Milestone 2

Kaylee Carstens (577331)- Group Leader

Kenneth McFerrier (577503)

Kealeboga Molefe (577482)

Keneilwe Ramatsui (577669)

Leon Botha (576378)

Table of Contents

System Overview 2

System architecture 2

Functional Description 3

Design Constraints 4

Detailed Design 5

Component Design 5

Interface Design 6

Data Design 7

User Interface Design 9

UI Layout 9

User Experience 11

Performance and Scalability 12

Performance requirements 12

Scalability 15

Security Design 18

Security requirements 18

Compliance 21

Testing and Validation 22

Test Plan 22

Validation Criteria 23

Deployment and Management 24

Deployment Plan 24

Maintenance Plan 25

System Overview

System architecture

The system is structured to automatically identify aloe forex phenology using image processing and machine learning, followed by the mapping of identified sightings using geographical data. The architecture consists of several key modules:

- *Data Collection and Storage Module:*
  This module uses the **Google Street View API** to gather images of landscapes where aloe forex phonologies may be present. The collected images are stored in a **Google Cloud Storage** database for further processing.
- *Preprocessing and Feature Extraction Module:*
  Using **OpenCV**, this module preprocesses the images by applying techniques like resizing, denoising, and normalization. This ensures that the images are uniform in size and quality, making them suitable for machine learning.
- *Machine Learning (ML) Classification Module:*
  A **TensorFlow-based neural network** is trained on labeled datasets of plant images to detect specific stages of flowering (buds, flowers, fruits). This model is responsible for identifying whether an image contains an aloe forex and at what life stage.

Code snippet for classification:

```
def classify_flowering_stage(image):
    preprocessed_image = preprocess_image(image)
    prediction = model.predict(preprocessed_image)
    return prediction
```

- *Phenology Visualization Module:*
  After classification, the system visualizes the flowering stages over time. Using **matplotlib** and **seaborn**, this module provides visual

representation of monthly flowering trends, helping researchers track the plant's life cycle across the year.

Code snippet for phenology visualization:

```
def visualize_phenology(monthly_flowering):
    plt.figure(figsize=(10, 6))
    sns.lineplot(data=monthly_flowering, palette="tab10", linewidth=2.5)
    plt.fill_between(monthly_flowering.index, monthly_flowering['Buds'], alpha=0.3, color='green')
    plt.fill_between(monthly_flowering.index, monthly_flowering['Flowers'], alpha=0.3, color='orange')
    plt.fill_between(monthly_flowering.index, monthly_flowering['Fruit'], alpha=0.3, color='blue')
    plt.xlabel('Month')
    plt.ylabel('Count')
    plt.title('Time-based Identification of Flowering Stages')
    plt.show()
```

- *Geolocation Mapping Module:*
  This module integrates the **Google Maps API** to display the geographical locations where invasive species are identified. The identified locations are mapped, providing a spatial view of species distribution.
- *Interactive Dashboard:*
  A **Tableau dashboard** allows stakeholders to view real-time data, including the spatial distribution of the plants and phenology reports. Users can explore flowering stages, map visualizations, and temporal trends interactively.

Functional Description

- *Image Collection:* The Google Street View API fetches images based on specified geographic locations. The system stores these images for further analysis. This can be automated based on set parameters

```
def fetch_images_from_street_view(location):
    # Call the Street View API for images of the specified location
    response = gmaps.streetview(location)
    return response
```

- *Image Preprocessing:* Preprocessing is performed on each collected image. Images are resized, denoised, and standardized using OpenCV to ensure consistency for the machine learning model.

```
def preprocess_image(image):
    # Resizing and standardizing image for ML input
    processed_image = cv2.resize(image, (224, 224))
    return processed_image
```

- *Plant Stage Classification:* The ML model classifies plants into life stages like Buds, Flowers, or Fruit based on preprocessed images. This is essential for tracking the plant's phenological stages over time.

- *Visualization and Reporting:* After classification, the system generates visual reports on the flowering stages by month. These reports are available to stakeholders via the Tableau dashboard.

- *Geolocation Mapping:* Identified plant species and their life stages are mapped using GPS data to show where specific aloe forex phenology has been detected.

Design Constraints

- *Data Quality:* The system relies on high-quality images from Google Street View. Variations in image resolution, lighting conditions, or obstructions can reduce the accuracy of classification. Thus, image preprocessing plays a critical role in maintaining model accuracy.

- *Accuracy Requirements:* The machine learning model aims for at least **90% accuracy** in classifying the flowering stages. A lower accuracy may lead to misidentifications, causing incorrect tracking of invasive species.

- *Scalability:* As the system grows and more regions are monitored for invasive species, it needs to handle increased data loads. This demands cloud infrastructure to store and process large volumes of images efficiently.

- *Performance:* To avoid delays in processing and classification, the system uses **GPU acceleration** where necessary. This ensures that the machine learning model can process multiple images concurrently and deliver insights quickly.

- *Security:* Access to collected images, geographical data, and machine learning results must be secured. Only authorized personnel should be able to view or modify data, which is implemented via Google Cloud IAM.

Detailed Design

Component Design

Navigation Component:

- Contains Exit buttons which are accessible from every page.
- Every page has specific buttons which are related to the content on the page e.g. Add Data, and Start Processing buttons for our image processing page

Image Upload Component:

- Includes an upload button with drag-and-drop support.
- Displays a preview of uploaded images with options to remove or edit.

Data Input Component:

- Edit boxes for users to input location, date.
- Data validation logic is part of this component.

Processing Bar Component:

- Displays a progress bar with real-time feedback on the processing of uploaded images.
- Includes a time estimation feature and cancel option.

Data Table Component:

- A table displaying processed images and corresponding data such as classification and location.
- Allows sorting, filtering, and pagination for better usability.

Export Data Component:

- A button for exporting the table data to an Excel file, with post-export confirmation.

Weather Viewing Component:

- Displays weather data in a visually accessible format and integrates with filters for weekly or daily weather.

Interface Design

The user interface follows a consistent and clean layout to ensure easy navigation and usability.

Page Layout:

- Header: includes page titles for ease of interpretation.
- Main Section: Centralized area for image upload, data input, and the table for displaying processed data.

Button Placement:

- Buttons are prominently placed at the bottom of the screen for easy access and navigation.
- All actions have clear, descriptive labels to avoid user confusion.

Form Fields and Inputs:

- Edit boxes for data input are large enough, well-spaced, and clearly labelled.
- Labels near the edit boxes to guide users about what information to input.

Progress Bar:

- Located on its own page to avoid confusion, and so that it can be clearly visible during processing.

Table Design:

- Neat, grid-like structure with alternate row shading to make it easy to read.
- Columns should be resizable and sortable.

Weather Data View:

- Graphical weather data is placed on a separate page, with intuitive filters that let users explore weather conditions by location, using the drop-down menu which will include all the uploaded images.

Data Design

The underlying data structures support the application crucially for smooth operation and data flow:

Input Data:

- Images: Uploaded images that need to be processed for phenology identification. These images will be stored temporarily during processing and then referenced in the table.
- User Input Data: Fields like location and date. These inputs are validated and stored temporarily during the session.

Processed Data:

- Phenological Classification: The result of processing each image (e.g., Bud, Flower, Fruit).
- Associated Metadata: Date, location, weather data, and other relevant tags.

Export Data:

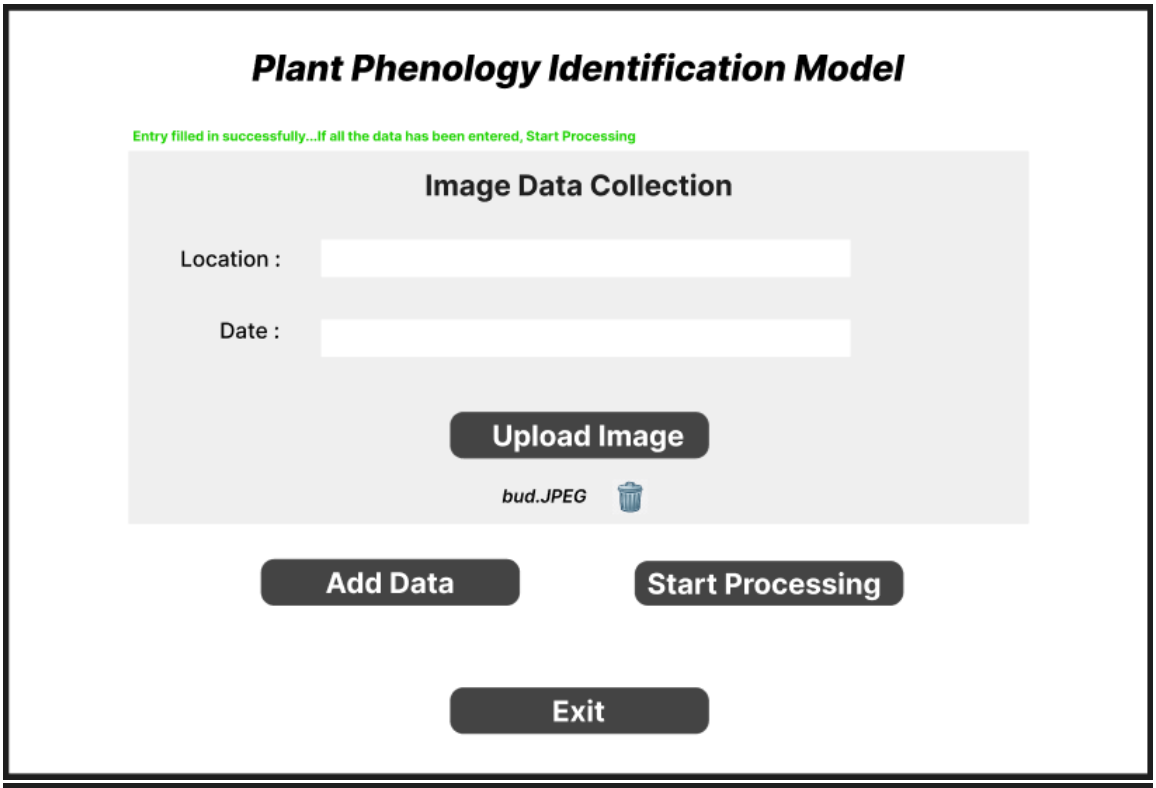
- The processed image data, along with metadata are exported to an Excel file.

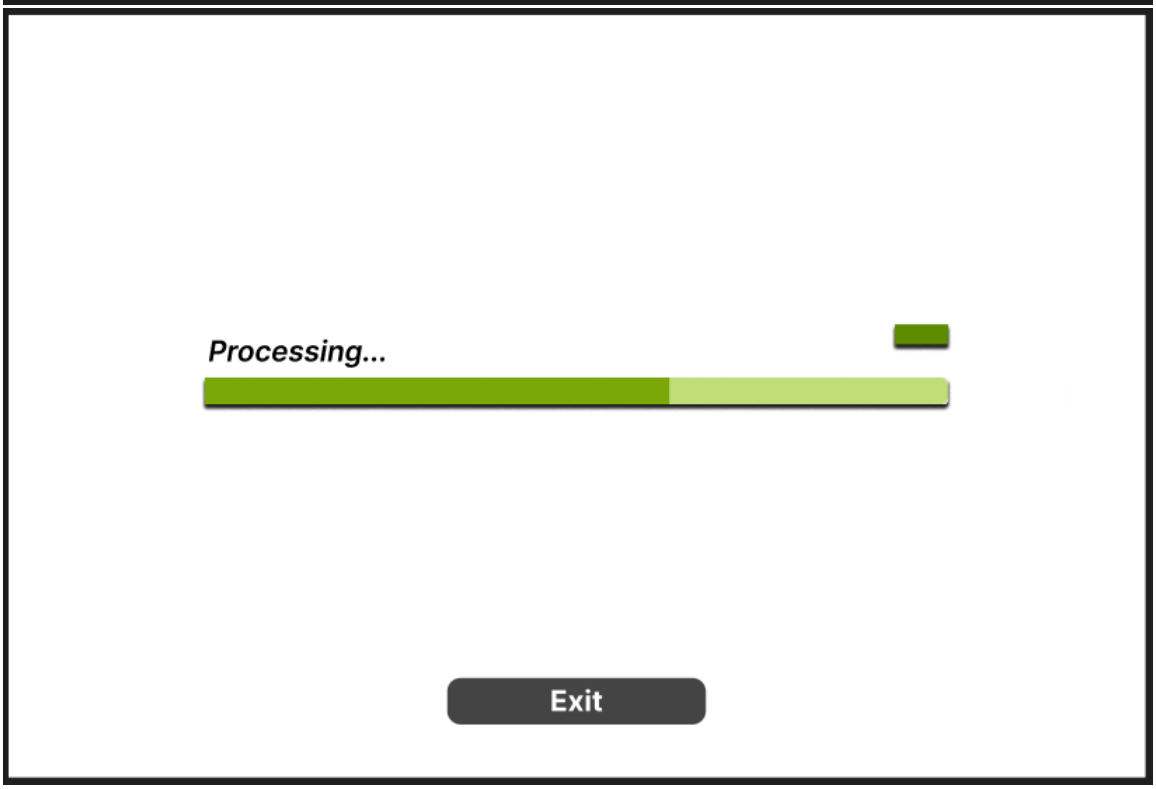
Weather Data:

- Integrated from a weather service, linked to each image via the location and date fields.
- Stored temporarily for analysis and displayed in the weather viewing component.

User Interface Design

UI Layout





Results

Image	Classification	Location	Date
bud.JPEG	Bud	Pretoria	05/09/24

Export To DB

Show Weather

Exit

Weather Page

Select Image From Menu : bud.JPEG

Display Weather

Back

Exit

User Experience

Users experience approach to ensure its seamless and user-friendly:

Smooth Navigation:

- Each page has clearly labelled buttons like Exit, Add Data, and Start Processing placed consistently at the bottom for easy access.
- Exit prompts the user to save any unsaved work before closing, ensuring data is not lost accidentally.
- Add Data ensure there are no empty spaces before submission, while Start Processing immediately initiates the analysis process without requiring additional steps.

Upload Experience:

- The Upload Button allows for multiple images to be uploaded at once and supports drag-and-drop functionality for ease of use.
- Once an image is uploaded, the interface displays a small preview to confirm successful upload before processing.

Data Input:

- Edit boxes provide clear labels/placeholder text to guide users on what data to enter.
- Validation is implemented to avoid errors in input, ensuring users cannot move forward without entering all required fields.

Process Feedback:

- The processing bar gives real-time feedback, updating users on the status of their task.
- The processing bar includes estimated time remaining and percentage completion to manage user expectations and reduce frustration if the process is lengthy.

Ordered Data Display:

- The table is well-structured, with sortable columns to allow users to quickly find the information they need.
- Scrollable table is implemented to keep the interface clean for large datasets.

Exporting Data:

- The Export Data button is easily accessible after processing and give users feedback when data is successfully exported with a pop-up message.

Weather Component:

- On the weather page, the weather viewing component presents data in a clear and visual format for different weather conditions.
- Filters are available for users to select specific locations or date ranges.
- The weather data updates dynamically based on user input.

Additional enhancements:

- Responsiveness: All elements (buttons, tables, progress bars) adapt well on mobile and tablet devices.
- Tooltips and Help Prompts: Context-sensitive tooltips are available, to guide users through complex or unfamiliar features like exporting data or understanding weather components.
- Confirmation Prompts: For important actions, confirmation from the user is required to avoid unintended outcomes.

Performance and Scalability

Performance requirements

The performance requirements should focus on ensuring that the machine learning model runs efficiently, that the system can handle large datasets and that users can interact with the system in a timely and responsive manner.

Response time

- The system should allow users to upload images quickly, the upload time should *be* **less than 5 seconds**.
- The time taken by the machine learning model to classify an image should be minimized and should **not take longer than 10-15 seconds**.
- The UI should respond to user actions **within 1 second** and with immediate feedback for any ongoing process
- Exporting the classifications should result to an excel file and should take **less than 10 seconds.**

Throughput

- The system should be able to handle **at least 50 concurrent users** uploading and classifying images without significant delays.
- The system should allow batch uploading and processing of **up to 100 images** in one session and process these within **a 2–3-minute window** depending on server capacity.

Data Handling Capacity

- The system should be able to handle large datasets with **up to 100 000 labelled images.**
- When gathering environmental data, the system should be able to process and integrate data in real-time without causing noticeable delays in image classification.

Machine Learning Model Performance

- The machine learning model's accuracy should **be above 90%** for the phenological stages (bud, flower, fruit).
- The initial model training should be completed within a reasonable time frame, with the full dataset being trained **in less than 4 hours** on appropriate hardware.
- Once the model is trained. The time to generate predictions for a new image should be **less than 10 seconds**.

System Load Management

- As the number of users and image uploads increases, the system should be able to handle the increase in volume. For instance, there should be no appreciable performance degradation when the system is able to accommodate **ten times** the initial user base or dataset size.
- For large-scale usage, consider implementing load balancing to distribute the computational load of classifying images and handling requests across multiple servers or cloud instances.

Network Performance

- When a user accesses the platform remotely, the system should reduce the amount of network latency. For most interactions (such as uploading images and receiving classification results), a reliable network connection should allow for a latency of **less than 100 ms**.
- Large datasets can be uploaded and exported using the system **at least 10 Mbps** data transfer speeds.

Storage Performance

- Performance optimisation should be applied to accommodate a growing image dataset, with most queries requiring less than a second to retrieve stored data.
- Scalable storage ought to be offered by the system to support an expanding image dataset. Solutions for cloud storage can manage a growing volume of photographs and their associated metadata.

Error Handling

- The system should be able to recover from failures (such as interrupted image uploads or classification failures) by providing users with options to retry the operation or automatically resuming processes without data loss.
- The system should be resilient, making certain that vital parts, such as the database or the machine learning engine, are backed up.

Resource Usage Optimization

- The machine learning tasks should make efficient use of the available CPU/GPU resources, particularly model training and inference. When doing intensive tasks, the CPU should only use **between 60 % and 80 %** of its power. To cut down on processing times overall, GPU acceleration should be used.
- Memory usage for processing images and training models should not exceed reasonable bounds.

Security Performance

- Ensure that all data exchanges (such as image uploads and classification exports) are encrypted to protect against unauthorized access.
- Processes for user login and access control shouldn't cause any noticeable delays. The process of authenticating a user should **take less than two seconds.**

Scalability

Scalability refers to the system's ability to handle increased workload or usage without compromising performance.

User Scalability

- The system should initially support at least 50 concurrent users
- As the project grows, the system should be built to scale, able to support **hundreds or thousands of users**. This could entail leveraging cloud infrastructure that automatically scales in response to demand, increasing load balancing, or adding additional servers.

Data Scalability

- There should be a considerable increase in data that the system can handle. Initially, it should manage datasets **with up to 100,000 images** and their associated metadata.

- The system needs to be ready for increasingly bigger datasets, maybe reaching the millions of records, particularly if additional Aloe Ferox photos or new plant species are supplied.

Machine Learning Scalability

- Additional data may be used as the system develops to enhance the machine learning model. The system should enable retraining models with big datasets, and the training process should be designed to employ distributed computing or GPUs for greater performance.
- When the number of users or images being processed increases, the inference process should not slow down significantly. The system should support the use of **GPU acceleration** or even **cloud-based inference** to handle higher throughput.

Compute Scalability

- At first, it might be enough to have a couple of high-performance servers. However, additional computer resources can be needed as the workload grows.
- It should be able to handle spikes in demand, the system should be able to scale using cloud-based infrastructure that dynamically adds more computing resources based on the current load.
- Using containers and orchestration tools like Kubernetes can help scale computing resources horizontally by adding more container instances to manage the increased workload.

Network Scalability

- The network should be able to manage increased data transfer, particularly for image uploads, as the system scales in terms of users without experiencing appreciable increases in latency. Response times can be accelerated and the burden on the primary servers decreased by using **CDNs (Content Delivery Networks)** to cache and serve static content, such as photos.
- Make sure the system supports geographically dispersed users with the least amount of latency possible if the project is utilised by researchers

or stakeholders in several locations by utilising edge computing or regional cloud data centres.

Storage Scalability

- Ensure that the system can easily expand its storage capacity without requiring major overhauls. Cloud storage options, such as Amazon S3 or Google Cloud Storage, provide seamless scalability, automatically handling increased data volume.
- As the database grows, sharding (splitting the database into smaller, more manageable parts) can help maintain performance by distributing the load across multiple databases.

UI/UX Scalability

- The user interface should continue to function well and be responsive as the user base expands, even when there are many people using it at once. This could mean optimising the UI rendering, caching specific parts, and shifting demanding computations to background jobs.
- To manage large datasets in the UI (such as displaying classified images), use pagination or lazy loading to only load a portion of the data at once, improving performance.

Cost Scalability

- As the project grows, cost control becomes crucial. Pay-as-you-go models in which you **only pay for the resources used** are frequently offered by cloud providers. Overprovisioning can be avoided, and resource allocation can be optimised with the use of cost monitoring tools.

Security Scalability

- As the system scales and more users upload sensitive geolocation or environmental data, ensure that the security infrastructure (such as encryption, access control, and data validation) scales proportionally to protect user data without degrading performance.
- As the system grows in size and more users contribute sensitive environmental or geolocation data, make sure that the security

architecture (including data validation, access control, and encryption) grows in line with the system to safeguard user data without compromising effectiveness.

Security Design

Security requirements

The security requirements for this project are crucial to ensure data integrity, protect sensitive information, and safeguard against unauthorized access.

Data Protection

- Encryption is required for all data, including location data, classification results, and photos, both in transit and at rest.
- Use **AES-256 encryption** for data at rest and **TLS (Transport Layer Security)** for data in transit to secure communication between clients and servers.
- To make sure that files are not changed or modified during the upload process, use a secure protocol like HTTPS and apply integrity checks.

Access control & Authentication

- Ensure secure user authentication via **username and password**. For further security, think about multi-factor authentication (MFA).
  To prevent brute force attacks, passwords should be saved securely using **hashing algorithms** like PBKDF2 with the appropriate amount of salting.
- Enable role-based access control (RBAC) to restrict access to various system components based on user roles. Only authorised users should be able to manage data, including exporting and uploading new photos.
- Integrate **OAuth** or **Single Sign-On (SSO)** for seamless and secure authentication, particularly when scaling to more users

Data Integrity & Validation

- Verify all user input thoroughly to guard against SQL injection, cross-site scripting (XSS), and file upload vulnerabilities. Also, make sure

uploaded images adhere to size and format specifications to prevent overload attacks.
- Use **checksums** or **hashing** to verify the integrity of files during transfer, ensuring that no corruption occurs.

Data Privacy & Compliance

- Make sure the system conforms with national and international data protection laws, especially when managing geolocation or personal data, such as the **GDPR** (General Data Protection Regulation) for users in Europe and the **POPIA** (Protection of Personal Information Act) for users in South Africa.
- Limit the quantity of personal or sensitive data acquired to the bare minimum needed for the project, and employ techniques to anonymise sensitive user data, such as geolocation information, if it is not required for the core functionality.

Network Security

- Use **firewalls** to protect the system from unauthorized external access. Configure firewalls to limit access to specific ports and protocols necessary for the operation of the web application.
- To protect the system from assaults that overwhelm the server with excessive traffic, use Distributed Denial of Service (DDoS) security. To lessen the impact of such attacks, use providers like Cloudflare or comparable services.

API Security

- Secure all APIs using **OAuth 2.0** or **JWT (JSON Web Tokens)** to ensure that only authorized users or systems can interact with the application
- Use rate restriction for API calls, particularly for endpoints that are accessible to the public, to stop abuse and overload assaults.

System Logging & Monitoring

- To track activity and identify suspect behaviour, keep thorough records of all user actions (such as logins, data uploads, and exports). These

logs should contain timestamps, user IDs, IP addresses, and the actions taken.
- Make sure logs are kept in a secure location and are routinely checked for odd activities.
- Use an **Intrusion Detection System (IDS)** to monitor network traffic and alert the system administrators in case of potential security breaches.

Backup & Recovery

- Implement automated daily or weekly backups for all critical data, including images, metadata, and machine learning models.
- Ensure backups are encrypted and stored securely off-site or in a cloud-based backup solution.
- Create and test a disaster recovery plan on a regular basis to make sure the system can recover from malfunctions, lost data, or security breaches. This covers data recovery, system restore processes, and service restart timeframes.

Security Testing

- Regularly conduct **vulnerability scans** on the system to detect security flaws, outdated dependencies, or misconfigurations.
- Use tools like **OWASP ZAP** or **Nessus** to test for vulnerabilities like SQL injection, cross-site scripting, or weak authentication mechanisms.
- Penetration testing should be done on a regular basis to find and fix any possible security holes, particularly before to system launch or following significant modifications.

User Security Awareness

- Train users on best security practices, such as creating strong passwords, recognizing phishing attempts, and protecting sensitive information.
- Provide users with a security guide explaining how to protect their accounts, securely upload data, and report any suspicious activity.

Physical Security

- If you are utilising an on-premises server, make sure that it is physically secure by establishing CCTV monitoring, limiting access to the server room, and implementing biometric or keycard access.
- When employing cloud infrastructure, make sure the cloud provider (such as AWS, Google Cloud, or Azure) follows industry requirements, delivers encryption, and conforms to high security standards.

Model Security

- Ensure the integrity of the machine learning model by implementing mechanisms to detect and prevent tampering or poisoning attacks during training or inference.
- Make sure that only authorised users can query or utilise the model by restricting access to model endpoints when deploying machine learning models.
- Put in place defences against adversarial assaults, in which malicious inputs are used to trick the model into classifying things incorrectly.

Compliance

- *Encryption* of sensitive data (in transit and at rest).
- *Access control* with proper authentication (e.g., OAuth, MFA).
- *User rights* to control, access, and delete their data.
- *Regular audits and monitoring* for potential data breaches or vulnerabilities.
- *Anonymization* of sensitive data when necessary.
- *Data breach notification* within specific timeframes (e.g., 72 hours for GDPR).

Testing and Validation

Test Plan

Unit testing:

Goal:

Make sure that every single part such as the machine learning model, the downloading and organising of the images, data splitting, image labelling and image preprocessing performs as intended.

Components that need to be tested:

- *Image preprocessing script:* Ensure that images are stored in the .npy format, scaled appropriately, and normalised. Make sure that nothing corrupts or loses data in the process.
- *Data Splitting Script:* Verify that the training, validation, and testing sets of data have been appropriately divided and that the appropriate labels have been applied.
- *Downloading and organizing script***:** Make sure all the images are downloaded and organized according to the sheets in the excel sheet.
- *Machine learning model:* Verify that the model architecture is constructed correctly and that it can be stored and trained.
- *Image labelling script:* Ensure that the pre-processed images are labelled correctly.
- *The prediction Aspect:* Verify that the model can be updated with new images and that the predictions it makes are correct.

Performance testing:

Goal:

Verify that the system satisfies performance benchmarks in terms of prediction time and model training speed.

Tests that will be run:

- Determine how long it takes to complete tasks like prediction, model training, and image preprocessing.

- Make sure the model is trained in an acceptable amount of time (for example, less than two minutes), and ensure that the model training does not use too much memory.

User acceptance testing:

Goal:

Make sure the user interface satisfies user expectations and is simple to use.

This test will focus on the following:

- *Weather data:* Verify that the data is presented in an understandable manner and that it flows naturally into the predictions.
- *Prediction outcomes:* Verify that information is presented in an understandable manner and that there is enough feedback regarding the weather and prediction accuracy.
- *Experience with image uploading:* Make sure the user interface (UI) makes it easy for users to upload images and view progress or error notifications.
- *Adaptability:* Test the user interface's responsiveness across various screen sizes and devices, such as desktops, tablets, and smartphones.

Validation Criteria:

Accuracy:

On the validation dataset, the machine learning model needs to get an accuracy of at least 90%.

Labelling accuracy:

Check that predictions match with the correct plant stage (flower, bud, or fruit).

User satisfaction:

Gather user opinions about the usefulness, visual appeal, and accuracy of the system's output.

Weather data integration:

Verify that changes in phenology predictions are associated with changes in weather data. For example, when the weather is ideal, the model predictions for flowers should increase.

Deployment and Management

Deployment Plan

Creating the Development Environment:

- *Environment:* Setting up separate environments for development, staging, and production will help to guarantee consistency and stability throughout the project.
- *Version Management:* To manage collaboration and code versions, we should use Github. Branches should be created on Github for updated functionality and problem fixes.
- *Testing:* Before going into production, we have to finish all unit, performance, and user testing.

Deploy to a server:

- Deploy the server on a cloud platform like Google Cloud to manage image processing, the model training and predictions and the integration of the weather data.

User Interface Deployment:

- Use a web server to host the user interface. Make sure the frontend and backend API are connected for predictions and real-time data access.

Training the model in production:

- *Initial Training:* After the system is up and running, use previously processed data to train the machine learning model. Save the

trained model for use in predictions in the production environment after it has finished training.
- *Retraining:* Implement an automated retraining process that retrains the model using new data on a regular basis to increase accuracy over time.

Integration of the weather data:

- To find and handle real-time weather data, establish a connection with a weather data API, such as the OpenWeather API.

Maintenance Plan

Continuous maintenance:

- *Bug fixes and enhancements:* Keep an eye out for any bugs or performance problems that users may report, and make changes as needed. Resolve any issues and provide updates frequently.
- *Periodic maintenance:* Plan regular system maintenance to verify server health, database integrity, and security patches.

Model Maintenance:

- We can use model versioning tools, like MLflow to monitor several iterations of the machine learning model. In the event that the new model doesn't work well, we can easily go back to a previous version.

Data backup:

- Make sure that all model files, weather data, and pre-processed image data are backed up on a weekly basis. Backups should be kept offsite in a safe location (like cloud storage).

Monitoring:

- Track system performance and spot possible bottlenecks in real time.

User support and instructions:

- Support Team: Provide a help desk or support team where users can report problems and ask for help.
- User instructions: Make sure end users have access to clear instructions on how to engage with weather data, upload images, and analyse predictions.

Scaling:

- *Developing New Features*: Make provisions for future extra features, such as including advanced weather models, introducing new plant species, or developing the model's architecture.
- *Expanding the infrastructure:* Make sure the infrastructure can support the increased load as the system expands. Make use of cloud platforms' auto-scaling features to adjust resources automatically in order to meet demand.