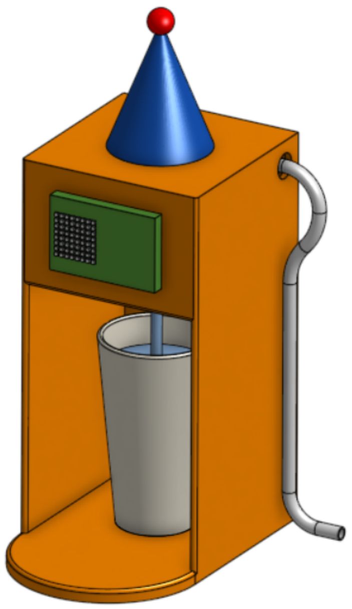
SYSC3010

Computer Systems Development Project

The Water Buddy

Detailed Design Document



Group L3-G7

Caleb Turcotte, 100929209
Michael Marsland, 101042414
Nicholas Milani, 101075096

TA: Roger Selzler

March 13th, 2022

Table of Contents

1.0 Problem Statement

The purpose of this project is to design and build a desk-based smart hydration system for the SYSC 3010 Computer Systems Development Project. Our goal is to help users stay on top of drinking water regularly and meeting the daily recommended intake of water, which is an important factor for overall health. By creating a device that tracks a user's water intake and automatically provides them with water at their desk when they need it, we can help remote workers or anyone who spends most of their day at their desk stay hydrated and healthy.

1.1 Functional Requirements

The following is a list of functional requirements for the Water Buddy System. The end product may meet more requirements than are listed here but these are the minimum functional requirements that will be met.

1. They system will calculate time intervals at which the user should be reminded to drink water based on user-inputted metrics and preferences
2. When a reminder notification is triggered, the sense hat will display a message and the buzzer will sound
3. When a cup is inserted into the Water Buddy Station, the Water Buddy Station will transfer water from a water jug using a pump to refill the cup to the user preset amount
4. The station will record when a cup is filled and save the time and amount in the database
5. A user can see the amount of water they have consumed over time as a graph in the application
6. A user can modify their information (Height, Weight, Thirst) from the application
7. A user can modify their station's settings (cupSize, mute, displayNotifcationsFromFriends) from the application
8. When the user modifies their settings in the Smartphone Application those changes are recorded and used on the Water Buddy Station
9. The user will receive an app-notification when they pass their daily and weekly water intake thresholds
10. The Water Buddy Station will maintain a local copy of the station's settings, the user's information, and a record of refills to allow the Water Buddy Station to function if it is offline.

2.0 Design Overview

2.1 System Overview Diagram

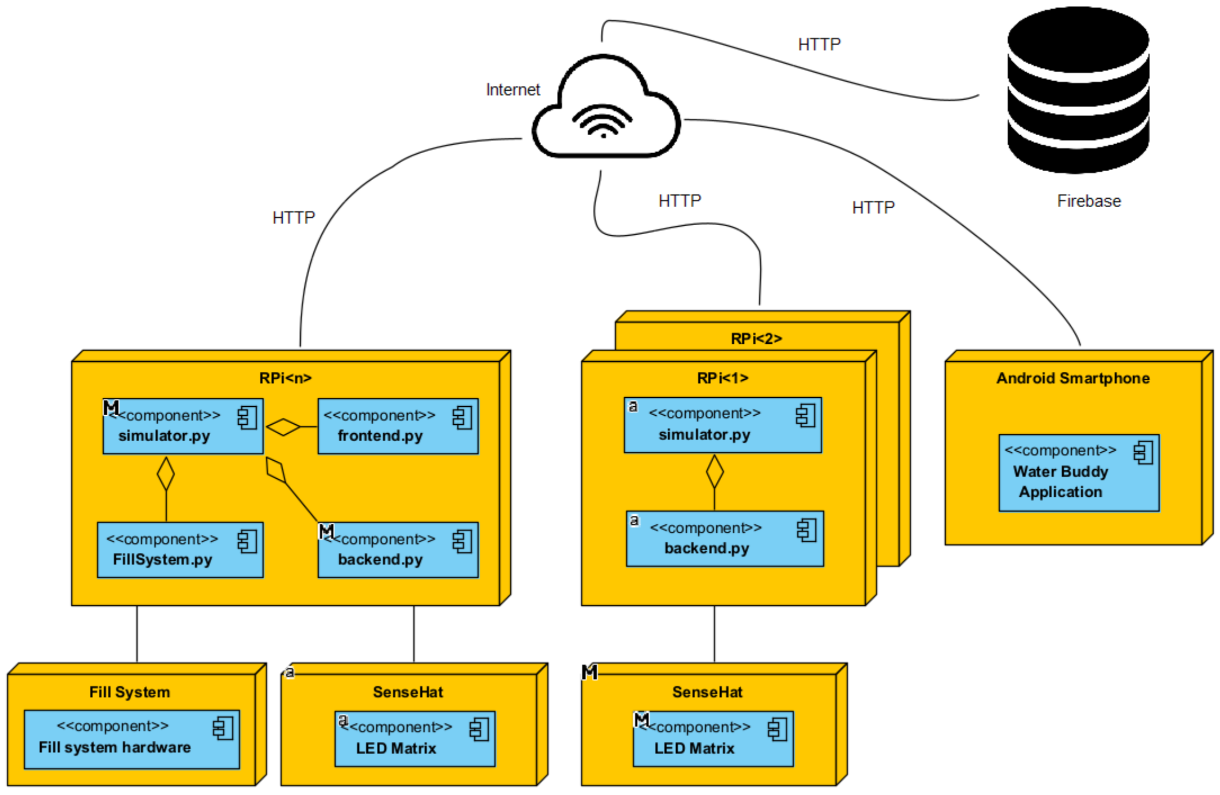


Figure 1: High Level UML Deployment Diagram of the Water Buddy System

Figure 1 above shows the high level UML Deployment Diagram of the system. For the scope of the project our system will contain only one functional Water Buddy with all the associated hardware. This was necessary to meet the budget requirements of the project. However, to showcase the entire functionality of the system and its scalability we will be utilizing our other Raspberry Pi's as Water Buddy simulators capable of displaying on their sense hat as a normal Water Buddy and receiving simulated input through the senseHat's joystick. The Water Buddy Station and the simulators comprise the first major component of our system.

The Water Buddy Station (Discussed further in Section 4, see Figure 12 (water buddy 3D mock up) will be a desk mounted water refilling station that is able to refill a cup with water drawn from a jug positioned under the desk. The Station will also display information and reminders to the user to encourage them to drink water throughout their workday.

The second component of the Water Buddy system is the Water Buddy Application. The Water Buddy Application will run on any android smartphone and be able to display data from a database about the users water intake as well as adjust the users settings and send messages to other users.

The third component of the system is the Firebase Database. We decided to use a firebase for our database since it was a database we have some experience with and it interfaces well with both Python and Android applications. The firebase database stores all the data for the system as well as allows a channel for the passing of messages between devices and the smartphone application.

On the whole the system is scalable for any number of users and Water Buddy devices provided we can scale our firebase database to keep up with demand. For a more detailed view of the Water Buddy system see Figure 2 in Section 2.2.

2.2 Communication Protocols

The system's hardware components (except for the water pump and its power supply) all communicate with the RPi through GPIO pins. The SenseHat uses the I^{2}C serial protocol to communicate sensor and control data to and from the RPi. The ultrasonic sensor uses two GPIO pins to communicate with the RPi: the *Trigger* pin is an input to the sensor that tells the sensor when to transmit an ultrasonic pulse, and the *Echo* pin is an output from the sensor that is driven high for the duration of time that it takes for the reflection to be received by the sensor. The water flow sensor uses one GPIO pin as an output from the sensor that creates a square wave signal with a frequency that represents the flow rate. The piezo buzzer uses one GPIO pin in pulse-width modulation (PWM) mode as an input to the buzzer. This allows the RPi to control the buzzer with a square wave signal and modify the buzzer's tone by changing the frequency of the signal. Finally, the water pump relay uses one GPIO pin as an input to the relay, which closes the water pump circuit when the signal is high, and opens it when the signal is low. The Detailed Deployment Diagram of the main Water Buddy components shown below (Figure 2) showcases how the various components communicate with the Station's Raspberry Pi.

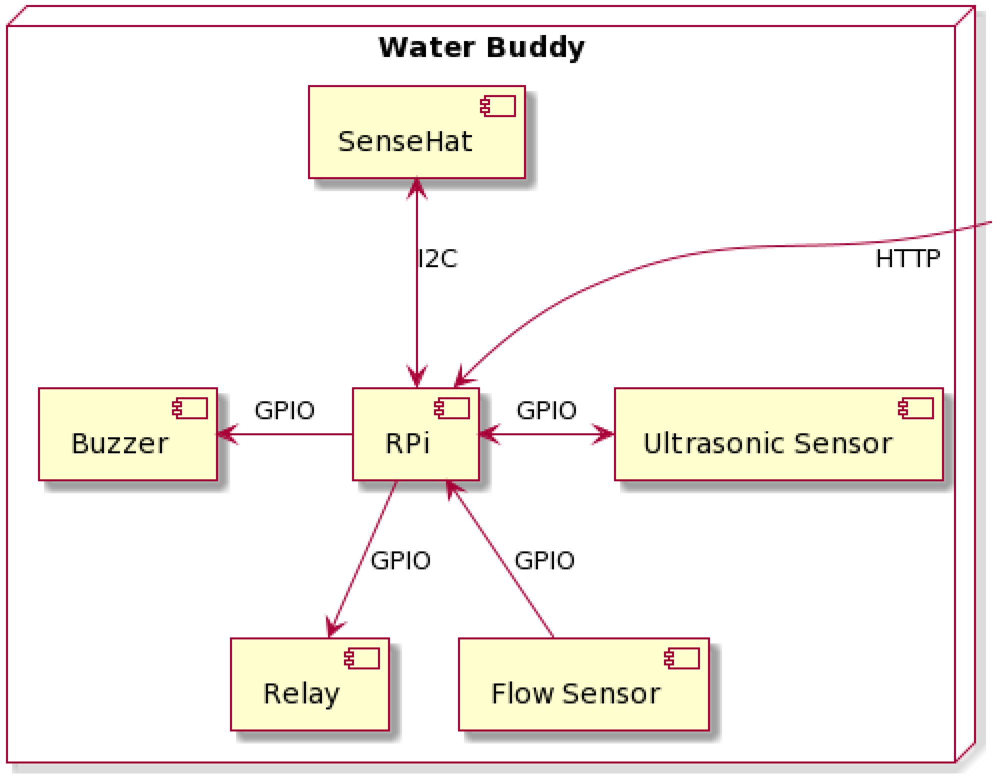


Figure 2: Detailed Deployment Diagram of the Water Buddy Station

For IoT communication between the major nodes of our system, we will be using Firebase. Firebase will allow us to upload information into the database to later be read by other nodes. For message passing we have defined a message area of the firebase database where messages can be added. Messages will be read asynchronously by the node it is intended for and then deleted from the database by the receiving node. Each message will have the following structure {“src”: STRING, “dest”: STRING, “message”: STRING, [“extras”: JSONOBJECT]}, where “extras” is an optional JSON object that can that can contain optional fields for the messages such as “color”, “priority”, etc…. This allows the message passing format to be expanded on as the project continues. More details about message passing and the structure of communication with the database can be determined by looking at the database schema defined in Section 2.4.1.

2.2.1 Communication Protocol Table

The following tables highlight how messages are passed from the given nodes of the system:

Table 1: Communication Protocols for the Water Buddy Station Software

Water Buddy Station Software Communication Protocol Table			
Sender	Receiver	Message Type	Description
WaterBuddy	WaterBuddy	Message	The Water Buddy pushes a message to the Firebase Database message child. The format of these messages is: {“src”: STRING, “dest”: STRING, “message”: STRING, [“extras”: JSONOBJECT]}

WaterBuddy	Smartphone Application	Station Initialization and updates	The database can be updated by pyrebase on any child node. The updates take the form of JSON objects. For example, registering a new station to the database the following pyrebase command would be used.
			`self.updateDatabase(`
			`f"stations/{stationID}",`
			`{“stationID”: stationID,`
			`"cupSize": 25,`
			`"humidity": 25,`
			`"mute": False,`
			`"waterFrequency": 0})`
			The data types and units passed with these JSON objects are defined in Section 2.4.1, Database Schema

Smartphone Application	WaterBuddy	UserData	For user data which the station uses to evaluate how often to dispense water to the user a python dataclass has been created, this is pulled from the database every cycle and the local data is updated. These are pulled from firebase as JSON Objects and then parsed into the following data structure:
			@dataclass
			class UserData:
			    id: str = "John Doe"
			    height: float = 0.0
			    weight: float = 0.0
			    thirst: str = "less"

Smartphone Application	WaterBuddy	StationData	For station data where the unit receives its settings from a python dataclass has been created, this is pulled from the database every

			cycle and the local data is updated. These are pulled from firebase as JSON Objects and then parsed into the following data structure: @dataclass class StationData: cupSize: float = 355 mute: bool = False waterFrequency: float = 3600 displayNotificationsFromFriends: bool = False

Table 2: Communication Protocols for the Smartphone Application

Smartphone Application Communication Protocol Table			
Sender	Receiver	Message Type	Description
Smartphone	Firebase Database	User Message	Java Object sent to Firebase Database, contains source, destination, and message string for Water Buddy Station to print out
Smartphone	Firebase Database	Station Message	Java Object sent to Firebase Database, contains Water Buddy Station information as detailed in Section 3.3

Table 3: Communication Protocols for the Water Buddy Station Hardware

Hardware Communication Protocol Table			
Sender	Receiver	Component	Description
RPi	Ultrasonic Sensor	Ultrasonic Sensor (TRIGGER)	The RPi sends a *Trigger* signal to the ultrasonic sensor's *Trigger* pin, prompting it to generate an ultrasonic pulse
Ultrasonic Sensor	RPi	Ultrasonic Sensor (ECHO)	The ultrasonic sensor drives the *ECHO* signal high until it receives the reflection of the ultrasonic pulse
Flow Sensor	RPi	Flow Sensor	Uses one GPIO pin as an output from

			the sensor that creates a square wave signal with a frequency that represents the flow rate
RPi	Water Pump Relay	Water Pump Relay	Uses one GPIO pin as an input to the relay, which closes the water pump circuit when the signal is high, and opens it when the signal is low
RPi	Piezo Buzzer	Piezo Buzzer	Uses one GPIO pin in pulse-width modulation (PWM) mode as an input to the buzzer
RPi	SenseHat	SenseHat Display	Uses the I^2C serial protocol to set the LEDs on the SenseHat display from the RPi
SenseHat	RPi	SenseHat Humidity Sensor	Uses the I^2C serial protocol to send humidity sensor data to the RPi

2.3 Message Sequence Diagram(s)

The following Message Sequence Diagrams comprise a list of the main communication methods between nodes of the server. From the specific shown sequence diagrams you can infer how other similar communications are performed.

2.3.1 Message Sequence Diagram 1: Communicating with the Ultrasonic Sensor

The following figure (Figure 3) depicts how the RPi controls the ultrasonic sensor. The RPi sends a *Trigger* signal to the ultrasonic sensor, which then drives it's *Echo* output high, transmits an ultrasonic pulse, and then drives the *Echo* output low when the pulse reflection is received

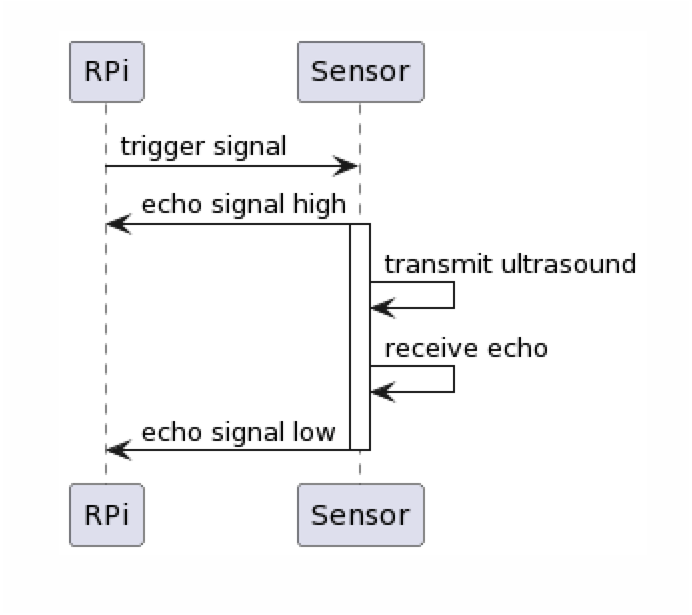


Figure 3: Communicating with the Ultrasonic Sensor

2.3.2 Message Sequence Diagram 2: Controlling the Water Pump

The following sequence diagram (Figure 4) depicts how the RPi controls the water pump through the hardware relay. To turn on the pump, the RPi sends a high signal to the relay through a GPIO pin, which closes the circuit between the water pump and its power supply. To turn off the pump, the RPi drives the GPIO pin low, which opens the relay and disconnects the pump from the power supply.

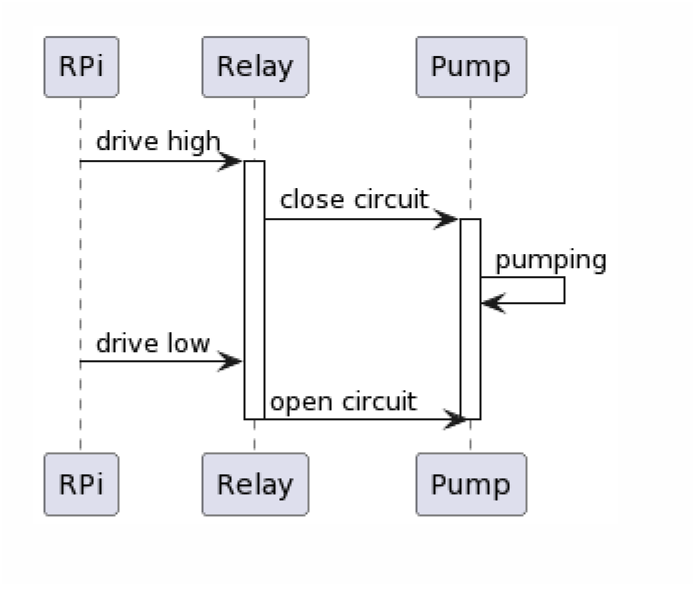


Figure 4: Controlling the Water Pump

2.3.3 Message Sequence Diagram 3: Smartphone Application Message Passing

The following sequence diagram (Figure 5) demonstrates the scenario where a User creates a message on their smartphone application which is then pushed asynchronously onto the firebase database and kept there until it is read by the intended Water Buddy station.

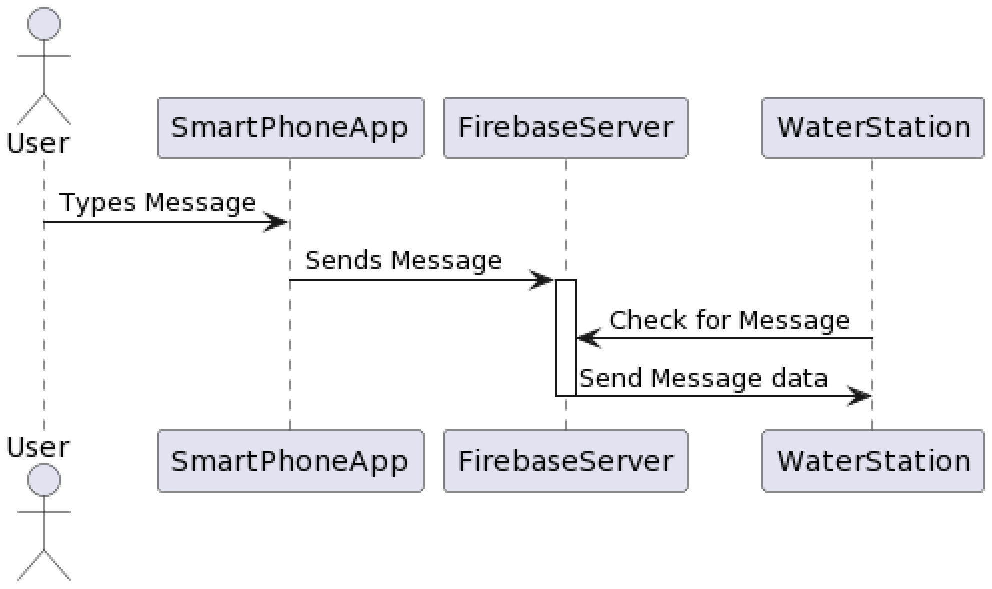


Figure 5: Message Sequence Diagram 2: Smartphone Application Message Passing

2.3.4 Message Sequence Diagram 4: Water Buddy Station Updating Firebase

The Water Buddy Station will be often updating the database with new information as well as messages. All this communication takes a very similar form thanks to the pyrebase library. The following sequence diagram (Figure 6) demonstrates the case where the Station updates it's humidity to the firebase database and the Smartphone Application reads this humidity value.

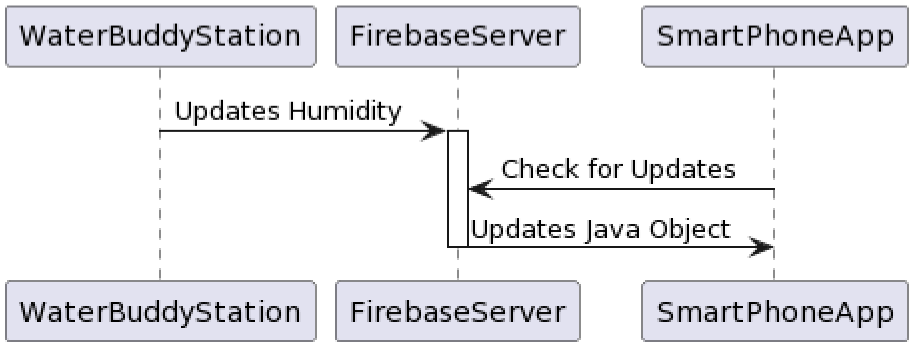
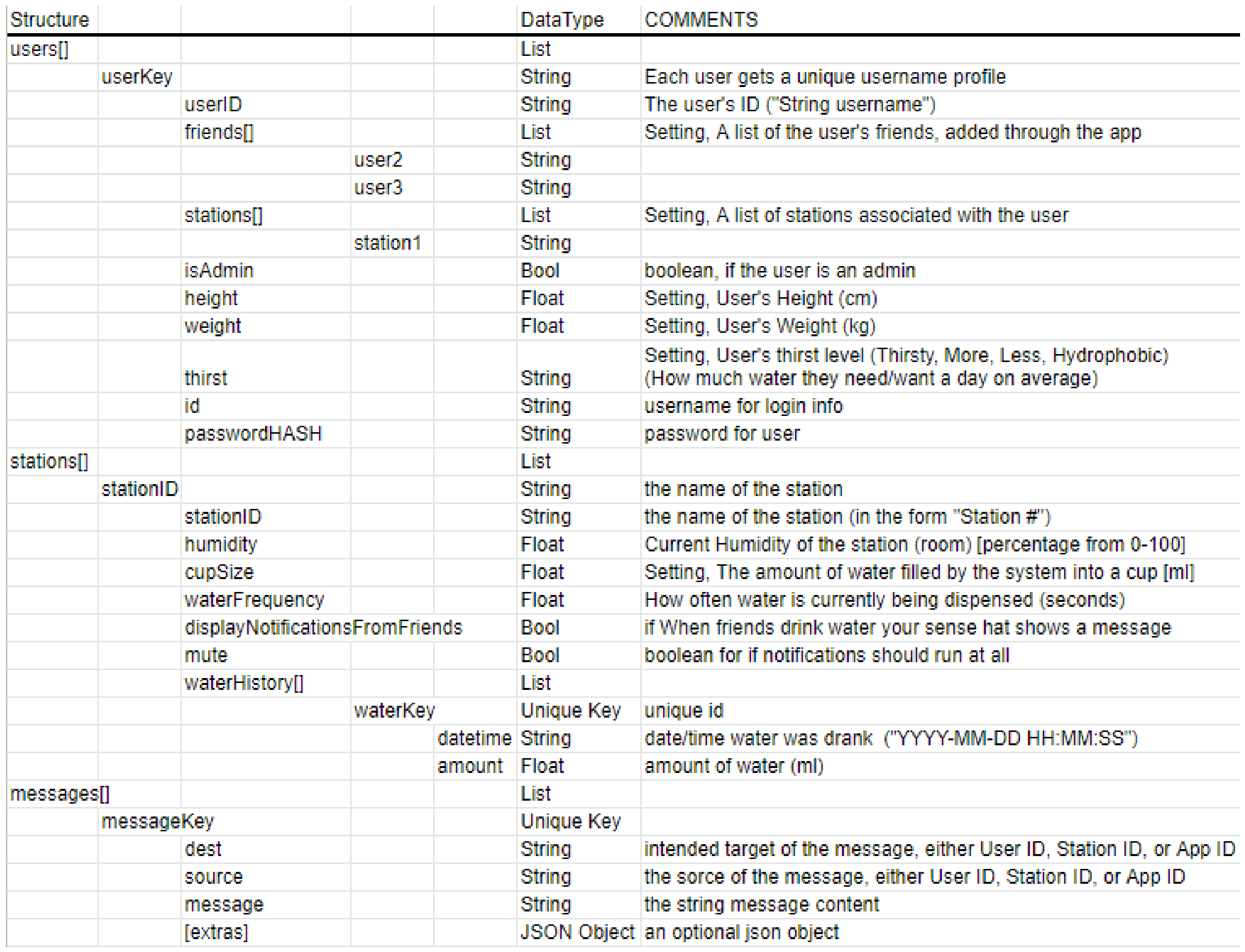


Figure 6: Message Sequence Diagram 4: Water Buddy Station Updating Firebase

2.4 Database Table Design/Schema

2.4.1 Firebase Database

A detailed representation of the Database Schema is shown in Figure 7 below. This schema has 3 main tables, user information, station information, and messages. This is done to keep information separate and ordered.



Structure					DataType	COMMENTS
users[]					List	
	userKey				String	Each user gets a unique username profile
		userID			String	The user's ID ("String username")
		friends[]			List	Setting, A list of the user's friends, added through the app
			user2		String	
			user3		String	
		stations[]			List	Setting, A list of stations associated with the user
			station1		String	
		isAdmin			Bool	boolean, if the user is an admin
		height			Float	Setting, User's Height (cm)
		weight			Float	Setting, User's Weight (kg)
		thirst			String	Setting, User's thirst level (Thirsty, More, Less, Hydrophobic) (How much water they need/want a day on average)
		id			String	username for login info
		passwordHASH			String	password for user
stations[]					List	
	stationID				String	the name of the station
		stationID			String	the name of the station (in the form "Station #")
		humidity			Float	Current Humidity of the station (room) [percentage from 0-100]
		cupSize			Float	Setting, The amount of water filled by the system into a cup [ml]
		waterFrequency			Float	How often water is currently being dispensed (seconds)
		displayNotificationsFromFriends			Bool	if When friends drink water your sense hat shows a message
		mute			Bool	boolean for if notifications should run at all
		waterHistory[]			List	
			waterKey		Unique Key	unique id
				datetime	String	date/time water was drank ("YYYY-MM-DD HH:MM:SS")
				amount	Float	amount of water (ml)
messages[]					List	
	messageKey				Unique Key	
		dest			String	intended target of the message, either User ID, Station ID, or App ID
		source			String	the sorce of the message, either User ID, Station ID, or App ID
		message			String	the string message content
		[extras]			JSON Object	an optional json object

Figure 7: Database Schema

2.4.2 Local Database

Each Water Buddy Station will also maintain a local SQL database with the station's settings, the user's information, and a record of refills. This will allow Water Buddy Station to function if it is offline and ensure the station's settings are saved in case it is unplugged. The local database will contain 3 tables: The "Settings" which will only contain one row, the user's current settings; The "UserInfo" table which will also only contain one row, the User's Information; and the "WaterHistory" table which will contain an entry for every time the user refills a glass of water. The settings and userInfo table will be pulled from the database whenever possible and the WaterHistory will push to the database and delete it's own entries whenever possible.

Table 4: SQL Schema for the Station Settings Table (“Settings”)

Column	Data Type	Description
mute	Boolean	If the station should display notifications or not.
cupSize	float	The amount of water to refill (mL)

Table 5: SQL Schema for the User Information Table (“UserInfo”)

Column	Data Type	Description
userId	Boolean	If the station should display notifications or not.
height	float	The height of the user (cm)
weight	float	The weight of the user (kg)
thirst	String	How must water the user wants to take in, one of the enum “Thirsty, More, Less, Hydrophobic”

Table 6: SQL Schema for the Water History Table (“WaterHistory”)

Column	Data Type	Description
dateTime	String ("YYYY-MM-DD HH:MM:SS")	The date and time that the water was dispensed
amount	float	The amount of water dispensed (mL)

3.0 Software Design

3.1 Software Design for Node 1: Water Buddy Station

The software running on the Water Buddy Station will be written in Python. This software will run on the Raspberry Pi and be responsible for controlling the fill system, displaying information and notifications on the SenseHat, alerting the user with the buzzer, storing its own information in a local database and communicating with the firebase database. The software is written using Object Oriented structure to break up pieces of the code and allow for modularity. Figure 8 below shows a basic UML class diagram highlighting how the classes work together to create the whole software.

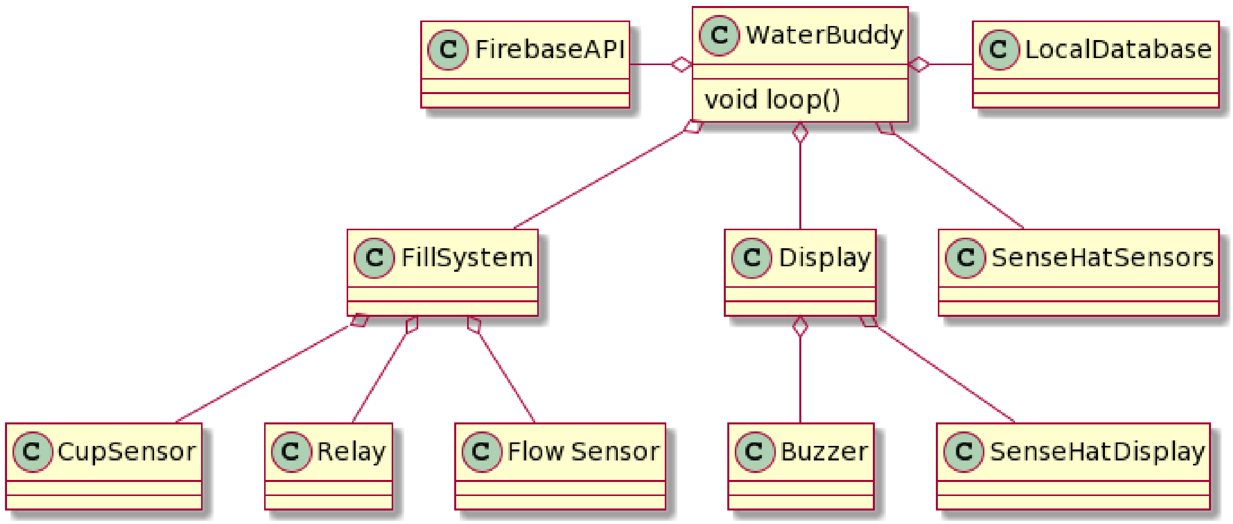


Figure 8: Basic Class Diagram for the Water Buddy Station

On Startup the WaterBuddy class will be initialized through a file called “waterbuddy.py” that is run when the Raspberry Pi is booted. Once the WaterBuddy is initialized it will connect to the firebase database through pyrebase and ensure there exists an entry for itself in the database. If not it will create one with default values. It will also ensure that it is maintaining a local database for it’s settings. If it is not then it will create a local database through it’s Local Database class and fill it with default values.

After these basic initialization checks are complete the WaterBuddy will begin it’s loop function which constantly runs through the following steps: uploading the current humidity to the firebase database, check for updates to the firebase database regarding the stations settings or user information, update the local database if required by a change to the settings, update the firebase database with new waterFrequency if required by new settings, tell the local database to push any updates of the water history to the firebase database, check the firebase database for messages directed at the station and display messages, dispatch a local notification to drink water if the calculated time between drinks has passed, and finally, poll the ultrasonic sensor to check if the fill system needs to be run.

Both the Display and FillSystem classes will be run on their own threads to ensure that both the fill system can function when long messages are being displayed and messages can be displayed while the fill system is running. If the station is ever disconnected from the internet, all the functions not involving the firebase database will still run, this includes both the fill system and the local notifications. Any fills that take place will be recorded in the local database and pushed to the firebase database by the mainloop when connection to firebase is reestablished. In the case where the WaterBuddy is unplugged and plugged back in, it will startup and run the system again. The settings for the station will be pulled from the local database and used until connection to the firebase database is restored.

3.2 Software Design for Node 2: Water Buddy Simulators

The software for the Water Buddy Simulators will be mostly the same as the software for the Station itself aside from a few classes mainly pertaining to the fill system that will be replaced with Simulator classes. These classes will, instead of controlling the fill system, simply display what they "would be doing" on the senseHat display so we can ensure they are interacting correctly. Mostly the simulators will be used for seeing notifications from other stations.

The simulator will also make use of the SenseHat's joystick to emulate a water glass triggering the non-existent ultrasonic sensor. This will allow us to emulate the behaviour of a full station without requiring all the supporting hardware which was out of the budget for this project. In the full scope of a production ready Water Buddy System, Water Buddy Simulators will not exist and instead there will be multiple actual Water Buddy Stations comprising the system. Figure 9 and Figure 10 below show basic UML class diagrams of the classes that must be simulated for the simulator to function as the regular station. By using interfaces all of the other code for the regular Water Buddy Station and the Simulators can be the same aside from deciding if the Water Buddy should use the real classes or the simulated classes. The interfaces will be implemented in python using simple duck typing but could be upgraded to use Python's Abstract Base Class functionality.

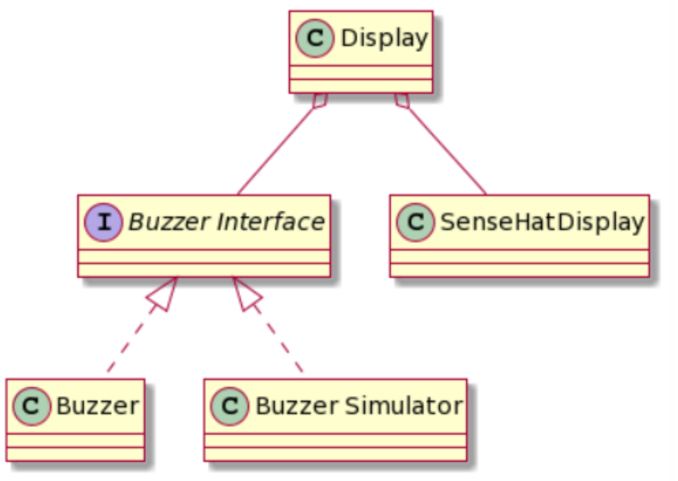


Figure 9: Required Simulated class "BuzzerSim" for the Display

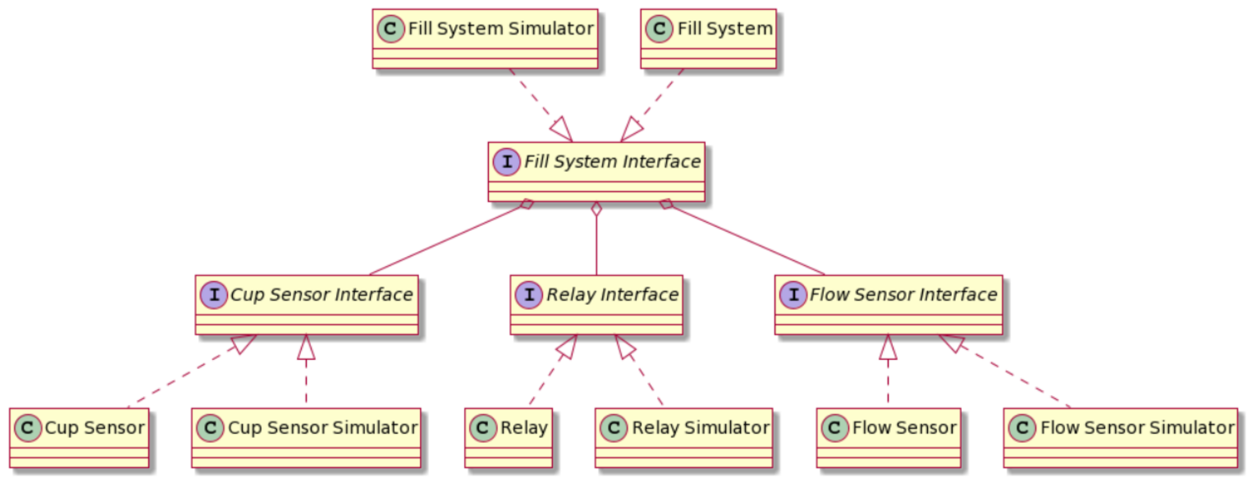


Figure 10: Required Simulated classes for the Fill System

3.3 Software Design for Node 3: Smartphone Application

The third node of the system is the smartphone application. This application is written in Java. Where data snapshots of the firebase database are represented by User and Station classes. And a MainActivity class is used to handle the functionality of the application itself. A class diagram of this application is shown below in Figure 11.

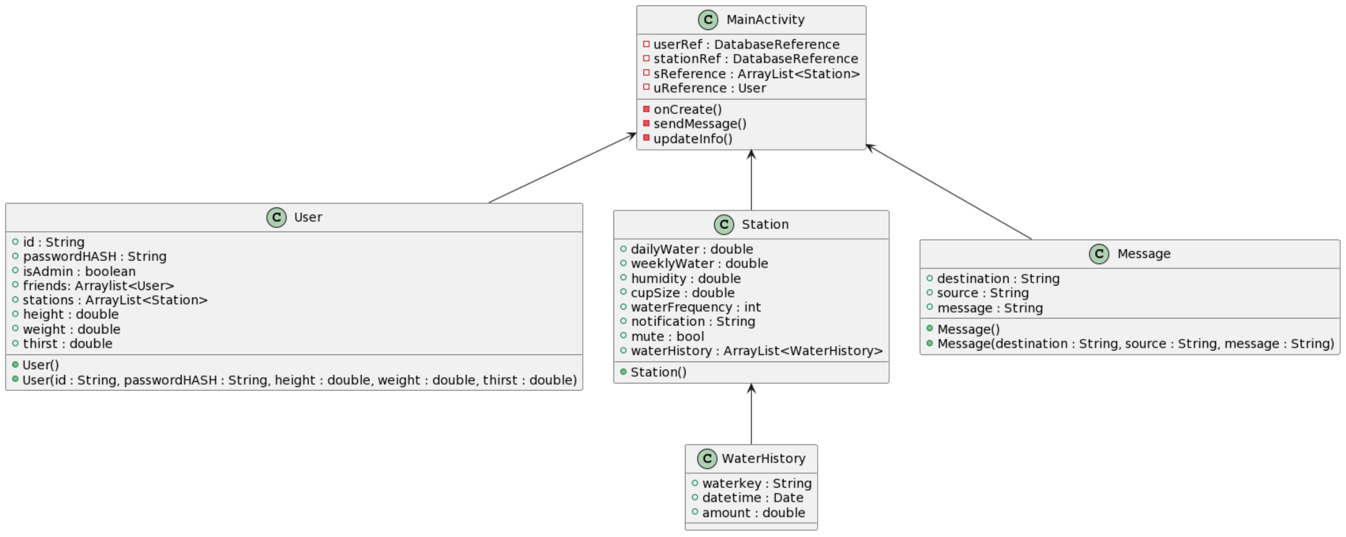


Figure 11: Smartphone Application UML Diagram

The Java application saves information from the Firebase Database as Java Objects. A User object is used to provide login information for the app as well as settings for the user that Water Buddy stations can use. A Station object contains all settings and information read from a Water Buddy station.

Once logged in a User can register Water Buddy stations to their account, edit settings for those stations such as the size of their cup in milliliters, and even add friends,

This functionality is detailed further in Section 5, GUI Design.

4.0 Hardware Design

The Water Buddy Station is a 3D-printed desk mounted unit for refilling a water glass and interacting with the user while they are working at their desk. Figure 12 below shows a 3D model mock-up of the Water Buddy Station (with party hat included). The following sections describe the specific hardware components that allow the Water Buddy Station to refill the user's glass upon request.

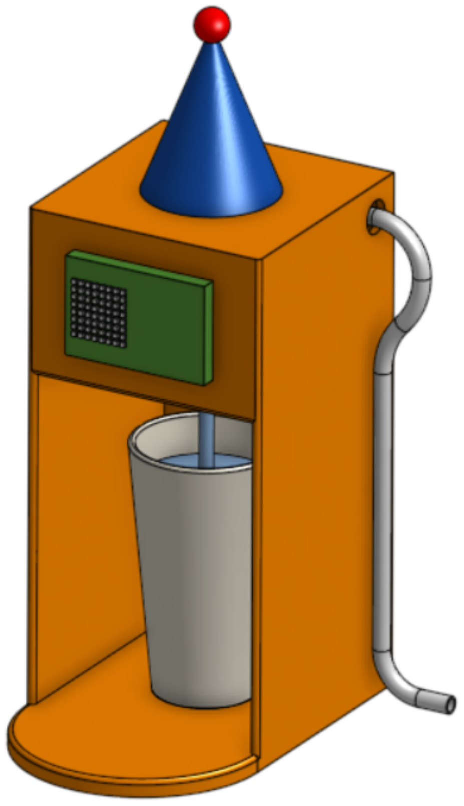


Figure 12: A 3D modelled mock-up of the Water Buddy Station

4.1 RCWL-1601 Ultrasonic Distance Sensor

This ultrasonic sensor was chosen as a means of detecting whether a cup is present in the system. It has a working voltage range of 3V to 5.5V, where the *Trigger* input and *Echo* output levels match the supplied voltage. This allows for direct communication with the RPi's 3.3V GPIO and power pins without the need for voltage division. The sensor's maximum current draw is 2.2mA, which is safely within the RPi's suppliable current range.

4.2 YF-B1 Water Flow Sensor

This water flow sensor was chosen in order to accurately measure the amount of water being dispensed by the system. It has a minimum working voltage of 4.5V, and a maximum working current of 15mA, which means that it can safely be powered by the RPi's 5V power pins. The sensor outputs a square wave with an amplitude of 5V, which means that a voltage divider must be used in order to drop the signal down to a voltage that is safe for the RPi's GPIO pins. The frequency of this square wave represents the flow rate using the following function from the datasheet: *F=11*Q,* where Q is the flow rate in L/min.

The schematic diagram below shows how the Flow Sensor is connected to the RPi using a voltage division circuit comprised of a 1kΩ resistor and a 2kΩ resistor:

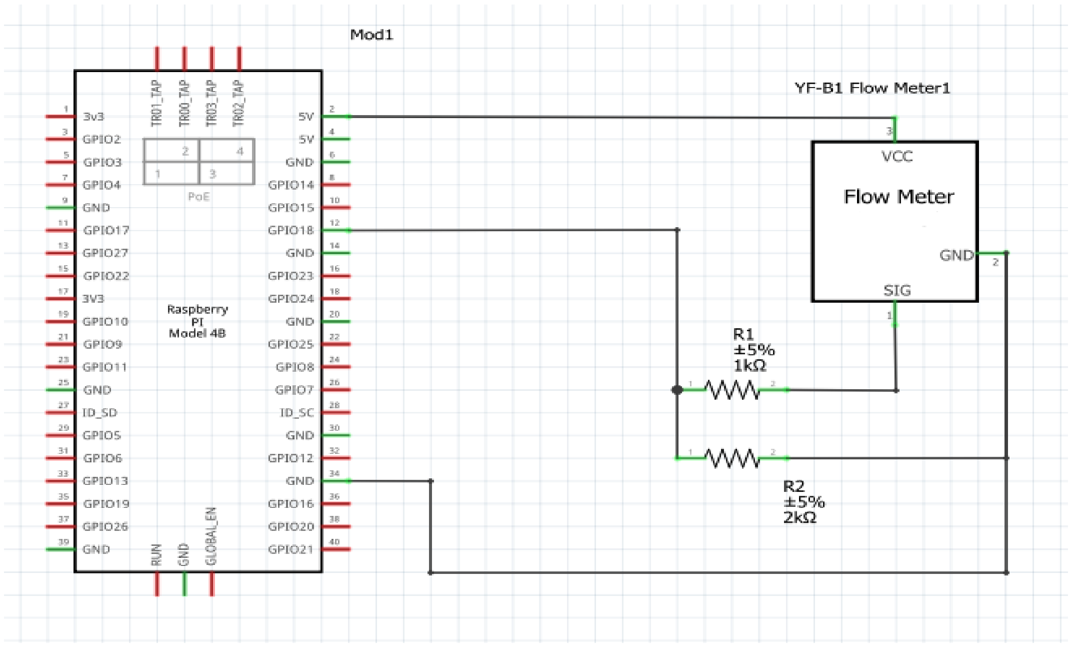


Figure 13: Flow sensor schematic diagram

4.3 Submersible DC Water Pump

In order to fill the user's cup with water, we are using a small submersible DC water pump. This pump has a working voltage range of 6V to 12V and a maximum working current of 500mA, which means that it must be powered by an external power supply, which will be connected to a relay which will be controlled by the Rpi.

4.4 Relay Module

This 5V relay module will allow the RPi to turn the water pump on and off. This relay requires a 5V power source and has a maximum current draw of 72mA, which means it can be safely powered through the RPi's 5V power pins. The relay's control input can be controlled directly by a 3.3V signal from the RPi's GPIO pins. Finally, the relay module has a maximum switching power of 240W (24VDC @ 10A) which is more than enough to handle the 12V 1.5A power supply.

4.5 12V DC Power Supply

This external power supply will be used to power the submersible water pump. It has a rated current of 1.5A at 12V and simply plugs into any AC wall outlet.

The schematic diagram below shows how the water pump, 12V power supply, relay module, and RPi are interconnected in order to control the water pump (Note: for simplicity's sake, the relay module pictured below is a 5-pin relay, whereas the relay module used in our design is a 6-pin relay that is connected to the RPi's 5V power rails in addition to the GPIO26 pin for the control signal):

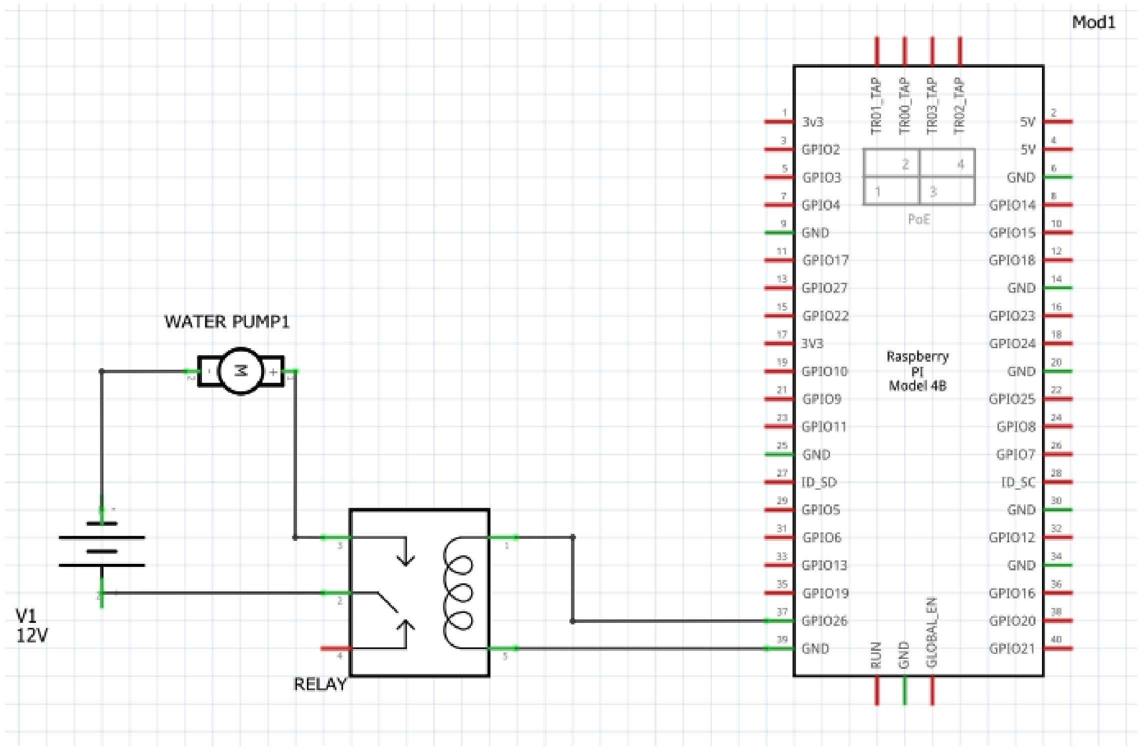


Figure 14: Water pump, power supply & relay setup

4.6 Piezo Buzzer

The piezo buzzer is used as a way to provide audible notifications to users for various system events. It has a minimum working voltage of 3V and a maximum current draw of 30mA. It provides an audible tone based on the frequency of the signal being generated by the pulse-width modulation (PWM) pin that it is connected to.

Below is a table showing the power specifications for each hardware component:

Table 7: List of hardware components and power characteristics

Device	Working Voltage	Max Current Draw
RCWL-1601 Ultrasonic Sensor	3-5V	2.2mA
YF-B1 Water Flow Sensor	4.5V	15mA
Relay Module	5V	72mA
DC Water Pump	6-12V	500mA
Piezo Buzzer	3-250V	30mA

Figure 15 shows a schematic view of the full hardware design. A breadboard view of this design can be found in Appendix A. (Note: this schematic diagram assumes that the RPi module has been fitted with a Sense HAT using long-pin headers to allow for easy connections to the RPi pins on top of the Sense HAT).

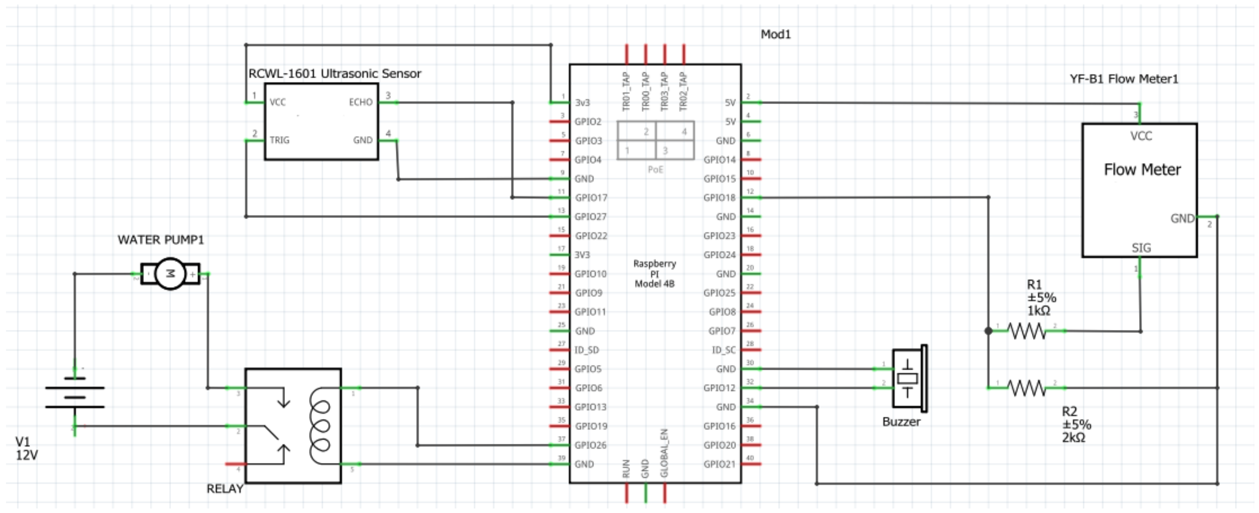


Figure 15: System Hardware Schematic

5.0 GUI Design

The main view of the smartphone application is shown below. This application presents key information from the firebase database regarding the logged in user as well as key information from their selected Water Buddy Station.

The intended use of the app is for a way that Users and Administrators can view key data about accounts and Water Buddy Stations without having direct access to the firebase database themselves.

An administrator account will have access to all water buddy station information regardless of ownership.

When first opening the application the user is prompted to log in or create a new account. The window that does this is shown as below in Figure 16.

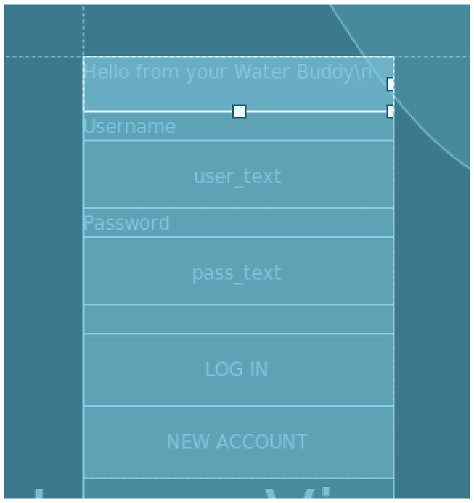


Figure 16: Wireframe of the Smart Application Login Page

A flow chart of this interaction can be seen in the figure below (Figure 17).

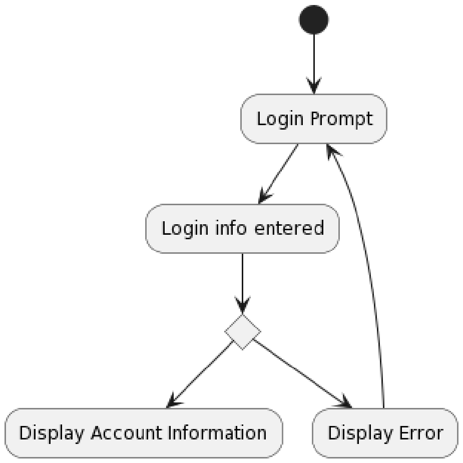


Figure 17: Flowchart of the Smart App Login Functionality

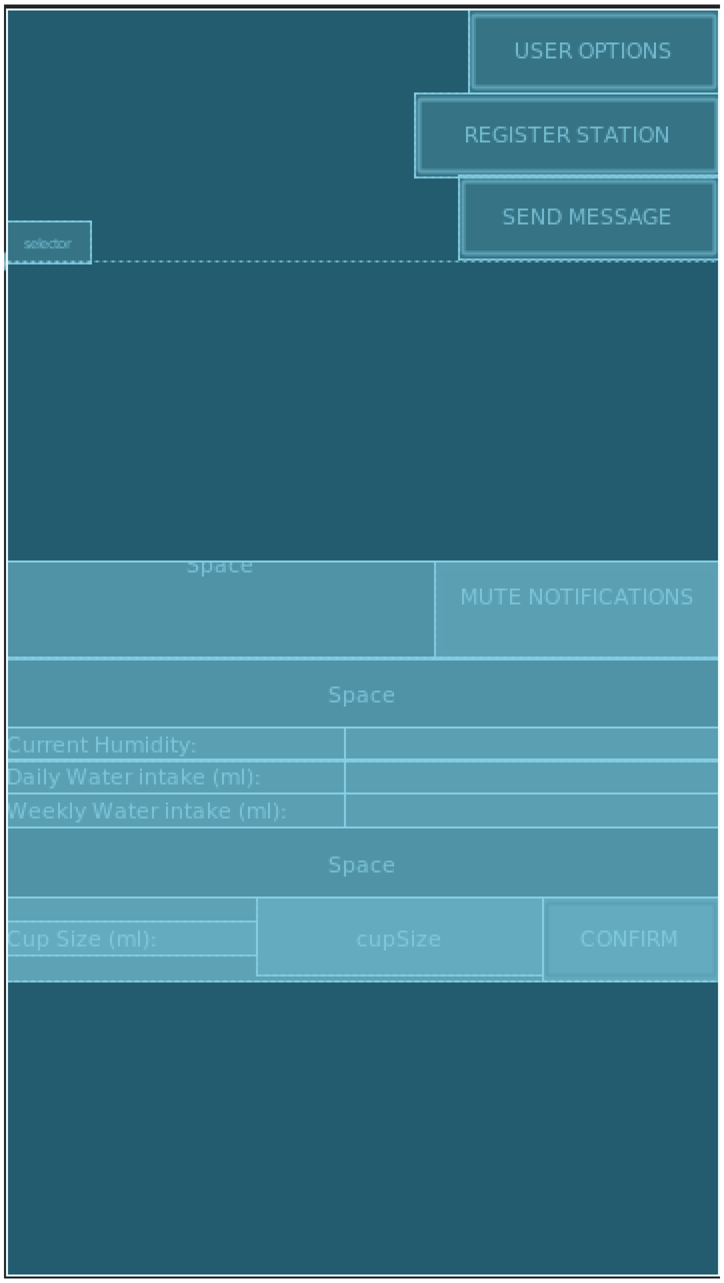


Figure 18: Wireframe of the Smart Application's Main Page

The main application view is shown as the figure above (Figure 18), where a User can use the "selector" spinner button to view information on any Water Buddy Station they have registered to their account.

The "REGISTER STATION" Button will prompt the User to enter a station name to register to their account, only existing stations may be added.

The "SEND MESSAGE" Button shall prompt the user to select a desired station belonging to themself or a friend and then enter a message to be displayed to that station.

A sequence diagram of the send message button and a flow chart are shown in the figures below (Figure 19 and Figure 20)

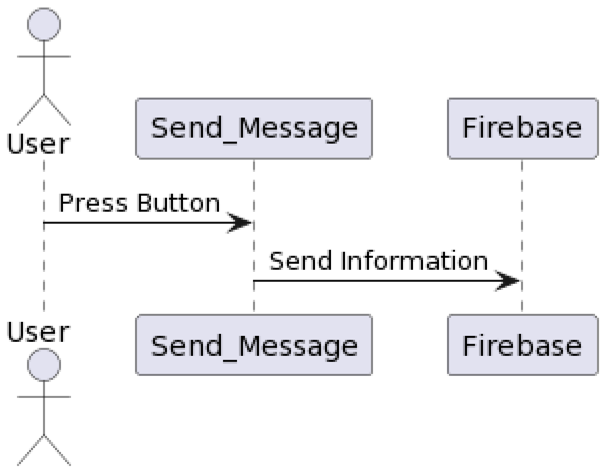


Figure 19: Sequence Diagram of the Smartphone App sending a message to the Database

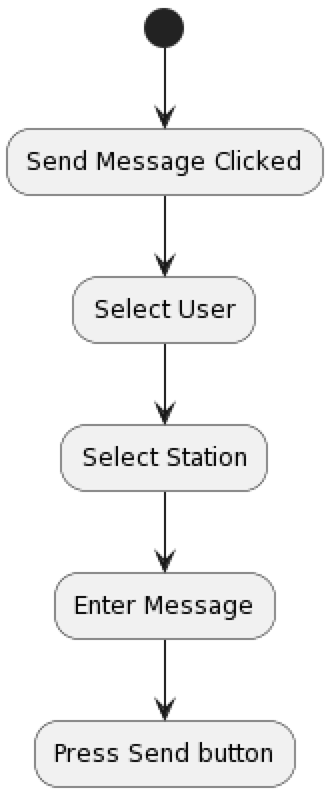


Figure 20: Flow Chart of Application Message Sending Behaviour

5.1 Table of Users/Roles

The system will have two intended types of Users. A Consumer and an Administrator. Details of each role is shown in the table below. The different users are intended to separate access privileges to the database information inside the system.

Table 8: System Roles

Roles	Description
Consumer	Intended user for the Water Buddy station, only has access to their registered water buddy stations
Administrator	Edit privileges for all users and access to all information

The view that both users have while using the GUI will be the same, however an Administrator can access all information from any station without needing to register any of them.

6.0 Test Plans

6.1 End-to-end Communication Demo Test Plan

To ensure that all communications are behaving as intended an end-to-end test plan was created.

For the End-to-End Communication Demo we plan to demonstrate the following behaviours:

Table 9: End to End Test Plan

Test	Pass Condition	Fail Condition
Water Buddy Station is running with Humidity sensor on	Humidity value on the Firebase Database updates every 5 seconds	Humidity value does not update every 5 seconds
Create and Send Message from User smartphone application to Water Buddy Station #1	Message is listed on Firebase Database then deleted once read by Water Buddy Station #1 Message is displayed on the Station's SenseHat	Message fails to publish on Database Message remains after being read Message is read by unintended Water Buddy Station
Ultrasonic sensor detects water cup	Object is detected when placed in close proximity to the sensor	Object is not detected
Buzzer Notifies that time to receive water	Buzzer play notification tune and then stops	Buzzer fails to make any noise at all Buzzer continues to sound after

		tune is complete
Fill System	Relay activates when prompted and flowSensor polls for flow rate	Relay is not activated Flow Sensor doesn’t output flow rate

6.2 Unit Test Demo Test Plan

The next phase in the project is to demonstrate each individual unit in the system can work independently with a unit test demo, as shown in Table 10 below.

Table 10: Unit Test Demo Test Plan

Test	Pass Condition	Fail Condition
Create User Account on Android application with name “test” and password “123”	Account successfully logged in and user information displayed	Account not added to database Account information not displayed or editable on app
Create User Account on Android application with name “test”	Application informs user that there was an error while creating the account due to one already existing	Previous account “test” overwritten on the database
Log in with incorrect information for name “test”	Access not granted to account	Access granted to account
WaterBuddy Initialization (Create Firebase Entry and Local Database)	When the WaterBuddy is initialized it ensures an entry exists for it in the firebase database and that it is maintaining a local database	Calls to the Firebase database are not made, a local database is not created if not existing.
WaterBuddy Loop is run periodically and makes calls to the subroutines mentioned in Section 3.1.1 (Connected to the internet and not connected)	Calls are made to the many subroutines and are caught by the test suite	Any of the calls are missing regardless of the WaterBuddy’s connection to the internet
Display multiple messages through the SenseHatDisplay module and	see the message in succession on the screen via separate threads	Messages are displayed in the wrong order or not displayed at all

Make calls to the fill system from the WaterBuddy main loop	The fill system module (or simulator) begins filling the cup on it's on thread. Successive calls from the mainloop are not processed until the fill system module is finished running.	Multiple threads are made for the fill system. The fill system is not triggered by the main loop. The fill system is not run on it's own thread.
Ultrasonic sensor	Ultrasonic sensor detects if an object is placed within a certain distance	There is no notification that the ultrasonic sensor has detected an object within the defined distance A notification is delivered without an object being placed within the defined distance
Water pump + Relay + Power supply	The relay properly closes the water pump circuit when triggered by the RPi	The water pump does not turn on despite the relay being triggered by the RPi
Flow Sensor	Flow sensor creates a GPIO interrupt when water flow is detected and displays an accurate flow rate reading	Water flow is not detected Flow rate is incorrect
Piezo Buzzer	Buzzer successfully plays the correct chime, depending on which code it is given	Buzzer does not sound at all Buzzer plays the wrong chime
Sense HAT Display	Displays the proper message when prompted, based on which code it is given	Displays the wrong message or no message when prompted Displays messages without being prompted
Sense HAT Humidity Sensor	Returns humidity readings when prompted	Does not return humidity readings when prompted Returns illogical or incorrect readings

6.3 Final Demo Test Plan

In order to run the final demo test plan, a User must have the Water Buddy Station, a smart phone, water readily available, and a cup of their preferred size.

Table 11: Final Demo Test Plan

Test	Pass Condition	Fail Condition
Inside the smartphone application create a User Account and update User Settings	Firebase Database reflects all user settings that were set in the app	Firebase database information does not match the user account
Register Station #1 to User	Station 1 pulls information from user account and is shown on the smartphone application	Smartphone application fails to sync information
Water Drank From Station # 1	Water Usage successfully recorded and updated on Database Recommendation time for next water drinking has been set	Water Usage not updated or seen by each Water Station Water updated but not properly dispensed from the system
Change cup size and fill on station	Amount of water deposited matches the amount specified on the User's Android application	Incorrect amount of water deposited
Disconnect WaterBuddy Station from Wifi	The Station continues running as normal, pulling it's settings from the local database and storing water history in the local database to be uploaded later.	The WaterBuddy crashes. The fill system is no longer activated by the ultrasonic sensor. The WaterHistory is not saved to the local database.
Reboot WaterBuddy and re-establish Wifi Connection	The Station boots up and begins running it's loop. All functionality is the same as the previous test until wifi is established, then the WaterBuddy pulls it's new settings, uploads it's water History, and displays any messages sent to it.	The WaterBuddy Station does not come back online. The fill system does not work post reboot. No Reminders are displayed while offline. Water history is not pushed to the database when connection is established. Messages are not displayed. Settings are not updated.

7.0 Project Update

Currently the project is progressing as expected. Parts arrived in an orderly fashion and no issues were presented while testing hardware usage. One issue was found where the sizing of the pump tube and the flow sensor diameter are different sizes, a remedy for this problem is to purchase two hose adapters to ensure that the parts fit properly together. We are currently a little bit behind on development of the Water Buddy Software but the work being completed to prepare for the End to End Demo should bring us back on schedule. Modelling of the Water Buddy hosing was pushed back until the hardware arrived so we could ensure the dimensions would be correct but can now proceed, a one week gap was left in the gantt chart for this area of the project and that has now been filled. Nick will be assisting Michael in completing the Water Buddy Station software by completing the classes relating to the fill system and buzzer hardware.

The team's progress has been very good thus far with each member of the group completing their required duties. Lots of work has already been done and a clear path to the completion of the system can now be seen. The following view weeks will be very exciting to see as the rest of the project unfolds. We are more or less still on our initial plan aside from some schedule changes shown in the gantt chart in Section 7.2.

7.1 Project Milestones

Table 12: Revised Project Milestones

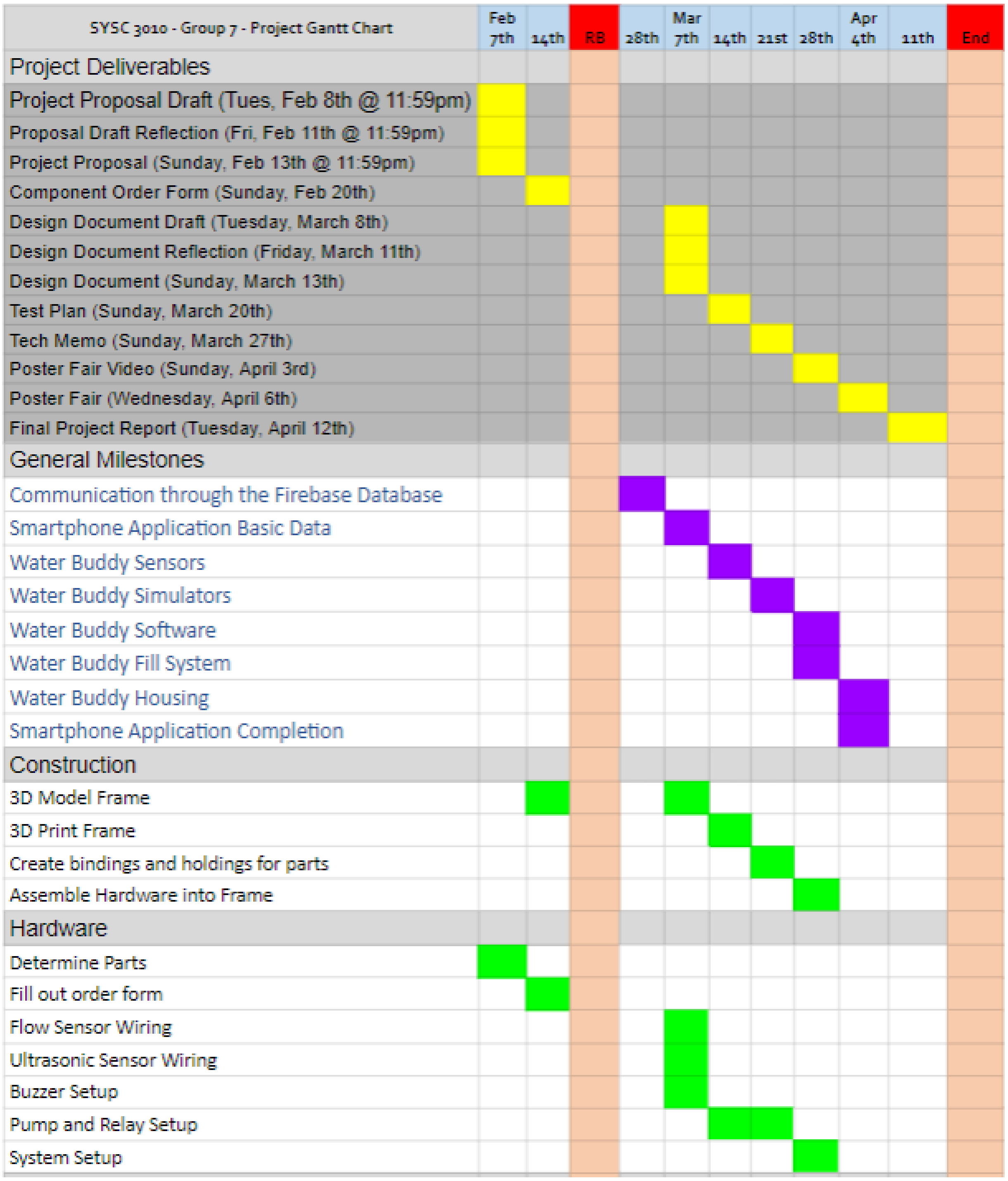
Milestone	Expected Date	Finished Date	Description
Communication through the Firebase Database	March 6th	March 6th	Our first milestone will be establishing proper connection to the firebase database from the Water Buddy RPi, the Water Buddy Simulator code, and the Smartphone application. This will require the establishment of a data standard (Schema) for the database and the code needed to utilize the firebase APIs. When we can read and write information and pass messages between all components this milestone will be complete.
Smartphone Application Basic Data	March 13th	March 6th	The next major milestone will be continuing development of the smartphone application to display specific information to the user and allow them to enter their personal information as well as adjust the settings for their Water Buddy.
Water Buddy Sensors	March 20th		The first step in the hardware development will be to connect the Raspberry Pi with the various sensors and write the code loop to poll and communicate to monitor all these sensors first individually and then simultaneously.

Water Buddy Simulators	March 27th		Along with the development of the software we will be creating simulators to demonstrate the scalability of the project without accruing additional hardware costs. This will mean writing functions to simulate the hardware components both in input from the senseHat joystick and then displaying the state of the simulated hardware on the senseHat display. These simulators will function the same as the regular water buddies but have entirely simulated hardware components.
Water Buddy Fill System	April 3rd		The next step in the hardware development will be to develop the Water Buddy fill system to respond to cup detection on the ultrasonic sensor, trigger the water pump relay, detect the cup being filled and turn off the relay when the flow sensor has detected that enough water has been added to the glass. This flow should run when triggered by the user for this milestone and later be controlled by the system.
Water Buddy Software	April 3rd		Along with the Water Buddys hardware development will be the development of the software, this includes controlling the fill system hardware and communicating with the database but also displaying information on the senseHat, alerting the user with the buzzer and overall giving the unit some life. This milestone will be completed when the system runs independently of a user, communicating with the database and controlling it’s hardware based on timing loops, settings from the database and user interaction.
Water Buddy Housing	April 10th		Once the hardware is connected and working it will be affixed in a 3D printed housing to hold components in place, hide ugly wiring, mount the station on a desk and give the user a distinct place to place their cup and a nice looking addition to their desk. This will complete the hardware set-up of the project.
Smartphone Application Completion	April 10th		To complete the project the smartphone application will be able to display data from the server based on updates from the Water Buddy unit. The user will be able to manually input glasses of water drank elsewhere, update their personal data, modify their

			units settings, add and remove friends, and send messages. Admin users will be able to log into the system to modify data as needed and issue community wide events and messages. The user interface for the application will be beautified and the application will be available on each of our smartphone devices.

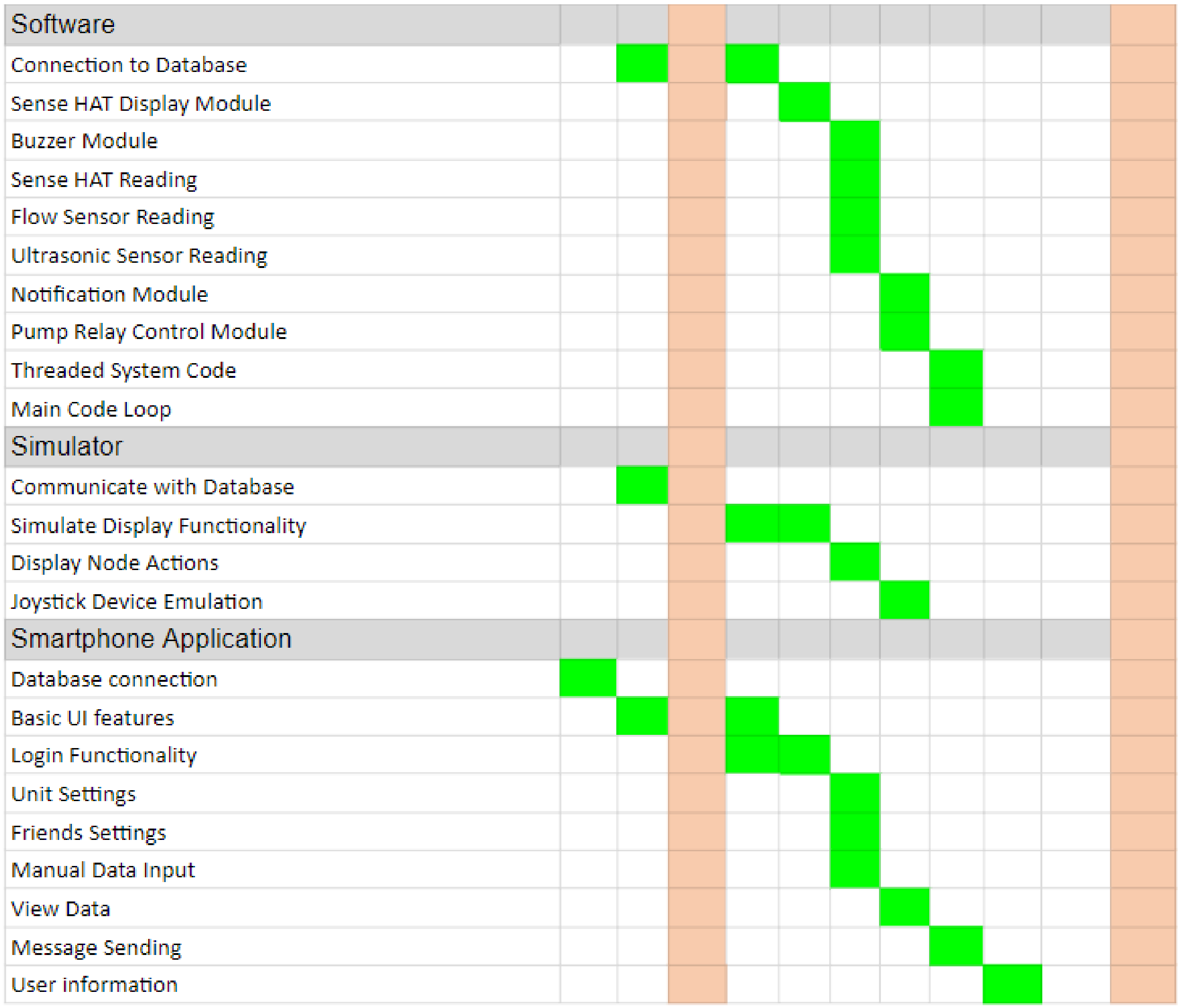
7.2 Schedule of Activities

The project deliverables, project milestones, and each detailed task (As described in Appendix A) was fit into the projects schedule as a row on the following Gantt Chart shown in Figure 21 and Figure 22. Yellow and purple boxes denote project deliverable and Project milestone due dates and each barren box represents the time that that specific task will be developed. The Gantt Chart has been revised to show changes to the schedule, specifically printing the housing, and some of the Water Buddy Software programming.



SYSC 3010 - Group 7 - Project Gantt Chart	Feb 7th	14th	RB	28th	Mar 7th	14th	21st	28th	Apr 4th	11th	End
Project Deliverables											
Project Proposal Draft (Tues, Feb 8th @ 11:59pm)	🟨										
Proposal Draft Reflection (Fri, Feb 11th @ 11:59pm)	🟨										
Project Proposal (Sunday, Feb 13th @ 11:59pm)	🟨										
Component Order Form (Sunday, Feb 20th)		🟨									
Design Document Draft (Tuesday, March 8th)					🟨						
Design Document Reflection (Friday, March 11th)					🟨						
Design Document (Sunday, March 13th)					🟨						
Test Plan (Sunday, March 20th)						🟨					
Tech Memo (Sunday, March 27th)							🟨				
Poster Fair Video (Sunday, April 3rd)								🟨			
Poster Fair (Wednesday, April 6th)									🟨		
Final Project Report (Tuesday, April 12th)										🟨	
General Milestones											
Communication through the Firebase Database				🟪							
Smartphone Application Basic Data					🟪						
Water Buddy Sensors						🟪					
Water Buddy Simulators							🟪				
Water Buddy Software								🟪			
Water Buddy Fill System								🟪			
Water Buddy Housing									🟪		
Smartphone Application Completion									🟪		
Construction											
3D Model Frame		🟩			🟩						
3D Print Frame						🟩					
Create bindings and holdings for parts							🟩				
Assemble Hardware into Frame								🟩			
Hardware											
Determine Parts	🟩										
Fill out order form		🟩									
Flow Sensor Wiring					🟩						
Ultrasonic Sensor Wiring					🟩						
Buzzer Setup					🟩						
Pump and Relay Setup						🟩	🟩				
System Setup								🟩			

Figure 21: Revised Project Gantt Chart Part 1



Software										
Connection to Database		🟩		🟩						
Sense HAT Display Module					🟩					
Buzzer Module						🟩				
Sense HAT Reading						🟩				
Flow Sensor Reading						🟩				
Ultrasonic Sensor Reading						🟩				
Notification Module							🟩			
Pump Relay Control Module							🟩			
Threaded System Code								🟩		
Main Code Loop								🟩		
Simulator										
Communicate with Database		🟩								
Simulate Display Functionality				🟩	🟩					
Display Node Actions						🟩				
Joystick Device Emulation							🟩			
Smartphone Application										
Database connection	🟩									
Basic UI features		🟩		🟩						
Login Functionality				🟩	🟩					
Unit Settings						🟩				
Friends Settings						🟩				
Manual Data Input						🟩				
View Data							🟩			
Message Sending								🟩		
User information									🟩	

Figure 22: Revised Project Gantt Chart Part 2

Appendices

Appendix A: Breadboard view of the system's hardware connections

Figure 23 shows a graphical view of the system's hardware components. The included components consist of (from left to right): 12V DC power supply, DC water pump, RPi 4B, relay module, piezo buzzer, Sense HAT, water flow sensor, ultrasonic distance sensor.

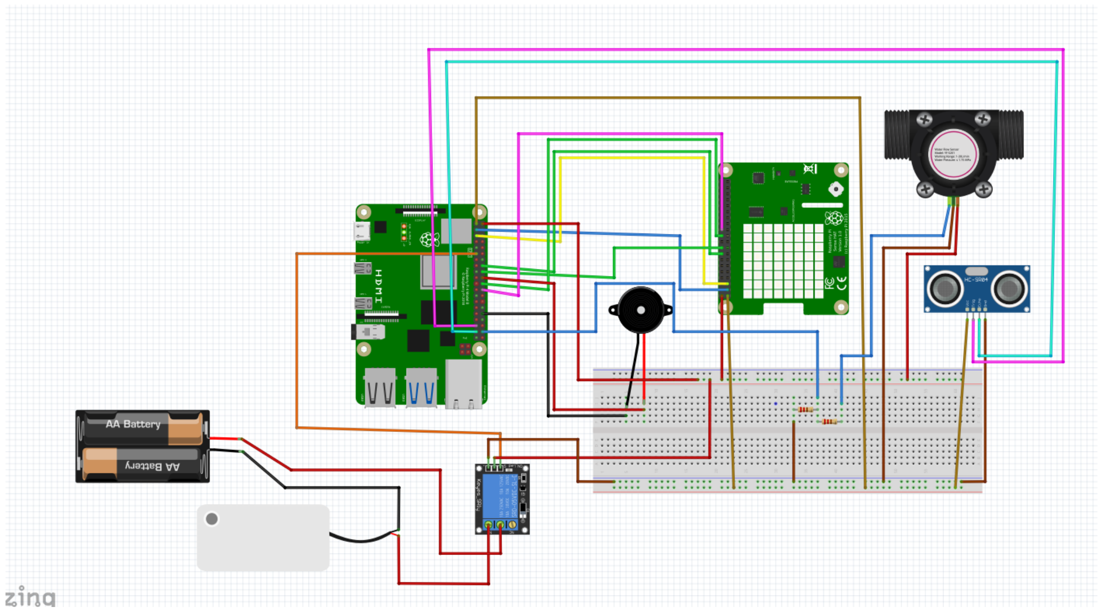


Figure 23: Breadboard view of the system's hardware connections