**AgriP - Testing**

Testing is a critical process in software development, applied at various stages to ensure the quality and functionality of the application. It involves evaluating a system or its components with the intent to find whether it satisfies the specified requirements. Testing is useful because it identifies defects, ensures reliability, and validates that the product meets user expectations and standards. It is typically applied during different phases of development, from individual units to the complete integrated system, helping to deliver a robust and error-free application.

**Features of the Application**

The application developed for my Bachelor’s thesis is an AI-based crop yield prediction system. It uses data such as levels of potassium (K), nitrogen (N), phosphorus (P), the area under cultivation, the irrigation method, etc. to forecast crop yields. The frontend of the application allows users to input these parameters through numerical inputs and dropdown menus, ensuring ease of use and preventing incorrect data entry. The core functionality of the application lies in its backend, where machine learning models process the input data to provide accurate yield predictions.

**Testing Techniques During Development**

During the development of the AI-based crop yield prediction system, various testing techniques should be employed to ensure the application’s reliability and accuracy. Unit testing should be used to validate the functionality of individual components, especially the prediction algorithms. Integration testing should focus on the interaction between the frontend input validation and the backend prediction API. System testing should encompass end-to-end scenarios, including data input, prediction generation, and result display, ensuring that the entire system works seamlessly together.

**Unit Testing**

For the main feature of predicting crop yield, unit testing can be performed on the prediction algorithm. This involves creating test cases with known input data for parameters such as K, N, P, area, and irrigation method, and verifying that the output yield predictions are as expected. Edge cases and a range of typical values should be tested to ensure the algorithm handles all possible inputs correctly. Mock data can be used to simulate various farming scenarios, ensuring the prediction model consistently provides accurate and reliable results.

**Integration Testing**

Integration testing for the crop yield prediction application should focus on the interaction between the frontend and backend, specifically how data is transmitted and processed. Although the frontend validation ensures that users can only input correct data types, it's crucial to test the backend API endpoints directly to verify robustness against potential misuse. Tools like Postman can be used to send HTTP requests directly to the API, bypassing frontend restrictions. This approach allows testing for various scenarios, including valid data submission, handling of edge cases, and ensuring that the API correctly processes or rejects invalid data. Testing should also cover how the backend handles missing or malformed inputs, ensuring that proper error messages are returned and the system remains secure against potential exploits.

**System Testing**

System testing for the crop yield prediction application should encompass comprehensive end-to-end scenarios to ensure the entire system functions correctly under real-world conditions. Given the presence of multiple AI algorithms—two that predict crop yields using different methodologies and a third that modifies inputs and results for a different prediction angle—it's essential to test these algorithms extensively. Scenarios should include submitting valid data through the frontend, verifying that each algorithm produces accurate and consistent results, and comparing the outputs of the two different yield prediction algorithms to ensure they align reasonably. Additionally, the third algorithm should be tested for its ability to correctly process modified inputs and generate accurate predictions. Testing should also include stress testing to ensure the system can handle high volumes of data and concurrent requests without performance degradation. This holistic approach ensures that the system remains robust, accurate, and reliable, providing valuable insights for users.