# CMPE 281 Robot Cloud Project Final Reort

Chirag Arora
*MSSE, SJSU*
*chirag.arora01@sjsu.edu*
*016726567*

Farazuddin Mohammad
*MSSE, SJSU*
*farazuddin.md@sjsu.edu*
*016176836*

FNU Butul Parveen
*MSSE, SJSU*
*butul.parveen@sjsu.edu*
*015918227*

Kanupriya Agrawal
*MSSE, SJSU*
*kanupriya.agrawal@sjsu.edu*
*016099057*

***Abstract*— The Food Delivery Robot Cloud is a software as a service (SaaS) that we created and put into use. AWS (Amazon Web Services) cloud technologies, such as EC2 (Elastic Computing Cloud), Auto Scaling Groups, VPC (Virtual Private Cloud), RDS (Relational Database Service), and ELB, are used to fully host the software as a web application (Elastic Load Balancing). Public users and administrative personnel are both supported by Robot Cloud. Robots with support for food delivery, robot position monitoring, controlling, and status updates are available for leasing to the general public. Administrative staff has the ability to remotely track and monitor connected robots online. Robot Cloud offers the groundwork for what might develop into a legitimate autonomous robot rental service and product.**

**Keywords — Rental, Autonomous, Robot, Cloud, Simulation, Webots, Amazon, Web, Services, AWS**

- **Github**
  **https://github.com/Butulparveen/281_robot_cloud**

## I. 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 takeaway 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 rush 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 [7].

### A. Purpose

To build a Cloud-based autonomous food delivery robot service system using the Webots simulator.

### 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.

### C. Market Analysis

The delivery robots' 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 – 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, worldwide growth of the e-commerce market are factors propelling the growth of the delivery robot market [1].

## II. Related Work

With the help of an open-source robotics framework called Robo Earth, robots can share their knowledge via a www-style database and gain access to robust cloud robot services. Via its page for software components, RoboEarth made the manual and the source code available. With the help of the ROS Java library, Android phones can now run ROS. A robot can communicate with just one ROS environment using ROS bridge, a cloud-based platform. The DAvinCi Project showed the advantages of cloud computing by employing a Hadoop cluster to parallelize a SLAM algorithm [6]

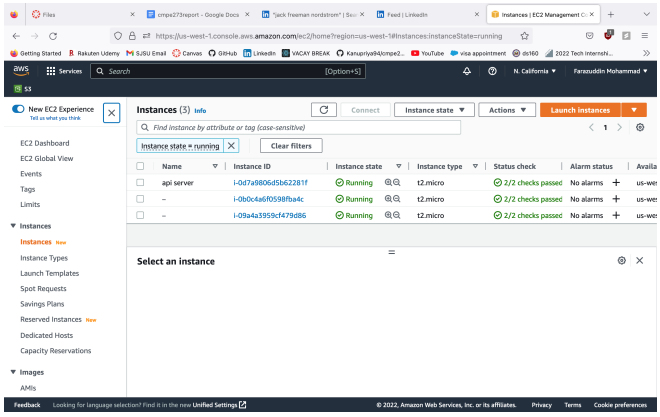

Fig. 1. EC2 Instances

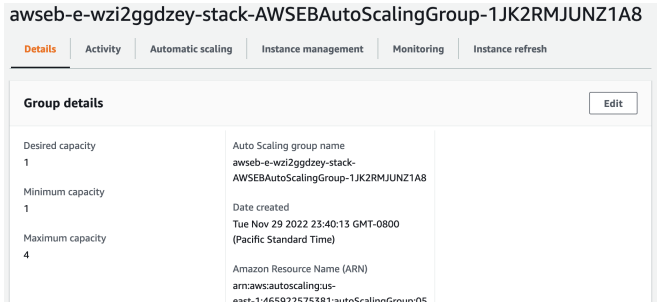


Fig. 2. Autoscaling 1

## III. Cloud-Based System Infrastructure and Component

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 [11].

### 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 [5]. 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.

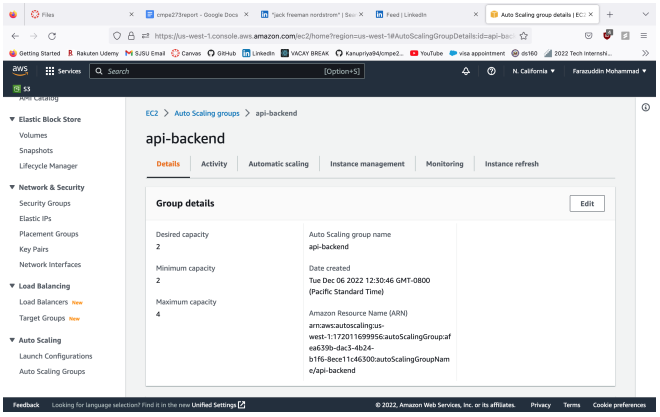

Fig. 3. Autoscaling 2

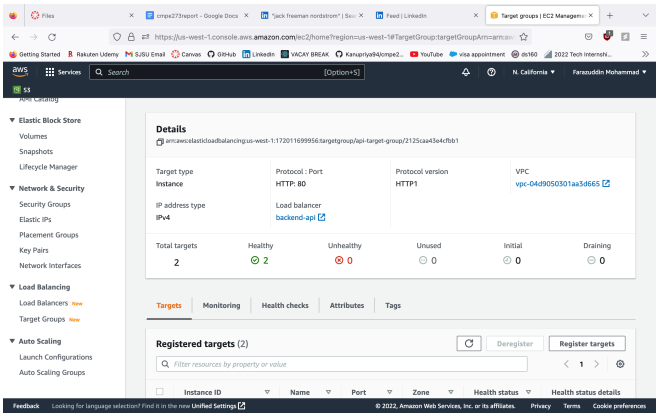

Fig. 4. Virtual Private Cloud

*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.

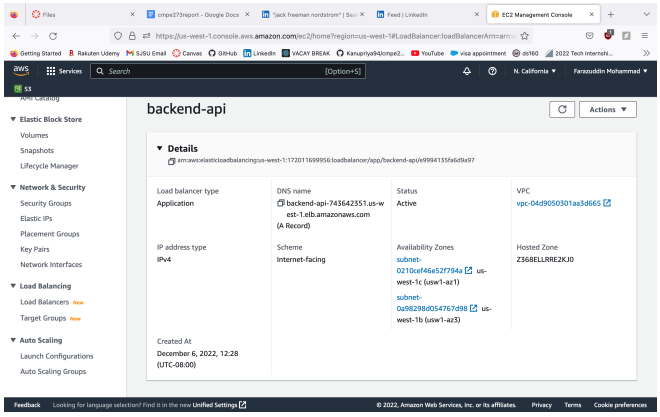


Fig. 5. Elastic Load Balancer

### 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. [3] 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 [4]. Our MongoDB collections were also hosted on an EC2 machine. Using inbound rules, data storage and retrieval from MongoDB were made possible for traffic from our dashboard and simulators API apps. Fig .

## IV. Simulator Design and Implementation

### A. Simulator Design and Implementation

The Food Delivery Robots that are scheduled in the Robot Cloud web service are simulated using Webots.[8] 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.[2]

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. 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 [2]. .

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.[9]

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.

### 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 Fig. 9.

## V. Cloud Data Design and Implementation

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 [10].

The project will use 2 main Database types:

- AWS RDS (MySQL) – This Database will store and manage the 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 Webots simulator.

### 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

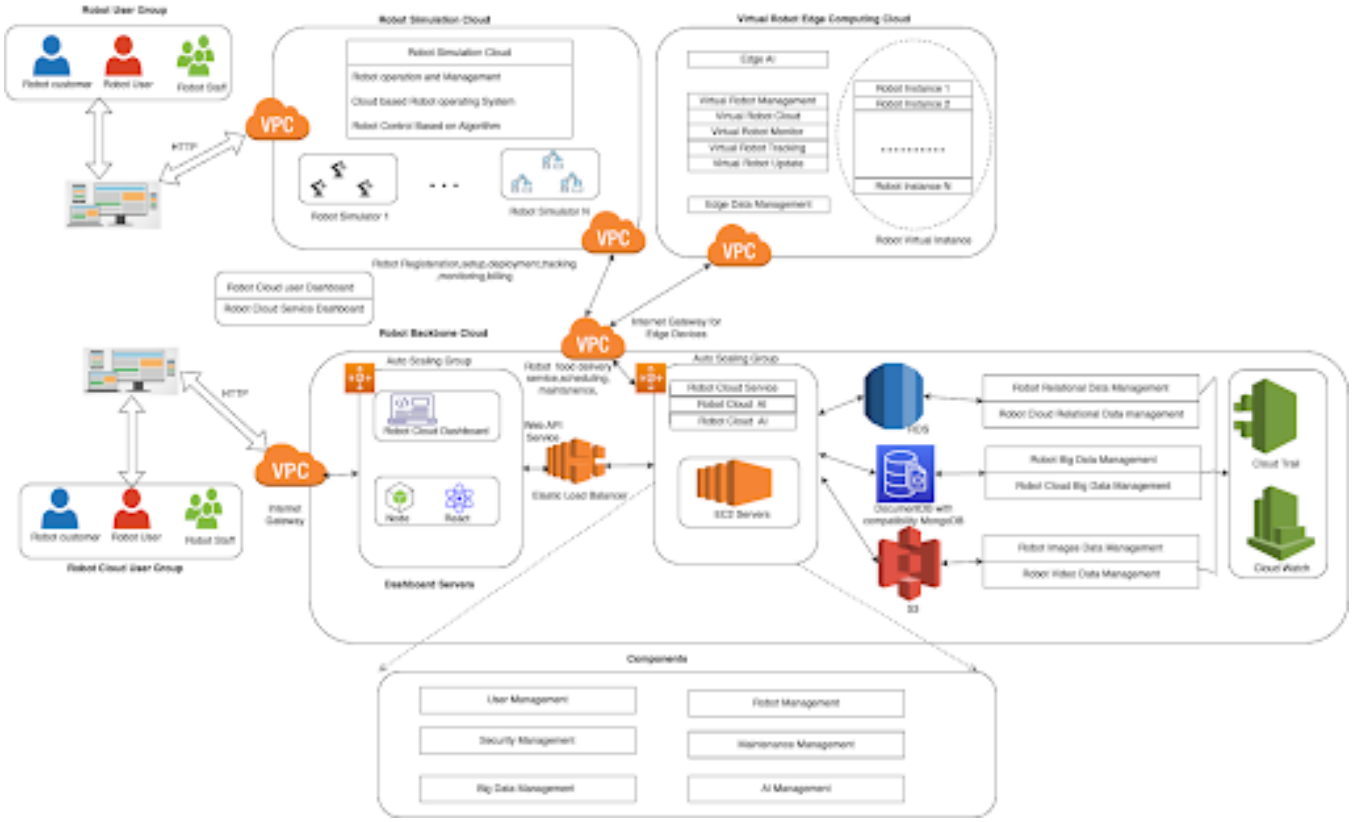


Fig. 6. Overall Architecture Diagram

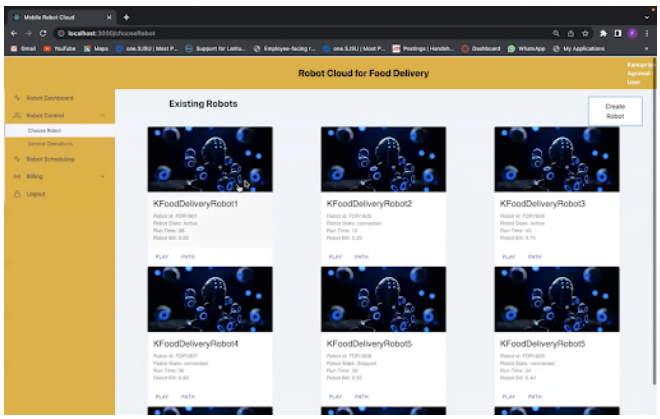


Fig. 7. All Robots Page

attributes and the entities is skillfully depicted in a single diagram Fig. **??**.

h!

- users – user˙id, username, password, name, email, address, role
- robot˙schedule - schedule˙id, robot˙id, user˙id, run˙time, hotel˙id, floor˙id, table˙id,
- billing˙details - billing˙id, robot˙id, user˙id, run˙time, created˙time

With the aid of AWS RDS, this MySQL Database is hosted in the cloud, enabling remote access to the cloud and supplying 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 scheduling 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 Fig. 13 Robot collections will be the sole schema in MongoDB. It includes details about the robots. The robot Path schema contains an array of the coordinates it has already traveled through. No matter if it is active or suspended, robot 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 Fig 21..

## VI. System Function Component Services

### *A. 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

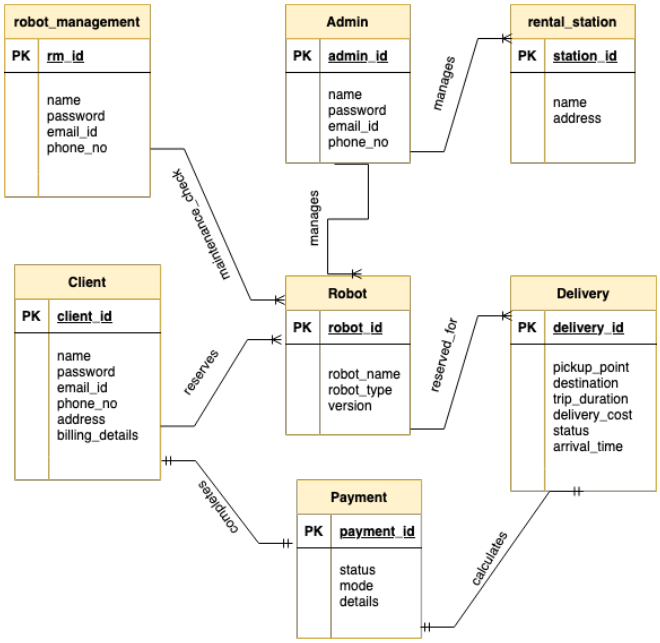


Fig. 8. Entity Relationship Diagram

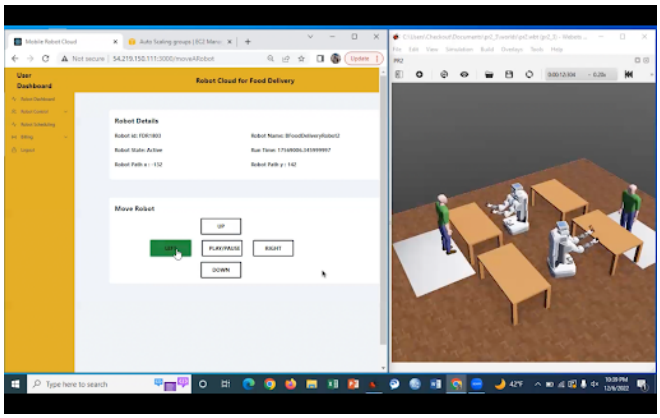

Fig. 9. Webots Simulator

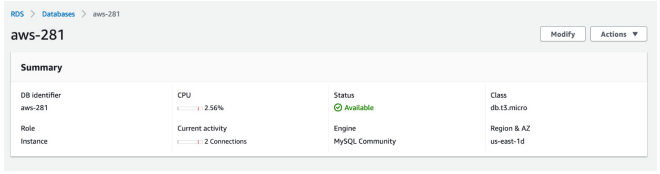


Fig. 10. RDS MySQL Database

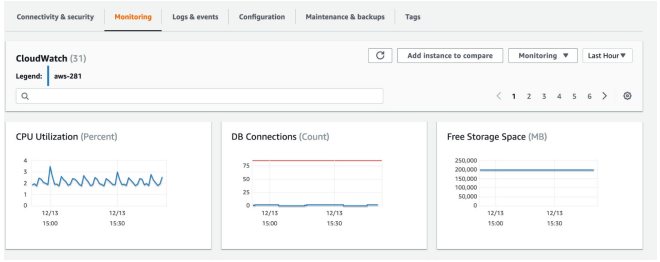


Fig. 11. RDS Monitoring

| URL | HTTP Method | Description |
|---|---|---|
| /users | GET | Get all Users |
| /users/{username} | GET | Get a user |
| /users | POST | Create a user |
| /users/{username} | DELETE | Delete a user |
| /users/{username} | PUT | Update a user |

Fig. 12. User Api Table

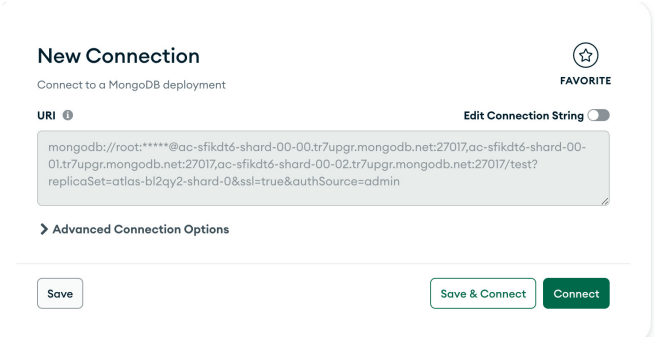


Fig. 13. MongoDb Atlas

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.

### *B. Database management component*

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.

### *C. 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.


Fig. 14. MongoDB Robot Collection


Fig. 15. MongoDB Robot Path

### D. Remote online robot tracking and controlling

The built graphic user interface can be used to monitor and manage the simulated robot. 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. .

### E. 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.

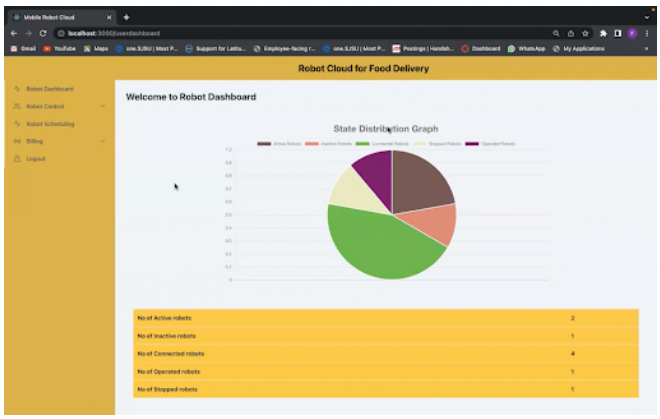


Fig. 16. Usge Statistics Page

## VII. 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. n 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.

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

*1) 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.

*2) 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.

*3) 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.

*4) 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 Webots 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.

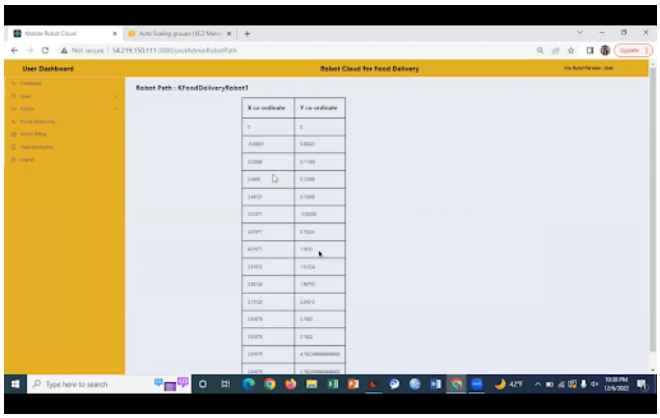

Fig. 17. Robot Tracking and Monitoring

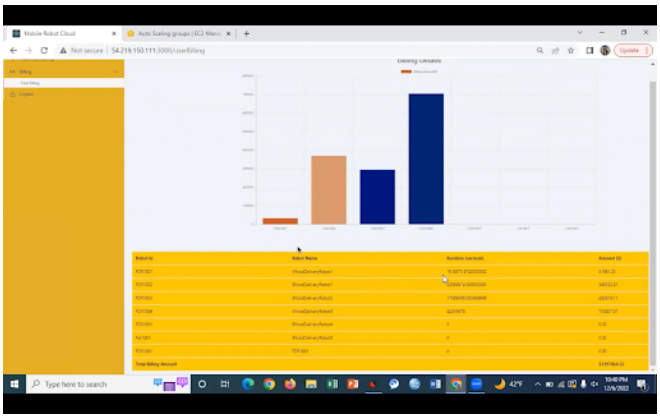

Fig. 19. Admin Dashboard

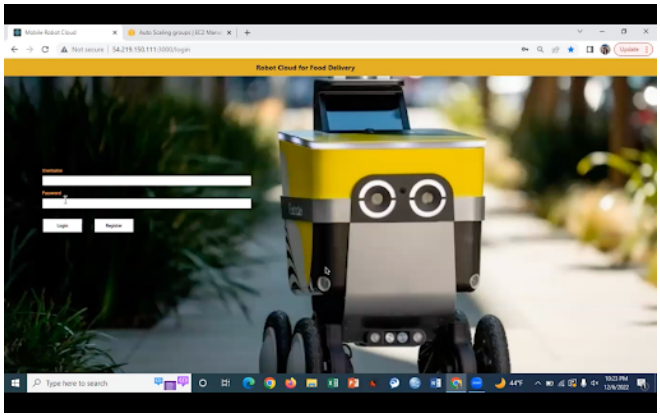

Fig. 18. Robot Scheduling Page

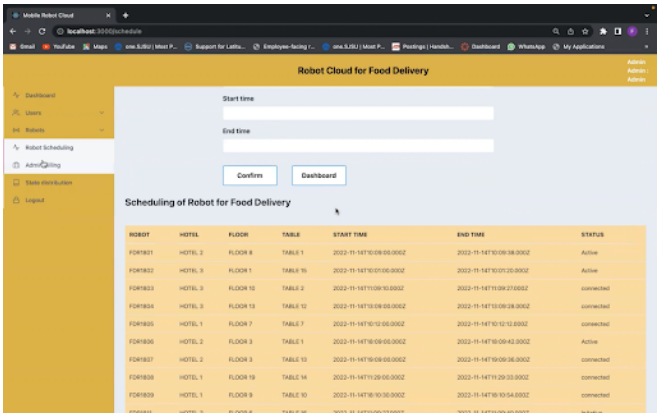

Fig. 20. Robot Tracking and Monitoring

*5) 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.

### *B. 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.

In our scenario, we defined 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.

### *C. System Load Balance Design and Implementation*

We employed 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.

## VIII. System GUI Design and Implementation

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. User: On the user website platform, user has the following alternatives to pursue: h!

- 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.

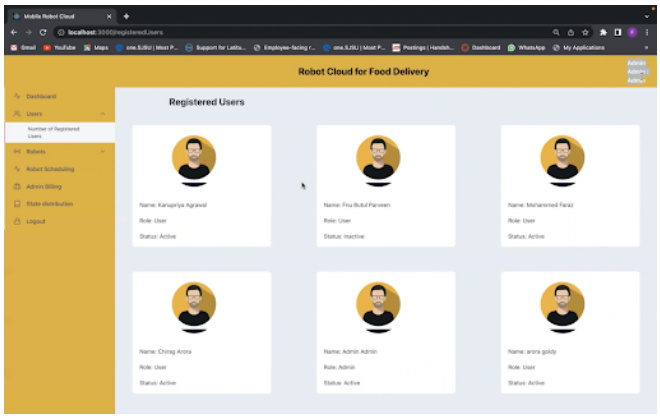


Fig. 21. All Users Page

- 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:

## IX. 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. Webots 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.

## X. System Application Examples

### A. Scenario-based system application example

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 program is employed. Think such a situation where a user wishes to emulate a robot that can move about and follow its path. An admin 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.

## XI. 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.

### A. Auto Scaling

- 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.

### B. Load Balancing

- We selected the application based on the received network requests and aimed to send a lot of request all at once to add sensors in order to test the app's use of Amazon load balancing.
- Using elastic load balancing, incoming traffic is automatically split up among several targets.
- Using ELB, you may monitor applications in real time. It was clear from the logs that the calls were just being routed to several instances.

### C. Scalability

- In order to test the system's scalability, we developed a script that would allow hundreds of sensors to be added from the backend while GUI tasks were being performed by the infrastructure and warehouse user.
- The application's user interface (UI) worked flawlessly, not even a little slowly.
- Multiple Platforms: To test the application's performance, we ran it on many browsers.

## XII. Conclusion - Experience and Lessons Learned

The cloud robotics project integrates a number of different technologies, including: cloud robot simulation (AWS), other cloud services, databases, web user interfaces, and backend. All of these technologies have been used in the system development process. Using Webots, we gained knowledge of Robot Operating System and simulation.

Working on React for the front-end, NodeJS for the back-end, and MongoDB and MySQL for the databases were all included in the three-tier architecture of web development. The implementation of cloud services like load balance-based hosting has also been made necessary for things to function on the web.

We have been able to investigate the costs associated with various cloud services available on the market during the research phase. We were able to create acceptable billing services thanks to our comprehension of the price strategy. We have also analyzed different configurations offered for the service devices and looked at analytics on resource usage. In light of a cloud-based development environment, it has been a wonderful project experience.

In order to make our project more accessible to potential clients and enable simple use of the services, our group would want to further investigate the incorporation of multi-tenancy features in future work.

## ACKNOWLEDGEMENTS

We acknowledge the support of Professor Dr. Jerry Gao for his guidance for the Robot Cloud project.

## REFERENCES

[1] Robert Bogue. Cloud robotics: a review of technologies, developments and applications. Emerald Publishing Limited, 2017.

[2] Yinong Chen, Zhihui Du, and Marcos Garcia-Acosta. Robot as a service in cloud computing. In *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, pages 151–158. IEEE, 2010.

[3] Zhihui Du, Ligang He, Yinong Chen, Yu Xiao, Peng Gao, and Tongzhou Wang. Robot cloud: Bridging the power of robotics and cloud computing. volume 74, pages 337–348. Elsevier, 2017.

[4] Zhihui Du, Weiqiang Yang, Yinong Chen, Xin Sun, Xiaoying Wang, and Chen Xu. Design of a robot cloud center. In *2011 Tenth International Symposium on Autonomous Decentralized Systems*, pages 269–275. IEEE, 2011.

[5] Péter Galambos. Cloud, fog, and mist computing: Advanced robot applications. *IEEE Systems, Man, and Cybernetics Magazine*, 6(1):41–45, 2020.

[6] Jerry Gao and Dayong Wang. Robot cloud computing: A tutorial-infrastructure, service, needs, and challenges.

[7] Miss Trupti Ghotkar and Mrs Dipalee D Rane. A survey on cloud robotics and automation. 1952.

[8] Koji Kamei, Shuichi Nishio, Norihiro Hagita, and Miki Sato. Cloud networked robotics. volume 26, pages 28–34. IEEE, 2012.

[9] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. volume 12, pages 398–409. IEEE, 2015.

[10] Yi Liu and Yuchun Xu. Summary of cloud robot research. In *2019 25th International Conference on Automation and Computing (ICAC)*, pages 1–5. IEEE, 2019.

[11] Patrick M Wensing and Jean-Jacques Slotine. Cooperative adaptive control for cloud-based robotics. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6401–6408. IEEE, 2018.