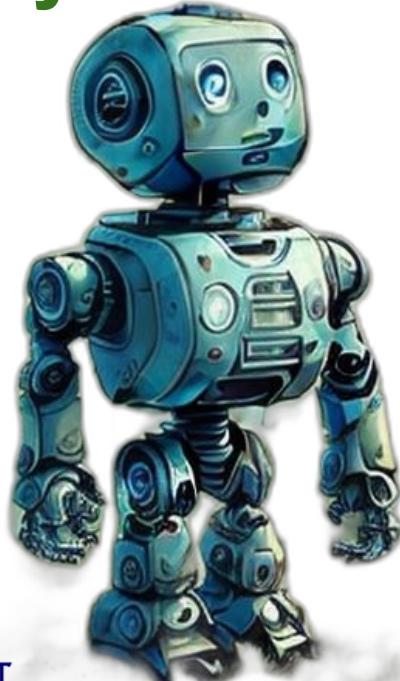


Advanced Plant Disease Analyzer: Automated Detection and Management System Quality Measurement - Vanilla



TMP-2023-24-105



Team Member

Supervisor: Dr. Sanika Wijayasekara

Co Supervisor: Ms. Tharushi Rubasinghe

Student Name

W.M.Janith chathuranga

Madurasingha M.A.C.A.

Gunawardhana O.W.

Herath H.M.R.K.

Student ID

IT20658854

IT20619244

IT20263980

IT20665548

Presentation Outline

- 1. Overview of the Project**
- 2. Development progress**

Understanding the Research Area

Process of UI Design

Wireframe

User Interface Design

Implemented Models

Technologies used

- 3. Future works**

- 4. Individual Process**

5.

INTRODUCTION

The cultivation of crops, including vanilla, is constantly evolving with the integration of advanced technologies. The success of growers and exporters in the vanilla industry depends on mitigating disease risks, refining growing techniques, and guaranteeing high-quality outputs. "Data Visualization and Decision Support," the research component, aims to provide vanilla farmers with real-time insights and data-driven assistance by utilizing data visualization techniques and intelligent decision support systems. This part seeks to improve plant health, expedite decision-making, and increase production in the vanilla sector by offering an easy-to-use interface and customized advice.



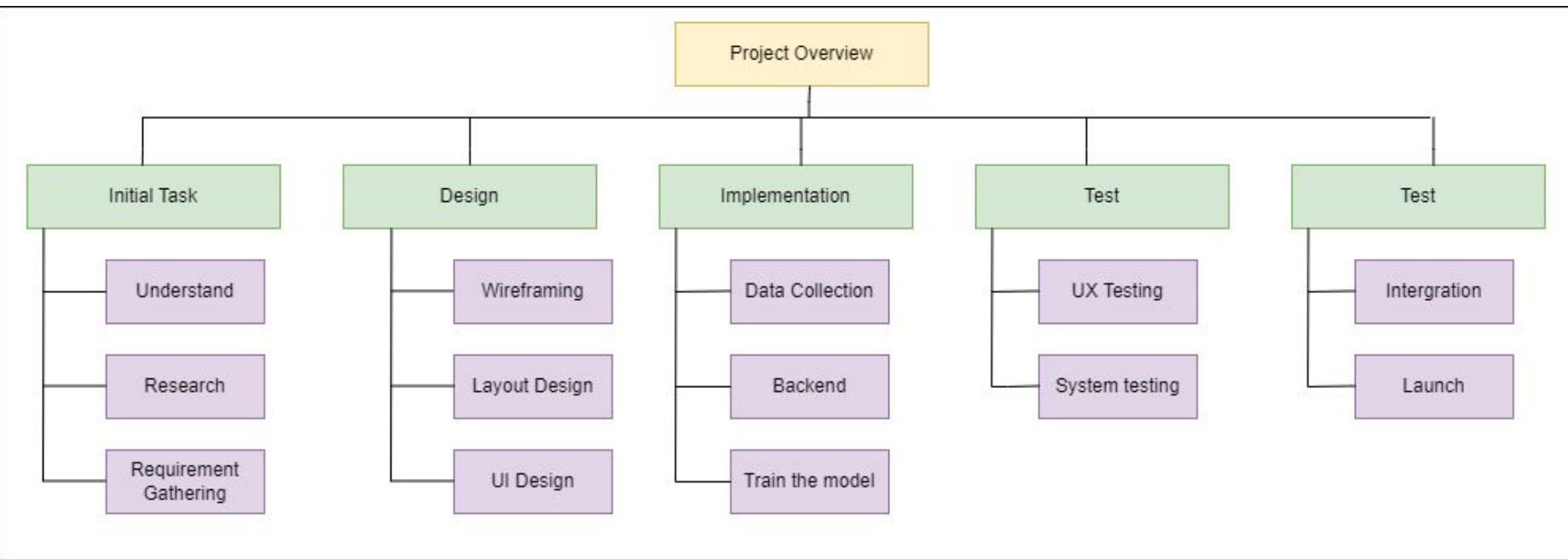
Research Question

What data sources and parameters are essential for personalized decision support in vanilla cultivation?

How to integrate machine learning for dynamic cultivation advice based on real-time data?

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

Quality Inspection Section

The screenshot displays a user interface for a design application. At the top, there is a horizontal toolbar with various icons: a magnifying glass (Search), a double arrow (Zoom), a trash can (Delete), a plus sign (New), a double arrow (Copy/Paste), a red 'X' (Delete), a double arrow (Move), a double arrow (Size), a double arrow (Align), a double arrow (Distribute), a double arrow (Group/UnGroup), and a double arrow (Format). Below the toolbar, a vertical toolbar on the left side contains icons for 'Logo' (a circular icon with a logo), 'Text' (a text input field icon), 'Image' (a camera icon), and 'Table' (a grid icon). The main workspace features a rounded rectangular container labeled 'Quality Inspection Section'. Inside this container, there is a smaller rounded rectangle containing a large red 'X' icon. To the right of the inner rectangle is a button labeled 'Button'. On the far right, there is a preview window showing the component's visual representation. The entire interface has a light gray background.

Harvest Reporting

Logo

Harvest Reporting

Name

Harvest (Weight/Kg)

Revenue (Rs.)

Update

Form

Chart

Bell

User Profile

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 icon

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

GREEN MIND

TODAY: 2024-3-15 27.7°C Clouds

Disease Statistics

Reported Cases Cases Chart Disease Upload

10 Total Recorded Diseases (Last 30 days)

13 Total Damaged Cases (Last 30 days)

Notice

Disease Information.

All Reported Cases

ID	Reported Date	Condition
2023-10-11_XN0X0T0dC0B0z0s0oG1	2023-10-11	Fungal Disease
2023-10-17_J579pWK7M1pJZRtDs	2023-10-17	Fungal Disease
2023-11-17_ZF6abu9h875neff1Wm8	2023-11-17	Damaged
2023-11-20_KdhUuInUlj3qOmxZYb	2023-11-20	Damaged
2023-11-25_a99r2HfYNNzhYRFvyy2P	2023-11-25	Damaged

© 2024 Manila Quality Inspection Site

GREEN MIND

TODAY: 2024-3-15 27.7°C Clouds

Disease Infection

Reported Cases Disease Upload

Upload or Drop Image HERE

Notice

Disease Information.

© 2024 Manila Quality Inspection Site

GREEN MIND

TODAY: 2024-3-15 27.7°C Clouds

Disease Chart

Reported Cases Cases Chart Disease Upload

Number of Cases

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

© 2024 Manila Quality Inspection Site

GREEN MIND

TODAY: 2024-3-15 27.7°C Clouds

IOT Control Panel

Manual Guided

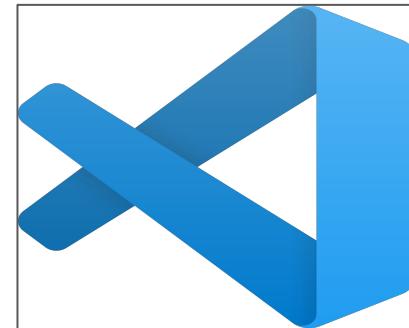
Temperature 31 °C Humidity 83

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 Manila Quality Inspection Site

Technologies Used



Future Works

- ❖ It also creates qualitative interfaces using advanced algorithms
- ❖ Develop mobile app
- ❖ Increase accuracy in ml



INDIVIDUAL PROGRESS

**W.M.Janith chathuranga
IT20658854**



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

Data Visualization and Decision Support

Background

Vanilla cultivation, a cornerstone of the agricultural industry, continually faces challenges exacerbated by disease risks, cultivation optimization complexities, and the imperative for stringent quality assessment. The conventional methodologies often fall short in providing real-time insights and tailored guidance to farmers, hindering their ability to make informed decisions promptly.

Recognizing this gap, our research endeavors to pioneer an integrated solution, focusing on the "Data Visualization and Decision Support" component to empower vanilla farmers with the tools they need for success. Tables and graphs provide detailed insights into harvest data and other crops, while a sophisticated machine learning system processes information from the other three components, offering predictions crucial for informed cultivation decisions.

Research Problem

Problem : Designing an intuitive interface that effectively caters to the diverse technological backgrounds of vanilla farmers.

Context: Vanilla farmers may have varying levels of technological proficiency, requiring the interface to be user-friendly and accessible to a broad audience.

Problem: Ensuring seamless integration of data from the Plant Disease Analyzer and Plant Growing and Fertilization components for real-time insights.

Context: Integrating data from multiple sources poses challenges in maintaining data accuracy, consistency, and timeliness.

Enhancing the accuracy and speed of real-time predictions generated by the machine learning system.

Main Objectives -

By leveraging data visualization techniques and intelligent decision support systems, this component aims to provide real-time, actionable information to farmers, enhancing their ability to make informed decisions, optimize cultivation practices, and ultimately elevate the productivity and success of vanilla farming.

Sub Objectives -

Integrate machine learning algorithms to process and analyze data from fault detection, condition measurement and disease detection components and seamlessly integrate machine learning predictions to provide real-time insights.

Design visually appealing representations of harvest and crop data using tables, graphs, and charts.

Ensure smooth data exchange between the Data Visualization and Decision Support component and other components such as Plant Disease Analyzer and Plant Growing and Fertilization.

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.

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



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

Plant Disease Analyzer



Background

- Vanilla is an economically and commercially important export crop in Sri Lanka. The main challenge for high commercial-value vanilla plants is susceptibility to various diseases. Vanilla farmers, agricultural researchers, and exporters face a huge crisis as vanilla plants are susceptible to various diseases.
- The "Plant Disease Analyzer" component contained in our research determines whether the vanilla plant is healthy or not, and if it has a disease, it gives details about the disease and what remedies should be done for it.
- 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

Functional Requirements

1. Image Capture and Upload:

- Users should be able to upload images of diseased plants.
- The system should support various image formats, including photos.

2. Image Processing:

- The system should process the uploaded images to extract relevant information.
- It should identify key plant parts, such as leaves, stems, and fruits.
- Image enhancement and noise reduction may be required to improve analysis accuracy.

3. Disease Detection:

- The system should be able to detect diseases or anomalies in the plant based on the processed images.
- It should use image analysis techniques, machine learning, or AI algorithms for disease identification.

4. Disease Classification:

- The system should classify the detected diseases into specific categories.
- It should provide information on the identified disease, including its name and symptoms.

5. Accuracy and Confidence Levels:

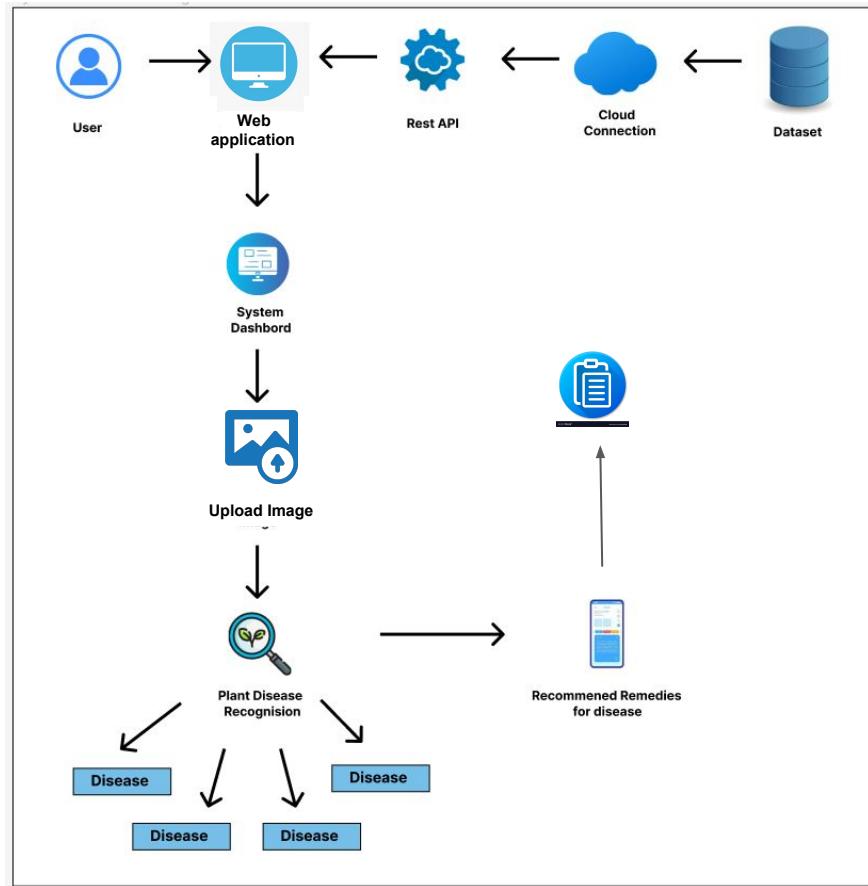
- The system should provide a confidence level or accuracy score for the disease detection.
- Users should be aware of how certain the system is in its diagnosis.

Non - Functional 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 the image directory (the folder where your images are stored)
image_directory = '/content/drive/MyDrive/Vaishu Project/PFI/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.open(img_path).convert('L')
        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')

[ ] /usr/local/lib/python3.10/dist-packages/keras/src/utils/image_utils.py:409: UserWarning: grayscale is deprecated. Please use color_mode = "grayscale"
  warnings.warn("grayscale is deprecated. Please use color_mode = "grayscale"
Epoch 0/150
3/3 [=====] - 0s 48ms/step - loss: 1.7223 - accuracy: 0.3333 - val_loss: 1.4299 - val_accuracy: 0.4545
Epoch 1/150
3/3 [=====] - 0s 22ms/step - loss: 1.4361 - accuracy: 0.4643 - val_loss: 1.4528 - val_accuracy: 0.4545
Epoch 2/150
3/3 [=====] - 0s 22ms/step - loss: 1.3292 - accuracy: 0.4643 - val_loss: 1.3292 - val_accuracy: 0.4545
Epoch 3/150
3/3 [=====] - 0s 22ms/step - loss: 1.3214 - accuracy: 0.4643 - val_loss: 1.3589 - val_accuracy: 0.4545
Epoch 4/150
3/3 [=====] - 0s 22ms/step - loss: 1.3229 - accuracy: 0.4643 - val_loss: 1.3183 - val_accuracy: 0.4545
Epoch 5/150
3/3 [=====] - 0s 22ms/step - loss: 1.3229 - accuracy: 0.4643 - val_loss: 1.3183 - val_accuracy: 0.4545
Epoch 6/150
3/3 [=====] - 0s 22ms/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 22ms/step - loss: 1.3294 - accuracy: 0.5238 - val_loss: 1.2926 - val_accuracy: 0.4545
Epoch 9/150
3/3 [=====] - 0s 22ms/step - loss: 1.2189 - accuracy: 0.4762 - val_loss: 1.2357 - val_accuracy: 0.4545
Epoch 10/150
3/3 [=====] - 0s 22ms/step - loss: 1.1862 - accuracy: 0.5476 - val_loss: 1.2357 - val_accuracy: 0.4545
Epoch 11/150
3/3 [=====] - 0s 22ms/step - loss: 1.1806 - 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 20ms/step - loss: 0.9869 - accuracy: 0.5952 - val_loss: 1.3551 - val_accuracy: 0.5909
Epoch 14/150
```

Model Training

disease_detection.ipynb

```
# Save the model
model.save("model_van5")
```

Epoch 19/150
3/3 [=====] - 0s 21ms/step - loss: 0.0156 - accuracy: 0.7500 - val_loss: 1.8084 - val_accuracy: 0.5000
Epoch 20/150
3/3 [=====] - 0s 20ms/step - loss: 0.5719 - accuracy: 0.7857 - val_loss: 1.8689 - val_accuracy: 0.6364
Epoch 21/150
3/3 [=====] - 0s 21ms/step - loss: 0.4732 - accuracy: 0.8214 - val_loss: 2.0555 - val_accuracy: 0.5099
Epoch 22/150
3/3 [=====] - 0s 21ms/step - loss: 0.4075 - accuracy: 0.8452 - val_loss: 2.0300 - val_accuracy: 0.5999
Epoch 23/150
3/3 [=====] - 0s 21ms/step - loss: 0.3280 - accuracy: 0.8029 - val_loss: 2.5176 - val_accuracy: 0.5455
Epoch 24/150
3/3 [=====] - 0s 21ms/step - loss: 0.3093 - accuracy: 0.9167 - val_loss: 2.2693 - val_accuracy: 0.5455
Epoch 25/150
3/3 [=====] - 0s 20ms/step - loss: 0.2030 - accuracy: 0.9524 - val_loss: 3.4512 - val_accuracy: 0.5988
Epoch 26/150
3/3 [=====] - 0s 24ms/step - loss: 0.1665 - accuracy: 0.9846 - val_loss: 3.3393 - val_accuracy: 0.5455
Epoch 27/150
3/3 [=====] - 0s 21ms/step - loss: 0.2850 - accuracy: 0.9643 - val_loss: 2.0405 - val_accuracy: 0.6364
Epoch 28/150
3/3 [=====] - 0s 20ms/step - loss: 0.1458 - accuracy: 0.9805 - val_loss: 2.0973 - val_accuracy: 0.5999
Epoch 29/150
3/3 [=====] - 0s 20ms/step - loss: 0.1007 - accuracy: 0.9762 - val_loss: 3.6885 - val_accuracy: 0.5999
Epoch 30/150
3/3 [=====] - 0s 20ms/step - loss: 0.0436 - accuracy: 1.0000 - val_loss: 3.6333 - val_accuracy: 0.5999
Epoch 31/150
3/3 [=====] - 0s 24ms/step - loss: 0.0250 - accuracy: 1.0000 - val_loss: 3.8173 - val_accuracy: 0.5999
Epoch 32/150
3/3 [=====] - 0s 21ms/step - loss: 0.0210 - accuracy: 1.0000 - val_loss: 4.2395 - val_accuracy: 0.5999

disease_detection.ipynb

```
# 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(x_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()
```

disease_detection.ipynb

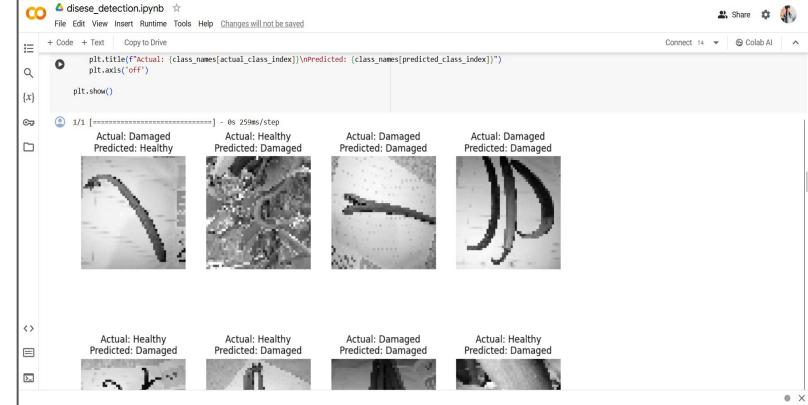
```
# Import matplotlib.pyplot as plt
import random
import numpy as np

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

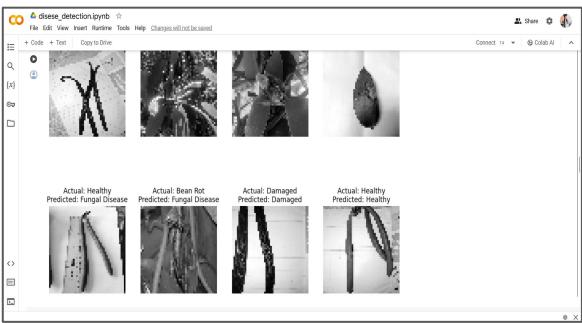
random_indices = random.sample(range(len(x_val)), sample_size)
sample_images = x_val[random_indices]
sample_labels_actual = y_val[random_indices]
sample_labels_predicted = model.predict(sample_images)

# 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, 1)):
    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



```
disease_detection.ipynb
```

```
+ Code + Test Copy to Drive
```

```
File Edit View Insert Runtime Tools Help Changes will not be saved
```

```
convert labels to numerical values
labels = pd.get_dummies(y)
X = np.array(X)
y = labels.values

# split the data into training sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Data augmentation
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

datagen.fit(X_train)

# Create a VGG model as a feature extractor
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3))

# Freeze the layers in the base model
for layer in base_model.layers:
    layer.trainable = False
```

```
disease_detection.ipynb
```

```
+ Code + Text Copy to Drive
```

```
File Edit View Insert Runtime Tools Help Changes will not be saved
```

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

```
# Define the image directory (the folder where your images are stored)
image_directory = '/content/drive/MyDrive/ML Project/PMI/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 = [f for f in os.listdir(category_path) if f.endswith('.jpg')]
    for file in image_files:
        img = image.load_img(os.path.join(category_path, file), target_size=(64, 64))
        img_array = np.array(img)
        X.append(img_array)
        Y.append(categories.index(category))
```

```
disease_detection.ipynb
```

```
+ Code + Test Copy to Drive
```

```
File Edit View Insert Runtime Tools Help Changes will not be saved
```

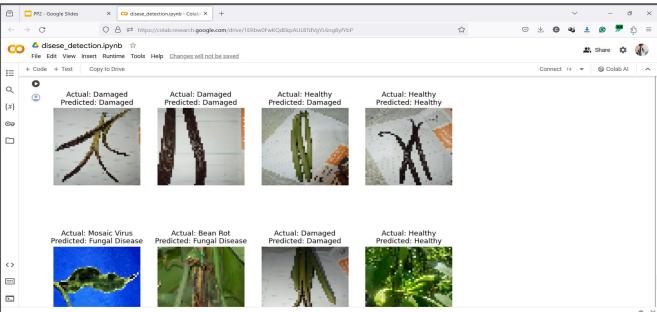
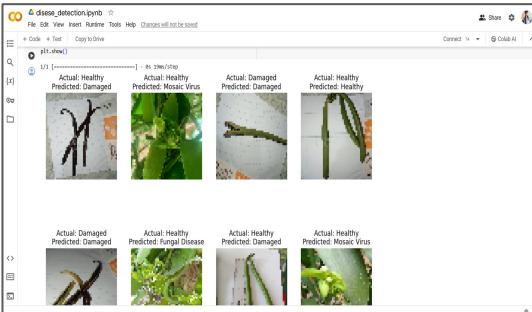
```
Epoch 27/50 - 0s/2ms/step - loss: 0.3024 - accuracy: 0.8824 - val_loss: 1.0300 - val_accuracy: 0.5800
5/5 [=====] - 0s/2ms/step - loss: 0.3241 - accuracy: 0.8824 - val_loss: 1.7344 - val_accuracy: 0.5800
```

```
[x]
import matplotlib.pyplot as plt
import numpy as np

# Data visualization
sample_size = 16
img_size = 64
num_rows = sample_size // num_cols

sample_index = np.random.randint(0, len(X), sample_size)
sample_images = np.array([X[i] for i in sample_index])
sample_labels_actual = y[sample_index]
sample_labels_predicted = model.predict(sample_images)

# Create a figure
plt.figure(figsize=(12, 12))
for i in range(sample_size):
    plt.subplot(num_rows, num_cols, i + 1)
    plt.imshow(sample_images[i], cmap='gray')
    plt.title(f'Actual: {category_names[sample_labels_actual[i]]}\nPredicted: {category_names[sample_labels_predicted[i]]}')
    plt.xlabel(f'Actual: {category_names[sample_labels_actual[i]]}\nPredicted: {category_names[sample_labels_predicted[i]]}')
    plt.title(f'Actual: {category_names[sample_labels_actual[i]]}\nPredicted: {category_names[sample_labels_predicted[i]]}')
```



VS Codes

This screenshot shows the Frontend project structure in VS Code. The Explorer sidebar shows a folder named 'ml-api' containing a file 'main.py'. The code in 'main.py' is as follows:

```
1 from fastapi import FastAPI, File, UploadFile
2 from fastapi.responses import FileResponse
3 from fastapi.middleware.cors import CORSMiddleware
4 from pathlib import Path
5 import uvicorn
6 import numpy as np
7 from io import BytesIO
8 from PIL import Image
9 import tensorflow as tf
10 from fastapi.responses import JSONResponse
11 import uv3
12 import temfile
13 import joblib
14 from pydantic import BaseModel
15 from spec_quality import measure_objects
16
17
18 app = FastAPI()
19 origins = [
20     "http://localhost:3000",
21     "http://localhost:3001"
22 ]
23 app.add_middleware(
```

The terminal shows the command: U2FsdGVkX18dfa9yuAYUv8x82qICvdexs0AMtFYB= /111222333

Frontend

This screenshot shows the Frontend project structure in VS Code. The Explorer sidebar shows a folder named 'ml-api' containing a file 'main.py'. The code in 'main.py' is as follows:

```
1 from fastapi import FastAPI, File, UploadFile
2 from fastapi.responses import FileResponse
3 from fastapi.middleware.cors import CORSMiddleware
4 from pathlib import Path
5 import uvicorn
6 import numpy as np
7 from io import BytesIO
8 from PIL import Image
9 import tensorflow as tf
10 from fastapi.responses import JSONResponse
11 import uv3
12 import temfile
13 import joblib
14 from pydantic import BaseModel
15 from spec_quality import measure_objects
16
17
18 app = FastAPI()
19 origins = [
20     "http://localhost:3000",
21     "http://localhost:3001"
22 ]
23 app.add_middleware(
```

The terminal shows the command: U2FsdGVkX18dfa9yuAYUv8x82qICvdexs0AMtFYB= /111222333

Backend

This screenshot shows the Backend project structure in VS Code. The Explorer sidebar shows a folder named 'ml-api' containing a file 'main.py'. The code in 'main.py' is as follows:

```
1 from fastapi import FastAPI, File, UploadFile
2 from fastapi.responses import FileResponse
3 from fastapi.middleware.cors import CORSMiddleware
4 from pathlib import Path
5 import uvicorn
6 import numpy as np
7 from io import BytesIO
8 from PIL import Image
9 import tensorflow as tf
10 from fastapi.responses import JSONResponse
11 import uv3
12 import temfile
13 import joblib
14 from pydantic import BaseModel
15 from spec_quality import measure_objects
16
17
18 app = FastAPI()
19 origins = [
20     "http://localhost:3000",
21     "http://localhost:3001"
22 ]
23 app.add_middleware(
```

The terminal shows the command: U2FsdGVkX18dfa9yuAYUv8x82qICvdexs0AMtFYB= /111222333

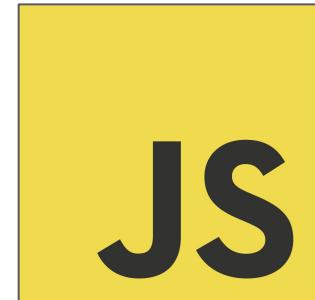
ml-api

This screenshot shows the ml-api project structure in VS Code. The Explorer sidebar shows a folder named 'ml-api' containing a file 'main.py'. The code in 'main.py' is as follows:

```
1 from fastapi import FastAPI, File, UploadFile
2 from fastapi.responses import FileResponse
3 from fastapi.middleware.cors import CORSMiddleware
4 from pathlib import Path
5 import uvicorn
6 import numpy as np
7 from io import BytesIO
8 from PIL import Image
9 import tensorflow as tf
10 from fastapi.responses import JSONResponse
11 import uv3
12 import temfile
13 import joblib
14 from pydantic import BaseModel
15 from spec_quality import measure_objects
16
17
18 app = FastAPI()
19 origins = [
20     "http://localhost:3000",
21     "http://localhost:3001"
22 ]
23 app.add_middleware(
```

The terminal shows the command: U2FsdGVkX18dfa9yuAYUv8x82qICvdexs0AMtFYB= /111222333

Technologies Used



Future work

- ❖ Improve the data set and Increase accuracy.
- ❖ Develop mobile app



Gunawardhana O.W.
IT20263980



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

Quantitative Quality Measurement

Background

The "Quantitative Quality Measurement" component is crucial in smart vanilla cultivation, analogous to disease management's significance in crop health. Like disease detection, timely intervention is vital. Vanilla pod quality assessment holds immense importance as well. Accurate measurements and standardization of attributes like size, color intensity, and surface texture are pivotal for market value and desirability. This component bridges the gap between traditional subjective methods and precise, data-driven evaluations, enhancing vanilla cultivation practices and establishing consistent quality standards.

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.

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

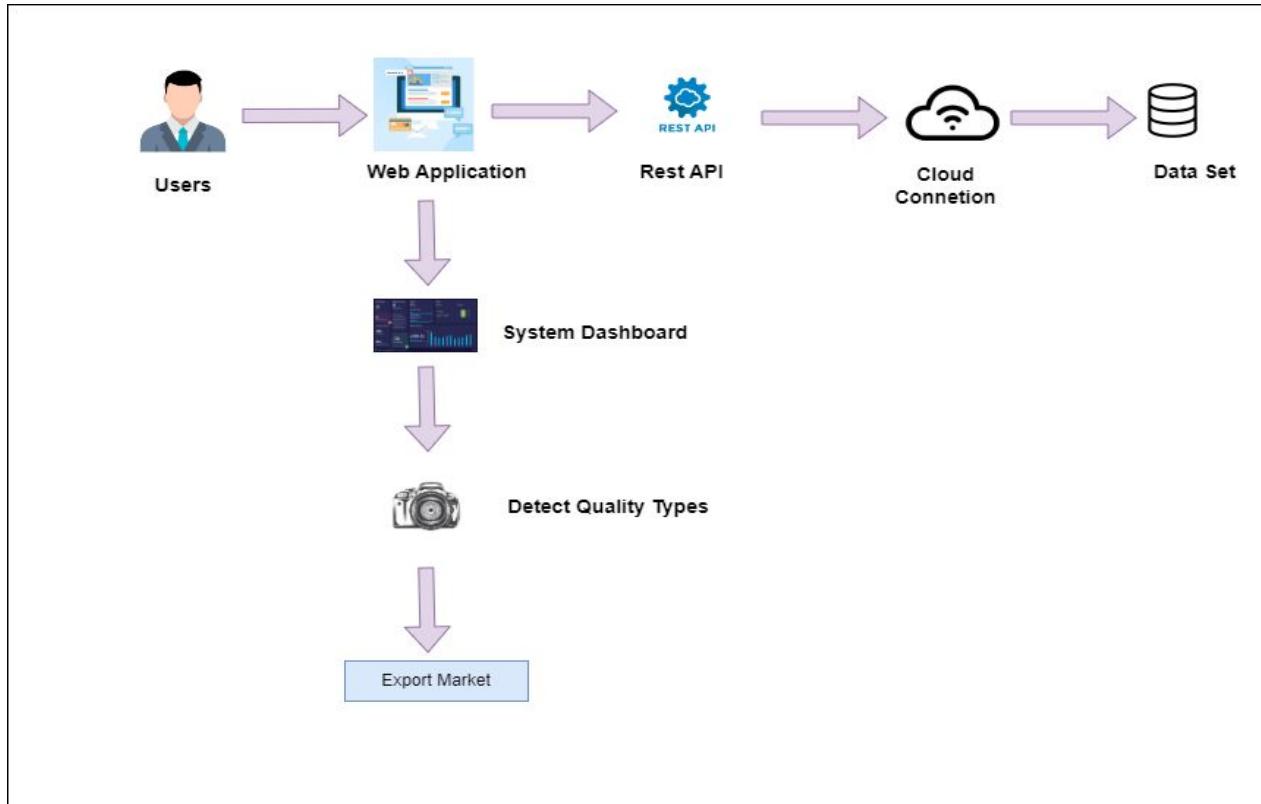
Functional Requirements

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

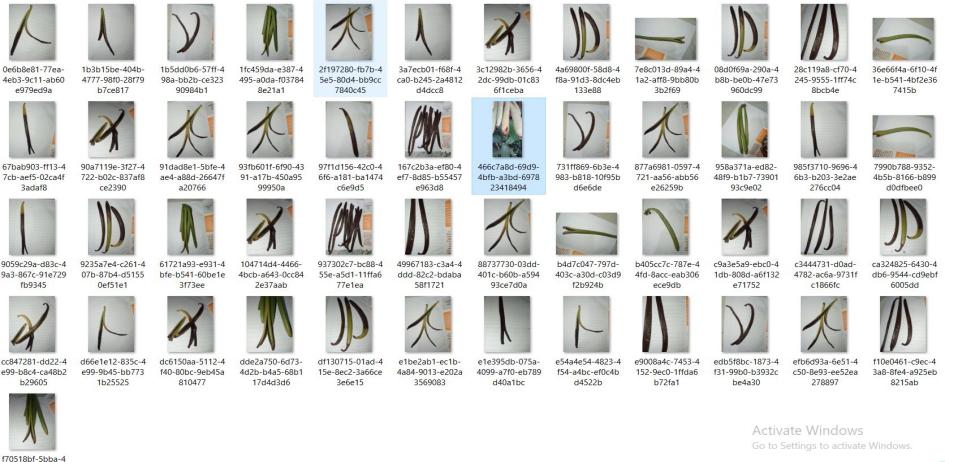
Non - Functional Requirements

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

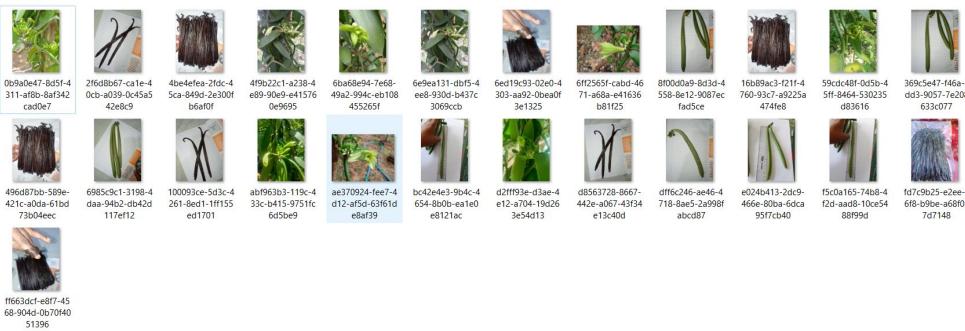
System Overview Diagram



DataSet



Activate Windows
Go to Settings to activate Windows.



Future work

- ❖ Looking forward to using the iot sensors to see the weight of the pods.



Demo

Herath H.M.R.K.
IT20665548



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

Plant Growing and Fertilization

Background

Optimizing plant development and guaranteeing robust yields requires an understanding of the complex interactions between environmental factors like light, water, temperature, and soil quality as well as the prudent use of fertilizers. Researchers seek to improve agricultural output while reducing environmental impact and promoting sustainable practices that protect crop health and ecosystem integrity. They do this by carefully examining the needs unique to each plants and developing customized fertilization tactics.

Research Problem

Sustainability of the environment and global food security depend on research on sustainable fertilizing methods for particular crop cultivation. In addition to reducing resource waste and environmental harm, effective fertilizers tailored to various soil types can boost output. Comprehending soil microbes improves plant health, while precision farming and controlled-release fertilizers maximize nutrient usage. Research over the long term preserves soil fertility and ecosystem resilience. There are sustainable substitutes by investigating other nutrient sources like compost and biochar. Resilient agricultural systems that protect natural resources and fulfill future demands are promoted by interdisciplinary research.

Main Objectives -

Using IoT devices and machine learning technology to create quality crops by controlling water and fertilizer application and environmental climate

Sub Objectives -

Determining the history, and future of fertilizer and water rates using IOT data

Using IoT devices to use rainwater for water conservation

Requirements

Functional Requirements

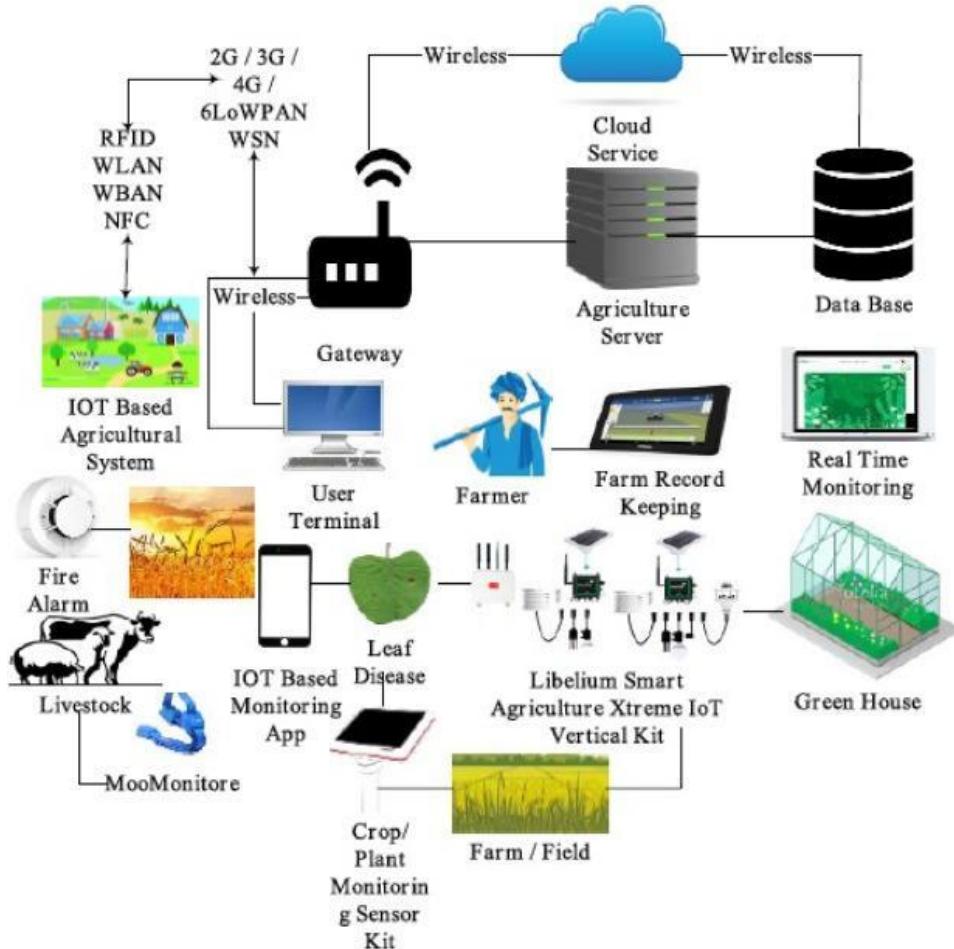
- Using IoT Devices
- Calculation of usage history
- Soil Moisture Monitoring
- Automated Watering System
- Fertilization Management
- Environmental Monitoring
- Predictive Analytics

Requirements

Non - Functional Requirements

- Reliability
- Scalability
- Usability
- Performance
- Security

System Overview Diagram



Future work

- Water and fertilizer collection using drones for a large area
- Using a camera to measure the quality of the vanilla plant
- Use of renewable energy
- Removal of diseased leaves and harvesting by robots
- Integration of additional environmental sensors for comprehensive monitoring
- Enhancement of machine learning algorithms for more accurate predictions
- Incorporation of remote monitoring and control features for ease of management

Demo



THANK YOU
