Haaga-Helia
University of Applied Sciences

Developing a full stack mobile application

Shan Liao-Mäkinen

Bachelor's Thesis
Degree Programme in Business Information Technology
2023

Haaga-Helia
University of Applied Sciences

Abstract

18.04.2023

Author(s)
Shan Liao-Mäkinen

Degree programme
Degree Programmer in Business Information Technology

Report/thesis title
Developing a full stack mobile application

Number of pages and appendix pages
51 + 2

This thesis explores software development from the perspective of developing a full stack mobile application for the Google and Apple marketplaces and devices in the year 2023.

The work begins with an introduction to the topic and presentation of the objectives. There is also an explanation to some premade choices such as JavaScript as the main programming language and the objective to use one shared codebase for both marketplaces.

After this, a short overview of the history of mobile software development is presented, followed by the theoretical framework for this thesis. The historical part goes over a short history of where touch screen enabled devices that we use today came from. It starts from the SMS-text message based mobile phones of the past and also has a small look into the future of mobile technology.

The theoretical framework part of the thesis will first explain the methodology chosen for this thesis, which are literary review and comparative analysis. The first topic to be reviewed is the mobile frontend framework of the application and choices that are available. Various comparisons are performed, and a choice as well based on the data available.

Next is the comparison and reasoning behind the choices of the backend. As JavaScript is a locked in premade technology choice from narrowing the scope of the thesis, this means JavaScript runtimes such as NodeJS are considered for the backend tool of choice. Various frameworks to combine both are also compared like React-Native, Flutter, Xamarin and Apache Cordova.

Following the backend is the database comparison, where a few options on the market are compared from a popularity standpoint as well as suitability with JavaScript native formats such as JSON. Alternatives such as MySQL, MariaDB and PostgreSQL are considered.

After this there is an introduction into what a web server is, how they were developed and what are the different choices are available for making an application based on JavaScript and web-technologies. There are performance considerations that are shown when choosing the right web server for this application, the comparisons are mostly between Apache and Nginx.

As the second last part comes the empirical part of how the programming work proceeded and what happened during development of the application and what choices were made.

Finally, a discussion part goes through what was learned during the development work and what could have been done differently, or what could be improved upon in the future, concluding into a thank you note for the reader of this thesis.

Keywords
applications, mobile apps, Android, iOS, web programming

Table of contents

1 Introduction

The purpose of this thesis is to document what parts are required to implement a full stack mobile application and whether it is possible and a good or a bad idea to create this kind of an application using a single shared code base or not. Modern web technologies of the year 2023 will be used with the goal to publish the work on the Google & Apple platforms.

A code base is everything that is needed to deploy or use an application, along with the help files to understand it and the complete body of source code. (Sheldon, 2023).

This code base for a project usually exists within one version control repository, but in the case of multiple platforms there are sometimes two or more version control repositories for code for each platform. This project aims to use one repository for two platforms.

A “repository” is a way to speak of all the copies of the same project, in a distributed version control system, such as git. Version control (git) is a way for tracking changes in a project that is especially helpful for sysadmins, programmers, and site reliability engineers (SRE) etc. (Broberg, 2019).

This topic was selected as the author wants to challenge herself to achieve a real-world result from her studies of the software industry to serve as a stepping-stone to further projects in the field of professional web development. This work will go through what considerations and steps are necessary to develop a full stack mobile application that can be published in the year 2023. With the completed work it is possible to see what were all the parts that were required to develop a full stack mobile application.

The problem this work will try to solve is which could be good choices for technologies on each technological stack level (frontend, backend, database) for the objective of this thesis, which is creating a successfully functioning mobile application with one shared code base instead of multiple code bases per platform.

A technology stack is a grouping of programs, frameworks, and other tools that aid in the creation, delivery, and upkeep of software. It frequently includes parts made for particular purposes. For example, Microsoft's social software, collaboration tools, and document management tools for convenient work and cooperation with colleagues are all part of the Microsoft SharePoint technologies stack. Companies in tech use tech stacks to build more modular solutions and reduce dependencies. (Bahrynovska, 2022).

This thesis will serve also as instructions considering the future career of the author for creating applications for mobile devices using technologies that have been proven to work in 2023. The results may also indicate that it might not be a good choice to create a mobile application with one shared code base. It is also possible that making custom code for each platform to create mobile applications without a shared code base is a better choice. Results to this come after this thesis work is finished and they are available for discussion.

1.1 A brief history of the mobile app marketplaces.

Mobile applications have become increasingly popular as the software company Apple Computer Inc. launched their mobile application marketplace called the App Store in July 2008, after which - in just one week - people downloaded approximately 10,000,000 million mobile applications, also known as apps. This newsworthy commercial success led to the word App winning the word of the year award for the year 2010 as awarded by the American Dialect Society (JetRuby 2017). Another software company called Google followed Apple in suite, after its amazingly successful business move and consolidated all of the Android operating systems app marketplaces and stores into one platform, called the Google Play Store. The Google Play store was previously known as the Android Market, which existed between 2008 and 2012 alongside Apples App Store. The launch of the store in 2012 from Google had 450 000 Android applications and it had grown to the second biggest source of revenue for Google outside of its internet advertisement business, now having 1,5 million applications where Apple only had 1,2 million apps in their store (Callaham 2017).

These two big software companies succeeded in creating strong foundations for markets where new mobile application software can be sold and used on. The applications and how they can be made for these marketplaces are the heart of this work.

1.2 Focus of the thesis and the full stack approach explained.

This paper concerns itself mostly about developing a full stack mobile application for both marketplaces, using a single code base or one set of source code. A key focus and objective are to keep code changes between platforms to a minimum, to reach a publishable Minimum Viable Product-style mobile application for the Google and Apple marketplaces with small available human resources that the author can provide for the work.

The work will focus on a specific mobile application called Mamabeibi™ which is a trademark owned by the company Zodiac Fox Oy (Ltd.). The goal is to implement this application for both the Apple and Google marketplaces. The author of this work is a founder of Zodiac Fox Oy. This objective may change in the future of the application if sharing code is found to be an impossible job to complete.

The main parts of this work will involve the implementation of single modern mobile application and the development of all the technological layers also known as the stack of the application. The work starts from the frontend, which is a word that refers to the user interface and other parts that the user sees first. The work will continue next with at least the theoretical parts of the backend and the database, which the user commonly does not see, but uses through the frontend. The backend is where the applications business logic usually exists and data processing. The database of the application is where all data is stored that the application saves for use later when the user closes the application and returns to it. This type of development work may be referred to as full stack application development.

In order to understand what this work refers to as full stack development, it is helpful to examine the definition of a full stack development. Currently the definition of full-stack refers to an application that contains three layers, one being the frontend or the user-inter-

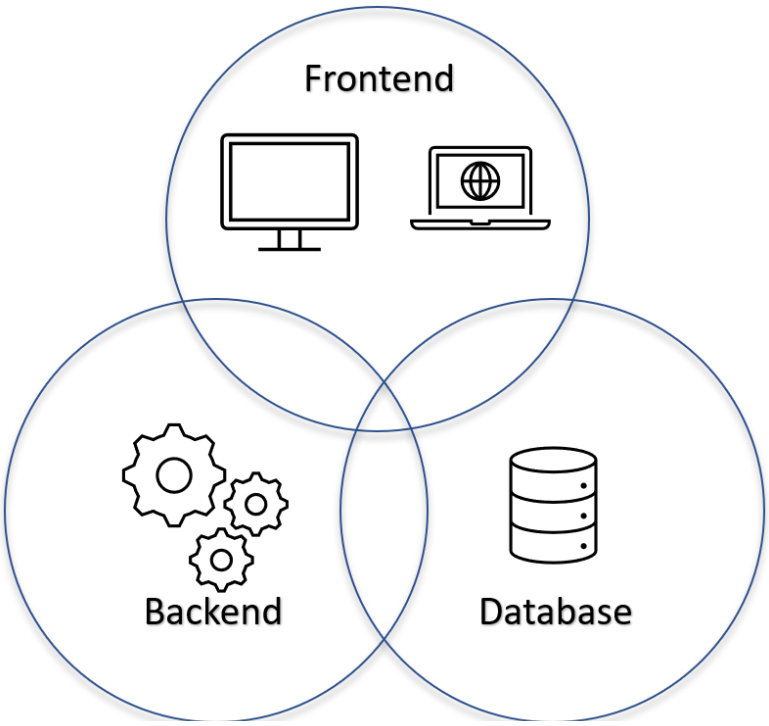

Figure 1. The full stack of an application, as illustrated by the author.

face layer. Another being the backend, of the application layer, and the last being the database-layer. These three together combined are the writers understanding of what full stack means, as seen in Figure 1 below.
The more official text on full-stack development explains, that it is a new concept of systems development, for which a consensus definition has been suggested in an article by Shropshire, Landry and Presley, 2018 called Towards a consensus definition of full-stack development. It says the following:

Full stack development is a methodology which addresses all stack layers and in doing so creates a complete, implementable solution to business requirements. Full stack developers have broad experience among all stack layers and expertise in a few layers. They should be able to render a minimum viable product in any given stack (Shropshire et. Al., 2018).

From this quote the development methodology concerning the concept of technology stack layers, are better explained with Figure 2. From this figure it is clear that there is a requirement for wider knowledge from various information technology industry techniques required to complete a full-stack mobile application.

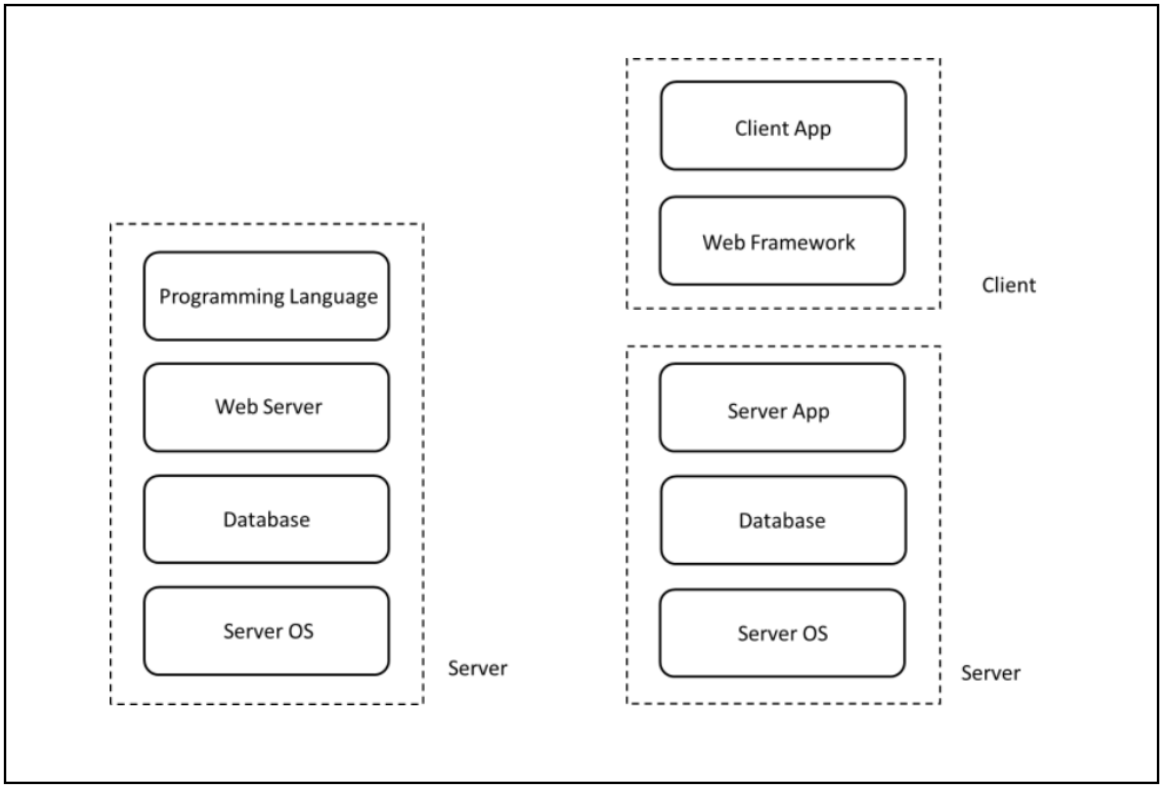

Figure 2. Core Server-Side and Distributed Stack Architectures (Shropshire et. Al., 2018).

1.3 Further narrowing of the scope of this thesis

To further manage the complexity of this thesis work and successfully handle completing it, the scope of it is to be narrowed. The narrowing focuses mostly on providing the reader with the frontend empirical part of the thesis, due to the big amount of work to document all the details of the backend, database and other systems related to the completion of a full stack mobile application. If the author has more time, there will also be backend and database sections added to the work of this thesis.

Also, for this full stack mobile application development thesis, there have been made multiple premade choices to focus and narrow down the scope of the work which includes the programming language known as JavaScript. On page 18 of the JavaScript Bible, the creator of JavaScript Brendan Eich describes JavaScript as a general language, useful apart from HTML and XML. Embedded in servers, authoring tools, browser plug-ins and other kinds of browsers (Goodman 2007, 18). JavaScript was also chosen due to its popularity, as seen in Figure 3 below where languages such as Java and Python are not as popular. JavaScript has been on top for 10 years in 2022 as the most popular programming language according to Stack Overflows Developer Survey of 2022. (Stack Overflow, 2022).

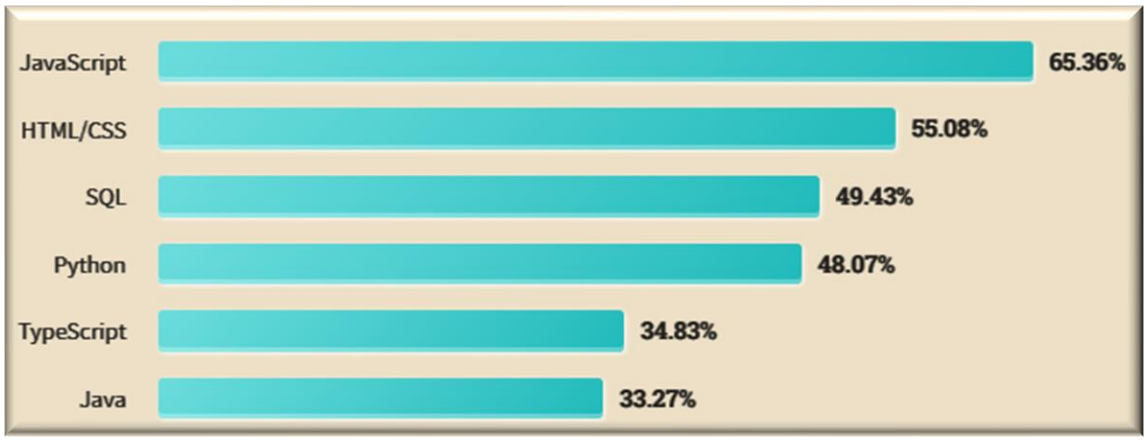

Figure 3. Most popular programming languages 2022. (Stack Overflow, 2022)

The general benefits from using JavaScript on the backend and on the frontend are one of the main reasons why this language has been pre-selected for this work. This is predicted to make sharing code easier between the layers of the full stack application.

The front-end framework will be chosen by a comparison of different frameworks from at least two different sources. The same will be done to backend and database layers with the limitation that the programming language must be JavaScript.

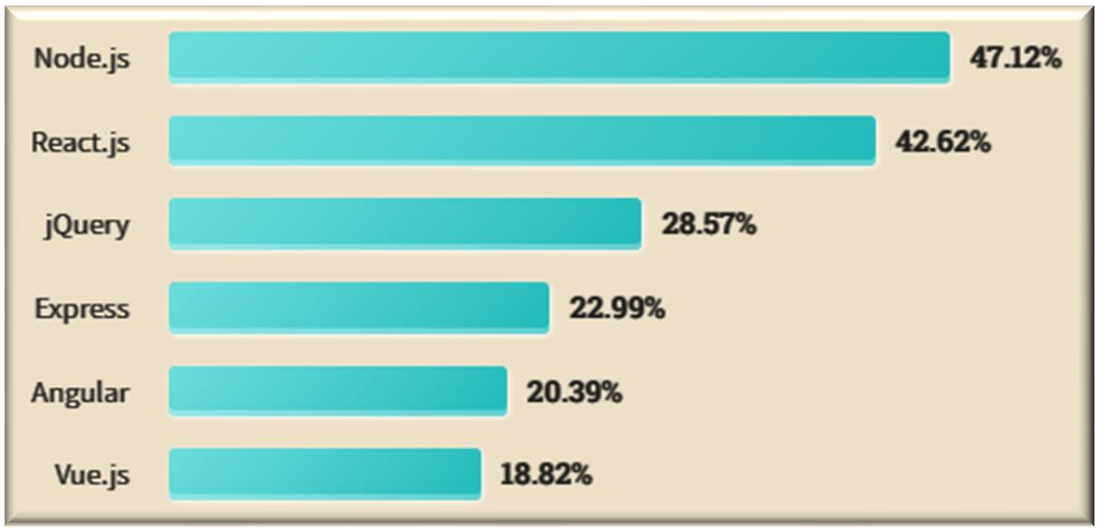

Figure 4. Top Web Technologies (Stack Overflow, 2022)

The front-end client framework is going to be one of variety of mainstream choices available in the year 2023, compared from recent and relevant sources on the Internet. First source is the Stack Overflow Developer Survey of 2022 as seen in Figure 4. It confirms that we can use JavaScript in the backend, as it is the most popular programming framework for backend too, which uses the same language as the frontend. The same kind of comparison process will be used for the backend and database selection of this work.

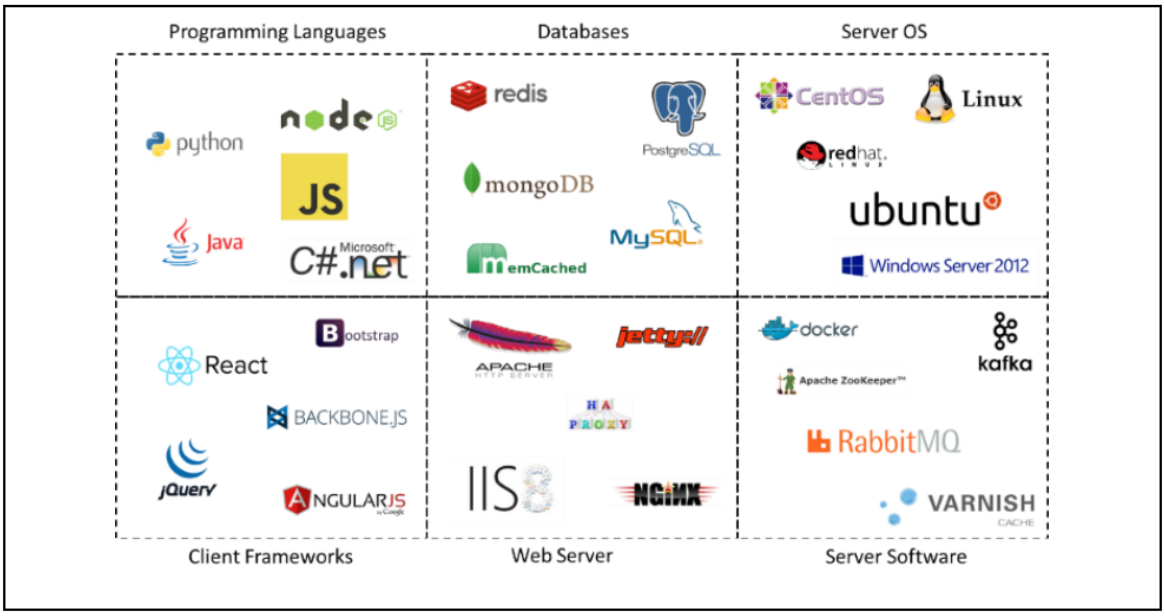

Figure 5. Mainstream and Emerging Stack Components (Shropshire, et al., 2018)

For comparison, Figure 5 shows mainstream technologies from a long time ago, in the year 2018. Some of these technologies are no longer relevant in the year 2023, but for the purpose of this thesis, they confirm that NodeJS and JavaScript are good popular choices when developing a full stack mobile application as they have a long stable history.

The technology selection will probably be JavaScript based React Native or Flutter or the Xamarin based on technology recommendations from various sources, as these save time by allowing a single code base for both iOS and Android platforms. (Furman, 2023).

1.4 A brief history of mobile applications and mobile technology

The history of mobile applications may have a starting point after twenty years of the events on 1973, on April the 3rd, when the very first mobile phone call was made by a certain Mr. Martin Cooper of Motorola to Doctor Joel S. Engler of Bell Labs. After approximately twenty years of research and development from this point in time, the first mobile applications for smart phones appeared courtesy of the IBM Simon. (Rajput 2015).
This means it took twenty years to go from the first mobile telephone call towards the first smart phone applications.

Technically the history of mobile phones on which mobile applications run on, dates all the way back to 1908 to a patent in Kentucky for a 'wireless telephone'. This though was more of a two way-radio than the phones that we are used to seeing in the modern day. (Dudley, 2018). It is interesting to note that the first ever text message that was sent, was sent by Neil Papworth on the 3rd of December 1992. He worked for the Sema Group as a test engineer, and the message simply said: "Merry Christmas". (Dudley, 2018).

The next big leap in mobile applications was the smartphone called BlackBerry, that was released in 2002. BlackBerry was formerly known as Research in Motion (RIM) and they developed an Interactive Pager (Inter@ctive Pager) that could receive and send messages over the Internet, which was the predecessor of the first BlackBerry smartphone. After these discoveries, these smart phones started supporting more and more advanced software and much of the early development was driven by Nokia, with the SymbianOS. (Rajput, 2015)

In 2003 the 3G network was beginning to become adopted worldwide, which encouraged faster speeds of data for mobile technology, such as 0.2MB/s at least. This megabyte limit is what was required to classify a network as 3G. (Dudley, 2018) This was a signal that the Internet speeds were getting faster, so mobile applications could actually start transmitting even more data than ever before.

The most significant moment in smartphone history that many experts note it to be, was June in the year 2007 when the iPhone was first launched. It was not the first touch

screen phone on the market, but it arrived at market when most of the mobile phone industry was physical keyboards, tiny screens and unwieldy designs. This was one of the main factors for Nokia's demise. (Dudley, 2018)

The Apple iPhone 3G was among the first phones to have mobile applications as a priority, as Apple launched their App Store with 552 applications at launch in the year 2008. Interestingly Android launched its 'Android Market' – which is currently known as the Google Play Store – in the same year. The 'Android Market' did not offer paid app support until the next year. From 2008, the 'Apple vs Android' debate was marked to start in every sense of the word. (Dudley, 2018).

2010 Was another meaningful year for mobile applications, as the word "app" was chosen as the word of the year by the American Dialect Society, following Android and Apple pushing apps on their marketplaces. (Dudley, 2018).

In 2018, the mobile phone industry has come a very long way. Giants no longer seem to dominate the industry, as multiple new manufacturers launch products putting Samsung and Apple to the test. There are never-before-seen concepts being published every other month, and it seems smartphone innovation is at an all-time high. Perhaps we will see transparent phones from movies such as Iron Man 2 in the future, you never know. (Dudley, 2018).

The history of mobile applications stays interesting as we move into a new time of mobile applications and the development of mobile applications.

2 Theoretical framework

This thesis uses literary review as its main research methodology, paired with comparative analysis of various technologies and techniques to develop a full stack mobile application. The literary reviews are used to investigate modern state-of-the-art topics, but there lies the danger of the author building the research on flawed assumptions (Snyder, 2019). By using a systematic literary review, these possible flawed assumptions at least are brought bare. This is important so the reader understands that what the author has based her assumptions on. (Tranfield, Denyer & Smart, 2003).

2.1 The mobile frontend framework for this project

According to one source, the most popular mobile application development frameworks for the year 2023 were 1. Flutter, 2. React Native, 3. Xamarin and 4. Apache Cordova, among others. (Herembourg, 2022).

Another source reported as the most popular choices for mobile cross platform mobile application development frameworks of 2023 to be 1. Ionic Framework, 2. React Native, 3. Flutter, and 4. Xamarin, and after this a few others. (Sachan, 2022).

It is worthwhile to note that the Ionic Framework based on Apache Cordova. (Jasnowska, 2019). Calculating Ionic as Apache Cordova, this makes the first comparison of sources show that the most popular mobile frontend frameworks are: 1. React Native, 2. Flutter, 3. Cordova and 4. Xamarin.

Next, I will briefly go over the technologies and their backgrounds so we can have a general idea of each framework to better understand what is available to complete this work.

2.1.1 React and React Native

The possibilities for front-end developers to create user-friendly interfaces have significantly increased since the release of React.js. Let’s first go through a quick history of Reacts beginnings to gain a better understanding of it.

The Facebook developers first encountered certain maintenance concerns in 2011. The team needed extra personnel as the Facebook Advertising app gained more capabilities in order to maintain faultless operation. They experienced a slowdown as a firm due to the expansion of their personnel and app functionality. As they dealt with numerous cascade updates over time, their software become more challenging to manage.

Facebook's programmers eventually ran out of time to handle these cascading upgrades. Their code needed to be urgently updated to become more effective. They had the appropriate model, but the user experience required action. As a result, Jordan Walke created a prototype that improved the procedure, and this is when React.js was born.

React is now a JavaScript library used to build web applications, mobile apps, and desktop applications using HTML, CSS and JavaScript. Jordan Walke created it in 2011 and it

was called FaxJS at that time, that was the early prototype of React which was shipped in a search element on Facebook. (Hámori, 2022).

Reacts usage of Components, the Virtual DOM, Lifecycle Methods, JSX, and React hooks are a few noteworthy elements that have contributed to the development community's appreciation for the framework. Each of these aspects deserves a deep dive study that examines the benefits, drawbacks, and application scenarios but it is out of scope of this work. Nevertheless, when taken as a whole, they provide React three key benefits that have contributed to and continue to maintain its enormous success. Clean programming, quick performance, and a healthy community are those three benefits.

Facebook released React in 2013 as open source. It wasn't until the year 2013 after Facebook released it as open source, that developers and designers started to take notice of this great new development.

The code basically allowed developers to create views in an interactive fashion and made it very responsive. Mr. Walke designed React to help developers build complex UIs that are both fast and easy to maintain. Although the library is still relatively new, it has already become very popular among web developers because of its ability to help create web applications with little code. (Arancio, 2021).

React is now the most popular of JavaScript frameworks in 2023 and it has kept this popular position all the way since 2016 as according to Figure 6.

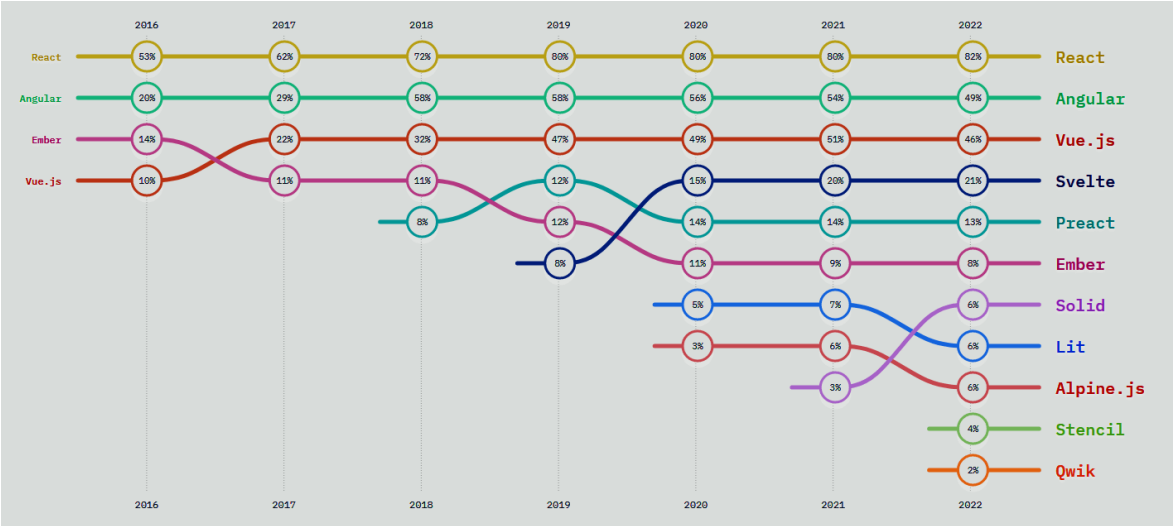

Figure 6. Usage data of JavaScript frontend frameworks in 2023 (The State of JS, 2023)

One of the main features of React is that it uses Functional Components, which means that they can be reused across multiple pages or even servers. This makes it easier for the user to create large applications with less effort than if you were writing all the code from scratch each time, they want something done differently in their application (for example: adding a new feature). (Sirgur, 2023).

One of the most popular technological movements today is the rise of JavaScript and its derivatives. JavaScript has become an integral part of so many apps that it's hard to think of a web app that doesn't rely on it in some way. React Native is mainly a framework that integrates JavaScript and React. It is a framework for building native mobile applications in JavaScript and has been an integral part of this movement as well by offering a solid alternative on iOS and Android platforms.

React Native is a mobile app development framework that enables developers to develop native apps on Android and iOS platforms allowing developers to create apps that look and feel like native apps, but still use the same JavaScript code.

React Native as an open-source project is led by members of the JavaScript, Clojure and React communities and its highly popular. A lot of developers also help keep React Native free of bugs and give lots of testing with the help of the open source community, submitting bugs and fixing them. It is maintained by Facebook. (Sachan, 2022).

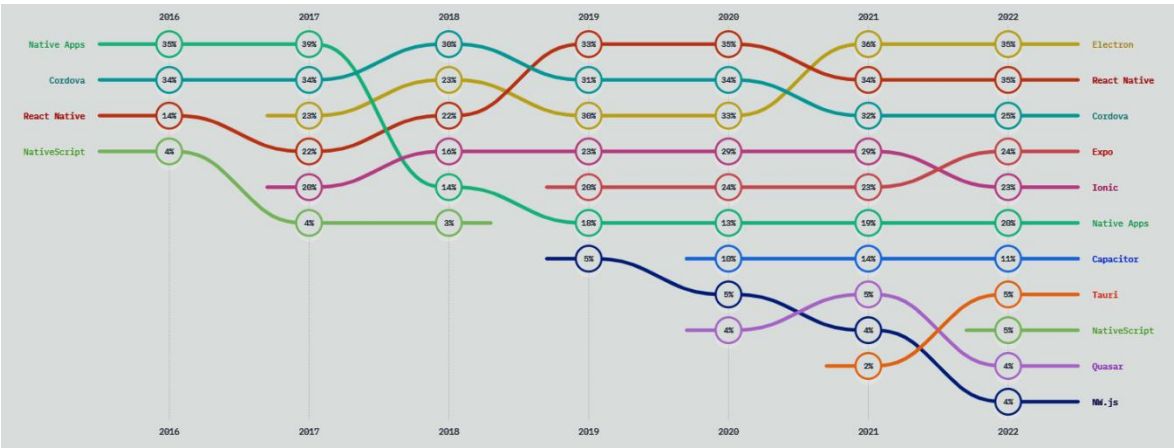

Figure 7. Mobile and Desktop JavaScript libraries by usage. (State of JS, 2023).

Figure 7 measures usage with the formula: (would use again + would not use again) / total votes. It shows us React Native has kept a highly popular position, as the 4th option Expo is also a part of the React Native deployment approach, so we can consider the top three libraries for mobile development to be: 1. Electron (35% people would use it again) 2. React Native (same 35% + 24% from Expo, totaling at 59%), 3. Cordova (25% + 23% from Ionic which is based on Cordova) so we have our top three Electron, React, Cordova.

2.1.2 What about Flutter?

The name Flutter came from The Sky Engine that was created as a result of the Google development team hearing the cry to "Open The Sky" in October 2014. It changed over time to become Flutter, an open-source UI development kit that has revolutionized the mobile app development industry.

Hot Reload for Android phones was the sole standout feature of Flutter in its early stages. But, this one feature alone changed the game, cutting the time it took to create an Android app from a long 7 minutes to a fast 400ms. But, the purpose of Flutter went well beyond creating mobile applications.

The goal of the Google engineers was to improve everyone's online experience by making it quicker, better, and more organized. This is why they started by getting rid of Chrome's outdated web compatibility support, which made it 20 times quicker than before. This straightforward experiment launched the development of a cutting-edge platform that is today the go-to option for developers everywhere, Flutter.

Eric Seidel, one of the Flutter co-founders, made the formal announcement at the DART dev summit in April 2015 about Flutter and the world of mobile app development was transformed permanently. The Flutter team set out with one goal in mind: to build a platform that enhanced the overall development experience while simultaneously being beautiful & highly functional. This was not enough for them so they took care to keep the platform open-source and adaptable so that developers could modify it to meet their needs.

Dart, the programming language used by Flutter, was at the center of everything. For programmers wishing to create cutting-edge mobile apps, this extremely portable and user-friendly language is the ideal option because it supports compilation to efficient JavaScript as well as Intel and ARM machine code. (Hardik, 2023).

Efficiency and reliability have been proven with Flutter because apps with millions of downloads and daily users have been built with Flutter, such as Google Ads, Alibaba, etc. Google Ads has over 10 million+ downloads so it’s very popular. (Sharma, 2020).

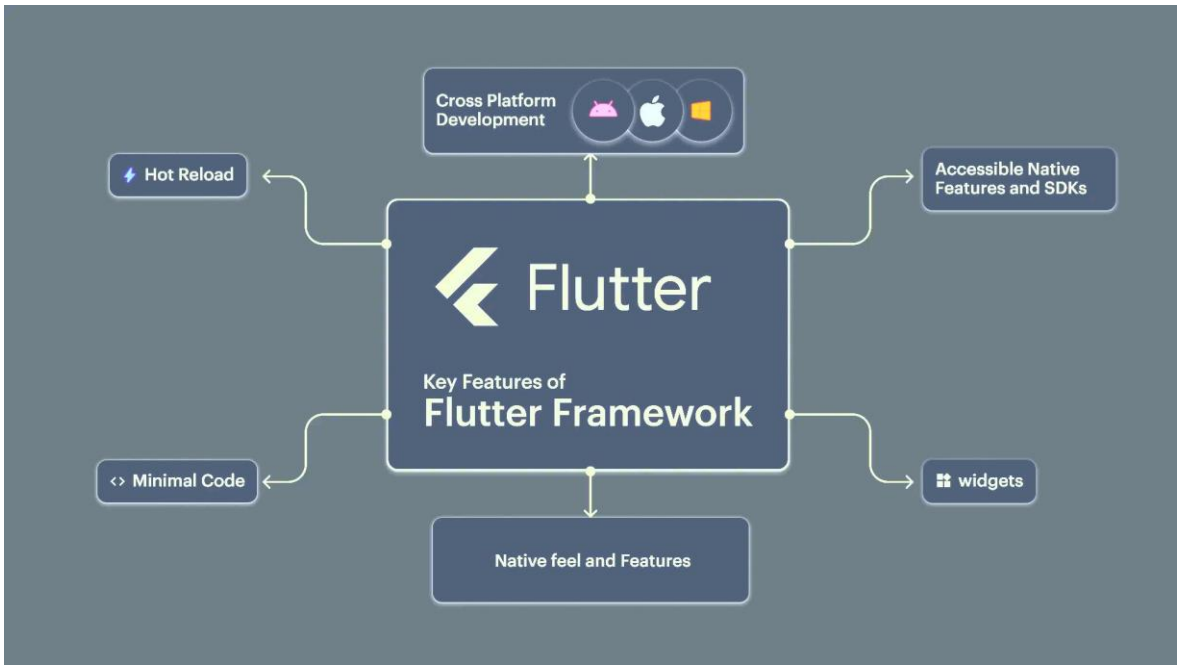

Figure 8. Features of Flutter. (Hardik, 2023).

Minimal code comes from the programming language of Flutter, Dart. It is strongly typed and object-oriented. Programming is done declaratively and reactively in Flutters Dart. Flutter speeds up app startup time because it does not require the JavaScript bridge. Because the Flutter framework was created in Dart and supports a variety of platforms, the written code can be used to support channels including Mobile, Desktop, and PWA. (Sharma, 2020).

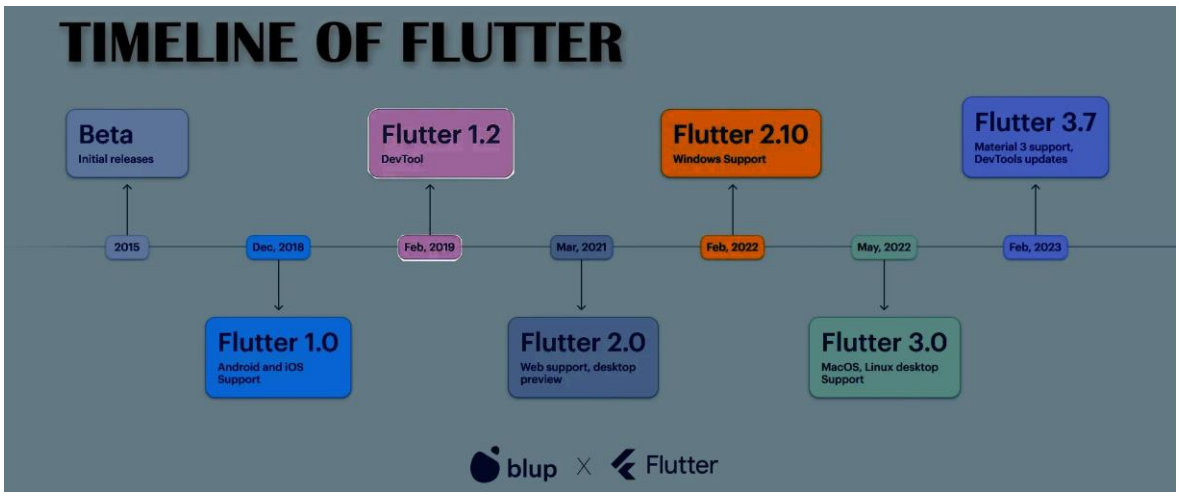

Figure 9. Timeline of Flutter. (Hardik, 2023).

Legal conflicts and technological developments were part of Flutter's timeline in Figure 9. to become the top platform for developing mobile apps. The platform's name changed from Sky Engine to Flutter in the space of a few short years (thanks to a few lawyers). (Hardik, 2023).

The first commit to Flutter was made in May 2017, and in February 2018, after much waiting, the beta version was eventually made available. Google waited until December 4th of 2018 to make the major announcement that Flutter 1.0 was now available and ready for use by everyone. This stable release changed the game by giving developers the resources they needed to create apps that were ready for production. In addition, Google had even higher intentions for their flexible platform than the early releases of Flutter, which were primarily focused on providing a mobile SDK to build native Android and iOS apps from a single codebase. (Hardik, 2023).

Sharing a single codebase is one of the objectives of this thesis as well. From this point of view, Flutter does seem like a good option to go for when making a mobile application.

With the addition of The Hummingbird, which was first presented as a preview in version 1.5, Flutter's growth impressed users even more. With the help of this exciting update, Flutter was able to expand its functionality to the web, increasing its adaptability and accessibility on other platforms.

More features including UI tools, widgets, and app bundles were included in later versions. With the addition of multi-device debugging, Layout Explorer, Flutter Web in beta, and support for iOS 13, Flutter version 1.12 stood out. These upgrades gave developers aiming to make attractive and responsive apps a lot of new opportunities. (Hardik, 2023).

Google introduced Flutter 2.0 on March 3, 2021, in a grand Flutter Engage Event, igniting the excitement of the developer community. The Null Safety feature, a new Canvas Kit renderer, more enhancements for iOS, a reliable Flutter Web, and other incredible features were all included in this version. (Hardik, 2023)

It's fascinating to note that some of the major people who helped Flutter get to where it is today have also moved on to explore new realms as Flutter continues to make groundbreaking advancements in the tech sector of application development.

The creative co-founder of Flutter, Eric Seidel, is currently enjoying a well-earned retirement. Tim Sneath, a founding member of the Flutter team, is now the Director of Product and UX for both Flutter and Dart, whereas Martin Aguinis, the Product Marketing Manager at Flutter, left to become the CEO and co-founder of Clipjoy. (Hardik, 2023)

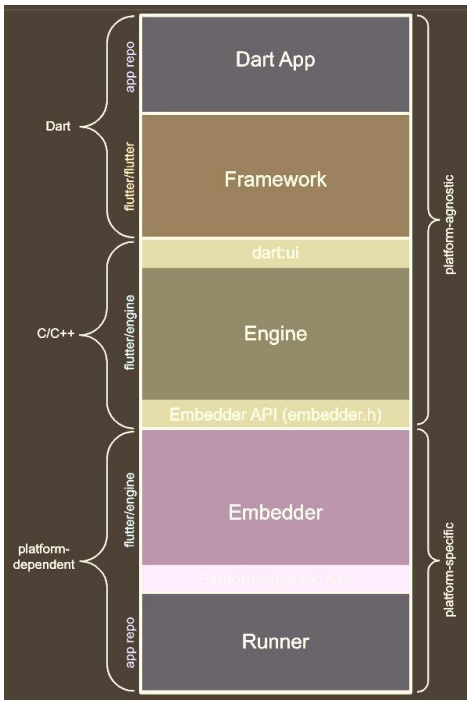

Figure 10. Anatomy of a Flutter App. (Flutter.dev, 2023).

On Figure 10. you can see how the architecture of a Flutter application looks like on the mobile platforms. This shows that the top layer requires the use of the Dart programming language. Because Flutter requires the use of the Dart programming language, it is unlikely that this work will be using Flutter. This is because learning a new language would cause delay for the writer of this thesis to complete the work with the small amount of time and resources available. It is good to learn about the background of why Flutter chose Dart as the programming language for Flutter.

Why did Flutter choose to use the dart programming language? The Flutter team considered a variety of languages and runtimes during the early stages of development before settling on Dart for the framework and widgets. Flutter took into account the requirements of the creators, developers, and end users of the framework when using four main evaluation dimensions. The team discovered other languages that complied with parts of the criteria, but only Dart received top marks across all of the evaluation criteria and dimensions. The four dimensions were: Developer productivity, object-orientation, predictable with high performance, and fast allocation. Dart had all of these.

The combination of two critical Flutter features is supported by Dart runtimes and compilers: a JIT(Just In Time)-based fast development cycle that enables shape changing and stateful hot reloads in a language with types, as well as an Ahead-of-Time compiler that generates effective ARM code for quick startup and predictable performance of production deployments. (Flutter.dev FAQ, 2023).

2.1.3 What about Apache Cordova and its history?

A native application for a variety of mobile platforms can be made using HTML, CSS, and JavaScript content thanks to Apache Cordova, a hybrid open source framework. The name Cordova is also the real name of the street where Nitobi's office is located in Vancouver, Canada. Nitobi is the company that originally built the predecessor of Cordova, Adobe PhoneGap later renamed to Apache Cordova.

Your web application is rendered within a native WebView by Cordova. An application component called a WebView is used to display web material inside of a native program (such a button or a tab bar). A WebView can be compared to a web browser without any of the common user interface components, like a URL field or status bar.

The web application operating inside of this container can open additional HTML pages, perform JavaScript code, play media files, and communicate with external servers exactly like any other web program that would operate within a mobile browser. A hybrid application is a common name for this kind of mobile application.

Web-based applications are typically run in a sandbox, which prevents them from having direct access to the device's different hardware and software characteristics. Your cell device's contact list is a nice illustration of this. A web program cannot access this database of names, phone numbers, emails, and other pieces of information. Cordova offers JavaScript APIs to enable access to a range of device functions, such as the contacts database, in addition to a basic framework for running a web app inside a native application.

Many plugins are used to make these features available. The native functionality of the device and our web application are connected using plugins. For teams who want the benefit of having Ionic manage ongoing updates, security patches, compatibility, and other aspects of native device access, the Ionic team has gone a step further and created Ionic Native, which has TypeScript interfaces for over 200 of the most popular plugins. Ionic Native also offers enterprise supported versions.

The distinction between PhoneGap and Apache Cordova is unclear to developers quite often. We must try to understand the project's beginnings to remove this unclarity. Many engineers from the Canadian web development firm Nitobi participated in an iPhone development camp at the Adobe offices in San Francisco in late 2008. (Griffith, 2023).

Nitobi as a company participating in the iPhone development camp in San Francisco, looked at the possibility of running their web applications in a native environment utilizing the native WebView as a shell. The test was a success and it worked. They increased their efforts over the following few months and were successful in using this solution to build a framework. The technology was given the name PhoneGap because it gave web developers the opportunity to connect the gap between their online applications to the device's native capabilities. As the project developed, more plugins were made so that the phone could access more functions. The effort gained additional contributors, increasing the variety of mobile systems it could support.

Nitobi was purchased by Adobe in 2011, and the PhoneGap technology was given to the Apache Foundation. Eventually, the project was given the new name Cordova, which also happens to be the street name of Nitobi's office in Vancouver, Canada.

It is simple to mix up the two projects as there is both Apache Cordova and Adobe PhoneGap. Due to the interdependence of the two projects, this name issue can be a source of annoyance when researching a problem during development and needing to use both Cordova and PhoneGap as keywords to discover the solutions, or even reading the appropriate documentation.

If you consider how Apple has its Safari browser, but it is based on the open source WebKit engine, you can better grasp the differences between Apache Cordova and PhoneGap. The same is true in this case: PhoneGap is the version of the framework that is branded by Adobe, whereas Cordova is the open-source version. There isn't much of a difference between the two in the end. The functionality is the same even though the command-line interfaces have a few minor variations. Just mixing the two projects in the same application is not recommended. There will be issues if Cordova and PhoneGap are used in the same project.

The primary difference between the two projects is the existence of paid services offered by Adobe under the PhoneGap name, most notably the PhoneGap Build service. By using this hosted service, you may have your application remotely built into native binaries, doing away with the requirement to locally install the SDKs for each mobile platform. This service can be used with the PhoneGap command line interface tool (CLI), but not with the Cordova CLI. Although it doesn't come with a UI SDK, Cordova offers ways to exploit native mobile features to build entirely native applications. With the help of Cordova, you can turn your project's HTML, CSS, and JavaScript into something that can be uploaded to other app stores. (Griffith, 2023).

2.1.4 What about Xamarin?

Microsoft offers an open-source framework for creating mobile and desktop applications called Xamarin. For iPhones, iPads, Android smartphones, Android tablets, etc., the user may create 100% native apps.

Xamarin could be the tool of choice if the user wants to create apps for the Universal Windows Platform (UWP) because Xamarin allows to utilize C# (C-sharp) and.NET for app development purposes.

Because common logic can be created for several app versions for platforms like Android, iOS, and UWP, the platform is even better for quicker app creation and little maintenance because of the shared code base.

Using Xamarin, the user can easily perform anything that the user could do on a variety of native mobile app development platforms including Java, Swift, Kotlin, Objective-C, etc. All of this is done on the well-known IDE for software development, Visual Studio. (Nath, 2022).

What is an IDE? This is what programmers use to assemble the many components of creating a computer program. This all is done using an IDE, or integrated development environment. IDEs boost programmer productivity by merging common software development tasks such as editing source code, creating executable files, and debugging into a single program. (Codeacademy Team, 2023).

Mono was the precursor to Xamarin. Mono was an open-source initiative to adapt.NET for the Linux platform. Afterwards, several businesses bought Mono and the original developers were fired from their jobs. Later in 2011 the same group of original developers established Xamarin. Novell gave Xamarin permission to utilize the Mono, MonoTouch, and MonoAndroid development licenses in the middle of 2011.

Then finally Microsoft purchased Xamarin in 2016. The Xamarin SDK which is now an open-source app development platform was introduced by Microsoft that same year. The platform makes use of C# and the .NET framework. All versions of the Visual Studio IDE support the SDK.

Xamarins features include **Native Emulators**, which are emulators for multiple mobile platforms and OS releases. They are built within the Xamarin IDE. You don't need to pay extra for emulators to run your program on devices like the Google Pixel, Samsung Galaxy, iPhone, iPad, Android TV, Apple TV, etc. that are emulators. (Nath, 2022).

Xamarin has **SDK Bindings** for every platform SDK like iOS, Android, etc. As a result of this, the user can create native apps of superior quality faster than with other products. The Xamarin IDE's primary coding language is **C#.** Many dynamic functional constructs, including LINQ, lambdas, parallel programming, and others, are supported by C#.

With the Xamarin IDE, you can find for yourself important **third-party scripts** to construct your apps. The code scripts could be written in any common language used to create mobile apps, such as Java, C++, Objective-C, Swift, etc.

The.NET **Basic Class Library** is utilized by the programs you create (BCL). Database, XML, Serialization, String, IO, Networking, and more technologies like these are all simplified and complete in BCL.

The well-known Microsoft **Visual Studio** program serves as the Xamarin coding environment. You won't have to learn any new coding languages or tools as a result if you know how to use this tool already. Other advantages provided by VS IDE include solution management, project management for app development, code auto-completion, a project template library, and more. (Nath, 2022).

The ability to create solutions with one codebase for iOS and Android is one of Xamarin's key selling factors. With between 60% and 95% reusable code, native performance, and C# code built within the.NET framework, Xamarin is a feasible solution.

Applications created using Xamarin have access to a wide range of features and can support platform-specific features like NFC, ARKit, CoreML, and Fingerprint. Third-party libraries can be incorporated, such as those from Facebook, Google Play Services, and Google APIs for iOS.

The fact that Microsoft supports Xamarin is among its many wonderful features. One can now anticipate constant developer support, a variety of educational options, stability, and performance. A wide range of industries, including healthcare, energy, media, transportation, and more, use Xamarin to create apps and service users. (Georgiou, 2023).

2.1.5 The choice from all the above mobile frontend frameworks

Based on our descriptions of the top four options, React Native, Flutter, Cordova and Xamarin the decision which one to use comes from the table below:

Table 1. Framework comparisons and decision of framework choice.

Framework	Pro	Con	Barrier of entry
React-Native	Popular	Learn JS XML	Learn the JSX/TSX language.
Flutter	Efficient	Learn DART	Learn the DART language.
Cordova	Lightweight	Not as popular	No need to learn new language.
Xamarin	Big support	Learn C#	Learn the C# language.

From Table 1 we can determine that Cordova is a good choice because there is no need to learn a new language such as JavaScript XML, DART or C# as JavaScript itself first the constraints of this work. Cordova supports using simply HTML, CSS and JavaScript so this will be the choice to go forwards with this work on the frontend framework side as it will save time avoiding to learn a new language for this work to be completed. Cordova does have it's disadvantages due to not being as popular as React-Native or Flutter, but it should save in development and publishing time by allowing a simple HTML CSS JavaScript website to be made of the Mamabeibi ™ application and having that uploaded into the Apple and Google marketplaces.

Some possible alternatives towards this goal could be the Ionic framework, NativeScript, or some other framework like Electron – which are all technologies to enable a HTML5 web application to function on a mobile platform as a hybrid application, but there is not enough time to go through these three alternatives at this point. Perhaps Electron could save in development time similarly as Cordova does and in the costs of not needing a custom language such as Reacts JSX but for now, we will proceed with Apache Cordova.

2.2 The backend framework

Considering the use of JavaScript on the frontend of the application, an effort is made to save time and use the same programming language on the backend. This sets a unique constraint to which programs can be used as the server software of the backend. NodeJS is a technology that the author has used before, which sets alternatives such as RingoJS, and others behind it for practical reasons. One of the key reasons being not having to learn the quirks of a a new runtime environment, especially one that differs in that the RingoJS environment runs on the Java Virtual Machine, which is an entirely different runtime

environment from a simple C application that NodeJS is (Ramón, 2019).

It could be a good asset to have access to the Java ecosystem, but as the Mamabeibi ™ software has a simple backend, it feels like it might not be necessary. It is true also, that in 2023 the NodeJS ecosystem is large as well because it has had time to develop.

As the filtering factor for the comparison of frameworks for the backend is set by JavaScript and NodeJS, the comparison of which backend frameworks to use in order to perform our application logic and data persistence actions over the Internet will focus on the following frameworks:

The comparison of backend frameworks starts from a point where JavaScript is the language of choice. The frameworks compared will be from a 2020 review of top 10 frameworks, which include Express, Next.js, Meteor, Koa, Nuxt.js, Nest.js, Fastify, LoopBack, Hapi, Restify (Md, 2020).

The purpose of the comparison reviews and the target of the application is to be used to build a simple backend to authenticate the user, save the users state as well as the users data for the application to user later on, in order to target advertising later on for the customers and to provide better user experiences as the application can remember which part of the Mamabeibi™ application was the user browsing upon shutdown, etc.

2.2.1 Express

Express is the de facto standard JavaScript Server Side Framework. It is a complete application framework with middleware, templates and routing. Content negotiation and the MVC pattern are both supported (Md, 2020), although not necessarily useful due to the simplicity of the Mamabeibi ™ applications backend needs.

This de facto standard choice seems like a rather good choice, although perhaps the content negotiation and MVC-pattern are not as useful as they could be in the use case of Mamabeibi™.

2.2.2 Next.js

Built on the most popular frontend framework, React. Supports easy data fetching and built-in CSS support. Next.js can run on mobile, web and the desktop following the mentality build once, run everywhere. It offers best in class server rendering and exceptional SEO support with a fast start up time (Md, 2020).

Next.js seems very interesting, but looks to be mainly geared towards React usage, which conflicts with the usage of Angular as the main web application language for the Mamabeibi™ project.

2.2.3 Meteor

Can build the whole stack, front and backend. Is an isomorphic platform, sharing the same API both on the client and server-side. Can develop cross-platform solutions for web, mobile and desktop. Has an integrated JavaScript stack, meaning minimum effort usage of technologies such as the MongoDB database or the React front-end. Has its own templating engine but can be used with either React or Angular (Md, 2020).

This option seems mostly aligned with React use, similarly as with Next.js so perhaps this is not the best fit for this project.

2.2.4 Koa

Is modular, offers pluggable middleware modules. Uses modern JavaScript practices such as the async/await callbacks, which produce cleaner and more expressive code with better error handling. Is lightweight, offers a smaller core without middleware. Is a cascading middleware, and offers slightly better performance than Node.js.

This framework seems like a possibly good choice, the code seems very lightweight and straightforward – much more so than any of the other compared frameworks (Md, 2020).

2.2.5 Nuxt.js

Built upon Vue.js, offering server-side rendering and exceptional SEO support with fast start up times. Support automatic splitting of code pages, or pre-rendered pages. Highly

modular with 50 standard modules supporting many needs for Web Application Development (Md, 2020).

This framework seems like a good choice, if we would have chosen Vue.js instead of Angular for the frontend framework beforehand.

2.2.6 NestJS

Similar architecture as Angular, offers out-of-the-box Enterprise grade solution with little configuration required. Very well suited for a Microservices architecture, thanks to its modular nature. Built on TypeScript and modern JavaScript (ES6+), combining functional programming, object-oriented programming, and functional reactive programming elements. Offers code generation and scaffolding tools similarly to Angular in the form of a Command Line Tool (CLI). Considered the best among Server Side Frameworks by the author Md on medium.com (Md, 2020).

This could very well be a great choice for the Angular application being developed, as it offers similar tooling and mentality as the Angular framework does. The TypeScript could either be a hindrance or a benefit if the framework is taken into use.

2.2.7 Fastify

A framework that's minimalistic, and one of the fastest Server Side Frameworks as per their own benchmark at https://www.fastify.io/benchmarks/. Has a robust plugin architecture for easy extension. Offers an excellent developer experience, as it is minimalistic and expressive to develop with. Has CLI tools for scaffolding and code generation, as well as easy startup. Currently not as popular as many of the other frameworks (Md, 2020).

This framework may be fast and minimalistic, but will it integrate with the way an Angular application functions or fight against it.

2.2.8 Loopback

The Loopback framework is especially suited for Microservice Architecture. It's a heavyweight framework offering OpenAPI Spec driven REST API request/response creation.

The framework is built on TypeScript and offers advanced features like Components, Dependency Injection and Mixins. LoopBack as a framework offers excellent developer experience with modular, expressive and clean code. The framework can create GraphQL for any REST API as it has excellent support for GraphQL (Md, 2020).

GraphQL is likely not going be used, as the project proceeds with Angular instead of other frameworks that prefer the usages of GraphQL.

2.2.9 Hapi

The Hapi framework offers integrated Authorization and Authentication, which is the best among Node.js frameworks. It is an end-to-end, enterprise-grade framework offering minimal overhead out-of-the-box functionality. Hapi offers an over-the-top Developer Experience with a special focus on expressiveness and Code readability. It has a Modular Architecture with huge eco-system of official plugins. This leads into the conclusion, that it is easily extensible in a secure way. Unfortunately, Hapi does not support any Middleware. Instead of middleware, Hapi offers extensibility model via Plugins that puts predictability and security first (Md, 2020).

The lack of middleware may be a question that is a deal breaker in order to even start developing with Hapi, as middleware may play important parts in getting the job done.

2.2.10 Restify

An older framework for Node.js focusing on highly scalable REST API services. Used by Netflix and a few other large companies. It is a minimalistic framework with focus on Microservices and APIs. The framework has out-of-the-box Client support for String Client, HTTP Client, and Json client. Restify also has first-class support for Dtrace as it automatically creates Dtrace probes for every route/handler. There is support for Sinatra style handle chaining as well as semantic API versioning based on semver (Md, 2020).

As Netflix uses this framework, it gives the outlook into the library that they may have the right idea on how to build a scalable and multi-user friendly environment. As for Dtrace, it’s not used by me in the current NodeJS work I’ve done, so perhaps that is something to learn and see if it could be the main choice.

In the end it would seem, that the safe choice is to go with the Express framework, and this is what the thesis will be built on. Reasons for this are many, but one key answer is that the hassle of reinventing the wheel seems to not have produced better wheels than the wheel we already have in use. The choice of framework for the backend is the Express framework.

2.3 The database

The choice of the database is based on the simple fact that users of the Mamabeibi™ application should be able to save user data and retrieve it in an orderly fashion. The writer of this thesis has experience with Relational Databases so any NoSQL databases will not be considered in this case. There is a strong bias towards MySQL and PostgreSQL as well, as the writer has previous experience from her education and professional life on how to use these two systems for database management. This work will attempt to stay neutral regardless of the above presented two biased details before the comparison.

358 systems in ranking, September 2020

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

Figure 11. Database management solutions based on their popularity (db-engines.com)

Rankings globally seem to indicate a top 3 of Oracle, MySQL and PostgreSQL. MongoDB is a NoSQL database and as it is not a relational database, it will be disregarded as a popular choice for this work in particular. It would seem like those two years after the initial sample, in the year 2022 the results are very similar, as can be seen in Figure 12 below.

391 systems in ranking, April 2022

Rank			DBMS	Database Model	Score		
Apr 2022	Mar 2022	Apr 2021			Apr 2022	Mar 2022	Apr 2021
1.	1.	1.	Oracle	Relational, Multi-model	1254.82	+3.50	-20.10
2.	2.	2.	MySQL	Relational, Multi-model	1204.16	+5.93	-16.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	938.46	+4.67	-69.51
4.	4.	4.	PostgreSQL	Relational, Multi-model	614.46	-2.47	+60.94
5.	5.	5.	MongoDB	Document, Multi-model	483.38	-2.28	+13.41
6.	6.	↑ 7.	Redis	Key-value, Multi-model	177.61	+0.85	+21.72
7.	↑ 8.	↑ 8.	Elasticsearch	Search engine, Multi-model	160.83	+0.89	+8.66
8.	↓ 7.	↓ 6.	IBM Db2	Relational, Multi-model	160.46	-1.69	+2.68
9.	9.	↑ 10.	Microsoft Access	Relational	142.78	+7.36	+26.06
10.	10.	↓ 9.	SQLite	Relational	132.80	+0.62	+7.74

Figure 12. Same comparison in 2022 as above (db-engines.com)

When Figure 11 and Figure 12 are combined, the most popular choice for databases is Oracle, followed by MySQL and PostgreSQL. This points to the logical choice to be MySQL or PostgreSQL because the author has experience previously with these two technologies. Another consideration is the technology choice of the backend, which is constrained by NodeJS. This is a JavaScript runtime technology for the backend, which needs to send messages to the database that arrive to it from the frontend. Appendix 1 of this work provides a more detailed a view of the differences of these three technologies, and confirms yet another positive point towards choosing MySQL or PostgreSQL, that JavaScript (NodeJS) is supported for both MySQL and PostgreSQL database systems.

There is another comparative analysis that the writer feels smart to perform on the choice of database technology for this project. As the full stack mobile application being created, uses JavaScript, this means that with it also comes the natural usage of the JSON data type. It is worth it to examine how the support of JSON as a datatype is working with both MySQL and PostgreSQL to make the final choice between the two.

PostgreSQL	MySQL
PostgreSQL supports JSON and other NoSQL features like native XML support and key-value pairs with HSTORE. It also supports indexing JSON data for faster access.	MySQL has JSON data type support but no other NoSQL feature. It does not support indexing for JSON.

Figure 13. PostgreSQL vs MySQL feature comparison, JSON (2ndquadrant.com, 2022)

According to Figure 13 above, MySQL is lacking with JSON data type indexing and as the application will be using JSON, this in combination with other details seen in Appendix 1 point to making the decision of going forwards with PostgreSQL. In combination with the author already knowing how to use the technology, these two reasons are why PostgreSQL is chosen as the database technology for this project. PostgreSQL has been an open-source system that has been on the market for a long time, making it a safe choice to go forwards with, without any commercial ties or vendor lock in. Also, MySQL has a proprietary language to program user functions, PostgreSQL follows the standard there.

2.4 Where will the finished product be running on and how

The choice of the frontend application deployment is dictated based on the target environment of this thesis, which are the two dominant mobile application platforms as of 2021. The iOS operating system is a UNIX variant, and the Android system is based on a modified Linux kernel, which are both the operating systems of the Apple and Google mobile marketplace devices. The backend allows for more choices for this work, as it is external to the mobile device, just like the database is. The development work is running on a separate web server on the local development computer that the author uses to develop the full stack mobile application on, which will be discussed later in this work.

2.4.1 The frontend deployment environment

The frontend of the application will be running in its own context on a higher level away from the operating systems low level Linux systems, so this work will leave that to be abstracted away to the chosen frontend framework. Rather what will be used is use modern web application development techniques to create the user interface of the application for the target environment.

2.4.2 The backend deployment environment

The target environment for the backend will be a Linux server, due to the cost effectiveness of not having to buy an operating system license, as well as the reliability and having the source code available for changes if these were to ever come to the mind of the developer.

Some of the best assets, when comparing to a commercially available system, of a Linux system are its price, the freedom it gives you, the reliability of the system. There is an argument to be made for scalability being one of its greatest assets. The initial price of Linux is free, which most people know, but mostly what people talk about in terms of Linux's affordability, they usually think of its total cost, which includes the ability to reuse all of the code as you choose, includes no (or low) cost of licensing fees, capability to use inexpensive hardware and free add-on applications that are compatible.

Reliability in terms of Linux is usually seen as it being more reliable than desktop systems and that there exists a general consensus that Linux is comparable in reliability to many commercial UNIX systems. For a web server this is especially true, as you do not have to reboot your system every time you change something, unless you've replaced the kernel itself.

The realm of safety is by extension touched by the reliability of Linux, as opposed to Windows users who are often plagued by malware and viruses. Deployments of larger scale to Linux do not need anti-virus software installations – a situation which you would never allow in a corporate setting with Windows. Anti-virus software on Linux is usually installed to scan e-mail messages or files for Windows viruses, to help the users of Windows.

Linux is a culture encouraging interoperability, because we can get the source code, we can change any part of the Linux systems along with any open-source software that comes with it unlike many self-contained commercial products which are not built in pieces that are meant to interact with other pieces, like open-source software is. We are left with the choice to choose the pieces we want by mixing and matching components to suit our needs. (Negus, page 10, 2008).

2.4.3 The selected web server on the operating system

First, there will be a small look into the history of web servers and a short overview of what a web server is and where it places itself in the operating system, as seen in Figure 14 below. After this look into the history, there will be a comparison of modern web servers and a choice will be made on which to use to accomplish this works objectives.

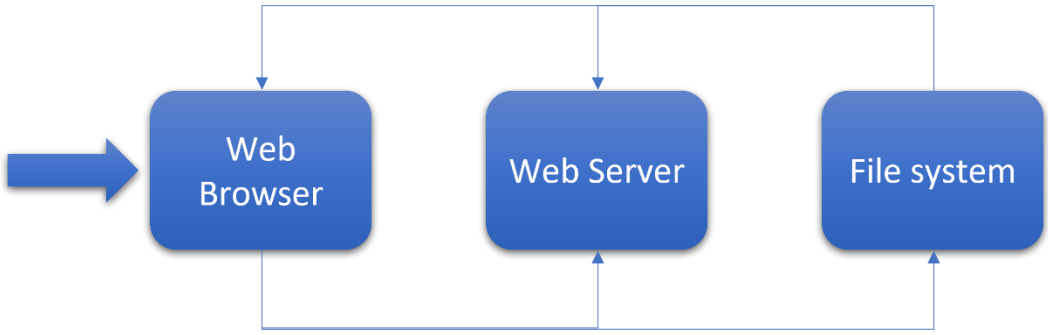

Figure 14. Basic functionality of a web server

Mr. Tim Berners-Lee at the European Center for Particle Physics (CERN), began the project that is known as the World Wide Web. The goal originally was to provide a single consistent interface for geographically spread scientists and researchers who needed access to a variety of formats of information. This idea to access data such as text, images, video, sounds and binary files was where the concept of using one client to access data came from, the Web Browser. The Web server on the other hand usually has a simpler job. That is to accept HyperText Transfer Protocol (HTTP) requests and sending responses to the client. The job may however become much more complex (just as the server can also), running functions such as:

- Logging any successful accesses, errors and failures.

- Document parsing (changing values for conditional fields in documents) before sending back to the client.
- Creating a Common Gateway Interface (CGI) script or a custom API (Application Programming Interface) to evaluate the contents of a submitted form or accessing a database or presenting a dynamically created document.
- Managing access control such as host name/IP address restrictions, or file permission-based access control, or username/password pairs.

The Apache Web server is based on a free server from the National Cener of Supercomputing Applications (NCSA), which was originally called HTTPd. During it's time, HTTPd was the only, and the first Web server on the Internet. The development lagged behind unfortunately with the needs of the Webmasters and many security problems had been discovered. Multiple Webmasters independently applied their own features and fixes to the source code originally from the NCSA.

A group of these developers later in 1995 created a new project based on this source code base called Apache – and that has since then now become the Apache Software Foundation (www.apache.org) where the code has largely been rewritten to create a stable multiplatform web server daemon. Apache is not the only web server available, but it is one of the most commonly used with Linux. (Negus, pages 834 – 835, 2008).

Today's web servers are quite more sophisticated than the early 1990s static file system content servers. It was the initial vision of Mr. Tim Berners-Lee is where all this media richness grew from, where today's web servers support multiple capabilities from user authentication, virtual hosting, server-side scripting to generate dynamic content, etc. We can now balance our check book, make appointments, share documents, all in real time using "web apps", and as a result the concept of what a "web server" is has become fuzzy. Handling requests to a web server is no longer a matter of "send back the content of this file" but it now can involve routing the request to the web application, which can then determine where the data comes from, be it a database, a stock ticker, or a file. (Clifton, page 9-11, 2015)

The question which web server was selected comes down to these two reviews you can see in Table 2 and Figure 15 further in this chapter. The observations of which technology is the better overall choice are interesting and not that apparent when choosing some web server to host or develop a full stack web application on – or at least the frontend of it. It comes down to questions of scaling and dynamic content, where the future of the application should be considered as well instead of just the ease of use at the start of development.

Table 2. Apache and NGINX webserver comparison (Javatpoint, 2021)

Apache	NGINX
Apache runs on all Unix like systems such as Linux, BSD, etc. as well as completely supports Windows.	Nginx runs on modern Unix like systems; however, it has limited support for Windows.
Apache uses a multi-threaded approach to process client requests.	Nginx follows an event-driven approach to serve client requests.
Apache cannot handle multiple requests concurrently with heavy web traffic.	Nginx can handle multiple client requests concurrently and efficiently with limited hardware resources.
Apache processes dynamic content within the web server itself.	Nginx can't process dynamic content natively.
Apache is designed to be a web server.	Nginx is both a web server and a proxy server.
Modules are dynamically loaded or unloaded, making it more flexible.	Since modules cannot be loaded dynamically, they must be compiled within the core software itself.
A single thread can only process one connection.	A single thread can handle multiple connections.
The performance of Apache for static content is lower than Nginx.	Nginx can simultaneously run thousands of connections of static content two times faster than Apache and uses little less memory.

The comparison in Table 2 we can see that Apache seems to have perhaps more flexibility in what modules (or plugins) to choose from, but the Nginx-side needs to be recompiled in order to do any meaningful changes into the core server itself.

Overall, it seems at least based on table 2, that Nginx can handle more connections and do handle more clients efficiently with lower resources. It seems to also function as a proxy server, although Apache also has reverse proxy functionality through modules, so this is a somewhat confusing point. When focusing on the handling of more connections, further proof for the argument is desirable so we can verify whether Nginx really outperforms Apache with concurrent connection handling or not. For this, the website Dreamhost.com ran a benchmark study on the performance of these two websites and the results that were found can be seen in Figure 12 below.

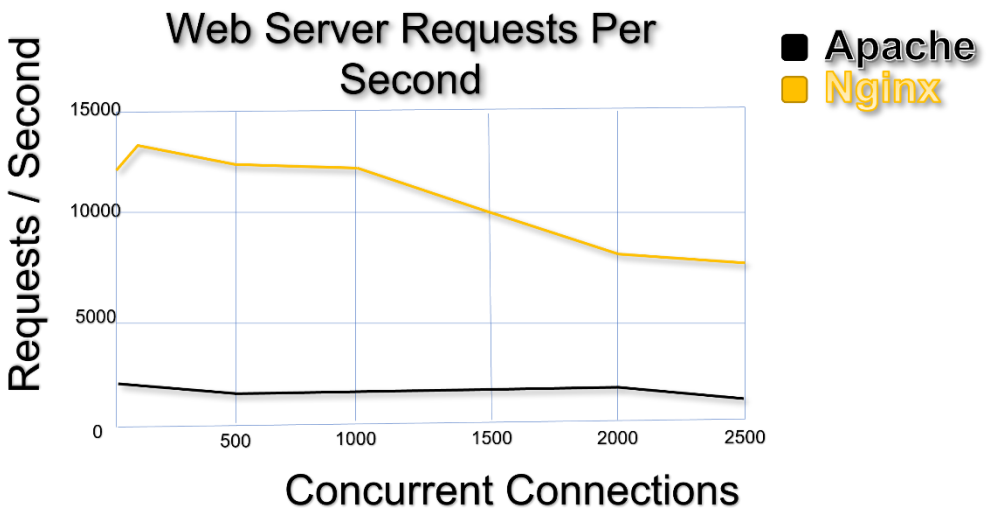

Figure 15. Nginx handling requests per second better than Apache

The winner of the requests per second test made by Dreamhost in the year 2016 was commented on (Kunda, Chihana et. Al., 2017, page 45) in a web server performance literary review, that in raw number of requests, Nginx clearly dominates Apache as a web server. This is the result of our review that indicates that Nginx should be a good choice if the application developed and hosted would become popular and need to handle multiple concurrent connections and still be able to function properly.

To further investigate our choice, a review of the memory usage of Apache and Nginx was investigated in figure 16, seen below. This tells of interesting details in performance.

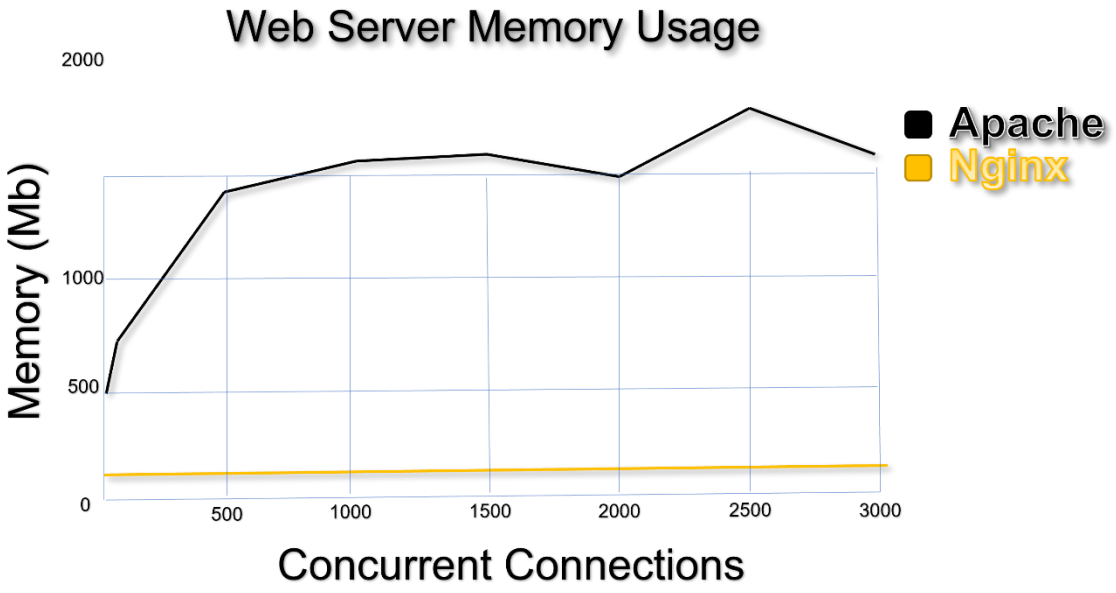

Figure 16. Apache seems to use significantly more memory than Nginx.

The memory usage of Apache tells us that when testing with a 5k PNG file locally from a VPS (Virtal Private Serve) to take into consideration potential variation in network conditions, making 25 000 requests to measure different levels of usage, Dreamhost test results show yet again that Nginx comes out as the clear leader in the test. The more connections come in, the more processes Apache spawns to handle them, which leads to growing memory usage quick. (Kunda, Chihana et. Al., page 44. 2017)

For good source criticism, a third source was added in order to investigate the seemingly obvious choice of Nginx, which had the interesting notion that both the web servers Apache and Nginx are flexible, capable and powerful. Both being competitive in all areas, with their latest versions. (Hari, 2021).

Another interesting notion was that the third source from Hackr.io revealed a use case, where Nginx was used to complement Apache by functioning as Apache's proxy server, offloading slow HTTP connections, serving static files and caching content, in order for the Apache server to run in a secure and safe environment, the application code. (Hari, 2021).

This further makes our choice sound reasonable, that the work can proceed with the choice of Nginx as the web server, as it is fully capable of delivering with its latest version most if not all the same services as an Apache server, but still using less memory than Apache making more room for development tools and other tools needed on the development machine that will be used to complete this full stack application.

Now that all parts have been chosen, the following chapters will detail the empirical results of the work and provide conclusions and discussion about the objectives of this work and how they were reached or not reached using the selections mentioned in this chapter.

3 Empirical part

The target of this thesis was to create a full stack mobile application, using a single shared codebase with the objective of publishing the finished product on the Google and Apple marketplaces. Another target was to describe all the parts that were required to implement a full stack application, for which in this thesis a decision was made to implement the frontend with JavaScript, HTML5 and CSS.

There were multiple problems in setting up the database and the backend on the local system so this is one of the reasons in the frontend a decision was made to put the database and backend of the application into the same frontend code. This also saved costs in setting up a server in the cloud or on a virtual machine somewhere on the Internet just for the backend and the database. This was achieved using JSON and making JavaScript calls to the JSON object to get the necessary data from the frontend using AJAX.

Asynchronous JavaScript with XML is referred to as AJAX. Web pages can change their content using AJAX without requiring users to reload the page. (Domantas, 2022).

The programming started with setup of the development environment. I used my Windows 10 computer system and Visual Studio Code to do most of the work and this is where I started. In the beginning I made a git repository by using the *git init* command to make a new repository to version control my work. This was to make sure that all my work would not get lost while I am making the app.

Another choice that needed to be made during development was the place where to save the code of the work if the local system would crash or fail, so there would be backups of the code and work. GitHub.com was selected as the hosting platform for the code during development and it turned out to be a good choice along the years as the platform only grew in popularity and was a very good tool in supporting the journey of finishing the app. The company Zodiac Fox Oy provided for a private GitHub repository to hold the code in for commercial development.

Overall the setup was rather easy and straightforward. I just had to go online to download everything that was needed. After this user interface of the application started to be worked on first. The user interface at the beginning was built to be mobile responsive so it works on mobile devices first, but also for desktop computers. The JavaScript/CSS library to help with this was Bootstrap.

3.1 Building of the frontend

The frontend building was rather fun to build as I already had experience with HTML5, CSS and JavaScript from previous projects. I started the project with Microsoft Visual Studio Code and built a system I had locally installed previously when installing the backend. The UI was built then from the ground up using basic HTML5, Bootstrap and jQuery. Hotloading was used with a plugin from Visual Studio Code called Live Server made by Mr. Dey Ritwick as a free accessory to “hotload” HTML files and serve them on the local computer for a fast way to see your HTML, CSS and JS changes immediately on the browser.

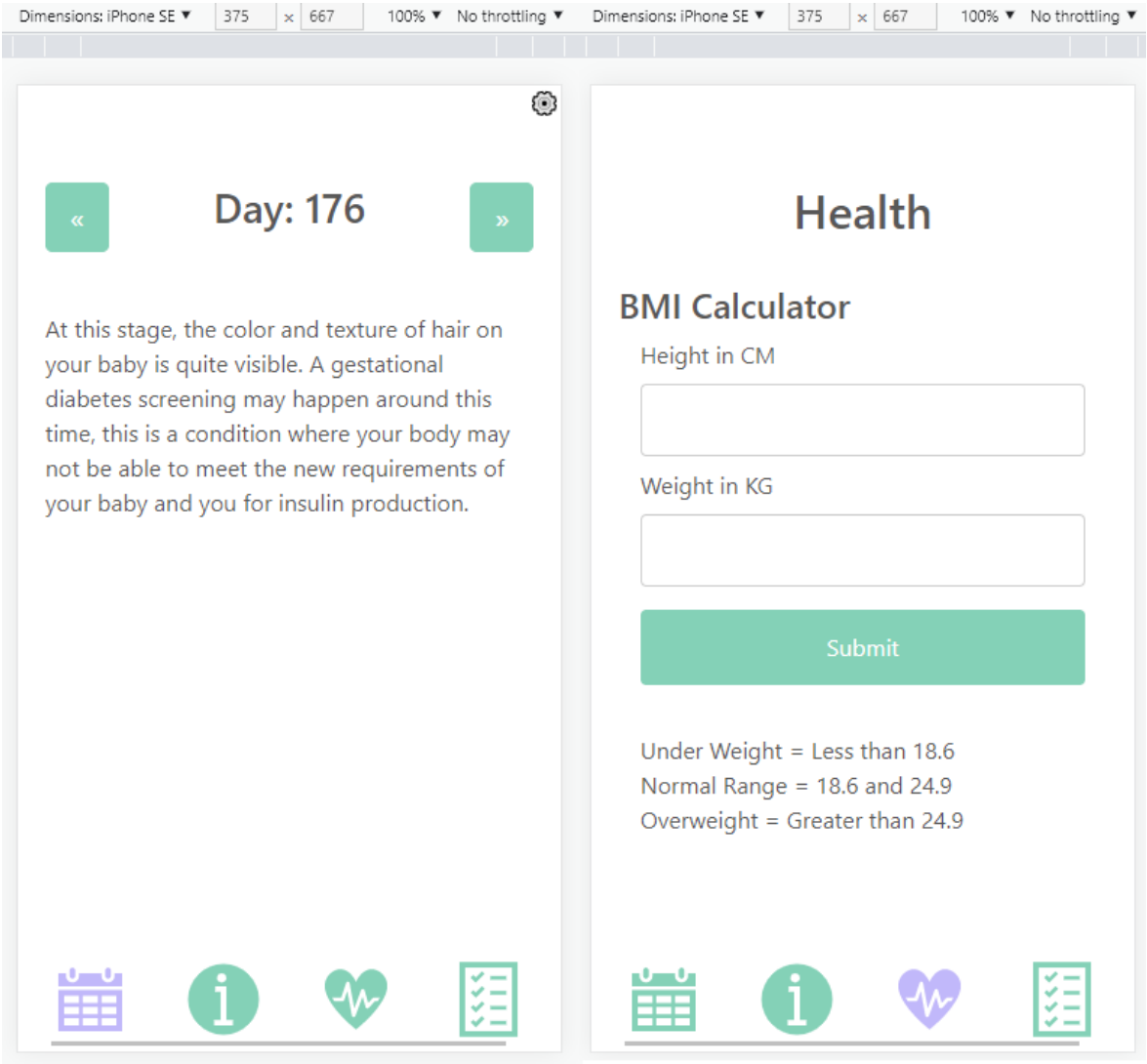

Figure 17. The Health and Diary parts of the app

The application worked great in responsive mode after programming for few days, but there was a lot of work to be done in order to get all the data into the application and to form the structure of the application. Figure 18 below shows how the structure of the application was created after publishing to the Google and Apple stores.

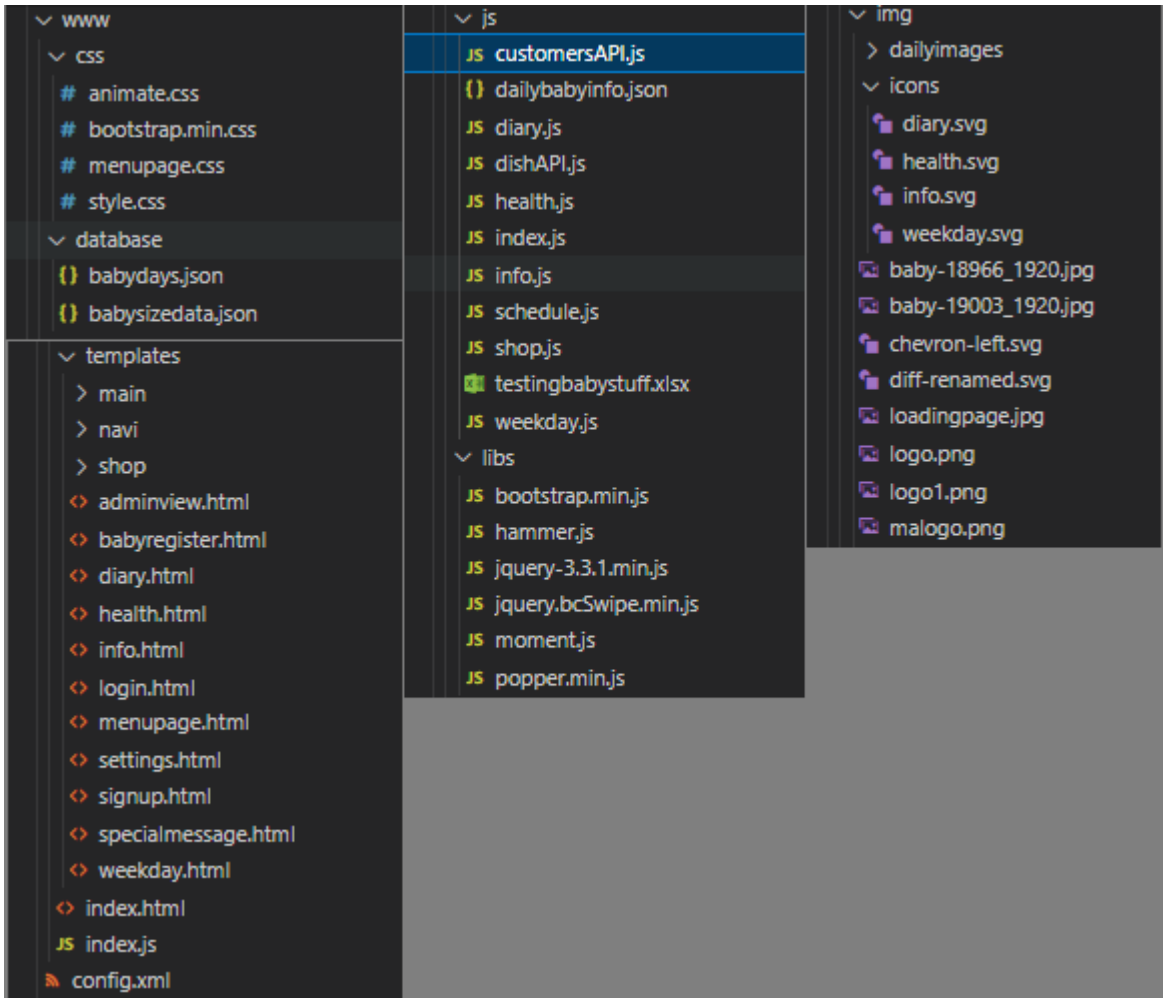

Figure 18. The structure of the Mamabeibi™ mobile application.

A folder structure under a folder of www/ was made, where the css/ folder had all the Cascading Style Sheets. The database/ folder had the data of the application as JSON objects for getting later with HTTP AJAX GET calls from the frontend. The templates/ folder had HTML files that were loaded dynamically with the index.js utility functions as seen below in Figure 19. There was also the libs/ folder for external libraries such as jquery and bootstrap, or the hammer library for mobile phone swiping gesture support. The img/ folder was for all images in png, jpg and svg file formats for use within the application menu buttons, background images, etc.

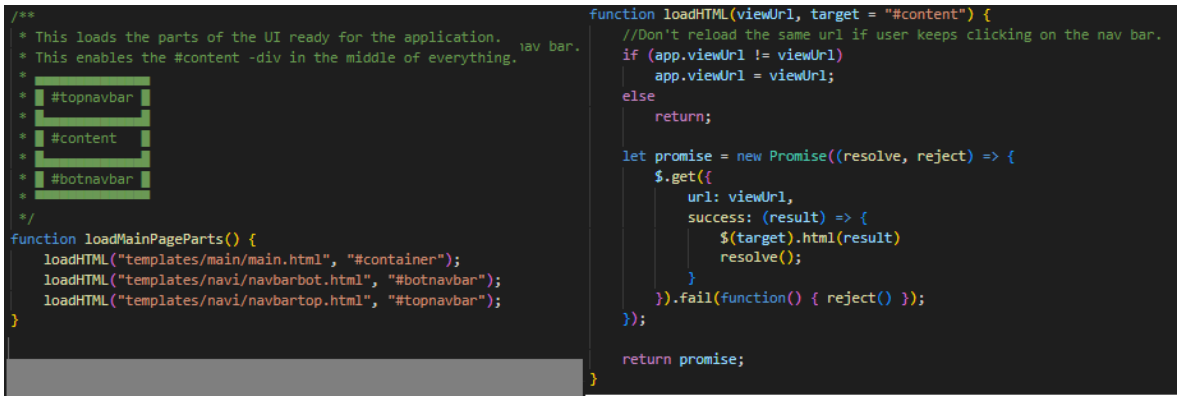

Figure 19. Dynamic loading of page parts from html files.

The dynamic loading worked with the document ready functionality of jQuery as seen below in Figure 20. After everything was loaded into memory of the browser, the application initialized itself using the browsers Local Storage to check if there was a saved date that the user was last checking as seen in Figure 21.

```
//
// Starting point of the application. Loads JS and HTML dynamically.
//
var app = {};

/**
 * This is called when the JavaScript and HTML have been loaded into the browsers memory.
 */
$(document).ready(() => {
```

Figure 20. The application startup used jQuery document ready.

```
if ( window.localStorage.getItem('currentWeekday') !== null ) {
    loadMainPageParts();
    goToWeekday(new Date().getTime());
} else {
    loadHTML("templates/menupage.html", "#container");
}
```

Figure 21. The first application code after startup.

The whole application was made of a very simple index.html file as seen on Figure 22.

```
<!DOCTYPE HTML>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1, maximum-scale=1, shrink-to-fit=no">
    <!-- Styles  -->
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <link rel="stylesheet" href="css/menupage.css">
    <link rel="stylesheet" href="css/style.css">
    <link rel="stylesheet" href="css/animate.css">
    <!-- Helper libraries -->
    <script src="libs/jquery-3.3.1.min.js"></script>
    <script src="libs/jquery.bcSwipe.min.js"></script>
    <script src="libs/popper.min.js"></script>
    <script src="libs/bootstrap.min.js"></script>
    <script src="libs/hammer.js"></script> <!-- For swiping left and right, see hammerjs.github.io -->
    <!-- Controllers -->
    <script src="js/weekday.js"></script>
    <script src="js/info.js"></script>
    <script src="js/health.js"></script>
    <script src="js/diary.js"></script>
    <!-- Main controller -->
    <script src="index.js"></script>
    <!-- Mobile app support -->
    <script type="text/javascript" src="cordova.js"></script>
    <title>MamaBeibi</title>
</head>

<body>

    <div class="container" id="container"></div>

</body>

</html>
```

Figure 22. The heart of the Mamabeibi™ mobile application, the index.html file.

Each of the sections of the app as seen in Figure 17 had their own JavaScript logic files loaded called weekday.js, info.js, health.js and diary.js. This way the concept of Separation of Concerns could be used for good software design instead of making a too big index.js file that cannot be maintained later after the application business logic grows.

Separation of concerns is a design principle used in computer science to divide a computer program into sections, each of which addresses a different concern. (Tamerlan, 2021).

With all this combined, the application was working great and I was able to proceed to publish and test it on Google and Apple marketplaces and their devices.

3.2 Building of the backend and the database

The backend was built first built using Express and NodeJS on my local computer, I had to first install NodeJS but it was not a complicated process to finish. After completing the installation, I started with npm install express and installed required packages more from there. Then I started to build the first backend endpoints to communicate with the frontend of the application. As seen in Figure 23. below.

```
const express = require('express')
const pgp = require('pg-promise')( /* options */ )
const app = express()
const port = 3000

const db = pgp('postgres://postgres:[redacted]@localhost:5432/mamabeibi')

db.one('SELECT * FROM BabyInfo')
    .then(function(data) {
        console.log('DATA:', data)
    })
    .catch(function(error) {
        console.log('ERROR:', error)
    })

app.get('/', (req, res) => res.send('Hello World!'))
```

Figure 23. The point where NodeJS coding stopped.

As seen in I started with an HTTP GET /helloworld as my first test RESTful API endpoint. In the end due to the difficulties of setting up PostgreSQL locally and its special users required for it to work with the backend, the backend was simplified into a local storage implementation of all the data during development work. This saved time and allowed for prototyping to proceed faster.

A big need to update the main data of the application did not seem to be as necessary in real time as for other applications, this is why the database could be embedded into the application. The NodeJS integration was left in the code that can use a PostgreSQL database, but it was not finally integrated in favour of the HTML5 Local Storage options speed of making the application in the end.

The data model was built first on paper, before it was built into application code and SQL table DDL format. After this, PostgreSQL was installed, and the data was input using DBeaver software into a local running instance of PostgreSQL for testing the basic SELECT-statements to see if the data model worked correctly.

First, I was running some tests with the NodeJS backend to try to get the data from the local database, but the NodeJS application seemed to lack the rights to use the database locally due to some setup error. I tried fighting with installing a proper user into the database and changing the NodeJS application with some user rights, but for some reason the connection between the NodeJS application and the database on the Linux server did not work out.

Then already came time to input all the data into the database regarding the Mamabeibi ™ apps details but it was decided that the applications data would fit into a local storage database to favour for fast implementation and loading speeds without the need for a full external database at this point of prototyping and making the first MVP version of the Mamabeibi ™ application so I decided to go forwards with this direction.

Here is a sample of the datamodel creation script for PostgreSQL from the early prototype:

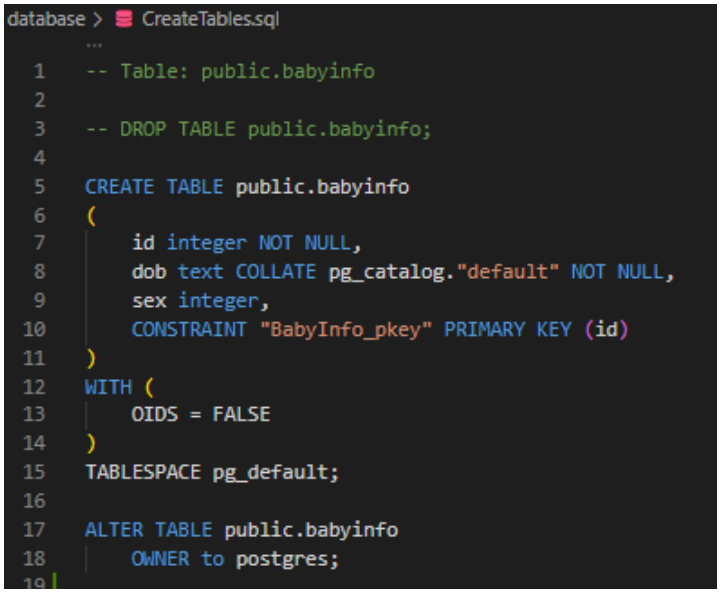

Figure 24. Creating the Mamabeibi™ database table foundations.

As seen on the Figure 24 above, the actual SQL structure of the backend was made for the maternity application Mamabeibi™ to determine when the babys dob (Date of Birth) was and what was its sex (male or female) in order to show the right days info. But as seen on Figure 25. below, the database was greatly simplified into a simple JSON file for each day as seen below:

Figure 25. The days from pregnancy to delivering a baby, data structure.

The audience this application is intended for is the largest country in the world, China, and all English-speaking countries as can be seen from the FI, CHI, ENG, description fields under each day. This means the market reach potential for this application could have been very big, if there was enough marketing budget or funding for it.

3.3 Combining everything on an external Linux Nginx web server

After completing all the above parts required for a full stack application, the parts were tried to be ran on a server online at OVH hosting where everything was combined (backend and database).

Fortunately, it was quickly realized that this might not be needed for the application to be launched into the AppStore and Play store as accessing an external server requires extra permissions to be enabled for the application at least on the AppStore side.

The operating system on OVH was chosen to be a Debian Linux and only had some minor configurations done to it get nginx running on the server. After this the process of uploading the app to the serve an installation of NodeJS on the server started. This went quite smoothly with *sudo apt-get install nodejs* but after this, the installation of the PostgreSQL database did not go so smoothly.

There were problems with the user access to the PostgreSQL database from the NodeJS app code, so I again decided that the better way to proceed would be to put the database into the main application as a JSON structured JavaScript object in LocalStorage.

There was some testing done with the Chrome Internet Browser after everything was half setup but in the end the setup of all the pieces was unnecessary because a simple JSON structure of all the data the application needs was able to be put into the application itself for the first version to be published.

3.4 Summary

In summary, all of the steps taken before going on this adventure of developing a full stack mobile application led me to find an efficient way to implement this project and I feel like the ending result is a success. There may have been better ways to do this but doing things this way completed the job that was to publish the application both on the Google and Apple marketplaces with one single shared codebase. I was able to learn a lot while testing NodeJS and PostgreSQL with the full stack approach. This was also a happy surprise that I could publish the application just with the frontend built using Apache Cordova.

4 Discussion

The application implementation was successfully published into the Google and Apple marketplaces. The conclusion is that a shared codebase does save time in development and publishing with shared data types such as JSON defined types which JavaScript can use natively. The results of this thesis can be questioned for not implementing fully the NodeJS and PostgreSQL backend and database work for being complete.

I would argue that the Mamabeibi™ application that was made with this thesis is a full stack application. It just all works on top of the Apache Cordova platform as a localhost HTML JavaScript and CSS server instead of an external Internet based NodeJS server of some kind. It still has a database (JSON format files loaded with HTTP GET), backend calls (The HTTP GET calls) and a frontend (The pure HTML, JavaScript and CSS files).

Overall, I am very happy with the results of this thesis work and the many comparisons that were made between multiple technologies on what could be the best approach to make an application given the low amount of resources that were available for me.

Further research could be done on more frameworks (Electron, Capacitor, and a few others) that could have been explored in the year 2023 but I think given the time limitations for this work comparing the top 4 frameworks was enough for finding a good enough solution to complete the job with.

In my own learning, I have already started to learn more of the React JavaScript XML language and use it in my daily work now. Maybe in the future I could also learn the C# language or the DART language to have better options available for making mobile applications with other tools. It could be better to learn one tool like React-Native first and focus more on the successful business cases the mobile full stack applications I make instead of just the technology parts of the work.

Thank you for reading my thesis on developing a full stack mobile application for the Google and Apple marketplaces. 😊

References

Arancio, S. 2021. ReactJS: A brief history. URL: https://medium.com/@sjarancio/reactjs-a-brief-history-3c1e969a477f Accessed: 7.1.2023.

Bahrynovska, T. 2022. What Is a Tech Stack: A Package of Means for Achieving Your Goals URL: https://forbytes.com/blog/what-is-a-tech-stack/ Accessed: 18.3.2023

Broberg, M. 2019. Getting started with Git: Terminology 101 URL: https://open-source.com/article/19/2/git-terminology Accessed: 18.3.2023

Callaham 2017. From Android Market to Google Play: a brief history of the Play Store. URL: https://www.androidauthority.com/android-market-google-play-history-754989/ Accessed: 20.12.2019

Chandra, A. 2020. What Are Full Form of APK And IPA [Updated 2021]. URL: https://thebigbrains.com/full-form-of-apk-and-ipa/ Accessed: 26.1.2023

Clifton 2015. Web Servers Succinctly. URL: https://www.syncfusion.com/succinctly-free-ebooks/confirmation/webservers Accessed: 19.9.2021

Codeacademy Team. 2023. What Is an IDE? URL: https://www.codecademy.com/article/what-is-an-ide Accessed: 10.4.2023

Domantas, G. 2022. What Is AJAX and How Does It Work? URL: https://www.hostinger.com/tutorials/what-is-ajax Accessed: 11.4.2023

Dudley, D. 2018 Helsinki. The evolution of mobile phones: 1973 to 2019. URL: https://flauntdigital.com/blog/evolution-mobile-phones/ Accessed: 30.3.2020.

Flutter.dev. 2023. Flutter Architectural Overview. URL: https://docs.flutter.dev/resources/architectural-overview Accessed: 9.4.2023

Flutter.dev FAQ. 2023. Why did Flutter choose to use Dart? URL: https://docs.flutter.dev/resources/faq Accessed: 9.4.2023

Furman, R. 2023. A Definite Overview on Mobile App Technology Stack. URL: https://www.uptech.team/blog/mobile-app-technology-stack Accessed: 8.1.2023

Georgiou, M. 2023. Xamarin vs. React Native: Which One Should You Choose for Mobile App Development in 2023? URL: https://imaginovation.net/blog/xamarin-vs-react-native-mobile-app-development-2021/ Accessed: 8.1.2023

Goodman, D. 2007. The JavaScript Bible. Wiley Publishing, Inc. Indianapolis, Indiana.

Ghauri, P. & Grønhaug, K. 2010. Research methods in business studies. 4th ed. Pearson Education. Harlow.

Griffith, C. 2023. What is Apache Cordova Framework and What is the Difference from PhoneGap? URL: https://ionic.io/resources/articles/what-is-apache-cordova Accessed: 10.4.2023

Hámori, F. 2022. The History of React.js on a Timeline. URL: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/ Accessed: 8.1.2023

Hardik, B. 2023. A Brief History of Flutter. URL: https://blup.in/blog/a-brief-history-of-flutter Accessed: 9.4.2023

Hari, S. 2021. NGINX vs Apache: Head to Head Comparison. URL: https://hackr.io/blog/nginx-vs-apache Accessed: 15.10.2021

Herembourg, K. 2022. 8 Most Popular Mobile App Development Frameworks in 2023. URL: https://www.purchasely.com/blog/mobile-app-frameworks Accessed: 1.4.2023

Jasnowska, J. 2019. Why you should migrate from Ionic, Cordova, or PhoneGap to React Native (Updated). URL: https://www.netguru.com/blog/react-native-comparison Accessed: 1.4.2023

JetRuby 2017. Brief History of Mobile Apps. URL: https://expertise.jetruby.com/brief-history-of-mobile-apps Accessed: 20.12.2019

Kunda, D., Chihana S., Sinyinda M. Web Serve Performance of Apache and Nginx: A Systematic Literature Review. URL: https://www.researchgate.net/publication/329118749_Web_Server_Performance_of_Apache_and_Nginx_A_Systematic_Literature_Review Accessed 15.10.2021

Nath, B. 2022. Introduction to Xamarin: Complete Guide and Learning Resources URL: https://geekflare.com/xamarin-introduction/ Accessed: 8.1.2023
Negus, C. 2008. Fedora 8® and Red Hat Enterprise Linux® Bible. Wiley Publishing, Inc. Indianapolis, Indiana.

Md, K. 2020. Top 10 JavaScript Frameworks for Server-Side Development in 2020. URL: https://medium.com/javascript-in-plain-english/top-10-javascript-frameworks-for-server-side-development-in-2020-6d265016c02 Accessed: 25.5.2020

Rajput Mehul 2015. Tech.co. Tracing the History and Evolution of Mobile Apps. URL: https://tech.co/news/mobile-app-history-evolution-2015-11 Accessed: 15.3.2020

Ramón, J. 2019. Everything you need to know about Node.js URL: https://dev.to/jorge_rockr/everything-you-need-to-know-about-node-js-lnc Accessed: 15.3.2020

Sachan, A. 2022. Top 6 Cross-Platform App Development Frameworks in 2023. URL: https://www.mobulous.com/blog/top-6-cross-platform-app-development-frameworks-in-2023/ Accessed: 7.1.2023

Schae, J. 2020. A RealWorld Comparison of Front-End Frameworks 2020. URL: https://medium.com/dailyjs/a-realworld-comparison-of-front-end-frameworks-2020-4e50655fe4c1 Accessed 16.6.2020

Sharma, S. 2020. The Growth of Flutter Development— 3 Years After The Birth of Alpha URL: https://medium.flutterdevs.com/the-growth-of-flutter-development-3years-after-the-birth-of-alpha-78baee809dff Accessed: 8.1.2023

Sheldon, R. 2023. What is a codebase (code base)? URL: https://www.techtarget.com/whatis/definition/codebase-code-base Accessed: 17.3.2023.

Shropshire, J. & Landry, J. & Presley, S. 2018. TOWARDS A CONSENSUS DEFINITION OF FULL-STACK DEVELOPMENT. URL:

https://aisel.aisnet.org/sais2018/17?utm_source=aisel.aisnet.org%2Fsais2018%2F17&utm_medium=PDF&utm_campaign=PDFCoverPages Accessed: 30.3.2020

Sirgur, B. 2023. React Functional Components: In-Depth Guide URL: https://www.knowledgehut.com/blog/web-development/react-functional-components#conclusion Accessed: 2.4.2023

Smith, Albert. 2023. Top Six Frontend Frameworks for Web Development in 2023. URL: https://medium.com/front-end-weekly/top-six-frontend-frameworks-for-web-development-caeaa6b1dcc4 Accessed: 25.3.2023

Snyder Hannah 2019. Literature review as a research methodology: An overview and guidelines. URL: https://www.sciencedirect.com/science/article/pii/S0148296319304564 Accessed: 24.12.2019

Stack Overflow. 2022. Developer Survey 2022. URL: https://survey.stackoverflow.co/2022/#most-popular-technologies-language Accessed: 25.3.2023

State of JS. 2023. Front-End Framworks: Ratios Over Time. Usage. URL: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/ Accessed: 9.4.2023
Sviatoslav, A. 2020. The Best JS Frameworks for Front End. URL: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end Accessed: 15.3.2020

Tamerlan, G. 2021. Separation of Concerns The Simple Way. URL: https://dev.to/tamerlang/separation-of-concerns-the-simple-way-4jp2 Accessed: 12.4.2023

Tanenbaum, A. 2001. Modern Operating Systems. Prentice Hall, Upper Saddle River, New Jersey.

Tranfield, D. & Denyer, D. & Smart, P. 2003. Towards a Methodology for Developing Evidence-Informed Management Knowledge by Means of Systematic Review*
URL: https://www.cebma.org/wp-content/uploads/Tranfield-et-al-Towards-a-Methodology-for-Developing-Evidence-Informed-Management.pdf Accessed: 24.12.2019

PostgreSQL vs MySQL. 2ndQuadrant. 2022. URL: https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/ Accessed: 12.4.2022

Appendices

Appendix 1. Comparison of MySQL, Oracle and PostgreSQL database technologies

Source: https://db-engines.com/en/system/MySQL%3BOracle%3BPostgreSQL Accessed: 12.4.2022

Name	MySQL X	Oracle X	PostgreSQL X
Description	Widely used open source RDBMS	Widely used RDBMS	Widely used open source RDBMS
Primary database model	Relational DBMS	Relational DBMS	Relational DBMS
Secondary database models	Document store Spatial DBMS	Document store Graph DBMS RDF store Spatial DBMS	Document store Spatial DBMS
DB-Engines Ranking Trend Chart	Score 1204.16 Rank #2 Overall #2 Relational DBMS	Score 1254.82 Rank #1 Overall #1 Relational DBMS	Score 614.46 Rank #4 Overall #4 Relational DBMS
Website	www.mysql.com	www.oracle.com/database	www.postgresql.org
Technical documentation	dev.mysql.com/doc	docs.oracle.com/en/database	www.postgresql.org/docs
Developer	Oracle	Oracle	PostgreSQL Global Development Group
Initial release	1995	1980	1989
Current release	8.0.28, January 2022	19c, February 2019	14.2, February 2022
License	Open Source	commercial	Open Source
Cloud-based only	no	no	no
DBaaS offerings (sponsored links)	ScaleGrid for MySQL: Fully managed MySQL hosting on AWS, Azure and DigitalOcean with high availability and SSH access on the #1 multi-cloud DBaaS.		ScaleGrid for PostgreSQL: Fully managed PostgreSQL hosting on AWS, Azure and DigitalOcean with high availability and SSH access on the #1 multi-cloud DBaaS.
Implementation language	C and C++	C and C++	C
Server operating systems	FreeBSD Linux OS X Solaris Windows	AIX HP-UX Linux OS X Solaris Windows z/OS	FreeBSD HP-UX Linux NetBSD OpenBSD OS X Solaris Unix Windows
Data scheme	yes	yes	yes
Typing	yes	yes	yes
XML support	yes	yes	yes
Secondary indexes	yes	yes	yes
SQL	yes	yes	yes
APIs and other access methods	ADO.NET JDBC ODBC Proprietary native API	JDBC ODBC ODP.NET Oracle Call Interface (OCI)	ADO.NET JDBC native C library ODBC streaming API for large objects
Supported programming languages	Ada C C# C++ D Delphi Eiffel Erlang Haskell Java JavaScript (Node.js) Objective-C OCaml Perl PHP Python Ruby Scheme Tcl	C C# C++ Clojure Cobol Delphi Eiffel Erlang Fortran Groovy Haskell Java JavaScript Lisp Objective C OCaml Perl PHP Python R Ruby Scala Tcl Visual Basic	.Net C C++ Delphi Java JavaScript (Node.js) Perl PHP Python Tcl
Server-side scripts	yes	PL/SQL	user defined functions
Triggers	yes	yes	yes
Partitioning methods	horizontal partitioning, sharding with MySQL Cluster or MySQL Fabric	Sharding, horizontal partitioning	partitioning by range, list and (since PostgreSQL 11) by hash
Replication methods	Multi-source replication Source-replica replication	Multi-source replication Source-replica replication	Source-replica replication
MapReduce	no	no	no
Consistency concepts	Immediate Consistency	Immediate Consistency	Immediate Consistency
Foreign keys	yes	yes	yes
Transaction concepts	ACID	ACID	ACID
Concurrency	yes	yes	yes
Durability	yes	yes	yes
In-memory capabilities	yes	yes	no
User concepts	Users with fine-grained authorization concept	fine grained access rights according to SQL-standard	fine grained access rights according to SQL-standard