Test Plan Description

The testing for Planters guide at the start will consist of implementing checks as we are initially coding and creating our database & application. This will be something as simple as including prints to display values in the app or where in the process the application currently is. This would be useful in determining how long communication takes between the Raspberry Pi, our database, and our app. Testing in this way will also help us in figuring out issues with our code early on so we don't spend hours trying to figure out *what* the issue is, and instead spend that time fixing the issues that arise.

Aside from these initial prints to display the values and where in the process we were, an important set of tests we need to do for our automatic watering is boundary tests. Once the moisture sensor ticks below the moisture threshold for a plant, how will our automatic watering react and is it how we want it to react. We also need to make sure that the water we release for the plant is not too much, which is important if we are dealing with a smaller moisture range. Assuming the boundaries work however, testing for the automatic watering will primarily concern whether it works, and how it communicates with our app which brings us to our final set of major tests.

The most important tests we need to conduct, which were mentioned earlier, is testing how long it will take for each of our parts to communicate with one or another. This means that performance tests will be more prevalent in our tests than in some other projects, since the time it takes for everything to communicate is relatively important for our project.

Test Case Descriptions

1. ANDR-01
2. Confirm basic functionality of Android app
3. The user will input their plant type, and the app will fetch that plant's moisture and sunlight thresholds as well as the plant's current moisture and sunlight levels. After this all this information will be displayed to the user.
4. User's plant type
5. User should see their plant's moisture and sun levels, and know if they are in the correct range
6. Normal
7. Blackbox
8. Functional
9. Unit

Results: This test went off without an issue, after selecting the plant species the user is brought back to the home screen and after a few seconds the app screen is updated to reflect the newly selected species and its associated thresholds.

1. ANDR-02
2. Assuring automatic watering can be turned on/off by the user
3. User will turn on automatic watering when moisture is below threshold, and the automatic watering device should receive the signal and water the plant. Likewise, if this is turned off the watering device should not water the plant if the moisture is below the threshold.
4. Automatic watering signal via app
5. The watering device should water if the moisture is below expected and auto-watering is turned on, and not do so when it is turned off.
6. Normal
7. Blackbox
8. Functional
9. Integration

Results: This test worked as expected. To force testing, the moisture sensor would be taken out of the plant to artificially lower the plant's moisture threshold and make the automatic watering occur.

1. ANDR-03
2. Testing communication speed between app and Raspberry Pi
3. Test will both assure the app can communicate with the Raspberry Pi as well as making sure the speed data is transferred between these entities is acceptable
4. User has entered plant type
5. The app should consistently be communicating with the Raspberry Pi to get the moisture and sunlight levels as well as when initially sending the plant type to the Raspberry Pi
6. Normal
7. Whitebox
8. Performance
9. Integration

Results: A partial success, given the time and scope of the project, the communication speed could not be made consistent across different networks, but after a few moments all screens would load as expected. As well, we were unable to implement communication while the app was closed / idle, but the app will update to the latest results from the sensors upon re-opening.

1. SQL-01
2. Ensure database query provides the correct output
3. Test will run queries to pull records that need to be handled from the database. The queries will be ran in SQL Management Studio to see results.
4. SQL query with plant name as parameter

5. Record with the same plant name should be pulled from the database. The record will include the amount of sunlight and water required for the plant.
6. Normal
7. Blackbox
8. Functional
9. Unit

Results: No issues, utilizing PHP and pulling the names listed in the database to comprise the plant list in the app, there is no opportunity for the user to select a plant not currently in the database.

1. SQL-02
2. Ensure database can be run on Raspberry Pi
3. Test will run queries on the Raspberry Pi to ensure that database can be store don the raspberry Pi
4. SQL query with plant name as parameter
5. Record with the same plant name should be pulled from the database. The record will include the amount of sunlight and water required for the plant.
6. Normal
7. Blackbox
8. Functional
9. Integration

Results: Non-successful. Hosting the database on the Raspberry Pi IS possible, however making the app communicate directly with the Pi is not an easy or secure option. Because of this, we opted to host the database remotely using XAMPP, and it acts as a bridge between the two applications.

1. SQL-03
2. Ensure database and app can communicate with one another in a timely manner
3. Test will have user input a plant on the app, and the query will be sent to the database. The database will return the info to the app and the time it takes will be monitored.
4. SQL query with plant name as parameter, input on Android app
5. Database should be received from the app and sent back to the app and the time taken should be reasonable.
6. Normal
7. Whitebox
8. Performance
9. Integration

Results: This test was successful. We can communicate the the SQL database as fast as the hardware can allow us (Raspberry Pi and Android). We opted to do it every 10s or so to not overwhelm the hardware with messages and readings.

1. RASP-01
2. Testing sunlight and moisture sensors
3. Test will have both sensors hooked up to a breadboard and connected to the Raspberry Pi. A string of data will be read out from each sensor to assure that data is being output
4. Light & Water (respectively), sensors
5. Both sensors will function as expected and output data to the console
6. Normal
7. Whitebox
8. Functional
9. Unit

Results: The sensors worked as expected. While not the most high quality sensors were used, the ones we chose got the job done.

1. RASP-02
2. Testing water pump
3. Test will have water pump hooked up to a relay and connected to the Raspberry Pi. Will make sure the Raspberry Pi can properly pump water.
4. Relay, Pump, Water
5. Pump will successfully pump water
6. Normal
7. Whitebox
8. Functional
9. Unit

Results: The pump worked as expected with no issues encountered.

1. RASP-03
2. Testing lost connection to client
3. Test will assure that if connection is lost while sending data between the Raspberry Pi and database/app, that the device will keep retrying until connection has returned.
4. Data being sent between devices
5. Raspberry Pi will continue to retry receiving/sending data
6. Abnormal
7. Whitebox
8. Function
9. Integration

Results: Successful, the Raspberry Pi was given the capability to run locally while not connected to the database, that way once a connection is re-established it will still have data to utilize.

1. AUTO-01
2. Assuring if water reservoir is low, it will let the user know that auto-watering was not fully completed if water runs out while watering
3. Test will have auto-watering start while the water reservoir is low. If the reservoir empties before finishing watering, an alert will be sent to the user letting them know of the issue
4. Reservoir being low, auto-watering on
5. Raspberry Pi will automatically alert the user if reservoir empties while watering
6. Boundary
7. Blackbox
8. Functional
9. Integration

Results: A success, we added a field to a SQL table that denotes if the water reservoir being used is either “full” or “empty”. Any value above empty is denoted as full, but if it is ever empty the device will not water.

Test Case Matrix

Test Name	Test Type	Blackbox/Whitebox	Functional/Performance	Unit/Integration
ANDR-01	NORMAL	Blackbox	Functional	Unit
ANDR-02	NORMAL	Blackbox	Functional	Integration
ANDR-03	NORMAL	Whitebox	Performance	Integration
SQL-01	NORMAL	Blackbox	Functional	Unit
SQL-02	NORMAL	Blackbox	Functional	Integration
SQL-03	NORMAL	Whitebox	Performance	Integration
RASP-01	NORMAL	Whitebox	Functional	Unit
RASP-02	NORMAL	Whitebox	Functional	Unit
RASP-03	ABNORMAL	Whitebox	Functional	Integration
AUTO-01	BOUNDARY	Blackbox	Functional	Integration