:
- Compatibility and Integration:

System Overview Diagram

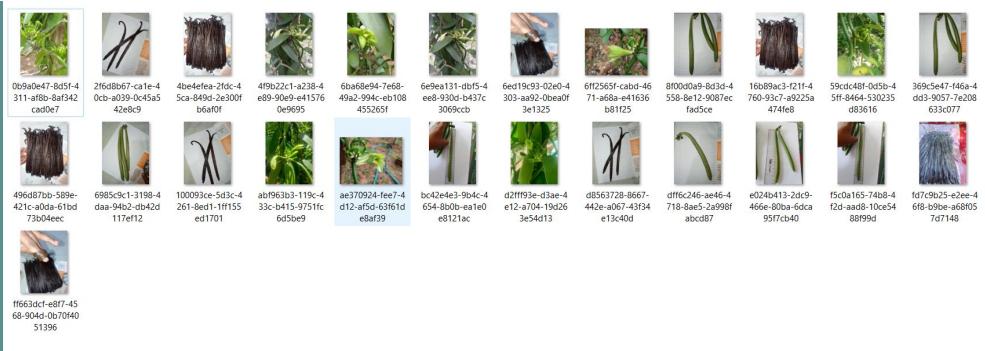


DataSet

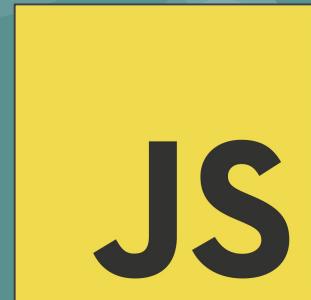
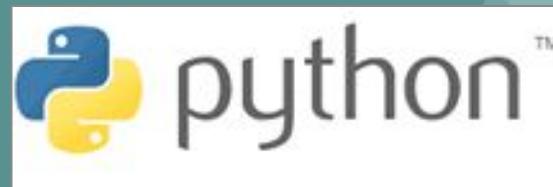


Activate Windows

Go to Settings to activate Windows.



Technologies Used



**Madurasingha M.A.C.A.
IT20619244**

**BSC (HONS) DEGREE IN INFORMATION TECHNOLOGY
(SPECIALIZATION IN INFORMATION TECHNOLOGY)**



Plant Disease Analyzer



Background

- ❖ Vanilla, a crucial export crop in Sri Lanka, faces significant challenges due to its susceptibility to various diseases, posing a significant crisis for farmers, researchers, and exporters.
- ❖ The "Plant Disease Analyzer" component uses to assess the health of vanilla plants, provide information on disease symptoms, and suggest appropriate remedies.
- ❖ By using this component, various diseases occurring in vanilla plants can be detected quickly and the necessary remedies can be obtained efficiently and accurately, so the impact on vanilla cultivation can be minimized.

Research Problem

- ❖ How to identify the various diseases of vanilla plants and how to efficiently and accurately give the details of the necessary remedies ?



Objectives

Main Objectives -

- ★ To develop a vanilla plant disease analyzer component that can accurately and reliably identify vanilla plant diseases based on images of affected Vanilla plant parts.

Sub Objectives -

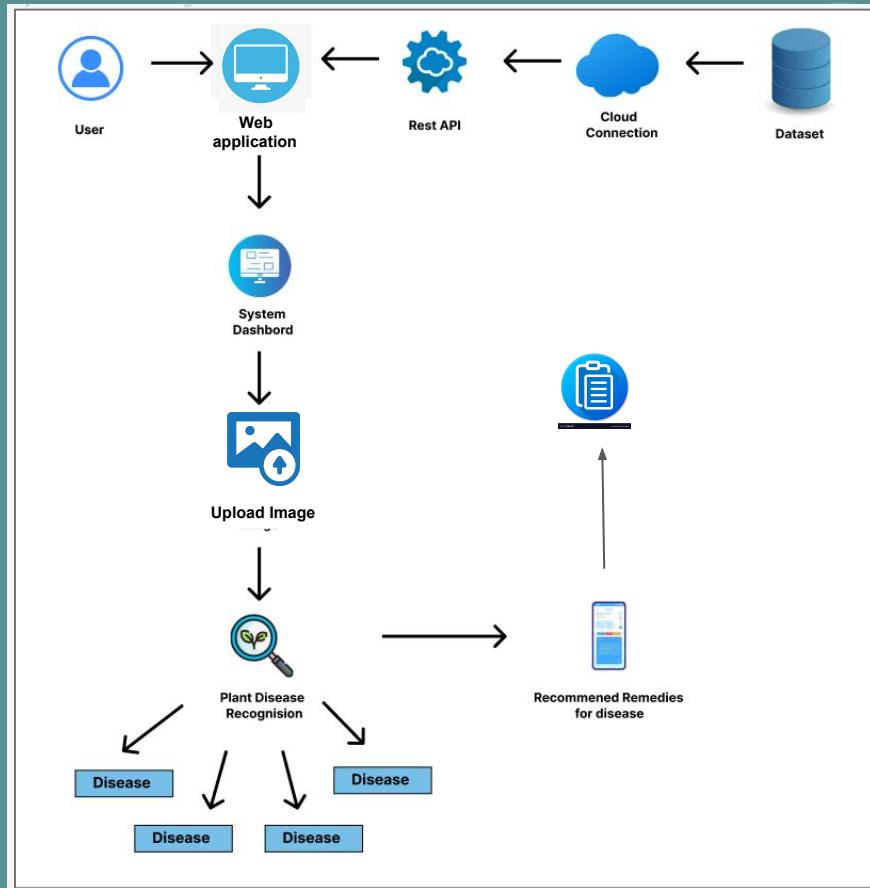
- ★ To collect and annotate a dataset of vanilla plant disease images.
- ★ To develop algorithms that can robustly handle variability in vanilla plant images.
- ★ To develop a user-friendly interface that makes it easy for users to use the web application.

Requirements

- ★ Performance and Efficiency
- ★ Scalability
- ★ Security and Data Privacy
- ★ Reliability and Availability
- ★ Usability and Accessibility

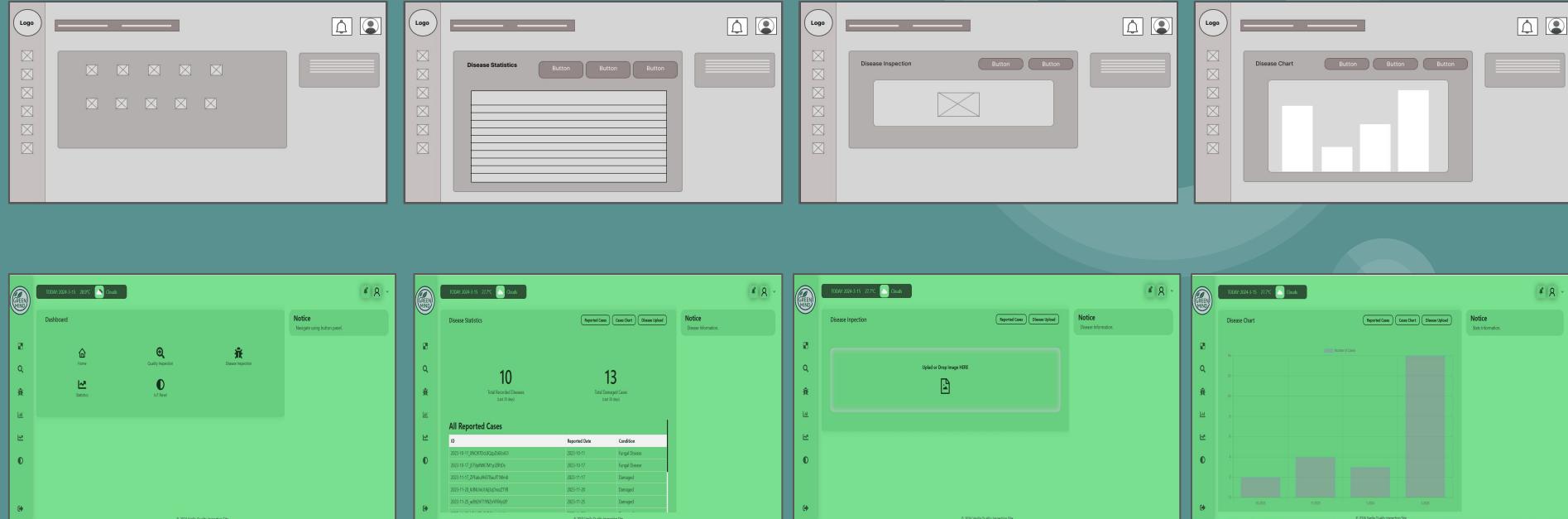


System Overview Diagram



Progress up to now

□ Wireframe and UI designing





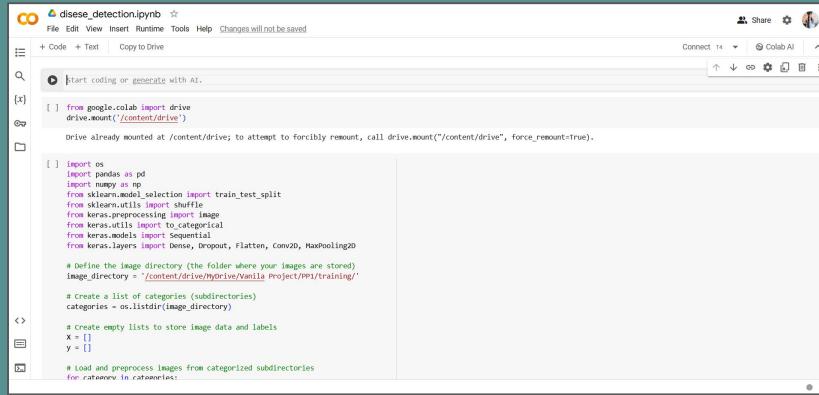
Diseases and Damage Vanilla



Healthy pods and flowers



Model Training - Google Co-Lab was used to train the model



```
[ ] from google.colab import drive
drive.mount('/content/drive')

# Define already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

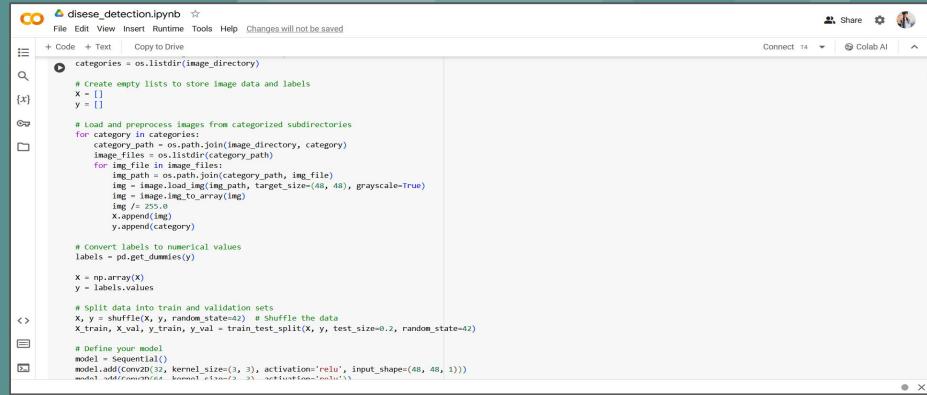
[ ] import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from tensorflow.keras import image
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

# Define the image directory (the folder where your images are stored)
image_directory = '/content/drive/MyDrive/Vaishu Project/P91/training/'

# Create a list of categories (subdirectories)
categories = os.listdir(image_directory)

# Create empty lists to store image data and labels
x = []
y = []

# Load and preprocess images from categorized subdirectories
for category in categories:
```



```
[ ] categories = os.listdir(image_directory)

# Create empty lists to store image data and labels
x = []
y = []

# Load and preprocess images from categorized subdirectories
for category in categories:
    category_path = os.path.join(image_directory, category)
    image_files = os.listdir(category_path)
    for img_file in image_files:
        img_path = os.path.join(category_path, img_file)
        img = image.load_img(img_path, target_size=(48, 48), grayscale=True)
        img = np.array(img)
        img /= 255.0
        x.append(img)
        y.append(category)

# Convert labels to numerical values
labels = pd.get_dummies(y)

X = np.array(x)
y = labels.values

# Split data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

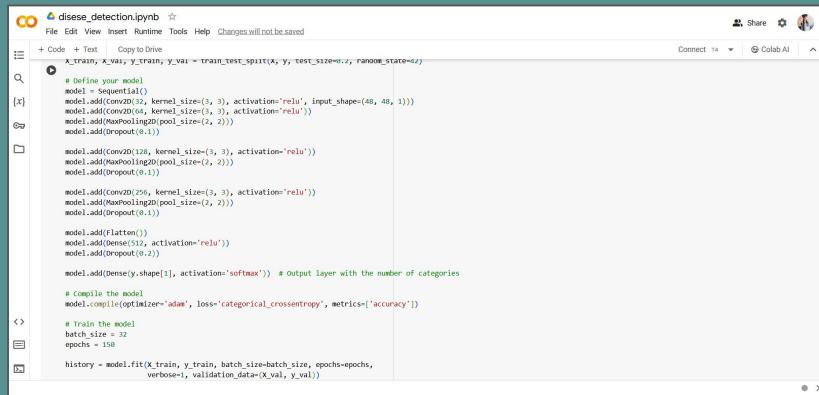
# Define your model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.1))

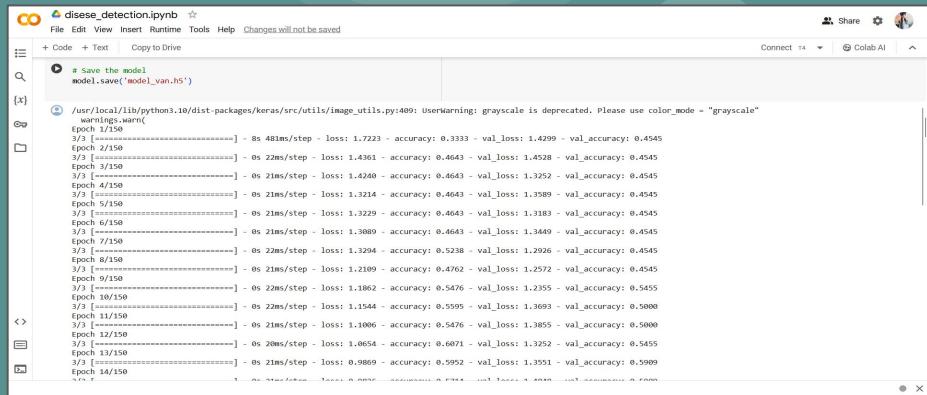
model.add(Dense(len(categories), activation='softmax')) # Output layer with the number of categories
```



```
[ ] # Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
batch_size = 32
epochs = 150

history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                     verbose=1, validation_data=(X_val, y_val))
```



```
[ ] # Save the model
model.save('model_van.h5')

[ ] Epoch 1/150
3/3 [=====] - 0s 48ms/step - loss: 1.7223 - accuracy: 0.3333 - val_loss: 1.4299 - val_accuracy: 0.4545
Epoch 2/150
3/3 [=====] - 0s 22ms/step - loss: 1.4361 - accuracy: 0.4643 - val_loss: 1.4528 - val_accuracy: 0.4545
Epoch 3/150
3/3 [=====] - 0s 22ms/step - loss: 1.4240 - accuracy: 0.4663 - val_loss: 1.3929 - val_accuracy: 0.4545
Epoch 4/150
3/3 [=====] - 0s 21ms/step - loss: 1.3214 - accuracy: 0.4661 - val_loss: 1.3589 - val_accuracy: 0.4545
Epoch 5/150
3/3 [=====] - 0s 21ms/step - loss: 1.3229 - accuracy: 0.4643 - val_loss: 1.3183 - val_accuracy: 0.4545
Epoch 6/150
3/3 [=====] - 0s 21ms/step - loss: 1.3089 - accuracy: 0.4643 - val_loss: 1.3440 - val_accuracy: 0.4545
Epoch 7/150
3/3 [=====] - 0s 22ms/step - loss: 1.3294 - accuracy: 0.5238 - val_loss: 1.2926 - val_accuracy: 0.4545
Epoch 8/150
3/3 [=====] - 0s 21ms/step - loss: 1.2189 - accuracy: 0.4762 - val_loss: 1.2572 - val_accuracy: 0.4545
Epoch 9/150
3/3 [=====] - 0s 22ms/step - loss: 1.1862 - accuracy: 0.5476 - val_loss: 1.2355 - val_accuracy: 0.5455
Epoch 10/150
3/3 [=====] - 0s 21ms/step - loss: 1.1544 - accuracy: 0.5595 - val_loss: 1.3693 - val_accuracy: 0.5900
Epoch 11/150
3/3 [=====] - 0s 21ms/step - loss: 1.1006 - accuracy: 0.5476 - val_loss: 1.3895 - val_accuracy: 0.5000
Epoch 12/150
3/3 [=====] - 0s 20ms/step - loss: 1.0654 - accuracy: 0.6071 - val_loss: 1.3295 - val_accuracy: 0.5455
Epoch 13/150
3/3 [=====] - 0s 21ms/step - loss: 0.9869 - accuracy: 0.5952 - val_loss: 1.3551 - val_accuracy: 0.5909
Epoch 14/150
```

Model Training

```
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text Copy to Drive
[x] # Save the model
model.save("model_vanish")
(x) Epoch 19/150
[1/1 [=====]] - 0s 21ms/step - loss: 0.0156 - accuracy: 0.7598 - val_loss: 1.8864 - val_accuracy: 0.5000
Epoch 20/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.5719 - accuracy: 0.7857 - val_loss: 1.8689 - val_accuracy: 0.6364
Epoch 21/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.4732 - accuracy: 0.8214 - val_loss: 2.0555 - val_accuracy: 0.5909
[3/3 [=====]] - 0s 21ms/step - loss: 0.4075 - accuracy: 0.8452 - val_loss: 2.0308 - val_accuracy: 0.5989
Epoch 22/150
[1/1 [=====]] - 0s 21ms/step - loss: 0.3280 - accuracy: 0.8029 - val_loss: 2.5176 - val_accuracy: 0.5455
Epoch 24/150
[1/1 [=====]] - 0s 21ms/step - loss: 0.3093 - accuracy: 0.9167 - val_loss: 2.2693 - val_accuracy: 0.5455
Epoch 25/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.2030 - accuracy: 0.9524 - val_loss: 3.4512 - val_accuracy: 0.5988
[3/3 [=====]] - 0s 20ms/step - loss: 0.1665 - accuracy: 0.9846 - val_loss: 3.3939 - val_accuracy: 0.5455
Epoch 26/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.2850 - accuracy: 0.9643 - val_loss: 2.0495 - val_accuracy: 0.6364
Epoch 27/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.1458 - accuracy: 0.9805 - val_loss: 2.0973 - val_accuracy: 0.5989
Epoch 28/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.1458 - accuracy: 0.9805 - val_loss: 2.0973 - val_accuracy: 0.5989
[3/3 [=====]] - 0s 20ms/step - loss: 0.1007 - accuracy: 0.9762 - val_loss: 3.6885 - val_accuracy: 0.5989
Epoch 29/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.0797 - accuracy: 0.9762 - val_loss: 3.7831 - val_accuracy: 0.5989
Epoch 30/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.0456 - accuracy: 1.0000 - val_loss: 3.6333 - val_accuracy: 0.5989
Epoch 31/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.0250 - accuracy: 1.0000 - val_loss: 3.8173 - val_accuracy: 0.5989
Epoch 32/150
[1/1 [=====]] - 0s 20ms/step - loss: 0.0210 - accuracy: 1.0000 - val_loss: 4.2395 - val_accuracy: 0.5989
[3/3 [=====]] - 0s 21ms/step - loss: 0.0210 - accuracy: 1.0000 - val_loss: 4.2395 - val_accuracy: 0.5989
```

```
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text Copy to Drive
[x] # Save the model
model.save("model_vanish")
(x) Epoch 146/150
[1/1 [=====]] - 0s 23ms/step - loss: 4.9556e-05 - accuracy: 1.0000 - val_loss: 8.1513 - val_accuracy: 0.5989
[3/3 [=====]] - 0s 22ms/step - loss: 6.3730e-05 - accuracy: 1.0000 - val_loss: 8.1657 - val_accuracy: 0.5989
[1/1 [=====]] - 0s 37ms/step - loss: 6.3735e-05 - accuracy: 1.0000 - val_loss: 8.1663 - val_accuracy: 0.5989
Epoch 148/150
[1/1 [=====]] - 0s 38ms/step - loss: 9.4124e-05 - accuracy: 1.0000 - val_loss: 8.1337 - val_accuracy: 0.5989
[3/3 [=====]] - 0s 35ms/step - loss: 1.1270e-04 - accuracy: 1.0000 - val_loss: 8.0820 - val_accuracy: 0.5989
Epoch 149/150
[1/1 [=====]] - 0s 31ms/step - loss: 7.6352e-05 - accuracy: 1.0000 - val_loss: 8.0197 - val_accuracy: 0.5989
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy and will be removed in a future version.
  saving_api.save_model()
(x) import matplotlib.pyplot as plt
import random
import numpy as np

# Data visualize
sample_size = 16
num_cols = 4
num_rows = sample_size // num_cols

random_indices = random.sample(range(len(y_val)), sample_size)
sample_images = x_val[random_indices]
sample_labels_actual = y_val[random_indices]
sample_labels_predicted = model.predict(sample_images)

# Classes
class_names = ["Bean Rot", "Black Crust", "Damaged", "Fungal Disease", "Healthy", "Mosaic Virus", "Root Rot"]

plt.figure(figsize=(12, 10))

for i in range(min(sample_size, num_rows * num_cols)):
    plt.imshow(sample_images[i].reshape(40, 40), cmap="gray")
    actual_class_index = np.argmax(sample_labels.actual[i])
    predicted_class_index = np.argmax(sample_labels.predicted[i])
    plt.title(f"Actual: {class_names[actual_class_index]}\nPredicted: {class_names[predicted_class_index]}")
    plt.axis('off')
    plt.show()
```

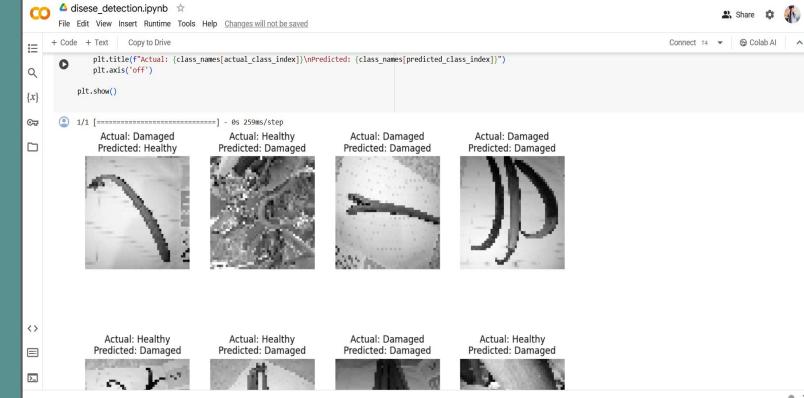
```
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text Copy to Drive
[x] # Save the model
model.save("model_vanish")
(x) # Import matplotlib.pyplot as plt
import random
import numpy as np

# Data visualize
sample_size = 16
num_cols = 4
num_rows = sample_size // num_cols

random_indices = random.sample(range(len(y_val)), sample_size)
sample_images = x_val[random_indices]
sample_labels_actual = y_val[random_indices]
sample_labels_predicted = model.predict(sample_images)

# Classes
class_names = ["Bean Rot", "Black Crust", "Damaged", "Fungal Disease", "Healthy", "Mosaic Virus", "Root Rot"]

plt.figure(figsize=(12, 10))
for i in range(min(sample_size, num_rows * num_cols)):
    plt.imshow(sample_images[i].reshape(40, 40), cmap="gray")
    actual_class_index = np.argmax(sample_labels.actual[i])
    predicted_class_index = np.argmax(sample_labels.predicted[i])
    plt.title(f"Actual: {class_names[actual_class_index]}\nPredicted: {class_names[predicted_class_index]}")
    plt.axis('off')
    plt.show()
```



Model Training

The screenshot shows a Jupyter Notebook interface with the title "disease_detection.ipynb". The menu bar includes File, Edit, View, Insert, Run cell, Tools, Help, and a "Help" link. A toolbar with icons for code, test, copy, and drive is visible. On the right, there are "Connect" and "Colab AI" buttons. Below the toolbar, four images of bean plant stems are displayed, each with a label below it:

- Image 1: Actual: Healthy, Predicted: Fungal Disease
- Image 2: Actual: Bean Rot, Predicted: Fungal Disease
- Image 3: Actual: Damaged, Predicted: Damaged
- Image 4: Actual: Healthy, Predicted: Healthy

The images show various stages of plant health, from healthy green stems to ones with brown spots or damage.



The screenshot shows a Google Colab notebook titled "disease_detection.pyrb". The code imports various libraries including pandas, numpy, and keras. It defines a function `get_data` that loads images from a specified directory, processes them, and stores them in lists for training and testing. The code then creates a list of categories and loads images from categorized subdirectories. A sample output is shown where the first image is labeled as 'No Disease'.

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from tensorflow import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Input, Flatten, Conv2D, MaxPooling2D
from keras.callbacks import ModelCheckpoint
from keras.preprocessing import image
from keras.applications import VGG16

# Define the image directory (the folder where your images are stored)
image_directory = "/content/drive/MyDrive/Project/1991/training"

# Create a list of categories (subdirectories)
categories = os.listdir(image_directory)

# Create empty lists to store image data and labels
X = []
Y = []

# Load and process images from categorized subdirectories
for category in categories:
    category_path = os.path.join(image_directory, category)
    image_files = os.listdir(category_path)
    for img_file in image_files:
        img_path = os.path.join(category_path, img_file)
        img = image.load_img(img_path, target_size=(150, 150))
        img_array = np.array(img)
        X.append(img_array)
        if category == 'No Disease':
            Y.append(0)
        else:
            Y.append(1)

# Convert lists to arrays
X = np.array(X)
Y = np.array(Y)

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

A screenshot of a Jupyter Notebook interface. The title bar says 'File Edit View Insert Runtime Tools Help Changes will not be saved Connect Colab AI'. The notebook contains a code cell and a text cell. Below these are eight image thumbnails arranged in two rows of four. Each thumbnail shows a close-up of a plant stem or leaf with a caption below it. The first row includes: 'Actual: Damaged Predicted: Damaged' (image of a damaged stem), 'Actual: Damaged Predicted: Damaged' (image of a damaged stem), 'Actual: Healthy Predicted: Damaged' (image of a healthy stem with a tool nearby), and 'Actual: Healthy Predicted: Healthy' (image of a healthy stem). The second row includes: 'Actual: Mosaic Virus Predicted: Fungal Disease' (image of a leaf with mosaic virus symptoms), 'Actual: Bean Rust Predicted: Fungal Disease' (image of a leaf with bean rust symptoms), 'Actual: Damaged Predicted: Damaged' (image of a damaged stem), and 'Actual: Healthy Predicted: Healthy' (image of a healthy stem).

VS Codes

This screenshot shows the Frontend project structure in the Explorer sidebar. The main file is `main.py`, which imports FastAPI, File, UploadFile, and FileResponse from fastapi, and CORSMiddleware from fastapi.middleware.cors. It uses pathlib to import Path, uvicorn to run the app, numpy for arrays, BytesIO from io, Image from PIL, tensorflow from tensorflow, and JSONResponse from fastapi.responses. The app is defined as a FastAPI instance with origins set to `["http://localhost:3000", "http://localhost:3001"]`. The `app.add_middleware` method is used to add CORS middleware. The terminal shows the command `uvicorn main:app --host 0.0.0.0 --port 8000` being run.

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import FileResponse
from fastapi.middleware.cors import CORSMiddleware
from pathlib import Path
import uvicorn
import numpy as np
from io import BytesIO
from PIL import Image
import tensorflow as tf
from fastapi.responses import JSONResponse
import uv3
import temfile
import joblib
from pydantic import BaseModel
from spec_quality import measure_objects

app = FastAPI()
origins = [
    "http://localhost:3000",
    "http://localhost:3001"
]
app.add_middleware(CORSMiddleware, allow_origins=origins)
```

Frontend

This screenshot shows the Frontend project structure in the Explorer sidebar. The main file is `main.py`, which imports FastAPI, File, UploadFile, and FileResponse from fastapi, and CORSMiddleware from fastapi.middleware.cors. It uses pathlib to import Path, uvicorn to run the app, numpy for arrays, BytesIO from io, Image from PIL, tensorflow from tensorflow, and JSONResponse from fastapi.responses. The app is defined as a FastAPI instance with origins set to `["http://localhost:3000", "http://localhost:3001"]`. The `app.add_middleware` method is used to add CORS middleware. The terminal shows the command `uvicorn main:app --host 0.0.0.0 --port 8000` being run.

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import FileResponse
from fastapi.middleware.cors import CORSMiddleware
from pathlib import Path
import uvicorn
import numpy as np
from io import BytesIO
from PIL import Image
import tensorflow as tf
from fastapi.responses import JSONResponse
import uv3
import temfile
import joblib
from pydantic import BaseModel
from spec_quality import measure_objects

app = FastAPI()
origins = [
    "http://localhost:3000",
    "http://localhost:3001"
]
app.add_middleware(CORSMiddleware, allow_origins=origins)
```

Backend

This screenshot shows the Backend project structure in the Explorer sidebar. The main file is `main.py`, which imports FastAPI, File, UploadFile, and FileResponse from fastapi, and CORSMiddleware from fastapi.middleware.cors. It uses pathlib to import Path, uvicorn to run the app, numpy for arrays, BytesIO from io, Image from PIL, tensorflow from tensorflow, and JSONResponse from fastapi.responses. The app is defined as a FastAPI instance with origins set to `["http://localhost:3000", "http://localhost:3001"]`. The `app.add_middleware` method is used to add CORS middleware. The terminal shows the command `uvicorn main:app --host 0.0.0.0 --port 8000` being run.

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import FileResponse
from fastapi.middleware.cors import CORSMiddleware
from pathlib import Path
import uvicorn
import numpy as np
from io import BytesIO
from PIL import Image
import tensorflow as tf
from fastapi.responses import JSONResponse
import uv3
import temfile
import joblib
from pydantic import BaseModel
from spec_quality import measure_objects

app = FastAPI()
origins = [
    "http://localhost:3000",
    "http://localhost:3001"
]
app.add_middleware(CORSMiddleware, allow_origins=origins)
```

ml-api

This screenshot shows the ml-api project structure in the Explorer sidebar. The main file is `main.py`, which imports FastAPI, File, UploadFile, and FileResponse from fastapi, and CORSMiddleware from fastapi.middleware.cors. It uses pathlib to import Path, uvicorn to run the app, numpy for arrays, BytesIO from io, Image from PIL, tensorflow from tensorflow, and JSONResponse from fastapi.responses. The app is defined as a FastAPI instance with origins set to `["http://localhost:3000", "http://localhost:3001"]`. The `app.add_middleware` method is used to add CORS middleware. The terminal shows the command `uvicorn main:app --host 0.0.0.0 --port 8000` being run.

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import FileResponse
from fastapi.middleware.cors import CORSMiddleware
from pathlib import Path
import uvicorn
import numpy as np
from io import BytesIO
from PIL import Image
import tensorflow as tf
from fastapi.responses import JSONResponse
import uv3
import temfile
import joblib
from pydantic import BaseModel
from spec_quality import measure_objects

app = FastAPI()
origins = [
    "http://localhost:3000",
    "http://localhost:3001"
]
app.add_middleware(CORSMiddleware, allow_origins=origins)
```

Technologies Used



Herath H.M.R.K.
IT20665548



BSC (HONS) DEGREE IN INFORMATION TECHNOLOGY (SPECIALIZATION IN
INFORMATION TECHNOLOGY)

Plant Growing and Fertilization

Background

- **Environmental Interactions:** Achieving optimal plant development and robust yields depends on understanding how environmental factors like light, water, temperature, and soil quality interact.
- **Prudent Fertilization:** Researchers emphasize the importance of judicious fertilizer use to maximize plant growth while minimizing environmental impact.
- **Customized Tactics:** By tailoring fertilization tactics to the unique needs of each plant species, researchers promote sustainable practices that protect crop health and ecosystem integrity.

Research Problem

- ❑ Identify the key indicators of vanilla quality. What are the physical, chemical, and sensory properties that contribute to the quality of vanilla?
- ❑ Develop methods for quantitatively measuring these indicators. This could involve using spectroscopy, chromatography, or other analytical techniques.
- ❑ Validate the accuracy and reliability of these methods. This could be done by comparing the results of the methods to those of human experts.

Main Objectives

- ❑ Create advanced image processing algorithms to extract relevant features from vanilla plant images, including size, color intensity, and surface texture.
- ❑ Promote healthy plant development by providing essential nutrients, water, and ideal environmental conditions.
- ❑ Enhance crop yield and quality, focusing on factors like size, flavor, and nutritional content.
- ❑ Implement sustainable practices to reduce the use of synthetic fertilizers, conserve water, and protect soil health.

Sub Objectives

- ❑ Conduct comparative analyses of vanilla pods assessed with traditional methods and the quantitative measurement system.
- ❑ Identify relevant environmental parameters affecting quality attributes (e.g., humidity, light).
- ❑ Collect a diverse dataset of vanilla plant images for algorithm training and validation.

Requirements

Functional Requirements

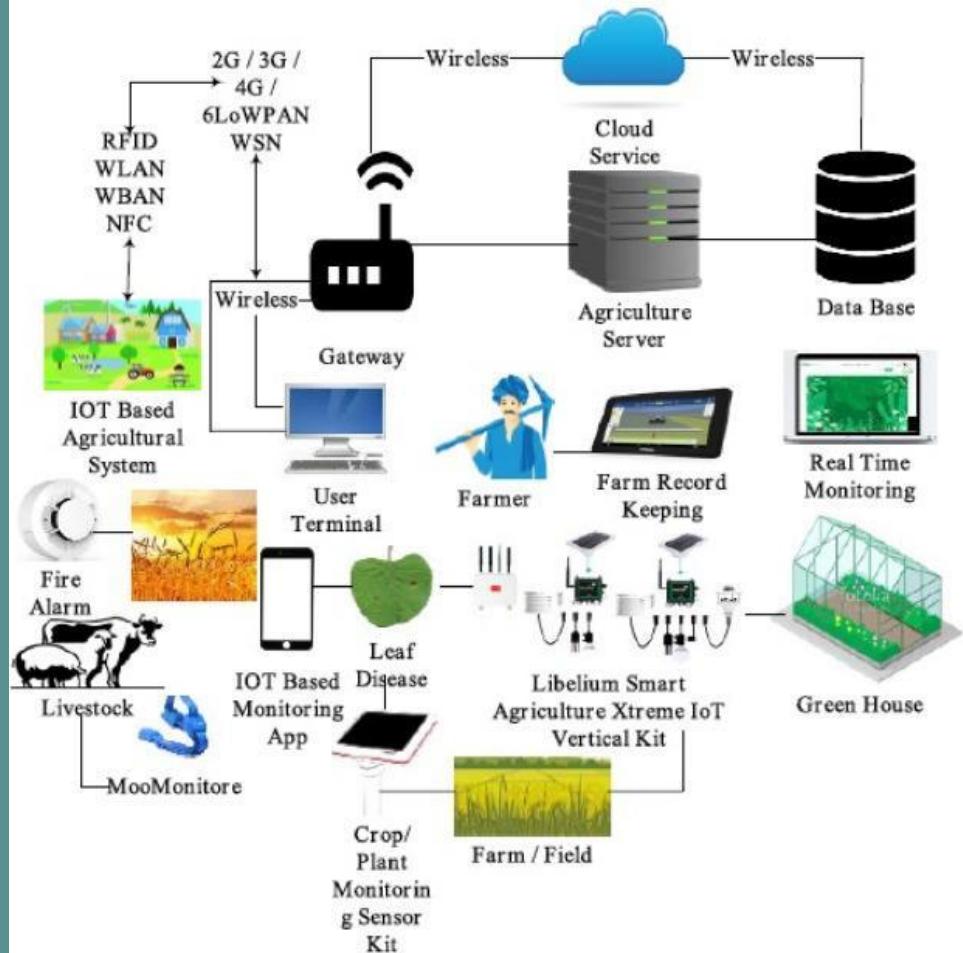
- Using IoT Devices
- Calculation of usage history
- Soil Moisture Monitoring
- Automated Watering System
- Fertilization Management
- Environmental Monitoring
- Predictive Analytics

Requirements

Non - Functional Requirements

- Reliability
- Scalability
- Usability
- Performance
- Security

System Overview Diagram



Technologies Used





THANK YOU
