WaiterBot System

Design Manual

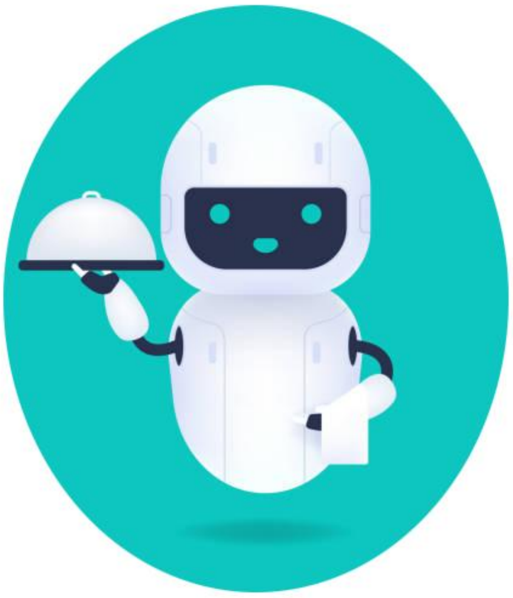


Table of Contents

1. Introduction

Robots have seen a wide array and continuous applications in various industries since their inception. The efficiency and versatility that robots possess can be molded into a vast area of services, and robots in restaurants can step up to be a huge breakthrough in terms of customer and owner satisfaction and improving the overall experience in diners.

The WaiterBot System is an automated system designed for placing and delivering orders in a restaurant. This system will replace the human waiters with robot waiters for an efficient delivery process and also give customers a new experience. Customers can place orders via the mobile application and once the orders are ready, the WaiterBots will deliver the orders to the customer.

Our solution is to replace the human waiter with a robot waiter and also to replace the traditional menu cards system with a more attractive and efficient order placing system. On an event where a customer visits the restaurant he/she can place the order via the order placing system and once the ordered items are ready, the items will be delivered to the customer. Our solution will help the customer to select the food items more efficiently with help of the reviews from previous customers and also if an item is unavailable that item will not be shown in the menu. The WaiterBot system will help the restaurant by providing an efficient delivery mechanism. WaiterBots will not mess up orders and they will deliver the food items to the correct table. Also this may be a new experience for the customers and the WaiterBot system will attract more customers to the restaurant.

2. Prerequisites

- Basic understanding of microcontroller programming with arduino framework
- Web development with React framework
- Mobile development with Flutter
- Backend development with NodeJS

3. WaiterBot

3.1. Overview

When a customer selects the preferred items, the person at the control unit will have to confirm the order. After the payment process is complete the person at the control unit will be notified and a Waiterbot will be automatically assigned to that table. The order can then be sent to the kitchen for preparation. Once the order is ready, the food items have to be kept on the assigned WaiterBot and the WaiterBot can be sent for delivery. These robots are stationed near the control unit.

Once the WaiterBot is deployed, the WaiterBot will follow the correct path to the table. If there is an obstacle in the path the WaiterBot will stop the movement and notify the control unit. If the path is clear, the WaiterBot will follow its path to the correct table. When the WaiterBot reaches the correct table it will stay there until the customer collects all the food items from its tray. When all the food items are collected, the WaiterBot will notify the control unit and it will return to its station.

3.2. Hardware Components Required

Electronic Components

● ESP-WROOM-32 module	-	1
● HC-SR04 module	-	2
● IR sensor module	-	7
● 5kg Load cell	-	1
● HX711 module	-	1
● 17HS4401 Stepper motor	-	4
● DVR8825 stepper motor controller	-	4
● 8 channel logic level converters	-	2
● LCD display 16x02	-	1
● LCD I2C module	-	1
● 18650 Lithium-ion battery	-	6
● LM2596 buck converter module	-	1
● 3s 20A Battery protection module	-	1
● 12.6V DC charger	-	1
● Resistors		
○ 33k	-	1
○ 10k	-	1

Other Components

● 6.5mm wheels	-	4
● Hexagonal coupling	-	4

3.3. ESP-WROOM-32 module

ESP32 is used as the micro controller of the waiterbot. It has the following specifications.

ESP-WROOM-32 Chip

- Xtensa® Dual-Core 32-bit LX6
- Upto 240MHz Clock Freq.
- 520kB internal SRAM
- 802.11b/g/n Wi-Fi transceiver
- Bluetooth 4.2/BLE

Power Requirement

- Operating Voltage: 2.2V to 3.6V
- On-board 3.3V 600mA regulator

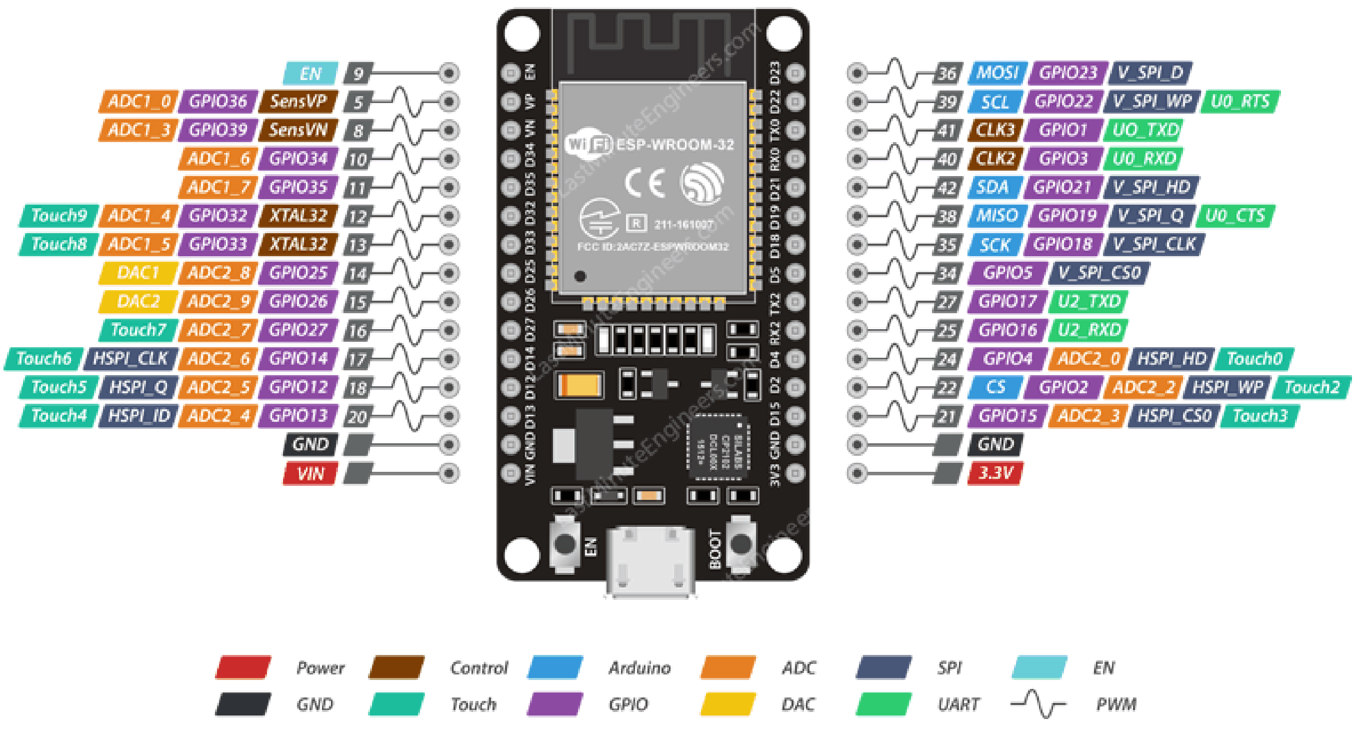
Multiplexed I/Os

- 25 GPIOs
- SPI, I2C & I2S interface
- 15 ADC channels

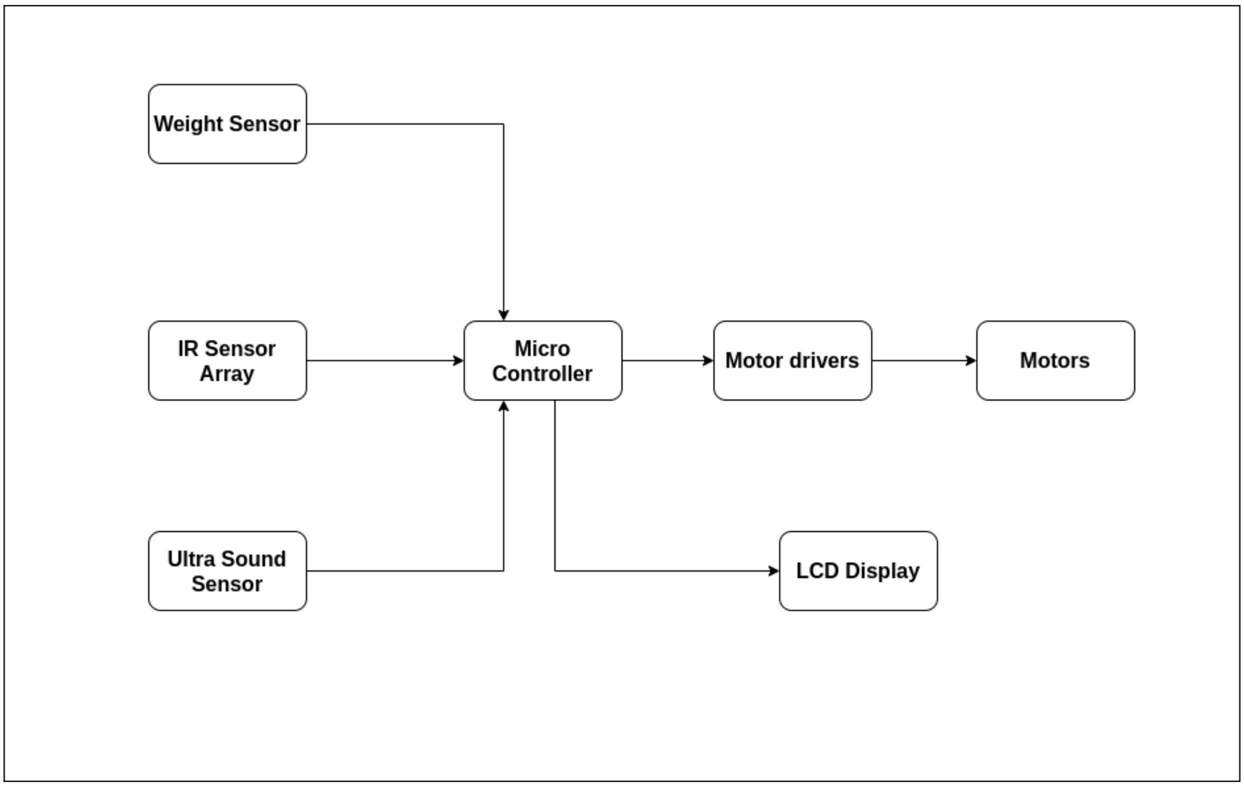
Warning!

- GPIOs are not 5V tolerant. Use a level shifter in order to interface with the 5V components.
- When powering the ESP32 module with 5V, make sure to power it using the VIN, to power the ESP32 Chip via the on board 3.3V regulator.

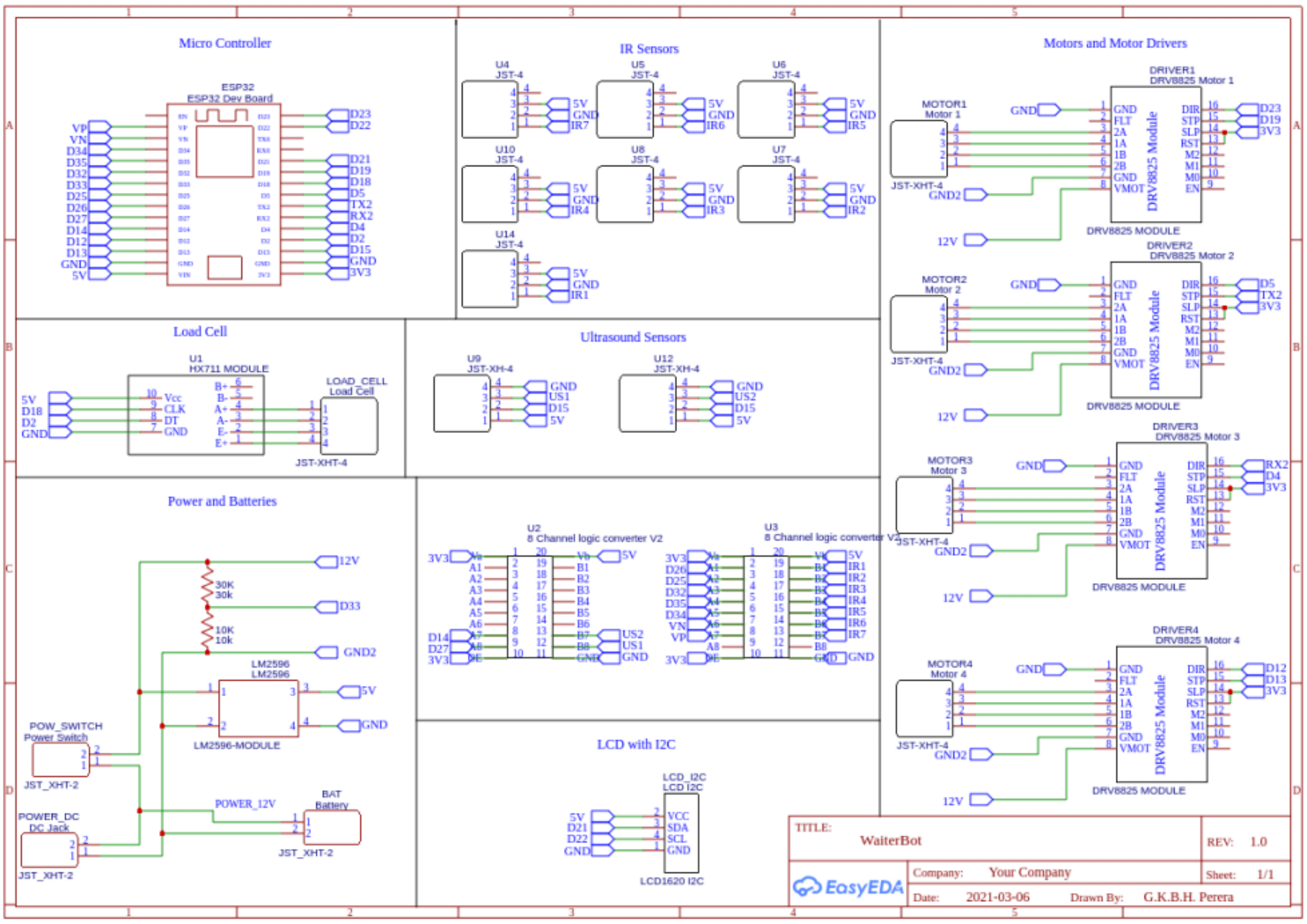
3.4. ESP32 Pinout Diagram


EN 9
ADC1_0 GPIO36 SensVP 5
ADC1_3 GPIO39 SensVN 8
ADC1_6 GPIO34 10
ADC1_7 GPIO35 11
Touch9 ADC1_4 GPIO32 XTAL32 12
Touch8 ADC1_5 GPIO33 XTAL32 13
DAC1 ADC2_8 GPIO25 14
DAC2 ADC2_9 GPIO26 15
Touch7 ADC2_7 GPIO27 16
Touch6 HSPI_CLK ADC2_6 GPIO14 17
Touch5 HSPI_Q ADC2_5 GPIO12 18
Touch4 HSPI_ID ADC2_4 GPIO13 20
GND
VIN
36 MOSI GPIO23 V_SPI_D
39 SCL GPIO22 V_SPI_WP U0_RTS
41 CLK3 GPIO1 U0_TXD
40 CLK2 GPIO3 U0_RXD
42 SDA GPIO21 V_SPI_HD
38 MISO GPIO19 V_SPI_Q U0_CTS
35 SCK GPIO18 V_SPI_CLK
34 GPIO5 V_SPI_CS0
27 GPIO17 U2_TXD
25 GPIO16 U2_RXD
24 GPIO4 ADC2_0 HSPI_HD Touch0
22 CS GPIO2 ADC2_2 HSPI_WP Touch2
21 GPIO15 ADC2_3 HSPI_CS0 Touch3
GND
3.3V
EN VP VN D34 D35 D32 D33 D25 D26 D27 D14 D12 D13 GND VIN
D23 D22 TX0 RX0 D21 D19 D18 D5 TX2 RX2 D4 D2 D15 GND 3V3
ESP-WROOM-32
EN
BOOT
Power
GND
Control
Touch
Arduino
GPIO
ADC
DAC
SPI
UART
EN
PWM

3.5. Block Diagram

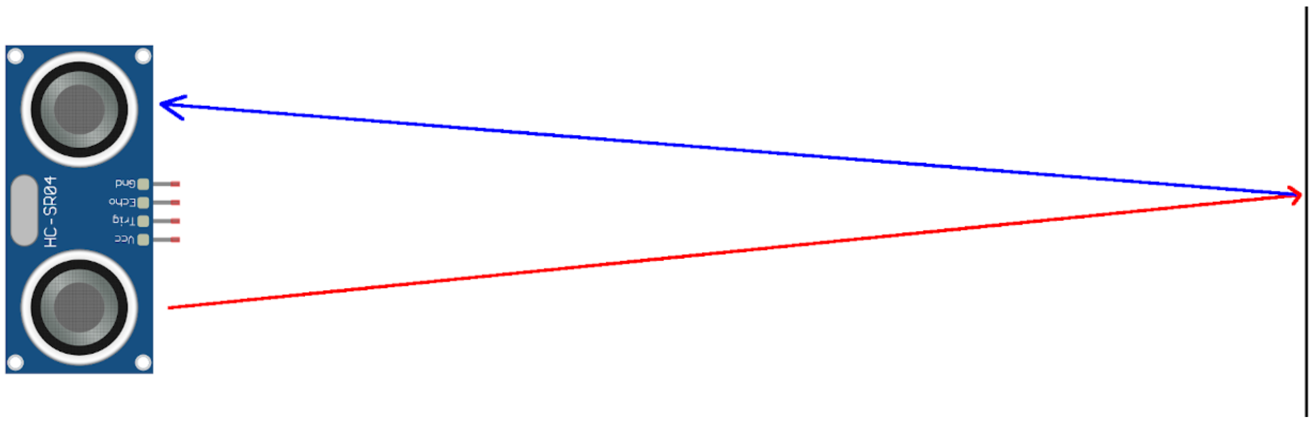


3.6. Circuit Diagram



3.7. Obstacle Avoidance Mechanism

HC-SR04 modules are used for detecting the obstacles in the path. The HC-SR04 module uses an ultrasonic sensor and receiver to measure the distance to an object. This is done by generating an ultrasonic sound wave that bounces off an object and returns to the sender. The sensor counts the time between the signal sent and received to determine how far the object is.



- Specifications
  - Operating voltage - 5V
  - Range - 3cm to 4m
  - Detection angle - 15 degrees

- Interfacing with ESP32 module

As shown in the circuit diagram, HC-SR04 module's TRIG and ECHO should be connected with the ESP32 through the logic level converter module, because for proper operation of the HC-SR04 module, it should be powered with 5V and ESP32 module's GPIOs cannot tolerate 5V, so logic level conversion was required.VCC and GND should be connected to 5V and GND respectively.

- Operation with the ESP32 module

The way this sensor is used is by setting a start pulse on the TRIG pin. The sensor will start the measurement and return the result to the ECHO pin.

The start signal on the TRIG pin needs to be,

- at least 2μs LOW (0V) before it can be high again
- 10μs HIGH (+5V) to be considered a start pulse

Once a valid start pulse is detected on the trigger pin, the sensor will send 8 pulses of 40kH, and count how long it takes for the signal to return. Finally, it will send 1 pulse on the ECHO pin of which the HIGH time (+5V) represents the time (in microseconds) it took for the signal to be sent, bounced back on an object and be received by the sensor.

- Calculating The Distance and Detecting the Obstacle

Microcontroller can measure the time that the signal from the ECHO pin was high and compute the distance between sensors and obstacles using the following equation.

$$Distance = (Speed\ of\ Sound\ *\ ECHO\ signal\ high\ time)/(1000000\ *\ 100\ *\ 2)$$

Here,

Speed of sound = 343 m/s
Distance in meters
ECHO signal in high time in microseconds

If the calculated distance is less than the defined value then the robot will stop its movement.

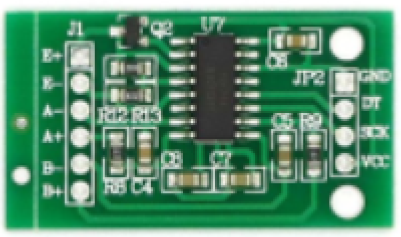
- Optimizing the performance in the code

Efficient polling can be used. The distance measurement can only be done in periodic intervals rather than measuring in each and every iteration in the loop.

When measuring the high time of the ECHO pulse interrupt mechanism can be used. Timer can be started when a low to high change on ECHO is detected and the timer can be stopped when the high to low change on ECHO is detected. This mechanism will not block the main loop when measuring the time in which the ECHO pulse was high.

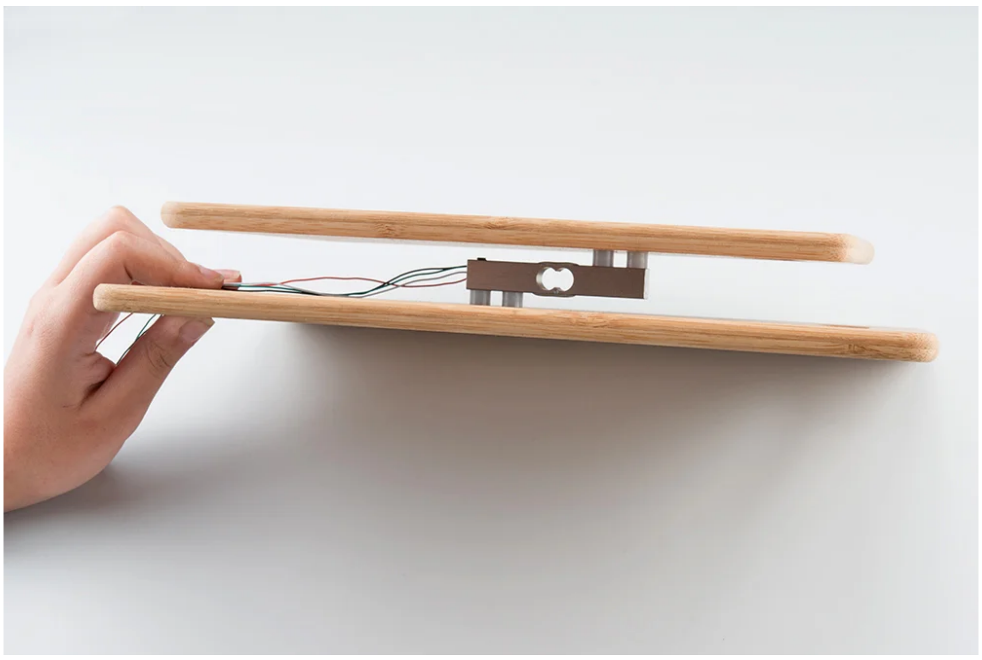
3.8. Food Item Detection

For food item detection the load cell and the HX711 module is used. Readings from the load cell are of millivolts and the HX711 module will amplify and convert the analog readings to digital and send them to the microcontroller.



- Mounting the load cell

Load cells should be mounted on two rigid plates and there should be spaces between the load cell and the rigid plates as shown in the figure below and the arrow marker on the load cell should point downwards.



- Interfacing with ESP32 module

As shown in circuit diagram, the load cell and HX711 should be connected as,

Load Cell	**HX711**
Red wire	E+
Black wire	E-
White Wire	A+
Green Wire	A-

The VCC and GND of the HX711 module should be connected to 5V and GND respectively. DT and SCK should be connected to D2 and D18 pins of the ESP32 module respectively.

- Measuring the weight

Measuring the weight can be done by using the ‘hx711.h’ library.

Before measuring the weight using the load cell and HX711 module, the calibration factor has to be determined by placing known weights.
To determine the calibration factor the calibration code can be uploaded and by placing the known weight the calibration factor can be adjusted until the measured weight is the same as weight of the object.

Calibration code
https://github.com/buddhiheshan/waiterbot-hardware/tree/4d137bc18638d81f204400325f3558340c5615a9/src

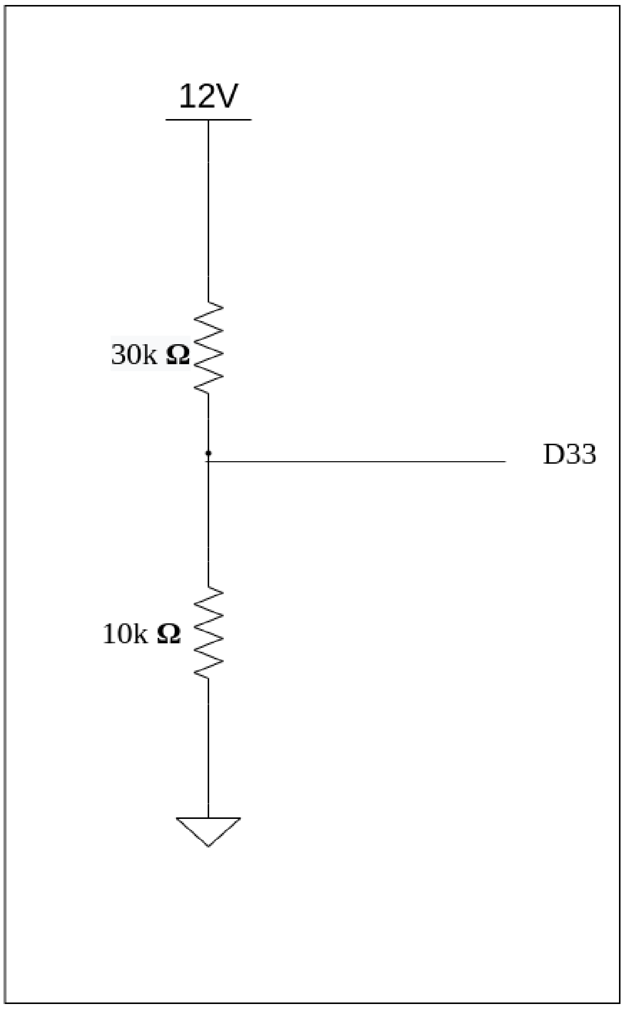
Once the calibration factor is determined, the weight can be measured.

- Detecting the food items on the tray

When the WaiterBot is deployed for delivery, first the weight of the food item is measured. If the weight is zero or close to zero then WaiterBot will inform the control unit that there is no food item on the tray. After placing the food item on the tray, the robot will deliver it to the correct table. If the food item is taken away from the tray before reaching the correct table, the robot will stop and will inform the control unit. After reaching the correct table the WaiterBot will stay there until the food items are taken away from the tray. This is also done by measuring the weight.

3.9. Battery Percentage Detection

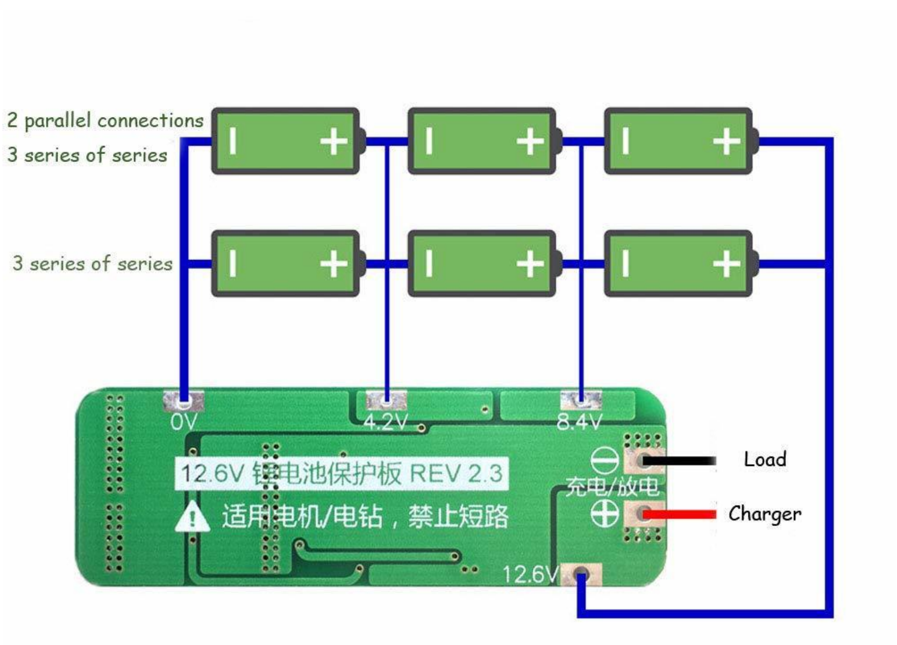
Battery percentage can be determined by reading the voltage value between the two resistors. It is connected to the D33 pin of the ESP32 and the analog value is converted to a digital value by using the ADC in the ESP32. This will result in a value between 0 and 4095. Then the value is mapped such that it is between 0 and 100.



3.10. Power

WaiterBot is powered by 6 18650 lithium-ion batteries. The 17HS4401 stepper motors require 12V and other components(microcontroller, sensors and LCD) require 5V. To achieve this requirement a LM2596 step down converter is used to step down from 12V to 5V.

3s 20A battery protection module is used to prevent the over discharging and overcharging of the batteries. Batteries are connected to the 3s 20A module as shown below,



3.11. Line Following Mechanism

An array of 7 IR sensors are used to detect the line. Depending on the IR sensor input an error will be computed and the speeds of the stepper motors will be controlled according to the error.

- IR Sensor Specifications

Operating voltage	-	5V

- Operation of the IR Sensors

IR sensors have an IR emitter and an IR receiver. Once the IR emitter emits the IR beam, it will incident on the surface and reflect back. Depending on the intensity of the reflected beam the receiver will create a digital signal. Using the potentiometer on the IR sensor the threshold value can be adjusted.

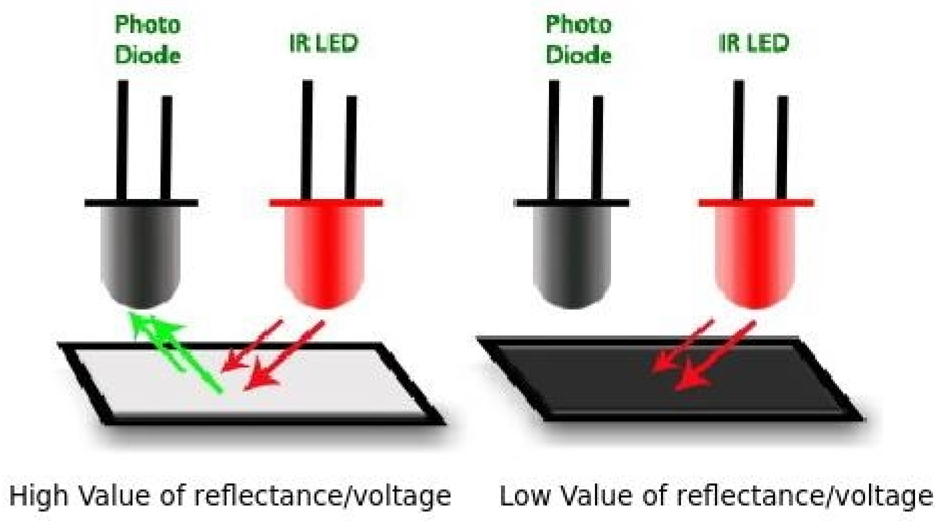


Photo Diode
IR LED
Photo Diode
IR LED
High Value of reflectance/voltage
Low Value of reflectance/voltage

As shown in the figure the black surface will absorb the IR beam and the white surface will reflect the IR beam. Using this we can detect the black line.

- Interfacing with the ESP32

As shown in the circuit diagram, IR sensors can be interfaced with the microcontroller. The VCC of the IR sensor should be connected to 5V and GND should be connected to the GND. The OUT pin should be connected to the microcontroller through a logic level converter since the IR sensors work with 5V and ESP32 works with 3.3V.

- IR sensor layout and the Error Detection

IR sensors are positioned as shown in the figure, 6 sensors are in line and one the remaining is in front. One sensor is placed at front to detect the end points and to compute errors when turning at bends.

Depending on the IR sensor readings, the error is calculated as shown in the following figure. When the line is detected at the middle, the error is considered as 0 and when the line moves towards the ends, the errors gradually increase and decrease. When all the sensors detect a black line that means it is a junction or an endpoint.

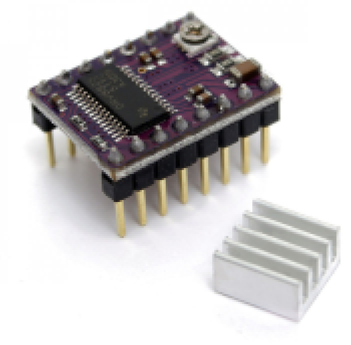
○ ○○○○○●	Error = 4
○ ○○○○●●	Error = 3
○ ○○○●●○	Error = 2
● ○○○●●○	Error = 1
● ○○●●○○	Error = 0
● ○●●○○○	Error = -1
○ ○●●○○○	Error = -2
○ ●●○○○○	Error = -3
○ ●○○○○○	Error = -4
● ●●●●●●	Junction/End

3.12. Stepper Motor Control

Stepper motors cannot be directly driven by the microcontroller. Micro controller will provide necessary logic for motor controlling and the DRV8825 stepper motor controller will drive the stepper motor depending on the logic from the microcontroller.

- 17HS4401 Stepper motor specifications
    - Operating voltage - 12V
    - Rated current - 1.7A
    - Step angle - 1.2 degrees
    - Steps per revolution - 200

- Operating the Stepper Motors with the Microcontroller and Motor Controller



Stepper Motors are controlled by sending pulses. Stepper motor will move one step for one pulse. And the frequency of the pulses will determine the speed of rotation of the stepper motor.

When the robot is out of the path (error will be -4, -3, -2, -1, 1, 2, 3, 4) the speed of the motors will be computed by using the error values. This can be used to correct the path and this method is known as proportionality control.

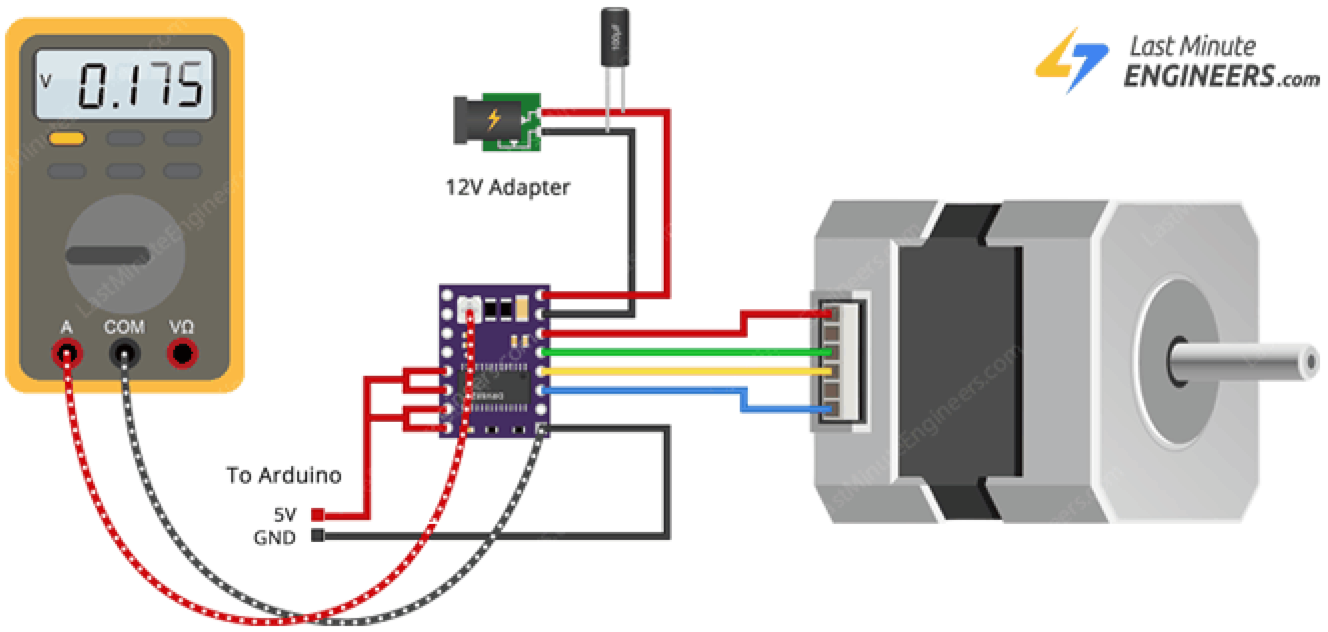
Warning!

Before connecting the stepper motors to the DRV8825 motor controller the current limit of the motor controller should be adjusted to the rated current to prevent damaging the stepper motor. The current limit can be adjusted by using the potentiometer on the DRV8825 motor controller module. The current limit can be commuted as follows,

$Current\ Limit\ =\ Vref\ *\ 2$

Where Vref is the voltage across the ref pin and the GND pin.

Following figure shows the measurement of Vref.

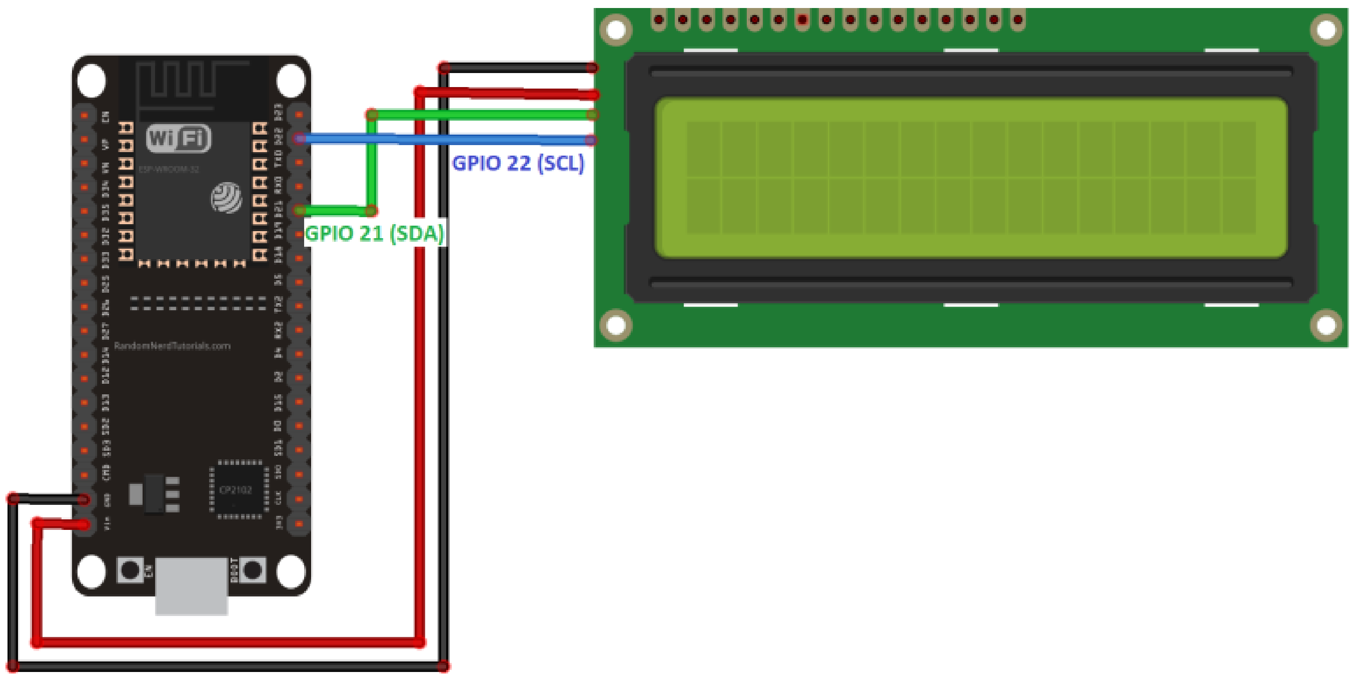


3.13. LCD Display

The 16x02 LCD display and the I2S port expander for the LCD are used to display the information to the user. I2C port expander uses I2C protocol to communicate between the microcontroller and the LCD I2C expander module. This method was used to reduce the number of GPIO interfaces used by the LCD.

- Library used - LiquidCrystal_I2C

Following diagram shows how the LCD with I2C expander can be interfaced with the ESP32 micro controller.



The VCC and the GND pins of the I2C port should be connected with the 5V and the GND respectively and the SCL and SDA should be connected with the D22 and D21 respectively.

3.14. Communication Protocols

WaiterBots will use Wifi to connect to the local network and will communicate using MQTT over TCP. Asynchronous implementations of Wifi and MQTT are used to connect the robots, so that if Wifi or MQTT disconnects the delivery process will not get interrupted.

- Libraries used - async-mqtt-client, 'WiFi.h'

- WaiterBots will subscribe the following topic,
  waiterbot/{shop_id}/{robot_id}

- WaiterBot will publish to the following topic,
  operator/{shop_id}/{robot_id}

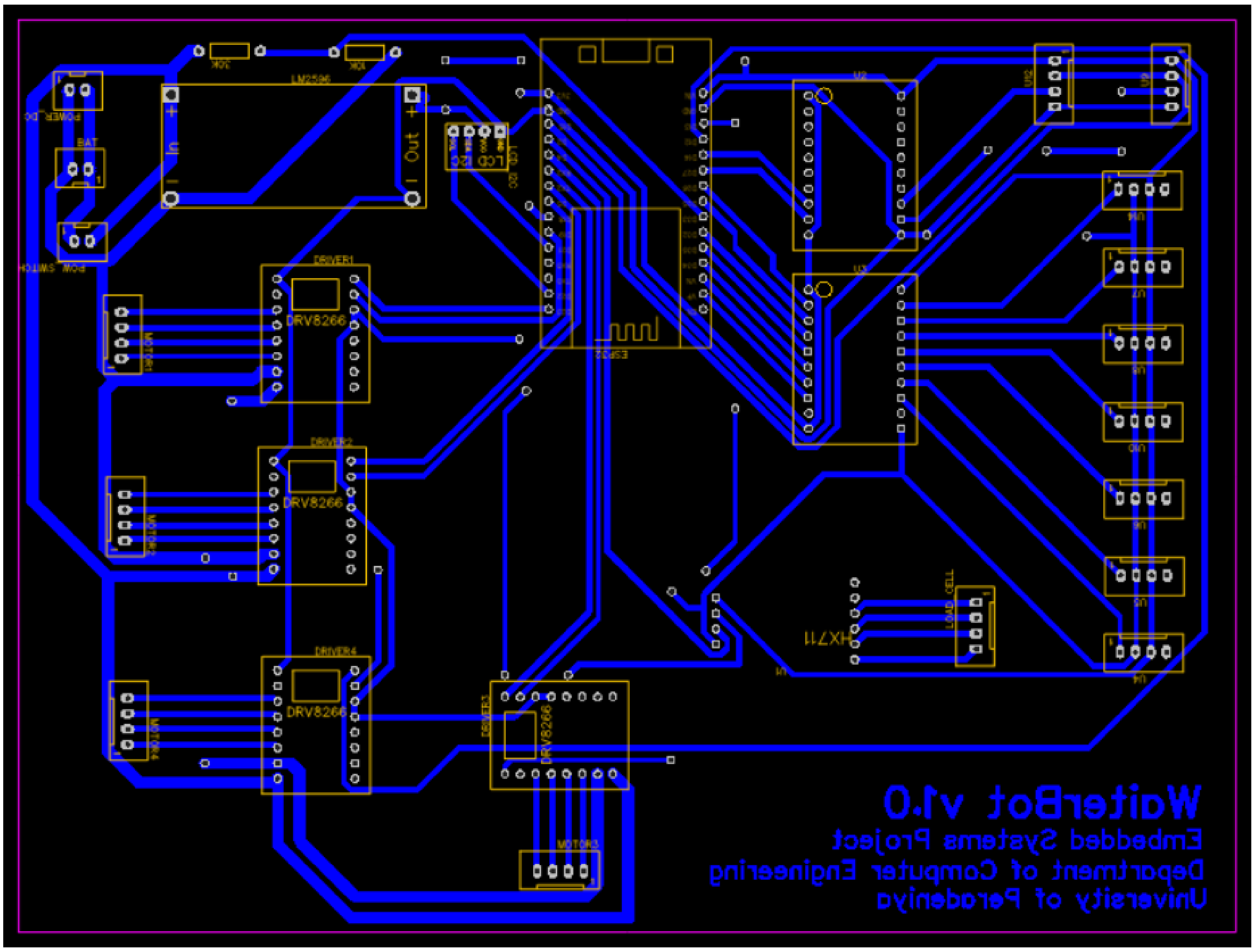
- The following MQTT messages are sent to the operator,

MQTT message	Meaning
ready	WaiterBot is ready for delivery and is at the station
empty	No food item on the tray, but robot is sent for delivery
delivering	Robot has started delivering
obstacle	Robot has stopped due to obstacle
stolen	Food item has been taken away from the tray before reaching the delivering table
delivered	Food item has been delivered

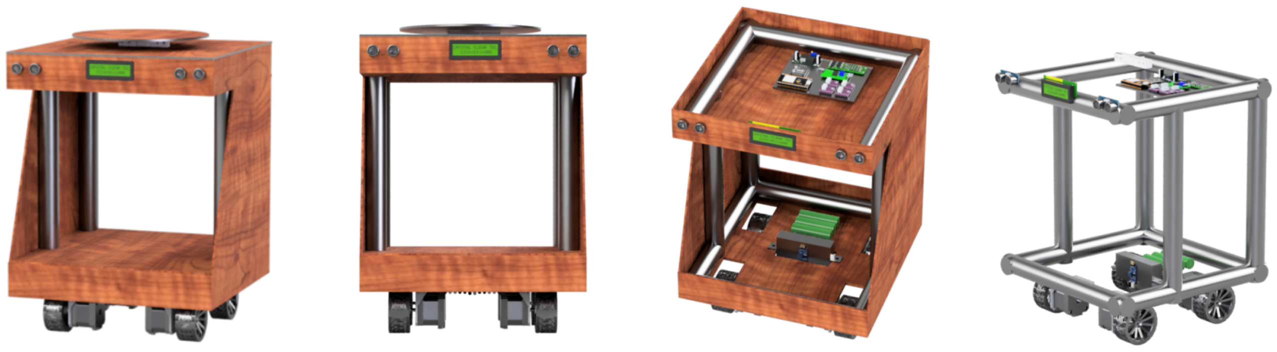
- The following MQTT messages are received by the waiterbot,

MQTT message	Meaning
‘deliver {destination_junction_count} {total_junction_count} {turn_direction}’	Deliver to the specific table

3.15. PCB Design



3.16. 3D Models



3.17. Complete Code

https://github.com/buddhiheshan/waiterbot-hardware/tree/4d137bc18638d81f204400325f3558340c5615a9/src

4. Waiter Bot API

4.1. Introduction

Since we have to provide the service to multiple types of devices, (desktop, mobile, IoT) we decide to have an API to facilitate the core service.

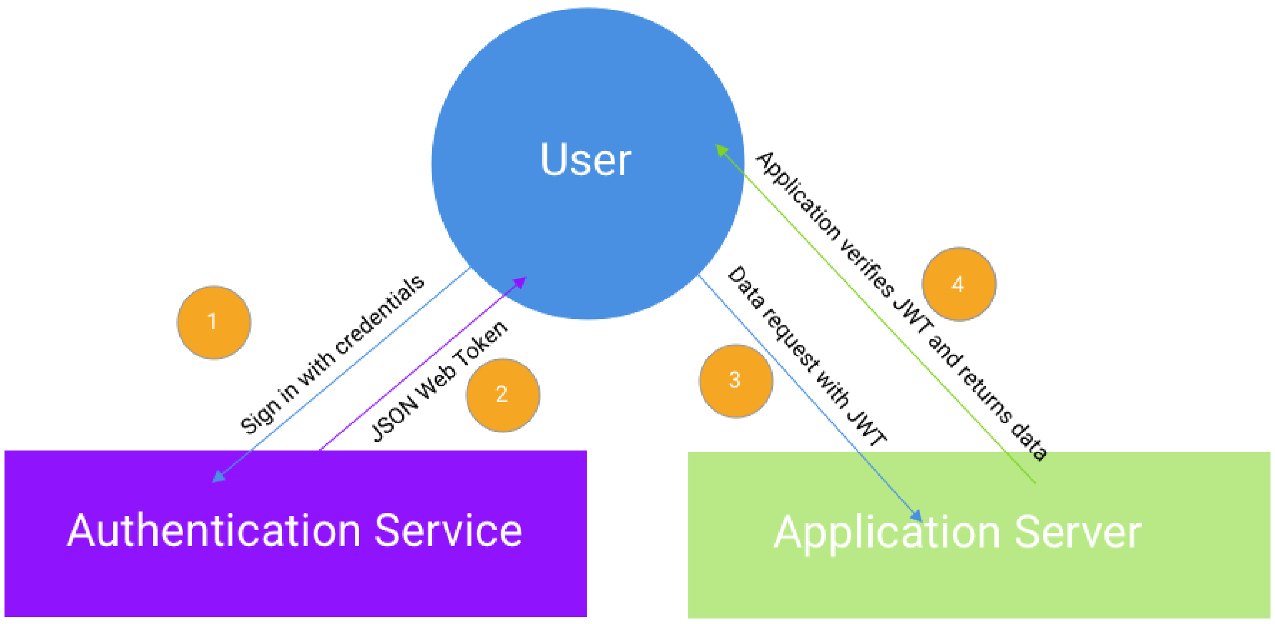
4.2. Used Technologies

- Nodejs with express.js framework
- Mongodb

To store the data on the backend we've used mongodb. So we have the flexibility to scale our application when it grows.
Since mongodb is schema less we have to have an application level schema to facilitate our service. In order to do that we used Mongoose ODM

4.3. Authentication and Authorization

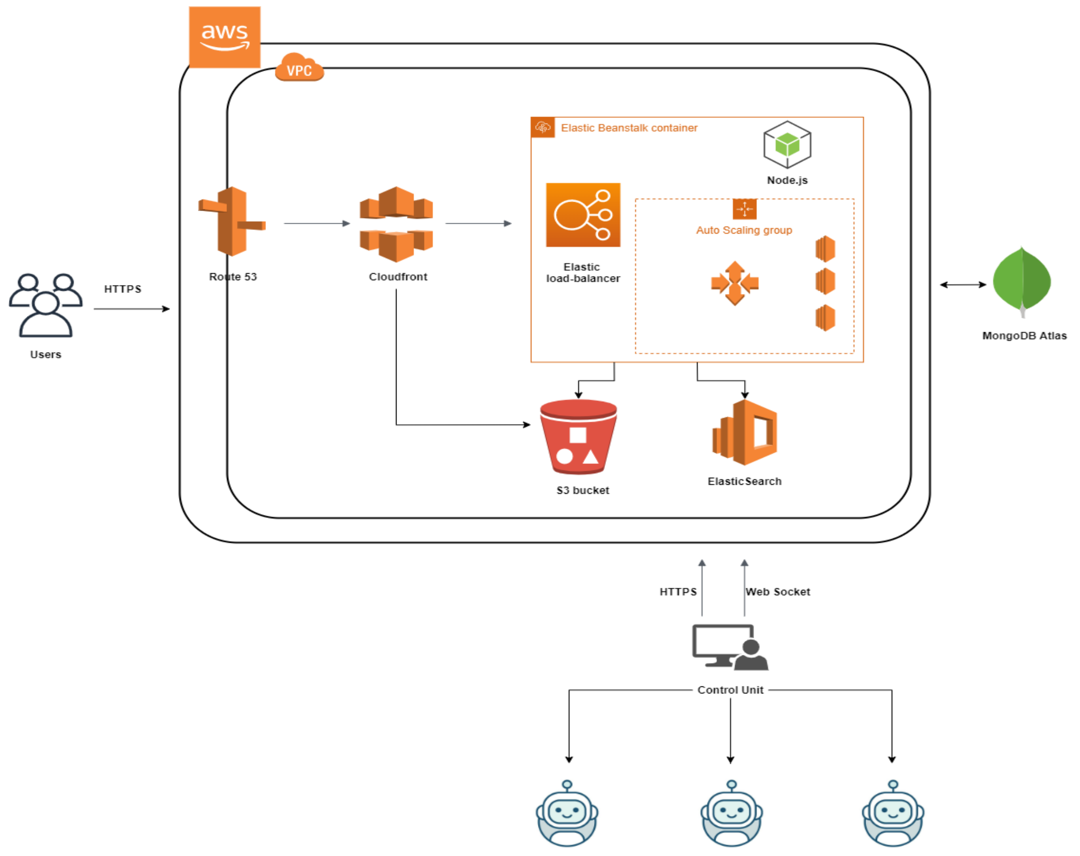
In the application, we use token based authentication and authorization. To generate the auth tokens and validation we use JWT (json-web-tokens)



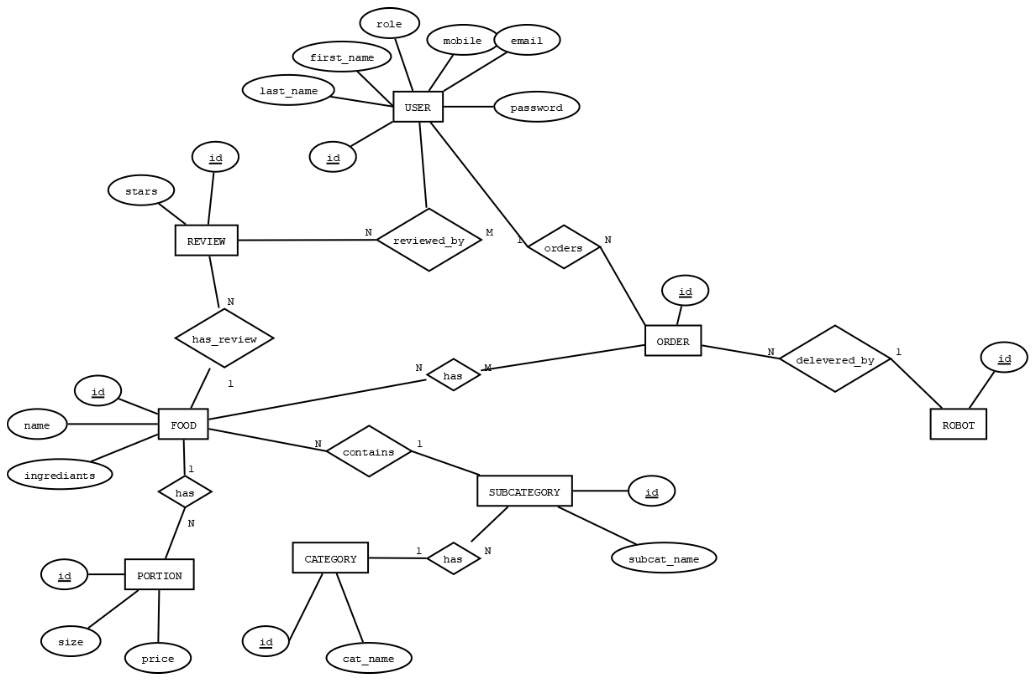
4.4. Real Time Communication

To enable the real time communication though the application, we use socket.io

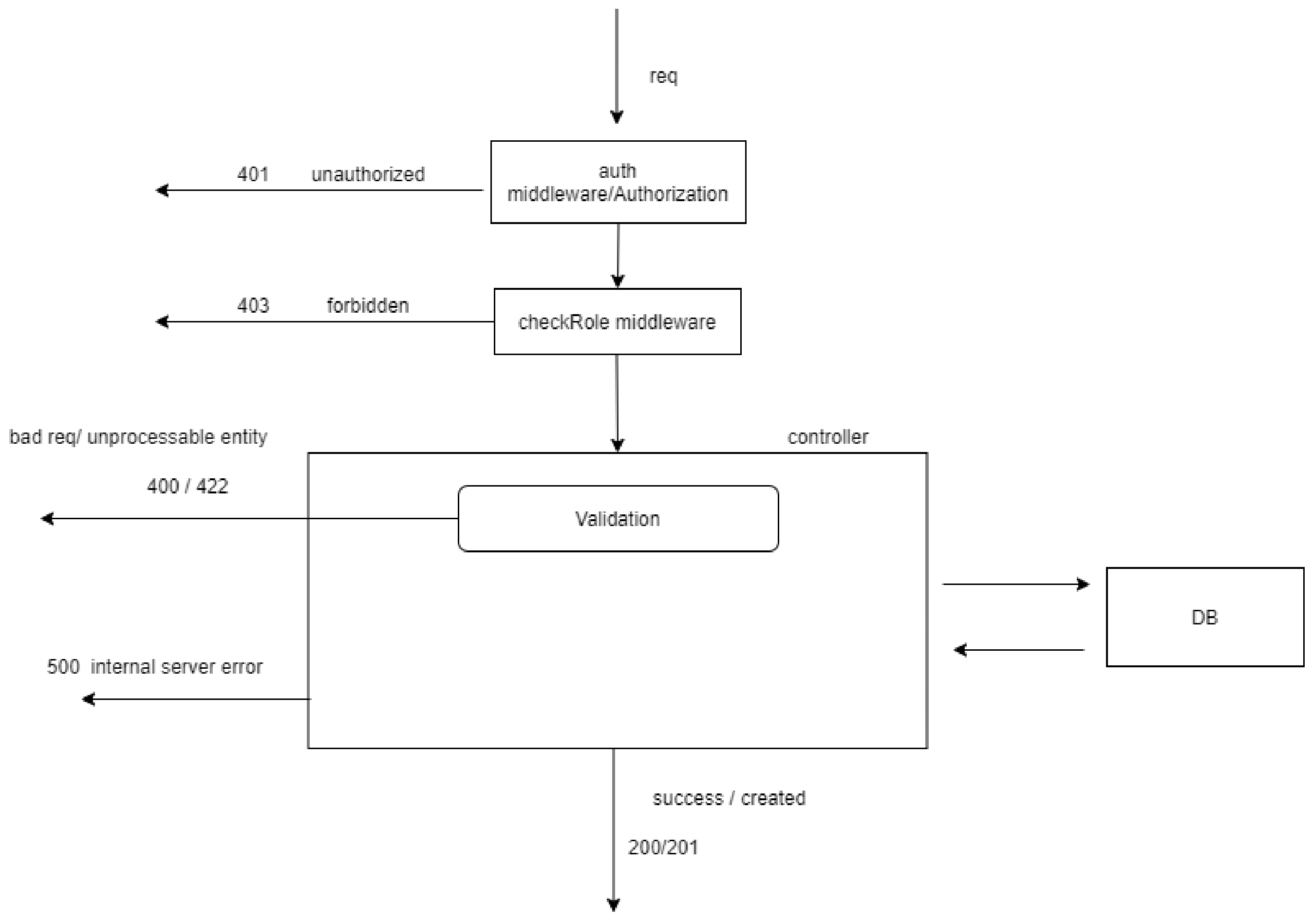
4.5. Main Solution Architecture



4.6. ER Diagram



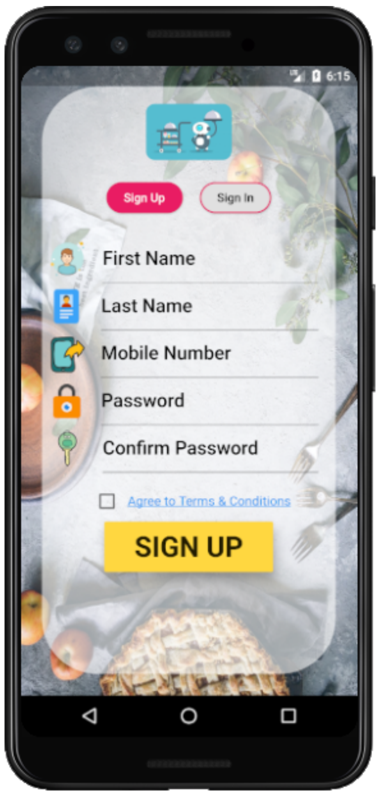
4.7. Routing of a request through the application

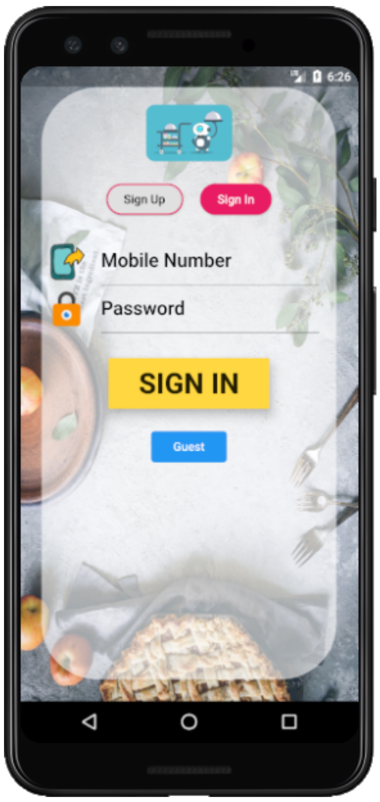


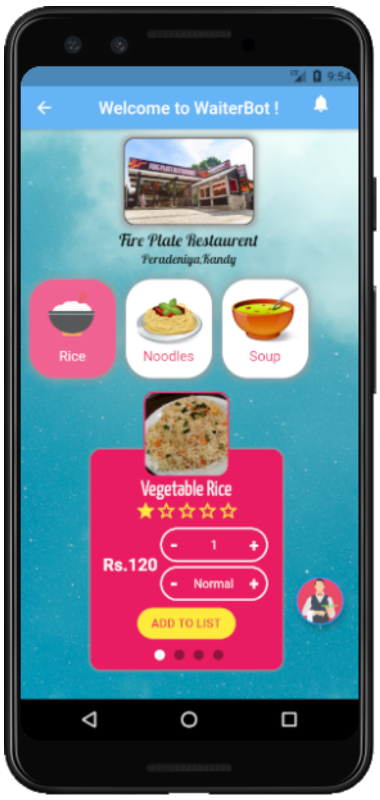
5. Mobile Application

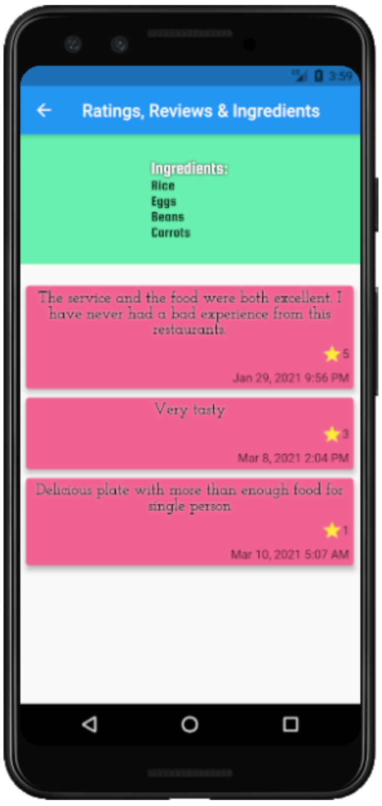
This mobile app is basically for the end customer of the restaurant. He can view the food menu, place orders and pay using credit/debit cards from this mobile app.

We have added some optional facility that is the **Guest** account option, where you can place the orders without actually signing in with your personal account.





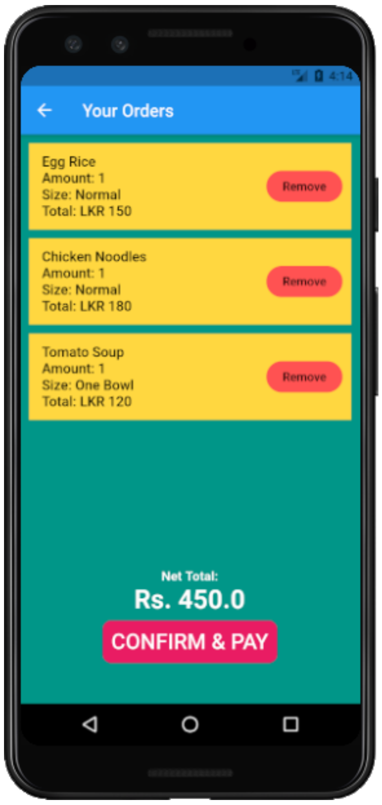




The service and the food were both excellent. I have never had a bad experience from this restaurants.
5
Jan 29, 2021 9:56 PM

Very tasty
3
Mar 8, 2021 2:04 PM

Delicious plate with more than enough food for single person
1
Mar 10, 2021 5:07 AM

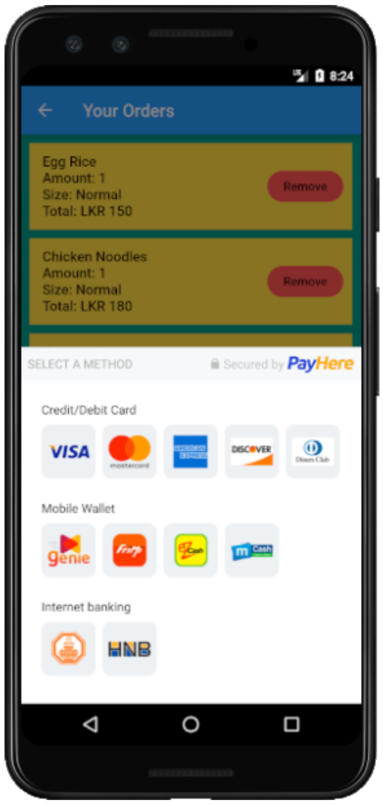


Egg Rice
Amount: 1
Size: Normal
Total: LKR 150
Remove

Chicken Noodles
Amount: 1
Size: Normal
Total: LKR 180
Remove

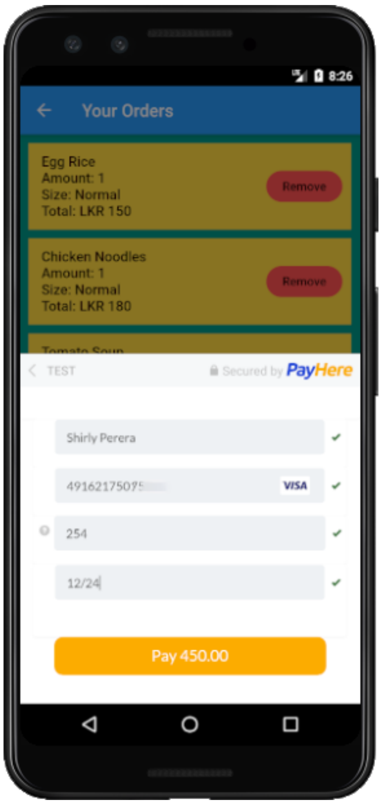
Tomato Soup
Amount: 1
Size: One Bowl
Total: LKR 120
Remove

Net Total:
Rs. 450.0
CONFIRM & PAY



Egg Rice
Amount: 1
Size: Normal
Total: LKR 150
Remove

Chicken Noodles
Amount: 1
Size: Normal
Total: LKR 180
Remove


8:26

Your Orders

Egg Rice
Amount: 1
Size: Normal
Total: LKR 150

Chicken Noodles
Amount: 1
Size: Normal
Total: LKR 180

Tomato Soup

TEST Secured by PayHere

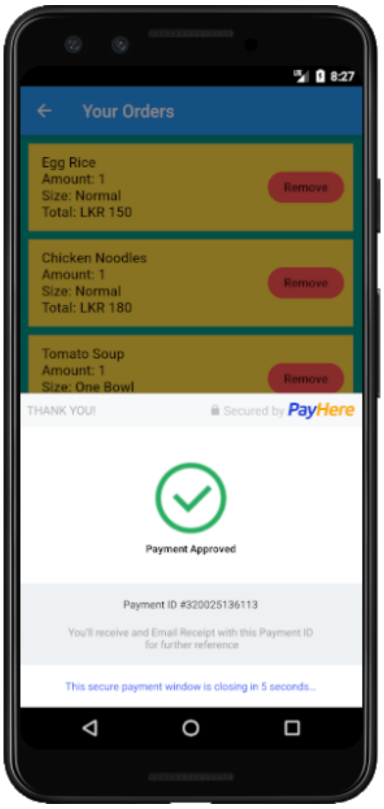
Shirly Perera

49162175075

VISA

254

12/24


8:27

Your Orders

Egg Rice
Amount: 1
Size: Normal
Total: LKR 150

Chicken Noodles
Amount: 1
Size: Normal
Total: LKR 180

Tomato Soup
Amount: 1
Size: One Bowl

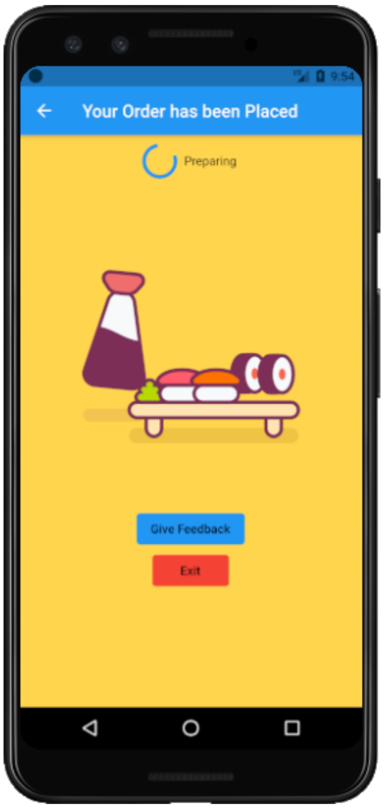
THANK YOU! Secured by PayHere

Payment Approved

Payment ID #320025136113

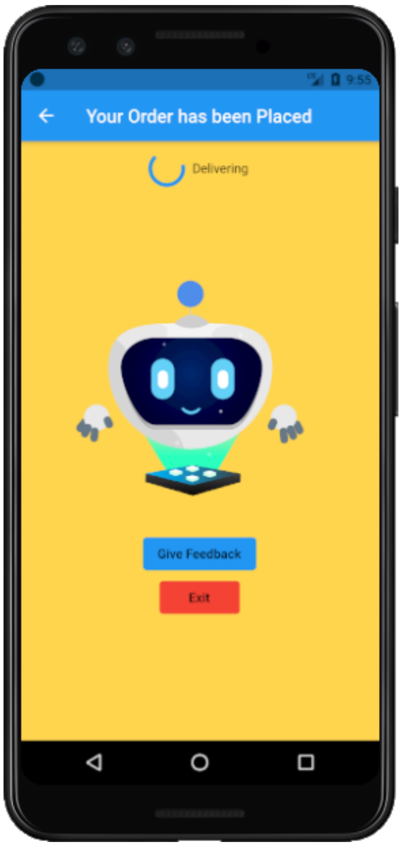
You'll receive and Email Receipt with this Payment ID for further reference

This secure payment window is closing in 5 seconds...


9:54

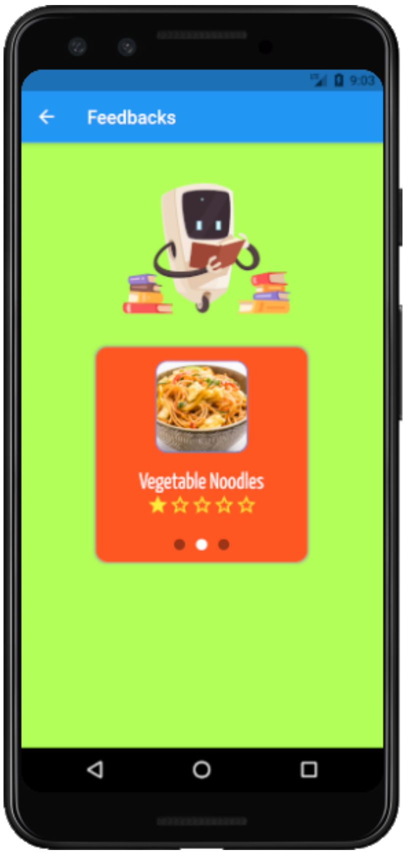
Your Order has been Placed

Preparing

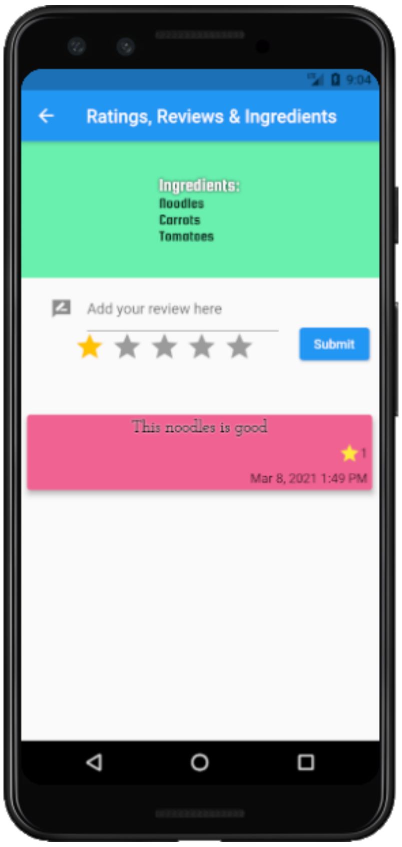

9:55

Your Order has been Placed

Delivering


9:03

Feedbacks


9:04

Ratings, Reviews & Ingredients

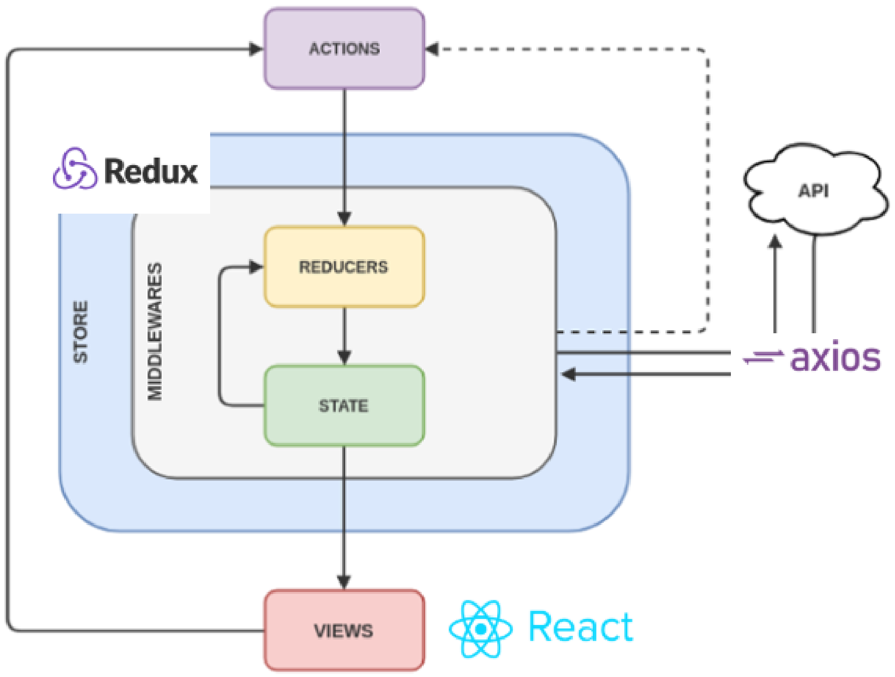
Ingredients:
Noodles
Carrots
Tomatoes

Add your review here

This noodles is good
1
Mar 8, 2021 1:49 PM

6. Web Application and Desktop Application

6.1. Design Architecture



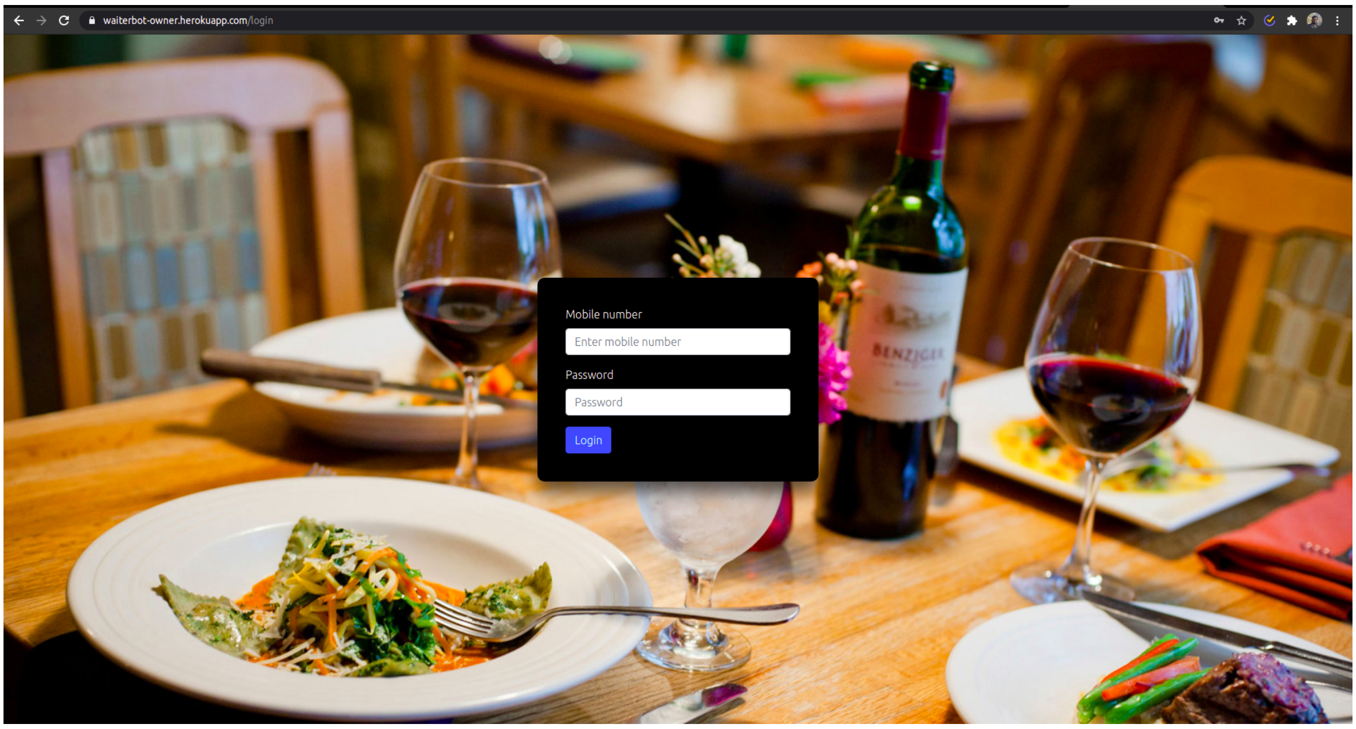
6.2. Web Application of Owner

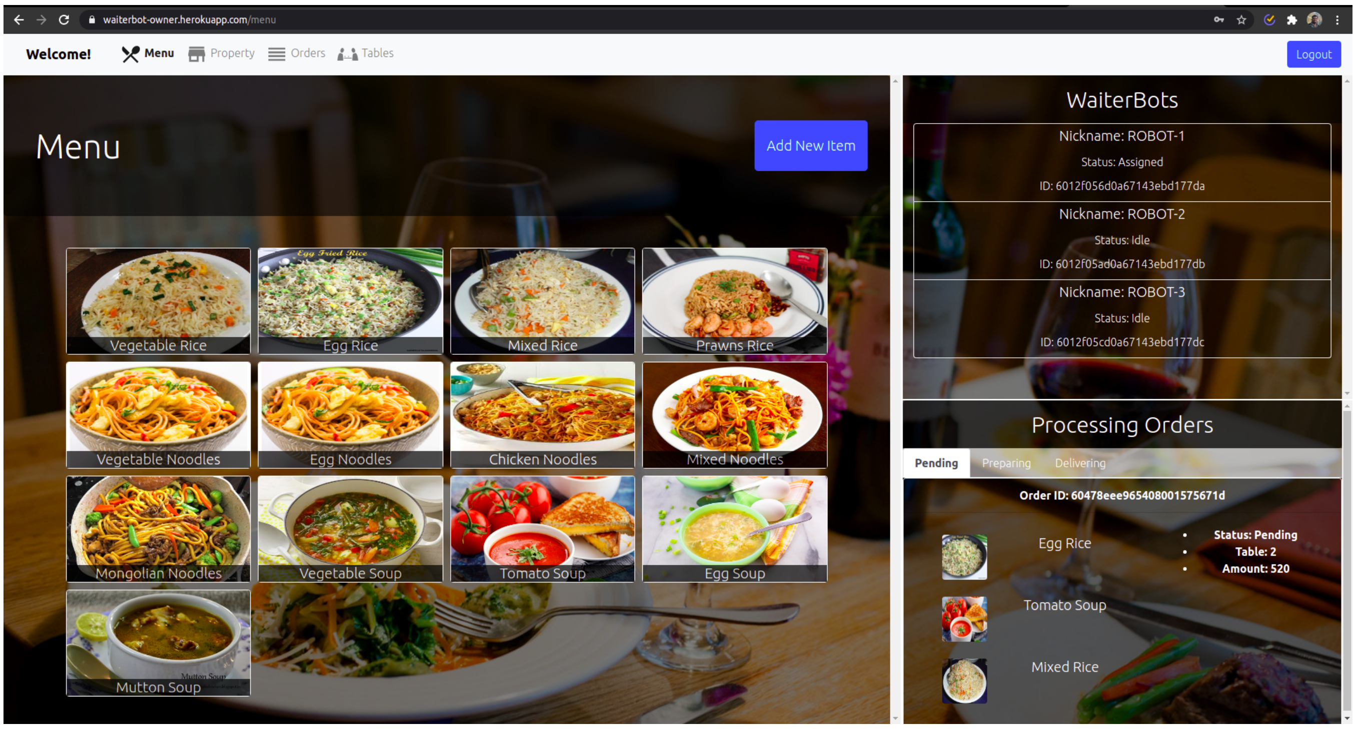
This Web Application is basically for the owner of the restaurant. The owner can login using his credentials and view and manage information about the restaurant using this web application. He can also review all the ongoing orders easily using this application as well. The owner get notified about the current status of the orders in the restaurant.

The web application is created using a responsive design and anyone can view this application using any device without any hassle.

- We used React.js for the development of the web application. React can build fast, scalable web applications due to its DOM rendering architecture. And it also has a larger community support and a vast number of libraries.
- Redux is used for state management.
- Axios is used for communication with the backend server.
- This is the UI of the restaurant owner. New orders, robot status, adding items to the menu can be done via the web application.

Screenshots of the Web Application





Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

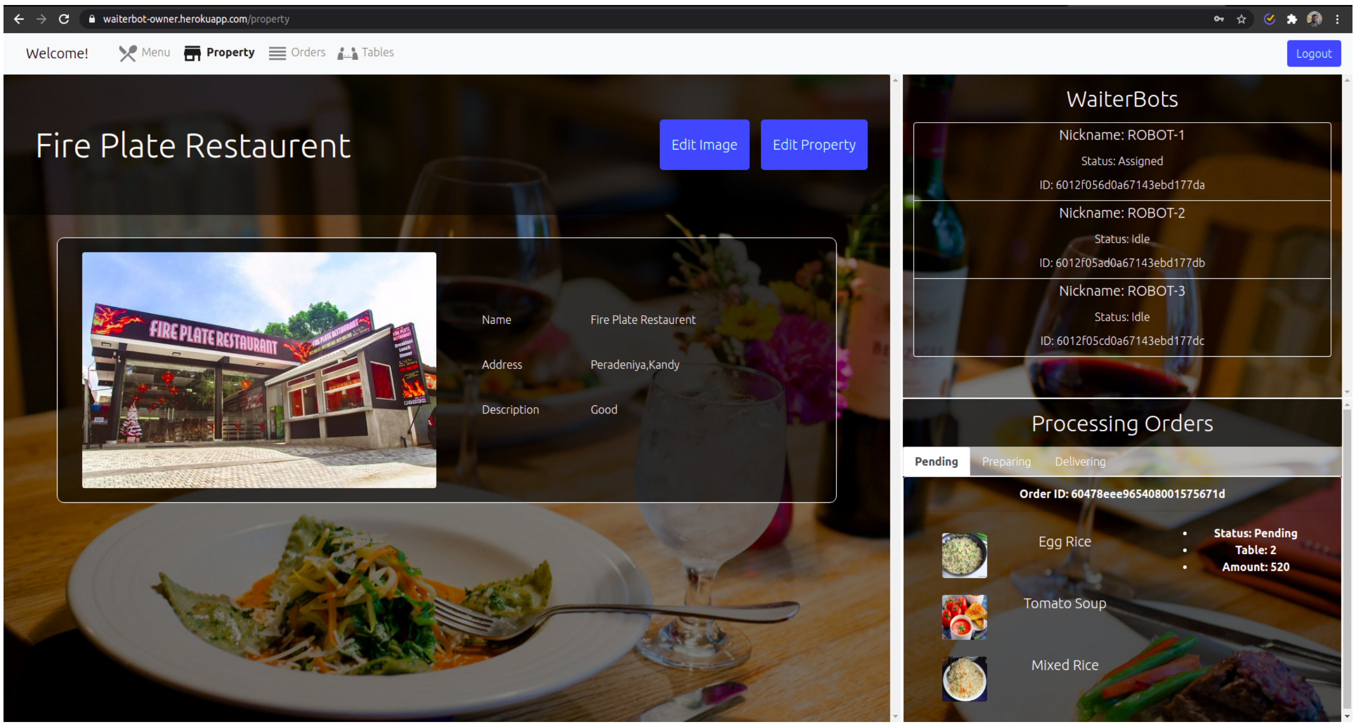
Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc

Order ID: 60478eee965408001575671d

Egg Rice
Tomato Soup
Mixed Rice

- Status: Pending
- Table: 2
- Amount: 520



Name	Fire Plate Restaurent
Address	Peradeniya,Kandy
Description	Good

Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

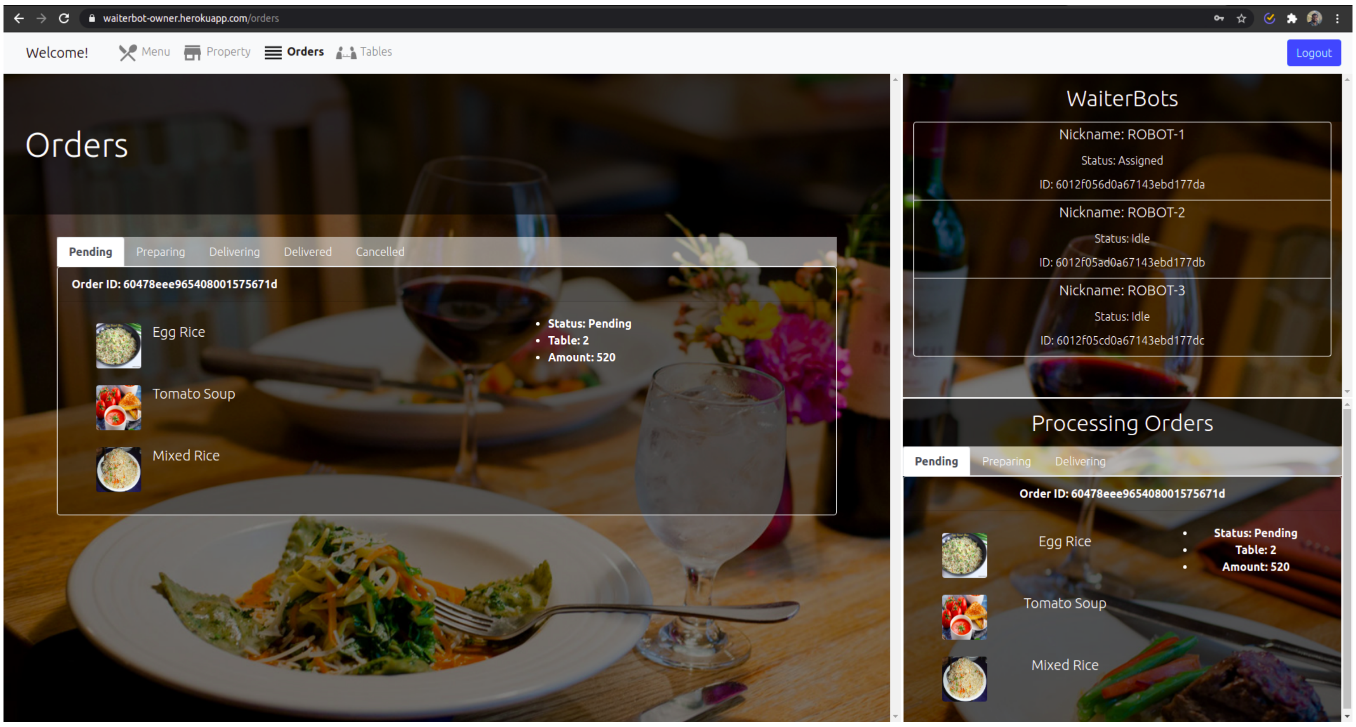
Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc

Order ID: 60478eee965408001575671d

Egg Rice
Tomato Soup
Mixed Rice

- Status: Pending
- Table: 2
- Amount: 520


waiterbot-owner.herokuapp.com/orders

Welcome! Menu Property **Orders** Tables Logout

Orders

Pending Preparing Delivering Delivered Cancelled

Order ID: 60478eee965408001575671d

Egg Rice
Tomato Soup
Mixed Rice

- **Status: Pending**
- **Table: 2**
- **Amount: 520**

WaiterBots

Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc

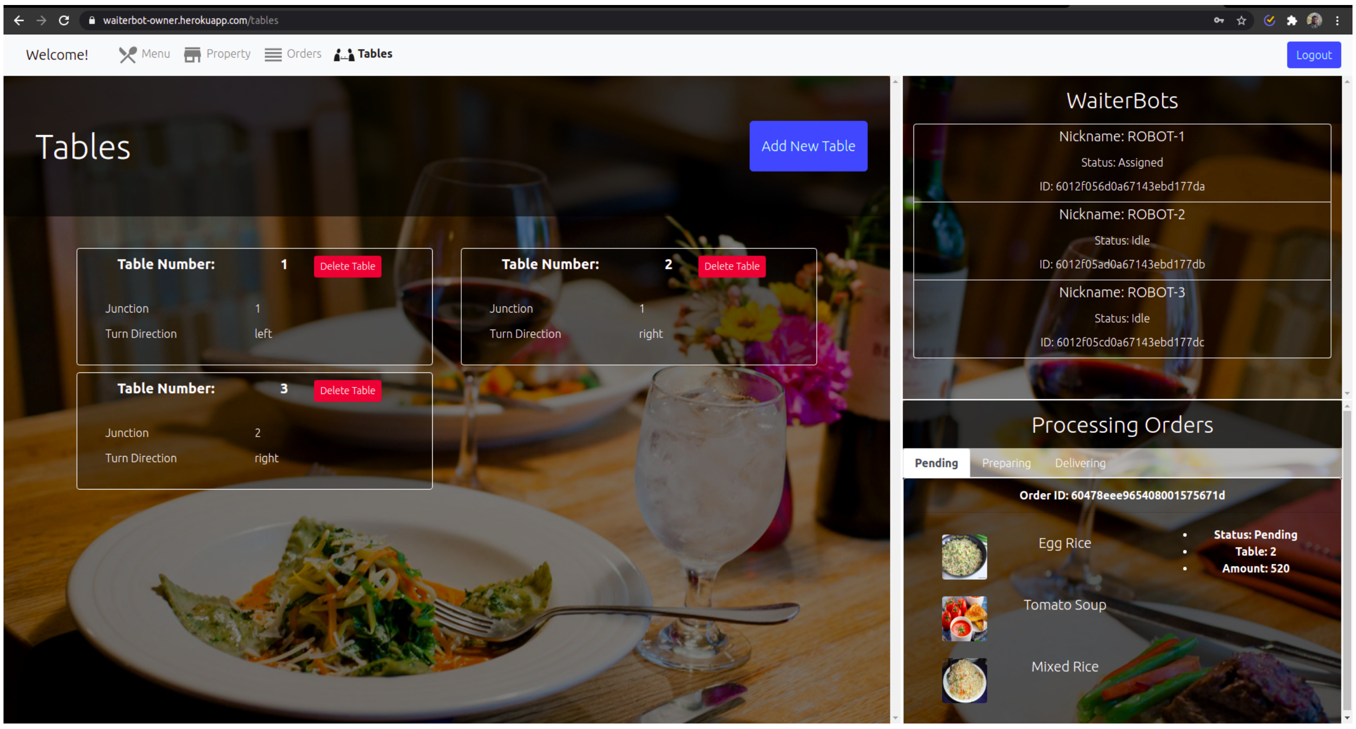
Processing Orders

Pending Preparing Delivering

Order ID: 60478eee965408001575671d

Egg Rice
Tomato Soup
Mixed Rice

- **Status: Pending**
- **Table: 2**
- **Amount: 520**


waiterbot-owner.herokuapp.com/tables

Welcome! Menu Property Orders **Tables** Logout

Tables

Add New Table

Table Number:	**1**	Delete Table
Junction	1	
Turn Direction	left	

Table Number:	**2**	Delete Table
Junction	1	
Turn Direction	right	

Table Number:	**3**	Delete Table
Junction	2	
Turn Direction	right	

WaiterBots

Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc

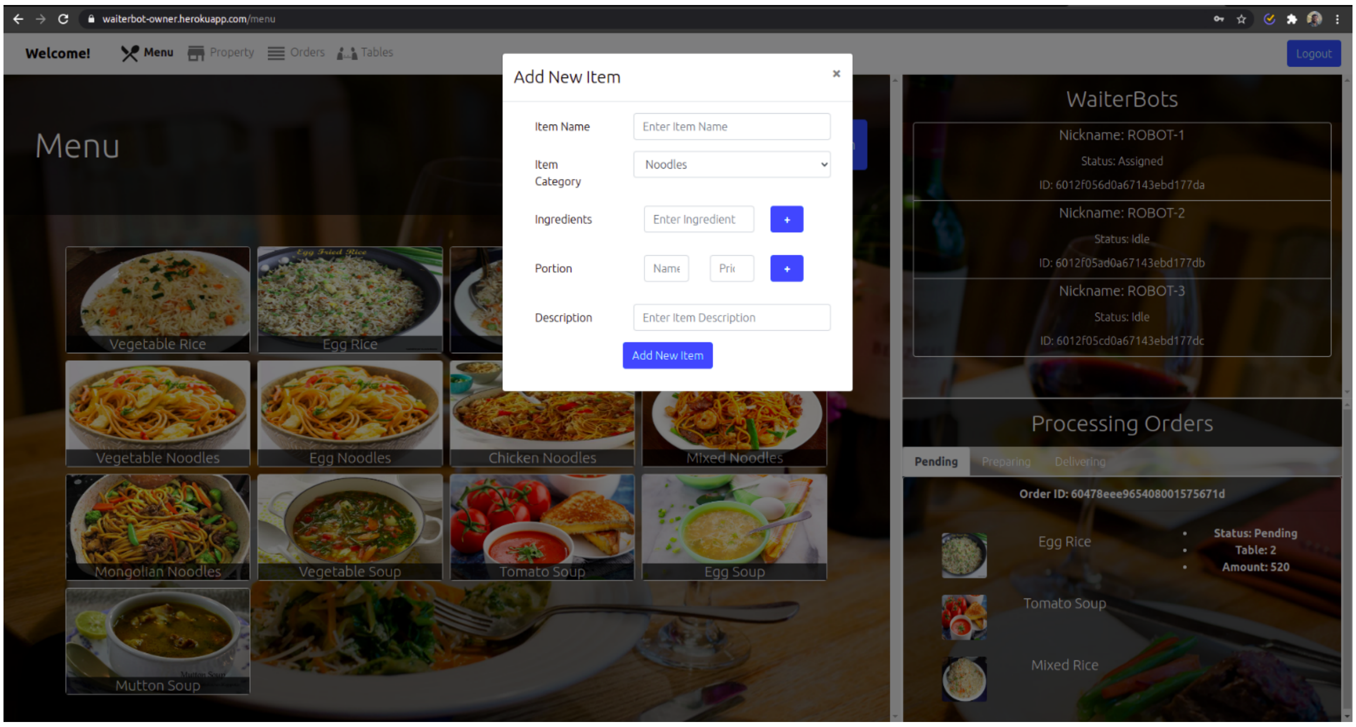
Processing Orders

Pending Preparing Delivering

Order ID: 60478eee965408001575671d

Egg Rice
Tomato Soup
Mixed Rice

- **Status: Pending**
- **Table: 2**
- **Amount: 520**



Add New Item

Item Name	Enter Item Name
Item Category	Noodles
Ingredients	Enter Ingredient
Portion	Name Pric
Description	Enter Item Description

WaiterBots

Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc

Processing Orders

Order ID: 60478eee965408001575671d

Egg Rice
Tomato Soup
Mixed Rice

- Status: Pending
- Table: 2
- Amount: 520

6.3. Desktop Application of Operator

The Desktop application is for the operator of the restaurant. The operator should login to this application using his credentials. Then he can accept and confirm or cancel orders using this desktop application. He will get real time notifications when a customer places an order, after the delivery is complete or the delivery is affected by some obstacle.

This desktop application is developed in a way that it can be installed and used in any operating system. You can install this application in Linux, Windows or even in macOS as well.

- We used React.js for the development of the desktop application. React can build fast, scalable web applications due to its DOM rendering architecture. And it also has a larger community support and a vast number of libraries.
- Redux is used for state management.
- Axios is used for communication with the backend server.
- Electron.js was used to build the cross-platform desktop application.
- This is the UI of the operator. Accepting orders, deploying robot and changing item status can be done via the desktop application.

6.4. Communication inside the Desktop application

Desktop application mainly communicates with WaiterBots via the MQTT protocol. There is a custom MQTT broker running inside the control unit and it will connect to the desktop application via WebSockets.

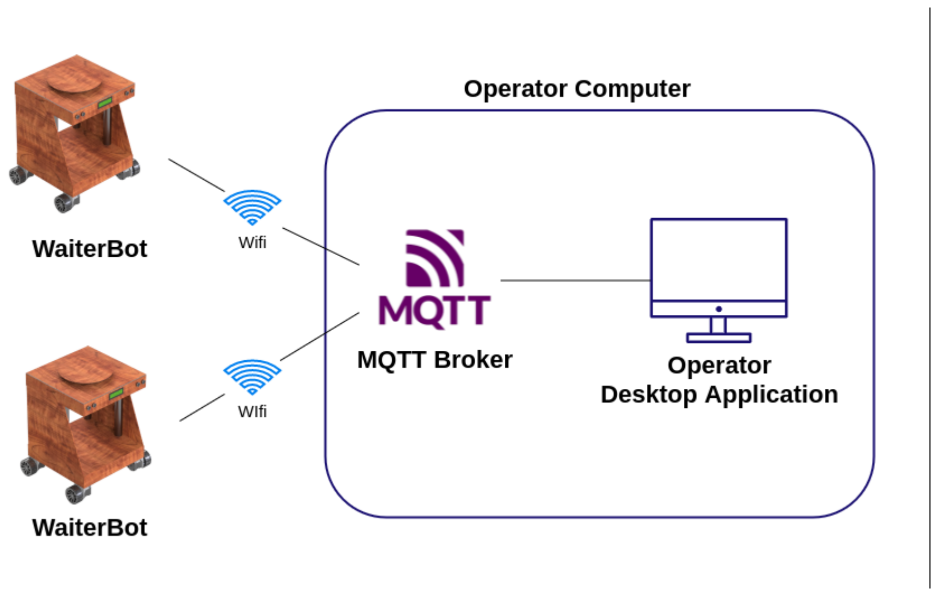
When publishing and subscribing to the WaiterBots, the Desktop application will use the following topics.

Subscribe:

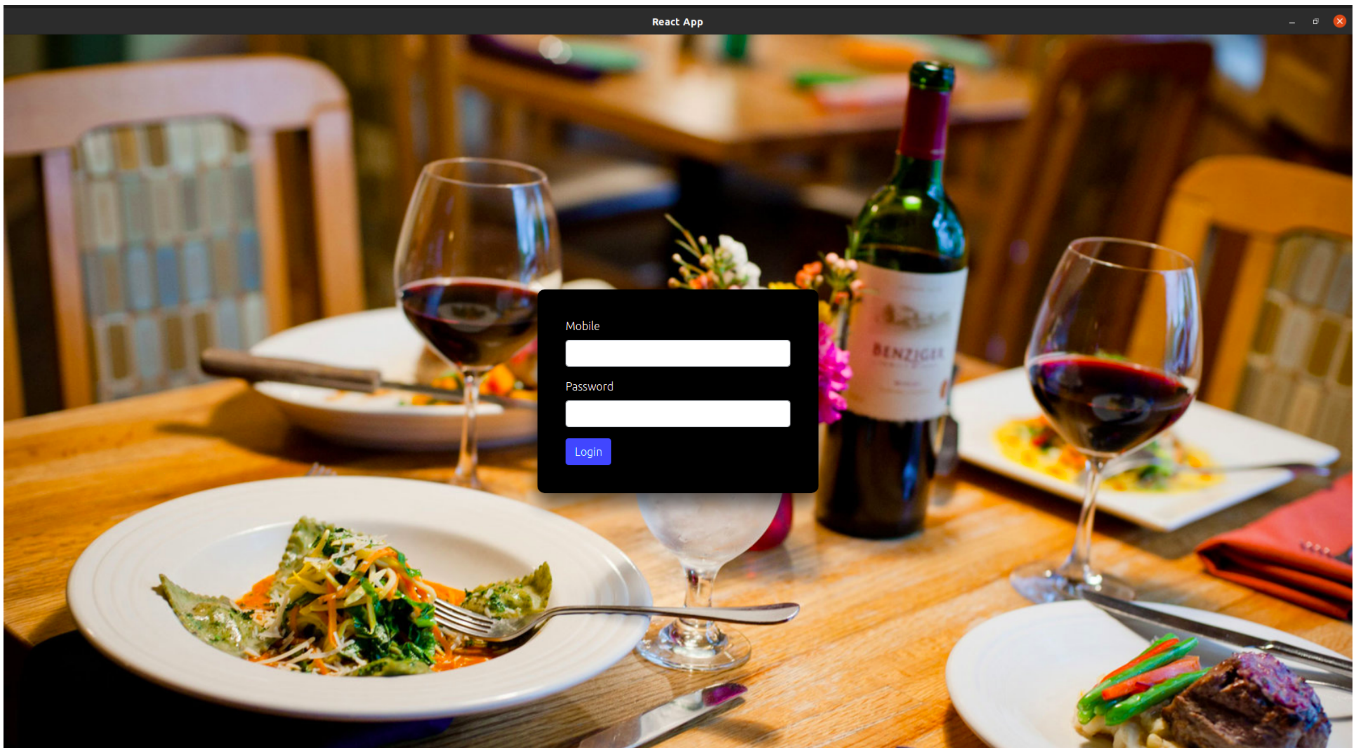
operator/{shop_id}/#

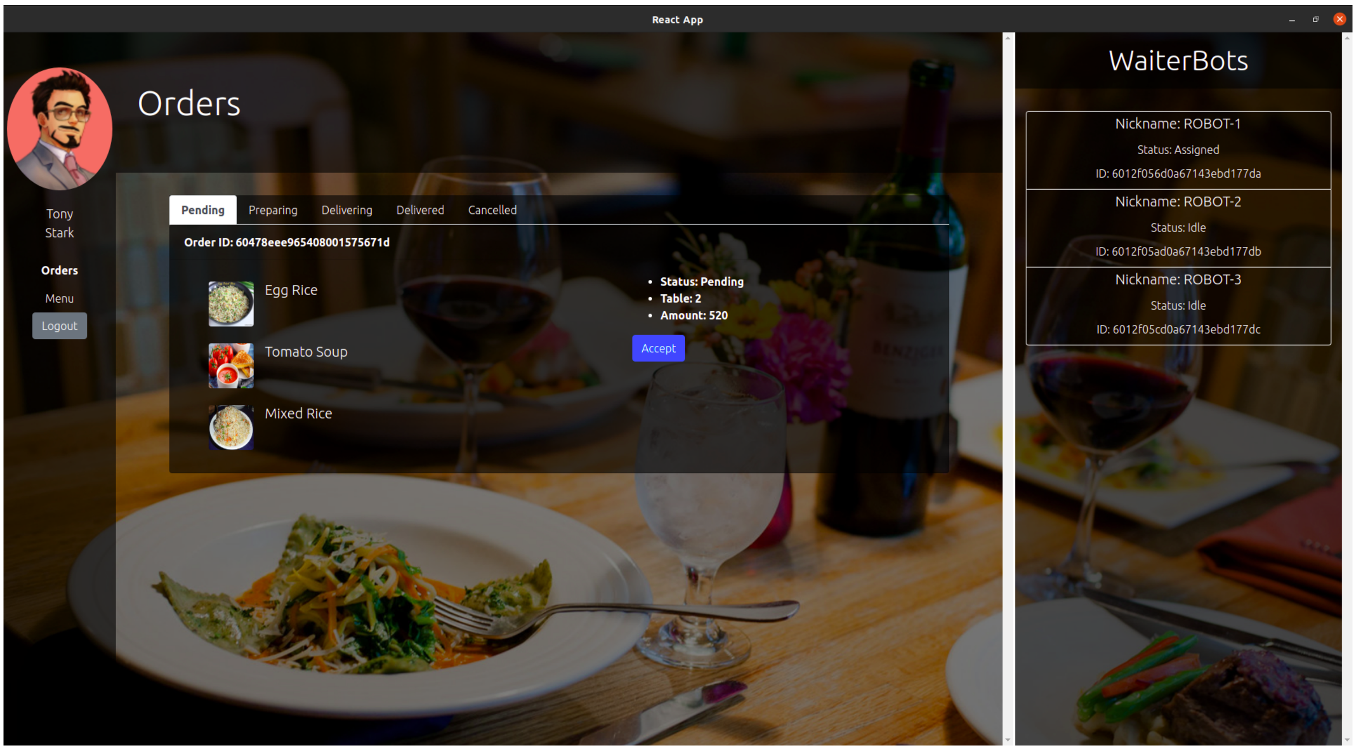
Publish:

waiterbot/{shop_id}/{robot_id}



6.5. Operator UI





Orders

Order ID: 60478eee965408001575671d

Egg Rice

Tomato Soup

Mixed Rice

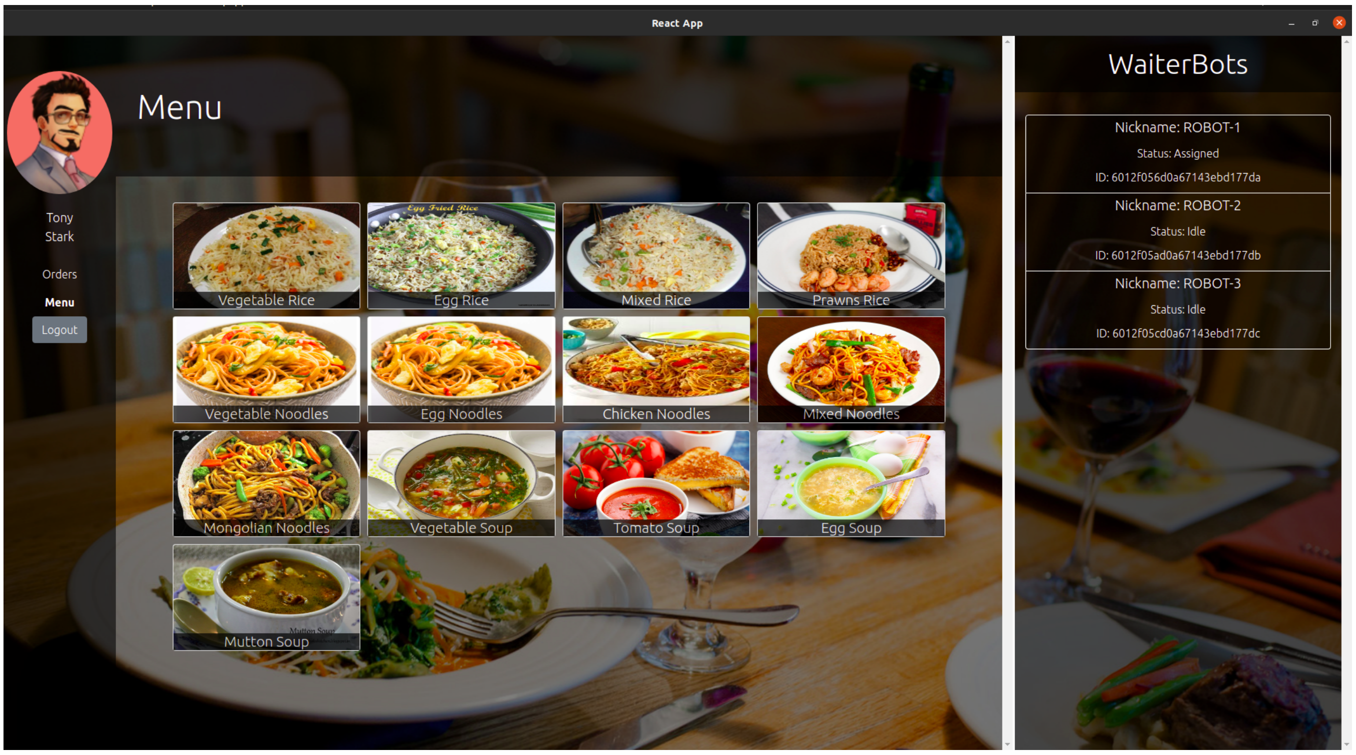
- **Status: Pending**
- **Table: 2**
- **Amount: 520**

WaiterBots

Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc



React App

Menu

Tony Stark

Orders

Menu

Logout

Vegetable Rice | Egg Rice | Mixed Rice | Prawns Rice

Vegetable Noodles | Egg Noodles | Chicken Noodles | Mixed Noodles

Mongolian Noodles | Vegetable Soup | Tomato Soup | Egg Soup

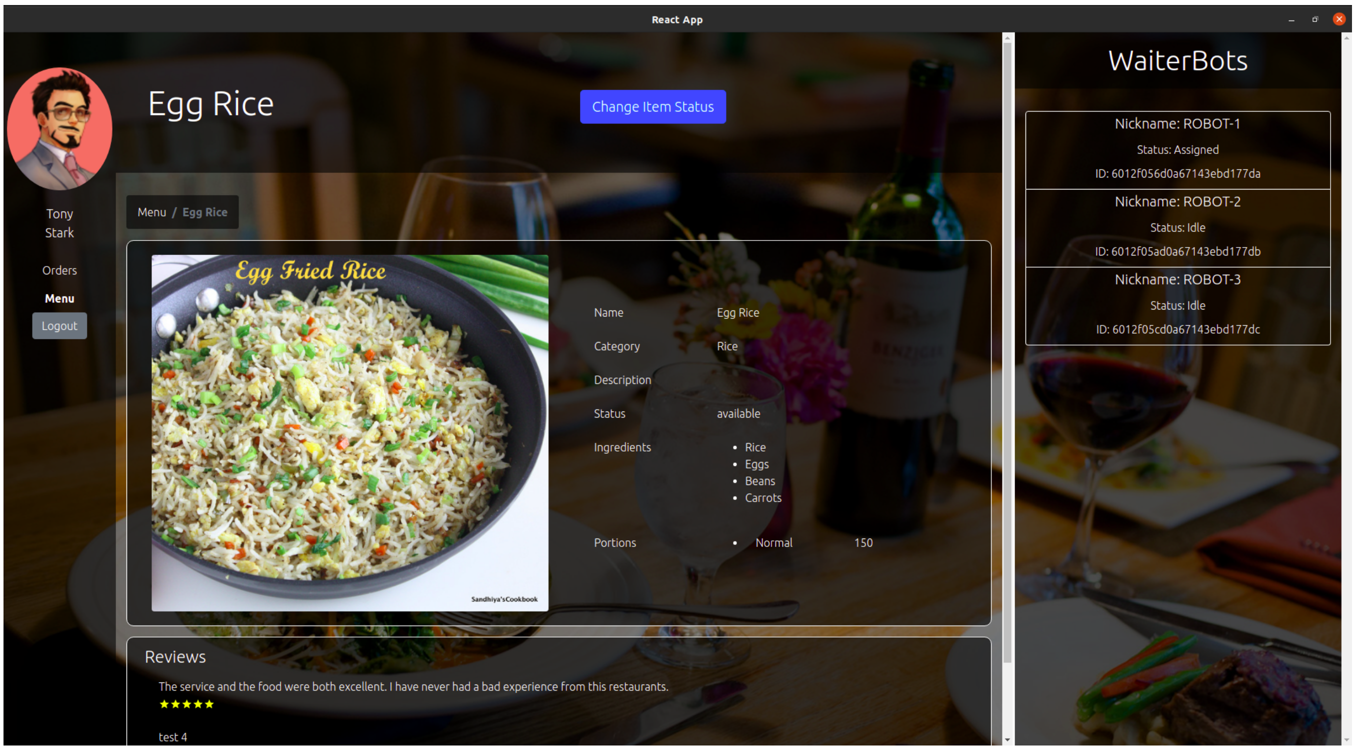
Mutton Soup

WaiterBots

Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc



React App

Egg Rice

Change Item Status

Tony Stark

Orders

Menu

Logout

Menu / Egg Rice

Name	Egg Rice
Category	Rice
Description	
Status	available
Ingredients	• Rice • Eggs • Beans • Carrots
Portions	• Normal 150

Reviews

The service and the food were both excellent. I have never had a bad experience from this restaurants.
★★★★★

test 4

WaiterBots

Nickname: ROBOT-1
Status: Assigned
ID: 6012f056d0a67143ebd177da

Nickname: ROBOT-2
Status: Idle
ID: 6012f05ad0a67143ebd177db

Nickname: ROBOT-3
Status: Idle
ID: 6012f05cd0a67143ebd177dc

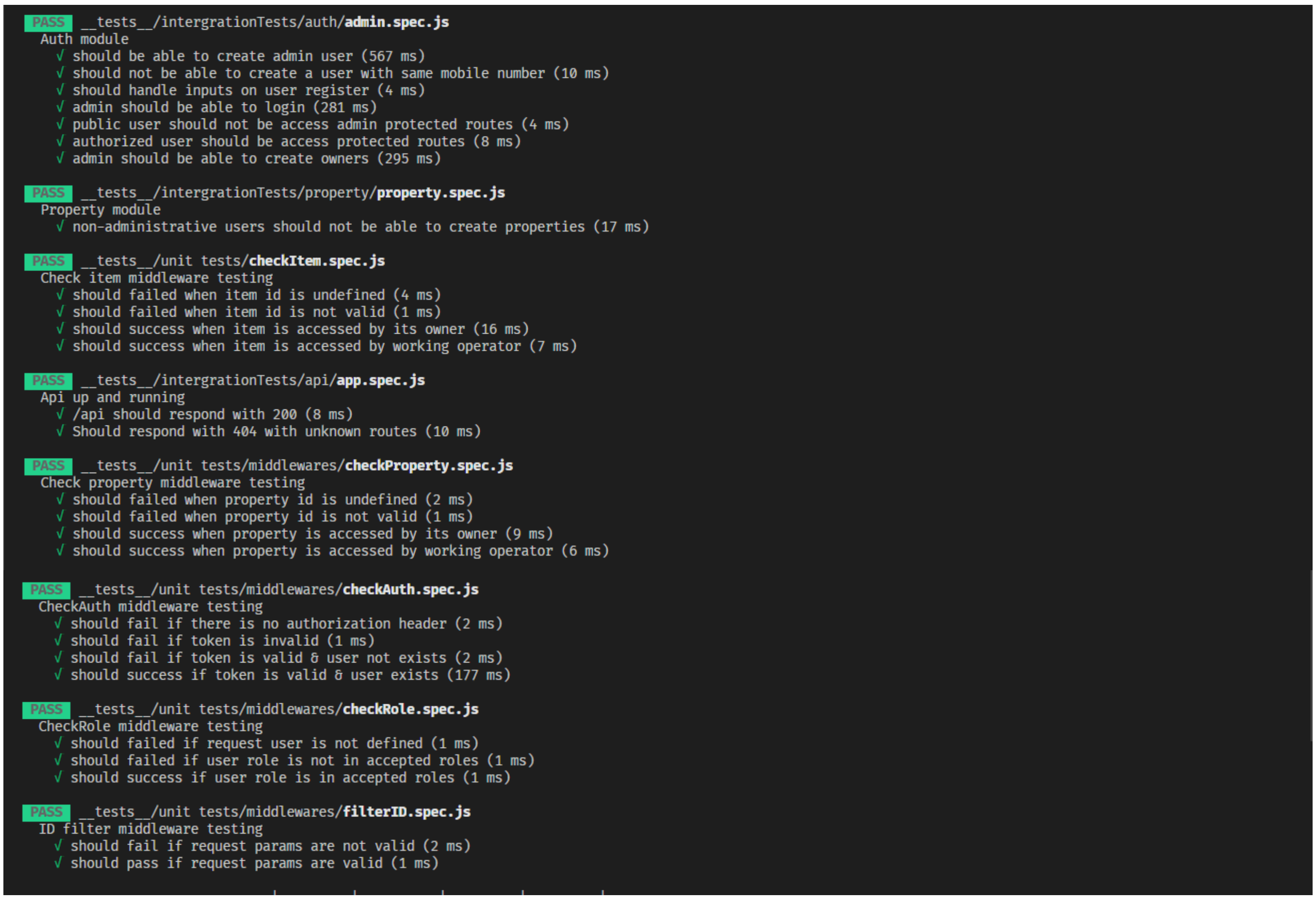
7. Testing

Mainly we are considering Integration testing and Unit testing. For unit testing we are using jest.js framework.And for integration testing we are using supertest with jest.js.

7.1. Software Testing Plan and Results

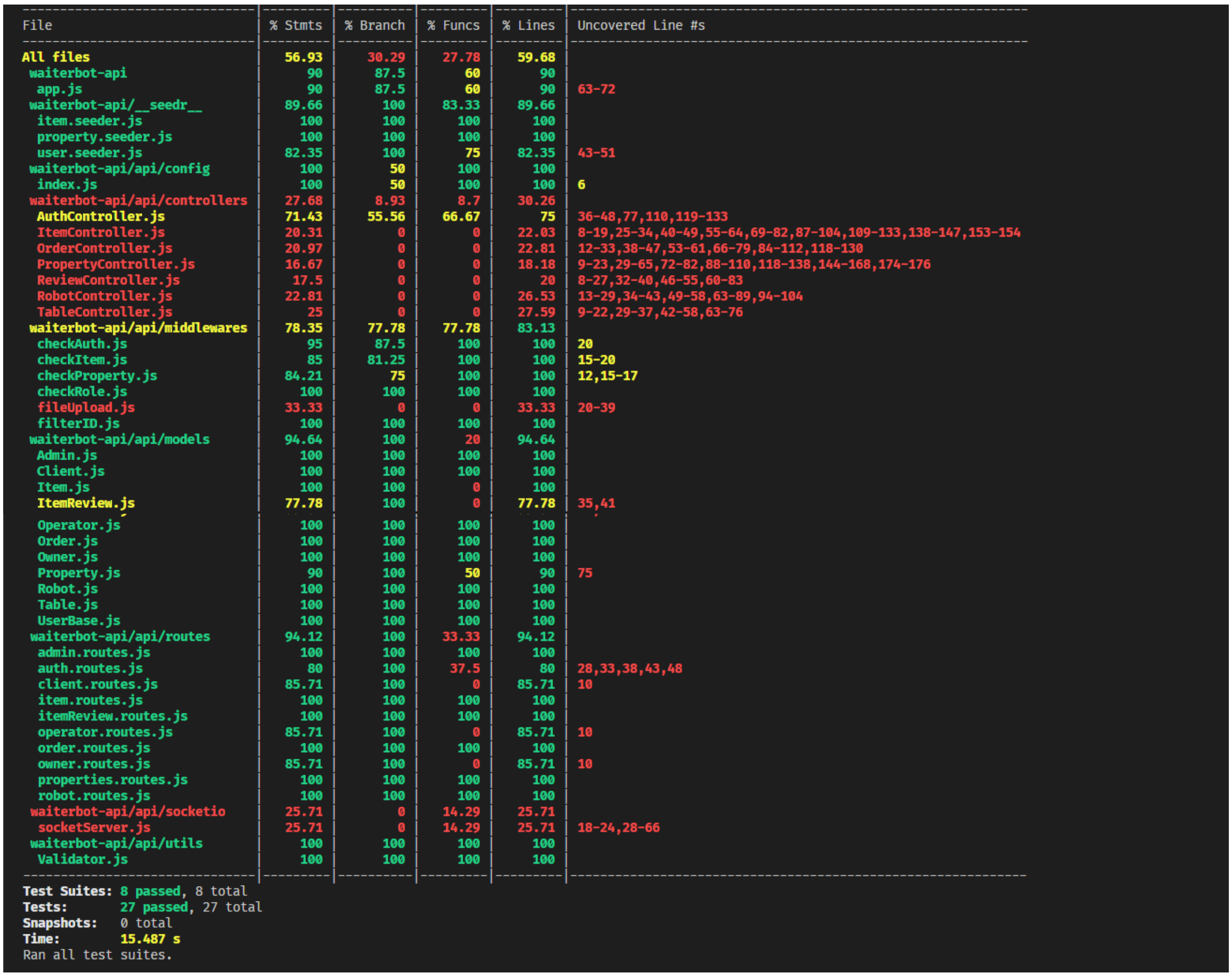
Test	Testing Components	Reason
Auth module functional test	Auth module	• To check authentication and authorization functionality.
Auth Middleware Unit testing	Auth middleware	• To check authorization process is functioning properly.
CheckRole Middleware Unit testing	CheckRole middleware	• To check RBAC is functioning properly.
Email Test	Email validation function (for an empty email, for a correct email)	• To test whether validation works properly for email validation function
Password Test	Password validation function (for a password less than 8 chars, for a correct password)	• To test whether validation works properly for password validation function

7.2. Software Testing Screenshots



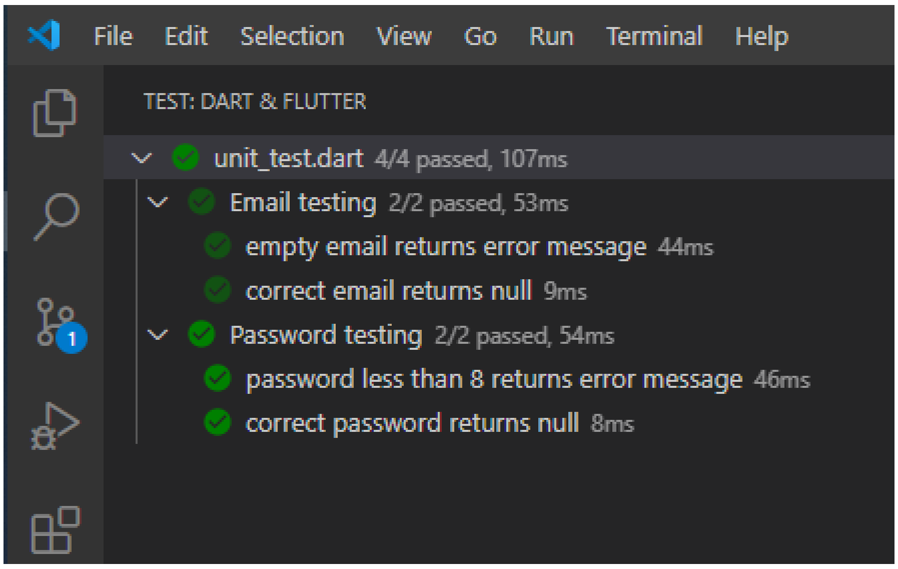
```
 PASS  __tests__/intergrationTests/auth/admin.spec.js
  Auth module
    √ should be able to create admin user (567 ms)
    √ should not be able to create a user with same mobile number (10 ms)
    √ should handle inputs on user register (4 ms)
    √ admin should be able to login (281 ms)
    √ public user should not be access admin protected routes (4 ms)
    √ authorized user should be access protected routes (8 ms)
    √ admin should be able to create owners (295 ms)

 PASS  __tests__/intergrationTests/property/property.spec.js
  Property module
    √ non-administrative users should not be able to create properties (17 ms)

 PASS  __tests__/unit tests/checkItem.spec.js
  Check item middleware testing
    √ should failed when item id is undefined (4 ms)
    √ should failed when item id is not valid (1 ms)
    √ should success when item is accessed by its owner (16 ms)
    √ should success when item is accessed by working operator (7 ms)

 PASS  __tests__/intergrationTests/api/app.spec.js
  Api up and running
    √ /api should respond with 200 (8 ms)
    √ Should respond with 404 with unknown routes (10 ms)

 PASS  __tests__/unit tests/middlewares/checkProperty.spec.js
  Check property middleware testing
    √ should failed when property id is undefined (2 ms)
    √ should failed when property id is not valid (1 ms)
    √ should success when property is accessed by its owner (9 ms)
    √ should success when property is accessed by working operator (6 ms)

 PASS  __tests__/unit tests/middlewares/checkAuth.spec.js
  CheckAuth middleware testing
    √ should fail if there is no authorization header (2 ms)
    √ should fail if token is invalid (1 ms)
    √ should fail if token is valid & user not exists (2 ms)
    √ should success if token is valid & user exists (177 ms)

 PASS  __tests__/unit tests/middlewares/checkRole.spec.js
  CheckRole middleware testing
    √ should failed if request user is not defined (1 ms)
    √ should failed if user role is not in accepted roles (1 ms)
    √ should success if user role is in accepted roles (1 ms)

 PASS  __tests__/unit tests/middlewares/filterID.spec.js
  ID filter middleware testing
    √ should fail if request params are not valid (2 ms)
    √ should pass if request params are valid (1 ms)
```

7.3. API Code Coverage Report



```
--------------------------------|---------|----------|---------|---------|--------------------------------------------------------------
File                            | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------------------|---------|----------|---------|---------|--------------------------------------------------------------
All files                       |   56.93 |    30.29 |   27.78 |   59.68 |
 waiterbot-api                  |      90 |     87.5 |      60 |      90 |
  app.js                        |      90 |     87.5 |      60 |      90 | 63-72
 waiterbot-api/__seedr__        |   89.66 |      100 |   83.33 |   89.66 |
  item.seeder.js                |     100 |      100 |     100 |     100 |
  property.seeder.js            |     100 |      100 |     100 |     100 |
  user.seeder.js                |   82.35 |      100 |      75 |   82.35 | 43-51
 waiterbot-api/api/config       |     100 |       50 |     100 |     100 |
  index.js                      |     100 |       50 |     100 |     100 | 6
 waiterbot-api/api/controllers  |   27.68 |     8.93 |     8.7 |   30.26 |
  AuthController.js             |   71.43 |    55.56 |   66.67 |      75 | 36-48,77,110,119-133
  ItemController.js             |   20.31 |        0 |       0 |   22.03 | 8-19,25-34,40-49,55-64,69-82,87-104,109-133,138-147,153-154
  OrderController.js            |   20.97 |        0 |       0 |   22.81 | 12-33,38-47,53-61,66-79,84-112,118-130
  PropertyController.js         |   16.67 |        0 |       0 |   18.18 | 9-23,29-65,72-82,88-110,118-138,144-168,174-176
  ReviewController.js           |    17.5 |        0 |       0 |      20 | 8-27,32-40,46-55,60-83
  RobotController.js            |   22.81 |        0 |       0 |   26.53 | 13-29,34-43,49-58,63-89,94-104
  TableController.js            |      25 |        0 |       0 |   27.59 | 9-22,29-37,42-58,63-76
 waiterbot-api/api/middlewares  |   78.35 |    77.78 |   77.78 |   83.13 |
  checkAuth.js                  |      95 |     87.5 |     100 |     100 | 20
  checkItem.js                  |      85 |    81.25 |     100 |     100 | 15-20
  checkProperty.js              |   84.21 |       75 |     100 |     100 | 12,15-17
  checkRole.js                  |     100 |      100 |     100 |     100 |
  fileUpload.js                 |   33.33 |        0 |       0 |   33.33 | 20-39
  filterID.js                   |     100 |      100 |     100 |     100 |
 waiterbot-api/api/models       |   94.64 |      100 |      20 |   94.64 |
  Admin.js                      |     100 |      100 |     100 |     100 |
  Client.js                     |     100 |      100 |     100 |     100 |
  Item.js                       |     100 |      100 |       0 |     100 |
  ItemReview.js                 |   77.78 |      100 |       0 |   77.78 | 35,41
  Operator.js                   |     100 |      100 |     100 |     100 |
  Order.js                      |     100 |      100 |     100 |     100 |
  Owner.js                      |     100 |      100 |     100 |     100 |
  Property.js                   |      90 |      100 |      50 |      90 | 75
  Robot.js                      |     100 |      100 |     100 |     100 |
  Table.js                      |     100 |      100 |     100 |     100 |
  UserBase.js                   |     100 |      100 |     100 |     100 |
 waiterbot-api/api/routes       |   94.12 |      100 |   33.33 |   94.12 |
  admin.routes.js               |     100 |      100 |     100 |     100 |
  auth.routes.js                |      80 |      100 |    37.5 |      80 | 28,33,38,43,48
  client.routes.js              |   85.71 |      100 |       0 |   85.71 | 10
  item.routes.js                |     100 |      100 |     100 |     100 |
  itemReview.routes.js          |     100 |      100 |     100 |     100 |
  operator.routes.js            |   85.71 |      100 |       0 |   85.71 | 10
  order.routes.js               |     100 |      100 |     100 |     100 |
  owner.routes.js               |   85.71 |      100 |       0 |   85.71 | 10
  properties.routes.js          |     100 |      100 |     100 |     100 |
  robot.routes.js               |     100 |      100 |     100 |     100 |
 waiterbot-api/api/socketio     |   25.71 |        0 |   14.29 |   25.71 |
  socketServer.js               |   25.71 |        0 |   14.29 |   25.71 | 18-24,28-66
 waiterbot-api/api/utils        |     100 |      100 |     100 |     100 |
  Validator.js                  |     100 |      100 |     100 |     100 |
--------------------------------|---------|----------|---------|---------|--------------------------------------------------------------
Test Suites: 8 passed, 8 total
Tests:       27 passed, 27 total
Snapshots:   0 total
Time:        15.487 s
Ran all test suites.
```

7.4. Mobile App Testing



```
unit_test.dart 4/4 passed, 107ms
  Email testing 2/2 passed, 53ms
    empty email returns error message 44ms
    correct email returns null 9ms
  Password testing 2/2 passed, 54ms
    password less than 8 returns error message 46ms
    correct password returns null 8ms
```

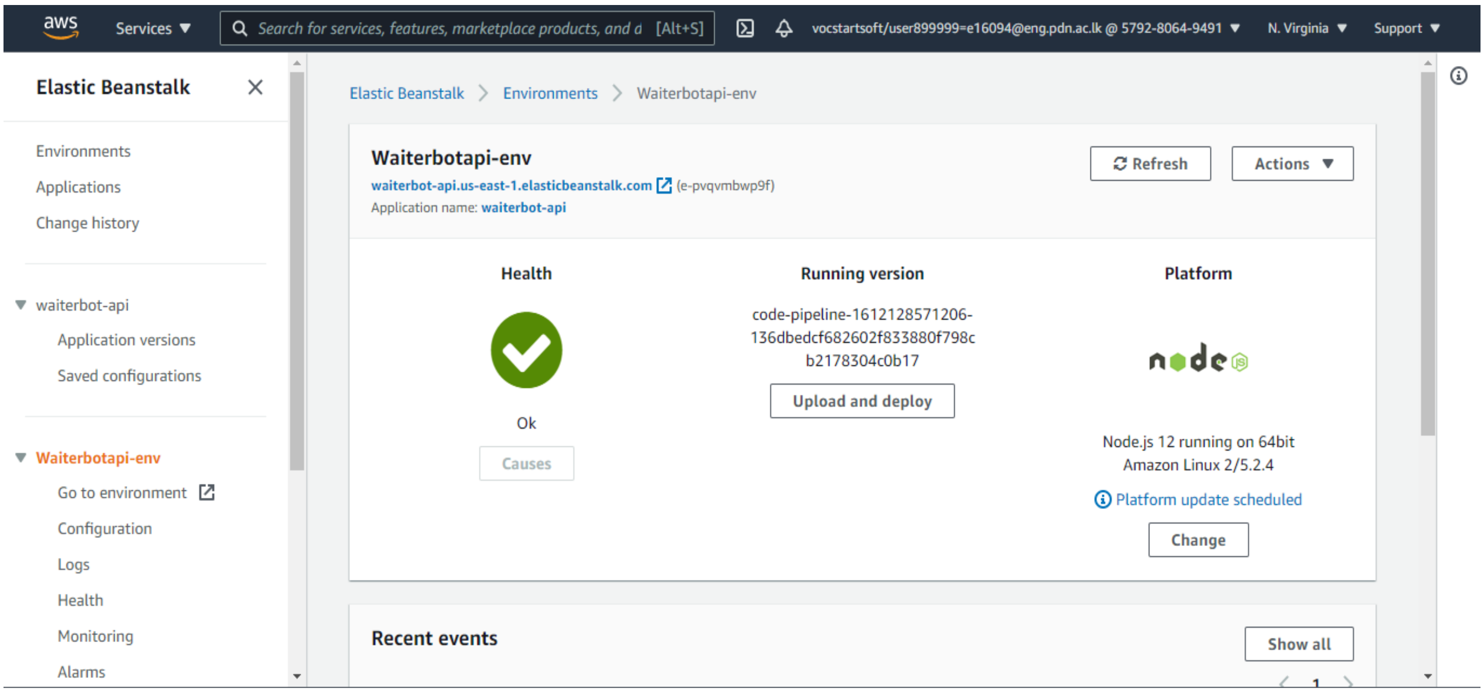
7.5. Hardware Testing Plan

Test	Testing Components	Reason	Testing Procedure
Hardware Structure	Structural components	• Should be able to sustain the weight	• Place different weights
PCB testing	Fabricated PCB	• Proper connectivity • Short circuits	• Test using multimeter
Power testing (Voltage and Current)	All electronic components	• Proper voltages and currents • Check for generation of noise(step down module) • Could damage components	• Before connecting components. • Test using a multimeter and an oscilloscope
Hardware Structure	Structural components	• Should be able to sustain the weight	• Place different weights
PCB testing	Fabricated PCB	• Proper connectivity • Short circuits	• Test using multimeter
Power testing (Voltage and Current)	All electronic components	• Proper voltages and currents • Check for generation of noise(step down module) • Could damage components	• Before connecting components. • Test using a multimeter and an oscilloscope

8. Deployment

8.1. WaiterBot API

- WaiterBot API is deployed on AWS.
  http://waiterbot-api.us-east-1.elasticbeanstalk.com/api



Waiterbotapi-env
waiterbot-api.us-east-1.elasticbeanstalk.com (e-pvqvmbwp9f)
Application name: waiterbot-api

Health: Ok

Running version: code-pipeline-1612128571206-136dbedcf682602f833880f798cb2178304c0b17

Platform: Node.js 12 running on 64bit Amazon Linux 2/5.2.4

8.2. Owner Web Application

- The WaiterBot Owner web application is deployed on heroku.
  https://waiterbot-owner.herokuapp.com/menu