

A Project Report On
“Plant Disease Prediction Using Deep Learning.”
Submitted in partial fulfillment of the Requirements for the award of the
Degree of
BACHELOR OF SCIENCE (COMPUTER SCIENCE)
SEM VI Examination

Submitted by
Surajkumar Yadav
ROLL NUMBER: 2225509
Under the esteemed guidance of:
Prof. Shreya Tiwari
Assistant Professor



DEPARTMENT OF COMPUTER SCIENCE
SEVA SADAN's R. K. Talreja College
of Arts, Science and Commerce,
Ulhasnagar-421 003
(Affiliated to University of Mumbai)
A.C. 2022-2023

Seva Sadan's R.K.Talreja College of Arts, Science & Commerce
(Affiliated to University of Mumbai)
Maharashtra, Ulhasnagar - 421003

DEPARTMENT OF COMPUTER SCIENCE



CERTIFICATE

This is to certify that the project entitled, "**Plant Disease Prediction Using Deep Learning**", is bonafied work of **Surajkumar Yadav** bearing Seat.No: (2225509) submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF SCIENCE in COMPUTER SCIENCE from University of Mumbai.

Internal Guide

Coordinator

External Examiner

Date:

College Seal

ABSTRACT

The project "Plant Disease Prediction using Deep Learning" aims to develop a system that can accurately identify various diseases in plants using machine learning algorithms. The system will be trained on a dataset of images of plants affected by different diseases and will use deep learning techniques to extract relevant features from the images. The extracted features will then be used to train a model that can classify new images of plants based on the presence or absence of disease. The project will involve the use of deep learning frameworks such as TensorFlow, Keras, and will require the use of image processing techniques such as convolutional neural networks (CNNs) and transfer learning. Other technologies that are used to built the webapps are React JS , Streamlit, javascript. The data used had thousands of images of various plant diseases in a very organized manner.

The project involves training a deep learning model using a large dataset of images of healthy and diseased plants to learn and recognize patterns that distinguish between the two. The trained model is then used to classify new images of plants as healthy or diseased, enabling early detection and effective treatment of plant diseases.

The ultimate goal of the project is to provide a tool that can help farmers and plant pathologists quickly identify and treat plant diseases, leading to higher crop yields and better food security.

Acknowledgment

It is a genuine pleasure to express my profound gratitude and deep regards to my guide "**Prof. Shreya Tiwari**" for her exemplary guidance, monitoring and constant encouragement. I would like to express my special thanks to the computer science department from my college who gave me the golden opportunity to do this wonderful project on the topic "**Plant Disease Prediction Using Deep Learning**", which helped me in doing a lot of Research and I came to know about so many new things.

We are also thankful to all of our teachers of the department who helped us in a number of ways by providing various resources and moral support. Last of all we are grateful to our family, who are, always with us in our every step of life.

TABLE OF CONTENTS

<u>CHAPTER 1</u>	6
1.Introduction, purpose, Existing system,	
Proposed system, Deep Learning, CNN,	
Transfer Learning.	6-16
<u>CHAPTER 2</u>	17
2.Project background, User requirement, user of system,	
Requirement analysis, SDLC, Agile model, design, phase,	
A&D	17-25
<u>CHAPTER 3</u>	26
3.System Analysis, UML diagrams	20-24
<u>CHAPTER 4</u>	25
4.Database structure, Snapshots	25-39
<u>CHAPTER 5</u>	40
5.System code, Limitation, Future scope	40-48
<u>REFERENCE</u>	49

CHAPTER 1

INTRODUCTION

Plant diseases can have a significant impact on crop yield and quality, resulting in financial losses for farmers and potentially affecting global food security. Early detection and accurate diagnosis of plant diseases are crucial for effective management and control of plant diseases. The use of deep learning algorithms in plant disease prediction offers a promising solution to this problem.

The Plant Disease prediction using Deep Learning project aims to develop an accurate and reliable system for early detection and diagnosis of plant diseases using machine learning techniques. The project involves the use of deep learning algorithms to analyze large datasets of images of healthy and diseased plants to identify patterns that distinguish between the two.

The deep learning model is trained to recognize visual cues and patterns associated with different types of plant diseases. Once the model is trained, it can be used to classify new images of plants and accurately identify the presence of any diseases.

The benefits of this project are significant, as early detection of plant diseases can help farmers take preventive measures and reduce the spread of diseases, leading to increased crop yields and reduced financial losses. Moreover, the system can help farmers make informed decisions about crop management, including the use of pesticides and fertilizers, thus reducing costs and minimizing environmental damage.

In summary, the Plant Disease prediction using Deep Learning project offers a promising solution to the challenge of early detection and diagnosis of plant diseases, with significant potential benefits for farmers and the agriculture industry as a whole.

1.1 Purpose of the project.

The world is changing very rapidly with Artificial Intelligence and other technological advances that are making the human more comfortable and easier. Our application aims the same to automate the recognition process of Diseases in plants and provide treatment . The purpose of the Plant Disease prediction using Deep Learning project is to develop a reliable and accurate system for early detection and diagnosis of plant diseases using machine learning techniques. The project aims to leverage the power of deep learning algorithms to analyze large datasets of images of healthy and diseased plants and identify patterns that can distinguish between the two.

The project's primary objective is to improve crop yield and quality while reducing financial losses caused by plant diseases. By detecting and diagnosing plant diseases at an early stage, the project can help farmers take preventive measures and reduce the spread of diseases, leading to increased crop yields and reduced financial losses.

1.2 Existing System

In the present system all work is done manually by checking the condition of plants in a field which requires accurate knowledge of the checker preferably a Doctor or a Biologist. The system is very time consuming and requires a lot of human labour. A major drawback is to keep the records of diseased plants and the way to treat it and also a large amount of time gets passed from checking and treating the plants which may worsen the condition of the plants.

Traditional systems used for disease prediction in plants have several weaknesses, including:

- **Reliance on Visual Symptoms**: Traditional disease prediction systems often rely on visual symptoms to diagnose plant diseases, which can be unreliable and subject to interpretation. Visual symptoms can also be difficult to detect in the early stages of disease, leading to delays in treatment and a higher risk of crop loss.
- **Lack of Data Integration**: Traditional disease prediction systems often do not integrate data from multiple sources, such as weather data or soil conditions, which can affect disease development. This can limit their accuracy and reliability, as they may fail to account for all the factors that contribute to disease outbreaks.
- **Limited Flexibility**: Traditional disease prediction systems are often inflexible, as they are designed to detect specific diseases or pests. This can make it difficult to adapt to new diseases or changing environmental conditions, leading to delays in response and potentially higher crop losses.

- High Cost: Many traditional disease prediction systems are costly to implement, as they require specialized equipment or expertise. This can make them inaccessible to small-scale farmers or those in developing countries, limiting their usefulness in areas where plant diseases are a significant threat to crop yield and food security.

Overall, the weaknesses of traditional disease prediction systems highlight the need for more advanced and flexible systems that can integrate data from multiple sources and accurately predict and diagnose plant diseases. The use of technologies such as deep learning and machine learning offers promising solutions to these challenges.

1.3 Proposed system

To overcome the drawbacks of the existing system, the proposed system has been evolved. This project aims to reduce the manual work and saving time to generate accurate results of plant Diseases. The system provides with the best user interface.

Advantages of Proposed System:

- It is trouble-free to use.
- It is a relatively fast approach to Predict Diseased plants.
- Is highly reliable, approximate result most of the times.
- Best user Interface
- Availability.

※ There are 2 different systems that I am proposing:

1. A React JS web App:

This app runs locally on a device with FastAPI as backend and React JS as frontend. The deep learning model is stored as .h5 file and is called by fastapi on localhost which is used on frontend for predictions. The frontend includes a basic structure of an app which has a input parameter for images

where we have to upload the plants leaves images which is sent to the backend.

2. A Streamlit Web app hosted on cloud:

This app is fully functional web app accessible from anywhere in the world and uses Streamlit as frontend and Google Cloud Platforms [Cloud Functions] as Backend. The following modules are there in the app:

- 1. Admin Login Module:** This module is for logging into the system and accessing all the services with proper credentials, You can signup by using the signup button and if you forget your password or username there are options to retrieve it .
- 2. Predict Disease Module:** This is the module where you upload you plants image to get the results, there are two options either you can upload a file from your device or you can click an image from the devices camera.
- 3. Disease Information module:** This module is where you can find all the information about different Diseases that affect the plants and how to identify them and treat them.

Theory:

- **DEEP Learning:**

Deep learning is a type of machine learning that involves the use of neural networks with many layers, also called deep neural networks. These networks can learn to recognize complex patterns and relationships in data, making them well-suited for tasks such as image recognition, speech recognition, natural language processing, and more.

The main advantage of deep learning over traditional machine learning methods is its ability to automatically learn feature representations from raw data. This eliminates the need for manual feature engineering, which can be time-consuming and require expert knowledge.

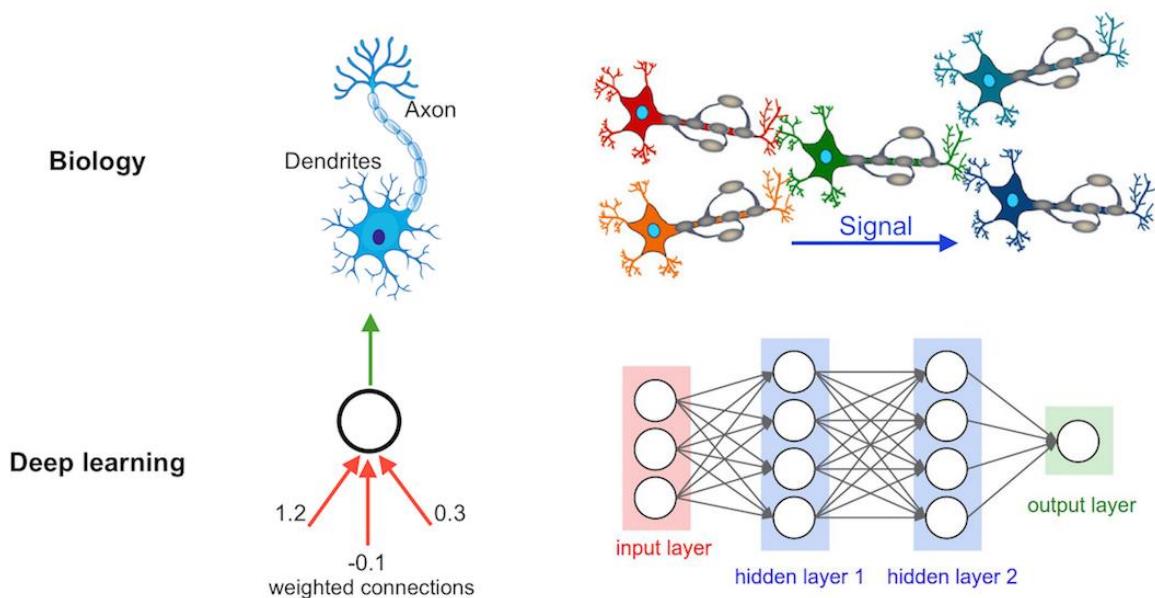
Deep learning models are typically trained using large datasets, and the learning process involves adjusting the weights and biases of the network in response to input data. This is done using an optimization algorithm such as stochastic gradient descent, which minimizes the error between the network's predicted output and the actual output.

Some popular types of deep learning models include convolutional neural networks (CNNs), which are commonly used for image and video processing; recurrent neural networks (RNNs), which are suited for sequential data such as time series or natural language processing; and generative adversarial networks (GANs), which can be used for generating new data based on existing datasets.

Despite their effectiveness, deep learning models can be computationally intensive and require significant computational resources. However, advances in hardware and software have made deep learning more accessible and easier to use for a wide range of applications.

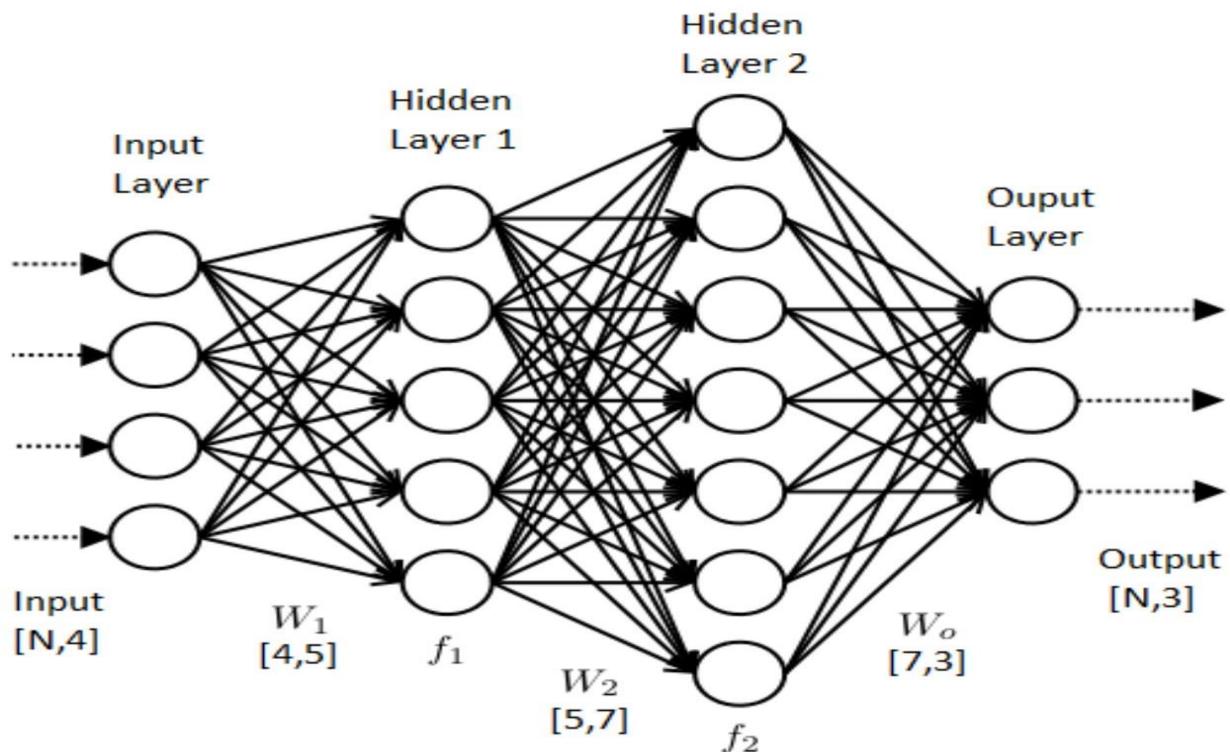
- **Convolution Neural Network:**

A neuron is a basic unit of a neural network, which is inspired by the structure and function of the human brain. A neuron receives input from other neurons or sensors, performs a computation, and produces an output that can be sent to other neurons or to an external system. The computation performed by a neuron involves a weighted sum of its inputs, followed by the application of an activation function. The weights determine the contribution of each input to the output, and they are adjusted during the learning process to optimize the performance of the network.

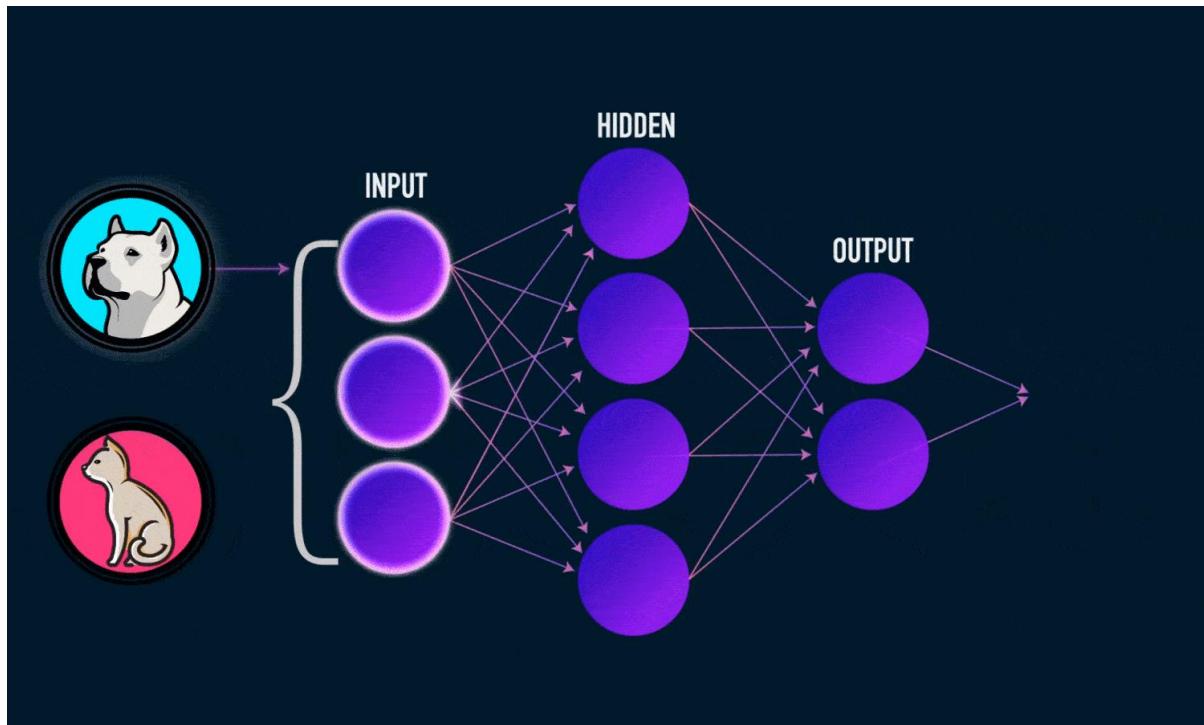


A neural network is a collection of interconnected neurons organized in layers. The input layer receives input from the external world, and the output layer produces the final output of the network. Between the input and output layers, there can be one or more hidden layers, each composed of multiple neurons. The neurons in one layer are connected to the neurons in the next layer, forming a directed graph. Neural networks can be trained using supervised or unsupervised

learning methods. In supervised learning, the network is presented with input-output pairs, and the weights are adjusted to minimize the error between the predicted and actual outputs. In unsupervised learning, the network learns to represent the input data without explicit labels.



Convolutional neural networks (CNNs) are a specialized type of neural network that is commonly used for image and video processing tasks. CNNs use convolutional layers to extract features from the input images, and pooling layers to reduce the dimensionality of the features. The output of the convolutional and pooling layers is flattened and fed into a fully connected layer that produces the final output. The convolutional layers in a CNN are designed to be translation-invariant, meaning that they can recognize patterns in different parts of the image regardless of their location. CNNs are often used in applications such as image classification, object detection, and facial recognition. They have achieved state-of-the-art performance on many benchmark datasets and are widely used in industry and academia.

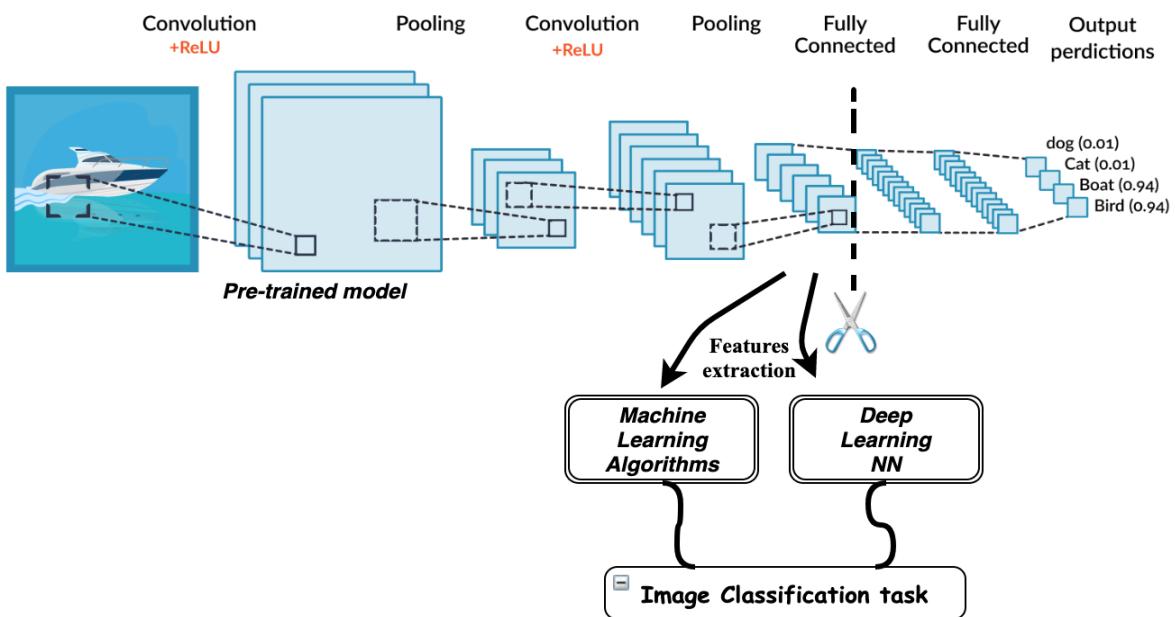


- **Transfer learning:**

Transfer learning is a technique in machine learning and deep learning where a pre-trained model is used as a starting point for a new task. Instead of starting from scratch and training a new model on a large dataset, transfer learning allows us to leverage the knowledge and feature representations learned by a pre-trained model on a related task. The idea behind transfer learning is that many real-world problems share some common underlying structure or features, and a model that has learned these features on a related task can be adapted to a new task with less data and computation.

There are two main approaches to transfer learning: fine-tuning and feature extraction. In fine-tuning, we start with a pre-trained model and train it on the new task using a small amount of new data. We typically freeze most of the layers of the pre-trained model and only fine-tune the last few layers that are specific to

the new task. This allows us to leverage the general feature representations learned by the pre-trained model while adapting to the specific characteristics of the new task. In feature extraction, we use the pre-trained model as a fixed feature extractor and train a new classifier on top of the extracted features. We typically remove the last few layers of the pre-trained model and use the output of the remaining layers as features for the new task. This approach is particularly useful when we have limited data for the new task or when the pre-trained model is too large to be fine-tuned on a new task.



Transfer learning has been shown to be effective in a wide range of applications, including computer vision, natural language processing, and speech recognition. It can significantly reduce the time and resources required to train a new model and improve the performance of the model on the new task. Pre-trained models such as VGG, ResNet, and BERT are widely used as starting points for transfer learning, and many open-source libraries provide pre-trained models and tools for fine-tuning and feature extraction.

CHAPTER 2

REQUIREMENT SPECIFICATION AND ANALYSIS PHASE

2.1 Project Background

The background for disease prediction in plants lies in the significant impact that plant diseases have on agricultural productivity, food security, and economic stability worldwide. Plant diseases can cause crop losses, reduce crop quality, increase production costs, and damage the environment. They can also pose a threat to human health, as some plant diseases can be transmitted to humans through contaminated food.

In the past, plant diseases were primarily managed through the use of chemical pesticides and fungicides. However, the overuse of these chemicals has led to resistance in plant pathogens and environmental damage, creating the need for more sustainable and effective disease management strategies.

Recent advances in computer vision, machine learning, and deep learning offer promising solutions for detecting and predicting plant diseases in a more accurate and timely manner. By analyzing large datasets of images of healthy and diseased plants, these technologies can identify patterns and features that are indicative of disease, allowing for early detection and diagnosis.

The development of disease prediction models using deep learning algorithms has the potential to revolutionize plant disease management by providing farmers with early warning systems that can help them take preventive measures and reduce the spread of diseases. This can lead to increased crop yields, reduced costs, and improved environmental sustainability.

2.2 User Requirements

To deliver the best service to the users we tried to find out the user's necessities which are below:

Administrator Aspect:

- Monitoring the whole system from the admin panel.
- Taking backup of the database.
- Creating, deleting and modifying the records.
- Add entities for the admin panel.
- Keeping the entities record like their details.

Users Aspect:

- **Getting smooth functionality and performance of the app.**
- **Getting to know the record of disease plants.**
- **Saving their time.**

2.4 Requirement Analysis

The **Plant Disease prediction using Deep Learning** is a web based application. The main purpose of the project is to provide a convenient and easy way for a farmer to identify and treat diseased plants. To develop this System. Software and Hardware Requirements are necessary. Requirements which are needed are given below briefly.

Hardware Requirements:

Processor-At least 3.0 GHZ (greater than i3 intel processors).

RAM-At least 4GB.

SSD would be great.

Your system must have camera for taking input.

Software Requirements:

Compatible operating system: Windows 8 and higher, Linux, Mac.

Front end design tool -Streamlit (python), CSS, React JS.

Back End-FastApi, Google Cloud Platforms.

Editor Tools -Notepad++, Visula Studio Code, Anaconda.

Web Browser-Google Chrome, Firefox, or any compatible updated browser.

Database tools: GCP [Cloud Storage Bucket]

2.5 Project Develop Languages

The Main language used in developing the web application is Python and its various libraries such as :

- **Tensorflow and keras** for model building and testing.
- **streamlit** for frontend.
- **Requests** for working with the model deployed on the GCP as backend.
- **Pillow** for image Manipulation.

- **Numpy** for mathematical calculations.
- **Streamlit_authenticator** for authentication purposes.
- **Streamlit_extras** for enhancing the look of frontend.
- **time** for Time related aspects.

2.6 Why Developers use python and its libraries :

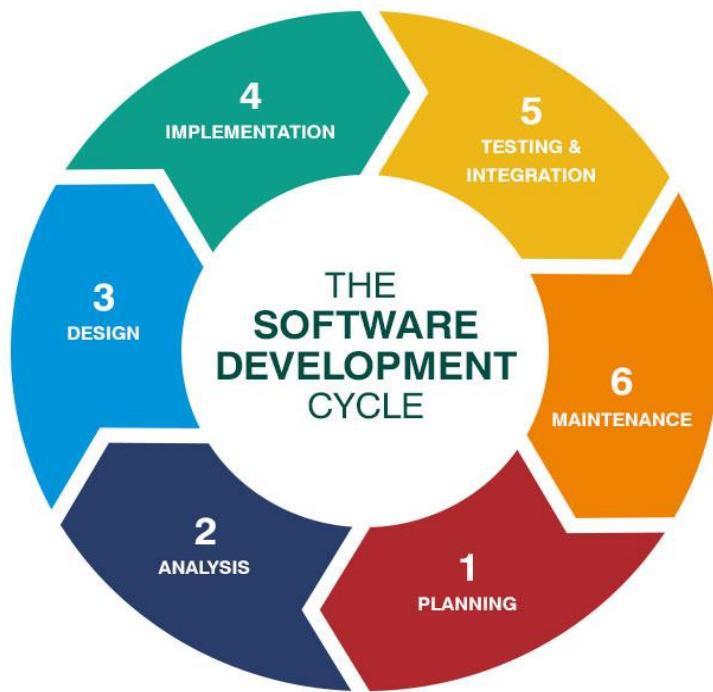
Python:

Python is a high-level, interpreted, and general-purpose dynamic programming language that focuses on code readability. It generally has small programs when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. Python ranks among the most popular and fastest-growing languages in the world. Python is a powerful, flexible, and easy-to-use language. In addition, the python community is very active. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

Advantages:

- 1 . Presence of third-party modules
2. Extensive support libraries (NumPy for numerical calculations, Pandas for data analytics,etc.)
3. Open source and large active community base
4. Versatile, Easy to read, learn and write
5. User-friendly data structures
6. High-level language
7. Dynamically typed language(No need to mention data type based on the value assigned, it takes data type).

2.7 Software Development Life Cycle



The software industry includes many different processes, for example, analysis, development, maintenance and publication of software. This industry also includes software services, such as training, documentation, and consulting. Our focus here is about software development life cycle (SDLC). So, due to that different types of projects have different requirements. Therefore, it may be required to choose the SDLC phases according to the specific needs of the project. These different requirements and needs give us various software development approaches to choose from during software implementation

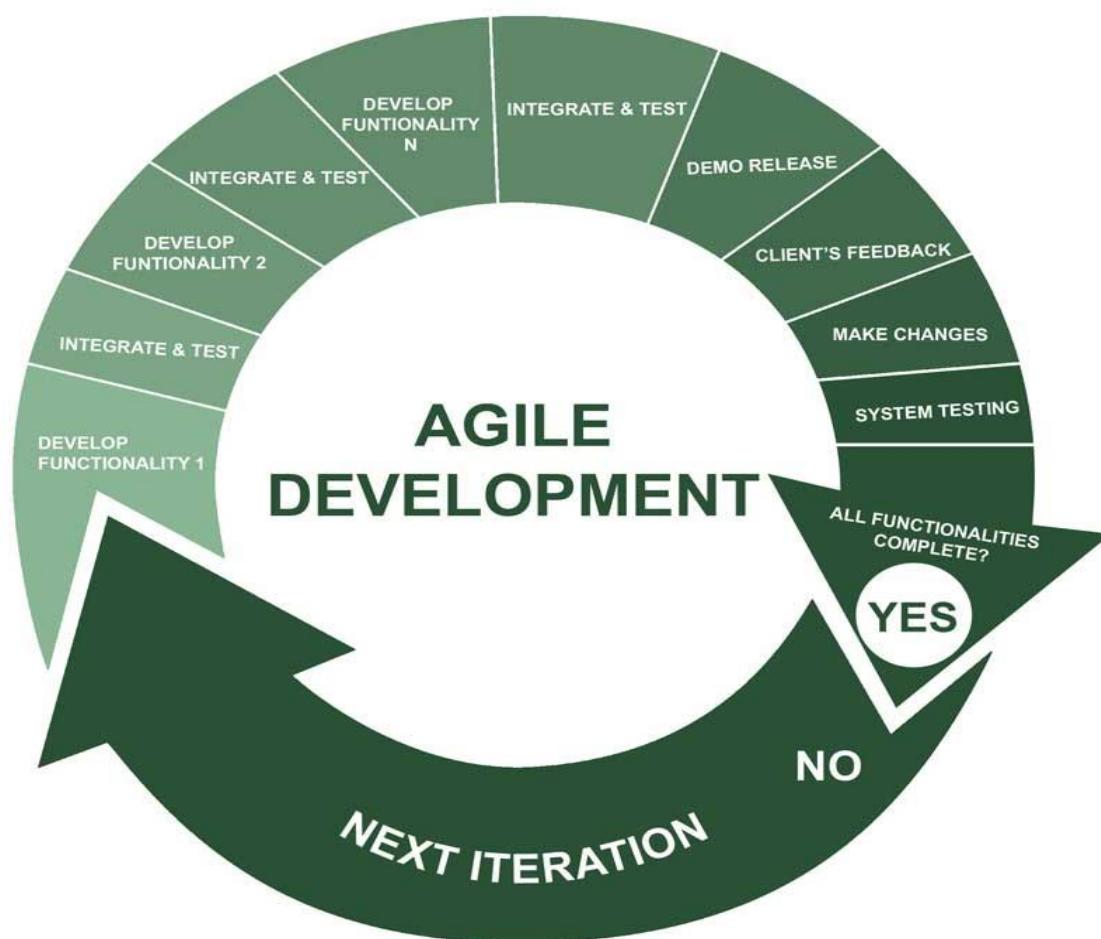
2.7.1. SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred to as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in the

process of software development. Following are the most important and popular SDLC models followed in the industry :

1. Waterfall model
2. V-Shaped model
3. Prototyping model
4. Spiral model
5. Iterative and Incremental model
6. Agile model
7. RAD model.

2.7.2 Agile model:



When it comes to data science projects, the Agile SDLC model is often considered the best fit. Here are some reasons why:

- **Flexibility:**

Data science projects are often exploratory in nature, with a high degree of uncertainty and risk involved. The Agile model is well-suited to this type of project as it allows for flexibility and adaptability throughout the development process. Teams can adjust their plans and priorities as they learn more about the problem domain, data, and model performance.

- **Iterative Development:**

The Agile model emphasizes iterative development and continuous improvement. This is particularly beneficial for data science projects, which require a lot of experimentation and fine-tuning to achieve optimal performance. The iterative approach allows for regular testing, evaluation, and feedback, enabling teams to refine their models and algorithms over time.

- **Cross-Functional Teams:**

Data science projects typically require collaboration between different teams, including data engineers, data scientists, and subject matter experts. The Agile model supports cross-functional teams, allowing for better communication and collaboration between team members with different skill sets.

- **Customer Collaboration:**

The Agile model emphasizes customer collaboration and feedback, ensuring that the end product meets the needs of the customer. For data science projects, this means involving subject matter experts and stakeholders in the development process to ensure that the models and algorithms are aligned with their needs and goals.

- **Short Timeframes:**

Data science projects often have short timeframes and tight deadlines, which can be challenging to manage using traditional SDLC models. The Agile model

emphasizes rapid prototyping and delivery, allowing teams to deliver working models and algorithms quickly and frequently.

Overall, the Agile SDLC model is well-suited to data science projects as it provides the flexibility, iterative development, cross-functional teams, customer collaboration, and short timeframes needed to develop effective and innovative solutions.

2.7.3 PhasesAgile model:

1. **Planning:** In this phase, the team defines the scope of the project, sets goals, and creates a roadmap for the development process. The team collaborates with the stakeholders to identify the requirements, prioritize them, and create a product backlog.
2. **Analysis:** In this phase, the team analyzes the requirements and creates user stories, which are short descriptions of the features and functionalities that the product should have. The team estimates the effort required for each user story and creates a sprint backlog, which is a list of user stories that will be addressed in the current sprint.
3. **Design:** In this phase, the team designs the architecture of the product, defines the technical requirements, and creates a plan for the implementation. The team collaborates with the stakeholders to ensure that the design meets their needs and expectations.
4. **Implementation:** In this phase, the team develops the product according to the plan and the sprint backlog. The team follows the Agile principles of collaboration, self-organization, and continuous improvement. The team delivers a working product at the end of each sprint, which is reviewed by the stakeholders.
5. **Testing:** In this phase, the team tests the product to ensure that it meets the quality standards and the user requirements. The team uses various testing

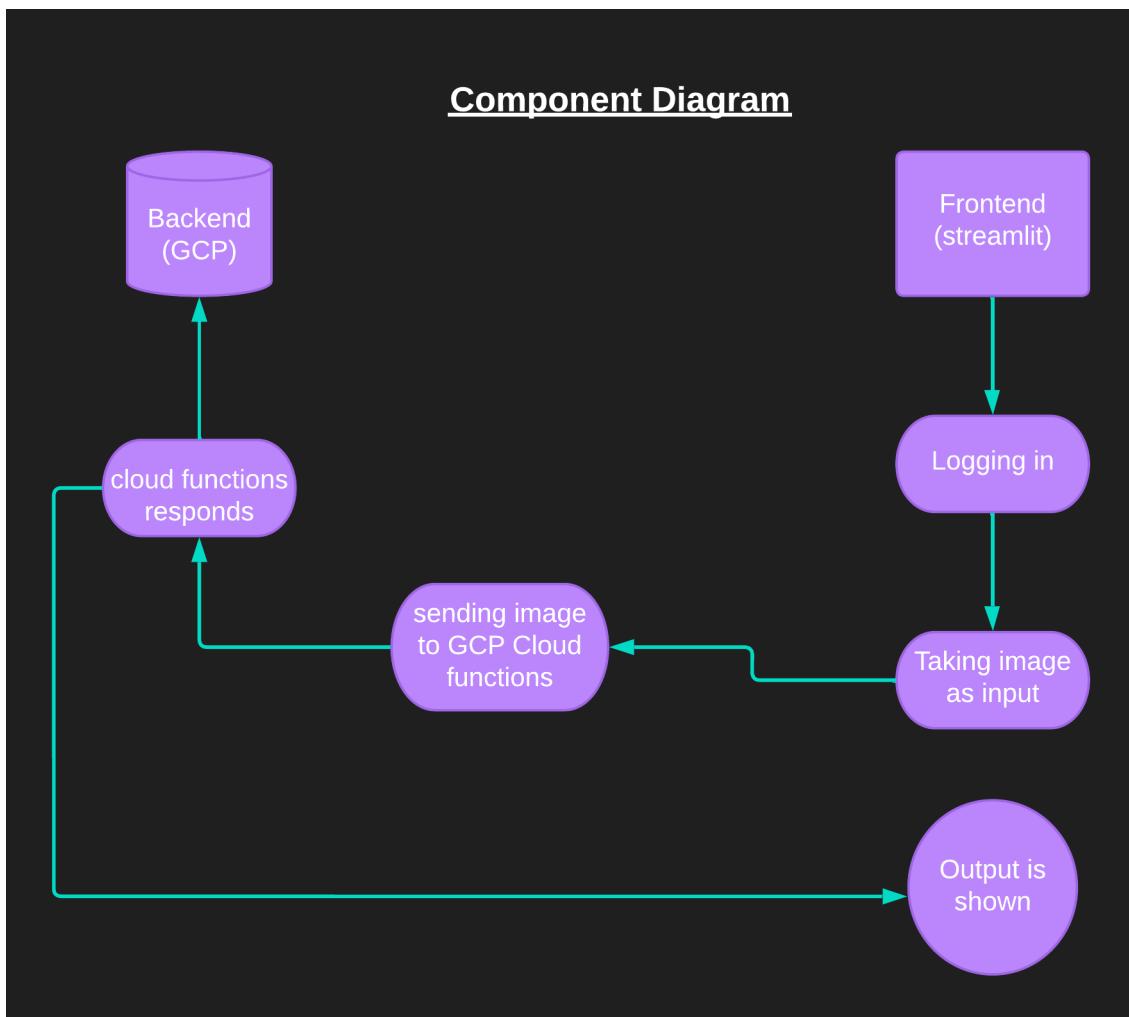
techniques, such as unit testing, integration testing, and acceptance testing, to identify and fix defects.

6. **Deployment:** In this phase, the team deploys the product to the production environment and makes it available to the users. The team monitors the product and collects feedback from the users to improve the product in future sprints.
7. **Review and Retrospective:** In this phase, the team reviews the sprint and the product with the stakeholders, demonstrates the progress made, and collects feedback. The team also conducts a retrospective, where they reflect on the sprint and identify opportunities for improvement. The team uses the feedback and the retrospective to plan the next sprint and improve the process.

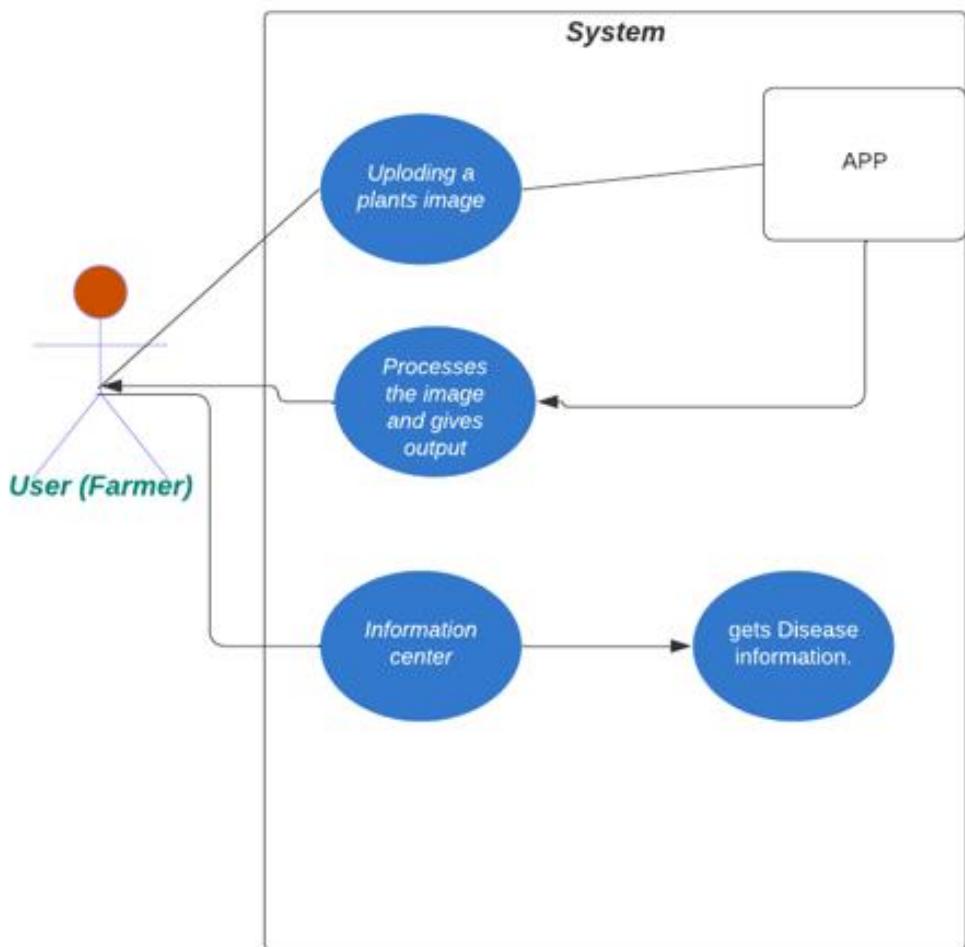
CHAPTER 3

SYSTEM DESIGN DETAILS/System Analysis

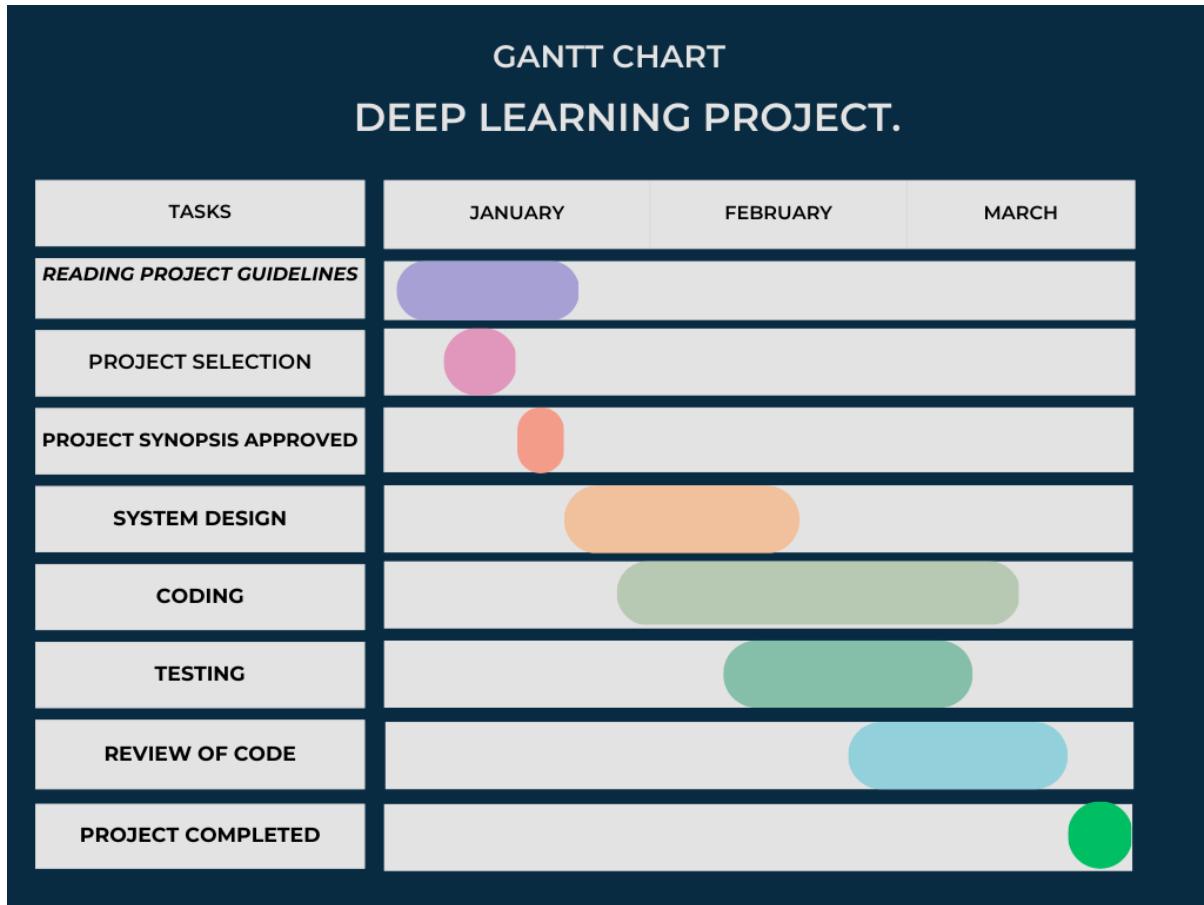
3.1 Component Diagram



3.2 Use Case Diagram



3.5 Gantt Chart



TASK	Start date	End date
Reading Project Guidelines	10/01/23	31/01/23
Project Selection	15/01/23	20/01/23
Creating Synopsis	20/01/23	22/01/23
Project Synopsis approved	25/01/23	
System Design	26/01/23	15/02/23
Requirement Analysis	16/02/23	17/02/23
Creating a Prototype	17/02/23	18/02/23
Coding	01/02/23	17/03/23

Testing	17/03/23	20/02/23
1 st review of code	28/03/23	
Worked on recommended things	28/03/23	29/03/23
2 nd review of code	30/03/23	
Project completed.	30/03/23	

CHAPTER 4

DATA DEFINITIONS AND FORM DESIGN/SNAPSHOTS

4.1 Cloud Storage:

The screenshot shows the Google Cloud Platform dashboard for a project named "Plant Disease Prediction".

- Project info:** Project name: Plant Disease Prediction, Project number: 561572815199, Project ID: light-ratio-381415.
- APIs & Services:** Shows the API section with no data available for the selected time frame.
- Resources:** Lists various services: BigQuery, SQL, Compute Engine, Storage, Cloud Functions, and App Engine.
- Billing:** Shows estimated charges for the billing period Mar 1 – 31, 2023, totaling INR ₹0.00.
- Monitoring:** Allows creating a dashboard, setting up alerting policies, and viewing uptime checks.

The screenshot shows the Google Cloud Storage browser interface.

	Name	Created	Location type	Location	Default storage class	Last modified	Public access
<input type="checkbox"/>	gmodel20	Mar 22, 2023, 8:42:55 PM	Region	asia-south1	Standard	Mar 22, 2023, 8:42:55 PM	Not public
<input type="checkbox"/>	gcf-sources-561572815199-us-central1	Mar 22, 2023, 9:37:55 PM	Region	us-central1	Standard	Mar 22, 2023, 9:37:55 PM	Not public
<input type="checkbox"/>	us.artifacts.light-ratio-381415.appspot.com	Mar 22, 2023, 9:38:18 PM	Multi-region	us	Standard	Mar 22, 2023, 9:38:18 PM	Subject to object AC

Google Cloud Storage Bucket details for **dlmodel20**

dlmodel20

Location	Storage class	Public access	Protection
asia-south1 (Mumbai)	Standard	Not public	None

OBJECTS **CONFIGURATION** **PERMISSIONS** **PROTECTION** **LIFECYCLE** **OBSERVABILITY**

Buckets > dlmodel20 > models

vg_final.h5

104.3 MB application/octet-stream Mar 22, 2023, 9:05:49 PM Standard Mar 22, 2023, 9:05:49 PM Not public

Google Cloud Functions

Functions

Environment	Name	Last deployed	Region	Recommendation	Trigger	Runtime	Memory allocated	Executed function	Actions
1st gen	predict	Mar 22, 2023, 9:53:51 PM	us-central1		HTTP	Python 3.8	1 GB	predict	⋮

The screenshot shows the 'Function details' page for a Google Cloud Function named 'predict'. The 'DETAILS' tab is selected. Key information includes:

- General Information:**
 - Last deployed: March 22, 2023 at 9:53:51 PM GMT+5
 - Region: us-central1
 - Memory allocated: 1 GB
 - Timeout: 60 seconds
 - Minimum instances: 0
 - Maximum instances: -
 - Service account: light-ratio-381415@appspot.gserviceaccount.com
 - Build Worker Pools: -
 - Container build log: eb9e7e99-4847-42a8-b649-97f12ce1ab52
- Networking Settings:**
 - Ingress settings: Allow all traffic
 - VPC connector: -
 - VPC connector egress: routing

Below the main sections, there is a 'EQUIVALENT REST' section which is currently empty.

The screenshot shows the 'Function details' page for the 'predict' function, with the 'SOURCE' tab selected. The code editor displays the following Python 3.8 code:

```

Runtime : Python 3.8      Entry point : predict      DOWNLOAD ZIP
main.py
requirements.txt
extra

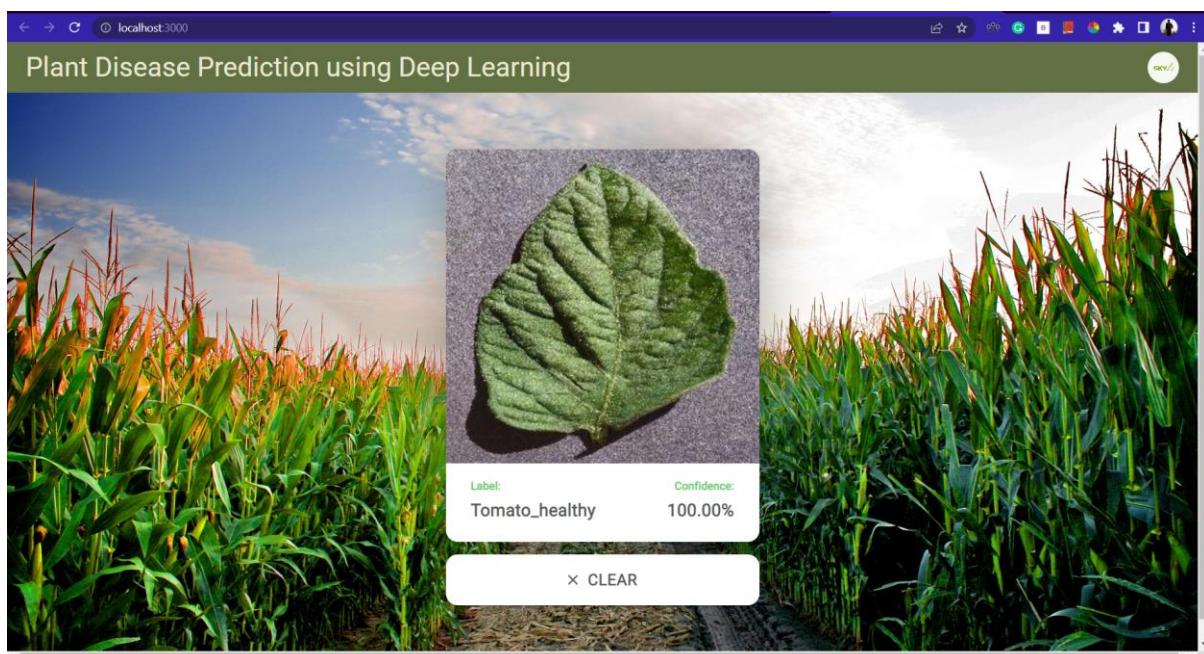
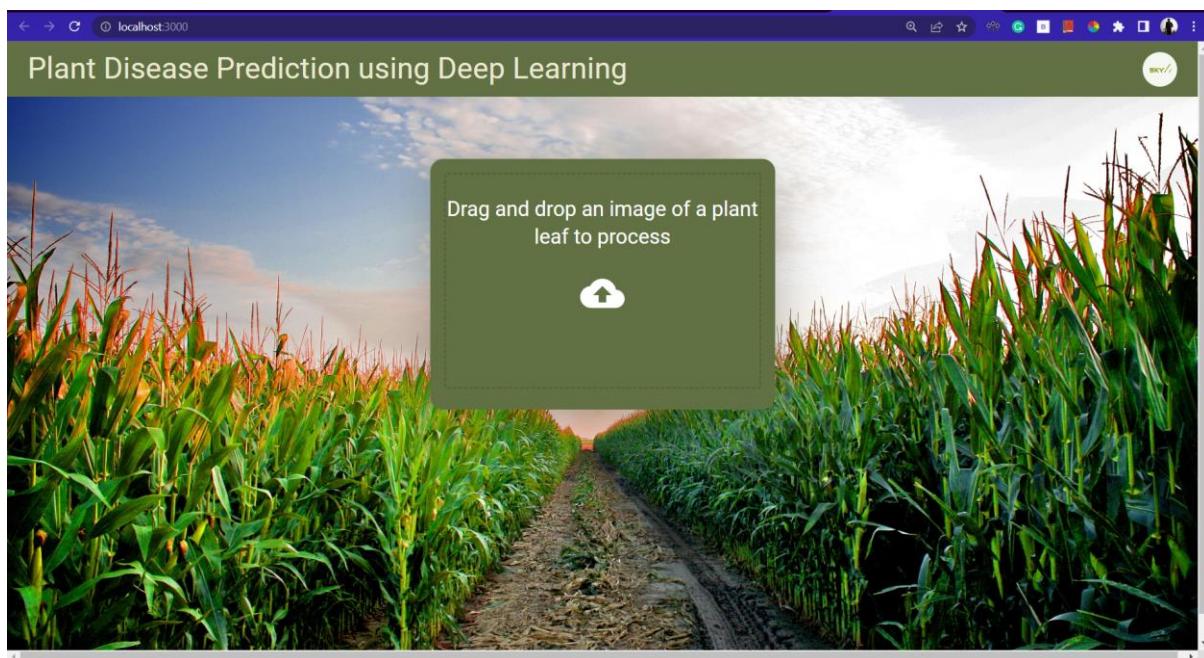
1  from google.cloud import storage
2  import tensorflow as tf
3  from PIL import Image
4  import numpy as np
5
6  model = None
7  interpreter = None
8  input_index = None
9  output_index = None
10
11 class_names = [
12     "Pepper_bell_Bacterial_spot",
13     "Pepper_bell_healthy",
14     "Potato_Early_blight",
15     "Potato_Late_blight",
16     "Potato_Bacterial_spot",
17     "Tomato_Early_blight",
18     "Tomato_Late_blight",
19     "Tomato_Mold",
20     "Tomato_Seporia_leaf_spot",
21     "Tomato_Spider_mites_Two_spotted_spider_mite",
22     "Tomato_Target_Spot",
23     "Tomato_Tomato_YellowLeaf_Curl_Virus",
24     "Tomato_Tomato_mosaic_virus",
25     "Tomato_healthy"
26
27 BUCKET_NAME = "dlmodel120" # Here you need to put the name of your GCP bucket
28
29
30 def download_file(bucket_name, source_file_name, destination_file_name):

```

4.2 Form Design /Snapshots

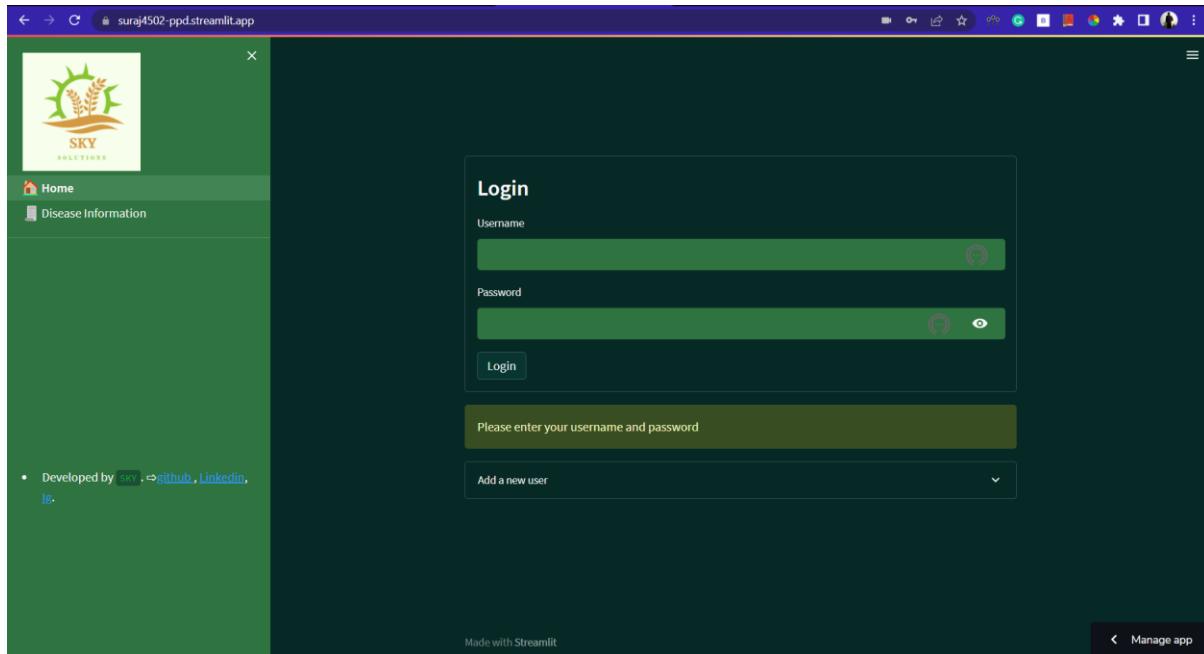
This section describes form those are used into the system.

4.2.1 React JS APP:

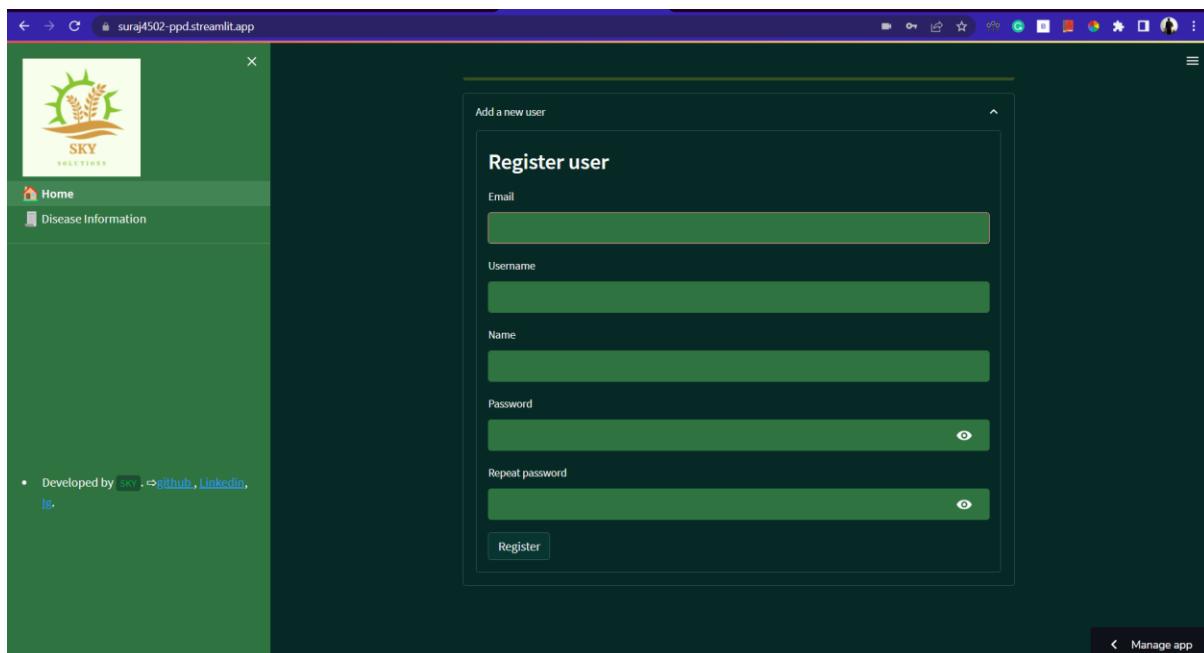


4.2.2 Streamlit APP:

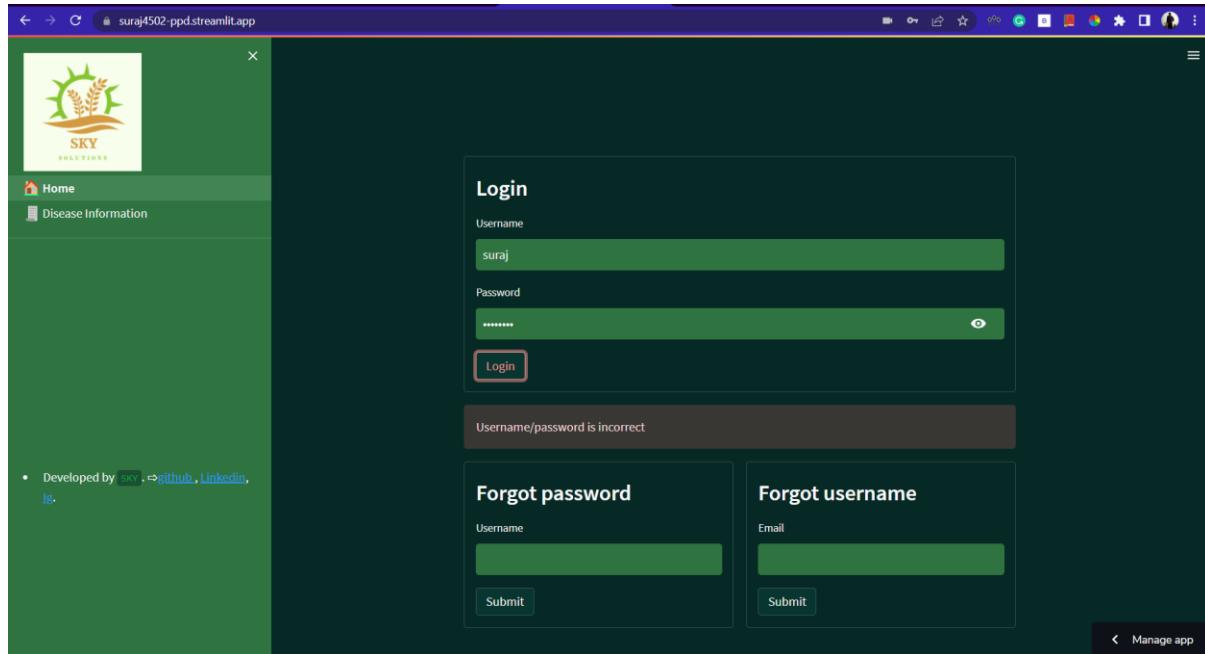
- Login page:



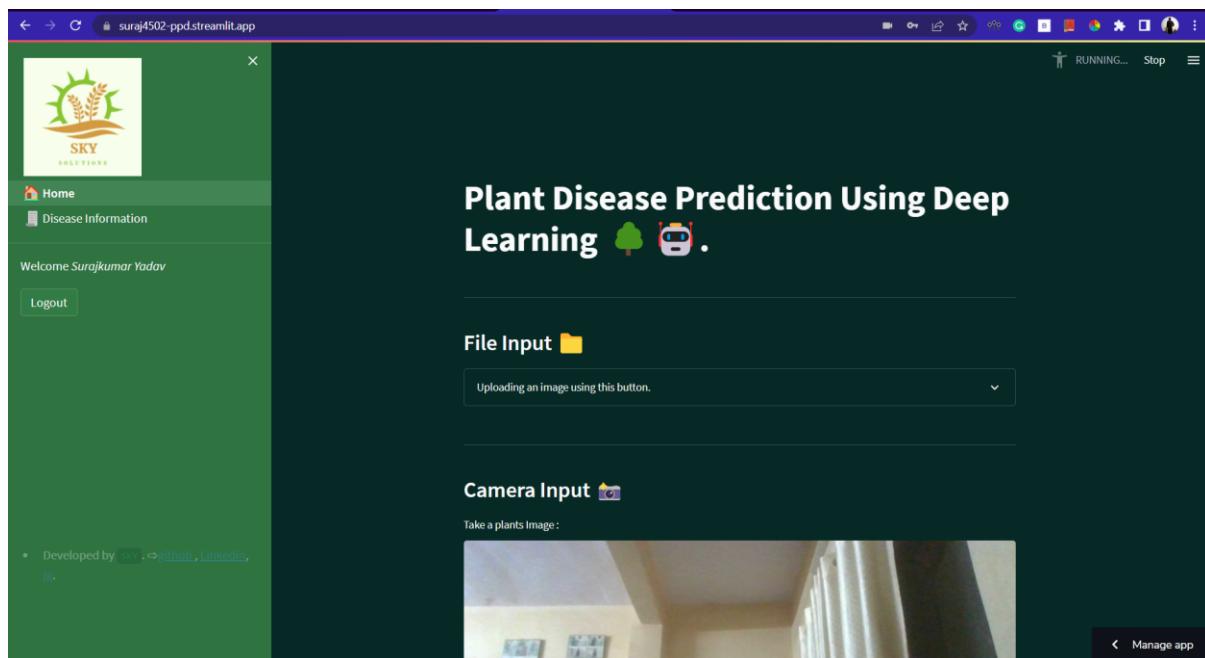
- Signing up:



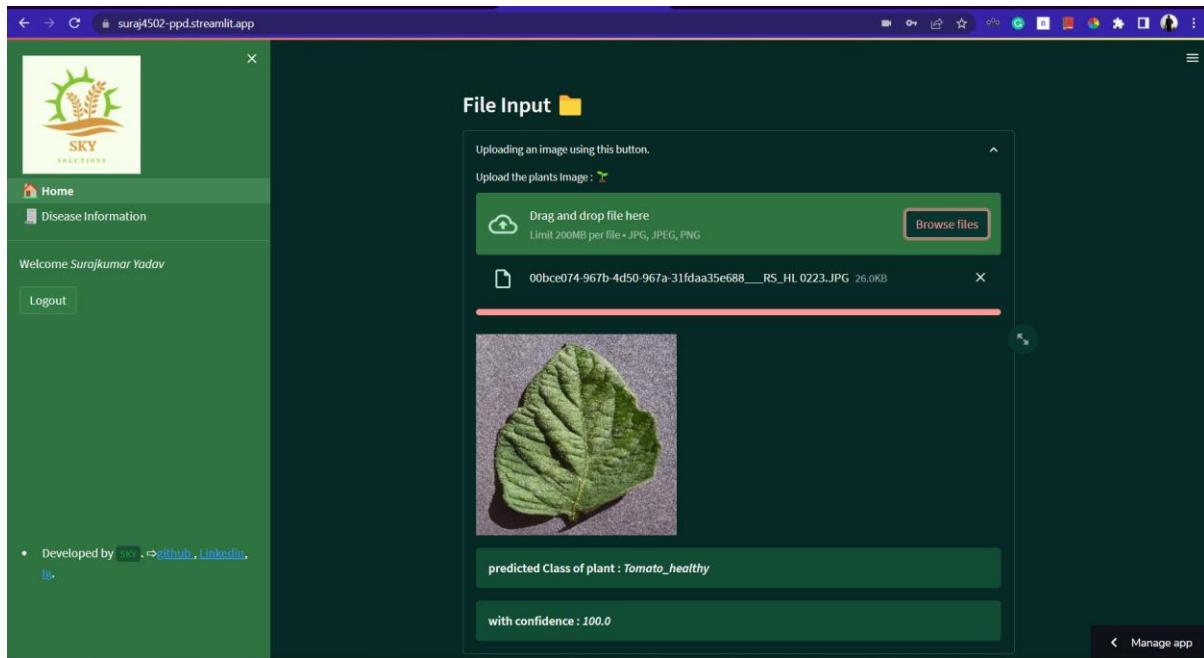
- **Retrieving Credentials:**



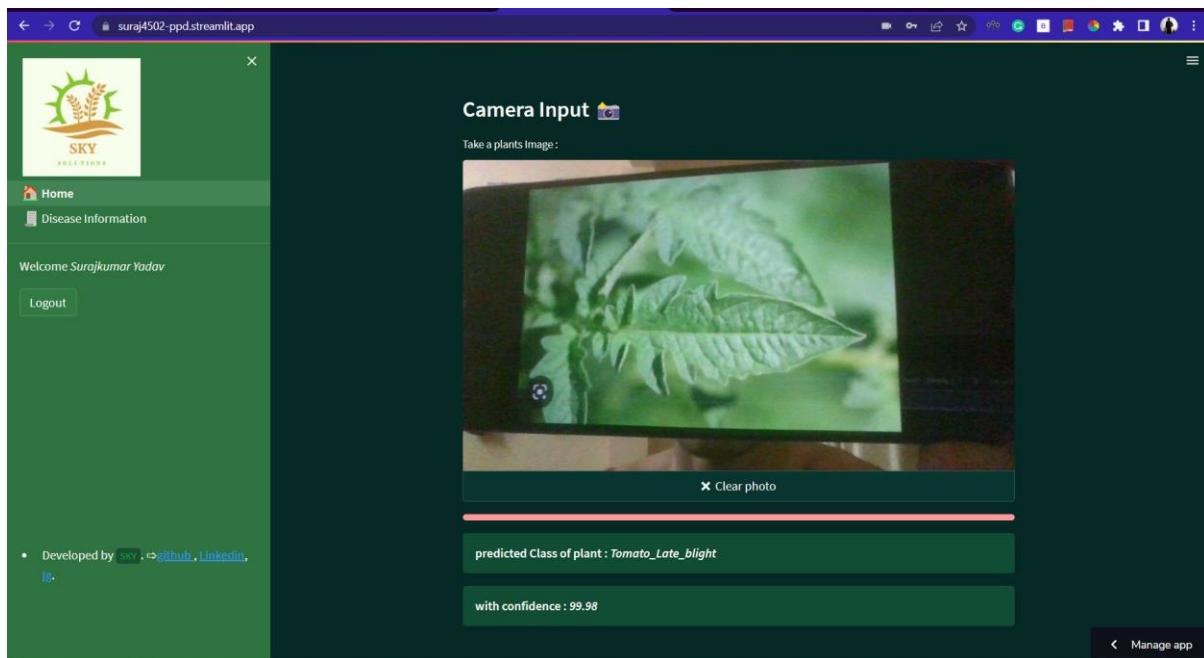
- **Main Section:**



- Uploading an image to get results:



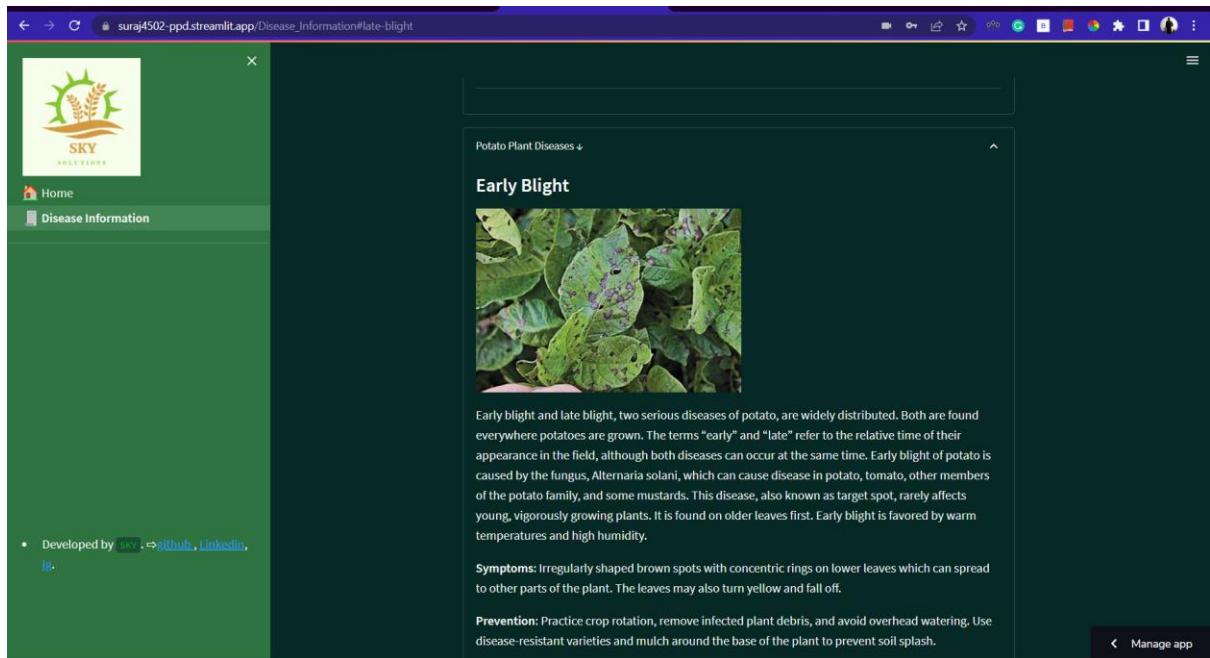
- Taking an image from camera to get results:



- Disease Information Page:

The screenshot shows a Streamlit application window titled "Disease Information". On the left, there is a sidebar with a logo for "SKY SOLUTIONS" featuring a stylized sun and plants, and two menu items: "Home" and "Disease Information". A red arrow points from the text "Disease Information Page:" to the "Disease Information" link in the sidebar. The main content area has a dark background and features a title "Common Plant Diseases and Prevention Methods". Below the title are three dropdown menus: "Tomato Plant Diseases", "Pepper Bell Plant Diseases", and "Potato Plant Diseases". At the bottom left of the main area, there is a note about development: "Developed by SKY, GitHub, LinkedIn, Ig.". At the bottom right, it says "Made with Streamlit" and "Manage app".

This screenshot shows the same Streamlit application window, but the "Pepper Bell Plant Diseases" dropdown is expanded to show the "Anthracnose" page. The title "Anthracnose" is displayed above a photograph of a green pepper with brown spots. Below the image, there is descriptive text: "Anthracnose, caused by species of the plant pathogenic fungus Colletotrichum, causes no obvious leaf lesions on tomato foliage and only occasionally on pepper foliage under high disease pressure. However, pepper and tomato fruits are very susceptible to the disease. The fungus can be introduced on seeds, and survives over the winter in temperate climates associated with crop debris." It also includes sections for "Symptoms" and "Prevention". The sidebar and footer are identical to the first screenshot.



• Mobile View:

Login

Username

Password

Login

Please enter your username and password

Add a new user

File Input

Uploading an image using this button.

Upload the plants Image :

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

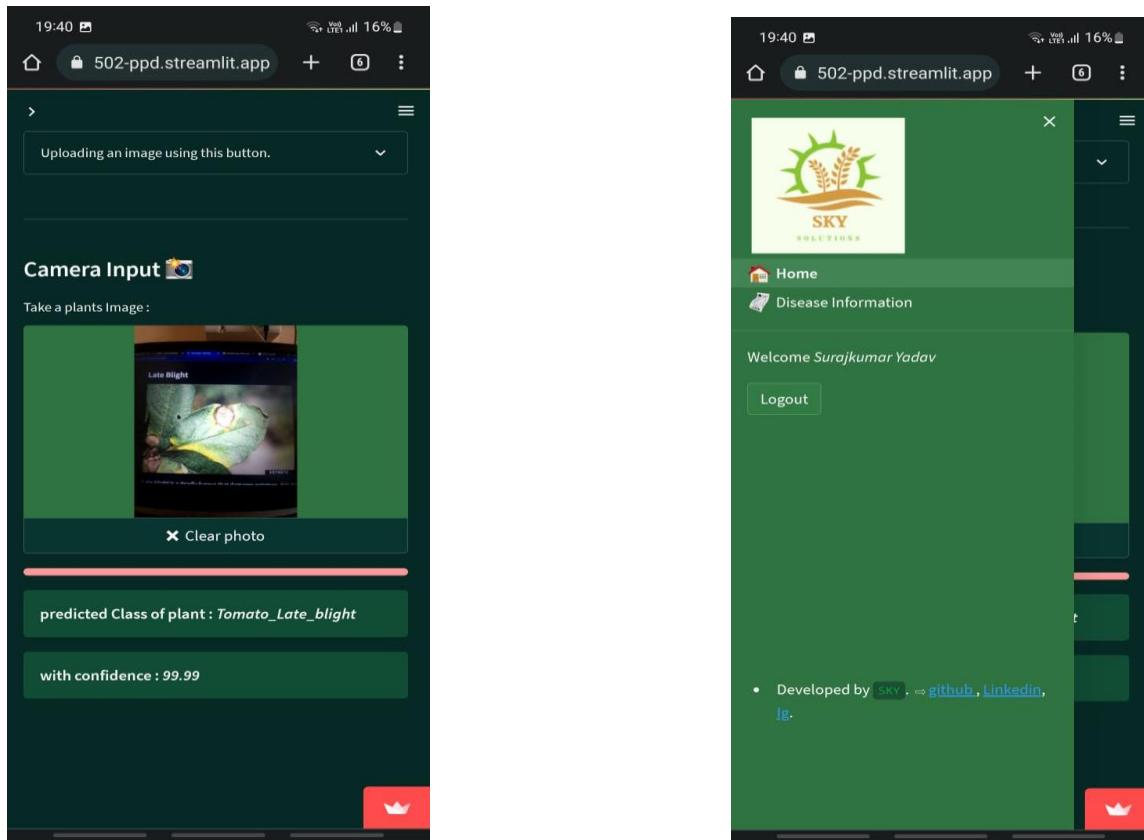
Browse files

SmartSelect_20230331-193008_... X
347.0KB



predicted Class of plant : *Tomato_Late_blight*

with confidence : 99.65



19:41 502-ppd.streamlit.app

Common Plant Diseases and Prevention Methods

Tamto Plant Diseases

Pepper Bell Plant Diseases

Potato Plant Diseases

Early Blight

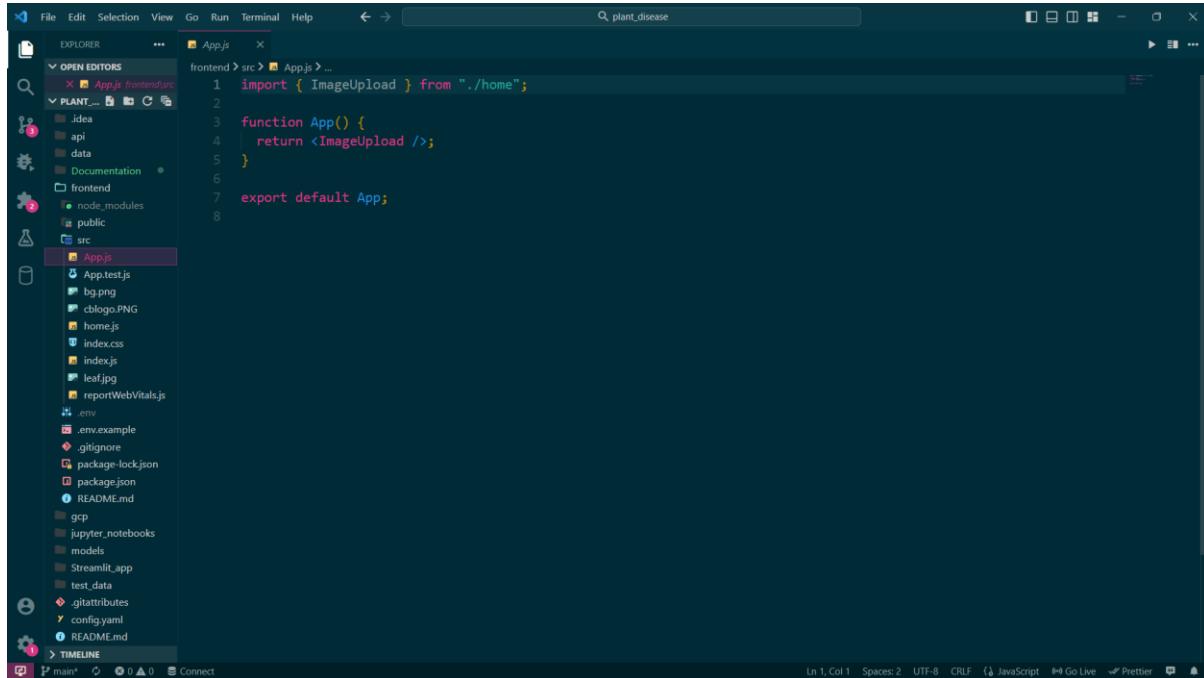


Early blight and late blight, two serious diseases of potato, are widely distributed. Both are found everywhere potatoes are grown. The terms "early" and "late" refer to the relative time of their

CHAPTER 5

SYSTEM CODE & FUTURE SCOPE

5.1 React JS app:

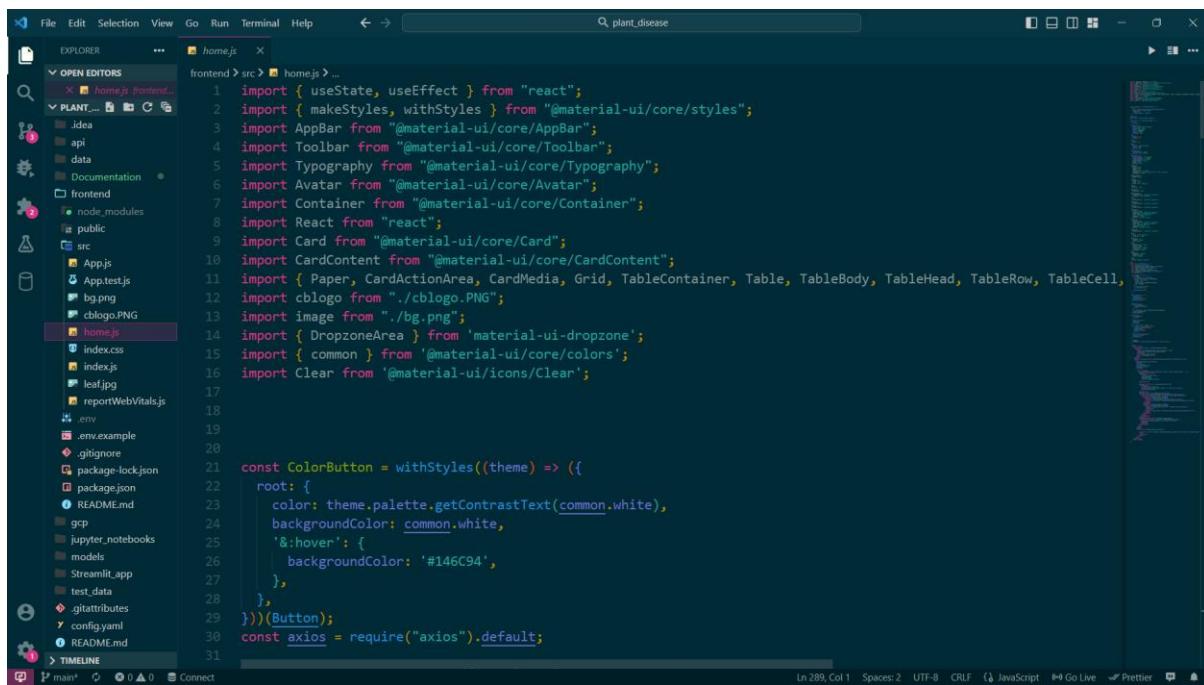


```

File Edit Selection View Go Run Terminal Help < > plant_disease
OPEN EDITORS App.js
frontend > src > App.js > ...
1 import { ImageUpload } from "./home";
2
3 function App() {
4   return <ImageUpload />;
5 }
6
7 export default App;
8

```

The screenshot shows the VS Code interface with the 'App.js' file open in the editor. The code defines a simple functional component named 'App' that returns a single 'ImageUpload' component from the 'home' module.



```

File Edit Selection View Go Run Terminal Help < > plant_disease
OPEN EDITORS home.js
frontend > src > home.js > ...
1 import { useState, useEffect } from "react";
2 import { makeStyles, withStyles } from "@material-ui/core/styles";
3 import AppBar from "@material-ui/core/AppBar";
4 import Toolbar from "@material-ui/core/Toolbar";
5 import Typography from "@material-ui/core/Typography";
6 import Avatar from "@material-ui/core/Avatar";
7 import Container from "@material-ui/core/Container";
8 import React from "react";
9 import Card from "@material-ui/core/Card";
10 import CardContent from "@material-ui/core/CardContent";
11 import { Paper, CardActionArea, CardMedia, Grid, TableContainer, Table, TableBody, TableHead, TableRow, TableCell, clogo } from "@material-ui/core";
12 import cblogo from "./cblogo.PNG";
13 import image from "./bg.png";
14 import { DropzoneArea } from 'material-ui-dropzone';
15 import { common } from '@material-ui/core/colors';
16 import Clear from '@material-ui/icons/Clear';
17
18
19
20
21 const ColorButton = withStyles((theme) => ({
22   root: {
23     color: theme.palette.getContrastText(common.white),
24     backgroundColor: common.white,
25     '&:hover': {
26       backgroundColor: '#146C94',
27     },
28   },
29 }))(Button);
30 const axios = require("axios").default;
31

```

The screenshot shows the 'home.js' file open in the VS Code editor. This file imports various Material-UI components and styles. It defines a 'ColorButton' component using the 'withStyles' HOC and sets up an 'axios' instance at the bottom.

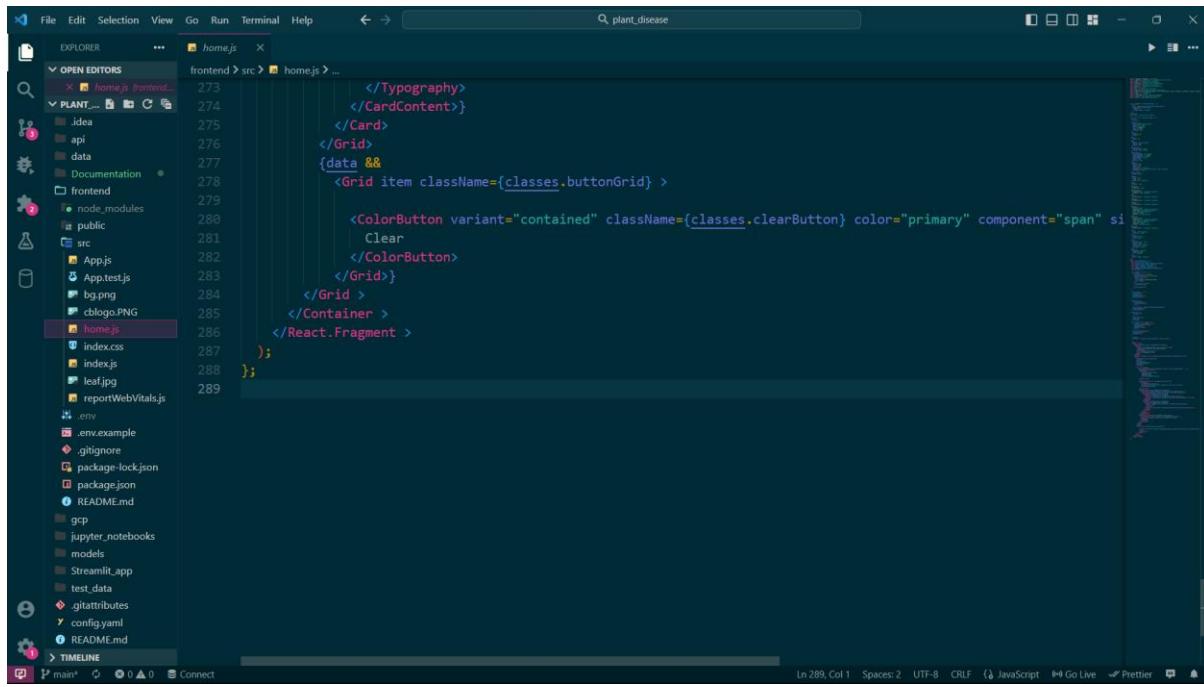
```

    return (
      <React.Fragment>
        <AppBar position="static" className={classes.appbar}>
          <Toolbar>
            <Typography variant="h4" noWrap>
              Plant Disease Prediction using Deep Learning
            </Typography>
            <div className={classes.grow} />
            <Avatar src={clogo}></Avatar>
          </Toolbar>
        </AppBar>
        <Container maxWidth={false} className={classes.mainContainer} disableGutters={true}>
          <Grid
            className={classes.gridContainer}
            container
            direction="row"
            justifyContent="center"
            alignItems="center"
            spacing={2}
          >
            <Grid item xs={12}>
              <Card className={`${classes.imageCard} ${!image ? classes.imageCardEmpty : ''}`}>
                {image && <CardActionArea>
                  <CardMedia
                    className={classes.media}
                    image={preview}
                    component="image"
                    title="Contemplative Reptile"
                  />
                </CardActionArea>
              </Card>
            </Grid item>
          </Grid>
        </Container>
      </React.Fragment>
    )
  
```

```

    <DropzoneArea
      acceptedFiles={['image/*']}
      dropzoneText="Drag and drop an image of a plant leaf to process"
      onChange={onSelectFile}
    />
  </CardContent>
  {data && <CardContent className={classes.detail}>
    <TableContainer component={Paper} className={classes.tableContainer}>
      <Table className={classes.table} size="small" aria-label="simple table">
        <TableHead className={classes.tableHead}>
          <TableRow className={classes.tableRow}>
            <TableCell className={classes.tableCell}>Label:</TableCell>
            <TableCell align="right" className={classes.tableCell}>Confidence:</TableCell>
          </TableRow>
        </TableHead>
        <TableBody className={classes.tableBody}>
          <TableRow className={classes.tableRow}>
            <TableCell component="th" scope="row" className={classes.tableCell}>
              {data.class}
            </TableCell>
            <TableCell align="right" className={classes.tableCell}>{confidence}%</TableCell>
          </TableRow>
        </TableBody>
      </TableContainer>
    </CardContent>
  <isLoading && <CardContent className={classes.detail}>
    <CircularProgress color="secondary" className={classes.loader} />
    <Typography variant="h6" noWrap>
      Processing...
    </Typography>
  </CardContent>

```



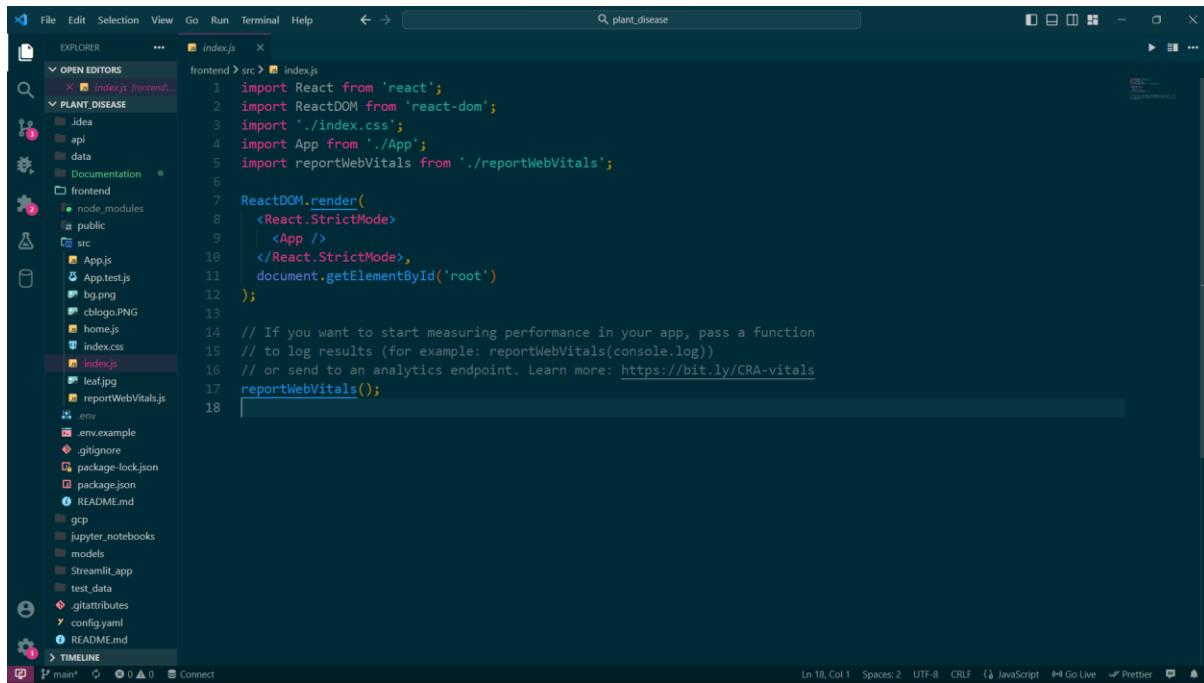
The screenshot shows the VS Code interface with the following details:

- File Path:** frontend > src > home.js
- Code Content (home.js):**

```

    273     </Typography>
    274     </CardContent>
    275   </Card>
    276   </Grid>
    277   {data &&
    278     <Grid item className={classes.buttonGrid} >
    279       <ColorButton variant="contained" className={classes.clearButton} color="primary" component="span" size="large" onClick={handleClear}>
    280         Clear
    281       </ColorButton>
    282     </Grid>
    283   </Container>
    284   </React.Fragment>
    285 };
    286 };
    287 };
    288 };
  
```

- Status Bar:** Ln 289, Col 1 Spaces:2 UTF-8 CRLF ⓘ JavaScript ⓘ Go Live ⓘ Prettier



The screenshot shows the VS Code interface with the following details:

- File Path:** frontend > src > index.js
- Code Content (index.js):**

```

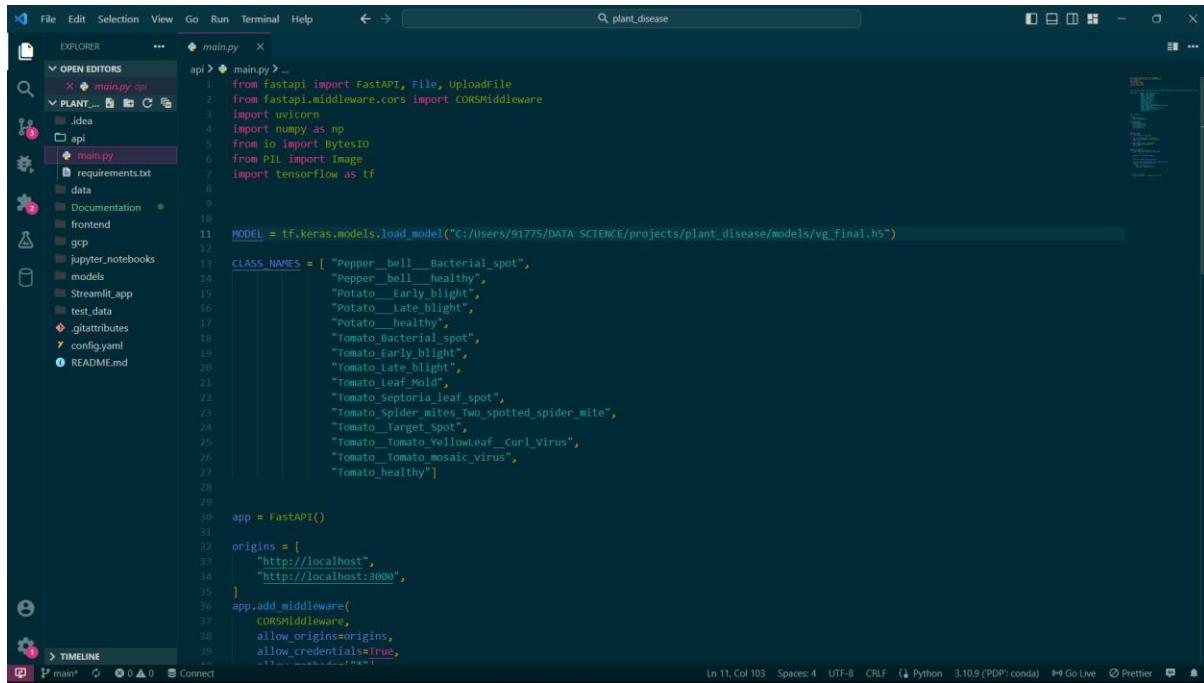
  1 import React from 'react';
  2 import ReactDOM from 'react-dom';
  3 import './index.css';
  4 import App from './App';
  5 import reportWebVitals from './reportWebVitals';

  6 ReactDOM.render(
  7   <React.StrictMode>
  8     <App />
  9   </React.StrictMode>,
 10   document.getElementById('root')
 11 );
 12 );

  13 // If you want to start measuring performance in your app, pass a function
 14 // to log results (for example: reportWebVitals(console.log))
 15 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
 16 // reportWebVitals();
 17 reportWebVitals();
 18
  
```

- Status Bar:** Ln 18, Col 1 Spaces:2 UTF-8 CRLF ⓘ JavaScript ⓘ Go Live ⓘ Prettier

5.2 Backend code (Fast API):

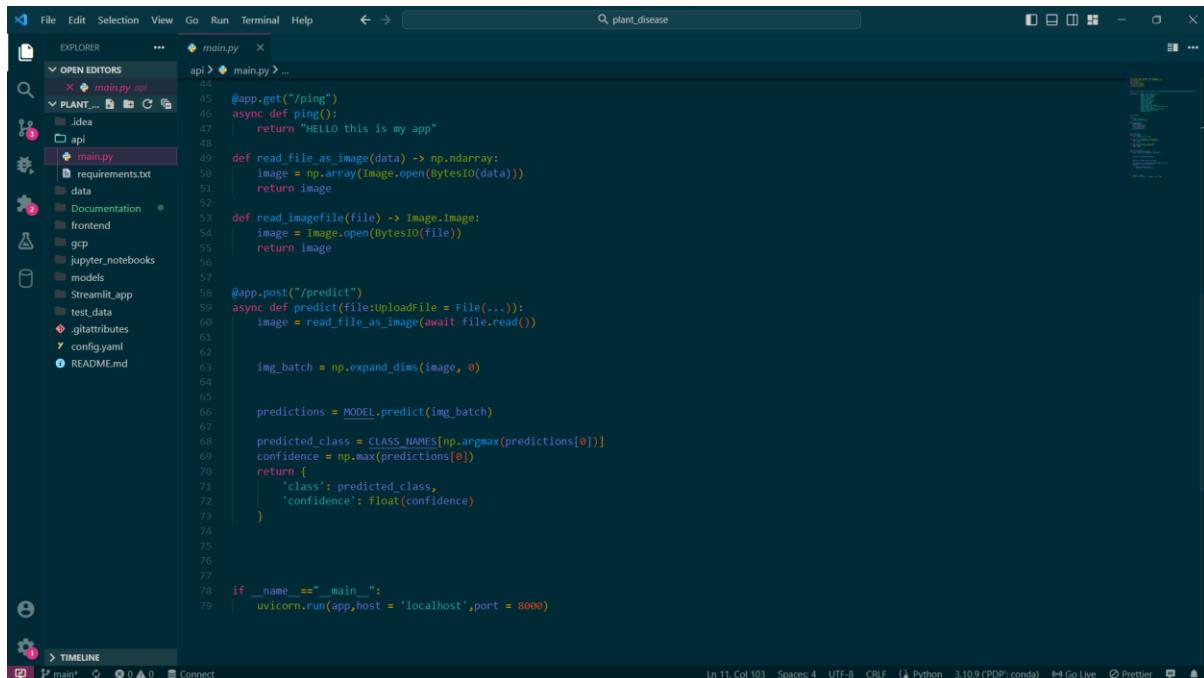


```

File Edit Selection View Go Run Terminal Help < > plant_disease
OPEN EDITORS main.py ...
api > main.py > ...
1 from fastapi import FastAPI, File, UploadFile
2 from fastapi.middleware.cors import CORSMiddleware
3 import uicorn
4 import numpy as np
5 from io import BytesIO
6 from PIL import Image
7 import tensorflow as tf
8
9
10 MODEL = tf.keras.models.load_model("C:/Users/91775/DATA SCIENCE/projects/plant_disease/models/vg_Final.h5")
11
12 CLASS_NAMES = [
13     "Pepper_bell_Bacterial_spot",
14     "Pepper_bell_healthy",
15     "Potato_Early_blight",
16     "Potato_healthy",
17     "Tomato_Bacterial_spot",
18     "Tomato_Early_blight",
19     "Tomato_late_blight",
20     "Tomato_Leaf_Mold",
21     "Tomato_Septoria_leaf_spot",
22     "Tomato_Spider_mites_Two_spotted_spider_mite",
23     "Tomato_target_Spot",
24     "Tomato_tomato_Yellowleaf_Curl_virus",
25     "Tomato_tomato_mosaic_virus",
26     "Tomato_healthy"
27 ]
28
29
30 app = FastAPI()
31
32 origins = [
33     "http://localhost",
34     "http://localhost:3000",
35 ]
36 app.add_middleware(
37     CORSMiddleware,
38     allow_origins=origins,
39     allow_credentials=True,
40 )

```

Ln 11, Col 103 Spaces: 4 UTF-8 CRLF ⓘ Python 3.10.9 (PDP:conda) ⓘ Go Live ⓘ Prettier



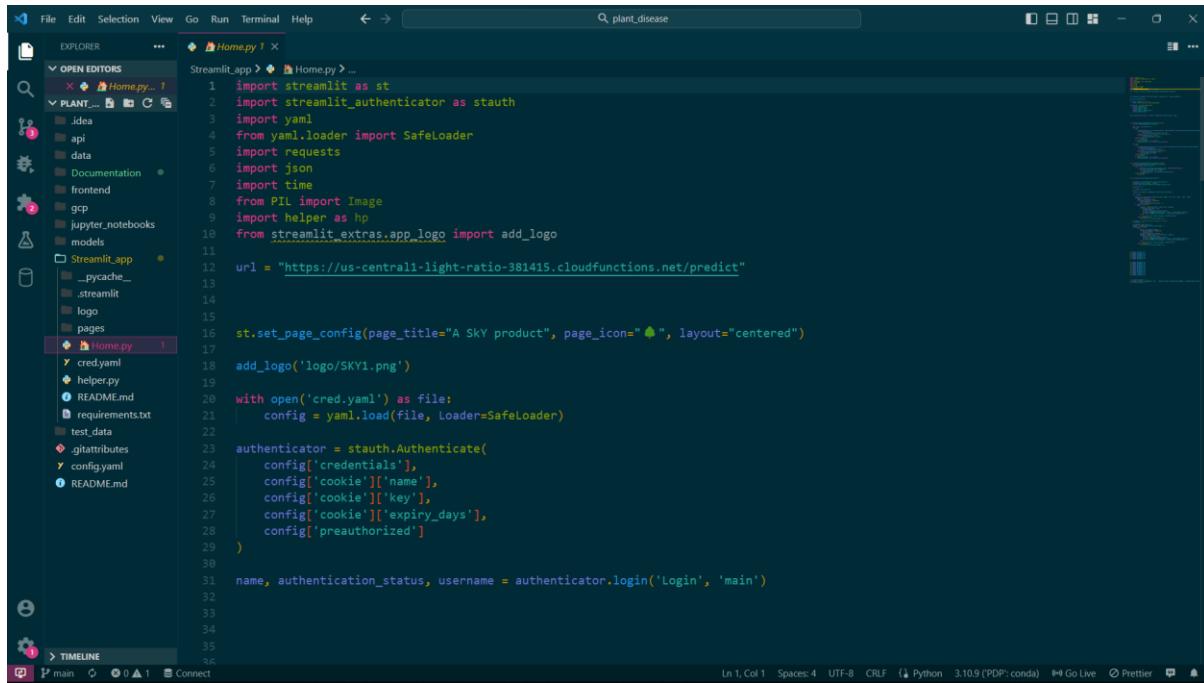
```

File Edit Selection View Go Run Terminal Help < > plant_disease
OPEN EDITORS main.py ...
api > main.py > ...
44 @app.get("/")
45 async def ping():
46     return "HELLO this is my app"
47
48 def read_file_as_image(data) -> np.ndarray:
49     image = np.array(Image.open(BytesIO(data)))
50     return image
51
52 def read_imagefile(file) -> Image.Image:
53     image = Image.open(BytesIO(file))
54     return image
55
56
57 @app.post("/predict")
58 async def predict(file:UploadFile = File(...)):
59     image = read_file_as_image(await file.read())
60
61     img_batch = np.expand_dims(image, 0)
62
63     predictions = MODEL.predict(img_batch)
64
65     predicted_class = CLASS_NAMES[np.argmax(predictions[0])]
66     confidence = np.max(predictions[0])
67     return {
68         'class': predicted_class,
69         'confidence': float(confidence)
70     }
71
72
73
74
75
76
77
78 if __name__ == "__main__":
79     uvicorn.run(app,host = 'localhost',port = 8000)

```

Ln 11, Col 103 Spaces: 4 UTF-8 CRLF ⓘ Python 3.10.9 (PDP:conda) ⓘ Go Live ⓘ Prettier

5.3 Streamlit App :

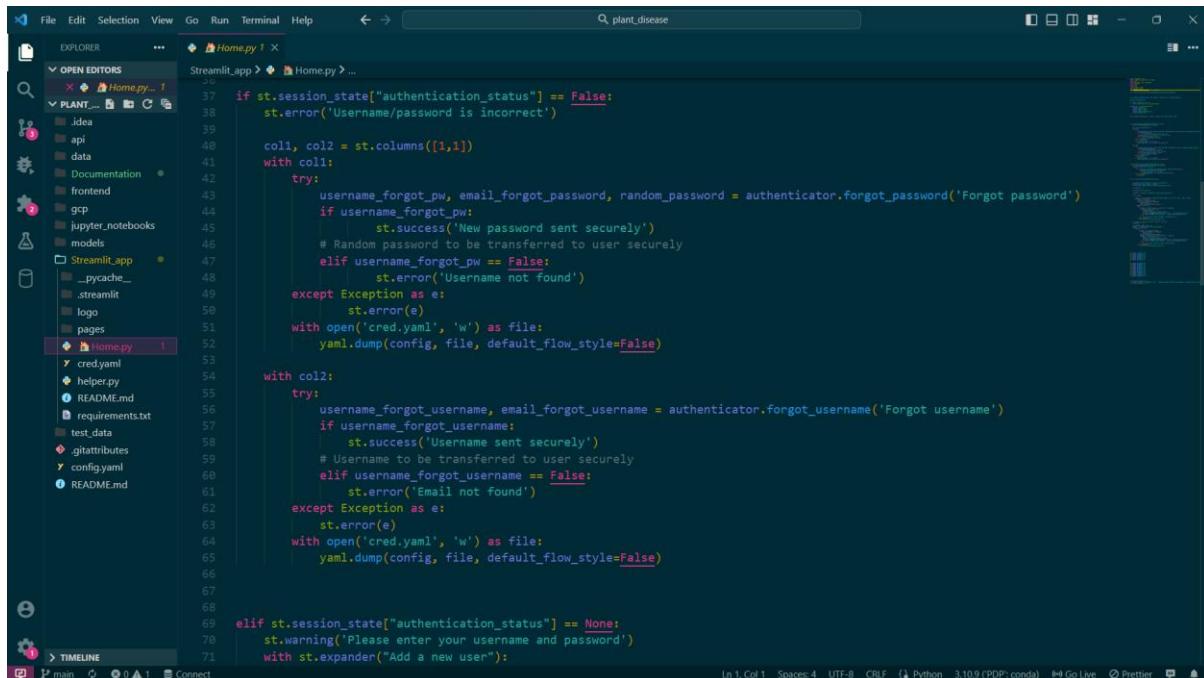


```

File Edit Selection View Go Run Terminal Help < > plant_disease
OPEN EDITORS Home.py ...
PLANT... Home.py ...
.idea api data Documentation frontend gcp jupyter_notebooks models Streamlit_app _pycache_ .streamlit logo pages
Streamlit_app > Home.py
1 import streamlit as st
2 import streamlit_authenticator as stauth
3 import yaml
4 from yamlloader import SafeLoader
5 import requests
6 import json
7 import time
8 from PIL import Image
9 import Helper as hp
10 from streamlit_extras.app_logo import add_logo
11
12 url = "https://us-central1-light-ratio-381415.cloudfunctions.net/predict"
13
14
15 st.set_page_config(page_title="A SKY product", page_icon="♣", layout="centered")
16 add_logo("logo/SKY1.png")
17
18 with open('cred.yaml') as file:
19     config = yaml.load(file, Loader=SafeLoader)
20
21 authenticator = stauth.Authenticate(
22     config['credentials'],
23     config['cookie']['name'],
24     config['cookie']['key'],
25     config['cookie']['expiry_days'],
26     config['preauthorized']
27 )
28
29
30 name, authentication_status, username = authenticator.login('Login', 'main')
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

This screenshot shows the initial state of the Home.py file in a Streamlit application. The code imports Streamlit and Streamlit Authenticator, sets the page configuration, adds a logo, and initializes the authenticator with credentials from a YAML file.



```

if st.session_state["authentication_status"] == False:
    st.error('Username/password is incorrect')

    col1, col2 = st.columns([1,1])
    with col1:
        try:
            username_forgot_pw, email_forgot_password, random_password = authenticator.forgot_password('Forgot password')
            if username_forgot_pw:
                st.success('New password sent securely')
            # Random password to be transferred to user securely
        elif username_forgot_pw == False:
            st.error('Username not found')
        except Exception as e:
            st.error(e)

    with open('cred.yaml', 'w') as file:
        yaml.dump(config, file, default_flow_style=False)

    with col2:
        try:
            username_forgot_username, email_forgot_username = authenticator.forgot_username('Forgot username')
            if username_forgot_username:
                st.success('Username sent securely')
            # Username to be transferred to user securely
        elif username_forgot_username == False:
            st.error('Email not found')
        except Exception as e:
            st.error(e)

    with open('cred.yaml', 'w') as file:
        yaml.dump(config, file, default_flow_style=False)

elif st.session_state["authentication_status"] == None:
    st.warning('Please enter your username and password')
    with st.expander("Add a new user"):

```

This screenshot shows the Home.py file with expanded logic for handling different session states. It includes code for forgot password and forgot username functionality, and a warning message for users who have not yet logged in.

```

File Edit Selection View Go Run Terminal Help < > plant_disease
OPEN EDITORS Streamlit_app > Home.py ...
PLANT... Home.py ...
.idea 72
.api 73
.data 74
Documentation 75
.frontend 76
.gcp 77
.jupyter_notebooks 78
.models 79
Streamlit_app 80
._pycache_ 81
._streamlit 82
.logo 83
.pages 84
Home.py 85
credyaml 86
helper.py 87
README.md 88
requirements.txt 89
test_data 90
.gitattributes 91
config.yaml 92
README.md 93
 README.md 94
 README.md 95
 README.md 96
 README.md 97
 README.md 98
 README.md 99
 README.md 100
 README.md 101
 README.md 102
 README.md 103
 README.md 104
 README.md 105
 README.md 106
 README.md 107
 README.md 108
 README.md 109
 README.md 110
 README.md 111
 README.md 112
 README.md 113
 README.md 114
 README.md 115
 README.md 116
 README.md 117
 README.md 118
 README.md 119
 README.md 120
 README.md 121
 README.md 122
 README.md 123
 README.md 124
 README.md 125
 README.md 126
 README.md 127
 README.md 128
 README.md 129
 README.md 130
 README.md 131
 README.md 132
 README.md 133
 README.md 134
 README.md 135

```

The screenshot shows a Jupyter Notebook interface with the file `Home.py` open. The code implements a function to register a user and another to handle file uploads. It includes logic to upload images to a server and process the responses to identify plant diseases.

```

try:
    if authenticator.register_user('Register user', preauthorization=False):
        st.success('User registered successfully')
        with open('cred.yaml', 'w') as file:
            yaml.dump(config, file, default_flow_style=False)
except Exception as e:
    st.error(e)

elif st.session_state["authentication_status"]:
    st.sidebar.write(f'Welcome {st.session_state["name"]}')
    authenticator.logout('Logout', 'sidebar')
    st.title('Plant Disease Prediction Using Deep Learning 🌱🤖')
    st.markdown("----")
    st.subheader("File Input 📄")

    f_expand = st.expander(" Uploading an image using this button.")

    with f_expand:
        image_1 = st.file_uploader("Upload the plants Image : 📸", type = ['jpg', 'jpeg', 'png'])
        if image_1 is not None:
            my_bar = st.progress(0)
            for percent_complete in range(100):
                time.sleep(0.001)
                my_bar.progress(percent_complete + 1)
        if image_1 is not None:
            try:
                response = requests.post(url, files={'file': image_1})
                if response.status_code == 200:
                    st.image(image_1)
                    json_obj = json.loads(response.content)
                    st.success(f'##### predicted Class of plant : {json_obj["class"]}')
                    st.success(f'##### with confidence : {json_obj["confidence"]}')
            except:
                st.error(f'Error processing image : {response.text}')
        except Exception as e:
            st.error(f'Error processing image: {e}')

# c_expand = st.expander("Taking an image of the plant")
st.markdown("----")
st.subheader("Camera Input 📸")
image_1 = st.camera_input("Take a plants Image :")
if image_1 is not None:
    try:
        my_bar = st.progress(value=0)
        for percent_complete in range(100):
            time.sleep(0.01)
        my_bar.progress(percent_complete + 1)
    response = requests.post(url, files={'file': image_1})
    if response.status_code == 200:
        json_obj = json.loads(response.content)
        st.success(f'##### predicted Class of plant : {json_obj["class"]}')
        st.success(f'##### with confidence : {json_obj["confidence"]}')
    else:
        st.error(f'Error processing image: {response.text}')
    except Exception as e:
        st.error(f'Error processing image: {e}')

```

```

File Edit Selection View Go Run Terminal Help < > plant_disease
OPEN EDITORS Streamlit_app > Home.py ...
PLANT... Home.py ...
.idea 100
.api 101
.data 102
Documentation 103
.frontend 104
.gcp 105
.jupyter_notebooks 106
.models 107
Streamlit_app 108
._pycache_ 109
._streamlit 110
.logo 111
.pages 112
Home.py 113
credyaml 114
helper.py 115
README.md 116
requirements.txt 117
test_data 118
.gitattributes 119
config.yaml 120
README.md 121
 README.md 122
 README.md 123
 README.md 124
 README.md 125
 README.md 126
 README.md 127
 README.md 128
 README.md 129
 README.md 130
 README.md 131
 README.md 132
 README.md 133
 README.md 134
 README.md 135

```

This screenshot shows the same `Home.py` code as the previous one, but with additional code added below it. This new code handles camera input, allowing users to take a photo of a plant and get a prediction.

The screenshot shows a Streamlit application interface. The left sidebar lists files and folders: 'OPEN EDITORS' (Disease_Information.py), 'PLANT...' (Streamlit_app, __pycache__, .streamlit, logo, pages), and 'DOCUMENTATION' (frontend, gcp, jupyter_notebooks, models). The main area displays a Python script for 'Disease_Information.py'. The script imports Streamlit and adds a logo. It sets the page title to 'Common Plant Diseases and Prevention Methods'. A section titled 'Early Blight' provides a definition and a link to a Bing search result. Another section titled 'Symptoms*' describes irregular brown spots with concentric rings. A 'Prevention*' section suggests crop rotation and overhead watering. A final section titled 'Late Blight' also includes a link to a Bing search result.

```
Disease_Information.py | X
Streamlit_app > pages > Disease_Information.py ...
1 import streamlit as st
2 from streamlit_extras.app_logo import add_logo
3
4 st.set_page_config(page_title="Description", page_icon="😊", layout="centered")
5 add_logo('logo/SKYI.png')
6
7 st.title("Common Plant Diseases and Prevention Methods")
8
9 texp = st.expander(" Tamoto Plant Diseases !")
10 with texp:
11
12     st.markdown("""
13         ***
14         #### Early Blight
15         |[Early Blight](https://th.bing.com/th/id/OIP.BFGZAKRWUUGIE1A9OKg7cgDXEh?pid=ImgDet&rs=1)
16
17             Early blight is a common tomato disease caused by the fungus Alternaria solani. It can affect almost all parts of the
18             tomato plants, including the leaves, stems, and fruits. The plants may not die, but they will be weakened and will set
19             fewer tomatoes than normal. Early blight generally attacks older plants, but it can also occur on seedlings. Stressed
20             plants or plants in poor health are especially susceptible. Early blight is also a problem with potatoes.
21
22             **Symptoms**: Irregularly shaped brown spots with concentric rings on lower leaves which can spread to other parts of the
23             plant. The leaves may also turn yellow and fall off.
24
25             **Prevention**: Practice crop rotation, remove infected plant debris, and avoid overhead watering. Use disease-resistant
26             varieties and mulch around the base of the plant to prevent soil splash.
27
28             ***
29             #### Late Blight
30
31         |[Late Blight](https://th.bing.com/th/id/OIP.vyLKEPBH3Rtc78sTNwsq0wHaE7?v=256&h=180&c=7&r=0&o=5&dpr=1.3&pid=1.7)
```

The screenshot shows a Streamlit application development environment. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The search bar contains the text "plant_disease". The left sidebar, titled "EXPLORER", displays the project structure:

- OPEN EDITORS:
 - Streamlit_app > helper.py
- PLANT_DISEASE:
 - idea
 - api
 - data
 - Documentation
 - frontend
 - gcp
 - jupyter_notebooks
 - models
- Streamlit_app:
 - _pycache_
 - .streamlit
 - logo
- pages:
 - Disease_Inform...
 - Home.py
 - cred.yaml
 - helper.py
 - README.md
 - requirements.txt
- test_data
- .gitattributes
- config.yaml
- README.md

The "helper.py" file in the "pages" directory is open in the code editor. The code defines a function to resize a logo image:

```
1 import streamlit_authenticator as stauth
2
3
4
5
6 hashed_passwords = stauth.Hasher(['admin123', 'suraj123']).generate()
7
8
9
10 from PIL import Image
11 def add_logo(logo_path, width, height):
12     """Read and return a resized logo"""
13     logo = Image.open(logo_path)
14     modified_logo = logo.resize((width, height))
15
16     return modified_logo
17
```

5.8 Limitations.

While the Plant Disease Prediction using Deep learning project has the potential to be highly beneficial, there are some limitations and challenges that need to be considered. Here are some limitations of this project:

- **Limited dataset:** Deep learning models require large amounts of data to be trained effectively. However, in the case of plant disease prediction, it can be challenging to obtain a sufficient amount of data for all plant species and disease types. This can limit the accuracy and generalization of the model.
- **Difficulty in labeling data:** Obtaining a large dataset is not the only challenge. The data needs to be properly labeled to train the deep learning model. The process of labeling data can be time-consuming and expensive, and errors in labeling can impact the accuracy of the model.
- **Variability in plant appearance:** Different plant species can exhibit varying appearances for the same disease type, and the appearance of a plant can be influenced by environmental factors. This can make it challenging to accurately identify the disease and train the model.
- **Model overfitting:** Deep learning models are prone to overfitting, where the model learns to recognize specific features in the training dataset that are not present in the test dataset. This can result in poor generalization performance of the model.
- **Model explainability:** Deep learning models are often referred to as "black boxes" because it can be difficult to understand how they arrive at their predictions. This can be problematic when trying to diagnose a disease and understand the underlying causes.
- **Deployment challenges:** Deploying deep learning models in the real world can be challenging due to computational requirements and compatibility with hardware and software infrastructure.

In conclusion, while the Plant Disease Prediction using Deep learning project has great potential, it is important to consider the limitations and challenges associated with the project to ensure that the results are accurate, reliable, and practical.

5.9 Future Scope.

The Plant Disease Prediction using Deep learning project has significant future scope as it can have a major impact on agriculture, food security, and environmental sustainability. Here are some potential future directions for this project:

- **Expansion of plant species and disease types:** Currently, this project may be limited to certain plant species and disease types due to the availability of data. However, as more data becomes available, the project can be expanded to cover a wider range of plant species and diseases, which will enhance its practicality and usefulness.
- **Integration with precision agriculture:** The use of deep learning for plant disease prediction can be integrated with precision agriculture techniques to optimize crop yields and minimize the use of fertilizers and pesticides. This can lead to more sustainable and efficient agricultural practices.
- **Real-time monitoring:** The project can be integrated with real-time monitoring systems that can continuously analyze plant health data and provide early warnings of disease outbreaks. This can enable timely interventions to prevent the spread of diseases and reduce crop losses.
- **Collaboration with plant scientists:** Collaboration with plant scientists can help to identify new features and data sources that can enhance the accuracy and robustness of the deep learning models. This can lead to the development of more advanced and reliable prediction models.

In conclusion, the Plant Disease Prediction using Deep learning project has significant future scope in agriculture and environmental sustainability. With continued research and development, this project has the potential to revolutionize the way we detect, diagnose, and manage plant diseases.

References.

- Articles:

<https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/>

https://www.researchgate.net/publication/360397084_Plant_Disease_Detection_Using_Cnn

- Youtubers:

[Misra Turp](#)

[CodingIsFun](#)

[Avra](#)

[krishnaik](#)

[Fanilo Andrianasolo](#)

- Other References :

<https://docs.streamlit.io/library/api-reference>

<https://docs.python.org/3/>

<https://towardsdatascience.com/how-to-add-a-user-authentication-service-in-streamlit-a8b93bf02031>