SYSTEM TESTING

This section focuses on thorough testing procedures used to confirm the system's ability to work as intended and perform well. A comprehensive testing plan was carefully crafted and put into action to fully assess the system across various aspects. This section details the specific testing methods used, why they were chosen, and the valuable information gained from the testing process.

Unit Testing

Unit tests were used to verify the working of functions in the system. This includes email, NPK, temperature, and humidity values' boundaries. These tests were implemented using pytest to validity of data entering into the system. The snippet below shows an example of a unit test verifying the functionality of APIs or services that provide soil and environmental data for a specific geographic location.

```python
@pytest.mark.asyncio
async def test_is_isda_available():

    data = await getNPK(latitude=-6.880122, longitude=39.160506)

    for key in ["Nitrogen","Potassium","Phosphorus","ph","bulk_density"]:
        assert key in data.keys(), f"Key '{key}' not found in the response"

@pytest.mark.asyncio
async def test_is_openmeteo_available():
    data = await getEnvironmentData(latitude=-6.880122, longitude=39.160506)

    for key in ["temperature","humidity","rainfall"]:
        assert key in data.keys(), f"Key '{key}' not found in the response"
```

Figure 1: Unit test checking for availability of iSDA and OpenMeteo

Integration Testing

The goal of integration testing was to ensure that different modules and components worked together seamlessly. This involved testing the interaction between data retrieval modules, the predictive engine, and the user interface using tool (fast API) for continuous integration. The process confirmed that the integrated system components interacted correctly, with data flowing smoothly from input to output.