CSCI 5410

Serverless Data Processing

Assignment 3

Name : Sagarkumar Pankajbhai Vaghasia

CSID : vaghasia

Banner ID : #B00878629

Gitlab Link :

https://git.cs.dal.ca/vaghasia/csci5410-f23-b00878629/-/tree/A3

PART-A Event driven serverless application using AWS Lex, DynamoDB and Lambda function.

Custom chatbot using AWS Lex to book office hours to meet a professor.

- Firstly, I logged in AWS account through AWS academy student login. After logging in, I selected the dashboard option from the left-hand side panel where a course is available under the name of “ALLFv1-17043”. After selection of that course AWS Academy Learner Lab – Foundation Services page is opened from which I selected “Modules” section which showed me three options. Amongst those options, I selected “Learner Lab – Foundational Services”. The screenshots from logging to the process of selecting modules are shown in Figure-1 to Figure-3.

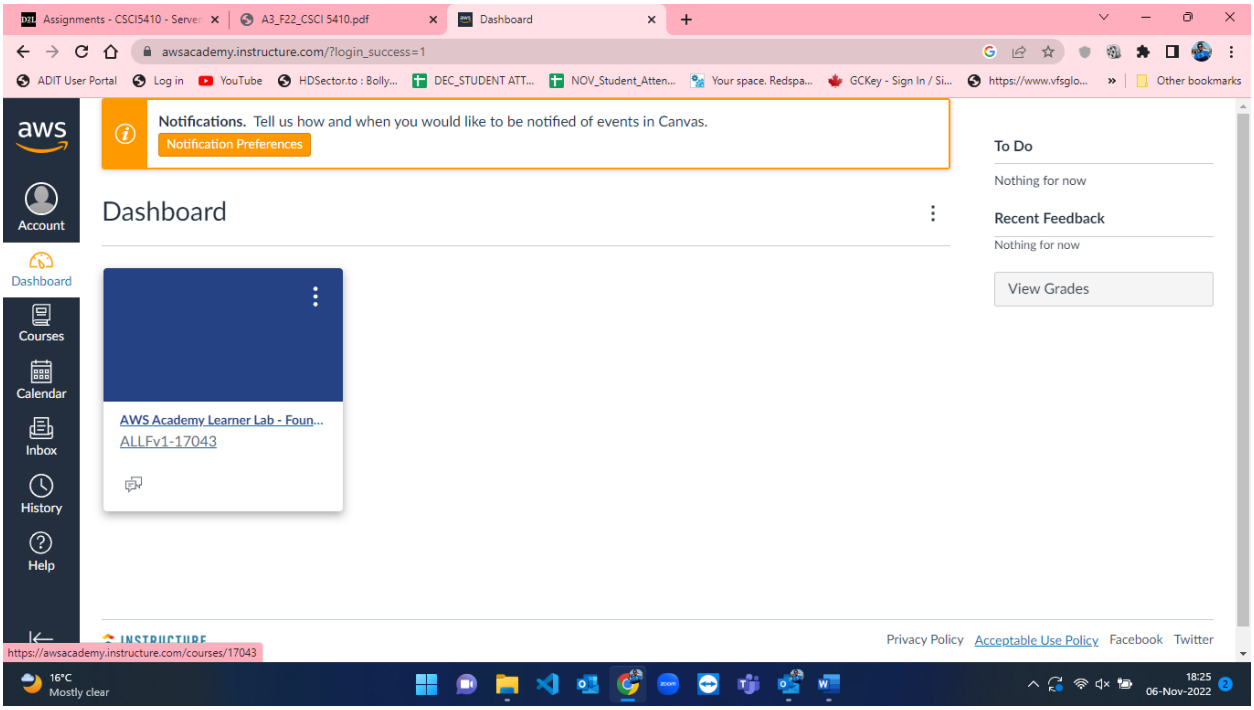

Figure 1: AWS Dashboard

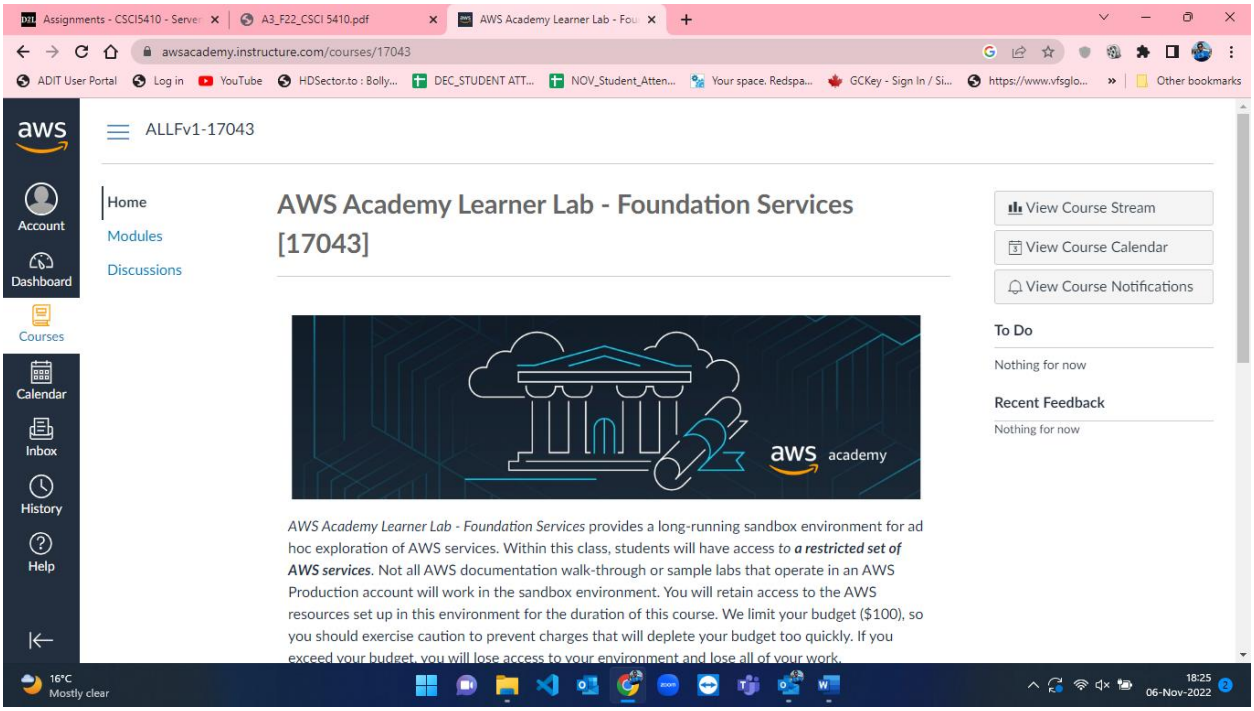

Figure 2: Homepage after course selection from dashboard in AWS.

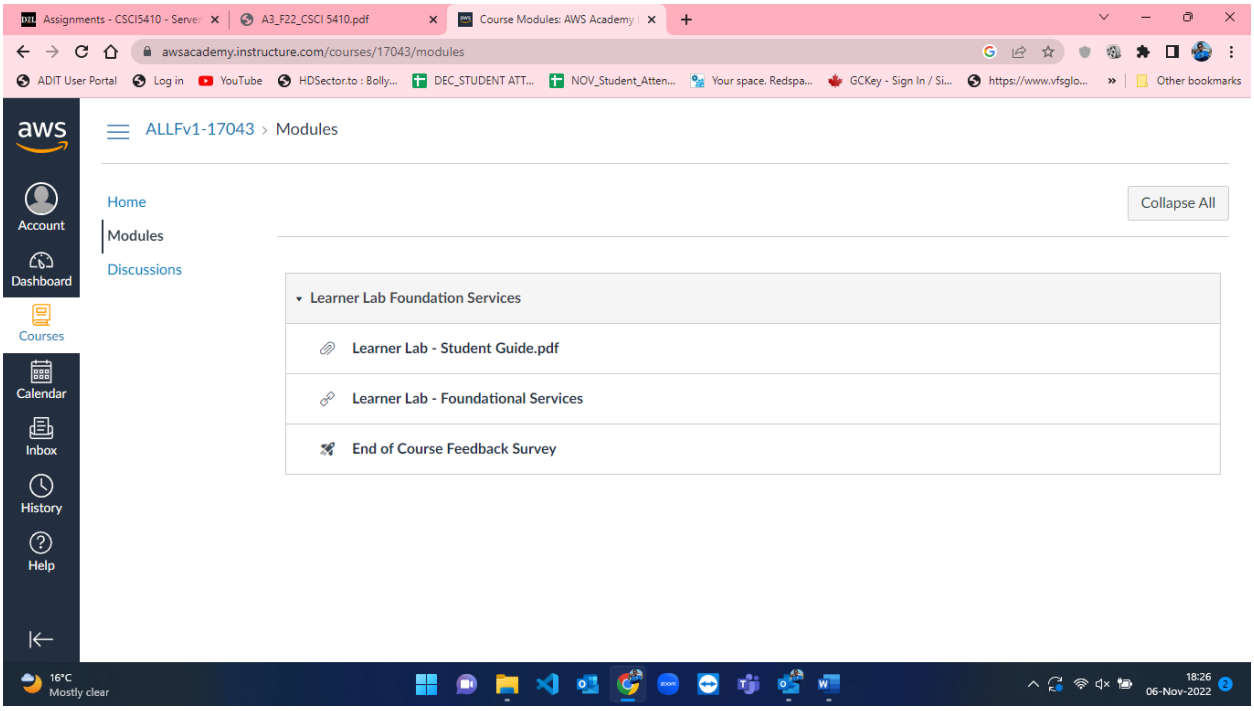

Figure 3: Modules page in AWS.

- After selecting “Learner Lab – Foundational Services”, the console page for the selected module is opened where the lab is currently not working. This can be seen in the below attached figure-4 where there is mark besides AWS text. To start working on AWS and its services we must start a lab. So, I clicked on “Start Lab” button to initiate lab. This can be seen in figure-5 where the lab is started, and the red mark turns to green besides AWS text.

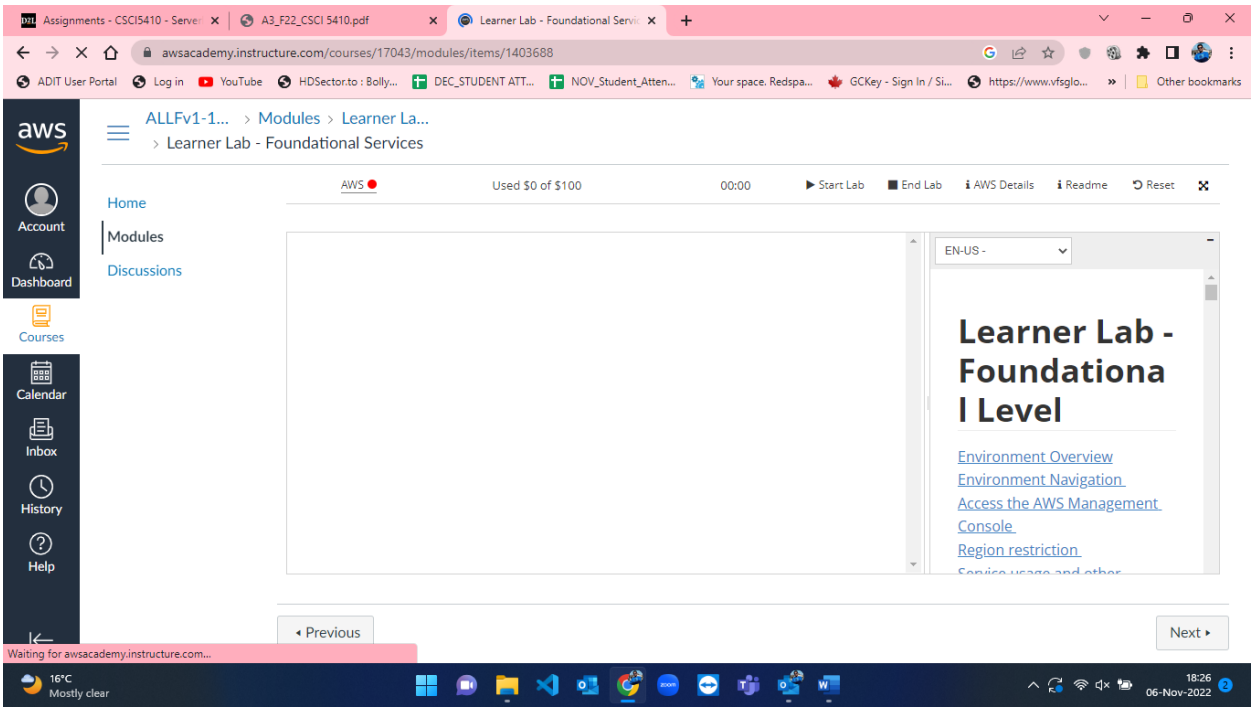

Figure 4: Lab not running in Modules page in AWS.

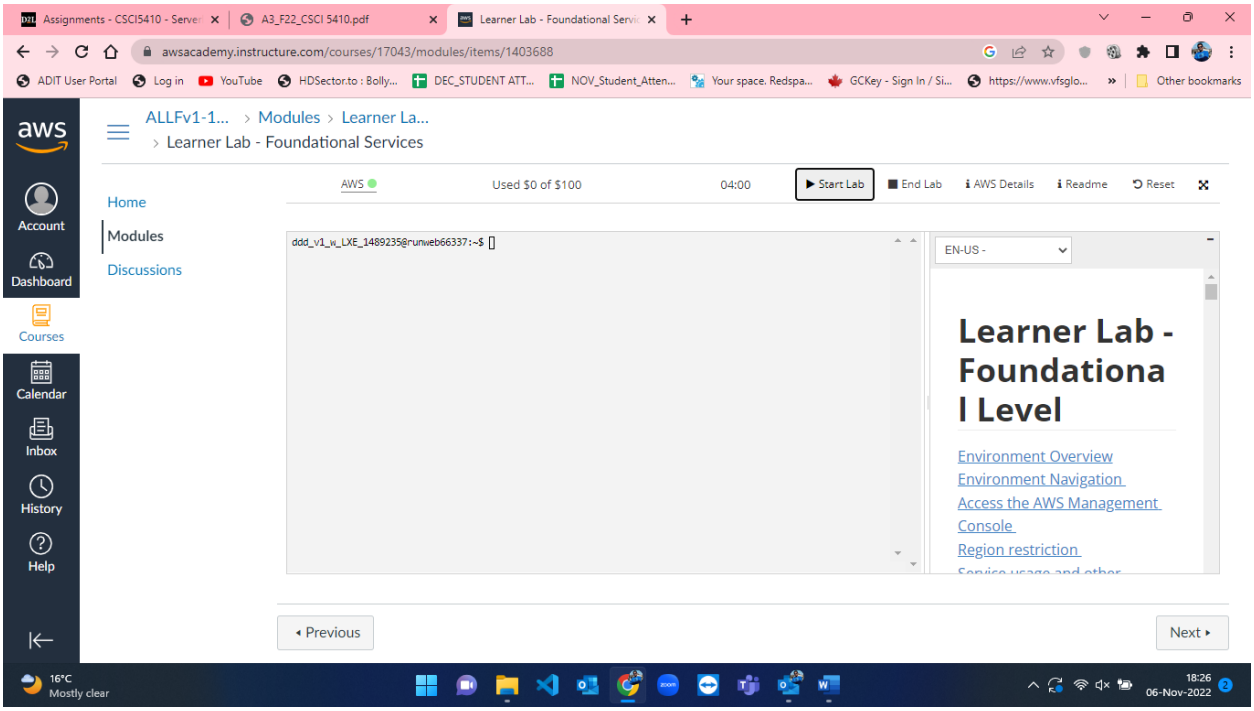

Figure 5: Lab started under Modules page in AWS.

- As soon as the lab started, I clicked on AWS text besides the green mark which took me the AWS console home (Figure-6). From that now we can access the services of AWS. To create a chatbot, I searched for AWS Lex from the search bar in console home(Figure-7).

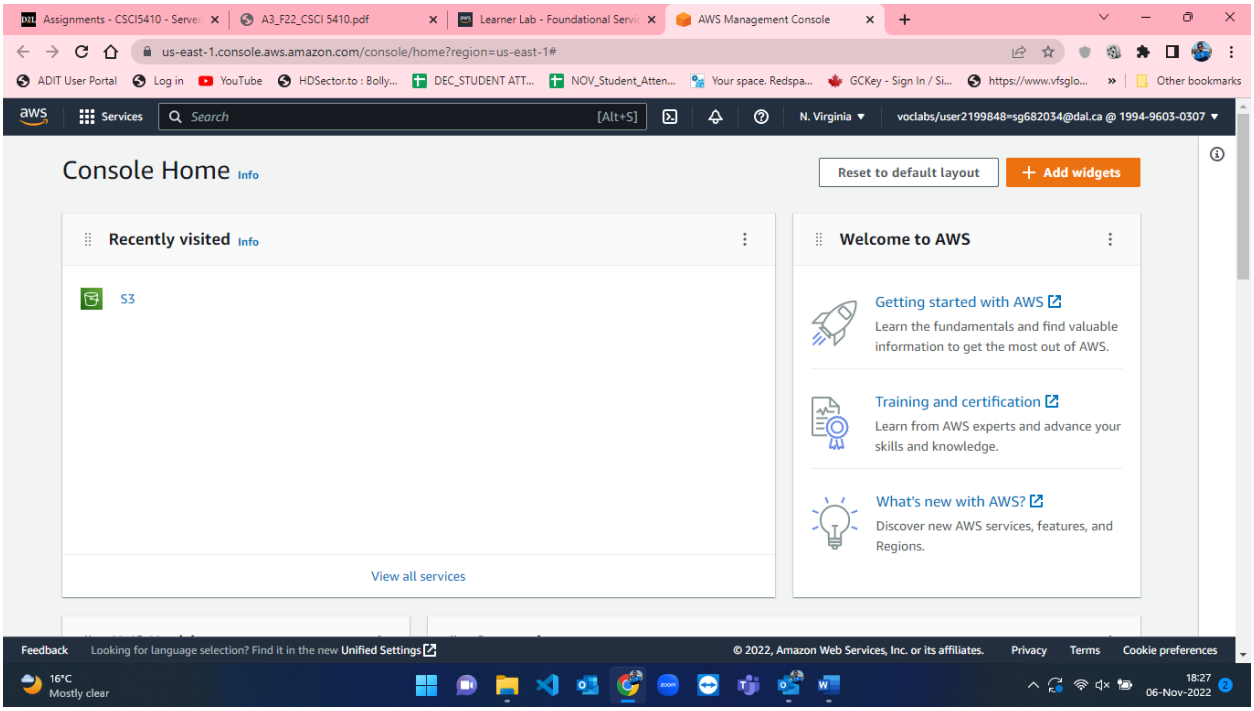

Figure 6: AWS Console Home.

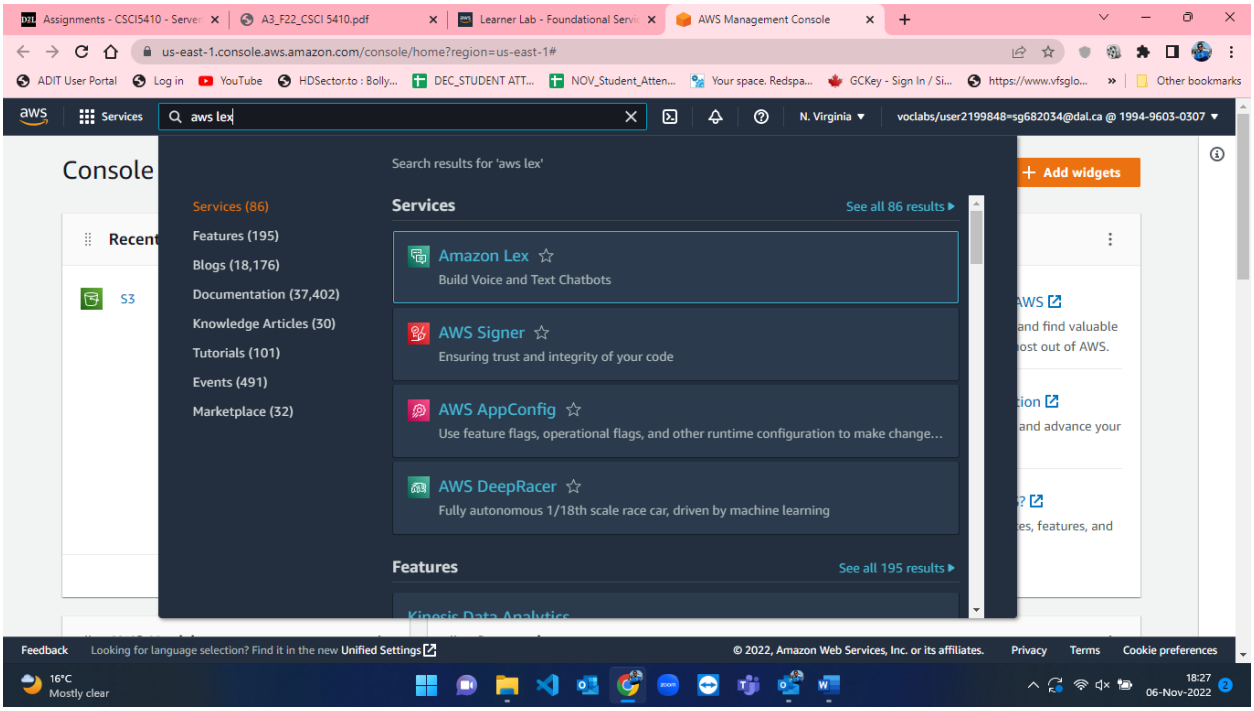

Figure 7: Searching for AWS Lex in search bar from Console Home in AWS.

When I selected AWS Lex, the homepage of AWS Lex [8] is opened, where it shows there are currently no bots by default in my AWS account. This can be seen in the below attached snapshot figure-8.

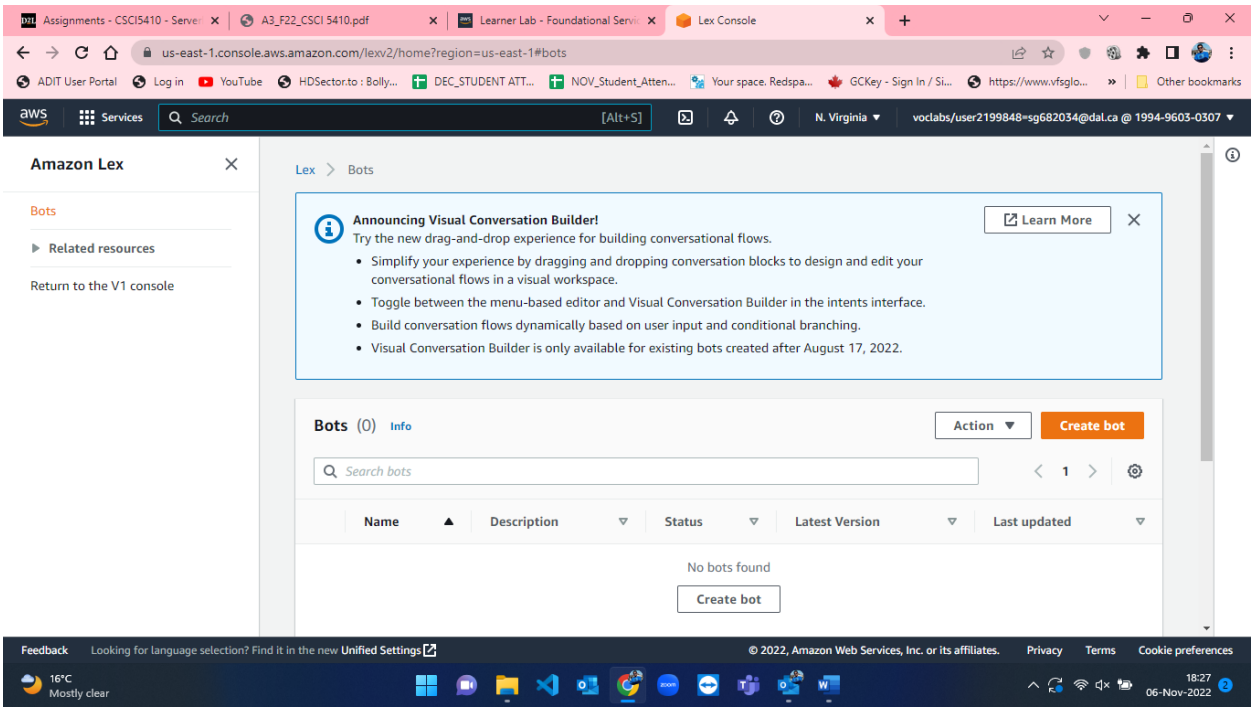

Figure 8: Homepage of AWS Lex with no bots by default.

- We now have to create a custom chatbot [1] to book office hours to meet a professor. For creating chatbot [1] I have selected “Create Bot” button which took me to the “Configure Bot settings” page where I have defined everything related to the custom chatbot. The empty “Configure Bot settings” page can be visualized in the presented figure-9 below.

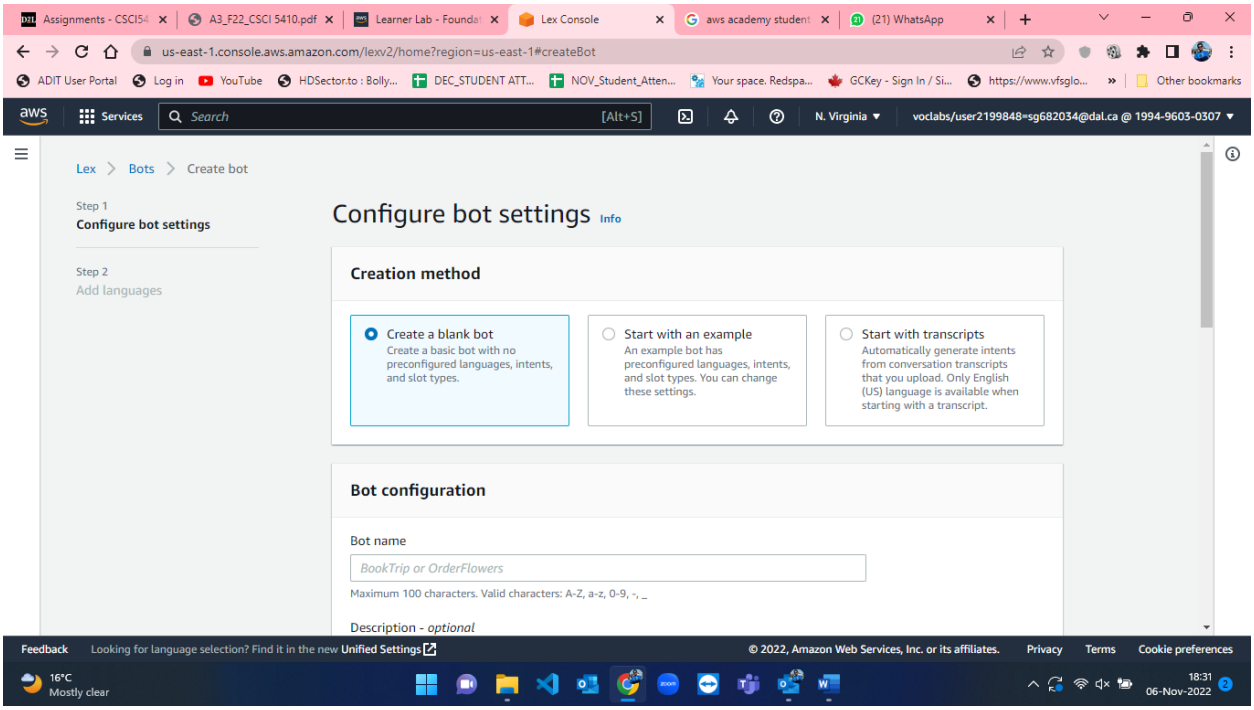

Figure 9: Blank Configure Bot Settings to create custom chatbot in AWS.

- The process for filling the fields in the bot creation is shown in the below mentioned figures from 10 to 14. I named the bot as BotProfAppointment and added a brief description for the bot (Figure-10). By default, for IAM permissions “Create a role with basic Amazon Lex permissions” is selected which I changed to “Use an existing role” and selected “Lab role” under role selection (Figure-11). After that, I selected “No” option for “Children’s Online Privacy Protection ACT (COPPA)” [1]. Idle Session timeout [1] was 5 minutes which I modified to 8 minutes (Figure-12). After pressing “Next” button, AWS took me to the language section where I can add languages for the chatbot (Figure-13). However, I have created the chatbot in default English language [1] only. Lastly, I hit on the “Create Bot” button available at the bottom of the language selection page and finally the bot is created (Figure-14).

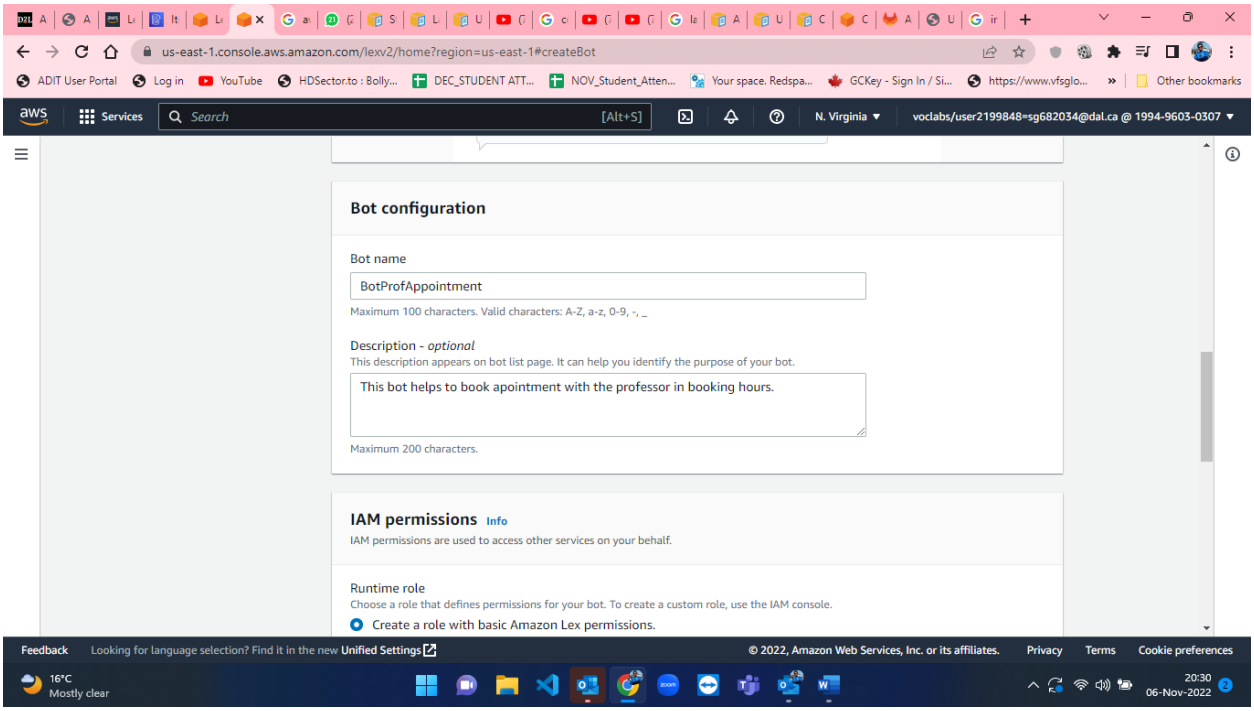

Figure 10: Adding name and description of the BotProfAppointment in AWS Lex.

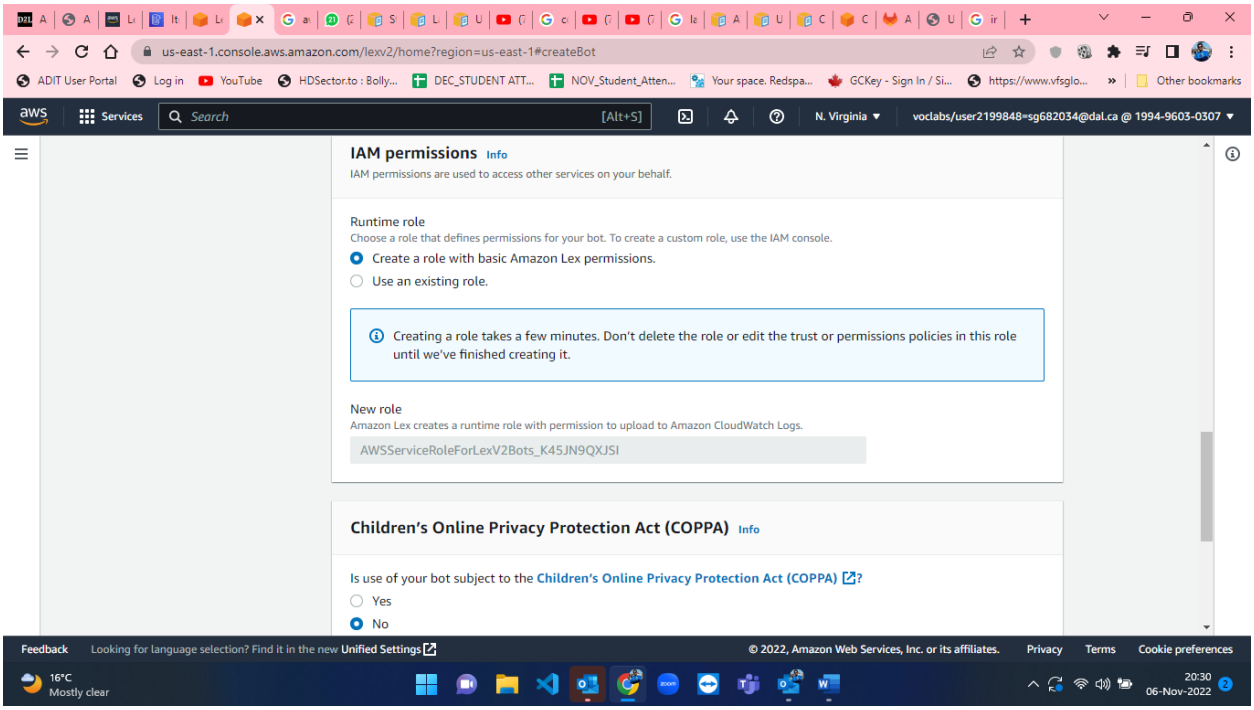

Figure 11: Default IAM permission in AWS Lex.

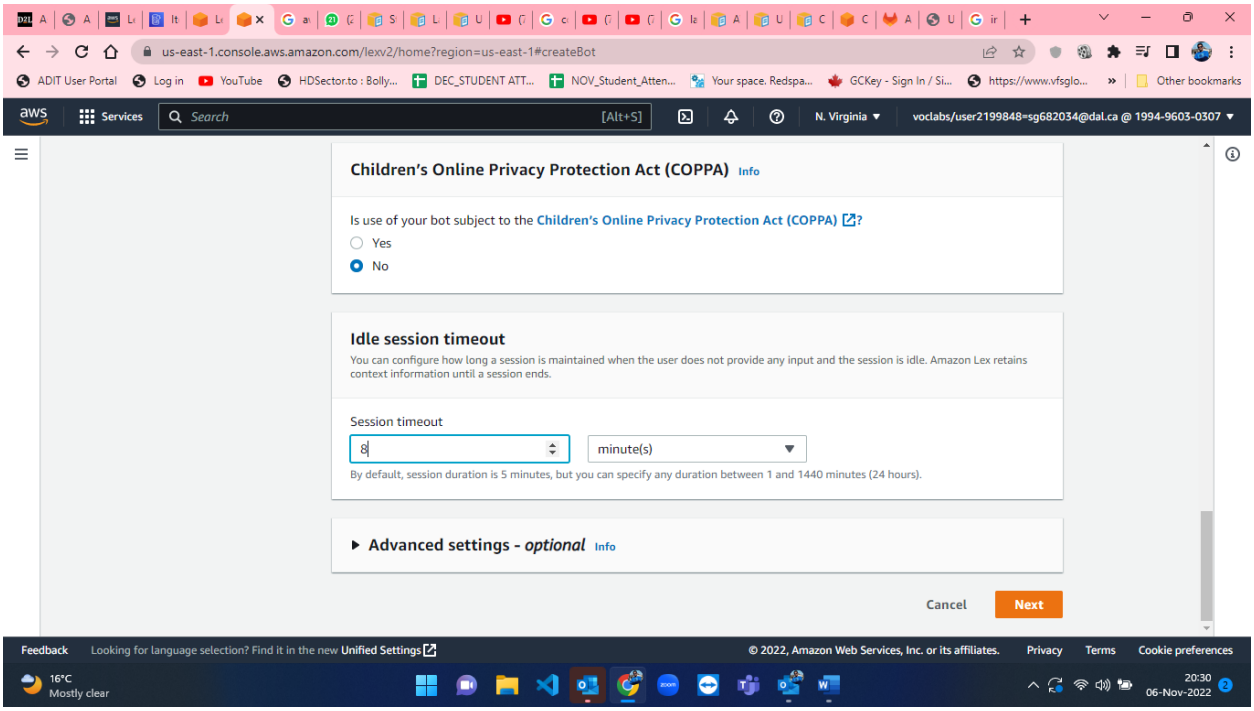

Figure 12: Idle session timeout for bot in AWS Lex.

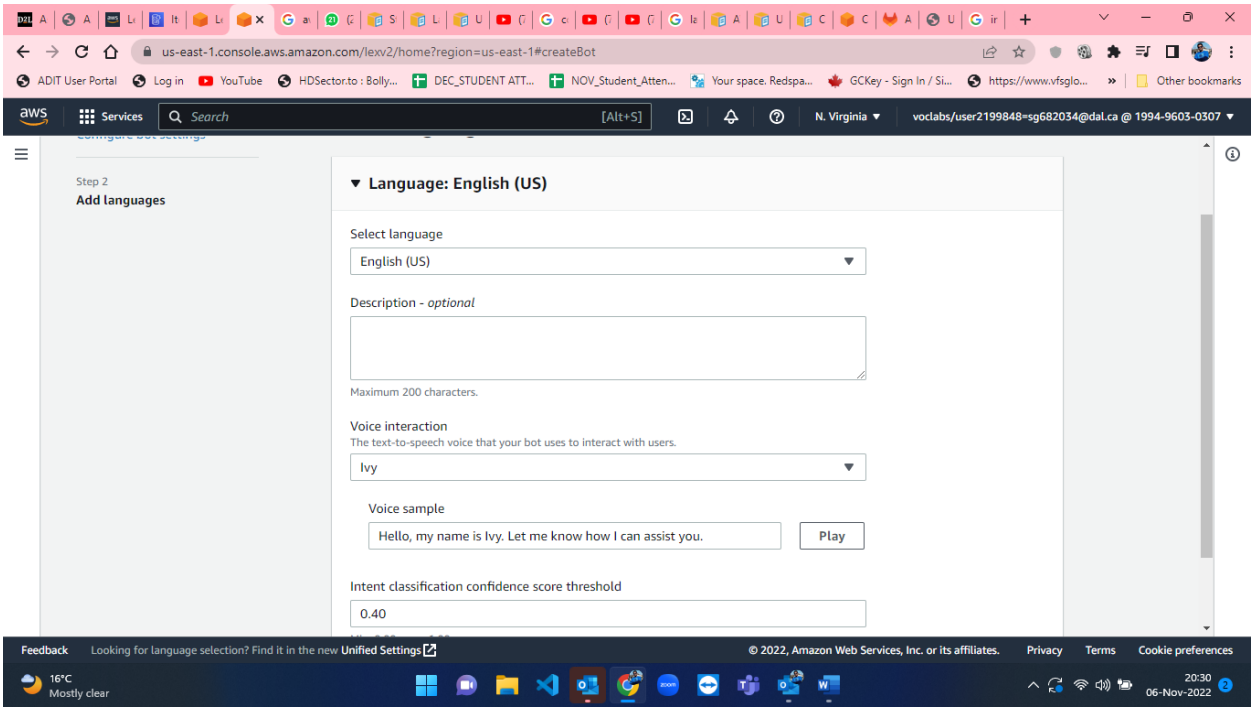

Figure 13: Bot language selection in AWS Lex.

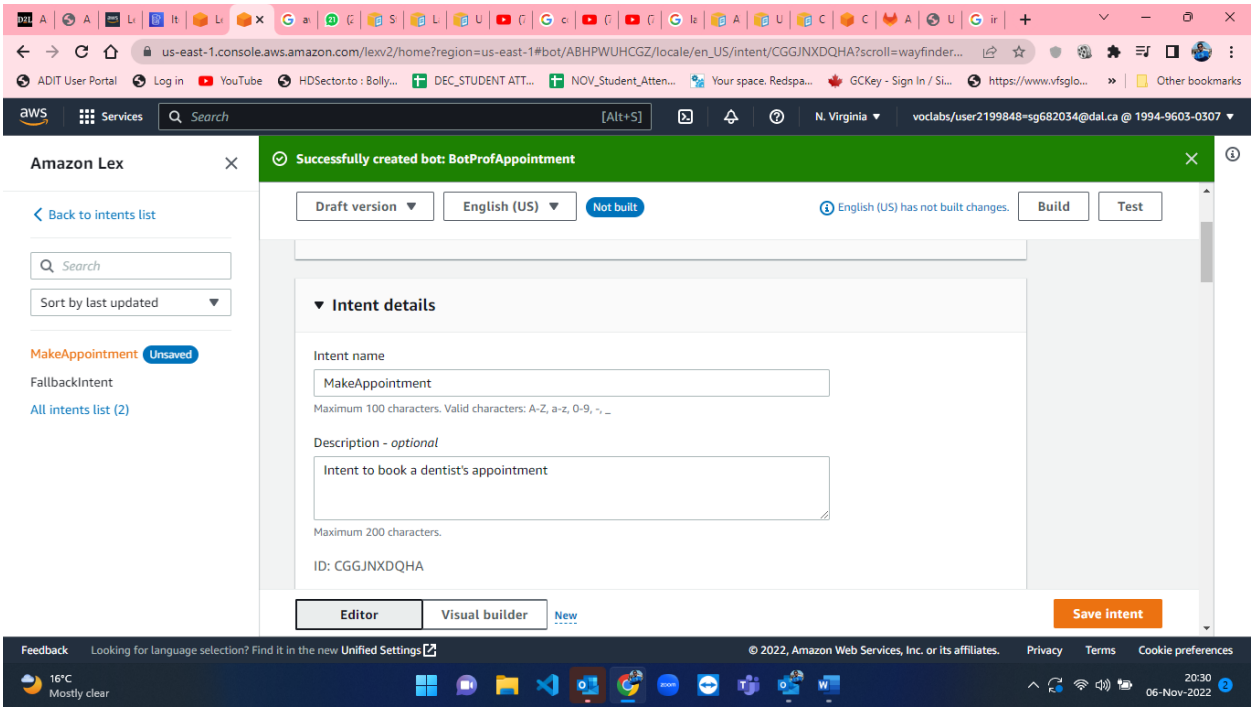

Figure 14: BotProfAppointment is successfully created.

- After creating the bot, we have to create intent [2], enter sample utterances [4], slots [3], confirmation and closing response for the bot. The details and steps for entering the details in the intent are shown from figure-15 to figure figure-19.
- Intent [2] is defined as the purpose of the Bot. I have named intent as MakeAppointment and added brief description of the intent.

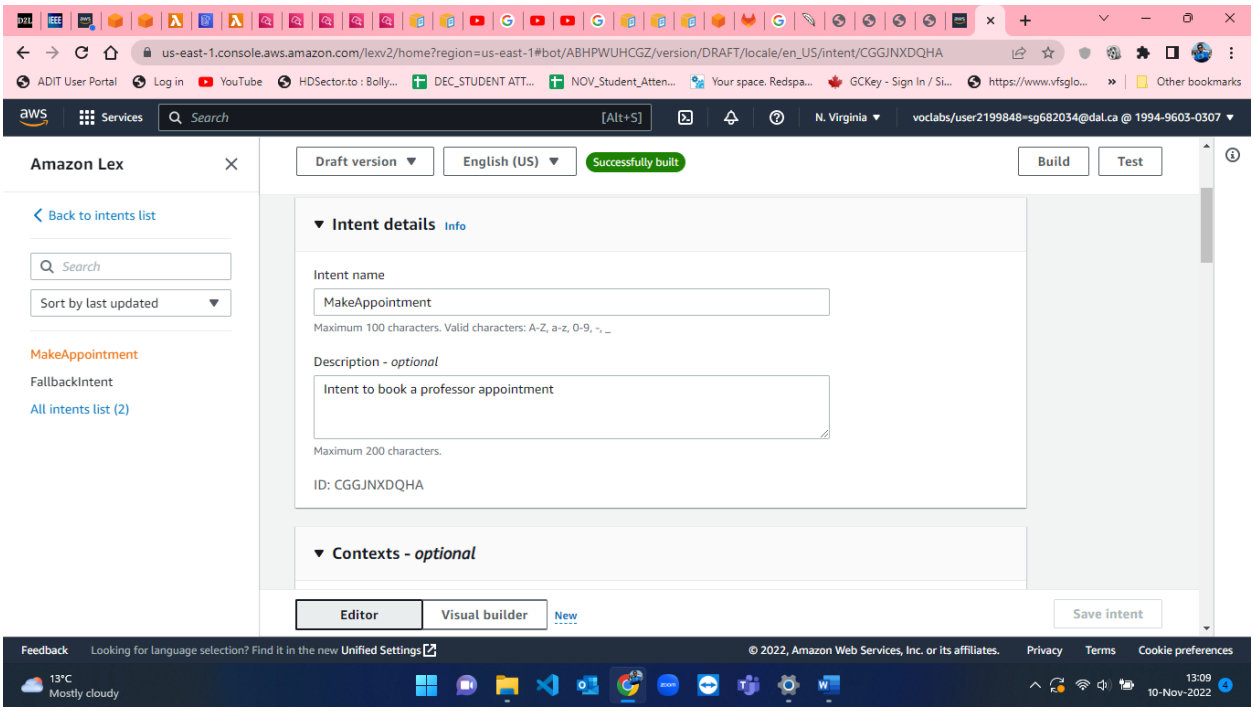

Figure 15: Intent details in Chatbot.

- Sample utterances [4] are the phrases that we expect a user to type to invoke the intent. I have added a few sample utterances which are displayed in the below image(figure-16).

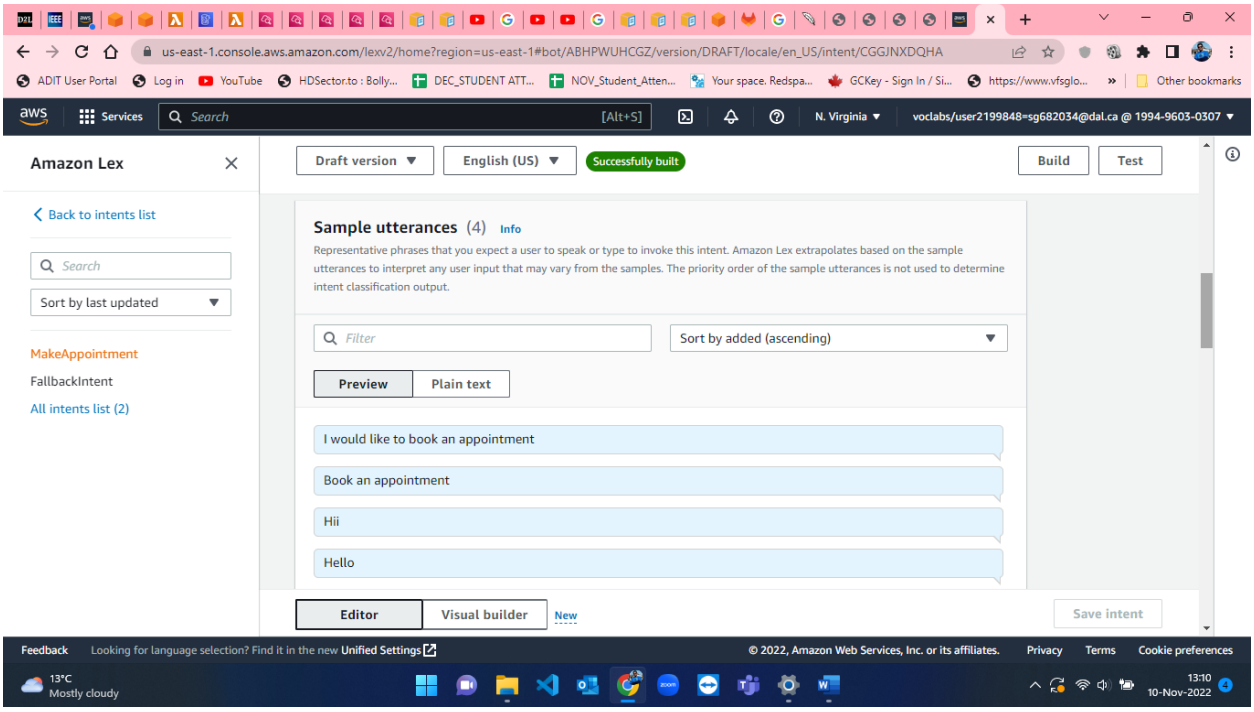

Figure 16: Sample utterances in AWS Lex

- Slots [3] are the information that a bot needs to fulfill the intent. The bot usually prompts for slots required for the given intent. Figure-17 shows the Date and Time prompt used by me for booking office hours with professor.

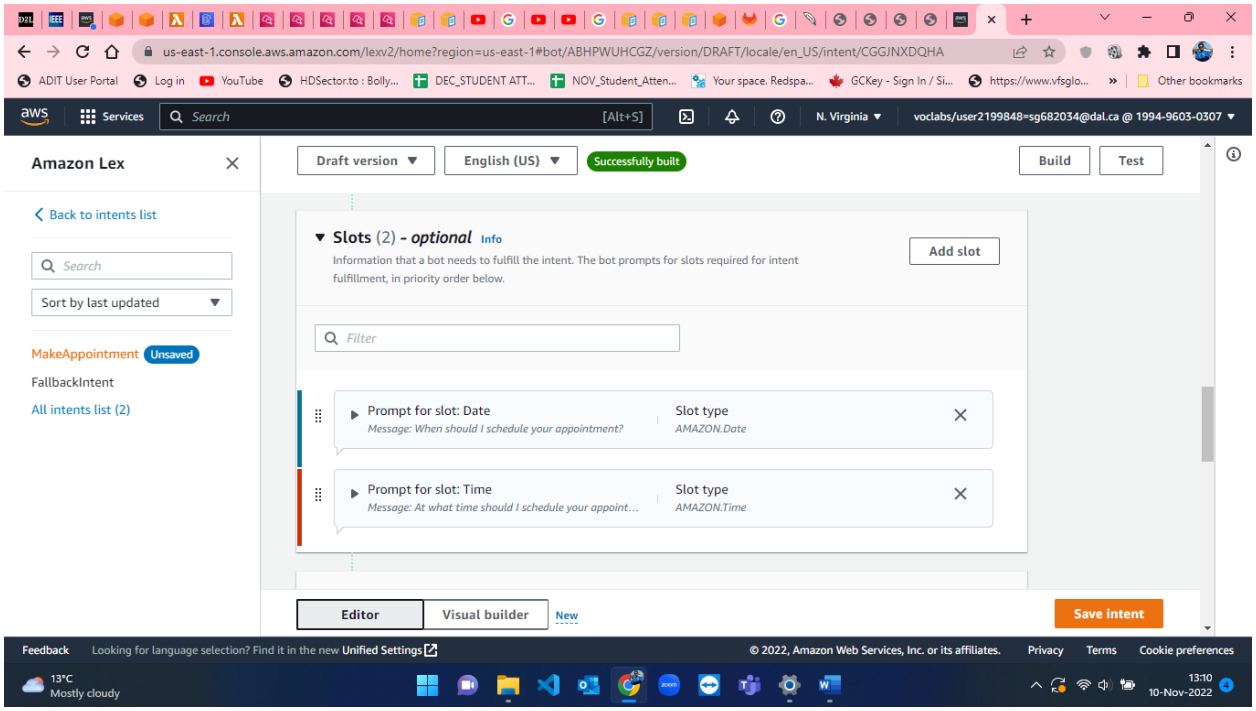

Figure 17: Slots in AWS Lex

- Confirmation message helps to clarify whether the user wants to fulfill the intent or cancel the intent. The below displayed figure-18 represents the confirmation message that I have used for the user.

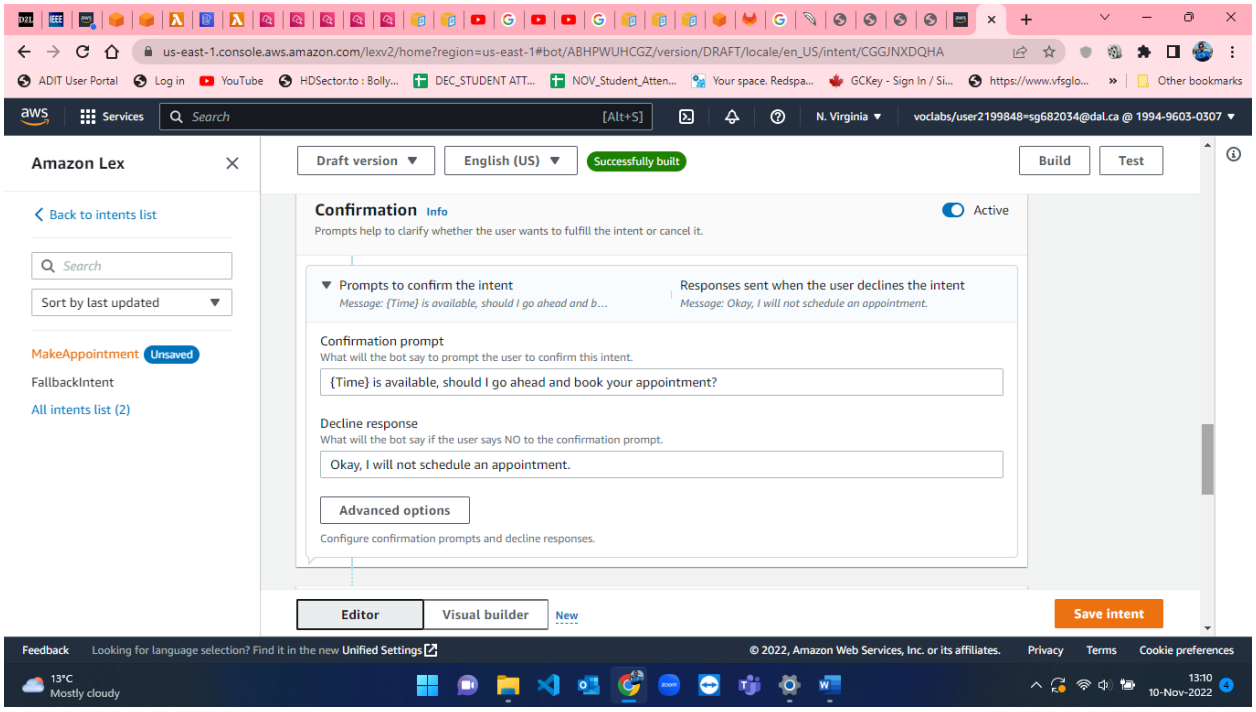

Figure 18: Confirmation message for user in AWS Lex.

- When closing the intent, we define the closing response for the end greetings. Here, I have displayed the message for successful appointment booking confirmation for the given date and time which is shown in figure-19.

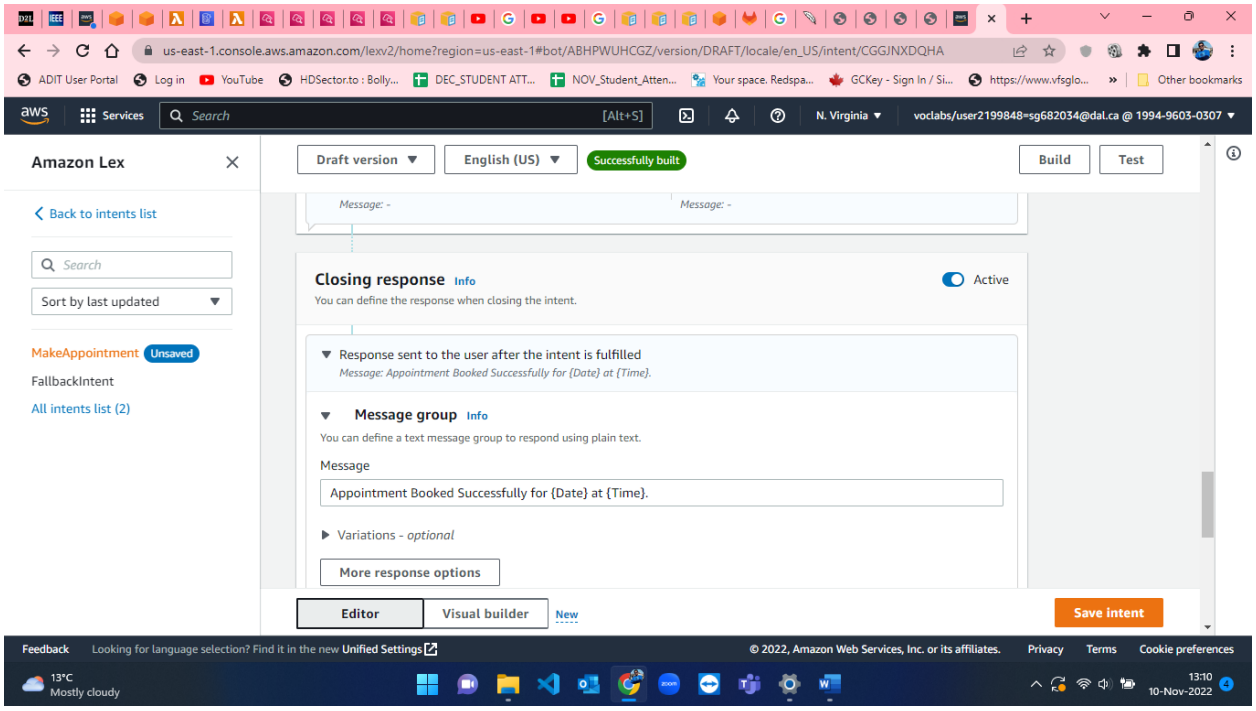

Figure 19: Closing response in AWS Lex

- After performing the steps which are shown in figures 15 to 19, we have to save intent and then we will build [5] the created intent and test [5] the chatbot. This process of building the intent and testing is shown in below figures from 20 to 26.

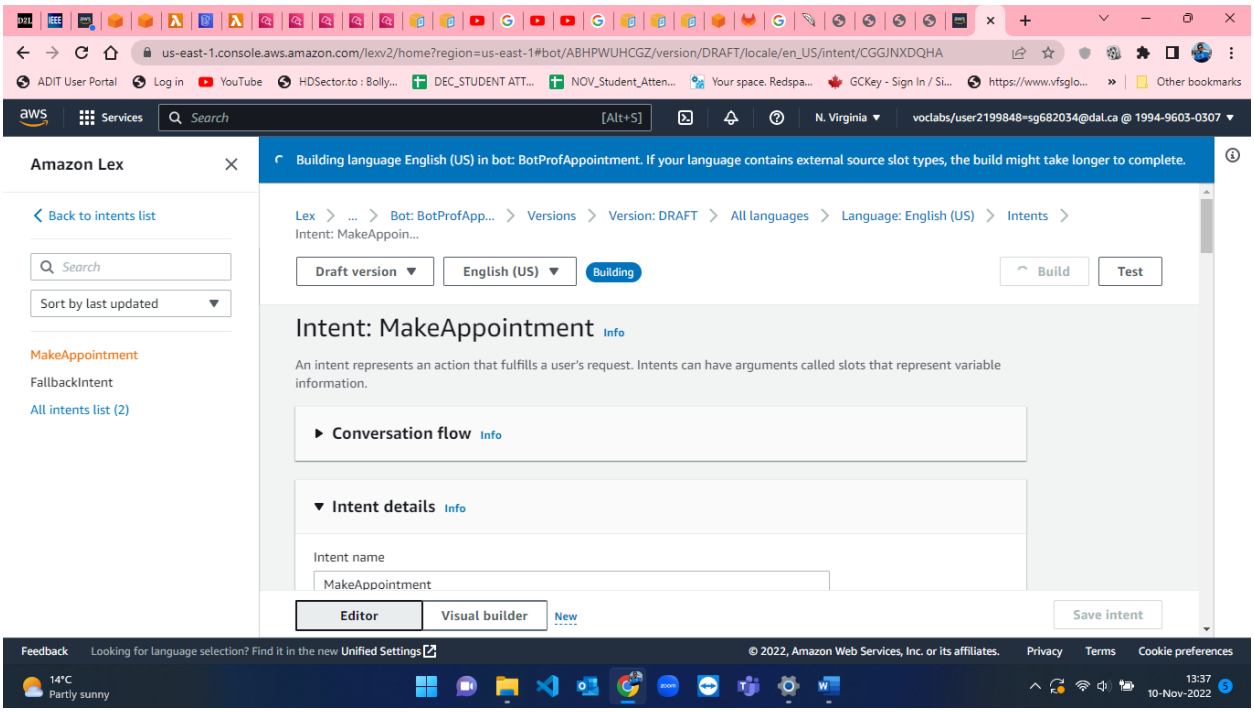

Figure 20: Building the intent for the bot in AWS Lex.

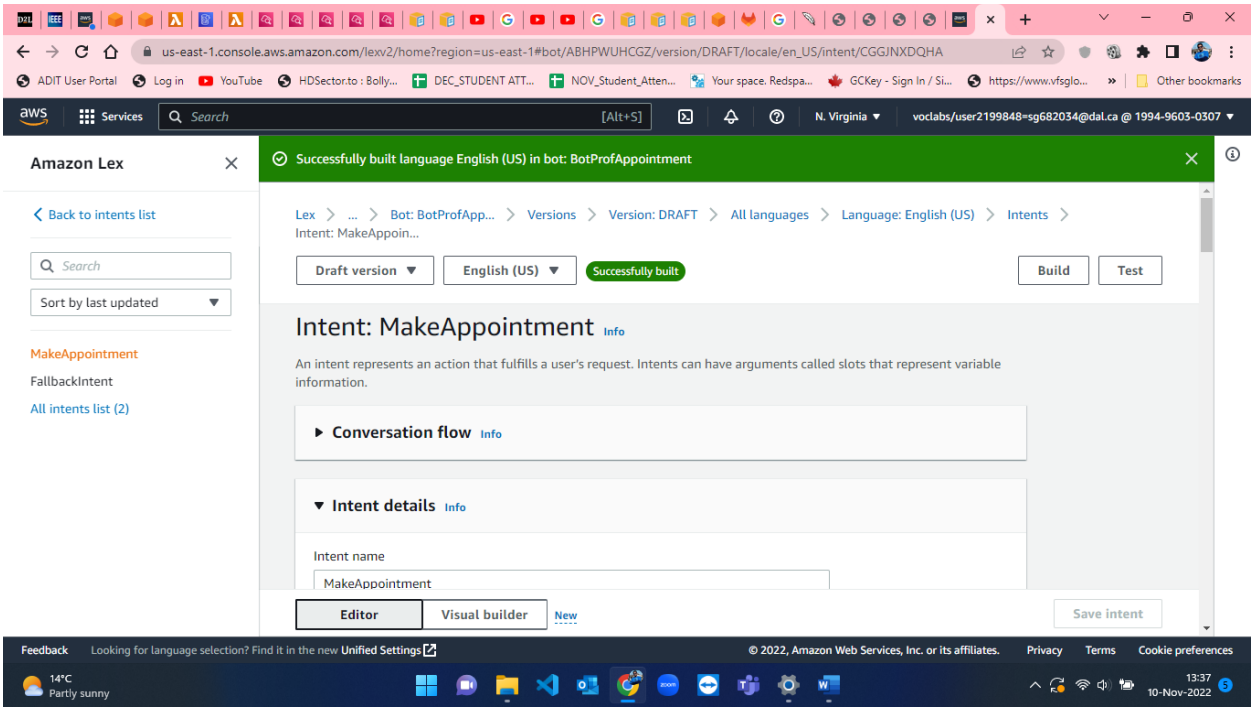

Figure 21: Bot Built successful in AWS Lex.

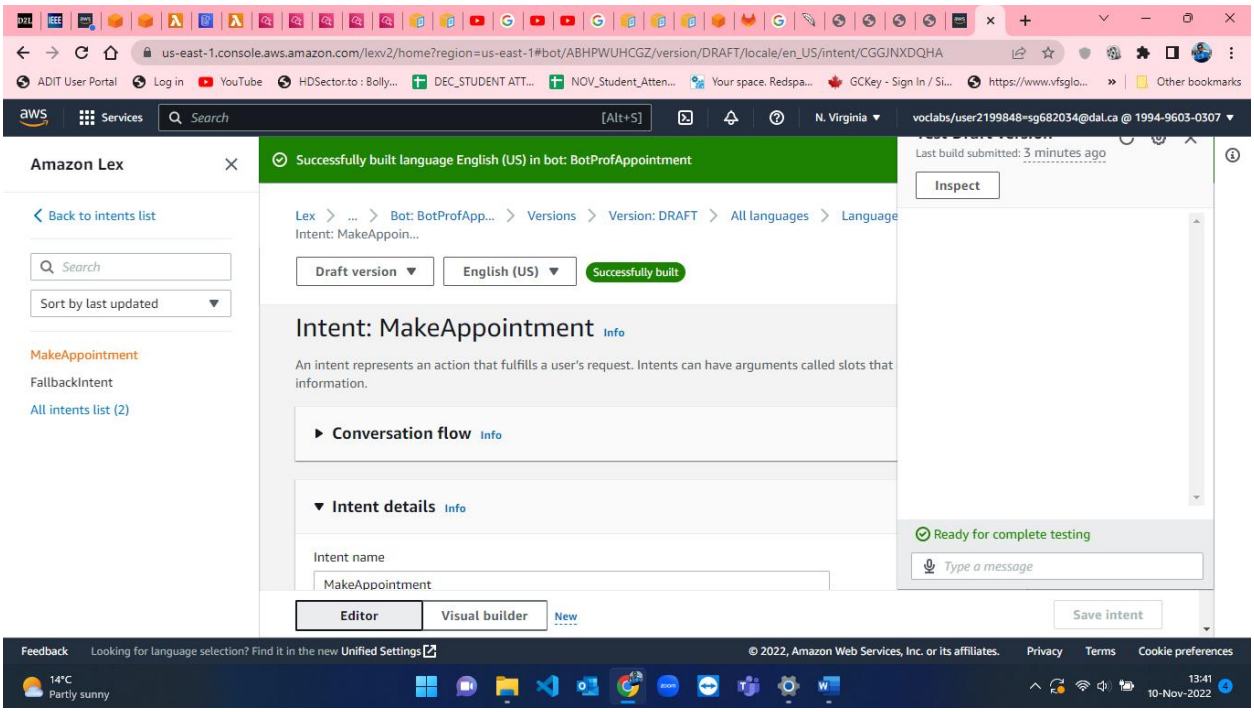

Figure 22: Chatbot Screen-1

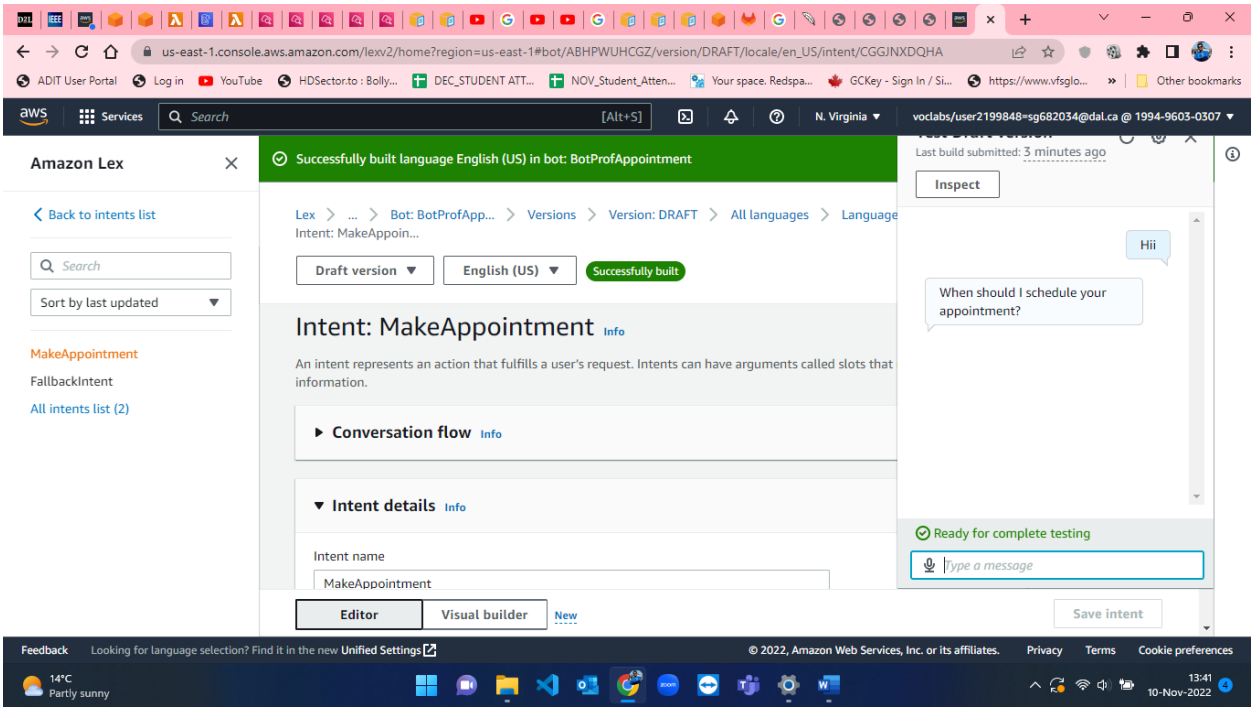

Figure 23: Chatbot Screen-2

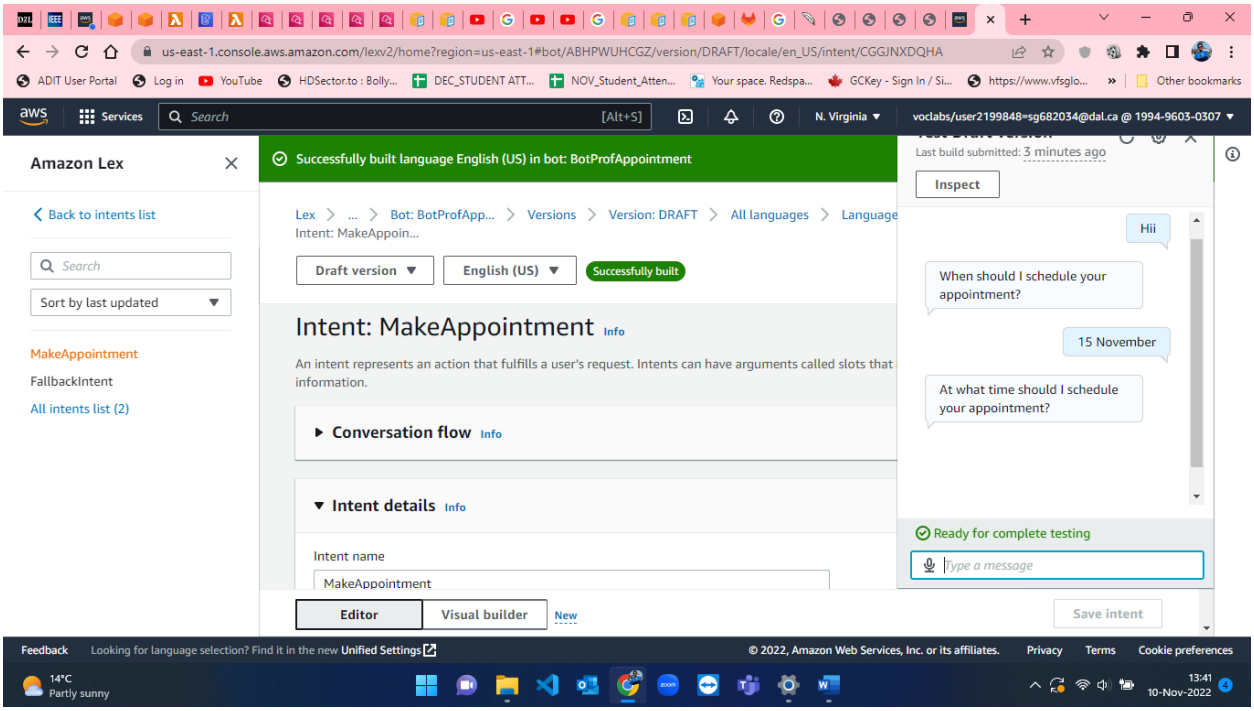

Figure 24: Chatbot Screen-3

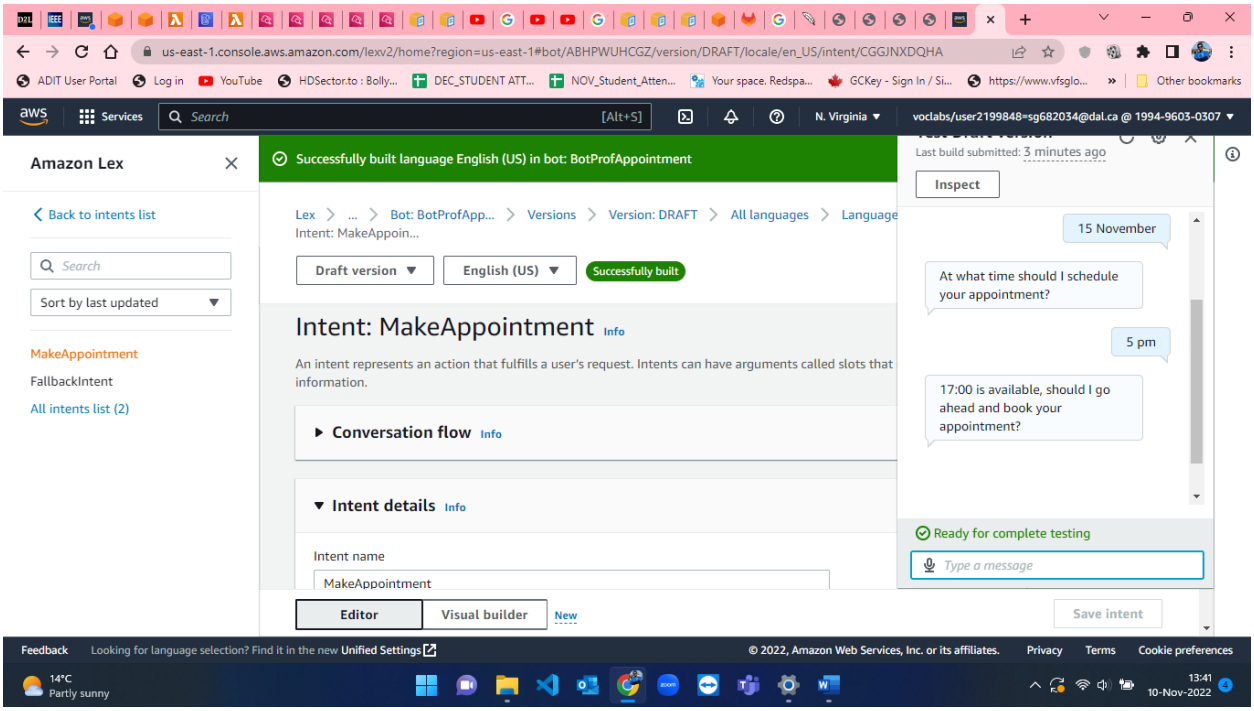

Figure 25: Chatbot Screen-4

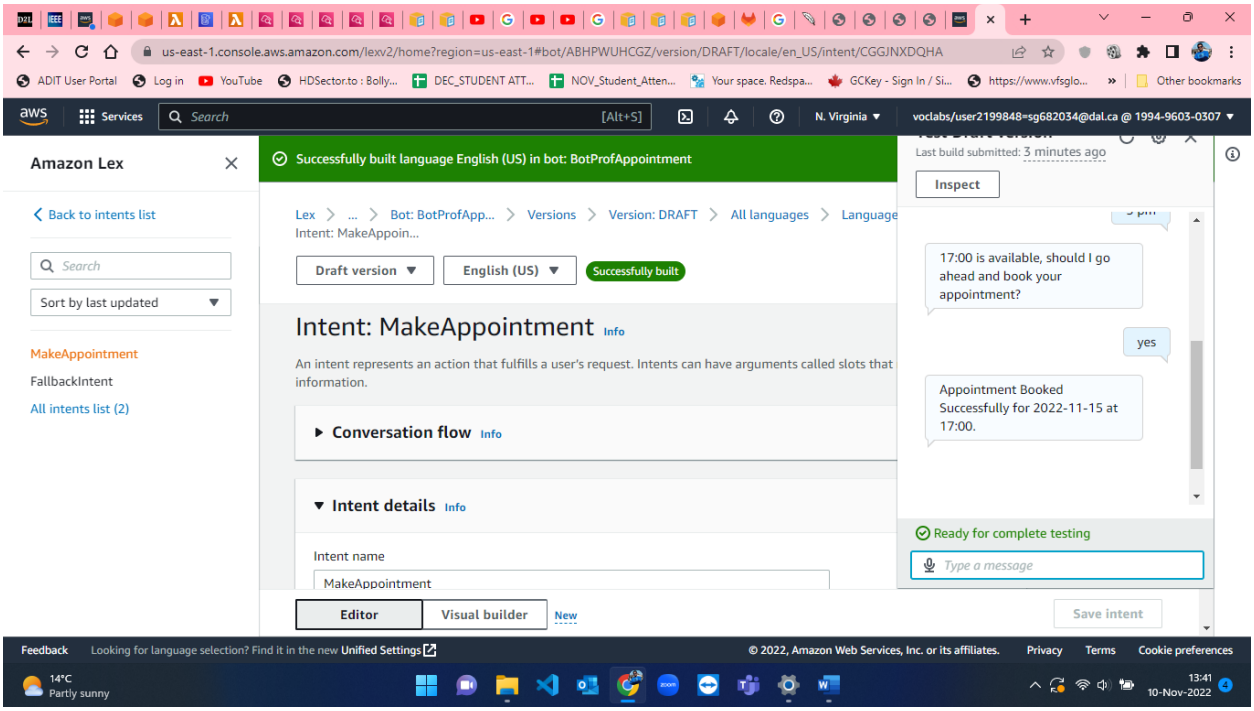

Figure 26: Chatbot Screen-5

Custom chatbot using AWS Lex to verify student's identity.

The login process and module selection process remain the same as we have done in the creation of the first bot for booking office hours with professor. So, now I'll be explaining the unique steps for creating and testing BotStdLookup.

Step-1: Creating Bot for student detail verification.

- The configuration for creating bot [1] for BotStdLookup is shown from figure-27 to figure-32.

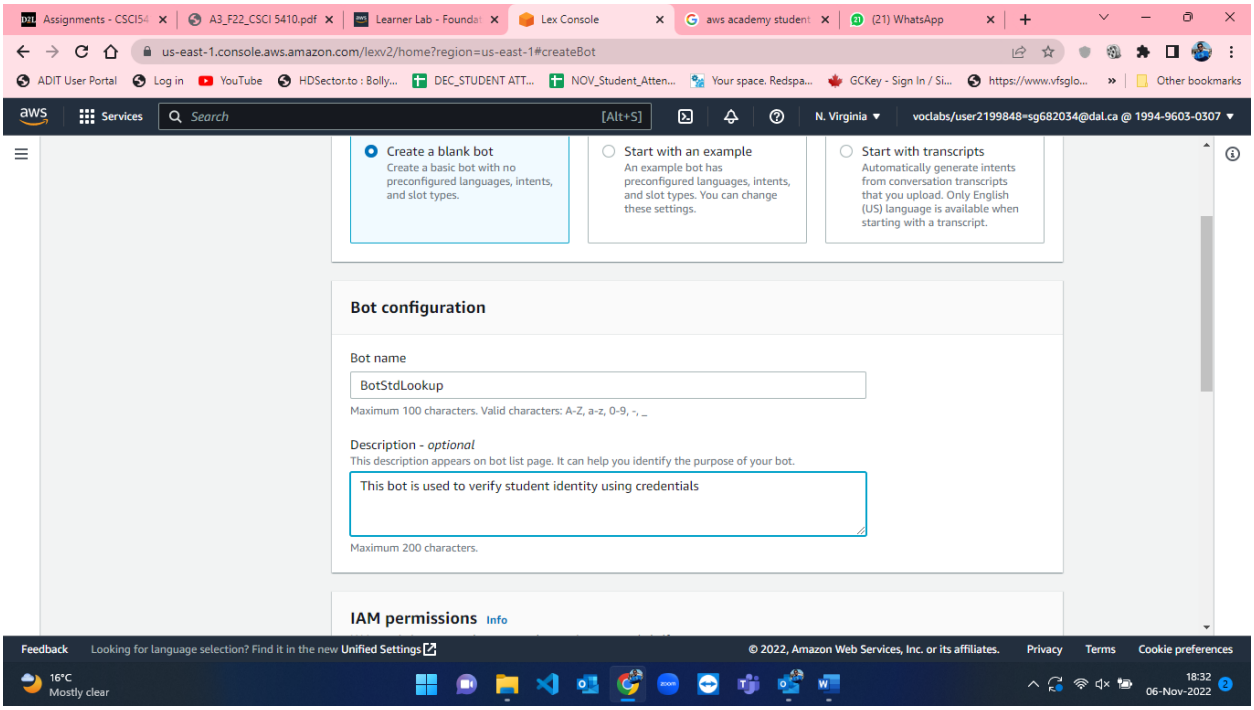

Figure 27: Configuring BotStdLookup-1.

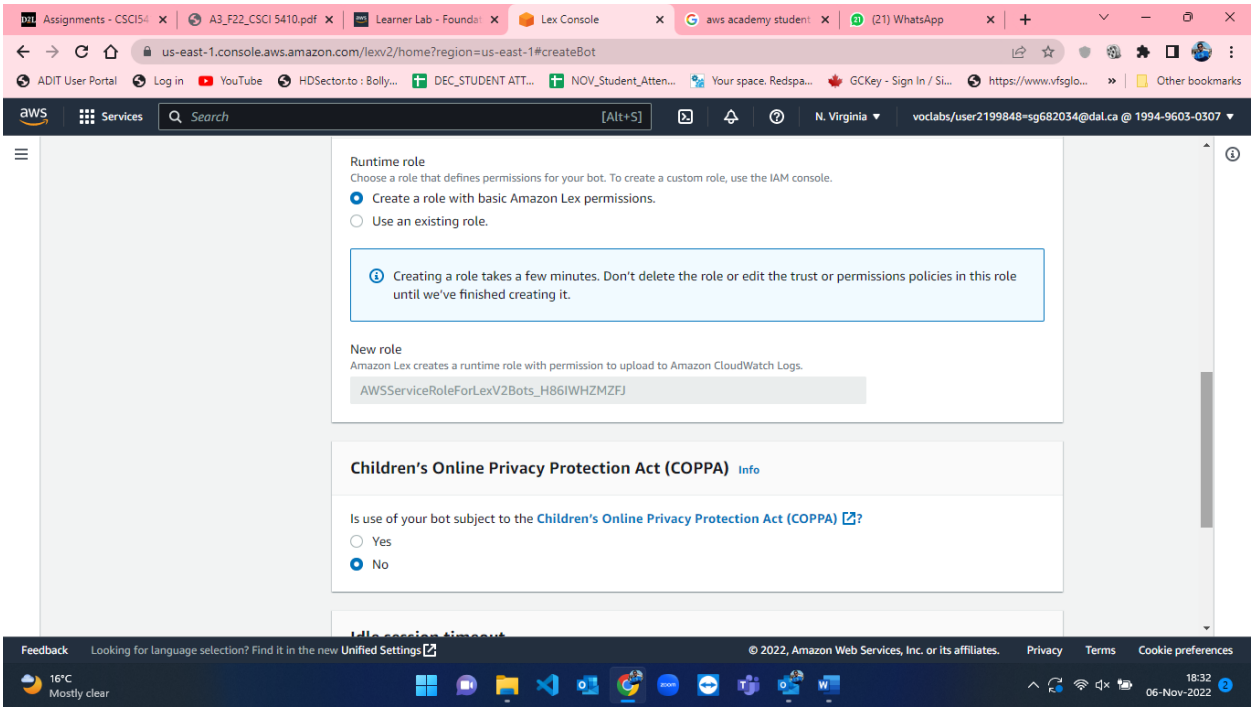

Figure 28: Configuring BotStdLookup-2.

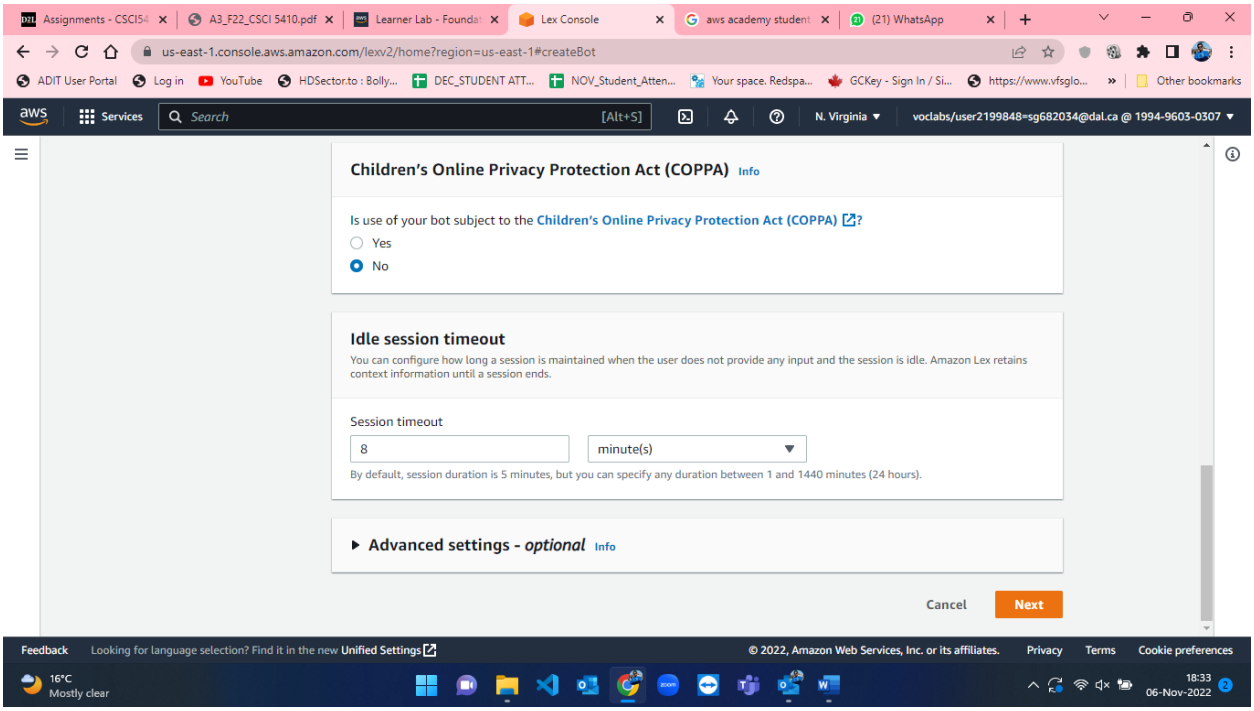

Figure 29: Configuring BotStdLookup-3.

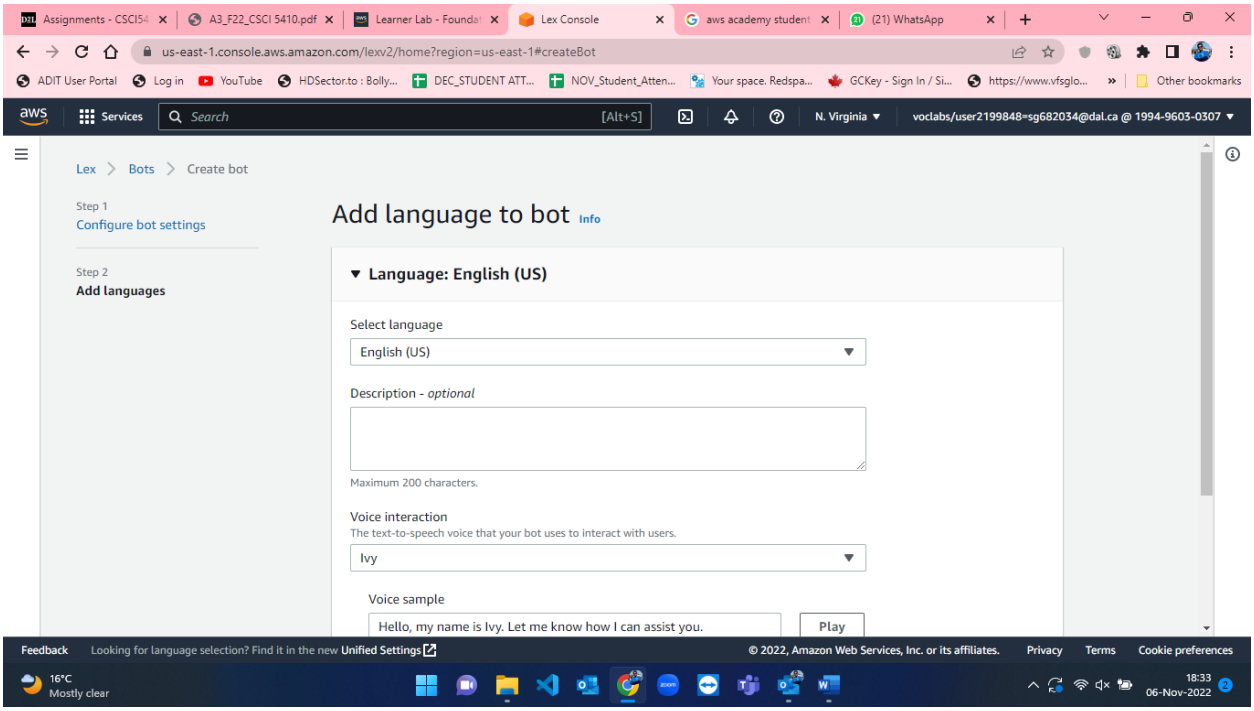

Figure 30: Configuring BotStdLookup-4.

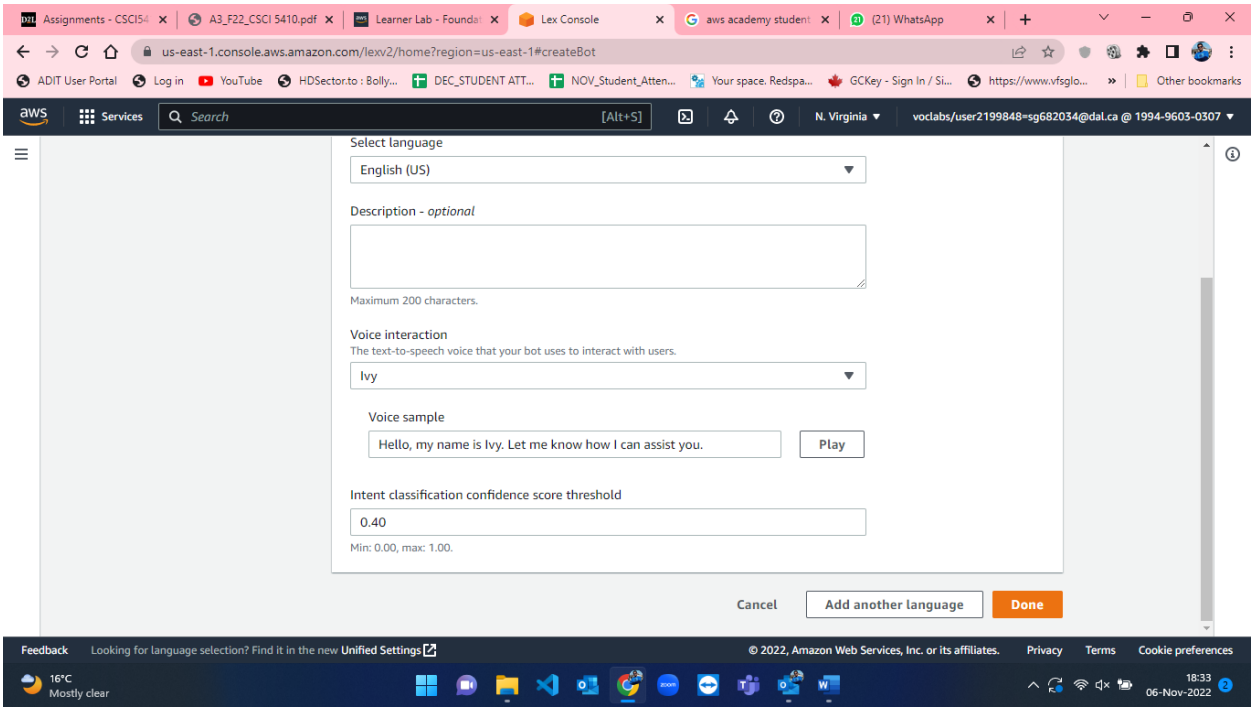

Figure 31: Configuring BotStdLookup-5.

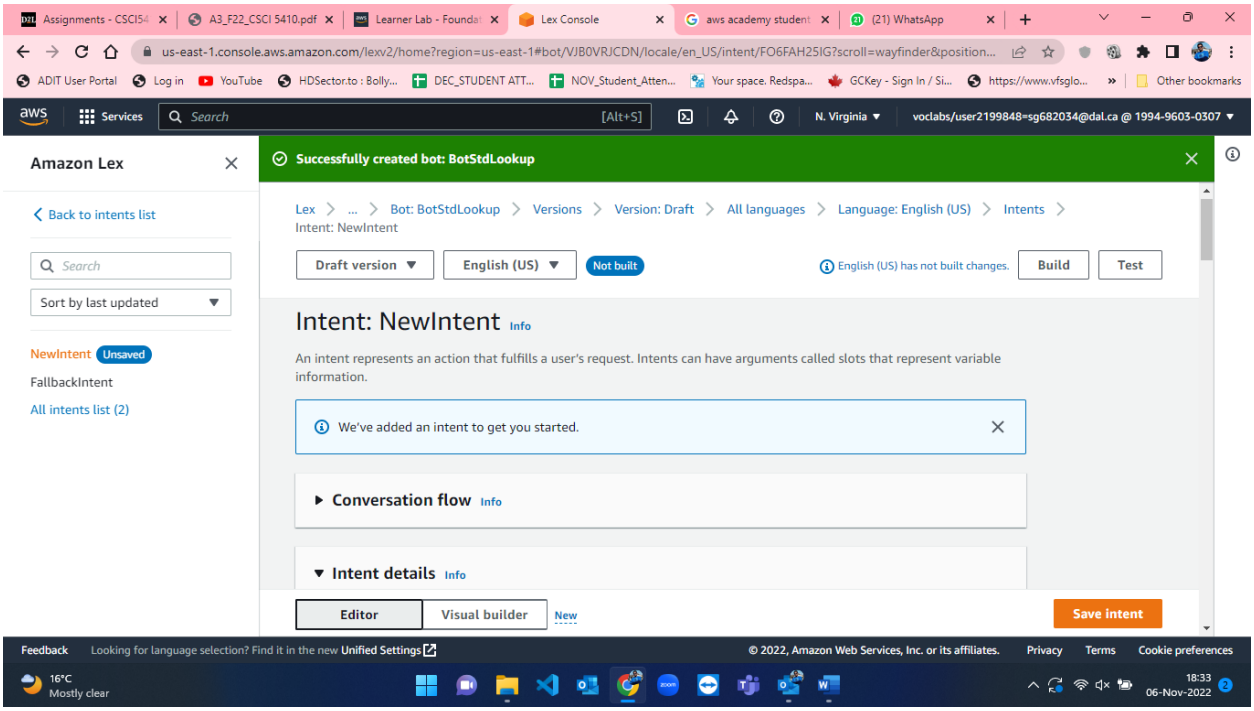

Figure 32: Configuring BotStdLookup-6.

- After successful creation of bot, we have to add intent details [2], sample utterances [4], slots [3], confirmation prompt and other details. The steps for that procedure are shown below from figure-33 to figure-38.

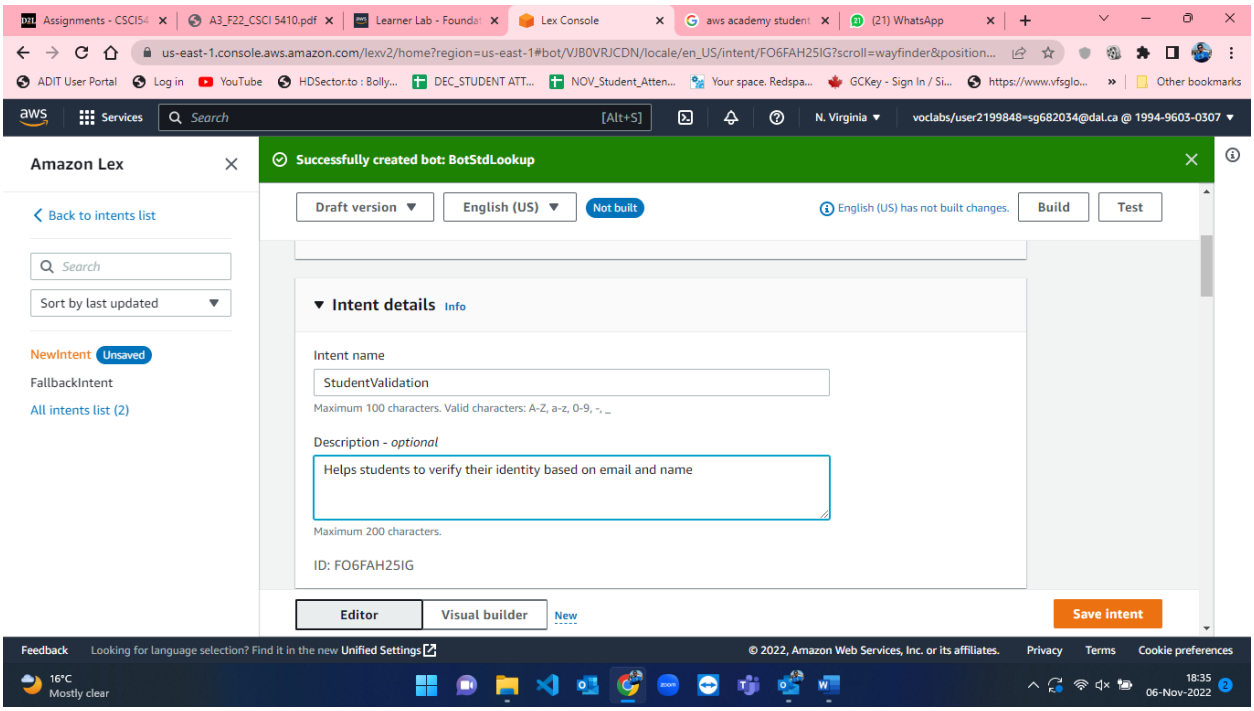

Figure 33: Intent name and description for BotStdLookup in AWS Lex.

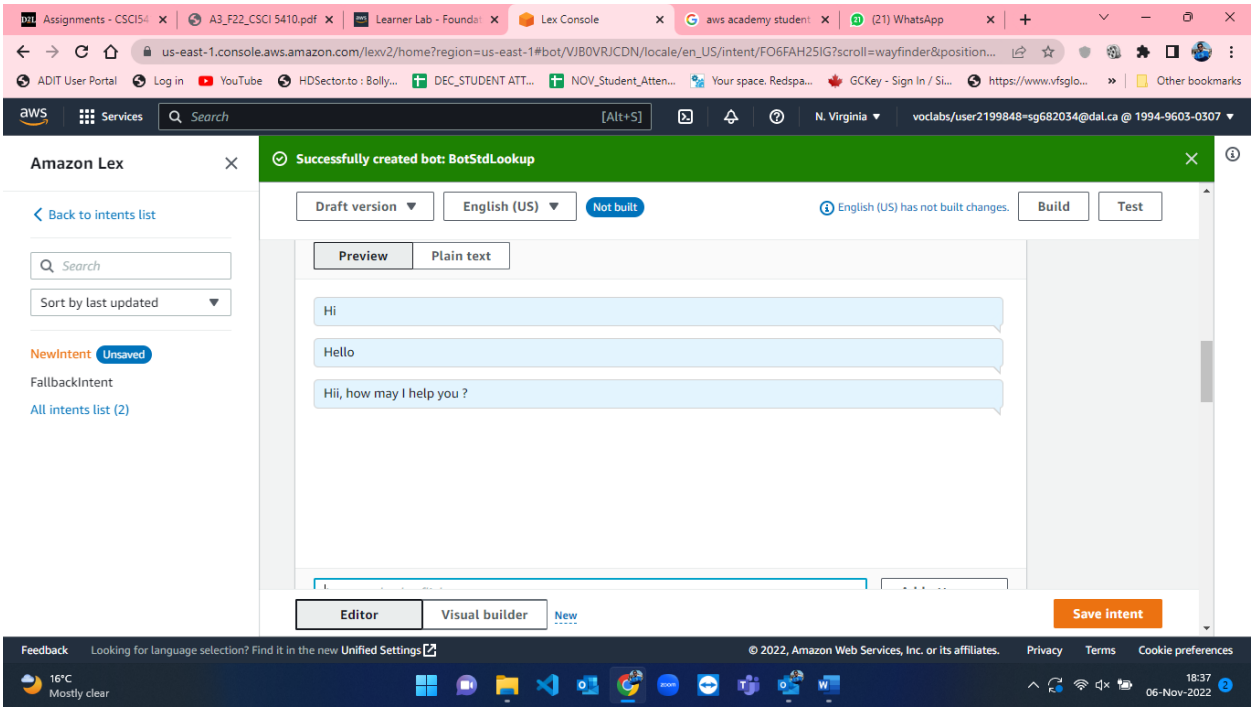

Figure 34: Sample utterances for BotStdLookup.

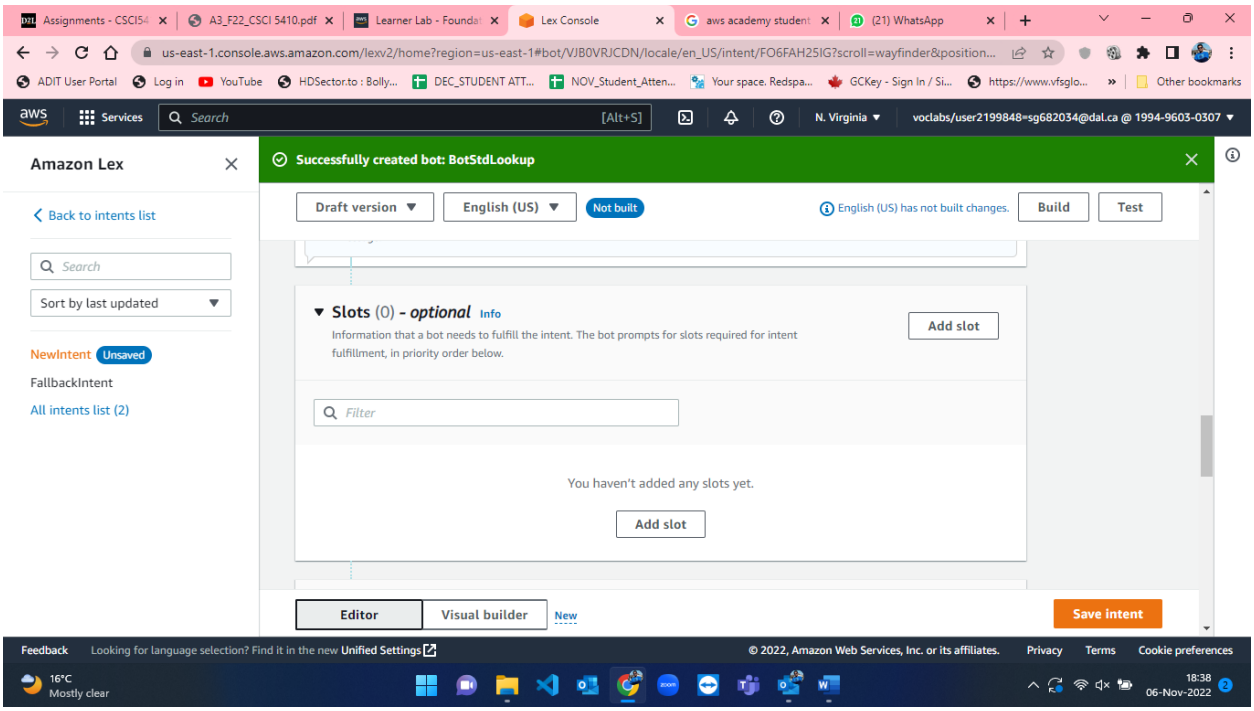

Figure 35: Initial empty slots in AWS Lex.

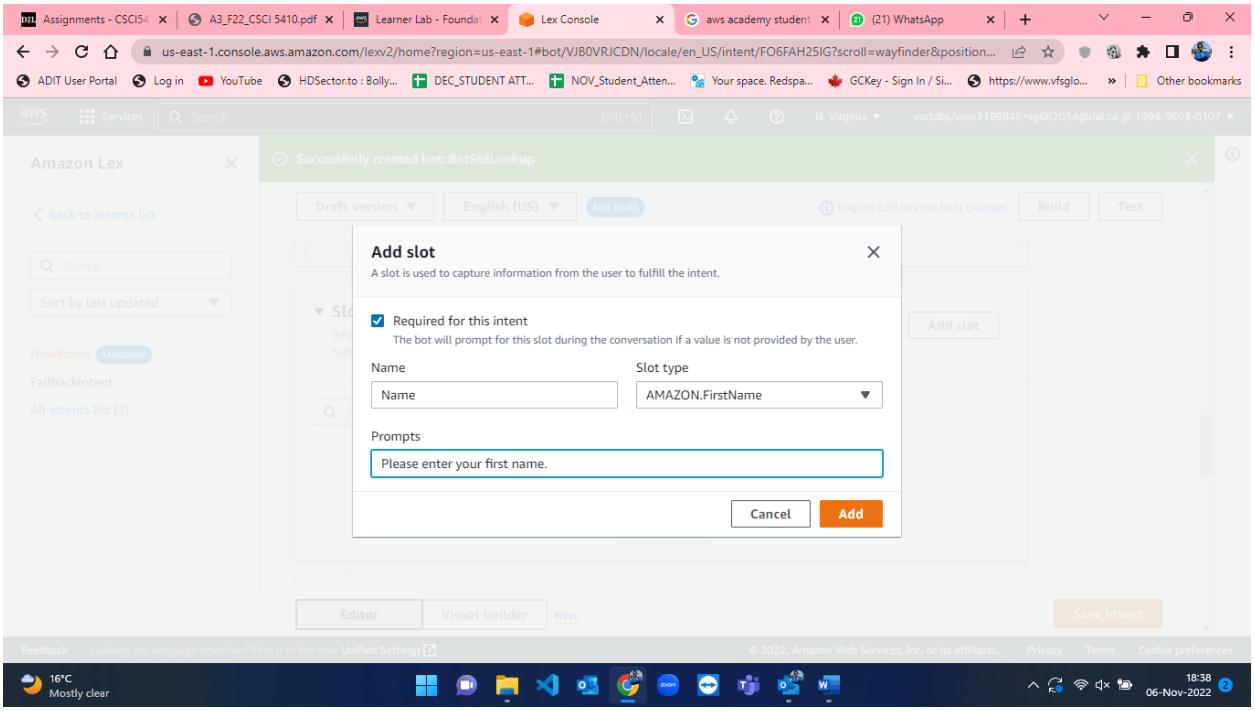

Figure 36: Slot-1 of Name field for BotStdLookup in AWS Lex.

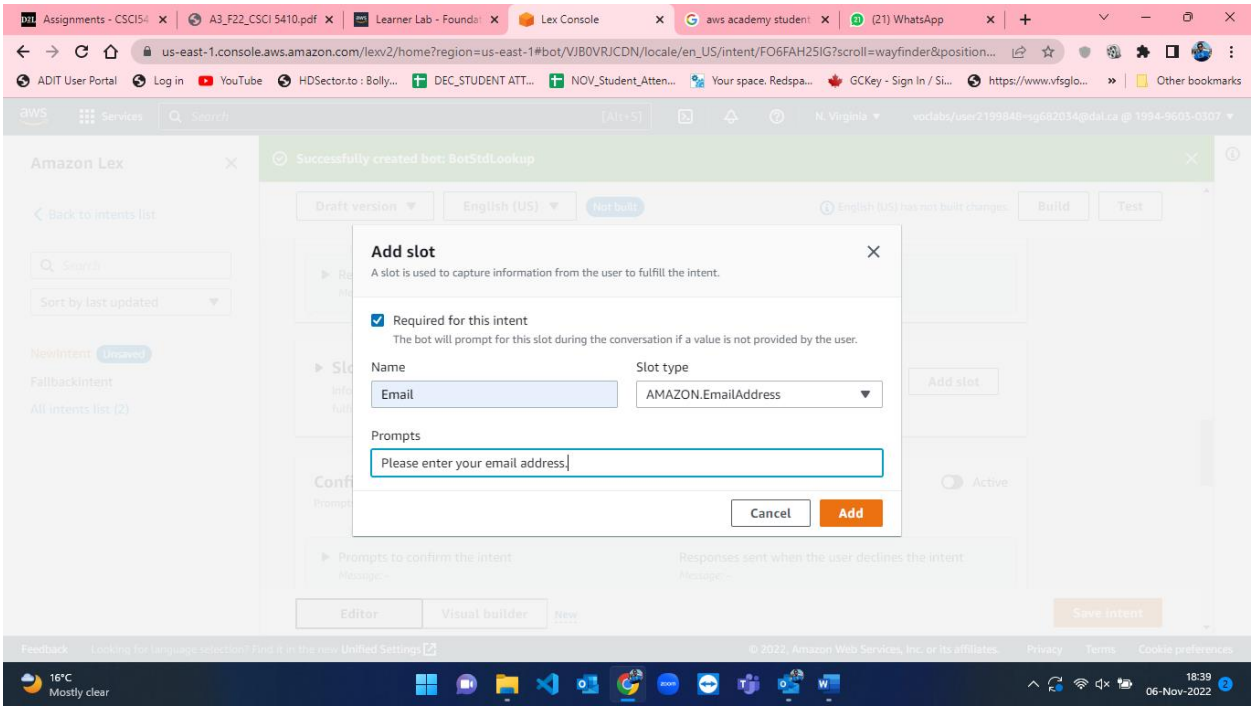

Figure 37: Slot-2 of Email field for BotStdLookup in AWS Lex.

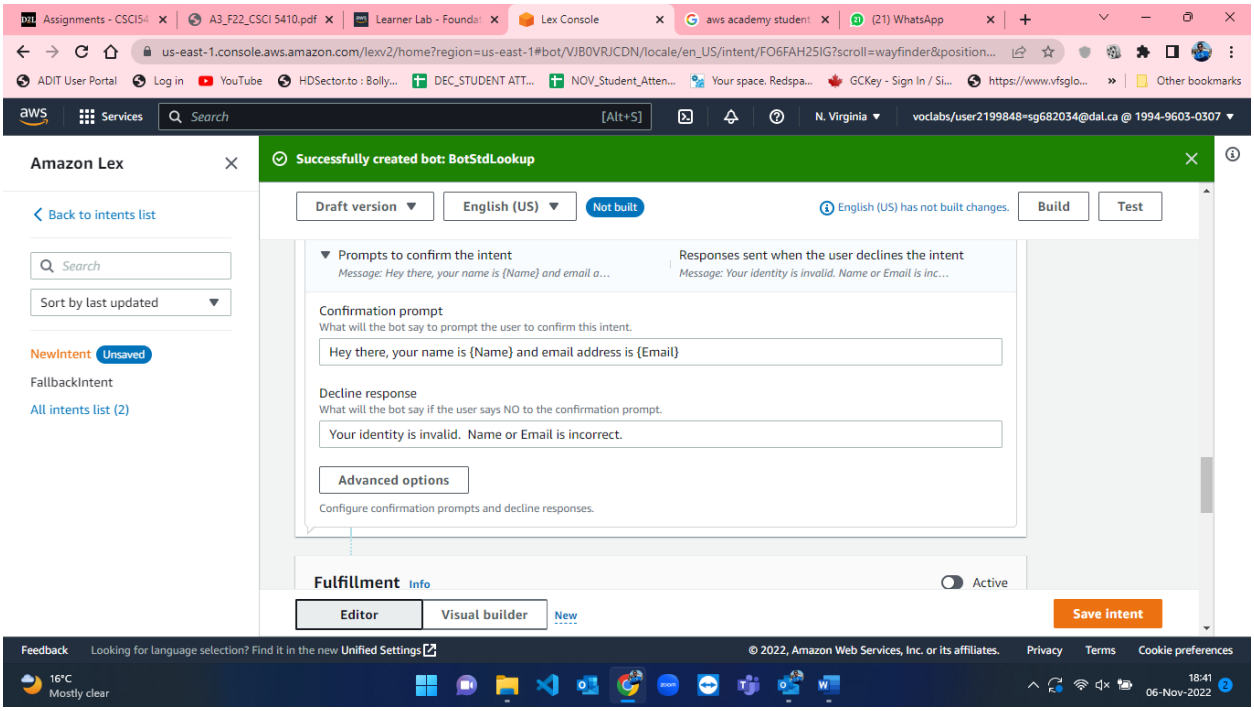

Figure 38: Successful creation of bot - BotStdLookup.

Step-2: Creating DynamoDB database for storing student details.

After creating the bot, we now have to set up DynamoDb database [9] for storing student details into the database such as name, email, and id. The steps for creating the database and storing the values in the table are explained below with figures.

- Firstly, I have searched for DynamoDb from the search bar on the top of the window(figure-39).

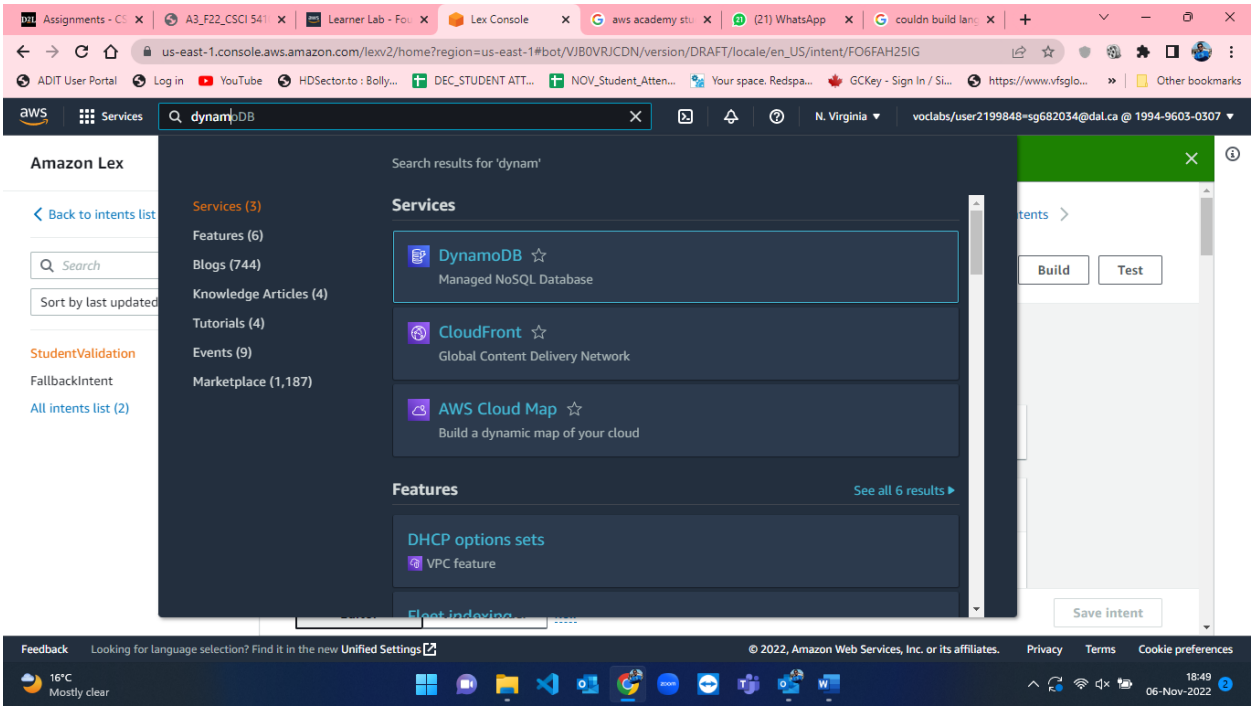

Figure 39: DynamoDB database search in AWS.

- After clicking on DynamoDB from the search bar, we came to DynamoDB homepage which is shown in figure-40.

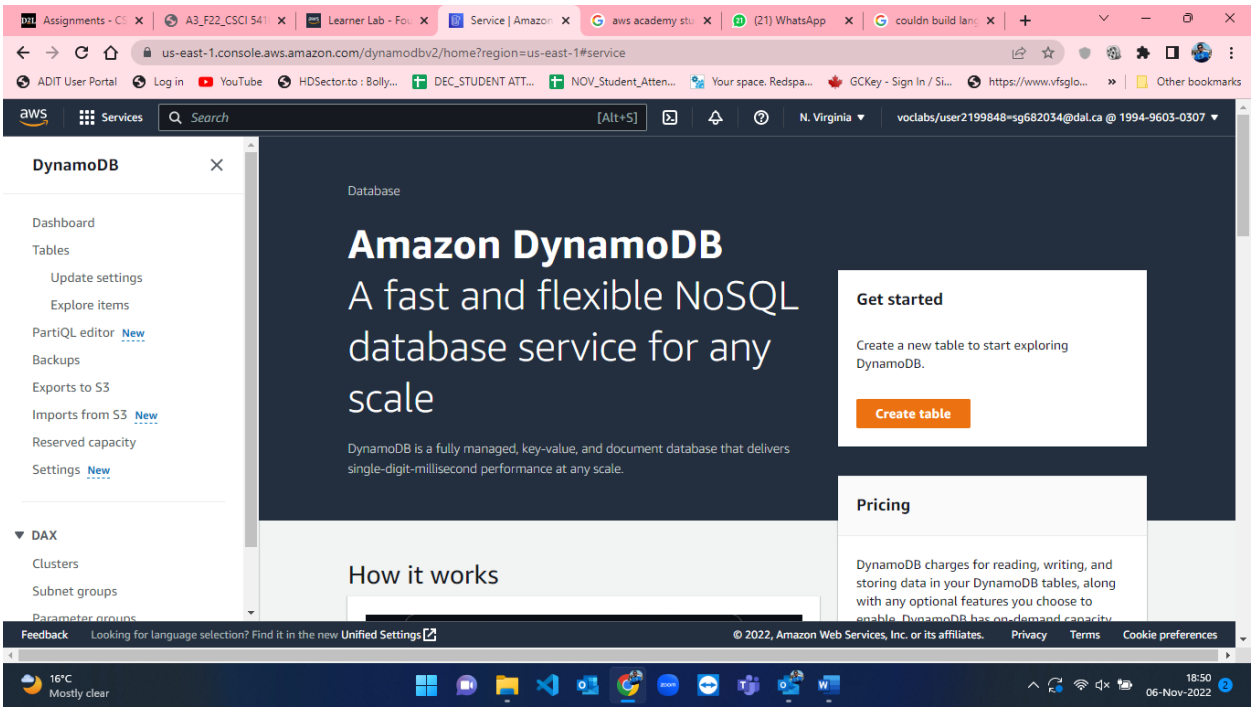

Figure 40: DynamoDB homepage in AWS.

- When we are on the homepage of DynamoDb, on the right-hand side there is an orange button for create table. I clicked on that button which took me to the create table page of DynamoDB (figure-41).

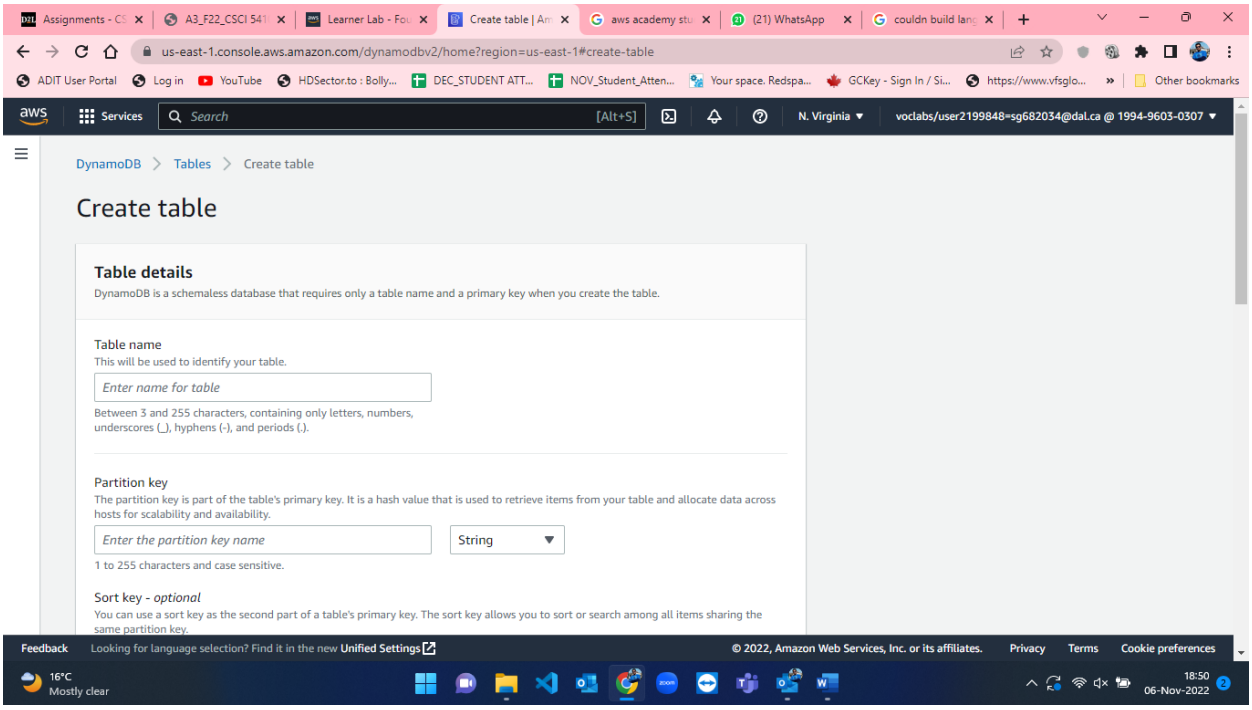

Figure 41: Default Create Table page in DynamoDB.

- I have named the table as “Student” and defined Email as the partition key for the table which represents primary key in NoSQL database (figure-42).

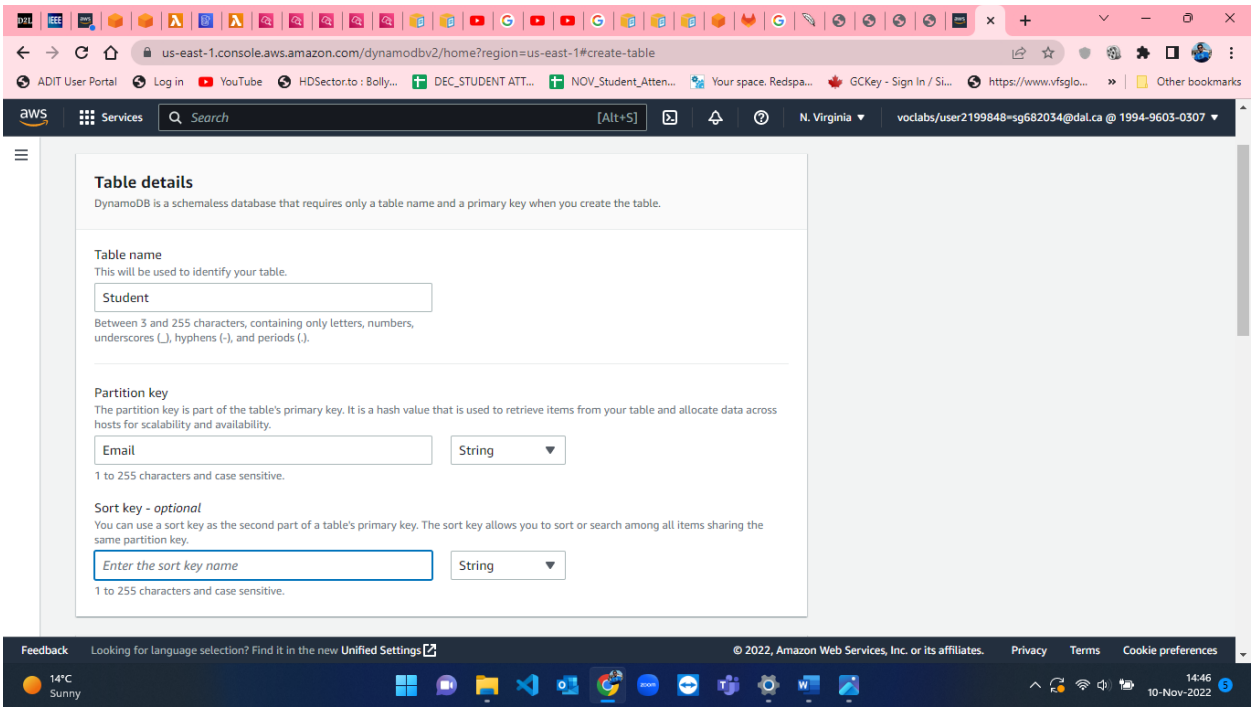

Figure 42: Create tale page in DynamoDB.

- The rest of the other options and configurations are not changed, and I finally pressed on “Create Table” button. Figure-43 shows creating table page and Figure-44 shows table successfully created page.

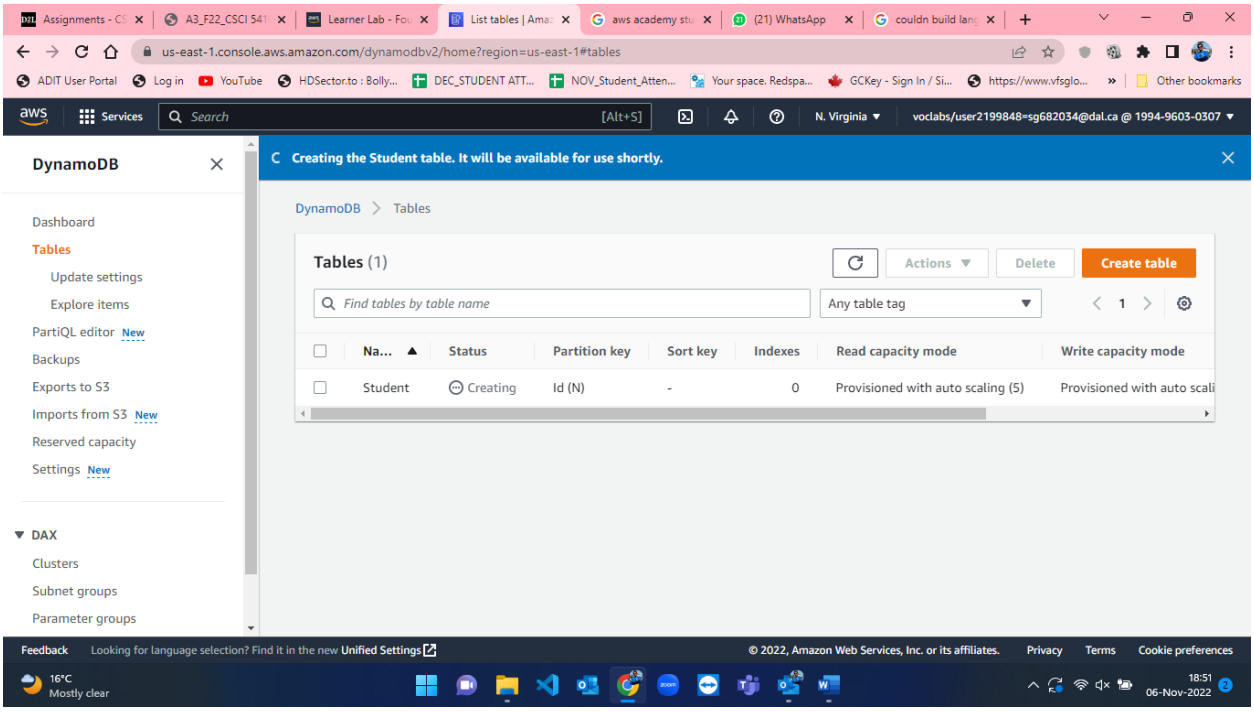

Figure 43: Creating Student table in DynamoDB.

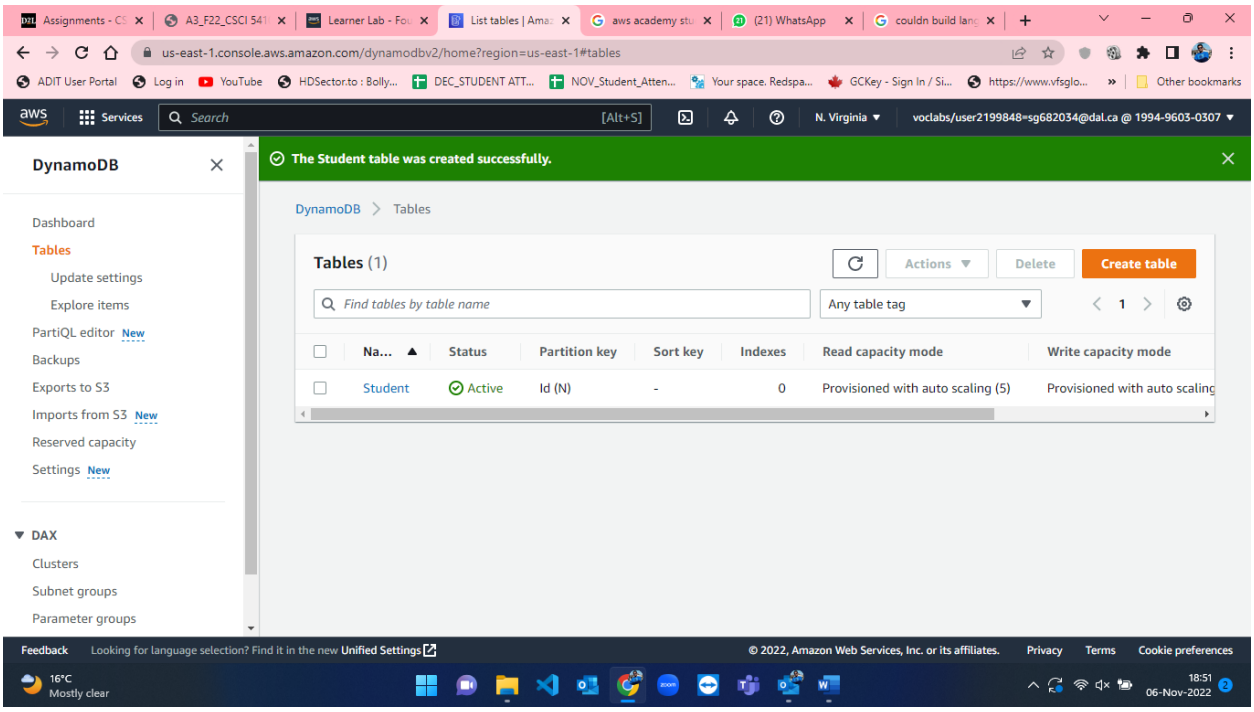

Figure 44: Student table successfully created in DynamoDB.

- After creating a table, we have to enter data in the table, so from the page of student table (Figure-45) clicking on the “Explore table items” will allow us to explore the items stored in the table. As the table is newly created it does not have any items in it which are shown in Figure-46. Therefore, clicking on “Create item” tab will allow us to enter item details in the DynamoDB database (Figure-47 to 49).

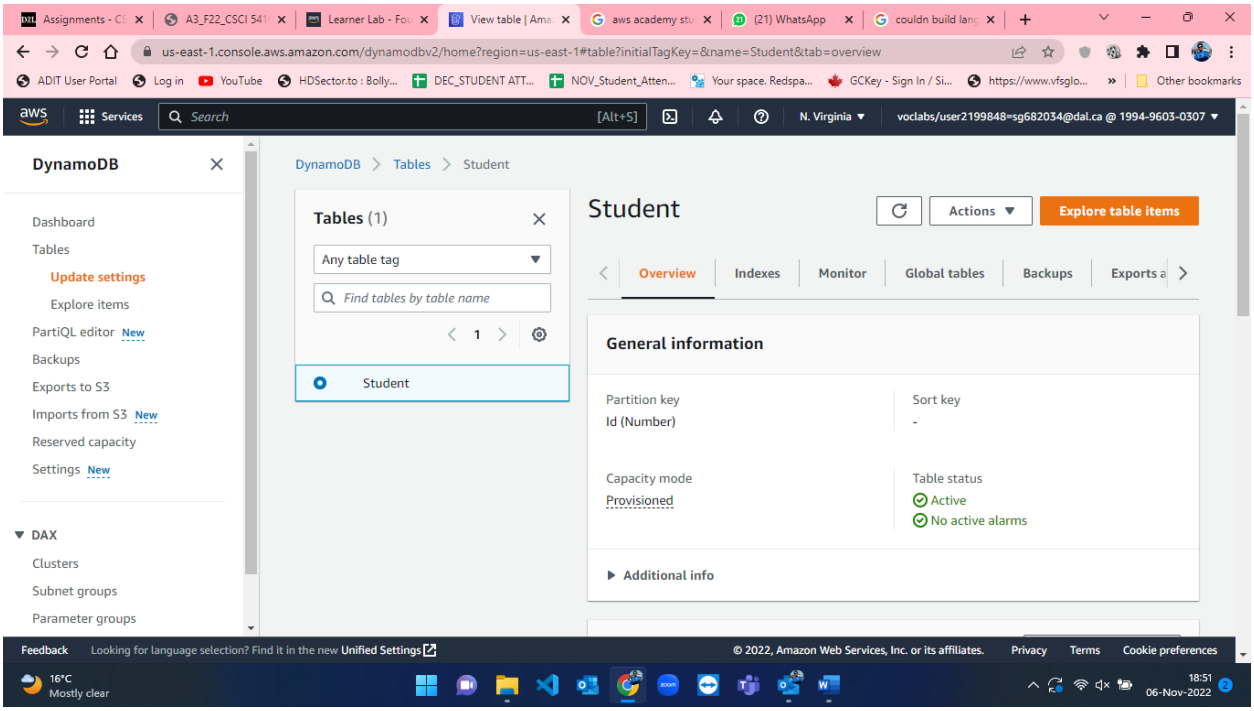

Figure 45: Student table page in DynamoDB.

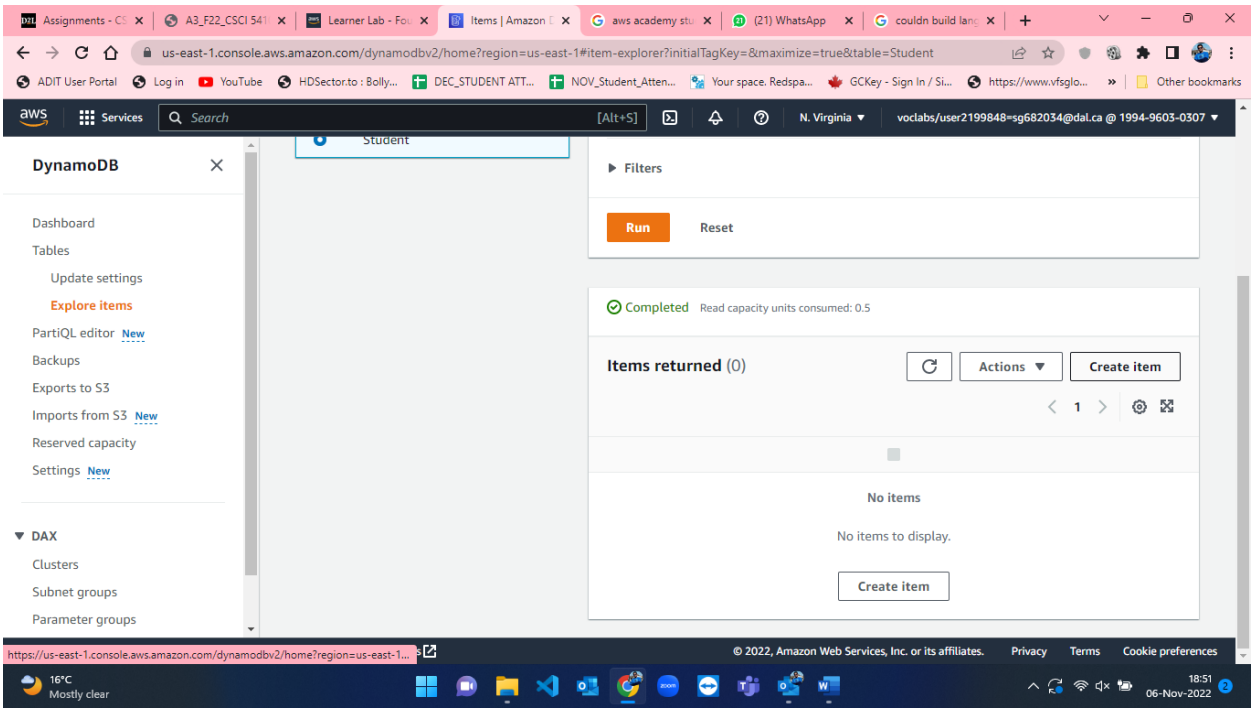

Figure 46: Create item tab having no items in table Student.

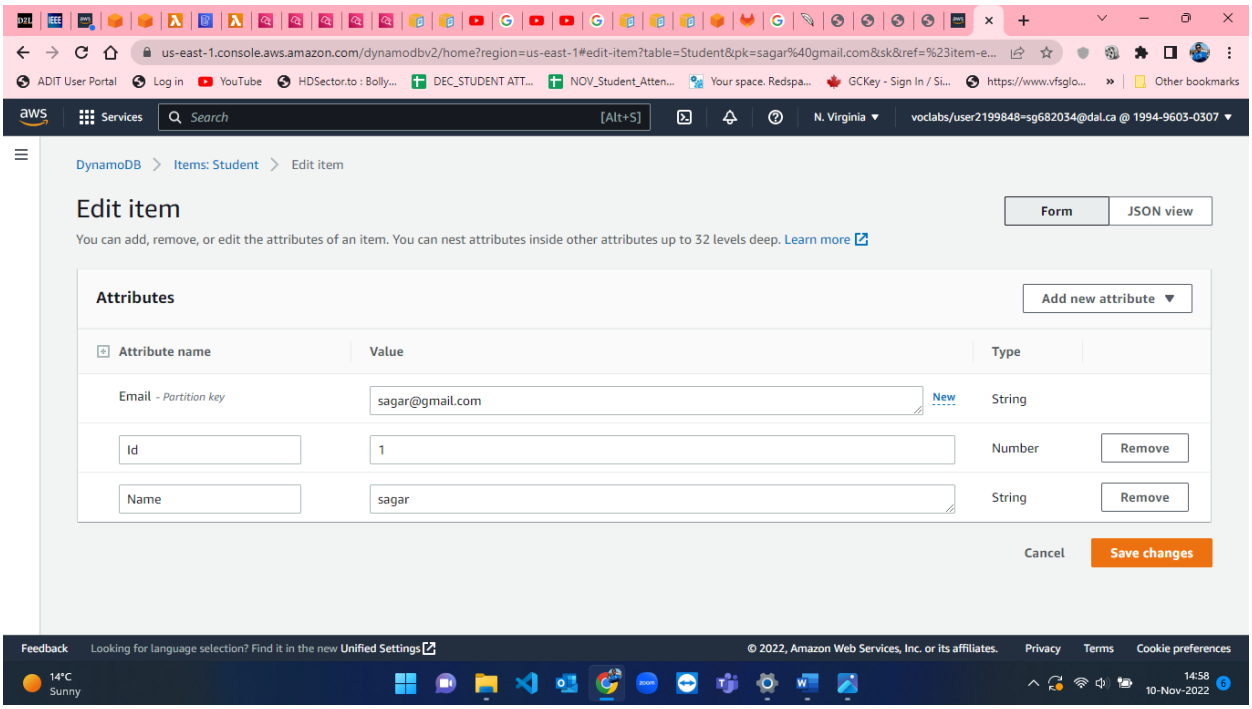

Figure 47: Creating item 1 in Student table.

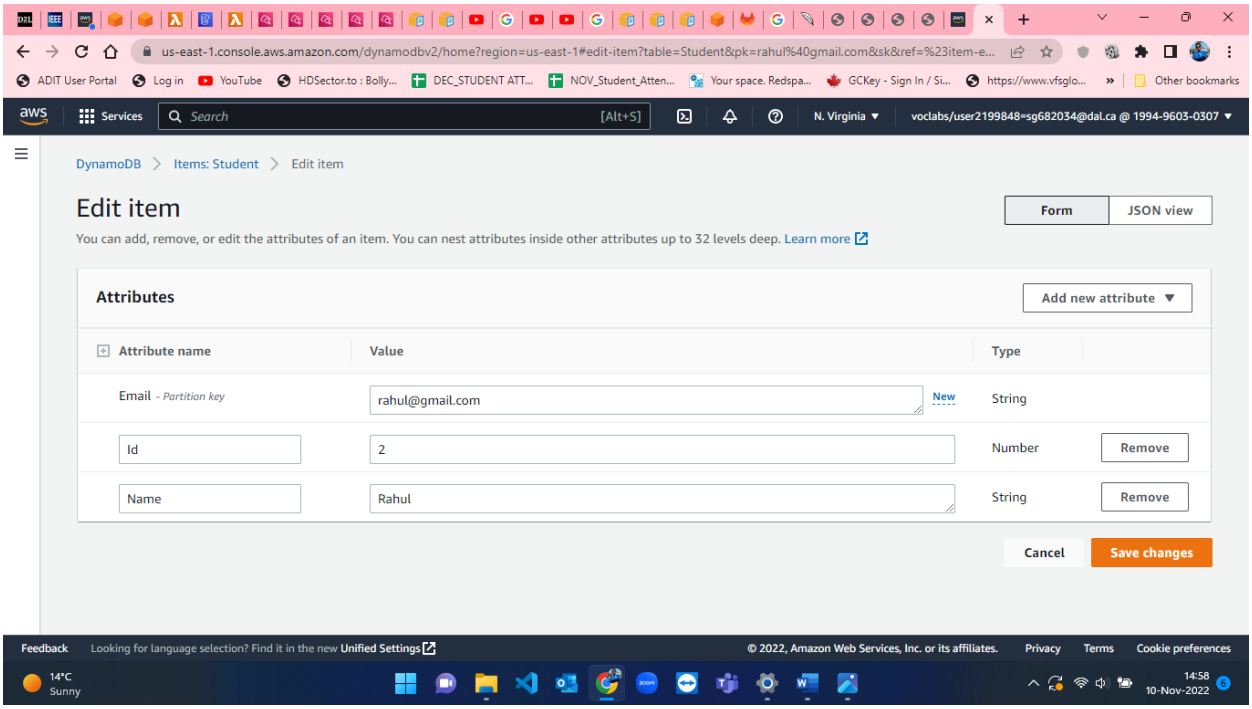

Figure 48:Creating item 2 in Student table.

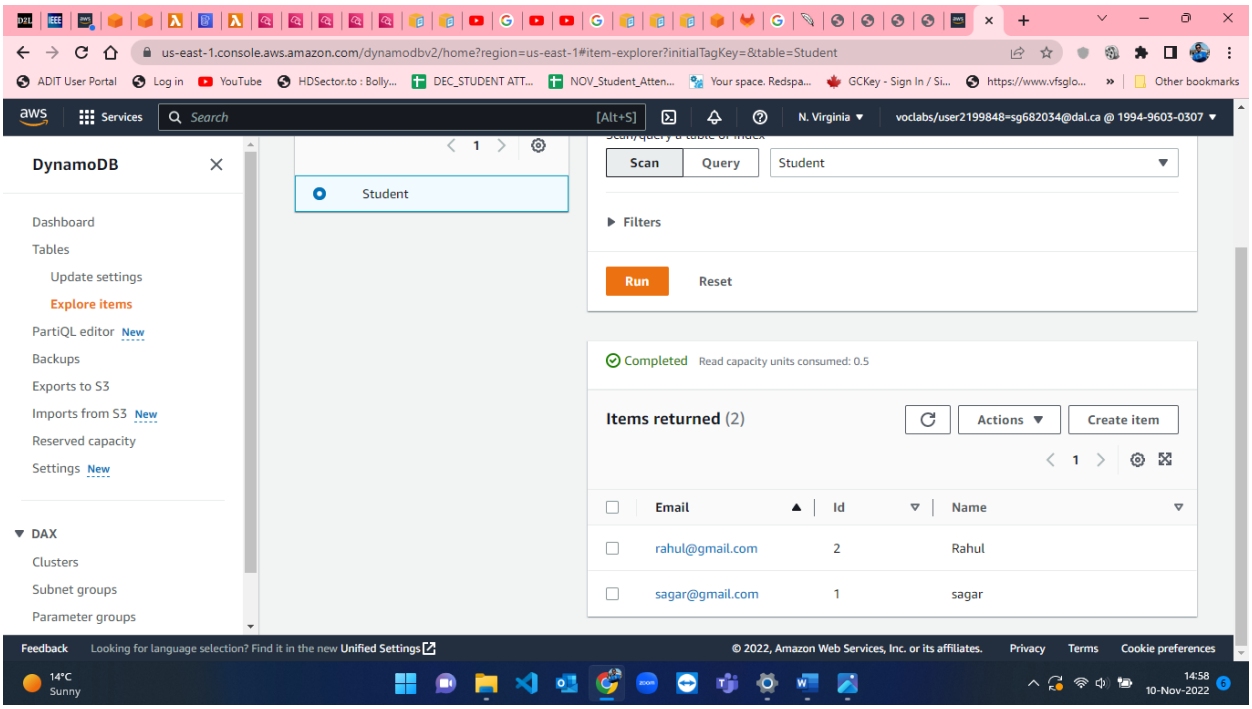

Figure 49: 2 items successfully created in Student table.

Step-3: Creating Lambda function.

When the bot and database are ready then we have to use some functionality to connect both of these. This can be achieved by Lambda function [10]. So, in this module, I am going to explain the steps for making lambda function [10].

I have searched for Lambda from the top search bar to look up for the lambda function (figure-50).

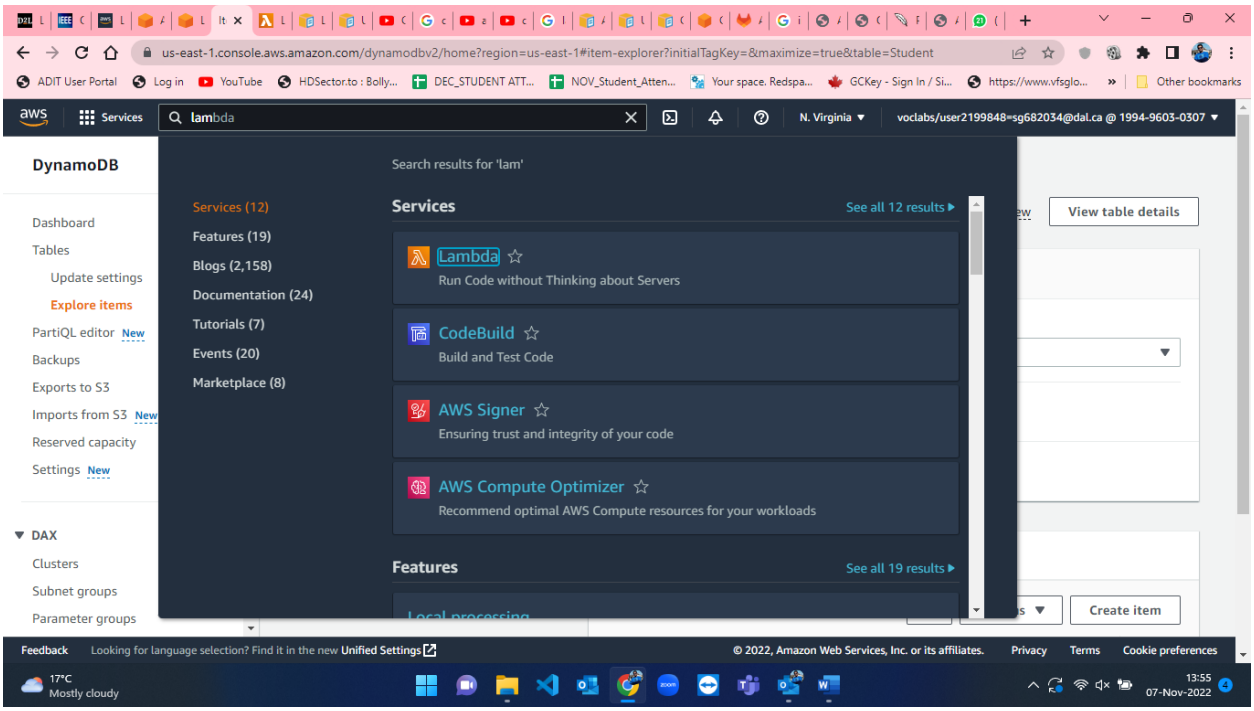

Figure 50: Searching for lambda function in AWS.

- After clicking on lambda, we have to create a lambda function. The default page of the create function is displayed in figure-51.

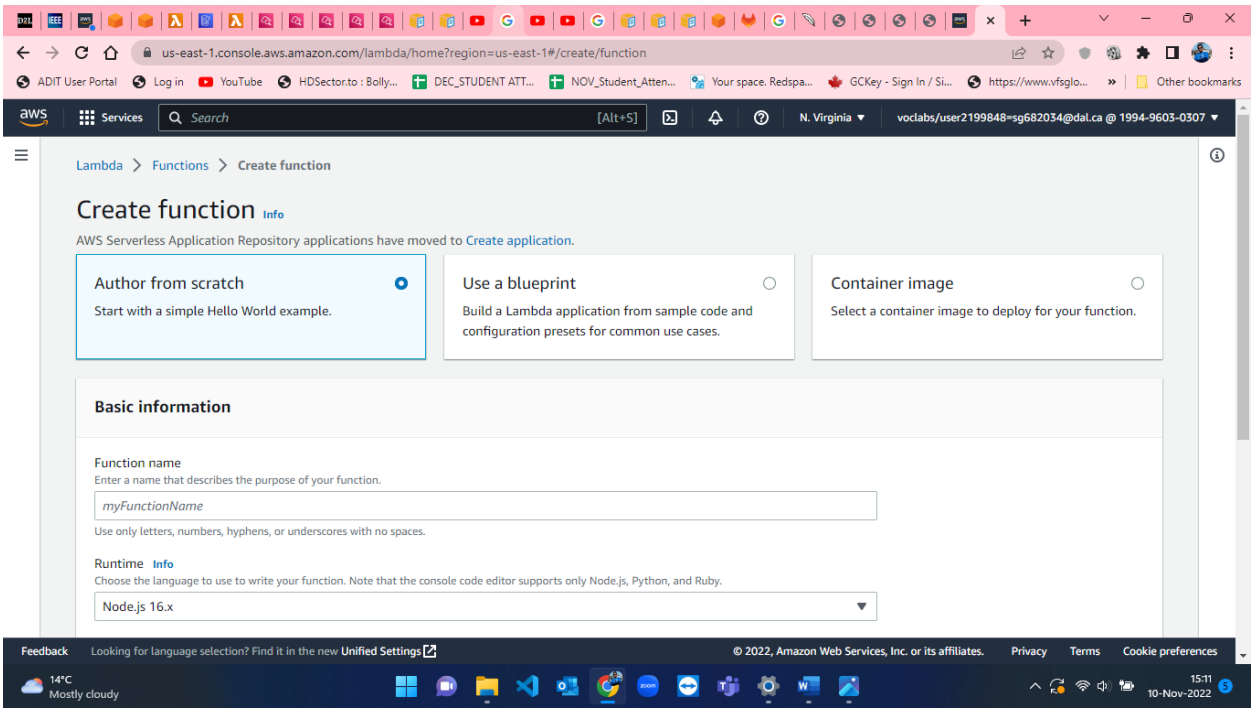

Figure 51: Create lambda function default page in AWS.

- I have named the lambda function as “StudentValidationLambda” and selected “Python” language for writing code which was by default “Node.js”. which can be seen in figure-52.

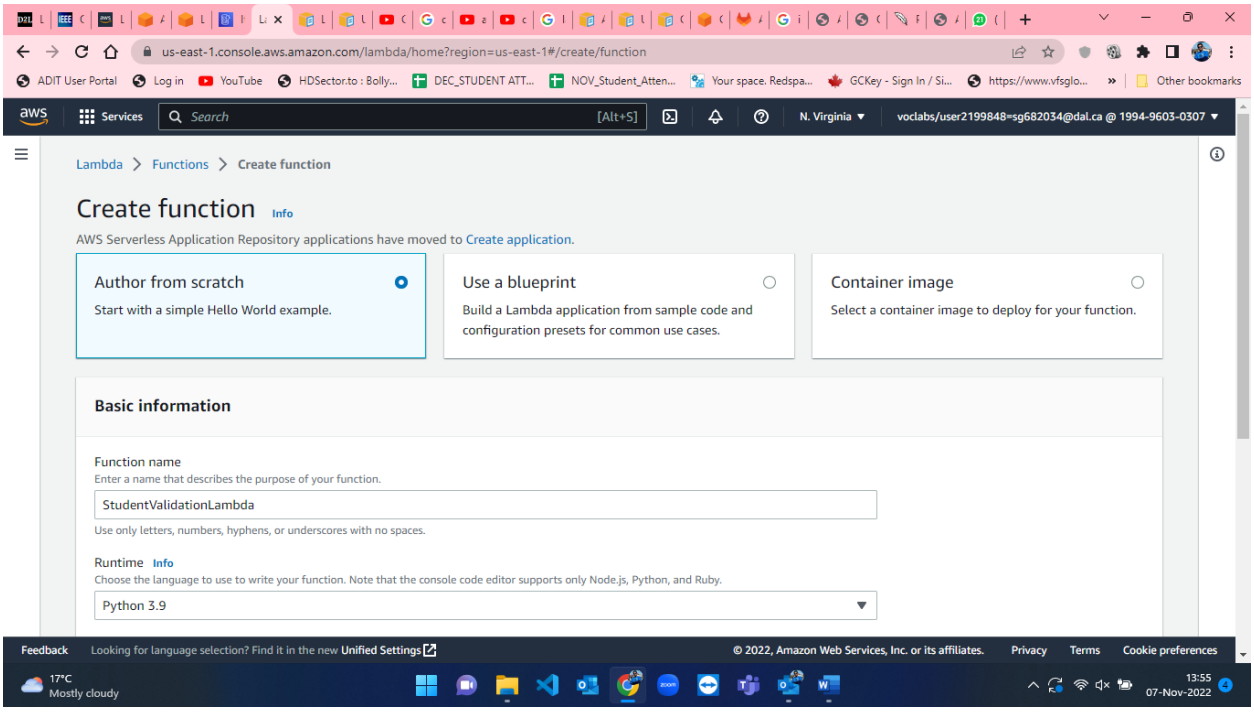

Figure 52: Lambda function creation Step-1.

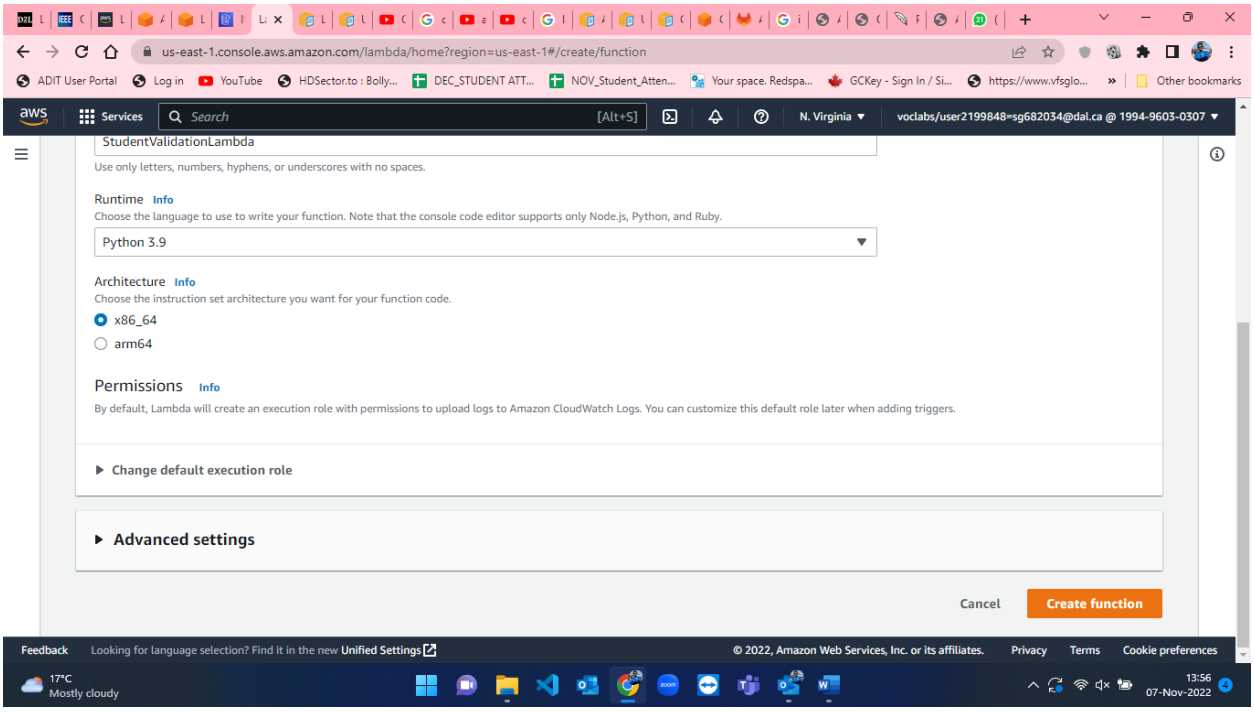

Figure 53: Lambda function creation Step-2.

- The default execution role is “Create a new role with basic Lambda permission” which I set to “Use an existing role” and in that I selected “Lab role” and it is displayed in figure-54. After selecting the lab role click on “Create Function” button so the lambda function will be created successfully (figure-56).

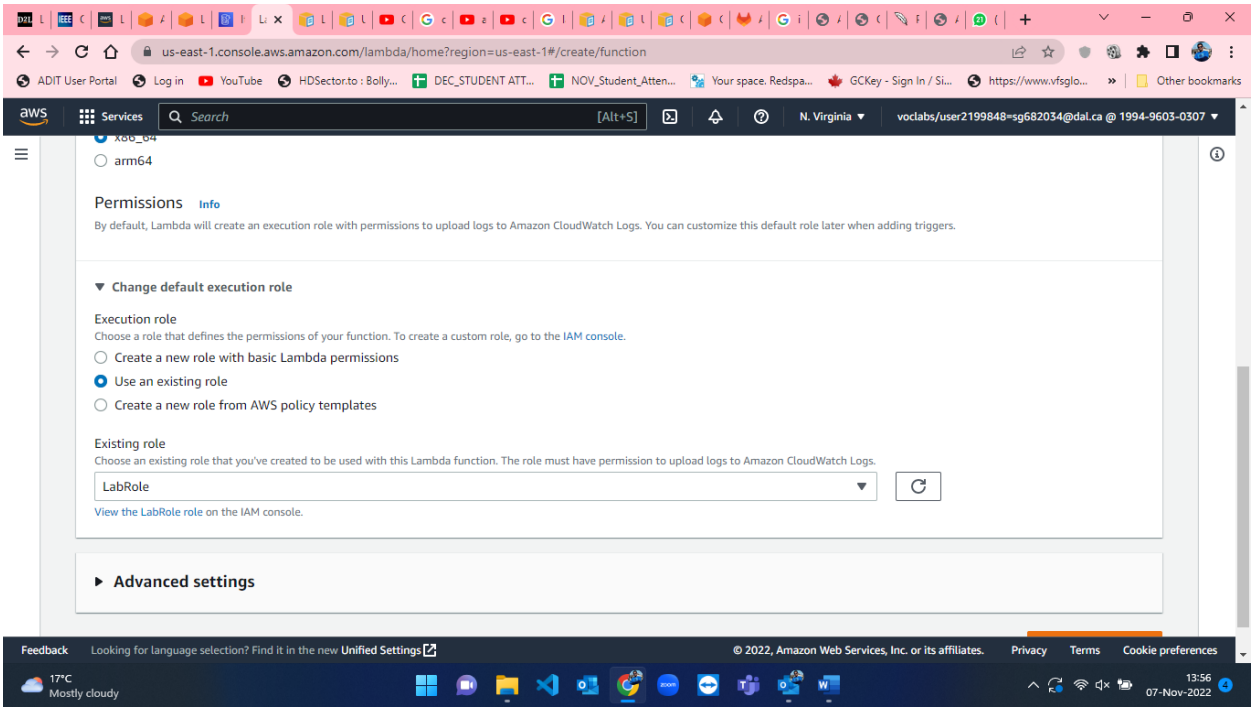

Figure 54: Lambda function creation Step-3.

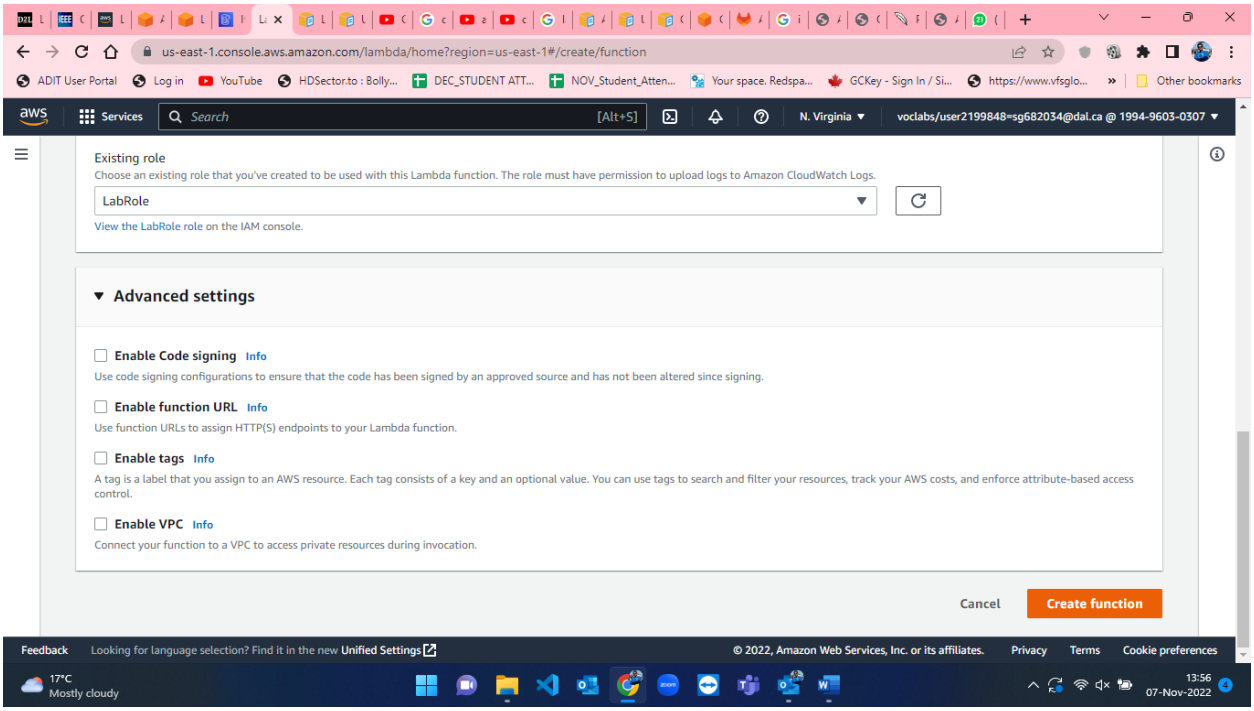

Figure 55: Lambda function creation Step-4.

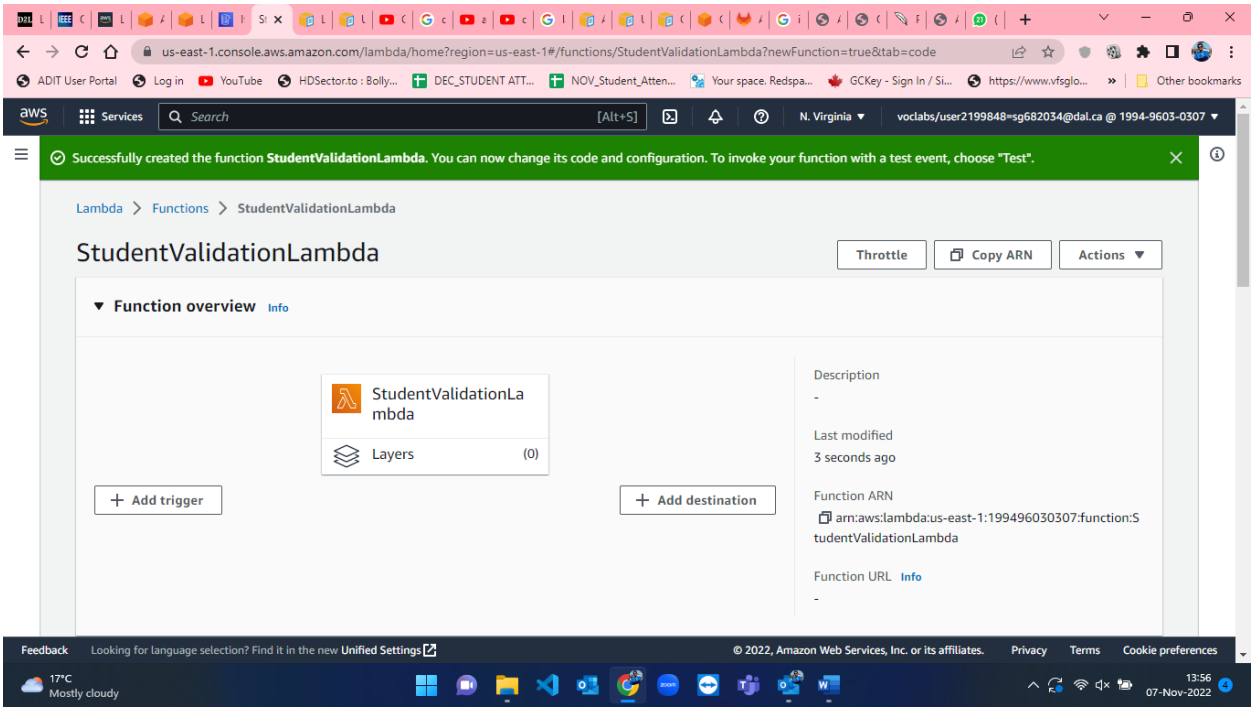

Figure 56: Successful creation of Lambda function in AWS.

- After creation of lambda function now I have to write code for lambda function to connect my lambda function with DynamoDB and Lex chatbot and also, I have to write code for validation of student details namely Email and Name [6][7]. The python script for the lambda function is attached below in figures 57,58,59 and 60.

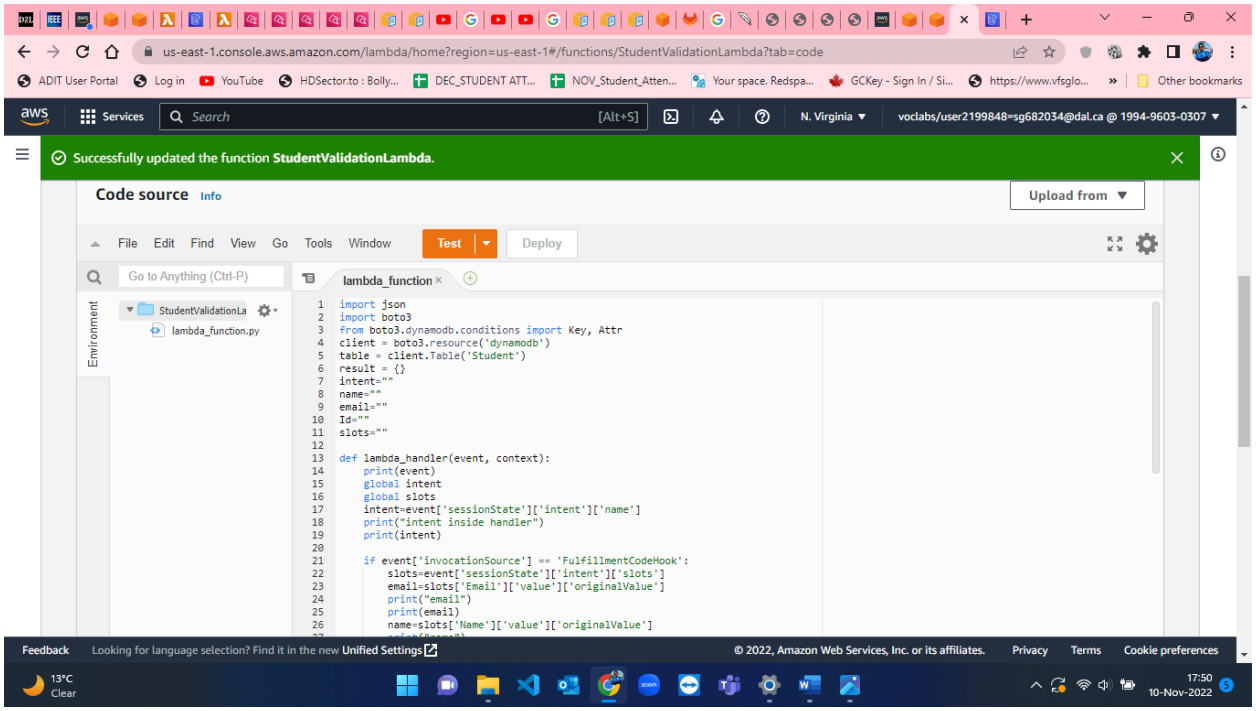

Figure 57: Lambda Function python script - 1.

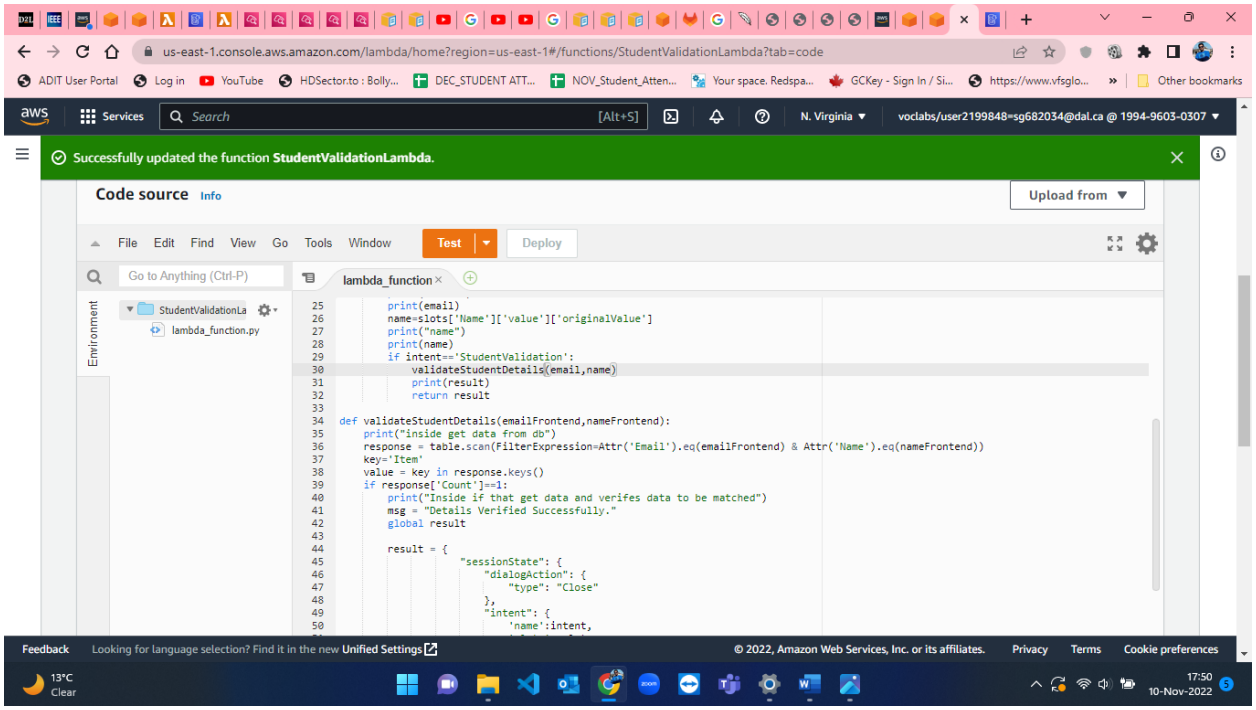

Figure 58: Lambda Function python script - 2.

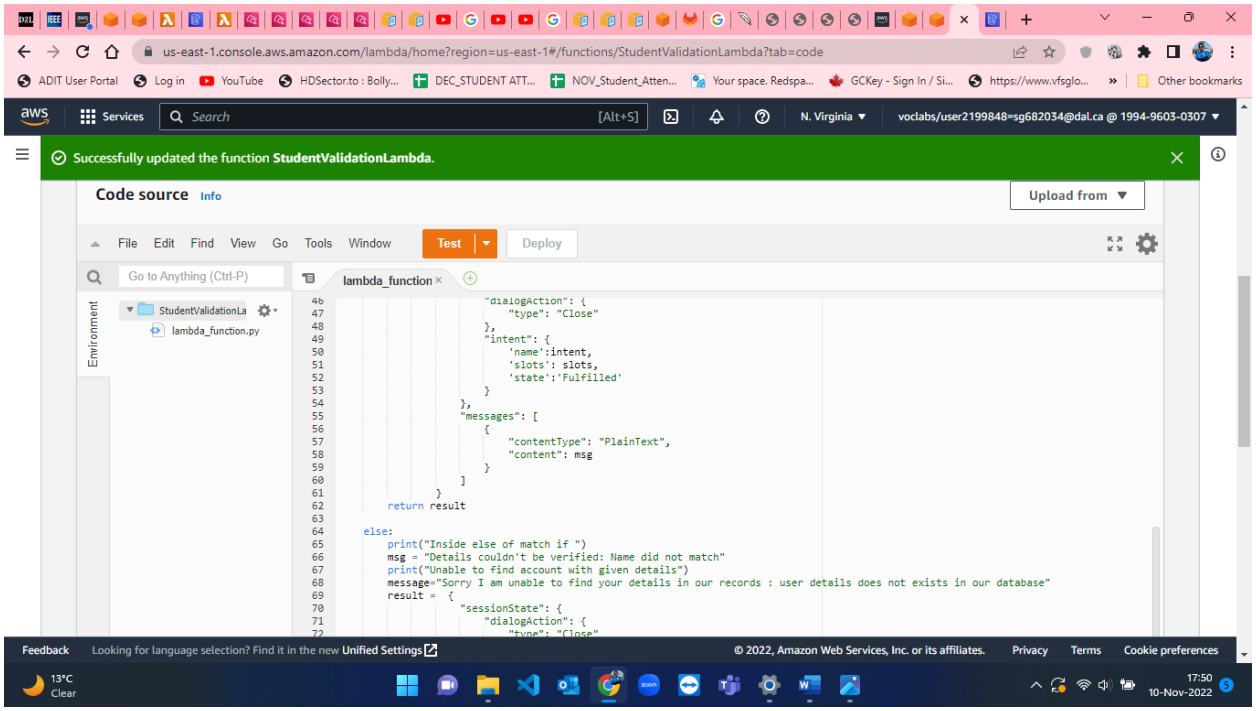

Figure 59: Lambda Function python script - 3.

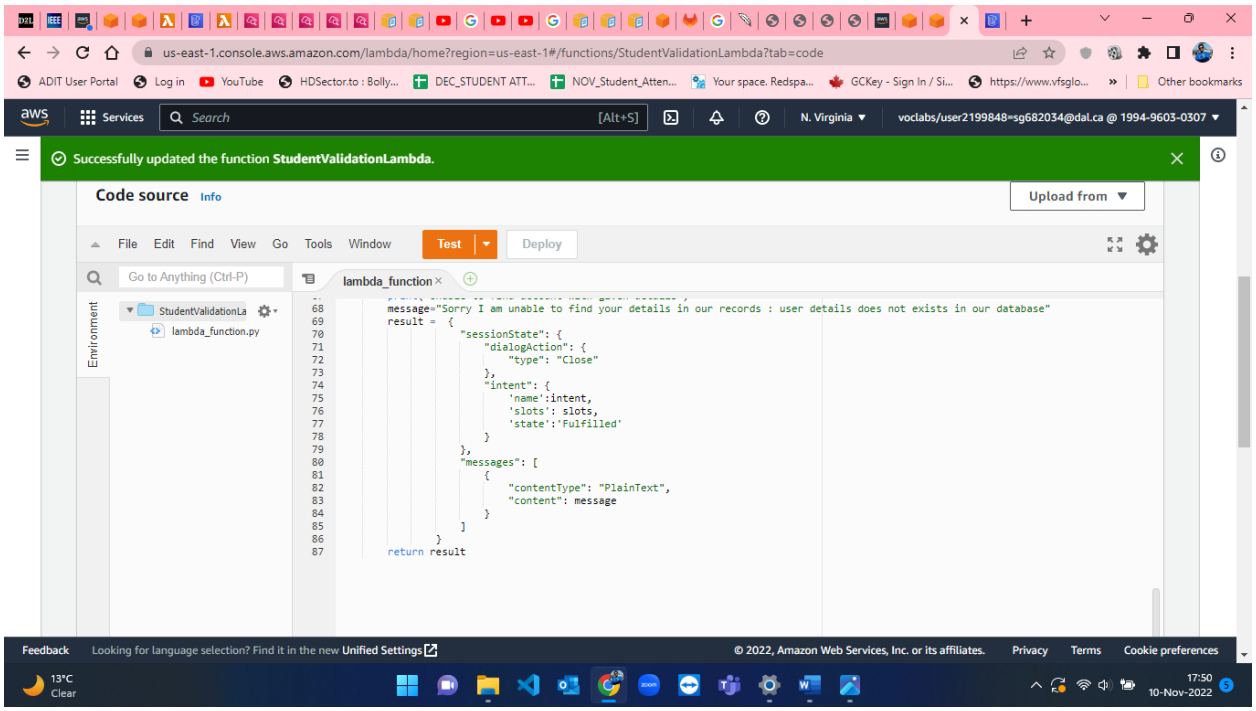

Figure 60: Lambda Function python script - 4.

Script: lambda_function.py

```
#https://github.com/venkateshkodumuri/Lex_Chatbot_to_fetch_data_from_dynamodb/blob/main/lambda_function.py
#https://stackoverflow.com/questions/36780856/complete-scan-of-dynamodb-with-boto3


import json
import boto3
from boto3.dynamodb.conditions import Key, Attr
client = boto3.resource('dynamodb')
table = client.Table('Student')
result = {}
intent=""
name=""
email=""
Id=""
slots=""

def lambda_handler(event, context):
    print(event)
    global intent
    global slots
    intent=event['sessionState']['intent']['name']
    print("intent inside handler")
    print(intent)

    if event['invocationSource'] == 'FulfillmentCodeHook':
        slots=event['sessionState']['intent']['slots']
        email=slots['Email']['value']['originalValue']
        print("email")
        print(email)
        name=slots['Name']['value']['originalValue']
        print("name")
        print(name)
        if intent=='StudentValidation':
            validateStudentDetails(email,name)
            print(result)
            return result

def validateStudentDetails(emailFrontend,nameFrontend):
    print("inside get data from db")
    response = table.scan(FilterExpression=Attr('Email').eq(emailFrontend) &
Attr('Name').eq(nameFrontend))
    key='Item'
```

```
    value = key in response.keys()
    if response['Count']==1:
        print("Inside if that get data and verifes data to be matched")
        msg = "Details Verified Successfully."
        global result

        result = {
                    "sessionState": {
                        "dialogAction": {
                            "type": "Close"
                        },
                        "intent": {
                            'name':intent,
                            'slots': slots,
                            'state':'Fulfilled'
                        }
                    },
                    "messages": [
                        {
                            "contentType": "PlainText",
                            "content": msg
                        }
                    ]
                }
        return result

    else:
        print("Inside else of match if ")
        msg = "Details couldn't be verified: Name did not match"
        print("Unable to find account with given details")
        message="Sorry I am unable to find your details in our records : user
details does not exists in our database"
        result =  {
                    "sessionState": {
                        "dialogAction": {
                            "type": "Close"
                        },
                        "intent": {
                            'name':intent,
                            'slots': slots,
                            'state':'Fulfilled'
                        }
                    },
                    "messages": [
                        {
```

```
                    "contentType": "PlainText",
                    "content": message
                }
            ]
        }
    return result
```

Step-4: Deploying Lambda function to AWS Lex.

- When the python script is done, I have to deploy that code in Lex and DynamoDB. The “Deploy” button can been seen beside “Test” button in figure-60. After deploying, we have to build AWS Lex chatbot and then we have to test it [5]. The screenshots from 61 to 66 successful testing of AWS Lex with DynamoDB using Lambda function.

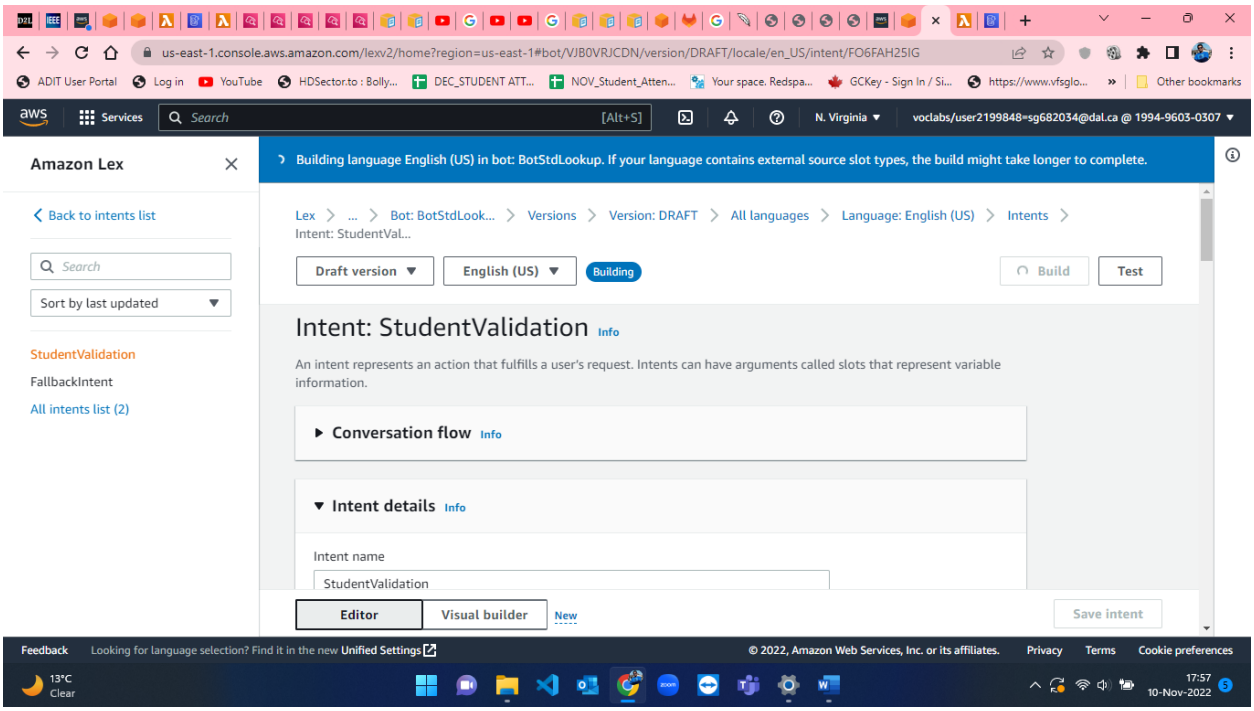

Figure 61: Building the intent for the BotStdLookup in AWS Lex.

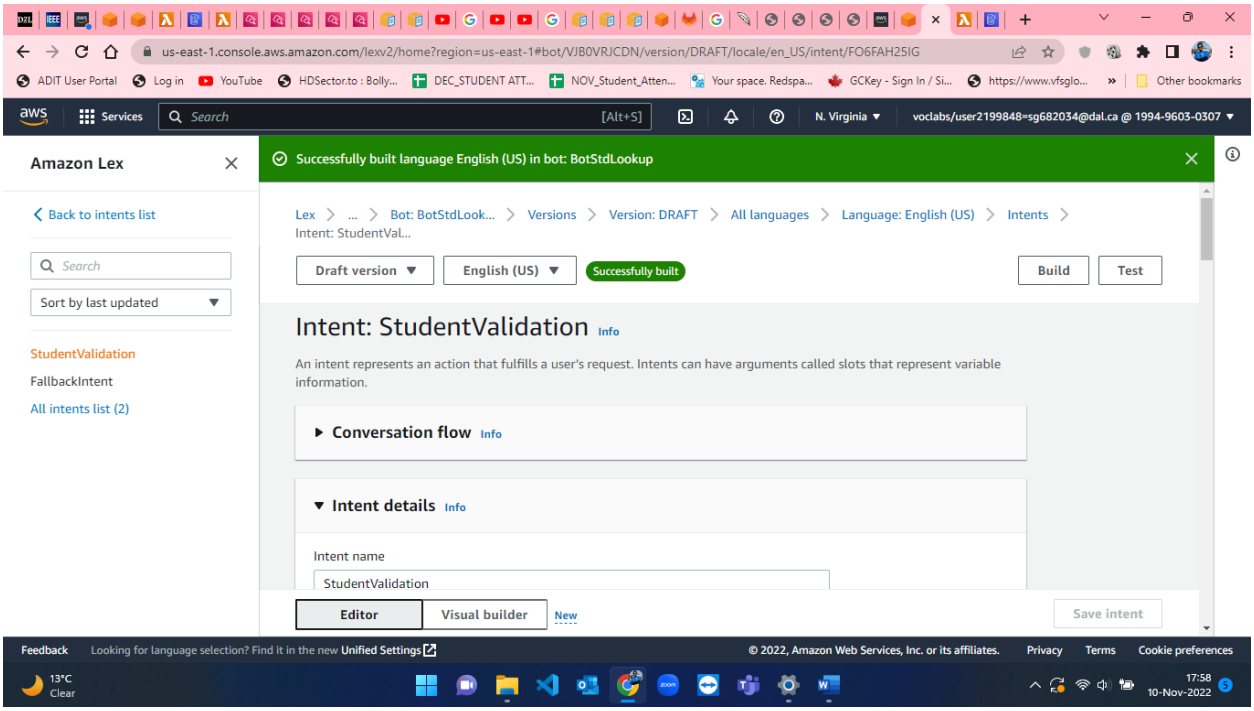

Figure 62: BotStdLookup Built successful in AWS Lex

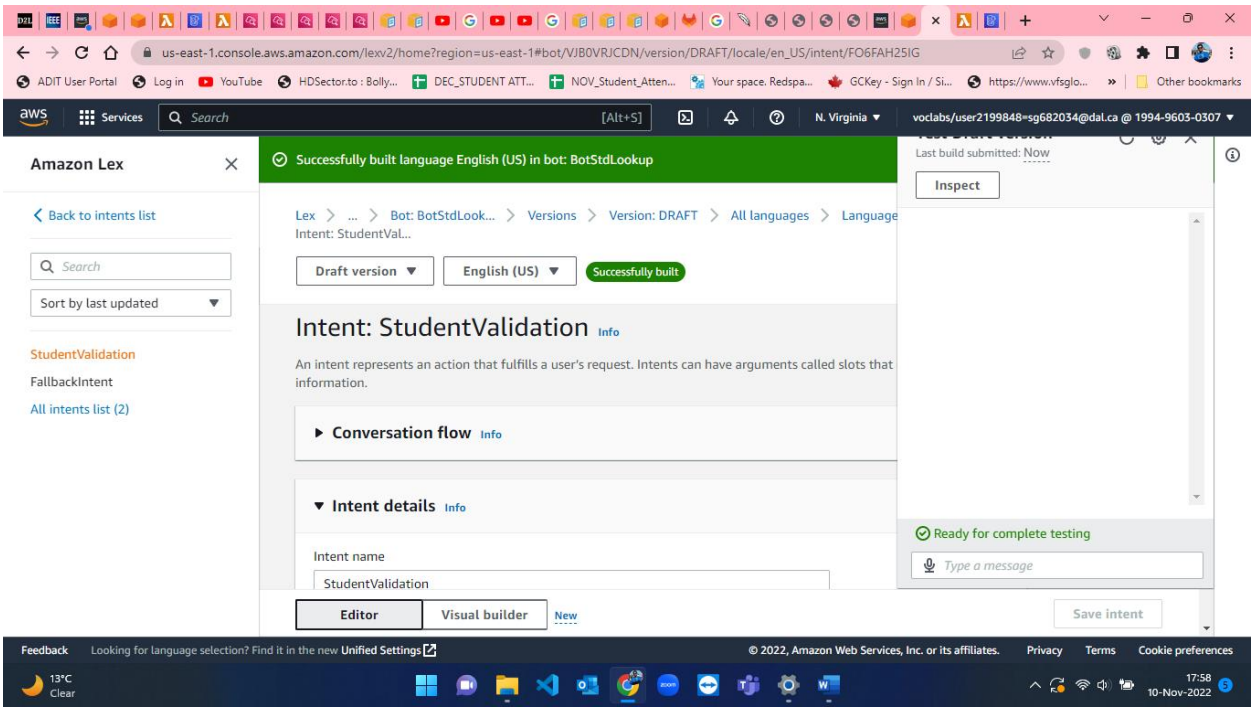

Figure 63: BotStdLookup Chatbot Screen-1.

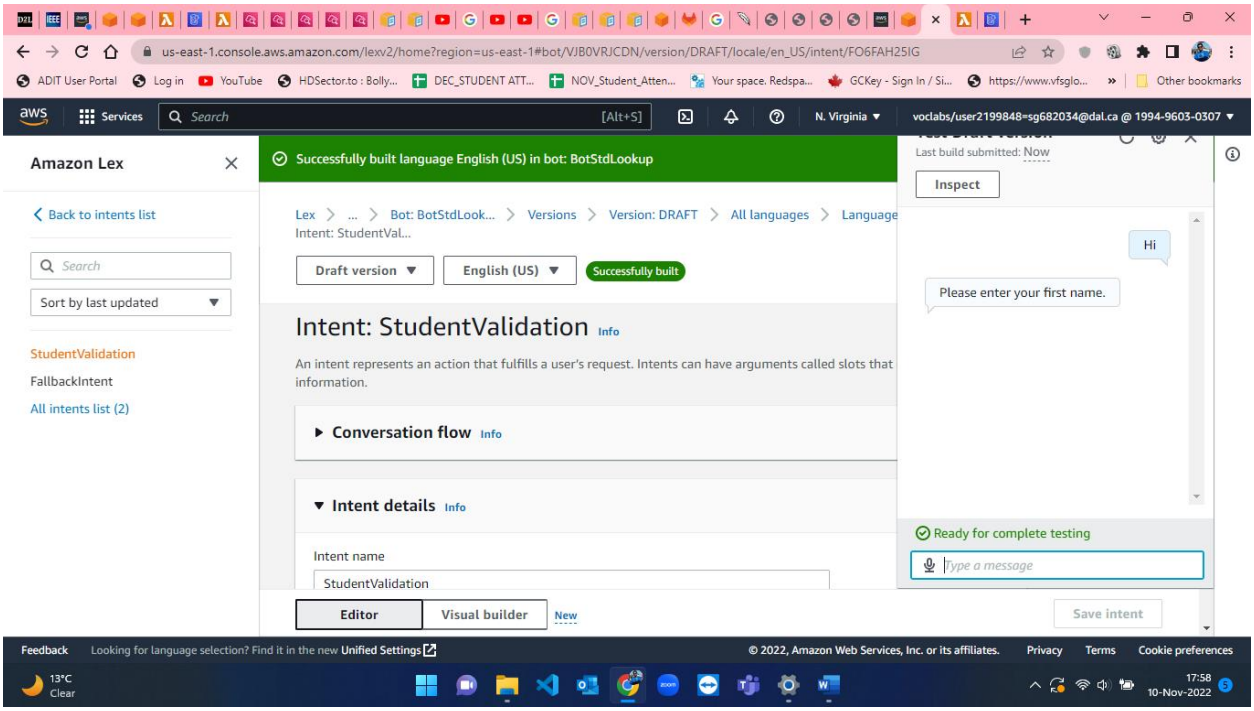

Figure 64: BotStdLookup Chatbot Screen-2.

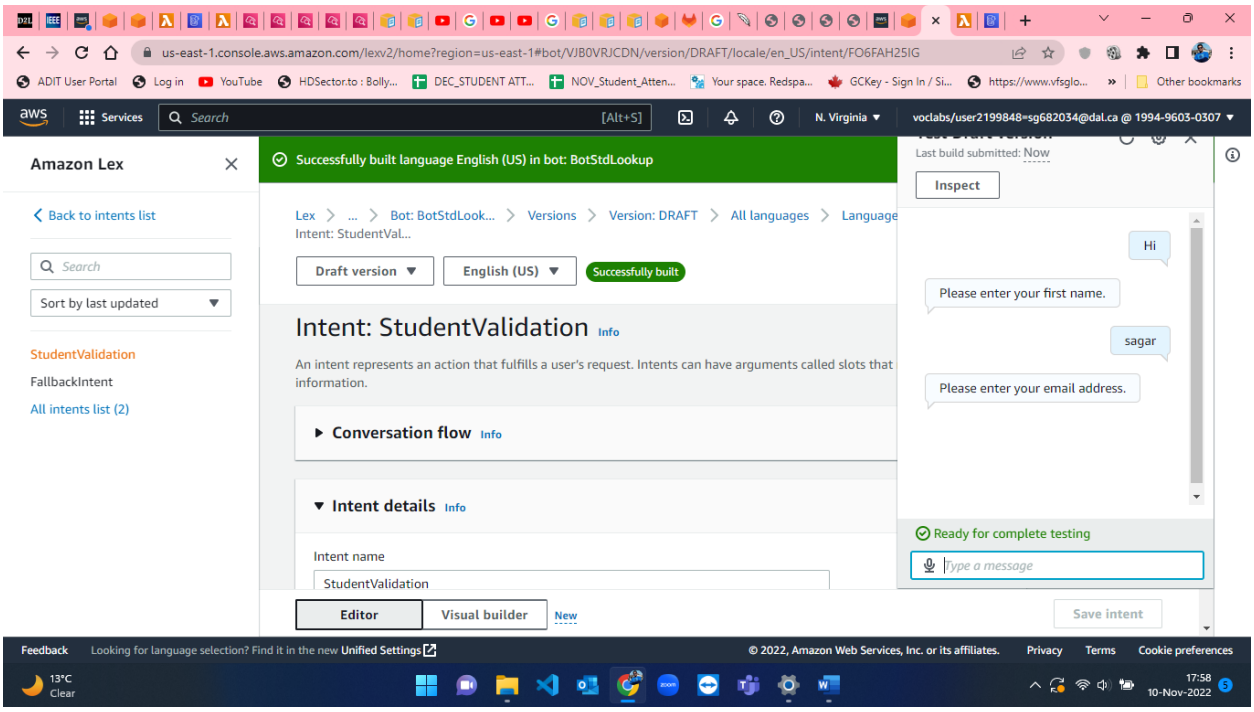

Figure 65: BotStdLookup Chatbot Screen-3.

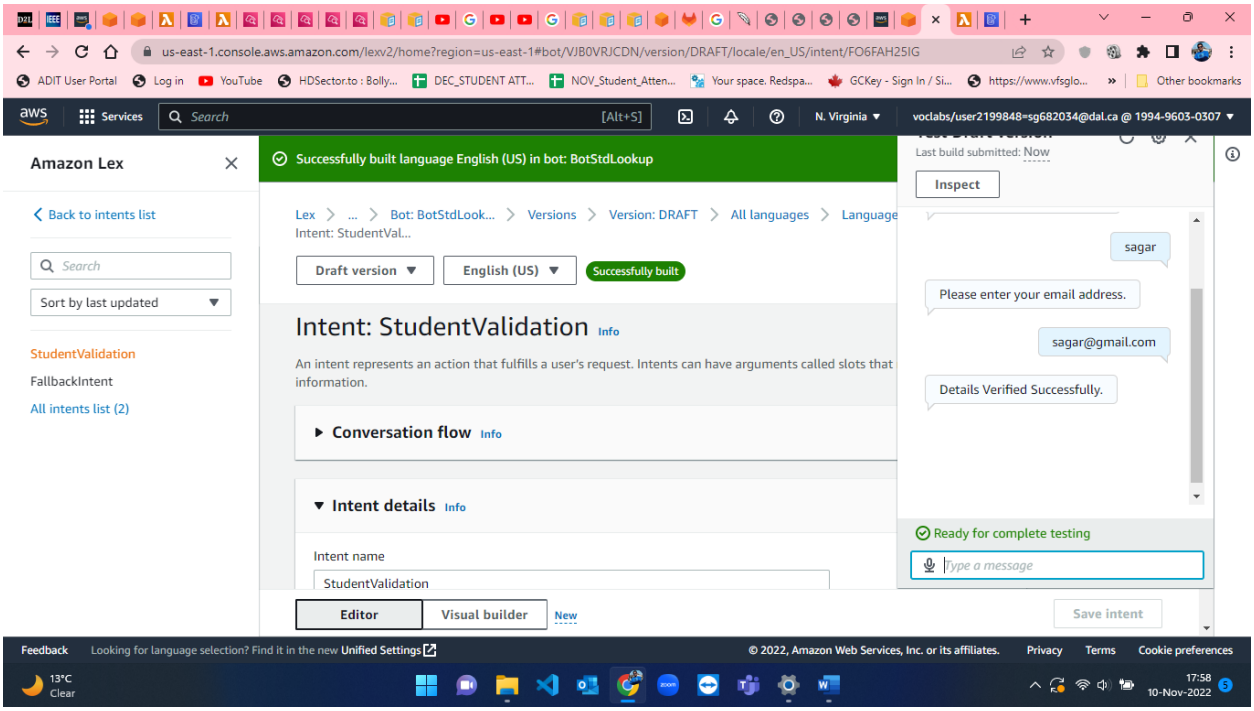

Figure 66: BotStdLookup Chatbot Screen-4.

Test Cases for student details verification chatbot (BotStdLookup) :

- Test case 1 - Entering correct first name and email address.

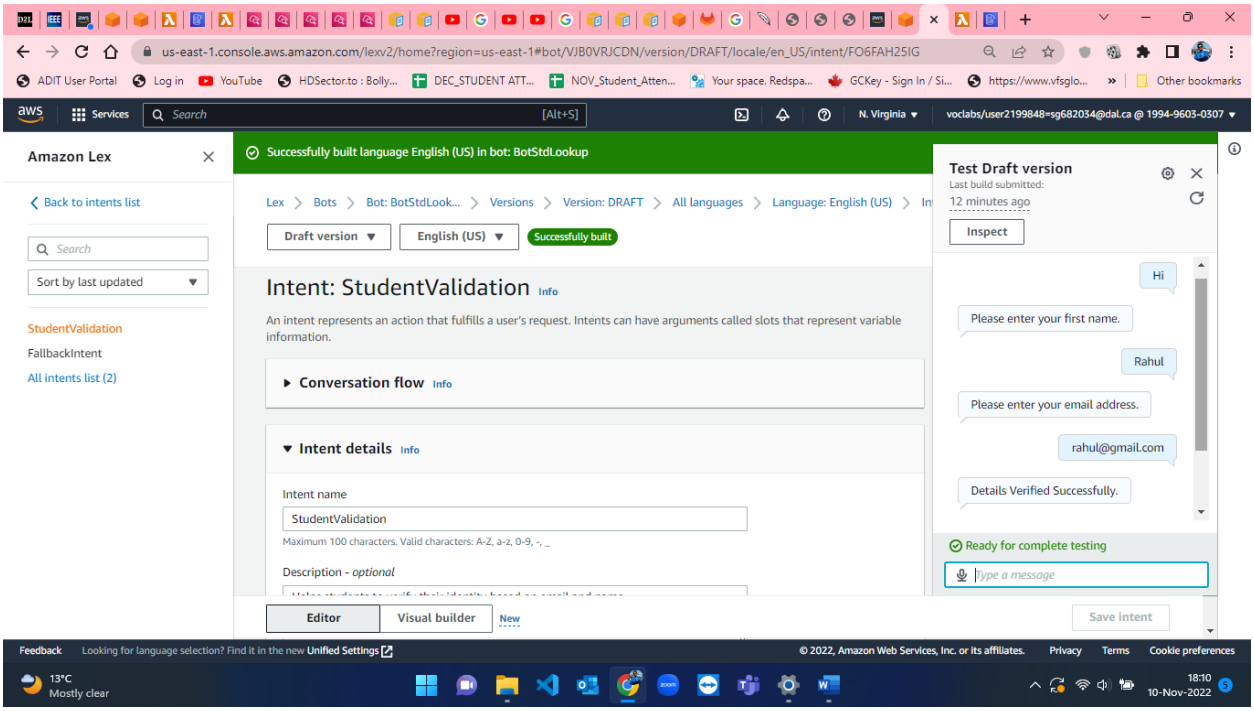

Figure 67: Test case for entering correct name and email address.

- Test case 2 - Entering correct first name and wrong email address.

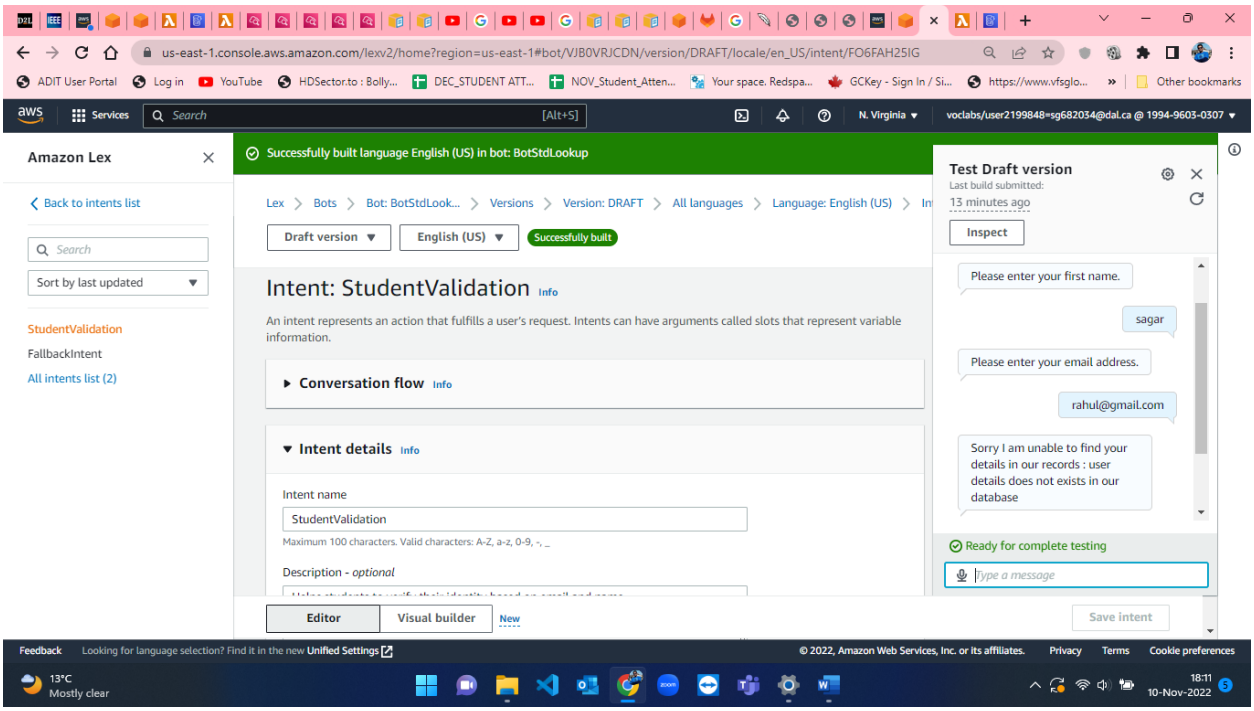

Figure 68: Test case for entering correct name and wrong email address.

- Test case 3 - Entering wrong first name and correct email address.

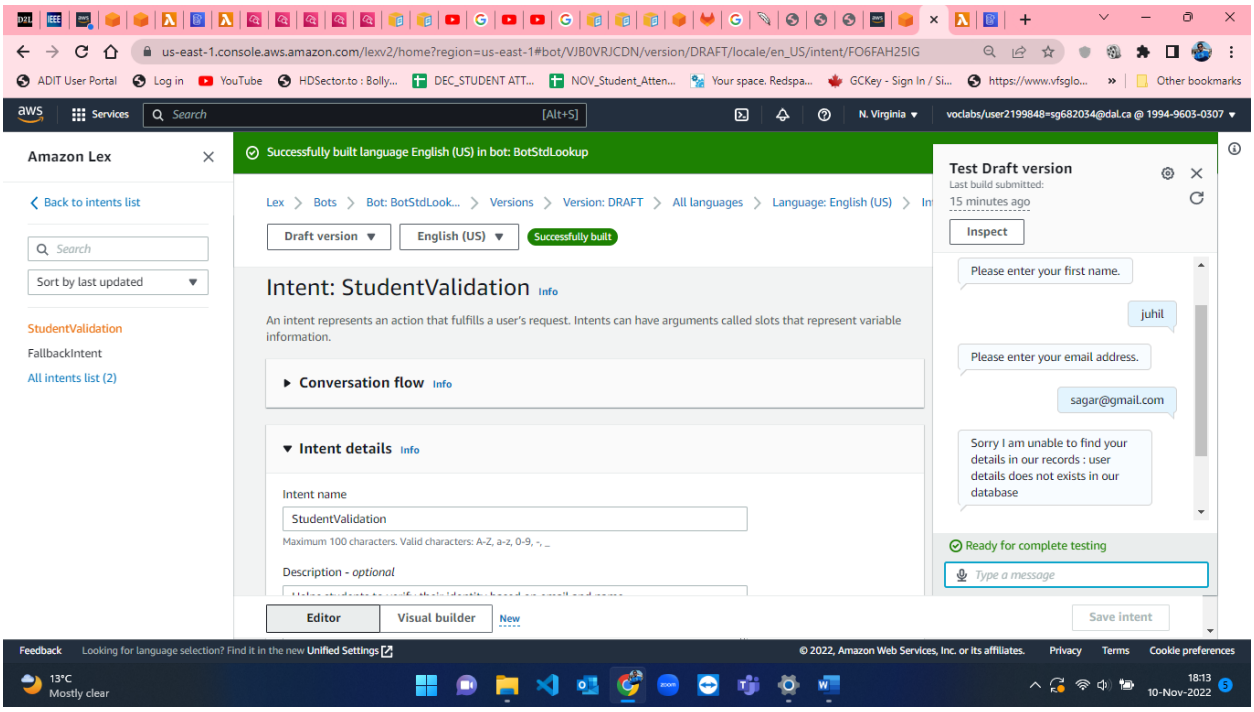

Figure 69: Test case for entering wrong name and correct email address.

- Test case 4 - Entering wrong first name and wrong email address.

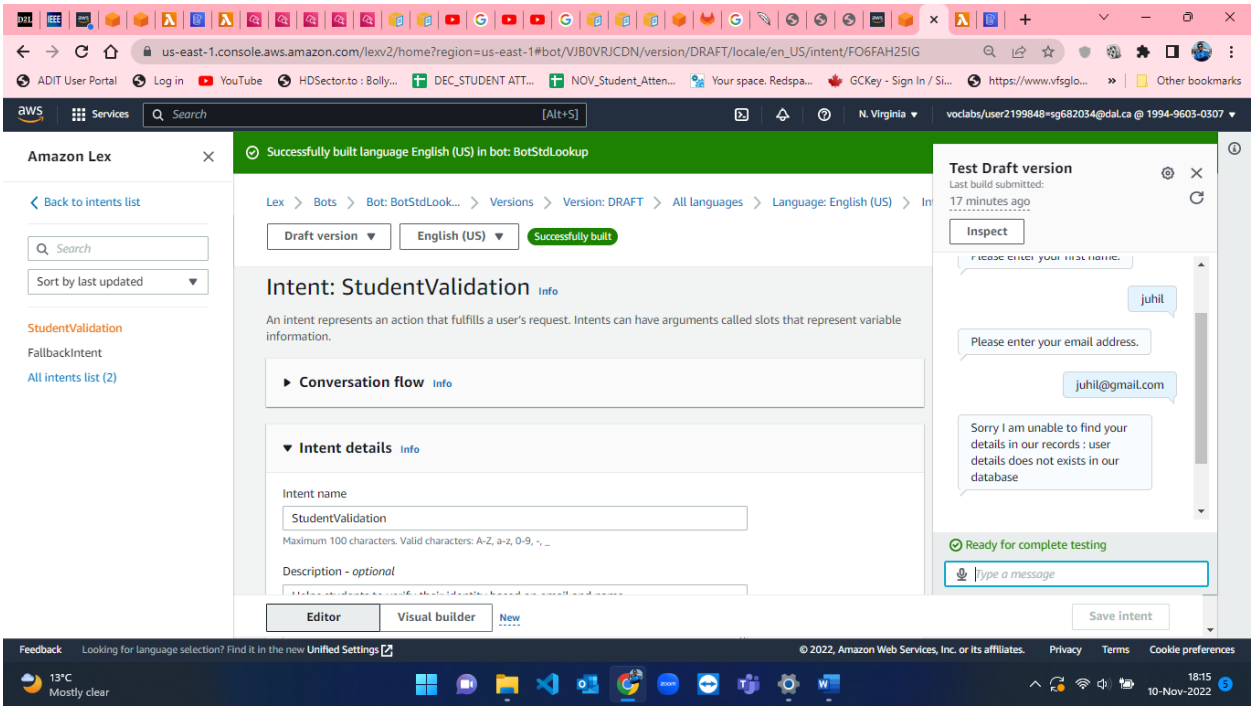

Figure 70: Test case for entering wrong name and wrong email address.

References :

[1] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/gs2-create-bot-create.html. [Accessed: 08-Nov-2022].

[2] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/gs2-create-bot-intent.html. [Accessed: 08-Nov-2022].

[3] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/gs2-create-bot-slot-types.html. [Accessed: 08-Nov-2022].

[4] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/gs2-create-bot-configure-intent.html. [Accessed: 08-Nov-2022].

[5] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/gs2-build-and-test.html. [Accessed: 08-Nov-2022].

[6] “Complete scan of dynamoDb with boto3,” Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/36780856/complete-scan-of-dynamodb-with-boto3. [Accessed: 08-Nov-2022].

[7] lambda_function.py at main · venkateshkodumuri/Lex_Chatbot_to_fetch_data_from_dynamodb. .

[8] Amazon.com. [Online]. Available: https://us-east-1.console.aws.amazon.com/lexv2/home?region=us-east-1#bots. [Accessed: 08-Nov-2022].

[9] Amazon.com. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SampleData.CreateTables.html. [Accessed: 08-Nov-2022].

[10] Amazon.com. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. [Accessed: 08-Nov-2022].

PART-B Reading Task

Quantifying Permissiveness of Access Control Policies

The secrecy of private information stored in computer clouds is becoming an increasingly pressing issue because of the pervasive use of software applications. There are access control specification languages and libraries provide solutions for preserving the secrecy of information which can help developers to write complex policy specifications without verification and validation but still this can lead to disastrous consequences[1]. In the given paper, the authors have presented a quantitative and differential policy analysis framework which not only identifies which policy is more permissive but also quantifies the relative permissiveness of different policies.

According to authors, they have analyzed that there were many situations where the hacker or the user get unauthorized access due to loopholes in the access control policies which lead to compromission of valuable user data. A similar breach happened in Capital One[1] which uses AWS for cloud computing. The server run by Capital One was breached by an outside attacker who gained unauthenticated access to an AWS IAM role and accessed all the data of S3 bucket[1]. So, in this case the attached IAM role is compromised by a malicious user which exposed a great deal of data. The authors here want to quantify the permissiveness between all policies whereas existing policy analysis techniques can only verify if a policy is more or less permissive than others. For this, they have made a policy model where they defined similar functions of access control model. They have defined some equations and sets to analyze the permissiveness of the access control policy. Researchers have modified the older translation of policy to SMT formula for precise reasoning about determining the permissiveness of a single policy or the relative permissiveness of two policies.

Authors have also provided the constraint transformation which enables us to compactly encode constraints on policy actions extracted from access control policies. They have conducted 4 experiments to evaluate the proposed approach and its implementation in QUACKY[1]. The first experiment evaluates QUACKY's performance and determines which factors influence the analysis[1]. The second experiment is how effectively QUACK is in reasoning of relative permissiveness of access control policies[1]. The third experiment compares the performance of QUACKY with other models and approaches based on SMT solvers[1]. Finally, the fourth experiment depicts that the approach proposed by authors can be implemented to Azure policies[1]. Obtaining the dataset for the experiment is tough as no one can share their access control policies publicly. So, they used two policies collected from the users and assumed that these are real-world policies. They have performed all these four experiments on AWS policies and Microsoft Azure policies by analyzing each policy twice, once without type constraints and once with type constraints. They have found that there is a tradeoff between time and permissiveness because without type constraints more requests are allowed and become more permissive compared to with type constraints.

Errors in the access control policies or misconfiguration while defining policies can have very bad and harsh consequences. In the given paper, the authors have presented the novel approach for modelling and quantifying permissiveness of access control policies. The proposed approach depends upon model counting constraint solvers such as SMT or SAT to assess the permissiveness of the access policy. It is implemented on AWS policies and Microsoft Azure policies which resulted in the quantitative approach proposed by the authors can be applied to real world practice[1]. The major limitation here we can see is that author have taken only specific policies for their experiment, and we don't know that this can be applicable to all the AWS or Azure policies or not as there are approximately more than 200 services in this platform. Another factor worth considering is how quantitative analysis technique can be applied to other policy analysis problems and what will be the outcomes of that.

My views:

I think so variety of cloud services are used by different companies such as AWS, Azure, GCP and many more. These services preserve user data by enforcing access control policies. To get surety about correction of policy, an automated verification technique is needed which can assess the policies. As mentioned in the article AWS uses declarative statements to either grant or revoke access. In real world scenario, I opine that the policies are not straightforward, they are complex and may be a tough job to understand. Thus, the paper discusses about an automated method for calculating the permissiveness and relative permissiveness of the policies by transforming them to SMT formulas and then feeding it to a model counting constraint solver. According to me, it is a crucial task to quantify the permissiveness of a policy using model counting constraint solver. By quantifying policies thus not only decrease the vulnerability of data protection and security but in my perspective, it also improves the performance of the model.

References :

[1] W. Eiers, G. Sankaran, A. Li, E. O'Mahony, B. Prince and T. Bultan, "Quantifying Permissiveness of Access Control Policies," 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), 2022, pp. 1805-1817, doi: 10.1145/3510003.3510233.
URL: https://ieeexplore-ieee-org.ezproxy.library.dal.ca/document/9794078