A Guide to the GBC Chatbot Service

Last updated by Julie Lee, December 2019

Table of Contents

[An Introduction to the Ask George Bot 1](#_Toc27398438)

[Terminologies 1](#_Toc27398439)

[Intent 1](#_Toc27398440)

[Utterance 1](#_Toc27398441)

[Session Attribute 1](#_Toc27398442)

[Escalation 1](#_Toc27398443)

[Trackers 1](#_Toc27398444)

[Repeat 1](#_Toc27398445)

[Slot 1](#_Toc27398446)

[Lambda 1](#_Toc27398447)

[Response Card 1](#_Toc27398448)

[Secret 1](#_Toc27398449)

[Region 1](#_Toc27398450)

[IFrame 2](#_Toc27398451)

[Elicit Intent 2](#_Toc27398452)

[Elicit Slot 2](#_Toc27398453)

[Confirm Intent 2](#_Toc27398454)

[Close 2](#_Toc27398455)

[Dialog Code Hook 2](#_Toc27398456)

[Fulfillment Code Hook 2](#_Toc27398457)

[Delegate 2](#_Toc27398458)

[JSON config 2](#_Toc27398459)

[Mismatch Error 2](#_Toc27398460)

[Processing Error 2](#_Toc27398461)

[How the Bot is Organized 3](#_Toc27398462)

[What AskGeorgeBot is Composed Of 3](#_Toc27398463)

[AWS 3](#_Toc27398464)

[Amazon Lex 3](#_Toc27398465)

[AWS Lambda 3](#_Toc27398466)

[Amazon DynamoDB 3](#_Toc27398467)

[Simple E-mail Service 3](#_Toc27398468)

[Secrets Manager 3](#_Toc27398469)

[Identity Access Management 3](#_Toc27398470)

[CloudWatch 3](#_Toc27398471)

[S3 4](#_Toc27398472)

[Categorized Intents 4](#_Toc27398473)

[Slots and Intents 4](#_Toc27398474)

[Public pre-canned answers 4](#_Toc27398475)

[Public user-custom answers 4](#_Toc27398476)

[User dependent answers 4](#_Toc27398477)

[Role dependent answers 4](#_Toc27398478)

[App dependent answers 4](#_Toc27398479)

[List of Intents 4](#_Toc27398480)

[Session Attributes 6](#_Toc27398481)

[GBCToken 6](#_Toc27398482)

[GBCuserId 6](#_Toc27398483)

[GBCappId 6](#_Toc27398484)

[order 6](#_Toc27398485)

[lastUtterance 6](#_Toc27398486)

[currentUtterance 7](#_Toc27398487)

[intentBeforeTheLastIntent 7](#_Toc27398488)

[lastIntent 7](#_Toc27398489)

[currentIntent 7](#_Toc27398490)

[switched 7](#_Toc27398491)

[authApp 7](#_Toc27398492)

[authUser 7](#_Toc27398493)

[authRole 7](#_Toc27398494)

[login 7](#_Toc27398495)

[hideBubble 7](#_Toc27398496)

[firstAuthApp 7](#_Toc27398497)

[placeholder 7](#_Toc27398498)

[How Features Work 7](#_Toc27398499)

[How the Authentication Works 7](#_Toc27398500)

[How getStudentId works 9](#_Toc27398501)

[How Changing Applications Works for Agent 12](#_Toc27398502)

[How the Transcript Works 12](#_Toc27398503)

[Uploading 12](#_Toc27398504)

[Downloading 13](#_Toc27398505)

[Response Cards 14](#_Toc27398506)

[How Push Message Works 14](#_Toc27398507)

[How Escalate Works 16](#_Toc27398508)

[Forced Escalation 16](#_Toc27398509)

[Insult 16](#_Toc27398510)

[Unrecognized (Mismatch Intent) 16](#_Toc27398511)

[Repeat 16](#_Toc27398512)

[Overall Workflow 16](#_Toc27398513)

[E-mail Sequence 17](#_Toc27398514)

[A How-to Guide 18](#_Toc27398515)

[How to Test 18](#_Toc27398516)

[Testing on Test Site 18](#_Toc27398517)

[Testing on Postman 18](#_Toc27398518)

[How to deploy 19](#_Toc27398519)

[How to Modify S3 22](#_Toc27398520)

[How to Invoke a Lambda function 22](#_Toc27398521)

[Code 22](#_Toc27398522)

[How to Change the Colors 23](#_Toc27398523)

[How to Create a New Intent for AskGeorgeBot 24](#_Toc27398524)

[How to Check Billing for AWS 25](#_Toc27398525)

[How to Fix some Common Problems 25](#_Toc27398526)

[How to deploy updated version of Grails project on TomCat 26](#_Toc27398527)

[How to SSH into the directory to see the log 26](#_Toc27398528)

[Pure Chat (Live Chat) 27](#_Toc27398529)

[Integration with International Applications 27](#_Toc27398530)

[How integration between chatbot and PureChat works 27](#_Toc27398531)

[Sending Information to Pure Chat on OAS site 27](#_Toc27398532)

[Features of Pure Chat 28](#_Toc27398533)

[Appendix A: AWS Chatbot Guide Version I 29](#_Toc27398534)

[AWS 29](#_Toc27398535)

[Amazon Lex 29](#_Toc27398536)

[Intent 29](#_Toc27398537)

[Utterances 29](#_Toc27398538)

[Slots 29](#_Toc27398539)

[Fulfillment 29](#_Toc27398540)

[Tabs 29](#_Toc27398541)

[Sample utterances 30](#_Toc27398542)

[Lambda Initialization 30](#_Toc27398543)

[Slots 30](#_Toc27398544)

[Confirmation prompt 30](#_Toc27398545)

[Fulfillment 30](#_Toc27398546)

[Response 30](#_Toc27398547)

[Lambda 30](#_Toc27398548)

[The purpose of using AWS Lambda Function 30](#_Toc27398549)

[Secrets Manager 31](#_Toc27398550)

[IAM 31](#_Toc27398551)

[Appendix B: Pure Chat Guides 32](#_Toc27398552)

[Appendix C: Things Need to be Worked On & Note for Production 36](#_Toc27398553)

[AWS Lex Console 36](#_Toc27398554)

[Backend Service 36](#_Toc27398555)

[AWS Lambda Function 36](#_Toc27398556)

[Documentation 37](#_Toc27398557)

[PureChat 37](#_Toc27398558)

[Other Things 37](#_Toc27398559)

# An Introduction to the Ask George Bot

## Terminologies

### Intent

The action the user intends to do, determined by Lex’s AI abilities.

### Utterance

The text that the user enters into the bot.

### Session Attribute

Variables that are preserved for saving the context of the conversation within the same session. They are modified in lex-web-ui.js, lex-web-ui-loader-config.json, and AWS Lambda.

### Escalation

The process of exiting the chatbot conversation and escalating human involvement to an e-mail.

### Trackers

Five session attributes (currentIntent, lastIntent, intentBeforeTheLastIntent, currentUtterance, lastUtterance) which track what the user had previously said into the conversation.

### Repeat

When the user repeats the same intent three times or the same utterance twice. (Excluding eliciting of slots and confirmation prompts)

### Slot

A variable in an intent. It expires after the intent closes.

### Lambda

A function that the bot uses; named for the service that it uses.

### Response Card

A square card that appears in the chatbot with text (optional), images (optional), and buttons.

### Secret

Obtained using the secrets manager in AWS, the username and password for the web service.

### Region

An Amazon term for where the information is stored in the cloud. AskGeorgeBot is stored in N. Virginia.

### IFrame

The chat window that appears on the web site.

### Elicit Intent

A dialog action type for Lex that expects the next input of the user to be an utterance to be routed to an intent.

### Elicit Slot

A dialog action type for Lex that expects the next input of the user to be a slot. It also controls how the elicit slot prompt is presented.

### Confirm Intent

A dialog action for Lex which expects the user to confirm whether or not they want to continue with the current intent beyond the dialog code hook.

### Close

A dialog action type for Lex that expects no response from user. It is usually a message of some kind. It finishes the intent and fulfilled or failed.

### Dialog Code Hook

Invocation source before the slots are elicited.

### Fulfillment Code Hook

Invocation source after slots are elicit and immediately before the intent is fulfilled (or failed).

### Delegate

A dialog action type for Lex that terminates the dialog code hook and resumes fulfillment code hook.

### JSON config

The JSON lex-web-ui-loader-config.json file.

### Mismatch Error

Someone has indicated that the utterance has been incorrectly matched with an intent.

### Processing Error

an error occurred when getting an answer to the chatbot. This could be an error in the lambda, a lack of response from API, or a disconnection from AWS servers. This is almost always exception lambda unhandled. To solve these, you would need to check the cloud watch logs.

# How the Bot is Organized

## What AskGeorgeBot is Composed Of

AskGeorgeBot uses Amazon Web Services and Intellisoft’s chatbot service.

### AWS

AWS Console Password is saved as “awsConsolePassword” in “ChatbotServiceConfig.groovy” file.

### Amazon Lex

Amazon Lex is the core of the chatbot. It acts as a dispatch for utterances (what the user types) and intents (what the user wants to do). It is edited on the AWS console. Monitoring utterances can also be done through Lex in the monitoring tab.

### AWS Lambda

Lambda does most of the actual work. loadTranscript updates only the DynamoDB transcript and uploadTranscript downloads the dynamo transcript, dynamo intent, dynamo escalation table to the webservice database.

### Amazon DynamoDB

DynamoDB is the Amazon transcript table. Every time that a message is pushed into the chat, the information is stored in dynamo. An explanation of this process is found in How the transcript works.

### Simple E-mail Service

Simple E-mail Service or SES work in conjunction with lambda to e-mail admission officers. It verifies the e-mail addresses. I like to think of it as AWS’ address book.

### Secrets Manager

The secret login information service for the chatbot is stored here.

### Identity Access Management

The policies for roles are stored here. If there are any access issues, this is the first place you should check.

### CloudWatch

CloudWatch triggers the uploadTranscript lambda to download information to webservice. Additionally, CloudWatch contains all the logs for the lambda functions to help debug them if something goes wrong. To add more logs, enter print(<information you want to print>) into the lambda function.

### S3

All the JavaScript, CSS, and other loader files are stored in the S3 bucket. To edit these, see Modifications to S3 bucket. All of the modifications need to be done locally.

## Categorized Intents

In Lex, I have classified five types of intents based on the level of computation and authentication they require.

### Slots and Intents

You can create a new slot type, but if your selection is too long and keeps expanding, you would need to find another way.

### Public pre-canned answers

These are the responses that every user has access to, regardless of authentication and is non-customizable. In other words, it is public and does not require a web service. An example of this is admissionTestPlacementTestDiff.

### Public user-custom answers

These are responses that are available to the public but requires a level of computation to respond. Usually, they will need to understand the user’s input or access the web service. The intent programDuration is in the category. It uses the lambda function publicInformation.

### User dependent answers

These are responses that require information about the user themselves. The parameter used is user id, not appid. This is an important distinction because agents, can access this information if not on an application and the authentication does not authenticate the application. An example of an intent in this category is getName. It uses the lambda function knowUserLambda.

### Role dependent answers

These are responses that require knowing the role of the user. This would allow routed frequently asked questions so that gents and applicants get different responses. A current example of this intent is getRole. It uses the lambda function knowRoleLambda.

### App dependent answers

These are responses that require the appId parameter. This is the highest level of authentication. All intents such as getStatus use knowAppStatus as the lambda function.

### List of Intents

This is the list of intents that are currently available on ChatBot service. As of December 2019, there are more intents that were updated, so please check out AWS Lex console to see more.

1. Admission requirement
2. What is the difference between the admission test and the placement test?
3. How long do I have to wait for my admissions test result?
4. When do international students need to apply by?
5. What is the application processing time?
6. Apply for sin
7. Will GBC help me to apply for my study permit?
8. I would like to book an assessment test
9. I would like to book a campus tour
10. Can't log into stu-view
11. Finding co-op placement
12. Who should I contact if I haven't received my student ID number yet?
13. Is the digital media marketing program available for international student?
14. I am an international student; do I need SIN?
15. EAP admission
16. I will be in the English for Academic Purposes EAP program. Can I work off campus during my studies?
17. What English pathway schools does GBC work with?
18. Enrolment letter for visa
19. How to change fee status to domestic?
20. How do I get a study permit?
21. How to get graduation letter?
22. How do I apply as an international student?
23. What PTE score do I need to apply as an international student?
24. What does implied status mean in terms of my study permit visitor status or work permit?
25. Where can I find information about international transactions?
26. How do I get my invoice?
27. do you offer master's program?
28. How can I meet with someone to get more information on how to apply?
29. Withdraw deadline passed
30. Payment options for new students
31. Payment options for returning students
32. Is my program eligible for the Post Graduate Work Permit PGWP?
33. How can I prepare and take the admission test?
34. What programs are available for international students?
35. What does it mean if a program is not available?
36. I am applying as an international student, but the program I am interested in is now closed
37. Proof of English proficiency
38. Will my children have access to public school education, and will they have to pay?
39. When will I receive my refund?
40. Where to pick up refund cheque
41. I would like to drop one course because of an exemption or because I want to take it in another semester. Will I get a refund for that one course?
42. Re-submitting my application
43. I am an International Student and I’m travelling with my spouse or common-law partner can they work while I study
44. I am graduating, and my study permit will expire before graduation. Do I have to extend my study permit?
45. Do I need my PTE results at the time of applying?
46. I would like to know the tuition for my program.
47. Can I take an unscheduled break from my program?
48. Where do I upload study permit?
49. How much is the refund amount for visa refusal?
50. I am in Canada on a visitor visa. Am I considered as an international student?
51. What is the Post-Graduation Work Permit?
52. Can I work off campus with my co-op work permit?
53. Can I work on campus?
54. I'm in Canada on a work permit, and I want to study at George Brown. What do I need to do?
55. Can I continue to work while my post-graduation permit application is being processed?

## Session Attributes

As stated in the definitions, session attributes remain for the duration of the conversation with the chatbot. Everything is stored as a string because there are issues with storing other types.

### GBCToken

The token generated in the JavaScript to identify session

### GBCuserId

Appuser id of the user passed in through browser script.

### GBCappId

App id that the user is looking at passed through browser script.

### order

Counter to track what number bubble has been passed into chat for transcript purposes. Updated when message is pushed into chat.

### lastUtterance

One of the trackers which store that last utterance that the user entered into the bot

### currentUtterance

One of the trackers which stores the current utterance that user entered in the bot to get current intent

### intentBeforeTheLastIntent

One of the trackers which stores the intent room two intents ago

### lastIntent

One of the trackers which stores the previous intent

### currentIntent

One of the trackers with stores the current intent

### switched

Session attribute for whether or not user has crossed intents for a break sequence

### authApp

Whether or not the user is authorized to view the application

### authUser

Whether or not the user is authenticated

### authRole

The role of the user (agent, applicant, agent counselor)

### login

whether or not user has logged in already in a login window that session

### hideBubble

Boolean to hide or show the bubble

### firstAuthApp

Indicator about whether or not the agent has already accessed an application because the first time they will be asked who they want to get information for.

### placeholder

placeholder for slots when there are issues in infinite loops and rerouting intents

# How Features Work

## How the Authentication Works

\*\*\*Under ChatbotService>grails-app>services>ca>georgebrown>chatbot>ApiService\*\*\*

There are three levels of authentication: user authentication, role authentication, and app authentication.

In user authentication, the userId and token is passed to web service to see if the userId was assigned that token. In role authentication, the web service returns the role and completes user authentication. Finally, in addition to user and role authentication. App authentication checks to see if the user is authorized to view that application by getting user role and searching for the user id of that role assigned to the application.

On OAS, the login attribute will be false and the GBCToken will be valid. Therefore, getAuthUser(), getAuthApp(), getAuthRole() will be executed. Otherwise, a login frame shows up in the chatbot frame. Please note that the login string as of August 2019 is ‘logging [in...fiwn23lsr@nd2#](mailto:in...fiwn23lsr@nd2)’. Here is a flowchart depicting the workflow. For the purposes of simplicity, the escalation workflow is excluded. There is a mistake: ‘GBCToken in session attributes or GBCToken = ‘testtoken’ should be ‘GBCToken not in session attributes or GBCToken=‘testtoken’.

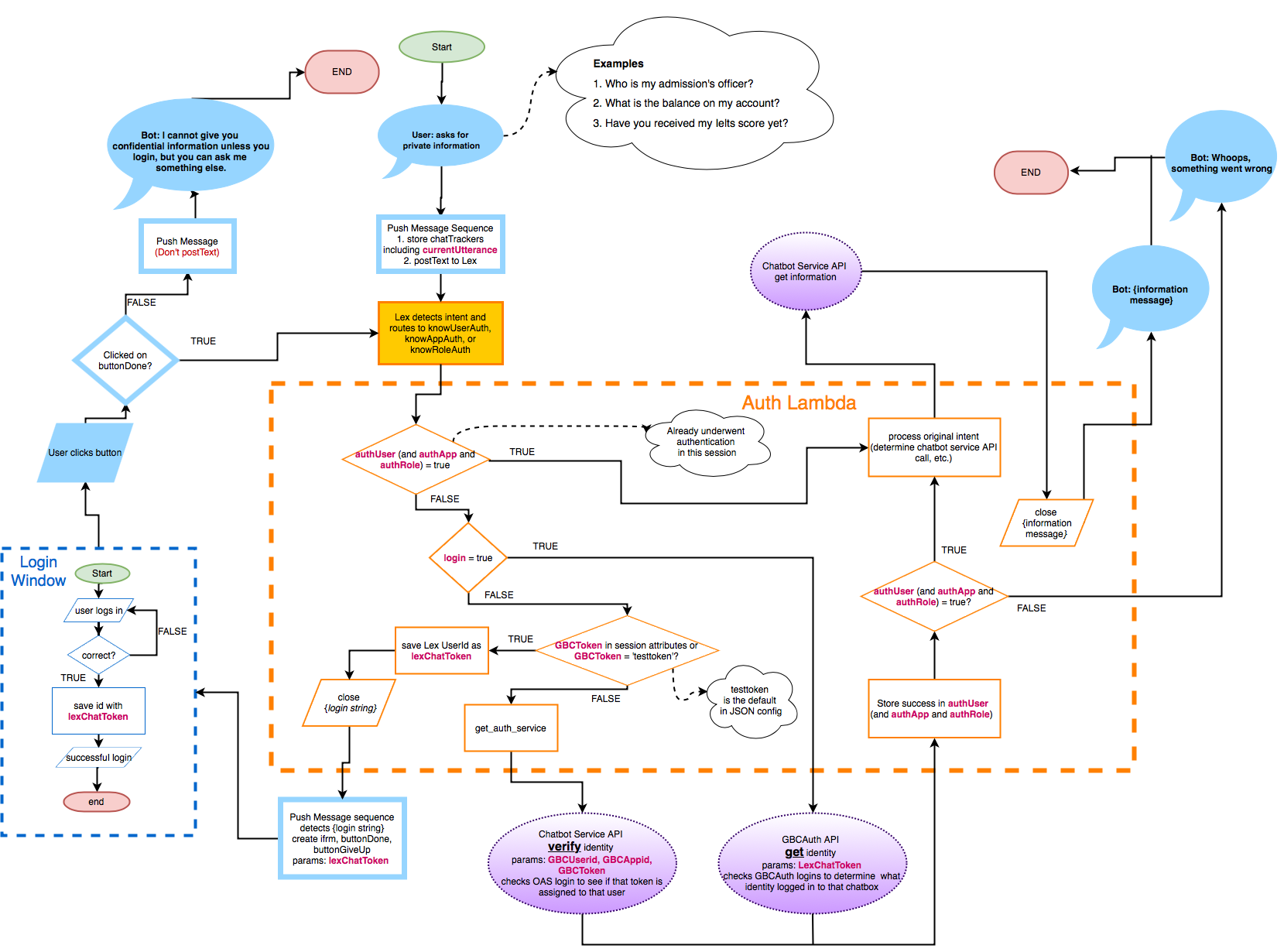


Figure 1. Workflow of AWS

## How getStudentId works

“getStudentId” is an intent that is supposed to get user’s student ID when the user is not logged in. If the user is not logged in and trying to use a chatbot service, the user will be prompted with a message saying, “Hi, I'm George, the GBC International Admissions Chatbot and I am here to answer your questions. To serve you better, can you please provide me with your GBC student ID number? It should be 9 digits long and start with number 1. If you do not have a GBC student ID number yet, please enter ‘no ID’.”

There are two AWS Lex intents that could be invoked by this: ‘getStudentId’ and ‘noStudentId’. If the user answers with actual student ID, ‘getStudentId’ will get invoked. As soon as it is checked that the student ID exists in OAS database, the provided student ID will be posted as session attribute, and it needs to be saved for the rest of the chat session.

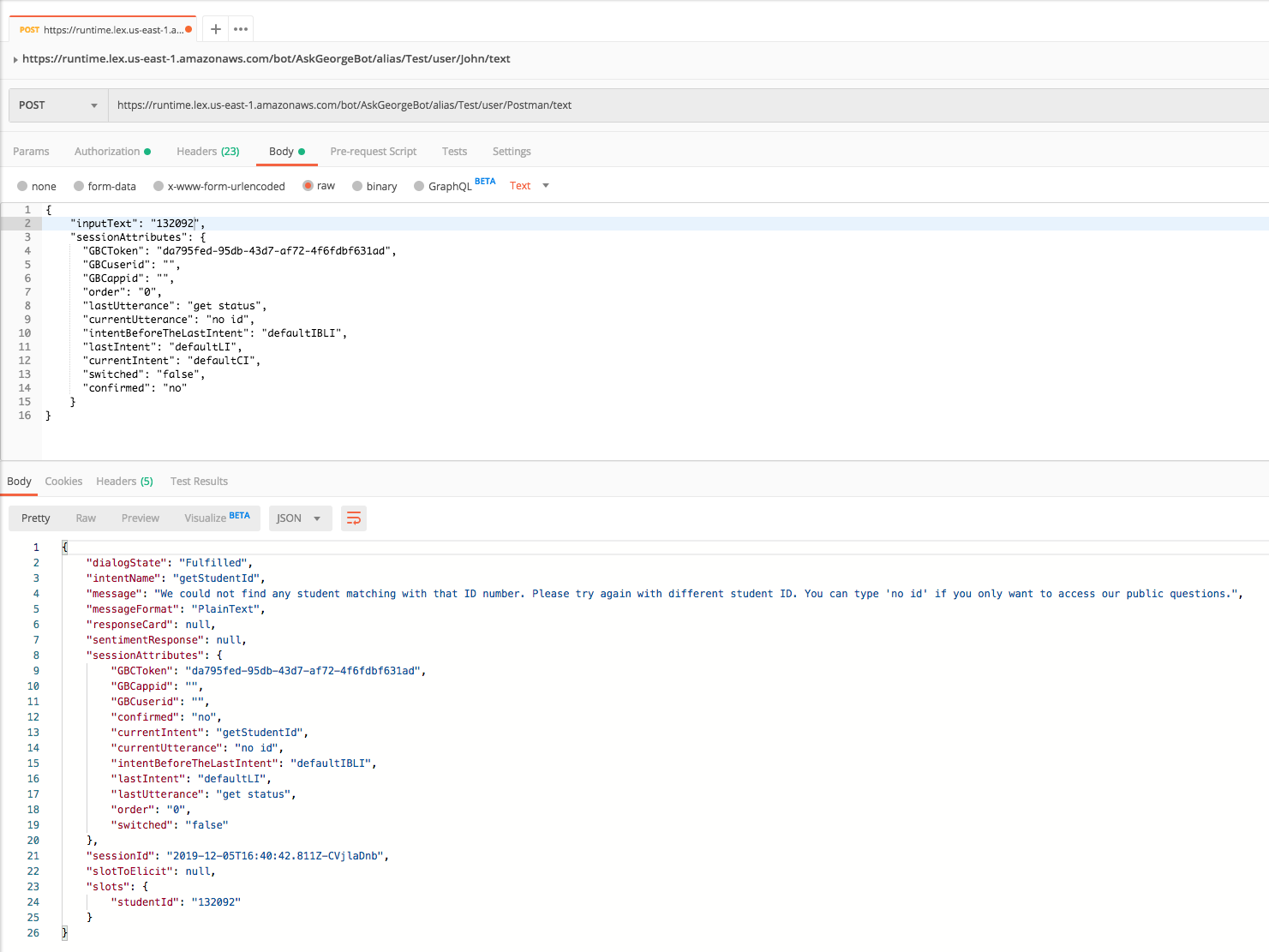


Figure . Case 1: when the public user answers with wrong student ID number (Postman)

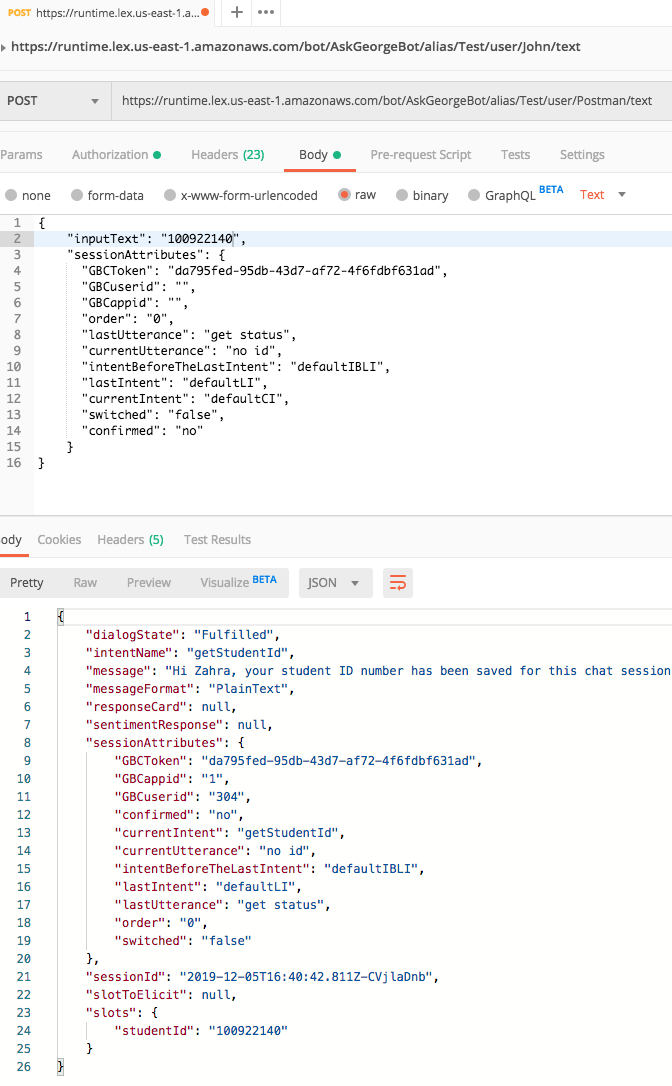


Figure . Case 2: when the public user answers with the matching student ID number (Postman)

If the user is logged in and somehow typed his/her student ID number, the chatbot will redirect the intent to “unrecognized” as if it does not have that intent. The user has to be NOT logged in to invoke “getStudentId” and “noStudentId” intent.

If the user answers with ‘no ID,’ that person would have no access to any of the private questions. The user will need to log in to invoke private intents.

The used AWS Lambda function for “getStudentId” and “noStudentId” is “publicInformation,” and you can check out “checkStudentId” on our backend service (Project: ChatbotService).

# How Changing Applications Works for Agent

## How the Transcript Works

The transcript is a combination of lambda, JavaScript, web service and dynamo. In the JavaScript, a Lambda function is invoked to pass a message to the dynamo.

### Uploading

Uploading the transcript is done from the chat iFrame to DynamoDB.

#### JavaScript

The transcript is updated every time a message is pushed into the chat. Push a message refers to the act of adding the text to the main chat dialog box for either the bot or the user. This function is found in the file lex-web-ui.js and is called push message.

There are five inputs to the lambda.

InputData is stored in ‘text’ of the message variable. It is the text inputted by user or bot. The dialog state is the speaker (bot or user).

The dialog state is determined by the message[‘type’].

The userid is the appuser\_id in the database. This was a session attribute passed by the web browser script when loading the chatbot. Similarly, token is a session attribute passed by the browser.

The most complicated parameter is order. The order tracks the number of bubbles in the chat. The order is stored in session attributes as a string. This needs to be converted into an int to increment and passed to lambda because the dynamo reads the integer. However, it needs to be converted back into a string to be stored in session attributes.

Session attributes are accessed using: context.state.lex.sessionAttributes.attributename;

#### Lambda

The lambda is called test loadTranscript. It reads in the event data (input from the JavaScript) and sets default values if unknown. The lambda writes to the TranscriptAskGeorgeV5 table. The table requires the keys: ‘UseridPrimaryKey’ and ‘OrderBubble’. Additional attributes created in lambda are ttl, TimeSent, and orderString.

Note that it only updates the “TRANSCRIPT”, not “INTENT” nor “ESCALATION”. These two tables in DynamoDB are updated separately in each Lambda function. (check escalate.py)

It needs to be assigned a role with permissions to access DynamoDB and to be invoked by the Cognito identity provider. This is done by configuring the Cognito\_askGeorgeTestUnauthRole policy called lambda-invoke-custom and adding the correct resources to be invoked.

#### DynamoDB

The table is called AskGeorgeTranscriptV4. The primary key is comprised of UseridPrimaryKey and its sort key Order. It stores the Dialog (speaker) and Input (message) as attributes. The values are printed by UseridPrimaryKey and then BubbleOrder.

To query the database, you must include UserIdPrimaryKey.

The UserIdPrimaryKey is the token.

#### Time to Live (TTL)

Time to live is the attribute determining when the items will expire. It takes up to 48 hours to expire. For instructions on how to deal with expired items in query use: <https://jun711.github.io/aws/how-to-set-ttl-for-amazon-dynamodb-entries/>. The time to live attribute is called ttl. TTL is set using epoch time. The attribute’s value is set in the lambda writing to dynamo.

week = datetime.datetime.today() + datetime.timedelta(days=7)

expiryDateTime = int(time.mktime(week.timetuple()))

ttl = expiryDateTime

### Downloading

Downloading the transcript refers to the process of retrieving the transcript data from DynamoDB and inserting it into the chatbot service database. It uses CloudWatch to trigger a AWS Lambda which passes information to the web service to add to the database.

#### CloudWatch

As of December 2019, the CloudWatch triggers the download process every 5 minutes for the testing purposes. It can be found listed as a rule called uploadToDatabase.

#### Lambda

The invoked lambda is called uploadTranscript. It scans the table and stringifies the JSON input, and then it is sent to the URL.

Since, the URL request cannot handle more than 25 items being passed, the lambda divides the string into individual objects.

The individual JSON item is stringified using dumps and then cycled through a loop to replace all the spaces with ‘%20’, curly brackets with their UTF-8 codes, and remove any HTML embedded links.

#### Web Service

The ‘\’ are removed from the JSON string before parsing it with JSON slurper. The primary key transcript\_id is a concatenation of the token and order string. Transcript\_id, appuserid, time sent, dialog, chat message, and order string are written into the table called transcript if the record does not already exist.

The function updateTranscriptDatabase in oasService does most of the work.

## Response Cards

Response card titles are disabled in the chat for two reasons. Firstly, the transcript does not receive data from the titles so the conversations are disjointed in the transcript without adding identical information to a bubble. Secondly, the response cards do not format nicely in the iframe. The only remaining aspect of the response card is the buttons.

The response card data can be modified in Lex, but most of the response card modifications are done in the corresponding lambda function.

## How Push Message Works

Push message is the process in lex-web-ui.js that prints the bubble to the chat window. This is also where I’ve put all of my modifications for the chatbot JavaScript. You can find it by searching ‘pushMessage: function pushMessage(context, message)’.

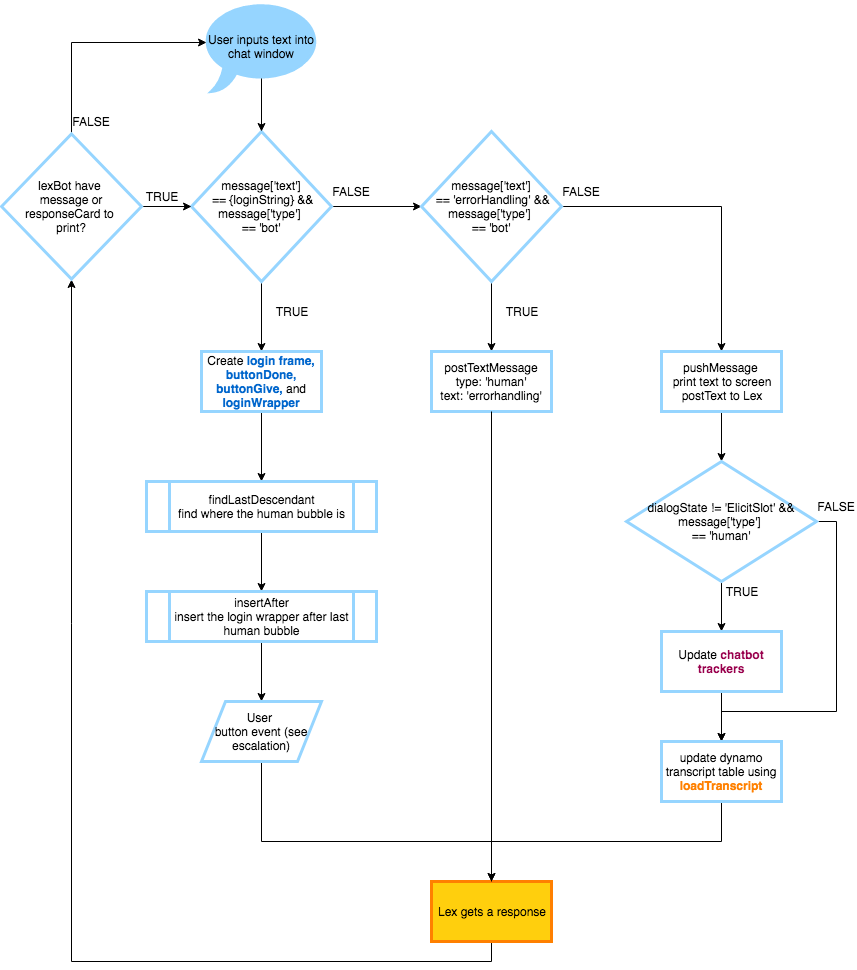


Figure : How Push Message Works

## How Escalate Works

### Forced Escalation

Forced escalation is done through the break intent. The break intent includes utterances including ‘can I talk to a person’, ‘leave’, ‘break’, and ‘exit’. During the Dialog Code Hook, since the switched session attribute is false, the current intent session attributes is updated and Lex executes a delegate action.

In the fulfilled code hook, the message is set to “I’m sorry that I couldn’t help today”. Then, the e-mail sequence is executed until close.

### Insult

Currently, frustration is detected through the insult intent which includes utterances including: poop, you are stupid, et cetera. The information flow is identical to force escalation with one exception is that the message is ‘You seem to be getting frustrated.’

### Unrecognized (Mismatch Intent)

An unrecognized utterance is when Lex cannot determine the intent from the input. After each unrecognized utterance, a clarification prompt is sent to the console. After three clarification prompts, the hang-up prompt is sent, initiating the email sequence. The message is “I can’t seem to process your request”. During the clarification stage, Lex gives a hint to the user saying, “Sorry, I don't understand. Could you rephrase? You can type "email" to email someone directly.” If the user types the keyword ‘email,’ it will go to forced escalation. Otherwise, after typing 3 unrecognized intent, Lex is going to wait for proper intent response from the user.

### Repeat

In every intent, the dialog code hook checks the trackers. If the past three intents are the same or the last two utterances, there is a repeat and the sequence continue. Otherwise, the bot executes a delegate to resume the normal answer flow in the fulfillment dialog.

When the sequence continues, the user is asked to confirm their intent. During the confirm intent action, the intent is changed to break.

A message states that the user keeps asking the bot the same thing and asks if they need help. If the user says no, the confirm intent action is used again to reroute the user to their original request. This is done by presenting an apology message and asking the user to click a continue button to re-enter the same input as before.

If the user says, yes, the e-mail sequence begins in the fulfillment code hook.

### Overall Workflow

The following figure shows the workflow.

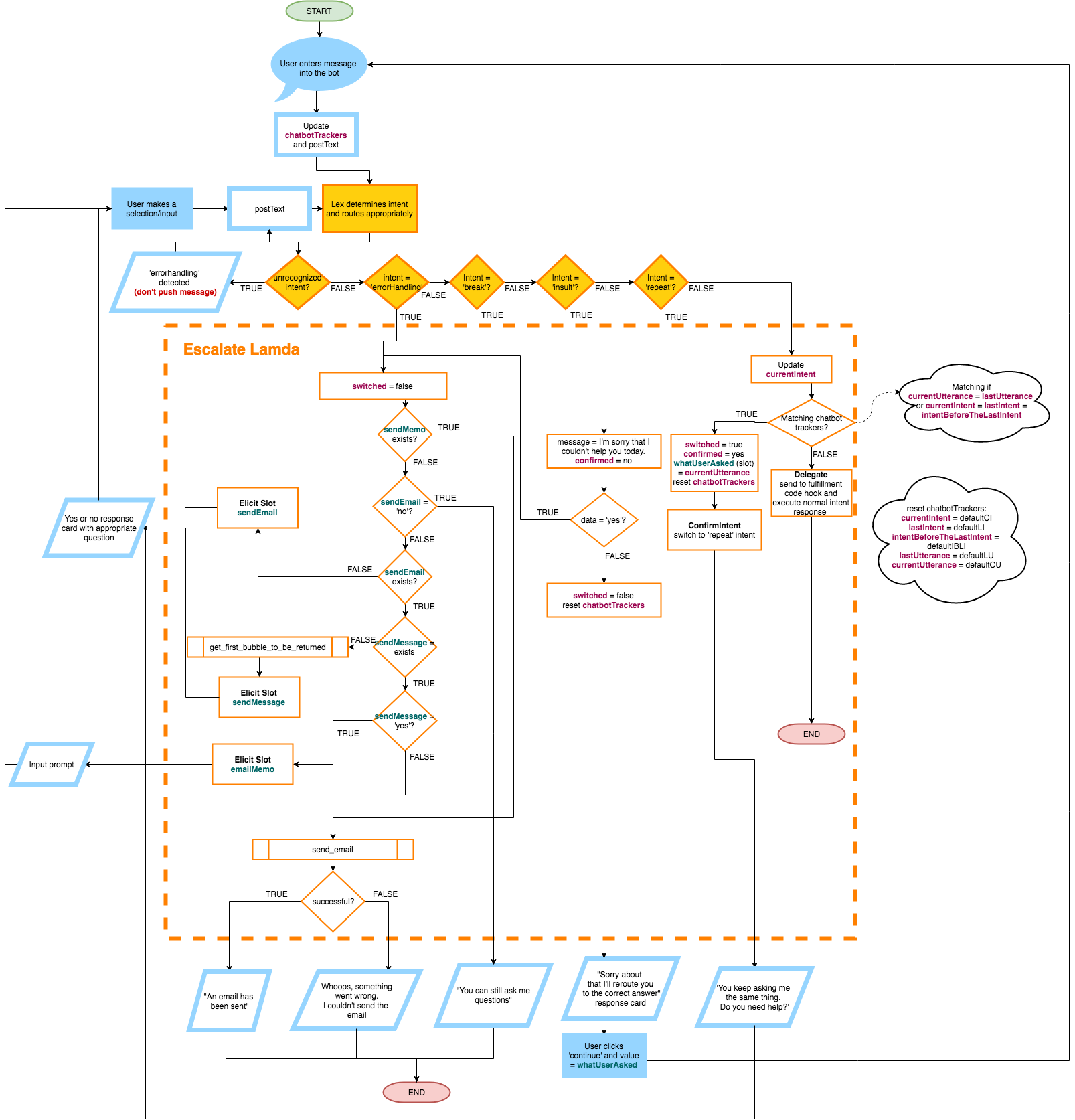


Figure 5: Escalation Sequence Workflow

### E-mail Sequence

The first question asks the user if they would like to send an email. This fills the sendEmail slot. If the user says no, the e-mail is sequence is closed with a message. If the user says yes, the sequence continues.

Note that if the chat bot says something like, “Whoops, something went wrong. Couldn’t send the email.” It might indicate that you are trying to send the email to non-verified user. You need to make sure that the receiving email address is verified on AWS SES server, also change the email setting on AWS Lambda escalate.

Then, the first bubble to be returned is obtained by getting the bubble whose order is the current order minus 19. This is printed in a message to the user. The user is asked if they want to add anything. This fills the sendMessage slot. If yes, they are prompted to enter emailMemo. Afterwards, the e-mail is sent. The HTML is built and compiles the transcript and additional message.

After that, the Lambda function (escalate.py) will try to access the database to get the user’s email address (“emerg\_email” column), and if the email address does not exist for the specific user, it will ask for it from the user. Same thing happens for the non-logged-in users, and the chatbot would also ask for their first name, too. The email sent to the admissions officer will always contain the user’s name and email address.

# A How-to Guide

Figure 6:Sample of Email Sent to Admissions Officer After Escalation Sequence

## How to Test

### Testing on Test Site

You cannot test full functionality in the Lex console because the session attributes are tracked in lex-web-ui.js. You must publish and refresh the page. You may need to clear the history.

(Use testing email addresses for this. Chat bot won’t pop up for non-applicant email addresses.)

On the test site, the GBC attributes are filled to test the login frame uncomment the following line in lex-web-ui.js

context.state.lex.sessionAttributes.GBCToken = 'testtoken';

### Testing on Postman

URL:https://runtime.lex.us-east-1.amazonaws.com/bot/AskGeorgeBot/alias/Test/user/Postman/text

Body:

{

"inputText": "exit",

"sessionAttributes": {

"GBCToken": "###############",

"GBCuserId": "####",

"GBCappId": "####",

"order": "0",

"lastUtterance": "What is the difference between the admission test and the test",

"currentUtterance": "get status",

"intentBeforeTheLastIntent": "defaultIBLI",

"lastIntent": "defaultLI",

"currentIntent": "defaultCI",

"switched": "false",

"confirmed": "no"

}

}

Authorization: See IAM users myBotUser. You may need to make a new user to get the secret access key.

Access key: ############

Secret Key: ############

Also, note that you cannot invoke private method on Postman as it requires authentication process.

For example, if you want to test “getName” intent on Postman, you would need to fake actual conversation token, GBCuserId, and GBCappId to make it work. (See the “**appuser**” table in OAS database.)

I am not sure if previous co-op made a separate postman account for herself, but since it does not share the regular AWS Console account, I have created a new one (Dec 2019), and the ID and password are the same as the AWS Console one.

**Use Postman app instead of the web version (it is already installed on this computer**,) as it is a lot easier to configure as you want.

## How to deploy

1. Set up cloud formation. See <https://aws.amazon.com/blogs/machine-learning/deploy-a-web-ui-for-your-chatbot/>. It is super important to put in correct web app parent origin. However, you can modify JSON config later.
2. Modify JSON config. I’ve highlighted the parts that made need to be changed. The new JSON config should match this one with the exception of the yellow highlighted parts. The top cognition section is from cloud formation build. Do not change after build.

{

"cognito": {

"poolId": "us-east-1: ############",

"appUserPoolClientId": "############",

"appUserPoolName": "us-east-1\_35LG1Ha2q",

"appDomainName": "############.auth.us-east-1.amazoncognito.com",

"appUserPoolIdentityProvider": ""

},

"lex": {

"botName": "AskGeorgeBot",

"botAlias": "Test", -- NOTE: DO NOT USE $LATEST

"sessionAttributes": {

"GBCToken": "testtoken", DO NOT CHANGE THIS!

"GBCuserid": "userid",

"GBCappid": "appid",

"order": "0",

"lastUtterance": "defaultLU",

"currentUtterance": "defaultCU",

"intentBeforeTheLastIntent": "defaultIBLI",

"lastIntent": "defaultLI",

"currentIntent": "defaultCI",

"switched": "false",

"confirmed": "no",

"lexChatToken": "defaultToken",

"login": "false"

},

"initialText": "Hi, I'm George. Ask me your questions here.",

"initialSpeechInstruction": "Say 'Buy Flowers' to get started.",

"reInitSessionAttributesOnRestart": false

},

"ui": {

"parentOrigin": "https://dmzgrg01u.georgebrown.ca",

"toolbarTitle": "Ask George",

"toolbarColor": "blue",

"textInputPlaceholder": "Type your questions here",

"toolbarLogo": "",

"enableLogin": false,

"AllowSuperDangerousHTMLInMessage": true,

"shouldDisplayResponseCardTitle": false,

"pushInitialTextOnRestart": false,

"directFocusToBotInput": false,

"showDialogStateIcon": false

},

"polly": {

"voiceId": "Salli"

},

"recorder": {

"preset": "speech\_recognition",

"enable": false

},

"iframe": {

"iframeOrigin": "https://lex-web-ui-codebuilddeploy-############-webappbucket-############.s3.amazonaws.com",

"shouldLoadIframeMinimized": false,

"iframeSrcPath": "/index.html#/?lexWebUiEmbed=true"

}

}

1. Set-up purechat. In unavailable settings, st “When unavailable show” to “No Chat Box”. Set the snippet link (from purechat website) and put it into host website’s JavaScript code.
2. Insert the following code to load the chatbot. I’ve highlighted the parts that would need to change.

<script src="https://s3.amazonaws.com/lex-web-ui-codebuilddeploy-############-############-############/lex-web-ui-loader.min.js"></script>

<script type="text/javascript">

var loaderOpts = {

baseUrl: 'https://lex-web-ui-codebuilddeploy-############-webappbucket-############.s3.amazonaws.com/'

};

var loader = new ChatBotUiLoader.IframeLoader(loaderOpts);

var loaderCfg = {

lex:

{

"sessionAttributes":

{"GBCToken": pToken,

"GBCuserid": pUserid,

"GBCappid": pAppid }

},

ui:

{

"parentOrigin":<https://dmzgrg01u.georgebrown.ca>

}

}

loader.load(loaderCfg).catch(function (error) {

console.error(error);

});

purechatApi.on('chatbox.available:change', function (args) {

if(args.available == true) {

var chatbot = document.getElementById(

"lex-web-ui-iframe");

chatbot.parentNode.removeChild(chatbot);

}

});

</script>

1. Delete the EC2 resource that cloud formation made. Maybe wait 24 hours so it doesn’t affect any delayed builds.

## How to Modify S3

1. Download desired file
2. Make changes and save with the same name
3. Re-upload - set to public read access
4. In top bar- click latest version and the top version (It may already look selected do it again.
5. Confirming proper file is read:
   * + - 1. If modifying JSON config file, you’re done.
         2. If modifying lex-web-ui-loader.min.js, you may need to change link in JavaScript to the Object URL of latest version. This is kind of odd, it sometimes needs this. Try without it first.
         3. If modifying any other file (lex-web-ui.js, lex-web-ui.css), check lex-web-ui-loader.js and go to DependencyLoader dependencies. Change the URL as appropriate. This should not be necessary, but is a good backup plan.

Note: It may take a while to update. This is especially the case with lex-web-ui-loader.js. You may need to close the browser before opening it again.

## How to Invoke a Lambda function

This will have some repetitive information from the transcript section, but it also contains some additional information if the Lambda function were to be invoked elsewhere in the function. To invoke the Lambda function, you need to call the function outside of Lex.

The code can be included inside a function and then called in the JavaScript.

### Code

Below is the code for invoking the Lambda function. I’ve highlighted the sections that would need to be changed.

/ /Initialize the Amazon Cognito credentials provider

AWS.config.region = 'us-east-1'; // Region

AWS.config.credentials = new AWS.CognitoIdentityCredentials ({

IdentityPoolId: 'us-east-1: ############', - unless using askGeorgeTest});

//input into lambda

var input = {

inputData: ‘hi’,

inputData2: ‘hi again’

};

lambda.invoke({

FunctionName: 'testLambdaInvoke',

Payload: JSON.stringify(input}, function(err, data) {

if (err) {

console.log(err, err.stack);

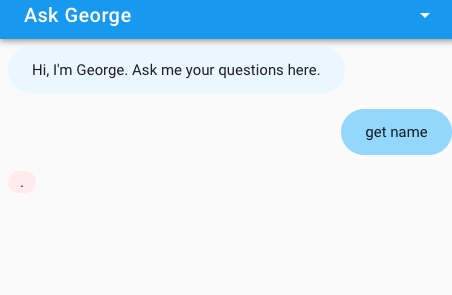
}

});

Extra configurations for lambda permissions may need to be added. I suggest, comparing it to the current loadTranscript.

## How to Change the Colors

Don’t forget to update the appropriate file. See modifications to S3 bucket.



**Bot Response Bubble**

**User bubble**

**Thinking Bubble**

**Toolbar**

Figure : Lex Web UI Colors

Toolbar: lex-web-ui-loader-config.json 🡪 ui, toolbar color

Bot Response bubble: lex-web-ui.css 🡪 .message-bot .message-bubble[data-v-46cd8705]{background-color:#ebf7ff}

User bubble: lex-web-ui.css 🡪 .message-human .message-bubble[data-v-46cd8705]{background-color:#96d5ff}

Thinking bubble: lex-web-ui.css 🡪 message-bot .message-bubble[data-v-6d624218]{background-color:#ebf7ff}.footer

## How to Create a New Intent for AskGeorgeBot

1. In Lex console, add slots, utterances, and message.
2. Set the initialization dialog code hook to escalate.

⇑ Note that due to the size limit of the AWS Lambda function, I have created “escalate2”, which is supposed to do the same thing as “escalate.” However, note that you do not have to add the initialization dialog code hook if you are using Lambda function itself as a response. (I.e., knowUserLambda, knowAppLambda, etc.) You only need this when you are directly creating a response from AWS Lex console to write a transcript and intent table in Dynamo DB.

1. Add appropriate slots and ‘placeholderSlot’ which is type YESORNO. Anything regarding app authentication needs ‘desiredApp’, ‘changeApp’ and ‘desiredAppEmail’. Look at getStatus or equivalent for types.
2. If applying lambda, apply appropriate function and edit it. Note that every change to lambda immediately goes to production I would make a test lambda first or production level lambas.
3. In dispatch, add an if statement directing to the correct function.
4. Create the intent’s fulfillment function under “Functions that control the bot’s behavior”.
5. Ensure that you obtain the session\_attributes first:

session\_attributes = intent\_request['sessionAttributes'] if intent\_request['sessionAttributes'] is not None else {}

1. If authentication is needed add:

authItem = auth\_session(intent\_request)

1. Divert action in another if statement depending on whether or not authItem is “true”. (Note that it is a string not a boolean)
2. In the true section, ensure that you get the secret if accessing the web service:

secrets\_resp = get\_secret()

secrets\_data = json.loads(secrets\_resp)

username = secrets\_data['username']

password = secrets\_data['password']

You will need to include the get\_secret function in the lambda if it is not already included.

1. At the end of if block, pick a message.
2. In else block set message to “Whoops. Something went wrong”
3. Finish with:

return close(

session\_attributes,

'Fulfilled',

{

'contentType': 'PlainText',

'content': message

}

)

## How to Check Billing for AWS

All page numbers are approximate.

1. Login to ITSWebSupport AWS console
2. Click ‘ITS Web Support’ in top navigation bar
3. Click ‘My Billing Dashboard’
4. Click ‘Payment History’ under ‘Billing’ in left tool bar
5. Click on ‘Invoice’ for date. It’s a pdf that you can print.
6. Total fee is on page 1
7. Details for both accounts together on pages 2-3
8. Totals per account is on page 4
9. Details per account on pages 5 – 7

Accounts should be merged for billing, but if payment is different check other account.

## How to Fix some Common Problems

This is a guide for all the problems that seem weird but actually have a little fix.   
When there is a problem you don’t know how to solve, look here first.

Table 1: Problems and the Corresponding Little Fix

|  |  |
| --- | --- |
| **Problem** | **Fix** |
| Everything on the console is gone. The lambda functions are missing; the bot is missing. Everything is deleted. | Check the region. Make sure that it is set to N. Virginia. The reason that it’s not Canada is that some services aren’t available in Canada. |
| In the browser, there is an error saying that it can’t access <insert file name here> | In S3, make the file public |
| The new file that has been uploaded to S3 is not the one loading in the browser. | Clear the browser’s cache and upload it again. |
| File won’t upload onto S3. | Try another browser. Google chrome is a good choice. |
| Bot won’t proceed past dialogCodeHook | Delegate state needs to have some slots. That’s why I have ‘placeholderSlot’??? |
| Bot isn’t updating | Check the cognito credentials in config JSON |
| Response card isn’t working | There can be a maximum of four slots |

#### Still Not Helping?

Amazon Support will only help you with anything on the console.

Anything JavaScript is part of Lex-Web-UI open source project on github: <https://github.com/aws-samples/aws-lex-web-ui>.

## How to deploy updated version of Grails project on TomCat

1. Check in the code -> remember the number
2. Clean the project -> Tools command “clean” -> command “test war”
   1. .war file is created under: ChatbotService>target
3. Go to: dmzmsa01.georgebrown.ca/manager
4. Undeploy chatbot service
5. Choose file -> Use the number to find the correct project (change name to ChatbotService##number) -> deploy

## How to SSH into the directory to see the log

1. Open the command window.
2. Type: ssh root@dmzmsa01.georgebrown.ca
3. cd /usr/share/apache-tomcat-7.0.75/
4. cd logs/
5. tail -f \*.out
   1. -f means it will print out whatever is at the end of the file, so it will dynamically update the terminal as something is added to the log file.

# Pure Chat (Live Chat)

## Integration with International Applications

There are two different presentations of the live chat widget depending on whether or not the chatbot is on. The chatbot being on or off is a setting in the international application.

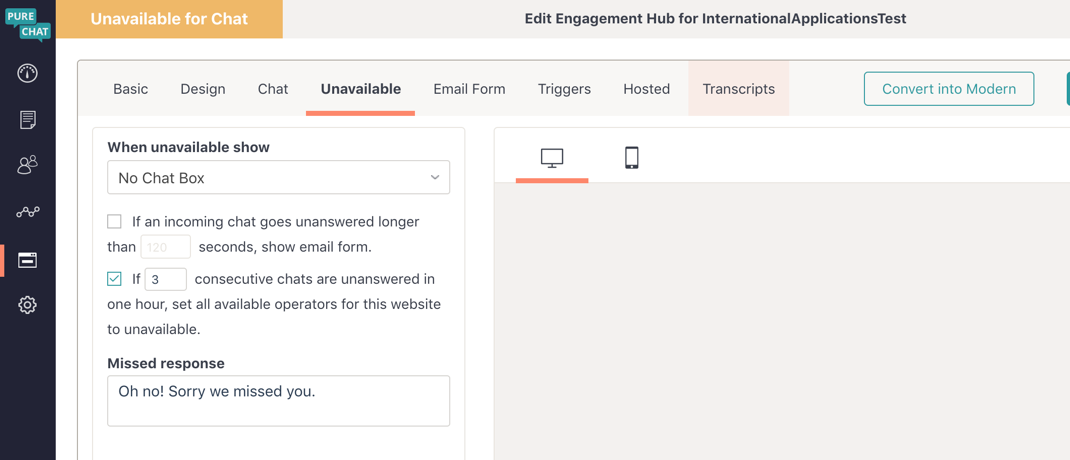
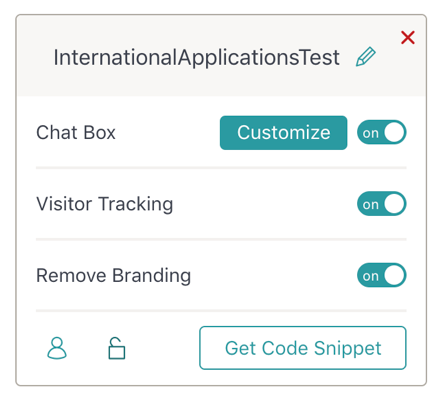
**Under no circumstances should you update the chat window to modern. Some API features are not supported.**

I have a concern with CORS. The chatbot is occasionally not loading on the page until the page is refreshed. The order of pure chat and chatbot seems to help.

### How integration between chatbot and PureChat works

**P**lease note that PureChat has higher priority than chatbot. With that said, PureChat always needs to be on when it is available, and chatbot should be always on when PureChat is NOT available. I have made the change on Chatbot test site, but the changes have not been made for the actual application site (only PureChat is being used as of Dec 2019.)

Figure 8. How to Modify PureChat



Check above three fields to deactivate ‘sending email’ on PureChat. If you select “No Chat Box” option, the chatbot will automatically activated when PureChat is unavailable.

## Sending Information to Pure Chat on OAS site

The pure chat API (<https://purechat.com/support/javascript-api>) is limited on what fields of visitor information can be set. As of July 2019, the banner id, OAS user id, and OAS app id are set to email, company, and phone respectively. The information is set using: purechatApi.set(**'visitor.company'**, ***infoOASApp***).

The user can change their visitor details by finding an HTML edit code snippet online and adding a button that calls to pure chat API. If they are able to succeed in this method, there is an additional call to the API that triggers whenever the visitor details are changed. It checks if the changes are invalid. If the changes do not match what the field is supposed to be, the field is set to INVALID. This cannot be changed from the web browser if the user is not using local host.

## Features of Pure Chat

Pure chat is pretty simple to use, please refer to Cheat Sheet for Pure Chat V2 for the Operator Guide and Cheat Sheet for Pure Chat for the developer guide.

The operators asked me to enable file transfer which is under chat/chatting in customize website. Also, I have operators filtered between IELTS and International.

# Appendix A: AWS Chatbot Guide Version I

## AWS

Once you are logged in, you should see something like this:

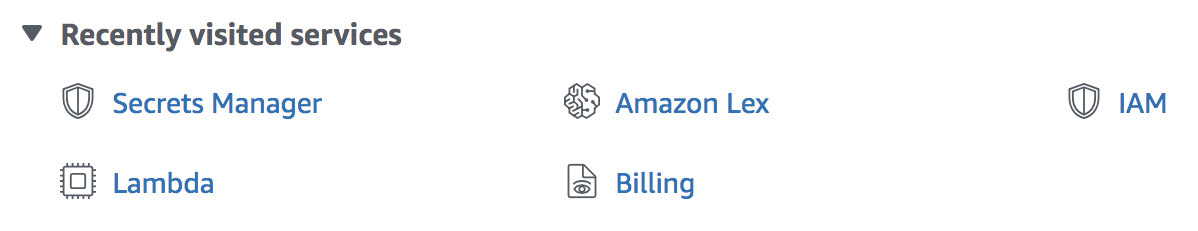


Figure . AWS Frequently Used Services

## Amazon Lex

### Intent

An intent is what the user is trying to do. An intent performs an action in response to natural language user input

E.g.: user wants to rent a car, check their application status

### Utterances

Spoken or typed phrases that invoke your intent

If a user wants to check their application status there are many ways in which they would express that. They could say “What is my status?” or “What is the status of my application” etc. These are all utterances.

### Slots

Slots are input data required to fulfill the intent (kind of like parameters)  
E.g.: to know which application the user is referring to, the chatbot might need to know what their id is

### Fulfillment

Fulfillment mechanism for your intent

E.g.: What steps should the chatbot take once it knows the user wants their application status?

### Tabs

When you open a bot, you will see 4 tabs on the top, let’s go through each of them briefly:

#### Editor

The main tab which contains all the intents that your chatbot understands

#### Settings

the tab which allows you to control which version of the bot your web app is using (this allows you to keep building new test bots without affecting the actual bot that is deployed on your website)

#### Channels

Not really applicable to us but used to connect your bot to things like Facebook, Kik, etc.

#### Monitoring

It contains the basic statistics relating to your bot including the number of times it failed or was successful. If you click the utterances button on the left, you will see the specific text inputted by the user that was missed or successful

### Sample utterances

Examples of what the user might type in order to invoke this intent.

### Lambda Initialization

Check this box if you want to run a Lambda function the moment this intent is called (refer to below to understand what lambda functions are)

### Slots

Any slots that are needed are specified here, and you can also specify the type of input that is allowed for this slot. There are many built-in slot types such as numbers and cities but you can also create your own custom ones if you need to

### Confirmation prompt

Sometimes you may want to check if this intent is the one the user is talking about so you can add a confirmation here.

### Fulfillment

Once you have all the information you need (slots, confirmation etc.), you can once again call a Lambda function to do something fancy with the information (or just return parameters).

### Response

Use HTML for formatting.

## Lambda

### The purpose of using AWS Lambda Function

#### Customize the user interaction

For example, when Joe asks for available pizza toppings, you can use prior knowledge of Joe's choices to display a subset of toppings.

#### Validate the user's input

Suppose that Jen wants to pick up flowers after hours. You can validate the time that Jen input and send an appropriate response.

#### Fulfill the user's intent

After Joe provides all of the information for his pizza order, Amazon Lex can invoke a Lambda function to place the order with a local pizzeria.

Head to the Lambda service console to see the functions we have. AskGeorgeBot contains a function named “get\_status” that calls the web service on our website to determine the “status”. This lambda function is called whenever the getStatus intent is triggered on the Lex chatbot. If you go back to the Lex console and go to the getStatus intent, you will see that this function is used for Fulfillment.

When a Lambda function is called from Lex, the dispatch function is called. The values from the intent are stored in intent\_request. To add new intents that the lambda function manages, first define a new function that will be run. Then, just add an elif statement in dispatch which will then call the correct function. (You could do this in other ways since it is just code, but this is how the examples did it so I feel like it would be wise to follow this style)

## Secrets Manager

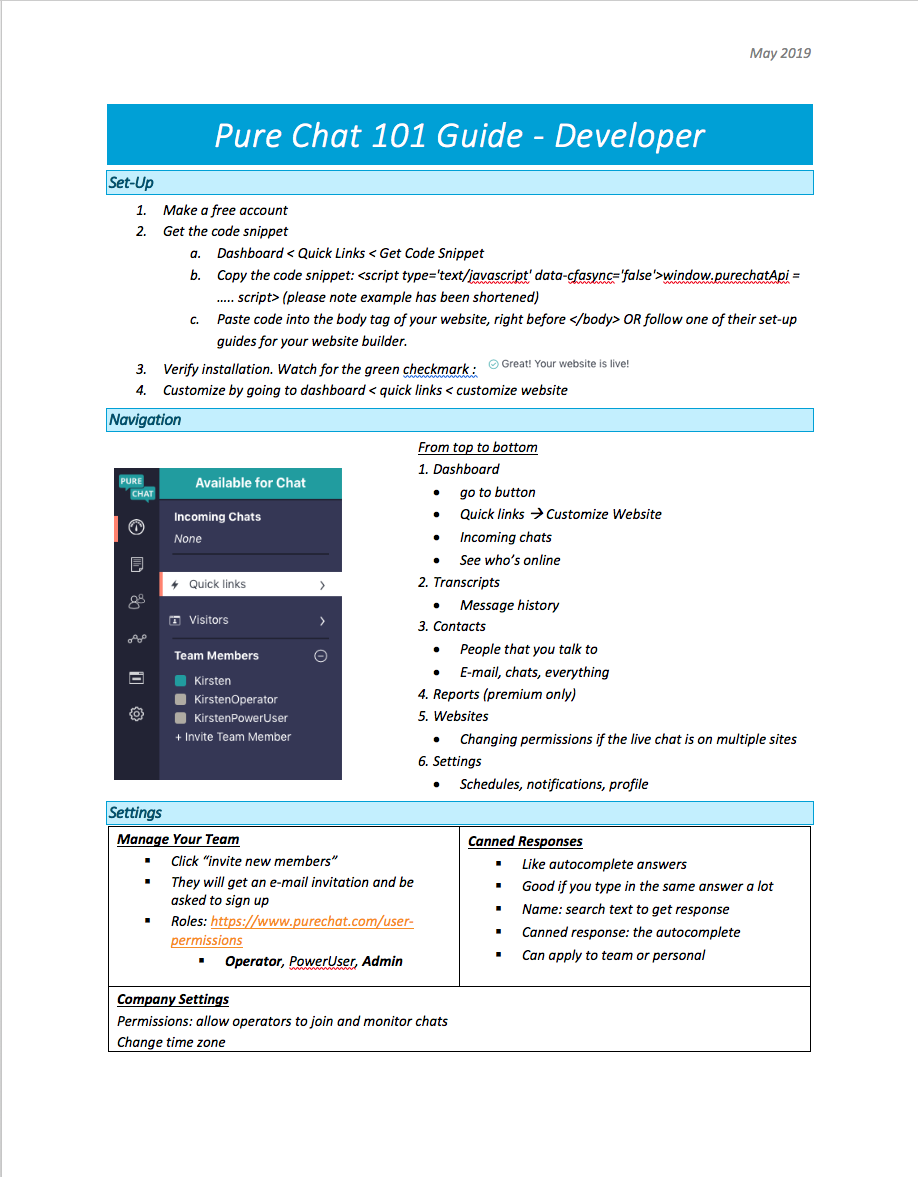
We have created a username and password that our Lambda function can use in order to call our chatbot web service. This way people who do not have a username and password cannot access the web service. We use the secrets manager to keep this information safe.

The secrets manager allows you to store key/value pairs securely. TestChatbot contains the username and password required to access our chatbot web service. You can change the values at any time, just go into TestChatbot > Retrieve Secret Value > Edit. This is one way that we are currently securing the information passed between our websites and Amazon Lex.

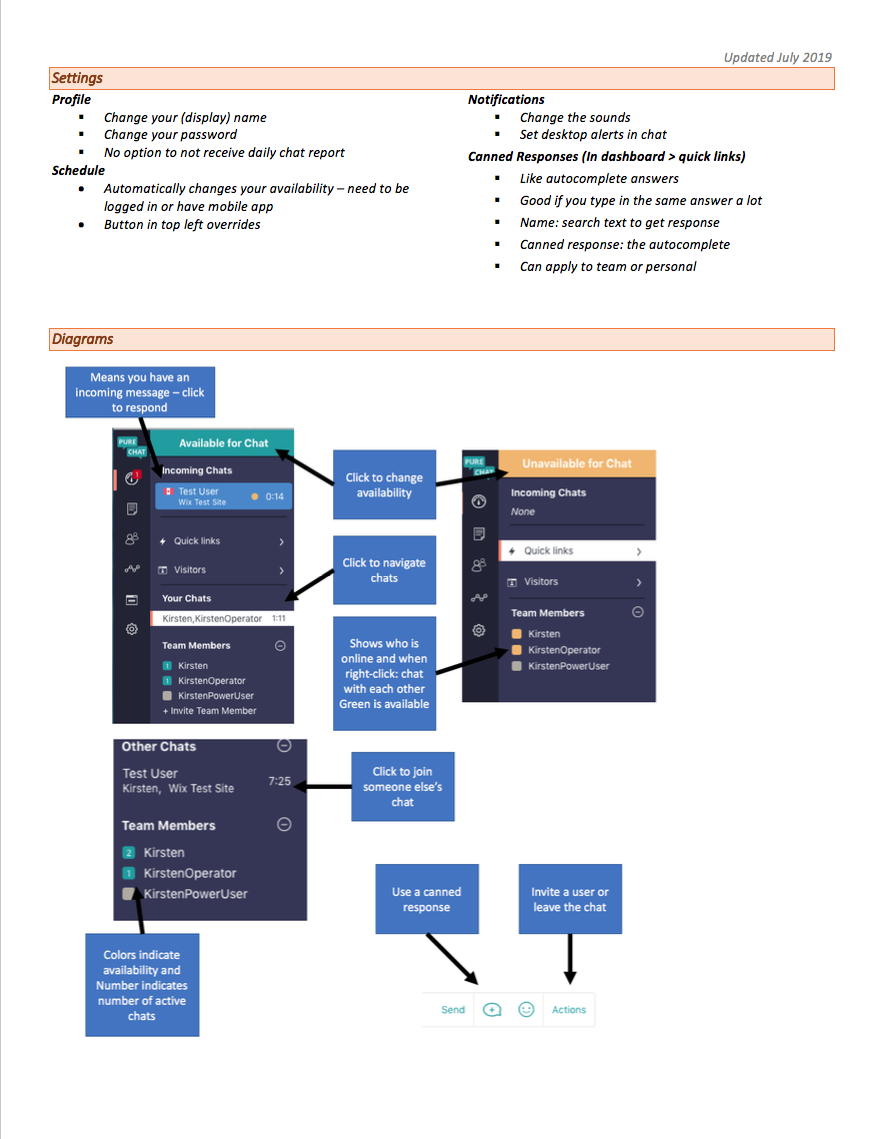
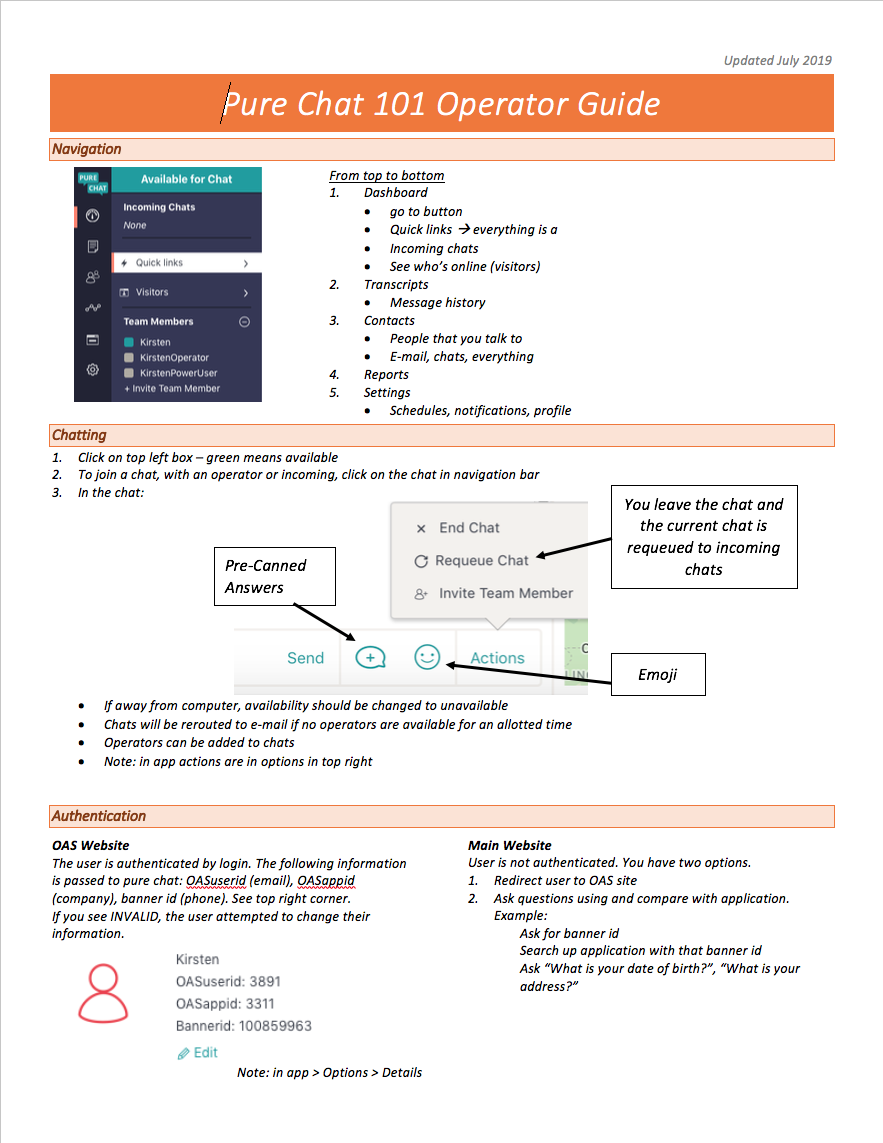
## IAM

The IAM allows you to manages roles within your account. By giving a role to each service, this restricts them from accessing things they are not supposed to. So, if you do not give your lambda function permission to access the secrets manager, it would not be able to get any of that information. This is a layer of security between the amazon services.

If you look at the existing roles, you will see a TestChatbotRole. This role has permission to both lambda and secrets manager. Right now, the lambda function AskGeorgeBot has this role which means it can access information stored in the secrets manager. You can attach new policies which give it more permissions if you need to.

Appendix B: Pure Chat Guides





# Appendix C: Things Need to be Worked On & Note for Production

This list is created in December 2019 by Julie Lee. As I think these are all essential part to make chatbot work as intended, please complete the followings.

## AWS Lex Console

1. Some of the utterances need to be updated or added for each intent as they do not cover entire possibilities.
2. Please make sure to test all the slot values are working and add more examples to the custom slot types.
3. You may find it difficult to find an exact slot type for specific values as there is no “Literal” type. For example, I used AMAZON.Organization in “getTuitionByCode” intent for “code” slot. You probably need to create a slot type if AMAZON.Organization does not work out. That is the closest slot type to the Literal, so you should try out that one first.
   1. “Literal” means wildcard string values.

## Backend Service

1. The database for the private information needs to be updated before the deployment. For example, the tuition information is not correct at this point.
2. For Chatbot Service website, please fix the code so that people can download the files on the “Help” tab on the test website, too. (It only works on local machine at this point.)
   1. There are currently 4 documents, and they can be found under: ChatbotService/documents.
3. Ask ***Ashley Sullivan*** about international admissions officers by the regions. When the users want to contact their admissions officer, the chatbot should be able to automatically detect who their admissions officers are by their country of origins. I have created sample function in OasService, so you can just change the format and put actual data in to use it.
4. There are two columns called “actual\_bannerId” and “bannerId” under “applicant” table, **YOU MUST USE “ACTUAL\_BANNERID”, NEVER USE “BANNERID”!**
5. For “help” section in ChatbotService, the document upload directory is saved in Config.groovy file as file.upload.directory = “/tmp” right now. However, I believe there is better directory to save all of the documents on the website. Ask Tony about the directory format.

## AWS Lambda Function

1. We are using too many Lambda functions, and it is not very convenient. *Maybe* we can refactor so that some of them can be combined together. (This is **optional**, and I am not even sure if we should do it since each Lambda function has a limit on the number of invocation sources, and if the number of the Lex intents increase, we would need to create another one anyway.)
2. “uploadTranscript” function updates the database on our backend service. Right now, it is getting triggered by AWS CloudWatch every 5 mins. When everything is ready to get deployed on the actual website, please change the cron trigger on CloudWatch Rules “uploadToDatabase” to every day at 11:59 PM.
3. All of the ‘sendee’ and ‘sender’ emails in “escalate” file must be changed to actual email address (currently they are all mine.) Read Backend Service 3 and 4 first to get the proper email for the specific user. Check **escalate.py** in Lambda.
4. When you call a backend API service, you need to provide a username and password. Otherwise, it will display “Authentication failure.” The username and password can be obtained using: “get\_secret()” function. Take a look at other lambda functions for a reference.
   1. If it is a public function that requires backend API service calls, then you can simply add the function name to the list in “oasService.getPublicFunctions()”.

## Documentation

1. I was trying hard to document everything so that if you are new to AWS Lex, you can understand most of the stuffs only by looking at these documents. However, as I was not new to AWS console, it might be hard to pick up for you as there are some new terms and functionalities. So, I would recommend writing everything down that made you confused for the future co-ops.
2. If you change anything on Research and Chatbot Service, please change the document accordingly, especially the UI part, since the documents will be also used for non-developers.

## PureChat

1. You need to look at the transcript regularly and update the AWS Lex intent as more questions are being asked by the user. Make sure you confirm with Ashley or Tony about the question and answer before you publish it to the actual website.
2. Please note that you need to change the setting as mentioned in How integration between chatbot and PureChat works before you deploy the Chatbot to the actual website.

## Other Things

1. Figure out the use of Testing on Postman. I am certain that it can be used somewhere, but I am not sure where exactly we can implement it.
   1. One of the options is that you can use it to test what it would display if the user is not logged in. For example, to test How getStudentId works for the users who are not logged in yet, you can delete GBCuserId and GBCappId parameters from the body.
   2. Postman testing does not work for the private intent unless you copy and paste actual person’s conversation record from “appuser” table under OAS database. (i.e. fake token, userId, and appId) -> This is caused by the authentication process in AWS Lambda. Check “authUser” function in the backend.
2. In AWS Secrets Manager, make sure you change the secrets. Discuss with Tony to decide on the password, but currently it is **NOT SECURE**.
3. The policies for IAM roles are pretty relaxed, so you may want to restrict them by adding more roles and making them more specific.
4. There is another project called “GBCAuth,” and this is like a pop-up window we use when the user is not logged in and try to ask a private question. In Figure 1. Workflow of AWS , it is described as a ‘log in window’ on the bottom-left side of the figure. The problem here is that there is no button to close the window, and you would need to make an API call to the ChatbotService project to let it know the person is authenticated. Also, it is hard to find out where this function is implemented, since you can’t mock it with Postman.