⋐ Technological Feasibility ⋑

October 4, 2023

DENDRO-DAWGZ

|Growing A Brighter Future|

- Team Members -

Nile Roth

Niklas Kariniemi

Zachariah Derrick

Asa Henry

- Sponsored by -

Prof. Andrew Richardson, SICCS/ECOSS

Prof. Mariah Carbone, ECOSS

Prof. George Kock, ECOSS

Austin Simonpietri, ECOSS

- Team Mentor -

Italo Santos

Table of Contents

1 Introduction

Dendrology is the study of trees, and it continues to be one of the most overlooked sciences. With the current rates of deforestation causing concern around the world, dendrology studies are more essential now than ever. Millions of acres of forest land are torn down every year for the establishment of businesses, farms, housing, and sometimes just for their natural resources. Tree physiology and ecology play a big role in providing the population with the understanding of tree's significant involvement in the balance of life. Our project focuses on easing and supporting research projects like these. More specifically, we are alleviating the processes of installing and obtaining data from dendrometers.

A dendrometer is a data-logging instrument that measures the changes in diameter growth over time. They are generally used on the trunks of trees, but can be used on many different types of plants and expanding objects. The data retrieval process from these devices in trees is very laborious due to the fact that the data is transferred through a short cable that connects to USB. This implies that a laptop must be transported up the tree in order to successfully obtain the data. This perilous process is not only physically demanding, but also puts costly equipment in jeopardy. Portability is the essential feature that this system lacks. To assess the issue we plan to implement a mobile application with an easy to navigate user interface. Users of this application can use their android phones or tablets to easily obtain and observe the dendrometer data. This application also allows for the analysis of multiple dendrometers' output simultaneously.

The simplification of this process will not only ease the lives of those who retrieve the dendrometer data, but will also help improve and increase the flow of data needed for research projects. Many research projects at Northern Arizona University (NAU) and around the globe depend on frequent data retrievals from these dendrometers, meaning the process should be simple and efficient. When we make this task less of a hassle, there will be an increase in visits to collect dendrometer data, overall leading to a larger existence of research relevant statistics.

In the current stage of this project, we are processing and analyzing the key technological challenges we may face in the future. Identifying the promising solutions for these roadblocks is also an essential part of this preparation stage. In this Technological Feasibility Analysis document, we begin by analyzing the major technological challenges we expect to hinder us in the implementation stage of this project. Each of these roadblocks are deeply analyzed and alternatives are evaluated in order to come up with the most feasible solution. Lastly, we will compile our analyses and findings into a structured system diagram. This diagram will walk through the system by connecting features and proving our project efficiently feasible.

2 Technological Challenges

The first challenge that we must evaluate is the communication with the dendrometer's embedded system, and reading in data from the logger. The communication needs to be possible on Android devices, and needs a language capable of interacting with low-level hardware directly. It is also required to have a supported library to make communication more straightforward.

Another challenge we must analyze is the development of a mobile application, in whatever form is most feasible. We want the languages and libraries that we use to be able to create an app capable of displaying all the necessary functions. It is also crucial for them to have a substantial amount of support and extensions that make developing the application easier and more efficient.

A software to read and analyze the dendrometer data already exists, but is very undesirable. The issue with this current software is that in order to share any data, a user would have to download the data, then upload it to Google Drive. When another user wants to view this data, they would have to redownload it from the drive and therefore require the necessary user permissions. This process is obviously very inefficient and requires attention and modification. In order to implement this feature, we must answer the following questions. How do we want to share and store the data? What cloud export service would be the easiest and most efficient in doing so?

The dendrometer currently uploads data into files created and stored on our USB device (laptop). A large bunch of files is very undesirable when it comes to data management. Currently, the graphical presentation of the output is limited to a single dendrometer reading on a time/growth plane. These graphs can hold extremely large amounts of data, and therefore can be very difficult to navigate. If one wants to view a certain time frame from this graph, they must use a sub-par manual zoom-in feature. This is why statistical analysis is a technological challenge that must be considered. It is essential to find the most supportive graphing/plotting library for our front end language.

Challenges to Address

Reading in data from the dendrometer
Creating a frontend for a mobile application
Exporting and sharing data using the cloud
Computing and displaying statistical analyses

3 Technology Analysis

3.1 Communicating with embedded system from a mobile device

The most critical component of this product is the communication with the dendrometer's embedded system from a mobile device. This is because each component of the final product relies on the data collected from the datalogger. Similarly, the reasoning behind creating the envisioned product is the elimination of several pain points related to lack of portability and unreliable platform support, which can only be solved if this component is feasible.

After research into the dendrometer's embedded system, we have discovered that they use Future Technology Devices International Limited (FTDI) chips, which are commonly used for serial communication. These chips are located inside the TOMST TMD adapter. When a user connects their computer to the adapter (already connected to the dendrometer), the data is read from the datalogger and converted to the necessary format before being transmitted to the computer. Our final mobile application will need to communicate with this FTDI chip in order to receive the data from the dendrometer, and will therefore need a strong backend with a dependable language.

The TMD adapters we will be working with are not a part of Apple's MFI program - meaning that Apple phones are not capable of communicating with the device. Joining the MFI program is a very involved process, and TOMST has made it clear they do not plan to join the program within the foreseeable future. This means that developing an iOS application is not feasible, so we will from here on only be considering Android as a mobile platform.

3.1.1 | Desired characteristics

- The language we choose to use for the backend must be able to communicate with hardware directly, and therefore must be a **very capable low-level language**.
- This chosen language must also have **reliable, detailed, and well-tested libraries** available to send and receive commands and data. This transition must occur directly from the FTDI chip present in the TMD adapter.
- This backend should also have a **strong community presence**. This generally correlates with a decent amount of resources at our disposal for when we encounter issues.
- Finally, our **team needs to be familiar with the language** and able to jump into implementation with as little workshopping, learning, and research time as possible. This will allow us to develop stronger code in less time.

3.1.2 | Alternatives

The alternatives for actualizing the communication with the dendrometer's embedded system are limited by the available libraries for the embedded chip. We will face significantly less limitations in our backend language choice. The application requires a backend language to handle connecting and communicating with the FTDI chip using a USB device. We will now present a brief description of three potential backend language options.

C: C is the gold-standard low-level language which has been implemented by countless organizations and applications for similar hardware communication purposes. C and its supersets have been the top language choice for embedded systems development since its initialization in Bell Labs between 1972 and 1973. This makes it one of the most documented and tested contenders available for our specific use case. From its inception, this language has been utilized in diverse domains spanning from operating systems development for Unix to applications in embedded systems. C is also great for aerospace and avionics, military and defense, robotics and home appliances, and countless more. Extensions and supersets like C++, C#, and Objective-C have been developed over the years to add additional functionality and better support for specific applications, and are generally adopted over plain C within each domain.

Rust: Rust is an upcoming low-level systems programming language which offers the same access to the system as C. However, Rust additionally provides very useful features including borrow checking, which is used to prevent a program from crashing due to common problems like reaching into unallocated memory.

Rust was developed by Graydon Hoare in 2006 and then acquired by Mozilla's sponsorship in 2009. Rust is a response to the problems encountered and caused by C and C++, focusing on features such as performance and type safety while also alleviating the stress of problems like memory management. Rust has been quickly adopted for game engines and other performance and memory critical applications. It has gained C levels of notoriety since it was integrated into the Linux kernel. Rust has also been Stack Overflow's favorite programming language since 2015.

Java: Java is a prominent high-level language that emerged as an alternative to C for cross-platform software development, amongst other uses. Java was created by James Gosling at Sun Microsystems during the 90's, and has quickly gained popularity due to its portability and platform independence. Since then, it has become the language of choice for top companies and applications due to its extensive libraries and "write once, run anywhere" philosophy. Java is typically used for Android app development, server applications, web development, and numerous other common applications, making it a viable option for our intended system's backend language.

3.1.3 | Analysis

We will analyze the viability of each option based on four factors: low-level communication ability, usability of available libraries, community support and presence, and the team's experience level with each option. After the analysis we will use these factors described above to determine the final backend language decision.

C and its supersets:

- **Low-Level Development:** As mentioned before, C is a widely known and extensively used language for low-level systems programming. C allows for extreme levels of access to kernel level functions, which is necessary for embedded systems development, and therefore a good fit for our application. We will need to communicate directly with USB hardware and the FTDI chip. It is also important that we have the ability to develop functions on the driver level, in which C can provide. Because C has so much functionality and access to low-level systems, it is extremely easy to write 'unsafe' code that is prone to error. Common issues surrounding memory allocation, undefined behavior, and logical errors are abundant. This entails that if we choose this language, special care should be taken to ensure that we address all potential issues. Luckily, there are numerous libraries and tools available to evaluate our code for these problems, like Valgrind, sanitizers, and static analyzers.
- **Library Strength:** Additionally, the libraries available in C for communication with FTDI chips are strong and well documented. The *libftdi* library for C was developed and publicly released in 2006, and is continually improved by open-source developers looking to add more functionality. This library will give us a straightforward option for developing the chip communication, without having to create our own solutions with existing USB libraries.
- **Community Support:** Since C is widely used across several domains - particularly embedded systems - there is an extensive community of developers and questions at our disposal. It is likely any issues we encounter can be resolved by an already existing question, article, or tutorial. This makes C a strong choice when considering the community support.
- **Team Experience:** The team has considerable experience with the C language, as NAU uses C as the preferred language in all applicable courses. This means that each member of our team should be able to immediately jump into development without spending time learning or workshopping. Additionally, this means that our team will write stronger backend code, due to the fact that all of us can understand the code and quickly debug, raise issues, or contribute new ideas.

Rust:

- **Low-Level Development:** Rust is a newer language built to rival C and solve several of its most common issues. Because of this, Rust is a strong low-level language, capable of interacting with driver and operating system functions at the same level as C. This means that in terms of low-level development, Rust is well suited for our use case and would be a strong option for an embedded system backend language. Rust alleviates several of the issues mentioned in the previous section, like memory allocation errors, undefined behavior, and logical issues with several novel features. It is able to combat these by including memory ownership, borrow checking, type-checking, null-safety, and garbage disposal. Because of these features, it would be easier to write safe code, which would be a real advantage when working on the hardware level.

- **Library Strength:** Due to the nature of Rust and its intended uses, any library available in C is available in Rust. This means that the same *libftdi* library mentioned in the previous section is compatible with this language, making it a similarly strong choice.
- **Community Support:** Because Rust is a relatively newer development, and has only recently begun being adopted by larger organizations and applications, the community is still in development as well. By competing against the monolith that is the C language, developers generally decide to write their applications in C instead of Rust despite the numerous advantages the language has. Consequently, this means that fewer codebases and examples exist currently. Demos, tutorials, and other learning resources are certainly available with Rust, and there is generally stronger support from the community despite their smaller numbers. Overall the community is strong, but the existing examples and codebases are more limited than C's expansive collection of resources.
- **Team Experience:** Our team is for the most part unfamiliar with the Rust programming language. Due to its novel nature, Rust's key advantages and features are all things that would require more learning, researching, and workshopping for us to all grasp. Our team would need to spend time learning the language before we could begin implementing the code, which would certainly slow down our development process.

Java:

- **Low-Level Development:** Java was designed as a high-level language more suitable for things like web development, servers, Android applications, and UI frameworks. As such, Java is considerably less suited for low-level device communication when compared to C and Rust, making it a less optimal option in this regard. However, Java still does have the ability to communicate with drivers and operating systems through its established libraries. It is not typically recommended to do this, and therefore is generally not chosen for low-level development.
- **Library Strength:** There is a library for FTDI chip communication available for Java named *FTD2xxj* that would allow our code to interact with the device. This library requires wrappers however, and the wrappers available are fairly undocumented and only tested on Windows and MacOSX. This means we would be operating using unclear software with minimal description of functionality. It is entirely possible that we might

have to implement our own functions based on already established Java USB libraries to work around the untested and non-agnostic nature of this library.

- **Community Support:** Java is widely-known, and has a very strong community of developers, tutorials, codebases, and questions. It would be easy to get help along the way with programming related questions, as they either already exist, or would be answered quickly. The general documentation of the Java language is very thorough, and would provide a solid community and foundation for us to develop in.
- **Team Experience:** Our team is moderately versed in this language. Several of the team members have developed projects or worked in industry using this language, and would therefore provide a decent starting point for our team. Similarly, several of the first few courses we completed at NAU were done in Java, before the university rolled over to C, giving us additional experience with this language.

3.1.4 | Chosen Approach

We will now decide a point value out of 10 for each language in each category analyzed in the previous section. We will then calculate the average score (total score) for each language, which will be used to determine the final backend language decision.

Table 1 | Backend Language Scores

Language	**Low-Level Development**	**Library Strength**	**Community Support**	**Team Experience**	**Total Score**
C and its supersets	9	7	9	8	8.25
Rust	10	7	8	3	7
Java	4	3	9	5	5.25

Based on our analysis of these options we have decided to use C as our backend language. This is due to the ratings and and total score based on the factors outlined in the previous section and provided by Table 1.

3.1.5 | Proving Feasibility

In order to properly prove that C is feasible for communicating with the embedded system, we will need to develop a demo/prototype where we are properly reading in data from the data logger using the C language. This demo will showcase that we are capable of interacting with the dendrometer and getting legitimate data from the device, resulting in a significant leg up in our eventual implementation process.

3.2 Frontend

This software is meant to alleviate a safety concern, as well as improve practicability of the existing software, and is most compatible with widely accepted Android devices. Therefore we must consider a capable frontend that we are experienced in to develop the app.

3.2.1 | Desired characteristics

- The software should be **run on Android** – having our software be run on a platform that is very common and easily acceptable is a key component of our application.
- The software must have **low-level access** – one piece of our software's core use is to collect data from devices which will be connected via USB; in addition, the manufacturer utilized a FTDI chip to facilitate communication and data transfer.
- The software must provide the user with a **graphical interface** from which they can visualize, analyze, compress, and write data to disk.

3.2.2 | Alternatives

Java: Java was originally developed by James Gosling ("the Father of Java") at Sun Microsystems Inc. in 1995, and was later acquired by Oracle Corporation. Many corporations, hobbyists, game developers, etc. have used Java to write small to enterprise ready applications. Java affords the developer with performant and platform agnostic code by utilizing a virtual machine called the "Java Virtual Machine (JVM)". The language has been the standard for Android development since the birth of the operating system.

React Native: React Native is a framework, undergirded by JavaScript, the team found when investigating solutions which could be adapted across mobile platforms. This framework was created by Meta Inc. and initially released in 2015. In contrast to its predecessor React, Meta intended React Native to offer a unified way to write a frontend for mobile devices while allowing for native UI calls.

Kotlin Multiplatform: Kotlin is a general use programming language developed by JetBrains. The language is built to operate with Java, so Kotlin has seen widespread adoption by Android developers, eventually becoming announced by Google as one of the "official" languages for Android development in 2017. A useful feature of Kotlin is the language's cross-platform capabilities officially understood as "Kotlin Multiplatform".

Flutter: Flutter is a framework for building cross-platform applications which was developed by Google and launched in 2017. The framework is built on top of the Dart language which was also developed by Google. Flutter was designed with the intent to write code for iOS and Android, as well as websites, with the advantage of being able to write code once and deploy across platforms.

3.2.3 | Analysis

Java:

- **Familiarity:** Java's established reputation made it a cornerstone of NAU Computer Science curriculum for years. Because of this, most capstone students in the Computer Science department this year have been taught the Object Oriented paradigm and Data Structures in Java.
- **Libraries:** Because Java has been around for about 3 decades now, the language has been used to build notable software such as the Android operating system. Therefore, the team is guaranteed to have a comprehensive collection of packages to lean upon to create a well-formed and intuitive mobile user interface.
- **Community:** Java version 1.0 was released in 1996, and since then has become a language of choice. This version has spawned a market for Java tutorials and books to create generations of Java developers and a thriving community.
- **Performance:** Compared to other languages or architectures which have the express purpose of creating user interfaces, Java stands with its predecessors as fast and reliable. In addition, Java was also built with the ability to use Just In Time compilation (JIT) to support and help the compilation process. This aids the overall development cycle by making it even shorter because compile times can be reduced without sacrificing performance to an interpreter. Not to mention, building a native Android application has benefits when traversing the bridge between frontend code to native system call since the aforementioned bridge is non-existent.

Kotlin:

- **Familiarity:** Kotlin was developed as a successor to Java, thus requiring interoperability and ease of translation. Kotlin maintains a syntax and structure similar to the C programming language. This elevates Kotlin since team members will have to spend little to no time learning the syntax and structure of a new language, and can immediately begin to construct a frontend.
- **Libraries:** Kotlin has already been used to build notable cross-platform applications such as Slack, Pinterest, and Netflix. This means there is a high likelihood any support for specific features of our application will already be available, and if not there will be significant support from third-party libraries the team could use to build a solution. In addition, Kotlin's ability to interact with Java means the team has the entire collection of Java libraries to create our application.
- **Community:** The Kotlin community is expansive and is supported by developers, dedicated to improving the language. In addition, these developers are easily accessible through podcasts, forums, and even on Twitter. Therefore, even if the language is still up and coming, help is very much easy to find and easy to access.
- **Performance:** Kotlin was designed specifically to be a "better language than Java" in which JetBrains could work with, while maintaining full compatibility with Java. In

addition, since Android Studios is the intended IDE for Kotlin development, the ability to see and test the user interface on the fly already exists and is well supported. Therefore development will not be bogged down, and the team can spend more time making the needed improvements to the frontend and seeing the changes made reflected in the updated emulation. However, using Kotlin as a cross-platform solution has already gained criticism for the same reason React Native has, which is the fact that bridging the gap to make rendering calls is still quite expensive.

React Native:

- **Familiarity:** JavaScript has been the language of choice for websites, web applications for many years after its creation in 1995. Because of React Native, the richness of the JavaScript community and libraries has been afforded to mobile developers. Using this framework is also a huge advantage to the team since all NAU Computer Science students are required to take a web development class; therefore, each member is familiar with the JavaScript paradigms.
- **Libraries:** Though this framework was developed and maintained by Meta, the support available to the team extends beyond official packages to the third party packages. This means there is a high likelihood the team will be able to find something to help solve most problems we run into during development. For example, when a developer installs React Native an accompanying emulation tool is installed which allows the developer to see and test their application as it is developed.
- **Community:** Given that JavaScript is the earliest and most used language for web development, React Native intrinsically has a burgeoning community of skilled developers. This is also partly evident in the amount of third party packages available to developers. A few applications which demonstrate the breadth of support that React Native's community provides include companies such as Discord, Instagram, and UberEats which have all successfully built performant applications for iOS and Android.
- **Performance:** The framework, however, does suffer from performance issues, specifically how native calls are handled impacts an application's performance. In part this is due to the fact rendering requires system calls which React Native must lean on some hoops to make this possible.

Flutter:

- **Familiarity:** Flutter being built on Google's language Dart, which was released in 2013, poses a small threat considering the team is not familiar with the Dart language, nor has anyone on the team used Flutter. Examining the structure of the Dart language, however, it is clear Google had a goal to create a language with the widest appeal by maintaining a C-like syntax while also being object oriented. This benefits the team considering little time will be needed to learn a new syntax and more time can be spent designing and writing code to get a functional application within our given time frame.

- **Libraries:** Given that Flutter is an open source solution, this provides the team a multitude of libraries - from industry to hobby work - which we can lean upon to implement the needed functionality for our application. Not to mention, Google has outfitted developers with project critical libraries such as dart:ffi which is one library the team has to use to communicate with our backend, written in C.
- **Community:** Considering Flutter's reception by people in computing spaces has kept up with other newcomers like Kotlin, the framework would be an understandable choice for frontend development due to the number of libraries and hubs that the team can rely upon to solve issues we face. However, the team should remain wary of Flutter's new arrival given Udemy only maintains about 500 Flutter tutorials, while Kotlin has double the number of tutorials. This entails that our team may be bargaining for time to create a new solution for some problem.
- **Performance:** It has been noted that Flutter ends up making applications greater in size. For cross-platform solutions, such as React Native, this is the norm. Going beyond, Flutter offers many benefits including "hot reloading" – something React Native offers, as well – which can greatly improve the time spent developing and not waiting for changes to be reflected in an emulator.

3.2.4 | Chosen Approach

We will now decide a point value out of 10 for each frontend language and category analyzed in the previous section, and calculate the average score (total score) for each service. This score will be used to determine which frontend language we use.

Table 2 | Frontend Framework Scores

Language/ Framework	Familiarity	Libraries	Community	Performance	Total Score
Java	9	9	9	9	9
React Native	9	9	9	7	8.25
Kotlin	8	10	7	8	9
Flutter	8	6	6	7	6.75

Using the scores from Table 2, we have decided to use Kotlin to build our frontend. Choosing Kotlin makes most sense from a perspective of longevity because Google has named Kotlin the preferred language for Android development. Kotlin is also well supported since the developer still has full access to Java libraries, as well as Kotlin specific libraries. This also helps avoid introducing any more complexity to the project given all of the team members will be familiar with Kotlin's structure and syntax given our experience with Java in and outside of the classroom.

3.2.5 | Proving Feasibility

To prove the feasibility of Kotlin for this project, the team will follow a few tutorials to get a mock application working. The team will take the opportunity to extend the mock application and begin designing the interface for data visualization and manipulation. This will provide us an opportunity to communicate with the client and start getting feedback on how they want the user interface to look and feel. Building up a mock app in this way will yield a codebase we can reference and pull from as we build the actual application, and give ourselves a head start to leave more time for other more laborious tasks.

3.3 Cloud Export

With their current software, our clients have to manually download all the data from the software that they want to share. They then would have to upload that data to a service like

Google Drive. This is a very time consuming task, that could be automated to make it much faster and easier for everyone involved. The purpose of a cloud export service would be to make sharing data and collaboration much easier and more efficient. Ideally, a user would be able to grab the data from the dendrometer, and have the software automatically push this data to the cloud. This way the data is stored on the cloud and not on some storage on a device. This adds the ability for any user that has access to the data to look at it from anywhere. This makes the whole collaboration process much easier to handle.

3.3.1 | Desired characteristics

There are a good amount of desired characteristics that we want for our cloud export service. These characteristics would make having a cloud export much easier and more efficient for everyone involved.

- We want the cloud service to be **easily integrated within a mobile application**. Along with this, it should have high readability and understandability. We also prefer a service that is familiar within the group.
- The cloud export service should be able to **store any kind of data**. Having the options between NoSQL and SQL databases would be a plus. The service should also be very efficient and include helpful features.
- The service should have the ability to **share data with other users**. We don't want a sharing system that is confusing and hard to use. We desire something that works for both developer and user. Having user authentication features would make sharing simpler.
- We want a service that is **cost efficient**. Something free would be ideal, but having to pay is not out of the question.

3.3.2 | Alternatives

Amazon Web Service(AWS): AWS is the largest cloud platform with over 1/3 of the market share. It is the most popular cloud platform among both developers and clients. Amazon started the idea of AWS back in 2003 when they realized that they were proficient at running scalable and reliable data centers. For this reason they decided to look into providing this service to other developers. This led to the creation of AWS and its launch in 2006. Companies will use AWS to host a myriad of software applications. Some of these companies include the BMW Group, Coca-Cola, Epic Games, and even Netflix. AWS has proved over the years that they can power innovation across many industries.

Microsoft Azure: Azure comes in as the second largest cloud platform with 1/5 of the market share. It was released in 2010 as a competitor to AWS by Microsoft. Azure has over 200 products and services that allow companies to build software applications on the cloud. Azure has three main areas it focuses on, including Infrastructure as a Service(IaaS), Platform as a service(PaaS), and Software as a service(SaaS). Over 95 percent of Fortune 500 companies rely on Microsoft Azure. Some of those companies include Verizon, LG Electronics, and CenturyLink.

Firebase: Firebase is a development application geared towards mobile applications. It helps developers handle backend services so they can focus on other application experiences. Some of these services include authentication, databases, and file storage. These services are run through the cloud and have proven to be very scalable. Firebase was launched in 2012 by Andrew Lee and James Tamplin. It was later acquired by Google in 2014 and has since been managed by them. Firebase is used all across the globe by companies such as Accenture, Alibaba, and Lyft.

3.3.3 | Analysis

For each of the services listed in section 3.3.2, we evaluated them based on the criteria of ease-of-use, data storage, sharing capability, and cost.

Amazon Web Service(AWS):

- **Easy-Of-Use:** Connecting AWS to mobile applications is fairly easy. With Kotlin you can use libraries and services like AWS Amplify and AWS AppSync. AWS Amplify is a mobile application library that uses the cloud service AWS. AWS AppSync is a fully managed GraphQL service that offers offline and real-time features. To connect a Kotlin application to AWS, Amplify is the best way to go. Amplify allows developers to easily use services like authentication, APIs, and storage. To use AWS Amplify, all you would have to do is pull in whatever category you want to use to the application. For example, if you wanted to use authentication from AWS, you would pull in Auth, which would then give you all the authentication features from AWS. Both AWS Amplify and AWS AppSync were developed by Amazon so they can be highly trusted.
- **Data Storage:** AWS offers the Simple Storage Service(S3). S3 is an industry-leading cloud storage service. It is highly scalable and has high-level security. Applications and projects of any size will be able to use S3 as a cloud storage service. Amazon S3 offers storage classes that are designed for different use cases. S3 also offers storage management features that help manage cost and reduce latency. With Amazon S3 you can restrict access to buckets and objects, adding a level of security. S3 lets developers have logging and monitoring to see what resources are being used. AWS also offers a NoSQL database service called DynamoDB. This can be used as an alternative to S3 if we decide to go down the database route.
- **Sharing Capability:** With Amazon S3 you can create access points for groups of users. This allows developers to easily handle what users have access to the data stored in S3. AWS also has a feature called Identity and Access Management(IAM). IAM lets developers control the access to resources stored using AWS. With IAM you can control what actions users can do on the data. Actions include things like reading, writing, or deleting.

- **Cost:** AWS offers a free tier for all of their products that is valid for 12 months. For Amazon S3 you can get 5 GB storage for free. For Amazon DynamoDB you can get 25GB of free storage. Outside the free tier, everything is pay-as-you-go. So you only pay for what you use within AWS. For standard S3 storage you would pay around $0.023 per GB that you use in that month.

Microsoft Azure:

- **Easy-Of-Use:** To use Azure with Kotlin you would need to use the Azure CLI, which is a cross-platform command line tool that allows developers to use Azure resources. It is recommended that developers use the Azure Cloud Shell, however developers can install it on Windows or MacOS. If developers are looking to use Windows or MacOS they should consider using Docker for the CLI. With the CLI, a developer gets access to a lot of tools that make it easy to use Azure with a mobile application.
- **Data Storage:** Microsoft offers the platform Azure Storage, which can be used to store a variety of data types. All the data stored in Azure Storage is encrypted and it is very scalable. It is designed to handle any data storage size. Microsoft does provide some libraries in a variety of languages like Node.js or Python. Azure Storage offers multiple different services like Azure Files and Azure Tables. With Azure Files you can manage files that are stored on the cloud. Azure Tables allows developers to use a NoSQL database. Microsoft also has a service called Azure SQL Database which operates like a standard SQL database, and would be straightforward to use.
- **Sharing Capability:** Developers can access data using HTTP or HTTPS connections anywhere in the world. With Azure Storage you get fine-grained control over who has access to the data. Microsoft provides a couple of different ways of controlling shared data. One of them is using Shared Access Signatures(SAS), which developers can use to grant access to resources in the storage account. Developers can also specify read, write, and delete permissions a user receives
- **Cost:** Microsoft offers a 12 month free trial to Azure. We also are given the ability to start with a $200 credit for Azure. Some services are always free for Azure customers like Azure SQL Database. Aside from free trial and free services, Microsoft uses the pay-as-go method. Microsoft also does price-matching with AWS services.

Firebase:

- **Ease-Of-Use:** To use Firebase with Kotlin you will need to add the Firebase configuration file to the mobile application. It is recommended to use Android Studio for this. Android Studio makes it really easy to add Firebase to a project. After adding the configuration you need to add the Firebase SDK. Once again, it is recommended to use Android Studio to add the SDK. Using Android Studio overall is better when using Firebase. After adding the configuration and SDK, you can access any of the Firebase resources. This is if you have created a Firebase project in the Firebase console.

- **Data Storage:** Firebase does offer cloud storage that is built on the Google Cloud infrastructure. It is designed to handle any data size and is cost-effective. There also is Firebase Realtime Database that allows developers to store data in a NoSQL cloud database. It remains available at all times even when the app is offline. The data would be stored as JSON and it supports cross-platform use. Google has recently come out with Cloud Firestore which is a new database based on the Realtime Database. Cloud Firestore is built for mobile application development and offers faster queries and better scaling.
- **Sharing Capability:** Firebase allows developers to control who can read, write, and delete files that are stored on Firebase Cloud Storage. Developers can use Firebase Authentication to grant access to specific users. All of this also applies to either Realtime Database or Cloud Firestore. With the databases you can also structure the data in a way to restrict access to certain users.
- **Cost:** Certain features are free like authentication. Firebase also allows the use of 1 GiB of stored data for free on Cloud Firestore and Realtime Database. Cloud Storage is free up to 5 GB of stored data. After the completion of the free trial, Firebase uses pay-as-you-go. For Cloud Firestore and Realtime Database it would be $0.108 per GiB per month. To use Cloud Storage it would be $0.026 per GB that is stored.

3.3.4 | Chosen Approach

We will now decide a point value out of 10 for each cloud export service and category analyzed in the previous section, and calculate the average score (total score) for each service. This score will be used to determine what cloud export service we use. The scoring can be seen in Table 3.

Table 3 | Cloud Export Scores

Cloud Service	**Ease-Of-Use**	**Data Storage**	**Sharing Capability**	**Cost**	**Total Score**
AWS	9	8	8	7.5	8.125
Azure	7	8.5	7.5	7.5	7.625
Firebase	9	8	8	7	8

Based on our analysis of the cloud export service options, we have decided on using AWS as our way of storing and sharing the data from the dendrometer. This is due to the ratings and total score based on the factors outlined in the previous section and the provided Table 3.

3.3.5 | Proving Feasibility

To further test out and investigate AWS, we plan on creating some demos using it as our cloud export service. What a demo would look like in this case is us taking some mock data and trying to store it on AWS. After storing it successfully, we'll test out the sharing functionality of this service. This way we can see how exactly we would use AWS as our cloud export service.

3.4 Statistics and Graph Display

The data retrieved from these dendrometers should be displayed and presented in the most efficient way possible. The presentation of the data is currently limited to a single dendrometer reading on a time/growth graph. This graph can hold extremely large amounts of data, and therefore can be difficult to navigate. If one wants to view a certain time frame from this graph, they must use a sub-par manual zoom-in feature.

The current dendrometer data display is undesirable and lacks essential features. Not only do we want to improve upon these criteria, we also want to add certain computations to our statistics. A report of total, maximum, and average growth per day will save our users the trouble of doing these basic yet time consuming calculations. We also want to supply our users with different time series charts that relate the data in different ways. We plan on creating a first-difference line graph that displays the change in diameter from one day to the next, as well as one that calculates and displays the daily amplitude. Our overall goal is to introduce statistical features and graphs that improve usability and simplicity.

3.4.1 | Desired characteristics

In order to find a feasible solution for this challenge, we must analyze the desired characteristics of the statistics we display in our application:

- We want the ability to **display multiple data sets on the same chart/graph**.
- **Viewing subsections of data** like certain smaller time periods is an ideal characteristic.
- **Extracting a set of data** from the original set, such as a certain window of time, is a characteristic we would like the application to include.
- We want the ability to **interact with charts through clicking/scrolling** mechanisms.
- Most importantly, we want the library to be compatible with **Android**.

3.4.2 | Alternatives

In order to make all these characteristics feasible with our program, we must analyze the provided features of different graphing/charting libraries for Android applications. The libraries in consideration include:

MPAndroidChart : This library is powerful yet lightweight. It provides us with a plethora of charting options, each with many useful features. These charts contain listeners for touch, gesture, and selection callbacks so that they are completely interactable. Zooming is also available for both axes of a graph. Because of the popularity of this library, there is a lot of support on both GitHub and stackoverflow. The availability of video and written tutorials is very prevalent in many sources. This library allows for the application of over thirty different possible animation functions. One downside we observed is that MPAndroid may struggle with supporting super large datasets.
AChartEngine : This library has a significantly smaller community than MPAndroidChart, yet it provides even more charting options. This program is well optimized allowing it to support large numbers of inputted values. This is crucial for our project because of the extensive amounts of data that dendrometers generally deliver. Unfortunately few sources are available for troubleshooting in this library, which is concerning for later stages of this project.

AndroidPlot : AndroidPlot is completely free and open source, and is actively maintained and developed. Consequently, this library is fairly well known and is used by thousands of apps on

Google Play. Support for both large data sets, and for dynamic modeling exists in AndroidPlot. It is also compatible with all versions of Android and works equally with Kotlin and Java codebases. There are some great sources that cover all of AndroidPlot's documentation, which also include many tutorials and examples. This library's downside (in our opinion) is its readability. With a syntax comparable to Java, it is hard to connect lines of code with its effect on a chart.

3.4.3 | Analysis

To determine the most feasible solution we will break down each library into the following sections: chart variety, difficulty (use), and aesthetic. Each library will receive a score from 1- 10 (10 being most desirable) per section. All ratings are decided based on extensive research. The library with the most points will be the one that will contribute most to our project success.

When analyzing the chart variety of a given library, we must consider what types of graphs are offered as well as any features one may include. The data we will be working with will most likely be displayed as a line graph. A stretch goal of ours is to discover different ways to compare and display the data in different chart types that will be useful and beneficial to our clients. In order to support this goal we must consider all offered chart types that may be applicable to our application.

It is important that we use a library that is simple and easy to use. We should be assured that our code maintains high readability when writing library based functions. Because charting libraries in Kotlin and Java are a foreign topic to most of our group, Information and tutorial accessibility should also be prevalent online.

An admirable aesthetic is a crucial factor of any charting library. Because the graphs and charts are the focal point of our user interface, we must ensure that they are pleasing to the eye. The charts created in these libraries should conserve their understandability throughout their customization. We want a charting library that is able to implement appealing yet precise graphs.

MPAndroidChart:

- **Chart Variety:** As stated previously, this library's charting options hold no restraint to our circumstances. supported chart types include line, scatter, bubble, pie, radar, bar, combined, and candlestick. Many of these graphs will not be applicable to our application, but having these options may be beneficial for later stages of our implementation phase.
- **Difficulty of Use:** Because MPAndroidChart is so widely used, there exists a lot of support online specifically in GitHub and stackoverflow. A web page created by *Weekly Coding* [18] supplies us with anything we may need to know regarding this library. From Initializing a graph to creating animations, this resource has it all. It will be very unlikely for us to run into a charting roadblock that cannot be resolved. Lastly, this library has a

very fair syntax with moderately easy readability. For these reasons, this charting library is a very strong competitor in this category.

- **Aesthetic:** This library provides us with predefined color templates, but is also fully customizable meaning the chart's fonts, legends, colors, backgrounds, and lines can all be chosen to best suit our application's user interface. Users are able to interact with the created charts through clicking, zooming, and even hovering.

AChartEngine:

- **Chart Variety:** This library is the top choice for this category. With the ability to implement any graph or chart we can think of, our statistical analysis is not bounded in any way. Many of these charts can be combined onto a single plane in order to compare and display the relationship between two different sets of data.
- **Difficulty of Use:** Troubleshooting and tutorial sources are existent for this library, but prove to be less prevalent than ideal. I have found a couple Stack Overflow posts regarding the issue of incorporating the library repository into a project. This may suggest that the startup process for creating charts may not be as smooth as we hope for. Aside from this, we approve of the function syntax used for this library. Each line of code for creating the chart deems to be very understandable even to unfamiliar users.
- **Aesthetic:** Because this library seems to have a lower level syntax, we are basically building these charts from scratch; manually setting all features associated with it. The benefit of this is that we have full control over how the chart looks and feels. Overall, the display of data using this library is sub-par and contains little animation features.

AndroidPlot:

- **Chart Variety:** The chart variety in this library is also more than sufficient. Bar, line, area, pie, candlestick, bubble and step charts can all be implemented with its use.
- **Difficulty of Use:** After reviewing the code example for creating a simple XY plot, we confirmed that understanding AndroidPlot's syntax will be a trivial task. Fortunately, this library is very well documented supplying us with tutorials, guides, examples, and common bug solutions.
- **Aesthetic:** The charts rendered with AndroidPlot are generally visually pleasing and can be customized significantly. Like the other libraries, colors, styles, labels, and other visual elements are manually set when writing the code.

3.4.4 | Chosen Approach

Table 4 | Charting Library Scores

RN Library	**Chart Variety**	**Difficulty**	**Aesthetic**	**Total Score**
MPAndroidChart	8	8	8	8
AChartEngine	10	7	6	7.667
AndroidPlot	9	6	6	7

The result of table 4 proves that MPAndroidChart is the best library for our project criteria. We are satisfied with this decision because this library has great reviews online and it has been exponentially increasing in popularity. Because of this expansion of use, resources and helpful information will be available from many different sources. This will be very beneficial in the case we hit a roadblock or some sort of bug that requires troubleshooting guidance. MPAndroidChart will provide us with a seamless front end user experience. This library is well known for its high potential of aestheticism and visual appeal. Many effects and animations are provided to give charts an attractive and modern appearance. The graphs created under this library will be clickable and scrollable allowing users to interact with their data. With its functions having high customizability, we will be able to create and adjust chart framework to best suit our application. One concern is ignited with the use of this library. We lack assurance that its framework will support extensively large data sets. Our second option AChartEngine will serve as a great backup. This library has been proven to support data sets of all sizes. The use of AChartEngine will resolve our large data set issue, but will most likely put a dent in the aestheticism of our chart(s). Either way, AChartEngine will be the chosen approach for our project because it holds all features of a charting library that we deemed necessary.

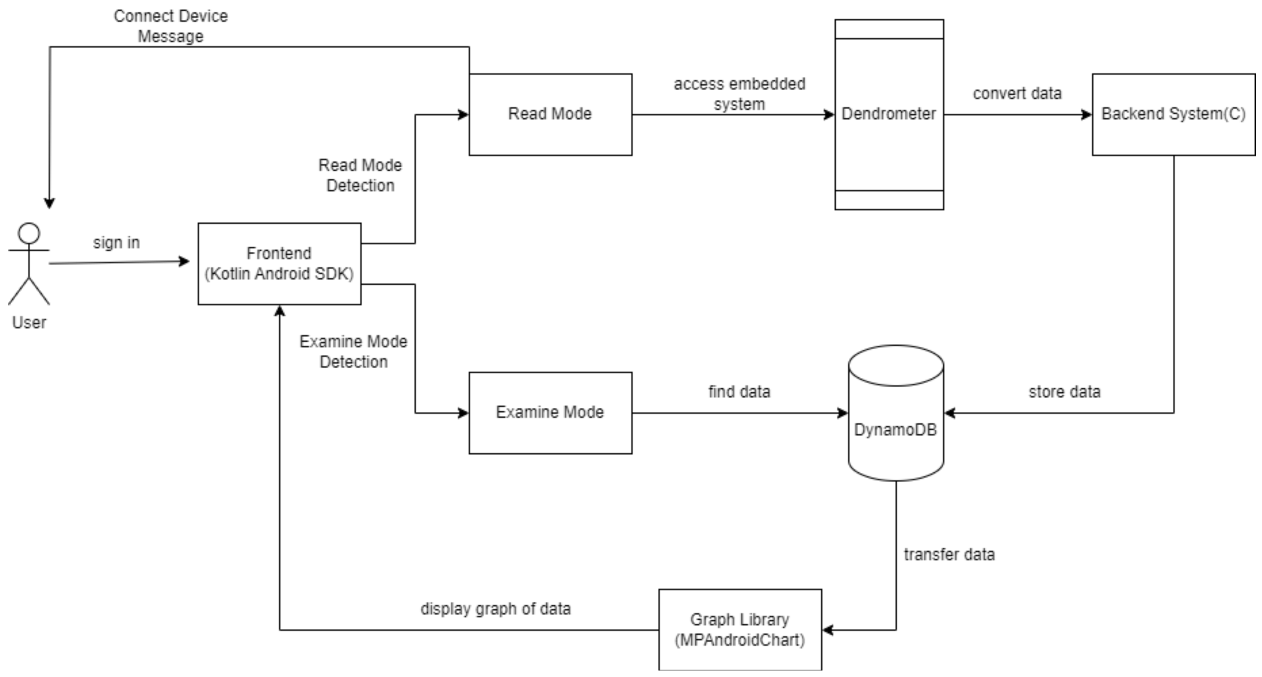
3.4.5 | Proving Feasibility

To prove our final decision as feasible we will create a line graph using the MPAndroidChart library. This will be done using mock data when our Android application is up and running. The mock data set will be significantly large to prove the library's feasibility with extensive amounts of data. In the case MPAndroidChart is proven infeasible, we will perform the same test on the AChartEngine library. We are very confident that this task will be implemented with ease using this library, if not both.

4 Technology Integration

In the previous sections we have analyzed the major technological challenges we expect to encounter as we implement this project, and described our intended solutions. In this section we will put all of these pieces together into a structured graphical representation of the architecture for our expected system seen in Figure 4.1.

Figure 4.1 | Envisioned System Diagram



When a user starts the DendroDoggie application, they will first need to specify whether they intend to download data from a dendrometer, or if they would like to examine a dataset. The graphical user interface will be built using Kotlin and the Android SDK, and it will have the ability to communicate with the backend in the case the user wants to download data. Upon selecting the download operation, the application will attempt to detect a reading device. If this device is not detected, it will ask the user to connect the reading device when the operation is initialized. Upon selecting to examine a dataset, the application will access the database to get the data, and then communicate with the graphing library to display points to the screen.

1. **Reading Data Case:** If the user specifies reading data, then the application will halt until the user connects the device needed to interface with the dendrometer. The application will write to the screen a message prompting the user to connect a device. If the user already has the device connected, a box will appear to display the progress of the download operation. The box will be prefaced with text explaining that the download is in progress. This box will be followed with a box showing the name of files as they are transferred and written to disk. When the download is complete, the box will be filled to indicate all files have been downloaded. The text proceeding will also explain that all files have been read and written to the disk, and it is safe to remove the reading device from the dendrometer.

2. **Examining Data Case:** For the situation in which the user specifies examining a dataset, the application will access the database. Once the application accesses the database, it can present the user with a list of directories and files where datasets have been stored under a

directory allocated for the application. This list will appear as a hierarchy with subdirectories and files justified right of the parent. In addition, a manilla folder icon will indicate a directory (i.e. folder), and a white box with lines will represent a file (i.e. piece of paper with text). The user will select subdirectories to expand or collapse until they select the file they wish to open. At which point, the filename will be presented with a blue background, and a grayed button reading “Open”. The open button will turn to blue to indicate a file has been selected and can be opened. It is at this point the user can make pinching and stretching motions with their finger within the bounding box of the graph to increase or decrease the window of time for which data points can be seen.

a. **Data Manipulation and Analysis:** The user will have the option to run different analyses and save the resulting graphs. This will be done via a series of buttons and dropdown menus. For basic functions such as running linear, quadratic, and other regressions, the user will have a collection of buttons below the graph. When one of these buttons is clicked, a dialogue box will appear over the main application screen to prompt the user for dynamic information needed to run a proper regression. Once the information has been filled, the user can click a button reading “Ok” to generate a new graph based on the operation specified. The user can continue to run analyses until they want to save the current graph. At this point, they can click a button reading “Save as” to indicate a new graph, and a new dialogue box will appear with the current working directory and a generic name. The user can edit the file path and name and click a button reading “Save”. In the case the user is working with an existing graph, there will be a button reading “Save” on the main application screen which will follow a similar process. Other lesser used analyses will be contained within a dropdown menu that they can scroll through or search within to find the tool they would like to use.

b. **Combining Graphs:** The user should also be able to put multiple graphs over each other. To do this, there will be a button reading “Add graph” which will rely upon the same function as choosing an initial graph, allowing the user to navigate to the dataset they would like to examine. Once another graph is chosen, the main application screen will contain the previous graph with the new data graphed over the other graphs. The user can continue to add, select analyses, or save the current graph. The application will also keep track of which datasets have been added, and in which order. In the event the user has added an unwanted dataset, they can press a button reading “Remove graph” which will provide a dialogue box with a list of datasets – in the order they were added – so that the user can select which data set to remove. When that dataset name is selected, and a button reading “Remove” is pressed, the graph will be updated to remove the unwanted points.

5 Conclusion

Our envisioned end product aims to alleviate the major pain points faced by current researchers and other users when they interact with dendrometers. The level of difficulty, frustration, and potential danger associated with the current system is due to several factors - lack of portability, unreliable platform support, limited statistical analyses, and painfully involved data sharing - and our product expects to provide the solution to all of these issues.

Over the course of the document we described our main technological challenges when investigating the feasibility of our intended system, namely the communication with the dendrometer's embedded system, development of a cross-platform mobile application, implementation of cloud export and data sharing, and application of statistical analyses and their graphical representation. For each of these challenges we analyzed available options and defined a final decision regarding each obstacle, specifically C and its supersets, Kotlin with the Android SDK, AWS, and MPAndroidChart, respectively. We have arrived at a solution and architecture that is feasible and will provide a system capable of satisfying the product requirements, and we intend to move forward with the project and its next steps.

6 References

1. Akashdubey-Ms. (n.d.). *Introduction to azure storage - cloud storage on Azure*. Cloud storage on Azure | Microsoft Learn. https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction
2. Amazon. (n.d.-a). App Data modeling - AWS amplify datastore - AWS. https://aws.amazon.com/amplify/datastore/
3. Amazon. (n.d.-a). Cloud storage on AWS. https://aws.amazon.com/products/storage/
4. Amazon. (n.d.-b). AWS product and Service Pricing | Amazon Web Services. https://aws.amazon.com/pricing/?nc2=h_ql_pr_ln&aws-products-pricing.sort-by=item.additionalFields.productNameLowercase&aws-products-pricing.sort-order=asc&awsf.Free

%20Tier%20Type=*all&awsf.tech-category=*all&aws-products-pricing.q=s3&aws-products-pricing.q_operator=AND

5. Androidplot. (n.d.). http://androidplot.com/
6. *Code examples*. FTDI. (2020, July 31). https://ftdichip.com/software-examples/code-examples/
7. *Core features*. Weeklycoding. (2019, April 12). https://weeklycoding.com/mpandroidchart/core-features/
8. *Core libraries*. Dart. (n.d.). https://dart.dev/guides/libraries
9. Ddanny. (n.d.). *DDANNY/achartengine: Charting Library for Android applications. automatically exported from code.google.com/p/achartengine*. GitHub. https://github.com/ddanny/achartengine
10. Editor. (2020, February 27). *The good and the bad of Swift programming language*. AltexSoft. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-swift-programming-language/
11. *FTD2xxj*. SourceForge. (2013, December 26). https://sourceforge.net/projects/ftd2xxj/
12. GeeksforGeeks. (2023, April 3). *Introduction to Java*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-java/
13. GeeksforGeeks. (2019, December 6). *Introduction to Kotlin*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-kotlin/#
14. GeeksforGeeks. (2023b, September 20). *What is flutter?*. GeeksforGeeks. https://www.geeksforgeeks.org/what-is-flutter/
15. Google. (n.d.). Google. https://firebase.google.com/?gad=1&gclid=CjwKCAjw-KipBhBtEiwAWjgwrBpsII_T4-CXaKaw9QyYn-kRo02ONf3WY4RykY92Lm20SV_SkAotNRoClC4QAvD_BwE&gclsrc=aw.ds
16. Google. (n.d.). *Index*. Google Drive API | Google for Developers. https://developers.google.com/drive/api/reference/rest/v3
17. legege. (n.d.). GitHub. https://github.com/legege/libftdi/blob/master/src/ftdi.h
18. *MPAndroidChart documentation*. Weeklycoding. (2020, January 20). https://weeklycoding.com/mpandroidchart-documentation/
19. O'Reilly Media, Inc. (n.d.). *Learning React Native*. O'Reilly Online Learning. https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html
20. Pay as you go—buy directly | microsoft azure. (n.d.-d). https://azure.microsoft.com/en-us/pricing/purchase-options/pay-as-you-go/?srcurl=https%3A%2F%2Fazure.microsoft.com%2Ffree%2Fsearch%2F%3Fef_id%3D_k_cjwkcajw-kipbhbteiwawjgwrga3suylisajxbvdv8sx4gj4kgxhnk6sepvxiv31nsvknvq2bgo1mroczc4qavd_bwe_k_%26ocid%3Daidcmm5edswduu_sem__k_cjwkcajw-kipbhbteiwawjgwrga3suylisajxbvdv8sx4gj4kgxhnk6sepvxiv31nsvknvq2bgo1mroczc4qavd_bwe_k_%26gad%3D1

%26gclid%3Dcjwkcajw-kipbhbteiwawjgwrga3suylisajxbvdv8sx4gj4kgxhnk6sepvxiv31nsvknvq2bgo1mroczc4qavd_bwe

21. Poclitari, R. (2022, December 9). *Flutter vs React Native vs Kotlin: Which will drive App Development in 2023?*. Flutter vs React Native vs Kotlin: Which will drive app development in 2023? https://www.index.dev/post/flutter-vs-react-native-vs-kotlin-which-will-drive-app-development-in-2023
22. Point dendrometer | TOMST. (n.d.-a). https://tomst.com/web/en/systems/tms/point-dendrometer/
23. Raza, M. (2021, September 23). *Integration of google drive API with react native*. Mobile App Development Services. https://www.folio3.com/mobile/blog/integration-of-google-drive-api-with-react-native/
24. *React-native-azure-auth*. npm. (n.d.). https://www.npmjs.com/package/react-native-azure-auth
25. *React native Firebase*. React Native Firebase. (n.d.). https://rnfirebase.io/
26. *React native · learn once, write anywhere*. React Native RSS. (n.d.). https://reactnative.dev/
27. Software & data | TOMST. (n.d.). https://tomst.com/web/en/systems/tms/software/
28. *The rustonomicon*. Introduction - The Rustonomicon. (n.d.). https://doc.rust-lang.org/nomicon/intro.html
29. Threlkeld, R. (2018, March 5). *Using AWS with react native · REACT native*. React Native RSS. https://reactnative.dev/blog/2018/03/05/AWS-app-sync
30. *React native vs. Swift: Which one to use for an IOS mobile app?*. Relevant Software. (2022, August 11). https://relevant.software/blog/react-native-vs-swift-which-one-to-use-for-an-ios-mobile-app/