S T U D Y O F

B M E T E C H N O L O G Y

Augmented Telepresence Robot using Raspberry Pi

by

Alyssa Png Kai Wen (S10192905)

Yap Shi Ting, Esther (S10194943)

Diploma in Biomedical Engineering

06 September – 24 January 2022

A minor thesis submitted in fulfilment for the diploma in Biomedical Engineering

Department of Biomedical Engineering

Ngee Ann Polytechnic

Singapore

ABSTRACT

This report summarizes the study of Augmented Telepresence Robot using Raspberry Pi, as well as the various usage of Amazon Web Services (AWS).

The aim of this project is to incorporate Augmented Reality to telepresence technology in hopes to provide users with a better sense of immersion in the simulated world using an EPSON BT-350 smart glasses. The movement and video captured on the Telepresence Robot can be remotely controlled by an authenticated user via a website.

With the help of Vue.js and AWS Cognito, the layout and authentication of the Website was constructed using JavaScript. This website would be hosted onto the wide-area network using AWS S3 and CloudFront, allowing users from anywhere with internet to access their robots.

In this report, the movement of the robot is stated to be controlled using specific messages send from the website to an AWS IoT Core MQTT topic via AWS API Gateway and Lambda. On the other hand, video and audio streaming from the robot was achieved using AWS Kinesis Video Streams with Real-Time Communication (WebRTC). To enhance the security of the streaming, AWS IoT Core credentials was used for this authentication.

Further enhancements to the project were also proposed. Facial Recognition could be added to enhance the security of the robot using TensorFlow and AWS Rekognition on the front end. To achieve two-way communication between the robot and the user, it was suggested for audio to be send from the website and back to the robot using AWS KVS WebRTC. For better sustainability and scalability, Amazon DynamoDB could be used to help store all the robots' data which could be retrieved and used on the website.

ACKNOWLEDGEMENTS

Our experience working together under this project has been very eye-opening and rewarding. We would like to express our gratitude to Mr. Foo Wan Juang for organizing the Biomedical Project Design module.

A special thank you to Mr Soon Hock Wei for his constant encouragement and guidance throughout these 22 weeks. Despite his busy schedule, he never fails to check up on our progress and takes the time to provide useful knowledge and resources to improve our projects. He has provided an environment for us to not be afraid to ask questions and learn many new things. We are very grateful for his willingness to work together with us to overcome all the challenges we have faced.

We would also like to thank Mr Liu Wei Dong for all the help and guidance he has given us. His constant support and feedback during discussions have helped to catalyse the productivity of our team. He always goes out of his way to find and provide any resource or materials we may need for our project. When faced with challenges, he would always give us the much-needed encouragement and advice to continue working hard for our project.

TABLE OF FIGURES

FIGURE 1-1: WORK BREAKDOWN STRUCTURE 10
FIGURE 1-2: GANTT CHART PART 1 10
FIGURE 1-3: GANTT CHART PART 2 10
FIGURE 1-4: R.A.C.I. TABLE 11
FIGURE 2-1: RASPBERRY PI 4B+ 12
FIGURE 2-2: FREENOVE 4WD SMART CAR KIT WITH RASPBERRY PI 12
FIGURE 2-3: TRANSMISSION PARTS 13
FIGURE 2-4: CAMERA MODULE AND FPC WIRE 13
FIGURE 2-5: ULTRASOUND SENSOR 14
FIGURE 2-6: ACRYLIC PIECES 14
FIGURE 2-7: PCB BOARD 14
FIGURE 2-8: 3.7V 18650 LITHIUM RECHARGEABLE BATTERIES 15
FIGURE 2-9: RESPEAKER MIC ARRAY V2.0 15
FIGURE 2-10 ADAFRUIT STEMMA SPEAKER 16
FIGURE 2-11: JUMPER WIRE SOLDERED ONTO GROUND WIRE OF AUX CABLE 16
FIGURE 2-12: SPEAKER ATTACHED TO THE ROBOT 17
FIGURE 2-13: EPSON BT350 SMART GLASSES 17
FIGURE 3-1: ARCHITECTURE DIAGRAM 19
FIGURE 3-2: PROJECT COMPARISON 20
FIGURE 4-1: URL AND ENDPOINT OUTPUT 23
FIGURE 4-2: AWS IOT THING AND CERTIFICATE 24
FIGURE 4-3: AUTHORIZER CREATED WITH CORRESPONDING COGNITO POOL 24
FIGURE 4-4: AUTHORIZATION USING POST REQUEST 25
FIGURE 4-5: INLINE POLICIES 25
FIGURE 4-6: PERMISSIONS ADDED TO POLICIES 26
FIGURE 4-7: REST API PRICES 28
FIGURE 4-8: INVOICE FOR DECEMBER 29
FIGURE 4-9: RPIROBOT AND RPIROBOT2 29
FIGURE 5-1: MAIN.PY CODE TO EXTRACT DESIRE INFORMATION 30
FIGURE 5-2: MAIN.PY CODE TO CONNECT RASPBERRY PI AS MQTT CLIENT 30
FIGURE 5-3: IOT POLICY 31
FIGURE 5-4: HANDLEMESSGAE() CODE 32
FIGURE 5-5: RECEIVED MESSAGE STATEMENT 33
FIGURE 5-6: MAIN.PY CODE TO IMPORT SERVO.PY AND MOTOR.PY 33
FIGURE 5-7: HANDLEMESSAGE() CODE TO CONTROL SERVO 34
FIGURE 5-8: SERVO.PY CODE 34
FIGURE 5-9: HANDLEMESSAGE () CODE THAT CONTROLS MOTOR 35
FIGURE 5-10: MOTOR.PY CODE 35
FIGURE 5-11: PCA9685.PY CODE 36
FIGURE 5-12: ROLE ALIAS POLICY 37
FIGURE 5-13: CLOUDFORMATION TEMPLATE.YAML CODE FOR ROLE ALIAS 37
FIGURE 5-14: AWS KVS VIEWER STATISTICS 38
FIGURE 5-15: MAIN.PY RUNKINESISVIDEOSTREAM() 38
FIGURE 5-16: CODE FOUND IN KVSWEBRTCCLIENTMASTERGSTREAMERSAMPLE 39
FIGURE 5-17: ENOENT ERROR MESSAGE 40
FIGURE 5-18: JUMPER WIRE SOLDERED ONTO BOARD 43
FIGURE 6-1: USER POOLS CREATED IN AWS COGNITO 45
FIGURE 6-2: AN EXAMPLE OF USER INFORMATION STORED IN DYNAMODB TABLE 46
FIGURE 6-3: SIGN-IN PAGE 46
FIGURE 6-4: “USERNAME CANNOT BE EMPTY!” ERROR MESSAGE 47
FIGURE 6-5: "PASSWORD CANNOT BE EMPTY!" ERROR MESSAGE 47

FIGURE 6-6: "INVALID USERNAME/PASSWORD" ERROR MESSAGE 47
FIGURE 6-7:SIGN-UP PAGE 48
FIGURE 6-8: "USERNAME CANNOT BE EMPTY!" ERROR MESSAGE 49
FIGURE 6-9: "ENTER EMAIL" ERROR MESSAGE 49
FIGURE 6-10: "PASSWORD CANNOT BE EMPTY!" ERROR MESSAGE 49
FIGURE 6-11: "SIGN UP ERROR" ERROR MESSAGE 50
FIGURE 6-12: CONFIRM SIGN-UP PAGE 50
FIGURE 6-13: NEW USER POOL CREATED IN AWS COGNITO 51
FIGURE 6-14: FORGOT PASSWORD PAGE 51
FIGURE 6-15: CHANGE PASSWORD PAGE 52
FIGURE 6-16: AFTER USER SIGN IN PAGE 52
FIGURE 6-17: CONFIGURE AUTH IN MAIN.JS 53
FIGURE 6-18: CONFIGURE ENDPOINTS IN MAIN.JS 53
FIGURE 6-19: CREATE VUE INSTANCE IN MAIN.JS 54
FIGURE 6-20: INDEX.HTML CODE 54
FIGURE 6-21: APP.VUE CODE 55
FIGURE 6-22: BEFORE USER SIGN IN 56
FIGURE 6-23: AFTER USER SIGN IN 56
FIGURE 6-24: AUTHENTICATION IN SIGNIN.VUE 57
FIGURE 6-25: CATCH() FUNCTION FOR ERRORS 58
FIGURE 6-26: LOADFACEBOOKSDK() METHOD 59
FIGURE 6-27: WAITFORINIT() METHOD 59
FIGURE 6-28: CODE TO SHOW RESPONSE AFTER USER ACCEPTS PERMISSIONS 60
FIGURE 6-29: AUTHENTICATION FOR FACEBOOK 60
FIGURE 6-30: BEFORE USER POOL CREATED 61
FIGURE 6-31: AFTER USER POOL CREATED 61
FIGURE 6-32: AUTHENTICATION IN SIGNUP.VUE (VERIFICATION CODE SENT TO USER) 62
FIGURE 6-33: AUTHENTICATION IN SIGNUP.VUE (CONFIRM SIGN-UP) 62
FIGURE 6-34: CATCH() FUNCTION FOR ERRORS 63
FIGURE 6-35: AUTHENTICATION IN FORGOTPASSWORD.VUE 63
FIGURE 6-36: AUTHENTICATION IN FORGOTPASSWORDVERIFICATION.VUE 64
FIGURE 6-37: INTERFACE.VUE CODE 64
FIGURE 6-38: LAMBDA FUNCTION FOR ROBOT MOVEMENT 65
FIGURE 6-39: POSTDATA() METHOD CODE 65
FIGURE 6-40: CREATE KVS CLIENT 66
FIGURE 6-41: GET SIGNALING CHANNEL ENDPOINTS 66
FIGURE 6-42: CREATE KVS SIGNALING CLIENT 67
FIGURE 6-43: GET ICE SERVER CONFIGURATIONS 67
FIGURE 6-44: CREATE RTCPEERCONNECTION 68
FIGURE 6-45: CREATE WEBRTC SIGNALING CLIENT 68
FIGURE 6-46: SIGNALING CLIENT EVENT LISTENERS (ABOVE 3 IMAGES) 69
FIGURE 7-1: RECEIVEGSTREAMERAUDIOVIDEO() FUNCTION 74

TABLE OF ABBREVIATION

Abbrevation	***Explanation***
API	Application Programming Interface
AR	Augmented Reality
AWS	Amazon Web Services
CLI	Command Line Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
HLS	HTTP Live Streaming
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IAM	Identity and Access Management
ICE	Interactive Connectivity Establishment
IoT	Internet of Things
KVS	Kinesis Video Stream
MFA	Multi-Factor Authorization
MQTT	MQ Telemetry Transport
NAT	Network Address Translation
PWM	Pulse Width Modulation
SAM	Serverless Application Model
SDK	Software Development Kit
SDP	Session Description Protocol
STUN	Session Traversal Utilities for NAT
TURN	Traversal Using Relays around NAT
VR	Virtual Reality
WebRTC	WebRTC
WSS	Wavelength Selective Switch

TABLE OF CONTENT

1. INTRODUCTION 9

1.1. PROJECT STATEMENT 9
1.1. PROJECT SCOPE 9
1.2. RESPONSIBILITIES 9
1.3. DIAGRAMS 10
1.4. REQUIREMENTS 11

2. PROJECT MATERIALS 12

2.1. RASPBERRY PI 12
2.2. FREENOVE 4WD SMART CAR KIT 12
2.3. AUDIO DEVICES 15
2.4. RESPEAKER MIC ARRAY V2.0 15
2.5. ADAFRUIT STEMMA SPEAKER (3885) 16
2.6. EPSON BT-350 SMART GLASSES 17

3. TELEPRESENCE ROBOT 18

3.1. AUGMENTATIVE REALITY 18
3.2. DIFFERENCE BETWEEN AUGMENTATIVE REALITY AND VIRTUAL REALITY 18
3.3. AMAZON WEB SERVICES (AWS) 19
3.4. ARCHITECTURE DIAGRAM 19
3.5. ENHANCEMENT 19
3.6. SERVERLESS 20
3.7. ADVANTAGE OF SERVERLESS 21

4. DEPLOYMENT 22

4.1. AUTHETICATION 24
4.2. MOVEMENT 24
4.3. STREAMING OF VIDEO 25
4.4. STATIC HOSTING OF THE WEBSITE 26
4.5. PRICING 26
4.5.1. ALWAYS FREE 27
4.5.2. 12-MONTHS FREE TIER 27
4.5.2.1. API GATEWAY 27
4.5.2.2. IOT CORE 28
4.5.3. AWS KVS WEBRTC 28
4.6. SCALABILITY 29

5. ROBOT MECHANISM (BACK-END) 30

5.1. MOVEMENT 31
5.1.1. PYTHON CODE 32
5.2. VIDEO STREAMING 36
5.2.1. PYTHON CODE 38
5.3. CHALLENGES 40

6. WEBSITE (FRONT-END) 44

6.1. AWS AMPLIFY 44
6.2. AWS COGNITO 45
6.3. AMAZON DYNAMODB 45
6.4. USER INTERFACE 46
6.4.1. SIGN-IN PAGE 46
6.4.2. SIGN-UP PAGE 48
6.4.3. CONFIRM SIGN-UP PAGE 50
6.4.4. FORGOT PASSWORD PAGE 51
6.4.5. CHANGE PASSWORD PAGE 52
6.4.6. AFTER USER SIGN IN PAGE 52
6.5. VUE.JS CODE 53
6.5.1. MAIN.JS 53
6.5.2. INDEX.HTML 54
6.5.3. APP.VUE 55
6.5.4. SIGN-IN PAGE 56
6.5.5. FACEBOOK SIGN-IN 58
6.5.6. SIGN-UP PAGE 61
6.5.7. FORGOT PASSWORD 63
6.5.8. INTERFACE 64
6.5.9. VIDEOVIEWER 66
6.6. CHALLENGES 70

7. REFLECTION 72

7.1. ENHANCEMENT 73
7.2. CONCLUSION 75

8. BIBLIOGRAPHY 76

1. INTRODUCTION

1.1. PROJECT STATEMENT

Incorporate Augmented Reality (AR) into a telepresence robot (smart car and Raspberry Pi) by allowing remote access to its movement and camera on a website and smart glasses (EPSON BT350).

1.1. PROJECT SCOPE

The purpose of this project is to improve telepresence technology by incorporating AR to provide users with a better sense of immersion in the simulated world. This offers the user the impression of being present where the augmented telepresence robot is located.

Objectives & deliverables of this project are:

1. Produce a functional telepresence robot based on Raspberry Pi by applying knowledge about Python Language programming
2. Program the Epson BT350 to receive the video stream from cloud resources, GUI design and robot movement control
3. Allow remote access to the telepresence robot by using AWS cloud services

1.2. RESPONSIBILITIES

Project requirements will be divided between the front-end (Esther) and back-end (Alyssa). The front-end consists of the website development using Vue.js and implementing authentication via Amazon Web Services (AWS) Cognito. The back end consists of connecting the Raspberry Pi robots to AWS Internet of Things (IoT) core to enable user control movement and, to AWS Kinesis Video Stream (KVS) to stream live video captured.

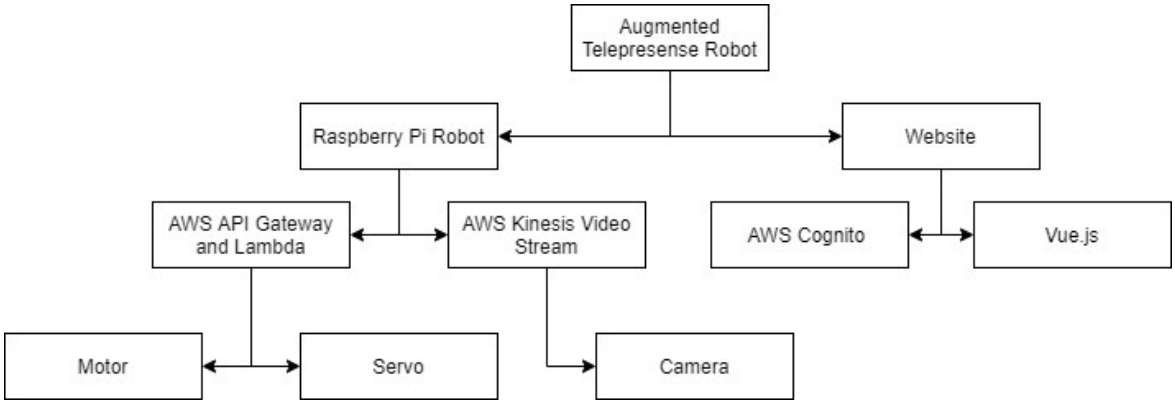

Figure 1-1: Work Breakdown Structure

1.3. DIAGRAMS

FYP-Augmented Telepresence Robot

Project Lead: Click to edit

Project Budget: Estimated: $0.00 | Baseline: $0.00 | Task Costs: Estimated: $0.00 | Baseline: $0.00 | Actual: $0.00

WBS	Task	Priority	Resource	Start	Finish	Duration	Done	% Complete
1	Understanding AWS Services	NORMAL		Mon 06-Sep-21	Fri 17-Sep-21	10		95%
2	Project Planning & Presentation	NORMAL		Fri 10-Sep-21	Mon 20-Sep-21	7		90%
3	Learn Python Language	NORMAL		Mon 06-Sep-21	Mon 13-Sep-21	6		60%
4	Learn Web Development by Vue.js	NORMAL		Thu 16-Sep-21	Wed 22-Sep-21	5		0%
5	Progress Review & Submission of Draft Design Report	NORMAL		Thu 23-Sep-21	Fri 01-Oct-21	7		0%
6	Setup AWS Services with Raspberry Pi	HIGH		Mon 20-Sep-21	Sun 26-Sep-21	5		0%
7	Web Development	HIGH		Wed 22-Sep-21	Sun 26-Sep-21	3		0%
8	Break	LOW		Mon 04-Oct-21	Sun 17-Oct-21	10		0%
9	Common Test & Break	NORMAL		Mon 13-Dec-21	Mon 03-Jan-22	16		0%
10	Combine Frontend with Backend	HIGH		Mon 20-Dec-21	Mon 03-Jan-22	11		0%
11	BPD Interim Report Writing	NORMAL		Mon 20-Dec-21	Mon 03-Jan-22	11		0%
12	Test & Scale the serverless approach	HIGH		Mon 03-Jan-22	Sun 09-Jan-22	5		0%
13	BPD Final Report Writing	HIGH		Mon 10-Jan-22	Fri 14-Jan-22	5		0%
14	BPD Supervisor's Review	HIGH		Mon 17-Jan-22	Fri 21-Jan-22	5		0%
15	BPD Panel Review	HIGH		Mon 24-Jan-22	Mon 24-Jan-22	1		0%

Figure 1-2: Gantt Chart Part 1

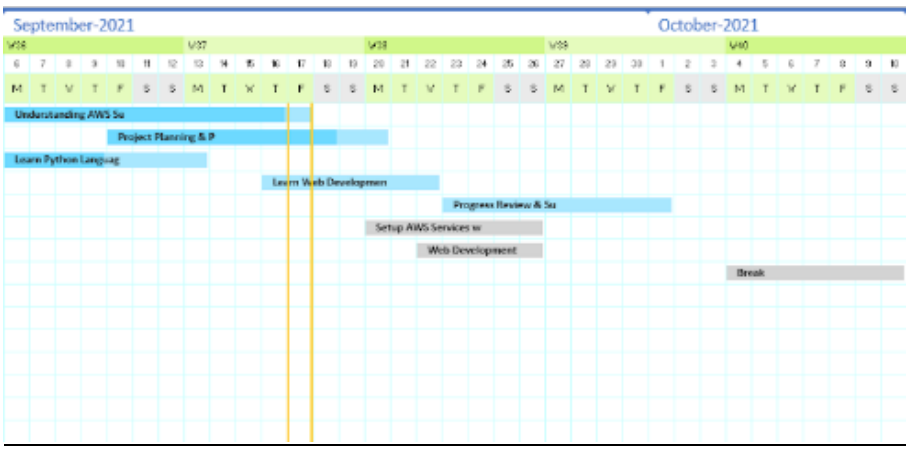

Figure 1-3: Gantt Chart Part 2

	Alyssa	Esther	Mr Soon	Mr Liu
Program and Enhance Raspberry Pi Robot	R/A	R	C	C
Connect Smart Glasses to Robot	R/A	R	C	C
Upload website onto the internet	R	R/A	C	I
Website Interface and Traffic Flow	R	R/A	I/C	I
Create AWS Cognito for authentication	R	R/A	C	I
Stream live video using AWS Kinesis and allow for motion detection	R/A	R	C	I
Use AWS API Gateway and Lambda to send messages to control robot's and camera movement	R/A	R	C	I

Figure 1-4: R.A.C.I. Table

1.4. REQUIREMENTS

The functional requirement for the project is to have a system that allows users to remotely control the robot. This includes the adjusting of its' camera angle and to be able to drive the robot. The system must also enable streaming of the live video captured by the Raspberry Pi robot camera. By creating a website, the system is accessible on any computer, handphone as well as the user's AR headset (Epson BT350).

The technical requirements for the project are to operate the web-based system on Vue.js and provide authentication using AWS Cognito. The website system controls the robot remotely using AWS application programming interface (API) Gateway and Lambda and allows video streaming via AWS Kinesis Video Stream with the help of Web Real-Time Communication (WebRTC). The Epson BT350 smart glasses will connect to the robot by accessing the website.

2. PROJECT MATERIALS

2.1. RASPBERRY PI

Figure 2-1: Raspberry Pi 4B+
(Element14, n.d.)

Raspberry Pi is an affordable, single board computer that connects to a monitor, and requires a keyboard and mouse. Raspberry Pi enables the exploration of computing and programming using languages such as Scratch and Python. For this project, the Raspberry Pi will serve as a microcontroller responsible for controlling the movement of the telepresence robot.

2.2. FREENOVE 4WD SMART CAR KIT

Freenove provides open-source electronic products and services worldwide. Their services include kits that promotes the learning of programming, robotics, and electronics as well as supplying various modules, components, and tools. Freenove kits are compatible with microprocessors such as Raspberry Pi and microcontrollers like Arduino. They help to cultivate creativity and innovation as their kits allow for design and customisation.

Figure 2-2: Freenove 4WD Smart Car Kit with Raspberry Pi
(Amazon, n.d.)

A Freenove 4WD Smart Car kit was used to serve as the telepresence robot and as seen in the figure above, the Raspberry Pi 4B+ will be attached to the car.

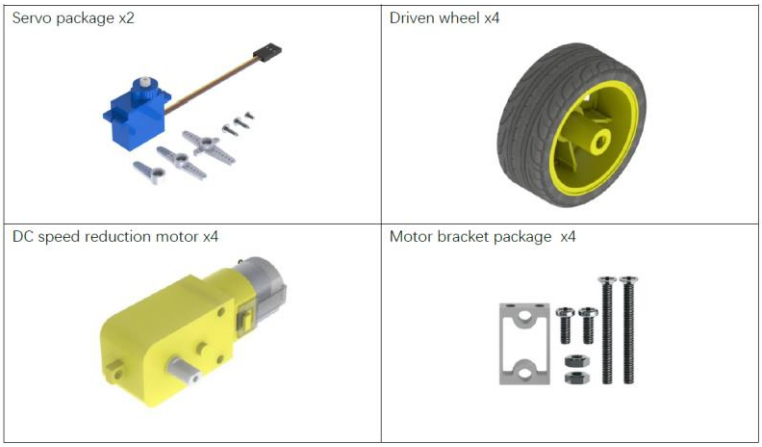

Figure 2-3: Transmission Parts
(Chen, Denzel, 2021)

The transmission parts include two servos, four DC speed reduction motor and brackets and four driven wheels. The two servos are responsible for controlling the angle of the camera. One servo is responsible for moving the camera left and right, while the other is to move it up and down. The motors were attached to the wheels and will control the direction at which the robot will move

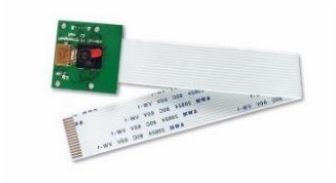

Figure 2-4: Camera module and FPC wire
(Newegg, n.d.)

The figure above shows the Camera module attached to the Raspberry Pi to capture live footage from robot's perspective, The kit also includes an ultrasound sensor and a light tracking module as seen in the figure below,

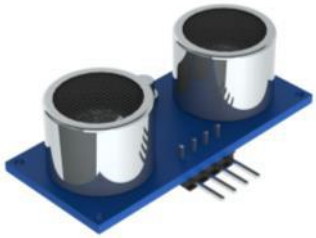

Figure 2-5: Ultrasound Sensor
(Chen, Denzel, 2021)

The servo, camera module, and the ultrasound sensor are all connected and supported in the desire position using the acrylic pieces shown below.

Figure 2-6: Acrylic pieces

All the different parts of the kit are placed together and attached to the PCB board shown in the Figure 2-7.

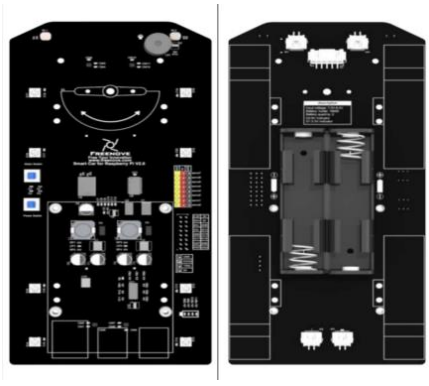

Figure 2-7: PCB board

The robot requires two 3.7V 18650 lithium rechargeable batteries, shown in Figure 2-8, to operate. The batteries help to power up the Raspberry Pi and all the electrical components attached to the car. It allows the Raspberry Pi to work with having to be plugged in and therefore, allow the smart car to move about freely,

Figure 2-8: 3.7V 18650 lithium rechargeable batteries (Chen, Denzel, 2021)

2.3. AUDIO DEVICES

To bring telepresence to the next level, a speaker and mic was added to allow two-way audio communication between website and the robot.

2.4. RESPEAKER MIC ARRAY V2.0

Figure 2-9: Respeaker Mic Array v2.0 (seeedstudio, n.d.)

Using the USB port, the ReSpeaker Mic Array v2.0 is connected to the Raspberry pi board. It is a far-field microphone array device capable of detecting voices up to 5m away even

with the presence of background noise. Therefore, making it a good microphone to pick up the surrounding sound of the robot. (seeedstudio, n.d.)

2.5. ADAFRUIT STEMMA SPEAKER (3885)

Figure 2-10 Adafruit STEMMA speaker (adafruit, n.d.)

Adafruit STEMMA Speaker with a Plug and Play Audio Amplifier was attached to the board to playback incoming audio from the client. It is small and compact making it ideal for the robot. The speaker consists of 3 connection pins, power, ground, and audio pins. The power pin was connected to PIN 17 of the Raspberry pi GPIO pins which supplies 3.3V.

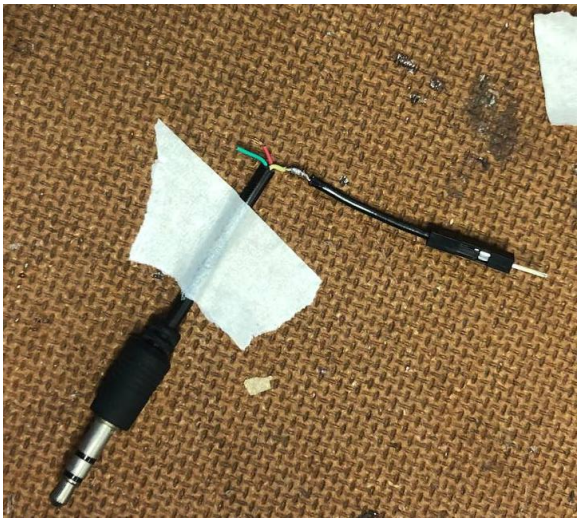

Figure 2-11: Jumper wire soldered onto ground wire of aux cable

For the ground and audio pins, they were attached to the ground and audio left wire of an aux cable respectively. To do this, the aux cable was cut, and the rubber casing of the wire was removed to reveal the audio right, audio left and ground wires. A multi-meter was then

used to identify these wires. Once identified, a jumper wire with the female header pin on one end was soldered onto the ground and the audio Left wires each. Electrical tape was then used to ensure there was no exposed wires and the speaker was connected to the aux cable accordingly.

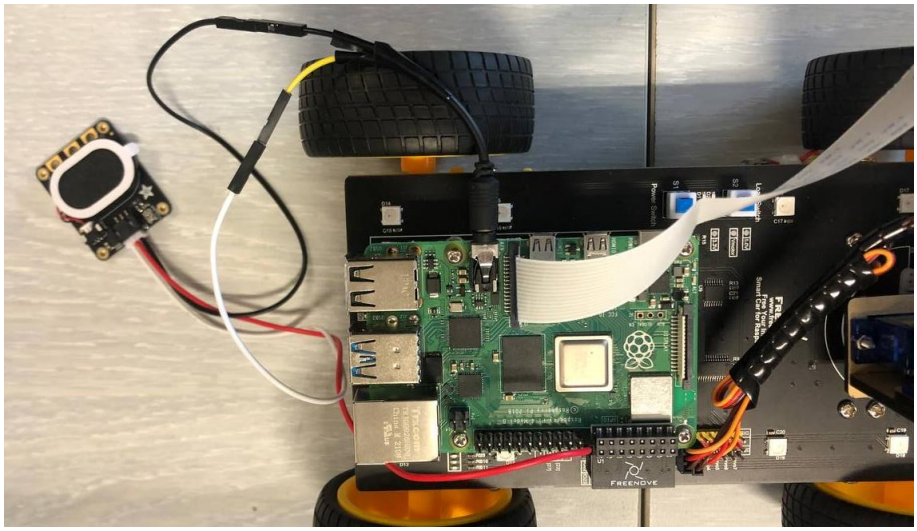

Figure 2-12: Speaker attached to the robot

2.6. EPSON BT-350 SMART GLASSES

In hopes to enhance the customers experience, an Epson BT350 smart glasses was use. This will be done by inserting the hosted URL of the web application into the smart glasses. The smart glasses take AR to the next level by allowing the user to be fully immersed with the viewing content.

Figure 2-13: Epson BT350 smart glasses (Epson, n.d.)

3. TELEPRESENCE ROBOT

Telepresence robots allow the user to control the movement of the robot throughout space in the remote location, or even communicate with objects in the secondary location by utilizing the arms of a robot. Therefore, it allows users to interact with their surrounding and people without being physically there themselves. These robots can potentially help with the mobility and adaptability of videoconferencing. In the healthcare industry, telepresence robots are used to reduce the exposure to infectious disease and allow patients to connect with their families. As Covid-19 pandemic becomes a growing issue, the use of such robots has become more relevant and important as ever.

3.1. AUGMENTATIVE REALITY

Augmented Reality (AR) is an enhanced version of the real physical world. It allows user to view the real-life environment using technology that makes use of visual, auditory, or other sensory elements to enhance their experience. The prime goal of implementing AR is to accentuate specific features of the real physical world, increase the understanding of these features, and obtain smart and accessible perception that can be applied to the real-world context. (Adam, 2020)

3.2. DIFFERENCE BETWEEN AUGMENTATIVE REALITY AND VIRTUAL REALITY

The most distinct difference between Augmented Reality (AR) and Virtual Reality (VR) is that AR uses a real-world setting while VR uses a virtual world setting. AR allows users to control their presence in the physical world where they can feel more connected and present. VR completely removes the physical world element and replaces it with a world that is simulated and animated, hence users are controlled by a system in the virtual world. AR

applications are best suited for remote assistance, on-the-job training, and computer-assisted tasks while VR applications include virtual tours, or 3D video games. (Tulane University, n.d.)

3.3. AMAZON WEB SERVICES (AWS)

Amazon web services (AWS) will be implemented to enable remote access to the robot. AWS is a cloud platform that offers multiple services from data centres globally. These services cover a wide range of functions such as security, storage, and networking. (Gillis, 2020)

3.4. ARCHITECTURE DIAGRAM

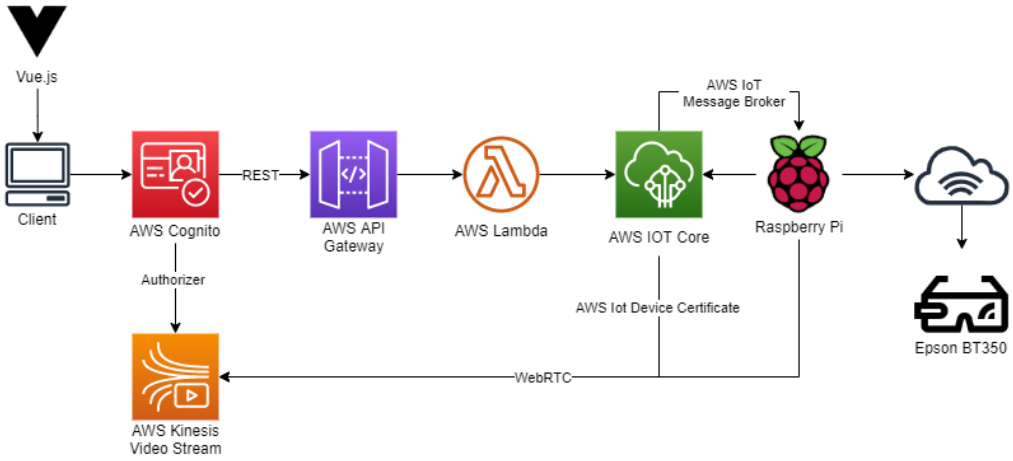

Figure 3-1: Architecture Diagram

3.5. ENHANCEMENT

Since this project was a continuation from last semester, some enhancements and changes in the approach was made. Firstly, the previous group made use of Flask Servers to communicate with the robot from the website. Therefore, the group had an additional task of managing these servers. This approach was replaced by a serverless approach which came with a lot of benefits. These benefits will be explored later in the report.

Current Project	Previous Project
Serverless Approach	Server Approach
Wide-area Network	Local-area Network
AWS KVS WebRTC (signalling channel)	AWS KVS (video stream)
Vue.js website	Firebase website

Figure 3-2: Project comparison

The video streaming of the robot could only be done in a local area network suggesting that the website and the robot had to be connected to the same Wi-Fi to be able to stream the video. This was because the previous group uses a Firebase server which only allows local hosting. Therefore, the website could not be hosted onto AWS. Furthermore, regarding the user authentication, Firebase authentication is powered under Google whereas AWS Amplify configured in Vue.js is powered under AWS. Therefore, by implementing third-party resources compatible with AWS and AWS services, the hosting and video streaming could be done over the wide-area network.

The current project made use of AWS KVS with WebRTC to stream the live video. Although it does not allow the ingesting of videos, WebRTC will allow for real time media communications directly between browser and devices. AWS WebRTC allows for a two-way media streaming which suggest that it is possible for the Website to stream media back to the Raspberry Pi.

3.6. SERVERLESS

Serverless is a cloud computing execution model that involves operating applications without having to manage servers. This refers to having developers to only write a code while the behind-the-scenes infrastructure is managed by the cloud providers. These cloud

providers are responsible for the allocation and provisioning of servers. Therefore, developers can focus solely on the individual function specified in the application code. (Parlette, 2018)

3.7. ADVANTAGE OF SERVERLESS

First, no server management is necessary. Developers do not have to deal with the management of servers as these are done by the vendor, this creates more time for developers to create and expand their applications without being pressured about the server capacity. Secondly, developers are only charged for the server space they use, leading to a decrease in cost. Like ‘pay-as-you-go’ phone plan, code only runs when the backend functions are needed by the serverless application and scales up automatically when needed. Therefore, developers are only charged for what they use. Thirdly, serverless architectures offer greater scalability. As the user base grows, the serverless architecture will scale automatically. Fourthly, quick deployments and updates can be made. Developers can upload their code all at once as the application is a collection of functions provisioned by the vendor, therefore updates or patches can be made immediately without the need to make changes to the whole application. Lastly, serverless architecture reduces latency as the code can run closer to the end user. Since the application is not hosted on an origin server, the code can be run on any server that are close to the end user. Hence, requests from the user will be delivered faster, reducing latency. (cloudflare, n.d.)

4. DEPLOYMENT

Following a blog post, the serverless backend was created and deployed using the GitHub AWS Serverless Application Model (SAM) source code provided and AWS Command Line Interface (CLI.) AWS SAM is an open-source framework use to build serverless application on AWS CloudFormation. AWS CloudFormation is an AWS service that provides developers to consolidate and build related AWS and third-party resources by treating infrastructure as code (Amazon Web Services, n.d.). The use of template files helps AWS CloudFormation to automate the setup of such resources making it easier and faster for developers to scale up. It allows for consistency when managing and configuring application and infrastructure as the same configuration can be used to deploy multiple copies of the stack. Therefore, it makes troubleshooting easier and helps minimise the error caused by configuring the resources manually.

To deploy the application on AWS CLI, SAM CLI, Docker and Node.js was also needed. SAM CLI is an extension of AWS CLI that adds functionality for building and testing Lambda application. Docker is an open-source containerisation platform that allow developers to store applications into containers. Docker was responsible for running the functions in the Amazon Linux environment that matches Lambda.

```
CloudFormation outputs from deployed stack
---------------------------------------------------------------------------------------------------------
Outputs
---------------------------------------------------------------------------------------------------------
Key                 ApiURL
Description         API endpoint URL for Prod environment
Value               https://ttwkebxdw8.execute-api.ap-southeast-1.amazonaws.com/Prod/

Key                 EndpointUrl
Description         The IoT Credentials Provider Endpoint
Value               c3ipei6doxxf9f.credentials.iot.ap-southeast-1.amazonaws.com
---------------------------------------------------------------------------------------------------------

Successfully created/updated stack - Rpirobot-app in ap-southeast-1


C:\Users\ASUS\OneDrive\Desktop\aws-serverless-telepresence-robot-master>sa
```

Figure 4-1: URL and Endpoint output

After deploying of the SAM template, a IoT get credential endpoint and API URL will be printed out. This API URL will be copied onto a config.json file in the Vue web application to invoke a Lambda function that will publish messages to the MQ Telemetry Transport (MQTT) Topic for the robot movement. Whereas, the endpoint will be copied onto a config,json file on the Raspberry Pi along with the IoT thing name, the IoT core endpoint, the role alias created for the AWS KVS and the AWS Default Region.

config.json

```
{
  "IOT_THINGNAME": "Rpirobot",
  "IOT_CORE_ENDPOINT": "a1mflxxq8bh5gw-ats.iot.ap-southeast-1.amazonaws.com",
  "IOT_GET_CREDENTIAL_ENDPOINT": "cbbz841phtb9a.credentials.iot.ap-southeast-1.amazonaws.com",
  "ROLE_ALIAS": "robot-camera-streaming-role-alias",
  "AWS_DEFAULT_REGION": "ap-southeast-1"
}
```

Figure 4-2: config.json file

An AWS IoT thing will also be created with the responding robot name, “Rpirobot”. For the robot to authenticate and connect with AWS IoT Core, an AWS IoT certificate was created. This in which produced a private key, certificate, and root certification authority (CA) which will be uploaded onto the Raspberry Pi. A root CA is the cornerstone authentication and security on the internet.

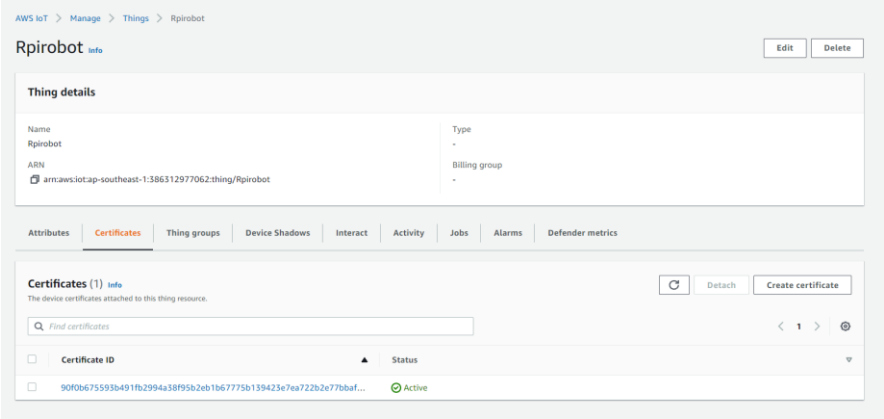

Figure 4-2: AWS IoT Thing and Certificate

4.1. AUTHETICATION

After the deployment of the new stack, several steps must be taken to ensure the security of the project. To achieve this, authentication was added for the movement and the streaming of the video.

4.2. MOVEMENT

To control the movement of the robot, API Gateway was used to create a REST endpoint to allow messages to be sent between the website and the robot as seen in Figure 3-1. Initially, anyone can send a request using the endpoint without authentication. Therefore, using the Cognito pool created in Figure 6-1, an Authorizer was created to limit access to the API Gateway.

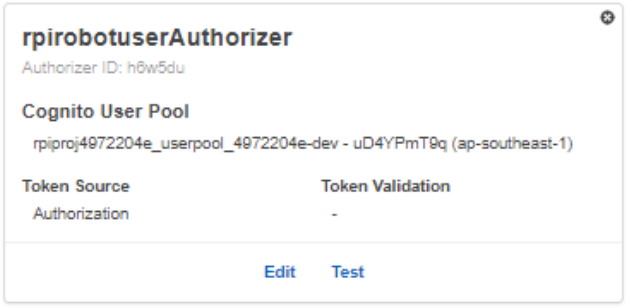

Figure 4-3: Authorizer created with corresponding Cognito pool

The authorizer created above was used to authorize both robots' API Gateways as shown below. This will ensure that only authenticated users will be able to control the robot.

Figure 4-4: Authorization using POST request

4.3. STREAMING OF VIDEO

When the stack was deployed, a role was created in AWS Identity and Access Management (IAM) which is used by Cognito to assume permissions for KVS to access the signalling channel. To ensure the user can only assume this role when they are authenticated, a Inline policy was created and attached to this role. As seen in the figure below, one policy was created for each robot and named "authRole" and "authrole2".

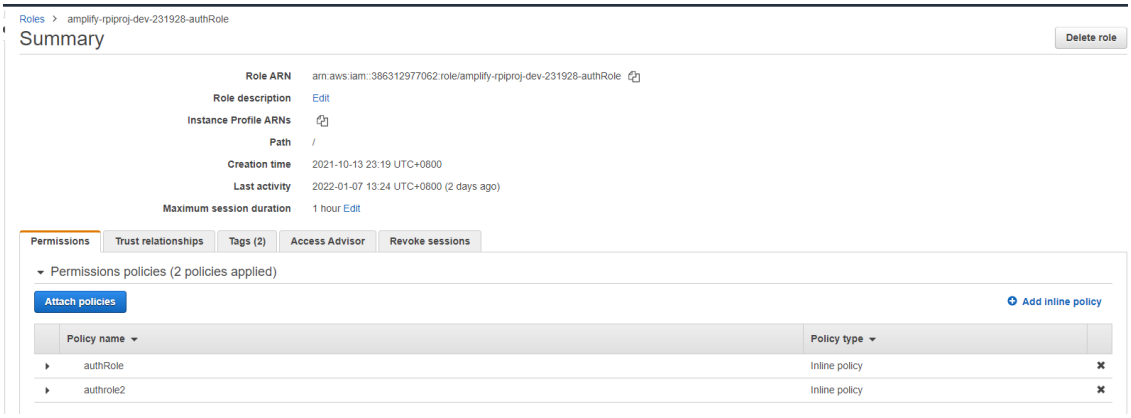

Figure 4-5: Inline Policies

Once the policies have been added, the "authenticated" user from the identity pool will be able to acquire the role with the permissions shown in the figure below and access the desired signalling channel. The <RobotName> will be replaced with the corresponding name of the robots. Therefore, the streaming of video will be more secured.

```JSON
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetSignalingChannelEndpoint",
                "kinesisvideo:ConnectAsMaster",
                "kinesisvideo:GetIceServerConfig",
                "kinesisvideo:ConnectAsViewer",
                "kinesisvideo:DescribeSignalingChannel"
            ],
            "Resource": "arn:aws:kinesisvideo:*:*:channel/<RobotName>/*"
        }
    ]
}
```

Figure 4-6: Permissions added to Policies

4.4. STATIC HOSTING OF THE WEBSITE

To host the Vue.js web application online, “amplify add hosting” command is used to configure the application, allowing all the resources created in Vue.js to automatically be deployed using S3 and CloudFront. An S3 bucket is created once hosting is added, it is a simple storage service where data can be stored, retrieved, accessed, and backed up at any time. Amazon CloudFront enables web application to be delivered through a worldwide network of data centres, distributing all static and dynamic web content to users. (Amazon Web Services, n.d.) An “amplify publish” command is then used to build and publish all backend and frontend resources of the web application. Finally, the URL of the hosted web application will be provided.

4.5. PRICING

As the project made use of various AWS services, there is a cost behind consuming these services. However, this relatively affordable as AWS offers you a “pay-as-you-go” approach for pricing for over 160 cloud services. Therefore, only the individual services that are needed will be charged. AWS provides different types of free tier offers. This includes short

term “Free trails” tier, “12-month free” tier, and “Always free” tier (Amazon Web Services, n.d.).

4.5.1. ALWAYS FREE

The AWS services used in this project that falls under the “Always free” tier is AWS CloudFront where 1 Terabyte of date transferred out is free. AWS CloudFormation is also under the Always Free Tier which includes free 1,000 handler operations per month per account. Handler operations are to CREATE, UPDATE, DELETE, READ, or LIST actions on a resource. For one robot, 20 handler operations are utilised, thus, 1000 handler messages will be enough for 50 robots to be created in a month for free. AWS Lambda is also classified under this tier. This allows for 1 million Lambda request per month. Another service under this tier is AWS Cognito which provides a free tier of 50,000 monthly active users (Amazon Web Services, n.d.).

4.5.2. 12-MONTHS FREE TIER

This free tier offers free AWS services for 12-months following the initial sign-up date of the AWS account.

4.5.2.1. API GATEWAY

API Gateway is a service offered under this tier that include 1 million free API Calls to be received per month. However, after the 12-months, the REST API request per month will be charged as seen in the figure below. Therefore, the higher the number of Request, the lower the cost. (Amazon Web Services, n.d.)

API Calls

Number of Requests (per month)	Price (per million)
First 333 million	$4.25
Next 667 million	$3.53
Next 19 billion	$3.00
Over 20 billion	$1.91

Figure 4-7: REST API prices (Amazon Web Services, n.d.)

4.5.2.2. IOT CORE

Since the robots connect to AWS as IoT Core things, IoT Core services such as IoT message broker are being used. Under the 12-month Free Tier, 250 thousand messages can be delivered or published per month for free. Without this offer, AWS IoT core charges $1.00 per million messages and $0.08 per million minutes of connectivity. (TrustRadius, n.d.)

4.5.3. AWS KVS WEBRTC

Currently, the video streaming is done using AWS Kinesis Video Stream Using WebRTC. AWS KVS WebRTC is not under any free tier and charges SGD $0.04 for the number of signalling channel that are active in each month. (Amazon Web Services, n.d.) If a device or application is connected to the signalling channel during any period of the month, the signalling channel is considered active. Since each robot uses one signalling channel, the cost of video streaming depends on the number of robot set-up. In this case, it would be a total of SGD $0.08, making this project relatively affordable to set-up and implement.

Amazon Kinesis Video Streams	$0.00
Charges	$0.08
Credits	($0.08)
VAT **	$0.00
GST	$0.00

Figure 4-8: Invoice for December

4.6. SCALABILITY

In hopes to enhance the project, an additional robot was built. This will display the scalability and the improvement of security in the project. Using authentication, one user will only to be connected to their own robot. One benefit of utilising AWS CloudFormation is that the same configuration can be used to deploy multiple copies of the same stack. Once the robot was set-up, a new stack, consisting of a different name, “Rpirobot2-app”, and a different IoT thing name, “Rpirobot2” was be deployed. This will output a different API Gateway URL from the URL printed out in figure 4-1. The certificate attached to this new robot will have different certificate.pem and private.pem.key files that are uploaded onto the new robot.

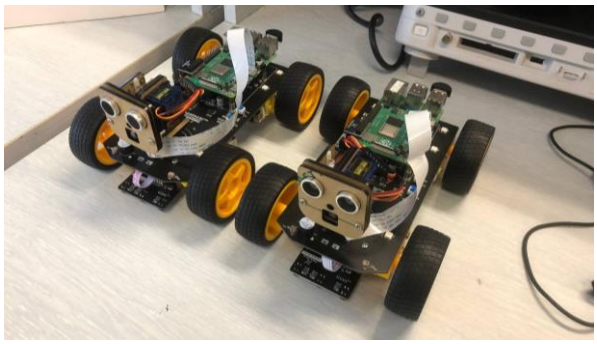

Figure 4-9: Rpirobot and Rpirobot2

5. ROBOT MECHANISM (BACK-END)

The main goal of the project was to allow the user to control the Raspberry Pi robot remotely via wide-area network. Therefore, AWS services were used to communicate between the user on the website and the Raspberry Pi.

```
main.py *×
29 with open('./config.json') as json_file:
30     config = json.load(json_file)
31
32 host = config['IOT_CORE_ENDPOINT']
33 clientId = config['IOT_THINGNAME']
34 topic = clientId + '/action'
35
36 rootCAPath = './certs/cacert.pem'
37 certificatePath = './certs/certificate.pem'
38 privateKeyPath = './certs/private.pem.key'
39 port = 443
40 useWebsocket = False
```

Figure 5-1: main.py code to extract desire information

In the main.py, the information provided in the config.json file and the IoT certificate files will be retrieved using the commands above and will be used in the code accordingly. This information, which was unique to each robot, was used to configure the AWS MQTT client endpoint and credentials to authenticate the connection between the Raspberry Pi and AWS IoT Core. This connection was done using the "awsClient.connect()" command as shown in the figure below.

```
main.py *×
95              pwm.setServoPwm('1',90)
96
97  # Init AWSIoTMQTTClient
98  awsClient = AWSIoTMQTTClient(clientId)
99  awsClient.configureEndpoint(host, port)
100 awsClient.configureCredentials(rootCAPath, privateKeyPath, certificatePath)
101
102 # AWSIoTMQTTClient connection configuration
103 awsClient.configureAutoReconnectBackoffTime(1, 32, 20)
104 awsClient.configureOfflinePublishQueueing(-1)  # Infinite offline Publish queueing
105 awsClient.configureDrainingFrequency(2)  # Draining: 2 Hz
106 awsClient.configureConnectDisconnectTimeout(10)  # 10 sec
107 awsClient.configureMQTTOperationTimeout(5)  # 5 sec
108 awsClient.onMessage = handleMessage
109
110 # Connect and subscribe to AWS IoT
111 awsClient.connect()
112 # Note that we are not putting a message callback here. We are using the general message notification callback.
113 awsClient.subscribeAsync(topic, 1)
114 time.sleep(2)
115
116 # Start the Kinesis Video Gstreamer Sample App using IoT Credentials
117 runKinesisVideoStream()
118 time.sleep(1)
119
120 while True:
121     time.sleep(0.2)
```

Figure 5-2: main.py code to connect Raspberry Pi as MQTT Client

5.1. MOVEMENT

One of the objectives of our project was to have the user be able to control the robots' movements. This refers to adjusting of the robot's camera angle which is controlled by the servo and driving the car in the desire direction which is controlled by the DC motor. Both robot parts are shown in Figure 2-3.

Version 1 **Edit policy document**


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iot:Connect",
        "iot:Subscribe",
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:*:*:topicfilter/Rpirobot/action",
        "arn:aws:iot:*:*:topic/Rpirobot/action",
        "arn:aws:iot:*:*:topic/Rpirobot/telemetry",
        "arn:aws:iot:*:*:client/Rpirobot"
      ],
      "Effect": "Allow"
    }
  ]
}
```


Figure 5-3: IoT policy

A IoT policy was created and attached to the created IoT thing, in this case was "Rpirobot", to allow the robot to connect as an IoT thing and to subscribe to a specific MQTT topic, as shown in the figure above was "Rpirobot/action". As shown in Figure 5-2, this subscription was done using the command "awsClient.subcribeAsync()". AWS IoT Core Message broker will be responsible for sending the messages published onto the topic to the robot to control its movement. According to the messages retrieved, the robot will move in the desired direction or camera angle.

5.1.1. PYTHON CODE

On the robot end, the command “awsClient.onMessage = handleMessage” will retrieve the message on the MQTT topic using the listener “onMessage” method as seen in Figure 5-2. The message will be stored in the variable “message” which will be interpreted using the function “handleMessage ()”.

main.py *

```
51 # Handle incoming messages and take action
52 def handleMessage(message):
53     print("Received a new message: ")
54     print(message.payload)
55     print("from topic: ")
56     print(message.topic)
57     print("--------------\n\n")
58     if message.topic == topic:
59         payload = json.loads(message.payload)
```

Figure 5-4: handleMessgae() code

The command “payload = json.loads(message.payload)” is responsible for parsing the desire part of the transmitted data which is a JSON string and converting it into a Python dictionary. The command “payload['action']” was used to extract the value stored under the key called ‘action’ in the dictionary. The possible values received were named corresponding to the desired movement of the robot. For example, if the value obtained was “moveleft” the DC Motors will move the wheels to the left. Similarly, if the value was “lookright”, the servo will move the camera angle to the right.

Figure 5-5: Received Message statement

As seen in Figure 5-5, when a message was received from the website via the REST API and AWS message broker, a "Received a new message" statement, which includes the message obtained and the topic name, was printed out. This aids with the confirmation of the correct message received and the troubleshooting if messages are not printed out.

```
from Motor import *
PWM=Motor()

from servo import *
pwm=Servo()
```

Figure 5-6: main.py code to import servo.py and motor.py

To integrate the Freenove code control the robot according to the specified value obtained, it had to be analysed and understood. Both motor.py and servo.py code was first imported into the main.py to allow the DC motors and servo to run.

```
# Servo
if payload['action'] == 'lookright': #lookright
    for i in range(0,180,1):
        pwm.setServoPwm('0',i)
        time.sleep(0.01)
if payload['action'] == 'lookleft':#lookleft
    for i in range(180,0,-1):
        pwm.setServoPwm('0',i)
        time.sleep(0.01)
if payload['action'] == 'up':  #up
    for i in range(80,150,1):
        pwm.setServoPwm('1',i)
        time.sleep(0.01)
if payload['action'] == 'down': #down
    for i in range(150,80,-1):
        pwm.setServoPwm('1',i)
        time.sleep(0.01)
if payload['action'] == 'reset': #down
    pwm.setServoPwm('0',50)
    pwm.setServoPwm('1',80)
```

Figure 5-7: handleMessage() code to control servo

For the servos, a for loop was used to allow the servo to gradually move over a range of angles. Within the loop, the "pwn.setServoPwm()" function, located in the servo, was called and used to position the desired servo at the angle determined by the variable "i". The value of variable "i" will start from the first angle stated and gradually increase or decrease to the second angle stated in the loop conditions.

servo.py

```
import time
from PCA9685 import PCA9685
class Servo:
    def __init__(self):
        self.PwmServo = PCA9685(0x40, debug=True)
        self.PwmServo.setPWMFreq(50)
        self.PwmServo.setServoPulse(8,1500)
        self.PwmServo.setServoPulse(9,1500)
    def setServoPwm(self,channel,angle,error=10):
        angle=int(angle)
        if channel=='0':
            self.PwmServo.setServoPulse(8,2500-int((angle+error)/0.09))
        elif channel=='1':
            self.PwmServo.setServoPulse(9,500+int((angle+error)/0.09))
        elif channel=='2':
            self.PwmServo.setServoPulse(10,500+int((angle+error)/0.09))
        elif channel=='3':
            self.PwmServo.setServoPulse(11,500+int((angle+error)/0.09))
        elif channel=='4':
            self.PwmServo.setServoPulse(12,500+int((angle+error)/0.09))
        elif channel=='5':
            self.PwmServo.setServoPulse(13,500+int((angle+error)/0.09))
        elif channel=='6':
            self.PwmServo.setServoPulse(14,500+int((angle+error)/0.09))
        elif channel=='7':
            self.PwmServo.setServoPulse(15,500+int((angle+error)/0.09))

```

Figure 5-8: servo.py code

Within the "pwn.setServoPwm()" function, the channel number stated controlled which servo will move. Channel "1" is responsible for the servo that moves the camera up and

down while channel “0” is for the servo that moves the camera right and left. The “setServopulse()” function, located in the PCA9685.py, was then called.

main.py

```
        # Motor
        if payload['action'] == 'forward': #forward
            PWM.setMotorModel(1000,1000,1000,1000)
        if payload['action'] == 'backwards':#back
            PWM.setMotorModel(-1000,-1000,-1000,-1000)
        if payload['action'] == 'moveleft': #moveleft
            PWM.setMotorModel(-1500,-1500,2000,2000)
        if payload['action'] == 'moveright': #moveright
            PWM.setMotorModel(2000,2000,-1500,-1500)
        if payload['action'] == 'stop': #stop
            PWM.setMotorModel(0,0,0,0)

        time.sleep(0.2)
        PWM.setMotorModel(0,0,0,0)
```

Figure 5-9: handleMessage () code that controls Motor

For the DC motor programme, the “PWM.setMotorModel()” function, located in the motor.py, was called. Four different variables were also inputted, each controlling the DC motor for each wheel. This function called upon the “setMotorPwm()” function found in the PCA9685.py.

Motor.py

```
            self.pwm.setMotorPwm(7,duty)
        elif duty<0:
            self.pwm.setMotorPwm(7,0)
            self.pwm.setMotorPwm(6,abs(duty))
        else:
            self.pwm.setMotorPwm(6,4095)
            self.pwm.setMotorPwm(7,4095)
    def right_Lower_Wheel(self,duty):
        if duty>0:
            self.pwm.setMotorPwm(4,0)
            self.pwm.setMotorPwm(5,duty)
        elif duty<0:
            self.pwm.setMotorPwm(5,0)
            self.pwm.setMotorPwm(4,abs(duty))
        else:
            self.pwm.setMotorPwm(4,4095)
            self.pwm.setMotorPwm(5,4095)


    def setMotorModel(self,duty1,duty2,duty3,duty4):
        duty1,duty2,duty3,duty4=self.duty_range(duty1,duty2,duty3,duty4)
        self.left_Upper_Wheel(-duty1)
        self.left_Lower_Wheel(-duty2)
        self.right_Upper_Wheel(-duty3)
        self.right_Lower_Wheel(-duty4)
```

Figure 5-10: motor.py code

Both servo.py and motor.py require the use of PCA9685.py which contain codes that sets the Pulse Width Modulation (PWM). PCA9685 is a 16-channel I2C-bus controlled PWM controller with a fixed frequency. PWM is method that helps to minimise the average power

supplied by a electrical signal. PWM controlled the DC motor and servo using series of "ON-OFF" pulses that vary depending on the variable "channel" inputted as seen in the "setPWM ( )" function shown below.

PCA9685.py

```
51
52      oldmode = self.read(self.__MODE1);
53      newmode = (oldmode & 0x7F) | 0x10        # sleep
54      self.write(self.__MODE1, newmode)        # go to sleep
55      self.write(self.__PRESCALE, int(math.floor(prescale)))
56      self.write(self.__MODE1, oldmode)
57      time.sleep(0.005)
58      self.write(self.__MODE1, oldmode | 0x80)
59
60    def setPWM(self, channel, on, off):
61      "Sets a single PWM channel"
62      self.write(self.__LED0_ON_L+4*channel, on & 0xFF)
63      self.write(self.__LED0_ON_H+4*channel, on >> 8)
64      self.write(self.__LED0_OFF_L+4*channel, off & 0xFF)
65      self.write(self.__LED0_OFF_H+4*channel, off >> 8)
66    def setMotorPwm(self,channel,duty):
67      self.setPWM(channel,0,duty)
68    def setServoPulse(self, channel, pulse):
69      "Sets the Servo Pulse,The PWM frequency must be 50HZ"
70      pulse = pulse*4096/20000        #PWM frequency is 50HZ,the period is 20000us
71      self.setPWM(channel, 0, int(pulse))
72
73  if __name__=='__main__':
74      pass
75
```

Figure 5-11: PCA9685.py code

5.2. VIDEO STREAMING

The other main requirement for the project to allow the user to view a live video streaming from the Raspberry Pi camera. This requirement was met using AWS KVS with WebRTC. WebRTC is an open-source project and specification that allow web applications to capture and stream audio or video media.

To allow authentication with KVS using IoT credentials, a role alias policy was created and attached to the certificate of the IoT Thing.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iot:Connect",
        "iot:AssumeRoleWithCertificate"
      ],
      "Resource": "arn:aws:iot:ap-southeast-1:386312977062:rolealias/rob
      "Effect": "Allow"
    }
  ]
}
```

Figure 5-12: Role alias policy

Once attached, the IoT thing, the robot, will be able to assume the role as stated in the code below with the correct IoT certificate credentials. This role will allow access to the robot to connect and transmit the captured video using WebRTC and onto a signalling channel.

```
KVSCertificateBasedIAMRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: 'Allow'
          Principal:
            Service: 'credentials.iot.amazonaws.com'
          Action: 'sts:AssumeRole'
      Policies:
      - PolicyName: !Sub "KVSIAMPolicy-${AWS::StackName}"
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
          - Effect: Allow
            Action:
              - kinesisvideo:ConnectAsMaster
              - kinesisvideo:GetSignalingChannelEndpoint
              - kinesisvideo:CreateSignalingChannel
              - kinesisvideo:GetIceServerConfig
              - kinesisvideo:DescribeSignalingChannel
            Resource: "arn:aws:kinesisvideo:*:*:channel/${credentials-iot:ThingName}/*"
```

Figure 5-13: Cloudformation template.yaml code for role alias

Once the python code is run and the connection is authenticated, the video stream and viewer statistics can be view on the AWS KVS signalling channel console as shown in Figure 5-14.

For video streaming, H264 frames are send from the master (Raspberry pi) to the viewer with the frame rate of 30 frames per second (fps) which an ideal fps for live video streaming. It can handle basic motion and does not involve a significant amount of bandwidth (Milazzo, 2021). While, for audio streaming, Opus audio codec was utilised.

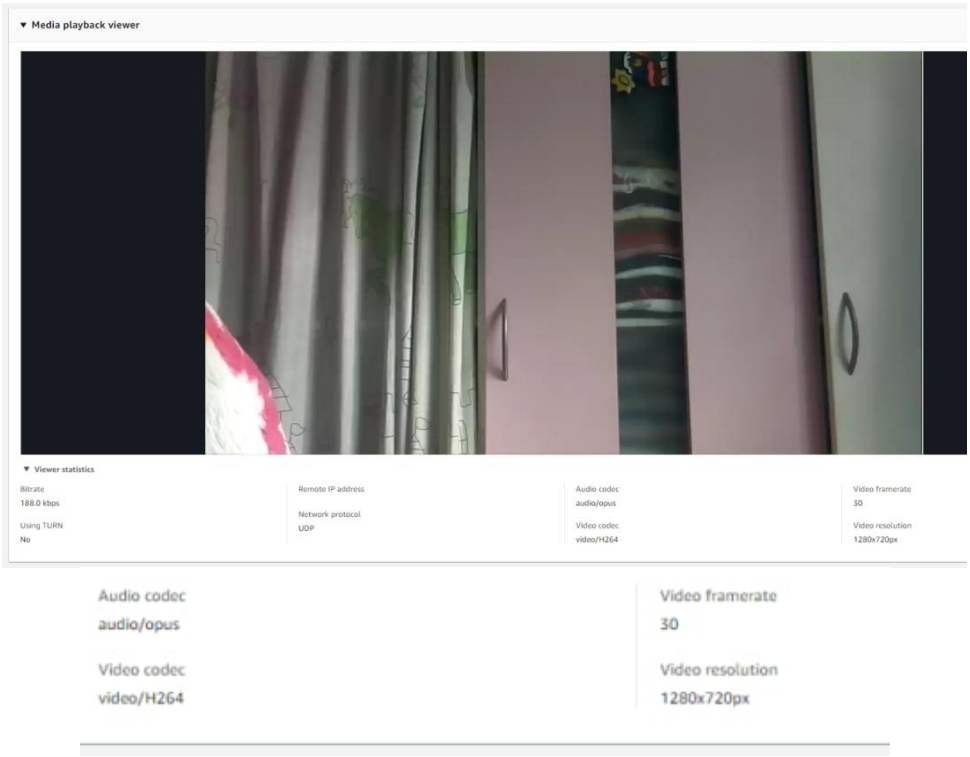

Figure 5-14: AWS KVS Viewer statistics

5.2.1. PYTHON CODE

```
main.py
28 def runKinesisVideoStream():
29     environmentVars  = 'IOT_GET_CREDENTIAL_ENDPOINT=' + config['IOT_GET_CREDENTIAL_ENDPOINT']
30     environmentVars += ' ROLE_ALIAS=' + config['ROLE_ALIAS']
31     environmentVars += ' AWS_DEFAULT_REGION=' + config['AWS_DEFAULT_REGION']
32     environmentVars += ' DEFAULT_KVS_CACERT_PATH=' + rootCAPath
33
34     command = environmentVars + ' ./kvsWebrtcClientMasterGstSample ' + clientId + ' audio-video'
35     proc = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
```

Figure 5-15: main.py runKinesisVideoStream()

On the Raspberry Pi, a “wget” command was used to fetch and execute the installer script to download all the necessary files needed on the Raspberry Pi. This included “kvsWebrtcClientMasterGstSample” application which is sends sample H264/Opus frames

from a GStreamer pipeline and playback incoming audio via an autoaudiosink. Gstreamer is a open source multimedia tool used for building streaming pipelines. When constructing the command to initialise the build of the sample, the order and type of parameter will determine the media-type of the streaming as shown in the figure below. Therefore, by adding 'audio-video' as the second parameter, both audio and video will be stream to the viewer(website).

```
if (argc > 2) {
    if (STRCMP(argv[2], "video-only") == 0) {
        pSampleConfiguration->mediaType = SAMPLE_STREAMING_VIDEO_ONLY;
        printf("[KVS Gstreamer Master] Streaming video only\n");
    } else if (STRCMP(argv[2], "audio-video") == 0) {
        pSampleConfiguration->mediaType = SAMPLE_STREAMING_AUDIO_VIDEO;
        printf("[KVS Gstreamer Master] Streaming audio and video\n");
    } else {
        printf("[KVS Gstreamer Master] Unrecognized streaming type. Default to video-only\n");
    }
} else {
    printf("[KVS Gstreamer Master] Streaming video only\n");
}
```

Figure 5-16: Code found in kvsWebRTCClientMasterGstreamerSample (github, n.d.)

As mentioned earlier, a config.json file was created and uploaded into the Raspberry Pi. The environmental variables provided in the file along with the root CA path will be used to determine the value of the variable "environmentVars".

The subprocess was initiated using the command "subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)". This will spawn an intermediate shell process and tell it to run the command provided which included the environmental variables, the KVS application and the robot's name. This command will initiate the creation of a signalling channel using the robot's name and allow the robot to stream the video and audio captured onto it.

5.3. CHALLENGES

The first challenge would be deploying the CloudFormation stack. As recommended by the blog post, a link was provided to deploy the stack using and already created template application on CloudFormation. However, this template could not be found in the AWS CloudFormation console. Therefore, the stack had to be deployed using AWS and SAM CLI, Node.js and Docker. When following the steps to build the application, an error with code "ENOENT" was encountered as seen in the figure below. This suggested that a "package.json" file could not be found. This file is required by Node.js to document the necessary package needed to build and run the application. Initially, it was thought that maybe the directory path to downloaded github source code was too long. However, after shortening it, the error was still occurring. After some research, it was discovered that the command "npm init" "npm init" can be used to initialise the project and the "package.json" file will be automatically created in the desired project directory. Once added, the application could be successfully built and deployed.

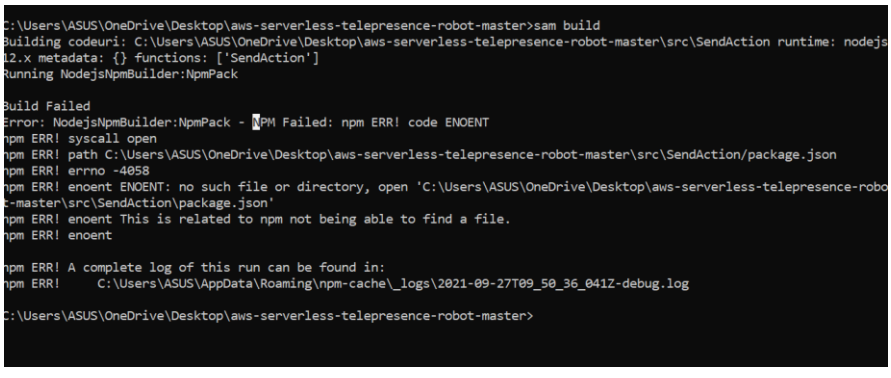

Figure 5-17: ENOENT error message

Once the signalling channel was created, the video captured by the robot should be displayed on the media playback card. However, this was not the case. A WebRTC peer to peer connection was not negotiated therefore the live video was not displayed. To gain a

better understanding to troubleshoot, a different blog post was used to create a video stream using AWS KVS producer Software Development Kit (SDK) for C++ instead of WebRTC. AWS KVS WebRTC is for real time communication and data cannot be ingested into the cloud for storage and processing. However, KVS with Producer SDK can ingest audio and video and throttles depending on the number fragments required instead of number of playback session (Christie, 2021). As the video was able to stream properly, it can be inferred that there was no error with the Raspberry Pi capturing video and streaming it onto AWS KVS. Therefore, this suggested that the issue could be caused by a technical error. After more research was done, it was discovered that the URL used for the IoT Credential Endpoint was wrong. This was due to the lack of understanding of the steps when following the instructions and carelessness. Once corrected, the video capture was displayed onto the signalling channel.

When dealing with both hardware and software, challenges will arise that require troubleshooting and research skills. Firstly, after connecting the speaker, it could not be detected on the Raspberry pi board and "speaker-test" command did not work. Initially thinking it was a connection issue, a multi-meter was used to ensure that power was being supplied to the speaker. As the test was successful, research was made online to source for potential solutions. Finally, a forum mentioning that the new Raspbian Operating System (OS) has various audio issues was discovered. Since there were two raspberry pi boards, the other board was checked, and it was discovered that the speaker was able to be detected and tested successfully on the other board. This meant that the OS of both boards were different. The "cat /etc/debian_version", "cat /etc/os-release" and "uname -a" commands were ran to check the Debain, OS Release and Kernel Version of both boards. It was identified that the Debain and Kernel version on both boards were different. However, attempting to

downgrade the Kernel of the faulty board to the correct one failed. Therefore, a new SD card was used to download the correct OS and inserted into the robot. Fortunately, this solved the audio issue, and the speaker could be detected.

Another hardware problem encountered was that the camera module was not being detected on one of the robots. When trying to run the command “raspistill -o image.jpg”, an “mmal: No data received from sensor. Check all connections, including the Sunny one on the camera board” error was shown. To narrow down the root cause of the error, the connection of the sunny connector was checked to ensure it was not loose. Next was to test if the camera was faulty by connecting to the other board. However, on the other board, the camera could capture an image. Another camera was attached to the faulty board, but the same error was received. The SD card was also exchanged with the other board but the command “raspistill -o image.jpg” was successful. Therefore, it could be concluded that it was not a software issue but that the camera connector on the board was faulty which could cause the connection between the FPC wire and the board to be loose.

Finding a compatible microphone and speaker for the robot was not easy. Firstly, due to the design of the robot, it was hard to implement the usage of sound cards with build in speakers and microphone as they require all the GPIO pins. Therefore, the USB port and the audio jack on the raspberry p had to be utilised. Eventually, a blog post that uses a mini-USB microphone and a Adafruit STEMMA speaker (Figure 2-10) was found. (SneakyHacker, 2020). However, after some testing, it was found out that the mini-USB microphone was not strong enough to pick up audio from 1m away. This was not ideal especially if the robot was placed on the floor and had to pick up audio from the people standing up. Hence, research was done to find microphones that are more sensitive and utilises the USB port. The ReSpeaker Mic Array v2.0 was chosen to replace the Mini-USB microphone.

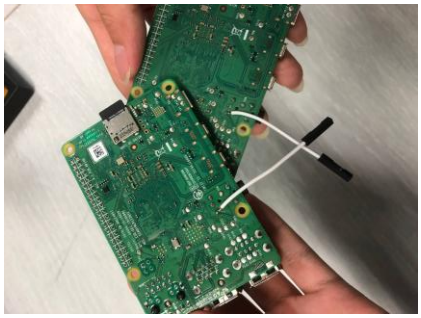

Figure 5-18: Jumper wire soldered onto board

Finding the speaker and the microphone was one problem but another was how to connect the speaker to the board. Initially the audio output wire of the Adafruit STEMMA speaker (figure) connected to the left-pin of the audio jack port. To do this, a jumper wire, with the female header pin on one end was cut and soldered it onto the back of the raspberry pi. The audio pin of the speaker was then attached to this jumper wire. However, this wire could pose as a risk of short circuiting the board as there could be traces of exposed wire. These expose wire could accidentally touch the wires on the raspberry pi board, thus, spoiling the board and jeopardising the project. Therefore, it was decided to change this approach to the current one. Through this, the importance of being cautious while working with hardware and identifying the risk and possibilities before making connections was learnt.

6. WEBSITE (FRONT-END)

In this project, the website will make use of the Vue.js Framework. Vue is a progressive JavaScript framework for building user interfaces and single-page applications, together with the knowledge of HTML and CSS. Compared to other frameworks such as React or Angular, Vue has the fastest learning curve to build web applications which is its trademark. Together with this framework, AWS services are implemented as well to allow user authentication. (Azam, 2021)

6.1. AWS AMPLIFY

AWS Amplify is a set of tools and services that makes it fast and easy to build and design a full-stack applications for web application developers. It consists of code libraries, ready-to-use components, and built-in CLI which supports quickness and efficiency in building these web applications. Amplify has components for first, data storage. It keeps any app data synced to the cloud and manages the distributed data. Secondly, analytics. Amplify tracks the user sessions and is accountable to any behaviour that is made. It also enables any custom attributes and inspects conversion funnels. Thirdly, push notifications. Amplify manages the campaigns and send messages to users across multiple channels, including text or email. Lastly, authentication. Ready-to-use workflows are accessed for multi-factor authorization (MFA), sign-in, sign-up pages etc. One authentication service used together with AWS Amplify is Amazon Cognito which we will talk about at the next section. (Matador, 2020)

6.2. AWS COGNITO

Amazon Cognito is a service that allows web developers to securely manage and synchronize the web application data for users. Unique identities or user pools are created for users through several public login providers such as Amazon, Google, or Facebook etc, it also supports unauthenticated guests. Furthermore, with Amazon Cognito, any data would be saved in the AWS cloud without the need to write any backend code, allowing the web developers to focus solely on creating web experiences instead of dwelling upon the backend solution to handle the management of identity, state, and storage. (Amazon Web Services, 2014)

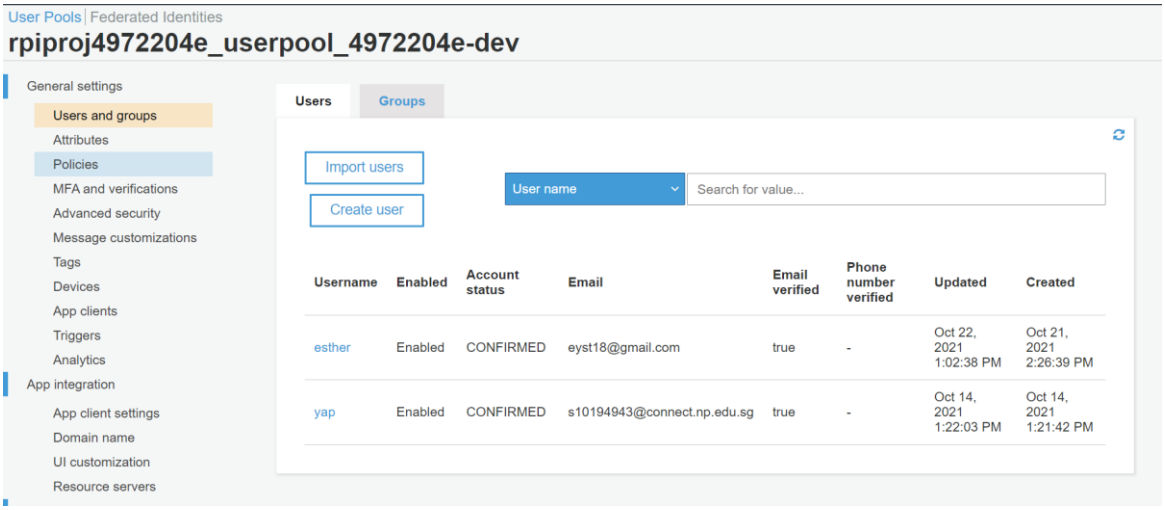

Figure 6-1: User pools created in AWS Cognito

6.3. AMAZON DYNAMODB

Amazon DynamoDB is a NoSQL database service that allows an efficient and predictable performance with large scalability capabilities. It prevents the need to worry about hardware provisioning, setup and configuration, replication, or cluster scaling. DynamoDB allows users to create database tables that can store and retrieve any amount of data, serving any level of request traffic. (Amazon Web Services, n.d.)

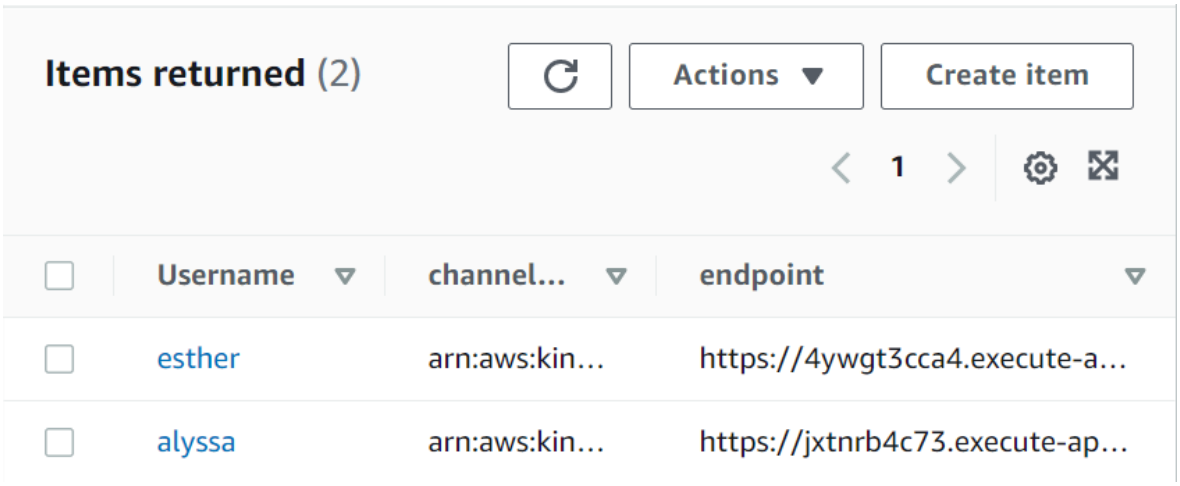

Figure 6-2: An example of user information stored in DynamoDB table

6.4. USER INTERFACE

The home page of the website consists of the sign-in page.

6.4.1. SIGN-IN PAGE

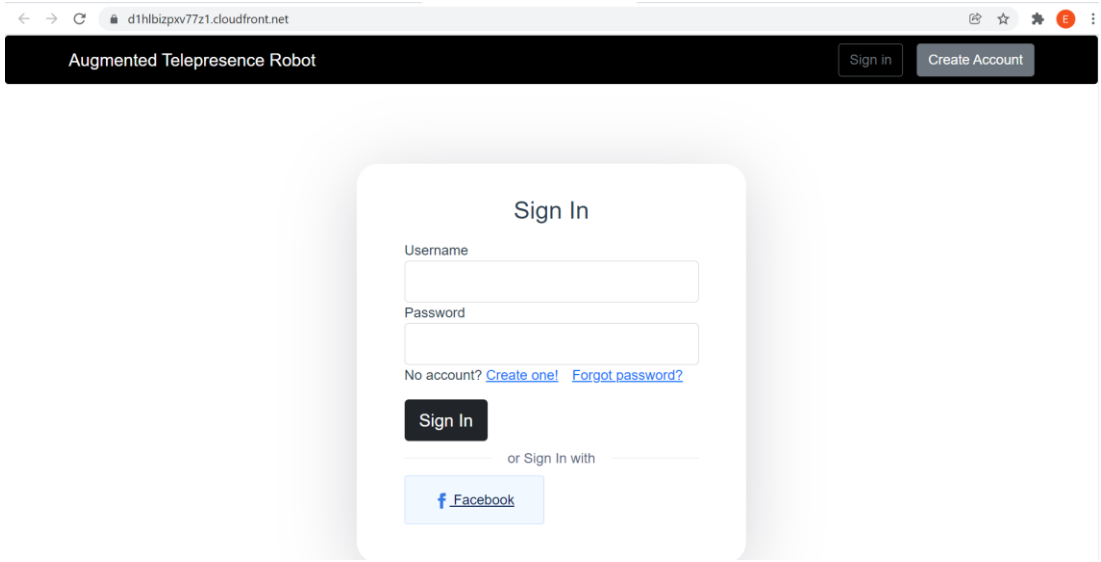

Figure 6-3: Sign-in page

If the user has no account, it can click to “Create one!” which will direct it to the Sign-up page. If the user already has an account but forgets its password, it can click to “Forgot password?” which will redirect it to the Forgot Password page. If the user wishes to sign in with a social provider, Facebook login is available for him/her to sign in as well.

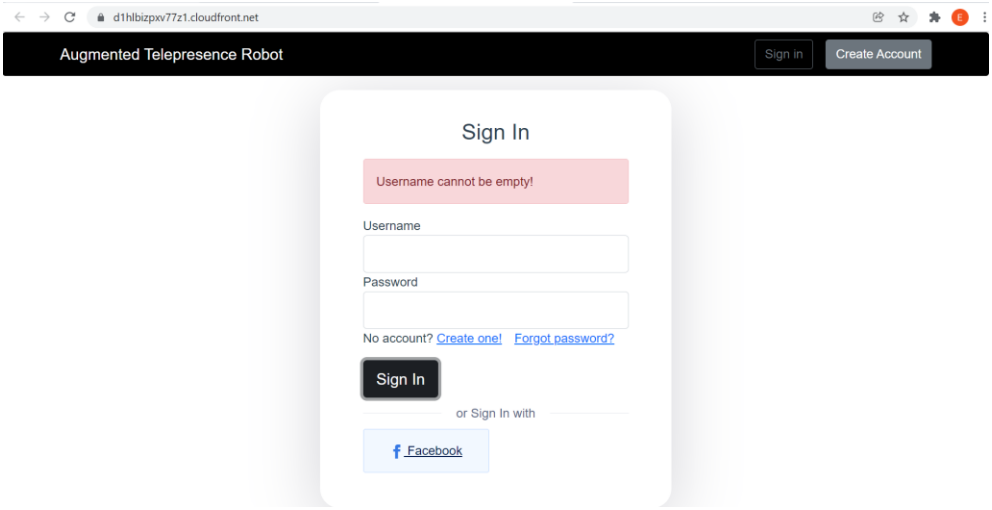

Figure 6-4: "Username cannot be empty!" error message

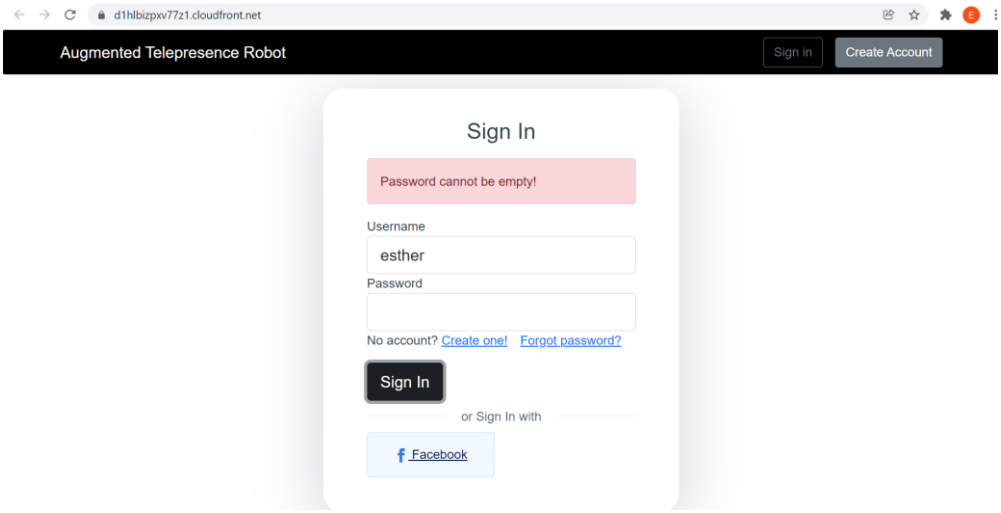

Figure 6-5: "Password cannot be empty!" error message

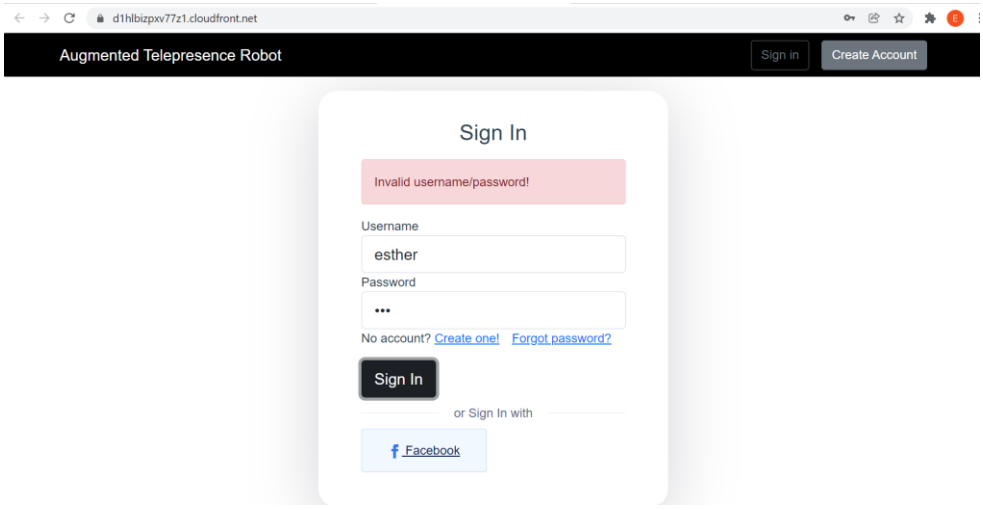

Figure 6-6: "Invalid username/password" error message

To ensure the interface is user-friendly, error messages will be displayed to allow the user to know if he/she's username or password has been inputted wrongly. Some examples of errors include, if the user did not input a username or password, "Username cannot be empty!" and "Password cannot be empty!" error messages will be displayed respectively. In addition, if the user enters an invalid username or password, "Invalid Username/Password" error message will be displayed.

6.4.2. SIGN-UP PAGE

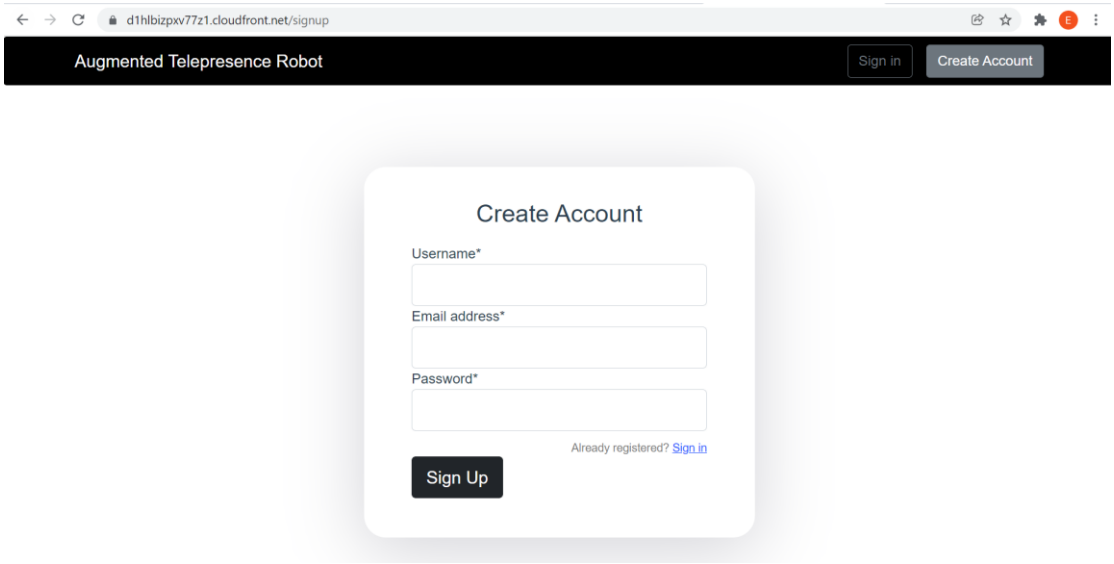

Figure 6-7:Sign-Up page

If a user has not created an account yet, it will create one in the Sign-up page where a valid username, email, and password must be entered.

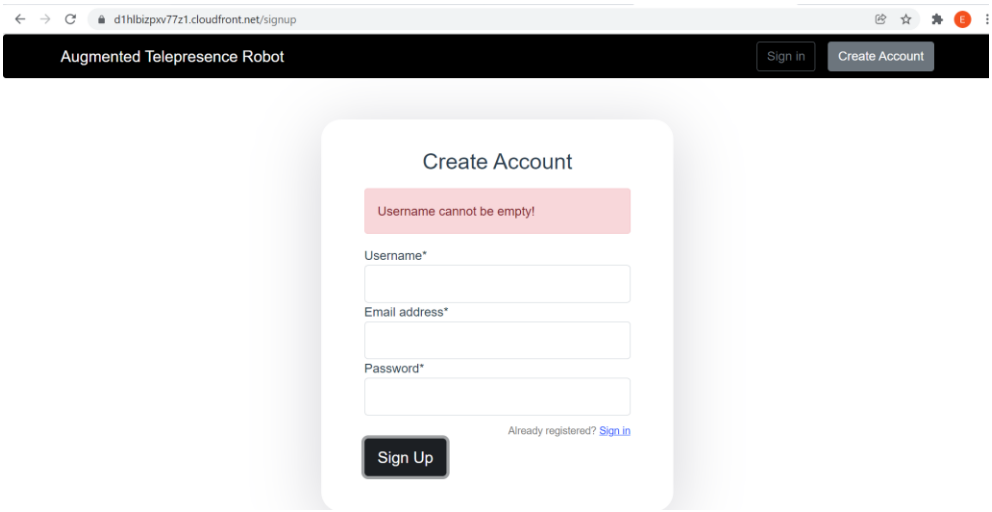

Figure 6-8: "Username cannot be empty!" error message

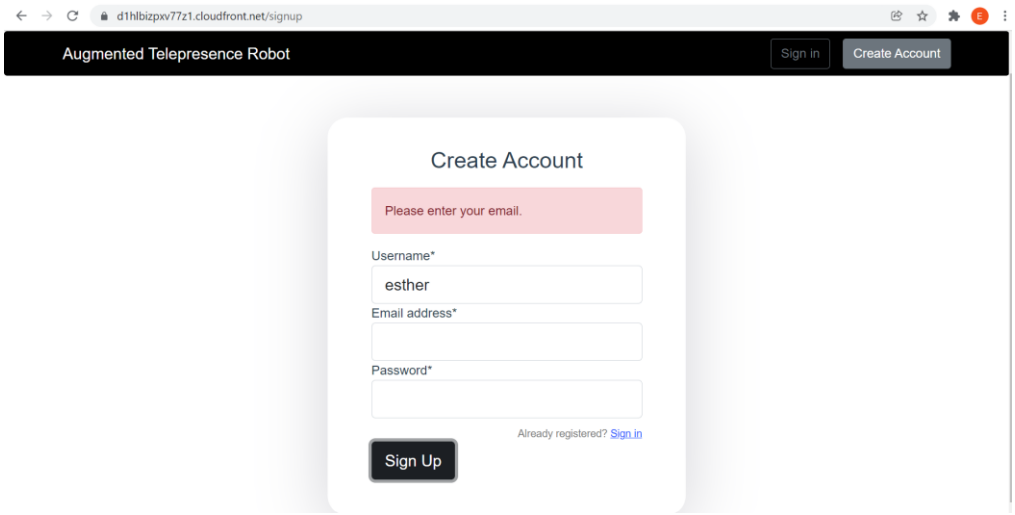

Figure 6-9: "Enter email" error message

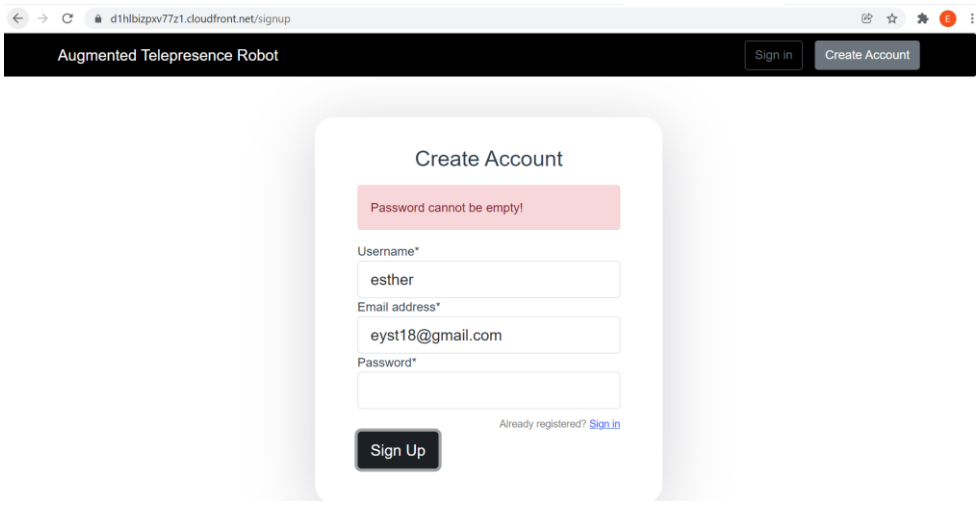

Figure 6-10: "Password cannot be empty!" error message

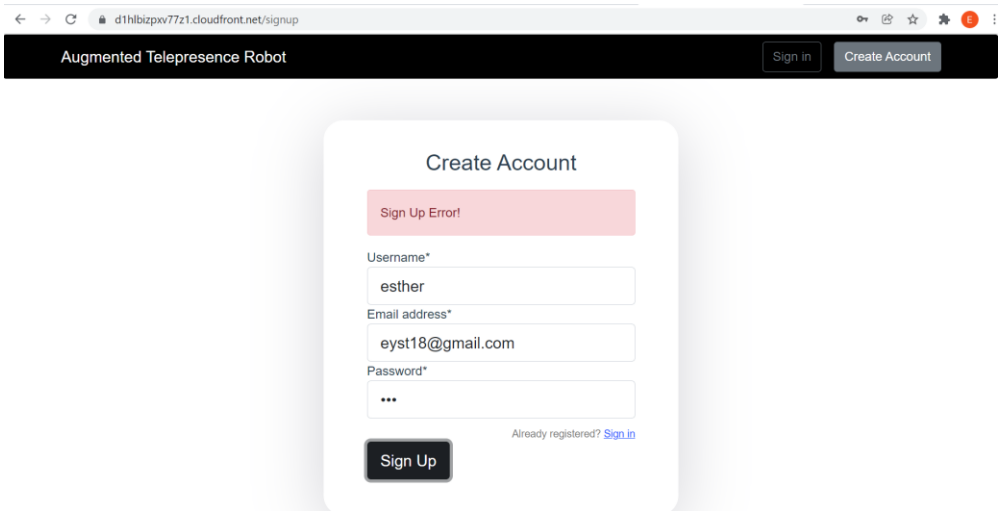

Figure 6-11: "Sign Up Error" error message

Like the Sign-In page, error messages will be displayed to allow the user to know if he/she's username or password has been inputted wrongly. Some examples of errors include, if the user did not input a username or password, "Username cannot be empty!" and "Password cannot be empty!" error messages will be displayed respectively. In addition, if the user enters an existing username or an invalid email, "Sign-Up Error!" error message will be displayed.

6.4.3. CONFIRM SIGN-UP PAGE

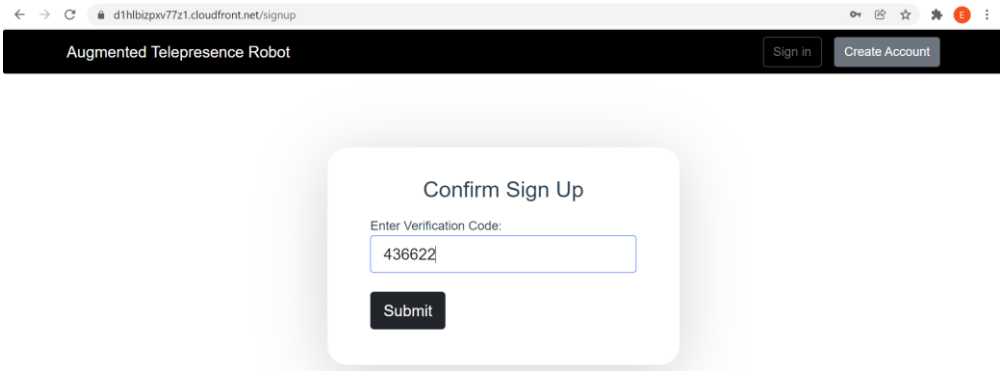

Figure 6-12: Confirm Sign-Up page

A verification code will be sent to the valid email and must be entered in the confirm Sign-up page to verify the user. Once the submit button is clicked, it will direct the user back to the sign-in page to access its account.

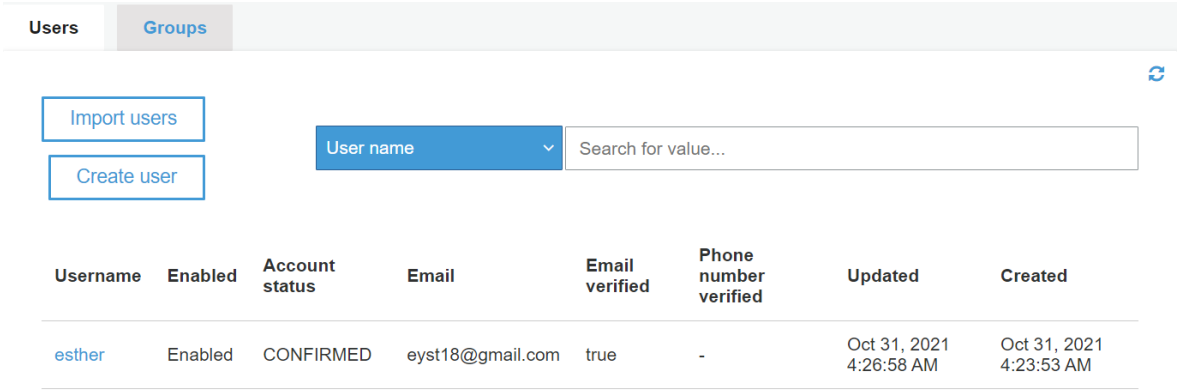

Figure 6-13: New user pool created in AWS Cognito

In Amazon Cognito, the user pool will be updated.

6.4.4. FORGOT PASSWORD PAGE

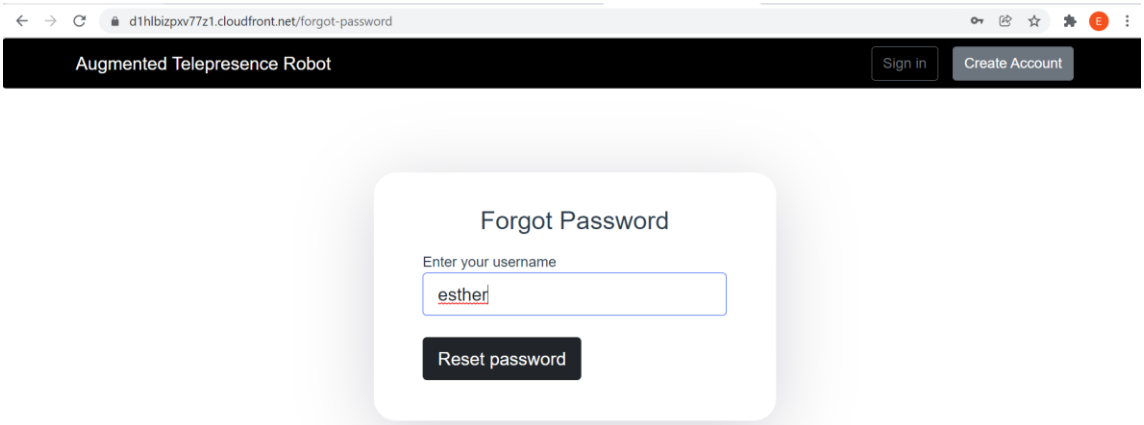

Figure 6-14: Forgot Password page

If a user forgets its password, it will click the “Forgot password” in the Sign-in page and it will direct it to the Forgot Password page where it prompts the user to enter its username.

6.4.5. CHANGE PASSWORD PAGE

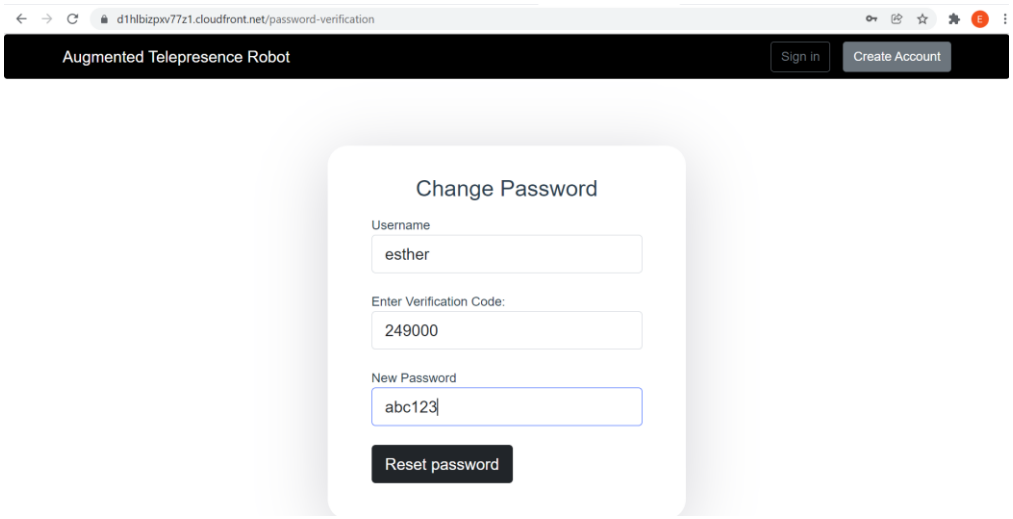

Figure 6-15: Change password page

The user will be directed to the Change Password page where it must enter its username, verification code sent to the user as well as its new password.

6.4.6. AFTER USER SIGN IN PAGE

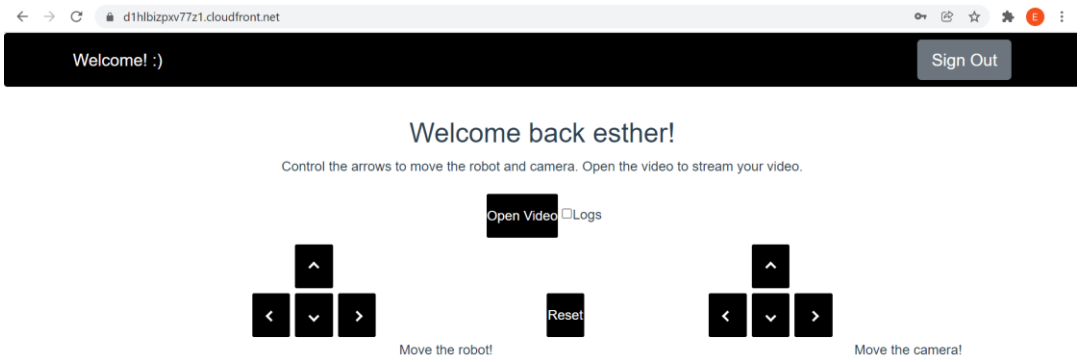

Figure 6-16: After user sign in page

After the user sign-in successfully, a personalized message will be shown, and the user will be able to display its video stream and control the movement of the robot and camera via the buttons.

6.5. VUE.JS CODE

6.5.1. MAIN.JS

This main.js file is the entry point of the Vue application.

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'
import Amplify, * as AmplifyModules from 'aws-amplify'
import { Auth } from 'aws-amplify'
import { AmplifyPlugin } from 'aws-amplify-vue'
import awsconfig from './aws-exports'
import * as config from './config.json'
import 'bootstrap/dist/css/bootstrap.min.css'

Amplify.configure({
    // OPTIONAL - if your API requires authentication
    Auth: {
        // REQUIRED - Amazon Cognito Identity Pool ID
        identityPoolId: awsconfig.aws_cognito_identity_pool_id,
        // REQUIRED - Amazon Cognito Region
        region: awsconfig.aws_project_region,
        // OPTIONAL - Amazon Cognito User Pool ID
        userPoolId: awsconfig.aws_user_pools_id,
        // OPTIONAL - Amazon Cognito Web Client ID (26-char alphanumeric string)
        userPoolWebClientId: awsconfig.aws_user_pools_web_client_id,
    },
```

Figure 6-17: Configure auth in Main.js

```
    API: {
        endpoints: [
            {
                name: "SendAction",
                endpoint: config.endpoint,
                custom_header: async () => {
                return {
                  Authorization: `Bearer ${(await Auth.currentSession()).getIdToken().getJwtToken()}`
                }
              }
            }
        ]
    }
})
```

Figure 6-18: Configure endpoints in Main.js

As shown on the picture, all the import files are at the top of the code which includes files for Vue itself, the app, the router, the store, as well as Amplify. The “Amplify.configure()” function configures Amazon Cognito into the web application for user authentication as well as the Amazon API Gateway. API Gateway creates REST APIs that enables real-time communication applications, therefore an endpoint had to be stated to allow this communication to occur and data to be transmitted.

```
Vue.use(AmplifyPlugin, AmplifyModules)

Vue.config.productionTip = false

new Vue({
  router,
  store,
  render: h => h(App),
}).$mount('#app')
```

Figure 6-19: Create Vue instance in Main.js

Next, a Vue instance is created which is required for all Vue application. It receives an option object that consists of information about the application, information includes the DOM element that supports the Vue instance as well as the data the instance will be using.

6.5.2. INDEX.HTML

The index.html file shown below is the standard index HTML file.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Figure 6-20: Index.html code

Focusing on , this div is where the Vue instance will be applied by the “#app” found in the main.js file, allow the homepage of the website to be shown.

6.5.3. APP.VUE

This is the first Vue component that was created which was imported and rendered in the main.js file shown earlier.

```
<template>
    <div id="app">
        <router-view />
    </div>
</template>

<style>
* {
    box-sizing: border-box;
  }
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  margin-top: 150px;
  color: #2c3e50;
}

</style>
```

Figure 6-21: App.vue code

This component allows routing to occur between the sign-in, sign-up, forgot password page which will be further discussed later. In almost all the components, <template> <script> <style> are used. <template> is a HTML-bind rendered DOM to the underlying Vue instances data, <script> contains all JavaScript codes consisting of the functions needed to run, and <style> is the CSS code used to apply a unique style to a particular HTML element.

6.5.4. SIGN-IN PAGE

```
<template>
  <div class="hello">
    <div v-if="!signedIn">
      <Nav />
      <div class="vertical-center">
        <div class="inner-block">
          <h3>Sign In</h3>
          <div v-if="error" class="alert alert-danger" role="alert">
            {{error}}
          </div>
          <div class="form-group">
            <label>Username</label>
            <input v-model="login" type="username" class="form-control form-control-lg" />
          </div>
          <div class="form-group">
            <label>Password</label>
            <input v-model="password" type="password" class="form-control form-control-lg" />
          </div>
          <p class="signup text-right col px-md-8">
            No account?
            <router-link :to="{name: 'signup'}">Create one!</router-link>
          </p>
          <button type="submit" class="btn btn-dark btn-lg btn-block" @click="signIn">Sign In</button>
          <p class="forgot-password text-right mt-2 mb-4">
              <router-link to="/forgot-password">Forgot password ?</router-link>
          </p>
        </div>
      </div>
    </div>
```

Figure 6-22: Before user sign in

The code shown above is before a user sign into the website. It also consists of router-links that brings the user to the sign-up page or forgot password page when they wish to.

```
    <div v-if="signedIn">
      <div id="leave">
        <nav class="navbar light bg-black rounded justify-content-between flex-nowrap flex-row fixed-top">
          <div class="container">
            <a class="navbar-brand float-left" target="_blank" style="color: #ffffff" href="#">
              Welcome! :)
            </a>
            <ul class="nav navbar-nav flex-row float-right">
              <li class="nav-item">
                <button type="button" class="btn btn-secondary" @click="signOut">Sign Out</button>
              </li>
            </ul>
          </div>
        </nav>
      </div>
      <h2>Welcome back, {{this.login}}!</h2>
      <p>Control the arrows to move the robot and camera. Open the video to stream your video.</p>
      <VideoViewer />
      <Interface />
    </div>
  </div>
</template>
```

Figure 6-23: After user sign in

Once the user is signed in, a personalized message will be displayed. <VideoViewer /> and <Interface /> are routed to display and allow the user to view their own video streams as well as to control the movement of the robot.

```
methods: {
  signIn(){
    Auth.signIn(this.login, this.password)
    .then(user =>{
          this.$store.state.signedIn = !!user;
          this.$store.state.user = user;
          if(user.challengeName === 'NEW_PASSWORD_REQUIRED') {
            const { requiredAttributes } = user.challengeParam;
            Auth.completeNewPassword(
              this.login,
              this.newPassword
            )
            .then(user => {
              console.log(user);
            })
            .catch(err => {
              console.log(err);
            });
          }
          else {
            //other situations
          }
      })
    .catch(err =>{
      console.log(err)
      this.error=err
      })
  },
```

Figure 6-24: Authentication in SignIn.vue

To allow authentication for user to sign in, “Auth.signIn()” method is created where the username and password are necessary for authentication to take place. In the then function, when the user clicks on the sign in button, the user information will be passed into the “this.login, this.password” function to check if the information is correct. If it is correct, the user object will be returned and sign in will be successful. However, if it is incorrect, a catch function will catch the error and display it in the console.

```
    <div v-if="error" class="alert alert-danger" role="alert" action="/something">
        {{error}}
    </div>
.catch(err =>{ console.log(err)
    if(this.login === '') {
      this.error = 'Username cannot be empty!'
    }
    else if(this.password === '') {
      this.error = 'Password cannot be empty!'
    }
    else {
      this.error = 'Invalid username/password!'
    }
})
```

Figure 6-25: catch() function for errors

The above code shows how the error messages are displayed. A element is created to declare the error variable and to display the red box, indicating an error. In the "Auth.SignIn()" method, a "catch ()" function is called for any errors occurred while signing in, and the if-else statements are used to check if the username or password is empty, and if an invalid username and password are inputted.

6.5.5. FACEBOOK SIGN-IN

To integrate and set up the Facebook Login into the Vue web application, "amplify update auth" command is used to configure Auth provider by choose "Default configuration with Social Provider (Federation)". Then, a Facebook App must be created in the Facebook Developers Dashboard to receive the Facebook App ID and Secret. This is required when initializing the Facebook JavaScript SDK. Also, Facebook must be added in the Identity Pools as an authentication provider, the serverless backend API will then generate an Id once a user sign in. (Serverless Stack, n.d.)

```
loadFacebookSDK() {
  window.fbAsyncInit = function() {
    window.FB.init({
      appId: "909059866705951",
      xfbml: true,
      version: "v3.2"
    });
    window.FB.AppEvents.logPageView();
  };

  (function(d, s, id) {
    var js,
      fjs = d.getElementsByTagName(s)[0];
    if (d.getElementById(id)) {
      return;
    }
    js = d.createElement(s);
    js.id = id;
    js.src = "https://connect.facebook.net/en_US/sdk.js";
    fjs.parentNode.insertBefore(js, fjs);
  })(document, "script", "facebook-jssdk");
},
```

Figure 6-26: loadFacebookSDK() method

First, "loadFacebookSDK()" method is created to initialize the Facebook JavaScript SDK by inserting the Facebook App ID. (Serverless Stack, n.d.)

```
waitForInit() {
  return new Promise(res => {
    const hasFbLoaded = () => {
      if (window.FB) {
        res();
      } else {
        setTimeout(hasFbLoaded, 300);
      }
    };
    hasFbLoaded();
  });
},
```

Figure 6-27: waitForInit() method

Second, "waitForInit()" method is created to wait for the Facebook JavaScript SDK to load. Once it is loaded, login with Facebook button will be enabled. (Serverless Stack, n.d.)

```
statusChangeCallback(response) {
  if (response.status === "connected") {
    this.handleResponse(response.authResponse);
  } else {
    this.handleError(response);
  }
},
checkLoginState() {
  window.FB.getLoginStatus(this.statusChangeCallback);
},
handleClick() {
  window.FB.login(this.checkLoginState, { scope: "public_profile,email" });
},
handleError(error) {
  alert(error);
},
```

Figure 6-28: Code to show response after user accepts permissions

When the user presses on the Facebookbutton, “handleClick ()” method will run to listen for the login status which will be changed in the “statusChangeCallback ()” method. While calling this method, “{scope: “public_profile_email”}” is configured to retrieve the user’s public profile and email address. (Serverless Stack, n.d.)

```
async handleResponse(data) {
  const { email, accessToken: token, expiresIn } = data;
  const expires_at = expiresIn * 1000 + new Date().getTime();
  const user = { email };

  this.isLoading = true;

  try {
    //const response = await Auth.federatedSignIn(
    await Auth.federatedSignIn("facebook", { token, expires_at }, user);
    this.isLoading = false;
    AmplifyEventBus.$emit("authState", "signedIn");

    //this.props.onLogin(response);
  } catch (e) {
    this.isLoading = false;
    this.handleError(e);
  }
},
```

Figure 6-29: Authentication for Facebook

Once the user has given the permissions to login, “Auth.federatedSignIn()” method will be called to receive the user information retrieved from Facebook, allowing the user to be signed in securely. (Serverless Stack, n.d.)

6.5.6. SIGN-UP PAGE

```
<template>
<div>
  <div v-if="!user">
    <Nav />
    <div class="vertical-center">
      <div class="inner-block">
        <h3>Create Account</h3>
        <div v-if="error" class="alert alert-danger" role="alert" action="/something">
            {{error}}
        </div>
        <div class="form-group">
          <label>Username*</label>
          <input v-model="login" type="username" class="form-control form-control-lg"/>
        </div>
        <div class="form-group">
          <label>Email address*</label>
          <input v-model="email" type="email" class="form-control form-control-lg" />
        </div>
        <div class="form-group">
          <label>Password*</label>
          <input v-model="password" type="password" class="form-control form-control-lg" />
        </div>
        <p class="forgot-password text-right">
          Already registered?
          <router-link :to="{name: 'signin'}">Sign in</router-link>
        </p>
        <button type="submit" class="btn btn-dark btn-lg btn-block" @click='submit'>Sign Up</button>
      </div>
    </div>
  </div>
```

Figure 6-30: Before user pool created

The code shown above is when a user pool is not created, the sign-up page will be displayed. It also consists of router-link that brings the user to the sign-in page if they already had created an account.

```
  <div v-if="user">
    <Nav />
    <div class="vertical-center">
      <div class="inner-block">
        <h3>Confirm Sign Up</h3>
          <div class="form-group">
            <label>Enter Verification Code:</label>
            <input v-model="code" type="text" class="form-control form-control-lg"><br>
          </div>
            <button type="confirm" class="btn btn-dark btn-lg btn-block" @click='confirm'>Submit</button>
      </div>
    </div>
  </div>
</div>
</template>
```

Figure 6-31: After user pool created

Once a user pool is created successfully, a verification code page will be displayed to ensure the email is under the correct user.

```
submit(){
  Auth.signUp({
    username: this.login,
    password: this.password,
    attributes: {
      email: this.email   // optional
    },
    validationData: [],  // optional
    })
    .then(data => this.user = data.user)
    .catch(error => {console.log(err)
    })
  }
}
```

Figure 6-32: Authentication in SignUp.vue (Verification code sent to user)

To allow authentication for user to sign up, "Auth.signUp()" method is created where the username and password are objects for authentication to take place. "this.login" is associated to username and "this.password" is associated to password, since email is also configured to receive the verification code to verify the user, therefore "this.email" will be associated to email under attributes. In the then function, user information is set in "this.user" to complete the sign up. If sign up is unsuccessful, the catch function will catch the error and display it in the console.

```
methods:{
  confirm(){
    // After retrieveing the confirmation code from the user
      Auth.confirmSignUp(this.login, this.code, {
          // Optional. Force user confirmation irrespective of existing alias. By default set to True.
          forceAliasCreation: true
      }).then(data => this.$router.push("/"))
        .catch(err => console.log(err));
  },
```

Figure 6-33: Authentication in SignUp.vue (Confirm Sign-Up)

Once the user inputs its details and sign up, a verification code will be sent to its email. "Auth.confirmSignUp()" method is created to prompt the user to enter its username and code for the sign-up procedure to be completed. If it is successful, a router push is enabled to return to the sign-in page. However, if it is unsuccessful, an error will be displayed in the console.

```
    <div v-if="error" class="alert alert-danger" role="alert" action="/something">
      {{error}}
    </div>
      .catch(err => {console.log(err)

      if(this.login === '') {
        this.error = 'Username cannot be empty!'
      }
      else if(this.email === '') {
        this.error = 'Please enter your email.'
      }
      else if(this.password === '') {
        this.error = 'Password cannot be empty!'
      }
      else {
        this.error = 'Sign Up Error!'
      }
    })
  }
}
```

Figure 6-34: catch() function for errors

The above code shows how the error messages are displayed. A element is created to declare the error variable and to display the red box, indicating an error. In the "Auth.SignUp()" method, a "catch()" function is called for any errors occurred while signing up, and the if-else statements are used to check if the username, email, or password is empty, and if an existing username and an invalid email is inputted.

6.5.7. FORGOT PASSWORD

Regarding forgot password, two components are created to allow user to change its password.

```
  methods: {
    forgotPassword() {
      Auth.forgotPassword(this.login)
        .then(data => this.$router.push('/password-verification'))
        .catch(err => console.log(err));
      }
  }
}
```

Figure 6-35: Authentication in ForgotPassword.vue

ForgotPassword.vue component is created where "Auth.forgotPassword()" method is used. It passes the username of the user into the method, if it is successful, a router push directs the user to another page that allows is to enter its verification code and its new password.

```
    methods: {
        forgotPasswordSubmit() {
        // Collect confirmation code and new password , then
        Auth.forgotPasswordSubmit(this.login,this.code, this.newPassword)
          .then(data => this.$router.push("/"))
          .catch(err => console.log(err));
        }
    }
}
```

Figure 6-36: Authentication in ForgotPasswordVerification.vue

ForgotPasswordVerification.vue component is created where "Auth.ForgotPasswordVerification()" method is used. The username, code and new password entered is passed into this method and router push the user back to the sign-in page. If any error occurs, it will be catch and displayed in the console.

6.5.8. INTERFACE

```
Auth.currentCredentials().then((info) => {
    console.log(info)
  });


async function postData(dir) {
  let apiName = 'SendAction';
  let path = '/publish';
  let myInit = {
    body: {
      action: dir
    }
  }
  return await API.post(apiName, path, myInit);
}
```

Figure 6-37: Interface.vue code

Interface.vue component is created to manage the buttons for the user to control the movement of the robot and the camera. "postData()" function is declared to post the desired direction of either the robot or the camera via the API Gateway to the MQTT topic and publishes the desired direction into the website for movement to occur. "API.post()" functions allows the apiName, path, and myInit variables to pass through this function and sends a data to the API server.

```
var AWS = require('aws-sdk');
var iot = new AWS.Iot();
var robotName = process.env.ROBOT_NAME;
var publishTopic = `${robotName}/action`

  publishMessage(params, (err, res) => {
    callback(null, {
        statusCode: err ? '400' : '200',
        body: err ? err.message : JSON.stringify(res),
        headers: {
            'Content-Type': 'application/json',
            "Access-Control-Allow-Origin": "*"
        }
    })
  })
}
```

Figure 6-38: Lambda function for robot movement

The above figure is coded into a Lambda function that stores the robot's name and the action under the variable "publishTopic". This variable is then used to determine and allow the lambda function to publish the desired MQTT topic with the message send via the API Gateway.

```
export default {
  name: 'Interface',
  methods: {
    forward: function () {
      postData('forward')
    },
    backwards: function () {
      postData('backwards')
    },
    moveleft: function () {
      postData('moveleft')
    },
    moveright: function () {
      postData('moveright')
    },
```

Figure 6-39: postData() method code

The above code is an example on how to "postData()" function is applied as a method for movement of the robot to occur. 'forward', 'backwards', 'moveleft', moveright' are names passed into the function to post the desired direction.

6.5.9. VIDEOVIEWER

VideoViewer.vue component is created to allow user to live stream its videos.

```
const kinesisVideoClient = new AWS.KinesisVideo({
    region,
    credentials
});
```

Figure 6-40: Create KVS Client

First, kinesisVideoClient (KVS Client) is created. This client uses libraries and SDK to securely connect the Kinesis Video Streams and other media data to the web application for users to view. Therefore, it receives data from the servo and manages them, as well as the stream lifecycle to the Kinesis Video Streams as the data flows.

```
const getSignalingChannelEndpointResponse = await kinesisVideoClient
    .getSignalingChannelEndpoint({
        ChannelARN: channelARN,
        SingleMasterChannelEndpointConfiguration: {
            Protocols: ['WSS', 'HTTPS'],
            Role: 'VIEWER'
        }
    })
    .promise()
```

Figure 6-41: Get Signaling Channel Endpoints

Secondly, Signaling Channel Endpoints are obtained by declaring "getSignalingChannelEndpointResponse" variable. These variable issues an endpoint for the signaling channel to send and receive messages. From the code above, the signaling channel is assigned a HTTPS protocol, where this API creates a secure websocket endpoint, and a WSS protocol, where this API creates an HTTPS endpoint, to connect to for data requests to be processed. Role determines the messaging permissions. As 'VIEWER' role is assigned, this API produces an endpoint where peer communication can only be made with a 'MASTER' by a client. (Amazon Web Services, 2021)

```
const kinesisVideoSignalingChannelsClient = new AWS.KinesisVideoSignalingChannels({
    region,
    credentials,
    endpoint: endpointsByProtocol.HTTPS
})
```

Figure 6-42: Create KVS Signaling Client

Thirdly, KVS Signaling Client is created. The HTTPS endpoint that was derived by the "getSignalingChannelResponse" variable is used with the KVS Client to produce ICE servers. ICE servers are also known an Interactive Connectivity Establishment Servers. (Amazon Web Services, 2021)

```
const getIceServerConfigResponse = await kinesisVideoSignalingChannelsClient
    .getIceServerConfig({
        ChannelARN: channelARN
    })
    .promise()

const iceServers = [
    { urls: `stun:stun.kinesisvideo.${region}.amazonaws.com:443` }
]

getIceServerConfigResponse.IceServerList.forEach(iceServer =>
    iceServers.push({
        urls: iceServer.Uris,
        username: iceServer.Username,
        credential: iceServer.Password
    })
)
```

Figure 6-43: Get ICE server configurations

Fourthly, ICE server configurations are obtained. The information consists of the URLs, username, and credentials which are used to configure the WebRTC connection. Furthermore, the code above collects both the STUN and TURN server configurations where "getIceServerConfig ()" API is used to get the TURN server. To allow peer-to-peer communication between the website and the robot to obtain the live video, they must exchange SDP which consists of a list of ICE candidates which are the IP and port pairs present to allow these exchanges to occur, connectivity checks are then checked to allow media to flow between these two applications. ICE candidates are built by making a chain of requests to a STUN server, this server returns the public IP address and port pair that originated the request and adds each pair to the list of ICE candidates, gathering all ICE

candidates which will then return an SDP, allowing communication to occur. (Amazon, 2021)

```
viewer.peerConnection = new RTCPeerConnection({ iceServers })
logger('Create peerConnection')
```

Figure 6-44: Create RTCPeerConnection

Fifthly, RTCPeerConnection is created as it is the primary interface for WebRTC communications in the web application. (Amazon Web Services, 2021)

```
viewer.signalingClient = new SignalingClient({
    channelARN,
    channelEndpoint: endpointsByProtocol.WSS,
    clientId,
    role: 'VIEWER',
    region,
    credentials
})
```

Figure 6-45: Create WebRTC Signaling Client

Sixthly, WebRTC Signaling Client is created. This client is used to send messages over the signaling channel where it contains information such as channelARN, 'VIEWER' role, region, and credentials etc. (Amazon Web Services, 2021)

```
// When an ICE candidate is received from the master, add it to the peer connection.
viewer.signalingClient.on('iceCandidate', candidate => {
    logger('Candidate received from Master')
    viewer.peerConnection.addIceCandidate(candidate)
})

viewer.signalingClient.on('close', () => {
    logger('Signaling Client Closed')
    stopViewer()
})

viewer.signalingClient.on('error', error => {
    logger(`ERROR: ${JSON.stringify(error)}`)
})
```

```
viewer.signalingClient.on('open', async () => {
    logger('Creating SDP offer')
    // Create an SDP offer and send it to the master
    const offer = await viewer.peerConnection.createOffer({
        offerToReceiveAudio: true,
        offerToReceiveVideo: true,
    })
    await viewer.peerConnection.setLocalDescription(offer)
    viewer.signalingClient.sendSdpOffer(viewer.peerConnection.localDescription)
})
// When the SDP answer is received back from the master, add it to the peer connection.
viewer.signalingClient.on('sdpAnswer', async answer => {
    await viewer.peerConnection.setRemoteDescription(answer)
    logger('Received sdpAnswer')
})

viewer.peerConnection.addEventListener('icecandidate', ({ candidate }) => {
    if (candidate) {
      logger('Received iceCandidate')
      viewer.signalingClient.sendIceCandidate(candidate)
    }
})

// As remote tracks are received, add them to the remote view
viewer.remoteView = document.querySelector('#myVideoEl')
viewer.remoteView.play()
logger('Start playing incoming video')

viewer.peerConnection.addEventListener('track', event => {
    if (viewer.remoteView.srcObject) {
        return
    }
    viewer.remoteView.srcObject = event.streams[0]
})
```

Figure 6-46: Signaling Client Event Listeners (Above 3 images)

Lastly, Signaling Client Event Listeners are added which waits for an event to occur before carrying out a certain action.

6.6. CHALLENGES

The first challenge was understanding the JavaScript code. With a lack of knowledge of this coding language, many confusions were made. Therefore, countless research and understanding must be made to truly understand the flow of codes. Also, while writing the codes and troubleshooting errors, it made apprehension of the codes significantly better. Secondly, as the project was referenced by a blog post, it uses the ready-made authenticator component which consists of the sign-in, sign-up page etc. By attempting to replace these components with self-customized front-end codes, it was difficult and time consuming as minor mistakes can cause the web application to not work. Therefore, individual parts of the code must be transferred slowly and diligently instead of transferring the whole code directly. In addition of troubleshooting and finding the cause of the errors, the authentication of the web application was created successfully. Thirdly, implementing Facebook Login was a problem at one point of time as well. One learning point about programming is the need to be extremely precise and cautious to prevent any errors from occurring, errors such as "JSSDK Unknown Host Domain" occurred as "Login with the JavaScript SDK" in the Developers for Facebook Dashboard settings was not enabled, it thus led to another error "Can't load URL: The domain of this URL isn't included in the app's domain" as incorrect URL was inputted for the domains that are allowed for the JavaScript SDK. The URL should be the hosted URL provided after "amplify publish" command was called rather than the localhost URL. Lastly, the idea of implementing scalability in the project was the most challenging. To allow writing and reading of data to and from a table, DynamoDB was utilized which is one of the AWS Services. Countless of research and knowledge on how the data information is stored and retrieved must be made. Furthermore, the importance of understanding how to integrate the code into the Vue.js application is vital as well. Many problems arose when trying to query the DynamoDB table. First, AWS credentials were not

returned when calling the function “getItem()”, resulting in the inability for the querying of data information from the DynamoDB table. As multiple ways of coding were integrated, various errors popped out such as “CognitoSocialIdentityProvider” is not a provider. Cognito SDK and its webpack were also installed into the application, however none could solve the problem. Secondly, in the VideoViewer.vue component, “config” variable was declared by importing the config.JSON file, and since the endpoints and channelARN for the first robot was stored, the endpoints and channelARN will always be the same despite whichever username is inputted. Thus, another approach was made by duplicating the Interface.vue and VideoViewer.vue components for each of the robot, as well as using the <v-if> and <v-else> to check which user is inputted, which will eventually display the endpoint and channelARN for the unique user.

7. REFLECTION

Through this project, we have learnt the importance of being independent learners. With little to no knowledge on AWS services, JavaScript programming and other necessary skills required for this project, there is a need to better equip ourselves for it. As we explore the various tools and websites available, we gradually adopt useful research skills. These skills have taught us how to look to the internet for solutions, when faced with challenges. To plan, create and improve our project, we need to have sufficient knowledge on the resources we will be using. Using platforms such as LinkedIn, we developed a better understanding of how and what are the ideal resources to use to achieve the objectives of this project. By referencing from projects done by other developers, we can piece together ideas and implement them to our project. We soon realise that our accomplishments would only be limited by our willingness to learn.

As we work towards the objectives of this project, we are bound to meet challenges and obstacles. Therefore, having resilience through this process is important. Without this never say die attitude, we will never be able to overcome and learn from such challenges. Although the learning curve is steep and time-consuming, we learnt to look at minor achievements as motivation and constantly encourage each other as we go. The biggest motivation was the thought that this project might be implemented to help improve the quality of life for patients in the healthcare industry one day.

Facing such challenges has allowed us to develop useful troubleshooting and research skills. Troubleshooting skills such as integrating “console.log ()” commands to check if the code is successful or not, or using a multi-meter to test the components, we have learnt the various ways to look out for when working with software and hardware. Though the internet

can provide us with countless solutions, but finding the right solution is a skill itself. By looking through forums and blog posts, we were able to understand the types of key words to look out for or the search.

Working with each other has taught us to stay open minded and respect each other opinions and ideas. These values are needed to create an environment where effective communication can thrive. As we may have different views and perspectives, listening to each other was vital to ensure that both voices could be heard. We learnt that we had to compromise with each other to find the best and most ideal way to improve our project. When solving problems we face, two brains will always be better than one. Teamwork also involves taking responsibility of our tasks by completing it on time. Setting datelines will allow us to plan our time wisely and progress together as a team.

7.1. ENHANCEMENT

The first enhancement is to implement facial analysis to the video stream using AWS Rekognition. AWS Rekognition is an AWS service that allows developers to add image and video analysis to their applications. For facial search, AWS Rekognition can search images, stored videos, and streaming videos to check if any faces match those stored in a container also referred to as a face collection (Amazon Web Service , n.d.). To enable facial search, many developers implemented AWS Rekognition stream processor onto AWS KVS. However, while trying this approach, it was discovered that this AWS Rekognition object was not supported in the ap-southeast-1 (Singapore) region. Therefore, another approach was to capture a picture and store it in a local directory on the Raspberry Pi. AWS Rekognition will work directly with the Raspberry Pi and the images captured in the directory. Unfortunately, although this worked, was realised that there was no way of streaming the live video footage onto AWS KVS and capturing the picture at the same time.

Therefore, more research was done on third-party software like OpenCV and Tensorflow. However, it was discovered that OpenCV does not support with WebRTC (zakaziko, 2020). This is because OpenCV requires a HTTP Live Streaming (HLS) URL to retrieve the video frames from the KVS video stream that will be used for the facial recognition, but the project uses WebRTC which is different from HLS. If given more time, implementation of other platforms like Tensorflow could be used on the front end to obtain the video frames and capture a process them using AWS Recognition (developer.mozilla.org, 2021).

```
257 PVOID receiveGstreamerAudioVideo(PVOID args)
258 {
259     STATUS retStatus = STATUS_SUCCESS;
260     GstElement *pipeline = NULL, *appsrcAudio = NULL;
261     GstBus* bus;
262     GstMessage* msg;
263     GError* error = NULL;
264     PSampleStreamingSession pSampleStreamingSession = (PSampleStreamingSession) args;
265     gchar *videoDescription = "", *audioDescription = "", *audioVideoDescription;
266 
267     if (pSampleStreamingSession == NULL) {
268         printf("[KVS GStreamer Master] receiveGstreamerAudioVideo(): operation returned status code: 0x%08x \n", STATUS_NULL_ARG);
269         goto CleanUp;
270     }
271 
272     // TODO: Wire video up with gstreamer pipeline
273 
274     switch (pSampleStreamingSession->pAudioRtcRtpTransceiver->receiver.track.codec) {
275         case RTC_CODEC_OPUS:
276             audioDescription = "appsrc name=appsrc-audio ! opusparse ! decodebin ! autoaudiosink";
277             break;
278 
279         case RTC_CODEC_MULAW:
280         case RTC_CODEC_ALAW:
281             audioDescription = "appsrc name=appsrc-audio ! rawaudioparse ! decodebin ! autoaudiosink";
282             break;
283         default:
284             break;
285     }
286 
287     audioVideoDescription = g_strjoin(" ", audioDescription, videoDescription, NULL);
288 
289     pipeline = gst_parse_launch(audioVideoDescription, &error);
290 
291     appsrcAudio = gst_bin_get_by_name(GST_BIN(pipeline), "appsrc-audio");
292     if (appsrcAudio == NULL) {
293         printf("[KVS GStreamer Master] gst_bin_get_by_name(): cant find appsrc, operation returned status code: 0x%08x \n", STATUS_INTERNAL_ERROR);
294         goto CleanUp;
295     }
296 
297     transceiverOnFrame(pSampleStreamingSession->pAudioRtcRtpTransceiver, (UINT64) appsrcAudio, onGstAudioFrameReady);
```

Figure 7-1: receiveGstreamerAudioVideo() function

(Github, 2021)

Currently, audio can only be transmitted from the robot to the website. Therefore, one further enhancement would be to stream audio from the client accessing the website to the Raspberry Pi, allowing audio to be played on the speaker attached to it. This can be done using the "receiveGstreamerAudioVideo()" function found in the "kvsWebRTCClientMasterGstreamer" sample. This function will allow the master

(Raspberry Pi) to receive audio from the viewer (website). On the website end, once the peer-to-peer connection is open, signalling client event listeners must be added. This will connect the default audio input of the viewer, and create an SDP offer to send the audio track to the master (raspberry pi). Once connected, a two-way audio communication to occur between the person with the robot and the person on the website. This will value add to the telepresence aspect of our project.

Further improve to the scalability of this project can be made as well. As mentioned previously, a less ideal approach was made for scalability. As items were first manually created inside the DynamoDB table, the application was coded to only allow reading and getting data information from the table. The application was also hardcoded which is not user-friendly. Therefore, thinking ahead on constructing an application that is user-friendly in the real-world context, the user's information should be inserted and stored in the DynamoDB table automatically.

7.2. CONCLUSION

In conclusion, this project has been fruitful and fulfilling where we were able to encounter many first-hand experiences. Through hands-on experience, we were able to develop communication, problem-solving and research skills and build teamwork. As we continue to pursue our interest in biomedical engineering, we hope to apply the skills and knowledge we have acquired from this project.

8. BIBLIOGRAPHY

adafruit. (n.d.). *Adafruit STEMMA Speaker - Plug and Play Audio Amplifier - JST PH 2mm*. Retrieved from adafruit: https://www.adafruit.com/product/3885

Adam, H. (02 December, 2020). *Augmented Reality*. Retrieved from Investopedia: https://www.investopedia.com/terms/a/augmented-reality.asp

Amazon. (2021). *Kinesis Video Streams Producer Libraries*. Retrieved from Amazon: https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/producer-sdk.html

Amazon. (n.d.). *Freenove 4WD Smart Car Kit for Raspberry Pi 4 B 3 B+ B A+, Face Tracking, Line Tracking, Light Tracing, Obstacle Avoidance, Colorful Light, Ultrasonic Camera Servo Wireless RC*. Retrieved from Amazon: https://www.amazon.sg/Freenove-Raspberry-Tracking-Avoidance-Ultrasonic/dp/B07YD2LT9D

Amazon Web Service . (n.d.). *What is Amazon Rekognition?* Retrieved from Amazon Web Service : https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html

Amazon Web Services. (10 July, 2014). *Introducing Amazon Cognito*. Retrieved from Amazon Web Services: https://aws.amazon.com/about-aws/whats-new/2014/07/10/introducing-amazon-cognito/

Amazon Web Services. (2021). *GetSignalingChannelEndpoint*. Retrieved from Amazon: https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/API_GetSignalingChannelEndpoint.html

Amazon Web Services. (2021). *Kinesis Video Streams with WebRTC*. Retrieved from Amazon: https://docs.aws.amazon.com/kinesisvideostreams-webrtc-dg/latest/devguide/kvswebrtc-how-it-works.html#how-webrtc-components-interwork

Amazon Web Services. (n.d.). *Amazon API Gateway pricing*. Retrieved from Amazon Web Services: https://aws.amazon.com/api-gateway/pricing/

Amazon Web Services. (n.d.). *Amazon Kinesis Video Streams pricing*. Retrieved from Amazon Web Services: https://aws.amazon.com/kinesis/video-streams/pricing/

Amazon Web Services. (n.d.). *AWS CloudFormation FAQs*. Retrieved from Amazon: https://aws.amazon.com/cloudformation/faqs/

Amazon Web Services. (n.d.). *AWS Free Tier*. Retrieved from Amazon Web Services : https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=categories%23compute&trk=ps_a134p000006gExDAAU&trkCampaign=acq_paid_search_brand&sc_

Amazon Web Services. (n.d.). *What is Amazon CloudFront?* Retrieved from Amazon Web Services: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html

Amazon Web Services. (n.d.). *What Is Amazon DynamoDB?* Retrieved from Amazon Web Services: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

Azam, S. (2021). *What is Vue.js, and Why is it Cool?* Retrieved from linuxhint: https://linuxhint.com/about_vue_js/

Chen, Denzel. (20 August, 2021). *Tutorial.pdf*. Retrieved from Github: https://github.com/Freenove/Freenove_4WD_Smart_Car_Kit_for_Raspberry_Pi/blob/master/Tutorial.pdf

Christie, A. A. (25 June, 2021). *Choose the right AWS video service for your use case*. Retrieved from Amazon: https://aws.amazon.com/blogs/iot/choose-the-right-aws-video-service-for-your-use-case/

cloudflare. (n.d.). *Why use serverless computing? | Pros and cons of serverless*. Retrieved from cloudflare: https://www.cloudflare.com/learning/serverless/why-use-serverless/

developer.mozilla.org. (15 September, 2021). *Taking still photos with WebRTC*. Retrieved from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Taking_still_photos

Element14. (n.d.). *RPI4-MODBP-8GB*. Retrieved from Element14: https://sg.element14.com/raspberry-pi/rpi4-modbp-8gb/raspberry-pi-4-model-b-cortex/dp/3369503

Epson. (n.d.). *Moverio BT-40S Smart Glasses with Intelligent Touch Controller*. Retrieved from Epson: https://epson.com/c/Moverio-BT-40S-Smart-Glasses-with-Intelligent-Touch-Controller/p/V11H969120

Gillis, A. S. (April, 2020). *Amazon Web Services (AWS)*. Retrieved from SearchAWS: https://searchaws.techtarget.com/definition/Amazon-Web-Services

Github. (02 November, 2021). *amazon-kinesis-video-streams-webrtc-sdk-c*. Retrieved from Github: https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-c

github. (n.d.). *kvsWebRTCClientMasterGstreamerSample.c*. Retrieved from github: https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-c/blob/master/samples/kvsWebRTCClientMasterGstreamerSample.c

Matador, B. (2 September, 2020). *What is AWS Amplify?* Retrieved from bluematador: https://www.bluematador.com/blog/what-is-aws-amplify

Milazzo, L. (16 August, 2021). *A production expert's guide to frame rates and FPS*. Retrieved from vimeo: https://vimeo.com/blog/post/fps-for-live-streaming/#:~:text=What%20is%20the%20best%20FPS,a%20significant%20amount%20of%20bandwidth.

Newegg. (n.d.). *Raspberry pi Camera Module Board REV 1.3 5MP Webcam Video 1080p 720p Fast For raspberry pi 3 starter kit*. Retrieved from Newegg: https://www.newegg.com/global/sg-en/p/1EF-007B-00003

Parlette, C. (18 January, 2018). *Why Serverless Computing Will Be Bigger Than Containers*. Retrieved from ParkMyCloud: https://www.parkmycloud.com/blog/serverless-computing/

seeedstudio. (n.d.). *ReSpeaker Mic Array v2.0*. Retrieved from seeedstudio: https://www.seeedstudio.com/ReSpeaker-Mic-Array-v2-0.html

Serverless Stack. (n.d.). *Facebook Login with Cognito using AWS Amplify*. Retrieved from serverless-stack: https://serverless-stack.com/chapters/facebook-login-with-cognito-using-aws-amplify.html

SneakyHacker. (22 December, 2020). *Fast Video Doorbell / Intercom on Raspberry Pi*. Retrieved from hackster.io: https://www.hackster.io/sneaky/fast-video-doorbell-intercom-on-raspberry-pi-63b063

TrustRadius. (n.d.). *AWS IoT Core Pricing*. Retrieved from TrustRadius: https://www.trustradius.com/products/aws-iot-core/pricing

Tulane University. (n.d.). *What's the Difference Between AR and VR*. Retrieved from Tulane University: https://sopa.tulane.edu/blog/whats-difference-between-ar-and-

vr#:~:text=The%20distinctions%20between%20VR%20and,while%20VR%20is%20completely%20virtual&text=VR%20requires%20a%20headset%20device,only%20enhances%20a%20fictional%20reality

zakaziko. (14 August, 2020). *how to stream the video capture from WebRTC in VLC or PYTHON OPENCV*. Retrieved from OpenCV answers : https://answers.opencv.org/question/233658/how-to-stream-the-video-capture-from-webrtc-in-vlc-or-python-opencv/

Zara, M. (25 March, 2020). *Building a Raspberry Pi telepresence robot using serverless: Part 1*. Retrieved from Amazon: https://aws.amazon.com/blogs/compute/building-a-raspberry-pi-telepresence-robot-using-serverless-part-1/

Zara, M. (13 April, 2020). *Building a Raspberry Pi telepresence robot using serverless: Part 2*. Retrieved from Amazon : https://aws.amazon.com/blogs/compute/building-a-raspberry-pi-telepresence-robot-using-serverless-part-2/