MSc Computer Science Thesis: Leonard Goldschmidt

*Smart Assistant for Data Collection in Epidemiology*

1. Main Body

# Aims and Objectives

Meeting the aims and objectives listed below should result in successful completion of this project. For clarity purposes, the aims are split into high level and low level. High level aims focus on the overall outcome of the study, whereas low level aims focus more on application development. The minimum work required is laid out in the key deliverables section.

* + 1. High Level Aims

The study is centered around epidemiology as well as utilizing modern technology for the relevant research. Therefore, the high-level aims are:

1. To determine if it is possible and how to implement and application that is able to run on an Amazon Alexa and/or Google Home device, as well as understanding the back-end differences for both platforms.
2. To build an application, runnable on a smart assistant, that allows a user to food and/or drink data by saying phrases like *“For lunch I had an apple and a diet coke.”*
3. To determine if an application, running on a smart assistant, can provide an alternative food logging technique for epidemiological research.
   * 1. Low Level Aims

These low-level aims are introduced to give clear guidelines for necessary development milestones for a successful project implementation. The low-level aims are:

1. To create a custom skill for a smart assistant that can translate verbal input and interpret food information.
2. To create a method of storing the food information in a centralized database.
3. To be able to distribute the skill to multiple users.
4. To assess this alternative food dairy method in comparison with traditional methods used in epidemiology.
   * 1. Objectives

The objectives of this study are:

1. To evaluate and compare *Amazon Alexa* and *Google Home* in terms of criteria including capabilities, performance, limitations, and back-end implementation options.
2. To determine how interchangeable back-end systems developed for these two devices are. For example, whether the same back-end system be used for both devices, or some degree of adaptation between them, and how to minimize any needed adaptation through appropriate software design.
3. To prototype an implementation of a voice-driven food diary system for collecting food and drink information from participants in research studies. Ideally, the system should be designed such that it can be integrated with both *Amazon’s Alexa* and the *Google Assistant*. If the project demonstrates that this is not feasible then *Alexa* or *Google Assistant* will be chosen, depending on which was deemed more appropriate from evaluation of objective 1.

# Key Deliverables

For successful completion of this study, a set of minimum key deliverables are set as follows:

1. An investigation of similarities and differences when programming applications on Amazon or Google developer platforms.
2. An application running on a smart assistant that enables translating and interpreting speech and storing key food data.
3. A study report with detailed information about the process throughout development of the application.
4. Background and Context

Health research is interested in understanding factors that influence people’s food intake and how different diets affect health. Collecting reliable and accurate dietary information is a difficult process in this area and can result in inconsistent and poorly evidenced data for epidemiological research.

*“By definition, epidemiology is the study (scientific, systematic, and data-driven) of the distribution (frequency, pattern) and determinants (causes, risk factors) of health-related states and events (not just diseases) in specified populations (neighborhood, school, city, state, country, global).”[[1]](#endnote-1)*

History provides multiple studies that show diseases that appeared as result of a certain diet. Most commonly, the affected had certain deficiencies in their diet. Traditional methods of identifying diseases caused by specific diets involve asking participants to complete food frequency questionnaires either on pen and paper, mobile or web applications. Methods like these require significant time and effort, possibly resulting in either inconsistent or inaccurate data. Reason being, that participants may enter wrong information, miss to log meals, or lack effort due to the tedious and time-consuming logging process. Therefore, an alternative method of collecting food data should be assessed. It may be possible to utilize arising technologies such as voice assistants to build an application that enables tracking food intake by simply speaking to a smart speaker. This process may come more natural to participants, thus, potentially being quicker than traditional methods. This section will describe cases of traditional epidemiological research and how effective the given nutrition logging techniques were.

A study carried out by James Lind in 1753[[2]](#endnote-2) showed that lemons and oranges had positive effects on the illness. Ultimately, scurvy was found to be a vitamin C deficiency. In the late nineteenth century the illness *Beriberi*, most commonly found with infants, occurred with a group of sailors, whose diet largely consisted of polished rice. Scientist *Takaki* found that the disease was eliminated by adding milk and vegetables to their diet. The cause was later tracked back to a lack of thiamine, which is a vitamin found in milk and vegetables. In 1987, a study showed that the common occurrence of *Kesahn* disease in central China, was caused by a selenium deficiency. It becomes apparent that nutritional epidemiology relates commonly to deficiency type diseases. These diseases usually take months of starting a deficient diet to become apparent and may be cured quickly by increasing the deficient vitamin amount. It may therefore be ideal if the nutritional reason for a deficiency within a large group can easily be identify. Thus, an easy and centralized way of tracking nutrition for a large group is needed. In many epidemiological studies, existing data is used. However, these studies often incorporate a small group of subjects. For accurate results, the field data needs to be clean, valid, reliable and should not take long to gather. Traditionally, collection of this type of data involves questionnaires that are handed to the subjects and filled in manually over a period of time.

*“The second generation of the Avon Longitudinal Study of Parent and Children”[[3]](#endnote-3)* involves logging and tracking behavioral data, including nutrition, of over 14,000 participants since their birth in 1992. Having such a large group of participants and collecting a wide-range of social, lifestyle, clinical, anthropometric and biological data on all family members repeatedly, requires reliable collection methods. As this study involves so many different areas of data, separate methods of collection can be investigated. Epidemiological studies have shown that nutrition can affect health majorly and thus provides an important section of the Avon Longitudinal study. The research was set up to provide a unique multi-generational cohort and represents the potential scale that is sometimes needed for epidemiological studies. The study has been ongoing for over 26 years and is now a study ranging over three generations.

1. Development

# Introduction

In order to successfully create an application that is runnable on a smart assistant device, that can be distributable to many users, as well as collecting and story data in a centralized database, several components were necessary. The diagram below (Figure 1: Full Stack Application) provides an overview of the stack used to implement the system. The application stack consists of several different parts and to understand it fully, one first has to define the individual components (from right to left):

1. **User**
   1. The users only responsibility is to speak to the smart assistant and tell it to log certain food and/or drinks items.
2. **Smart Assistant**
   1. The smart assistant (Echo Dot for this study), running a custom Alexa Skill, waits for the correct invocation phrase and then interprets the users’ phrase.
   2. The custom Alexa Skill populates custom dietary slots. The skill was trained using sample phrases, so that it populates the slots with the right information and prepares a JSON object.
3. **Web Connectivity**
   1. The smart assistant sends the JSON Object to a configured endpoint. Jojo.epi.bris.ac.uk:9449
4. **Server**
   1. The endpoint is configured to speak to the server jojo.epi.birs.ac.uk, which is located at the University of Bristol. This server contains the backend logic, including the database connectivity.
5. **Docker**
   1. *“Docker is a tool designed to make it easier to create, deploy, and run application s by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.”[[4]](#endnote-4)*
   2. The server contains a docker cluster, running different applications. A docker image was created containing the back-end logic of this application.
6. **Docker Image**
   1. The docker image has its own stack containing the following components:
      1. *Apache Tomcat Server*
         1. *“The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies”[[5]](#endnote-5)*
         2. This server runs and deploys the java application.
         3. The server is configured to support SSL (https), which increases security and is necessary to work with Amazon skills.
      2. *Java Application*
         1. This application contains the bulk logic of the web application.
         2. Intent-handlers handle the triggered intent that was sent to the server from the smart assistant via the JSON object.
         3. The handlers implement logic such as storing the data or returning useful responses back to the smart assistant.
         4. To connect to the Alexa skill, a java servlet is used, containing the skill ID.
7. **Database**
   1. To close the stack loop, the nutrition data needs to be stored in a save location.
   2. A *mySQL* database, running on the server, is used to store the nutrition data for each user.

By putting all these components together, it is possible to provide a full stack application that allows a user to use a skill in a food diary like fashion. However, due to the integrity of the full stack, each component required basic proof of concepts, which later increased in complexity to implement full desired functionality. The following section provides a detailed explanation of each development step.



Figure 1: Full Stack Application

# Building a basic Alexa Skill

To get familiar with developing skills for an Alexa supported device, a simple custom skill was set up, with no custom functionality. There mains steps needed to create a custom skill are:

1. *Create an invocation name*

The name given to a custom skill, also acts as the invocation name. This means that the custom skill can be lunched by saying “*Alexa, open {invocation name}”*.

1. *Create Intents*

An intent represents a functionality of the skill. For example, if the skills purpose is to book a flight, one intent might be called “*bookFlightIntent*”, which processes the speech input and populates slots with the necessary flight information. A skill can have any number of intents. A custom skill comes with five pre made built in intents, provided by Amazon. These are basic intents that handle simple operations such as closing the app or providing help. To ensure the skill understand which and when a certain intent should be selected, the developer should provide a list of sample utterances. A minimum of thirty different utterances should be given. To ensure the correct slot types are populated, the slots can be included in a sample utterance. See Figure 2: Sample Utterance for an example of such utterance. Slot types can be included by wrapping them in curly brackets “{}”.A close up of a logo

Description automatically generated

Figure 2: Sample Utterance

1. *Create Slot Types*

A slot type represents a value that can be populated by the user. Taking the flight booking intent example, one may require a slot “*departureCity*” and slot “*destinationCity*”. To ensure that the skill understands what kind of values may go into these slots, the developer should provide each slot with a large set of sample values. Alexa uses these sample values to train the model once the user has finished configuring the skill.

1. *Set Endpoint*

To publish the skill and make it available for testing, an endpoint needs to be set. This should generally be the URL that connects to the server, which handles the back-end logic. There are two options for the endpoint. The recommended one is *“AWS Lambda ARN”.* Essentially, this means hosting the skill on the Amazon cloud. To use a custom endpoint, the second option “HTTPS” should be chosen. For basic skills and to test basic functionality, provides a more viable option. Later on, throughout development for this study, this was changed to using HTTPS.

1. *Build Model and Test Skill*

Once the previous steps have been configured, the model can be built. To test the skill, the Amazon developer portal provides a test tab. The user can either user the microphone on the computer, or manually type, utterances to test the skill (see Figure 3: Alexa Test Tab).



Figure 3: Alexa Test Tab

The five steps above are required to build an Alexa skill. For the purpose of this study a skill has to be created, that enables logging food and drink data. A detailed explanation of the skill setup can be found in the next section.

# Creating Food Diary Alexa Skill

Before creating any intents or slots, a storyboard was created. The storyboard represents a simple flow of conversation a user may have with the voice assistant. The storyboard helps shape the intents and slot types needed for a successful skill implementation that makes conversation as natural as possible. Using a well-developed storyboard, it is possible to simulate turns in a conversation between a user and the voice assistant. Figure 4: Storyboard shows the initial storyboard created for this application. Alexa responses are shown in blue. Creating of a storyboard aids with not only identifying correct conversation flow, but also realizing possible false information that a user might give, as noted in the column on the far right in Figure 4: Storyboard.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| “Alexa, open my food diary! “ | “Alexa, tell food diary that I just had a bagel. “ | “Alexa, tell food diary to log a drink.” | “Alexa, open food diary and log a roast and 3 glasses of wine for my dinner.” | “Alexa, open food diary and book me a flight.” |
| “Welcome to your food diary. You can use me to log food” | “That’s logged for you.” | “Alright, what did you have to drink?” | “Ok, I’ve logged that for you.” | “Sorry, I don’t know that one. You can use me to log food. Say ‘help’ for further information”.! |
| “Ok, I just had porridge with peanut butter and a coffee for breakfast.” |  | “A bottle or water” |  |  |
| “Cool, that’s been logged for you! Anything else you would like to log?” |  | “Cool, I logged a bottle of water for you.” |  |  |
| No! |  |  |  |  |

Figure 4: Storyboard

By using the storyboard, it becomes apparent that the skill should be able to log food items, drink items separately as well as together. Three separate intents are needed for this.

* + Log Food Intent
  + Log Drink Intent
  + Log Food and Drink Intent

To create sample utterances for each intent, slot types are needed. The slot types are the information slots that are eventually sent to the database server. At this point it is essential to decide what information should be stored inside the database. To do this, the larger picture needs to be taken into account. Who will use this skill? How many users will use the skill? How will the users be distinguished? Is the amount of food and drink taken into account? Thus, the following slot types, with some of the following samples have been created to ensure full visibility when storing nutrition data in the mySQL database.

* + **Food**

*Samples: Apple, pasta, bread, bagel, eggs, avocado, burger, chips, fish, salad, tuna, tuna salad, Greek salad, chicken curry*

* + **Food Amount**

*Samples: Small, medium, largen, plate, one plate, two plates, one serving, two servings, bag, one bowl, two bowls*

* + **Drink**

*Samples: Water, juice, coke, diet, coke, Fanta, soda, beer, coffee, tea, red wine, cider, iced coffee*

* + **Drink Amount**

*Samples: 1,2,3, small, medium, large, cup, one glass, one cup, can, bottle*

* + **Meal**

*Samples: Snack, breakfast, lunch, dinner*

* + **User ID**

*Samples: 12345, 38582, 11111, 23456*

The samples given for each slot above, are not a complete list of those used for training the final skill. Once the slot types are defined, they can be used to create sample phrases for each intent. The algorithm behind Alexa uses these for training the model to a standard where it can understand similar utterances and trigger the correct each intent. This has its limitation which will be discussed later. Sample utterances are phrases that a user might say to trigger a certain intent. Below is a list of the intents along with an incomplete list of the sample utterances used to train the model. The slot types are combined with the utterance suing curly brackets {SLOT\_TYPE}.

* + **LogDrinkIntent**
    1. Sample utterances:
       1. *I just had a {drink}*
       2. *Log a {drink}*
       3. *I just had {amount\_drink} {drink}*
       4. *For {meal} I had a {drink}*
       5. *For {meal} I had {amount\_drink} {drink}*
       6. *Log {amount\_drink} {drink}*
  + **LogFoodIntent**
    1. Sample utterances:
       1. *I just had {food}*
       2. *I just had {amount\_food} {food}*
       3. *Log {food}*
       4. *Log {amount\_food} {food}*
       5. *For {meal} I had {food}*
       6. *I had {amount\_food} {food} for {meal}*
  + **LogMealIntent**
    1. Sample Utterances:
       1. *For {meal} I had {food} and a {drink}*
       2. *I had {food} and {drink}*
       3. *I had {food} and {amountDrink} {drink}*
       4. *For {meal} I had {food} and {amountDrink} {drink}*
       5. *{meal} I am going to have {food} with a {amountDrink} {drink}*
       6. *Let me log a meal*
       7. *For {meal} I had {amountFood} {food} and {amountDrink} {drink}*

For a complete list of the sample utterances und sample slot values, see APPENDIX. It is important to note that the log drink intent only has the slot type {drink} and not the slot type {food}. This is necessary to avoid triggering a wrong intent when the model detects a {food} value. The opposite is true for the log food intent and the log meal intent is only triggered when both {food} and {drink} values are available. This enables the user to log drinks and foods individually, as well as together. To ensure that the backend application, running on the server, gets provided with the correct and complete information, slot values can be set to being required for a certain intent. The user ID, for example, is essential for storing the data in the correct space and to distinguish users. In order to ensure that the database on the server is populated with correct and all data, it was decided to set each slot to be required for all intents. The developer console enables this, and speech prompts can be customized in case a slot had not been filled. It is to note, how some sample utterances in the intents above, don’t incorporate all slots and therefore expects a speech prompt from Alexa to ask for the missing slots. Below is a list of the speech prompts given to each slot, along with possible answers by the user to fulfill the corresponding slot:

**Log Drink Intent:**

* + Drink
    1. What did you have to drink?
       1. {drink}
       2. I had a {drink}
       3. {amount\_drink} {drink}
  + Amount\_drink
    1. How much did you drink?
       1. {amount\_drink}
       2. I had {amount\_drink}
       3. {amount\_drink} {drink}
  + Meal
    1. When did you have this drink?
       1. For {meal}
       2. {meal}
       3. Just now
  + User ID
    1. Please confirm your user ID
       1. {userID}

**Log Food Intent**

* + Food
    1. What did you eat?
       1. {food}
       2. {amount\_food} {food}
       3. I had {food}
  + Amount\_food
    1. How much of this food did you have?
       1. {amount\_food}
       2. I had {amount\_food}
       3. {amount\_food} {food}
  + Meal
    1. When did you eat this?
       1. For {meal}
       2. At {meal}
       3. {meal}
  + User ID
    1. Please confirm your user ID
       1. {userID}

**Log Meal Intent**

All speech prompts from the intents above are used here.

**Testing the intents without the back end**

To ensure that the trained model understands the user and that the correct intents are triggered, the developer console provides the “***Utterance Profiler”.*** The utterance profiler takes typed speech input and provides responses based on the intents that the model was trained for. This provided a vital tool to ensure that throughout each step of conversation the correct slots were populated. When providing an utterance, the profiler pulls out the available slot values and if any are missing, continues the conversation by prompting for these. An example of a step by step conversation can be seen in the figures below. After each prompt the skill populates the slots with the corresponding values it prompted for.

Figure 5: Utterance Profiler (1) show the first part of a conversation to log a food item and a drink item. The algorithm selects an intent based on the trained model. As both, food and drink are available in the users utterance, the *LogMealIntent* was triggered. In the lower section, the profiler shows which slots have been filled and which have not. In this example, the slot ‘meal’ and ‘userID’ have not yet been filled. Therefore, Alexa prompts the user for the first unfilled slot ‘meal’. Alexa waits for a reply and once given, fills the corresponding slot. Figure 6: Utterance Profiler (2) shows this interaction and the filled slot. The final slot to be filled is the ‘userID’ and thus the user is prompted. The final slot values after a conversation may look like shown in Figure 7: Final Slots with Values. This are the values that would be packaged in a JSON object and sent to the configured endpoint.



Figure 5: Utterance Profiler (2)



Figure 6: Utterance Profiler (1)



Figure 7: Final Slots with Values

However, looking closely at certain slot values, they do not make complete sense. The value for ‘*amountFood’* shown in Figure 7: Final Slots with Values is ‘a’ and the ‘*amountDrink’* is filled with ‘a cup of’. These values are not a logical representation to the corresponding slot. These values materialize from the flow of natural conversation by the user. Alexa needs to be able to deal with these responses. To do so, the sample value for each slot can have synonyms. In Figure 7 the given ‘*amountDrink’* value would be a synonym for *‘one cup’,* which represents a better measure of a drink size. Synonyms can be provided for each sample slot value as shown in grey in Figure 9: Sample Slot Value with Synonym. The more synonyms given to a slot value the more robust the skill model should be. Even with the synonyms the initial populated slot value will be the exact answer given by the user as seen in Figure 7. Pulling out the correct value using synonyms is done on the back-end side, which will be explained in a later section. Another example for this concerns the slot ‘meal’. The phrase to prompt the user for this slot is along the lines of “*When did you have this?”*. This might lead the user to respond with a time rather than one of the pre-defined slot values. Therefore, as many synonyms as possible should be created to ensure that the skill understands the user. These can be seen in Figure 8: 'meal' Slot with Synonyms.



Figure 8: Sample Slot Value with Synonym



Figure 9: 'meal' Slot with Synonyms

For certain slots only a few values are desired and should be allowed. For the slot ‘*meal’* the only values that should be allowed are ‘snack’, ‘breakfast’, ’lunch’, ’dinner’. Even though many synonyms have been created to account for many different user inputs, the right one cannot be guaranteed. It is possible to set validation rules for a slot type present within an intent. The three possible validation rules are:

* + Accepts only a set of values
  + Reject only a set of values
  + Accept only Slot Type’s values and synonyms

The last rule was applied to the ‘meal’ slot to ensure that only one of four possible value would be sent to the back end. To successfully create a validation rule, a response needs to be configured in case of it being triggered. Figure 10: Triggered Validation Rule shows an example of a conversation with the validation rule having been called.



Figure 10: Triggered Validation Rule

Once prompted the skill waits for a reply by the user until a valid response has been given.

By continuously populating the intents with sample utterances as well as the slot types with sample value, increase the robustness of the skill. However, the skill needs to be able to handle speech input by the user that does not match an intent. Alexa provides several built-in intents, including the ‘*FallbackIntent’,* which is triggered when the phrase does not match any intent. The backend can be configured to provide a useful response once this intent is triggered. Using the utterance profiler, this intent can be tested by saying a phrase that does not match an intent, however the profiler cannot return a response. An example can be seen in Figure 11: FallbackIntent, where the ‘*FallbackIntent’* is selected.



Figure 11: FallbackIntent

Having configured these intents, slots and utterances, the Alexa skill needs an endpoint to communicate with. In order to create an endpoint, a web server is needed.

# Creating a custom Web Server

A custom web server is needed to host the custom Alexa skill as a web service. For security purposes the following requirements are needed to host an Alexa skill as a web service[[6]](#endnote-6):

1. The service must be accessible over the internet.
2. The service must accept HTTP requests on port 443.
3. The service must support HTTP over SSL/TLS, using an [Amazon-trusted certificate](https://wiki.mozilla.org/CA/Included_Certificates). Your web service's domain name must be in the Subject Alternative Names (SANs) section of the certificate. For testing, you can provide a self-signed certificate.
4. The service must verify that incoming requests come from Alexa.
5. The service must adhere to the [Alexa Skills Kit interface](https://developer.amazon.com/docs/custom-skills/request-and-response-json-reference.html).

As Java was chosen for the back-end programming language for this study, the toolkit ‘*ASK SDK for Java’* was required. Amazon recommends usage of java servlets to publish the skill logic. A popular implementation to host Java servlets is ‘*Apache Tomcat’.* By definition “*The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies*”.[[7]](#endnote-7)

1. (*What is Epidemiology? | Teacher Roadmap | Career Paths to Public Health | CDC*, no date) [↑](#endnote-ref-1)
2. (Willett, 2013) [↑](#endnote-ref-2)
3. (Northstone *et al.*, 2019) [↑](#endnote-ref-3)
4. (*What is Docker? | Opensource.com*, no date) [↑](#endnote-ref-4)
5. (*Apache Tomcat® - Welcome!*, no date a) [↑](#endnote-ref-5)
6. (*Host a Custom Skill as a Web Service | Alexa Skills Kit*, no date) [↑](#endnote-ref-6)
7. (*Apache Tomcat® - Welcome!*, no date b) [↑](#endnote-ref-7)