A Guide to Ask George Bot, Amazon Web Services, Pure Chat and all the Other Chatbot Things

Table of Contents

[An Introduction 1](#_Toc19517701)

[Definitions 1](#_Toc19517702)

[How the Bot is Organized 2](#_Toc19517703)

[What AskGeorgeBot is Composed Of 2](#_Toc19517704)

[Amazon Lex 2](#_Toc19517705)

[AWS Lambda 2](#_Toc19517706)

[Amazon DynamoDB 2](#_Toc19517707)

[Simple E-mail Service 2](#_Toc19517708)

[Secrets Manager 3](#_Toc19517709)

[Identity Access Management 3](#_Toc19517710)

[CloudWatch 3](#_Toc19517711)

[S3 3](#_Toc19517712)

[Categorized Intents 3](#_Toc19517713)

[Public pre-canned answers 3](#_Toc19517714)

[Public user-custom answers 3](#_Toc19517715)

[User dependent answers 3](#_Toc19517716)

[Role dependent answers 3](#_Toc19517717)

[App dependent answers 3](#_Toc19517718)

[List of Intents 3](#_Toc19517719)

[Session Attributes 5](#_Toc19517720)

[How Features Work 6](#_Toc19517721)

[How the Authentication Works 6](#_Toc19517722)

[7](#_Toc19517723)

[How Changing Applications Works for Agents 7](#_Toc19517724)

[How the Transcript Works 8](#_Toc19517725)

[Uploading 8](#_Toc19517726)

[Downloading 9](#_Toc19517727)

[Response Cards 9](#_Toc19517728)

[How Push Message Works 9](#_Toc19517729)

[How Escalate Works 10](#_Toc19517730)

[Forced Escalation 11](#_Toc19517731)

[Frustration 11](#_Toc19517732)

[Unrecognized 11](#_Toc19517733)

[Repeat 11](#_Toc19517734)

[Overall Workflow 11](#_Toc19517735)

[E-mail Sequence 12](#_Toc19517736)

[A How-to Guide 15](#_Toc19517737)

[How to Test 15](#_Toc19517738)

[Testing on Test Site 15](#_Toc19517739)

[Testing on Postman 15](#_Toc19517740)

[How to deploy 15](#_Toc19517741)

[How to Modify S3 18](#_Toc19517742)

[How to Invoke a Lambda 18](#_Toc19517743)

[Code 18](#_Toc19517744)

[How to Change the Colors 19](#_Toc19517745)

[How to Create a New Intent for AskGeorgeBot 20](#_Toc19517746)

[How to Check Billing for AWS 20](#_Toc19517747)

[How to Fix some Common Problems 21](#_Toc19517748)

[How to deploy updated version of Grails project on TomCat 22](#_Toc19517749)

[How to SSH into the directory to see the log 22](#_Toc19517750)

[Pure Chat (Live Chat) 23](#_Toc19517751)

[Integration with International Applications 23](#_Toc19517752)

[Chatbot Off 23](#_Toc19517753)

[Chatbot On 24](#_Toc19517754)

[Sending Information to Pure Chat on OAS site 24](#_Toc19517755)

[Features of Pure Chat 24](#_Toc19517756)

[Chatbot Service 24](#_Toc19517757)

[Documentation 24](#_Toc19517758)

[Transcripts 24](#_Toc19517759)

[Intent Monitoring 24](#_Toc19517760)

[Errors Escalation 25](#_Toc19517761)

[Prompting to log-in page for private information 25](#_Toc19517762)

[Appendix A: AWS Chatbot Guide Version I 26](#_Toc19517763)

[Appendix B: Pure Chat Guides 30](#_Toc19517764)

[Appendix C: Notes for Future Developers 34](#_Toc19517765)

[Support Request August 23, 2019 34](#_Toc19517766)

[Concerns with Current Implementation August 2019 34](#_Toc19517767)

A Guide to AskGeorgeBot, AWS, and all the Other Chatbot Things

# An Introduction

This guide is divided into five sections (exclusive of this one). How the Bot is Organized reviews the particular structure of the AskGeorgeBot. The objective of this section is to inform the reader about the structure of the bot so that if it needs to be developed or modified, it can. The next section, How Features Work, pertains to the AskGeorgeBot. These are already working features that are explained for debugging purposes, documentation, or so that a developer understands the process before making any modifications. A How-to Guide is the only section of the document that can apply to multiple bots. It includes instructions on how to make some modification to the AskGeorgeBot and how to add certain bot features to any bot. This section includes a guide for fixing some issues with unusual but little fixes. The next section is everything about Pure Chat which is the live chat. Finally, chatbot service is the custom chatbot monitoring system.

## Definitions

*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:* A variable that is preserved for the entire duration of the chat. They are modified in lex-web-ui.js, lex-web-ui-loader-config.json, and 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)

*Slots:* 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.

### 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. As of June 2019, AskGeorgeBot has seven important lambda functions. knowUserLambda, knowAppLambda, publicInformation and knowRoleLambda all deal with intent fulfillment. They are described in categorized intents and authentication. Escalate is responsible for the e-mailing function of the bot when the user gets upset with chatbot and checking whether or not to break out of chat. loadTranscript updates only the DynamoDB transcript and uploadTranscript downloads the dynamo transcript, dynamo intent, dynamo escalation table to the webservice database. The remaining function in lambda are either examples or defaults created by the stack.

### 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.

Note for production: change the secret. It’s currently not that secure.

### Identity Access Management

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

A note for production, the policies are pretty relaxed so you may want to restrict them by adding more roles and making them more specific.

### 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.

### 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.

We intentionally disabled all the intents other than public pre-canned answers as of September 2019, until we make sure everything (including private intents) is working properly.

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 : Flowchart of Authentication Flow

### How Changing Applications Works for Agents

Changing the application can be automatic, as is the case for the first question involving application information indicated by ‘firstAppAuth’ session attribute or triggered by the ‘changeApplicant’ intent. The applicant role cannot change their application.

This process occurs after the authentication sequence and before information is returned.

The only difference between triggering the change with ‘changeApplicant’ and the automatic process that an additional slot ‘changeApp’ is elicited. It’s a yes or no answer to whether or not they would like to change the information.

When triggered, a response card with four of the agent’s authorized applications and an ‘other’ button is printed. The authorizedApps are stored in a session attribute and obtained in app authentication. This response card elicits the ‘desiredApp’ slot. If the slot is one of the buttons, authenticate\_app\_chosen verifies that the value is in the authorized app and sets the application to the selected button value. If the user selects ‘other’, another slot, ‘desiredAppEmail’ is elicited. The user has to enter the email because there are too many common names. The input is validated using validate\_app which calls the chatbot web service to find the appid associated with the email and ensure authorization of the user.

## How the Transcript Works

The transcript is a combination of lambda, JavaScript, web service and dynamo. In the JavaScript, a lambda 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 lambda which passes information to the web service to add to the database.

#### CloudWatch

As of June 2019, the CloudWatch triggers the download process every Friday at 11:59 p.m. E.S.T. 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.

The stringified JSON input 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)’.

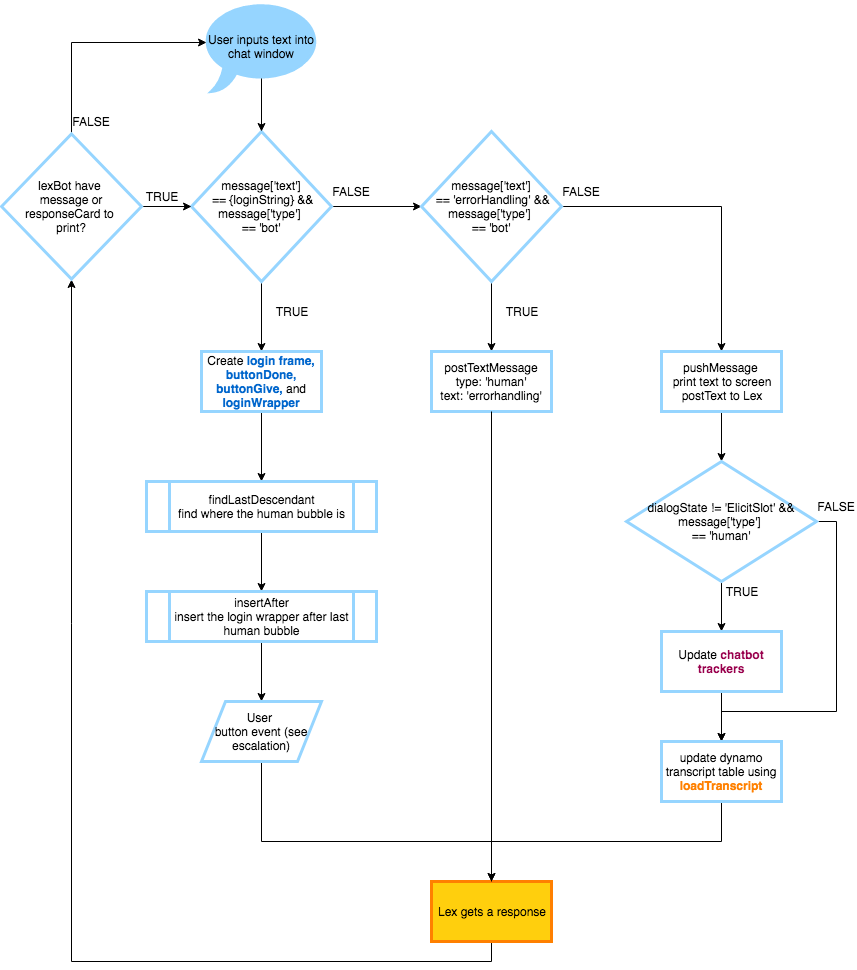


Figure : How Push Message Works

## How Escalate Works

The escalate sequence is initiated in four ways: when the user forces an escalation, when the user displays frustration, when user inputs an unrecognized utterance and when the user repeats the same utterance. Once, the escalation is confirmed, the e-mail sequence begins.

### 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.

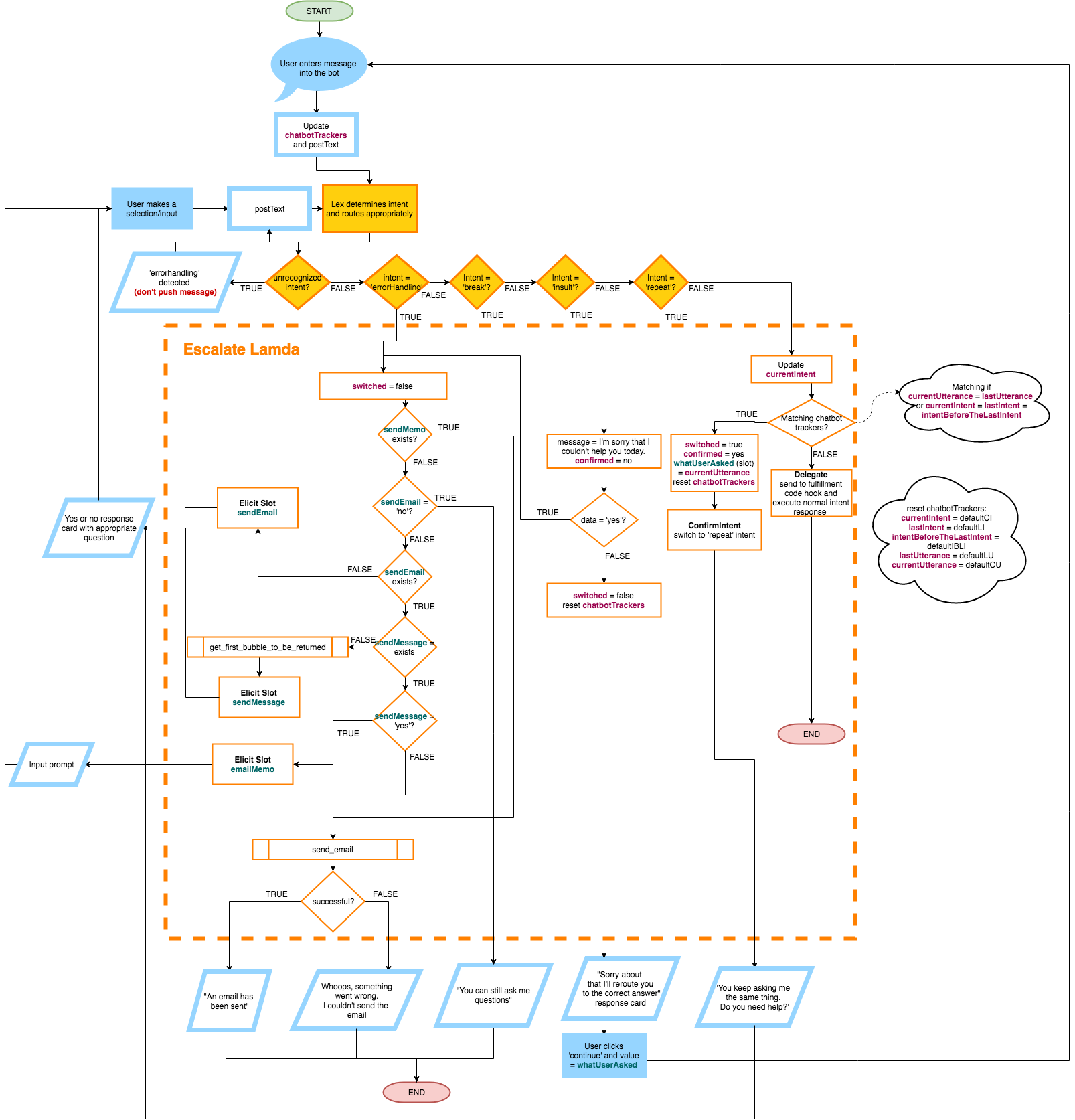


Figure : Escalation Sequence Workflow

### 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 23. 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.

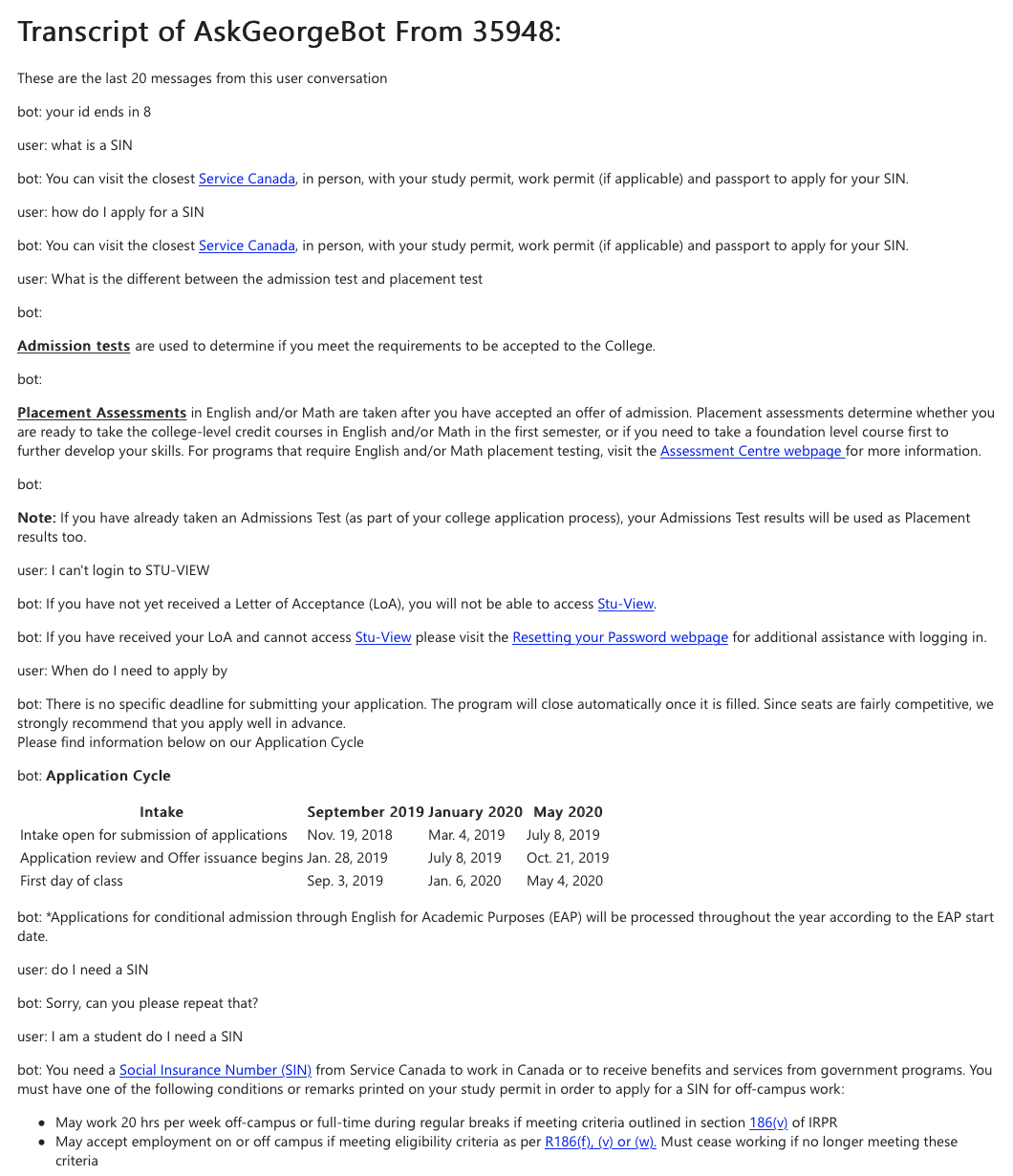


Figure :Sample of Email Sent to Admissions Officer After Escalation Sequence

# A How-to Guide

## 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.

Test site: <https://dmzgrg01u.georgebrown.ca/International/login/auth>

(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

I still haven’t figured out post content for slots and such but here is the format for a simple conversation item.

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

Body:

{

"inputText": "exit",

"sessionAttributes": {

"GBCToken": "da795fed-95db-43d7-af72-4f6fdbf631ad",

"GBCuserId": "3891",

"GBCappId": "3311",

"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: AKIASZUNEYGFEOFNTVWU

Secret Key: j6gKP/MOGshq1gXOGYDR4/Fk5CbPDLHVVno71ZeH

## 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-1:58c67c32-6950-4c1e-ba5a-44a18eb31066",

"appUserPoolClientId": "1i2tbskrlc7d21afg8ald9763u",

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

"appDomainName": "lexkwebkuikidentitypoolconfigkdopbwnnftbfc937127992510.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-vta1xvwx7-webappbucket-28hnhqlor04o.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-vta1xvwx7-webappbucket-28hnhqlor04o/lex-web-ui-loader.min.js"></script>

<script type="text/javascript">

var loaderOpts = {

baseUrl: 'https://lex-web-ui-codebuilddeploy-vta1xvwx7-webappbucket-28hnhqlor04o.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.

A note for production: the published intents should be a version number not latest.

## 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

This will have some repeat information from the transcript section, but contains some additional information if a lambda were to invoked elsewhere in the function. To invoke a lambda is to call a lambda 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 a lambda. 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-1:9852261b-15e1-4d32-9070-fe7dcb89e4bf', - 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);

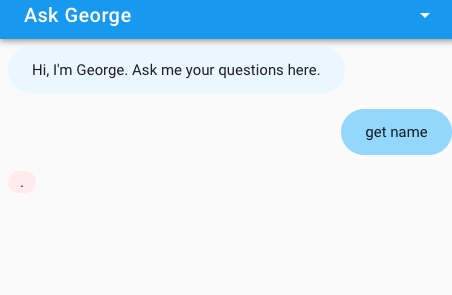
}

});

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.



**Thinking Bubble**

**User bubble**

**Bot Response 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.
   1. *Note that this cannot be done due to the size limit of invocation on AWS Lambda. The reason why this need to be done was 1. to update intent and escalation table and 2. to get the data from the backend service if needed, but now this can be done in every Lambda function we have currently.*
   2. *If you wish to add a separate Lambda function, simply add a escalation file within the function, and make sure you have write\_to\_intent\_table(intent\_request) and update\_transcript() functions.*
3. 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.
4. 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.
5. In dispatch, add an if statement directing to the correct function.
6. Create the intent’s fulfillment function under “Functions that control the bot’s behavior”.
7. 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>.

My other favourite links are here:

<https://devdocs.io>

<http://sk.kapsi.fi/interactive-jquery-tester.html>

<https://docs.aws.amazon.com/lex/latest/dg/lambda-input-response-format.html>

<https://purechat.com/support/javascript-api>

## 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 -> ID: tomcat, Password: tc4GBC17
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. Password: fhp4Psdm
4. cd /usr/share/apache-tomcat-7.0.75/
5. cd logs/
6. 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.**

NOTE: PURECHAT AND CHATBOT LOADER NEED TO BE IN THAT ORDER

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.

### Chatbot Off

If the chatbot is off and an operator is available, the pure chat window is displayed as pictured in figure 1. When no operator is available, there is a link to <https://www.georgebrown.ca/international/contact_us/>.

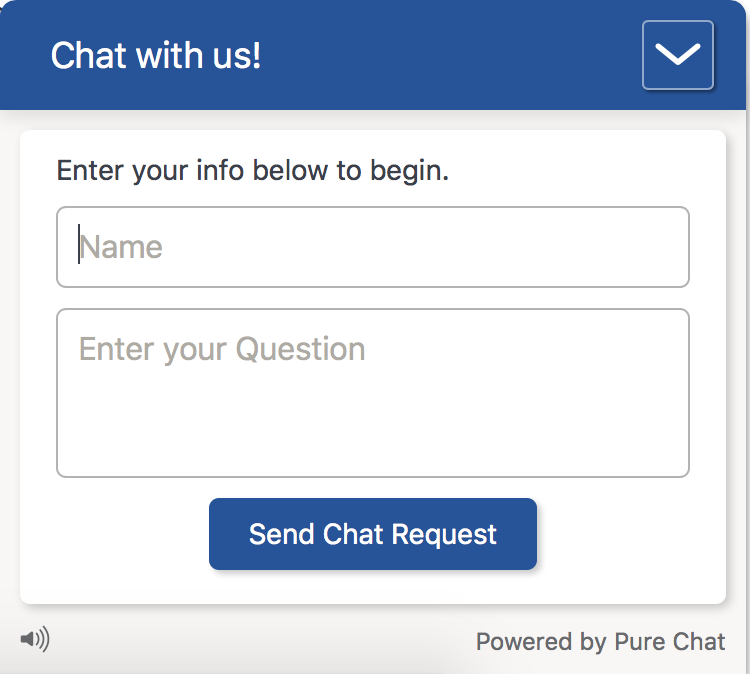


Figure : pure chat view when chatbot off and operator available

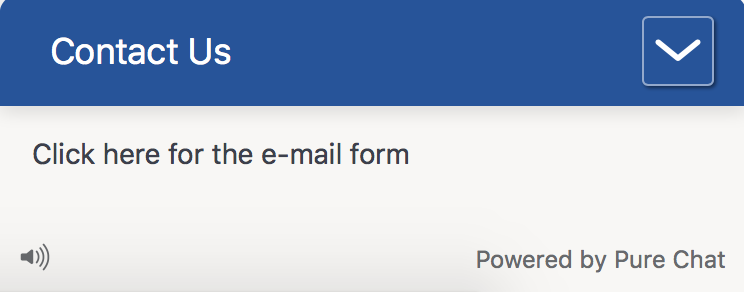


Figure : pure chat when chatbot off and no available operator

### Chatbot On

Regardless of whether or not an operator is available, the chatbot loads. If an operator is available, the display is set to none so that the chatbot is not visible. If there is no available operator, the pure chat is hidden using the display property. There is a cleaner way to hide the pure chat using the pure chat API (purechatApi.set(**'chatbox.unavailableBehavior'**, **'hidden**). However, as of July 2019, the pure chat API has a bug where it does not work. Their customer service team has informed me that they are working on it.

## 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.

# Chatbot Service

There are four important pages on chatbot service.

## Documentation

The documentation describes what all the API calls that the chatbot uses does.

## Transcripts

Transcripts is a responsive page that can be used to read the transcripts loaded into the chatbot database nightly. Users can search by user id (OAS user id) or the chat token. They can also search browse all the conversations. The symbols indicate certain conditions. The \* means that the user escalated the conversations to an email. The ! means there was a processing error. The X means there was a mismatch error.

When a transcript is selected, the user can easily switch the date in drop down search bar.

## Intent Monitoring

Intent monitoring has six sub-pages.

The main stats page displays four information boxes. The first box shows the popularity of each category. There are currently four categories (personal info, FAQ, escalation, and look up). Personal info is when an API call is used to get private information about the user. Look up is a dynamic answer that is public information and example would be getProgramDuration. The second box shows the top five most popular intents. The third box shows the total number of conversations. Currently, the date is yesterday because I assumed the nightly updating would always be successful. Finally, there is a box that displays the escalation frequencies and reasons.

Browse by date and browse by popular are very similar. The only difference being the way that the utterances are ordered. It displays a bunch of information about the utterances and allows the user to classify the intent as correct or incorrect. This is where mismatch errors are determined.

Intent data shows popularity of intents, the category and a description.

Escalation data is same as main stats but the information is bigger.

In search, a user can search by intent or utterance to select one intent. They can see all the utterances, descriptions, select conversations using utterances, and popularity, and classify utterances as incorrect.

## Errors Escalation

The errors escalation page displays all the errors. As a developer fixes the issues, they should be marked as resolved so they no longer appear as errors. The boxes do not contain a lot of space for errors because hopefully there should never be more than a few errors.

## Prompting to log-in page for private information

Use UserId as a Lex token to connect Lex to the chatbot service. Following link will be provided if the user has not been logged in yet, and this link will only work when the lexChatToken is provided: <https://dmzgrg01u.georgebrown.ca/gbcauth/login/index?lexChatToken=123>

lexChatToken will be used to ensure that the same user that has logged in is asking for his/her information.

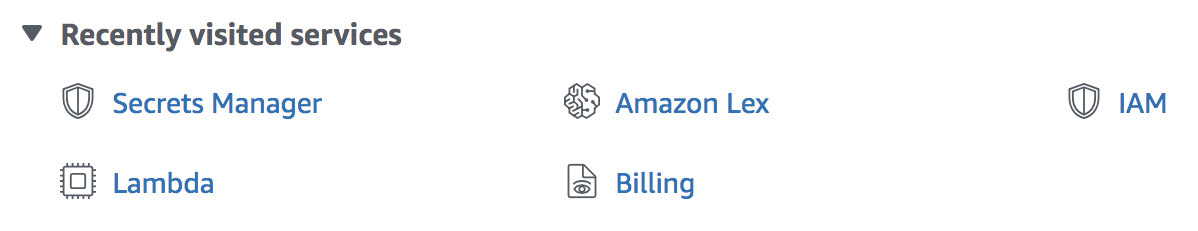
# Appendix A: AWS Chatbot Guide Version I

Sofia Fong for Intellisoft Jan - Apr 2019

Hello new co-op student! Here is (almost) everything you need to know about the current state of the chatbot project. We are trying to compare two chatbot frameworks: Microsoft Azure and AWS to determine which one is better for our purposes. Both of these services have a lot of different parts which work together to give security to your bot and create a better user experience. Let’s start by looking at AWS.

AWS

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



These are the main services that you will be using. Here is how they work:

**1. Amazon Lex**

Amazon Lex is the main chatbot service. The bot we are working on is the AskGeorgeBot. First, some keywords you need to know:

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

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

* Utterances – Spoken or typed phrases that invoke your intent

o 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)  
  o 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

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

**IMPORTANT**: Before you start working on the AskGeorgeBot, you might want to first open up the BookTripLambda bot to get an idea of the features available. This bot is a good example which uses all the different features of Lex and there is even documentation for you to read if you get confused. It is also very helpful to try out the bot yourself to see how it works. To do this, click the build button. Once the bot is built, a tab to test the chatbot will appear on the right side of the screen.

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 – 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

The editor is the most important part of this bot. If you look at the BookTripLambda bot, a lot of these things will be very easy to understand. Anyways, after you create a new intent, you need to specify some (or all) of these things:

* Sample utterances
  + Examples of what the user might type in order to invoke this intent
  + If you look at the BookHotel intent (in the BookTripLambdabot) you can even do things like {slotName} to specify that people might directly type in the information you want. This will make your bot seem smarter!
* 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
  + You can specify what question the bot will ask to get this information
  + 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
  + For things like Q&A you can put the response here
  + Use HTML for formatting

The AskGeorgeBot is very simple right now. I added around 50 intents that are question and answer (so it only uses utterance and response, nothing fancy). There is also a getStatus intent which uses the AskGeorgeBot lambda function to return data from our systems using our

chatbot web service (ChatbotService). If any of the above is confusing or you want a more in- depth understanding of everything, explore the complete documentation for Amazon Lex.

**2. Lambda**

This is the fancy Lambda function that was mentioned above. Most of the work will be done here! The purpose of these functions (for a chatbot at least) are to:

* 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 pickup 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.

Lambda functions contain code that can be run before and after the intent is called. This means you can basically make your bot do ANYTHING you want! They’re great!!

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)

**3. 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.

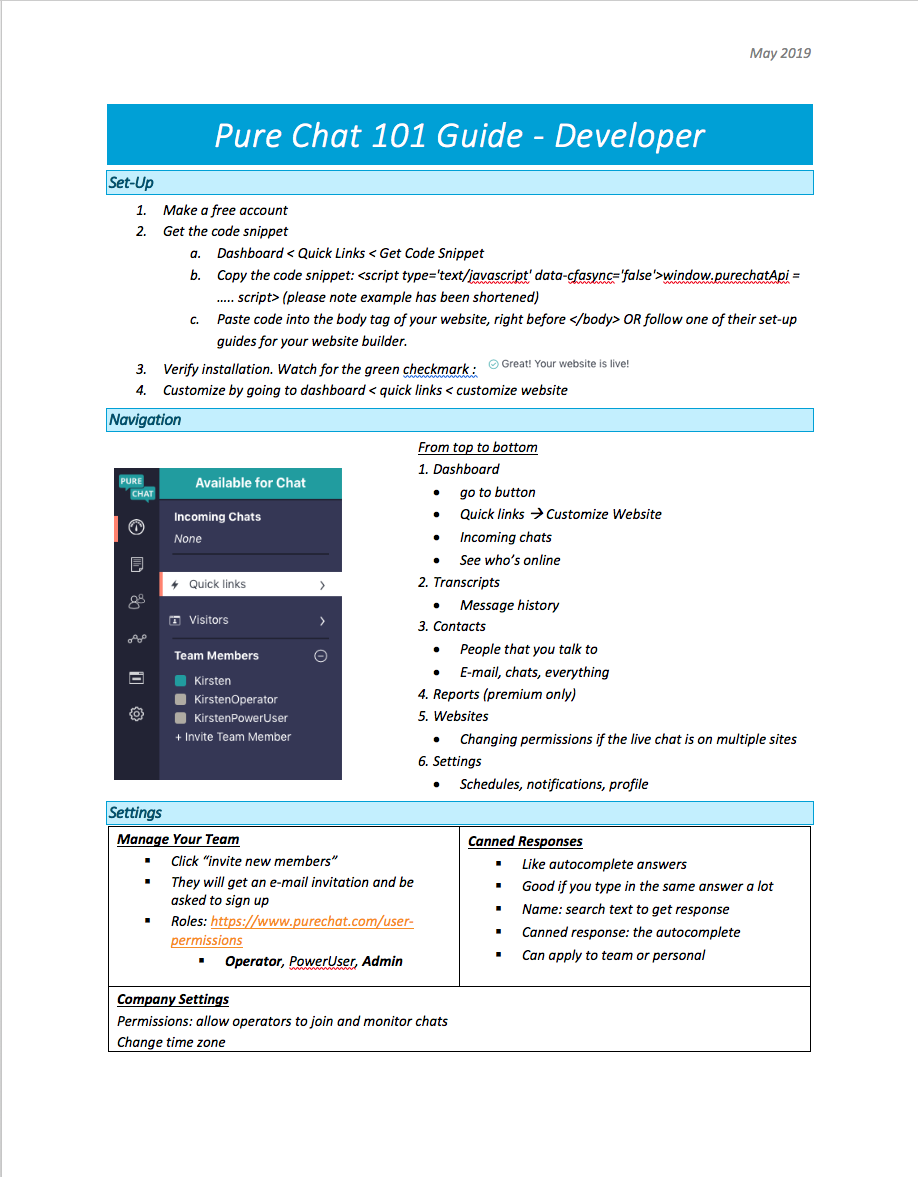
**4. IAM**

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.

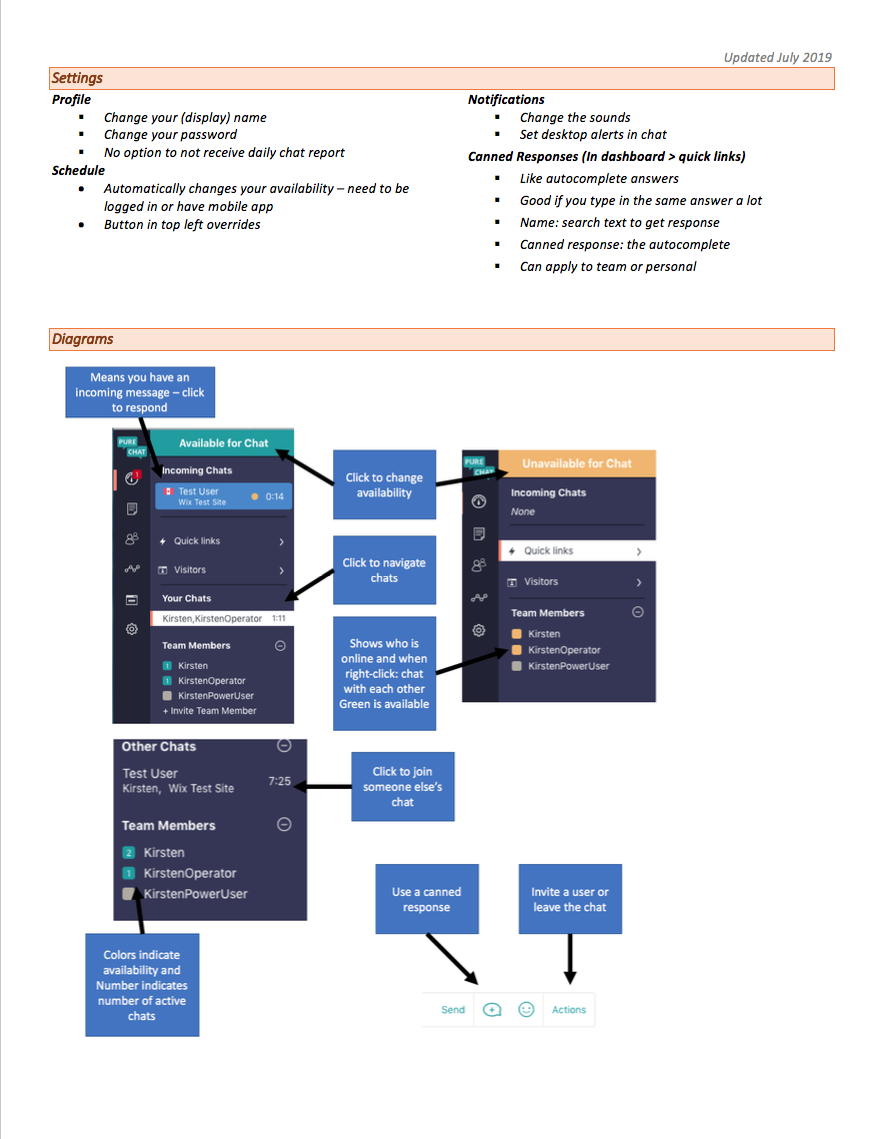
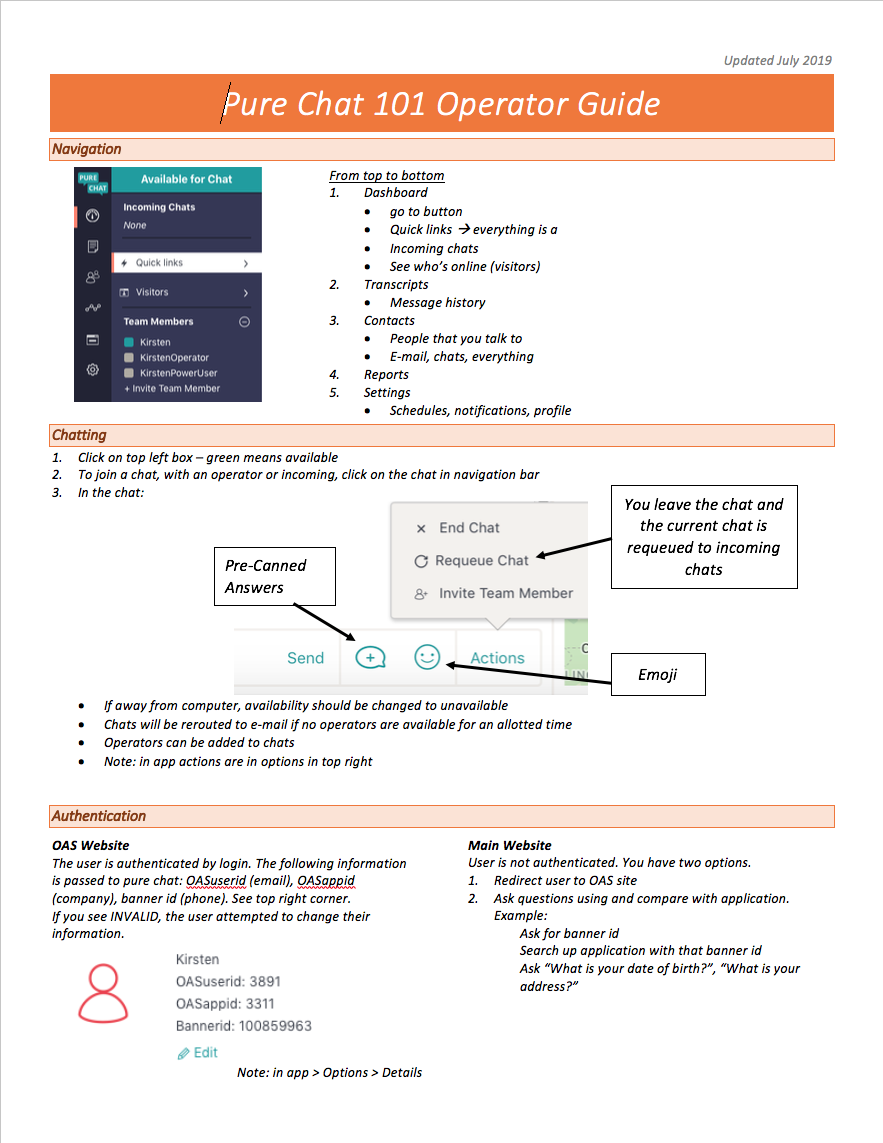
If you look at the existing roles, you will see a TestChatbotRole. This role has permission to both lambda and secrets manager. Right now, the lambda function AskGeorgeBot has this role which means it can access information stored in the secrets manager. You can attach new policies which give it more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

**5. Billing**

Not really a service you need to use to create the bot but I think it is useful to know anyways. Billing basically shows you all the information relating to the costs by clicking Launch Cost Explorer > Explore > Cost and Usage. This data can be exported to excel for easy viewing. This service can also alert you once you are near your monthly budget (have to this set up). If you know how much your monthly cost need to be, you can set up a budget with alerts by clicking Budgets > Create budget.

Appendix B: Pure Chat Guides





# Appendix C: Notes for Future Developers

## Support Request August 23, 2019

There is a major flaw with Amazon Lex’s slot types. Open ended slot types do not exist. This creates two problems.

1. The email sequence ‘emailMemo’ slot

The email sequence starts because of ‘exit’ utterance. Here is a sample conversation to replicate the problem.

User: What is sin

Bot: You can visit the closest Service Canada, in person, with your study permit, work permit (if applicable) and passport to apply for your SIN.

User: exit

Bot: I'm sorry that I couldn't help you today. I can e-mail someone to help you. Would you like to e-mail an admissions officer?

User: yes

Bot: Every bubble after [what is sin] will be e-mailed to an admission officer to further assist you. Is there anything else you would like to add?

User: yes

Bot: type your additional information:

If you type ‘hi’ you’ll replicate the issue. If you type ‘klshrkjdgf’ you’ll get the proper response.

The reason that this happens is that Lex always tries to elicit an intent before a slot. If the input matches an utterance for a slot, it will try to execute that intent. This isn’t a problem for any other slot because I see no reason why someone would ask a question in any other slot but emailMemo.

Update (Sept 11th 2019) :

This is handled by making the users directly email to the admissions officer.

Lex will give out the email address, and the users can click the link to add any information they want.

## Concerns with Current Implementation August 2019

As of August 2019, there is some concerns that I have with how the code is set up and some of the sequences are set up.

In the changing application sequence, there isn’t a lot of clarity about how to change the application later. You wouldn’t want to constantly output how to change the application after every application request. It should just be done the first time.

All the session attributes are stored as strings. I had problems storing them as other types but it’s weird in the code. Other Boolean variables are unnecessarily strings.

I don’t like Amazon types. It seems to miss something and makes the user continuously enter their input. I wish there was just a “string” type. There’s a work around see above.

In the login sequence, when the login window is on the screen, you shouldn’t be able to type into the user text box.

There are way too many session attributes, maybe double up on them?

When is this used??

Dmzgrg01u.georgebrown.ca

root: grg3Bto