

Technological Feasibility

Team Shining Sky

Rosze Voronin, Skyler Hanson, Ashleea Holloway, Logan O'Donnell

November 1st, 2021 - Version 3 (Final)

Sponsored by: Dr. Morgan Vigil-Hayes

Mentored by: Felicity Escarzaga



	2
1. Introduction	3
2. Technological Challenges	4
2.1. Challenge 1 - Framework	5
2.2. Challenge 2 – Framework Platform	10
2.3. Challenge 3 - Database	20
2.4. Challenge 4 - System Integration	25
3. Technology Integration	33
4. Conclusion	34
5. References	35

1. Introduction

Native Americans living in tribal communities lack access to things such as economic opportunity, well-funded education, and youth programs [1]. Youth feel a disproportionate, negative impact from these unfortunate realities, leading to higher rates of suicide, depression, anxiety, and substance abuse [2]. Behavioral and mental health resources are scarce in tribal communities, and where they do exist, there is often a negative stigma associated with using the services [3].

Those looking to address these problems invest in mentorship programs and other ways to provide mental health resources or direct assistance to those who are struggling. The client, Dr. Vigil-Hayes of Community Aware Networks and Information Systems (CANIS), has been working with the Hopi to address the youth mental health crisis. The ARORA app, created by CANIS lab, aims to provide youth users on the Hopi reservation access to mental and behavioral health resources and gives them the ability to fill out mood reports to track their feelings and progress. While testing the app with members of the Hopi community, Dr. Vigil-Hayes was informed of a desire to integrate the ARORA project with an upcoming mentorship program created by the Hopi community. To do this, they would need ARORA to be expanded to include some of the new desired features and an additional app to provide software support for the mentorship program, which led to the creation of this project. The mentorship app project represents a way to boost the efficacy of both the ARORA project and the new mentorship program by allowing them to work in tandem.

The solution is an additional mobile application (for both Android and iOS) that implements the desired features needed to support the mentorship program. The app will serve as a companion for the existing ARORA app, connecting through the ARORA server and allowing the mentorship program to interact with the resources already present in the ARORA project. It will provide the following major features:

- Integration with Existing System
- Anonymous Questions/Answers
- Chat and Emailing

- Mentor/Mentee Meeting Scheduling
- Question Board
- Mood Report Display
- Mentor Authentication
- Server Management
- Database Management

With a more abstract view of the ARORA mentorship app project established, the specific technological challenges inherent to the client's vision of the mentorship app will now be outlined.

2. Technological Challenges

The technological challenges involved in the development of the ARORA mentorship app are as follows:

- Framework - This challenge addresses the need for an app development framework from which the ARORA mentorship app can be built.
- User Interface / User Experience - This challenge addresses the need for a framework platform capable of providing a usable mobile interface.
- Database - This challenge addresses the need for a database framework that allows us to store and work with information both unique to our application and sent in from the ARORA server.
- Integration - This challenge addresses the need for the new ARORA mentorship app to interact with and work alongside the existing ARORA server.

The following sections will provide detailed descriptions of each technological challenge, as well as the criteria that will be used to evaluate the candidate solutions for those challenges. Finally, the integration of all chosen candidates will be discussed to create a complete overview of the technologies to be used in the development of the ARORA mentorship app.

2.1. Challenge 1 - Framework

The ARORA mentorship app needs a framework with which the program can be developed. Each framework is suited for specific languages and mobile or web-based platforms, factors which must be considered when finding a framework that is well suited for developing this project. Potential frameworks were found by researching app development frameworks, as well as the differences between them. The frameworks were further verified by searching the name of each along with the word “alternatives”, to confirm that these were popular or widely recommended frameworks. Finally, each framework was confirmed to currently be supported for app development, to control for potentially outdated information.

Desired Characteristics

The characteristics considered when deciding on which framework to use are the framework having reusability of code through codebases, the popularity of its primary language within the software development community, and the platforms for which the framework supports development. A framework with shared codebases will allow for the same code to be compiled for all platforms without the need to refactor a version of the code for each platform. An ideal framework would offer development support for Android, iOS, and if possible, web development. It is also important to consider the popularity of the language to be used when creating the app, as it will impact the ability of future developers to add additional features to the ARORA app. If too obscure of a language is chosen, it may be difficult for the development to continue after the initial release.

Alternatives

Flutter was created by Google starting in 2015 and was released in 2017. While its engine is written in C++, most Flutter development is done in the Dart language. Flutter offers wide platform support, including Android, iOS, Linux, Mac, Windows, and web [4].

React Native was created by Facebook as an attempt to improve the overall experience of the mobile Facebook application and was released in early 2015. It is primarily written in JavaScript. React Native supports development for Android, iOS, macOS, Windows, and web [4].

Xamarin was created by a company of the same name in 2011, though it is now owned by Microsoft. It is written in C# and offers support for iOS, Android, and Windows App development [4].

Cordova was created by Nitobi in 2009, which was purchased by Adobe in 2011. The Cordova framework was renamed Apache Cordova, also known as PhoneGap. Updates to PhoneGap were officially discontinued in October of 2020, though it is still usable as a development tool. This framework allows for app development using the JavaScript language. The supported platforms include Android, iOS, Ubuntu, and Windows [4].

Analysis

To analyze the candidates, research was conducted on each framework to determine the popularity of its primary language, which of the three relevant platforms (Android, iOS, and web) are supported by it, and whether the framework uses a shared codebase between these platforms [4]. While Android support is mandatory for the chosen framework, and iOS support is strongly desired, a web version is optional, meaning that the chosen framework may not necessarily support it, if that framework is the best choice in other categories. Finally, the popularity of the primary language of each framework was researched.

Information about the popularity of each language was gathered from a 2021 Statista survey [5] with 83,052 respondents. Points will be awarded in 20% intervals for this metric, such that a language known by fewer than 20% of developers will receive a score of 1, one known by 20% to 40% of developers would receive a score of 2, and so on, such that a language will receive the maximum of 5 points only if at least 80% of developers know it.

If a framework uses a shared codebase between all of its supported platforms, it will receive 5 points. If it uses a shared codebase between some of the platforms relevant to this project (being Android, iOS, and web), but not for all of them, it will receive 3 points. If the framework does not use a shared codebase, it will receive 1 point.

For each of the non-mandatory platforms, the candidates will receive a score of either a 1 (if the platform is not supported) or a 5 (if the platform is supported). For the optional criteria of web support, the points given will only be used as a tiebreak if all other metrics are equal between two candidates. Now that how each candidate will be scored has been established, the findings and final rankings of each candidate will be described in the next section.

Flutter

Flutter uses the Dart language, which is only known by 6.02% of developers, according to the Statista survey [2]. This means that it will receive 1 point in the popularity category. Flutter offers support for iOS and web support, so it will receive 5 points for each of these criteria. Flutter uses a shared codebase between all of its supported platforms, so it will receive 5 points in the final category.

React Native

React Native uses JavaScript, a language known by 64.96% of developers. It will receive 4 points in the popularity category. As it offers support for both iOS and web support, it will also receive 5 points for each of the remaining categories. React Native uses a shared codebase between all of its supported platforms, so it will receive 5 points in the final category.

Xamarin

Xamarin is written in C#, which is known by 27.86% of developers. This means Xamarin will receive 2 points for language popularity. Xamarin does offer iOS support

and will receive 5 points as a result. However, it does not support web development, so it will only receive 1 point in this category. Xamarin uses a shared codebase between all of its supported platforms, so it will receive 5 points in the final category.

Apache Cordova

Apache Cordova, much like React Native, is written using JavaScript, and will likewise receive 4 points for language popularity. It also offers support for iOS, earning it 5 points in that category, but does not support web development, so it will only receive 1 point in the optional web category. Apache Cordova uses a shared codebase between all of its supported platforms, so it will receive 5 points in the final category.

Table 2.1.1: Framework Candidates and Metrics

	Flutter	React Native	Xamarin	Apache Cordova
Primary Language	Dart	JavaScript	C#	JavaScript
Popularity	1/5	4/5	2/5	4/5
iOS Support	5/5	5/5	5/5	5/5
Web Support (optional)	5/5	5/5	1/5	1/5
Codebase	5/5	5/5	5/5	5/5

Chosen Approach

React Native and Apache Cordova are the most appealing options with regards to language, as JavaScript is known by 64.96% of developers. However, while all four options support iOS app development, only Flutter and React Native match our optional requirement of support for web development. All four frameworks also offer a shared

codebase between all of their supported platforms, meaning that code will not need to be refactored when developing for more than one platform.

When looking at Table 2.1.1, React Native seems to be the clear choice for a framework. It is written in JavaScript, the most popular language according to the Statista survey [5]. This means that finding developers to continue work on the ARORA app after its initial release will be relatively easy. It also provides support for both the required platforms, as well as the optional web support. Apache Cordova is a possible alternative, should React Native not be able to be integrated with the other chosen technologies, as it is also written in JavaScript. Unlike React Native, however, Apache Cordova does not offer web support. Xamarin would be the next best option, as at least 27.86% of developers are familiar with C#. This choice would also mean dropping support for a web version. Finally, Flutter seems to be the worst choice, as despite offering support for a potential web version of the app, it is written in a language with which only 6.02% of developers are familiar.

Proving Feasibility

The feasibility of this framework choice will be proven by confirming that it can be used to build an Android and iOS ready demo which can meet the basic requirements of the project. These requirements include the ability of users to send anonymous messages to mentors in the program, and for mentors to be able to respond to these messages, as well as to be able to store information about participants in a database. If possible, this demo would also be ported to a web version, to confirm that the optional project goals are also attainable.

2.2. Challenge 2 – Framework Platform

Framework platforms are interactive IDEs that serve as a location for a developer to write, format, and run a program in. Framework platforms may have support for pre-made modular code packages, live previewing or server-based runtime, and various time-saving tools for coding implementation. The platform best suited for this project will be one meeting the needs of the client and the needs of connecting to the existing ARORA app.

To start, the existing ARORA app's server already used by CANIS lab can send information and be accessed by the ARORA mentor app being implemented. The server information sent by the existing ARORA app includes anonymous questions, mentee mood reports, and emails or chat messages sent by mentees addressed to a mentor. Therefore, the ARORA mentor app to be implemented must be able to support the concepts of a filterable scrollable list of all answered and unanswered questions, a large interactive data display of mood reports, and a chatbox system for sending and receiving messages. Implementing these functional needs with the framework platform chosen will contribute to creating a functional utility for managing Hopi's Community Mentorship Program, allowing mentors to be able to directly connect with their mentees. In the next section, the specific characteristics the team will be looking for to implement these functions will be described.

The platform should also support exporting a single implemented program to iOS, Android, and potentially a web application. The higher the number of platforms it can export to, the better, to drastically lower time spent on implementation. It also eliminates the need to duplicate and rewrite the same program for the different app platforms.

Desired Characteristics

The desired qualities for the framework platform will include various functionality, starting with a chat box and scrollable list implementation. Dedicated packages or components for chat boxes and scrollable lists are desired, as the minimum basic

requirements for a framework platform should be to at least give the ability to create the functionality from scratch. Then we are looking for our interactive data display to easily update based on incoming server information, which would include the mentee mood reports. Therefore, the framework platform should have a wide range of database and server support. This is to give the client the ability to change or update the database or server being used, with minimal code being revised. Next, the platform's modularity, components, and troubleshooting should have a large amount of documentation. This not only supports the current Shining Sky team in learning the platform, but also any future developers underneath the client. Finally, the platform framework would preferably be free to use, as there is no budget for the mentor ARORA app being created by the team. In the next section, the potential candidates that may meet these desired characteristics will be described. The candidates will include Ionic, Expo, FlutterFlow, and Xamarin.

Alternatives

The first candidate, Ionic, was found recommended in mobile app development forums. Ionic is an open-source mobile development framework that is cross-platform for iOS, Android, and web applications. It was created by Max Lynch, Ben Sperry, and Adam Bradley of Drifty Co. in 2013, with a recent stable release in mid-2021[6]. Ionic has been used in the past for nonspecific various apps.

The second candidate is Expo, which is a platform for React Native specifically. React Native is the framework that the existing ARORA app uses, and is recommended by the client. Research of React Native led to the discovery of Expo and its documentation. Expo was created by Charlie Cheever who started working on it in 2015 [7].

The third candidate is FlutterFlow, which had been mentioned as an option in mentor meetings and was found through research on cross-platform options. It was initially released in May 2021 by Google. Flutter overall is the main choice of developers wanting many cross-platform options, with Flutter currently including Android, iOS,

Linux, Mac, and Windows [8]. Since FlutterFlow is a recent creation, it is unlikely that any major apps have been developed using FlutterFlow.

The last candidate is Xamarin, which came up in research as a framework with one of the largest contributing communities, with its total members being over 100,000 people [9]. It was released by the Xamarin software company, which is owned by Microsoft, in May 2011. The framework has been used for a variety of utility and gaming apps. Concluding our candidate Xamarin, the following section will describe how each of the candidates was scored and the final results.

Analysis

To analyze the potential candidates, they will be judged using the previously stated desired qualities for the Framework Platform challenge. The characteristics, and therefore the metrics, will be functionality support, database and server support, range of documentation, and cost to use. Each of these metrics will be scored on a 5-point system, with each point being Very Low, Low, Medium, High, and Very High respectively. A description of each candidate's scores will now be provided.

Ionic

Starting with the functionality metric, Ionic shows direct support for an infinite scrolling list system. This is done with their "ion-infinite-scroll" component, which allows for a program action to be called when a user has scrolled down or up a specified distance. This action can be for the program to call for a refresh of incoming anonymous questions, which can then be immediately displayed. Ionic also supports the use of Stream Chat, which is a chat API for creating chat messaging. However, since this is an external API that must be used, and Ionic does not have an internal package or component for a chat app, Ionic loses a point in the functionality metric. As Ionic meets both of the desired components for the ARORA mentee app, it scores High, which is a 4/5 on the 5 point system.

Next, Ionic then shows support for multiple databases, including MySQL, SQLite, MariaDB, Node JS, and its own named Ionic Secure Storage for our database and

server metric. However, it does not use direct integration, and any database queries must use an API as a middleman connection. Although this does not eliminate its ability to access databases, it is something to be considered when compared to other framework platforms that may have direct integration. Therefore, Ionic loses a point for its lack of direct integration for the functionality metric. For server support, Ionic has an HTTP module that gives the ability for a program to retrieve data directly or utilize an API to receive passed server information. The existing ARORA mentee app owned by the client uses a REST API for their server, which is one of the supported APIs Ionic can use. Therefore, Ionic receives full points for the server implementation subsection of our functionality metric. Overall, Ionic received High, or 4/5 in functionality, with the one point lost representing the inability for direct database integration.

Ionic also has strong documentation for component creation for the documentation metric. Ionic is essentially a UI toolkit, with a documented library of building blocks that can be immediately plugged in and used in a program. For specifically UI components, 87 documented elements include descriptions, tutorials, and example code shown in multiple framework languages. There is also general documentation related to developing, available utilities, deployment, troubleshooting, and more. Ionic, therefore, receives full points for the documentation metric, with the score being 5/5, or Very High.

Next, Ionic meets the standards for the exportation metric. Ionic can export a program implementation to iOS, Android, and a web application. The implementation does not need to be different between the platforms, eliminating the need to duplicate and edit the program structure to fit each target platform. Since the three platforms are the target platforms of the client, and Ionic supports all of them, Ionic receives full points in the App and Website Exportation metric.

Finally, Ionic is free to use and therefore meets the cost to use metric. Ionic is free to use by itself, and the Stream Chat API it utilizes to create a chat box is also free, complimentary to small companies under a specified amount in revenue. Since Team Shining Sky is a student team with no revenue and therefore Stream Chat is free, Ionic

receives full points for the cost to use, as the cost to use is \$0. The next candidate to look at will be Expo.

Expo

The third candidate, Expo, has internal support for a chat app implementation using a package called “react-native-gifted-chat” [10]. Expo is also capable of creating a scrolling list, with its “ScrollView” or “FlatList” component. Both of the list components use infinite scroll loading, enabling refresh of incoming server information to display incoming mentee mood reports from the existing ARORA app. Since Expo has an internal ability to do both of the functionality requirements without the use of external APIs or sources, Expo earns full points, 5/5, in our functionality metric.

For the database and server support metric, Expo can utilize WebSQL, SQLite, and Node JS. For specifically an SQLite database, Expo has internal components for accessing, opening, and editing the database, without an external API. Similar to Ionic, Expo also uses an HTTP request system when attempting to obtain information from a server. It also does support the REST server API, previously stated as the API that the existing ARORA app uses. Expo’s scoring on the database and server support metric is therefore 5/5, as it can access a database and server internally.

Expo is also a well-documented platform with component documentation being written in a tutorial-based format. It provides examples of functional code and explains in detail how to use it, where to use it, and what the outcome of the code can be. It has 120+ documented components, as well as numerous guides including app distribution workflow, UI programming, and more. It also includes a document for database interfaces, including a database object which would be in alignment with the ARORA mentor app the team is creating. Therefore, Expo earns full points, or 5/5, on the 5 point scale for the documentation metric.

Expo, however, does not support reusing the same code across platforms for our exportation metric. Most often Expo does not have web support for a large number of its components, including database queries for specifically the web. As one of the goals of

the ARORA mentor app is having web support, Expo loses one point in the exportation metric, for not supporting one of the target exportation platforms. Having exportation to Android, iOS, and the Web allows the team to meet all target platforms the client has specified, as well as utilizing time by not rewriting the program for each of the platforms. Expo would have lost two points if it lacked iOS or Android support, but web support is not necessarily required and is a stretch goal. Therefore, Expo receives a 4/5 in the app and web exportation metric. For the cost to use metric, Expo is currently open-source and free to use, therefore earning the full 5/5 points. The next candidate to look at will be FlutterFlow.

FlutterFlow

The third candidate, FlutterFlow, lacks packages and components for a chat box interface and must use an external API called Firebase. Since the program does not handle chat box implementation internally, besides manually, and instead relies on an external API, FlutterFlow loses a point for the functionality metric. FlutterFlow also lacks exact components for the creation of an infinite list. The infinite scrolling list must manually be created using container and list item widgets, and cannot intuitively access incoming server information to update the list. Since FlutterFlow lacks both an API and manual creation components for infinite scrolling creation, FlutterFlow loses another two points for the functionality metric, with a final score of 2/5.

Documentation for FlutterFlow components is lacking, likely because it aims to be a drag and drop implementation platform specifically inclined to make the design flow of visual UI faster. It had only 30 documented components, and then only a few general sections such as working with an API named FireBase, deploying an app, and troubleshooting. As we are unable to demo FlutterFlow to test to see if more documentation is shown within the app due to the paywall, FlutterFlow receives a 2/5 on the 5 point scale system. The 3 points lost relate to lack of database documentation, server access documentation, and component documentation.

For the app and web exportation metric, FlutterFlow lacks support for exporting to a web application. FlutterFlow does support the ability to export a single program to

both iOS and Android, but cannot export to a web application even with separate program implementation. Previous candidates lost a point in the exportation metric for not supporting a program being able to export to a mobile application as well as a web application, but FlutterFlow cannot create a web application at all. Therefore, FlutterFlow loses two points for such an inability.

Finally, FlutterFlow loses all points for the cost of use, because a membership is required to use the platform. FlutterFlow does have a free plan, but it lacks needed tools such as being able to download the program's APK, being able to test the code on a mobile device, and being able to use server and database APIs. Server and database APIs are strongly needed for meeting the requirements of our client and for the app, and the ability to use such with FlutterFlow is under a \$70 a month membership plan. As there is no budget for Team Shining Sky, FlutterFlow receives 0/5 in the cost to use metric. The next candidate will be Xamarin.

Xamarin

Our final candidate, Xamarin, had supported the chat box and infinite scrolling functionality needs using its "Forms" component. However, Microsoft announced in 2020 that its forms component would be deprecated and replaced with a new system. Since the relevant information for specifically our chat box implementation will be using brand new components, there is a potential for lack of community documentation. As the team cannot determine whether or not Xamarin will still have the functionality we need for the chat box and infinite scrolling elements after the deprecation, it currently loses all points for the functionality metric.

For the database and server metric, Xamarin has some support for database access in their "Forms" component, but only supports an SQLite.NET database. Due to the lack of a wide range of supported databases, and the supported databases not including the current database being used by our client, Xamarin loses 3 points. Then, Xamarin's incoming server access is another concept in our desired characteristics that uses Xamarin's deprecated "Forms" component. Therefore, Xamarin loses the last 2 points and results in having a 0/5 in the database and server metric.

Xamarin's documentation is severely lacking compared to the other candidates. Xamarin only has around 12 tutorials for using its elements, along with some documentation for its "Forms" component that will be deprecated. Since we lack documentation for chat box, infinite scrolling, database access, and server access due to "Forms" deprecation, Xamarin loses 4 points. Xamarin keeps 1 point for its unrelated documentation, with its final score being 1/5 in the documentation metric.

For the app exportation metric, Xamarin does not support sharing code across platforms, and it does not support web-based applications at all. Since we need both Android and iOS support, the team's time utilization would have to account for duplicating and editing a program solution for each platform. Then, since Xamarin does not support web-based applications at all, a potential stretch goal cannot be met. Therefore, Xamarin loses 4 points, only keeping 1/5 points in the app expiration metric since it still can create an Android app and iOS app.

Xamarin is currency-free, along with the .NET platform that it resides under. There are no needed APIs we must use with Xamarin to meet our requirements and desired characteristics, so the cost of Xamarin stays at \$0. Since Team Shining Sky has no budget to use, and therefore \$0 is ideal and needed, Xamarin receives a full 5/5 points in the cost to use metric.

Now that each candidate has been scored with metrics based on our desired characteristics, the results have been specified in Table 2.2.1, titled "Framework Platform Candidates and Metrics". The table has been color-coded based on the individual metric scores and serves as a visualization of the results to help the team conclude which framework platform may be the best choice for the ARORA mentor app to be created. In the next section, the table is used to decide the best choice.

Table 2.2.1: Framework Platform Candidates and Metrics

	Ionic	Expo	FlutterFlow	Xamarin
Functionality	5/5	5/5	4/5	0/5
Database and Server Support	4/5	5/5	2/5	0/5
Documentation	5/5	5/5	2/5	1/5
App Exportation	5/5	4/5	3/5	1/5
Cost To Use	5/5	5/5	0/5	5/5

Chosen Approach

After researching, Ionic proved to have excellent documentation, including example code to be utilized for its basic element structure. However, it lacks dedicated elements for chatbox creation, which would have to be done manually. Expo shows the most features for specifically react native, which is the framework the ARORA app is made in, but is less intuitive in its use compared to Ionic. FlutterFlow lacked the components, app exportation, documentation, and database support we needed for our requirements, all while being the only pay-to-use platform out of our candidates. Xamarin at one point did have a component that would enable all of the functions we needed, but the component is in the process of being deprecated, and only supports one database.

Using this table, the final ranking has come down to Ionic being tied with Expo, followed by FlutterFlow, and then Xamarin in last. Ionic's main flaw was only in the functionality metric, as it relies on an external API for chat box implementation, which

was its only lost point. Expo on the other hand does have such an internal package, but it does not allow code to be shared across platforms as Ionic does. For this reason, Ionic takes the lead over Expo, as although Expo has more packages, Ionic can export to multiple platforms. Since the app needs to be able to export to iOS, Android, and potentially a web application, the weight of Ionic's app exportation score puts it at a higher ranking than Expo. Xamarin comes in third, mostly due to the recent depreciation of a major component. Finally, FlutterFlow comes in last, as it lacks components for our functionality, database, and server requirements. The next section will describe our plans for going forward with our top candidate.

Proving Feasibility

Ionic is the top candidate out of the demoed four, although further demoing must be done. Ionic will be further tested through more demos which will include setting up an aesthetic mockup/prototype of the UI, with dynamically changing elements. Being able to do such a realistic mockup and working to match the elements to the existing ARORA app UI will be able to support in-depth analysis on if the framework is a good fit for the current user base and client. If it demonstrates that it can create a chat box UI, create an infinite scrolling list from server input, and have database and server support, it is a good candidate for this project.

2.3. Challenge 3 - Database

The ARORA application needs to be storing data from the user and sending it to the ARORA server. Furthermore, data from the server's database needs to be available on the app. The project needs a platform that will allow it to transfer data between the ARORA server and application with ease. The language used to carry out these requests is Structured Query Language, or SQL.

Desired Characteristics

The first characteristic measured is cohesiveness with the preexisting ARORA system. The ARORA system utilizes Django and manage.py to modify the database and its records. Next, a platform that is easy to use is also desired to reduce the time for developers to make changes and implement new ideas. A candidate's popularity is another factor gauged. If the team has not worked with a given candidate before and/or there are few users in total, developers will be spending much of their time learning how to make things work instead of actually working.

Alternatives

The first candidate is MySQL. MySQL is a relational database management system that is based on SQL. It is free and open-source through the "GNU's Not Unix", or GNU, general public license. MySQL can be run in the terminal or users may download software like MySQL workbench. It also had clients that allow users to interact directly using SQL. It is far more common though for users to pair MySQL with other programs to implement relational database capabilities with existing applications. MySQL is used by many entities including NASA, US Navy, Facebook, Netflix, Amazon, Twitter, etc [11].

The second candidate is MariaDB. Although MariaDB is a fork of MySQL, it is still commercially supported and community-developed. When Oracle Corporation purchased MySQL in 2009, some of the original developers forked it due to Oracle Corporation's plans to no longer provide a free and open-source product. MariaDB is

built to remain compatible with MySQL APIs and commands. Since 2009, new features have started to further define their individuality [12].

The last candidate is SQLite. SQLite is another relational database management system but is contained in a C library. Unlike other candidates, SQLite is not a client-server database engine. Instead, the developers have put it into the end program. In regards to the project, it is important to remember that it uses a dynamically and weakly typed SQL syntax that does not guarantee domain integrity. SQLite is a popular choice for local/client storage in application software such as web browsers. It is used by several widespread browsers, operating systems, and embedded systems (such as mobile phones) [13].

Analysis

To score these candidates, several attributes that are believed to be the most relevant and important were selected. The characteristics are as follows: the ability to work with pre-existing systems, ease of use, and popularity.

A candidate's cohesiveness was judged by its ability to work with software already in use on the ARORA server and by making a simple table of data and attempting to manipulate it. If a candidate can manipulate the data with ease, it will receive a 5/5. If it is difficult or not possible to work with, it will receive a 0/5.

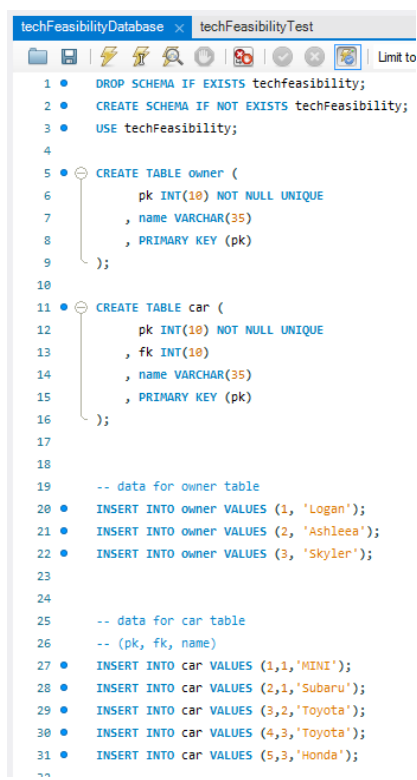
To calculate the subjective metric of "ease of use", unexpected errors and the simplicity of their resolution will be noted. If a candidate is generating many errors and the solutions are difficult to understand or find, it will receive a 0/5. If the candidate does not produce errors, suggests corrections, and/or the solutions are simple and popular, it will receive a 5/5.

The last metric is popularity. If the team is already familiar with a database system and there are a plethora of other users, then the implementation of a database would be more efficient. The more a candidate is with the team and other users, the higher score the candidate will receive. If the team has never worked with a candidate

and there is a small user base, it will receive a 0/5. If the team is familiar with a product and there is a large user base, a 5/5 is appropriate.

MySQL

The first candidate that was tested was MySQL. For the first half of cohesiveness, MySQL works well with both Django REST API and manage.py. For the second half, a table of car companies and owners was created and is shown in Figure 2.3.1. The purpose of creating these tables was to gauge how straightforward the process was. After the tables were made, they were manipulated to see the information in a clean format; this is shown in Figure 2.3.2. MySQL did very well in these tasks. The team is familiar with MySQL workbench which makes the setup simple. Furthermore, MySQL documentation is abundant and helpful. When errors are introduced to the code, MySQL Workbench does a good job of explaining the error. MySQL received a 5/5 for cohesiveness, a 5/5 for ease of use, and a 5/5 for popularity.

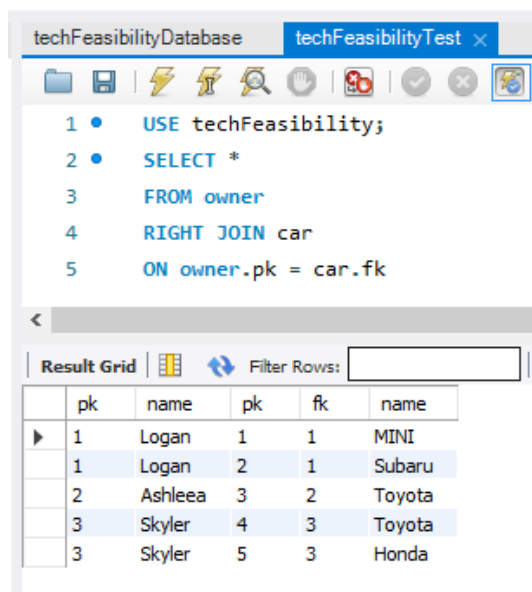


```

techFeasibilityDatabase x techFeasibilityTest
1 DROP SCHEMA IF EXISTS techFeasibility;
2 CREATE SCHEMA IF NOT EXISTS techFeasibility;
3 USE techFeasibility;
4
5 CREATE TABLE owner (
6     pk INT(10) NOT NULL UNIQUE
7     , name VARCHAR(35)
8     , PRIMARY KEY (pk)
9 );
10
11 CREATE TABLE car (
12     pk INT(10) NOT NULL UNIQUE
13     , fk INT(10)
14     , name VARCHAR(35)
15     , PRIMARY KEY (pk)
16 );
17
18 -- data for owner table
19
20 INSERT INTO owner VALUES (1, 'Logan');
21 INSERT INTO owner VALUES (2, 'Ashleea');
22 INSERT INTO owner VALUES (3, 'Skyler');
23
24
25 -- data for car table
26 -- (pk, fk, name)
27 INSERT INTO car VALUES (1,1,'MINI');
28 INSERT INTO car VALUES (2,1,'Subaru');
29 INSERT INTO car VALUES (3,2,'Toyota');
30 INSERT INTO car VALUES (4,3,'Toyota');
31 INSERT INTO car VALUES (5,3,'Honda');
32
33

```

Figure 2.3.1: Generating Tables



```

techFeasibilityDatabase x techFeasibilityTest x
1 USE techFeasibility;
2 SELECT *
3 FROM owner
4 RIGHT JOIN car
5 ON owner.pk = car.fk

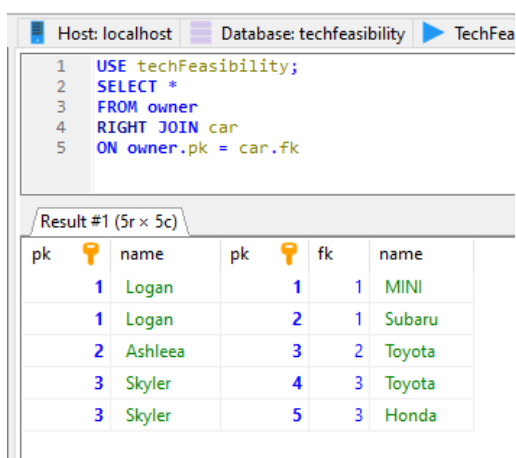
```

	pk	name	pk	fk	name
▶	1	Logan	1	1	MINI
	1	Logan	2	1	Subaru
	2	Ashleea	3	2	Toyota
	3	Skyler	4	3	Toyota
	3	Skyler	5	3	Honda

Figure 2.3.2: Combining Tables

MariaDB

The second candidate is MariaDB, which was tested the same way as MySQL as seen in figure 2.2.3. MariaDB also works with pre-existing software from the ARORA server. MariaDB is similar enough to MySQL that code was able to be reused from the previous test. The tool used when downloading MariaDB is HeidiSQL which is less user-friendly than MySQL Workbench. When introducing a syntax error into the code, HeidiSQL introduces a popup and plays the audio cue that an error occurred. This is both unnecessary and bothersome. It received a 5/5 for cohesiveness, a 4/5 for ease of use, and a % for popularity.



```

1  USE techFeasibility;
2  SELECT *
3  FROM owner
4  RIGHT JOIN car
5  ON owner.pk = car.fk

```

pk	name	pk	fk	name
1	Logan	1	1	MINI
1	Logan	2	1	Subaru
2	Ashleea	3	2	Toyota
3	Skyler	4	3	Toyota
3	Skyler	5	3	Honda

Figure 2.3.3: MariaDB Tables

SQLite

The last candidate is SQLite. While attempting to test SQLite, the team was unable to understand how to get it running. On the SQLite website, it is recommended that users simply enter “sqlite3.db” into a shell or prompt, and that is all. From there, users should be able to create and populate new databases. Furthermore, they give an example program in the tool command language. For these reasons, SQLite was given a 0/0 for every metric.

Table 2.3.1: Database Candidates and Metrics

	MySQL	MariaDB	SQLite
Cohesiveness	5/5	5/5	2/5
Ease of Use	5/5	4/5	1/5
Popularity	5/5	2/5	0/5
User Base	5/5	3/5	4/5

Chosen Approach

Based on the results from testing, the best choice is MySQL. Fortunately, MySQL can work alongside Django REST API and manage.py which is already used by the existing ARORA server. MySQL is easy to use, works with the existing systems and has over 5 Million users which made it the best choice for the project.

Proving Feasibility

To prove the feasibility of using MySQL rudimentary tasks within the application's environment will be performed. These include generating sample tables and data, manipulating the tables, and appending new data. MySQL will be used regularly because it is necessary for backend functionality. Completion of these tasks will prove the ability to use MySQL as the chosen database management system.

2.4. Challenge 4 - System Integration

As this project involves programming an additional app into the existing ARORA system, new code needs to be smoothly and properly integrated. Currently, the ARORA project consists of a central server and a youth-focused ARORA app. This project's app, the mentor-focused ARORA app (mentorship app hereafter), must be able to communicate to the server to both receive data from the existing app and send its data into the pipeline. The server uses a Django-REST framework. The mentorship app will communicate primarily with the server and get any data it needs from the other app using the server as a middleman; this project focuses primarily on that Django-REST interface in terms of the framework used to integrate new code. Of course, there is very little choice in this, since the client likely would not respond well to a request to rewrite their entire server architecture to support a new framework. The choice that does need to be made is how exactly to go about the integration process.

Desired Characteristics

The candidates for this challenge are various methods of integrating new code as it is developed. The first desired characteristic is the complexity of the method. A desirable integration methodology is easy to continually monitor and use for testing and does not cause problems on its own that would take effort away from the development of the mentorship app itself. The second desired characteristic is time investment, that is, how much time utilizing the candidate methodology will take up. This ensures that testing code integration is not an unreasonable time drain on development as a whole, even if it is low effort or low complexity. Finally, the third metric will be the results of research into the sensibility of each methodology. A desirable method should be appropriate and judged to be generally sensible and useful for projects like this one within the larger sphere of the software development industry. These metrics will be used to judge three candidate methodologies: integration testing at the end of development, integration testing against a testing environment managed by the development team, and integration testing against a testing environment managed by the client.

Alternatives

The first candidate methodology is to leave the integration testing until a stable prototype has been completed. The idea behind this would be to develop all or most of the code that needs to be integrated, test it thoroughly, and then finally test how it fits into the larger system. This makes surface-level sense as the mentorship app is the primary priority, and while integration is important, when forced to choose between the two the base features and functionality definitely should receive more attention. However, it does introduce the potential challenge of missing a compatibility issue or otherwise constructing the app in a way that is difficult to integrate, proceeding with development while ignorant of this, and then finding out later on when there are issues with integration and aspects of the code are potentially more difficult to fix or change.

The second candidate methodology is to perform integration testing against a version of the ARORA system managed by the development team. The idea is that code would be tested against a version of the ARORA system under developer control and the client would be asked for support if/when necessary. Since the client has provided access to the source code for the server and the existing ARORA youth app, developers can set up a test environment, run integration tests within that, and bring any issues or questions to the client's attention as needed. This method could prove to be the most complex, however, as it requires that developers familiarize themselves with the backend of the server and the existing ARORA youth app, which could require a lot of time.

The third candidate methodology is to perform integration testing against a version of the ARORA system managed by the client. The underlying idea is that regularly testing integration is good practice, and the client is most familiar with their existing environment. Therefore collaborating with the CANIS Lab on integration testing as directly as possible should be advantageous. This method could be a good way to frequently ensure that the mentorship app is not being developed in a way that introduces compatibility issues, but it has its own challenge in that it introduces more overhead due to requiring extra communication with the client and work on the client's end.

Analysis

The three methodologies will be evaluated using the metrics of complexity, time investment, and sensibility. Complexity will be scored based on the effort for developers to set up a testing environment within the confines of the methodology. It will be expressed on a scale of 1-5, where 1 indicates Very High effort and 5 indicates Very Low effort. An overly complex integration testing process will negatively impact development by forcing developers to focus on setting up integration tests over higher priority tasks, so a lower complexity value represents a better methodology.

The time investment will be scored based on the amount of time taken to implement the chosen methodology on a regular basis (ie, every major iteration of the prototype) as well as estimations on time costs for items outside of direct implementation, such as communications. The time investment will also be scored on the same 1-5 scale, from Very High investment at 1 to Very Low investment at 5. A lower time investment score represents a better methodology since integration testing should not take up development time that is better spent on implementing features.

Sensibility will be scored based on whether the methodology in question is an appropriate choice for the project, as well as whether or not it fits in with the rest of the envisioned development cycle. As sensibility is a subjective metric, part of scoring will include research into industry best practices and which methods are generally considered appropriate for this type of project. Sensibility will use a 1-5 scale, from Very Low sensibility at 1 to Very High sensibility at 5. A high sensibility score represents a methodology that is suited to the details of the project and has the best chance of ensuring smooth and effective code integration throughout the development cycle.

Integration Testing Using a Stable Prototype

Testing code integration once a stable prototype has been produced proved to be an inadvisable method for this type of project. Sources suggested that this methodology was suited best for projects where integration is either very simple or impossible to test easily during development [14]. In this case, integration is an important feature and

developers have access to a preexisting environment, so there is no good reason to wait on integration testing. Therefore, waiting on integration testing's sensibility score will be 1, Very Low sensibility, seeing as it is poorly suited to a project of this nature.

This methodology is overall somewhat complex. It would require only one round of integration testing, performed towards the end of development. The main area where complexity is introduced is potential errors that prevent smooth integration. In the case of errors cropping up, which is safe to assume, the complexity of this methodology is raised, as developers may need to change things on the backend of the mentorship app, which could have cascading effects in terms of development costs [15]. This method would reduce complexity by reducing the number of tests, but likely create a high complexity in what tests remained. Therefore, integration testing against a stable prototype receives a 3, Medium complexity.

Testing code integration against a stable prototype has a middling level of time investment. Theoretically, performing one round of integration testing at the end of the development cycle to prove successful integration is the most time-efficient in terms of time spent on integration testing itself. However, finding out about integration problems later in development vastly increases development costs associated with fixing errors. Therefore, this methodology will receive a 3, Medium time investment, on the grounds that it has a low time investment upfront but likely creates a higher one later on.

Integration Testing Using a Developer-Controlled Environment

The second candidate methodology, integration testing using a developer-controlled environment, proved to be promising. In terms of complexity, the appropriate scoring depended on how difficult it was to run an own instance of the ARORA server. Using documentation provided by the client, setting up an instance of the ARORA server took about thirty minutes and did not create any major issues, as pictured in Figure 2.4.1. The web interface introduced some difficulty since there was no clear documentation on it. Due to the ease of setup but the need to consult with the client to learn how to use the web interface, this methodology will receive a score of 4,

Low complexity. Learning how to use the web interface with the client's help is a non-recurring task, so it does not hold as much weight as server setup.

```
^C(aroraenv) grimoire@pop-os:~/Downloads/ARORA-Server-master$ python3 manage.py runserver
127.0.0.1:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified some issues:

WARNINGS:
?: (urls.W005) URL namespace 'admin' isn't unique. You may not be able to reverse all URLs
  in this namespace

System check identified 1 issue (0 silenced).
October 18, 2021 - 21:17:44
Django version 2.2.1, using settings 'arora.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figure 2.4.1: Running the ARORA Server

To score the second metric, time investment, consideration was put towards both the setup time and the time required in working with the server. Setting up and starting the server is very quick with the aid of the client's documentation, and while developers need to learn the command line interface and the web interface, it is more an issue of practice than of actual problematic levels of time investment. The more straightforward functionalities worked well, so once developers are familiar with the server integration testing should be quick. Therefore, since the major time investment required is learning how the server code works, testing on a developer-controlled environment will receive a time investment score of 4, Low investment.

Finally, sensibility. Sources say that continuous integration testing is the generally preferred method if at all possible [14], [16]. It enables a repeated-prototyping style of development and reduces the chances of long-term errors sneaking in. Since integration is a critical feature in this project, integration testing should be an integral part of the development cycle. Developers also have access to the other parts of the ARORA project code, as is necessary to use this type of integration testing. Therefore, testing on a developer-controlled environment will receive a score of 5, Very High sensibility.

Integration Testing Using a Client-Controlled Environment

As integration testing with a client-controlled setup is similar to using a developer-controlled setup, this section will mainly highlight the differences between the two. In terms of complexity, having the client run setup and handle the server-side of things reduces complexity on the developers' end. Therefore this candidate receives a score of 5, Very Low complexity. However, the only advantage the client has over the development team in setting up a testing environment is having done it before, so the gain is relatively small.

Time investment is the metric where the most difference is apparent. Time working with the server would be about the same either way, so this candidate has the same baseline score as the last. However, working with the client to create a test environment introduces a lot of overhead. Back and forth communication is required, and the simplest means of doing the integration testing this way would likely require a dedicated meeting or at least a teleconference, introducing a coordination aspect as well. Using a client-controlled environment, therefore, receives a score of 2, High time investment.

The reasoning for the client-controlled testing environment sensibility score is the same as for the developer-controlled environment. Continuous integration testing is the generally preferred method. It seems best suited to the details of the project, and it meets the need for integration testing while reducing the chance of compounding errors. Developers have access to the rest of the ARORA system code, and so can implement continuous integration testing. Therefore, the client-controlled testing environment scores a 5, Very High sensibility.

Table 2.4.1: System Integration Analysis Results

	Integration testing post-development	Integration testing with developer's testing environment	Integration testing with client's testing environment
Complexity	3/5	4/5	5/5
Time Investment	3/5	4/5	2/5
Sensibility	1/5	5/5	5/5

Chosen Approach

Based on the results of the above analysis, integration testing with a developer-controlled version of the ARORA system is the best option. Integration testing at the end of development is not suited to the details of the project and thus has a poor sensibility score, as well as a high amount of risk in terms of problems that could come to light later in development when they are costly to fix. Of the remaining two options, integration testing with a developer-controlled setup works the best because it scores much better on time investment. The potential complexity reduction through a client-controlled testing environment is not much of a factor as it relies on a knowledge gap that can be quickly closed. Removing the overhead of relying on the client saves a lot of time, as developers can run their setup on demand due to the client providing access to the code. Developers can still communicate with the client if any issues come up, with less overhead than having to do so for every integration test, which makes testing in a developer-controlled environment the clear choice.

Proving Feasibility

Further proof of this concept will be created during the development cycle. Running quick integration tests on all developed mockups and demos ensures that new

code remains able to be integrated and that the chosen methodology does not pose any problems. These tests will serve as both an implementation of the methodology of continuous integration testing as well as a demonstration that the methodology is indeed performing appropriately for the needs of the project. In addition, it allows developers to learn the interface of the server both on the command line and web ends step by step as needed, which keeps time and effort costs low.

3. Technology Integration

With the solutions to the main technological challenges determined, the places that they fit into the project can now be outlined. The four challenges represent parts of the application architecture that enable the app to meet product requirements. Framework, framework platform, and the database all contribute collaboratively to enabling the core functionality of our app, while system integration ensures that the app can be used within the broader ARORA system. Figure 3.1 below illustrates, on the abstract level, where our four challenges sit within our project.

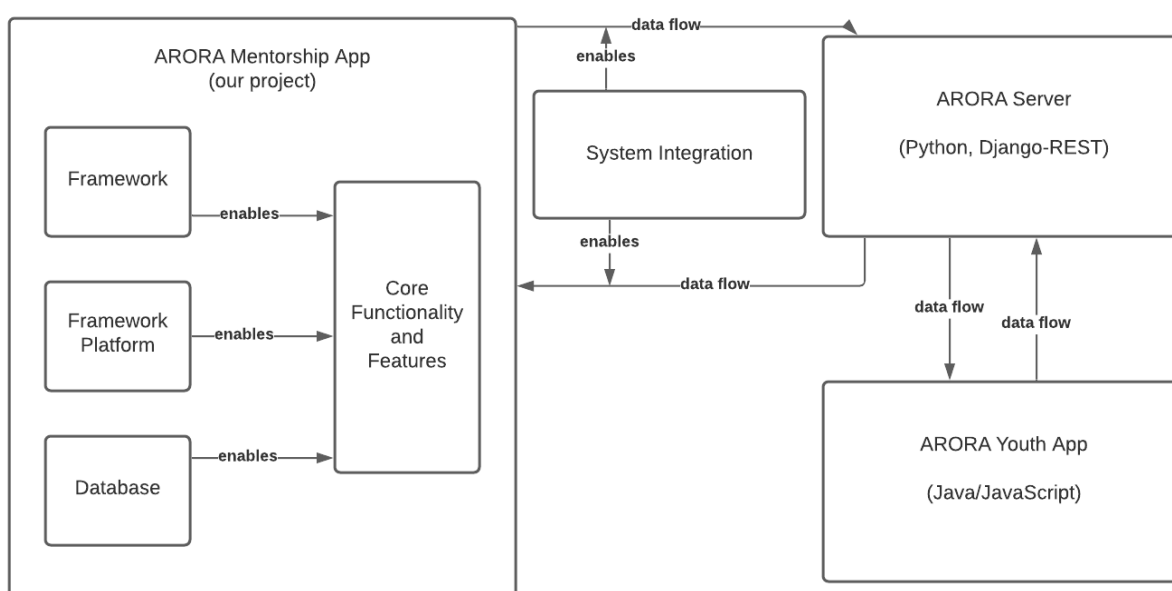


Figure 3.1: Abstract Project Diagram

Within the ARORA mentorship app, three of the challenges above contribute directly towards core functionality, and the fourth facilitates the app's connection with the rest of ARORA. Within the mentorship app is the chosen development framework React Native, the framework platform, Ionic, and the chosen database system MySQL. React Native and MySQL enable the implementation of our key features. These include processing anonymous questions from users of the youth app as well as facilitating mentor-mentee connections for users who are not anonymous. Ionic allows developers to build an interface for the mentors to use when interacting with our backend, which is

necessary since there is no guarantee that mentors will use the app without a graphic user interface, or GUI. The chosen method of system integration is the continuous integration methodology which facilitates data flow between our mentorship app and the ARORA server. Checking that the chosen solution is usable with ARORA as a whole ensures that the app can receive user information and anonymous questions and send back the mentor's responses. Any additions to the ARORA project must be compatible with the rest of the project or they will not be useful.

The team also checked the compatibility between chosen solutions. The framework and platform solutions mesh well through Ionic React, a version of Ionic that runs more seamlessly on the React framework. MySQL is a compatible backend for both React Native and Ionic due to its general widespread support. Passing compatibility checks show that the chosen options are both effective and efficient for solving development concerns.

4. Conclusion

In conclusion, the ARORA mentor app will be a mobile app (for both Android and iOS) that implements the desired features to support the mentorship program, integrates well with the existing ARORA ecosystem and ARORA server. These features are needed and used as metrics to conclusively narrow down candidate solutions. Through research, demoing, and comparisons, the framework, UI platform, database, and integration methodology that are best for this project were conclusively chosen. The chosen framework is React Native, the chosen framework platform is Ionic, the chosen database system is MySQL, and the chosen method of integration is continuous integration testing. These candidates were chosen through the thorough application of metrics that judge their suitability for the project. Further and more extensive demoing will be done on these candidates to demonstrate that they can work in harmony and achieve the design and implementation goals of the ARORA mentor app as development progresses, but it is anticipated that the choices will remain the same.

5. References

- [1] US GAO. (n.d). Tribal and Native American Issues. Retrieved October 30, 2021, from <https://www.gao.gov/tribal-and-native-american-issues>
- [2] Youth.gov. (n.d). Physical and Mental Health. Retrieved October 30, 2021, from <https://youth.gov/youth-topics/american-indian-alaska-native-youth/physical-mental-health>
- [3] Mental Health America. (n.d). Native And Indigenous Communities And Mental Health. Retrieved October 30, 2021, from <https://www.mhanational.org/issues/native-and-indigenous-communities-and-mental-health>
- [4] Back4App. (2021). The Best Flutter Alternatives. Retrieved October 20, 2021, from <https://blog.back4app.com/flutter-alternatives/>
- [5] Liu, S. (2021, August 5). *Most used languages among software developers globally 2021*. Statista. Retrieved October 30, 2021, from <https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/>
- [6] Ionic. (n.d.). About. Retrieved October 20, 2021, from <https://ionic.io/about>
- [7] Combinator. (n.d). Expo. Retrieved October 20, 2021, from <https://www.ycombinator.com/companies/expo>
- [8] FlutterFlow. (n.d.). Features. Retrieved October 20, 2021, from <https://flutterflow.io/features.html>
- [9] Microsoft. (n.d). Xamarin. Retrieved October 20, 2021, from <https://dotnet.microsoft.com/apps/xamarin>
<https://ionicframework.com/docs>
- [10] Stream. (n.d.). Chat Messaging. Retrieved October 20, 2021, from

<https://getstream.io/chat/>

- [11] MySQL. (n.d.). MYSQL.COM. Retrieved October 20, 2021, from <https://www.mysql.com/>
- [12] MariaDB. (2009). MariaDB Server. Retrieved October 20, 2021, from <https://mariadb.org/>
- [13] SQLite. (n.d.). SQLite. Retrieved October 20, 2021, from <https://www.sqlite.org/index.html>
- [14] DZone. (2019). Integration Testing. Retrieved October 18, 2021, from <https://dzone.com/articles/integration-testing-what-it-is-and-how-to-do-it-ri>
- [15] QA Test Lab. (2018). Big Bang Testing Specifics. Retrieved October 18, 2021, from <https://qatestlab.com/resources/knowledge-center/big-bang-testing/>
- [16] Atlassian. (n.d). What is Continuous Integration?. Retrieved October 18, 2021 from <https://www.atlassian.com/continuous-delivery/continuous-integration>