Alexa Enabled IoT Device Simulation Using C# And AWS Lambda

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Vice Chancellor, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area of the Paper: Computer Science
Type of the Paper: Experimental Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed In: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.8329375
Google Scholar Citation: IJCSBE

How to Cite this Paper:
Chakraborty, S., & Aithal, P. S. (2023). Alexa Enabled IoT Device Simulation Using C# And AWS Lambda. *International Journal of Case Studies in Business, IT, and Education (IJCSBE)*, *7*(3), 359-368. DOI: https://doi.org/10.5281/zenodo.8329375

International Journal of Case Studies in Business, IT and Education (IJCSBE)
A Refereed International Journal of Srinivas University, India.

Crossref DOI: https://doi.org/10.47992/IJCSBE.2581.6942.0305

Paper Submission: 06/08/2023
Paper Publication: 09/09/2023

Alexa Enabled IoT Device Simulation Using C# And AWS Lambda

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Vice-Chancellor, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: *Nowadays, device operation using voice commands is becoming popular. Several voice command services are available. Among them, Alexa from Amazon is the most popular. Almost Every day, vendors are integrating Alexa into their new products worldwide. We need physical devices for the device operation or understanding of the process flow of Alexa. Sometimes, physical devices are costly, or availability is poor. Here, we demonstrated how to create a simulated Alexa-enabled IoT device. We used several Amazon cloud services to execute the process flow. Alexa's skill is developed inside the Alexa developer console. To process the command, we use the AWS Lambda function. AWS IoT cloud service is used to trigger IoT devices over MQTT. For simulated devices, we are using a C# MQTT client. The researcher trying to simulate an Alexa-enabled device into their project can get some reference information from this work.*

Design/Methodology/Approach: *We create a graphical user interface (GUI) to interact with the user or display the device's status. The GUI is connected with the C# AWS IoT Device Shadow client. We created IoT Things in the AWS cloud. Under the IoT Shadow, we created a Device shadow. The Alexa service triggers the lambda function. The Lambda function updates the Shadow register, which resides inside the cloud. The c# shadow client receives a notification when the device Shadow updates and updates the GUI element that represents the equipment. We can use any Alexa device to send voice commands, like the Alexa mobile app, Echo Dot, or Alexa PC app.*

Findings/Result: *Through this research work, we created and tested virtual devices that can be experiments or research work using Alexa voice commands. It has been tested for a long time. It performed well, and no issue was found. Using more code-level protection it can be robust for practical implementation.*

Originality/Value: *We found several fragmented documents over the web to implement Alexa-enabled virtual devices. After lots of study, we did practical and included the procedure in this research work. From voice input to load trigger, there are lots of steps involved. The complete guidance is not available easily. So through this research work, if someone follows, they can easily create a voice-operated device. This research work may add value to the researcher experimenting with the Alexa command-activated device.*

Paper Type: *Experimental-based Research.*

Keywords: Alexa device in virtual form, How to create virtual IoT device, simulated device on Alexa.

1. INTRODUCTION :

Alexa-enabled devices are standard nowadays. Voice command-based device operation is also becoming popular. It has several advantages. It is completely isolated from the user. In the industry, it is safe for the operator from electrical hazards. We can efficiently operate our electrical equipment using voice commands in our domestic. To operate voice commands, we need a voice-captured device. Amazon provides several channels to receive the audio. Some physical devices are available, like the

Echo Dot. Amazon provides a PC application that acts like Alexa to receive voice input, send the voice token to the Alexa cloud server, back to our PC, and play response through the speaker.
Here, we will simulate the Alexa devices like real hardware. For this purpose, we need some software framework. We use most of the framework as real hardware does. Instead of hardware like an Equipment module, we will use software clients. Several steps need to be followed to trigger the virtual equipment. First, we need to create Alexa skills in the AWS developer console. Then, we create the AWS Lambda function using Visual Studio and deploy it to the AWS cloud. The Lambda function processes the Alexa command string. Moreover, it is responsible for responding to the Alexa voice terminal. Then, we created the Aws IoT profile. In the profile, we create a Shadow register. It is like a variable to exchange the device status. The lambda function updates the shadow register so that The MQTT client can get the latest commands. In the C#, we are creating a Graphical user interface. It represents real-world hardware. When the application starts, one MQTT client also starts. The main thread fetches the changes in the shadow register.
Once receive the command it is pursued. Based on the command, it triggers the virtual load or GUI element. To input the voice command, we need an echo dot. Alternatively, other options are available. We can use our mobile phone as Alexa voice input. Install Alexa apps. Log in to the account. Open the skill we created and say the command to trigger the load. Other options are installing PC Alexa apps from the Microsoft store and opening the apps, login and opening Skill and Talk. The other option is inside the AWS cloud, with a test interface. Open the test interface and talk or send the command by typing.

2. RELATED WORKS :

Kim et al. (2020) demonstrate how Alexa's voice-activated technology enhances telehealth services [1]. Liu, Chen, and Li (2019) focus on issues like data security, sharing user information, etc. [2]. Wang, Liu, and Lu (2021) focus on user interactions with Alexa regarding its usability and effectiveness [3]. Zhou and Zhang (2020) discuss how Alexa's conversation is used for language learning applications [4]. Zhou, Zhang, and Chen (2018) examine Alexa's voice in understanding and processing human speech on its language comprehension capabilities [5]. Sharma and Ali (2021) demonstrate virtual assistants' strengths and weaknesses [6]. Su and Liang (2020) analyze how Alexa's integration affects consumer behavior, purchase decisions, and interaction patterns [7]. Dubey and Kumar (2019) focus on how Alexa's voice command with music streaming platforms influences music consumption patterns [8]. Zhang and Li (2021) scrutinize the security and privacy challenges posed by Amazon Alexa, offering insights into potential vulnerabilities and safeguards [9]. Liu and Chen (2018) explore Alexa's potential in shaping the smart home ecosystem, including its integration with home automation devices and its impact on daily living [10]. These studies shed light on Alexa's capabilities extending to language learning, music consumption, and smart homes. However, challenges related to privacy, security, and user experience also emerge as significant areas for consideration in the ongoing development and adoption of Amazon Alexa.

3. OBJECTIVES :

The research aims to provide reference information to the new researcher to build a test environment for Alexa-enabled devices. Using this procedure without hardware cost, we can simulate the Alexa command and trigger the electrical equipment using voice command. There are several ways to experiment. Here, we described a practical approach to execute the experiment. Following the step-by-step procedure, the researcher can experiment easily. Once the framework is established, it can customize the experiment according to the project's needs.
Figure 1 depicts the diagram of the project. The flow initiates from the left side. The input is received from any Alexa client device that can receive the voice input. The device converts the voice into a voice token. That token transmits to Alexa voice service in the AWS cloud. The AWS Alexa voice service converts voice to text. That text triggers the AWS Lambda function. The Lambda function updates the device Shadow. When the device shadow is updated, it triggers the C# IoT client, and the IoT Shadow client receives the updated data. Parsing the data, the C# UI thread triggers the virtual load. Now, we will see the procedure to execute the described flow. To build the Alexa virtual devices, we must follow the sequence below to understand and create the device quickly.

4. APPROACH AND METHODOLOGY :

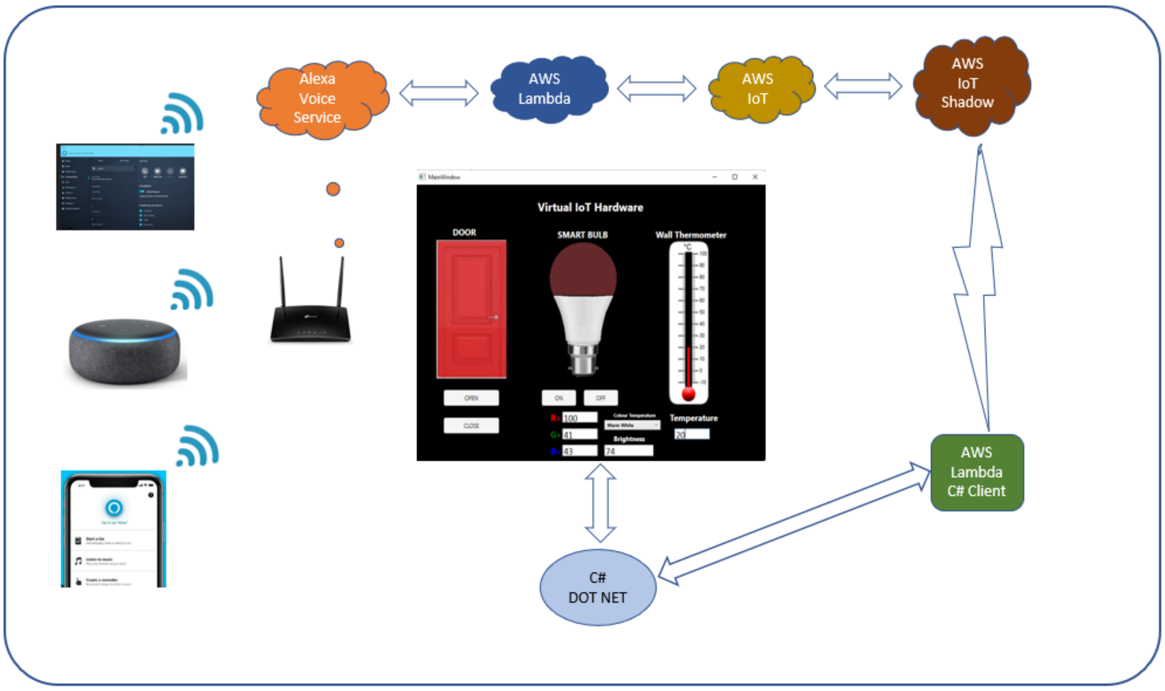

Fig. 1: The block diagram of the project

1) Create GUI Virtual Device
2) Test Virtual Devices.
3) Create a Things using the AWS IoT console
4) Create policies
5) Create a Shadow
6) Attach policies with a certificate
7) Test using MQTT test client
8) Convert certificate pem to pfx- done
9) Communicate with the IoT cloud from the C# application.
10) Create Alexa Skill
11) Test Alexa skill
12) Integrate command with Virtual Load element
13) Overall test

5. EXPERIMENT :

5.1 Create GUI in C#:

(1) Create a Repository in GitHub. Create a folder called “Alexa Enabled IoT Device Simulation” on the desktop. Clone the repository inside the desktop using a source tree [1].
(2) Create a visual studio C#, Desktop, Widows **Form** application.
(3) Build and run. A blank form will open.
(4) Open source tree. Ignore /debug and /bin folder.
(5) On the Windows form, create an electrical load graphically and make such a way that when entering the text, the load should trigger [2].
(6) Our GUI design and test are complete.

5.2 Create AWS IoT things:

1. Create an IoT account in AWS. Open AWS Console > Search IoT Core and Open.
2. Select a zone like US East(N. Virginia), i.e., us-east-1.

3. Open IoT Dashboard>from the left side. Click Manage
4. All Devices>Things>click on the "**Create things**" button>select a radio button, "**Create single thing**">click "Next."
5. Thing name: "**My_Virtual_Device"**
6. Device Shadow: "No shadow."
7. Click **Next**.
8. Under the Device certificate page, keep selecting "Auto-generate a new certificate(recommended)" >click **Next**> click "**Create things**."
9. Download four certificates Under the "Download certificates and keys" page.
    a. Click on the download button of the Device certificate. It looks like "*5dc3d7a7bc4...te.pem.crt*"
    b. Download the public key file. It looks like "*5dc3d7a7bc41ffb5349d09b...1a007a2-public.pem.key*".
    c. Download the Private key file. It looks like "*5dc3d7a7bc41ffb5349d09b...a007a2-private.pem.key*".
    d. Download Root CA certificates. It looks like *"RSA 2048 bit key: Amazon Root CA 1".*
10. Click the "**Done**" button.

5.3 Create Policies:

1. From the Left side, under security, click the "Policies" menu.
2. Click on the "Create policy" button> Name: "**My_virtual_Device_Policy**."
3. Select *, policy resource type *, and click the "Create" button in the Policy action.

5.4 Attach Policies with Certificate:

1) Manage>All devices>Things> click on things name "My_Virtual_Device**."**
2) Under the Certificates tab, click on Certificate ID. It looks like "*5dc3d7a7bc41ffb5349d09b3744fcb6....0379152b7ceef0731a007a2.*"
3) On the right side, Click on "Attach policies."
4) From the combo box, select policy name "My_virtual_Device_Policy" > click on "**Attach policies**."

5.5 Create Shadow:

1) Under Things>click on Things Name "My_Virtual_Device" > select the tab "Device Shadows."
2) Click on "Create Shadow." Keep selecting "Named Shadow."
3) Under "Device Shadow name," type "**My_Virtual_Device_Shadow**" and click **Create**.

5.6 Configure MQTT C# Client:

1) Under settings, copy the Device data endpoint. It looks like "*a3go8….i56ev-ats.iot.ap-northeast-1.amazonaws.com*"
2) Keep the downloaded certificate in the \bin\Debug\net6.0 folder. "AmazonRootCA1.pem"
3) Open https://www.sslshopper.com/ssl-converter.html
4) In the Certificate File to convert field, select "*5dc3d7a7bc41ffb5349……8b67e5854d0379152b7ceef0731a007a2-certificate.pem*" (from the downloaded file)
5) Select Private key File. It looks like the "*5dc3d7a7bc41ffb5349d09b……67e5854d0379152b7ceef0731a007a2-private.pem.key*" (from the downloaded file)
6) The type of current certificate selects "Standard PEM."
7) In the "Type to Convert To" field, select "PFX/PKCS#12."
8) In the PFX Password field, type a password like MyDevice123.

9) Click "Convert Certificate." The certificate will download automatically. Click on the file name that was just generated. Change the name to "**device_certificate.cert**". It is placed inside the bin/Debug/net6.0 folder.

5.7 Enable GUI MQTT Enable:

1) Open the GUI project. Add "AWS_MQTT_Client.cs" to the project from the download folder.
2) Add the packages below from the Nuget package manager to run the MQTT module.
    a. M2Mqtt by Paolo Patierno
    b. M2MqttClientDotnetcore by M2MqttClientDotnetCore1.0.1
    c. Plt.M2Mqtt by Prolucid Technologies
    d. Newtonsoft.Json by James Newton-King
3) The below modification needs inside the AWS_MQTT_Client module:
4) Change the iotEndpoint variable. It is available from the AWS IoT console. Open the AWS IoT console. From the left side, click on settings. Copy Endpoint. It looks like "a3go…l5i56ev-ats.iot.ap-northeast-1.amazonaws.com" and replaces the module variable.
5) Next, we need to change the "topic" variable. Go to Things> click on things name "My_Virtual_Device"> click on "Device Shadows"> click on shadow name "My_Virtual_Device_Shadow" > click "MQTT topics" > copy /get/accepted topic. It looks like "$aws/things/My_Virtual_Device/shadow/name/My_Virtual_Device_Shadow/get/accepted." Replace the text.
6) Change the "password" when converting a certificate from pem to pfx.
7) Now build the project. It should build okay if something is going wrong and needs to be debugged.
8) We need some lines in our primary or GUI threads. First, create an MQTT object.
9) AWS_MQTT_Client client=new AWS_MQTT_Client();
10) Add a timer and add the code. When the MQTT module receives data, time will get the data.

```
private void timer1_Tick(object sender, EventArgs e)
 {
    if(client.received)
    {
            string s = client.Get_Data("command");
            txt_receive_msg.Text=s;
            Trigger_Load(s);
            client.received=false;
    }
}
```

11) Now we can test our module is receiving the data from the AWS cloud—open the AWS console in the browser.
12) From the left side, click on "MQTT test client." Select the "Publish to a Topic" tab. Inside the topic name, add the topic. It looks like: *"$aws/things/My_Virtual_Device/shadow/name/My_Virtual_Device_Shadow/update/accepted"*
13) Type the context below into the Message payload box and click publish. We will notice that our MQTT module has received the context:

```
{
  "message": "Hello from AWS IoT console"
}
```

14) After receiving the message, we must parse and pass it to the trigger load function. According to the command, it will trigger the specific load.

5.8 Create an Interaction model (Lambda function):

1. Open Visual Studio.

2. Create New project> AWS Lambda project>project name> location> select Empty Function>Finish.
3. From Nuget packge manager> install "**alexa.net**" and "**Amazon.Lambda.Serialization**". Packages.
4. Add /Write Code.
5. Build.
6. Add credentials.
7. Select a region from the dropdown list. Here, we are selecting "US East (N. Virginia)."
8. From Solution Explorer, click on "Publish to AWS Lambda."
9. Add Function Name: "My_Virtual_Device,"
10. Add a Description like "This is the Alexa-Enabled Virtual Device." Click the Next button.
11. Select IAM Role: "*New role based on AWS managed policy: AWS Lambda Full access*."
12. Now, we keep it simple. Skip Memory selection, Timeout> click on the "Upload" button.

5.9 Endpoint Addition:

1. Open the Alexa developer console. Click on skill. Click on the endpoint. Copy "Your skill ID."
2. In another browser tab, open the AWS Lambda console and check that the region is ok. Here we are in "Asia Pacific (**Tokyo) ap-northeast-1**".
3. Click the "+ Add trigger" button. Select source: "Alexa Skills Kit." Paste the copied skills inside the "Skill ID" textbox. It looks like "*amzn1.ask.skill.7c8dbcdd-feaf-4ee1-93c4-dd….ee8d7*". Click the "**Add**" button.
4. Click the Copy button of Lambda function ARN. It looks like "*arn:aws:lambda:ap-northeast-1:143….01433:function: My_Virtual_Device*"
5. Paste the Lambda ARN inside the Default Region textbox. At the top, click "Save endpoints."
6. Click the test button.

5.11 Update IoT Endpoint to the Lambda Function:

1. Open in another browser AWS IoT Console. Go to All devices> Things> click on "My_Virtual_Device"> click on "Device Shadows" tab> click on "My_Virtual_Device_Shadow"> press the copy button of Device shadow URL. It looks like "*https://a3g....i56ev-ats.iot.ap-northeast-1.amazonaws.com/things/My_Virtual_Device/shadow?name=My_Virtual_Device_Shadow*"
2. Replace the device URL text in the Lambda function inside the quotation.
3. Copy and paste Things name "*My_Virtual_Device*" and shadows name "*My_Virtual_Device_Shadow*."

5.12 Add permission IoT access to the Lambda function:

1. Open our Lambda funcrtiion
2. From the left pan, click "Permission."
3. Click on "**Roll name**" "lambda_exec_My_Virtual_Device-0". It will open the IAM console.
4. In the middle left, under "Add permissions," click on "Attach policies." In the textbox, type "IoT"> "AWSIoTFullAccess"> at the bottom, click on "Add permission."

5.13 Create Alexa Skill:

5. Log in Alexa developer portal: https://developer.amazon.com/alexa/console/ask
6. Click on the "**Create Skill**" button.
7. Enter the Skill name "**My Virtual Device**."
8. Set default language or keep "**English (US)**"> press the "Next" button.
9. Under "1. Choose a type of experience", select "**Other**."
10. Under "2. Choose a model" Keep Custom Model "Selected."

11. Under "3. Hosting services"> keep "Alexa-hosted (Node.js)"
12. Under "Hosting region," keep "**US East (N. Virginia)**" selected.> Click on "Next."
13. Under "Templates," select "**Start from scratch**" selected.> Click on "Next."

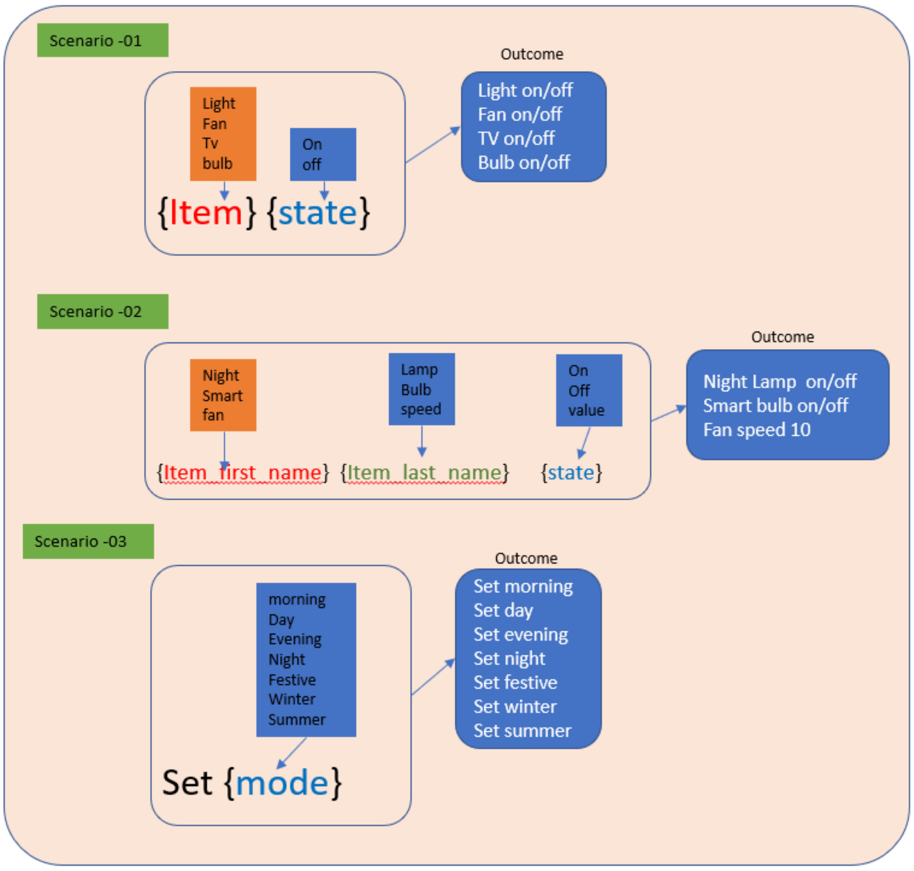

Fig. 2: Voice Token template

14. The summary will be displayed. If all information is ok, Click the "**Create Skill**" button.
15. Creating skills may take a couple of seconds to minutes. The completion message will be displayed after the successful build.
16. Skill Invocation Name: "**My virtual device**."
17. Click the "**Save Model**" button.
18. Expand the interaction model. Expand "**Intents**." Click the "**+ Add Intent**" button. Name: "**DeviceOperation**" and Click on "Create custom intent."
19. In figure 2 depicts our voice command template. Under the Sample utterance, type "**{item} {state}**" and Click on the "+" sign to add it. Right-click on the item and state, and click on the "add" button to add both as new slots.
20. Inside the sample utterances textbox, type:
21. {item_first_name} {item_last_name} {state}.
22. {item_first_name} {item_last_name} {value}.
23. Set {mode}.
24. After adding lines right, click on the name inside the curly braces and add intent, as depicted in Figure 3.
25. Intent Slots are visible at the bottom, just created. Set all SLOT Types as "AMAZON.FirstName."
26. Click on Save the model and build the model. A message box will appear when the build is complete.
27. At the top, there is a test button; click on it.
28. We will see at the top that "test is disabled for the skill." Select "Development."

29. Now we can ask or type the text. Type "open my virtual device."

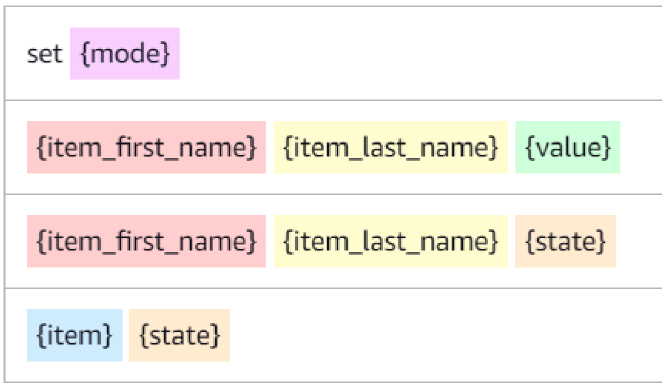
set {mode}
{item_first_name} {item_last_name} {value}
{item_first_name} {item_last_name} {state}
{item} {state}

Fig. 3: Utterence template

6. RECOMMENDATIONS :

- GUI repository: https://github.com/sudipchakraborty/Alexa-Enabled-IoT-Device-Simulation-Using-C-Sharp-And-AWS-Lambda.git
- Lambda function for Alexa: https://github.com/sudipchakraborty/Lambda_For_Alexa_Using_C_Sharp.git
- The paper on IoT device creation Inside the AWS: https://doi.org/10.5281/zenodo.7726980 [11]
- The article to create a physical IoT Device: https://doi.org/10.5281/zenodo.7779097 [12]
- The article Create Multiple IoT Device Controller: https://doi.org/10.5281/zenodo.7857660 [13]
- The paper to make Alexa Skill in the AWS cloud server: https://doi.org/10.5281/zenodo.7940237 [14]
- The article to create a Lambda function inside the AWS server: https://doi.org/10.5281/zenodo.7995727 [15]
- Experience researchers can add more functionality.
- We skip exception handling due to the complexity of the Code. The researcher must add it before going for production.

7. CONCLUSION :

Through this research work, we observed how to create and enable virtual devices that act like actual devices. It can help our research work by testing and debugging. We can use this procedure or framework as debug tools to test our custom hardware development without purchasing real-world hardware. We also learned how to create Alexa skills in the developer console and how to create an AWS lambda function. We created an IoT profile to find the data from the AWS IoT server. The complete framework applies to any project where we need IoT enabled.

REFERENCES :

[1] Kim, H., Kim, Y., & Kim, D. (2020). The potential of Amazon Alexa for telehealth services. *Telemedicine and e-Health, 26*(7), 446-449. https://doi.org/10.1089/tmj.2019.0160.

[2] Liu, J., Chen, X., & Li, D. (2019). Privacy protection in virtual assistant devices: A review of Amazon Alexa. *Future Generation Computer Systems, 98*(1), 602-612. https://doi.org/10.1016/j.future.2019.05.009.

[3] Wang, S., Liu, Y., & Lu, J. (2021). The user experience of Amazon Alexa: A review of the current state. *Human-Computer Interaction, 36*(2), 127-144. https://doi.org/10.1080/07370024.2020.1851137.

[4] Zhou, Y., & Zhang, D. (2020). Investigating the potential of Amazon Alexa for language learning. *Journal of Educational Technology Development and Exchange*, *3*(1), 1–17. https://doi.org/10.11648/j.jetde.20200301.11.

[5] Zhou, Y., Zhang, D., & Chen, W. (2018). A study of Amazon Alexa's speech recognition and natural language processing capabilities. *Journal of Ambient Intelligence and Humanized Computing*, *9*(5), 1323–1332. https://doi.org/10.1007/s12652-017-0501-9.

[6] Sharma, A., & Ali, A. (2021). A comparative analysis of Amazon Alexa, Google Assistant, and Apple Siri. *Journal of Computer Science and Technology*, *26*(1), 1–12. https://doi.org/10.1007/s11390-021-2363-x.

[7] Su, L., & Liang, Y. (2020). The adoption of Amazon Alexa and its impact on consumer behavior. *Journal of Business and Economics Research*, *18*(7), 381-393. https://doi.org/10.19030/jber.v18i7.13107.

[8] Dubey, A., & Kumar, V. (2019). A review of Amazon Alexa's impact on the music industry. *Journal of Digital Music and Sound*, *2*(3), 95-100. https://doi.org/10.1007/s42879-019-00034-2.

[9] Zhang, L., & Li, Y. (2021). The security and privacy issues of virtual assistant devices: A review of Amazon Alexa. *Journal of Computer Science and Security*, *15*(4), 198-210. https://doi.org/10.1007/s11416-021-0066-1.

[10] Liu, J., & Chen, X. (2018). Amazon Alexa's role in the future of smart homes. *Journal of Ambient Intelligence and Smart Environments*, *10*(3), 243-252. https://doi.org/10.3233/AIS-170562

[11] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create an IoT Inside the AWS Cloud. *International Journal of Case Studies in Business, IT, and Education (IJCSBE)*, *7*(1), 211-219. Google Scholar↗

[12] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create a Physical IoT Device Using AWS and ESP Module. *International Journal of Management, Technology and Social Sciences (IJMTS)*, *8*(1), 224-233. Google Scholar↗

[13] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create Multiple IoT Device Controller Using AWS, ESP32 And C. *International Journal of Applied Engineering and Management Letters (IJAEML)*, *7*(2), 27-34. Google Scholar↗

[14] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create an Alexa Skill for Our IoT Device Inside the AWS Cloud. *International Journal of Case Studies in Business, IT and Education (IJCSBE)*, *7*(2), 214-225. Google Scholar↗

[15] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create A Lambda Function For Our IoT Device In The AWS Cloud Using C. *International Journal of Management, Technology and Social Sciences (IJMTS)*, *8*(2), 145-155. Google Scholar↗

****