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 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.

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

Objectives - 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.

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.

2. System Requirements and Analysis

2.1. Functional Requirements

System Goals:
The primary goals of our projects are as follows:

- Provide an alternative to traditional delivery services, leveraging automation and artificial intelligence. Unlike other robots they aren't humanoids looking like persons, they look like moving boxes.
- Keeps the system Centralized - All forms of data and system operations will be centralized in one location, making it easier to monitor the system and deliver the best possible service to users.
- Handles the system's scalability and reliability features.
- To ensure that robots provide an automated door-to-door service, bringing food or other items bought online to the end-customer as soon as possible, the autonomous food delivery system will respond more quickly to complete the process.
- Provides an interface with minimum downtime and minimal performance effect.

User groups are discussed in detail in the User Perspective diagram of the cloud system.

(a) User group specifications:

Users	Responsibility
User/Customer	Deploy Robot.
Admin	Manage the real-time updation of Robots
Robot Maintenance	Maintain each Robot and Service it.

(b) User group service function specifications

Services	Functions
User/Customer	Deploying Food Delivery Robots.
Admin	Manages all the backend, requests and system.
Fleet maintenance	Takes care of the conditions of every Robot in the system.

2.1.1 Current system overview for Robot Rental Service System

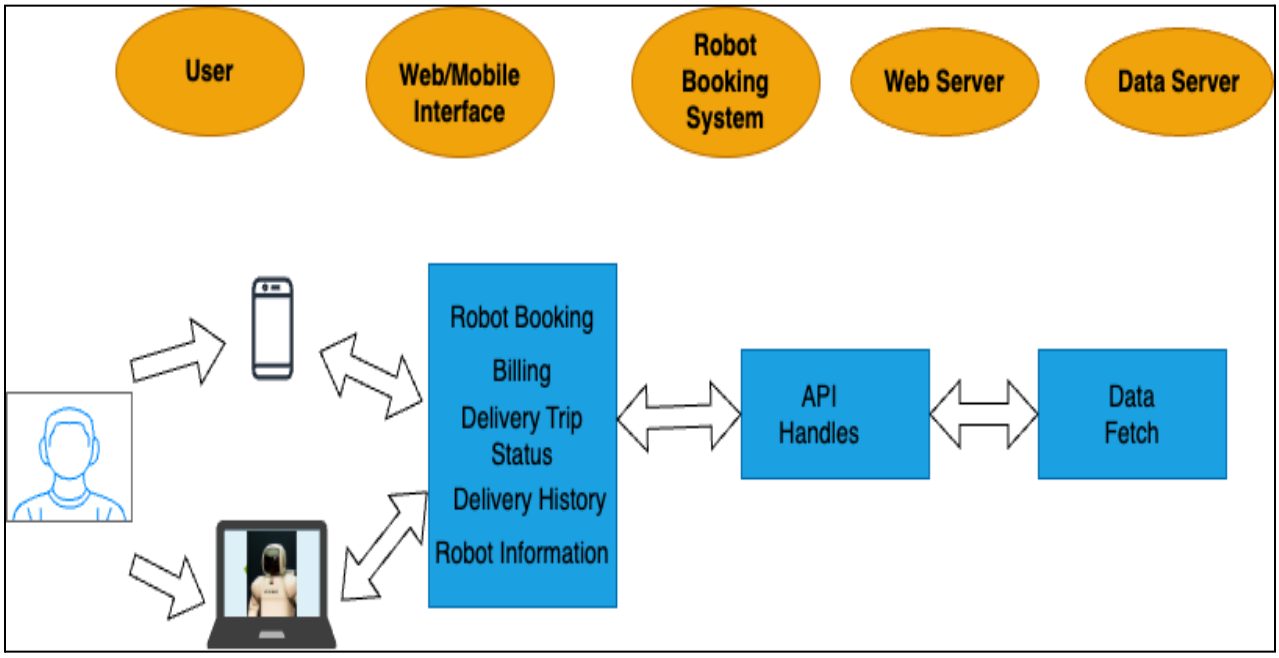


Figure 1 : System Overview

2.1.2 Proposed Model for Robot Service System

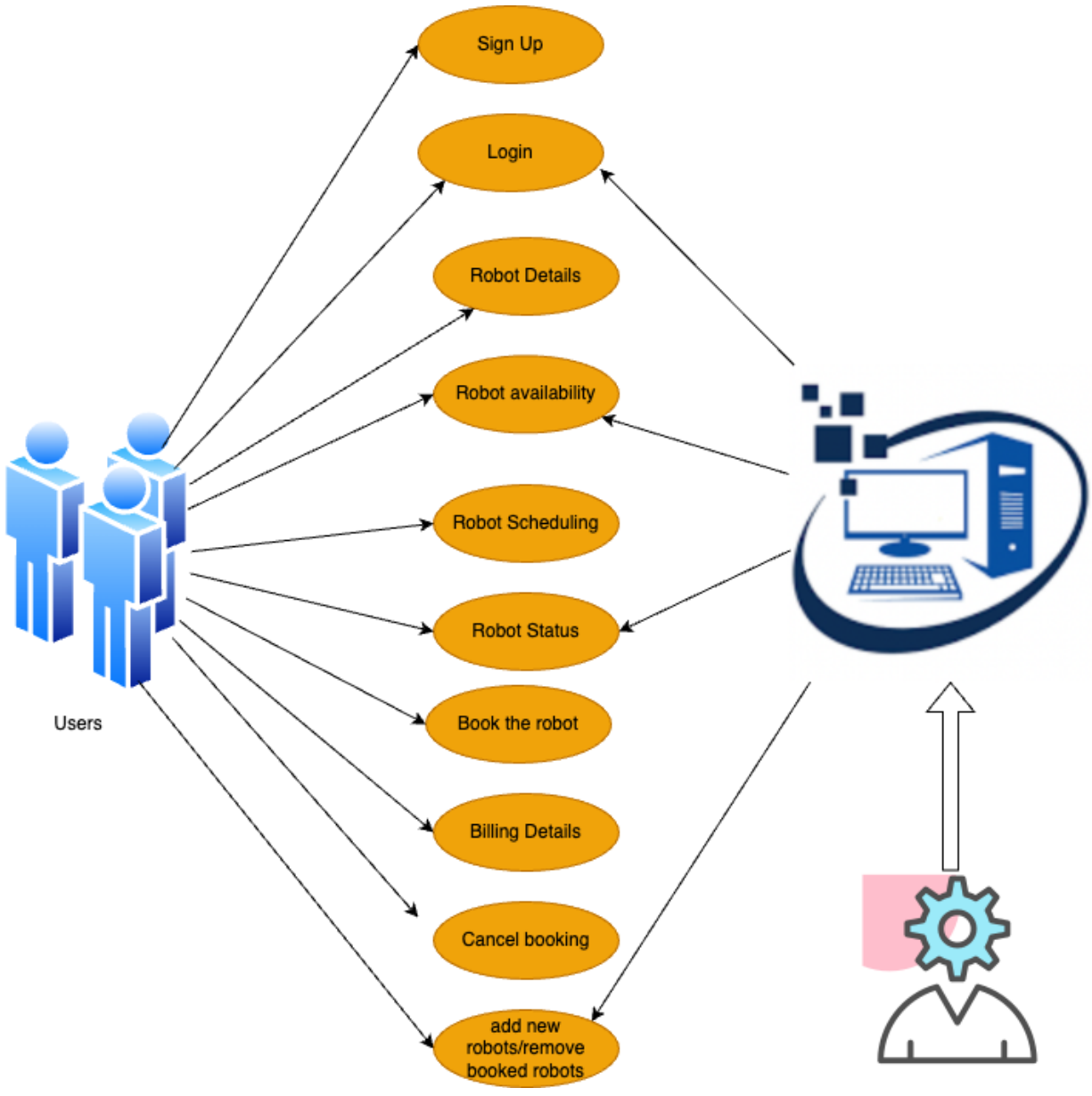


Figure 2 : Proposed Model.

2.2 Cloud System User Scenario analysis

The autonomous food delivery robot service system that we aspire to build has a total of four actors: Admin, customer, robot, delivery.

Below is a class diagram illustrating the system.

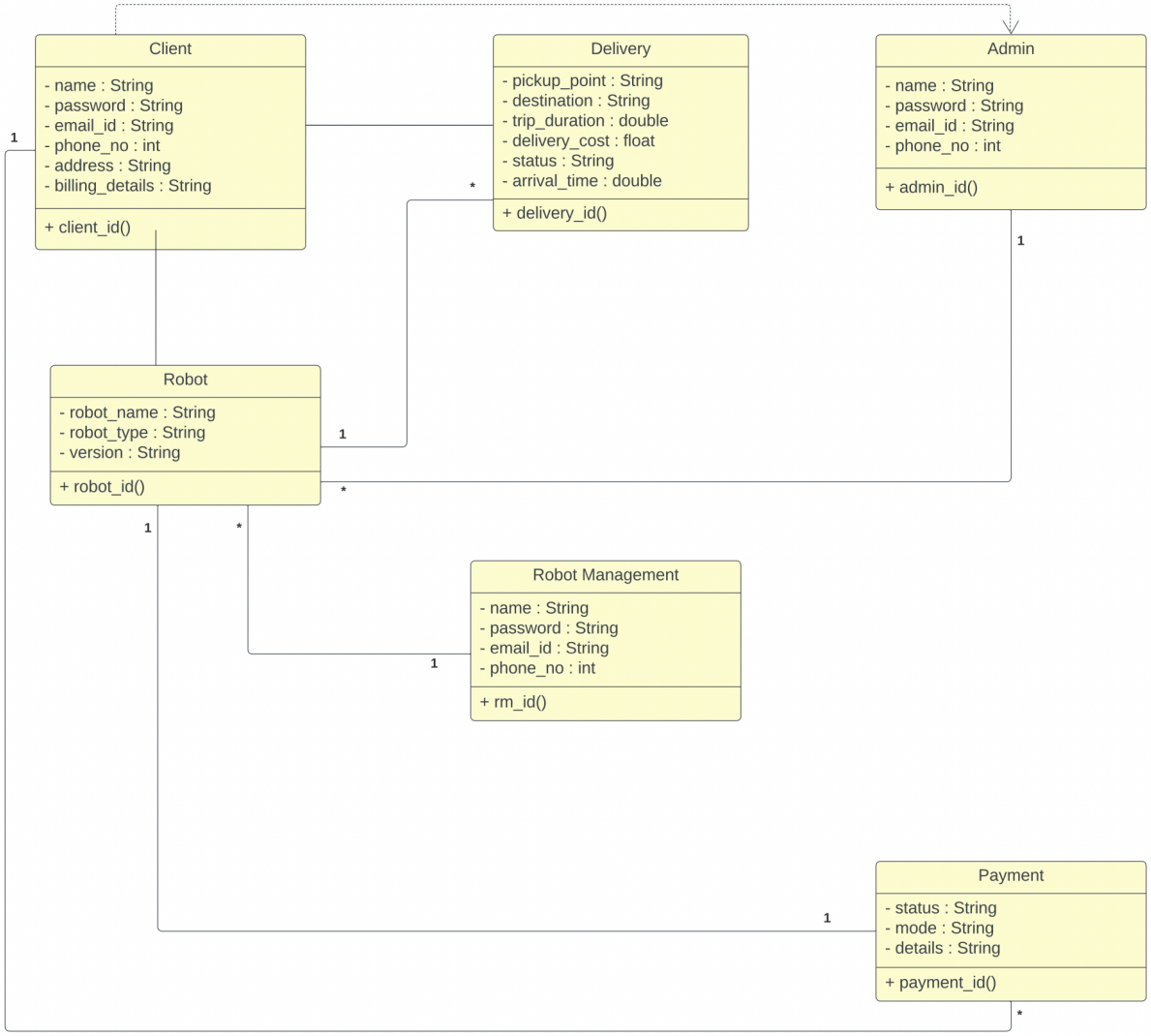


Figure 3 : Class Diagram of a Cloud Based Robot Service System

The system architecture contains the following classes:

Admin	
admin_id : primary key	int
admin_name	String
admin_email	String
admin_password	String
contact_number	String

Robot	
robot_id	int
version	int
robot_type	String

Billing	
billing _id	int
bill_payment_mode	String

Robot Maintenance	
rm_id : primary key	int
rm_name	String
rm_email	String
rm_password	String
contact_number	String

According to the class diagram shown above, the following classes make up the system architecture:

Classes	Responsibilities
Admin	The ability to update the Robot belongs to the Admin.
User/Customer	The user can modify the delivery information.
Delivery	The delivery may be updated by one or more actors.
Robot	All three parties have access to a robot's details, but their updating rights are constrained.
Payment	The system's most crucial component is billing.
Robot Maintenance	Every person involved in robot maintenance has the ability to update and repair the robot.

Role	Responsibilities
User	Enter the pick-up and drop-off locations through text or geolocation.
	Search for all the available robots for rental and select one of the options.
	Manage the booking and access the details of the delivery and the robot.
	Manage billing for the delivery taken.
Admin	View, edit, delete, modify the details of the robots available.
	Have access to all the users' information.
	Manage the locations where the rentals are available.
Robot_maintenance	Add, delete any robot
	Repair/fix a damaged robot.
	Manage the robot in real-time.

Use Case Diagrams

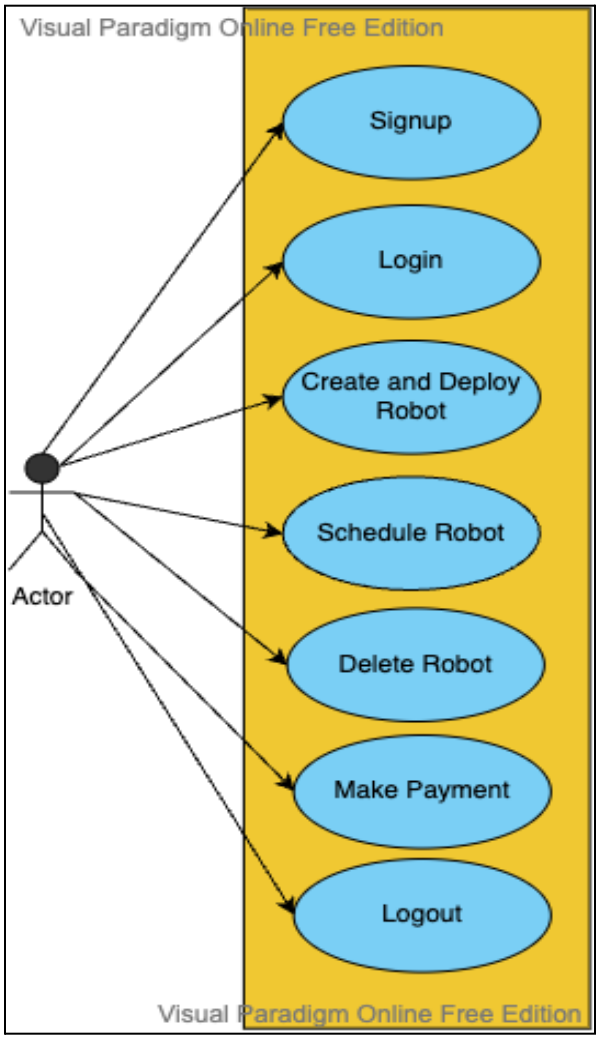


Figure 4 : Admin Use Case Diagram.

The following are the functions handled by the 'Admin' in our Robot Rental System:

- Admins can access the system at any time and log in or log out.
- Admin can access all the relevant user's details and can use them further to their liking.
- Admin can also retrieve the robot's details.
- Once, accessing the data, Admin can add robot(s) and also can delete robot(s).
- Admin is granted the privilege to manage booking.
- Admin can manage billing.
- Admin can Manage Payment Options - adding, deleting and modifying, solving issues in payments.

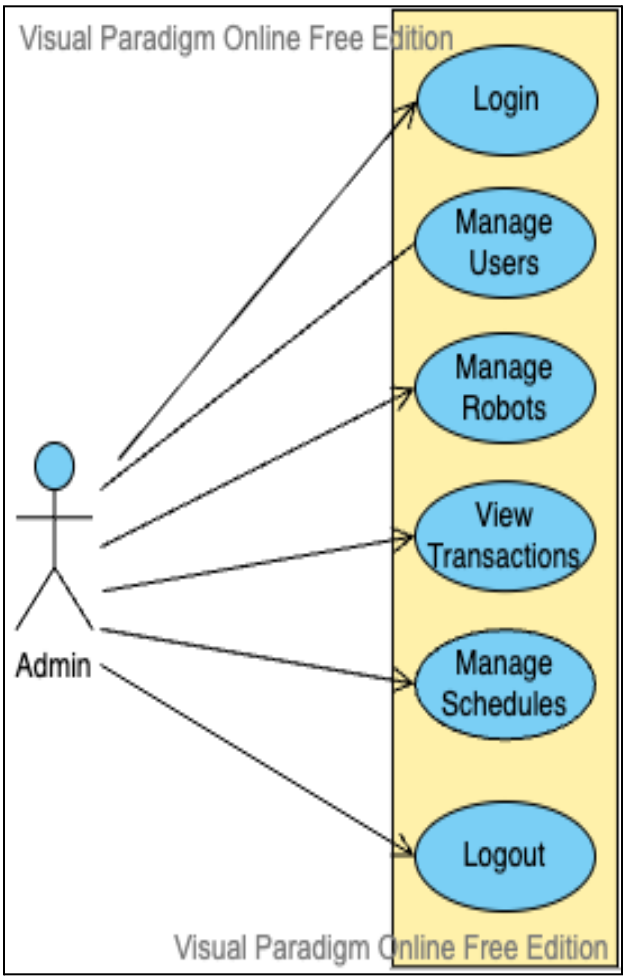


Figure 5 : User Use Case Diagram.

Sequence Diagram:

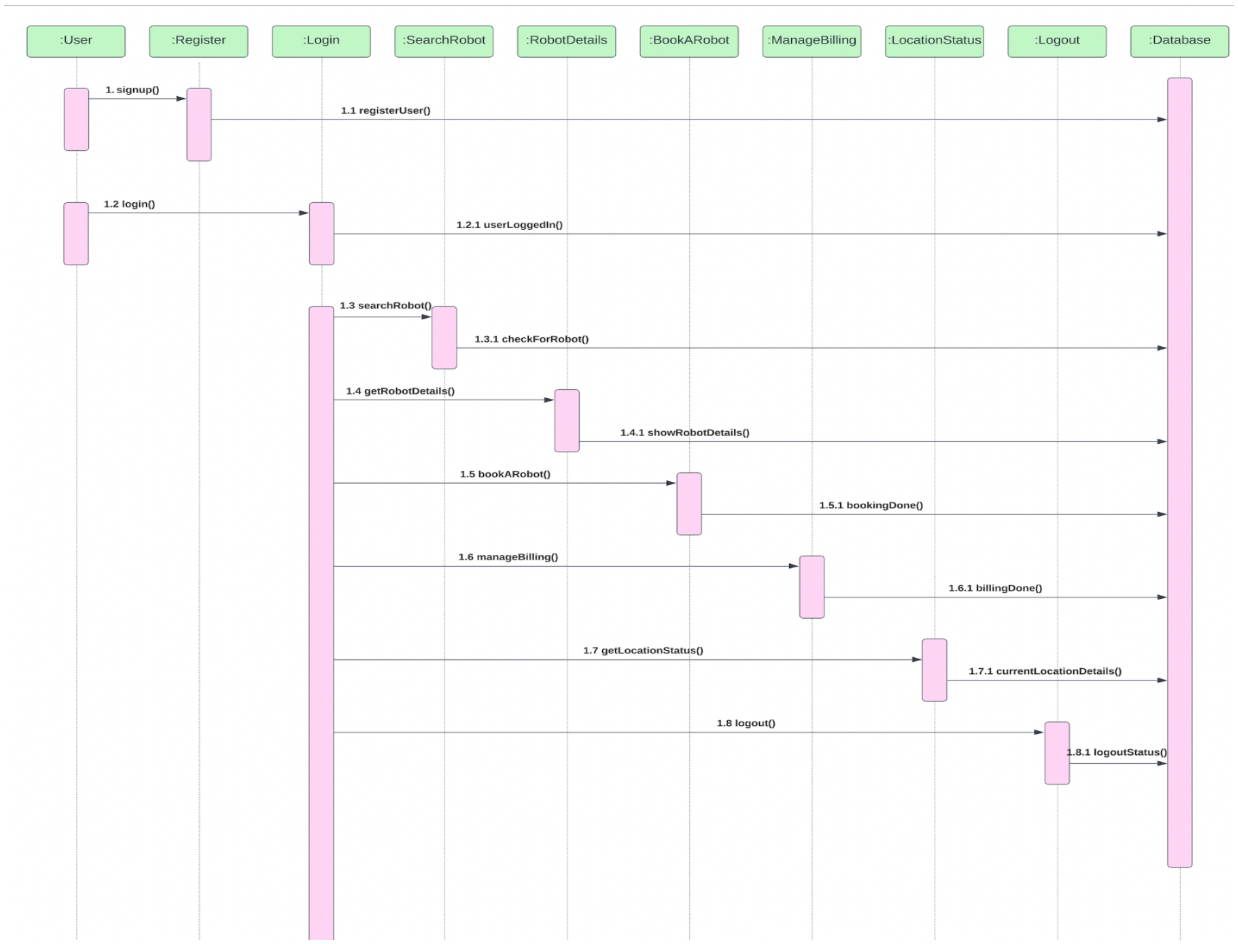
The event chronology from the admin side is depicted in the sequence diagram below. The procedure happens as follows:

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.



- The user logs out at the conclusion of the session.

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.

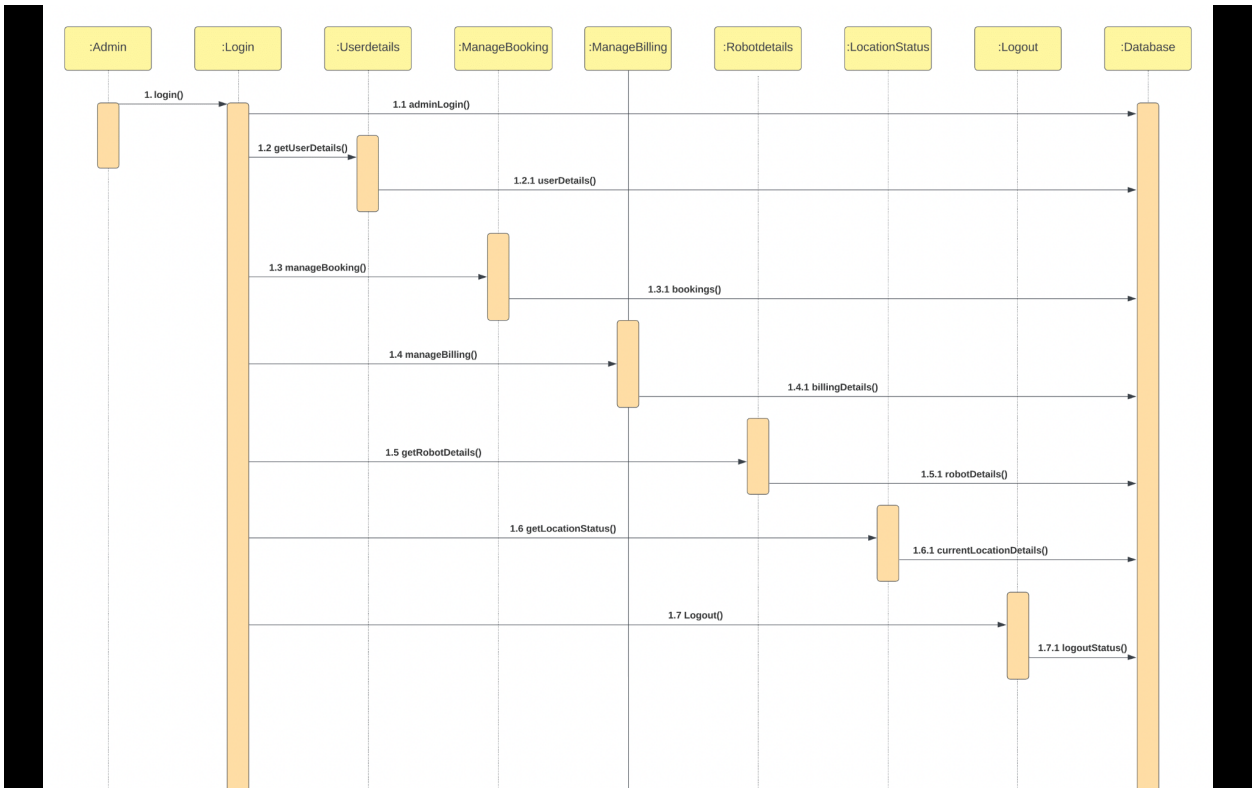


Figure 7 : Admin Side Sequence Diagram

3. System Infrastructure and Architecture

3.1 Cloud-based System Infrastructure

Cloud-based system infrastructure is implemented and deployed based on the architecture of the diagram and infrastructure diagram is represented below, using the following elements in a system:

- Different user groups
- Cloud-based systems and servers
- Data storage and DBs
- Load balance and scalability
- Networking connectivity
- API Gateway

3.1.1 Different user groups:

Our project application is a school campus security robot, so any age group can access it. Basically, here we are considering students, faculty, or any new person's first time visiting a school campus. Here we are considering San Jose State University's campus for our project. For security purposes, any user needs a robot security system. Basically, it charges based on the usage of the robot for billing purposes. After designing the application, it is basically its user-friendly user interface, so anybody can easily access the school campus security robot system application.

3.1.2. Cloud-based systems and servers:

Presentation Server:

The web-based TrueSight console capabilities are hosted by the TrueSight Presentation Server, which also uses data from other TrueSight products to create a consolidated set of views for capacity planning, real and simulated application monitoring, and infrastructure monitoring. The TrueSight Presentation Server also handles tasks like role-based access control and data management tasks like persistence and storage, in addition to data presentation.

TrueSight console:

The Presentation Server serves as the host for the TrueSight console. System administrators set up the TrueSight Presentation Server via the TrueSight console, connect to the data sources, and link the TrueSight Presentation Server with other components and products. Users and user groups defined in the Remedy SSO Admin Console can also be viewed by system administrators

using the TrueSight Console. The TrueSight console is used by IT Operations and other professionals to keep an eye on devices, events, and applications, or to track down issues.

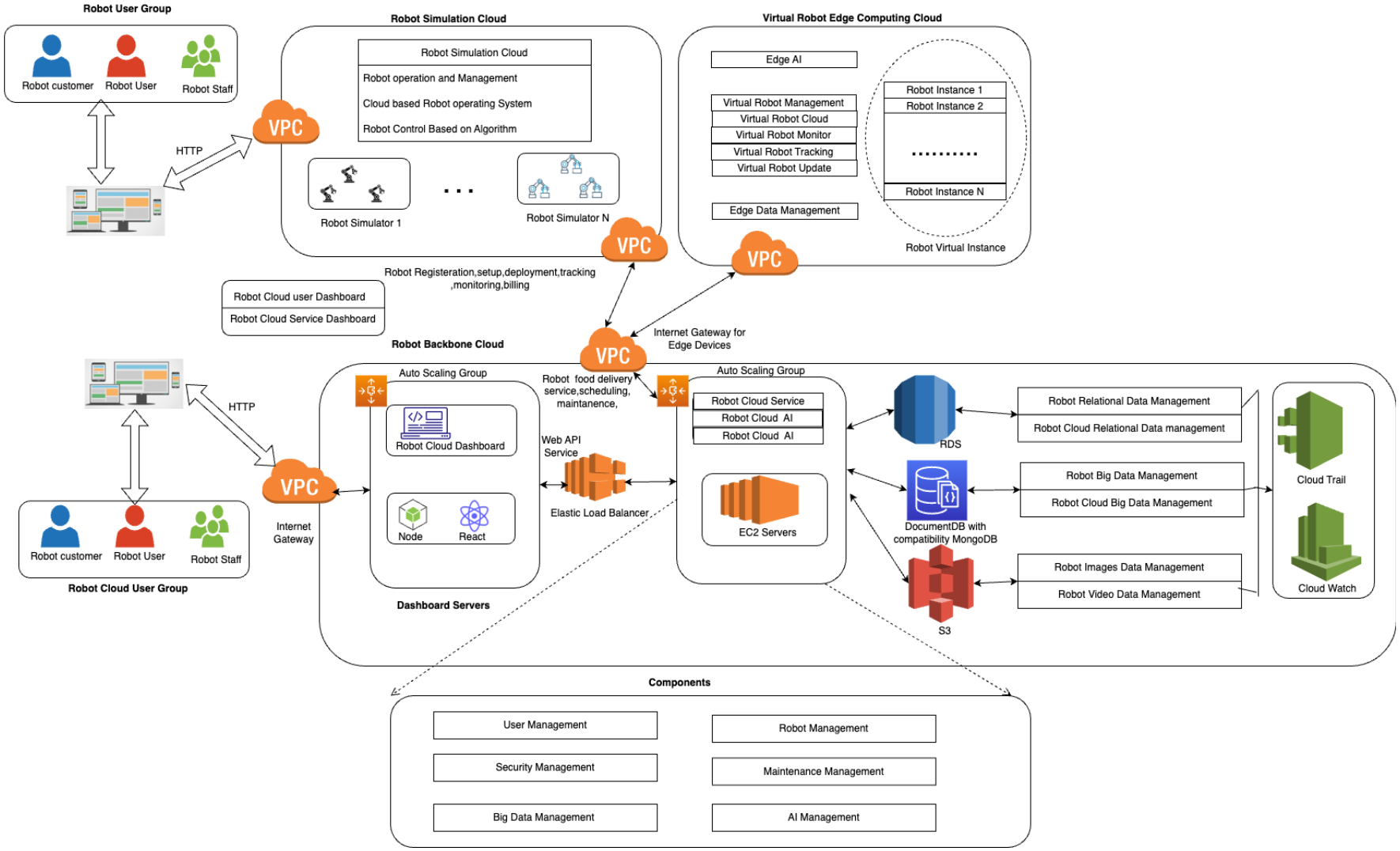


Figure 8 : Cloud-based System Infrastructure for Robot.

Remedy SSO console:

Different users, groups, and roles are created and managed by system administrators using the Remedy SSO Admin Console, which may be viewed on the TrueSight Console.

3.1.3 Data storage and DBs:

Data Storage using in our project Amazon RDS:
A relational database on the AWS Cloud may be set up, run, and scaled more easily thanks to Amazon Relational Database Service (Amazon RDS), a web service. It performs typical database administration activities and offers affordable, resizable capacity for an industry-standard relational database.

- Comparing fully managed database deployments to those using Amazon RDS, the following benefits are particularly advantageous:
- You can use MariaDB, Microsoft SQL Server, MySQL, Oracle, and PostgreSQL, as well as other database programs that you are already familiar with.
- Backups, software patches, automatic failure detection, and recovery are all managed by Amazon RDS.
- You can manually create backup snapshots or enable automated backups. These backups can be used to restore a database. The Amazon RDS restore procedure operates effectively and dependably.
- With a primary instance and a synchronous secondary instance that you may fail over to in the event of issues, you can achieve high availability. To improve read scaling, you may also employ read replicas.
- You can help limit who has access to your RDS databases in addition to the security features in your database package. You can accomplish this by defining users and permissions using AWS Identity and Access Management (IAM). Additionally, storing your databases in a virtual private cloud can help protect them (VPC).

The database used in our project, Mongodb:

A document database called MongoDB is used to create scalable and highly accessible Web applications. It's well-liked by agile development teams due to its flexible schema approach.

3.1.4 Load balance and scalability:

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.

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.

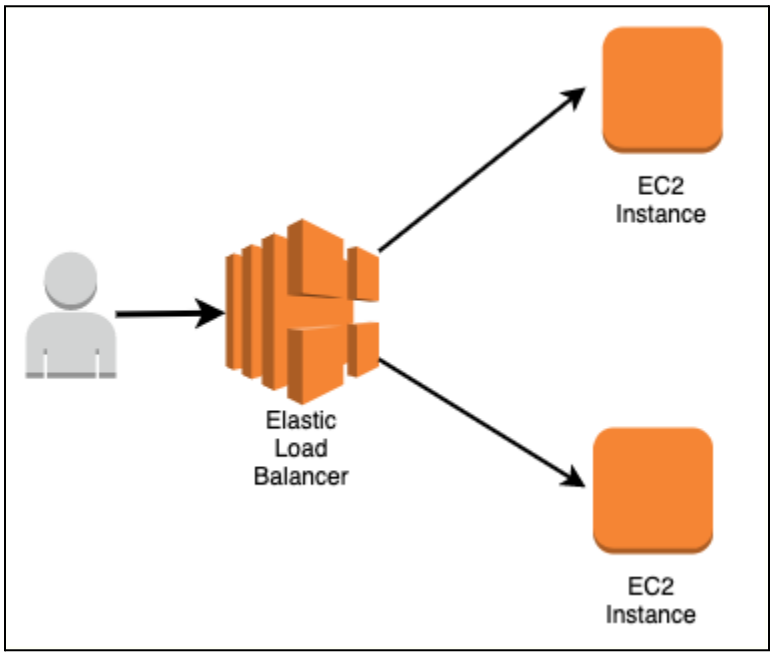


Figure 9 : Elastic Load Balancer.

When a new instance needs to be created, the metrics need to be configured. 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.

3.1.5 Networking connectivity:

RoboMaker's cloud-based simulation service, without having to worry about managing any infrastructure, may perform, scale, and automate simulation with the help of AWS.
The reachability analyzer is located in the VPC Management Console's left navigation. In the new windows that appear after selecting the Reachability Analyzer and the Create and Analyze Path buttons, you may enter a path between a source and destination and begin analysis.

Amazon VPC:

For the databases, we'll set up our own virtual private network (VPC) on Amazon. This will give us complete control over the infrastructure of the virtual network, including resource allocation, connectivity, and encryption.

CloudFront:

AWS CloudFront is a global network provided by Amazon Web Services that provides customers with speedy and safe access to applications, development tools, video, and other types of content.

Route S3:

An easy-to-use and scalable DNS web service is Route 53. Users can be forwarded to other AWS services like SNS, Elastic Beanstalk, or Amazon S3 buckets.

Elastic Beanstalk:

You can quickly install and maintain apps in the AWS Cloud using AWS Elastic Beanstalk without worrying about the infrastructure that supports them. Elastic Beanstalk makes management easier while preserving flexibility and control. Elastic Beanstalk applications are logical collections of Elastic Beanstalk components, such as environments, versions, and environment configurations. Depending on the modification, it either deletes old resources and deploys new ones when the configuration settings for an environment are updated.

SNS:

A cloud-based messaging service for A2A (application-to-application) and A2P (application-to-person) communication is called Amazon Simple Notification Service (Amazon SNS). It will be used to push notifications and messages to users.

Amazon S3 Bucket:

The generated log files as well as the application files will be kept in an Amazon S3 bucket.

RDS:

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.

DynamoDB:

A key-value NoSQL database designed to run high-performance applications at any scale, Amazon DynamoDB is cloud-hosted and fully managed. We need payment providers to process transactions quickly since we want to maintain data statistics on robots, robot delivery, and billing transactions.

RedShift:

This Amazon web service offers a data warehouse. We chose it because it offers numerous analytical application integrations, including those with Amazon QuickSight and PowerBi. As a result, we will be able to extract data and provide pertinent insights and business information.

CloudWatch:

AWS CloudWatch is the monitoring tool we've selected. A unified view of AWS resources, apps, and software operating on AWS and on-premises web servers is provided by CloudWatch, which gathers and analyzes surveillance and operational data in the form of logs, metrics, and events.

PowerBi:

Power BI is a group of software services, applications, and connections that work together to transform various data sources into logical, immersive visual insights that are also interactive. Your data may be presented as a set of cloud-based or on-premises hybrid data warehouses or as an Excel spreadsheet. Power BI makes connecting to and viewing your data sources simple.

API Gateway:

In order to construct, publish, maintain, monitor, and protect APIs of any size, developers can use the fully managed service known as Amazon API Gateway. Applications use APIs as their "front door" to data, business logic, or functionality in their backend services. You can build WebSocket and RESTful APIs with API Gateway to enable apps with real-time two-way communication. Web apps, serverless workloads, and containerized workloads are all supported by API Gateway.

API Design of the proposed System

API Name	Functionality	Method
register	Register a user for application	POST
login	To login and authenticate a user	POST
registerRobot	Register a robot	POST
getUsers	List of registered users	GET
getAllRobots	A list of all registered robots	GET
getActiveRobots	All live robots listed	GET
getRobotState	learn the robot's condition	GET
getDistributionMetrics	Obtain the robots' distribution metrics.	GET
getNavigatedPath	Obtain the robot's journey since it started.	GET
getRobotLocation	To obtain the robot's location's coordinates	GET
deactivateRobot	Deactivate a robot and remove it from the user	DELETE
getRobotLocation	To obtain the robot's location's coordinates	GET
getBillingDetails	To obtain the user's billing information	GET
getUsageDetails	To enable a user's use of various robots	GET
initiateMap	To start the robot's navigational map	GET
initiateRobot	To start a robot and assign it to a user	GET
moveRobot	Move a robot in a specific direction.	POST

AWS RoboMaker:

We are simulating a mobile robot that can travel to a certain spot on a defined map using the AWS robomaker. The backend sends instructions to the robo maker component, which updates the robot's position on the map using the robo maker SDK. The backend then receives a response from the robomaker SDK in JSON format.

3.2. System Component-Oriented Function Architecture Design

The system component-oriented function architecture is designed using a function-oriented tree diagram and a system functional block diagram. This diagram must present the following items:

- Different function components and their connectivity and dependencies
- High-level and low-level system partitions

Different function components and their connectivity and dependencies.

The system is partitioned into six different categories, such as:

1. Frontend component:
2. Backend Component
3. AWS Robomaker Component
4. Database component
5. Billing services
6. Tracking and monitoring

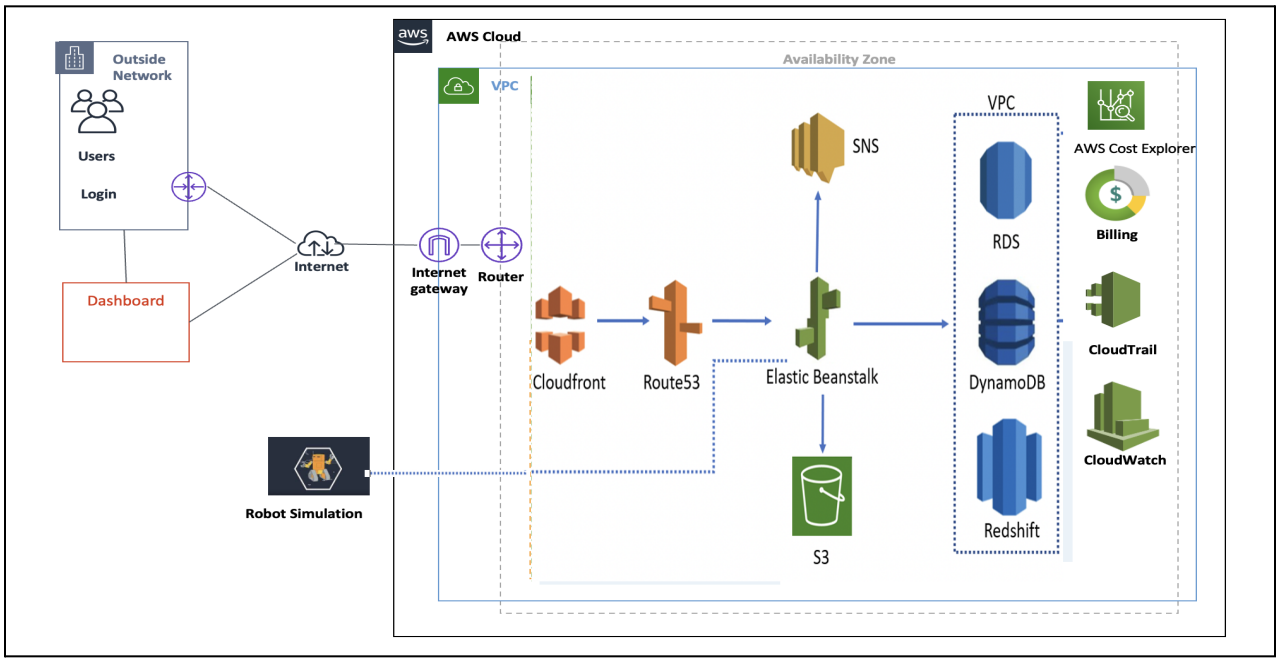


Figure 10 : System component-oriented function architecture design

1. User Interface:

ReactJS is used to build the program's front end, and NodeJS is used to build the back end. Customers can register a robot if you are a new user to apply,schedule a robot, track the robot using simulation, monitor the robot and pay for it using the User Interface. Both mobile phones and computers with higher resolutions, including PCs and laptops, can use our app.

2.Frontend component:

For user interaction, the frontend component only communicates with the backend component. It inputs commands into the backend component and receives the AWS Cloud Builder Robot component's answer in return. The frontend communicates with each other using a REST architecture that is based on HTTP, and sends and receives data in JSON format.

3.Backend Component:

The backend component receives commands from the frontend and communicates with the AWS Cloud Maker component to change the robot's direction. If the interaction is successful, the cloud maker component sends a message with a JSON answer to the backend component. It also communicates with the database component for user access, retrieving and storing billing information, and any other database operations.

4.AWS Robomaker Component:

We are simulating a mobile robot that can travel to a certain spot on a defined map using the AWS robomaker. The backend sends instructions to the robo maker component, which updates the robot's position on the map using the robo maker SDK. The backend then receives a response from the robomaker SDK in JSON format.

5.Database component:

The only component that communicates with the database component is the backend. To insert, delete, update, or fetch the database's data, it makes calls through either Hibernate or MongoSDK.

6.Billing services:

Based on the usage of the robot, the billing services will be charged to the user based on their time duration, either in seconds, hours, days, weeks, months, or years, etc. After the usage of the

school service campus robot, the bill will be generated in the application, and you can track the billing, payment, and services in the portal.Aws billing and budget used for these billing services.

7.Tracking and monitoring:

The usage of the robot admin can track the application and monitor where it is and keep status using cloud trail and cloud watch.

High-level and low-level system partitions:

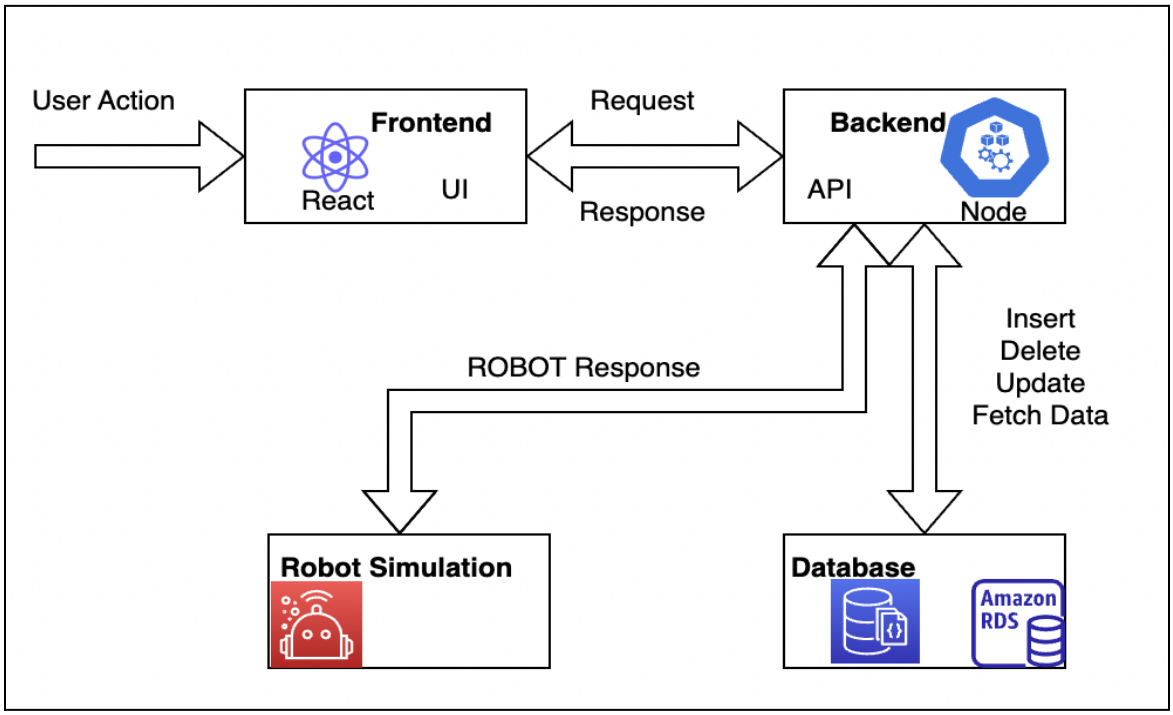


Figure 11: High-level and low-level system partitions

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, delivering the behavior of the robot.

3.3 System Deployment Infrastructure

The above system deployment diagram is used to visualize the hardware, i.e.nodes/devices of a system, 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

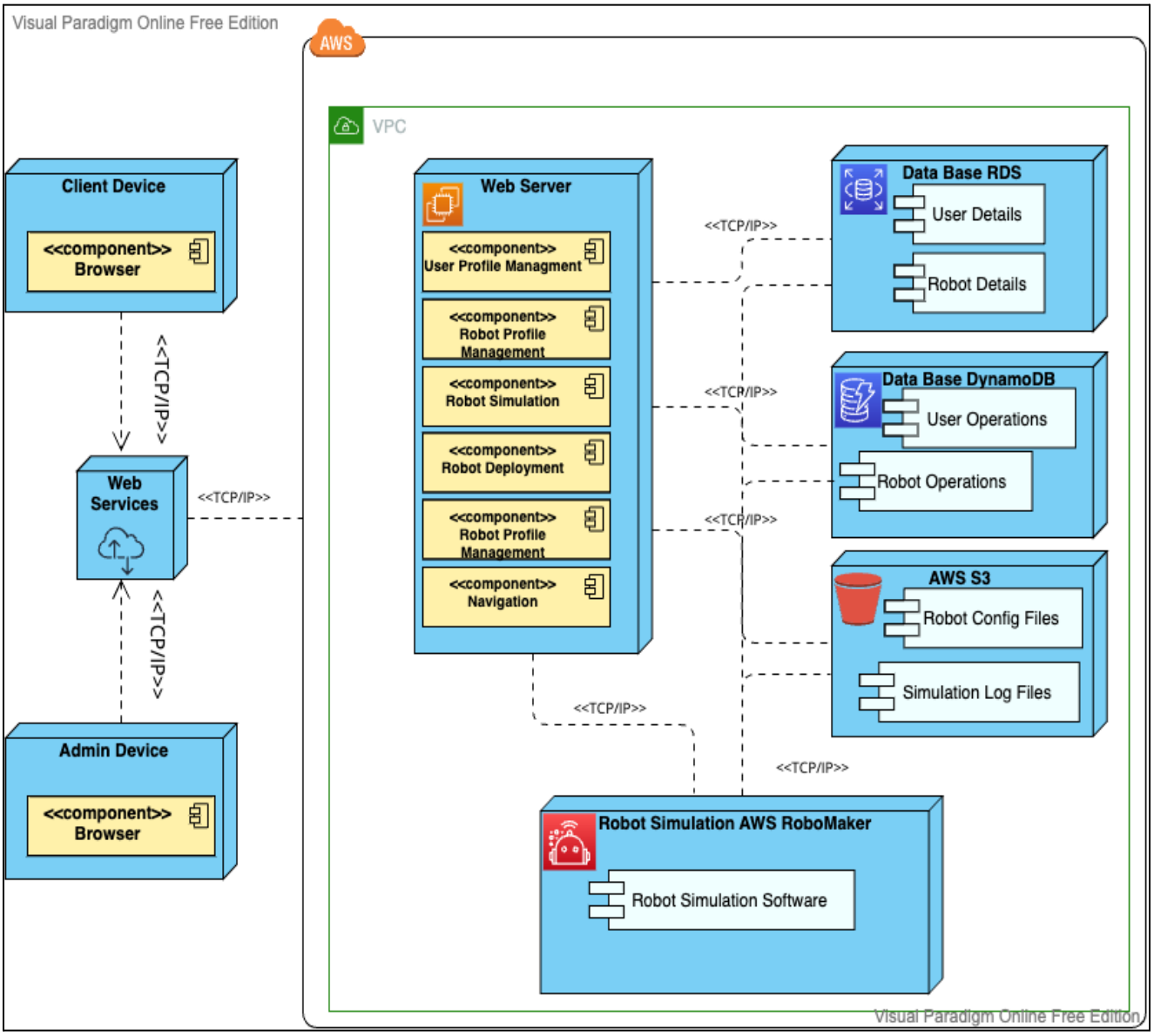


Figure 12 : System Deployment Infrastructure

4. Cloud-Based System Design And Component Interaction Design

4.1. System Database Design

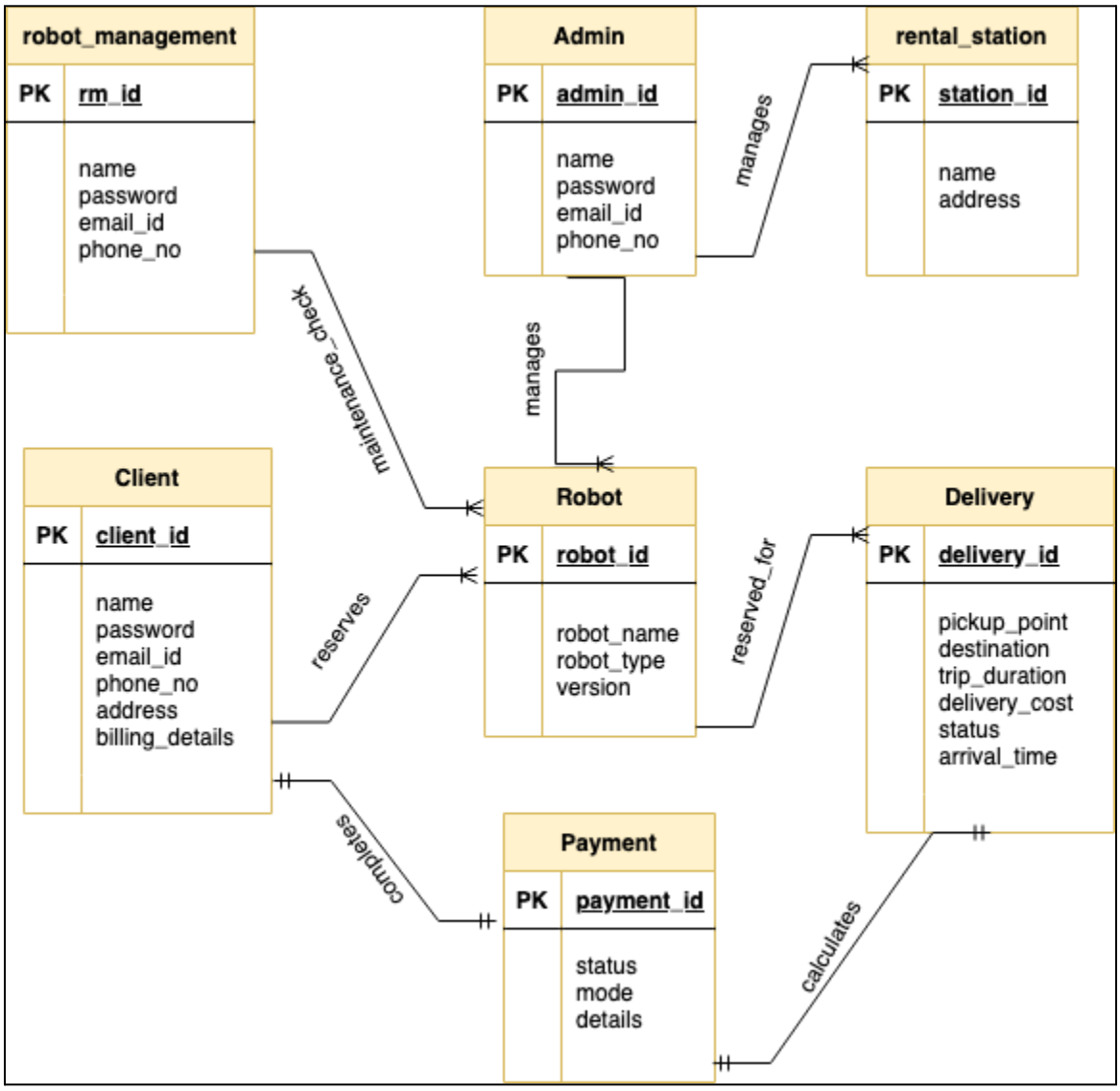
The Food Delivery Robot Cloud System collects a significant amount of sensor data from sensors placed all around the location as a result of real-time data streaming. We furthermore need to manage user and robot data in addition to this substantial data set. We have designed a database architecture to 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 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.

System Design Database:

Client data, roles, robot	MySQL
Sensor data	MongoDB
Edge-based data repository	Amazon EMR

Let's have a closer look at each type of database that we will be using.



4.1.1 Edge-based data repository design : Amazon EMR elastic map reduce

Leveraging an edge-based data repository would be the ideal option to review data given the vast amount of data that needs to be received, changed, and evaluated as well as the diverse kinds of data and formats involved. It would be able to use an Amazon Elastic Map Reduce to compile and organize the data into a regularized pattern before storing it in an edge-based data repository. One of these tools allows us to retrieve the required data.

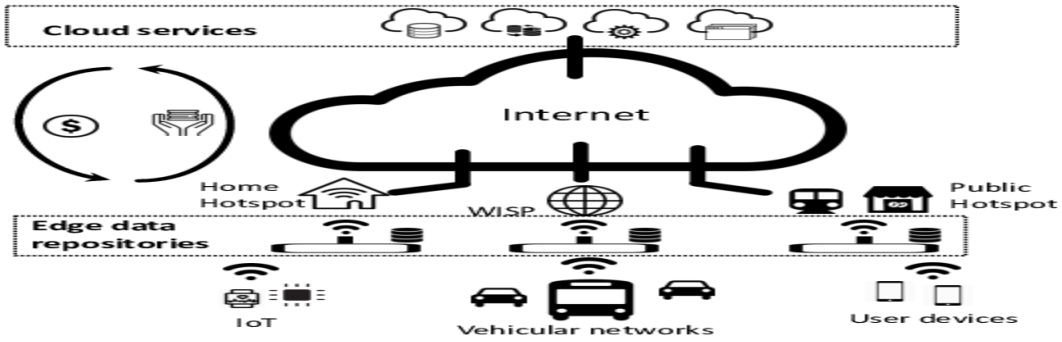


Figure 13 : Edge-based data repository design

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.

Figure 14 : 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.

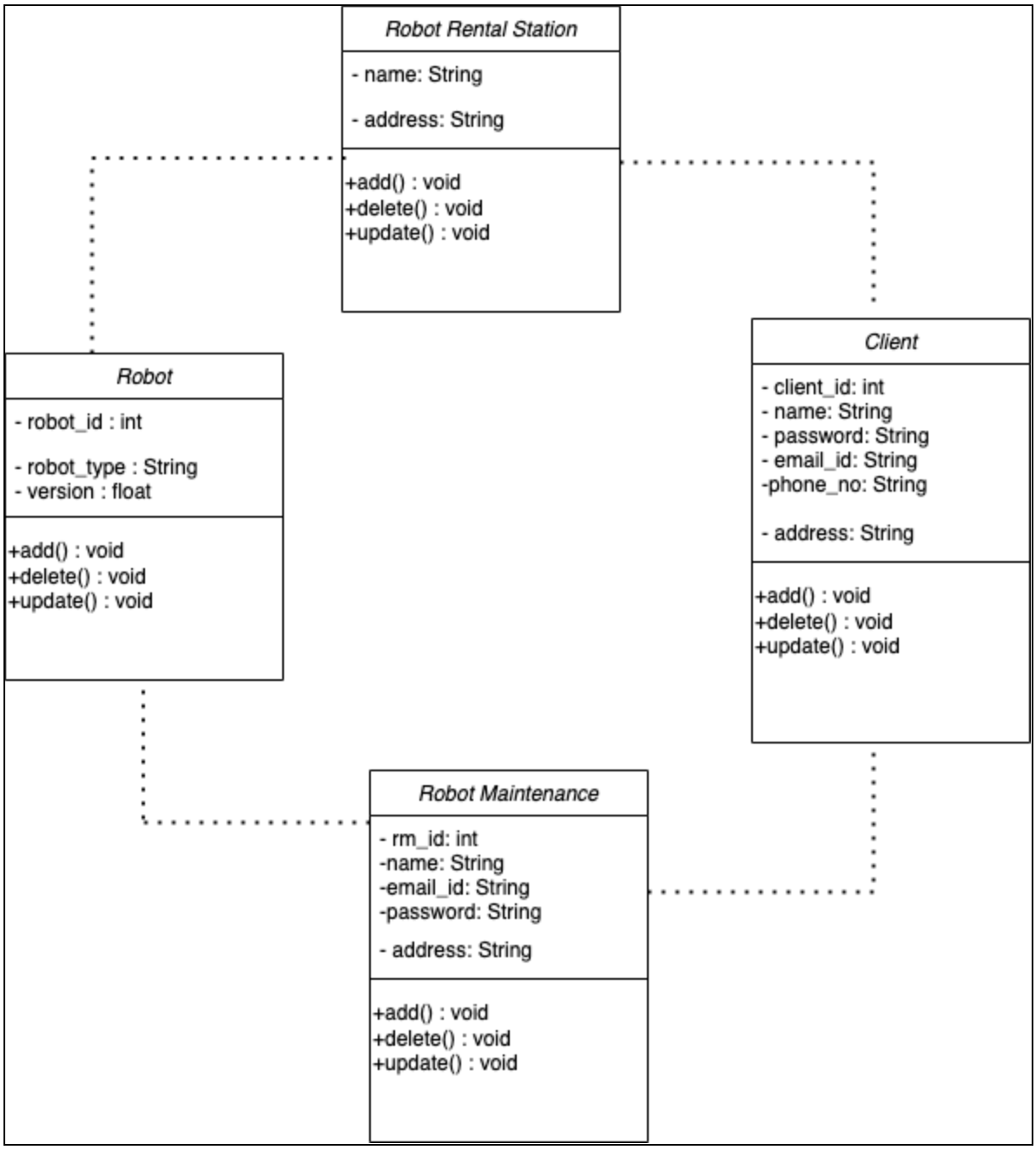


Figure 15 : NoSQL DB Design

4.2. Cloud-Based System Communication Design

This section presents the communications between the back-end cloud server and the front system components.

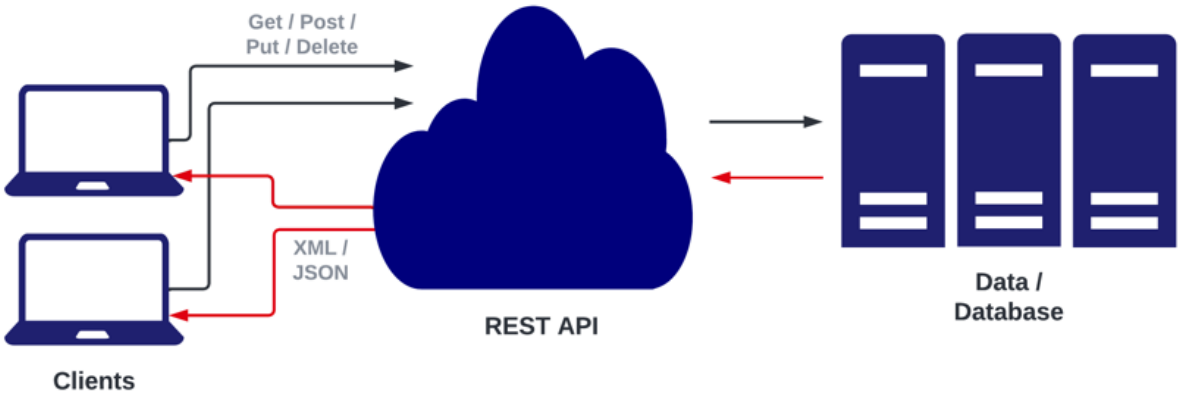


Figure 16 : Cloud Based communication through Postman in JSON Format

Two key parts, the frontend and backend, make up the Cloud-based system architecture. In this design, the front end acts as the client and connects with the back end across a network or the internet. In the architecture of cloud computing, the user can see the client-side or frontend. Through the middleware, the frontend transmits inquiries to the backend.

Data is safeguarded by the backend, which also responds to frontend questions. The cloud computing architecture's backend makes up a larger portion of the total system, as seen below:

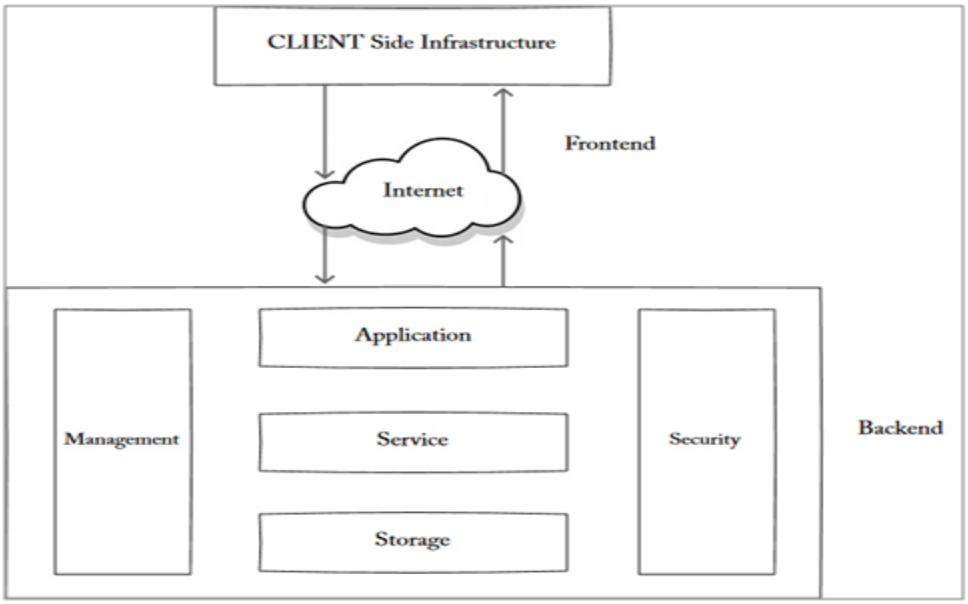


Figure 17 :Cloud Computing Architecture

Backend-as-a-service, or BaaS, is the name of the entire cloud service delivery model. The following are the basic elements of the architecture for cloud computing:

1. Front-end platform
2. Back-end platform
3. Cloud-based delivery

Front-end component

For user interaction, the frontend component only communicates with the backend component. It inputs commands into the backend component and receives the AWS cloud maker robot component's answer in return. The frontend communicates with each other using a REST architecture that is based on HTTP, and it sends and receives data in JSON format.

Back-end Component

When the robot's course is changed as a result of commands received from the front end, the back end component communicates with the AWS cloud maker component and, if all goes well, receives a message with a JSON answer. Additionally, it communicates with the database component for user access, retrieving and storing billing information, as well as all other database transactions.

AWS-Robomaker Component

We are simulating a mobile food delivery robot that can travel to a certain spot on a defined map using the AWS robomaker. The backend sends instructions to the robo maker component, which updates the robot's position on the map using the robo maker. The backend then receives a response from the robomaker SDK in JSON format.

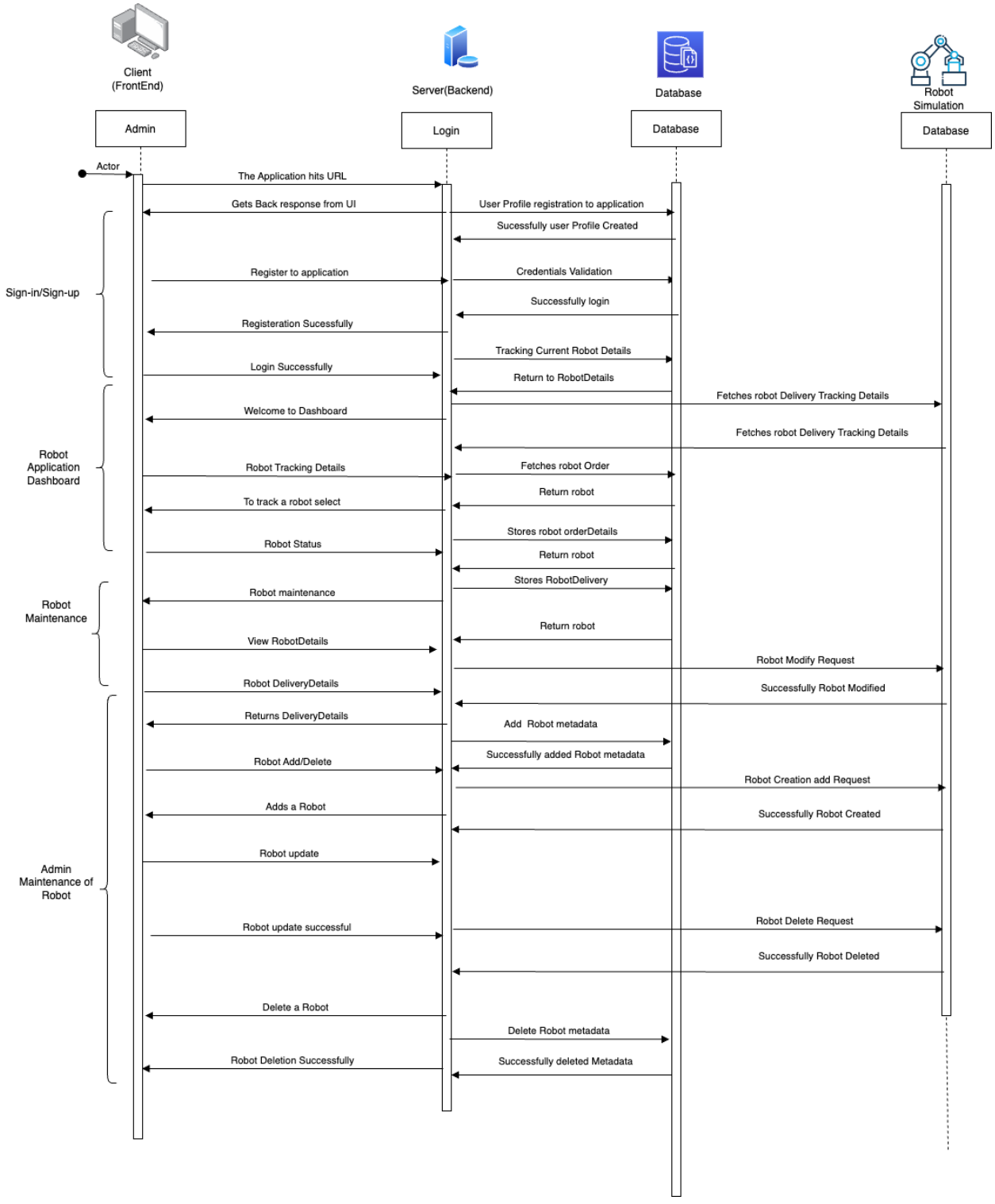


Figure 18 : Communication diagram

4.3. High-level cloud computing design

This section presents your high-level cloud computing design, including load balance, scalability, and multi-tenancy. (This section will be added later in your final project package)

According to the CAP theorem, every distributed system design contains the following three features.

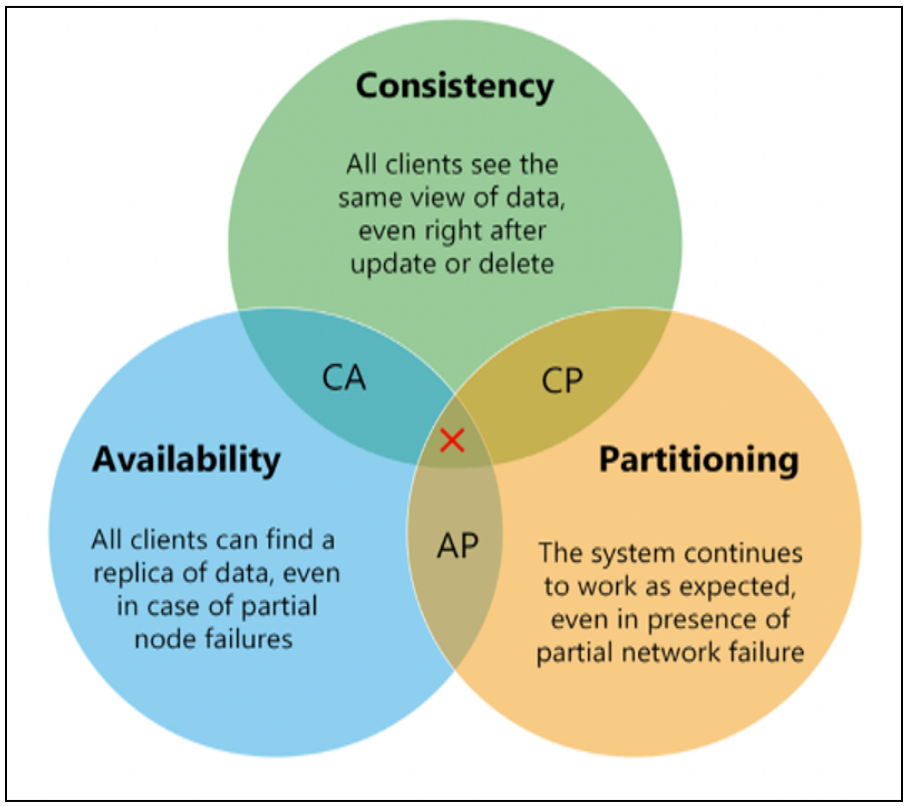


Figure 19 : CAP Diagram

The three most important factors to consider when creating any large-scale systems are safety, availability, and consistency.

In a food delivery robot cloud, the following problems could arise when client load rises.

1. There is a chance that a component of the system could malfunction in specific circumstances, such as a system failure, leading to the loss of some data. Therefore, in that situation, we can just periodically save all the data and transactions in the data storage blocks to ensure that the data won't be lost forever and can be easily recovered.
2. Similar to when a system fails, an active switchover will be made from one system to another system, allowing the system to continue operating without incurring any more expenses or losses. To make sure they are constantly in sync, they are kept in an active and ready state at all times.
3. The load balancer may automatically scale as the number of API calls to the web servers increases.

The food delivery robot cloud system architecture will be developed and implemented with all potential roadblocks in mind, as well as a suitable set of algorithms to deal with any problems that may occur. Approaches and remedies for cloud application resource management, scalability, and security problems:

Scalability and Resource Utility Management

The main objective is to dynamically assign resources and capabilities to handle the workload. Concerns about cloud database design are addressed by the resource management components that are outlined below.

- **Scheduling:** To allocate resources among the workload, an event calendar must be created.
- **Discovery:** This is the process of figuring out what resources are available to lighten the load.
- **Allocation:** In this instance, resources are distributed among competing workloads based on demand.
- **Provisioning:** Allocating resources to workloads takes place in this process.

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 Scaling 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.A load balancer scaling device is set up with rules connecting an internal pool of servers to external IP and port numbers. Clients only have access to the external IP address via DNS responses. The servers' health is monitored by the load-balancing control plane, which also chooses which ones can accept requests.

Application-Level Load Balancer Scaling: In situations where applications are properly coded, load balancing is carried out within the application and is used to configure load balancing in the application. Load balancing is implemented between tiers in the application stack. Designers can monitor flows between application stack levels using free tools, DNS, or another technique.

Scalability of Network-Level Load Balancers: DNS round-robin, Anycast, and L4-L7 load balancers are all examples of network-level load balancing. The lack of application layer redundancy in most web browser clients forces designers to go to the network layer for load-balancing services. Load balancing would not be a network-layer function if apps were properly built.

Techniques for Dealing with Workload Limitations:

Network Load Balancing:

In order to prevent any one backend server from becoming overwhelmed, multiple requests from the client(s) must be load-balanced throughout the backend servers. The incoming traffic is efficiently distributed among the backend servers. Load balancers sit in the center, taking inbound requests and distributing them to any available servers.

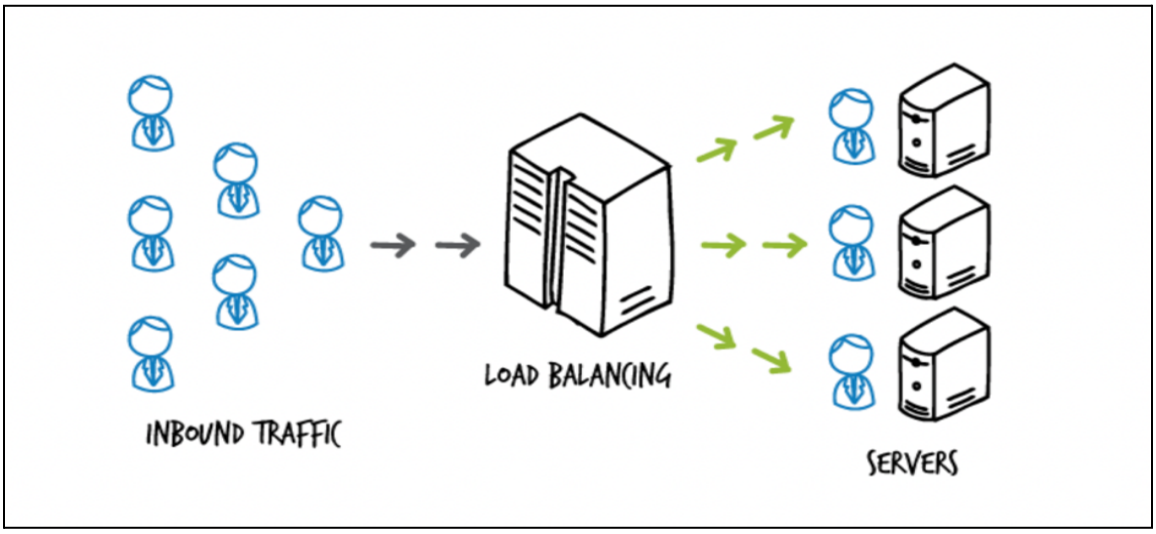


Figure 20: Load Balancing Of Network

Server-side load-balancing:

A load balancer is inserted between the application servers and backend server instances in our configuration. It serves as an intermediate component for all requests made by clients of the application. Following that, it routes client requests to the appropriate servers using algorithms like round-robin, etc.It prioritizes responses to certain network requests made by clients.

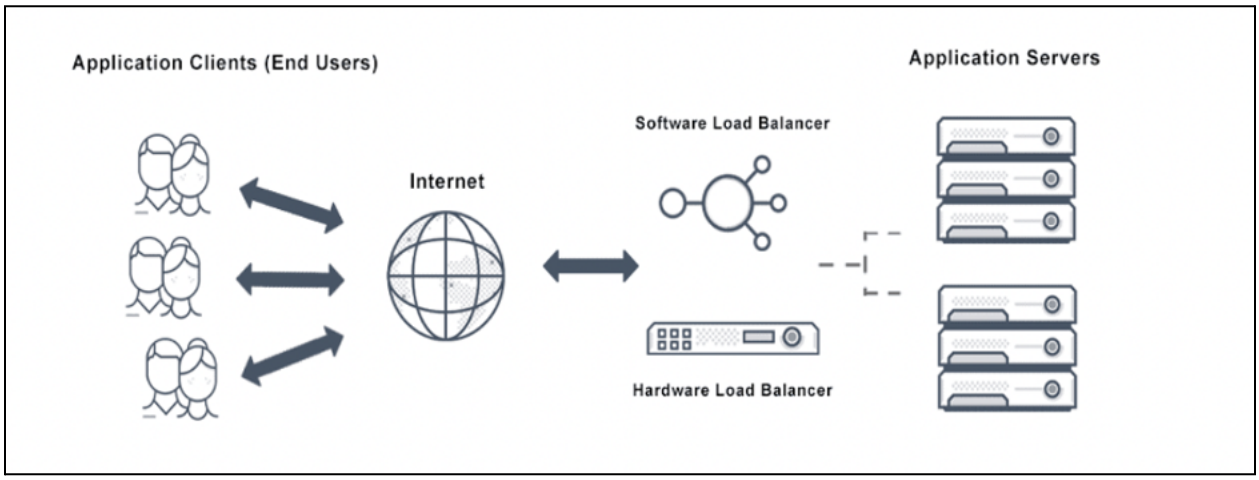


Figure 21: Server Side Load Balancing

Client-side load-balancing

In this setup, the client is in charge of making the load balancing decision. Here, the client asks the server for assistance in locating the server instances before sending the request to a particular backend server instance using a client-side load balancer.

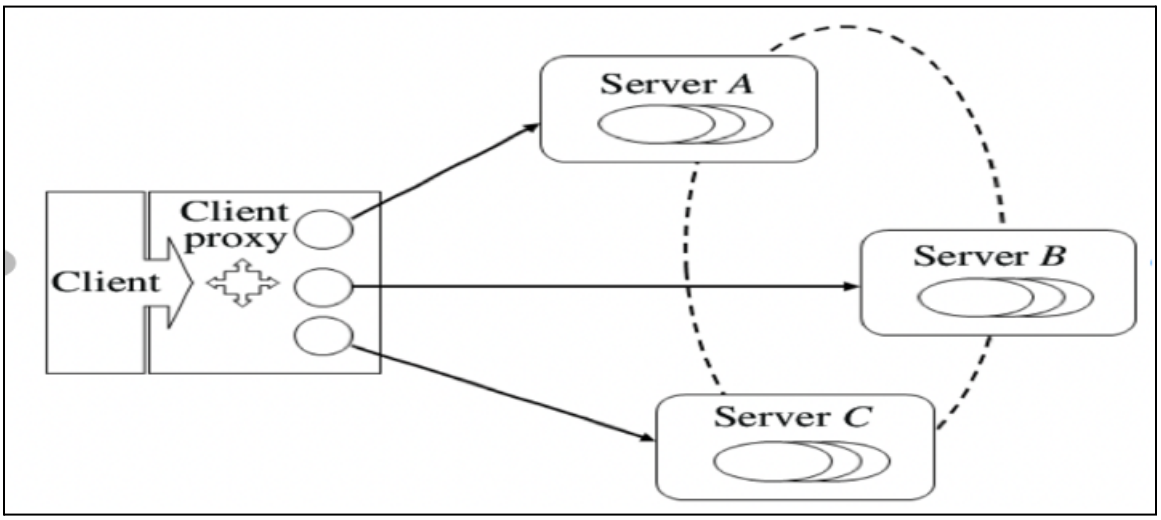


Figure 22: Client-Side Load Balancing

Round-robin algorithm:

The round-robin algorithm is a popular load balancing technique that functions best when all of the available servers have comparable setups, computing, and storage capacities. The client/user requests are cyclically distributed across the available servers to guarantee that no server is overwhelmed while the others are underutilized.

The only drawback to utilizing this approach is that it presupposes that all servers have comparable capabilities, which is not always the case. All of these algorithms, in addition to helping with load balancing, also contribute to network and system security since they prevent cyber threats from materializing and keep systems operational at all times, which in turn helps firewalls.

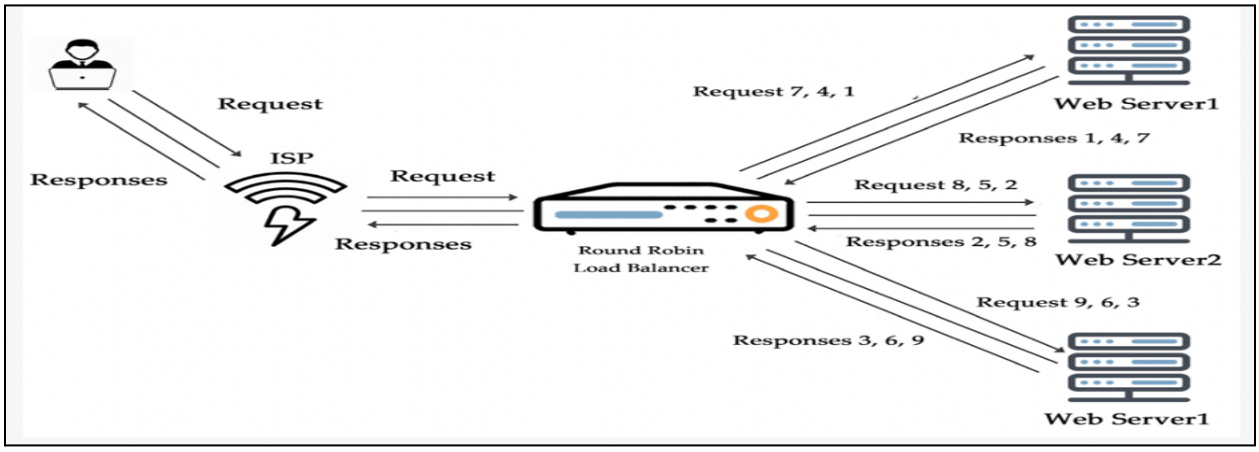
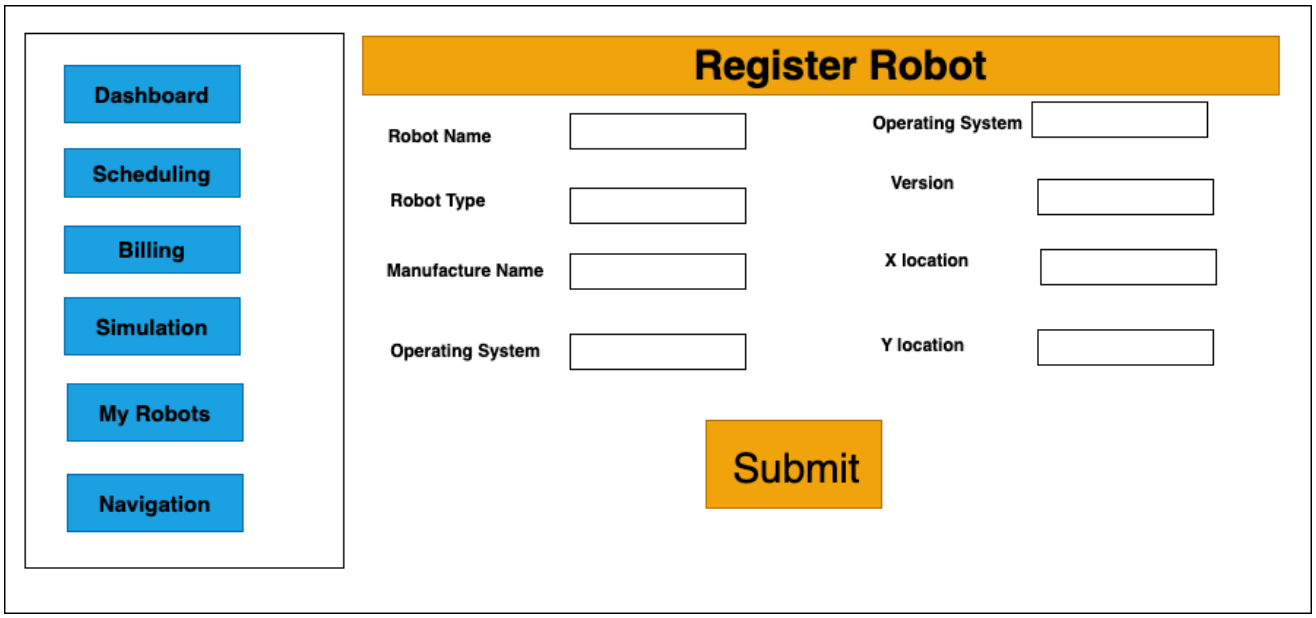


Figure 23: Algorithm about Least Connection

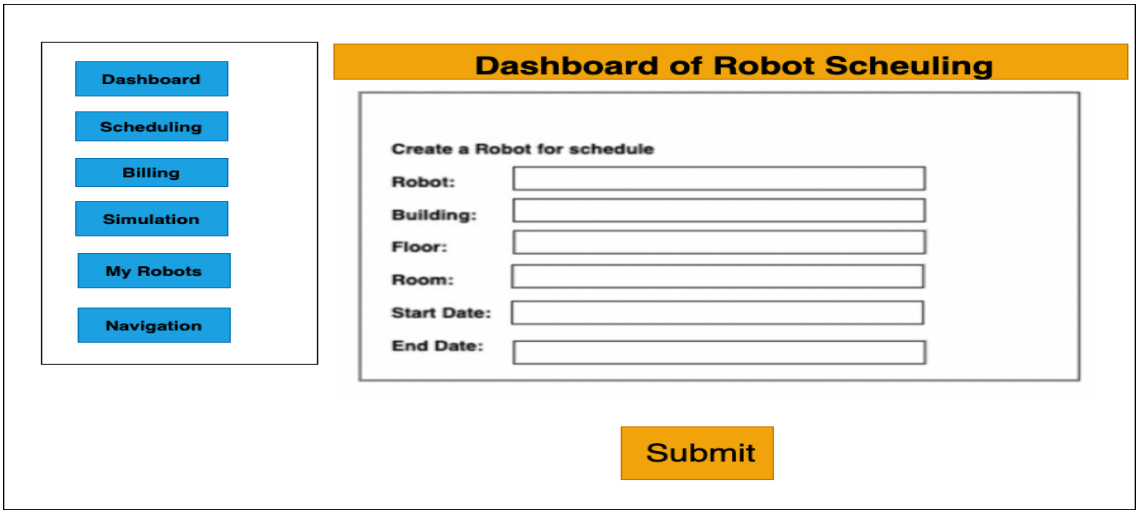
GUI System Design

The system UI frame/page design and the navigation flow diagram for different UI frames and pages The necessary functions and features of our project are Dashboard,Scheduling,Billing , Simulation, MyRobots, and Navigation The below figure represents the GUI System Design based on our project Robot Cloud, shown below in the figure.

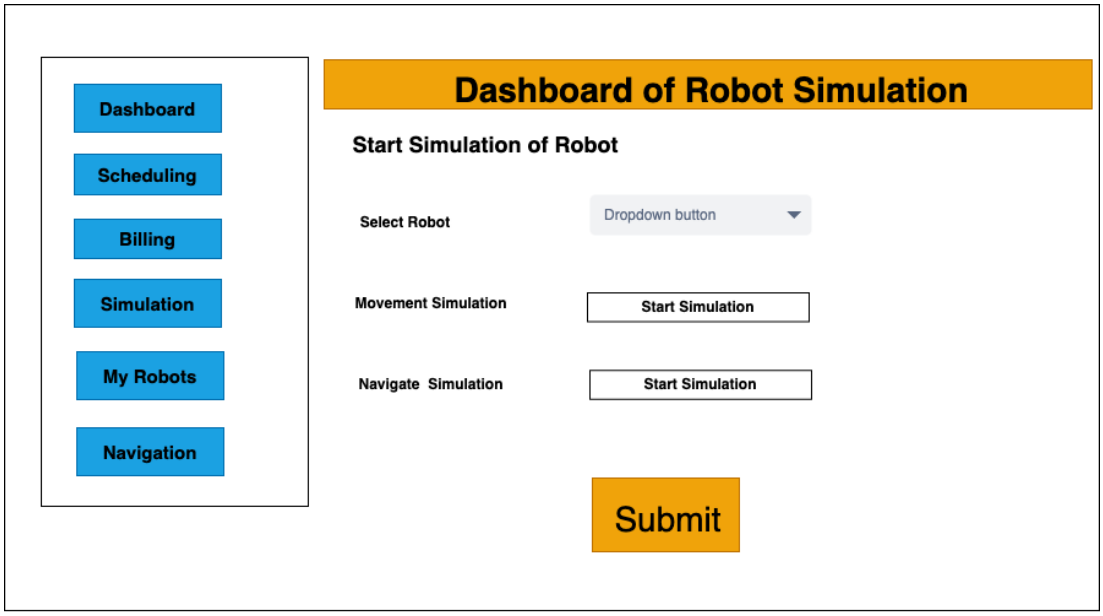
Dashboard: The flow specifications for our project application are such that once the user registers with the application as a robot, it will ask the robot to sign-in/sign up for the application if it is a new user to the robot application.



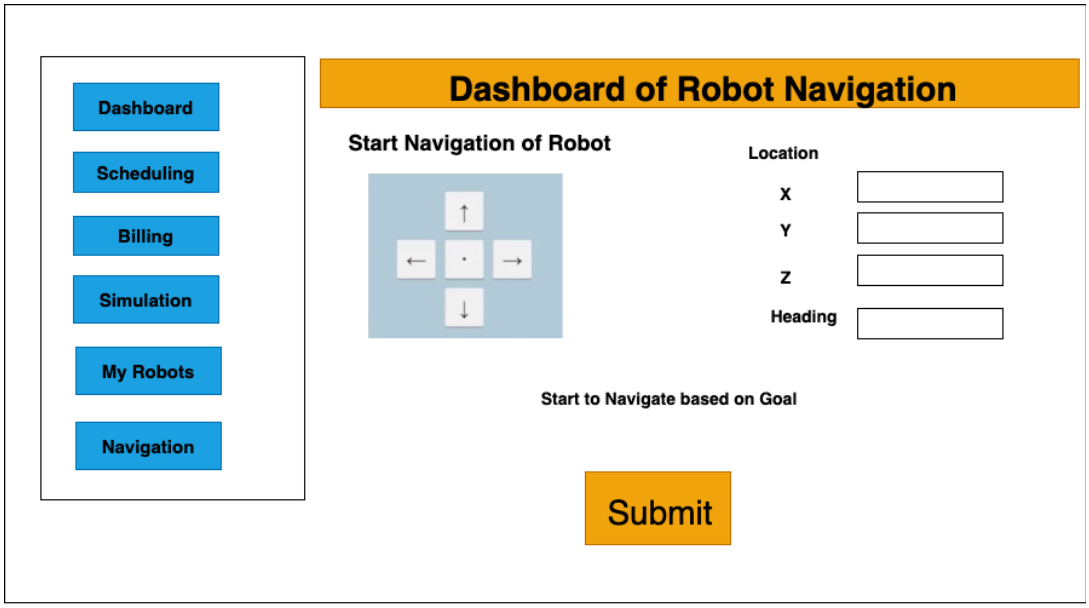
Scheduling: After registering to the application then clicking on scheduling,it will navigate to the creation of the robot schedule based on the requirements such as building,floor and room etc.



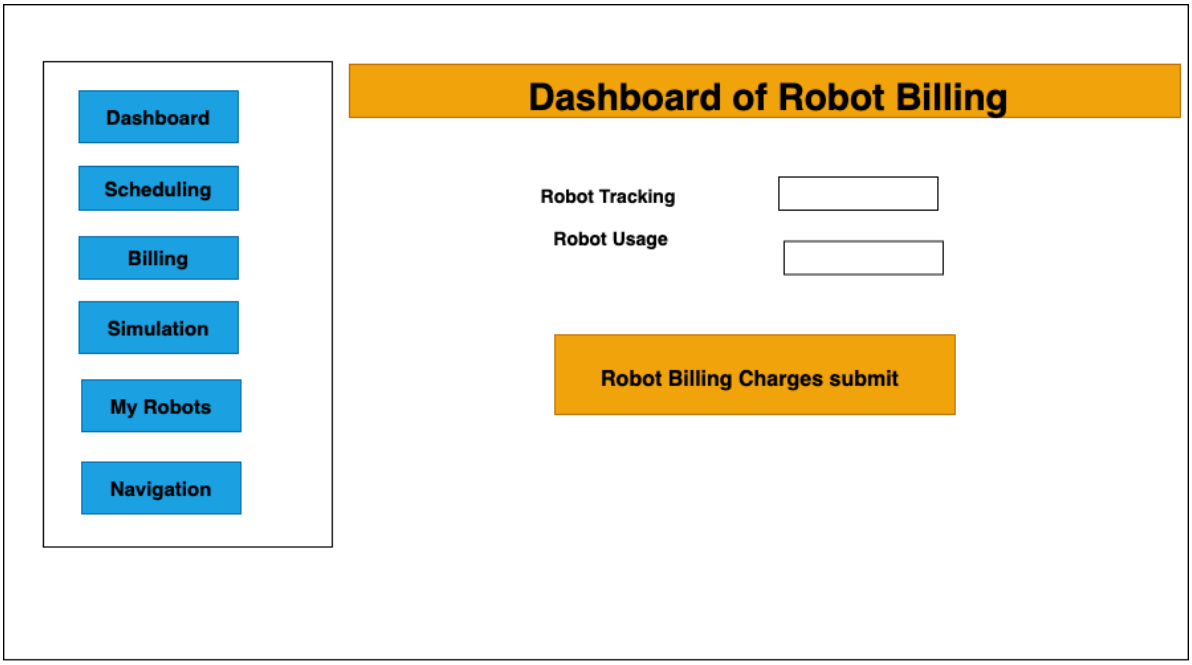
Simulation: After scheduling, click on simulation to check the operation of the robot is it working or not based on the movement simulation and navigation simulation.



Navigation: After completing the simulation, it will navigate to the direction on the three different axes to show the visualization of the robot in the simulation by taking the navigator's help.



Robot Billing: It will charge based on the usage of the robot such as seconds,minutes,hours and day etc.



5. Project Development plan and schedule

5.1 Project Team and roles

Team Member names:

1. FNU Butul Parveen
2. Kanupriya Agarwal
3. Farazuddi Mohammad
4. Chirag Arora

Roles and responsibilities:

Front End Design and Implementation:

- FNU Butul Parveen
- Farazuddin Mohammad

Backend API Design, Implementation and Testing

- Kanupriya Agarwal
- Chirag Arora

Database Design and Implementation:

SQL:

- Farazuddin Mohammad
- Kanupriya Agarwal

NoSQL:

- Chirag Arora
- FNU Butul Parveen

AWS Robomaker Configuration and Connection:

- Kanupriya Agarwal
- FNU Butul Parveen

Hosting Dashboard (Deployment on EC2)

- Farazuddin Mohammad
- Chirag Arora

5.2 Project Schedule

Tasks:

- Tech Stack Selection: Days 2
- Database Design SQL and NoSQL: Days 6
- API Design, Implementation and Testing: : Days 6
- UI Design and Implementation: Days 10
- Web Service Development: : Days 5
- Robot Simulation: : Days 4
- Dashboard Communication between Robot Simulator: : Days 2
- Unit Testing : Days 2
- Integration : Days 2
- Integration Testing : Days 3
- Deployment on Public Ip: Days 1

S.No	Name	Roles and Responsibilities
1	Chirag Arora	Backend API Design,Implementation and Testing, Database Design and Implementation:NoSQL,Hosting Dashboard (Deployment on EC2)
2	FNU Butul Parveen	Front End Design and Implementation, Database Design and Implementation:NoSQL, AWS Robomaker Configuration and Connection
3	Farazuddin Mohammad	Front End Design and Implementation,Database Design and Implementation:SQL,Hosting Dashboard (Deployment on EC2)
4	Kanupriya Agarwal	Backend API Design, Implementation and Testing,Database Design and Implementation:SQL, AWS Robomaker Configuration and Connection

PERT Network Chart

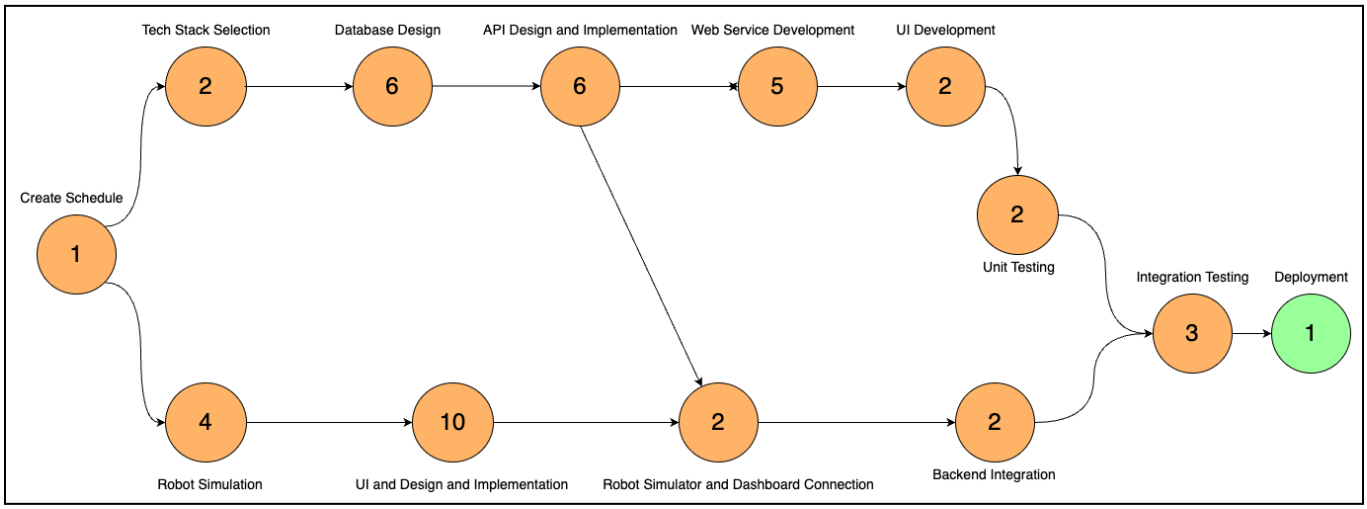


Figure 23: PERT Network Chart

Project Workflow

S.No	Deliverables	Task	Due Date	Time (day)	Status
1	Project Analysis	• Analysis of the requirements and market needs		2	Completed
2	Project Design	• Project Infrastructure Design and Architecture • Database Design • Delivery Plan	10/17/2022	4	Completed
3	Component Design	• Detailed Features/Component • API Design	10/26/2022	10	Pending
4	Component Design with Demo	• Demo of all the component level features • Submission of Component Design doc		2	Pending
5	Component Implementation	• Complete component level development		6	Pending
6	Integration and Testing	• End-to-end integration testing with simulators		6	Pending
7	Component Implementation Demo	• Demo of all the features	11/01/2022	5	Pending
8	Complete project Integration	• Submission of the analysis report of each component developed		2	Pending
9	Project Demo	• System Performance Testing • Final Report	12/06/2022		Pending

References:

1. https://www.researchgate.net/publication/316888159_An_Edge-Based_Smart_Mobile_Service_System_for_Illegal_Dumping_Detection_and_Monitoring_in_San_Jose
2. https://docs.aws.amazon.com/robomaker/latest/dg/aws-robomaker-dg.pdf
3. https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc
4. https://www.researchgate.net/publication/316888159_An_Edge-Based_Smart_Mobile_Service_System_for_Illegal_Dumping_Detection_and_Monitoring_in_San_Jose
5. https://medium.com/analysts-corner/how-to-draw-effective-it-architecture-diagrams-3da74f8288f4
6. https://docs.bmc.com/docs/TSInfrastructure/110/truesight-infrastructure-management-architecture-and-components-719678737.html
7. https://www.researchgate.net/publication/316888159_An_Edge-Based_Smart_Mobile_Service_System_for_Illegal_Dumping_Detection_and_Monitoring_in_San_Jose
8. https://www.researchgate.net/publication/2955094_Engineering_on_the_Internet_for_global_software_production
9. https://en.wikipedia.org/wiki/Deployment_diagram