Tech Feasibility Final

Apr 1, 2022

Project MealWrite

Team Rat in a Hat:

Charles Saluski Matthew Martinez Krystian Bednarz

Clients: Jennifer Hoffmann and Martin Sector

Mentors: Anirban Chetia & Tomos Oliver Prys-Jones

Table of Contents

1. Introduction

The aim of Team Rat in a Hat is to develop MealWrite, which is an Alexa skill and companion mobile application, whose goal is to streamline the home-cooking process to make it more convenient, accessible and time-efficient. The majority of people enjoy home-cooked meals. Whether it be a pastime, a health initiative, or a love for favorite family recipes, there is a satisfaction that comes from creating a delicious meal from scratch. One of the many problems with cooking is the process of making the food, especially when following an unfamiliar recipe. A few problems with home-cooking include the hassle of creating an ingredient list, and failing to correctly prioritize different parts of the meal and the associated time inefficiencies when missed, especially when the recipe is new or complex. Cleaning is often left until the end of the meal, however there may be windows of time during preparation for this, which reduce the post-meal cleanup time. Inefficiency and complexity deter many people from cooking.

The clients of project MealWrite brought these issues to light, and plan on using their capstone team's software development skills to help solve these issues. Jenny Hoffmann and Martin Sector are individuals who have experienced cooking issues in their daily lives, especially when it comes to balancing work with food preparation for others. Jenny's children are an example of those who are unable to adequately feed themselves because of their young age, and require someone to give them their daily food related needs. Combine this requirement with the busy lifestyle Jenny has with her work schedule, and it becomes readily apparent how cooking in a small time frame can be a burden.

Current solutions to the problems that come with the cooking process often involve removing that process altogether, like going out to eat at a sit-down restaurant, ordering fast food, buying frozen meals, or having food delivered. These options often have the downside of additional cost and ambiguity as to the nutritional value of the meal, as well as potential delivery delays, failures, or order errors. Other digital home cook assistants exist, but simply do not adequately solve the core issues cooking involves in an adequate manner. Such existing mobile apps make prior preparations to cooking a bit less difficult, but there is no currently available mobile app that intelligently solves the problems of time management. These existing apps lack the capability to intelligently schedule the cooking steps themselves, leading to many inefficiencies in the cooking process. This is the main core goal that MealWrite aims to achieve, while also providing a platform for future developments, including monetization and more features that will put it even further ahead of its competitors.

Team Rat in a Hat's solution is a cooking assistant Alexa skill that compiles lists of ingredients, provides spoken instructions to the cook with appropriate timing, and reduces time inefficiencies to prevent steps from being missed. Ingredient handling would include a means for the user to select what meals they wish to prepare at certain times, providing optional features of both developing a shopping list, and recipe recommendations based on what ingredients the user already has available. A mobile companion app will also be developed, allowing the user to schedule future cooking operations and search for recipes. As for the recipes themselves, the Alexa skill and companion app will have access to a database containing recipes with their corresponding steps, rerouted to be as optimally timed as possible. As the project aims to be foundational in scope, the recipes that will be available at the time of project completion will be limited, but will be ample for the current development and testing purposes. Timing for how long a recipe will take, when a recipe will reach completion, and how long the cleaning process afterwards will take will be determined. This is to ensure the completion of the cooking process can occur in predetermined time frames, giving the users the best means to plan for their current schedule accordingly.

The development of these software systems will aid in the cooking process, eliminating and mitigating the issues explained above with the other extant systems. Developing these solutions will take a significant investment of time and will have a number of challenges that need to be addressed. This document will serve to identify these initial challenges and find solutions to address them. Team Rat in a Hat is confident that through this early investment into resolving these challenges that the overall development process will proceed much more smoothly. These challenges are identified next, and analyzed to find solutions later in the document.

2. Technological Challenges

Team Rat in a Hat has recognized several challenges that need to be addressed because of their key relevance to development of the project, including selecting a service host for the backend services, selecting an optimal programming language to use to create MealWrite’s companion mobile application, selecting an optimal storage solution for the recipes and user data, and selecting an optimal programming language to use to implement MealWrite’s skill sessions and backend server.

1. The backend service selected will enable the development of the MealWrite software by supporting the envisioned service modalities, and having easy interoperability with Echo devices.

2. The language for the companion mobile application will allow rapid development of the initial prototype while also offering room for future development.
3. The storage solution for recipes and user data must be easy to use, able to scale to a large number of users, and prove cost effective during both development and deployment.
4. The language for the backend and skill sessions must be effective at connecting to all other parts of the architecture and scaling up to match the user demand for the service.

Because Alexa is prescribed as the target smart home assistant, all of the chosen technologies must be able to interface with it without much difficulty, so this is heavily considered in these analyses. The challenges identified above are now explained in more detail.

Backend Service Host

The first challenge identified by Team Rat in a Hat is selecting an appropriate backend service host. This will highly influence the selections made in the next sections, as selecting tools that integrate with the selected backend service host will greatly ease the development process. The Alexa skill, backend, and database will all be hosted by this provider, possibly using different services under the provider, depending upon the available options. It will ideally offer a short term serverless compute platform to run each user's Alexa skill session on, as this will enable scaling this service to easily meet demand. The service must also offer a range of available servers for which to host the backend on, and either offer a dedicated database service, or offer server options which are optimized for hosting databases on. Finally, the service provider must also allow easy options for connecting with Echo devices, as if this is difficult it would greatly slow down the process of development.

Mobile Application Programming Language

The second design challenge identified by Team Rat in a Hat is identifying an appropriate language in which to create the mobile companion app. MealWrite requires this application so that users can search for recipes to prepare in the future without having to utilize an Alexa capable device, as the voice based interface of these devices makes it difficult to effectively search for this type of content. A programming language in which to implement the application is required, which will be evaluated on a multitude of various factors. Given that the expected users will have a wide range of technical skills, this application must feature an intuitive user interface that will be familiar and easy for new users to grasp.

Recipe and User Storage and Representation Requirements

The third challenge is addressing what technology to utilize in order to store recipe and user data. MealWrite must have a central store for the recipes that a user can request to cook. Recipes must be easily stored and retrieved in a representation that the developed programs can easily load, manipulate, and transmit over network connections. These recipes are expected to take the form of a directed acyclic graph of steps, accompanied by supporting metadata. This data must be relatively simple to create by hand, as automatically creating recipe data structures from plain text data is out of scope for this project. JSON (JavaScript Object Notation) will be used for this textual representation and transmission, as it is highly human readable and it is supported in the vast majority of these systems. Additionally, at this point these recipes will only be able to be loaded into the storage system manually, as creating an administration interface for this purpose is also out of scope. The recipes must be able to be searched by several criteria, including name, cuisine, and options such as being vegetarian. This storage solution will also be used for storing user account information. The user information stored will include recipe history and family size, with the ability to add further information in the future. This user information will be simpler to work with than the recipe data and does not introduce any new constraints, so recipe storage will be the main challenge considered. The selection of services considered for this analysis will depend on the selection of the backend service host made previously.

Client Session and Backend Programming Language

The last major challenge that requires analysis is selecting a language in which to implement the client sessions and backend server. MealWrite needs to be able to run a session for each active user, keeping track of their recipe cooking status and supporting skill information. MealWrite also needs to run a back end server to manage user profiles and handle recipe searching. The sessions need to be able to process and schedule recipes while interfacing with the backend server. The backend server must be able to manage communication with many sessions at once. This analysis will also depend on the backend service host selected, as some service hosts and products make working with certain languages much more streamlined.

3. Technological Analysis

For each analysis, the desired characteristics will be listed and potential solutions will be identified, which will then be ranked by order of the solution best fulfilling those characteristics. Each section within this technological analysis will have its own unique criteria, as each topic has its own specific qualities that are unique. Each criteria in each section has been determined by the development team, so as to better highlight the positive and negative characteristics each topic has. In addition, each criteria will be ranked on a scale of how well each option fairs compared to others based on those criteria, where a first place score indicates the best possible quality in a topic, and last place scores indicate the worst traits. All criteria will have a score for each topic, and corresponding tables will contain the overall placement the topic received based on the scores of the criteria. Placements of equal rank indicate a tie between two or more options, meaning that the quality of one option matches the quality of another.

3.1: Backend Service Host

Team Rat in a has identified the key criteria for a suitable backend service host for the backend services of MealWrite. These criteria are explained next, and then summarized in table 3.1.

3.1.1: Criteria Descriptions

- **Serverless Compute Platform**

A serverless compute platform is required for MealWrite to enable scaling the service to be able to service many concurrent users without issue. If this service were hosted on traditional always-up servers it would require complicated logic to balance between these servers, and these servers would incur unnecessary cost while they are underutilized. Additionally, these servers would have the possibility of being overloaded with user sessions, leading to noticeable delays in user interaction, which would cause degradation of the user experience. As this category does not have a range of different possible options, service providers that offer this sort of platform will be ranked first, and service providers that do not will be ranked second.

- **Interoperability with Echo**

Interoperability with Echo without a large amount of overhead for connection and authentication is required to allow development of the MealWrite skill to be achieved without causing unnecessary development overhead, allowing the development team to proceed with development quickly and efficiently. This category will also be ranked with a binary scale, with a first place ranking indicating that this is able to be achieved with

minimal effort, and a second place ranking indicating that the service has significant difficulties achieving this connection.

- **Dedicated Database Platform**

Having a dedicated database platform within the service host's products is another service that will greatly simplify the development and deployment of MealWrite, as manual configuration not being required will make development much more streamlined. Ranking will again be binary, with a first place ranking indicating that the service provides a dedicated database platform, and a second place ranking indicating that it does not.

Table 3.1: Backend Service Host Criteria

Criteria	Description
Serverless Compute Platform	Existence of a serverless compute platform to utilize for the hosting of the Alexa skill.
Echo Interoperability	Operability with Echo devices has a small amount of overhead, enabling developers to progress quickly.
Dedicated Database Platform	Existence of a dedicated database platform to utilize in order to enable developers to ease deployment and development.

3.1.2: Proposed Technologies

Amazon Web Services (AWS)

AWS is an encompassing term for all of Amazon's online services and solutions, of which there are tens of services, offering native solutions for nearly any use case. As this platform is provided by Amazon, the providers of Alexa services, it is expected that this host will offer first class support for the required criteria.

Microsoft Azure

Azure is an encompassing term for Microsoft's online services and solutions, also offering tens of services that should be able to meet nearly any use case.

DigitalOcean

DigitalOcean provides a smaller number of services and solutions than AWS and Azure, potentially requiring its users to manually manage services that AWS or Azure provide as fully managed services, or even to develop their own solution. This would not be

acceptable for MealWrite unless no other solution is available, as this would make development much more difficult.

3.1.3: Analysis

These proposed technologies are now analyzed according to the identified criteria.

Amazon Web Services

- **Serverless Compute Platform - 1st**
    - AWS provides Lambda, a serverless compute platform. It is fully supported within the AWS ecosystem, with examples of how it can be integrated with many of the other services available.
    - As this platform provides a serverless compute platform, it receives a first place ranking.
- **Interoperability with Echo - 1st**
    - AWS Lambda has first class interoperability with Echo, provided through "Alexa Hosted Skills."[1] This service automatically sets up a Lambda endpoint for the skill session, which will run an individual session dedicated to each user.
    - This service has no comparison in other service hosts, offering by far the easiest interoperability, and thus receives a first place rank.
- **Dedicated Database Platform - 1st**
    - AWS offers a wide range of dedicated database services of many types, including relational, document, and graph. Some of these options provide a serverless version as well, which would further simplify development and deployment.
    - As this host provides a range of dedicated database platforms it receives a first place ranking.

Microsoft Azure

- **Serverless Compute Platform - 1st**
    - Azure provides a serverless compute platform. It is well supported within the Azure ecosystem, with examples of how it can be integrated into many different larger solutions.
    - As this platform provides a serverless compute platform, it receives a first place ranking.

[1] "Alexa-hosted Skills | Alexa Skills Kit - Amazon Developer." https://developer.amazon.com/en-US/docs/alexa/hosted-skills/build-a-skill-end-to-end-using-an-alexa-hosted-skill.html. Accessed 30 Mar. 2022.

- **Interoperability with Echo - 2nd**
    - As Azure is not an Amazon service, it cannot provide first class integration with Echo devices and Alexa services.
    - The steps required to set up a connection to an Alexa skill are complicated and involve many steps of setup, including advanced web administration skills that none of the developers are familiar with.[2] Because of this, this provider receives a second place ranking.
- **Dedicated Database Platform - 1st**
    - Azure offers another wide range of dedicated database services that are very comparable to AWS's offerings. It thus receives a 1st place ranking.

DigitalOcean

- **Serverless Compute Platform - 2nd**
    - DigitalOcean does not currently provide a serverless compute platform. They currently have a service in closed beta that is aiming to provide this capability, but it is not yet able to be utilized.[3]
    - Because this service is not yet available, DigitalOcean receives a second place ranking.
- **Interoperability with Echo - 2nd**
    - See discussion in the Azure section, as it applies to this service as well. It thus received a second place ranking.
- **Dedicated Database Platform - 1st**
    - DigitalOcean provides a number of dedicated database services. They are not as wide as the options provided by Azure and AWS, but the provided options would be sufficient. This service thus receives a first place ranking.

3.1.4: Analysis Overview

Table 3.2 aggregates the ranks of the considered hosts for the backend, based on the analysis of each of the predetermined criteria.

[2] "Host a Custom Skill as a Web Service - Alexa - Amazon Developer." 13 May. 2019, https://developer.amazon.com/en-US/docs/alexa/custom-skills/host-a-custom-skill-as-a-web-service.html. Accessed 30 Mar. 2022.
[3] "DigitalOcean + Nimbella - Serverless computing comes to ...." https://www.digitalocean.com/nimbella. Accessed 30 Mar. 2022.

Table 3.2: App Programming Language Criteria Ranking

Host	Serverless	Interoperability	Dedicated Database	**Average**
AWS	1st	1st	1st	1st
Azure	1st	2nd	1st	2nd
DigitalOcean	2nd	2nd	1st	3rd

This shows that AWS will provide the best solution for hosting MealWrite's services, as it ranks first in every category, and will be able to fulfill all of the requirements of MealWrite most optimally.

3.1.5: Proving Feasibility

The feasibility of this solution will be shown through the development of the demos for the feasibility of the Lambda skill, backend, and database, as all of these components will utilize the services of this host.

3.2: Mobile Application Programming Language

Team Rat in a Hat has identified the key criteria for a suitable programming language for the development of the mobile companion app component of the MealWrite service. These criteria are explained next, and then summarized in Table 3.3.

3.2.1: Criteria Descriptions

- **Usability**

This criteria determines how usable a certain language is for the development team. Usability is a measure of how much prior experience the team has in a certain language. Languages that the team has coded in before are more preferable. Familiar languages would allow the team to spend more time coding instead of having to research basic facets of unknown ones, such as proper coding conventions and syntax. Ranking for this category is determined by both the total hours of work the team has in experience with each language and the number of similarities less familiar languages have with other languages the team is experienced with. A first place ranking in usability indicates the best understanding of the programming language in question.

- **Connectivity**

This project will consist of multiple operations running in tandem with one another, meaning that determining how well a language can take in information from supporting

programs takes a high precedence. Acquiring data from a database of recipes, for example, is a process that is absolutely required for the project to operate at all, meaning that any language chosen needs to have some ability to acquire that data. Languages that do not offer this ability directly would result in developing workarounds, and could hinder performance as well as overall usability. Ranking for this category is determined by the amount of possible features each language has in regards to each languages' ability to send data to outside running programs. A first place ranking in connectivity indicates a language has the best possible means of sending data between the mobile companion app itself and supporting running programs beyond the mobile companion app.

- **Onboarding**

Given that this project is intended to be the foundation for future mobile app developers to build off, it is imperative that the language used can be easily adapted by programmers outside the current working group. The decision on what language to choose must be sensible enough that incoming mobile companion app developers can both refactor and build upon what would already have been coded in this mobile companion app. The clients intend on furthering the development of their ideas beyond the current scope, so having a language that makes this process as seamless as possible would be ideal. Ranking for this category is determined similarly to the usability category, where the number of hours the team has in experience with a language is comparable to the number of hours incoming programmers have from their own experience in coding within academia. A first place ranking in onboarding indicates that the language would be better understood by incoming programmers, making what code was written appear as understandable as possible.

- **Extensibility**

This category is similar to onboarding, but different in the sense that additional functionality can be added upon based on the chosen language. Should future developers of this mobile companion app decide to reimplement facets of the code in a means that they feel would be more optimal, it is necessary to choose a language that allows for that transition to be both possible and easy to achieve. This category focuses more on refining the addition of functionality rather than ease of understanding what is already present in the finished product. Ranking in this category is determined similarly to the connectivity category, where the number of features a language has is the same for both the team and incoming programmers. Features the team decides to use in addition to those not used are considered here, operating under the assumption that more features would be implemented in the future beyond what the team intends to produce. A first place score in extensibility indicates that incoming programmers would have the easiest time adding additional features to the mobile companion app.

Table 3.3: App Programming Language Criteria

Criteria	Description
Usability	Existing knowledge of the team in the proposed language.
Connectivity	What options the language has for connecting to the other required components.
Onboarding	How well incoming code viewers can understand both the language and what is coded in that language.
Extensibility	As this product is being created for a startup company that will continue work on it in the future, it is needed to ensure that chosen tools will be well supported in the future and will be easy to work on.

3.2.2: Proposed Technologies

Dart

The first suitable language for Android mobile companion app development considered was Dart. The coding team's initial assumptions on Dart as the most competent language for use was confirmed when further research was conducted into other options. Dart offers the Flutter framework, which would allow for development of all mobile companion app requirements within a single codebase. Further optimization in Dart is seen client-side, which is important for this case as intended for the mobile companion app to be as user-friendly as possible, and the team has significant data that needs to be sent over from each client to some supporting main database. Additionally, development of user interfaces would be readily testable, as Dart allows for hot-reload changes to the UI instantaneously, so any changes made are immediately visible during the UI coding process.

Kotlin

The second candidate language was Kotlin due to its dominance in coding Android apps since 2019, and is recommended for use by Google. In terms of personal usability, since Kotlin is based on Java in coding standards, the team would be well placed to learn this language due to prior experience with Java. Hence, Kotlin can be viewed as a suitable candidate in that respect, as languages similar to those studied in depth would allow for better usage of the full functionality faster. More specifically, efficiency and testability are easier to accomplish with other options. While Kotlin has means of

achieving the same level of testability, it would take more time and effort when compared to other options, and if ease of codability along with testability can be better achieved elsewhere, would rather be chosen the more convenient capabilities other languages provide.

Java

The third candidate viewed for Android mobile companion app programming was Java. Java appears to be a suitable candidate for mobile companion app development, as it once was the language used to code most Android mobile companion apps since 2019. In terms of personal usability, all of the team members have some degree of experience with Java code development, mostly from coding in Java in previously taken classes. While technically Java is still supported by Android, and mobile companion apps are still able to function properly using just Java code, Kotlin has taken over as the primary coding language past 2019, and is the language that most modern Android mobile companion apps are coded in today. Because of this even if Java was the main choice for mobile companion app development, Kotlin would be superior over Java as both Kotlin is based on Java, and Kotlin contains better connectivity and functionality in the realm of Android mobile companion app development.

Python

The fourth candidate in the programming language analysis was Python. Python as a language on its own is viewed as one of the most easily applicable languages, as the coding conventions for Python in general are simplistic in design, allowing for ease of codability. Additional facets of other coding languages involve unnecessary fine details that Python complies on its own. In addition, prior coding experience in Python from subsequent coding classes gives the team an edge, despite how easy Python as a language is to pick up as a novice programmer. The only issue with Python in this case, however, is that it needs to use some language that actually works within an Android environment, and Python does not fall into that category as well as other possible languages. It would be necessary to have a more difficult time connecting any of the code to the Android environment, so in terms of mobile companion app development, Python is not ideal.

3.2.3: Analysis

These proposed technologies are now analyzed according to identified criteria.

Dart

- **Usability - 2nd**
    - None of the team has any experience coding in Dart beyond research involved in the development of this feasibility report.
    - Dart has stark similarities in coding structure and syntax when compared to Java. Since the team has extensive knowledge in Java from advanced computer science courses in academia, this knowledge can be adapted for use in Dart. This would require additional time and research to fully implement that adaptation, but that feat should be entirely possible and more or less seamless.
    - Despite the lack of prior experience the team has coding in this language, Dart receives a 2nd place ranking for usability due to the language's similarities to Java, a language the team has extensive prior experience with.
- **Connectivity - 1st**
    - Dart is designed to enable easy connection to services outside of the Android ecosystem. Both the establishment of a proper connection and the sending of necessary data between the mobile companion app and outside programs should be an intuitive process.
    - Due to the more open nature of Dart for other outside programs, there would be less issues with a larger variety of outside programs as a whole. As a result, the team is less limited in the options in deciding what outside framework the team wishes to choose from. This would allow for the team to pick the best option to connect to the mobile companion app coded in Dart, rather than picking whatever works purely for the sake of actually establishing a connection from the mobile companion app to outside programs in the first place.
    - Dart receives a 1st place ranking in connectivity, due to the vast amount of features Dart has in data sending to outside running programs.
- **Onboarding - 1st**
    - While Dart is not necessarily the main language used in Android app development, it is still a popular choice for mobile development in general. Additionally, Dart as a language is fully supported by Google themselves, meaning that it is viewed as a prime candidate for mobile app development. This means that it is more likely that incoming programmers would have some form of experience with the language and a less difficult time understanding how the finished product works, and would have an easier time refactoring code to match their own desires.

    - The scope of Dart reaches beyond Android app development, as Dart is cross platform by design. This means that incoming programmers new to Android app development specifically would have a better time understanding what the finished product does despite their lack of prior experiences. Developers with mobile app coding experience outside of Android can more easily adapt their prior knowledge, meaning less time will be spent understanding how the code works and more time can be spent refining such code in ways they see fit.
    - Dart receives a 1st place ranking in onboarding due to its popularity based on plentiful qualities in the scope of usability beyond the current coding team's experience. In addition, the number of hours incoming programmers have in this language is most likely on the higher end.
- **Extensibility - 1st**
    - For similar reasons as the extensive onboarding capabilities of Dart, the possibilities for additional features to be implemented are much more feasible. Essentially, since Dart is a popular language for mobile app development, it is more likely incoming programmers will know of it, thus minimizing time learning the language and maximizing time developing additional features after the completion of this foundational project.
    - For similar reasons as described in the connectivity criteria, Dart would be able to offer the same features for use by both the team and for incoming programmers after the completion of this foundational project. The same measures the team will use to connect outside running programs with the mobile companion app can be used by incoming programmers, giving them an avenue to design additional features within the same framework as the foundational project with relative ease.
    - Any additional features of Dart that the team chose not to implement can still be implemented in the future by incoming programmers. More specifically, incoming programmers can add additional functionality for connectivity purposes for additional outside running programs. Those new features can easily coincide within the framework built off Dart as a language.
    - Dart receives a 1st place ranking in extensibility due to Dart's various qualities in terms of connectivity, in addition to more connectivity features beyond the scope of the current team's intended foundation for the mobile companion app.

Kotlin

- **Usability - 3rd**
    - None of the team has been introduced to Kotlin as of now. Despite that, Kotlin is built off of Java, which is a language the team has significant experience in at an advanced academic level from college courses. Adapting from Java to Kotlin should therefore be more or less a seamless process.
    - Kotlin receives a 3rd place ranking in usability, despite its connection to Java, a language the team has numerous hours of prior experience in. The low ranking is justified based on the somewhat limited features Kotlin has for mobile development, making the usage of Java experience in Kotlin hold less significance in terms of ranking.
- **Connectivity - 2nd (Tied)**
    - Kotlin has been the main language used in Android app development since 2019, officially recommended by Google. Because of this, all of the most recent updates to Android phones are designed for usage of Kotlin apps specifically. This means that any running programs outside of the mobile companion app itself should have a seamless means of accessing requests from the mobile companion app if the team chooses to code in Kotlin.
    - Limitations could occur from using Kotlin, as it is by design a subset of the Java language. Some running programs outside of the mobile companion app would require connecting to Kotin-based requests. Considering that, and considering that Kotlin is relatively new to the Android environment, some messages sent from the mobile companion app to outside programs might not have established a means to correctly achieve that connection at all. Should this be the case, workarounds for achieving mobile companion app to outside program connections might need to be developed, which would categorize Kotlin connectivity specifically as more of a hindrance than as a benefit.
    - Kotlin receives a 2nd place ranking in connectivity, due to the limited amount of features Kotlin has to more easily develop data distribution between the mobile companion app and outside running programs. Kotlin does have expansive connectivity features, but such features are lacking in understanding by the current coding team, dropping the ranking considerably.

- **Onboarding - 2nd**
    - Since Kotlin has been the main language used for Android mobile app development since 2019, it is fair to assume that most incoming Android app developers would have a fair understanding as to how Kotlin as a language works. This is further proven by the nature of Kotlin as a subset of the Java language, which is a widely understood programming language and used to be the primary language for Android app development prior to 2019.
    - It is entirely possible that incoming developers might still be using Java as their main coding language for Android mobile companion app development. As a result, they would need to undergo the same level of research done when developing the app. Seasoned programmers might not find this task too difficult, but the main goal is to ensure as many incoming programmers can accurately understand what the finished product actually does as possible. There is no guarantee that all possible incoming programmers both understand and are willing to code in Kotlin.
    - Kotlin receives a 2nd place ranking for onboarding, due to the probable popularity and consequently better understanding of the language for mobile app software developers beyond the current coding team. Incoming programmers would have more hours of experience in Kotlin due to that conclusion. The lesser number of hours the current team has in Kotlin drops the ranking slightly, as more seasoned Kotlin developers continuing the project might prefer alternative coding procedures instead of what the current team developed prior.
- **Extensibility - 2nd**
    - Since Kotlin has been the main language used for Android mobile app development since 2019, fully updated versions of Android should be capable of any additional functionality incoming programmers are willing to develop on top of the finished product. Purely in the scope of the capabilities Kotlin has in the realm of the Android app environment, Kotlin should be capable of being applied to most incoming additions to the final product.
    - For similar reasons described in the onboarding section, incoming programmers might either prefer using Java, or have less experience using Kotlin as Kotlin is relatively new to the Android environment. Full integration of Kotlin development is described by Google themselves, but the experience of incoming programmers is not a guarantee. This would result in more research into how Kotlin as a language is structured instead of applying additional features to the foundational final product.

    - For similar reasons as described in the connectivity section, Android apps coded in Kotlin might not have full integration with outside running programs. This could result in limitations in the options both the team and incoming programmers would have when deciding to implement new functionality. Should some caveat in Kotlin connectivity occur for a crucial component of new features incoming programmers wish to integrate, they would need to implement their own workarounds. This could conflict with previously developed workarounds, if any, and could make the process of adding additional functionality more difficult after the foundational product is finished.
    - Kotlin receives a 2nd place ranking in extensibility, due to the probable popularity and subsequent understanding of the connectivity of incoming programmers beyond the current coding team. The plentiful connectivity features Kotlin has can be better applied by more seasoned Kotlin programmers, but the features the current team integrated prior might warrant refactoring in the perspective of such incoming programmers.

Java

- **Usability - 1st**
    - The team has prior experience coding in Java, especially in higher level college computer science courses. This means that the team can spend less time learning how to develop more complex coding procedures and more time actually coding said procedures.
    - Since Java was once the main language for Android app development, the team can apply prior experiences with Java development with relative ease. The team would still need to understand how to code a mobile app specifically, but the baseline concepts and higher level understanding the team already has would help immensely with research in Java mobile app development. This would lead to less time being spent figuring out how to code in Java and more time being spent learning specifically how to create mobile apps.
    - Java receives a 1st place ranking in usability, as the current team has a firm grasp on how to code in this language overall due to the current team's numerous hours in Java coding. Mobile app development is less known by the team, but is the most familiar out of the available options.
- **Connectivity - 2nd (Tied)**
    - Java was once the main language for Android mobile apps, so Java compatibility for connecting the mobile app to other programs would be more or less achievable.

    - Since Java is no longer the main language for Android mobile app development, connecting to supporting programs outside of the mobile app would most likely require usage of legacy features still present in the most updated version of Android. Such legacy features might not have the same level of ease in terms of connectivity, as newer features that use Kotlin instead might be more preferable to use in terms of speed and data transfer. If this is the case, and the legacy features are far less optimal, the team might need to develop a workaround that still would not have the same level of optimization as if the team coded in Kotlin instead.
    - Java receives a 2nd place ranking in connectivity, as Java is more dated in the Android mobile application app development process. As a result, the number of connectivity features are limited, reducing the ranking placement considerably.
- **Onboarding - 3rd**
    - Despite Java being dated in its development in Android mobile apps, it is at least familiar to current mobile app developers. Since Kotlin was based on Java, incoming mobile app developers can apply their coding experience with Kotlin with the finished product. Understanding is achievable, but might not be guaranteed with all current professional Android developers. This means that it is possible that they would need to spend more time researching Java mobile app development.
    - Some incoming programmers might find the usage of Java for the mobile app confusing, as they could have a better preference for Kotlin. The reasoning behind the transition from Java to Kotlin for Android mobile app development has merit due to the increase in optimization and connectivity. While Java might not be an unfavorable choice in the perspective of incoming mobile app developers, it would definitely be less favorable than Kotlin.
    - Incoming programmers might not have experience with Java mobile app development, as they would be more used to using Kotlin due to the previously described transition. Optimization for mobile app development provides benefits that would result in incoming programmers having either abandoned Java as a mobile app language overall, or having not learned to code apps in Java at all. It is fair to assume that Kotlin is the norm for mobile app development, resulting in more research in Java mobile app development, further resulting in more time necessary to understand what would have already developed.
    - Java receives a 3rd place ranking in onboarding, due to the dated nature of the language in the Android mobile app environment. It is likely that

incoming programmers would have more hours of experience in Kotlin, a language that is most similar to Java. Despite that, full integration of Kotlin experience in Java mobile app development specifically is more difficult, further reducing incoming programmers' understanding of the finished foundational product written in Java, dropping the rank placement considerably.

- **Extensibility - 4th**
    - Extensibility is hindered for similar reasons as onboarding is hindered, in the sense that learning how to code mobile apps in Java would require time better spent implementing additional mobile app features had Kotlin been chosen instead.
    - Implementation of additional mobile app features by future incoming programmers would result in usage of legacy connectivity code in the Android environment. Such legacy implementation in general would most likely require more time spent understanding how to implement new features in this way.
    - Some additional features available in choosing Kotlin for adding additional features to the mobile app might be either less possible and or more difficult if Java was the chosen language. Some updates to the Android environment could result in loss of functionality using Java, as the updates are more geared towards Kotlin based applications.
    - Java receives a 4th place ranking in extensibility, since incoming programmers might have a more difficult time distributing data from the mobile app to their own outside running programs they might wish to incorporate. Incoming programmers have more numerous hours in Kotlin experience, relevant in Java's case as Kotlin shares similarities to Java. As such, better understood avenues of achieving data distribution between the mobile app and outside running programs through Kotlin might be less possible in Java, reducing both relevant Kotlin connectivity experience in Java and the placement of Java's extensibility.

Python

- **Usability - 2nd**
    - Python is one of the easiest languages for coding in general.
    - Prior experience coding in Python is enough to give a fair baseline for more advanced coding procedures.
    - Syntactical facets of Python are as simplistic as possible, omitting trivial coding conventions present in other languages.

    - Prior Python experiences do not exceed beyond more than introductory college-level courses, excluding choosing Python for more advanced courses out of preference. For the team in general, the team has at least the guarantee that the team has once coded in Python during introductory computer science courses.
    - Python receives a 2nd place ranking in usability due to both the current team's plentiful hours of prior coding experience and the nature of Python as one of the easier languages to understand in general. Python mobile app development is less known by the current team however, dropping the ranking score by a degree as a result.
- **Connectivity - 4th**
    - Python by itself is incapable of being used in an Android environment without tedious workarounds
    - Kivy is a language based on Python which does work in an Android environment, but currently is not fully implemented, and would still require further workarounds to achieve full functionality
    - Outside of mobile app development, Python is reasonably capable of data transfer from other working programs. This is outside the realm of mobile app development specifically, however.
    - Python receives a 5th place ranking in connectivity, due to Python not having a reasonable means to accomplish data distribution to outside running programs. Python is not fully implemented in the Android mobile app environment, resulting in a limited number of features available to achieve proper data distribution between the mobile app and outside running programs.
- **Onboarding - 4th**
    - Python is a widely understood language, as Python's more simplistic nature is viewed as more attractive in the realm of computer science outside of the group.
    - Python is easy to pick up as a coding language, further proved by the fact that it is taught at an introductory level in computer science college classrooms.
    - In terms of mobile app development, the workarounds needed to be developed for Python to be able to run on an Android environment at all might prove confusing for incoming programmers. The connectivity of Python for Android mobile app development hinders the onboarding capabilities heavily.
    - Choosing Python would be a confusing choice in the opinions of most current Android mobile app developers. As a result, most mobile app

developers would be unfamiliar with the process of coding an Android mobile app in Python in general. Certain procedures present in other apps in other languages could somewhat help this unfamiliarity, but the process of mobile app development in other languages would most likely require completely different procedures that are not present in Python mobile app development.
    - Python receives a 4th place ranking in onboarding. The number of hours in prior experience in Python overall might be plentiful for both the current team and incoming programmers, but the number of hours becomes less significant in the realm of Python Android mobile app development specifically.
- **Extensibility - 4th**
    - Since Python is not fully integrated as a language in an Android environment, further development of additional functionality by incoming programmers would require them to implement their own workarounds. This would most likely be considered more of a nuisance, resulting in incoming programmers abandoning any further development on the mobile app overall.
    - Any additional functionality coded in Python could result in incoming developers migrating languages on their own, as most Android mobile app developers prefer using other languages for connectivity purposes. This would be a long and difficult procedure, but such would be seen as preferable over trying to implement workarounds in the opinions of most Android mobile app developers. This process can be easily mitigated by simply choosing a better language that the majority of Android mobile app developers already prefer. In addition, this process would require more work time aside from the main goal of extensibility criteria, meaning less time and effort is placed in actually improving the foundational app.
    - Workarounds for developing Android mobile apps in Python could vary in development, meaning that incoming programmers might develop their own workarounds that differ from what the team would develop. This could result in conflicting changes that could damage the ability for the mobile app to run at all. This means that the incoming developer would either have to be forced to use a pre-established workaround, or to completely refactor what the team already developed that might not work. This would also mean less time adding to the mobile app and more time refactoring, meaning less of the mobile app would be improved upon in more time in general.

- Python receives a 4th place ranking in extensibility, due to Python not being fully implemented as a language in the Android mobile app environment. This means a limited amount of connectivity features possible in Python Android mobile app development, reducing rank placement for similar reasons described in the connectivity criteria. Limited features in connectivity means limited features in extensibility, as both the current team and incoming programmers would not have the means of achieving full data distribution between the mobile app and outside running programs.

3.2.4: Analysis Overview

Table 3.4 aggregates the scores of the analyzed solutions of each mobile app programming language option, based on the analysis of each of the predetermined criteria.

Table 3.4: App Programming Language Criteria Ranking

Language	Usability	Connectivity	Onboarding	Extensibility	**Average**
Dart	2nd	1st	1st	1st	1st
Kotlin	3rd	2nd (Tied)	2nd	2nd (Tied)	3rd
Java	1st	2nd (Tied)	3rd	2nd (Tied)	2nd
Python	2nd	4th	4th	4th	4th

It was concluded that Dart should be the best mobile app language option, as the previously described features give Dart a significant edge over the other language options. While the other language options might not be unfavorable by themselves, Dart contains better functionality the team found more favorable for use in the mobile app development process.

3.2.5: Proving Feasibility

Feasibility of the selection of Dart as the language of choice for developing MealWrite's mobile companion app will be proved by creating a mobile app with it that reads recipes from the demo backend database and server.

3.3: Recipe and User Storage and Representation Requirements

As previously described, Project MealWrite requires a solution that will allow it to store, retrieve, and load, recipes and user data. As decided in section 3.1, AWS will be utilized as the backend service provider, so all of the considered solutions are provided by AWS. There are many products and services available that solve variations of this challenge for different use cases, so this analysis will require identifying both the general class of technology to utilize, and which specific technology to utilize. The criteria that these technologies will be analyzed on is presented next, and are summarized in Table 3.5.

3.3.1: Criteria Descriptions

- **Ease of CRUD (Create, Read, Update, Delete) Operations**

This criterion is based on evaluating the ease of conducting standard basic database operations. This is evaluated together with the next criteria, and described further in the next point. Ranking is determined by the number of lines of code required to conduct these operations and the computational overhead required to complete these operations.

- **AWS Integration**

This criterion is based on the technology's ability to integrate with other technology in the AWS ecosystem, providing ease of integration of multiple technologies, and enabling data security and integrity. To evaluate this and the previous criteria, a table or store was created in each option, and then both a Lambda function and standalone script were created in Python to load existing data to the database, add a single record, read the data back, query for data matching given criteria, search for a specific element, and then update that element before deleting it. This sequence verifies that the technology is able to perform the necessary requirements, and demonstrates how well it integrates with other pieces of technology within the AWS ecosystem. Ranking is determined by the number of lines of code and steps of setup required to be able to establish a connection so that CRUD operations can be conducted.

- **Scalability**

This criterion is based on the technology's ability to meet the needs of a growing base of users without the quality of the service being diminished. This is much harder to directly test than the previous criteria, so to evaluate this the options that each technology provides for scaling are evaluated based on the apparent ease of scaling the technology up to meet these increased demands. Ranking is determined by the feasibility of scaling the database's capacity to meet demand automatically, and if this

is not an option then they are ranked by how much manual intervention is required in order to increase the available capacity.

- **Price**

This criterion is based on the technology's price, evaluated both in terms of price during development, and the expected price during deployment of MealWrite to a large user base. For the price comparison the available price calculators for each service are used to estimate how much each service would cost for use during development, and how much it would cost with a large user base, with a work load intended to emulate 100,000 daily users comprising 20,000 concurrent users at peak sustained for 4 hours, and the other 20,000 daily users being spread roughly evenly throughout the day. This amounts to roughly 5 users per second at peak, and one user every several seconds during off-peak hours. Ranking is determined by the price of the solution during deployment and the price during development, with the price of deployment being given higher weight.

Table 3.5: Storage Technology Ranking Criteria

Criteria	Description
Ease of CRUD Operations	Operations on the data must be easy for the programmers to implement. This especially includes ease of retrieval of records and the ease of querying records based on different criteria.
AWS Integration	Staying within the AWS ecosystem will simplify development and enable additional services to be integrated more easily if they become required.
Scalability	The service must be able to easily scale to accommodate growth when the product is launched, potentially up to tens of thousands of concurrent users.
Price	During development the price of the services must be minimal, and the price when deployed must remain manageable.

3.3.2: Proposed Technologies

The major categories of storage technologies that were investigated are document stores, file storage, and relational database storage. These categories and examples of each are presented next.

Document stores

Document store databases are based on the idea of a document consisting of any number of keys, each containing data, which may be a base data type, list of data, or another nested document. These databases are usually slow to read from with advanced queries, but the flexibility of their storage can make reading complicated documents from them faster than reading equivalent rows from a relational database, which would take multiple joins and complicated logic to reconstruct it into a usable form after reading.

Amazon DynamoDB

Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. DynamoDB offers built-in security, continuous backups, automated multi-region replication, in-memory caching, and data export tools.[4]

DynamoDB provides serverless access to data, is integrated with Alexa hosted skills, and is free for up to 25 gigabytes of storage and 200 million read/write requests per month.

Amazon DocumentDB

Amazon DocumentDB (with MongoDB compatibility) is a fast, scalable, highly available, and fully managed document database service that supports MongoDB workloads. As a document database, Amazon DocumentDB makes it easy to store, query, and index JSON data. Developers can use the same MongoDB application code, drivers, and tools as they do today to run, manage, and scale workloads on Amazon DocumentDB. Enjoy improved performance, scalability, and availability without worrying about managing the underlying infrastructure.[5]

DocumentDB has a one month free trial, but after that point requires allocating instances, I/O operations, storage space, and S3 backup space, each of which have an additional cost.

MongoDB

MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which

[4] "Fast NoSQL Key-Value Database – Amazon DynamoDB." https://aws.amazon.com/dynamodb/. Accessed 4 Mar. 2022.
[5] "Amazon DocumentDB (with MongoDB compatibility) FAQs." https://aws.amazon.com/documentdb/faqs/. Accessed 4 Mar. 2022.

are the basic unit of data in MongoDB. Collections contain sets of documents and functions which is the equivalent of relational database tables. MongoDB is a database which came into light around the mid-2000s.[6]

MongoDB will be evaluated by hosting a cluster on an EC2 compute unit, as the free cloud hosted option cannot be easily accessed with a Lambda session.

MongoDB has a free cloud hosted option limited to 512 megabytes, and can be self hosted for free using their community edition. However, this requires dedicated hosting resources, which would incur a cost.

Other Storage Types

Other options that could be utilized are file based storage and relational database storage, however neither of these is well suited for recipe storage, so they will not be thoroughly analyzed as options in the next sections, but are included for completeness. File based storage, such as Amazon S3, is not suitable because any read or write operation has to access the entire file for every operation, which is slow and would quickly become expensive because of S3's pricing model. The shortcomings of relational databases were mentioned briefly in the previous section, but to reiterate their issues are that they would take many read and write operations for each recipe, and would be complicated to reconstruct after reading. Relational databases are well suited for user data storage, however fragmentation of storage technologies would lead to an architecture that is more complicated, and would be likely to incur additional costs, while providing minimal benefits.

Amazon S3

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, the team can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.[7]

---

[6] "What is MongoDB? Introduction, Architecture, Features & Example." 1 Jan. 2022, https://www.guru99.com/what-is-mongodb.html. Accessed 4 Mar. 2022.
[7] "Amazon S3 - Cloud Object Storage." https://aws.amazon.com/s3/. Accessed 4 Mar. 2022.

Amazon RDS

Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while automating time-consuming administration tasks, such as hardware provisioning, database setup, patching, and backups. It frees the team to focus on the applications so they are given the fast performance, high availability, security, and compatibility they need.[8]

3.3.3: Analysis

The proposed technologies are now analyzed according to the specified requirements.

DynamoDB

- **Ease of CRUD operations – 1st**
    - Data tables in this system require a primary key to uniquely identify their items, and supports creating secondary indices to make querying operations much more efficient.
    - As required, writing arbitrary structured data to the database is trivial, requiring only a single call on the initialized table object that passes a dictionary to store.
    - Retrieval is similarly easy, with a single call on the same table object, passing a dictionary of the primary key.
    - Searching is less trivial, as there are two separate approaches to it. The first is querying, which efficiently searches the table based on any primary or secondary key. The second approach is scan based searches, which read complete rows of the table and then filter them based on the scan parameters. These searches are slower and take more resources to perform. These two types of queries can be combined, initially selecting rows with a query and then filtering them with a scan, to provide almost the same flexibility as scanning based approaches while diminishing their overhead. These operations also require a few different types of syntax depending on the combination of operations being performed, further adding complexity.
    - Updating items is a return to simple operations. It is again a single call, with a selection key, expression used to update the value, and a key that specifies what to return from the update operation.
    - Delete operations are performed identically to read operations.

[8] "Amazon RDS | Cloud Relational Database | Amazon Web Services." https://aws.amazon.com/rds/. Accessed 4 Mar. 2022.

    - As this was the only option in which these operations were able to be performed, this solution receives a first place ranking.
- **AWS Integration – 1st**
    - Accessing the DynamoDB table in the standalone script was trivial once the simple process of AWS CLI account setup was completed, only taking two lines of code to initialize the connection and select the table of interest.
    - Accessing the table was slightly more complicated in the Lambda function, requiring the setup of AWS permissions, specifically assigning the Lambda function an execution role with access to DynamoDB. After this was enabled, the process of using the code was exactly the same as the standalone script.
    - This option has by far the strongest AWS integration, as every operation was accomplished with a minimal number of steps and lines of codes, gaining a first place ranking.
- **Scalability – 1st**
    - Because of its fully managed and serverless nature, DynamoDB purports to be able to scale to effectively any size workload without major slowdown, and to maintain very low read and write latency as long as scans are not performed on large tables. Additionally, their auto scaling options are designed to allow automatically scaling the provisioned read/write units of each table, making operations cheaper by dynamically raising and lowering the provisions during peak and low traffic hours.
    - This option provides the easiest options for scaling, providing the ability to scale to meet demand automatically with minimal setup, gaining a first place ranking.
- **Price – 1st**
    - During development there will be no cost for this resource, as DynamoDB has a generous indefinite free tier, with 25 gigabytes of free storage and 25 each of provisioned read capacity units and write capacity units. Read and write capacity units are roughly equivalent to this number of operations being allowed per second.
    - It is assumed that during deployment a user's session may require up to 50 reads searching for different recipes to use over its duration, and involve two writes per session. This would be roughly equivalent to 250 provisioned read capacity units during peak hours. According to the AWS calculator[9], using on demand pricing this would cost approximately

[9] "AWS Pricing Calculator." https://calculator.aws/. Accessed 5 Mar. 2022.

$100/month, and similar results were reached with the provisioned capacity.
    - This is the smallest price of the evaluated options by a considerable margin, and should be negligible for the clients. With these minimal costs in both development and deployment, this option gains a strong first place ranking.

DocumentDB

- **Ease of CRUD Operations – N/A**
    - As CRUD operations could not be successfully conducted for this option it is not possible to accurately rank it, so it is not given a rank. As will be seen in the summary, this does not affect the overall decision.
- **AWS Integration – Tied for 2nd**
    - There are many powerful options available to control access to DocumentDB instances, however they get in the way of being able to effectively utilize the technology. After attempting to connect to the database through both the Lambda function and a standalone script, neither managed to connect to the database after a large number of attempts to change security settings to allow access. Because of this, the CRUD operations are not able to be evaluated.
    - This option does not perform well in this required use case, with approximately the same magnitude of issues as MongoDB, so these two options are tied for second.
- **Scalability – 2nd**
    - Scaling DocumentDB involves provisioning additional document replica instances to accommodate more read requests and connecting to those replicas to distribute the read load between them. Scaling to enable more write requests requires allocating a more powerful instance to the primary instance, which can not be done automatically.[10]
    - This option does not provide options for automatic scaling, however it does offer options for manual scaling, so it comes in second.
- **Price – Tied for 2nd**
    - During development this resource will cost a small amount, as there is no indefinite free tier available for it. Thus, during development a small instance would have to be paid for, which would cost several dollars a month.

[10] "Scaling Amazon DocumentDB Clusters - AWS Documentation." https://docs.aws.amazon.com/documentdb/latest/developerguide/db-cluster-manage-performance.html . Accessed 6 Mar. 2022.

    - It is hard to calculate the approximate cost of this option, as the calculator relies on allocating a number of instances, but does not estimate how many instances will be required for a given workload. However, if the team assumes that a single instance can service approximately 2,000 users this would require 10 instances at peak hours. This gives an estimate of approximately $500/month, which is a more considerable price.
    - This option is the second most expensive in both considered aspects, tied with MongoDB. Both of these options receive a second place ranking because of this.

MongoDB

- **Ease of CRUD Operations – N/A**
    - Like DocumentDB, CRUD operations could not be successfully conducted in this option, so it is not possible to accurately rank it, and is not given a rank. As will be seen in the summary, this does not affect the overall decision.
- **AWS Integration – Tied for 2nd**
    - Similar issues to those with DocumentDB were encountered here, as it requires connecting a Lambda session to an EC2 instance, and many issues were encountered with attempting to connect these services together. Thus this option was unable to be evaluated.
    - As discussed in DocumentDB's section, large issues were encountered with this option, tying them for second.
- **Scalability – 3rd**
    - MongoDB can be scaled by adding additional shards in a similar way to DocumentDB, however since this service is not native to AWS there are no automatic mechanisms available to scale, and would require manual creation of new instances and configuring them to work together.
    - Due to this requirement for manual intervention this option receives a third place ranking, as it is the option with the worst available options for scaling.
- **Price – Tied for 2nd**
    - Since this database option is based on the same underlying components as DocumentDB but explicitly managed, the price comparison is effectively the same.
    - As mentioned in DocumentDB's analysis, these options tie and both receive a second place ranking.

3.3.4: Analysis Overview

DynamoDB is selected as the optimal solution for MealWrite's recipe and user account storage, as it outperforms the other options on every category. The rankings in each criteria and overall ranking are presented next in table 3.6.

Table 3.6: Storage Technology Criteria Ranking

Language	CRUD	AWS Integration	Scalability	Price	**Average**
DynamoDB	1st	1st	1st	1st	1st
DocumentDB	N/A	2nd	2nd	2nd	2nd
MongoDB	N/A	2nd	3rd	2nd	3rd

3.3.5: Proving feasibility

Basic feasibility was already proved in the previous section with the creation of a script to perform all of the required basic operations, so the next step to prove overall feasibility will be triggering this from an EC2 server on the prompt of an Alexa skill.

3.4: Client Session and Backend Management

Team Rat in a Hat has identified the key criteria for a suitable programming language for the development of the client sessions and backend of the MealWrite service. As AWS was selected as the service host in section 3.1, the languages selected are those that specifically work well with AWS Lambda. These criteria are explained next, and then summarized in Table 3.7.

3.4.1: Criteria Descriptions

- **AWS Integration**

The current plan for MealWrite heavily relies on web services that Amazon provides. In order to make the best use of these services, a strong, well-integrated programming language is required. A first place ranking in AWS Integration indicates the given language has the best integration with all Amazon web services. Therefore, ranking is determined by the number of AWS services that are supported by the language.

- **Ease of use with Alexa**

The Alexa skill is the most important part of MealWrite as it is the major selling point. The programming language that will be selected must have existing documentation on Alexa APIs (Application Programming Interface). The API itself must be well-rounded

and versatile. A first place ranking in Ease of Use With Alexa indicates the language has the best API or SDK, with substantial documentation. Therefore, ranking is determined based on the number of features provided by the respective SDK, according to the documentation provided by Amazon.

- **Scalability**

Team Rat in a Hat is planning for a large and continuously expanding user base, and the plan is to allow for continuous scalability. For client sessions and backend, this means the language must be able to handle and process a growing number of Lambda functions. There is a distinction to be made between two different types of scaling: horizontal and vertical. Horizontal scaling is when additional nodes are added to an infrastructure to meet new demands. Vertical scaling is when additional resources are added to an infrastructure to meet new demands. For the sake of MealWrite and this document, horizontal scaling will be the main focus when discussing scalability. This will be the case because Lambda sessions are going to require horizontal scalability, while the backend can scale up and/or out. If the Lambda functions are supported by the given language, then it will satisfy the scalability requirement. Therefore, ranking will be in boolean fashion: 1st if the language has Lambda functionality, 2nd if it doesn't.

Table 3.7: Client Session and Backend Programming Language Criteria

Criteria	Description
AWS Ecosystem Integration	How well the option connects with Amazon's existing ecosystem.
Ease of use with Alexa	The option must be very compatible with Alexa, as Alexa skill development is a main factor in this project..
Backend Scalability	How well the option can handle multiple sessions at a time given one ongoing backend session to handle them all.

3.4.2: Proposed Technologies

Node.js

If Node.js is chosen, a decision between JavaScript and TypeScript will have to be made. JavaScript is a scripting language that has been very popular for backend since the emergence of Node.js. Node.js has an existing SDK for Alexa skills so it is what would be relied on if the decision is JavaScript. It provides a non-blocking I/O and

event-driven model. It is scalable in nature and dispenses an efficient concurrency model.[11]

TypeScript is recognized as an Object-oriented language that many are using for their backends. It compiles to JavaScript so Node.js is what would be used if this language is chosen. This gives the team benefits that are essentially identical to JavaScript. Key differences to note: TypeScript provides Static typing, and supports Interfaces.[12]

Python

Python is an object-oriented language with dynamic typing and binding. It is popular in backend development because the code is highly readable, and there is a huge selection of libraries and frameworks. Amazon provides an existing SDK which simplifies the development for the Alexa skill.[13]

Java

Java is a programming language that is class-based and object-oriented. It has a long history of being used in backend development because of its cross platform, multi-threading capabilities. Amazon has a pre-existing SDK for Java as well.[14]

3.4.3: Analysis

Node.js

- **AWS Integration – 1st (tied)**
    - Amazon provides a SDK for its Web Services. There is extensive documentation for developing server-side apps, web apps, and mobile apps. With Node.js, integration is simple. Installation of the Amazon S3 package and its dependencies are the only initialization requirement. Although, it appears that EC2 connection is also a mandatory requirement for certain functions. Once this is done, loading and using individual AWS services requires just one line of code. In addition, its SDK is written in

[11] "About Node.js." https://nodejs.org/en/about/. Accessed 7 Mar. 2022.
[12] "Difference between TypeScript and JavaScript." https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/. Accessed 7 Mar. 2022
[13] "What is Python? Executive Summary." https://www.python.org/doc/essays/blurb/. Accessed 7 Mar. 2022.
[14] "Java - Overview." https://www.tutorialspoint.com/java/java_overview.htm. Accessed 7 Mar. 2022.

TypeScript. This means things would be most consistent if TypeScript is kept as the standard while developing.[15]
    - All services are supported by Node.js, giving it a 1st place ranking.
- **Ease of use with Alexa – 2nd**
    - Amazon offers an ASK (Alexa Skills Kit) SDK for Node.js. Installing it as a NPM (Node Package Manager) module is the only step of initialization. The documentation exemplifies the simple step-by-step process of creating Alexa skills in Node.js: import dependencies, add request handlers, then create the skill package. Thanks to the SDK, a wide range of open source request handlers exist which could be integrated into MealWrite.
    - Node.js has almost all features in its SDK except for the ability to implement exception handling in different ways, and Python has this ability. This gives Node.js a 2nd place ranking.
- **Scalability – 1st (tied)**
    - Node.js would give the team great scalability, as that is one of its greatest features. Horizontal scaling is natively supported by tools built into Node.js, greatly simplifying this process. The main mechanism of achieving this is through duplication. Duplication simply means the team can duplicate their application instance, which in turn will allow it to handle a larger number of connections. Vertical scalability is also possible in Node.js, but it is not as simple. The best option for doing this is "The Cluster Module", but it introduces complexities in the management of child processes.[16]
    - Node.js is supported by Lambda functions so it gets a 1st place ranking.

Python

- **AWS Integration – 1st (tied)**
    - AWS has a SDK for Python that is composed of two different packages: Botocore and Boto3. To use Boto3, authentication credentials for AWS are needed, which is a simple process. Once the initialization is done, importing AWS services is as simple as one line of code. There is also an

[15] "AWS SDK for JavaScript." https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-started-nodejs.html. Accessed 7 Mar. 2022.
[16] "How to Scale… Scalability." https://medium.com/@OPTASY.com/how-to-scale-your-node-js-app-best-strategies-and-built-in-tools-for-scalability-a1725df082f5. Accessed 7 Mar. 2022.

AWS Code Catalog that displays how to interact and connect with every single AWS service.[17]
    - Similar to Node.js, all services are supported by Python, giving it a 1st place ranking too.
- **Ease of use with Alexa – 1st**
    - There exists an ASK SDK for Python that serves to handle most boiler-plate code for the user. Downloading and setting up the SDK is simple and can be done from the Python Package Index using pip. When coding, there are two main options. The first option is to implement code by using handler classes. There are a ton of handler classes already out there via the SDK. The second option is to implement code by using function decorators. These two options provide the same effectiveness; this means there are options and flexibility while developing a program with Python.
    - All features are supported by Python, including the ability to implement exception handling in different ways. This gives Python a solid 1st place ranking.
- **Scalability – 1st (tied)**
    - Python is very well supported by AWS Lambda. This makes horizontal scalability very easy, because it is built-in to the Lambda functions. Lambda will initialize another instance and have the two functions process the two MealWrite sessions concurrently. On the other hand, the language Python itself has scalability limitations due to its scripting nature. This could lead to issues with the backend server being a bottleneck, and would require additional work to mitigate. As previously mentioned, this is not a large concern due to the power of Lambda functions.[18]
    - Python is supported by Lambda functions so it gets a 1st place ranking.

Java

- **AWS Integration – 3rd**
    - AWS has a SDK for Java that includes some useful resources such as a good way to program asynchronously. Setting up the SDK is a bit tedious, but it is not terrible. The team would have to define many dependencies, and the dependencies differ depending on what type of project is being set up (Maven, Gradle, Graal, etc.). After initialization, importing AWS

[17] "Quickstart." https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html. Accessed 7 Mar. 2022.
[18] "Lambda function scaling." https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html. Accessed 7 Mar. 2022.

services is confusing and requires multiple “import” lines of code. In addition, there are many features that are not supported in the SDK for Java yet: DynamoDB APIs for example.[19]
    - As mentioned above, many services are missing. This gives Java a 3rd place ranking.
- **Ease of use with Alexa – 3rd**
    - Just as it provided an AWS SDK for Java, Amazon has an ASK SDK for Java. The set up for this SDK is quite perplexing and there are many modules and components that must be incorporated. In addition, there is a short list of stable Alexa Capabilities supported by this SDK as of today. Barring these shortcomings, it is possible to use Java for Alexa skills and the code itself gets the job done through request handlers.[20]
    - Due to the missing features and limited capabilities, Java gets a 3rd place ranking.
- **Scalability – 3rd**
    - Java’s horizontal scalability is decent when speaking of the language itself. It has the facilities to have transparent clusters and multiple instances. However, as previously mentioned, Java is not completely supported by AWS or ASK SDKs yet. A huge factor in determining scalability for this application resides in the capabilities of Lambda functions. These Lambda functions are not fully supported by Java at this time so it makes for a tougher time when trying to handle multiple instances.
    - Java is not supported by Lambda functions so it gets a 2nd place ranking.

3.4.4: Analysis Overview

Python is selected as the optimal solution for MealWrite’s client session and backend management, as it stands out from the other options in every category. The rankings in each criteria and overall ranking are presented next in table 3.8.

[19] “AWS SDK for Java.” https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/home.html. Accessed 7 Mar. 2022.
[20] “ASK SDK for Java.” https://developer.amazon.com/en-US/docs/alexa/alexa-skills-kit-sdk-for-java/overview.html. Accessed 7 Mar. 2022.

Table 3.8: Client Session and Backend Programming Language Criteria Ranking

Language	AWS Integration	Ease of use with Alexa	Scalability	**Average**
Python	1st	1st (tied)	1st (tied)	1st
Node.js	2nd	1st (tied)	1st (tied)	2nd
Java	3rd	3rd	3rd	3rd

3.4.5 Proving feasibility

The feasibility of this solution will be shown through a demo of the backend, written in Python, reading information from DynamoDB and sending it to an Alexa skill or the mobile app.

4. Technological Integration

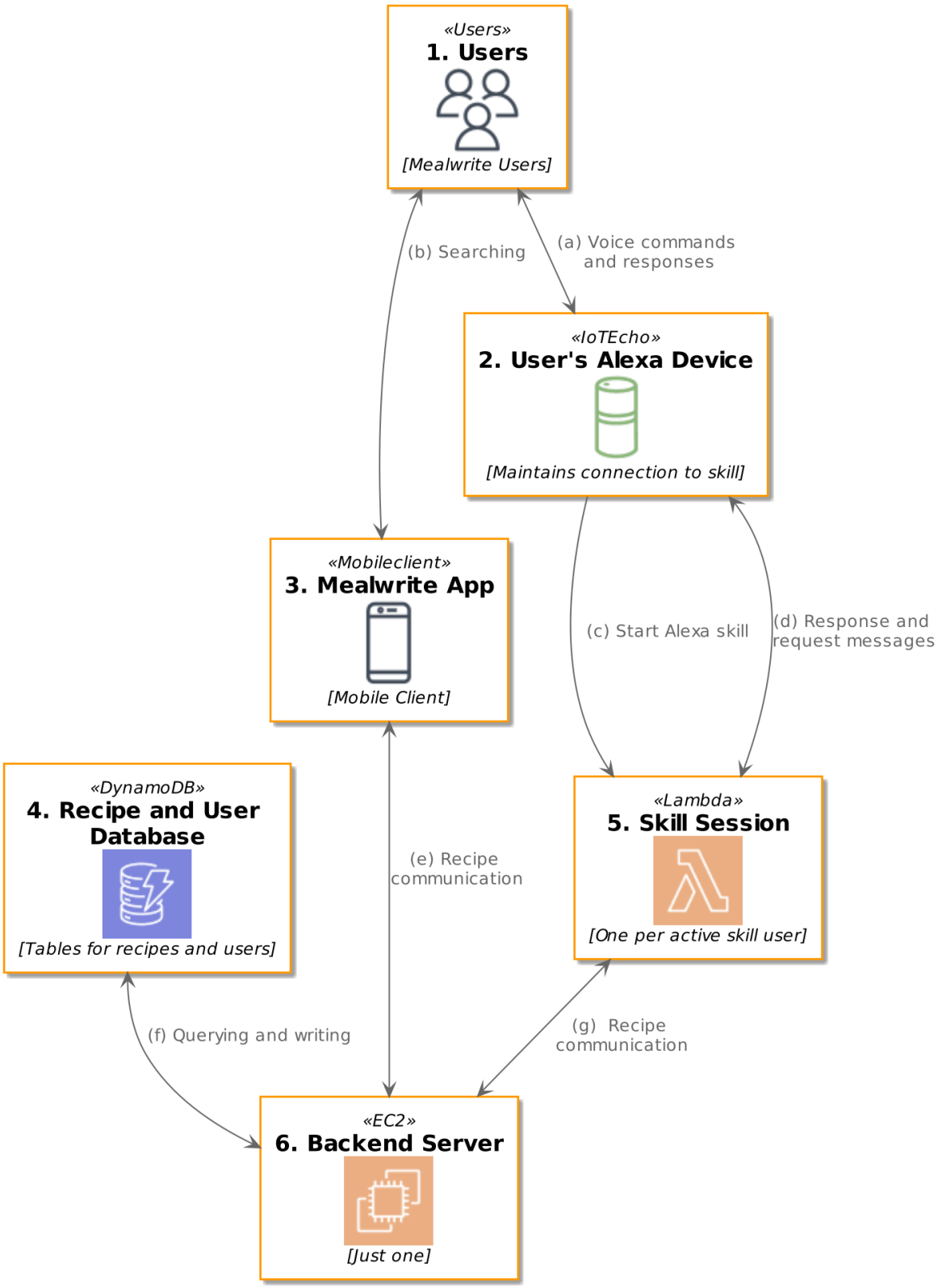
This section explains the technological choices that have been decided upon and how they will combine together into a cohesive architecture. Figure 4.1 shows the envisioned architecture, including the preferred technologies assessed in this document. The components which did not require analysis are AWS's EC2 server instances, AWS's Lambda sessions, and Android as the mobile companion app target platform. EC2 is AWS's general purpose compute server platform, offering long term servers that are ideal for hosting the application's backend. Lambda is AWS's serverless short term compute platform, and is tightly integrated with Alexa, being the preferred platform to host Alexa skill sessions on. Android was selected because development targeting iOS requires a Mac device to compile on, which would exclude several of the development team members from being able to effectively work on the app. Each component in Figure 4.1 is discussed in its role within the architecture next.

1. **Users** – These are the users of MealWrite. (a)(b) They directly interact with the Alexa devices and the MealWrite app, the Mobileclient, which provides transparent access to the back end services.
2. **User's Alexa Device** – This is the user's Alexa capable device. (a)(c) The user tells the Alexa device to start the MealWrite skill and this initializes a skill session. (a)(d) The user then talks to the device, and these communications are translated into communication objects that are sent to the Lambda skill session. The Lambda skill session then does its processing and returns a communication object instructing the Alexa device how to respond.

3. **MealWrite App** – This is MealWrite's mobile app for Android. (b) The user will use this app to search for recipes to prepare with Alexa later. (e) It does this by taking textual and conditional logic from the user and constructing this into a query object that is sent to the backend server. The backend server then does its processing and returns a query response object.
4. **Recipe and User Database** –This is the database holding MealWrite's data. It will contain two tables, those being the recipe table and the user table. (f) The backend server is the only component allowed to communicate with this component, conducting queries and updates, providing security for it.
5. **Skill Session** – This is the main processing center of MealWrite. It is a session that holds the current processing of the recipe state and user's session. (c) When it is started by the Alexa it sends a communication to the Alexa asking the user what they want to cook, then receives a response object specifying the recipe to be prepared. (g) The skill session then constructs a query object from this and sends the request to the backend server. The backend server then responds with the desired recipe. From this point the session does not communicate with the backend any more. (d) The skill session then goes into processing mode, where it manages the scheduling of the recipe, processes incoming communications from the Alexa, and constructs reply objects to return to the Alexa.
6. **Backend Server** – This is the backend of MealWrite. (e)(g)(f) It processes incoming query objects from the app and skill sessions by making appropriate calls to the DynamoDB database, and when a query object from the skill comes in, looks up the associated user and updates to keep track of the recipe they have requested as well. It then returns the found recipe or recipes to the requesting service, for them to process in their appropriate methods.

Figure 4.1: Proposed Architecture

Diagram illustrates the proposed architecture of the solution



5. Conclusion

Home-cooked meals can be hard to make. There is a lot that goes into making breakfast, lunch, and/or dinner. This document has previously mentioned these issues, identifying points such as time management and task handling. MealWrite plans to solve these issues through offering an Alexa skill to people who want home-cooked meals, but want the process to be as easy as possible.

Table 5.1 showcases the challenges that were identified for MealWrite, and the chosen solutions for each challenge. Team Rat in a Hat will be using Dart for their mobile app as it was discovered it fits the requirements the best. For database and recipe storage, DynamoDB will be used because it was by far the most appropriate for the project. Finally, Python will be used for all backend and client session duties.

Table 5.1: Selected Solutions

Technological Challenge	**Envisioned Solution**
App Programming Language	Dart
Recipe Storage Requirements	DynamoDB
Client session requirements	Python

Through this analysis, the main challenges that will have to be dealt with during development have been recognised, and the team has assessed optimal technological solutions for them. Team Rat in a Hat are confident that they will deliver a functional meal preparation app and make home-cooking that much easier and efficient.