
**CMPE 281-01: Cloud Technologies.**

**Deliverable 2: Component Design and Analysis Document**
**Component #1 – Edge-based mobile robot (Simulator)**
Option #1: Robot Cloud
Robot Type: Food Delivery Robot
**Submitted to:**
Dr. Jerry Gao

**Date of Submission 10/17/2022**
Submitted by Group 24



| S.No | Name | Student ID | Email ID |
|---|---|---|---|
| 1. | Farazuddin Mohammad | 016176836 | farazuddin.mohammad@sjsu.edu |

## Table of Contents

# Section 1: Edge Based Food Delivery Robot Overview

## 1.1 Purpose

The food service industry has not seen much growth in the usage of industrial robots as the industry is mostly on the small or medium scale of the spectrum. With the current rate of development of sophisticated robot technology, usage of robotics in restruant is becoming a lucrative option. In the near future there will be a need for software that can help control these robots remotely.

To manage these robots we propose to build a cloud based Software as a Service (SaaS) platform to provision, deprovision, monitor and charge the consumers based on a billing model. The platform will be available over the public internet and will be used to perform the previous mentioned operations.

Edge Based Robot Simulator for food delivery robots is used to simulate the end to end process of a customer i.e restaurant owner using our dashboard available over the internet to register, set up payment methods and start deploying food delivery robots. By connecting to backend elements like the database and cloud, users and owners may keep track of their usage, location, billing, and previous bookings. It consists of storage and hardware, both of which are housed on a distant server.

## 1.2 Objective

The Edge Based Robot Simulator Component will help with the simulation of food delivery robots and will help us visualize the real time functioning of the dashboard and robots.
The function of the component is to give users and owners an intuitive user interface that enables them to rapidly request and monitor robot delivery services. In order to display tracking and other information for users and owners in the user interface, this component also attempts to send and receive data from backend databases.

## 1.3 Function Scope

We plan to control the robots via the dashboard i.e the dashboard will provide an interface to receive and transfer the instruction to a third party simulator software that will render the simulation.
AWS robomaker is our selected third party simulator, it is AWS's cloud-based simulation service that can enable robotics developers to automate, run and scale simulations without managing any infrastructure.

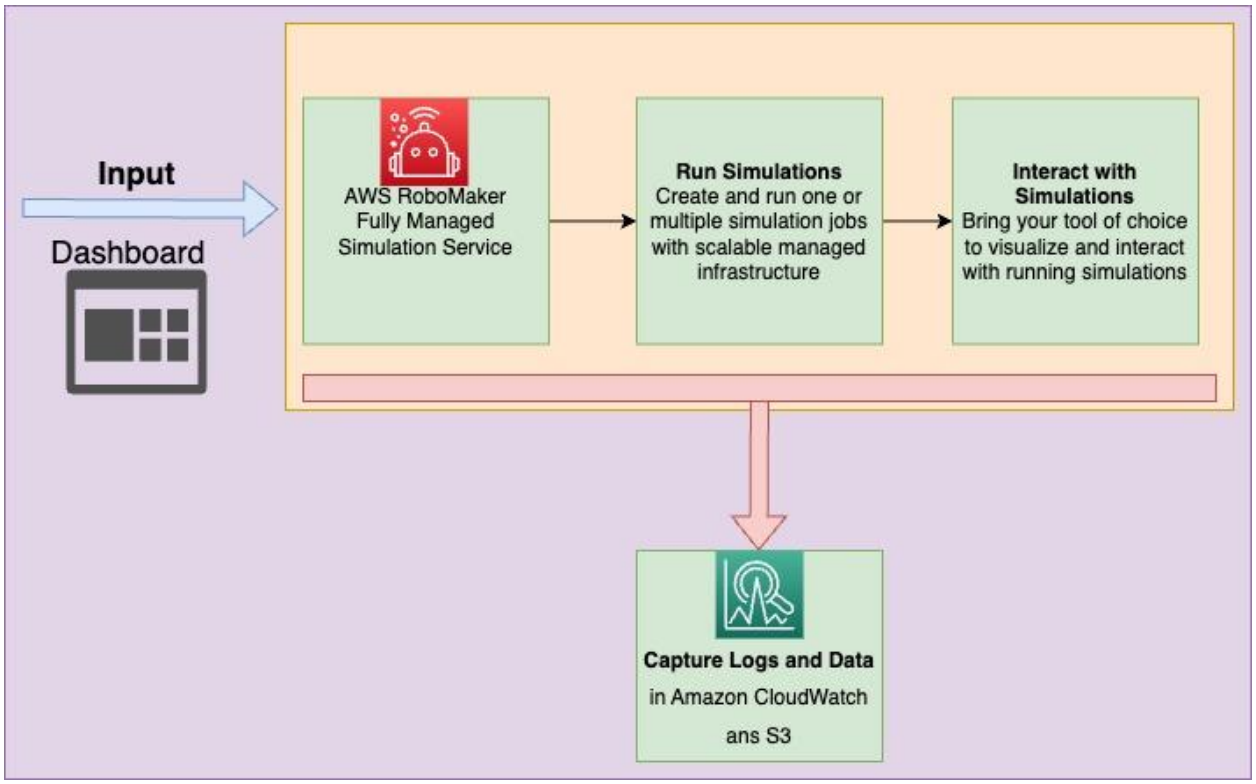



**Fig 1. AWS Robomaker Overview**

## 1.4 Usage

The Food Service Industry will soon adapt to using robotics to automate tasks in this way the service time will be optimized and the restaurant owners will be able to save labor costs as well. We plan to build a dashboard to deploy food delivery robots in closed spaces like restaurants where the robots will have food drop off locations and pickup locations. First the restaurant owner will have to register on

our dashboard, set up account and billing information and then he/she can start deploying robots and will be charged on usage basis.

## 2. Edge Based Mobile Food Delivery Robot Application Interface, Design and Analysis

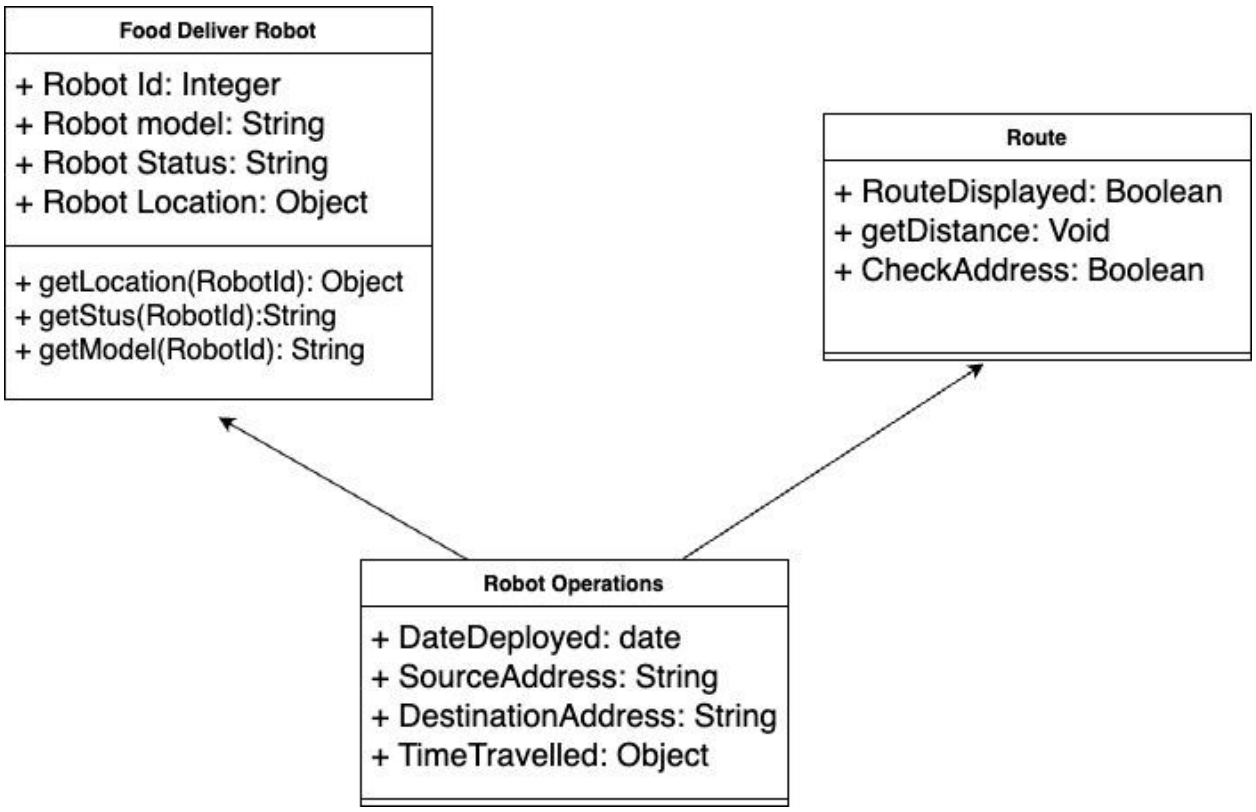
### 2.1 Interface Rest APIs Details

The table below describes the APIs being called while setting up, running and modifying the simulation.

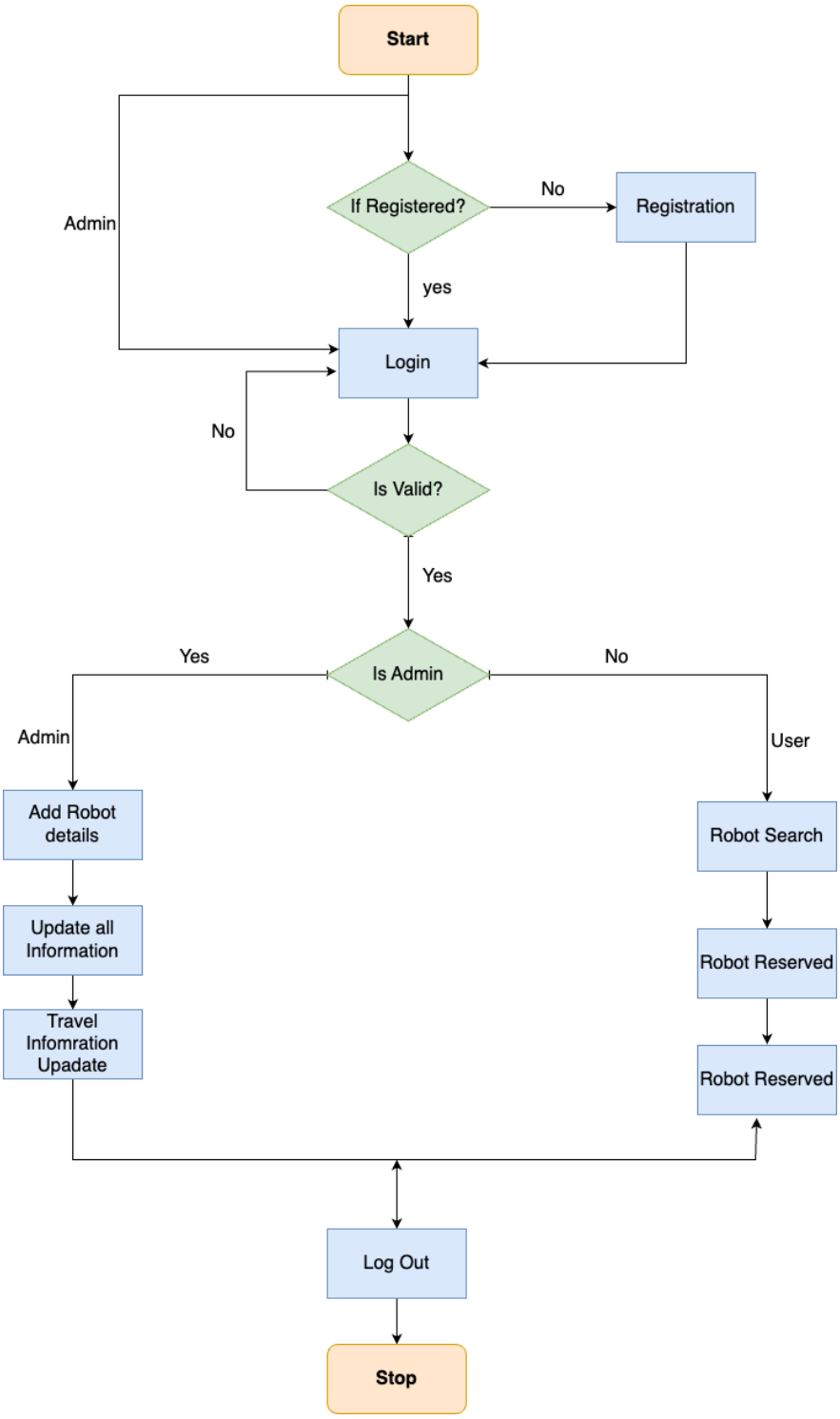
| API Method | URL | Parameters | Response Type |
|---|---|---|---|
| POST | createSimulationJob | No Parameters | None |
| POST | createRobot | No Parameters | None |
| POST | createRobotApplication | No Parameters | None |
| POST | MoveDirection | Int:direction | None |
| SET | SetRobotLocation | Int: x, Int: y | None |
| SET | SetDropOffLocation | Int: x, Int: y | None |
| SET | SetPickupLocation | Int: x, Int: y | None |
| GET | RobotStatus | Int: RobotID | Integer |
| GET | RobotLocation | Int: RobotID | JSON |

## 3. Edge Based Mobile Food Delivery Robot Function Design, Data and DB Design and Behavior Analysis

3.1-Class Diagram for Users, Robot State tracking and back-end communication



## 3.2 Component Flow Diagram



## 3.3 Sequence Diagram:

The sequence diagram below shows the event chronology from the administrative perspective. The process goes like this.:

**User Side:**

The user completes the following tasks in the prescribed order:

Users must first register for the online application in order to log in. After a successful login, the following events take place in the following 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.

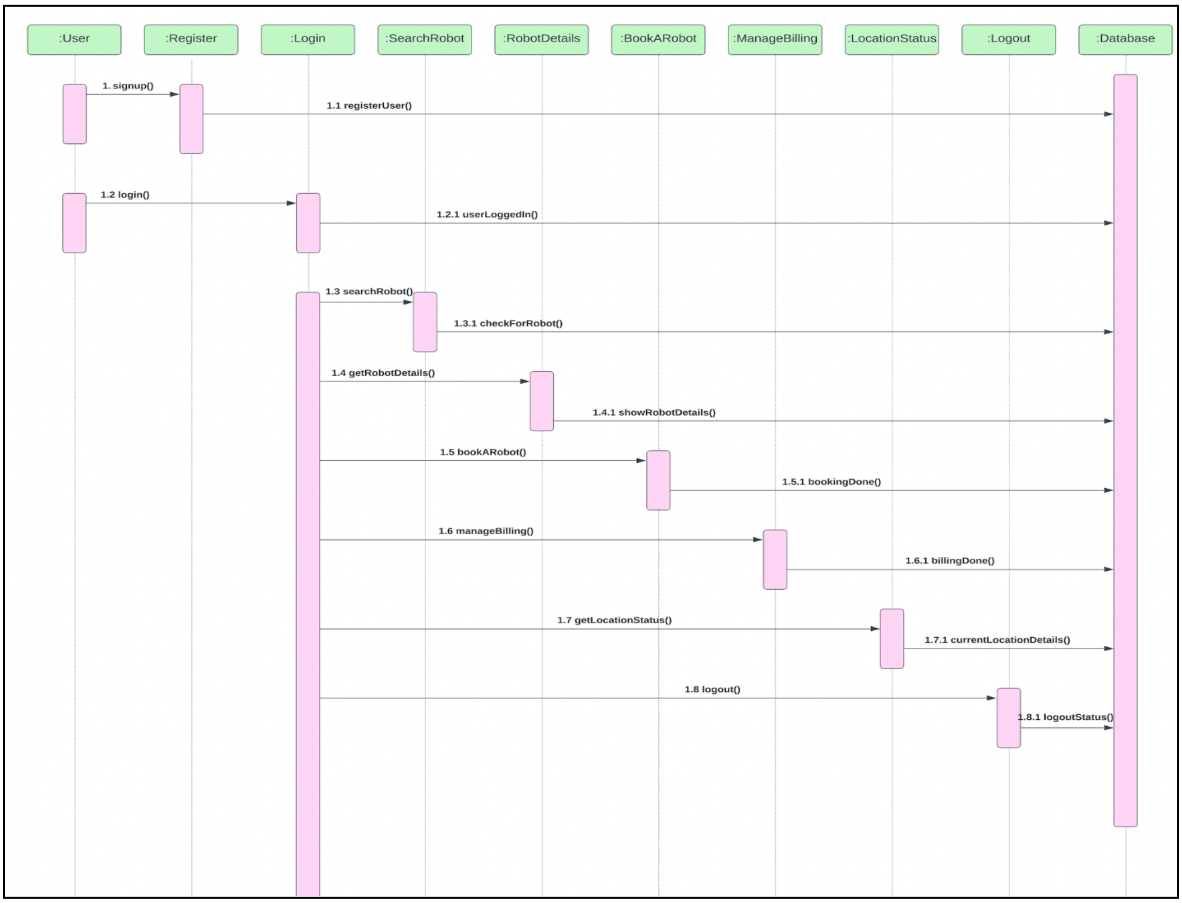



**Figure 6 : User Side Sequence Diagram**

**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.

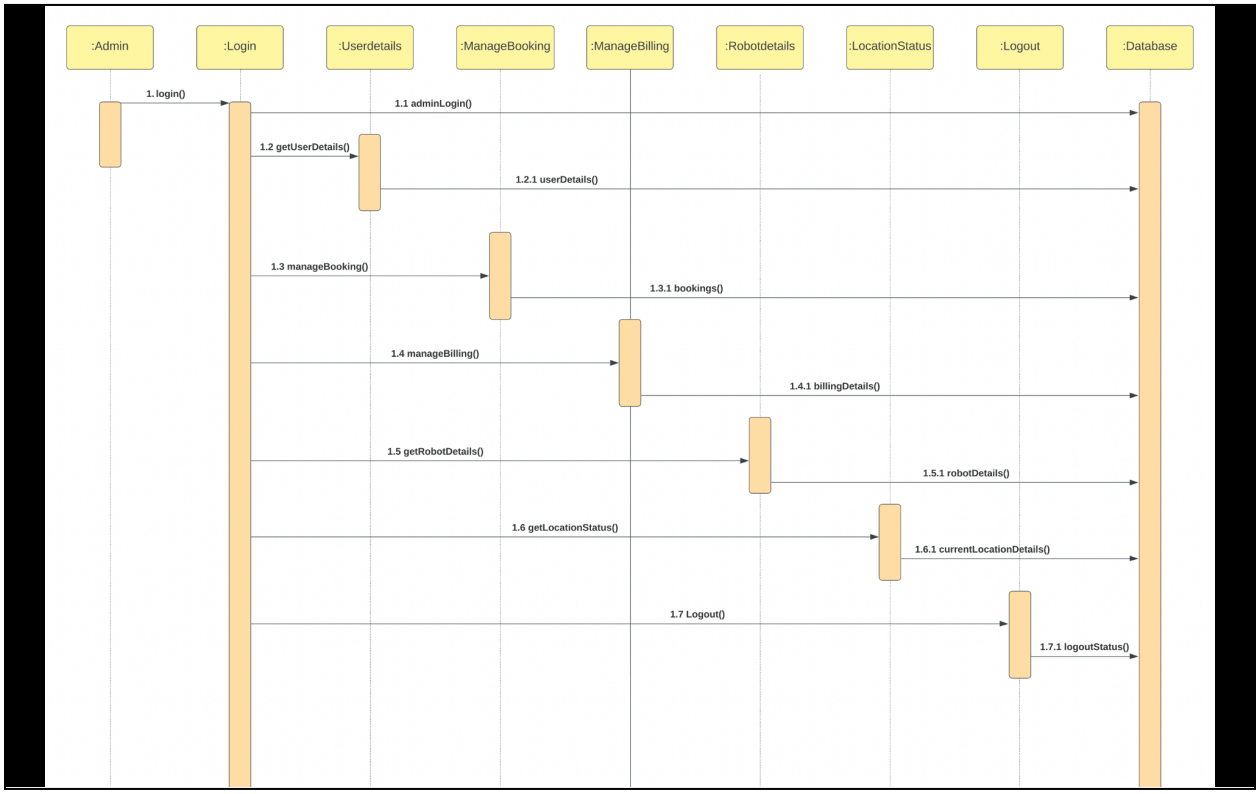
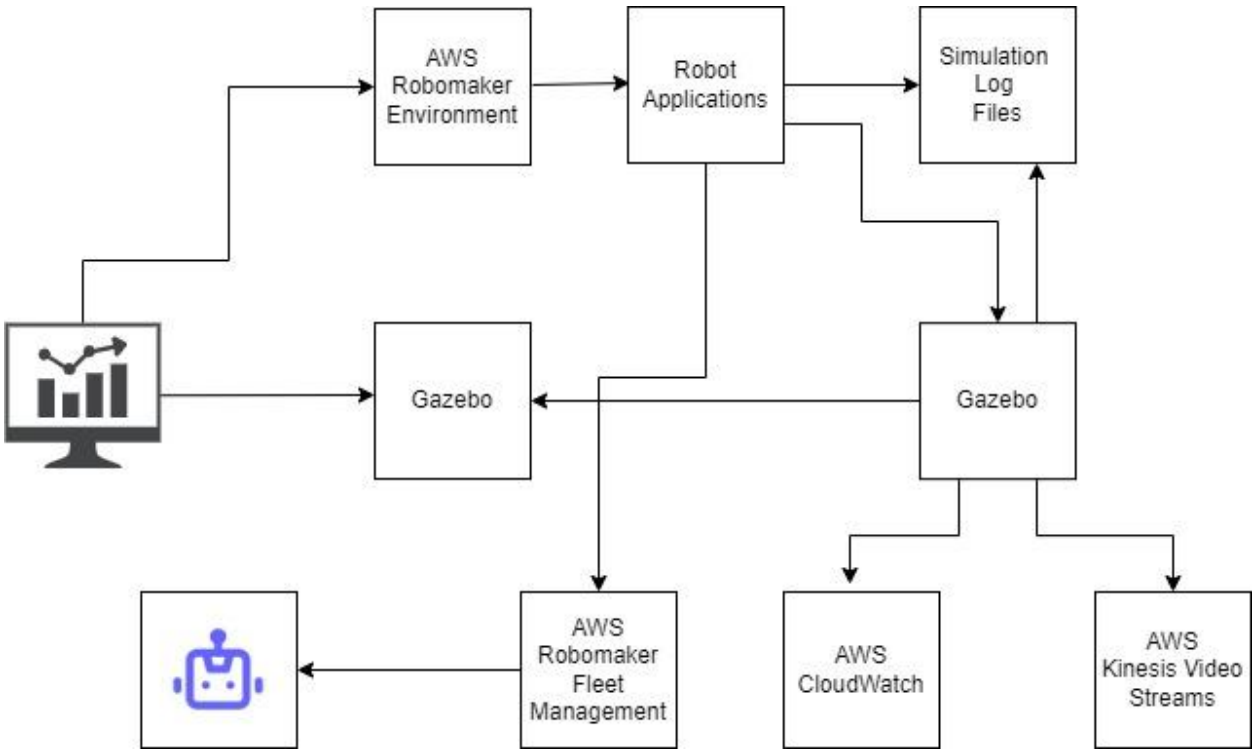


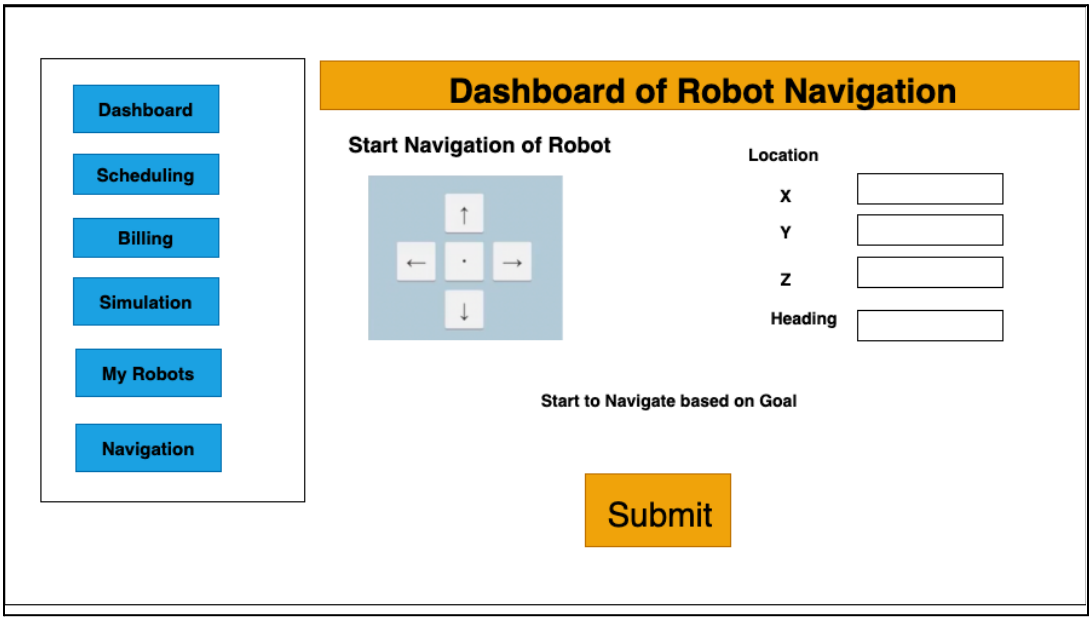

**Figure 7 : Admin Side Sequence Diagram**

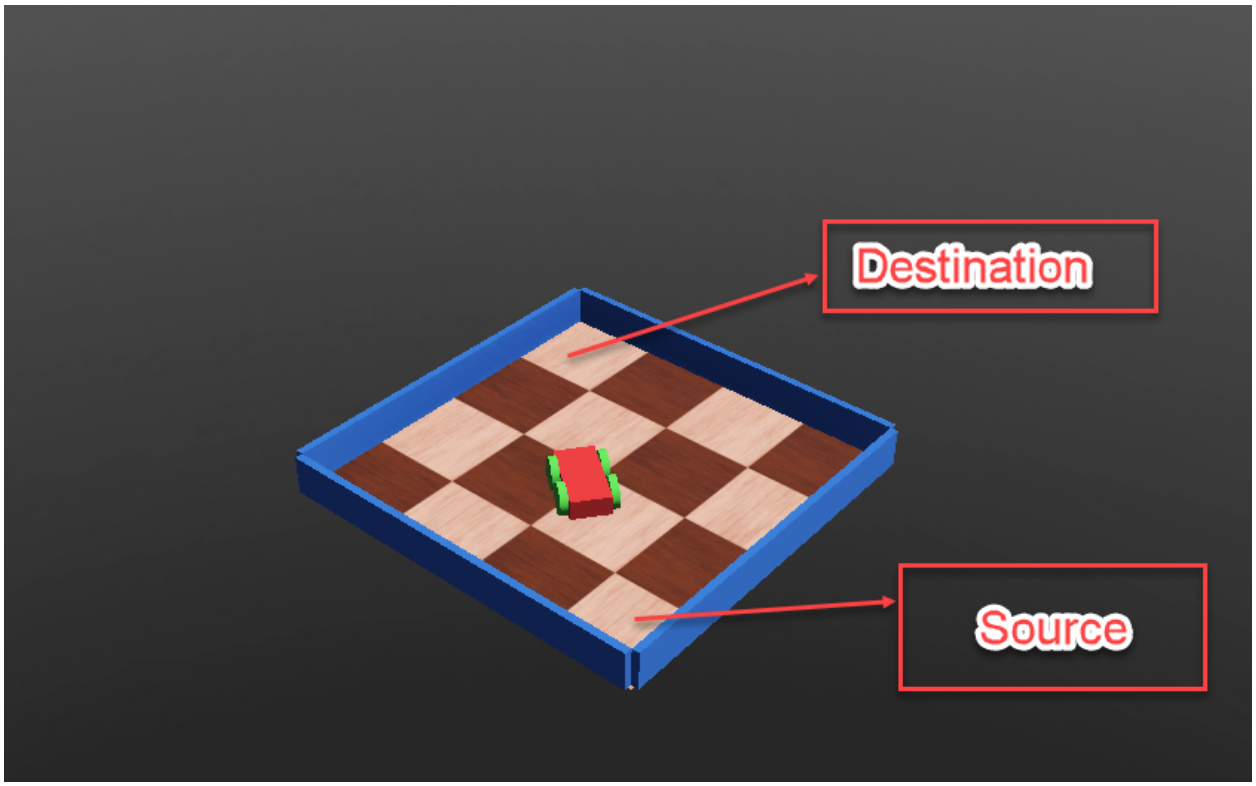
# 4. Edge Based Mobile Food Delivery Robot Business Logic

## 4.1 AWS Robomaker Data Design Diagram



# 5. Component Graphic User Interface Design



**Pseudo Code For Robot**

```
while (wb_robot_step(TIME_STEP) != -1) {
  // init speeds
  double left_speed = 1.0;
  double right_speed = 1.0;

  if (avoid_obstacle_counter > 0) {
    avoid_obstacle_counter--;
    left_speed = 1.0;
    right_speed = -1.0;
  } else {
    // read sensors outputs
    double ds_values[2];
    for (i = 0; i < 2; i++)
      ds_values[i] = wb_distance_sensor_get_value(ds[i]);

    // increase counter in case of obstacle
    if (ds_values[0] < 950.0 || ds_values[1] < 950.0)
      avoid_obstacle_counter = 100;
  }

  // write actuators inputs
  wb_motor_set_velocity(wheels[0], left_speed);
  wb_motor_set_velocity(wheels[1], right_speed);
  wb_motor_set_velocity(wheels[2], left_speed);
  wb_motor_set_velocity(wheels[3], right_speed);
}
```