

# SYSC 3010

## Test Plan

*Strawberi E-Tank*

---

**Due Date:**

November 19th 2019

**Group:**

M2 (Stawberi E-Bois)

**Written By:**

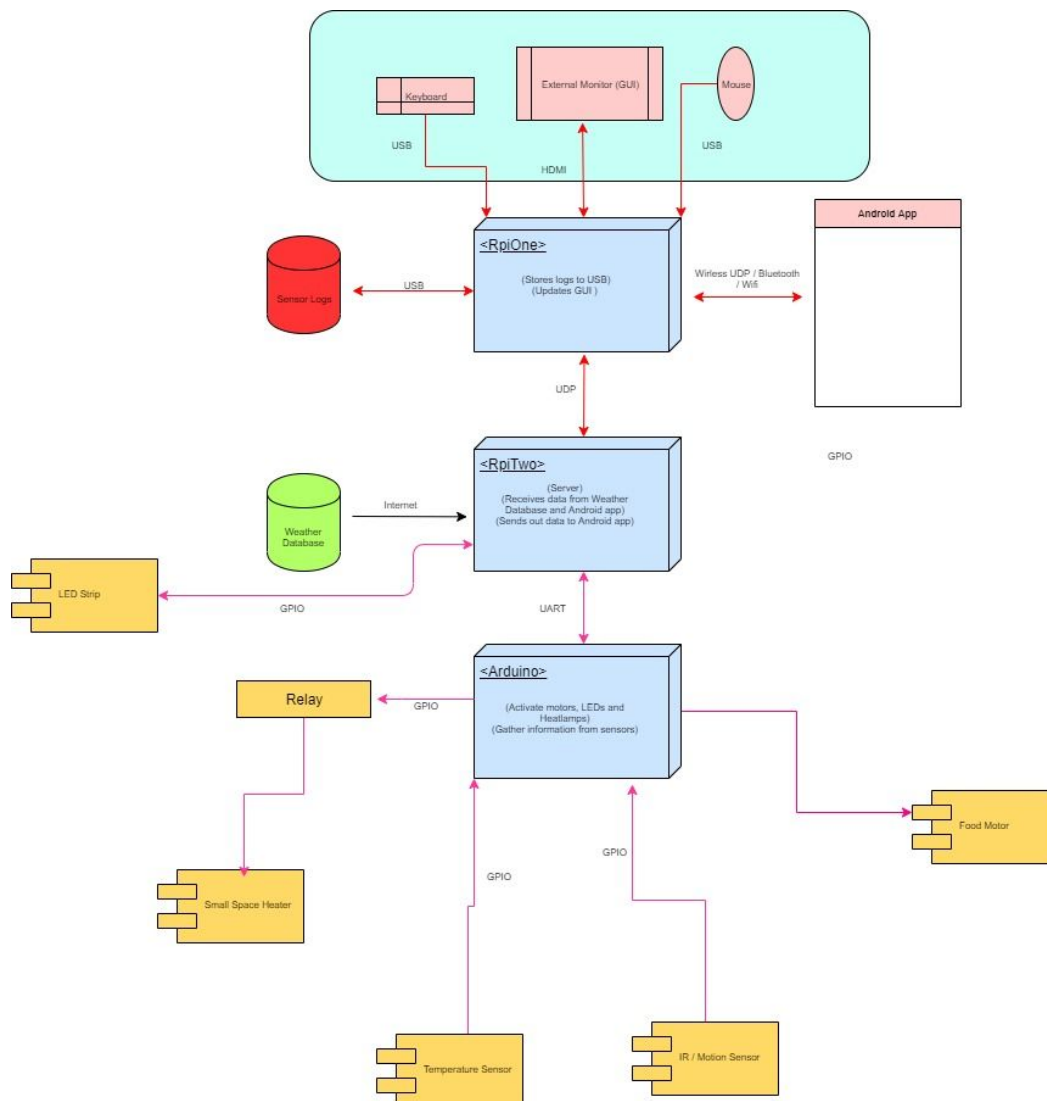
Chase Badalato

Chase Fridgen

Haydn Brown

## Problem Statement:

The objective of this project is to make caring for an animal easier through the use of automation, remote controlled systems, data collection, and visualization. The remote systems allow a user to care for the animals needs at any time and any place by giving it food, water, and changing its habitat temperature. The data collection and visualization allow the user to track habitat conditions and have a record of information if needed. It also allows the system to regulate itself in an event such as overheating. This design has practicality in a commercial setting such as PetSmart. Each animal would have its own E-Tank, with its own set of sensors. The database, GUI, and the Android App have the ability to have multiple E-Tanks connected at once, allowing for an easy way to view any desired E-Tank from one place.



# Distributed Systems Unit Test:

## Purpose / Context:

The database must deal with each subsystem at some point throughout the E-Tank's runtime. This means that it is effectively the central controller of our whole project. It receives and sends UDP packets to all other parts of the system as well as contains the logic regarding what and when to send the packets. This entails that the database receives many different types of information from various subsystems at any given time. Therefore it is vital that the database has minimal bugs and errors so the system will not crash. The database is manipulated through a Python script called `mainDatabase.py` (and ran through `startDatabase.py`). This Python file will be the code that is going to be tested as it is the only way to access the database from within our design. This Python script is located on the Rpi1. The database will always have two tables. `SensVals` and `Tanks`. `SensVals` get saved to the respective tank and `Tank` contains the name, type and location of the pet in the specific tank. The tank table has a foreign key which means that for a sensor value to be added there must already be a tank entry with that same ID. The RED arrows in the deployment diagram indicate the area that the Distributed Systems Test code will deal with.

## Method:

A single python script called `testDatabase.py` will be used to test different cases against the `mainDatabase.py` code. The testing process will be very straightforward. The user will select a test to run, and receive a resulting output (PASS or FAIL). The code will tell if the system failed the test or succeeded. The script will create lists of 10 values, (String, or Numbers) to be randomly selected and tested again the system when applicable. There are 3 mocks to be created. The GUI, Android App, and Arduino. There is no difference between how the GUI and Android app will function, except that they will have different addresses. For testing purposes, because it is on a local Rpi, I will use the same tests for both.

Local (Rpi1): - The GUI communicates with the `mainDatabase` script through UDP

- The script saves information to the database
- Initialize the database

UDP (Rpi2): - The Rpi2 sends data gathered from the Arduino to the Rpi1. The Rpi1 sends data to the Rpi2 to send to the Arduino. Saved and retrieved from database

UDP (App): - The Android app sends and receives to the Rpi1. This data is saved and retrieved from the database

**Local:**

<b>Test #</b>	<b>Test Name</b>	<b>Description (Test if...)</b>	<b>Inputs</b>	<b>Expected Outputs</b>
1	testInitializeDatabase	Database will be successfully initialized	Null	Database has correct tables
2	testAddNewTank	Tank entry is added	Int, Str, Str, Str, Str	New tank entry is added
3	testAddExistingTank	Tank entry attempts to be added but isn't as that tank value already exists (Test primary key)	Int, Str, Str, Str, Str	New tank entry is NOT added
4	testSendSensorValue	Send a sensor value based on the time it recorded (entered by the user)	Str	An appropriate sensor value(s) based on inputted time

**UDP (Rpi2):**

<b>Test #</b>	<b>Test Name</b>	<b>Description (Test if...)</b>	<b>Inputs</b>	<b>Expected Outputs</b>
5	testReceiveSensorValues	Database will receive sensor values and correctly save them into the database	Int, Str, Str, Str, Str Bool	Database added sensor vals to the right tank
6	testSensTargetTemp	Arduino receives target temperature update	Null (Should target newest added SensVal targetTemp)	Target Temp is changed

**UDP (App):**

Test #	Test Name	Description (Test if...)	Inputs	Expected Outputs
2	testAddNewTankApp	Tank entry is added	Int, Str, Str, Str, Str	New tank entry is added
3	testAddExistingTankApp	Tank entry attempts to be added but isn't as that tank value already exists (Test primary key)	Int, Str, Str, Str, Str	New tank entry is NOT added
4	testSendSensorValueApp	Send a sensor value based on the time it recorded (entered by the user)	Str	An appropriate sensor value(s) based on inputted time

**Communication:**

The script emulates sending and receiving data to the mainDatabase.py script from within the same Rpi. For testing all scripts sent and received will be 127.0.0.1 however the mainDatabase.py system is able to handle different addresses. Below are the steps to being the testing process:

- I. Make sure that startDatabase.py, mainDatabase.py and testDatabase.py are in the same folder along with the project database .db file.
- II. Run the startDatabase.py file in the terminal, and this should be all you need to do with this file. If everything is working a message ("Waiting to receive data on port 1025:") should be displayed.
- III. Run the databaseTest.py file. You should be presented with a menu of test options. Type the associated number to run that Test.
- IV. If you select the summary results option it will print out a current list of the tests ran and whether they passed or failed.

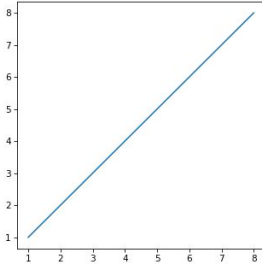
## General Utility Unit Test

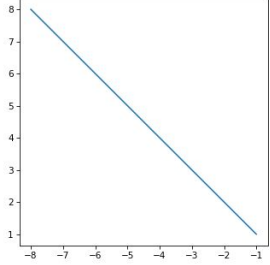
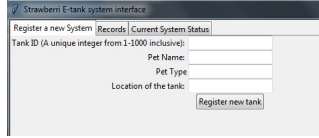
### Purpose:

The purpose of the general utility test is to test the important software functionality of the system, of which the GUI is a large part. The GUI file and its contents must be thoroughly tested as it is responsible for displaying the GUI which provides functionality like registering new tanks and pets, looking up past sensor data, monitoring the live status of a tank, and ensuring the temperature is regulated. While the GUI file has code used to communicate with the main database file, the purpose of these tests is not to test the udp communication, instead it is to test code entirely contained to this file.

### Method:

A python script containing the testing functions will be used in order to test the appropriate functions in both GUI.py and mainDatabase.py. The user will be able to enter a number corresponding to the test they wish to run, and will have the option to run more tests after the first is completed. The test functions will test a variety of cases that the functions should either process normally or handle accordingly to not cause an error. The method with which the functions are tested will vary by function. Some functions may simply return a value and therefore the value can be checked whether it is valid based on the input or not while some functions may draw a graph, which must be manually checked.

Test #	Test Name	Description (Test if...)	Inputs	Expected Outputs
1	testGraphs	Test if graphs are properly drawn given x and y values	<p>1. Int list [1, 2, 3, 4, 5, 6, 7, 8] , int list [1, 2, 3, 4, 5, 6, 7, 8].</p> <p>2. Int list [-1, -2, -3, -4, -5, -6, -7, -8], int list [1, 2, 3, 4, 5, 6, 7, 8]</p> <p>3. Int list [1, 2, 3, 4, 5, 6, 7, 8] , int list [1, 2, 3, 4, 5,</p>	<p>1.</p>  <p>2.</p>

			6]	 <p><b>3.</b> No output, as the lengths of the input lists are not equal</p>
2	testThreading	Test if a thread can be properly created, run, and terminate without interrupting the functionality of the GUI	none	Statements printed to console
3	testCreateWid gets	Create GUI elements of first tab of interface	none	<p>A working GUI as our group would have on the first tab of our system interface. Able to click button and navigate between tabs</p> 
4	testTemperatu reControl	Tests if the function responsible for monitoring/regulating the current temperature of the tank is working properly and can detect temperatures which are too high or too low	<p><b>1.</b> Temperature within acceptable range</p> <p><b>2.</b> Temperature above threshold</p> <p><b>3.</b> Temperature below threshold</p>	<p><b>1.</b> True</p> <p><b>2.</b> False</p> <p><b>3.</b> False</p>

**Communication:**

The tests call the functions in the GUI file as they would be called normally. The tests are executed from a GUI, which is necessary to display the output graphs from the testGraphs test.

**How to perform tests:**

1. Have GUI.py and testGUI.py in the same file, run testGUI.py
2. From the GUI, click the button corresponding to the test you wish to run
3. Any output graphs will be visible on the GUI, output results for other tests will be printed in the console

**Hardware Unit Test:****Purpose / Context:**

The most complex hardware component in our system is the temperature sensor attached to the arduino board. It is the most complex to test as the temperature sensor must read the temperature of the animal enclosure and relay that information back to the Rpi2 so that it can compare the value and adjust the temperature of the enclosure accordingly. The hardware driver component of the testing for the temperature sensor will be manipulated through an arduino script called Menu.ino. Menu.ino will use the Arduinos ports (specifically port 7) to read the temperature in Celsius inside the room. The hardware stub will be implemented to send a set temperature to Rpi2 from the arduino. The stub will also perform other tests like receiving information from the Rpi2 to the arduino, comparing the Rpi2 temperature value to the arduino temperature value, and lastly test if the values are the same. This will happen separate from the hardware. The script will take the set temperature and send the value over UART to the Rpi2, The test function will return true if the temperature value is sent to the Rpi2 and will return false otherwise. The stub will also compare the value sent from the arduino and report back whether true or false if it is the same as the set value on the Rpi2. it will do the same to compare and receive values sent back and forth.



**Method:**

A arduino script called tempStub.ino and a python script called testStub.py will be used to test the stub portion of the hardware testing. testStub.py will send values through UART to perform the requested test functions in tempStub.ino and output if the test function passes or fails. tempStub.ino will send and receive a set temperature to the RPi2 through UART and return true if the temperature was successfully sent and received and false otherwise. tempStub.ino will also test to see if the temperature sent is the same as the set temp value in the Rpi2 (set in testStub.py) and return true if it is and false otherwise. It will also compare these two values and say whether its larger or smaller so the arduino will know to heat or cool the enclosure. The Other arduino script being used is called Menu.ino and this will run a menu interface where the user can press a button and the attached temperature sensor will output two real time values for the temperature in the guven room in a 10 second interval. Afterwards, the user can stop the menu interface or press the button again and read the temperature once more.

**Local:**

Test #	Test Name	Description (Test if...)	Inputs	Expected Outputs
0001	getTemp	the arduino properly gets uses the temperature sensor	Null	The temperature of the room i.e) 25 C 23 C
0010	testSendTemp	The temperature is sent to Rpi2	char	True if sent. False otherwise
0011	testRecieveTemp	The arduino receives the temperature from the Rpi2	char	True if received. False otherwise
0100	testTempisSame	The temperature sent to from the Arduino is the same as the set one in Rpi2	char	True if sent. False otherwise
0101	testCompareTemp	The temperature sent from the arduino is greater or less than the set value in the Rpi2	char	Greater if arduino temp val is greater. Less otherwise

### Communication:

testStub.py will send a signal to the arduino based on user input and run the specified test with tempStub.ino and output whether the test has passed or not based on the functions in tempStub.ino.

How to perform tests:

1. Upload tempStub.ino onto the arduino board
2. Open testStub.py on Rpi2
3. Connect the Rpi2 and arduino board through UART connection to the serial port on the arduino and the usb port on the RPI2
4. Select what test you would like to see run and see whether it passed or failed

### Class Diagram for Stub classes and Driver component:

