Plant Disease Detection Apps

CMPE 187 Deliverable 3 - AI Test Automation

Professor Jerry Gao

Team 1

Tejas Kulkarni, Umesh Singh, Nathan Kim, Mitchell Sayer

1. Test Automation Introduction
    - 1.1 Test Automation Focus
    - 1.2 Test Automation Strategy
2. Test Automation Solutions
    - 2.1 Test Scripts
3. Test Complexity & Coverage
4. Test Automation Summary

1. Test Automation Introduction

1.1 Test Automation Focus

Our AI testing automation focus is to accurately determine whether our selected applications (FarmAssistX, Sick Plant Disease Identifier, PlantDiseaseIdentifier, and DoctorP) can correctly identify the Potato Blight, Tomato Leaf Mold, Strawberry Leaf Scorch, and Corn Common Rust diseases.

We will vary test conditions by testing photos with varying attributes: plant picture clarity, plant picture zoom level, plant life stage, picture angle, and disease area. We believe that testing using these varying attributes will give us a comprehensive understanding as to how accurate the selected AI applications are at detecting disease. Overall, out automation testing focuses on verifying that these applications can produce accurate results, regardless of the state they are in. The Testing model below shows all context and input criteria used for our test cases

1.2 Test Automation Strategy

Test Planning

Our test planning involves initially creating the proper testing scope and defining the testing objectives. For this, we refer to our initial deliverables regarding test scope and objectives. We will be testing the Potato Blight, Tomato Leaf Mold, Strawberry Leaf Scorch, and Corn Common Rust diseases for several different contexts and input variations.

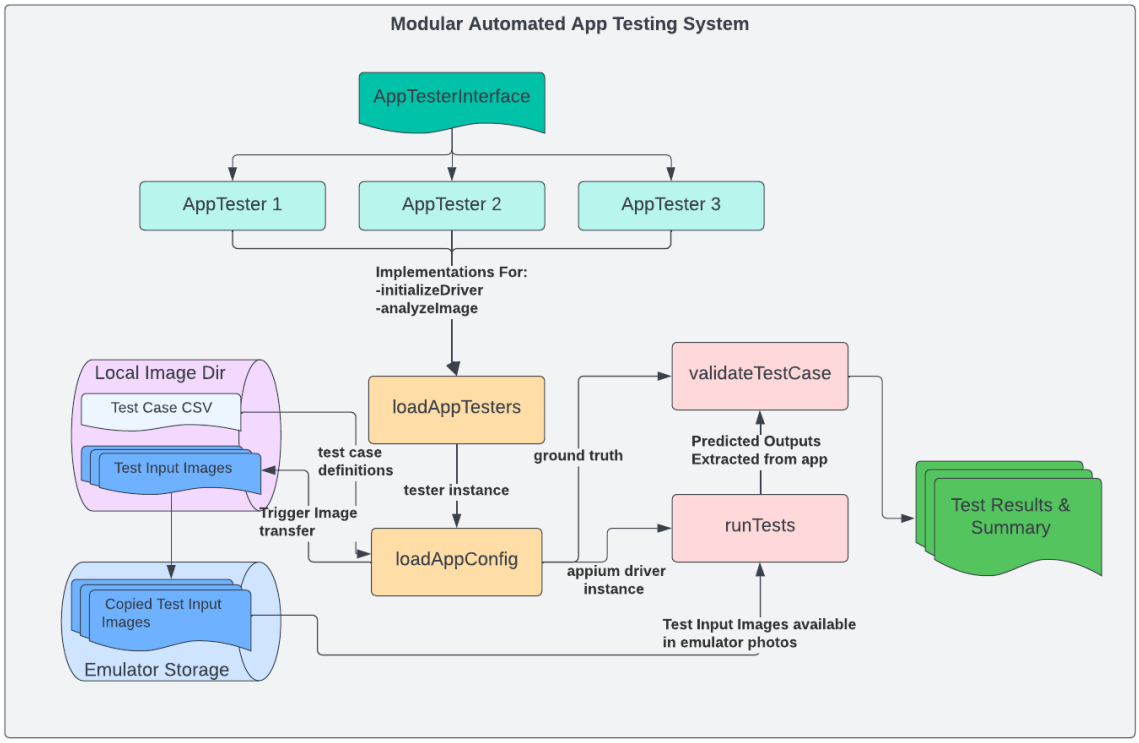
Test Execution

In order to conduct AI Plant Disease detection testing for various apps, we must set up the testing environment which will be primarily done through appium.
The diagram created below goes more in depth about our test automation strategy

Essentially, enables systematic automated testing of multiple APK files simple interface for integrating new applications automatic configuration of submodules automated test reporting CSV configuration file defines mapping of input image files -> expected prediction values for each test case
Automatically parses input image files and pushes them to the Android Emulator file system

This can be seen in the diagram below

2. Test Automation Solutions



Test Cases Used for all Apps

1	Birds-eye view, multiple spots, potato blight, green background
2	Low clarity, dark lighting, front view potato plant - blight
3	Potato blight, front view
4	Potato blight, dark setting
5	Strawberry leaf scorch, high clarity
6	Strawberry leaf scorch, low clarity
7	Corn common rust, one leaf, green background
8	Corn common rust, one leaf
9	Tomato leaf mold, birds eye view and regular
10	Tomato leaf mold, front view
11	Not a plant

12	Baby plant, strawberry leaf scorch
13	Potato blight, one singular spot on leaf, front view, high clarity and light
14	Tomato Leaf Mold, one singular spot on plant, front view, high clarity and light
15	Potato blight, multiple spots, one leaf, front view, high clarity and light
16	Potato blight, multiple spots, one plant, front view, high clarity and light
17	Strawberry leaf scorch, multiple spots, one leaf, front view, high clarity and light
18	Strawberry leaf scorch, multiple spots, one plant, front view, high clarity and light
19	Corn common rust, multiple spots, one leaf, front view, high clarity and light
20	Corn common rush, multiple spots, one plant, front view, high clarity and light
21	Tomato leaf mold, multiple spots, one leaf, front view, high clarity and light
22	Tomato leaf mold, multiple spots, one plant, front view, high clarity and light
23	Potato blight, multiple spots, one leaf, birds-eye view, high clarity and light
24	Potato blight, multiple spots, one plant, birds-eye view, high clarity and light
25	Strawberry leaf scorch, multiple spots, one leaf, birds-eye view, high clarity and light
26	Strawberry leaf scorch, multiple spots, one plant, birds-eye view, high clarity and light
27	Corn common rust, multiple spots, one leaf, birds-eye view, high clarity and light
28	Corn common rush, multiple spots, one plant, birds-eye view, high clarity and light
29	Tomato leaf mold, multiple spots, one leaf, birds-eye view, high clarity and light
30	Tomato leaf mold, multiple spots, one plant, birds-eye view, high clarity and light

These above test cases correspond to the images stated in the csv below.

2.1 Test Automation System Scripts

CSV file containing input image filenames and corresponding output ground truth data.

pic-data.csv

```
input_image,expected_plant,expected_disease
01.png,potato,blight
02.png,potato,blight
```

```
03.png,potato,blight
04.png,potato,blight
05.png,strawberry,scorch
06.png,strawberry,scorch
07.png,corn,rust
08.png,corn,rust
09.png,tomato,mold
10.png,tomato,mold
11.png,not plant,not plant
12.png,strawberry,scorch
13.png,potato,blight
14.png,tomato,mold
15.png,potato,blight
16.png,potato,blight
17.png,strawberry,scorch
18.png,strawberry,scorch
19.png,corn,rust
20.png,corn,rust
21.png,tomato,mold
22.png,tomato,mold
23.png,potato,blight
24.png,potato,blight
25.png,strawberry,scorch
26.png,strawberry,scorch
27.png,corn,rust
28.png,corn,rust
29.png,tomato,mold
30.png,tomato,mold
```

Interface to be implemented for each plant disease identification application to test. The initializeDriver function instantiates the Appium WebDriver instance and connects to a locally running Appium server.
The analyzeImage function contains the core logic for interacting with the applications to be tested.
The ‘index’ input corresponds to the image index in a lexicographically sorted order as uploaded by the core testing framework.

src/app_tester_interface.py

```
# Author: Mitchell Sayer
from abc import ABC, abstractmethod
```

```
class AppTesterInterface(ABC):
    @abstractmethod
    def initializeDriver(self):
        pass

    @abstractmethod
    def analyzeImage(self, driver, index):
        pass
```

Core implementation of the automated testing framework. Includes functionality to transfer test images to Android emulator, parse and process ground truth and results, and automatically instantiate and run all AppTester instances.

src/app_tester_core.py

```
# Author: Mitchell Sayer
import os
import csv
import base64
import importlib

from src.report_generator import generateReport

# ------- UTILITIES -------

def image_to_base64(image_path):
    with open(image_path, "rb") as image_file:
        binary_data = image_file.read()
        base64_data = base64.b64encode(binary_data).decode('utf-8')

    return base64_data

# ------- TESTING INFRASTRUCTURE -------

# Load AppTester implementations from app_scripts directory and instantiate.
def loadAppTesters():
    implementation_dir = f'{os.getcwd()}/src/app_scripts'

    # Load all Python files in the directory
    implementation_files = [f[:-3] for f in os.listdir(implementation_dir) if
f.endswith('.py') and f != '__init__.py']

    app_tester_instances = []
```

```
    # Import each implementation dynamically and run functions
    for implementation_file in implementation_files:
        module = importlib.import_module(f'.{implementation_file}', 'src.app_scripts')

        # Assuming each implementation class has the same name as the file
        app_tester_class = getattr(module, implementation_file)

        # Create an instance of the implementation class
        app_tester = app_tester_class()
        app_tester_instances.append(app_tester)
        print(f'\n######### Testing App: {implementation_file} #########')

    return app_tester_instances

# Load Image paths and ground truth values from pic-data.csv
def loadConfig(driver, csv_file_path, upload_images=True):
    path_base = os.getcwd()
    index = 0
    img_gt_pairs = []

    with open(csv_file_path, 'r') as csv_file:
        if upload_images:
            print(f'\nUploading test images...')

        csv_reader = csv.DictReader(csv_file)
        for row in csv_reader:
            if upload_images:
                # Push image from config to emulator storage.
                input_image = row['input_image']
                img_path = os.path.join(path_base, 'pic', input_image)
                destination_path = f"/storage/emulated/0/Pictures/{'0'*(3 -
len(str(index)))}{index}.png"

                img_base64 = image_to_base64(img_path)

                print(f'\t- {destination_path}')
                driver.push_file(destination_path, img_base64)

            # Store test case row
            img_gt_pairs.append(row)
            index += 1
```

```
    print(f'\nLoaded config from {csv_file_path}\n')
    return img_gt_pairs

# Both expected_result and actual_result are in the format:
#   - [PLANT_NAME, DISEASE_NAME]
def validateTestCase(expected_result, actual_result):
    # Check for malformed result lists
    if len(expected_result) != 2 or len(actual_result) != 2:
        raise ValueError("Both results must have size 2.")

    # Convert all values to lowercase
    expected_plant = expected_result[0].lower()
    expected_disease = expected_result[1].lower()
    predicted_plant = actual_result[0].lower()
    predicted_disease = actual_result[1].lower()

    # Check expected values are substring of predicted values
    correct_plant = expected_plant in predicted_plant
    correct_disease = expected_disease in predicted_disease

    return correct_plant, correct_disease

# Run test cases on a given AppTester instance
def runTests(driver, tester, test_cases):
    results = []
    init_index = 1 # lexically sorted image list index

    passed_tests = 0
    passed_plant = 0
    passed_disease = 0

    # Run analyzeImage with AppTester instance for each defined test image
    for test_index in range(init_index, len(test_cases)):
        row = test_cases[test_index-1]
        input_image = row['input_image']
        ground_truth = [row['expected_plant'], row['expected_disease']]

        print(f'\nAnalyzing {input_image}')
        print(f'\tExpected Result: {ground_truth}')

        extracted_result = tester.analyzeImage(driver, test_index)
        results.append(extracted_result)
```

```
        print(f'\tActual Result: {extracted_result}')

        # Determine test case outcomes
        if None not in extracted_result:
            plant_correct, disease_correct = validateTestCase(ground_truth,
extracted_result)
        else:
            plant_correct = False
            disease_correct = False

        if plant_correct and disease_correct:
            print('\t----------PASS-----------')
            passed_tests += 1
        else:
            print('\t----------FAIL-----------')
            if plant_correct:
                passed_plant += 1
            if disease_correct:
                passed_disease += 1

    return (passed_tests, passed_plant, passed_disease, test_index)
```

Script to generate accuracy report pdfs for each AppTester instance

src/report_generator.py

```
# Author: Mitchell Sayer
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from reportlab.lib import colors
from reportlab.platypus import Table, TableStyle
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from reportlab.lib import colors
from reportlab.graphics.shapes import Drawing
from reportlab.graphics.charts.piecharts import Pie
from reportlab.lib.validators import Auto
from reportlab.graphics.charts.legends import Legend

def generatePieChart(pdf, pass_count, fail_count):
    data=[pass_count, fail_count]
    labels = ['Pass Count','Fail Count']
    pie_colors=[colors.lightgreen, colors.indianred]
```

```
    drawing = Drawing(400, 300)

    pie = Pie()
    pie.x = 150
    pie.y = 50
    pie.width = 200
    pie.height = 200
    pie.data = data
    pie.labels = labels
    pie.slices.strokeWidth = 0.5

    for i, color in enumerate(pie_colors):
        pie.slices[i].fillColor = color

    drawing.add(pie)

    legend = Legend()
    legend.alignment = 'left'
    legend.x = 10
    legend.y = 70
    legend.colorNamePairs = Auto(obj=pie)
    drawing.add(legend)

    drawing.wrapOn(pdf, 50, 400)
    drawing.drawOn(pdf, 50, 400)

    return pdf


def generateReport(metrics):
    # metric = {
    #     'tester_name': f"{type(tester).__name__}",
    #     'total_tests': total_tests,
    #     'passed_tests': passed_tests,
    #     'plant_only_pass': plant_accuracy,
    #     'disease_only_pass': disease_accuracy
    # }

    try:
        for metric in metrics:
            report_file = f"./generated_reports/{metric['tester_name']}.pdf"
```

```
        pdf = canvas.Canvas(report_file, pagesize=letter)

        data = [
            ['Number of Test Cases',
            'Passed Cases Count',
            'Failed Cases Count',
            'Pass Rate',
            'Fail Rate']
        ]

        pass_count = metric['passed_tests']
        fail_count = metric['total_tests'] - pass_count
        plant_partial_pass = metric['plant_only_pass']
        disease_partial_pass = metric['disease_only_pass']
        total_test_cases = pass_count + fail_count

        data.append([
            total_test_cases,
            pass_count,
            fail_count,
            f'{pass_count/total_test_cases:.2%}',
            f'{fail_count/total_test_cases:.2%}'
        ])

        table_style = TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 12),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ])

        table = Table(data)
        table.setStyle(table_style)

        table.wrapOn(pdf, 50, 700)
        table.drawOn(pdf, 50, 700)

        title_text = f"Automation Test Report: {metric['tester_name']}"
```

```
            pdf.setFont('Helvetica-Bold', 16)
            pdf.drawCentredString(300, 750, title_text)

            pdf=generatePieChart(pdf, pass_count, fail_count)

            pdf.save()
    except Exception as error:
        print("Oops from Report Generation")
        print("The Error is",error)
```

Automated Testing Infrastructure entrypoint

./base_script.py

```
# Author: Mitchell Sayer
import time

from src.app_tester_core import loadAppTesters, loadConfig, runTests
from src.report_generator import generateReport

# ------- Main Entrypoint -------
def main():
    upload_images = False

    csv_file_path = './pic-data.csv'
    app_tester_instances = loadAppTesters()

    metrics = []
    for tester in app_tester_instances:
        driver = tester.initializeDriver()

        test_cases = loadConfig(driver, csv_file_path, upload_images=upload_images)
        time.sleep(5)

        passed_tests, passed_plant, passed_disease, total_tests = runTests(driver,
tester, test_cases)

        print('------- Tests Complete -------')
        print(f'\n\tTotal Test Count: {total_tests}')
        print(f'\tTotal Passed Tests: {passed_tests}')
```

```
        metric = {
            'tester_name': f"{type(tester).__name__}",
            'total_tests': total_tests,
            'passed_tests': passed_tests,
            'plant_only_pass': passed_plant,
            'disease_only_pass': passed_disease
        }
        metrics.append(metric)

        accuracy = passed_tests/total_tests * 100
        plant_accuracy = passed_plant / total_tests * 100
        disease_accuracy = passed_disease / total_tests * 100

        print(f'\tTotal Accuracy: {accuracy}%')
        print(f'\t\t- Plant Identification Only Accuracy: {passed_plant}/{total_tests}
{plant_accuracy}%')
        print(f'\t\t- Disease Identification Only Accuracy:
{passed_disease}/{total_tests} {disease_accuracy}%')

        driver.quit()

    generateReport(metrics)

if __name__ == '__main__':
    main()
```

2.1.1 PlantDiseaseIdentifier

Includes special logic to scroll through the image picker in the event that the desired image index is not present when the page first loads.

./app_scripts/PlantDiseaseIdentification.py

```
# Author: Mitchell Sayer
import os
import time

from src.app_tester_interface import AppTesterInterface

from appium import webdriver

from appium.options.common.base import AppiumOptions
```

```
from appium.webdriver.common.appiumby import AppiumBy

# For W3C actions
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

class PlantDiseaseIdentification(AppTesterInterface):
   def initializeDriver(self):
       base_path = os.getcwd()

       options = AppiumOptions()
       options.load_capabilities({
           "platformName": "Android",
           "appium:platformVersion": "14",
           "appium:appPackage": "com.faisalkabirgalib.plant_disease_detection",
           "appium:appActivity":
"com.faisalkabirgalib.plant_disease_detection.MainActivity",
           "appium:app": f"{base_path}/apks/Plant Disease
Detector_1.0.0_apkcombo.com.apk",
           "appium:deviceName": "emulator-5554",
           "appium:automationName": "UiAutomator2",
           "appium:ensureWebviewsHavePages": True,
           "appium:nativeWebScreenshot": True,
           "appium:newCommandTimeout": 10000,
           "appium:connectHardwareKeyboard": True
       })

       driver = webdriver.Remote("http://127.0.0.1:4723", options=options)

       return driver

   def analyzeImage(self, driver, index):
       wait = WebDriverWait(driver, 30)

       try:
           # Upload Image
           el1 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID,
"Pick Image")))
           el1.click()
           time.sleep(1)
```

```
            # File picker More Options
            el2 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID,
"More options")))
            el2.click()
            time.sleep(1)

            # Browse
            el3 = wait.until(EC.element_to_be_clickable((AppiumBy.ID,
"com.google.android.providers.media.module:id/title")))
            el3.click()
            time.sleep(1)

            # Google Photos More Options
            el4 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID,
"More options")))
            el4.click()
            time.sleep(1)

            # Sort by
            el5 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,

"//android.widget.TextView[@resource-id=\"com.google.android.documentsui:id/title\"
and @text=\"Sort by...\"]")))
            el5.click()
            time.sleep(1)

            # File name (A to Z)
            el6 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
                "//android.widget.CheckedTextView[@resource-id=\"android:id/text1\" and
@text=\"File name (A to Z)\"]")))
            el6.click()
            time.sleep(1)

            # Handle case of index > 15 (needs scroll)
            # This method only supports up to 30 files
            if index > 15:
                #swipe(startX, startY, endX, endY, duration)
                driver.swipe(530,2115, 525,200, 1000)
                adjusted_index = index - 9
            else:
                adjusted_index = index
            time.sleep(1)
```

```
            el7 = wait.until(EC.element_to_be_clickable((By.XPATH,
f"(//android.widget.ImageView[@resource-id=\"com.google.android.documentsui:id/icon_th
umb\"])[{adjusted_index}]")))
            el7.click()
            time.sleep(1)

            el8 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID,
"Run Model")))
            el8.click()

            result_path =
"//android.widget.FrameLayout[@resource-id=\"android:id/content\"]/android.widget.Fram
eLayout/android.view.View/android.view.View/android.view.View/android.view.View/androi
d.view.View[2]/android.view.View/android.view.View[7]/android.view.View[1]"
            model_result_locator = (By.XPATH, result_path)
            new_view_text =
wait.until(EC.presence_of_element_located(model_result_locator))
            model_result = new_view_text.get_attribute('content-desc').split('\n')[:-1]
            time.sleep(1)
            return model_result

        except Exception as error:
            print(f'error: {error}')
```

PlantDiseaseIdentifier Output - Testing System 1st Pass
mitchellsayer@Mitchells-MBP 187 % python3 base_script.py

######### Testing App: PlantDiseaseIdentification #########

Uploading test images...
- /storage/emulated/0/Pictures/000.png
- /storage/emulated/0/Pictures/001.png
- /storage/emulated/0/Pictures/002.png
- /storage/emulated/0/Pictures/003.png
- /storage/emulated/0/Pictures/004.png
- /storage/emulated/0/Pictures/005.png
- /storage/emulated/0/Pictures/006.png

- /storage/emulated/0/Pictures/007.png
- /storage/emulated/0/Pictures/008.png
- /storage/emulated/0/Pictures/009.png
- /storage/emulated/0/Pictures/010.png
- /storage/emulated/0/Pictures/011.png
- /storage/emulated/0/Pictures/012.png
- /storage/emulated/0/Pictures/013.png
- /storage/emulated/0/Pictures/014.png
- /storage/emulated/0/Pictures/015.png
- /storage/emulated/0/Pictures/016.png
- /storage/emulated/0/Pictures/017.png
- /storage/emulated/0/Pictures/018.png
- /storage/emulated/0/Pictures/019.png
- /storage/emulated/0/Pictures/020.png
- /storage/emulated/0/Pictures/021.png
- /storage/emulated/0/Pictures/022.png
- /storage/emulated/0/Pictures/023.png
- /storage/emulated/0/Pictures/024.png
- /storage/emulated/0/Pictures/025.png
- /storage/emulated/0/Pictures/026.png
- /storage/emulated/0/Pictures/027.png
- /storage/emulated/0/Pictures/028.png
- /storage/emulated/0/Pictures/029.png

Loaded config from ./pic-data.csv

Analyzing 01.png
Expected Result: ['potato', 'blight']
Actual Result: ['Tomato', 'Early blight']
----------FAIL----------

Analyzing 02.png
Expected Result: ['potato', 'blight']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

Analyzing 03.png
Expected Result: ['potato', 'blight']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

Analyzing 04.png
Expected Result: ['potato', 'blight']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

Analyzing 05.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Apple', 'Black rot']
----------FAIL----------

Analyzing 06.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Strawberry', 'Leaf scorch']
----------PASS----------

Analyzing 07.png
Expected Result: ['corn', 'rust']
Actual Result: ['Corn (maize)', 'Common rust ']
----------PASS----------

Analyzing 08.png
Expected Result: ['corn', 'rust']
Actual Result: ['Corn (maize)', 'Common rust ']
----------PASS----------

Analyzing 09.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Grape', 'Black rot']
----------FAIL----------

Analyzing 10.png
Expected Result: ['tomato', 'mold']

Actual Result: ['Apple', 'Black rot']
----------FAIL----------

Analyzing 11.png
Expected Result: ['not plant', 'not plant']
Actual Result: ['Potato', 'healthy']
----------FAIL----------

Analyzing 12.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Tomato', 'Early blight']
----------FAIL----------

Analyzing 13.png
Expected Result: ['potato', 'blight']
Actual Result: ['Cherry (including sour)', 'Powdery mildew']
----------FAIL----------

Analyzing 14.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Apple', 'Black rot']
----------FAIL----------

Analyzing 15.png
Expected Result: ['potato', 'blight']
Actual Result: ['Potato', 'Early blight']
----------PASS----------

Analyzing 16.png
Expected Result: ['potato', 'blight']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

Analyzing 17.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Strawberry', 'Leaf scorch']
----------PASS----------

Analyzing 18.png
    Expected Result: ['strawberry', 'scorch']
    Actual Result: ['Strawberry', 'Leaf scorch']
    ----------PASS----------

Analyzing 19.png
    Expected Result: ['corn', 'rust']
    Actual Result: ['Strawberry', 'Leaf scorch']
    ----------FAIL----------

Analyzing 20.png
    Expected Result: ['corn', 'rust']
    Actual Result: ['Strawberry', 'Leaf scorch']
    ----------FAIL----------

Analyzing 21.png
    Expected Result: ['tomato', 'mold']
    Actual Result: ['Strawberry', 'Leaf scorch']
    ----------FAIL----------

Analyzing 22.png
    Expected Result: ['tomato', 'mold']
    Actual Result: ['Apple', 'Black rot']
    ----------FAIL----------

Analyzing 23.png
    Expected Result: ['potato', 'blight']
    Actual Result: ['Strawberry', 'Leaf scorch']
    ----------FAIL----------

Analyzing 24.png
    Expected Result: ['potato', 'blight']
    Actual Result: ['Tomato', 'Septoria leaf spot']
    ----------FAIL----------

Analyzing 25.png

Expected Result: ['strawberry', 'scorch']
Actual Result: ['Strawberry', 'Leaf scorch']
----------PASS----------

Analyzing 26.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Apple', 'Black rot']
----------FAIL----------

Analyzing 27.png
Expected Result: ['corn', 'rust']
Actual Result: ['Corn (maize)', 'Cercospora leaf spot Gray leaf spot']
----------FAIL----------

Analyzing 28.png
Expected Result: ['corn', 'rust']
Actual Result: ['Corn (maize)', 'Common rust ']
----------PASS----------

Analyzing 29.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

------- Tests Complete -------
Total Test Count: 29
Total Passed Tests: 8

Total Accuracy: 27.586206896551722%
- Plant Identification Only Accuracy: 1/29 3.4482758620689653%
- Disease Identification Only Accuracy: 1/29 3.4482758620689653%

2.1.2 DoctorP

./app_scripts/DoctorP.py

```
import os
import time
```

```
    from ..app_tester_interface import AppTesterInterface

    from appium import webdriver

    from appium.options.common.base import AppiumOptions
    from appium.webdriver.common.appiumby import AppiumBy

    # For W3C actions
    from selenium.webdriver.support.ui import WebDriverWait
    from selenium.webdriver.support import expected_conditions as EC
    from selenium.webdriver.common.by import By

    class DoctorP(AppTesterInterface):
        def initializeDriver(self):
            base_path = os.getcwd()

            options = AppiumOptions()
            options.load_capabilities({
                "platformName": "Android",
                "appium:platformVersion": "12",
                "appium:appPackage": "com.pdd.pdd",
                "appium:appActivity": "com.pdd.pdd.MainActivity",
                "appium:app":
"C:\\Users\\Checkout\\Desktop\\DoctorPAPK.apk",
                "appium:deviceName": "emulator-5554",
                "appium:automationName": "UiAutomator2",
                "appium:ensureWebviewsHavePages": True,
                "appium:nativeWebScreenshot": True,
                "appium:newCommandTimeout": 3600,
                "appium:connectHardwareKeyboard": True
            })

            driver = webdriver.Remote("http://127.0.0.1:4723",
options=options)

            return driver

        def analyzeImage(self, driver, index):
            wait = WebDriverWait(driver, 30)
```

```
            try:
                if index == 9:
                    index = index+1
                if index == 12:
                    index = index+1
                # Upload Image
                if index == 1:
                    el1 =
wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID,
"Explore")))
                    el1.click()
                    time.sleep(1)
                else:
                    driver.press_keycode(4)

                # Picture Icon
                el2 =
wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"//android.widget.FrameLayout[@resource-id=\"android:id/content\"]/android
.widget.FrameLayout/android.widget.FrameLayout/android.view.View/android.v
iew.View/android.view.View/android.view.View/android.widget.ImageView[1]")
))
                el2.click()
                time.sleep(1)

                # File picker More Options
                if index == 1:
                    el3 =
wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID,
"Understood, I’m ready")))
                    el3.click()
                    time.sleep(1)

                # Permission
                if index == 1:
                    el4 =
wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"//android.widget.Button[@resource-id=\"com.android.permissioncontroller:i
d/permission_allow_foreground_only_button\"]")))
```

```
                    el4.click()
                    time.sleep(1)

                # Permission
                if index == 1:
                    el5 =
wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"//android.widget.Button[@resource-id=\"com.android.permissioncontroller:i
d/permission_allow_foreground_only_button\"]")))
                    el5.click()
                    time.sleep(1)

                # Image Icon
                el6 =
wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID,
"Photos")))
                el6.click()
                time.sleep(1)

                # Permission
                if index == 1:
                    el7 =
wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"//android.widget.Button[@resource-id=\"com.android.permissioncontroller:i
d/permission_allow_button\"]")))
                    el7.click()
                    time.sleep(1)

               # Google Photos More Options
                el8 =
wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "More
options")))
                el8.click()
                time.sleep(1)

                # Sort by
                el9 =
wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
```

```
"//android.widget.TextView[@resource-id=\"com.google.android.documentsui:id/title\" and @text=\"Sort by...\"]")))
                el9.click()
                time.sleep(1)

                # File name (A to Z)
                el10 =
wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,

"//android.widget.CheckedTextView[@resource-id=\"android:id/text1\" and @text=\"File name (A to Z)\"]")))
                el10.click()
                time.sleep(1)

                # Handle case of index > 15 (needs scroll)
                # This method only supports up to 30 files
                if index > 15:
                    #swipe(startX, startY, endX, endY, duration)
                    driver.swipe(530,2115, 525,200, 1000)
                    adjusted_index = index - 9
                else:
                    adjusted_index = index
                time.sleep(1)

                el11 = wait.until(EC.element_to_be_clickable((By.XPATH,

f"(//android.widget.ImageView[@resource-id=\"com.google.android.documentsui:id/icon_thumb\"])[{adjusted_index}]")))
                el11.click()
                time.sleep(1)

                # Press Send button
                el12 =
wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"//android.widget.FrameLayout[@resource-id=\"android:id/content\"]/android.widget.FrameLayout/android.widget.FrameLayout/android.view.View/android.view.View/android.view.View/android.widget.ImageView[2]")
))
                el12.click()
```

```
                time.sleep(7)

                result_path =
"//android.widget.ScrollView/android.view.View[3]/android.view.View/androi
d.view.View"
                model_result_locator = (By.XPATH, result_path)
                new_view_text =
wait.until(EC.presence_of_element_located(model_result_locator))
                model_result =
new_view_text.get_attribute('content-desc')

                result_path2 =
"//android.widget.ScrollView/android.view.View[2]/android.view.View/androi
d.view.View[1]/android.widget.ImageView"
                model_result_locator2 = (By.XPATH, result_path2)
                new_view_text2 =
wait.until(EC.presence_of_element_located(model_result_locator2))
                model_result2 =
new_view_text2.get_attribute('content-desc')

                time.sleep(1)
                result = [model_result, model_result2]
                return result


            except Exception as error:
                print(f'error: {error}')
```

Output file:-

Analyzing 01.png

Expected Result: ['potato', 'blight']

Actual Result: ['Potatoes', 'Black spots']

----------FAIL----------

Analyzing 02.png

Expected Result: ['potato', 'blight']

Actual Result: ['Rosemary', 'Anthocyanosis']

----------FAIL----------

Analyzing 03.png
Expected Result: ['potato', 'blight']
Actual Result: ['Blueberry', 'Powdery mildew']
----------FAIL----------

Analyzing 04.png
Expected Result: ['potato', 'blight']
Actual Result: ['Blueberry', 'Powdery mildew']
----------FAIL----------

Analyzing 05.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Strawberry', 'Leaf spot']
----------FAIL----------

Analyzing 06.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Strawberry', 'Leaf spot']
----------FAIL----------

Analyzing 07.png
Expected Result: ['corn', 'rust']
Actual Result: ['Corn', 'Rust']
----------PASS----------

Analyzing 08.png
Expected Result: ['corn', 'rust']
Actual Result: ['Corn', 'Rust']
----------PASS----------

Analyzing 09.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Tomatoes', 'Mosaic virus']
----------FAIL----------

Analyzing 10.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Tomatoes', 'Mosaic virus']
----------FAIL----------

Analyzing 11.png
Expected Result: ['not plant', 'not plant']
Actual Result: ['Orchid', 'Mechanical damage']
----------FAIL----------

Analyzing 12.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Tomatoes', 'Mosaic virus']
----------FAIL----------

Analyzing 13.png
Expected Result: ['potato', 'blight']
Actual Result: ['Orchid', 'Mechanical damage']
----------FAIL----------

Analyzing 14.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Blueberry', 'Powdery mildew']
----------FAIL----------

Analyzing 15.png
Expected Result: ['potato', 'blight']
Actual Result: ['Potato', 'Early blight']
----------PASS----------

Analyzing 16.png
Expected Result: ['potato', 'blight']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

Analyzing 17.png
Expected Result: ['strawberry', 'scorch']

Actual Result: ['Strawberry', 'Leaf scorch']
----------PASS----------

Analyzing 18.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Strawberry', 'Leaf scorch']
----------PASS----------

Analyzing 19.png
Expected Result: ['corn', 'rust']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

Analyzing 20.png
Expected Result: ['corn', 'rust']
Actual Result: ['Blueberry', 'Powdery mildew']
----------FAIL----------

Analyzing 21.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Blueberry', 'Powdery mildew']
----------FAIL----------

Analyzing 22.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Apple', 'Black rot']
----------FAIL----------

Analyzing 23.png
Expected Result: ['potato', 'blight']
Actual Result: ['Strawberry', 'Leaf scorch']
----------FAIL----------

Analyzing 24.png
Expected Result: ['potato', 'blight']
Actual Result: ['Blueberry', 'Powdery mildew']
----------FAIL----------

Analyzing 25.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Strawberry', 'Leaf scorch']
----------PASS----------

Analyzing 26.png
Expected Result: ['strawberry', 'scorch']
Actual Result: ['Blueberry', 'Powdery mildew']
----------FAIL----------

Analyzing 27.png
Expected Result: ['corn', 'rust']
Actual Result: ['Corn (maize)', 'Cercospora leaf spot Gray leaf spot']
----------FAIL----------

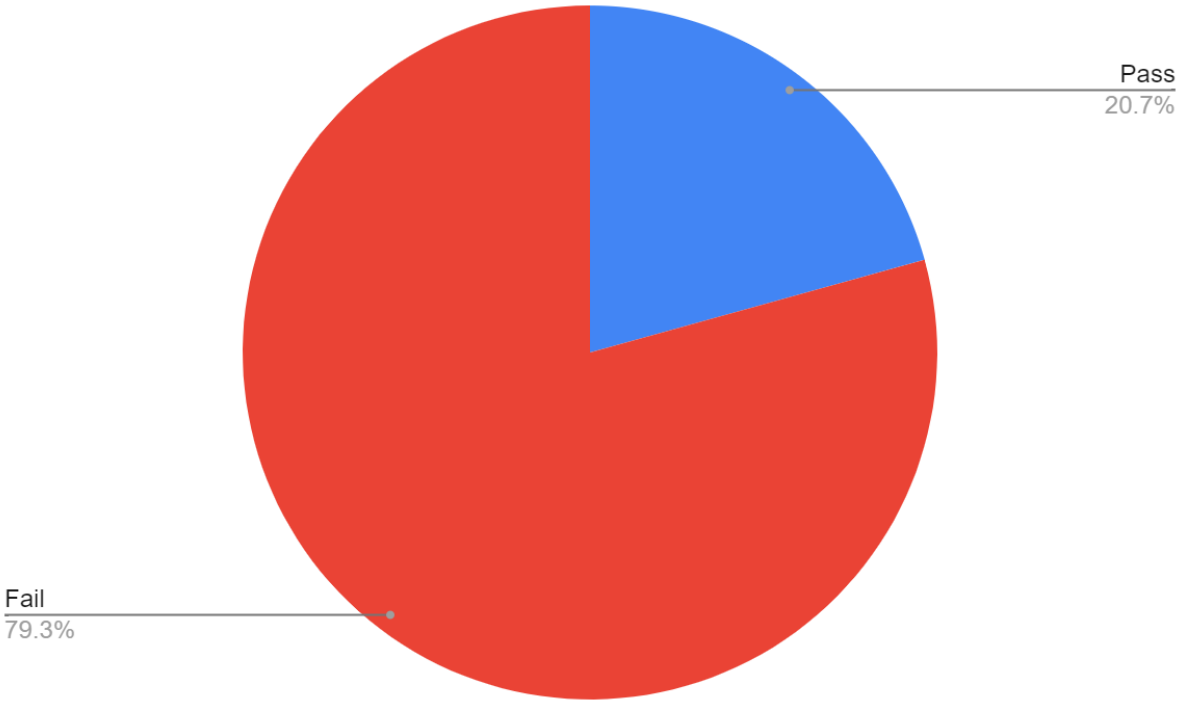
Analyzing 28.png
Expected Result: ['corn', 'rust']
Actual Result: ['Orchid', 'Mechanical damage']
----------FAIL----------

Analyzing 29.png
Expected Result: ['tomato', 'mold']
Actual Result: ['Strawberry', 'Leaf spot']
----------FAIL----------

------- Tests Complete -------
Total Test Count: 29
Total Passed Tests: 6

Total Accuracy: 20.689655172324672%

DoctorP Results:
6/29 test cases passed
The graph depicting this result is shown below



Software Used:
Utilizing and integrating various AI testing applications: Appium (application script runner), Android Studio, Node.js, NPM
Writing JSON files and python scripts which initiated the execution and evaluated our test cases.

2.1.3 FarmAssistX

./app_scripts/FarmAssistX.py

```
import os
import time

from ..app_tester_interface import AppTesterInterface

from appium import webdriver

from appium.options.common.base import AppiumOptions
from appium.webdriver.common.appiumby import AppiumBy
```

```
# For W3C actions
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

class PlantDiseaseIdentification(AppTesterInterface):
    def initializeDriver(self):
        base_path = os.getcwd()

        options = AppiumOptions()
        options.load_capabilities({
            "platformName": "Android",
            "appium:platformVersion": "14",
            "appium:appPackage": "com.example.farmassist",
            "appium:appActivity": "com.example.farmassist.MainActivity",
            "appium:app": f"{base_path}/apks/app-arm64-v8a-release.apk",
            "appium:deviceName": "emulator-5554",
            "appium:automationName": "UiAutomator2",
            "appium:ensureWebviewsHavePages": True,
            "appium:nativeWebScreenshot": True,
            "appium:newCommandTimeout": 10000,
            "appium:connectHardwareKeyboard": True
        })

        driver = webdriver.Remote("http://127.0.0.1:4723",
options=options)

        return driver

    def analyzeImage(self, driver, index):
        wait = WebDriverWait(driver, 30)

        try:

            username_field = wait.until(EC.element_to_be_clickable((By.ID,
"your_username_field_id")))
            username_field.send_keys("tejninja7@gmail.com")
            time.sleep(5)
```

```
            password_field = wait.until(EC.element_to_be_clickable((By.ID,
"your_password_field_id")))
            password_field.send_keys("Hello123!")
            time.sleep(5)
            # Detect Button Bottom Nav Bar
            el3 =
wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "LOG
IN")))
            el3.click()
            time.sleep(5)

            # Detect Button Bottom Nav Bar
            el4 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"//android.view.View[@content-desc=\"Manage Monitor Detect
Me\"]/android.view.View[3]")))
            el4.click()
            time.sleep(1)


            # Add Image Button
            el5 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"//android.view.View[@content-desc=\"No image
selected\"]/android.view.View[2]")))
            el5.click()
            time.sleep(1)

            # Browse
            el6 = wait.until(EC.element_to_be_clickable((AppiumBy.ID,
"com.google.android.providers.media.module:id/title")))
            el6.click()
            time.sleep(1)

            # More Options
            el7 =
wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "More
options")))
            el7.click()
            time.sleep(1)

            # More Options (Browse Button)
```

```
            el8 =
wait.until(EC.element_to_be_clickable((AppiumBy.ID,"com.google.android.pro
viders.media.module:id/title")))
            el8.click()
            time.sleep(1)

            #Google Photos Button
            el9 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
"(//android.widget.ImageView[@resource-id='com.google.android.documentsui:
id/app_icon'])[3]")))
            el9.click()
            time.sleep(1)

            # Browse
            el10 = wait.until(EC.element_to_be_clickable((AppiumBy.ID,
"//android.widget.RelativeLayout")))
            el10.click()
            time.sleep(1)


            # File name (A to Z)
            el11 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,

"//android.widget.CheckedTextView[@resource-id=\"android:id/text1\" and
@text=\"File name (A to Z)\"]")))
            el11.click()
            time.sleep(1)

            # Handle case of index > 15 (needs scroll)
            # This method only supports up to 30 files
            if index > 15:
                #swipe(startX, startY, endX, endY, duration)
                driver.swipe(530,2115, 525,200, 1000)
                adjusted_index = index - 9
            else:
                adjusted_index = index
            time.sleep(1)

            el7 = wait.until(EC.element_to_be_clickable((By.XPATH,
```

```
f"(//android.widget.ImageView[@resource-id=\"com.google.android.documentsu
i:id/icon_thumb\"])[{adjusted_index}]")))
            el7.click()
            time.sleep(1)

            el12 =
wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "Run
Model")))
            el12.click()

            result_path =
"//android.widget.FrameLayout[@resource-id=\"android:id/content\"]/android
.widget.FrameLayout/android.view.View/android.view.View/android.view.View/
android.view.View/android.view.View[2]/android.view.View/android.view.View
[7]/android.view.View[1]"
            model_result_locator = (By.XPATH, result_path)
            new_view_text =
wait.until(EC.presence_of_element_located(model_result_locator))
            model_result =
new_view_text.get_attribute('content-desc').split('\n')[:-1]
            time.sleep(1)
            return model_result

        except Exception as error:
            print(f'error: {error}')
```

Analyzing 01.png

Expected Result: ['potato', 'blight']

Actual Result: [Potato, Blight]

----------PASS----------

Analyzing 02.png

Expected Result: ['potato', 'blight']

Actual Result: [Tomato, Leaf Mold]

----------FAIL----------

Analyzing 03.png

Expected Result: ['potato', 'blight']
Actual Result: [Potato, Blight]
----------PASS----------

Analyzing 04.png
Expected Result: ['potato', 'blight']
Actual Result: [Potato, Blight]
----------PASS----------

Analyzing 05.png
Expected Result: ['strawberry', 'scorch']
Actual Result: [Strawberry, Leaf Scorch]
----------PASS----------

Analyzing 06.png
Expected Result: ['strawberry', 'scorch']
Actual Result: [Strawberry, Leaf Scorch]
----------PASS----------

Analyzing 07.png
Expected Result: ['corn', 'rust']
Actual Result: [Corn, Common rust]
----------PASS----------

Analyzing 08.png
Expected Result: ['corn', 'rust']
Actual Result: [Corn, Common rust]
----------PASS----------

Analyzing 09.png
Expected Result: ['tomato', 'mold']
Actual Result: [Tomato, Leaf Mold]
----------PASS----------

Analyzing 10.png
Expected Result: ['tomato', 'mold']
Actual Result: [Healthy, Healthy]

----------FAIL----------

Analyzing 11.png
Expected Result: ['not plant', 'not plant']
Actual Result: [Healthy, Healthy]
----------FAIL----------

Analyzing 12.png
Expected Result: ['strawberry', 'scorch']
Actual Result: [Tomato, Leaf Mold]
----------FAIL----------

Analyzing 13.png
Expected Result: ['potato', 'blight']
Actual Result: [Potato, Blight]
----------PASS----------

Analyzing 14.png
Expected Result: ['tomato', 'mold']
Actual Result: [Tomato, Leaf Mold]
----------FAIL----------

Analyzing 15.png
Expected Result: ['potato', 'blight']
Actual Result: [Potato, Blight]
----------PASS----------

Analyzing 16.png
Expected Result: ['potato', 'blight']
Actual Result: [Potato, Blight]
----------PASS----------

Analyzing 17.png
Expected Result: ['strawberry', 'scorch']
Actual Result: [Healthy, Healthy]
----------FAIL----------

Analyzing 18.png
Expected Result: ['strawberry', 'scorch']
Actual Result: [Healthy, Healthy]
----------FAIL----------

Analyzing 19.png
Expected Result: ['corn', 'rust']
Actual Result: [Corn, Common Rust]
----------PASS----------

Analyzing 20.png
Expected Result: ['corn', 'rust']
Actual Result: [Corn, Common Rust]
----------PASS----------

Analyzing 21.png
Expected Result: ['tomato', 'mold']
Actual Result: [Strawberry, Leaf scorch]
----------FAIL----------

Analyzing 22.png
Expected Result: ['tomato', 'mold']
Actual Result: [Potato, Blight]
----------FAIL----------

Analyzing 23.png
Expected Result: ['potato', 'blight']
Actual Result: [Strawberry, Leaf scorch]
----------FAIL----------

Analyzing 24.png
Expected Result: ['potato', 'blight']
Actual Result: [Potato, Blight]
----------PASS----------

Analyzing 25.png
Expected Result: ['strawberry', 'scorch']

Actual Result: [Strawberry, Leaf scorch]
----------PASS----------

Analyzing 26.png
Expected Result: ['strawberry', 'scorch']
Actual Result: [Strawberry, Leaf Scorch]
----------PASS----------

Analyzing 27.png
Expected Result: ['corn', 'rust']
Actual Result: [Corn, Common Rust]
----------FAIL----------

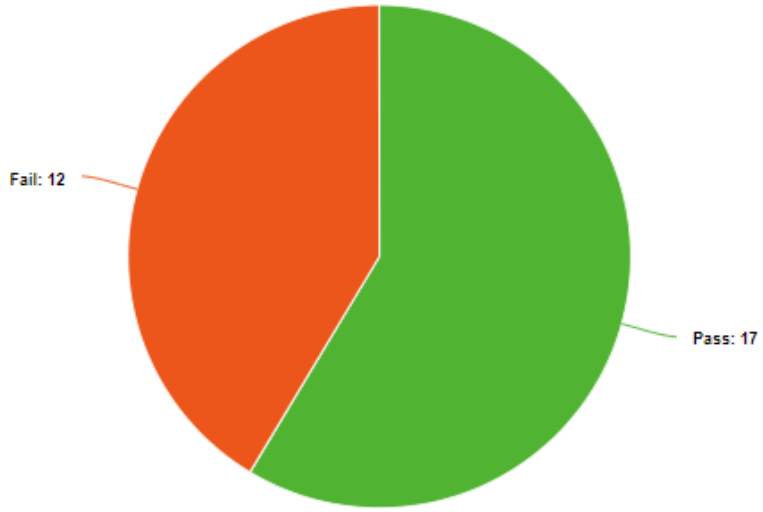
Analyzing 28.png
Expected Result: ['corn', 'rust']
Actual Result: [Corn, Common rust]
----------PASS----------

Analyzing 29.png
Expected Result: ['tomato', 'mold']
Actual Result: [Strawberry, Leaf scorch]
----------FAIL----------

------- Tests Complete -------
Total Test Count: 29
Total Passed Tests: 17

Total Accuracy: 58.62069%

FarmAssistX Results:
17/30 test cases passed
The graph depicting this result is shown below



Software Used:
Utilizing and integrating various AI testing applications: Appium (application script runner), Android Studio, Node.js, NPM
Writing JSON files and python scripts which initiated the execution and evaluated our test cases.

2.1.4 Sick Plant Disease Identifier

./app_scripts/sickPlantDiseaseIdentifier.py
***NOTE:** Script did not work, was unable to integrate it into the testing application

```
import os
import time

from ..app_tester_interface import AppTesterInterface

from appium import webdriver

from appium.options.common.base import AppiumOptions
from appium.webdriver.common.appiumby import AppiumBy

# For W3C actions
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

class PlantDiseaseIdentification(AppTesterInterface):
    def initializeDriver(self):
        base_path = os.getcwd()

        options = AppiumOptions()
        options.load_capabilities({
            "platformName": "Android",
            "appium:platformVersion": "14",
            "appium:appPackage": "com.faisalkabirgalib.sick_plant_disease_identifier",
            "appium:appActivity": "com.faisalkabirgalib.sick_plant_disease_identifier.MainActivity",
            "appium:app": "sick_plant_disease_identifier_1.0.9_apkcombo.com.apk"",
            "appium:deviceName": "emulator-5554",
            "appium:automationName": "UiAutomator2",
            "appium:ensureWebviewsHavePages": True,
            "appium:nativeWebScreenshot": True,
            "appium:newCommandTimeout": 10000,
            "appium:connectHardwareKeyboard": True
        })

        driver = webdriver.Remote("http://127.0.0.1:4723", options=options)
```

```
    return driver

def analyzeImage(self, driver, index):
    wait = WebDriverWait(driver, 30)

    try:
        # Upload Image
        el1 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "Pick Image")))
        el1.click()
        time.sleep(1)

        # File picker More Options
        el2 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "More options")))
        el2.click()
        time.sleep(1)

        # Browse
        el3 = wait.until(EC.element_to_be_clickable((AppiumBy.ID, "com.google.android.providers.media.module:id/title")))
        el3.click()
        time.sleep(1)

        # Google Photos More Options
        el4 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "More options")))
        el4.click()
        time.sleep(1)

        # Sort by
        el5 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
            "//android.widget.TextView[@resource-id=\"com.google.android.documentsui:id/title\" and @text=\"Sort by...\"]")))
        el5.click()
        time.sleep(1)

        # File name (A to Z)
        el6 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
            "//android.widget.CheckedTextView[@resource-id=\"android:id/text1\" and @text=\"File name (A to Z)\"]")))
```

```
        time.sleep(1)

        # File name (A to Z)
        el6 = wait.until(EC.element_to_be_clickable((AppiumBy.XPATH,
            "//android.widget.CheckedTextView[@resource-id=\"android:id/text1\" and @text=\"File name (A to Z)\"]")))
        el6.click()
        time.sleep(1)

        # Handle case of index > 15 (needs scroll)
        # This method only supports up to 30 files
        if index > 15:
            #swipe(startX, startY, endX, endY, duration)
            driver.swipe(530,2115, 525,200, 1000)
            adjusted_index = index - 9
        else:
            adjusted_index = index
        time.sleep(1)

        el7 = wait.until(EC.element_to_be_clickable((By.XPATH,
            f"(//android.widget.ImageView[@resource-id=\"com.google.android.documentsui:id/icon_thumb\"])[{adjusted_index}]")))
        el7.click()
        time.sleep(1)

        el8 = wait.until(EC.element_to_be_clickable((AppiumBy.ACCESSIBILITY_ID, "Run Model")))
        el8.click()

        result_path = "//android.widget.FrameLayout[@resource-id=\"android:id/content\"]/android.widget.FrameLayout/android.view.View/android.view.View/
        model_result_locator = (By.XPATH, result_path)
        new_view_text = wait.until(EC.presence_of_element_located(model_result_locator))
        model_result = new_view_text.get_attribute('content-desc').split('\n')[:-1]
        time.sleep(1)
        return model_result

    except Exception as error:
        print(f'error: {error}')
```

3. Test Complexity & Test Coverage

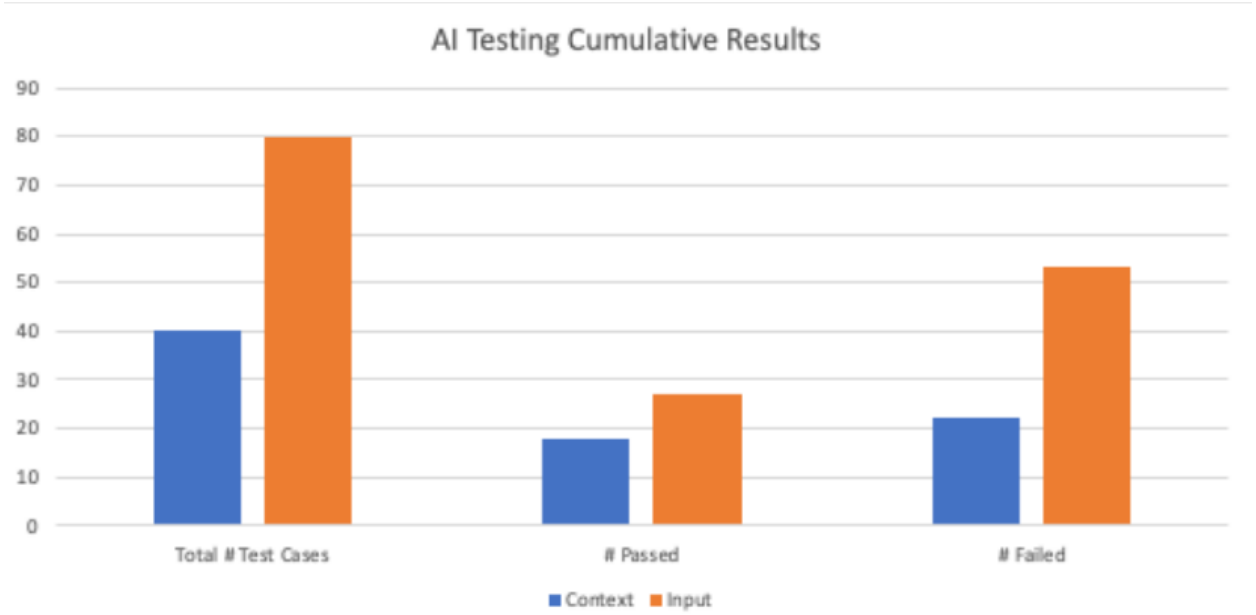
App	Results	Positive Test Coverage	Negative Test Coverage

PlantDisease Identifier	Positive Result: 8 Negative Result: 21	(8/29) * 100 = 27.58%	(21/29) * 100 = 72.41%
DoctorP	Positive Result: 6 Negative Result: 23	(6/29) * 100 = 20.69%	(23/29) * 100 = 79.31%
FarmAssistX	Positive Result: 17 Negative Result: 12	(17/29) * 100 = 58.62%	(12/29) * 100 = 41.3%
Sick Plant Disease Identifier	Positive Result: 0 Negative Result: 0	*Was unable to integrate application into testing application	*Was unable to integrate application into testing application
Total	Positive Results: 31 Negative Result: 56	(31/87) * 100 = 35.63%	(56/87) * 100 = 64.37%

Overall Test Complexity

Test complexity: W*L*H
→ 48 Input rows
→ 48 Output rows
→ 16 Context rows
→ 48*48*16 = **36864**

4. Test Automation Summary



The figure above shows the cumulative results of our AI testing. Overall our testing procedure utilized more input test cases than context test cases. Specifically, for the number of passed test cases, the input test cases performed much better than the context test cases. This relationship is maintained even in the total number of failed test cases (as there were more failed input test cases than context test cases). This relationship is to be expected given that there were twice the amount of input test cases than context test cases.