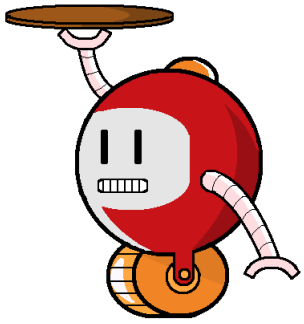

LazyBots

McMaster University

Design Document

SE 4GA6 & TRON 4TB6

Group 9

Karim Guirguis	001307668
David Hemms	001309228
Marko Laban	001300989
Curtis Milo	001305877
Keyur Patel	001311559
Alexandra Rahman	001305735

Table of Contents

List of Tables

List of Figures

1 Revisions

Date	Revision Number	Authors	Comments
December 24th, 2017	Revision 0	Karim Guirguis David Hemms Marko Laban Curtis Milo Keyur Patel Alexandra Rahman	-
March 12th, 2018	Revision 1	Karim Guirguis David Hemms Marko Laban Curtis Milo Keyur Patel Alexandra Rahman	Added the section to include the assumptions of the project as well as added some traceability between the components and the requirements. Finally, included the components that were likely to change.
March 16th, 2018	Revision 2	Karim Guirguis David Hemms Marko Laban Curtis Milo Keyur Patel Alexandra Rahman	Combined System Design Document and Component Design Document and updated the MIS, MID and Design Notes.

Table 1: LazyBots Table of Revisions

2 Purpose

The purpose of this project will be to create an autonomous robot that will navigate to and serve the requested drink to the user who request a drink. Currently in an office setting, workers must leave their offices to get their own drinks. In restaurants, drinks are served by waiters and waitresses, which hinders them from doing other work at that time. Alfred will be designed to make the serving drinks autonomous.

Alfred will allow users to request drinks. These requests will form a queue which Alfred will serve in order using a FIFO protocol. Alfred will go to the table of each user and pour the drinks ordered from that table. Alfred will also have an administrator user which will be able to call Alfred back and override any action that is being taken at the time.

The following document will outline the overall system components, as well as the overall system behaviour, operation and undesired error handling.

2.1 Scope

The system implemented is one that is meant to automate the dispensing of beverages to customers within a restaurant at the respective customers' table. The customer will be able to order a drink from their table which will be relayed to Alfred, where he will arrive at their table and dispense the requested drinks. The staff will be able to request Alfred to come back for charging and refilling when desired.

2.2 Context Diagram

Figure 1 is a context diagram of the drink serving robot, Alfred.

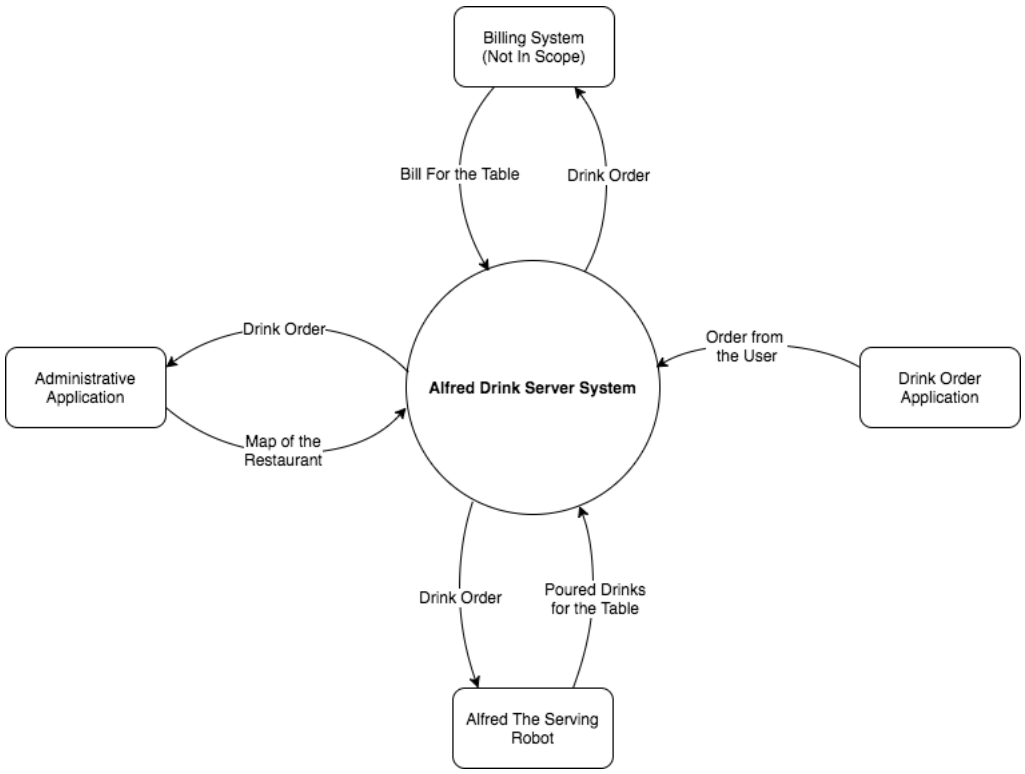

Figure 1: Drink Serving Robot Context Diagram

2.3 Diagram of Components

Figure 2 is a diagram that shows the interaction of components of the drink serving robot system.

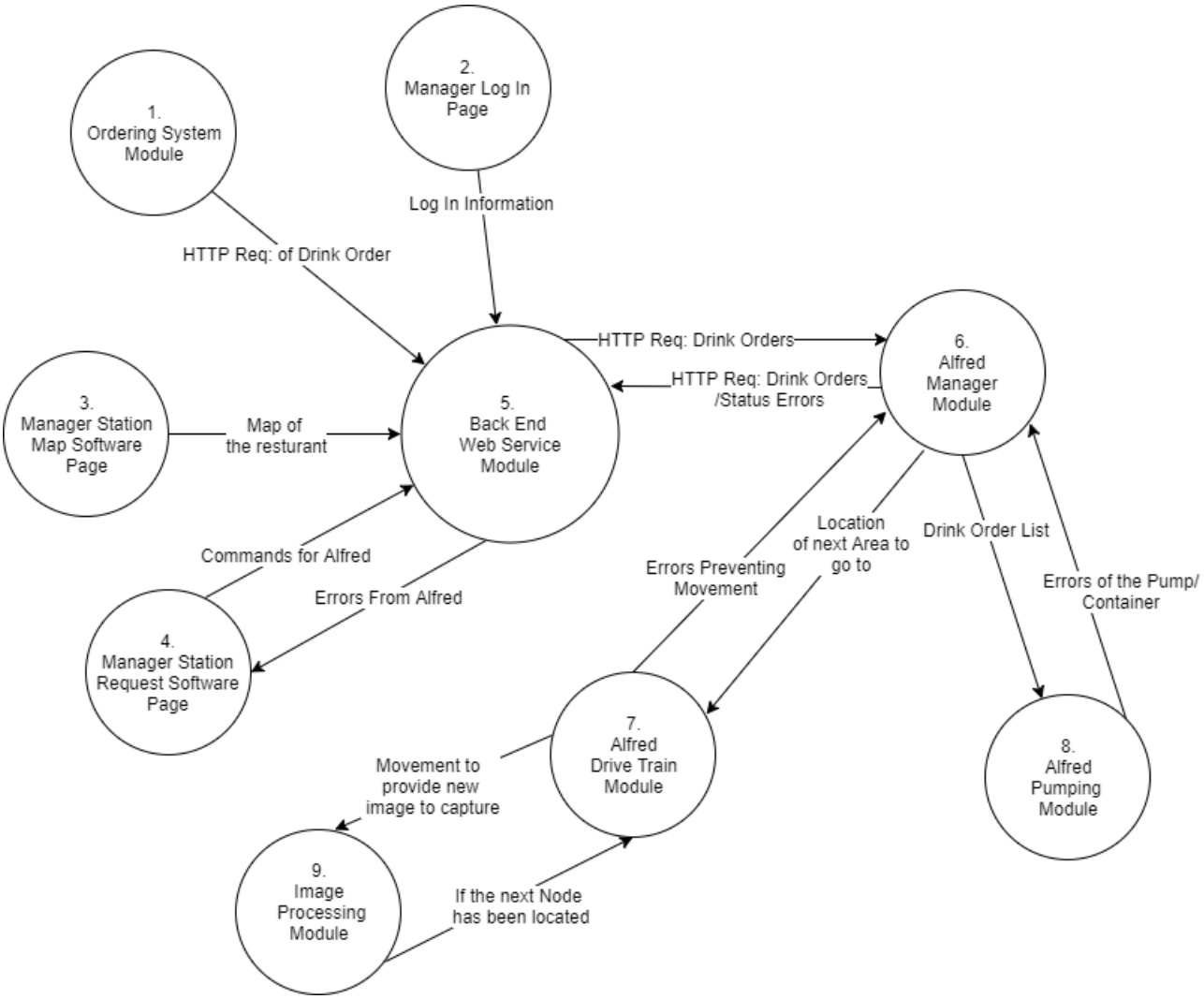

Figure 2: Drink Serving Robot System Component Diagram

2.4 Assumptions

Alfred's assumption are represented in the tables below.

A1	The environment will only be comprised of a one story building with no steps.
Rationale	Different environment elevations are beyond the scope of the project. Alfred will not be able to navigate around steps.

A2	The ground within the establishment will be smooth without defect and with moderate friction.
Rationale	Imperfect floors with too much or too little friction are beyond the scope of the project. Alfred will not be able to navigate properly with imperfect floors and the wheels will slip with too little friction.

A3	The establishment will be well light with a drop ceiling of a maximum height of 9 feet.
Rationale	Poor lighting and different ceiling heights are beyond the scope of the project. A level ceiling is needed for node placement as well as proper lighting will be needed so that Alfred will be able to read the nodes.

A4	The establishment will be able to support a strong and stable Wi-Fi connection (internet connection).
Rationale	Poor internet connections are beyond the scope of the project. To support communication between all components a strong and stable internet connection is needed.

A5	The width of the walkways will be wide enough to accommodate all people.
Rationale	If a table is not accessible to a human, it will not be accessible for Alfred.

A6	Orders will be placed via an Android or iOS application.
Rationale	Eliminates the need for human interactions, making Alfred completely autonomous.

A7	The height of a table will not exceed 30".
Rationale	This will help simplify the scope of the project and reduce the customers' discomfort when reaching for a drink.

A8	The serving size of a medium sized cup will not vary largely in terms of ounces.
Rationale	The standard ounces in a cup will be restricted to 12oz. to accommodate as many users as possible and to limit the scope.

2.5 Components to Requirements

3 System Variables

3.1 Monitored and Controlled Variables

The following is a list of variables that will be monitored.

Monitor Name	Monitor Type	Range	Units	Comment(s)
$w_{wheel_{left}}$	Speed	[0, 100]	rad/s	Left Wheel Speed
$w_{wheel_{right}}$	Speed	[0, 100]	rad/s	Right Wheel Speed
$m_{container}$	Mass	[0, 1.0]	Kg	Weight of the storage device
m_{drink}	Mass	[0, 1.0]	Kg	Weight of the drink
$b_{cuptaken}$	Boolean	[0,1]	N/A	If the cup has been taken
$d_{objects}$	Distance[]	[0, 10.0]	m	Set of distances to closest obstacle
V_{batt}	Voltage	[0, 20.0]	m	Voltage levels of batteries

The following is a list of variables that will be controlled.

Controlled Name	Controlled Type	Range	Units	Comment(s)
w_{motor}	Speed	[0, 100]	rad/s	Motor Speed
$percent_{dutycycle_{left}}$	Percent	[0,1.0]	%	The duty cycle of the left side of the drive-train
$percent_{dutycycle_{right}}$	Percent	[0, 1.0]	%	The duty cycle of the right side of the drive-train
V_{pump}	Voltage	[0, 5.0]	m	Voltage going to the liquid pump
$errors_{drivetrain}$	Unsigned Byte	[0, 2^8]	N/A	Errors from drive-train
$errors_{pump}$	Unsigned Byte	[0, 2^8]	N/A	Errors from pumping system module
$errors_{Alfred}$	Unsigned Byte	[0, 2^8]	N/A	Errors from Alfred
$LED_{drinksignal}$	Boolean	[0, 1]	N/A	Signal of drink that it is ready to be picked up
Q_{pump}	Flow Rate	[0, 100]	(m^3/s)	Flow rate of the pump

3.2 Constants

The following is a list of system constants.

Constant Name	Constant Type	Value	Units	Comment(s)
$V_{battMin}$	Voltage	9.0	m	The minimum voltage necessary for drive-train movement
$m_{DrinkMin}$	Mass	0.35	Kg	The minimum weight of the drink to be considered as ready for the customer.
$t_{timeout}$	Time	30	s	The maximum time Alfred or the ordering application will wait for a message from the server before timing out.
$t_{pumptimeout}$	Time	5	s	The maximum time Alfred will try to pump without noticing a change in the weight of the tank.
$t_{maxpump}$	Time	10	s	The maximum time Alfred will try to dispense a drink within.
Q_{pump}	Flow Rate	TBD	(m^3/s)	Flow rate of the pump
$freq_{bauderate}$	Frequency	9600	(Herts)	The rate at which UART communication will be performed at.
$obstruction_{timeout}$	Time	10	s	The amount of time that Alfred will stop movement due to an object in its path before ruling that there is an obstruction and throwing an error.

4 Behaviour Overview

1. **Ordering System Module**: Provided the order information, this module will communicate with the server to "order a drink".
2. **Manager Login Page**: Given the login credentials, will authenticate administrator of the system with the server.
3. **Manager Station Map Software Page**: Will allow administrator to create or modify the map of the area where Alfred will deliver drinks. This map will then be sent and stored on the server.
4. **Manager Station Request Software Page**: Will allow administrator to execute commands for Alfred, as well as view incoming error codes from Alfred. Said commands, will be sent to the server, which will then communicate with Alfred.

5. **Back End Web Service Module**: Will route communication to different components of the system. Will also be responsible for authentication and management of queue.
6. **Alfred Manager Module**: Endpoint for communication with Alfred. Will manage communication with server, as well as send any errors that Alfred is experiencing.
7. **Alfred Drive Train Module**: Responsible for driving and managing the motors based on desired route. Will also be sending errors preventing movement to Alfred Manager Module.
8. **Alfred Pumping Module**: Will control pumping system in regards of when to pour, how long, rate of dispensing, etc. Will also be sending errors pertaining to the pump or container to Alfred Manager Module.
9. **Image Processing Module**: Will detect any obstacles in the way as well as locate incoming nodes. Will communicate with Alfred Drive Train Module, to determine wether any required action based on results.

5 Component Traceability

Component Module:	**Functional and Non-Functional Requirement:**
Ordering System Module	Table Ordering Functional Requirement 1
	Table Ordering Functional Requirement 2
	Non Functional Requirement 4
	Non Functional Requirement 5
	Non Functional Requirement 8
	Non Functional Requirement 9
	Non Functional Requirement 15
	Non Functional Requirement 30
	Non Functional Requirement 32

Table 13: Ordering System Traceability

Component Module:	**Functional and Non-Functional Requirement:**
Manager Login Page Module	Non Functional Requirement 4
	Non Functional Requirement 5
	Non Functional Requirement 8
	Non Functional Requirement 9
	Non Functional Requirement 30

Table 14: Manager Login Page Traceability

Component Module:	Functional and Non-Functional Requirement:
Manager Station Map Software Module	Administrator Functional Requirement 1
	Administrator Functional Requirement 2
	Non Functional Requirement 4
	Non Functional Requirement 5
	Non Functional Requirement 8
	Non Functional Requirement 9
	Non Functional Requirement 29
	Non Functional Requirement 30
	Non Functional Requirement 33

Table 15: Manager Station Map Traceability

Component Module:	Functional and Non-Functional Requirement:
Manager Station Request Software Module	Administrator Functional Requirement 3
	Administrator Functional Requirement 4
	Administrator Functional Requirement 5
	Non Functional Requirement 4
	Non Functional Requirement 5
	Non Functional Requirement 8
	Non Functional Requirement 9
	Non Functional Requirement 29
	Non Functional Requirement 30
	Non Functional Requirement 31
	Non Functional Requirement 33

Table 16: Manager Station Request Traceability

Component Module:	Functional and Non-Functional Requirement:
Back End Web Service Module	Alfred Functional Requirement 1
	Alfred Functional Requirement 2
	Alfred Functional Requirement 4
	Alfred Functional Requirement 6
	Alfred Functional Requirement 7
	Alfred Functional Requirement 8
	Alfred Functional Requirement 9
	Alfred Functional Requirement 10
	Table Ordering Functional Requirement 1
	Table Ordering Functional Requirement 2
	Non Functional Requirement 26
	Non Functional Requirement 27
	Non Functional Requirement 31
	Non Functional Requirement 32
	Non Functional Requirement 33

Table 17: Back End Web Service Traceability

Component Module:	Functional and Non-Functional Requirement:
Alfred Manager Module	Alfred Functional Requirement 3
	Alfred Functional Requirement 7
	Alfred Functional Requirement 8
	Alfred Functional Requirement 9
	Alfred Functional Requirement 10
	Alfred Functional Requirement 11
	Alfred Functional Requirement 12
	Alfred Functional Requirement 13
	Alfred Functional Requirement 14
	Non Functional Requirement 11
	Non Functional Requirement 16
	Non Functional Requirement 17
	Non Functional Requirement 19
	Non Functional Requirement 20
	Non Functional Requirement 21
	Non Functional Requirement 26
	Non Functional Requirement 27
	Non Functional Requirement 31

Table 18: Alfred Manager Traceability

Component Module:	Functional and Non-Functional Requirement:
Alfred Drive-Train Module	Alfred Functional Requirement 3
	Alfred Functional Requirement 12
	Non Functional Requirement 1
	Non Functional Requirement 2
	Non Functional Requirement 11
	Non Functional Requirement 13
	Non Functional Requirement 16
	Non Functional Requirement 18
	Non Functional Requirement 20
	Non Functional Requirement 38

Table 19: Alfred Drive-Train Traceability

Component Module:	Functional and Non-Functional Requirement:
Alfred Pumping System Module	Alfred Functional Requirement 1
	Alfred Functional Requirement 4
	Alfred Functional Requirement 5
	Alfred Functional Requirement 6
	Alfred Functional Requirement 7
	Alfred Functional Requirement 8
	Alfred Functional Requirement 9
	Alfred Functional Requirement 11
	Alfred Functional Requirement 13
	Non Functional Requirement 1
	Non Functional Requirement 2
	Non Functional Requirement 10
	Non Functional Requirement 12
	Non Functional Requirement 17
	Non Functional Requirement 19

Table 20: Alfred Pumping System Traceability

Component Module:	Functional and Non-Functional Requirement:
Alfred Image Processing Module	Alfred Functional Requirement 10
	Non Functional Requirement 16
	Non Functional Requirement 20

Table 21: Alfred Image Processing Traceability

6 Component Overview

6.1 Ordering System Module

6.1.1 Description

This module will be used for user input by taking the orders of the client. This user input will be taken in by the mobile application based on different button inputs/ radio button selections. These orders are then

packaged by the module to be sent to the server based on an HTTP request.

6.1.2 Inputs and Outputs

Inputs: User input defining:

Input Name	Input Type	Range	Units	Comment(s)
$Order_{Num}$	Unsigned Integer User Input	[0,5]	count	Number of Orders
$Order_{Drinks}$	User Input	N/A	N/A	List of Ordered Drinks

Outputs: Packaged Information within HTTP Request for:

Output Name	Output Type	Range	Units	Comment(s)
$Order_{Num}$	Unsigned Integer User Input	[0,5]	count	Number of Orders
$Order_{Drinks}$	User Input	N/A	N/A	List of Ordered Drinks
$Order_{TableNum}$	Unsigned Integer User Input	$[0,2^{16}]$	N/A	Table Number
$Order_{Rid}$	Unsigned Integer User Input	$[0,2^{16}]$	N/A	Identification of the restaurant

6.1.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
$Order_{Num}$	Unsigned Integer User Input	$Order_{Num}$ of type string	Input Regulation
$Order_{Drinks}$	User Input	$Order_{Drinks}$ not of type string	Input Regulation

6.1.4 Timing Constraints

Timing Constraints are based on the server sending a success signal within $t_{timeout}$ seconds.

6.1.5 Initialization

At startup/new order of the application, it will start a blank order page where the user will be able to add drink orders and use radio buttons to select the desired drink.

6.2 Manager Login Page Module

6.2.1 Description

This module is a web based application for the managers to be able to log into the management systems.

6.2.2 Inputs and Outputs

Inputs:

Input Name	Input Type	Range	Units	Comment(s)
$UserName$	String User Input	10 chars	N/A	N/A
$Password$	String User Input	20 chars	N/A	N/A
$GotoMapPage$	Button User Input	[0,1]	N/A	N/A
$GotoAlfredInfo$	Button User Input	[0,1]	N/A	N/A

Outputs: To be displayed to the user.

Output Name	Output Type	Range	Units	Comment(s)
FailureMessage	String	10 chars	N/A	Failure message if there is an incorrect information
GotoPage	Action	N/A	N/A	navigating to the correct page if it was a success.

6.2.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
UserName	String User Input	N/A	N/A
Password	String User Input	N/A	N/A
GotoMapPage	Button User Input	N/A	N/A
GotoAlfredInfo	Button User Input	N/A	N/A

6.2.4 Timing Constraints

Given optimal networking conditions, the server must respond with $t_{timeout}$ seconds.

6.2.5 Initialization

This will default to a HTML page with no information.

6.3 Manager Station Map Software Module

6.3.1 Description

This module will be used for the user input to create a map where the manager of the restaurant will be able to define the details about the restaurant that will help Alfred with it's navigation to different tables.

6.3.2 Inputs and Outputs

Inputs: User input defining:

Input Name	Input Type	Range	Units	Comment(s)
$Resturant_{Width}$	Integer User input	[0,20]	m	N/A
$Resturant_{Length}$	Integer User input	[0,20]	m	N/A
$Resturant_{CanTravel}[][]$	Button[][] User input	N/A	N/A	A set of areas in which Alfred can travel to
$Resturant_{Tables}[][]$	Button[][] User input	N/A	N/A	A set of areas in which contains tables
$Resturant_{CannotTravel}[][]$	Button[][] User input	N/A	N/A	A set of areas in which Alfred cannot travel to
$Resturant_{Home}$	Button User input	N/A	N/A	An area that defines Alfred's home location

Outputs: A text file that contains the following information.

Input Name	Input Type	Range	Units	Comment(s)
$Resturant_{Width}$	Integer	[0,20]	m	N/A
$Resturant_{Length}$	Integer	[0,20]	m	N/A
$Resturant_{D}ata$	Char	[H,X,0,T]	N/A	Table Information

6.3.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
$Resturant_{Width}$	Integer User input	$Resturant_{Width}$ of type string	Input Regulation
$Resturant_{Length}$	Integer User input	$Resturant_{Length}$ of type string	Input Regulation
$Resturant_{CanTravel}[][]$	Button[][] User input	N/A	N/A
$Resturant_{Tables}[][]$	Button[][] User input	N/A	N/A
$Resturant_{CannotTravel}[][]$	Button[][] User input	N/A	N/A
$Resturant_{Home}$	Button User input	N/A	N/A

6.3.4 Timing Constraints

Timing Constraints are based on the server sending a success signal within $t_{timeout}$ seconds.

6.3.5 Initialization

At startup/new order of the application, it will load in the users map that is associated with their profile. If this is the first time using the application then it will default to a 1x1 map.

6.4 Manager Station Request Software Module

6.4.1 Description

This module is used for the manager to determine Alfred's state from an office as well as being able to override Alfred's system to come back to the home base.

6.4.2 Inputs and Outputs

Inputs:

Input Name	Input Type	Range	Units	Comment(s)
req_{Home}	Boolean User input	[0,1]	N/A	Calling back Alfred
$errors_{Alfred}$	Unsigned Byte	[0, 2^8]	N/A	Errors from Alfred

Outputs: To be displayed to the user.

Output Name	Output Type	Range	Units	Comment(s)
$warn_{LowLiquid}$	Boolean	[0, 1]	N/A	Low liquid levels
$error_{LiquidLeak}$	Boolean	[0, 1]	N/A	Leaking of liquid error
$error_{NotPumping}$	Boolean	[0, 1]	N/A	Not pumping error
$error_{NoMovement}$	Boolean	[0, 1]	N/A	Not able to move error
$error_{LowBatt}$	Boolean	[0, 1]	N/A	Low battery Error

6.4.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
req_{Home}	Boolean User input	N/A	N/A
$errors_{Alfred}$	Unsigned Byte	N/A	N/A

6.4.4 Timing Constraints

One timing constraints are based on the server sending a success signal within $t_{timeout}$ seconds. Another constraint is that Alfred will return within the time of w_{wheel}*distance.

6.4.5 Initialization

At startup this interface should pull the last status of the robot and display it to the user. If no previous data is found then it will return that there are currently no errors.

6.5 Back End Web Service Module

6.5.1 Description

This module is a server component that will hold order information for different tables. This will authorize different users to be able to accept different drink requests from the ordering system. It will return the next drink order for the restaurants Alfred robot.

6.5.2 Inputs and Outputs

Inputs:

Input Name	Input Type	Range	Units	Comment(s)
$HttpReq_{Orders}$	HTTP Request	N/A	N/A	HTTP request package with the information for drink orders.
$Map_{textfile}$	Text File	N/A	N/A	A text file with a map of the restaurant
$errors_{Alfred}$	Unsigned Byte	[0, 2^8]	N/A	Errors from Alfred
$UserName$	String	10 chars	N/A	N/A
$Password$	HashCode	N/A	N/A	Success result for authenticated results.

Outputs:

Output Name	Output Type	Range	Units	Comment(s)
$HttpReq_{Orders}$	HTTP Request	N/A	N/A	HTTP request package with the information for drink orders.
$Map_{textfile}$	Text File	N/A	N/A	A text file with a map of the restaurant
$errors_{Alfred}$	Unsigned Byte	[0, 2^8]	N/A	Errors from Alfred
$Login_{Status}$	String	10 chars	N/A	Login Success or Fail
$Http_{Result}$	String	10 chars	N/A	N/A

6.5.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
$HttpReq_{Orders}$	HTTP Request	$HttpReq_{Orders}$ formatting error	Server Validation
$Map_{textfile}$	Text File	N/A	N/A
$errors_{Alfred}$	Unsigned Byte	N/A	N/A
$UserName$	String	N/A	N/A
$Password$	HashCode	N/A	N/A

6.5.4 Timing Constraints

Given optimal networking conditions, the server must respond with $t_{timeout}$ seconds.

6.5.5 Initialization

This server will be initialized by a database with one restaurant which will be used for the purposes of testing the system.

6.6 Alfred Manager Module

6.6.1 Description

This module acts as a manager to the different components of Alfred. It will run on a raspberry pi to command the drive-train to move to different nodes of the map. The manager will also communicate through serial communication using a USB when ordering the next drink to the pumping system.

6.6.2 Inputs and Outputs

Inputs:

Input Name	Input Type	Range	Units	Comment(s)
Map	Graph	N/A	N/A	Graph of the text file with a map of the restaurant
$Orders_{drinks}$	Order[]	N/A	N/A	A list of drink orders ordered by table.
$b_{requestHome}$	Boolean	[0, 1]	N/A	A request to come back to the base
$errors_{drivetrain}$	Unsigned Byte	[0, 2^8]	N/A	Errors from drive-train
$errors_{pump}$	Unsigned Byte	[0, 2^8]	N/A	Errors from pumping system module

Outputs:

Output Name	Output Type	Range	Units	Comment(s)
$NextOrder$	Order	N/A	N/A	Next Drink Order
$NextNode$	Node	[(0,0), (length(Map), width(Map)]	N/A	The next node to travel to
$errors_{Alfred}$	Unsigned Byte	[0, 2^8]	N/A	Errors from Alfred

6.6.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
Map	Graph	N/A	N/A
$Orders_{drinks}$	Order[]	N/A	N/A
$b_{requestHome}$	Boolean	N/A	N/A
$errors_{drivetrain}$	Unsigned Byte	N/A	N/A
$errors_{pump}$	Unsigned Byte	N/A	N/A

6.6.4 Timing Constraints

Given optimal networking conditions, the system must receive a response within $t_{timeout}$ seconds. The communication with the pumping system must be done at the specified $freq_{bauderate}$.

6.6.5 Initialization

This will assume that there is no requests at startup. It will request from the server asking for the next drink order. If there is no map previously defined within a text file then it will request this as well.

6.6.6 Raspberry PI Specifications

Manufacturer: Raspberry PI
Processor: Broadcom BCM2387 chipset.
Memory: 1GB
Power:Micro USB socket 5V1, 2.5A
GPIO: 17 pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector: 15-pin MIPI Camera Serial Interface (CSI-2)
Memory Card Slot: Push/pull Micro SDIO

6.7 Alfred Drive-Train Module

6.7.1 Description

This module will provide power to Alfred's drive-train. It will use the feedback of the left and right encoders and take the error to perform PI control on them. This PI control output will then be translated into a duty cycle for each side to be able to power the DC motors with pulse width modulation. This module will communicate to the image recognition software to receive the position of the next marker and use this information of where it currently is on its path. This module will use ultrasonic sensors to get a set of distances (d_{object}) to the nearest object to determine if it is safe to continue moving.

6.7.2 Inputs and Outputs

Inputs:

Input Name	Input Type	Range	Units	Comment(s)
Map	Graph	N/A	N/A	Graph of the text file with a map of the restaurant
$NextNode$	Node	[(0,0), (length(Map), width(Map)]	N/A	The next node to travel to
$w_{wheel_{left}}$	Float	[0, 100]	rad/s	Left Encoder Input

$w_{wheel_{right}}$	Float	[0, 100]	rad/s	Right Encoder Input
$d_{objects}$	Float[]	[0, 10.0]	m	N/A
$Marker_{PosX}$	Unsigned Integer	[0,2^{16}]	Pixels	N/A
$Marker_{PosY}$	Unsigned Integer	[0,2^{16}]	Pixels	N/A
$b_{NextMarkerFound}$	Boolean	[0,1]	N/A	N/A
V_{batt}	Float	[0, 20.0]	m	N/A

Outputs:

Output Name	Output Type	Range	Units	Comment(s)
$percent_{dutycycle_{left}}$	Float	[0,1.0]	%	N/A
$percent_{dutycycle_{right}}$	Float	[0, 1.0]	%	N/A
$errors_{drivetrain}$	Unsigned Byte	[0, 2^8]	N/A	N/A

6.7.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
Map	Graph	*Map* formatting error	Map set from User Interface which constricts format of file.
NextNode	Node	N/A	N/A
$w_{wheel_{left}}$	Float	N/A	N/A
$w_{wheel_{right}}$	Float	N/A	N/A
$d_{objects}$	Float[]	N/A	N/A
$Marker_{PosX}$	Unsigned Integer	N/A	N/A
$Marker_{PosY}$	Unsigned Integer	N/A	N/A
$b_{NextMarkerFound}$	Boolean	N/A	N/A
V_{batt}	Float	N/A	N/A

6.7.4 Timing Constraints

This module will have to deliver speed requirements that of walking speed so that it will be able to serve tables at a timely manner.

6.7.5 Initialization

This module will not be started until manager starts the process and initializes the proper information.

6.7.6 Diagram of Simulink Control System

Figure 3 is a diagram that shows the top level of the dc motor control system.

6.7.7 DC Motors Circuit Diagram

Figure 4 is a circuit diagram for showing the DC motor drive-train.

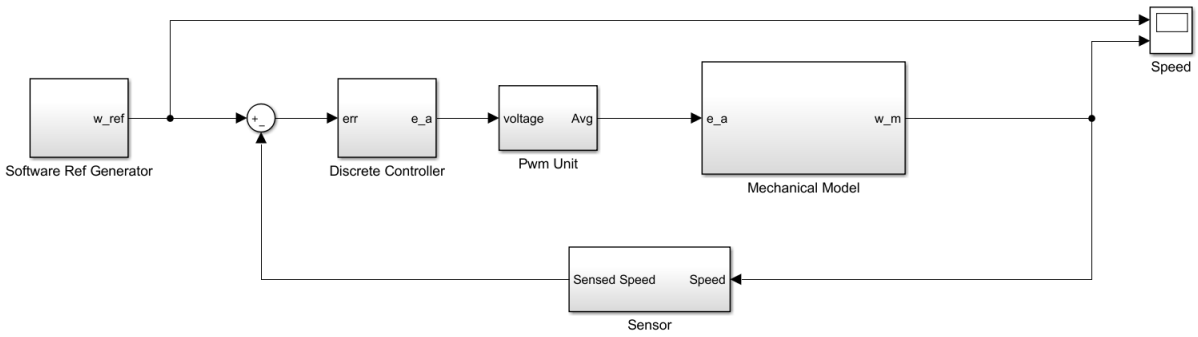

Figure 3: Top level of the dc motor control system

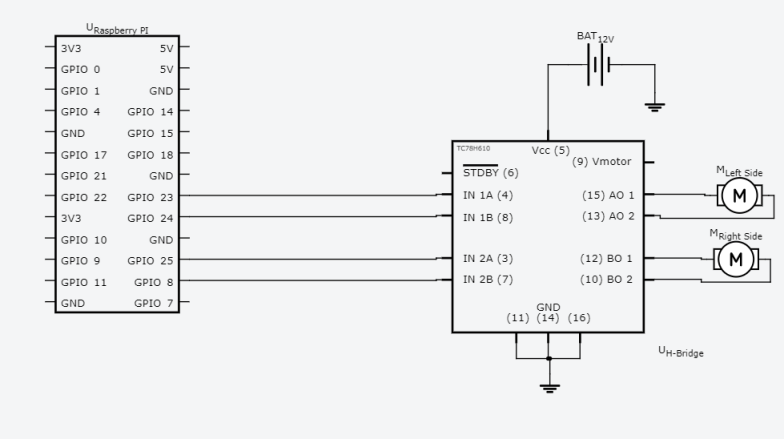

Figure 4: H-Bridge Circuit Diagram

6.7.8 DC Motor Specifications

Manufactured Part Number: 393111-01
Rated Voltage: 18V
Current: 3A-5A

6.7.9 H-bridge Specifications

Manufactured Part Number: IRF3205
Rated Voltage: 3 36V
Rated Current: 10A continuous, Peak 30A

6.7.10 Encoder Specifications

Shaft Diameter 6mm
Working Range: 3-5V DC

6.8 Alfred Pumping System Module

6.8.1 Description

This module consists of an Arduino Mega which will receive information from the Alfred manager module through UART to receive the next drink order. This will then begin to dispense the specific drink until it has reached $m_{containerMin}$. It will then show the customer that the drink has been completed by turning on: $LED_{drinksignal}$. It will then read $b_{cuptaken}$ from a light sensor to determine if the cup has been taken at which point it will then wait for the next drink cup to get in place and begin pouring. If it is not able to pump liquid, or it is losing fluid when not dispensing, then it will send the appropriate errors through UART back to the manager.

6.8.2 Inputs and Outputs

Inputs:

Input Name	Input Type	Range	Units	Comment(s)
$b_{cuptaken}$	Boolean	[0,1]	N/A	N/A
$m_{container}$	float	[0, 1.0]	Kg	N/A
m_{drink}	float	[0, 1.0]	Kg	N/A
$Order_{drinks}$	Order[]	N/A	N/A	List of Drinks

Outputs:

Output Name	Output Type	Range	Units	Comment(s)
$LED_{drinksignal}$	Boolean	[0,1]	N/A	N/A
V_{pump}	float	[0, 5.0]	V	N/A
$errors_{pump}$	Unsigned Byte	[0, 2^8]	N/A	N/A

6.8.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
$b_{cuptaken}$	Boolean	N/A	N/A
$m_{container}$	float	N/A	N/A
m_{drink}	float	N/A	N/A
$Order_{drinks}$	Order[]	$Order_{drinks}$ formatting error	Server Validation

6.8.4 Timing Constraints

This module will have to dispense the drink within $t_{maxpump}$

6.8.5 Initialization

This module start by waiting for the manager to send the drink information for the system to pour. All errors within the system will start as false until they have been triggered.

6.8.6 Diagram of DC Pump Control System

Figure 5 is a diagram that shows the top level of the dc pump control system.

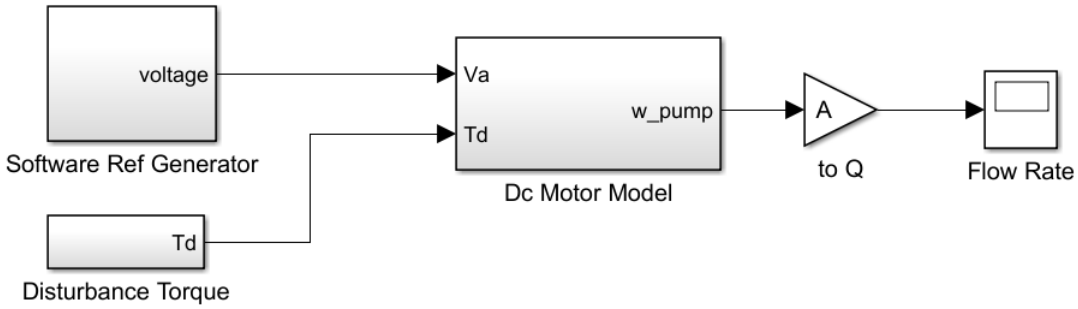

Figure 5: Top level of the dc motor control system

6.8.7 DC Pump Circuit Diagram

Figure 6 is a diagram that shows the DC Pump Circuit.

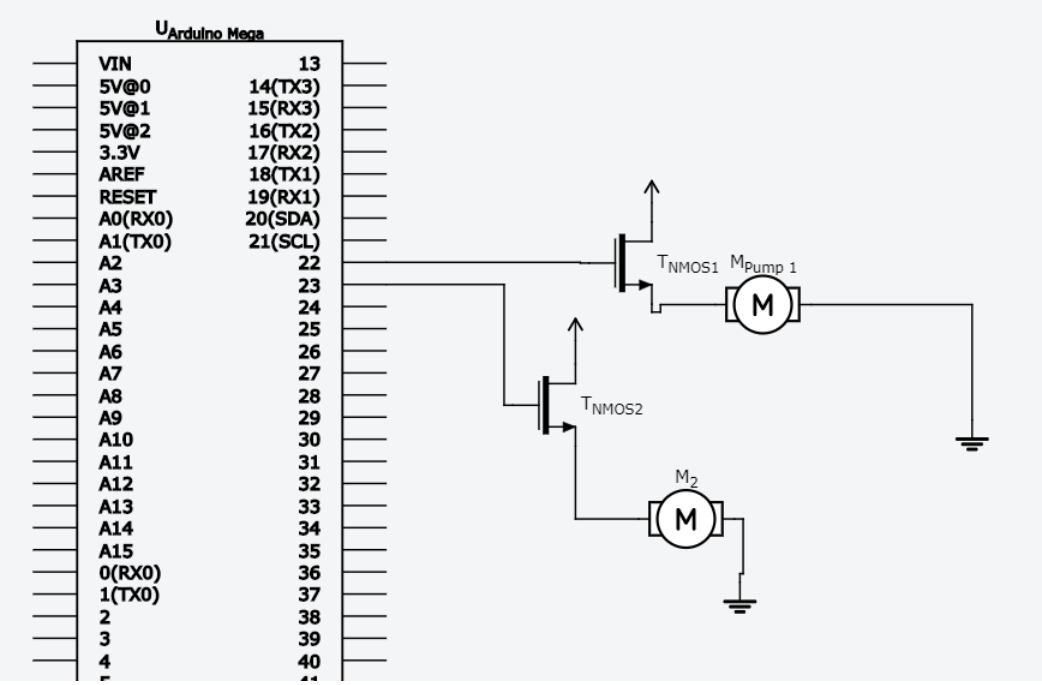

Figure 6: DC Pump Circuit Diagram

6.8.8 Liquid Temperature Circuit Diagram

Figure 7 is a circuit diagram for sensing the temperature of the liquid.

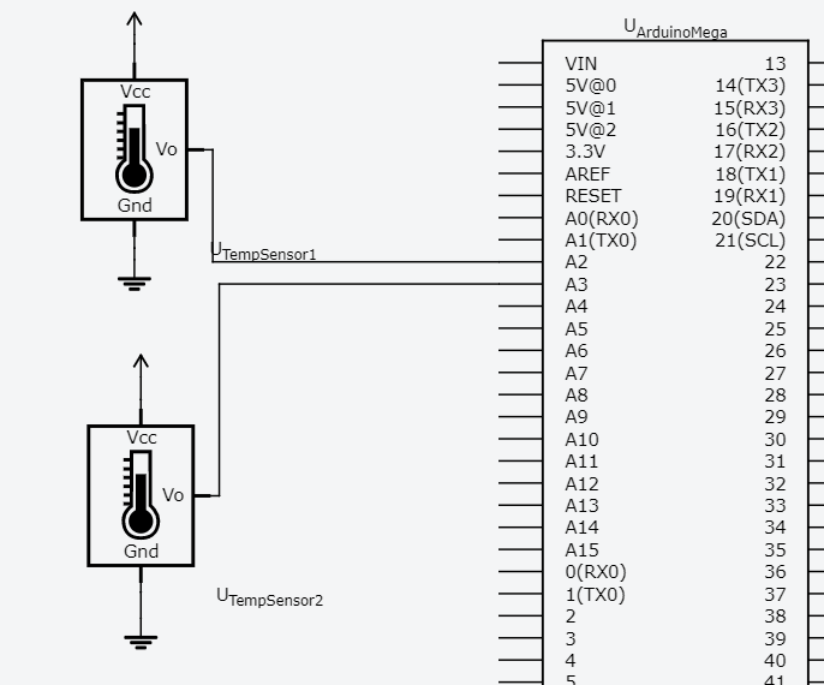

Figure 7: Top level of the dc motor control system

6.8.9 Weight Detection Circuit Diagram

Figure 8 is a circuit diagram for sensing the weight of the storage containers.

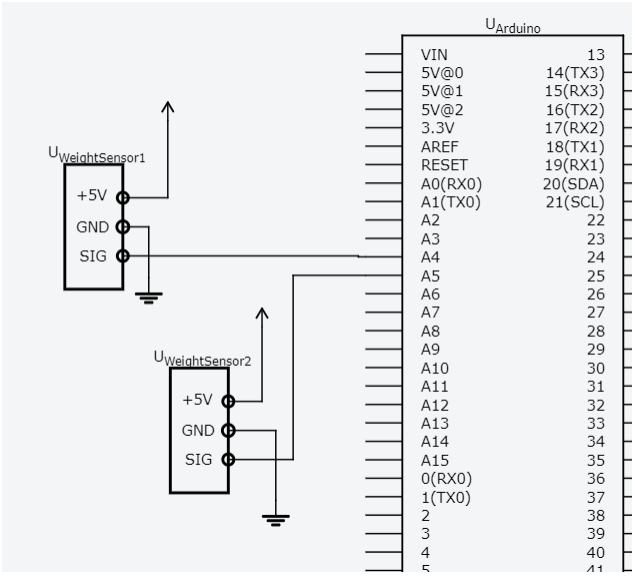

Figure 8: Weight Detection Circuit Diagram

6.8.10 DC Pump Specifications

Manufacture : Yosoo
DC Voltage: 3-6V
Flow rate: 80-120L/H
Material: engineering plastic
Diameter: 24.5mm by 46mm
Outside diameter: 0.3"

6.8.11 Arduino Mega Specifications

Microcontroller: ATmega1280
Operating Voltage: 5V
Digital I/O Pins: 54 (of which 15 provide PWM output)
Analog Input Pins: 16
DC Current per I/O Pin: 40 mA
DC Current for 3.3V Pin: 50 mA
Flash Memory: 128 KB
Clock Speed: 16MHz

6.8.12 Container Specifications

Manufacture : Rubbermaid
Dimensions: 10 1/2" x 9" x 4" Capacity: 18 Cups/4.2 L

6.8.13 Tubing Specifications

Manufacture : Plumb Craft
Diameter: 1/4"ID Food Grade Tubing will be used.

6.9 Alfred Image Processing Module

6.9.1 Description

This module will receive information from a camera while the drive-train is in motion. This camera will be looking at the ceiling looking for positions of circles that will denote one meter distances set up within a grid around the ceiling of the restaurant. This module will determine if there is a new marker within the image and if so where is the X and Y position of that marker to provide to the drive-train.

6.9.2 Inputs and Outputs

Inputs:

Input Name	Input Type	Range	Units	Comment(s)
$Image_{input}$	Bitmap	N/A	N/A	Image of the ceiling

Outputs:

Output Name	Output Type	Range	Units	Comment(s)
$Marker_{PosX}$	Unsigned Integer	[0,2^{16}]	Pixels	N/A
$Marker_{PosY}$	Unsigned Integer	[0,2^{16}]	Pixels	N/A
$b_{NextMarkerFound}$	Boolean	[0,1]	N/A	N/A

6.9.3 Exception Handling

Input Name	Input Type	Exception	Exception Handling
$Image_{input}$	Bitmap	N/A	N/A

6.9.4 Timing Constraints

This information must process in time for the drive train to be able to navigate based off of it.

6.9.5 Initialization

This module start by the drive-train application.

6.9.6 Raspberry PI Camera Specifications

Manufacturer: Raspberry PI
Resolution: 1080p30, 720p60 and 640 ÃŮ 480p60/90
Field of View (FOV): 62.2 degrees by 48.8 degrees

7 Likelihood of Change

Module	Likelihood of Change	Rationale
Ordering System Module	Very Unlikely	Key implementation aspect
Manager Login Page Module	Very Unlikely	Key implementation aspect
Manager Station Map Software Module	Very Unlikely	Key implementation aspect
Manager Station Request Software Module	Very Unlikely	Key implementation aspect
Back End Web Service Module	Very Unlikely	Key implementation aspect
Alfred Manager Module	Very Unlikely	Key implementation aspect
Alfred Pumping System Module	Very Unlikely	Key implementation aspect
Alfred Image Processing Module	Very Unlikely	Key implementation aspect

8 Normal Operation

Alfred is a mostly autonomous robot, only requiring human intervention in the event of an error or warning. Alfred will be able to navigate the restaurant by itself, and will serve drinks to tables. Customers will be able to place orders via a mobile application, which will be sent to a server. Orders to serve will be sent to Alfred using a FIFO protocol. Once Alfred has finished with an order, it will be able to request for a new order to serve. Management will be able to recall Alfred to the kitchen at any point using the admin console. In the event of a recall, Alfred will finish the current job and will return to the kitchen afterwards. Management will also be able to create a map of the restaurant and upload it to Alfred, which will give Alfred the means to navigate the restaurant.

9 Undesired Event Handling

Alfred will be able to detect undesired behaviours and conditions such as low liquid levels below threshold $m_{containerMin}$, any issues with pumping liquid, low battery level, leaking liquids, and any blockages in the current path once Alfred has been blocked for a time greater than $obstruction_{timeout}$. In the event of any error condition, Alfred will send an error code to the kitchen, to alert the staff of the issue. Wherever possible, Alfred will return to the kitchen in an error condition to request a fix. Otherwise, if movement is not possible, the kitchen staff will have to pick Alfred up from the dining room. Along with alerting management of any current errors, Alfred will also be able to indicate whether it is returning to the kitchen or requires pickup.

10 ClientApp

10.1 Purpose

The following will describe the component software design associated with the Client Application. This will be carried out within android based tools to allow customers to order drinks from any table..

10.2 Scope

The scope of this section is associated with any front end user interfaces that the customers will use.

10.3 Module Decomposition

Activity_Login: GUI class for admin to log into client app device.
Activity_Settings: GUI class for admin to reset device cart, as well as change the table number of the client device. Secrets include parsing and storing table info coming from the server.
Activity_DrinksList [Requirement T01]: GUI class for users to go through available drinks and choose their order. Secrets include parsing drink info coming from the server.
Activity_OrderCart [Requirement T01]: GUI class for users to view their current cart, as well as their previous orders. Secrets include parsing and dealing with server's response when order is sent to the server.
DrinksViewAdapter: Adapter class for each visible drink in Activity_DrinksList
DrinkCartListAdapter: Adapter class for each cart item in Activity_OrderCart
Drink: Object class representing a drink item.
DrinkOrder: Object class representing a table's order to be sent to the server.
NetworkCalls [Requirement T02]: Asynchronous class to perform network calls for the client app. Secrets include how network requests/responses are handled.
AsyncResponse [Requirement T02] Interface to deliver NetworkCall results to the current activity for processing.

10.4 Uses Relation

Please refer to Figure 1 on following page.

10.5 MIS

10.5.1 Drink

Uses : None
Public Functions
Drink(String name,int calories,int image,double price) Constructor for Drink class. Takes the name, image, and price of the drink, and the amount of calories in a serving.
setAmount(int amt):void Sets the number of drinks of this type in the cart.
getName():String Returns the name of the drink.

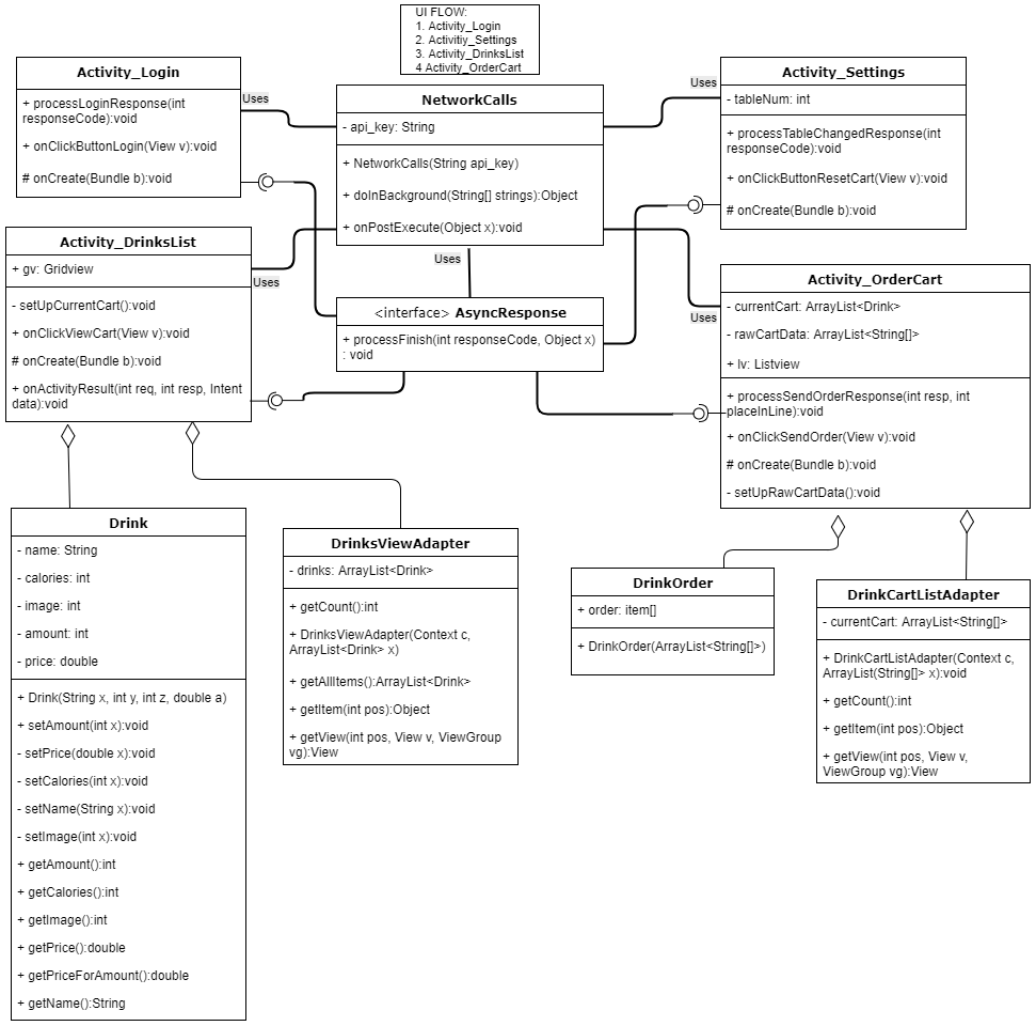

Figure 9: Android Client Application Uses Relation Diagram

getCalories():int Returns the amount of calories for the restaurant's serving size.
getImage():int Returns the id number for the image of this drink type.
getPrice():double Returns the price of this drink.
getPriceForAmount():double Returns the total price of the amount of this drink.
getAmount():int Returns the amount of the drink currently in the cart.

10.5.2 DrinkOrder

Uses : None
Public Functions
DrinkOrder(ArrayList<String[]> rawCartData) Constructor for DrinkOrder class. Takes the raw cart data for the currently selected drinks.

10.5.3 Activity_DrinksList

Uses

- DrinksViewAdapter (Class)
- NetworkCalls (Class)
- AsyncResponse (Interface)

onActivityResult(int requestCode, int responseCode, Intent data):void Process the results of any activities launched by this activity.
onClickViewCart(View v):void Listener for the view cart button. Launches Activity_OrderCart.
processDrinksInfoResponse(int responseCode):void Processes the response of the drink info retrieval from the server.
onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created.

10.5.4 Activity_Login

Uses

- NetworkCalls (Class)
- AsyncResponse (Interface)

Public Functions
onClickButtonLogin(View v):void Listener for the login button. Attempts to log in with filled in username and password.
onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created.

10.5.5 Activity_Settings

Uses

- NetworkCalls (Class)
- AsyncResponse (Interface)

onClickButtonResetCart(View v):void Listener for the reset cart button. Resets the cart for the next guests.
processTableChangeResponse(int responseCode, String token):void Processes the class specific changes based on the response from the server.
onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created.

10.5.6 Activity_OrderCart

Uses

- DrinkCartListAdapter (Class)
- NetworkCalls (Class)
- AsyncResponse (Interface)

Public Functions
processSendOrderResponse(int responseCode, int placeInLine):void Processes the response of attempt to place an order with the server.
onClickSendOrder(View v):void Listener for the send order button. Sends the current cart to the server.
onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created.

10.5.7 NetworkCalls

Uses

- AsyncResponse (Interface)

Public Functions
NetworkCalls(String api_key) Constructor for NetworkCalls. Takes the api key.
doInBackground(String[] strings):Object Executes the HTTP request based on the api key provided.
onPostExecute(Object result):void Processes the response after doInBackground is completed.

10.5.8 Interface AsyncResponse

Uses : None
Public Functions
processFinish(int responseCode, Object x): void Processes data in calling activity after NetworkCalls is completed execution.

10.5.9 DrinksViewAdapter

Uses

- Drink (Class)

Public Functions
DrinksViewAdapter(Context context, ArrayList<Drink> drinks) Constructor for DrinksViewAdapter. Takes the context of this adapter and the list of drink info.
getCount():int Returns the amount of items in the list of drinks.
getAllItems():ArrayList<Drink> Returns the list of drink info.
getItem(int position):Object Returns the item for the position id passed to it.
getView(int position, View v, ViewGroup parent):View Returns the view for the position id passed to it. Takes the position id, current View, and ViewGroup.

10.5.10 DrinkCartListAdapter

Uses None
Public Functions
DrinkCartListAdapter(Context context, ArrayList<String[]> currentCartInfo) Constructor for DrinkCartListAdapter. Takes the context of this adapter and the current cart information.
getCount():int Returns the amount of items in the cart.
getItem(int position):Object Returns the item for the position id passed to it.
getView(int position, View v, ViewGroup parent):View Returns the view for the position id passed to it. Takes the position id, current View, and ViewGroup.

10.6 MID

10.6.1 Drink

Uses : None
Internal Variables name: String - Name of the drink
calories: int - Number of calories in a serving of this drink
image: int - id of the image of this drink
amount: int - the amount of this drink currently selected by the user
Functions
Drink(String name,int calories,int image,double price) Constructor for Drink class. Takes the name, image, and price of the drink, and the amount of calories in a serving. Sets the internal variable values.

public setAmount(int amt):void Sets the number of drinks of this type in the cart.
private setName(String name):void Sets the name of the drink. Takes in the name of the drink.
private setCalories(int cals):void Sets the amount of calories for the restaurant's serving size. Takes in amount of calories.
private setImage(int imgID):void Sets the id number for the image of this drink type. Takes in an integer image id.
private setPrice(double price):void Sets the price of this drink. Takes in the price of the drink.
public getName():String Returns the name of the drink.
public getCalories():int Returns the amount of calories for the restaurant's serving size.
public getImage():int Returns the id number for the image of this drink type.
public getPrice():double Returns the price of this drink.
public getAmount():int Returns the amount of this drink currently in the cart.
public getPriceForAmount():double Returns the total price of the amount of this drink.

10.6.2 DrinkOrder

Uses : None
Internal Variables
order: item[] - Array of drink items
Functions
public DrinkOrder(ArrayList<String[]> rawCartData) Constructor for DrinkOrder class. Takes the raw cart data for the currently selected drinks.

10.6.3 Activity_DrinksList

Uses

- DrinksViewAdapter (Class)
- NetworkCalls (Class)
- AsyncResponse (Interface)

Internal Variables gv: GridView - The view holding the list of drink items
Functions
public onActivityResult(int requestCode, int responseCode, Intent data):void Process the results of any activities launched by this activity.
public onClickViewCart(View v):void Listener for the view cart button. Launches Activity_OrderCart.
public processDrinksInfoResponse(int responseCode):void Processes the response of the drink info retrieval from the server.
protected onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created. Retrieve the drink info from the server and initialize each item in the gridview as a DrinksViewAdapter.
private setUpCurrentCart():ArrayList<Drink> Retrieves and returns the items stored in gv.

10.6.4 Activity_Login

Uses

- NetworkCalls (Class)
- AsyncResponse (Interface)

Internal Variables None
Functions public processLoginResponse(int responseCode):void Processes class specific changes based on the log in attempt. If (responseCode == 200) then launch settings page. Else show alert.

public onClickButtonLogin(View v):void Listener for the login button. Attempts to log in with filled in username and password. Executes a NetworkCall task with the correct api key passed.
protected onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created.

10.6.5 Activity_Settings

Uses

- NetworkCalls (Class)
- AsyncResponse (Interface)

Internal Variables tableNum: int - The table number associated with this device.
Functions public onClickButtonResetCart(View v):void Listener for the reset cart button. Resets the cart for the next guests. Clears all cart information.
public processTableChangeResponse(int responseCode, String token):void Processes the class specific changes based on the response from the server. If (responseCode == 200) then store the token and new tableNum. Else show alert and keep tableNum the same.
onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created.

10.6.6 Activity_OrderCart

Uses

- DrinkCartListAdapter (Class)
- NetworkCalls (Class)
- AsyncResponse (Interface)

Internal Variables currentCart: ArrayList<Drink> - ArrayList of Drink objects representing the current cart selections.
rawCartData: ArrayList<String[]> - ArrayList of String[] representing the current cart. This is data used for the DrinkCartListAdapter.
lv: ListView - The view which contains and displays the cart items.
Functions public processSendOrderResponse(int responseCode, int placeInLine):void Processes the response of attempt to place an order with the server.
public onClickSendOrder(View v):void Listener for the send order button. Sends the current cart to the server.
protected onCreate(Bundle savedInstanceState):void Initialize this Activity(app menu) when created.
private setUpRawCartData():void Uses the information from the currentCart to set up an ArrayList<String[]>. Each item in the ArrayList is a String array consisting of [name, amount, price for amount]

10.6.7 NetworkCalls

Uses

- AsyncResponse (Interface)

Internal Variables
api_key: String - String representing the api being targeted. Used for conditionals.

- "table_token" - target (HOST)\table?tableId=?
- "placeOrder" - target (HOST)\placeOrder?tableId=?

- "login" - target (HOST)\login

Functions
public NetworkCalls(String api_key) Constructor for NetworkCalls. Takes the api key.
public doInBackground(String[] strings):Object Executes the HTTP request based on the api key provided.
public onPostExecute(Object result):void Processes the response after doInBackground is completed.

10.6.8 Interface AsyncResponse

Uses:None
Internal Variables : None
Functions
public processFinish(int responseCode, Object x): void Processes data in calling activity after NetworkCalls is completed execution.

10.6.9 DrinksViewAdapter

Uses

- Drink (Class)

Internal Variables
drinks: ArrayList<Drink> - ArrayList of Drink objects representing the current cart selections.
Functions
public DrinksViewAdapter(Context context, ArrayList<Drink> drinks) Constructor for DrinksViewAdapter. Takes the context of this adapter and the list of drink info.
public getCount():int Returns the amount of items in the list of drinks.
public getAllItems():ArrayList<Drink> Returns the list of drink info.
public getItem(int position):Object Returns the item for the position id passed to it.
public getView(int position, View v, ViewGroup parent):View Returns the view for the position id passed to it. Takes the position id, current View, and ViewGroup.

10.6.10 DrinkCartListAdapter

Uses None
Internal Variables currentCart: ArrayList<String[]> - ArrayList of Drink objects representing the current cart selections.
Functions public DrinkCartListAdapter(Context context, ArrayList<String[]> currentCartInfo) Constructor for DrinkCartListAdapter. Takes the context of this adapter and the current cart information.
public getCount():int Returns the amount of items in the cart.
public getItem(int position):Object Returns the item for the position id passed to it.
public getView(int position, View v, ViewGroup parent):View Returns the view for the position id passed to it. Takes the position id, current View, and ViewGroup.

11 Manager System

11.1 Purpose

The following will describe the component software design associated with Manager System. This will be carried out within web based tools to allow management to access there information anywhere.

11.2 Scope

The scope of this section is associated with any front end user interfaces that the management staff will use. This includes the MIS/MID and uses relation in regards to the Map Making Page, the Error Viewing page and the Login System.

11.3 Module Decomposition

Manager Login Page: Given the login credentials, will authenticate administrator of the system with the server. Secrets include how it goes about verifying with the server if the credentials are valid. **Manager Station Map Software Page**: Will allow administrator to create or modify the map of the area where Alfred will deliver drinks. This map will then be sent and stored on the server. The secrets of this module includes how the mapping system will translate user input into the map file **Manager Station Request Software Page**: Will allow administrator to execute commands for Alfred, as well as view incoming error codes from Alfred. Secrets include how the errors are decoded from the server.

11.4 Uses Relation

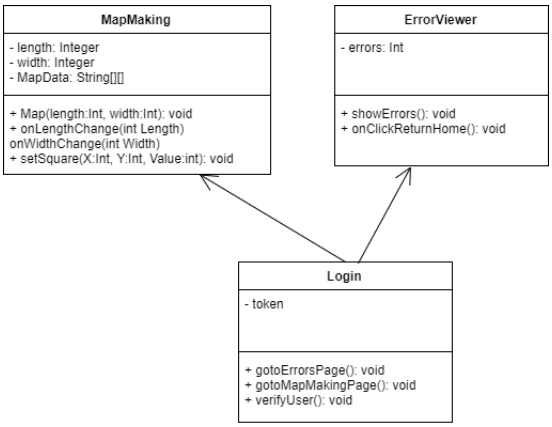

Figure 10: Alfred Uses Relation Diagram

11.5 MIS

11.5.1 Login Page

Uses

- MapMaking (WebPage)
- ErrorViewer (WebPage)

Public Functions
gotoErrorsPage(): void Navigates to the page associated with showing the Managers page for Errors with Alfred.
gotoMapMakingPage(): void Navigates to the page associated with showing the Managers page for creating a resturant Map.
verifyUser(): void Determines if the information that the user put into the form on the webpage is correct.

11.5.2 MapMaking Page

Uses None
Public Functions
Map(length:Int, width:Int): void Constructor to object for page load
onLengthChange(int Length) Sets the length of the map when the user changes it in the form. This is based on requirement AD1.
onWidthChange(int Width) Sets the width of the map when the user changes it in the form.This is based on requirement AD1.
setSquare(X:Int, Y:Int, value:Int): void Sets the value of the square based on its X,Y position when there is an on click event. This is based on requirement AD2.

11.5.3 ErrorViewer Page

Uses None
Public Functions
showErrors(): void Shows the Errors Associated to Alfred. Based on requirement NFR26. **onClickReturnHome()** Signals the Robot to return home. Based on requirement NFR11.

11.6 MID

11.6.1 Login Page

Uses

- MapMaking (WebPage)
- ErrorViewer (WebPage)

Internal Variables
token: String - Token for a session with the server.
Functions
public gotoErrorsPage(): void Navigates to the page associated with showing the Managers page for Errors with Alfred.
public gotoMapMakingPage(): void Navigates to the page associated with showing the Managers page for creating a resturant Map.
public verifyUser(): void Determines if the information that the user put into the form on the webpage is correct.

11.6.2 MapMaking Page

Uses None
Internal Variables
length: Integer - Length of the Map
width: Integer - Width of the Map
MapData: String[][]- Storage of the map values
Public Functions
public Map(length:Int, width:Int): void

Constructor to object for page load
public onLengthChange(int Length)
Sets the length of the map when the user changes it in the form.
public onWidthChange(int Width)
Sets the width of the map when the user changes it in the form.
setSquare(X:Int, Y:Int, value:Int): void
Sets the value of the square based on its X,Y position when there is an on click event. The Values corresponds to:

- 0: Free to move
- 1: Path is blocked
- 2: Table
- 3: Base

11.6.3 ErrorViewer Page

Uses None
Public Functions
showErrors(): void Shows the Errors Associated to Alfred. where the Errors are in the following format

- LowLiquid: 0x00000001
- LeakingTank: 0x00000010
- LowBattery: 0x00000100
- NoMovement: 0x00001000

onClickReturnHome() Signals the Robot to return home.

12 Alfred System

12.1 Purpose

The following will describe the component software, mechanical and electrical design associated with Alfred's Manager System, Alfred's Drivetrain and Alfred's Image Processing system. These three systems will be ran on the Raspberry Pi.

12.2 Scope

The scope of this section is associated with Alfred's Manager System, Alfred's Drivetrain and Alfred's Image Processing system. The software documentation will provide the MIS and MID, uses relations to describe how the system will be designed to preform its function of being able to drive to a specific location. The mechanical and electrical design will focus on the aspects to provide motion and navigation.

12.3 Module Decomposition

Alfred Manager Module: Endpoint for communication with Alfred. Will manage communication with server, as well as send any errors that Alfred is experiencing. Secrets include Parsing of messages from the server and the pumping system. **Alfred Drive Train Module**: Responsible for driving and managing the motors based on desired route. Will also be sending errors preventing movement to Alfred Manager Module. Secrets include how the robot will preform navigation based on the map, how the robot will control and drive the robot and the inputs from the image processing module. **Image Processing Module**: Will detect any obstacles in the way as well as locate incoming nodes. Will communicate with Alfred Drive Train Module, to determine whether any required action based on results. Secretes include how the image processing will be carried out

12.4 Uses Relation

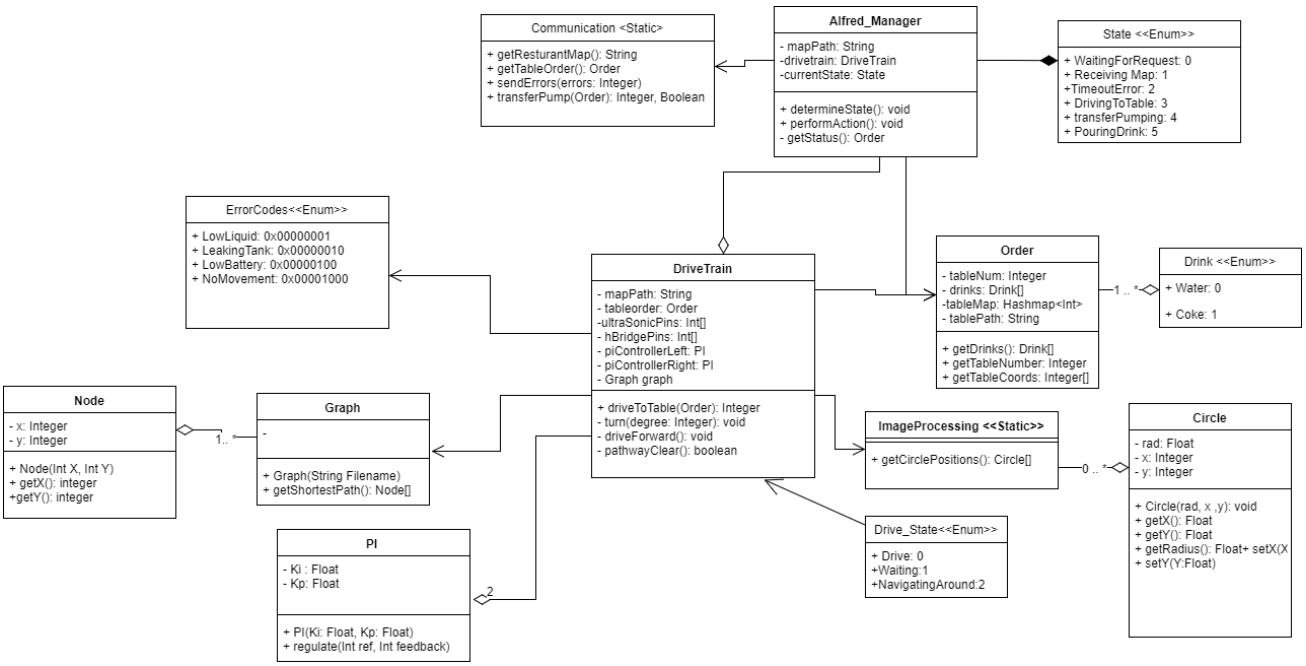

Figure 11: Alfred Uses Relation Diagram

12.5 MIS

12.5.1 Alfred Manager Class

Uses

- Communication (Static Class)
- State (Enum)
- Drivetrain (Class)
- Order (Class)

Public Functions
determineState(): void
Determines the overall state of Alfred based on the inputs to Alfred.
performAction(): void
Will preform the desired state based on the state of Alfred.

12.5.2 Order Class

Uses

- Drink (Enum)

Public Functions
getDrinks()
Drink[] - Returns the list of drinks. Based on requirement AF1.
getTableNumber

Integer - Returns the table number reference. Based on requirement AF3.
getTableCoords
Integer[] - Returns the coordinates to the table. Based on requirement AF3.

12.5.3 Drivetrain Class

Uses

- Order (Class)
- ImageProcessing (Class)
- Drive_State (Enum)
- ErrorCodes (Enum)
- Circle (Class)
- Graph (Class)
- Node (Class)
- PI (Class)

Public Functions driveToTable(Order): Integer This function will preform the driving operation in order to navigate towards the specific table. Based on requirement AF3.

12.5.4 Communication Static Class

Uses
None
Public Functions
getResturantMap(): String
Retrieves and stores the map to be used for navigation. Returns the path of the map. Based on requirement AF3.
getTableOrder(): Order
Retrieves and returns the Table's Order. Based on requirement AF3.
sendErrors(errors: Integer)
Sends an integer with the described set of integers. Based on requirement NFR26.
transferPump(Order): Integer
Preforms communication with the pumping system. Sends the Order data and receives the errors from the pumping system and if it complete. Based on requirement AF1.

12.5.5 Graph Class

Uses

- Node (Class)

Public Functions
Graph(String Filename)
Constructor to create a graph object. Builds the graph based on the path to the map. Based on requirement AF3.
getShortestPath(): Node[]
Returns a list of nodes that describes the shortest way to get to the destination.

12.5.6 Node Class

Uses None
Public Functions
Node(Int X, Int Y)
Constructor to the node class, takes in the position of the point along the X Y plane.
getX(): integer
Returns the x coordinate of the node
getY(): integer
Returns the y coordinate of the node

12.5.7 PI Class

Uses None
Public Functions
PI(Ki: Float, Kp: Float)
Constructor for the PI controller object taking in the Ki and Kp terms
regulate(Int ref, Int feedback)
Based on the reference to control to and the feedback, will determine the value to set the outputs too. Based on requirement AF3 to allow regulated movement to the location.

12.5.8 Imaging Processing Static Class

Uses

- Circle (Class)

Public Functions
getCirclePositions(): Circle[]
Gives a list of circles that were within the view of the camera

12.5.9 Circle Class

Uses None
Public Functions
Circle(radius, x, y)
Constructor of the circle class, takes in initial radius, x and y values. Based on requirement AF3 to allow regulated movement to the location.
getX(): Float
Gives the X position of the circle relative to the image
getY(): Float
Gives the Y position of the circle relative to the image
getRadius(): Float
Gives the radius of the circle
setX(X:Float)
Sets the X position of the circle.
setY(Y:Float)
Sets the Y position of the circle.
setRadius(rad : Float)
Sets the radius of the circle

12.6 MID

12.6.1 Alfred Manager Class

Uses

- Communication (Static Class)
- State (Enum)
- Drivetrain (Class)
- Order (Class)

Internal Variables
mapPath: String - The absolute path to the map directory
drivetrain: DriveTrain - An object encapsulates information in regards to the drivetrain
currentState: State - Holds the information in regards to which action will be preformed by Alfred
Functions
public determineState(): void
Determines the overall state of Alfred based on the inputs to Alfred based on the following tabular expression:

Previous State	Conditions	Next State
WaitingForRequest	Order == Null && time < timeout	WaitingForRequest
	Order == Null && timeout <= time	TimeoutError
	Order! = Null	ReceivingMap
ReceivingMap	Order == Null && time < timeout	WaitingForRequest
	Order == Null && timeout <= time	TimeoutError
	Order! = Null	DrivingToTable
TimeoutError	time< try_again	TimeoutError
	try_again <= time	WaitingForRequest
DrivingToTable	Drive_errors == 0	transferPumping
	Drive_errors != 0	WaitingForRequest
transferPumping	Pump_errors ==0 && table_done	WaitingForRequest
	Pump_errors !=0	WaitingForRequest

public performAction(): void
Will preform the desired state based on the state of Alfred based on the following table:

State	Action
WaitingForRequest	getTableOrder() sendErrors(errors)
ReceivingMap	getResturantMap()
TimeoutError	Sleep()
DrivingToTable	driveToTable(Order)
transferPumping	transferPump(Order)

12.6.2 Order Class

Uses

- Drink (Enum)

Internal Variables
tableNum: Integer - the reference to the table
drinks: Drink[] - List of drinks for the user's table

tableMap: Hashmap<Int> - Map that takes in a table reference number and returns its X,Y Coord
tablePath: String - gives the path of the table for the hashmap
Functions
public getDrinks(): Drink[]
Returns the list of drinks
public getTableNumber: Integer
Returns the table number reference
public getTableCoords: Integer[]
Returns the coordinates to the table from the Hashmap of table numbers

12.6.3 Drivetrain Class

Uses

- Order (Class)
- ImageProcessing (Class)
- Drive_State (Enum)
- ErrorCodes (Enum)
- Circle (Class)
- Graph (Class)
- Node (Class)
- PI (Class)

Internal Variables mapPath: String - The absolute path to the map directory
tableorder: Order - The order from the next table.
ultraSonicPins: Int[] - The pins dedicated for the ultrasonic sensors
hBridgePins: Int[] - The pins dedicated for the H-bridge
graph: Graph - Object to the graph object to find the shortest path
piControllerLeft: PI - Object used for PI control for the left side of the drivetrain
piControllerRight: PI - Object used for PI control for the left side of the drivetrain
Functions
public driveToTable(Order): Integer
This function will preform the driving operation in order to navigate towards the specific table.
private turn(degree: Integer): void
This function will turn relative to its current position the amount desired within the argument. The robot will use the references of the circles to help with alignment by knowing that every node is 90 degrees from one another.
private driveForward(void): void
This function will control the robot to move forward provided: $\forall frontultrasonicsensors : d_{ultrasonic} > d_{min}$. This motion will use the PI regulators to provide motion at human speed and will continue until a the next circle is within the middle of the camera.
private pathwayClear (void): boolean
This function will determine if robot to move forward provided: $\forall frontultrasonicsensors : d_{ultrasonic} > d_{min}$.

12.6.4 Communication Static Class

Uses None **Internal Variables**
None
Functions
getResturantMap(): String
Retrieves the map file based off of an FTP protocol and stores the map to be used for navigation. Returns the path of the map.
getTableOrder(): Order
Retrieves and returns the Table's Order.
sendErrors(errors: Integer)
Sends an integer with the described set of integers.
transferPump(Order): Integer
Performs communication with the pumping system using UART communication. The first 32 bits are the error code of the pumping system and the last bit is the status of the pumping system. Sends the Order data and receives the errors from the pumping system.

12.6.5 Graph Class

Uses

- Node (Class)

Internal Variables None
Functions
Graph(String Filename)
Constructor to create a graph object. Builds the graph based on the path to the map
getShortestPath(): Node[]
Returns a list of nodes that describes the shortest way to get to the destination. The shortest path will be preformed using the map and Dijkstra's algorithm.

12.6.6 Node Class

Uses None
Public Functions
Node(Int X, Int Y)
Constructor to the node class, takes in the position of the point along the X Y plane.
getX(): integer
Returns the x coordinate of the node.
getY(): integer
Returns the y coordinate of the node.
Internal Variables
x: Integer - The x coordinate of the node
y: Integer - The y coordinate of the node
Functions
public Node(Int X, Int Y)
Constructor to the node class, takes in the position of the point along the X Y plane.
public getX(): integer
Returns the x coordinate of the node
public getY(): integer
Returns the y coordinate of the node

12.6.7 PI Class

Uses None **Internal Variables Ki: Float** - Integral temp for the PI controller **Kp: Float** - The y coordinate of the node.
Functions
public PI(Ki: Float, Kp: Float)
Constructor for the PI controller object taking in the Ki and Kp terms
public regulate(Int ref, Int feedback)
Based on the reference to control to and the feedback, will determine the value to set the outputs too. Determines the output based along the following formula: $out = Kp * error + Ki * \int_0^t (error)dt$. Note that the derivative term is not used due to the error associated with derivatives within computing systems.

12.6.8 Imaging Processing Static Class

Uses

- Circle (Class)
- Open CV (External Library)

Internal Variables None
Functions
Public getCirclePositions(): Circle[]
Gives a list of circles that were within the view of the camera. Open CV returns a list of objects which can then be checked to see if they are black circles.

12.6.9 Circle Class

Uses None **Internal Variables**
rad: Float - The radius of the circle
x: Integer - The X position of the circle relative to the image
y: Integer - The Y position of the circle relative to the image
Functions
Circle(radius, x, y)
Constructor of the circle class, takes in initial radius, x and y values
getX(): Float
Gives the X position of the circle relative to the image
getY(): Float
Gives the Y position of the circle relative to the image
getRadius(): Float
Gives the radius of the circle
setX(X:Float)
Sets the X position of the circle.
setY(Y:Float)
Sets the Y position of the circle.
setRadius(rad : Float)
Sets the radius of the circle.

12.6.10 Raspberry Pi Pin Information

From	PIN # (physical)	GPIO # (BCM)	To	Comments
Raspberry Pi GND GPIO	6	-	H-Bridge GND	Ground for Motor controller
Raspberry Pi 5V GPIO	4	-	H-Bridge PWR	5V supply to H-bridge ***can utilize the 5v pin of the ultrasonic sensors instead of a new pin
Raspberry Pi GPIO	12	18	H-Bridge DIR 1	Direction of motor 1
Raspberry Pi GPIO	16	23	H-Bridge PWM 1	PWM of motor 1
Raspberry Pi GPIO	18	24	H-Bridge DIR 2	Direction of motor 2
Raspberry Pi GPIO	22	25	H-Bridge PWM 2	PWM of motor 2
Raspberry Pi GND GPIO	14	-	Ultra sonic sensors GND	Ground for all ultra sonic sensors ***can utilize GND from H-Bridge GND pin
Raspberry Pi 5V GPIO	2	-	Ultra sonic sensors VCC	5V supply for all ultrasonic sensors, connected parallel ***can utilize the 5v pin of the H-bridge supply instead of using a new pin
Raspberry Pi GPIO	24	8	Ultra sonic sensor #1 TRIG	Send trigger signal
Raspberry Pi GPIO	26	7	Ultra sonic sensor #1 ECHO	
Raspberry Pi GPIO	3	2	Ultra sonic sensor #2 TRIG	Send trigger signal
Raspberry Pi GPIO	5	3	Ultra sonic sensor #2 ECHO	
Raspberry Pi GPIO	7	4	Ultra sonic sensor #3 TRIG	Send trigger signal
Raspberry Pi GPIO	11	17	Ultra sonic sensor #3 ECHO	
Raspberry Pi GND GPIO	20	-	Encoders GND	Shared between encoders ***can utilize ultra sonic sensors' GND or H-Bridge GND instead of taking new pin
Raspberry Pi 3.3V GPIO	1	-	Encoders VCC	Shared between encoders in parallel ***can utilize 5V from either H-Bridge pin or ultra sonic sensors' pin instead of 3.3v
Raspberry Pi GPIO	13	27	Encoder #1 DT	
Raspberry Pi GPIO	15	22	Encoder #1 CLK	
Raspberry Pi GPIO	19	10	Encoder #2 DT	
Raspberry Pi GPIO	21	9	Encoder #2 CLK	
Raspberry Pi GPIO	8	14	Arduino	Communication between Arduino and Pi
Raspberry Pi GPIO	10	15	Arduino	Communication between Arduino and Pi
36V Positive terminal	-	-	H-Bridge power	Power supply to motors
36V Negative terminal	-	-	H-Bridge ground	Motor ground
Motor 1 positive terminal	-	-	H-Bridge Motor 1 +	Positive connection to controller
Motor 1 negative terminal	-	-	H-Bridge Motor 1 -	Negative connection to controller
Motor 2 positive terminal	-	-	H-Bridge Motor 2 +	Positive connection to controller
Motor 2 negative terminal	-	-	H-Bridge Motor 2 -	Negative connection to controller
Pi Camera bus terminals	-	-	Raspberry Pi camera bus terminal input	Communication between the pi camera and the raspberry pi

13 Pumping System

13.1 Purpose

The following will describe the component software, mechanical and electrical design associated with Alfred's Pumping System, Alfred's Drivetrain and Alfred's Image Processing system. This system will be ran on the Arduino Mega.

13.2 Scope

The scope of this section is associated with Alfred's Pumping System. The software documentation will provide the MIS and MID, uses relations to describe how the system will be designed to preform its function of being able to communicate to the raspberry pi and pump drinks. The mechanical and electrical design will focus on the different pumps/sensors that will be associated with the pumping system.

13.3 Module Decomposition

Alfred Pumping Module: Will control pumping system in regards of when to pour, how long and rate of dispensing. Will communicate to the raspberry pi errors pertaining to the pump or container to Alfred Manager Module. secrets include how the system preforms the dispensing of drinks and determination of errors.

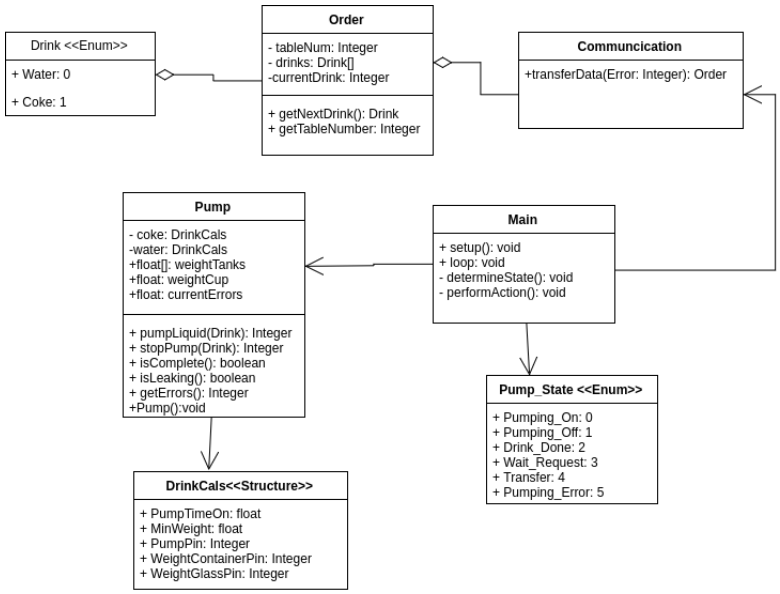

Figure 12: Uses Relation Diagram for the pumping system

13.4 Uses Relation

13.5 MIS

13.5.1 Main Class

Uses

- Order (Class)
- Drink (Enum)
- Pump (Class)
- Pump_State (Enum)
- Communication (Class)

Internal Variables None
Functions
public setup(): void
Setup function for Arduino that initializes the pins that will be used
publicloop(): void
Main loop that will preform the main logic of the pumping system
public determineState(): void
Based on the previous state of the pumping machine will look at factors such as weight of the liquid, need for communication and errors to determine the next state.
public performAction(): void
Based on the state of the device it will:

- Pumping_On: Turn on the voltage for the pump
- Pumping_Off: Turn off the voltage for the pump
- Wait_Request: Will Sleep for a specific amount of time before checking again
- Transfer: Preform transferring via UART
- Pumping_Error: Perform No Action, and ensure all pumping devices are off

13.5.2 Class Pump

Uses

- DrinkCals

public Functions
Pump():void
Initializes the values of the pumping module.
pumpLiquid(Drink): Integer
Turns on the pump for the specific drink type. Based on requirement AF1.
stopPump(Drink): Integer
Turns off the pump for the specific drink type. Based on requirement AF5.
isComplete(): Boolean
Determines if the drink has been completely filled or not based. Based on requirement AF5.
isLeaking(): Boolean
Determines if the containers are losing fluid when there is no pumping. Based on requirement NFR22.
isSafeTempature()
Determines if the containers are still storing the liquids are at a safe temperature. Based on requirement AF6.

13.5.3 Communication Class

Uses

- DrinkCals

Internal Values None
Functions
transferData(Integer): Order
Communication performed used where Orders are received and errors are transferred with the Manager system.

13.6 MID

13.6.1 Main Class

Uses

- Order (Class)
- Drink (Enum)
- Pump (Class)
- Pump_State (Enum)
- Communication (Class)

Internal Variables
None
Functions
public setup(): void
Setup function for Arduino that initializes the pins that will be used
public loop(): void
Main loop that will preform the main logic of the pumping system
public determineState(): void

Previous State	Conditions		New State
Pumping_On	Errors ==0	Time<TimeOff	Pumping_On
		Time>=TimeOff	Pumping_Off
	Errors !=0		Pumping_ Error
Pumping_Off	Errors ==0	M_cup >= M_Min	Drink_Done
		Time<TimeOn	Pumping_Off
		Time>=TimeOn && M_cup < M_Min	Pumping_On
	Errors !=0		Pumping_ Error
Wait_Request	Order==null		Wait_Request
	Order!=null		Pumping_On
Drink_Done	Order.getNextOrder() == null		Wait_Request
	Order.getNextOrder() != null && !CupTaken		Pumping_On
	Order.getNextOrder() != null && CupTaken		Drink_Done
Pumping_Error	Errors !=0		Pumping_Error
	Errors ==0		Wait_Request

Based on the previous state of the pumping machine will look at factors such as weight of the liquid, need for communication and errors to determine the next state. Which state is summarized in the following table:

public performAction(): void
Based on the state of the the pumping system, will preform the following actions:

Previous State	Action
Pumping_On	V_Pump[tank_gpio] = ON
Pumping_Off	V_Pump[tank_gpio] = OFF
Wait_Request	TransferData()
Drink_Done	
Pumping_Error	isLeaking() || isSafeTempature()

13.6.2 Pump Class

Uses

- DrinkCals

Internal Values
DrinkCals coke: Structure holding the calibrations related to Coke products
DrinkCals water: Structure holding the calibrations related to Water
Functions
public Pump():void
Initializes the values of the pumping module.
public pumpLiquid(Drink): Integer

Turns on the pump for the specific drink type.
public stopPump(Drink): Integer
Turns off the pump for the specific drink type.
public isComplete(): Boolean
Determines if the drink has been completely filled or not based on the following equation: $Filled := M_min < M_cup$
public isLeaking(): Boolean
Determines if the containers are losing fluid when there is no pumping based on the following equation: $Leaking := [M_minleak < (M_container1 - M_container_prev1) \wedge Pin7 == 0] \vee [M_minleak < (M_container2 - M_container_prev2) \wedge Pin8 == 0]$
public isSafeTempature()
Determines if the containers are still storing the liquids at a temperature greater then the minimum temperature for the liquids based off of the following equations. $OverTempature := (T_container1 < T_min) \vee (T_container2 < T_min)$

13.6.3 Communication Class

Uses

- DrinkCals

Internal Values None
Functions
transferData(Integer): Order
Communication performed used using UART where Orders are received and errors are transferred. The first 32 bits will be the errors associated with the pumping system and the last bit will be if the robot is done.

14 Server

14.1 Purpose

The following will describe the component software design associated with the server. This system will be run on an external RHEL server.

14.2 Scope

The scope of this section is associated with the REST API server that will be responsible for the communication within the system. The software documentation will provide the MIS and MID, uses relations to describe how the system will be designed to perform its function of being able to route communication between difference modules in the system.

14.3 Module Decomposition

Serer Module: Will have different REST API endpoints available to be able to "perform actions" in different parts of the system, as well as route data.

14.4 Uses Relation

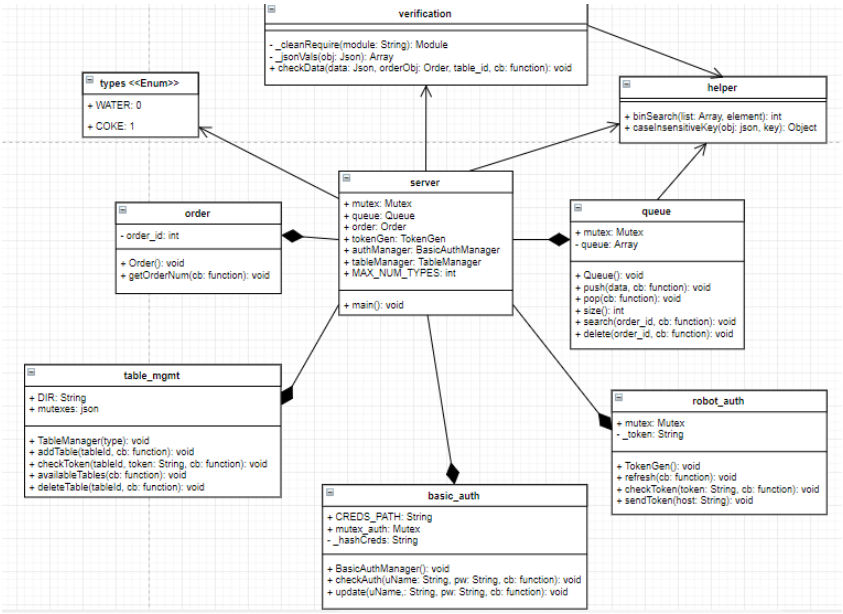

Figure 13: Alfred Uses Relation Diagram

14.5 MIS

14.5.1 server

Uses

- order (Class)
- types (Enum)
- verification (Class)
- helper (Class)
- queue (Class)
- robot_auth (Class)
- basic_auth (Class)
- table_mgmt (Class)
- http (External Class)
- url (External Class)
- fs (External Class)
- locks (External Class)

Public Functions main(): void Running the server, intercepting REST API requests and taking required action.

14.5.2 queue

Uses

- locks (External Class)
- helper (Class)

Public Functions Queue(): void Constructor function that creates a new empty queue.
push(data, cb): void Push new order information into the queue and callback function with the place in line.
pop(cb): void Pops the first element from the queue.
size(): int calls the callback function with the current size of the queue.
search(order_id, cb): void Searches the queue and calls callback function with the place in line for the order specified by the order_id.
delete(order_id, cb): void Searches the queue for order identified by given order_id and removes it from the queue.

14.5.3 helper

Uses None
Public Functions binSearch(list, element): int Binary search algorithm to find an element and return its position. If element not found, returns -1.
caseInsensitiveKey(obj, key): obj[i] Extracts information from a JSON object for a given key. Comparison between the key and the JSON object's keys are case insensitive.

14.5.4 basic_auth

Uses

- fs (External Class)
- locks (External Class)
- util (External Class)
- crypto (External Class)

Public Functions BasicAuthManager(): void Constructor function to create the object.
checkAuth(uName, pw, cb): void Will check the validity of the username and password given as parameters, and will call the callback function with a boolean of whether username/password combination is valid.
update(uName, pw, cb): void Will update the stored username and password combination, with the given ones.

14.5.5 robot_auth

Uses

- locks (External Class)
- rand-token (External Class)
- unirest (External Class)

Public Functions TokenGen(): void Constructor function to create the object with new token.
refresh(cb): void Updates current token.
checkToken(token, cb): void Check that the given token in the parameters matches the token stored in the object, and calls the callback function with boolean result.
sendToken(host): void Will send current token to the 'host' given in the parameters.

14.5.6 verification

Uses

- helper (Class)
- types (Enum)

Public Functions checkData(data, orderObj, table_id, cb): void Verify that the order information in the HTTP request has all necessary information and is in the correct format. Information required includes drink types, sizes, and quantities. Then calls callback function with error, if one exists.

14.5.7 table_mgmt

Uses

- locks (External Class)
- rand-token (External Class)
- fs (External Class)
- crypto (External Class)

Public Functions TableManager(): void Constructor function that reads all currently listed tables from the tables filesystem and creates a mutex for each.
addTable(tableId, cb): void Register a new table with the filesystem and generate an authentication token for that table to be used with server communication, then calls the callback function with the token.
checkToken(tableId, token, cb): void Verify that an inputted token is correct for the table, and calls callback function with boolean result.
availableTables(cb): void Search the filesystem to find all tables currently registered, and call callback function with list of all tables.
deleteTable(tableId, cb): void Remove a table from the filesystem.

14.5.8 order

Uses

- locks (External Class)

Public Functions Order(): void Constructor function that creates a new Order object, with order_id == 0.
getOrderNum(cb): void Call the callback function with the next order_id and increment the counter.

14.6 MID

14.6.1 server

Uses

- order (Class)
- types (Enum)
- verification (Class)
- helper (Class)
- queue (Class)
- robot_auth (Class)

- basic_auth (Class)
- table_mgmt (Class)
- http (External Class)
- url (External Class)
- fs (External Class)
- locks (External Class)

Internal Values

- Mutex mutex: Mutex to synchronize requests updating value of 'types.json' in the filesystem.
- Queue queue: A FIFO queue where orders are stored.
- Order order: Object to keep track of order ID's.
- TokenGen tokenGen: A token generator and manager for the robot token for authentication.
- BasicAuthManager authManager: Object responsible for authenticating administrator.
- TableManager tableManager: Object responsible for authenticating each table.
- final int MAX_NUM_TYPES: Variable storing maximum number of types of drinks that the robot can hold.

Functions main(): void

GET	POST	DELETE
placeinline	placeorder	cancelorder
nextorder	gentoken	drinks
checktoken	updatecreds	tables
drinks	returntobase	
sizes	login	
numoftanks	errors	
	drinks	
	tables	

Placeinline Call checkToken for table authentication If passed, search queue for order_id res.write(placeInLine.toString());
Placeorder Call checkToken for table authentication If passed, parse the body of the request Verify that all data is correct If correct, push order to queue
cancelOrder Call checkToken for robot authentication If passed, delete order from queue for specific order_id
Nextorder Call checkToken for robot authentication If passed, call queue's pop function res.write(placeInLine.toString());
Gentoken Call token refresh function Send token to robot host
Checktoken Call checkToken for robot authentication res.write(JSON.stringify(resp_auth));
Updatecreds Call checkAuth for admin authentication If passed, parse body of request Call authManager update function to update new user/password combination
Login Call checkAuth for admin authentication res.writeHead(200, 'Logged in', 'Content-Type': 'text/html'); if credentials were authenticated
Returntobase Call checkAuth for admin authentication Call returnToBase function to have robot return to the kitchen
Sizes Print out list of drinks that are available
Numoftanks Print out maximum number of tanks that the robot can hold

Errors Call checkToken for robot authentication If passed, list all error messages thrown by the robot
Drinks (GET) Print out list of drinks that are available
Drinks (POST) Call checkAuth for admin authentication Check that the total number of drink types is not already at the maximum Parse the body of the request Lock mutex Store new drink type and tank number into drinks object and write them to file Unlock mutex
Drinks (DELETE) Call checkAuth for admin authentication If passed, parse body of the request. Lock the mutex Delete specified drink type from DRINKS object. Write drink type list to file Unlock the mutex
Tables (POST) Call checkAuth for admin authentication If passed, call addTable function res.write(JSON.stringify(token: token, token_type: 'bearer'));
Tables (DELETE) Call checkAuth for admin authentication If passed, call deleteTable function

14.6.2 queue

Uses

- locks (External Class)
- helper (Class)

Internal Values

- Mutex mutex: Mutex used to lock the resource to prevent from problems arising from asynchronicity.
- Array queue: An array to store elements in queue.

Functions Queue(): void This.queue = []
push(data, cb): void Lock queue Add data to the queue Unlock queue cb(null, length of queue)
pop(cb): void Lock queue element = queue.pop() Unlock queue cb(null, element)
size(): int Return queue.length
search(order_id, cb): void index = helper.binSearch(queue, order_id) if (index >= 0) cb(null, (that.queue.length - index)); else cb('order_id not in queue');
delete(order_id, cb): void Lock queue index = helper.binSearch(queue, order_id) Remove element from queue at index if it exists cb()

14.6.3 helper

Uses None
Internal Values None
Functions binSearch(list, element): int Binary search algorithm based on element being the order_id. Return index if found or -1 if not found
caseInsensitiveKey(obj, key): obj[i] For k in keys of obj If k.toLowerCase() == key.toLowerCase() Return obj[k] Return null

14.6.4 basic_auth

Uses

- fs (External Class)
- locks (External Class)
- util (External Class)
- crypto (External Class)

Internal Values

- Mutex mutex_auth: Mutex to synchronize token from asynchronicity.

- final String CREDS_PATH: Path to store hashed credentials in.
- String _hashCreds: Hashed value of current user credentials.

Functions BasicAuthManager(): void If CREDS_PATH exists _hashCreds = readFile(CREDS_PATH) Else _hashCreds = hash('admin:admin') writeToFile(CREDS_PATH, _hashCreds)
checkAuth(uName, pw, cb): void mutex_auth.lock() passed = _hashCreds == hash(uName + ':' + pw) mutex_auth.unlock() cb(null, passed)
update(uName, pw, cb): void mutex_auth.lock() _hashedCreds = hash(uName + ':' + pw) writeToFile(CREDS_PATH, _hashedCreds) mutex_auth.unlock() cb()

14.6.5 robot_auth

Uses

- locks (External Class)
- rand-token (External Class)
- unirest (External Class)

Internal Values

- Mutex mutex: Mutex to synchronize token from asynchronicity.
- String _token: Value of current token to authenticate robot.

Functions TokenGen(): void _token = rand-token.generate(32)
refresh(cb): void mutex.lock() _token = rand-token.generate(32) mutex.unlock() cb()
checkToken(token, cb): void mutex.lock() passed = token == _token mutex.unlock() cb(null, passed)
sendToken(host): void data = token_type: 'bearer', access_token : _token unirest.post(host, data)

14.6.6 verification

Uses

- helper (Class)
- types (Enum)

Internal Values None
Functions checkData(data, orderObj, table_id, cb): void
order = helper.caseInsensitiveKey(data, 'order')
If order missing values or in wrong format
cb(error)

orders = []

For i of order
temp =
type = helper.caseInsensitiveKey(i, 'type')
size = helper.caseInsensitiveKey(i, 'size')
quantity = helper.caseInsensitiveKey(i, 'quantity')
If type is valid
temp.type = type
Else
Continue

If size valid
temp.size = size
Else
Temp.size = M
If quantity valid
temp.quantity = quantity
Else
temp.quantity = 1

orders.push(temp)

If no orders
Return cb(no valid 'orders')

orderObj.getOrderNum(function(err, order_id)
if (err)
return cb(err);

//return order information
cb(null,
table_id: table_id,
order_id: order_id,
orders: orders
);
);

_cleanRequire(module): Module
Delete cached data for module
Return require(module)

_jsonVals(obj): Array
ret = []
For i of Object.keys(obj)
ret.push(obj[i])
Return ret

14.6.7 table_mgmt

Uses

- locks (External Class)
- rand-token (External Class)
- fs (External Class)
- crypto (External Class)

Internal Values

- final String DIR: Directory to store hashed tokens for tables
- Json mutexes: A mutex for each table, to synchronize tokens from asynchronous calls.

Functions TableManager(): void Create DIR filesystem if not already created Read all files from filesystem, create a mutex for each, and load the filename/mutex pairs into a JSON object

addTable(tableId, cb): void Create new mutex for new table and add to JSON object Lock mutex for new table Create new file in tables filesystem Generate authentication token for table Write hashed token to corresponding file in tables filesystem Unlock mutex for table cb(null, token)
checkToken(tableId, token, cb): void Verify that an inputted token is correct for the table, and calls callback function with boolean result.
availableTables(cb): void Read all files from filesystem cb(null, files)
deleteTable(tableId, cb): void Lock mutexes JSON object Unlink table's path from filesystem Delete mutexes[tableId] Unlock JSON object cb()

14.6.8 order

Uses

- locks (External Class)

Internal Values

- int order_id: Current order number.

Public Functions Order(): void This.order_id = 0
getOrderNum(cb): void Lock counter Add one to counter Unlock counter cb(null, new order_id)

15 Scheduling

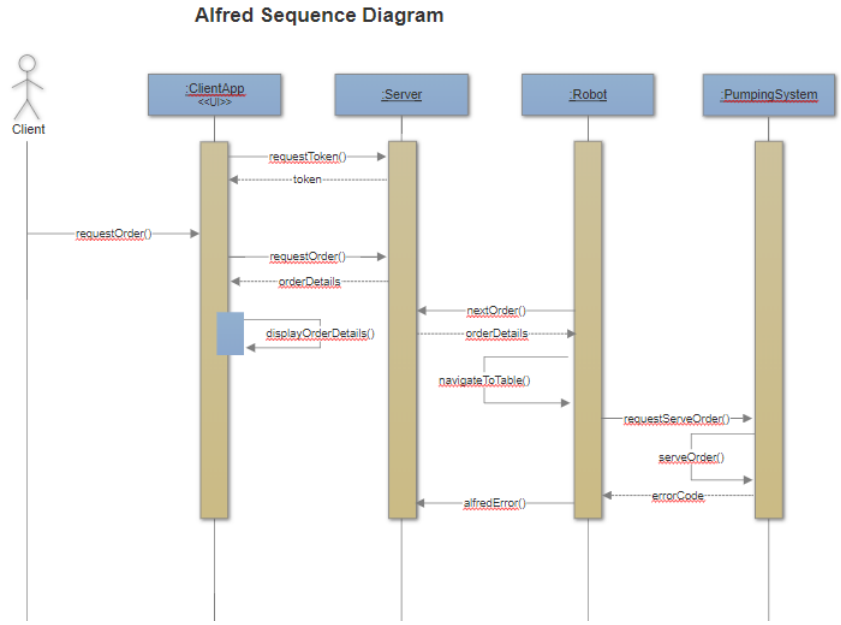

Figure 14: Alfred Sequence Diagram

16 Design Notes

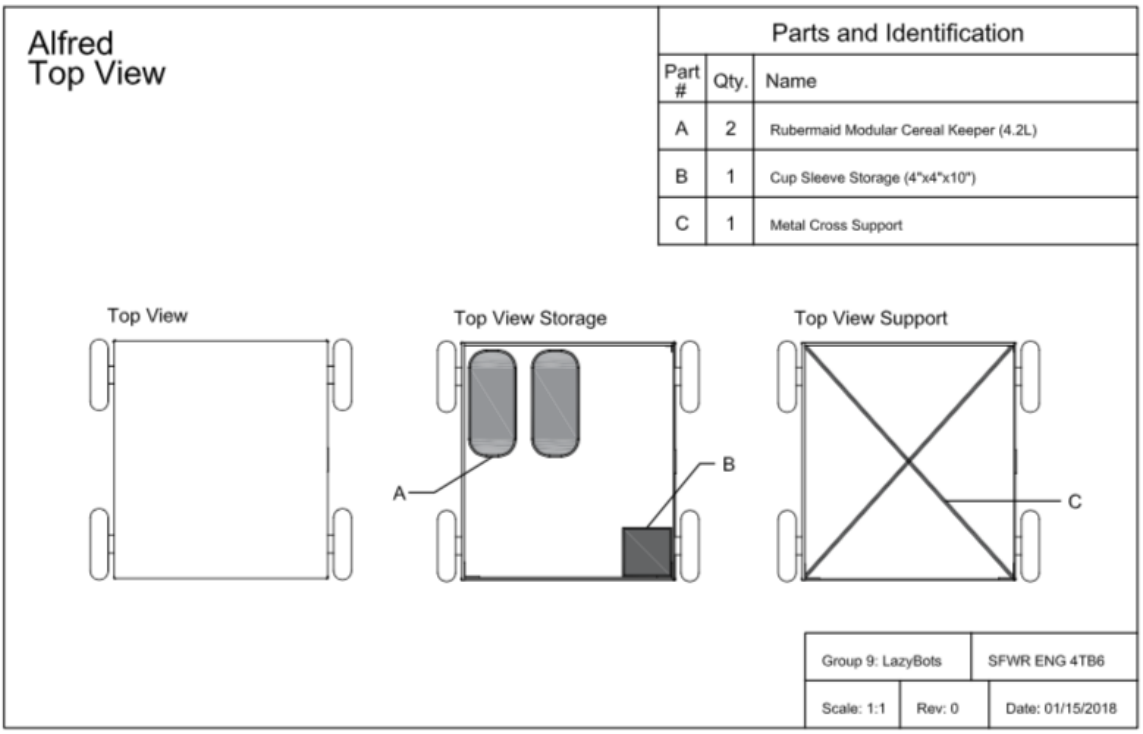

Figure 15: Engineering Model - Top View

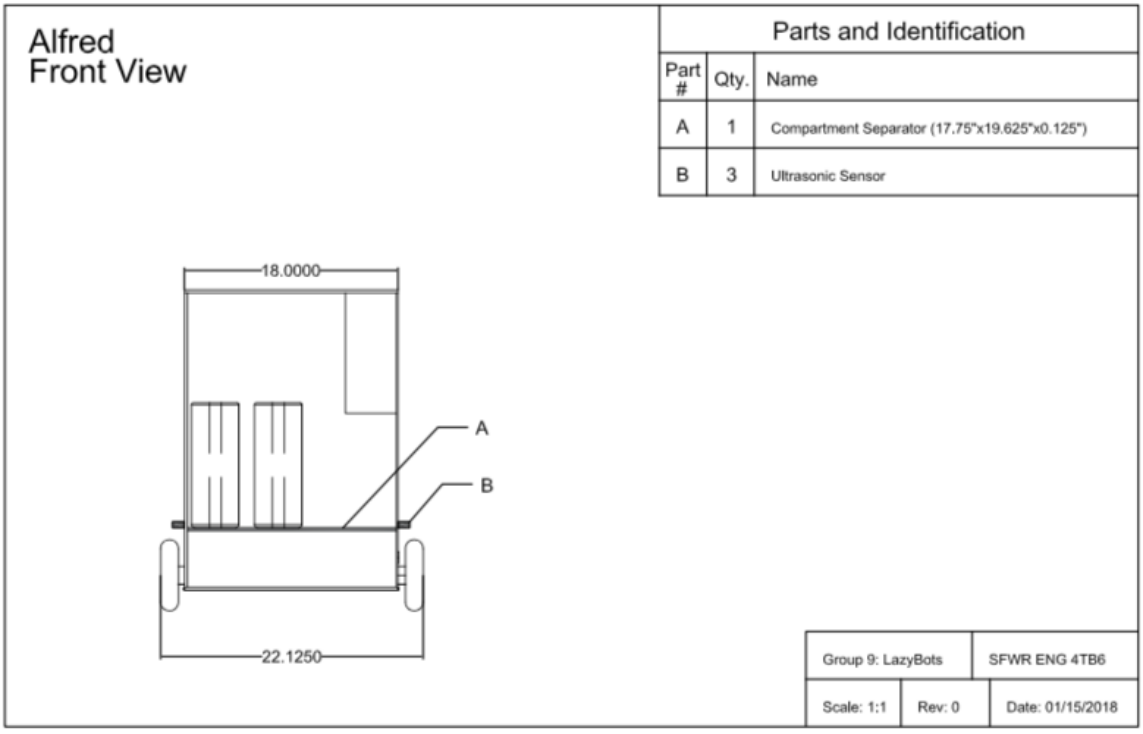

Figure 16: Engineering Model - Front View

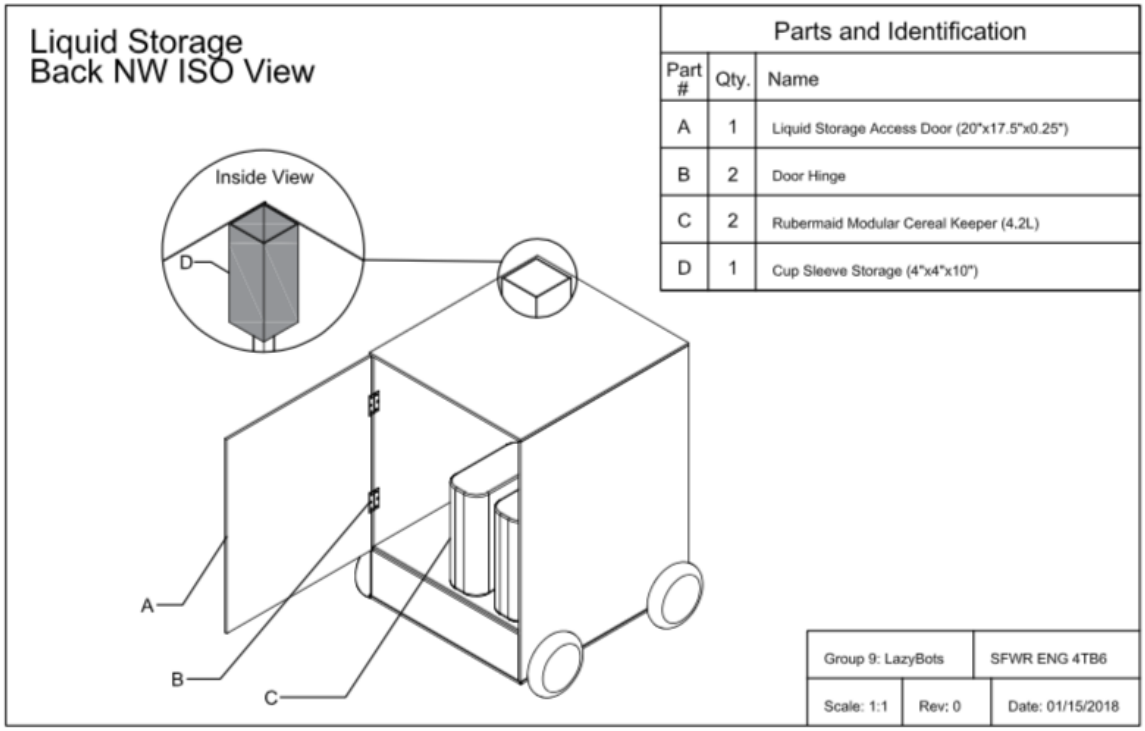

Figure 17: Engineering Model - Liquid Storage

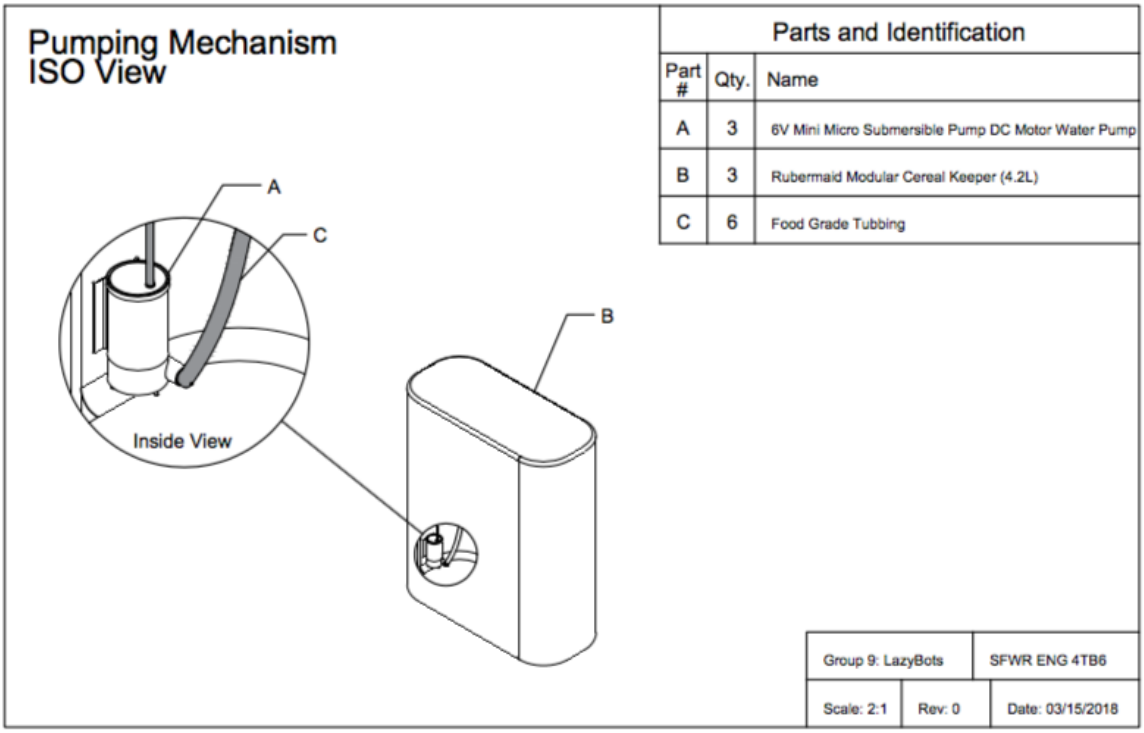

Figure 18: Engineering Model - Pumping System

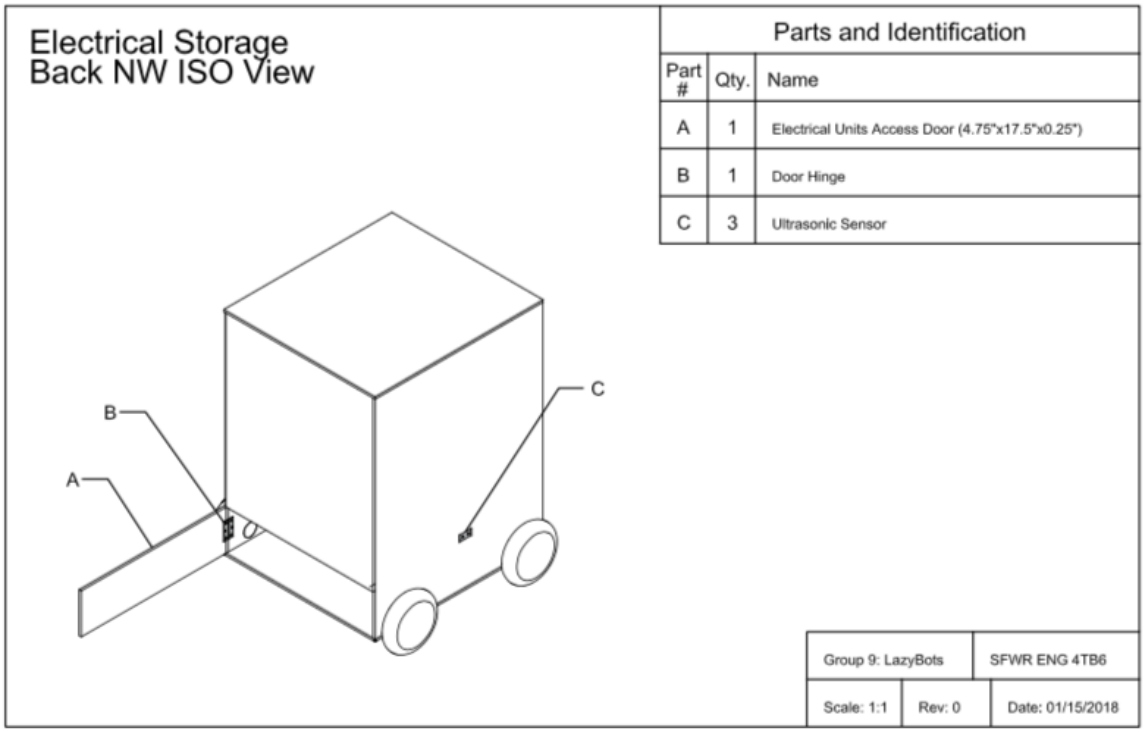

Figure 19: Engineering Model - Electrical Storage

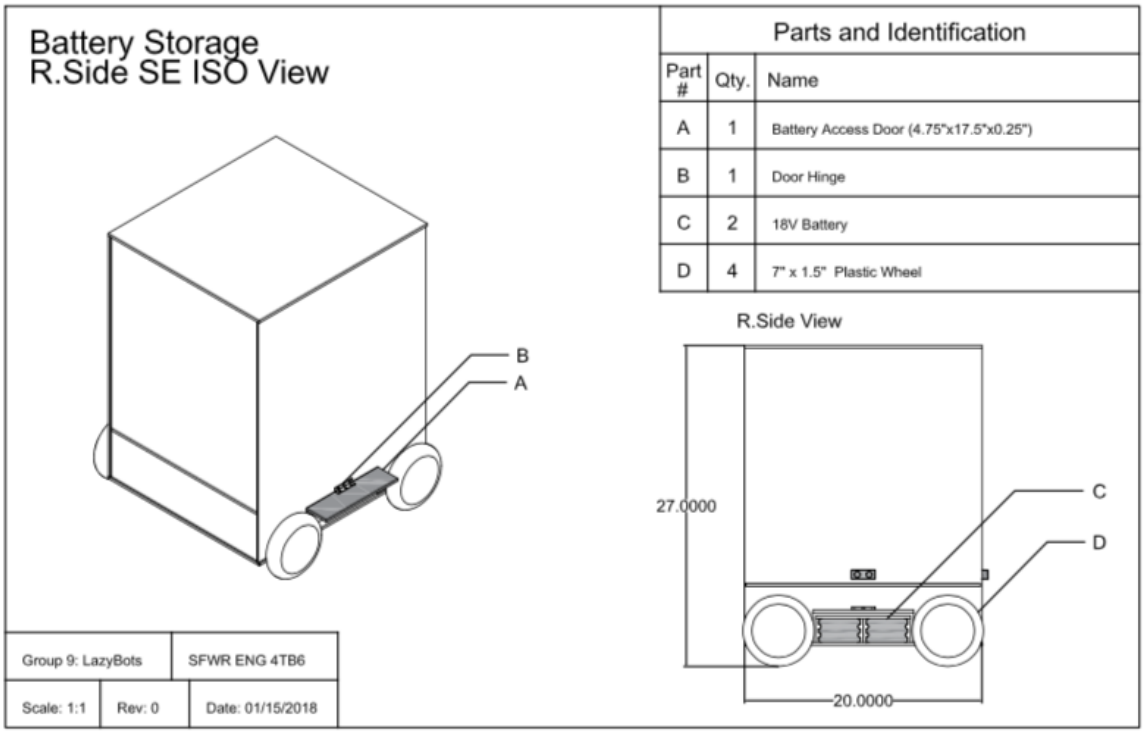

Figure 20: Engineering Model - Battery Storage

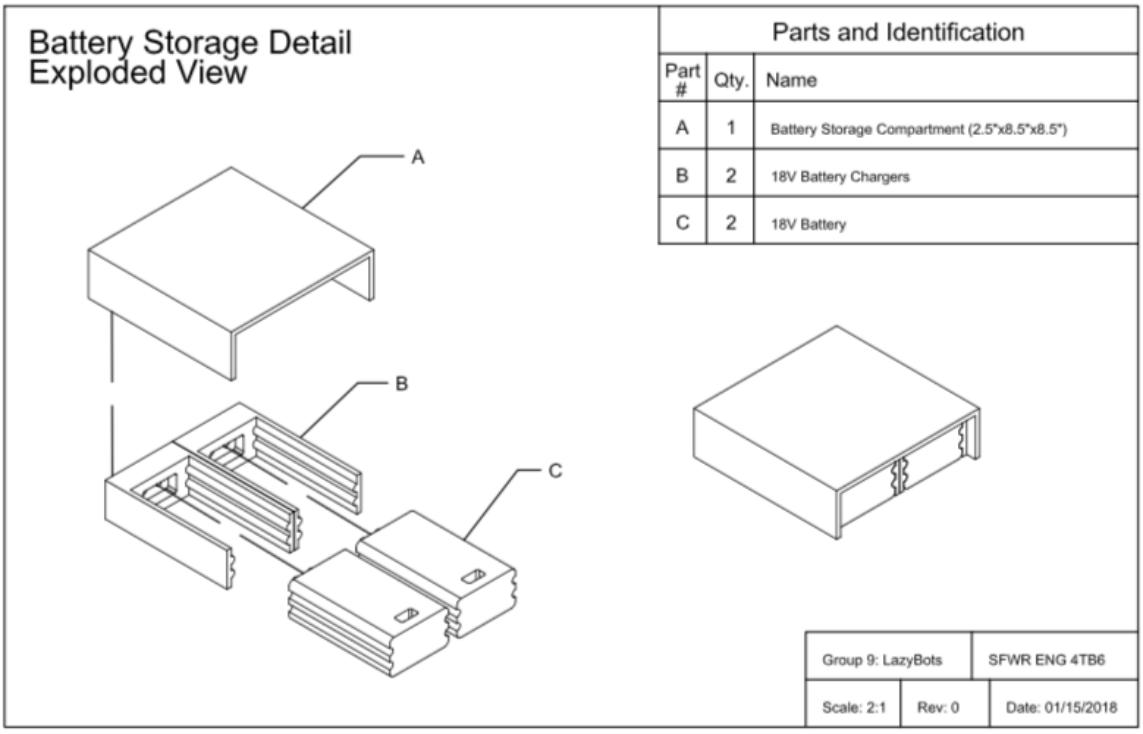

Figure 21: Engineering Model - Battery Compartment Exploded View

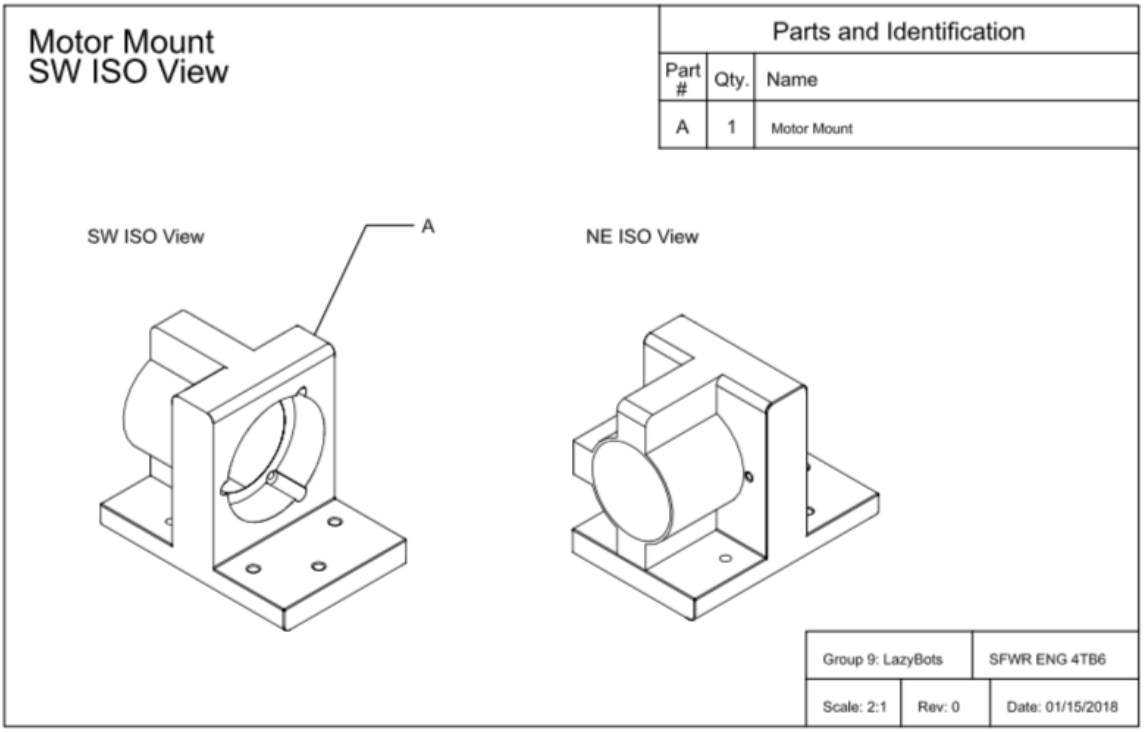

Figure 22: Engineering Model - Motor Mount

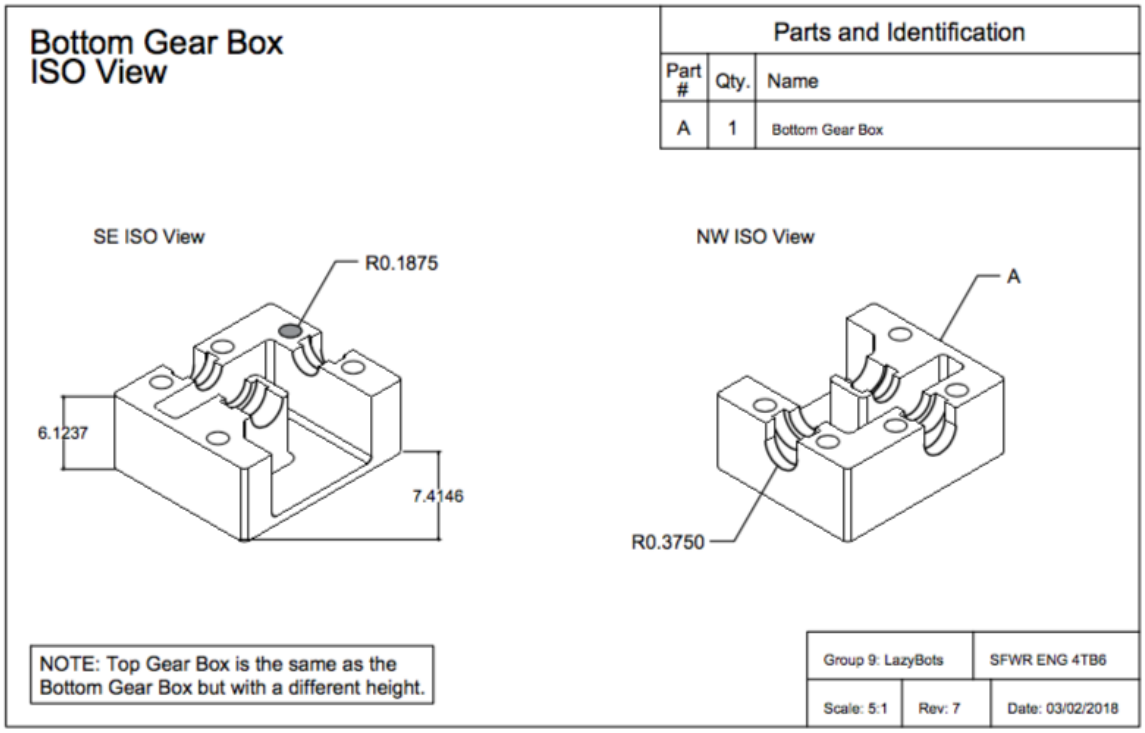

Figure 23: Engineering Model - Bottom Gear Box

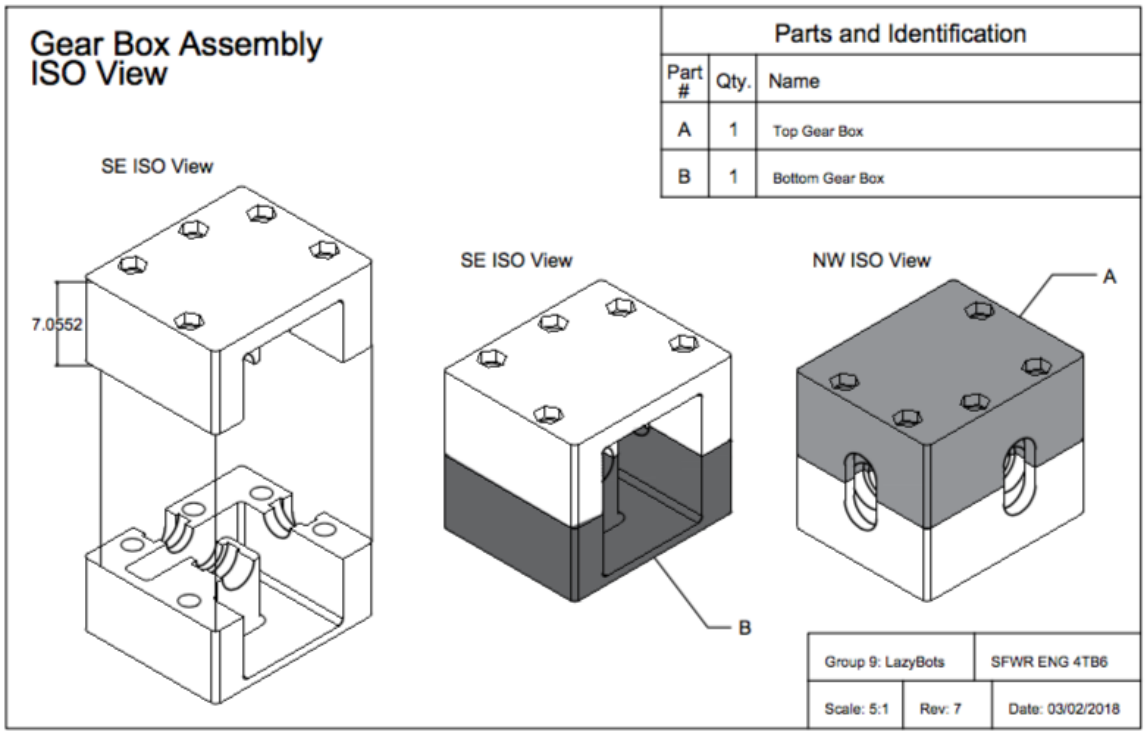

Figure 24: Engineering Model - Gear Box Assembly

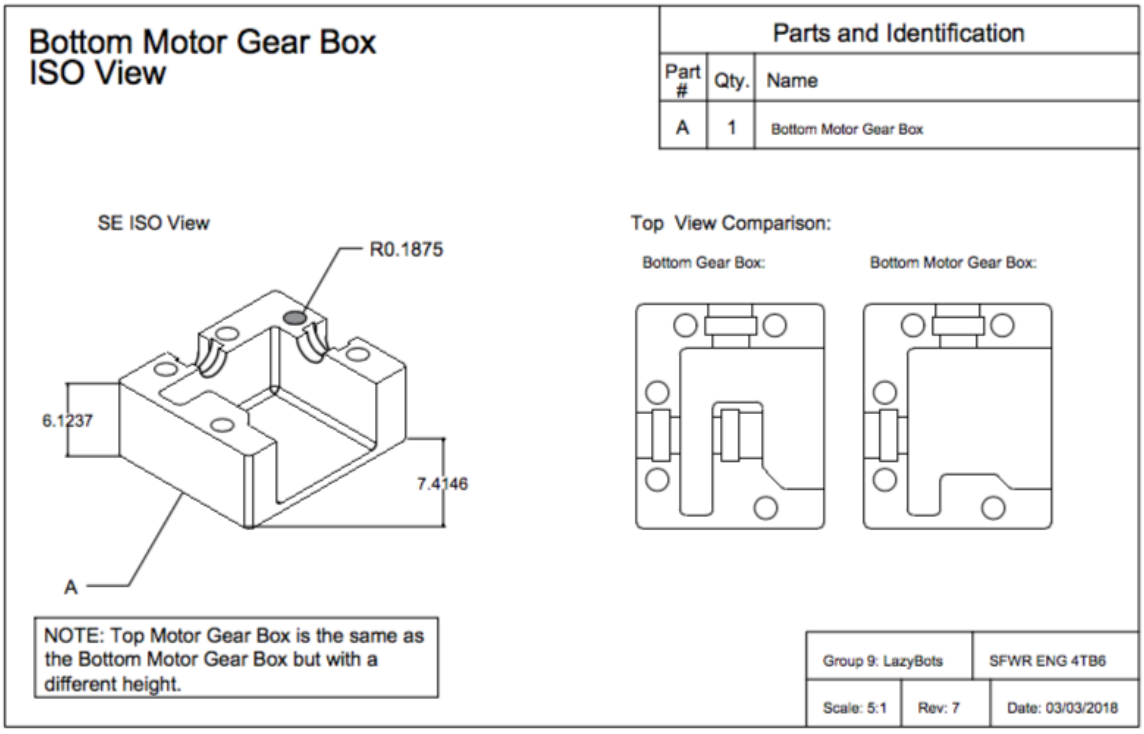

Figure 25: Engineering Model - Gear Box Assembly

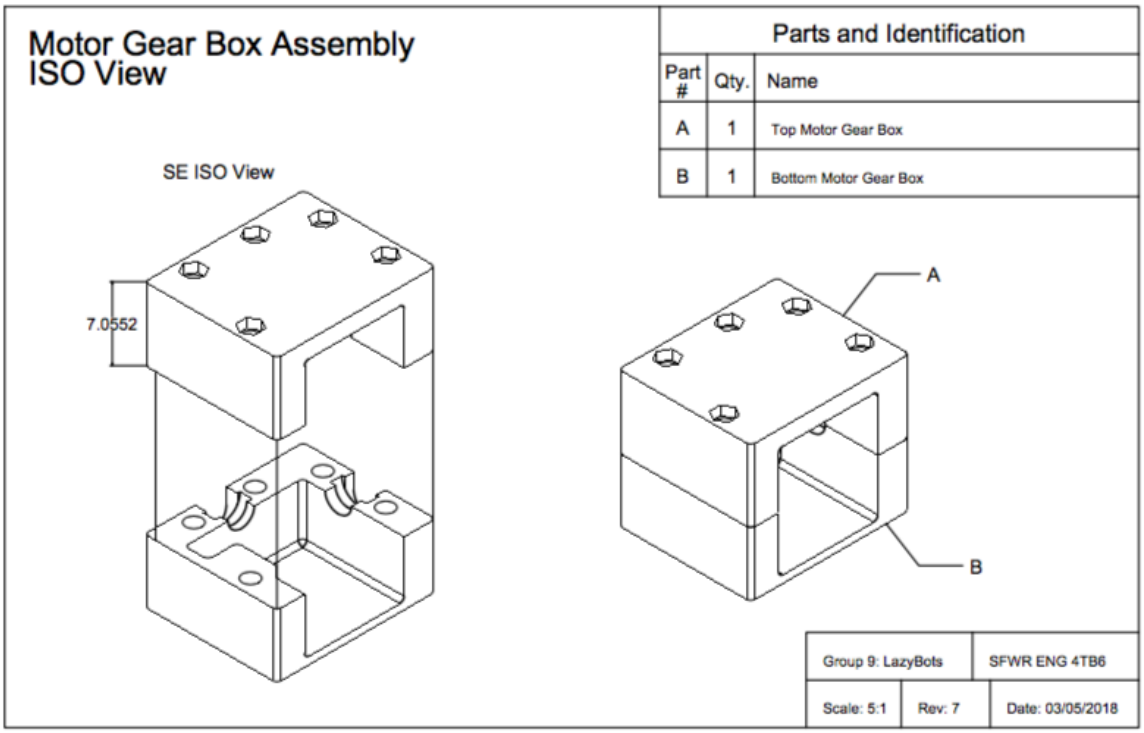

Figure 26: Engineering Model - Bottom Motor Gear Box

17 References

Raspberry pi data sheet

[1]"Exploring Raspberry Pi", 2017. [Online].
Available: http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf.
[Accessed: 23- Dec- 2017].

Arduino Mega Information

[2] "Arduino Mega". [Online]. Available:
https://www.arduino.cc/en/Main/ArduinoBoardMega. [Accessed: 23- Dec- 2017].

DC Pump Information

[3] "3-6V Mini Micro Submersible Pumps DC Motor Pump Water Pump - 80-120L/H". [Online].
Available:
https://www.amazon.ca/gp/product/B01LWQCXEL/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1.
[Accessed: 23- Dec- 2017].

Pump Container Information

[4] "Rubbermaid 1856059 Modular Cereal Keeper". [Online].
Available:
https://www.amazon.ca/gp/product/B00BEUDXRW/ref=oh_aui_detailpage_o09_s00?ie=UTF8&psc=1.
[Accessed: 23- Dec- 2017].

Raspberry camera pi data sheet

[4] "CAMERA MODULE". [Online]. Available:
https://www.raspberrypi.org/documentation/hardware/camera/. [Accessed: 23- Dec- 2017].