SYSC3010 | Computer Systems Development Project

The Mocktender Mocktail Mixing Machine

Detailed Design

Figure 1. Example of mocktail drinks to be made by the Mocktender machine [1].

Group L3-G8

Matt Reid, 101140593
Ethan Bradley, 101158848
Duncan MacLeod, 101160585

TA: Roger Selzler

March 15th, 2023

Table of Contents

1 Problem Statement

A mocktail is a non-alcoholic drink that captures the essence of a cocktail without the downsides of alcohol consumption. From work offices where you want to stay sharp while having fun drinks for lunch to wanting a healthier alternative to drinking cocktails at home, there are lots of reasons to enjoy a mocktail. As more people look towards health and wellness trends, the number of people drinking non-alcoholic beverages is rapidly increasing. A NielsenIQ study in October 2021 found that non-alcoholic beverage sales had gone up 33.2% in 12 months to a total market of $331 million [2]. In the United States, 30% of people who are of drinking age do not drink alcohol [3], and according to a 2021 survey by Statistics Canada, 1 in 5 Canadians said they have been drinking less than before the COVID-19 pandemic [4].

The objective of the project is to create an automated mocktail mixing machine that provides a web interface to allow dispensing drinks, administrative access, real-time liquid level monitoring, and recipe creation. It replaces the need for at home mixing equipment, and bartenders at events to eliminate the inflated cost of making mixed drinks. It will provide timely mocktails from specified recipes, allowing for easy access to mocktails and less time spent mixing with more time spent drinking.

1.1 Functional Requirements

To create an easy to use and fully functional mocktail mixing machine, The Mocktender will meet the following functional requirements:

Automatic Liquid Mixing: The machine will automatically mix liquids together based on defined recipes in a database. The database will contain recipes that specify liquid percentages, and the Raspberry pi will use a pump to dispense the correct amount of each liquid into a cup.

Administrative Access: The machine will allow an admin user to login to the web interface and lock the machine from use or modify the liquid types that are available in the machine. Admin users can also control who has access to the machine.

Remote Dispensing Access: The machine will allow users to access the drink pouring menu remotely on a web interface. GUI (Graphical User Interface) buttons will be available to dispense selected drink recipes from a list.

Recipe Creation and Storage: The machine will allow users to create new recipes in a web interface and store them in a database. Recipes consist of liquid names and the percentage of each liquid that should be pumped into the cup.

Remote Liquid Level Monitoring: The machine will allow users to view the current liquid levels through a web interface page. An ultrasonic sensor will be used to provide real-time updates of the liquid levels. Notifications will be sent to registered users that liquid levels are low.

2 Design Overview

2.1 System Overview Diagram

The Mocktender System consists of a Raspberry Pi device which provides a user interface (UserInterface node) that is connected to one or more Raspberry pi devices which represent I/O controllers (IOController/SimulationSystem nodes) of a Mocktender system. In our project, we use one Raspberry Pi device to act as a physical Mocktender IOController as well as a SimulationSystem Pi device which simulates the behaviour of a Mocktender device to showcase that more than one device can be utilized at once (scalability). The UserInterface device and the IOController/SystemSimulator devices communicate over HTTP using a Firebase Realtime Database which stores data (messages) sent between the devices. These messages include requests for real-time liquid levels and instructions for the pumps to dispense a desired recipe. The Flask webserver hosted by the UserInterface Pi device is accessed by a user's web browser over HTTP, providing the GUI for the system. This deployment diagram is shown in Figure 2 below.

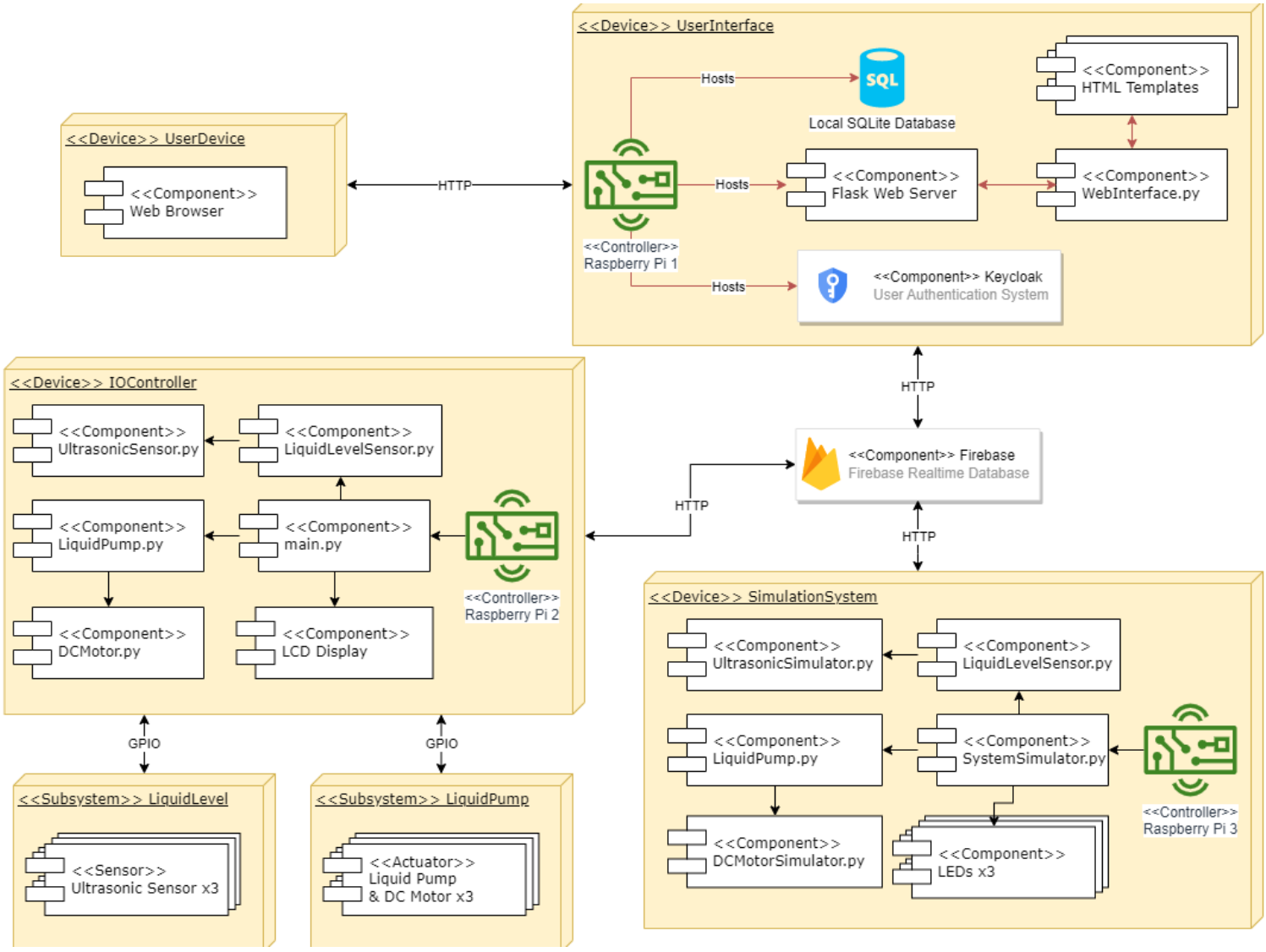

Figure 2. Deployment diagram of The Mocktender Mocktail Mixing Machine.

As discussed above, there are three device systems which are controlled by a different Raspberry Pi. The first Raspberry Pi controls the "UserInterface" node, and hosts the local Flask based web interface as well as the SQLite database and Keycloak server to provide local data storage and user authentication. The web interface provides the user with access to the Mocktender system and connects to the Firebase Realtime Database to send instructions to the LiquidPump subsystem and receive liquid level data from the LiquidLevel subsystem. More information about the functionality of the Flask web interface can be found in Section 5 (GUI Design). The second Raspberry Pi in our project controls the "IOController" node and is connected over GPIO to the LiquidLevel and LiquidPump subsystems which contain physical Ultrasonic Sensors, and Liquid Pump actuators. The device runs a Python script which controls getting the real-time liquid level and sending instructions to the liquid pumps. It also has an LCD display which provides output to the user that is getting a drink from the machine. Additional information about the software systems can be found in Section 3, and the hardware systems can be found in Section 4. The internal systems that make up the IOController, including the Liquid Pump Subsystem and the Liquid Level Subsystem can be seen below in Figure 3.

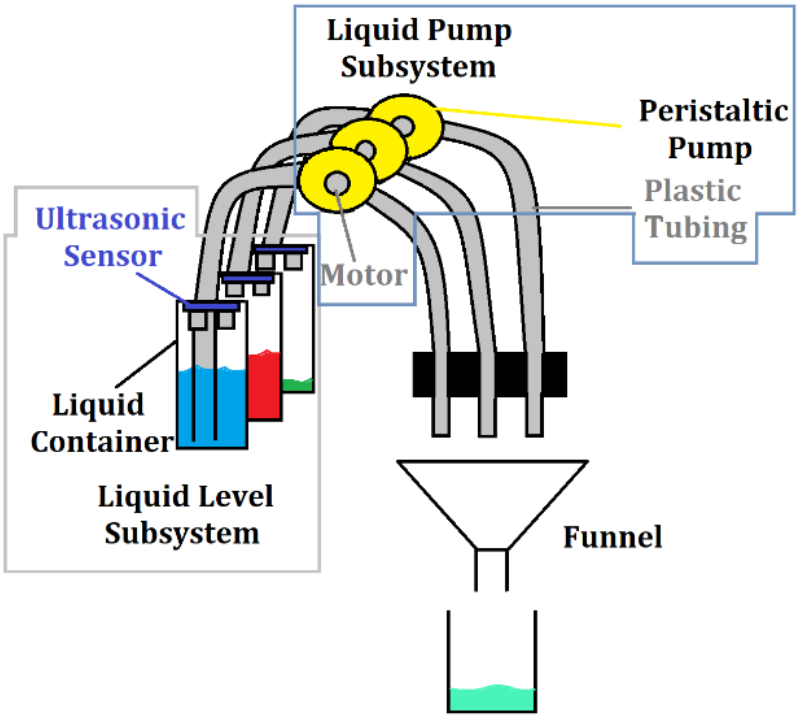

Figure 3. Sketch of internal Mocktender systems.

The third Raspberry Pi controls the "SimulationSystem" node, which simulates the functionality of the IOController. Like the IOController, it provides liquid level data and takes liquid pump instructions, however instead of using physical Ultrasonic Sensors, and Liquid Pumps, they are simulated in software. Instead of an LCD device providing output, the simulation provides output in a terminal simulating what would go on the LCD device. The SimulationSystem also has three LEDs which simulate the pumps and light up when the pumps are being run to show that the instructions are being receive and processed correctly. The SystemSimulator both showcases that the system can work with more than one device, and that the software works independent of the physical hardware sensors and actuators.

2.2 Communication Protocols

There are two different communication types used by the Mocktender system, HTTP connections for the Firebase Realtime Database, and GPIO connections for the hardware sensors and actuators.

The liquid level sensors can be controlled using GPIO as they operate using digital signals. Similarly, the pump controllers can be controlled using GPIO. As we are implementing the pumps with DC motors, we will their speed using a PWM GPIO signal so that the liquid flow rate can be more accurately controlled. The Liquid Level system will be on a periodic timing loop and will provide the script in the UserInterface with the current liquid level in real-time.

Each Raspberry Pi node will upload to and read from the Firebase Realtime Database. The Firebase database will be used to both store data necessary for the operation of the Mocktender as well as to provide a platform for the Raspberry Pi nodes to issue and receive commands. The Raspberry Pi nodes will communicate with the Firebase database using HTTP. Section 2.2.1 below shows the various message types and data formats that will be sent between nodes via the Firebase Realtime Database.

2.2.1 Communication Protocol Tables

The internode communication between the UserInterface, IOController, and SimulationSystem through the Firebase Realtime Database, and the LiquidPump subsystem are outlined in the communication protocol tables in the following sections.

2.2.1.1 User Interface

The User Interface communicates with every node of the system. It primarily communicates with the Firebase Database to read the Liquid Level of a Mocktender Machine, fetch a recipe, change the liquid names, and adding users to a Mocktender. A route to control the IOController has also been left exposed so that the user is able to send commands to the hardware devices to make drinks using the Frontend Web Interface. The messages received by and sent from the UserInterface can be seen below in Table 2.2.1.1.

Table 2.2.1.1 Communication Protocol Table for the UserInterface node.

Sender	Receiver	Message	Request	Data Format

User Interface	Firebase	getLiquidLevel	GET	**Request**:
				JSON {
				"subject": "liquid-level",
				"deviceId": INTEGER,
				"containerNo": INTEGER
				}
				
				Response:
				JSON {
				“liquidLevel”: NUMBER,
				}

User Interface	Firebase	getRecipe	GET	**Request:** JSON: {

				“subject”: “recipes”,
				“deviceId”: INTEGER,
				“drink”: STRING
				}
				
				Response:
				JSON {
				“recipe”: STRING
				}

User Interface	Firebase	changeLiquid	PUT	**Request:**
				JSON {
				“deviceId”: INTEGER,
				"containerNo": INTEGER,
				"liquid": "newLiquid"
				}
				Response:
				JSON {
				“containerNo”: INTEGER,
				“liquid”: “newLiquid”
				}

User Interface	IOController	makeDrink	POST	**Request:**
				JSON {
				“deviceId”: INTEGER,
				“drinkId”: INTEGER,
				“recipe”: STRING
				}
				
				Responses:
				JSON (updateUI) {
				“deviceId”: INTEGER,
				“containerNo”: INTEGER,
				“liquidLevel”: NUMBER
				}
				
				JSON (notifyDrinkComplete) {
				“deviceId”: INTEGER,
				“drinkId”: INTEGER,
				“status”: BOOLEAN
				}

User Interface	WebServer	login	POST	**Request:** JSON { “subject”: “login”, “username”: STRING, “password”: STRING

				}

User Interface	WebServer	addUser	PUT	**Request:**
				JSON {
				“subject”: “addUser”,
				“deviceId”: INTEGER,
				“username”: STRING,
				“password”: STRING
				}
				
				Response:
				JSON (reject) {
				“deviceId”: INTEGER,
				“reason”: STRING
				}

User Interface	Firebase	subscribe	PUT	**Request**
				JSON {
				“subject”: “subscribe”,
				“deviceId”: INTEGER,
				“username”: STRING,
				“email”: STRING
				}

User Interface	Firebase	unsubscribe	DELETE	**Request**
				JSON {
				“subject”: “unsubscribe”,
				“deviceId”: INTEGER,
				“username”: STRING,
				“email”: STRING
				}

2.2.1.2 IOController and SimulationSystem

The IOController's communication is responsible for periodically updating the liquid level of the Mocktender's liquids on the Firebase Real-time database. It is also responsible for communicating with the Frontend of the UserInterface to update the liquid level values in real time. The SimulationSystem node is designed to replicate the IOController and will send the same messages as the IOController. Table 2.2.1.2 below shows the messages sent from the IOController/SimulationSystem to the other nodes of the system.

Table 2.2.1.2 Communication Protocol Table for the IOController and SimulationSystem node.

Sender	Receiver	Message	Request	Data Format

IOController/ SimulationSystem	Firebase	updateLiquidLevel	PUT	**Request:**
				JSON {
				"subject": "liquid-level",
				"deviceId": INTEGER,
				"containerNo": INTEGER,
				"liquidLevel": NUMBER
				}

IOController/ SimulationSystem	User Interface	updateUI	PUT	**Requests:**
				JSON (updateLiquidLevel) {
				"subject": "currentLevel",
				"deviceId": INTEGER,
				"containerNo": INTEGER,
				"liquidLevel": NUMBER
				}
				
				JSON (updateDrinkStatus) {
				"subject": "drinkComplete",
				"deviceId": INTEGER,
				"drinkStatus": BOOLEAN,
				}

IOController/ SimulationSystem	User Interface	notifyLowLiquid	POST	**Requests:**
				JSON {
				"subject": "lowLiquid",
				"deviceId": INTEGER,
				"containerNo": INTEGER,
				"liquidLevel": NUMBER
				}

2.2.1.3 Liquid Pump Subsystem

The liquid pump system is controlled by the UserInterface using the IOController as a proxy. It is responsible for listening the commands from the UserInterface to turn a pump on or off for a specified container. Table 2.2.1.3 below shows the commands that the liquid pump system will listen for.

Table 2.2.1.3 Communication Protocol Table for the LiquidPump.

Sender	Receiver	Message	Data Format

UserInterface	IOController::LiquidPump	activatePump	JSON {
			“subject”: “liquidPump”,
			“deviceId”: INTEGER,
			“containerNo”: INTEGER,
			“pumpState”: True
			}

UserInterface	IOController::LiquidPump	disablePump	JSON {
			“subject”: “liquidPump”,
			“deviceId”: INTEGER,
			“containerNo”: INTEGER,
			“pumpState”: False
			}

2.3 Message Sequence Diagrams

2.3.1 Message Sequence Diagram 1: MakeDrink Use Case

The most basic use case demonstrating the process to make a drink is the MakeDrink use case, which is done through the local web interface. The sequence diagram below in Figure 4 demonstrates the use case of making a drink using the web interface. The user will connect to the Frontend Flask web server using HTTP from their web browser. The Mocktender web interface will display all saved Mocktails that are currently available based on the liquids in the machine. The user will be able to select the drink they want to make. When the user confirms their choice, the SystemInterface will pour the drink by communicating with IOController, signalling which pumps to start and stop. When the drink is done, the UI will update indicating that the drink has been poured.

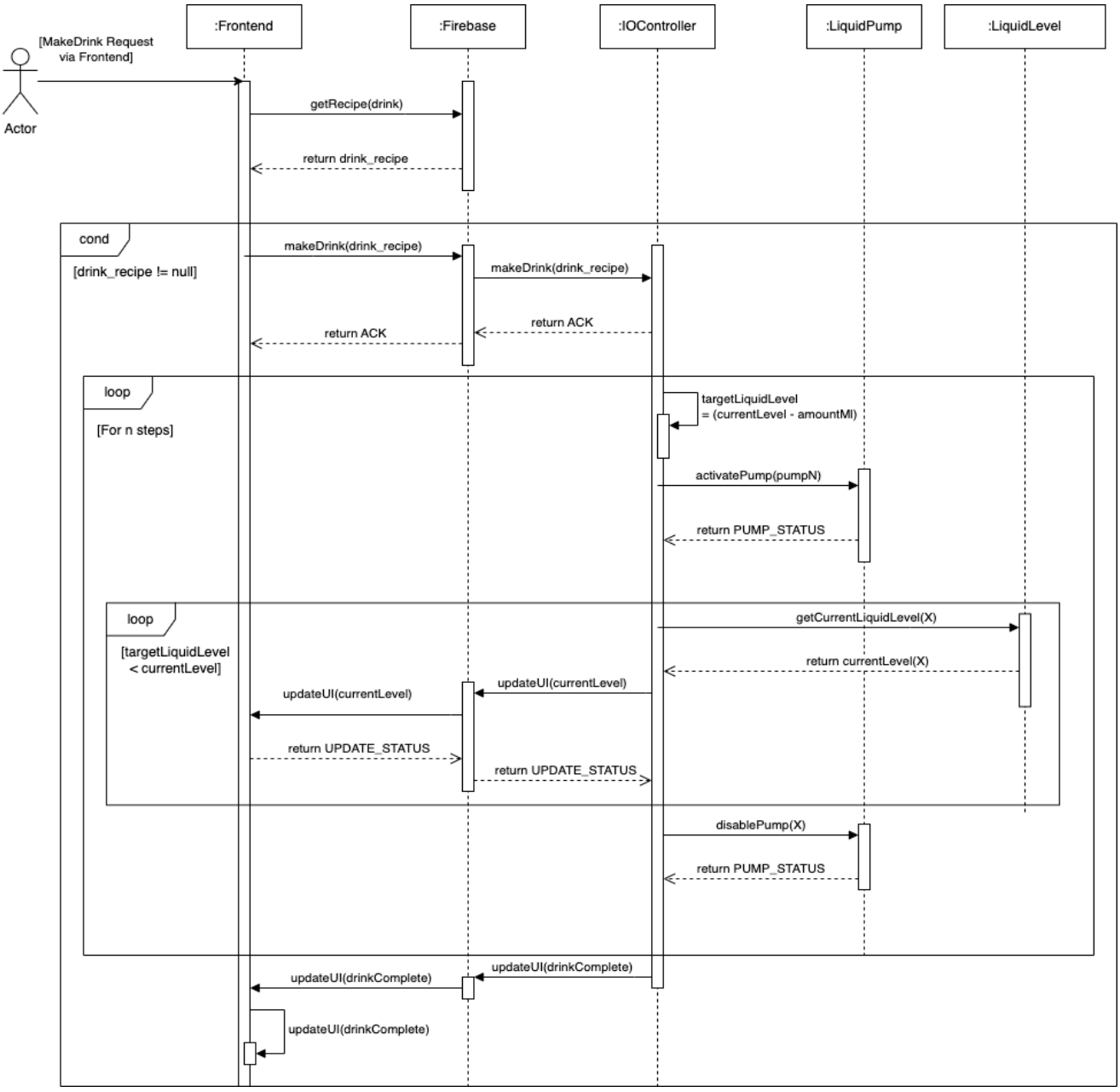

Figure 4. Sequence diagram for use case 1: MakeDrink.

2.3.2 Message Sequence Diagram 2: CheckLiquidLevels Use Case

The Mocktender is equipped with sensors to track how much liquid is left in each container, which the user can view through the web interface. The CheckLiquidLevels use case outlines the process of checking the current liquid levels as a user of the system. As in the previous use case, the user is connected to the Frontend Flask web server using HTTP from their web browser. Viewing the liquid levels can be completed by pressing a button on the UI, which will make a call to the local database to fetch the current liquid levels and display them to the user (See Section 5 for GUI design). After the button is pressed, the Flask Web Server sends a message to the local SQLite database to request the most recent liquid level sensor data. The SQLite database returns the result of the query, and it is displayed to the user on the web interface (LocalUI). The sequence diagram showcasing the process of checking a liquid level can be seen below in Figure 5.

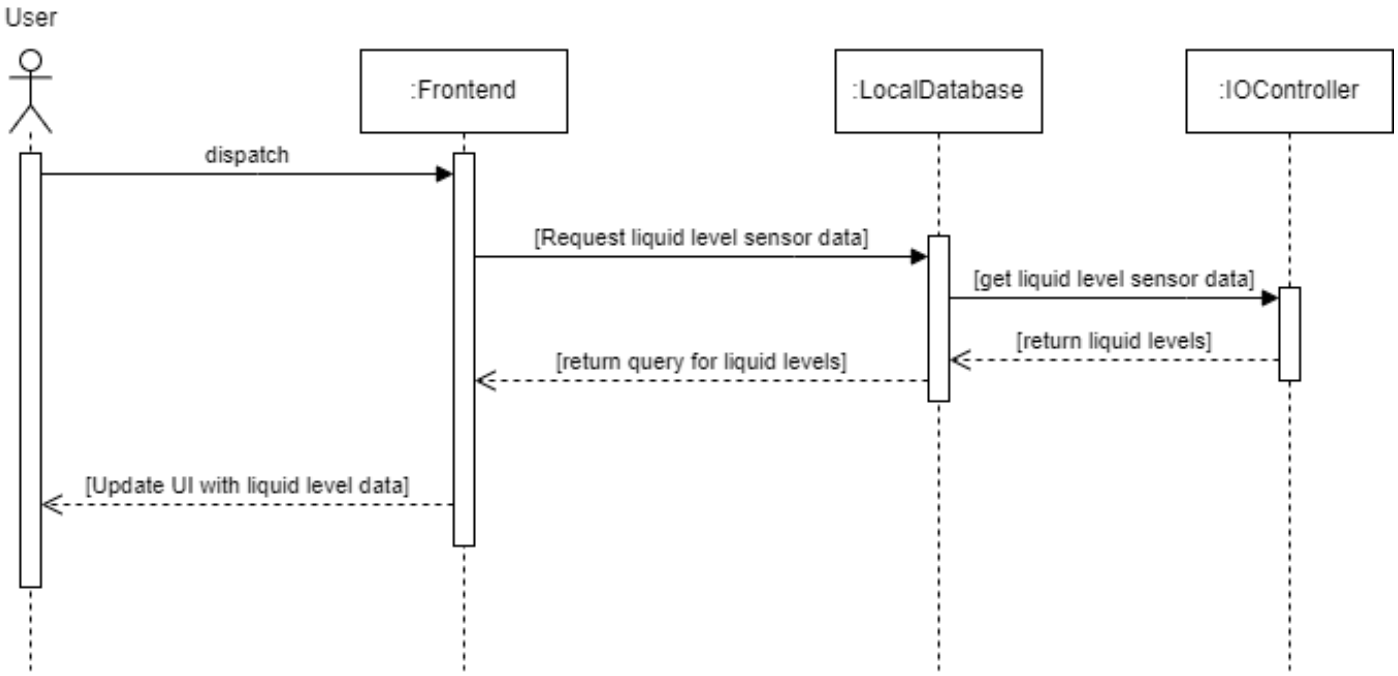

Figure 5. Sequence diagram for use case 2: CheckLiquidLevels.

2.3.3 Message Sequence Diagram 3: CreateNewRecipe Use Case

The MakeRecipe use case outlines the basic flow of the process to create a new mocktail recipe, which is done through the web interface. As in the other use cases, the user is connected to the web interface (LocalUI) through HTTP from their web browser. Figure 6 shows the process of interacting through the web interface to create a recipe. The user navigates through the web interface by going to “Settings” ->” Update UI” ->” View/Edit Saved Recipes.” From there, the user will be able to specify a sequence of drinks to pour to create their new mocktail. Once saved, the new mocktail is now available to make.

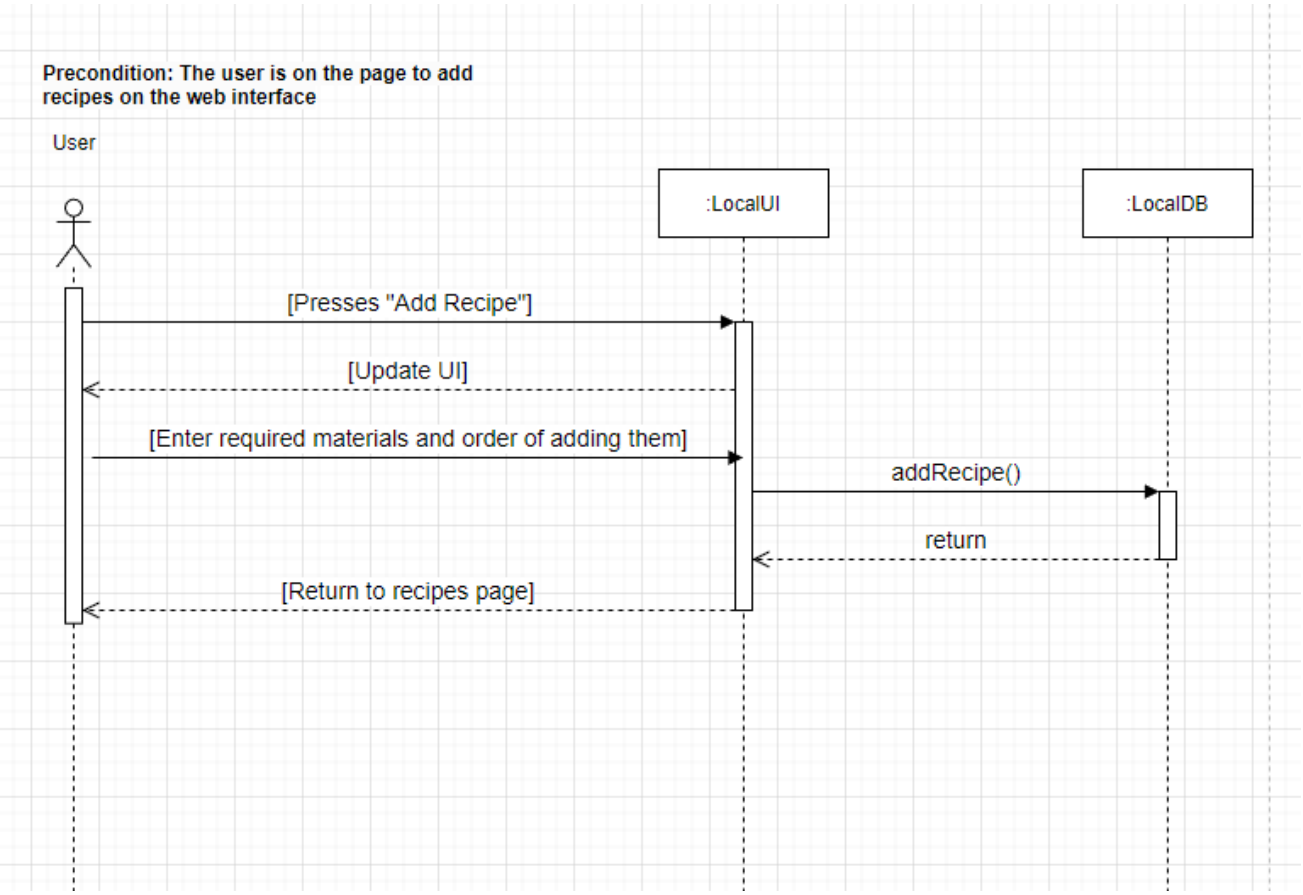

Figure 6. Sequence diagram for use case 3: CreateNewRecipe.

2.4 Database Table Design/Schema

The Mocktender will have both a local SQLite database and a cloud Firebase Realtime Database. The SQLite database is hosted on the UserInterface Pi device, and each instance is specific to each Mocktender machine (each IOController/SystemSimulation node). The information that is received from the IOController and SystemSimulation nodes through the cloud Firebase Database are stored in the local database to be retrieve by the frontend web server for display to the user. As shown above in section 2.2, the cloud Firebase Realtime Database is used to send messages between the nodes of the system containing liquid level and recipe information. Figure 7 below shows the machine-specific database schema which is merged between both database types. The database schema can be split into two categories, which are colour coded in red and purple. Database tables in red store data that is primarily used for the Mocktender's main function, creating mixed drinks. Database tables in purple and the Keycloak table are related to user information and authentication. The machine-specific database stores liquid level info saved mocktail recipes, the liquids that are in the containers, and data on whether the recipe can be created with the current liquid levels. It also stores the user information and user types which are used by the Keycloak user authentication service.

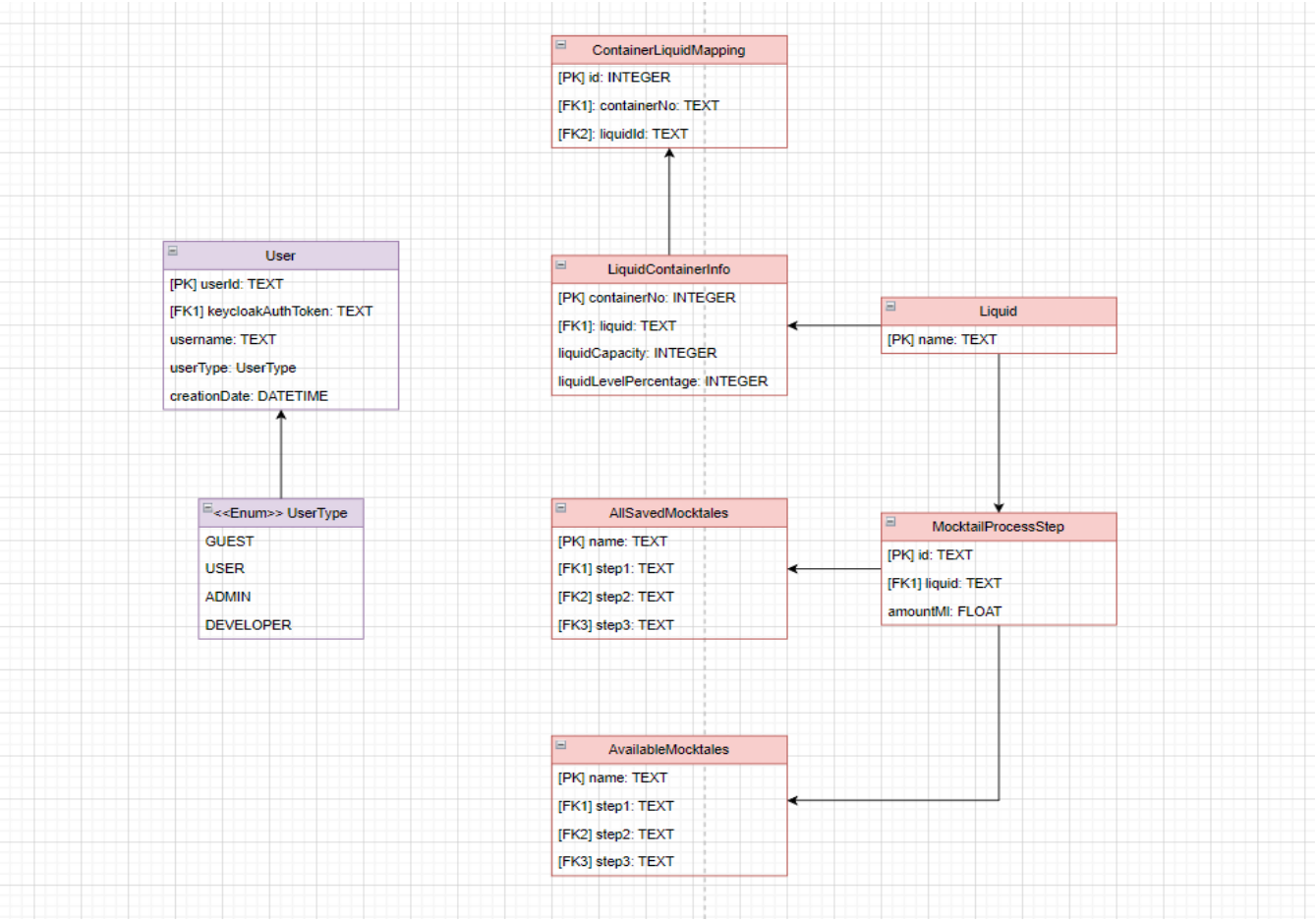

Figure 7. Machine-specific database schema.

In addition to the machine-specific database schema for each Mocktender, there will also be an internal management database schema to keep a record of machines, users, and the public Mocktail library, which

will be a public collection of mocktails open for users to browse, make, and contribute to. Figure 8 below shows the structure for the management database schema.

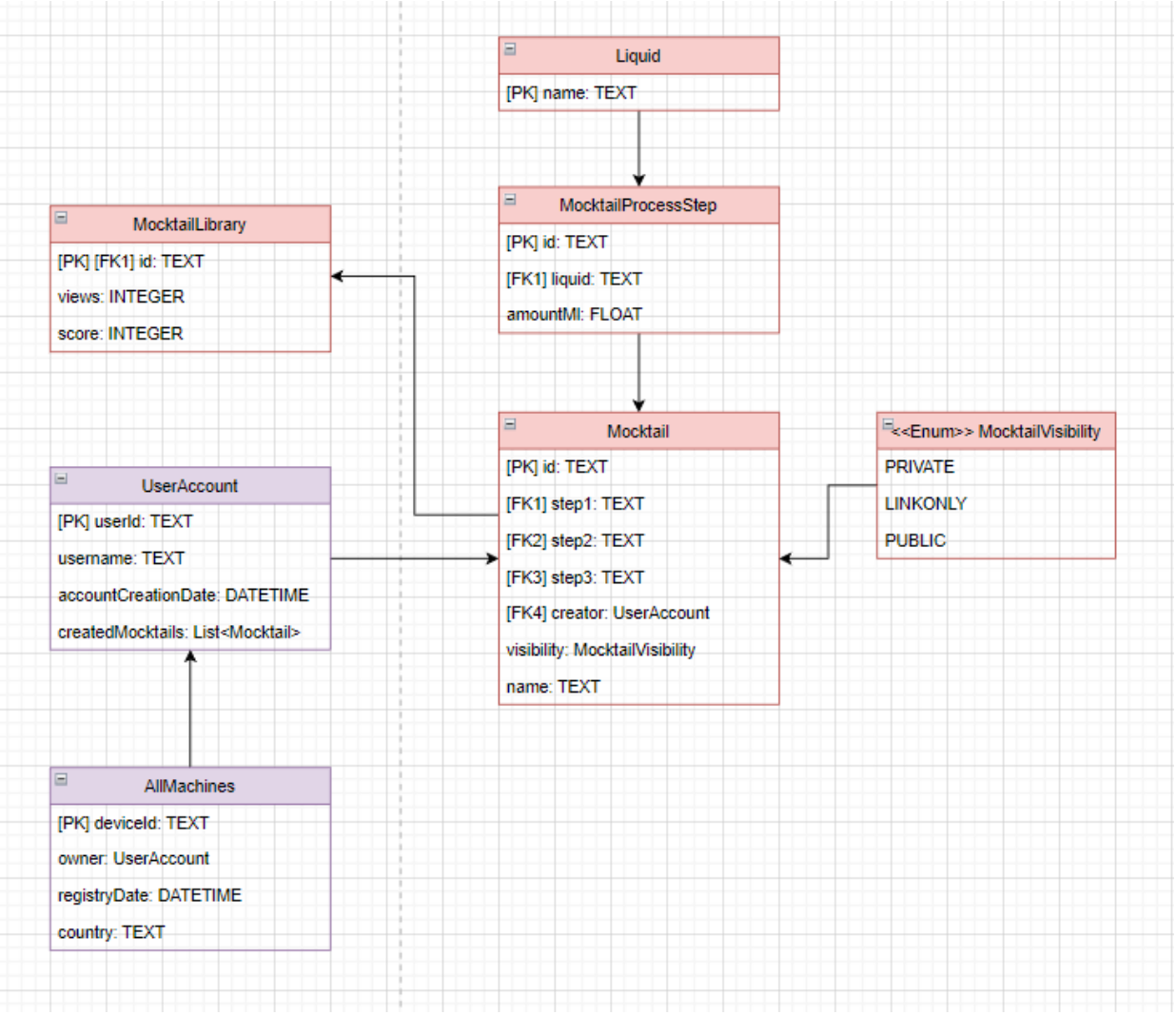

Figure 8. Cloud-Based Mocktail Management Schema.

3 Software Design

The Mocktender system will follow a Microservice architecture, with the core of the application being on the System Controller. The System Controller will make calls out to the microservices running beside it as needed. For the Mocktender, the system controllers are the 3 Raspberry Pi devices that control each node of the deployment diagram seen in Figure 2.

The implementation of the Mocktender will combine several languages and frameworks. The communication backbone of the system to send messages and commands between the nodes of the Mocktender will be implemented in JavaScript running in the NodeJS runtime environment. The Frontend Flask Webserver will be created using Python and rendered HTML templates.

The microservices will be developed using Python and will serve as the software interface for the Mocktender's hardware devices. These microservices will be implemented using an object-oriented design pattern using GPIO Zero as the framework for the hardware interface.

3.1 Software Design for a UserInterface Node

The UserInterface node hosts a Flask web server as a frontend to the system, and an SQLite database (LocalDB) as a backend. It receives messages through connection to the Firebase Realtime Database (discussed in Section 3.4).

The frontend Flask webserver is hosted in the UserInterface node by the Raspberry Pi device. The Flask webserver will utilize the Flask library in Python to create and run the Flask web server, as well as HTML templates to render the desired web pages. The web interface flow chart showcasing the web pages rendered on the flask server is shows below in Section 5 (GUI Design).

The local backend consists of an SQLite database which will allow for storing local data received through the Firebase database about the liquid level, as well as recipes created through the user interface. The SQLite database is accessed by the Flask web server when user input is received. The sequences of data being received by the database are shown in the sequence diagrams in Section 2.3. The database schema is described above in Section 2.4. A flow chart showcasing the process of moving between the web pages on the Flask web server is shown in Figure 9 below.

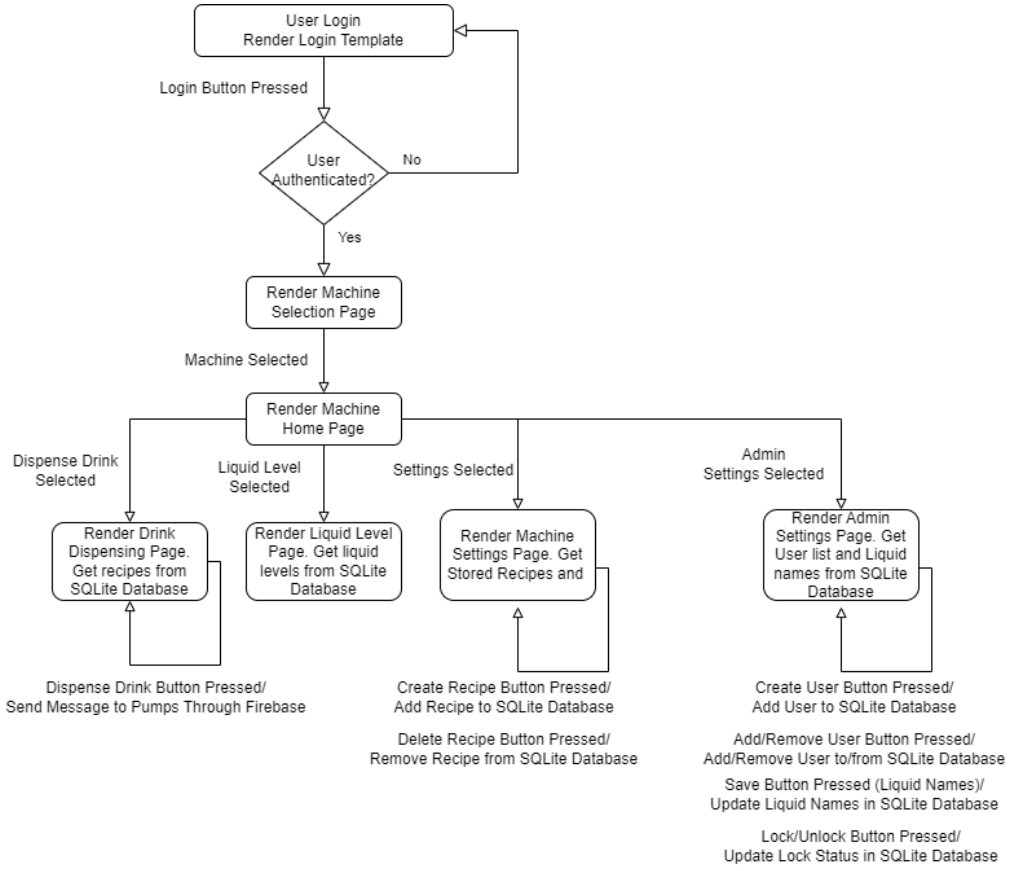

Figure 9. Flow Chart for Fronted Flask Web Server hosted by the UserInterface Node.

3.2 Software Design for Physical IOController Node

The IOController controls the hardware devices used for the Mocktender. Each IOController node has 3 liquid level sensors and 3 liquid pumps corresponding to the 3 liquids that can be mixed by the Mocktender. The IOController receives messages from the UserInterface which can either be a pump related command or a liquid level related command. The pumps that we are going to use for the Mocktender use DC Motors as the underlying actuator. This is modelled in the class diagram below (Figure 10) as the DCMotor class. UltrasonicSensor and DCMotor implement an interface of the same name which seems odd at first glance. It, however, allows for a more modular system if for example, it is decided to change out the DC Motors for stepper motors or, more relevantly, it is required to simulate hardware devices to prove a scalable system.

For the physical IOController node however, the UltrasonicSensor class will be composed of a DistanceSensor from the gpiozero library which provides an easy-to-use API for measuring distance. A LiquidLevelSensor will be composed of an UltrasonicSensor and the IOController will be composed of 3 LiquidLevelSensors stored in a list.

The DCMotor class will be composed of a Motor from the gpiozero library which will allow for the easy and precise control of a DC motor. A LiquidPump will be composed of a DCMotor and the IOController will be composed of 3 LiquidLevelSensors.

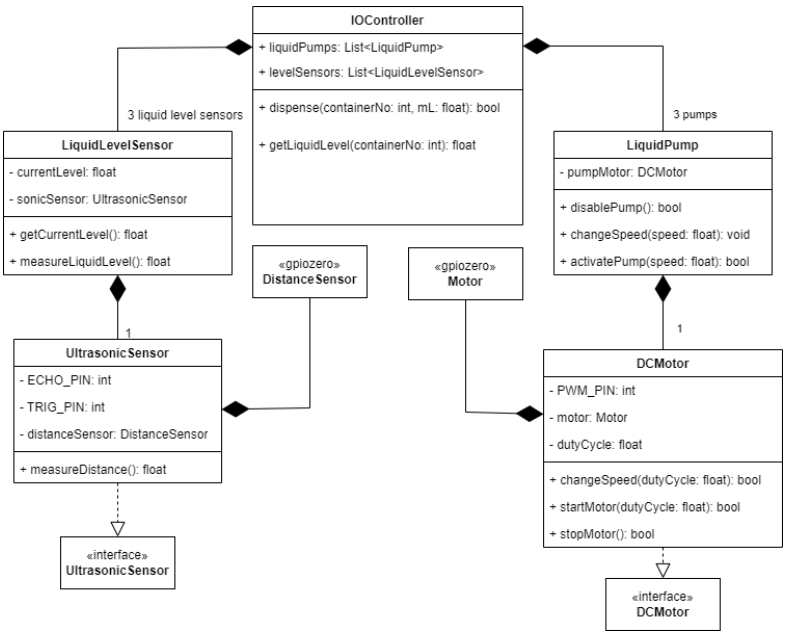

Figure 10. UML Class Diagram for the IOController Node.

3.3 Software Design for SystemSimulator Node

The SystemSimulator node acts as a proof-of-concept meant to demonstrate the scalability of the project. The SystemSimulator node is meant to be identical to the IOController node from the perspective of the UserInterface, except the input hardware in the IOController node will be simulated with software

and the output hardware will be simulated with LEDs. In Figure 11 below, an UltrasonicSimulator notably has no association with a gpiozero DistanceSensor as the distance data will be generated randomly.

Additionally, the DCMotorSimulator implements a DCMotor and is composed of a PWMLED instead of a Motor. As an LED can be driven in the same manner as a DC motor, the implementation of the DCMotorSimulator would not need to be changed from the DCMotor class. The UltrasonicSimulator class implements a UltrasonicSensor but would need to be rewritten from the UltrasonicSensor class to generate random data representing distance. This can be seen in Figure 11 below in the UltrasonicSimulator block where a private method has been added to make random distances.

For the SystemSimulator node, a LiquidLevelSensor will be composed of an UltrasonicSimulator. A DCMotorSimulator will be composed of a PWMLED. A LiquidPump will be composed of a DCMotorSimulator. The SystemSimulator will be composed of 3 of each LiquidPumps and LiquidLevelSensors.

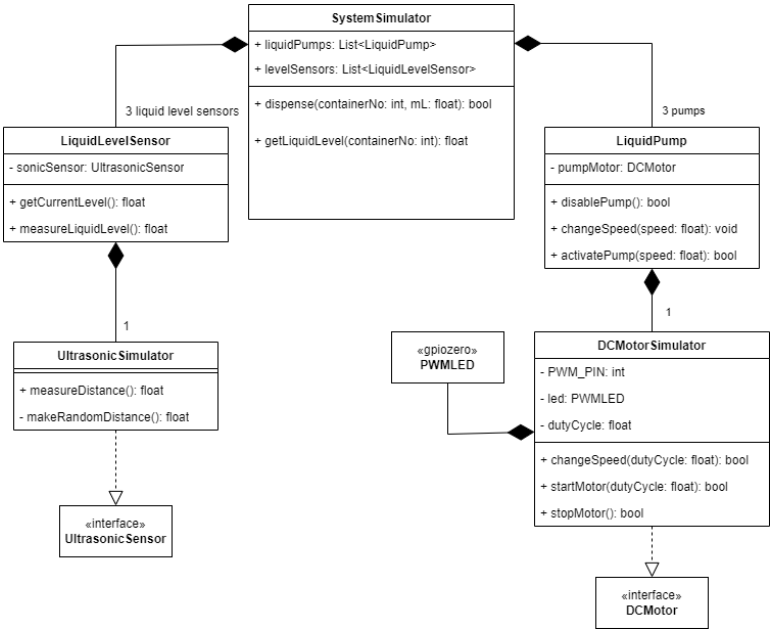

Figure 11. UML Class Diagram for the SystemSimulator Node.

3.4 Software Design for Backend Firebase Realtime Database

The cloud backend consists of a non-relational RESTful Firebase real-time database which will allow for internode communication. The database structure will resemble a tree, and each Raspberry Pi will be associated with a unique name. Messages intended for a given node will be sent under the node with the name associated with the desired destination Pi. Figure 12 below shows the bare-bones backend structure.

```
{
    "UserInterface": {
        "IOController": [],
        "SystemSimulator": []
    },
    "IOController": {
        "UserInterface": []

    },
    "SystemSimulator": {
        "UserInterface": []
    }
}
```

Figure 12. Backend Firebase Realtime Database Bare-bones Data Structure

4 Hardware Design

The hardware designs of the various components and subcomponents of the Mocktender system are discussed in the sections below.

4.1 Liquid Level Sensor

The liquid level sensors are part of the IOController subsystem and are responsible for measuring the volume of the liquids stored in the Mocktender. There will be three liquids that can be mixed at a time in the Mocktender. Each liquid will get its own liquid level sensor which is modelled below in Figure 13. The liquid level sensors will be implemented using 3 Ultrasonic Sensor (US) modules, specifically the HC-SR04 US. The HC-SR04 requires a 5V supply which can be provided using the Pi's 5V pin. The US can communicate with the Raspberry Pi using a by sending a digital GPIO pulse to the TRIG pin (see Figure 13) and then interpreting the response on the ECHO pin as a digital data stream using GPIO.

The TRIG pin accepts voltages between 3-5V as a logical "1". The GPIO pins on the Raspberry Pi can output 3.3V so the TRIG pin can connect directly to a GPIO pin. The ECHO pin however can output up to 5V. As this would damage the 3.3V GPIO pin on the Raspberry Pi, a voltage divider using 2 resistors in series will be used to drop the voltage at the GPIO pin down to 3V. The specific configuration can be seen below in Figure 13 The negative terminal of R1 and positive terminal of R2 will connect to the GPIO pin.

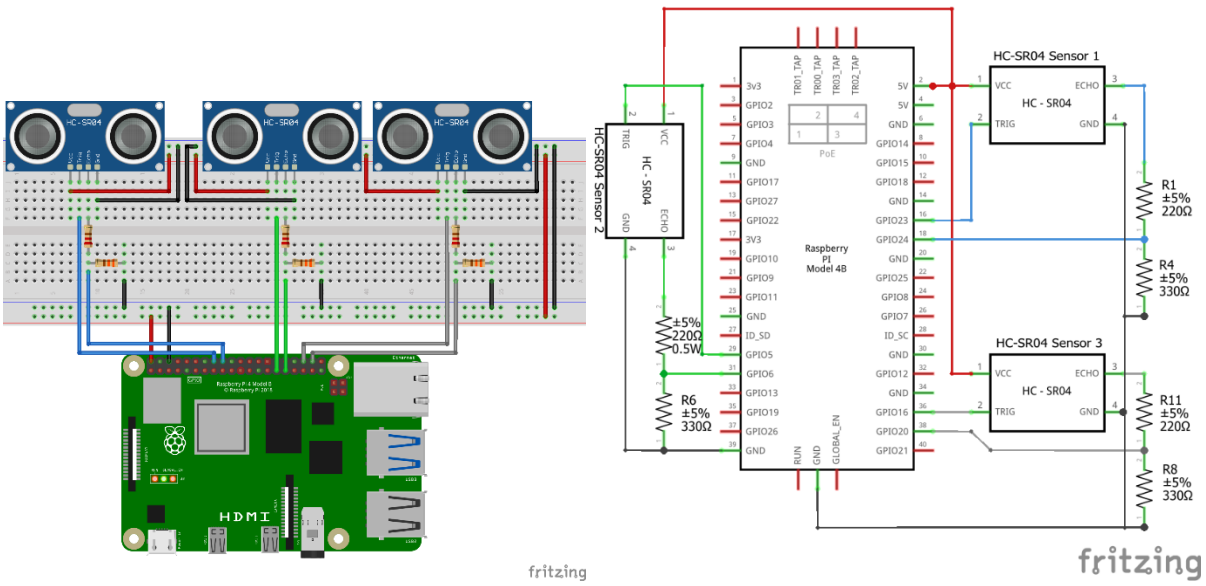

Figure 13. Breadboard and circuit schematic for Liquid Level Sensors

4.2 Drink Dispenser System

The Drink Dispenser System is a subcomponent of the IOController Subsystem. It is responsible for the controlled dispensing of the various liquids to be mixed. For the system, we will need 3 pumps which will each use a DC motor to actuate. The DC motors will use an H-Bridge module which will allow us to control the speed of the pumps using a PWM signal from the GPIO pins on the Raspberry Pi. The H-Bridge module we decided on uses the L298N Dual H-Bridge driver. The input pins on the H-Bridge driver accept voltages from 2.4-5V as logical "1" which means the 3.3V GPIO pins alone will be sufficient for controlling the motors. The H-Bridge driver can accept up to 12V to drive the motors. It also provides a stepped down 5V source which can be used to supply the Raspberry Pi with power. This can be seen below in Figure 14 as the orange wires. Since a single H-Bridge driver can only driver two motors at once, two drivers are required to meet our three liquid requirements.

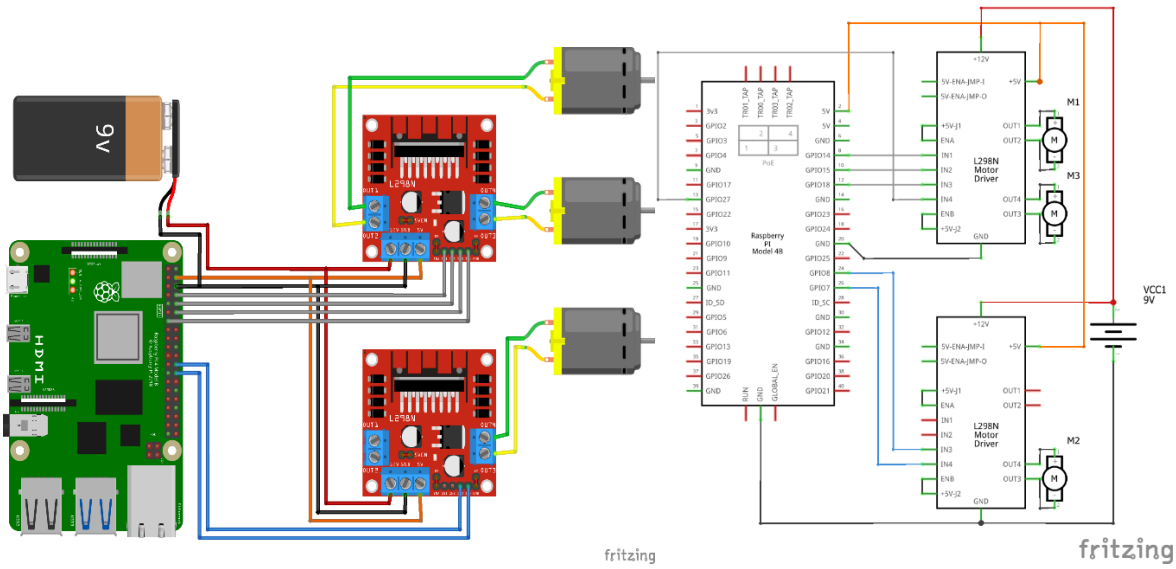

Figure 14. Drink Dispenser System component and circuit schematic

4.3 LCD Feedback System

The LCD (Liquid Crystal Display) Feedback system is a subcomponent of the IOController Subsystem. It is responsible for providing onboard visual confirmation for the active state of the system. The LCD used for the feedback system is a standard 1602 LCD, the numbers indicating that it is a 2 row LCD with each row containing 16 characters. The LCD can take 3.3V for power and for logical "1" meaning that it can be operated using the Raspberry Pi's 3.3V power source and GPIO pins. A resistor must be placed between the V_O pin on the LCD and ground to be able to read the text on screen which can be seen in the schematics in Figure 15 where a 2200Ω resistor has been placed between V_O and the GND rail. The backlight of the LCD must also be powered to read the text on screen. This too can be seen below in Figure 15 where Pin 16 (LED-) is connected to GND and Pin 15 (LED+) is connected to the 3.3V power source through a 220Ω resistor.

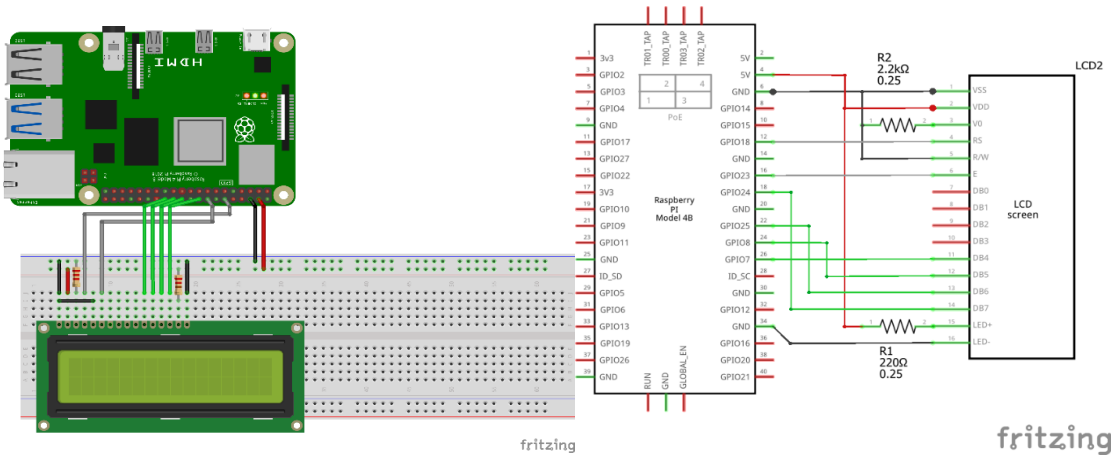

Figure 15. LCD Feedback System component and circuit schematic

4.4 IOController Simulator

To prove the scalability of the Mocktender, a Raspberry Pi device was reserved to be a shadow copy of the IOController Subsystem. It will simulate the function of the Sensor Subsystem, acting as a second machine. The Pump modules will be mimicked using LEDs, using the on state to pretend that the "pump" is dispensing the liquid and the off state to pretend that the "pump" is not dispensing anything. All LEDs have an associated "forward voltage" which is simply the voltage required to turn the LED on. This value is generally within 1.8-3.1V which is well within the 3.3V GPIO voltage. LEDs are unable to limit the current across them sufficiently while active, so series resistors (see Figure 16 below) are required to keep the current at or below 20mA to prevent damage to both the LED and GPIO pin.

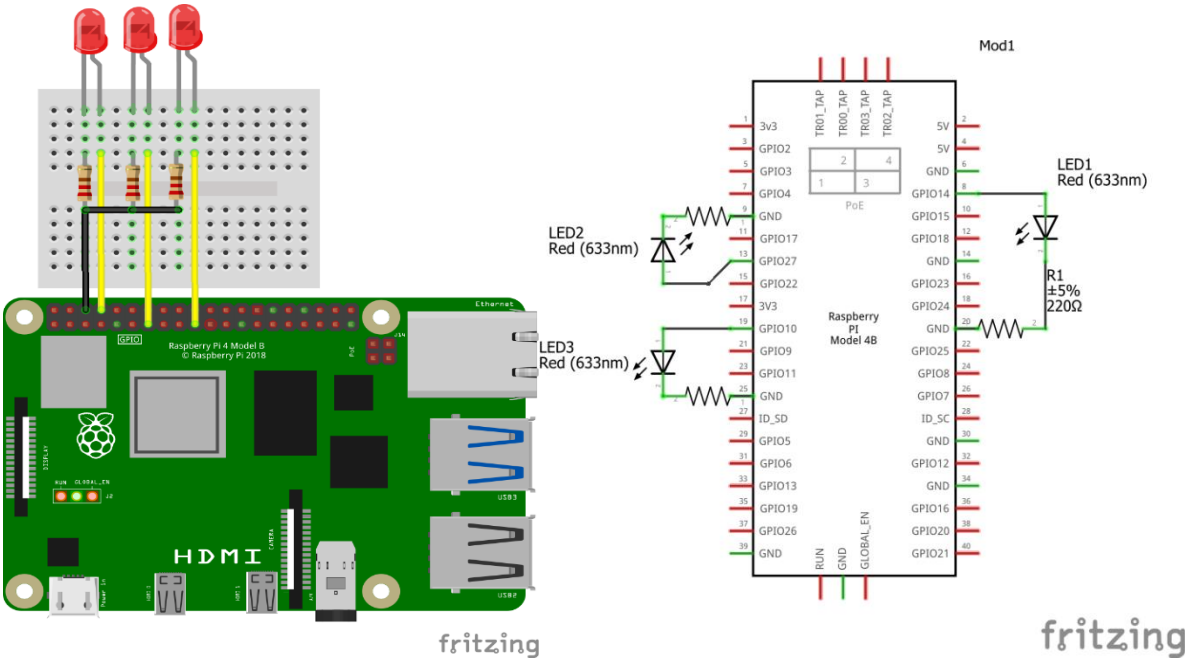

Figure 16. IOController Simulator Component and circuit schematic.

5 GUI Design

The graphical user interface of the Mocktender system serves as the user's access to the system, providing liquid level information, dispensing, and recipe creation. The GUI will be hosted on a Flask webserver that uses HTML templates to graphically render web pages that a user can interact with (web interface). There are two types of users, an administrator, and a customer, whose roles are further outlined in Section 5.1 below. To provide an overview of the functionality of the web interface, a flowchart of the web pages can be seen below. A brief overview and wireframe showcasing each page of the web interface will follow in the Sections 5.1.1 – 5.1.7. A GUI Design flow chart is shown below in Figure 17. For information on the software design including a flow chart showcasing the rendering of pages and messages sent, refer to Section 3.1 above.

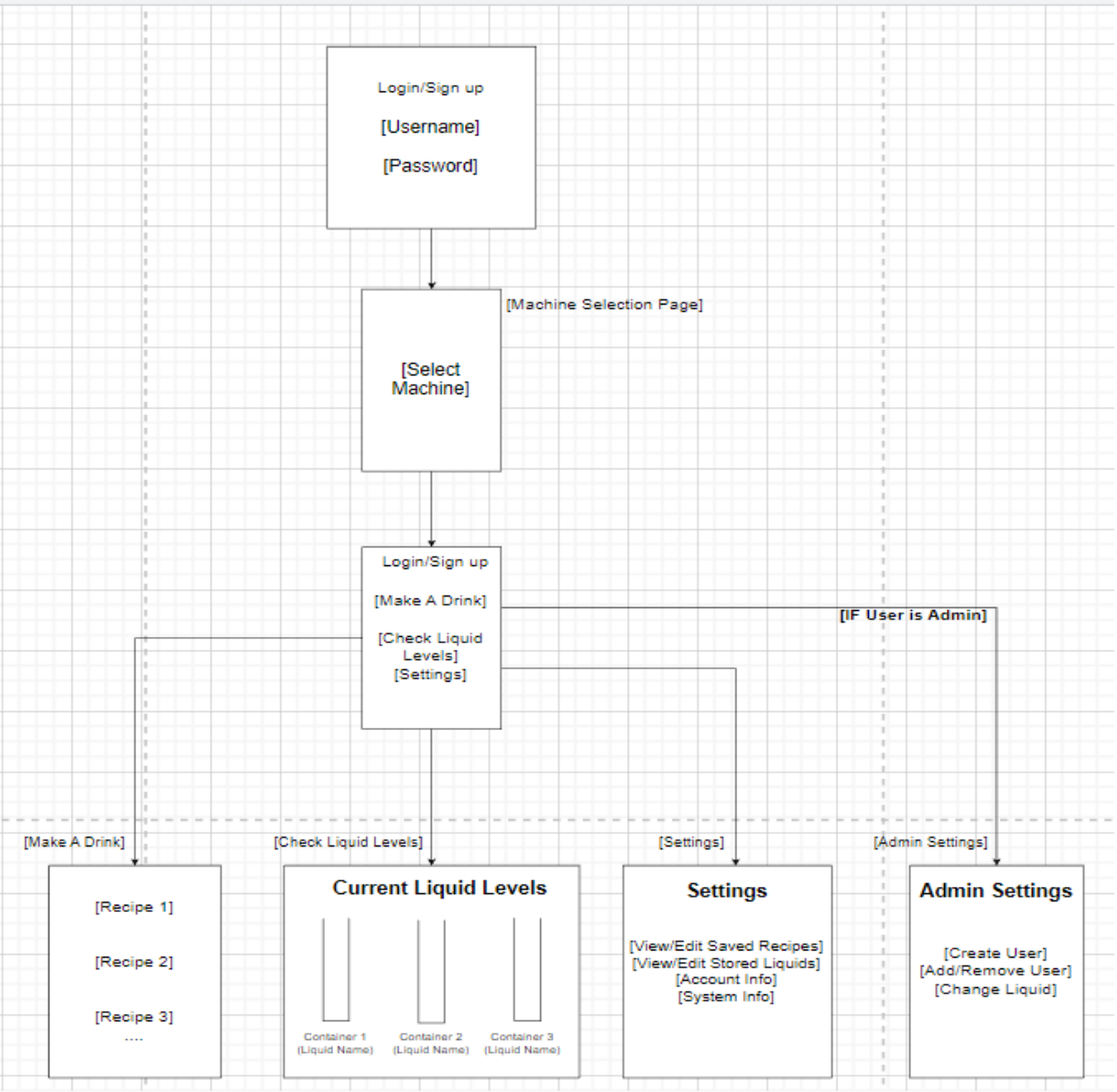

Figure 17. GUI design flow chart

5.1.1 User Login Page

When a Mocktender machine is received, an initial admin username and password will be provided. This will be used to login for the first time and access the Mocktender machine interface. The admin can then access the Admin Settings page and create new admin and customer accounts (discussed further in Section 5.2). The login page will take user input of a username and password, then send the data to the SQLite authentication database when the “login” button is pressed. If the login is accepted, the user will be redirected to the machine selection page discussed in Section 5.1.2, and if the login is denied, a message will inform the user that an incorrect username or password was entered. A wireframe showcasing the prototype design of the login page can be seen below in Figure 18.

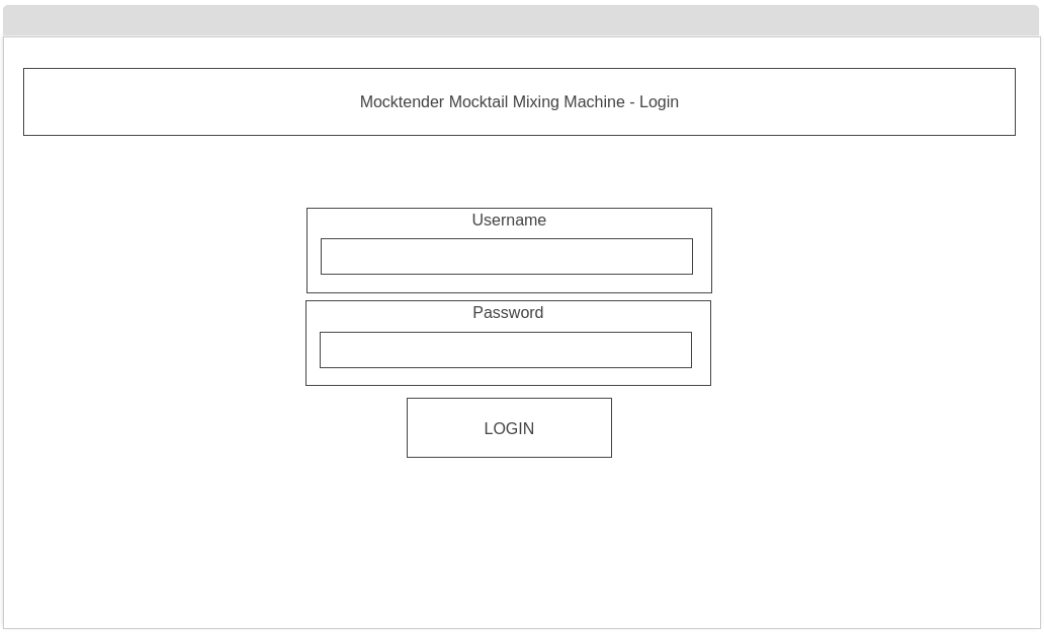

Figure 18. Wireframe of the Login page of the Web Interface.

5.1.2 Machine Selection Page

It is possible that a user is associated with multiple machines in the UserInterface system. The machine selection page will allow such a user to select which machine they would like to access. A list will be generated of the machines that the user has access to, and they will be able to select the machine to access and then press the "Select" button to access the home page of the machine (discussed in Section 5.1.3). A wireframe showcasing the prototype design of the machine selection page can be seen in Figure 19 below.

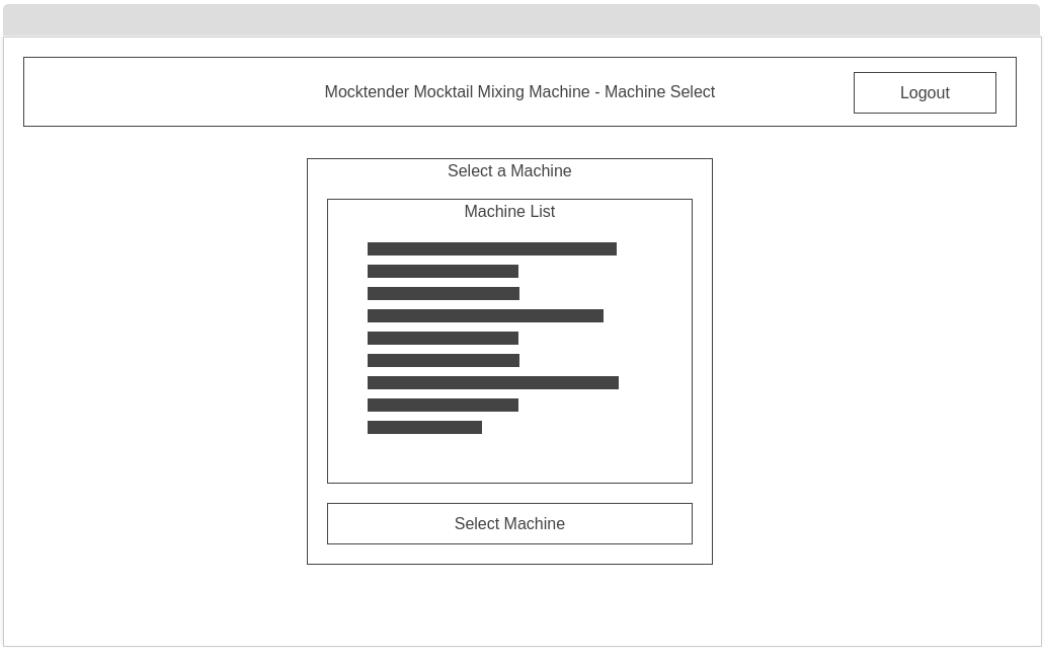

Figure 19. Wireframe of the Machine Selection Page of the Web Interface.

5.1.3 Machine Home Page

The machine home page allows for a user to select an action that they would like to complete. The action selections are, dispensing a drink, checking the liquid levels, accessing settings (recipe creation), and if they are an Admin user, accessing the admin settings. Each action option is associated with a button that will redirect the user to the page associated with that action. A wireframe showcasing the prototype design of the machine home page can be seen below in Figure 20.

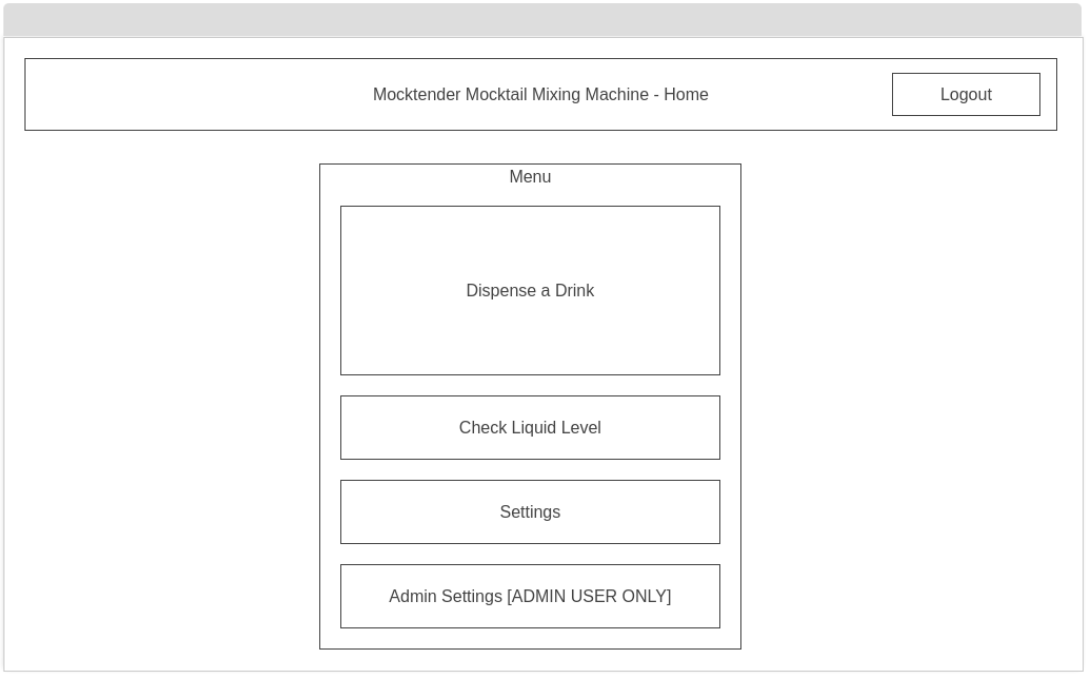

Figure 20. Wireframe of the Machine Home Page of the Web Interface.

5.1.4 Drink Dispensing Page

The drink dispensing page allows for the user to select a recipe from a list of recipes, and dispense the drink specified by that recipe. When a recipe is selected from the list, and the “Dispense Drink” button is pressed, the machine will dispense the drink. If the machine does not have the correct liquids, or not enough liquid, it will display an error message to the user and prompt them to select a different option. A wireframe showcasing the prototype design of the drink dispensing page can be seen in Figure 21 below.

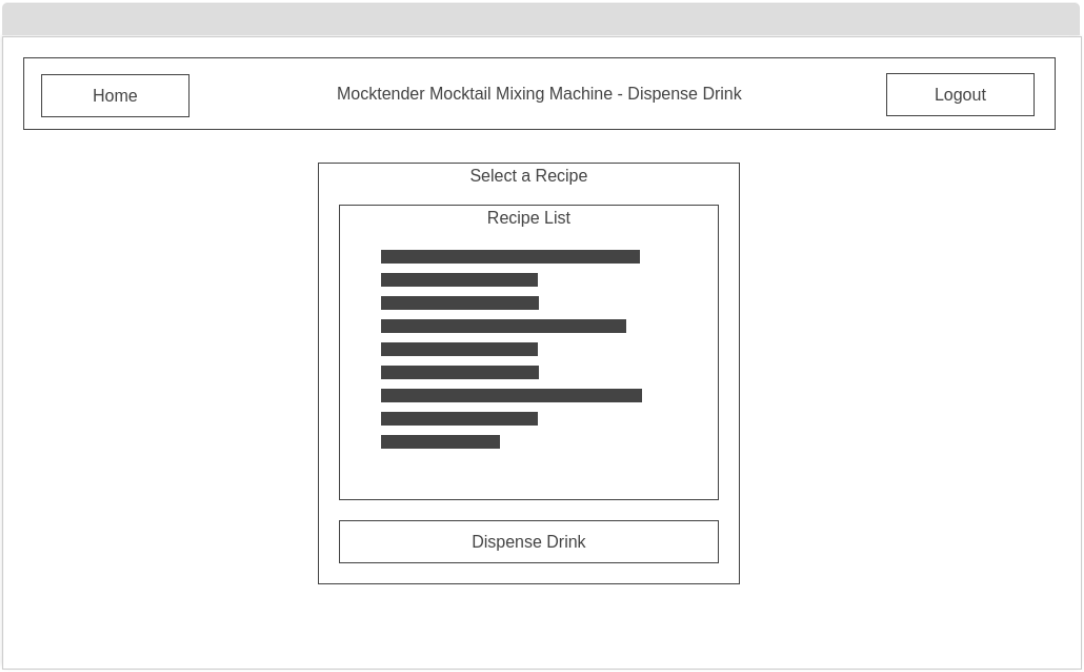

Figure 21. Wireframe of the Drink Dispensing Page of the Web Interface.

5.1.5 Liquid Level Page

The liquid level page provides an overview of the current levels of each of the liquids in the machine. The page is updated in real-time by the ultrasonic sensor system. A wireframe showcasing the prototype design of the liquid level page can be seen below in Figure 22.

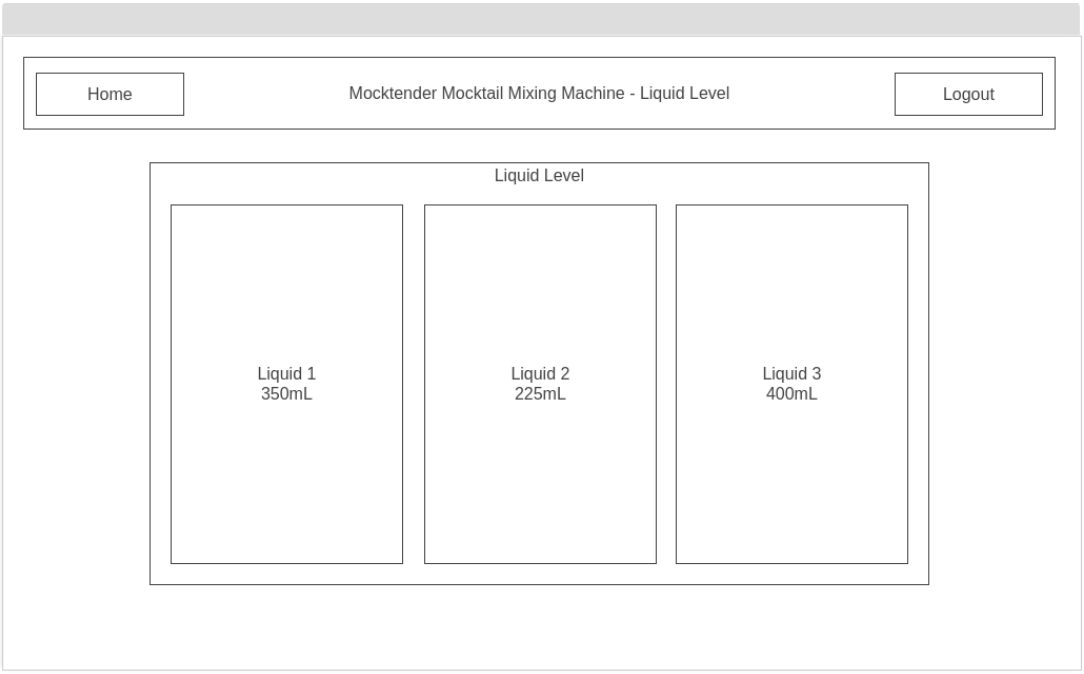

Figure 22. Wireframe of the Liquid Level Page of the Web Interface.

5.1.6 Settings Page (Recipe Creation)

The settings page allows for the user to create new recipes using the liquids currently in the machine and view the recipes currently stored in the machine. A recipe can be deleted by selecting it from the list and pressing the “Delete Recipe” button. The settings page also shows account and system information for the user. A wireframe showcasing the prototype design of the settings page can be seen below in Figure 23.

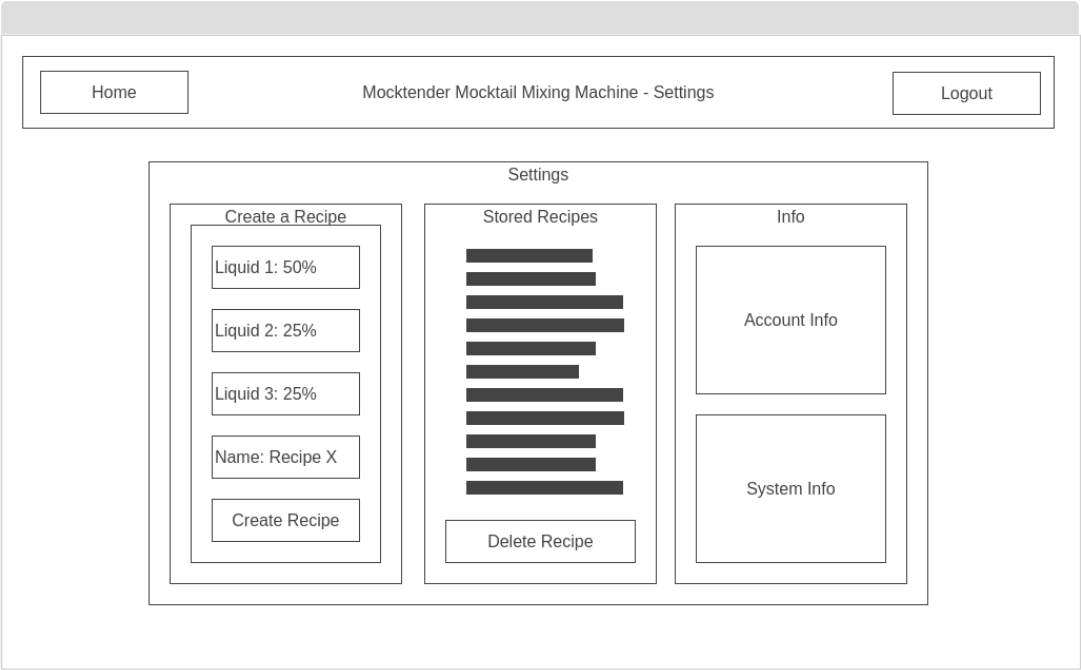

Figure 23. Wireframe of the Settings Page of the Web Interface.

5.1.7 Admin Settings Page

The admin settings page allows for an admin user (discussed in Section 5.2 below) to perform admin actions on the machine. These actions are, creating a new user, adding, or removing an existing user, changing the names of the liquids stored in the machine, and locking/unlocking the machine. To create a new user, under the Create a New User panel, the username and password of that user must be entered, and a selection of if the user is an admin or not must be made. The admin can then hit the “Create User” button to add the user to the system. The page will also display a list of all the currently registered users available in the system, and their status on the current machine under the User List panel. A user on another machine may be added, or a user currently on the machine can be removed using the “Add/Remove User” button. To liquid names can be modified in the text boxes under the Liquid Settings panel and will be saved when the user pressed the “Save” button. The “Lock/Unlock Machine” button in the Machine Admin panel allows for the admin to lock or unlock the machine. A wireframe showcasing the prototype design of the admin settings page can be seen in Figure 24 below.

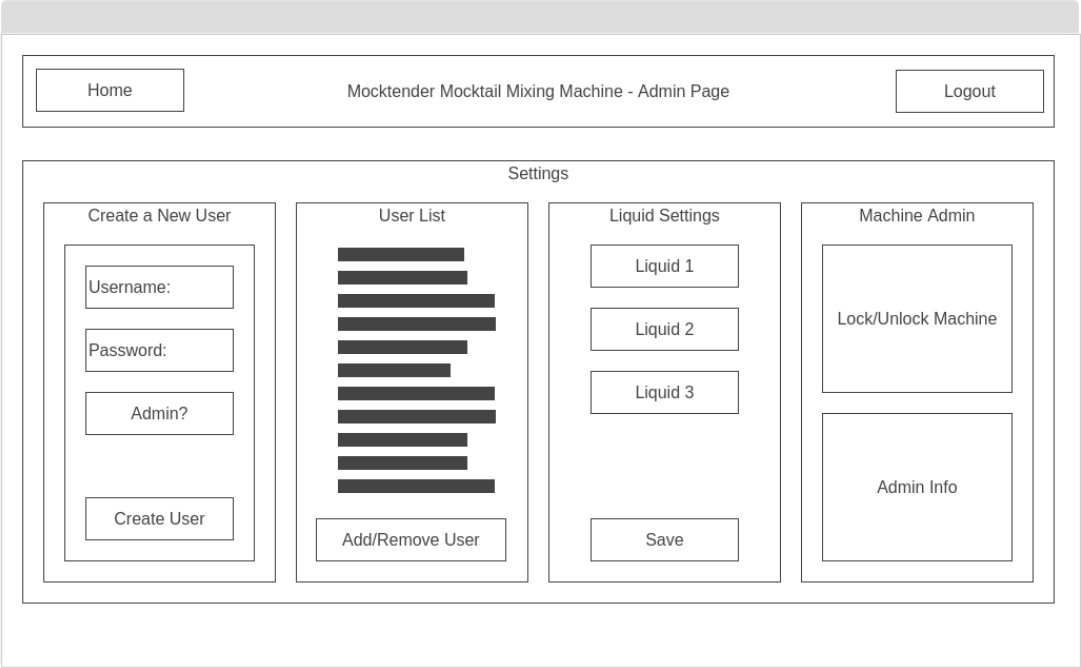

Figure 24. Wireframe for the Admin Settings Page of the Web Interface.

5.2 Table of Users/Roles

The Mocktender system has two types of users that will interact with the machine. An administrator user who owns the machine, and a customer that has permission to use the machine. The descriptions are provided below in Table 5.1.

Table 5.2. Users and Roles in the Mocktender system.

User/Role	Description	Usage/Access
Administrator (Owner)	The owner of the Mocktender device. Can own 1 or more Mocktender devices that will be associated with their account in the database. For our project, one admin account will be associated with both the physical device and the simulation device.	The administrator has access to the admin settings page of the web interface in addition to all the other pages a user has access to. They can lock/unlock the machine, create or add/remove users, change liquids, as well as use the machine like a normal user.
Customer (Regular User)	A user of the Mocktender device that has an account authorized by the administrator to use the machine. An administrator can also be a consumer of the machine.	The user can access all pages of the web interface including, dispensing, liquid level, and the settings page of the user interface. If authorized on the device, a user can dispense the recipe of their choice, create a new recipe, or view the current liquids and their levels. If the machine is locked by the administrator,

		the user will be unable to use the remote dispensing functionality until it is reenabled.

6 Test Plans

The test plans for the Mocktender system for end-to-end communication, unit testing, and the final demo are outlined in the sections below.

6.1 End-to-end Communication Demo Test Plan

The Mocktender machine consists of multiple Raspberry Pi devices that need to communicate amongst each other to meet our project goals. The system is distributed between the Pi devices with one device hosting the front-end web interface for viewing the state of the machine and controlling the hardware devices. To facilitate node to node communication, a Firebase Realtime Database is used. The User Interface device is responsible for updating its local SQLite database with the liquid level sensor data received from the Firebase and sending instructions to the hardware devices. The IOController and SystemSimulator nodes receive and act on instructions from the User Interface node, and send liquid level data to the UserInterface, both through the Firebase.

To test the end-to-end communication between the components, all the messages discussed in Section 2.1 will need to be created. The test scripts created will generate the messages, send them to the correct end destination, and the end destination will confirm that the message was received, and the contents are as expected. The test will pass if the message is the same as was expected, and will fail if either no message was received, or the contents are different than was expected. To provide test feedback, the messages will be printed to a console that will also be available on the Raspberry Pi devices whenever needed as a debugging option. The database contents on the Firebase real time database will also be tested after the messages are sent to ensure that the contents are as expected.

The following steps will be performed in our end-to-end testing demonstration:

1. First, the connections between the Raspberry Pi devices, and the Firebase real time database will be tested. A test script will verify that all the Pi devices are able to connect to the Firebase server.

2. Next, messages will be sent from the User Interface Pi device to the Sensor System and Simulation System to test that messages to control the motors can be received and acted on. These messages will go through the Firebase database.

3. The Sensor and Simulation Systems will then send messages to the User Interface Pi device through the Firebase to acknowledge that the message was successfully received, and the motors were run (drink was dispensed).

4. Next, sending the sensor data from the Sensor and Simulation Systems will be tested by using a test script to create mock sensor data to send to the User Interface System through the Firebase server.

5. The User Interface system will take the data from the Firebase cloud database and update its local SQLite database. The database contents will then be printed by the test script to showcase that they have been updated correctly.

6. Finally, the Flask web server sending messages will be tested by creating a demo Flask web page with a button that sends instructions to the IOController to pump a test recipe.

6.2 Unit Test Demo Test Plan

To test that each individual hardware and software component works as expected, the following hardware and software unit tests will be performed.

6.2.1 Hardware Testing

Table 6.2.1. Hardware device testing plan

Hardware Device	Tester	Device Description	Test Description	Expected Test Results
Ultrasonic Sensor (Liquid level sensing)	Matt	The ultrasonic sensor will be used to track the water level in the 3 liquid tanks in the Mocktender system. The distance from the sensor to the surface of the water is used to calculate the current volume of the liquid in the container.	To test the liquid level system is working correct, measured amounts of water will be added and removed from the liquid containers.	The distance from the sensor to the surface of the water should match the expected level based on the amount of water added or removed.
Peristaltic Pump & DC Motor	Ethan	The liquid pump will be used to dispense a drink. Based on the recipe provided by the user, each pump will be run for a specified amount of time so that the correct amount of each liquid is pumped (mixed) into the dispensed drink.	To test the functionality of the pump, the GPIO pin connected to the motor controller will be pulsed with PWM signals. The PWM duty cycle will gradually increase to 100% over 5 seconds, maintain that 100% duty cycle for another 2 seconds, and then decrease to 0% duty cycle over 5 seconds.	The user should see the motor gradually speed up for 5 seconds until it reaches its max RPM, maintain that max RPM for 2 seconds, and then see the motor gradually slow down for 5 seconds until it is no longer active.
LCD Display	Duncan	The LCD display will display machine	To test the LCD output, messages	The LCD display should display

		operation information such as what it is dispensing and any errors.	will be displayed on the LCD.	the correct messages.
Simulation LEDs	Duncan	The simulation LEDs will light up when the motors on the simulation device are ON. Based on the recipe provided by the user, each LED will turn on for a specified amount of time so that it can be demonstrated that the simulation is working as expected.	To test that the Simulation LEDs accurately turn ON when the motors should be running, the LEDs will be pulsed with PWM similarly to the Peristaltic Pump & DC Motor test. This is since the simulation LEDs are simulating the pump and motor system	Similar to the pump and motor test, the user should see the LEDs gradually get brighter as the PWM is increased until they reach their max brightness. They will maintain the maximum brightness for 2 seconds before gradually slowing down until they are OFF.

6.2.2 Software testing

Table 6.2.2 Software testing plan

Software	Tester	Software Description	Test Description	Expected Test Results
Liquid Level Monitoring System	Matt	The liquid level monitoring system will take the sensor data and calculate the volume of liquid currently in the container in real time. It sends this information to the user interface to display to the user	To test the conversion from distance to liquid level, a simulated liquid level will be provided to the software and tested that the correct liquid level is output for sending to the user interface.	The calculated liquid level from the software should match the calculated liquid level when calculated by hand using the simulated data.
Motor Control System	Ethan	The motor control system takes the amount of liquid needed for the selected recipe and converts it into the	To test that the motor control system can convert the recipe into the correct PWM outputs for each of the motors, a sample	The time that the system “turns on the motor” should match the expected time

		time that the motor needs to run based on the calculated flow rate.	recipe will be provided to the motor software, and it will be tested that it outputs the correct motor data.	based on the recipe and hand calculated flow rate.
Frontend Web Interface	Matt	The frontend web interface allows a user to view liquid levels, create recipes, and dispense a drink. It also allows an admin login to give administrative access to locking a machine and	The interface will be tested by rendering the components discussed in Section 5 and asserting that the outputs messages sent when buttons are pressed are as expected. A simulated liquid level will be sent to the interface and should display as corrected.	When a button is pressed, it should generate the correct message to send to the system. The liquid level should be received and displayed in the component correctly.
Backend Firebase Database	Duncan	The backend firebase database is used for internode communication in the Mocktender system.	The database will be tested using a script which generates messages to be sent over the database.	The database will be verified by pulling the stored information and testing that the information stored in the database is the same as the expected information to be stored in the database from the test script.
Local SQLite Database	Duncan	The local SQLite database is used to store machine specific data for display on the web interface.	The database will be tested using a test script which creates the expected tables outlined in Section 2.4.	The data stored in the SQLite database after the script is run will be pulled and compared with the expected contents of the database.

6.3 Final Demo Test Plan

To validate the finished Mocktender system, the following test plan will be used to show that the system functions as expected, and the requirements are met.

6.3.1 Automatic Liquid Mixing

The system must be able to automatically mix liquids as specified by the user. The correct amount of each liquid must be dispensed into the cup so that the drink is as expected. This will be tested by providing recipes with known liquid amounts and ensuring that the correct amount of each liquid was mixed and dispensed into the cup.

6.3.2 Administrative Access

The administrative access requirement of the system ensures ease of control of the system for use in a public place, and in a household. The administrator user can access the web interface and add new users, as well as lock the machine. To test this, an admin user will login, add a new user, that user will login and show that they can use the machine with the correct permissions. The admin user will also test the ability to lock the machine from non-admin users to be able to login.

6.3.3 Remote Dispensing Access

Remote dispensing access allows for a user to control the machine from the web interface to dispense their drink of choice. To test this, the web interface will be used to dispense a drink.

6.3.4 Recipe Creation and Storage

The recipe creation and storage requirement of the system allows for a user to create a new recipe and select a recipe from the database stored in the system. To test the recipe creation and storage, a recipe will be created by a user, and the database will be pulled to ensure that the recipe is present.

6.3.5 Remote Liquid Level Monitoring

The remote liquid level monitoring requirement of the system allows for a user to view the current liquid levels of the system in real time and for the machine to ensure that the selected recipe can be dispensed. To test this objective, a measured amount of liquid will be poured into an empty liquid container, and a liquid container with a known amount of liquid. The web interface should reflect that the measured amount was added, and if a recipe with too much of that liquid is selected, it should be denied.

7 Project Update

The Mocktender project has started slowly, however we have planned to mitigate the delays and get the project back on track for completion as scheduled. Our team had struggles completing the simulation system for the end-to-end demo, but we met as a group and discussed how to get back on track before the unit testing demo in two weeks.

Looking back at our proposal, most of our milestones still make sense and we are following them to continue work on our project. We are a bit behind schedule due to not receiving the hardware parts ordered from Digi-Key until March 10th, so the hardware components of milestones 3 and 4 had to be moved back. Work towards the software components of milestones 3 and 4 have gone well and are near completion. Once we receive the hardware devices, we will work to mitigate the delays to our schedule and keep the project on schedule.

The updated project milestones can be seen below in Section 7.1, and the updated Gantt chart showcasing our project timeline can be seen below in Section 7.2.

7.1 Project Milestones

The Mocktender system will continue to be developed following the milestones shown in the table below. Table 7.1 below was updated from the proposal to reflect updates discussed above in Section 7.

Table 7.1 Milestones for the Mocktender System

Milestone #	Name	Description	Date
1	Uploading and Fetching Data	Computer devices can read from and write data to a real-time database independently of each other, which can then be accessed and displayed on an external device.	February 21st **[Complete February 28th]**
2	Communication Between Nodes	Communication established between computer devices (HTTP etc.) to transfer data stored locally between nodes and issue commands.	February 28th **[Complete March 10th]**
3	Measurement and Conditional Actuation	Liquid level of a fixed container can be measured accurately using sensors can uploaded to the database. Data can be used to turn on an LED when a threshold is reached.	March 3rd **[All Hardware Moved to March 17th]**
4	Exposing Data to the World	The liquid level of the fixed container will be displayed as text on a Flask web server, exposing the data to the world. The simulation system will display three liquid levels at once.	March 7th **[All Hardware Moved to March 17th]**
5	Hardware Assembly	Assembled liquid dispensing system utilizing peristaltic pumps. Liquid level on webserver should be updated (as text) in real-time with data from the Ultrasonic sensor(s).	March 14th **[Moved to March 17th]**
6	Graphical User Interface	Webserver displays the volume of the liquid containers as an animation. Users can navigate between different webpages on the server.	March 21st
7	Controlling the Hardware System with UI	Webserver component can control all 3 liquid pumps implementing the drink mixing. Users can login and save customs mixtures locally and in the real-time database.	March 28th

7.2 Schedule of Activities

Our updated Project Gantt Chart can be seen below in Figure 25. Green indicates completed items, yellow indicates items in progress, and red indicates items not yet started.

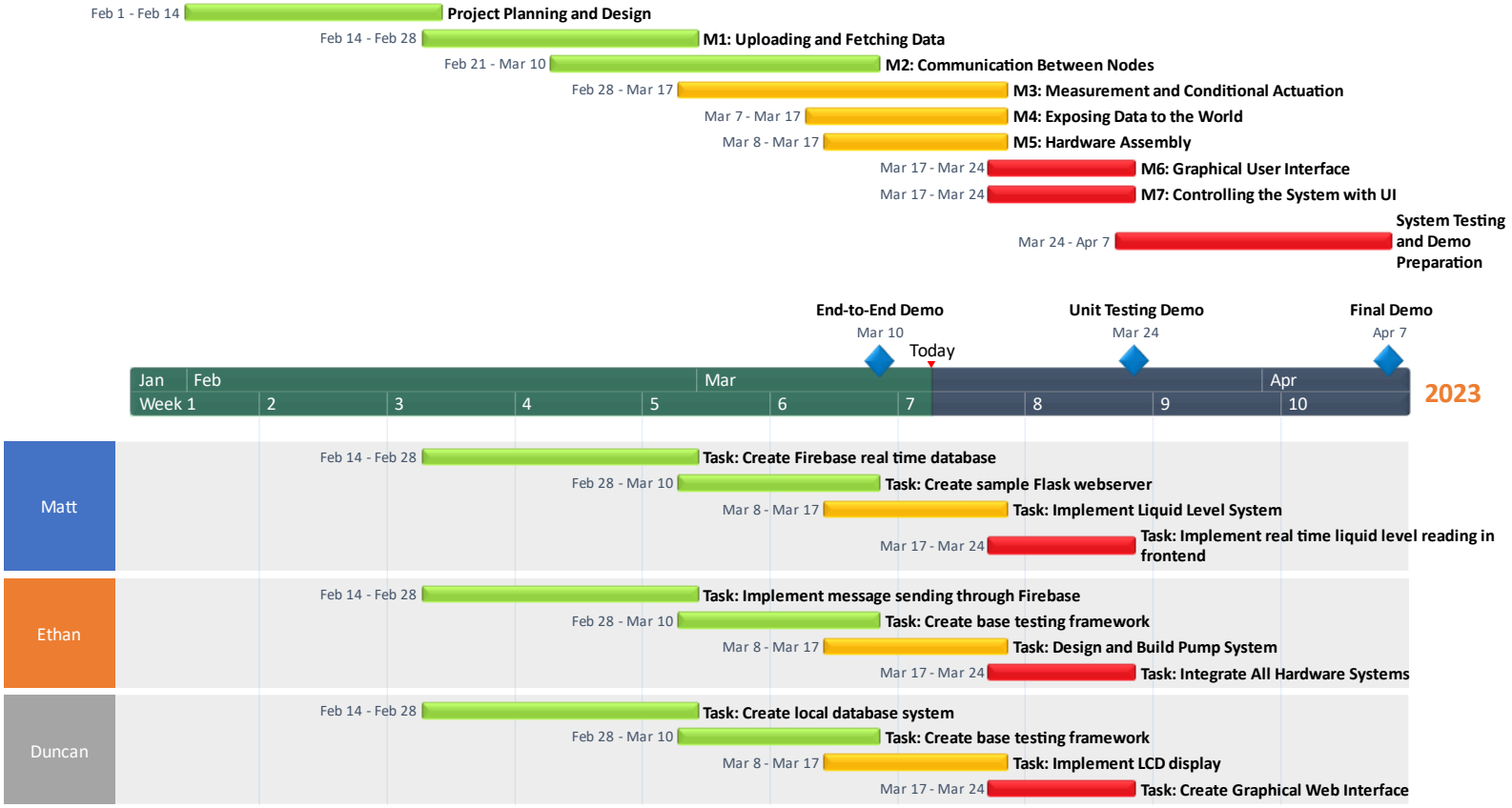

Figure 25. Updated Gantt Chart for the Mocktender project

References

[1] Colourful drinks, abstract, new, HD wallpaper. Peakpx. (n.d.). Retrieved February 2023, from https://www.peakpx.com/en/hd-wallpaper-desktop-gfuhh.

[2] Cox, K. (2021, October 12). NielsenIQ: Newly Released Low & Non-alcoholic beverage trends. Wine Industry Advisor. Retrieved February 2023, from https://wineindustryadvisor.com/2021/10/12/nielseniq-newly-released-beverage-trends.

[3] Mocktails are gaining popularity. will meetings and events catch the trend? Meetings Today. (n.d.). Retrieved February 2023, from https://www.meetingstoday.com/articles/143402/mocktails-gaining-popularity-will-meetings-events-catch-trend.

[4] Turcotte, S. (2023, January 12). Here's why alcohol-free drinks are gaining in popularity. CTV Kitchener. Retrieved February 2023, from https://kitchener.ctvnews.ca/here-s-why-alcohol-free-drinks-are-gaining-in-popularity-1.6227268.