

**San Jose State University**

**Computer Engineering Department**



**CMPE 281 -01: Cloud Technologies**

**Project Title:**

Option #1: Robot Cloud (Food Delivery Robot)  
Deliverable #2 - Component Design

**Under the supervision of the professor:**

Dr. Jerry Gao

**Submitted by:**

Name : FNU BUTUL PARVEEN

Email ID : [butul.parveen@sjsu.edu](mailto:butul.parveen@sjsu.edu)

Student ID : 015918227

## Table of Contents

Contents
<b>1. Component overview</b>
1.1 Component purpose
1.2 Component objective
1.3 Function scope
1.4 Usage
<b>2. Component application interface design and analysis</b>
2.1 Detailed component API interfaces and descriptions in terms of their parameters, such as RESTFUL APIs
<b>3. Component function design, data and DB design, and behavior analysis</b>
3.1 Component logic design, such as class diagrams
3.2 Program function descriptions using tables, flowcharts, and program pseudo codes.
3.3. Program behaviors using state diagrams and class sequence diagrams.
<b>4. Component business logics</b>
4.1 Detailed business logics using decision tables
<b>5. Component graphic user interface design</b>
5.1 Component GUI style
5.2 Storyboard
5.3 Function partition tree
5.3 UI markup diagram and GUI templates

# Component #3 – Remote online robot tracking and controlling

## 1. Component Overview

### 1.1 Component purpose:

The remote online robot tracking and controlling component's reason is to supply the clients with the interactive system to control and track the robot remotely through a cloud stage. This stage will serve as an interactive layer between the user and the versatile food delivery robot, which in this case will be a simulation through AWS Robomaker.

### 1.2 Objectives:

The purpose of the Food Delivery Robot remote online for tracking and controlling the component destinations is to supply the user destinations with the food delivery with the capacity to alter the status of the robot from one state to another state. For example, the robot will deliver the food from one state to another state based on the robot's usage. We can track the billing application for the robot and be able to control the robot by monitoring the status. This component also incorporates following the robot remotely through the front-end stage to recognize the area of the robot.

### 1.3 Function scope:

The functional scope of this component is restricted to the following operations and the controlling capabilities given to the user through an intuitively graphical user interface stage.

### 1.4 Usage:

The following robot cloud for food delivery is based on the source and destination addresses and delivers the food to the user based on the robot booking and use of the robot for food delivery. The user will interact with the robot via the GUI dashboard after registering for the food delivery robot application. The following gives the user intelligent buttons within the GUI to distinguish the current status and position of the robot. The control of the robot gives buttons to the client within the GUI to alter the status of the robot to the following states: connected, operated, stopped, inactive, Idle and active.

**Track user location:** The Robot service management connectivity protocol is used to track the user's street position, longitude, and latitude data.

**Track robot location:** The robot service management connectivity protocol is used to track the robot street location, longitude, and latitude data.

**Track simulator status:** The status of the simulator, including whether it is connected via the robot service management connectivity protocol or is active or inactive.

**Track the status of the robot:** The robot service management connectivity protocol's clearly defined APIs allow it to be determined whether the robot is stationary, moving, going forward, or moving backward.

**Track the delivery:** This component's APIs that retrieve information about deliveries that are encountered along the way at the destination are among the most crucial.

## 2. Component Application Interface Design And Analysis

This component's application interface contains APIs to control the robot's states and track it using its position.

### 2.1 Detailed component API interfaces and descriptions of their parameters, such as RESTFUL APIs

#### Robot.Actor:

The AWS robot simulation using a simulator identifies each robot that is crucial to the simulation, including users, robots, sensors, and food delivery locations based on the source and destination point. Since we cannot send data to the robot simulator, we use the standard RESTful API components instead of POST for this.

S.No	Response Type	API Method	URL	Parameters	Attributes
1	Response : String	POST	RegisterRobot(userId)	userID:int	name:String password:Hashed password email_id:String Phone_no:double int
2	Response : Object	POST	getUserRobot (UserID)	UserID: int	robot_name:String Robot_type:String Version:float
3	Response Status: String	GET	getRobotState(RobotID)	robotID:int	RobotStatus:String Robotmode:String RobotDetails:dict
4	Response :String	GET	getDeactivatedRobot(Robot ID)	robotID:int	name:String password:Hashed password email_id:String Phone_no:double int Robot_type:String Version:float

### Robot.rental\_station

S.No	Response Type	API Method	URL	Parameters	Attributes
1	Response : String	GET	getRental_station(station_id)	station_id:int	name:String address:String

### Robot.user(Client):

S.No	Response Type	API Method	URL	Parameters	Attributes
1	Response : String	POST	Registeruser(userID)	UserID:int	name:String address:String password email_id:String Phone_no:double int billing_details:float
2	Response : String	GET	robot.management(userID)	aws_robotSimulation robo_management	name:String address:String password email_id:String Phone_no:double int

### Robot\_management:

S.No	Response Type	API Method	URL	Parameters	Attributes
1	Response : String	POST	Registeruser(userID)	UserID:int	name:String address:String password email_id:String Phone_no:double int billing_details:float

### Robot\_Map:

S.No	Response Type	API Method	URL	Parameters	Attributes
1	location : Object	POST	getRobot Location (robotID)	robotID:int	Pickup_point : Timestamp Destination:Timestamp TripDuration:Int Delivery_cost:float status:String arrival_time:time
2	response : String	GET	updateRobotStatus(robotID,status)	robotID:int Status :String	maintenance_check:String
3	Path:Object	GET	getNavigatedPath (robotID)	robotID:int	name:String Distance:float Robot.location:object robot.TrackwayPoint:object

### Robot service RESTFUL APIS:

S.No	Response Type	API Method	URL	Parameters	Functionality
1	response : String	GET	initiateRobotstatus (robotID)	robotID:int	This API focuses on the essential data needed, such as current location and current status.
2	location : Object	POST	getRobot Location (robotID)	robotID:int robot_location: object	This API handles everything from the current, pickup, and destination locations to determining the robot condition, including whether it is moving forward or backward or sitting still, as well as the status of the food delivery of the robot.

3	response: String	GET,POST	getadmin (admin_id) postRent al_station (station_id)	admin_id:int robotID:int	obtains the critical data the administrator needs to make wise judgments.
4	response: String	GET	getRobot Maintenance(rm_id)	robotID:int rm_id:int	The api manages the robot location and keeps tracking of the robot.



### 3. Component Function Design, Data And Db Design, And Behavior Analysis

#### 3.1 Component Logic Design, Such As Class Diagrams

The component class diagram is shown below, and it shows the different actions the user can take to control the robot. To manage the robot's initialization, we provide methods like create robots and quitRobot. We have techniques like robot movement that assist the user in controlling the robot's movement.

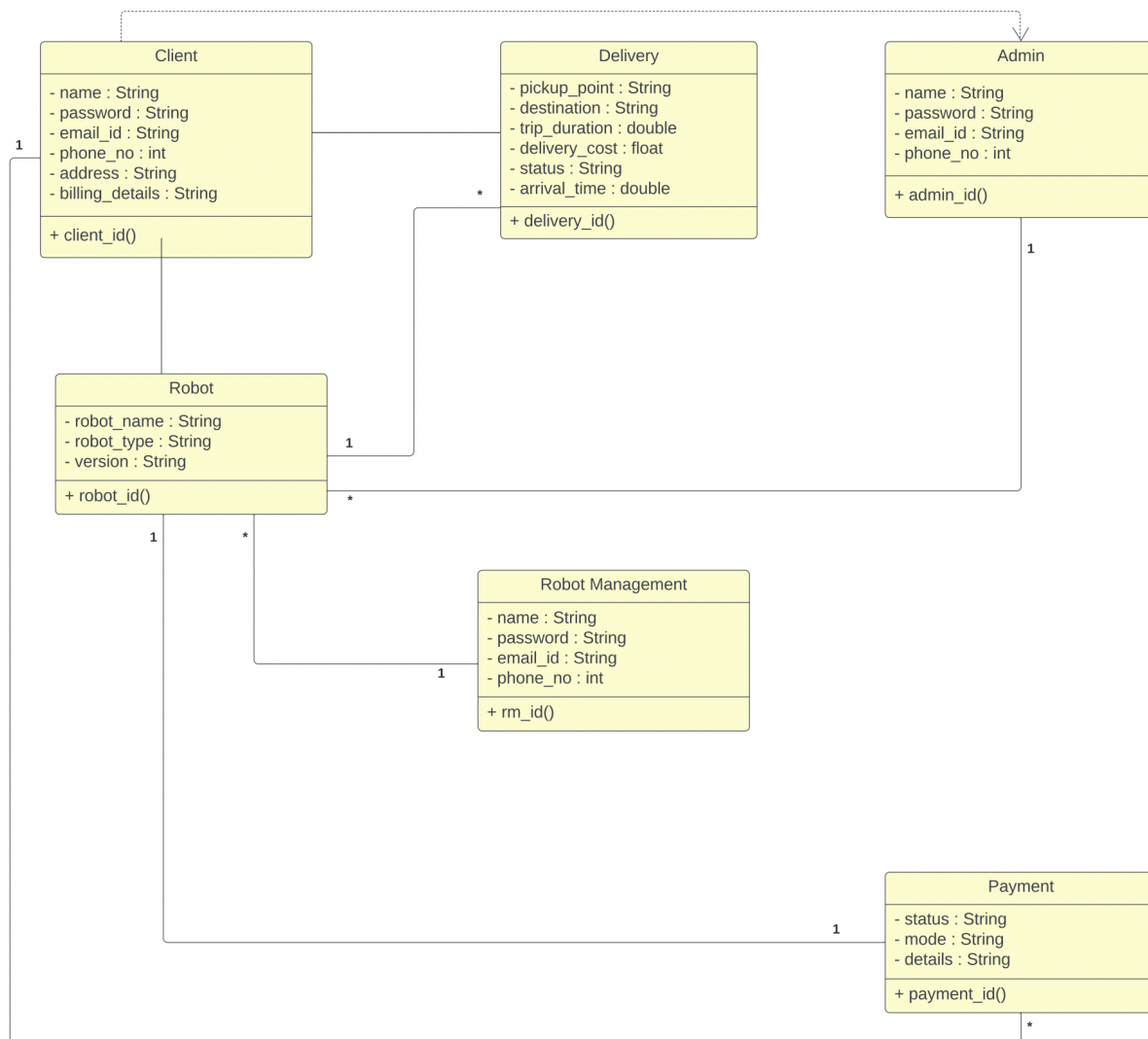


Fig 1:Class Diagram of Robot Management and Control

#### 4.1.2 Relation DB design - MySQL:

The interconnected entities in the system are correctly described using an entity-relationship diagram. It explains the connections between six different things in the diagram below. Each entity is also mapped to its attributes, and a single diagram skillfully illustrates the link between the attributes and the entities as well.

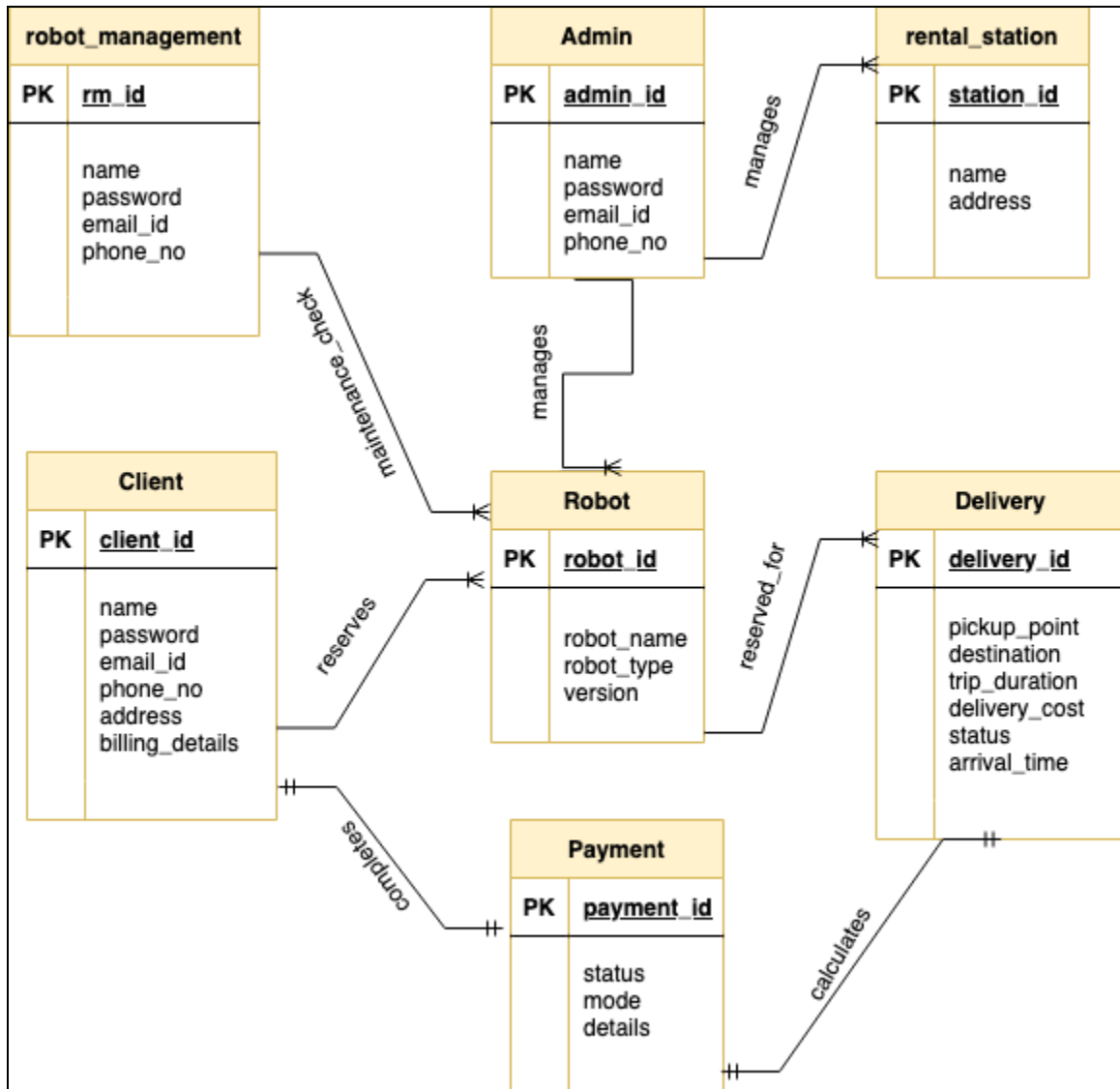


Fig 2 : Relation DB design - MySQL

### 4.1.3 NoSQL DB Design

Traditional databases may manage user and robot data, but non-relational databases are required for sensor data, including photos from all angles, robot movement data, robot tracking information, and position information that will be continuously received, stored, and used for processing.

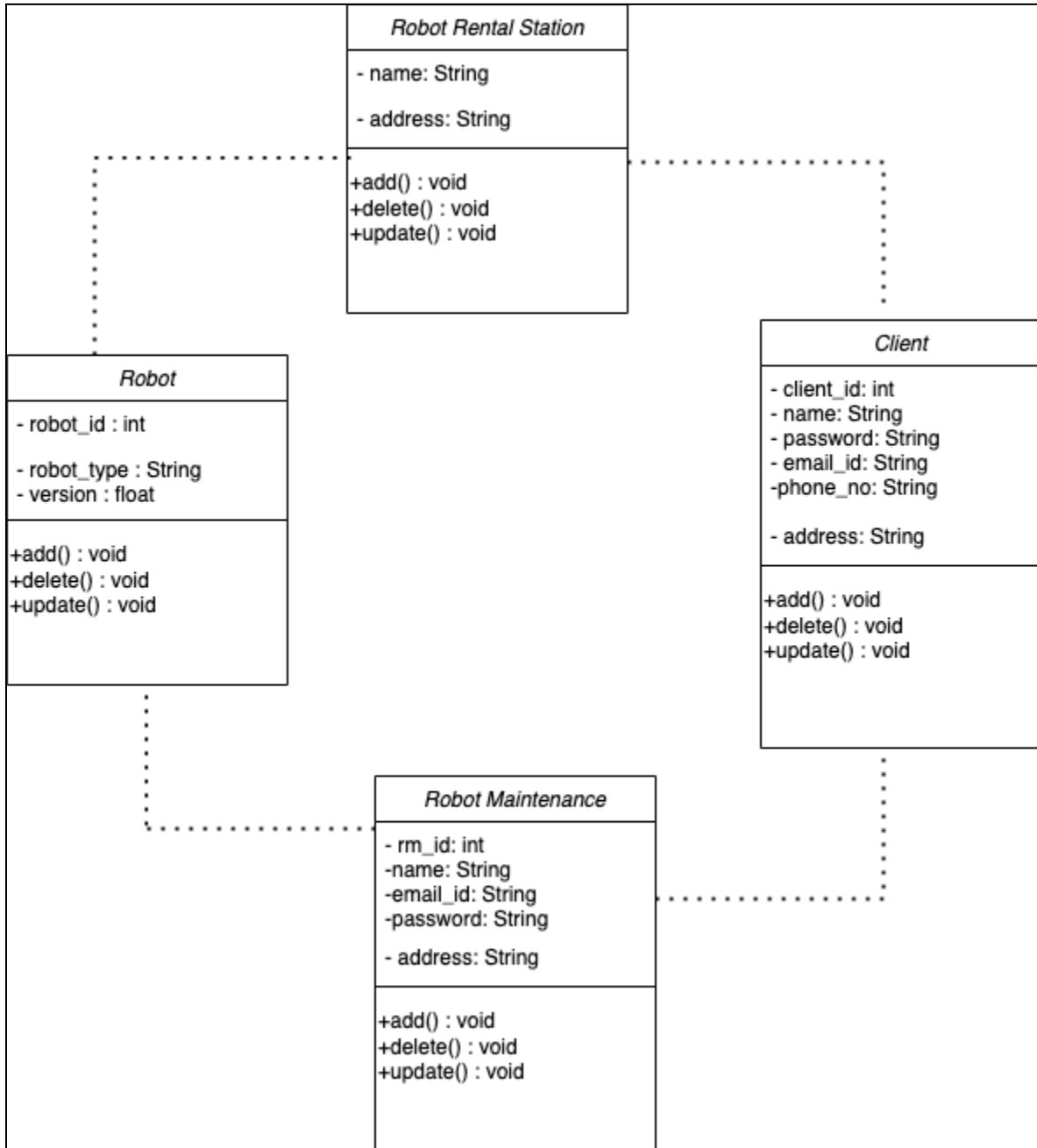


Fig 3 : NoSQL DB Design

### 3.2 Program function descriptions using tables, flowcharts, and program pseudo codes.

#### MongoDB:

MongoDB, a NoSQL database, houses robot information. We maintain a database of robots that contains details about each robot, such as its Robo id, Robo path, which is an array of the coordinates it has already traveled through, Robo state, which will be a string indicating whether it is active or suspended, and Run time, which computes the billing information based on user session.

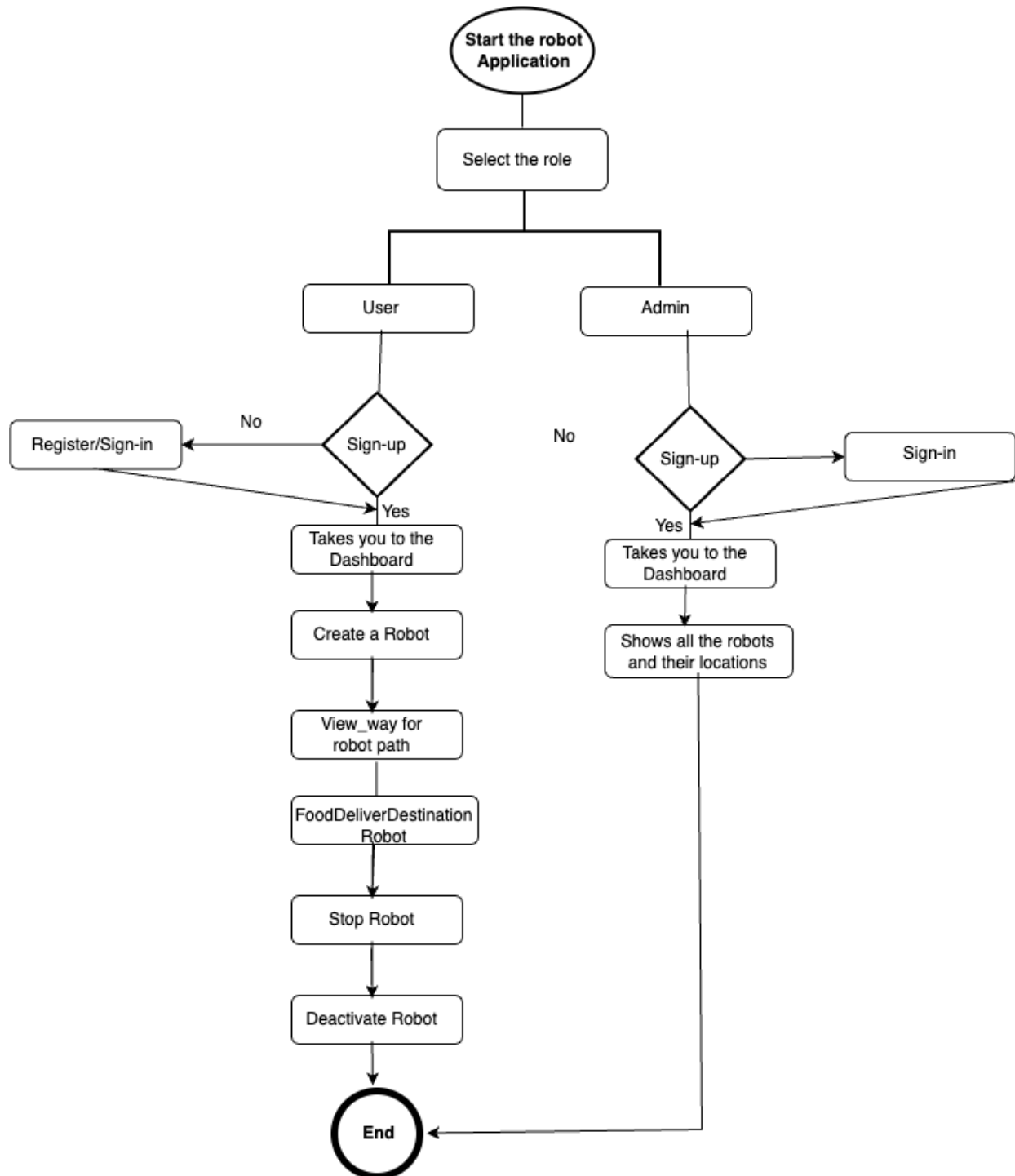
Attribute	Description
robot_id	This attribute is stored as a string for robot_id
robot_path	A list of coordinates in the X and Y plane representing the robot's journey is kept in memory.
robot_state	A string is used to store the robot's current state.
userID	The user's ID is recorded as a string.
run_time	A list of sessions is used to store the duration of the robot.
startSession_time	A string representing the session start time is kept.
endSession_time	A string representing the session end time is kept.

**Table 1 :Program function descriptions using Mongoddb**

#### FlowChart:

The following two characters can be used for each of the characters in the flow chart to view the program function descriptions using the flowchart:

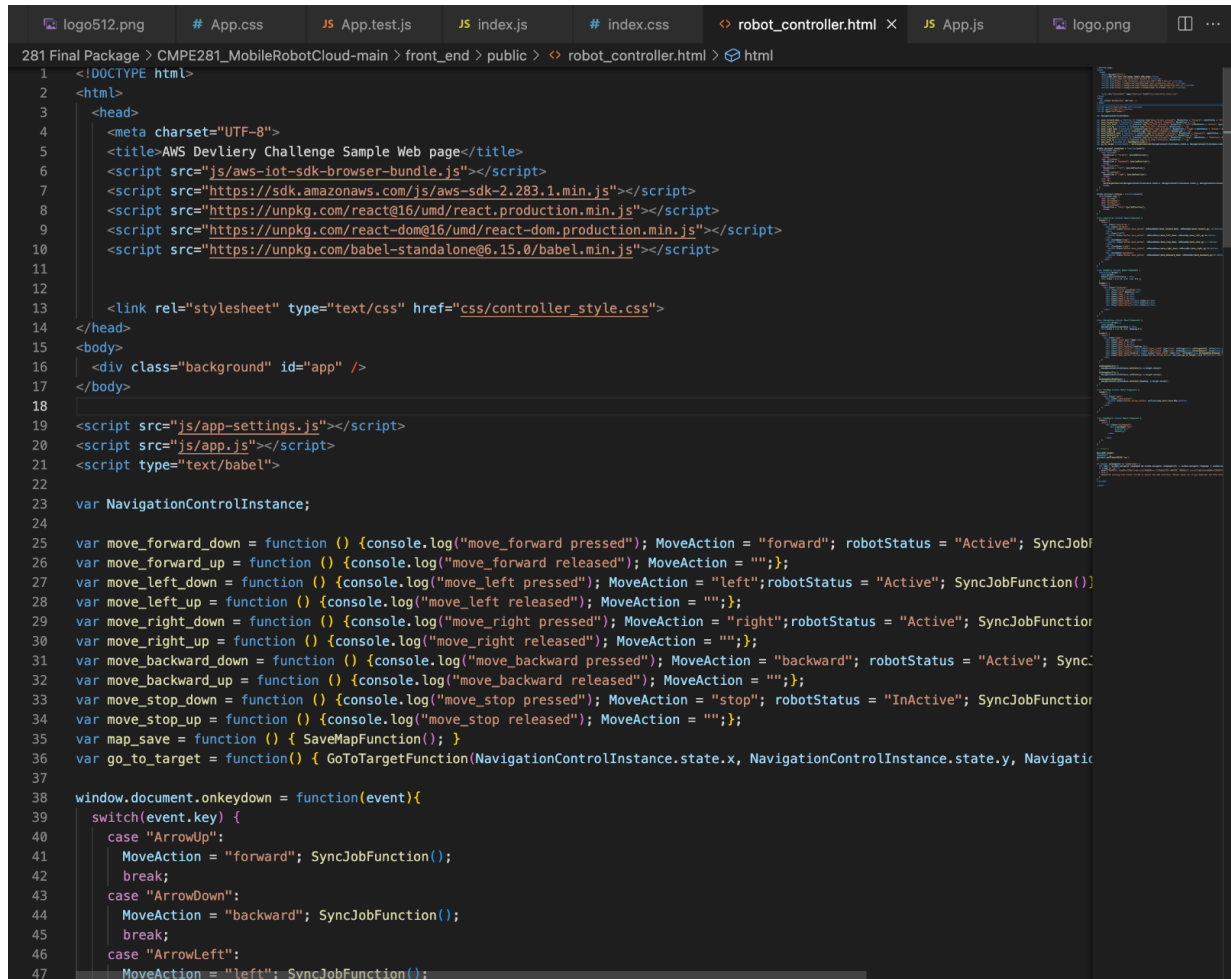
Consider the following use case scenario by following the flowchart. Once a user has logged in, a user can access the dashboard and conduct tasks including creating a new robot, moving the robot, observing the robot's course, halting the robot, and ultimately quitting the robot. The admin dashboard is accessible after logging in. This dashboard has tools for seeing all the robots built into the system and their positions.



**Fig 4 : FlowChart For robot tracking and Controlling for Food Delivery**

## Pseudo Codes for Robot Tracking and Controlling :

The pseudo code for robots is based on the movement of the robot location based on the 3 dimensional axes of the food delivery. The pseudo code of robot tracking and controlling is shown below in the figure.



```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>AWS Devliery Challenge Sample Web page</title>
6     <script src="js/aws-iot-sdk-browser-bundle.js"></script>
7     <script src="https://sdk.amazonaws.com/js/aws-sdk-2.283.1.min.js"></script>
8     <script src="https://unpkg.com/react@16/umd/react.production.min.js"></script>
9     <script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>
10    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
11
12    <link rel="stylesheet" type="text/css" href="css/controller_style.css">
13  </head>
14  <body>
15    <div class="background" id="app" />
16  </body>
17
18  <script src="js/app-settings.js"></script>
19  <script src="js/app.js"></script>
20  <script type="text/babel">
21
22    var NavigationControlInstance;
23
24    var move_forward_down = function () {console.log("move_forward pressed"); MoveAction = "forward"; robotStatus = "Active"; SyncJobf
25    var move_forward_up = function () {console.log("move_forward released"); MoveAction = ""};
26    var move_left_down = function () {console.log("move_left pressed"); MoveAction = "left";robotStatus = "Active"; SyncJobFunction()
27    var move_left_up = function () {console.log("move_left released"); MoveAction = ""};
28    var move_right_down = function () {console.log("move_right pressed"); MoveAction = "right";robotStatus = "Active"; SyncJobFunction
29    var move_right_up = function () {console.log("move_right released"); MoveAction = ""};
30    var move_backward_down = function () {console.log("move_backward pressed"); MoveAction = "backward"; robotStatus = "Active"; Sync
31    var move_backward_up = function () {console.log("move_backward released"); MoveAction = ""};
32    var move_stop_down = function () {console.log("move_stop pressed"); MoveAction = "stop"; robotStatus = "InActive"; SyncJobFunction
33    var move_stop_up = function () {console.log("move_stop released"); MoveAction = ""};
34    var map_save = function () { SaveMapFunction(); }
35    var go_to_target = function() { GoToTargetFunction(NavigationControlInstance.state.x, NavigationControlInstance.state.y, Navigati
36
37    window.document.onkeydown = function(event){
38      switch(event.key) {
39        case "ArrowUp":
40          MoveAction = "forward"; SyncJobFunction();
41          break;
42        case "ArrowDown":
43          MoveAction = "backward"; SyncJobFunction();
44          break;
45        case "ArrowLeft":
46          MoveAction = "left"; SyncJobFunction();
47
```

**Fig 5: Pseudo Code for Robot Tracking and Monitoring**

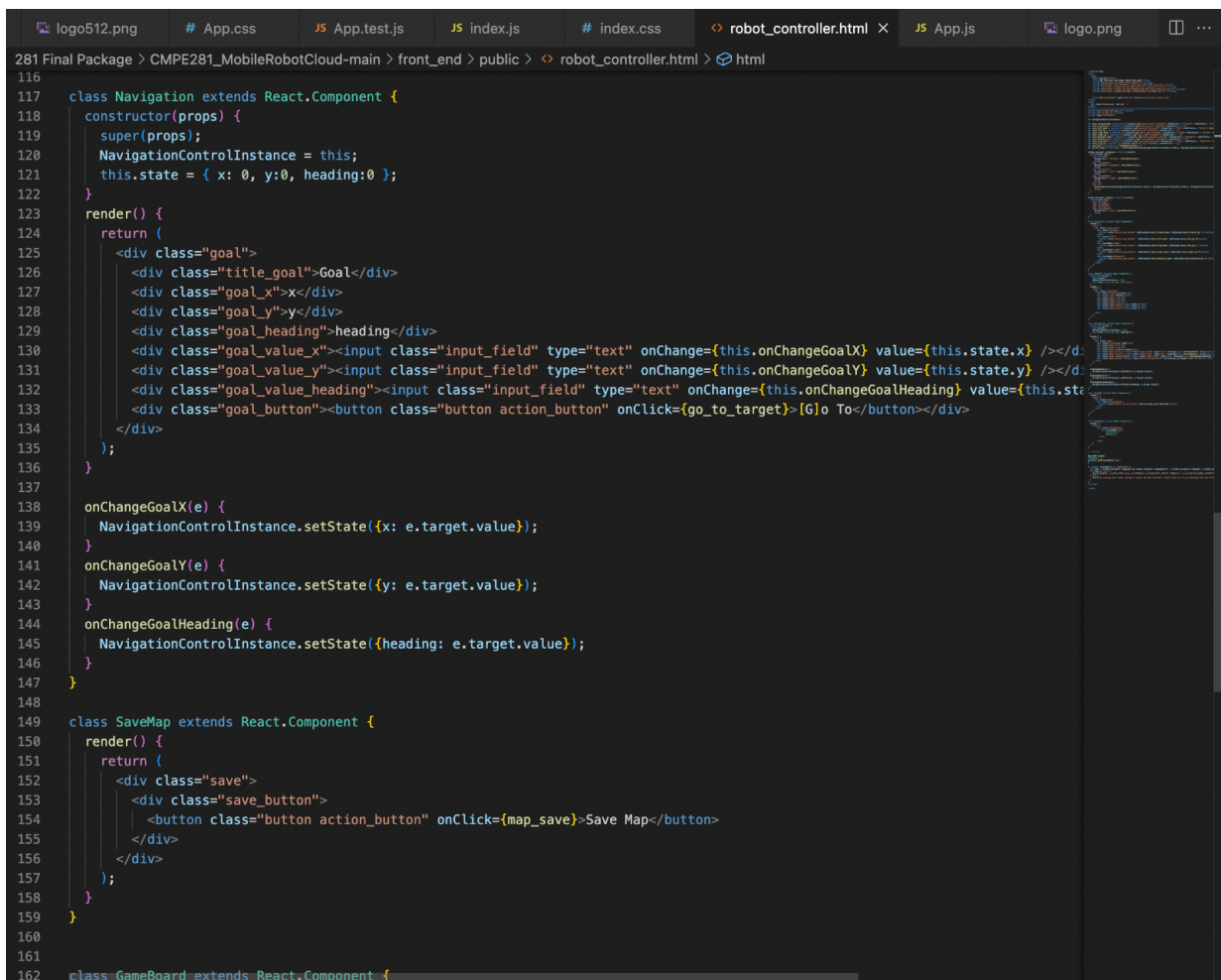
The robot's movement left and right,top, bottom, and center based on the starting position of the robot is considered as the source and destination is considered as the destination for the food delivery robot.

The robot tracking is done by navigating the robot through the status of active or inactive based on the monitoring of the robots in this application . The robot simulation is done by using the AWS robomaker simulator using the simulation.

The below Pseudo code is shown with respect to the different directions of the robot are:

- Left
- Right
- Front
- Back
- Stop
- Start

The Pseudo code for navigating the robot is tracked by the status of the robot, how many robots are active and the pseudo code is shown below in the figure.



```
116
117 class Navigation extends React.Component {
118   constructor(props) {
119     super(props);
120     NavigationControlInstance = this;
121     this.state = { x: 0, y:0, heading:0 };
122   }
123   render() {
124     return (
125       <div class="goal">
126         <div class="title_goal">Goal</div>
127         <div class="goal_x">x</div>
128         <div class="goal_y">y</div>
129         <div class="goal_heading">heading</div>
130         <div class="goal_value_x"><input class="input_field" type="text" onChange={this.onChangeGoalX} value={this.state.x} /></div>
131         <div class="goal_value_y"><input class="input_field" type="text" onChange={this.onChangeGoalY} value={this.state.y} /></div>
132         <div class="goal_value_heading"><input class="input_field" type="text" onChange={this.onChangeGoalHeading} value={this.state.heading} /></div>
133         <div class="goal_button"><button class="button action_button" onClick={go_to_target}>Go To</button></div>
134       </div>
135     );
136   }
137
138   onChangeGoalX(e) {
139     NavigationControlInstance.setState({x: e.target.value});
140   }
141   onChangeGoalY(e) {
142     NavigationControlInstance.setState({y: e.target.value});
143   }
144   onChangeGoalHeading(e) {
145     NavigationControlInstance.setState({heading: e.target.value});
146   }
147 }
148
149 class SaveMap extends React.Component {
150   render() {
151     return (
152       <div class="save">
153         <div class="save_button">
154           <button class="button action_button" onClick={map_save}>Save Map</button>
155         </div>
156       </div>
157     );
158   }
159 }
160
161
162 class GameBoard extends React.Component {
```

**Fig 6: Pseudo Code for Robot Tracking and Monitoring by using Navigation**

Here, controlling the robot by navigating and tracking the status of the robot by monitoring the movement of the robot based on the usage of the user(Client).

### 3.3. Program Behaviors Using State Diagrams And Class Sequence Diagrams

#### State charts

To view the robot's dashboard page, the user logs in. The user can make a new robot right here. The user can activate the new robot, which is initially inactive. Users can also create moments for these robots and resume an existing robot.

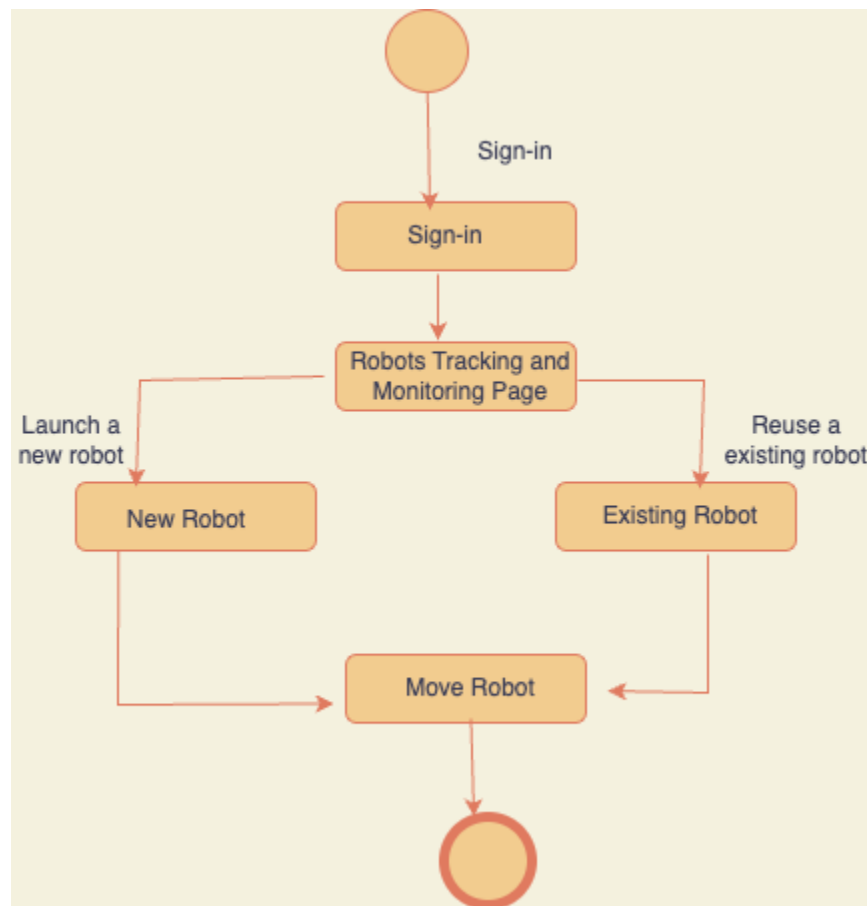


Fig.7 :State Diagram of Robot Tracking and Monitoring System.

#### Class Sequence Diagram:

The class sequence diagram shown below illustrates the different orderly steps that make up the system.

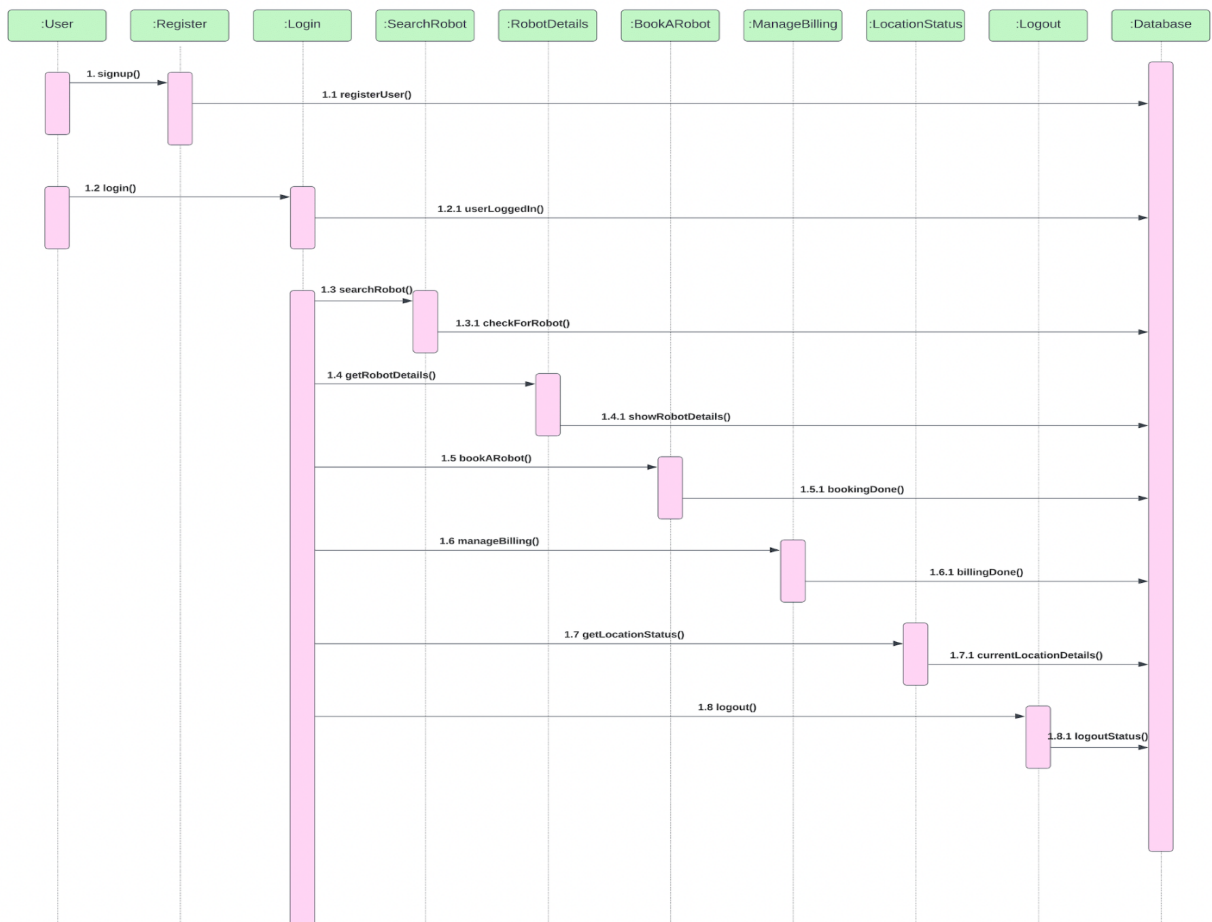
#### User Side:

The user completes the tasks listed below in the order given:

Before they may log in, users must first register for the online application. Following successful login, the following things happen in the prescribed order:



- The user searches for such a robot in a specific area.
- The user can then book the robot after he has examined its specifics.
- After making the money and making the robot booking, the internet portal will display the robot details.
- The user can verify the robot real-time location at any time thanks to the online application, which is constantly updated.
- The database keeps track of all the data and changes that take place at each step

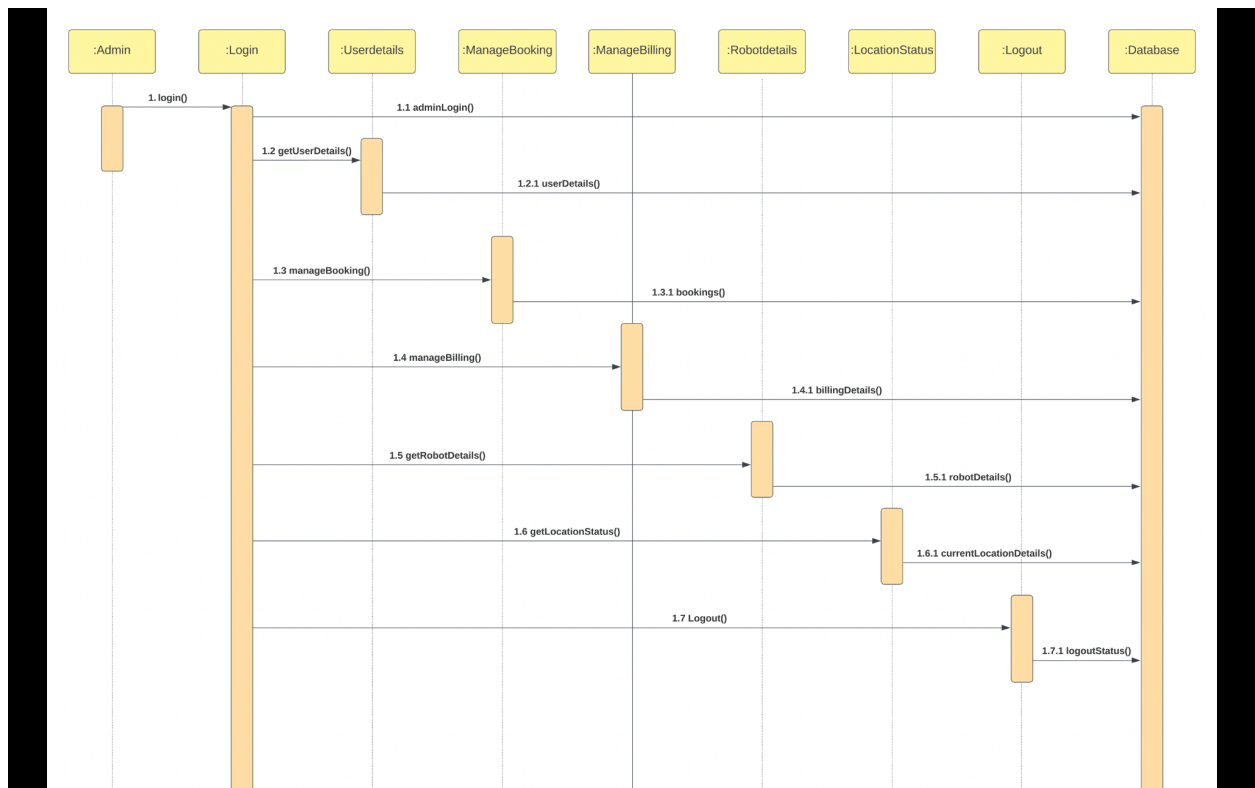


**Fig.8: Sequence diagram for robot tracking and monitoring for the user side.**

#### Admin Side :

- The system administrator logs in.
- After logging in, the admin can manage or view the following information.
- Information about the user will be visible to the administrator.
- He would check Manage Booking, which the user completes.
- Set up the prices for the robots to book orders.

- Adding and removing car details as necessary
- Observe where you are at the moment. Real-time updates are made to the location.
- The database records all the specifics and adjustments made throughout the process.
- The administrator exits.



**Fig.9 : Sequence diagram for robot tracking and monitoring for the admin side**

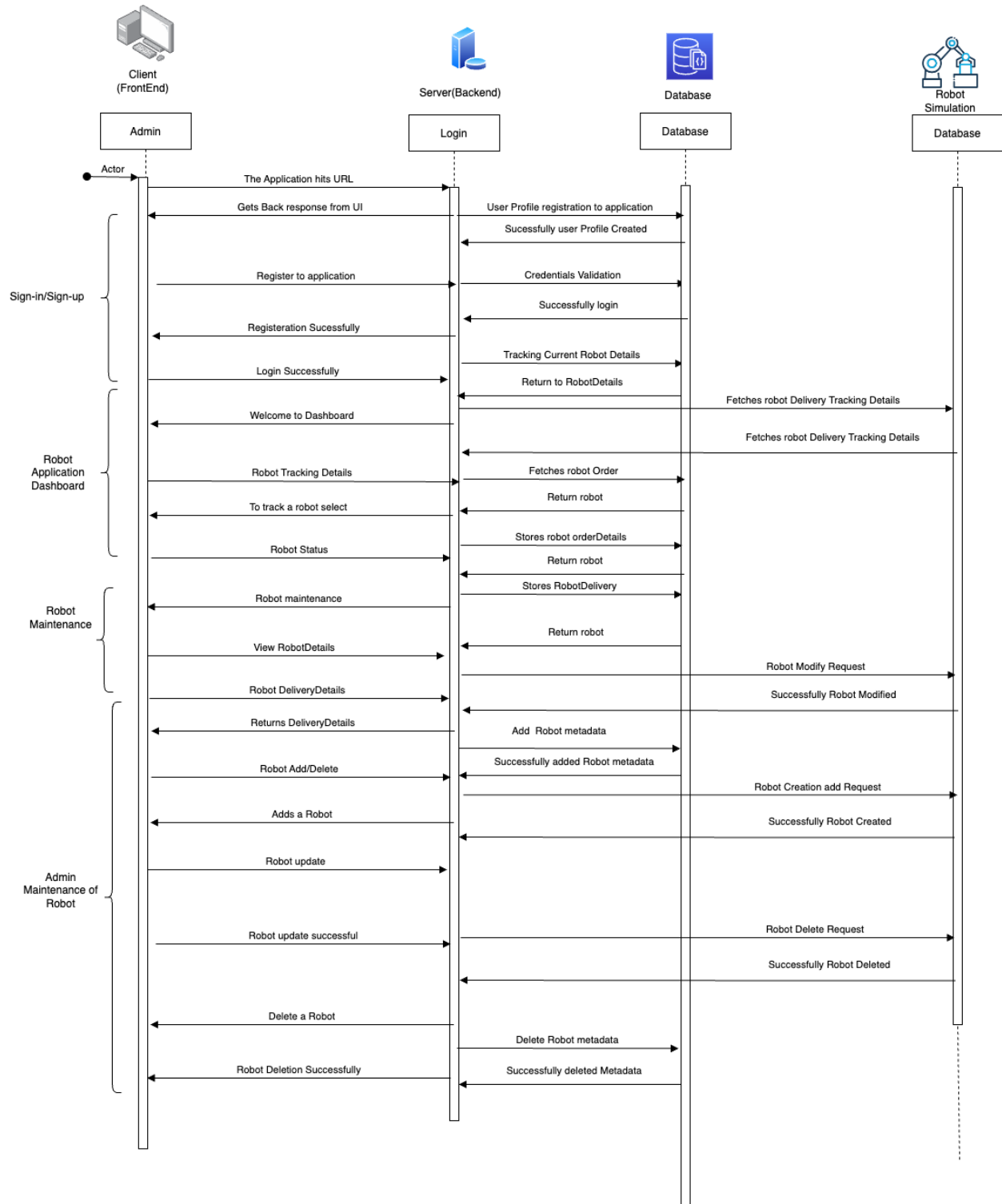


Fig.10 :Sequence diagram for robot for communication Design

## 4. Component Business Logics

### 4.1 Detailed business logics using decision tables

RESTful APIs are included in the business logic to manage a variety of interactions between the user interface and the databases. A count is kept in the SQL user table that automatically increases for that user when they construct a new robot. The user can see how many robots are being made at any given time thanks to this. The admin can also see every robot in the system by using this count.

When the user commands the robot to move, we retrieve the robot's current position using the robotId from the mongo robot collection and then adjust the robot's position in accordance with the user's instructions. This new position is added to the robot's path and changed in the mongo document for that robot's current position key so that it may be used when the user requests the path.

Simply use the robotId to filter the specific document from the mongo Robo collection and read the most recent coordinates from the robot path attribute to observe the robot position.

Given a robotId, the full set of coordinates is sent as a response that may be used to follow the robot's movement in order to obtain the traveled path of a certain user. The user must enter the robotId in order to activate or deactivate a robot. The appropriate action is taken on the robot with the help of this robotId, and the robot's status is modified. Additionally, if the status is stopped, the robot is deleted from MongoDB's robot collection, and the user relational database table's count is adjusted appropriately.

The robot's status can be changed so that when the user chooses to deactivate it, the robot ceases to be their property and its count is reduced in the user's RDBMS table. Additionally, the userId will no longer be included in the robot document in Mongo. On the other hand, when the robot is unplugged, it will be deleted from the mongo document and is no longer part of the system or accessible to any users.

## 5. Component graphic user interface design

### 5.1 Component GUI style

Using web UI wireframes, the graphical user interface style is demonstrated in section 5.3. These wireframes contain a preliminary illustration of the various user-accessible functions. The user can browse to different pages with the use of a persistent sidebar.

### 5.2 Storyboard

The stories for this component include the creation of various user interface (UI) elements that are available to the user, such as constructing a robot, moving a robot, and changing the robot status, so by using these elements we can track and monitor the robot for food delivery applications based on the pick up point throughout the trip duration time and delivery cost can be calculated. The status of the delivered order or not delivered can be tracked by the arrival time, and the delivery time for destination can be tracked and monitored by the admin.

### 5.3 Function Partition Tree:

Here, only the admin has the authority to activate an already-installed robot or to build a new robot according to the requirements of the business owner. This will link the specified company owner account id to the enabled robot id.

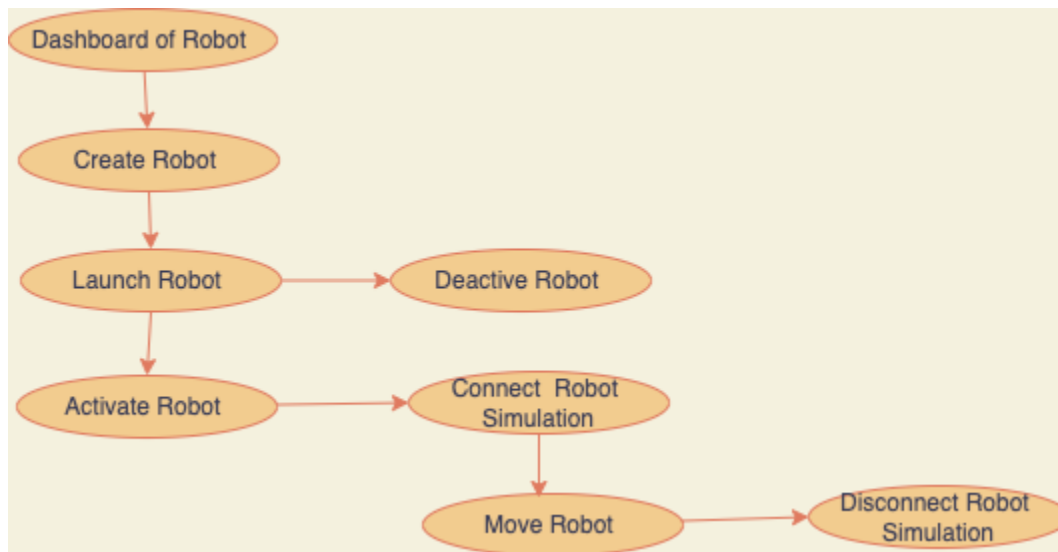


Fig.11:Function Partition Tree

The robot will process the order and deliver it to the customer in accordance with the customer's order request and the specific request received id.

In order to decrease the number of robots, a business owner should ask the admin to deactivate any robots that are assigned to accounts.

### **5.3 UI markup diagram and GUI templates**

The robot's graphical user interfaces are shown below.

The five various robot states are given below


- connected-The robot is connected to the application for simulation
- operated -The robot is operated by tracking and monitoring the simulation
- stopped - The robot is stopped by operating the application for simulation
- active -The active status will be displayed the robots are tracked how many robots are active.
- Idle -The idle status will be displayed the robots are tracked how many robots are idle.
- Inactive-The inactive status will be displayed the robots are tracked how many robots are inactive.

These are displayed together with the robot's status. The UI markup diagram and GUI templates are shown below in the figure.

### **GUI System Design**

The architecture of the system's UI frames and pages and the flowchart for navigating between them Our project requires the following features and functions: Dashboard, Scheduling, Billing, Simulation, MyRobots, and Navigation. The GUI System Design based on our project Robot Cloud is represented in the figure below.

**Robot Cloud Food Delivery**



USERNAME

PASSWORD

Sign-in

Sign-up

**Dashboard:** Our project application's flow specifications provide that if a robot is a new user to the robot application after registering as one, the application will prompt the robot to sign-in or sign up.

Dashboard

Scheduling

Billing

Simulation

My Robots

Navigation

**Register Robot**

Robot Name	<input type="text" value="TestRobot"/>		
Robot Type	<input type="text" value="Test1"/>	Version	<input type="text" value="11.0"/>
Manufacture Name	<input type="text" value="CloudRobot"/>	X location	<input type="text" value="10.5"/>
Operating System	<input type="text" value="mac"/>	Y location	<input type="text" value="20.4"/>

Submit

**Simulation:** After scheduling, select Simulation to see if the robot is actually operating as intended based on movement and navigation simulations.

Dashboard

Scheduling

Billing

Simulation

My Robots

Navigation

Dashboard of Robot Simulation

Start Simulation of Robot

Select Robot

Dropdown button

Movement Simulation

Start Simulation

Navigate Simulation

Start Simulation

Submit

**Robot Billing:** Using the robot ID, we can determine whether a robot is present and, if so, whether it is being used to deliver food to users. From there, we can add a charge for robot utilization and keep track of the robot's condition. It will bill based on how long the robot is used for, as in seconds, minutes, hours, or days.

Dashboard

Scheduling

Billing

Simulation

My Robots

Navigation

Dashboard of Robot Billing

[Dashboard](#) > [Robot Management](#) > [Billing](#)

Robot ID

1

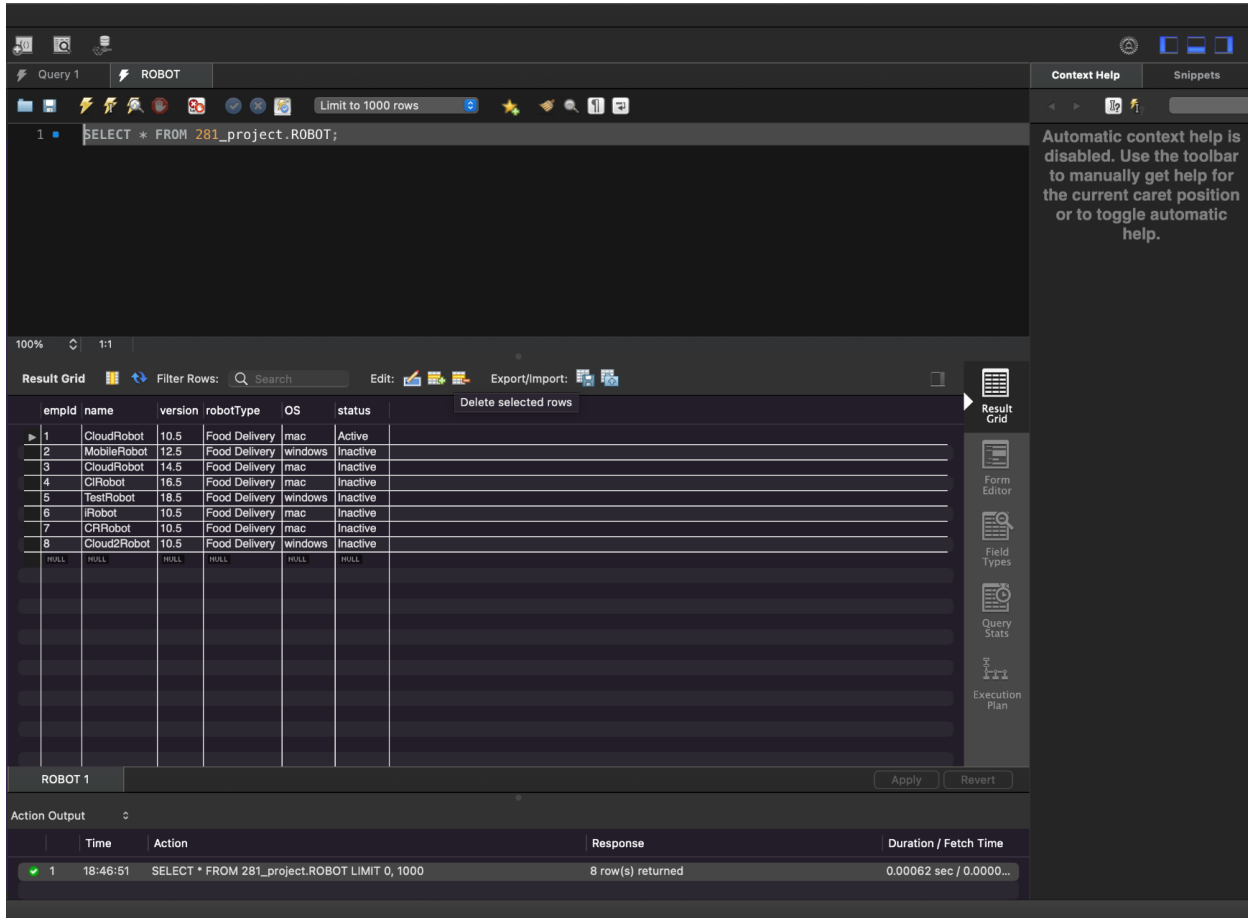
Activate/Deactivate Robot

Connect/Disconnect Robot

Robot Billing Charges Tacking  
usage of Robot



Here you can able to see the active robots for the application and in the backend database side also it will be automatically updated the status of the robot based on the ID with respect to activate or deactivate the robot or else should connect or disconnect the robot the submit the robot billing charges tracking and controlling the robot.



The screenshot displays a database management tool interface. At the top, a query editor shows the SQL statement: `SELECT * FROM 281_project.ROBOT;`. Below the query editor, a table titled "Result Grid" displays the data. The table has columns: `empId`, `name`, `version`, `robotType`, `OS`, and `status`. The data is as follows:

empId	name	version	robotType	OS	status
1	CloudRobot	10.5	Food Delivery	mac	Active
2	MobileRobot	12.5	Food Delivery	windows	Inactive
3	CloudRobot	14.5	Food Delivery	mac	Inactive
4	CloudRobot	16.5	Food Delivery	mac	Inactive
5	TestRobot	18.5	Food Delivery	windows	Inactive
6	IRobot	10.5	Food Delivery	mac	Inactive
7	CRRobot	10.5	Food Delivery	mac	Inactive
8	Cloud2Robot	10.5	Food Delivery	windows	Inactive

Below the table, the "Action Output" section shows the execution details of the query:

	Time	Action	Response	Duration / Fetch Time
1	18:46:51	SELECT * FROM 281_project.ROBOT LIMIT 0, 1000	8 row(s) returned	0.00062 sec / 0.0000...

Robot tracking and monitoring its status to determine which one is active and which one is inactive. Here you can see the status of robots in the database, which are active and inactive.

**Navigation:** After the simulation is complete, it will use the navigator's assistance to go in the direction on the three separate axes to display the visualization of the robot in the simulation.

Dashboard

Scheduling

Billing

Simulation

My Robots

Navigation

Dashboard of Robot Navigation

Start Navigation of Robot

↑

← · →

↓

Location

X10.5

Y20.4

Z5.8

HeadingLeft

Start to Navigate based on Goal

Submit

After the billing, it will take us to the dashboard of the navigation of the robot. Based on the movement of the robot, we can track the location of the robot based on the coordinates of x -10.5 and y- 20.4,z-5.8 and heading to the left. In a similar way, we can identify the position of the robot to which direction it is heading to deliver the food delivery order.