Flashcard Application Powered by Alexa

CAPSTONE PROPOSAL

ADVISOR

Dr. Curry Guinn

COMMITTEE MEMBERS

Dr. Judith Gebauer

Dr. Ellie Ebrahimi

PREPARED BY

Kavyashree Kushnoor

University of North Carolina Wilmington

Apr 24, 2019

Table of Contents

Abstract

This paper discusses the ways in which Alexa skills can be developed and enhanced for customizable utility. The aim of this project is to enable an end-user to develop flashcards on a website. This data is captured and sent to an Amazon Web Services (AWS) powered database management system called DynamoDB. Alexa then reads the data from the table and gives the user the ability to practice those customized flashcards using Alexa's voice interaction model. The user is able to get audio-visual aid in memorization of the flashcards using the web interface and Alexa's voice user interface.

Motivation

The motivation of the project is to enrich the language learning process of a student using an app powered by Alexa (also called as Alexa Skill or Alexa App). Since a lot of the second language speaking and pronunciation skills rely on memory of the new words learned, this skill helps in creating and retaining them using three different senses. The use of this project gives learners an immersive learning experience for memory retention using sensory input; via,

1. Visual Memory - In the form of visual display of flashcards on the web browser
2. Auditory Memory - In the form of Alexa's voice interface which enhances practice by simultaneously playing the audio questions and answers in the background
3. Muscular Memory - In the form of speech practice where Alexa takes user input and checks whether the answer is correct or not

Since there is currently no application which enables a non-technical user who has no programming experience create customized flashcards for Alexa based practice, a need for this web application was noticed. A web interface design would allow for easy customization by a non-programming user. The user does not need to visit the developer website or know any coding. They can enter flashcards/ quiz data into a table-like sheet on the website interface.

Alexa is the choice for this application because it is a popular hardware device that is found in many customers' homes. Customizing what Alexa says using Alexa Skills Kit is convenient and user-friendly and can be made compatible with an external website from the developer's perspective.

Introduction

Alexa's developer console provides the ability to customize skills. It provides a platform to design the front end of Alexa's speech interface. It is also the place where the skill can be tested for its usability. The backend, AWS Lambda is where the code resides. AWS Lambda is a serverless computing platform for event-driven processes which stores and executes code. It tells Alexa when and how to respond to a user's utterances. AWS lambda can be written in Java, Go, PowerShell, Node.js, C#, Python or Ruby. The AWS infrastructure has many features like EC2 instances, storage volumes, databases, monitoring configurations, networking components or packaged AWS Marketplace products, which can be used for the development of web-based, mobile-based or Alexa based Applications. For this proposed capstone project, AWS lambda is programmed in node.js and other features of AWS like DynamoDB, API and Alexa-Development Consode are used. These features are further discussed in detail in Section 'Architecture Description'. These tools are chosen because of their popularity among Alexa developers. Most discussion forums, blogs and quality tutorials and textbooks use these tools.

Objective

1. Create an application powered by Alexa which enables a user to practice Flashcards

2. Enable users to create flashcards on a web-browser and practice the customized set using Alexa's Voice Interface.

3. Make Alexa flashcards accessible to one or more users.

Progress

Developed independent Alexa skills, which interact with users by invoking the apps through voice commands.

1. Verbiage
    - Uses Alexa Skill and AWS-lambda function
    - Interacts with user for quiz on verb (past and past participle). The collection of words (verbs) are in lambda function.
2. Atomic Flash
    - Uses Alexa Skill, AWS lambda functions, DynamoDB and a website.
    - This app displays flashcard of chemical elements on user commands. User can ask for a single flashcard by invoking the Alexa-skill with atomic numbers or can ask for all flashcards.

Current Architecture Model for ‘Verbiage’ App

Figure 1: Architecture diagram of ‘ Verbiage’ App

Current Architecture Model for ‘Atomic Flash’ App

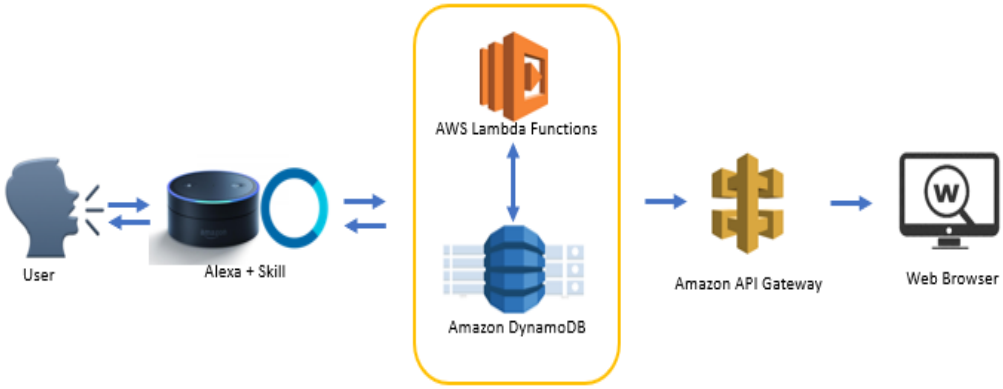

Figure 2: Architecture diagram of ‘Atomic Flash’ App

Architecture Description

Amazon Web Services provides a platform for cloud computing. It has a wide array of services to choose from. An effective architecture can be built by considering the cost, scalability, usage requirements, training resources and popularity among the developer community. This project uses 4 main Amazon Web Services features and one web-server:

1. Alexa Skills Kit

   Alexa Skills Kit is the front end of the Alexa voice service. The developer can control Alexa's responses and invocation from ASK. It configures the back-end by connecting **endpoints with servers**, enables in-built support with other echo devices. It also allows **testing** before skill deployment.

   It is used primarily to build an interaction model. Each time a user has a dialog exchange with Alexa, it is done based on an **Intent** which is stored in the skill's developer account. For each intent created, a function in the back-end is triggered which tells Alexa what to say, listen and respond. It is activated by **Utterances** which a user can say to use the intent.

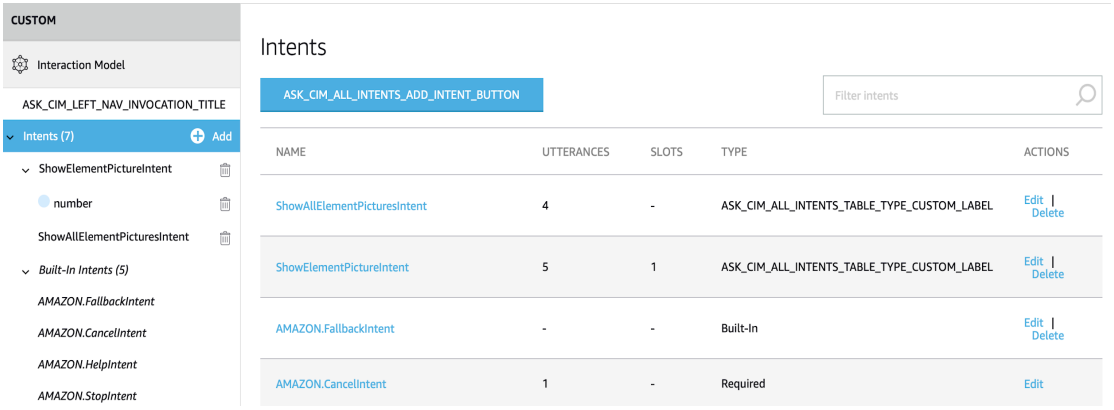

Figure 3: Example of Intents/handlers in Alexa-Skill

Intents:
The functions within a skill that a user can call.
Example: say hello, add to favorite, delete from favorite, next story, etc.

General Model:
Alexa, <invocation phrase> **<invocation name>** --- <connecting word> <*some action*>

Alexa,	tell	Verbiage	to	<practice Simple-past>

Model:	Example:	Where connecting word is:
<some action> **<connecting word>** <invocation name>	*give me <action>* **using** <app>	by, from, in, using, with
Ask <invocation name> **<connecting word>***<some action>*	**Ask** <app> **about** *<action>*	*to, about, for, if, whether*
Ask <invocation name> *<some action>*	**Ask** <app> *the <action> report today*	to, that

Invocation phrases:

- Ask
- tell
- search
- open
- launch
- start
- resume
- run
- load
- begin
- use
- Run
- Talk to
- play

Connecting word:

- From
- Using
- For
- about
- if
- Whether
- To

2. AWS Lambda

AWS Lambda is a serverless computing platform for event-driven processes. It stores and executes code. It also manages the computing resources that are required by the functions in the code. For computing, AWS Lambda has been used as a platform. It is a serverless form of computing without provisioning servers or managing them. It incurs low costs since the developer is billed only for the time while the code is running.

2.1 Verbiage

The index.js file of Verbiage's AWS-Lambda (at AWS.AMAZON.COM) consists of a bank of verbs (past and past participle) and intents or handlers are defined in Alexa-Skill at developer.amazon.com

The Intents/handlers are :

- LaunchRequest: function()

  On calling this intent, Alexa speaks the welcome message that is stored in the lambda function. The syntax for it is:

  ```
  this.response.speak('Welcome to Flashcards. In this session, do you want to test' + ' your knowledge in Simple-past, Past-participle or both?').listen('What would you like to practice?');

  this.emit(':responseReady');
  ```

- VerbIntent: function()

  On calling this intent, the corresponding handler in lambda runs. It selects the choice of verb category to practice from. The syntax for it is:

  ```
  this.attributes['Verb''] =
  this.event.request.intent.slots.verb.value;
  var verb = this.attributes['verb''];
  this.response.speak('Okay, I will ask you some questions about ' + verb + '. Here is your first question. ' + AskQuestion(this.attributes)).listen(AskQuestion(this.attributes));
  this.emit(':responseReady');
  ```

- AnswerIntent: function()

This intent waits on the user's answer. It matches it with the slot value stored in ASK to check accuracy. Resulting score is displayed based on answer.

The JSON request containing our slot looks like this:

```
"intent": {
  "name": "VerbIntent",
  "slots": {
    "verb": {
      "name": "verb",
      "value": "simple-past"
    }
  }
}
```

You can access the input using the following syntax:

```
this.event.request.intent.slots.yourSlotName.value
```

In the JSON example above, we can access our user's language response value with the following:

```
this.event.request.intent.slots.verb.value
const myVariableExample=this.event.request.intent.slots.yourSlotName.value;
```

- AMAZON.StopIntent: function()

  This is a built-in function in Amazon. The syntax for it is:

  ```
  this.response.speak('Sure, catch you later.');

  this.emit(':responseReady');
  ```

- AMAZON.CancelIntent: function()

  This is a built-in function in Amazon. The syntax for it is:

  ```
  this.response.speak('Sure, catch you later.');

  this.emit(':responseReady');
  ```

2.2 Atomic Flash

The index.js file of Atomic Flash's AWS-Lambda function 1 gets the information of atomic number by individual atomic number flashcard or all available atomic flashcards (at AWS.AMAZON.COM) from user voice prompt(Alexa). This information is then stored in DynamoDB.

AWS-Lambda function 2 reads the data from DynamoDB and then transmits to user-visual-interface (website) to display the flashcards. The code that is fed into AWS Lambda runs each time the Alexa skill is invoked. The code used in this app inside lambda functions so far can be found in the following Github repository: https://github.com/kavyakushnoor/Alexa-Atomic-Flashcards.

The Intents/handlers of AWS Lambda - Part 1:

- LaunchRequest: function()

  On calling this intent, Alexa speaks the welcome message that is stored in the lambda function. The syntax for it is:

```
'LaunchRequest': function () {
this.emit(':say', 'welcome to Atomic flash. To see elements,
Tell me an atomic number between 1 and 20 or, say all,  to see
all elements.');}
```

- ShowElementIntent: function()

  This intent shows the element flashcards based on cue from user. It has the syntax:

```
'ShowElementIntent': function () {
const docClient = new AWS.DynamoDB.DocumentClient();
const elementFlashNumber = 0;
const params = {TableName: "ElementFlash",
Key: {"Flashid": 0,},
UpdateExpression: "set flashToShow = :newFlashNumber",
ExpressionAttributeValues:{
":newFlashNumber" : elementFlashNumber}};
docClient.update(params, (() => {
this.emit(':say', 'These are all available 20 elements on
Atomic Flash');}));}
```

- ShowAllElementsIntent: function()

  This intent causes a display of all atomic flashcards on the interface. It has the syntax:

```
const docClient = new AWS.DynamoDB.DocumentClient();
const elementFlashNumber =
this.event.request.intent.slots.number.value;
const params = {
TableName: "ElementFlash",
Key: {"Flashid": 0,},UpdateExpression: "set flashToShow =
:newFlashNumber",
ExpressionAttributeValues:{":newFlashNumber" :
elementFlashNumber}};
```

```
docClient.update(params, (() => {
this.emit(':say', 'element with atomic number '
+elementFlashNumber + 'is'+arr12[elementFlashNumber-1]);}));},
```

- AMAZON.StopIntent: function()

  This is a built-in function in Amazon. The syntax for it is:

```
this.emit(':say', 'Thank you for visiting atomic flash, have a
good day');
```

- AMAZON.CancelIntent: function()

  This is a built-in function in Amazon. The syntax for it is:

```
this.emit(':say', 'Thank you for visiting atomic flash, have a
good day');
```

Figure 4: Lambda functions and DynamoDB in ‘Atomic Flash’

The index.js file of Atomic Flash’s Lambda function of Part 2 interacts mainly with DynamoDB and retrieves values for display of atomic flashcards.

The Intent/handler of AWS Lambda - Part 2 are below:

```
const dynamodb = new AWS.DynamoDB();
const docClient = new AWS.DynamoDB.DocumentClient();
const params = {TableName: "ElementFlash",
Key: {"Flashid": 0}};
const elementFlashToDisplay = "flashcard not set";
exports.handler = (event, context, callback) =>
{docClient.get(params, function(err, data){
```

```
const payload = JSON.stringify(data, null, 2);
const obj = JSON.parse(payload);
elementFlashToDisplay = obj.Item.flashToShow;
callback(null,
{"elementFlash":elementFlashToDisplay});});};
```

3. DynamoDB

   DynamoDB is a NoSQL database management service provided by Amazon Web Services. It stores its data in JSON format. It is selected because of its ability to scale based on data produced.

   For the Atomic Flash App, a number is stored in the database that corresponds with the atomic flashcard to be displayed as per Alexa's input. The atomic number that the user says is read through a variable in the lambda function. This number is written into the DynamoDB table so that the appropriate atomic flashcard is displayed.

4. API Gateway

   API Gateway is an entryway into a system that provides cohesion for interactivity between several microservices. These microservices are meant to work together to make a larger system function uniformly. Thus, the intended service is provided to an end user. The API can be deployed and written into the HTML source code of the required web page.

   In Atomic-Flash App, a GET method is used to retrieve values from Lambda. The required lambda function is selected from the choices which are offered by the API gateway interface.

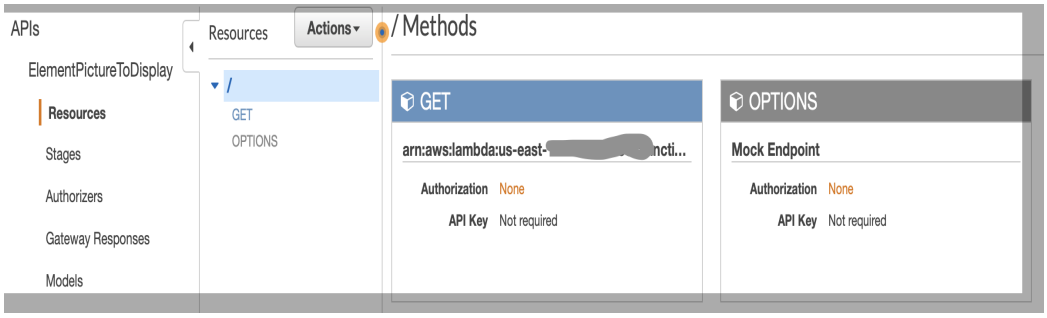

Figure 5: Example of AWS API gateway used in 'Atomic Flash' App

5. Visual User Interface

   For the demonstration of Alexa-Skill App Atomic Flash a web server powered by iPage is used. It has been tested on a local server throughout the development process. This webpage is where the flashcards representing each element are displayed (Figure 6). The API generated using API gateway retrieves the data

from the database (DynamoDB) and invokes the function to load the corresponding atomic flashcard to be posted on the interface.

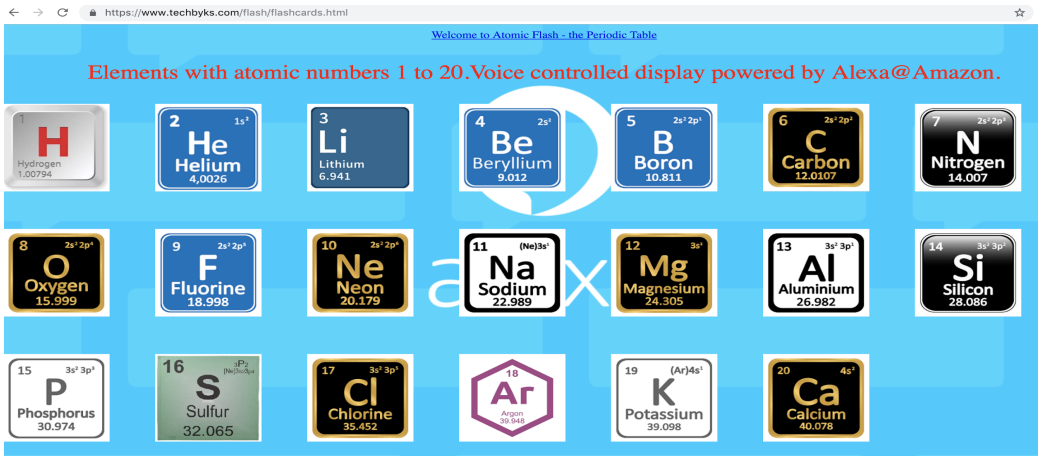

Figure 6: Visual user interface to display atomic flashcards for Atomic Flash App

Conversation Model Between Alexa and User

Demo of Verbiage

Alexa: Welcome to Flashcards App! Let's begin practice. Would you like to practice {Subject1}, {Subject2} or {Subject3}?
User: {Subject3}
Alexa: Here is your first question. {Question}?
User: {Correct Answer}
OR
{Wrong Answer}
Alexa: Correct Answer. You score 1 point.
OR
Wrong answer. The correct answer is: {Correct Answer}
User: Stop
Alexa: Sure, catch you later!

Demo of Atomic Flash

Alexa: Welcome to Atomic Flash! To see elements, tell me an atomic number between 1 to 20 or say 'all' to see all elements.
User: {number}
Alexa: Element with atomic number {number} is {element}
User: {all}
Alexa: These are all available elements on Atomic Flash
User: Help
Alexa: You can ask for an element flash by saying the atomic numbers or say all to see all elements on the webpage.
User: Stop
Alexa: Thank you for visiting atomic flash. Have a good day!

A video containing the demonstration of Atomic Flash can be found here at:
https://www.youtube.com/watch?v=ciUEdOIqBo8

Methods Used to Connect the Database Layer

Methods that got Ruled Out

1. Php and MySql
2. External domain to AWS RDS
3. S3 to DynamoDB
4. AWS Lightsail based Wordpress site connected to DynamoDB

Methods that Worked

1. External domain connects with DynamoDB using API
2. Alexa reads data from dynamoDB

Though the above mentioned approaches have not been executed completely, they seem promising because quality textbooks and articles suggest this approach.

Timeline

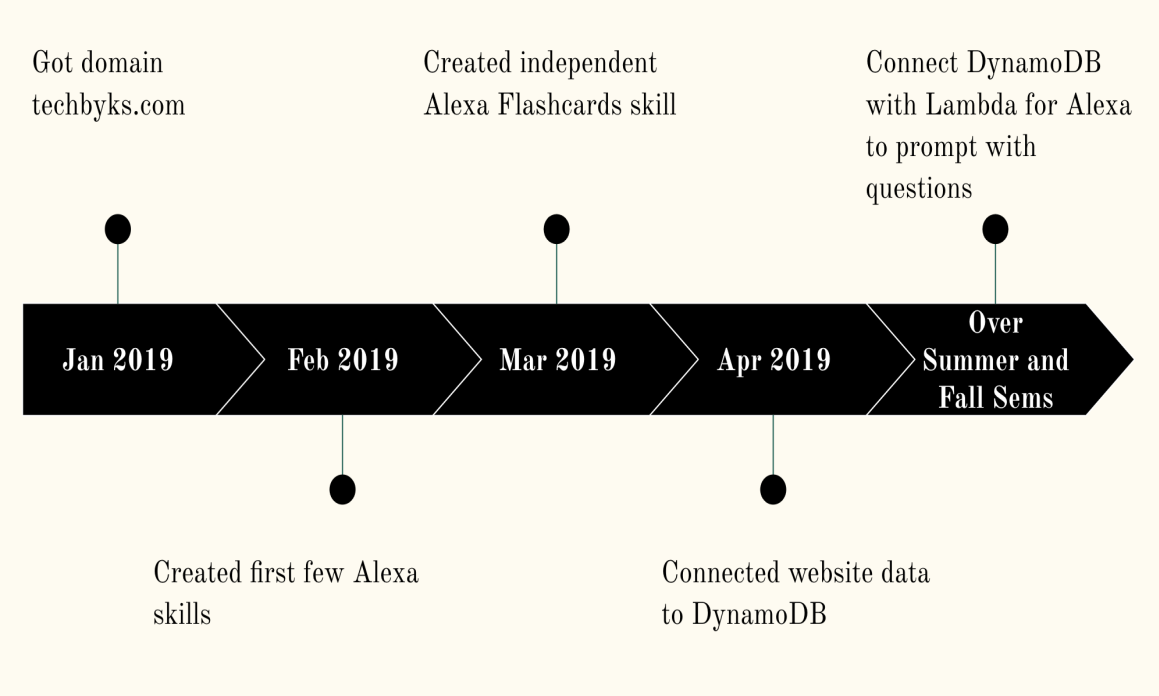

Figure 7: Capstone project timeline and progress made

Next Steps

1. Connect Alexa's front-end with the DynamoDB data.
2. Improvise the website's user-interface and HCI component.
3. Develop a user evaluation plan with a human subjects' experiment to determine the ease of use and acceptability of the tool.
4. Deploy the final application developed.

User Study

A user study is to be conducted to verify the user experience for an end-user. The functionality, usability and ease of use are the parameters based on which these assessments are made. The intended users for this application are teachers and students who wish to improve and enhance the learning experience using technology. The research participants are students of UNCW with little or no programming experience.

The tasks intended for the research participants are to:

1. Create flashcards using the web-browser
2. Test if the flashcards are readable and practicable by Alexa
3. Fill a survey which asks them to rate their experience in terms of convenience, ease of use and functionality

The test is to be evaluated based on the research participants' response in the survey questionnaire that they can fill at the end of the experiment. The following details are entered by them:

1. The quality of their user experience and the extent to which they were able to do the tasks
2. Their assessment of the ease of using this application
3. The likability or frustration in working with the application

Having a realistic picture of the end-user experience can help make necessary accommodations in the code that triggers the dialog model. The application can be tweaked to better suit the user needs.

Future Architecture Diagram

The anticipated model of the App for the final product is below in Figure 8. The planned App will have features from both the apps ‘Verbiage’ and ‘Atomic Flash’. Users will also have flexibility to create content on the web-browser and practice the content using voice interaction with Alexa.

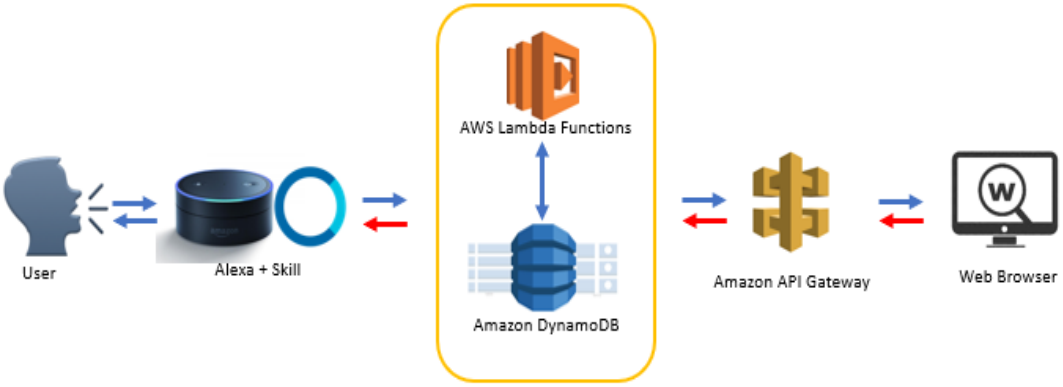

Figure 8: Future planned architecture diagram for final product of the Capstone

References

1. (n.d.). Retrieved from https://developer.amazon.com/docs/ask-overviews/build-skills-with-the-alexa-skills-kit.html
2. (n.d.). Retrieved from https://developer.amazon.com/alexa-skills-kit
3. (n.d.). Retrieved from https://www.codecademy.com/courses/learn-alexa
4. (n.d.). Retrieved from https://medium.freecodecamp.org/how-to-create-an-alexa-skill-that-manages-to-do-lists-11c4bab29ea5
5. (n.d.). Retrieved from https://www.youtube.com/playlist?list=PL2KJmkHeYQTNwlZqLh_ptZhSNZf93e8Sp
6. (n.d.). Retrieved from https://developer.amazon.com/alexa-skills-kit/big-nerd-ranch
7. (n.d.). Retrieved from https://developer.amazon.com/alexa-skills-kit/tutorials
8. Retrieved from https://developer.amazon.com/alexa-skills-kit/tutorials
9. (n.d.). Retrieved from https://github.com/alexa
10. (n.d.). Retrieved from https://pythonprogramming.net/intro-alexa-skill-flask-ask-python-tutorial/
11. (n.d.). Retrieved from https://pythonprogramming.net/intro-alexa-skill-flask-ask-python-tutorial/
12. Poccia, D. (n.d.). *AWS Lambda in Action, Manning Publications*. Manning.
13. (n.d.). Retrieved from https://learningenglish.voanews.com/a/build-muscle-memory-to-improve-your-pronunciation/3918000.html
14. (n.d.). Retrieved from http://www.avspeechtherapy.com/2012/06/22/speech-muscle-memory/
15. Creating Muscle Memory. (n.d.). Retrieved from https://www.speechimprovement.com/creating-muscle-memory/