Cps 479 Computer Science Seminar

B. J. U. Quotes Alexa Skill

Project Documentation

Karyn Wetzel
11-15-2018

Table of Contents

Introduction

This document will describe the B. J. U. Quotes Alexa Skill App. Part of the purpose of this documentation will be to explain the different components of Alexa apps and what they do, as well as give some insight into how one makes an Alexa App. This paper will discuss the different components of the project and their role, how to test the app, its usage, a bug report, and planned future features. The Lambda Function Code will be provided in the Appendix section.

Components

The skill app is made up of two parts: the Alexa Skill and a Lambda Function.

The B. J. U. Quotes Alexa Skill

When a developer creates an Alexa skill, they are specifying configurations for how Alexa should route requests to an endpoint and what information should show up in the Amazon Alexa App about the skill. When a user starts a session with the skill, this configuration will be used by Alexa to determine what information to listen for and where to send the information too.

The B.J.U. Quotes skill is the Alexa skill for this project. It was created in the Alexa developer console.

This console walks developers step by step through the process of setting up an Alexa skill with helpful hints and links to relevant sections of documentation.

The two categories of configurations modified for this project were the Interaction Model and the Endpoint. The Interaction Model specifies what information a user's interaction should contain. This is represented by:

- Invocation – how the user starts a session with the skill
- Intents – the different actions a user can take with the skill
- Slot Types – what parameters of information a user can give to the skill

The Endpoint is how the developer specifies where Alexa should send the JSON post request that represents the user's request. The only accepted types of endpoints are AWS Lambda Function numbers or HTTPS URLs.

Invocation Name:

b. j. u. quotes

Intents:

Built In & Required Intents:

AMAZON.CancelIntent – a way for users to cancel an intent

AMAZON.HelpIntent – a way for users to ask for help with the skill usage

AMAZON.StopIntent – a way for users to end the session with the B. J. U. Quotes Skill

AMAZON.NavigateHomeIntent – a way for users to get back to the device home screen and end the session

Built in Intents:

AMAZON.FallbackIntent – the default action if the user doesn't specify an intent

Custom Intents:

GetNewQuoteIntent – a way for users to ask for a B. J. U. Quote. Uses the person intent slot to let the user specify a speaker to quote

Intent Slots

Person – uses the AMAZON.FirstName slot type because there is no full or last name type. This represents the last name of a speaker to quote.

Endpoint

This app uses an AWS Lambda ARN for an endpoint, because it is the recommended type. The Lambda's id is arn:aws:lambda:us-east-284979954717:function:PythonBJUQuotes 1:

The PythonBJUQuotes Lambda Function

The AWS Lambda Function is the part of the project that does the logic of the skill. When a user interacts with the B. J. U. Quotes Skill, Alexa will send a JSON POST request to this function. The lambda function for this app was developed in the Lambda Management Console.

The function has a main handler that determines which intent is being invoked, then passes the information to the specified intent's handler function.

```
# --------------- Main handler ------------------

def lambda_handler(event, context):
    """ Route the incoming request based on type (LaunchRequest, IntentRequest,
    etc.) The JSON body of the request is provided in the event parameter.
    """
    print("event.session.application.applicationId=" +
          event['session']['application']['applicationId'])

    """
    Uncomment this if statement and populate with your skill's application ID t
    prevent someone else from configuring a skill that sends requests to this
    function.
    """
    # if (event['session']['application']['applicationId'] !=
    #         "amzn1.echo-sdk-ams.app.[unique-value-here]"):
    #     raise ValueError("Invalid Application ID")

    if event['session']['new']:
        on_session_started({'requestId': event['request']['requestId']},
                           event['session'])

    if event['request']['type'] == "LaunchRequest":
        return on_launch(event['request'], event['session'])
    elif event['request']['type'] == "IntentRequest":
        return on_intent(event['request'], event['session'])
    elif event['request']['type'] == "SessionEndedRequest":
        return on_session_ended(event['request'], event['session'])
```

```
def on_intent(intent_request, session):
    """ Called when the user specifies an intent for this skill """

    print("on_intent requestId=" + intent_request['requestId'] +
          ", sessionId=" + session['sessionId'])

    intent = intent_request['intent']
    intent_name = intent_request['intent']['name']

    # Dispatch to your skill's intent handlers
    if intent_name == "GetNewQuoteIntent":
        return get_quote(intent, session)
    if intent_name == "AMAZON.HelpIntent":
        return ask_for_help(intent, session)
    if intent_name == "AMAZON.CancelIntent":
        return stop(intent, session)
    if intent_name == "AMAZON.StopIntent":
        return stop(intent, session)
    if intent_name == "AMAZON.FallbackIntent":
        return get_quote(intent, session)
    else:
        raise ValueError("Invalid intent")
```

Each handler function builds a JSON Response object with the version number, session attributes, and the response. The response includes what Alexa should say, what the card in the Alexa App on the user's phone should say, what Alexa should say if she has to re-prompt the user and whether the session should end.

```
# --------------- Helpers that build all of the responses ----------------------

def build_speechlet_response(title, output, reprompt_text, should_end_session):
    return {
        'outputSpeech': {
            'type': 'PlainText',
            'text': output
        },
        'card': {
            'type': 'Simple',
            'title': " B.J.U. Quote",
            'content': output
        },
        'reprompt': {
            'outputSpeech': {
                'type': 'PlainText',
                'text': reprompt_text
            }
        },
        'shouldEndSession': should_end_session
    }


def build_response(session_attributes, speechlet_response):
    return {
        'version': '1.0',
        'sessionAttributes': session_attributes,
        'response': speechlet_response
    }
```

Since most of the intents are simple built in help and stop type intents, this document will only discuss the handler of the custom `GetNewQuoteIntent`. The other handlers can be found in the appendix section at the end of this document and are heavily commented.

The quotes for this skill are represented in a dictionary, with the speaker's full name as the key and a list of strings as the value.

To account for Alexa's imperfect voice recognition skills and the difficulty of recognizing last names, there is an `alternative_names` dictionary that has the full name as the key and a list of strings of similar names as the value. For example, at the key `"Dr. Stephen Schaub"` there is a list that includes the string `"job"` since this is a likely mistake Alexa will make when trying to recognize `"Schaub"`. There are also other parts of their name such as `"Stephen"` in the list in case the user says the whole name but Alexa captures just the first name.

So when the user tries to get a quote, if they specify a speaker, the `get_speaker` function tries to recognize the speaker name, if not it is noted that the requested speaker couldn't be found so that Alexa can apologize for it, and instead return a random speaker, just like it would have if no speaker was requested.

Then the function selects a random quote from that speaker's list of quotes, and it is formatted to return to the user.

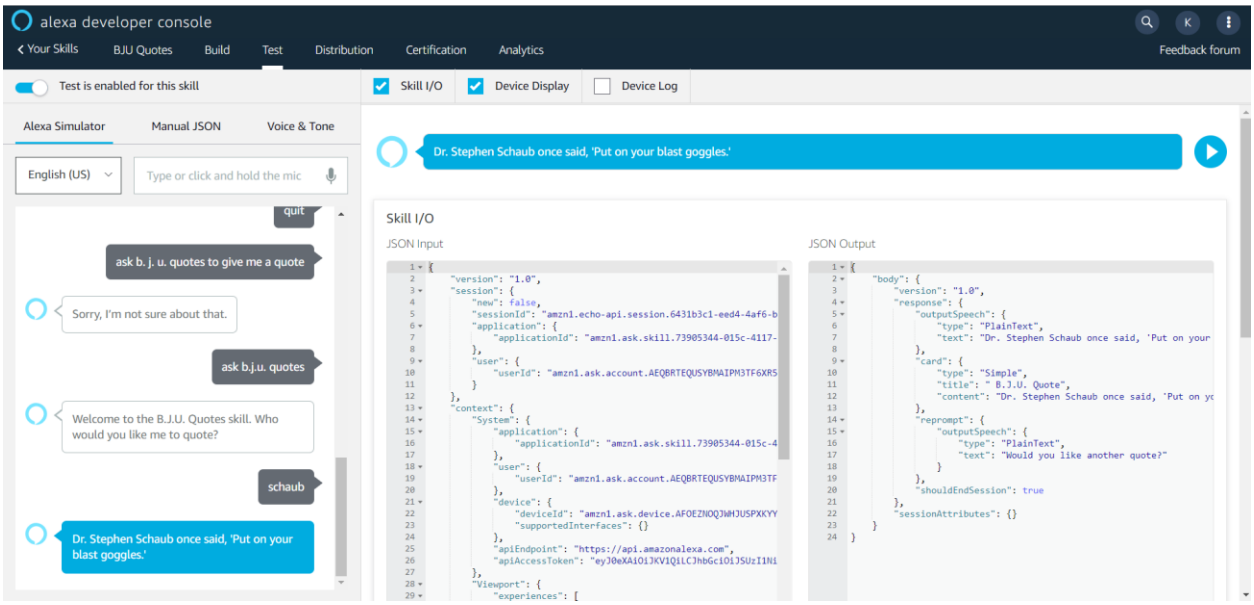
```
def get_quote(intent, session):   # TODO change function name
    session_attributes = {}
    reprompt_text = "Would you like another quote?"
    try:
    # get quote
        person = None   # TODO how to get intent slot
        try:
            person = intent['slots']['person']['value']
        except Exception as e:
            print("no particular person requested")
        print("person: ", person)
        speaker, found_match = get_speaker(person)
        quote = get_ran_arr_val(data[speaker])
        quote_intro = f"{speaker} once said,"
        if not found_match:
            quote_intro = f"Sorry, we couldn't find a quote by {person}, but " + quote_intro
        speech_output = f"{quote_intro} '{quote}'"
        should_end_session = True

    except Exception as e:
        print("something went wrong. " + str(e))
        print(e)
        speech_output = "I'm sorry, There was an error while proccessing your request."
        should_end_session = False

    # Setting reprompt_text to None signifies that we do not want to reprompt
    # the user. If the user does not respond or says something that is not
    # understood, the session will end.
    return build_response(session_attributes, build_speechlet_response(
            intent['name'], speech_output, reprompt_text, should_end_session))
```

Testing

The Alexa Developer Console's Test tab is the best way to test the project. You can test by entering the text to use the skill or you can click and hold the mic icon instead to test how well the project handles difficulties with Alexa's voice recognition.



Simply type what you want your users to be able to say, only you don't have to include the "Hey Alexa, " part at the beginning.

Usage

First, get Alexa's attention by saying her name. Next, tell her you want to use the B. J. U. Quotes skill by saying "tell b. j. u. quotes" or "open b. j. u. quotes." Finally if you know what intent you want to use, say something that indicates that intent. For example, if you know you want a quote from Dr. Jim Knisely, add "for a knisely quote" or "to give me a knisely quote." If you don't care who said the quote you can just add "for a quote."

So all together now, you would say a sentence like:

- "Alexa, open b. j. u. quotes for a knisely quote"
- "Alexa, tell b. j. u. quotes to give me a knisely quote"
- "Alexa, open b. j. u. quotes for a quote"

If you don't know what intent you want to use, you can just say "Alexa, open b. j. u. quotes" and Alexa will say "Welcome to the B.J.U. Quotes skill. Who would you like me to quote?" If you still are confused, at any point you can say "help" and Alexa will inform you that "You can say tell me a B.J.U. quote, or, you can say exit... What can I help you with?"

Bug Report

No known bugs except name recognition is not very robust.

Future Features

I would like to add more quotes and speakers in the future.

Appendices

B. J. U. Quotes Skill Interaction Model in JSON Format

```
{
  "interactionModel": {
    "languageModel": {
      "invocationName": "b. j. u. quotes",
      "intents": [
        {
          "name": "AMAZON.CancelIntent",
          "samples": []
        },
        {
          "name": "AMAZON.HelpIntent",
          "samples": []
        },
        {
          "name": "AMAZON.StopIntent",
          "samples": []
        },
        {
          "name": "AMAZON.NavigateHomeIntent",
          "samples": []
        },
        {
          "name": "AMAZON.FallbackIntent",
          "samples": []
        },
        {
          "name": "GetNewQuoteIntent",
          "slots": [
            {
              "name": "person",
              "type": "AMAZON.US_FIRST_NAME"
            }
          ],
          "samples": [
            "{person}",
            "a b.j.u. quote",
            "tell me a b.j.u. quote",
            "give me a b.j.u. quote",
            "give me a {person} quote",
            "give me a quote",
            "tell me a {person} quote",
            "tell me a quote",
            "a {person} quote",
            "a quote",
            "tell me something",
            "give me something"
          ]
        }
      ],
      "types": []
    }
  }
}
```

lambda_function.py

```
import json
"""
Alexa Skill that Gives Quotes from people at Bob Jones University

For additional samples, visit the Alexa Skills Kit Getting Started guide at
http://amzn.to/1LGWsLG
"""

import os
import atexit
import random


# --------------- Helpers that build all of the responses ----------------------

def build_speechlet_response(title, output, reprompt_text, should_end_session):
    return {
        'outputSpeech': {
            'type': 'PlainText',
            'text': output
        },
        'card': {
            'type': 'Simple',
            'title': " B.J.U. Quote",
            'content': output
        },
        'reprompt': {
            'outputSpeech': {
                'type': 'PlainText',
                'text': reprompt_text
            }
        },
        'shouldEndSession': should_end_session
    }


def build_response(session_attributes, speechlet_response):
    return {
        'version': '1.0',
        'sessionAttributes': session_attributes,
        'response': speechlet_response
    }


# --------------- Functions that control the skill's behavior ------------------

def get_welcome_response():
    """ If we wanted to initialize the session to have some attributes we could
    add those here
    """

    session_attributes = {}
    card_title = "Welcome"
    speech_output = "Welcome to the B.J.U. Quotes skill. " \
                    "Who would you like me to quote?"
    # If the user either does not reply to the welcome message or says something
    # that is not understood, they will be prompted again with this text.
```

```
    reprompt_text = "Is there anyone you wan't me to quote specifically?"
    should_end_session = False
    return build_response(session_attributes, build_speechlet_response(
        card_title, speech_output, reprompt_text, should_end_session))


def handle_session_end_request():
    card_title = "Session Ended"
    speech_output = "Thank you for trying the B.J.U. Quotes skill. " \
                    "Have a nice day! "
    # Setting this to true ends the session and exits the skill.
    should_end_session = True
    return build_response({}, build_speechlet_response(
        card_title, speech_output, None, should_end_session))

def ask_for_help(intent, session):
    session_attributes = {}
    reprompt_text = "Would you like me to repeat that?"

    try:
        print("Helping . . .")
        speech_output = 'You can say tell me a B.J.U. quote, or, you can say exit... What can I help you with?'
    except Exception as e:
        print("something went wrong. " + e.message)
        print(e)
        speech_output = "I'm sorry, There was an error while proccessing your request."
    should_end_session = False

    # Setting reprompt_text to None signifies that we do not want to reprompt
    # the user. If the user does not respond or says something that is not
    # understood, the session will end.
    return build_response(session_attributes, build_speechlet_response(
        intent['name'], speech_output, reprompt_text, should_end_session))

def stop(intent, session):
    session_attributes = {}
    reprompt_text = "Goodbye!"

    try:
        print("Stopping . . .")
        speech_output = 'Goodbye!'
    except Exception as e:
        print("something went wrong. " + e.message)
        print(e)
        speech_output = "I'm sorry, There was an error while proccessing your request."
    should_end_session = True

    # Setting reprompt_text to None signifies that we do not want to reprompt
    # the user. If the user does not respond or says something that is not
    # understood, the session will end.
    return build_response(session_attributes, build_speechlet_response(
        intent['name'], speech_output, reprompt_text, should_end_session))

# quotes Data
data = {
    'Dr. Bob Jones Sr.': [
        "It is a sin to do less than your best.",
        "The door to the room of success swings on the hinges of opposition.",
```

```
        'The two biggest little words in the English language are the two little words "do right."',
        "It is one thing to know there is a God; it's another thing to know the God that is.",
        "A man is a fool who leans on the arm of flesh when he can be supported by the arm of Omnipotence.",
        "What you love and what you hate reveal what you are.",
        "Jesus never taught men how to make a living; He taught men how to live.",
        "Do not ask God to give you a light burden; ask Him to give you strong shoulders to carry a heavy burden.",
        "It is better to die for something than to live for nothing.",
        "A Christian does good deeds, but just doing good deeds does not make a man a Christian.",
        "Your character is what God knows you to be; your reputation is what men think you are.",
        "You and God make a majority in your community.",
        "It is never right to do wrong in order to get a chance to do right.",
        "Jesus said that He would be in the midst of two or three gathered in His name, but this does not mean that our Lord does not like to have a larger crowd.",
        "It is no disgrace to fail; it is a disgrace to do less than your best to keep from failing.",
        'The religions of the world say, "do and live." The religion of the Bible says, "live and do."',
        "God will not do for you what He has given you strength to do for yourself.",
        'Dying men have said, "I am sorry I have been an atheist, an infidel, an agnostic, a skeptic, or a sinner"; but no man ever said with his last breath, "I am sorry I have lived a Christian life."',
        "The drunkard in the ditch has gone to the dogs. According to the Bible, the self-righteous man who thinks he doesn't need God has gone to the Devil.",
    ],
    'Dr. Stephen Schaub': [
        'Put on your blast goggles.',
    ],
    'Dr. Jim Knicely': [
        "If it was easy, it wouldn't be so hard",
    ]
}

alternative_names = {
    'Dr. Bob Jones Sr.': ['doctor bob', 'doctor bob jones senior', 'bob jones', 'jones', 'bob'],
    'Dr. Stephen Schaub':  ['doctor schaub', 'doctor stephen schaub', 'stephen schaub', 'schaub', 'shaub', 'shab', 'shob', 'job'],
    'Dr. Jim Knicely': ['doctor knisely', 'doctor jim knisely', 'jim knisely', 'knisely', 'nicely']
}

no_names = [None, 'None', '', 'no', 'no one', 'no thanks', 'anyone', 'i do not care', "i don't care"]

def get_ran_arr_val(arr):
    index = random.randint(0, len(arr) - 1)
    return arr[index]

def get_speaker(person):
    """
    enables more name options
    indicates if we need to apologize for not finding the requested speaker
    """
    found_requested = True
    if person in no_names:
        return get_ran_arr_val(list(data.keys())), found_requested
    for key, arr in alternative_names.items():
        if person in arr:
            return key, found_requested
    found_requested = False
    return get_ran_arr_val(list(data.keys())), found_requested

def get_quote(intent, session):   # TODO change function name
    session_attributes = {}
    reprompt_text = "Would you like another quote?"
```

```
    try:
    # get quote
        person = None   # TODO how to get intent slot
        try:
            person = intent['slots']['person']['value']
        except Exception as e:
            print("no particular person requested")
        print("person: ", person)
        speaker, found_match = get_speaker(person)
        quote = get_ran_arr_val(data[speaker])
        quote_intro = f"{speaker} once said,"
        if not found_match:
            quote_intro = f"Sorry, we couldn't find a quote by {person}, but " + quote_intro
        speech_output = f"{quote_intro} '{quote}'"
        should_end_session = True

    except Exception as e:
        print("something went wrong. " + str(e))
        print(e)
        speech_output = "I'm sorry, There was an error while proccessing your request."
        should_end_session = False

    # Setting reprompt_text to None signifies that we do not want to reprompt
    # the user. If the user does not respond or says something that is not
    # understood, the session will end.
    return build_response(session_attributes, build_speechlet_response(
            intent['name'], speech_output, reprompt_text, should_end_session))


# --------------- Events ------------------

def on_session_started(session_started_request, session):
    """ Called when the session starts """

    print("on_session_started requestId=" + session_started_request['requestId']
          + ", sessionId=" + session['sessionId'])


def on_launch(launch_request, session):
    """ Called when the user launches the skill without specifying what they
    want
    """

    print("on_launch requestId=" + launch_request['requestId'] +
          ", sessionId=" + session['sessionId'])
    # Dispatch to your skill's launch
    return get_welcome_response()


def on_intent(intent_request, session):
    """ Called when the user specifies an intent for this skill """

    print("on_intent requestId=" + intent_request['requestId'] +
          ", sessionId=" + session['sessionId'])

    intent = intent_request['intent']
    intent_name = intent_request['intent']['name']
```

```
    # Dispatch to your skill's intent handlers
    if intent_name == "GetNewQuoteIntent":
        return get_quote(intent, session)
    if intent_name == "AMAZON.HelpIntent":
        return ask_for_help(intent, session)
    if intent_name == "AMAZON.CancelIntent":
        return stop(intent, session)
    if intent_name == "AMAZON.StopIntent":
        return stop(intent, session)
    if intent_name == "AMAZON.FallbackIntent":
        return get_quote(intent, session)
    else:
        raise ValueError("Invalid intent")


def on_session_ended(session_ended_request, session):
    """ Called when the user ends the session.

    Is not called when the skill returns should_end_session=true
    """
    print("on_session_ended requestId=" + session_ended_request['requestId'] +
          ", sessionId=" + session['sessionId'])
    # add cleanup logic here


# --------------- Main handler ------------------

def lambda_handler(event, context):
    """ Route the incoming request based on type (LaunchRequest, IntentRequest,
    etc.) The JSON body of the request is provided in the event parameter.
    """
    print("event.session.application.applicationId=" +
          event['session']['application']['applicationId'])

    """
    Uncomment this if statement and populate with your skill's application ID to
    prevent someone else from configuring a skill that sends requests to this
    function.
    """
    # if (event['session']['application']['applicationId'] !=
    #         "amzn1.echo-sdk-ams.app.[unique-value-here]"):
    #     raise ValueError("Invalid Application ID")

    if event['session']['new']:
        on_session_started({'requestId': event['request']['requestId']},
                           event['session'])

    if event['request']['type'] == "LaunchRequest":
        return on_launch(event['request'], event['session'])
    elif event['request']['type'] == "IntentRequest":
        return on_intent(event['request'], event['session'])
    elif event['request']['type'] == "SessionEndedRequest":
        return on_session_ended(event['request'], event['session'])
```