****

**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** 10](#_Toc200932825)

[**1.1.** **Brief** 10](#_Toc200932826)

[**1.2.** **Relevance to Course Modules** 10](#_Toc200932827)

[**1.3.** **Project Background** 10](#_Toc200932828)

[**1.4.** **Literature Review** 10](#_Toc200932829)

[**1.5.** **Analysis from Literature Review (in the context of your project)** 10](#_Toc200932830)

[**1.6.** **Methodology and Software Lifecycle for This Project** 11](#_Toc200932831)

[**1.6.1.** **Rationale behind Selected Methodology** 11](#_Toc200932839)

[**2.** **Problem Definition** 11](#_Toc200932840)

[**2.1.** **Problem Statement** 11](#_Toc200932845)

[**2.2.** **Deliverables and Development Requirements** 11](#_Toc200932846)

[**2.3.** **Current System (if applicable to your project)** 12](#_Toc200932847)

[**3.** **Requirement Analysis** 13](#_Toc200932848)

[**3.1.** **Use Cases Diagram(s)** 13](#_Toc200932853)

[**3.2.** **Detailed Use Case** 13](#_Toc200932854)

[**3.3.** **Functional Requirements** 17](#_Toc200932855)

[**3.4.** **Non-Functional Requirements** 18](#_Toc200932856)

[**4.** **Design and Architecture** 19](#_Toc200932857)

[**4.1.** **System Architecture** 19](#_Toc200932866)

[**4.2.** **Data Representation [Diagram + Description]** 20](#_Toc200932867)

[**4.3.** **Process Flow/Representation** 21](#_Toc200932868)

[**Activity Diagram** 21](#_Toc200932869)

[**4.4.** **Design Models [along with descriptions]** 22](#_Toc200932870)

[**5.** **Implementation** 24](#_Toc200932871)

[**5.1.** **Algorithm** 24](#_Toc200932873)

[**5.2.** **User Interface** 24](#_Toc200932874)

[**6.** **Testing and Evaluation** 24](#_Toc200932875)

[**6.1.** **Manual Testing** 24](#_Toc200932884)

[**6.1.1.** **System testing** 24](#_Toc200932892)

[**6.1.2.** **Unit Testing** 24](#_Toc200932893)

[**6.1.3.** **Functional Testing** 26](#_Toc200932894)

[**6.1.4.** **Integration Testing** 26](#_Toc200932895)

[**6.2.** **Automated Testing:** 27](#_Toc200932896)

[**6.2.1.** **Tools used:** 27](#_Toc200932897)

[**7.** **Conclusion and Future Work** 27](#_Toc200932898)

[**7.1.** **Conclusion** 27](#_Toc200932900)

[**7.2.** **Future Work** 27](#_Toc200932901)

[**8.** **References** 28](#_Toc200932902)

**LIST OF FIGURES**

Fig 2.1 Plantix 12

Fig 2.2 Plantix 12

Fig 3.1 Use Case Diagram 13

Fig 4.1 System Arcitecture 19

Fig 4.2 Data Representation 20

Fig 4.3 Activity Diagram 21

Fig 4.4 Sequence Diagram 22

Fig 4.5 Sequence Diagram 22

Fig 4.6 Sequence Diagram 23

**LIST OF TABLES**

Table 3.2.1 Use Case UC-1: User Login / Signup 13

Table 3.2.2 Use Case UC-2: User Management 14

Table 3.2.3 Use Case UC-3 Model Monitoring 14

Table 3.2.4 Use Case UC-4 Data Upload 15

Table 3.2.5 Use Case UC-5 Statistics 15

Table 3.2.6 Use Case UC-6 Upload Plant Image 15

Table 3.2.7 Use Case UC-7 View Prediction Results 16

Table 3.2.8 Use Case UC-8 Manage User Profiles 16

Table 5.1 Login Unit Testcase 25

Table 5.2 Edit Profile Unit Testcase 25

Table 5.3 Login Function Testcase 26

Table 5.4 Integration Testcase 26

Table 5.5 Tools Used 27

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

An intelligent system with deep learning potential, the Plant Disease Diagnose System was designed to help farmers efficiently identify and manage plant diseases. The users can enter images of plants, which are processed by a trained machine learning model to produce diagnoses ailments and suggest solutions. To enhance prediction accuracy, the system also includes meteorological data, such as temperature and humidity. This enhances the usefulness and usability of the system for farmers in rural areas and reduces plant loss by utilizing accurate and timely interventions.

* 1. **Relevance to Course Modules**

Artificial Intelligence, Software Engineering, Database Management Systems, Image Processing, and Mobile Application Development are some of the core BCS courses that are related to this project.

* 1. **Project Background**

Farmers, particularly have lack of recourses to detect disease and struggle to diagnose diseases on time, leading to worst 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**

Plantix and AgroAI are used to detect plant disease, though adopted widely, have notable limitations. Studies show that improper image inputs can reduce prediction reliability by up to 30% in plant disease detection systems. These platforms often don’t provide proper input validation, allowing unrelated or blury images to be processed. In contrast, our system is built on a supervised deep learning model with automatic conversion of all images to PNG format, ensuring uniformity and better model accuracy. Additionally, we follow Human-Computer Interaction (HCI) principles to make the interface simple and intuitive. These enhancements aim to improve overall prediction accuracy and usability.

* 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. 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. Object-Oriented Design principles were followed to ensure modular, maintainable code structure.

1. 6. 1. **Rationale behind Selected Methodology**

Agile enables iterative development.

OOD supports modular, scalable design.

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.

4. 1. **Problem Statement**

Agricultural productivity is heavily impacted by plant diseases, especially in rural areas where farmers lack access 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**

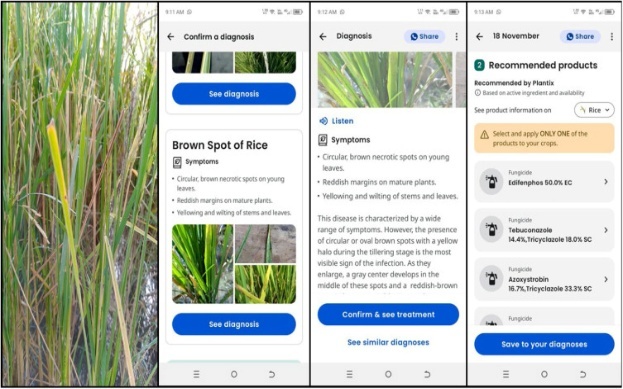
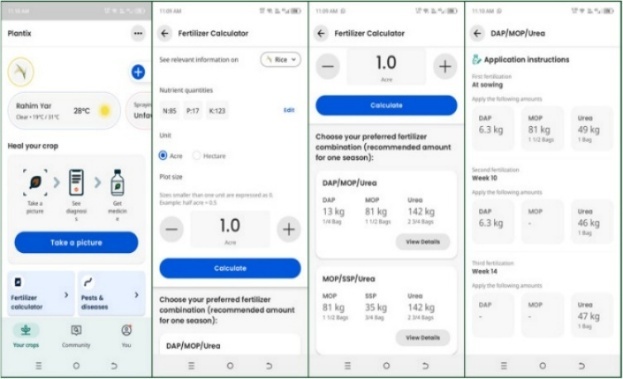
**Deliverables**

* Mobile 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, Admin)
* Disease history and plant tracking 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 brodely 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. Additionally, some users may find it difficult to use the user interface due to its incomplete optimization for those with low levels of computer literacy, despite its functionality. Its need on a steady internet connection to post photos and get forecast findings is another drawback, which could not be practical for farmers in rural or isolated locations. Additionally, while detecting illnesses, Plantix ignores environmental factors like temperature, humidity, and soil conditions, which might lower the precision of its forecasts under different circumstances. These drawbacks emphasize the need for a more user-friendly, ecologically sensitive, offline-capable, and localized solution, such as the Plant Disease Diagnose System.

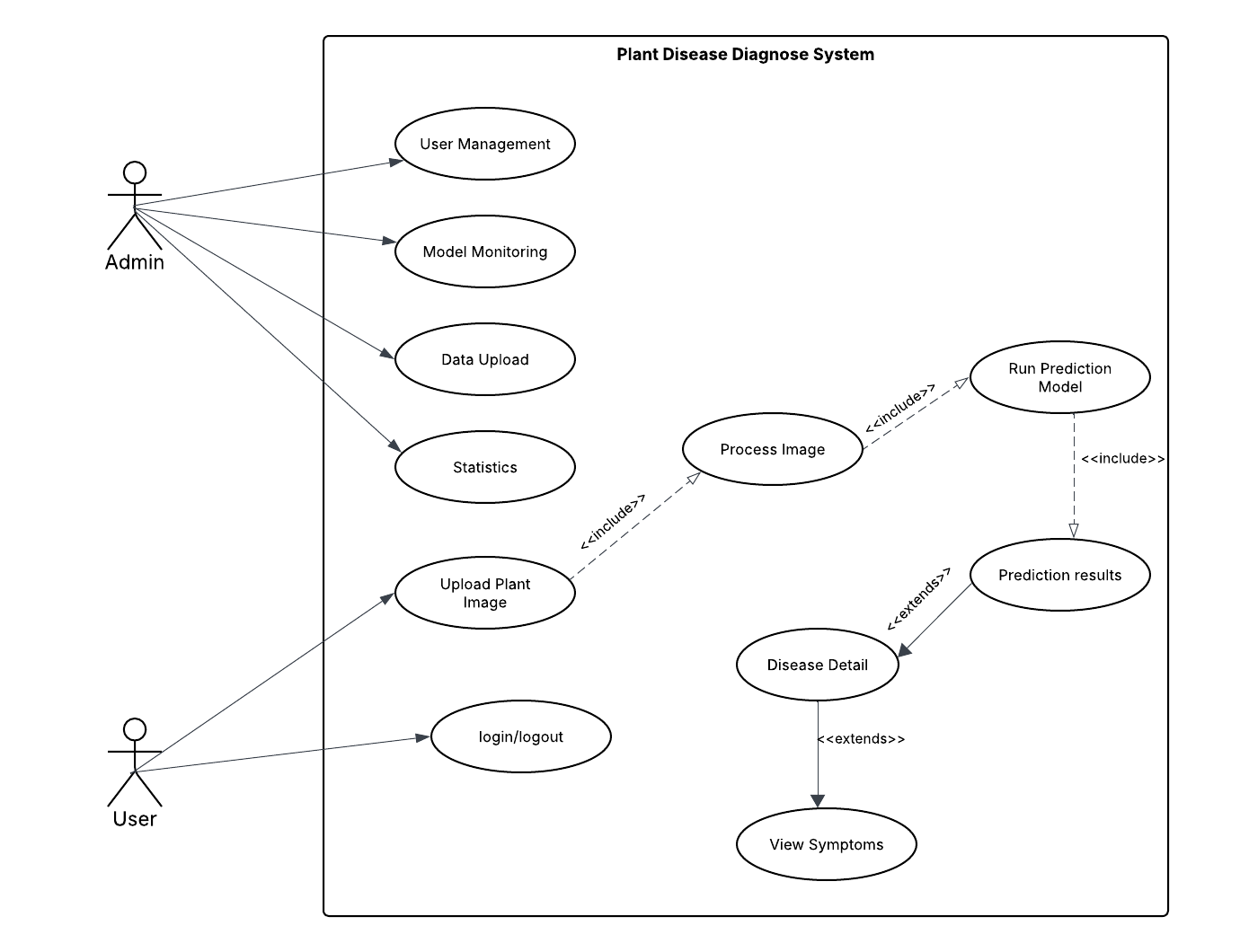
**

*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.

4. 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 3.2.2 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 3.2.3 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 3.2.4 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 3.2.5 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 3.2.6 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 3.2.7 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 3.2.8 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 allows users to register by providing an email address, username, and password.
* The system validates user credentials during login.

**Feature 2: Plant Image Upload**

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

**Feature 3: Disease Detection and Diagnosis**

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

**Usability:**

* The system will be easy to use that a non-tech user ca easily use it .

**Performance:**

* The system must process images and display predictions within 5 seconds.
* It should handle up to 500 concurrent users without any latency.

**Reliability:**

* The system downtime should be less or near to 99% uptime .

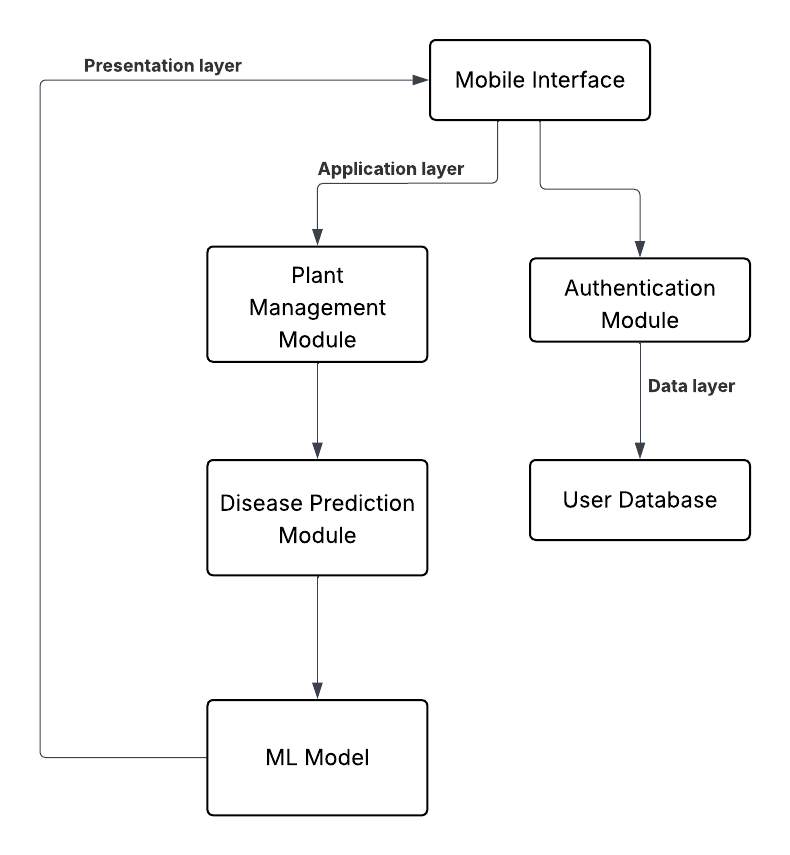
**Security:**

* All user data and its provided images should be encrypted that none user other than can see.
* Role-based access control should be implemented to restrict admin functionalities

1. **Design and Architecture**

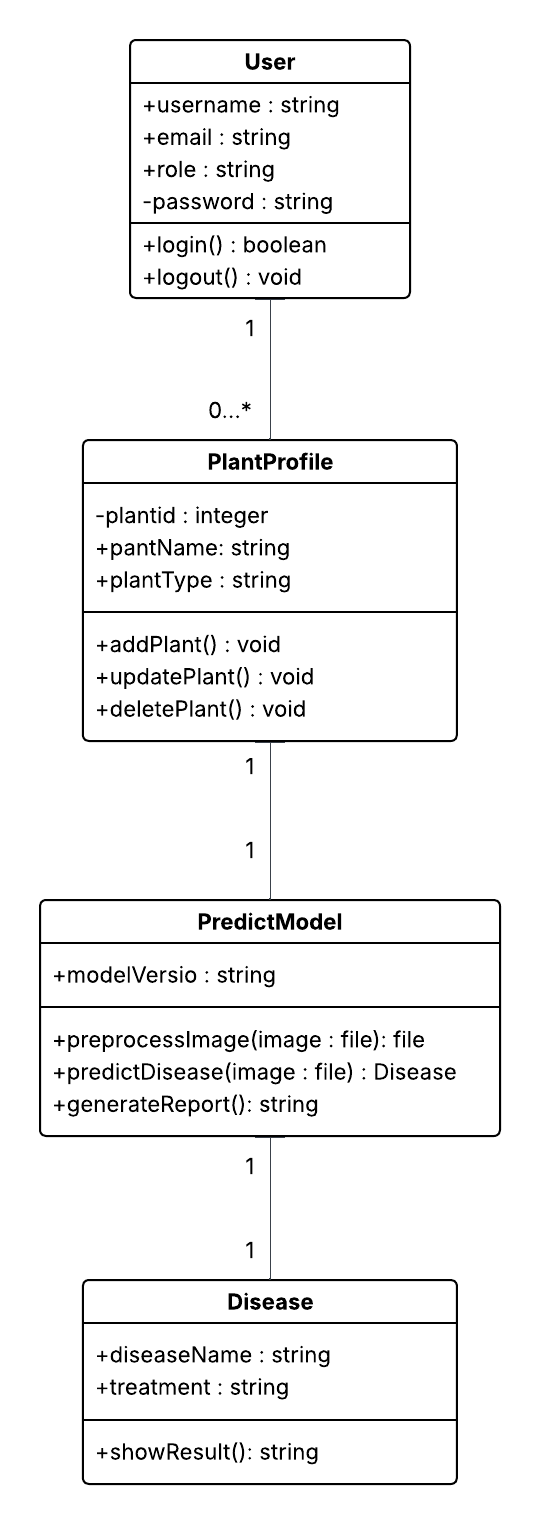
The following parts of Software Design Description (SDD) report should be included in this chapter.

8. 1. **System Architecture**

****

*Figure 4.1: Architecture Diagram*

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

****

*Figure 4.2: Class diagram*

* 1. **Process Flow/Representation**

**Activity Diagram**

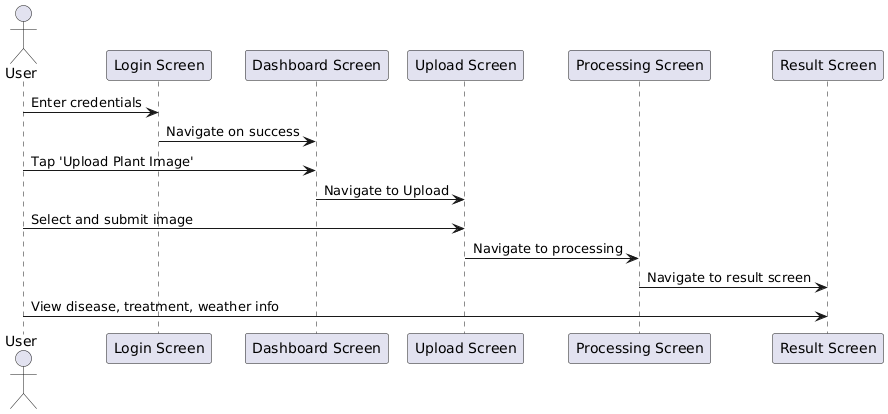
A screenshot of a diagram

AI-generated content may be incorrect.

*Figure 4.3: Activity Diagram*

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

**Sequence Diagram Mobile Application**

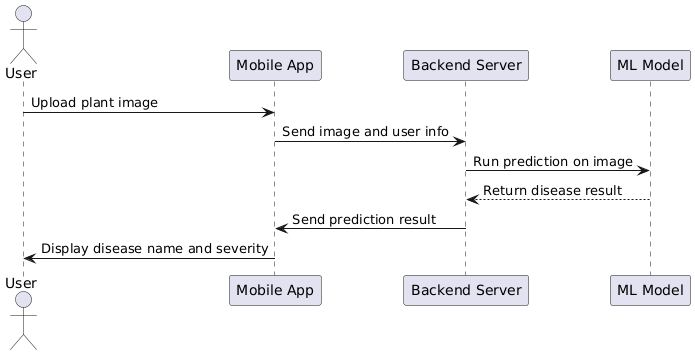
****

*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**

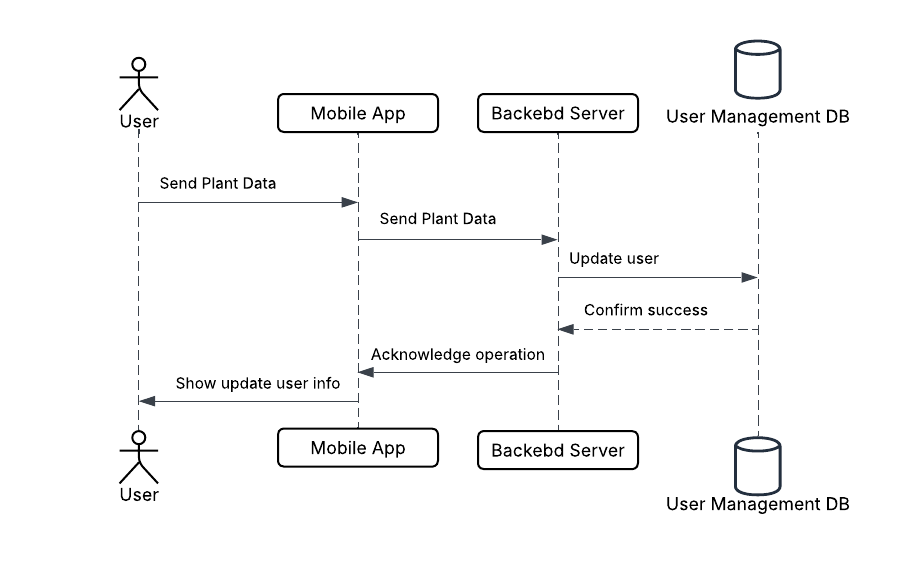
****

*Figure 4.5: Sequence Diagram*

**Description:**

When an image and its information are sent to the backend server via the mobile app, the interaction starts. After cleaning and resizing the picture, the preprocessing module receives it from the server and sends it to the deep learning (ML) model. After processing the picture, the model provides the anticipated illness. After that, the app receives this prediction from the backend, and the outcome is shown on the user interface.

**Sequence Diagram User Management**

****

*Figure 4.8: Sequence Diagram*

**Description:**  
By creating an account, signing in, or changing their profile information, the user may control their account information. The User Management Module creates, authenticates, or updates the user's records by interacting with the database and backend server after receiving the user credentials or updated data from the app. The system notifies the user of the status of their request by sending a confirmation back to the app when the action is successfully finished. This procedure guarantees a seamless authentication process and the safe management of user data.

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

The illness prediction algorithm, which makes use of a convolutional neural network (CNN), is the central component of the Plant Diagnose System. This system determines the most likely illness by analysing submitted leaf photos.

**Pseudocode for Disease Prediction:**

* Input: Plant image
* Preprocess image (resize, normalize, remove noise)
* Load trained CNN model
* Pass image through model
* Get prediction probabilities for each disease
* Return disease with highest probability and confidence score
* 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**






9. 1. **Manual Testing**




15. 1. 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:** 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 |

* + 1. **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:**
     1. **Tools used:**

*Table 5.5: Tools used*

|  |  |  |  |
| --- | --- | --- | --- |
| Tool Name | Tool Description | Applied On | Results |
| Postman | API testing tool | Functional API: login, prediction route | Pass |
| React Testing Library | Simulate UI interactions | UI components: Button clicks, inputs | Pass |

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

The Plant Diagnose System helps farmers by predicting plant diseases through a mobile interface using machine learning. With real-time diagnosis, weather awareness, and treatment suggestions, it enhances agricultural productivity.

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

1. **References**

TensorFlow Documentation: <https://www.tensorflow.org>

OpenCV Documentation: <https://opencv.org>

React Native Documentation: <https://reactnative.dev>

Matplotlib Documentation: https://matplotlib.org