



Flashcards Application Powered by Alexa

CAPSTONE DEFENSE

ADVISOR

Dr. Curry Guinn

COMMITTEE MEMBERS

Dr. Judith Gebauer

Dr. Ellie Ebrahimi

PREPARED BY

Kavyashree Kushnoor

University of North Carolina Wilmington

Table of Contents

Chapter 1 - Introduction.....	5
Abstract.....	5
Introduction.....	5
Motivation.....	5
Courses.....	6
Objective.....	6
Current Architecture Model Diagrams.....	7
Conversation Models.....	8
Chapter 2 - Design & Progression.....	10
Inspiration.....	10
Gaps in Design of Final App.....	11
Role of Slots in the Conversation Model.....	12
Challenges in Implementing Quiz Wire.....	16
Chapter 3 - Implementation.....	18
Web UI.....	19
AWS Lambda.....	20
AWS DynamoDB.....	24

AWS IAM.....	26
AWS API Gateway.....	28
Alexa Voice Interface.....	31
Chapter 4 - User Study Experiment.....	33
Chapter 5 - Conclusion.....	41
Methods Considered to Connect the Database Layer.....	41
Next Steps.....	42
Timeline.....	42
References.....	43

Table of Figures

Figure 1 - <i>Architecture model of Verbiage</i>	7
Figure 2 - <i>Architecture model of Atomic Flash</i>	7
Figure 3 - <i>Apps developed as part of Capstone work</i>	10
Figure 4 - <i>Intended architecture for Quiz Wire</i>	12
Figure 5 - <i>Alexa processes the users' words using Slots</i>	13
Figure 6 - <i>Examples of in-built slots in Alexa Development Console</i>	13
Figure 7 - <i>Sample Utterances that contain the in-built slot 'number'</i>	14
Figure 8 - <i>Custom Slot sample implemented in Verbiage</i>	15
Figure 9 - <i>Sample Utterances that contain a Custom Slot 'answer'</i>	15
Figure 10 - <i>Two layered response cushion saved for Alexa to process</i>	16
Figure 11 - <i>Challenge of Slots in Quiz Wire</i>	17
Figure 12 - <i>Architecture of Quiz Wire Application</i>	18
Figure 13 - <i>React JS based Web UI used in Quiz Wire</i>	19
Figure 14 - <i>Web UI used in Atomic Flash</i>	19
Figure 15 - <i>Lambda Triggers CRUD functions</i>	20
Figure 16 - <i>Lambda Triggers the Alexa application</i>	20
Figure 17 - <i>DynamoDB connected to multiple lambda functions</i>	25

Figure 18 - <i>Identity Access Management</i>	26
Figure 19 - <i>IAM access process</i>	27
Figure 20 - <i>General Voice Only Architecture Model</i>	31
Figure 21 - <i>Highest level of education attained by participants</i>	34
Figure 22 - <i>Field of education of highest degree</i>	35
Figure 23 - <i>Industry of current occupation</i>	35
Figure 24 - <i>Level of technological expertise of participants</i>	36
Figure 25 - <i>Overall Evaluation of the App</i>	36
Figure 26 - <i>Ease of use of the App</i>	37
Figure 27 - <i>Look and feel rating of the App</i>	37
Figure 28 - <i>Likelihood of participants' recommending the App to a friend</i>	38
Figure 29 - <i>Likelihood of participants' reusing the App after experiment</i>	39
Figure 30 - <i>Most favorable features as per participants</i>	40
Figure 31 - <i>Project Timeline</i>	43

Chapter 1 - Introduction

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 users to create Alexa-powered flash card games without special programming knowledge. Users will fill in a simple template 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.

Introduction

Alexa's developer console provides the ability to customize skills (14). 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 Console are used. These features are further discussed in detail in the section 'Architecture Description'. These tools have been chosen because of their popularity among Alexa developers. Most discussion forums, blogs and quality tutorials and textbooks use these tools.

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 the 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 (11) - In the form of visual display of flashcards on the web browser

2. Auditory Memory (12) - 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 (13) - In the form of creating flashcards on the website

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.

Courses

The idea of the project has evolved from its crude form to one that has implementation value and scope for scalability. Numerous courses and certificates have been instrumental in developing the skills required for this project. Some courses are technological while others are related to curriculum development. They are:

1. Programming Foundations with JavaScript, HTML and CSS - on Coursera, offered by Duke University
2. Teach English Now! Theories of Second Language Acquisition - on Coursera, offered by Arizona State University
3. Alexa for Developers - AWS training, offered by Amazon
4. Exam Readiness: AWS Certified Solutions Architect - Associate - AWS training, offered by Amazon
5. AWS Certified Cloud Practitioner - Issued by AWS
6. Responsive Website Basics: Code with HTML, CSS, and JavaScript - on Coursera, offered by the University of London
7. Teach English Now! Foundational Principles - on Coursera, offered by Arizona State University
8. Teach English Now! Lesson Design and Assessment - on Coursera, offered by Arizona State University
9. Introduction to Alexa - on Codecademy
10. Conversational Design with Alexa - on Codecademy
11. CSC 592 - at UNCW, Cloud Computing Using AWS, Spring 2018

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, accessible to one or more users.

Current Architecture Model for 'Verbiage' App



Figure 1: Architecture model of Verbiage that uses the Developer Console as the front-end and AWS Lambda as the back-end. They are connected using the ARNs generated by Amazon.

Current Architecture Model for 'Atomic Flash' App

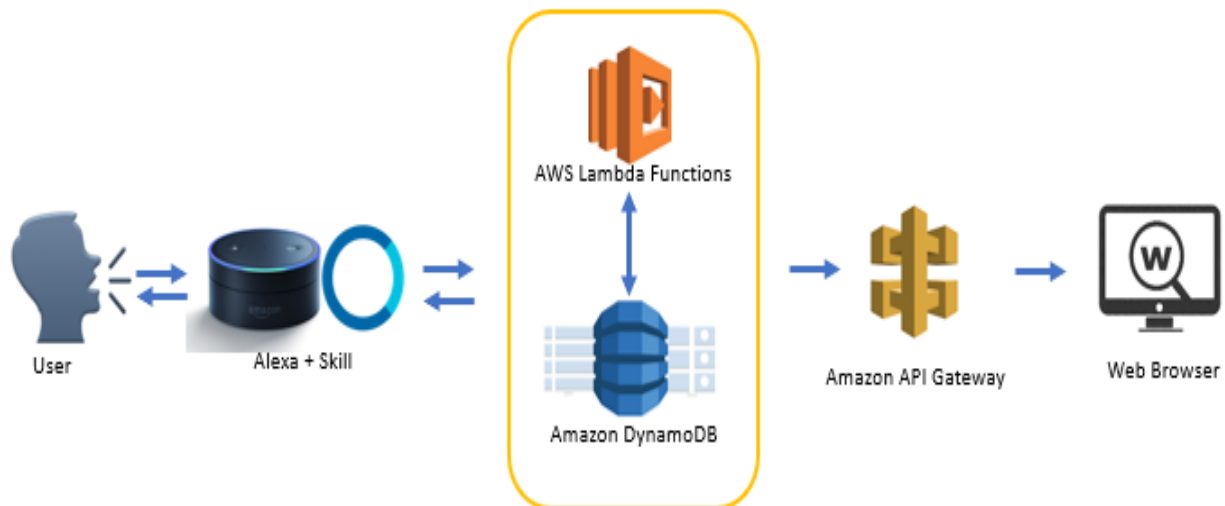


Figure 2: Architecture model of Atomic Flash that uses the Developer Console + Web browser as the front-end and AWS Lambda + DynamoDB + API Gateway as the back-end. They are connected using the ARNs generated by Amazon.

A video containing the demonstration of Atomic Flash can be found here at:

<https://www.youtube.com/watch?v=ciUEdOlqBo8>

Conversation Model Between Alexa and the User

The conversation model indicates the dialog flow that would take place between Alexa and a user. Each time a user speaks to Alexa, the words spoken by the user are termed as *Utterances*. These specific words are programmed to trigger a function in AWS Lambda (the back-end) that further prompts Alexa to speak the next sentence in the dialog model displayed below. An *Intent* represents the front-end name of the trigger (as stored on the Alexa Development Console). A *handler* represents the back-end name of the same trigger (as stored in AWS Lambda)(7). In summary, a user speaks the Utterance. The development console invokes the Intent, which further triggers the lambda code into action using the Handler. Following is the demo of the conversation models of Verbiage and Atomic Flash:

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!

Demo of Quiz Wire

The in-built slot of 'number' is used in the existing Quiz Wire application. The current conversation model in Quiz Wire is different from the one intended during the planning and proposal. A number of challenges and technological changes have led to this difference. The current conversation model is as follows:

Alexa: Welcome to Quizwire! Let's begin practice. Tell me the number.

User: {number}

Alexa: The answer for {front side1} is {back side1}

User: {number}

Alexa: The answer for {front side2} is {back side2}

User: Stop

Alexa: Sure, have a good day!

Chapter 2 - Design and Progression

Inspiration

The project is inspired by the idea of providing a platform where audio and speech can be made a part of technologically enriched learning. In creating the final web + voice interface for this application, a series of several apps got created as stepping stones and incremental iterations. Out of the numerous applications that came out of the practice and development, three applications stood out most because of their utility and inherent value that a user can derive out of them.

As indicated in Figure 3 below, the skills and lessons learned from each app (5) has lead to a cumulative progression in the proceeding app development.

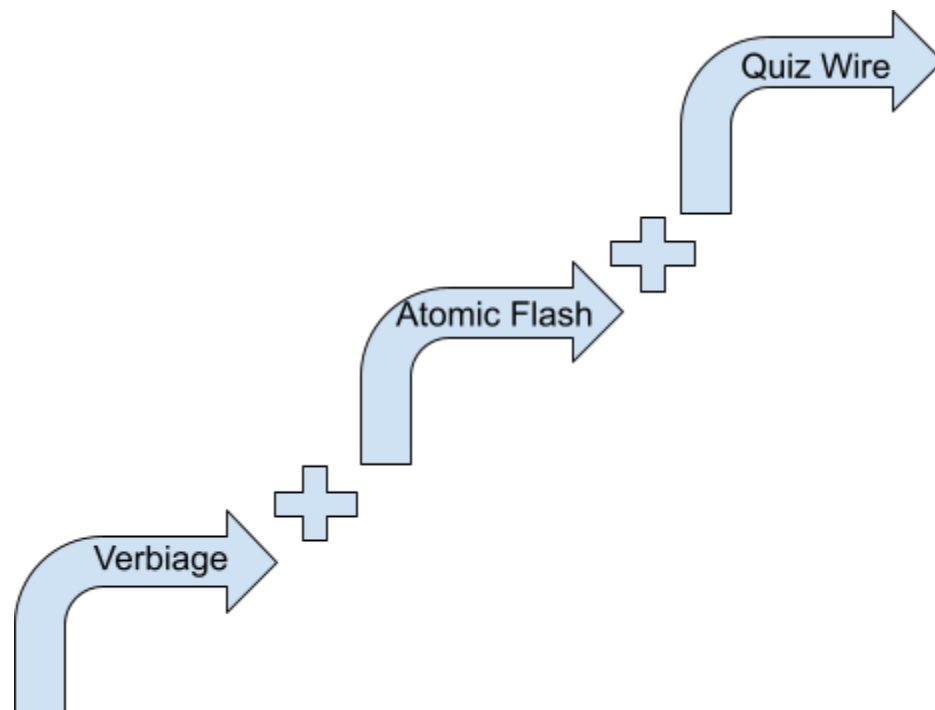


Figure 3: The three apps developed as part of Capstone work are Verbiage, Atomic Flash and Quiz Wire. Each app leads to a cumulative progression in the proceeding app development.

In this report, we shall discuss the implementation, similarities and differences in all three of these applications. They are:

- a. Verbiage

Verbiage is an all-audio flashcard set powered by Alexa. Using Verbiage a user interested in learning English as a second language can practice irregular verbs for memorization.

b. Atomic Flash

Atomic Flash uses a web interface wherein flashcards representing different elements of the periodic table are displayed. The display is controlled by voice commands that a user can say to Alexa.

c. Quiz Wire

Quiz Wire is the final application developed as part of the capstone project whereby a user can create flashcards on a web-page. The flashcards are then played using by Alexa. A user can invoke the Quiz Wire skill and have Alexa review the questions and answers in the flashcards, thereby using audio-visual technological aid to enhance memorization.

Gaps in the Design of the Final App

In the process of designing the function, voice user interface, web interface and the conversation model there were several hurdles and setbacks because of the available technology. The skills from each skill had a cumulative effect that lead to a sturdier app in the progression sequence. However, each application had additional features and signature styles which required customization. It meant that there would be elimination of several features from the precursor apps and addition of other technology to give each app its unique functionality. (8)

If we were to follow an ideal cumulative effect, the intended conversation model for Quiz Wire would be as follows:

Alexa: Welcome to Quizwire! Let's begin practice. Are you ready? To begin say 'yes' or 'start'.

User: Yes/ start

Alexa: Here is your first question. {Question}?

User: {Correct Answer}

OR

{Wrong Answer}

Alexa: Correct Answer.

OR

Wrong answer. The correct answer is: {Correct Answer}

User: Stop

Alexa: Sure, have a good day!

However, after considering the technological features available and the changes in the AWS platform, the actual finalized conversation model is different from the one above. We shall see the existing model in place in the next chapter titled 'Implementation'. For now, we shall look into the factors that caused the detour in the design and algorithm. The architecture however, remains similar to the one planned during the proposal. It is presented in Figure 4 below.

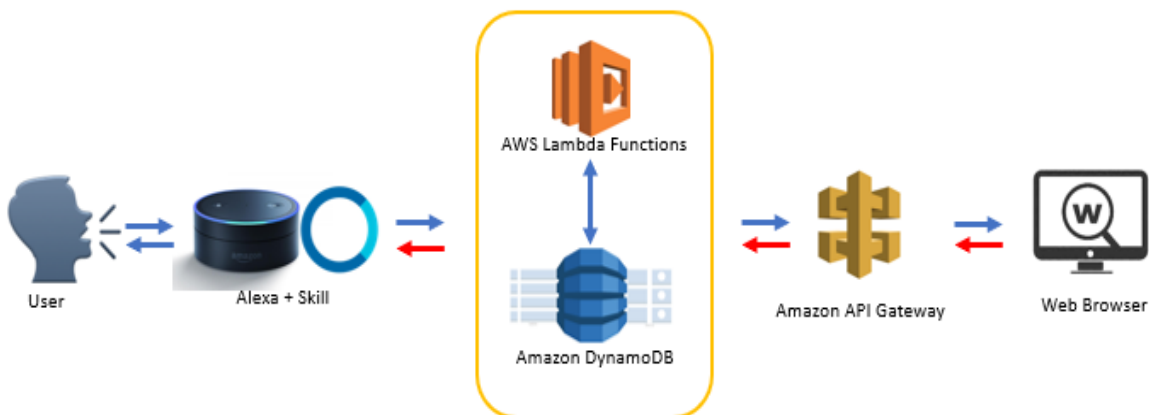


Figure 4: Intended architecture for Quiz Wire that uses the Developer Console + Web browser as the front-end and AWS Lambda + DynamoDB + API Gateway as the back-end. They are connected using the ARNs generated by Amazon.

Role of Slots in the Conversation Model

Slots aid in Alexa's processing of the possible words that a user is speaks (6). When a user says an *Utterance*, a huge subset of words can be covered under a single category represented by a *Slot*. For security purposes, the expected replies and words that a user utters must be pre-saved in the code. This prevents any vulnerability where Alexa may end up hearing/ recording any unintended speech.

In case of an in-built slot, the lambda function does not need to have the possible slot values stored. However, if the slot is customized, then there needs to be a 2-layered cushion that helps Alexa identify, process and respond to the words that the user is speaking. Figure 5 below indicates how a users' words are processed by Alexa.

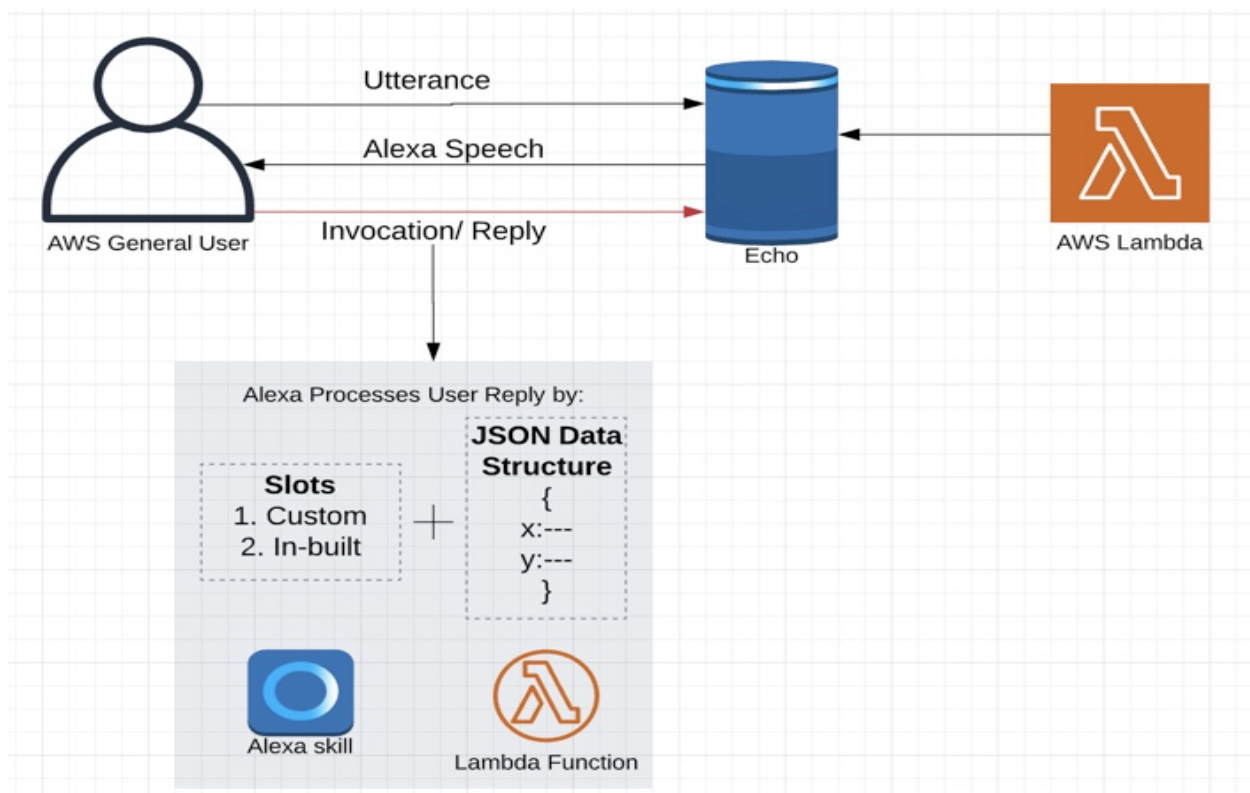


Figure 5: Alexa processes the users' words using Slots. There are two kinds of slots that can be implemented in the Alexa Development Console: Custom & In-built

There are two kinds of slots that can be implemented in the Alexa Development Console (4). They are:

a. In-built slots

In-built slots are in the format `AMAZON.xxxxxx`, as represented in Figure 6 below. They do not require any additional entry of words within the specific category on

behalf of the developer. Amazon has several examples within each category that the user can implement as per the need.

AMAZON.Corporation	AMAZON.NUMBER	AMAZON.DE_CITY
AMAZON.Country	AMAZON.NUMBER	AMAZON.DE_FIRST_NAME
AMAZON.CreativeWorkType	AMAZON.Actor	AMAZON.DE_REGION
AMAZON.DATE	AMAZON.AdministrativeArea	AMAZON.Dessert
AMAZON.DayOfWeek	AMAZON.AggregateRating	

Figure 6: Examples of in-built slots in Alexa Development Console. They use the format that begins with 'AMAZON.' and have an in-built list of data items that a developer can use.

Figure 7 below shows a screen-shot that provides an example of how Alexa processes slots inside an utterance.

Intents / ShowElementPictureIntent

Sample Utterances (5) ?

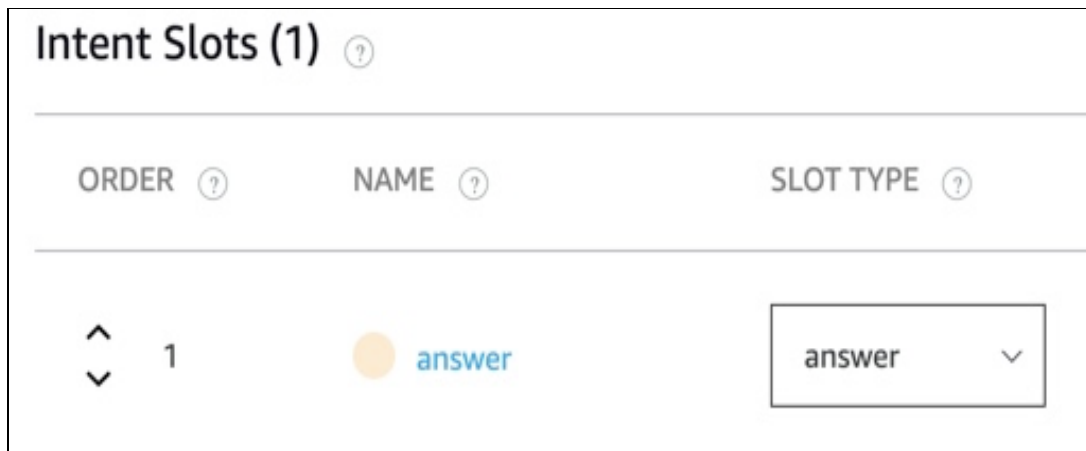
What might a user say to invoke this intent?

atomic number	{number}
show me element	{number}
image	{number}
element	{number}
element number	{number}

Figure 7: Sample Utterances that contain the in-built slot 'number' which corresponds to any number that the user may say. It is allocated to a slot since the values are varied.

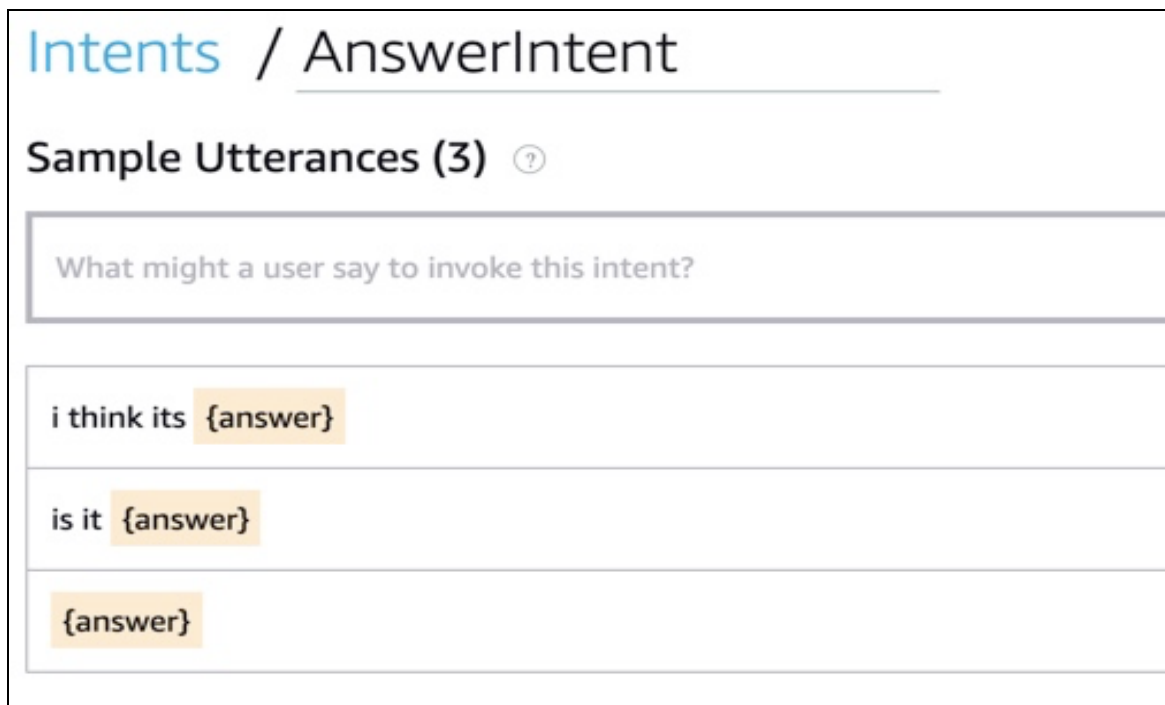
b. Custom slots

Custom slots can be in any format as per the wish of the developer. Figure 8 below shows an example custom slot that has been implemented in Verbiage. The implementation is more complex than an in-built slot since the user has to feed all the words within the category that has been generated. Figure 9 shows the way in which the slot is being used inside the sample utterances.



ORDER	NAME	SLOT TYPE
1	answer	answer

Figure 8: Custom Slot sample implemented in Verbiage. It does not have to follow a required format as in Atomic Flash.



Intents / AnswerIntent

Sample Utterances (3)

What might a user say to invoke this intent?

i think its {answer}

is it {answer}

{answer}

Figure 9: Sample Utterances that contain a Custom Slot 'answer'. Alexa responds to the utterances if they match any of these samples.

A further and more detailed look of how a custom slot provides a two layered cushion for better functionality can be viewed in Figure 10 below. Since the slot values are not in-built, the developer needs to go the extra mile by saving the values in the values of Alexa Developer Console + in Lambda within a JSON data structure.

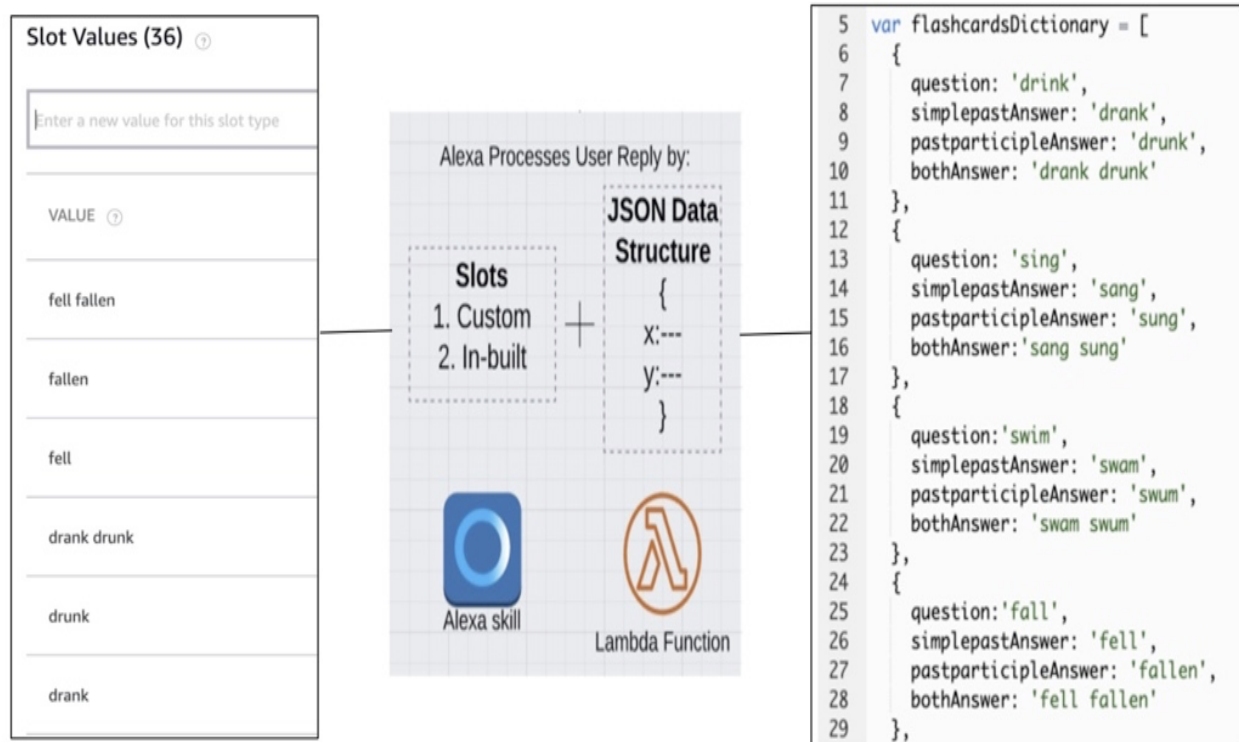


Figure 10: A customized two layered response cushion saved for Alexa to process. The column on the left contains the front end values and the column on the right contains the back-end Lambda values.

Challenges in Implementing Quiz Wire

The changes in the underlying AWS platform and the usage of slots have proven to be the major challenges in implementing the Quiz Wire app as intended. Following is a brief description of each of these challenges:

a. Open ended slots

Since the data entered into the flashcards comes from the world wide web, it is difficult to predict what kind of a slot would be ideal in the Alexa Developer Console. A diagrammatic representation of this challenge is presented in Figure 11.

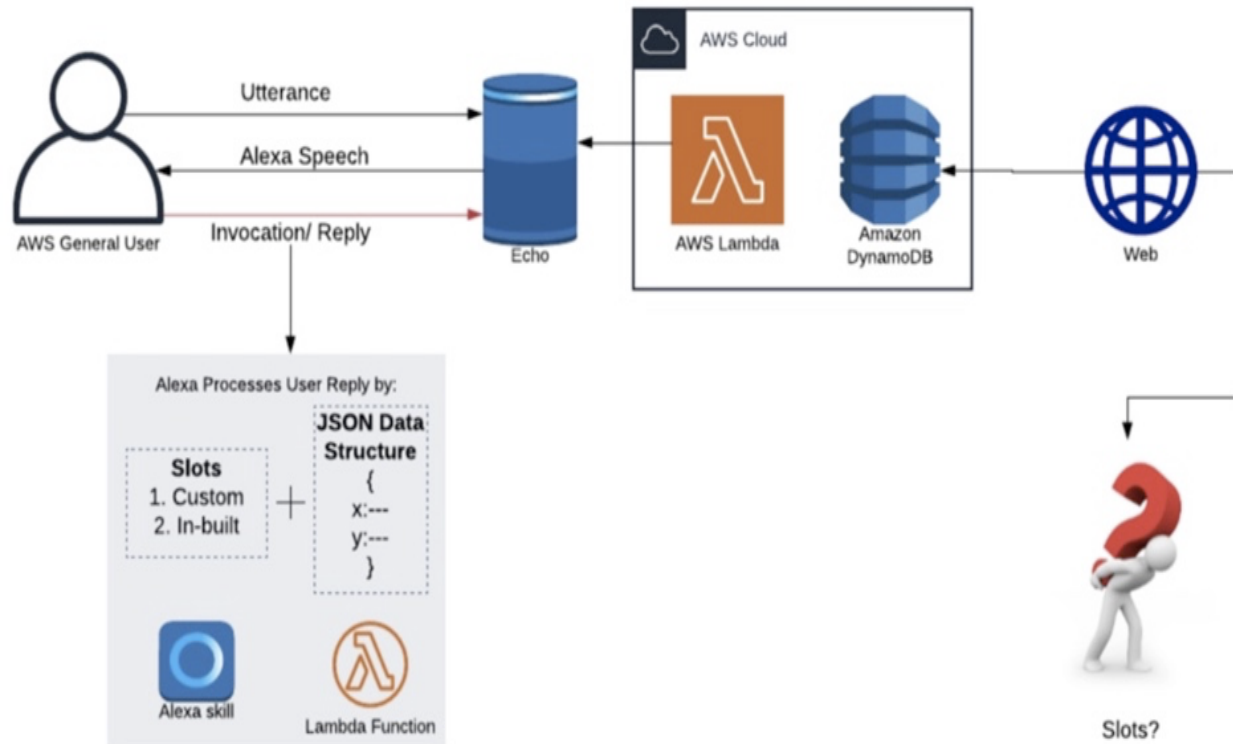


Figure 11: Challenge of Slots in Quiz Wire comes from the fact that the data comes from the website and hence is open-ended. It is difficult to create customized slots that are not hard-coded and with no predefined predictable pattern.

b. AWS Lambda Blueprint for factskill has changed

Most of the tutorials, blogs and articles used for e-learning implemented the 'factskill blueprint' as the foundation for coding. This sample app is a shortcut provided by Amazon to get everything set-up to run the skill. It is a template of the lambda function that automates Alexa's permissions, set-up of IAM roles and required dependencies are set-up without any extra manual work on behalf of the developer/

A recent change in AWS Lambda is that the factskill code has moved from the 'Blueprint' section to the 'Serverless App Repository' section. This transformation has drastically changed the way in which the app is deployed, saved and integrated into the whole functionality, compared to how they are taught on the e-learning forums. It also means that new research would have to be conducted to figure out the ways in which to work with this transition and set-up of the complete architecture.

Chapter 3 - Implementation

The Quizwire application demonstrated in this Capstone project is a prototype. It proves that the concept of flashcards powered by Alexa is functional. Additional work would be required to make it secure and scalable. A user is able to integrate two separate devices; a screen (via desktop, tablet or phone) and Alexa in order to accomplish the intended task of creating and using flashcards.

Quizwire contains a single page React JS application for the front-end user interface. The back-end contains a serverless computing platform using AWS Lambda along with several other AWS features. The input collected through an HTML form is stored in the data layer. It is stored using the DynamoDB database. It gives the data persistence.

The CRUD (Create, Read, Update, Delete) operations are powered by Lambda. API Gateway integrates the web based UI with the back end by providing the REST API. AWS IAM is used to create an Identity and Access Management role that represents the Quizwire application. This role inter-connects the various AWS features together by uniformly integrating them. The integration is done by adding permissions and specific in-line policies. An overall view of this architecture is presented in Figure 12.

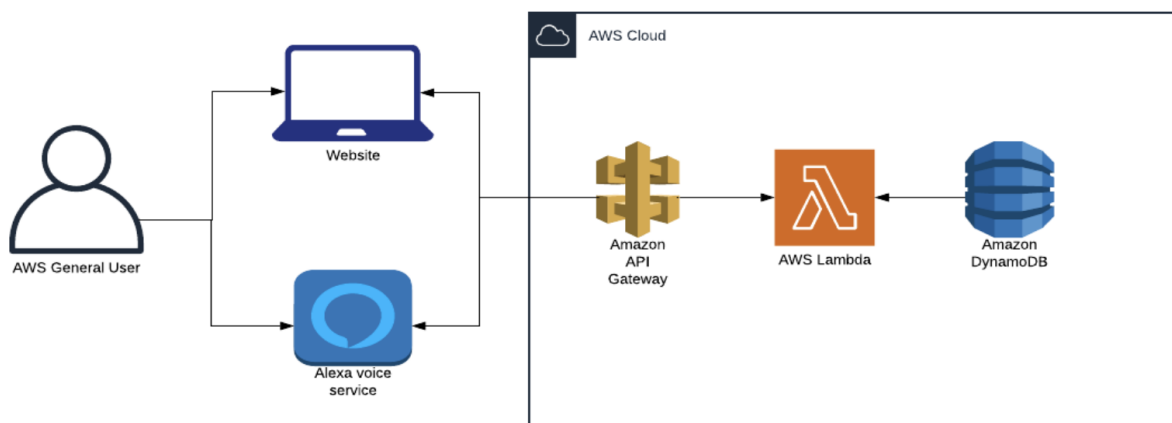


Figure 12: Architecture of Quiz Wire Application that uses the Developer Console + Web browser as the front-end and AWS Lambda + DynamoDB + API Gateway as the back-end. They are connected using the ARNs generated by Amazon.

Each resource (9) in the AWS cloud above has an Amazon Resource Number (ARN). This ARN allows cross utility of resources within a given information system. Its function

is similar to an API but it works internally for AWS based cross integration. Following is an algorithm of the overall architecture process:

1. Receive and record data that user enters in the form to create flashcards. Save this data in DynamoDB.
2. A lambda function creates a new entry in the table, retrieves, updates or deletes it based on the function that is invoked.
3. Each of the functions are connected to the web interface through an API
4. A separate lambda function is responsible for reading data from the table and having Alexa say the values in the said format.

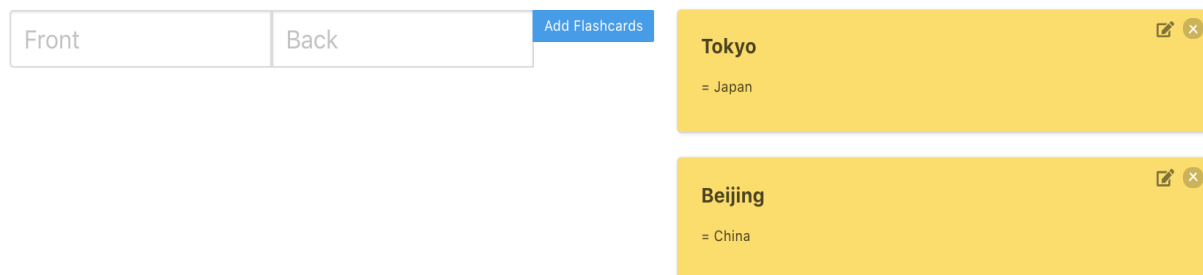
Following is the further description of each of the resources used in the architectures:

1. Web User Interface

The Web application contains a form that allows the user to create flashcards by entering the front and back side in two separate text-boxes. This data is reflected in small rectangular cards, on the right side of the form, that serve as a representation of the flashcards. Figure 13 shows a screen-shot of the visual interface used in Quiz Wire.

Flashcard Creator for Alexa

Create or delete flashcards using the form below:



The screenshot displays a web interface for creating flashcards. On the left, there is a form with two text input fields labeled 'Front' and 'Back'. To the right of these fields is a blue button labeled 'Add Flashcards'. To the right of the form, there are two yellow rectangular tiles representing flashcards. The top tile contains the text 'Tokyo' followed by '= Japan'. The bottom tile contains the text 'Beijing' followed by '= China'. Each tile has a small icon in the top right corner consisting of a pencil and an 'x'.

Figure 13: React JS based Web UI used in Quiz Wire. The user writes the front and back side for each flash card and the card reflects the said values on the yellow tiles alongside

Figure 14 shows a screen-shot of the visual interface used in Atomic Flash.

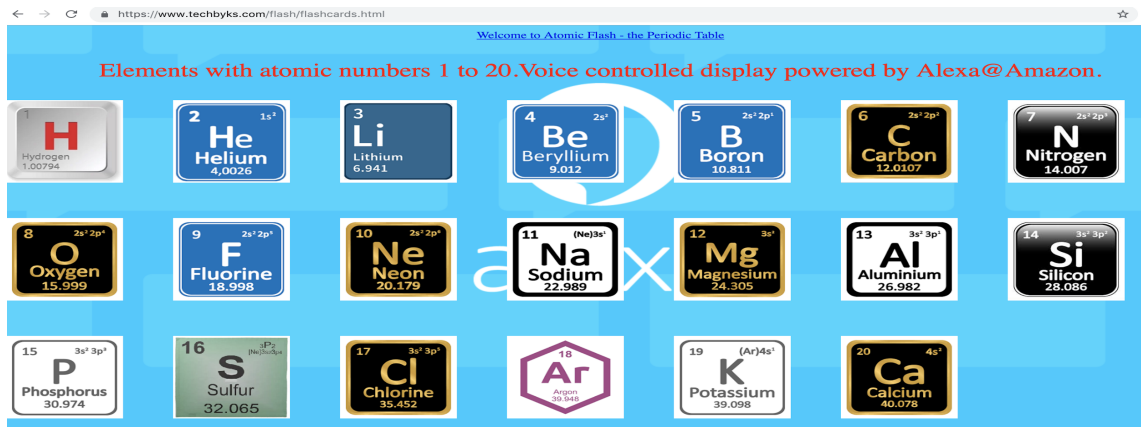


Figure 14: Atomic Flash web interface wherein flashcards representing different elements of the periodic table are displayed. The display is controlled by voice commands that a user can say to Alexa.

2. AWS Lambda

Lambda (10) is the platform for serverless computing. It controls the CRUD (Create, Read, Update, Delete) operations in Quiz Wire. There are four different functions responsible for each task. Each of the functions are unified using a common IAM role that helps identify the application which they are addressing. These back-end functions will be connected to the database, website and Alexa's Developer Console. This connection is done using APIs, ARNs, trigger configurations. The triggers used for each of the CRUD functions in Lambda resembles the format presented in Figure 15 below:

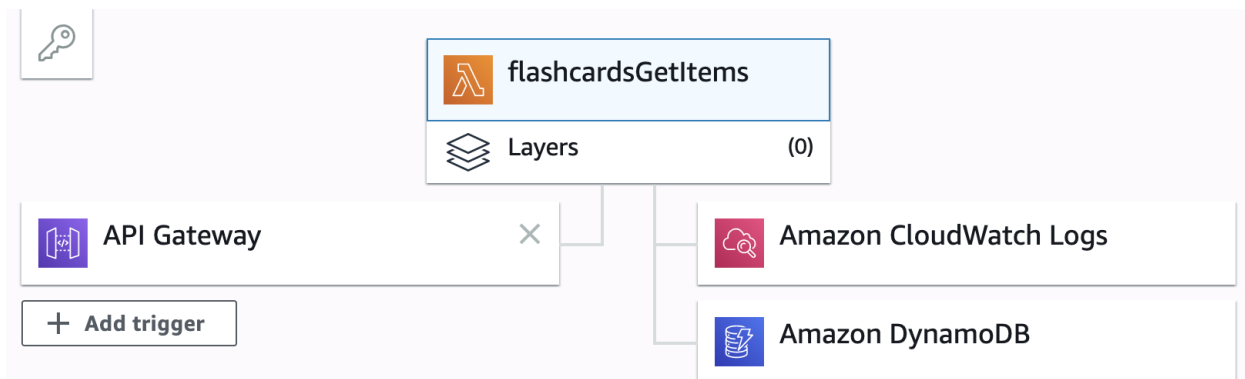


Figure 15: Lambda Triggers CRUD functions. A tree diagram representing the resources are presented in the image. API Gateway, Amazon CloudWatch Logs and DynamoDB are the primary resources here.

The triggers used in the function connecting Lambda with Alexa Skills Kit is presented in Figure 16 in the format below:

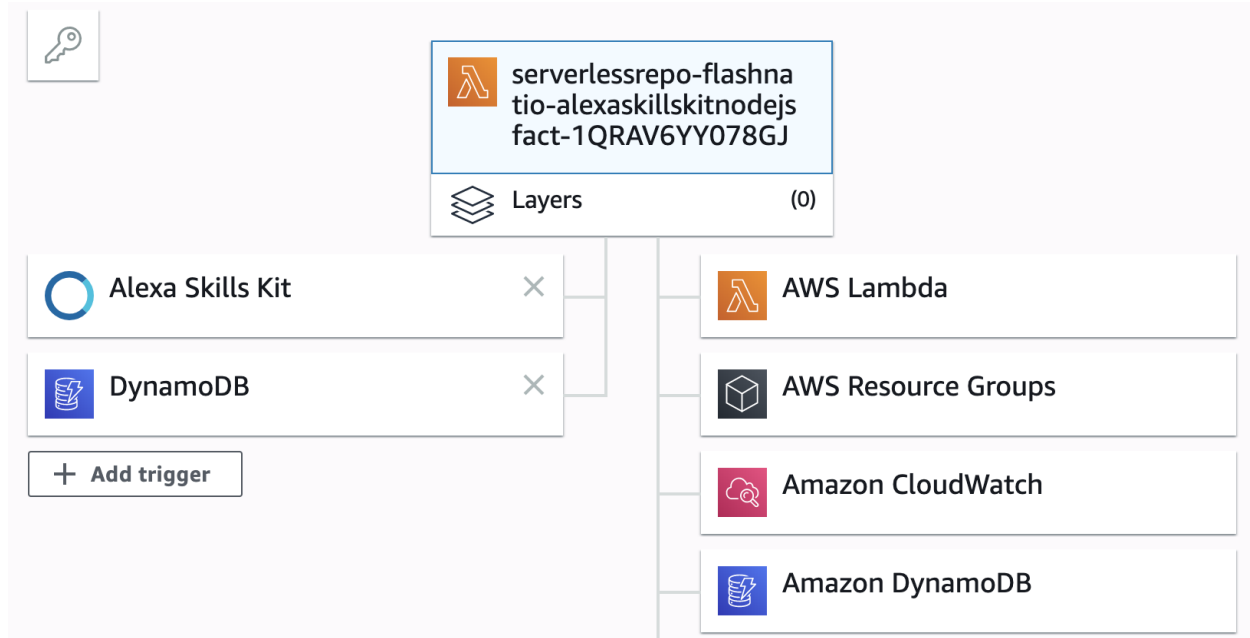


Figure 16: Lambda Triggers in the Alexa application. A tree diagram representing the resources are presented in the image. Alexa Skills Kit, API Gateway, Amazon CloudWatch Logs, Resources Groups and DynamoDB are the primary resources.

The index.js file of Verbiage's AWS-Lambda (1) (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 (3) 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');
```

- VerblIntent: 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');
```

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');

```

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. AWS DynamoDB

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

```
{
  "Item": {
    "Flashcards": {
      "front": "xxxxxx1",
      "back": "-----1"
    },
    {
      "front": "xxxxxx2",
      "back": "-----2"
    }
  }
}
```

Each table has a separate Amazon Resource Name (ARN) that helps identify and inter-connect various AWS features with each other. Using DynamoDB adds persistence to the application being built. A user can enter data, take a break and revisit the application after a long time and still find the saved data since it is stored and retrieved from the DynamoDB repository. The permission to read and write to this database comes from the IAM manager where a user can manage permissions and policies.



Figure 17: DynamoDB connected to multiple lambda functions. One of them handles the CRUD functions while the other helps Alexa read the values from the table.

For the application used in this project, DynamoDB is connected to different set of lambda functions - those that are responsible for the data coming from the web

interface and those functions that connect the table to Alexa's back-end. This is presented in Figure 17 above.

4. AWS IAM

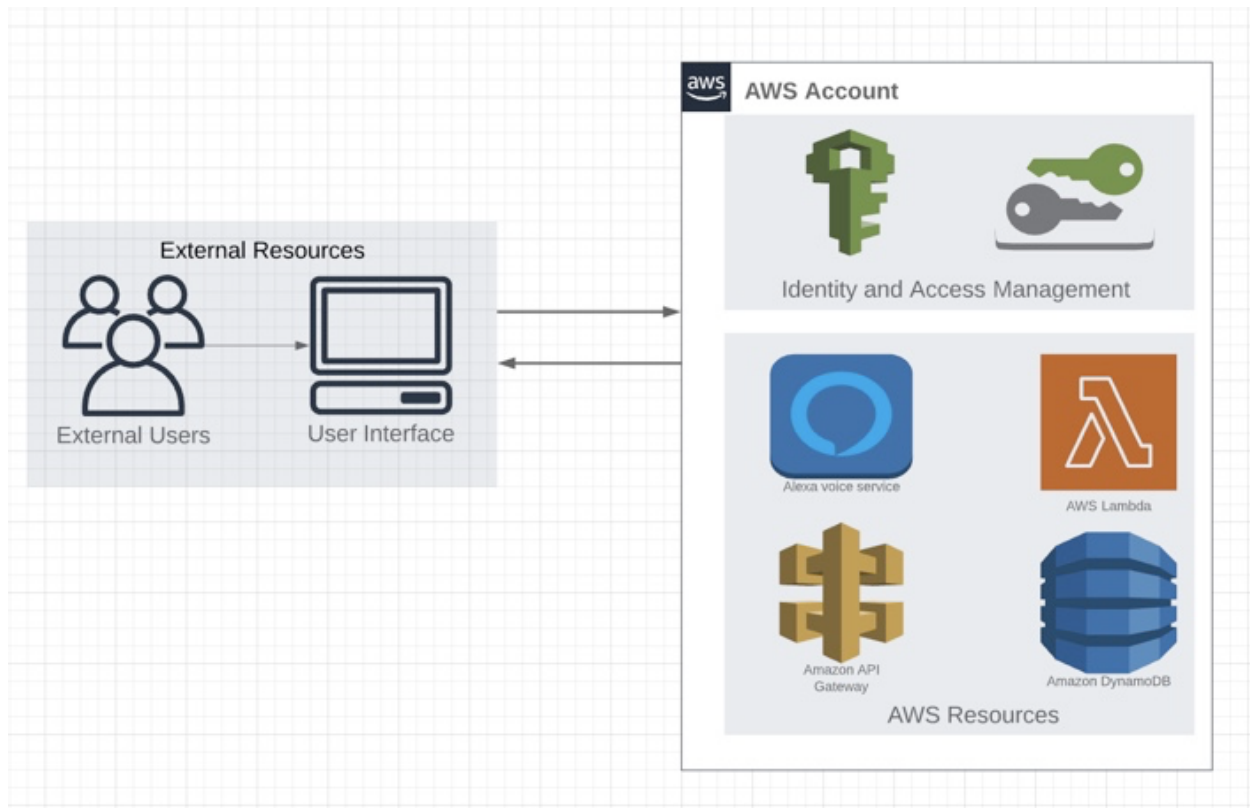


Figure 18: Identity Access Management manages access of users or groups to the AWS resources. It provides secure access to multiple users

Identity and Access Manager is used to manage access of users or groups to the AWS resources used in a particular account, as presented in Figure 18 above. IAM provides secure access to multiple users by controlling access based on predefined permissions saved in the system. Any user can access the resources of an information system securely without having a separate IAM account for it. The process is outlined in Figure 19 below. IAM can be activated by creating a role and attaching the required policies to it. The permissions to access this policy are set based on the need. Whenever an AWS resource is used, it requires identification using an IAM role that determines access level control.



Figure 19: IAM access process has five steps. Roles are defined, policies attached, Permissions managed and access tested. IAM policy that is attached to the role is present in a JSON format

The IAM policy that is attached to the role is present in a JSON format. In the case of this application, one of the policies attached is Cloudwatch access. The policy attached has an auto-populated JSON structure in this format:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateGroup",
        "logs:CreateStream",
        "logs:PutEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

The policy attached to connect DynamoDB and Lambda is in the format:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:Put",
        "dynamodb:Delete",
        "dynamodb:Get",
        "dynamodb:Patch",
        "dynamodb:Update"
      ],
      "Resource":
        "arn:aws:dynamodb:us-west-2:71x9x7x4xxx6:table/Flashcards"
    }
  ]
}
```

5. AWS API Gateway

API Gateway is an AWS resource for the design, management and maintenance of REST and Web-socket APIs. It facilitates the cross-integration of data between AWS with external web-services or with cloud-based data.

For this project, the APIs allow a flow of data in:

- a. An external web-UI with AWS
- b. Between DynamoDB and Lambda, within AWS

The four lambda functions created above representing Create, Read, Update and Delete are targeted using the APIs created in this step. The methods used in the APIs are:

- a. Put
- b. Delete
- c. Get
- d. Update

	API Method	Corresponding Lambda Function	API End-point Format
1	Put	Create	domain.com/DynamoDBtablename/{id}
2	Delete	Delete	domain.com/DynamoDBtablename/{id}
3	Get	Read	domain.com/DynamoDBtablename
4	Patch	Update	domain.com/DynamoDBtablename/{id}

The APIs created using REST format and they have an endpoint that is edge-optimized. This helps in the case of users who are located in different parts of the world. AWS deploys requests that go to the nearest edge location where the user is located and not just the region where the developer's AWS account exists.

The authorization is given to AWS_IAM in each API method. A model schema is entered using JSON with the following format:

```
{
  "$schema": "http://json-schema.org/draft-xx/schema#",
  "title": "InputModelFlashCards",
  "type": "object",
  "properties": {
    "front": {"type": "string"},
    "back": {"type": "string"}
  }
}
```

In order to test the POST or PATCH methods, an example command in the request body is in the format:

```
{
  "front": "Japan",
  "back": "Tokyo"
}
```

A successful test will return an output in the format:

```
Sun Nov 17 00:20:13 UTC 2019 : Method request headers:
{"Content-Type":"application/json"}
Sun Nov 17 00:20:13 UTC 2019 : Method request body before
transformations: {
  "front": "Japan",
  "back": "Tokyo"
}
Sun Nov 17 00:20:13 UTC 2019 : Request validation succeeded for
content type application/json
```

There is no entry in the request body for the DELETE or GET methods. A successful test for DELETE will return an output in the format:

```
Sun Nov 17 00:26:51 UTC 2019 : Method request headers:
{"Content-Type":"application/json"}
Sun Nov 17 00:26:51 UTC 2019 : Method request body before
transformations:
Sun Nov 17 00:26:51 UTC 2019 : Request validation succeeded for
content type application/json
```

A successful test for GET will return an output in the format:

```
Sun Nov 17 00:29:37 UTC 2019 : Endpoint response body before
transformations:
{"statusCode":200,"headers":{"Content-Type":"application/json","
access-control-allow-origin":"*"},"body":[{"\"front\": \"China\",
\"back\": \"Beijing\"},{\"front\": \"Japan\", \"back\": \"Tokyo\"}]}
}
Sun Nov 17 00:29:37 UTC 2019 : Method response body after
transformations:
Sun Nov 17 00:29:37 UTC 2019 : Method response headers:
{Content-Type=application/json, access-control-allow-origin=*,
X-Amzn-Trace-Id=Root=1-5dd09470-5cae4fd72ae83fa49;Sampled=0}
```

6. Alexa Voice user interface

Alexa's Voice user interface(2) comprises the Echo device. It has a front-end programming using Alexa Skills Kit console and a back-end using AWS lambda. It also allows for an HTTPS end-point. For this project, Lambda has been used as the preferred back-end. The resources are connected using a skill ID generated by the Alexa Skills Kit and an ARN generated by AWS Lambda.

Alexa Skills Kit facilitates control of a user's invocations, utterances and Alexa's responses to each one of them. It is also the forum where the end-points data is entered for inter-connection between servers. The features for publishing a skill, giving it a visual interface or peripheral connecting devices and testing the skill that a developer has created can be found in the ASK console.

The primary and essential usage is to build interaction models. The design of a dialogue between Alexa and a user is structured through Intents that are saved in the particular skill stored in the console. A function in the corresponding Lambda code is triggered each time an intent is called. The function code determines what Alexa's response is going to be. An Utterance from the user is what sets the whole mechanism in action. It is presented in figure 20 below.



Figure 20: General Voice Only Architecture Model that uses the Developer Console as the front-end and AWS Lambda as the back-end. They are connected using the ARNs generated by Amazon.

The model for an invocation is in the format of:

Alexa, <invocation phrase> <invocation name> --- <connecting word> <some action>

Alexa, tell <Quizwire> to <begin practice>

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

The various invocation phrases are: ask, tell, search, open, launch start, resume, run, load, begin, use, run, talk to, play. Some example connecting words are: from, using, for, about, if, whether, to, etc.

Chapter 4 - User Study Experiment

Research Questions

It is essential to gauge the intended effects of the application objectively. This assessment could be done by having users test the app and relay their experience regarding its usage. Some of the questions that required measuring and assessment are mentioned below. The user study has been conducted to give these answers a number and a measure of where the app stand in terms of its human computer interaction. Some of the research questions are:

1. Is the application user-friendly or does the design require more instructions and changes for a better user experience?
2. What are some features that were liked least by the users? (This question helps gauge the difficulty in question 1 above and improvisations can be made accordingly.)
3. Does the app perform the basic functionality of creating flashcards manually and practicing them using a voice interface?
4. Would the users re-use the app themselves or recommend it to a friend, if a newer version of it were to be available?
5. Does the design of the app play a role in the users' perspective in question 4 above?

User Study

A user study has been conducted to verify the quality of 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 technology professionals who mostly work at Intel.

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. A summary of the 13 research participants' demographics can be found in the images below. Figures 21, 22, 23 and 24 are related to the demographics of the research participants. These questions help gauge the level of technical know-how of the research participants.

Figures 25, 26 and 27 assess the participants' opinions regarding the rating of the app. The assessment is done for the overall app, ease of use and the look and feel of the application. Figures 28 and 29 gauge the users' returning probability; either by reusing the app or by recommending it to a friend. Five features have been considered for ranking by the participants. They are ranked in the order: 1- Ability to customize learning using audio, 2- Functionality, 3- Speed, 4- Design, 5- Ease of Use. Figure 30 represents the ranks of the features that the research participants found most valuable.

Questions	Average Ratings	Out of
What is your overall evaluation of this app on a scale of 1-5?	4.46	5
How easy is this product to use on a scale of 1-5?	3.62	5
How likely are you to recommend this app to a friend or colleague?	4.15	5
How satisfied are you with the look and feel of this product?	4	5

Question 1: What is your highest level of education attained?

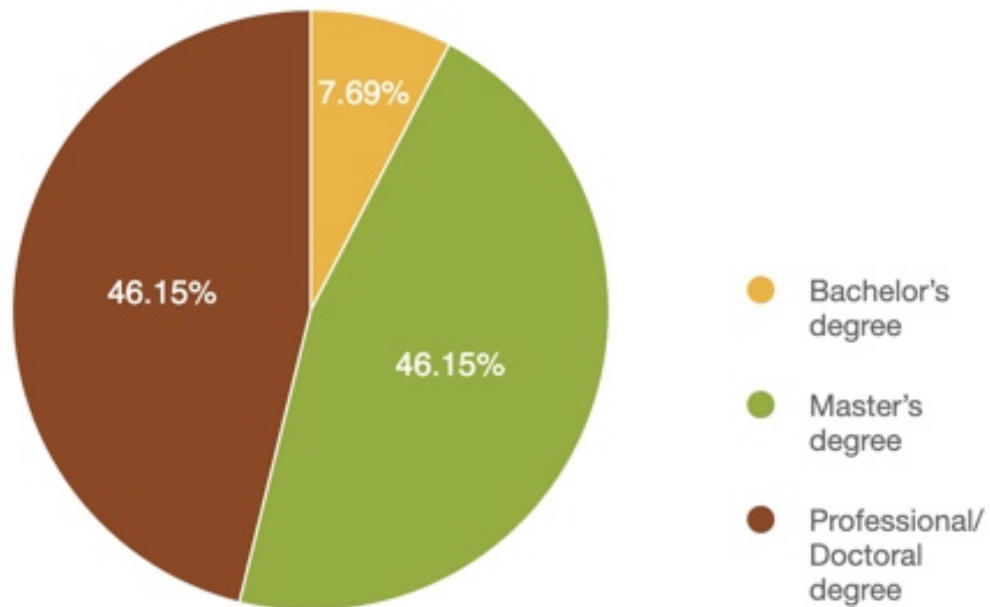


Figure 21: Highest level of education attained by participants

Question 2: What is the field in which you received your highest degree?

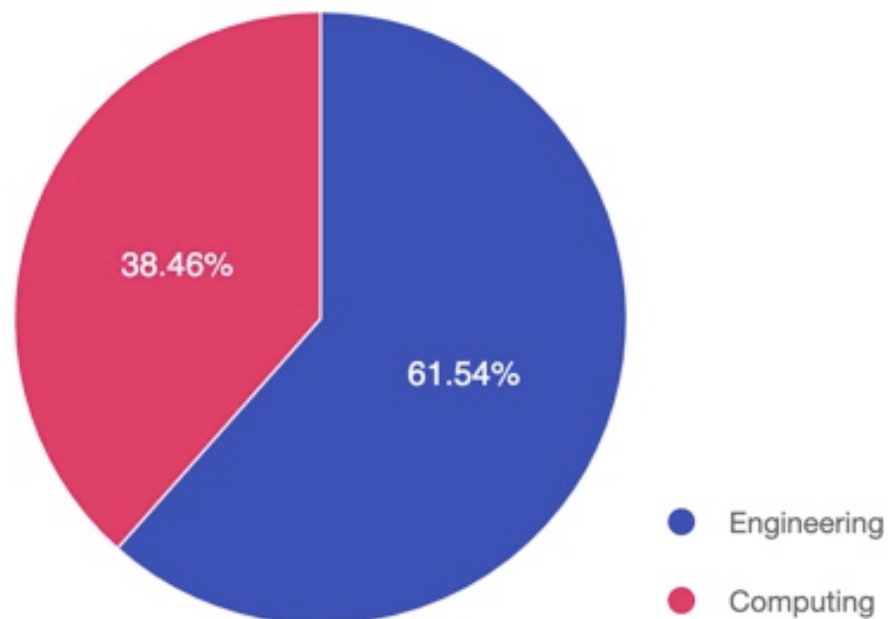


Figure 22: Field of education of highest degree

Question 3: What is the industry of your current occupation?

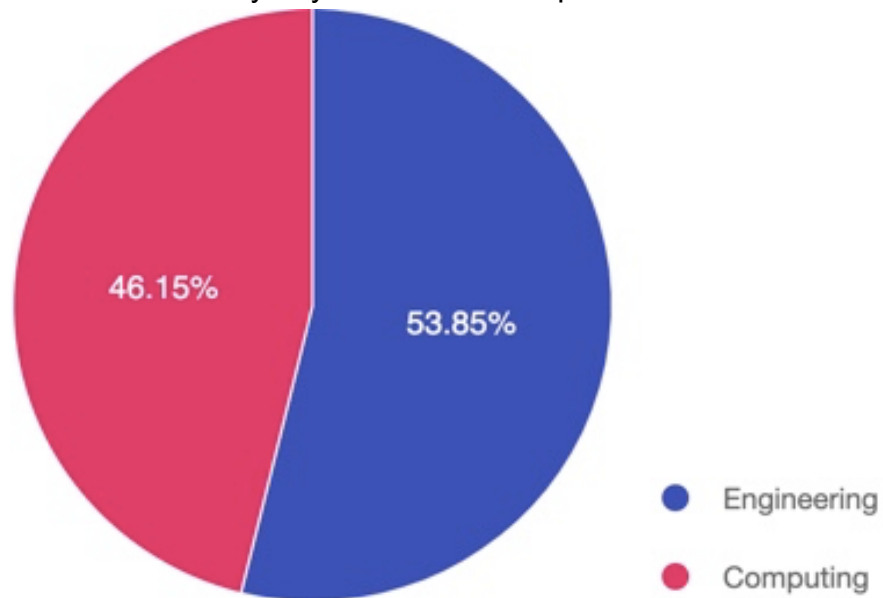


Figure 23: Industry of current occupation

Question 5: What is your overall evaluation of this app on a scale of 1-5?

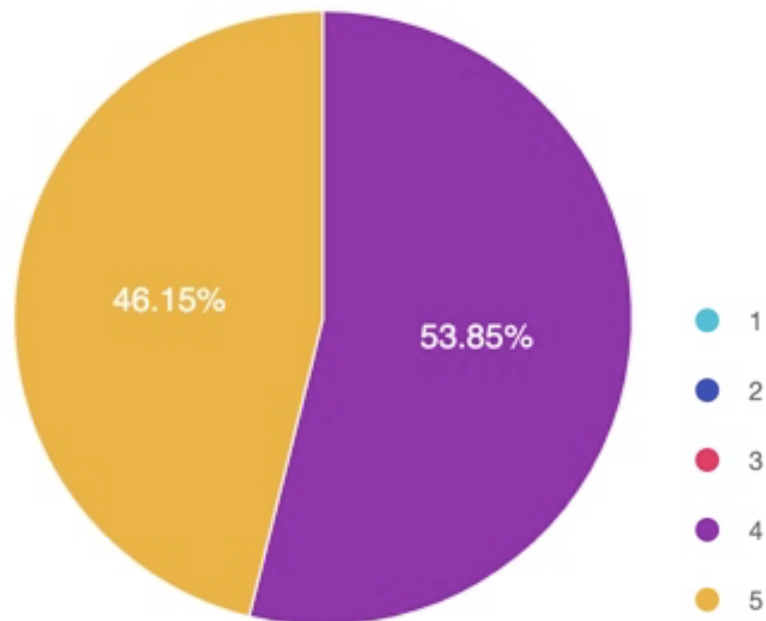
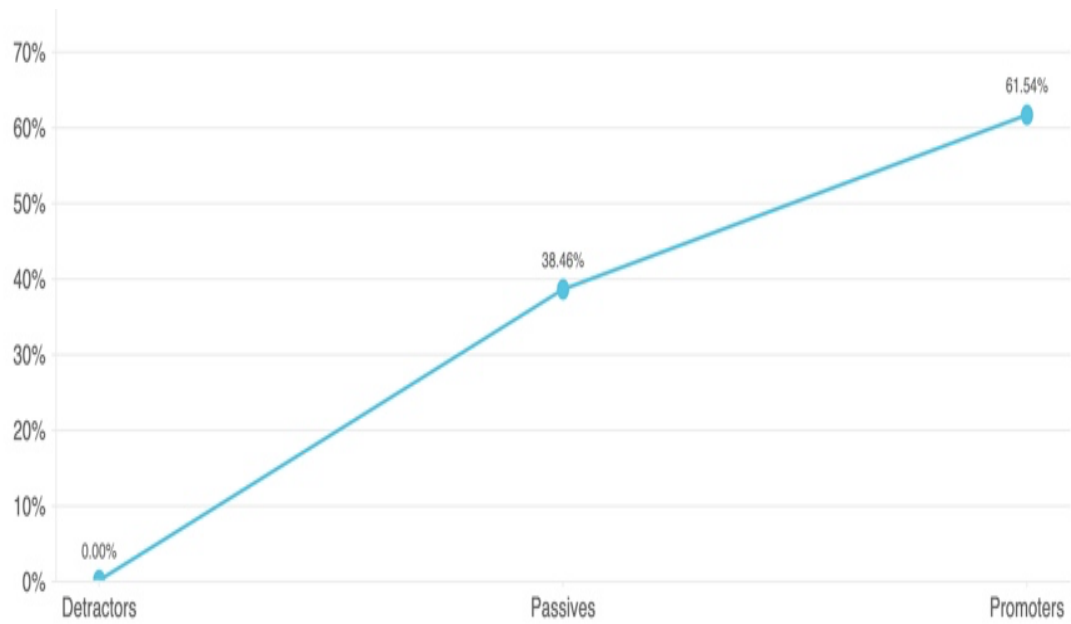


Figure 25: Overall Evaluation of the App on a scale of 1 to 5. 1 is the lowest, while 5 is the highest.

Question 4: On a scale of 1-10, how computer savvy are you?



(1) Detractors (0-6)	(2) Passives (7-8)	(3) Promoters (9-10)	Net Promoter Score
0	5	8	61.54

Range		Least frequent		Most frequent		Mean	Median	Standard deviation	Variance
From	To	Frequency	Value	Frequency	Value				
Passives	Promoters	5	Passives	8	Promoters	2.62	3	0.51	0.26

Figure 24: Level of technological expertise of participants

Question 6: On a scale of 1-5, how easy is this product to use?

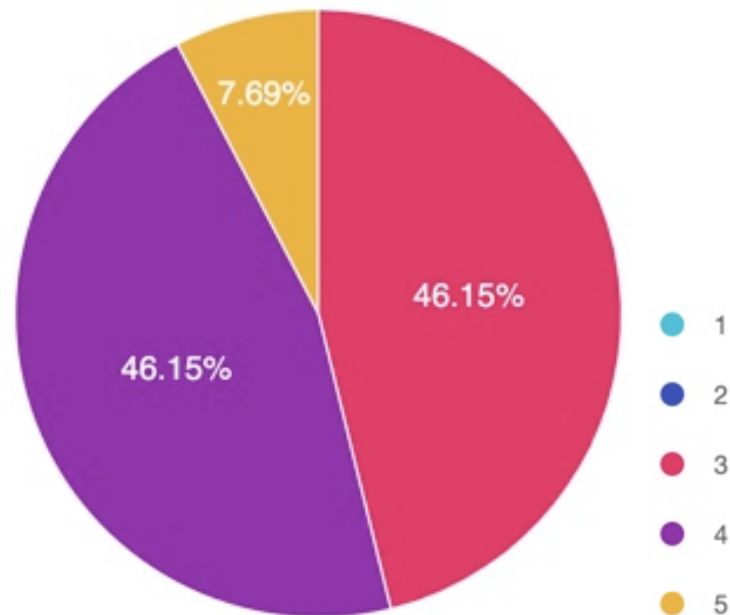


Figure 26: Ease of use of the App on a scale of 1 to 5. 1 is the lowest, while 5 is the highest.

Question 7: How satisfied are you with the look and feel of this product?

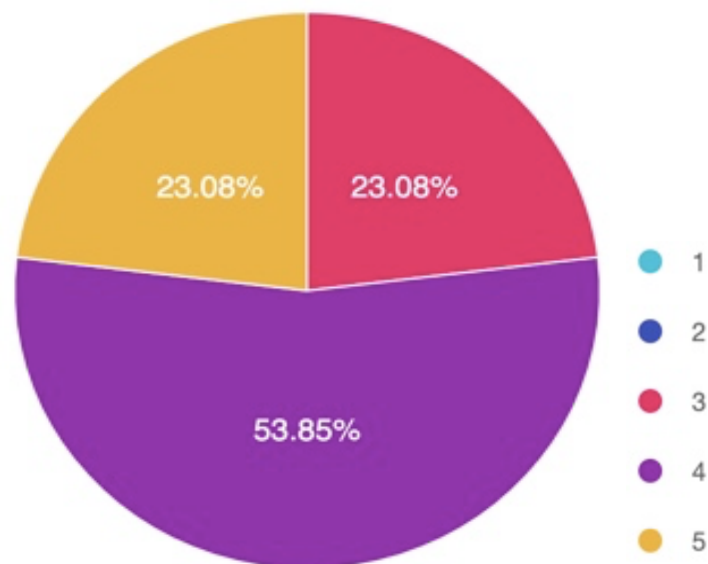


Figure 27: Look and feel rating of the App on a scale of 1 to 5. 1 is the lowest, while 5 is the highest.

Question 8: On a scale of 1-5, how likely are you to recommend this app to a friend or colleague?

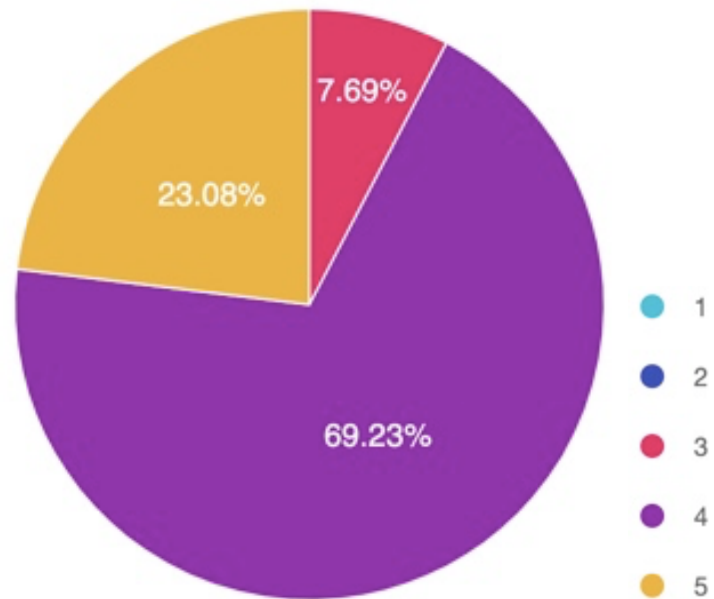
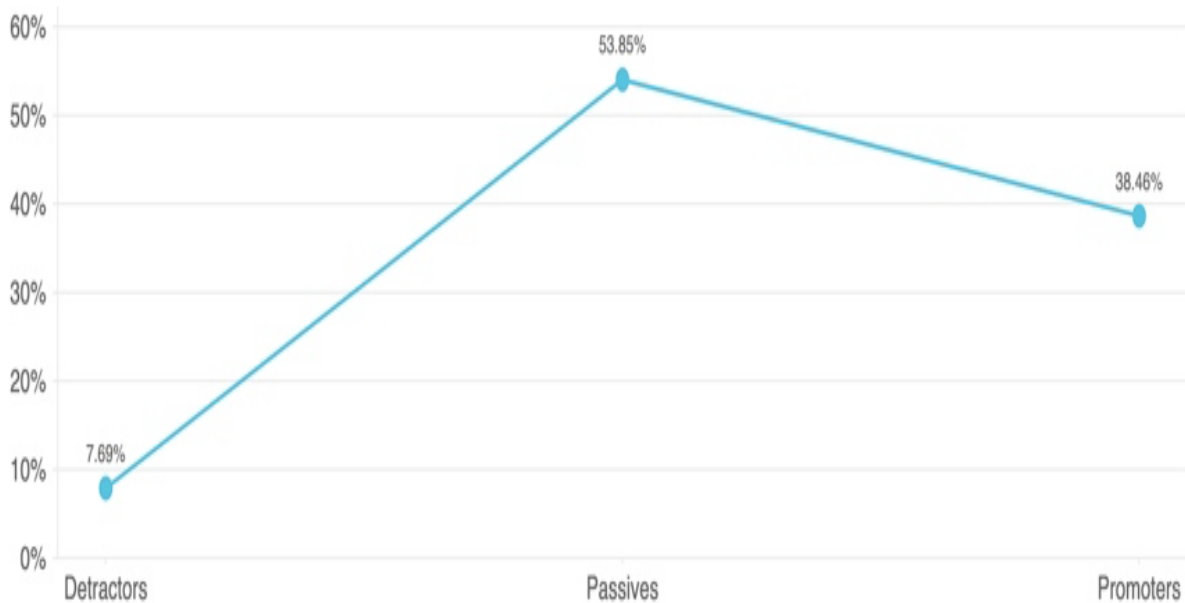


Figure 28: Likelihood of participants' recommending the App to a friend on a scale of 1 to 5. 1 is the lowest, while 5 is the highest.

Question 9: If a new version of this product were to be available, how likely are you to try it on a scale of 1-10?



(1) Detractors (0-6)	(2) Passives (7-8)	(3) Promoters (9-10)	Net Promoter Score
1	7	5	30.77

Range		Least frequent		Most frequent		Mean	Median	Standard deviation	Variance
From	To	Frequency	Value	Frequency	Value				
Detractors	Promoters	1	Detractors	7	Passives	2.31	2	0.63	0.4

Figure 29: Likelihood of participants' reusing the App after experiment on a scale of 1 to 10. 1 is the lowest, while 10 is the highest.

Question 10: Which features do you like best about this product? Select any 3:

- Design
- Ease of use
- Speed
- Functionality
- Ability to customize learning using audio

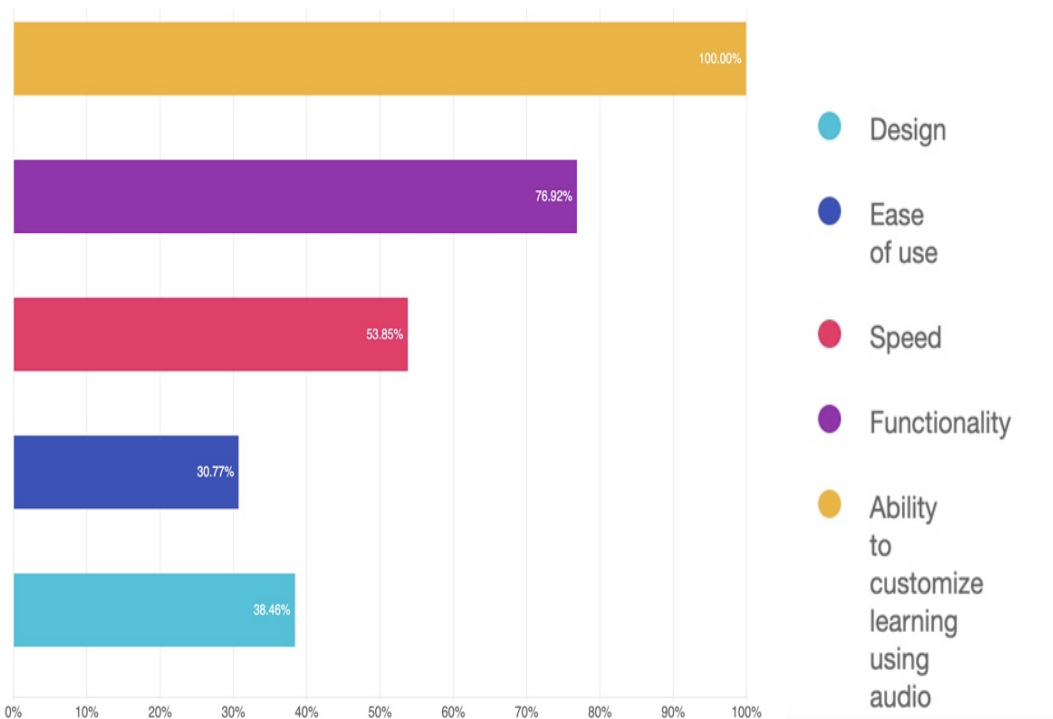


Figure 30: Most favorable features of the app as per the participants from an options set of : design (38.46%), ease of use (30.77%), speed (53.85%), functionality (76.92%), ability to customize learning using audio (100%)

Chapter 5 - Conclusion

Methods Used to Connect the Database Layer

Several methods have been considered to manage the MVC aspect of the web interface. A list of all the methods that worked out and did not is mentioned below.

Methods that Got Ruled Out

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

The first two approaches were rejected because of higher billing amounts on AWS. The third approach required tutorials and resources that were not available/popular at the time of this research. Though method four could have been a plausible solution, it is not a very popular technology as compared to the ones implemented. Tutorials and resources to learn this technology was relatively more expensive and time consuming. Method five got ruled out because it was difficult to estimate how to connect MySql with the AWS platform.

Though there may have been feasible ways to work with these ruled out technologies, the relatively easier choice for learning was picked. It was also a personal preference based on adaptability.

Methods that Worked

1. Design Web UI using ReactJS
2. External domain connects with DynamoDB using API
3. Alexa reads data from dynamoDB

The above mentioned methods were selected because they seemed promising. Quality textbooks and articles suggested this approach. That made it a safe bet in terms of the time taken to experiment and implement the said technology. ReactJS has plenty of resources to self-learn through an online platform. There are several discussion forums, tutorials and blog articles that can be found through a simple google search with this popular technology. Since the web technology was learned along the process of implementing and developing the project, relying on recommendations and industry best practices has been a huge time saver.

Next Steps

The next steps have been determined based on the user study survey results. The innovativeness, functionality and speed were rated as the best features. However, design and ease of use need more work.

1. Improve the website's user-interface and HCI component in such a way that it explains the usage instructions better. The design of the web UI can be made more visually appealing too.
2. Improve the Voice UI to make it more conversational. This can be done by making the flashcards more interactive. Removing the 'number' slot and finding a better solution to the slots challenge would be a good solution.
3. Deploy the final application developed.

Timeline

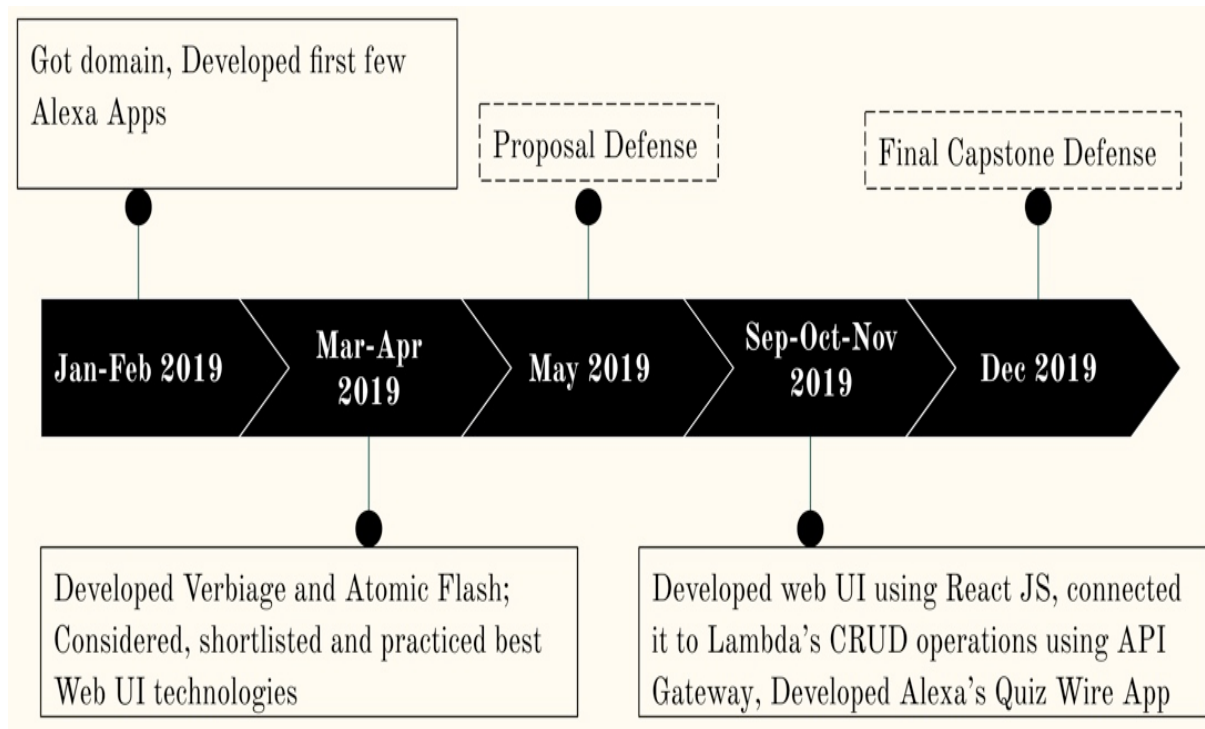


Figure 31: Project Timeline

Contribution

The entire project, including the three applications have been developed by the individual developer under the guidance of the advisor. No other application with a functionality similar to Quiz-Wire exists as on the date of this Capstone defense. Besides learning the skills on the job and implementing a combination of the various resources, tools and technologies, the major contribution is the innovation in designing and developing an app such as this.

References

1. Amazon Web Services. (2019). Build Skills with the Alexa Skills Kit. Retrieved from <https://developer.amazon.com/docs/ask-overviews/build-skills-with-the-alexa-skills-kit.html>
2. Amazon Web Services. (2019). Alexa Skills Kit. Retrieved from <https://developer.amazon.com/alexa-skills-kit>
3. Code Academy. (2019). Introduction to Alexa Skills. Retrieved from <https://www.codecademy.com/courses/learn-alexa>
4. Peterson, T. (2017). How to Create an Alexa Skill that Manages To-Do-Lists. (<https://medium.freecodecamp.org/how-to-create-an-alexa-skill-that-manages-to-do-lists-11c4bab29ea5>)
5. Alexa Developers. (2019). Developing Alexa Skills from Scratch. Retrieved from https://www.youtube.com/playlist?list=PL2KJmkHeYQTNwIZqLh_ptZhSNZf93e8Sp
6. Big Nerd Ranch. (2018). AWS Alexa Skills Kit. Retrieved from <https://developer.amazon.com/alexa-skills-kit/big-nerd-ranch>
7. Amazon Alexa Training Resources. (2017). Alexa Developer Training. Retrieved from <https://developer.amazon.com/alexa-skills-kit/tutorials>
8. Amazon Web Services. (2018). Alexa Github Repository. Retrieved from <https://github.com/alexa>
9. Python Programming. (2017). Alexa Skills with Python. Retrieved from <https://pythonprogramming.net/intro-alexa-skill-flask-ask-python-tutorial/>
10. Poccia, D. (2018). AWS Lambda in Action. Manning Publications. Manning.
11. VOA News. (2017). Build Muscle Memory to Improve Your Pronunciation. Retrieved from <https://learningenglish.voanews.com/a/build-muscle-memory-to-improve-your-pronunciation/3918000.html>
12. AV Speech Therapy (2016). Speech Muscle Memory. Retrieved from <http://www.avspeechtherapy.com/2012/06/22/speech-muscle-memory/>
13. Speech Improvement. (2018). Creating Muscle Memory. Retrieved from <https://www.speechimprovement.com/creating-muscle-memory/>
14. Kushnoor K. (2019). Flashcards Powered by Alexa - Capstone Proposal Report presented at UNCW