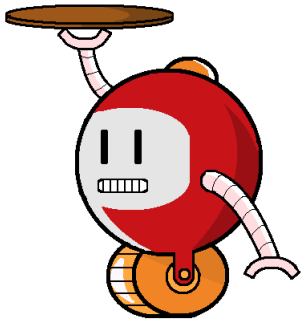

LazyBots

McMaster University

Draft Component Design

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

Table 1: VIC Table of Revisions

Date	Revision Number	Authors	Comments
November 24th, 2017	Revision 0	Karim Guirguis David Hemms Marko Laban Curtis Milo Keyur Patel Alexandra Rahman	-

2 ClientApp

2.1 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..

2.2 Scope

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

2.3 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: GUI class for users to go through available drinks and choose their order. Secrets include parsing drink info coming from the server. **Activity_OrderCart**: GUI class for users to view their current cart, as well as their previous orders. Secrets include parsing and dealing with serverâĂŹs 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 tableâĂŹs order to be sent to the server. **NetworkCalls**: Asynchronous class to perform network calls for the client app. Secrets include how network requests/responses are handled.

2.4 Uses Relation

Please refer to Figure 1.

2.5 MIS

2.5.1 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.
getCalories():int Returns the amount of calories for the restaurantâĂŹs 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.

2.5.2 DrinkOrder

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

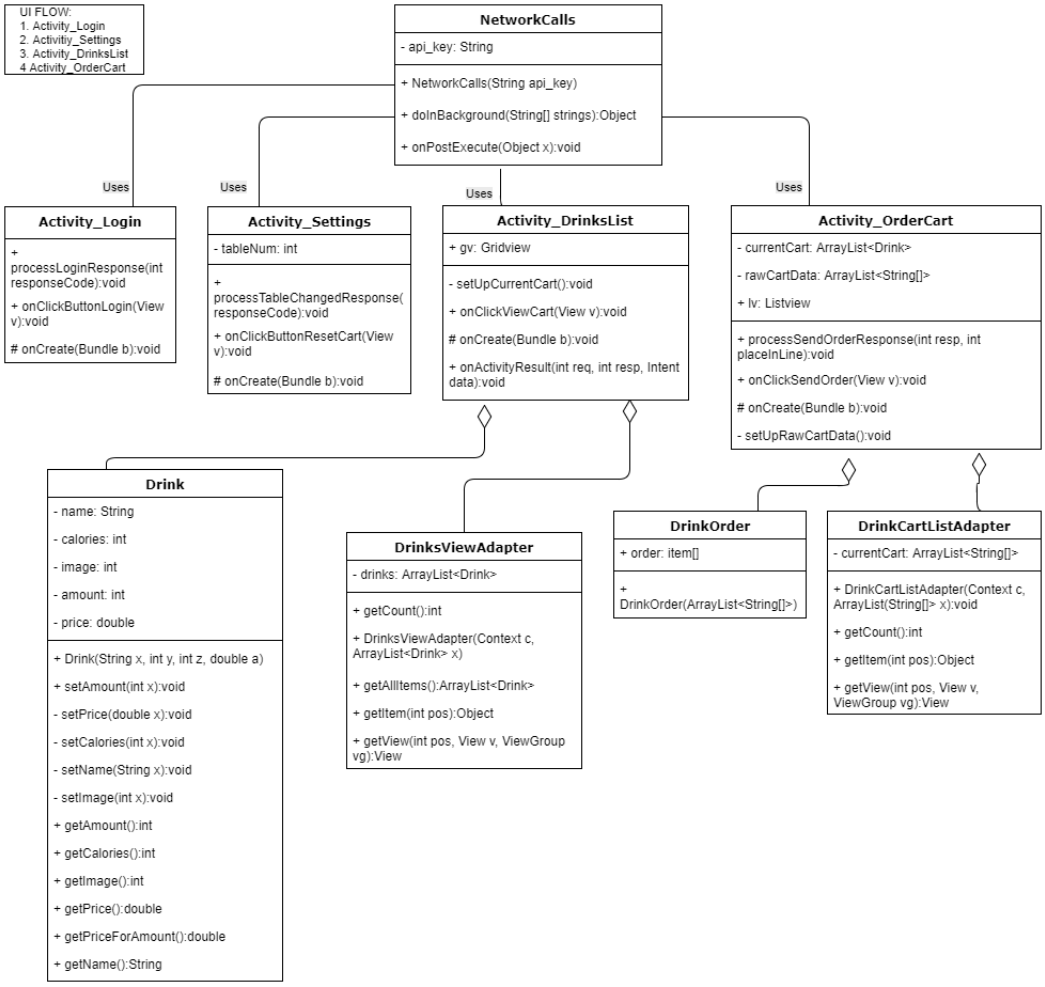

Figure 1: Android Client Application Uses Relation Diagram

2.5.3 Activity_DrinksList

Uses

- DrinksViewAdapter (Class)
- NetworkCalls (Class)

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.

2.5.4 Activity_Login

Uses

- NetworkCalls (Class)

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.

2.5.5 Activity_Settings

Uses

- NetworkCalls (Class)

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.

2.5.6 Activity_OrderCart

Uses

- DrinkCartListAdapter (Class)
- NetworkCalls (Class)

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.

2.5.7 NetworkCalls

Uses:None
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.

2.5.8 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.

2.5.9 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.

2.6 MID

2.6.1 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 restaurantâĂŹs 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 restaurantâĂŹs 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.

2.6.2 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.

2.6.3 Activity_DrinksList

Uses

- DrinksViewAdapter (Class)
- NetworkCalls (Class)

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.

2.6.4 Activity_Login

Uses

- NetworkCalls (Class)

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.

2.6.5 Activity_Settings

Uses

- NetworkCalls (Class)

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.

2.6.6 Activity_OrderCart

Uses

- DrinkCartListAdapter (Class)
- NetworkCalls (Class)

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]

2.6.7 NetworkCalls

Uses:None
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.

2.6.8 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.

2.6.9 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.

3 Manager System

3.1 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.

3.2 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.

3.3 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.

3.4 Uses Relation

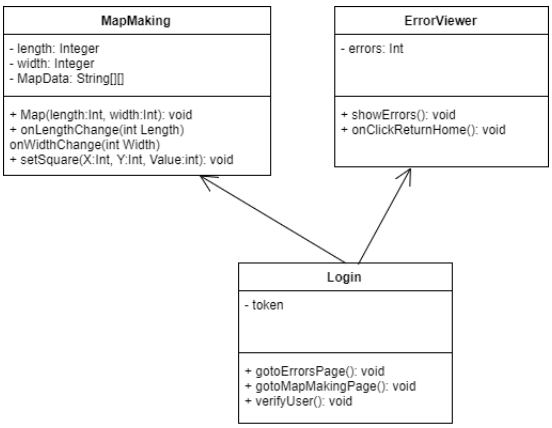

Figure 2: Alfred Uses Relation Diagram

3.5 MIS

3.5.1 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.

3.5.2 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.

3.5.3 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.

3.6 MID

3.6.1 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.

3.6.2 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

3.6.3 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.

4 Alfred System

4.1 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.

4.2 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.

4.3 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

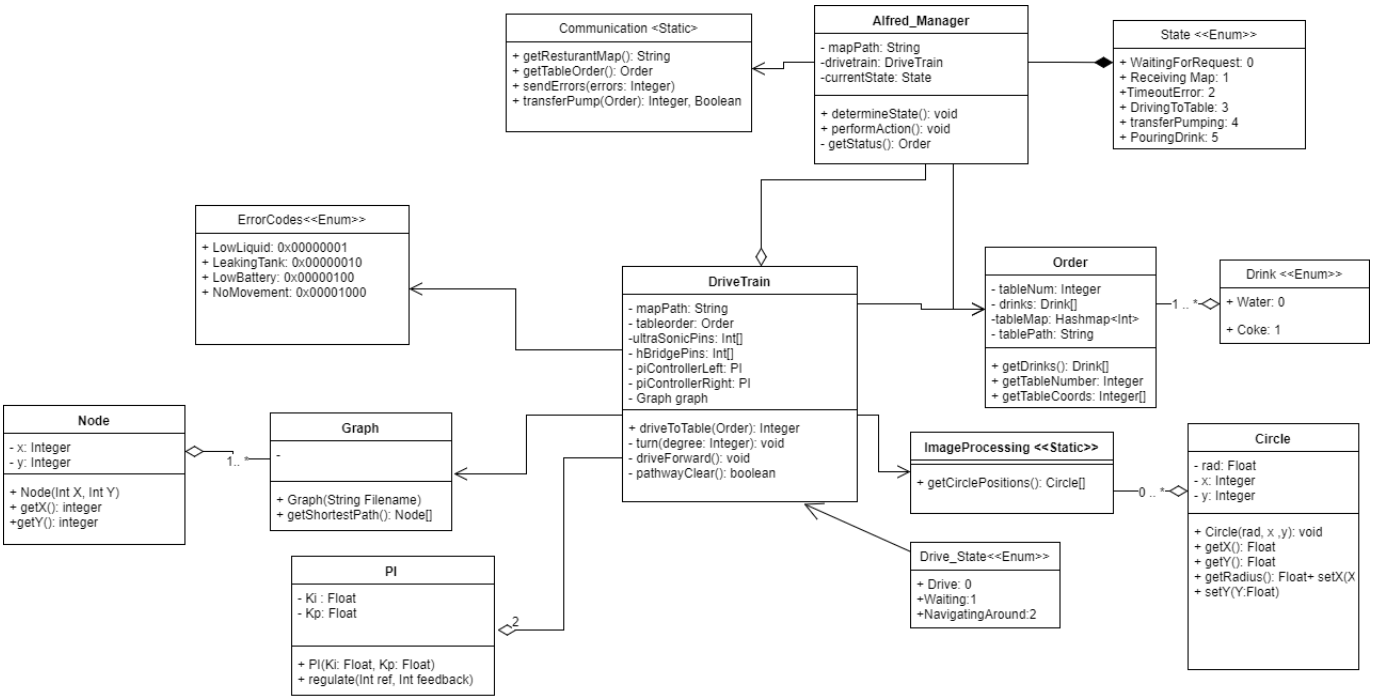

Figure 3: Alfred Uses Relation Diagram

4.4 Uses Relation

4.5 MIS

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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.

4.5.5 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.

4.5.6 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

4.5.7 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.

4.5.8 Imaging Processing Static Class

Uses

- Circle (Class)

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

4.5.9 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

4.6 MID

4.6.1 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)

4.6.2 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

4.6.3 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: $\forall frontultrasonicsensors : d_{ultrasonic} > d_{min}$.

4.6.4 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.

4.6.5 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.

4.6.6 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

4.6.7 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.

4.6.8 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.

4.6.9 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.

4.6.10 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 sensorsâĂŹ 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 sensorsâĂŹ 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

5 Pumping System

5.1 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.

5.2 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.

5.3 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.

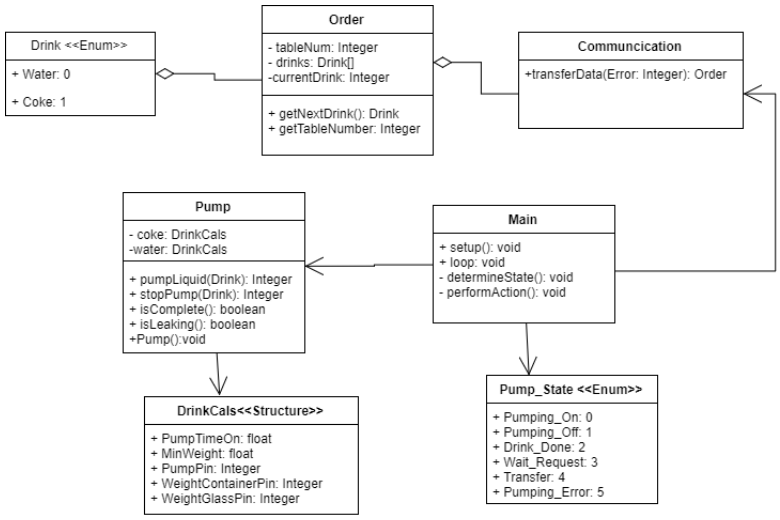

Figure 4: Uses Relation Diagram for the pumping system

5.4 Uses Relation

5.5 MIS

5.5.1 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

5.5.2 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.

5.5.3 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.

5.6 MID

5.6.1 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()

5.6.2 Class Pump

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. $OverTempature := (T_container1 < T_min) \vee (T_container2 < T_min)$

5.6.3 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.

6 Server

6.1 Purpose

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

6.2 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.

6.3 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.

6.4 Uses Relation

6.5 MIS

6.5.1 server

Uses

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

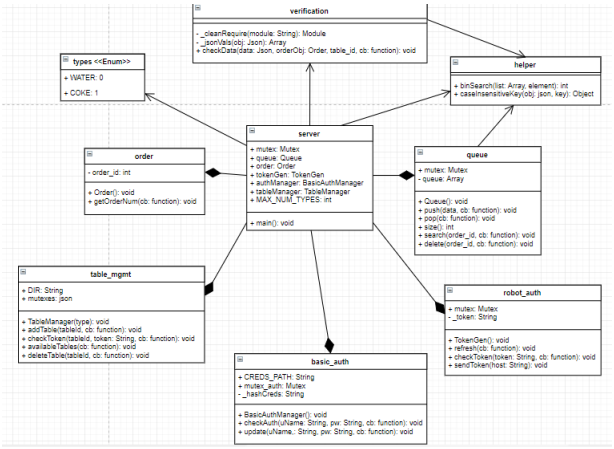

Figure 5: Alfred Uses Relation Diagram

- 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.

6.5.2 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.

6.5.3 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.

6.5.4 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.

6.5.5 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.

6.5.6 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.

6.5.7 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.

6.5.8 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.

6.6 MID

6.6.1 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 queueâĂŹs 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

6.6.2 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()

6.6.3 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

6.6.4 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(âĂŸadmin:adminâĂŹ) writeToFile(CREDS_PATH, _hashCreds)
checkAuth(uName, pw, cb): void mutex_auth.lock() passed = _hashCreds == hash(uName + âĂŸ:âĂŹ + pw) mutex_auth.unlock() cb(null, passed)
update(uName, pw, cb): void mutex_auth.lock() _hashedCreds = hash(uName + âĂŸ:âĂŹ + pw) writeToFile(CREDS_PATH, _hashedCreds) mutex_auth.unlock() cb()

6.6.5 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: âĂŸbearerâĂŹ, access_token : _token unirest.post(host, data)

6.6.6 verification

Uses

- helper (Class)
- types (Enum)

Internal Values None
Functions checkData(data, orderObj, table_id, cb): void order = helper.caseInsensitiveKey(data, âĂŸorderâĂŹ) If order missing values or in wrong format cb(error)
orders = []
For i of order temp = type = helper.caseInsensitiveKey(i, âĂŸtypeâĂŹ) size = helper.caseInsensitiveKey(i, âĂŸsizeâĂŹ) quantity = helper.caseInsensitiveKey(i, âĂŸquantityâĂŹ) 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(âĂŸno valid ordersâĂŹ)
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

6.6.7 table_mgmt

Uses

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

Internal Values

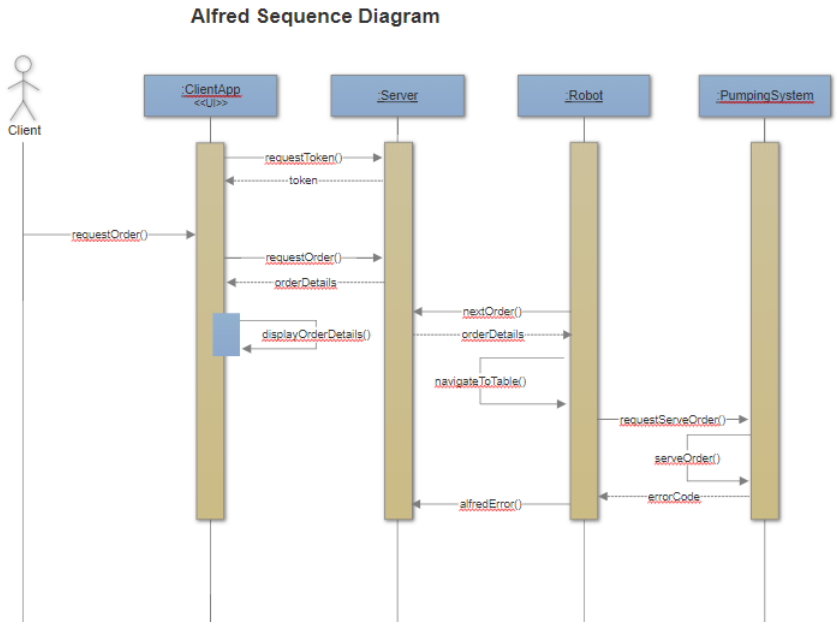

Figure 6: Alfred Sequence Diagram

- 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 tableâĂŹs path from filesystem Delete mutexes[tableId] Unlock JSON object cb()

6.6.8 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)

7 Scheduling

8 Design Notes

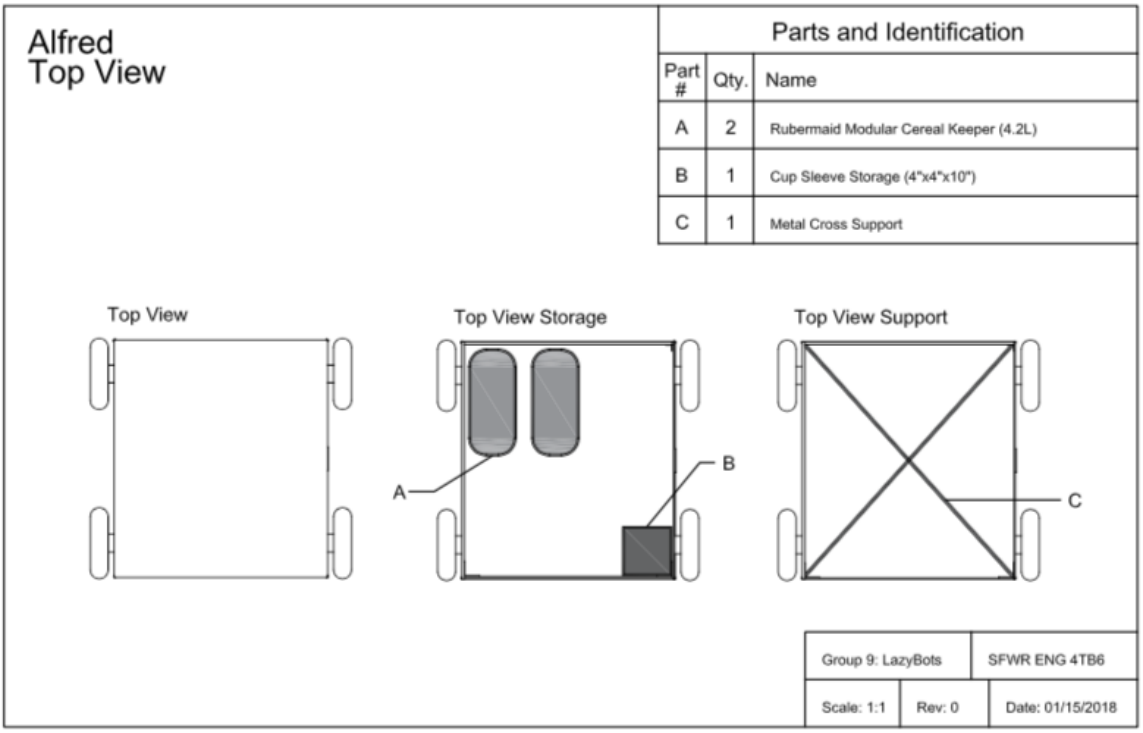

Figure 7: Engineering Model - Top View

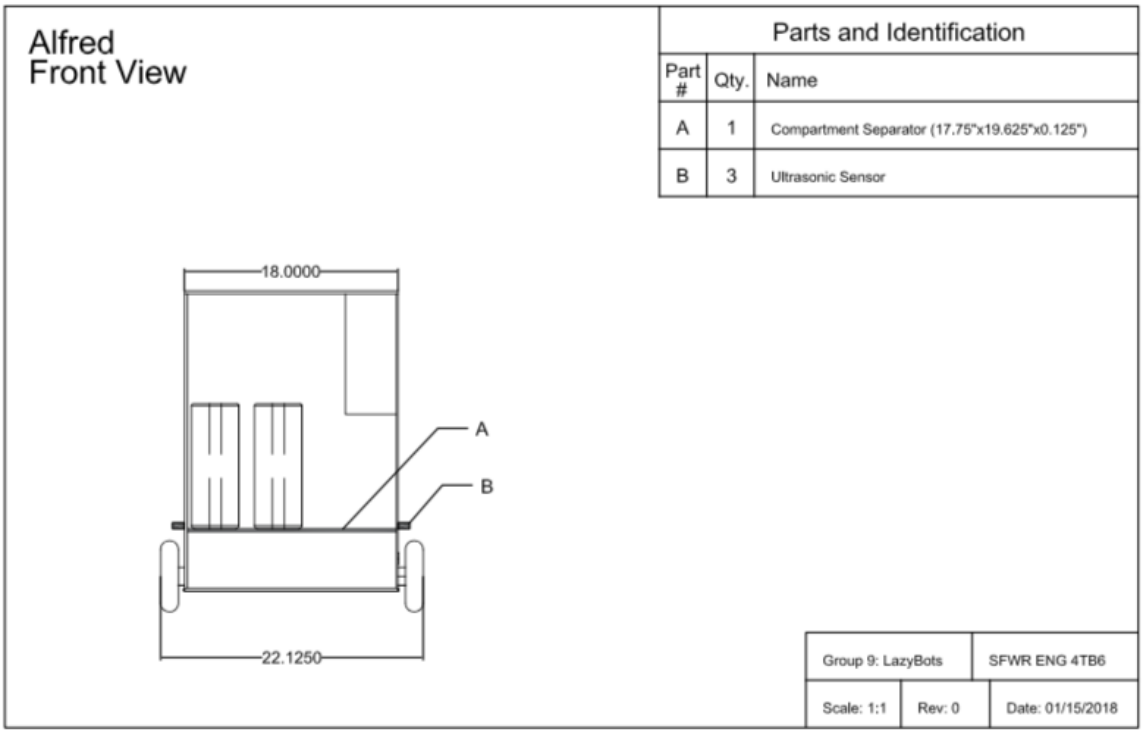

Figure 8: Engineering Model - Front View

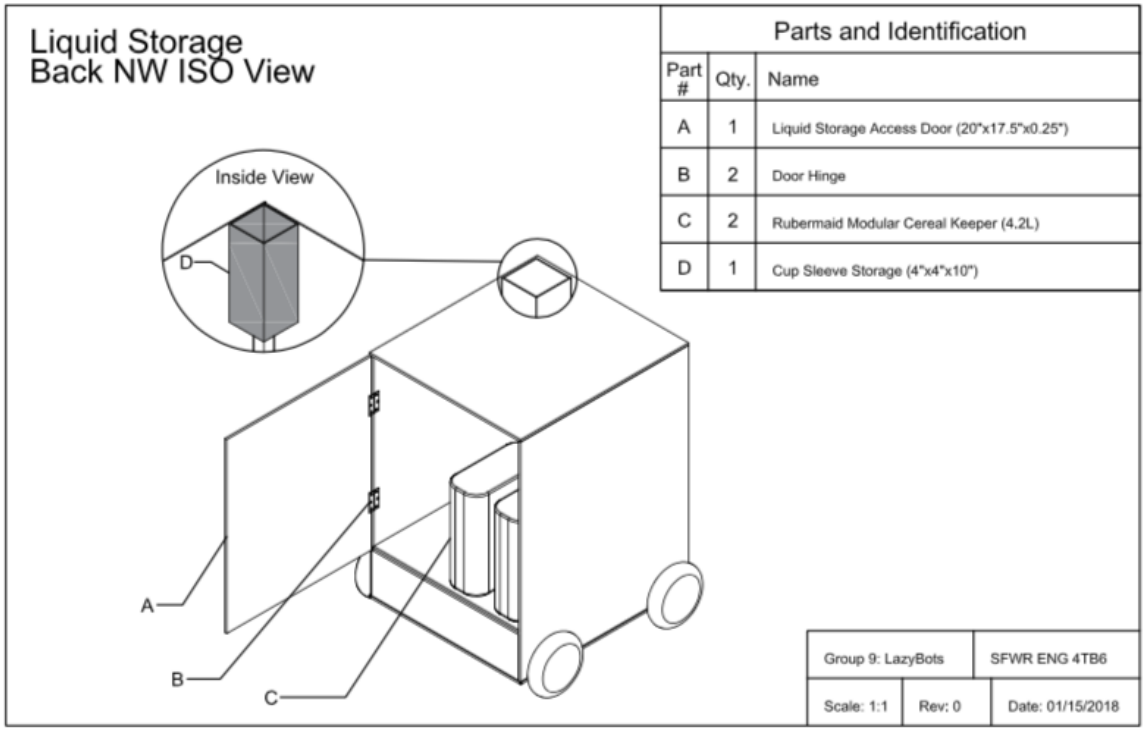

Figure 9: Engineering Model - Liquid Storage

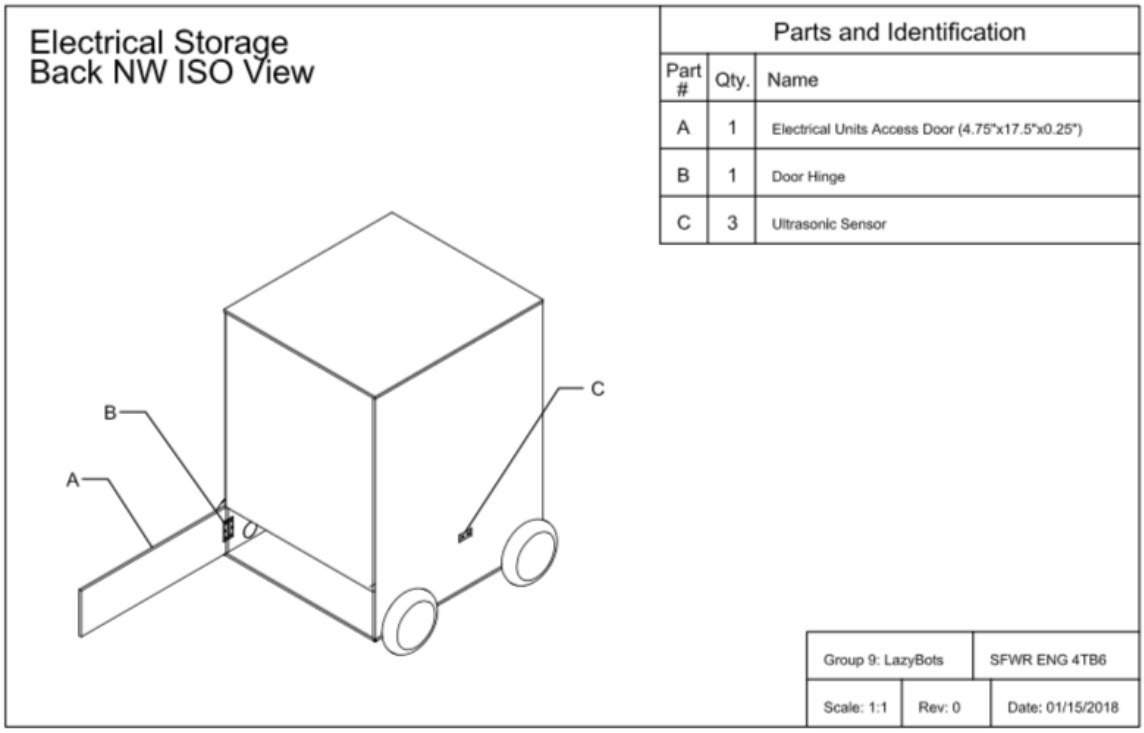

Figure 10: Engineering Model - Electrical Storage

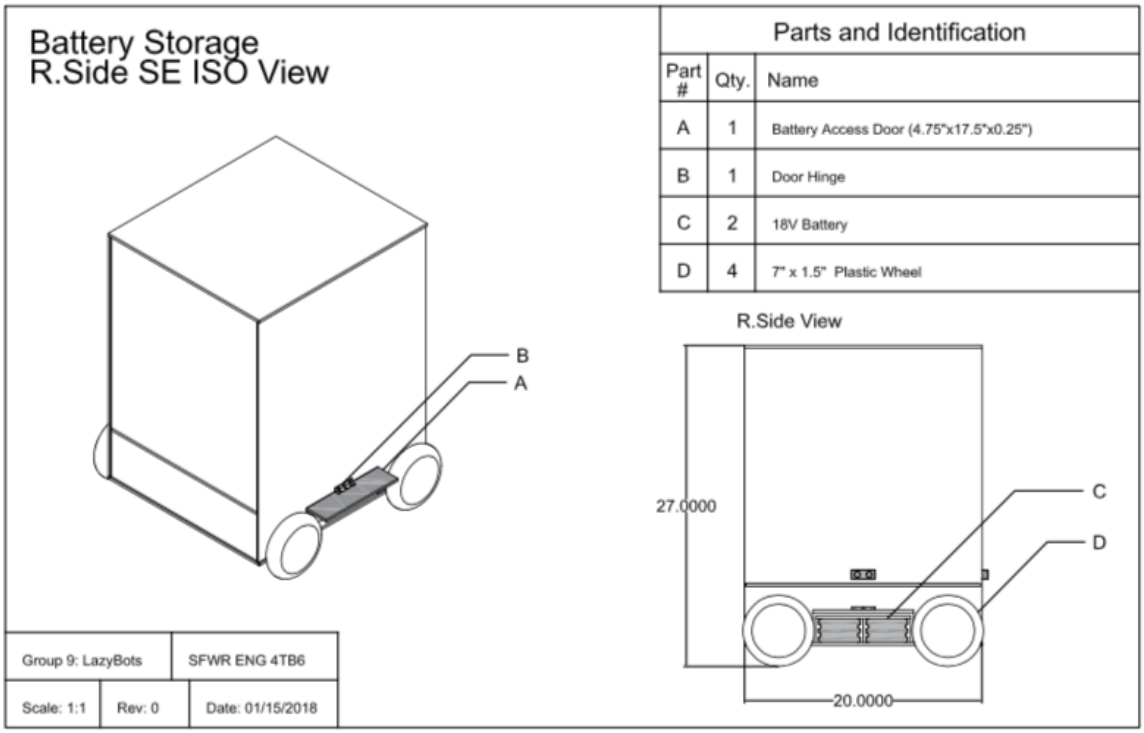

Figure 11: Engineering Model - Battery Storage

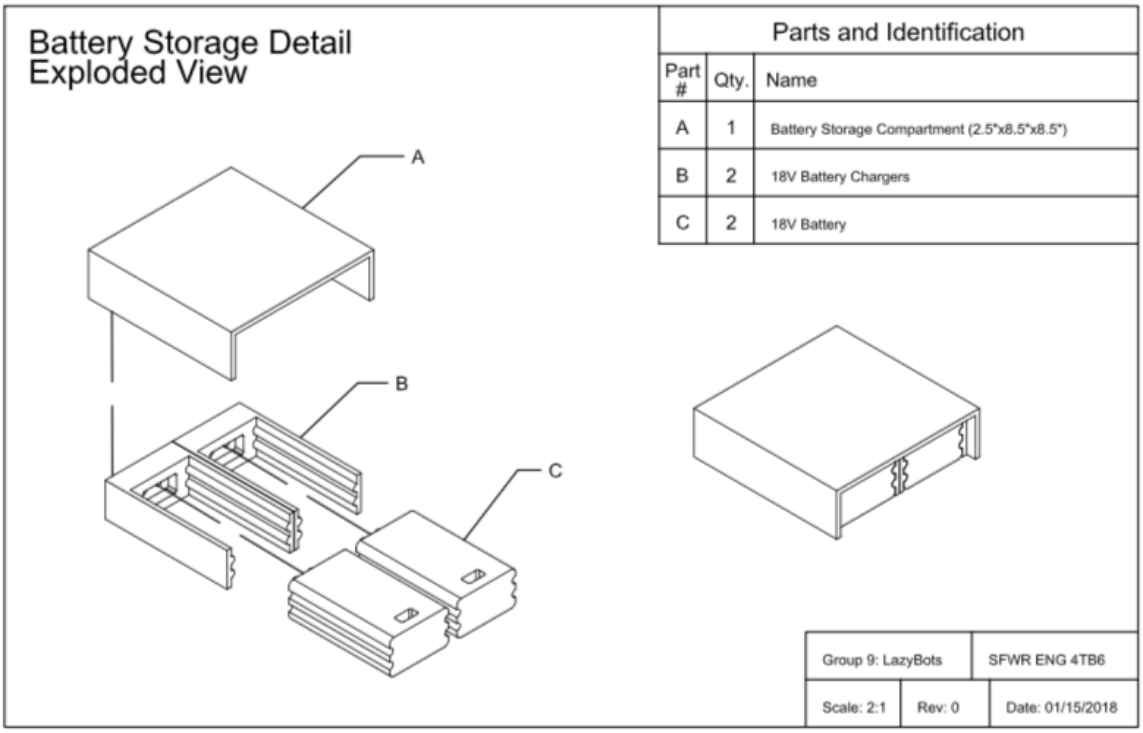

Figure 12: Engineering Model - Battery Compartment Exploded View

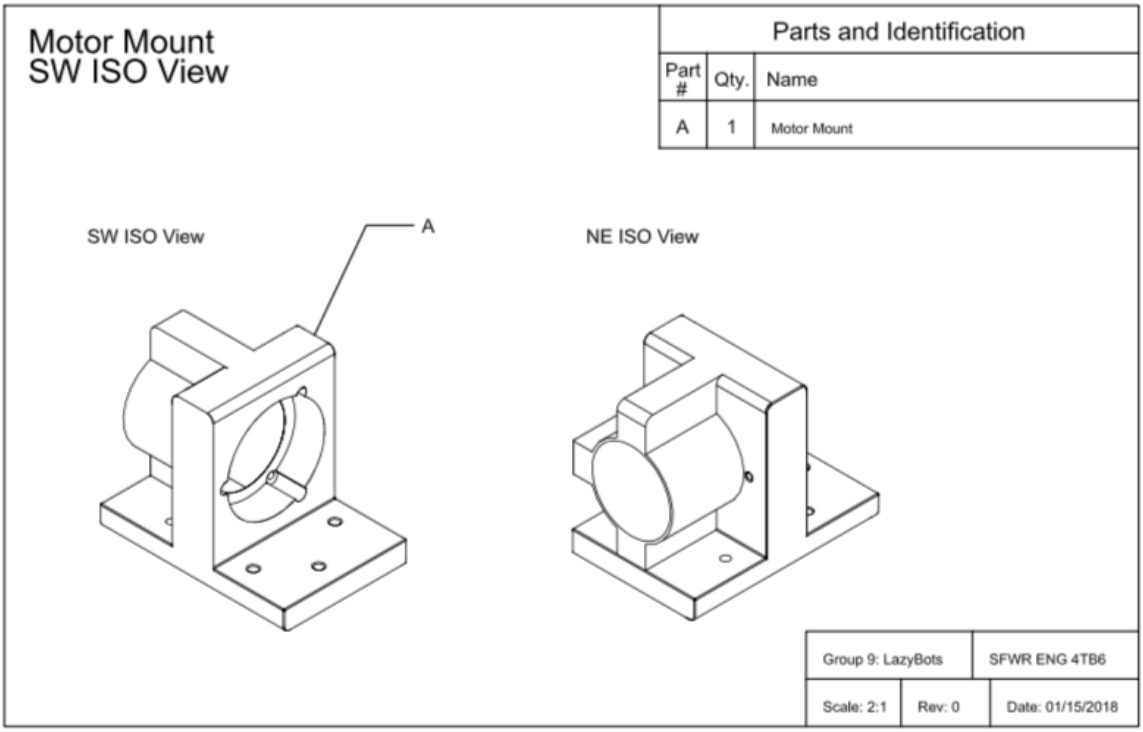

Figure 13: Engineering Model - Motor Mount