****

**COMSATS University Islamabad**

**Abbottabad, Pakistan**

**Plant Disease Diagnose System**

***By***

**Furqan Ahmad CIIT/FA21-BSE-089/ATD**

**Asad Khan CIIT/FA21-BSE-081/ATD**

**Abdul Muhaimin CIIT/FA21-BSE-061/ATD**

***Supervisor***

**Javed Raza**

***Bachelor of Science in Computer Science (2021-2025)***

**The candidate confirms that the work submitted is their own and appropriate  
 credit has been given where reference has been made to the work of others**.

****

**COMSATS University, Islamabad Pakistan**

**Plant Disease Diagnose System**

**A project presented to**

**COMSATS Institute of Information Technology, Islamabad**

**In partial fulfillment**

**of the requirement for the degree of**

***Bachelor of Science in Computer Science (2021-2025)***

**By**

**Furqan Ahmad CIIT/FA21-BSE-089/ATD**

**Asad Khan CIIT/FA21-BSE-081/ATD**

**Abdul Muhaimin CIIT/FA21-BSE-061/ATD**

**DECLARATION**

We hereby declare that this software, neither whole nor as a part has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other. We will stand by the consequences. No Portion of the work presented has been submitted of any application for any other degree or qualification of this or any other university or institute of learning.

Student Name 1 Student Name 2

Furqan Ahmad Abdul Muhaimin

--------------------------- ---------------------------

Student Name 3

Asad Khan

--------------------------

**CERTIFICATE OF APPROVAL**

It is to certify that the final year project of BS (CS) “Plant Disease Diagnose System” was developed by   
**Asad Khan (FA21-BSE-081)** and Abdul Muhaimin **(FA21-BSE-061)** and **Furqan Ahmad (FA21-BSE-089)** under the supervision of “Javed Raza” and co supervisor “CO-SUPERVISOR NAME” and that in (their/his/her) opinion; it is fully adequate, in scope and quality for the degree of Bachelors of Science in Computer Sciences.

---------------------------------------

**Supervisor**

---------------------------------------

**External Examiner**

---------------------------------------

**Head of Department**

**(Department of Computer Science)**

**EXECUTIVE SUMMARY**

Plant diseases are a leading cause of reduced agricultural productivity and poor plant quality, especially in regions where farmers lack access to expert consultations. The Plant Disease Diagnose System is a comprehensive solution that allows users—primarily farmers—to upload images of affected plants. It leverages deep learning for disease detection and integrates real-time weather data to enhance diagnostic accuracy. The system offers treatment guidance, preventive measures, a history-tracking feature, and supports multiple languages(English , Urdu). It aims to minimize plant loss and empower farmers through accessible, tech-driven agricultural support.

**ACKNOWLEDGEMENT**

All praise is to Almighty Allah who bestowed upon us a minute portion of His boundless knowledge by virtue of which we were able to accomplish this challenging task.

We are greatly indebted to our project supervisor “Dr. Majid Iqbal Khan” and our Co-Supervisor “Mr. Mukhtar Azeem”. Without their personal supervision, advice and valuable guidance, completion of this project would have been doubtful. We are deeply indebted to them for their encouragement and continual help during this work.

And we are also thankful to our parents and family who have been a constant source of encouragement for us and brought us the values of honesty & hard work.

Student Name 1 Student Name 2

Furqan Ahmad Abdul Muhaimin

--------------------------- ---------------------------

Student Name 3

Asad Khan

---------------------------

**ABBREVIATIONS**

|  |  |
| --- | --- |
| **SRS** | Software Require Specification |
| **PC** | Personal Computer |
| **SDD** | Software Design Description |
| **ML** | Deep learning |
| **UI** | User Interface |
| **API** | Application Programming Interface |

**TABLE OF CONTENTS**

[**1 Introduction** 11](#_Toc520773734)

[1.1 Brief 11](#_Toc520773735)

[1.2 Relevance to Course Modules 11](#_Toc520773736)

[1.3 Project Background 11](#_Toc520773737)

[1.4 Literature Review 11](#_Toc520773738)

[1.5 Analysis from Literature Review (in the context of your project) 11](#_Toc520773739)

[1.6 Methodology and Software Lifecycle for This Project 12](#_Toc520773740)

[1.6.1 Rationale behind Selected Methodology 12](#_Toc520773741)

[**2** **Problem Definition** 12](#_Toc520773742)

[2.1 Problem Statement 12](#_Toc520773745)

[2.2 Deliverables and Development Requirements 12](#_Toc520773746)

[2.3 Current System (if applicable to your project) 13](#_Toc520773747)

[**3** **Requirement Analysis** 14](#_Toc520773748)

[3.1 Use Cases Diagram(s) 14](#_Toc520773750)

[3.2 Detailed Use Case 14](#_Toc520773751)

[3.3 Functional Requirements 18](#_Toc520773752)

[3.4 Non-Functional Requirements 18](#_Toc520773753)

[**4 Design and Architecture** 20](#_Toc520773754)

[4.1 System Architecture 20](#_Toc520773759)

[4.2 Data Representation [Diagram + Description] 21](#_Toc520773760)

[4.3 Process Flow/Representation 22](#_Toc520773761)

[4.4 Design Models [along with descriptions] 23](#_Toc520773762)

[**5** **Implementation** 23](#_Toc520773763)

[5.1 Algorithm 25](#_Toc520773765)

[5.2 External APIs 25](#_Toc520773766)

[5.3 User Interface 25](#_Toc520773767)

[**6 Testing and Evaluation** 26](#_Toc520773768)

[6.1 Manual Testing 26](#_Toc520773771)

[6.1.1 System testing 26](#_Toc520773772)

[6.1.2 Unit Testing 26](#_Toc520773773)

[6.1.3 Functional Testing 27](#_Toc520773774)

[6.1.4 Integration Testing 27](#_Toc520773775)

[6.2 Automated Testing: 28](#_Toc520773776)

[Tools used: 28](#_Toc520773777)

[**7** **Conclusion and Future Work** 28](#_Toc520773778)

[7.1 Conclusion 28](#_Toc520773780)

[7.2 Future Work 28](#_Toc520773781)

[**8** **References** 29](#_Toc520773782)

**LIST OF FIGURES**

Fig 2.1 Plantix 13

Fig 2.2 Plantix 13

Fig 3.1 Use Case Diagram 14

Fig 4.1 System Arcitecture 21

Fig 4.2 Data Representation 22

Fig 4.3 Activity Diagram 23

Fig 4.4 Sequence Diagram 24

Fig 4.5 Sequence Diagram 24

Fig 4.6 Sequence Diagram 25

Fig 4.7 Sequence Diagram 25

Fig 4.8 Sequence Diagram 26

**LIST OF TABLES**

Table 1.1 Table1 8

Table 2.1 Table 2 9

1. **Introduction**
   1. **Brief**

The Plant Disease Diagnose System is a smart, deep learning-powered application designed to help farmers identify and manage plant diseases efficiently. It allows users to upload plant images, which are processed using a trained ML model to diagnose diseases and recommend treatments. The system also integrates weather data (such as temperature and humidity) to enhance prediction accuracy. A key feature of the system is its offline capability, allowing users in low-connectivity areas to use core functionalities without needing continuous internet access. This makes the system more practical and inclusive for rural farmers, helping reduce plant loss through timely and accurate interventions

* 1. **Relevance to Course Modules**

This project relates to core BCS courses, including Artificial Intelligence, Software Engineering, Database Management Systems, Image Processing, and Web & Mobile Application Development.

* 1. **Project Background**

Farmers, particularly in under-resourced regions, struggle to detect diseases in time, leading to severe agricultural losses. Inspired by these real-world challenges, this system addresses the gap by delivering intelligent disease detection tools directly into the hands of users.

* 1. **Literature Review**

Studies show that applications like Plantix and AgroAI, while useful, are limited in functionality or accessibility. Most do not offer multilingual support (English and Urdu), and do not recommend result approximately 100% or near to it or the trained model do not have any checks on picture that are being uploaded,it doesn’t care either it is the correct picture related to the application or not but still the model recommend suggestions.

* 1. **Analysis from Literature Review (in the context of your project)**

The Plant Disease Diagnose System improves upon existing solutions by offering a broader disease database and validating uploaded plant images for quality and accuracy. With these checks in place, the system aims to deliver highly accurate or near-accurate disease predictions. It also integrates regional weather data to enhance the contextual relevance of its diagnoses. Additionally, the system provides role-based access limited to two main user types: farmers and administrators, ensuring secure and streamlined interactions with the platform.

* 1. **Methodology and Software Lifecycle for This Project**

We adopted the Agile development model for its adaptability and feedback-centric nature. Object-Oriented Design principles were followed to ensure modular, maintainable code structure.

1. * 1. **Rationale behind Selected Methodology**

Agile enables iterative development.

OOD supports modular, scalable design.

* + 1. **Rationale behind Selected Methodology**

It is the example of third and last level heading. Please do not insert further levels in numbers. Use different format style e.g. italic to highlight the important text.

1. **Problem Definition**

The problematic statement and its solutions are given in this chapter. Also, deliverable and development requirements are explained below in this chapter.

3. 1. **Problem Statement**

Agricultural productivity is heavily impacted by plant diseases, especially in rural areas where farmers lack access to expert agronomists or diagnostic tools. Current solutions either require internet access or lack detailed recommendations tailored to specific environmental conditions. This often leads to delayed or improper treatment of plant, resulting in loss of yield and income.

* 1. **Deliverables and Development Requirements**

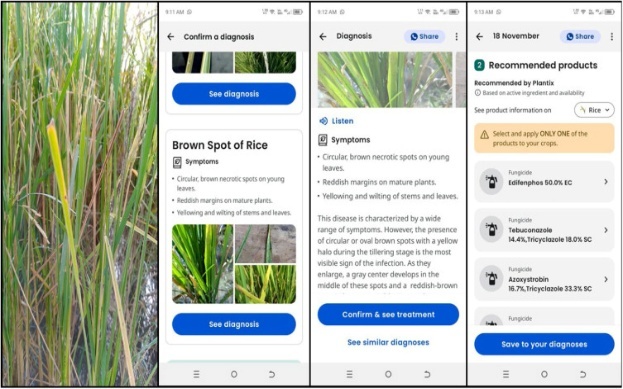
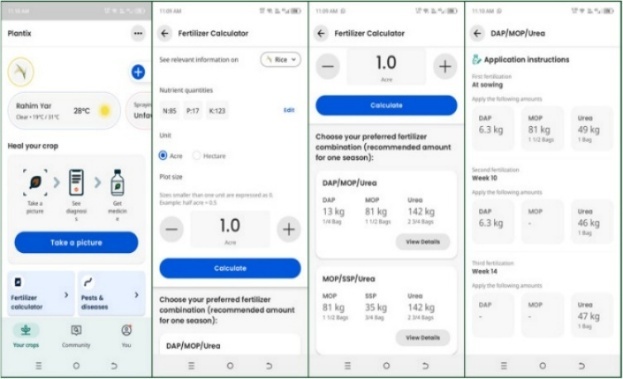
**Deliverables**

* Mobile and web-based application for users to upload plant images
* Deep learning model trained to identify plant diseases
* Admin dashboard for disease management and user control
* Role-based access control (Farmer, Expert, Admin)
* Disease history and plant tracking module
* Integration with weather data API for better predictions
* Notifications and alerts module

**Development Requirements**

* MS Visual Studio Code: The IDE for writing and debugging both frontend and backend code.
* Agile Methodology: Ensures flexibility and iterative development with continuous feedback.
* User Testing: Conduct sessions with farmers or test groups to gather feedback on usability and functionality.
* Functionality Testing: To validate the correctness and reliability of all app features before release .
  1. **Current System (if applicable to your project)**

Plantix is a widely used mobile application that helps farmers detect plant diseases by analyzing uploaded images. While it has made significant contributions to agriculture, particularly in plant health monitoring, it has certain limitations. The disease database in Plantix is relatively narrow, covering only a limited number of commonly grown plants. This can hinder its usefulness for farmers growing regional or less-commercial plants. Furthermore, the user interface, though functional, is not fully optimized for users with low digital literacy, making navigation challenging for some. Another limitation is its dependence on a stable internet connection to upload images and retrieve prediction results, which may not be ideal for farmers in remote or rural areas. Plantix also does not take environmental data such as temperature, humidity, or soil conditions into account when diagnosing diseases, which may reduce the accuracy of its predictions in varying conditions. These limitations highlight the need for a more localized, offline-capable, user-friendly, and environmentally responsive solution like the Plant Disease Diagnose System.

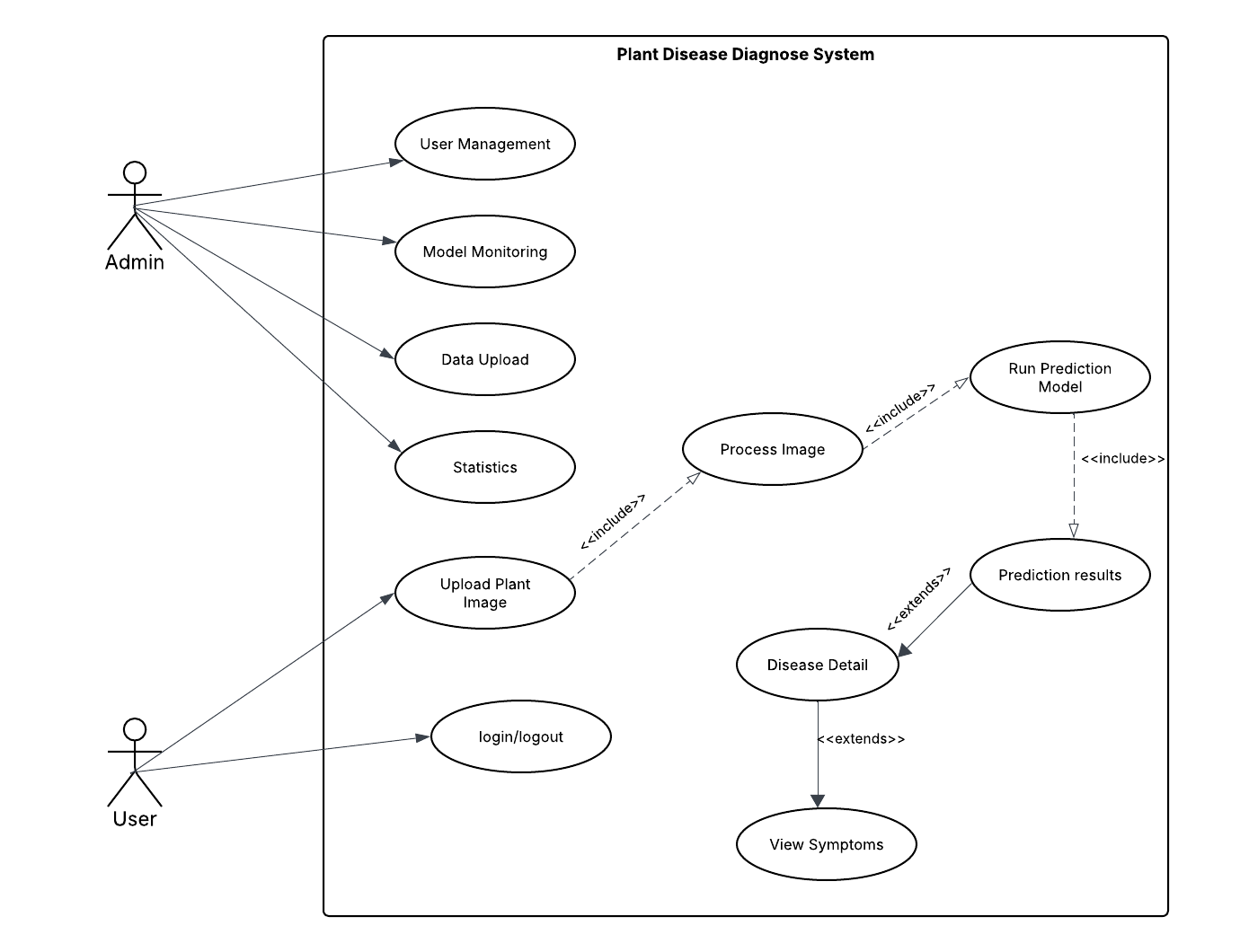
**

*Figure 2.1: Plantix Figure 2.2: Plantix*

1. **Requirement Analysis**

The following parts of Software Requirements Specification (SRS) report should be included in this chapter.

2. 1. **Use Cases Diagram(s)**



*Figure 3.1: Use Cases Diagram*

* 1. **Detailed Use Case**

*Table 3.2.1 Use Case UC-1: User Login / Signup*

|  |  |
| --- | --- |
| Use Case ID | UC-1 |
| Use Case Name | User Login / Signup |
| Actors | Admin, Farmer |
| Description | Enables users to register or log in to access system features. |
| Trigger | User selects "Login" or "Sign up" option. |
| Preconditions | User must have a valid internet connection. |
| Postconditions | User is authenticated and redirected to the dashboard. |
| Normal Flow | 1. User enters login credentials (email and password). 2. System validate credential. 3. User is logged in successfully or redirected to the signup form for registration. |
| Alternative Flow | If the credentials are invalid, the system prompts the user to retry or reset their password. |
| Exceptions | Database connection failure or invalid credentials |

*Table 3.2.1 Use Case UC-2: User Management*

|  |  |
| --- | --- |
| Use Case ID | UC-2 |
| Use Case Name | User Management |
| Actors | Admin |
| Description | Enables the admin to manage user accounts, including adding, editing, and deleting users. |
| Trigger | Admin selects the "User Management" option. |
| Preconditions | Admin must have logged into the system. |
| Postconditions | User accounts are updated based on the admin's actions. |
| Normal Flow | 1. Admin selects a user account.  2. Admin performs the desired action (add, edit, delete).  3. System updates the user account. |
| Alternative Flow | If user details are invalid, the system displays an error message. |
| Exceptions | Database connection |

*Table 3.2.1 Use Case UC-3: Model Monitoring*

|  |  |
| --- | --- |
| Use Case ID | UC-3 |
| Use Case Name | Model Monitoring |
| Actors | Admin |
| Description | Allows the admin to monitor the performance of the disease prediction model. |
| Trigger | Admin selects the "Model Monitoring" option. |
| Preconditions | The prediction model must be operational. |
| Postconditions | Admin views performance metrics such as accuracy, precision, and efficiency. |
| Normal Flow | 1. Admin accesses the monitoring dashboard.  2. System retrieves and displays model metrics. |
| Exceptions | Model monitoring metrics unavailable due to |

*Table 3.2.1 Use Case UC-4: Data Upload*

|  |  |
| --- | --- |
| Use Case ID | UC-4 |
| Use Case Name | Data Upload |
| Actors | Admin |
| Description | Provides functionality for the admin to upload new datasets for model training or updates. |
| Trigger | Admin selects the "Data Upload" option. |
| Preconditions | Admin must be logged into the system. |
| Postconditions | The dataset is successfully uploaded and ready for processing. |
| Normal Flow | 1. Admin uploads a dataset file.  2. System validates the file format and content.  3. System stores the dataset. |
| Alternative Flow | If the file format is invalid, the system displays an error message. |
| Exceptions | File upload fails due to connectivity or storage issues |

*Table 3.2.1 Use Case UC-5: Statistics*

|  |  |
| --- | --- |
| Use Case ID | UC-5 |
| Use Case Name | Statistics |
| Actors | Admin |
| Description | Allows the admin to generate detailed reports of prediction results. |
| Preconditions | Admin must have performed a prediction task. |
| Postconditions | Report is generated and made available for download or viewing. |
| Normal Flow | 1. User selects report type.  2. System compiles report data.  3. System generates the report. |
| Alternative Flow | If no data is available for the report, the system displays a message. |
| Exceptions | Report generation fails due to data retrieval errors |

*Table 3.2.1 Use Case UC-6: Upload Plant Image*

|  |  |
| --- | --- |
| Use Case ID | UC-6 |
| Use Case Name | Upload Plant Image |
| Actors | User |
| Description | Allows users to upload an image of a plant for diagnosis. |
| Trigger | User clicks the "Upload Image" button. |
| Preconditions | 1. User is logged in. 2. Image file is available on the user's device. |
| Postconditions | Image is processed and ready for analysis. |
| Normal Flow | 1. User selects an image file. 2. System checks for valid image format. 3. Image is uploaded successfully. |
| Alternative Flow | If the image format is invalid, the system prompts the user to upload a supported file type. |
| Exceptions | Image upload fails due to network issues |

*Table 3.2.1 Use Case UC-7: View Prediction Results*

|  |  |
| --- | --- |
| Use Case ID | UC-7 |
| Use Case Name | View Prediction Results |
| Actors | User |
| Description | Displays the health diagnosis and prediction results for the uploaded image. |
| Trigger | User completes image upload and waits for analysis results. |
| Preconditions | 1. User has uploaded a valid image. 2. System has successfully processed the image. |
| Postconditions | Results and recommendations are displayed on the dashboard. |
| Normal Flow | 1. System processes the image. 2. Prediction results are displayed, including potential disease, cause, and treatment recommendations. |
| Alternative Flow | If the model cannot make a prediction, the system provides troubleshooting suggestions (e.g., upload a clearer image). |
| Exceptions | Processing fails due to incomplete data or system error. |

*Table 3.2.1 Use Case UC-8: Manage User Profiles*

|  |  |
| --- | --- |
| Use Case ID | UC-8 |
| Use Case Name | Manage User Profiles |
| Actors | User |
| Description | Users can create, edit, or delete plant profiles for monitoring multiple plants. |
| Trigger | User accesses the plant profile section. |
| Preconditions | User must be logged in. |
| Postconditions | Changes to the plant profiles are saved successfully. |
| Normal Flow | User views the plant profiles dashboard. User selects an option to add, edit, or delete a profile. Changes are saved in the database. |
| Alternative Flow | If the user cancels, no changes are made. |
| Exceptions | Changes fail to save due to network issues |

* 1. **Functional Requirements**

**Feature 1: User Authentication**

* The system shall allow users to register by providing an email address, username, and password.
* The system shall validate user credentials during login.
* The system shall provide a password recovery option using a registered email.

**Feature 2: Plant Image Upload**

* The system shall allow users to upload an image file of a plant.
* The system shall validate the image file format (e.g., JPEG, PNG).
* The system shall notify the user if the upload fails due to an unsupported format or network issue.

**Feature 3: Disease Detection and Diagnosis**

* The system shall analyze the uploaded plant image using a trained deep learning model.
* The system shall display diagnosis results, including the detected disease, causes, and severity level.
* The system shall recommend treatments and preventive measures based on the diagnosis results.
  1. **Non-Functional Requirements**

**Usability:**

* The system should have an intuitive interface to accommodate non-technical users, such as farmers.

**Performance:**

* The system must process images and display predictions within 5 seconds.
* It should handle up to 1,000 concurrent users without any latency.

**Reliability:**

* The system must maintain 99% uptime and ensure uninterrupted access.

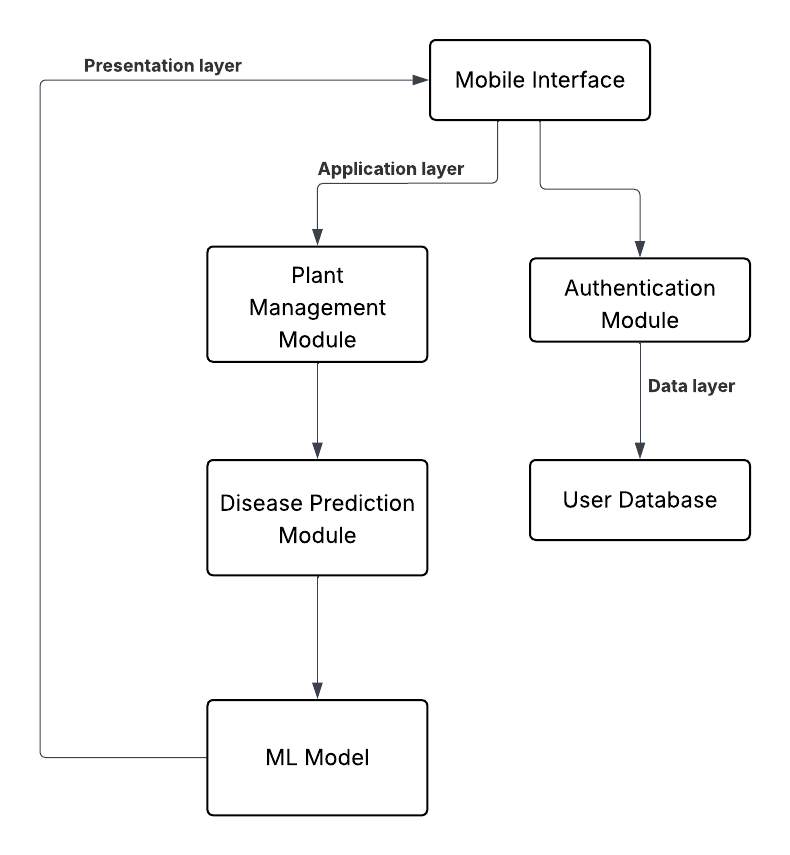
**Security:**

* All user data, including uploaded images, must be encrypted.
* Role-based access control should be implemented to restrict admin functionalities

1. **Design and Architecture**

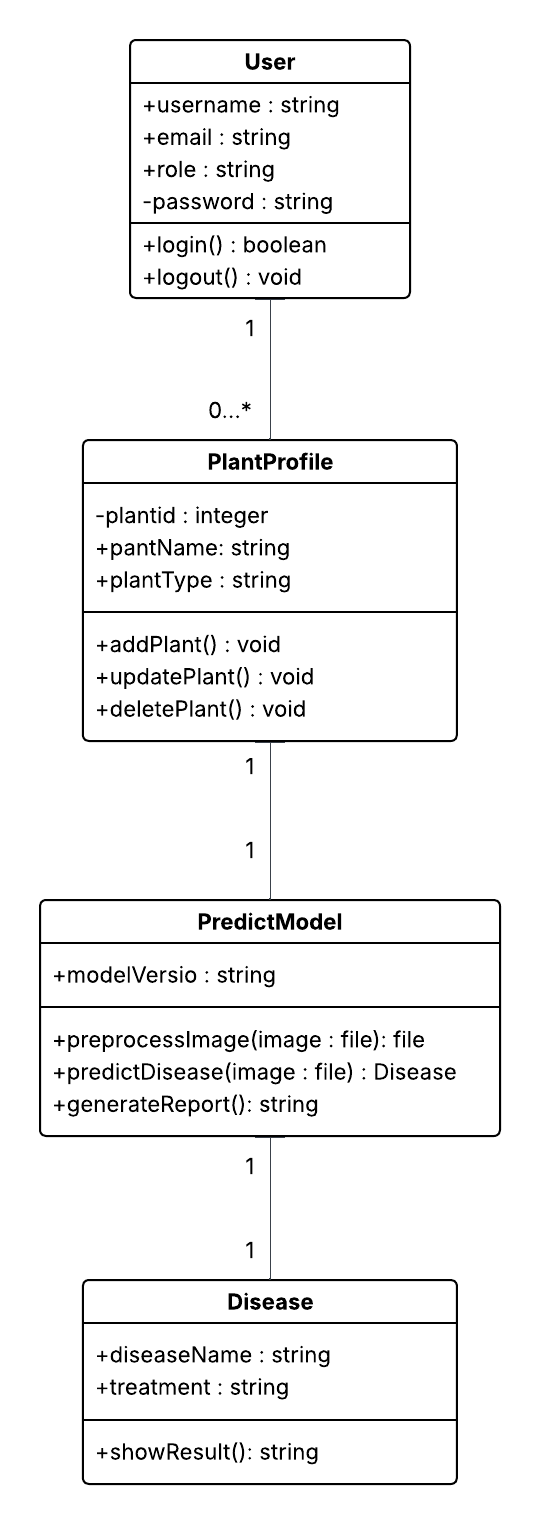
The following parts of Software Design Description (SDD) report should be included in this chapter.

5. 1. **System Architecture**

****

*Figure 4.1: Architecture Diagram*

* 1. **Data Representation [Diagram + Description]**

****

*Figure 4.2: Class diagram*

* 1. **Process Flow/Representation**

**Activity Diagram**

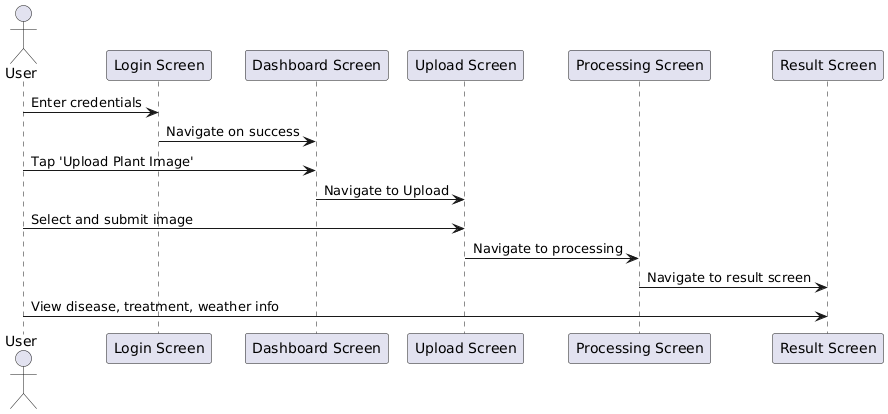
A screenshot of a diagram

AI-generated content may be incorrect.

*Figure 4.3: Activity Diagram*

* 1. **Design Models [along with descriptions]**

**Sequence Diagram Mobile Application**

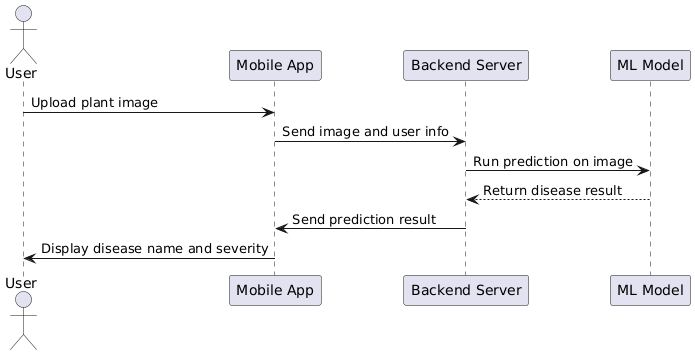
****

*Figure 4.4: Sequence Diagram*

**Description:**

The sequence starts when the user logs into the mobile app via the Login Screen. Upon successful login, they are directed to the Dashboard Screen, which serves as the central hub for the user. The user can choose to upload an image by tapping the 'Upload Image' button, leading them to the Plant Upload Screen. Once the user selects an image, it is sent for processing, and the app transitions to the Processing Screen. After the prediction process is complete, the user sees the result on the Prediction Result Screen. Lastly, the user can navigate to the Profile Screen to view their plant history and past diagnoses.

**Sequence Diagram Prediction Model**

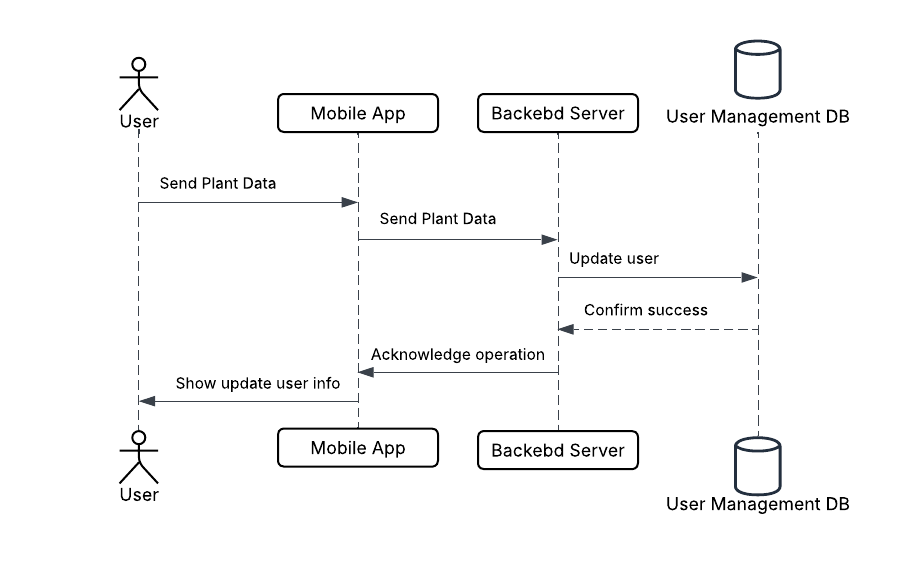
****

*Figure 4.5: Sequence Diagram*

**Description:**

The interaction begins when the mobile app sends an image, along with its metadata, to the backend server. The server then forwards the image to the preprocessing module, which cleans and resizes it before passing it to the deep learning (ML) model. The model processes the image and returns the predicted disease. This prediction is then sent back through the backend to the app, where the result is displayed on the user interface.

**Sequence Diagram User Management**

****

*Figure 4.8: Sequence Diagram*

**Description:**  
The user can manage their **account information** by registering, logging in, or updating their profile details. The app sends the user credentials or updated data to the **User Management Module**, which interacts with the **backend server and database** to create, authenticate, or update the user's records. Once the operation is successfully completed, the system sends a confirmation back to the app, ensuring the user is notified about the status of their request. This process ensures secure handling of user data and a smooth authentication experience.

1. **Implementation**
2. 1. **Algorithm**

The core module of the Plant Diagnose System is the **disease prediction algorithm** which uses a convolutional neural network (CNN). This algorithm analyses uploaded leaf images to detect the most probable disease.

### Pseudocode for Disease Prediction:

1. Input: Plant image
2. Preprocess image (resize, normalize, remove noise)
3. Load trained CNN model
4. Pass image through model
5. Get prediction probabilities for each disease
6. Return disease with highest probability and confidence score
7. Output: Predicted disease + confidence
   1. **User Interface**

The interface has been designed with HCI heuristics:

* **Simple login**/signup page with validation
* **Dashboard** for image upload, recent predictions, weather data
* **Disease** results screen with diagnosis and treatment
* **History** screen showing all user uploads and outcomes
* **Admin** panel with access to user logs, model accuracy, and dataset uploads

1. **Testing and Evaluation**

4. 1. **Manual Testing**
5. * 1. **System testing**

System testing was carried out after development to ensure the features work as intended. It includes unit testing, functional testing, and integration testing.

* + 1. **Unit Testing**
* **Unit Testing 1:** Login Form

**Testing Objective:** To ensure the login form is functioning correctly and handling invalid credentials.

*Table 5.1: Login Unit Testcase*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Verify user login with correct input | Email:[asadkhan@gmail.com](mailto:asadkhan@gmail.com)  Password: pass123 | Successfully logged in to the user dashboard | Pass |
| 2 | Login with incorrect password | Email:[muhaiman@gmail.com](mailto:muhaiman@gmail.com)  Password: pass677 | Show error: "Invalid credentials" | Pass |
| 3 | Submit login form with empty fields | Email: (empty)  Password: (empty) | Show error: "Email and password are required" | Pass |
| 4 | Login with unregistered email | Email: [zarar@gmail.com](mailto:zarar@gmail.com)  Password: 1234 | Show error: "Account not found" | Pass |
| 5 | Login with valid credentials but server down (simulated) | Email:[furqan@gmail.com](mailto:furqan@gmail.com)  Password: pass123 | Show error: "Unable to connect to server" | **Fail** |

* **Unit Testing 2:** Edit Profile

**Testing Objective:** To ensure the edit profile form updates data and handles errors.

*Table 5.2: Edit Profile Unit Testcase*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Update valid name and email | Name: Asad Khan  Email:[asad@gmail.com](mailto:asad@gmail.com) | Show success message and save to DB | Pass |
| 2 | Submit with invalid email format | Name: Furqan  Email: furqan@yahoo.com | Show validation error | Pass |
| 3 | Submit with empty name field | Name: (empty)  Email:[muhaiman@gmail.com](mailto:muhaiman@gmail.com) | Show error: "Name is required" | Pass |
| 4 | Update profile when database is disconnected (simulated) | Valid data | Show error: "Failed to update profile" | **Fail** |

* + 1. **Functional Testing**
* **Functional Testing 1:** Role-Based Login

**Objective**: Ensure that each user type is directed to their respective dashboard with appropriate navigation.

*Table 5.3: Login Functional Testcase*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Login as a Farmer | Email:[farmer@gmail.com](mailto:farmer@gmail.com)  Password: 1234 | Farmer dashboard loads with prediction tools | Pass |
| 2 | Login as an Admin | Email:[admin@gmail.com](mailto:admin@gmail.com)  Password: admin | Admin panel with user activity appears | Pass |

**Integration Testing**

Table 5.4 shows the integration testing

*Table 5.4: Integration Testcase*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Login and upload plant image | Image: valid.jpg | Image uploaded, passed to prediction model, result displayed | Pass |
| 2 | Predict disease and view result | Image: leaf.png | System shows predicted disease and suggested treatments | Pass |
| 3 | View prediction history | Logged-in user | History of uploaded images with results shown | Pass |
| 4 | Upload image when model offline | Image: valid.jpg (simulate model crash) | Show error: "Unable to process image at this time" | Fail |

* 1. **Automated Testing:**

This is the sample text

* + 1. **Tools used:**

Table 5.5 shows the

*Table 5.5: Tools used*

|  |  |  |  |
| --- | --- | --- | --- |
| Tool Name | Tool Description | Applied On | Results |
| Jest | JS test framework for component testing | Unit tests: login, form validation | Pass |
| Postman | API testing tool | Functional API: login, prediction route | Pass |
| React Testing Library | Simulate UI interactions | UI components: Button clicks, inputs | Pass |
| Firebase Emulator | Simulated backend DB & Auth | Integration testing for auth + database | Pass |
| Playwright | End-to-end browser testing tool | Simulated full user login and upload flow | Fail(image validation error simulated) |

1. **Conclusion and Future Work**
2. 1. **Conclusion**

The Plant Diagnose System helps farmers by predicting plant diseases through a web/mobile interface using machine learning. With real-time diagnosis, weather awareness, and treatment suggestions, it enhances agricultural productivity. The use of external APIs and cloud storage ensures scalability and global usability.

* 1. **Future Work**
* Offline mobile support using TensorFlow Lite
* IoT device integration for live soil/environment data
* Multi-language voice support for illiterate users
* Expand to livestock disease prediction
* Integration with agricultural expert helpline

1. **References**

References to any book, journal paper or website should properly be acknowledged. Please consistently follow the style. The following are few examples of different resources i.e. journal article, book, and website.

* 1. Lyda M.S. Lau, Jayne Curson, Richard Drew, Peter Dew and Christine Leigh, (1999), Use Of VSP Resource Rooms to Support Group Work in a Learning Environment, ACM 99, pp-2. **(Journal paper example)**
  2. Hideyuki Nakanishi, Chikara Yoshida, Toshikazu Nishmora and TuruIshada, (1996), FreeWalk: Supporting Casual Meetings in a Network, pp 308-314 **(paper on web)** http://www.acm.org/pubs/articles/proceedings/cscw/240080/p308-nakanishi.pdf
  3. Ali Behforooz& Frederick J.Hudson, (1996), Software Engineering Fundamentals, Oxford University Press. Chapter 8, pp255-235. **(book reference example)**
  4. Page Author, Page Title, http://www.bt.com/bttj/archive.htm, Last date accessed**. (web site)**