Ministry Of Higher Education

Modern Academy

For Computer Science and

Management Technology

In Maadi

Agriculture Assistant

Submitted to

Modern Academy

Computer Science and Management Technology in Maadi

Computer Science Department

Prepared by

Isaac Malak Isaac

Mai Nasser AbdElkhaleq

Mahmoud Ahmed AbdElmo'men

Mahmoud Nabil ElShazly

AbdElrahman Adel Mohamed

Supervised by

Dr. Mohammed Awad
Eng. Ahmed Hany

Academic Year :2023/2024

Team Members

Description	ID	Name
Back End (Team Leader)	12100026	اسحق ملاك اسحق
Embedded System	12100317	مي ناصر عبدالخالق
Embedded System	12100521	محمود احمد عبد المؤمن
Artificial Intelligence	12100366	محمود نبيل الشاذلي
Flutter Developer	12100375	عبدالرحمن عادل محمد

Acknowledgement

First and foremost, we express our deep gratitude to “God”, the Most Merciful, who granted us the blessing of knowledge, creativity, and willpower to undertake this project. This endeavor has not only provided us with invaluable experience but has also given us insight into the complexities of real-life projects, equipping us with the skills needed to design and analyze real projects.

We also gratefully thank our **“Modern Academy”** and its board members for their professionalism, supervision, and support that helped us grow on our educational journey.

“Prof. Dr. Nabil Deabes” For the exceptional quality of education we have received that could not have been without his supervision which in return helped us prepare for our professional career.

“Prof. Dr. Mahmoud Gadallah” For his guidance and keenness on providing us with the best educational facilities along with the department staff that helped us so much throughout our academic years.

“Dr. Mohammed Awad” For her valuable guidance and support in completion of this project. And for always pushing us to do better. We are truly thankful to have been given the opportunity to learn from him and be inspired by his encouragement since day one.

Finally, special thanks to **“Eng. Ahmed Hany”** for his help and effort through the year and for always guiding us with patience and care.

All the help and support that we have received during the project implementation is very much appreciated and it is the reason we were finally able to present our idea and make it real, hoping it helps many people out there.

Abstract

Considering the rapid developments in computerization and software, a major transitional stage is taking place in the sciences of artificial intelligence across all its branches.

Therefore, we have decided to take up the challenge and apply what we have studied over the past years to create the Agriculture Assistant, which marks the beginning of a boom in the world of agriculture and plant diseases. Our Agriculture Assistant falls under the field of smart agriculture and is specifically designed for greenhouses, where the plants require special care. “Smart agriculture” is a system that utilizes machine learning techniques to assist farmers in farming.

The Agriculture Assistant represents a groundbreaking advancement in agricultural technology, harnessing the power of artificial intelligence, specifically deep learning, to revolutionize plant monitoring and disease detection within greenhouse environments. By training on vast datasets encompassing a multitude of plant diseases and health conditions, the deep learning model achieves unparalleled accuracy in disease identification. Through the extraction of intricate features from plant images, the model discerns nuanced patterns indicative of various diseases, enabling it to distinguish between healthy and diseased plants with exceptional precision.

The integration of deep learning technology not only enhances the efficiency and accuracy of disease detection but also enables early intervention, mitigating the risk of crop loss and maximizing yields. Furthermore, the system's adaptive nature allows it to continually refine its disease identification capabilities through ongoing learning and adaptation to new data.

This project is conceived as an integrated and interconnected system to serve its owner, whether they are a large company, a modern traditional farmer, or an enthusiast of agriculture and plants. As trying to solve the problem of poor quality of plants and crops and we strive to preserve the agricultural system as a cornerstone of food security by deploying cutting-edge technologies to monitor plants throughout their lifecycle, from planting to harvest, and to achieve our objectives, we have implemented a multi-faceted approach:

Firstly, we have created a self-moving model equipped with a camera and sensors to monitor agricultural land as well as the plants.

Secondly, we have developed an artificial intelligence model that analyzes plant images taken by the robot and detects the type of plant and the diseases it may suffer from.

Thirdly, we have developed a mobile application to facilitate remote management, including a special section for processing images and information about diseases and suggesting appropriate treatments.

Upon detection of a disease, the system generates a comprehensive report detailing the specific name of the disease and its precise location within the greenhouse. This information is seamlessly communicated to users via a user-friendly mobile application interface, providing real-time updates on the health status of their plants.

Table of index

Contents

Table of Figures

1. Chapter One

1.1 Overview

In our modern era, the rapid advancement of technology has become indispensable, permeating every aspect of our lives, from agriculture and industry to commerce and tourism. Its profound impact on daily life is undeniable, revolutionizing how we access resources and acquire information. Technology has become ubiquitous, shaping our behaviors, and functioning in profound ways. From networking and healthcare to communication and transportation, it has significantly enhanced our quality of life. Looking ahead, emerging technologies like artificial intelligence are poised to reshape various fields, including agriculture, which will be the focus of our project.

The field of "Smart agriculture" is a system that uses modern technological methods and information management and analysis systems to make optimal decisions in agriculture in a sustainable and clean way, in addition to controlling pests at the lowest cost.

Under this field we created The Agriculture Assistant which represents a groundbreaking advancement in agricultural technology, harnessing artificial intelligence, specifically deep learning, to monitor plants and detect diseases inside greenhouses. By training on extensive datasets, our deep learning model achieves unprecedented accuracy in disease identification, distinguishing between healthy and diseased plants with exceptional precision. This integration of deep learning technology not only enhances efficiency and accuracy but also facilitates early intervention, minimizing the risk of crop loss and maximizing yields.

1.2 Background

The agriculture industry is one of the most vital sectors for contribution to the national income in many countries. Throughout the years, many agriculture components and processes have become automated to ensure faster production and to ensure products are of the highest quality standards. Because of the increased demand in the agricultural industry, it is vital that agriculture produce is cultivated using an efficient process [1]. Diseases and defects found in plants and crops have a great impact on production in the agriculture industry, and lead to significant economic losses [2]. A loss of an estimated 33 billion dollars every year was the result of plant pathogens found in crops in the United States. Pathogenic species affect plants significantly, introducing diseases such as chestnut blight fungus and huanglongbing citrus disease [3]. Insect infestation along with bacterial, fungal, and viral infections are another main contribution to diseases found in plants [4]. Changes in climate and temperature are also a few factors that may contribute to the increase in diseases found in plants. Once a plant has been infected, symptoms develop on various segments of the plant, ultimately degrading the growth of the subsequent fruit or vegetable [5].

Apple production is a very large industry especially in China with over 17 million tons of produce every year [2]. Apple infections do not only significantly reduce grade and yield but can also affect the return bloom of the following season [6]. These factors have a drastic impact on countries that rely heavily on its agriculture sector as its main method of income. To overcome these losses and issues of plant diseases, farmers tend to look to chemical pesticides as a remedy solution. This solution may be effective in eliminating plant diseases but has drastic drawbacks. As well as being costly, the increased use of pesticides creates dangerous levels of toxic residue levels on agriculture products [7]. This leads to concerns about wholesomeness and healthiness of products raised by the public when pesticides are commonly used in the produce they purchase [8]. Therefore, the use of pesticides must be controlled, and used only when necessary. This controlled or monitored method of pesticide use is known as selective pesticide spraying.

For the purpose of minimizing losses found in defective plants many techniques have been introduced. Manual techniques, such as hand inspection and naked eye observation are very common methods used by farmers. Plant diseases are detected and characterized by observation from experts, which can

be very expensive and time consuming [2]. Because these methods are very tedious it is prone to sorting errors and judgmental errors from different farmers [6]. Therefore, disease detection systems were introduced that tackle many of the issues faced with labor-intensive techniques.

1.3 Motivation

The motivations for this project are multifaceted and stem from various needs and aspirations within the realm of agriculture and technology. Here are some key motivations:

1. Enhancing Efficiency: There is a growing need to optimize agricultural practices to meet the demands of a rapidly growing population. Implementing technology-driven solutions can enhance efficiency in farming processes, leading to increased productivity and sustainability.

2. Improving Crop Health: Plant diseases pose a significant threat to crop yield and quality. By developing a system capable of early disease detection and intervention, we aim to minimize crop loss and ensure healthier plants.

3. Empowering Farmers: Providing farmers with access to advanced technological tools empowers them to make informed decisions and manage their farms more effectively. Through features like real-time monitoring and disease alerts, this project aims to support farmers in optimizing their agricultural practices.

4. Advancing Technology: By exploring the application of artificial intelligence and mobile technology in agriculture, we contribute to the advancement of these fields. This project serves as a platform for innovation and experimentation in utilizing cutting-edge technology for practical agricultural solutions.

5. Promoting Sustainability: Sustainable agriculture is essential for preserving natural resources and mitigating environmental impact. By facilitating more efficient and precise farming methods, this project contributes to the promotion of sustainable agricultural practices.

6. Addressing Food Security: Ensuring food security is a global challenge, particularly in the face of climate change and population growth. By improving crop health and productivity, this project contributes to efforts aimed at enhancing food security and resilience in agricultural systems.

Overall, the motivations for this project revolve around leveraging technology to address key challenges in agriculture, empower farmers, advance scientific knowledge, and contribute to global sustainability efforts.

1.4 Problem Definition

Traditional farming practices often face challenges in effectively detecting and addressing common diseases that affect plants. Farmers struggle to accurately identify plant diseases in a timely manner, resulting in substantial crop losses and reduced yields. Additionally, limited access to expert knowledge and effective treatments.

1.5 Statement of Purpose

Bacterial and fungal viral infections have a significant impact on plant health and introduce diseases that affect growth of produce. Moreover, the over-reliance on fungicides and pesticides to remedy this issue, is not only costly, but has a considerably negative impact on the environment. Therefore, there is a need to detect and target plant diseases at an early stage to aid farmers to take appropriate precautions to help preserve the defective plant.

The purpose of this project is to develop an innovative solution leveraging advanced technology to address key challenges in agricultural management, particularly in greenhouse environments. Our project aims to revolutionize plant monitoring and disease detection processes, ultimately enhancing crop health, productivity, and sustainability.

An easy-to-use system to detect plant leaf disease severity is designed for farmers and agriculturists to measure disease severity levels of plants. In addition, an automated approach is designed and implemented for early leaf-based plant health monitoring using a robotized system in real field-based environments.

1.6 Objectives

The objective of this project is to:

1. Develop an innovative solution: Leveraging advanced technology to address key challenges in agricultural management, with a focus on greenhouse environments.

2. Revolutionize plant monitoring and disease detection processes: By implementing automated systems for early detection and targeting of plant diseases, ultimately enhancing crop health, productivity, and sustainability.

3. Design and develop an application that aids in the identification of plant diseases, providing users with valuable insights for effective plant care, facilitating prompt intervention and preservation of plant health.

4. Design and development of an automated approach: Using automated systems for early monitoring of leaf-based plant health in real field environments, which helps growers and growers to easily follow plants at any time and improve efficiency and accuracy in disease detection.

1.7 Expected Outcome

1. A fully functional self-moving model equipped with cameras and sensors for plant monitoring.

2. An artificial intelligence model trained to analyze plant images and detect diseases accurately.

3. A user-friendly mobile application interface for remote management and real-time monitoring of plant health.

4. Comprehensive reports on disease detection, detailing the specific name and location of diseases within the greenhouse.

1.8 Stakeholders

Stakeholders play various roles and have different interests and perspectives related to the project. Engaging with them effectively and addressing their needs and concerns is essential for the success and sustainability of our project, and they are:

1. Farmers: Primary stakeholders who will directly benefit from the implementation of the project. They are responsible for managing the greenhouse environments and ensuring the health and productivity of the plants.

2. Plant Enthusiasts with Limited Time: Individuals who have a love for planting and gardening but may lead busy lifestyles or have limited time to dedicate to plant care. They may struggle to identify and address plant diseases due to their lack of expertise and understanding. Engaging with this stakeholder group and providing them with accessible tools and resources for plant monitoring and disease detection could greatly enhance their gardening experience and help them maintain healthier plants.

3. Agricultural Experts and Researchers: Professionals with expertise in plant pathology, agricultural technology, and data science. They may provide insights and guidance during the development and validation of the system.

4. Technology Developers and Engineers: Individuals or teams responsible for designing, developing, and implementing the hardware and software components of the system. They play a crucial role in ensuring the technical feasibility and functionality of the project.

5. Government Agencies and Regulatory Bodies: Stakeholders involved in regulating agricultural technologies and ensuring compliance with relevant laws and regulations. They may provide guidance on safety standards, data privacy, and environmental regulations.

6. End Users: Individuals who will interact with the system on a daily basis, including greenhouse operators, farm workers, and agricultural technicians. Their feedback and input are essential for ensuring the usability and effectiveness of the system.

7. Environmental Organizations: Stakeholders concerned with environmental

sustainability and conservation. They may assess the project's impact on natural resources, biodiversity, and ecosystem health.

8. Educational Institutions: Universities, colleges, and research institutions with an interest in agricultural technology and innovation. They may collaborate with the project team on research, development, and knowledge dissemination activities.

2. Chapter Two

2.1 Related Works

There are several people and companies that have implemented similar agricultural robotics projects in the past. These people may be leaders in agricultural technology or researchers working to develop smart solutions for agriculture, including:

1. Simon Blackmore is an agricultural scientist and specialist in agricultural robotics. He works at Harper Adams University in the United Kingdom and has developed several agricultural robots to assist in harvesting and cultivation operations.

2. Blue River Technology: Founded in 2011, Blue River Technology is an agricultural technology company specializing in developing agricultural robots to improve agricultural productivity and reduce the use of pesticides. The company has developed a device that uses computer vision and deep learning techniques to spray pesticides only on weeds.

2.2 Comparative Studies

Numerous studies have explored the intersection of technology and agriculture, focusing on advancements in plant monitoring, disease detection, and agricultural automation. The following key areas of related research provide valuable insights into the development and implementation of similar projects:

1. Plant Disease Detection Technologies: Research in plant pathology has led to the development of various methods for detecting and diagnosing plant diseases. Techniques such as image processing, machine learning, and sensor-based systems have been employed to accurately identify disease symptoms and assess plant health. Previous studies have demonstrated the efficacy of machine learning algorithms in analyzing plant images to detect diseases with high accuracy rates.

2. Autonomous Agricultural Monitoring Systems: The emergence of autonomous agricultural monitoring systems has revolutionized farming practices by enabling real-time monitoring of crop health and environmental conditions. Robotic platforms equipped with sensors and cameras can navigate fields, collect data, and provide actionable insights to farmers. These systems offer a scalable and cost-effective solution for monitoring large agricultural areas while minimizing human intervention.

3. Mobile Applications for Plant Health Management: Mobile applications play a crucial role in modern agriculture by providing farmers with convenient tools for plant health management. These applications often utilize image recognition technology to identify plant diseases from user-uploaded photos, offering recommendations for disease management and treatment. Additionally, mobile applications facilitate remote monitoring and data analysis, empowering users to make informed decisions about crop management practices.

Our project is not like other similar projects in the field of smart agriculture as it is an integrated solution that includes hardware and software, integrating hardware and software solutions is a key trend in agricultural technology, and it is available to all individuals and organizations.
We also tried to take the cost into account, as there is no such low price with the same features, it allows comprehensive and effective monitoring of plants and diseases. By combining sensor-equipped robotic platforms with cloud-based analytics and mobile applications, we have developed comprehensive plant health management systems in greenhouse environments. These integrated solutions provide real-time monitoring, early disease detection and data-driven insights to improve crop productivity and minimize losses.

2.3 Scope of the Problem

The project will focus on designing and developing an integrated system for plant monitoring and disease detection in greenhouse environments. It is a self-moving model that helps us monitor plants. The device will be equipped with advanced hardware and software technologies that enable it to identify plants and their diseases.

As well as providing a user-friendly mobile application interface for remote management, providing real-time updates on plant health status and disease alerts to users.

2.4 Challenges

Here are some possible obstacles that may hinder the functioning of the agricultural robot system:

1. Technical Complexity: Integrating hardware (such as cameras, sensors, and robotic platforms) with software (including deep learning models) can be technically complex. Ensuring seamless communication and interoperability between different components of the system may pose challenges.

2. Data Quality and Quantity: Training machine learning models for accurate disease detection requires a large quantity of high-quality labeled data. Collecting and annotating such datasets can be time-consuming and resource-intensive, especially for rare or complex diseases.

3. Disease Identification: Accurate identification of plant diseases through the camera can be challenging, especially if the diseases require complex laboratory analysis. There may be difficulties in accurately identifying and classifying diseases using only captured images.

4. Real-Time Processing: Processing plant images and analyzing them for disease detection in real-time requires efficient algorithms and computing resources. Ensuring timely responses and minimizing processing latency may be challenging, especially in resource-constrained environments.

5. Damage to one of the hardware parts.

6. The Terrain: The robot may face difficulties in navigating rough terrains or uneven ground. It may require special design considerations or modifications to its structure to handle such terrains effectively.

7. Cloud Connectivity: This system relies on cloud connectivity to send data and receive analysis and results. Connectivity issues or network availability in remote or sparsely populated areas may pose challenges to seamless cloud connectivity.

8. Power Consumption and Battery Life: The robot may face challenges related to power consumption and battery life. Ensuring an efficient power supply system and optimizing energy usage are crucial for the robot to operate for extended

periods without the need for frequent recharging or battery replacement.

9. Environmental Variability: Greenhouse environments can exhibit significant variability in factors such as lighting conditions, humidity levels, and temperature fluctuations. Ensuring the robustness and reliability of the system across diverse environmental conditions may be challenging.

10. User Acceptance and Adoption: Introducing new technologies and workflows into agricultural settings may face resistance from users who are accustomed to traditional methods. Ensuring user acceptance and adoption of the system through effective training and support mechanisms is crucial for successful implementation.

11. Regulatory Compliance: Depending on the jurisdiction, agricultural technologies may be subject to regulatory requirements and standards. Ensuring compliance with relevant regulations and obtaining necessary approvals may add complexity to the project.

12. Cost Constraints: Developing and deploying advanced technologies such as robotic platforms and machine learning algorithms can be costly. Balancing the project's technical requirements with budget constraints and ensuring cost-effectiveness may pose challenges.

13. Maintenance and Upkeep: Once deployed, the system will require regular maintenance and upkeep to ensure optimal performance. Addressing issues such as hardware malfunctions, software bugs, and data drift over time requires ongoing attention and resources.

It is important to note that these obstacles can be addressed through continuous improvement and maintenance of the agricultural robot system.

3. Chapter Three

3.1 Requirements & Analysis

Some simple but essential specifications are acquired during the program's implementation and data collection.

3.1.1 Functional Requirement

- Real-time Plant Monitoring: The system should be able to continuously monitor plants in greenhouse environments in real-time, capturing images and sensor data at regular intervals.

- Disease Detection: The system should be capable of accurately detecting plant diseases from images captured by the cameras. This includes identifying common bacterial, fungal, and viral infections that affect plant health.

- Automated Analysis: The system should automatically analyze plant images using deep learning algorithms to detect disease symptoms and abnormalities. It should classify diseases based on visual cues such as discoloration, lesions, or deformities.

- Alert Generation: Upon detecting a disease or abnormality, the system should generate alerts or notifications to inform users of the specific plant affected, the type of disease detected, and its severity level.

- Mobile Application Interface: The system should have a user-friendly mobile application interface that allows users to remotely monitor plant health, view disease reports, and receive real-time updates on their mobile devices.

3.1.2 Non-Functional Requirement

- Accuracy: The disease detection algorithms should achieve a high level of accuracy to minimize false positives and negatives, ensuring reliable identification of plant diseases.

- Reliability: The system should be reliable and robust, capable of operating continuously without frequent downtime or interruptions. It should be resilient

to environmental factors such as temperature fluctuations and lighting conditions.

- Scalability: The system should be scalable to accommodate varying greenhouse sizes and configurations. It should be able to handle large volumes of data and adapt to changes in the number of monitored plants.

- Usability: The mobile application interface should be intuitive and easy to use, catering to users with varying levels of technical expertise. It should provide clear instructions and guidance for navigating the system and interpreting disease reports.

- Security: The system should implement security measures to protect sensitive data, such as plant images and user information, from unauthorized access or tampering. This includes encryption of data transmission and access controls for user authentication.

- Performance: The system should exhibit high performance, with fast image processing and analysis times to ensure timely detection of plant diseases. It should also be able to handle peak loads during periods of high activity.

- Compliance: The system should comply with relevant regulations and standards for agricultural technology, data privacy, and environmental protection. This includes adherence to legal requirements and industry best practices for data handling and processing.

3.1.3 Software requirements

3.1.3.1 Android

- Operating system: Android
- API level: 21
- Android Version: 5.0 (Lollipop)

3.1.4 Hardware requirements

3.1.4.1 Android

- Device capabilities: touch screen, Internet connectivity (Wi-Fi or mobile data connection (3G/4G/5G)) and GPSconnectivity.
- RAM: 1GB, preferably higher.
- Storage Space: At least 16 GB of available storage space (although 8GB of storage may be sufficient, recommending a minimum of 16GB ensures a more comfortable user experience and accommodates a wider range of usage scenarios and future needs).
- Processor (CPU): Dual-core 1.5 GHz or higher.

3.1.4.2 ESP32

- Dual-core Tensilica Xtensa LX6 processor
- Built-in Wi-Fi (802.11 b/g/n) and Bluetooth connectivity
- Low power consumption with advanced power management features
- Rich peripheral set including GPIO, SPI, I2C, UART, ADC, DAC, PWM, etc.
- Secure communication protocols for data transmission
- Flexible development environment supporting Arduino IDE, Espressif IDF, MicroPython, etc.

3.1.4.3 ESP32Cam

- Onboard ESP32-S module supports Wi-Fi + Bluetooth.
- OV2640 camera with built-in flash lamp.
- Onboard micro-SD card slot supports up to 4G TF card for data storage.
- Supports Wi-Fi video monitoring and Wi-Fi image upload.
- Supports multi sleep modes, deep sleep current as low as 6mA.

3.1 UML Diagrams

3.2.1 Use Case Diagram

The purposes of use case diagram can be said to be as follows:

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction between the system and actors.

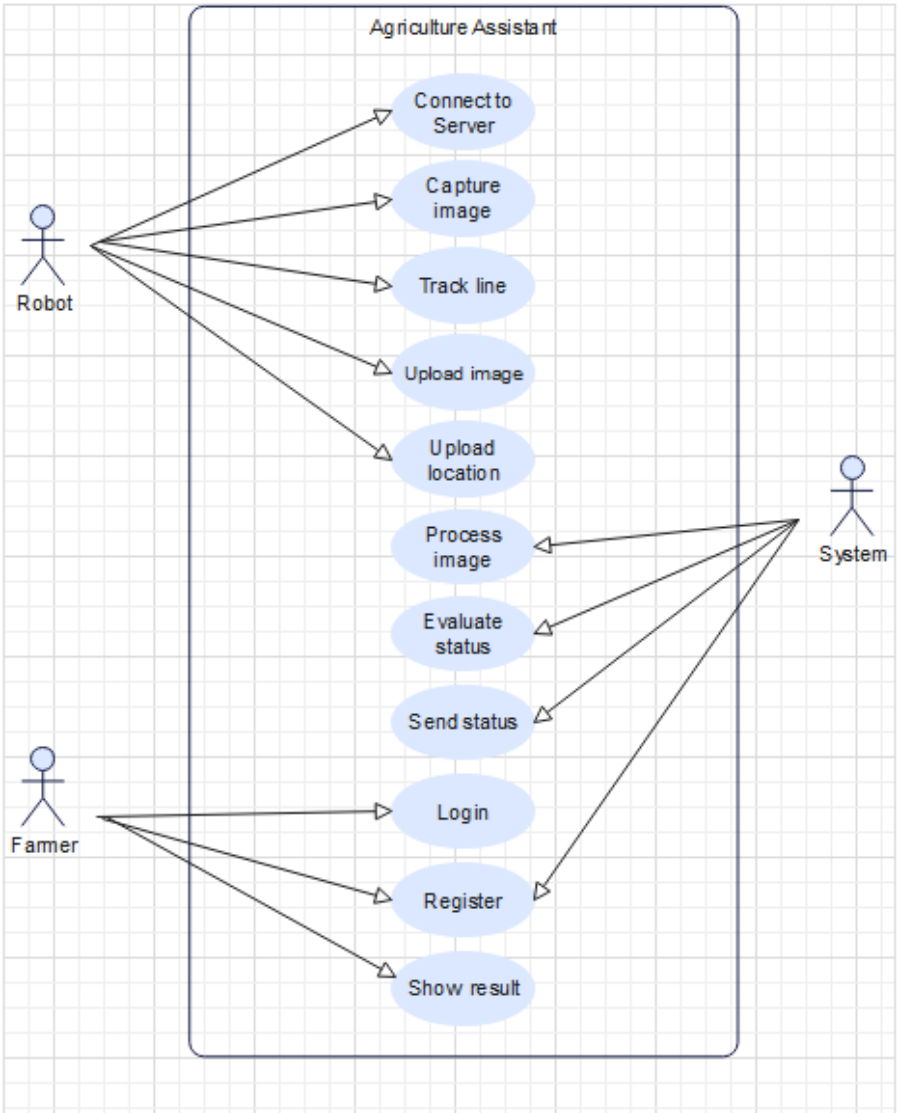


Fig 3.1 Use Case Diagram

3.2.2 Sequence diagram

It explains how and in what order a collection of items interacts, a sequence diagram is similar to an interaction diagram. A sequence diagram focuses on lifelines, or concurrent processes and objects, and the messages that are passed back and forth to complete a function before the lifeline terminates.

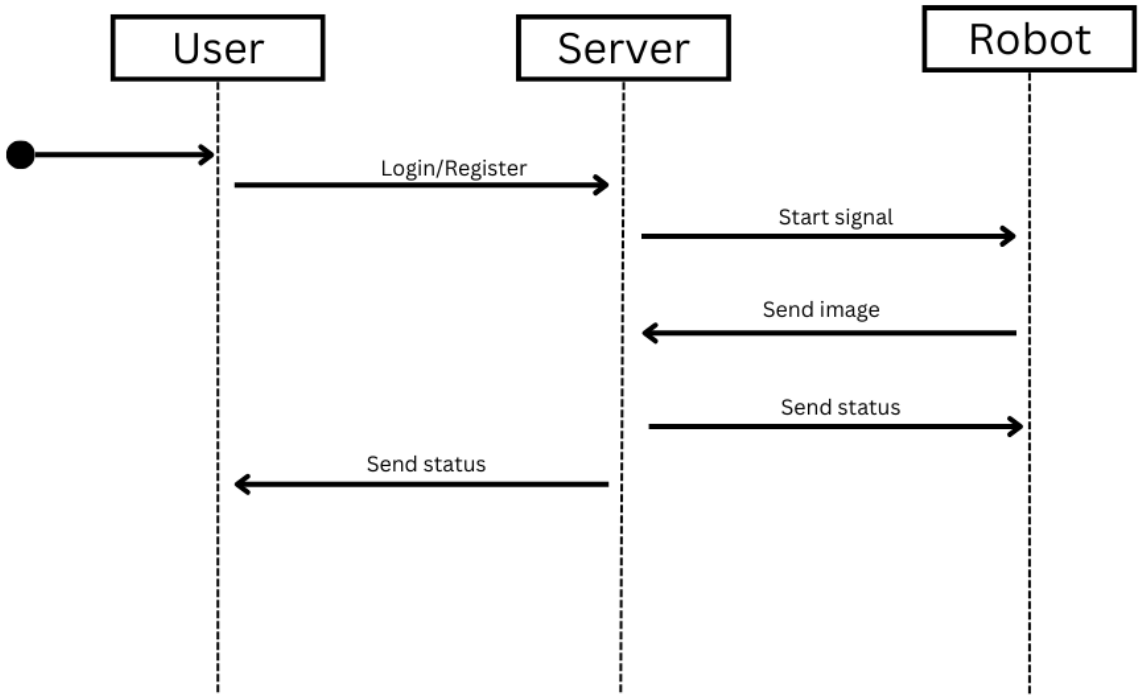


Fig 3.2 Sequence Diagram

3.2.3 Activity diagram

It shows the system's main operations and restrictions, which lead to the project's pathways. They were classified to inform programmers and users about the conduct of Vehicle Rental System.

Software operations are also depicted as a set of actions in the activity diagram. These diagrams are used to describe and define system processes and use cases. It has the potential to simplify and improve any process by highlighting difficult use cases. An activity diagram also shows the system's actions, functions, and processes.

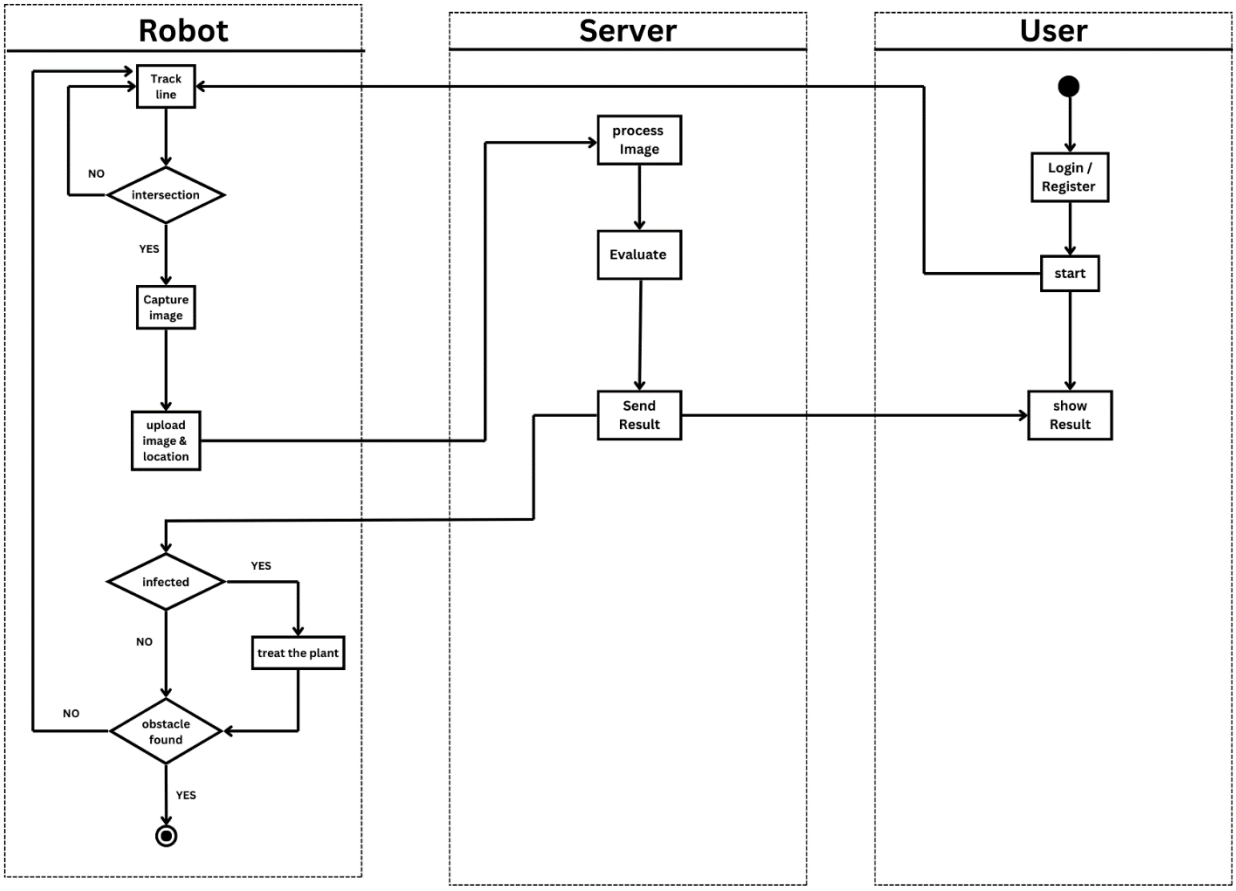


Fig 3.3 Activity Diagram

3.2.4 ERD

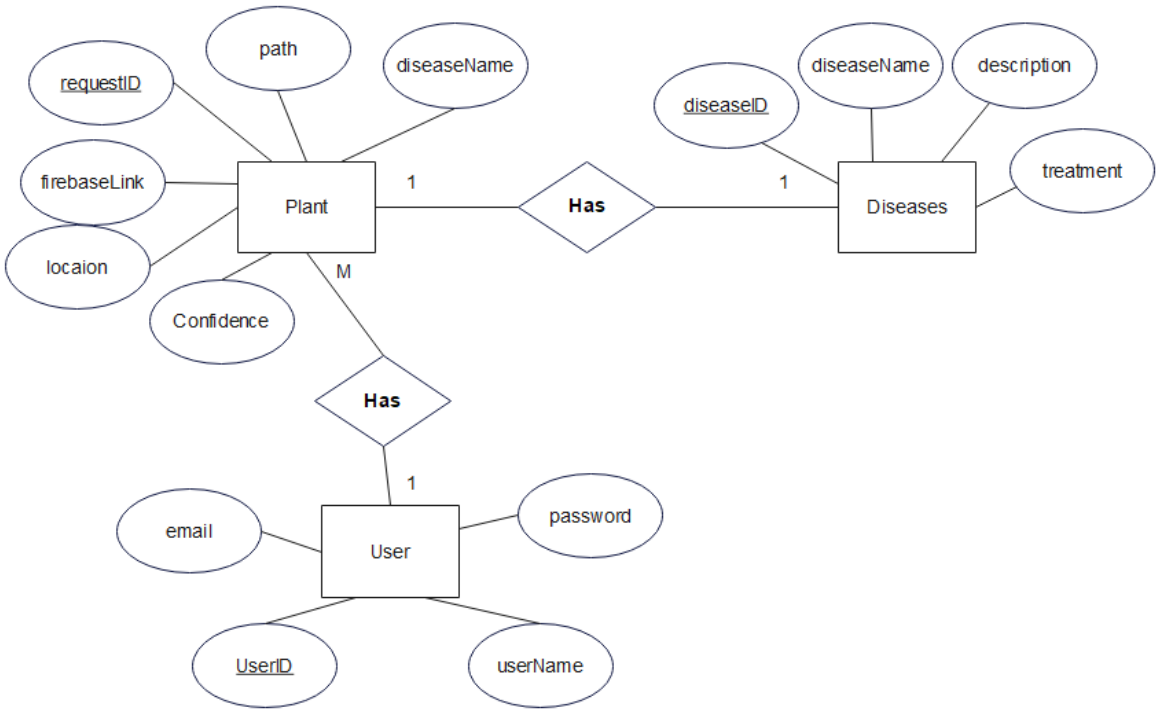


Fig 3.4 ERD

3.2.5 Class Diagram

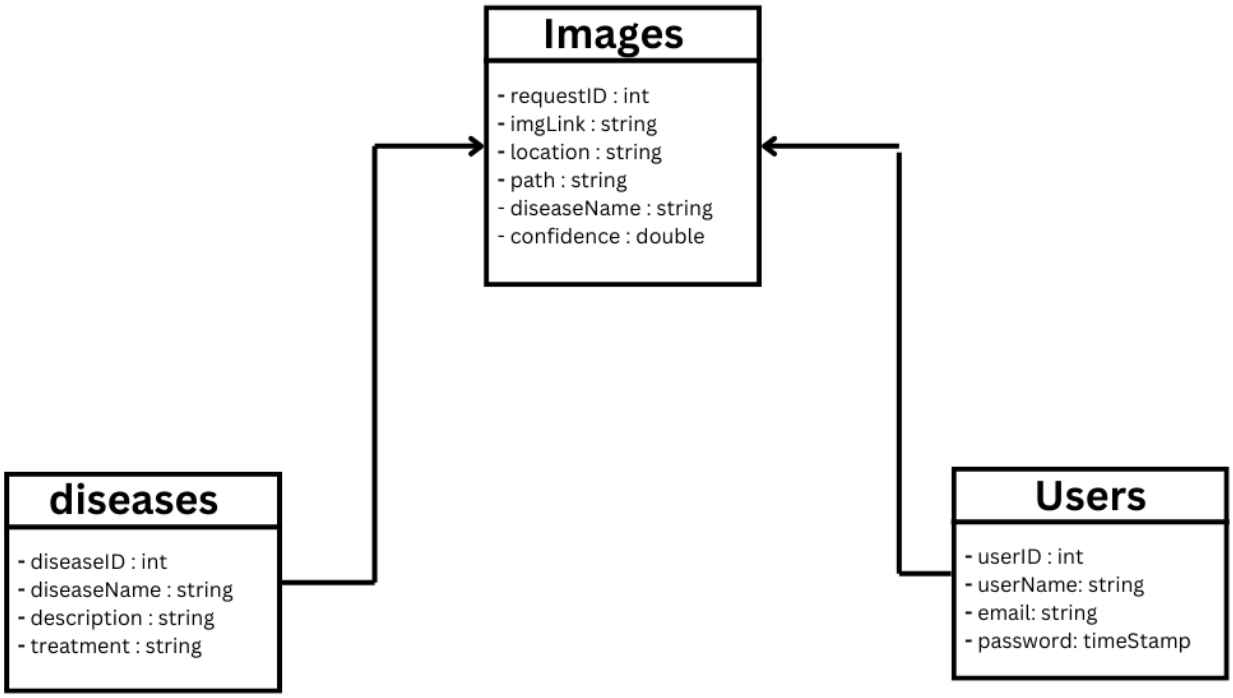


Fig 3.5 Class Diagram

4. Chapter Four

4.1 Design:

This chapter introduces the design of Agriculture Assistant. And what is the approaches and technology used to construct this framework were used.

4.1.1 Mobile Application:

4.1.1.1 Introduction:

The application serves as a comprehensive platform for users to access detailed information about scanned plants. It meticulously presents key data points such as the location and time of each scan, accompanied by high-quality images of the plants in question. Leveraging sophisticated AI algorithms, the application generates insightful analyses of the plants' conditions, highlighting any detected diseases along with pertinent details. Moreover, it goes a step further by offering tailored treatment recommendations, empowering users with actionable steps to address and mitigate any identified issues. Through this multifaceted approach, the application not only facilitates plant monitoring but also promotes proactive care and management practices.

4.1.1.2 Flutter:

Flutter, a UI software development kit developed by Google, is an open-source framework. It facilitates the creation of natively compiled applications for mobile, web, and desktop platforms from a unified codebase. Flutter offers a comprehensive collection of pre-built widgets, tools, and libraries, streamlining the app development process.

- Key Features :
  - Cross-Platform Development: Flutter enables developers to write code once and deploy it across multiple platforms including iOS, Android, web, and desktop.
  - Fast Development: Hot Reload feature allows developers to see changes instantly without restarting the app, which accelerates the development cycle.
  - Expressive UI: Flutter offers a rich set of customizable widgets and layouts to create beautiful, expressive user interfaces.
  - High Performance: Flutter's architecture is designed for high performance, delivering smooth animations and 60fps (frames per second) experiences.
  - Access to Native Features: Developers can access platform-specific features and APIs using plugins, or by writing platform-specific code.

- Rich Ecosystem: Flutter has a vibrant ecosystem with extensive documentation, community support, and third-party packages available on pub.dev.

- Architecture

Flutter follows a reactive, composable architecture that consists of the following key components:

- Widgets: Everything in Flutter is a widget, from structural elements like `Container` and `Row` to UI components like `Button` and `Text`. Widgets are lightweight, immutable, and declaratively composed to build the UI hierarchy.
- Flutter Engine: The Flutter engine is a C++ runtime and a set of low-level rendering libraries that render Flutter widgets into pixels on the screen. It provides APIs for platform integration, input handling, and rendering.
- Dart Programming Language: Flutter apps are written in Dart, a modern, object-oriented language developed by Google. Dart features a strong type system, ahead-of-time (AOT) compilation, and just-in-time (JIT) compilation for optimal performance.
- Platform Channels: Flutter apps can communicate with platform-specific code using platform channels. This allows developers to access native features and APIs not available in Flutter.

- Benefits of Utilizing Flutter
  - Unified Codebase, Diverse Platforms: With Flutter, developers can write code once and deploy it seamlessly across iOS, Android, web, and desktop platforms, minimizing development time, effort, and costs associated with maintaining separate codebases.
  - Expedited Development: Flutter's Hot Reload feature expedites the development cycle by allowing developers to instantly view changes without app restarts, fostering rapid iteration and experimentation.
  - Customizable UI: Flutter offers an extensive array of customizable widgets and layouts, granting developers full control over the visual aesthetics of their applications, resulting in highly polished and visually captivating user interfaces.
  - Optimal Performance: Engineered for high performance, Flutter ensures smooth animations and consistent 60fps experiences, even on older devices, leveraging Skia, a robust 2D rendering engine.

4.1.1.3 Application architecture

- MVC architecture pattern:

  MVC (Model-View-Controller) is a fundamental software architecture pattern commonly used in developing user interfaces. It provides a structured approach to organizing code, separating concerns, and enhancing maintainability and scalability of software projects.

- The key components of MVC architecture are:

Model:

- The Model represents the application's data and business logic.
- It encapsulates data access, manipulation, and validation operations.
- his layer is independent of the user interface and communicates with both the Controller and the View.
- In an Android application, the Model layer could include data sources such as databases, network APIs, or local files.

View:

- The View represents the presentation layer of the application.
- It displays data to the user and handles user input.
- The View is passive and does not contain any application logic.
- In Android, the View layer typically consists of XML layout files and UI components such as Text Views, Buttons, Recycler Views, etc.

Controller:

- The Controller acts as an intermediary between the Model and the View.
- It receives input from the user via the View and interacts with the Model to perform appropriate actions.
- The Controller updates the View based on changes in the Model and vice versa.
- In Android development, the Controller is often represented by Activities or Fragments, which handle user interactions and orchestrate data flow.

4.1.1.4 packages used in the application:

1. State Management with GetX

   GetX is a lightweight and powerful package for state management in Flutter applications. It offers a simple and intuitive API for managing state, handling dependencies, and navigating between screens.

- Core Concepts
  - Observables
    - Observables in GetX are variables that can be observed for changes. They enable reactive programming by automatically updating the UI in response to changes in state.

  - Controllers
    - Controllers are classes that manage the state of a widget or a group of widgets. They encapsulate business logic and expose state through observables.

  - Reactive Updates
    - GetX provides reactive updates, allowing widgets to automatically rebuild when observables change. This ensures that the UI remains synchronized with the underlying state.

- Benefits of GetX
  - Lightweight: GetX has a minimal footprint and low overhead, making it suitable for projects of any size.
  - Reactive: GetX utilizes reactive programming principles to streamline state management and UI updates.
  - Easy to Use: GetX offers a simple and intuitive API, reducing boilerplate code and complexity.
  - Performance: GetX is highly optimized for performance, with efficient memory management and minimal re-renders.
  - Dependency Injection: GetX includes built-in support for dependency injection, making it easy to manage dependencies and decouple components.
  - Route Management: GetX provides a powerful routing system with named routes, parameter passing, and route transitions.

2. Get Storage

Get Storage is a lightweight and fast key-value storage library for Flutter applications. It offers a simple and efficient way to store and retrieve data persistently on the device.

- Features
  - Simplicity: Get Storage provides a simple API for storing and retrieving data using key-value pairs.
  - Performance: Get Storage is optimized for performance, offering fast read and write operations.
  - Cross-Platform: Get Storage works seamlessly on both Android and iOS platforms, ensuring consistent behavior across devices.
  - Data Types: Get Storage supports various data types including strings, integers, doubles, booleans, lists, and maps.
  - Asynchronous Operations: Get Storage supports asynchronous read and write operations, enabling non-blocking access to data.
  - No External Dependencies: Get Storage does not depend on any external libraries or plugins, simplifying integration into Flutter projects.

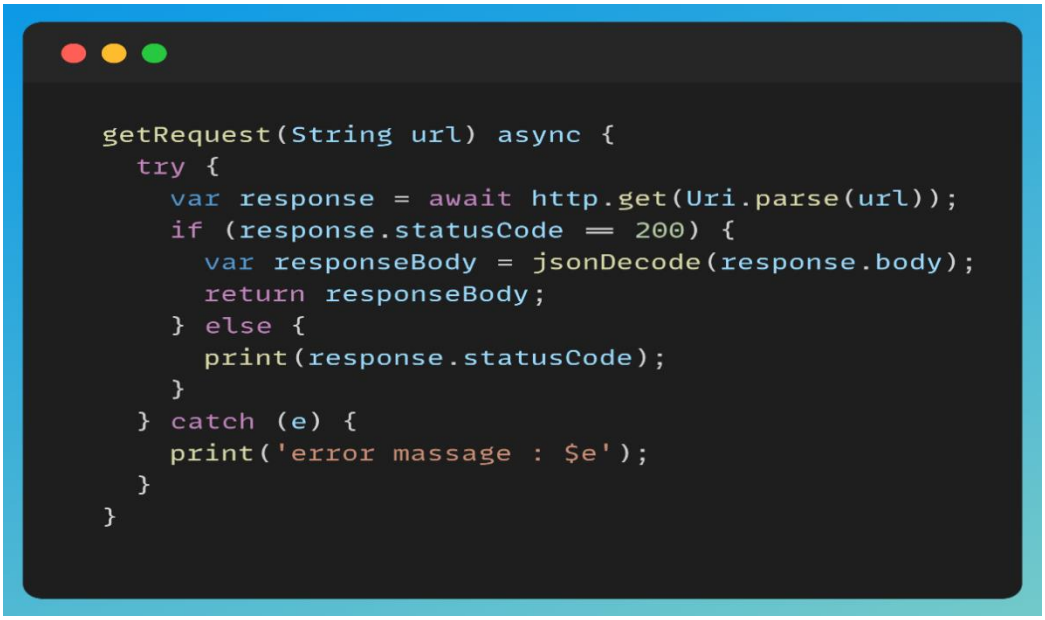
- Code example

Fig 4.1 GetStorage

3. http

The http package is a Flutter library for making HTTP requests and interacting with web servers. It provides a simple and flexible API for performing various HTTP operations such as GET, POST, PUT, DELETE, and more.

- Features
  - HTTP Requests: The http package allows you to make HTTP requests to remote servers using various HTTP methods.
  - Response Handling: It provides methods for handling HTTP responses including status codes, headers, and response bodies.
  - Asynchronous Operations: HTTP requests in the http package are asynchronous, allowing for non-blocking network operations.
  - Error Handling: The package offers error handling mechanisms for dealing with network errors, timeouts, and other exceptions.
  - Customization: It supports customization of HTTP requests and headers to meet specific requirements.
  - Interceptors: Interceptors can be used to intercept and modify requests and responses before they are sent or received.
- Code:
  - Get request:



```
getRequest(String url) async {
  try {
    var response = await http.get(Uri.parse(url));
    if (response.statusCode == 200) {
      var responseBody = jsonDecode(response.body);
      return responseBody;
    } else {
      print(response.statusCode);
    }
  } catch (e) {
    print('error massage : $e');
  }
}
```

Fig 4.2 GetRequest

- Post request:

```
postRequest(String url, Map body) async {
  try {
    var response = await http.post(Uri.parse(url), body: body);
    if (response.statusCode == 200) {
      var responseBody = jsonDecode(response.body);
      return responseBody;
    } else {
      print("error : ${response.statusCode}");
    }
  } catch (e) {
    print('error massage : $e');
  }
 }
}
```

Fig 4.3 PostRequest

4. Google fonts

The google fonts package is a Flutter library that allows developers to easily use Google Fonts in their applications. It provides a convenient API for loading and using custom fonts from the Google Fonts collection, enhancing the typography and visual design of Flutter apps.

- Features
  - Google Fonts Integration: The google fonts package seamlessly integrates with the Google Fonts API, providing access to hundreds of free and open-source fonts.
  - Simple Usage: It offers a simple API for loading fonts and applying them to text widgets in Flutter applications.
  - Font Styles and Variants: The package supports various font styles and variants including regular, bold, italic, and custom weights.
  - Customization: Developers can customize font properties such as font size, weight, style, and color to achieve the desired typographic design.
  - Performance: google fonts ensure optimal performance by caching fonts and minimizing network requests during app startup.

4.1.1.5 Design:

- Onboarding Screen

Empower users with a captivating onboarding experience that unveils the innovative features and seamless functionalities of our project, fostering engagement and guiding them towards a rewarding journey within the app.

Fig 4.4 Onboarding Screen1

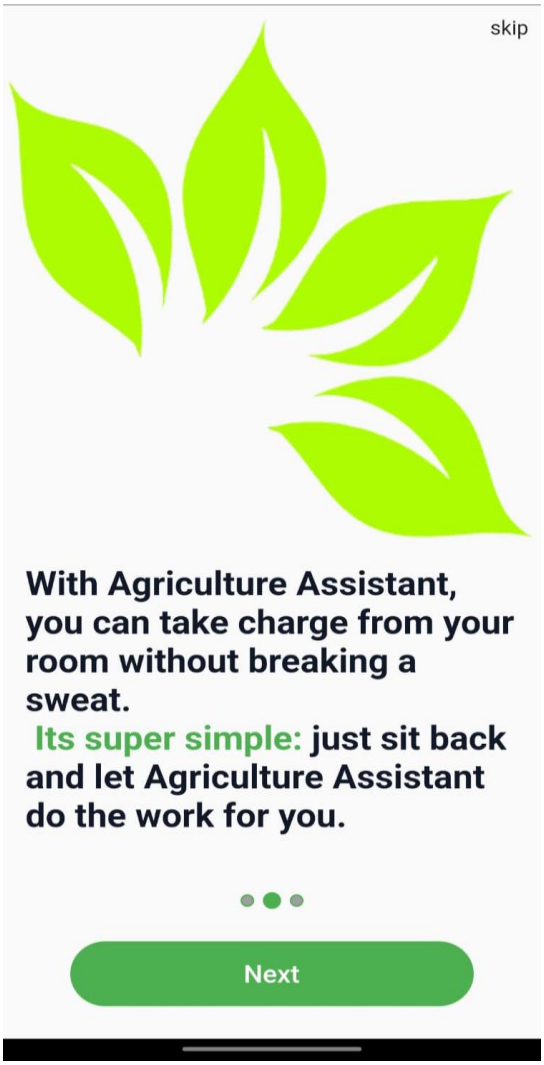


Fig 4.5 Onboarding Screen2

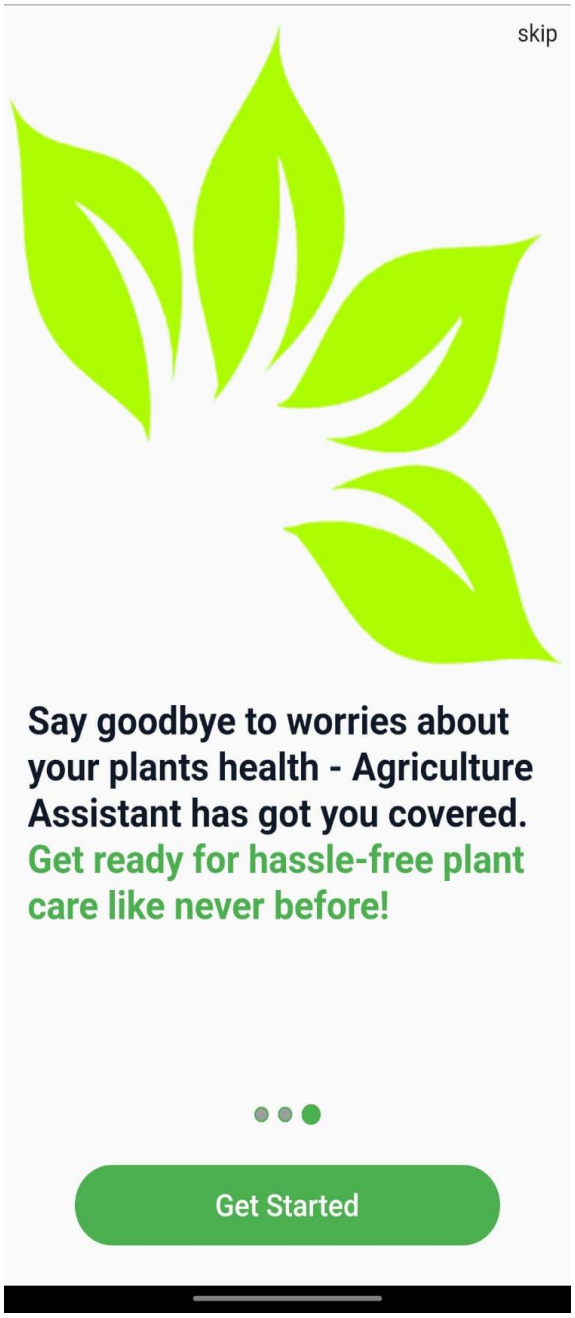


Fig 4.6 Onboarding Screen3

Fig 4.7 Onboarding Screen4

- Login and Register Screen

This page allows a user to gain access to the application by entering them.

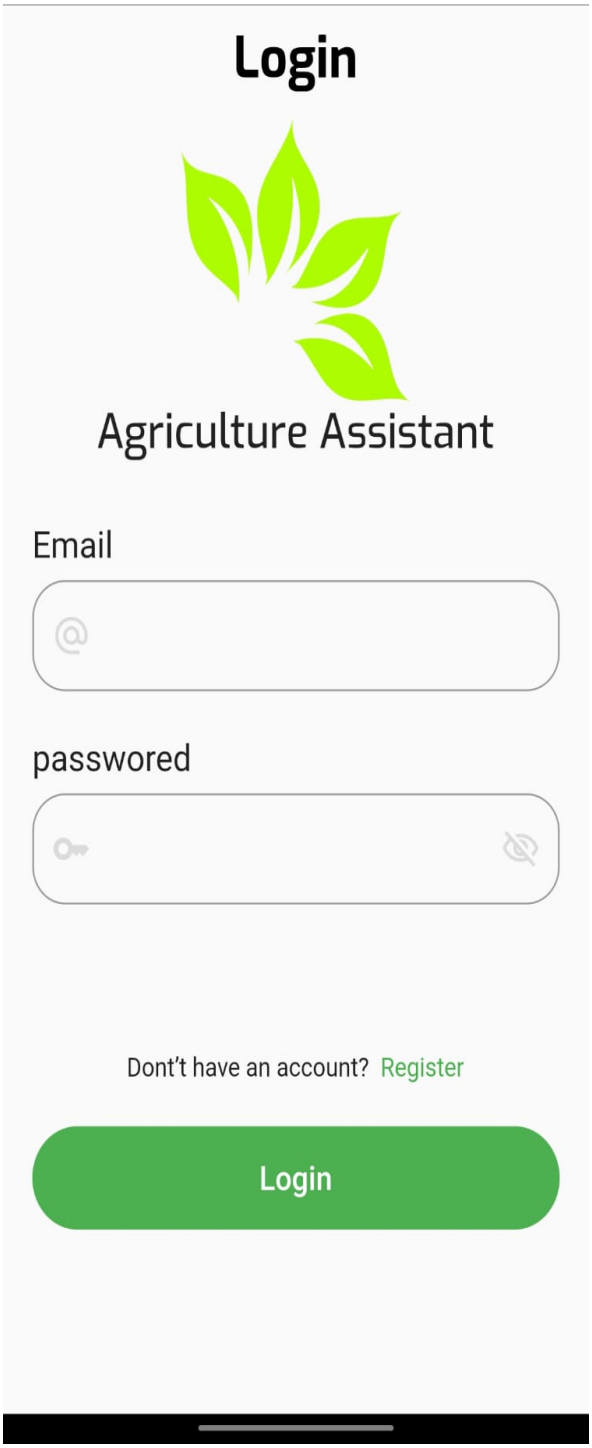


Fig 4.8 Login Screen

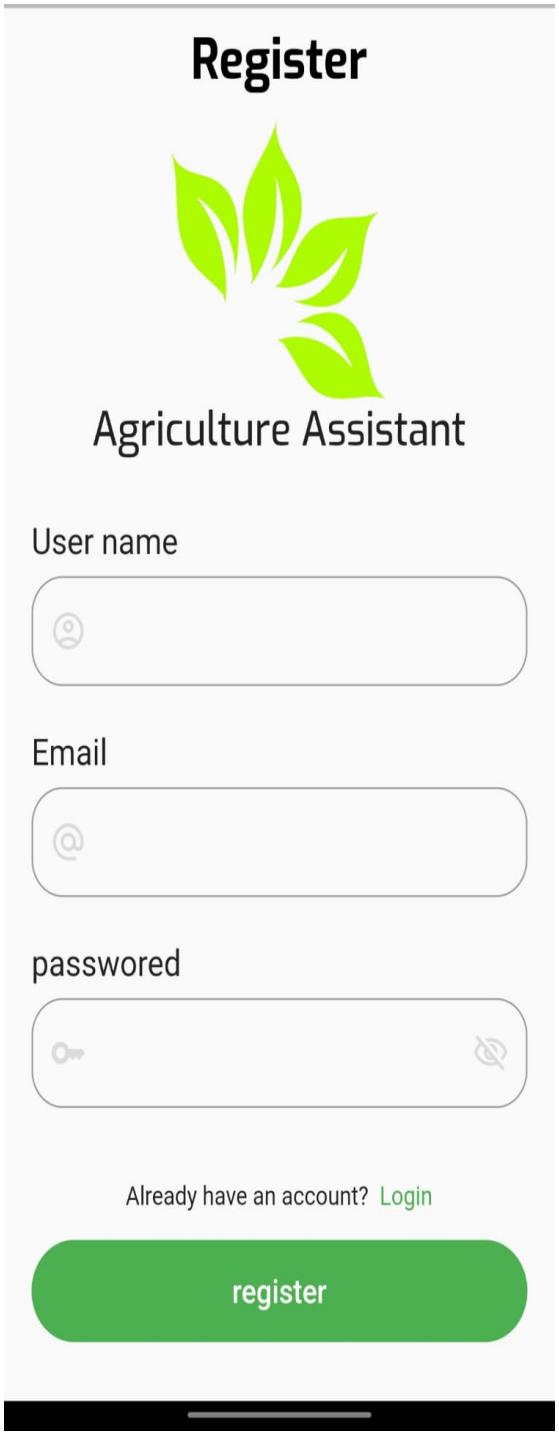


Fig 4.9 Register Screen

- Home Screen

Discover a versatile home screen offering seamless transitions to diverse sections – All Plants, Health Plants, or Sick Plant – catering to users' specific interests and ensuring an intuitive browsing experience tailored to their preferences.

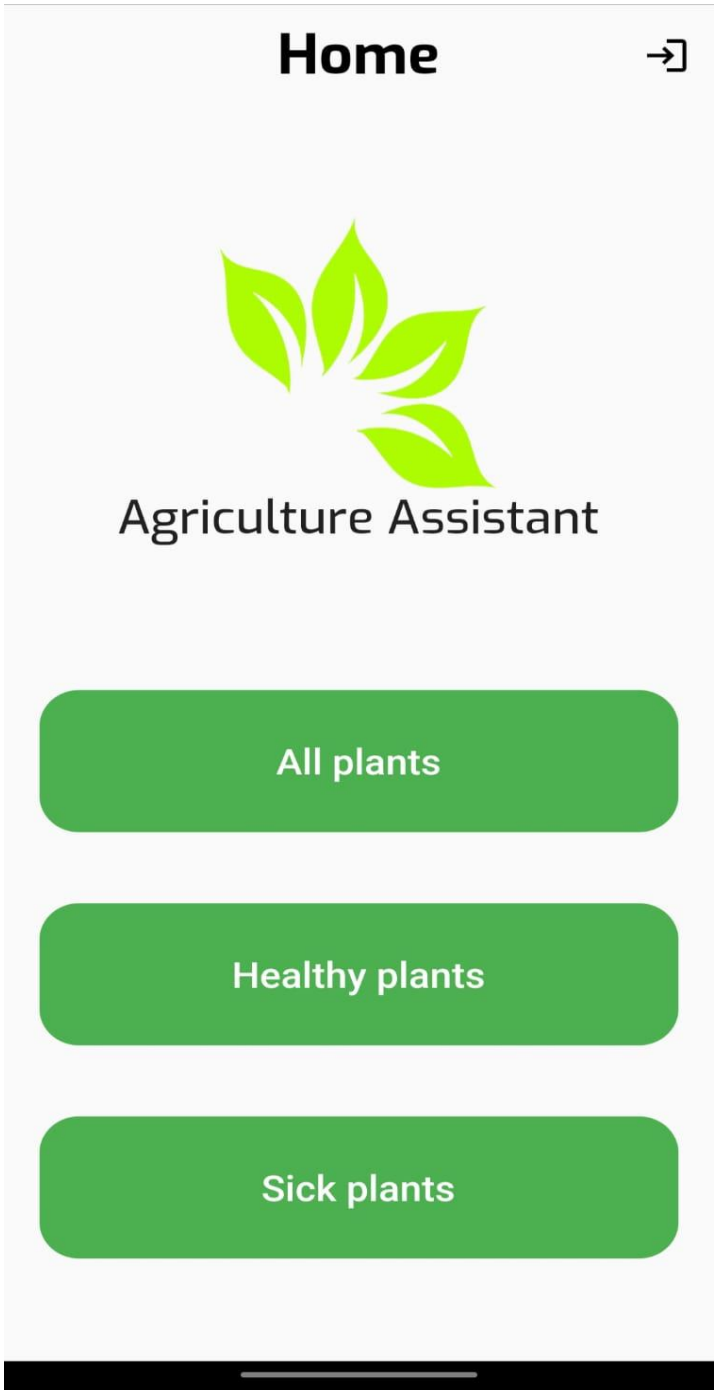


Fig 4.10 Home Screen

- All Plants Screen

screen seamlessly blending plant locations, disease information, and vibrant imagery, delivering a comprehensive botanical experience briefly.

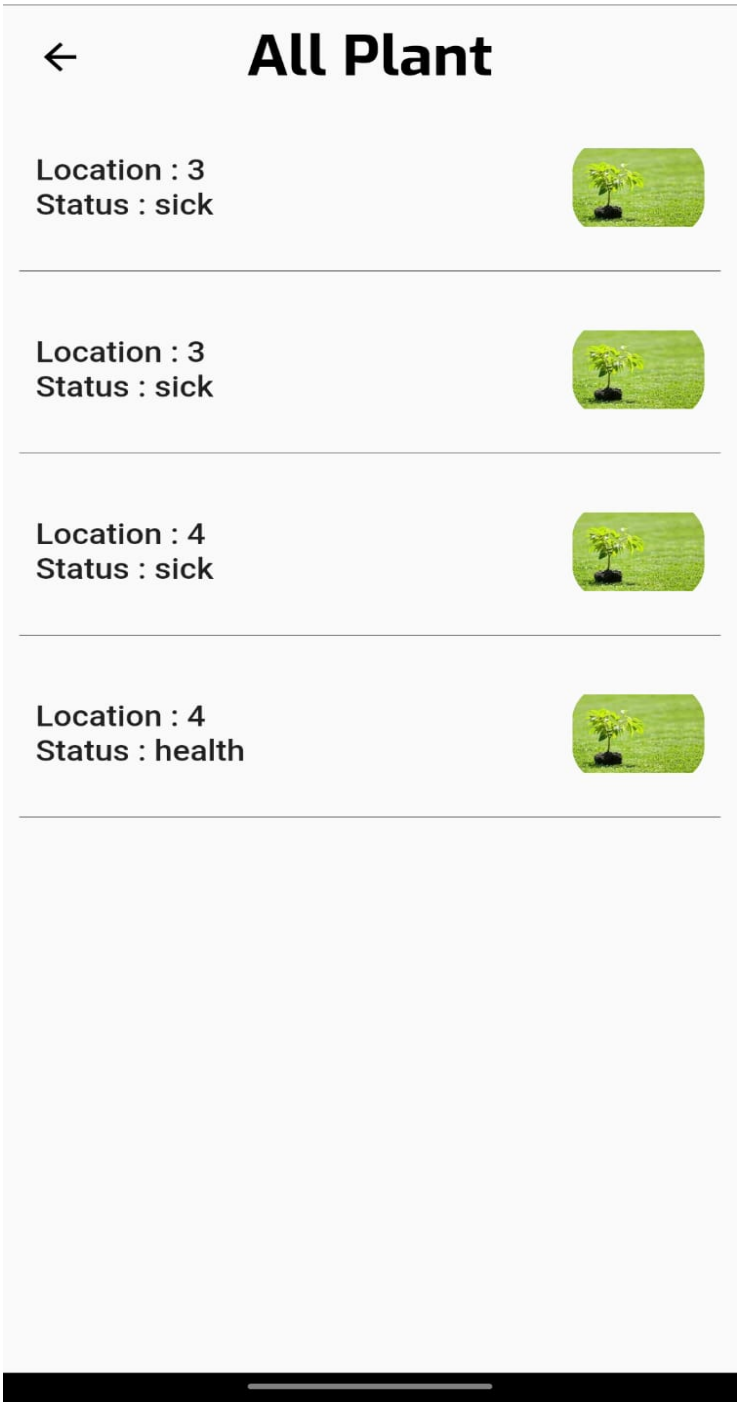


Fig 4.11 All Plants Screen

5. Health Plants Screen

Experience a screen seamlessly blending plant locations, vibrant imagery, and health status, offering a succinct yet comprehensive.

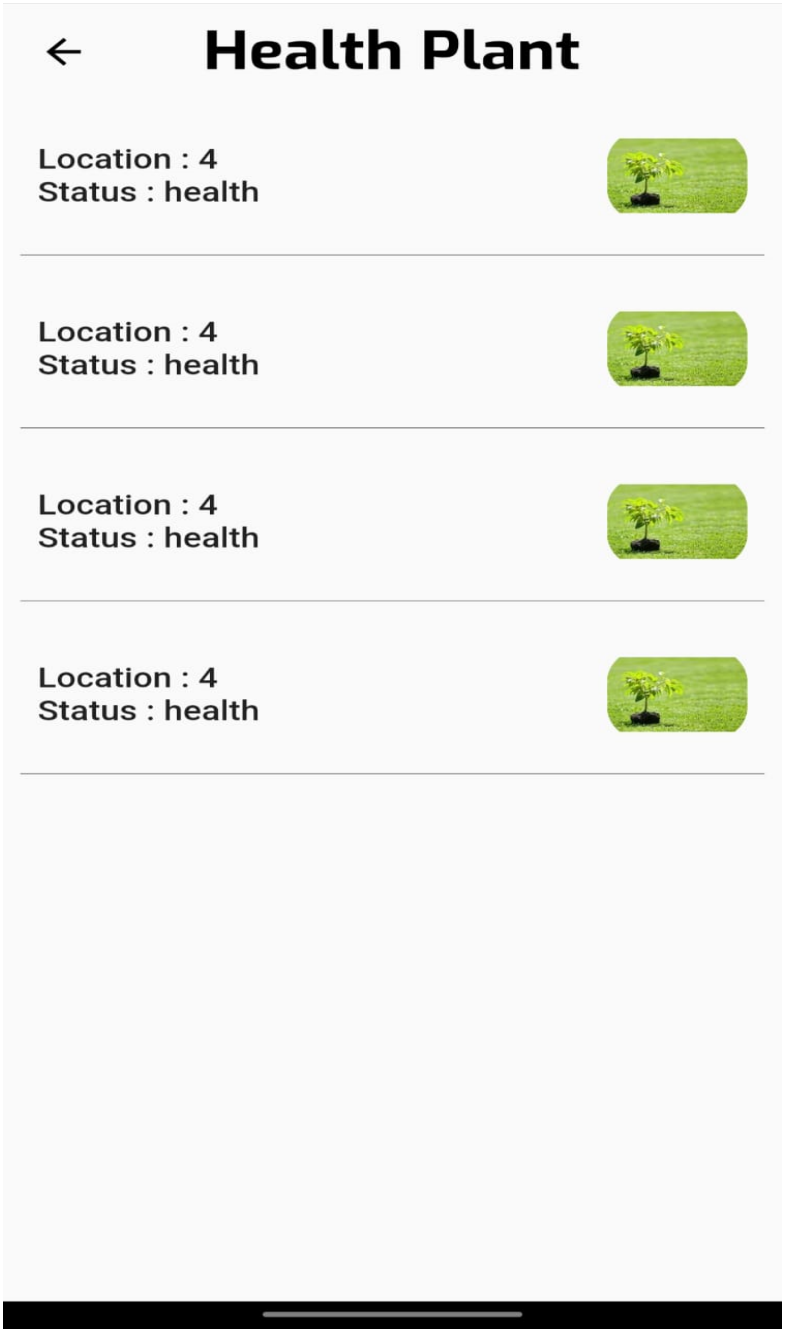


Fig 4.12 Health Plants Screen

6. Sick Plants Screen

screen seamlessly blending plant locations, vibrant imagery, and health status, providing a concise yet detailed insight into sick plant conditions for informed gardening decisions.

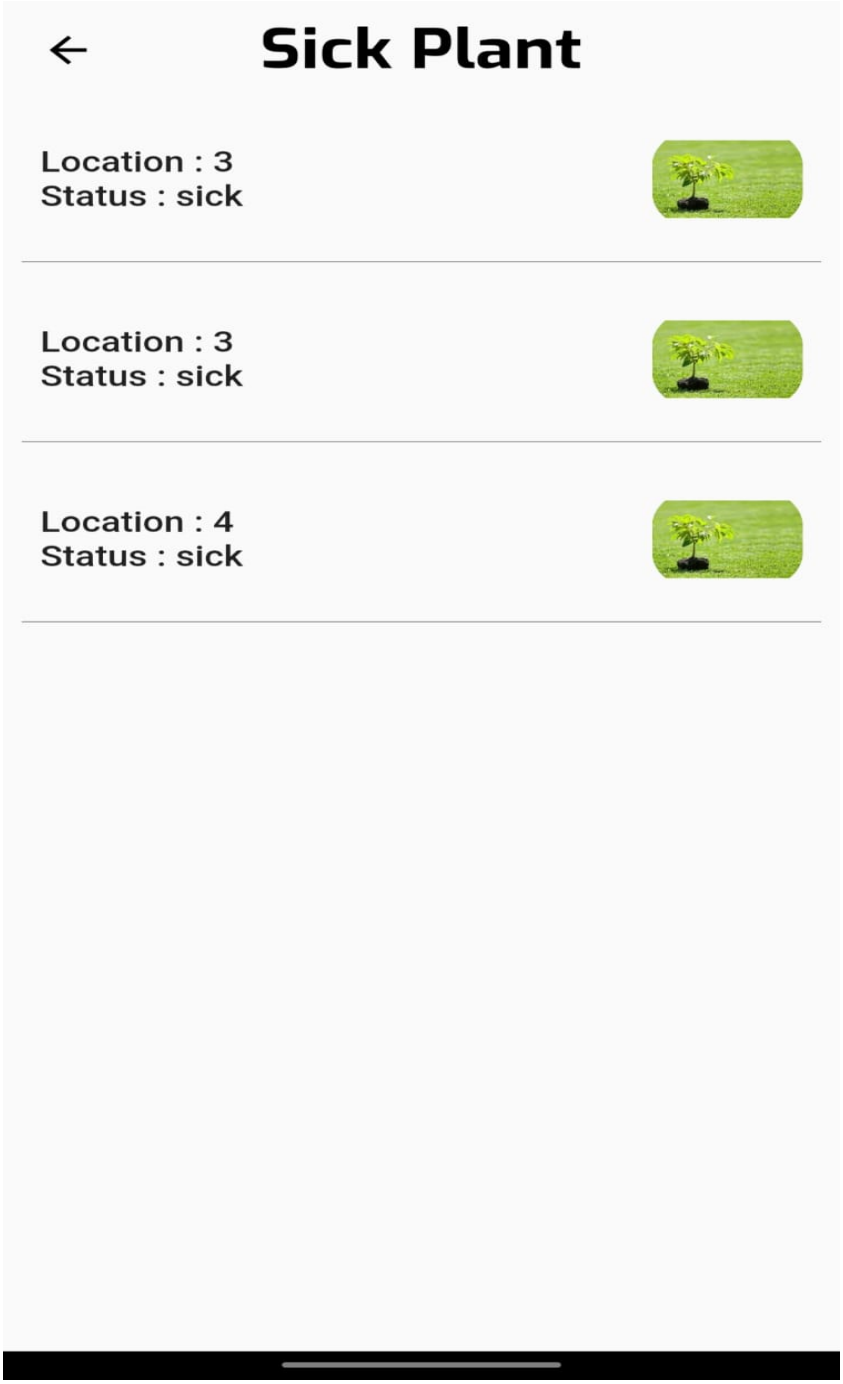


Fig 4.13 Sick Plants Screen

7. Plant Details Screen

screen detailing plant location, health status, treatment options, and accompanied by vivid imagery, providing users with a holistic view to effectively manage plant care.

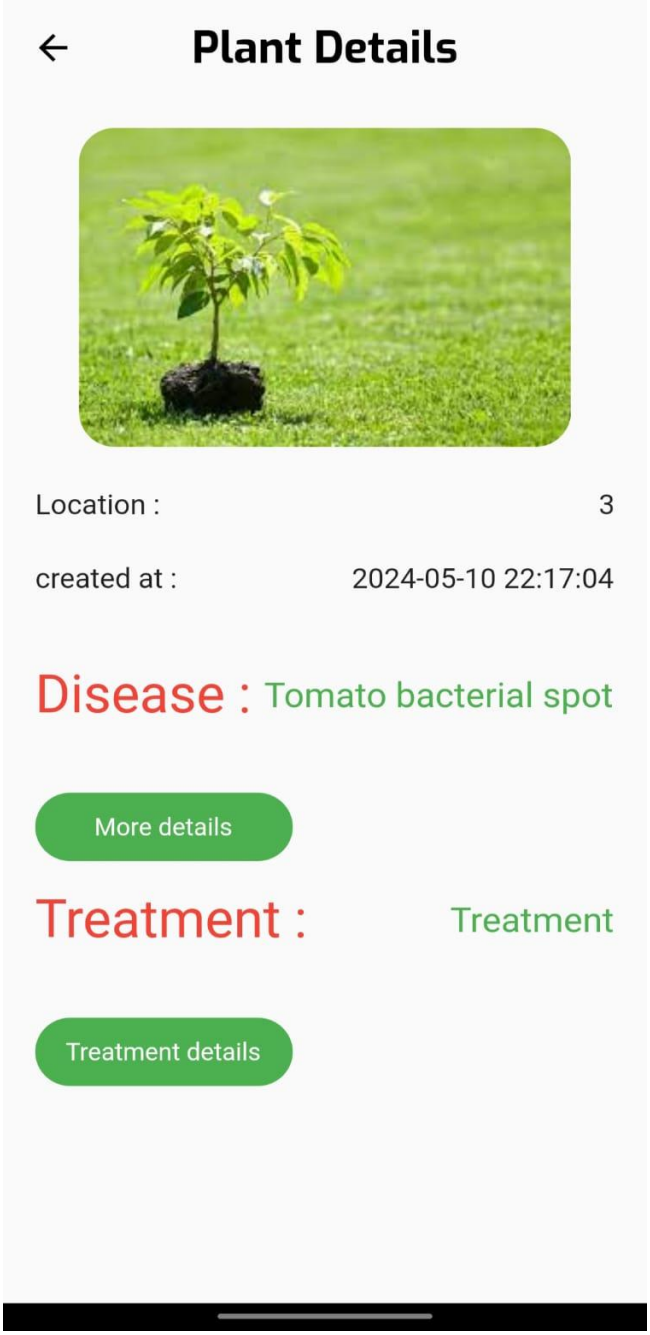


Fig 4.14 Plant Details Screen1

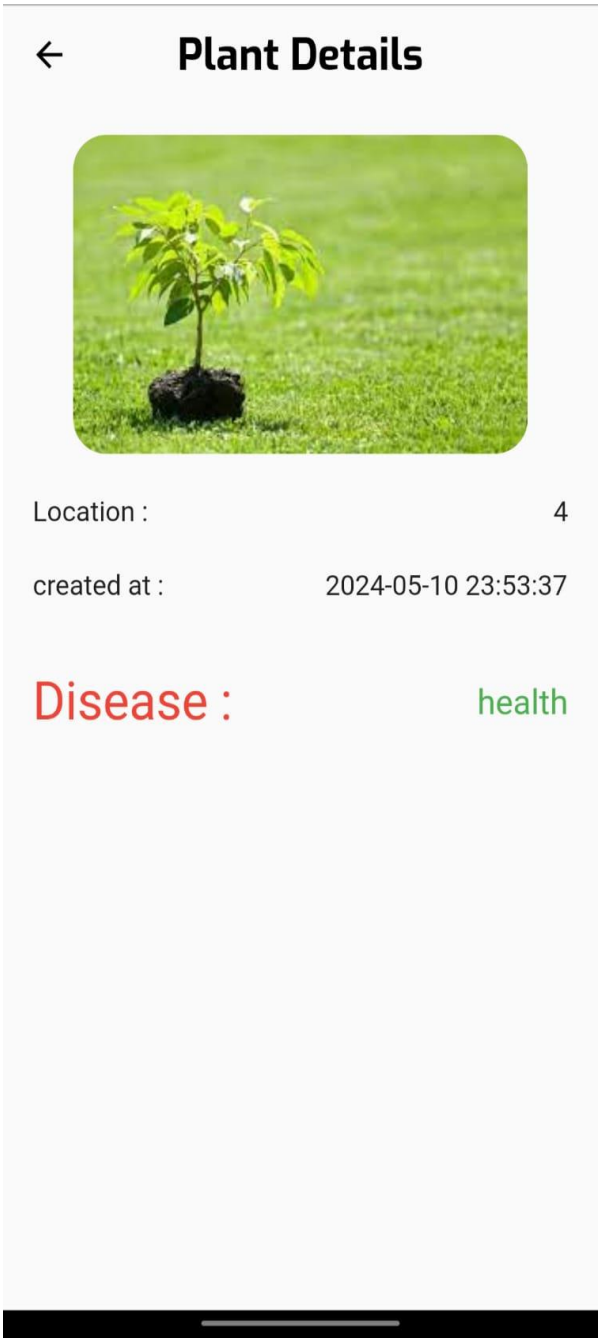


Fig 4.15 Plant Details Screen2

8. Disease Details Screen

screen dedicated to displaying diseases and their detailed information, offering an enriched experience tailored for in-depth understanding and informed decision-making in plant care management.

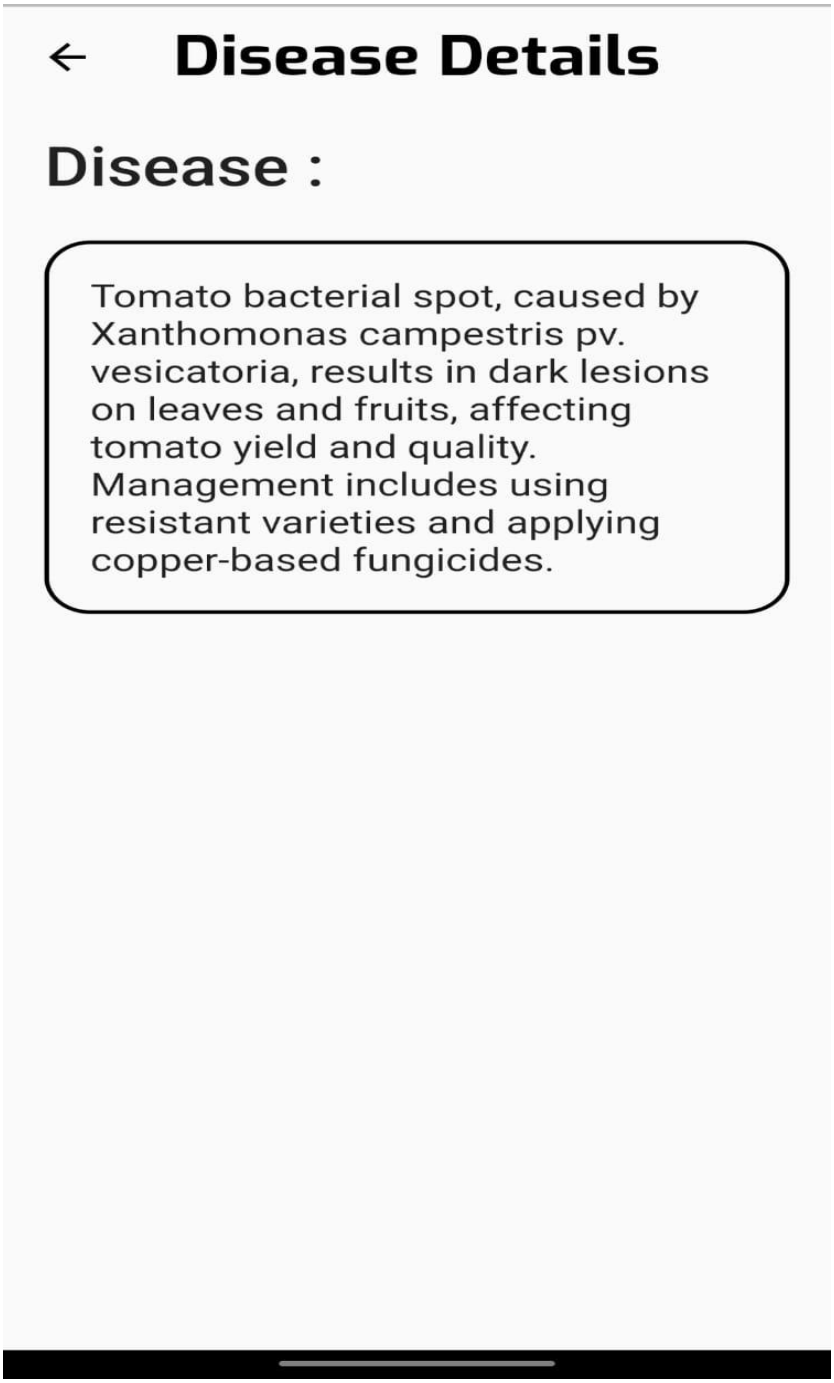


Fig 4.16 Disease Details Screen

9. Treatment Details Screen

screen showcasing detailed plant treatment information, thoughtfully presented to provide users with comprehensive insights for optimal plant care management.

Fig 4.17 Treatment Details Screen

4.1.2 Embedded System:

4.1.2.1 Introduction:

An agricultural robot is an autonomous device equipped with various components and sensors designed to assist in agricultural operations. This model incorporates a camera and a set of sensors to navigate agricultural land, track plants, and identify diseases. The robot captures images using the camera, processes them, and sends the results to the user through cloud-based communication.

4.1.2.2 How the robot works:

- When the robot commences its operation, it enters a preliminary phase wherein it remains in a standby state until the camera successfully establishes a connection with the server. This preparatory step ensures that the robot is fully equipped and ready to capture images at a moment's notice, ensuring seamless integration with the data processing system. Once connectivity is confirmed, the robot dynamically begins its traversal, methodically tracking the designated black line etched onto the ground surface. This line serves as a navigational guide, ensuring the robot maintains its trajectory and avoids straying from the intended path.

- As the robot advances along the pre-defined route, its sensors continuously monitor the environment, actively seeking out intersections where two black lines intersect. These intersections serve as critical checkpoints along the route, marking points of interest such as plant locations or designated stopping areas. Upon reaching an intersection, the robot promptly halts its forward motion, indicating a pause in its traversal.

- At this juncture, the robot swiftly engages its camera system, directing it toward the nearest plant situated adjacent to the intersection. With precision and accuracy, the camera captures a detailed image of the plant, capturing critical visual data for subsequent analysis and processing. This image is then transmitted to the server, where sophisticated algorithms and machine-learning techniques are employed to assess the plant's health status.

- Upon receipt of the image data, the server meticulously evaluates the plant's condition, analyzing various visual cues and indicators to determine whether the plant is healthy or infected. This comprehensive analysis encompasses factors such as coloration, texture, and morphology, providing valuable insights into the plant's

overall well-being.

- Once the assessment is complete, the server promptly transmits its findings back to the robot, conveying the plant's health status along with any pertinent recommendations or actions to be taken. Armed with this invaluable information, the robot adjusts its course of action accordingly, proceeding with its mission with heightened awareness and efficiency.

- With the task completed on one side of the designated route, the robot seamlessly transitions to the opposite side, repeating the same sequence of actions with unwavering precision. Upon fulfilling its duties on both sides of the route, the robot resumes its forward motion, gracefully navigating along the black line while remaining vigilant for future intersections and waypoints along its journey.

4.1.2.3 Components used:

1. 4x Car wheel:

These components enable the robot to move across the agricultural field, ensuring efficient coverage. (As shown in Fig 4.18)

Fig 4.18 Car wheel

2. 2x Car Chassis:

The main chassis of the robot that serves as the foundation and structural support for the robot's components. (As shown in Fig 4.19)

Fig 4.19 Car Chassis

3. 4x DC Motor:

These DC motors are used to drive the wheels of the robot, enabling controlled movement across the agricultural field [9]. (As shown in Fig 4.20)

Fig 4.20 DC Motor

- DC (Direct Current) motors convert electrical energy into mechanical rotation through the interaction of magnetic fields.

- DC motors operate within specific voltage and current ranges. Exceeding these limits can damage the motor.

- Motors require a stable power supply appropriate for their voltage and current ratings. Voltage regulators or motor drivers may be necessary for safe and efficient operation.

- Motors can be mounted using brackets, couplings, or other mechanical fixtures. Proper mounting ensures smooth and reliable operation.

- DC motors are used in a wide range of applications, including robotics, automation, electric vehicles, drones, and industrial machinery, due to their versatility and controllability.

4. Screws:

These screws are essential for assembling and securing various components together to create a sturdy and functional robot structure. (As shown in Fig 4.21)

Fig 4.21 Screws

5. Copper Spacer:

These copper spacers are used to increase the size of the chassis and create a gap between components for proper installation and ventilation. We also used it to mount the IR sensor to the chassis. (As shown in Fig 4.22)

Fig 4.22 Copper Spacer

6. L298N DC Motor driver:

The motor driver provides power and control for DC motors used in the robot's movement system [10]. (As shown in Fig 4.23)

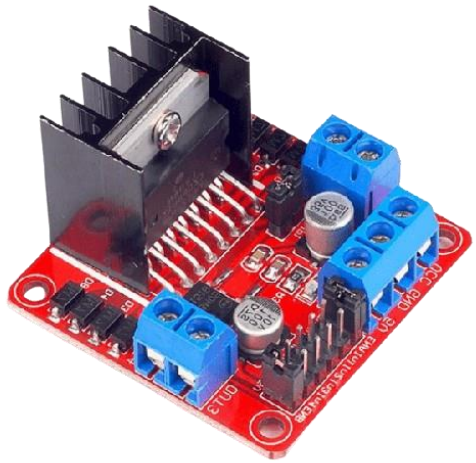


Fig 4.23 L298N DC Motor driver

- The L298N is a dual H-bridge motor driver IC, which means it can control two motors independently. It provides bidirectional control for DC motors and stepper motors, allowing forward and reverse rotation as well as speed control.

- The L298N contains two H-bridges, each consisting of four transistors arranged in an H configuration. This arrangement allows the motor driver to control the direction of current flow through the motor windings, enabling forward and reverse motion.

- The L298N is compatible with a wide range of DC motors and stepper motors, including brushed and brushless types. It can handle motor voltages up to 46V and continuous currents up to 2A per channel (with proper heat sinking).

- The L298N accepts input control signals to determine the motor direction (forward or reverse) and speed. It typically requires two digital control inputs per motor (one for direction and one for speed), allowing for precise motor control.

- The L298N includes built-in protection features such as thermal shutdown and overcurrent protection to prevent damage to the motor driver IC and connected motors under fault conditions.

- In addition to the L298N IC itself, the motor driver circuit typically requires external components such as diodes (for freewheeling protection), capacitors (for filtering), and resistors (for current sensing).

- The L298N requires two separate power supplies: one for the logic circuitry (typically 5V), and one for the motor(s) (up to 48V).

- The L298N can generate significant heat during operation, especially when driving high-current motors. Proper heat sinking and ventilation are necessary to prevent the IC from overheating.

- The L298N is commonly used in robotics projects, RC vehicles, CNC machines, 3D printers, and other applications requiring motor control. Its versatility, ease of use, and relatively low cost make it a popular choice among hobbyists and professionals alike.

While the L298N is a capable motor driver, it does have some limitations, including its

relatively high-power dissipation, limited current handling compared to more advanced motor drivers, and lack of support for advanced features such as microstepping (for stepper motors).

7. 3x 18650 Battery Holder:

These copper spacers are used to increase the size of the chassis and create a gap between components for proper installation and ventilation. We also used it to mount the IR sensor to the chassis. (As shown in Fig 4.24)

Fig 4.24 3x 18650 Battery Holder

8. 3x Li-Ion Battery 18650:

provides a high-capacity power source for the robot, enabling extended operation periods [11]. (As shown in Fig 4.25)

Fig 4.25 Li-Ion Battery 18650

- The 18650 battery is a common cylindrical lithium-ion battery format, named for its

dimensions of 18mm in diameter and 65mm in length. These batteries are widely used in various electronic devices due to their high energy density, rechargeability, and relatively compact size.

- Each individual 18650 lithium-ion battery typically has a nominal voltage of 3.7 volts. When three batteries are connected in series, the total voltage of the battery pack is multiplied by the number of batteries, resulting in a total nominal voltage of approximately 11.1 volts.
- The capacity of the battery pack remains the same as that of a single 18650 battery. However, the increased voltage allows for higher power output and longer operating times in devices that require higher voltages.

9. Male & Female DC Power Socket Jack prewired:

These prewired sockets are used for connecting power wires and providing a reliable power supply to the components. (As shown in Fig 4.26)

Fig 4.26 Male & Female DC Power Socket Jack prewired

10.Adapter Charger 12V:

This adapter charger is used to charge the rechargeable battery, providing a stable power source for the robot's operations. It can also supply power to the robot directly without battery. (As shown in Fig 4.27)

Fig 4.27 Adapter Charger 12V

11.2x Breadboard:

A rectangular board with a grid of holes into which electronic components can be inserted and connected without soldering. (As shown in Fig 4.28)

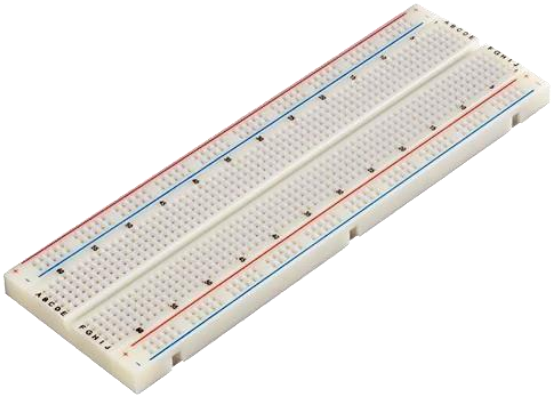


Fig 4.28 Breadboard

12.ESP32:

The ESP32 microcontroller serves as a foundational component in our project, providing a multitude of features and capabilities essential for its success. With its powerful dual-core processor, built-in Wi-Fi, and Bluetooth connectivity, the ESP32 enables seamless wireless communication and networking, facilitating real-time data exchange and control [12]. (As shown in Fig 4.29)

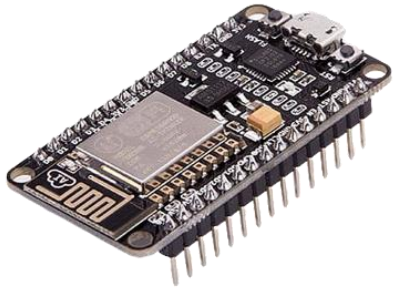


Fig 4.29 ESP32

The ESP32 is a powerful microcontroller developed by Espressif Systems and it is a successor to the ESP8266 microcontroller.
Here's a comprehensive overview of the ESP32:

- The ESP32 features a dual-core Tensilica Xtensa LX6 processor, which allows for multitasking and more complex applications. Each core can be individually controlled and programmed.

- One of the standout features of the ESP32 is its built-in Wi-Fi and Bluetooth capabilities. It supports both 2.4 GHz Wi-Fi (802.11 b/g/n) and Bluetooth 4.2 and Bluetooth Low Energy (BLE). These features enable IoT (Internet of Things) applications where devices need to communicate wirelessly.

- The ESP32 offers a wide range of peripheral interfaces including UART, SPI, I2C, I2S, PWM, ADC, and DAC, making it versatile for interfacing with various sensors, displays, actuators, and other external components.

- Despite its powerful features, the ESP32 is designed to be energy-efficient. It includes various power-saving modes and features such as multiple sleep modes, which can be crucial for battery-powered or energy-constrained applications.

- The ESP32 can be programmed using various development frameworks and languages, including the Arduino IDE (Integrated Development Environment), Espressif's native ESP-IDF (IoT Development Framework), MicroPython, and others. This flexibility makes it accessible to a wide range of developers.

- The ESP32 is widely used in various IoT applications including home automation, industrial automation, smart agriculture, wearables, environmental monitoring, and more. Its combination of processing power, wireless connectivity, and rich peripheral support makes it suitable for diverse projects.

- The ESP32 has a large and active community of developers and enthusiasts who contribute libraries, tutorials, and support resources. Espressif Systems also provides comprehensive documentation, forums, and software development kits (SDKs) to support developers working with the ESP32.

Overall, the ESP32 microcontroller serves as the backbone of our project, offering high performance, connectivity, and flexibility in a compact and cost-effective package. Its robust features and capabilities make it the ideal choice for powering our innovative solution for plant monitoring and disease detection in greenhouse environments.

13. USB Cable to Micro USB Cable:

This cable is used to connect and power the ESP32, and ESP32-CAM-MB enables us to upload code and firmware updates to the microcontroller, as well as debug and troubleshoot any issues that may arise during development. (As shown in Fig 4.30)

Fig 4.30 USB Cable to Micro USB Cable

14. Jumper Wires:

jumper wires are electrical wires used to create temporary connections between components on a breadboard. (As shown in Fig 4.31)

Fig 4.31 Jumper Wires

15.Infrared sensor (IR):

Infrared sensors are indispensable components for line tracing applications, providing reliable detection capabilities that enable precise navigation along predefined paths [13]. (As shown in Fig 4.32)

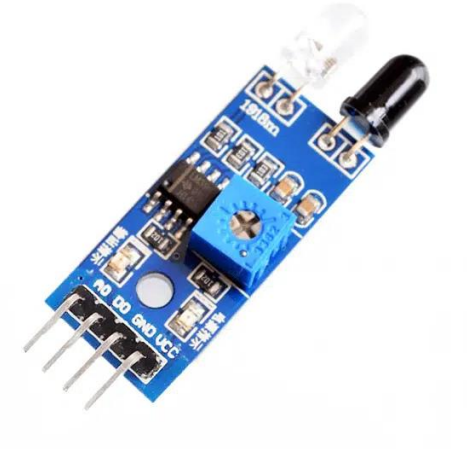


Fig 4.32 IR (infrared) sensor

Here's how an infrared sensor is utilized for line tracing purposes:

- Detection Principle: Infrared sensors used for line tracing typically consist of an infrared emitter and a receiver. The emitter emits infrared light onto the surface, while the receiver detects the reflected light. When the sensor is positioned over a contrasting surface (e.g., a dark line on a light-colored background), the reflected infrared light varies depending on whether the sensor is over the line or the background.
- Line Detection: As the sensor moves along the surface, it continuously detects the intensity of the reflected infrared light. When the sensor is positioned over the line, the intensity of the reflected light decreases due to absorption by the darker surface. Conversely, when the sensor is over the background, the intensity of the reflected light increases.
- Signal Processing: The output from the infrared sensor is processed by a microcontroller or other processing unit. Algorithms are used to analyze the sensor readings and determine the position of the line relative to the sensor. Based on this information, control signals can be generated to adjust the motion of a robotic vehicle or other automated system to follow the line.

16.SG90 Micro Servo Motor:

This servo motor is responsible for controlling the camera holder of the robot. It can rotate to adjust the position of the camera as needed [14]. (As shown in Fig 4.33)

Fig 4.33 SG90 Micro Servo Motor

- The SG90 servo motor operates based on the principle of PWM (Pulse Width Modulation) control. It consists of a DC motor mechanically linked to a gearbox and a potentiometer feedback mechanism. By sending PWM signals to the servo motor, the controller adjusts the position of the motor shaft to a specific angle within its range of motion.

- The SG90 servo motor typically accepts control signals in the form of PWM pulses with a frequency of around 50Hz. The duration of the pulses determines the position of the motor shaft, with pulse widths ranging from approximately 1ms to 2ms corresponding to the full range of motion (0 to 180 degrees).

- Servo motor operates in a closed-loop control system, where it compares the desired position (setpoint) with the actual position (feedback) and adjusts its output accordingly to minimize error.

- Servo motor offers high precision accuracy in positioning, making them ideal for applications requiring precise control.

- It can vary its speed based on the control signal, allowing for smooth and controlled motion.

17.ESP32-CAM:

ESP32-CAM is used to take clear images of plants, enabling the robot to monitor their growth and detect diseases [15]. (As shown in Fig 4.34)

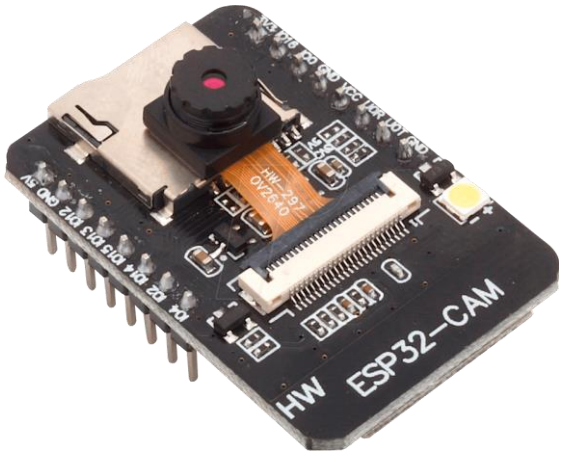


Fig 4.34 ESP32-CAM

- The ESP32-CAM integrates an ESP32-S chip, an OV2640 camera module, microSD card slot, and several GPIO pins onto a single board. It enables developers to easily add camera capabilities to their projects, making it suitable for applications such as surveillance cameras, video streaming, and image recognition.

- The ESP32-S chip is the heart of the ESP32-CAM board. It features a dual-core Tensilica LX6 processor, Wi-Fi and Bluetooth connectivity, hardware encryption, and various peripheral interfaces for sensor integration and communication.

- The ESP32-CAM is equipped with an OV2640 camera module, which is capable of capturing still images in a maximum resolution of 1600x1200 pixels (UXGA) and video at resolutions up to VGA (640x480 pixels). The camera module communicates with the ESP32 chip via the Serial Camera Control Bus (SCCB) interface.

- The board includes a microSD card slot, allowing for storage of images, videos, and other data captured by the camera. This enables standalone operation without the need for continuous connection to a host device.

- The ESP32-CAM exposes several GPIO pins, which can be used for interfacing with external components such as sensors, displays, or actuators. These pins support various communication protocols including UART, SPI, I2C, and digital input/output.

- The board can be powered via a micro-USB port or an external power source. It includes voltage regulators to provide stable power supply to the ESP32 chip and other components.

- The ESP32-CAM can be programmed using the Arduino IDE or other compatible development environments. Espressif provides libraries and example code to facilitate camera-related functions such as capturing images, recording video, and streaming content over Wi-Fi.

- The ESP32-CAM has some limitations to be aware of, including limited RAM and flash memory compared to other ESP32 development boards. Additionally, the OV2640 camera module has fixed focus and limited low-light performance.

18.ESP32-CAM-MB:

The ESP32-CAM-MB is a micro-USB (or recently, USB-C also) programmer that we attach to the ESP32-CAM board (also known as AI Thinker Cam Board). The ESP32-CAM-MB programmer is a shield for the ESP32-CAM, with the same form factor as the ESP32-CAM. It comes with a USB port that we connect directly to our computer. We can upload code to the ESP32-CAM board (or flash our ESP32-CAM), using the ESP32-CAM-MB programmer easily, without having to deal with manual wiring [16]. (As shown in Fig 4.35)

Fig 4.35 ESP32-CAM-MB

- The module features a USB interface, typically in the form of a micro-USB or USB Type-C connector. This interface allows the ESP32-CAM-MB USB module to be connected to a computer for programming, debugging, and data transfer purposes.

- The USB interface on the module enables serial communication between the ESP32-CAM board and the computer. This allows developers to upload firmware, monitor debug output, and communicate with the ESP32-CAM board using serial terminal software.

- The USB interface simplifies the programming process for the ESP32-CAM board. Developers can use tools such as the Arduino IDE or PlatformIO to write and upload firmware to the board via the USB connection. This eliminates the need for external programming hardware such as USB-to-serial converters.

- In addition to facilitating communication, the USB interface also provides power to the ESP32-CAM board. When connected to a computer via USB, the module can be powered directly from the USB port, eliminating the need for an external power supply in many cases.

19. Ultrasonic sensor:

Ultrasonic sensors are commonly used in robotics and automation applications for distance measurement, object detection, and navigation. These sensors emit ultrasonic sound waves, typically at frequencies above 20 kHz, and measure the time it takes for the sound waves to reflect off an object and return to the sensor. By calculating the time delay, the sensor can determine the distance to the object with high accuracy. (As shown in Fig 4.36)

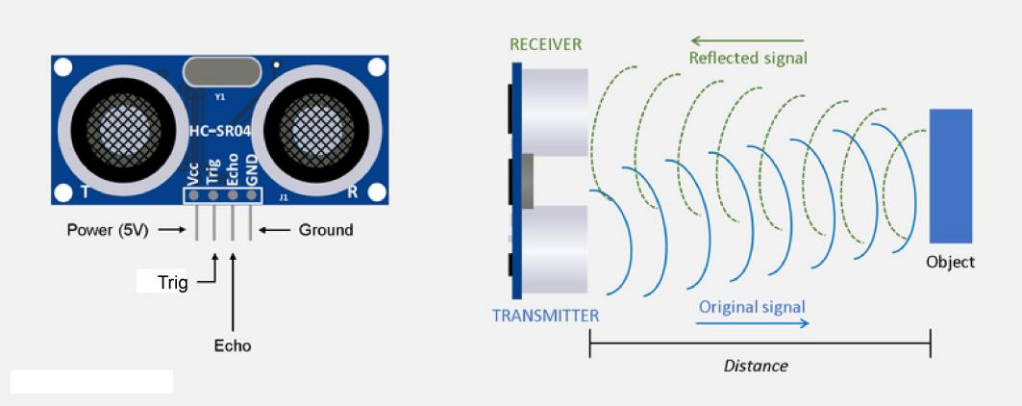


Fig 4.36 Ultrasonic sensor

4.1.2.4 Connection Steps:

In this section we will discuss the components used in the robot and how they are integrated with each other.

1. The wheels are connected to the DC motors, firmly mounted onto the robot's chassis.
2. The IR sensor is attached to the robot's body.
3. The motor shield is connected to the DC motors and ESP32, facilitating precise control over the robot's movement.
4. The ESP32 is securely affixed to the breadboard.
5. Sensors and motors are connected to ESP32 to control them.
6. The robot navigates via the IR sensors.
7. The servo is connected to its horn and the ESP32CAM is mounted on it from the top.
8. The ESP32CAM is connected to the ESP32 microcontroller.
9. The entire system draws power from a lithium battery, providing the necessary energy for the entire system.

4.1.2.5 Software:

❖ **Arduino IDE:**

The Arduino Integrated Development Environment (IDE) serves as a comprehensive software platform for developing and uploading code to Arduino-compatible microcontrollers, including the ESP32 and ESP32-CAM used in this project. It offers a user-friendly interface equipped with various tools and features tailored for both beginners and experienced developers [17]. (As shown in Fig 4.37)

Fig 4.37 Arduino IDE

Key Features of Arduino IDE:

- Code Editor: Arduino IDE provides a versatile code editor that supports the Arduino programming language, which is based on C and C++. The editor offers syntax highlighting, auto-indentation, and code completion features, enhancing the coding experience.

- Library Manager: The IDE includes a Library Manager, allowing users to easily install, manage, and update libraries for interfacing with external hardware components and sensors. This extensive library ecosystem simplifies the process of integrating additional functionality into projects.

- Serial Monitor: Arduino IDE features a built-in Serial Monitor tool that enables bidirectional communication between the microcontroller and the computer via the serial port. This tool is invaluable for debugging, monitoring sensor data, and displaying output messages during program execution.

- Board Manager: With the Board Manager, users can select the appropriate board variant, such as the ESP32 or ESP32-CAM, and install the necessary board definitions and drivers. This streamlines the process of configuring the IDE for specific hardware platforms.

- Upload Tool: Arduino IDE offers a straightforward upload tool that allows users to compile and upload their code directly to the target microcontroller. This seamless integration simplifies the deployment process and facilitates rapid prototyping and testing.

- Cross-Platform Compatibility: Arduino IDE is compatible with multiple operating systems, including Windows, macOS, and Linux, ensuring broad accessibility and ease of use across different platforms.

- Open-Source Community: Arduino IDE is supported by a vibrant open-source community of developers and enthusiasts who contribute to its ongoing development and enhancement. This collaborative ecosystem fosters innovation and knowledge sharing within the Arduino community.

4.1.3 Artificial Intelligence (AI):

4.1.3.1 Python:

Python, a versatile high-level programming language, has seen a significant rise in popularity due to its readability, extensive libraries, and cross-platform compatibility. This paper explores Python's core characteristics, diverse applications, and its pivotal role in advancing artificial intelligence (AI). We delve into Python's impact on machine learning, deep learning, and computer vision, culminating in a case study on image classification of plant diseases using advanced neural network models. Additionally, we discuss the future trajectory and potential challenges associated with Python in the AI domain. (As shown in Fig 4.38)

Fig 4.38 Python

- Core Characteristics:
  - Readability: Python's syntax resembles natural language, promoting code clarity and reducing maintenance costs.
  - Interpreted Language: Python code is executed line by line, eliminating the need for explicit compilation, fostering a faster development cycle.
  - Cross-Platform Compatibility: Python programs can run on various operating systems without modification, enhancing portability.
  - Extensive Libraries: Python boasts a vast collection of third-party libraries, offering pre-built functionalities for diverse tasks, from data analysis (NumPy, Pandas) to web development (Django, Flask) and scientific computing (SciPy, Matplotlib).
- Applications of Python:
  - Data Science: Python’s rich ecosystem of libraries empowers data scientists to clean, analyze, and visualize data efficiently.
  - Web Development: Frameworks like Django and Flask enable rapid web application development, making Python a popular choice for web development projects.

- Scientific Computing: Python's numerical computing libraries facilitate complex scientific calculations and simulations.
- Machine Learning: Python is a prominent language in the field of machine learning, with libraries like TensorFlow and PyTorch supporting the development of intelligent systems.
- Automation: Python's scripting capabilities are well-suited for automating repetitive tasks, improving efficiency across various domains.

4.1.3.2 AI With Python:

Python provides a clear and readable syntax hence provides a smooth path to learn and build intelligent models without complex code structures. The best part of using Python is its rich ecosystem of libraries and frameworks specially tailored for AI and machine learning. Python has strong community of AI enthusiasts, researchers and developers who share knowledge, insights and resources. The collaborative spirit of the Python AI community ensures that help is always within reach.
(As shown in Fig 4.39)

Fig 4.39 AI With Python

- Machine Learning
  - Supervised Learning: Algorithms are trained on a labeled dataset. Applications include classification and regression tasks.
  - Regression Algorithms: Linear Regression, Polynomial Regression, Support Vector Regression (SVR)
  - Classification Algorithms: Logistic Regression, Decision Trees, Ensemble Classifiers, Support Vector Machines (SVM), k-Nearest Neighbors (kNN), Naive Bayes
  - Unsupervised Learning: The algorithm provides unlabeled data and is tasked with finding patterns or relationships. Applications include clustering and

dimensionality reduction.

- Clustering Algorithms: K-means, Hierarchical Clustering, DBSCAN
- Dimensionality Reduction: Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), Linear Discriminant Analysis (LDA)
- Reinforcement Learning: The algorithm learns by interacting with an environment and receiving feedback. Applications include game playing, robotics, autonomous systems.
- Algorithms: Q-learning, Model-Based Reinforcement Learning, Deep Q Network (DQN), REINFORCE, Actor-Critic, Monte Carlo Policy Evaluation, SARSA (State-Action-Reward-State-Action).

- Deep Learning
  - Fundamentals of Deep Learning:
    Gradient Descent Algorithm, Backpropagation, Hyperparameters, Activation Functions, Epochs, Loss Function, Optimizers, Batch Size, Learning Rate

  - Deep Learning Architectures:
    - Feedforward Neural Networks (FNN)
    - Multi-Layer Perceptron (MLP)
    - Convolutional Neural Networks (CNN)
    - Recurrent Neural Networks (RNN)
    - Long Short-Term Memory (LSTM) networks
    - Gated Recurrent Units (GRU)
    - Autoencoders
    - Capsule Networks

- Case Study: Image Classification of Plant Diseases
- Data Collection: High-quality annotated datasets are fundamental for training deep learning models. Publicly available datasets like Plant Village provide a comprehensive collection of images covering a wide range of plant species and diseases.

❖ **We have 6 classes in our dataset:**

- Potato Early blight

Fig 4.40 Potato Early blight

- Potato late blight

Fig 4.41 Potato late blight

- Healthy potato

Fig 4.42 Healthy potato

- Tomato Bacterial spot

Fig 4.43 Tomato Bacterial spot

- Tomato Early blight

Fig 4.44 Tomato Early blight

- Tomato healthy

Fig 4.45 Tomato Healthy

4.1.3.3 Convolutional Neural Networks:

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision, achieving state-of-the-art performance in a wide range of tasks, including image classification, object detection, image segmentation, and even facial recognition. Their ability to automatically learn features from data, coupled with their inherent understanding of spatial relationships, makes them uniquely suited for analyzing visual information. This paper will delve into the inner workings of CNNs, exploring their architecture, key components, and training process. We will discuss the advantages CNNs hold over traditional methods and explore their extensive applications across various domains. Finally, we will address the challenges associated with CNNs and future research directions in this exciting field. **(As shown in Fig 4.46)**

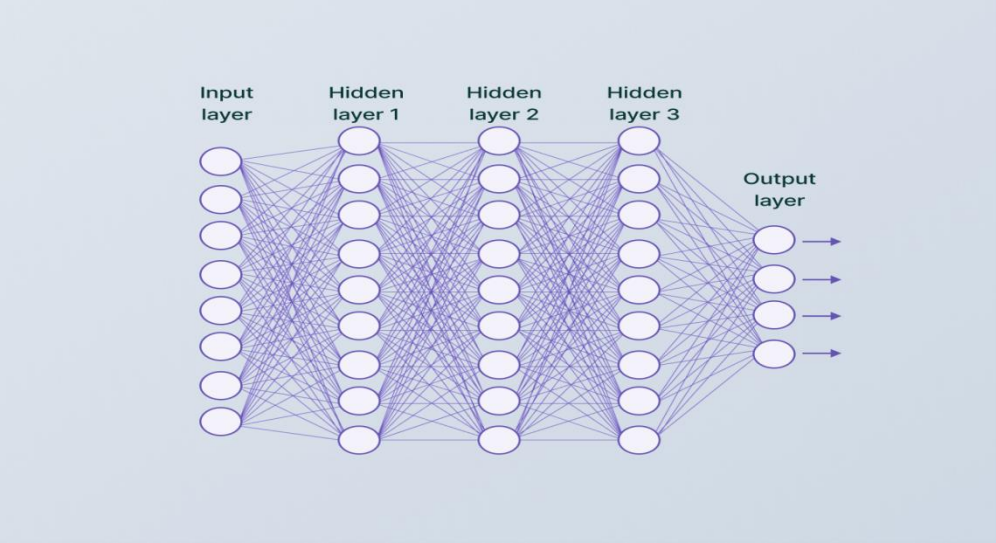


Fig 4.46 Convolutional Neural Networks

- Architecture of a CNN:

  A typical CNN architecture consists of several alternating layers:

  - Convolutional Layer: The core building block of a CNN, the convolutional layer applies a filter (kernel) to the input data, extracting features like edges, lines, and shapes. This process is repeated using multiple filters, generating feature maps.
  - Pooling Layer: This layer downsamples the feature maps, reducing the dimensionality of data while preserving important features. Techniques like max pooling select the highest value within a local region.
  - Fully Connected Layer: Similar to traditional neural networks, fully connected layers perform classification tasks based on the extracted features.

- Training a CNN:

Training a CNN involves iteratively feeding labeled data (images with corresponding classifications) and adjusting the network weights through an optimization algorithm like backpropagation. During each iteration, the CNN calculates the difference between the predicted output and the actual label (error). The backpropagation algorithm then propagates this error back through the network, updating the weights to minimize the error in future predictions.

- Advantages of CNNs
  - Feature Extraction: CNNs excel at automatically learning relevant features from raw data, eliminating the need for hand-crafted features, a laborious and domain-specific task.
  - Spatial Relationships: The convolutional layers inherently capture spatial relationships between pixels, making CNNs ideal for tasks where image structure is crucial.
  - Parameter Sharing: By sharing weights across filters, CNNs promote efficient learning and reduce the number of parameters to be trained.

- Applications of CNNs

  CNNs have permeated various fields:
  - Computer Vision: Image classification (e.g., recognizing objects in images), object detection (e.g., identifying pedestrians in traffic), image segmentation (e.g., segmenting a tumor in a medical scan).
  - Natural Language Processing: Sentiment analysis, text classification.
  - Recommender Systems: Personalizing product recommendations based on user image preferences.

- CNN Testing:

  Testing a CNN model is a crucial step in evaluating its performance and identifying areas for improvement.

Here's a breakdown of the testing process:

1- Data Preparation:
   - Test Set: Set aside a portion of your labeled data specifically for testing. This data should not be used during training to ensure unbiased evaluation.
   - Preprocessing: Ensure the test data undergoes the same preprocessing steps (resizing, normalization) as the training data.

2- Model Evaluation Metrics: The choice of metric depends on the specific task your CNN is designed for:
   - Classification: Accuracy, precision, recall, F1-score. These metrics compare the

model's predicted labels with the actual labels in the test set.

- Object Detection: Mean Average Precision (mAP) is a common metric that considers both how many objects the model correctly identified (precision) and how many it missed (recall).
- Segmentation: Intersection over Union (IoU) measures the overlap between the model's predicted segmentation mask and the ground truth mask.

3- Testing Process:

- Feedforward Pass: Run the test data through the trained CNN model.
- Evaluation: Calculate the chosen metrics based on the model's predictions and the actual labels in the test set.

4- Analysis and Interpretation:

- Analyze the evaluation metrics to understand the model's strengths and weaknesses.
- Look for patterns in errors to identify areas for improvement.
- Visualize results (confusion matrix for classification, bounding boxes for detection) to gain insights into model behavior.

4.1.3.4 Preprocessing:

Image preprocessing techniques such as normalization, augmentation, and resizing are employed to enhance the training process. Augmentation techniques like rotation, flipping, and zooming help in creating a more robust model by artificially increasing the dataset size.

4.1.3.5 YOLO for Leaf Detection and Cropping

- Model Architecture and Usage:

  YOLO (You Only Look Once) is an advanced real-time object detection system that has been widely adopted for various applications due to its speed and accuracy. YOLO divides the image into a grid and predicts bounding boxes and probabilities for each grid cell simultaneously, making it highly efficient for real-time object detection.

- Application in Plant Disease Detection:

  In the context of plant disease detection, YOLO can be employed to detect and crop leaves from images, which can then be fed into a classification model for disease identification. The process involves the following steps:

  1. Data Collection: Gather a diverse set of images containing leaves from various plants. Publicly available datasets like PlantVillage can be used for this purpose.

2. Model Training:
    - Annotation: Manually annotate the images to create bounding boxes around the leaves. This annotated data is used to train the YOLO model.
    - Training: Use the annotated dataset to train the YOLO model.

    The training process involves feeding the images and their corresponding annotations into the YOLO network, which learns to predict bounding boxes around the leaves.

3. Detection and Cropping:
    - Detection: Once trained, the YOLO model can detect leaves in new images. The model outputs bounding boxes around detected leaves with high confidence scores.
    - Cropping: The detected bounding boxes are used to crop the leaves from the original images. This step isolates the leaves, which can then be used as inputs for further analysis.

- Implementation Details
    - YOLOv8 for Detection: In this study, we utilize YOLOv8, an improved version of YOLO with enhanced accuracy and speed. YOLOv8's architecture allows for precise detection even in complex and cluttered backgrounds, making it ideal for detecting leaves amidst dense foliage.
    - Cropping Technique: The coordinates of the bounding boxes output by YOLOv8 are used to crop the detected leaves. This is achieved by slicing the original image array using the bounding box coordinates, ensuring that only the relevant portions of the image (the leaves) are extracted.

Fig 4.47 YOLO8 for Leaf Detection and Cropping

- Advantages of Using YOLO for Leaf Detection
  - Real-Time Performance: YOLO's ability to process images in real-time makes it suitable for applications requiring quick and accurate detection.
  - High Accuracy: YOLO's grid-based approach and simultaneous prediction of bounding boxes and class probabilities result in high detection accuracy.
  - Robustness: YOLO is robust to variations in lighting, occlusion, and complex backgrounds, ensuring reliable leaf detection in diverse conditions.
  - By employing YOLO for leaf detection and cropping, the subsequent classification model can focus solely on the relevant parts of the image, thereby improving the accuracy and efficiency of plant disease identification.

4.1.3.6 Custom CNN for Classification:

A custom-trained CNN model classifies the cropped leaf images into different disease categories. The model is trained using transfer learning from a pre-trained network such as ResNet or Inception, followed by fine-tuning on the specific plant disease dataset.

- Image Recognition Architectures

  Models created for identifying and categorizing objects within images include:
  - Object Detection Architectures
  - Two-stage Detectors: R-CNN, Fast R-CNN, Faster R-CNN, Cascade R-CNN
  - Single Shot Detectors: YOLO, SSD
  - Image Segmentation Architectures: U-Net, K-means Clustering, Mask R-CNN, YOLOv8, Cascade Mask R-CNN, PSPNet

4.1.4 Back End:

First, the technology we used in this part of the project is crucial. We faced challenges choosing the backend framework and workflow that best fit our needs.

4.1.4.1 Why PHP?

PHP, created by Rasmus Lerdorf in 1993 and released in 1995, is a popular server-side scripting language with a 77.4% market share. Its versatility makes it ideal for our project, despite not being as modern as frameworks like Laravel or Node.js. PHP effectively meets all our server-side requirements.

4.1.4.2 System Flow

Here’s how our system works:

1. Embedded System Interaction: The embedded system detects diseased or healthy plants, sends images to Firebase, and forwards the image link to the server. The server downloads the image and saves it in the database with the plant's location.
2. Image Processing: The server converts the image to a base64 string to efficiently send it to the AI model for disease detection.
3. AI Model Response: The AI model returns the disease type (if any) and the confidence level. The server then instructs the embedded system to continue scanning and updates the mobile app to display the disease information.

This cycle repeats until all plants are scanned. These steps detail the server's role in our project.

Before we get into the code we must know how PHP works with HTTP/HTTPS requests, what is JSON, and finally how we managed all of connections between other sections.

4.1.4.3 PHP $_REQUEST [ ]

PHP contains some predefined arrays one of them is the $_REQUEST associative array that by default contains the contents of $_GET, $_POST and $_COOKIE associative arrays, this is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. There is no need to do global $variable; to access it within functions or methods.

The variables in $_REQUEST is provided to the script via the GET, POST, and COOKIE input mechanisms and therefore can be modified by the remote user and cannot be trusted. The presence and order of variables listed in this array is defined according to the PHP request order, and variables order configuration directives.

To clarify what $_REQUEST can do here is the code which tries some of the attributes of this predefined array:

```php
<?php

$_GET['foo'] = 'a';
$_POST['bar'] = 'b';
var_dump($_GET); // Element 'foo' is string(1) "a"
var_dump($_POST); // Element 'bar' is string(1) "b"
var_dump($_REQUEST); // Does not contain elements 'foo' or 'bar'



if ($_REQUEST['currency']) # change currency on user request
{
    $currency = $_REQUEST['currency']; # use it
    setcookie('currency', $_REQUEST['currency'], 0, 'eshop.php'); # store it
}
else # use default currency
{
    $currency = 'USD';
}

# display shop contents with user selected currency
echo 'All prices are shown in ', $currency;

# let the user switch currency
echo '<a href="eshop.php?currency=USD">Switch to USD</a>';
echo '<a href="eshop.php?currency=EUR">Switch to EUR</a>';
?>

?>



<?php

switch($_SERVER['REQUEST_METHOD'])
{
case 'GET': $the_request = &$_GET; break;
case 'POST': $the_request = &$_POST; break;
default:
}
?>
```

Fig 4.48 PHP $_REQUEST [ ]

4.1.4.4 $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.
The example below shows how to use some of the elements in $_SERVER:

Fig 4.49 $_SERVER

This global variable can help us identify some of the problems along the development or testing phases specifically for the server problems.

4.1.4.5 $_FILES

After we demonstrated the $_SERVER and $ _REQUEST variables we need to send data any kind of data we can use the $_FILES array to help us fix errors and bugs related to the files when receiving or sending.
When a file is uploaded through a form, PHP populates the $_FILES array with information about the uploaded file. This array contains several elements that provide details such as the file's name, type, temporary location on the server, error codes (if any), and the file's size.

To process an uploaded file using $_FILES, you typically use its associative array structure. For instance, if you have a form field named <input type="file" name="uploadFile">, you can access the uploaded file's details like this
Example:

```php
<?php
echo "Filename: " . $_FILES['file']['name']."<br>";
echo "Type : " . $_FILES['file']['type'] ."<br>";
echo "Size : " . $_FILES['file']['size'] ."<br>";
echo "Temp name: " . $_FILES['file']['tmp_name'] ."<br>";
echo "Error : " . $_FILES['file']['error'] . "<br>";
?>
```

Fig 4.50 $_ FILES

Output
Filename: hello.html
Type: text/html
Size: 56
Temp name: C:\xampp\tmp\php32CE.tmp
Error: 0

Now we know the basic structure to use the PHP for the server side, moreover we should see how we can use these variables to our on good.

4.1.4.6 Database

It's commonly known if I used PHP, the database will be MariaDB, we could use SQL LITE but for more convenient development its better to use the default database in the XAMPP, which is easier in installation, XAMPP framework containing all the tools we need for the project.

4.1.4.7 XAMPP

Is a free and open-source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the A pache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages. Since most actual web server deployments use the same components as XAMPP, it makes transitioning from a local test server to a live server possible.

- XAMPP's ease of deployment means a WAMP or LAMP stack can be installed quickly and simply on an operating system by a developer.
- XAMPP is regularly updated to the latest releases of Apache, MariaDB, PHP and Perl. It also comes with several other modules, including OpenSSL, phpMyAdmin, Media Wiki, Joomla, WordPress and more. Self-contained,

multiple instances of XAMPP can exist on a single computer, and any given instance can be copied from one computer to another. XAMPP is offered in both a full and a standard version (Smaller version).

5. Chapter Five

5.1 Implementation

5.1.1 Mobile Application:

❖ Code:

- Register and login API

```dart
import 'package:graduation_project/API/api_helper.dart';
import 'package:graduation_project/main.dart';
import 'api.dart';

class LoginApi{
  Api api = Api();
  Future <dynamic> login({required email ,required password})async{
    try{
      var response = await api.postRequest(
          '${ApiHelper.baseUrl}${ApiHelper.login}', {"email": email, "password": password});
        getStorage.write('password', 'password');
      print(response);
      return response ;
    }catch (e){
      print(e);
    }
  }
}
```

Fig 5.1 Login API

```dart
import '../main.dart';
import 'api.dart';
import 'api_helper.dart';

class RegisterApi{
  Api api = Api();
Future <dynamic> register({required userName ,required email ,required password})async{
  try{
    var response = await api.postRequest('${ApiHelper.baseUrl}${ApiHelper.register}', {
      "username": userName,
      "email": email,
      "password": password
    });
    getStorage.write('password', 'password');
    print(response);
    return response;
  }catch (e){
    print(e);
  }
}
}
```


Fig 5.2 Register API

- Plant Model

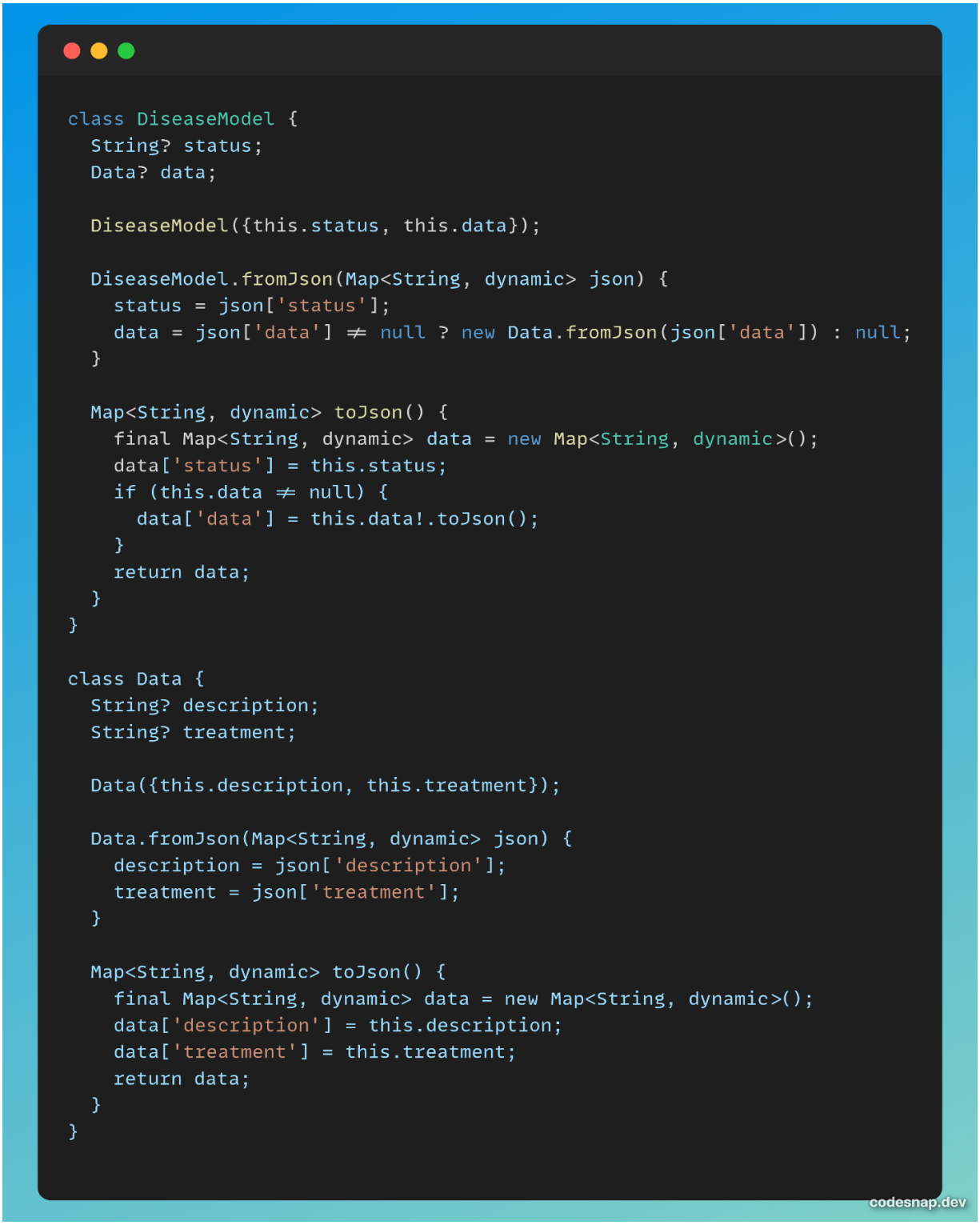
```dart
class PlantModel {
  String? status;
  List<PlantData>? data;

  PlantModel({
    this.status,
    this.data,
  });

  factory PlantModel.fromJson(Map<String, dynamic> json) {
    List<dynamic>? dataList = json['data'];
    List<PlantData>? parsedData =
    dataList?.map((data) => PlantData.fromJson(data)).toList();

    return PlantModel(
      status: json['status'],
      data: parsedData,
    );
  }
}

class PlantData {
  int? requestID;
  String? firebaseLink;
  String? location;
  String? timestamp;
  String? diseaseName;

  PlantData({
    this.requestID,
    this.firebaseLink,
    this.location,
    this.timestamp,
    this.diseaseName,
  });

  factory PlantData.fromJson(Map<String, dynamic> json) {
    return PlantData(
      requestID: json['requestID'],
      firebaseLink: json['firebaseLink'],
      location: json['location'],
      timestamp: json['timestamp'],
      diseaseName: json['diseasename'],
    );
  }
}
```


Fig 5.3 Plant Model

- Get Plant API

Fig 5.4 Get Plant API

- Disease Model



```dart
class DiseaseModel {
  String? status;
  Data? data;

  DiseaseModel({this.status, this.data});

  DiseaseModel.fromJson(Map<String, dynamic> json) {
    status = json['status'];
    data = json['data'] != null ? new Data.fromJson(json['data']) : null;
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = new Map<String, dynamic>();
    data['status'] = this.status;
    if (this.data != null) {
      data['data'] = this.data!.toJson();
    }
    return data;
  }
}

class Data {
  String? description;
  String? treatment;

  Data({this.description, this.treatment});

  Data.fromJson(Map<String, dynamic> json) {
    description = json['description'];
    treatment = json['treatment'];
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = new Map<String, dynamic>();
    data['description'] = this.description;
    data['treatment'] = this.treatment;
    return data;
  }
}
```


Fig 5.5 Disease Model

- Get Disease Api

Fig 5.6 Get Disease API

5.1.2 Embedded System:

❖ Code:

- To ensure that it does not move forward until it receives the start signal from the camera.

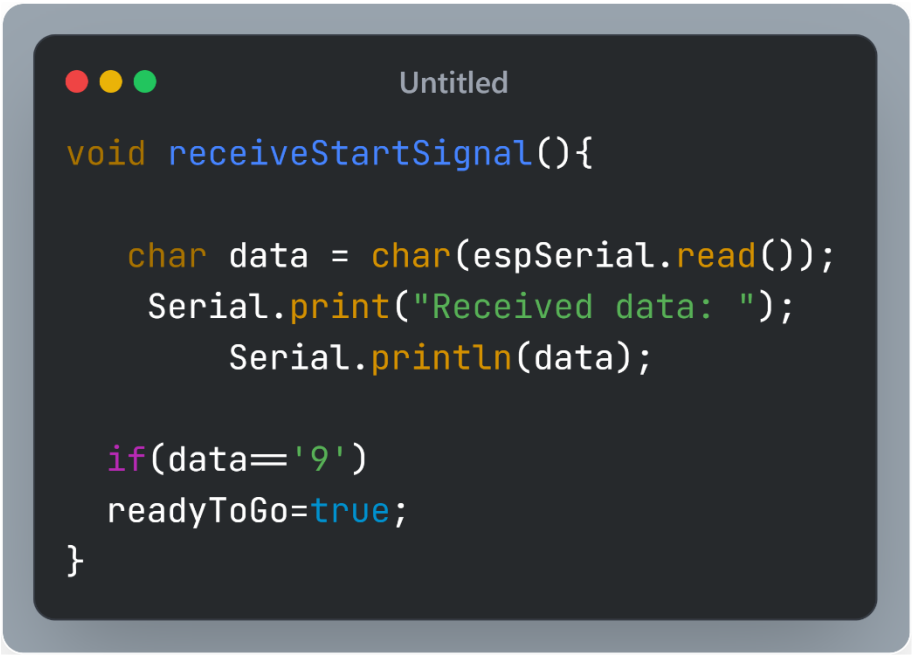


Fig 5.7 ReceiveStartSignal

- To Track Line

Fig 5.8 GoForword

- When the robot detects the intersections representing plant locations, the robot immediately stops its forward movement, moves the servo motor to the plant location (left/right) and sends a request to the camera to take the photo.

Fig 5.9 Stop and send the request

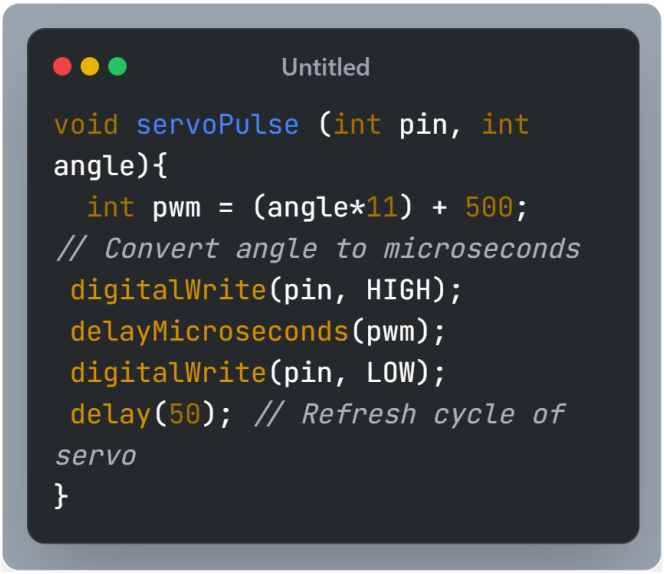


Fig 5.10 ServoMotor

- Connecting Camera to WIFI

```
Untitled
void initWiFi(){
  WiFi.begin(ssid, password);
  while (WiFi.status() ≠ WL_CONNECTED)
{
    delay(1000);
    Serial.println("Connecting to
WiFi...");
  }
    Serial.println("WiFi Connected");

}
```

Fig 5.11 WIFI

- Initialize File system

```
Untitled
void initLittleFS(){
  if (!LittleFS.begin(true)) {
    Serial.println("An Error has
occurred while mounting LittleFS");
    ESP.restart();
  }
  else {
    delay(500);
    Serial.println("LittleFS mounted
successfully");
  }
}
```

Fig 5.12 initialize File system

- Capture photo and save it to file system

```
Untitled
void capturePhotoSaveLittleFS( void ) {
  // Dispose first pictures because of bad quality
  camera_fb_t* fb = NULL;
  // Skip first 3 frames (increase/decrease number as needed).
  for (int i = 0; i < 4; i++) {
    fb = esp_camera_fb_get();
    esp_camera_fb_return(fb);
    fb = NULL;
  }

  // Take a new photo
  fb = NULL;
  fb = esp_camera_fb_get();
  if(!fb) {
    Serial.println("Camera capture failed");
    delay(1000);
    ESP.restart();
  }

  // Photo file name
  Serial.printf("Picture file name: %s\n", filePath);
  File file = LittleFS.open(filePath, FILE_WRITE);

  // Insert the data in the photo file
  if (!file) {
    Serial.println("Failed to open file in writing mode");
  }
  else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.print("The picture has been saved in ");
    Serial.print(filePath);
    Serial.print(" - Size: ");
    Serial.print(fb->len);
    Serial.println(" bytes");
  }
  // Close the file
  file.close();
  esp_camera_fb_return(fb);
}
```

Fig 5.13 CapturePhotoSaveLittleFS

- Upload the captured photo to the Firebase Storage

```
Firebase.Storage.upload(&fbdo, STORAGE_BUCKET_ID
,filePath   , mem_storage_type_flash ,  filePhoto ,
"image/jpeg" ,fcsUploadCallback);
```

Fig 5.14 UpToFirebase

- Upload the link to the database

```
void uplodePic() {
    digitalWrite(FLASH_GPIO_NUM, LOW);

  String queryString= String("firebase_link=") +
String(urlencode(firebase_link))+ String("&location=") + String(location++);

  Serial.printf("\nDownload URL: %s\n", urlencode(firebase_link));

   HTTPClient http;

  http.begin(HOST_NAME + PATH_NAME);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");

 int httpCode = http.POST(queryString);

  if(httpCode > 0) {

    if(httpCode == HTTP_CODE_OK) {
      String payload = http.getString();
            Serial.println("payload : ");
      Serial.println(payload);
          Serial.println("data after conv : ");

      jsonToIno(payload);
    } else {

      Serial.printf("[HTTP] POST... code: %d\n", httpCode);
    }
  } else {
    Serial.printf("[HTTP] POST... failed, error: %s\n",
http.errorToString(httpCode).c_str());
  }
  http.end();

}
```

Fig 5.15 UploadPic

- To inform the robot when it reaches the end of its designated path

```
Untitled
long Ultrasonic_read(){
  digitalWrite(trigger, LOW);
  delayMicroseconds(2);
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);
  long time = pulseIn (echo, HIGH);
  return time / 29 / 2;
}
```

Fig 5.16 Ultrasonic_read

5.1.3 Artificial Intelligence (AI):

❖ **Code:**

- Training a CNN

```
model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

...
Total params: 183,942
Trainable params: 183,942
Non-trainable params: 0
```

Fig 5.17 Training a CNN1

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
)
```

Fig 5.18 Training a CNN2

```
Epoch 2/50
100/100 [==============================] - 62s 618ms/step - loss: 0.4820 - accuracy: 0.8034 - val_loss: 0.4667 - val_accuracy: 0.8021
Epoch 3/50
...
Epoch 49/50
100/100 [==============================] - 61s 610ms/step - loss: 0.0394 - accuracy: 0.9881 - val_loss: 0.0462 - val_accuracy: 0.9818
Epoch 50/50
100/100 [==============================] - 61s 612ms/step - loss: 0.0173 - accuracy: 0.9944 - val_loss: 0.1483 - val_accuracy: 0.9557
```

Fig 5.19 Training a CNN3

- CNN Testing

```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(test_ds)
print(f"Test Accuracy: {round(scores[1],4)*100}%")
```

```
[INFO] Calculating model accuracy
14/14 [==============================] - 19s 115ms/step - loss: 0.1578 - accuracy: 0.9587
Test Accuracy: 95.87%
```

Fig 5.20 CNN Testing1

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

Fig 5.21 CNN Testing2

- YOLO for Leaf Detection and Cropping

yolo_object_detection_custom_train

```
!pip install ultralytics
from ultralytics import YOLO
import os
from IPython.display import display, Image
from IPython import display
display.clear_output()
!yolo mode=checks

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="***************")
project = rf.workspace("project-*****").project("final-yolo-test")
version = project.version(1)
dataset = version.download("yolov8")

!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs= 80 imgsz=256 plots=True
```

Fig 5.22 YOLO8 for Leaf Detection and Cropping

5.1.4 Back End:

❖ Connection file

In this file we should give the basic connectivity for other PHP files we might use along the project as we cannot write these codes over and over again it's not efficient and takes a lot of time, let's begin with it.

1. **Setting up the connection variables:**

- $dsn (Database source name) variable stores what database we will interact with.

```
1  $dsn = "mysql:host=localhost;dbname=plantsdatabase"; //DataBase source name
```

Fig 5.23 $dsn (Database source name)

- Every database should have the username and the password of it for integrity and security matters, therefore there is $user & $pass variables meant to carry the username and password for our database.

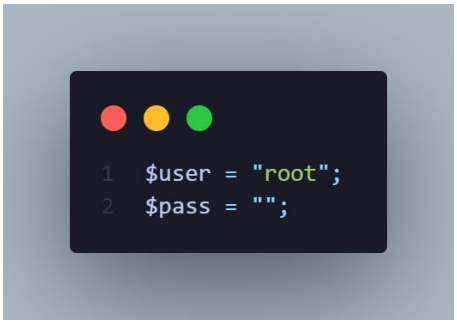


Fig 5.24 Username&password

- If we ever need to store some Arabic strings, we set this option in the database.

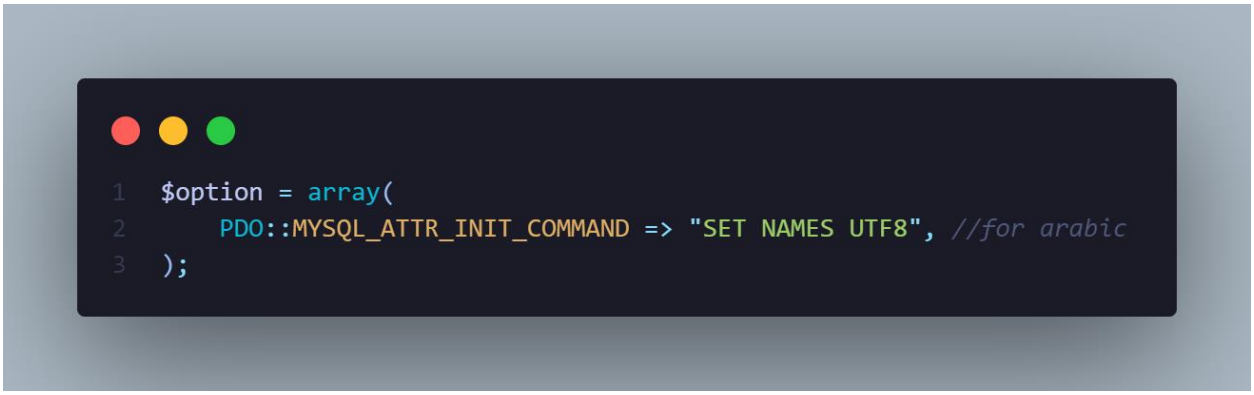


Fig 5.25 Store Arabic Strings

2. Instance creation:

- Next step we will try to create an instance for the class PDO in PHP.

Fig 5.26 create an instance for the class PDO

- Catching any error might occur in this process.

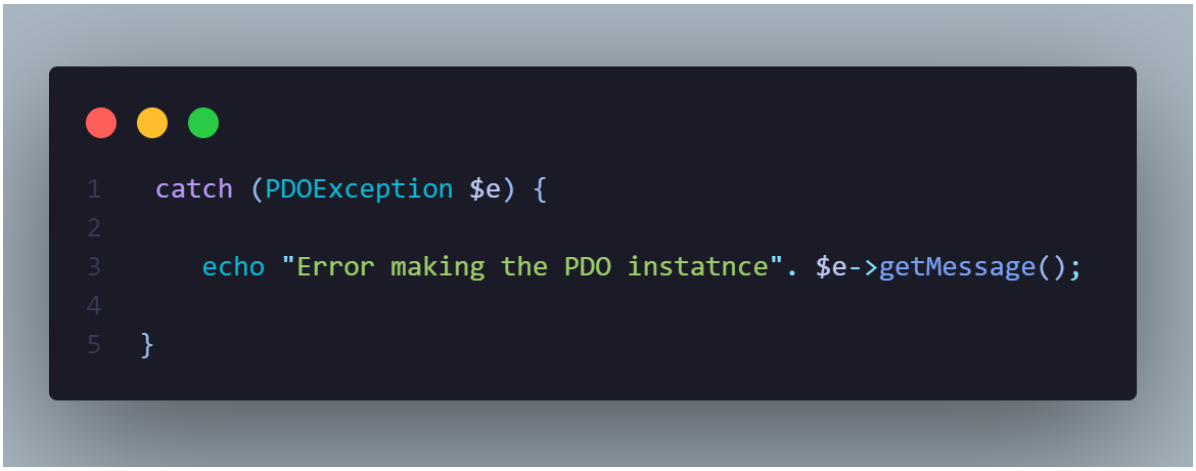


Fig 5.27 Catching errors

- PDO (PHP Data Objects) is a lightweight, consistent framework for accessing databases in PHP. Database-specific features may be exposed as standard extension functions by any database driver that implements the PDO interface. Note that the PDO in PHP extension by itself cannot perform any database functions; to connect to a database server, you must use a database specific PDO driver.

- In PHP waiting for a post request is quite easy by using the $_POST predefined array simply it is listening for any receivable request with the same key.

- Waiting for the location of the plant and the firebase link which carries the plant captured image. (As shown in Fig 5.28)

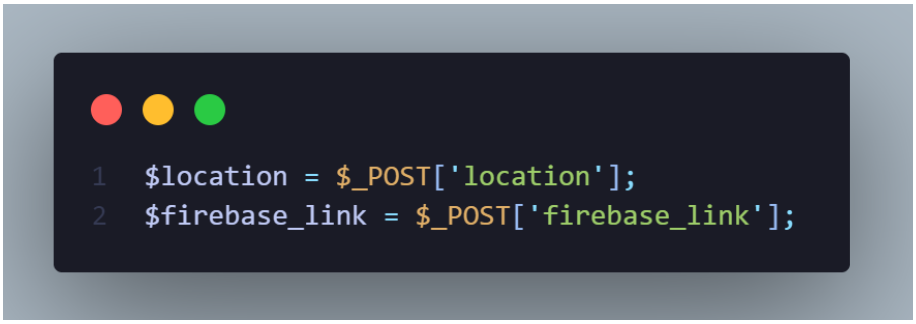


Fig 5.28 location&firbaseLink

- Using the prepare function with the PDO object in order to insert the link and the location of the intended plant in our database as we will use them later on.

Fig 5.29 Insert location&firbaseLink

- Selecting the timestamp from our database specifically the last timestamp for the latest image we had added.

Fig 5.30 Select the latest timestamp

- Using the curl connection.

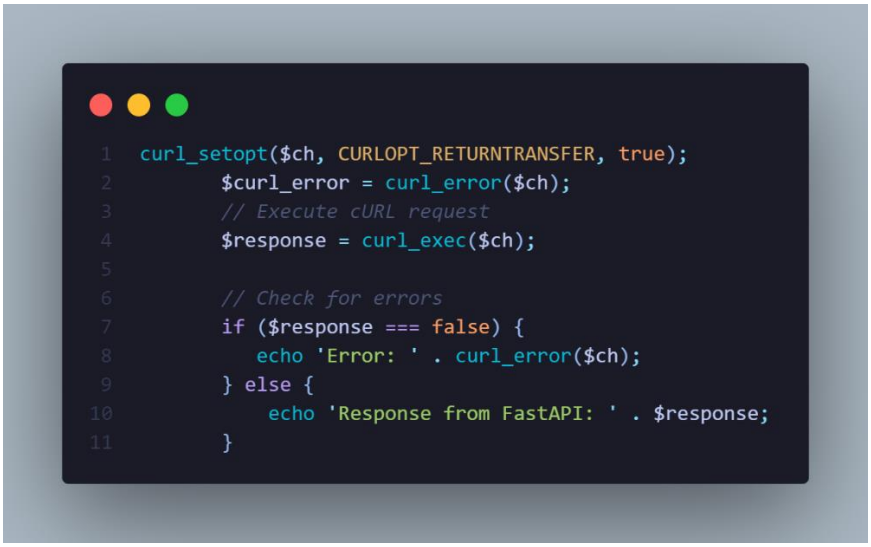


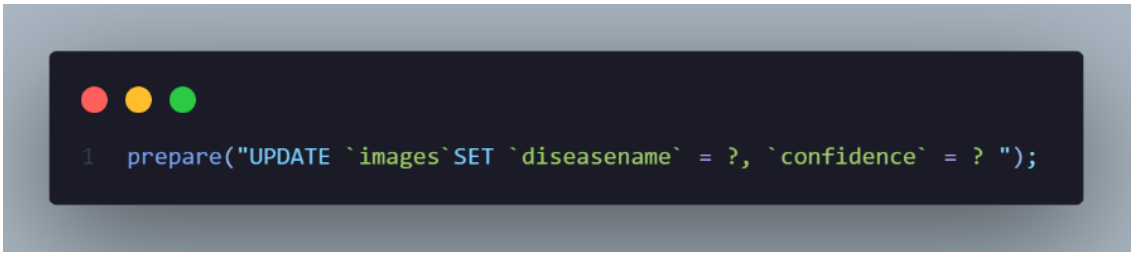
Fig 5.31 curl connection

- Listing to the response form the AI which carries two things the result of the detection of the diseases and how confident the AI model is with this result.

```
        // ...
if (isset($responseDecoded['class']) && isset($responseDecoded['confidence'])) {
    $class = strval($responseDecoded['class']);
    $confidence = strval($responseDecoded['confidence']);

  //  echo "Class: " . $class;
}
```

Fig 5.32 AI Response

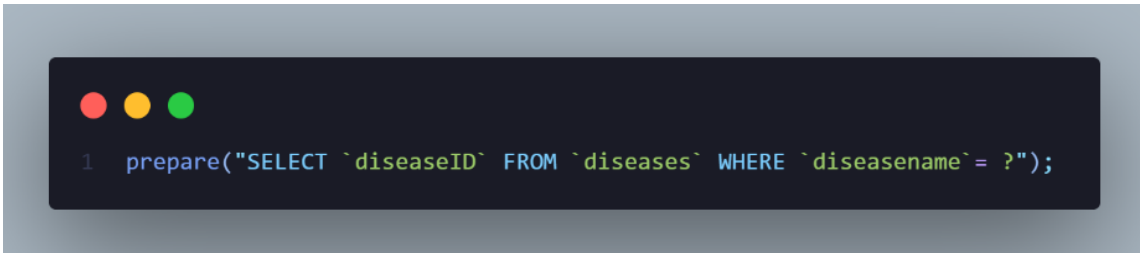
- Next step is to store these results with every image we have captured.



```
prepare("UPDATE `images` SET `diseasename` = ?, `confidence` = ? ");
```

Fig 5.33 Store Results with images

- After we insert the disease (if exists) we are selecting the disease information from the phpMyAdmin database to send it to the mobile app.



```
prepare("SELECT `diseaseID` FROM `diseases` WHERE `diseasename`= ?");
```

Fig 5.34 select disease information

- As we have finished all the query for entering all data needed for the database we must send the plant disease to the embedded system.

Fig 5.35 send plant disease

5.2 Testing

5.2.1 Testing Implementation

Is the process of putting a strategy into action. Before it can be implemented, the strategy must be finished, and the target must be determined. Any activity mentioned in the plan is evaluated for success.

Test Case	Input	Expected Outcome	Obtained Outcome	Pass/Fail	Test On
Registration	Register Name, E-mail, Password, Phone	Register Successful	Register Successful	Pass	20/4/2024
Login	E-mail, Password	Login Successful	Login Successful	Pass	20/4/2024

Data Request	Request	Request sent successfully	Request sent successfully	Pass	1/5/2024
Taking Photo	Open Camera	Photo taken successfully	Photo taken successfully	Pass	10/5/2024
Manage	Create, Read, Update, Delete	All operations work	All operations work	Pass	15/5/2024

Table 5-2. Testing Duration

5.2.2 Testing Report

This test report is required in order to acquire formal test results, allowing for a quick and detailed review of the results. It's a report that uses data from the patch test office to explain the environmental and operating circumstances, as well as how the test results compare to the test objectives. The test report is quite important, and you must comprehend it. The test report is vital because you need toknow whether or not the equipment is ready. It's a text that documents the results of a research study:

- Registration Test
- Reversion Tests
- Safety Tests
- Practical Tests
- Presentation Testing
- Usability Tests
- Scalability Testing

Here are some benefits of testing.

- Make the UI better and simpler for interaction.
- Improve The functionality of the application.
- Enhance the system's performance
- Make the system very easy to use.

6. Chapter Six

6.1 Conclusion

The Agriculture Assistant at the forefront of agricultural innovation, harnessing artificial intelligence, specifically deep learning, to monitor plants and detect diseases inside greenhouses. By training on extensive datasets, our deep learning model achieves unprecedented accuracy in disease identification, distinguishing between healthy and diseased plants with exceptional precision. This integration of deep learning technology not only enhances efficiency and accuracy but also facilitates early intervention, minimizing the risk of crop loss and maximizing yields.

Our project is designed as an integrated system to cater to diverse stakeholders, from large corporations to traditional farmers, aimed at addressing challenges such as poor crop quality and plant diseases comprehensively, in order to preserve the agricultural system as a supporter for food security, and

To achieve our objectives, we have implemented a multi-faceted approach:

We developed a self-moving model equipped with cameras and sensors for monitoring agricultural land and plants.

Created an artificial intelligence model to analyze plant images captured by the robot, identifying plant types and diseases.

Design and deployment of a mobile application for remote management, featuring a dedicated section for processing images and disease information, and suggesting appropriate treatments.

Automated generation of comprehensive reports upon disease detection, detailing the specific name and precise location of the disease within the greenhouse. These reports are seamlessly communicated to users through a user-friendly mobile application interface, providing real-time updates on plant health status.

6.2 Future Work

There are several potential future improvements that could be considered for our project:

1. Enhanced Disease Detection Algorithms: Continuously refine and optimize the deep learning algorithms used for plant disease detection. This could involve collecting more diverse and extensive datasets, it will allow us to add more plant species and their associated diseases.

2. Expansion to Outdoor Environments: Extend the capabilities of the system to outdoor agricultural environments, allowing farmers to monitor and manage crops in fields rather than just in greenhouses. This may require additional considerations for weatherproofing, robustness to environmental conditions, and connectivity options for remote locations.

3. User Feedback Integration: Collecting feedback from users of the mobile app and the robot can provide valuable insights for improvement. Incorporating user feedback into future updates and iterations will help address any usability issues, allow users to provide feedback on disease detection accuracy, report any issues or false positives encountered, improve user experience, and improve system features and functionality.

4. Integration with Agricultural IoT Devices: Explore integration with other agricultural IoT devices and sensors to provide a more comprehensive monitoring and management solution. This could include soil moisture sensors, weather stations, and environmental sensors to gather additional data and enhance decision-making capabilities.

5. Collaboration with agricultural experts: Collaboration with agricultural experts, botanists, and plant pathologists can provide valuable expertise and guidance in the

field. Partnering with professionals in the field can help verify the accuracy of the system, ensure it meets industry standards, and facilitate knowledge sharing to continually improve the project.

6. By increasing the budget, we can let the robot learn its path without preparing the path beforehand.

7. Transform the car into a drone and completely modify the shape to fit where you are using it.

By incorporating these future improvements, our project can continue to evolve and address the evolving needs of farmers and agricultural practitioners, ultimately contributing to more sustainable and productive farming practices.

6.3 References

[1] M. Jhuria, A. Kumar and R. Borse, "Image processing for smart farming: Detection of disease and fruit grading," in Image Information Processing (ICIIP), 2013 IEEE Second International Conference, 2013.

[2] S. R. Dubey and A. S. Jalal, "Detection and classification of apple fruit diseases using complete local binary patterns," in Computer and Communication Technology (ICCCT), 2012 Third International Conference, 2012.

[3] D. Pimentel, R. Zuniga and D. Morrison, "Update on the environmental and economic costs associated with alien-invasive species in the United States," Ecological Economics, vol. 52, (3), pp. 273-288, 2005.

[4] S. Sankaran, "A review of advanced techniques for detecting plant diseases," Computer Electronics in Agriculture, vol. 72, (1), pp. 1-13, 2010.

[5] M. M. López, "Innovative tools for detection of plant pathogenic viruses and bacteria," International Microbiology, vol. 6, (4), pp. 233-243, 2003.

[6] Q. Li, M. Wang and W. Gu, "Computer vision-based system for apple surface defect detection," Computer Electronics in Agriculture, vol. 36, (2), pp. 215-223, 2002.

[7] S. Sannakki, "Leaf Disease Grading by Machine Vision and Fuzzy Logic," International Journal of Computer Technology and Applications, vol. 2, (5), pp. 1709-1716, 2011.

[8] R. Oberti, "Selective spraying of grapevines for disease control using a modular agricultural robot," Biosystems Engineering, vol. 146, pp. 203-215, 2016.

[9] DC Motor.Retrieved from Zddriver Website, https://www.zddriver.com/news/industry-news/how-dc-motors-are-used-in-robotics.html , Accessed on Jan 29, 2024.

[10] L298N DC Motor driver.Retrieved from Vayuyaan Website, https://vayuyaan.com/blog/everything-you-want-to-know-about-l298n/#google_vignettel , Accessed on 15 Apr 2024.

[11] Li-Ion Battery 18650.Retrieved from Fogstar Website, https://www.fogstar.co.uk/blogs/fogstar-blog/what-are-18650-batteries , Accessed on March 13, 2024.

[12] ESP32.Retrieved from Wikipedia Website, https://en.wikipedia.org/wiki/ESP32 , Accessed on 29 April 2024.

[13] Infrared sensor (IR).Retrieved from Robocraze Website, https://robocraze.com/blogs/post/ir-sensor-working , Accessed on Jun 20, 2024.

[14] Micro Servo Motor.Retrieved from Robocraze Website, https://robocraze.com/blogs/post/what-is-a-servo-motor-guide-to-servo-motor-working?_pos=15&_sid=2a259aa1b&_ss=r , Accessed on Jun17, 2024.

[15] ESP32-CAM.Retrieved from Robocraze Website, https://robocraze.com/blogs/post/all-about-esp32-camera-module , Accessed on Jan 15, 2024.

[16] ESP32-CAM MB.Retrieved from Espboards Website, https://www.espboards.dev/blog/flash-any-esp32-with-esp32-cam-mb/#google_vignette , Accessed on Jan 20, 2024.

[17] Arduino IDE.Retrieved from Arduino Website,https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics , Accessed on Jun17, 2024.

[18] ESP32-CAM.Retrieved from RandomNerdTutorials Website, https://randomnerdtutorials.com/projects-esp32-cam/ , Accessed on Oct 20, 2024.

[19] ESP32.Retrieved from RandomNerdTutorials Website, https://randomnerdtutorials.com/projects-esp32/, Accessed on Oct 20, 2024.

[20] https://docs.flutter.dev/

[21] https://pub.dev/packages/get

[22] https://docs.flutter.dev/cookbook/networking/fetch-data

[23] https://pub.dev/packages/get_storage

[24] https://www.php.net/manual/en/reserved.variables.request.php

[25] https://www.php.net/manual/en/reserved.variables.post.php

[26] https://www.php.net/manual/en/reserved.variables.argv.php

[27] https://www.php.net/manual/en/index.php

[28] Python Software Foundation. (n.d.). Python documentation. Retrieved from https://docs.python.org/

[29] Van Rossum, G., & Drake, F. L. (2009). Python 3 reference manual. Scotts Valley, CA: CreateSpace.

[30] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.

[31] Chollet, F. (2018). Deep learning with Python. Manning Publications.

[32] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

[33] Hughes, D. P., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv preprint arXiv:1511.08060.

[34] Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, 1419. doi:10.3389/fpls.2016.01419

[35] Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[36] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[37] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[38] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).