CMPE 281-01: Cloud Technologies.

Deliverable 1: System Design and Architecture Document

Option #1: Robot Cloud (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.	Fnu Butul Parveen	015918227	butul.parveen@sjsu.edu
2.	Kanupriya Sanjay Agrawal	016099057	kanupriyasanjay.agrawal@sjsu.edu
3.	Farazuddin Mohammad	016176836	farazuddin.mohammad@sjsu.edu
4.	Chirag Arora	016726567	chirag.arora01@sjsu.edu

Table of Contents

1. INTRODUCTION

The food delivery industry has invaded our lives because of the Internet's quick expansion, giving consumers new dining options and a lot of convenience. It is challenging to ensure the quality of take-out food using the popular takeout applications now available. Additionally, manual delivery is currently the preferred method of delivery for the takeaway market. You frequently see people riding electric bikes in a hurry because the delivery platform's delivery time is so demanding. For getaways in a hurry, speeding and retrograding are particularly typical. This is quite risky on campuses with heavy traffic. Speeding is a major hidden risk for students since it frequently results in collisions with delivery people.

The primary source of food for college students is the canteen. College canteens are mostly confronted with college students who are pursuing freshness due to the increasing meal delivery sector today. They also need to strengthen their own sales model, move toward a new model, and offer students more convenient and effective services. They also need to keep up with the times.

At the same time, significant advancements have been made in the field of intelligent robot technology, and the application scenarios for robots have moved from the industrial to the service sectors. As soon as you enter the cafeteria, meals can be picked up. This approach eliminates the difficulty of queuing in college canteens and will improve, streamline, and humanize dining options. However, some college students feel that it is easier for takeaways to deliver meals downstairs in the dormitory due to the big campus and the living area being far from the dorm, so they opt to choose takeaway meals. This study develops a campus takeaway robot system in response to this issue, to make it more convenient for college students and encourage them to eat at the school canteen. To make it easier for students to eat takeout from the canteen, the system substitutes takeout robots for people.

A. Purpose

To build a cloud-based autonomous food delivery robot service system using the Webots simulator. Mobile robotics is now a powerful technology, but in order for these robots to properly handle, regulate, and manage the resources in this cloud era, cloud services must be enabled. Cloud platforms and services make it simple to provide many services, including remote configurations, user account management, user service management, and billing services.

B. Objective

To design a cloud-based autonomous food delivery robot service system using the Webots simulator, offering the best possible user interface and interactions. It will leverage a web development framework to deploy the model in real time. We build a system using the latest

architecture called Elastic Beanstalk. The goal of developing the cloud-based application is to give users and system administrators access to cloud platform features so that they may interact with the mobile robot remotely. The given cloud environment should allow users to access their cloud accounts and manage the mobile robots. They ought to be able to see how many robots are associated with their accounts, manage those robots, see how they're doing, get billing details, and deactivate any that aren't being used.

C. Market Analysis and Expected outcome

The delivery robot market is expected to grow from USD 212 million in 2021 to USD 957 million by 2026, at a CAGR of 35.1% from 2021 to 2026. Reduction in delivery costs in last mile deliveries and increase in venture funding are the key factors driving the growth of the delivery robot's market. Further, the worldwide growth of the e-commerce market is one factor propelling the growth of the delivery robot market.

According to ABI Research forecasts, the market for mobile robotics with cloud infrastructure, also known as robotics as a service, is expected to rise significantly. According to estimates, the robotics sector will rise by a total of 30%, from 3 billion US dollars in 2019 to 160 billion US dollars by the end of 2033. Three different types of sectors will primarily use cloud-based robotic services: robotic solution suppliers, cloud platform providers, third-party IoT service providers, and cloud service providers like Amazon, Google, and Microsoft.

1.2 Structure of the Project Report

The project report for CMPE 281, Cloud Technologies, is the final document in the series. This encompasses the following areas: associated work, cloud-based infrastructure, then the design and execution of the simulation. Then we discuss subjects like data design and system design. The design of Edge stations, the implementation of the GUI, and examples of system applications follow. We go on to the subjects of performance evaluation and conclusion after these chapters.

2. RELATED WORK

Webots is an open-source robotics framework for the cloud that allows robots to communicate and share knowledge via a www-style database, giving them access to robust cloud robot services. Through its page for software components, Webots made the documentation and source code available. Because of the RosJava library, ROS may now be run on Android phones. A robot can connect to a single ROS environment using Rosbridge, a cloud-based communication technology. The Webots Project illustrated the advantages of cloud computing by parallelizing a SLAM technique using a Hadoop cluster.

3. Cloud-Based System Infrastructure and Components

Utilizing AWS cloud resources, our food delivery robot cloud system is hosted. Our system's cloud-based components, such Webots, the user dashboard, and the simulator, are hosted there and make use of several AWS services. Our cloud-based architecture and components are supported by resources such as virtual machines, load balancers, virtual private clouds, and database instances. In order to deploy our application as a cloud-based, multi-tenant system, connectivity between all resources was essential. the building blocks for what might eventually be a viable autonomous robot rental product and company.

A. Components

1) EC2: Elastic Computing Cloud: The instances are simulated computing environments. Our cloud-based programs, including Webots, the user dashboard, the simulator, and MongoDB are hosted by virtual machines. With the aid of launch templates, it was possible to specify the kind of EC2 instances that should be created as well as the set of commands that should be run in order to launch each respective application or instance.

2) Auto Scaling Groups: By using limit thresholds as a guide, auto scaling groups help generate new EC2 instances. The auto scaling group will launch a new instance based on a predetermined launch template once a maximum threshold determined by a dynamic scaling policy is met.

3) VPC: Virtual Private Cloud: Our AWS resources are encased in a private network within AWS. builds a private network to house the AWS resources needed by our system and application. A route table, matching subnets, and an internet gateway make up a VPC. There are related subnets and a route table in our VPC. Within our region, each subnet corresponds to a separate availability zone and has a set of IPv4 CIDR addresses. When EC2 instances are launched inside the corresponding availability zone, these subnets assist in allocating private IP addresses.

4) RDS: Relational Database Service: A relational database can be easily installed, run, and deployed in the cloud thanks to Amazon RDS (Amazon Relational Database Service). The user, robot,delivery, and billing data will be stored using the "MySQL" engine. We used an RDS MySQL instance to store our relational data.

5) ELB: Elastic Load Balancing: Requests were redirected to EC2 groups and instances that could scale automatically using application and classic load balancing. To efficiently manage incoming traffic, load balancing helps divide requests among several EC2 instances. In order to

control load, a load balancer is connected to an auto scaling group and directs traffic to the group's instances.

B. Architecture

Our architecture encapsulates all of our resources into a VPC. In order to reflect the various availability zones within the VPC, various subnets with unique IPv4 CIDR addresses were constructed. The route table inside the VPC was connected to these subnets. A private IP is assigned depending on one of the availability zones when resources are created. On EC2 instances set up using a launch template, our sensor APIs and dashboard application are hosted. These instances are created by spawning them with the launch-template definition and wrapping them in an auto-scaling group. According to the auto-scaling policy, a new instance is created if one of the instances' CPU use exceeds a predetermined threshold. To uniformly distribute requests among several instances, load balancers are connected to the auto-scaling groups. Additionally, our Webots simulator is load-balanced and hosted on EC2 instances. The Webots simulation will run on that instance when the load balancer directs requests to the appropriate EC2. Our MongoDB collections were also hosted on an EC2 machine. Using inbound rules, data storage and retrieval from MongoDB were made possible.

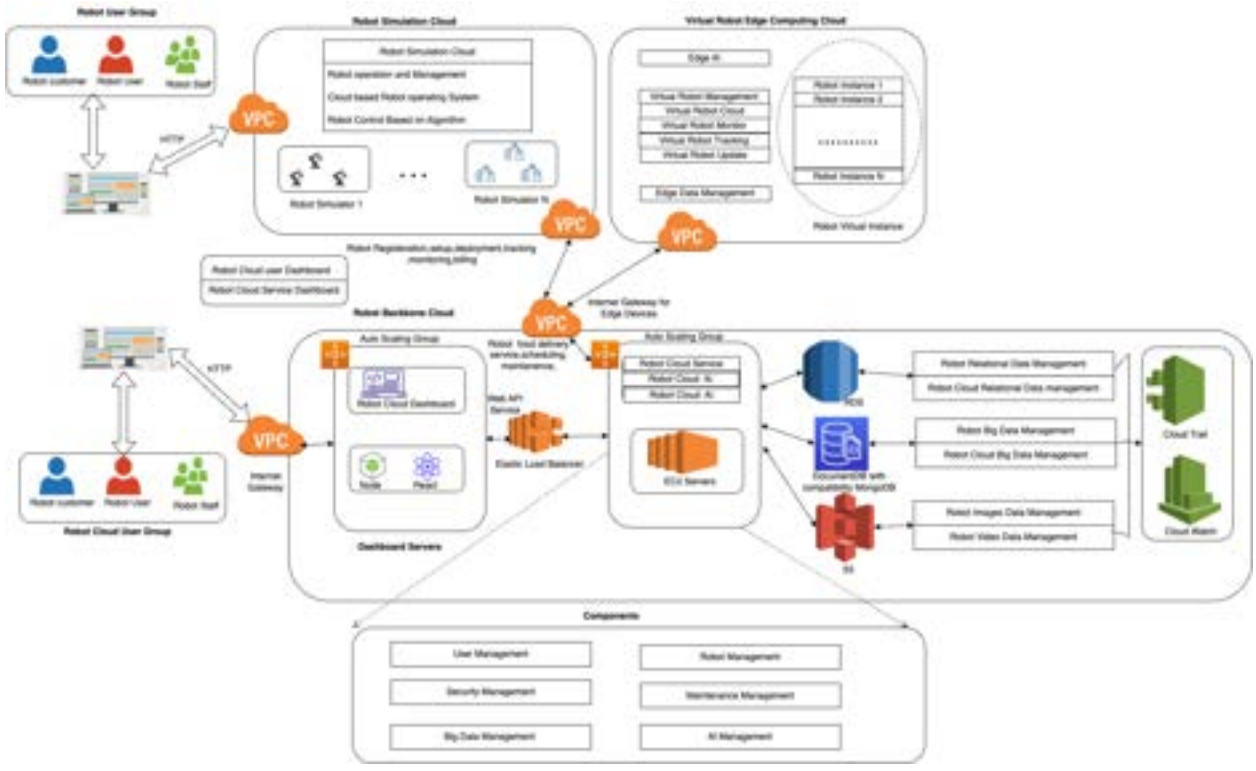

Figure 1: Cloud-based System Infrastructure for Robots

For the user and admin roles, we've chosen a React-based frontend. We created an express server based on Node JS for the backend to interface with the robot and database. The entire program is

running on the AWS cloud. For scaling and load balancing, we took advantage of AWS ELB. Backend servers scale based on load using an auto scaling cluster. Further information on the subject can be found in the pertinent chapters' parts.

3.2 System component-oriented function architecture design

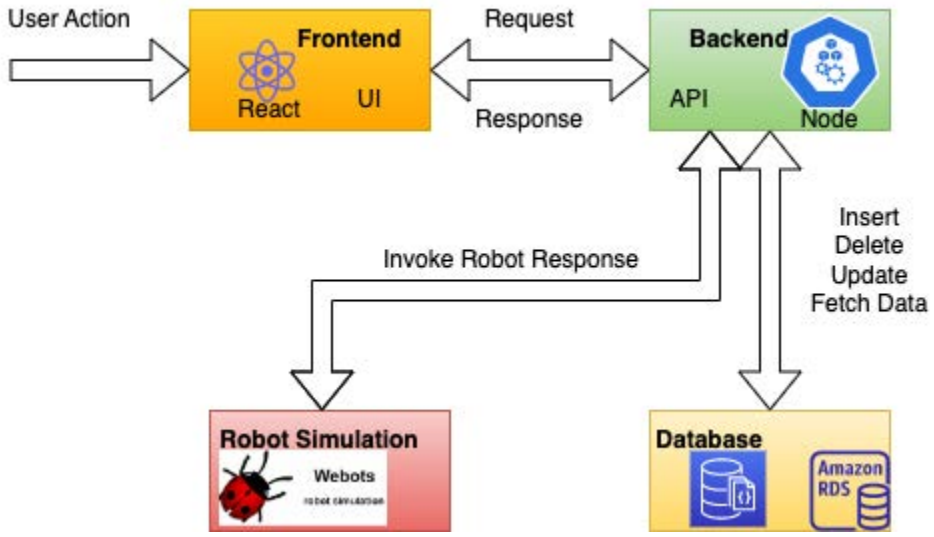

Figure 2 :System component-oriented function architecture design

ReactJS:

It is an open-source, free JavaScript library that offers straightforward, quick, and adaptable frontends for online applications. A virtual DOM program and server-side delivery are two key components of the React system that enable even the most complex apps to run quickly.

NodeJS:

It is an operating cross-stage server that is free and open-source and used to manage spikes in JavaScript demand. With NodeJS, engineers may run several portable programs and capacities simultaneously without the servers crashing or slowing down. The environment's non-impeding, input-yield activities make it the most accessible choice. Faster code execution benefits the overall server operating environment.

AWS:

The cloud framework platform provided by AWS is simple, flexible, and incredibly reliable. With its unlimited server capacity, AWS can handle any load. It uses auto-scaling to build a framework for self-monitoring that adapts to the real needs depending on the vehicle being utilized. As a result, developing an application on AWS will enable us to operate more effectively and without interruption.

Service Manager:
We will employ the Python and C++ libraries provided by the robot community to regulate the simulation of data. To mimic a real-world robot issue, utilize the Robo Control Manager.

RobotSimulator:
It is a simulator for single robots and multi robots. It was made specifically to aid in the creation and instruction of automated robots. It is free to use because the source code is open-source. It has a strong API that enables users to control every aspect of the simulation, including robot creation, and the robot's behavior.

3.3 System deployment infrastructure

The above system deployment diagram is used to visualize the hardware, i.e.nodes/devices nodes and devices, the communication links between them, and the placement of software files, i.e., artifacts, between them.Since ours is a cloud based application, only the client system and the dashboard admin, which are individual computers with a web browser installed, are used to access the robot management dashboard outside of the cloud environment. The communication between the cloud's virtual hardware and the client happens via the HTTP protocol, which sits on the application layer that functions over TCP/IP, and the requests are made using REST APIs.

UML Deployment Diagram

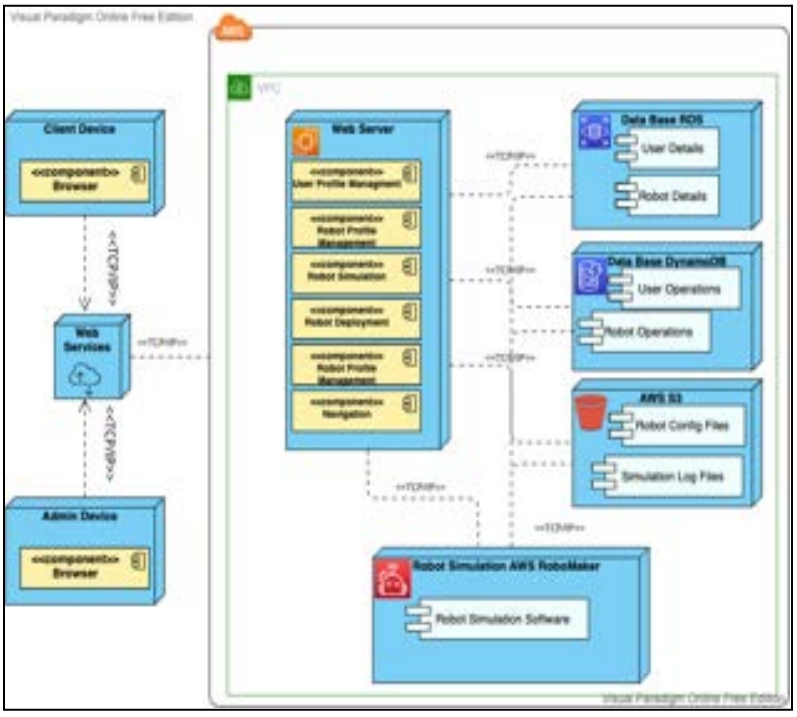

Figure 3 : System Deployment Infrastructure

The system was set up on the AWS cloud. The backend servers and Web application were deployed using EC2 instances. Webots simulation has been utilized with a Webots client that is continuously subscribed to the live robot in order to integrate and communicate with it.

4. Simulator Design and Implementation

The food delivery Robots that are scheduled in the Robot Cloud web service are simulated using Webots. Without it, the frontend would be a mere empty shell with no engaging client interaction because it acts as the primary backend component of the Robot Cloud project.

This component makes use of the open-source robot simulator Webots, which offers a highly flexible and interactive environment for simulating autonomous robots using Python scripts. The user can choose from a list of available vehicles and enter starting and ending locations through the frontend web graphical user interface. The Flask application running in front of the Webots simulator application receives a POST request once the user clicks the button to start a robot.

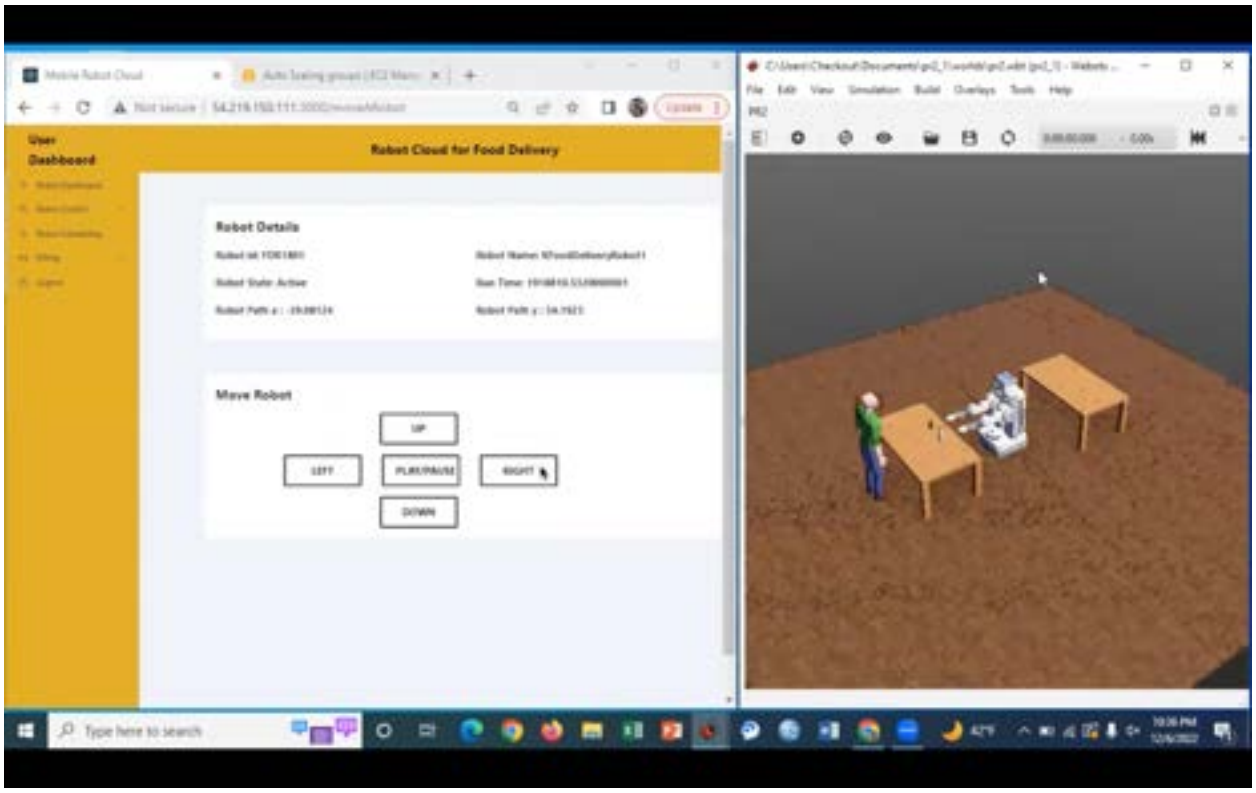

Figure 4

The frontend web GUI sends POST requests to the application, which is a REST API endpoint hosted on EC2. Included in the POST data are the robot type, robot ID, schedule ID, starting location, and finishing location. The Webots client script is executed by the Flask application, and the POST request arguments are passed along. During its 30-second running time, the

simulation gathers a variety of sensor data. The simulation updates the NoSQL database API with data collected by the GNSS, IMU, and LIDAR sensors once every second. Every time an event occurs, such as a collision, an obstruction, or a lane invasion, event-based sensor data is also transmitted to the NoSQL database. Using a Classical Load Balancer, the Webots simulators are load balanced. Since all traffic is handled equally, we do not require an application load balancer. The load balancer merely directs traffic to an idle EC2 instance that is hosting WeBots.

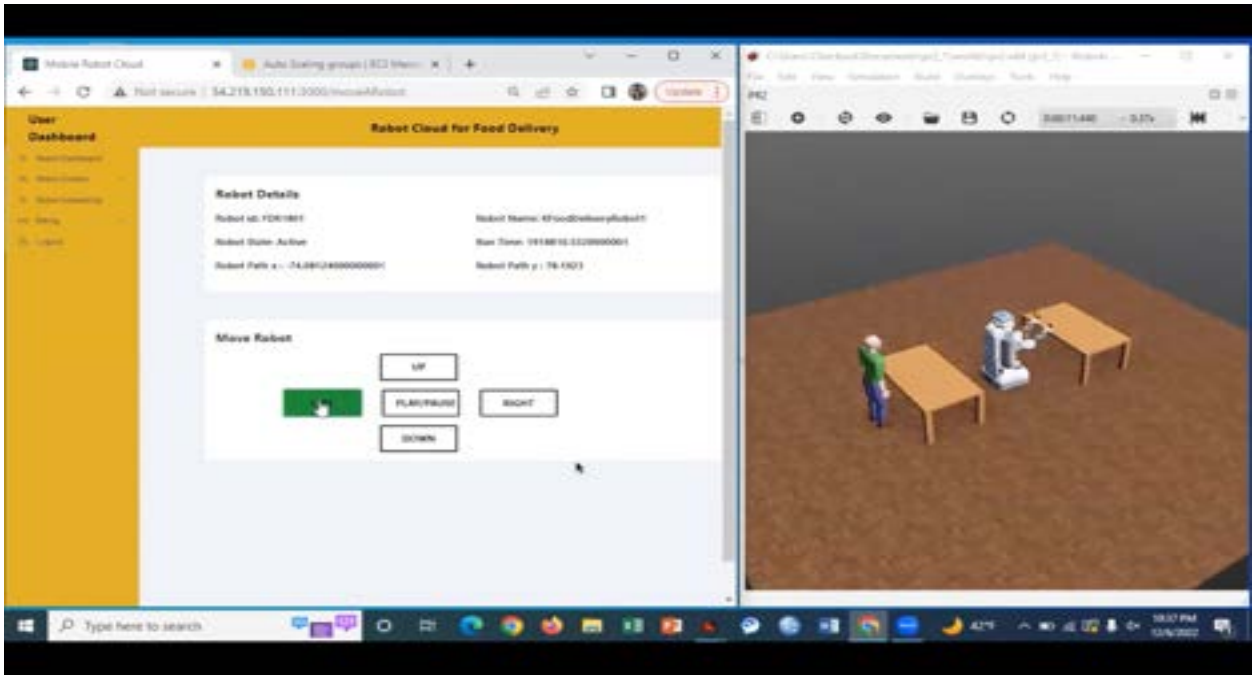

Figure 5:Single Robot Simulation

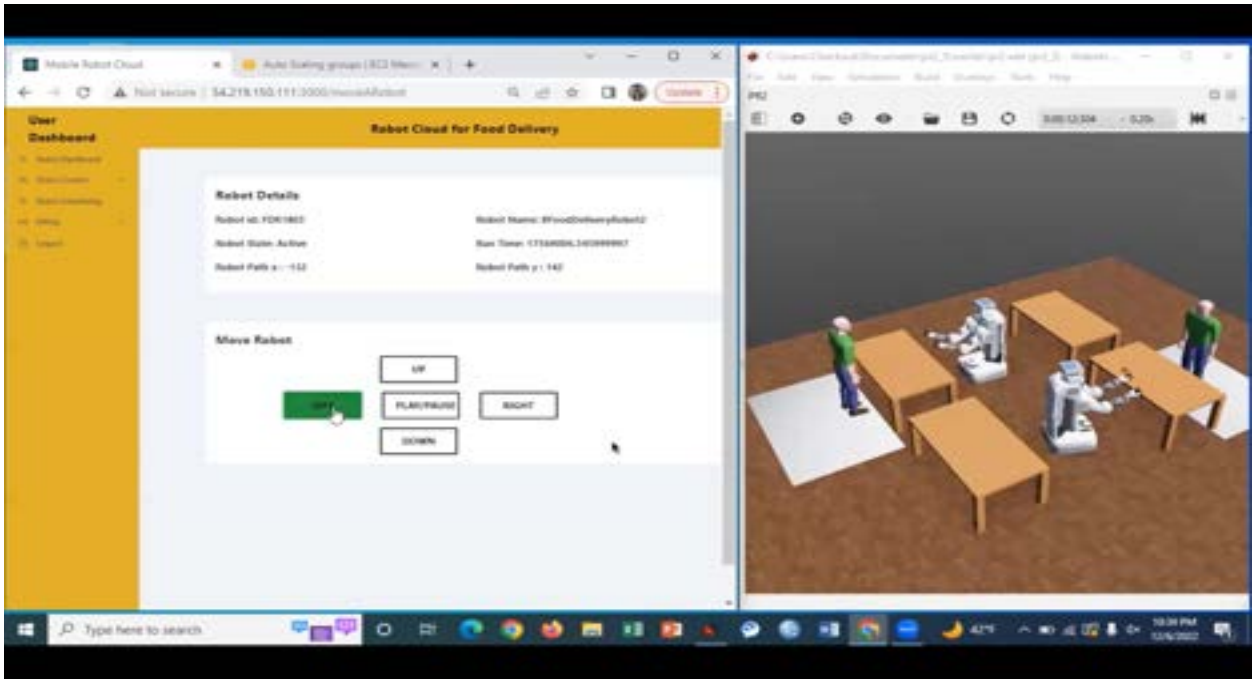

Figure 6:Multi Robot Simulation

B. Simulation Connectivity Design

The control panel for operating a robot is depicted in the image above. A development environment has been established in the EC2. The robotic functionalities' code will be written in this development environment. Robot Operating System, or ROS, was used for all of this coding. The robot can go in four directions on the map using this code. A detected impediment causes the robot to stop as well. The environment for doing this is provided by the AWS cloud, where everything is done. The whole piece of code is then executed on the robot via a simulation task. One simulation task that will activate an EC2 instance and deploy the code has been created. A robot application that will be created as a result of this will be used to simulate the robot.

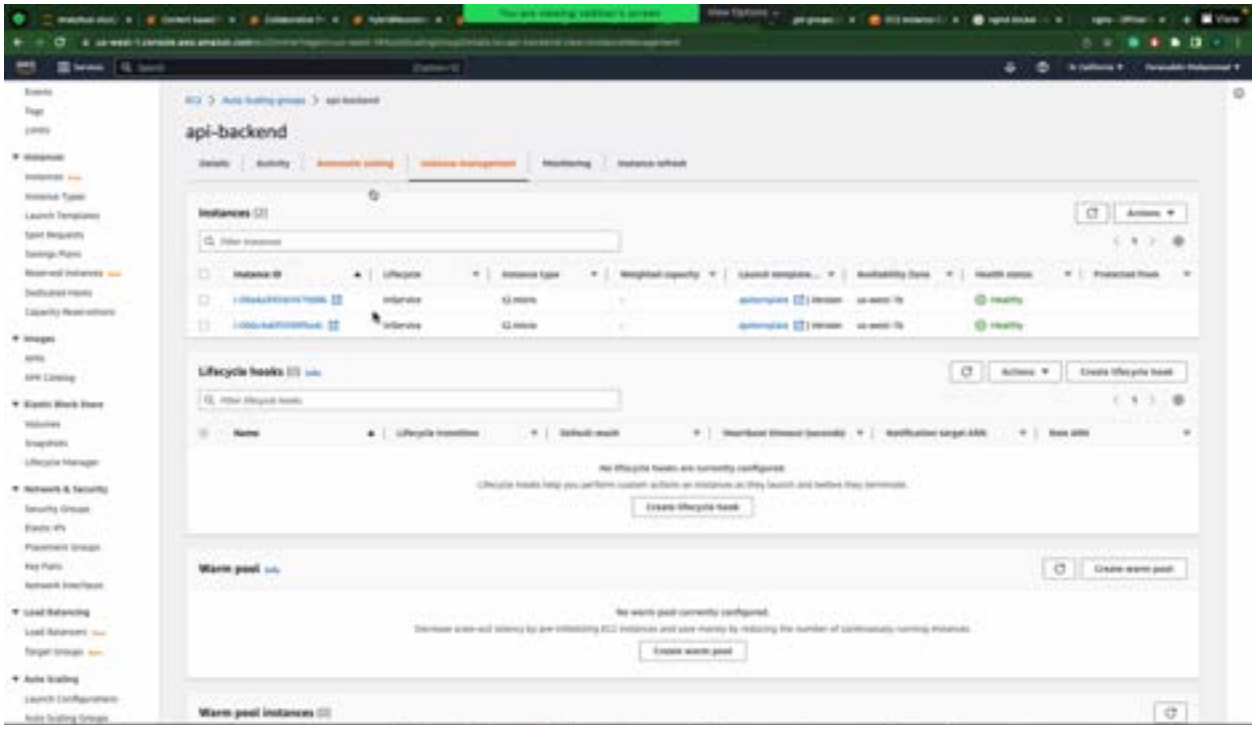

Figure 7:Deployment of instance management

5. CLOUD DATA DESIGN AND IMPLEMENTATION

The Food Delivery Robot Cloud System collects a significant amount of sensor data from sensors placed all around the location as a result of real-time data streaming. In addition to this substantial data set, we also need to manage user and robot data. We have designed a database architecture to manage this enormous quantity of data that will decrease the need for storage, speed up UI replies, and take care of backups in the event of failure.

While the conventional MySQL database would usually be sufficient, we might need MongoDB or another type of NoSQL database for the larger and more continuous data. While standard

approaches may work well for services that solely involve users, robots, booking, and scheduling, we may choose a different database management system for robot tracking and location tracking.

The database management component is in charge of keeping track of the data for the system's numerous assets, including robots, administrators, users, and maintenance units. The database component's function is to create, read, update, and delete data from the robot application's system. It enables the system's numerous user groups to share information and use the application's various functions.
The project will use two main database types:

AWS RDS (MySQL): This database will store and manage user information, robot scheduling, and booking and billing information.

MongoDB (NoSQL): This database will store the robot movement data, robot tracking information, and position information that will be continuously received, stored, and used for processing by the Webots simulator.

A hybrid database architecture using both MySQL and MongoDB is what we're going with. MySQL will be used to store the more structured data, such as user records and billing details. MongoDB will house more unstructured data about robots than the former. Let's examine the database's attribute level design. The following details will be included in the user's schema, with the user type being stored as an integer. 1 represents the end user, and 2 represents the administrative staff.

A. Cloud DB Design and Implementation (SQL DB Design)

An entity-relationship diagram is a useful tool for accurately describing the system's interrelated elements. It describes how six different things in the diagram below are related to one another. Each entity is also mapped to its associated attributes, and the relationship between the attributes and the entities is skillfully depicted in a single diagram.

• users
user_id : int
email_id: String
password: String Encrypted
first_name: String
last_name:String
role: int
address: Varchar

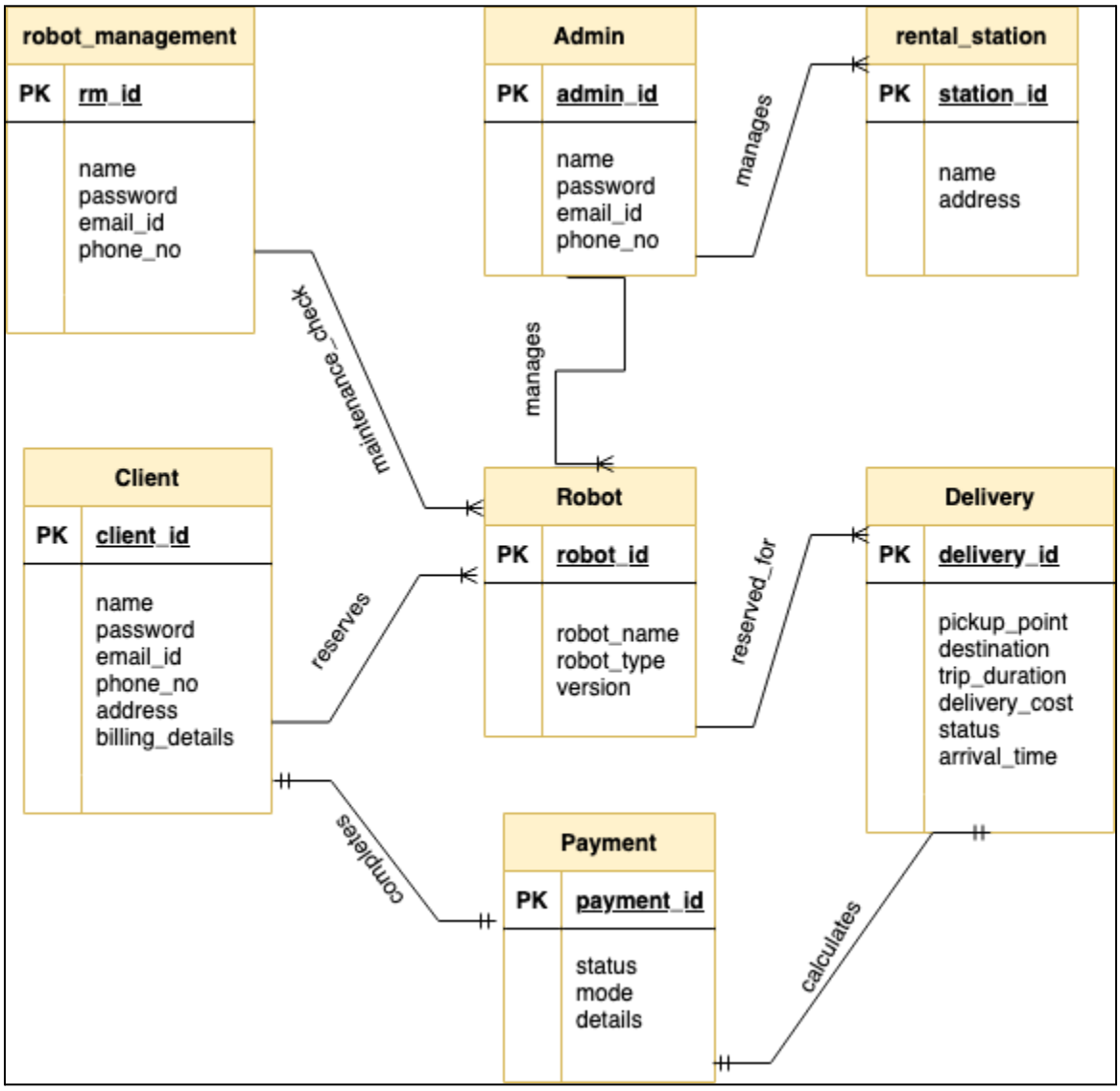

Figure 8 : ERD Diagram for Database

• **robot˙schedule**
schedule˙id: int
Robot˙id : int
user˙id:: int
run˙time:timestamp
hotel˙id: int
Floor˙id: int
table˙id: int
• **billing˙details**
billing˙id: int

Robot˙id: int
user˙id: int
run˙time:timestamp
Created˙time:timestamp
With the aid of AWS RDS, this MySQL database is hosted in the cloud, enabling remote access to the cloud and providing scalability and usability. The application frontend and Webots server are able to send and receive various queries to post and retrieve data about a user, robot, or schedule thanks to a Node Express backend API. The diagram that follows shows the API routes for the table "Users."

B. System Big Data Design and Implementation (NOSQL DB Design)

In our project, MongoDB is used to store and track robot data transmitted through WeBots APIs. To work with massive data, such as the robot tracking data posted through the robot objects, MongoDB offers flexibility and ease of use. The application Atlas houses the MongoDB instance, making it safe and secure.

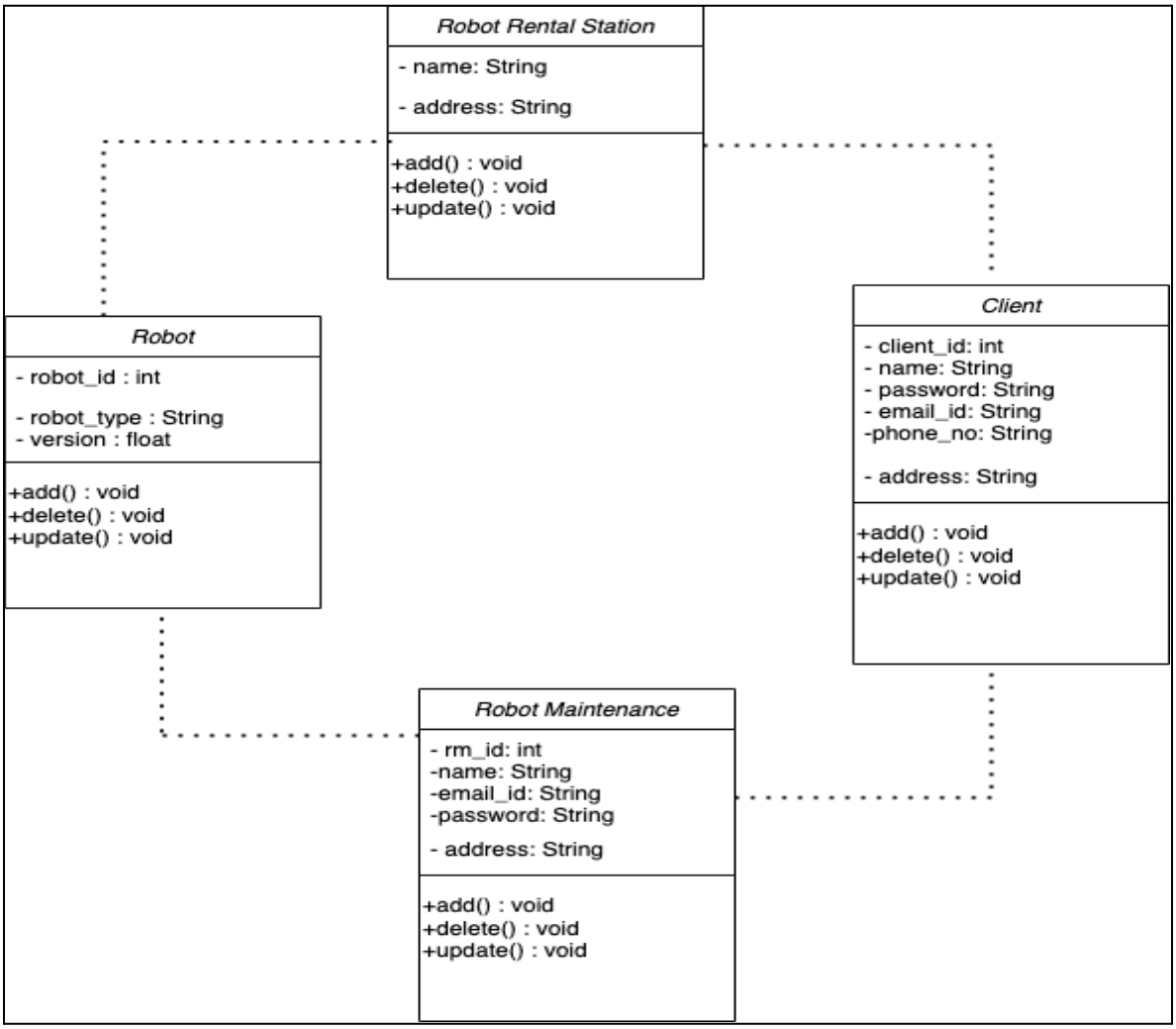

Figure 9: NoSQL DB Design

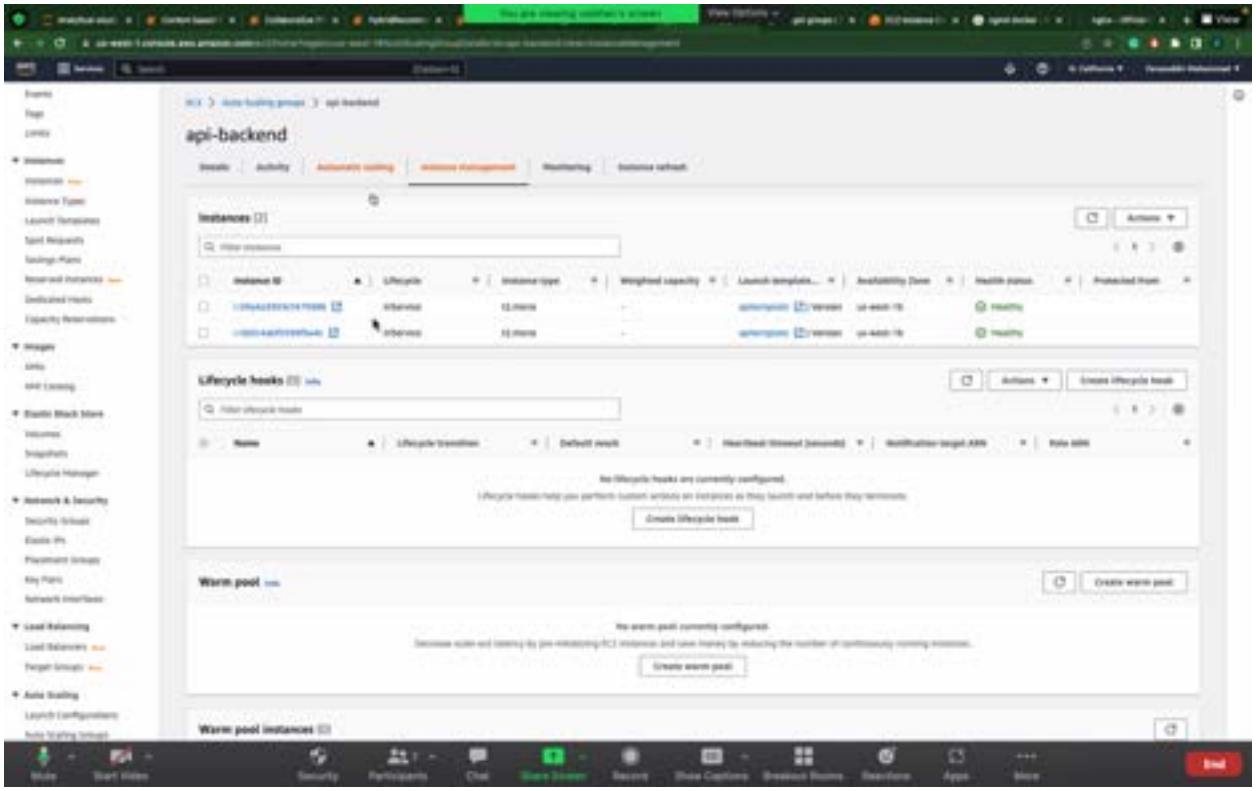

Figure 10:Deployment Health check Status of instance

Robot collections will be the sole schema in MongoDB. It includes details about the robots. The robopath schema contains an array of the coordinates it has already traveled through. No matter if it is active or suspended, Robo State will always be a string. In order to determine the billings based on the number of sessions the user utilized the robot, we are also maintaining the run times.

Robots

● Robo_id : Sting

● Robo_path: array of objects which are coordinates

[{

X-coordinate: int

Y-coordinate:int

}]

● Robo_state : String

● User_id: String

● Run_time: array of objects which are sessions

[{

Start_session_time: timeStamp

End_session_time: timeStamp

}

6. Cloud system design

6.1. System Function Component Services (such as Billing, Connectivity,..)

Edge-based mobile robot

WeBots are being used for testing and modeling of the edge-based mobile robot. With the help of the developed GUI and the WeBots, we are building a simulated robot that can be watched. Here, we may build several worlds and maps, populate the navigation, and use this map for testing and simulation. WeBots can be used to deploy the ROS-developed robotic control to the simulated environment.

Database management component(s)

For the project's database management, both SQL and NoSQL technologies have been deployed. The relational database is Amazon RDS, and the no-SQL database is MongoDB. The Amazon RDS stores any transactional data that needs a relational schema, such as user and robot details and user-robot relationships. The MongoDB is used to store information on robot navigation, service operations, the path of the robot, billing details, etc.

User service management

The platform's users are managed using this component. Public MRC users will use the platform to operate the robots, and admin users will be able to track, monitor, and manage the billing and other details for the users. Anyone can use the platform to create and manage mobile robots by signing up with their email address.

Remote online robot tracking and controlling

The built graphic user interface can be used to monitor and manage the simulated robot. This GUI will communicate with the AWS Robomaker to control the robot's navigation. This controlling element allows the robot to be moved in all directions. Additionally, we may follow the robot's route using its x and y coordinates. This component can also be used to create and deactivate robots. The servicing activities carried out on the robot will be noted and billed appropriately.

Online system dashboard

The admin users of the platform can utilize this component to track various parameters and data. This is particularly useful for keeping track of the number of users who have registered, the number of robots who have registered, the number of active and inactive robots, the pathways and locations of various robots, the number of service operations carried out on each robot, etc. Only users who are registered as Admins have access to this.

6.2 System Scalability Design and Implementation

We will employ an AWS Auto Scaling Group, which is an effective method of scaling the servers on demand, to increase the application's scalability. It is necessary to provide a minimum and maximum number of instances for the AWS Auto cluster. By default, a minimal number of instances are used to launch the cluster. If the cluster is overloaded, it immediately creates a new instance with the aforementioned specifications, which can relieve some of the strain on the existing servers. When a new instance needs to be created, the metrics need to be configured.

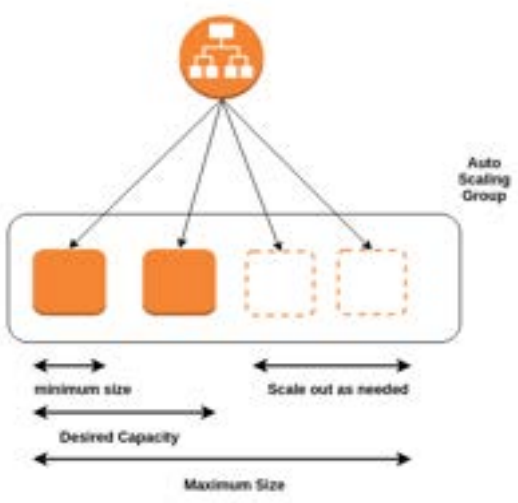

Figure 11: Auto Scaling

In our scenario, we will define the metric as a CPU usage of 60%, meaning that if the CPU usage of any of the initial servers exceeds 60%, a new server will be created with the predefined configuration.

6.3. System Load Balance Design and Implementation

We're going to employ an AWS elastic load balancing method for the frontend servers to equalize the demand on them. By distributing the requests in a round-robin method, it evenly spreads the load among all front end servers.

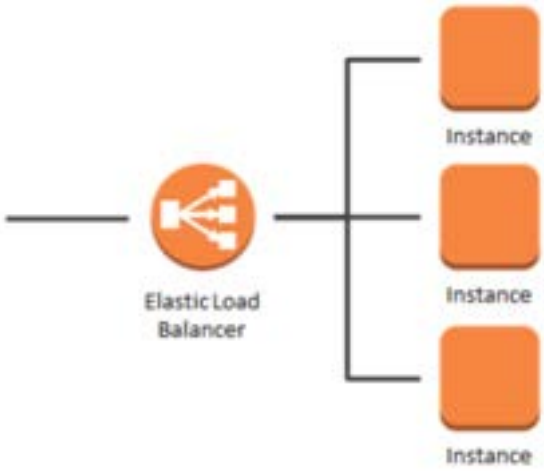

Figure 12 :Elastic Load Balancer

The architecture of frontend servers is depicted in the diagram below. The load-balanced system will be used by the end user. The load balancer evenly distributes the workload across all of the frontend servers.

7. System GUI Design and Implementation

7.1 System GUI Design

We created a simple, reusable GUI so that administrators and users may interact with the platform. Because React JS makes the single page web application architecture simple to develop, we chose it to implement the GUI. On the UI end, we managed stores using Redux, and we interacted with the backend using HTTP using the "Axios" package.

7.2 Implementation

- User: On the user website platform, user has the following alternatives to pursue:
- Users can start a simulator work for themselves and develop a robot in the AWS Robo creator by using the program.
- Through the web application, users can begin simulating the robot and doing so in the specified world. Users can use the guidance provided in the application's control panel to traverse the virtual environment.
- Through the integrated backend APIs, the robot's line of travel in the simulated environment is constantly synchronized with the application every second.
- Depending on his use, the user can control how to view the robot's condition. The robot is in an active state if he is utilizing it in the simulated world; otherwise, it is in an inactive or disconnected condition.
- Several screenshots of the user application are shown below:

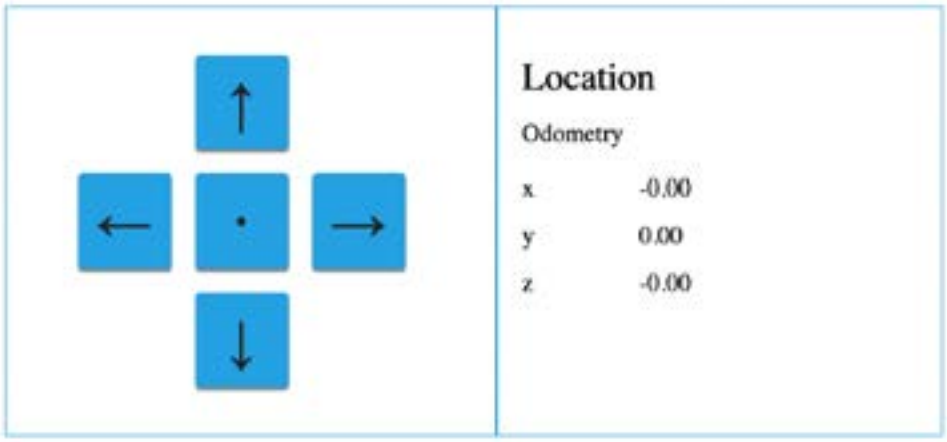

Figure 13 : Tracking and monitoring

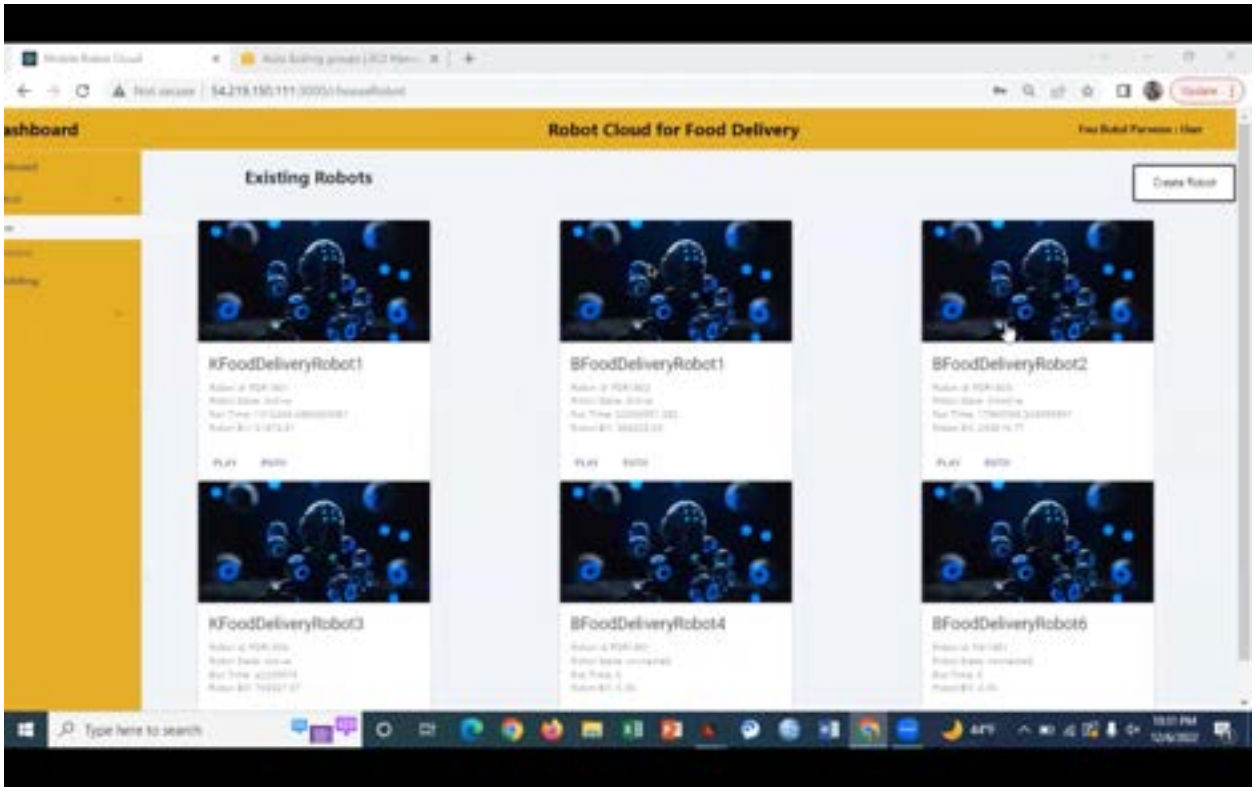

Figure 14 : Number of Registered Robots

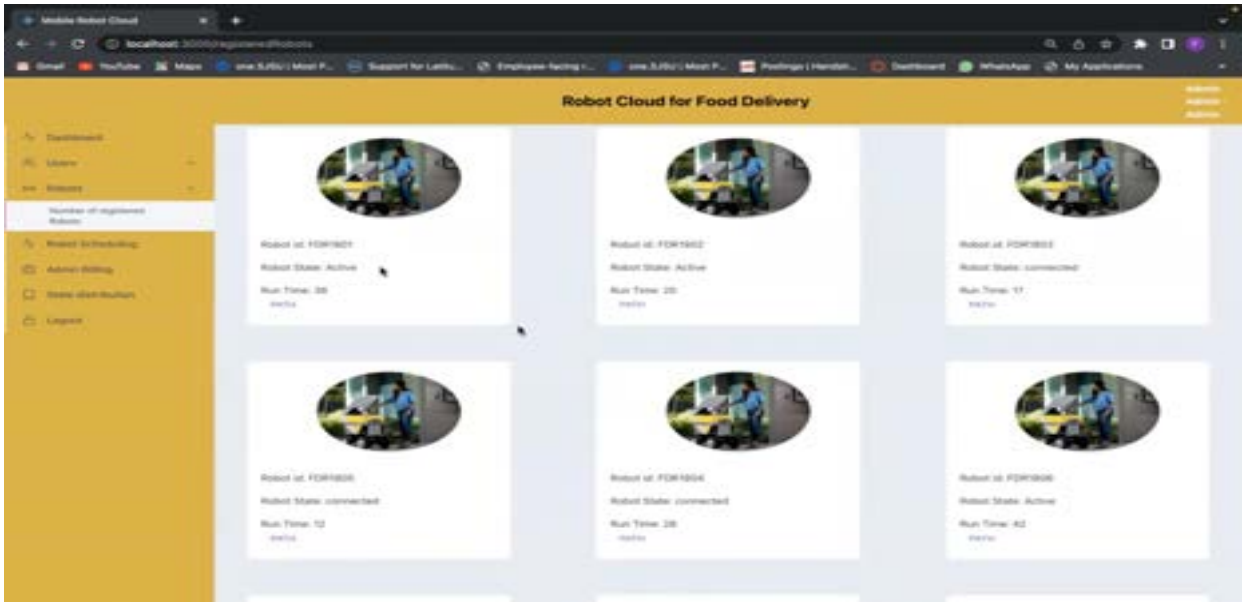

Figure 15

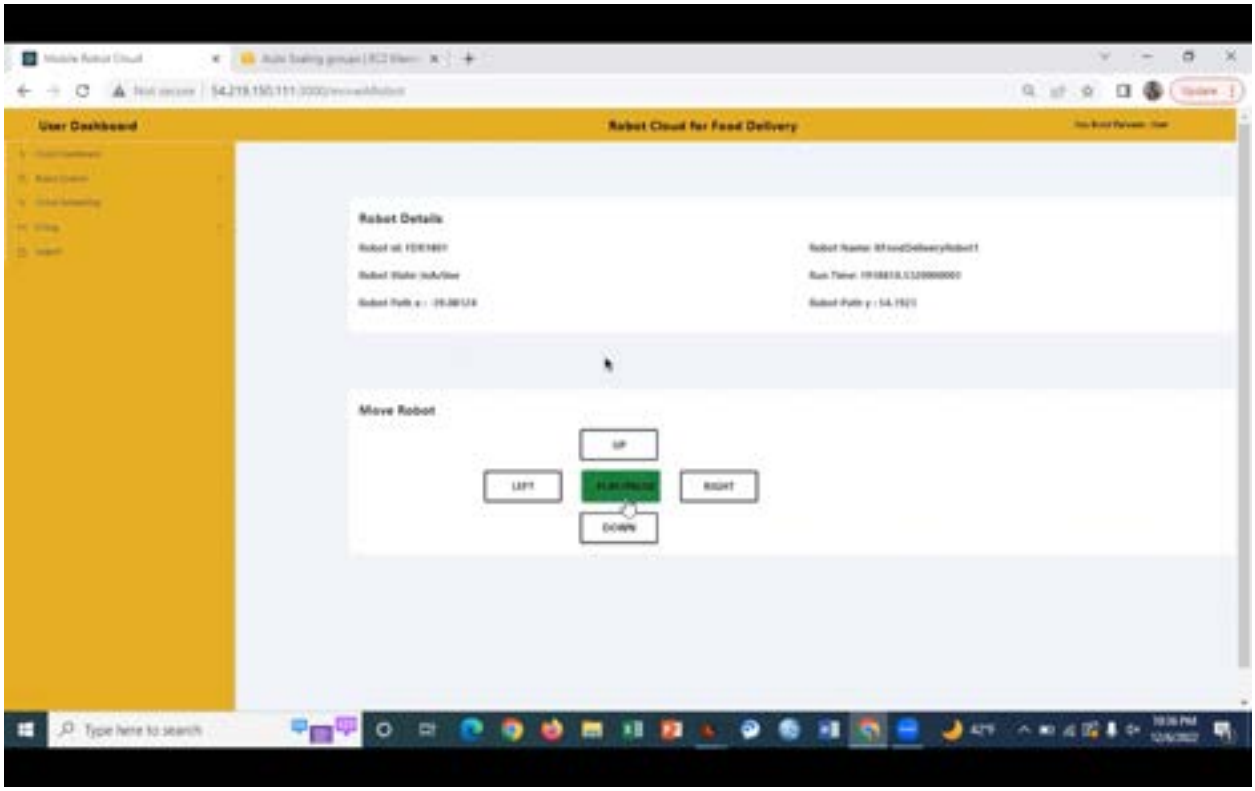

Figure 16

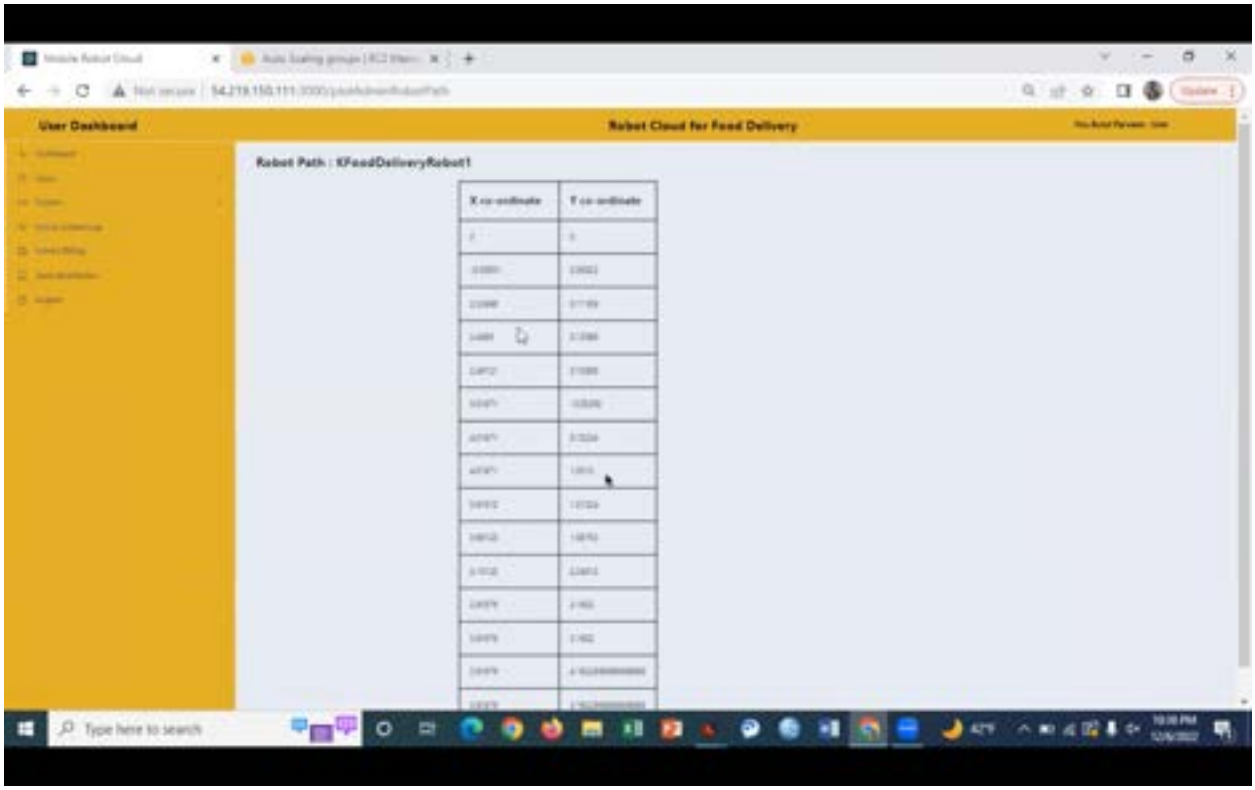

Figure 17

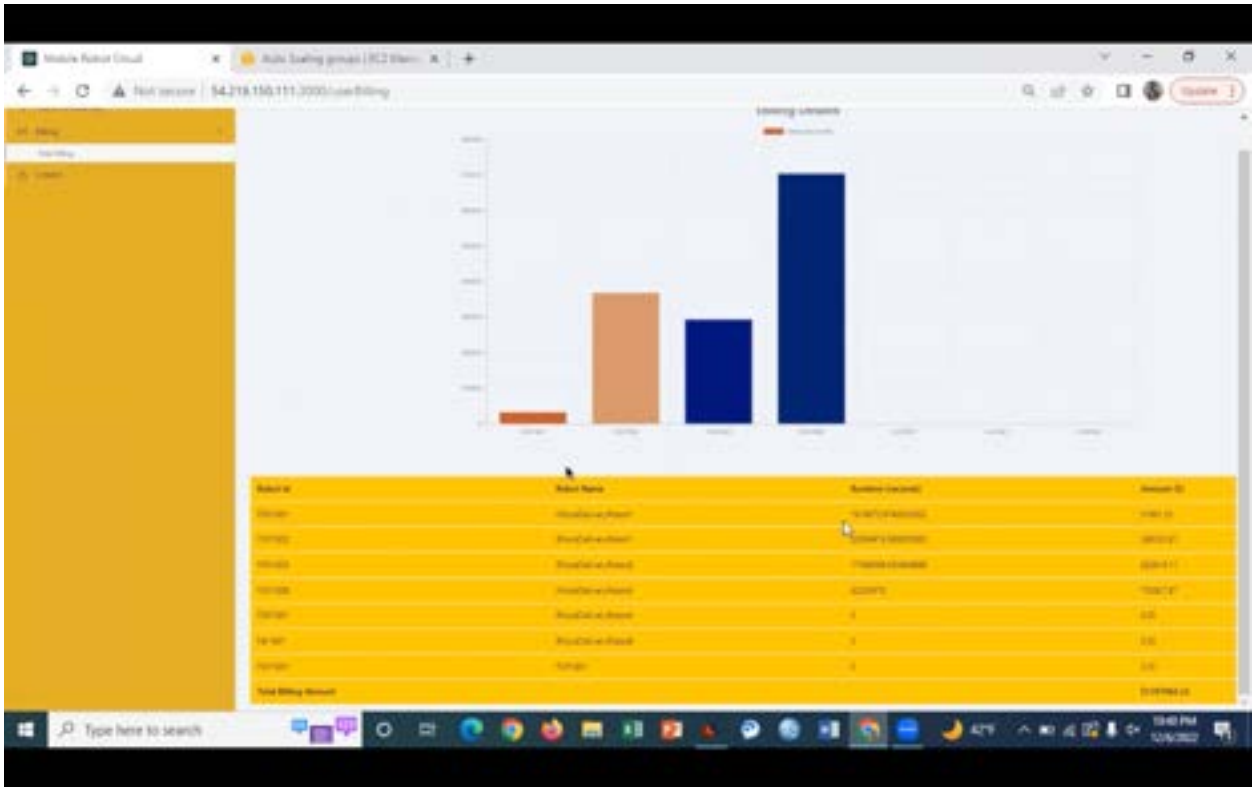

Figure 18

Admin: This application's admin user or service user can log in to the system and view the list of registered users and robots as well as the list of current users and robots using the application. The admin has the right to view the state distribution of the robots on the platform, as well as the billing information for each user on the platform. Several images of the application's admin dashboard:

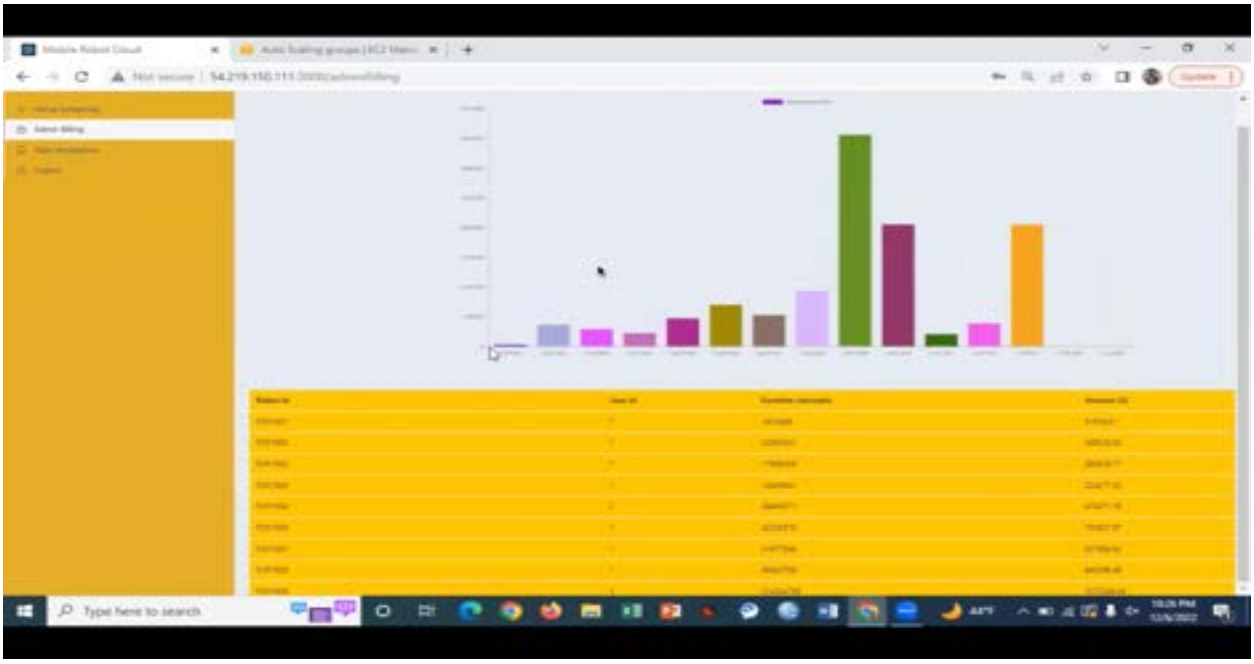

Figure 19

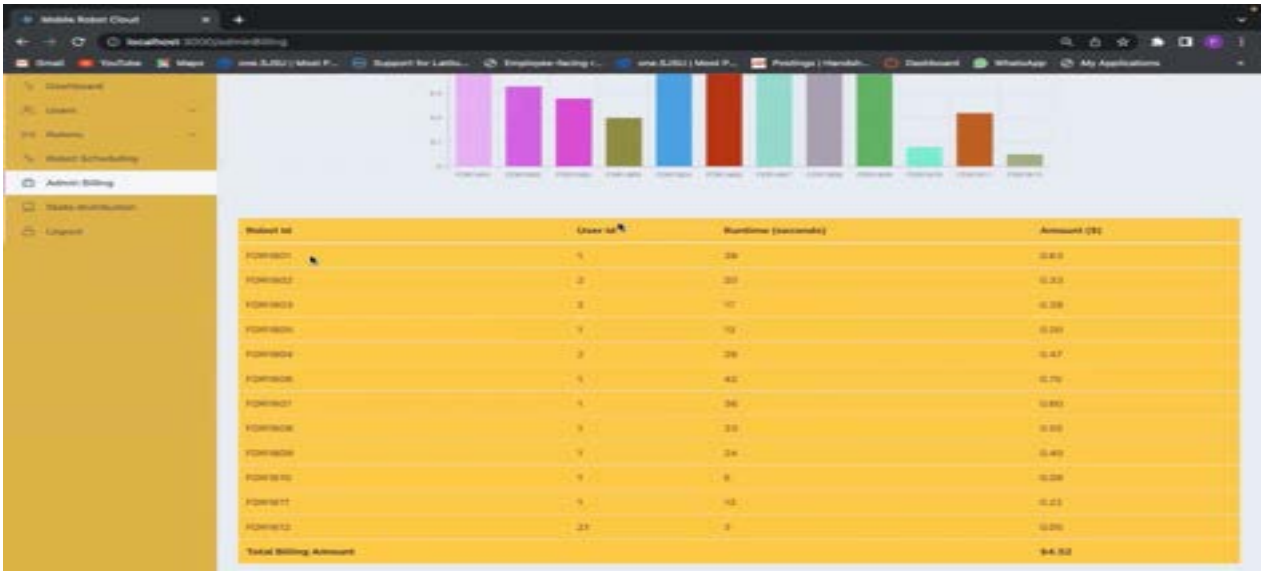

Figure 20

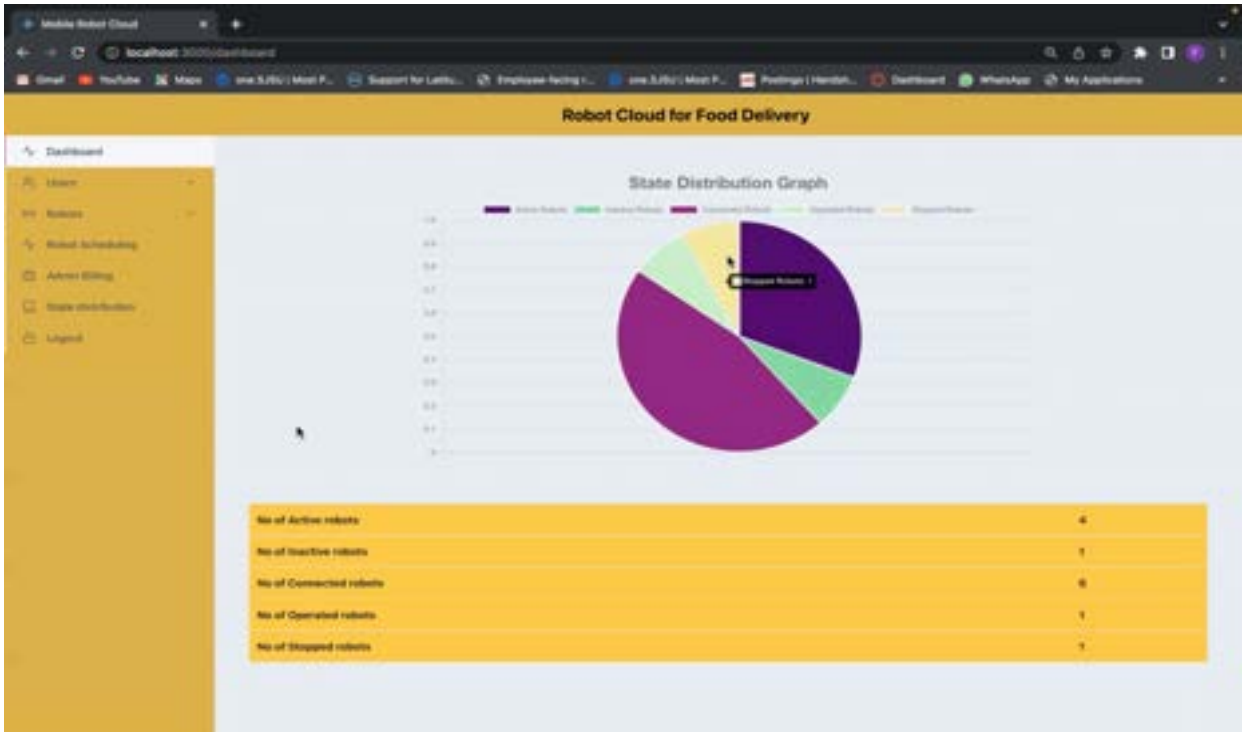

Figure 21

8. Edge Station Design and Simulation

WeBots are being used for testing and modeling of the edge-based mobile robot. With the help of the developed GUI and the Webots, we are building a virtual robot that can be observed and navigated. Here, we may build several worlds and maps, populate the navigation, and use this map for testing and simulation. Webtos can be used to deliver the robotic control to the simulated world after it has been constructed using ROS.

The built graphic user interface can be used to monitor and manage the simulated robot. To direct the navigation of the robot, this GUI will communicate with the Webots. This controlling element allows the robot to be moved in all directions. Additionally, we may follow the robot's route using its x and y coordinates. The Robot Component component can also be used to create and deactivate robots. The servicing activities carried out on the robot will be noted and billed appropriately.

9. System Application Examples

In situations where the persona (admin or user) needs a web interface to simulate and run robots with cloud support, save and track the status of their robots, the Mobile Robot Cloud for Food Delivery (MRC) program is employed.

Imagine such a situation where a user wishes to emulate a robot that can move about and follow its path. An administrator is the one who provides this user interface. The administrator can see the operations carried out by the user and charge him according to his usage. The following figure displays the application's flowchart.

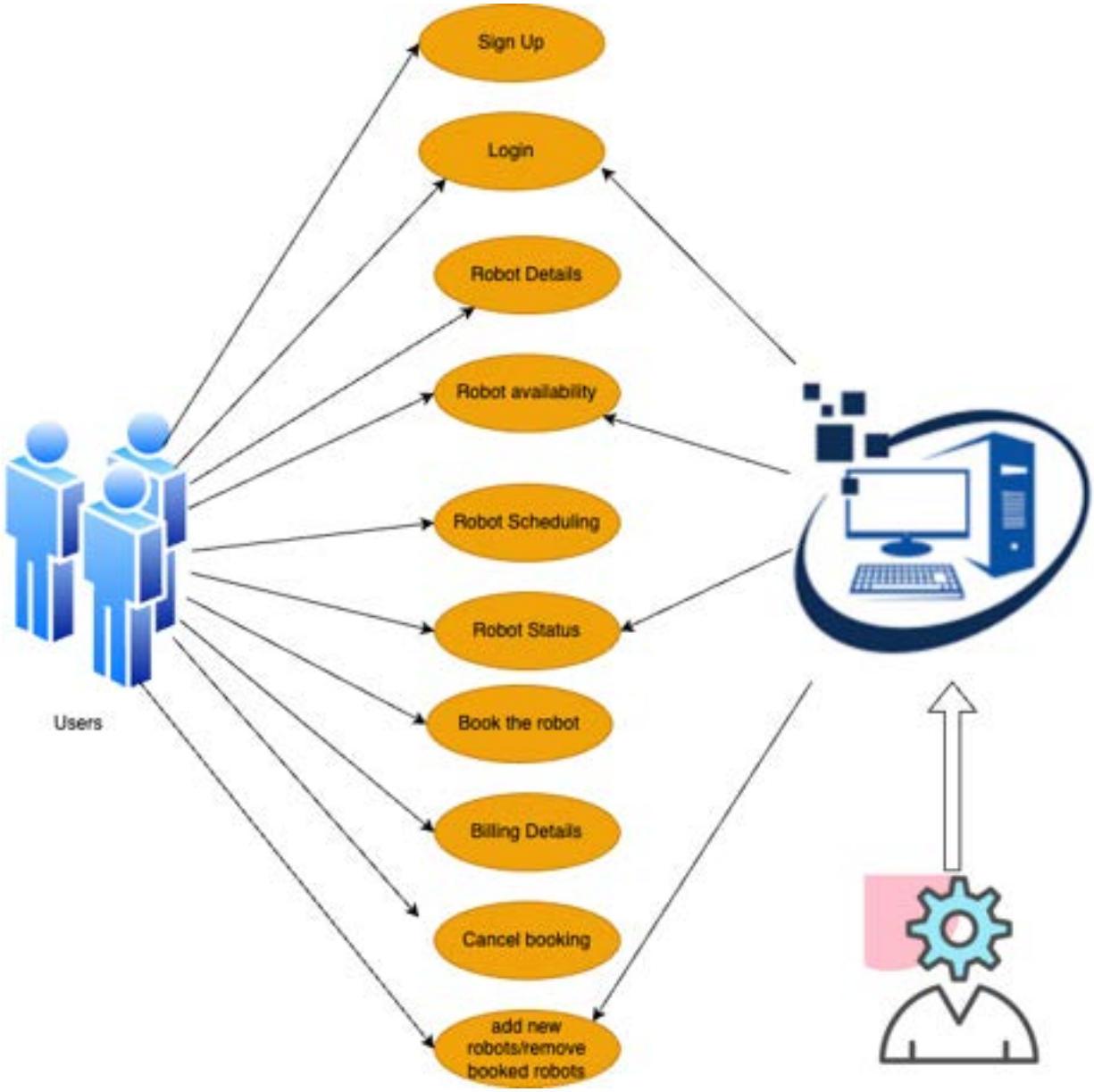

Figure 9.1

User Scenario:

1. The user can register and login to the system as shown in Figures 9.2 and 9.3.

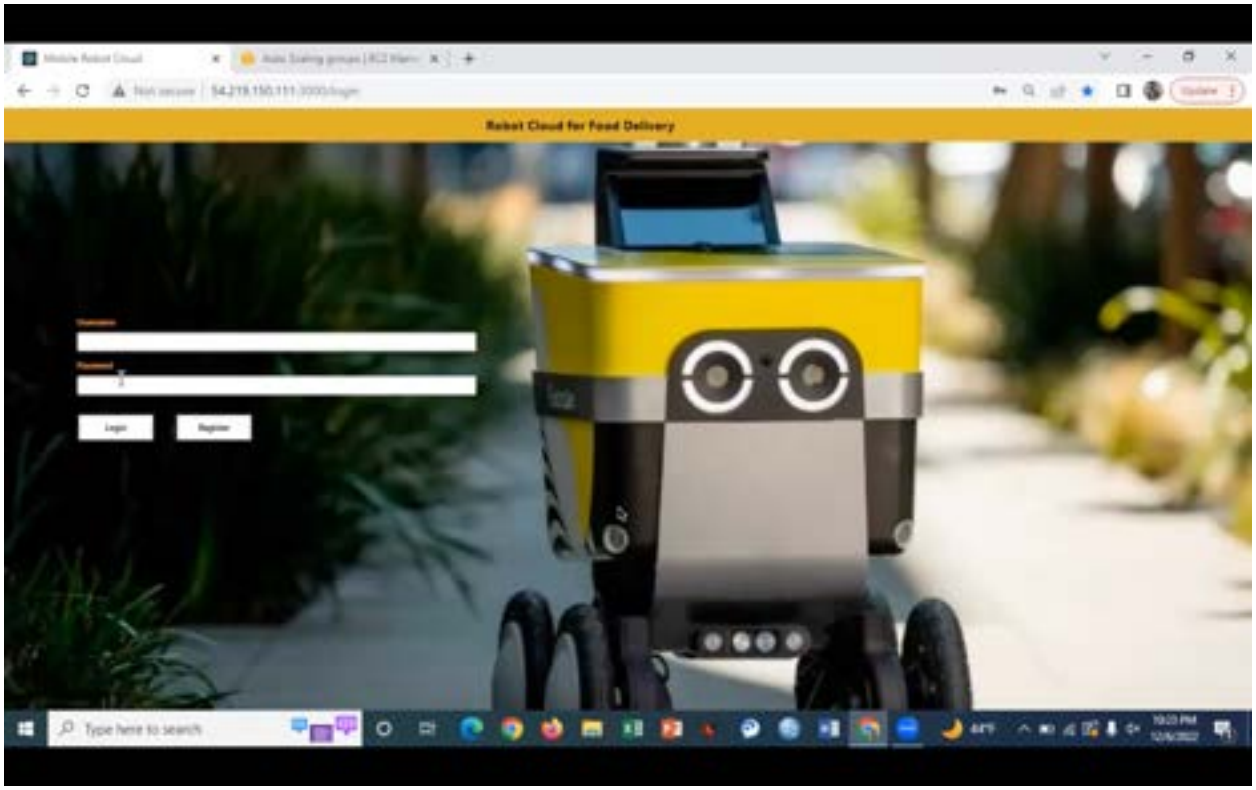

Figure 9.2 : Login

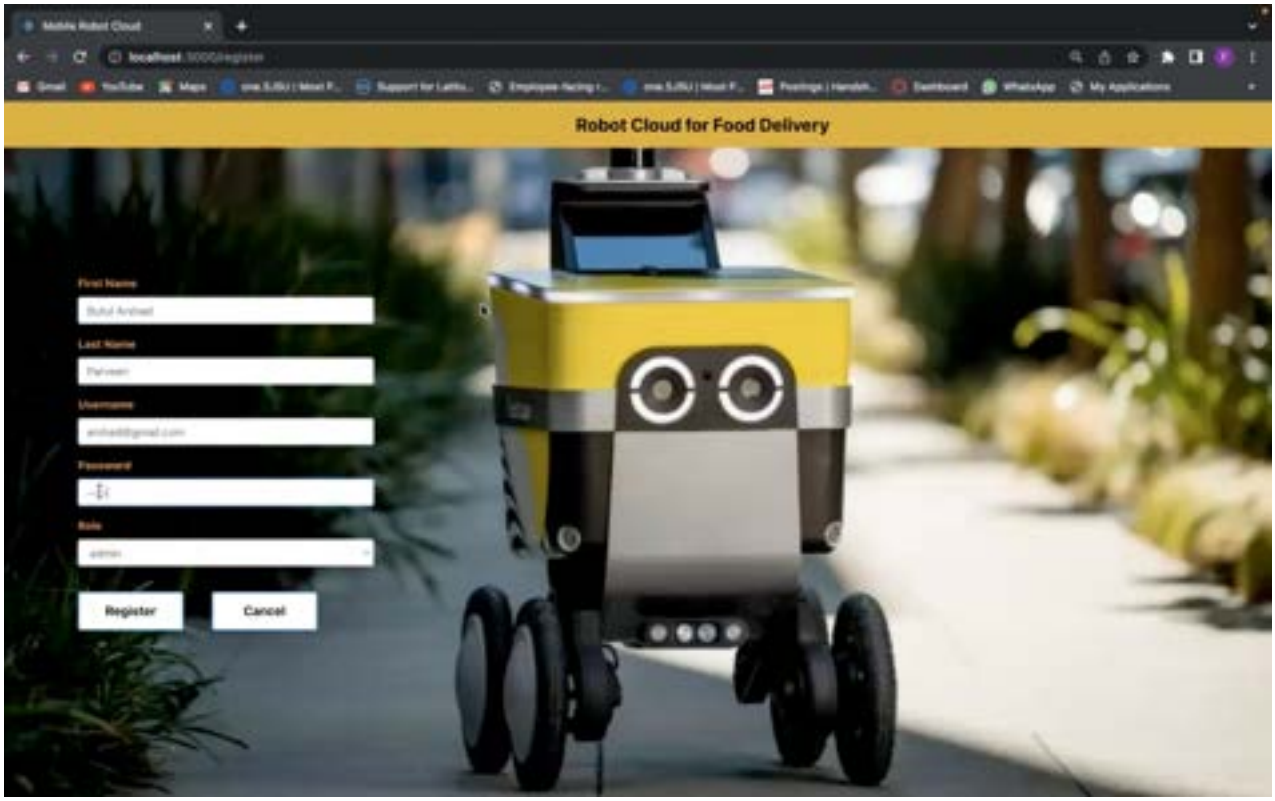

Figure 9.3 :Register

2. He can check out his dashboard, as seen in Figure 9.4, where he can monitor the status of his robots and their running times.

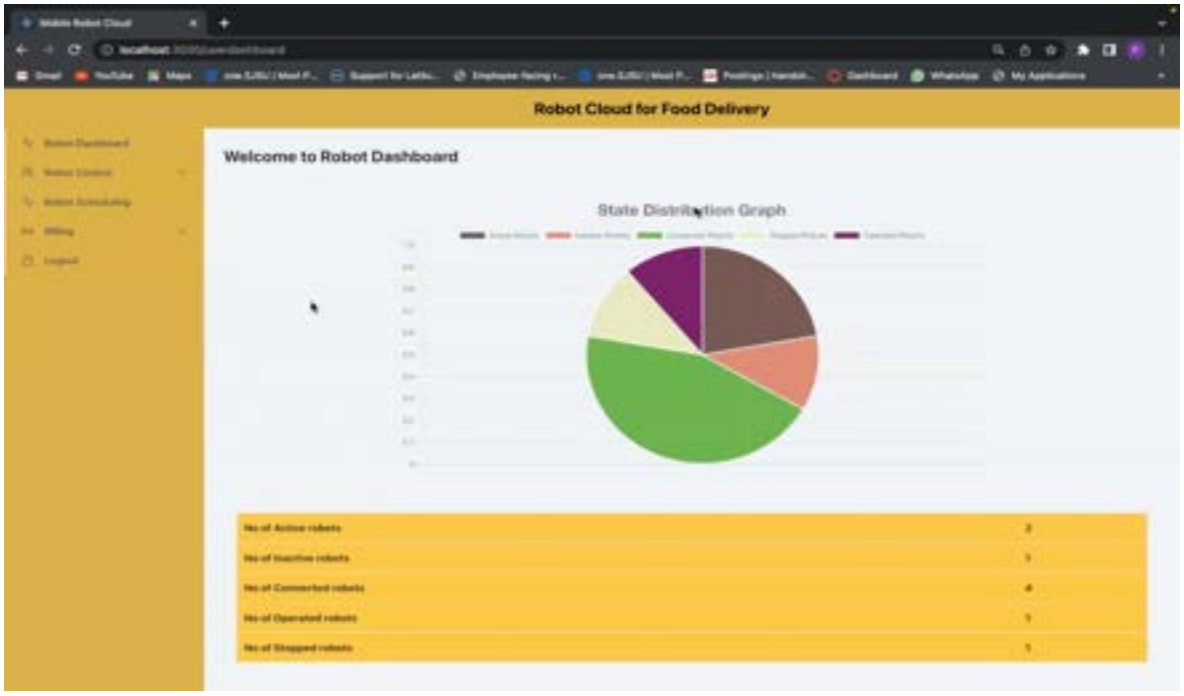

Figure 9.4 :Dashboard

3. He can choose to view all of his currently deployed robots, as shown in Figure 9.5, and the path that he has taken in relation to a certain robot, as shown in Figure 9.6.

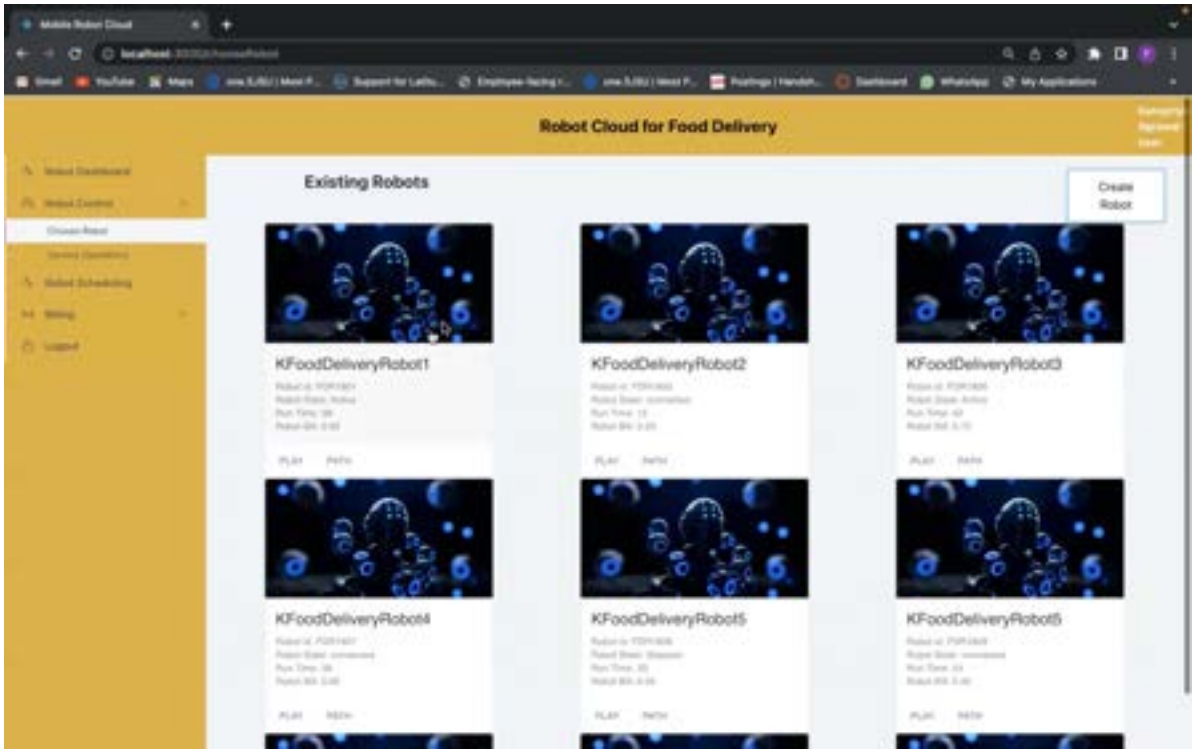

Figure 9.5

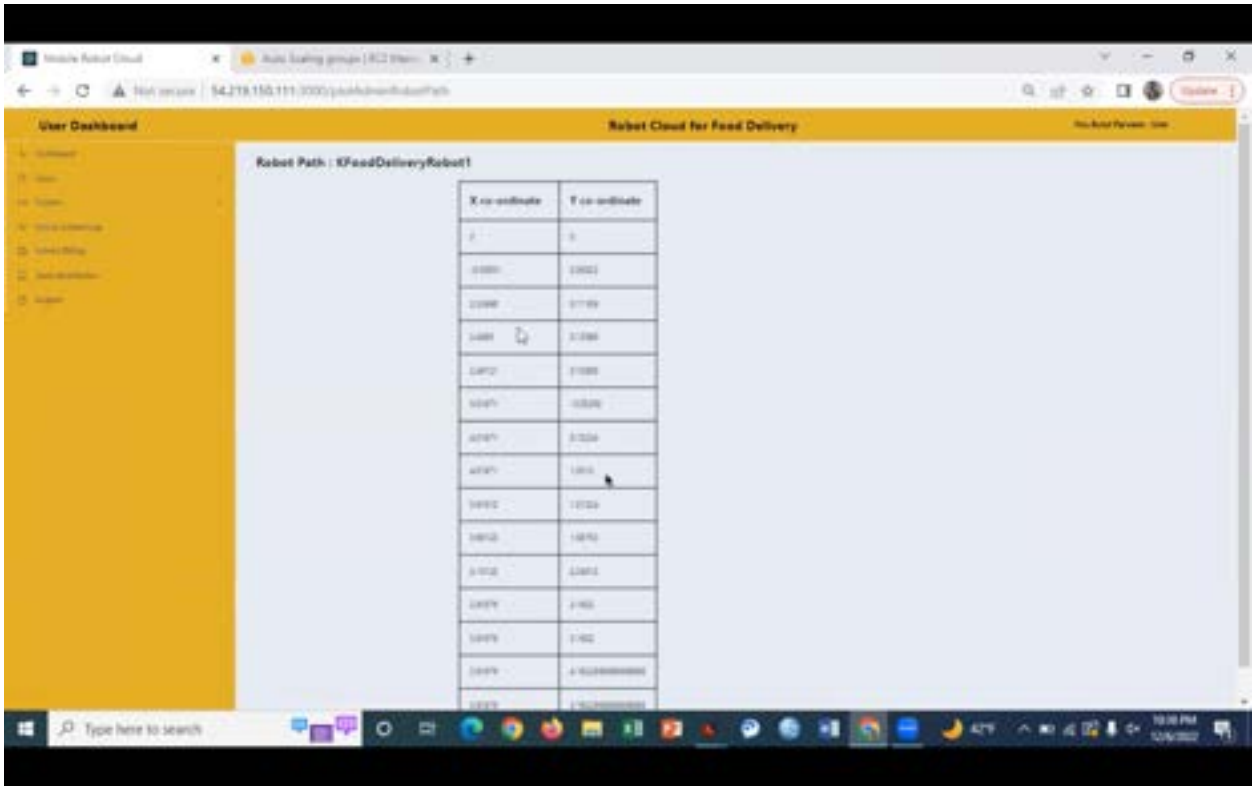

Figure 9.6

4. He can then perform several operations in a robot-like simulation, check his billing, and more, as depicted in Figures 9.7, 9.8, and 9.9.

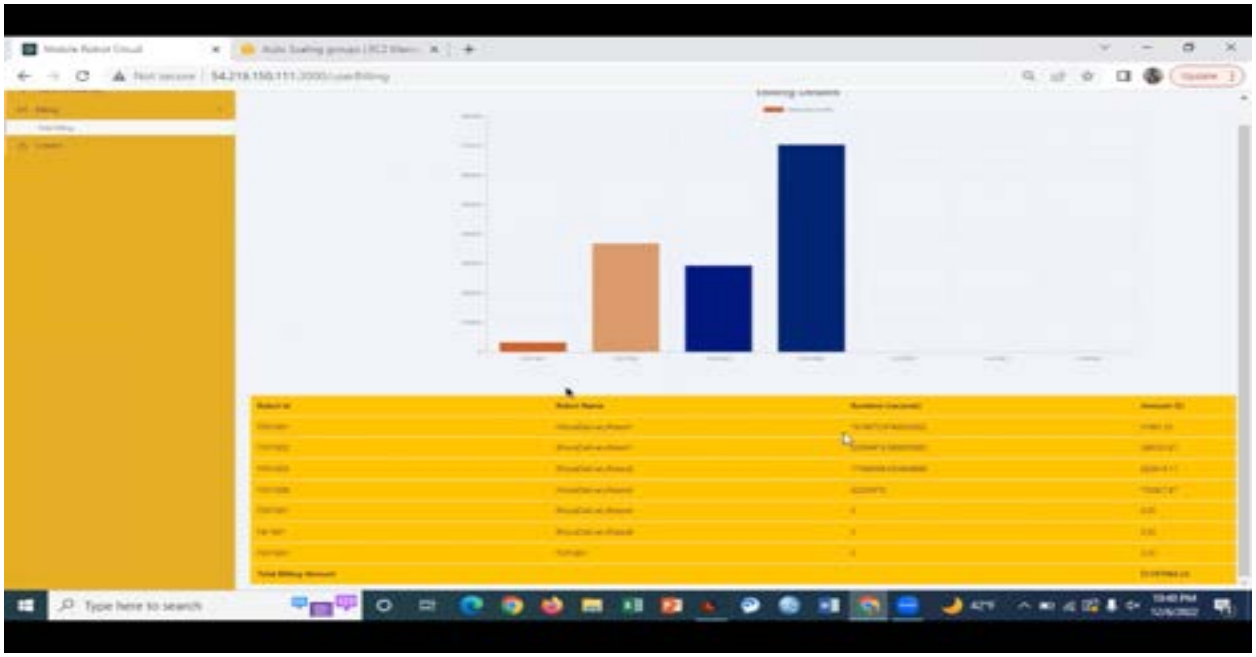

Figure 9.7

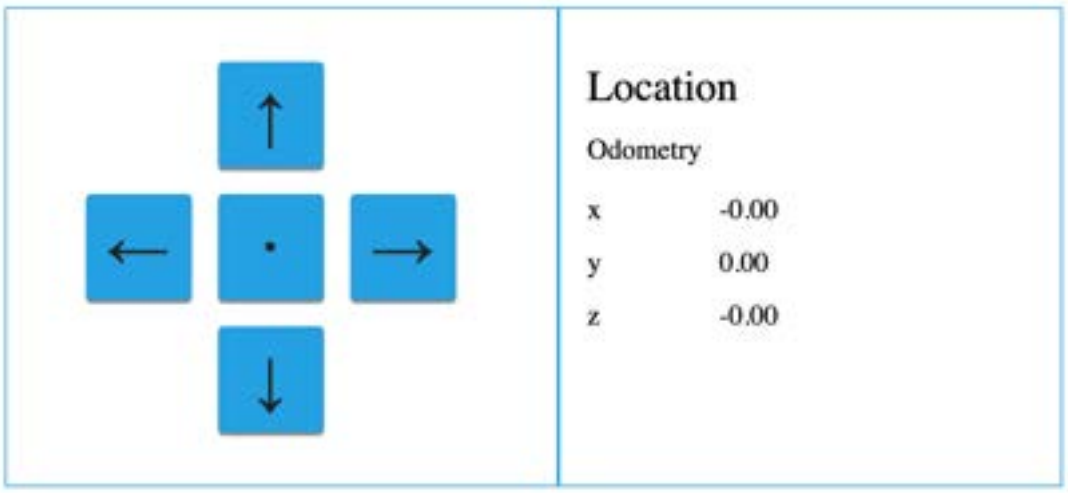

Figure 9.8

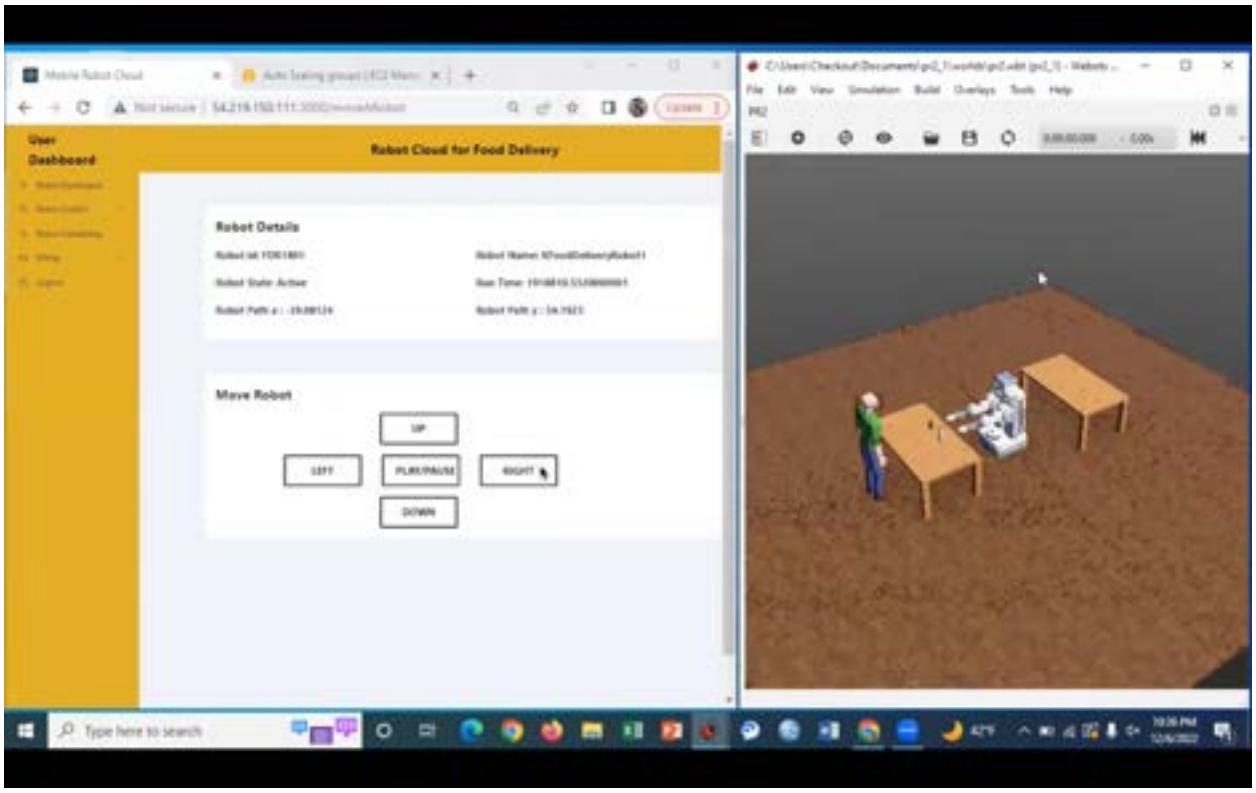

Figure 9.9

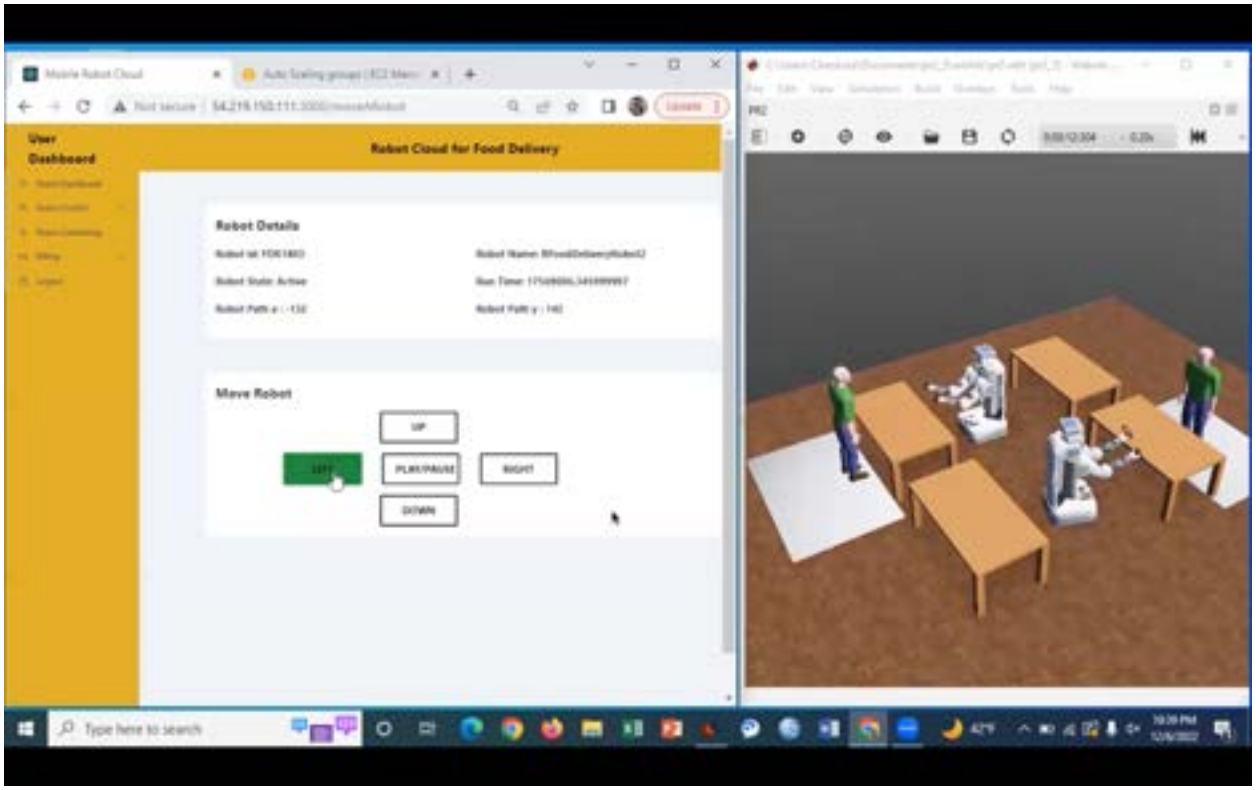

Figure 9.10

5. The administrator can log in to the system and view his dashboard, which is depicted in Figures 9.10 and 9.11, to see how his system is distributed.

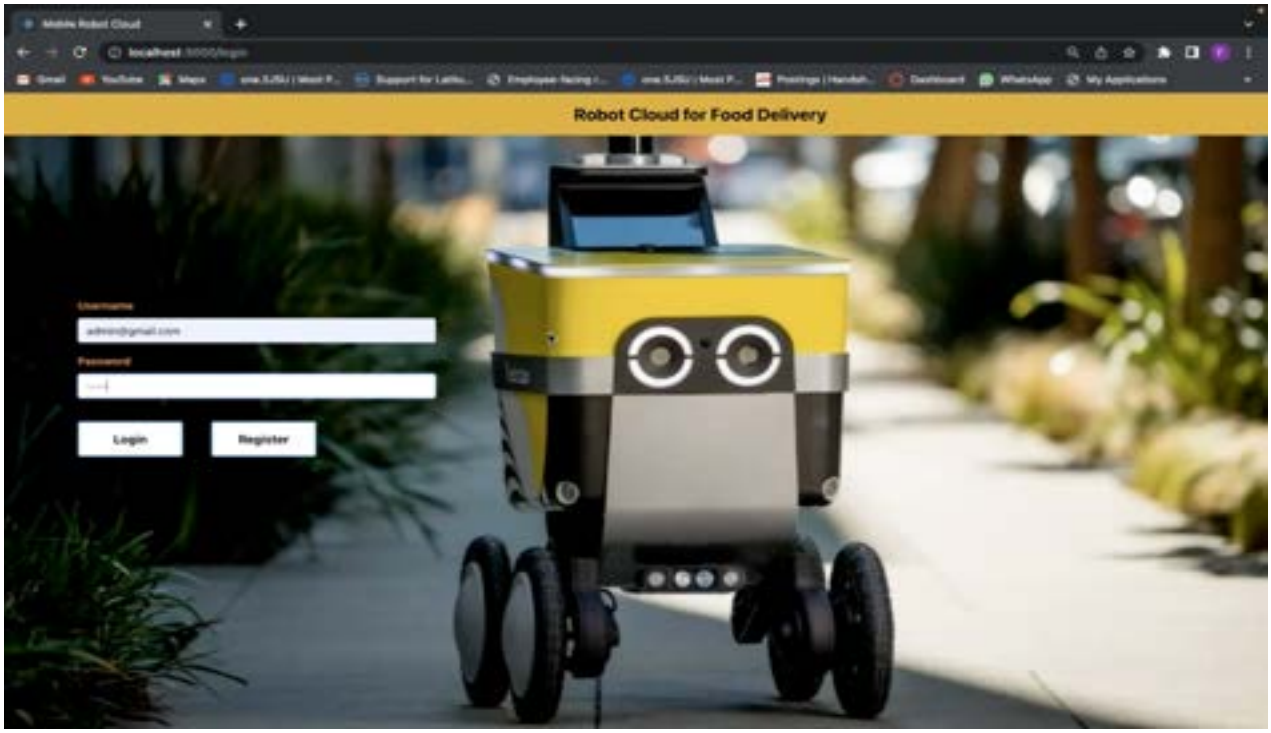

Figure 9.11

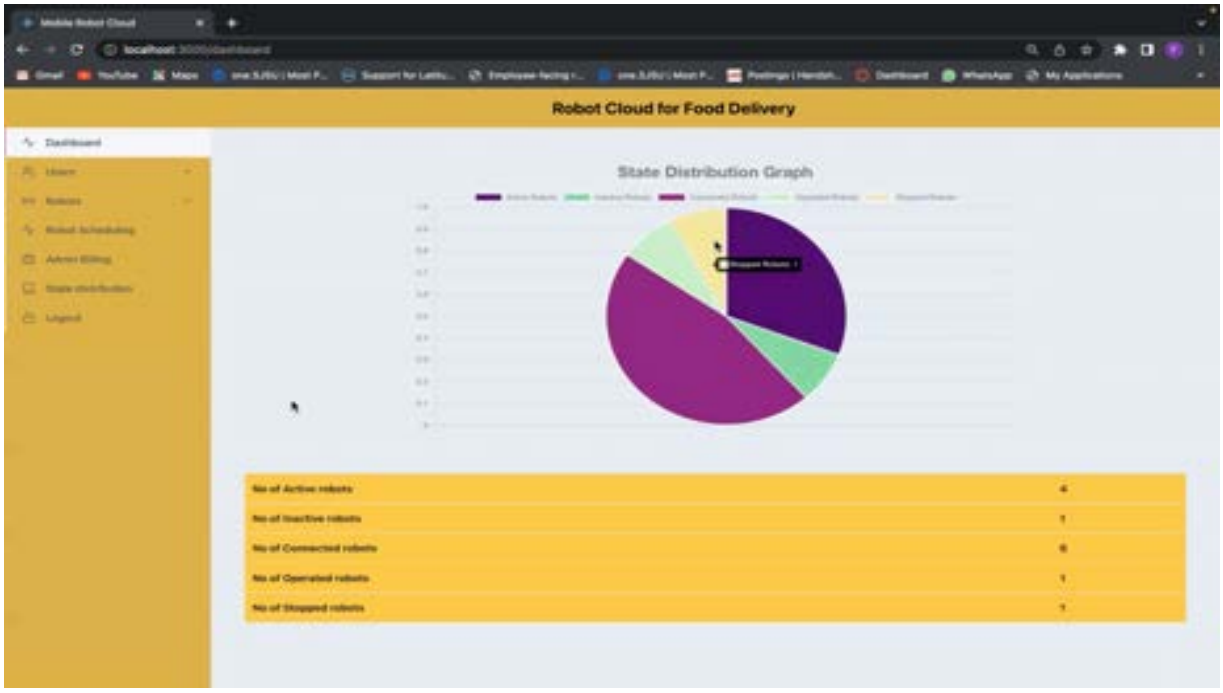

Figure 9.12

6. He can review all of the current users, robots, billing information, and operations as shown in Figures 9.12 and 9.13.

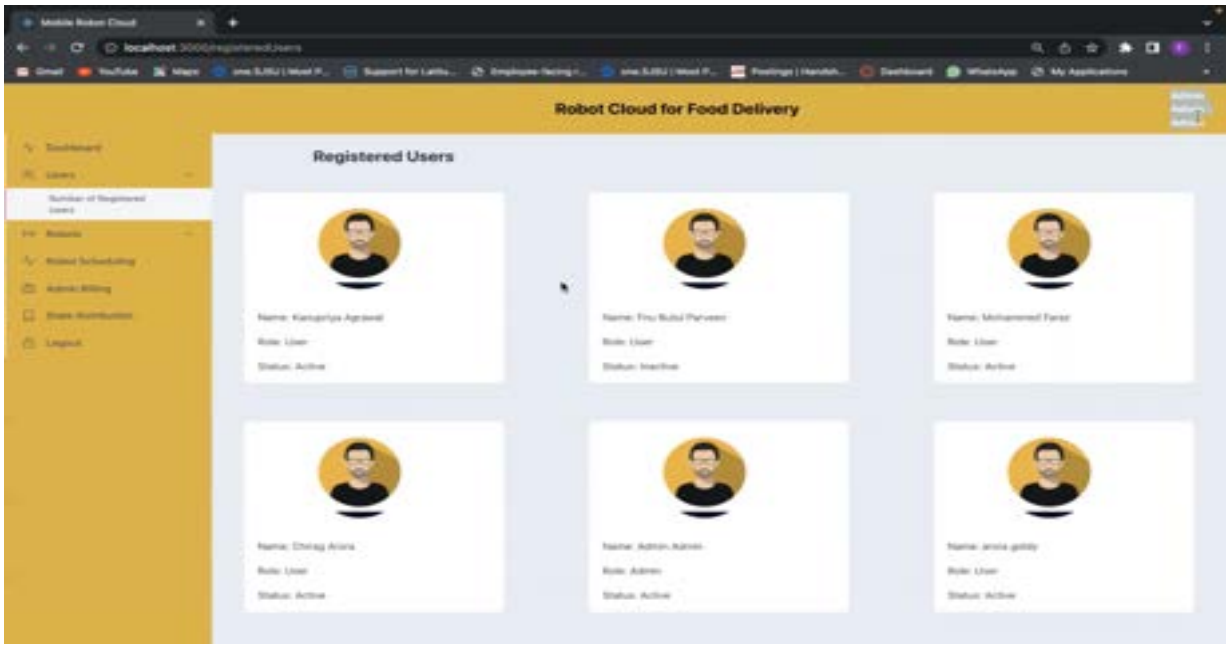

Figure 9.13

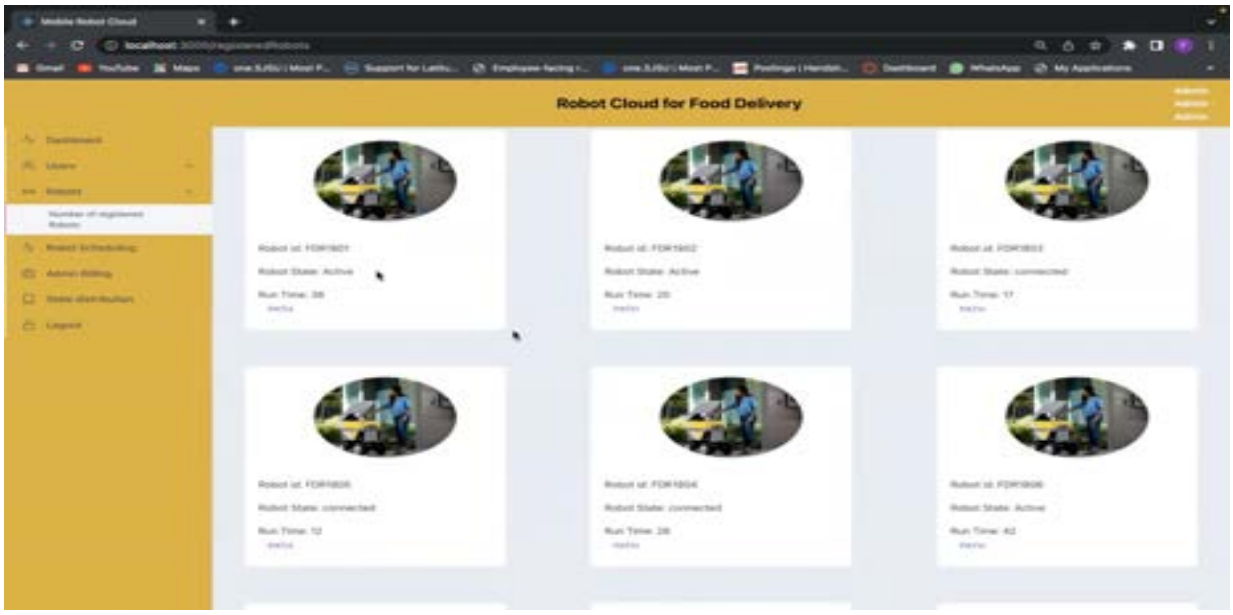

Figure 9.14

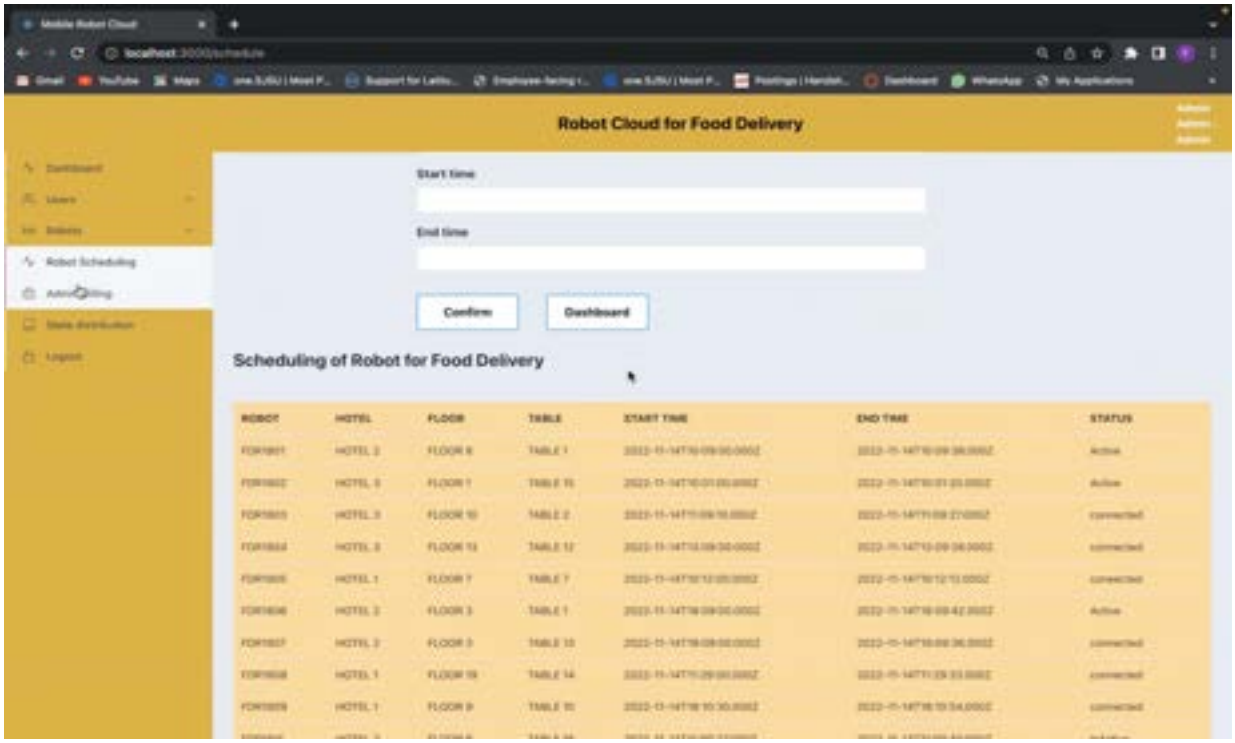

Figure 9.15

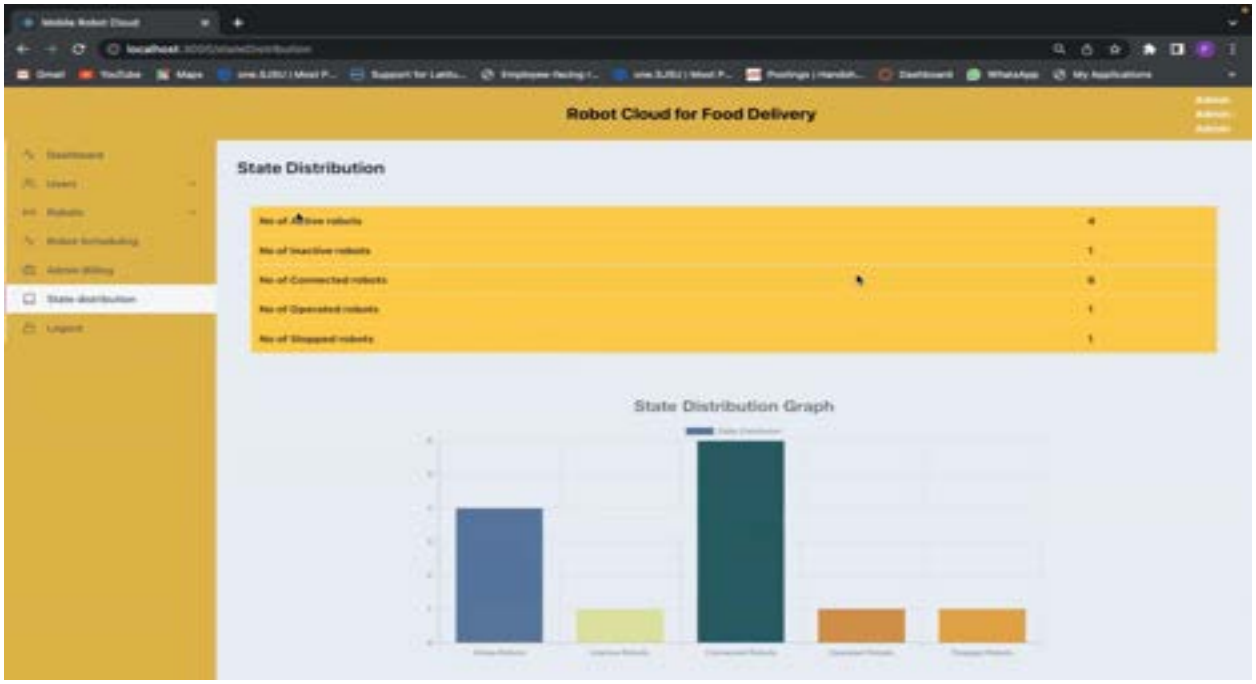

Figure 9.16

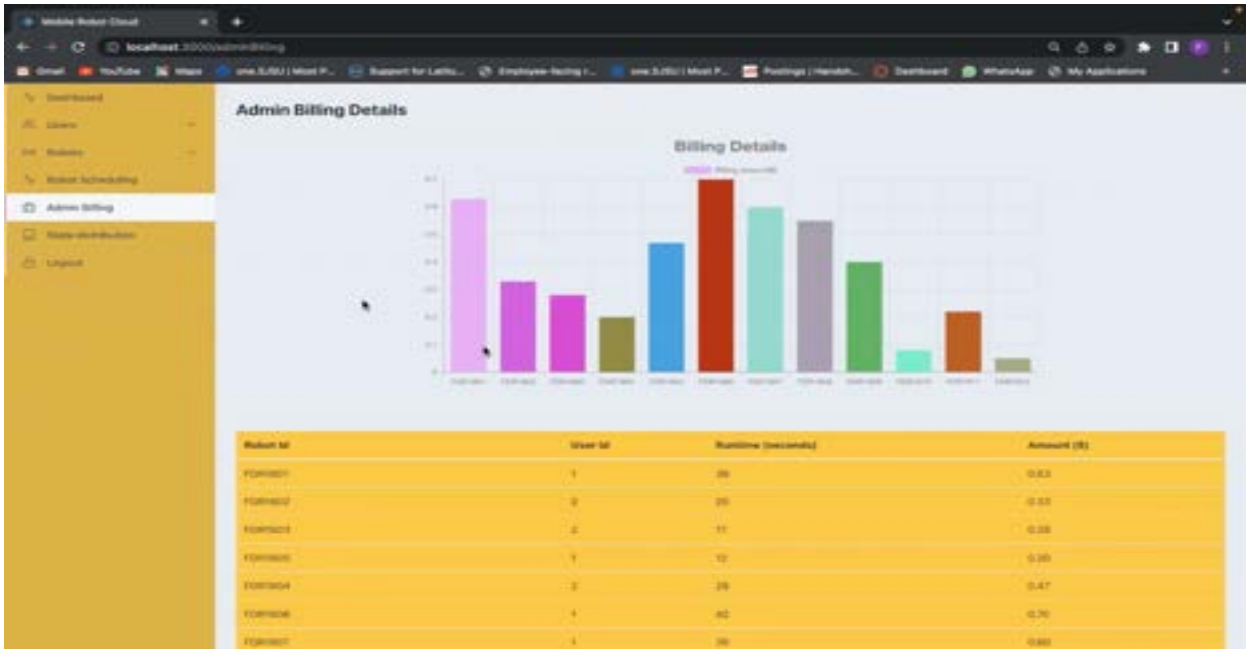

Figure 9.17

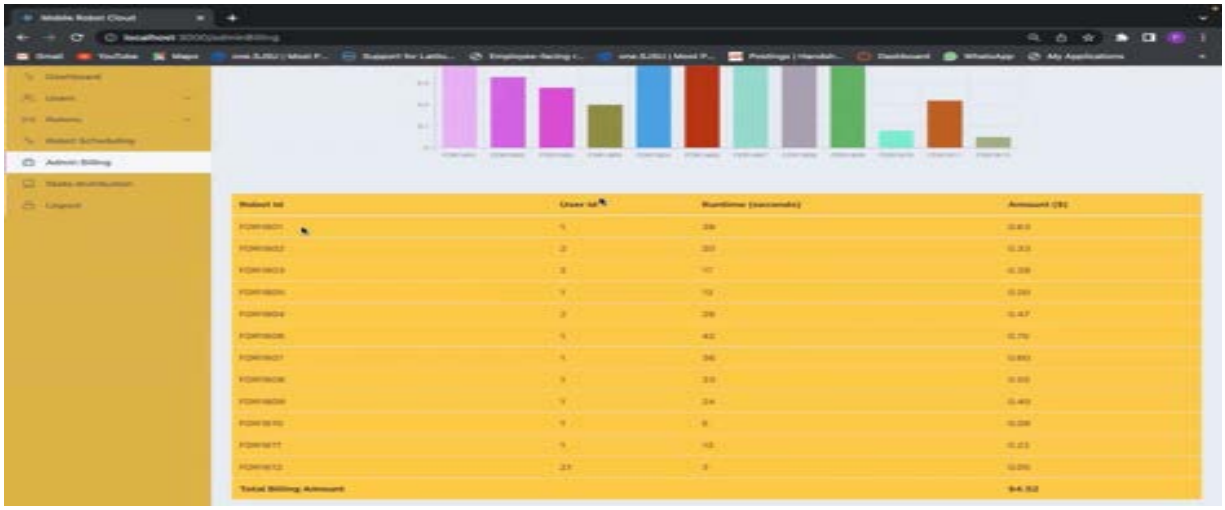

Figure 9.18

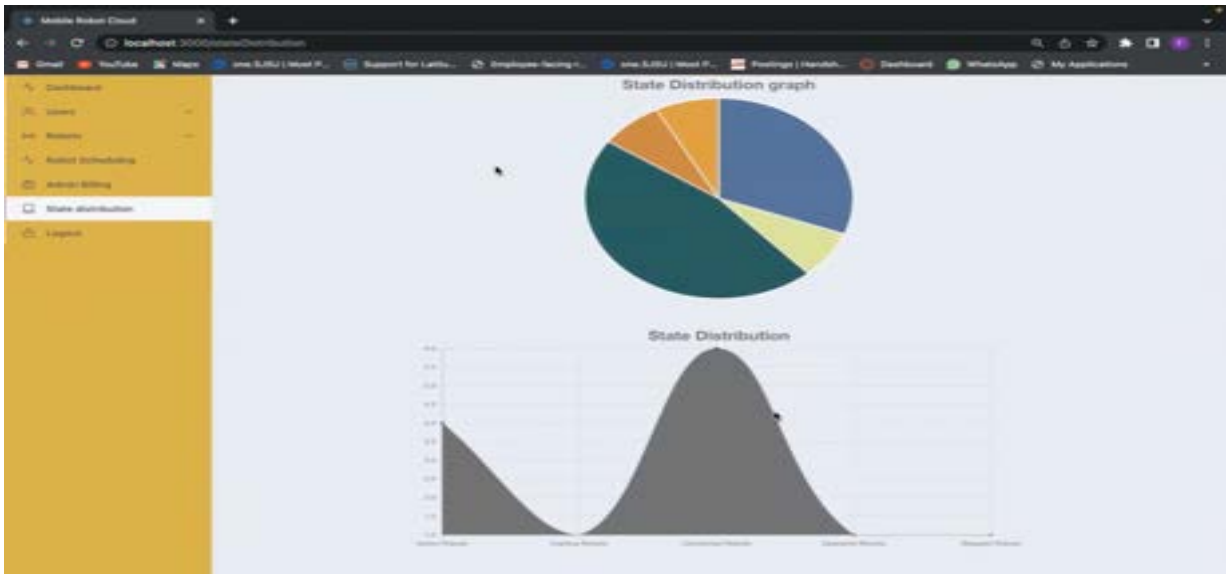

Figure 9.19

10.System Performance Evaluation and Experiments (Extra points)

- Application performance is critical when dealing with systems that manage enormous volumes of data, as in our situation. To determine how well our application worked, we ran the following tests.
- We deliberately manually terminated our Amazon EC2 instance, and the auto-scaling system quickly established a new instance.
- Due to charges being applied, we have only established a maximum of 2 instances as of now.

11. Conclusion - Experience and Lesson Learned

Webots cloud robot simulation, other cloud services, databases, web UI, and backend are only a few of the technologies that are used in the cloud robotics project. All these technologies are included in the experience gained in constructing such a system. Using WeBots, we gained knowledge about the cloud services offered by AWS. Additionally, the three-tier design of web development involved working with MongoDB and MySQL for the databases, React for the front-end, and NodeJS for the backend. Additionally, cloud services like load balancing based hosting have been developed in order for everything to function on the web.

During the study phase, we were able to investigate the prices of numerous cloud service during the study phase. We were able to create suitable billing services since we understood the pricing strategy. We also evaluated the different configurations offered for the service devices and looked at resource use metrics. Therefore, from the perspective of a cloud-based development environment, it has been a terrific project experience.

References

[1] Ghotkar, M. T. (2020). A survey on cloud robotics and automation. International Journal for Research in Applied Science and Engineering Technology, 8(6), 1952-1955. https://doi.org/10.22214/ijraset.2020.6319

[2]. Bogue, R. (2017). Cloud robotics: A review of technologies, developments and applications. Industrial Robot: An International Journal, 44(1), 1-5. https://doi.org/10.1108/ir-10- 2016-0265

[3]. Wensing, P. M., & Slotine, J. (2018). Cooperative adaptive control for cloud-based robotics. 2018 IEEE International Conference on Robotics and Automation (ICRA). https://doi.org/10.1109/icra.2018.8460856

[4]. Liu, Y., & Xu, Y. (2019). Summary of cloud robot research. 2019 25th International Conference on Automation and Computing (ICAC). https://doi.org/10.23919/iconac.2019.8895254

[5]. Technavio. (2022, April 19). Cloud robotics market size to grow by USD 10.11 Bn| 44% of the growth to originate from APAC| Technavio. PR Newswire: press release distribution, targeting, monitoring and marketing. https://www.prnewswire.com/news-releases/cloud-roboticsmarket-size-to-grow-by-usd-10-11-bn-44-of-the-growth-tooriginate-from-apac-technavio-301527213.html

[6]. Mobile cloud computing. (2016). Mobile Cloud Computing, 77- 104. https://doi.org/10.1201/b19208-9 [7]. Kamei, K., Nishio, S., Hagita, N., & Sato, M. (2012). Cloud networked robotics. IEEE Network, 26(3), 28-34. https://doi.org/10.1109/mnet.2012.6201213

[8]. Cloud-enabled smart enterprises! (2012). Cloud Enterprise Architecture, 1-40. https://doi.org/10.1201/b13088-2

[9]. Galambos, P. (2020). Cloud, fog, and mist computing: Advanced robot applications. IEEE Systems, Man, and Cybernetics Magazine, 6(1), 41-45. https://doi.org/10.1109/msmc.2018.2881233

[10]. Du, Z., Yang, W., Chen, Y., Sun, X., Wang, X., & Xu, C. (2011). Design of a robot cloud center. 2011 Tenth International Symposium on Autonomous Decentralized Systems. https://doi.org/10.1109/isads.2011.36